

32.043.02  
119

**К**ОМП'ЮТИНГ



Пасічник В. В., Пасічник О. В., Угрин Д. І.

# ВЕБ-ТЕХНОЛОГІЇ ТА ВЕБ-ДИЗАЙН



Книга I

ВЕБ-ТЕХНОЛОГІЇ

Підручник



**Computing**

**Пасічник В. В.**

**Пасічник О. В.**

**Угрин Д. І.**

# **ВЕБ-ТЕХНОЛОГІЇ**

**Підручник**

**І частина**

НБ ПНУС



783923

**Львів  
2013**



УДК 004.7  
ББК 32.973-018.02  
П 19

Відтворення цієї книги або будь-якої її частини  
заборонено без письмової згоди видавництва.  
Будь-які спроби порушення авторських прав  
переслідуватимуться у судовому порядку.

Гриф надано Міністерством освіти і науки, молоді та спорту України  
(протокол № 1/11-8051 від 28.05.2012 р.).

Веб-технології [підручник] – Львів, «Магнолія 2006», 2013. – 336 с.

ISBN 978-617-574-093-4

«Магнолія 2006»

Пропонований підручник присвячено теоретичним та практичним аспектам веб-технології та веб-дизайну, наведено типові приклади веб-програмування. Розглянуто різні принципи побудови та функціонування веб-сайтів, використання сучасних веб-технологій та мов веб-програмування, будування веб-сторінок із заданими характеристиками і алгоритмами функціонування. У підручнику також розглядається широкий спектр протоколів, стандартів і технологій, що мають безпосереднє відношення до розробки веб-застосовувань.

Зміст підручника відповідає галузевому стандарту вищої освіти України з напрямку підготовки 6.050101 «Комп'ютерні науки» для дисципліни «Веб-технології та веб-дизайн».

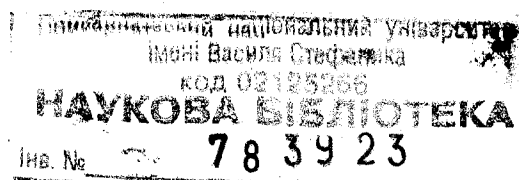
Підручник адресовано студентам вищих навчальних закладів спеціальності «Комп'ютерні науки» та викладачам.

УДК 004.7  
ББК 32.973-018.02  
П 19

ISBN 978-617-574-093-4

© Угрин Д. І., 2013

© «Магнолія 2006», 2013



## ЗМІСТ

Передмова.....	7
РОЗДІЛ 1	
ОСНОВИ ВЕБ.....	8
ТЕМА 1.1	
СТРУКТУРА І ПРИНЦИПИ ВЕБ.....	8
1.1.1. Поняття Інтернету.....	8
1.1.2. Роль стандартизації в Інтернеті.....	9
1.1.3. Стек протоколів TCP/IP.....	12
1.1.4. Система доменних імен DNS.....	13
1.1.5. Структура і принципи WWW.....	14
1.1.6. Проксі-сервери.....	15
1.1.7. Протоколи Інтернету прикладного рівня.....	15
ТЕМА 1.2	
УВЕДЕННЯ В КЛІЄНТ-СЕРВЕРНІ ТЕХНОЛОГІЇ ВЕБ.	
ПРОТОКОЛ HTTP.....	18
1.2.1. Протокол HTTP.....	18
1.2.2. Забезпечення безпеки передачі даних HTTP.....	27
1.2.3. Cookie.....	28
ТЕМА 1.3	
КЛІЄНТСЬКІ СЦЕНАРІЇ І ЗАСТОСУВАННЯ.....	30
1.3.1. Програми, що виконуються на клієнт-машині.....	30
1.3.2. Програми, що виконуються на сервері.....	30
1.3.3. Насичені інтернет-застосування.....	31
ТЕМА 1.4	
СЕРВЕРНІ ВЕБ-ЗАСТОСУВАННЯ.....	33
1.4.1. Стандарт CGI.....	33
1.4.2. Сценарії.....	37
1.4.3. Мова Python.....	37
1.4.3.1. Опис мови Python.....	39
1.4.3.2. Основні алгоритмічні конструкції.....	40
1.4.3.3. Функції в Python.....	42
1.4.3.4. Підтримка мережі та функціональне програмування на Python.....	44
1.4.3.5. Розробка Web-додатків.....	50
1.4.3.6. Мережні додатки на Python.....	58
1.4.4. Мова Ruby.....	70
1.4.4.1. Початкові дані.....	72
1.4.4.2. Об'єкти та методи Ruby.....	73
1.4.5. Технологія ASP.....	81
1.4.6. Інтерфейс ISAPI.....	83
Контрольні запитання.....	85
Тести для закріплення матеріалу.....	86

## РОЗДІЛ 2

<b>ВЕБ-ПРОГРАМУВАННЯ</b> .....	88
<b>ТЕМА 2.1</b>	
<b>JAVASCRIPT. ПРОГРАМНА ВЗАЄМОДІЯ З HTML ДОКУМЕНТАМИ НА ОСНОВІ DOM API</b> .....	88
2.1.1. Загальний огляд мови JavaScript .....	88
2.1.2. Об'єктна модель JavaScript .....	94
2.1.3. Забезпечення ефективності WEB – сайтів .....	102
2.1.4. Адаптація веб-сайту до клієнтського програмного забезпечення .....	108
2.1.5. Коротка характеристика VBScript .....	109
2.1.6. Java-апплети .....	109
2.1.7. ActionScript, XAML і Microsoft Silverlight – загальна характеристика .....	110
2.1.8. Поняття про DOM та HTML DOM .....	111
<b>ТЕМА 2.2</b>	
<b>МОВИ РОЗРОБКИ СЦЕНАРІЇВ PHP, PERL, JSP</b> .....	116
2.2.1. Мова розроблення сценаріїв PHP .....	116
2.2.2. Мова сценаріїв Perl .....	124
2.2.3. Мова розроблення сценаріїв JSP .....	134
<b>ТЕМА 2.3</b>	
<b>РОЗРОБКА CGI-ЗАСТОСУВАНЬ НА PERL, PHP</b> .....	147
2.3.1. Розробка CGI-застосовувань на Perl .....	147
2.3.2. Основи розробки сценаріїв на мові PHP .....	163
<b>ТЕМА 2.4</b>	
<b>ОСНОВИ РОЗРОБКИ ВЕБ-ЗАСТОСУВАНЬ З ДОПОМОГОЮ ASP.NET</b> .....	168
2.4.1. Основи ASP.NET .....	168
2.4.2. Серверні елементи управління ASP.NET .....	170
2.4.3. Основи розробки веб-застосовувань з допомогою J2EE .....	171
<b>ТЕМА 2.5</b>	
<b>ІНТЕРФЕЙСИ ВЗАЄМОДІЇ ВЕБ-ЗАСТОСУВАНЬ З СКБД</b> .....	185
2.5.1. Інтерфейси взаємодії веб-застосовувань з СКБД .....	185
2.5.2. ACTIVEX Data Objects та ADO.NET .....	185
<b>ТЕМА 2.6</b>	
<b>ВЕБ-СЕРВІСИ ТА МОВИ ЇХ ОПИСУВАННЯ</b> .....	187
2.6.1. Протокол XML-RPC .....	190
2.6.2. Протокол SOAP .....	193
2.6.3. Опис Web-служби .....	203
Контрольні запитання .....	222
Тести для закріплення матеріалу .....	224

## РОЗДІЛ 3

<b>ОСНОВИ XML</b> .....	227
<b>ТЕМА 3.1</b>	
<b>МОВИ ОПИСУВАННЯ СХЕМ XML</b> .....	227
3.1.1. Вступ в XML .....	227
3.1.2. Мови опису схем XML .....	231
3.1.3. DTD схема .....	232
3.1.4. XDR схема .....	235
<b>ТЕМА 3.2</b>	
<b>DOM XML. ПЕРЕТВОРЕННЯ XML ДОКУМЕНТІВ</b> .....	239
3.2.1. Передумови перетворення XML документів .....	239
3.2.2. XSLT та XPath .....	244
3.2.3. XSL-FO .....	245
3.2.4. XQuery .....	245
<b>ТЕМА 3.3</b>	
<b>ПРОГРАМНА ОБРОБКА XML ДОКУМЕНТІВ ЗА ДОПОМОГОЮ XML DOM</b> .....	250
3.3.1. Структурний аналіз (парсинг) XML .....	250
3.3.2. Програмний інтерфейс XML DOM .....	252
3.3.3. Переміщення між вузлами дерева .....	253
3.3.4. Ігнорування порожніх текстових вузлів .....	254
3.3.5. Зміна значення атрибуту .....	254
3.3.6. Властивості об'єкту Node .....	255
<b>ТЕМА 3.4</b>	
<b>ФОРМАТУВАННЯ І ПЕРЕТВОРЕННЯ XML-ДОКУМЕНТА З ДОПОМОГОЮ CSS І XSL. XSLT ПЕРЕТВОРЕННЯ XML-ДОКУМЕНТА</b> .....	257
3.4.1. Форматування і перетворення XML-документа з допомогою CSS і XSL .....	257
3.4.2. Оголошення XSL .....	258
3.4.3. Реалізація перетворення за допомогою JavaScript .....	258
Контрольні запитання .....	259
Тести для закріплення матеріалу .....	260
<b>РОЗДІЛ 4</b>	
<b>ВЕБ-ПОРТАЛИ</b> .....	262
<b>ТЕМА 4.1</b>	
<b>ІНТЕГРАЦІЯ ТА ВЗАЄМОДІЯ У ВЕБ-МЕРЕЖІ</b> .....	262
4.1.1. Веб-інтеграція .....	262
4.1.2. Інтеграція на основі XML .....	263
4.1.3. Веб-сервіси .....	265
4.1.4. Специфікація WSDL .....	267
4.1.5. Протокол SOAP .....	267
4.1.6. Стандарт DISCO .....	268
4.1.7. Специфікація UDDI .....	268



<b>ТЕМА 4.2</b>	
<b>РОЗРОБКА ВЕБ-СЛУЖБИ В ASP.NET .....</b>	<b>270</b>
4.2.1. Створення за допомогою ASP.NET веб-служби.....	270
4.2.2. Розробка веб-служби в ASP.NET. Створення проксі-збірки для веб-служби.....	277
<b>ТЕМА 4.3</b>	
<b>РОЗРОБКА ВЕБ-КОНТЕНТА. CMS/CMF.....</b>	<b>287</b>
4.3.1. Організація процесу розробки веб-контенту. CMS/CMF .....	287
4.3.2. Типи WCMS-систем. WCMS Drupal .....	287
<b>ТЕМА 4.4</b>	
<b>РОЗРОБКА RSS-ДЖЕРЕЛ І RSS-РІДЕРІВ.....</b>	<b>289</b>
4.4.1. Синдикація і агрегування веб-контенту.....	289
4.4.2. Формат RSS .....	290
4.4.3. Приклади розробки RSS-джерел і RSS-рідерів.....	291
4.4.4. Публікація RSS файлу .....	294
4.4.5. Додавання RSS-каналу за допомогою Microsoft Internet Explorer .....	294
4.4.6. Задача розробки RSS-джерел і RSS-рідерів .....	295
Контрольні запитання .....	298
Тести для закріплення матеріалу.....	299
<b>РОЗДІЛ 5</b>	
<b>ТЕХНОЛОГІЯ AJAX.....</b>	<b>302</b>
<b>ТЕМА 5.1</b>	
<b>ВСТУП У ТЕХНОЛОГІЮ AJAX. РОЗРОБКА МОБІЛЬНИХ ВЕБ-ЗАСТОСУВАНЬ .....</b>	<b>302</b>
5.1.1. Вступ у технологію AJAX.....	302
5.1.2. Розробка мобільних веб-застосунків.....	305
<b>ТЕМА 5.2</b>	
<b>РЕАЛІЗАЦІЯ АСИНХРОННОЇ ВЗАЄМОДІЇ ВЕБ-БРАУЗЕРА З ВЕБ-СЕРВЕРОМ ЗА ДОПОМОГОЮ ТЕХНОЛОГІЇ AJAX.....</b>	<b>308</b>
5.2.1. Виконання асинхронних запитів з JavaScript і Ajax .....	308
5.2.2. Об'єкт XMLHttpRequest та деталізація технології Ajax .....	309
Контрольні запитання .....	326
Тести для закріплення матеріалу.....	327
<b>ВИКОРИСТАНА ЛІТЕРАТУРА .....</b>	<b>330</b>
<b>ГЛОСАРІЙ .....</b>	<b>334</b>

## Передмова

Ефективне використання інформації, формування оптимальних рішень у різноманітних сферах в значній мірі залежать від рівня впровадження та використання інформаційних систем і технологій. На сучасному етапі розробка та впровадження досконалих інформаційних систем неможлива без їх адаптації до використання в глобальній мережі Internet, а також однієї з її основних частин мережі WWW. Широке розповсюдження веб-сайтів призвело до появи нового напрямку в галузі програмування – веб-технології та веб-дизайну.

В сучасних умовах невеликі компанії нерідко створюють та підтримують веб-сайти силами своїх працівників. Крім того, досвід впровадження сучасних складних веб-орієнтованих систем свідчить про необхідність розробки подібних систем шляхом активної взаємодії між компанією-розробником програмного забезпечення та представниками замовника таких систем. Для цього необхідно знати принципи проектування та розробки веб-сайтів, мати чітке уявлення про їх структуру, принципи функціонування, місце і роль в організації роботи підприємства, мати уявлення про перспективи подальшого розвитку веб-систем, а також вміти їх використовувати на високому рівні.

Таким чином, в сучасних умовах фахівці різного профілю повинні опановувати веб-технології та веб-дизайн. Відзначимо, що на сьогодні по причині своєї новизни напрям веб-технології та веб-дизайн не має чітких меж. Досить часто до цього напрямку відносять розробку серверного та клієнтського програмного забезпечення, адміністрування веб-сайту, реалізацію заходів по підвищенню ефективності його функціонування та формування самого дизайну веб-сайту. Тому основна увага у підручнику спрямована на принципи та прийоми засвоєння основ веб, веб-програмування, XML, веб-порталів, технології AJAX та веб-дизайну.

Підручник написано відповідно до освітньо-професійної програми підготовки бакалавра напряму 6.050101 – «Комп'ютерні науки» галузевого стандарту вищої освіти затвердженого Міністерством освіти і науки, молоді та спорту України.

Бажаємо нашим читачам натхнення та успіху в освоєнні дисципліни «Веб-технології та веб-дизайн» та сподіваємося, що даний підручник буде надійною допомогою при вивченні веб-технологій та веб-дизайну.

## РОЗДІЛ 1



## ОСНОВИ ВЕБ

## ТЕМА 1.1. СТРУКТУРА І ПРИНЦИПИ ВЕБ

## 1.1.1. Поняття Інтернету

## 1.1.2. Роль стандартизації в Інтернеті

## 1.1.3. Стек протоколів TCP/IP

## 1.1.4. Система доменних імен DNS

## 1.1.5. Структура і принципи WWW

## 1.1.6. Проксі-сервери

## 1.1.7. Протоколи Інтернету прикладного рівня

## 1.1.1. Поняття Інтернету

Оскільки фізичною основою мережі Веб є Інтернет, то для глибшого розуміння багатьох питань даного курсу потрібно буде коротко ознайомитися із структурою та протоколами Інтернету.

\* *Інтернет – це найбільша в світі мережа, що не має єдиного центру управління та, що працює по єдиних правилах і надає своїм користувачам єдиний набір послуг.*

Інтернет можна розглядати як «мережу мереж», кожна з яких управляється незалежним оператором – постачальником послуг Інтернету (ISP, Internet Service Provider).

З точки зору користувачів Інтернет є набором інформаційних ресурсів, розосереджених по різних мережах, включаючи ISP-мережі, корпоративні мережі і окремі комп'ютери домашніх користувачів. Кожен окремий комп'ютер в даній мережі називається хостом (від англійського терміну host).

Мережа ARPANET була створена в 1969 р. і складалася спочатку всього з чотирьох вузлів з комутацією пакетів, використовуваних для взаємодії декількох хостів і терміналів. Перші лінії зв'язку, що сполучали вузли, працювали на швидкості всього 50 Кбіт/с. Мережа ARPANET фінансувалася управлінням перспективного планування науково-дослідних робіт ARPA (Advanced Research Projects Agency) міністерства оборони США і призначалася для вивчення технології і протоколів комутації пакетів, які могли б використовуватися для кооперативних розподілених обчислень.

Чи можливе централізоване управління в такій глобальній мережі? Відповідь на дане питання буде негативною, оскільки, по-перше, дана мережа є транснаціональною і, по-друге, через історичні передумови її формування.

Проте, в Інтернеті можуть виявлятися опосередковані форми централізації у формі єдиної технічної політики, узгодженого набору технічних стандартів, призначенні імен і адрес комп'ютерів та мереж, що входять в Інтернет. Тобто, Інтернет є децентралізованою мережею, що має свої переваги і недоліки.

Перевагою Інтернету є легкість нарощування Інтернету шляхом укладання договору між двома ISP.

## 1.1.2. Роль стандартизації в Інтернеті

Інтернет є дуже складною мережею, тому відповідно складними є завдання організації взаємодії між пристроями мережі. Для вирішення такого роду завдань використовується декомпозиція, тобто розбиття складного завдання на декілька простіших завдань-модулів. Одна з концепцій, що реалізовує декомпозицію, називається багаторівневим підходом. Такий підхід дає можливість проводити розробку, тестування і модифікацію кожного окремого рівня незалежно від інших рівнів. Ієрархічна декомпозиція дозволяє, переміщаючись в напрямі від нижчих до вищих рівнів, переходити до простішого представлення вирішуваного завдання.

Специфіка багаторівневого представлення мережевої взаємодії полягає в тому, що в процесі обміну повідомленнями беруть участь як мінімум дві сторони, для яких необхідно забезпечити узгоджену роботу двох ієрархій апаратно-програмних засобів. Кожен з рівнів повинен підтримувати інтерфейс з вище- і нижчеіснуючими рівнями власної ієрархії засобів та інтерфейсом із засобами взаємодії іншої сторони на тому ж рівні ієрархії. Даний тип інтерфейсу називається протоколом (див. рисунок 1.1.2.1).

Ієрархічно організований набір протоколів, достатній для організації взаємодії вузлів в мережі, називається стеком протоколів.

На початку 80-х років міжнародні організації по стандартизації ISO (International Organization for Standardization), ITU (International Telecommunications Union) та інші розробили стандартну модель взаємодії відкритих систем OSI (Open System Interconnection). Призначення даної моделі полягає в узагальненому представленні засобів мережевої взаємодії. Її також можна розглядати як універсальну мову мережевих фахівців (довідкова модель).

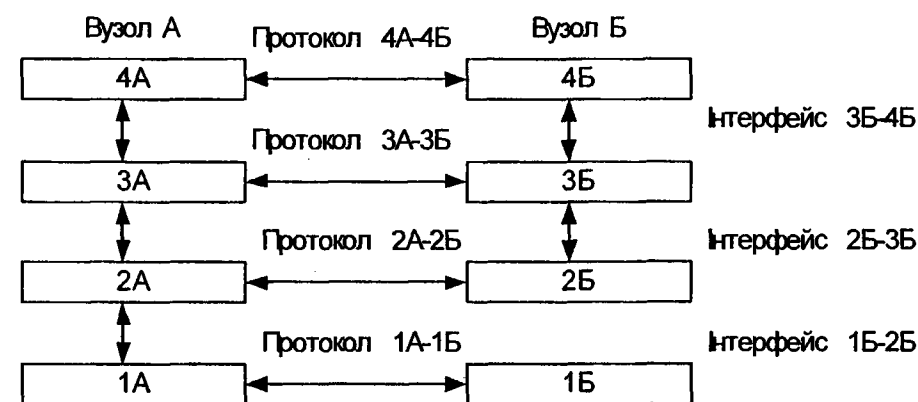


Рис. 1.1.2.1 Організація взаємодії між рівнями ієрархії при ієрархічній декомпозиції в мережі Інтернет

Оскільки мережа – це з'єднання різноманітного устаткування, то актуальною є проблема сумісності, що у свою чергу вимагає узгодження з усіма виробниками загальноприйнятих стандартів. Відкритою є система, яка побудована відповідно до відкритих специфікацій.

Специфікація є формалізованим описом апаратних (програмних) компонентів, способів їх функціонування, взаємодії з іншими компонентами, умов експлуатації, особливих характеристик. Під відкритими специфікаціями розуміють опубліковані, загальнодоступні специфікації, що відповідають стандартам і прийняті в результаті досягнення згоди після різностороннього обговорення всіма зацікавленими сторонами. Використання відкритих специфікацій при розробці систем дозволяє третім сторонам розробляти для цих систем апаратно-програмні засоби розширення і модифікації, а також створювати програмно-апаратні комплекси з продуктів різних виробників.

Якщо дві мережі побудовано з дотриманням принципів відкритості, то це дає наступні переваги:

- можливість побудови мережі із апаратних і програмних засобів різних виробників, що притримуються стандарту;
- безболісна заміна окремих компонентів мережі іншими, досконалішими;
- легкість сполучення однієї мережі з іншою.

В рамках моделі OSI засоби взаємодії поділяються на сім рівнів: прикладний, представлення, сеансовий, транспортний, мережевий, каналний і фізичний. У розпорядження програмістів надається прикладний програмний інтерфейс, що дозволяє звертатися із запитами до самого верхнього рівня, а саме, до рівня застосувань.

Мережа Інтернет будувалася у повній відповідності з принципами відкритих систем. У розробці стандартів цієї мережі брали участь тисячі фахівців-користувачів мережі з вузів, наукових організацій і компаній. Результат роботи по стандартизації втілюється в документах RFC.

RFC (англ. Request for Comments) – документ з серії пронумерованих інформаційних документів Інтернету, що містять технічні специфікації і стандарти, що широко використовуються у Всесвітній мережі. У даний час первинною публікацією документів RFC займається IETF під егідою відкритої організації Суспільство Інтернету (ISOC). Правами на RFC володіє саме Суспільство Інтернету. Формат RFC з'явився в 1969 р. при обговоренні проекту ARPANET. Перші RFC поширювалися в друкарському вигляді на папері у вигляді звичайних листів, але вже з грудня 1969 р., коли запрацювали перші сегменти ARPANET, документи почали поширюватися в електронному вигляді. У таблиці 1.1.2.1 приведені деякі з найбільш відомих документів RFC.

Таблиця 1.1.2.1

#### Приклади популярних RFC-документів

Номер RFC	Тема
1	2
RFC 768	UDP
RFC 791	IP
RFC 793	TCP
RFC 822	Формат електронної пошти, замінений RFC 2822

Закінчення таблиці 1.1.2.1

1	2
RFC 959	FTP
RFC 1034	DNS – концепція
RFC 1035	DNS – впровадження
RFC 1591	Структура доменних імен
RFC 1738	URL
RFC 1939	Протокол POP версії 3 (POP3)
RFC 2026	Процес стандартизації в Інтернеті
RFC 2045	MIME
RFC 2231	Кодування символів
RFC 2616	HTTP
RFC 2822	Формат електронної пошти
RFC 3501	IMAP версії 4 видання 1 (IMAP4rev1)

Основним організаційним підрозділом, що координує роботу по стандартизації Інтернет є ISOC (Internet Society), що об'єднує порядка 100 тисяч учасників, які займаються різними аспектами розвитку даної мережі. ISOC займається роботою IAB (Internet Architecture Board), що включає дві групи:

- ⇒ IRTF (Internet Research Task Force) – координує довгострокові дослідницькі проекти, що відносяться до TCP/IP;
- ⇒ IETF (Internet Engineering Task Force) – інженерна група, що визначає специфікації для подальших стандартів Інтернет.

Розробкою стандартів для мережі Веб-серверу, починаючи з 1994 року, займається Консорціум W3C (World Wide Web Consortium).

Консорціум W3C – організація, що розробляє і впроваджує технологічні стандарти для Інтернету і WWW. Місія W3C формулюється таким чином: «Повністю розкрити потенціал всесвітньої павутини шляхом створення протоколів і принципів, що гарантують довгостроковий розвиток мережі». Два інші найважливіші завдання консорціуму – забезпечити повну «інтернаціоналізацію мережі» та зробити її доступною для людей з обмеженими можливостями.

W3C розробляє для WWW єдині принципи і стандарти, що мають назву «Рекомендації», які потім впроваджуються розробниками програм і устаткування. Завдяки Рекомендаціям досягається сумісність між програмними продуктами і устаткуванням різних компаній, що робить мережу WWW досконалішою, універсальною і зручною у використанні.

Всі Рекомендації W3C відкриті, тобто, не захищені патентами і можуть впроваджуватися будь-якою людиною без яких-небудь фінансових відрахувань Консорціуму.

Для зручності користувачів Консорціумом створені спеціальні програми-валідатори (англ. Online Validation Service), які доступні по мережі і можуть за декілька секунд перевірити документи на відповідність популярним Рекомендаціям W3C. Консорціумом також створено багато інших утиліт для полегшення роботи веб-майстрів і програмістів. Більшість утиліт – це програми з відкритим початковим кодом, всі вони безкоштовні. Останнім часом, слідкуючи за світовими тенденціям, Консорціум, в цілому, значно більше уваги приділяє проектам з відкритим початковим кодом.



Перш ніж перейти до опису структури, принципів роботи і основних протоколів мережі Веб, розглянемо основний стек протоколів мережі Інтернет – стек TCP/IP.

### 1.1.3. Стек протоколів TCP/IP

Протоколи TCP/IP спочатку були орієнтовані на глобальні мережі, в яких якість каналів зв'язку не ідеальна. Стек протоколів TCP/IP дозволяє створювати глобальні мережі, комп'ютери в яких з'єднані один з одним різними способами від високошвидкісних оптоволоконних кабелів і супутникових каналів до комутованих телефонних ліній. TCP/IP відповідає моделі OSI досить умовно і містить чотирі рівні. Прикладний рівень стеку відповідає трьом верхнім рівням моделі OSI: прикладному, представлення і сеансовому.

У мережі дані завжди передаються блоками відносно невеликого розміру. Кожен блок має префіксну частину (заголовок), що описує вміст блоку, і суфіксну, що містить, наприклад, інформацію для контролю цілісності блоку даних, що передається.

Назва стеку протоколів TCP/IP складається з назв двох різних протоколів. Протоколом IP (Internet Protocol) є протокол нижнього (мережевого) рівня і відповідає за передачу пакетів даних в мережі. Він відноситься до так званих протоколів датаграм і працює без підтверджень. Останнє означає, що при його використанні доставка пакетів даних не гарантується і не підтверджується. Не гарантується також і те, що пакети досягнуть пункту призначення в тій послідовності, в якій вони були відправлені.

До протоколів мережевого рівня відноситься також протокол міжмережевих повідомлень управління – ICMP (англ. Internet Control Message Protocol), який призначений для передачі маршрутизатором джерела інформації про помилки при передачі пакету.

Очевидно, що набагато зручніше передавати дані по каналу, який працює коректно, доставляючи всі пакети по порядку. Тому над протоколом IP працює протокол передачі даних більш високого рівня – TCP (англ. Transmission Control Protocol). Посилаючи і приймаючи пакети через протокол IP, протокол TCP гарантує доставку всіх переданих пакетів даних в правильній послідовності.

Слід зазначити, що при використанні протоколу IP забезпечується швидша передача даних, оскільки не витрачається час на підтвердження прийому кожного пакету. Є й інші переваги. Одна з них полягає в тому, що він дозволяє розсилати пакети даних в широкому режимі, при якому вони досягають всіх комп'ютерів фізичної мережі. Що ж до протоколу TCP, то для передачі даних з його допомогою необхідно створити канал зв'язку між комп'ютерами. Він створюється з використанням протоколу IP.

Для ідентифікації мережеских інтерфейсів використовуються 3 типи адрес:

- апаратні адреси (або MAC-адреси);
- мережескі адреси (IP-адреси);
- символічні (доменні) імена.

У рамках IP протоколу для створення глобальної системи адресації, не залежної від способів адресації вузлів в окремих мережах, використовується пара ідентифікаторів, що складається з номеру мережі і номеру вузла. При цьому IP-адреса іденти-

фікує не окремий комп'ютер або маршрутизатор, а одне мережеске з'єднання у складі мережі, в яку він входить; тобто кінцевий вузол може входити в декілька IP-мереж.

### 1.1.4. Система доменних імен DNS

Не дивлячись на те, що апаратне і програмне забезпечення в рамках TCP/IP мереж для ідентифікації вузлів використовує IP-адресу, користувачі віддають перевагу символічним іменам (доменним іменам).

Спочатку в локальних мережах з невеликого числа комп'ютерів застосовувалися плоскі імена, що складаються з послідовності символів без розділення їх на окремі частини, наприклад, MYCOMP. Для встановлення відповідності між символічними іменами і числовими адресами використовувалися ширококомвні запити. Проте для великих територіально розподілених мереж, що працюють на основі протоколу TCP/IP, такий спосіб виявився неефективним. Тому для встановлення відповідності між доменним ім'ям і IP-адресою використовується спеціальна система доменних імен (DNS, Domain Name System), яка заснована на створюваних адміністраторами мережі таблицях відповідності.

У мережах TCP/IP використовується доменна система імен, що має ієрархічну (у вигляді дерева) структуру. Дана структура імен нагадує ієрархію імен, використовуваних в багатьох файлових системах. Запис доменного імені починається з наймолодшої складової, потім після крапки слідує наступна за пріоритетом символічна частина імені і так далі. Послідовність закінчується кореневим ім'ям, наприклад: company.yandex.ua.

Побудована таким чином система імен дозволяє розділяти адміністративну відповідальність по підтримці унікальності імен в межах свого рівня ієрархії між різними людьми або організаціями.

Сукупність імен, біля яких декілька старших складових частин збігаються, утворюють домен імен.

Кореневий домен управляється центральними органами Інтернету: IANA і Internic.

Домени верхнього рівня призначаються для кожної країни, а також для різних типів організацій. Імена цих доменів повинні слідувати міжнародному стандарту ISO 3166. Для позначення країн використовуються двохбуквені аббревіатури, наприклад, ua (Україна), ru (Росія), us (США), it (Італія), fr (Франція).

Для різних типів організацій використовуються трьохбуквені аббревіатури:

- ⇒ net – мережескі організації;
- ⇒ org – некомерційні організації;
- ⇒ com – комерційні організації;
- ⇒ edu – освітні організації;
- ⇒ gov – урядові організації.

Адміністрування кожного домену покладається на окрему організацію, яка делегує адміністрування піддоменів іншим організаціям.

Для отримання доменного імені необхідно реєструватися у відповідній організації, якій організація INTERNIC делегувала свої повноваження по розподілу доменних імен.

У TCP/IP мережах відповідність між доменними іменами і IP-адресами може встановлюватися як локальними засобами, так і централізованими службами. Первинна відповідність задавалася за допомогою створюваного вручну на хості файлу `hosts.txt`, що складається з рядків, які містять пару виду «доменне ім'я – IP-адреса». Однак з активним зростанням Інтернету таке рішення виявилось таким, що не масштабується.

Альтернативне рішення – централізована служба DNS, що використовує розподілену базу відображень «доменне ім'я – IP-адреса». Сервер домену зберігає лише імена, які закінчуються на наступному нижче по дереву рівні. Це дозволяє розподіляти більш рівномірно навантаження по дозволу імен між всіма DNS-серверами. Кожен DNS-сервер, окрім таблиці відображення імен, містить посилання на DNS-сервери своїх піддоменів.

Існують дві схеми дозволу DNS-імен:

➤ **Нерекурсивна процедура:**

- ⇒ DNS-клієнт звертається до кореневого DNS-серверу з вказівкою повного доменного імені;
- ⇒ DNS-сервер відповідає клієнтові, вказуючи адресу наступного DNS-серверу, що обслуговує домен верхнього рівня, заданий в наступній старшій частині імені;
- ⇒ DNS-клієнт робить запит наступного DNS-серверу, який відсилає його до DNS-серверу потрібного піддомену і так далі, поки не буде знайдений DNS-сервер, в якому зберігається відповідність імені IP-адреси за яким здійснювався запит. Сервер дає остаточну відповідь клієнтові.

➤ **Рекурсивна процедура:**

- ⇒ DNS-клієнт запрошує локальний DNS-сервер, що обслуговується піддоменом, якому належить клієнт;
- ⇒ якщо локальний DNS-сервер знає відповідь, він повертає його клієнтові;
- ⇒ якщо локальний сервер не знає відповіді, то він виконує ітеративні запити до кореневого серверу. Після отримання відповіді сервер передає його клієнтові.

Таким чином, при рекурсивній процедурі клієнт фактично передоручає роботу своєму серверу. Для прискорення пошуку IP-адрес DNS-сервери широко застосовують кешування (на якийсь час: від години до декількох днів) відповідей, що проходять через них.

### 1.1.5. Структура і принципи WWW

Мережу WWW утворюють мільйони веб-серверів, розташованих по всьому світу. Веб-сервер є програмою, що запускається на підключеному до мережі комп'ютері і передає дані по протоколу HTTP.

Для ідентифікації ресурсів (часто файлів або їх частин) у WWW використовуються ідентифікатори ресурсів URI (Uniform Resource Identifier). Для визначення місцезнаходження ресурсів в цій мережі використовуються локатори ресурсів URL (Uniform Resource Locator). Такі URL-локатори є комбінацією URI і системи DNS.

Доменне ім'я (або IP-адреса) входить до складу URL для позначення комп'ютера (його мережевого інтерфейсу), на якому працює програма веб-сервер.

На клієнтському комп'ютері для переглядання інформації, отриманої від веб-серверу, застосовується спеціальна програма – веб-браузер. Основна функція веб-браузеру – відображення гіпертекстових сторінок (веб-сторінок). Для створення гіпертекстових сторінок у WWW спочатку використовувалася мова HTML. Набір веб-сторінок утворює веб-сайт.

### 1.1.6. Проксі-сервери

Проксі-сервер (англ. proxy-server) – служба в комп'ютерних мережах, що дозволяє клієнтам виконувати непрямі запити до інших мережевих служб.

Спочатку клієнт підключається до проксі-серверу і запрошує якийсь ресурс, розташований на іншому сервері. Потім проксі-сервер або підключається до вказаного серверу і отримує ресурс у нього, або повертає ресурс з власного кешу (якщо є). В деяких випадках запит клієнта або відповідь сервера може бути змінений проксі-сервером з певною метою. Також проксі-сервер дозволяє захищати клієнтський комп'ютер від деяких мережевих атак.

Найчастіше проксі-сервери застосовуються для наступних цілей:

- забезпечення доступу з комп'ютерів локальної мережі в Інтернет;
- кешування даних: якщо часто відбуваються звернення до одних і тих же зовнішніх ресурсів, то можна тримати їх копію на проксі-сервері і видавати за запитом, знижуючи тим самим навантаження на канал в зовнішню мережу і прискорюючи отримання клієнтом інформації за запитом;
- стиснення даних: проксі-сервер завантажує інформацію з Інтернету і передає інформацію кінцевому користувачу в стислому вигляді;
- захист локальної мережі від зовнішнього доступу: наприклад, можна настроїти проксі-сервер так, що локальні комп'ютери звертатимуться до зовнішніх ресурсів тільки через нього, а зовнішні комп'ютери не зможуть звертатися до локальних взагалі (вони «бачать» тільки проксі-сервер);
- обмеження доступу з локальної мережі до зовнішньої: наприклад, можна заборонити доступ до певних веб-сайтів, обмежити використання Інтернету якимсь локальним користувачам, встановлювати квоти на трафік або смугу пропускання, фільтрувати рекламу і віруси;
- анонімізація доступу до різних ресурсів. Проксі-сервер може приховувати інформацію про джерело запиту або користувача. У такому разі цільовий сервер бачить лише інформацію про проксі-сервер, наприклад, IP-адрес, але не має можливості визначити дійсне джерело запиту. Існують також спотворюючі проксі-сервери, які передають цільовому серверу помилкову інформацію про дійсного користувача.

### 1.1.7. Протоколи Інтернету прикладного рівня

Найвищий рівень в ієрархії протоколів Інтернету займають наступні протоколи прикладного рівня:

- ⇒ *DNS – розподілена система доменних імен, яка за запитом, що містить доменне ім'я хоста повідомляє IP адресу;*
- ⇒ *HTTP – протокол передачі гіпертексту в Інтернеті;*
- ⇒ *HTTPS – розширення протоколу HTTP, що підтримує шифрування;*
- ⇒ *FTP (File Transfer Protocol – RFC 959) – протокол, призначений для передачі файлів в комп'ютерних мережах;*
- ⇒ *Telnet (TELEcommunication NETwork – RFC 854) – мережевий протокол для реалізації текстового інтерфейсу по мережі;*
- ⇒ *SSH (Secure Shell - RFC 4251) – прикладний протокол, що дозволяє проводити віддалене управління операційною системою і передачу файлів. На відміну від Telnet шифрує весь трафік;*
- ⇒ *POP3 – протокол поштового клієнта, який використовується поштовим клієнтом для отримання повідомлень електронної пошти з серверу;*
- ⇒ *IMAP – протокол доступу до електронної пошти в Інтернеті;*
- ⇒ *SMTP – протокол, який використовується для відправки пошти від користувачів до серверів і між серверами для подальшої пересилки до одержувача;*
- ⇒ *LDAP – протокол для доступу до служби каталогів X.500, є широко використовуваним стандартом доступу до служб каталогів;*
- ⇒ *XMPP (Jabber) – заснований на XML, розширюваний протокол для миттєвого обміну повідомленнями в майже реальному часі;*
- ⇒ *SNMP – базовий протокол управління мережі Internet.*

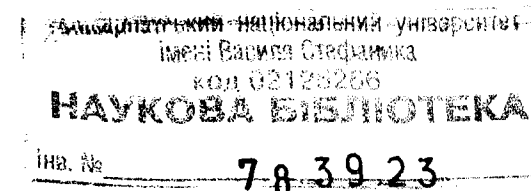
Розглянемо детальніше деякі з цих протоколів.

**FTP** дозволяє підключатися до серверів FTP, проглядати вміст каталогів і завантажувати файли з сервера або на сервер; крім того, можливий режим передачі файлів між серверами; FTP дозволяє обмінюватися файлами і виконувати операції над ними через TCP-мережі. Даний протокол працює незалежно від операційних систем. Історично протокол FTP запропонував відкриту функціональність, забезпечуючи прозоре перенесення файлів з одного комп'ютера на інший по мережі. Це не так тривіально, як може здатися, оскільки в різнотипних комп'ютерів можуть розрізнятися розміри слів, біти в словах можуть зберігатися в неоднаковому порядку або використовуватися різні формати слів.

**Telnet.** Назву «telnet» мають також деякі утиліти, що реалізують клієнтську частину протоколу. Протокол telnet працює відповідно до принципів архітектури «клієнт-сервер» і забезпечує емуляцію алфавітно-цифрового термінала, обмежуючи користувача режимом командного рядка. Застосування telnet надало мову для спілкування терміналів з віддаленими комп'ютерами. Коли з'явилася мережа ARPANET, для кожної комп'ютерної системи були потрібні власні термінали. Застосування telnet стало спільним знаменником для терміналів. Досить було написати для кожного комп'ютера програмне забезпечення, що підтримує «термінал telnet», щоб один термінал міг взаємодіяти з комп'ютерами всіх типів.

**SSH.** Схожий по функціональності з протоколами telnet і rlogin, але, на відміну від них, шифрує весь трафік, включаючи і передавані паролі. SSH-клієнти і SSH-сервери є для більшості операційних систем.

**Поштові протоколи.** Хоча telnet і FTP були (і залишаються) корисними, та першим застосуванням, що зробило переворот у свідомості користувачів комп'ютерів мережі ARPANET стала електронна пошта. До мережі ARPANET існували системи електронної пошти, але всі вони були однокомп'ютерними системами. У 1972 р. Рей Томлінсон (Ray Tomlinson) з компанії BBN написав перший пакет, що надає розподілені поштові послуги в комп'ютерній мережі з декількох комп'ютерів. Вже до 1973 р. дослідження управління ARPA показали, що три чверті всього трафіку мережі ARPANET складала електронна пошта. Користь електронної пошти опинилася настільки велика, що все більше користувачів прагнули підключитися до мережі ARPANET, внаслідок чого зростала потреба в додаванні нових вузлів і використанні високошвидкісних ліній. Таким чином, з'явилася тенденція, що зберігається і до цього дня.





## ТЕМА 1.2. УВЕДЕННЯ В КЛІЄНТ-СЕРВЕРНІ ТЕХНОЛОГІЇ ВЕБ. ПРОТОКОЛ HTTP

### 1.2.1. Протокол HTTP

#### 1.2.2. Забезпечення безпеки передачі даних HTTP

#### 1.2.3. Cookie

Базовим протоколом мережі гіпертекстових ресурсів Веб є протокол HTTP. У його основу покладена взаємодія «клієнт-сервер», тобто передбачається, що:

- споживач-клієнт ініціюючи з'єднання з постачальником-сервером посилає йому запит;
- постачальник-сервер, отримавши запит, проводить необхідні дії і повертає назад клієнтові відповідь з результатом.

При цьому можливі два способи організації роботи комп'ютера-клієнта:

- тонкий клієнт – це комп'ютер-клієнт, який переносить всі завдання по обробці інформації на сервер. Прикладом тонкого клієнта може служити комп'ютер з браузером, що використовується для роботи з веб-застосуваннями;
- товстий клієнт, навпаки, проводить обробку інформації незалежно від сервера, використовує останній, в основному, лише для зберігання даних.

Перш ніж перейти до конкретних клієнт-серверних веб-технологій, розглянемо основні принципи і структуру базового протоколу HTTP.

### 1.2.1. Протокол HTTP

HTTP (Hyper Text Transfer Protocol – RFC 1945, RFC 2616) – протокол прикладно-го рівня для передачі гіпертексту.

Центральним об'єктом в HTTP є ресурс, на який вказує URI в запиті клієнту. Зазвичай такими ресурсами є файли, що зберігаються на сервері. Особливістю протоколу HTTP є можливість вказати в запиті і відповіді спосіб представлення одного і того ж ресурсу по різних параметрах: формату, кодуванню, мові і так далі. Саме завдяки можливості вказівки способу кодування повідомлення клієнт і сервер можуть обмінюватися двійковими даними, хоча спочатку даний протокол призначений для передачі символічної інформації. На перший погляд це може здатися зайвою витратою ресурсів. Дійсно, дані в символічному вигляді займають більше пам'яті, повідомлення створюють додаткове навантаження на канали зв'язку, проте подібний формат має багато переваг. Повідомлення, що передаються по мережі, легкі для читання, і, проаналізувавши отримані дані, системний адміністратор може легко знайти помилку та усунути її. При необхідності роль одного із взаємодіючих застосувань може виконувати людина, вводячи уручну повідомлення в необхідному форматі.

На відміну від багатьох інших протоколів, HTTP є протоколом без пам'яті. Це означає, що протокол не зберігає інформацію про попередні запити клієнтів і відповіді серверів. Компоненти, що використовують HTTP, можуть самостійно здійснювати зберігання інформації про стан, пов'язаний з останніми запитами і відповідями. Наприклад, клієнтське веб-застосування, що посилає запити, може відстежувати за-

тримки відповідей, а веб-сервер може зберігати IP-адресу і заголовки запитів останніх клієнтів.

Все програмне забезпечення для роботи з протоколом HTTP розділяється на три основні категорії:

- ⇒ сервери – постачальники послуг зберігання і обробки інформації (обробка запитів);
- ⇒ клієнти – кінцеві споживачі послуг сервера (відправка запитів);
- ⇒ проксі-сервери для підтримки роботи транспортних служб.

Основними клієнтами є браузери, наприклад: Internet Explorer, Opera, Mozilla Firefox, Netscape Navigator та інші. Найбільш популярними реалізаціями веб-серверів є: Internet Information Services (IIS), Apache, lighttpd, nginx. Найбільш відомі реалізації проксі-серверів: Squid, UserGate, Multiproxy, Naviscope.

«Класична» схема HTTP-сеансу виглядає так:

- встановлення TCP-з'єднання;
- запит клієнта;
- відповідь сервера;
- розрив TCP-з'єднання.

Таким чином, клієнт посилає серверу запит, отримує від нього відповідь, після чого взаємодія припиняється. Зазвичай запитом клієнта є вимога передати HTML-документ або який-небудь інший ресурс, а відповідь сервера містить код цього ресурсу.

До складу HTTP-запиту, що передається клієнтом серверу, входять наступні компоненти:

- ⇒ рядок стану (іноді для її позначення використовують також терміни *рядок-статус*, або *рядок запиту*);
- ⇒ поля заголовку;
- ⇒ порожній рядок;
- ⇒ тіло запиту.

Рядок стану разом з полями заголовка інколи називають також заголовком запиту.

Рядок стану має наступний формат:

метод\_запиту      URL\_ресурсу      версія\_протоколу\_HTTP

Розглянемо компоненти рядка стану, при цьому особливу увагу приділимо методам запиту. Нижче приведено рисунок утворення структури запиту клієнта.

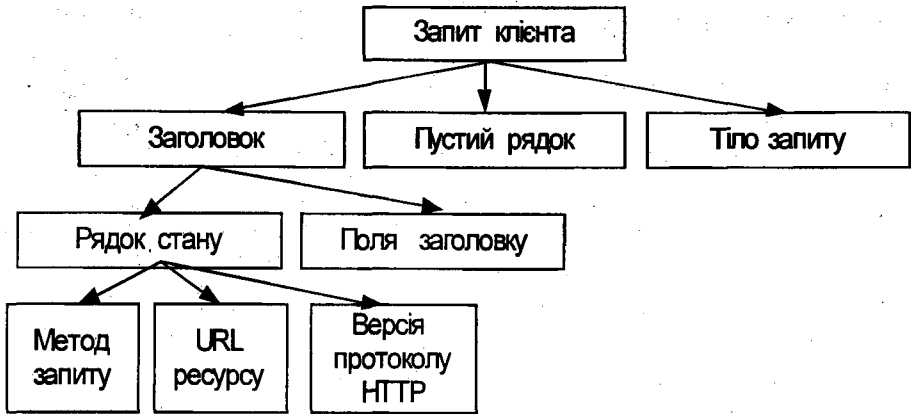


Рис. 1.2.1.1. Структура запиту клієнта.

Метод, вказаний в рядку стану, визначає спосіб дії на ресурс, URL якого заданий в тому ж рядку. Метод може приймати значення GET, POST, HEAD, PUT, DELETE і так далі. Не дивлячись на велику кількість методів, для веб-програміста по-справжньому важливі лише два з них: GET і POST.

➤ **GET.** Згідно формального визначення, метод GET призначається для отримання ресурсу з вказаним URL. Отримавши запит GET, сервер повинен прочитати вказаний ресурс і включити код ресурсу до складу відповіді клієнтові. Ресурс, URL якого передається у складі запиту, не обов'язково має бути HTML-сторінкою, файлом із зображенням або іншими даними. URL ресурсу може вказувати на виконуваний код програми, який, при дотриманні певних умов, має бути запущений на сервері. У цьому випадку клієнтові повертається не код програми, а дані, що згенерували в процесі її виконання. Не дивлячись на те що, за визначенням, метод GET призначений для отримання інформації, він може застосовуватися і в інших цілях. Метод GET цілком підходить для передачі невеликих фрагментів даних на сервер;

➤ **POST.** Згідно тому ж формальному визначенню, основне призначення методу POST – передача даних на сервер. Проте, подібно до методу GET, метод POST може застосовуватися по-різному і нерідко використовується для отримання інформації з серверу. Як і у випадку з методом GET, URL, заданий в рядку стану, вказує на конкретний ресурс. Метод POST також може використовуватися для запуску процесу;

➤ методи **HEAD** і **PUT** є модифікаціями методів GET і POST.

Версія протоколу HTTP, як правило, задається в наступному форматі:  
**HTTP/версія.модифікація**

Поля заголовку, що йдуть за рядком стану, дозволяють уточнювати запит, тобто передавати серверу додаткову інформацію. Поле заголовка має наступний формат:

**Імя\_поля: Значення**

Призначення поля визначається його ім'ям, яке відокремлюється від значення двокрапкою.

Імена деяких, які найчастіше зустрічаються в запиті клієнта полів заголовку і їх призначення приведені в таблиці 1.2.1.1.

Таблиця 1.2.1.1

Поля заголовка запиту HTTP

Поля заголовку HTTP-запиту	Значення
Host	Доменне ім'я чи IP-адреса вузла, куди звертається клієнт
Referer	URL документу, який посилається на ресурс, вказаний в рядку стану
From	Адреса електронної пошти користувача, що працює з клієнтом
Accept	MIME-типи даних, що обробляються клієнтом. Це поле може мати декілька значень, відокремлених одне від іншого комами. Часто поле заголовку Асепт використовується для того, щоб повідомити сервер про те, яких типів графічних файлів підтримує клієнт
Accept-Language	Набір двосимвольних ідентифікаторів, розділених комами, які позначають мови, підтримувані клієнтом
Accept-Charset	Перелік підтримуваних наборів символів
Поля заголовку HTTP-запиту	Значення
Content-Type	MIME-тип даних, що містяться в тілі запиту (якщо запит не складається з одного заголовку)
Content-Length	Число символів, що містяться в тілі запиту (якщо запит не складається з одного заголовку)
Range	Присутній в тому випадку, якщо клієнт запрошує не весь документ, а лише його частину
Connection	Використовується для управління TCP-з'єднанням. Якщо в полі міститься Close, це означає, що після обробки запиту сервер повинен закрити з'єднання. Значення Keep-Alive пропонує не закривати TCP-з'єднання, щоб воно могло бути використане для подальших запитів
User-Agent	Інформація про клієнта

У багатьох випадках при роботі у Веб тіло запиту відсутнє. При запуску CGI-сценаріїв дані, що передаються для них у запиті, можуть розміщуватися в тілі запиту.

Нижче представлений приклад HTML-запиту, згенерованого браузером.

**GET http://oak.oakland.edu/ HTTP/1.0**

**Connection: Keep-Alive**

**User-Agent: Mozilla/4.04 [en] (Win7; I)**

**Host: oak.oakland.edu**

**Accept: image/gif, image/x-bitmap, image/jpeg, image/pjpeg, image/png \*/\***

**Accept-Language: en**

**Accept-Charset: iso-8859-1,\*,utf-8**

Отримавши від клієнта запит, сервер повинен відповісти йому. Знання структури відповіді сервера необхідне розробнику веб-додатків, оскільки програми, які виконуються на сервері, повинні самостійно формувати відповідь клієнтові.

Подібно до запиту клієнта, відповідь серверу також складається з чотирьох перерахованих нижче компонентів:

- рядок стану;
- поля заголовка;
- порожній рядок;
- тіло відповіді.

Відповідь сервера клієнтові починається з рядка стану, який має наступний формат:

- Версія\_протоколу                      Код\_відповіді                      Пояснююче\_повідомлення
- ⇒ *версія\_протоколу задається в тому ж форматі, що і в запиті клієнта, і має той же зміст;*
- ⇒ *код\_відповіді – це тризначне десяткове число, що представляє в закодованому вигляді результат обслуговування запиту сервером;*
- ⇒ *пояснююче\_повідомлення дублює код відповіді в символному вигляді. Це рядок символів, який не обробляється клієнтом. Він призначений для системного адміністратора або оператора, що займається обслуговуванням системи, і є розшифровкою коду відповіді.*

Із трьох цифр, складових коду відповіді, перша (старша) визначає клас відповіді, останні дві є номером відповіді в середині класу. Так, наприклад, якщо запит був оброблений успішно, клієнт отримує наступне повідомлення:

HTTP/1.0                      200                      ОК

Як видно, за версією протоколу HTTP 1.0 слідує код 200. У цьому коді символ 2 означає успішну обробку запиту клієнта, а решта дві цифри (00) – номер даного повідомлення.

У використовуваних в даний час реалізаціях протоколу HTTP перша цифра не може бути більше 5 і визначає наступні класи відповідей:

- 1 – спеціальний клас повідомлень, що називаються інформаційними. Код відповіді, що починається з 1, означає, що сервер продовжує обробку запиту. При обміні даними між HTTP-клієнтом і HTTP-сервером повідомлення цього класу використовуються досить рідко;
- 2 – успішна обробка запиту клієнта;
- 3 – перенаправлення запиту. Щоб запит був виконаний, необхідно зробити додаткові дії;
- 4 – помилка клієнта. Як правило, код відповіді, що починається з цифри 4, повертається в тому випадку, якщо в запиті клієнта зустрілася синтаксична помилка;
- 5 – помилка сервера. По тих або інших причинах сервер не в змозі виконати запит.

Приклади кодів відповідей, які клієнт може отримати від сервера, і пояснюючі повідомлення приведені в таблиці 1.2.1.2.

Таблиця 1.2.1.2

Класи кодів відповіді сервера

Код	Розшифрування	Інтерпретація
100	Continue	Частина запиту прийнята, і сервер чекає від клієнта продовження запиту
200	OK	Запит успішно оброблений і у відповіді клієнта передаються дані, вказані в запиті
201	Created	В результаті обробки запиту був створений новий ресурс
202	Accepted	Запит прийнятий сервером, але обробка його не закінчена. Даний код відповіді не гарантує, що запит буде оброблений без помилок.
206	Partial Content	Сервер повертає частину ресурсу у відповідь на запит, що містить поле заголовка Range
301	Multiple Choice	Запит вказує більш ніж на один ресурс. У тілі відповіді можуть міститися вказівки на те, як правильно ідентифікувати запрошуваний ресурс
302	Moved Permanently	Запрошуваний ресурс більше не розташовується на сервері
302	Moved Temporarily	Запрошуваний ресурс тимчасово змінив свою адресу
400	Bad Request	У запиті клієнта виявлена синтаксична помилка
403	Forbidden	Ресурс, що є на сервері, недоступний для даного користувача
404	Not Found	Ресурс, вказаний клієнтом, на сервері відсутній
405	Method Not Allowed	Сервер не підтримує метод, вказаний в запиті
500	Internal Server Error	Один з компонентів сервера працює некоректно
501	Not Implemented	Функціональних можливостей сервера недостатньо, щоб виконати запит клієнта
503	Service Unavailable	Служба тимчасово недоступна
505	HTTP Version not Supported	Версія HTTP, вказана в запиті, не підтримується сервером

У відповіді використовується така ж структура полів заголовка, як і в запиті клієнта. Поля заголовка призначені для того, щоб уточнити відповідь сервера клієнтові.

У тілі відповіді міститься код ресурсу, що передається клієнтові у відповідь на запит. Це не обов’язково має бути HTML-текст веб-сторінки. У складі відповіді можуть передаватися зображення, аудіо-файл, фрагмент відеоінформації, а також будь-який інший тип даних, підтримуваний клієнтом. Про те, як слід обробляти отриманий ресурс, клієнтові повідомляє вміст поля заголовка Content-type.

Таблиця 1.2.1.3

Поля заголовка відповіді веб-сервера

Ім’я поля	Опис вмісту
1	2
Server	Ім’я і номер версії сервера
Age	Час в секундах, що минув з моменту створення ресурсу
Allow	Список методів, допустимих для даного ресурсу
Content-Language	Мови, які повинен підтримувати клієнт для того, щоб коректно відображати ресурс, що передається
Content-Type	MIME-тип даних, що містяться в тілі відповіді сервера
Content-Length	Число символів, що містяться в тілі відповіді сервера



Закінчення таблиці 1.2.1.3

1	2
Last-Modified	Дата і час останньої зміни ресурсу
Date	Дата і час, що визначають момент генерації відповіді
Expires	Дата і час, що визначають момент, після якого інформація, передана клієнтові, вважається застарілою
Location	У цьому полі вказується реальне розташування ресурсу. Воно використовується для перенаправлення запиту
Cache-Control	Директиви управління кешуванням. Наприклад, по-cache означає, що дані не повинні кешуватися

Нижче представлений приклад відповіді сервера на запит, приведений в попередньому розділі. У тілі відповіді міститься початковий текст HTML-документу.

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.1
X-Powered-By: ASP.NET

Date: Mon, 20 OCT 2012 11:25:56 GMT
Content-Type: text/html
Accept-Ranges: bytes
Last-Modified: Sat, 18 Oct 2012 15:05:44 GMT
ETag: «b66a667f948c92:8a5»
Content-Length: 426

<html>
<body>

<form action='http://localhost/Scripts/test.pl'>
<p>Operand1: <input type='text' name='A'></p>
<p>Operand2: <input type='text' name='B'></p>
<p>Operation:<br>
<select name='op'>
<option value='+'>+</option>
<option value='-'>-</option>
<option value='*'>*</option>
<option value='/'>/</option>
</select></p>
<input type='submit' value='Calculate!'>
</from>
</body>
</html>
```

Поля заголовку і тіло повідомлення можуть бути відсутніми, але рядок стану є обов'язковим елементом, оскільки вказує на тип запиту чи відповіді.

Поле з ім'ям Content-type може зустрічатися як в запиті клієнта, так і у відповіді серверу. В якості значення цього поля вказується MIME-тип вмісту запиту або відповіді. MIME-тип також передається в полі заголовку Асерт, присутнього в запиті.

Специфікація MIME (Multipurpose Internet Mail Extension – багатоцільове поштове розширення Internet) спочатку була розроблена для того, щоб забезпечити передачу різних форматів даних у складі електронних листів. Проте вживання MIME не вичерпується електронною поштою. Засоби MIME успішно використовуються у WWW і, по суті, стали невід'ємною частиною цієї системи.

Стандарт MIME розроблений як розширювана специфікація, в якій число типів даних зростатиме у міру розвитку форм представлення даних. Кожен новий тип в обов'язковому порядку має бути зареєстрований в IANA (Internet Assigned Numbers Authority).

До появи MIME комп'ютери, що взаємодіють по протоколу HTTP, обмінювалися виключно текстовою інформацією. Для передачі зображень, як і для передачі будь-яких інших двійкових файлів, доводилося користуватися протоколом FTP.

Відповідно до специфікації MIME, для опису формату даних використовуються тип і підтип. Тип визначає, до якого класу відноситься формат вмісту HTTP-запиту або HTTP-відповіді. Підтип уточнює формат. Тип і підтип відділяються один від одного косою рискою:

тип/підтип

Оскільки в переважній більшості випадків у відповідь на запит клієнта сервер повертає початковий текст HTML-документа, то в полі Content-type відповіді зазвичай міститься значення text/html. Тут ідентифікатор text описує тип, повідомляючи, що клієнтові передається символічна інформація, а ідентифікатор html описує підтип, тобто вказує на те, що послідовність символів, що міститься в тілі відповіді, є описом документу на мові HTML.

Перелік типів і підтипів MIME досить великий. У таблиці 1.2.1.4 наведені приклади MIME-типів, що найчастіше зустрічаються в заголовках HTML-запитів і відповідей.

Таблиця 1.2.1.4

MIME типи даних

Тип/підтип	Розширення файлу	Опис
1	2	3
application/pdf	.pdf	Документ, призначений для обробки Acrobat Reader
application/msexcel	.xls	Документ у форматі Microsoft Excel
application/postscript	.ps .eps	Документ у форматі PostScript
application/x-tex	.tex	Документ у форматі Tex
application/msword	.doc	Документ у форматі Microsoft Word
application/rtf	.rtf	Документ у форматі RTF, що відображається за допомогою Microsoft Word
image/gif	.gif	Зображення у форматі GIF
image/jpeg	.jpeg .jpg	Зображення у форматі JPEG
image/tiff	.tiff .tif	Зображення у форматі TIFF
image/x-bitmap	.xbm	Зображення у форматі XBitmap
text/plain	.txt	ASCII-текст
text/html	.html .htm	Документ у форматі HTML

Закінчення таблиці 1.2.1.4

1	2	3
audio/midi	.midi .mid	Аудіофайл у форматі MIDI
audio/x-wav	.wav	Аудіофайл у форматі WAV
message/rfc822		Поштове повідомлення
message/news		Повідомлення з групи новин
video /mpeg	.mpeg .mpg, .mpe	Відеофрагмент у форматі MPEG
video/avi	.avi	Відеофрагмент у форматі AVI

Для однозначної ідентифікації ресурсів в мережі Веб-сервера використовуються унікальні ідентифікатори URL.

Ідентифікатор ресурсу URI (Uniform Resource Identifier) є короткою послідовністю символів, що ідентифікує абстрактний або фізичний ресурс. URI не вказує на те, як отримати ресурс, а тільки ідентифікує його. Це дає можливість описувати за допомогою RDF (Resource Description Framework) ресурси, які не можуть бути отримані через Інтернет (імена, назви тощо).

Найвідоміші приклади URI – це URL і URN:

- URL (Uniform Resource Locator) – це URI, який, окрім ідентифікації ресурсу, надає ще й інформацію про місцезнаходження цього ресурсу;
- URN (Uniform Resource Name) – це URI, який ідентифікує ресурс в певному просторі імен, але, на відміну від URL, URN не вказує на місцезнаходження цього ресурсу.

URL має наступну структуру:

```
<схема>://<логін>:<пароль>@<хост>:<порт>/<URL-шлях>
```

де:

- ⇒ *схема* – схема звернення до ресурсу (зазвичай мережевий протокол);
- ⇒ *логін* – ім'я користувача, використовуване для доступу до ресурсу;
- ⇒ *пароль* – пароль, що асоціюється з вказаним ім'ям користувача;
- ⇒ *хост* – повністю прописане доменне ім'я хоста в системі DNS або IP-адреси хоста;
- ⇒ *порт* – порт хоста для підключення;
- ⇒ *URL-шлях* – уточнююча інформація про місце знаходження ресурсу.

Загальноприйняті схеми (протоколи) URL включають протоколи: ftp, http, https, telnet, а також:

- gopher – протокол Gopher;
- mailto – адреса електронної пошти;
- news – новини Usenet;
- nntp – новини Usenet через протокол NNTP;
- irc – протокол IRC;
- prospero – служба каталогів Prospero Directory Service;
- wais – база даних системи WAIS;
- xmpp – протокол XMPP (частина Jabber);
- file – ім'я локального файлу;

- data – безпосередні дані (Data: URL).

1.2.2. Забезпечення безпеки передачі даних HTTP

Оскільки протокол HTTP призначений для передачі символічних даних у відкритому (незашифрованому) вигляді, то особи, що мають доступ до каналу передачі даних між клієнтом і сервером, можуть без зусиль проглядати весь трафік і використовувати його для здійснення несанкціонованих дій. У зв'язку з цим запропонована низка розширень базового протоколу спрямованих на підвищення захищеності інтернет-трафіку від несанкціонованого доступу.

Найпростішим є розширення HTTPS, при якому дані, що передаються по протоколу HTTP, «упаковуються» в криптографічний протокол SSL або TLS, тим самим забезпечуючи захист цих даних. На відміну від HTTP, для HTTPS за замовчуванням використовується TCP-порт 443. Щоб підготувати веб-сервер для обробки HTTPS з'єднань, адміністратор повинен отримати і встановити в систему сертифікат для цього веб-серверу.

SSL (Secure Sockets Layer) – криптографічний протокол, що забезпечує безпечну передачу даних по мережі Інтернет. При його використанні створюється захищене з'єднання між клієнтом і сервером. SSL спочатку розроблений компанією Netscape Communications. Згодом на основі протоколу SSL 3.0 був розроблений і прийнятий стандарт RFC, що отримав назву TLS. Цей протокол використовує шифрування з відкритим ключем для підтвердження достовірності передавача і одержувача. Підтримує надійність передачі даних за рахунок використання кодів, що коректують, і безпечних хеш-функцій. На нижньому рівні багаторівневого транспортного протоколу (наприклад, TCP) він є протоколом запису і використовується для інкапсуляції різних протоколів (POP3, IMAP, SMTP або HTTP). Для кожного інкапсульованого протоколу він забезпечує умови, при яких сервер і клієнт можуть підтверджувати один одному свою достовірність, виконувати алгоритми шифрування і проводити обмін криптографічними ключами, перш ніж протокол прикладної програми починатиме передавати і отримувати дані.

Для доступу до веб-сторінок, захищеним протоколом SSL, в URL замість схеми http, як правило, підставляється схема https, яка вказуватиме на те, що буде використовуватися SSL-з'єднання. Стандартний TCP-порт для з'єднання по протоколу https – 443. Для роботи SSL потрібно, щоб на сервері був SSL-сертифікат.

У мережі веб-сервера підтримуються 3 типи аутентифікації при клієнт-серверних взаємодіях:

- ⇒ **Basic** – базова аутентифікація, при якій ім'я користувача і пароль передаються в заголовках http-пакетів. Пароль при цьому не шифрується і присутній в чистому вигляді в кодуванні base64. Для даного типу аутентифікації використання SSL є обов'язковим;
- ⇒ **Digest** – дайджест-аутентифікація, при якій пароль користувача передається в хешованому вигляді. По рівню конфіденційності паролів цей тип мало чим відрізняється від попереднього, оскільки тому, хто атакує все одно, чи дійсно це справжній пароль або тільки хеш від нього: перехопивши посвідчення, він все одно дістає доступ до кінцевої точки. Для даного типу аутентифікації використання SSL є обов'язковим;

⇒ **Integrated** – інтегрована аутентифікація, при якій клієнт і сервер обмінюються повідомленнями для з'ясування достовірності один одного за допомогою протоколів NTLM або Kerberos. Цей тип аутентифікації захищений від перехоплення посвідчень користувачів, тому для нього не потрібен протокол SSL. Тільки при використанні даного типу аутентифікації можна працювати за схемою http, в решті всіх випадків необхідно використовувати схему https.

### 1.2.3. Cookie

Оскільки HTTP-сервер не пам'ятає передісторії запитів клієнтів, то кожен запит обробляється незалежно від інших, і у сервера немає можливості визначити, чи виходять запити від одного клієнта чи різних клієнтів.

Якщо сервер перевірятиме TCP-з'єднання і запам'ятовуватиме IP-адреси комп'ютерів-клієнтів, він все одно не зможе розрізнити запити від двох браузерів, що виконуються на одній машині. І, навіть, якщо допустити, що на комп'ютері працює лише одна клієнт-програма, то ніхто не може стверджувати, що в проміжку між двома запитами вона не була завершена, а потім запущена знову вже іншим користувачем.

Проте, якщо ви коли-небудь користувалися поштовою скринькою на mail.ru або на іншому сервері, що надає поштові послуги користувачам Веб, пригадайте, як поводився клієнт після того, як ви створили для себе поштову скриньку на сервері. Коли ви наступного разу звернулися з того ж комп'ютера до mail.ru ви, ймовірно, відмітили, що після завантаження веб-сторінки ваше реєстраційне ім'я вже відображалось у відповідному полі введення.

Такі відомості дозволяє отримати додатковий засіб під назвою cookie. Механізм cookie дозволяє серверу зберігати інформацію на комп'ютері клієнта і витягувати її звідти.

Ініціатором запису cookie виступає сервер. Якщо у відповіді сервера присутнє поле заголовку Set-cookie, клієнт сприймає це як команду на запис cookie. Надалі, якщо клієнт звертається до сервера, від якого він раніше прийняв поле заголовку Set-cookie, окрім іншої інформації він передає серверу дані cookie. Для передачі вказаної інформації серверу використовується поле заголовку Cookie.

Для того, щоб у загальних рисах уявити собі, як відбувається обмін даними cookie, розглянемо наступний приклад. Припустимо, що клієнт передає запити на сервери А, В і С. Припустимо, також, що сервер В, на відміну від А і С, передає клієнтові команду записати cookie. Послідовність запитів клієнта серверу і відповідей на них виглядатиме приблизно таким чином:

- передача запиту серверу А;
- отримання відповіді від сервера А;
- передача запиту серверу В;
- отримання відповіді від сервера В. До складу відповіді входить поле заголовка SetCookie. Отримавши його, клієнт записує cookie на диск;
- передача запиту серверу С. Не дивлячись на те, що на диску зберігається запис cookie, клієнт не робить ніяких спеціальних дій, оскільки значення cookie було записане за ініціативою іншого сервера;

- отримання відповіді від сервера С;
- передача запиту серверу А. В цьому випадку клієнт також ніяк не реагує на той факт, що на диску зберігається cookie;
- отримання відповіді від сервера А;
- передача запиту серверу В. Перед тим як сформулювати запит, клієнт визначає, що на диску зберігається запис cookie, створений після отримання відповіді від сервера В. Клієнт перевіряє, чи задовільняє даний запит деяким вимогам, і, якщо перевірка дає позитивний результат, включає в заголовок запиту поле Cookie.

Таким чином, процедуру запису і отримання cookie можна уявити собі як своєрідний «запит» сервера, інкапсульований в його відповіді клієнтові. Відповідно отримання cookie також можна уявити собі як відповідь клієнта, інкапсульовану у складі запиту тому ж серверу.

Розглянемо докладніше, які дані передаються в полі заголовка Set-cookie і, як вони впливають на поведінку клієнта.

Поле Set-cookie має наступний формат:

**Set-cookie:** ім'я = значення; expires = дата; path = шлях; domain = ім'я\_домену, secure

де

- ⇒ пара ім'я = значення – іменовані дані, що зберігаються за допомогою механізму cookie. Ці дані повинні зберігатися на клієнт-машині і передаватися серверу у складі чергового запиту клієнта;
- ⇒ дата, що є значенням параметру expires, визначає час, після закінчення якого інформація cookie втрачає свою актуальність. Якщо ключове слово expires відсутнє, дані cookie видаляються після закінчення поточного сеансу роботи браузера;
- ⇒ значення параметру domain визначає домен, з яким зв'язуються дані cookie. Щоб дізнатися, чи слід передавати у складі запиту дані cookie, браузер порівнює доменне ім'я сервера, до якого він збирається звернутися, з доменами, які пов'язані із записами cookie, що зберігаються на клієнт-машині. Результат перевірки вважатиметься за позитивний, якщо сервер, якому направляється запит, належить домену, пов'язаному з cookie. Якщо відповідність не виявлена, дані cookie не передаються;
- ⇒ шлях, вказаний як значення параметру path, дозволяє виконати подальшу перевірку і ухвалити остаточне рішення про те, чи слід передавати дані cookie у складі запиту;
- ⇒ останній параметр secure вказує на те, що дані cookie повинні передаватися по захищеному каналу.

Для передачі даних cookie серверу використовується поле заголовка Cookie. Формат цього поля наступний:

**Cookie:** ім'я=значення; ім'я=значення; ...

З допомогою поля Cookie передається одна або декілька пар ім'я = значення. Кожна з цих пар належить запису cookie, для якої URL запрошуваного ресурсу відповідають імені домену і шляху, вказаним раніше в полі Set-cookie. В другому розділі буде детальніше розкрито програмування та використання Cookie.



## ТЕМА 1.3. КЛІЄНТСЬКІ СЦЕНАРІЇ І ЗАСТОСУВАННЯ

### 1.3.1. Програми, що виконуються на клієнт-машині

### 1.3.2. Програми, що виконуються на сервері

### 1.3.3. Насичені інтернет-застосування

Як правило, Веб-застосування – це застосування, в якому клієнтом виступає браузер, а сервером – веб-сервер.

Розглянемо типи програм, що забезпечують роботу Веб-серверу і, що використовують HTTP-протокол.

Жоден HTTP-обмін неможливий без клієнта і сервера. Проте крім клієнта і сервера у веб-сеансі можуть брати участь і інші програми, які і є об'єктом веб-програмування.

Результатом роботи веб-застосування є веб-сторінка, що відображається у вікні браузеру. При цьому саме веб-застосування може виконуватися як на комп'ютері клієнта, так і на комп'ютері сервера.

Розглянемо докладніше обидві схеми.

### 1.3.1. Програми, що виконуються на клієнт-машині

Одним з типів програм, призначених для виконання на клієнт-машині, є сценарії, наприклад, JavaScript (VBScript). Початковим текстом сценарію є частина веб-сторінки, тому сценарій JavaScript передається клієнтові разом з документом, до складу якого він входить. Обробляючи HTML-документ, браузер виявляє початковий текст сценарію і запускає його на виконання.

До всіх програм, які передаються з сервера на клієнт-машини і запускаються на виконання, пред'являється одна спільна вимога: ці програми мають бути позбавлені можливості звертатися до ресурсів комп'ютера, на якому вони виконуються. Така вимога цілком обґрунтована. Адже передача по мережі і запуск Java-апплетів і JavaScript-сценаріїв відбувається автоматично без участі користувача, тому робота цих програм має бути абсолютно безпечною для комп'ютера. Іншими словами, мови, призначені для створення програм, що виконуються на клієнт-машині, мають бути абсолютно непридатні для написання вірусів і подібних програм.

### 1.3.2. Програми, що виконуються на сервері

Код програми, що працює на сервері, не передається клієнтові. При отриманні від клієнта спеціального запиту, що передбачає виконання такої програми, сервер запускає її і передає параметри, що входять до складу запиту. Засоби для генерації подібного запиту зазвичай входять до складу HTML-документу.

Результати своєї роботи програма оформляє у вигляді HTML-документа і передає їх веб-серверу, а останній, у свою чергу, доповнює отримані дані HTTP-заголовком і передає їх клієнтові. Взаємодія клієнта і сервера в цьому випадку показана на рисунку 1.3.2.1.



Рис. 1.3.2.1. Взаємодія клієнта з програмою, що виконується на сервері.

### 1.3.3. Насичені інтернет-застосування

Насичене інтернет-застосування (Rich Internet application) – ще один підхід, який полягає у використанні Adobe Flash або Java-апплетів для повної або часткової реалізації призначеного для користувача інтерфейсу, оскільки більшість браузерів підтримують ці технології (як правило, за допомогою плагінів).

Виникнення даного підходу обумовлене тим, що в рамках веб-застосувань з «тонким» клієнтом взаємодія користувача із застосуванням реалізується в істотній мірі через сервер, що вимагає відправки даних на сервер, отримання відповіді від сервера і перезавантаження сторінки на стороні клієнта.

При використанні Java-апплетів до складу HTML-документа включається спеціальний дескриптор, що описує розташування файлу, що містить код апплету на сервері. Після того, як клієнт отримує HTML-код документа, що включає апплет, він генерує додатковий запит серверу. Після того, як сервер пересилає клієнтові код апплету, сам апплет запускається на виконання. Взаємодія між клієнтом і сервером при отриманні апплету показана на рисунку 1.3.3.1.

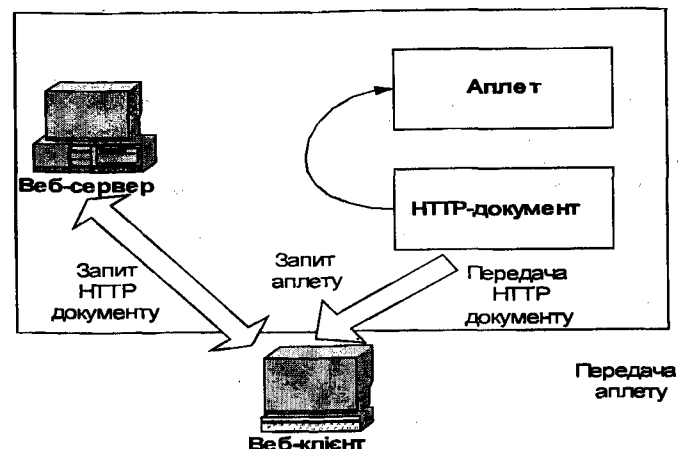


Рис. 1.3.3.1. Передача клієнтові Java-аплету

При використанні насичених інтернет-застосунків доводиться стикатися з наступними проблемами:

- необхідність забезпечення безпечного середовища виконання;
- для виконання коду має бути дозволено виконання сценаріїв;
- втрата в продуктивності (оскільки виконується на клієнтській стороні);
- потрібно багато часу на завантаження.
- Для розробки насичених інтернет-застосунків використовуються пакети Adobe Flash і Microsoft Silverlight.

## ТЕМА 1.4. СЕРВЕРНІ ВЕБ-ЗАСТОСУВАННЯ

### 1.4.1. Стандарт CGI

### 1.4.2. Сценарії

### 1.4.3. Мова Python

### 1.4.4. Мова Ruby

### 1.4.5. Технологія ASP

### 1.4.6. Інтерфейс ISAPI

Для розширення можливостей клієнт-серверної взаємодії в рамках протоколу HTTP, крім створення на клієнтській стороні розширень стандартних можливостей, що надаються мовами розмітки і браузерами, можна також розробляти на стороні веб-серверу застосування, плагіни і сценарії, що розширюють можливості самого веб-серверу.

Плагін (plug-in) – незалежно компільований програмний модуль, що динамічно підключається до основної програми, призначений для розширення або використання її можливостей. Зазвичай виконуються у вигляді бібліотек, що розділяються.

Сценарій (скрипт, script) – програма, яка автоматизує деяке завдання, яке користувач виконує вручну, використовуючи інтерфейси програми.

### 1.4.1. Стандарт CGI

Коло завдань, що вирішуються Web-сервером обмежені. В основному він зводиться до підтримки HTTP-взаємодії і доставки клієнтові Web-документів. Будь-які «нестандартні» дії реалізуються за допомогою спеціальної програми, яка взаємодіє з веб-сервером і клієнтом. Ця взаємодія здійснюється згідно певних правил.

Основний набір таких правил – стандарт CGI (Common Gateway Interface – інтерфейс спільного шлюзу), який визначає порядок запуску програми на комп'ютері-сервері, способи передачі програм і параметрів та доставки результатів її виконання клієнтові. Програма, написана по правилах CGI, називається CGI-сценарієм (script CGI), хоча це не означає, що на сервері не може виконуватися двійковий файл.

Завдяки цьому інтерфейсу для розробки застосунків можна використовувати будь-яку мову програмування, яка має в своєму розпорядженні засоби взаємодії із стандартними пристроями вводу або виводу. Такими можливостями володіють також сценарії для вбудованих командних інтерпретаторів операційних систем.

Виконання будь-якої програми (у тому числі CGI-сценарію) можна умовно розділити на п'ять етапів.

- ⇒ запуск програми;
- ⇒ ініціалізація і читання вихідних даних;
- ⇒ обробка даних;
- ⇒ виведення результатів виконання;
- ⇒ завершення програми.

Відмінності між CGI-сценарієм і консольним застосуванням стосуються першого, другого і четвертого етапів виконання.

Кожного разу, коли веб-сервер отримує запит від клієнта, він аналізує вміст запиту і повертає необхідну відповідь:

- якщо запит містить вказівку на файл, що знаходиться на жорсткому диску, то сервер повертає у складі відповіді цей файл;
- якщо запит містить вказівку на програму і необхідні для неї аргументи, то сервер виконує програму і результат її роботи повертає клієнтові.

CGI визначає:

- ⇒ яким чином інформація про сервер і запит клієнта передається програмі у формі аргументів і змінних оточення;
- ⇒ яким чином програма може передавати назад додаткову інформацію про результати (наприклад про тип даних) у формі заголовків відповіді сервера.

У переважній більшості випадків запуск CGI-сценарію здійснюється натисканням на кнопку Submit, сформованій за допомогою дескриптору `<input type = «submit»>`, який знаходиться на HTML-сторінці між `<form>` і `</form>`. Не знаючи призначення атрибутів `action` і `method`, неможливо зрозуміти, як відбувається виклик програми і передача параметрів.

Значенням атрибуту `action` дескриптора `<form>` є URL файлу, що містить код CGI-сценарію. Так, приведений нижче вираз означає, що файл з кодом CGI-сценарію знаходиться на сервері `www.myhp.edu` у каталозі `cgi-bin` у файлі `script.pl`.

```
<form action=»http://www.myhp.edu/cgi-bin/script.pl» method=»post»>
```

Існує два способи розпізнавання файлів, що містять тексти CGI-сценаріїв.

Перший спосіб полягає в тому, що при установці веб-серверу один з каталогів спеціально виділяється для зберігання сценаріїв. Зазвичай такий каталог отримує ім'я `cgi-bin` (або `Scripts` для веб-серверу IIS). В цьому випадку, якщо клієнт запрошує файл з каталогу `cgi-bin`, сервер сприймає такий запит як команду на запуск сценарію. Файли з інших каталогів інтерпретуються як HTML-документи.

Другий спосіб використовує розширення файлу. При налаштуванні серверу вказується, що файли з певними розширеннями містять коди сценаріїв.

Ідентифікація по розширенню використовується відносно рідко. Найчастіше всі сценарії поміщаються в `cgi-bin`, `/Scripts` або в інший каталог, спеціально виділений для їх зберігання.

Виведення результатів виконання CGI-сценарію здійснюється надзвичайно просто. Для того, щоб дані були передані клієнтові, досить вивести їх в стандартний вихідний потік. Проте, розробляючи CGI-сценарій, не слід забувати про те, що він все ж відрізняється від консольної програми і має наступні особливості. Інформація, що передається клієнтові, повинна відповідати протоколу HTTP, тобто складатися із заголовку і тіла відповіді. Як правило, отримавши дані від сценарію, сервер самостійно додає перший рядок заголовку.

```
HTTP/1.0      200      OK
```

Формування інформаційних полів, що входять до складу заголовку і є завданням сценарію. Щоб дані, передані сценарієм, були правильно інтерпретовані клієнтом, необхідно, щоб в заголовку було присутнє як мінімум поле `Content-type`. За заголовком

повинен слідувати порожній рядок. За відсутності полів заголовка реакція браузера буде непередбачуваною. У подібних випадках браузер зазвичай намагається відобразити отриману інформацію як текстовий файл.

Найприродніший формат для браузера – формат HTML. Результати роботи сценарію зазвичай оформляються у вигляді веб-сторінки, тобто дані, що повертаються, слід доповнити дескрипторами HTML. Таким чином, відповідь CGI-сценарію клієнтові зазвичай виглядає так:

```
Content-type: text/html
```

```
<html>
<head>
<title>відповідь сценарію</title>
</head>
<body>
.....
</body>
</html>
```

Зверніть увагу на порожній рядок після виразу `Content-type: text/html`. Він обов'язково має бути присутнім у відповіді, інакше клієнт сприйме всі подальші дані як продовження заголовку.

Після компіляції програми необхідно скопіювати виконуваний файл в каталог `cgi-bin` (або в інший каталог, призначений для розміщення виконуваних файлів) з якого він може запускатися веб-сервером на виконання по запиту клієнта.

Для виклику даного сценарію досить включити у веб-сторінку наступний фрагмент HTML-коду:

```
<form method=»post» action=»/cgi-bin/hello.exe»>
<input type=»submit»>
</form>
```

Якщо сценарій викликається з форми, йому передаються ті дані, які користувач ввів за допомогою інтерактивних елементів, що відображаються на веб-сторінці. Передача інформації CGI-сценарію здійснюється в два етапи: спочатку браузер передає дані веб-серверу, потім веб-сервер передає їх сценарію.

У більшості випадків окрім кнопки Submit форма містить інші інтерактивні елементи, кожен з яких має ім'я (атрибут `NAME`) і значення (атрибут `VALUE`, або послідовність символів, введена користувачем). З імен елементів і їх значень формується рядок параметрів, який має наступний формат.

```
ім'я=значення&ім'я=значення&... &ім'я=значення
```

Кожним параметром є ім'я елементу, що управляє, і його значення, розділені знаком рівності, а декілька таких пар об'єднують рядок за допомогою символу `&`. Якщо до складу імені або значення входить символ `&` або `=`, то подібні символи кодуються в послідовність знаку відсотка `%`, за яким слідує дві шістнадцяткові цифри, що визначають код символу. Так, наприклад, послідовністю `%21` кодується знак оклику `!`. Як правило, при передачі параметрів трьохсимвольними послідов-

ностями замінюються всі знаки, окрім латинських букв, цифр і символу пропуску (останній замінюється знаком «+»).

Таким чином, перед використанням рядка параметрів її треба декодувати. Алгоритм декодування простий і включає наступні дії:

- виділити з рядка параметрів пари ім'я = значення;
- виділити з кожної пари ім'я і значення;
- у кожному імені і кожному значенні замінити символи «+» пропусками;
- кожен послідовність з символу «%» і два шістнадцяткових числа перетворити в ASCII-символ.

Атрибут `method` дескриптору `<form>` має або значення «GET», або значення «POST». Значення «GET» і «POST» визначають два різні методи передачі параметрів сценарію:

- ⇒ якщо атрибут `method` має значення «GET», рядок параметрів передається разом з URL сценарію, що викликається. Роздільником між URL і рядком параметрів є символ «?»;
- ⇒ якщо атрибут `method` має значення «POST», рядок параметрів передається в тілі HTTP-запиту.

Розглянемо, як повинен поводитися CGI-сценарій, щоб правильно обробити дані залежно від методу, використаного при передачі даних. Рядок параметрів надсилається CGI-сценарію різними способами. Якщо атрибут `METHOD` дескриптора `<FORM>` мав значення «GET», рядок параметр передається серверу як значення змінної оточення `QUERY_STRING`.

При використанні методу `POST` дані доставляються сценарію по-іншому. Вони передаються через стандартний потік введення (`STDIN`). Щоб сценарій зміг визначити, скільки символів слід читати із стандартного введення, веб-сервер встановлює значення змінної оточення `CONTENT_LENGTH`, рівним довжині рядка параметрів.

Отримавши управління, сценарій насамперед повинен з'ясувати, за допомогою якого методу виконувалася передача параметрів. Ця інформація міститься в змінній оточення `REQUEST_METHOD`.

Таким чином, в простому випадку, щоб виконати обробку рядка параметрів, досить знати призначення трьох змінних оточення: `REQUEST_METHOD`, `QUERY_STRING` і `CONTENT_LENGTH`.

Приклад сценарію на мові Perl, яка повертає клієнтові рядок параметрів, наведений нижче. Сценарій визначає, який метод використовувався для передачі даних, читає рядок параметрів і передає його клієнтові, заздалегідь доповнивши HTML-дескрипторами.

```
$method = $ENV{'REQUEST_METHOD'};
if ($method eq «GET»)
{ $pars = $ENV{'QUERY_STRING'}; }
else
{ $length = $ENV{'CONTENT_LENGTH'}; }

read (STDIN $pars, $ length);
print «Content-type: text/html\n\n»;
```

```
print «<HTML><BODY>\n»;
print «<P>METHOD = «, $method;
print «<P>String of parameters: <P>\n»;
print $pars;
print «</HTML></BODY>\n»;
```

При розробці складніших сценаріїв може бути потрібна додаткова інформація. Інформація про типи серверу і браузеру, адресу клієнт-машини і багато інших відомостей передаються за допомогою змінних оточення.

### 1.4.2. Сценарії

До основних переваг розробки застосувань на стороні веб-серверу у формі сценаріїв можна віднести наступні:

- оскільки сценарії не компілюються, а інтерпретуються, то помилки в сценарії спричинятимуть тільки діагностичне повідомлення, але не приведуть до дестабілізації веб-серверу або операційної системи;
- кращі виразні можливості. Мова сценаріїв, як правило, має власний проблемно-орієнтований набір команд, і один рядок сценарію може робити те ж, що декілька десятків рядків на традиційній мові. Як наслідок, на цій мові може писати програміст низької кваліфікації;
- підтримка кроссплатформеності.

Оскільки сценарії інтерпретуються з початкового коду динамічно при кожному виконанні, вони виконуються зазвичай значно повільніше за готові програми, що транслюються в машинний код на етапі компіляції.

У плані швидкодії сценарні мови можна розділити на:

- мови динамічного розбору (наприклад `command.com`). Інтерпретатор прочитає інструкції з файлу програми мінімально потрібними блоками, і виконуватиме ці блоки, не читаючи подальший код;
- заздалегідь компільовані (наприклад Perl). Спочатку прочитується вся програма, потім компілюється або в машинний код, або в один з внутрішніх форматів, після чого отриманий код виконується.

### 1.4.3. Мова Python

\* **Python** – високорівнева мова програмування загального призначення з акцентом на продуктивність і читаність коду.

Мова Python поєднує в собі мінімалізм синтаксису ядра і великий обсяг корисних функцій в стандартній бібліотеці.

Мова Python («пітон», хоча деякі говорять «пайтон») – це інтерпретована об'єктно-орієнтована високорівнева мова програмування з динамічною семантикою. Вбудовані високорівневі структури даних в поєднанні з динамічною типізацією і зв'язуванням роблять мову привабливою для швидкої розробки додатків (RAD, Rapid Application Development). Крім того, її можна використовувати як сценарну мову для зв'язку про-

грамних компонентів. Синтаксис Python простий у вивченні, в ній надається особливе значення читаності коду, а це скорочує витрати на супровід програмних продуктів. Python підтримує модулі та пакети, заохочуючи модульність і повторне використання коду. Інтерпретатор Python і велика стандартна бібліотека доступні безкоштовно у вигляді вихідних і виконуваних кодів для всіх основних платформ і можуть вільно поширюватися.

Python підтримує структурну, функціональну, об'єктно-орієнтовану, імперативну, і аспектно-орієнтовану парадигми.

Основні архітектурні риси мови Python:

- ⇒ динамічна типізація;
- ⇒ автоматичне управління пам'яттю;
- ⇒ повна інтроспекція;
- ⇒ механізм обробки винятків;
- ⇒ підтримка багатопотокових обчислень;
- ⇒ зручні високорівневі структури даних.

Код в Python організовується у функції і класи, які можуть об'єднуватися в модулі (які у свою чергу можуть бути об'єднані в пакети).

Для всіх основних платформ Python має підтримку характерних для даної платформи технологій (наприклад, Microsoft COM/DCOM). Існує, навіть спеціальна версія Python для віртуальної машини Java – Jython, що дозволяє інтерпретатору виконуватися на будь-якій системі, підтримуючою Java, при цьому класи Java можуть безпосередньо використовуватися з Python і, навіть бути написаними на Python. Декілька проектів забезпечують інтеграцію з платформою Microsoft.NET, основні з яких – IronPython і Python.Net.

Стандартна бібліотека Python має засоби для роботи з багатьма мережевими протоколами і форматами Інтернету, наприклад, модулі для написання HTTP-серверів і клієнтів, для розбору і створення поштових повідомлень, для роботи з XML тощо. Набір модулів для роботи з операційною системою дозволяє писати крос-платформні застосування. Існують також модулі для роботи з регулярними виразами, текстовими кодуваннями, мультимедійними форматами, криптографічними протоколами, архівами, серіалізації даних, підтримки юніт-тестування і ін.

В процесі вивчення буде розкритий сенс цього визначення, а зараз досить знати, що Python – це універсальна мова програмування. Вона має – свої переваги і недоліки, а також сфери застосування. У поставку Python входить велика стандартна бібліотека для вирішення широкого кола завдань. В Інтернеті доступні якісні бібліотеки для Python по різних предметних областях: засоби обробки текстів і технології Інтернет, обробка зображень, інструменти для створення додатків, механізми доступу до баз даних, пакети для наукових обчислень, бібліотеки побудови графічного інтерфейсу і т.п. Крім того, Python має досить прості засоби для інтеграції з мовами C, C++ (і Java) як шляхом вбудовування (embedding) інтерпретатора в програми на цих мовах, так і навпаки, за допомогою використання бібліотек, написаних на цих мовах, в Python-програмах. Мова Python підтримує кілька парадигм програмування: імперативне (процедурний, структурний, модульний підходи), об'єктно-орієнтоване і функціональне програмування.

Можна вважати, що Python – це ціла технологія для створення програмних продуктів (і їх прототипів). Вона доступна майже на всіх сучасних платформах (як 32-бітових, так і на 64-бітних) з компілятором C і на платформі Java.

### 1.4.3.1. Опис мови Python

У цій темі не ставиться мета систематично описати Python: для цього існує оригінальне довідкове керівництво. Тут пропонується розглянути мову одночасно в декількох аспектах, що досягається набором прикладів, які дозволять швидше долучитися до реального програмування, ніж у випадку строгого академічного підходу. Проте варто звернути увагу на правильний підхід до опису мови. Створення програми – це завжди комунікація, в якій програміст передає комп'ютеру інформацію, необхідну для виконання останньому дій. Те, як ці дії розуміє програміст, можна назвати семантикою. Засобом передачі цього розуміння є синтаксис мови програмування. Ну а те, що робить інтерпретатор на підставі переданого, зазвичай називають прагматикою. При написанні програми дуже важливо, щоб в цьому ланцюжку не виникало збоїв. Синтаксис – повністю формалізована частина: його можна описати на формальній мові синтаксичних діаграм (що й робиться в довідкових посібниках). Виявом прагматики є сам інтерпретатор мови. Саме він читає записане відповідне до синтаксису «повідомлення» і перетворює його в дії по закладеному в ньому алгоритму. Неформальним компонентом залишається тільки семантика. Саме в перекладі сенсу в формальний опис і криється найбільша складність програмування. Синтаксис мови Python має потужні засоби, які допомагають наблизити розуміння проблеми програмістом до її «розуміння» інтерпретатором.

Створення Python було розпочато Гвідо ван Россум (Guido van Rossum) в 1991 році, коли він працював над розподіленою операційною системою Амеба. Йому потрібна була розширювана мова, яка би забезпечила підтримку системних викликів. За основу були взяті ABC і Модуль-3. В якості назви він вибрав Python в честь комедійних серій BBC «Літаючий цирк Монті Пітона», а зовсім не за назвою змії. З тих пір Python розвивався за підтримки тих організацій, в яких Гвідо працював. Особливо активно мова вдосконалюється в даний час, коли над ним працює не тільки команда творців, але й ціле співтовариство програмістів зі всього світу. І все-таки останнє слово про напруження розвитку мови залишається за Гвідо ван Россумом.

Програма на мові Python може складатися з одного або декількох модулів. Кожен модуль являє собою текстовий файл у кодуванні, сумісному з 7-бітним кодуванням ASCII. Для кодувань, що використовують старший біт, необхідно явно вказувати назву кодування. Наприклад, модуль, коментарі або рядкові літерали якого записані в кодуванні KOI8-R, повинен мати в першому або другому рядку наступну специфікацію:

# - \* - Coding: koi8-r - \* -

Завдяки цій специфікації інтерпретатор Python буде знати, як коректно переводити символи літералів Unicode-рядків в Unicode. Без цього рядка нові версії Python будуть видавати попередження на кожен модуль, в якому зустрічаються коди до встановленого восьмим бітом. Про те, як робити програму модульною, стане відомо далі.



У прикладах нижче використовуються як фрагменти модулів, записаних в файл, так і фрагменти діалогу з інтерпретатором Python. Останні відрізняються характерним запрошенням `>>>`. Символ решітки (#) визначає коментар до кінця рядка. Програма на Python, з точки зору інтерпретатора, складається з логічних рядків. Один логічний рядок, як правило, розташовується в одному фізичному, але довгі логічні рядки можна явно (за допомогою зворотної косої риски) або неявно (в середині дужок) розбити на кілька фізичних:

```
print a, «- дуже довгий рядок, який не поміщається в», \ 80.
```

### 1.4.3.2. Основні алгоритмічні конструкції

Послідовність операторів. Послідовні дії описуються послідовними рядками програми. Варто, що правда, додати, що в програмах важливі відступи, тому всі оператори, що входять до послідовності дій, повинні мати один і той же відступ:

```
a = 1
b = 2
a = a + b
b = a - b
a = a - b
print a, b
```

Що робить цей приклад? Перевірити свою здогадку можна за допомогою інтерактивного режиму інтерпретатора Python. При роботі з Python в інтерактивному режимі вводиться одна велика програма, що складається з послідовних дій. У прикладі вище використані оператори присвоювання і оператор print.

При написанні алгоритмів використовується ще й розгалуження:

```
if a > b:
    c = a
else:
    c = b
```

Оператор розгалуження має в даному випадку дві частини, оператори кожної з яких записуються з відступом вправо щодо оператора розгалуження. Більш загальний випадок – оператор вибору можна записати за допомогою наступного синтаксису (приклад обчислення знака числа):

```
if a < 0:
    s = -1
elif a == 0:
    s = 0
else:
    s = 1
```

Варто зауважити, що `elif` – це скорочений `else if`. Без скорочення довелося б застосувати вкладений оператор розгалуження:

```
if a < 0:
    s = -1
```

```
else:
    if a == 0:
        s = 0
    else:
        s = 1
```

На відміну від оператора `print`, оператор `if-else` – складений оператор.

Третьою необхідною алгоритмічною конструкцією є цикл. За допомогою циклу можна описати повторювані дії. В Python є два види циклів: цикл ПОКИ (Виконується деяка умова) і цикл ДЛЯ (всіх значень послідовності). Наступний приклад ілюструє цикл ПОКИ на Python:

```
s = «abcdefghijklmnop»
while s! = «»:
    print s
    s = s [1: -1]
```

Оператор `while` говорить інтерпретатору Python: «поки вірна умова циклу, виконувати тіло циклу». У мові Python тіло циклу виділяється відступом. Кожне виконання тіла циклу буде називатися ітерацією. У наведеному прикладі забирається перший і останній символ рядка до тих пір, поки не залишиться порожній рядок.

Для більшої гнучкості при організації циклів застосовуються оператори `break` (Перервати) і `continue` (Продовжити). Перший дозволяє перервати цикл, а другий – продовжити цикл, перейшовши до наступної ітерації (якщо, звичайно, виконується умова циклу).

Наступний приклад читає рядки з файлу і виводить ті, у яких довжина більше 5:

```
f = open («file.txt», «r»)
while 1:
    l = f.readline ()
    if not l:
        break
    if len (l) > 5:
        print l,
    f.close ()
```

У цьому прикладі організований нескінченний цикл, який переривається лише при отриманні з файлу порожнього рядка (`l`). Що позначає кінець файлу. У мові Python логічне значення несе кожен об'єкт: нулі, порожні рядки і послідовності, спеціальний об'єкт `None` і логічний літерал `False` мають значення «брехня», а інші об'єкти значення «істина». Для позначення істини зазвичай використовується `1` або `True`.

*Примітка:*

Літерали `True` і `False` для позначення логічних значень з'явилися в Python 2.3. Цикл ДЛЯ виконує тіло циклу для кожного елементу послідовності. У наступному прикладі виводиться таблиця множення:

```
for i in range (1, 10):
    for j in range (1, 10):
        print «% 2i» % (i * j),
    print
```

Тут цикли `for` є вкладеними. Функція `range()` утворює список цілих чисел з напіввідкритого інтервалу `[1, 10)`. Перед кожною ітерацією лічильник циклу отримує чергове значення з цього списку. Напіввідкриті діапазони загальноприйняті в Python. Вважається, що їх використання більш зручне і викликає менше програмістських помилок. Наприклад, `range(len(s))` утворює список індексів для списку `s` (В Python-послідовності перший елемент має індекс 0). Для гарного виведення таблиці множення застосована операція форматування `%` (Для цілих чисел той же символ використовується для позначення операції отримання залишку від ділення). Рядок форматування (задається ліворуч) буде майже як рядок форматування для `printf` з C.

### 1.4.3.3. Функції в Python

Програміст може визначати власні функції двома способами: за допомогою оператора `def` або прямо у виразі, за допомогою `lambda`. Другий спосіб (та й взагалі робота з функціями) буде розглянуто детальніше нижче, а тут слід привести приклад визначення і виклику функції:

```
def cina (grn, kop = 0):
    return "% і грн.% і коп.» % (Grn, kop)
print cina (8, 50)
print cina (7)
print cina (grn = 23, kop = 70)
```

У цьому прикладі визначена функція двох аргументів (з яких другий має значення за замовчуванням – 0). Варіантів виклику цієї функції з конкретними параметрами також декілька. Варто лише зауважити, що при виконанні функції спочатку повинні йти позиційні параметри, а потім, іменовані. Аргументи зі значеннями за замовчуванням повинні слідувати після звичайних аргументів. Оператор `return` повертає значення функції. З функції можна повернути тільки один об'єкт, але він може бути кортежем з декількох об'єктів. Після оператора `def` ім'я `cina` виявляється пов'язаним з функціональним об'єктом.

Винятки:

У сучасних програмах передача управління відбувається не завжди так гладко, як в описаних вище конструкціях. Для обробки особливих ситуацій (таких як ділення на нуль або спроба читання з неіснуючого файлу) застосовується механізм винятків. Краще всього пояснити синтаксис оператора `try-except` таким прикладом:

```
try:
    res = int (open ('a.txt'). read ()) / int (open ('c.txt'). read ())
    print res
except IOError:
    print «Помилка вводу-виводу «
except ZeroDivisionError:
    print «Ділення на 0 «
except KeyboardInterrupt:
    print «Переривання з клавіатури «
except:
    print «Помилка»
```

У цьому прикладі беруться числа з двох файлів і діляться одне на інше. В результаті цих нехитрих дій може виникнути кілька виняткових ситуацій, деякі з них відзначені в частинах `except` (тут використані стандартні вбудовані винятки Python). Остання частина `except` в цьому прикладі вловлює всі інші винятки, які не були спіймані вище. Наприклад, якщо хоча б в одному з файлів знаходиться нечислові значення, функція `int()` порушить виняток `ValueError`. Його-то і зможе відловити остання частина `except`. Зрозуміло, що виконання частини `try` у разі виникнення помилки вже не продовжується після виконання однієї з частин `except`. На відміну від інших мов програмування, в Python виключення нерідко служать для спрощення алгоритмів. Записуючи оператор `try-except`, програміст може думати так: «спробую, а якщо зірветься – виконається код в `except`». Особливо часто це використовується для виразів, в яких значення виходить по ключу з відображення:

```
try:
    value = dict[key]
except:
    value = default_value
Замість
if dict.has_key(key):
    value = dict[key]
else:
    value = default_value
```

*Примітка:*

Приклад уже дещо застарілої ідіоми мови Python ілюструє тільки дух цього підходу: у сучасному Python краще записати так `value = dict.get(key, default_value)`. Винятки можна порушувати і з програми. Для цього служить оператор `raise`. Заодно наступний приклад показує канонічний спосіб визначення власного виключення:

```
class MyError (Exception):
    pass
try:
    ...
    raise MyError, «my error 1»
    ...
except MyError, x:
    print «Помилка:», x
```

До речі, всі винятки збудовані в ієрархію класів, тому `ZeroDivisionError` може бути спіймана як `ArithmeticError`, якщо відповідна частина `except` буде йти раніше. Для тверджень застосовується спеціальний оператор `assert`. Він звертається до `AssertionError`, якщо задано в ньому умову невірно. Цей оператор використовують для самоперевірки програми. В оптимізованому коді він не виконується, тому будувати на ньому логіку алгоритму не можна. *Приклад:*

```
c = a + b
assert c == a + b
```

Крім описаної форми оператора є ще форма `try-finally` для гарантованого виконання деяких дій при передачі управління з оператора `try-finally` зовні. Він може застосовуватися для звільнення зайнятих ресурсів, що вимагає обов'язкового виконання, незалежно від того, що відбулося в середині:

```
try:
...
finally:
print «Обробка гарантовано завершена»
```

#### 1.4.3.4. Підтримка мережі та функціональне програмування на Python

Майже всі модулі з цієї категорії, які обслуговують клієнтську частину протоколу, побудовані на одному принципі: з модуля необхідний тільки клас, об'єкт якого містить інформацію про з'єднання з сервером, а методи реалізують взаємодію з сервером за відповідним протоколом. Таким чином, чим складніше протокол, тим більше методів та інших деталей потрібно для реалізації клієнта. Приклади серверів використовуються за іншим принципом. У модулі з реалізацією сервера описаний базовий клас, з якого користувач модуля повинен успадковувати свій клас, який реалізує необхідну функціональність. Правда, іноді заміщати потрібно всього один або два методи.

**Підтримка Internet.** У стандартній бібліотеці Python є різноманітні модулі для роботи з різними форматами, що застосовуються для кодування даних в мережі. Сьогодні найбільш потужним інструментом для обробки повідомлень в форматі RFC 2822 є пакет `email`. З його допомогою можна як розбирати повідомлення в зручному для програмної обробки вигляді, так і формувати повідомлення на основі даних про поля і основний зміст (включаючи вкладення).

Мова Python є рефлексивною мовою, в якій можна «заглянути» глибоко у власні внутрішні структури коду і даних. Модулі цієї категорії дають можливість доторкнутися до внутрішнього пристрою Python.

Майже всі сучасні програми мають графічний інтерфейс користувача. Такі програми можна створювати і мовою Python. У стандартному постачанні є модуль `Tkinter`, який є не що інше, як інтерфейс до мови `Tcl/Tk`, на якій можна описувати графічний інтерфейс. Слід зазначити, що існують і інші пакети для програмування графічного інтерфейсу: `wxPython` (заснований на `wxWindows`), `PyGTK` і т.д. Серед цих пакетів в основному такі, що працюють на одній платформі (рідше – на двох). Крім можливостей програмного опису графічного інтерфейсу, для Python є кілька комерційних і некомерційних розбудовників графічного інтерфейсу (`GUI builders`), проте в даному курсі вони не розглядаються.

Функції є абстракціями, в яких деталі реалізації деякої дії ховаються за окремим ім'ям. Добре написаний набір функцій дозволяє використовувати їх багато разів. Стандартна бібліотека Python містить безліч готових і налагоджених функцій, багато з яких досить універсальні, щоб працювати з широким спектром вхідних даних. Навіть якщо певна ділянка коду не використовується кілька разів, але по вхідним і вихідним даним вона досить автономна, то її сміливо можна виділити в окрему функцію.

Далі будуть детально розглянуті опис і використання функцій в Python, рекурсія, передача та повернення функцій як параметрів, обробка послідовностей та ітератори, а також таке поняття як генератор. Буде продемонстровано, що в Python функції є об'єктами (а, отже, можуть бути передані в якості параметрів і повернуті в результаті виконання функцій). Крім того, мова піде про те, як можна реалізувати деякі механізми функціонального програмування, що не мають в Python прямої синтаксичної підтримки, але широко поширені в мовах функціонального програмування. Функціональне програмування – це стиль програмування, що використовує тільки композиції функцій. Іншими словами, це програмування у виразах, а не в імперативних командах.

Як відзначає Девід Мертц (David Mertz) в своїй статті про функціональне програмування на Python, «функціональне програмування – це програмування на функціональних мовах (`LISP`, `ML`, `OCAML`, `Haskell`, ...)», основними атрибутами яких є:

- наявність функцій першого класу (функції нарівні з іншими об'єктами можна передавати в середину функцій);
- рекурсія є головною керуючою структурою у програмі;
- обробка списків (послідовностей);
- заборона побічних ефектів у функцій, що в першу чергу означає відсутність присвоювання (в «чистих» функціональних мовах);
- заборона операторів, де основна увага робиться на вираз. Замість операторів вся програма в ідеалі – один вислів із супутніми визначеннями;
- ключове питання: що потрібно обчислити, а не як;
- використання функцій більш високих порядків (функції над функціями).

**Функціональна програма.** Математичні функції (їх називають класичними) знаходять результат по заданим аргументам. Класичні функції не повинні зберігати в собі будь-які дані між двома викликами. Їх можна уявляти собі чорними ящиками, про які відомо тільки те, що вони роблять, але зовсім не важливо як.

Програми у функціональному стилі конструюються як композиція функцій. При цьому функції розуміються майже так само, як і в математиці: вони відображають одні об'єкти в інші. У програмуванні «чисті» функції – ідеал, не завжди досяжний на практиці. Практично корисні функції зазвичай мають побічний ефект: зберігають стан між викликами або змінюють стан інших об'єктів. Наприклад, без побічних ефектів неможливо уявити собі функції вводу-виводу. Власне, такі функції заради цих «ефектів» і використовуються. Крім того, математичні функції легко працюють з об'єктами, які вимагають нескінченного обсягу інформації (наприклад, дійсні числа). У загальному випадку комп'ютерна програма може виконати лише наближені обчислення.

Бінарні операції «+», «-», «\*», «/», які записуються у виразах, є «математичними» функціями над двома аргументами – операндами. Їх використовують настільки часто, що синтаксис мови програмування має для них більш короткий запис. Модуль `operator` дозволяє представляти ці операції у функціональному стилі:

```
>>> From operator import add, mul
```

```
>>> Print add (2, mul (3, 4))
```

**Функція: визначення і виклик.** Як уже говорилося, визначити функцію в Python можна двома способами: за допомогою оператора `def` і `lambda`-висловлювання. Перший спосіб дозволяє використовувати оператори. При другому способі – визначення функції може бути тільки виразом.

Забігаючи вперед, можна помітити, що методи класів визначаються так само, як і функції. Відмінність полягає в спеціальному сенсі першого аргументу `self` (у ньому передається екземпляр класу).

Найкраще розглянути синтаксис визначення функції на декількох прикладах. Після визначення відповідної функції показаний один або кілька варіантів її виклику (деякі приклади взяті зі стандартної бібліотеки).

Визначення функції має містити список формальних параметрів і тіло визначення функції. У випадку з оператором `def`–функції також задається деяке ім'я. Формальні параметри є локальними іменами в середині тіла визначення функції, а при виконанні функції вони виявляються пов'язаними з об'єктами, переданими як фактичні параметри. Значення за замовчуванням обчислюються в момент виконання оператора `def`, і тому в них можна використовувати видимі на момент визначення імена.

Виклик функції синтаксично виглядає як об'єкт функції (фактичні параметри). Звичай об'єкт функції – це просто ім'я функції, хоча це може бути і будь-який вираз, який в результаті обчислення дає виконуваний об'єкт.

Функція одного аргументу:

```
def swapcase (s):
    return s.swapcase ()
print swapcase («ABC»)
```

Функція двох аргументів, один з яких необов'язковий і має значення за замовчуванням:

```
def inc (n, delta = 1):
    return n + delta
print inc (12)
print inc (12, 2)
```

Функція з одним обов'язковим аргументом, у яких значення за замовчуванням і невизначеним числом іменованих аргументів:

```
def wrap (text, width = 70, ** kwargs):
    from textwrap import TextWrapper
    # Kwargs – словник з іменами та значеннями аргументів
    w = TextWrapper (width = width, ** kwargs)
    return w.wrap (text)
print wrap («my long text ...», width = 4)
```

Функція довільного числа аргументів:

```
def max_min (* args):
    # Args – список аргументів у порядку їх вказівки при виклику
    return max (args), min (args)
print max_min (1, 2, -1, 5, 3)
```

Функція зі звичайними (позиційними) і іменованими аргументами:

```
def swiss_knife (arg1, * args, ** kwargs):
    print arg1
    print args
    print kwargs
    return None
print swiss_knife (1)
print swiss_knife (1, 2, 3, 4, 5)
print swiss_knife (1, 2, 3, a = 'abc', b = 'sdf')
# Print swiss_knife (1, a = 'abc', 3, 4) #! помилка
lst = [2, 3, 4, 5]
dct = {'a': 'abc', 'b': 'sdf'}
print swiss_knife (1, * lst, ** dct)
```

Приклад визначення функції за допомогою `lambda`-виразу подана нижче:

```
func = lambda x, y: x + y
```

В результаті `lambda`-висловлювання виходить безіменний об'єкт функції, який потім використовується, наприклад, для того, щоб зв'язати з нею деяке ім'я. Однак, як правило, визначаються `lambda`-вирази функції, що застосовуються в якості параметрів функцій.

У мові Python функція може повернути тільки одне значення, яке може бути кортежем. У наступному прикладі видно, як стандартна функція `divmod ()` повертає ціле і залишок від ділення двох чисел:

```
def bin (n):
    «» «Цифри двійкового представлення натурального числа» «»
    digits = []
    while n > 0:
        n, d = divmod (n, 2)
        digits = [d] + digits
    return digits
print bin (69)
```

Примітка:

Важливо зрозуміти, що під ім'ям функції розуміється об'єкт. Цей об'єкт можна зв'язати з іншим ім'ям:

```
def add (x, y):
    return x + y
addition = add # тепер addition і add – різні імена одного і того ж об'єкта
```

Приклад, в якому у якості значення за замовчуванням аргументу функції використовують змінний об'єкт (список). Цей об'єкт – один і той же для всіх викликів функцій, що може привести до казусів:

```
def mylist (val, lst = []):
    lst.append (val)
    return lst
print mylist (1),
print mylist (2)
```

Замість очікуваного [1] [2] виходить [1] [1, 2], так як додаються елементи за «значенням за замовчуванням».

Правильний варіант вирішення буде, наприклад, таким:

```
def mylist (val, lst = None):
    lst = lst or []
    lst.append (val)
    return lst
```

Наведена вище форма може використовуватися для зберігання у функції деякого стану між її викликами, проте, практично завжди замість функції з таким побічним ефектом краще написати клас і використовувати його екземпляр.

**Функції як параметри і результат.** Як уже не раз говорилося, функції є такими ж об'єктами Python як числа, рядки або списки. Це означає, що їх можна передавати в якості параметрів функцій або повертати з функцій.

Функції, які беруть в якості аргументів чи повертають інші функції в результаті, називають функціями вищого порядку. В Python функції вищого порядку застосовуються програмістами досить часто. У більшості випадків таким чином будується механізм зворотних викликів (callbacks), але зустрічаються й інші варіанти. Наприклад, алгоритм пошуку може викликати передану йому функцію для кожного знайденого об'єкта.

Застосовувати для обробки даних явні послідовності не завжди ефективно, так як на зберігання тимчасових даних може витрачатися багато оперативної пам'яті. Більш ефективним рішенням є використання ітераторів – спеціальних об'єктів, що забезпечують послідовний доступ до даних контейнера. Якщо у виразі є операції з ітераторами замість контейнерів, проміжні дані не будуть вимагати багато місця для зберігання – адже вони запитуються у міру необхідності для обчислень. При обробці даних з використанням ітераторів пам'ять буде вимагатися тільки для вихідних даних і результату, та й то необов'язково вся відразу – адже дані можуть читатися і записуватися у файл на диску.

Ітератори можна застосовувати замість послідовності в операторі for. Більше того, внутрішньо оператор for запитує від послідовності її ітератор. Об'єкт файлового типу теж (порядковий) ітератор, що дозволяє обробляти великі файли, не прочитуючи їх цілком в пам'яті.

Там, де потрібен ітератор, можна використовувати послідовність.

Використовувати ітератор можна і «вручну». Будь-який об'єкт, що підтримує інтерфейс ітератора має метод next (), який при кожному виклику видає чергове значення ітератора. Якщо більше значень немає, порушується виняток StopIteration. Для отримання ітератора по деякому об'єкту необхідно перш застосувати до цього об'єкта функцію iter () (цикл for робить це автоматично).

В Python є модуль itertools, який містить набір функцій, комбінуючи які, можна скласти досить складні схеми обробки даних за допомогою ітераторів. Далі розглядаються деякі функції цього модуля.

**Функція iter ()** має два варіанти використання. У першому вона приймає лише один аргумент, який повинен «вміти» надавати свій ітератор. У другому один з аргументів – функція без аргументів, інший – шукані значення. Ітератор викликає за-

значену функцію до тих пір, поки та не поверне шукане значення. Другий варіант зустрічається багато рідше першого і зазвичай у середині методу класу, так як складно породжувати значення «на порожньому місці»:

```
it1 = iter ([1, 2, 3, 4, 5])
def forit (mystate = []):
    if len (mystate) < 3:
        mystate.append («»)
    return «»
it2 = iter (forit, None)
print [x for x in it1]
print [x for x in it2]
```

Примітка: якщо функція не повертає значення явно, вона повертає None, що й використано в прикладі вище.

**Прості генератори.** Розробники мови не зупинилися на ітераторах. Як виявилось, в інтерпретаторі Python досить просто реалізувати прості генератори. Під цим терміном фактично розуміється спеціальний об'єкт, обчислення, в якому тривають до вироблення чергового значення, а потім припиняються до виникнення необхідності у видачі такого значення. Простий генератор формується функцією-генератором, яка синтаксично схожа на звичайну функцію, але використовує спеціальний оператор yield для видачі такого значення. При виклику така функція нічого не обчислює, а створює об'єкт з інтерфейсом ітератора для отримання значень. Іншими словами, якщо функція повинна повертати послідовність, з неї досить просто зробити генератор, який буде функціонально еквівалентним реалізації вихідної дії функції. Вихідною дією функції називаються обчислення, які відкладаються до самого останнього моменту, коли отримується в результаті значення, яке відразу використовується в іншому обчисленні.

Для прикладу з послідовністю Фібоначчі можна побудувати такий ось генератор:

```
def Fib (N):
    a, b = 0, 1
    for i in xrange (N):
        yield a
        a, b = b, a + b
```

Використовувати його не складніше, ніж будь-який інший ітератор:

```
for i in Fib (100):
    print i,
```

Однак слід зауважити, що програма значною мірою спростилася.

**Генераторний вираз.** В Python 2.4 за аналогією зі списковим включенням з'явився генераторний вираз. За синтаксисом він аналогічний списковому, але замість квадратних дужок використовуються круглі. Спискове включення утворює список, а, значить, можна ненароком зайняти дуже багато пам'яті. Генератор ж, що виходить в результаті застосування генераторного виразу, списку не створює, він обчислює кожне наступне значення строго на вимогу (при виклику методу next ()).



У наступному прикладі можна прочитати з файлу рядки, в яких виробляються деякі заміни:

```
for line in (l.replace («-», «>») for l in open («input.dat»)):
    print line
```

Ніщо не заважає використовувати ітератори і для запису у файл:

```
open («output.dat», «w»).writelines (
    l.replace («-», «>») for l in open («input.dat»))
```

Тут для генераторного виразу не потрібно додаткових дужок, так як вони розташовані в середині дужок виклику функції.

Бібліотека Xoltar toolkit (автор Bryn Keller) включає модуль functional, який дозволяє спростити використання можливостей функціонального програмування. Модуль functional застосовує «чистий» Python. Бібліотеку можна знайти за адресою: <http://sourceforge.net/projects/xoltar-toolkit>.

При каррінгу (частковому застосуванні) функції створюється нова функція, використовуючи деякі аргументи вихідної. Наступний приклад ілюструє часткове застосування віднімання:

```
from functional import curry
def subtract (x, y):
    return x - y
print subtract (3, 2)
subtract_from_3 = curry (subtract, 3)
print subtract_from_3 (2)
print curry (subtract, 3) (2)
```

У всіх трьох випадках буде виведено 1. У наступному прикладі виходить нова функція, підставляючи другий аргумент. Замість іншого аргументу вставляється спеціальне значення Blank:

```
from functional import curry, Blank
def subtract (x, y):
    return x - y
print subtract (3, 2)
subtract_2 = curry (subtract, Blank, 2)
print subtract_2 (3)
print curry (subtract, Blank, 2) (3)
```

Слід зазначити, що ітератори – це практичне продовження функціонального початку в мові Python. Ітератори дозволяють організувати лінійні обчислення (lazy computations), при яких значення обчислюються лише коли вони безпосередньо потрібні.

#### 1.4.3.5. Розробка Web-додатків

Під web-додатком буде розумітися програма, основний інтерфейс користувача у якій працює в стандартному WWW-браузері під управлінням HTML і XML-документів. Для поліпшення якості інтерфейсу користувача часто застосовують

JavaScript, однак це дещо знижує універсальність інтерфейсу. Слід зауважити, що інтерфейс можна побудувати на Java-або Flash-аплетах, однак, такі додатки складно назвати web-додатками, так як Java або Flash можуть використовувати власні протоколи для спілкування з сервером, а не стандартний для WWW протокол HTTP.

При створенні web-додатків намагаються відокремити Форму (зовнішній вигляд, стиль), Зміст і Логіку обробки даних. Сучасні технології побудови web-сайтів дають можливість підійти досить близько до цього ідеалу. Тим не менше, навіть без застосування багаторівневих додатків можна дотримуватися стилю, що дозволяє змінювати будь-який з цих аспектів, не зачіпаючи (або майже не зачіпаючи) двох інших.

Як вже згадувалось, класичний шлях створення додатків для WWW – написання CGI-сценаріїв (іноді кажуть – скриптів). CGI (Common Gateway Interface, загальний шлюзовий інтерфейс) – це стандарт, що регламентує взаємодію сервера із зовнішніми додатками. У випадку з WWW, web-сервер може направити запит на генерацію сторінки за певним сценарієм. Цей сценарій, отримавши на вхід дані від web-сервера (той, в свою чергу, міг отримати їх від користувача), генерує готовий об'єкт (зображення, аудіодані, таблицю стилів і т.п.).

При виклику сценарію Web-сервер передає йому інформацію через стандартний ввід, змінні оточення і, для ISINDEX, через аргументи командного рядка (вони доступні через sys.argv).

Два основні методи передачі даних із заповненої в браузері форми Web-сервера (і CGI-сценарієм) – GET і POST. В залежності від методу дані передаються по-різному. У першому випадку вони кодуються і поміщаються прямо в URL, наприклад: <http://host/cgi-bin/a.cgi?a=1&b=3>. Сценарій отримує їх у змінній оточення з ім'ям QUERY\_STRING. У випадку методу POST вони передаються на стандартний ввід.

Як вже згадувалось, для коректної роботи сценарії поміщаються в призначений для цього каталог на web-сервері (зазвичай він називається cgi-bin) або, якщо це дозволено конфігурацією сервера, в будь-якому місці серед документів HTML. Сценарій повинен мати ознаку наповнюваності. В системі Unix його можна встановити за допомогою команди `chmod a+x`.

Наступний найпростіший сценарій виводить значення із словника `os.environ` і дозволяє побачити, що ж було йому передано:

```
#!/usr/bin/python
import os
print «» «Content-Type: text/plain»
% S «» «% os.environ
```

За допомогою нього можна побачити встановлені Web-сервером змінні оточення. Відправлений CGI-сценарієм web-серверу файл містить заголовкову частину, в якій вказані поля з мета-інформацією (тип вмісту, час останнього поновлення документа, кодування і т.п.).

Основні змінні оточення, достатні для створення сценаріїв:

```
{
'DOCUMENT_ROOT': '/var/www/html',
'SERVER_ADDR': '127.0.0.1',
'SERVER_PORT': '80',
```

```

'GATEWAY_INTERFACE': 'CGI/1.1',
'HTTP_ACCEPT_LANGUAGE': 'en-us,en;q=0.50',
'REMOTE_ADDR': '127.0.0.1',
'SERVER_NAME': 'rnd.onego.ru',
'HTTP_CONNECTION': 'close',
'HTTP_USER_AGENT': 'Mozilla/5.0 (X11; U; Linux i586; en-US;
rv:1.0.1) Gecko/20121003',
'HTTP_ACCEPT_CHARSET': 'ISO-8859-1, utf-8;q=0.66, *;q=0.66',
'HTTP_ACCEPT': 'text/xml,application/xml,application/xhtml+xml,
text/html;q=0.9,text/plain;q=0.8,video/x-mng,image/png,image/jpeg,
image/gif;q=0.2,text/css,*/*;q=0.1',
'REQUEST_URI': '/cgi-bin/test.py?a=1',
'PATH': '/sbin:/usr/sbin:/bin:/usr/bin:/usr/X11R6/bin',
'QUERY_STRING': 'a=1&b=2',
'SCRIPT_FILENAME': '/var/www/cgi-bin/test.py',
'HTTP_KEEP_ALIVE': '300',
'HTTP_HOST': 'localhost',
'REQUEST_METHOD': 'GET',
'SERVER_SIGNATURE': 'Apache/1.3.23 Server at rnd.onego.ru Port 80',
'SCRIPT_NAME': '/cgi-bin/test.py',
'SERVER_ADMIN': 'root@localhost',
'SERVER_SOFTWARE': 'Apache/1.3.23 (Unix) (Red-Hat/Linux)
mod_python/2.7.8 Python/1.5.2 PHP/4.1.2',
'SERVER_PROTOCOL': 'HTTP/1.0',
'REMOTE_PORT': '39251'
}

```

Наступний CGI-сценарій видає чорний квадрат (в ньому використовується модуль Image для обробки зображень):

```

#!/usr/bin/python
import sys
print «»Content-Type: image/jpeg
«»
import Image
i = Image.new(«RGB», (10,10))
i.im.draw_rectangle((0,0,10,10), 1)
i.save(sys.stdout, «jpeg»)

```

В Python є підтримка CGI у вигляді модуля cgi. Наступний приклад показує деякі з його можливостей:

```

#!/usr/bin/python
# -*- coding: cp1251 -*-
import cgi, os

# аналіз запиту
f = cgi.FieldStorage()

```

```

if f.has_key(«a»):
    a = f[«a»].value
else:
    a = «0»

# обробка запиту
b = str(int(a)+1)
mytext = open(os.environ[«SCRIPT_FILENAME»]).read()
mytext_html = cgi.escape(mytext)

# формування відповіді
print «»Content-Type: text/html

<html><head><title>Вирішення прикладу: %(b)s = %(a)s + 1</title></head>
<body>
%(b)s
<table width=»80%»><tr><td>
<form action=»me.cgi» method=»GET»>
<input type=»text» name=»a» value=»0» size=»6»>
<input type=»submit» name=»b» value=»Опрацювати»>
</form></td></tr></table>
<pre>
%(mytext_html)s
</pre>
</body></html>»»»» % vars()

```

У цьому прикладі до заданого у формі числа додається 1. Крім того, виводиться вихідний код самого сценарію. Слід зауважити, що для екранування символів >, <, & використана функція cgi.escape (). Для формування Web-сторінки застосована операція форматування. Як словник для виконання підстановок використаний словник vars () з усіма локальними змінними. Знаки відсотка довелося подвоїти, щоб вони не інтерпретувалися командою форматування. Варто звернути увагу на те, як отримано значення від користувача. Об'єкт FieldStorage «майже» словник, з тим винятком, що для отримання звичайного значення потрібно додатково подивитися атрибут value. Справа в тому, що в сценарій можуть передаватися не тільки текстові значення, але і файли, а також множинні значення з одним і тим же ім'ям.

Зауваження. При обробці вхідних значень CGI-сценаріїв потрібно уважно і скрупульозно перевіряти допустимі значення. Краще вважати, що клієнт може передати на вхід все, що завгодно. З цього все необхідно вибрати і перевірити тільки те, що чекає сценарій.

Наприклад, не слід підставляти отримані від користувача дані в шлях до файлу, в якості аргументів до функції eval () і їй подібних параметрів командного рядка; частин в SQL-запитах до бази даних. Також не варто вставляти отримані дані безпосередньо у сформовані сторінки, якщо ці сторінки буде бачити не тільки клієнт, який замовив URL (наприклад, така ситуація звичайна у web-чатах, форумах, гостьових книгах), і навіть в тому випадку, якщо єдиний читач цієї інформації – адміністратор сайту. Той, хто дивиться сторінки з неперевіраним HTML-кодом, що надійшли без-

посередньо від користувача, ризикують обробити в своєму браузері шкідливий код, який використовує пролом в його захисті.

Навіть якщо CGI-сценарій використовується виключно іншими сценаріями через запит на URL, потрібно перевіряти вхідні значення настільки ж ретельно, як би дані вводив користувач. (Так як недоброзичливець може подати на web-сервер будь-які значення). У прикладі вище перевірка на допустимість проведена при виконанні функції `int()`: якщо було б задано нечислові значення, сценарій аварійно завершився, а користувач побачив Internal Server Error. Після аналізу вхідних даних можна виділити фазу їх обробки. У цій частині CGI-сценарію обчислюються змінні для подальшого виведення. Тут необхідно враховувати не тільки значення переданих змінних, а й факт їх присутності або відсутності, так як це теж може впливати на логіку сценарію.

І, нарешті, фаза виведення готового об'єкту (тексту, HTML-документа, зображення, мультимедіа-об'єкта і т.п.). Простіше всього заздалегідь підготувати шаблон сторінки (або її великих частин), а потім просто заповнити вмістом зі змінних.

У наведених прикладах імена з'являлися в рядку запиту тільки один раз. Деякі форми породжують кілька значень для одного імені. Отримати всі значення можна за допомогою методу `getlist()`:

```
lst = form.getlist («fld»)
```

Список `lst` буде містити стільки значень, скільки полів з ім'ям `fld` отримано з web-форми (він може бути і порожнім, якщо жодне поле з заданим ім'ям не було заповнено).

У деяких випадках необхідно передати на сервер файли (зробити upload). Наступний приклад та коментар до нього допоможуть розібратися з цим завданням:

```
#!/usr/bin/env python
import cgi
form = cgi.FieldStorage()
file_contents = «»
if form.has_key («filename»):
    fileitem = form[«filename»]
    if fileitem.file:
        file_contents = «»«<P>
        <PRE>%s</PRE>»»» % fileitem.file.read()

print «»»Content-Type: text/html

<HTML><HEAD><TITLE>Завантаження файлу</TITLE></HEAD>
<BODY><H1> Завантаження файлу </H1>
<P><FORM ENCTYPE=»multipart/form-data»
ACTION=»getfile.cgi» METHOD=»POST»>
<br>Файл: <INPUT TYPE=»file» NAME=»filename»>
<br><INPUT TYPE=»submit» NAME=»button» VALUE=»Передача файлу»>
</FORM>
%s
</BODY></HTML>»»» % file_contents
```

На початку слід розглянути web-форму, яка приведена в кінці сценарію: саме вона буде виводитися користувачеві при зверненні з CGI-сценарієм. Форма має поле типу `file`, яке у web-браузері представляється рядком вводу і кнопкою «Browse». Натискаючи на кнопку «Browse», користувач вибирає файл, доступний в операційній системі на його комп'ютері. Після цього він може натиснути кнопку «Передати файл» для передачі файлу на сервер.

Для налагодження CGI-сценарію можна використовувати модуль `cgitb`. При виникненні помилки цей модуль видасть барвисту HTML-сторінку із зазначенням місця порушення винятку. На початку відлаштовуваного сценарію потрібно поставити

```
import cgitb
```

```
cgitb.enable (1)
```

Або, якщо не потрібно показувати помилки в браузері:

```
import cgitb
```

```
cgitb.enable (0, logdir = «/ tmp»)
```

Тільки необхідно пам'ятати, що слід прибрати ці рядки, коли сценарій буде налагоджений, так як він видає шматочки коду сценарію. Це може бути використано зловмисниками, з тим щоб знайти уразливості в CGI-сценарії або підглянути паролі (якщо такі присутні в сценарії).

Для створення Web-додатків застосовуються і більш складні засоби, ніж web-сервер з розташованими на ньому статичними документами і CGI-сценаріями. В залежності від призначення такі програмні системи називаються серверами web-додатків, системами управління вмістом (CMS, Content Management System), системи web-публікації та засобами для створення WWW-порталів. Причому CMS-система може бути виконана як web-додаток, а засоби для створення порталів можуть базуватися на системах web-публікації, для яких CMS-система є підсистемою. Тому, вибираючи систему для конкретних потреб, варто уточнити, які функції вона повинна виконувати.

Мова Python, хоча і поступається PHP за кількістю створених на ньому web-систем, має декілька досить популярних додатків. Самим яскравим прикладом засобу для створення сервера web-додатків є Zope (вимовляється «Зоп») (див. <http://zope.org>) (Z Object Publishing Environment, середовище публікації об'єктів). Zope має вбудований web-сервер, але може працювати і з іншими Web-серверами, наприклад, Apache. На основі Zope можна будувати web-портали, наприклад, за допомогою Plone/Zope, але можна розробляти і власні web-додатки. При цьому Zope дозволяє розділити Форму, Зміст і Логіку до такої міри, що Змістом можуть займатися одні люди (менеджери по вмісту), Формою – інші (web-дизайнери), а Логікою – треті (програмісти). У випадку з Zope Логіку можна задати за допомогою мови Python (або, як варіант, Perl), Форма може бути створена в графічних або спеціалізованих web-редакторах, а робота з вмістом організована через Web-форми самого Zope.

В рамках цієї теми неможливо детально розглянути такий інструмент як Zope, тому варто лише зауважити, що він досить цікавий не тільки як середовище розробки web-додатків, але і з точки зору підходів. Наприклад, унікальна об'єктно-орієнтована модель Zope дозволяє досить гнучко описувати програми.

Zope містить у собі такі можливості:

- Web-сервер. Zope може працювати з Web-серверами через CGI або використовувати свій вбудований Web-сервер (ZServer);
- середовище розробника виконується як Web-додаток. Zope дозволяє створювати Web-додатки через Web-інтерфейс;
- підтримка сценаріїв. Zope підтримує кілька мов сценаріїв: Python, Perl і власний DTML (Document Template Markup Language – мова розмітки шаблону документа);
- база даних об'єктів. Zope використовує в своїй роботі стійкі об'єкти, що зберігаються в спеціальній базі даних (ZODB). Є досить простий інтерфейс для управління цією базою даних;
- інтеграція з реляційними базами даних. Zope може зберігати свої об'єкти та інші дані в реляційних СУБД: Oracle, PostgreSQL, MySQL, Sybase і т.п.

Розробники Zope виходили з того, що в основі WWW об'єктна модель, в якій завантаження документа по URI, можна порівняти з відправкою повідомлення об'єкту. Об'єкти Zope розкладені по папках (folders), до яких прив'язані політики доступу для користувачів, які мають певні ролі. В якості об'єктів можуть виступати документи, зображення, мультимедіа-файли, адаптери до баз даних і т.п.

Документи Zope можна писати на мові DTML – додаток HTML з синтаксисом для включення значень подібне SSI (Server-Side Include). Наприклад, для вставки змінної з назвою документа можна використовувати

```
<! - # Var document_title ->
```

Слід зауважити, що об'єкти Zope можуть мати свої атрибути, а також методи, зокрема, написані на мові Python. Змінні ж можуть з'являтися як із заданих користувачем значень, так і з інших джерел даних (наприклад, з бази даних за допомогою виконання вибірки функцією SELECT).

Зараз для опису документа Zope все частіше застосовується ZPT (Zope Page Templates, шаблони сторінок Zope), які в свою чергу використовують TAL (Template Attribute Language, мова шаблонних атрибутів). Він дозволяє замінювати, повторювати або пропускати елементи документа описуваного шаблоном документа. «Операторами» мови TAL є XML-атрибути з простору імен TAL. Простір імен сьогодні описується таким ідентифікатором:

```
xmlns: tal = «http://xml.zope.org/namespaces/tal»
```

Оператор TAL має ім'я і значення (що виражається ім'ям і значенням атрибута). У середині значення зазвичай записано TAL-вираз, синтаксис якого описується іншою мовою – TALES (Template Attribute Language Expression Syntax, синтаксис виразів TAL).

Таким чином, ZPT наповнює вмістом шаблони, інтерпретуючи атрибути TAL. Наприклад, щоб Zope підставив назву документа (тег TITLE), шаблон може мати наступний код:

```
<title tal:content=»here/title»> Doc Title </ title>
```

Варто зауважити, що наведений код зійде за код на HTML, тобто, Web-дизайнер може на будь-якому етапі роботи над проектом редагувати шаблон в HTML-редакторі (за умови, що той зберігає незнайомі атрибути з простору імен tal). У цьому прикладі

here / title є виразом TALES. Текст Doc Title служить орієнтиром для web-дизайнера і замінюється значенням вираз here / title, тобто, буде взято властивість title документа Zope.

Примітка:

В Zope об'єкти мають властивості. Набір властивостей залежить від типу об'єкта, але може бути розширений в індивідуальному порядку. Властивість id присутня завжди, властивість title зазвичай теж вказується.

У якості більш складного прикладу можна розглянути організацію повтору в середині шаблону (для випробування цього прикладу в Zope потрібно додати об'єкт Page Template):

```
ul>
  <li tal:define=»s modules/string»
    tal:repeat=»el python:s.digits»>
    <a href=»DUMMY»
      tal:attributes=»href string:/digit/$el»
      tal:content=»el»>SELECTION</a>
    </li>
</ul>
```

Цей шаблон створює наступний результат:

```
<ul>
  <li><a href=»/digit/0»>0</a></li>
  <li><a href=»/digit/1»>1</a></li>
  <li><a href=»/digit/2»>2</a></li>
  <li><a href=»/digit/3»>3</a></li>
  <li><a href=»/digit/4»>4</a></li>
  <li><a href=»/digit/5»>5</a></li>
  <li><a href=»/digit/6»>6</a></li>
  <li><a href=»/digit/7»>7</a></li>
  <li><a href=»/digit/8»>8</a></li>
  <li><a href=»/digit/9»>9</a></li>
</ul>
```

Тут потрібно звернути увагу на два основні моменти:

- в шаблоні можна використовувати вирази Python (в даному прикладі змінна s визначена як модуль Python) і змінну-лічильник циклу el, яка проходить ітерації по рядку string.digits;
- за допомогою TAL можна задавати не тільки вміст елемента, а й атрибуту тега (в даному прикладі використовувався атрибут href).

Деталі можна дізнатися по документації. Варто лише зауважити, що ітерація може відбуватися з найрізноманітніших джерел даних: вмісту поточної папки, вибірці з бази даних або, як у наведеному прикладі, по об'єкту Python.

Будь-який програміст знає, що програмування є тим ефективніше, чим краще вдалося «розставити дужки», вивівши «загальний множник за дужки». Іншими словами, хороші програмісти мають бути достатньо «ліниві», щоб знайти оптимальну деком-

позицію розв'язуваної задачі. При проектуванні динамічного web-сайту Zope дозволяє розмістити «множники» і «дужки» так, щоб досягти максимального повторного використання коду (як розмітки, так і сценаріїв). Допомогає цьому унікальний підхід до побудови взаємин між об'єктами: запозичення (acquisition).

Нехай деякий об'єкт (документ, зображення, сценарій, підключення до бази даних тощо) розташований в папці Example. Тепер об'єкти цієї папки доступні до будь-яких внутрішніх папок. Навіть політики безпеки запозичуються більш глибоко вкладеними папками від папок, які ближче до кореня. Запозичення є дуже важливою концепцією Zope, без розуміння якої Zope складно грамотно застосовувати, і навпаки, її розуміння дозволяє економити сили і час, повторно використовуючи об'єкти в розробці.

Найцікавіше, що запозичувати об'єкти можна також з паралельних папок. Нехай, наприклад, поруч з папкою Example знаходиться папка Zigzag, в якій лежить потрібний об'єкт (його найменування note). При цьому в папці Example програміста цікавить об'єкт index\_html, в середині якого викликається note. Звичайний шлях до об'єкта index\_html відбуватиметься по URI за адресою `http://zopeserver/Example/`. А от, якщо потрібно використовувати note з Zigzag (і в папці Example його немає), то URI буде: `http://zopeserver/Zigzag/Example/`. Таким чином, вказівка шляху в Zope відрізняється від традиційного шляху, скажімо, в Unix: у шляху можуть бути присутніми «зигзаги» через паралельні папки, що дають можливість запозичувати об'єкти з цих папок. Таким чином, можна зробити конкретну сторінку, комбінуючи один або кілька незалежних аспектів.

### 1.4.3.6. Мережні додатки на Python

Застосовувана в IP-мережах архітектура клієнт-сервер використовує IP-пакети для комунікації між клієнтом і сервером. Клієнт відправляє запит серверу, на який той відповідає. У випадку з TCP/IP між клієнтом і сервером встановлюється з'єднання (зазвичай з двосторонньою передачею даних), а у випадку з UDP/IP – клієнт і сервер обмінюються пакетами (дейтаграмами) з негарантованою доставкою.

Кожен мережевий інтерфейс IP-мережі має унікальну у цій мережі адресу (IP-адресу). Спрощено можна вважати, що кожен комп'ютер в мережі Інтернет має власну IP-адресу. При цьому в рамках одного мережевого інтерфейсу може бути кілька мережевих портів. Для встановлення з'єднання з мережею додатку клієнта треба вибрати вільний порт і встановити з'єднання з серверним додатком, що слухає (listen) порт з певним номером на віддаленому мережевому інтерфейсі. Пара IP-адреси та порт характеризують сокет (гніздо) – початкову (кінцеву) точку мережевої комунікації. Для створення з'єднання TCP/IP необхідно два сокета: один на локальній машині, а інший – на віддаленій. Таким чином, кожне мережеве з'єднання має IP-адресу і порт на локальній машині, а також IP-адресу і порт на віддаленій машині.

Модуль socket забезпечує можливість працювати з сокетами з Python. Сокети використовують транспортний рівень згідно семирівневої моделі OSI (Open Systems Interconnection, взаємодія відкритих систем), тобто відносяться до більш низького рівня, ніж більшість описаних у цьому розділі протоколів.

Рівні моделі OSI:

- фізичний. Потік бітів, що передаються з фізичної лінії. Визначає параметри фізичної лінії;
- канальний (Ethernet, PPP, ATM і т.п.) Кодує і декодує дані у вигляді потоку бітів, справляючись з помилками, що виникають на фізичному рівні в межах фізично єдиної мережі;
- мережевий (IP). Маршрутизує інформаційні пакети від вузла до вузла;
- транспортний (TCP, UDP і т.п.). Забезпечує прозору передачу даних між двома точками з'єднання;
- сеансовий. Управляє сеансом з'єднання між учасниками мережі. Починає, координує і завершує з'єднання;
- подання. Забезпечує незалежність даних від форми їх подання шляхом перетворення форматів. На цьому рівні може виконуватися прозоре (з точки зору вищого рівня) шифрування і дешифрування даних;
- додатків (HTTP, FTP, SMTP, NNTP, POP3, IMAP і т.д.). Підтримує конкретні мережні додатки. Протокол залежить від типу сервісу.

Кожен сокет відноситься до одного з комунікаційних доменів. Модуль socket підтримує домени UNIX та Internet. Кожен домен має на увазі своє сімейство протоколів і адресацію. Даний виклад буде торкатися лише доменів Internet, а саме протоколи TCP/IP і UDP/IP, тому для вказівки комунікаційного домену при створенні сокета буде вказуватися константа `socket.AF_INET`.

Як приклад варто розглянути найпростішу клієнт-серверну пару. Сервер буде приймати рядок і відповідати клієнтові. Мережевий пристрій іноді називають хостом (host), тому буде вживатися цей термін по відношенню до комп'ютера, на якому працює мережевий додаток.

Сервер:

```
import socket, string
def do_something(x):
    lst = map(None, x);
    lst.reverse();
    return string.join(lst, «»)

HOST = «»          # localhost
PORT = 33333
srv = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
srv.bind((HOST, PORT))
while 1:
    print «Слухаю порт 33333»
    srv.listen(1)
    sock, addr = srv.accept()
    while 1:
        pal = sock.recv(1024)
        if not pal:
            break
```



```
print «Отримано від %s:%s:» % addr, pal
lap = do_something(pal)
print «Відправлено %s:%s:» % addr, lap
sock.send(lap)
sock.close()
```

Клієнт:

```
import socket
```

```
HOST = «»          # (localhost)
PORT = 33333        # порт на віддаленому комп'ютері
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect((HOST, PORT))
sock.send(«ПАЛНДРОМ»)
result = sock.recv(1024)
sock.close()
print «Отримано:», result
```

*Примітка:*

У прикладі використані українські літери: необхідно вказувати кодування.

Перш за все, потрібно запустити сервер. Сервер відкриває сокет на локальній машині на порту 33333, і адресу 127.0.0.1. Після цього він слухає (listen ()) порт. Коли на порту з'являються дані, приймається (accept ()) і входить з'єднання. Метод accept () повертає пару – Socket-об'єкт та адресу віддаленого комп'ютера, який встановлює з'єднання (пара – IP-адреса, порт на віддаленій машині). Після цього можна застосовувати методи recv () і send () для спілкування з клієнтом. В recv () задається число байтів в черговій порції. Від клієнта може прийти і менша кількість даних.

Код програми-клієнта досить очевидний. Метод connect () встановлює з'єднання з віддаленим хостом (в наведеному прикладі він розташований на тій же машині). Дані передаються методом send () і приймаються методом recv () – аналогічно тому, що відбувається на сервері.

Модуль socket має декілька допоміжних функцій. Зокрема, функції для роботи з системою доменних імен (DNS):

```
>>> Import socket
>>> Socket.gethostbyname ('www.onego.ru')
('Www.onego.ru', [], ['195.161.136.4'])
>>> Socket.gethostbyaddr ('195.161.136.4')
('Www.onego.ru', [], ['195.161.136.4'])
>>> Socket.gethostbyname ()
'Rnd.onego.ru'
```

У нових версіях Python з'явилася така функція як socket.getservbyname (). Вона дозволяє перетворювати найменування Інтернет-сервісів в загальноприйняті номери портів:

```
>>> For srv in 'http', 'ftp', 'imap', 'pop3', 'smtp':
... print socket.getservbyname (srv, 'tcp'), srv
...
```

```
80 http
21 ftp
143 imap
110 pop3
25 smtp
```

Модуль також містить велику кількість констант для вказівки протоколів, типів сокетів, комунікаційних доменів і т.п. Інші функції модуля socket можна при необхідності вивчити по документації.

Повідомлення електронної пошти в Інтернеті передаються від клієнта до сервера і між серверами в основному по протоколу SMTP (Simple Mail Transfer Protocol, простий протокол передачі пошти). Протокол SMTP і ESMTP (розширений варіант SMTP) описані в RFC 821 і RFC 1869. Для роботи з SMTP в стандартній бібліотеці модулів є модуль smtplib. Для того щоб почати SMTP-з'єднання з сервером електронної пошти, необхідно спочатку створити об'єкт для управління SMTP-сесії за допомогою конструктора класу SMTP:

```
smtplib.SMTP ([host [, port]])
```

Параметри host і port задають адресу і порт SMTP-серверу, через який буде відправлятися пошта. За замовчуванням, port = 25. Якщо host заданий, конструктор сам встановить з'єднання, інакше доведеться окремо викликати метод connect (). Екземпляри класу SMTP мають методи для всіх поширених команд SMTP-протоколу, але для відправки пошти досить виклику конструктор і методи sendmail () і quit ():

```
# - * - Coding: cp1251 - * -
```

```
from smtplib import SMTP
fromaddr = «student@mail.ru» # Від кого
toaddr = «rnd@onego.ru» # Кому
message = «» «From: Student <% (fromaddr) s>
To: Lecturer <% (toaddr) s>
Subject: From Python course student
MIME-Version: 1.0
Content-Type: text / plain; charset = Windows-1251
Content-Transfer-Encoding: 8bit
Доброго дня! Я вивчаю курс Веб-технології та Веб-дизайн і
відправляю лист його автору.
«» «
connect = SMTP ('mail.onego.ru')
connect.set_debuglevel (1)
connect.sendmail (fromaddr, toaddr, message% vars ())
connect.quit ()
```

Слід зауважити, що toaddr в повідомленні (в поле To) і при відправленні можуть не збігатися. Справа в тому, що одержувач і відправник у ході SMTP-сесії передається командами SMTP-протоколу. При запуску зазначеного вище прикладу на екрані з'явиться відлагоджувач інформація (адже рівень налагодження заданий рівнем 1):

```
send: 'ehlo rnd.onego.ru\r\n'
```

```
reply: '250-mail.onego.ru Hello as3-042.dialup.onego.ru [195.161.147.4], pleased
to meet you\r\n'
```

```
send: 'mail FROM:<student@mail.ru> size=270\r\n'
```

```
reply: '250 2.1.0 <student@mail.ru>... Sender ok\r\n'
```

```
send: 'rcpt TO:<rnd@onego.ru>\r\n'
```

```
reply: '250 2.1.5 <rnd@onego.ru>... Recipient ok\r\n'
```

```
send: 'data\r\n'
```

```
reply: '354 Enter mail, end with «.» on a line by itself\r\n'
```

```
send: 'From: Student <student@mail.ru>\r\n ... '
```

```
reply: '250 2.0.0 iBPFgQ7q028433 Message accepted for delivery\r\n'
```

```
send: 'quit\r\n'
```

```
reply: '221 2.0.0 mail.onego.ru closing connection\r\n'
```

З цієї (дещо скороченої) налагоджувальної інформації можна побачити, що клієнт відправляє (send) команди SMTP-серверу (EHLO, MAIL FROM, RCPT TO, DATA, QUIT), а той виконує команди і відповідає (reply), повертаючи код повернення.

В ході однієї SMTP-сесії можна відправити відразу кілька листів підряд, якщо не викликати quit ().

В принципі, команди SMTP можна подавати і окремо: для цього в об'єкті-з'єднанні є методи (helo (), ehlo (), expn (), help (), mail (), rcpt (), vrfy (), send (), noop (), data ()), відповідні однойменним командам SMTP-протоколу.

Можна задати й довільну команду SMTP-серверу за допомогою методу docmd (). У наступному прикладі показаний простий сценарій, який можуть використовувати ті, хто час від часу приймає пошту на свій сервер по протоколу SMTP від поштового сервера, на якому зберігається черга повідомлень для деякого домену:

```
from smtplib import SMTP
connect = SMTP ('mx.abcde.ua')
connect.set_debuglevel (1)
connect.docmd («ETRN rnd.abcde.ua»)
connect.quit ()
```

Цей простенький сценарій пропонує серверу mx.abcde.ua спробувати зв'язатися з основним поштовим сервером домену rnd.abcde.ua і переслати всю накопичену для нього пошту.

При роботі з класом smtplib.SMTP можуть порушуватися різні винятки. Призначення деяких з них наведено нижче:

**smtplib.SMTPException**

Базовий клас для всіх винятків модуля.

**smtplib.SMTPServerDisconnected**

Сервер несподівано перервав зв'язок (або зв'язок з сервером не було встановлено).

**smtplib.SMTPResponseException**

Базовий клас для всіх винятків, які мають код відповіді SMTP-сервера.

**smtplib.SMTPSenderRefused**

Відправник відкинутий.

**smtplib.SMTPRecipientsRefused**

Всі одержувачі відкинуті сервером.

**smtplib.SMTPDataError**

Сервер відповів невідомим кодом на дані повідомлення.

**smtplib.SMTPConnectError**

Помилка встановлення з'єднання.

**smtplib.SMTPHeloError**

Сервер не відповів правильно на команду HELO або відкинув її.

Ще один протокол – POP3 (Post Office Protocol, поштовий протокол) – служить для прийому пошти з поштового ящика на сервері (протокол визначений в RFC 1725).

Для роботи з поштовим сервером потрібно встановити з ним з'єднання і, подібно розглянутому вище прикладу, за допомогою SMTP-команд отримати необхідні повідомлення. Об'єкт-з'єднання POP3 можна встановити за допомогою конструктора класу POP3 з модуля poplib:

**poplib.POP3 (host [, port])**

Де host – адреса POP3-сервера, port – порт на сервері (за замовчуванням 110), pop\_obj – об'єкт для управління сеансом роботи з POP3-сервером.

Наступний приклад демонструє основні методи для роботи з POP3-з'єднанням:

**import poplib, email**

**# Облікові дані користувача:**

**SERVER = «pop.server.com»**

**USERNAME = «user»**

**USERPASSWORD = «secretword»**

**p = poplib.POP3 (SERVER)**

**print p.getwelcome ()**

**# Етап ідентифікації**

**print p.user (USERNAME)**

**print p.pass\_ (USERPASSWORD)**

**# Етап транзакцій**

**response, lst, octets = p.list ()**

**print response**

**for msgnum, msgsize in [i.split () for i in lst]:**

**print «Повідомлення% (msgnum) s має довжину% (msgsize) s»% vars ()**

**print «UIDL =>, p.uidl (int (msgnum)). split () [2]**

**if int (msgsize)> 32010:**

**(Resp, lines, octets) = p.top (msgnum, 0)**

**else:**

**(Resp, lines, octets) = p.retr (msgnum)**

**msgtxt = «\n». join (lines) + «\n\n»**

**msg = email.message\_from\_string (msgtxt)**

**print «\* Від:% (from) s \n \* Кому:% (to) s \n \* Тема:% (subject) s \n»% msg**

# Msg містить заголовки повідомлення або всі повідомлення (якщо воно не-велике)

# Етап оновлення

print p.quit ()

Примітка: Щоб приклад спрацював коректно, необхідно внести реальні облікові дані.

При виконанні сценарій виведе на екран приблизно наступне:

+ OK POP3 pop.server.com server ready

+ OK User name accepted, password please

+ OK Mailbox open, 68 messages

+ OK Mailbox scan listing follows

Повідомлення 1 має довжину 4202

UIDL = 4152a47e00000004

\* Від: online@kaspersky.com

\* Кому: user@server.com

\* Тема: KL Online Activation

...

+ OK Sayonara

Робота з POP3-сервером складається з трьох фаз: ідентифікації, транзакцій та оновлення. На етапі ідентифікації відразу після створення POP3-об'єкта дозволені тільки команди USER, PASS (іноді APOP і RPOP). Після ідентифікації сервер отримує інформацію про користувача і настає етап транзакцій. Тут доречні інші команди. Етап поновлення викликається командою QUIT, після якої POP3-сервер оновлює поштову скриньку користувача відповідно до поданих командами, а саме – видаляє помічені для видалення повідомлення.

Стандартні засоби мови Python дозволяють отримувати з програми доступ до об'єктів WWW як в простих випадках, так і при складних обставинах, зокрема при необхідності передавати дані форми, ідентифікації, доступу через проксі і т.п.

Варто зазначити, що при роботі з WWW використовується в основному протокол HTTP, однак WWW охоплює не тільки HTTP, а й багато інших схем (FTP, gopher, HTTPS і т.п.). Використовувана схема зазвичай вказана на самому початку URL.

Простий випадок отримання WWW-об'єкта за відомим URL показаний в наступному прикладі:

```
import urllib
doc = urllib.urlopen («http://python.onego.ru»). read ()
print doc [: 40]
```

Функція urllib.urlopen () створює файлоподібний об'єкт, який читає методом read (). Інші методи цього об'єкту: readline (), readlines (), fileno (), close () працюють як і у звичайного файлу, а також є метод info (), який повертає відповідний отриманому з сервера Message-об'єкт. Цей об'єкт можна використовувати для отримання додаткової інформації:

```
>>> Import urllib
>>> F = urllib.urlopen («http://python.onego.ru»)
```

```
>>> Print f.info ()
```

```
Date: Sat, 25 Dec 2004 19:46:11 GMT
```

```
Server: Apache/1.3.29 (Unix) PHP/4.3.10
```

```
Content-Type: text / html; charset = windows-1251
```

```
Content-Length: 4291
```

```
>>> Print f.info () ['Content-Type']
```

```
text / html; charset = windows-1251
```

За допомогою функції urllib.urlopen () можна робити і більш складні речі, наприклад, передавати web-серверу дані форми. Як відомо, дані заповненої web-форми можуть бути передані на web-сервер з використанням методу GET або методу POST. Метод GET пов'язаний з кодуванням всіх переданих параметрів після знака «?» в URL, а при методі POST дані передаються в тілі HTTP-запиту. Обидва варіанти передачі представлені нижче:

```
import urllib
data = {«search»: «Python»}
enc_data = urllib.urlencode(data)
# метод GET
f = urllib.urlopen («http://searchengine.com/search» + «?» + enc_data)
print f.read()
# метод POST
f = urllib.urlopen («http://searchengine.com/search», enc_data)
```

В деяких випадках дані мають повторювані імена. В цьому випадку в якості параметра urllib.urlencode () можна використовувати замість словника послідовність пар ім'я-значення:

```
>>> Import urllib
>>> Data = [(«n», «1»), («n», «3»), («n», «4»), («button», «Привіт»),]
>>> Enc_data = urllib.urlencode (data)
>>> Print enc_data
n = 1 & n = 3 & n = 4 & button =% F0% D2% C9% D7% C5% D4
```

Модуль urllib дозволяє завантажувати web-об'єкти через проксі-сервер. Якщо нічого не вказувати, буде використовуватися проксі-сервер, який був заданий прийнятим в конкретній ОС способом. В Unix проксі-сервери задаються в змінних оточення http\_proxu, ftp\_proxu і т.п., у Windows проксі-сервери записані в реєстрі, а в Mac OS вони беруться з конфігурації Internet. Поставити проксі-сервер можна і як іменований параметр proxies до urllib.urlopen ():

```
# Використовувати вказаний проксі
proxies = {«http»: «http://www.proxy.com:3128»}
f = urllib.urlopen (some_url, proxies = proxies)
# Не використовувати проксі
f = urllib.urlopen (some_url, proxies = {})
# Використовувати проксі за замовчуванням
f = urllib.urlopen (some_url, proxies = None)
f = urllib.urlopen (some_url)
```

Функція `urlretrieve()` дозволяє записати заданий URL мережевий об'єкт у файл. Вона має такі параметри:

`urllib.urlretrieve(url [, filename [, reporthook [, data]]])`

Тут `url` – URL мережного об'єкту, `filename` – ім'я локального файлу для приміщення об'єкта, `reporthook` – функція, яка буде викликатися для повідомлення про стан завантаження, `data` – дані для методу POST (якщо він використовується). Функція повертає кортеж (`filepath, headers`), де `filepath` – ім'я локального файлу, в який закачаний об'єкт, `headers` – результат методу `info()` для об'єкта, повернутого `urlopen()`.

Для забезпечення інтерактивності функція `urllib.urlretrieve()` викликає час від часу функцію, задану в `reporthook()`. Цій функції передаються три аргументи: кількість прийнятих блоків, розмір блоку і загальний розмір об'єкта, що приймається в байтах (якщо він невідомий, цей параметр дорівнює `-1`).

У наступному прикладі програма приймає великий файл і, щоб користувач не нудьгував, пише відсоток від виконаного завантаження і передбачуваний час, що залишився:

`FILE = 'boost-1.31.0-9.src.rpm'`

`URL = 'http://download.fedora.redhat.com/pub/fedora/linux/core/3/SRPMS/' + FILE`

```
def download(url, file):
    import urllib, time
    start_t = time.time()
    def progress(bl, blsize, size):
        dldsize = min(bl*blsize, size)
        if size != -1:
            p = float(dldsize) / size
            try:
                elapsed = time.time() - start_t
                est_t = elapsed / p - elapsed
            except:
                est_t = 0
            print «%6.2f%% %6.0f s %6.0f s %6i / %-6i bytes» % (
                p*100, elapsed, est_t, dldsize, size)
        else:
            print «%6i / %-6i bytes» % (dldsize, size)
    urllib.urlretrieve(URL, FILE, progress)
    download(URL, FILE)
```

Ця програма виведе приблизно таке (відсоток від повного обсягу закачування; кількість пройдених секунд; передбачуваний час, що залишився; закачані байти; повна кількість байтів):

```
0.00 % 1 s 0 s 0 / 6952309 bytes
0.12 % 5 s 3941 s 8192 / 6952309 bytes
0.24 % 7 s 3132 s 16384 / 6952309 bytes
0.35 % 10 s 2864 s 24576 / 6952309 bytes
0.47 % 12 s 2631 s 32768 / 6952309 bytes
```

```
0.59 % 15 s 2570 s 40960 / 6952309 bytes
0.71 % 18 s 2526 s 49152 / 6952309 bytes
0.82 % 20 s 2441 s 57344 / 6952309 bytes ...
```

Згідно з документом RFC 2396 URL повинен будуватися за таким шаблоном:  
**scheme :// netloc / path; parameters? query # fragment**

де

**scheme**

Адресна схема. Наприклад: `http, ftp, gopher.`

**netloc**

Місцезнаходження в мережі.

**path**

Шлях до ресурсу.

**params**

Параметри.

**query**

Рядок запиту.

**fragment**

Одна з функцій уже використовувалася для формування URL – `urllib.urlencode()`. Окрім неї в модулі `urllib` є й інші функції:

**quote(s, safe = '/')**

Функція екранує символи в URL, щоб їх можна було відправляти на web-сервер. Вона призначена для екранування шляху до ресурсу, тому залишає `'/'` як є. Наприклад:

```
>>> Urllib.quote («rnd@onego.ru»)
'Rnd% 40onego.ru'
>>> Urllib.quote («a = b + c»)
'A% 20% 3D% 20b% 20% 2B% 20c'
>>> Urllib.quote («0/1/1»)
'0 / 1/1 '
>>> Urllib.quote («0/1/1», safe = «»)
'0% 2F1% 2F1 '
quote_plus(s, safe = '')
```

Функція екранує деякі символи в URL (у рядку запиту), щоб їх можна було відправляти на web-сервер. Вона аналогічна `quote()`, але замінює прогалини на плюси.

**unquote(s)**

Перетворення, зворотне `quote()`. Приклад:

```
>>> Urllib.unquote('a% 20% 3D% 20b% 20% 2B% 20c')
'A = b + c'
```

Перетворення, зворотне `quote_plus()`. Приклад:

```
>>> Urllib.unquote_plus('a + = + b +% 2B + c')
'A = b + c'
```

Для аналізу URL можна використовувати функції з модуля `urlparse`:

```
urlparse (url, scheme = "", allow_fragments = 1)
```

Розбирає URL в 6 компонентів (зберігаючи екранування символів):

```
scheme // netloc / path; params? Query # frag
```

```
urlsplit (url, scheme = "", allow_fragments = 1)
```

Розбирає URL в 5 компонентів (зберігаючи екранування символів):

```
scheme // netloc / path? Query # frag
```

```
urlunparse ((scheme, netloc, url, params, query, fragment))
```

Збирає URL з 6 компонентів.

```
urlunsplit ((scheme, netloc, url, query, fragment))
```

Збирає URL з 5 компонентів.

Приклад:

```
>>> From urlparse import urlsplit, urlunsplit
>>> URL = «http://google.com/search?q=Python»
>>> Print urlsplit (URL)
('Http', 'google.com', '/ search', 'q = Python',)
>>> Print urlunsplit (
... ('Http', 'google.com', '/ search', 'q = Python',))
http://google.com/search?q=Python
```

Ще одна функція того ж модуля urlparse дозволяє коректно з'єднати дві частини URL – базову і відносну:

```
>>> Import urlparse
>>> Urlparse.urljoin ('http://python.onogo.ru', 'itertools.html')
'Http://python.onogo.ru/itertools.html'
```

Функціональності модулів urllib і urlparse вистачає для більшості завдань, які вирішують сценарії на Python як web-клієнти. Тим не менш, іноді потрібно більше. На цей випадок можна використовувати модуль для роботи з протоколом HTTP – httpplib і створити власний клас для HTTP-запиту. Однак цілком імовірно, що потрібна функціональність вже є в модулі urllib2.

Одна з корисних можливостей цих модулів – доступ до web-об'єктів, що вимагає авторизації. Нижче буде розглянуто приклад, який не тільки забезпечить доступ з авторизацією, але і позначить основну ідею модуля urllib2: використання обробників (handlers), кожен з яких вирішує вузьку специфічну задачу.

Наступний приклад показує, як створити власний відкривач URL за допомогою модуля urllib2 (цей приклад взято з документації по Python):

```
import urllib2
# Підготовка ідентифікаційних даних
authinfo = urllib2.HTTPBasicAuthHandler ()
authinfo.add_password ('My page', 'localhost', 'user1', 'secret')
# Доступ через проксі
proxy_support = urllib2.ProxyHandler ({'http': 'http://localhost:8080'})
# Створення нового відкривача із зазначеними обробниками
opener = urllib2.build_opener (proxy_support,
```

```
authinfo,
urllib2.CacheFTPHandler)
# Установка поля з назвою клієнта
opener.addheaders = [('User-agent', 'Mozilla/5.0')]
# Установка нового відкривача за замовчуванням
urllib2.install_opener (opener)
# Використання відкривача
f = urllib2.urlopen ('http://localhost/mywebdir/')
print f.read () [: 100]
```

У цьому прикладі отриманий доступ до сторінки, яку охороняє mod\_python. Перший аргумент при виклику методу add\_password () задає область дії (realm) ідентифікаційних даних (він заданий директивою AuthName «My page» в конфігурації web-сервера). Інші параметри досить зрозумілі: ім'я хоста, на який потрібно отримати доступ, ім'я користувача і його пароль. Зрозуміло, для коректної роботи прикладу потрібно, щоб на локальному web-сервері був каталог, що вимагає авторизації.

У даному прикладі явно порушені всього три обробники: HTTPBasicAuthHandler, ProxyHandler і CacheFTPHandler. У модулі urllib2 їх більше десятка, призначення кожного можна дізнатися з документації до використовуваної версії Python. Є і спеціальний клас для управління відкривачами: OpenerDirector. Саме його примірник створила функція urllib2.build\_opener ().

Модуль urllib2 має і спеціальний клас для втілення запиту на відкриття URL. Називається цей клас urllib2.Request. Його примірник містить стан запиту. Наступний приклад показує, як отримати доступ до каталогу з авторизацією, використовуючи додавання заголовка в HTTP-запит:

```
import urllib2, base64
req = urllib2.Request ('http://localhost/mywebdir')
b64 = base64.encodestring ('user1: secret'). strip ()
req.add_header ('Authorization', 'Basic% s'% b64)
req.add_header ('User-agent', 'Mozilla/5.0')
f = urllib2.urlopen (req)
print f.read () [: 100]
```

Як видно з цього прикладу, нічого загадкового в авторизації немає: web-клієнт вносить (закодовані base64) ідентифікаційні дані в поле Authorization HTTP-запиту.

Примітка: наведені два приклади майже еквівалентні, тільки в другому прикладі проксі-сервер не призначений явно.

До цього часу високорівневі протоколи розглядалися з точки зору клієнта. Не менш просто створювати на Python їх серверні частини. Для ілюстрації того, як розробити програму на Python, що реалізує сервер, був обраний протокол XML-RPC. Незважаючи на свою назву, кінцевому користувачеві не обов'язково знати XML, так як він прихований від нього. Скорочення RPC (Remote Procedure Call, виклик віддаленої процедури) пояснює суть справи: за допомогою XML-RPC можна викликати процедури на віддаленому хості. Причому за допомогою XML-RPC можна абстрагуватися від конкретної мови програмування за рахунок використання загальноприйнятих



типів даних (рядки, числа, логічні значення і т.п.). У мові Python виклик віддаленої функції по синтаксису нічим не відрізняється від виклику звичайної функції:

```
import xmlrpclib
# Встановити з'єднання
req = xmlrpclib.ServerProxy («http://localhost:8000»)
try:
# Викликати віддалену функцію
print req.add (1, 3)
except xmlrpclib.Error, v:
print «ERROR»,
```

А ось як виглядає XML-RPC-сервер (для того щоб спробувати приклад вище, необхідно спочатку запустити сервер):

```
from SimpleXMLRPCServer import SimpleXMLRPCServer
srv = SimpleXMLRPCServer (('localhost', 8000)) # Запустити сервер
srv.register_function (pow) # Зареєструвати функцію
srv.register_function (lambda x, y: x + y, 'add') # І ще одну
srv.serve_forever () # Обслуговувати запити
```

За допомогою XML-RPC (а цей протокол достатньо «легкий» серед інших подібних протоколів) додатки можуть спілкуватися один з одним на зрозумілій їм мові виклику функцій з параметрами основних загальноприйнятих типів і такими ж поверненнями значень. Перевагою ж Python є зручний синтаксис виклику віддалених функцій.

#### 1.4.4. Мова Ruby

Ruby – інтерпретована мова високого рівня для швидкого і зручного об'єктно-орієнтованого програмування. Ruby володіє незалежною від операційної системи реалізацією багатопоточності, строгою динамічною типізацією, «складальником сміття» і багатьма іншими можливостями. Багато особливостей синтаксису і семантики мови Perl запозичено в Ruby.

Перша загальнодоступна версія Ruby з'явилася в 1995 р.

Ruby – повністю об'єктно-орієнтована мова:

- всі дані є об'єктами, на відміну від багатьох інших мов, де існують примітивні типи;
- кожна функція є методом;
- змінні Ruby містять не самі об'єкти, а посилання на них;
- присвоювання – це не передача значення, а копіювання посилання на об'єкт;
- у Ruby можна додавати методи не тільки в будь-які класи, але і в будь-які об'єкти. Наприклад, можна добавляти до деякого рядка довільний метод. Масиви в Ruby можуть автоматично змінювати розмір, можуть містити будь-які елементи і мова надає потужні засоби для їх обробки;
- Ruby поставляється з великою стандартною бібліотекою. Це, перш за все, бібліотеки для роботи з різними мережевими протоколами на стороні сервера і клієнта,

засоби для роботи з різними форматами представлення даних (XML, XSLT, YAML, PDF, RSS, CSV, WSDL). Також є бібліотеки для роботи з архівами, датами, кодуваннями, матрицями, засоби для системного адміністрування, розподілених обчислень, підтримка багатопоточності і так далі.

У мові Ruby також реалізований простий і зручний механізм для розширення мови за допомогою бібліотек, написаних на мові C, що дозволяє легко розробляти додаткові бібліотеки. Для уніфікованого доступу до баз даних розроблена бібліотека Ruby DBI.

До недоліків інтерпретатора Ruby можна віднести наступні:

- ⇒ *невисока швидкість роботи;*
- ⇒ *відсутність підтримки потоків операційної системи (для Unix-подібних операційних систем є підтримка процесів ОС) в експериментальній версії 1.9;*
- ⇒ *відсутність вбудованої підтримки юнікоду (можлива робота з використанням додаткових бібліотек в експериментальній версії 1.9);*
- ⇒ *відсутність компіляції в байткод. (При цьому є можливість компілювати Ruby в Java і .NET байткод, використовуючи компілятор JRuby і Ruby.NET). У експериментальну версію 2.0 входить віртуальна машина YARV, компілююча Ruby в байткод та істотно прискорююча виконання.*

Ruby – одна з наймолодших мов програмування. Своїм ім'ям вона зобов'язана дорогоцінному каменю рубіну (за аналогією з іншою широко розповсюдженою мовою програмування Perl – перли). От як описує Ruby його творець, японський програміст Юкіхіро Мацумото (Yukihiro Matsumoto): «Це потужна і динамічна об'єктно-орієнтована мова з відкритими вихідними кодами, яку я почав розробляти в 1993 році. Ruby працює на багатьох платформах, включаючи Linux і багато реалізацій Unix, MS-DOS, Windows 9x/NT/XP/7, BeOS й MacOS. Головна мета Ruby – ефективність розробки програм.

В Японії Ruby сильно потіснила такі відомі мови як Python і Perl та почала поширюватися в усьому світі. За останній час з'явилися три англomовних книги, присвячені Ruby (на жаль, поки не мають українського перекладу). У цієї мови дуже непогані шанси стати дійсно популярною, адже вона увібрала у себе переваги інших мов, врахувавши їхні недоліки. Вона входить у стандартну поставку ОС Linux (починаючи з версії 7.2), а користувачам MS Windows для першого знайомства варто порекомендувати його трохи застарілу версію, що включає, крім інтерпретатора мови й бібліотек, керівництво користувача, FAQ та безліч прикладів. Ruby є вільно розповсюджуваним продуктом, тому ви можете не турбуватися ні про його вартість, ні про обмеження в його використанні.

Ruby є одною з кращих як перша мова програмування, яку вивчатимуть студенти й школярі. Швидкий цикл розробки (редагування – запуск – редагування), використання інтерпретатора, споконвічна об'єктно-орієнтованість мови, нетипізовані змінні, які не вимагають оголошення – все це дозволяє учням сконцентрувати свою увагу на загальних принципах програмування. Надалі ми будемо орієнтуватися на роботу в ОС Linux.

Використання Ruby в інших операційних системах практично нічим не відрізняється, а результати виконання завдань не залежать від використовуваної ОС.

### 1.4.4.1. Початкові дані

Спочатку перевіримо, чи встановлений інтерпретатор Ruby у системі чи ні. У вікні shell введіть команду `ruby -v` (ключик `-v` вимагає вказівки інтерпретатором версії мови). Якщо наступне повідомлення з'явиться, то Ruby встановлений (версія, дата й платформа можуть відрізнятись):

```
ruby 1.6.4 (2012-06-04) [i386-linux-gnu]
```

Файли, що містять програми на мові Ruby мають розширення `*.rb`. По давній програмістській традиції наша перша програма буде друкувати фразу «Hello, World!». За допомогою будь-якого редактора plain-тексту (emacs, kwrite, notebook і т.п.) створимо файл `hello.rb`, у якій помістимо текст

```
puts «Hello, World!»
```

Для виконання цієї програми в командному рядку введіть `ruby hello.rb`

У результаті виконання програми в командному вікні буде надрукована необхідна фраза.

Другий спосіб виконання програм доступний користувачам не всіх операційних систем, у яких функціонує Ruby. Користувачам ОС Linux варто помістити в початок файлу з текстом програми наступний рядок:

```
#!/usr/bin/env ruby
```

Вона обов'язково повинна починатися з першої позиції. Потім потрібно змінити права доступу файлу із програмою, зробивши його таким, що виконується:

```
chmod +x hello.rb
```

Тепер для запуску програми досить увести команду `./hello.rb`

Для того, щоб зробити програму більш зрозумілою людині, що її читає, вставляються коментарі. Однорядкові коментарі починаються із символу `#` і діють до кінця рядка.

Багаторядкові коментарі вкладають у спеціальні «дужки», тоді усе, що розташовується між рядками `=begin` й `=end`, вважається коментарем. Наприклад,

```
#!/usr/bin/env ruby
=begin
Це
коментар
=end
puts «Hello, World!»
# Це теж коментар
```

Програма мовою Ruby, яку часто звуть скриптом, є послідовністю інструкцій (тверджень, пропозицій). Кожна інструкція за замовчуванням закінчується кінцем рядка.

Якщо ж за якихось причин потрібно розмістити кілька інструкцій на одному рядку, то їх потрібно розділяти символом `;` (крапка з комою). З іншого боку, іноді інструк-

ція не міститься на одному рядку. У цьому випадку символ `\` сигналізує про те, що її продовження розташовується в наступному рядку.

Приклад:

```
#!/usr/bin/env ruby
# Інструкція закінчується кінцем рядка
puts «Hello, World!»
# Кілька інструкцій в одному рядку
puts «Це тест, «; puts «який демонструє роботу Ruby.»
# Незакінчена інструкція,
# продовження якої на наступному рядку
puts «Програмування на Ruby - « +
«приємне заняття.»
# Твердження, розділене на кілька рядків
puts \
«І ми обов'язково цьому навчимося!»
```

Інструкція (твердження) найчастіше являє собою якусь послідовність операторів, застосованих до різних виразів, і (можливо) викликів функцій.

Далі рядок

```
#!/usr/bin/env ruby
```

у тексті програм не включається.

### 1.4.4.2. Об'єкти та методи Ruby

При описі Ruby було відзначено, що вона є об'єктно-орієнтованою мовою програмування. Це означає що все, з чим працює програма, є об'єктом, при цьому обчислення здійснюється за допомогою взаємодії об'єктів. Кожен об'єкт є представник (екземпляр) деякого класу і його поведінки (функціональності) визначається саме класом. Тим самим всі об'єкти, які є екземплярами одного класу, можуть виконувати ті самі дії, що називаються методами. Для того, щоб застосувати метод до деякого об'єкту використовується оператор виклику методу, який позначається символом `.` (крапка): після вказівки об'єкта ставиться крапка, а потім вказується застосований метод.

Знайомство із представниками різних класів почнемо із чисел. Цілі числа в Ruby є екземплярами класу `Integer`, що поєднує два підкласи: `Fixnum` або `Bignum`. Об'єктами класу `Fixnum` є ті цілі числа, чиє двійкове подання здатне розміститися в машинному слові (31 біт на більшості комп'ютерів). Якщо число виходить за межі зазначеного діапазону, то воно автоматично перетвориться в об'єкт класу `Bignum`, чий діапазон змін обмежується тільки об'ємом доступної пам'яті. Якщо в результаті роботи з об'єктами класу `Bignum` підсумкове значення попадає в діапазон, що задається класом `Fixnum`, то вихідний результат перетвориться в екземпляр класу `Fixnum`. При записі цілих чисел спочатку вказується його знак (знак `+` звичайно опускається). Далі йде основа системи числення, у якій задається число (якщо вона відмінна від 10): `0` – для вісімкової, `0x` – для шістнадцяткової, `0b` – для двійкової. Потім йде послідовність цифр, що виражає число в даній системі числення. При записі більших чисел можна використати

символ підкреслення, що ігнорується при обробці числа. Дробові числа задаються в десятковій системі числення, при цьому для відділення дробової частини використовується символ `.` (крапка). Такі числа є екземплярами класу `Float`, наприклад, `12.34`. Для задання дробових чисел може бути застосована й експонентна форма запису: два різних подання `-1.234e2` й `1234e-2` задають одне й теж число `12.34`.

Для обчислення арифметичних виразів застосовуються наступні оператори: `+` (додавання), `-` (віднімання), `*` (множення), `/` (ділення), `%` (остача від ділення), `**` (піднесення в степінь). Порядок обчислення визначається стандартними пріоритетами операцій, а для його зміни використовуються круглі дужки. Зауважимо, що якщо всі аргументи виразу цілі числа, то й результат буде цілим, якщо ж хоча б одне із чисел, що входять у вираз – дробове, то й результат буде екземпляром класу `Float`.

Приклад.

Створіть файл із ім'ям `object.rb`, у який помістіть наступний текст

```
puts 5/8 # 0
puts 5.0/8 # 0.625
puts 2**1000
puts ((2*500+1)*(2**500-1))
```

Виконайте програму й поясніть результат.

В Ruby є кілька методів, що дозволяють перетворювати об'єкти одного класу в інший.

Метод	Призначення методу	Приклад використання	Результат
<code>to_f</code>	Перетворити об'єкт в екземпляр класу <code>Float</code>	<code>1234.to_f</code>	<code>1234.0</code>
<code>to_i</code>	Перетворити об'єкт в екземпляр класу <code>Fixnum</code> або <code>Bignum</code>	<code>-12.34.to_i</code>	<code>-12</code>

В останньому прикладі перша крапка відокремлює дробову частину числа, а друга є оператор виклику методу.

Відзначимо ще кілька методів, які використовуються при роботі з числами (тобто представниками класу `Numeric`). Серед них: `ceil` (знаходження найменшого цілого не меншого, ніж дане), `floor` (найбільше ціле, не більше даного), `round` (округлення до найближчого цілого), одержання абсолютної величини числа `abs`. Нижче наведені приклади використання цих методів.

```
puts 12.34.ceil # 13
puts 12.34.floor # 12
puts -12.ceil # -12
puts -12.floor # -12
puts 12.34.round # 12
puts 12.54.round # 13
puts -34.56.abs # 34.56
```

Іншим настільки ж розповсюдженим класом є клас `String`. До цього класу відносяться довільні рядки символів, вкладені в апострофи або лапки, наприклад, `'hello'`, `'раз, два, три'`, `«привіт усім»`. Є кілька альтернативних способів задання кожного із цих видів рядків.

Для задання рядка в апострофах можна використати кожний зі способів представлених нижче:

```
puts 'hello' # hello
puts %q/hello/ # hello
puts %q(hello) # hello
```

Два символи `\`, що йдуть один за одним, у середині такого рядка замінюються на один.

```
puts 'hell\\o' # hell\o
puts %q(hell\\o) # hell\o
```

Для того, щоб у середині рядка, взятого у лапки, вставити апостроф, треба попередньо «екранувати» його символом `\`.

```
puts 'hell\'o' # hell'o
puts %q(hell\'o) # hell'o
puts 'hel\b'o' # hel\b\o
```

Для створення рядка, взятого в лапки, треба або просто взяти його в лапки, або використати конструкції `%Q` або `%`, після яких вказується рядок, виділений з двох сторін символом, відмінним від цифр і букв (можна використати пару будь-яких дужок).

Нижче перераховано кілька способів задання рядка «hello»:

```
<hello> %Q/hello/ %Q{hello} %Q(hello) %Q!hello!
%<hello> %(hello) %!hello! %*hello* %+hello+
```

Для включення у середину рядка, взятого в лапки, символу `«`, варто екранувати його символом `\`:

```
puts «Say \»Hello\»» # Say «Hello»
```

Рядок, взятий у лапки, дозволяє інтерпретувати послідовності символів, що починаються із символу `\` (зворотний слеш), такі, наприклад, як `\n` (символ переходу на новий рядок) і `\t` (табуляція). Іншою особливістю рядків, взятих у лапки, є можливість використання підстановки: якщо рядок містить деякий вираз, обмежений символами `{ }` й `|`, тоді він замінюється на результат його обчислення. Додайте в програму наступні рядки й проаналізуйте отриманий висновок.

```
puts «a \t b»; puts 'a \t b'
puts «вираз 3*5+8 дорівнює #{3*5+8}»
puts 'вираз 3*5+8 дорівнює #{3*5+8}'
puts «робота із цілими числами: 5/8 = #{5/8}»
puts «переведення у клас Float: 5/8 = #{5/8.to_f}»
```

До рядків також можуть застосовуватися методи `to_i` й `to_f`. При перетворенні до цілого числа відкидається кінцева частина рядка, як тільки зустрічається символ відмінний від цифри (виключення – знак плюс або мінус у першій позиції рядка). Аналогічні правила застосовуються й при перетворенні до дробового числа. Єдиною відмінністю є те, що перша знайдена крапка розцінюється як символ відділення дробової частини. Наступний фрагмент ілюструє сказане:

```
puts <-12.34>.to_i
puts <12.34>.to_f
```

```
puts «+12:34».to_i
puts «12qq34».to_f
```

Для одержання рядка, що містить символ із заданим ASCII кодом, використовується метод `chr`, наприклад,

```
puts 209.chr
```

Варто пам'ятати, що цей метод може бути застосований тільки до позитивного цілого числа, що не перевищує 255.

Окремого згадування заслуговує метод `eval`, що дозволяє динамічно виконувати інші методи. Цей метод аналізує переданий йому рядок, розглядаючи його як частину програми, і виконує її. Зверніть увагу, що при виклику цього методу використовується не крапкова, а функціональна форма запису:

```
puts eval («2**10»)
puts eval («»мол».size * «око».size») # 9
```

Екземпляр класу `Time` у мові Ruby містить інформацію про дату, час і тимчасову зону. Для створення об'єкту цього класу, що містить поточну дату й час, використовується метод `now`. Такі об'єкти можуть бути аргументами операцій `+` та `-`:

```
puts Time.now
puts Time.now+60 # додали 60 секунд до поточного часу
puts Time.now-60 # відняли 60 секунд від поточного часу
```

У першій частині нашої теми ми вже згадували, що з погляду ОС Linux вхідний і вихідний потоки даних (за замовчуванням – введення з клавіатури й вивід у вікно shell) нічим не відрізняються від інших файлів – той же потік байтів. Тому введення комбінації клавіш `Ctrl+D`, що означає переривання введення даних, розцінюється аналогічно кінцю файлу. У нашому випадку дія методів `readline` й `gets` відрізняється лише реакцією на виявлення кінця файлу: у першому випадку видається помилка `EOFError` (End Of File Error), а в другому повертається об'єкт `nil`.

Зверніть увагу, що останній символ введеного рядка (незалежно від методу, що використовувався) є `\n`. Це є наслідком натискання на клавішу `Enter` (в інших операційних системах кінець рядка може кодуватися іншими символами).

```
print «Введіть рядок: «
a=gets
b=a.size
print «ASCII код останнього символу дорівнює «, a[b - 1], «\n»
```

Вище вже описувався метод `chop`, що дозволяє видалити останній символ рядка (або два останніх символи, якщо вони є `\r\n`).

```
print «Уведіть ваше ім'я: «
a=gets
print a, «, привіт!\n»
b=a.chop
print b, «, привіт!\n»
```

Як бачите, символ перекладу на новий рядок вилучений.

В Ruby багато методів мають своїх «двійників», що містять знак `!` після свого імені. Це так звані «небезпечні» методи, які руйнують об'єкт, до якого вони застосовуються. Так, для об'єктів типу `String` припустимий метод `chop`, так й `chop!`. Перший з них, застосований до рядка, повертає новий об'єкт – рядок з вилученим останнім символом, а метод `chop!` змінює сам вихідний об'єкт.

```
a=»123456»
b=a.chop
puts a, b
b=a.chop!
puts a, b
```

Змінімо програму, додавши до неї метод `chop!`:

```
print «Введіть ваше ім'я: «
a=gets.chop!
print «Привіт «, a, «! Як поживаєте?\n»
```

Тепер наше вітання друкується на одному рядку.

При введенні числової інформації варто примусово перетворити отриманий рядок у число за допомогою методів `to_i` та `to_f`. Нижче приводиться фрагмент програми, що запитує два числа й друкує результати їхньої обробки.

```
print «Введіть ціле число: «
num=gets.to_i
print «Остача від ділення на 5 дорівнює #{num%5}\n»
print «Введіть число (крапка відокремлює дробову частину): «
num=gets.to_f
print «При округленні вийшло число #{num.round}\n»
```

Раніше розглянутий метод `eval` дозволяє значно розширити можливості введення даних.

Припустимо, що нам потрібно ввести не число, а арифметичний вираз (можливо, він містить операції `*`, `/`, `+`, `-`, `**` або імена методів для роботи з числами). Тоді, після видалення символу переводу рядка, треба до отриманого рядка застосувати метод `eval`:

```
puts «Ruby не боїться величезних чисел.»
puts «Введіть страшний арифметичний вираз, «
puts «наприклад, 14**256+3*17»
puts eval gets.chop!
```

За допомогою цього методу можна виконати команди, введені з клавіатури.

```
puts «Введіть команди, розділяючи їх крапкою з комою:»
eval gets.chop!
```

Нижче приводиться приклад виконання цієї програми:

Введіть команди, розділяючи їх крапкою з комою:

```
puts Time.now; print «2+3=»; puts 2+3
Sat Mar 30 11:47:42 MSK 2012
2+3=5
```

Методи, поряд зі змінними, є основними будівельними блоками в Ruby. Ми вже застосовували методи, визначені для таких класів, як числа й рядки.

При написанні програм часто доводиться створювати додаткові методи (які зветься також функціями або підпрограмами). Кожен метод повинен бути визначений до свого використання за допомогою ключових слів `def` та `end`. Як ми вже відзначали, ім'я методу повинне починатися з рядкової латинської букви (від `a` до `z`). Також як і змінним, методам рекомендується давати осмислені імена. Якщо ім'я складається з декількох слів, то для їхнього поділу рекомендується використати символ підкреслення, або кожне слово, починаючи із другого виділяти великою буквою.

Методам, обумовленим поза визначенням класу, присвоюється спеціальна позначка `private` (приватний), яка зветься специфікатором доступу. При звертанні до такого методу використовується функціональна форма виклику методу, а не оператор виклику методу `.` (крапка). З іншої сторони методи, визначені у середині визначення класу, за замовчуванням позначаються як `public`. При виклику методів, визначених таким чином, використовується «крапкова» форма виклику методу. Через те, що ми поки не створюємо нових класів у своїх програмах, то всі методи, розроблені нами, будуть викликатися тільки у функціональному стилі.

Перепишемо нашу першу програму так, щоб вона використала виклик методу. Для цього у файл із ім'ям `method.rb` помістимо наступний текст:

```
def helloWorld
  puts «Hello, World!»
end
helloWorld
```

Частина програми між словами `def` й `end` (у нашому випадку єдиний оператор `puts`) становить тіло методу.

Змінимо метод так, щоб він міг друкувати будь-який текст:

```
def saySomething(text)
  puts text
end
saySomething(«Hello, World!»)
```

Ми додали параметр `text` у наш метод. При визначенні методу параметри завжди беруть у круглі дужки, які йдуть відразу за ім'ям методу та розділяються комами.

Параметри методу є локальними змінними. Це означає (у термінах наших наклеїнок й об'єктів), що метод оперує копіями змінних, переданих йому (часто говорять, що параметри передаються за значенням). Інакше кажучи, перед тим як передати мітки й створити метод Ruby створює копії всіх цих об'єктів тимчасово, тільки на час виконання методу поміщає імена параметрів на них. Проілюструємо це наступним прикладом:

```
def printOneMoreThan(x)
  x += 1
  puts «Під час виконання: #{x}»
end
x = 1
puts «До: #{x}»
```

```
printOneMoreThan(x)
puts «Після: #{x}»
```

Виконаємо цю програму й переконаємося, що значення змінної `x` не змінилося після завершення методу `printOneMoreThan`.

При створенні методу Ruby дозволяє задати значення деяких аргументів. Ці, так звані параметри за замовчуванням, дають можливість виклику без обов'язкової вказівки значення всіх параметрів. Нижче приводиться приклад такого методу.

```
def saySomething(text = 'Hello, World!')
  puts text
end
saySomething
```

При цьому виклику методу параметр не вказувався. Ruby дозволяє викликати метод декількома різними способами:

```
saySomething
saySomething()
saySomething «Пробіл між ім'ям методу й аргументами»
saySomething(« або використання дужок»)
```

Рекомендується завжди розміщати аргументи методу в круглих дужках. Це дозволить уникнути деяких помилок і непорозумінь. Загальноприйнятими виключеннями є методи друку – `puts`, `print` і т.д., хоча й при їхньому виклику аргументи також можна брати в дужки:

```
puts(2 + 3)
print(«Hello, », 3 + 4, «\n»)
```

Значеннями за замовчуванням можуть бути будь-які вирази Ruby. Вони обчислюються в момент виклику методу, причому обчислення відбувається зліва направо. У цих виразах можуть використовуватись й інші параметри. Наступний приклад демонструє поведінку методу залежно від кількості аргументів при виклику:

```
def options(a=99, b=a+1)
  [a, b]
end
p options # [99, 100]
p options(1) # [1, 2]
p options(2, 4) # [2, 4]
```

Після всіх звичайних аргументів як аргумент може бути зазначений масив, що позначається символом `*` перед його ім'ям. При виклику в такий спосіб певного методу Ruby підраховує кількість переданих аргументів і, якщо їхня кількість більше ніж число звичайних параметрів, то ті, що залишилися, розміщає в новому об'єкті класу `Array`. Наприклад,

```
def varargs(a, *b)
  [a, b]
end
p varargs(1) # [1, []]
```

```
p varargs(1, 2) # [1, [2]]
p varargs(1, 2, 3) # [1, [2, 3]]
```

Якщо аргумент-масив йде за аргументами, що мають значення за замовчуванням, то при недостатці аргументів використовуються значення за замовчуванням, у протилежному випадку розміщення параметрів аналогічне попередньому. Проілюструємо сказане прикладом:

```
def mixed(a, b=99, *c)
  [a, b, c]
end
p mixed(1) # [1, 99, []]
p mixed(1, 2) # [1, 2, []]
p mixed(1, 2, 3) # [1, 2, [3]]
p mixed(1, 2, 3, 4) # [1, 2, [3, 4]]
```

Дуже часто від методу потрібне повернення значення, отриманого в результаті його виконання. Для цього використовується конструкція return виразу (якщо ви опустите вираз, залишивши тільки return, то повертається значення nil). Якщо не використати оператор return, то значення, що повертає метод, буде значенням останнього виразу, обчисленого в методі.

Синтаксис умовного оператора if у мові Ruby аналогічний синтаксису в більшості інших мов програмування. Наприклад,

```
# Визначення методу оцінки величини виразу
def howBig(i)
  if i < 10
    puts «менше 10»
  elsif i < 20
    puts «між 10 й 20»
  elsif i < 30
    puts «між 20 й 30»
  else
    puts «більше або дорівнює 30»
  end
end
# Використання методу ...
howBig(7); howBig(15)
howBig(23); howBig(105)
Загальна форма оператора if така:
if <логічний_вираз> [then]
  тіло_оператора
elsif <логічний_вираз> [then]
  тіло_оператора
...
else
  тіло_оператора
end
```

Тут <логічний\_вираз> може бути будь-яким фрагментом коду мовою Ruby, результатом обчислення якого є логічна величина (з урахуванням сказаного вище). Слово then відокремлює тіло оператора від умови. Запис його у квадратних дужках означає, що воно може бути опущено, якщо тіло починається з нового рядка. Значення, що повертає оператор if, є результат останнього обчисленого виразу. Змінимо наш приклад, щоб продемонструвати сказане:

```
# Наш метод оцінки величини виразу
# тепер повертає рядок
def howBig(i)
  if i < 10 then «менше 10»
  elsif i < 20 then «між 10 й 20»
  elsif i < 30 then «між 20 й 30»
  else «більше або дорівнює 30»
  end
end
# Використання методу ...
puts howBig(7); puts howBig(15)
puts howBig(23); puts howBig(105)
```

#### 1.4.5. Технологія ASP

ASP (Active Server Pages) – технологія, розроблена компанією Microsoft, яка дозволяє легко створювати застосування для Веб-сервера.

Програмування на ASP дає розробникам доступ до інтерфейсу програмування застосувань Internet Information Server за допомогою мови сценаріїв VBScript і JScript.

ASP працює на платформі операційних систем лінії Windows NT і на веб-сервері Microsoft IIS (Рис. 1.4.5.1).

Файлами ASP є сценарії, що інтерпретуються у міру надходження запитів. ISAPI-розширення ASP.DLL зв'язано в IIS з розширеннями файлів .asp або .asa.

Порядок обробки таких файлів виглядає таким чином:

- ASP.DLL проглядає файли із вказаними розширеннями на наявність тегів, що позначають введення код для виконання на сервері і передає знайдений код у Windows Script Host (WSH);
- WSH виконує цей код і повертає результат файлу ASP.DLL;
- ASP.DLL передає IIS цей результат і вміст самого файлу ASP;
- IIS повертає відповідь клієнтові, від якого поступив запит.

Розглянемо основи синтаксису ASP. IIS, який розрізняє код, що виконується на сервері, і вміст, що відправляється клієнтові за допомогою ASP.DLL, аналізуючи файл ASP на наявність початкового «<%» і кінцевого «%>» тегів та виконує код, розташований між ними, за допомогою WSH.

Розглянемо приклад:

```
<% Language=VBScript %>
<HTML>
<BODY>
```



```

<%
Response.Write(«<p>Hello world!</p>»)
%>
</BODY>
</HTML>

```

У прикладі перший рядок коду `<% Language=VBScript %>` повідомляє про необхідність використовувати інтерпретатор мови VBScript. Для вставки рядка в документ був використаний метод `Write` стандартного об'єкту `Response`.

Подія веб-запиту в ASP обробляється за допомогою наступних об'єктів:

- ⇒ *Response*. Використовується для запису даних в запит HTTP, що повертається клієнтові;
- ⇒ *Application*. Містить параметри і конфігурації по налаштуванню роботи ASP для даного веб-сайту;
- ⇒ *Request*. Зберігає вміст HTTP-запиту і забезпечує допоміжні функції для обробки даних HTTP-запиту;
- ⇒ *Server*. Містить інформацію про веб-сервер, веб-сайт, а також забезпечує підтримку викликаючої програми;
- ⇒ *Session*. Є станом заданого веб-сеансу із заданим хостом клієнтом.

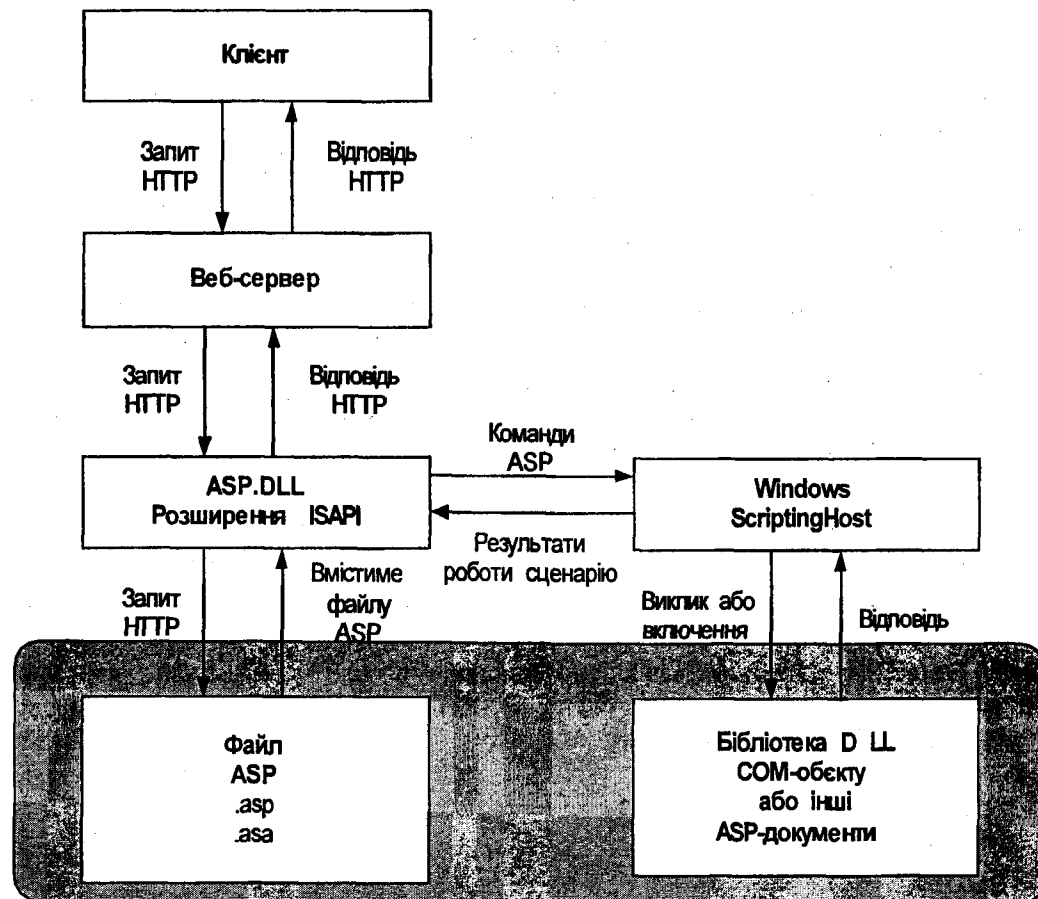


Рис. 1.4.5.1 Архітектура ASP

### 1.4.6. Інтерфейс ISAPI

Для веб-сервера IIS (Internet Information Server) був розроблений спеціальний програмний інтерфейс для створення застосунків, що розширюють стандартні можливості веб-сервера.

ISAPI (Internet Server Application Programming Interface) – багатоланковий API для IIS.

ISAPI також реалізований у вигляді модуля `mod_isapi` для веб-сервера Apache. Таким чином, серверні застосунки, розроблені для MS IIS, можуть також виконуватися в Apache і інших веб-серверах.

У протилежність CGI – ISAPI-застосування завантажуються в тому ж адресному просторі, що і веб-сервер IIS. Це дозволяє підвищити продуктивність застосунків завдяки скороченню витрат на запуск окремих процесів. Проте збій ISAPI-застосування може привести до нестійкої роботи самого веб-сервера. У 6-й версії IIS є можливість запуску застосунків в рамках окремого процесу.

ISAPI включає 2 компоненти: розширення і фільтри.

Таким чином, все різноманіття ISAPI-застосунків, що розробляються, зводиться тільки до цих двох типів. І фільтри і розширення компілюються в DLL файли, що динамічно запускаються веб-сервером.

ISAPI застосування можуть розроблятися за допомогою будь-яких мов, що підтримують експорт стандартних C-функцій, наприклад C, C++, Delphi. Для розробки є обмежене число бібліотек для розробки ISAPI застосунків, наприклад, IntraWeb-компоненти Delphi Pascal, спеціальні MFC-класи, спеціальна C++ бібліотека серверних технологій ATL.

До найбільш важливих особливостей ISAPI-розширень можна віднести наступні:

- ISAPI-розширення мають доступ до всіх функціональних можливостей IIS;
- реалізуються у вигляді DLL-модулів, що завантажуються в просторі процесу, контрольованого IIS;
- клієнти можуть звертатися до ISAPI-розширень також як до статичних сторінок HTML;
- ISAPI-розширення можуть бути асоційовані з окремими розширеннями файлів, з цілими каталогами або сайтами;
- ISAPI-фільтри необхідні для зміни або вдосконалення функціональності IIS. Вони зазвичай працюють з IIS-сервером і фільтрують кожен запит. Фільтри застосовуються для аналізу і модифікації вхідних і вихідних потоків даних.
- фільтри також як і розширення реалізуються у вигляді DLL файлів.

Зазвичай ISAPI-фільтри використовуються для вирішення наступних завдань:

- ⇒ зміна даних в запиті клієнта (URL або заголовків);
- ⇒ управління відображенням URL у фізичні файли;
- ⇒ управління іменами і паролями користувачів при анонімній або базовій аутентифікації;
- ⇒ аналіз і модифікація запитів після закінчення аутентифікації;

- ⇒ модифікація відповіді веб-сервера;
- ⇒ ведення журналів і аналіз трафіку;
- ⇒ реалізація власної аутентифікації;
- ⇒ управління шифрацією і стискуванням.

Варто відзначити, що існують реалізації у вигляді ISAPI-розширень для таких інструментальних засобів як:

- ⇒ ASP (Active Server Pages);
- ⇒ ASP.NET;
- ⇒ ColdFusion;
- ⇒ Perl ISAPI (Perlis);
- ⇒ PHP.

## Контрольні запитання

1. Що називається Інтернет?
2. Дати визначення проксі-серверу?
3. Назвіть недоліки Інтернету?
4. Розкрийте назву поняття http?
5. Що таке W3C?
6. Для яких завдань використовуються ISAPI-фільтри. Що у веб-застосуванні виступає клієнтом, а що сервером?
7. Без чого неможливий жоден HTTP-обмін?
8. Чи передається клієнтові код програми, що працює на сервері?
9. Що таке насичене інтернет-застосування. У використанні чого він полягає?
10. Які пакети використовуються для розробки інтернет-застосувань?
11. Що таке плагін?
12. Сформулюйте означення сценарію?
13. Що являє собою стандарт CGI?
14. Як у переважній більшості випадків здійснюється запуск CGI-сценарію?
15. Скільки існує способів розпізнавання файлів, що містять тексти CGI-сценаріїв?
16. Який найприродніший формат для браузера?

## Тести для закріплення матеріалу

1. Фізичною основою мережі є:

- А) ARPANET;
- Б) NSFnet;
- В) Інтернет.

2. Інтернет – це :

- А) винахід розподіленої електронної пошти;
- Б) найбільша в світі мережа, що не має єдиного центру управління;
- В) служба в комп'ютерних мережах, що дозволяє клієнтам виконувати непрямі запити до інших мережеслужб.

3. Мережа ARPANET була створена в :

- А) 1959р.;
- Б) 1969р.;
- В) 1979р..

4. Перші лінії зв'язку, що сполучали вузли, працювали на швидкості:

- А) 45 кбіт/с.;
- Б) 50 кбіт/с.;
- В) 65 кбіт/с.

5. Проксі-сервер – це :

- А) винахід розподіленої електронної пошти;
- Б) служба в комп'ютерних мережах, що дозволяє клієнтам виконувати непрямі запити до інших мережеслужб;
- В) з'єднання різноманітного устаткування.

6. RFC – це :

- А) документ з серії пронумерованих інформаційних документів Інтернету;
- Б) з'єднання різноманітного устаткування;
- В) найбільша в світі мережа, що не має єдиного центру управління.

7. Основним організаційним підрозділом, що координує роботу по стандартизації Інтернету є :

- А) W3C;
- Б) ISOC;
- В) OSI.

8. Консорціум W3C – це :

- А) організація, що розробляє і впроваджує технологічні стандарти для Інтернету і WWW;
- Б) інженерна група, що визначає специфікації для подальших стандартів Інтернет;
- В) з'єднання різноманітного устаткування.

9. W3C розробляє для WWW єдині принципи і стандарти, що мають назву:

- А) «зручності»;

Б) «рекомендації»;

В) «структури».

10. HTTP – це :

- А) протокол прикладного рівня для передачі гіпертексту;
- Б) інженерна група, що визначає специфікації для подальших стандартів Інтернет;
- В) винахід розподіленої електронної пошти.

11. У веб-застосуванні сервером виступає:

- А) веб-сервер;
- Б) веб-плагін;
- В) веб-браузер;
- Г) сервер.

12. Жоден HTTP- обмін неможливий без :

- А) клієнта і сервера;
- Б) веб-застосування і сторінки;
- В) тегів.

## РОЗДІЛ 2



## ВЕБ-ПРОГРАМУВАННЯ

ТЕМА 2.1. JAVASCRIPT. ПРОГРАМНА ВЗАЄМОДІЯ  
З HTML ДОКУМЕНТАМИ НА ОСНОВІ DOM API

## 2.1.1. Загальний огляд мови JavaScript

## 2.1.2. Об'єктна модель JavaScript

## 2.1.3. Забезпечення ефективності WEB – сайтів

## 2.1.4. Адаптація сайту до клієнтського програмного забезпечення

## 2.1.5. Коротка характеристика VBScript

## 2.1.6. Java-апплети

## 2.1.7. ActionScript, XAML і Microsoft Silverlight – загальна характеристика

## 2.1.8. Поняття про DOM та HTML DOM

## 2.1.1. Загальний огляд мови JavaScript

JavaScript - це мова програмування, що використовується в складі HTML-сторінок для збільшення їх функціональності та можливостей взаємодії з користувачем. JavaScript є однією із складових динамічного HTML. Ця мова програмування була створена фірмами Netscape та Sun Microsystems на базі мови програмування Sun's Java. На сьогодні є декілька версій JavaScript. Однією із найбільш поширених є версія JavaScript 1.3. За допомогою JavaScript на HTML-сторінці можливо зробити те, що не можливо зробити за допомогою стандартних тегів HTML.

Код програми JavaScript розміщується або в середовищі HTML-сторінки, або в текстовому файлі, що пов'язаний за допомогою спеціальних команд з HTML-сторінкою. Цей код, як правило, розміщується в середині тегу HTML та завантажується в браузер разом з кодом HTML-сторінки. Програма JavaScript не може існувати самостійно, тобто без HTML-сторінки.

Виконання програми JavaScript відбувається при перегляді HTML-сторінки в браузері, звичайно, тільки в тому випадку, коли браузер містить інтерпретатор JavaScript. Практично всі сучасні популярні браузери оснащені таким інтерпретатором. Нагадаємо, що крім JavaScript на HTML-сторінках можливо використовувати інші мови програмування. Наприклад, VBScript або JScript, яка є варіантом JavaScript від фірми Microsoft. Але виконання програм VBScript та JScript гарантовано коректне тільки при перегляді HTML-сторінки за допомогою браузера Microsoft Internet Explorer. Тому в більшості випадків використання JavaScript доцільніше, хоча функціональність програм VBScript та JScript дещо краща.

Досить часто програму JavaScript називають скриптом або сценарієм. Скрипти виконуються в результаті того, що відбулась деяка подія, пов'язана з HTML-сторінкою. В багатьох випадках виконання вказаних подій ініціюється діями користувача.

Скрипт може бути пов'язаний з HTML-сторінкою двома способами:

- за допомогою парного тегу SCRIPT;
- як оброблювач події, що стосується конкретного тегу HTML.

Сценарій, вбудований в HTML-сторінку з використанням тегу SCRIPT, має наступний формат:

```
<SCRIPT>
// Код програми
</SCRIPT>
```

Все, що розміщується між тегами <SCRIPT> та </SCRIPT>, інтерпретується як код програми на мові JavaScript. Обсяг вказаного коду не обмежений. Інколи скрипти розміщують в середині HTML-коментаря. Це роблять для того, щоб код JavaScript не розглядався старими браузерами, які не мають інтерпретатора JavaScript. В цьому випадку сценарій має формат:

```
<SCRIPT>
<!--
// Код програми
-->
</SCRIPT>
```

Тег SCRIPT має декілька необов'язкових параметрів. Найчастіше використовуються параметри language та src. Параметр language дозволяє визначити мову та версію мови сценарію. Параметр src дозволяє задати файл з кодом сценарію. Для пояснення використання параметрів тегу SCRIPT розглянемо задачу.

**Задача.** Необхідно для HTML-сторінки hi.htm створити сценарій на мові JavaScript 1.3 для показу на екрані вікна повідомлення з текстом «Привіт!».

Відзначимо, що для показу на екрані вікна повідомлення можна використати функцію alert.

Для ілюстрації можливостей пов'язування скриптів з HTML-кодом вирішення задачі реалізуємо двома варіантами.

**Варіант 1.** Визначення сценарію безпосередньо на HTML-сторінці hi.htm

```
<html><head>
<title>Використання JavaScript</title>
</head>
<body>
<script language=»JavaScript1.3»>
  alert('hi');
</script>
</body></html>
```

**Варіант 2.** Визначення сценарію в файлі a.js, пов'язаному з HTML-сторінкою hi.htm за допомогою параметру src тегу SCRIPT. Код HTML-сторінки hi.htm:

```
<html><head>
```

```
<title>Використання JavaScript</title>
<script language=»JavaScript1.3» src=»a.js»> </script>
</head><body>
</body></html>
```

Програмний код, записаний в файлі a.js:  
alert('hi');

Результат виконання обох варіантів вирішення задачі однаковий і показаний на рис.2.1.1.1.

Змінні та вирази JavaScript можна використовувати в якості значень параметрів тегів HTML. В цьому випадку елементи JavaScript розміщуються між амперсандом (&) та крапкою з комою (;), але повинні бути обмежені фігурними дужками {} і використовуватись тільки в якості значень параметрів тегів.

Наприклад, нехай визначена змінна c і їй присвоєно значення green. Наступний тег буде виводити текст зеленого кольору :

```
<font color=»&{c};»> текст зеленого кольору </font>
```

Відзначимо, що мінімальним комплектом програмного забезпечення для розробки та тестування програм JavaScript є текстовий редактор та браузер з підтримкою JavaScript.

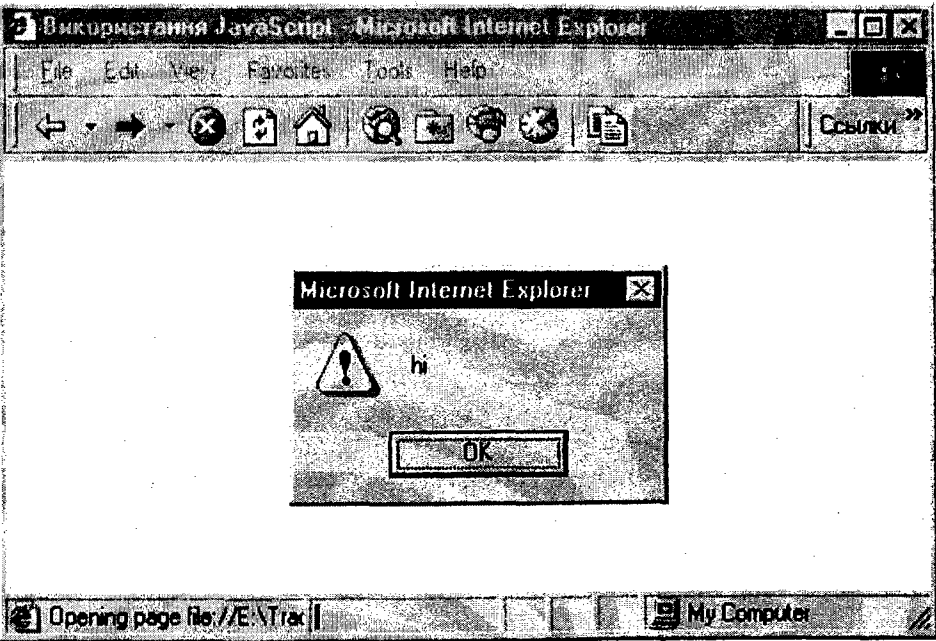


Рис. 2.1.1.1 Показ вікна повідомлення засобами JavaScript.

Сценарій JavaScript являє собою набір операторів, що послідовно інтерпретуються браузером. Оператори можна розміщувати як в одному, так і в окремих рядках. Якщо оператори розміщені в одному рядку, то між ними необхідно поставити ;. В протилежному випадку ; не обов'язкова. Будь-який оператор можна розмістити в декількох рядках без символу продовження.

Будь-яка послідовність символів, розміщених в одному рядку, якій передує //, розглядається як коментар. Для визначення багаторядкових коментарів використовується конструкція:

```
/*
Багаторядковий коментар
*/
```

В мові JavaScript рядкові та приписні букви вважаються різними символами. JavaScript використовує змінні для зберігання даних визначеного типу. При цьому JavaScript є мовою з вільним використанням типів. Тобто не обов'язково задавати тип змінної, який залежить від типу даних, що в ній зберігаються. При зміні типу даних автоматично змінюється і тип змінної.

JavaScript підтримує чотири простих типи даних. Ілюстрацією цього є табл. 2.1.1.1. Для присвоєння змінним значень основних типів використовуються літерали.

Таблиця 2.1.1.1

Типи даних JavaScript

Тип даних	Пояснення	Приклад літерала
Цілий	Послідовність цілих чисел	789 +456 -123
З плаваючою крапкою	Числа з крапкою, яка відділяє цілу частину від дробової, або числа в науковій нотації	7.25 0.525e01 71.2E-4
Рядковий	Послідовність алфавітно-цифрових символів, взятих в одинарні ('), або подвійні («») лапки.	«Привіт» «234» «Hello World!!!»
Булевий або логічний	Використовуються для оброблення ситуацій так/ні в операторах порівняння	true false

Ім'я змінної повинно містити тільки букви латинського алфавіту, символ підкреслення \_, арабські цифри та починатись з букви або символу підкреслення \_. Довжина імені повинна бути менша від 255 символів. Заборонено використовувати імена, що збігаються з ключовими словами JavaScript. Приклади імен: \_hello, go, go123.

Визначити змінну можна:

- оператором var, наприклад, var a;
- при ініціалізації, за допомогою оператора присвоєння (=), наприклад, b=126.

Визначення та ініціалізацію змінних можна реалізувати в будь-якому місці програми.

**\*** *Вираз – це комбінація змінних, літералів та операторів, в результаті обчислення яких можна отримати тільки одне значення, яке може бути числовим, рядковим або булевим.*

Для реалізації обчислень в JavaScript використовуються арифметичні, рядкові, логічні вирази та декілька типів операторів:

1. Арифметичні вирази – обчислюють число, наприклад, a=7+5;
2. Рядкові вирази – обчислюють рядок символів, наприклад, "Джон" або "234";
3. Логічні вирази – обчислюють true (істина) або false (хибність).

Оператор присвоювання (=) - присвоює, значення лівому операнду, базуючись на значенні правого операнда. Наприклад, для присвоєння змінній a значення числа 5 необхідно записати:

```
a=5
```

До стандартних арифметичних операторів відносяться: оператори додавання (+), віднімання (-), множення (\*), ділення (/), остача від ділення чисел (%), збільшення числової змінної на 1 (++), зменшення числової змінної на 1 (--).

Відзначимо, що оператор додавання можна використовувати не тільки для чисел, але й для додавання (конtrakтації/конкатенації) текстових рядків.

Для створення логічних виразів використовуються логічні оператори та оператори порівняння.

До логічних операторів відносяться - логічне І (&&), логічне АБО (||), логічне НІ (!).

Оператори порівняння не відрізняються від таких операторів в інших мовах програмування. До операторів порівняння відносяться (==, >, >=, <=, !=).

Оператори вибору відносяться до операторів управління, призначенням яких є зміна напрямку виконання програми. Крім операторів вибору до операторів управління відносяться: оператори циклу та оператори маніпулювання об'єктами.

Оператори вибору призначені для виконання деяких блоків операторів в залежності від істинності деякого логічного виразу. До операторів вибору відносяться: оператор умови if...else та перемикач switch.

Синтаксис оператора умови такий:

```
if (умова) {
    група операторів 1
    .....
}
[else] {
    група операторів 2
    .....
}
```

Перша група операторів виконується при умові істинності виразу умови. Необов'язковий блок else визначає другу групу операторів, яка буде виконуватись у випадку хибності умови, заданої в блоці if. В середині групи операторів можуть бути використані будь-які інші оператори, в тому числі і інші оператори умови. Це дозволяє створювати групу вкладених операторів умови if та реалізовувати складні алгоритми перевірки. Однак, якщо кількість вкладених операторів if більша ніж три, то програма стає складною для розуміння. В такому випадку доцільно використовувати оператор switch. В цьому операторі обчислюється деякий вираз та порівнюється зі значенням, заданим в блоках case. Синтаксис оператора switch такий:

```
switch (вираз) {
    case значення1:
        [оператори1]
        break;
    case значення2:
```

```
[оператори2]
```

```
break;
```

```
...
```

```
default:
```

```
[оператори]
```

```
}
```

Якщо значення виразу в блоці switch дорівнює значення1, то виконується група операторів оператори1, якщо дорівнює значення2, то виконується група операторів оператори2 і так далі. Якщо значення виразу не дорівнює ні одному із значень, що задані в блоках case, то обчислюється група операторів блоку default, якщо це блок заданий, інакше - виконується вихід із оператору switch. Необов'язковий оператор break, який можна задавати в кожному із блоків case, виконує безумовний вихід із оператору switch. Якщо він не заданий, то продовжується виконання операторів в наступних блоках case до першого оператору break або до кінця оператору switch.

Цикл - це деяка група команд, що повторюється доки вказана умова не буде виконана. JavaScript підтримує дві форми циклу: for та while. Крім того оператори break та continue використовуються разом з циклами.

Цикл for повторює групу команд до тих пір, поки вказана умова хибна. Синтаксис оператору for такий:

```
for ([initial-expression]; [condition]; [increment-expression])
{
    statements
}
```

Виконання циклу for проходить в такій послідовності:

1. Вираз initial-expression служить для ініціалізації змінної лічильника. Цей вираз розраховується один раз на початку виконання циклу.

2. Вираз condition розраховується на кожній ітерації циклу. Якщо значення виразу condition дорівнює true, виконується група операторів statements в тілі циклу. Якщо значення виразу condition дорівнює false, то цикл for закінчується. Якщо вираз condition пропущено, то він вважається рівним true. В цьому випадку цикл продовжується до оператора break.

3. Вираз increment-expression використовується для зміни значення змінної лічильника.

4. Розраховується група операторів statements та реалізується перехід на наступну ітерацію циклу, тобто на крок 2.

*Приклад.* Цикл для розрахунку суми цілих чисел від 1 до 100.

```
s=0
for (i=1;i<101;i++) {
    s=s+1;
}
```

Оператор while повторює цикл, доки вказана умова істинна. Оператор while виглядає таким чином:

```
while (condition) {
```



## statements

}

Цикл `while` виконується таким чином. Спочатку перевіряється умова `condition`. Якщо умова істинна, то виконується група операторів `statements` в середині циклу. Перевірка істинності виконується на кожному кроці циклу. Якщо умова хибна, то цикл закінчує своє виконання.

Іноді необхідно закінчити цикл не по умові, що задана в його заголовку, а в результаті виконання деякої умови в тілі циклу. Для цього використовуються оператори `break` та `continue`. Оператор `break` завершує цикл `while` або `for` та передає керування програмою першому оператору після циклу. Оператор `continue` передає управління оператору перевірки істинності умови в циклі `while` та оператору оновлення значення лічильника в циклі `for` і продовжує виконання циклу.

Функція JavaScript – це іменована група команд, які вирішують певну задачу та можуть повернути деяке значення. Функція визначається за допомогою оператора `function`, що має такий синтаксис:

```
function Ім'я_функції (параметри)
{
  [оператори]
  return [значення_що повертається]
}
```

Параметри, що передаються функції, розділяються комами. Необов'язковий оператор `return` в тілі функції (блок операторів, що обмежений фігурними дужками), визначає значення, що повертається функцією. Визначення функції тільки задає її ім'я і визначає, що буде робити функція при її виклику. Безпосереднє виконання функції реалізується, коли в сценарії відбувається її виклик та передаються необхідні параметри. Відзначимо, що визначення функції необхідно реалізувати на HTML-сторінці до її виклику. Наприклад, для показу на екрані вікна повідомлення з текстом «Це виклик функції» визначимо функцію `Go` та реалізуємо її виклик:

```
<html><head><title>Використання JavaScript</title>
<script>
  function Go() {
    alert(«Це виклик функції»);
  }
</script>
</head><body>
<script>
  Go();
</script></body></html>
```

### 2.1.2. Об'єктна модель JavaScript

JavaScript відноситься до об'єктно-орієнтованих мов програмування. Об'єкт – це цілісна конструкція, що має властивості, які є змінними JavaScript та методи їх обробки. Властивості можуть бути іншими об'єктами. Функції, пов'язані з об'єктом, нази-

ваються методами об'єкта. Для звернення до властивостей об'єкту використовується наступний синтаксис:

**objectName.propertyName**

Ім'я об'єкта, імена властивостей та методів чуттєві до регістру. Для визначення властивостей їм необхідно присвоїти значення. Наприклад, якщо існує об'єкт з іменем `myCar`, то для визначення властивості `model` необхідно:

**myCar.model = «Таврія»**

Для визначення методів необхідно спочатку визначити звичайну функцію, а після цього, необхідно зв'язати цю функцію з існуючим об'єктом:

**object.methodname = function\_name**

де `object` – існуючий об'єкт, `methodname` – ім'я, що призначається методу, `function_name` – ім'я функції.

Виклик методу в контексті об'єкту реалізується так:

**object.methodname (params);**

Для створення екземпляру об'єкта необхідно:

- написати функції, які будуть використані в якості методів об'єкту;
- за допомогою звичайної функції визначити об'єкт;
- за допомогою оператора `new` створити екземпляр об'єкту.

Наприклад необхідно створити об'єкт з іменем `car` та властивостями `model` та `color` та методом `go`. Для цього необхідно написати функцію `when`, яка буде використана для визначення методу `go`:

```
function when() {
  //код функції
}
```

Після цього необхідно написати функцію для визначення об'єкта:

```
function car( model, color) {
  this.model = model;
  this.color = color;
  this.go =when;
}
```

Відзначимо, що оператор `this` використовується для того, щоб присвоїти значення властивостям об'єкту, базуючись на параметрах, що передаються функції.

Створення об'єкту з іменем `myCar` можливо реалізувати так:

**myCar = new car(«Таврія», «Зелений»)**

В JavaScript всі елементи (теги) на HTML-сторінці вибудовані в ієрархічну структуру. Причому кожен елемент представлений у вигляді об'єкту, з визначеними властивостями та методами. Керувати об'єктами на HTML-сторінці можна багато в чому за рахунок того, що JavaScript дозволяє одержати доступ до цих властивостей та методів. При реалізації доступу необхідно враховувати ієрархію об'єктів на HTML-сторінці. Відзначимо, що загальним об'єктом-контейнером є об'єкт `window`, який відповідає вікну браузера. В свою чергу цей об'єкт містить деякі елементи оформ-

лення, наприклад рядок стану. Завантажений у вікно браузера HTML-сторінці відповідає об’єкт document. Всі без виключення елементи HTML-сторінки є властивостями об’єкту document. Прикладами об’єктів HTML є таблиця, гіперпосилання або форма. Для доступу до методів/властивостей елементів на HTML-сторінці використовується наступний синтаксис:

`document.ім’я_об’єкту.ім’я_методу()`

Важливою ознакою інтерактивних HTML-сторінок є можливість реакції на дії користувача. Наприклад, натиснення на кнопки повинен викликати появу діалогового вікна, або виконання перевірки правильності введених користувачем даних. В JavaScript інтерактивність реалізована за допомогою перехвату та обробки подій, викликаних в результаті дій користувача. Для цього в теги деяких елементів введені параметри обробки подій. Ім’я параметру обробки події починається з префіксу on, за яким йде назва події. Наприклад, події клік кнопкою миші Click, відповідає параметр обробки події з назвою onClick. Назви та характеристики деяких подій наведені в табл. 2.1.2.1.

Таблиця 2.1.2.1

### Події JavaScript

Подія	Характеристика події	Обробник події
Click	Клік кнопкою миші на елементі форми або гіперпосилання	onClick
KeyDown	Натиск на клавіші клавіатури	onKeyDown
Load	Завантажується документ в браузер	onLoad
MouseDown	Натиснення на кнопки миші	onMouseDown
MouseOver	Курсор знаходиться над елементом	onMouseOver
MouseOut	Курсор залишає зону над елементом	onMouseOut

**Задача.** Необхідно, щоб при наведенні курсору на комірку таблиці із написом «Привіт» з’являлось вікно повідомлення з фразою «Hello». Можливі рішення:

*Варіант 1:*

`<td onClick=>alert(‘Hello’)>> Привіт </td>`

Варіант 2:

```
<script>
function Go() {
  alert(«Hello»)
}
</script>
<td onClick=>Go()>> Привіт </td>
```

У варіанті вирішення 1, код JavaScript був записаний безпосередньо в тезі, а в варіанті 2 наслідком кліку став виклик функції. Варіант 2 слід використовувати, якщо код обробки події великий за обсягом.

В ядрі JavaScript визначені об’єкти та функції, які можна використовувати не застосовуючи контекст завантаженої сторінки. До основних об’єктів відносяться: Array, Date, Math, String.

97

Array - масив. Масив це впорядкований набір однотипних даних, до елементів якого можна звернутись по імені або по індексу. Для створення масиву необхідно використати одну із двох конструкцій:

```
ім’я_масиву = new Array([елемент1], [елемент2], [елемент3],...)
ім’я_масиву = new Array([довжина масиву])
```

Відзначимо, що перший елемент масиву має номер 0.

В першій конструкції в якості параметрів використовуються елементи масиву, в другій конструкції використовується довжина масиву.

Наприклад:

```
ar1 = new Array(1, 2, 3)
ar2 = new Array(3)
```

Для доступу до значень елементів масиву в квадратних дужках біля імені масиву необхідно вказати порядковий номер елемента. Наприклад:

```
a = ar1[2]
ar1[0] = 7
```

В цьому прикладі, змінній a присвоюється значення елемента масиву за номером 2, а в елемент масиву за номером 0 записується значення 7.

Особливістю масивів JavaScript є те, що розмір масиву може встановлюватись динамічно. Наприклад, якщо для масиву із попереднього прикладу написати:

```
ar1[100] = 7,
```

то розмір масиву буде автоматично установлений рівним 101.

Для визначення довжини масиву можна скористатись властивістю length. Наприклад:

```
a = ar1. length
```

Зручність використання масивів забезпечується рядом методів, представлених в табл. 2.1.2.2.

Таблиця 2.1.2.2

### Методи об’єкту Array

Метод	Призначення
concat	Об’єднує два масиви в один
join	Об’єднує всі елементи масиву в один рядок
pop	Знищує останній елемент із масиву і повертає його значення
push	Додає один або декілька елементів в кінець масиву і повертає останній добавлений елемент
reverse	Переставляє елементи масиву в зворотному порядку: перший елемент стає останнім, а останній першим
shift	Знищує перший елемент масиву і повертає його значення
slice	Створює перетин масиву у вигляді нового масиву
splice	Додає та/або знищує елементи масиву
sort	Сортує елементи масиву
unsift	Додає один або більше елементів в початок масиву та повертає нову довжину масиву

Об’єкт Date використовується для роботи з датами. Синтаксис оператора створення екземпляра об’єкту дати:

```
ім'я_об'єкту_дати = new Date([параметри])
```

Якщо параметри відсутні, то значенням об’єкту буде поточна дата. Параметром може бути рядок типу: «місяць день, рік, час: хвилини;секунди».

Наприклад, для створення дати – «5 лютого 2012 року 23:12:07» необхідно:

```
day = new Date(«February 5, 2012 23:12:07»)
```

Прочитати або змінити параметри створеного об’єкту Date можна за допомогою ряду методів.

Найбільш вживані методи показані в табл. 2.1.2.3.

Таблиця 2.1.2.3

Методи об’єкту Date

Метод	Призначення
getDate	Повертає число місяця для вказаної дати
getDay	Повертає день тижня для вказаної дати
getHours	Повертає годину для вказаної дати
getMinutes	Повертає хвилини для вказаної дати
getMonth	Повертає місяць для вказаної дати
getSeconds	Повертає секунди для поточної дати
getFullYear	Повертає рік для вказаної дати
setDate	Встановлює число місяця для вказаної дати
setDay	Встановлює день тижня для вказаної дати
setHours	Встановлює годину для вказаної дати
setMinutes	Встановлює хвилини для вказаної дати
setMonth	Встановлює місяць для вказаної дати
setSeconds	Встановлює секунди для вказаної дати
setYear	Встановлює рік для вказаної дати

Об’єкт Math дозволяє використовувати вбудовані в JavaScript математичні функції та константи. При зверненні до методів та властивостей цього об’єкту створювати його не потрібно, але необхідно явно вказувати його ім’я.

Наприклад для того, щоб записати в змінну a результат розрахунку функції sin від 1 радіану необхідно:

```
a = Math.sin(1)
```

Для того, щоб записати в змінну a результат виразу 5 в степені 6 необхідно:

```
a = Math. pow(5,6)
```

Методи об’єкту Math, що використовуються найбільш часто представлені в табл. 2.1.2.4.

Таблиця 2.1.2.4

Методи об’єкту Math

Метод	Призначення
abs	Повертає абсолютне значення змінної
sin, cos, tan	Повертають значення тригонометричних функцій. Аргументи задаються в радіанах
acos, asin, atan	Повертають значення обернених тригонометричних функцій
exp, log	Повертають значення експоненціальної функції та функції натурального логарифму
ceil	Повертає найменше ціле число, більше або рівне значенню аргументу
floor	Повертає найменше ціле число, менше або рівне значенню аргументу
min, max	Повертає найбільше/найменше значення з двох аргументів
pow	Повертає значення функції: pow(x,y)=xy
round	Повертає значення аргументу, округлене до найближчого цілого числа
sqrt	Повертає квадратний корінь аргументу

Об’єкт String використовується для роботи з рядковими типами даних. Створення об’єкту String відбувається, коли змінній присвоюється рядковий літерал:

```
a = «Не явний спосіб створення рядкового об’єкту»
```

Крім того, можна явно створити рядковий об’єкт, використовуючи оператор new та конструктор String:

```
ім'я_об'єкту = new String(Рядок)
```

Параметром конструктору може бути будь-який рядок. Наприклад:

```
a=new String(«Явний спосіб створення рядкового об’єкту»)
```

Єдиною властивістю об’єкту String є length, що зберігає довжину рядка. Наприклад для запису в змінну h довжини рядка a необхідно:

```
h=a.length
```

Методи об’єкту String, що використовуються найбільш часто перераховані в табл. 2.1.2.5. Наведемо приклад використання методу toLowerCase для перевodu рядкової змінної a у верхній регістр:

```
a=a.toLowerCase();
```

Таблиця 2.1.2.5

Методи об’єкту String

Метод	Призначення
1	2
anchor	Створює HTML якір, який використовується, як гіпертекстове посилання
link	Створює гіпертекстове посилання, по якому можна перейти на інший URL
fontsize	Виводить рядок, зі встановленим розміром шрифту
bold	Виводить рядок, що відображається напівжирним шрифтом
italics	Виводить рядок, що відображається курсивом

Закінчення таблиці 2.1.2.5

1	2
strike	Виводить рядок, що відображається перекресленим шрифтом
substring	Повертає частину рядка об'єкта string
sub	Виводить рядок, що відображається як нижній індекс
sup	Виводить рядок, що відображається як верхній індекс
toLowerCase, toUpperCase	Переводить зміст рядка в нижній/верхній регістр

На додаток до стандартних об'єктів JavaScript існує декілька функцій, для виклику яких не потрібно створювати об'єктів. Ці функції дістали назву «функцій верхнього рівня». До цих функцій відносяться: `parseFloat(параметр)` та `parseInt(параметр)`. Їх призначенням є аналіз рядкового аргументу та повернення відповідно числа з плаваючою крапкою або цілого числа. Також досить часто використовуються функції `Number(об'єкт)` та `String(об'єкт)`, які перетворюють об'єкт, що використовується в якості параметру в число або рядок.

Об'єкт `window` створюється автоматично при запуску браузера. Крім того нове вікно можна створити і засобами JavaScript. Для цього необхідно використати метод `open`. Синтаксис методу такий:

```
Ім'я_змінної = window.open([Ім'я_файлу],[Ім'я_вікна],[Параметри])
```

Ім'я\_змінної – це ім'я для звернення до нового вікна в програмі JavaScript.

Ім'я\_файлу – це повна або відносна URL-адреса документу, що буде відкриватись у вікні браузера.

Ім'я\_вікна – це ім'я, що буде вказане в якості цілі в гіпертекстовому посиланні на це вікно із іншого HTML-документу.

Параметри – задають значення параметрів вікна. Основні параметри представлені в табл. 2.1.2.6. Якщо можливе значення властивості `yes` або `no`, то при стандартних настройках використовується значення `yes`.

Таблиця 2.1.2.6

Основні параметри вікна

Назва	Призначення	Можливі значення
directories	Наявність/відсутність панелі «Посилання»	yes/no
height	Висота вікна	кількість пікселів
location	Наявність/відсутність адресного рядка	yes/no
menubar	Наявність/відсутність рядка меню	yes/no
resizable	Можливість/не можливість зміни розмірів вікна користувачем	yes/no
scrollbars	Наявність/відсутність смуг прокрутки вікна	yes/no
status	Наявність/відсутність рядка стану браузера	yes/no
toolbar	Наявність/відсутність панелей інструментів	yes/no
width	Ширина вікна	кількість пікселів

Наприклад, для створення нового вікна браузера, в якому буде завантажено файл `a.html` необхідно:

```
<script>
myw=window.open(«a.html»,»displayWindow», «width=400,height=300,status=no,toolbar=no,menubar=no»)
</script>
```

При цьому, ширина вікна дорівнює 400 пікселів, висота вікна 300 пікселів, рядок стану вікна, панель інструментів та рядок меню будуть відсутні. Відзначимо, що наведений код необхідно записати в одному рядку.

Для закриття вікна браузера використовується метод `close`. Наприклад для закриття вікна попереднього прикладу необхідно:

```
myw.close()
```

Відзначимо, що при зверненні до методів та властивостей вікна браузера, в якому знаходиться програма JavaScript імені вікна або ключового слова `window` можна не вказувати. Наприклад, закрити поточне вікно браузеру можна так:

```
close()
```

Цікавим методом об'єкту `window` є метод `setTimeout`, за допомогою якого можна запрограмувати виконання деяких команд після закінчення встановленого терміну часу. Синтаксис методу такий:

```
setTimeout(«Код_JavaScript»,інтервал_часу)
```

В якості першого параметру функції `setTimeout`, як правило використовують функцію. Відзначимо, що цю функцію необхідно визначити на HTML-сторінці до використання функції `setTimeout`. Другим параметром функції `setTimeout` є інтервал часу закінчення якого буде сигналом про початок виконання команд JavaScript. Цей параметр задається в мілісекундах. Наприклад, виконання функції `myfunction` через 3000 мілісекунд після завантаження HTML-сторінки можна запрограмувати так:

```
setTimeout(«myfunction()»,3000)
```

Досить часто функція `setTimeout` використовується для створення анімаційних ефектів. Це може бути, наприклад, циклічна зміна кольору тексту, або циклічна зміна одного зображення іншим.

Об'єкт `document` містить інформацію про завантажену сторінку. Всі елементи HTML-сторінки є властивостями цього об'єкту. Найбільш використовуваним методом об'єкту `document` є метод `write`, за допомогою якого можливо зробити запис в HTML-сторінку. Наприклад, для запису рядка «Привіт JavaScript» в документ, у якому знаходиться сценарій необхідно занести:

```
document.write(«Привіт JavaScript»)
```

Ще одним широко вживаним методом цього об'єкту є `getElementById`, що реалізує доступ до будь-якого об'єкту з визначеним `id`. Синтаксис методу такий:

```
document.getElementById(ім'я_id)
```

Наприклад, встановлення значення стилю `display` елемента з `id=myid` рівним `block`, можна реалізувати так:

```
document.getElementById(«myid»).style.display=»block»
```

Властивості об'єкту location дозволяють одержати інформацію про URL-адресу завантаженої HTML-сторінки. Метод reload дозволяє перезавантажити в браузер поточну HTML-сторінку. Метод replace завантажує у вікно браузера сторінку, адреса якої використана в якості параметру цього методу.

### 2.1.3. Забезпечення ефективності WEB – сайтів

На сьогодні практично всі корпоративні інформаційні системи (KIC) досить тісно інтегровані з глобальною мережею Інтернет. При цьому представницьку роль KIC в мережі Інтернет, як правило, відіграє Web-сайт. Крім представницької ролі, сайт може забезпечувати: функціонування електронного магазину, доступ до розподіленої бази даних, рекламу корпорації в мережі Інтернет. Цей не повний перелік функцій сайту вказує на значний вплив ефективності його функціонування та ефективності функціонування всієї KIC, тим більше, що для його створення, розміщення в мережі і підтримки потрібні досить значні фінансові витрати.

Проблема ускладнюється тим, що на сьогодні не існує загальноприйнятого критерію оцінки ефективності сайту. Це пояснюється труднощами з визначенням прибутків корпорації завдяки наявності сайту. По цій причині рекомендується проводити оцінку ефективності сайту по кількості його відвідувачів. Така оцінка ефективності отримала і практичне визнання. Більш того, наявність популярного сайту дозволяє корпорації одержувати прибуток завдяки розміщенню на ньому банерної реклами. Наприклад, по матеріалам сайту www.rambler.ru, в 2011 році вартість розміщення в його верхній частині банерної реклами коштує близько 10000 доларів США. При цьому, кількість відвідувачів сайту становить більше 100000 на добу.

Забезпечити достатню кількість відвідувачів сайту можливо за рахунок:

- реалізації заходів спрямованих на те, щоб потенційні клієнти знали про сам факт існування сайту;
- заохоченні клієнтів, які повністю переглянули сайт при першому візиті та відвідувати його надалі.

Відзначимо, що потенційні клієнти можуть виявити сайт за допомогою реклами в традиційних джерелах інформації, або за допомогою Інтернет-орієнтованих шляхів виявлення сайту.

Досить поширеним методом реклами в традиційних джерелах інформації є рекламування самого сайту та його адреси в пресі та телебаченні. Хоча вказаний метод реклами досить ефективний, але потребує значних витрат на реалізацію, що не завжди доцільно.

В загальному випадку, виходячи з можливостей Інтернет - орієнтованих шляхів виявлення сайту потенційними користувачами, методика просування сайту може містити наступні заходи: використання пошукових систем, каталогів та рейтингів, банерний обмін, застосування систем активної реклами, використання дошок оголошень, поштових розсилок, форумів та чатів.

Відомо, що пошукові системи, каталоги та рейтинги залучають від 20% до 60% відвідувачів сайту. При пошуку сайтів користувач вводить ключові слова, що, на його думку, найкраще відображають тему цікавлячого сайту. В деяких випадках користувач може вказати загальну тематику пошуку. Результатом роботи пошукової системи

є список адрес сайтів, в яких зустрічаються ключові слова. Крім адреси в списку відображаються заголовки та опис сайту. Зазначимо, що список адрес формується в певній ієрархії та може відображатись на декількох сторінках. Принциповим моментом є місце адреси (рейтинг) сайту в цій ієрархії. Відомо, якщо адреса сайту знаходиться не на першій сторінці, то ймовірність його відвідування практично дорівнює нулю. Для визначення методів підвищення рейтингу сайту розглянемо механізми пошуку по ключовим словам та формування рейтингу. На різних типах пошукових систем вказані механізми дещо відрізняються і, як правило, офіційно не документуються. Але скрізь пошук реалізований на основі: ключових слів та опису сайту, що заносяться в базу даних при реєстрації сайту в пошуковій системі, ключових слів у тілі сайту або ключових слів та опису сайту в його заголовку в спеціальних тегах <meta>. Тому при реєстрації сайту в пошукових системах слід вказати ключові слова, що найбільш характерні для теми сайту, а також зробити відповідний опис сайту. В описі необхідно також застосувати ключові слова. Використання механізму пошуку на базі ключових слів та опису сайту в його заголовку можна за рахунок наступної програмної конструкції:

```
<head><title>придуманий вами заголовок</title>
```

```
<meta http-equiv=»KEYWORDS» content=»список ключових слів через коми»>
```

```
<meta http-equiv=»DESCRIPTION» content=»опис сторінки з декількох фраз, з використанням ключових слів»></head>
```

Відзначимо, що ключові слова та опис сайту при реєстрації та в його заголовку повинні співпадати. Як правило пошукові системи вимагають, щоб ключові слова та опис сайту не перевищували 200 знаків. Слід звернути увагу на зміст заголовка сайту, що міститься між тегами <title>. Саме він відображається в результатах запитів на першому місці і має першорядне значення в справі залучення відвідувачів. Довжина заголовка сайту не повинна перевищувати 75 знаків. Теги <meta> не видні при перегляді документа, але помітно збільшують його розмір, тому їх додаткове застосування (наприклад <meta name =»title» content=»Ключові слова, словосполучення»>) недоцільне. Для застосування ключових слів в тілі сайту необхідно на головній сторінці в перших 30-50 словах сформулювати основну тематику. Для збільшення кількості таких ключових слів можна записати їх на початку тіла сайту в тегах коментарів або (та) кольором фону. Ці ключові слова не будуть відображені на сайті, але можливо будуть використані пошуковою системою. Окремим питанням стоїть застосування ключових слів, які по статистиці є взагалі найбільш використовуваними при пошуку. На наш погляд недоцільно використовувати ключові слова пов'язані з розвагами, як з етичних позицій, так із позицій використання не цільової аудиторії. Хоча використання комп'ютерної, спортивної, радіотехнічної та автомобільної лексики доцільне. В тих пошукових системах, які передбачають процес реєстрації необхідно внести сайт в максимальну кількість розділів. Слід зазначити, що на сьогодні існує досить велика кількість пошукових систем. Реєстрація сайту в кожній із них потребує певних матеріальних витрат. Тому, на наш погляд доцільно провести реєстрацію сайту тільки в загальнопопулярних пошукових системах. До найбільш популярних пошукових систем мож-

на віднести – [www.meta.ua](http://www.meta.ua), [www.rambler.ru](http://www.rambler.ru), [www.aport.ru](http://www.aport.ru), [www.yandex.ru](http://www.yandex.ru), [www.google.com.ua](http://www.google.com.ua). Також, слід зареєструвати сайт в пошукових системах, якими користуються спеціалісти в галузі діяльності сайту. Відзначимо, що декілька років тому були розроблені спеціальні програми – павуки, призначенням яких є автоматична реєстрація сайту у великій кількості пошукових систем. Проте, в багатьох сучасних пошукових системах вмонтовані засоби, що не дозволяють проводити автоматичну реєстрацію, тому користуватись програмами-павуками не доцільно. Для участі в рейтингу відвідувачів на сторінках сайту необхідно розмістити банер (лічильник відвідувачів) пошукової системи. Наявність цих банерів вповільнює завантаження сайту, тому слід їх розміщувати тільки на вузлових сторінках. Деякі пошукові системи не передбачають механізм реєстрації сайту його адміністратором. В таких пошукових системах використовується спеціальна програма-робот, яка автоматично індексує сайти. Перешкодою для роботи такої програми може бути заборона на індексацію, встановлена на Web-сервері. Рекомендується цю заборону зняти, хоча це дещо зменшує безпеку сервера.

Ще одним, загальновідомим, Інтернет-орієнтованим шляхом рекламування сайту та адреси сайту є використання банерів. Банер представляє із себе текстову або графічну рекламу сайту, розміщену на іншому сайті. Використання банерів доцільне в тому випадку, коли сайт, носій банеру відвідується тими клієнтами, які є цільовою аудиторією сайту, що рекламується. Наприклад, ефективність розміщення на сайті фірми продавця автомобілів банеру фірми продавця автозапчастин може бути досить високою. Основним недоліком банерної реклами є те, що по статистиці, кількість відвідувачів, що знайшли сайт за допомогою банерної реклами становить близько 5% від загальної кількості відвідувачів. При цьому, реалізація банерної реклами можлива або шляхом обміну банерами, або потребує матеріальних витрат для публікації власних банерів на інших сайтах. Крім того, шлях обміну банерами досить часто не доцільний з точки зору зміни дизайну сайту та появи на ньому не тематичної інформації. Таким чином, в загальному випадку, використання банерної реклами пов'язане з певними труднощами та принципово не в змозі стрімко підвищити ефективність функціонування сайту.

Наступним можливим шляхом збільшення кількості відвідувачів може стати розміщення на сайті цікавої динамічної інформації, наприклад, чату, новин, розкладу занять для сайту учбового закладу. Хоча здійснення цих заходів по технічним причинам досить складне, але їх реалізація досить перспективна. Так відвідуваність сайту [www.gala.net](http://www.gala.net) можна пояснити саме розміщенням на ньому досить популярного чату. Ще одним, вже популярним шляхом збільшення кількості відвідувачів є застосування автоматизованої системи розсилки електронної пошти, яка рекламує діяльність корпорації.

Після того, як клієнт знайшов та вибрав адресу сайту необхідно, щоб він дочекався відображення домашньої сторінки сайту, повністю переглянув інші сторінки, а згодом знову відвідав сайт. Для цього слід продумати змістову частину сторінок сайту, створити дизайн, що відповідає направленості сайту та знищити технічні причини внаслідок яких користувач може відмовитись від перегляду сай-

ту. Найважливішою причиною є величина проміжку часу, за який сайт буде завантажуватись.

Проведемо аналіз проміжку часу, який проходить від подачі клієнтом запиту на перегляд сайту до його відображення на екрані. Будемо мати на увазі, що сайт переглядається клієнтом при стандартних настройках браузера, а також, що клієнту для перегляду необхідно використати мережу WWW.

Вказаний проміжок часу можна розділити на проміжок часу для відображення основного та допоміжних файлів Web-сторінки. Як правило, в якості основного виступає HTML-файл. В якості допоміжних виступають графічні, java (class) файли та об'єктні файли, до яких відносяться ActiveX та Flash. Важливим є той факт, що браузер спочатку робить спробу знайти означені файли в тимчасовій пам'яті і тільки в разі негативного результату робить запит в мережу WWW. Відзначимо, що негативним вважається результат, коли файли не знайдені, або їх параметри не співпадають з настройками браузера.

Час, необхідний на відображення основного та кожного з допоміжних файлів витрачається на втрати в мережі, виконання серверної частини Web-сторінки, інтерпретацію браузером та на відображення на екрані.

Втрати в мережі складаються з проміжків часу на встановлення з'єднання з сервером провайдера, встановлення з'єднання сервера провайдера з сервером, що обслуговує сайт, послідовну передачу основного файлу на сервер провайдера та на комп'ютер клієнта. Для їх зменшення можна застосувати організаційні та програмні заходи. Організаційні заходи полягають в розміщенні сайту на одному потужному сервері, який забезпечений потужними мережевими зв'язками. Таким вимогам відповідають тільки сервери спеціалізованих організацій – провайдерів.

В Україні загальноприйнятою є схема, коли організації, що мають власну локальну мережу, а відповідно і великий трафік та з'єднані з мережею WWW малопотужним виділеним каналом, розміщують Web-сервер на своїй території. Перевагами такої схеми є відносна дешевизна, дещо простіше керування, використання практично будь-якого необхідного програмного забезпечення, відсутність виходу в мережу WWW при перегляді власного сайту з локальної мережі організації. Але така схема має серйозний недолік. У випадку великого трафіку сайт стає практично недоступним для зовнішніх клієнтів. Крім цього, обслуговування Web-серверу потребує залучення висококваліфікованого технічного персоналу. Тому, кращою є дзеркальна схема, коли сайт розміщують на сервері спеціалізованої організації – провайдера, а його дублікат (дзеркало) розміщують на сервері, який знаходиться в середині локальної мережі. Головними перевагами цієї схеми є незалежність доступності сайту від трафіку організації та відсутність виходу в мережу WWW при перегляді власного сайту.

Основним програмним засобом зменшення втрат часу в мережі є зменшення загального обсягу сторінок сайту. Можна використовувати два напрямки.

Перший напрямок полягає в зменшенні обсягу кожної із сторінок за рахунок оптимізації графіки та HTML-коду сторінки. Оптимізація графіки полягає в мінімізації кількості графічного матеріалу, збереженні графічних файлів з достатньою якістю та розмірами зображення, можливому використанні одного малюнка з різним масштабом. Мінімізацію кількості графічного матеріалу можна проводити шляхом відмови



від графічних меню, надписів та фонів. В якості альтернативи можливо використовувати стилі. Графічні файли слід зберігати з безпечною кольоровою гамою, розмірами, які будуть використані на сайті та якістю, достатньою для відображення на екрані комп'ютера. Зауважимо, що для публікації в WWW рекомендується виготовляти фотокартки з мінімальною кольоровою гамою з чіткими переходами кольорів.

Оптимізацію HTML-коду слід проводити шляхом врахування особливостей парних тегів, широкого застосування стилів та відмовою від використання візуальних HTML-редакторів. Особливості парних тегів полягає в тому, що багато з них не потребують закриваючого тегу, хоча це і є відступом від стандарту. При цьому такі теги коректно відображаються браузерами. Наприклад, можна не використовувати закриваючі (парні) теги рядка (`</tr>`) та комірки (`</td>`) таблиці, так як їх дія закінчується закриваючим тегом таблиці (`</table>`). Застосування стилів доцільне при складному повторюваному форматуванні однотипних елементів сайту. В цьому випадку необхідні параметри тегів слід визначити за допомогою стилів, винесених в заголовок сторінки або в окремий css-файл. Відмова від використання візуальних редакторів пояснюється великим обсягом лишнього HTML-коду, який вставляється ними на сайт. Можна рекомендувати використання редакторів типу HomeSite або Microsoft Visual Studio, які дозволяють будувати оптимальний код сайту.

Іншим напрямком зменшення загального обсягу сторінок сайту є виділення спільної для сайту інформації в окремі файли (файл). При першому відвідуванні сайту ці файли завантажуються в тимчасову пам'ять комп'ютера клієнта (кеш). При відвідуванні інших сторінок сайту з мережі завантажуються тільки файли з новою інформацією. Його загальновідома реалізація є фреймова структура. Проте більшість комерційних організацій відмовляються від фреймів по причинах трудомісткості їх модифікації та складності реєстрації з пошуковими системами. Відзначимо, що в багатьох пошукових системах дуже складно зробити посилання на сторінки сайту з фреймовою структурою. Тому рекомендується використати пов'язаний з HTML-сторінками сайту js-файл. В цьому випадку, загальну для більшості сторінок сайту інформацію можна записати у вигляді винесених в окремий файл функцій мови програмування JavaScript. Таку структуру сайту називають динамічною. Як показує досвід, js-файл коректно кешується загальнопоширеними браузерами, а час завантаження неосновних сторінок такого сайту практично не відрізняється від сайту з фреймами. Модифікація статичної частини сайту з динамічною структурою полягає в зміні тільки одного js-файлу. Таким чином, до переваг сайту з динамічною структурою слід віднести швидкість завантаження, низьку трудомісткість модифікації та достатню адаптованість до пошукових систем.

Для зменшення втрат часу, що пов'язані зі встановленням з'єднання браузера клієнта з сервером, на якому розміщений сайт рекомендується обмежити числом 12 кількість файлів, з яких складається Web-сторінка. Така кількість файлів обумовлена практичним досвідом, а також рекомендується в літературі. В протилежному випадку втрати часу на з'єднання непропорційно зростають, особливо при низькій якості зв'язку в мережі, що характерно для більшості телефонних ліній.

Ще один шлях для зменшення втрат часу можна застосувати у випадку наявності на сайті декількох невеликих за розмірами блоків інформації, що потребують логічно-

го розподілу на декілька сторінок, доступ до яких необхідно проводити за допомогою різних пунктів меню. В цьому випадку можливо об'єднати ці блоки інформації в одну сторінку, розділену на частини за допомогою внутрішніх посилань (якорів), або за допомогою декількох таблиць з різними значеннями параметра id та стилем display, що дорівнює none. Стиль display відповідає за відображення елемента на екрані. Доступ до необхідних блоків інформації можна проводити за допомогою внутрішніх посилань, або присвоюючи необхідній таблиці стиль display, що дорівнює block. Вказати таблицю можна за допомогою її ідентифікатора – id.

Для зменшення втрат часу на виконання серверної частини сайту рекомендується будувати сайт з використанням тільки клієнтських технологій програмування. Особливо це стосується основної сторінки сайту, яка завантажується першою. В багатьох випадках застосування на першій сторінці серверних технологій пояснюється реєстрацією відвідувачів сайту, але така інформація фіксується Web-сервером і є доступною для Web-адміністратора. В якості ще однієї альтернативи можна запропонувати зовнішній для сайту лічильник відвідувань. Відзначимо, що широке застосування серверних технологій призводить до зменшення рівня безпеки Web-сервера з точки зору реалізації атаки на відмову.

Розглянемо втрати часу на інтерпретацію браузером сторінки сайту та відображення її на екрані. По специфікації інтерпретація сторінки проводиться браузером послідовно зверху вниз. Але для найбільш розповсюдженого браузера Microsoft Internet Explorer послідовна інтерпретація та відображення мають місце тільки для тегів HTML. У випадку використання на сторінці скриптових мов програмування, аплетів або Flash об'єктів вказана послідовність порушується. Це яскраво видно при перегляді сторінок сайтів у верхній частині яких є анімація, що в багатьох випадках реалізовано за допомогою викликів рекурсивних функцій. В таких випадках браузер не утворює новий обчислювальний процес. Анімаційні обчислення проводяться в тому ж потоці обчислень, в якому проходить завантаження сторінки. Відбувається різке навантаження на центральний процесор комп'ютера. Як показує досвід при розміщенні у верхній частині сторінки трьох анімаційних об'єктів, час її відображення на екрані може зрости в 2-5 рази. Положення ускладнюється тим, що для браузера Microsoft Internet Explorer почати запуск анімації при здійсненні події "Завершення завантаження сторінки" не вдається. По вказаним причинам рекомендується обмежити кількість анімаційних ефектів на сторінці двома, затримати виконання анімації на час достатній для завантаження всієї сторінки та проводити анімацію в окремих обчислювальних потоках. Наприклад, затримати час виконання анімації можна за рахунок вставки в анімаційну функцію animhead наступного програмного коду:

```
time=0;
if (time==0) setTimeout('anim()',7000);
else setTimeout('anim()',100);
time=1;
```

Вказаний програмний код написаний на мові JavaScript затримує виконання анімації на 7 секунд, при виконанні анімації через 0,1 секунди.

### 2.1.4. Адаптація веб-сайту до клієнтського програмного забезпечення

Адаптація сайту до клієнтського програмного забезпечення означає адаптацію до параметрів екрану комп'ютера, типу та настройок браузера і операційної системи користувача.

Для проведення адаптації сайту до характеристик екрану необхідно в першу чергу його пристосувати до перегляду на екранах з різною кольоровою гамою та різною роздільною здатністю. Хоча сучасні екрани, як правило, здатні передавати більше 1500000 кольорів, але відображатись ці кольори можуть по-різному. По цій причині рекомендується на сайті використовувати так звані «безпечні кольори». Кількість цих кольорів 256 і вони гарантовано однаково відображаються на всіх екранах.

Адаптацію сайту до роздільної здатності екрану провести складно. Погано адаптований сайт або займає лівий верхній кут екрану, залишаючи справа і знизу пусті місця, або далеко виходить за межі екрану. По цим причинам страждає дизайн сайту, крім того ускладнюється навігація по сайту. Найпростішими методами адаптації є оформлення сторінки сайту у вигляді таблиці, вирівняної по центру сторінки. У випадку використання в заголовку малюнку має сенс розділити його на дві частини. Першу частину, яка містить інформаційну частину малюнку вставити за допомогою тегу `<img>` в рядок або комірку таблиці. Іншу, фонову частину вставити як фоновий малюнок рядка або комірки таблиці. Це дозволить автоматично розмножувати фонову частину малюнку в залежності від ширини сторінки. Відзначимо, що інформаційна частина малюнку повинна коректно відображатись на невеликих екранах, для цього її ширина повинна бути в межах 100 – 200 пікселів. Ширина фонові частини малюнка повинна бути в межах 15-25 пікселів. Таким чином заголовок сайту буде коректно виглядати на всіх екранах, а сумарний розмір його графічних файлів буде становити 15 – 20 кілобайт. Пристосованість текстової частини сайту до роздільної здатності екрану можна здійснити шляхом встановлення коефіцієнта пропорційності між розміром шрифту та роздільною здатністю екрану.

Для коректного відображення сайту різними браузерами слід використовувати при його розробці тільки стандартні теги HTML, а також не використовуючи ті теги, які не сприймаються ведучими браузерами. Так, наприклад не рекомендується використовувати стандартний тег шару `<div>`, який некоректно сприймається браузером Netscape Navigator.

Адаптацію сторінки до настройок браузера можна проводити шляхом врахування розмірів вікна браузера та встановлених клієнтом розмірів тексту. Погана адаптація до вказаних настройок може негативно позначитися на дизайні сторінок сайту, на яких важливим є взаємне розміщення графічного та текстового матеріалу. Негативний вплив відбувається завдяки переходу малюнків в не передбачені для них рядки. Щоб цього не сталось, можна за допомогою тегу `<nobr>` заборонити розрив найдовшого рядка на сторінці, або взагалі заборонити зміну користувачем розмірів вікна браузера. В такому випадку рекомендується відкрити вікно браузера на весь екран. Це можна зробити обчисливши ширину (w) та висоту (h) екрану. Після цього слід сполучити ліві верхні кути вікна браузера та екрану і перерисувати вікно браузера

відповідно до розмірів екрану. Відповідний програмний код на мові програмування JavaScript має вигляд:

```
h=window.screen.height; w=window.screen.width;
window.moveTo(0,0); window.resizeTo(w,h);
```

При необхідності заборони користувачем розмірів тексту сайту можна встановити цей розмір за допомогою стилів. Наприклад:

```
<font style=>font-size: 14px;>>приклад</font>
```

Розміри букв слова прикладу практично завжди будуть 14 пікселів.

### 2.1.5. Коротка характеристика VBScript

Visual Basic Scripting Edition (зазвичай просто VBScript) – сценарна мова програмування, що інтерпретується компонентом Windows Script Host. Вона широко використовується при створенні скриптів в операційних системах сімейства Microsoft Windows.

Мова була створена компанією Microsoft, як заміна застарілій пакетній мові, що інтерпретується застосуванням command.com. Синтаксис VBScript є спрощеною версією синтаксису мови Visual Basic.

Сценарії на мові VBScript найчастіше використовуються в наступних областях, що використовують програмні продукти Microsoft:

- автоматизація адміністрування систем Windows;
- серверний програмний код в сторінках ASP;
- клієнтські сценарії в браузері Internet Explorer.

### 2.1.6. Java-апплету

**Java-апплет** – це програма, написана на мові Java і відкомпільована в байт-код, який виконується в браузері з використанням віртуальної Java-машини (JVM). Аплети використовуються для надання інтерактивних можливостей веб-застосувань, які не можливі в HTML. Оскільки байт-код Java платформо-незалежний, то Java-аплети можуть виконуватися браузерами на багатьох операційних платформах.

Java-сервлети є серверними застосуваннями, але вони відрізняються від аплетів мовою, функціями і іншими характеристиками.

Java-аплети призначені для виконання в безпечному середовищі з метою запобігання їх доступу до локальних ресурсів клієнтського комп'ютера.

Код аплету завантажується з веб-серверу і браузер або вставляє аплет у веб-сторінку, або відкриває окреме вікно з власним призначенням для користувача інтерфейсом аплета.

Аплет може бути занесений у веб-сторінку за допомогою використання HTML тегу `<applet>`, або (що рекомендується) тегу `<object>`.

Переваги Java-аплетів такі:

- ⇒ працюють практично на більшості операційних платформах;
- ⇒ підтримуються більшістю браузерів;

- ⇒ *кешуються в більшості браузерів, що істотно прискорює їх завантаження при поверненні на веб-сторінку;*
- ⇒ *після першого запуску аплету, коли Java-машина вже виконується і швидко запускається, виконання аплетів відбувається істотно швидше;*
- ⇒ *завантажуються швидко, порівняно з програмами на інших компільованих мовах, наприклад C++, але у багато разів швидше чим на JavaScript.*

При цьому в Java-апплетів є і недоліки:

- ⇒ *потрібна установка Java-розширення, які доступні за замовчуванням не у всіх браузерах;*
- ⇒ *проблеми реалізації Java-розширень для 64-розрядних процесорів;*
- ⇒ *не можуть запускатися до першого завантаження віртуальної Java-машини, що може займати значний час;*
- ⇒ *розробка призначеного для користувача інтерфейсу з використанням аплетів є складнішим завданням в порівнянні з HTML;*
- ⇒ *не мають прямого доступу до локальних ресурсів клієнтського комп'ютеру;*
- ⇒ *деякі аплети прив'язані до використання певного середовища часу виконання Java (JRE).*

### 2.1.7. ActionScript, XAML і Microsoft Silverlight – загальна характеристика

**ActionScript** – об'єктно-орієнтована мова програмування, яка є одним з діалектів EcmaScript, який додає інтерактивність, обробку даних і багато іншого у вміст Flash-застосувань. ActionScript виконується віртуальною машиною (ActionScript Virtual Machine), яка є складовою частиною застосування Flash Player. ActionScript компілюється в байткод, який включається в SWF-файл.

SWF-файли виконуються Flash Player. Сам Flash Player існує у вигляді плагіну до веб-браузера, а також як самостійне виконання застосування. У другому випадку можливе створення виконуваних exe-файлів, коли swf-файл включається у Flash Player.

За допомогою ActionScript можна створювати інтерактивні мультимедіа-застосування, ігри, веб-сайти і багато іншого.

**XAML (eXtensible Application Markup Language)** – мова інтерфейсів платформи Windows Vista.

Модель застосувань Vista включає об'єкт Application. Його набір властивостей, методів і подій дозволяє об'єднувати веб-документи в зв'язане застосування. Об'єкт Application контролює виконання програми і генерує події для призначеного для користувача коду. Документи застосування створюються за допомогою мови XAML, яка описує, перш за все, призначений для користувача інтерфейс. Логіка застосування управляється процедурним кодом (C#, VB і ін.). XAML включає основні чотири категорії елементів: панелі, елементи управління, елементи, пов'язані з документом і графічні фігури.

Microsoft Silverlight є офіційною назвою заснованою на XML і .NET-технології під кодовим іменем WPF/E (Windows Presentation Foundation Everywhere), що є альтернативною Adobe Flash. Microsoft Silverlight є підмножиною Windows Presentation Foundation, в якій реалізовані векторна графіка, анімація і засоби відтворення відео. У версії 1.1 включено повну версію .NET CLR, яка називається CORECLR, що дозволяє розробляти Silverlight застосування на будь-якій з мов .NET. Silverlight містить модуль браузера, що підключається, для обробки XAML і кодеки для відтворення мультимедійного вмісту у форматах WMV, WMA і MP3.

Microsoft Silverlight представляє браузеру внутрішню модель DOM, керовану з JavaScript коду. Оскільки мова XAML заснована на XML, то документ, що визначає завантажуваний клієнтові призначений для користувача інтерфейс – текстовий і тому цілком придатний для індексування пошуковими системами. Використовуючи модель DOM JavaScript може динамічно оновлювати вміст Silverlight, аналогічно DHTML.

Також можна викликати методи управління презентацією (наприклад, запуску анімації або припинення відтворення відео).

Silverlight-застосування починається з виклику об'єкту Silverlight з HTML-сторінки, завантажуючого XAML файл. XAML файл містить об'єкт Canvas, який являється підміною для інших елементів. Об'єкти XAML здатні генерувати події, що перехоплюються з JavaScript.

### 2.1.8. Поняття про DOM та HTML DOM

DOM (Document Object Model) – об'єктна модель документу. Це незалежний від платформи і мови програмний інтерфейс, що дозволяє програмам діставати доступ до вмісту документів, а також змінювати вміст, структуру і вид документів.

У рамках DOM будь-який документ представляється у вигляді дерева вузлів. Кожним вузлом є елемент, атрибут, текстовий, графічний або будь-який інший об'єкт. Вузли між собою перебувають у відношенні «батько-нащадок».

Спочатку різні браузери мали власні моделі DOM, не сумісні з останніми. Для того, щоб забезпечити взаємну і зворотну сумісність, консорціум W3C класифікував цю модель по рівнях, для кожного з яких була створена своя специфікація. Всі ці специфікації об'єднані в загальну групу, що носить назву W3C DOM:

Рівень 0. Включає всі специфічні моделі DOM, які існували до появи Рівня 1, наприклад document.images, document.forms. Ці моделі формально не є специфікаціями DOM, опублікованими W3C, а швидше відображають те, що існувало до початку процесу стандартизації.

Рівень 1. Базові функціональні можливості DOM (HTML і XML) в документах, такі як отримання дерева вузлів документа, можливість змінювати і додавати дані.

Рівень 2. Підтримка простору імен XML, filtered views і подій.

Рівень 3. Складається з шести різних специфікацій:

- DOM Level 3 Core;
- DOM Level 3 Load and Save;
- DOM Level 3 XPath;
- DOM Level 3 Views and Formatting;

- Level 3 Requirements;
- DOM Level 3 Validation.

Поточним рівнем специфікацій DOM є Рівень 2, але, тим не менше, деякі частини специфікацій Рівня 3 є для W3C рекомендованими.

HTML DOM дозволяє читати і змінювати вміст HTML-елементів сторінки зі скриптів.

HTML DOM може використовуватися з багатьма мовами програмування, але найчастіше DOM використовують у зв'язці з JavaScript.

HTML DOM є W3C стандартом, тому всі сучасні браузері підтримують дану технологію.

Приклад використання HTML DOM:

```
x=prompt(«Введіть Ваше ім'я», «Ім'я»);
document.getElementById(«dialog»).style.visibility=»visible»;
document.getElementById(«name»).innerHTML=x;
document.getElementById(«header»).innerHTML=»Діалог»;
document.getElementById(«fw»).innerHTML=x;
document.getElementById(«sw»).innerHTML=x;
```

В результаті вміст сторінки змінюватиметься залежно від введенного ім'я.

**Об'єктна структура.** При відкритті будь-якого HTML документу браузер заздалегідь проводить розбір його вмісту і на основі цього розбору створює об'єктну модель HTML документу або коротше DOM.

DOM складається з вкладених один в одного у ієрархічному порядку об'єктів, які називаються вузлами. Кожен вузол в структурі представляє розташований на сторінці HTML елемент.

Використовуючи DOM можна взаємодіяти (прочитувати, змінювати, видаляти) з вмістом HTML документів зі скриптів.

Нижче розташовується код та рисунок HTML-документа і DOM, яка б була створена браузером на основі цього коду:

```
<html>
<head>
<title>HTML DOM</title>
</head>
<body>
<h1>HTML DOM.</h1>
<p style=»color:green»>Привіт усім.</p>
</body>
</html>
```

Всі прямокутники показані на рисунку є об'єктами (або вузлами). Різним кольором на зображенні відмічені вузли різного типу.

Червоним кольором відмічений вузол Document. Будь-яке звертання до DOM повинне починатися із звернення до даного вузла.

Зеленим кольором відмічені елементні вузли. Для кожного HTML елементу на сторінці браузер створює відповідний елементний вузол.

Вміст елементів зберігається в текстових вузлах. Текстові вузли відмічені на нашій схемі синім кольором.

Для кожного атрибуту HTML елементу створюється атрибутний вузол. Атрибутний вузол відмічений на схемі рожевим кольором.

Зверніть увагу: не можна забувати, що текст завжди зберігається в текстових вузлах і не є властивістю елементу. Тобто для того, щоб звернутися до вмісту HTML елементу треба звернутися до властивості його текстового вузла.

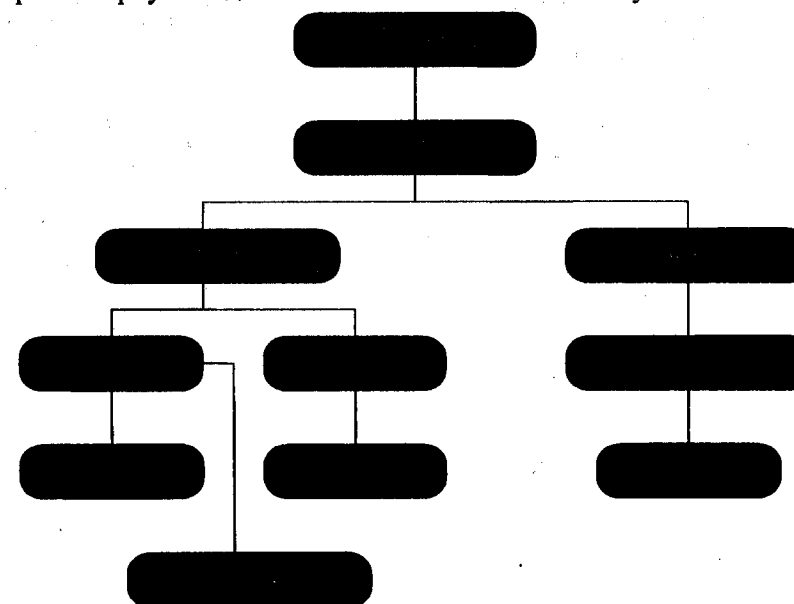


Рис. 2.1.8.1. Об'єктна структура HTML DOM.

**Звернення до елементів.** Умовно можна сказати, що звертатися до елементів в DOM можна двома різними способами:

- ⇒ використовувати послідовні переміщення по об'єктній структурі до знаходження необхідного елементу;
- ⇒ використовувати пряме звернення до елементу по його ідентифікатору або імені теги.

Другий спосіб безумовно простіше та зручніше і в повсякденній практиці завжди використовувати саме його. Проте в учбових цілях корисно розібрати обидва способи.

Послідовне переміщення. Повернемося до DOM і уявімо, що нам потрібно прочитати текстовий вміст її елементу p.

Стрілкою відмічено, як послідовно виглядатиме переміщення по об'єктній структурі.

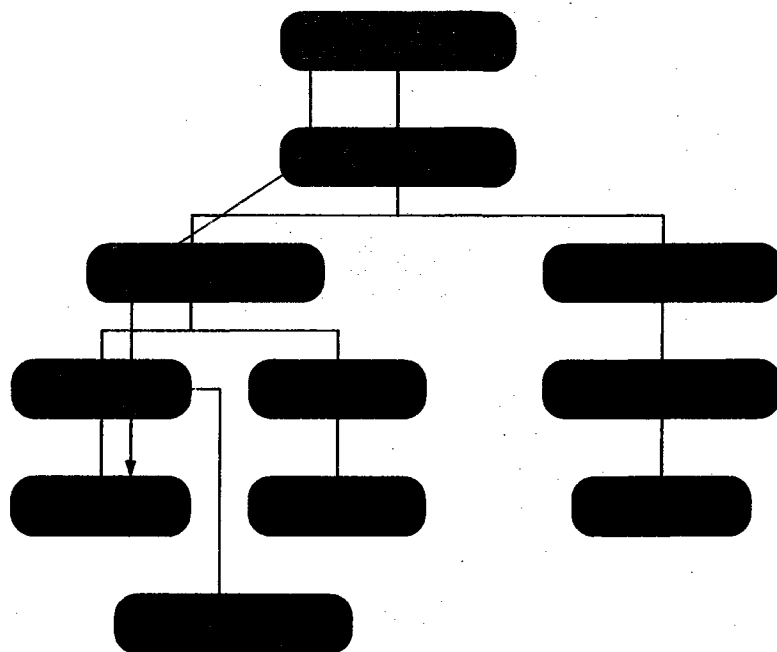


Рис. 2.1.8.2. Схема послідовного переміщення.

Розберемо процес переміщення докладніше:

1. Звертаємося до об'єкту `document`, в якому знаходиться DOM. Код на даному кроці має вигляд: `document`;
2. Звертаємося до кореневого вузла (тобто теги `<html>`), який знаходиться в середині об'єкту `document`. Код на даному кроці має вигляд: `document.documentElement`;
3. Звертаємося до другого нащадка (оскільки в коді сторінки `body` розташовується після `head`) кореневого вузла. Код на даному етапі матиме вигляд: `document.documentElement.childNodes[1]`. Також можна безпосередньо звернутися до `body` використовуючи наступний код (далі вважатимемо, що ми вибрали цей варіант): `document.body`;
4. Звертаємося до другого нащадка `body` (елемент `p` заданий в коді після `h1`). Код на даному етапі матиме вигляд: `document.body.childNodes[1]`;
5. Звертаємося до текстового вузла, який є першим нащадком `p` і дізнаємося значення його властивості. Код на даному етапі матиме вигляд: `document.body.childNodes[1].childNodes[0].nodeValue`.

//Виведемо вміст абзацу на сторінку

```
document.write(document.body.childNodes[1].childNodes[0].nodeValue);
```

Пряме звернення до елемента. За допомогою методу `getElementById` Ви можете безпосередньо звертатися до елементів по їх ідентифікатору (атрибут `id`), а за допомогою властивості `innerHTML` можна швидко прочитувати їх текстовий вміст.

//Виведемо вміст другого абзацу на сторінку

```
document.write(document.getElementById(<par>).innerHTML);
```

DHTML дозволяє сценарним мовам змінювати змінні мови опису представлення документа, таким чином, змінюючи вигляд і поведінку раніше статичного вмісту HTML-документа вже після повного завантаження документа і в процесі переглядати його користувачем. Таким чином, динамічність, що здійснюється з допомогою DHTML, проявляє себе в процесі переглядання сторінки, але не має жодного відношення до генерації вмісту сторінки при кожному її завантаженні.

У протилежність DHTML, сторінка, що динамічно генерується – ширше поняття, що має на увазі, наприклад, генерацію вмісту веб-сторінки індивідуально для кожного користувача. Це досягається створенням сторінок за допомогою клієнтських або серверних (наприклад, на PHP або Perl) сценаріїв.

Регулярні вирази – система пошуку текстових фрагментів в електронних документах, заснована на спеціальній системі запису зразків для пошуку. Зразок, що задає правило пошуку, називається «шаблоном». Вживання регулярних виразів принципово змінило технології електронної обробки текстів.

Багато мов програмування вже підтримують регулярні вирази для роботи з рядками або у вигляді окремих функцій, чи мають вже вбудований в них синтаксис механізм обробки регулярних виразів, наприклад, Perl і Tcl. Популяризації поняття регулярних виразів сприяли утиліти, що поставляються в дистрибутивах Unix.

За допомогою регулярних виразів можна задавати структуру шуканого шаблону і його позицію в середині рядка. При описі структури шаблону використовуються:

- гнучка система квантифікаторів (операторів повторення);
- оператори опису наборів символів і їх типу (числові, нечислові, спеціальні).

## ТЕМА 2.2. МОВИ РОЗРОБКИ СЦЕНАРІЇВ PHP, PERL, JSP

### 2.2.1. Мова розроблення сценаріїв PHP

### 2.2.2. Мова сценаріїв Perl

### 2.2.3. Мова розроблення сценаріїв JSP

#### 2.2.1. Мова розроблення сценаріїв PHP

PHP (англ. PHP: Hypertext Preprocessor – PHP: гіпертекстовий процесор), попередня назва: Personal Home Page Tools – скриптова мова програмування була створена для генерації HTML сторінок на стороні серверу. PHP є однією з найпоширеніших мов, що використовуються у сфері веб-розробок (разом із Java, .Net, Perl, Python, Ruby). PHP підтримується переважною більшістю хостинг-провайдерів PHP – проект відкритого програмного забезпечення.

PHP інтерпретується веб-сервером в HTML-код, який передається на сторону клієнта. На відміну від скриптової мови JavaScript, користувач не бачить PHP-коду, бо браузер отримує готовий html-код. Це є перевага з точки зору безпеки, але водночас погіршується інтерактивність сторінок. Але ніщо не забороняє використовувати PHP для генерування і JavaScript-кодів, які виконуються вже на стороні клієнта.

PHP – це мова, яка може бути вбудована безпосередньо в html-код сторінок, які коректно будуть оброблені PHP-інтерпретатором. Механізм PHP просто починає виконувати код після першої екрануючої послідовності (<?) і продовжує виконання до того моменту, коли він зустріне парну екрануючу послідовність (?>).

Особливості PHP:

- велика різноманітність функцій PHP дають можливість уникнути написання багаторядкових призначених для користувача функцій на C або Pascal;
- наявність інтерфейсів до багатьох баз даних:
  - ⇒ в PHP вбудовані бібліотеки для роботи з MySQL, PostgreSQL, Oracle, Informix, Interbase, Sybase;
  - ⇒ через стандарт відкритого інтерфейсу зв'язку з базами даних (Open Database Connectivity Standard – ODBC) можна підключатися до всіх баз даних, до яких існує драйвер;
- традиційність. Мова PHP здаватиметься знайомою програмістам, що працюють в різних областях. Багато конструкцій мови запозичені з C та Perl. Код PHP дуже схожий на той, який зустрічається в типових програмах на C або Pascal. Це помітно знижує початкові зусилля при вивченні PHP. PHP – мова, що поєднує переваги Perl і C та спеціально спрямована на роботу в Інтернеті, мова з універсальним й зрозумілим синтаксисом. І хоча PHP є досить молодою мовою, вона здобула таку популярність серед web-програмістів, що в наш час є мало не найпопулярнішою мовою для створення веб-застосунків (скриптів).
- наявність вихідного коду та безплатність;
- стратегія Open Source і розповсюдження початкових текстів програм в масах справили позитивний вплив на багато проектів, в першу чергу – Linux хоч і

успіх проекту Apache сильно підкріпив позиції прихильників Open Source. Сказане відноситься і до історії створення PHP, оскільки підтримка користувачів зі всього світу виявилася дуже важливим чинником в розвитку проекту PHP;

- ухвалення стратегії Open Source і безплатне розповсюдження початкових текстів PHP надало неоціниму послугу користувачам. Додатково, користувачі PHP в усьому світі є свого роду колективною службою підтримки, і в популярних електронних конференціях можна знайти відповіді навіть на найскладніші питання;
- ефективність є дуже важливим чинником при програмуванні для середовищ розрахованих на багато користувачів, до яких належить і web. Важливою перевагою PHP є те, що ця мова належить до інтерпретованих. Це дозволяє обробляти сценарії з достатньо високою швидкістю. За деякими оцінками, більшість PHP-сценаріїв (особливо не дуже великих розмірів) обробляються швидше за аналогічні їм програми, написані на Perl. Проте, щоб не робили розробники PHP, виконувані файли, отримані за допомогою компіляції, працюватимуть значно швидше – в десятки, а іноді і в сотні разів. Але продуктивність PHP цілком достатня для створення цілком серйозних веб-застосунків.

Історія PHP починається з 1995 року, коли Расмус Лерддорф (Rasmus Lerdorf) створив просте застосування мовою Perl, що аналізувало відвідування користувачами його резюме на веб-сайті. Потім, коли цим застосуванням вже користувалися кілька чоловік, а число охочих одержати його постійно збільшувалося, Лерддорф назвав своє творіння Personal Home Page Tools версія 1 і виставив для вільного завантаження. З цієї миті почався небувалий зліт популярності PHP.

Як це завжди буває, терміново було потрібне доопрацювання і нові доповнення. Для їхньої реалізації Расмус створює нову версію пакету, тепер уже написану на C. Отриманий таким чином інструмент набуває робочої назви PHP/FI (Personal Home Page/Forms Interpreter – Персональна Домашня сторінка/Інтерпретатор Форм), надалі він також буде відомий під назвою PHP 2. Ця версія вже більшою мірою схожа на сьогоденний PHP. Вона мала синтаксис і спосіб іменування змінних в стилі мови Perl, можливість вбудовування PHP операторів в html-код сторінки, автоматичну інтерпретацію форм, інтеграцію з базами даних. При цьому все працювало досить швидко, оскільки PHP прикомпілювалось до веб-серверу Apache. До 1997 року PHP використовувався вже на 50000 доменах (не більше 1% всіх веб-серверів).

У тому ж 1997 році до проекту PHP підключилися Зев Сураскі (Zeev Suraski) і Енді Гутманс (Andi Gutmans). Ці студенти Техніону, одного з ізраїльських університетів, намагалися використовувати PHP/FI для одного з комерційних університетських проектів. При цьому їм довелося зіткнутися з багатьма труднощами і обмеженнями цієї технології. Вивчаючи початковий код PHP 2, Зев і Енді дійшли висновку про необхідність доопрацювання, а точніше істотної переробки PHP, особливо в плані синтаксису мови. Протягом декількох місяців вони блискуче впоралися з цим завданням.

Закінчивши роботу Зев і Енді домовились з Расмусом про співпрацю в галузі розвитку та вдосконалення мови. З цієї миті з'являється PHP Group – група однодумців,



що працюють над розвитком технології PHP. Одержаний продукт з'явився на світ у 1998 році під назвою PHP 3.

При цьому головною особливістю PHP 3 була можливість розширення ядра, що привернуло до роботи над PHP безліч сторонніх розробників, що створюють спеціалізовані модулі. Їх наявність дала PHP можливість працювати з величезною кількістю баз даних, протоколів, підтримувати велике число API. До кінця 1998 кількість користувачів PHP перевищила за 100000, а PHP був вже встановлений на не менше ніж 10% серверах Інтернету. У той же час значному поширенню даної мови сприяли публікації в електронній пресі та вихід книжок для вивчення PHP.

Відразу ж після виходу PHP 3, Енді Гутманс і Зев Сураскі почали переробку ядра PHP. В першу чергу належало вирішити проблему підвищення продуктивності. Новий продукт, названий Zend Engine (від імен творців: Zeev і Andi), успішно справлявся з поставленим завданням і був реалізований в 2009 році. Основними реалізованими ідеями є можливість компіляції сценарію у виконуваний модуль, за рахунок чого продуктивність можна було підняти на порядок.

PHP 4, що працює на цьому ядрі, вийшов в 2010 році. На додаток до збільшення продуктивності PHP 4 мав нові можливості щодо підтримки сесій, буферизацію виводу, безпечні способи обробки інформації, що вводиться користувачем, і нові мовні конструкції. З виходом 4 версії PHP став використовуватися вже на більш ніж 20% доменів Інтернету.

За час з 2004 по 2010 рік продовжувалися активні роботи з покращення 4 версії, але майже відразу PHP Group приступила до продумування можливостей нової версії. В першу чергу було вирішено підсилити об'єктні можливості мови, що дозволяло використовувати його для реалізації масштабних проєктів. Роботи із створення версії 5 велися тривалий час, в них брало участь рекордна кількість фахівців, зокрема Стерлінг Хьюз (Sterling Hughes) і Маркус Бергера (Marcus Boerger).

У липні 2004 року виходить офіційний реліз PHP 5. В першу чергу, як і планувалося, було перероблено весь механізм роботи з об'єктами. І якщо в попередніх версіях об'єктно-орієнтоване програмування на PHP було можливе в мінімальному ступені, а тому і використовувалося на практиці не часто, то PHP 5 володіє прекрасним потенціалом реалізації об'єктного програмування. Окрім цього, PHP збагатився рядом цінних розширень для роботи з XML, різними джерелами даних, генерації графіки та інше.

Серед інших украй корисних доповнень в PHP 5 слід зазначити нову схему обробки виключень. Конструкція try/catch/throw дозволяє весь код обробки помилок локалізувати в одному місці сценарію.

Всі основні бібліотеки для роботи з XML, запозичені в PHP 4, були піддані серйозній переробці. Такі популярні розширення, як SAX, DOM і XSLT, тепер використовують інструмент libxml2, що робить їх ще ефективнішими.

У PHP 5 також включені два нові модулі для роботи з протоколами – SimpleXML і SOAP. SimpleXML дозволяє значно спростити роботу з XML-даними, представляючи вміст XML-документа у вигляді PHP-об'єкта. Розширення SOAP дозволяє будувати на PHP сценарії, що обмінюються інформацією з іншими застосуваннями за допомогою XML-повідомлень поверх існуючих веб-протоколів, наприклад HTTP. Модуль

для роботи з SOAP для PHP 5 надає розробникам засіб для достатньо швидкого створення ефективних SOAP-клієнтів і SOAP-серверів.

Новий модуль PHP 5 MySQLi (MySQL Improved) призначений для роботи з MySQL-сервером версій 4.1.2 і вище, реалізуючи не тільки процедурний, але і об'єктно-орієнтований інтерфейс до MySQL. Додаткові можливості цього модуля включають – SSL, контроль транзакцій, підтримка реплікації і ін. Очевидно, що, на цьому історія PHP не закінчується. Слід очікувати наступних версій мови із розширеними можливостями.

Всі сценарії оформляються у вигляді блоків коду. Ці блоки можуть бути поміщені в HTML-код, але відділені від нього відповідними обмежувачами. Код PHP в HTML повинен знаходитись між початковим тегом `<?php` та кінцевим `?` (або між `<script language=»php»` та `</script>`) Бажаним варіантом виділення PHP-коду є варіант `<?php ?>`, оскільки саме такі початковий та кінцевий теги дозволять використовувати PHP-код в документах, які відповідають правилам XML. Також можна користуватися скороченим записом: `<? ?>` (інколи потрібно активізувати даний стиль внісши вручну зміни у файл `php.ini`: змінна `short_open_tag` повинна мати значення `On`) і записом в стилі ASP: `<% %>` (в `php.ini` змінна `asp_tags` повинна мати значення `On`). Проте стиль ASP не рекомендується і очікується, що він буде відсутній у PHP6.

Найпростіша програма Hello world на PHP виглядає так:

```
<?php
    echo 'Hello, world!';
?>
```

PHP виконує код, що знаходиться в середині обмежувачів, таких як `<?php ?>`. Все, що знаходиться поза межами обмежувачів виводиться без змін. Таким чином виконується вставка PHP коду в HTML код. Наприклад, код html-сторінки з попереднім прикладом виглядатиме так:

```
<html>
<head>
    <title>Тестуємо PHP</title>
</head>
<body>
    <?php echo 'Hello, world!'; ?>
</body>
</html>
```

Інструкції в PHP відокремлюються символом `;`. Перед закінченням скрипту (перед тегом `?`) крапку з комою ставити необов'язково.

Коментарі в PHP можуть бути як однорядкові так і багаторядкові. Однорядкові коментарі починаються із `//` або `#` (коментар скриптових мов UNIX). Багаторядкові коментарі починаються символами `/*` і закінчуються `*/`. Слід зазначити, що однорядкові коментарі йдуть або до кінця поточного рядка, або до завершального тега `?`.

Основою будь-якого програмування є змінні. PHP, як і деякі інші Unix-скриптові мови, не потребує їх опису. Існує три стилі оформлення змінних у PHP:

➤ короткий стиль.

Змінні записуються у вигляді: `$variable`. Використовується в більшості випадків. Змінна, яка створюється програмним кодом, записується таким чином. Можливо також використання такого стилю для добування змінної із ФОРМИ, якщо ввімкнено `register_globals` у файлі конфігурації `php.ini`;

➤ середній стиль: `$_POST['variable'], $_GET['variable'], $_REQUEST['variable']`.

Використовується для отримання даних з полів ФОРМИ. В залежності від типу передачі даних, встановлюється POST або GET, або REQUEST для обох випадків;

➤ довгий стиль: `$HTTP_POST_VARS['variable'], $HTTP_GET_VARS['variable']`.

Найповніший стиль запису, найменш використовуваний. Починаючи з PHP 5.0.0 ці змінні можна виключити. Починаючи з PHP 6 ці змінні недоступні.

Приклад запису змінної з плаваючою точкою та рядкової змінної: `$variable = 0.00; $variable1 = 'hello PHP-Master!'; $variable2 = «hello PHP-Coder!»;`

З точки зору системи типізації, PHP є мовою програмування з динамічною типізацією. Немає необхідності явного визначення типу змінних, хоча така можливість існує. В разі звернення до змінної, інтерпретатор PHP трактує її тип відповідно до контексту. За необхідності можливе приведення змінної до певного типу за допомогою відповідних конструкцій мови. Це може знадобитись, якщо зважити, що значення змінної можуть трактуватись по-різному в залежності від її типу. Також можливе визначення типу відповідної змінної на певному етапі виконання сценарію. Імена змінних чутливі до регістру символів.

До базових типів належать булеві дані, цілі та дійсні числа із плаваючою комою, а також рядки. Булеві дані виражають істинність значення. Цілі числа можуть бути подані у вісімковому, десятковому та шістнадцятковому вигляді. Розмір цілого числа може змінюватись залежно від платформи. Як правило, розрядність становить 32 біти. PHP не підтримує беззнакові цілі числа. Дійсні числа із плаваючою комою можуть бути подані в десятковій або експоненційній формі.

**Робота з рядками.** Рядки ділять на два класи: рядки, що підлягають аналізу, та ті, що не підлягають. Перший клас досліджується інтерпретатором на наявність посилань на інші змінні, і за умови їхньої наявності робиться підстановка значень у відповідне місце. Крім того, клас дозволяє проводити маніпуляції з керівними символами. Символ рядка може мати лише одне з 256 значень, але є можливість працювати з багатобайтовими символами. Доступ до символів рядка можливий з використанням синтаксису, схожого на доступ до елементів масивів.

PHP надає широкий спектр функцій для пошуку та заміни тексту в рядках. Для цього використовують як традиційний підхід, так і спеціальний, що базується на використанні регулярних виразів. При цьому в мові реалізована підтримка двох видів регулярних виразів – Perl-сумісні та POSIX-сумісні, що розрізняються за синтаксисом та особливостями роботи.

**Змішані типи даних.** До змішаних типів належать масиви, хеші та об'єкти. Масиви в сенсі мови є наборами змінних, що згруповані в єдину змінну. Вимога однотипності наповнення масивів не ставиться. Технічно, масиви – це впорядковані карти, що відображають ключові значення на позиції змінних даних. Вмістом значення, на

яке вказує ключ може бути будь-чим, що можна подати у вигляді змінної. Не існує жодних обмежень, крім обсягу пам'яті, що накладаються на кількість ключів масиву.

Особливістю мови є відмова від рівномірного розподілу ключів масивів. Реалізовано і модель багатовимірних масивів, причому без явного обмеження глибини вкладеності. Корисною властивістю PHP є можливість асоціації масивів із функцією зворотного виклику. Ці функції дозволяють проводити дії над одним чи кількома масивами в пакетному режимі.

PHP є мовою програмування з динамічною типізацією, що не вимагає вказівки типу при оголошенні змінних, так само як і самого оголошення змінних. Перетворення між скалярними типами може здійснюватися автоматично (хоча і є можливість для явного перетворення типів).

До скалярних типів даних відносяться:

- цілий тип (integer);
- дійсний тип даних (float, double);
- логічний тип (boolean);
- рядковий тип (string);
- спеціальний тип NULL.

До нескаларних типів відноситься

- ⇒ «ресурс» (resource);
- ⇒ масив (array);
- ⇒ об'єкт (object).

Тип NULL призначений для змінних без певного значення. Значення NULL набувають неініціалізовані змінні, змінні ініціалізовані константою NULL, а також змінні, видалені за допомогою конструкції `unset()`.

Посилання на зовнішні ресурси мають тип resource. Змінні даного типу, як правило, є дескриптором, що дозволяє управляти зовнішніми об'єктами, такими як файли, динамічні зображення, результуючі таблиці бази даних і тому подібне.

Масиви підтримують числові і рядкові ключі та є гетерогенними. Масиви можуть містити значення будь-яких типів, включаючи інші масиви. Суперглобальними масивами (superglobal arrays) в PHP називаються наперед визначені масиви, які видно в будь-якому місці початкового коду без використання ключового слова `global`:

- `$GLOBALS` – масив всіх глобальних змінних (у тому числі і призначених для користувача);
- `$_SERVER` – містить множину інформації про поточний запит і сервер;
- `$_ENV` – поточні змінні середовища. Їх набір специфічний для кожної конкретної платформи, на якій виконується сценарій;
- `$_GET` – асоціативний масив з параметрами GET-запиту. У початковому вигляді ці параметри доступні в `$_SERVER['QUERY_STRING']` і в `$_SERVER['REQUEST_URI']` у складі URI;
- `$_POST` – асоціативний масив значень полів HTML-форми при відправці методом POST;

- `$_FILES` – асоціативний масив з відомостями про відправлені файли методом POST. Кожен елемент має індекс ідентичний значенню атрибуту «name» у формі `i`, у свою чергу, також є масивом з наступними елементами:
  - `['name']` – початкове ім'я файлу на комп'ютері користувача;
  - `['type']` – вказаний агентом користувача MIME-тип файлу;
  - `['size']` – розмір файлу в байтах;
  - `['tmp_name']` – повний шлях до файлу в тимчасовій папці;
  - `['error']` – код помилки;
- `$_COOKIE` – асоціативний масив з переданими агентом користувача значеннями cookie;
- `$_REQUEST` – спільний масив даних, що вводиться в запиті користувача як в масивах `$_GET`, `$_POST`, `$_COOKIE`. Починаючи з версії PHP 4.1 включається і вміст `$_FILES`.
- `$_SESSION` – інформація про поточну сесію користувача.

PHP підтримує широкі об'єктно-орієнтовані можливості, повна підтримка яких була введена в п'ятій версії мови. Клас в PHP оголошується за допомогою ключового слова `class`. Методи і поля класу можуть бути загальнодоступними (`public`, за замовчуванням), захищеними (`protected`) і прихованими (`private`). PHP підтримує всі три основні механізми ООП – інкапсуляція, поліморфізм і спадкоємство (батьківський клас вказується за допомогою ключового слова `extends` після імені класу). Підтримуються інтерфейси (ставляться у відповідність з допомогою `implements`). Вирішується оголошення фінальних, абстрактних методів і класів. Множинне спадкоємство класів не підтримується, проте клас може реалізовувати декілька інтерфейсів. Для звернення до методів батьківського класу використовується ключове слово `parent`. Екземпляри класу створюються за допомогою ключового слова `new`, звернення до полів і методів об'єкту проводиться з використанням символів `->`. Для доступу до членів класу з його методів використовується змінна `$this`.

Серед найчастіше використовуваних можливостей PHP варто відзначити наступні:

- ⇒ *є великий набір функцій для роботи з рядками;*
- ⇒ *робота з регулярними виразами PCRE;*
- ⇒ *робота з базами даних, що здійснюється за допомогою модулів:*
  - ⇒ *php5-mysql для MySQL;*
  - ⇒ *php5-pgsql для PostgreSQL;*
- ⇒ *для PHP розроблені засоби шаблонування веб-сторінок, що дозволяють ефективно розділити застосування від моделі, наприклад Smarty;*
- ⇒ *є бібліотека для роботи з графічними зображеннями GD, що дозволяє проводити перетворення з графічними файлами, і створювати зображення миттєво.*

Також, існують два спеціальні типи даних – ресурс та `NULL`. Ресурс – спеціальна змінна, що містить посилання на зовнішній ресурс. Ресурси створюються та використовуються в спеціалізованих функціях. Оскільки тип містить спеціальні вказівники

на відкриті файли, під'єднання та інше, то будь-які дії над значенням ресурсу не мають сенсу.

Область видимості змінної – середовище, в якому вона визначена. Розрізняють локальні та глобальні змінні. За замовчуванням, всі змінні мають локальний характер дії. Виділяють особливий тип змінних – статичні, що дозволяє повторне звернення до змінної в певному сегменті коду, причому змінна буде зберігати попередньо отримане значення. Існує також поняття суперглобальних змінних, які є місцем збереження даних оточення або даних, отриманих ззовні. Підтримується концепція динамічних змінних та функцій.

Константи в PHP – ідентифікатори простих значень. Можливе визначення константи, причому після її оголошення стає неможливою зміна її значення чи анулювання. Константи можуть мати лише скалярні значення. Підтримується можливість отримання значення константи за динамічним ім'ям. Область видимості констант буде глобальною для сценарію та всіх під'єднаних компонентів. Також в ядрі мови визначено чимало системних констант.

Оператори в мові PHP дозволяють виконувати відповідну дію над одним чи кількома операндами. Оператори бувають трьох типів – унарні, бінарні та тернарні. Оператори, як і в інших мовах характеризуються не лише дією, а й асоціативністю та пріоритетністю. Особливістю булевих операцій порівняння – розрізнення двох класів – з врахуванням типу і без врахування типу, при якому відбувається приведення до відповідного типу. Округлення відбуваються завжди в меншу сторону. В мові реалізовані особливі класи операторів – виконання, управління помилками та перевірки приналежності до класу.

Функції в сенсі мови є контейнерами коду, причому можливе включення інших функцій та класів. На цьому і базується можливість умовного визначення функції. В цьому випадку висувається вимога попередньої декларації викликаної функції, що не обов'язкове в інших випадках. Можливості перевизначення чи деактивації функції не існує. Результат, який повертає функція може мати будь-який тип.

В мові реалізована функціональність посилань. Можна створити скільки завгодно псевдонімів, що посилаються на єдиний сегмент даних. При вивільненні будь-якого з псевдонімів, сегмент даних залишається в пам'яті до моменту завершення сценарію або вивільнення усіх посилань.

Що стосується функцій в PHP, то замість прийнятого в багатьох мовах принципу перевантаження функцій, що дозволяє змінити хід виконання певної функції в залежності від типу та кількості переданих параметрів, використовується метод динамічних аргументів. Це дає змогу не визначати кількість параметрів для функцій при їх оголошенні, а працювати із тими аргументами, які були отримані на момент виклику функції. У тілі функції можна отримати кількість переданих їй аргументів і проводити відповідні маніпуляції. При оголошенні функції звичайним чином, можливе задання значень аргументів за замовчуванням. Функції можуть повертати лише одне значення, проте це обмеження можна оминати, використавши не лише масиви, а й посилання. Передача аргументів за посиланням неможлива під час виконання та оголошення функції.

Після виконання сценаріїв, простір пам'яті, що займається ними очищується збиранням сміття. Проте, за необхідності можливе виконання очищення пам'яті від надлишкових сегментів даних під час виконання скриптів. Використання функцій очищення пам'яті є невиправданим, хоча така можливість існує.

Для побудови програмних комплексів можна використовувати модульний підхід, виконуючи розділення різнорідного коду. При потребі, можливе виконання під'єднання необхідних модулів, причому операція виконання може бути і умовною. Під'єднані до скрипта файли можуть повертати значення.

Мова явно підтримує HTTP cookies відповідно до специфікацій Netscape. Це дозволяє проводити встановлення та читання невеликих сегментів даних на стороні клієнта.

RНР надає можливість організації роботи з користувачем протягом сеансів (сесій). В сесії можна зберігати різні дані, включаючи об'єкти.

### 2.2.2. Мова сценаріїв Perl

Мова Perl (Practical Extraction and Report Language) – це мова програмування, сильними сторонами якої вважаються її великі можливості для роботи з текстом, у тому числі реалізовані за допомогою регулярних виразів. Також, мова відома тим, що має величезну колекцію додаткових модулів CPAN.

Щоб запустити програму на мові Perl на виконання, її компіляція не потрібна, вона повністю може виконуватися під управлінням інтерпретатора. Щоб файл з початковим текстом Perl можна було запускати на виконання, треба аби перший його рядок виглядав так:

```
#!/шлях_до_інтерпретатора_Perl
```

Perl – Практична Мова для Витягання текстів і Генерації звітів (Practical Extraction and Reporting Language), як і раніше популярна не лише серед лінукоїдов, але і серед Web-програмістів.

Почнемо з найпростішого. Введемо у файл test1.pl наступні рядки:

```
#!/usr/local/bin/perl  
# Вміст файлу test1.pl  
print «Вітаємо читача\n»;
```

Розберемо кожен рядок та пояснимо його значення:

```
#!/usr/local/bin/perl
```

Даний рядок має бути першим в будь-якій Perl-програмі. Вона вказує системному інтерпретатору, що даний файл – це Perl-програма.

```
# Вміст файлу test1.pl
```

Це рядок коментаря. Він завжди починається символом '#'.

```
print «Вітаємо читача\n»;
```

Останній рядок просто виводить на екран «Вітаємо читача».

Тут слово print – це команда «вивести». Все, що в лапках – це символи, \n – переведення рядка і ';' – ознака кінця команди. Вона обов'язкова. У одному рядку може

бути декілька команд і всі вони повинні завершуватися символом ';'. Після неї може бути символ '#' – тоді залишок рядка вважається коментарем.

Щоб цей приклад запрацював, вам треба мати встановлений Perl і набрати в командному рядку: perl test1.pl (у Windows) або ./test.pl (у \*nix).

Perl-програма (скрипт) складається з послідовності декларацій і пропозицій. Що має обов'язково декларувати perl-програма, так це формати звітів і підпрограми (функції). Всі неоголошені змінні, масиви мають значення 0 або null.

**Декларації (оголошення).** Perl має вільний формат. Коментарі починаються з символу '#' і продовжуються до кінця рядка. Декларації можуть використовуватися в будь-якому місці програми так само як і пропозиції (statements), але діють вони тільки у фазі компіляції програми. Зазвичай їх поміщають або на початку чи в кінці програми. Декларація підпрограм дозволяє використовувати ім'я підпрограми як списковий оператор, починаючи з моменту декларування:

```
sub test; # Декларація підпрограми test  
Svar1 = test $0; # Використання як оператора списку.
```

Декларації підпрограм можуть бути завантажені з окремого файлу пропозицією require або завантажені й імпортовані в поточну область імен пропозицією use.

Проста пропозиція обов'язково закінчується символом ';', якщо тільки це не остання пропозиція в блоці, де ';' можна опустити. Існують оператори, такі як eval{ і do{, які виглядають як складні пропозиції, але насправді це терми і вони вимагають обов'язкової вказівки кінця пропозиції.

Будь-яка проста пропозиція може містити модифікатор ';'. Існують наступні модифікатори:

```
if EXPR, unless EXPR, while EXPR, until EXPR
```

де EXPR – вираз, що повертає логічне значення true або false. Модифікатори while і until обчислюються на початку пропозиції, крім do, який виконується першим.

if EXPR – модифікатор «якщо». Пропозиція виконується, якщо EXPR повертає true.

```
Svar = 1;  
if Svar > 0 Svar2 = 3; # Результат: Svar2 = 3
```

```
while EXPR – модифікатор «поки». Пропозиція виконується поки EXPR = true  
Svar = 1;  
print Svar++ while Svar < 5; # Результат: 1234
```

until EXPR – модифікатор «до». Пропозиція виконується до тих пір, поки EXPR = false

```
Svar = 1;  
print Svar++ until Svar > 5; # Результат: 12345
```

unless EXPR – модифікатор «якщо не». Зворотний до if. Вираз виконується, якщо EXPR = false.

```
Svar = 1;  
print Svar++ unless Svar > 5; # Результат: 1
```

**Складні пропозиції.** Послідовність простих пропозицій, обмежена функціональними обмежувачами, називається блоком. У Perl це може бути цілий файл, послідовність пропозицій в операторі eval{ або найчастіше це безліч простих пропозицій, обмежених круглими дужками '{'.

Існують наступні види складних пропозицій:

**if (EXPR) BLOCK, if (EXPR) BLOCK else BLOCK, if (EXPR) BLOCK elsif (EXPR) BLOCK ... else BLOCK**

**LABEL while (EXPR) BLOCK, LABEL while (EXPR) BLOCK continue BLOCK  
LABEL for (EXPR; EXPR; EXPR) BLOCK, LABEL foreach VAR (LIST) BLOCK  
LABEL BLOCK continue BLOCK**

Зверніть увагу, що складні пропозиції описані в термах блоків, а не пропозицій, як у мові C. Тому необхідно використовувати круглі дужки для позначення блоку.

**if (EXPR) BLOCK** – обчислюється логічний вираз EXPR і, якщо true блок виконується.

```
$var=1;
if ($var == 1)
{ print $var; }
}
```

Результат: 1

**if (EXPR) BLOCK else BLOCK2** – якщо EXPR=true виконується BLOCK інакше BLOCK2.

```
$var=2;
if ($var == 1)
{ print "\$var = 1\n"; }
else
{ print "\$var не рівне 1\n"; }
}
```

Результат: \$var не рівне 1

**if (EXPR1) BLOCK1 elsif (EXPR2) BLOCK2 ... else BLOCK** - Якщо EXPR1=true виконується BLOCK1 інакше якщо EXPR2=true виконується BLOCK2 інакше ... інакше BLOCK.

```
$var = 1;
if ($var == 0)
{ print "\$var = 0\n"; }
elseif ($var == 1)
{ print "\$var = 1\n"; }
else
{ print «Невідоме \$var\n»; }
}
```

Результат: \$var = 1

Цикл While виконує BLOCK до тих пір, поки EXPR = true. Мітка LABEL не обов'язкова і складається з ідентифікатора, що завершується символом ':'. Мітка необхідна при використанні у середині блоку циклу операторів next, що управляють, last і redo. Якщо мітка відсутня, то ці оператори посилаються на початок найближчого циклу. Блок після continue виконується завжди перед тим, як обчислюється логічний вираз EXPR. Це подібно EXPR3 в пропозиції for, тому в цьому блоці зручно змінювати лічильники і прапори циклу, навіть якщо застосовується оператор next.

**Оператори управління циклом.** next – подібний continue в C, переходить на початок поточного циклу, тобто повторює ітерацію.

```
M1:
while ($i < 6)
{
++$i;      # Збільшуємо лічильник на 1
next M1 if $i < 3; # Переходимо в початок якщо $i < 3
++$i;      # інакше збільшуємо лічильник ще раз на 1
}
continue
{
print «$i <»; # Результат: 1 2 4 6
}
```

last – подібний до оператора break в мові C, що негайно перериває цикл. Блок continue пропускається.

```
M1:
while ($i < 6)
{
++$i;      # Збільшуємо лічильник на 1
last M1 if $i > 3; # Вихід з циклу якщо $i > 3
++$i;      # інакше збільшуємо лічильник ще раз на 1
}
continue {
print «$i <»; # Результат: 2 4
}
```

redo – почати новий цикл, не обчислюючи EXPR і не виконуючи continue блок.

```
M1:
while ($i < 6)
{
++$i;      # Збільшуємо лічильник на 1
redo M1 if $i == 3; # Далі пропустити для $i = 3
++$i;      # інакше збільшуємо лічильник ще раз на 1
}
continue {
print «$i <»; # Результат: 2 5 7
}
```

Цикл **for**. **for** (EXPR1; EXPR2; EXPR3) BLOCK. Оператор **for** повністю аналогічний операторові **for** в С. Перед початком циклу виконується EXPR1, якщо EXPR2 = true виконується блок, потім виконується EXPR3.

```
for ($i = 2; $i < 5; ++$i){
    print $i, « »; # Результат: 2 3 4
}
print «\nПісля циклу i = $i\n»; # Після циклу i = 5
```

Цикл **foreach**.

**LABEL foreach VAR (LIST) BLOCK.**

Змінною VAR присвоюється по черзі кожен елемент списку LIST і виконується блок. Якщо VAR опущене, то елементи присвоюються вбудованій змінній `$_`. Якщо в тілі блоку змінювати значення VAR, то це викличе зміну і елементів списку, оскільки VAR фактично вказує на поточний елемент списку. Замість слова **foreach** можна писати просто **for** – це слова синоніми.

```
@month = («січень», «лютий», «березень»); # Створили масив
foreach $i (@month)
{ print $i, « »; # Результат: січень лютий березень
}
foreach $i (@month)
{ $i = uc($i); # Перевели у верхній регістр
}
print @ month; # Результат: СІЧЕНЬ ЛЮТИЙ БЕРЕЗЕНЬ
for $i (3,5,7)
{ print «$i »; # Результат: 3 5 7
}
```

**Блоки і оператор switch.** Блок, не залежно від того має він мітку або ні, семантично є циклом, який виконується один раз. Тому дія операторів циклу **next**, **last**, **redo** – аналогічна описаному вище. Блоки зручні для побудови **switch** (перемикач) структур. У Perl немає спеціального оператора **switch** подібного до мови С, тому можна самостійно створювати зручні для вас конструкції. Досвід показує, що для простоти написання краще всього підходить конструкція виду **if ... elsif ... else ...** хоча можна скласти і щось подібне:

**SWITCH:**

```
{
    if ($i == 1) { .....; last SWITCH; }
    if ($i == 2) { .....; last SWITCH; }
    if ($i == 3) { .....; last SWITCH; }
    $default = 13;
}
```

У Perl існує оператор **goto**. При створенні великих виробничих завдань на останньому етапі, особливо при відробці помилкових ситуацій **goto** потрібен. У Perl реалізовано три форми **goto**: **goto** – мітка, **goto** – вираз і **goto** – підпрограма.  
**goto**-мітка – виконує безпосередній перехід на вказану мітку.

**goto**-вираз – обчислює ім'я мітки і робить відповідний перехід. Наприклад, якщо ми хочемо зробити перехід на одну з трьох міток «M1:», «M2:» або «M3:» залежно від значень змінної `$i` рівній 0, 1 або 2, то це краще зробити таким чином:

```
goto («M1», «M2», «M3»)[$i];
```

Тут `$i` використовується як індекс масиву вказаного безпосередньо у виразі.

**goto**-підпрограма – досить окремий випадок, оскільки завжди простіше і надійніше викликати підпрограму простим чином.

**POD оператори. Документування програм.** У Perl реалізований дуже зручний механізм для написання документації у момент створення програми. Для цього застосовуються спеціальні оператори POD. Якщо в тілі програми інтерпретатор зустрічає оператор, що починається з символу '=' наприклад:

```
= head Набір стандартних процедур
```

то пропускається все до слова '=cut'. Це зручно для включення довгих на декілька рядків або сторінок коментарів. Потім за допомогою спеціальної програми **pod** можна відокремити текст документації від тексту програми.

**Змінні.** У Perl існує три типи структур даних: скаляри, масиви скалярів і хеши (**hashes**) – асоціативні масиви скалярів. Зазвичай елементи масивів індексуються цілими числами, перший елемент – нульовий. Від'ємне значення індексу позначає номер позиції елемента з кінця. Хеши індексуються рядками символів.

Імена скалярів завжди починаються з символу '\$' – навіть, коли позначають елемент масиву:

```
$var1          # Простий скаляр 'var1'
$var1[0]        # Перший елемент масиву 'var1'
$var1{'first'}  # Елемент з індексом 'first'
```

В разі використання імені масиву «повністю» або його «зрізу» перед ім'ям масиву ставиться символ '@':

```
@var1          # Всі елементи масиву var1 ($var1[0], $var1[1]..., $var1[n])
@var1[1,3,10]   # Елементи $var1[1], $var1[3], $var1[10]
@var1{'first', 'last'} # те ж, що і ($var1{'first'}, $var1{'last'})
```

Хеш «повністю» починається з символу '%':

```
%var %key, %years
```

Імена підпрограм починаються символом '&', якщо з контексту не видно, що це підпрограма:

```
&sub1 &test_prog, test(12)
```

Імена таблиць символів завжди починаються символом '\*':

Кожен тип змінних має свою область пам'яті, тому `$var1` і `$var1[0]` абсолютно різні змінні, хоча `$var1[0]` частина масиву `@var1`. Так само `@var1` і `%var1` – різні масиви змінних.

Імена змінних можуть містити будь-які буквенно-цифрові символи за винятком пропуску і табуляції. Ці символи використовуються як роздільники. Великі і малі букви розрізняються, тому `$var1` і `$Var1` – різні змінні. У Perl за замовчуванням імена міток і покажчиків файлів пишуть великими буквами.



**Контекст.** Велике значення для правильного вживання вбудованих функцій має контекст використання результату цих функцій, оскільки інакше вони повертають абсолютно «незрозумілий» результат. У Perl є два головні контексти: скалярний і список (list). Якщо в лівій частині виразу є одне єдине значення, то це скалярний контекст. Якщо багато значень – то це список:

```
Svar1 = <>; # Прочитати один рядок файлу
@var1 = <>; # Прочитати всі рядки файлу в масив @var1
Svar1 = (1,2,3); # Svar = 3 - кількість елементів
@var1 = (1,2,3); # Створення масиву @var1 з елементами 1,2,3
```

**Скалярні значення.** Всі дані в Perl це скаляри, масиви скалярів і хеши скалярів. Скалярні змінні можуть містити числа, рядки і посилання. Перетворення числа в рядки відбувається автоматично за замовчуванням. Скаляр може мати тільки одне єдине значення, хоча це може бути посилання на масив скалярів. Оскільки Perl сам перетворює числа в рядки і навпаки, то програмістові немає необхідності думати про те, що повертає функція.

У Perl не існує типів «рядок» або «число», або «файл» чи щось ще. Це контекстно залежна поліморфна мова для роботи з текстами. Скаляр має логічне значення «TRUE» (істина), якщо це не нульовий рядок або число не рівне 0.

У Perl існує два типи нульових (null) скалярів – це визначені (defined) і невизначені (undefined). Невизначене значення повертається, коли щось не існує. Наприклад, невідома змінна, кінець файлу або помилка. За допомогою функції defined() можна заздалегідь виявити подібний стан.

Кількість елементів масиву так само є скаляром і починається символами \$#. Фактично \$#var1 – це індекс останнього елементу масиву. Потрібно пам'ятати, що перший елемент має індекс 0, тому кількість елементів визначається як \$#var1+1. Присвоєння значення \$#var1 змінить довжину масиву і зруйнує «залишені» значення. Присвоєння значення елементу масиву з індексом більше ніж \$#var1 збільшить розмір масиву, а присвоєння йому нульового списку – обнулить.

У скалярному контексті ім'я масиву повертає його довжину (для списку повертається останній елемент):

```
@var1 = (4, 3, 2, 1); # Присвоєння значення елементам масиву
$# = @var1; # Використання скалярного контексту
print $#; # Друк результату 4 - к-ть елементів
print @var1; # Списковий контекст, друк всіх елементів.
```

Для примусового отримання скалярного значення зручно застосовувати функцію scalar():

```
print scalar (@var1); # Виведення довжини масиву, а не його значень
```

Хеш в скалярному контексті повертає «true», якщо існує хоча би одна пара «ключ-значення». Фактично повертається рядок типу 2/8, де 8 – кількість виділених «елементів» пам'яті, а 2 – кількість використаних.

Конструктори скалярів. Числа пишуться стандартно:

```
123
123.123
```

```
0.12
.12E-10
0xABCD # Шістнадцятковий запис
0377 # Якщо 0 на початку – вісімкова
123_456_123 # Так теж можна для зручності читання.
```

Рядки обмежуються одинарними (') або подвійними (») лапками: 'Володіння інформацією – знання!' або «Знання – це світло.» У хеші можна опускати лапки, якщо індекс не містить пропусків: Svar1{first} те ж що і Svar1{'first'}

Зверніть увагу на те, що перед першою одинарною лапкою повинен стояти пропуск, інакше рядок сприйметься як ім'я змінної, оскільки в іменах дозволено використання одинарних лапок. Забороняється в лапках застосовувати зарезервовані літерали \_\_LINE\_\_ (номер поточного рядка програми), \_\_FILE\_\_ (поточний файл). Для позначення кінця програми можна застосовувати літерал \_\_END\_\_. Весь наступний текст ігнорується, але його можна прочитати, використовуючи вказівники файлу DATA.

Слова в програмі, що непіддаються ніякій інтерпретації, сприймаються як рядки в лапках, тому рекомендується імена міток і покажчиків файлів писати великими буквами, щоб уникнути можливого «конфлікту» із зарезервованими словами.

У Perl є можливість вставляти текст документа прямо в програму, використовуючи «here-doc» (тут текст) метод. Позначається символами <<, за якими йде слово-обмежувач:

```
print <<EOF; # Всі рядки до EOF – текст для друку.
Колись звалась територія України Руссю,
А повна назва її була – Київська Русь
EOF
```

**Конструктори списків.** Список – множина значень, перерахованих через кому і вкладених у круглі дужки. У списковому контексті список повертає останній елемент списку:

```
@var1 = (1, 2, 'привіт', 1.2); # Присвоїти значення елементам, де
Svar1[0]= 1
Svar1[1]= 2
Svar1[2]= 'привіт'
Svar1[3]= 1.2
Svar1 = (1, 2, 'привіт', 1.2);
```

а тут Svar1 = 1.2 останнє значення списку.

Допускається застосовувати в списку інші списки, але в отриманому списку вже неможливо розрізнити початок і кінець включених списків:

```
@s1 = (1, 2, 3); # Перший список
@s2 = (6, 7, 8); # Другий
@s = (0 @s1, 4, 5 @s2, 9, 10); # Включаємо списки @s1 і @s2
print @s; # Результат: 012345678910 – значення без пропусків.
```

Список без елементів позначається, як `()`, і називається нуль-списком. Списковий вираз можна вживати як ім'я масиву, але при цьому брати в круглі дужки:

```
print ('січень','лютий','березень')[1];
```

Результат: лютий.

Список може бути присвоєний списку тільки, якщо кожен елемент в списку у лівій частині виразу допустимий за типом списку в правій частині:

```
($a, $b, $c) = (1, 2, 3); # $a = 1 $b
```

**Оператори і пріоритети.** У Perl асоціативність і пріоритетність операторів аналогічна мові C.

**Створення бібліотеки.** Якщо потрібно створити модуль окремим файлом і використовувати як бібліотеку підпрограм, але при цьому викликати підпрограми бібліотеки, не вказуючи імені модуля, вам необхідно оформити модуль таким чином:

```
package ім'я_модуля;      #Таке ж, як ім'я файлу без розширення '.pm'
require Exporter; # Обов'язковий рядок для експорту імен
@ISA = qw(Exporter);     # -//
@EXPORT = qw(func1 func2) #Перерахуємо імена функцій. Немає коми!
@EXPORT_OK = qw($змінна @масив); #Вказати публічні #змінні, масиви і
```

так далі, якщо необхідно

```
{ # Початок блоку модуля
```

```
.....
sub func1
```

```
.....
sub func2
```

```
.....
1;
}
```

Даний файл з розширенням «.pm» повинен зберігатися в одній з бібліотечних директорій Perl. Вони перераховані в масиві `@INC`, одна з них зазвичай `</usr/local/lib/perl/>`.

У головній програмі ви вказуєте:

```
use ім'я_модуля;
```

і вам стають доступні імена підпрограм даного модуля.

**Perl-бібліотеки.** Стандартний набір бібліотек, зазвичай, поставляється з дистрибутивом Perl, вони розділяються на pragma бібліотеки (працюють як директиви компілятора) і стандартні бібліотеки.

**Pragma бібліотеки.** Дані бібліотеки використовують як:

```
use ім'я;
```

коли хочуть включити дію і

```
no ім'я;
```

коли хочуть вимкнути.

У стандартний набір входять наступні pragma:

➤ `diagnostics` – включити режим розширеної діагностики;

- `integer` – використовувати цілочисельну арифметику;
- `less` – режим мінімального завантаження компілятора;
- `overload` – режим перевизначення операторів;
- `sigtrap` – режим стеження за перериваннями;
- `strict` – режим обмеженого використання «небезпечних» операторів;
- `subs` – режим обов'язкового декларування підпрограм.

**Стандартні бібліотеки.** З точки зору web-програмування, найбільший інтерес представляють бібліотеки для роботи з тегами HTML і для доступу до баз даних.

**CPAN.** Програмісти всього світу, що працюють з Perl, створили загальнодоступну бібліотеку модулів CPAN. Вона доступна через Інтернет і містить величезну кількість різних за призначенням модулів. До них відносяться документатори, системні інтерфейси, інтерфейси роботи з базами даних, робота в мережі, з файлами, Інтернет-браузери, системи пошуку, величезна кількість CGI-скриптів для Web-серверів і багато чого інше.

**Формати.** У Perl реалізований зручний метод створення форматованих звітів. За допомогою оператора `format` ви описуєте заголовки, розміри полів, вказуєте положення даних на листі в зручній текстовій формі. Потім виконуєте команду `write(файл)`, яка виводить дані, що відформатували, у вказаний файл.

Оператор `format` має наступний синтаксис:

```
format ім'я =
FORMLIST
```

Зверніть увагу на те, що опис формату йде після рядка `format` і закінчується символом `'.'` на початку рядка. Тут `'ім'я` – це ім'я формату, таке ж як і ім'я вказівника вихідного файлу. Якщо `'ім'я` відсутнє, то значення за замовчуванням – `STDOUT`.

**FORMLIST** – це рядки формату. Вони бувають трьох типів:

1. Коментар. Рядок починається символом `'#'`;
2. Описувач полів даних (`picture`);
3. Рядок аргументів використовуваних описувачем.

Описувач – це рядок, який виводиться у вигляді «як є» за винятком спеціально позначених форматів полів даних. Кожне поле починається або символом `'@'`, або `'^'`. У описовому рядку вказується тільки положення і вид даних, що виводяться, але не імена полів і змінних. Для цього призначений наступний рядок аргументів, який слідує завжди після описувача і містить імена змінних або цілі вирази в порядку вказаному описувачем. Розмір і вид поля в описувачі позначається символами:

«<<<<»	– вирівняти значення по правому краю;
«>>>>»	– вирівняти значення по лівому;
«   »	– вирівняти значення по центру;
«####.###»	– формат числа з крапкою;
«@*»	– багаторядковий рядок. Дані виводяться в колонку.

Розмір поля дорівнює кількості вказаних символів. Символ `'^'` на початку поля має спеціальне значення. Так:

- `«^####»` – пусто, якщо змінна не визначена,

А для рядкового скаляра:

- «^<<<<<<» – виводиться скільки можна символів, а значення змінної міняється на залишок, виведення якого можна продовжити на наступних рядках, які можуть мати свої поля.

Спеціальні змінні:

- \$~ – порядковий формат вмісту;
- \$^ – формат заголовку листа;
- \$% – номер листа;
- \$= – рядків у листі.

Якщо ви хочете використовувати одні формати для різних файлів, то найпростіший шлях:

```
use FileHandle;           # Вказати на початку програми
format_name файл ім'я_формата; # Формат вмісту листа.
format_top_name файл ім'я_формата; # Формат заголовка листа.
write(файл);              # виведення даних.
```

Тут 'файл' маєтись на увазі покажчик файлу, отриманий командою open();

Якщо вам потрібно в тілі листа виводити різного роду формати (наприклад, заголовки груп або відбиття листа), то застосовуйте format\_name.

### 2.2.3. Мова розроблення сценаріїв JSP

JavaServer Pages (JSP) дозволяють відокремити динамічну частину сторінок від статичного HTML. Запишемо звичайний код в HTML, використовуючи для цього будь-яку програму для створення Web-сторінок. Потім Ви укладете динамічну частину коду в спеціальні теги, більшість яких починаються з «<%» і завершуються «%>». Як приклад розглянемо секцію JSP-сторінки, результатом якої буде щось подібне до «Спасибі за покупку Core Web Programming» за запитом з URL: <http://host/OrderConfirmation.jsp?title=Core+Web+Programming>:

Спасибі за покупку

```
<I><%= request.getParameter(«title») %></I>
```

Ви даєте вашому файлу розширення .jsp і розміщуєте там же, де повинні розміщуватися звичайні сторінки Web. Хоча те, що ви написали більше схоже на звичайний файл HTML чим на сервлет, просто за кадром JSP-сторінка перетвориться в звичайний сервлет із статичним HTML, який просто прямує в потік виводу, пов'язаний з методом сервлета service. Зазвичай це відбувається при першому запиті сторінки і розробники можуть відразу після установки самі виконати цей запит, якщо хочуть щоб перший реальний користувач при зверненні до сторінки не зіткнувся з невеликою затримкою, викликану трансляцією JSP-сторінки в сервлет і його подальшою компіляцією і завантаженням. Також відзначимо, що більшість Web-серверів дозволяють задавати посилання (aliases), так що адреса URL, вказуюча на HTML-файл насправді вказуватиме на сервлет або сторінку JSP.

Окрім стандартних конструкцій HTML існують ще три основні типи конструкцій JSP, які ви можете включити в сторінку: елементи скриптів, директиви і дії. Елементи

скриптів дозволяють вам вказати код на мові Java, яка згодом стане частиною в кінцевому сервлеті, директиви дадуть вам можливість управляти всією структурою сервлета, а дії служать для завдання існуючих використовуваних компонентів, а також для контролю поведінки апарату JSP. Для спрощення елементів скриптів, ви маєте доступ до декількох заздалегідь визначених змінних, таким, наприклад, як змінна request, використана в приведеному вище уривку.

Зверніть увагу що це керівництво охоплює JSP версії 1.0. А починаючи з версії 0.92 JSP зазнала безліч змін, і не дивлячись на те, що ці зміни були лише на розвиток, ви повинні пам'ятати, що JSP версії 1.0 практично повністю не сумісна зі старішими версіями JSP. Також слід не забувати про те, що пропонуване вашій увазі керівництво є частиною повнішого керівництва по сервлетам і JSP, доступного за адресою <http://www.apl.jhu.edu/~hall/java/Servlet-Tutorial/>.

Текст шаблону: статичний HTML. Як правило велику частину вашої JSP-сторінки складає статичний HTML, що зветься текстом шаблону. В усіх відношеннях (окрім одного) цей HTML виглядає як звичайний HTML, що використовує ті ж правила синтаксису, і він просто «передається» клієнтові сервлетом, що створюється для обробки сторінки. При цьому не лише сам HTML виглядає нормальним, він може створюватися із застосуванням тих інструментів, які раніше використовувались при створенні Web-сторінок. При створенні більшості JSP-сторінок для цього керівництва, зазвичай, використовується Allaire's HomeSite.

Єдиним виключенням з правила, що «текст шаблону передається в незмінному вигляді» є ситуація, коли в тексті Ви хочете відображати послідовність символів «<%», для цього в тексті шаблону треба використовувати поєднання символів «<%».

**Елементи скриптів JSP.** Елементи скриптів JSP дозволяють вставляти код на Java в сервлет, що створюється з поточної сторінки JSP. Існують три форми:

- вирази, що мають форму <%= вираз %>, які обробляються і прямують на вивід;
- скриплети, що мають форму <% код %>, які вставляються в метод service сервлета;
- оголошення, що мають форму <%! код %>, які вставляються в тіло класу сервлета, поза існуючими методами.

Всі вони окремо детально розглядатимуться далі.

Вирази JSP застосовуються для того, щоб вставити значення Java безпосередньо у вивід. Для цього використовується наступна форма:

```
<%= Вираз на Java %>
```

Вирази Java обчислюються, конвертуються в рядок і вставляються в сторінку. Ці обчислення відбуваються під час виконання (тобто при запиті сторінки), а тому існує повний доступ до інформації про сам запит. Наприклад, наступний код служить для відображення дати і часу запиту даної сторінки:

Поточний час:

```
<%= new java.util.Date() %>
```

Для того, щоб спростити ці вирази існує декілька заздалегідь визначених змінних, які ви можете використовувати. Детальніше вони розглядатимуться нижче. Приведемо декілька найбільш важливих з них:

```
request, HttpServletRequest;
response, HttpServletResponse;
session, HttpSession асоціюється із запитом (якщо такий є);
out, PrintWriter (буферизований варіант типу JspWriter), що використовується для відсилання даних, що виводяться, клієнтові.
```

Приклад:

```
Ім'я вашого хоста: <%= request.getRemoteHost() %>
```

І, нарешті, програмісти, що використовують XML, можуть застосувати альтернативний синтаксис для виразів JSP:

```
<jsp:expression>
```

Вирази на Java:

```
</jsp:expression>
```

Пам'ятаймо, що елементи XML на відміну від HTML чутливі до регістру. Тому переконайтеся в тому, що використовуєте рядкові символи.

**Скриплети JSP.** Якщо потрібно зробити щось більше чим вставка простих виразів, скриплети JSP дадуть можливість вставити будь-який код в метод сервлета, який буде створений при обробці даної сторінки. Скриплети мають наступний вигляд:

```
<% Код на Java %>
```

Скриплети також мають доступ до тих же автоматично певних змінних, що і вирази. Тому, наприклад, якщо Ви хочете вивести що-небудь на сторінку, то потрібно скористатися змінною out.

```
<%
```

```
String queryData = request.getQueryString();
```

```
out.println(«Додаткові дані запиту: « + queryData);
```

```
%>
```

Зверніть увагу на те, що код у середині скриплету вставляється в тому вигляді, як він записаний, і весь статичний HTML (текст шаблону) до або після скриплету конвертується за допомогою оператора print. Це означає, що скриплети не обов'язково повинні містити завершені фрагменти на Java і, що залишені відкритими блоки можуть зробити вплив на статичний HTML зовні скриплету. Наприклад, наступний фрагмент JSP містить змішаний текст шаблону і скриплет:

```
<% if (Math.random() < 0.5) { %>
```

```
<B>Щасливого</B> Вам дня!
```

```
<% } else { %>
```

```
<B>Нещасливого</B> Вам дня!
```

```
<% } %>
```

після перетворення буде приведено до:

```
if (Math.random() < 0.5) {
```

```
out.println(«<B> Щасливого </B> вам дня!»);
} else {
out.println(«<B> Нещасливого</B> вам дня!»);
}
```

Якщо ви хочете використовувати послідовність символів «%>» у середині скриплету, замість неї використовуйте «%\>». Еквівалентом <% код %> для XML є:

```
<jsp:scriptlet>
```

Код

```
</jsp:scriptlet>
```

Оголошення JSP дозволять задати методи або поля, для вставки в тіло класу сервлета (поза методом service, що обробляє запит). Вони мають наступну форму:

```
<%! Код на Java %>
```

Оскільки оголошення не здійснюють виводу, зазвичай вони використовуються спільно з JSP-виразами або скриплетами. У приведеному в якості прикладу фрагменті JSP відображається кількість запитів до даної сторінки з моменту завантаження сервера (або з моменту останньої зміни і перезавантаження сервлета):

```
<%! private int accessCount = 0; %>
```

Кількість звернень до сторінки з моменту завантаження сервера:

```
<%= ++accessCount %>
```

Також як і в скриплетах, якщо необхідно використовувати послідовність символів «%>», використовуйте для цього послідовність «%\>». XML еквівалентом <%! Код %> є:

```
<jsp:declaration>
```

Код

```
</jsp:declaration>
```

**3. Директиви JSP.** Директиви JSP впливають на всю структуру класу сервлета. Зазвичай вони мають наступну форму:

```
<%@ директива атрибут=»значення» %>
```

Ви також можете об'єднати установку декількох атрибутів для однієї директиви:

```
<%@ директива атрибут1=»значення1»
```

```
атрибут2=»значення2»
```

```
...
```

```
атрибутN=»значенняN» %>
```

Існують два основні типи директив: page, яка дозволяє здійснювати такі операції, як імпорт класів, зміну суперкласу сервлета, і т.п.; і include, яка дає можливість вставити файл в клас сервлета при трансляції JSP-файлу в сервлет. Також слід згадати директиву taglib, яка не підтримується у JSP версії 1.0, але дозволяє авторам JSP давати свої власні теги.

**Директива JSP page.** Директива page дозволяє задати один і більше наступних чутливих до регістра атрибутів:

1. import=»пакет.class» або

**import=>»пакет.class1,...,пакет.classN».**

Дозволяє задати пакети, які мають бути імпортовані. Наприклад:

```
<%@ page import=>»java.util.*» %>
```

Атрибут import – єдиний атрибут, що допускає багатократне використання.

**2.contentType=>»MIME-Тип» або**

**contentType=>»MIME-Тип; charset=Кодування-Символів»**

Задає тип MIME для виводу. За замовчуванням використовується text/html. Наприклад, директива

```
<%@ page contentType=>»text/plain» %>
```

приводить до того ж результату, що і використання скриплету

```
<% response.setContentType(«text/plain»); %>
```

**3.isThreadSafe=>»true|false».**

Значення true («істина», приймається за замовчуванням) задає нормальний режим виконання сервлета, коли множинні запити обробляються одночасно з використанням одного екземпляра сервлета, виходячи з міркування, що автор синхронізував доступ до змінних цього екземпляра. Значення false («брехня») сигналізує про те, що сервлет повинен успадковувати SingleThreadModel (однопоточну модель), при якій послідовні або одночасні запити обробляються окремими екземплярами сервлета.

**4.session=>»true|false».**

Значення true («істина», приймається за замовчуванням) сигналізує про те, що заздалегідь визначена змінна session (тип HttpSession) має бути прив'язана до існуючої сесії, якщо така є, інакше створюється нова сесія, до якої і здійснюється прив'язка. Значення false («брехня») визначає, що сесії не використовуватимуться, і спроби звернення до змінної session приведуть до виникнення помилки при трансляції JSP сторінки в сервлет.

**5.buffer=>»розмір kb|none».**

Задає розмір буфера для JspWriter out. Те значення, що приймається за замовчуванням залежить від налаштувань сервера, але повинно перевищувати 8kb.

**6.autoflush=>»true|false».**

Значення true («істина», що приймається за замовчуванням) встановлює, що при переповнюванні буфер повинен автоматично очищатися. Значення false («брехня»), яке у край рідко використовується, встановлює, що переповнювання буфера повинне приводити до виникнення виняткової ситуації. При установці значення атрибута buffer=>»none» установка значення false для цього атрибута не припустима.

**7.extends=>»пакет.class».**

Задає суперклас для сервлета, що генерується. Цей атрибут слід використовувати з великою обережністю, оскільки можливо, що сервер вже використовує який-небудь суперклас.

**8.info=>»повідомлення».**

Задає рядок, який може бути отриманим при використанні методу getServletInfo.

**9.errorPage=>»url».**

Задає JSP-сторінку, яка викликається в разі виникнення яких-небудь подій Throwables, які не обробляються на даній сторінці.

**10.isErrorPage=>»true|false».**

Сигналізує про те, чи може ця сторінка використовуватися для обробки помилок для інших JSP-сторінок. За замовчуванням набуває значення false («брехня»).

**11.language=>»java».**

Даний атрибут призначений для задавання використовуваної мови програмування. За замовчуванням набуває значення «java», оскільки на сьогоднішній день це єдина підтримувана мова програмування.

Синтаксис задавання директив на XML

```
<jsp:directive.тип_директиви атрибут=значення />
```

Приклад. Еквівалентом XML для

```
<%@ page import=>»java.util.*» %>
```

є

```
<jsp:directive.page import=>»java.util.*» />
```

**Директива JSP include.** Ця директива дозволяє вам включати файли в процесі трансляції JSP-сторінки в сервлет. Використання директиви виглядає таким чином:

```
<%@ include file=>»відносний url» %>
```

Заданий URL зазвичай інтерпретується відносно JSP-сторінки, на якій розташовано посилання, але, як і при використанні будь-яких інших відносних URL можна задати системі положення ресурсу, що цікавить Вас, відносно домашнього каталогу Web-сервера, додавши в початок URL символ «/». Вміст файлу, що підключається, обробляється як звичайний текст JSP і тому може включати такі елементи як статичний HTML, елементи скриптів, директиви і дії.

Наприклад, багато сайтів використовують невелику панель навігації на кожній сторінці. У зв'язку з проблемами використання фреймів HTML часто це завдання вирішується розміщенням невеликої таблиці зверху або в лівій половині сторінки, HTML-код якої багато разів повторюється для кожної сторінки сайту. Директива include найбільш природний спосіб рішення цієї задачі, що позбавляє розробника від рутини копіювання HTML в кожен окремий файл. Це відбувається таким чином:

```
<!DOCTYPE HTML PUBLIC «-//W3C//DTD HTML 4.0 Transitional//EN»>
```

```
<HTML>
```

```
<HEAD>
```

```
<TITLE>Інформація по сервлетам: JavaServer Pages (JSP) 1.0</TITLE>
```

```
<META NAME=>»author» CONTENT=>»webmaster@somesite.com»>
```

```
<META NAME=>»keywords» CONTENT=>»...»>
```

```
<META NAME=>»description» CONTENT=>»...»>
```

```
<LINK REL=STYLESHEET
```

```
  HREF=>»Site-Styles.css»
```

```
  TYPE=>»text/css»>
```

```
</HEAD>
```

```
<BODY>
<%@ include file=»/navbar.html» %>
<!-- Специфічний фрагмент цієї сторінки ... -->
</BODY>
</HTML>
```

Потрібно врахувати, що оскільки директива `include` підключає файли в ході трансляції сторінки, то після внесення змін до панелі навігації буде потрібна повторна трансляція всіх, що використовують її JSP-сторінки. В даному випадку це є хорошим компромісом, оскільки, як правило, панель навігації міняється досить рідко і процес підключення не втрачає своєї ефективності. Якщо ж підключені файли міняються досить часто, Ви можете використовувати замість цього дію `jsp:include`. Цю дію підключає файл в процесі звернення до JSP-сторінки.

Розглянемо приклад використання елементів скриптів і директив. На цьому простому прикладі показано використання різних конструкцій JSP: виразів, скриплетів, оголошень і директив.

```
<!DOCTYPE HTML PUBLIC «-//W3C//DTD HTML 4.0 Transitional//EN»>
<HTML>
<HEAD>
<TITLE>Використання JavaServer Pages</TITLE>
<META NAME=»author» CONTENT=»Marty Hall -- hall@apl.jhu.edu»>
<META NAME=»keywords»
  CONTENT=»JSP,JavaServer Pages,servlets»>
<META NAME=»description»
  CONTENT=»Швидкий приклад чотирьох основних тегів JSP.»>
<LINK REL=STYLESHEET
  HREF=»My-Style-Sheet.css»
  TYPE=»text/css»>
</HEAD>
<BODY BGCOLOR=»#FDF5E6» TEXT=»#000000» LINK=»#0000EE»
  VLINK=»#551A8B» ALINK=»#FF0000»>
<CENTER>
<TABLE BORDER=5 BGCOLOR=»#EF8429»>
<TR><TH CLASS=»TITLE»>
  Використання JavaServer Pages</TH>
</TR>
</TABLE>
</CENTER>
<P>
```

Деякий динамічний зміст, створений з використанням різних механізмів JSP:

```
<UL>
<LI><B>Вираз.</B><BR>
  Ім'я вашого хоста: <%= request.getRemoteHost() %>.
<LI><B>Scriptlet.</B><BR>
```

```
<% out.println(«Додаткові дані запиту: « +
  request.getQueryString()); %>
<LI><B>Оголошення (спільно з виразом).</B><BR>
<%! private int accessCount = 0; %>
  Кількість звернень до сторінки з моменту завантаження сервера: <%=
  ++accessCount %>
<LI><B>Директива (спільно з виразом).</B><BR>
<%@ page import = «java.util.*» %>
  Поточна дата: <%= new Date() %>
</UL>
</BODY>
</HTML>
```

Наперед визначені змінні. Для спрощення коду у виразах JSP і скриплетів надається набір їх восьми автоматично визначених змінних, які деколи називають неявними об'єктами. Доступні змінні – це `request`, `response`, `out`, `session`, `application`, `config`, `pageContext` і `page`. Далі ми розглянемо кожну з них окремо.

`Request` – це об'єкт `HttpServletRequest`, пов'язаний із запитом, який дозволяє звертатися до параметрів запиту (через метод `getParameter`), типу запиту (`GET`, `POST`, `HEAD`, і так далі.), і вхідних заголовків `HTTP` (`cookies`, `Referer`, і так далі.). Простіше кажучи, `request` є підкласом `ServletRequest` і може відрізнятися від `HttpServletRequest`, якщо використовується протокол, відмінний від `HTTP`, що на практиці практично ніколи не зустрічається.

`Response` – це об'єкт типу `HttpServletResponse`, пов'язаний з відповіддю на запит клієнта. Зверніть увагу, що, оскільки потік виводу (`out`) буферизує, можна змінювати коди стану `HTTP` і заголовки відповідей навіть, якщо це неприпустимо в звичайному сервлеті, але лише в тому разі, якщо якісь дані виводу вже були відправлені клієнтові.

`Out` – це об'єкт типу `PrintWriter`, використовуваний для відправки виводу клієнтові. Проте, щоб зробити об'єкт `response` корисним, слід використовувати буферизований варіант `PrintWriter` – `JspWriter`. Пам'ятайте, що Ви можете змінювати розмір буфера і навіть відключити буферизацію, змінюючи значення атрибуту `buffer` директиви `page`. Також зверніть увагу, що `out` використовується практично виключно скриплетами, оскільки вирази JSP автоматично поміщаються в потік виводу, що позбавляє від необхідності явного звернення до `out`.

`Session` – це об'єкт типу `HttpSession`, пов'язаний із запитом. Сесії створюються автоматично і ця змінна існує навіть, якщо немає посилань на вхідні сесії. Єдиним виключенням є ситуація, коли Ви відключаєте використання сесій використовуючи атрибут `session` директиви `page`. В цьому випадку посилання на змінну `session` приводять до виникнення помилок при трансляції JSP-сторінки в сервлет.

`Application` – це об'єкт типу `ServletContext`, отриманий через використання методу `getServletConfig().getContext()`.

`Config` – це об'єкт типу `ServletConfig` для поточної сторінки.

`PageContext`. У JSP представлений новий клас `PageContext` для ізольованого використання специфічних особливостей сервера, таких як ефективніші `JspWriters`. Ідея



полягає в тому, що, якщо Ви звертаєтесь до них через цей клас, а не безпосередньо, Ваш код може виконуватися на «звичайних» апаратах сервлет/JSP.

Page – по суті є синонімом для this, і не потрібний при роботі з Java. Ця змінна створювалася з розрахунком на перспективу, коли можливо, з'являться інші мови програмування скриптів, відмінні від Java.

**Дії JSP.** Дії JSP використовують конструкції з синтаксисом XML для управління роботою апарату сервлета. Ви можете динамічно підключати файл, багато разів використовувати компоненти JavaBeans, направити користувача на іншу сторінку або згенерувати HTML для Java plugin. Допустимо вживання наступних дій:

- `jsp:include` – підключає файл у момент запиту сторінки;
- `jsp:useBean` – пошук або створення нового екземпляру JavaBean;
- `jsp:setProperty` – установка пріоритетів JavaBean;
- `jsp:getProperty` – вставити властивість JavaBean в потік виводу;
- `jsp:forward` – перенаправляє запит на іншу сторінку;
- `jsp:plugin` – генерує код (залежно від типу використовуваного браузера), який створює тег OBJECT або EMBED для Java plugin.

Всі ці дії детально розглянемо далі. Пам'ятайте, що як і у всьому XML, імена елементів і атрибутів регістрозалежні.

**Дія `jsp:include`.** Ця дія дозволяє вставляти файли в сторінку, що генерується. Синтаксис дії:

```
<jsp:include page=»відносний URL» flush=»true» />
```

На відміну від директиви `include`, яка вставляє файл на етапі трансляції JSP-сторінки цю дію вставляє файл при запиті сторінки. Це приводить до деякої втрати ефективності і унеможливорює наявності у файлі коду JSP (наприклад, Вам не вдасться задати заголовки HTTP), що вставляється, та зате дає істотну перевагу в гнучкості. Візьмемо, наприклад, JSP сторінку, яка вставляє чотири різні уривки у Web-сторінку з новинами сайту. Кожного разу, коли міняються заголовки авторіві досить змінити вміст чотирьох файлів, тоді як головна сторінка JSP залишається незмінною.

WhatsNew.jsp. Ви можете скачати вихідні дані файлу або викликати цей скриплет з Інтернету.

```
<!DOCTYPE HTML PUBLIC «-//W3C//DTD HTML 4.0 Transitional//EN»>
<HTML>
<HEAD>
<TITLE>Новини</TITLE>
<LINK REL=STYLESHEET
  HREF=»My-Style-Sheet.css»
  TYPE=»text/css»>
</HEAD>
<BODY BGCOLOR=»#FDF5E6» TEXT=»#000000» LINK=»#0000EE»
  VLINK=»#551A8B» ALINK=»#FF0000»>
<CENTER>
```

```
<TABLE BORDER=5 BGCOLOR=»#EF8429»>
<TR><TH CLASS=»TITLE»>
  Новини на JspNews.com</TABLE>
</CENTER>
<P>
```

Нижче приведені фрагменти чотирьох найпопулярніших статей:

```
<OL>
<LI><jsp:include page=»news/Item1.html» flush=»true»/>
<LI><jsp:include page=»news/Item2.html» flush=»true»/>
<LI><jsp:include page=»news/Item3.html» flush=»true»/>
<LI><jsp:include page=»news/Item4.html» flush=»true»/>
</OL>
</BODY>
</HTML>
```

**Дія `jsp:useBean`.** Ця дія дозволяє завантажувати JavaBean для подальшого використання на JSP-сторінці. Це дуже важлива можливість, оскільки вона дозволяє використовувати багатократне використання класів Java не відмовляючись при цьому від переваг, що надаються сервлетами JSP. Простий синтаксис для вказівника використовуваного bean:

```
<jsp:useBean id=»ім'я» class=»пакет.class» />
```

Як правило це означає «створення нового екземпляра об'єкта класу, заданого через `class`, і його зв'язок із змінною з ім'ям, заданим за допомогою `id`». Проте, як Ви скоро переконаєтесь, можна задати атрибут `scope`, який асоціює bean не лише з поточною сторінкою. У такому разі, корисно отримати посилання на існуючі beans, і дію `jsp:useBean`, що створює екземпляр нового об'єкту лише в тому разі, якщо не існує жодного об'єкту з тими ж значеннями `id` і `scope`. Тепер, коли у є bean, Ви можете змінювати його властивості за допомогою `jsp:setProperty`, або використовуючи для цього скриплет і явно викликаючи метод об'єкту з ім'ям змінної, заданої раніше через атрибут `id`. Recall that with beans, коли Ви говорите «біля цього bean є властивість типу X з назвою foo», то Ви насправді будете мати на увазі «біля цього класу є метод `getFoo`, який повертає дані типу X, і інший метод `setFoo`, якому як параметр передається X». Дія `jsp:setProperty` детальніше розглядатиметься далі, але зараз Ви повинні запам'ятати, що Ви можете або явно задавати `value`, задаючи атрибут `param`, щоб набути значення з відповідного параметра запиту, або просто перерахувати властивості, щоб отримати значення з параметрів запиту з тими ж іменами, що і властивості. Ви можете набути значень існуючих властивостей за допомогою виразів JSP або скриплетів, викликавши відповідний метод `getXxx`, або (найчастіше), скориставшись дією `jsp:getProperty`.

Пам'ятаймо, що клас заданий для bean повинен знаходитися в звичайному каталозі класів сервера, а не в частині, зарезервованій для класів, автоматично перезавантажених після редагування. Наприклад, для Java Web Server, всі використовувані класи

повинні розміщуватися в каталозі classes або в jar файлі каталога lib, а не в каталозі servlets.

Нижче приведений дуже простий приклад, завантажуючий bean і отримуючий простий рядковий параметр.

**BeanTest.jsp.** Можна скачати вихідні дані файлу або викликати цей скриплет з Інтернету.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Багатократне використання JavaBeans в JSP</TITLE>
<LINK REL=STYLESHEET
    HREF=»My-Style-Sheet.css«
    TYPE=»text/css«>
</HEAD>
<BODY>
<CENTER>
<TABLE BORDER=5>
<TR><TH CLASS=»TITLE«>
    Багатократне використання JavaBeans в JSP</TABLE>
</CENTER>
<P>
<jsp:useBean id=»test« class=»hall.SimpleBean« />
<jsp:setProperty name=»test«
    property=»message«
    value=»Привіт, WWW« />

<H1>Повідомлення: <I>
<jsp:getProperty name=»test« property=»message« />
</I></H1>

</BODY>
</HTML>
```

**SimpleBean.java.** Тут приведений початковий код для bean, використаного на JSP-сторінці BeanTest.

```
package hall;

public class SimpleBean {
    private String message = «Текст повідомлення не заданий»;

    public String getMessage() {
        return(message);
    }
}
```

```
}

public void setMessage(String message){
    this.message = message;
}
}
```

Дія jsp:setProperty. Ви можете використовувати jsp:setProperty для присвоєння значень властивостям раніше описаних beans. Це можна робити двома способами. По-перше, можна використовувати jsp:setProperty, але поза елементом jsp:useBean, як це показано в прикладі:

```
<jsp:useBean id=»myName« ... />
...
<jsp:setProperty name=»myName«
    property=»someProperty« ... />
```

В цьому випадку jsp:setProperty виконується незалежно від того, чи був знайдений існуючий bean або був створений новий екземпляр. Інший варіант полягає в розміщенні jsp:setProperty в тілі елемента jsp:useBean, як це показано в іншому прикладі:

```
<jsp:useBean id=»myName« ... >
...
<jsp:setProperty name=»myName«
    property=»someProperty« ... />
</jsp:useBean>
```

При цьому jsp:setProperty виконується лише в тому випадку, якщо був створений новий екземпляр об'єкту, а не тоді, коли знаходиться той, що вже існує.

Далі наведений приклад, в якому використовується bean для створення таблиці простих чисел. Якщо існує параметр numDigits в даних запиту, то він передається у властивість bean numDigits. Аналогічно для numPrimes.

Дія jsp:getProperty. Цей елемент визначає значення властивості bean, конвертує його в рядок і направляє в потік виводу. Для виконання дії потрібне задання двох атрибутів: імені bean, яке заздалегідь задається у дії jsp:useBean, і ім'я властивості, значення якої має бути визначене. Далі наведений приклад використання цієї дії.

```
<jsp:useBean id=»itemBean« ... />
...
<UL>
<LI>Кількість предметів:
    <jsp:getProperty name=»itemBean« property=»numItems« />
<LI>Ціна за штуку:
    <jsp:getProperty name=»itemBean« property=»unitCost« />
</UL>
```

Дія jsp:forward. Ця дія дозволяє передати запит іншій сторінці. Вона використовує один атрибут page, який повинен містити відносний URL. У неї може бути як статич-

не значення, так і обчислюване в процесі запиту, що і показано на наступних двох прикладах:

```
<jsp:forward page=»/utils/errorReporter.jsp» />
<jsp:forward page=»<%= який небудь вираз Java %>» />
```

Дія jsp:plugin. Ця дія дозволяє вам вставити елемент OBJECT або EMBED (залежно від типу використовуваного браузера), необхідний для запуску аплетів, що використовують plugin Java.

## ТЕМА 2.3. РОЗРОБКА CGI-ЗАСТОСУВАНЬ НА PERL, PHP

### 2.3.1. Розробка CGI-застосовувань на Perl

### 2.3.2. Основи розробки сценаріїв на мові PHP

#### 2.3.1. Розробка CGI-застосовувань на Perl

Розглянемо питання, як створюється CGI сценарій? Теоретично це дуже просто – програма CGI, як і будь-яка інша програма на Perl виконує звичайні команди Perl, коли вона викликається браузером (тобто, коли браузеру як URL задається CGI-сценарій). Все, що Ви направляєте в стандартний вивід, передається браузеру. Так, якщо CGI-сценарій виконує команду `print «Hello!»`, цей текст буде повернений браузеру, і на сторінці з'явиться напис «Hello». Але це рудиментарний спосіб. Чи потрібно прочитати дані, введені користувачем за допомогою елементів управління, розташованих на сторінці?

Інтерпретатор Perl містить серед інших модулів стандартний модуль `CGI.pm`. Тому, якщо у вас встановлений Perl, то, швидше за все, є і `CGI.pm`. Починаючи з п'ятої версії Perl `CGI.pm` став об'єктно-орієнтованим, хоча спрощений функціонально-орієнтований інтерфейс все ще існує. У наших прикладах ми використовуватимемо об'єктно-орієнтоване програмування. Створюючи за допомогою `CGI.pm` об'єкти CGI, ми викликатимемо різні методи цього об'єкту. Існують методи, відповідні практично всім основним тегам HTML, і при їх виклику створюється потрібний тег з вказаними атрибутами. Всі вони можуть отримувати іменовані параметри (за винятком методів, що вимагають один аргумент), іншими словами, потрібно вказати не лише значення атрибуту HTML, але і його ім'я. У наступному прикладі об'єкт CGI створює Web-сторінку за допомогою вбудованих методів. Зверніть увагу на іменовані параметри методу `textarea`, задаючи ім'я області редагування тексту ('`textarea`'), значення за замовчуванням і розміри:

Приклад створення Web-сторінки.

```
#!/usr/local/bin/perl
use CGI;
$co = new CGI;
print $co->header, $co->start_html(-title=>'CGI Example'),
$co->center($co->h1('Welcome to CGI!'))
$co->textarea (
-name => 'textarea'
-default => 'No opinion'
-rows => 10
-columns => 60 ),
$co->end_html;
```

Створення і використання елементів управління HTML. Розглянемо нижче приведені два CGI-сценарії: один створює Web-сторінку з елементами управління – полями введення тексту, перемикачами, кнопками, включаючи Submit, а другий читає дані, введені користувачем на цій сторінці. Обидва сценарії – варіації з невеликими доповненнями на тему оператора `print`, який, власне, і створює сторінку.

Перший сценарій зберігається у файлі `cgil.pl`, і в довідкових цілях повністю приводиться в прикладі, що буде приведений нижче. Коли користувач відкриває сценарій в браузері (переходячи за його адресою – наприклад, `http://www.yourserver.com/user/cgi/cgil.pl`), сценарій повертає сторінку з елементами управління HTML і текстом. В даному випадку це сторінка анкети.

Сторінка містить привітання і повідомлення про те, що відвідувачі, що не бажають заповнювати анкету, можуть перейти по посиланню на сервер CPAN (Comprehensive Perl Archive Network – всеосяжний архів, присвячений мові Perl).

Потім ви побачите запит імені користувача з текстовим полем і питання про його думку з областю редагування тексту (багаторядкове текстове поле).

Проглядаючи анкету далі ви побачите ще декілька елементів управління – кнопки із залежною і незалежною фіксацією, списки, а також кнопку підтвердження і очищення анкети. Пізніше ми розглядатимемо, як створити всі ці елементи управління на прикладі запропонованих сценаріїв. Коли користувач натискає на кнопку Submit, розташовану в кінці анкети, браузер збирає дані, введені на сторінці, і передає їх іншому CGI-сценарію, `cgi2.pl`. У довідкових цілях він приведений в прикладі, що буде відображений далі.

Загалом, немає ніякої необхідності створювати CGI-сценарій, що генерує анкету – можна просто написати сторінку HTML, яка викликати `cgi2.pl` по натисненню користувачем кнопки Submit. Підхід, що вимагає генерації коду, вибраний лише потім, щоб продемонструвати обидві сторони процесу – це, як створити елементи управління HTML з CGI-сценарієм і, як прочитати дані з цих елементів управління.

Приклад. `cgil.pl`

```
#!/usr/local/bin/perl
use CGI; $co = new CGI;
$labels{'1'} = 'Sunday'; $labels{'2'} = 'Monday';
$labels{'3'} = 'Tuesday'; $labels{'4'} = 'Wednesday';
$labels{'5'} = 'Thursday'; $labels{'6'} = 'Friday';
$labels{'7'} = 'Saturday';
print $co->header(-charset=>'windows-1251', -lang=>'ru'),
$co->start_html(
    -title => 'CGI Example'
    -author => 'Volodumur'
    -meta => {'keywords' => 'CGI Perl'},
    -BGCOLOR => 'white'
    -LINK => 'red' ),
$co->center($co->h1('Here is the Survey!'))
$co->h2('Please fill out survey...')
'Reasons for filling out our survey:', $co->p,
$co->ul( $co->li('Fame'), $co->li('Fortune'), $co->li('Fun') )
'If you would rather not fill out our survey ',
'you might be interested in ',
$co->a({href=>'http://www.cpan.org/'}),»CPAN»»»»,
$co->hr $co->startform(
```

```
-method=>'POST'
-action=>'http://inna/scripts/web/l6/e8.pl'),
'Please enter your name: ', $co->textfield('text'), $co->p,
'Please enter your opinion: ', $co->p,
$co->textarea( -name => 'textarea'
    -default => 'No opinion'
    -rows => 10
    -columns => 60 ),
$co->p, 'Please indicate what products you use: ', $co->p,
$co->checkbox_group(
    -name => 'checkboxes'
    -values => ['Shampoo', 'Toophpaste', 'Bread', 'Cruise missiles'],
    -defaults => ['Bread', 'Cruise missiles'] ), $co->p,
'Please indicate your income level: ', $co->p,
$co->scrolling_list('list', ['Highest', 'High', 'Medium', 'Low'], 'High') $co->p,
'Please indicate your day of a week: ', $co->p,
$co->radio_group(
    -name => 'radios'
    -values => ['1', '2', '3', '4', '5', '6', '7'],
    -default => '1'
    -labels => \%labels), $co->p,
'Thank you for filling out our survey. Please indicate ',
'How much unsolicited mail you like to get: ', $co->popup_menu(
    -name => 'popupmenu'
    -values => ['Very much', 'A lot', 'Not so much', 'None'] ), $co->p,
$co->hidden(-name=>'hiddendata', -default=>'Rosebud')
$co->center( $co->submit, $co->reset, )
$co->hr $co->endform, $co->end_html;
```

Приклад. `cgi2.pl`

```
#!/usr/local/bin/perl
use CGI; $co = new CGI;
print $co->header(-charset=>'windows-1251', -lang=>'ru'),
$co->start_html(
    -title => 'CGI Example'
    -author => 'Volodymyr'
    -meta => {'keywords'=>'CGI Perl'},
    -BGCOLOR => 'white'
    -LINK => 'red' ),
$co->center($co->h1('Thanks for filling out our survey.'))
$co->h3('Here is your responses...') $co->hr;
if ($co->param()) { print «Your name is: »,
$co->em($co->param('text')), «.» $co->p, «Your opinions are: »,
$co->em($co->param('textarea')) «.»,$co->p,
«You use these products: »,
```

```

Sco->em(join(«», «», Sco->param('checkboxes'))), «.»
Sco->p, «Your income level is:», Sco->em(Sco->param('list'))»,»,
Sco->p, «Today is day»,
Sco->em(Sco->param('radios')), «</em> of the week.», Sco->p,
«How much unsolicited mail you like: «,
Sco->em(Sco->param('popupmenu'))»,», Sco->p,
«The hidden data is «, Sco->em(join(«», «»,
Sco->param('hiddendata'))))}
print Sco->hr; print Sco->end_html;

```

Починаємо HTML-документ. Початок роботи над документом HTML будується таким чином: спочатку ви створюєте CGI-об'єкт, потім за допомогою методу header цього об'єкту – HTTP-заголовок (у даному прикладі створюється проста шапка документа, але допустимі, скільки завгодно, складні шапки з будь-якими атрибутами, наприклад –charset=>'windows-1251'). Заголовок можна сформулювати і самостійно, наприклад, командою print. Метод start\_html починає сам документ HTML. Цей метод створює секцію <HEAD>, а також дозволяє вказати деякі атрибути <BODY>, як, наприклад: колір для зображення фону і посилань. Нижче приведений фрагмент коду cgil.pl, що відкриває сторінку. Зверніть увагу: щоб результати роботи методів header і start\_html потрапили на сторінку необхідно використовувати функцію print:

```

#!/usr/local/bin/perl
$co = new CGI
print $co->header,
$co->start_html(
-title => 'CGI Example'
-author => 'Volodymyr'
-meta => {'keywords'=>'CGI Perl'},
-BGColor => 'white'
-LINK => 'red')

```

Створюємо заголовки HTML. Після створення шапки CGI-методи типу h1, h2, h3 і ін. допоможуть створити заголовки, які відповідні тегам <H1>, <H2>, <H3> і так далі. Нижче приведений фрагмент коду, що генерує заголовки <H1> і <H2> на початку Web-сторінки з анкетною. В даному випадку це просте запрошення користувачу.

```

#!/usr/local/bin/perl
$co = new CGI; print
$co->h1('Here is the Survey!'),
$co->h2('Please fill out survey...')

```

Центруємо елементи. Щоб центрувати текст за допомогою тегів <CENTER>, використовується CGI-метод center. У наступному прикладі центрується заголовок, створений в попередньому прикладі:

```

#!/usr/local/bin/perl
$co = new CGI; print
$co->center($co->h1('Here is the Survey!'))
$co->h2('Please fill out survey...')

```

Створюємо маркований список. CGI-методи ul і li створюють несортований маркований список (теги <UL> і <LI> відповідно). Нижче приведений фрагмент коду, що представляє користувачеві декілька аргументів, запрошуючих заповнити анкету:

```

#!/usr/local/bin/perl
$co = new CGI; print «Reasons for filling out our survey:»,
$co->p $co->ul(
$co->li('Fame') $co->li('Fortune '), $co->li('Fun'), )

```

Створюємо гіперпосилання. Гіперпосилання поміщаються на сторінку CGI-методом так, як в прикладі нижче, де виводиться URL для переходу (на випадок, якщо користувач не зацікавлений в заповненні анкети, створеної сценарієм cgil.pl):

```

#!/usr/local/bin/perl
$co = new CGI; print
«If you would rather not fill out our survey «,
«you might be interested in»
$co->a({href=>}http://www.cpan.org/»), «CPAN») «. «

```

Створюємо горизонтальну смугу. Для створення горизонтальної лінії (мітка <HR>) використовується CGI-метод hr:

```

#!/usr/local/bin/perl
$co = new CGI; print
$co->hr

```

Створюємо HTML-форму. Елементи управління HTML групуються у форми. У прикладі з анкетною для створення форми використовувався CGI-метод startform. Після натиснення кнопки Submit дані з елементів управління мають бути прочитані і передані сценарію, що формує зведення даних, тобто cgi2.pl. URL цього сценарію вказується в атрибуті action форми:

```

#!/usr/local/bin/perl $co = new CGI; print
$co->startform(
-method=>'POST'
-action=>'http://www.yourself.com/user/cgi/cgi2.pl')
#$co->startform()

```

Всі подальші елементи управління будуть включені у форму, тому що метод startform генерує тег <FORM>.

Зауваження. Якщо startform викликається без аргументів, кнопка Submit повертає введені дані тій же формі. Нижче розповідається, як використовувати таку можливість.

Працюємо з текстовими полями. Для створення текстового поля, що дозволяє вводити текст, використовується CGI-метод textfield. У прикладі нижче створюється текстове поле, призначене для зберігання імені користувача.

```

#!/usr/local/bin/perl
«Please enter your name: »,
$co->textfield('text')

```

Читання даних з елементів управління HTML. Елементи управління створені (точніше, поки тільки текстове поле). Але як рахувати з них дані? Коли користувач натискуватиме на кнопку Submit, браузер відправить дані форми сценарію cgi2.pl. CGI-метод param в нім якраз і призначений для читання даних. Йому досить передати ім'я, присвоєне текстовому полю, а в даному випадку – це 'text'. Вивід виконується таким чином:

```
#!/usr/local/bin/perl
$co = new CGI;
print «Your name is: «, $co->em($co->param( 'text' )), «»;
```

Далі наведений приклад читання і друку вибору користувача, узятий з сценарію cgi2.pl:

```
print «How much unsolicited mail you like»
$co->param('popupmenu');
```

Працюємо з прихованими полями даних. Дані, що зберігаються в прихованому полі на Web-сторінці, невидимі для користувача. (Це зручно, коли сценарій чекає отримати деякі незмінні відомості про сторінку.) Такі поля створюються таким чином:

```
#!/usr/local/bin/perl
$co = new CGI; print
$co->hidden(-name=>'hiddendata', -default=>'Rosebud')
```

І ось як вивести ці дані з cgi2.pl:

```
print «The hidden data is: «,
join(«, «, $co->param('hiddendata')));
```

Створюємо кнопки відміни і підтвердження. Щоб відправити на сервер дані форми, користувач повинен натискувати кнопку Submit. Вона створюється CGI-методом submit. Аналогічно, кнопка Reset, яка очищає дані форми, створюється методом reset. Нижче наведений приклад коду, що створює кнопки Submit і Reset на Web-сторінці:

```
#!/usr/local/bin/perl
$co = new CGI; print
$co->center(
$co-> submit(-value=>'Відправити')
$co->reset(-value=>'Відміна'))
```

Після натиснення на кнопку Submit дані відправляються сценарію cgi2.pl.

Закриваємо HTML-форму. Всі елементи управління, описані раніше, є частиною однієї форми анкети, створеної в cgil.pl. Для відкриття форми використовувався метод startform, а для її закриття – endform:

```
#!/usr/local/bin/perl
$co = new CGI; print
$co->endform
```

Закриваємо HTML-документ. Щоб завершити роботу з HTML-документом, використовуйте метод end\_html, який виводить теги </BODY></HTML>. От як закінчується сторінка з анкетною в сценарії cgil.pl:

```
#!/usr/local/bin/perl $co = new CGI;
```

```
print
$co->end_html;
```

На цьому cgil.pl кінчається. Коли користувач введе дані і натискуватиме кнопку Submit, буде викликаний сценарій cgi2.pl, який виведе зведення анкети.

**Функціонально-орієнтоване CGI-програмування.** До цих пір ми використовували об'єктно-орієнтовані методи. Проте пакет CGI має і функціонально-орієнтований інтерфейс (Втім, при зверненні до нього деякі можливості об'єктно-орієнтованого інтерфейсу стають недоступними). У прикладі нижче використовується функціонально орієнтований інтерфейс пакету CGI. Код генерує текстове поле з пропозицією ввести ім'я користувача. Після натиснення на кнопку Submit дані повертаються до того ж CGI-сценарію, який за допомогою функції param виводить введені дані в нижній частині Web-сторінки:

Приклад. Функціонально-орієнтований інтерфейс пакету CGI.

```
#!/usr/local/bin/perl
use CGI qw/:standard/;
print header(-charset=>'windows-1251', -lang=>'ua')
start_html(' CGI Functions Example' ),h1('CGI Functions Example')
start_form, «Please enter your name: «, textfield('text'), p
submit(-value=>'Відправити'), reset(-value=>'Відміна'), end_form,hr;
if(param()) {print «Your name is: «,em(param('text')), hr;}
print end_html;
```

Використання cgi-lib.pl. Раніше розповідалося про CGI-програмування на базі методів стандартного модуля CGI.pm. Не менш популярний серед програмістів пакет cgi-lib.pl. Оскільки багато CGI-сценаріїв на Perl написано з його допомогою, то далі розповідається саме про нього. Авторські права на цей пакет належать його творцеві Стівену Е. Бреннеру, на домашній сторінці якого (<http://cgi-lib.stanford.edu/cgi-lib>) можна отримати копію cgi-lib.pl. Вам дозволяється працювати з cgi-lib.pl і навіть змінювати його до тих пір, поки ваші дії не ущемлюватимуть описані на початку файлу авторські права. Особлива процедура установки не потрібна – файл cgi-lib.pl копіюється в каталог, де зберігаються CGI-сценарії і за допомогою команди require підключається до них:

```
require 'cgi-lib.pl';
```

Якщо у вас немає цього пакету, то просто використовуйте модуль CGI.pm у такому контексті:

```
use CGI qw/:ReadParse, PrintHeader, HtmlTop, HtmlBot, SplitParam/;
```

Далі створюються два сценарії, що генерують ті ж сторінки, що і згадувались раніше, але замість CGI.pm цього разу буде використаний пакет cgi-lib.pl. Попередні сценарії називались cgil.pl і cgi2.pl, а тепер мова піде про lib1.pl і lib2.pl.

Пакет cgi-lib.pl підключається до сценарію за допомогою команди require. (Не заборонено використовувати також use, але навряд чи вам зустрінеться такий варіант.) На відміну від модуля CGI.pm, кількість функцій, що генерують теги HTML в cgi-lib.pl, украй обмежено. Зазвичай ці теги доводиться виводити вручну. (Пакет cgi-lib.pl призначався насамперед для розбору посланих сценарію даних.) Втім, деякі



теги HTML все ж генеруються автоматично: підпрограма PrintHeader створює шапку HTML, необхідну для сторінки, розділ HtmlTop (мітки <HEAD> і <BODY>). Також з її допомогою можна створити заголовок сторінки, як показано в наступному прикладі, що задає сторінку із заголовком «My Web-Page»:

```
#!/usr/local/bin/perl
require 'cgi-lib.pl';
print &PrintHeader;
print &HtmlTop («My Web Page»);
```

Після опису початку сторінки решта розмітки HTML (у тому числі форми) створюється шляхом безпосереднього виведення тегів. Наприклад, ось так задається заголовок <H1>:

```
print «<CENTER><H1>Hello!</H1></CENTER>»;
```

Щоб прочитати дані, передані CGI-сценарію, використовується підпрограма ReadParse. Вона створює хеш (зазвичай має назву %in) і записує в нього значення елементів даних, переданих сценарію. Елементи даних адресуються по іменах, присвоєних відповідним елементам HTML. Наприклад, наступний код створює хеш %in і, читаючи дані текстового поля 'text', виводить їх:

```
&ReadParse(*in);
print «Here is the text: <EM>» $in('text'), «</EM>.»;
```

Щоб завершити Web-сторінку мітками HTML </BODY> і </HTML>, можна використовувати підпрограму HtmlBot (вона просто повертає рядок «</BODY>\n</HTML>\n»):

```
print &HtmlBot;
```

Ось так, коротко, і працює cgi-lib.pl.

Приклад. lib1.pl

```
#!/usr/local/bin/perl
#require 'cgi-lib.pl';
use CGI qw/:ReadParse, PrintHeader, HtmlTop, HtmlBot, SplitParam/;
print &PrintHeader;
print &HtmlTop («CGI Example Using cgi-lib.pl»);
print «<BODY BGCOLOR=\ white\ LINK=\ red><p>
<CENTER><H1>Here is the Survey!</H1></CENTER>
<H2>Please fill out our survey</H2>
Reasons for filling out our survey: <P><UL>
<LI>Fame</LI><LI>Fortune</LI><LI>Fun</LI></UL>
If you would rather not fill out our survey, you might be interested in <A
HREF=http://www.cpan.org>CPAN</A>
<HR><FORM METHOD=\POST\>
ACTION=\http://inna/scripts/web/16/lib2.pl\>
ENCTYPE=\application/x-www-form-urlencoded\>>
Please enter your name.
<INPUT TYPE=\text\> NAME=\text\> VALUE=\>><P>
Please enter your opinion:<P><TEXTAREA NAME=\textarea\>
```

```
ROWS=10 COLS=60>No opinion</TEXTAREA><P>
```

```
Please indicate what products you use <P>
```

```
<INPUT TYPE=\checkbox\> NAME=\checkboxes\>VALUE=\Shampoo\>>
```

Shampoo

```
<INPUT TYPE=\checkbox\> NAME=\checkboxes\>VALUE=\Toothpaste\>>
```

Toothpaste

```
<INPUT TYPE=\checkbox\> NAME=\checkboxes\> VALUE=\Bread\>>
```

CHECKED> Bread

```
<INPUT TYPE=\checkbox\> NAME=\checkboxes\> VALUE=\Cruise missiles\>>
```

CHECKED>Cruise missiles </P>

```
Please indicate your income level:<P><SELECT NAME=\list\> SIZE=4>
```

```
<OPTION VALUE=\Highest\>>Highest
```

```
<OPTION SELECTED VALUE=\High\>>High
```

```
<OPTION VALUE=\Medium\>>Medium
```

```
<OPTION VALUE=\Low\>>Low
```

```
</SELECT><P>
```

```
Please indicate your day of the week:<P>
```

```
<INPUT TYPE=\radio\> NAME=\radios\> VALUE=\1\> CHECKED>Sunday
```

```
<INPUT TYPE=\radio\> NAME=\radios\> VALUE=\2\>>Monday
```

```
<INPUT TYPE=\radio\> NAME=\radios\> VALUE=\3\>>Tuesday
```

```
<INPUT TYPE=\radio\> NAME=\radios\> VALUE=\4\>>Wednesday
```

```
<INPUT TYPE=\radio\> NAME=\radios\> VALUE=\5\>>Thursday
```

```
<INPUT TYPE=\radio\> NAME=\radios\> VALUE=\6\>>Friday
```

```
<INPUT TYPE=\radio\> NAME=\radios\> VALUE=\7\>>Saturday <P>
```

Thank you for filling out our Survey. Please indicate how much unsolicited mail you like to get.

```
<SELECT NAME=\popupmenu\>>
```

```
<OPTION VALUE=\Very much\>>Very much
```

```
<OPTION VALUE=\A lot\>>A lot
```

```
<OPTION VALUE=\Not so much\>>Not so much
```

```
<OPTION VALUE=\None\>>None</SELECT><P>
```

```
<INPUT TYPE=\hidden\> NAME=\hiddendata\> VALUE=\Rosebud\>>
```

```
<CENTER><INPUT TYPE=\submit\> NAME=\submit\>>
```

```
<INPUT TYPE=\reset\>></CENTER><HR></FORM>»;
```

```
print &HtmlBot;
```

Приклад. lib2.pl

```
#!/usr/local/bin/perl
```

```
#require 'cgi-lib.pl';
```

```
use CGI qw/:ReadParse, PrintHeader, HtmlTop, HtmlBot, SplitParam/;
```

```
print &PrintHeader;
```

```
print &HtmlTop («CGI Example Using cgi-lib.pl»);
```

```
print «<BODY BGCOLOR=\white\ LINK=\ red><p>
```

```
<CENTER><H1>Thank you for filling out our survey.</H1></CENTER>
```

```
<H3>Here are your responses...</H3>»;
```

```
if (&ReadParse(*in)) {print «Your name is: <EM>» $in{'text'}
«</EM>», «<p>», «Your opinions are. <EM>» $in{'textarea'},
«</EM>», «<p>», «You use these products. <EM>»
join(«, &SplitParam($in{'checkboxes'})), «</EM>.<p>»,
«Your income level is: <EM>», $in{'list'}
«</EM>.<p>Today is day <EM>» $in{'radios'},
«</EM> of the week.<p>», «How much uncolicted mail you like<EM>»
$in{'popupmenu'}, «</EM>», «<p>»
«The hidden data is <EM>» $in{'hiddendata'}, «</EM>.»; }
print &HtmlBot;
```

Приведемо список підпрограм, що входять до складу cgi-lib.pl:

- CgiDie – як і CgiError, друкує повідомлення про помилку і, крім того, зупиняє програму;
- CgiError – друкує повідомлення про помилку, використовуючи стандартні заголовки і HTML-код;
- HtmlBot – повертає рядок «</BODY>\n</HTML>\n»;
- HtmlTop – повертає розділ <HEAD> документа HTML і відкриває розділ <BODY>. Необов'язковий рядковий параметр використовується як назва Web-сторінки: додається тег HTML <H1> з цієї назвою;
- MethGet – повертає значення істина, якщо поточний виклик CGI зроблений за допомогою методу GET. Інакше повертається значення брехня;
- MethPost – повертає значення істина, якщо поточний виклик CGI зроблений за допомогою методу POST. Інакше повертається значення брехня;
- MyBaseUrl – повертає базову адресу (base URL) CGI-сценарію, без додаткового шляху або рядків запиту;
- MyFullUrl – повертає базову адресу (base URL) CGI-сценарію, включаючи додатковий шлях і рядки запиту;
- PrintEnv – форматує і друкує змінну середовища, доступну сценарію;
- PrintHeader – повертає рядок «Content-type: text/html\n\n». З неї повинні починатися всі Web-сторінки, створювані cgi-lib.pl;
- PrintVariables – форматує і друкує значення даних. Їй передається хеш або запис таблиці символів (для виведення елементів відповідного масиву). Без аргументів PrintVariables виводить вміст хешу %in;
- ReadParse – основна підпрограма бібліотеки cgi-lib.pl. Вона читає і розбирає дані, передані CGI-сценарію методами GET або POST. Зазвичай вона використовується для створення хеша %in: їй передається запис таблиці символів \*in. Хеш містить дані, передані сценарію, впорядковані по іменах відповідних елементів управління. Необов'язкові другий, третій і четвертий параметри, які вказують на те, що треба заповнити відповідні хеші даними з прийнятих файлів;
- SplitParam – розбиває параметр, що містить декілька значень, на список з одиничних параметрів. Ця підпрограма призначена для роботи з елементами HTML, здатними зберігати декілька значень, наприклад, групою кнопок.

**Захист CGI.** Забезпечення безпеки завжди було серйозною проблемою. В наші дні вона ще актуальніша, оскільки по мірі розвитку операційних систем стає все складнішим і складніше затицати проломи в захисті. Тому на Unix-системах CGI-сценарії зазвичай запускаються від імені ідентифікатора користувача «nobody» («ніхто»). Такий процес має мінімум привілеїв. Вважалося, що процес, що має мінімум привілеїв, принесе менше шкоди. Проте і до цього дня можуть виникати проблеми – зокрема, із-за неакuratності в CGI-сценаріях. Нижче розповідається, як обійти деякі найбільш вірогідні неприємності.

Ось декілька Web-сторінок, присвячених безпеці CGI, які я рекомендував би прочитати до того, як ви починатимете створювати для широкого використання щонебудь серйозніше за прості CGI-сценарії:

- сторінка WWW-консорціума, присвячена безпеці CGI (The World Wide Web Consortium's CGI security page), – [www.w3.org/Security/Faq/www-security-faq.html](http://www.w3.org/Security/Faq/www-security-faq.html);
- частина збірки питань і відповідей (FAQ) по CGI-програмуванню на Perl, присвячена проблемам безпеки – [www.perl.com/CPAN-local/doc/FAQs/cgi/perl-cgi-faq.html](http://www.perl.com/CPAN-local/doc/FAQs/cgi/perl-cgi-faq.html);
- сторінка Селени Сол (Selena Sol), що розповідає, як ви ризикуєте при установці чужих сценаріїв – [Stars.com/Authoring/Scripting/Security](http://Stars.com/Authoring/Scripting/Security);

Наступний крок – безпосереднє вивчення коду. У прикладах ви знайдете інформацію про безпеку і про те, як писати CGI-сценарії для лічильників і гостей книг.

Серйозно беремося за захист. CGI-сценарії можуть породжувати безліч потенційних проломів у безпеці. Як граничний випадок розглянемо сценарій, що запускає програми, імена яких передаються йому як аргумент. Дані форм HTML посилаються у вигляді рядків, причому як роздільник аргументів використовується знак питання. Рядок даних записується в кінці URL, який означає, що якщо Ви хочете просто запустити сценарій Perl, URL повинен виглядати, наприклад, так:

<http://www.yourserver.com/user/perl.exe?script.pl>

Але, якщо хакер побачить, що Ви використовуєте техніку на зразок цієї, він може надіслати власний рядок такого вигляду:

<http://www.yourserver.com/user/perl.exe?-e+ nasty commands>

В результаті він зможе виконати будь-які команди Perl, що навряд чи Вас порадує. Цей приклад вказує на одну з найбільших небезпек CGI-сценаріїв, написаних на Perl – виклики зовнішніх програм без перевірки коду, що передається в кінці рядка.

У Perl зовнішні програми викликаються багатьма способами, наприклад за допомогою рядка, вставленого в зворотні апострофи (backtics), викликів system або exec. Навіть оператори eval вимагають обережного звернення. Дуже важливо настроїти CGI так, щоб не можна було легко зробити нічого небезпечного.

Насправді в Perl існує чудовий механізм безпеки, призначений для латання дірок подібного типу – мічені дані. Якщо дозволено відстежування даних, Perl не дозволяє передавати, що прийшли ззовні дані функціям system, exec і так далі. Просте правило, що дозволяє забезпечити безпеку – ніколи не передавати неперевірені дані зовнішній програмі і завжди прагнути обійтися без запуску командної оболонки.

Якщо ж це неможливо, слід завжди перевіряти аргументи на предмет наявності метасимволів командної оболонки і, принаймні, видалення їх. Ось метасимволи командної оболонки Unix:

```
&,"\"`*?`<^(){$_n\r
```

Ще одне важливе зауваження: будьте особливо уважні до прав доступу до файлів, щоб їх не можна було поміняти.

І, звичайно ж, звичні обмеження: не посилайте паролі по електронній пошті, не набирайте їх при роботі з безкоштовними утилітами. Не залишайте ваш рахунок в системі (account) на довгий час невживаним – хакери стежать за такими речами, щоб отримати контроль над ними. Не дозволяйте CGI-сценаріям отримувати дуже багато системної інформації.

Працюємо з міченими даними. Однією з найбільших дір в захисті CGI-сценаріїв є передача неперевіраних даних командному інтерпретатору. У Perl для запобігання таким ситуаціям можна використовувати механізм мічених даних (tainted data). В цьому випадку будь-які змінні, пов'язані з даними, отриманими ззовні (включаючи змінне середовище, стандартний потік введення і командний рядок), вважаються за мічені. Поки вони залишаються такими, їх не можна використовувати для чого б то не було за межами вашої програми. Якщо мічена змінна використовується для установки іншої змінної, то остання також стає міченою, а це означає, що помічені (або «забруднені») дані можуть поширюватися за програмою скільки завгодно далеко і скільки завгодно складними шляхами, але вони все одно будуть акуратно помічені.

Підказка. Цей механізм працює тільки для скалярних значень. Деякі елементи масиву можуть бути міченими, тоді як останні – ні.

Загалом, мічені дані не можуть бути використані за допомогою викликів eval, system, exec. Perl стежить за тим, щоб вони не потрапили в команди, що викликають оболонку, в команди, що модифікують файли, каталоги або процеси. Проте є одне важливе виключення: якщо викликом system або eval передається список аргументів, він не перевіряється на наявність мічених елементів. Якщо Ви спробуєте провести яку-небудь операцію з міченими даними за межами програми, Perl зупиниться із застережливим повідомленням. У режимі мічених даних Perl припиняє роботу також в разі виклику зовнішньої програми без попередньої установки змінного середовища PATH. У Perl для включення відстежування мічених даних використовується спеціальна версія інтерпретатора, що зветься taintperl:

```
#!/usr/local/bin/taintperl
```

Проте у версії 5 перевірка мічених даних включена до складу Perl, і ви можете включити її, передавши інтерпретатору Perl ключ -T:

```
#!/usr/local/bin/perl -T
```

У наступному прикладі включається відстежування мічених даних, але програма не робить нічого небезпечного – відповідно, проблем немає:

```
#!/usr/local/bin/perl -T
print «Hello!\n»;
```

Проте при виконанні потенційно небезпечних операторів типу system при включеній перевірці мічених даних Perl повідомить про можливий пролом в захисті, обу-

мовлений використанням даних околу. Навіть, якщо Ви не використовуєте PATH при виклику зовнішньої програми, не виключено, що його використовує програма, яка викликається. Ось повідомлення про помилку, яке Ви побачите:

```
#!/usr/local/bin/perl -T
print system( 'date' );
Insecure $ENV{PATH} while running with -T switch at taint.pl
line 5, < chunk 1
```

Щоб виправити це Ви можете при включеній перевірці мічених даних самостійно встановити \$ENV{'PATH'}:

```
#!/usr/local/bin/perl -T
$ENV{'PATH'} = '/bin:/usr/bin:/usr/local/bin';
print system( 'date' )
Thu Nov 12 19-55 53 NSK
```

Ще один приклад, в якому робиться спроба передати системному виклику мічені дані. Навіть якщо \$ENV{'PATH'} встановлюється в програмі, сценарій все одно припиняє роботу, оскільки намагається передати мічені дані операторові system:

```
#!/usr/local/bin/perl -T
$ENV{'PATH'} = '/bin:/usr/bin:/usr/local/bin';
while(< >){
    $command = $_;
    system($command); }
Insecure dependency in system while running with -T switch at taint.pl line 5, <
chunk 1
```

Дані, навіть будучи переданими в \$command з \$\_, все одно вважаються міченими. Як очистити дані, якщо ви упевнені в них?

Очищення даних. Єдиний спосіб очистити мічену змінну – використовувати шаблони, по яких з неї вибираються підрядки. У наступному прикладі передбачається, що мічена змінна \$tainted містить електронну адресу. Ми можемо витягувати її і зберегти, як не мічену в іншій змінній таким чином:

```
$tainted =~ /(^[w]+)\@(^[w]+)/
$username = $1;
$domain = $2;
print «$username\n»
print «$domain\n»;
```

Таким чином, ми витягували безпечні дані. Тобто спосіб створення «чистих» даних полягає у витяганні з мічених даних підрядків, які апіорі безпечні (і, звичайно ж, не містять метасимволи командного інтерпретатора).

Створюємо лічильник відвідин. Створення лічильника відвідин – досить просте завдання: Ви просто повинні зберігати поточне значення лічильника у файлі і показувати його при необхідності. Наведемо приклад створення лічильника counter.pl.

Підказка. Відмітимо, що цей лічильник просто виводить поточне значення як текстовий рядок, але в принципі можна зробити цікавіші речі, наприклад, створити графічний лічильник, маючи набір файлів із зображеннями цифр і виводячи їх один за

іншим на Web-сторінці. Можна також відтворювати цифри за допомогою тега HTML `<IMG>`, якщо встановити атрибут SRC відповідно до URL-сценарію, що виводить цифри.

Сценарій називається counter.pl, який приведений далі в прикладі. Щоб він працював, в тому ж каталозі, що і counter.pl, повинен знаходитися файл counter.dat. Для початку відліку потрібно записати 0 (нуль) в counter.dat.

Приклад. counter.pl

```
#!/usr/bin/perl
use CGI;
$co = new CGI;
open (COUNT, «<counter.dat»)
or die «Could not open counter data file!»;
$count = <COUNT>;
close COUNT;
$count++;
open (COUNT, «>counter.dat»);
print COUNT $count;
close COUNT;
print
$co->header
$co->start_html(
-title=>'Counter Example'
-author=>'Volodymyr'
-BGColor=>'white'),
$co->center($co->h1('Counter Example')) $co->p,
$co->center($co->h3('Current count ', $count)) $co->p,
$co->center($co->h3('Reload the page to update the count'))
$co->end_html;
```

Цей сценарій дуже простий: все, що він робить – це читає число, що зберігається в counter.dat, збільшує його на одиницю, записує назад в counter.dat і потім показує збільшений лічильник.

Створюємо гостьову книгу. Створення гостьової книги – крок вперед в порівнянні з лічильником. Гостьова книга збирає коментарі користувачів і зберігає їх у файлі, що зазвичай має формат HTML, щоб потім виводити їх на сторінці. Наша гостьова книга використовує три файли, що зберігаються в одному каталозі: guestbook.htm, guestbook.pl і book.htm в подальших прикладах. Перший є обличчям гостьової книги, тобто саме ця сторінка вказує користувачеві, що він може додати запис в книгу відвідувачів. Вона отримує ім'я користувача і коментар. Коли користувач натискає на кнопку підтвердження, дані посилаються сценарію guestbook.pl; іншими словами, якщо Ви використовуєте цей сценарій, то слід змінити вказаний URL в guestbook.htm на реальний URL guestbook.pl:

```
<BODY><H1>Please add to my guestbook</H1>
<FORM METHOD=POST ACTION=
«http://www.yourself.com/cgi/guestbook.pl»>
```

У guestbook.pl (дивіться далі приклади) ми відкриваємо гостьову книгу, що зберігається у файлі book.htm. Основна ідея – додати в неї ім'я користувача і його коментар, але book.htm закінчується тегами `</BODY></HTML>`. Тому спочатку треба встановити покажчик файлу перед цими словами за допомогою наступного коду:

```
open (BOOK, «>book.htm») or die «Could not open guestbook!»;
seek (BOOK, -length($co->end_html), 2);
```

Оскільки рядки `</BODY></HTML>` в даному випадку створюються за допомогою CGI-методу end\_html, ми відкочуємося назад на довжину рядка, яка генерується, що дозволяє нам не залежати від того, що метод end\_html виводитиме в наступних версіях модуля CGI.pm.

Після цього код записує замість тегів `</BODY></HTML>` нові дані, додаючи в кінці ті ж теги викликом CGI-методу end\_html. Потім guestbook.pl створює сторінку, на якій розташовується подяка користувачеві за коментарі і гіперпосилання, що дозволили проглянути вміст гостьової книги. Іншими словами, якщо Ви використовуєте цей сценарій, то слід змінити URL, приведений в прикладі, на реальний URL book.htm (переконавшись, що права доступу для цього файлу досить низькі, щоб guestbook.cgi міг записувати в нього дані):

```
<If you want to take a look at the guest book <
$co->a({href=> «http://www.yourserver.com/cgi/book.htm»},
«click here»),»»;
```

Якщо користувач клацає на гіперпосиланні, відкривається гостьова книга, а посилання на неї можна розташувати на будь-якій іншій Web-сторінці Вашого розділу. Ім'я користувача і коментарі відображаються в гостьовій книзі разом з часом додавання запису. Файл guestbook.pl приводить в безпечний стан будь-який код HTML, який користувач може спробувати ввести в гостьову книгу, замінюючи будь-які символи `<` HTML-кодом `&lt;` (це робиться так: \$username =~ s/</&lt; і \$text =~ s/</&lt;), який виводить `<`, щоб не дозволяти браузеру розібрати коментарі користувача як HTML. Це означає, що будь-який код HTML, який користувач спробує ввести в гостьову книгу, буде виведений як текст і не виконуватиметься.

Відмітимо, що Ви можете настроїти guestbook.pl так, щоб він приймав електронні адреси відвідувачів (втім, все більше і більше користувачів не бажають залишати свої адреси не стільки з міркувань секретності, скільки із-за програм, які сканують мережу у пошуках адрес електронної пошти, а потім продають отримані списки розповсюджувачам реклами). Ви також можете видозмінити файл гостьової книги book.htm, додавши графіку з потужністю тега HTML `<IMG>`, встановивши фонове зображення і так далі. Просто потрібно стежити, щоб останнім, що Ви виводите в book.htm був текст `</BODY></HTML>` (або виведення поточної версії CGI-методу end\_html, оскільки в новій версії пакету CGI.pm він може змінитися), щоб guestbook.pl міг відкочуватися на необхідне число символів і замінити ці теги новим коментарем.

Підказка. Якщо Ви не хочете залежати від версії модуля CGI.pl, потрібно записувати в окремий файл довжину тексту, виведеного методом end\_html при останньому записі в гостьову книгу і використовувати для установки вказівника це значення, а не довжину рядка, end\_html, що виводиться.

Приклад. guestbook.htm

```
<HTML><HEAD><TITLE>Add to the guest book</TITLE></HEAD><BODY>
<H1>please add to my guestbook</H1>
<FORM METHOD = POST ACTION =
http://www.yourself.com/cgi/guestbook.pl >
<BR><CENTER>Please enter your name<P>
<INPUT TYPE = «TEXT» NAME =>username»></INPUT>
<BR>Please enter your comments
<TEXTAREA ROWS      = 8 COLS = 40 NAME = «comments»>
</TEXTAREA><BR><BR>
<INPUT TYPE =SUBMIT VALUE =Send><INPUT type = RESET VALUE
=Reset>
</CENTER></FORM>
</BODY></HTML>
```

Приклад. guestbook.pl

```
#!/usr/bin/perl
```

```
use CGI; $co = new CGI;
```

```
open (BOOK «>>book.htm») or die «Could not open guestBOOK»;
```

```
seek (BOOK -length($co->end_html), 2)
```

```
$date = date; chop($date);
```

```
$username = $co->param(«username»); $username => s/<l&lt;/;
```

```
print BOOK
```

```
$co->h3(«New comments by :, $username, «on «,
```

```
$date,$co->p,$text),
```

```
$co->hr $co->end_html;
```

```
close BOOK;
```

```
print $co->header, $co->start_html(
```

```
-title=> «Guest Book Example»
```

```
-author=> «Volodymyr»
```

```
-BGCOLOR=> «white»
```

```
-LINK=> «red|»);
```

```
print
```

```
$co->center($co->h1('Thanks for adding to the guest book!'))
```

```
«If you want to take a look at the guest book «,
```

```
$co->a({href=> http://www.yourserver.com/cgi/book.htm}),
```

```
« click here»),»»,»»,
```

```
$co->hr $co->end_html;
```

Приклад. book.htm

```
<HTML><HEAD><TITLE>The Guest Book</TITLE></HEAD>
```

```
<BODY><CENTER>
```

```
<H1>Here is the guest book </H1><HR>
```

```
</BODY></HTML>
```

### 2.3.2. Основи розробки сценаріїв на мові PHP

PHP – сценарії можуть розміщуватися в окремому файлі (з розширенням .php) або вбудовуються безпосередньо в HTML документ.

Існує декілька способів введення коду PHP в HTML документи:

- за допомогою відкриваючого тегу <?php і закриваючого тегу ?>;
- за допомогою коротких тегів <? і ?>. Дана можливість доступна тільки при спеціальному налаштуванні;
- за допомогою тегів <Script language=>php> і </script>;
- шляхом використання echo тегів в стилі ASP: <% і %>. Така можливість доступна при відповідному конфігураційному налаштуванні.

Надалі в прикладах використовуватиметься перший з варіантів впровадження PHP коду.

Код, який знаходиться в середині вказаних тегів, обробляється інтерпретатором PHP, решта всього коду залишається незмінним.

Для того, щоб побачити поточні налаштування PHP, і для перевірки його працездатності корисно використовувати спеціальну функцію phpinfo().

Приклад. Використання php-функції phpinfo() (html txt):

```
<?php
phpinfo();
?>
```

Після виконання цього коду у веб-браузері можна буде побачити Рис. 2.3.2.1. (показана невелика частина).

Directive	Local Value	Default Value
allow_call_time_pass_reference	On	On
allow_url_fopen	On	On
always_populate_raw_post_data	Off	Off
arg_separator_input	&	&
arg_separator_output	&	&
asp_tags	Off	Off
auto_append_file	no value	no value
auto_globals_jit	On	On
auto_prepend_file	no value	no value
browscap	no value	no value
default_charset	no value	no value
default_mimetype	text/html	text/html
define_syslog_variables	Off	Off
disable_classes	no value	no value
disable_functions	no value	no value
display_errors	On	On
display_startup_errors	Off	Off

Рис. 2.3.2.1. Відображення інформації php-функції phpinfo

Загалом приклад містить інформацію про встановлені опції і розширення PHP, версію PHP, інформацію про веб-сервер і змінні оточення, інформацію про версію ОС, шляхи, налаштування конфігураційних змінних, поля заголовку HTTP і PHP ліцензії.

Наступний приклад демонструє варіант з впровадженням PHP коду в HTML (html.txt):

```
<html>
<body>
<p>Hello! </p>
<p>Today is:
<?php
    $today = date(«F j, Y, g:i a»);
    echo($today);
?>
</p>
</body>
</html>
```

Результат обробки цього документа представлений на рисунку Рис. 2.3.2.2.

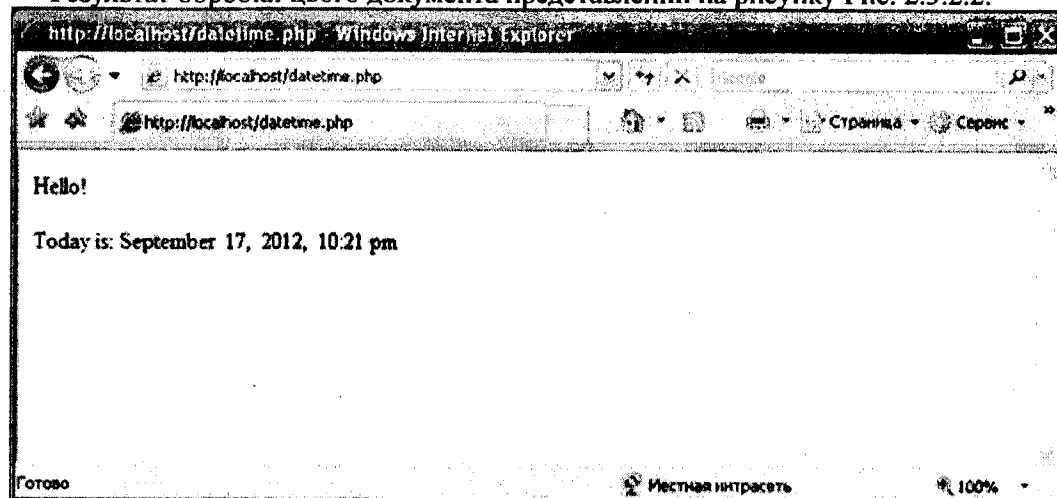


Рис. 2.3.2.2. Відображення прикладу впровадженням PHP коду в HTML

Одна з головних задач, що вирішуються за допомогою PHP, є обробка даних, що отримуються від користувача через веб-форми. Розглянемо, яким чином в PHP реалізується така обробка.

Приклад. Виведення списку параметрів, отриманих сервером в запиті від клієнта (html.txt).

```
<?php
$method = $_SERVER[«REQUEST_METHOD»];

if ($method == «GET») $query = «_GET»;
elseif ($method == «PUT») $query = «_PUT»;
else die(«$method is not supported!»);
```

```
print «<p><b>Method</b>: $method </p>»;
print «<p><u>Params:</u></p>»;
```

```
foreach ($$query as $name => $value)
{
    print «<b>$name</b> = <i>$value</i> <br>»;
}
?>
```

У першому рядку сценарію за допомогою змінної оточення REQUEST\_METHOD з глобального асоціативного масиву \$\_SERVER визначається метод передачі даних в запиті клієнта. Залежно від вибраного методу передані дані витягуються або з глобальної змінної \$\_GET або з \$\_PUT. Якщо метод відрізняється від GET або PUT, або невизначений, то відбувається примусове завершення роботи сценарію з видачею повідомлення через виклик функції die(). У принципі, можна також використовувати глобальний масив \$\_REQUEST, що містить у середині себе масиви \$\_GET, \$\_POST і \$\_COOKIE, що дозволяє позбавитися від перевірки методу передачі.

Конструкція вигляду \$\$query демонструє непряме звернення до змінної, тобто змінна \$query містить ідентифікатор іншої змінної і для звернення до неї необхідно додати ще один знак \$.

Обидві змінні \$\_GET і \$\_POST є асоціативними масивами, тому для перебору елементів був використаний спеціальний оператор.

```
foreach (ім'я_масиву as ключ => значення)
```

Слід звернути увагу на те, що в середині рядкових констант, які знаходяться в лапках – « » можна вставляти змінні. Після обробки такого рядка інтерпретатором замість змінної вставляється її фактичне значення. Також для конкатенації рядків можна використовувати оператор «.». В цілому можна сказати, що в PHP можна використовувати оператори розгалуження, вибору і циклів, які аналогічні тим, що використовуються в мові C.

Код сценарію, що реалізовує чотири арифметичні операції над цілими числами представлений в прикладі нижче.

Приклад. Калькулятор арифметичних операцій для цілих операндів (html.txt).

```
<?php
$method = $_SERVER[«REQUEST_METHOD»];

if ($method == «GET») $query = «_GET»;
elseif ($method == «PUT») $query = «_PUT»;
else die(«$method is not supported!»);

$х = $$query;

$a = $х[«A»];
$b = $х[«B»];
$op = $х[«op»];

switch ($op)
```



```

{
case '+': $result = (int)$a + (int)$b; break;
case '-': $result = (int)$a - (int)$b; break;
case '*': $result = (int)$a * (int)$b; break;
case '/':
{
if ($b == '0') die(«divide by zero!»);
}
default: die(«operator $op is not defined»);
}
print «<p>Result: $a $op $b = $result </p>»;
print «<p><a href=».$_SERVER['HTTP_REFERER'].»>Back</a></p>»;
?>

```

У PHP є широкий діапазон функцій для роботи з файлами. Наприклад, наступний приклад демонструє читання файлу, в якому знаходиться виконуваний код PHP.

Приклад. Читання файлу PHP, що містить код виконуваного сценарію (html.txt).

```

<?php
$fh = fopen(«read.php»,»r»);
if (!$fh) die(«Cannot open file»);
while (!feof($fh))
{
$line = fgets($fh);
echo $line.»<br>»;
}
fclose($fh);
?>

```

У даному прикладі використовуються практично ті ж самі функції, що і в мові C:

fopen(шлях\_до\_файлу, тип\_доступу) – відкриття файлу;

feof(вказівник\_на\_файл) – перевірка на наявність ознаки кінця файлу;

fgets(вказівник\_на\_файл) – читання рядка з файлу;

fclose(вказівник\_на\_файл) – закриття файлу.

Для запису даних у файл можна використовувати функцію fputs(вказівник\_на\_файл, рядок). У наступному прикладі згенеровані функцією rand() псевдовипадкові числа зберігаються у файлі rand.dat.

Приклад. Збереження у файлі послідовності псевдовипадкових чисел (html.txt).

```

<?php
$n = 10;
$fh = fopen(«rand.dat»,»w»);
if (!$fh) die(«Cannot open file»);
srand();
for ($i=0; $i < $n; $i++)

```

```

{
    $d = rand(0,100);
    fputs($fh,»$d\n»);
}

fclose($fh);
?>

```

Перед виконанням даного сценарію слід правильно налаштувати права доступу для веб-сервера до каталогу.

## ТЕМА 2.4. ОСНОВИ РОЗРОБКИ ВЕБ-ЗАСТОСУВАНЬ З ДОПОМОГОЮ ASP.NET

- 2.4.1. Основи ASP.NET
- 2.4.2. Серверні елементи управління ASP.NET
- 2.4.3. Основи розробки веб-застосовувань з допомогою J2EE

### 2.4.1. Основи ASP.NET

Платформа .NET Framework надає можливість розробки і інтеграції веб-застосовувань. ASP.NET є однією із складових інфраструктури .NET Framework і фактично є платформою для створення веб-застосовувань і веб-сервісів, IIS, що працюють під управлінням.

ASP.NET зовні багато в чому нагадує старішу технологію ASP, але в той же час внутрішній устрій ASP.NET істотно відрізняється від ASP. Компанія Microsoft ASP.NET побудована на базі CLR (Common Language Runtime), який є основою всіх застосовувань .NET. Розробники можуть створювати код для ASP.NET, використовуючи мови програмування, що входять в .NET Framework: C#, Visual Basic.NET, JScript.NET та інші.

Розглянемо детальніше, чим відрізняється ASP.NET від ASP.

Класичний ASP має наступні недоліки:

- використовуються тільки мови сценаріїв, які дають великий програш в продуктивності (через інтерпретованість) і не підтримують багато можливостей об'єктно-орієнтованого програмування;
- логіка застосування (у вигляді коду HTML) не відокремлена від бізнес-логіки (виконуваного коду), що приводить до перемішування в одному файлі коду HTML з кодом сценарію;
- неможливо повторно використовувати готові рішення в інших проектах (можливе тільки копіювання коду сценаріїв).

У файлах ASP.NET включається код на таких мовах програмування як C#, JScript.NET, VisualBasic.NET, що дозволяє застосовувати безпосередньо у веб-застосуваннях можливості об'єктно-орієнтованого програмування. Також істотно скорочується об'єм коду, написаного вручну за рахунок застосування серверних об'єктів, що автоматично генерують код елементів управління HTML. Можливе використання стандартного середовища розробки Visual Studio.NET, тобто ASP.NET має перевагу в швидкості в порівнянні зі сценарними технологіями, оскільки при першому зверненні код компілюється і поміщається в спеціальний кеш, а згодом тільки виконується, не вимагаючи витрат часу на парсинг, оптимізацію і так далі.

Не дивлячись на можливість спільної роботи ASP і ASP.NET на одному веб-сервері, вони не можуть використовувати спільний сеанс. Файли ASP.NET обробляються бібліотекою aspnet\_isapi.dll (а не asp.dll), яка, у свою чергу, використовує для виконання коду технологію .NET.

Бібліотека базових класів .NET містить простори імен 3 основних груп:

- ⇒ елементи веб-застосовувань (протоколи, безпека і ін.);
- ⇒ елементи графічного інтерфейсу (WebForms);

⇒ *web-служби*.

Як вже згадувалося раніше, ASP.NET використовує можливості стандартного середовища розробки Visual Studio.Net і, зокрема, класи бібліотеки FCL (Framework Class Library).

Розробникові веб-застосовувань на ASP.NET доступні класи, що входять в наступні простори імен:

Простір імен	Вміст
System.Web	Організація взаємодії web-клієнта (браузера) з web-сервером (запит-відповідь, cookie і і ін.)
System.Web.Caching	Підтримка кешування при роботі web-застосовувань
System.Web.Configuration	Налаштування web-застосування відповідно до файлів конфігурації проекту
System.Web.Security	Реалізація системи безпеки web- застосовувань
System.Web.Services	Організація роботи web-сервісів
System.Web.Services.Description	
System.Web.Services.Discovery	
System.Web.Services.Protocols	
System.Web.UI	Побудова графічного інтерфейсу користувачів web- застосовувань
System.Web.UI.WebControls	
System.Web.HtmlControls	

У свою чергу простір імен System.Web включає простори імен, назви яких знайомі розробникам веб-застосовувань на ASP:

Простір імен	Вміст
HttpApplication	Даний клас визначає спільні члени для всіх web-застосовувань
HttpApplicationState	У даному класі міститься спільна інформація web-застосування для множини запитів, сеансів і каналів передачі даних
HttpBrowserCapabilities	Цей клас використовується для отримання інформації про можливості клієнтського браузера, що звертається до web-серверу
HttpCookie	Підтримка механізму безпечної роботи з об'єктами HTTP cookie
HttpRequest	Надає доступ до інформації, переданої web-клієнтом
HttpResponse	Використовується для формування HTTP-відповіді сервера

В основу розробки веб-застосовувань на ASP.NET покладена модель розділення коду представлення і коду реалізації, що рекомендується Microsoft при створенні динамічних документів за допомогою програмних кодів. Це робиться шляхом розміщення програмного коду або в окремий файл, або у середині спеціального тегу для сценаріїв. Файл такого роду, зазвичай, має розширення \*.aspx.cs (\*.aspx.vb) і має ім'я, що співпадає з ім'ям основного файлу ASPX. В принципі такий підхід дозволяє веб-дизайнерам концентруватися на роботі з кодом розмітки документу із мінімальними змінами програмного коду, в звичайному ASP впроваджуваної безпосередньо в код розмітки.

Взаємодія користувача з веб-застосуванням, реалізованим на ASP.NET включає наступні процеси:

- при запиті сторінки ASPX ініціюється подія Page\_Init, що проводить початкову ініціалізацію сторінки і її об'єкту;
- далі ініціюється подія Page\_Load, яка може бути використана, наприклад, для установки початкових значень для елементів управління. При цьому також можна визначити чи була завантажена сторінка вперше або звернення до неї здійснюється повторно в рамках зворотного відсилання у відповідь на подію, пов'язані з елементами управління, розміщеними на сторінці, тобто перевірити властивість Page.IsPostBack;
- далі виконується перевірка валідності елементів сторінки з погляду коректності введених користувачем даних;
- і, нарешті, слідує обробка всіх подій, пов'язаних з діями користувача з моменту останнього зворотного відсилання.

Для збереження даних веб-сторінки в проміжках між зверненнями до неї в ASP.NET використовуються стани відображення (view state).

Якщо дані, введені у веб-форму, необхідно зробити доступними іншим веб-формам того ж застосування, то ці дані необхідно зберегти в об'єктах Application і Session. Об'єкти Application доступні всім користувачам застосування і можуть розглядатися як глобальні змінні, звернення до яких можливе з будь-яких сеансів. Об'єкти Session доступні тільки в рамках одного сеансу і тому вони виявляються доступними тільки одному користувачу.

### 2.4.2. Серверні елементи управління ASP.NET

Важливою особливістю ASP.NET є використання серверних елементів управління на веб-сторінці (елементи WebForm), які є фактично тегами, зрозумілими веб-серверу. Ці елементи визначені в просторі імен System.Web.UI.WebControls.

Прийнято виділяти три типи серверних елементів управління:

- ⇒ серверні елементи управління HTML – звичайні теги HTML;
- ⇒ елементи управління веб-серверу – нові теги ASP.NET;
- ⇒ серверні елементи управління для перевірки даних (валідації) – застосовуються для валідації вхідних даних від клієнтського застосування (зазвичай веб-браузера).

Переваги від використання таких елементів при розробці веб-застосувань:

- скорочується кількість коду, написаного вручну (що особливо помітно для складних елементів документу). Елемент просто «перетягується» з панелі інструментів, після чого виконується налаштування його параметрів в спеціальному вікні. При цьому всі зміни автоматично заносяться безпосередньо в \*.aspx файл;
- з програмної точки зору кожному з цих елементів управління відповідає певний клас в бібліотеці базових класів .NET, що дозволяє писати для них такий же код як і для будь-яких інших класів;
- для будь-якого елементу управління WebForm визначено набір подій, що обробляються на веб-сервері;
- для будь-якого елементу управління WebForm надається можливість для перевірки введення даних користувачем.

За замовчуванням серверні елементи управління HTML в ASP.NET файлах розглядаються як текст. Для їх програмування потрібне додавання атрибуту runat=»server» у відповідний елемент HTML. Крім того, всі серверні елементи управління HTML мають бути розміщені у середині зони дії тегу <form>, що також має атрибут runat=»server».

Подібно до серверних елементів управління HTML-елементи управління веб-серверу також створюються на веб-сервері і передбачають додавання атрибуту runat=»server». Проте вони можуть і не відповідати конкретним елементам HTML, але представляти складніші елементи.

Загальний синтаксис для опису таких елементів:

```
<asp:тип_елементу id=»ідентифікатор» runat=»server»/>
```

Серверні елементи валідації застосовуються для перевірки, що вводяться користувачем даних. Вони мають наступний синтаксис:

```
<asp:тип_елементу id=»ідентифікатор» runat=»server» />
```

У ASP.NET використовуються два елементи управління WebForm для управління відображенням даних, що отримуються з джерела даних:

- ⇒ DataGrid – елемент управління, що відображає вміст об'єкту ADO.NET DataSet у вигляді таблиці;
- ⇒ DataList – елемент управління для вибору значень, що заповнюються з джерела даних.

Якщо необхідно відображати дані, отримані по запиту користувача з джерела даних у вигляді таблиці на веб-сторінці, то ASP.NET надає в розпорядження веб-програмісту зручний елемент управління DataGrid.

### 2.4.3. Основи розробки веб-застосувань з допомогою J2EE

У даному пункті теми розглянемо техніку розробки компонентів Web-застосувань на основі платформ J2EE і .NET. Обидві платформи надають спеціальну підтримку для розробки компонентів на двох рівнях: рівні інтерфейсу користувача (WEBUI) і рівні зв'язку з даними.

Призначений для користувача інтерфейс Web-застосувань заснований на генерації динамічних сторінок HTML, що містять дані, які запрошує користувач. Рівень моделі даних надає застосуванню можливість працювати з даними, що зазвичай зберігаються у вигляді набору таблиць і зв'язків між ними, як з набором зв'язаних об'єктів.

Основні відмінності між технікою розробки компонентів цих двох рівнів, використовуваних в рамках J2EE і .NET можна сформулювати таким чином:

- у J2EE компоненти EJB призначені не лише для представлення даних застосування у вигляді об'єктів, але і для реалізації його бізнес-логіки, тобто об'єктів наочної області і основних способів роботи з ними;
- у .NET немає спеціально виділеного виду компонентів, призначених для реалізації бізнес-логіки. Вона може реалізовуватися за допомогою звичайних класів, що часто зручніше;
- EJB-компоненти є узгодженим з об'єктно-орієнтованим підходом представленням даних застосування. Робота з ними організовується так само, як з об'єктами звичайних класів (з точністю до деяких деталей).

У .NET-застосуваннях всі пропонувані способи представлення даних є об'єкними обгортками навколо реляційного представлення – в будь-якому разі доводиться працювати з даними як з набором таблиць. У .NET немає автоматичної підтримки їх перетворення в систему взаємозв'язаних об'єктів і назад.

Рівень бізнес-логіки і моделі даних в J2EE. В рамках застосувань, побудованих по технології J2EE, зв'язок з базою даних і бізнес-логікою, прихованою від користувача, прийнято реалізовувати за допомогою компонентів Enterprise JavaBeans.

Можливі й інші способи реалізації цих функцій. Наприклад, бізнес-логіка може бути реалізована безпосередньо в методах об'єктів, призначених для користувача інтерфейсу, а обмін даними з базою даних – через інтерфейс JDBC. При цьому втрачається можливість перевикористання функцій бізнес-логіки в різних застосуваннях на основі єдиної бази даних, а також стає неможливим використання автоматичних транзакцій при роботі з даними. Транзакції в цьому випадку потрібно організовувати за допомогою явних звернень до JTA.

Для розробки набору компонентів EJB потрібно, по-перше, для кожного компоненту створити один або декілька класів і інтерфейсів Java, що забезпечують реалізацію самої функціональності компоненту та визначення інтерфейсів для видалених звернень до нього, і, по-друге, написати дескриптор розгортання – XML-файл, що описує наступне:

- ⇒ набір EJB-компонентів застосування;
- ⇒ сукупність елементів коду на Java, створюючих один компонент;
- ⇒ зв'язок властивостей компоненту з полями таблиць БД і зв'язками між таблицями;
- ⇒ політику компонентів і їх методів по відношенню до транзакцій;
- ⇒ набір ресурсів, якими компоненти можуть користуватися в своїй роботі.

Схема життєвого циклу компоненту даних показана на рис. 2.4.3.1.

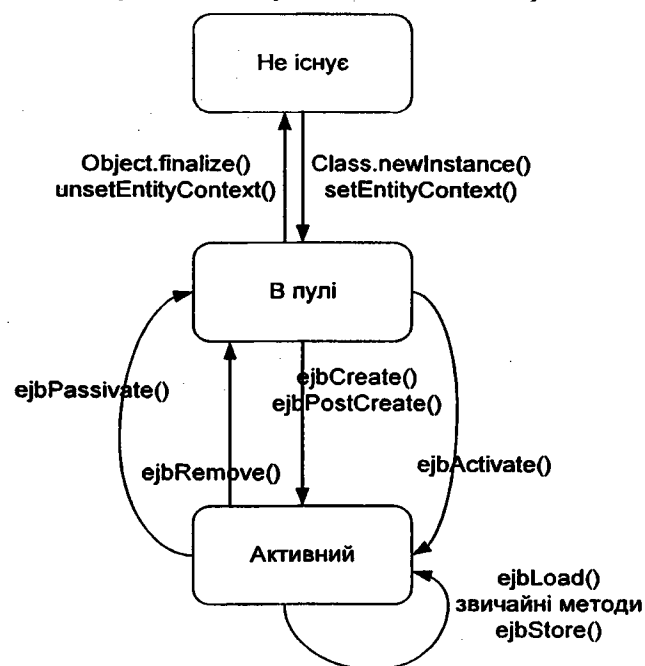


Рис. 2.4.3.1. Життєвий цикл EJB компоненту даних.

Життєвий цикл сеансового компоненту розрізняється залежно від того, чи підтримує компонент стан сеансу або ні.

Схема життєвого циклу сеансового компоненту із станом показана на рис. 2.4.3.2. Різниця від життєвого циклу компоненту даних єдина – метод `ejbCreate()` відразу переводить компонент в активний стан.

Життєвий цикл сеансового компоненту без стану набагато простіший. Його схема представлена на рис. 2.4.3.3. Клас первинного ключа (primary key class) декларується тільки для компонентів даних, якщо в цій якості не можна використовувати відповідний бібліотечний клас. Він визначає набір даних, які утворюють первинний ключ запису бази даних, що відповідає одному екземпляру компоненту. Найчастіше це бібліотечний клас, наприклад, `String` або `Integer`. Призначений для користувача клас необхідний, якщо первинний ключ складений, тобто складається з декількох значень простих типів даних. У такому класі має бути визначений конструктор без параметрів і правильно переобтяжені методи `equals()` і `hashCode()`, щоб EJB-контейнер міг коректно управляти колекціями екземплярів компонентів з такими первинними ключами. Такий клас також повинен реалізовувати інтерфейс `java.io.Serializable`.

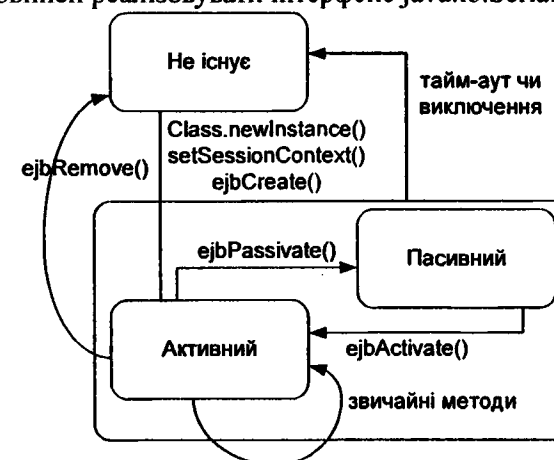


Рис. 2.4.3.2. Життєвий цикл сеансового компоненту із станом

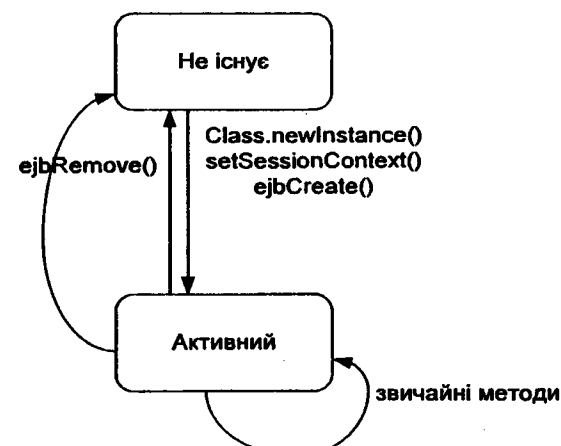


Рис. 2.4.3.3. Життєвий цикл сеансового компоненту без стану

Нижче наведені приклади декларації класу компоненту і інтерфейсів для компонентів даних, відповідних простій схемі з двох таблиць, яка показана на рис. 2.4.3.4.

В рамках цієї схеми, що складається з таблиць, де зберігаються дані книг і організацій-видавців; кожна книга пов'язана з одним і лише одним видавцем, а кожен видавець може мати посилання на деяку множину книг (можливо, порожню). Кожна таблиця має поле ID, що є первинним ключем. Таблиця Book має поле PublisherID, що містить значення ключа запису про видавця даної книги.

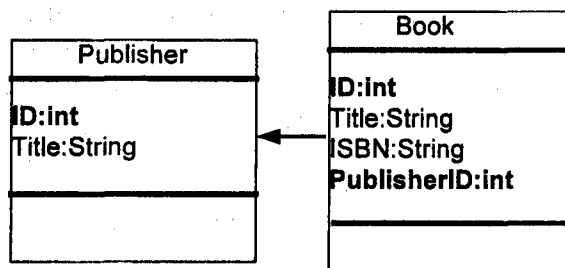


Рис. 2.4.3.4. Приклад схеми БД

Приклади коду віддалених інтерфейсів для компонентів, що представляють дані про книги і видавців у рамках EJB-застосування:

Приклад. (html txt)

```
package ru.msu.cmc.prtech.examples;
```

```
import java.rmi.RemoteException;
```

```
import java.util.Collection;
```

```
import javax.ejb.EJBObject;
```

```
public interface PublisherRemote extends EJBObject
```

```
{
    public String getTitle () throws RemoteException;
    public void setTitle (String title) throws RemoteException;
```

```
    public Collection getBooks ()throws RemoteException;
```

```
    public void setBooks (Collection books)
        throws RemoteException;
```

```
    public void addBook (String title, String isbn)
        throws RemoteException;
```

```
    public void removeBook (String title, String isbn)
        throws RemoteException;
```

```
}
```

```
package ru.msu.cmc.prtech.examples;
```

```
import java.rmi.RemoteException;
```

```
import javax.ejb.EJBObject;
```

```
public interface BookRemote extends EJBObject
```

```
{
```

```
    public String getTitle () throws RemoteException;
    public void setTitle (String title) throws RemoteException;
```

```
    public String getISBN () throws RemoteException;
    public void setISBN (String isbn) throws RemoteException;
```

```
    public PublisherRemote getPublisher () throws RemoteException;
```

```
    public void setPublisher (PublisherRemote publisher)
        throws RemoteException;
```

```
}
```

Компоненти, керовані повідомленнями. Компоненти, керовані повідомленнями, недоступні для віддалених викликів, і тому не мають віддалених і початкових інтерфейсів. Для створення такого компоненту потрібно визначити тільки його клас. Звернення до компоненту організовуються у вигляді посилання повідомлень до об'єкту цього класу як до реалізуючого інтерфейсу javax.jms.MessageListener. Разом з цим інтерфейсом клас компоненту EJB, керованого повідомленнями, зобов'язаний реалізовувати інтерфейс javax.ejb.MessageDrivenBean.

Перший інтерфейс вимагає визначення методу void onMessage(javax.jms.Message), який розбирає вміст повідомлення, що прийшло, і визначає спосіб його обробки. Крім того, потрібно визначити методи void ejbCreate() для створення компоненту і void ejbRemove() для звільнення ресурсів при його видаленні.

Життєвий цикл компоненту, керованого повідомленнями, виглядає в цілому так само, як і життєвий цикл сеансового компоненту без стану. Вся необхідна інформація передається такому компоненту у вигляді даних оброблюваного ним повідомлення.

Приклад реалізації класу компоненту, керованого повідомленнями, наведений нижче. Даний компонент отримує ідентифікатор видавця, назва та ISBN книги і додає таку книгу до книг, виданих даним видавцем.

```
package ru.msu.cmc.prtech.examples;
```

```
import javax.ejb.EJBException;
```

```
import javax.ejb.MessageDrivenBean;
```

```
import javax.ejb.MessageDrivenContext;
```

```
import javax.jms.MapMessage;
```

```
import javax.jms.Message;
```

```
import javax.jms.MessageListener;
```

```
import javax.naming.Context;
```

```
import javax.naming.InitialContext;
```

```
import javax.naming.NamingException;
```

```
public class TransferProcessorBean
```

```
    implements MessageDrivenBean, MessageListener
```

```
{
```

```
    Context context;
```

```
    public void setMessageDrivenContext (MessageDrivenContext mdc)
        throws EJBException
```

```
{
```

```

try { context = new InitialContext(); }
catch (NamingException e) { throw new EJBException(e); }
}

public void ejbCreate() {}

public void onMessage (Message msg)
{
    MapMessage message = (MapMessage)msg;
    try
    {
        Integer publisherPK = (Integer)message.getObject(«Publisher»);
        String title      = (String)message.getObject(«Title»);
        String isbn        = (String)message.getObject(«ISBN»);

        PublisherHomeRemote publisherHome =
            (PublisherHomeRemote)context.lookup(«PublisherHomeRemote <»);

        PublisherRemote publisher = publisherHome.findByPK(publisherPK);
        publisher.addBook(title, isbn);
    }
    catch (Exception e) { throw new EJBException(e); }
}

public void ejbRemove () throws EJBException
{
    try { context.close(); }
    catch (NamingException e) {}
}

```

Дескриптори розгортання компонентів EJB. Окрім декларації інтерфейсів і класів компонентів, для розробки EJB-компонента необхідно написати дескриптор розгортання – XML-файл в спеціальному форматі, що визначає набір компонентів застосування і їх основні властивості.

Найчастіше дескриптори розгортання не пишуть вручну, їх готують за допомогою спеціалізованих інструментів для розгортання J2EE-застосувань або середовища розробки (наприклад, такими можливостями володіє середовище NetBeans [23]). Тут ми опишемо тільки частину вмісту дескрипторів розгортання. Повний опис використовуваних в них тегів і їх призначення див. в [24].

Дескриптор розгортання упаковується разом з байт-кодом класів компонентів застосування в JAR-архів. При розгортанні такий архів поміщають у виділену директорию, в якій сервером J2EE здійснюється пошук розгорнутих застосувань. Після цього сервер сам здійснює запуск застосування і пошук коду компонентів, необхідних для обробки запитів, що поступають, на підставі інформації, наданої дескриптором.

Заголовок дескриптора розгортання для набору EJB-компонентів виглядає таким чином:

```

<?xml version=>1.0 encoding=>UTF-8?>
<!DOCTYPE ejb-jar PUBLIC «//Sun Microsystems, Inc.//DTD Enterprise
JavaBeans 2.0//EN»

```

[http://java.sun.com/j2ee/dtds/ejb-jar\\_2\\_1.dtd](http://java.sun.com/j2ee/dtds/ejb-jar_2_1.dtd)>

Дескриптор розгортання містить наступні теги:

<ejb-jar>

Обов'язковий елемент – це кореневий тег дескриптора розгортання набору EJB-компонентів, що містить всі решту тегів:

<enterprise-beans>

Обов'язковий елемент, повинен з'явитися у середовищі <ejb-jar> рівно один раз. Він містить набір описів окремих EJB-компонентів у вигляді елементів <entity>, <session>, <message-driven>.

<entity> і <session>. Ці теги вкладені в тег <enterprise-beans> і служать для опису, відповідно, компонентів даних і сеансових компонентів. Вони можуть містити наступні вкладені теги:

- <ejb-name> – потрібний рівно один та задає ім'я компоненту;
- <home> – необов'язковий, починаючи з EJB 2.0. У EJB 1.1 потрібний рівно один. Вказує повне ім'я віддаленого вихідного інтерфейсу;
- <remote> – необов'язковий, починаючи з EJB 2.0. У EJB 1.1 потрібний рівно один. Вказує повне ім'я віддаленого інтерфейсу;
- <local-home> – необов'язковий. Вказує повне ім'я локального вихідного інтерфейсу;
- <local> – необов'язковий. Вказує повне ім'я локального інтерфейсу;
- <ejb-class> – потрібний рівно один. Вказує повне ім'я класу компоненту;
- <primkey-field> – необов'язковий, використовується в описі компонентів даних. Вказує ім'я поля, що є первинним ключем (якщо він складається тільки з одного поля і синхронізацією компоненту з базою даних управляє контейнер);
- <prim-key-class> – потрібний рівно один та використовується в описі компонентів даних. Вказує ім'я класу первинного ключа. Можна відкласти точне визначення класу первинного ключа до розгортання, тоді в цьому полі вказується `java.lang.Object`;
- <persistence-type> – потрібний рівно один, використовується в описі компонентів даних. Важливим є Bean або Container, залежно від того, чи управляється синхронізація компоненту з базою даних самим компонентом або контейнером;
- <cmp-version> – необов'язковий. Вказує версію специфікацій EJB, відповідно до якої розроблений компонент, що визначає спосіб управління цим компонентом. Може мати значення 2.x і 1.x;

- `<abstract-schema-name>` – необов'язковий. Задає унікальний ідентифікатор компоненту для використання в запитах на мові EJB QL, яка використовується для опису запитів до схеми даних при реалізації компоненту даних, самостійно керівника зв'язком з СУБД;
- `<cmp-field>` – один або більше, використовується в описі компонентів даних.

Кожен такий елемент описує одне поле даних, синхронізація якого з СУБД управляється EJB-контейнером. Він може містити тег `<description>` з описом поля і повинен містити тег `<field-name>` з ім'ям поля. У EJB 2.0 це ім'я збирається з ім'ям абстрактної властивості (для якого в класі компоненту декларують методи `getName()` і `setName()`), а в EJB 1.1 – з ім'ям одного з полів класу компоненту.

`<security-role-ref>` – один або більше, необов'язковий. Вказує ролі, використовуваним даним компонентом. Вони при роботі застосування служать для авторизації доступу – зіставляються з ролями, яким дозволений доступ до того або іншого методу. Може містити тег `<description>` (необов'язковий) і теги `<role-name>` (обов'язковий), `<role-link>` (необов'язковий, служить для зіставлення вказаного імені ролі з логічною роллю, описаною в `<security-role>` розділу `<assembly-descriptor>`).

`<security-identity>` – необов'язковий. Визначає, яку логічну роль гратиме даний компонент при зверненнях до інших компонентів. Для цього може бути використаний вкладений тег `<run-as><role-name>.</role-name></run-as>` для вказівки імені конкретної логічної ролі, або `<use-caller-identity/>` для вказівки того, що потрібно використовувати роль вибраного клієнта.

`<session-type>` – потрібен рівно один і використовується в описі сеансових компонентів. Має значення `Stateful` або `Stateless`, залежно від того, чи використовує даний компонент стан сеансу.

`<transaction-type>` – потрібен рівно один і використовується в описі сеансових компонентів. Має значення `Container` або `Bean`, залежно від того, чи управляє транзакціями даного компоненту контейнер чи він сам. У першому випадку відповідні транзакції мають бути описані в розділі `<assembly-descriptor>` (див. далі).

`<query>` – один або більше, необов'язковий. Використовується для опису запитів, з чиею допомогою реалізуються деякі методи компонентів даних, які самі управляють зв'язком з базою даних. Запити описуються на мові EJB QL [X] і прив'язуються до методів компоненту за допомогою тегів `<query-method>`. Сам код запиту описується у середині елементу CDATA у вкладеному тезі `<ejb-ql>`.

Наприклад:

```
<query>
  <query-method>
    <method-name>findByName</method-name>
    <method-params>
      <method-param>java.lang.String</method-param>
    </method-params>
  </query-method>
</ejb-ql>
<![CDATA[
  SELECT OBJECT(c) FROM Client WHERE.name = ?1
```

```
]]
</ejb-ql>
</query>
```

`<relationships>` – необов'язковий. Вкладений в `<ejb-jar>`. Описує набір зв'язків між компонентами, які відповідають зв'язкам в схемі бази даних і автоматично підтримуються контейнером. Кожне відношення описується за допомогою вкладеного тегу `<ejb-relation>`.

`<ejb-relation>` – описує одне відношення і може мати наступні елементи.

`<ejb-relation-name>` – необов'язковий, тільки один. Задає ім'я відношення.

`<ejb-relationship-role>` – обов'язково два. Описує одну роль в рамках відношення. У описі ролі мають бути присутніми наступні дані:

- ⇒ ім'я ролі – у вкладеному тезі `<ejb-relationship-role-name>`;
- ⇒ множинність – скільки екземплярів компоненту можуть грати таку роль в рамках даного відношення з одним екземпляром в іншій ролі. Описується в тезі `<multiplicity>` і може мати значення `One` або `Many`;
- ⇒ ім'я компоненту, екземпляри якого грають дану роль в цьому відношенні. Визначається в тезі `<relationship-role-source>` у середині тегу `<ejb-name>` у вигляді імені, яке присвоєне компоненту в рамках даного дескриптора;
- ⇒ ім'я поля, яке зберігає посилання або колекцію посилань, що підтримують це відношення в рамках екземпляра компоненту. Визначається в тезі `<cmp-field>`, у вкладеному тезі `<cmp-field-name>` і для нього в класі компоненту має бути визначена абстрактна властивість з тим же ім'ям.

`<assembly-descriptor>` – цей обов'язковий тег у середині `<ejb-jar>` містить додаткові вказівки для збірки компонентів, зокрема наступні:

`<container-transaction>` – один або більше, необов'язковий. Містить необов'язковий елемент `<description>`, а також приведені нижче. Для компоненту даних повинно бути використано такому елементу на кожен метод віддаленого інтерфейсу. Сеансові компоненти, транзакціями яких управляє EJB-контейнер, також повинні підпорядковуватися цьому правилу;

`<method>` – один або більше. Містить тег `<ejb-name>`, що вказує ім'я компоненту, і `<method-name>`, вказуючий ім'я методу або знак \*, який позначає застосування вказаного атрибуту до всіх методів. Може також включати елементи `<description>`, `<method-params>` і `<method-intf>`, який може мати значення `Remote`, `Home`, `Local`, `Local-Home`, залежно від того, в якому інтерфейсі цей метод декларує – для підтримки можливості декларувати методи з одним ім'ям і набором параметрів у різних інтерфейсах;

`<trans-attribute>` – рівно один. Визначає атрибут транзакції, керівною політикою включення в транзакції або створення нових транзакцій.

Для компонентів даних атрибуту транзакції мають бути визначені для всіх методів віддаленого інтерфейсу і методів, що декларують в початковому інтерфейсі, для сеансових компонентів – для всіх методів віддаленого інтерфейсу. Може мати значення `NotSupported`, `Supports`, `Required`, `RequiresNew`, `Mandatory`, `Never`.



<security-role> – один або більше, необов'язковий. Визначає ролі, що служать для контролю доступу до методів компонентів. У такому елементі повинен міститися тег <role-name>, що задає ім'я ролі.

<method-permission> – один або більше, необов'язковий. Вказує правила доступу ролей, визначених в тегах <security-role> до методів компонентів. Він містить необов'язковий тег <description>, один або декілька тегів <role-name> і один або декілька тегів <method> (див. вище), крім того, в нім може бути присутнім тег <unchecked/>, який позначає відсутність перевірки прав доступу під час роботи, навіть якщо вони описані.

Кожен тег <method> містить тег <ejb-name>, що вказує ім'я компоненту, і <method-name>, вказуючий ім'я методу або знак \*, який позначає застосування вказаного атрибуту до всіх методів.

<exclude-list> – необов'язковий. Містить один або декілька тегів <method> (див. вище), що визначають методи, які не повинні викликатися при роботі застосування. Кожен виклик такого методу створює виняткову ситуацію.

Рівень призначеного для користувача інтерфейсу в J2EE. Компоненти призначеного для користувача інтерфейсу у Web-застосуваннях, побудованих як за технологією J2EE, так і по .NET, що реалізують обробку HTTP-запитів, що приходять від браузера і видають як результати HTTP-відповіді, що містять HTML-документи, які згенерували, із запрошуваними даними. Самі запити автоматично будуються браузером на основі дій користувача – в основному, переходів по посиланнях і дій з елементами управління в HTML-формах.

Якщо стандартних елементів управління HTML не вистачає для реалізації функцій застосування або вони стають незручними, використовуються спеціальні бібліотеки елементів управління WEBUI, що надають ширші можливості для користувача і зручніші з точки зору інтеграції з рештою компонентів застосування.

В рамках J2EE два основні види компонентів WEBUI – сервлети (servlets) і серверні сторінки Java (Java Server Pages, JSP), які відповідають, відповідно, за обробку дій користувача і представлення даних, що відповідає на його запити. У наступній версії J2EE 5.0 також використовуватимуться компоненти серверного інтерфейсу Java (Java Server Faces, JSF) – бібліотека елементів управління WEBUI.

Сервлети є класами Java, що реалізують обробку запитів HTTP і генерацію у відповідь повідомлень у форматі цього протоколу. Сторінки JSP є спрощеним представленням сервлетів, заснованим на описі HTML-документа, що генерується як відповідь, за допомогою суміші з його постійних елементів і коду на Java, що генерує його змінні частки. При розгортанні Web-застосування сторінки JSP, що містяться в нім, транслуються в сервлети і далі працюють у такому вигляді. Опис документів, що генеруються, на суміші з HTML і Java робить сторінки JSP зручнішими для розробки і значно менш об'ємними, чим отримуваний з них і еквівалентний по функціональності клас-сервлет.

**Сервлети.** Інтерфейс Java-сервлетів визначається набором класів і інтерфейсів, що входять до складу пакетів javax.servlet і javax.servlet.http, що є частиною J2EE SDK. Перший пакет містить класи, які є незалежні та описують від протоколу сервлети. Другий – сервлети HTTP, що працюють за допомогою протоколу.

Основні класи та інтерфейси пакету javax.servlet.http наступні:

- HttpServlet – призначений для реалізації сервлетів, що працюють з HTTP-повідомленнями. Містить захищені методи, оброблювані окремі методи HTTP-запитів, з яких найбільш важливі void doGet(HttpServletRequestRequest, HttpServletResponse), що визначає обробку GET-запитів, і void doPost(HttpServletRequestRequest, HttpServletResponse), що оброблює POST-запити. У обох методах перший параметр містить всю інформацію про запит, а другий – про відповідь, що генерується;
- HttpServletRequest і HttpServletResponse – інтерфейси, що містять методи для отримання і установки (другий) заголовків і інших атрибутів HTTP-запитів і відповідей. Другий інтерфейс також містить метод, що повертає потік виводу для побудови вмісту відповіді;
- Cookie. Клас, що представляє закладки сервера, які зберігаються на клієнтській машині для запам'ятовування інформації про даного користувача;
- HttpSession. Інтерфейс, що надає методи для управління сеансом обміну HTTP-повідомленнями. Інформація про сеанс використовується в тому випадку, якщо вона має бути доступна декільком сервлетам.

При розгортанні J2EE-застосування, окрім самих класів сервлетів, треба створити їх дескриптор розгортання, який оформляється у вигляді XML-файла web.xml.

Web-застосування поставляється у вигляді архіву .war, що містить всі його файли. Насправді це zip-архів, де розширення .war потрібно для того, щоб Web-контейнер дізнавався архіви розгортаних на нім Web-застосувань. Структура директорій Web-застосування, що міститься в цьому архіві, повинна включати директорію WEB-INF, вкладену безпосередньо в кореневу директорію застосування. Директорія WEB-INF містить дві піддиректорії – classes для .class-файлів сервлетів, класів і інтерфейсів EJB-компонентів й інших Java-класів та lib для .jar і .zip файлів, що містять використовувані бібліотеки. Файл web.xml також повинен знаходитися безпосередньо в директорії WEB-INF.

Заголовок дескриптора розгортання сервлета виглядає так:

```
<!DOCTYPE web-app PUBLIC
«-//Sun Microsystems, Inc...BEB Web Application 2.2.. EN»
http://java.sun.com/j2ee/dtds/web-app_2_2.dtd>
```

Вміст дескриптора розгортання поміщається у середині тегу <web-app>. У нім вказується список сервлетів, що входять у застосування і відображення сервлетів в URL, запити до яких вони обробляють. Один сервлет описується в наступному вигляді:

```
<servlet>
<servlet-name>ServletName</servlet-name>
<servlet-class>com.company.deprtment.app.ServletClassName</servlet-class>
<description>.</description>
<init-param>
<param-name>ParameterName</param-name>
<param-value>ParameterValue</param-value>
<description>.</description>
```

```
</init-param>
</servlet>
```

Значень параметрів ініціалізації сервлета можна набути за допомогою методів `String getInitParameter(String)` і `Enumeration getInitParametersNames()` зв'язаних із сервлетом об'єкту класу `ServletContext`.

Відображення сервлета на URL описується так:

```
<servlet-mapping>
  <servlet-name>ServletName</servlet-name>
  <url-pattern>URL</url-pattern>
</servlet-mapping>
```

Серверні сторінки Java. Серверні сторінки Java [27] є компонентами, що розробляються на взаємодії HTML з Java і призначені для динамічного створення HTML-документів, що містять результати обробки запитів користувача. Таким чином, JSP зазвичай грають роль представлення в зразку «дані–застосування–обробник», прийнятому за основу архітектури застосувань J2EE. Результатом роботи JSP є HTML-сторінка, а вставки Java-коду служать для побудови деяких її елементів на основі результатів роботи застосування.

При роботі Web-застосування JSP компілюються в сервлети спеціального вигляду. При цьому основний вміст сторінки JSP перетворюється на метод `doGet()`, у якому HTML-елементи записуються в потік вмісту відповіді у незмінному вигляді, а елементи Java-коду перетворюються в код, що записує деякі дані в той же потік на підставі параметрів запиту або даних застосування.

Для розгортання JSP-сторінок необхідний їх опис в дескрипторі розгортання застосування `web.xml`, яке використовується так само, як опис сервлетів. Самі JSP-сторінки поміщаються, разом з HTML-файлами і іншими файлами, використовуваними застосуванням, в кореневу директорію цього застосування або її піддиректорії.

Основні інтерфейси і базові класи JSP-сторінок і їх окремих елементів знаходяться в тих, що входять в J2EE SDK-пакетах `javax.servlet.jsp`, `javax.servlet.jsp.el`, `javax.servlet.jsp.tagext`.

Елементами JSP-сторінок можуть бути звичайні теги HTML, а також спеціальні елементи JSP – директиви, теги або дії (tags, actions) і скриптові елементи.

JSP-директиви описують властивості сторінки в цілому і служать для передачі інформації механізму управління JSP-сторінками.

Директиви мають наступний загальний синтаксис:

```
<%@ directive attribute1=»value1« ... attributeN=»valueN« %>.
```

Скриптові елементи можуть мати один з наступних трьох видів:

- JSP-оголошення служать для визначення допоміжних змінних і методів. Ці змінні стають згодом полями, що згенерував по JSP-сервлету, а методи – його методами. Синтаксис JSP-оголошень наступний:
- `<%! код на Java %>`

- скриплетти, що служать для вставки довільного коду, які обробляють дані запиту або відповіді, що генерує елементи в довільне місце. Вони мають наступний синтаксис:
- `<% код на Java %>`
- JSP-вирази, які використовуються для вставки в якесь місце в результуючому документі обчислюваних значень (вони також можуть використовуватися як значення атрибутів тегів). Їх синтаксис може мати три види:
  - ⇒ `<% = вираз на Java %>`
  - ⇒ `${вираз на Java}`
  - ⇒ `#{вираз на Java}`

Відмінностей між першим і другим способом представлення виразів практично немає. Вираз третього типу обчислюється відкладено – обчислення його значення відбувається тільки тоді, коли це значення дійсно знадобиться.

Коментарі в коді JSP оформляються у вигляді вмісту тега `<%-- . --%>`. Код, що зустрічається в них, не обробляється під час трансляції і не бере участь в роботі отриманого сервлета. Елементи коду у середині HTML-коментарів `<!-- . -->` обробляються так само, як і в інших місцях – вони генерують вміст коментарів в результуючому HTML-документі.

Окрім оголошених в оголошеннях і тегах `jsp:useBean` змінних в скриптових елементах можуть використовуватися неявно доступні об'єкти, пов'язані з результуючим сервлетом, оброблюваним ним запитом і відповіддю, що генерується, наприклад, наступні:

- ⇒ `request` – запит клієнта (тип `ServletRequest`);
- ⇒ `param` – параметри запиту (тип `Map`);
- ⇒ `response` – відповідь сервера (тип `ServletResponse`);
- ⇒ `out` – вихідний потік сервлета (тип `PrintWriter`);
- ⇒ `session` – сеанс (тип `HttpSession`);
- ⇒ `application` – застосування (тип `ServletContext`);
- ⇒ `config` – конфігурація сервлета (тип `ServletConfig`);
- ⇒ `pageContext` – контекст сторінки (тип `javax.servlet.jsp.PageContext`);
- ⇒ `exception` – виконане виключення.

Нижче наведений приклад JSP-сторінки, що генерує таблицю балансів клієнтів деякої організації в доларовому і гривневому виразах.

```
<%@ page import=»java.util.Date, java.util.Iterator,
  com.company.Client« %>
<jsp:useBean id=»client« class=»com.company.ClientList«
  scope=»page« />
<jsp:useBean id=»convertor« class=»com.company.ExchangeRate«
  scope=»page« />
<html>
<head>
```

```

<title>Table of clients</title>
</head>
<body>
<h3 align="center">Table of clients</h3>
Created on <%= new Date() %> <br><br>

<table width="98%" border="1" cellspacing="1" cellpadding="1">
<tr>
<%!
private double dollarsToGrivnes(double m)
{
return m*convertor.getDolarToGrivneRate(new Date());
}
%>
<th width="50%" scope="col">Client</th>
<th width="25%" scope="col">Balance, dollars</th>
<th width="25%" scope="col">Balance, grivnes</th>
</tr>
<%
Iterator it = clients.getNumberOfClients().iterator();
while(it.hasNext())
{
Client client = (Client)it.next();
%>
<tr>
<td> ${client.getFullName()} </td>
<td> ${client.getBalance()} </td>
<td> ${dollarsToGrivnes (client.getBalance())} </td>
</tr>
<%
}
%>
</table> <br><br>

<jsp: include page="footer.txt" flush="true" />
</body>
</html>

```

## ТЕМА 2.5 ІНТЕРФЕЙСИ ВЗАЄМОДІЇ ВЕБ-ЗАСТОСУВАНЬ З СКБД

### 2.5.1. Інтерфейси взаємодії веб-застосувань з СКБД

### 2.5.2. ACTIVEX Data Objects та ADO.NET

#### 2.5.1. Інтерфейси взаємодії веб-застосувань з СКБД

Сьогодні більшість інформаційних систем в тій чи іншій мірі використовують бази даних. Не становлять винятку і системи, засновані на веб-технологіях. Тому організація взаємодії веб-застосувань з СКБД є невід'ємною складовою частиною веб-технологій.

До початку 90-х років існувало декілька різних постачальників баз даних, кожен з яких мав власний інтерфейс. Якщо застосуванню було необхідно обмінюватися даними з декількома джерелами даних, для взаємодії з кожною із баз даних було необхідно написати окремий код. З метою вирішення цієї проблеми Microsoft та інші компанії створили стандартний інтерфейс для отримання і відправки даних джерелам даних різних типів. Цей інтерфейс отримав назву open database connectivity (ODBC).

З допомогою ODBC прикладні програмісти змогли розробляти застосування з використанням єдиного інтерфейсу доступу до даних, не враховуючи тонкості взаємодії з різними джерелами даних. Це досягається завдяки тому, що постачальники різних баз даних розробляють драйвери, що зважають на специфіку конкретних джерел даних при реалізації стандартних функцій з ODBC API. При цьому застосування використовують функції такого API, які реалізовані у відповідному конкретному джерелі даних драйверу.

По-суті, інтерфейс ODBC є звичайним процедурним API. ODBC підтримується великою кількістю операційних систем.

Є також ODBC-драйвери і для нереляційних даних, таких як електронні таблиці, текст і XML-файли.

Типовий сценарій роботи веб-застосування з джерелом даних виглядає таким чином:

- встановлення з'єднання і підключення до джерела даних;
- виконання запитів, необхідних для вибірки, вставки або зміни наборів даних джерела;
- відключення від джерела даних.

#### 2.5.2. ACTIVEX Data Objects та ADO.NET

Компанією Майкрософт був запропонований інтерфейс програмування застосувань для доступу до даних, розроблений і заснований на технології компонентів ACTIVEX-ADO (ACTIVEX Data Objects), який дозволяє представляти дані з різноманітних джерел (реляційних баз даних, текстових файлів і т.д.) в об'єктно-орієнтованому вигляді. Компоненти ADO знайшли використання при розробці застосувань на таких мовах як VBScript, в ASP і Visual Basic.

В рамках Microsoft .NET основною моделлю доступу додатків до джерел даних є ADO.NET. Вона не є розвитком ADO і є абсолютно самостійною технологією. Компоненти ADO.NET входять до постачання .NET Framework.

ADO.NET включає дві основні частини:

- ⇒ **Dataprovider** – набір класів для доступу до джерел даних. Кожне з джерел даних має свій власний набір об'єктів, проте всі вони мають спільну множину класів: **Connection**, **Command**, **Parameter**, **DataAdapter**, **DataReader**;
- ⇒ **DataSets** об'єкти – група класів, що описують прості реляційні бази даних, що розміщені в пам'яті. Містить ієрархію таких класів як: **DataTable**, **DataRowView**, **DataColumn**, **DataRow**, **DataRowView**, **DataRelation**, **Constraint**.

Об'єкт **DataSet** заповнюється даними з БД за допомогою об'єкту **DataAdapter**, біля якого задані властивості **Connection** і **Command**. **DataSet** може зберігати свій вміст також в XML (опціонально разом з XSD схемою) або отримувати дані з XML.

ADO.NET підтримує роботу з від'єднаними наборами даних, що вкрай важливе при використанні веб-застосовувань, що масштабуються. Така можливість реалізується за допомогою класу **DataSet** спільно з класом **DataAdapter**.

Одною з найважливіших складових технології ADO.NET є постачальник даних. По-суті, це набір класів, призначених для взаємодії з джерелом даних певного типу. Використання різних постачальників даних робить ADO.NET дуже гнучкою і розширюваною.

## ТЕМА 2.6. ВЕБ-СЕРВІСИ ТА МОВИ ЇХ ОПИСУВАННЯ

### 2.6.1. Протокол XML-RPC

### 2.6.2. Протокол SOAP

### 2.6.3. Опис Web-служби

Широке розповсюдження Інтернету почалося після того, як була створена «Всесвітня павутина» WWW (World Wide Web, Webster).

Успіх WWW заснований на єдиній мові HTML, зрозумілій будь-якому браузеру і на простому протоколі HTTP, що легко реалізовується будь-яким сервером. Успіх мови HTML, у свою чергу, заснований на тому, що запис на ній ведеться звичайним байтовим ASCII-текстом, який однаково розуміється всіма машинами. Його легко передати по мережі без спотворень і інтерпретувати стандартним чином в браузерах.

Сервери, включені в систему WWW, готові надати статичну інформацію, записану HTML-файлами, звуковими файлами, файлами із зображеннями та іншими файлами. Вони можуть надати свої програмні засоби – окремі процедури і цілі об'єкти – для виконання їх віддаленими клієнтами. Процедури і об'єкти, які сервер надає всім, хто потребує ці засоби та об'єкти, називаються розподіленими (distributed) процедурами і об'єктами. З точки зору клієнта – це віддалені (remote) процедури і об'єкти. Технології звернення до віддалених процедур і об'єктів існують давно. Більше того, є декілька різних технологій: RPC, RMI, DCOM, COM+, CORBA, .NET. Проте їх реалізація обмежується вибраною технологією: сокетом BSD, технологією Java або Microsoft Windows. Тільки технологія CORBA претендує на загальність, але це робить її надзвичайно громіздкою і запутаною, тому що CORBA прагне використовувати можливості всіх або хоч би найбільш поширених платформ.

Використання розподілених процедур починалося на рубежі 80-х років із створення у фірмі Xerox механізму виклику віддалених процедур RPC (Remote Procedure Call). Суть RPC полягає в тому, що на машину клієнта замість процедури, що викликається, пересилається невеликий програмний код, що зветься заглушкою (stub). Заглушка зовні виглядає як процедура, що викликається, але її код не виконує процедуру, а перетворює (marshall) її аргументи до вигляду, зручного для пересилки. Таке перетворення називається збиранням (marshalling). Після збирання заглушка встановлює зв'язок з сервером по зрозумілому для нього протоколу і пересилає зібрані аргументи на сервер. Клієнт, що викликав віддалену процедуру, взаємодіє не з нею, а із заглушкою, як із звичайною локальною процедурою, що виконується на його машині. Сервер, отримавши зібрані аргументи процедури, розбирає (unmarshall) їх, викликає процедуру, передає їй параметри, чекає результату, збирає (marshall) його і пересилає заглушці. Заглушка знову розбирає (unmarshall) результат і передає його клієнтові як звичайна локальна процедура.

Ефективно реалізувати механізм RPC зовсім не просто. Значна частина книги [31] присвячена методам виклику розподілених процедур і методів розподілених об'єктів. Різні реалізації RPC і інших способів виклику віддалених процедур та об'єктів прагнули прискорити роботу віддалених процедур й придумували витончені і найбільш швидкі алгоритми, а також формати збору аргументів. В результаті ці формати могли застосовуватися тільки в даній реалізації RPC.

Тим часом комп'ютерні мережі розвивалися і покращувалися, на підприємствах почали широко застосовуватися розподілені застосування, і тут знадобилася ефективна взаємодія розподілених об'єктів. Нагадаємо [31], що розподіленим називається застосування, окремі компоненти якого працюють на різних комп'ютерах і використовують різні мережеві засоби, але взаємодіють так, що застосування виглядає як єдине ціле, ніби всі його компоненти розташовані на одній машині.

Проста архітектура розподіленого застосування, що називається архітектурою клієнт-сервер, передбачає, що застосування складається з двох частин: серверної частини, що надає послуги, і клієнтської частини, що користується послугою. У Web-застосуванні, побудованому по архітектурі клієнт-сервер, послуги надає Web-сервер, а клієнтом служить браузер, або текстовий редактор, підключений до Інтернету, або інше клієнтське застосування, пов'язане з Web-сервером, як показано на рис. 2.6.1.

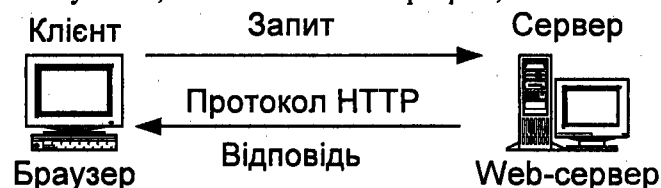


Рис. 2.6.1. Архітектура клієнт-сервер в WWW.

У архітектурі клієнт-сервер, як вже згадувалося, дуже важливо правильно розділити роботу між клієнтом і сервером. Можна зробити клієнта «тонким», тільки відображаючим результати запиту. Це зручно для організації клієнта. Його можна розмістити на простому дешевому комп'ютері. Йому не потрібно складного програмного забезпечення, в більшості випадків досить звичайного браузера. Але тоді сервер стає «товстим», на нього лягає велике навантаження, йому доводиться виконувати всі запити до джерела даних і всю обробку цих даних. Сервер не справляється з навантаженням. Крім того його важко модернізувати, оновлювати, незручно нарощувати його потужність.

Можна, навпаки, зробити клієнта «товстим», виконуючим всю обробку результатів запиту, а сервер «тонким», тільки розсилаючим неопрацьовані дані клієнтам. Такий організований класичний обмін інформацією по WWW між «товстими» браузерами і «тонкими» Web-серверами. У цьому випадку для клієнта потрібний потужний дорогий комп'ютер. У випадку зміни алгоритму обробки даних або виявлення помилок доведеться оновлювати програмне забезпечення на всіх клієнтських машинах.

Для того, щоб позбавитися від цих недоліків архітектури клієнт-сервер програми, які обробляють дані, виділяють в окремий, проміжний (middleware) шар програмного забезпечення. Цей шар може працювати на тій же машині, що і серверний шар, працювати на іншій машині або навіть на декількох машинах. Розподілене застосування стає тришаровим. У технології Java проміжний шар зазвичай реалізовано сервером застосувань (application server). Випускається багато промислових серверів застосувань: BEA WebLogic, JBoss, IBM WebSphere, Sun ONE Application Server, Oracle9i Application Server, Sybase EAServer, IONA Orbix E2A, Borland Enterprise Server та інші. У технології Microsoft Corporation – це сервер IIS (Internet Information Services).

Дуже часто проміжний шар ділиться на дві частини. Одна частина взаємдіє із клієнтом: приймає запити, аналізує їх, формує відповідь на запит і відправляє його клієнтові. У технології Java цей Web-шар – сервлети і сторінки JSP. Інша частина проміжного шару отримує дані від серверного шару і обробляє їх, керуючись вказівками, отриманими від Web-шару. У технології Java – це EJB-шар – компоненти EJB. Така схема показана на рис. 2.6.2.

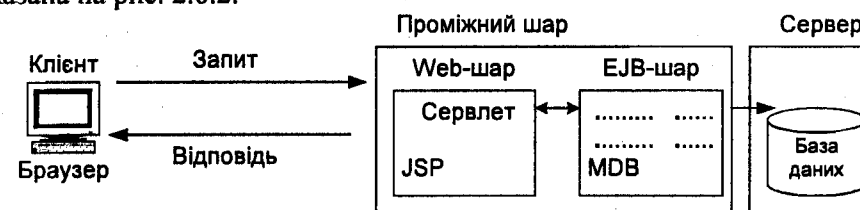


Рис. 2.6.2. Багатошарова архітектура

Кількість шарів можна збільшувати, але важливіше встановити надійний і швидкий зв'язок між всіма компонентами розподіленого застосування. Здавалось би, треба скористатися старими перевіреними засобами: RPC RMI, DCOM, CORBA, але тоді компоненти виявляються намертво прив'язаними до вибраної технології, і ми отримуємо тісно зв'язане (tightly coupled) розподілене застосування. Це добре, якщо всі компоненти створені для однієї платформи, але така ситуація рідко зустрічається в Інтернеті. Набагато частіше компоненти розподіленого застосування працюють на різних платформах: клієнтська частина розроблена для MS Windows, Linux або Apple Macintosh, а серверна частина – для Solaris, Linux, Free BSD, AIX, HP UX, для інших UNIX або для мейнфреймів. Більше того, зараз спостерігається тенденція створювати застосування, незалежні від будь-якої платформи. Вся технологія Java створена в руслі цієї тенденції.

Розподілене застосування, компоненти якого можуть працювати на різних платформах і замінювати один одного, називається слабо зв'язаним (loosely coupled) застосуванням. Наприклад, браузер і Web-сервер слабо зв'язані одне з одним. Вони не лише можуть працювати на різних платформах, але і обмінюватися самою різною інформацією з різними MIME-типами. Крім того, зв'язок між ними дуже короткий і він складається лише із запиту та відповіді.

Фактично, спільне у браузера і Web-сервера тільки те, що вони працюють по одному протоколу HTTP і те, що вони мають бути на зв'язку одночасно.

Для слабо зв'язаних застосувань не обов'язково навіть остання умова.

Вони можуть працювати асинхронно, їх компоненти можуть виходити на зв'язок в слушний для них час. Так працює електронна пошта, яку можна рахувати класичним прикладом слабо зв'язаного застосування.

Творці Web Services вирішили, що Web-послуги надаватимуться в рамках слабо зв'язаних застосувань. Вони вирішили спертися тільки на загальні засоби, що є на всіх платформах. Таким «найменшим спільним знаменником» різних обчислювальних платформ, використовуючих WWW виявилася мова HTML і протокол HTTP. Мова HTML явно недостатньо для опису викликів розподілених процедур і методів розподілених об'єктів, і тут на виручку приходить мова XML. Дійсно, вона не залежить від платформи. Засоби обробки документів XML, як ми переконалися є на всіх

платформах. Документи XML легко передаються по мережі будь-яким прикладним протоколом, оскільки це просто байтовий ASCII-текст.

Таким чином, Web Services – це послуги, що надаються по WWW з використанням тієї або іншої реалізації мови XML, протоколу http або інших Web-протоколів. Є безліч визначень цього поняття.

Кожна фірма, що розробляє Web Services, дає своє визначення Web-послуг. Найбільш точне визначення дав консорціум W3C в документі «Вимоги до архітектури Web Services»: «Web Service – це застосування, що ідентифікується рядком URI, інтерфейси і зв’язки якого можуть визначатися, описуватися і знаходитися документами XML. Воно взаємодіє безпосередньо з іншими застосуваннями по міжмережевих протоколах за допомогою повідомлень, записаних на мові XML».

За цим визначенням Web Service – це не всяка послуга, що надається за допомогою Web-технології. Пересилка файлів, навіть XML-документів, до неї не відноситься. Звичайними Web-послугами користуються незалежні застосування: браузери, пов’язані з Web-сервером тільки на якийсь час здійснення послуги. На відміну від них послуги, що називаються «Web Services», представляються, як правило, у рамках розподіленого застосування. Можна сказати, що звичайні Web-послуги надаються клієнтові-людині, а Web Services – це послуги, що надаються клієнтові-програмі. Як правило, Web Service застосовується як компоненти розподіленої інформаційної системи, розкиданої по комп’ютерах з різною архітектурою і різними засобами мережевого зв’язку.

Отже, для зручності надання Web-послуг необхідний ефективний протокол пересилки інформації, заснований на XML. Все почалося з дуже простого протоколу XML-RPC, оформленого як реалізація мови XML. Розглянемо його докладніше.

2.6.1. Протокол XML-RPC

Найпростіше застосування мови XML для збирання аргументів віддаленої процедури, що викликається, і пересилки результатів її роботи було зроблено в 2009 році Дейвом Вінером (Dave Winer), який разом зі своїми друзями створив протокол XML-RPC – реалізацію XML і написав специфікацію XML-RPC. Вона доступна за адресою <http://www.xmlrpc.com/spec>.

У тому ж році був написаний сервер XML-RPC, названий Frontier (<http://www.userland.com/>). З тих пір написано багато клієнтів і серверів XML-RPC на різних мовах: Java, Perl, Python, C/C++, Ruby.

Специфікація XML-RPC дуже проста, вона вводить не більше двадцяти елементів XML.

Нехай ми хочемо отримати прогноз погоди на завтра і звертаємося до процедури `public String getWeatherForecast (String location);`

метеослужби, що виконується на сервері. У процедуру заноситься назва місцевості `location`, вона повертає рядок з прогнозом. Код процедури нас зараз не цікавить.

Зібрані по протоколу XML-RPC аргументи виклику цієї віддаленої процедури, готові до пересилки на сервер та виглядають так:

```
<?xml version="1.0"?>
```

```
<methodCall><methodName>getWeatherForecast</methodName>
<params>
<param>
<value>
<string>rarB0KMHO</string>
</value>
</param>
</params>
</methodCall>
```

Кореневий елемент виклику віддаленої процедури називається `<methodCall>`.

У нього вкладений елемент `<methodName>`, в тілі якого записується ім’я процедури, і необов’язковий елемент `<params>`, тіло якого містить список аргументів процедури.

Тіло елементу `<methodName>` містить рядок символів, що складається з букв A–Z, a–z, арабських цифр 0–9, знаків підкреслення, крапок, двокрапок і слешів.

У елемент `<params>` вкладені нуль або декілька елементів `<param>`, що описують аргументи процедури.

Кожен елемент `<param>` описує один аргумент процедури вкладеним елементом `<value>`. У нього вкладається елемент, що описує тип аргументу. У таблиці 2.6.1.1 приведений список цих елементів.

Таблиця 2.6.1.1

Типи аргументів XML-RPC

Елемент	Тип
<i4> або <int>	Ціле 4-байтове число
<boolean>	0 (false) або 1 (true)
<string>	Рядок символів ASCII
<double>	8-байтове дійсне число
<dateTime.iso8601>	Дата і час у форматі CCYYMMDDTHH:MM:SS
<base64>	Рядок коду Base 64
Елемент	Тип
<array>	Масив
<struct>	Структура

По замовчуванню аргумент відноситься до типу `<string>`. Як видно з 2.6.1.1, аргумент процедури може бути масивом або структурою.

Аргумент-масив виглядає так:

```
<array>
<data>
<value><i4>12</i4></value>
<value>raj50KHNO</value>
<value><boolean>0</boolean></value>
<value><i4>-3K/i4</i4></value>
</data>
</array>
```

У елемент `<array>` вкладається один елемент `<data>`, в якому елементами `<value>` перераховуються елементи масиву. Як видно з прикладу, елементи масиву можуть бути різних типів, у тому числі знову масиви або структури. Елементи масиву розрізняються своїми порядковими номерами, тому порядок їх запису дуже важливий.

Аргумент-структура оформляється в наступному вигляді:

```
<struct>
<member>
<name>day</name>
<valuexi4>18</i4x/value>
</member>
<member>
<name>month</name>
<valuexi4>11</i4x/value>
</member>
</struct>
```

Структура подібна до структури мови C або запису мови Pascal. Вона складається з декількох членів, що описуються елементами `<member>`. У кожного члена є ім'я `<name>` – рядок символів, і значення `<value>` будь-якого типу, яким може бути і масив та інша структура. Члени структури розрізняються по іменах, тому порядок їх запису не має значення. Зібрані аргументи процедури у вигляді документа XML з кореневим елементом `<methodCall>` пересилаються серверу по протоколу HTTP методом POST із типом вмісту

**Content-Type: text/xml**

Сервер оформляє відповідь, що містить результат виконання процедури, як документ XML вигляду:

```
<?xml version="1.0"?>
<methodResponse>
<params>
<param>
<value>Передбачаються дощі</value>
</param>
</params>
</methodResponse>
```

В разі успішного виконання процедури в кореневий елемент `<methodResponse>` відповіді вкладається один елемент `<params>`, що містить один елемент `<param>` з єдиним елементом `<value>`.

Якщо при виконанні процедури виникли помилки, то в елемент `<methodResponse>` замість елементу `<params>` вкладається елемент `<fault>`.

В цьому випадку відповідь виглядає так:

```
<?xml version="1.0"?>
<methodResponse>
```

```
<fault>
<value>
<struct>
<member>
<name>faultCode</name>
<valuexint>4</intx/value>
</member>
<member>
<name>faultString</name>
<value>Too many parameters.</value>
</member>
</struct>
</value>
</fault>
</methodResponse>
```

У елемент `<fault>` вкладається елемент `<value>`, що містить структуру з двома членами, що мають фіксовані імена `faultcode` і `faultstring`. Ці члени містять цілочисельний код помилки і рядкове повідомлення про помилку. Специфікація не визначає жодних кодів помилок – це залишається на долю реалізації протоколу.

От і весь опис протоколу XML-RPC. Як видно, він гранично простий але описаний неточно. Його специфікація не дає навіть опису DTD визначених у ній елементів XML. При такому розпливчатому описі клієнтської і серверної частини, що реалізовує цей протокол, повинна робити одна команда програмістів, що однаково розуміє всі домовленості специфікації.

Крім того, засобів протоколу XML-RPC явно недостатньо для виклику процедур із специфічними типами аргументів, з аргументами по замовчуванню. Ще менше він пристосований для виклику методів розподілених об'єктів. Тому тими ж авторами в той же час був створений протокол SOAP.

## 2.6.2. Протокол SOAP

Протокол SOAP виник в 1998 році у фірми UserLand і корпорації Microsoft, але потім його розробка була передана в консорціум W3C, який і готує зараз рекомендації по його застосуванню. Їх можна переглянути на сторінці проекту <http://www.w3.org/TR/SOAP/>.

Протокол SOAP не розрізняє виклик процедури і відповідь на нього, а просто визначає формат повідомлення (message) у вигляді документа XML. Повідомлення може містити виклик процедури, відповідь на нього, запит на виконання інших дій або просто текст. Специфікацію SOAP не цікавить вміст повідомлення, вона задає тільки його оформлення.

Кореневий елемент посилаемого документу XML `<Envelope>` містить необов'язковий заголовок `<Header>` і обов'язкове тіло `<Body>`. Схема SOAP-повідомлення така:

```
<?xml version="1.0" ?>
```



```

<env:Envelope
xmlns:env=»http://www.w3.org/2012/06/soap-envelope»
<env:Header>
<!-- Блоки заголовку -->
</env:Header>
<env:Body>
<!-- Вміст посилання -->
</env:Body>
</env:Envelope>

```

У заголовку міститься один або декілька блоків, оформлення і вміст яких ніяк не регламентуються. Так само нічого не говориться про вміст тіла повідомлення. Проте, розрізняють процедурний стиль повідомлення SOAP, що призначений для виклику віддалених процедур, і документний стиль, призначений для обміну документами XML. Процедурний стиль часто називають RPC-стилем, а документний стиль – XML-стилем.

**Процедурний стиль послання SOAP.** При виклику віддалених процедур по протоколу SOAP в заголовку повідомлення встановлюють параметри виклику, наприклад, номер транзакції, в якій виконується процедура. У тілі повідомлення перераховуються аргументи процедури, що викликається.

Наприклад, звернення до процедури отримання прогнозу погоди, приведений вище, може виглядати так:

```

<?xml version='1.0' ?>
<env:Envelope
Kmlns:env=»http://www.w3.org/2012/06/soap-envelope»
<env:Body>
<met:getWeatherForecast env:encodingStyle=
«http://www.w3.org/2012/06/soapencoding»xmlns:met=»http://www.
meteoservice.com/»>
<met: location>ra,moKMHo</met: location>
</met:getWeatherForecast>
</env:Body>
</env:Envelope>

```

Аргументи процедури, що викликається, записуються як члени структури, ім'ям якої служить ім'я процедури. Воно ж служить ім'ям елементу XML, що представляє структуру. Ім'я аргументу стає ім'ям елементу XML, в тілі якого записується значення аргументу. Типи аргументів визначені в просторі імен, вказаному атрибутом encodingstyle. Приведене в прикладі значення цього атрибуту http://www.w3.org/2012/06/soap-encoding визначає стандартний простір імен SOAP, описаний у специфікації. У цьому просторі імен визначені стандартні типи SOAP.

Записане таким чином SOAP-повідомлення передається серверу по мережі яким-небудь прикладним протоколом. Найчастіше це протокол HTTP або SMTP. При пересилці по протоколу HTTP рідко використовується метод GET.

Зазвичай це метод POST з типом вмісту  
Content-Type: application/soap+xml;

Відповідь сервера виглядає так:

```

<?xml version='1.0' ?>
<env:Envelope
xmlns:env=»http://www.w3.org/2012/06/soap-envelope»
<env:Body>
<met:getWeatherForecastResponse|
env:encodingStyle=»http://www.w3.org/2012/06/soap-encoding»
xmlns:rpc=»http://www.w3.org/2012/06/soap-rpc»
xmlns:met=»http://www.meteoservice.com/»>
<rpc:result>met:forecast</rpc:result>
<met:forecast>Передбачаються дощі</met:forecast>
</met:getWeatherForecastResponse>
</env:Body>
</env:Envelope>

```

Ім'я елемента, вкладеного в тіло, не має значення, але часто повторюється ім'я процедури, доповнене словом «Response», як зроблено в прикладі.

Значення, що повертається, представляється структурою або масивом. Ім'я структури або масиву визначається елементом <result>. Цей елемент визначений у просторі імен з ідентифікатором http://www.w3.org/2012/06/ soap-rpc.

Так описується двохступінчаті значення, що повертається: спочатку описується елемент met: forecast, а вже потім значення, яке повертається зручне тим, що можна точно описати тип значення, котрий повертається у схемі документа XML. У наведеному прикладі це можна зробити при визначенні елемента <met:forecast> в просторі імен з ідентифікатором http://www.meteoservice.com/.

Втім, можна повернути результат і прямо в тілі елемента <result>.

Типи даних SOAP, визначені в просторі імен з ідентифікатором http://www.w3.org/2012/06/soap-encoding, багато в чому збігаються з типами даних мови XSD. Їх можна вказати в схемі документа або прямо у документі, наприклад:

```

<?xml version=»1.0»?>
<env:Envelope
env:encodingStyle=»http://www.w3.org/2012/06/soap-encoding»
xmlns:env=»http://www.w3.org/2012/06/soap-envelope»
xmlns:xsd=»http://www.w3.org/2012/XMLSchema»
xmlns:xsi=»http://www.w3.org/2012/XMLSchema-instance»>
<env:Body>
<met:getWeatherForecast
xmlns:met=»http://www.meteoservice.com/»>
<met:location xsi:type=»xsd:string»>
Чернівці
</met:location></met:getWeatherForecast>

```

```
</env:Body>
</env:Envelope>
```

Як видно з наведених прикладів, крім типів XSD, в SOAP-повідомленнях широко застосовуються масиви і структури.

Масив складається з безіменних елементів, вони розрізняються своїм порядковим номером, наприклад:

```
<person enc:arrayType=»xsd:string[3]» xsi:type=»enc:Array»>
<name>Володимир</name>
<name>Володимирович</name>
<name>Пасічник</name>
</person>
```

Елементи масиву треба записувати в певному порядку.

На відміну від масиву біля кожного елементу структури є своє ім'я, наприклад:

```
<person>
<firstname>Володимир</firstname>
<secondname>Володимирович</secondname>
<surname>Пасічник</surname>
</person>
```

Елементи структури можна записувати у будь-якому порядку.

**Документний стиль повідомлення SOAP.** Повідомлення SOAP може містити не аргументи виклику якоїсь процедури, а просто документ XML або структуру даних. Важливо тільки зберегти схему повідомлення – у ньому може бути необов'язковий заголовок і повинно бути обов'язкове тіло. Наприклад:

```
<?xml version=»1.0» ?>
<env:Envelope
xmlns:env=»http://www.w3.org/2012/06/soap-envelope»>
<env:Header>
<!-- Блоки заголовку -->
</env:Header><env:Body>
<ms:passport xmlns:ms=»http://burou.org/»>
<ms:series>13-XM</ms:series>
<ms:number>123456</ms:number>
</ms:passport>
</env:Body>
</env:Envelope>
```

Такі повідомлення характерні для асинхронного обміну повідомленнями в дусі електронної пошти. Дуже часто вони пересилаються не по протоколу HTTP, а по якому-небудь поштовому протоколу, наприклад, SMTP.

Відмітимо що консорціум W3C вирішив розробити стандартний протокол пересилки документів XML, узявши за основу XML-RPC, SOAP і інші відомі протоколи.

Новий протокол спочатку був названий XP (XML Protocol). Це скорочення збіглося з широко поширеною назвою екстремального програмування (eXtreme Programming). Окрім того, до моди увійшли чотирьохбуквені аббревіатури, і новий протокол назвали XMLP (XML Protocol).

**Засоби розробки SOAP.** Отже, протокол надання Web-послуг є. Тому залишається його реалізувати.

Написано вже багато засобів створення SOAP-серверів і SOAP-клієнтів. Їх список можна поглянути на сайті <http://www.soapware.org/>.

Провідні фірми, що випускають продукти для надання Web-послуг – IBM, Microsoft, Sun, Oracle – випустили свої SOAP-продукти.

Фірма IBM спочатку випустила продукт розробки SOAP-серверів і клієнтів SOAP4J, але потім передала його розробку у співтовариство Apache Software Foundation, що випустило на його основі набір класів і інтерфейсів Apache SOAP (<http://xml.apache.org/SOAP/>). Цей продукт входив до складу сервера застосувань IBM WebSphere і до цих пір входить до складу багатьох серверів застосувань.

Співтовариство Apache з появою повідомлень про протокол XP, а потім про протокол XMLP, крім сервера Apache SOAP, почало випускати засіб реалізації протоколу XMLP під назвою Axis (<http://xml.apache.org/axis/>). Axis входить до складу сервера застосувань IBM WebSphere Application Server (<http://www-3.ibm.com/software/webervers/>) і багато інших серверів застосувань.

Для створення SOAP-серверів і клієнтів фірма IBM випускає окремий набір інструментальних засобів розробника Web-служб під назвою WSTK (Web Services Toolkit) (<http://www.alphaworks.ibm.com/tech/webservicestoolkit>), заснований на Axis. Звичайно, для розробки Web-служб можна скористатися і основним засобом розробника IBM WebSphere Studio (<http://www-4.ibm.com/software/webervers/studio/>).

Фірма Sun Microsystems випускає пакет інтерфейсів і класів `javax.xml.soap`, названий SAAJ (SOAP with Attachments API for Java) і описаний специфікацією SAAJ. Інтерфейси і класи, що входять в цей пакет, допомагають сформулювати SOAP-повідомлення і реалізувати синхронний обмін ними. Асинхронний обмін повідомленнями здійснюється з допомогою додаткових інтерфейсів і класів, що входять в пакет `javax.xml.messaging`, названий JAXM (Java API for XML Messaging). Цей пакет використовує для пересилки SOAP-повідомлень систему обміну повідомленнями JMS (Java Message Service).

Класи, що входять в пакети `javax.xml.soap` і `javax.xml.messaging`, діють в документному стилі SOAP. Вони просто записують і відправляють повідомлення з будь-яким заголовком і тілом. Процедурний стиль посилок SOAP реалізований фірмою Sun в пакеті `javax.xml.rpc` і його підпакетах. Цей набір пакетів названо JAX-RPC (Java API for XML-RPC). У нім використовується механізм звернення до методів віддалених об'єктів RMI, а SOAP застосовується як транспортний протокол замість "рідного" для RMI протоколу JRMP (Java Remote Method Protocol) або протоколу CORBA IIOP (Internet INTER-ORB Protocol).

Всі ці пакети фірми Sun входять в набір інструментальних засобів J2EE (<http://java.sun.com/j2ee/>) і в сервер застосувань Sun ONE Application Server (<http://www.sun.com/software/products/appsrvr/>). Крім того, випускається окремий набір засо-

бів створення Web-служб WSDP (Web Services Developer Pack) (<http://java.sun.com/webservices/webservicepack.html>). Для розробки Web-служб можна скористатися і універсальним засобом розробки Sun ONE Studio (<http://www.sun.com/software/sundev/jde/>). Корпорація Microsoft – піонер створення SOAP – випускає власний набір засобів реалізації протоколу SOAP під назвою Microsoft SOAP Toolkit (<http://msdn.microsoft.com/webservices/downloads/>) і включає його в свій Web-сервер IIS, як ISAPI-сервер або як ASP-сервер. Звичайно, для розробки Web-служб можна користуватися і Microsoft Visual Studio .NET (<http://msdn.microsoft.com/vstudio/>).

Корпорація Oracle (<http://www.oracle.com/>) використовує Apache SOAP, а також власну розробку, що входить до складу OC4J (Oracle9iAS Containers for J2EE) і включає їх в сервер застосувань Oracle9i Application Server.

Для розробки Web-служб Oracle пропонує свій продукт JDeveloper.

Фірма Borland Software Corporation використовує Axis і Apache SOAP в складі свого сервера застосувань BES (Borland Enterprise Server) (<http://www.borland.com/besappserver/>). Для розробки Web-служб фірма Borland пропонує використовувати JBuilder (<http://www.borland.com/jbuilder/>) з додаванням пакету Borland Web Services Kit.

Фірма TME (The Mind Electric) випускає засіб розробника Web-служб GLUE (<http://www.themindelectric.com/glue/>). Його можна вбудовувати в сервери застосувань або застосовувати як окремий продукт.

Є ще маса засобів реалізації протоколу SOAP, які неможливо перерахувати хоч би тому, що кожного дня з'являються все нові і нові продукти.

**Створення простої Web-служби Java.** Продовжимо розгляд прикладу метеослужби і опишемо її класом Java.

Назвемо цей клас Meteoservice. Наша метеослужба надає тільки одну Web-послугу – прогноз погоди на завтра. Нехай ця послуга представляється методом `getWeatherForecast` о. Складемо прогноз погоди, який найбільш вірогідний для України.

У прикладі далі приведений опис нашої Web-служби.

Приклад. Клас, що описує просту Web-службу.

```
public class MeteoServicef
public String getWeatherForecast(String location);
return location + «: піде дощ або сніг»;
```

Скористаємося Axis для реалізації нашої Web-служби. Для цього запишемо початковий текст класу Meteoservice у файл MeteoService.jws (розширення імені файлу «jws» означає «Java Web Service»), а файл занесемо в каталог axis нашого сервера застосувань, що використовує Axis. Каталог Axis повинен бути вкладений в каталог webapps сервера застосувань.

Тепер Web-служба створена і готова надати свої послуги будь-якому клієнту. Не потрібна ні компіляція, ні установка класу Meteoservice у Web-контейнер. Отримавши SOAP-повідомлення, що звертається до Web-служби прямо через файл MeteoService.jws, сервер застосувань, побачивши розширення імені файлу «jws», звернеться до

Axis, точніше до сервлету AxisServlet. З цього сервлету Axis починає роботу на сервері.

У наступному прикладі приведений клієнт метеослужби, написаний просто як Java-застосування, що використовує Axis.

Приклад. Клієнт метеослужби, написаний за допомогою Axis.

```
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import javax.xml.namespace.QName;
import java.net.*;
public class MeteoServiceClient{
public static void main(String[] args throws Exception;
if (args.length != 1){
System.err.println(«Usage: « +
«Java MeteoServiceClient <location>»»);
System.exit(1);
}
Service service = new Service();
Call call = (Call)service.createCallO;
String endpoint =
«http: //www.meteo.com: 8080/axis/MeteoService. jws»;call.
setTargetEndpointAddress(new URL|(endpoint));
call.setOperationName(new QName(«getWeatherForecast»));
String result =
(String)call.invoke(new Object[] {new String(args[0])});
System.out.println(«Погода на завтра: « + result);
}
}
```

Відкомпілюємо програму прикладу на машині клієнта:

```
$ javac MeteoServiceClient.java
```

і запустимо програму-клієнт:

```
$ java MeteoServiceClient Чернівці
```

На консолі побачимо прогноз погоди на завтра.

Як результат, для отримання послуги треба тільки знати адресу URL Web-служби, розміщеної у файлі MeteoService.jws, і передати їй ім'я населеного пункту. Все решта Axis бере на себе.

Поглянемо докладніше на те, що робить Axis в приведеному прикладі. Спочатку створюється об'єкт класу service. Цей об'єкт встановить зв'язок з Web-службою. Він надає екземпляр класу call, в якому формується HTTP-запит, що містить SOAP-повідомлення. Для створення запиту в об'єкті класу call заноситься адреса Web-служби і ім'я Web-послуги «getWeatherForecast».

Після того, як запит сформований, Axis звертається до Web-послуги методом `invoke ()`. Аргумент цього методу – масив об'єктів, що містить аргументи Web-послуги, що надається.

Запит прямує по мережі серверу застосувань, що працює по вказаній в запиті адресі URL. Сервер застосувань запускає сервлет `AxisServlet`, що грає роль диспетчера і контейнера, в якому працюють Web-служби. Він розбирає SOAP-повідомлення, що прийшло в запиті, відшукує вказаний в повідомленні файл `MeteoService.jws` і компілює його вміст. Потім Axis завантажує `MeteoService.class` і звертається до методу `getWeatherForecast ()`, передавши йому аргумент. Потім виконання цього методу Axis формує SOAP-відповідь і відправляє її клієнтові.

Клієнтська частина Axis розбирає отриману відповідь і передає її як результат виконання методу `invoke ()`.

Програму-клієнт Web-служби, показану в попередньому прикладі, можна вкласти в клієнтське застосування або використовувати окремо, забезпечивши її графічним інтерфейсом. Запит, сформований Axis і надісланий клієнтом нашої метеослужби на сервер, без HTTP-заголовку виглядає буквально так, як показано в прикладі далі.

Приклад. SOAP-повідомлення клієнта метеослужби

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<getWeatherForecast>
<arg0 xsi:type="xsd:string">Чернівці</arg0>
</getWeatherForecast>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Як бачите, ніде в запиті немає згадування про платформу, на якій працює клієнт, і тим більше не сказано, що клієнтом має бути Java-застосування. Тому клієнтом Axis може бути будь-яка програма, що може скласти такий запит і розібрати відповідь сервера. Її можна написати на будь-якій мові.

Приведемо в прикладі далі клієнт, написаний лише стандартними мережевими засобами Java, без використання Axis.

Приклад. Клієнт метеослужби

```
import java.net.*;
import java.io.*;
public class MeteoServiceClient2{
public static void main(String[] args){
if (args.length != 1){
System.err.println(«Usage: « +
```

```
«java MeteoServiceClient2 <location>»);
System.exit(1);
String message =
«<?xml version="1.0" encoding="UTF-8"?>» +
«<SOAP-ENV:Envelope>» +
«<SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">» +
«<xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">» +
«<xmlns:xsd="http://www.w3.org/2001/XMLSchema">» +
«<xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">» +
«<xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">» +
«<SOAP-ENV:Body>» +
«<getWeatherForecast>» +
«<arg0 xsi:type="xsd:string">» + args[0] + «</arg0>» +
«</getWeatherForecast>» +
«</SOAP-ENV:Body>» +
«</SOAP-ENV:Envelope>» ;
try{
byte[] data = message.getBytes(«KOI8-R»);
URL url =
new URL(«http://www.meteo.com:8080/axis/MeteoService.jws»);
URLConnection uc = url.openConnection();
uc.setDoOutput(true);
uc.setDoInput(true);
uc.setUseCaches(false) ;
uc.setRequestProperty(«Content-Type»
«text/xml; charset="utf-8"»); uc.setRequestProperty(«Content-Length»
«» + message.length ());
uc.connect();
DataOutputStream dos =
new DataOutputStream(uc.getOutputStream());
dos.write(data, 0, data.length);
dos.close();
BufferedReader br = new BufferedReader(
new InputStreamReader(
uc.getInputStream(), «KOI8-R»));
String res = null;
while ((res = br.readLine()) != null)
System.out.println(res);
br.close();
}catch(Exceptione){
System.err.println(«From client: « + e);
e.printStackTrace(System.out);
}
```

```
}
}
```

Відповідь, отримана клієнтом в даному прикладі, приведена в прикладі наступному.

Приклад. SOAP-відповідь метеослужби

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<SOAP-ENV:Envelope
```

```
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
```

```
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><SOAP-ENV:Body>
```

```
<getWeatherForecastResponse
```

```
SOAP-ENV:encodingStyle=
```

```
'http://schemas.xmlsoap.org/soap/encoding/1>
```

```
<getWeatherForecastResult xsi:type='xsd:string'>
```

```
Чернівці: піде дощ або сніг
```

```
</getWeatherForecastResult>
```

```
</getWeatherForecastResponse>
```

```
</SOAP-ENV:Body>
```

```
</SOAP-ENV:Envelope>
```

Звичайно, клієнт, показаний в попередньому прикладі, повинен ще розібрати відповідь надіслану сервером у вигляді SOAP-повідомлення останнього прикладу, виділивши з нього результат – прогноз погоди. Для цього доведеться застосувати SAX-парсер.

Слід відмітити, що та частина Axis на сервері, яка приймає SOAP-повідомлення, оформлена у вигляді сервлету AxisServlet. Дійсно, в технології Java Web-служби реалізуються сервлетами і сторінками JSP, які, можливо, звертаються до компонентів EJB типу SLJB (Stateless Session Bean) або типу MDB (MessageDriven Bean). Нашу просту метеослужбу можна реалізувати без Axis одним сервлетом, наприклад, таким, який показаний в прикладі далі.

Приклад. Сервлет, що реалізовує метеослужбу

```
import java.io.*;
```

```
import java.util.*;
```

```
import javax.servlet.http.*;
```

```
public class MeteorServiceServlet extends HttpServletf
```

```
public String getWeatherForecast(String location))
```

```
return location + «: піде дощ або сніг»;
```

```
}
```

```
public void doPost(HttpServletRequest req,
```

```
HttpServletResponse resp){
```

```
try{
```

```
req.setCharacterEncoding("K0I8-R");
```

```
BufferedReader br = req.getReader();
```

```
String message = «», buf = «», location = «»;
```

```
while ((buf = br.readLine()) != null)
```

```
message += buf;
```

```
// Аналіз SOAP-повідомлення message SAX-парсером і
```

```
// виділення аргументу location.
```

```
resp.setContentType("text/xml; charset="utf-8");
```

```
String response =
```

```
<<?xml version="1.0" encoding="UTF-8"?>> +
```

```
<<SOAP-ENV:Envelope> +
```

```
<<Kmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"> +
```

```
<<xmlns:xsd="http://www.w3.org/2001/XMLSchema"> +
```

```
<<xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">> +
```

```
<<SOAP-ENV:Body>> +
```

```
<<getWeatherForecastResponse> +
```

```
<<SOAP-ENV:encodingStyle=> +
```

```
'http://schemas.xmlsoap.org/soap/encoding/'>> +
```

```
<<getWeatherForecastResult xsi:type='xsd:string'>> +
```

```
getWeatherForecast(location)+
```

```
<</getWeatherForecastResult>> +
```

```
<</getWeatherForecastResponse>> +
```

```
<</SOAP-ENV:Body>> +
```

```
<</SOAP-ENV:Envelope>>;
```

```
PrintWriter pw = resp.getWriter();
```

```
pw.println(response);pw. flush () ;
```

```
pw. close () ;
```

```
}catch(Exception e){
```

```
System.err.println(e);
```

```
}
```

```
}
```

```
}
```

Вся наша проста метеослужба уміщалася в одному методі getWeatherForecast (). У реальній ситуації її краще винести в один або декілька компонентів EJB.

Отже, побудувати будь-яку Java Web-службу можна стандартними вільно доступними засобами Java, які є в пакетах інструментальних засобів J2SDK SE, J2SDK EE і WSDP. Їх можна вільно завантажити з сайту <http://java.sun.com/>.

### 2.6.3. Опис Web-служби

Після створення Web-служби і розміщення її на сервері у вигляді сервлету сторінки JSP, JWS-файлу або іншого об'єкту, слід подумати, як про цю службу дізнаються клієнти. На кожному сервері необхідно створити список Web-служб, що є на ній, і послуг, що надаються ними. Але для цього треба точно знати, які саме послуги надає кожна Web-служба і описати їх.

Засоби для такого опису є в багатьох мовах – це інтерфейси Java, мова IDL, на якій описуються об'єкти CORBA, заголовкові header-файли мови C. Для опису Web-служб потрібна мова, не залежна від платформи, операційної системи, системи програмування. Тому для точного опису Web-послуг консорціумом W3C створено спеціальну мову WSDL. Ця мова – ще одна реалізація XML. Її специфікація опублікована на сторінці <http://www.w3.org/TR/wsdl>.

Кореневим елементом документа XML-опису WSDL служить елемент `<definitions>`. В цьому елементі атрибуту `name` можна дати ім'я опису. Крім того, це зручне місце для введення використовуваних в описі просторів імен. Описи WSDL активно використовують різні простори імен. Крім власних імен WSDL часто використовує мову опису схем XSD і мову протоколу SOAP. Простір імен WSDL часто описується як простір імен за замовчуванням, тоді цільовий простір імен одержує префікс, зазвичай `tns` (Target NameSpace). В кореневий елемент вкладаються п'ять основних елементів:

- `<types>` – визначає складні типи XSD, що використовуються Web-службою. Цей елемент не потрібний, якщо служба застосовує тільки прості типи XSD;
- `<message>` – описує кожне SOAP-посилання: запит, відповідь, пересилку документів. У цей елемент вкладаються елементи `<part>`, що описують ім'я і тип кожного аргументу запиту або значення, що повертається;
- `<portType>` – описує Web-службу, що називається пунктом призначення (endpoint) або портом (port) прибуття повідомлення. Вона описується як набір послуг, що звуться операціями. Операції описуються вкладеними елементами `<operation>`, що описують кожну послугу: відправку запиту – отримання відповіді або, навпаки, отримання запиту – відправку відповіді. Отримання і відправка, у свою чергу, описується вкладеними елементами `<input>` і `<output>`, а повідомлення про помилку – елементом `<fault>`;
- `<binding>` – описує конкретний формат пересилки: протоколи SOAP, HTTP, тип вмісту HTML, XML або інший MIME-тип повідомлення. У кожній Web-службі може бути декілька таких елементів, по одному для кожного способу пересилки;
- `<service>` – вказує місце розташування Web-служби як один або декілька портів. Кожен порт описується вкладеним елементом `<port>`, що містить адресу Web-служби, заданої по правилах вибраного в елементі `<binding>` способу пересилки.
- Крім цих п'яти основних елементів є ще два додаткові елементи:
- `<import>` – включає файл з XSD-схемою опису WSDL або інший WSDL-файл;
- `<documentation>` – коментар. Його можна включити в будь-який елемент опису WSDL.

Можна сказати, що елементи `<types>`, `<message>` і `<portType>` показують, що є в описуваній Web-службі, які послуги вона надає, як організовані послуги, які типи даних цих послуг.

Елементи `<binding>` пояснюють, як реалізована Web-служба, який протокол передачі повідомлень: HTTP, SMTP або щось ще, задає технічні характеристики передачі даних.

Нарешті, елементи `<service>` показують, де знаходиться Web-служба, пов'язуюча опис `<binding>` з конкретними адресами Web-служби.

У прикладі нижче показано, як виглядатиме WSDL-опис нашої метеослужби.

Приклад. Опис WSDL метеослужби

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="MeteoService"
targetNamespace="http://www.meteo.com/wsdl/MeteoService.wsdl"
xmlns="http://www.w3.org/2012/07/wsdl"
xmlns:soap="http://www.w3.org/2012/07/wsdl/soap12"
xmlns:tns="http://www.mi.com/wsdl/MeteoService.wsdl"
xmlns:xsd="http://www.w3.org/2011/XMLSchema">
  <message name="ForecastRequest">
    <part name="location" type="xsd:string" />
  </message>
  <message name="ForecastResponse">
    <part name="forecast" type="xsd:string" />
  </message>
  <portType name="Meteo_PortType">
    <operation name="getWeatherForecast">
      <input message="tns:ForecastRequest" />
      <output message="tns:ForecastResponse" />
    </operation>
  </portType>
  <binding name="Meteo_Binding" type="tns:Meteo_PortType">
    <soap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http" />
    <operation name="getWeatherForecast">
      <soap:operation soapAction="getWeatherForecast" />
      <input>
        <soap:body encodingStyle=
"http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:MeteoService"
use="encoded" />
      </input>
      <output>
        <soap:body encodingStyle=
"http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:MeteoService"
use="encoded" />
      </output>
    </operation>
  </binding>
  <service name="Meteo_Service">
    <documentation>WSDL File for MeteoService</documentation>
    <port binding="tns:Meteo_Binding" name="Meteo_Port">
```

```
<soap:address
location=>http://www.meteo.com:8080/servlet/MeteoService />
</port>
</service>
</definition>
```

У наведеному прикладі ми дали в елементі `<definitions>` опис ім'я «MeteoService» і визначили префікси всіх потрібних нам просторів імен. Далі ми описали запит і відповідь в двох елементах `<message>`. Ми дали їм імена «ForecastRequest» і «ForecastResponse». У запиті один аргумент типу `xsd: string`. Цей тип визначений в мові XSD. Ми дали аргументу ім'я `location`, що співпадає з ім'ям аргументу методу `getweatherForecast()`. Значенню, що повертається методом, ми дали ім'я `forecast`. Його тип також `xsd: string`.

Імена «ForecastRequest» і «ForecastResponse» використані в наступному елементі `<portType>` для вказівки входних і вихідних параметрів Web-послуги. У ньому один елемент `<operation>`. Це означає, що Web-служба «Meteo\_PortType» надає одну послугу, ім'я якої «getweatherForecast», яка збігається з ім'ям методу, що виконує цю послугу. У елементі `<operation>` вказано входний `<input>` і вихідний `<output>` параметри послуги.

Потім, елементом `<binding>` ми вказуємо один спосіб пересилки повідомлень, а саме, SOAP-повідомлення в процедурному стилі, що пересилаються по протоколу HTTP, на що вказує елемент:

```
<soap:bindingstyle=>rpc
transport=>http://schemas.xmlsoap.org/soap/http />
```

Документний стиль SOAP наголошується значенням «document» атрибуту `style`.

Далі повторюється опис операції, але вже в термінах SOAP.

Нарешті, елементом `<service>` вказуємо адресу, по якій розташована Web-служба. Оглянемо інструментальні засоби для створення WSDL-описів.

**Інструменти створення описів WSDL.** Опис WSDL написаний в попередньому прикладі вручну. Однак строгість опису дозволяє автоматизувати цей процес. Багато інструментальних засобів створення Web-служб містять утиліти, які автоматично створюють WSDL-файли, що описують готові Web-служби. Наприклад, вже згадуваний засіб створення Web-служб Apache Axis містить у своєму складі клас `Java2WSDL`, що створює WSDL-файл по класу або інтерфейсу Java. Пакет IBM WSTK, до складу якого входить Axis, містить утиліту `java2wsdl`, що створює об'єкт цього класу і, що працює з командного рядка. Досить набрати в командному рядку

```
$ java2wsdl MeteoService
```

і буде створений файл `MeteoService.wsdl`, що містить по одному елементу `<portType>` для кожного відкритого методу класу, вказаного в командному рядку.

Цікаво, що Axis може виконати і зворотну дію: по наявному WSDL-файлу клас `WSDL2java` створить всі класи Java, які необхідні для роботи Web-служби. У па-

кеті IBM WSTK клас `WSDL2java` можна викликати з командного рядка утилітою `wsdl2java`. Наприклад:

```
$ wsdl2java MeteoService.wsdl
```

Такі ж утиліти є у складі вже згадуваного продукту GLUE фірми The Mind Electric.

Пакет Microsoft SOAP Toolkit містить графічну утиліту `wsdigen3.exe`, що викликається із стартового меню WSDL Generator, і утиліти командного рядка `wsdlstb3.exe`, яку створюють WSDL-файли.

Фірма Sun Microsystems готує до випуску пакет інтерфейсів JWSDL (Java API for WSDL), що перетворює опис WSDL в класи Java і назад.

Це дозволяє програмно створювати, змінювати, читати описи WSDL.

Фірма IBM вже реалізувала цей пакет в своєму продукті WSTK, назвавши набір інтерфейсів і класів `WSDL4J`, що реалізують їх (WSDL for Java). Цей набір можна використовувати у складі WSTK або окремо, завантаживши його з сайту <http://www-124.ibm.com/developerworks/projects/wsdl4j/>.

Найцінніше в описах WSDL те, що клієнт Web-служби може звернутися не до неї самої, а до її WSDL-опису. До складу GLUE входить утиліта командного рядка `invoke`, що звертається до Web-служби по її WSDL-опису. Наприклад, досить набрати в командному рядку:

```
$ invoke http://www.meteo.com:8080/services/MeteoService.wsdl Чернівці
```

— і на консолі з'явиться прогноз погоди.

Фірма IBM випускає пакет класів WSIF (Web Services Invocation Framework), що працює у Web-контейнері Tomcat під управлінням Apache SOAP. За допомогою цього пакету можна зробити ту ж роботу:

```
$ Java clients.DynamicInvoker \
http://www.meteo.com:8080/services/MeteoService.wsdl \
getWeatherForecast Чернівці
```

Нагадаємо, що зворотна похила риска тут означає продовження командного рядка на наступний рядок тексту.

Після цього на консолі з'являються повідомлення WSIF і прогноз погоди.

**Реєстрація Web-служб.** У всіх наведених вище прикладах Web-служба викликала за її адресою записаною рядком URI. Це прийнятно для виклику Web-послуг з командного рядка або графічного застосування, але незручно для роботи розподіленого застосування, оскільки адреса Web-служби може помінятися і доведеться зробити заміни у всіх клієнтських застосуваннях. Для стабільної роботи розподілених застосувань потрібний засіб автоматичного пошуку і зв'язку з Web-службами, подібно до реєстру RMI, служб іменування JNDI або CORBA Naming Service, мережевою інформаційною службою NIS, що використовується в UNIX. Це особливо важливо для слабозв'язаних застосувань: адже основна їх перевага — швидкий пошук «на льоту» необхідних для роботи компонентів. Створено вже декілька систем пошуку Web-служб. Найбільш поширені дві системи: універсальна система опису, виявлення і інтеграція UDDI та електронний бізнес-реєстр ebXML Registry (electronic business XML Registry). Обидві системи громіздкі і складні у використанні. Тому фірми IBM



і Microsoft розробили полегшену систему виявлення Web-служб, що зветься WS-Inspection (Web Services Inspection Language).

Схему взаємодії клієнта Web-служб з її постачальником через реєстр показано на рис. 2.6.3.1.

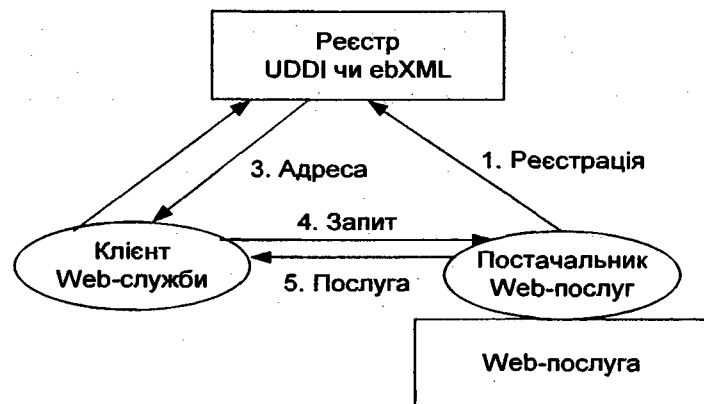


Рис. 2.6.3.1. Взаємодія клієнта Web-служби з її постачальником

Розглянемо послідовно ці три найбільш використовувані служби пошуку Web-служб.

**Система опису і виявлення UDDI.** Система опису, виявлення і інтеграції UDDI створена фірмами IBM (<http://www-3.ibm.com/services/uddi/>) і Microsoft (<http://uddi.microsoft.com/>).

Зараз вона розвивається групою крупних компаній. На офіційному сайті співтовариства UDDI <http://www.uddi.org/> приведений список близько трьохсот компаній-учасників проекту. Співтовариство випустило специфікацію UDDI, яку можна отримати на тому ж сайті. Специфікація вже реалізована низкою продуктів розробки реєстру UDDI.

Реєстр UDDI (UDDI Business Registry) складається з безлічі вузлів (nodes) розміщених в Інтернеті. Вони зберігають інформацію про Web-служби, доступну на всіх вузлах, що утворюють розподілений UDDI-реєстр. Клієнт «бачить» UDDI-реєстр, як єдине ціле, абсолютно не відчувачи того, що він розміщений на декількох машинах. Звичайно, самі вузли організовані як Web-служби, а реєстр UDDI – як слабо зв'язане розподілене застосування.

Багато крупних компаній організували і містять свої UDDI-реєстри, наприклад, реєстр фірми IBM розташований за адресою <https://www-3.ibm.com/services/uddi/v2beta/protect/registry.html>, адреса реєстру компанії Hewlett Packard <https://uddi.hp.com/uddi/index.jsp>, реєстр корпорації Microsoft знаходиться на сайті <https://uddi.rte.microsoft.com/register.aspx>. Ці UDDI-реєстри зв'язані між собою і постійно обмінюються інформацією. Крім того, це відкриті (public) реєстри. Так, що будь-який бажаючий може зареєструвати в них свою Web-службу або відшукати потрібну Web-послугу.

Фірми можуть організувати і закриті приватні (private) реєстри, доступні тільки зареєстрованим учасникам. Список UDDI-реєстрів можна бачити на сайті проекту UDDI.

Реєстр UDDI розбиває інформацію, що зберігається в ньому, на декілька груп.

Чотири основні групи складаються з наступних елементів:

- ⇒ **бізнес-інформація** – XML-елемент `<businessEntity>` – це опис фірми-постачальника Web-послуг: її ключ UUID (Unique Universal Identifier), унікальний в межах реєстру і описаний атрибутом `businessKey`, назва фірми – вкладений елемент `<name>`, короткий опис сфери її діяльності, типи послуг, що надаються, контактна інформація, посилання URL. Ця інформація призначена для всіх, хто хоче скористатися послугами фірми;
- ⇒ **бізнес-послуга** – елемент `<businessServices>`, вкладений в елемент `<businessEntity>` – список послуг, що надаються фірмою. Кожна послуга описується вкладеним елементом `<businessservice>`. У опис входить ключ UUID кожної послуги, описаний атрибутом `serviceKey`, ім'я послуги – вкладений елемент `<name>`, її короткий опис і посилання на докладну інформацію. Послуги можуть бути будь-якими, не обов'язково Web-послугами;
- ⇒ **вказівники на послуги** – елемент `<bindingTemplates>`, вкладений в елемент `<businessservice>` – це способи отримання кожної послуги. Вони можуть бути прямими, наприклад, URL-адреса Web-служби, або непрямими, наприклад, опис WSDL або IDL. Кожен спосіб описується елементом `<bindingTemplate>`. Його атрибут `bindingKey` визначає унікальний ключ UUID вказівника. Елемент `<bindingTemplate>` містить посилання на відповідний елемент `<tModel>`;
- ⇒ **модель послуги** – елемент `<tModel>` (technical Model) – докладний формальний опис кожної послуги. Він використовується програмним забезпеченням вузла. Зазвичай – це окремий документ XML. В реєстрі є ще декілька додаткових елементів:
- ⇒ **твердження** – елемент `<pubiisherAssertion>` – опис встановлених раніше відношень між фірмами (твердження peer-peer) або фірмою і її підрозділами (твердження parent-child). Фірма затверджує, що вона тісно пов'язана з перерахованими фірмами або що це – її підрозділи. Третій вид твердження – `identity` – відношення між однаковими фірмами – це фактично псевдонім. Відношення входить у силу, коли його затвердять обидва учасники. Це окремий документ, що використовує елементи `<businessEntity>` обох фірм;
- ⇒ **підписка** – елемент `<subscription>` – це список фірм і відомостей, які треба послати перерахованим фірмам при яких-небудь змінах в діяльності фірми-виготовлювача.

Отже, схема одного з документів XML, що зберігаються в UDDI-реєстрі, така:

```

<businessEntity businessKey=»ключ UUID»>
<businessServices>
<businessService serviceKey=» ключ UUID»>
<bindingTemplates>
<bindingTemplate bindingKey =»ключ UUID»>
<!-- Опис вказівника -->
</bindingTemplate>
<!-- Описи інших вказівників -->
</bindingTemplates>
</businessService>
  
```

```
<!-- Описи інших послуг -->
</businessServices>
</businessEntity>
```

Клієнт звертається до UDDI-реєстру для того, щоб зареєструвати свою Web-службу, змінити її, або для того, щоб відшукати потрібну Web-послугу. Реєстр надає необхідний для цього інтерфейс. Реєстрація і подальші зміни виконуються методами `save_xxx()`, пошук Web-служби і отримання послуги – методами `find_xxx()` і `get_xxx()`. Всі ці методи описані в специфікації UDDI.

Є вже багато реалізацій інтерфейсу UDDI на різних мовах. У технології Java найбільш популярний пакет класів UDDI4J (UDDI for Java) розроблений фірмою IBM (<http://www.uddi4j.org/>). Класи, що входять в цей пакет, забезпечують виконання всіх дій клієнта в UDDI-реєстрі.

UDDI-інтерфейс реалізований в пакеті UDDI4J класом `UDDIProxy`. Точніше кажучи, цей клас служить посередником (проку), що звертається до відповідних методів інтерфейсу UDDI. Методи цього класу повертають об'єкти класів, що надають додаткову інформацію.

У прикладі нижче показано, як можна зареєструвати нашу метеослужбу тому або іншому UDDI-реєстру, а потім знайти її.

Приклад. Реєстрація метеослужби в UDDI-реєстрі:

```
import com.ibm.uddi.*;
import com.ibm.uddi.datatype.business.*;
import com.ibm.uddi.response.*;
import com.ibm.uddi.client.*;
import org.w3c.dom.Element;
import java.util.Vector;
import java.util.Properties;
public class SaveMeteoService{
public static void main (String[] args){
SaveMeteoService app = new SaveMeteoService();
UDDIProxy proxy = new UDDIProxy();
try{
// Вибираємо UDDI-реєстр:
// Тестовий реєстр IBM
proxy.setInquiryURL(«http://www-3.ibm.com/services/» +
«uddi/testregistry/inquiryapi»);
proxy.setPublishURL(«https://www-3.ibm.com/services/» +
«uddi/testregistry/protect/publishapi»);
// Офіційний реєстр IBM
// proxy.setInquiryURLC'http://www-3.ibm.com/services/» +
// «uddi/inquiryapi»);
// proxy.setPublishURL(«https://www-3.ibm.com/services/» +
// «uddi/protect/publishapi»);
// Тестовий реєстр Microsoft
```

```
// proxy.setInquiryURL(
// «http://test.uddi.microsoft.com/inquire»);
// proxy.setPublishURL(
// «https://test.uddi.microsoft.com/publish»);
// Офіційний реєстр Microsoft
// proxy.setInquiryURLC'http://uddi.microsoft.com/inquire»);
// proxy.setPublishURL(«https://uddi.microsoft.com/publish»);
// Реєстр Hewlett Packard
// proxy.setInquiryURL(«http://uddi.hp.com/inquire»);
// proxy.setPublishURL(«https://uddi.hp.com/publish»);
// Налаштовувальний локальний реєстр з пакету WSTK
// proxy.setInquiryURL(
// «http://localhost:8080/services/uddi/inquiryapi»);
// proxy.setPublishURL(
// «http://localhost:8080/services/uddi/publishapi»);
// Заходимо на сайт UDDI-реєстра.
AuthToken token = proxy.get_authToken(
«userid», «password»);
System.out.println(
«Реєстраційний код: « + token.getAuthInfoString());
System.out.println(«Реєструємо Meteo Service»); Vector entities = new Vector();
// Створюємо елемент <businessEntity>.
// Перший аргумент – UUID – поки невідомий.
BusinessEntity be = new BusinessEntity(«», «MeteoService»);
entities.addElement(be);
// Реєструємо Web-службу
BusinessDetail bd =
proxy.save_business(token.getAuthInfoString(), entities);
// Для перевірки отримуємо UUID.
Vector businessEntities = bd.getBusinessEntityVector();
BusinessEntity returnedBusinessEntity =
(BusinessEntity)(businessEntities.elementAt(0));
System.out.println(«Отримали UUID: « +
returnedBusinessEntity.getBusinessKey());
System.out.println(«Список Web-служб:»);
// Отримуємо список зареєстрованих
// Web-служб, чії імена починаються з букви «М».
BusinessList = proxy.find_business(«М», null, 0);
Vector businessInfoVector =
bl.getBusinessInfos().getBusinessInfoVector();
for (int i = 0; i < businessInfoVector.size(); i++){
BusinessInfo businessInfo =
(BusinessInfo)businessInfoVector.elementAt(i);
System.out.println(businessInfo.getNameString());
```

```

}catch(UDDIException e){
DispositionReport dr = e.getDispositionReport();
if (dr != null){
System.out.println(
«ODDIException faultCode:» + e.getFaultCode() +
«\n operator:» + dr.getOperator() +
«\n generic:» + dr.getGeneric() +
«\n errno:» + dr.getErrno() +
«\n errCode:» + dr.getErrCode() +
«\n errInfoText:» + dr.getErrInfoText());
}
e.printStackTrace();
}catch(Exception e){
System.err.println(«From proxy: « + e);
}
}
}

```

**Система взаємодії фірм ebXML.** Електронний бізнес-реєстр ebXML спочатку був створений двома організаціями: центром міжнародної торгівлі і електронного бізнесу ООН UN/CEFACT (United Nations Centre for Trade Facilitation and Electronic Business) (<http://www.uncefact.org/>) і громадською організацією по стандартизації структурної інформації OASIS (Organization for the Advancement of Structured Information Standards) (<http://www.oasis-open.org/>).

Потім в розробку проекту ebXML включилися інші фірми, зараз їх вже декілька сотень, в їх числі IBM і Sun. Всю інформацію про ebXML можна отримати на сайті проекту <http://www.ebxml.org/>.

Мета проекту ebXML широка – забезпечити взаємодію між діловими партнерами по Інтернету, що зазвичай позначається скороченням B2B (Business to Business). Взаємодія може бути будь-якою, Web-служби – це тільки окремий випадок, причому не найважливіший. Тому в множині специфікацій проекту ebXML (<http://www.ebxml.org/specs/>) описується тільки каркас побудови реєстру. Специфікація ebXML робить це дуже розпливчато, в найзагальніших термінах.

З точки зору ebXML є дві фірми, що називаються сторонами (parties), які здійснюють ділову співпрацю (Business Collaboration). У реєстрі ebXML зберігається інформація про кожну сторону і їх співпрацю, яка розділена на дві частини: заява про співпрацю CPP (Collaboration Protocol Profile) і угода про співпрацю CPA (Collaboration Protocol Agreement).

**Заява про співпрацю CPP.** Заява про співпрацю CPP – це документ XML, що містить інформацію про фірму-сторону, послуги, що надаються нею, можливих угод з іншими сторонами, способах обміну послугами і отримання інформації, у тому числі про транспортні протоколи. Заява CPP відкрита для всіх, хто бажає співробітничати з фірмою. Кожна сторона може зареєструвати в реєстрі одну заяву CPP, декілька заяв, або не реєструвати їх взагалі.

Структура документа XML, що містить заяву CPP, така:

```

<CollaborationProtocolProfile
xmlns=»http://www.ebxml.org/namespaces/tradePartner»
xmlns:ds=»http://www.w3.org/2010/09/xmldsigft»
xmlns:xlink=»http://www.w3.org/2009/xlink»
version=»1.1»>
<PartyInfo>
<!-- Відомості про фірму-сторону -->
</PartyInfo>
<Packaging id=»ID»>
<!-- Опис XML-структури повідомлень -->
</Packaging>
<ds:Signature>
<!-- Цифровий підпис (необов'язковий) -->
</ds:Signature>
<Соттегл коментар (необов'язковий) </Comment>
</CollaborationProtocolProfile>

```

У кожного елемента цього документа є вкладені елементи, що детально описують заяву CPP. Угода про співпрацю CPA – це документ XML, що містить технічні відомості про угоди двох сторін. Ці відомості збираються із заяв CPP двох фірм-сторін, що бажають співробітничати, або складаються фірмою, що бажає отримати послуги.

Схема документа XML, що містить угоду CPA така:

```

<CollaborationProtocolAgreement
xmlns=»http://www.ebxml.org/namespaces/tradePartner»
xmlns:bpm=»http://www.ebxml.org/namespaces/businessProcess»
xmlns:ds = «http://www.w3.org/2010/09/xmldsigft»
xmlns:xlink = «http://www.w3.org/2009/xlink»
cpa1c1=»Ідентифікатор документа»
version=»1.2»>
<!-- Стан документа: proposed, agreed, signed -->
<Status value = «proposed»/>
<Start>Дата і час набирання чинності угоди</Start>
<End>Дата і час закінчення угоди</End>
<!-- Домовленість про тривалість переговорів -->
<ConversationConstraints invocationLimit = «100»
concurrentConversations = «4»/>
<PartyInfo>
<!-- Відомості про одну сторону -->
</PartyInfo>
<PartyInfo>
<!-- Відомості про іншу сторону -->
</PartyInfo>
<Packaging id=»Н
<!-- Структура повідомлень -->

```

```

</Packaging>
<ds:Signature><!-- Цифровий підпис (необов'язково) -->
</ds:Signature>
<Comment>Коментар (необов'язковий)</Comment>
</CollaborationProtocolAgreement>

```

В елементів цього документа є вкладені елементи дуже складної структури, які ми зараз не розглядатимемо.

Вся інформація, включаючи CPP і CPA, заноситься в сховище (repository) разом з будь-якою додатковою інформацією.

Під час переговорів і після них сторони обмінюються повідомленнями (messages).

Угода досягається так:

1. Сторона А, що надає послуги, складає заяви CPP і реєструє CPP в реєстрі ebXML;
2. Сторона В, що бажає отримати послуги, знаходить в реєстрі потрібну заяву CPP сторони А;
3. Сторона В створює свою заяву CPP, потім на підставі обох заяв складає проект угоди CPA і посилає його стороні А;
4. Сторони досягають угоди, обмінюючись повідомленнями;
5. Після досягнення угоди сторони зберігають копії CPA на своєму сервері або в сховищі ebXML;
6. В процесі співпраці сервери обох сторін обмінюються інформацією згідно досягнутої угоди CPA.

**Реалізація ebXML.** Фірма Sun готує пакет сервлетів і компонентів EJB, що заздалегідь названо Sun ebXML Registry/Repository Implementation. Цей пакет встановлюється в сервер застосувань iPlanet Application Server. Як сховище пакет використовує СУБД Oracle.

Познайомитися з цією розробкою фірми Sun і отримати початкові тексти її компонентів можна за адресою <http://www.sun.com/software/xml/developers/regist/>.

Інша реалізація реєстру ebXML готується в рамках проекту EBXMIRR (OASIS ebXML Registry Reference Implementation Project).

Поточний стан проекту можна побачити на сайті <http://ebxmirr.sourceforge.net/>.

**Мова WS-Inspection для пошуку Web-служб.** Системи UDDI і ebXML вирішують не лише задачу пошуку Web-служби, але і завдання їх опису, зміни, інтеграції. Це ускладнює ведення реєстру і пошуку в ньому потрібної Web-служби. Якщо ж потрібно лише відшукати дану Web-службу на конкретному сайті, то краще скористатися іншими засобами пошуку.

Один такий засіб пошуку – WS-Inspection – створено фірмами IBM і Microsoft в 2011 році. Однією з цілей створення нового засобу було запропоноване спрощення пошуку Web-служби на Web-сайті. За допомогою WS-Inspection це завдання вирішується так: складається опис Web-служби на спеціально розробленій мові WSIL (WS-Inspection Language) – ще одній реалізації XML. Воно записується у файл з розширенням wsil і заноситься в певний каталог сервера застосувань. Сервер, отримавши запит, проглядає всі файли з розширенням wsil і знаходить запрошену Web-службу.

Опис Web-служби на мові WSIL виконується всього одним документом XML з кореневим елементом <inspection>, що містить декілька вкладених елементів.

У кореневий елемент <inspection> вкладається необов'язковий короткий опис Web-служби <abstract>, що служить коментарем, і одне або декілька описів Web-послуг <service> та одне або декілька посилань на описи <link>. У документі обов'язково має бути присутнім хоча б один елемент <service> або <link>.

У елементі <service> також може бути короткий опис Web-послуги <abstract>, одне або декілька імен <name> і обов'язковий один або декілька описів <description>, в яких необов'язковим атрибутом location задаються URI-адреси документів XML, що містять описи Web-служби. Опис зазвичай виконується на мові WSDL, але може бути зроблено і на іншій мові. Простір імен опису задається іншим атрибутом елементу <description> – атрибутом referencedNamespace.

У елемент <description> можна вкласти опис <abstract>, що служить коментарем, і один довільний елемент з додатковим описом. У цьому описі може бути адреса Web-служби – у такому разі воно замінить значення атрибуту location.

Елемент <link> зовні відрізняється від елементу <service> тим, що в ньому немає опису <description>, а атрибути referencedNamespace і location відносяться безпосередньо до елементу <link>. Ці атрибути описують простір імен і адресу іншого WSIL-опису або сховища UDDI.

Замість атрибуту location можна записати вкладений елемент, в якому вказується посилання на інший опис.

Отже, WSIL-опис будується за наступною схемою:

```

<inspection
xmlns=»http://schemas.xmlsoap.org/ws/2011/10/inspection/»>
<abstract>Короткий опис Web-служби</abstract>
<service>
<abstract>Короткий опис Web-служби</abstract>
<name>Довільне ім'я послуги</name>
<description referencedNamespace=»адреса URI»
location=» адреса URI»>
<abstract>Короткий опис Web-служби</abstract>
<!-- додатковий опис -->
</description>
</service>
<link referencedNamespace=»адреса URI»
location=» адреса URI»>
<abstract>Короткий опис Web-служби</abstract>
<!-- додатковий опис -->
</link>
</inspection>

```

В наступному прикладі показано, як просто виглядає WSIL-опис нашої метеослужби:

**Приклад. Опис WSIL метеослужби**

```
<?xml version="1.0"?>
<inspection
xmlns="http://schemas.xmlsoap.org/ws/2011/10/inspection/">
<service>
<name>MeteoService</name>
<description
referencedNamespace="http://schemas.xmlsoap.org/wsdl/"
location="http://www.meteo.com/wsdl/MeteoService.wsdl" />
</service>
</inspection>
```

Після того, як WSDL-опис зроблений, його треба записати у файл з іменем, наприклад, `MeteoService.wsil`, а файл помістити в певний каталог серверу застосувань. Якщо звернутися до серверу із браузера з боку клієнта, наприклад, так: `http://www.meteo.com/common/wsdl/inspection.wsil`, то браузер покаже список всіх Web-служб, що є на сервері. Специфікація WS-Inspection пропонує, щоб в кореновому каталозі сайту реалізовуючого WS-Inspection, розташовувався або створювався динамічно файл `inspection.wsil`. Ви можете набрати в браузері, наприклад, адресу фірми IBM `http://www.ibm.com/inspection.wsil` або адресу `http://www.xmethods.com/inspection.wsil` і отримати список всіх Web-служб, зареєстрованих на цьому сайті. При цьому файлу `inspection.wsil` на сервері немає, він створюється динамічно по запиті клієнта.

Сервер застосувань повинен "розуміти" WS-Inspection. Для цього на ньому має бути зроблена реалізація специфікації WS-Inspection, наприклад, встановлено пакет інтерфейсів і класів WSIL4J, що входить в склад IBM WSTK.

У пакеті WSIL4J розроблені Java-інтерфейси WS-Inspection і зроблена їх реалізація класами Java. Основну роботу на сервері виконує сервлет `WSILServlet`. Він проглядає всі файли із розширенням імені `wsil` і відправляє їх вміст клієнтові. Якщо у Web-служби немає WSIL-опису, то сервлет створює його динамічно.

У клієнта працює клас `WSILProxy`. Він запрошує `WSILServlet`, читає отримані від нього WSIL-документи і вибирає посилання на опис WSDL або інший, який там є опис потрібної Web-служби. Після цього можна звернутися до знайденої Web-служби.

У наступному прикладі наведений приклад пошуку нашої метеослужби за допомогою WS-Inspection і пакету WSIL4J. Програма просто виводить отримані від серверу описи метеослужби.

Приклад. Пошук метеослужби за допомогою WS-Inspection

```
import com.ibm.wsil.*;
import com.ibm.wsil.client.*;
import com.ibm.wstk.wsdl.*;
import java.util.*;
public class MeteoWSInspection{
public static void main(String[] args){
if (args.length < 2){
System.err.println(
«Usage: MeteoWSInspection <URL> <serviceName>»);
```

```
System.exit(1);
try{
WSILProxy proxy = new WSILProxy(args[0]);
WSDLDocument[] wsdlDocs =
proxy.getWSDLDocumentByServiceName(args[1]);
for (int i = 0; i < wsdlDocs.length; i++)
System.out.println(wsdlDocs[i].serializeToXML());
}catch(Exception e){
System.err.println(e);
}
}
```

**Пакет JAXR.** Отже, вже розроблено декілька різних систем реєстрації і пошуку Web-служб: UDDI, ebXML, WS-Inspection. Всі ці системи вимагають різних методів доступу до реєстру і роботи з ним. Для кожної системи доводиться створювати свого клієнта, що працює з реєстром по правилах даної системи пошуку Web-служб.

Фірма Sun вирішила розробити єдину методику доступу до реєстрів різних типів і створила набір інтерфейсів JAXR (Java API for XML Registries).

Цей набір реалізований в пакеті Sun WSDP. Там же наведений приклад клієнта JAXR. Це графічна утиліта переглядання реєстрів Registry Browser.

Вона викликається з командного рядка просто набором імені командного файлу `$ jaxr-browser`

У складі WSDP є і простий тренувальний реєстр, реалізований у вигляді сервлету `RegistryServerServlet`. Реєстр працює у Web-контейнері Tomcat і зберігає інформацію в невеликій базі даних Apache Xindice, що теж входить до складу WSDP. Для запуску реєстру треба запустити Tomcat і стартувати базу даних:

```
$ cd $WSDP_HOME/bin
$ startup
$ xindice-start
```

Для перевірки роботи реєстру і для посилання йому повідомлень застосовується спеціальна утиліта, що працює з командного рядка. Для входу в реєстр треба набрати наступний командний рядок:

```
$ registry-server-test run-cli-request -Dxml/GetAuthToken.xml
```

Для роботи з реєстром прямо через базу даних Xindice у складі WSDP є графічна утиліта `Indri`. Вона запускається з командного рядка:

```
$ registry-server-test run-indri
```

Оскільки набір JAXR розрахований на роботу з реєстрами різних типів, то він містить тільки інтерфейси, які повинен реалізувати постачальник послуг (service provider) конкретного реєстру. Інтерфейси будуть по-різному реалізовані постачальником послуг UDDI і постачальником послуг ebXML, але клієнт цього не помічає,

він просто зв'язується з постачальником послуг і користується інтерфейсами пакету JAXR. Схема звертання клієнта до реєстру через постачальника послуг показана на рис. 2.6.3.2.

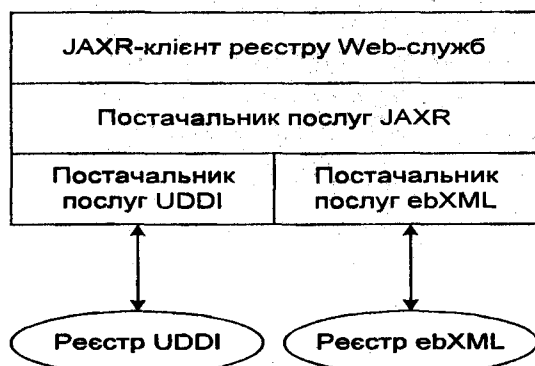


Рис. 2.6.3.2. Звернення до реєстру Web-служб.

Набір JAXR складається з двох пакетів `javax.xml.registry` і `javax.xml.registry.infomodel`. Інтерфейси першого пакету застосовуються клієнтом для роботи з реєстром, інтерфейси другого пакету – постачальником послуг для перетворення інформації до вигляду, придатного для збереження в базі даних реєстру.

Для зв'язку з постачальником послуг клієнт користується інтерфейсом `connection`. Екземпляр цього класу клієнт отримує за допомогою класу-фабрики `ConnectionFactory`. Спочатку статичним методом `newInstance()` створюється об'єкт цього класу, потім методом `setProperty` в нього заносяться характеристики реєстру, після цього методом `createConnection()` створюється об'єкт типу `connection`.

Після того, як зв'язок з постачальником послуг встановлено, методом `getRegistryService()` інтерфейсу `Connection` отримуємо об'єкт типу `RegistryService`, в якому зосереджені методи роботи з реєстром. Наприклад, методом `getBusinessQueryManager()` можна отримати об'єкт типу `BusinessQueryManager`, що містить методи `findxxx()` отримання інформації з реєстру.

У прикладі нижче показано, як можна зв'язатися з реєстром і отримати з нього різну інформацію за допомогою JAXR.

Приклад. Отримання інформації з реєстру за допомогою JAXR

```
import javax.xml.registry.*;
import javax.xml.registry.infomodel.*;
import java.net.*;
import java.util.*;
public class JAXRClient{
    public static void main(String[] args){
        // Декілька адрес реєстрів на вибір.
        String queryURL =
            "http://www-3.ibm.com/services/uddi/inquiryapi";
        //»http://uddi.rte.microsoft.com/inquire»;
        //»http://localhost:8080/registry-server/RegistryServerServlet»;
        if (args.length < 1){
```

```
System.out.println("Usage: Java JAXRClient <name>»");
System.exit(1);
}
JAXRClient jq = new JAXRClient();
String httpProxyHost = "localhost";
String httpProxyPort = "8080";
Properties props = new Properties();
props.setProperty("javax.xml.registry.queryManagerURL»,
    queryURL);
props.setProperty("com.sun.xml.registry.http.proxyHost»,
    httpProxyHost);
props.setProperty("com.sun.xml.registry.http.proxyPort»,
    httpProxyPort);
Connection connection = null;
try{
    ConnectionFactory factory =
        ConnectionFactory.newInstance();
    factory.setProperties(props);
    connection = factory.createConnection();
    System.out.println("Зв'язок з реєстром встановлений»");
    RegistryService rs = connection.getRegistryService();
    BusinessQueryManager bqm = rs.getBusinessQueryManager();
    Collection findQualifiers = new ArrayList();
    findQualifiers.add(FindQualifier.SORT_BY_NAME_DESC);
    Collection namePatterns = new ArrayList();
    namePatterns.add("%» + args[0] + "%»");
    // Пошук фірми по імені.
    BulkResponse response =
        bqm.findOrganizations(findQualifiers
            namePatterns, null, null, null, null);
    Collection orgs = response.getCollection();
    // Відомості про фірму.
    Iterator orgIter = orgs.iterator();
    while (orgIter.hasNext()){
        Organization org = (Organization) orgIter.next();
        System.out.println("Назва: " +
            org.getName().getValue());
        System.out.println("Опис: " +
            org.getDescription().getValue());
        System.out.println("Ідентифікатор: " +
            org.getKey().getId() );
        // Контактна інформація
        User pc = org.getPrimaryContact();if (pc != null){
            PersonName pcName = pc.getPersonName();
```

```
System.out.println("Назва: " +
pcName.getFullName());
Collection phNums =
pc.getTelephoneNumbers(null);
Iterator phlter = phNums.iterator();
while (phlter.hasNext()){
    TelephoneNumber num =
    (TelephoneNumber)phlter.next();
    System.out.println("Номер телефону: " +
    num.getNumber() );
}
Collection eAddrs = pc.getEmailAddresses();
Iterator ealter = eAddrs.iterator();
while (ealter.hasNext()){
    EmailAddress eAd =
    (EmailAddress) ealter.next();
    System.out.println("E-mail: " +
    eAd.getAddress());
}
}
// Послуги і доступ до них
Collection services = org.getServices();
Iterator svcIter = services.iterator();
while (svcIter.hasNext()){
    Service svc = (Service) svcIter.next();
    System.out.println("Назва послуги: " +
    svc.getName().getValue());
    System.out.println("Опис послуги: " +
    svc.getDescription().getValue());
    Collection serviceBindings =
    svc.getServiceBindings();
    Iterator sblter = serviceBindings.iterator();
    while (sblter.hasNext()){
        ServiceBinding sb =
        (ServiceBinding)sblter.next();
        System.out.println("Адреса URI: " +
        sb.getAccessURK() );
    }
}
System.out.println(" ");
}
}catch(Exception e){
    e.printStackTrace();
}finally!
```

```
if (connection != null)
try{
    connection.close();
}catch(JAXRException je){}
}
}
}
```

Стек протоколів Web Services. Отже, ми описали архітектуру Web-служб і зробили короткий огляд частин її компонентів. Ми побачили, що архітектуру Web-служб складає маса протоколів і специфікацій. Їх можна розбити на чотири частини, що утворюють стек протоколів, в якому кожен верхній рівень спирається на нижній рівень. Основні протоколи цього стеку показані на рис. 2.6.3.3.

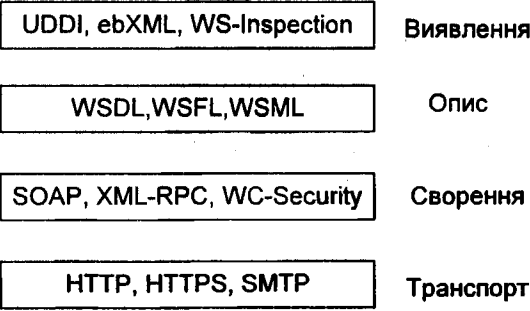


Рис. 2.6.3.3. Стек протоколів Web Services



## Контрольні запитання

1. Що таке JavaScript?
2. Якій мові аналогічний синтаксис JScript?
3. У якому форматі пишеться код на JavaScript?
4. Який знак застосовується при порівнянні двох значень на рівність у мові JavaScript?
5. Скільки типів даних використовується у JavaScript?
6. Які значення допускає логічний тип?
7. Що означає Undefined?
8. Скільки видів функцій є у JavaScript?
9. Дайте означення Visual Basic Scripting Edition(VBScript)?
10. Що являє собою програма Java-апплет?
11. Яка загальна характеристика ActionScript?
12. Яким плеєром виконуються SWF-файли?
13. Яку модель представляє браузеру Microsoft Silverlight?
14. Що дозволяє DHTML?
15. Що представляє собою мова програмування Perl?
16. Що ви можете сказати про мову PHP?
17. З чого складається програма на мові Perl?
18. Що ви можете сказати про ASP.NET?
19. Простори імен скількох груп містить бібліотека базових класів .NET?
20. Прийнято виділяти три типи серверних елементів управління. Назвіть їх?
21. Як виглядає типовий сценарій роботи веб-застосунку з джерелом даних?
22. Для чого використовується мова JavaScript?
23. Як визначити в HTML- документі код JavaScript?
24. Який синтаксис запису змінних в JavaScript?
25. Який синтаксис запису виразів в JavaScript?
26. Охарактеризуйте типи даних JavaScript?
27. Назвіть оператори JavaScript?
28. Назвіть оператори вибору JavaScript?
29. Назвіть оператори циклів JavaScript?
30. Як достроково закінчити цикл?
31. Як достроково перейти на наступну ітерацію циклу?
32. Які правила запису функцій?
33. Що таке об'єкт в мові JavaScript?
34. Як визначити об'єкт користувача в JavaScript?
35. Охарактеризуйте стандартні об'єкти JavaScript?
36. Як реалізована інтерактивність в JavaScript?
37. Які стандартні функції JavaScript?
38. Назвіть методи та властивості об'єкту Array?
39. Назвіть методи та властивості об'єкту Date?
40. Назвіть методи та властивості об'єкту Math?
41. Назвіть методи та властивості об'єкту String?
42. Назвіть методи та властивості об'єкту Window?

43. Як за допомогою JavaScript відкрити нове вікно браузеру?
44. Як за допомогою JavaScript запрограмувати виконання деяких команд після закінчення встановленого терміну часу?
45. Назвіть методи та властивості об'єкту Document?
46. Назвіть методи та властивості об'єкту Location?

## Тести для закріплення матеріалу

1. ... – інтерпретована, об'єктивно-орієнтована мова.

- A) JavaScript;
- Б) DOM;
- В) Perl.

2. – мова з нестрогим контролем типів.

- A) Undefined;
- Б) JavaScript;
- В) PHP;
- Г) Microsoft.

3. ... – означає, що тип невизначений.

- A) Undefined;
- Б) JavaScript;
- В) Unfined;
- Г) DHTML.

4. У JavaScript є два види функцій:

- A) true і false;
- Б) вбудовані і визначувані;
- В) адміністрування і сценарії;
- Г) розширення і доступ.

5. SWF-файли виконуються ....

- A) KMPlayer;
- Б) MediaPlayer;
- В) SWFPlayer;
- Г) Flash Player.

6. Мова ... – це інтерактивна мова програмування, сильними сторонами якої вважаються її великі можливості для роботи з текстом.

- A) Perl;
- Б) CPAN;
- В) XeSh;
- Г) Swinter.

7. Мова ... одна з найбільш популярних сценарних мов.

- A) HFD;
- Б) GHF;
- В) MJN;
- Г) PHP.

8. ... є мовою програмування з динамічною типізацією, що не вимагає вказівки типу при оголошенні змінних.

- A) NHJ;

Б) JHF;

В) PHP;

Г) BED.

9. Програма на мові Perl складається з :

- A) декларацій;
- Б) операторів;
- В) серверів;
- Г) правильна відповідь А і В.

10. Оператори мови Perl поділяються на:

- A) прості;
- Б) буквенні;
- В) складені;
- Г) правильна відповідь А і С.

11. ... можуть розміщуватися в окремому файлі або вбудовуються безпосередньо в HTML документ.

- A) PHP-сценарії;
- Б) веб-сервери;
- В) веб-технології;
- Г) сценарії.

12. У ... є широкий діапазон функцій для роботи з файлами.

- A) HGF;
- Б) PHP;
- В) VGH;
- Г) NHJK.

13. Для запису даних у файл можна використовувати функцію ...

- A) hyoik;
- Б) goirt;
- В) fputs;
- Г) buglo.

14. ... зовні нагадує багато в чому старішу технологію ASP.

- A) ASP.NET;
- Б) WORK.NET;
- В) ASP.COM;
- Г) ASP.RU.

15. Файли ASP.NET обробляються бібліотекою ....

- A) aspru\_isapi.dmm;
- Б) asp/dll;
- В) dfuasp.djj;
- Г) aspnet\_isapi.dll.

16. Простори імен скількох груп містить бібліотека базових класів .NET

- А) 2;
- Б) 8;
- В) 3;
- Г) 6.

17. Прийнято виділяти такі типи серверних елементів управління:

- А) звичайні теги HTML;
- Б) нові теги ASP.NET;
- В) серверні елементи управління для перевірки даних;
- Г) усі вище вказані відповіді правильні.

## РОЗДІЛ 3



## ОСНОВИ XML

### ТЕМА 3.1. МОВИ ОПИСУВАННЯ СХЕМ XML

#### 3.1.1. Вступ в XML

#### 3.1.2. Мови опису схем XML

#### 3.1.3. DTD схема

#### 3.1.4. XDR схема

#### 3.1.1. Вступ в XML

У 1986 році, задовго до того, як ідея створення мережі Веб-сервером була втілена в життя, універсальна стандартизована мова розмітки SGML (Standardized Generalized Markup Language) була затверджена як міжнародний стандарт (ISO 8879) визначення мов розмітки, хоча SGML існував ще з кінця шестидесятих. Він використовувався для того, щоб описувати мови розмітки, надаючи при цьому авторові можливість давати формальні визначення кожному елементу і атрибуту мови.

Мова HTML спочатку була всього лише одним з SGML-застосувань. Вона описувала правила, по яких має бути підготовлена інформація для World Wide Web. Таким чином, мова HTML – це набір розпоряджень SGML, сформульованих у вигляді визначення типу документа (DTD), пояснюючих, що саме значать теги і елементи. Схема DTD для мови HTML зберігається у веб-браузері.

До недоліків мови HTML можна віднести наступні:

- HTML має фіксований набір тегів. Не можна створювати свої теги, зрозумілі іншим користувачам;
- HTML – це виняткова технологія представлення даних. HTML не несе інформації про значення вмісту, що знаходиться в тегах;
- HTML – «плоска» мова. Значущість тегів в ній не визначена, тому з її допомогою не можна описати ієрархію даних;
- як платформи для застосувань використовуються браузери. HTML не володіє достатньою потужністю для створення веб-застосувань на тому рівні, до якого в даний час прагнуть веб-розробники. Наприклад, на мові HTML неможливо розробити застосування для професійної обробки і пошуку документів;
- великі об'єми трафіку мережі. Існуючі HTML-документи, використовувані як застосування, що перенавантажують Інтернет великими об'ємами трафіку.

ку в системах клієнт-сервер. Прикладом може служити пересилка по мережі великого за об'ємом документу, тоді як необхідна тільки невелика частина цього документа.

Таким чином, з одного боку, мова HTML є дуже зручним засобом розмітки документів для використання у веб-сервері, а з іншої – документ, розмічений в HTML, має мало інформації про свій вміст. Якщо той або інший документ несе досить повну інформацію про свій вміст, то з'являється можливість порівняно легко провести автоматичну узагальнену обробку і пошук у файлі, що зберігає документ. Мова SGML дозволяє зберігати інформацію про вміст документа, проте внаслідок особливої складності він ніколи не використовувався так широко, як HTML.

Група експертів по мові SGML, очолювана Джоном Боузеком (Jon Bosak) із компанії Sun Microsystems, приступила до роботи із створення підмножини мови SGML, яка могла б бути прийнята Web-суспільством. Вирішено було видалити багато несуттєвих можливостей SGML. Перебудовану таким чином мову назвали XML. Спрощений варіант виявився значно доступнішим, ніж оригінал, його специфікації займали всього 26 сторінок в порівнянні з більш ніж 500 сторінками специфікації SGML.

Розглянемо детальніше структуру і особливості цієї мови.

XML (eXtensible Markup Language) – рекомендована W3C мова розмітки. XML – текстовий формат, призначений для зберігання структурованих даних, для обміну інформацією між програмами, а також для створення на його основі спеціалізованих мов розмітки. XML є спрощеною підмножиною мови SGML.

Мова XML має наступні переваги:

- ⇒ *це людино-орієнтований формат документу, він зрозумілий як людині, так і комп'ютеру;*
- ⇒ *підтримує Юнікод;*
- ⇒ *у форматі XML можуть бути описані основні структури даних – такі як записи, списки і дерева;*
- ⇒ *це самодокументований формат, який описує структуру і імена полів також як і значення полів;*
- ⇒ *має строго певний синтаксис і вимоги до аналізу, що дозволяє йому залишатися простим, ефективним і несуперечливим;*
- ⇒ *широко використовується для зберігання і обробки документів;*
- ⇒ *це формат, заснований на міжнародних стандартах;*
- ⇒ *ієрархічна структура XML підходить для опису практично будь-яких типів документів;*
- ⇒ *є простим текстом, вільним від ліцензування і яких-небудь обмежень;*
- ⇒ *не залежить від платформи;*
- ⇒ *є підмножиною SGML, для якої накопичений великий досвід роботи і створені спеціалізовані застосування;*

До відомих недоліків мови можна віднести наступні:

- синтаксис XML є надмірний;
- розмір XML документу суттєво більше бінарного представлення тих же даних (порядку 10 разів);
- розмір XML документу суттєво більший, ніж документу в альтернативних текстових форматах передачі даних (наприклад JSON, YAML) і особливо у форматах даних, оптимізованих для конкретного випадку використання;
- надмірність XML може вплинути на ефективність застосування. Зростає вартість зберігання, обробки і передачі даних;
- для великої кількості задач не потрібна вся потужність синтаксису XML і можна використовувати значно простіші і продуктивніші рішення;
- простори імен XML складно використовувати і їх складно реалізовувати в XML парсерах;
- XML не містить вбудованої в мову підтримки типів даних. У ній немає понять «Цілих чисел», «рядків», «дат», «булевих значень» і так далі;
- ієрархічна модель даних, запропонована XML, обмежена в порівнянні з реляційною моделлю і об'єктно-орієнтованими графами.

XML можна розглядати не тільки як нову мову розмітки, але і як основу для цілого сімейства технологій.

По-суті, XML служить метамовою для опису структури інших мов. Взаємозв'язок між SGML, XML, HTML і деякими іншими мовами показана на рисунку 3.1.1.1.

Важливою різницею XML від HTML є та велика увага, яка приділяється контролю за тим, наскільки точно дотримуються правила мови при розмітці документів. Залежно від цього прийнято виділяти правильно побудовані і дійсні документи XML.

Документ XML вважається за правильно побудований, якщо він відповідає всім синтаксичним правилам XML.

Перевірка дійсності документу передбачає виконання наступних дій:

- ⇒ *перевірка використання тільки заданого набору дескрипторів;*
- ⇒ *перевірка повної відповідності порядку дотримання елементів і атрибутів вмісту документу або певним правилам;*
- ⇒ *контроль типів даних (досягається при використанні відповідної схеми);*
- ⇒ *контроль цілісності даних для забезпечення оптимального обміну інформацією через Веб-сервер за допомогою транзакцій.*

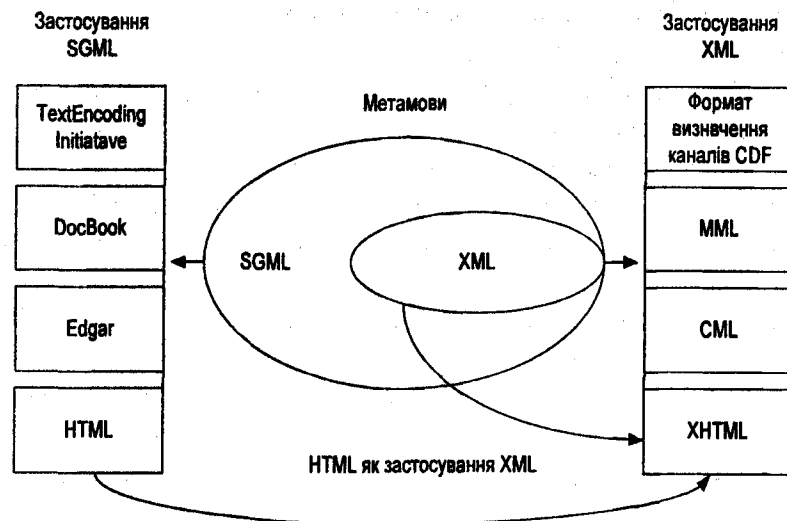


Рис. 3.1.1.1. Взаємозв'язок між SGML, XML, HTML і деякими іншими мовами

Розглянемо тепер основні синтаксичні правила побудови XML документів:

- XML-документ містить один і лише один кореневий елемент, що містить решту всіх елементів;
- дочірні елементи, що містяться в кореневому елементі, мають бути правильно вкладені;
- імена елементів підляглі правилам:
- ім'я починається з букви, знаку підкреслення або двокрапки;
- після першого символу в імені можуть бути букви, цифри, знаки переноси, підкреслення, крапка або двокрапка;
- імена не можуть починатися з буквосполучення XML.
- XML документ має наступну структуру:
- перший рядок XML документа називається оголошенням XML. Це необов'язковий рядок, що вказує версію стандарту XML. Також тут може бути вказано кодування символів і зовнішні залежності;
- коментар може бути розміщений в будь-якому місці дерева. XML коментарі розміщуються у середині пари тегів `<!--` і закінчуються `-->`. Два знаки дефіс (`--`) не можуть бути застосовані ні в якій частині в середині коментаря;
- решта частини цього XML-документа складається з вкладених елементів, деякі з яких мають атрибути і вміст;
- елемент зазвичай складається із відкриваючого і закриваючого тегів, що вміщують текст та інші елементи;
- відкриваючий тег складається з імені елементу в кутових дужках;
- закриваючий тег складається з того ж імені в кутових дужках, але перед ім'ям ще додається коса риска;

- вмістом елементу називається все, що розташоване між тим, що відкривається і закривається тегами, включаючи текст і інші (вкладені) елементи;
- крім вмісту біля елементу можуть бути атрибути – пари ім'я=значення, що додаються в середину відкриваючого тегу після назви елементу;
- значення атрибутів завжди вставляються в лапки (одинарні або подвійні), одне і те ж ім'я атрибуту не може зустрічатися двічі в одному елементі;
- не рекомендується використовувати різних типів лапок для значень атрибутів одного тегу;
- для позначення елементу без вмісту, що має назву порожній елемент, необхідно застосовувати особливу форму запису, яка складається з косої риски `</>`.

На жаль, описані вище правила дозволяють контролювати тільки формальну правильність XML-документу, але не змістовну. Для вирішення другого завдання використовуються так звані схеми.

Схема чітко визначає ім'я і структуру кореневого елементу, включаючи специфікацію всіх його дочірніх елементів. Програміст може задати, які елементи і в якій кількості обов'язкові, а які – необов'язкові. Схема також визначає, які елементи містять атрибути, допустимі значення цих атрибутів, в т.ч. значення за замовчуванням.

Найчастіше для опису схеми використовуються наступні специфікації:

- ⇒ DTD (Document Type Definition) – мова визначення типу документів;
- ⇒ XDR (XML Data Reduced) – діалект XML, розроблений Майкрософт;
- ⇒ XSD (мова визначення схем XML) – рекомендована консорціумом W3C.

XML-документ відрізняється від HTML-документу також і тим, що він відображається у веб-браузері. Без використання CSS або XSL XML-документ відображається як простий текст в більшості веб-браузерів. Деякі веб-браузери, такі як Internet Explorer, Mozilla і Firefox відображають структуру документа у вигляді дерева, дозволяючи згорнути і розгорнути вузли за допомогою натиснень клавіші миші.

Найбільш поширеними є три способи перетворення XML-документа в той вигляд, що відображається користувачу:

- ⇒ застосування стилів CSS;
- ⇒ застосування перетворення XSLT;
- ⇒ написання на якій-небудь мові програмування обробника XML-документу.

### 3.1.2. Мови опису схем XML

Ідея створення власних тегів, які мають спеціальне значення і, що допомагають описати вміст документу, сама по собі є необхідною в даній ситуації. Але якщо кожен користувач може створювати свої власні описи, яким чином їх розпізнавати? З цією метою в специфікації XML для опису подібних «самодіяльних» тегів використовуються схеми. Вони необхідні для того, щоб:

- описати, що саме являється розміткою;
- описати точно, що означає розмітка.

Найбільш відомими мовами опису схем є наступні:

- ⇒ *DTD (Document Type Definition)* – мова визначення типу документів, який спочатку використовувався в якості мови опису структури SGML-документу;
- ⇒ *XDR (XML Data Reduced)* – діалект схеми XML, розроблений Microsoft, який підтримувався в Internet Explorer 4 і 5 версій;
- ⇒ *XML Schema* або просто *XSD (мова визначення схем XML)* – рекомендація консорціуму W3C з 2001 року.

### 3.1.3. DTD схема

Схема DTD надає шаблон розмітки документу, в якому вказуються наявність, порядок дотримання і розміщення елементів і їх атрибутів в документі XML.

В рамках DTD модель вмісту XML-документа можна описати таким чином:

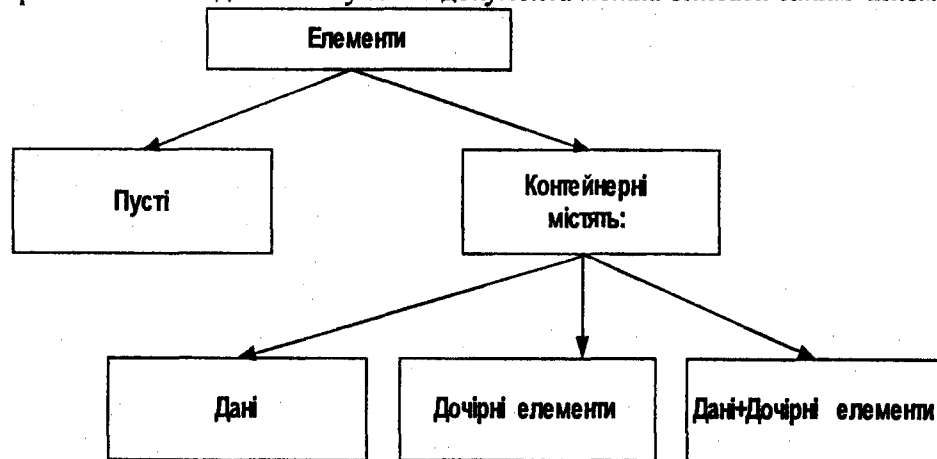


Рис. 3.1.3.1 Модель вмісту XML-документа в рамках DTD

Кожен елемент документу може мати один з типів:

Вміст	Синтаксис	Коментар
Дані	<!ELEMENT ім'я (#PCDATA)>	Містить тільки текстові дані
Інші елементи	<!ELEMENT ім'я (дочірній елемент 1, дочірній елемент 2)>	Містить тільки дочірні елементи
Змішане	<!ELEMENT ім'я (#PCDATA, дочірній елемент)*>	Містить комбінацію текстових даних і дочірніх елементів
EMPTY	<!ELEMENT ім'я EMPTY>	Нічого не містить
ANY	<!ELEMENT ім'я ANY>	Може містити текстові дані або дочірні елементи

Атрибути, що знаходяться в середині тегів документу, описуються окремо за допомогою синтаксису:

<!ATTLIST

ім'я\_елементу ім'я\_атрибуту1 (тип) значення\_по\_замовчуванню

.....

ім'я\_елементу ім'я\_атрибутуN (тип) значення\_по\_замовчуванню >

При цьому атрибут в DTD може мати один з трьох типів:

- рядок;
- маркований атрибут;
- атрибут з перерахуванням;

Крім типу атрибуту можна також задавати і його модальність:

Значення	Опис
#REQUIRED	Атрибут обов'язково має бути вказаний
#FIXED	Значення атрибуту не повинне відрізнятися від вказаного
#IMPLIED	Необов'язкове значення

Розглянемо, як приклад опис атрибутів рядкового типу для елемента, що описує деяке повідомлення:

<!ATTLIST message

|        |       |           |
|--------|-------|-----------|
| number | CDATA | #REQUIRED |
| date   | CDATA | #REQUIRED |
| from   | CDATA | #FIXED    |
| status | CDATA | #IMPLIED> |

Якщо цей елемент містить атрибути з перерахуванням, то їх опис може виглядати, наприклад, таким чином:

<!ATTLIST message

|        |                         |           |
|--------|-------------------------|-----------|
| number | ID                      | #REQUIRED |
| from   | CDATA                   | #REQUIRED |
| alert  | (low   normal   urgent) | "normal"> |

Марковані атрибути елемента можуть бути чотирьох типів:

Значення	Опис
ID	Унікальний ідентифікатор елемента (починається з букви, двокрапки або підкреслення)
IDREF	Посилання на елемент, що містить атрибути ID
ENTITIES	Посилання на зовнішній елемент
NMTOKEN	Містить букви, цифри, крапки, знаки підкреслення, перенесення, двокрапки, але не пропуски

В DTD можна використовувати наступні індикатори входження послідовностей:

Символ	Приклад	Опис
,	(a, b, c)	Послідовне використання елементів списку
	(a   b   c)	Використовується один з членів списку
	date	Використовується один і лише один елемент
?	subject?	Необов'язкове використання (0 або 1 раз)
+	paragraph+	Використовується один або кілька разів
*	brother*	Використовується нуль або кілька разів

Як приклад приведемо DTD схему, що описує структуру електронної поштової скриньки:

<!ELEMENT mailbox (message\*)>

<!ELEMENT message (head, body)>

```

<!ATTLIST message uid CDATA #REQUIRED>
<!ELEMENT head ( from,to+, subject?, CC*, notify?) >
<!ELEMENT from (#PCDATA)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT subject (#PCDATA)>
<!ELEMENT CC (#PCDATA)>
<!ELEMENT notify EMPTY>
<!ELEMENT body (#PCDATA)>

```

Початковий документ XML, що задовольняє даній схемі, може виглядати, наприклад, так:

```

<?xml version="1.0" ?>
<!DOCTYPE mailbox SYSTEM «mailbox.dtd»>
<mailbox>
  <message uid="1">
    <head>
      <from>user1@myhp.edu</from>
      <to>user2@myhp.edu</to>
      <subject>Re:</subject>
    </head>
    <body>
      What's up!
    </body>
  </message>
  <message uid="2">
    <head>
      <from>user3@myhp.edu</from>
      <to>user2@myhp.edu</to>
      <subject>Remind</subject>
      <CC> user1@myhp.edu </CC>
      <notify/>
    </head>
    <body>
      Remind me about meeting.
    </body>
  </message>
</mailbox>

```

Зверніть увагу на 2-гий рядок документу, в якому вказується зовнішнє посилання на файл DTD, що містить, схему.

DTD допускає два способи використання в XML документі:

➤ оголошення внутрішньої схеми:

```

<!DOCTYPE кореневий_елемент [
<!ELEMENT кореневий_елемент (модель вмісту)>

```

]>

➤ оголошення зовнішньої схеми:

```

<!DOCTYPE кореневий_елемент SYSTEM «name.DTD»>

```

На закінчення звернемо увагу на наступні недоліки DTD схем:

- ⇒ не є екземплярами XML. Потрібне вивчення абсолютно іншої мови;
- ⇒ не надають контроль за типами даних, за винятком найпростіших текстових даних;
- ⇒ не є екземплярами XML, тому їх не можна легко розширити або перетворити до інших мов розмітки – HTML або DHTML;
- ⇒ не забезпечують підтримки просторів імен XML.

### 3.1.4. XDR схема

XML-Data – повне ім'я мови опису схем, запропонованої Майкрософтом, аXML-DataReduced – це «частина» повної рекомендації. Схема XDR – це екземпляр XML, тобто відповідає всім синтаксичним правилам і стандартам XML.

Реалізуючи перевірки даних на рівні документа за допомогою схеми, застосування, генеруючі і приймаючі транзакції можна оптимізувати для забезпечення максимальної швидкодії. Відповідність полів і правильність записів перевіряються на рівні екземплярів XML.

Кореневим елементом в схемі XDR завжди є елемент Schema:

```

<Schema
  name="ім'я_схеми" xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <-- Оголошення інших елементів -->
</Schema>

```

Елемент ElementType має синтаксис:

```

<ElementType
  content="{empty texOnly eltOnly mixed}">
  dt:type «datatype»
  model="{open closed}"
  name = «idref»
  order="{one seq many}"
>

```

Елемент ElementType може мати наступні атрибути:

Ім'я атрибуту	Опис
1	2
name	Ім'я елементу
content	Вміст елементу. Допустимі значення: empty (порожній елемент), eltOnly (може бути тільки контейнером для інших елементів), textOnly (тільки текстові дані), mixed (змішані дані).



Закінчення

1	2
dt:type	Тип даних елементу
model	Може набувати значень:
	Open – дозволено використовувати елементи, не визначені в схемі
	Closed – заборонено використовувати елементи, не визначені в схемі
order	Порядок дотримання дочірніх елементів в екземплярі XML. Допустимі значення:
	one – передбачається наявність одного документа
	many – будь-яка кількість елементів у будь-якому порядку
	seq – елементи вказуються в строго заданому порядку.

Як дочірні елементи для ElementType можна використовувати наступні:

Ім'я елементу	Опис
element	Оголошує дочірній елемент
description	Забезпечує опис елементу ElementType
datatype	Забезпечує тип даних елементу ElementType
group	Визначає порядок дотримання елементів
AttributeType	Визначає атрибут
attribute	Визначає відомості про дочірній елемент AttributeType

Для оголошення атрибутів використовується синтаксис:

```
<AttributeType
  default=>default-value>
  dt:type=>primitive-type>
  dt:values=>enumerated-values>
  name=>idref>
  required=>{yes|no}>
```

>

У свою чергу елемент AttributeType може мати атрибути:

Значення	Опис
default	Значення за замовчуванням
dt:type	Один із наступних типів: entity, entities, enumeration, id, idref, nmtoken, nmtokens, notation, string
dt:values	Допустимі значення
name	Ім'я атрибуту
required	Вказує на обов'язкову наявність атрибуту в описі

Синтаксис для опису елементу attribute виглядає таким чином:

```
<attribute
  default=>default-value>
  type=>attribute-type>
  [required=>{yes|no}]>
```

>

та його можливі значення можуть бути такими:

Значення	Коментар
default	Значення за замовчуванням
type	Ім'я елементу AttributeType, визначеного в даній схемі повинно відповідати атрибуту name елементу AttributeType
required	Вказує на обов'язкову наявність атрибуту в описі

На відміну від DTD-схем XDR підтримує типи даних. Елемент Schema має наступний атрибут:

**Xmlns:dt=>urn=schemas-microsoft-com:datatypes>**

З повним списком типів даних можна ознайомитися на сторінці за адресою: [http://msdn.microsoft.com/en-us/library/ms256121\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms256121(VS.85).aspx)

Індикатори входження в схемах XDR мають синтаксис:

```
<element
  type=>element-type>
  [minOccur=>{0|1}]>
  [maxOccur=>{1|*}]>
>
```

На закінчення наведемо приклад XSD схеми, що описує структуру XML документа, що містить листи електронної пошти:

```
<?xml version = «1.0»?>
<xsd:schema xmlns:xsd=>http://www.w3.org/2010/10/XMLSchema>

  <xsd:element name=>m_box>>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref=>message> minOccurs=>0>
          maxOccurs=>unbounded/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name=>message>>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref=>head> minOccurs=>1> maxOccurs=>1/>
        <xsd:element ref=>body> minOccurs=>1> maxOccurs=>1/>
      </xsd:sequence>
      <xsd:attribute name=>uid> use=>required> type=>xsd:string/>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name=>head>>
    <xsd:complexType>
```

```

<xsd:sequence>
  <xsd:element ref="to" minOccurs="1" maxOccurs="unbounded"/>
  <xsd:element ref="from" minOccurs="1" maxOccurs="1"/>
  <xsd:element ref="date" minOccurs="1" maxOccurs="1"/>
  <xsd:element ref="subject" minOccurs="1" maxOccurs="1"/>
  <xsd:element ref="cc" minOccurs="0" maxOccurs="unbounded"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:element name="to" type="xsd:string"/>
<xsd:element name="from" type="xsd:string"/>
<xsd:element name="date" type="xsd:string"/>
<xsd:element name="subject" type="xsd:string"/>
<xsd:element name="cc" type="xsd:string"/>

</xsd:schema>

```

Для перевірки дійсності XML документу можна використовувати спеціальні валідатори, наприклад, W3C валідатор (<http://validator.w3.org/>).

Для перевірки схем також існують спеціальні валідатори, наприклад, XML Schema валідатор (<http://www.w3.org/2011/03/webdata/xsv>).

Згідно специфікації W3C XML програма повинна припинити обробку XML-документу, як тільки буде виявлена помилка в цьому документі.

## ТЕМА 3.2. DOM XML. ПЕРЕТВОРЕННЯ XML ДОКУМЕНТІВ

### 3.2.1. Передумови перетворення XML документів

#### 3.2.2. XSLT і XPath

#### 3.2.3. XSL-FO

#### 3.2.4. XQuery

### 3.2.1. Передумови перетворення XML документів

Для програмної обробки XML документів використовується модель XML DOM, яка визначає об'єкти і властивості всіх XML-елементів і методи (інтерфейс) для доступу до них. Інакше кажучи, XML DOM описує, яким чином необхідно отримувати, змінювати, додавати і видаляти XML елементи.

Згідно DOM:

- все, що міститься в середині XML-документу, є вузлом;
- весь документ представляється вузлом документу;
- кожен XML-елемент – вузол елемента;
- текст у середині XML-елементів – текстовий вузол;
- кожен атрибут – вузол атрибуту;
- коментарі – вузли коментарів.

XML-документ відповідно до моделі XML DOM подається як дерево з вузлів, при цьому:

- ⇒ всі вузли дерева знаходяться в певних зв'язках один з одним;
- ⇒ всі вузли доступні через дерево. Їх вміст може бути змінено, видалено; нові елементи можуть бути додані в дерево;
- ⇒ дерево починається з кореневого вузла і розгалужується вниз аж до текстових вузлів на самому нижньому рівні дерева;
- ⇒ всі вузли знаходяться в ієрархічних зв'язках між собою;
- ⇒ ці зв'язки описуються за допомогою понять батько, дочірній і нащадок (всі дочірні на одному рівні).

Альтернативним інтерфейсом для обробки XML-документів є SAX.

SAX (Simple API for XML) – прикладний програмний інтерфейс для парсера з послідовним доступом до XML. Цей інтерфейс надає механізм читання даних з XML-документу.

SAX-парсер є потоковим і керованим подіями. Завданням користувача SAX API полягає в описі методів, що викликаються подіями, які виникають при аналізі документу.

Такими подіями можуть бути наступні:

- текстовий вузол;
- вузол елемента XML;
- інструкція обробки XML;
- коментар XML.

Події викликаються появою як відкриваючого тегу, так і закриваючого тегу будь-якого з цих елементів документа. Атрибут XML також розглядається як подія.

Аналіз документа є однонаправленим (тобто без повернень по дереву).

На відміну від DOM формальної специфікації для SAX не існує. Як нормативна розглядається Java реалізація SAX.

Слід зазначити такі переваги і недоліки SAX.

Переваги SAX:

- ⇒ *витрати пам'яті істотно менші (залежить від максимальної глибини дерева документа і кількості атрибутів у вузлі елементу), ніж в разі DOM (потрібно зберігати в пам'яті все дерево документа);*
- ⇒ *швидкість роботи вище за рахунок скорочення витрат часу на виділення пам'яті для елементів дерева в разі DOM;*
- ⇒ *потокове читання даних з диска в разі DOM неможливе. Якщо для розміщення всього документа в пам'яті недостатньо місця, то використання SAX є безальтернативним.*

Недоліки:

- ⇒ *процедура перевірки правильності передбачає доступ до всього документа одночасно;*
- ⇒ *це також потрібно і в разі XSLT перетворення.*

Якщо завантажити «чистий» документ XML у веб-браузер, то можна буде побачити деревовидну структуру цього документа. У цьому якраз і полягає головна відмінність між XML і HTML, а саме поділ структури документа і його представлення у браузері. Конкретний вид XML-документу описується окремо за допомогою CSS або XSL.

CSS і XSL – принципово різні технології, що мають лише часткове припинення сфер застосування. CSS-форматування застосовується до HTML-документу браузером на клієнтській стороні, а XSL-перетворення виконується, як правило, на сервері, після чого результат відправляється браузеру клієнта. XSL базується на XML, завдяки чому XSL гнучкіший і універсальний. У розробників є можливість використовувати засоби контролю за коректністю складання стилєвих списків (використовуючи схеми XML).

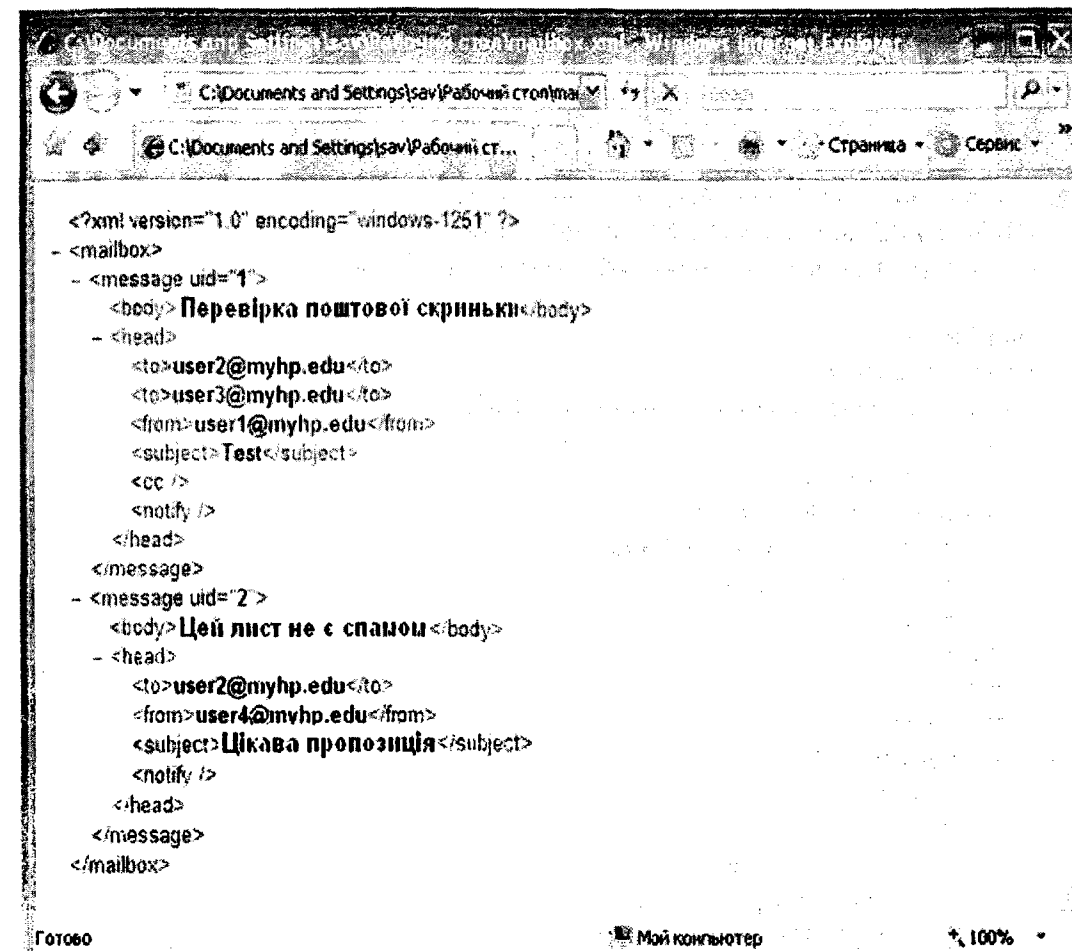


Рис. 3.2.1.1. Деревовидна структура XML-документу у веб-браузері

За допомогою XSL можна перетворити XML-документ у формат HTML, WML, RTE, PDF, SQL, SWF, а також в інший XML і XSL документ. XSL вказує, як буде оформлений документ, де і яким чином повинні розмішуватися дані.

Специфікація XSL складається з трьох частин:

- XSLT (XSL Transformations), мова для перетворення XML;
- XPath – мова шляхів і виразів, що використовується в XSLT для доступу до окремих частин XML-документу;
- XSL-FO (XSL Formatting Objects) – мова для верстки XML.

Найбільш поширеним механізмом XSLT перетворень для систем, що працюють на платформі Microsoft Windows є MSXML, а для систем на основі GNU – xsltproc.

Для того, щоб обробити XML-документ за допомогою XSL, необхідно в XML-документі написати наступну інструкцію:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="my-style.xsl"?>
<root>
```

```
<!-- ... -->
</root>
```

Повертаючись до прикладу, що розглядався раніше, додавши в XML-файл посилання на XSL-файл, отримаємо наступний код розмітки:

```
<?xml version="1.0" encoding="windows-1251" ?>
<!DOCTYPE mailbox SYSTEM «mailbox.dtd»>
<?xml-stylesheet href="mailbox.xsl" type="text/xsl" ?>
```

```
<mailbox>
<message uid='1'>
<body>Перевірка поштового ящика</body>
<head>
<to>user2@myhp.edu</to>
<to>user3@myhp.edu</to>
<from>user1@myhp.edu</from>
<subject>Test</subject>
<cc></cc>
<notify></notify>
</head>
</message>
<message uid='2'>
<body>Цей лист не є спамом</body>
<head>
<to>user2@myhp.edu</to>
<from>user4@myhp.edu</from>
<subject>Цікава пропозиція</subject>
<notify></notify>
</head>
</message>
</mailbox>
```

Після завантаження даного документу у веб-браузері його вигляд кардинально зміниться:

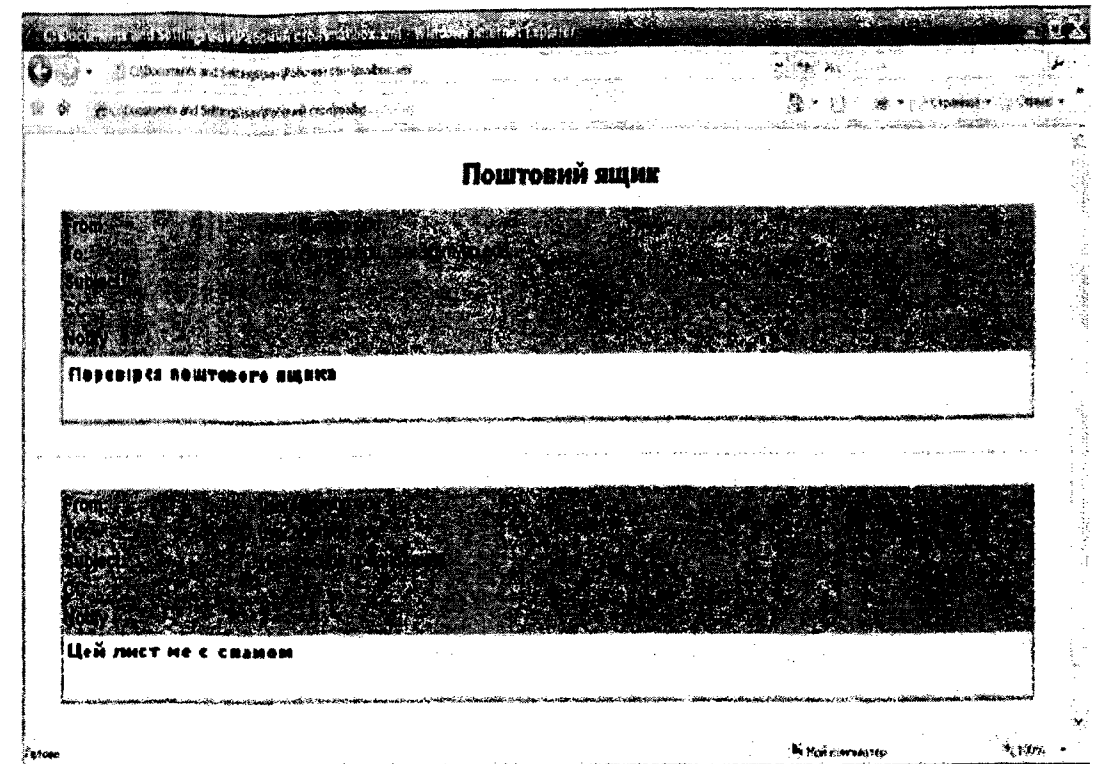


Рис. 3.2.1.2. Документ після завантаження у веб-браузер

Вміст XSL файлу mailbox.xsl приводиться нижче:

```
<?xml version="1.0" encoding="windows-1251" ?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/2012/XSL/Transform">

<xsl:template match = «mailbox»>
<h2 align="center" color="red">Поштовий ящик</h2>
<xsl:apply-templates/>
</xsl:template>

<xsl:template match = «message»>
<table align="center" border="0" width="80%" bgcolor="gray">
<tr>
<td width="20%"><b>From:</b></td>
<td>
<xsl:value-of select="head/from">
</td>
</tr>
<tr>
<td width="20%"><b>To:</b></td>
```

```

<td> <xsl:for-each select=»head/to»>
  <xsl:apply-templates/>
</td>
</tr>
<tr>

  <td width=»20%»><b>Subject:</b></td>
  <td><xsl:value-of select=»head/subject»/></td>
</tr>
<tr>

  <td width=»20%»><b>CC</b></td>
  <td><xsl:value-of select=»head/cc»/></td>
</tr>
<tr>

  <td width=»20%»><b>Notify</b></td>
  <td><xsl:value-of select=»head/notify»/></td>
</tr>
<tr>

  <td colspan=»2»>
    <textarea cols=»120%» rows=»3»>
      <xsl:value-of select=»body»/>
    </textarea>
  </td>
</tr>
</table>
<br/><hr/><br/>
</xsl:template>

</xsl:stylesheet>

```

### 3.2.2. XSLT та XPath

Специфікація XSLT є рекомендацією W3C.

В результаті застосування таблиці стилів XSLT, що складається з набору шаблонів, з XML-документу (початкове дерево) утворюється кінцеве дерево, яке може бути іншою XML-структурою, HTML-документом або звичайним текстом. Правила вибору даних з початкового дерева записуються на мові запитів XPath. XSLT застосовується в основному у веб-програмуванні і для генерації звітів.

Завдяки XSLT реалізується відділення даних від їх представлення в рамках парадигми MVC (Model-view-controller).

XPath (XML Path Language) – мова запитів до елементів XML-документу. XPath був розроблений для організації доступу до частин документа XML у файлах трансформації XSLT і є стандартом консорціуму W3C. У мові XPath використовується компактний синтаксис, відмінний від прийнятого в XML. Починаючи з версії 2.0, XPath є складовою частиною мови XQuery. XPath допомагає обходити всілякі дерева, отримувати необхідні елементи з іншої гілки щодо точки обходу, розпізнавати предків, нащадків, атрибути елементів. Це повноцінна мова навігації по дереву.

Для знаходження елементів у дереві документа використовуються шляхи адресації.

Кожен крок адресації складається з трьох частин:

- осі, наприклад, child::;
- умови перевірки вузлів, наприклад, імена елементів документа body, html;
- предикату, наприклад, attribute::class.

Доповненням до ядра мови є набір функцій, які діляться на 5 груп: системні функції, функції з множинами, рядкові функції, логічні функції, числові функції.

### 3.2.3. XSL-FO

XSL-FO (eXtensible Markup Language Formatting Objects) – рекомендована W3C мова розмітки переддрукарських матеріалів. По-суті, XSL-FO – це уніфікована мова застосувань. Вона не має семантичної розмітки (як в HTML) і зберігає всі дані документа у середині себе (на відміну від CSS, який модифікує застосування за замовчуванням для зовнішнього HTML або XML-документу).

В результаті застосування XSLT-перетворення до початкового документа XML виходить його опис на мові XSL-FO. FO-процесор конвертує XSL-FO-документ в який-небудь читаний або друкований формат. Найчастіше використовується перетворення в PDF або PS; деякі FO-процесори можуть давати на виході RTF-файли або просто показувати документ у вікні.

### 3.2.4. XQuery

XQuery – мова запитів, розроблена для обробки даних у форматі XML.

Консорціум World Wide Web Consortium (W3C) утворив робочу групу для мови запитів до джерел даних, представлених на мові XML. Ця мова запитів, що отримала назву XQuery, розвивається до цих пір. XQuery – функціональна мова, що складається з декількох видів виразів, які можуть використовуватися в різних поєднаннях. Мова базується на системі типів XML Schema і сумісна з іншими стандартами, пов'язаними з XML.

Мова XML [17] все частіше застосовується як формат для обміну інформацією між різними застосуваннями в Internet. Популярність XML багато в чому пояснюється його гнучкістю при представленні різних видів інформації. Застосування тегів робить XML-дані самоописуваними, а розширювана природа XML дозволяє визначати новий вигляд спеціалізованих документів. У міру зростання значущості XML створюється ціла серія стандартів, багато хто з яких був підготовлений консорціумом W3C [24]. Так, XML Schema забезпечує нотацію для визначення нових типів елемен-

тив і документів; XML Path Language (XPath) [24] – нотацію для вибору елементів в документі XML; Extensible Stylesheet Language Transformations (XSLT) [25] – нотацію для перетворення документів XML з одного представлення в інше.

XML дозволяє застосуванням обмінюватися даними в стандартному форматі, не залежному від способу їх зберігання. Скажімо, одне застосування може використовувати природний для XML формат зберігання, а інше – зберігати дані в реляційній базі даних. Оскільки XML все більше затверджується в ролі стандарту для обміну даними, природно, що запити, які поступають від застосувань, мають бути виражені як запити до даних у форматі XML. Це викликає потребу в мові запитів, явно орієнтованій на джерела XML-даних. У жовтні 1999 року W3C утворив робочу групу XML Query Working Group з метою розробки такої мови запитів, що отримала назву XQuery.

У XML-документах є внутрішній порядок, а реляційні дані не врегульовані, якщо не брати до уваги ті випадки, коли порядок можна визначити на основі значень даних. Реляційні дані зазвичай є «щільними» (тобто майже в кожному стовпці є значення), а відсутня інформація в реляційних системах часто представляється спеціальним значенням null. XML-дані часто бувають «розрідженими», а відсутність інформації може представлятися відсутністю елементу. По цих і іншим причинам наявні мови реляційних запитів не підходять безпосередньо для запитів XML-даних.

Розробка XQuery все ще продовжується. XML Query Working Group опублікувала попередні робочі версії декількох документів, що описують існуючий стан розробки. Можливо, найбільш важливим з них є документ XQuery: An XMLQuery Language [27], що містить синтаксис і неформальний опис мови. Крім того, робоча група опублікувала список вимог [28], опис моделі даних, покладеної в основу мови, формальний опис семантики, список функцій і операторів, а також приклади, що ілюструють застосування цієї мови [12]. Кожен з цих документів оновлюється у міру подальшого розвитку XQuery.

Мета XML Query Working Group – визначення двох видів синтаксису XQuery: один з них виражається на XML, а інший оптимізований для сприйняття людиною. Розглянемо спочатку другий варіант XQuery.

У початковому вигляді XQuery спрямований тільки на видобування інформації і не включає засобів для модифікації існуючих документів XML. Можливо, XML Query Working Group займеться додаванням засобів модифікації після завершення роботи над першою версією XQuery.

Формально вхідні і вихідні дані XQuery визначаються в термінах моделі даних. «Запитальна» модель даних забезпечує абстрактне представлення одного або декількох документів або фрагментів XML-документів. Модель даних спирається на поняття послідовності. Послідовність (sequence) – це впорядкований набір нульового або більшого числа об'єктів. Об'єкт (item) може бути вузлом або атомарним значенням. Атомарне значення (atomic value) – екземпляр одного з вбудованих типів даних, визначених в XML Schema, таких як рядки, цілі і десяткові числа, дати. Вузол (node) відповідає одному з семи видів: елементи, атрибути, тексти, документи, коментарі, команди обробки і простори імен. Вузол може мати інші вузли як нащадків, що дозволяє утворювати одну або декілька ієрархій вузлів. Деякі види вузлів, такі як елементи і атрибути, що мають імена або значення, які типізуються, або і те, і інше. Значення

(typed value), що типізується – це послідовність з нуля або більшого числа атомарних значень. Вузли індивідуальні (тобто два вузли можна розрізнити, навіть якщо вони мають однакові імена і значення), але атомарні величини такою індивідуальністю не володіють. Для всіх вузлів ієрархії є порядок, що зветься порядком документа (document order), відповідно до якого кожен вузол передує своєму порядку. Порядок документа відповідає порядку, в якому слідували б вузли, якби ієрархія вузлів представлялася у форматі XML. Порядок документа між вузлами в різних ієрархіях визначається в реалізації, але він має бути послідовним, тобто всі вузли однієї ієрархії повинні розташовуватися або до, або після всіх вузлів іншої ієрархії.

Послідовності можуть бути неоднорідними, тобто можуть містити зв'язок вузлів і атомарних значень різного типу. Проте послідовність ніколи не може бути об'єктом в іншій послідовності. Всі операції, що створюють послідовність, визначені так, що результат операції – однорівнева послідовність. Не проводиться відмінність між об'єктом і послідовністю одиначної довжини, тобто вузол і атомарне значення величини вважаються ідентичними послідовностями одиначної довжини, що містить цей вузол або атомарне значення.

Допускаються послідовності нульової довжини, і інколи вони використовуються для представлення відсутньої або невідомої інформації, багато в чому так само, як в реляційних системах використовуються невизначені значення.

Окрім послідовностей в запитальній моделі даних визначається спеціальне значення, яке називається значенням помилки (error value), яке є результатом обчислення виразу, що містить помилку. Значення помилки не може бути присутнім в послідовності разом з яким-небудь іншим значенням.

Вхідні XML-документи можуть бути перетворені в запитальну модель даних за допомогою процесу, який називається перевіркою коректності по схемі (schema validation). Цей процес виконує граматичний розбір документа, перевіряє його коректність відповідно до деякої схеми і представляє документ у вигляді ієрархії вузлів і атомарних значень. Якщо вхідний документ не має схеми, перевірка його коректності виконується відповідно до використовуваної за замовчуванням рекомендаційної схеми, яка присвоює родові типи – вузли маркуються як anyType, а атомарні величини – як anySimpleType.

Результат запиту може бути перетворений із запитальної моделі даних запитів в XML-представлення за допомогою процесу, що має назву серіалізації (serialization). Слід зазначити, що результат запиту не завжди є правильно побудованим XML-документом. Наприклад, запит може повертати атомарне значення або послідовність елементів, що не мають спільного предка.

Дані для прикладів. Щоб проілюструвати запитальну модель даних і забезпечити основу для подальших прикладів, розглянемо невелику базу даних, що містить дані інтерактивного аукціону і засновану на Use Case R [22]. Ця база даних містить два XML-документа з іменами items.xml і bids.xml.

Документ items.xml містить кореневий елемент з ім'ям items, який, у свою чергу, містить елемент item для кожного з товарів, запропонованих на продаж на аукціоні. Кожен елемент item має атрибут status і піделементи з іменами itemno, seller,

description, reserve-price і end-date. Елемент reserve-price вказує мінімальну продажну ціну, встановлену власником товару, а end-date визначає дату закінчення торгів.

Документ bids.xml містить кореневий елемент з ім'ям bids, який, у свою чергу, містить елемент bid для кожної ставки, яка пропонується за товар. Кожен елемент bid має піделементи з іменами itemno, bidder, bid-amount і bid-date.

На рисунках 3.2.4.1. і 3.2.4.2. показані модельні представлення документів items.xml і bids.xml відповідно. Круги, помічені буквами D, E, A і T, позначають вузли документів, елементів, атрибутів і тестів відповідно.

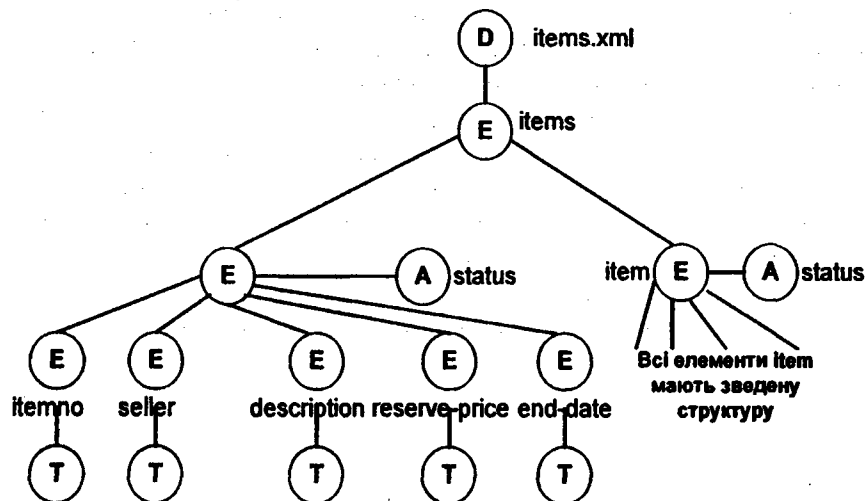


Рис. 3.2.4.1. Представлення моделі даних з items.xml

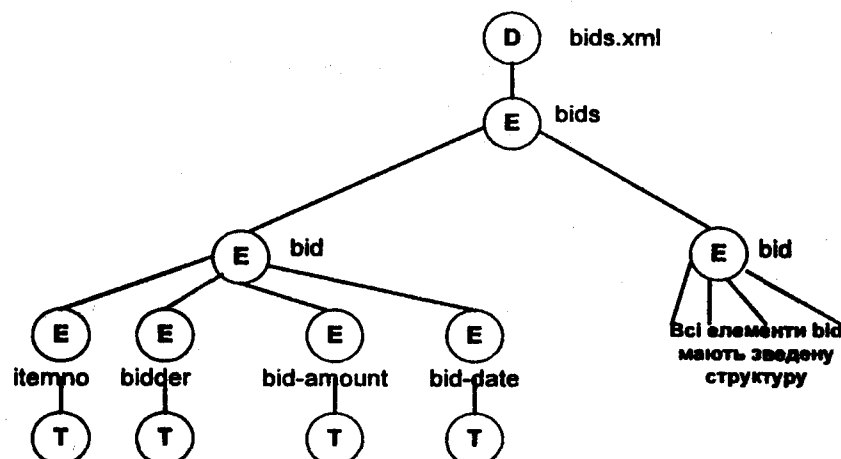


Рис. 3.2.4.2. Представлення моделі даних з bids.xml

XQuery визначається в термінах моделі даних, заснованої на неоднорідних послідовностях вузлів і атомарних значеннях. Екземпляр цієї моделі даних може містити один або декілька документів або фрагментів документів XML. Запит забезпечує відображення одного екземпляра моделі даних на інший екземпляр. Запит складається

з прологу, який встановлює середовище обробки, і виразу, яке генерує результат запити.

В даний час XQuery визначений на рівні попередніх робочих документів; створенням мови продовжує займатися W3C XML Query Working Group. Ця робоча група активно обговорює систему типів XQuery і питання про взаємне відображення цієї системи типів і системи типів XML Schema. Також обговорюються функції повнотекстового пошуку, серіалізація результатів запиту, обробка помилок і низка інших питань. Швидше за все, остаточна специфікація XQuery включатиме декілька рівнів відповідності; наприклад, в ній може бути визначено, як слід проводити статичну перевірку типів, але не вимагатиметься, щоб вона виконувалася в кожній реалізації, відповідній специфікації. Також очікується, що підмножина XQuery буде оголошена як XPath і стане можливим вбудовування цієї підмножини в інші мови, такі як XSLT.

Повніший опис XQuery і його граматику у формі Бекуса-науера можна знайти в [27]. Мова продовжує розвиватися, тому специфікація XQuery може змінитися.

Подібно до того, як XML застосовується як універсальний формат обміну інформацією в Мережі, XQuery покликаний служити як універсальний формат обміну запитом. Якщо XQuery отримає визнання як стандартний засіб витягання інформації з джерел XML-даних, це допоможе реалізувати потенціал XML.



## ТЕМА 3.3. ПРОГРАМНА ОБРОБКА XML ДОКУМЕНТІВ ЗА ДОПОМОГОЮ XML DOM

- 3.3.1. Структурний аналіз (парсинг) XML
- 3.3.2. Програмний інтерфейс XML DOM
- 3.3.3. Переміщення між вузлами дерева
- 3.3.4. Ігнорування порожніх текстових вузлів
- 3.3.5. Зміна значення атрибуту
- 3.3.6. Властивості об'єкту Node

XML DOM визначає об'єкти і властивості всіх XML елементів і методи (інтерфейс) для доступу до них. Інакше кажучи, XML DOM описує, яким чином необхідно отримувати, змінювати, додавати і видаляти XML-елементи.

Відповідно до моделі DOM все, що міститься в середині XML-документа – є вузлом. Тобто XML-документ подається у вигляді дерева вузлів, якими є елементи, атрибути і текст.

Оскільки структури HTML і XML-документів дуже схожі, а HTML DOM і XML DOM є частинами більш загального стандарту DOM, то і багато аспектів HTML DOM легко переносяться в XML DOM. Тому основна увага буде приділена саме специфічним особливостям саме XML DOM.

### 3.3.1. Структурний аналіз (парсинг) XML

Всі сучасні браузерери мають вбудовані аналізатори (парсери) XML для читання і обробки XML. Аналізатор прочитує XML-документ, розміщує його в пам'яті і перетворює в XML DOM об'єкт, доступний для мов програмування. Всі приклади тут наведені на JavaScript.

Є деякі відмінності між аналізаторами в Microsoft і в інших браузерах. Перший підтримує як завантаження XML-файлів, так і текстових рядків XML, що містять код, тоді як в інших браузерах використовуються роздільні аналізатори. При цьому всі аналізатори мають функції для переміщення по дереву XML-документа, доступу, вставки і видалення вузлів в дереві.

Розглянемо приклад завантаження XML об'єктів (файлів і рядків) за допомогою XML аналізатора Microsoft.

```
xmlDoc=new ActiveXObject(«Microsoft.XMLDOM»);
xmlDoc.async=>false»;
xmlDoc.load(«timetable.xml»);
```

У першому рядку програми створюється порожній об'єкт XML-документу Microsoft. Далі для запобігання роботі сценарію до повного завантаження документа прапорець асинхронності встановлюється в «false». У третьому рядку міститься інструкція завантажити XML-файл «timetable.xml».

У наступному прикладі відбувається завантаження рядка з XML кодом для подальшого аналізу.

```
xmlDoc=new ActiveXObject(«Microsoft.XMLDOM»);
```

```
xmlDoc.async=>false»;
xmlDoc.loadXML(txt);
```

Слід звернути увагу на різницю між методами load() і loadXML() по їх призначенню.

*Зауваження.* Сучасні браузерери не допускають міждоменні звернення до файлів з міркувань безпеки, тобто сама веб-сторінка (з програмним кодом) і XML файл повинні фізично знаходитися на одному сервері. Інакше браузер видасть повідомлення про помилку доступу.

Нижче приведені також кросплатформенні реалізації завантаження XML-файлу і XML-рядка відповідно.

Приклад 1. (html.txt)

```
<html>
<body>
<script type=>text/javascript>
try //Internet Explorer
{
xmlDoc=new ActiveXObject(«Microsoft.XMLDOM»);
}
catch(e)
{
try //Firefox, Mozilla, Opera, etc.
{
xmlDoc=document.implementation.createDocument(«»,null);
}
catch(e) {alert(e.message)}
}
try
{
xmlDoc.async=false;
xmlDoc.load(«timetable.xml»);
document.write(«xmlDoc is loaded, ready for use»);
}
catch(e) {alert(e.message)}
</script>
</body>
</html>
```

```
<html>
<body>
<script type=>text/javascript>
text=><timetable>»;
text=text+><lesson>»;
text=text+><timeFrom>08.00</timeFrom>»;
```

```

text=text+»<subject>Deutsch</subject>»»;
text=text+»<teacher>Borisova</teacher>»»;
text=text+»</lesson>»»;
text=text+»</timetable>»»;

```

```

try //Internet Explorer

```

```

{
    xmlDoc=new ActiveXObject(«Microsoft.XMLDOM»);
    xmlDoc.async=»false»;
    xmlDoc.loadXML(text);
}
catch(e)
{
    try //Firefox, Mozilla, Opera, etc.
    {
        parser=new DOMParser();
        xmlDoc=parser.parseFromString(text,»text/xml»);
    }
    catch(e) {alert(e.message)}
}

```

```

document.write(«xmlDoc is loaded, ready for use»);
</script>
</body>
</html>

```

### 3.3.2. Програмний інтерфейс XML DOM

В рамках DOM моделі XML можна розглядати як множину вузлових об'єктів. Доступ до них здійснюється за допомогою JavaScript або інших мов програмування. Програмний інтерфейс DOM включає набір стандартних властивостей і методів.

Властивості представляє деяка суть (наприклад, <day>), а методи – дії над ними (наприклад, додати <lesson>).

У XML DOM використовуються практично ті ж властивості і методи, що і в HTML DOM.

Наприклад, результатом виконання наступного JavaScript-коду буде текстовий вміст елемента <subject> у файлі timetable.xml.

```

txt = xmlDoc.getElementsByTagName(«subject»)[0].childNodes[0].nodeValue;

```

Результат: «Deutsch».

В рамках DOM XML можливі 3 способи доступу до вузлів:

- за допомогою методу `getElementsByTagName(name)`. При цьому повертаються всі вузли з вказаним ім'ям тегу (у вигляді індексованого списку). Перший елемент в списку має нульовий індекс;

- шляхом обходу вузлів дерева з використанням циклічних конструкцій;
  - шляхом переміщення по дереву з використанням зв'язків між вузлами.
- Для визначення довжини списку вузлів використовується властивість `length`.

### 3.3.3. Переміщення між вузлами дерева

У XML DOM зв'язки між вузлами визначені у вигляді наступних властивостей вузлів:

```

parentNode
childNodes
firstChild
lastChild
nextSibling
previousSibling

```

Характер зв'язків між вузлами представлений на наступному рисунку 3.3.3.1.

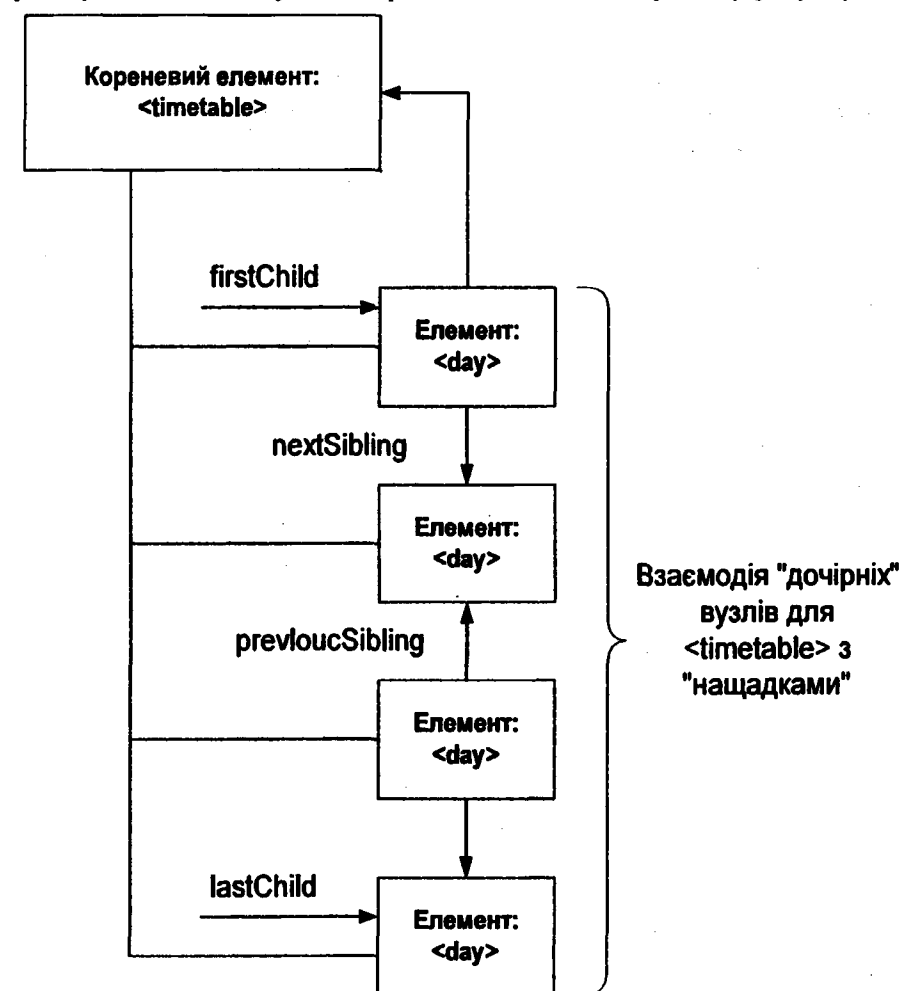


Рис. 3.3.3.1. З'язки між вузлами в XML DOM

3.3.4. Ігнорування порожніх текстових вузлів

Firefox і деякі інші браузері сприймають символи, що не відображаються, як текстові вузли (на відміну від Internet Explorer). Така ситуація призводить до проблем при використанні властивостей firstChild, lastChild, nextSibling, previousSibling. Для того, щоб ігнорувати такі порожні текстові вузли можна використовувати наступний прийом:

```
function get_nextSibling(n)
{
    y= n.nextSibling;
    while (y.nodeType!=1)
    {
        y = y.nextSibling;
    }
    return y;
}
```

Оскільки вузли елементів мають тип 1, то у такому випадку, коли вузол-нащадок не є вузлом елементу, відбуватиметься переміщення до наступного вузла до тих пір, поки не буде знайдений вузол елементу.

3.3.5. Зміна значення атрибуту

Вузли атрибутів можуть набувати текстових значень. Зміна цього значення реалізується або через метод setAttribute(), або через властивість вузла атрибуту nodeValue.

Метод setAttribute() змінює значення існуючого атрибуту або створює новий атрибут.

```
Наприклад:
xmlDoc = loadXMLDoc(<timetable.xml>);
x = xmlDoc.getElementsByTagName('lesson');
x[0].setAttribute(<type>,<lab>);
```

Властивість nodeValue можна використовувати для зміни значення атрибуту вузла:

```
xmlDoc = loadXMLDoc(<timetable.xml>);
x = xmlDoc.getElementsByTagName(<lesson>)[0];
y = x.getAttributeNode(<type>);
y.nodeValue = <lab>;
```

Видалення вузла з дерева реалізується за допомогою методу removeChild():

```
xmlDoc=loadXMLDoc(<timetable.xml >);
y=xmlDoc.getElementsByTagName(<lesson>)[0];
xmlDoc.documentElement.removeChild(y);
```

3.3.6. Властивості об'єкту Node

IE: Internet Explorer, F: Firefox, O: Opera, W3C: (Стандарт).

Властивість	Опис	IE	F	O	W3C
		Версія			
baseURI	Повертає абсолютний URI вузла	Hi	1	Hi	Так
childNodes	Повертає властивість NodeList (список дочірніх вузлів)	5	1	9	Так
firstChild	Повертає перший дочірній вузол	5	1	9	Так
lastChild	Повертає останній дочірній вузол	5	1	9	Так
localName	Повертає локальну частину імені вузла	Hi	1	9	Так
namespaceURI	Повертає URI вузла в просторі імен	Hi	1	9	Так
nextSibling	Повертає наступний дочірній вузол	5	1	9	Так
nodeName	Повертає ім'я вузла залежно від типу	5	1	9	Так
nodeType	Повертає тип вузла	5	1	9	Так
nodeValue	Встановлює або повертає значення вузла залежно від типу	5	1	9	Так
ownerDocument	Повертає кореневий елемент (об'єкт document) для вузла	5	1	9	Так
parentNode	Повертає батьківський вузол	5	1	9	Так
prefix	Встановлює або повертає префікс простору імен вузла	Hi	1	9	Так
previousSibling	Повертає безпосередньо попередній вузол	5	1	9	Так
textContent	Встановлює або повертає текстовий вміст вузла	Hi	1	Hi	Так
xml	Повертає XML код вузла	5	Hi	Hi	Hi

Методи об'єкту Node:

Метод	Опис	IE	F	O	W3C
1	2	3	4	5	6
appendChild()	Додати новий вузол в кінець списку дочірніх вузлів	5	1	9	Так
cloneNode()	Клонування вузла	5	1	9	Так
Метод	Опис	IE	F	O	W3C
compareDocumentPosition()	Порівняння позицій двох вузлів	Hi	1	Hi	Так
getFeature(feature,version)	Повертає об'єкт DOM, що реалізовує спеціалізований API			Hi	Так
getUserData(key)	Повертає об'єкт, що асоціюється з ключем поточного вузла. Перед цим об'єкт має бути асоційований з поточним вузлом шляхом виклику setData з тим же ключем			Hi	Так
hasAttributes()	Повертає дійсне значення, якщо вузол має атрибути	Hi	1	9	Так
hasChildNodes()	Повертає дійсне значення, якщо вузол має дочірні вузли	5	1	9	Так

Закінчення таблиці «Методи об'єкту Node»

1	2	3	4	5	6
insertBefore()	Вставляє новий вузол перед існуючим вузлом	5	1	9	Так
isDefaultNamespace(URI)	Визначає, чи є вказаний namespace URI значенням за замовчуванням			Hi	Так
isEqualNode()	Перевіряє рівність двох вузлів	Hi	Hi	Hi	Так
isSameNode()	Перевіряє ідентичність двох вузлів	Hi	1	Hi	Так
isSupported(feature, version)	Визначає чи підтримується вказана характеристика вузлом			9	Так
removeChild()	Видаляє дочірній вузол	5	1	9	Так
replaceChild()	Замінює дочірній вузол	5	1	9	Так
setUserData(key, data, handler)	Асоціює об'єкт з ключем у вузлі			Hi	Так

### ТЕМА 3.4. ФОРМАТУВАННЯ І ПЕРЕТВОРЕННЯ XML-ДОКУМЕНТА З ДОПОМОГОЮ CSS І XSL. XSLT ПЕРЕТВОРЕННЯ XML-ДОКУМЕНТА

3.4.1. Форматування і перетворення XML-документа з допомогою CSS і XSL

3.4.2. Оголошення XSL

3.4.3. Реалізація перетворення за допомогою JavaScript

#### 3.4.1. Форматування і перетворення XML-документа з допомогою CSS і XSL

XSLT можна визначити таким чином:

- XSLT означає XSL Transformations;
- XSLT є найважливішою частиною XSL перетворення;
- XSLT дозволяє перетворювати один XML в інший XML документ;
- XSLT використовує XPath для переміщення по структурі XML документу;
- XSLT є W3C рекомендацією.

XSLT використовується для перетворення XML-документа в інший XML-документ або в документ іншого розпізнаваного браузерами типу, наприклад, HTML або XHTML. Зазвичай XSLT робить це, перетворюючи кожен XML-елемент у відповідний йому (X) HTML елемент.

За допомогою XSLT можна додавати або видаляти елементи і атрибути в результатуючому документі. Також можливе перегруповування і сортування елементів, фільтрація елементів при відображенні і багато що інше.

Фактично XSLT перетворить початкове дерево XML в результуюче дерево XML.

XSLT використовує XPath для пошуку інформації в XML документі, тобто XPath є інструментом навігації по елементах і атрибутах XML-документів.

В процесі перетворення XSLT використовує XPath для пошуку частин вихідного документу, відповідно одному або більше заданих шаблонів. Коли відповідність знайдена, XSLT перетворить знайдену частину вихідного документу у відповідну частину результуючого документу.

XSLT є W3C рекомендацією з листопада 2009 року.

Переважає більшість браузерів мають підтримку XML і XSLT:

- ⇒ починаючи з 6 версії, Internet Explorer підтримує XML, простори імен, CSS, XSLT і XPath. Версія 5 не сумісна з офіційною W3C XSL рекомендацією;
- ⇒ починаючи з версії 1.0.2, Firefox підтримує XML і XSLT (CSS);
- ⇒ Mozilla містить Expat for XML парсер підтримує відображення XML+CSS. Також має підтримку простору імен. Реалізує XSLT перетворення;
- ⇒ починаючи з версії 8, Netscape використовує як ядро Mozilla, і тому має таку ж підтримку XML/XSLT;
- ⇒ починаючи з версії 9, Opera підтримує XML і XSLT (CSS). Версія 8 підтримує тільки XML+CSS.

### 3.4.2. Оголошення XSL

Кореневим елементом, вказуючим на те, що документ є XSL-таблицею стилів наступний:

```
<xsl:stylesheet>
```

або повністю рівноцінний йому:

```
<xsl:transform>
```

Відповідно до W3C XSLT-рекомендації коректний спосіб оголошення таблиці стилів XSL виглядає таким чином:

```
<xsl:stylesheet version="1.0">
```

```
  xmlns:xsl="http://www.w3.org/2009/XSL/Transform">
```

або

```
<xsl:transform version="1.0">
```

```
  xmlns:xsl="http://www.w3.org/2009/XSL/Transform">
```

Для того, щоб XSLT-елементи, атрибути і характеристики були доступні на початку документу необхідно оголосити простір імен XSLT:

```
  xmlns:xsl="http://www.w3.org/2009/XSL/Transform">
```

що вказує на офіційний простір імен W3C XSLT. При цьому також слід вказати атрибут version="1.0".

### 3.4.3. Реалізація перетворення за допомогою JavaScript

У наведених вище прикладах XSLT-перетворення з XML в XHTML виконувалося самими браузером на основі таблиці стилів XSL. Однак таке рішення, пов'язане з включенням XSL-таблиці стилів в XML-документ є не завжди бажаним, оскільки може підтримуватися не всіма браузерами.

Використання як альтернативи JavaScript дозволяє:

- виконувати перевірку типу браузера;
- використовувати відповідні таблиці стилів залежно від типу браузера і потреб користувачів.

Іншим рішенням для браузерів, не підтримуючих XSLT є перетворення XML в XHTML на веб-сервері.

## Контрольні запитання

1. Що собою являє XML?
2. Розкрийте назву поняття XDR схеми. Чим відрізняється ця схема від інших?
3. Назвіть переваги XML?
4. Які недоліки XML?
5. Що робить структурний аналізатор XML?
6. Розкажіть про DTD схему. Що це за схема?
7. Чим собою представляє програмний інтерфейс XML DOM?
8. Які є зміни значення атрибуту XML DOM?
9. З допомогою чого можна перетворити XML-документ з CSS і XSL?
10. Як можна зробити реалізацію перетворення за допомогою JavaScript?
11. Які є властивості об'єкту NODE?
12. Що таке структурний аналіз XML?

## Тести для закріплення матеріалу

### 1. XML –це:

- А) набір розпоряджень SGML, сформульованих у вигляді визначення типу документа (DTD);
- Б) винятково технологія представлених даних;
- В) текстовий формат, призначений для зберігання структурованих даних, для обміну інформацією між програмами, а також для створення на його основі спеціалізованих мов розмітки.

### 2. Мова XML має наступну перевагу:

- А) підтримує Юнікод;
- Б) має фіксований набір тегів;
- В) «плоска» мова.

### 3. До недоліків мови XML можна віднести:

- А) є простим текстом, вільним від ліцензування і яких-небудь обмежень;
- Б) розмір XML документу суттєво більше бінарного представлення тих же даних (порядку 10 разів);
- В) має фіксований набір тегів.

### 4. XML має наступну структуру:

- А) елемент зазвичай складається із відкриваючого і закриваючого тегів;
- Б) сценарна мова програмування;
- В) програма, що написана на мові Java і відкомпільована в байт-код.

### 5. Найчастіше для опису схеми використовуються наступні специфікації:

- А) SGML;
- Б) XDR;
- В) JAVA.

### 6. Атрибут в DTD може мати одним з трьох типів:

- А) рядок;
- Б) атрибут побудови меню;
- В) шрифт.

### 7. Схема XDR – це:

- А) екземпляр XML;
- Б) не є екземпляром XML;
- В) не забезпечує підтримку просторів імен XML.

### 8. Згідно DOM:

- А) XML не залежить від платформи;
- Б) кожен XML елемент – вузол елемента;
- В) можна створювати інтерактивні мультимедіа-застосування.

### 9. Переваги SAX

- А) швидкість роботи вище за рахунок скорочення витрат часу на виділення пам'яті для елементів дерева в разі DOM;
- Б) виконання аплетів відбувається істотно швидше;
- В) у регулярності виразів для роботи.

### 10. Специфікація XSL складається з таких частин:

- А) PHP;
- Б) CGI;
- В) XSLT.

### 11. Завдяки XSLT реалізується відділення даних від їх представлення в:

- А) MVC;
- Б) XQUERY;
- В) XHTML.

### 12. Специфікація XSLT є рекомендацією :

- А) XQUERY;
- Б) W3C;
- В) JAVA.

### 13. XQUERY – це :

- А) мова запитів, розроблена для обробки даних у форматі XML;
- Б) є простим текстом, вільним від ліцензування і яких-небудь обмежень;
- В) набір розпоряджень SGML, сформульованих у вигляді визначення типу документа (DTD).

### 14. В рамках DOM XML можливий спосіб доступу до вузлів це:

- А) шляхом обходу вузлів дерева з використанням циклічних конструкцій;
- Б) результат виконання JAVASCRIPT коду;
- В) додання значення параметра q, яке рівне вмісту текстового поля, до url.

### 15. Властивість nodeValue можна використовувати для:

- А) того, щоб XSLT елементи, атрибути і характеристики були доступні;
- Б) виконання перевірки типу браузера;
- В) зміни значення атрибуту вузла.

### 16. Вузли атрибутів:

- А) можуть набувати текстових значень;
- Б) не можуть набувати текстових значень;
- В) можуть набувати текстових значень шляхом Юнікоду.

### 17. XSLT можна визначити таким чином:

- А) XSLT є екземпляром XML;
- Б) XSLT є результатом виконання JAVASCRIPT коду;
- В) XSLT є найважливішою частиною XSL-перетворення.

## РОЗДІЛ 4



## ВЕБ-ПОРТАЛИ

## ТЕМА 4.1. ІНТЕГРАЦІЯ ТА ВЗАЄМОДІЯ У ВЕБ-МЕРЕЖІ

## 4.1.1. Веб-інтеграція

## 4.1.2. Інтеграція на основі XML

## 4.1.3. Веб-сервіси

## 4.1.4. Специфікація WSDL

## 4.1.5. Протокол SOAP

## 4.1.6. Стандарт DISCO

## 4.1.7. Специфікація UDDI

## 4.1.1. Веб-інтеграція

У багатьох компаніях вже склалася тенденція надавати своїм співробітникам, партнерам і клієнтам доступ до всіх типів інформації і сервісів за допомогою мережі Веб. Проте в корпоративних мережах компаній функціонує величезне число різномірних бізнес-застосувань, створених в різний час, різними організаціями, на базі різних технологій. Завдання веб-інтеграції полягає в тому, щоб об'єднати різномірні веб-застосування і системи в єдине середовище на базі мережі Веб.

Практикуються наступні підходи до веб-інтеграції:

- інтеграція на рівні представлення. Даний рівень дозволяє користувачеві взаємодіяти із застосуванням. Інтеграція на рівні представлення дає доступ до призначеного для користувача інтерфейсу віддалених застосувань;
- інтеграція на рівні функціональності. Дана інтеграція має на увазі забезпечення прямого доступу до бізнес-логіки застосувань. Це досягається безпосередньою взаємодією застосувань з API (програмному інтерфейсі застосувань) або ж взаємодією за допомогою веб-сервісів;
- інтеграція на рівні даних. В даному випадку передбачається доступ до однієї або декількох баз даних, використовуваних віддаленим застосуванням;
- комплексна інтеграція. Комерційні рішення по веб-інтеграції, як правило, включають всі три типи інтеграції.

Використання веб-інтеграції вигідне з багатьох причин:

- ⇒ веб-інтеграція дозволяє розгорнути інформаційні системи на базі сторонніх застосувань без необхідності розбиратися на їх батьківських системах, програмному середовищі і архітектурі баз даних;
- ⇒ SOA і веб-сервіси використовують програмну мову і незалежні інтерфейси між застосуваннями корпоративної інфраструктури ІТ. Це дає очевидні переваги в підтримці, керованості, розгортанні інформаційних мереж;
- ⇒ веб-інтеграція дозволяє конструювати комплексну функціональність, комбінуючи різномірні компоненти за допомогою протоколів веб-сервісів;
- ⇒ веб-інтеграція дозволяє використовувати веб-сервіси розробників;
- ⇒ веб-інтеграція дозволяє розвивати програмні інтерфейси застосувань через протоколи веб-сервісів без програмування.

Для веб-інтеграції, зазвичай, використовується комерційне ПЗ або популярні технології, такі як PHP, Python, Perl, XForms, SOAP і так далі.

## 4.1.2. Інтеграція на основі XML

Велика кількість систем, стандартів і технологій приводить до того, що ефективно зв'язати різні джерела даних в одну систему не вдається. Навіть такі, на перший погляд однорідні джерела, як системи управління базами даних, застосовують мови запитів і формати представлення вибраної інформації, які рідко повністю сумісні між собою. Як наслідок, проекти інтеграції в таких умовах вимагають великих зусиль – потрібно вникати в деталі різних баз даних, протоколів, операційних систем і так далі. В результаті інтеграція декількох застосувань або систем реалізується по схемі рис. 4.1.2.1.

Змусити різні системи працювати разом – надзвичайно трудомістке завдання. Ідея використання XML в інтеграції інформаційних систем зводиться до створення спільної XML-мови, якою могла б користуватися кожна з них.

Таке рішення відразу ж набагато спрощує проект. Замість реалізації взаємодії між кожною парою систем слід всього лише навчити кожен з них «говорити» на XML мові. Інакше кажучи, все зводиться до розробки декількох впаперів (wrapper – пакувальник, програмний засіб створення системної оболонки для стандартизації зовнішніх звернень і зміни функціональної орієнтації системи, що діє), які перекладатимуть із стандартної XML-мови інтегрованої системи на мову, зрозумілу кожній системі окремо.



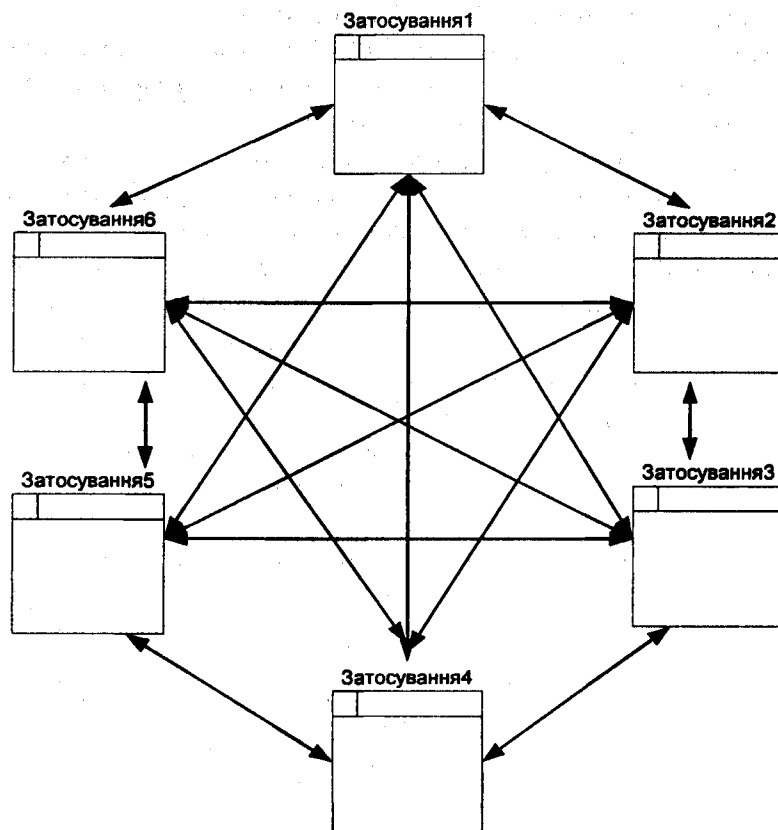


Рис. 4.1.2.1. Реалізація схеми інтеграції декількох застосунків або систем.

В принципі, інтеграція по XML-схемі не відрізняється головним чином від інтеграції на основі будь-якого іншого загального стандарту. Разом з тим, вона має низку вагомих переваг:

- засоби розробки і стандартні бібліотеки для XML існують практично на всіх платформах і для більшості популярних мов програмування;
- методи роботи з XML досить стандартні для того, щоб в різних системах можна було користуватися однаковими прийомами;
- інформація, оформлена у вигляді XML, може оброблятися не тільки машинами, але і людиною (що набагато полегшує відлагодження);
- XML-мова не залежить від апаратних і програмних платформ, що дозволяє зв'язувати різноманітні системи;
- виразна потужність XML досить велика для того, щоб описати дані практично будь-якої складності.

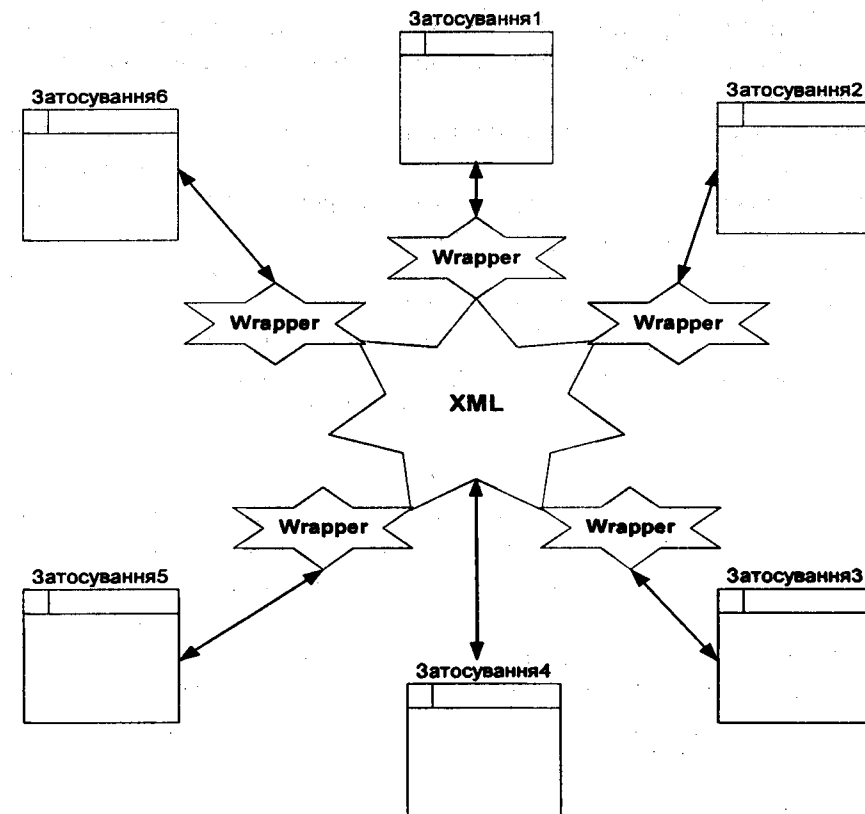


Рис. 4.1.2.2. Схема використання XML застосуваннями з допомогою вранперів.

Інтеграція на основі XML практично реалізується в рамках протоколів:

- ⇒ *XML-RPC*. Це протокол віддаленого виклику процедур з передачею даних у форматі XML через TCP-порт 80, тобто http-порт;
- ⇒ *WDDX (Web Distributed Exchange)* є механізмом обміну складними структурами даних по протоколу HTTP. Протокол базується не на структурах, а на подіях;
- ⇒ *ebXML (electronic business XML)* – XML для електронного бізнесу. Основне призначення – надання відкритої XML-інфраструктури, що забезпечує безпечне глобальне використання інформації електронного бізнесу;
- ⇒ *веб-сервіси (веб-служби)*.

### 4.1.3. Веб-сервіси

Веб-сервіс (web-service) – програмна система, що має ідентифікатор URI і загальнодоступні інтерфейси, які визначені на мові XML. Опис цієї програмної системи може бути знайдено іншими застосуваннями, які можуть взаємодіяти з нею відповідно до цього опису за допомогою повідомлень, заснованих на XML, і таких, що передаються за допомогою інтернет-протоколів. Веб-служба є одиницею модульності при використанні сервіс-орієнтованої архітектури застосування.

Сервіс-орієнтована архітектура (SOA, service-oriented architecture) – модульний підхід до розробки програмного забезпечення, заснованого на використанні сервісів із стандартизованими інтерфейсами.

У основі SOA лежать принципи багатократного використання функціональних елементів ІТ, уніфікації типових операційних процесів. Компоненти програми можуть бути розподілені по різних вузлах мережі, де пропонуються як незалежні і слабо зв'язані, так замінювані сервіси-застосування.

Інтерфейс компонентів SOA-програми здійснює інкапсуляцію деталей реалізації конкретного компоненту (ОС, мови програмування і т. п.).

SOA добре зарекомендувала себе при побудові крупних корпоративних програмних систем. Багато розробників та інтеграторів пропонують інструменти і рішення на основі SOA (наприклад, платформи Microsoft .NET, IBM WebSphere, SAP NetWeaver, Diasoft та ін.).

Веб-сервіси .NET мають наступні переваги:

- відкритість стандартів. У веб-сервісах відсутні які-небудь приховані або недоступні елементи. Кожен аспект технології, від способу пошуку веб-сервісу до її опису і організації зв'язку з нею, визначений загальнодоступними стандартами;
- міжплатформеність. Мова програмування, яка дозволяє створювати XML-документи і відправляти інформацію за допомогою HTTP, дозволяє взаємодіяти з будь-яким веб-сервісом. Можна отримувати веб-послугу з системи, відмінної від .NET;
- простота;
- підтримка повідомлень на зрозумілій людині мові. Перехід від двійкових стандартів, що вживаються в COM і CORBA, до XML-тексту дозволив спростити виправлення помилок і забезпечив можливість здійснювати взаємодію з веб-сервісами по звичайних каналах HTTP.

Реалізація веб-сервісів .NET здійснюється так само просто, як і активізація віддаленого веб-сервісу або виклик методу локального класу. Це досягається за рахунок застосування інструментів, що надаються системою .NET Framework, які дозволяють створити повноцінний веб-сервіс, без необхідності вивчення деталей роботи таких стандартів, як SOAP, WSDL і UDDI. При цьому виконуються наступні дії:

- ⇒ веб-сервіс розробляється як .NET-клас з атрибутами, які ідентифікують його як веб-сервіс з деякими функціями;
- ⇒ у середовищі .NET автоматично створюється документ WSDL, де описується, як клієнт повинен взаємодіяти з веб-сервісом;
- ⇒ споживач знаходить створений веб-сервіс і може добавляти відповідне веб-посилання в проект Visual Studio .NET;
- ⇒ у середовищі .NET здійснюється автоматична перевірка документа WSDL і генерується проксі-клас, який дозволяє споживачеві взаємодіяти з веб-сервісом;
- ⇒ споживач викликає один з методів вашого класу веб-сервісу. З його точки зору цей виклик зовні нічим не відрізняється від виклику методу будь-якого

іншого класу, хоча взаємодія відбувається насправді з проксі-класом, а не з веб-сервісом;

- ⇒ проксі-клас перетворює передані параметри в повідомлення SOAP і відправляє його веб-сервісу;
- ⇒ потім проксі-клас отримує SOAP-відповідь, перетворює його у відповідний тип даних і повертає його як звичайний тип даних .NET;
- ⇒ споживач використовує отримані дані.
- ⇒ При роботі веб-сервісів .NET використовується технологія ASP .NET, що є частиною системи .NET Framework. Вона також вимагає підтримки з боку серверу Microsoft IIS.

Робота веб-сервісів побудована на використанні декількох відкритих стандартів:

- ⇒ XML – розширювана мова розмітки, призначена для зберігання і передачі структурованих даних;
- ⇒ SOAP – протокол обміну повідомленнями на базі XML;
- ⇒ WSDL – мова опису зовнішніх інтерфейсів веб-сервісів на базі XML;
- ⇒ UDDI – універсальний інтерфейс розпізнавання, опису і інтеграції (Universal Discovery, Description, and Integration). Каталог веб-сервісів і відомостей про компанії, що надають веб-сервіси в загальне користування або конкретним компаніям.

#### 4.1.4. Специфікація WSDL

Кожен веб-сервіс представляє документ WSDL (Web Service Description Language – мова опису веб-сервісу), в якому описується все, що клієнтові необхідно для роботи з цим сервісом. WSDL-документ надає простий і послідовний спосіб задавання розробником синтаксису виклику будь-якого веб-методу. Більше того, цей документ дозволяє використовувати інструменти автоматичного генерування проксі-класів, підібні включеним в середовище Visual Studio .NET і .NET Framework. Завдяки вказаним засобам використання веб-сервіс є таким же простим, як і застосування локального класу.

WSDL-документ має заснований на XML формат, відповідно до якого інформація підрозділяється на п'ять груп. Перші три групи це абстрактні визначення, не залежні від особливостей платформи, мережі або мови, а дві групи, що залишилися, включають конкретні описи.

#### 4.1.5. Протокол SOAP

Зв'язок між веб-сервісами і їх клієнтами здійснюється за допомогою повідомлень у форматі XML.

SOAP (Simple Object Access Protocol – простий протокол доступу до об'єктів) є протоколом повідомлень для вибору веб-сервісів.

Основна ідея стандарту SOAP полягає в тому, що повідомлення мають бути закодовані в стандартизованому XML-форматі.

Крім повідомлень SOAP, для обміну даними з сервісами .NET можна використовувати методи GET і POST протоколу HTTP.

Переваги застосування формату SOAP перед іншими форматами для передачі даних:

- кодувати в XML структури даних і набори DataSet з використанням SOAP так само легко, як і дані простих скалярних типів;
- при використанні SOAP-повідомлень надаються додаткові інструменти, що дозволяють легко додавати, наприклад, функції забезпечення безпеки або трасування;
- є набори інструментів SOAP для різних мов програмування (і навіть для попередніх версій Microsoft C++ і Visual Basic). Інакше, для того, щоб забезпечити зв'язок з сервісом за допомогою методів GET і POST протоколу HTTP, доведеться самостійно конструювати рядок запиту, а потім проводити синтаксичний аналіз відповіді.

#### 4.1.6. Стандарт DISCO

Стандарт DISCO надає простий спосіб отримання доступу до файлів маніфестів, що дозволяє групувати посилання на веб-сервіси.

DISCO-файл може включати файли різних веб-серверів і підтримує «динамічний пошук» – автоматичний пошук каталогу файлів веб-сервісів на сервері.

Файли маніфесту корисні тим, що об'єднують множину веб-сервісів в єдиному списку, однак вони не дозволяють клієнтам відшукувати веб-сервіси певного типу без вказівки найменування компанії-розробника.

#### 4.1.7. Специфікація UDDI

Специфікація UDDI (Universal Description, Discovery, and Integration – універсальний опис, пошук і інтеграція) дозволяє уникнути вказаних проблем за допомогою використання спеціального сховища (репозиторія), де підприємства і організації можуть розмішувати дані про сервіси, що надаються ними. Ініціаторами створення технології UDDI стали більше 100 компаній, включаючи Sun і Microsoft. Об'єднавши свої зусилля, ці компанії розробили проект специфікації UDDI, яка після закінчення 18 місяців була стандартизована.

Інформація в цьому репозиторії повинна оновлюватися вручну. З цією метою деякі «вузлові оператори» зберігають ідентичні копії репозиторія UDDI. Ці компанії забезпечують зберігання вказаного репозиторія і безкоштовний доступ до нього для популяризації веб-сервісів. Крім того, Майкрософт включила версію UDDI в програмне забезпечення сервера Windows .NET для використання в корпоративних мережах Інтранету.

У сховищі UDDI містяться відомості про підприємства, що надають веб-сервіси про тип кожного сервісу і зв'язки з інформацією та специфікаціями, що відносяться до цих сервісів. Інтерфейс UDDI сам по собі є веб-сервісом. Для реєстрації або пошуку служби слід відправити SOAP-повідомлення.

Головними недоліками веб-сервісів є менша продуктивність і більший розмір мережевого трафіку в порівнянні з такими технологіями як RMI, CORBA, DCOM за рахунок використання текстових XML-повідомлень.

## ТЕМА 4.2. РОЗРОБКА ВЕБ-СЛУЖБИ В ASP.NET

### 4.2.1. Створення за допомогою ASP.NET веб-служби

### 4.2.2. Розробка веб-служби в ASP.NET. Створення проксі-збірки для веб-служби

### 4.2.1. Створення за допомогою ASP.NET веб-служби

Розглянемо, як приклад, створення за допомогою ASP.NET веб-служби, яка переводить будь-яке ціле десяткове число в один з форматів по вибору: двійковий, вісімковий, десятковий.

Приклад 1. Створимо новий проект (тип: ASP Web Service), наприклад, під ім'ям ASPNETCalcWebService. В результаті автоматично буде згенерований файл Service1.asmx.cs.

Програмна логіка веб-служби буде реалізована на мові C# в CodeBehind файлі Service1.asmx.cs:

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
```

```
namespace ASPNETCalcWebService
{
    /// <summary>
    /// Summary description for Service1
    /// </summary>
    [WebService(Namespace = «http://tempuri.org/»)]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    [ToolboxItem(false)]
    public class Service1 : System.Web.Services.WebService
    {
        [WebMethod]
        public string HelloWorld()
        {
            return «Hello World»;
        }
    }
}
```

2. Для реалізації логіки веб-служби в цьому файлі замінимо у файлі Service1.asmx.cs метод HelloWorld() на 3 нових методи, за допомогою яких виконуватимуться всі перетворення:

```
using System;
```

```
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
```

```
namespace ASPNETCalcWebService
{
    /// <summary>
    /// Summary description for Service1
    /// </summary>
    [WebService(Namespace = «http://tempuri.org/»)]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    [ToolboxItem(false)]
    public class Service1 : System.Web.Services.WebService
    {
        //Uncomment the following line if using designed components
        //InitializeComponent();
    }
```

// Перетворення в двійкову систему числення

```
[WebMethod]
public string Binary(int x){
    return Convert.ToString(x, 2);
}
```

// Перетворення у вісімкову систему числення

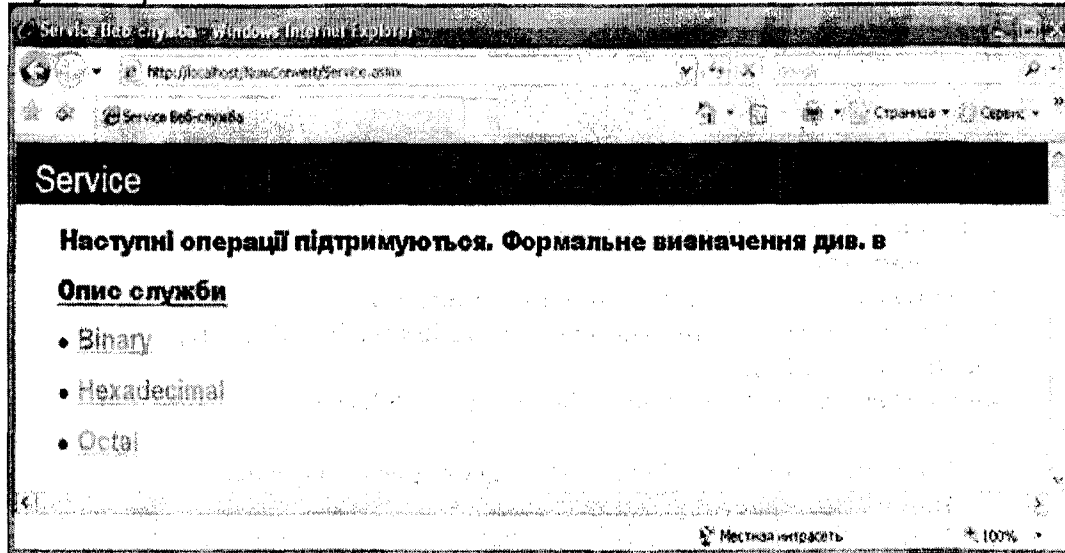
```
[WebMethod]
public string Octal(int x)
{
    return Convert.ToString(x, 8);
}
```

// Перетворення в шістнадцяткову систему числення

```
[WebMethod]
public string Hexadecimal(int x)
{
    return Convert.ToString(x, 16);
}
```

Атрибут `WebMethod` в цьому файлі вказує на те, що описуваний метод має бути доступний по протоколу HTTP для користувачів.

3. Відкомпілюємо і запустимо проект. В результаті у браузері буде отримана наступна сторінка:



4. Активуємо гіперпосилання «Опис служби». У вікні браузера повинен з'явитися опис веб-служби у форматі WSDL.

Приклад 2. (html, txt)

```
<?xml version="1.0" encoding="utf-8" ?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
  xmlns:tns="http://tempuri.org/"
  xmlns:s="http://www.w3.org/2001/XMLSchema"
  xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
  xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
  targetNamespace="http://tempuri.org/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <s:schema elementFormDefault="qualified" targetNamespace="http://tempuri.org/">
      <s:element name="Binary">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="x" type="s:int" />
          </s:sequence>
        </s:complexType>
      </s:element>
```

```
<s:element name="BinaryResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="BinaryResult"
        type="s:string" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="Octal">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="1" maxOccurs="1" name="x" type="s:int" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="OctalResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="OctalResult" type="s:string" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="Hexadecimal">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="1" maxOccurs="1" name="x" type="s:int" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="HexadecimalResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="HexadecimalResult"
        type="s:string" />
    </s:sequence>
  </s:complexType>
</s:element>
</s:schema>
</wsdl:types>
<wsdl:message name="BinarySoapIn">
  <wsdl:part name="parameters" element="tns:Binary" />
</wsdl:message>
<wsdl:message name="BinarySoapOut">
```

```

<wsdl:part name="parameters" element="tns:BinaryResponse" />
</wsdl:message>
<wsdl:message name="OctalSoapIn">
  <wsdl:part name="parameters" element="tns:Octal" />
</wsdl:message>
<wsdl:message name="OctalSoapOut">
  <wsdl:part name="parameters" element="tns:OctalResponse" />
</wsdl:message>
<wsdl:message name="HexadecimalSoapIn">
  <wsdl:part name="parameters" element="tns:Hexadecimal" />
</wsdl:message>
<wsdl:message name="HexadecimalSoapOut">
  <wsdl:part name="parameters" element="tns:HexadecimalResponse" />
</wsdl:message>
<wsdl:portType name="ServiceSoap">
  <wsdl:operation name="Binary">
    <wsdl:input message="tns:BinarySoapIn" />
    <wsdl:output message="tns:BinarySoapOut" />
  </wsdl:operation>
  <wsdl:operation name="Octal">
    <wsdl:input message="tns:OctalSoapIn" />
    <wsdl:output message="tns:OctalSoapOut" />
  </wsdl:operation>
  <wsdl:operation name="Hexadecimal">
    <wsdl:input message="tns:HexadecimalSoapIn" />
    <wsdl:output message="tns:HexadecimalSoapOut" />
  </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="ServiceSoap" type="tns:ServiceSoap">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
<wsdl:operation name="Binary">
  <soap:operation soapAction="http://tempuri.org/Binary" style="document" />
<wsdl:input>
  <soap:body use="literal" />
</wsdl:input>
<wsdl:output>
  <soap:body use="literal" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="Octal">
  <soap:operation soapAction="http://tempuri.org/Octal" style="document" />
<wsdl:input>
  <soap:body use="literal" />
</wsdl:input>

```

```

<wsdl:output>
  <soap:body use="literal" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="Hexadecimal">
  <soap:operation soapAction="http://tempuri.org/Hexadecimal"
style="document" />
  <wsdl:input>
    <soap:body use="literal" />
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal" />
  </wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:binding name="ServiceSoap12" type="tns:ServiceSoap">
  <soap12:binding transport="http://schemas.xmlsoap.org/soap/http" />
<wsdl:operation name="Binary">
  <soap12:operation soapAction="http://tempuri.org/Binary" style="document"
/>
  <wsdl:input>
    <soap12:body use="literal" />
  </wsdl:input>
  <wsdl:output>
    <soap12:body use="literal" />
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="Octal">
  <soap12:operation soapAction="http://tempuri.org/Octal" style="document" />
  <wsdl:input>
    <soap12:body use="literal" />
  </wsdl:input>
  <wsdl:output>
    <soap12:body use="literal" />
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="Hexadecimal">
  <soap12:operation soapAction="http://tempuri.org/Hexadecimal"
style="document" />
  <wsdl:input>
    <soap12:body use="literal" />
  </wsdl:input>
  <wsdl:output>
    <soap12:body use="literal" />
  </wsdl:output>

```

```

</wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="Service">
  <wsdl:port name="ServiceSoap" binding="tns:ServiceSoap">
    <soap:address location="http://localhost/NumConvert/Service.asmx" />
  </wsdl:port>
  <wsdl:port name="ServiceSoap12" binding="tns:ServiceSoap12">
    <soap12:address location="http://localhost/NumConvert/Service.asmx" />
  </wsdl:port>
</wsdl:service>
</wsdl:definitions>

```

Якщо необхідно повідомити докладнішу інформацію для користувачів по кожному з доступних в цій веб-службі методів досить буде додати параметр Description в атрибуті WebMethod, наприклад:

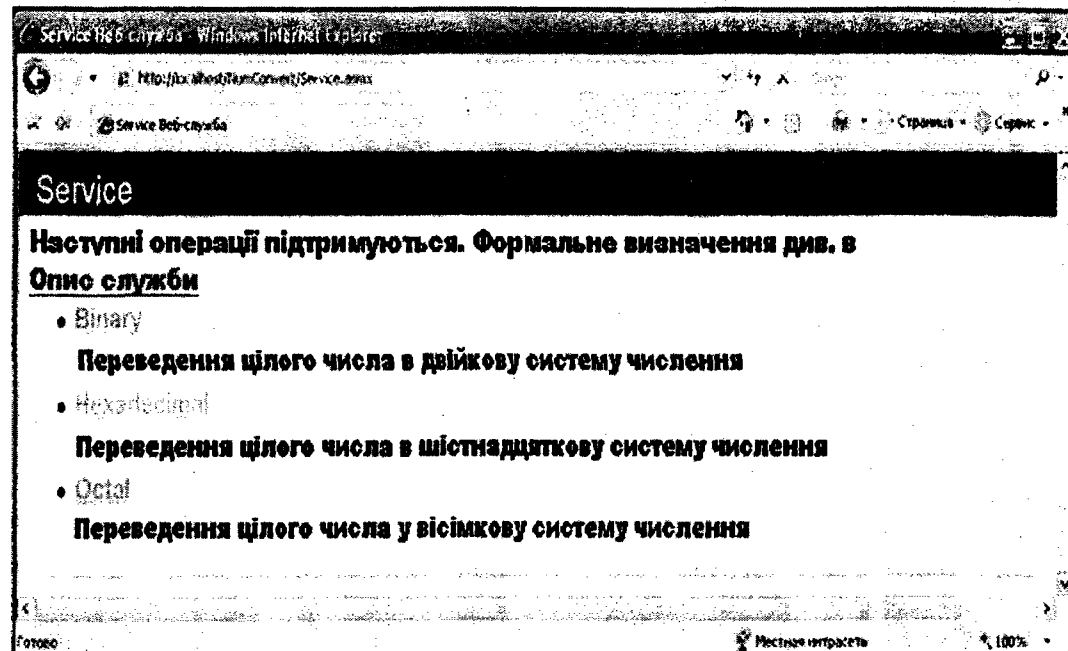
[WebMethod (Description = «Переведення цілого числа в двійкову систему числення»)]

```

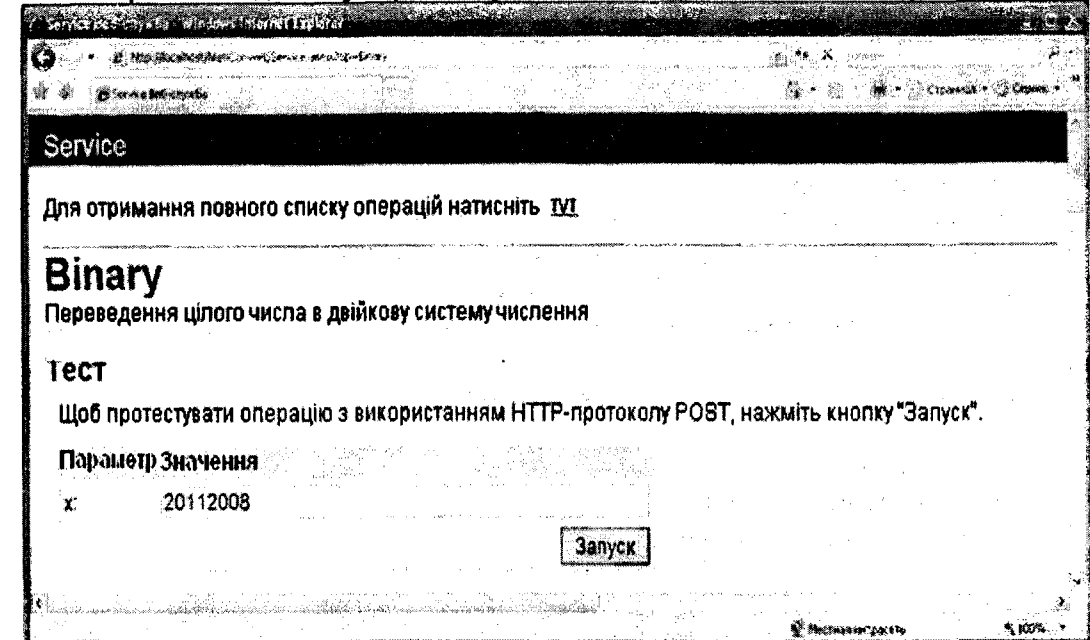
public string Binary(int x)
{
    return Convert.ToString(x, 2);
}

```

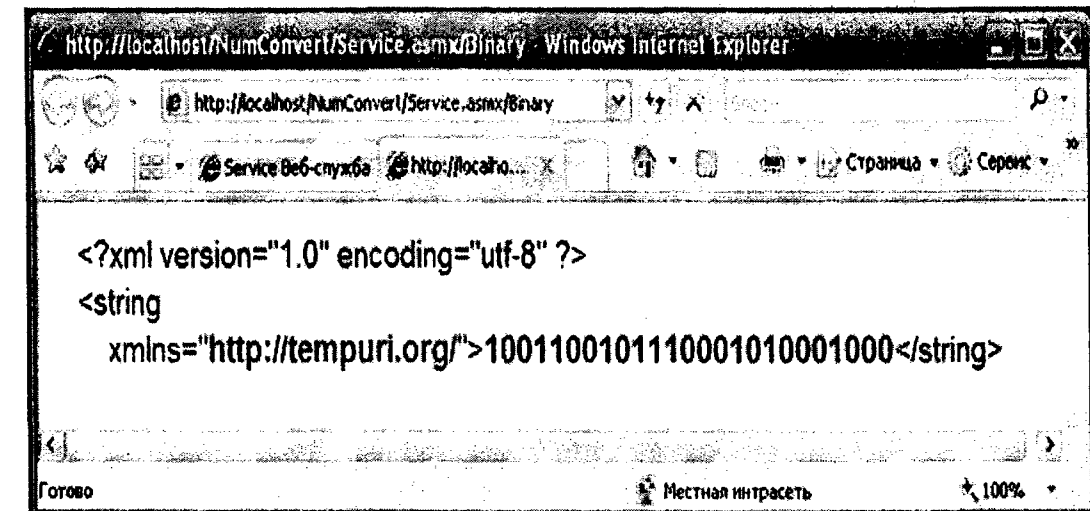
В результаті у браузері буде отримана наступна сторінка:



Виберемо метод Binary. При цьому повинна завантажитися веб-сторінка вигляду:



В результаті роботи веб-служби (після введення числа і натиснення кнопки «Запуск») буде завантажуватися наступна сторінка:



#### 4.2.2. Розробка веб-служби в ASP.NET. Створення проксі-збірки для веб-служби

У попередньому пункті розглядалося створення веб-служби ASPNETCalcWebService, що виконувала переведення десяткового цілого числа в систему числення з основами 2, 8, 16. При цьому передача даних клієнтом (по протоколу SOAP) і отримання їх проводилося в XML-форматі. Це може бути незручним для



клієнта і вимагати розробку додаткових компонентів для конвертації даних в звичний для кінцевого користувача формат.

Уникнути цього можна, якщо звертатися до веб-служби не безпосередньо, а з допомогою проксі-збірки на C# або іншій мові програмування. Така проксі-збірка створюється автоматично у Visual Studio.NET, наприклад, у вигляді клієнта Windows Forms, і бере на себе всю роботу по взаємодії з веб-службою.

#### 1. Створення веб-служби.

Створимо веб-сайт для проекту типу «ASP.NET Web Service»

У складі нового проекту будуть автоматично створені файли Service.asmx (містить код представлення) і Service.cs (містить програмний код).

Замінімо у файлі Service.cs метод HelloWorld() на методи, що реалізують всі перетворення:

// Перетворення в двійкову систему числення

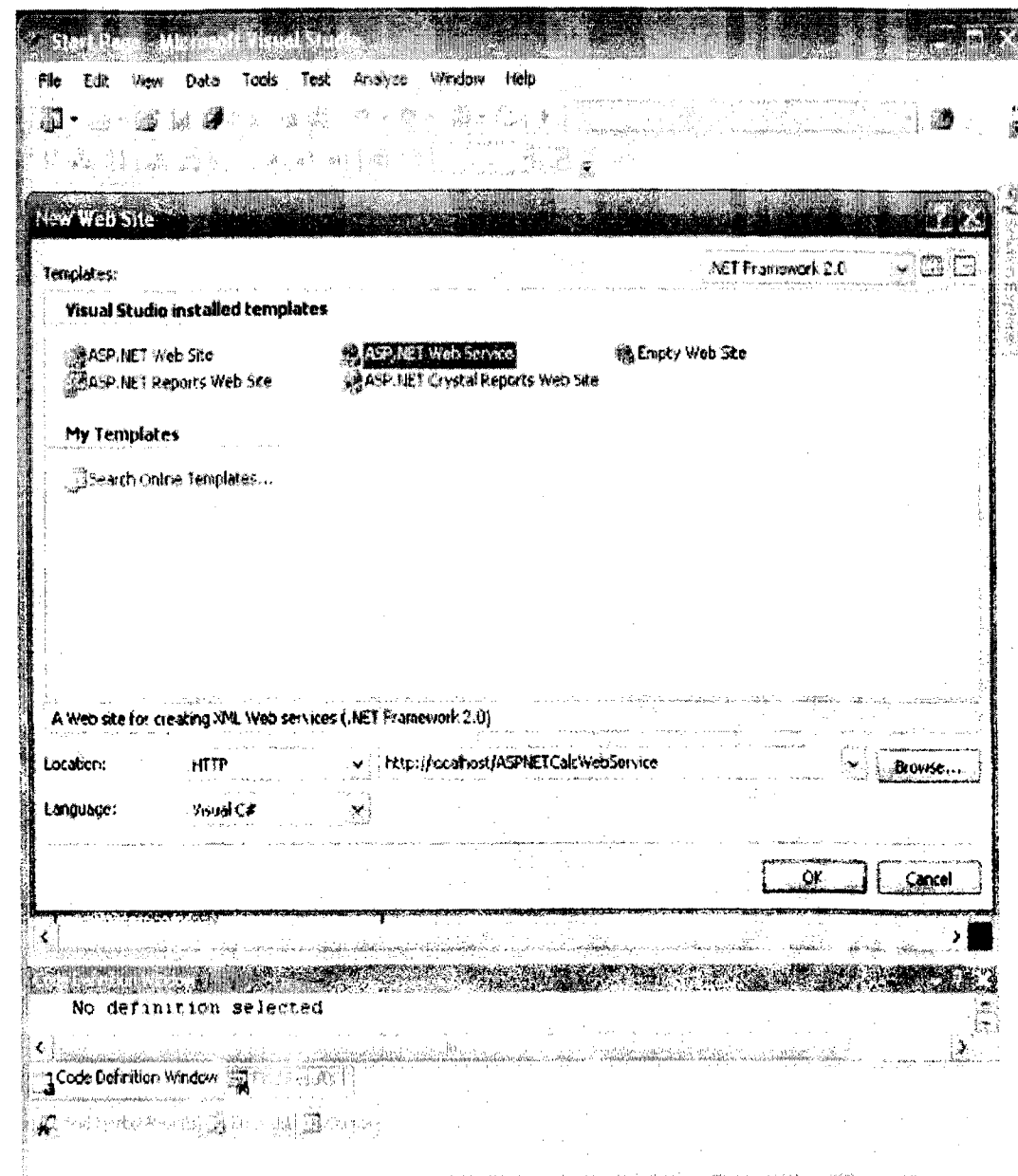
```
[WebMethod]
public string Binary(int x){
    return Convert.ToString(x, 2);
}
```

// Перетворення у вісімкову систему числення

```
[WebMethod]
public string Octal(int x)
{
    return Convert.ToString(x, 8);
}
```

// Перетворення в шістнадцяткову систему числення

```
[WebMethod]
public string Hexadecimal(int x)
{
    return Convert.ToString(x, 16);
}
```

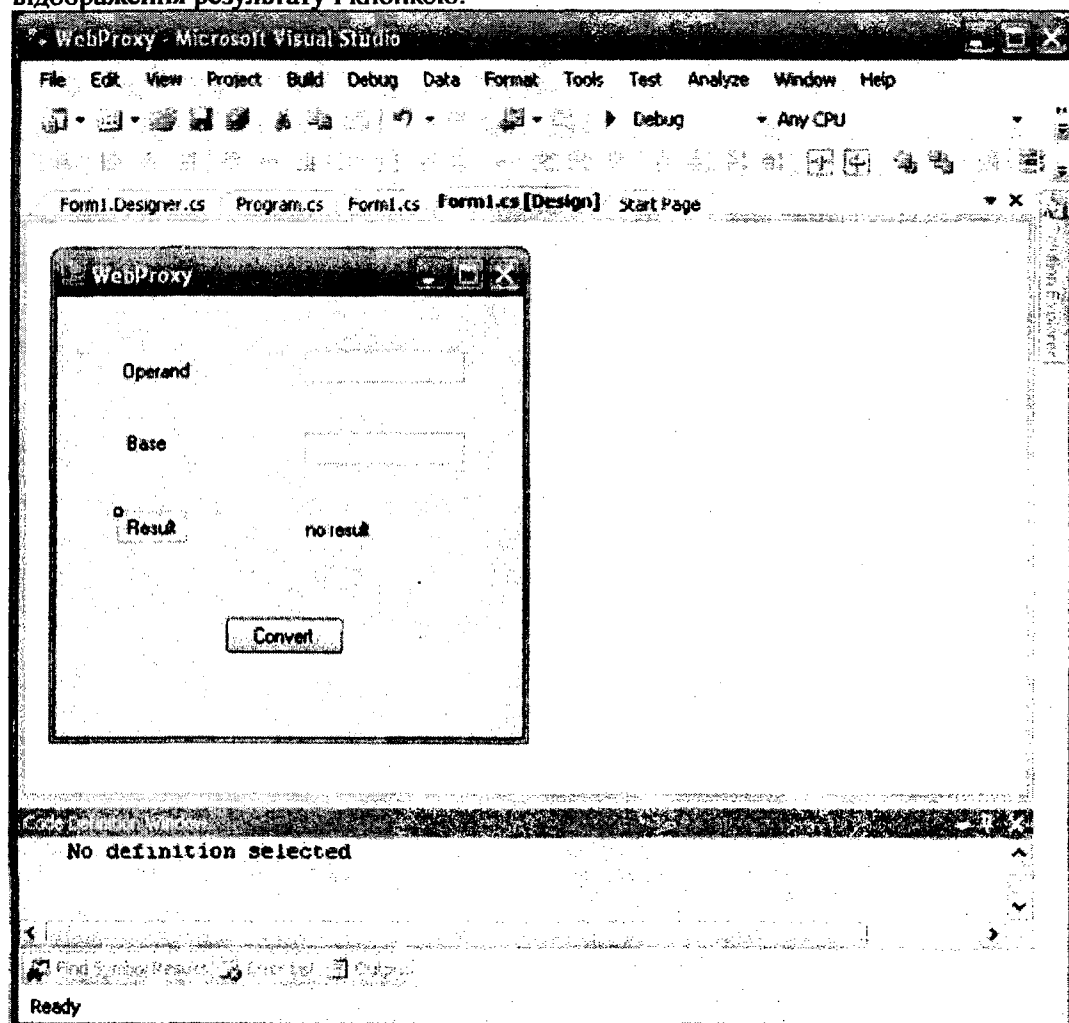


Відкомпілюємо проект і виконаємо пробний запуск веб-служби. URL служби для завантаження в браузері: <http://localhost/ASPNETCalcWebService/Service.asmx>

#### 2. Створення проксі-збірки.

Створимо проект з ім'ям, наприклад, WebProxy (тип проекту Windows Forms Application).

За допомогою дизайнера форм MS Visual Studio підготуємо інтерфейс взаємодії з користувачем у вигляді форми з двома текстовими полями введення даних, полем відображення результату і кнопкою:



Код призначеної для користувача форми, що автоматично згенерований середовищем MS VS, знаходиться у файлі Form1.Designer.cs.

Приклад 1. (html, txt)

namespace WebProxy

```
{
    partial class WebProxy
    {
        private System.ComponentModel.IContainer components = null;

        protected override void Dispose(bool disposing)
        {
```

```
        if (disposing && (components != null))
        {
            components.Dispose();
        }
        base.Dispose(disposing);
    }
}
```

#region Windows Form Designer generated code

```
private void InitializeComponent()
{
    this.label1 = new System.Windows.Forms.Label();
    this.textBox1 = new System.Windows.Forms.TextBox();
    this.textBox2 = new System.Windows.Forms.TextBox();
    this.button1 = new System.Windows.Forms.Button();
    this.label2 = new System.Windows.Forms.Label();
    this.label3 = new System.Windows.Forms.Label();
    this.label4 = new System.Windows.Forms.Label();
    this.SuspendLayout();
    //
    // label1
    //
    this.label1.AutoSize = true;
    this.label1.Location = new System.Drawing.Point(152, 135);
    this.label1.Name = «label1»;
    this.label1.Size = new System.Drawing.Size(47, 13);
    this.label1.TabIndex = 0;
    this.label1.Text = «no result»;
    //
    // textBox1
    //
    this.textBox1.Location = new System.Drawing.Point(155, 33);
    this.textBox1.Name = «textBox1»;
    this.textBox1.Size = new System.Drawing.Size(100, 20);
    this.textBox1.TabIndex = 1;
    //
    // textBox2
    //
    this.textBox2.Location = new System.Drawing.Point(155, 83);
    this.textBox2.Name = «textBox2»;
    this.textBox2.Size = new System.Drawing.Size(100, 20);
    this.textBox2.TabIndex = 2;
    //
    // button1
```

```

//
this.button1.Location = new System.Drawing.Point(105, 194);
this.button1.Name = «button1»;
this.button1.Size = new System.Drawing.Size(75, 23);
this.button1.TabIndex = 3;
this.button1.Text = «Convert»;
this.button1.UseVisualStyleBackColor = true;
this.button1.Click += new System.EventHandler(this.button1_Click);
//
// label2
//
this.label2.AutoSize = true;
this.label2.Location = new System.Drawing.Point(38, 39);
this.label2.Name = «label2»;
this.label2.Size = new System.Drawing.Size(48, 13);
this.label2.TabIndex = 4;
this.label2.Text = «Operand»;
//
// label3
//
this.label3.AutoSize = true;
this.label3.Location = new System.Drawing.Point(41, 83);
this.label3.Name = «label3»;
this.label3.Size = new System.Drawing.Size(31, 13);
this.label3.TabIndex = 5;
this.label3.Text = «Base»;
//
// label4
//
this.label4.AutoSize = true;
this.label4.Location = new System.Drawing.Point(41, 134);
this.label4.Name = «label4»;
this.label4.Size = new System.Drawing.Size(37, 13);
this.label4.TabIndex = 6;
this.label4.Text = «Result»;
//
// WebProxy
//
this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
this.ClientSize = new System.Drawing.Size(292, 266);
this.Controls.Add(this.label4);
this.Controls.Add(this.label3);
this.Controls.Add(this.label2);

```

```

this.Controls.Add(this.button1);
this.Controls.Add(this.textBox2);
this.Controls.Add(this.textBox1);
this.Controls.Add(this.label1);
this.Name = «WebProxy»;
this.Text = «WebProxy»;
this.Load += new System.EventHandler(this.WebProxy_Load);
this.ResumeLayout(false);
this.PerformLayout();
}

```

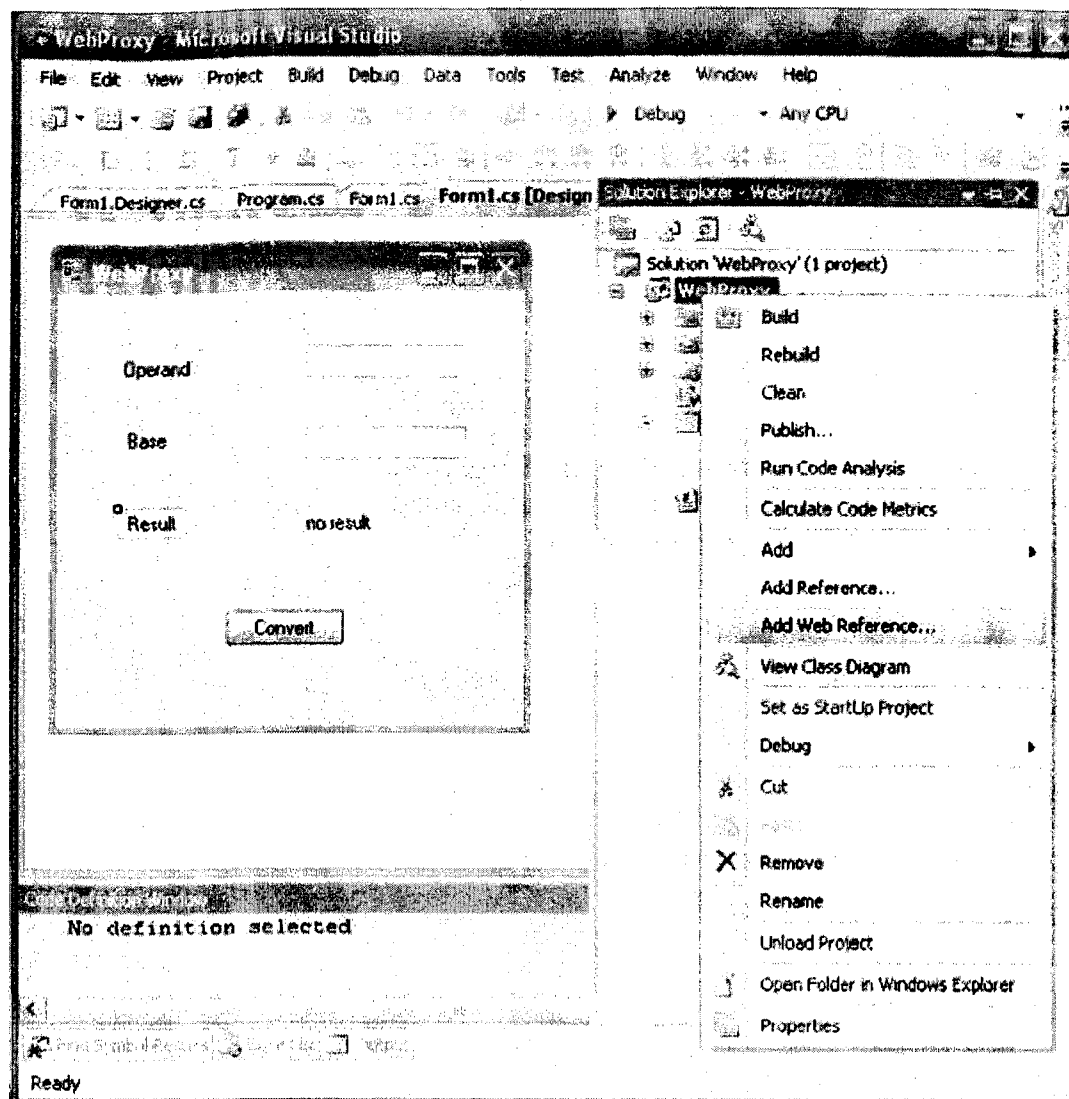
#endregion

```

private System.Windows.Forms.Label label1;
private System.Windows.Forms.TextBox textBox1;
private System.Windows.Forms.TextBox textBox2;
private System.Windows.Forms.Button button1;
private System.Windows.Forms.Label label2;
private System.Windows.Forms.Label label3;
private System.Windows.Forms.Label label4;

```

Додамо веб-посилання <http://localhost/ASPNETCalcWebService/Service.asmx> на веб-службу ASPNETCalcWebService в проект:



Додамо обробник події «натиснення кнопки» у файлі Form1.cs.

Приклад 2.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
```

```
namespace WebProxy
{
```

// додано простір імен localhost

```
using localhost;
public partial class WebProxy : Form
{
    public WebProxy()
    {
        InitializeComponent();
    }
}
```

// обробка події натиснення кнопки

```
private void button1_Click(object sender, EventArgs e)
{
```

// створення об'єкту, пов'язаного з веб-службою

```
    localhost.Service w = new localhost.Service();
```

```
    string ans;
```

// перетворення даних з текстових полів введення у формі в цілі числа

// operand – операнд, NumBase – основа системи числення

```
    int NumBase = int.Parse(textBox2.Text);
    int operand = int.Parse(textBox1.Text);
```

// вибір відповідної функції веб-служби і взаємодія з нею

```
    switch (NumBase)
    {
        case 2: ans = w.Binary(operand); break;
        case 8: ans = w.Octal(operand); break;
        case 16: ans = w.Hexadecimal(operand); break;
        default: ans = «base is undefined»; break;
    }
```

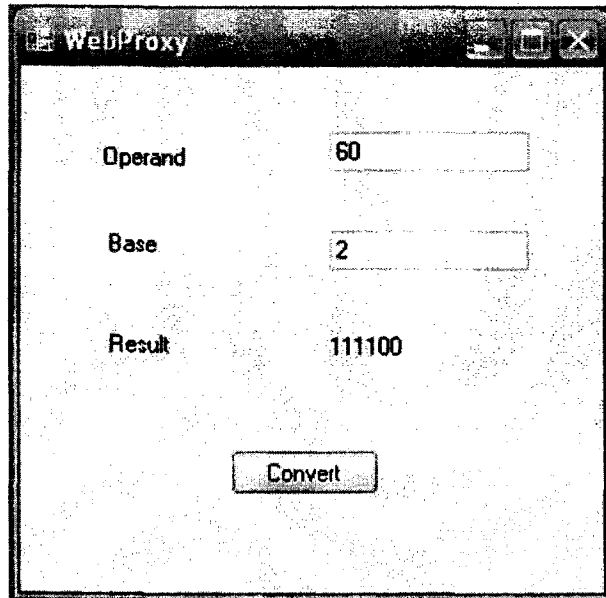
// відображення результату обчислення у формі

```
    label1.Text = ans;
}
```

```
}
}
```

3. Відкомпілюємо проект.

4. Перевірка роботи програми. На скріншоті нижче показана робота програми WebProxy (спільно з веб-службою ASPNETCalcWebService):



## ТЕМА 4.3. РОЗРОБКА ВЕБ-КОНТЕНТА. CMS/CMF

### 4.3.1. Організація процесу розробки веб-контенту. CMS/CMF

### 4.3.2. Типи WCMS-систем. WCMS Drupal

### 4.3.1. Організація процесу розробки веб-контенту. CMS/CMF

Система управління контентом (Content management system, CMS) – комп'ютерна програма, що використовується для створення, редагування, управління і публікації контенту деяким систематичним чином.

Зазвичай такі системи використовуються для зберігання і публікації великої кількості документів, зображень, музики або відео.

Система управління веб-контентом (Web content management system, WCMS або Web CMS) – програмне забезпечення CMS класу, реалізоване у вигляді веб-застосування, і призначене для створення, і управління HTML вмістом. WCMS використовується для управління і контролю великими, динамічно змінними колекціями веб-матеріалу (HTML-документами і пов'язаними з ними картинками). Така система спрощує процес створення, управління, редагування контенту і багато інших важливих завдань, пов'язаних з підтримкою цих процесів.

WCMS надає наступні можливості:

- застосування автоматичних шаблонів відображення (у HTML або XML форматі), що автоматично використовуються для нового або існуючого контенту. Тим самим вид всіх документів може задаватися з одного місця;
- простота редагування контенту. Користувачеві досить легко створювати і управляти контентом, оскільки йому або взагалі не потрібно знання мов програмування або мов розмітки, чи потрібне мінімальне їх знання;
- масштабованість. Можливість розширення функціональності існуючого сайту шляхом установки, що поставляється з дистрибутивом WCMS плагінів і модулів;
- управління документами. Є засоби управління життєвим циклом документів з моменту створення до видалення;
- візуалізація контенту. Будь-який користувач може працювати з віртуальною копією всього веб-сайту, безліччю документів або кодами програм, що дозволяє побачити всі зміни безлічі взаємозв'язаних ресурсів перед їх остаточним використанням.

### 4.3.2. Типи WCMS-систем. WCMS Drupal

Залежно від способу вживання шаблонів для генерації веб-сторінок прийнято виділяти три основних типи WCMS-систем: з автономною обробкою, он-лайн обробкою і гібридні системи. Розглянемо кожну з них:

- ⇒ автономні системи обробляють весь вміст шляхом вживання шаблонів перед публікацією веб-сторінок;
- ⇒ on-line системи застосовують шаблони у момент відвідувань сайту користувачами (або витягують сторінки і кеш);

- ⇒ гібридні системи комбінують перші два підходи. Деякі з них замість статичних сторінок HTML генерують виконувані коди (JSP, PHP, Perl), позбавляючи від необхідності установки WCMS-системи на кожному веб-сервері.

Як приклад системи розглянемо WCMS Drupal.

Drupal – це WCMS система, розроблена на мові PHP, що використовує як сховище даних реляційну базу даних (підтримуються MySQL, PostgreSQL та інші). Архітектура Drupal дозволяє застосовувати його для побудови різних типів сайтів – від блогів і форумів, до інформаційних архівів або сайтів новин.

Функціональність забезпечується модулями, що підключаються та звертаються до спільного API Drupal. Стандартний набір модулів включає, наприклад, такі функції як стрічка новин, блог, форум, завантаження файлів, складальник новин, голосування, пошук і ін.

Найбільш важливі функції, що надаються модулями, які входять в постачання Drupal:

- єдина категоризація всіх видів вмісту (таксономія) – від форумних повідомлень до блогів і новинних статей;
- широкий набір властивостей при побудові рубрикаторів: плоскі списки, ієрархії, ієрархії із спільними предками, синоніми, споріднені категорії;
- вкладеність категорій будь-якої глибини;
- пошук по вмісту сайту, у тому числі пошук по таксономії і користувачах;
- розмежування доступу користувачів до документів;
- динамічна побудова меню;
- підтримка XML-форматів:
- виведення документів в RDF/RSS;
- агрегація матеріалів з інших сайтів;
- BlogAPI для публікації матеріалів за допомогою зовнішніх застосунків;
- підтримка змінних тем оформлення сайту з наданням декількох готових варіантів;
- переклади інтерфейсу сайту різними мовами, а також підтримка ведення різномовного контенту;
- можливість створення сайтів з пересічним вмістом (наприклад спільною базою користувачів або загальними налаштуваннями);
- роздільні конфігурації сайту для різних віртуальних хостів (у тому числі власні набори модулів і тем оформлення для кожного підсайту);
- механізм для обмеження навантаження на сайт (автоматичне відключення при високій відвідуваності частини інформаційних блоків і модулів).

Існує величезна кількість систем як комерційних, так і безкоштовних. Наприклад Майкрософт пропонує реалізацію WCMS системи на базі Windows SharePoint Services.

Каркасна система управління вмістом (Content Management Framework, CMF) – це інструментарій для створення систем управління вмістом, а також окремих веб-застосунків. Деякі CMS, що надають API для розширення своєї функціональності, можна розглядати як CMF, наприклад WCMS Drupal.

## ТЕМА 4.4. РОЗРОБКА RSS-ДЖЕРЕЛ І RSS-ПІДЕРІВ

### 4.4.1. Синдикація і агрегування веб-контенту

### 4.4.2. Формат RSS

### 4.4.3. Приклади розробки RSS-джерел і RSS-підерів

### 4.4.4. Публікація RSS файлу

### 4.4.5. Додавання RSS-каналу за допомогою Microsoft Internet Explorer

### 4.4.6. Задача розробки RSS-джерел і RSS-підерів

### 4.4.1. Синдикація і агрегування веб-контенту

Веб-синдикація – форма синдикації, при якій вміст веб-сайту надається іншим багаточисельним веб-сайтам. Інакше кажучи, веб-синдикація означає створення доступних з сайту веб-потоків (feed), що надають всім користувачам у формі короткого зведення інформації про новий вміст, що з'явився на сайті (це можуть бути новини, повідомлення з форуму і ін.).

Веб-потік – формат даних, що використовується для надання користувачам часто оновлюваного контенту. Розповсюджувачі контенту об'єднують (синдикують) веб-потоки, даючи користувачам можливість підписатися на них. Інша назва для веб-потіку – синдикуваний потік. Створення набору веб-потоків, які доступні одночасно в одному місці називається агрегуванням. Для цього використовуються спеціальні агрегатори.

Агрегатор потоків (feed aggregator) – клієнтське веб-застосування, що збирає синдикуваний веб-контент, такий як заголовки новин, блоги, підкасти та інші в одному місці для зручнішого перегляду.

Для приймаючого сайту веб-синдикація є ефективним способом розміщення більш вичерпної і своєчасної інформації на своїх сторінках.

Для сайту, що передає інформацію, яка синдикується, вигода полягає в його представленості серед різних он-лайн платформ. Крім того, утворюється додатковий трафік, що є простою і безкоштовною формою реклами сайту в мережі веб-серверу.

Взаємодія веб-потоків і агрегаторів відбувається в наступному порядку:

- ⇒ провайдер контенту публікує посилання на потік зі свого сайту;
- ⇒ користувач може зареєструвати це посилання за допомогою програми-агрегатора на своєму комп'ютері;
- ⇒ програма-агрегатор потім опитує всі сервери, що входять в список зареєстрованих потоків, з метою отримання нового контенту;
- ⇒ за наявності нового контенту програма-агрегатор або інформує користувача про наявність такого або відразу ж завантажує його.

Контент веб-потіку є веб-сторінками, гіперпосиланнями або мультимедіа. Витягання контенту з сайту у формі веб-потіку проводиться засобами самого веб-сайту. Проте, не всі веб-сайти можуть мати веб-потік. В цьому випадку можуть бути використані засоби сторонніх агентів. Веб-потік – це веб-документ у XML форматі, що містить тематичні елементи, які містять посилання на повнішу версію матеріалу. Веб-потік є зручним інструментом для доставки структурованої інформації. Користувачі можуть підписуватися на веб-потоки за допомогою агрегаторів або програм для чи-

тання потоків, які комбінують вміст декількох веб-потоків для відображення на одній сторінці (або декількох послідовних сторінках).

Деякі з веб-браузерів містять вбудовані можливості для агрегування потоків. Це робиться шляхом простого введення URL веб-потoku або кліком на гіперпосиланні в браузері. Формат веб-потоків не призначений для безпосереднього читання користувачем, оскільки дозволяє автоматично переносити контент сайт на сайт. Для представлення інформації з веб-потoku зазвичай використовуються 2 формати: RSS і Atom.

Якщо порівнювати веб-потік з більш традиційною поштовою технологією доставки часто оновлюваної інформації, то можна вказати на наступні переваги першого:

- оскільки при підписці користувач не вказує свою адресу електронної пошти, то ця технологія позбавлена таких потенційних загроз як спам, віруси, фішинг і крадіжка особистої інформації;
- при відмові від використання веб-потoku немає необхідності відправляти запит на відмову від підписки; користувач просто виключає даний потік зі свого агрегатора;
- є широкі можливості для автоматичного сортування повідомлень від веб-потоків аж до використання складних правил і регулярних виразів;
- браузери Internet Explorer, Opera, Safari, Firefox та інші можуть працювати з веб-потками через інструменти панелі Закладок, Вибраного і інших. Є також спеціалізовані програми для читання веб-потоків, наприклад FeedDemon, Thunderbird, Outlook та інші.

Агрегатор дозволяє об'єднати інформацію з різних потоків в одному вікні веб-браузера або веб-застосування. Таке застосування називається RSS-каналом, стрічкою новин, агрегатором потоків або пошуковим агрегатором. Підкастинг-агрегатори можуть автоматично завантажувати медіа-файли. Об'єднаний контент агрегатор отримує і інтерпретує у форматі RSS або інших форматах, заснованих на XML, наприклад RDF/XML або Atom. Найбільш розвинені методи агрегування веб-потоків реалізуються на основі технологій AJAX і XML компонентів – веб-віджетів (web widgets).

Багато мов програмування мають бібліотеки функцій, що дозволяють завантажувати, обробляти, генерувати і виконувати віддалене завантаження каналів. Наприклад в Perl є підтримка декількох бібліотек в просторі імен XML – RSS.

#### 4.4.2. Формат RSS

RSS – сімейство XML-форматів, призначених для опису стрічок новин, анонсів статей, змін в блогах і тому подібне.

У різних версіях абревіатура RSS мала різні розшифровки:

- ⇒ *Rich Site Summary* (RSS 0.9x);
- ⇒ *RDF Site Summary* (RSS 0.9 і 1.0);
- ⇒ *Really Simple Syndication* (RSS 2.x).

Першою відкритою офіційною версією RSS стала версія 0.90. Формат був заснований на RDF (Resource Description Framework – стандарт схеми опису потоків) і багатьом здався дуже складним, після чого з'явилася спрощена версія – 0.91.

У 2010 році сталося розділення формату:

- група розробників із списку розсилки «RSS-DEV» запропонувала формат RSS 1.0, який був заснований на стандартах XML і RDF організації W3C. Розширення формату пропонувалося робити через модулі розширень, що описуються у своїх просторах імен. Оскільки проект використовує вже існуючі стандарти, розглядається його використання в рамках технології Semantic Web;
- специфікація RSS 0.92 є розвитком версії 0.91 і орієнтується на тих користувачів, яким RDF-описання здалося надмірно складним. Подальшим розвитком цього напрямку став формат RSS 2.0, який теж підтримує розширення за допомогою модулів, що знаходяться у своїх просторах імен;
- у червні 2006 року з'явився конкурент RSS – формат Atom.

Із-за існування декількох різних версій формату RSS-каналів програми-агрегатори повинні уміти працювати зі всіма варіантами, що створює певні труднощі їх розробникам. Проблеми сумісності виникають також при вставці в RSS-описання невеликих HTML-фрагментів, які в одних випадках оформляються як CDATA вузли, а в інших – як HTML-кодовані PCDATA вузли. Існують проблеми з різними форматами представлення дат і метаданих.

#### 4.4.3. Приклади розробки RSS-джерел і RSS-підерів

RSS – це метод розповсюдження веб-контенту з веб-сайту на інших веб-сайтах. RSS дозволяє виконувати швидкий перегляд новин і змін.

Насамперед RSS призначений для використання на дуже часто оновлюваних веб-сайтах, наприклад:

- ⇒ *сайти новин* (списки новин);
- ⇒ *сайти компаній* (списки новин і продуктів);
- ⇒ *календарі* (списки майбутніх подій і знаменних днів);
- ⇒ *зміни сайтів* (список оновлених і нових сторінок).

Офіційного стандарту RSS як такого не існує. Реально використовуються наступні формати:

- ⇒ *порядка 50% RSS-потоків використовують RSS 0.91;*
- ⇒ *порядка 25% використовують RSS 1.0;*
- ⇒ *останні 25% RSS-потоків розподілено між RSS 0.9 і RSS2.0.*

Створюється RSS документ у вигляді файлу з розширенням .xml. Цей файл розміщується на веб-сайті. RSS-потік реєструється в RSS-агрегаторах.

Приклад RSS документу:

```
<?xml version="1.0" encoding="windows-1251" ?>
<rss version="2.0">
<channel>
<title>My Home Page</title>
<link>http://www.MyHP.edu</link>
<description>Free web building tutorials</description>
<item>
<title>RSS Demo</title>
```



```

<link>http:// www.MyHP.edu/rss</link>
<description>New RSS demo on my home page</description>
</item>
<item>
<title>XML Demo</title>
<link> www.MyHP.edu/xml</link>
<description>New XML demo on my home page</description>
</item>
</channel>
</rss>

```

У першому рядку розміщено оголошення версії XML і кодування документа.

Другий рядок ідентифікує даний документ як RSS документ версії 2.0.

У третьому рядку міститься елемент <channel>, потік, що описує, RSS. Він у свою чергу містить дочірні елементи:

- <title> – описує заголовок RSS каналу;
- <link> – описує гіперпосилання на канал;
- <description> – містить коротку характеристику каналу.

Кожен елемент <channel> може містити один або більше за <item> елементи. Кожен елемент <item> описує окрему статтю RSS джерела. У свою чергу кожен елемент <item> має три обов'язкові дочірні елементи:

- <title>;
- <link>;
- <description>.

Окрім обов'язкових елементів <channel> можуть міститися деякі додаткові дочірні елементи:

- елемент <category> описує категорію потоку і використовується RSS агрегаторами для групування сайтів на основі категорій;
- елемент <copyright> інформує про авторське право на даний документ;
- елемент <image> призначений для відображення зображення при показі потоку агрегатором. У свою чергу він містить три обов'язкові дочірні елементи:
  - <url> – URL-зображення;
  - <title> – альтернативний текст, що відображається при неможливості показу зображення;
  - <link> – містить гіперпосилання на веб-сайт, що містить канал.

Приклад:

```

<image>
<url>http://www.myhp.edu/images/me.gif</url>

<title>My home page</title>

<link>http://www.myhp.edu</link>
</image>

```

Елемент <language> інформує про мову документа. Може бути використаний агрегаторами для угруповання сайтів по мові.

#### Список дочірніх елементів для елементу <channel>:



Елемент	Опис	Статус
<category>	Вказується одна або декілька категорій потоку	Необов'язковий
<cloud>	Реєструє процеси, що одержують негайне сповіщення в разі оновлення потоку	Необов'язковий
<copyright>	Сповіщає про авторське право на документ	Необов'язковий
<description>	Короткий опис каналу	Обов'язковий
<docs>	Вказує URL документів по форматах, використовуваних в потоці	Необов'язковий
<generator>	Вказує на програму, що генерує потік	Необов'язковий
<image>	Дозволяє показувати зображення, що представляє потік в агрегаторі	Необов'язковий
<language>	Вказується мова потоку	Необов'язковий
<lastBuildDate>	Визначається дата останнього оновлення вмісту потоку	Необов'язковий
<link>	Вказується гіперпосилання на канал	Обов'язковий
<managingEditor>	Міститься поштова адреса редактора вмісту потоку	Необов'язковий
<pubDate>	Визначає дату останньої публікації вмісту потоку	Необов'язковий
<skipDays>	Вказує дні, в які агрегатори не повинні оновлювати потік.	Необов'язковий
<skipHours>	Вказується кількість годин, протягом яких агрегатори не повинні оновлювати потік	Необов'язковий
<textInput>	Описується текстове поле вводу, що відображається в потоці	Необов'язковий
<title>	Розміщується ім'я каналу	Обов'язковий
<ttl>	Вказується кількість хвилин протягом яких потік може знаходитися в кеші до оновлення з джерела	Необов'язковий
Елемент	Опис	Статус
<webMaster>	Поміщається адреса електронної пошти веб-дизайнера потоку	Необов'язковий

#### Список дочірніх елементів для елементу <item>:

Елемент	Опис	Статус
<author>	Адреса електронної пошти автора теми	Необов'язковий
<category>	Описує одну або більше категорій для даної теми	Необов'язковий
<comments>	Дозволяє пов'язати тему з коментарями по даній темі	Необов'язковий
<description>	Короткий опис теми	Обов'язковий
<enclosure>	Дозволяє включити мультимедіа файл в дану тему	Необов'язковий
<guid>	Містить унікальний ідентифікатор теми	Необов'язковий
<pubDate>	Містить останню дату публікації теми	Необов'язковий
<source>	Вказує зовнішнє джерело для даної теми	Необов'язковий
<title>	Назва теми	Обов'язковий

#### 4.4.4. Публікація RSS файлу

Створення RSS потоку не обмежується розробкою RSS документу. Необхідно ще опублікувати цей файл. Для цього потрібно буде виконати наступну послідовність дій:

- вибрати відповідну назву для RSS файлу. Розширення має бути .xml;
- перевірити RSS файл на правильність за допомогою відповідної програми-валідатора, наприклад, взятою за адресою <http://www.feedvalidator.org>;
- розмістити RSS файл у відповідному веб-каталозі веб-серверу;
- скопіювати одну з «кнопок»:  або  у веб-каталог;
- вставити вибрану «кнопку» на витікаючу сторінку RSS потоку у вигляді гіперпосилання на RSS файл, наприклад:

```
<a href = «www.myhp.edu/rss/myrss.xml»>
<img src=»www.myhp.edu/rss/rss.gif» width=»35» height=»15»>
</a>
```

- розмістити створений потік RSS в популярних каталогах RSS потоків, при цьому URL потоку повинен посилатися саме на сам XML файл потоку, тобто <http://www.myhp.edu/rss/myrss.xml>;
- можна також зареєструвати потік в популярних пошукових системах, наприклад:
- Yahoo – [http://publisher.yahoo.com/rss\\_guide/submit.php](http://publisher.yahoo.com/rss_guide/submit.php);
- Google – <http://www.google.com/ig>;
- MSN <http://w.moreover.com/site/products/ind/pingserver.html>;
- періодичне оновлення потоку.




В принципі, можна самостійно займатися формуванням даних у відповідному форматі RSS для роботи потоку. Однак є досить зручні високорівневі засоби для автоматизації даної роботи. Наприклад:




- ⇒ *MyRSSCreator* – <http://www.myrsscreator.com/>;
- ⇒ *FeedFire* – <http://www.feedfire.com/site/index.html>.

Для читання RSS потоків використовуються спеціальні програми читання – RSS рідери. Деякі з браузерів мають вбудовані RSS рідери. Зокрема, веб-браузер MS Internet Explorer може інтерпретувати XML файли RSS-потоків і коректно їх відображати.

#### 4.4.5. Додавання RSS-каналу за допомогою Microsoft Internet Explorer

При використанні MS Internet Explorer і Office Outlook можна додавати і проглядати RSS-канали за допомогою будь-якої з цих програм.

При прогляданні веб-сторінки RSS, що міститься в Internet Explorer поряд з кнопкою  (Домашня сторінка) відображається кнопка . Натиснемо кнопку . На веб-сторінці відобразатиметься список доступних RSS-каналів. Виберемо RSS-канал,

який необхідно додати. Можна також натискати кнопки ,  або , розташовані на веб-сторінці.

#### 4.4.6. Задача розробки RSS-джерел і RSS-підєрєв

1. Створення RSS документу за допомогою PHP сценарію і з використанням інтерфейсу DOM XML.

Створимо файл, що містить сценарій на мові PHP.

Приклад 1.

```
<?php
```

```
// Створюється новий документ XML.
```

```
$dfeed = new DOMDocument('1.0', 'utf-8');
```

```
// Створення кореневого елементу <rss>
```

```
$erss = $dfeed->createElement('rss');
```

```
$erss->setAttribute('version', '2.0');
```

```
// І додавання його в дерево документу
```

```
$dfeed->appendChild($erss);
```

```
// Створення елементу <channel> і додавання його до <rss>
```

```
$sechannel = $dfeed->createElement('channel');
```

```
$erss->appendChild($sechannel);
```

```
// Створення і додавання в channel
```

```
// вузлів <title>, <link>, <description>, <language>, <pubDate>
```

```
$sechannel->appendChild( $dfeed->createElement('title', 'RSS-channel title') );
```

```
$sechannel->appendChild( $dfeed->createElement('link', 'http://www.myhp.edu')
```

```
);
```

```
$sechannel->appendChild( $dfeed->createElement('description', 'My RSS demo')
```

```
);
```

```
$sechannel->appendChild( $dfeed->createElement('language', 'en') );
```

```
$sechannel->appendChild( $dfeed->createElement('pubDate', date('r')) );
```

```
// Додавання до вузла <channel> 2 вузлів <item>
```

```
for ( $i = 1; $i <= 2; $i++ )
```

```
{
```

```
    $seitem = $dfeed->createElement('item');
```

```
    $sechannel->appendChild($seitem);
```

```
// Створення дочірніх елементів для <item>
```

```
$seitem->appendChild( $dfeed->createElement('title', 'Item' . $i) );
```

```
    $seitem->appendChild( $dfeed->createElement('link', 'http://www.myhp.edu/
rss/' . $i . '.xml') );
```

```
Seitem->appendChild( $dfeed->createElement('description', 'Description for
'.Si.' item') );
}
```

```
// Збереження документа у файлі demo.rss
$dfeed->save('demo.rss');
?>
```

Створений файл розмістимо на веб-сервері та настроїмо права доступу (право на запис) для веб-сервера до директорії, в якій розміщується сценарій, або до спеціальної директорії, в якій буде створений файл demo.rss (це безпечніше).

Виконаємо сценарій за запитом з веб-браузера. В результаті виконання сценарію вийде наступний документ:

```
<?xml version="1.0" encoding="utf-8" ?>
<rss version="2.0">
<channel>
<title>RSS-channel title</title>
<link>http://www.myhp.edu</link>
<description>my RSS demo</description>
<language>en</language>
<pubDate>Thu, 06 Nov 2012 01:50:53 +0300</pubDate>
<item>
<title>Item1</title>
<link>http://www.myhp.edu/rss/1.xml</link>
<description>Description for 1 item</description>
</item>
<item>
<title>Item2</title>
<link>http://www.myhp.edu/rss/2.xml</link>
<description>Description for 2 item</description>
</item>
</channel>
</rss>
```

2. Читання RSS документа за допомогою PHP сценарію і з використанням інтерфейсу DOM XML.

Підготуємо файл, що містить сценарій на мові PHP:

```
<?php
// Завантаження документа
$dfeed = DOMDocument::load('demo.rss');
```

```
// Читання елементу <channel>
$sechannel = $dfeed->firstChild->firstChild;
```

```
//Читання першого дочірнього елементу вузла <channel> тобто< title>
```

```
$sechild = $sechannel->firstChild;
```

```
// Виведення вмісту дочірніх елементів вузла <channel> до елементу <item>:
```

```
print "<<strong><br>>";
while ( $sechild->tagName != 'item' )
{
    print $sechild->nodeValue. "<<br>>";
    $sechild = $sechild->nextSibling;
}
print "<</strong><br>>";

// Отримання списку вузлів <item>
$slitems = $sechannel->getElementsByTagName('item');

print "<<br>>";
foreach ( $slitems as $seitem )
{
    $stitle = $seitem->firstChild->nodeValue;
    $slink = $seitem->firstChild->nextSibling->nodeValue;
    $sdescr = $seitem->lastChild->nodeValue;

    print "<<a href=" . $slink . ">>". $stitle . "</a> :: <<". $sdescr . "><br>>";
}
print "<<br>>";
?>
```

У методі load() як параметр вкажемо шлях до RSS-файлу.

Створений файл розмістимо на веб-сервері, настроїмо права доступу (право на запис) для веб-сервера до директорії, в якій розміщується сценарій, або до спеціальної директорії, в якій буде створений файл demo.rss (це безпечніше).

Виконаємо сценарій за запитом з веб-браузера.

Результат повинен виглядати таким чином (Рис. 4.4.6.1.):

```
RSS-channel title
http://www.myhp.edu
my RSS demo
en
Thu, 06 Nov 2008 02:15:55 +0300

Item1 :: Description for 1 item
Item2 :: Description for 2 item
```

Рис. 4.4.6.1. Результат виконання сценарію за запитом з веб-браузера

## Контрольні запитання

1. Які підходи практикуються до веб-інтеграції?
2. Що таке веб-сервіс?
3. Які можливості надає WCMS (Web content management system)?
4. Що таке Drupal? Які найбільш важливі функції, що надаються модулями, входять в постачання Drupal?
5. Що таке Веб-синдикація та Веб-потік?
6. В якому порядку відбувається взаємодія веб-потоків та агрегаторів?
7. Які популярні технології використовуються для веб-інтеграції?
8. Чому вигідне використання веб-інтеграції?
9. Що таке система управління контентом? Для чого ці системи потрібні?
10. Які є типи WCMS-систем?
11. Що включає стандартний набір модулів?
12. Що таке каркасна система управління вмістом?

## Тести для закріплення матеріалу

**1. Інтеграція на рівні представлення має доступ до:**

- А) бізнес-логіки застосувань;*
- Б) одної або декількох баз даних;*
- В) призначеного для користувача інтерфейсу віддалених застосувань;*
- Г) призначеного для зберігання інтерфейсу.*

**2. RSS – це сімейство форматів:**

- А) XML;*
- Б) SOAP;*
- В) DISCO;*
- Г) UDDI.*

**3. У червні 2006 р. з'явився конкурсант RSS:**

- А) Atom;*
- Б) XML;*
- В) SOAP;*
- Г) UDDI.*

**4. За допомогою яких команд можна додавати і програмувати RSS-канали:**

- А) Microsoft Internet Explorer ;*
- Б) Mozilla Firefox ;*
- В) Opera;*
- Г) Hrome.*

**5. Основна ідея стандарту SOAP полягає в тому, що повідомлення мають бути:**

- А) закодовані в XML-форматі;*
- Б) закодовані в Atom-форматі;*
- В) закодовані в UDDI-форматі;*
- Г) закодовані в RSS-форматі.*

**6. Файли маніфесту корисні тим, що:**

- А) розділяють множину веб-сервісів;*
- Б) об'єднують множину веб-сервісів в єдиний список;*
- В) видаляють множину веб-сервісів;*
- Г) збільшують множину веб-сервісів.*

**7. Інтерфейс UDDI сам по собі є:**

- А) антивірусною програмою;*
- Б) інтернет браузер;*
- В) веб-сайтом;*
- Г) веб-сервісом.*

**8. Головним недоліком веб-сервісів є:**

- А) безпека;*

- Б) менша продуктивність і більший розмір мережевого трафіку;
- В) більша продуктивність і менший розмір мережевого трафіку;
- Г) менша продуктивність і менший розмір мережевого трафіку.

**9. Wrapper –це:**

- А) пакувальник;
- Б) інтерфейс;
- В) веб-сайт;
- Г) антивірусна програма.

**10. Яка інтеграція передбачає доступ до 1 або декількох баз даних:**

- А) інтерфейс на рівні функціональності;
- Б) інтерфейс на рівні даних;
- В) комплексна інтеграція;
- Г) інтерфейс на рівні представлення.

**11. Веб-інтеграція дозволяє:**

- А) використання веб-сайтів;
- Б) використовувати веб-сервіси розробників;
- В) забезпечувати безпеку;
- Г) забезпечує Windows антивірусну захищеність.

**12. На якій основі інтеграція реалізовується в рамках протоколів XML-RPC, WDDX:**

- А) XML ;
- Б) SOAP;
- В) RSS;
- Г) DISCO.

**13. Веб-сервіс .NET має такі переваги:**

- А) відкритість стандартів, простоту;
- Б) складність;
- В) кодування;
- Г) додавання методів GET і POST

**14. UDDI дозволяє уникати вказаних проблем за допомогою використання:**

- А) Wrapper;
- Б) спеціального сховища;
- В) протокола;
- Г) вузлового оператора.

**15. WCMS має такі можливості:**

- А) кодування, відкритість стандартів;
- Б) складність;
- В) простота і управління документами;
- Г) кодування.

**16. Drupal – це система розроблена на мові:**

- А) PHP;
- Б) XML;
- В) Pascal;
- Г) Basic.

## РОЗДІЛ 5



## ТЕХНОЛОГІЯ AJAX

## ТЕМА 5.1. ВСТУП У ТЕХНОЛОГІЮ AJAX. РОЗРОБКА МОБІЛЬНИХ WEB-ЗАСТОСУВАНЬ

### 5.1.1. Вступ у технологію AJAX

#### 5.1.2. Розробка мобільних веб-застосовувань

#### 5.1.1. Вступ у технологію AJAX

AJAX (Asynchronous JavaScript and XML) – це концепція використання декількох суміжних технологій, орієнтована на розробку високоінтерактивних застосовувань, що швидко реагують на дії користувача, виконують велику частину роботи на стороні клієнта і взаємодіють з сервером за допомогою позаполосних звернень.

Ajax не можна розглядати як щось завершене. Даний напрямок продовжує розвиватися. Ajax – не одна конкретна технологія, скоріше це сукупність чотирьох технологій, що доповнюють одна одну (рис. 5.1.1.1):

1. **JavaScript** – це мова сценаріїв призначена для включення коду у Web-додаток. Інтерпретатор JavaScript у складі Web-браузера забезпечує взаємодію із вбудованими засобами браузера. Дана мова використовується для створення Ajax-додатків;

2. **CSS** надає можливість визначати стилі елементів Web-сторінки. За допомогою даної технології можна без зусиль забезпечити узгодженість зовнішнього вигляду компонентів додатку. У Ajax CSS використовується для зміни представлення інтерфейсу в процесі інтерактивної взаємодії;

3. **DOM** представляє структуру Web-сторінки у вигляді набору об'єктів, які можна обробляти засобами JavaScript. Це дає можливість змінювати зовнішній вигляд інтерфейсу Ajax-додатку в процесі роботи;

4. **Об'єкт XMLHttpRequest** дозволяє програмістові отримувати дані з Web-серверу у фоновому режимі. Як правило, інформація, що повертається, надається у форматі XML, але даний об'єкт дозволяє також працювати з будь-якими текстовими даними. Не дивлячись на те, що XMLHttpRequest є найбільш гнучким з усіх інструментів спільного призначення, що дозволяють вирішувати подібні завдання, існують й інші способи отримання даних з серверу.

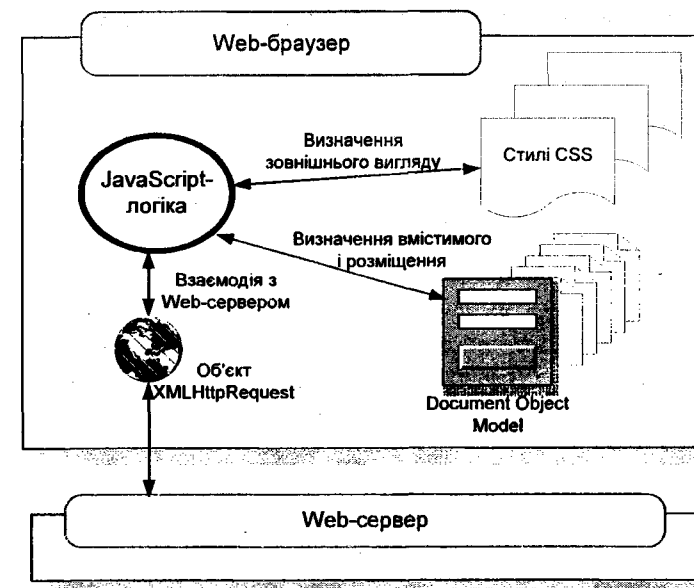


Рис. 5.1.1.1. Основні компоненти Ajax.

У складі кожної сучасної операційної системи є браузер. Таким чином, переважна більшість персональних і портативних комп'ютерів уже готові для запуску Ajax-додатків. Розроблювачі програм на базі Java або .NET можуть лише мріяти про це. Браузери для КПК і мобільних телефонів мають усічений набір можливостей і не підтримують повний набір Ajax-технологій. Варто зауважити, що якби засоби підтримки Ajax і були в наявності, все одно розміри екрану й особливості введення даних створювали б істотні проблеми при роботі з Ajax-додатками. На сьогоднішній день інфраструктура Ajax орієнтована в більшості на персональні й портативні комп'ютери, а для КПК та мобільних телефонів використовується WAP (Wireless Application Protocol) – протокол доступу до ресурсів Інтернет безпосередньо з мобільного телефону, про який буде розказано нижче.

JavaScript визначає бізнес-правила і потік виконання. Document Object Model і каскадні таблиці стилів дозволяють змінювати зовнішній вигляд додатку відповідно до даних, отриманих із серверу в асинхронному режимі. Одержання даних забезпечується об'єктом XMLHttpRequest або іншими подібними засобами.

Позаполосним зверненням називається запит до серверу, який призводить до оперативного оновлення сторінки замість її заміни. Позаполосний виклик HTTP – це HTTP-запит, який видається за межами вбудованого модуля, що забезпечує відправку форм HTTP. Виклик ініціюється подією, пов'язаною із сторінкою HTML і обслуговується компонентою-посередником, зазвичай об'єктом XMLHttpRequest.

Популярність AJAX пов'язана з появою сервісу Google Suggest в 2005 році. Даний сервіс на основі об'єкту XMLHttpRequest надає в розпорядження користувача досить динамічний веб-інтерфейс. В процесі введення символів користувачем в полі пошукового запиту JavaScript відправляє їх на сервер і отримує від нього список підказок.

AJAX застосовується для розробки веб-застосовувань, до яких ставляться наступні вимоги:

- застосування повинне передавати користувачам свіжі дані, отримані з серверу;

➤ нові дані повинні інтегруватися в існуючу сторінку без її повного оновлення. Для роботи з такими застосуваннями в браузері, необхідно, щоб він відповідав вимогам:

- ⇒ підтримка посередників (для позаполосних викликів HTTP). Зазвичай реалізується у формі об'єкту *XmlHttpRequest*;
- ⇒ підтримка оновлюваної моделі DOM.

Об'єкт *XmlHttpRequest* є компактною об'єктною моделлю для відправки сценарієм звернень HTTP в обхід браузеру. Клієнтський код сценарію не може впливати на процес розміщення запиту і результат відправки запиту. *XmlHttpRequest* дозволяє сценарію відправляти http-запити і обробляти отримані відповіді.

Як формат передачі даних зазвичай використовуються JSON або XML.

JSON (JavaScript Object Notation) – текстовий формат обміну даними, заснований на JavaScript, який використовується саме з цією мовою. Не дивлячись на походження від JavaScript, формат вважається незалежним і може використовуватися практично з будь-якою мовою програмування. Для багатьох мов існує готовий код для створення і обробки даних у форматі JSON.

JSON будується на двох структурах даних:

- набір пар ім'я/значення. У різних мовах це реалізовано як об'єкт, запис, структура, словник, хеш-таблиця, список з ключем або асоціативний масив;
- пронумерований набір значень. У багатьох мовах це реалізовано як масив, вектор, список або послідовність.

Наступний приклад показує JSON-застосування об'єкту, що описує викладача.

```
{
  «firstName»: «Дмитро»
  «lastName»: «Угрин»
  «profile»: {
    «position»: «доцент, к.т.н.»
    «department»: «кафедра автоматизованих систем управління»
    «age»: 40
  },
  «subjects»: [
    «веб-технології»
    «веб-дизайн»
  ]
}
```

Microsoft .NET надає в розпорядження розробників свою реалізацію AJAX технології – ASP.NET 2.0 AJAX.

З архітектурної точки зору, інфраструктура ASP.NET 2.0 AJAX складається з двох елементів:

- ⇒ бібліотека клієнтських сценаріїв (реалізована на JavaScript).

Працює в будь-якому сучасному браузері;

- ⇒ набір серверних розширень.

Повністю інтегрується з серверними службами і елементами ASP, що управляють .NET. Розробники можуть створювати веб-сторінки з розширеною функціональністю, використовуючи практично таку ж методику, яка використовується при розробці серверних сторінок ASP.NET.

### 5.1.2. Розробка мобільних веб-застосувань

Для отримання доступу до ресурсів Інтернет за допомогою тільки мобільного телефону, не вдаючись до допомоги комп'ютера або модему був розроблений спеціальний стандарт WAP.

WAP (Wireless Application Protocol) – протокол доступу до ресурсів Інтернет безпосередньо з мобільного телефону, оминаючи комп'ютер та модем.

Для розмітки документів при завантаженні їх в стільникових телефонах та інших мобільних пристроях за стандартом WAP також була розроблена і спеціальна мова – WML (Wireless Markup Language).

Спочатку WAP створювався для широкого кола технологій і стандартів бездротового мобільного зв'язку: стільникового, транкового, пейджингового і мікростільникового, а також для підтримки мереж 3G. Даний стандарт інваріантний до операційного ядра, з яким взаємодіє WAP-браузер і розроблявся як відкритий стандарт для бездротової передачі даних, не залежний від постачальників пристроїв і послуг, оптимізований для мобільних телефонів, що мають дисплей з маленьким розширенням, обмеженою пам'яттю і невисокою продуктивністю.

WAP 2.0 – вдосконалена версія WAP, яка використовує урізаний варіант XHTML і CSS. Це дозволяє працювати з WAP 2.0 сайтами за допомогою звичайного браузера на комп'ютері без установки яких-небудь додаткових плагінів.

Сучасні мобільні телефони, тобто ті, що випущені після 2010 року підтримують звичайний HTML та CSS. WAP-майстри також додають на свої сайти JavaScript для динамічності. Використовуються найрізноманітніші додатки CSS для більш приємного та зручного перегляду сайту. Сучасний WAP вже не є тим вапом яким він був 7-8 років тому. Стара мова розмітки WML фактично не використовується. На даний момент розвивається багато різних мобільних порталів із завантаженнями. Веб-хостинги створюють мобільні версії своїх сервісів. Найпопулярніші соціальні мережі мають також полегшені версії. Розвиваються різноманітні CMS для wap-сайтів, як наприклад, найпопулярніші в укрвапі JohnCMS (має українську локалізацію), DCMS (існують українські лангпаки), MobileCMS (лангпаки).

XHTML MP (XHTML Mobile Profile) – мова розмітки у WAP 2.0, розроблена для мобільних пристроїв.

Архітектура WAP аналогічна WWW. У WAP використовується такий же самий спосіб адресації ресурсів і ті ж позначення типів даних. В якості клієнта виступає мобільний пристрій із вбудованим WAP-браузером, запити від якого через WAP-шлюз передаються веб-серверу і відповідь від останнього через нього ж відправляється клієнтові.

В якості серверу може виступати найзвичайніший веб-сервер. В цьому випадку між WAP-шлюзом і сервером використовується протокол HTTP. З метою зменшення



об'єму даних, що передюються, текстові ресурси, які прийшли від сервера, передаються клієнтові в двійковому вигляді.

Мова WML нагадує HTML, але вона орієнтована на пристрої з екраном низького розширення і з невеликим розміром пам'яті. Вся інформація у WML міститься в так званих «деках».

Дек – це мінімальний блок даних, який може бути переданий сервером. У деках знаходяться «карти» (кожна карта обмежена тегами <card>). На екрані пристрою в кожен момент часу відображається тільки одна карта, а користувач може перемикається між ними переходячи по посиланнях. Розмір WML-сторінки повинен знаходитися в межах від 1-4 кілобайт.

Наприклад, WML-сторінка:

```
<?xml version='1.0'?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM/DTD WML 1.1//EN"
'http://www.wapforum.org/DTD/wml_1.1.xml'>
<wml>
<head>
<meta http-equiv=>Cache-Control content=>max-age=0 />
</head>
<card>
<p>Hello world!</p>
</card>
</wml>
```

буде відображатися на дисплеї мобільного телефону таким чином (був використаний програмний емулятор телефону):

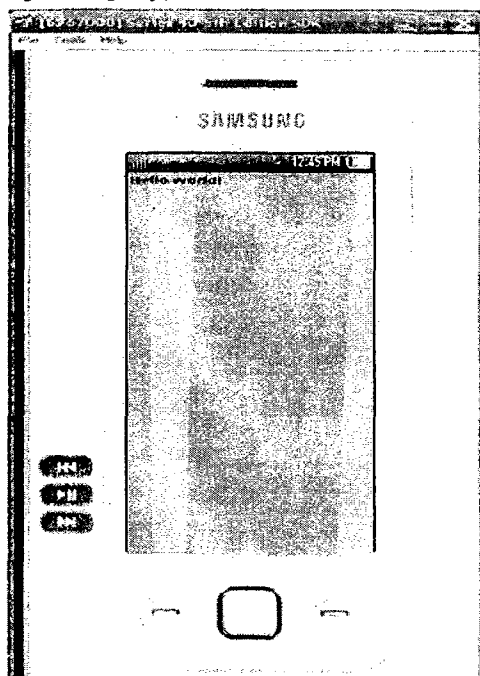


Рис. 5.1.2.1. Результат використання програмного емулятора телефону

Багато мобільних пристроїв можуть відображати документи тільки в WBXML-форматі.

WBXML (WAP Binary XML) – формат компактного бінарного представлення XML. WBXML використовується для передачі через бездротові з'єднання з низькою швидкістю.

Деякі браузері із спеціальним плагінами, дозволяють проглядати WML-сторінки на звичайному комп'ютері.

В даний час відходять від використання WML та більше працюють на XHTML.

.NET Mobile є розширенням Microsoft ASP.NET і Microsoft .NET Framework. По суті, .NET Mobile являє собою набір серверних елементів, що управляють, для форм, орієнтованих на використання в бездротових мобільних пристроях. Ці елементи управління генерують різний код для різних пристроїв на мовах WML, HTML або Compact HTML (cHTML).

Елементи ASP, що управляють .NET Mobile розширюють функції SDP (Smart Device Programmability) і платформу .NET Compact Framework. Вони дозволяють використовувати можливості платформи .NET Compact Framework і середовище розробки Visual Studio .NET для створення мобільних веб-застосунків завдяки можливості доставки даних на різноманітні мобільні пристрої за допомогою технології ASP.NET. Цей підхід дозволяє створити в середовищі Visual Studio .NET єдине мобільне веб-застосування, яке автоматично проводить формування даних для відображення на різноманітних пристроях: мобільних телефонах, пейджерів, смартфонів, Pocket PC і ін. При цьому інтегроване середовище розробки дозволяє створювати мобільні веб-застосунки, просто перетягуючи елементи управління в дизайнер форм.

Система .NET Mobile не встановлює ніяких компонентів на клієнтський пристрій. Для адаптації форматування під конкретні браузері використовується серверна логіка. Вона генерує дані у форматах WML, HTML, і cHTML. Окрім адаптивної генерації веб-сторінок, технологія .NET Mobile надає багатий набір засобів індивідуалізації і розширення, забезпечуючи простий спосіб здійснення підтримки нових пристроїв. Крім того, технологія .NET Mobile дозволяє розробникам управляти представленням даних для конкретного пристрою або класу пристроїв в одній програмній моделі і забезпечує легку реалізацію підтримки нових пристроїв без переробки існуючих веб-застосунків.

## ТЕМА 5.2. РЕАЛІЗАЦІЯ АСИНХРОННОЇ ВЗАЄМОДІЇ ВЕБ-БРАУЗЕРА З ВЕБ-СЕРВЕРОМ ЗА ДОПОМОГОЮ ТЕХНОЛОГІЇ AJAX

### 5.2.1. Виконання асинхронних запитів з JavaScript і Ajax

### 5.2.2. Об'єкт XMLHttpRequest та деталізація технології Ajax

#### 5.2.1. Виконання асинхронних запитів з JavaScript і Ajax

Більшість Web-додатків використовують модель запит-відповідь, в якій отримують від сервера HTML-сторінку повністю. В результаті отримуємо роботу по типу «вперед і назад», що зазвичай складається з натискання кнопки, очікування сервера, натискання іншої кнопки і знову очікування сервера. За допомогою Ajax і об'єкта XMLHttpRequest можна використовувати модель запит-відповідь, яка ніколи не залишає користувачів в очікуванні відповіді сервера.

Зробимо останній маленький огляд перед засвоєнням вихідного коду Web 2.0. Перш за все термін Web 1.0 означає посилання на традиційний Web, в якому використовується абсолютно чітка модель запиту і відповіді. Наприклад, відкриємо сайт Amazon.com і використаємо рядок пошуку. Цей запит містить значно більше, ніж просто список книг; це фактично інша HTML-сторінка. В результаті побачите деяке миготіння і мерехтіння під час перемальовування Web-браузером цієї нової HTML-сторінки. Насправді, можна ясно побачити запит і відповідь, які розмежовані кожною новою сторінкою, яка буде відображена.

Web 2.0 обходиться без цієї дуже помітної процедури «вперед і назад» (значною мірою). Як приклад відвідаємо сайт Google Maps або Flickr. На Google Maps, наприклад, можна повернути карту, наблизити або видалити її з дуже невеликим перемальовуванням. Природно, запити та відповіді виконуються і тут, але це відбувається в фоновому режимі. З точки зору користувача, враження дуже хороше; є таке відчуття, що працюєш з персональним додатком. Ця нова парадигма використовується Web 2.0.

Єдине, про що потрібно подбати – як зробити можливими ці нові взаємодії. Очевидно, що при роботі потрібно виконувати запити і отримувати відповіді, але це перемальовування HTML для кожної операції запит-відповідь і дає відчуття повільного, великовагового Web-інтерфейсу. Тому зрозуміло, що такий новий підхід необхідний, бо він дозволяє виконувати запити і отримувати відповіді, які містять тільки необхідні дані замість цілої HTML-сторінки. Єдиний момент, коли Ви хочете отримати повністю нову HTML-сторінку – це коли Ви хочете, щоб користувач побачив нову сторінку.

Але більшість взаємодій додають деталі: змінюють текстове поле або перекривають дані на існуючих сторінках. У всіх цих випадках підхід Ajax і Web 2.0 уможливають передачу і прийом даних без оновлення HTML-сторінки повністю. І для кожного Web-розробника ця здатність застосування дасть відчуття швидкості, більшої чутливості і буде приводити його до вас знову і знову.

### 3.5.2.2. Об'єкт XMLHttpRequest та деталізація технології Ajax

Для того, щоб використовувати модель запит-відповідь передачі і прийому даних без оновлення HTML-сторінки більш детально познайомимося з об'єктом JavaScript, який має назву XMLHttpRequest. Цей об'єкт є ключем до Web 2.0 і Ajax. Щоб зробити дійсно швидкий огляд, зупинимося лише на декількох методах і властивостях, які будемо використовувати в цьому об'єкті:

- open () : – встановлює новий запит до сервера;
- send () : – передає запит серверу;
- abort () : – припиняє поточний запит;
- readyState : – зберігає поточний стан готовності HTML;
- responseText : – текст, який сервер передав назад як відповідь на запит.

Зверніть увагу, що кожен з цих методів і властивостей відноситься до передачі запиту і працює з відповіддю. Насправді, якщо ви бачили всі методи і властивості XMLHttpRequest, всі вони будуть мати відношення до цієї дуже простої моделі запит-відповідь.

Створимо нову змінну і присвоїмо їй примірник об'єкта XMLHttpRequest. Використаємо ключове слово new з ім'ям об'єкта в JavaScript, як показано в наступному прикладі:

```
<script language=»javascript» type=»text/javascript»>
var request = new XMLHttpRequest ();
</script>
```

В приведеному прикладі створено змінну в JavaScript за допомогою var, присвоєно їй ім'я (наприклад, «request»), і присвоєно їй новий примірник XMLHttpRequest.

**Обробка помилок.** У реальному житті речі можуть псуватися, і код теж не може забезпечити ніякої обробки помилок. Дещо кращий спосіб – створити цей об'єкт і дати йому потерпіти невдачу, якщо що-небудь піде не так. Наприклад, багато старих браузерів (деякі користувачі все ще використовують старі версії Netscape Navigator) не підтримують XMLHttpRequest, і вам потрібно попередити таких користувачів, що щось не вийшло. У наступному прикладі показано, як можна створити цей об'єкт так, що коли щось зламається, він покаже попередження на JavaScript.

Виявляється, що Microsoft підтримує Ajax, але називає свою версію XMLHttpRequest по-іншому. Вірніше він називає цей об'єкт кількома різними іменами. Якщо ви використовуєте більш нову версію Internet Explorer, то повинні використовувати об'єкт Msxml2.XMLHTTP; деякі старі версії використовують Microsoft.XMLHTTP. Ви повинні підтримувати обидва типи об'єктів (без втрати вже наявної підтримки браузерів не від Microsoft). У наступному прикладі наведено вже знайомий код з доданою підтримкою Microsoft.

Приклад. Додавання підтримки для браузерів Microsoft.

```
<script language=»javascript» type=»text/javascript»>
var request = false;
try
{request= new XMLHttpRequest
```

```

    0; } catch (trymicrosoft)
    { try { request
      = new ActiveXObject («Msxml2.XMLHTTP
    «); } catch (othermicrosoft)
    { try { request
      = new ActiveXObject («Microsoft.XMLHTTP
    «); } catch (failed)
    { request =
      false; } } }

    if (! request)
      alert («Error initializing XMLHttpRequest! «);
  </script>

```

Легко загубитися у фігурних дужках, тому розглянемо поетапно всі дії:

1. Створимо нову змінну з ім'ям request і надамо їй значення false. Будемо використовувати false як умову, яка означає, що об'єкт XMLHttpRequest ще не був створений;
2. Додамо блок try/catch. Спробуємо створити об'єкт XMLHttpRequest. Якщо це не вдалося (catch (trymicrosoft)). Спробуємо створити сумісний з Microsoft об'єкт, використовуючи більш нові версії Microsoft (Msxml2.XMLHTTP);
3. Якщо не вдалося зробити з (catch (othermicrosoft)), спробуємо створити сумісний з Microsoft об'єкт, використовуючи старі версії Microsoft (Microsoft.XMLHTTP);
4. Якщо це не вдалося (catch (failed)), упевніться, що request все ще дорівнює false;
5. Перевіримо і дізнаємося, чи дорівнює ще false значення request (якщо все пройшло нормально, цього не станеться). Якщо була проблема (і request дорівнює false), використаємо попередження JavaScript для повідомлення користувачу про виникнення проблеми;
6. Виконаймо ці зміни в своєму коді і спробуємо знову виконати його в Internet Explorer; в результаті побачимо створену нами форму (без повідомлення про помилку).

**Статика проти динаміки.** Розглянемо знову попередні приклади і звернімо увагу, що весь цей код безпосередньо вкладений в середині тегів сценарію. Коли JavaScript закодований подібним чином і не розміщений в середині тіла методу чи функції, він називається статичним JavaScript. Це означає, що код виконується в якийсь момент часу до того, як сторінка відобразиться користувачу (із специфікації не ясно з точністю 100%, коли цей код виконується, і браузері надходять по-різному, але все ж є гарантія, що код виконується до того, як користувачі зможуть взаємодіяти з вашою сторінкою). Зазвичай більшість Ajax-програмістів саме так і створюють об'єкт XMLHttpRequest.

Як було згадано, ми можемо помістити цей код в метод відповідно до наступного прикладу.

Приклад. Переміщення коду створення XMLHttpRequest в метод.

```

<script language=»javascript» type=»text/javascript»>

```

```

var request;

function createRequest ()
{ try { request
  = new XMLHttpRequest
  0; } catch (trymicrosoft)
  { try { request
    = new ActiveXObject
    («Msxml2.XMLHTTP»); } catch (othermicrosoft)
    { try { request
      = new ActiveXObject
      («Microsoft.XMLHTTP»); } catch (failed)
      { request =
        false; } } }

    if (! request)
      alert («Error initializing XMLHttpRequest!
    «); }
  </script>
<script language=»javascript» type=»text/javascript»>

var request;

function createRequest ()
{ try { request
  = new XMLHttpRequest
  0; } catch (trymicrosoft)
  { try { request
    = new ActiveXObject
    («Msxml2.XMLHTTP»); } catch (othermicrosoft)
    { try { request
      = new ActiveXObject
      («Microsoft.XMLHTTP»); } catch (failed)
      { request =
        false; } } }

    if (! request)
      alert («Error initializing XMLHttpRequest!
    «); }
  </script>

```

Поступивши так, Ви повинні викликати цей метод до початку будь-якої роботи з Ajax. Тобто, Ви повинні мати щось схоже на наступний приклад.

Приклад. Використання методу для створення XMLHttpRequest

```
<script language=»javascript» type=»text/javascript»>
```

```
var request;
```

```
function createRequest ()
{try{request
= new XMLHttpRequest
();}catch (trymicrosoft)
{try{request
= new ActiveXObject (
«Msxml2.XMLHTTP»);}catch (othermicrosoft)
{try{request
= new ActiveXObject
(«Microsoft.XMLHTTP»);}catch (failed)
{request=
false;}}}
if (! request)
alert («Error initializing XMLHttpRequest!
»);}
```

```
function getCustomerInfo ()
{createRequest();
// Зробити щось із змінною
request}
</ script>
```

Єдине зауваження з приводу цього коду (і причина того, чому багато Ажах-програмістів не використовують такий підхід) – він затримує виведення повідомлення про помилку. Уявіть, що є складна форма з 10 або 15 полями, рамками вибору варіантів і т.п., і Ви активізуєте якийсь Ажах-код при введенні користувачем тексту в поле 14 (внизу форми). Активізується `getCustomerInfo ()`, який намагається створити об'єкт `XMLHttpRequest`, і (для даного прикладу) зазнає невдачі. Користувачеві виводиться попередження, повідомляє про те (у багатьох словах), що він не може користуватися цією програмою. Але користувач вже витратив час для заповнення форми. Це досить неприємно і ця неприємність не входить в число тих речей, через які користувач захотів би повернутися на Ваш сайт.

У разі використання статичного JavaScript, користувач отримає повідомлення про помилку, як тільки зайде на Вашу сторінку. Це так само неприємно? Можливо. Можна роздратувати користувача тим, що Ваш Web-додаток не працює в його браузері. Однак це безперечно краще, ніж видати цю ж помилку після 10 хвилин введення користувачем інформації. Тільки з цієї причини рекомендується встановлювати Ваш код статично і заздалегідь попереджати користувачів про можливі проблеми.

**Передача запитів з XMLHttpRequest.** Після того як Ви створили об'єкт `request`, можете почати цикл запит/відповідь. Згадайте, що єдиною метою об'єкта

`XMLHttpRequest` є надання Вам можливості посилати запити та отримувати відповіді. Все інше (зміна користувацького інтерфейсу, заміна зображень, навіть інтерпретація переданих сервером даних) – це робота JavaScript, CSS або іншого коду на Ваших сторінках. З готовим до використання `XMLHttpRequest` Ви можете зараз послати запит серверу.

**Модель безпеки Ажах.** Ажах має модель безпеки по типу «пісочниці». В результаті ваш Ажах-код (і, зокрема, об'єкт `XMLHttpRequest`) може посилати запити тільки того домену, на якому виконується. Код, що виконується на вашому локальному комп'ютері, може виконувати запити тільки до серверних сценаріїв на вашому локальному комп'ютері. Якщо ваш Ажах-код працює для прикладу на `www.breakneckpizza.com`, він повинен посилати запити до сценаріїв, які працюють на `www.breakneckpizza.com`.

**Асинхронність.** Розглянемо написання і використання асинхронного коду, де потрібно мати уявлення про те, чому останній параметр в `open ()` настільки важливий. У звичайній моделі запит/відповідь клієнт (браузер або код, що виконується на вашій локальній машині) надсилає запит серверу. Цей запит є синхронним; іншими словами, клієнт чекає відповіді від сервера. Поки клієнт чекає, Ви зазвичай отримуєте одну з декількох форм повідомлення про те, що ви чекаєте:

- пісочний годинник (головним чином у Windows);
- крутиться м'яч (зазвичай на Mac-комп'ютерах).
- додаток завмирає і іноді змінюється курсор.

Саме це дає відчуття, що Web-додатки незграбні або повільні – відсутність реальної інтерактивності. Коли Ви натискаєте кнопку, то застосування головним чином стає непридатним до використання, поки не буде отримана відповідь на щойно переданий запит. Якщо Ви надішлете запит, що вимагає інтенсивних обчислень на сервері, час очікування може бути значним (по крайній мірі, для сьогоденного світу, що використовує багато процесорів, DSL і не звик чекати).

Асинхронний запит не чекає відповіді від сервера. Ви посилаєте запит, і Ваш додаток продовжує роботу. Користувачі можуть вводити дані у Web-форму, натискати інші кнопки, навіть піти з форми. Немає крутіння м'яча або обертового пісочного годинника, і немає великих завмирань програми. Сервер мовчки відповідає на запит, і коли завершується передача відповіді, він дає знати про це ініціатору запиту (способами, про які Ви незабаром дізнаєтеся). В результаті ми маємо програму, що не виглядає незграбною або повільною, а, навпаки, чутливою, інтерактивною і швидкою. Це тільки один компонент Web 2.0, але це дуже важливий компонент. Будь-які важливі GUI-компоненти і парадигми Web-дизайну не можуть подолати повільну, синхронну модель запит/відповідь.

**Передача запиту.** Після настройки запиту з використанням `open ()` Ви готові до передачі запиту. Існує метод для передачі запиту названий більш правильно, ніж `open ()`; його просто назвали `send ()`.

`Send ()` приймає тільки один параметр – вміст для передачі. Але перш ніж Ви серйозно задумаетесь про це, згадайте, що Ви вже передали дані самому URL:

```
var url = «/ cgi-local/lookupCustomer.php?Phone =» + escape (phone);
```

І хоча Ви можете передати дані з використанням `send ()`, Ви можете також передати їх через сам URL. В дійсності, у GET-запити (які складатимуть до 80% типового використання Ajax) набагато легше передавати дані в URL. Коли Ви починаєте передавати захищену інформацію або XML, потрібно розглянути передачу вмісту через `send ()`. Якщо Вам не треба передавати дані через `send ()`, встановіть аргумент у значення `null`. Щоб надіслати запит в прикладі, використовуваному в даній темі, це саме те, що нам потрібно (див. приклад далі).

Приклад. Передача запиту.

```
function getCustomerInfo ()
{varphone = document.getElementById («phone»). Value;
var url = «/ cgi-local/lookupCustomer.php? Phone =>» + escape (phone);
request.open («GET», url, true);
request.send
(null);}
```

Вказівка методу зворотного виклику. До цих пір Ви зробили дуже небагато такого, що було б чимось новим, революційним чи асинхронним. Маленьке ключове слово «true» у методі `open ()` встановлює асинхронний запит. На стороні сервера код пишеться на Java-сервлетах і JSP, PHP або Perl. Так в чому ж великий секрет Ajax або Web 2.0? Секрет крутиться навколо простої властивості `XMLHttpRequest` з назвою `onreadystatechange`.

Перш за все, Ви повинні розуміти процес, який Ви створили в цьому коді (переглянемо попередній приклад при необхідності). Запит настраюється і потім передається. Крім того, оскільки це асинхронний процес, JavaScript-метод (`getCustomerInfo ()` в прикладі) не буде чекати сервер. Тому код буде продовжувати виконуватися; в даному випадку це означає, що відбудеться вихід з методу і керування повернеться у форму. Користувач може продовжити введення інформації, і додаток не буде чекати сервера.

При цьому виникає цікаве питання: що відбувається, коли сервер завершить обробку запиту? Насправді сервер повинен мати інструкцію якогось типу про те, що робити, коли він завершить обробку запиту, переданого йому об'єктом `XMLHttpRequest`.

**Посилання на функцію в JavaScript.** JavaScript є слабо універсальна мова і з допомогою змінної Ви можете посилатися майже на все, що завгодно. Тобто, якщо Ви оголосили функцію з назвою `updatePage ()`, JavaScript також розглядає ім'я цієї функції як змінну. Іншими словами, Ви можете посилатися на функцію у Вашому коді як на змінну з назвою `updatePage`.

Тут виходить на сцену властивість `onreadystatechange`. Ця властивість дозволяє Вам вказати метод зворотного виклику (`callback`). Метод зворотного виклику дозволяє серверу виконати зворотний виклик в коді Web-сторінки. Це додає в сервер рівень контролю; коли сервер завершує обробку запиту, він шукає об'єкт `XMLHttpRequest` і, зокрема, його властивість `onreadystatechange`. Потім активізується будь-який метод, зазначений у цій властивості. Він називається методом зворотного виклику, тому що сервер ініціює виклик у зворотному напрямку (у Web-сторінці) незалежно від того, що відбувається в самій Web-сторінці. Наприклад, він може викликати цей метод тоді,

коли користувач сидить в кріслі, не чіпаючи клавіатуру, а проте, він може викликати цей метод і тоді, коли користувач вводить інформацію, переміщує мишку, виконує скролінг сторінки, натискає кнопку, при цьому не важливо, що робить користувач.

Ось тут і виявляє себе асинхронність: користувач працює з формою на одному рівні, тоді як на іншому рівні сервер відповідає на запит і, потім, активізує метод зворотного виклику, зазначений у властивості `onreadystatechange`. Тому Ви повинні вказати цей метод у Вашому коді, як показано в наступному прикладі.

Приклад. Установка методу зворотного виклику.

```
function getCustomerInfo ()
{varphone = document.getElementById («phone»). Value;
var url = «/ cgi-local/lookupCustomer.php? Phone =>» + escape (phone);
request.open («GET», url, true);
request.onreadystatechange = updatePage;
request.send
(null);}
```

Зверніть особливу увагу на те, де в коді встановлюється ця властивість – перед викликом `send ()`. Ви повинні встановити цю властивість перед передачею запиту, тому сервер може переглянути його після завершення обробки запиту. Все, що нам залишилося – написати `updatePage ()`, що і розглядатиметься в кінці теми.

**Стани готовності HTTP.** Раніше згадувалося, що сервер, закінчивши обробку запиту, шукає, який метод викликати у властивості `onreadystatechange` об'єкта `XMLHttpRequest`. Але, насправді, він викликає цей метод кожен раз, коли змінюється стан готовності HTTP. Стан готовності HTTP відображає стан або статус запиту. Воно використовується для визначення того, чи розпочато запит, чи приймається відповідь, або чи завершений цикл запит/відповідь. Цей стан допомагає також при визначенні того, чи можна читати текст або дані, які сервер міг надати у відповіді. Для цього розглянемо п'ять станів готовності у Ajax-додатках:

0:Запит не ініціалізовано (перед викликом `open ()`);

1:Запит ініціалізовано, але він не був переданий (перед викликом `send()`);

2:Запит був переданий і обробляється (на даному етапі Ви можете зазвичай отримати заголовки вмісту з відповіді);

3: Запит оброблюється; часто у відповіді доступні деякі часткові дані, але сервер не закінчив відповідь;

4: Відповідь завершена; Ви можете отримати відповідь від сервера і використовувати її.

Як і майже всі крос-браузерні проблеми, ці стани готовності використовуються дещо непослідовно. Ви могли б очікувати, що стани завжди змінюються від 0 до 1, від 1 до 2, від 2 до 3, від 3 до 4, але на практиці це рідкісний випадок. Деякі браузери ніколи не видають 0 або 1, а відразу перестрибують до 2, потім до 3 і 4. Інші браузери видають всі стани. Але деякі видають стан 1 кілька разів. Як ми бачили в останньому прикладі, сервер викликав `updatePage ()` кілька разів і кожен виклик приводив до появи спливаючого вікна попередження.

Для Ajax-програмування єдиний стан, з яким Ви повинні мати справу – це стан готовності 4, яке вказує на те, що відповідь сервера завершено і можна перевірити відповідні дані та використовувати їх. Для врахування цього перший рядок у Вашому методі зворотного виклику повинен бути таким, як показано в наступному прикладі.

Приклад. Перевірка стану готовності.

```
function updatePage ()
{if(request.readyState == 4)
  alert («Server is
done!»);}
```

Ця змінна перевіряє, чи дійсно сервер завершив процес. Спробуймо попрацювати з цією версією коду і ми повинні отримати тільки одне попередження, як і повинно бути.

Коди стану http. Незважаючи на явний успіх коду в попередньому прикладі, є одна проблема – що, коли сервер відповів на Ваш запит і завершив обробку, але повертає помилку? Згадайте, ваш серверний код не знає, чи був викликаний він Ajax-додатком, JSP-сторінкою, звичайною HTML-формою або будь-яким іншим типом коду; він має тільки традиційні Web-методи оповіщення. І у Web-світі HTTP-коди можуть мати справу з різними речами, які можуть виникнути у запиті.

Наприклад, ми задали запит до URL, ввели URL неправильно і отримали код помилки 404, який вказує, що сторінка відсутня. Це один з багатьох кодів стану, які можуть отримувати HTTP-запити. Також поширеними є коди 403 і 401, які вказують на те, що запитуються захищені або заборонені дані. У кожному з цих випадків є коди, які приходять у відповіді. Іншими словами, сервер виконав запит (означає, що стан готовності HTTP рівний 4), але, можливо, не повертає дані, очікувані клієнтом.

Тобто, крім стану готовності ми повинні також перевіряти HTTP-стан. Нам підходить код стану 200, який просто означає, що все в порядку. Зі станом готовності 4 і кодом стану 200 ми готові обробити серверні дані, і ці дані повинні бути тими, які ми запитували (і немає помилок або іншої проблемної інформації). Додайте ще одну перевірку у наш метод зворотного виклику, як показано в прикладі далі.

Приклад. Перевірка коду стану http.

```
function updatePage ()
{if(request.readyState == 4)
  if (request.status == 200)
    alert («Server is
done!»);}
```

Для більш стійкої обробки помилок (з мінімальним ускладненням коду) Ви можете додати перевірку або дві для інших кодів стану; Подивіться на модифіковану версію updatePage () в прикладі нижче.

Приклад. Додавання деякої обробки помилок.

```
function updatePage ()
{if(request.readyState == 4)
  if (request.status == 200)
    alert («Server is done!»);}
```

```
Else if (request.status == 404)
  alert (« Request URL does not exist »);
else
  alert (« Error: status code is «+
request.status»);}
```

Тепер змінимо URL в getCustomerInfo () в неіснуючий URL і подивимось, що відбудеться. В результаті побачимо попередження, яке говорить про те, що запитаний нами URL не існує, що нам і потрібно. Дуже трудомістко обробляти всі помилки, але ця проста зміна вирішить 80% проблем, які можуть виникнути в звичайному Web-додатку.

Читання тексту відповіді. Тепер, коли ми переконані, що запит повністю оброблений (через стан готовності) і сервер повернув нормальну позитивну відповідь (через код стану), можна, нарешті, зайнятися даними, переданими від сервера. Вони зручно збережені у властивості responseText об'єкту XMLHttpRequest.

Подробиці про текст в responseText, в термінах формату або довжини, навмисно залишені невизначеними. Це дозволяє серверу встановити цей текст віртуально під все, що завгодно. Наприклад, один сценарій може повернути розділені комами значення, інший – розділені вертикальною рисою значення (символ «|»), а третій може повернути один довгий рядок тексту. Це вирішує сервер.

У прикладі нижче сервер повертає останнє замовлення клієнта та його адресу, розділену символом вертикальної риски. Замовлення та адреса використовуються для установки елементів форми; в наступному прикладі наведено код, що оновлює дисплей.

Приклад. Робота з відповіддю сервера.

```
function updatePage ()
{if(request.readyState == 4)
  {if(request.status == 200)
    {varresponse = request.responseText.split («|»);
    document.getElementById («order» ). value = response [0];
    document.getElementById («address»). innerHTML =
      response [1]. replace (/ \ n / g, «<br
/>»);}else
    alert («status is» +
request.status);}}
```

Спочатку витягується responseText і розділяється на частини по символам вертикальної риски («|») за допомогою методу JavaScript split (). Одержаний масив значень присвоюється масиву response. Перше значення (останнє замовлення клієнта) знаходиться в response [0] і встановлюється як значення поля з ID «order.» Друге значення масиву, response [1], є адресою клієнта і вимагає дещо більшої обробки. Оскільки рядки в адресі розділені звичайними роздільниками рядків (символ «\ n»), то потрібно замінити їх XHTML-роздільниками рядків <br />. Це виконується за допомогою функції replace () з регулярним виразом. Нарешті, модифікований текст встановлюється як

inner HTML-елементу div у формі HTML. В результаті цього форма несподівано оновлюється користувацькою інформацією.

Перед вкладанням розглянемо ще одну важливу властивість XMLHttpRequest – responseXML. Ця властивість містить XML-відповідь у випадку, якщо сервер вирішив відповісти в XML-форматі. Робота з XML відповіддю суттєво відрізняється від роботи зі звичайним неформатованим текстом і включає роботу з синтаксичним аналізатором, Document Object Model (DOM). Поки ж, оскільки responseXML зазвичай розглядається при обговоренні responseText, ми просто згадаємо його. Для багатьох простих Ajax-додатків responseText – це перша необхідність.

**Удосконалені запити та відповіді в Ajax.** Для багатьох Web-розробників виконання простих запитів і отримання простих відповідей – це все, що коли-небудь може їм знадобитися, але для розробників, які хочуть освоїти Ajax, необхідно повне розуміння кодів стану HTTP, станів готовності та об'єкта XMLHttpRequest. Тут буде розказано про різні коди стану і ми розглянемо, як браузері їх сприймають, а також про маловикористовувані HTTP-запити, які ви можете застосовувати з Ajax.

Зараз нам надана вступна частина в об'єкт XMLHttpRequest, центральний елемент Ajax-додатків, який керує запитом до серверного додатку або сценарію, а також працює з повернутими даними з цього серверного компоненту. Кожен Ajax-додаток використовує об'єкт XMLHttpRequest, тому Ви повинні бути дуже близько знайомі з ним, для того щоб змусити Ajax-додатки виконуватися на високому рівні.

Далі ми вийдемо за межі в попередньому матеріалі представлених основ і сконцентруємося більш детально на трьох ключових частинах цього об'єкту:

- стан готовності http;
- код стану http;
- типи запитів, які Ви можете зробити.

Кожен з них є, як правило, частиною структури запиту; в результаті про них відомо мало подробиць. Однак Ви повинні вільно розбиратися в станах готовності, кодах стану та запитах, якщо хочете не просто пограти в Ajax-програмування, а зробити більше. Коли у Вашому додатку щось йде не так (все завжди йде не так), знання кодів стану, способів передачі HEAD-запиту або того, що означає код стану 400, може вилитися в різницю між п'ятьма хвилинами налагодження і п'ятьма годинами очікування.

**XMLHttpRequest або XMLHttpRequest.** Microsoft™ і Internet Explorer використовують об'єкт XMLHttpRequest замість об'єкту XMLHttpRequest, що використовується в браузерах Mozilla, Opera, Safari і більшості інших браузерів. Заради простоти будемо посилалися на обидва цих об'єкти просто як XMLHttpRequest. Це відповідає загальноприйнятій практиці у Web і також збігається з намірами Microsoft використовувати XMLHttpRequest в якості імені об'єкта запиту в Internet Explorer

Спочатку розглянемо стани готовності HTTP. Потрібно пам'ятати з попереднього згаданого, що об'єкт XMLHttpRequest має властивість readyState. Ця властивість засвідчує, що сервер завершив запит і, зазвичай, функція зворотного виклику використовує дані від серверу для оновлення Web-форми або сторінки. Це відображено у наступному прикладі.

Приклад. Робота з відповіддю сервера у функції зворотного виклику.

```
function updatePage ()
```

```
{if(request.readyState == 4)
  {if(request.status == 200)
    {var response = request.responseText.split («|»);
    document.getElementById («order»). value = response [0];
    document.getElementById («address»). innerHTML =
      response [1]. replace (/ \ n / g, «<br
    />»);}else
      alert (« status is «+
    request.status);}}
```

Це безперечно найбільш загальне (і найбільш просте) використання станів готовності. Як Ви, можливо, здогадалися по числу «4», існує і кілька інших станів готовності:

- 0: Запит не ініційований (перед викликом open ());
- 1: Запит ініційований, але не був переданий (перед викликом send ());
- 2: Запит був переданий і обробляється (на даному етапі Ви можете звичайно отримати заголовки вмісту з відповіді);
- 3: Запит обробляється; часто у відповіді доступні деякі часткові дані, але сервер не закінчив свою відповідь;
- 4: Відповідь завершено, і Ви можете отримати відповідь серверу і використовувати його.

Якщо Ви хочете вийти за рамки основ Ajax-програмування, то повинні знати не тільки ці стани, але і коли вони виникають, а також як Ви можете використовувати їх. Перше і найголовніше – ми повинні вивчити, на якому етапі запиту виникає кожен стан. На жаль, це не така вже інтуїтивна річ, і є спеціальні випадки.

**Приховані значення станів готовності.** Перший стан готовності, що позначається значенням властивості readyState рівним 0 (readyState == 0), являє неініціалізований запит. Як тільки Ви викличете метод open () вашого об'єкта запиту, ця властивість встановлюється в 1. Оскільки Ви майже завжди викликаєте open () відразу після ініціалізації вашого запиту, рідко можна побачити readyState == 0. Більше того, неініціалізований стан готовності досить незручний в реальних додатках.

Тим не менш, в інтересах повноти викладу поглянемо на наступний приклад, в якому показано, як отримати стан готовності, коли він встановлений в 0.

Приклад. Отримання стану готовності 0.

```
function getSalesData ()
  {// Створити об'єкт request
  createRequest ();
  alert («Ready state is:» + request.readyState);

  // Налаштувати (ініціалізувати) запит
  var url = «/ boards / servlet / UpdateBoardSales» ;
  request.open («GET», url, true);
  request.onreadystatechange = updatePage;
  request.send
  (null);}
```



У цьому простому прикладі `getSalesData ()` – це функція, яку викликає Ваша Web-сторінка для запуску запиту (при натисканні кнопки). Зверніть увагу, що Ви повинні перевірити стан готовності перед викликом `open ()`.

**Коли 0 дорівнює 4.** В тому випадку, коли кілька JavaScript-функцій використовують один і той же об'єкт запиту, перевірка на рівності 0 стану готовності (для гарантії того, що об'єкт не використовується) може викликати проблеми. Оскільки `readyState == 4` вказує на завершеність запиту, то Ви часто зможете знайти невживані об'єкти, що мають стан готовності все ще рівні 4, тобто дані від сервера були використані, але нічого після цього не сталося для скидання стану готовності. Існує функція, яка скидає стан готовності об'єкта запиту – це функція `abort ()`, але вона призначена не для цього. Якщо потрібні декілька функцій, можливо, кращим варіантом буде створення і використання об'єкта запиту для кожної функції, а не загальне використання одного об'єкта. Очевидно, це не дуже добре, оскільки існує дуже мало випадків, коли Ви повинні перевіряти, що `open ()` не була викликана. Єдиним використанням цього стану готовності в «майже реальному» Аях-програмуванні буде ситуація, коли Ви виконете кілька запитів, використовуючи один і той же об'єкт `XMLHttpRequest` з декількох функцій. В цій ситуації (досить незвичайній) Ви, можливо, захочете переконатися, що об'єкт запиту знаходиться в неініціалізованому стані (`readyState == 0`) перед виконанням нових запитів. Це, по суті, гарантує, що інша функція не використовує об'єкт в цей же час.

**Огляд стану готовності виконання запиту.** Від стану готовності 0 ваш об'єкт запиту повинен проходити через кожен інший стан в звичайному запиті і відповіді, та, врешті-решт, закінчити на стані 4. Ось чому Ви бачите рядок коду `if (request.readyState == 4)` у більшості функцій зворотного виклику вона гарантує, що сервер закінчив свою роботу із запитом і можна без побоювань оновити Web-сторінку або виконати дію, засновану на отриманих від сервера даних.

Побачити цей процес під час його протікання є тривіальною задачею. Замість виконання тільки при стані готовності рівному 4 коду у Вашій функції зворотного виклику просто виводимо стан готовності кожного разу при виклику функції. Приклад наведено нижче.

Приклад. Перевірка стану готовності.

```
function updatePage ()
{
  // Відобразити поточний стан готовності
  alert («updatePage () called with ready state of» +
    request.readyState);
}
```

Ви повинні створити функцію для виклику з Web-сторінки і передати в ній запит серверному компоненту (просто така ж функція, як і наведена в попередньому прикладі). Переконайтеся в тому, що при налаштуванні Вашого запиту Ви встановили функцію зворотного виклику в `updatePage ()`; для цього встановіть властивість `onreadystatechange` Вашого об'єкта запиту в `updatePage ()`.

Цей код є чудовою ілюстрацією того, що насправді означає `onreadystatechange` – щоразу при зміні стану готовності запиту викликається `updatePage ()`, і Ви бачите попередження. Спробуйте цей код виконати самостійно. Помістимо його на Вашу Web-

сторінку і активізуємо Ваш обробник подій (натиснемо кнопку, вийдемо з поля або використаємо будь-який метод, який Ви встановили для ініціювання запиту). Ваша функція зворотного виклику виконається кілька разів (кожен раз при зміні стану готовності запиту), і Ви побачите попередження для кожного стану. Це найкращий спосіб слідувати за запитом через кожний його стан.

**Несумісність браузерів.** Отримавши загальне поняття про цей процес, спробуйте звернутися до Web-сторінки з кількох різних браузерів. Ви повинні помітити деяку несумісність в тому, як обробляються ці стани готовності. Наприклад, у Firefox Ви побачите наступні стани готовності:

1  
2  
3  
4

Це не повинно бути сюрпризом, оскільки тут представлений кожен стан запиту. Однак, якщо Ви звернетесь до цього ж додатку в Safari, то повинні побачити:

2  
3  
4

Safari насправді пропускає перший стан, і немає здорового пояснення, чому; просто Safari працює таким чином. Це також виявляє важливий момент. В той час як використання значення стану запиту рівне 4 перед використанням даних від сервера є непоганою ідеєю, а написання коду, що залежить від кожного проміжного стану готовності, безсумнівно, шлях до отримання різних результатів у різних браузерах.

Наприклад, при використанні Opera відображення станів готовності стає навіть ще гірше:

3  
4

І останнє, але найважливіше – Internet Explorer відображає наступні стани:

1  
2  
3  
4

Якщо із запитом є проблеми, це найперше місце, де слід їх шукати. Додайте висновок попередження, щоб побачити стан готовності запиту і переконатися в тому, що все працює нормально. Ще краще – протестуйте і в Internet Explorer, і в Firefox. Як наслідок Ви отримаєте всі чотири стани готовності і зможете перевірити кожен етап запиту.

Тепер подивимося з боку відповіді. Розібравшись з різними станами готовності, що відбуваються під час запиту, ми готові дослідити іншу важливу частину об'єкта `XMLHttpRequest` – властивість `responseText`. Як тільки сервер завершить обробку за-

питу, він розміщує усі дані, необхідні для відповіді на запит, у властивість `responseText` запиту. Після цього Ваша функція зворотного виклику може використовувати ці дані, як показано в наступному прикладі.

Приклад. Використання відповіді від сервера.

```
function updatePage ()
{if(request.readyState == 4)
{var newTotal = request.responseText;
var totalSoldEl = document.getElementById («total-sold»);
var netProfitEl = document.getElementById («net-profit »);
replaceText (totalSoldEl, newTotal);

/* Визначити новий чистий прибуток */
var boardCostEl = document.getElementById (« board-cost »);
var boardCost = getText (boardCostEl);
var manCostEl = document.getElementById (« man -cost »);
var manCost = getText (manCostEl);
var profitPerBoard = boardCost - manCost;
var netProfit = profitPerBoard * newTotal;

/* Оновити чистий прибуток на формі продажу */
netProfit = Math.round (netProfit * 100) / 100;
replaceText (netProfitEl,
netProfit);}}
```

Даний приклад більш складніший за попередні, але на початку обидва перевіряють стан готовності і потім витягують значення (або значення) з властивості `responseText`.

**Перегляд текстової відповіді під час запиту.** Аналогічним станом готовності значення властивості `responseText` є зміна на всьому життєвому циклі запиту. Щоб побачити це в дії, використаємо код, аналогічний наведеному в наступному прикладі, для тестування текстової відповіді і стану готовності.

Приклад. Тестування властивості `responseText`

```
function updatePage ()
{// Відобразити поточний стан готовності
alert («updatePage () called with ready state of» + request.readyState +
«and a response text of ‘» + request.responseText +
«’»);}}
```

Тепер відкриємо ваш Web-додаток в браузері і активуємо запит. Для отримання найбільшої інформації з цього коду використаємо або Firefox, або Internet Explorer, оскільки обидва обробляють всі можливі стани готовності під час запиту. Наприклад, коли стан готовності дорівнює 2, властивість `responseText` не визначено, де Ви повинні побачити помилку, якщо відкрилася консоль JavaScript. А при стані готовності 3 сервер помістить значення у властивість `responseText`, по крайній мірі, в цьому прикладі. Тут Ви побачите, що Ваші відповіді при стані готовності 3 відрізняються від

сценарію до сценарію, від сервера до сервера і від браузера до браузера. Але все ж цей підхід продовжує залишатися надзвичайно корисним при налагодженні даного застосування.

**Отримання надійних даних.** У всій документації та специфікаціях стверджується, що тільки при стані готовності 4 дані можна використовувати безпечно. Якщо ми напишемо код, що залежить від завершеності даних при стані готовності 3, майже гарантується, що настане момент, коли дані будуть не повні.

Кращою ідеєю є надання певного роду зворотного зв'язку з користувачем, коли стан готовності дорівнює 3 та відобразити, що відповідь на підході. У той час, як використання такої функції як `alert ()`, очевидно, погана ідея, використання Ajax і блокування користувача діалоговим вікном зрозуміти досить важко. Для цього ми могли б оновити поле на Вашій формі або сторінці при зміні стану готовності. Наприклад, спробуйте встановити довжину індикатора ходу процесу в 25% при рівності стану готовності 1, 50% при 2, 75% при 3 і 100% (завершено) при 4.

Такий підхід зрозуміліший, але залежить він від використовуваного браузера. У Opera Ви ніколи не отримаєте перших двох станів готовності, а Safari пропустить 1.

**Коди стану http.** Маючи в багажі знань Ваш досвід Ajax-програмування стану готовності та відповіді сервера, Ви готові додати ще один рівень удосконалення Ajax-додатків – роботу з кодами стану HTTP. Ці коди не є чимось новим для Ajax. Вони існують у Web з часів його появи. Ви, ймовірно, вже бачили деякі з них у вашому браузері:

**401:Unauthorized (Не авторизований)**

**403:Forbidden (Заборонено)**

**404:Not Found (Не знайдено)**

Щоб додати ще один рівень керованості та оперативності (і особливо більш надійної обробки помилок) до Ваших Ajax-додатків необхідно перевіряти коди стану в запиті і відповіді відповідно.

**200: Все OK**

У багатьох Ajax-додатках Ви побачите функцію зворотного виклику, яка перевіряє стан готовності і продовжує роботу з даними відповіді сервера, як в наступному прикладі.

Приклад. Функція зворотного виклику, що ігнорує код стану.

```
function updatePage ()
{if(request.readyState == 4)
{var response = request.responseText.split («<»);
document.getElementById («order»). Value = response [0 ];
document.getElementById («address»). innerHTML =
response [1]. replace (/ \ n / g, «<br
/>»);}}
```

Це недалекоглядний і схильний до помилок підхід Ajax-програмування. Якщо сценарій вимагає аутентифікації, а Ваш запит не надає коректних даних, сервер по-

верне код помилки 403 або 401. Проте стан готовності буде встановлено в 4, оскільки сервер відповів на запит (навіть якщо відповідь на Ваш запит не така, якої ви хотіли або чекали). В результаті користувач не отримає правильних даних і може навіть отримати неприємну помилку, коли Ваш JavaScript-код спробує використовувати неіснуючі серверні дані.

Гарантувати те, що сервер не тільки завершив обробку запиту, а й повернув код стану «Все в порядку», вимагає мінімальних зусиль. Цей код дорівнює «200» і повертається у властивості `status` об'єкта `XMLHttpRequest`. Отже, додамо додатковий рядок у Вашу функцію зворотного виклику, як показано в наступному прикладі.

Приклад. Перевірка коду стану.

```
function updatePage ()
{if(request.readyState == 4)
  {if(request.status == 200)
    {varresponse = request.responseText.split («|»);
    document.getElementById («order»). value = response [0];
    document.getElementById («address»). innerHTML =
      response [1]. replace (/ \ n / g, «<br
    />»);}else
      alert («status is» + request .
status);}}
```

З цими невеликим числом додаткових рядків коду Ви можете бути впевнені, що якщо щось піде не так, Ваші користувачі отримають корисне повідомлення про помилку замість сторінки з перекрученими даними без всяких пояснень.

**Переадресація і перенаправлення.** Перед детальним розглядом помилок є сенс поговорити про те, про що Ви, можливо, не повинні турбуватися при використанні Ajax-переадресації. У кодах стану HTTP є сімейство кодів стану 300, що включають:

**301:Moved permanently** (переміщений постійно)

**302:Found** (Знайдений) – запит переадресовано на інший URL/URI

**305:Use Proxy** (Використовувати проксі) – запит повинен використовувати проксі для доступу до затребуваних ресурсів

Ajax-програмісти, можливо, не турбуються про переадресації з двох причин: перш за все, Ajax-додатки майже завжди пишуться для конкретного серверного сценарію, сервілета або програми. Зникнення або переміщення в інше місце цього компонента без Вашого, Ajax-програміста, відома зустрічається досить рідко. Тому майже завжди Ви знаєте, що ресурс перемістився (тому що Ви його перемістили, або він вже був переміщений), змінюєте URL у Вашому запиті і ніколи не стикаєтеся з такого роду проблемами. І ще одна, навіть більш важлива, причина: Ajax-додатки та запити виконуються в «пісочниці» безпеки. Це означає, що доменом, обслуговуючим Web-сторінку, з якої виконуються Ajax-запити, є домен, який повинен їм відповідати. Тому Web-сторінка, яку обслуговує `ebay.com`, не може виконати Ajax-запит сценарієм, що виконує на `amazon.com`; Ajax-додатки `ibm.com` не можуть виконувати запити сервілетам, які працюють на `netbeans.org`.

В результаті Ваші запити не можуть бути переадресовані на інший сервер без генерування помилки захисту. У цих випадках Ви зовсім не отримаєте код стану. Ви просто отримаєте помилку JavaScript в консолі налагодження. Тому, думаючи про різноманіття кодів стану, Ви можете майже зовсім ігнорувати коди переадресації.

## Контрольні запитання

1. Що таке технологія AJAX?
2. Який запит до серверу називається позаполосним зверненням?
3. З появою чого пов'язана популярність AJAX?
4. Що таке JSON?
5. На яких структурах даних будується JSON?
6. З яких елементів складається інфраструктура ASP.NET 2.0 AJAX?
7. WAP- це ...?
8. Що називається WAP 2.0?
9. Яке означення XHTML MP?
10. Яке визначення слова дек?
11. Який розмір WML-сторінки?
12. Що таке WBXML. Для чого він використовується?
13. На яких веб-стандартах заснований AJAX?
14. У яких серверах підтримується XMLHttpRequest?
15. Як виглядає загальна послідовність роботи .NET Mobile?
16. Які є найбільш поширені елементи управління формами для мобільних пристроїв?
17. Як виглядає загальна послідовність розробки веб-застосування з використанням .NET Mobile?

## Тести для закріплення матеріалу

### 1. AJAX розшифровується як ...

- A) *Asynchronous JavaScript and XML*;
- Б) *AsynsObject JavaScript and XML*;
- В) *Asynchronous JavaScript and XJSON*;
- Г) *Asynchronous JavaSpirt and XML*.

### 2. Популярність AJAX пов'язана з появою сервісу ... в 2005 році.

- A) *XMLH*;
- Б) *Internet Explorer*;
- В) *Google Suggest*;
- Г) *Firefox*.

### 3. Об'єкт ... є компактною об'єктною моделлю для відправки сценарієм звернень HTTP в обхід браузеру.

- A) *XMLHttpRequest*;
- Б) *XML*;
- В) *HTTP*;
- Г) *JSON*.

### 4. Як формат передачі даних зазвичай використовуються ... або XML.

- A) *XmlHttpRequest*;
- Б) *JSON*;
- В) *Object*;
- Г) *JavaScript*.

### 5. Як розшифровується JSON?

- A) *Json Span Object Natyrate*;
- Б) *JavaScript Object Naryrate*;
- В) *JavaSafari Opera Netscape*;
- Г) *JavaScript Object Notation*.

### 6. JSON будується на двох структурах даних:

- A) набір пар ім'я/значення, пронумерований набір значень;
- Б) структура веб-технологій та веб-дизайну;
- В) бібліотека клієнтських сценаріїв, набір серверних розширень;
- Г) пронумерований набір значень, бібліотека клієнтських сценаріїв.

### 7. Як розшифровується WAP?

- A) *Wireless Application Protocol*;
- Б) *Warcyp Application Protocol*;
- В) *Warcyp Application Program*;

Г) *Warcup Ajacs Program*.

**8. Як розшифровується мова для мобільних телефонів WML?**

А) *Wireless Maper Lfoot*;

Б) *Wireless Mobody Laption*;

В) *Wireless Markup Language*;

Г) *Width Meploy Language*.

**9. Що таке XHTML MP?**

А) мова розмітки у WAP;

Б) мова розмітки у WAP 2.0;

В) веб-сервер;

Г) додатковий плагін.

**10. Вся інформація в WML міститься в так званих «...».**

А) *деках*;

Б) *теках*;

В) *тезисах*;

Г) *картах*.

**11. ...- це мінімальний блок даних, який може бути переданий сервером.**

А) *карта*;

Б) *байт*;

В) *сервер*;

Г) *дек*.

**12. Багато мобільних пристроїв можуть відображати документи тільки в ...-форматі.**

А) *WML*;

Б) *XHTML*;

В) *JSON*;

Г) *WBXML*.

**13. Як розшифровується WBXML?**

А) *WAP BX Msxml Lego*;

Б) *WAP Binary XML*;

В) *Wireless Befault Xplorer ML*;

Г) *Wixit Beni Xit ML*.

**14. AJAX не залежить від програмного забезпечення веб-сервера і заснований на наступних веб-стандартах:**

А) *JavaScript, XML, HTML, CSS*;

Б) *JavaScript, CSS*;

В) *HTML, XML*;

Г) *AJAX, XML, CSS*.

**15. Об'єкт XMLHttpRequest підтримується в ...**

А) *Internet Explorer, Opera 8+*;

Б) *Safari 1.2, Mozilla 1.0, Netscape 7*;

В) *Firefox*;

Г) *усі відповіді правильні*.

## ВИКОРИСТАНА ЛІТЕРАТУРА

1. Якоб Нильсен, Кара Перниче Веб-дизайн: анализ удобства использования веб-сайтов по движению глаз Eyetracking Web Usability – М.: «Вильямс», 2010. – С. 480. – ISBN 978-5-8459-1652-5.
2. Якоб Нильсен Веб-дизайн – СПб: Символ-Плюс, 2003. – 512 с. – ISBN 5-93286-004-9.
3. Якоб Нильсен, Хоа Лоранжер Web-дизайн: удобство использования Web-сайтов Prioritizing Web Usability – М.: «Вильямс», 2007. – С. 368. – ISBN 0-321-35031-6.
4. Роббинс Д. Web-дизайн. Справочник. – «КУДИЦ-ПРЕСС», 2008. – С. 816. – ISBN 978-5-91136-039-9.
5. Гончаров А. Ю. Web-дизайн: HTML, JavaScript и CSS. Карманный справочник. – «КУДИЦ-ПРЕСС», 2007. – С. 320. – ISBN 978-5-91136-024-5.
6. Бородаев Д. В. Веб-сайт как объект графического дизайна. Монография – Х.: «Септима ЛТД», 2006. – С. 288. – ISBN 996-674-026-5.
7. Ед Титтел, Джефф Ноубл HTML, XHTML и CSS для чайников, 7-е издание HTML, XHTML & CSS For Dummies, 7th Edition – М.: «Диалектика», 2011. – С. 400 – ISBN 978-5-8459-1752-2.
8. Питер Лабберс, Брайан Олберс, Френк Салим HTML5 для профессионалов: мощные инструменты для разработки современных веб-застосовань Pro HTML5 Programming: Powerful APIs for Richer Internet Application Development – М.: «Вильямс», 2011. – С. 272 – ISBN 978-5-8459-1715-7.
9. Стивен Шафер HTML, XHTML и CSS. Библия пользователя, 5-е издание HTML, XHTML, and CSS Bible, 5th Edition – М.: «Диалектика», 2010. – С. 656. – ISBN 978-5-8459-1676-1.
10. Фримен Ерик, Фримен Елизабет. Изучаем HTML, XHTML и CSS Head First HTML with CSS & XHTML – 1-е изд. – М.: «Питер», 2010. – С. 656. – ISBN 978-5-49807-113-8.
11. Ед Титтел, Джефф Ноубл HTML, XHTML и CSS для чайников, 7-е издание HTML, XHTML & CSS For Dummies, 7th Edition – М.: «Диалектика», 2011. – С. 400 – ISBN 978-5-8459-1752-2.
12. Стивен Шафер HTML, XHTML и CSS. Библия пользователя, 5-е издание HTML, XHTML, and CSS Bible, 5th Edition – М.: «Диалектика», 2010. – С. 656 – ISBN 978-5-8459-1676-1.
13. Энди Бадд, Камерон Молл, Саймон Коллизон CSS: профессиональное применение Web-стандартов CSS Mastery: Advanced Web Standards Solutions – М.: «Вильямс», 2008. – С. 272 – ISBN 978-5-8459-1199-5.
14. Кристофер Шмитт. CSS. Рецепты програмування CSS. Cookbook – СПб.: БХВ-Петербург, 2007. – С. 592 – ISBN 978-5-9775-0075-3.

15. Ерик А. Мейер. CSS-каскадные таблицы стилей: подробное руководство Cascading Style Sheets: The definitive Guide – М.: Символ, 2006. – С. 576 – ISBN 5-93286-075-8.
16. Zakas N. Professional JavaScript for Web Developers – 2nd ed. – USA, Canada: Wiley Publishing, Inc., 2009. – P. 840 – ISBN 978-0-470-22780-0.
17. Flanagan D. JavaScript: The Definitive Guide – 5th ed. – O'Reilly Media, Inc., 2006. – P. 1018 – ISBN 0596102012.
18. Фленеган Д. JavaScript. Подробное руководство JavaScript. The Definite Guide / Перевод А. Киселева – 5-е изд. – СПб.: «Символ-Плюс», 2008. – С. 992 – ISBN 5-93286-103-7.
19. Crockford D. JavaScript: The Good Parts – 2008. – P. 170 – ISBN 978-0-596-51774-8.
20. Resig J. Pro JavaScript™ Techniques – 1st ed. – Apress, 2006. – 384 p. – ISBN 1590597273.
21. Рейсиг Д. JavaScript. Профессиональные приёмы програмування = Pro JavaScript™ Techniques / Перевод Н. Вильчинский – СПб.: Питер, 2008. – С. 352 – (Библиотека программиста). – 2500 экз. – ISBN 978-5-91180-904-1.
22. Stefanov S. Object-Oriented JavaScript: Create scalable, reusable high-quality JavaScript applications and libraries – 1st ed. – Packt Publishing, 2008. – P. 356 – ISBN 184719414.
23. Harnes R., Diaz D. Pro JavaScript™ Design Patterns – 1st ed. – Apress, 2007. – С. 269 – ISBN 159059908X.
24. Keith J. DOM Scripting: Web Design with JavaScript and the Document Object Model – 1st ed. – friends of ED, 2005. – P. 368 – ISBN 1590595335.
25. Sambells J., Gustafson A. AdvancED DOM Scripting: Dynamic Web Design Techniques – 1st ed. – friends of ED, 2007. – P. 592 – ISBN 1590598563.
26. Koch P.-P. ppk on JavaScript – 1st ed. – New Riders Press, 2006. – P. 528 – ISBN 0321423305.
27. Zakas N., McPeak J., Fawcett J. Professional Ajax – 2nd ed. – Wrox, 2007. – P. 624 – (Programmer to Programmer). – ISBN 0470109491.
28. Keith J. Bulletproof Ajax – 1st ed. – New Riders, 2007. – P. 216 – ISBN 0321472667.
29. Официальный учебный курс Adobe Flash CS4 + CD – М.: «Ексмо», 2009. – С. 400. – ISBN 978-5-699-35343-9, 978-0-321-57382-7.
30. Бурлаков Михаил Викторович Adobe Flash CS3. Самоучитель – М.: «Диалектика», 2007. – С. 624. – ISBN 978-5-8459-1319-7.
31. Шон Пакнелл, Брайан Хогг, Крейг Суонн Macromedia Flash 8 для профессионалов Macromedia Flash Demystified – М.: «Вильямс», 2006. – С. 672. – ISBN 0-7357-1397-9.
32. И. Розенсон Основы теории дизайна – СПб.: Питер, 2006. – С. 224 – ISBN 5-469-01143-7.
33. Е. Туемлоу Графический дизайн. Фирменный стиль, новейшие технологии и креативные идеи – М.: АСТ, 2007. – С. 256 – ISBN 5-17-041011-5.

34. Fiell, Charlotte & Peter (Editors). Contemporary Graphic Design. TASCHEN Publishers, 2008. ISBN 978-3-8228-5269-9
35. Wiedemann, Julius & Taborda, Felipe (Editors). Latin-American Graphic Design. TASCHEN Publishers, 2008. ISBN 978-3-8228-4035-1
36. Якоб Нильсен, Хоа Лоранжер. Web-дизайн: удобство использования Web-сайтов Prioritizing Web Usability – М.: «Вильямс», 2007. – С. 368. – ISBN 0-321-35031-6.
37. Стив Круг. Как сделать сайт удобным. Юзабилити по методу Стива Круга Rocket Surgery Made Easy: The Do-It-Yourself Guide to Finding and Fixing Usability Problems – СПб.: Питер, 2010. – С. 208. – ISBN 978-5-49807-515.
38. Джеф Раскин. Интерфейс: новые направления в проектировании компьютерных систем – Символ-Плюс. – 2004. – С. 368. – ISBN 5-93286-030-8.
39. Гарретт Д. Веб-дизайн: книга Джесса Гарретта. Элементы опыта взаимодействия The Elements of User Experience: User-Centered Design for the Web – Символ-Плюс, 2008. – С. 192. – ISBN 5-93286-108-8.
40. Дмитрий Котеров, Алексей Костарев PHP. В подлиннике – СПб.: «БХВ-Петербург», 2005. – С. 1120. – ISBN 5-94157-245-X.
41. Костарев А. Ф. PHP 5 – СПб.: «БХВ-Петербург», 2008. – С. 1104. – ISBN 978-5-9775-0315-0.
42. Метт Зандстра PHP: объекты, шаблоны и методики програмування, 3-е издание PHP Objects, Patterns and Practice, Third Edition – М.: «Вильямс», 2010. – С. 560. – ISBN 978-5-8459-1689-1.
43. Кристиан Дари, Емилиан Баланеску PHP и MySQL: создание интернет-магазина Beginning PHP and MySQL E-Commerce: From Novice to Professional – М.: «Вильямс», 2010. – С. 1404. – ISBN 978-5-8459-1602-0.
44. Джейсон Ленгсторф PHP и jQuery для профессионалов Pro PHP and jQuery – М.: «Вильямс», 2010. – С. 352. – ISBN 978-5-8459-1693-8.
45. Стив Суеринг, Тим Конверс, Джойс Парк PHP и MySQL. Библия программиста, 2-е издание PHP 6 and MySQL 6 Bible – М.: «Диалектика», 2010. – С. 912. – ISBN 978-5-8459-1640-2.
46. Квентин Зервас Web 2.0: создание застосовань на PHP Practical Web 2.0 Applications with PHP – М.: «Вильямс», 2009. – С. 544. – ISBN 978-5-8459-1590-0.
47. Кузнецов Максим, Симдянов Игорь PHP 5/6 – СПб.: «БХВ-Петербург», 2009. – С. 1024. – ISBN 978-5-9775-0304-4.
48. Кузнецов Максим, Симдянов Игорь Объектно-ориентированное програмування на PHP – СПб.: «БХВ-Петербург», 2007. – С. 608. – ISBN 978-5-9775-0142-2.
49. Ед Леки-Томпсон, Алек Коув, Стивен Новицки, Хью Айде-Гудман PHP 5 для профессионалов Professional PHP 5 – М.: «Диалектика», 2006. – С. 608. – ISBN 0-7645-7282-2.
50. Кузнецов Максим, Симдянов Игорь Самоучитель PHP 5/6 – 3-е изд., перераб. и доп.. – СПб.: «БХВ-Петербург», 2009. – С. 672. – ISBN 978-5-9775-0409-6.

51. Кузнецов Максим, Симдянов Игорь PHP. Практика создания Web-сайтов – 2-е изд. перераб. и доп.. – СПб.: «БХВ-Петербург», 2008. – С. 1264. – ISBN 978-5-9775-0203-0.
52. Кузнецов Максим, Симдянов Игорь Головоломки на PHP для хакера – 2-е изд. перераб. и доп.. – СПб.: «БХВ-Петербург», 2008. – С. 544. – ISBN 978-5-9775-0204-7.
53. Кузнецов Максим, Симдянов Игорь, Голышев Сергей PHP 5 на примерах – СПб.: «БХВ-Петербург», 2005. – С. 576. – ISBN 5-94157-670-6.



## ГЛОСАРІЙ

DOM (Document Object Model - DOM) – інтерфейс прикладного програмування браузера, пов'язує між собою HTML, CSS та мови сценаріїв.

JavaScript - це мова програмування, що використовується в складі HTML-сторінок для збільшення їх функціональності та можливостей взаємодії з користувачем.

Web-орієнтовані інформаційні системи – це інформаційні системи, призначені та пристосовані для використання мережі WWW.

Web-сервер – сервер, який обслуговує запити користувачів (клієнтів) згідно з протоколом HTTP, що забезпечує актуалізацію, збереження інформації WEB-сторінки, зв'язок з іншими серверами.

Web-сторінка (Web-сайт) – окремий інформаційний ресурс мережі WWW, що має власну адресу.

HTML-документ – файл текстової або нетекстової природи (звук, відео, зображення), створений за допомогою мови гіпертекстової розмітки HTML (Hyper Text Markup Language).

TCP/IP – це множина комунікаційних протоколів, що визначають комп'ютери різноманітних типів з різноманітними операційними системами, що можуть спілкуватись між собою.

Браузер – спеціальне програмне забезпечення, що надає користувачам інтерфейс для доступу до інформації Web-сторінок та їх перегляду.

Впровадження стилів в HTML-документ – це технологія, що дозволяє задати всі правила таблиці стилів безпосередньо в самій HTML-сторінці.

Впровадження стилів в тег HTML-документу – це технологія, що дозволяє змінювати форматування конкретних елементів HTML-сторінки.

Гіперпосилання – методика зв'язування одного Web-ресурсу з іншим. Гіперпосилання складається із двох частин. Перша частина – це об'єкт, який користувач бачить у вікні браузера і вибір якого призводить до переходу на цільовий Web-ресурс. Гіперпосилання складається із двох частин: перша частина – це об'єкт який користувач бачить у вікні браузера і вибір якого призводить до переходу на цільовий Web-ресурс.

Динамічний HTML (DHTML) – це термін, який використовується для позначення HTML-сторінок з динамічно змінюваним змістом. Реалізація DHTML складається із трьох компонентів: безпосередньо HTML, каскадних таблиць стилів (CSS) та мови сценаріїв (JavaScript або VBScript).

Інтернет (мережа Інтернет) – сукупність мереж та обчислювальних засобів, які використовують стек протоколів TCP/IP (Transport Control Protocol/Internet Protocol), спільний простір імен та адрес для забезпечення доступу користувачів до інформаційних ресурсів мережі.

Зв'язування стилів з HTML-документом – це технологія, що дозволяє використовувати одну таблицю стилів для форматування декількох HTML-сторінок.

Каскадна таблиця стилів (CSS) - це технологія визначення та з'єднання стилів з HTML-сторінкою.

Клієнт мережі WWW – браузер, що використовується клієнтом мережі Інтернет для доступу до ресурсів мережі WWW.

Метод об'єкту JavaScript – це функція, що пов'язана з об'єктом.

Об'єкт JavaScript – це цілісна конструкція, що має властивості, які є змінними JavaScript та методи їх обробки.

Провайдер – юридична або фізична особа, яка надає користувачам доступ до мережі Інтернет.

Сервер – об'єкт комп'ютерної системи (програмний або програмно-апаратний засіб), що надає послуги іншим об'єктам за їх запитом.

Списки – широко розповсюджена форма показу даних. Мовою HTML передбачено використання трьох стандартних видів списків: маркірованого, нумерованого та списку визначень.

Стиль - це все те, що визначає зовнішній вигляд HTML-сторінки при її відображенні у вікні браузеру.

Таблиця стилів - це шаблон, який керує форматуванням тегів HTML-сторінки.

Тег HTML – команда мови HTML.

Скрипт (сценарій) – програма JavaScript.

Форма – засіб для передачі інформації від клієнта на Web-сервер. Як правило, результатом такої передачі є відображення у клієнта нового HTML-документа, сформованого Web-сервером на основі переданої інформації.

Фрейм – прямокутна область вікна браузера, в яку можна завантажити окремий HTML-документ. В основному фрейми застосовуються для організації управління завантаженням HTML-документів в певну область вікна браузеру при роботі користувача в іншій області, тобто для організації меню.

Функція JavaScript – це іменована група команд, які вирішують певну задачу та можуть повернути деяке значення.

**Пасічник В. В.**

**Пасічник О. В.**

**Угрин Дмитро Ілліч**

# **ВЕБ-ТЕХНОЛОГІЇ**

**Підручник**

**І частина**

НБ ПНУС



783923

Підписано до друку 01.02.2013 р.

Формат 70x100/16. Папір друк. №2. Гарнітура Times New Roman

Умовн. друк. арк. 21. Тираж 300 прим.

ПП “Магнолія 2006”

а/с 431, м. Львів-53, 79053, Україна, тел./факс 240-54-84; 245-63-70

e-mail: magnol@lviv.farlep.net

Свідоцтво про внесення суб’єкта видавничої справи

до Державного реєстру видавців, виготівників і розповсюджувачів видавничої

продукції: серія ДК № 2534 від 21.06.2006 року,

видане Державним комітетом інформаційної політики,

телебачення та радіомовлення України

Надруковано у друкарні видавництва “Магнолія 2006”

м. Львів, вул. Луганська, 1 Б.





**Пасічник Володимир Володимирович** – доктор технічних наук, професор, завідувач кафедри інформаційних систем та мереж Національного університету «Львівська політехніка» у 1994-2011 рр.

Вихованець наукової школи київського Інституту кібернетики імені В. М. Глушкова. Кандидат фізико-математичних наук (1982 рік, наукова спеціальність «Математична кібернетика»), доктор технічних наук (1994 рік, наукова спеціальність «Теоретичні основи інформатики»).

Учасник і керівник багатьох міжнародних наукових проєктів та перспективних науково-дослідних розробок. Автор багатьох наукових праць і навчальних посібників, у тому числі декількох підручників з грифом Міністерства освіти і науки, молоді та спорту України. Лауреат Державної премії України в галузі науки і техніки, відмінник освіти України.

Наукові інтереси: інформаційне моделювання, системи баз даних та знань, інтелектуальні системи прийняття рішень.



**Пасічник Оксана Володимирівна** – фахівець в області комп'ютерної лінгвістики, комп'ютерних мережеских технологій, дистанційної освіти. 2004 року закінчила з відзнакою Національний університет «Львівська політехніка» за фахом «Прикладна лінгвістика» (профілізація «Комп'ютерна лінгвістика») та отримала суміжну вищу освіту за фахом «Викладач інформатики та іноземних мов».

Автор ряду наукових праць та методичних розроблень в галузі застосування комп'ютерних інформаційних технологій в освітніх процесах для дітей з обмеженими можливостями здоров'я. Ініціатор створення освітнього центру для дітей, що за станом здоров'я не мають змоги відвідувати заняття в загальноосвітній школі.

Працює викладачем предмету «Інформатика» в гімназії «Сихівська» міста Львова.

Активно співпрацює в рамках міжнародних освітніх програм з навчальними закладами Швеції, Польщі, Чехії, Німеччини. Володіє рядом європейських мов. Учасник міжнародних лінгвістичних семінарів та конференцій (Варшава – 2000 р., Прага – 2003 р., 2004 р., Будапешт – 2006 р., Хапаранда – 2007 р., Стокгольм – 2008 р.).



**Угрин Дмитро Ілліч** – кандидат технічних наук, доцент кафедри автоматизованих систем управління факультету комп'ютерних систем і технологій приватного вищого навчального закладу «Буковинський університет».

Область наукових інтересів: простори та сховища даних, методи та засоби інтеграції даних, проєктування систем у концепції хмарних обчислень.

НБ ПНУС



783923

ISBN 978-617-574-093-4



9 786175 740934