Р. А. Мельник

Видавництво

ПРОГРАМУВАННЯ ВЕБ-ЗАСТОСУВАНЬ (ФРОНТ-ЕНД ТА БЕК-ЕНД)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Р. А. Мельник

ПРОГРАМУВАННЯ ВЕБ-ЗАСТОСУВАНЬ (ФРОНТ-ЕНД ТА БЕК-ЕНД)

Навчальний посібник

Рекомендувала Науково-методична рада
Національного університету "Львівська політехніка"

Львів
Видавництво Львівської політехніки
2018

УДК 004.738.5(075.8)
М 482

Рецензенти:

Грицюк Ю. І., д-р техн. наук, проф. кафедри програмного забезпечення Національного університету "Львівська політехніка";

Демида Б. А., канд. техн. наук, доц., доц. кафедри автоматизованих систем управлінн Національного університету "Львівська політехніка";

Хвищун І. О., канд. техн. наук, доц., доц. кафедри радіофізики та комп'ютерних технологій Львівського національного університету імені Івана Франка

В оформленні обкладинки використано логотипи технологій мережевих застосувань.

Рекомендувала Науково-методична рада
Національного університету "Львівська політехніка" як навчальний посібник
для студентів спеціальності 121 "Інженерія програмного забезпечення"
(протокол № 34 від 15.03.2018 р.)

Мельник Р. А.

М 482 Програмування веб-застосувань (фронт-енд та бек-енд) : навч. посібник / Р. А. Мельник. – Львів : Видавництво Львівської політехніки, 2018. – 248 с.

ISBN 978-966-941-195-2

Викладено матеріал для проектування динамічних web-сторінок. Розділи розкривають мови кодування HTML, HTML5, CSS, JQUERY, W3.CSS, Bootstrap, AngularJS, мови програмування JavaScript та PHP, технологію Ajax, зберігання даних у масивах та базах даних MySQL-сервера. Наведено приклади доступу до них з Web-сторінок. Подано основи проектування Web-сторінок у технологіях .NET та JAVA, зокрема за допомогою класів з бібліотек ASP.NET, класів побудови сервлетів javax.servlet та JSP засобів проектування динамічних web-сторінок. Усі розділи завершуються контрольними запитаннями. Матеріал містить фрагменти програм, що можуть бути корисними студентам під час самостійного розроблення власних програм до лабораторних та курсових завдань.

Для студентів ВНЗ та коледжів, які вивчають сучасні технології створення програмного забезпечення, зокрема за ефективною архітектурою "клієнт-сервер".

УДК 004.738.5(075.8)

ISBN 978-966-941-195-2

Частина I

ФРОНТ-ЕНД ЗАСОБАМИ HTML, CSS, JAVASCRIPT, BOOTSTRAP, ANGULAR

Глава 1
ЕЛЕМЕНТИ ДИНАМІКИ НА КЛІЄНТІ

1.1. Сценарій взаємодії в Інтернеті

У разі підтримки динамічних змін на web-сторінці на стороні сервера взаємодію програм-учасників динамічного сценарію проілюстровано на рис. 1.1 за умови, що сервером є програмний продукт Apache, сервер СУБД – MySQL, а скрипти оновлення змісту сторінок написані мовою PHP.

Послідовність дій за цим сценарієм є такою:

1. Браузер користувача надсилає HTTP-запит певної web-сторінки на web-сервер.

2. Web-сервер приймає запит і передає його інтерпретатору PHP на опрацювання.

3. Механізм PHP починає синтаксичний аналіз сценарію. У сценарії передбачено команду під'єднання до бази даних і виконання запиту в ній. Інтерпретатор PHP відкриває з'єднання з сервером MySQL і відсилає запит на отримання чи модифікацію даних.

4. Сервер MySQL приймає запит до бази даних, опрацьовує його, після чого надсилає результати назад у механізм PHP.

5. Інтерпретатор PHP завершує виконання сценарію, формуючи результати опрацювання запиту у вигляді HTML, і відсилає результати в HTML форматі для web-сервера.

6. Web-сервер пересилає HTML-документ до браузера, за допомогою якого користувач переглядає результати.

Така схема, як правило, реалізується незалежно від того, який сценарний механізм і який сервер баз даних використовується. Переважно програмне забезпечення web-сервера, механізм PHP і сервер баз даних розміщені на одному комп'ютері. Це роблять з огляду на правила безпеки, збільшення обсягу запитів або поділу потоків тощо.

Для взаємозв'язку клієнта і сервера використовують форми HTML. Тому для проектування модулів на боці сервера програміст

повинен володіти знаннями про основні елементи мови HTML – інструменту створення web-сторінок і особливо ланки для зв'язку сторінок з програмами, розташованими на сервері (рис. 1.1).

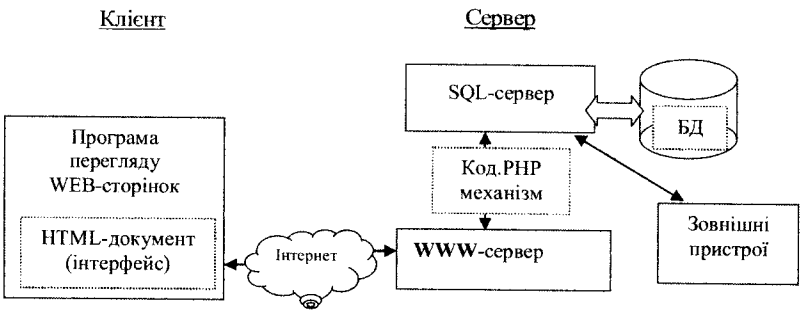


Рис. 1.1. Схема взаємодії клієнта і сервера через PHP

Комп'ютери об'єднуються у мережах на основі IP-протоколу. World Wide Web (WWW) – це об'єднання web-сайтів та сторінок з усього світу. Internet є значно ширшим поняттям, він передбачає інші послуги: електронну пошту, зберігання файлів, конференції, ігри тощо. IP- протокол – це протокол для пересилання даних між двома комп'ютерами, який містить IP-адресу на 32 біти. Адреса формується як чотири 8-бітні числа (між 0 та 255), наприклад, 145.10.34.3. Знайти Internet IP-адресу можна за допомогою ресурсу: whatismyip.com. Знайти свою локальну IP-адресу можна за допомогою утиліт: ipconfig (Windows), ifconfig (Mac/Linux).

Важливими є транспортні протоколи TCP та UDP. Порт призначений для програм чи сервісів, наприклад, 80: Web-браузер, 25: електронна пошта.

Тексти сторінок між комп'ютерами пересилаються за протоколом прикладного рівня HTTP. Web-сервер повертає браузеру коди виконання, наприклад: 200 (OK); 301–303 (сторінка відсутня); 403 (доступ заборонено); 404 (сторінка не знайдена); 500 (внутрішня помилка сервера).

Простір назв DNS – це ієрархічна структура назв серверів (карта відповідних IP-адрес), наприклад, lp.edu.ua.

1.2. Структура DOM та методи доступу до вузлів дерева

Web-сторінка може мати вигляд дерева, вузли якого є об'єктами, до властивостей яких доступаються операторами мови програмування Javascript.

Модель DOM (Document Object Model, об'єктна модель документа) містить низку стандартних глобальних об'єктів. Зокрема, це `window`, `navigator`, `document`, `screen`, `history`, `location`. Ієрархію основних об'єктів у моделі DOM подано на рис. 1.2.

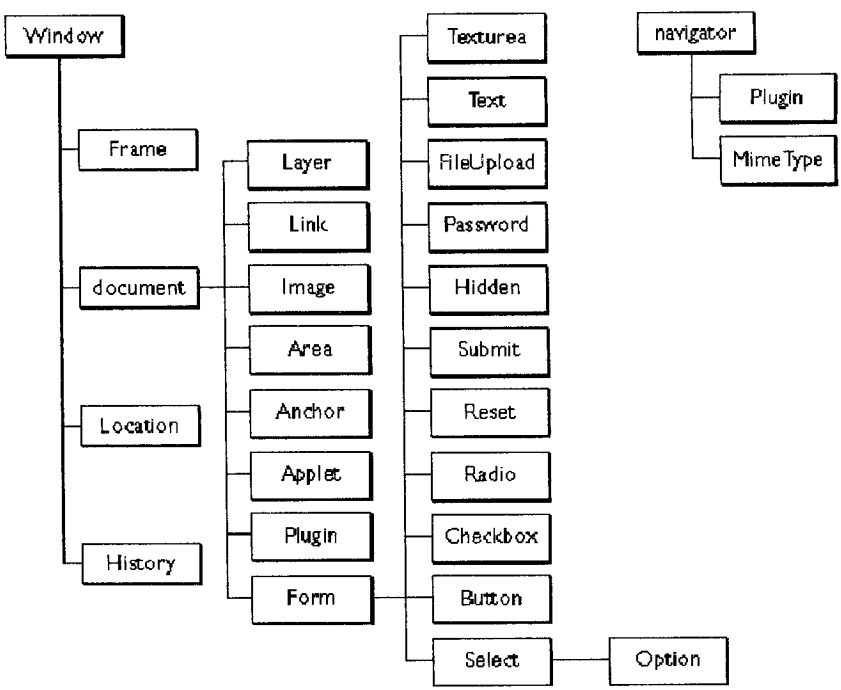


Рис. 1.2. Дерево моделі DOM об'єкта `window` браузера

Перерахуємо глобальні об'єкти та їх призначення, а також розглянемо методи та властивості глобальних об'єктів DOM:

• window – вікно браузера, найвищий об'єкт ієрархії DOM. Усі глобальні змінні стають частинами об'єкта window. Його методи такі:

alert, blur, clearInterval, clearTimeout, close, confirm, focus, moveBy, moveTo, open, print, prompt, resizeBy, resizeTo, scrollBy, scrollTo, setInterval, setTimeout

Властивості: document, history, location, name

• navigator – інформація про браузер.

Властивості: appName, appVersion, browserLanguage, cookieEnabled, platform, userAgent

• screen – інформація про екран, який використовує браузер.

Властивості: availHeight, availWidth, colorDepth, height, pixelDepth, width

• history – список викликаних сторінок.

Властивості: length

Методи: back, forward, go

• location – URL поточної web-сторінки.

Властивості: host, hostname, href, pathname, port, protocol, search

Методи: assign, reload, replace

• document – поточна Web-сторінка.

Властивості: anchors, body, cookie, domain, forms, images, links, referrer, title, URL

Методи: close, getElementById, getElementsByName, open, write, writeln, getElementsByTagName

Приклад ієрархічного дерева DOM для об'єкта document наведено на рис. 1.3.

За змістом вузли дерева поділяють на три типи:

• елементів (HTML теги) – породжені вузли або атрибути;

• тексту (текст або блоки) – не мають породжених вузлів чи атрибутів;

• атрибутів (пари атрибут/значення) – не мають породжених вузлів чи атрибутів.

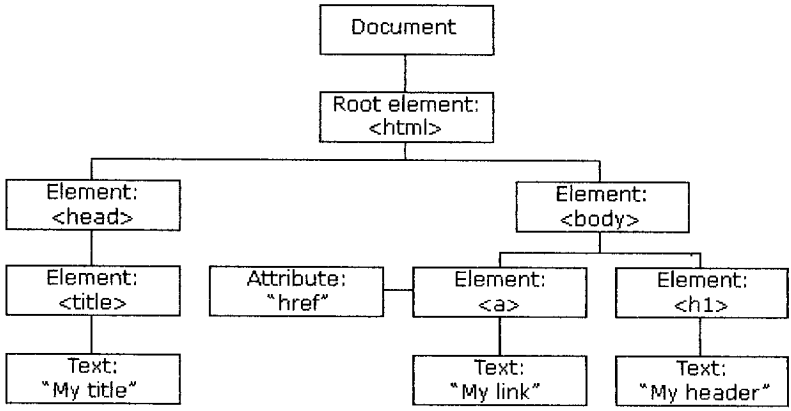


Рис. 1.3. Дерево об'єкта `document` *у моделі DOM*

Наведемо фрагмент сторінки і відповідне йому дерево (рис. 1.4)

```
<p>This is a paragraph of text with a
<a href="/path/to/another/page.html">link inside
 </a>.
```

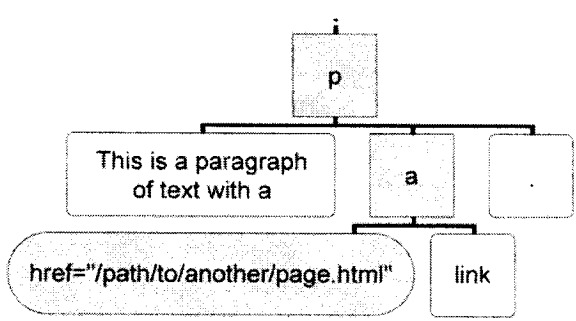


Рис. 1.4. Фрагмент дерева з вузлом параграфа

За місцем розташування на дереві виділено такі типи вузлів: `parent` – батьківський, `child` – породжений, `sibling` – сусідній, `firstchild` – перший породжений, `lastchild` – останній породжений.

Назви вузлів є складовими назвами властивостей вузлів та параметрів методів, які ними оперують (ліквідують, модифікують, додають тощо). Вузли DOM-дерева мають такі властивості:

- `firstChild, lastChild` – початковий та кінцевий породжені вузли;
- `childNodes` – масив породжених вузлів;
- `nextSibling, previousSibling` – сусідні вузли (по одному), що мають одного батька;
- `parentNode` – елемент, що породив цей вузол.

Мова програмування дає змогу змінювати склад вузлів, вигляд та зміст web-сторінки. На моделі DOM програміст має можливість виконувати операції двох типів: на рівні вузлів для зміни структури дерева додаванням або відніманням вузла та заміни вузла; на рівні вузла для зміни змісту елемента. Зокрема, для першого типу операцій застосовуються методи:

- `appendChild(node)`: розміщає вузол у кінці списку породжених вузлів до заданого;
- `insertBefore(newChild, oldChild)`: розміщає новий вузол перед вузлом `oldChild` у списку породжених вузлів до заданого;
- `removeChild(node)`: знищує наведений вузол зі списку породжених вузлів до заданого;
- `replaceChild(newChild, oldChild)`: заміняє породжений на новий вузол.

Властивості набувають значення

- `nodeType`: 1 – елемент, 2 – атрибут, 3 – текст, 4 – коментарі, 9 – документ.
- `nodeName`: повертає версію тега, наприклад, `"div"` чи `"article"`. Текстові вузли мають назву `"#text"` . Назва вузла `document` є `"#document"`,
- `nodeValue`: текст вузла або значення атрибута.

Доступ до вузлів DOM за тегами чи ідентифікаторами здійснюється методами:

- `document.getElementById("id")` – елемент;

• element.getElementsByTagName("tag") – усі породжені вузли;

• document.createElement("tag") – команда створює новий порожній вузол для подання елемента певного типу.

Розглянемо декілька прикладів застосування методів внесення змін у структуру дерева та змін у змісті вузла – елемента сторінки:

1) створення нового вузла <h2> в області сторінки (div), позначеної як id="content":

```
var newHeading = document.createElement("h2");
newHeading.style.color = "green";// встановлення
властивостей
newHeading.innerHTML = "This is a heading";
var contentArea =
document.getElementById("content");
contentArea.appendChild(newHeading);
```

2) зміна фону тексту пунктів списку:

для тегів параграфів змінити фон тексту

```
<body>
<p>This is the first paragraph</p>
<p>This is the second paragraph</p>
</body>
. . . . . .
var allParas = document.getElementsByTagName("p");
for (var i = 0; i < allParas.length; i++) {
    allParas[i].style.backgroundColor = "yellow";
}
```

3) додавання вузла DOM як реакція на подію з використанням innerHTML

```
. . . . . .
document.getElementById("thisslide").onclick =
slideClick;
. . . . . .
function slideClick() {
    var p = document.createElement("p");
    p.innerHTML = "A new paragraph!";
    this.appendChild(p);
}
```

або з занесенням додаткових (зокрема події) атрибутів параграфа:

```
function slideClick() {
```

```
    var p = document.createElement("p");
    p.style.color = "red";
    p.style.marginLeft = "50px";
    p.onclick = myOnClick;
    p.innerHTML = "A paragraph!"; // занесення в
innerHTML
    this.appendChild(p);
}
```

1.3. Форма зворотного зв'язку HTML

Форма HTML містить групу елементів для приймання від користувача та пересилання даних у рядку запиту на web-сервер. Ними є `button`, `checkbox`, поля типу `text`, `input` тощо.

Важливий атрибут `action` вказує на URL сервера та програму на ньому для опрацювання даних запиту. Атрибут `method` вказує на вибір метода передачі даних: `get` чи `post`. Наведемо приклад тегу форми,

```
<form action="http://www.facerec.com/app.php"
method="get">
 <fieldset>
  <label>Name: <input type="text" name="name"
/></label>
  <label>sex: <input type="text" name="sex"
/></label>
  <label>country?
    <input type="checkbox" name="country" />
  </label>
  <input id="sbutton" type="submit" value="Send me
your picture!"/>
 <fieldset>
</form>
```

Для перевірки правильності внесених на форму даних використовуються скрипти JavaScript.

Наприклад, якщо під час натискання кнопки відсилання даних певне поле даних не заповнене, то запит не надсилається. Фрагмент коду для реалізації цього сценарію наведено нижче:

```
. . . . .
var submit = document.getElementById("sbutton");
submit.onclick = submitClick();
. . . . .
    function submitClick(event) {
        if (document.getElementById("name").value == "")
  {
            event.preventDefault();  // відмінити submit
        }
    }
```

Код передбачає зупинку пересилання даних – `prevent Default`. (IE9 використовує `return false;` замість `preventDefault`).

Після натискання кнопки `submit` браузер формує запит у вигляді послідовності двох груп даних : заголовок `http`-протоколу та рядок з `DNS` адресою сервера, назвою програми для виконання та пар параметрів (назва–значення) :

```
GET /index.html HTTP/1.1
Host: localhost
Accept: image/gif, image/x-xbitmap, image/jpeg,
image/pjpeg, image/xbm, */*
Accept-Language: en
Connection: Keep-Alive
User-Agent: Mozilla/4.0 (compatible; MSIE 4.5;
Mac_PowerPC)
```

http://www.xul.ua/somefile.xml'

```
first=this+is+a+field&second=was+it+clear+%28already%
29%3F
```

Форма `HTML` приймає параметри для передачі на сервер та опрацювання їх відповідним програмним засобом. Для зменшення кількості помилок на сервері, зменшення трафіку пересилання на клієнті використовують скрипти попередньої перевірки правильності та адекватності введених даних.

Розглянемо приклад перевірки відсутності незаповненого поля, правильності формату електронної адреси. На формі (після тегу `form`) використано два поля для імені та адреси:

```
<form name="myForm" action="demo_form.asp"
onsubmit="return validateForm()" method="post">
```

```
First name: <input type="text" name="fname">
Email: <input type="text" name="email">
<input type="submit" value="Submit">
</form>
```

Наступний код JavaScript функції перевіряє правильність заповнення полів параметрами для передачі. Функція викликається під час натискання кнопки submit:

```
function validateForm()
{
var x=document.forms["myForm"]["fname"].value;
if (x==null || x=="")
  {
  alert("First name must be filled out");
  return false;
  }
var x=document.forms["myForm"]["email"].value;
var atpos=x.indexOf("@");
var dotpos=x.lastIndexOf(".");
if (atpos<1 || dotpos<atpos+2 || dotpos+2>=x.length)
  {
  alert("Not a valid e-mail address");
  return false;
  }
}
```

Для перевірки збігу введених значень пароля використовується така функція:

```
function PwdValidation()
{
  var frm = document.forms["myform"];
  if(frm.pwd1.value != frm.pwd2.value)
  {
    sfm_show_error_msg('The Password and verified
password does not match!',frm.pwd1);
    return false;
  }
  else
  {
    return true;
  }
}
```

1.4. Нові можливості HTML5

1.4.1. Задачі та зміст HTML5

HTML5 буде новим стандартом для HTML. У попередній версії HTML 4.01 отримав стандарт у 1999 році. HTML5 знаходиться на стадії розроблення і вдосконалення. Основні браузери підтримують багато нових елементів HTML5 і API. HTML5 є результатом співпраці між World Wide Web Consortium (W3C) і робочої групи Web-додатків гіпертекстової технології (WHATWG). WHATWG працює з web-формами і додатками, а W3C працює з XHTML 2.0. У 2006 році вони вирішили об'єднатися і створити нову версію HTML.

Для HTML5 було розроблено деякі правила:

- нові функції повинні бути засновані на `HTML, CSS, DOM і JavaScript`;
- зниження потреби у зовнішніх плагінах (наприклад, Flash);
- покращене оброблення помилок;
- ефективніша розмітка для нових сценаріїв;
- HTML5 має бути незалежною від пристроїв;
- процеси розвитку мають бути доступні громадськості.

Вдосконалення заплановані у таких розділах:

- нові елементи;
- нові атрибути;
- повна підтримка CSS3 ;
- підтримка відео та аудіо;
- 2D/3D-рафіка;
- локальне зберігання даних;
- локальний доступ до SQL баз даних;
- підтримка web-додатків.

Нижче наведено приклад з використання відеофайлів:

```
<!DOCTYPE HTML>
<html>
<body>
<video width="320" height="240" controls="controls">
  <source src="movie.mp4" type="video/mp4">
```

```
    <source src="movie.ogg" type="video/ogg">
    <source src="movie.webm" type="video/webm">
  </video>
  </body>
  </html>
```

У HTML5 є тільки одне просте оголошення `<тип документа>`:

```
<! DOCTYPE HTML>
```

Запропоновано спростити документи HTML5, мінімізуючи кількість тегів:

```
<!DOCTYPE html>
<html>
<head>
<title>Title of the document</title>
</head>
<body>
Зміст документу ......
</body>
</html>
```

Найцікавішими новими можливостями HTML5 є:

- тег `<canvas>` для 2D малювання;
- `<video>` і `<audio>` елементи для відтворення мультимедіа;
- підтримка для локального зберігання;
- новий зміст конкретних елементів, такий як `<article>`, `<footer>`, `<header>`, `<nav>`, `<section>`;
- нові елементи управління форми, такі як календар, дата, час, адреса електронної пошти, URL, пошук.

Нові елементи HTML5 наведено в таблиці:

Тег	Опис
1	2
<article>	Визначає статті
<aside>	Визначає зміст осторонь від вмісту сторінки
<bdi>	Ізолює частину тексту, який може бути відформатований у напрямку, відмінному від іншого тексту за її межами
<command>	Визначає командну кнопку, яку користувач може викликати

Продовження таблиці

<details>	Визначає додаткову інформацію, яку користувач може переглянути або приховати
<summary>	Визначає видимий заголовок для елемента <details>
<figure>	Визначає автономний контент – ілюстрації, діаграми, фотографії, лістинги тощо.
<figcaption>	Визначає заголовок для елемента <figure>
<footer>	Визначає нижній колонтитул документа або розділу
<header>	Визначає заголовок документа або розділу
<hgroup>	Групи <h1> до <h6> елементів, коли заголовок має декілька рівнів
<mark>	Визначає виокремлений / виділений текст
<meter>	Визначає скалярні вимірювання у відомому діапазоні (датчик)
<nav>	Визначає навігаційні посилання
<progress>	Подає послідовність виконання завдання
<ruby>	Визначає анотацію
<rt>	Визначає пояснення / вимову символів
<rp>	Визначає, що показати в браузерах, які не підтримують Ruby-анотації
<section>	Визначає розділ у документі
<time>	Визначає дату / час
<wbr>	Визначає можливі розриви рядка

Нові елементи HTML5 для вбудованих медіа-ресурсів наведено в таблиці:

Тег	Опис
<audio>	Визначає звук змісту
<video>	Визначає відео або фільм

Продовження таблиці

1	2
<source>	Визначає кілька медіа-ресурсів для <video> і <audio>
<embed>	Визначає контейнер для зовнішнього застосування або інтерактивного контенту (плагін)
<track>	Визначає текст треків для <video> і <audio>

Елементи для створення кращих фікційних можливостей HTML5 наведено в таблиці:

Тег	Опис
<datalist>	Визначає список варіантів управління введенням
<keygen>	Визначає ключові пари генераторів поля (для форми)
<output>	Визначає результат розрахунку

У HTML5 вилучені елементи HTML 4.01: <acronym>, <applet>, <basefont>, <big>, , <dir>, <font>, <frame>, <frameset>, <noframes>, <strike>, <tt>.

1.4.2. Засоби Canvas

Елемент HTML5 <canvas> використовується для малювання графіки під час роботи зі сторінкою за допомогою сценаріїв JavaScript. <canvas> є контейнером для методів графіки. Полотно має декілька методів для малювання ліній, прямокутників, кіл, персонажів і додавання зображень.

Полотно – це прямокутна ділянка на HTML-сторінці. Розмітка виглядає так:

```
<canvas id="myCanvas" width="200"
height="100"></canvas>
```

Можна позначити границі прямокутного поля:

```
<canvas id="myCanvas" width="200" height="100"
style="border:1px solid #000000;">
</canvas>
```

Усі малювання на полотні програмуються в JavaScript:

```
<script>
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
ctx.fillStyle="#FF0000";
ctx.fillRect(0,0,150,75);
</script>
```

За полотном закріплено двовимірну сітку. Верхній лівий кут полотна має координати (0,0). Так, метод `FillRect()` набуває параметрів (0,0,150,75). Прямокутник має 150×75 пікселів.

Прямі лінії на полотні зображаються методами:

- `MoveTo (x, y)` - визначає початкову точку лінії;
- `LineTo (x, y)` – визначає кінцеву точку лінії.

Для малювання лінії використовується метод `stroke()`:

```
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
ctx.moveTo(0,0);
ctx.lineTo(300,150);
ctx.stroke();
```

Для малювання кола – метод `arc(x,y,r,start,stop)` або `fill()`:

```
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
ctx.beginPath();
ctx.arc(95,50,40,0,2*Math.PI);
ctx.stroke();
```

Для представлення тексту у Canvas використовуємо методи:

- `font("font")` - позначення шрифту;
- `fillText(text,x,y)` - друк тексту на полотні;
- `strokeText(text,x,y)` - друк тексту на полотні;

```
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
ctx.font="30px Arial";
ctx.fillText("Hello World",10,50);
```

або

```
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
```

```
ctx.font="30px Arial";
ctx.strokeText("Hello World",10,50);
```

Градієнти у Canvas призначені для малювання різних фігур у вигляді зображень з градієнтами кольорів. Є два типи градієнтів:

- createLinearGradient(*x,y,x1,y1*) – лінійний;
- createRadialGradient(*x,y,r,x1,y1,r1*) – радіальний.

Для об'єкта градієнту необхідно задати позиції за градієнтом. Градієнт позицій може бути де завгодно в діапазоні від 0 до 1. Щоб використовувати градієнт, треба встановити FillStyle або StrokeStyle властивість градієнта, а потім малювати прямокутник, текст або лінії.

- createLinearGradient():

```
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
var grd=ctx.createLinearGradient(0,0,200,0);
grd.addColorStop(0,"red");
grd.addColorStop(1,"white");
ctx.fillStyle=grd;
ctx.fillRect(10,10,150,80);
```

- createRadialGradient():

```
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
var grd=ctx.createRadialGradient(75,50,5,90,60,100);
grd.addColorStop(0,"red");
grd.addColorStop(1,"white");
ctx.fillStyle=grd;
ctx.fillRect(10,10,150,80);
```

Для малювання зображення використовується метод drawImage(*image,x,y*):

```
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
var img=document.getElementById("scream");
ctx.drawImage(img,10,10);
```

1.4.3. Вбудована векторна графіка SVG

У HTML5 розроблено SVG – векторну графіку змінного масштабу (Scalable Vector Graphics). SVG використовують для визначення векторної графіки для web-сторінок. Графіки задаються

у форматі XML. SVG графіка не втрачає якості, якщо вона змінює масштаб або змінюються розміри. Кожен елемент і кожен атрибут у SVG-файлах може бути анімований. W3C рекомендує SVG.

Переваги використання SVG порівняно з іншими форматами зображень (наприклад, JPEG і GIF) є:

- SVG зображення можуть бути створені і відредаговані в будь-якому текстовому редакторі;
- SVG зображення можна знайти за індексами, можна стискати;
- SVG зображення можуть бути надруковані з високою якістю;
- SVG зображення змінюють масштаб без погіршення якості.

Подамо приклад використання SVG:

```
<!DOCTYPE html>
<html>
<body>
<svg xmlns="http://www.w3.org/2000/svg"
version="1.1" height="190">
  <polygon points="100,10 40,180 190,60 10,60
160,180"
  style="fill:lime;stroke:purple;stroke-
width:5;fill-rule:evenodd;">
</svg>
</body>
</html>
```

Для малювання інших фігур використовують коди:

```
Коло   -- <svg id="svgelem" height="200"
xmlns="http://www.w3.org/2000/svg">
    <circle id="redcircle" cx="50" cy="50" r="50"
fill="red" />
</svg>
Прямокутник -- <svg id="svgelem" height="200"
xmlns="http://www.w3.org/2000/svg">
    <rect id="redrect" width="300" height="100"
fill="red" />
</svg>
Лінія --   <line x1="0" y1="0" x2="200" y2="100"
          style="stroke:red;stroke-width:2"/>
</svg>
```

```
Полілінія -- <svg id="svgelem" height="200"
xmlns="http://www.w3.org/2000/svg">
 <polyline points="0,0 0,20 20,20 20,40 40,40 40,60"
fill="red" />
</svg>
```

1.4.4. Переміщення у HTML5

У HTML5 переміщення є частиною стандарту і застосовується до елементів сторінки:

```
<!DOCTYPE HTML>
<html>
<head>
<script>
function allowDrop(ev)
{
ev.preventDefault();
}
function drag(ev)
{
ev.dataTransfer.setData("Text",ev.target.id);
}
function drop(ev)
{
ev.preventDefault();
var data=ev.dataTransfer.getData("Text");
ev.target.appendChild(document.getElementById(data))
;
}
</script>
</head>
<body>
<div id="div1" ondrop="drop(event)"
ondragover="allowDrop(event)"></div>
<img id="drag1" src="img_logo.gif" draggable="true"
ondragstart="drag(event)" width="336" height="69">
</body>
</html>
```

Щоб зробити елемент доступним до переміщення, треба використати атрибут:

```
<img draggable="true">
```

Метод `dataTransfer.setData()` вказує тип даних і значення перетягуваних даних.

Атрибут `ondragsart` викликає функцію `drag(event)`, яка вказує на дані для переміщення:

```
function drag(ev)
{
ev.dataTransfer.setData("Text",ev.target.id);
```

Дані "Text" і значення ідентифікатора (id) елемента ("drag1").

Атрибут ondragover з подією вказує куди перемістити. За замовчуванням елемент не можна перенести на інший елемент. Мусить бути передбачено оброблення події на елементі (на місце якого відбувається перенесення):

```
event.preventDefault()
```

Атрибут `ondrop` для обробника події:

```
function drop(ev)
{
ev.preventDefault();
var data=ev.dataTransfer.getData("Text");
ev.target.appendChild(document.getElementById(data))
;
}
```

У тексті:

- метод `preventDefault()` для заборони опрацьовувати дані за замовчуванням;
- метод `dataTransfer.getData("Text")` для отримання даних;
- переносні дані ("drag1");
- додати переносні дані до вказаного елемента.

1.4.5. Доступ до значень пікселів у canvas

Значення пікселів зображення у canvas можна прочитати методом `getImageData()`. Метод надає десяткові значення складових кольору (rgb) та прозорості (alpha значення).

Метод `getImageData(x,y,length,breath)` надає значення пікселів з довільної прямокутної поверхні, де:

`x,y, length,breath` – координати прямокутника.

Виклик *`var imageData = ctx.getImageData(10,10,100,50);`* надає значення пікселів у прямокутнику 100 px на 50 px.

Виклик *`var imageData = ctx.getImageData(10,10,1,1);`* надає значення одного пікселя.

Значення кольорів пікселя знаходяться у змінних :

```
imageData.data[0] – r  ( 0 - 255)
imageData.data[1] – g  (0 - 255)
imageData.data[2] - b   (0 - 255)
imageData.data[3] - Alpha (0 - 255)
```

Для масиву з чотирьох пікселів маємо:

`var Pixel = ctx.getImageData(10,10,2,2);`

а відповідні значення кольорів подано на рис. 1.5.

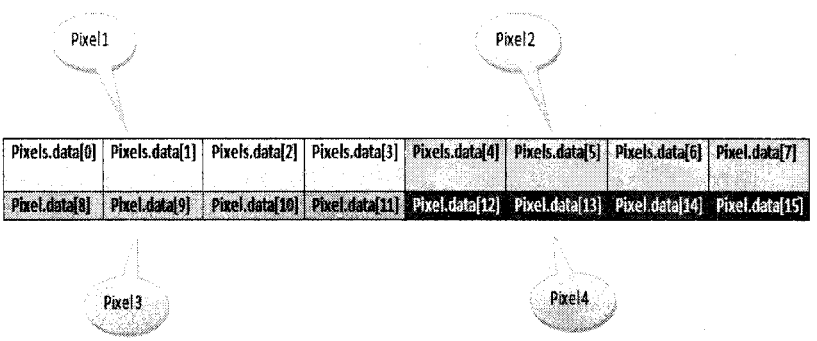


Рис. 1.5. Розташування значень кольорів пікселів

Розміри області пікселів знаходяться у змінних `pixel.width, pixel.height`. Приклади використання методів наведено у коді сторінки:

```
<html>
<head>
  <script type="application/javascript">
function init() {
var canvas = document.getElementById("canvas");
if (canvas.getContext)
{
var canvas = document.getElementById("canvas");
```

```
if (canvas.getContext) { var ctx =
canvas.getContext("2d");
ctx.fillStyle = "rgb(0,127,0)"; // Let's draw a
green square
ctx.fillRect(10,10,20,20);
var Pixel = ctx.getImageData(29,10,2,1);

alert("Pixel 1: "+ Pixel.data[0]+",
"+Pixel.data[1]+", "+ Pixel.data[2]+", "+
Pixel.data[3]);

alert("Pixel 2: "+ Pixel.data[4]+", "+
Pixel.data[5]+", "+ Pixel.data[6]+", "+
Pixel.data[7]);

alert("Pixels Width: " + Pixel.width); alert("Pixels
Height: " + Pixel.height);
}
}
  </script>
<title> HTML5 Canvas - Pixel Manipulation </title>
</head>
<body onload="init();">
<canvas id="canvas" width="900"
height="500"></canvas><br>
</body>
</html>
```

Дані зберігаються в асоціативних масивах (ключ-значення) і є доступними для команд конкретної web-сторінки. Сервер їх подає за запитом. Розмір даних не обмежується. Для зберігання є два типи об'єктів:

- `localStorage` – без дати завершення зберігання;
- `sessionStorage` – дані для однієї сесії.

```
localStorage.lastname="Smith";
document.getElementById("result").innerHTML="Last
name: "
+ localStorage.lastname;
```

У наведеному прикладі у об'єкті зберігається кількість натискань на кнопку:

```
if (localStorage.clickcount)
  {
```

```
localStorage.clickcount=Number(localStorage.clickcou
nt)+1;
  }
else
  {
  localStorage.clickcount=1;
  }
document.getElementById("result").innerHTML="You
have clicked the button " + localStorage.clickcount
+ " time(s).";
```

Об'єкт sessionStorage зберігає дані тільки протягом однієї сесії. Під час закриття вікна вони втрачаються:

```
if (sessionStorage.clickcount)
  {
sessionStorage.clickcount=Number(sessionStorage.clic
kcount)+1;
  }
else
  {
  sessionStorage.clickcount=1;
  }
document.getElementById("result").innerHTML="You
have clicked the button " +
sessionStorage.clickcount + " time(s) in this
session.";
```

1.5. Таблиці каскадних стилів (CSS)

1.5.1. Синтаксис каскадного стилю

Засоби каскадного стилю призначені для опису, подання, планування та вигляду інформації на web-сторінці, на противагу мові HTML, призначеній для опису змісту.

Стиль називається каскадним через керування атрибутами елементів у каскадному порядку:

- стилі браузера за замовчуванням;
- зовнішні файли стилів (<link>);
- внутрішні стилі (в межах тега <style> у заголовку);
- рядковий стиль (атрибути стилів елементів HTML).

Вони можуть бути розміщені в тексті коду сторінки або окремим файлом (раціональніший спосіб).

Базовим є синтаксис правила CSS

```
селектор {
    властивість: значення;
    властивість: значення;
    ...
    властивість: значення;
}
```

Наприклад,

```
p {
    font-family: sans-serif;
    color: red;
}
```

Файл CSS складається, як правило, з багатьох правил, кожне з яких починається із селектора. Останній прив'язується до елемента HTML і позначає властивості стилю подання даних.t

Під час створення окремого CSS-файла він приєднується до HTML коду тегом <link> у заголовку коду:

• Назвою без розширення

```
<link rel="stylesheet" type="text/css"
href="filename" />
```

• Файлом з розширенням

```
<link rel="stylesheet" type="text/css"
href="style.css" />
```

• Посиланням на сервер

```
<link rel="stylesheet" type="text/css"
href="http://www.google.com/uds/css/gsearch.css" />
```

Файлів може бути декілька. У разі конфлікту назв приймається останній.

Три способи керування кольором:

```
p { color: red; }
h2 { color: rgb(128, 0, 196); }
h4 { color: #FF8800; }
```

• Кольори можуть бути із списку : `aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white, yellow`

• RGB коди: червоний, зелений та голубий набувають значення від 0 до 255.

• !6-ві код: RGB значення від 00 (0) до FF (255)

Властивості шрифтів

• font-family: який шрифт;

• font-size: розмір шрифта;

• font-style: дозволити / заборонити стиль italic;

• font-weight: дозволити / заборонити стиль bold.

Приклади

• p { font-family: "Georgia"; };

• h2 { font-family: "Arial Narrow"; };

• p { font-family: "Garamond", "Times New Roman", serif; }.

Назви шрифтів:
serif, sans-serif, cursive, fantasy, monospace

Назва шрифту дасть змогу кожному комп'ютеру застосувати потрібний наявний шрифт

Керування розміром

Для розмірів використовують: xx-small, x-small, small, medium, large, x-large, xx-large

Відносні розміри: smaller, larger

Процентні розміри.: 90%, 120%

одиниці : пікселі абсолютні (px) . крапки (1/72 дюйма) (pt) m-розмір (p 1 em дорівнює розміру поточного шрифту) (em) – 16px, 16pt, 1.16em

• p { font-size: 14pt; }

• p { font-size: x-large; }

Керування товщиною та стилем:

```
p {
    font-weight: bold;
    font-style: italic;
}
```

Приклади інструкції, що буде застосована до всього тексту body

```
body { font-size: 16px; }
strong { font-weight: normal; color: red; }
```

```
em { font-style: normal; background-color: #DDDDDD;
}
```

Керування текстом

text-align: вирівнювання тексту : `left, right, center, justify`

text-decoration: декорування, наприклад, підкреслювання: `underline`, overline, blink

line-height, word-spacing, letter-spacing: проміжки між фрагментами тексту

text-indent: indents першу літеру параграфа

text-align

```
blockquote { text-align: justify; }
h2 { text-align: center; }
p { text-decoration: underline; }
```

Керування розміром

Для блочних елементів можна застосувати параметри (що ігноруються для рядкових елементів):

- width, height: ширина та висота елемента;
- max-width, max-height, min-width, min-height: максимальні чи мінімальні розміри елемента sion.

Приклади:

```
p { width: 400px; background-color: yellow; }
h2 { width: 50%; background-color: aqua; }
```

Коментарі CSS : /* ... */

Групування стилів для елементів:

- `p,h1,h2 { color: blue; }`
- `h2 { background-color: yellow; }`

Наведемо приклад HTML коду та відповідного йому дерева

```
<html><head><title>Моя сторінка</title></head>
<body><h1>Моя сторінка</h1>
<p>Мої улюблені Композитори :</p>
<ul><li>Петро Чайковський</li>
<li>Микола Лисенко</li>
<li>Джузеппе Верді</li>
</ul></body></html>
```

Наведемо коди керування сторінкою

```
body { font-family: sans-serif; background-color:
  yellow; }
```

```
p { color: red; background-color: aqua;
text-decoration: overline underline; }
li { font-weight: bold; text-align: center; }
```

Для перевірки команд керування стилем використовується W3C CSS Validator

```
<p><a href="http://jigsaw.w3.org/css-
validator/check/referer">
<img src="http://jigsaw.w3.org/css-
validator/images/vcss"
alt="Valid CSS!" /></a></p>
```

jigsaw.w3.org/css-validator/ перевіряє виконання офіційних специфікацій CSS

1.5.2. Класи

Для керування стилем CSS параграфа, що є частиною позначеного класу, використовують селектор класу

```
p.special {
    background-color: yellow;
    font-weight: bold;
}
```

Наведемо атрибут класу в HTML-коді

```
<p>Lviv City!  Lviv City!</p>
<p class="special">See our special proposals!</p>
<p class="special">Today only: New stadium in
Lviv.</p>
<p>We'll see entrance price!</p>
```

Тут два позначені параграфи підлягатимуть селектору `p.special`

Селектор класу без елемента належить до будь-якого позначеного як `standout` елемента

```
.standout {
    color: red;
    font-family: cursive;
}
<h2 class="standout">Lviv City!  LvivCity!</h2>
<p class="special">See our special proposals!</p>
!
```

Елемент може бути членом багатьох класів, розподілених у просторі, тоді використовується селектор CSS ID

```
p#mission {
    font-style: italic;
    font-family: "Garamond", "Century Gothic",
serif;
}
```

Цей селектор застосовується лише до параграфів, що мають відповідну ID-назву `mission` у цьому прикладі. Відрізняється від селектора класу, в якому ID використовується один раз у всьому HTML-документі.

```
<p>Lviv City!  Lviv City!</p>
<p id="mission">Our mission is to present our
customers <q>esplode</q> by addresses!</p>
```

Посилання можуть також містити ID на кінці зі знаком #
Браузер завантажить сторінку і перегляне елемент з заданим ID

```
<p>Visit <a href=
"http://www.textpad.com/download/index.html#download
s">
textpad.com</a> to get the TextPad editor.</p>
<p><a href="#mac">Directions for Mac OS X</a></p>
```

Логічні групи в HTML:

Секція – блок рядків на HTML-сторінці позначається `<div>`

```
<div class="standout">
<h2>Lviv City!  Lviv City!</h2>
<p class="special">See our special proposals!</p>
<p>We'll offer good  price!</p>
</div>
```

До тегу чи ID застосовуються стилі.

Рядкова секція:

```
<h2> Lviv City!  Lviv City!</h2>
<p>See our <span class="special">specialr</span>
proposals!</p>
<p>We'll offer <span class="standout">good
price</span>!</p>
```

Вбудований у заголовок фрагмент керування стилем: `<style>`

```
<head>
<style type="text/css">
```

```
p { font-family: sans-serif; color: red; }
h2 { background-color: yellow; }
</style>
</head>
```

Вбудована в рядок команда керування атрибутом

```
<p style="font-family: sans-serif; color: red;">
This is a paragraph</p>
```

1.5.3. Каскадні стилі для властивостей фону

Для керування фоном засоби CSS мають такі властивості:

background-color : заповнити фон кольором;

background-image : зображення у фоні;

background-position : розміщення bg image в елементі;

background-repeat : чи/як bg image має бути повторене;

background-attachment : чи bg image scrolls цю сторінку;

background : встановити властивості фону нашвидкуруч.

Приклади:

- background-image

```
body {
    background-image: url("draft.jpg");
}
```

- background-repeat : repeat-x, repeat-y, or no-repeat

```
body {
    background-image: url("draft.jpg");
    background-repeat: repeat-x;
}
```

- background-position

```
body {
    background-image: url("draft.jpg");
    background-repeat: no-repeat;
    background-position: 370px 20px;
}
```

Для позиції значення складається з двох позицій `top, left, right, bottom, center`, проценти або значення довжини в px, pt. Значеннння може бути від'ємним для зсуву ліворуч/вверх на задану величину

- Частина зображення

```
.partialimage1, .partialimage2 {
```

```
        background-image: url("sex_and_the_city.jpg");
        background-repeat: no-repeat;
        width: 70px;   height: 200px;
    }
    .partialimage1 { background-position: 0px 0px; }
    .partialimage2 { background-position: -115px 0px; }
```

1.5.4. Розширені можливості CSS

Контекстні селектори

```
selector1 selector2 {
    properties
}
```

– застосовуються властивості до selector2, якщо він розміщений у межах selector1 на сторінці

```
selector1 > selector2 {
    properties
}
```

– застосовуються властивості до selector2, тільки якщо він всередині selector1 на сторінці без проміжних тегів.

Приклади:

```
li strong { text-decoration: underline; }
        <p>Shop at <strong>Hardwick's
  Hardware</strong>...</p>
        <ul>
        <li>The <strong>best</strong> prices in
  town!</li>
        <li>Act while supplies last!</li>
        </ul>
```

Складніший приклад

```
    div#ad li.important strong { text-decoration:
underline; }
    <div id="ad">
    <p>Shop at <strong>Hardwick's
      Hardware</strong>...</p>
    <ul>
    <li class="important">The <strong>best</strong>
    prices in town!</li>
    <li>Act <strong>while supplies last!</strong></li>
    </ul>
    </div>
```

Псевдокласи:

```
a:link {color: #FF0000}    /* не відвідане посилання
a:visited {color: #00FF00} /* відвідане посилання */
a:hover {color: #FF00FF}   /* mouse over link */
a:active {color: #0000FF}  /* selected link */
```

- :active : активований або вибраний елемент;
- :focus : з фокусом на клавіатуру;
- :hover : миша над елементом;
- :link : посилання не реалізоване;
- :visited : посилання реалізоване;
- :first-child : породжений перший елемент.

Приклади з псевдокласами

```
a:link {color: red}
a:visited {color: green}
a:hover {color: purple; background-color: yellow;}
a:active {color: blue}
<a href="http://www.google.com">Gooooooogle</a>
```

Властивість `list-style-type`

```
ol { list-style-type: lower-roman; }
```

Можливі значення:

- `none : No marker`
- `disc (default), circle, square`
- `decimal : 1, 2, 3, etc.`
- `decimal-leading-zero : 01, 02, 03, etc.`
- `lower-roman : i, ii, iii, iv, v, etc.`
- `upper-roman : I, II, III, IV, V, etc.`
- `lower-alpha : a, b, c, d, e, etc.`
- `upper-alpha : A, B, C, D, E, etc.`
- `lower-greek : alpha, beta, gamma, etc.`
- `others: hebrew, armenian, georgian, cjk-ideographic, hiragana, katakana, hiragana-iroha, katakana-iroha`

Властивість `display`

```
h2 { display: inline; background-color: yellow; }
```

`display` – `none`, `inline`, `block`, `run-in`, `compact` вказує на тип CSS рамки, з якою показується елемент, (використовувати обережно, бо може порушити структуру сторінки).

Властивість `visibility` (visible (замовчування) hidden) вказує, чи виводити на екран

```
p.secret {
    visibility: hidden;
}
```

Порожнє місце може бути заповнене як реакція на подію.

Для виділення тексту застосовуються прямокутники з різними типами сторін (border) та проміжками всередині (padding) між вмістом та границями, полем (margin) між сторонами та вмістом назовні. Ширина width = ширина змісту + L/R padding + L/R border + L/R margin. Висота height = висота змісту height + T/B padding + T/B border + T/B margin

CSS має властивість для границь

- `border-color`, `border-width`, `border-style`: властивості границь:

товщина (px, pt, em, %, або : thin, medium, thick)

- `border-bottom`, `border-left`, `border-right`, `border-top`: всі властивості до конкретної сторони
- `border-bottom-color`, `border-bottom-style`, `border-bottom-width`, `border-left-color`, `border-left-style`, `border-left-width`, `border-right-color`, `border-right-style`, `border-right-width`, `border-top-color`, `border-top-style`, `border-top-width`: властивості границі на конкретній стороні
- `border`: всі властивості до чотирьох сторін з відповідними стилями : hidden, dotted, dashed, double, groove, inset, outset, ridge, solid

та кольорами, позначеними раніше до тексту та фону.

Наведемо приклад опису

```
h2 {
    border-left: thick dotted #CC0088;
    border-bottom-color: rgb(0, 128, 128);
    border-bottom-style: double;
}
```

Якщо пропустити певні властивості, то їхні значення приймаються за замовчуванням.

Для проміжків є такі властивості (кожна сторона задається окремо)

padding: проміжки на чотири сторони

`padding-bottom`: на нижню сторону;

`padding-left`: на ліву сторону;

`padding-right`: на праву сторону;

`padding-top`: на верхню сторону.

Приклад

```
p { padding: 20px; border: 3px solid black; }
h2 { padding: 0px; background-color: yellow; }
```

Заповнення набуває того самого кольору фону, що і елемент, наприклад,

```
p { padding-left: 200px; padding-top: 30px;
    background-color: fuchsia; }
```

Керують полями такі властивості:

`margin`: поля на чотири сторони;

`margin-bottom`: поля на чотири сторони;

`margin-left`: поля на ліву сторону;

`margin-right`: поля на праву сторону;

`margin-top`: поля на верхню сторону.

Приклад з полями (які завжди прозорі)

```
p {
    margin: 70px;
    background-color: fuchsia;
}
```

Або

```
p {
    margin-left: 200px;
    background-color: fuchsia;
}
```

Кожне поле задається окремо.

Приклад на розміри

```
p { width: 400px; background-color: yellow; }
h2 { width: 50%; background-color: aqua; }
```

- `width, height:` ширина і висота елементів
- `max-width, max-height, min-width, min-height:` максимальний та мінімальний розмір елемента в заданих розмірах. Вони застосовуються до блочних елементів, для рядкових ігноруються.

Центрування блочних елементів : auto margins

```
p { width: 500px; margin-left: auto; margin-right:
auto; }
```

Працює, коли встановлено width (в іншому разі елемент займає всю ширину сторінки).

Для центрування рядкових елементів всередині блочних елементів можна використати

```
text-align: center;
```

float може бути left, right, default

Блок float повинен мати заповнену властивість width. Без вказання значення ширини елемент може зайняти 100 % ширини сторінки.

Властивість `clear: left, right, both , default`

```
p { background-color: fuchsia; }
h2 { clear: right; background-color: yellow; }
```

не дає змоги плаваючому елементу перекриватись з іншими елементами.

Властивість `position` набуває значення

- `static:` за замовчуванням;
- `relative:` зміщення від статичного положення;
- `absolute:` на фіксованій позиції в межах охоплюваного елемента;
- `fixed:` на фіксованій позиції в межах об'єкта браузера window;
- `top, bottom, left, right` значення позиціонують кути прямокутника.

```
div#rightside {
    position: fixed;
```

```
    right: 10%;
    top: 36%;
}
```

Властивість `vertical-align` вказує, як рядковий елемент вирівнюється вертикально щодо змісту в межах прямокутника `top, middle, bottom, baseline (default), sub, super, text-top, text-bottom`, значення довжини або в %.

Наведемо приклад використання властивості `vertical-align`

```
<p style="background-color: yellow;">
<span style="vertical-align: top; border: 1px solid
red;">
Don't be sad!  Turn that frown
<img src="sad.jpg" alt="sad" /> upside down!
<img style="vertical-align: bottom" src="smiley.jpg"
alt="smile" />
Smiling burns calories, you know.
<img style="vertical-align: middle" src="puppy.jpg"
alt="puppy" />
Anyway, look at this cute puppy; isn't he adorable!
So cheer up,
and have a nice day.  The End.
</span></p>
```

Контрольні запитання

1. Вказати головні функції web-сервера.
2. Яке ПЗ необхідне для створення динамічних web-сторінок?
3. Які способи побудови web-сторінок?
4. Типи елементів HTML для занесення даних на web-сторінку?
5. Побудувати таблицю мовою HTML.
6. Роль параметрів середовища оточення, навести приклади.
7. Які основні елементи мови HTML5?
8. Навести посилання абсолютного та відносного типу на web-сторінці.
9. Побудувати посилання на зображення на web-сторінці.

10. Навести приклади створення текстового змісту на web-сторінці.

11. Навести приклади з елементами керування розміщення тексту на web-сторінці.

12. Які типи вузлів у дереві DOM?

13. Для простого прикладу web-сторінки намалюйте дерево DOM.

14. Які властивості вузлів DOM-дерева?

15. Які методи вузлів дерева DOM?

16. Які об'єкти DOM?

17. Що таке форми зворотного зв'язку HTML?

18. Які основні елементи форми HTML?

19. Якими методами пересилається запит?

20. Як надсилаються параметри і що їх приймає на сервері?

21. Навести приклади фрагмента скрипту для перевірки даних, внесених у форму.

22. Як внести зміни у дерево DOM?

23. Навести приклади внесення змін у XHTML-сторінку.

24. Засоби малювання та опрацювання зображень у HTML5.

25. Програма побудови гістограми зображення.

26. Який синтаксис CSS?

27. Різні способи приєднання CSS.

28. Керування в CSS шрифтами. Навести приклади.

29. Керування в CSS фоном. Навести приклади.

30. Які групи властивостей у CSS-технології можна виділити?

31. Чому треба розділяти зміст і стилі CSS?

32. Як керувати вирівнюванням тексту засобами CSS? Навести приклади.

33. Як керувати полями у стилі CSS? Навести приклади.

34. Які властивості пов'язані з border?

Глава 2
ТЕХНОЛОГІЇ JAVASCRIPT ТА JQUERY

2.1. Особливості програмування мовою JavaScript

Одним з інструментів підтримки динамічних сценаріїв при перегляді Web-сторінок у межах комп'ютера користувача є мова програмування JavaScript – спрощений варіант мови програмування Java. Нею запезпечується рух об'єктів на сторінці, введення та виведення параметрів, зміна зображень вікон тощо. Програми модифікації, створення гіпертекстових сторінок традиційно називають скриптами (scripts), які інтерпретуються програмою перегляду. Спосіб базується на ідеології об'єктно-орієнтованого програмування. Зупинимось на скриптах, написаних мовою JavaScript.

Мова програмування для реалізації інтерактивності web-сторінок: введення тексту, реакції на події, отримання інформації з сервера, здійснення обчислень, підлягає стандартизації, але не підтримується всіма браузерами. Javascript інтерпретується на браузері та інтегрується за змістом HTML/CSS. Наведемо приклад внесення динамічного тексту:

```
document.write("message");  -друкує текст на сторінці
```

Ввімкнення коду Javascript в HTML можливе трьома способами:

1. У тіло сторінки (виконується під час завантаження):

```
<body>
    ...
    <script type="text/javascript">
    Javascript code
    </script>
    ...
</body>
```

2. У заголовок сторінки (виконується як реакція на подію (викликом)):

```
<head>
```

```
    ...
    <script type="text/javascript">
    Javascript code
    </script>
    ...
</head>
```

3. Посиланням на зовнішній .js файл (розміщений у заголовку чи тілі):

```
<script src="filename"
type="text/javascript"></script>
<script src="example.js"
type="text/javascript"></script>
```

Розглянемо приклади типів даних та команд мови програмування. Тип змінної визначається за присвоєним значенням. Перепризначення значення об'єктів робиться операціями присвоювання. Оголошення змінної здійснюється ключовим словом `var` (`var text = "text"`).

В `JavaScript` реалізовані всі типи операторів мов програмування : +,-,*, /, %, >>,<<, +=, -=, ... До того ж оператор "+" під час роботи з рядками означає конкатенацію:

```
s = "string1"+"string2";
```

Операторами структурного програмування є :

- *break* – примусовий вихід з циклу

```
while(i &lt 6)
{       if(i==3) break;   }
```

- *continue* – перехід на кінець циклу

```
while(i &lt 6)
{       if(i==3) continue;        }
```

- *for* – цикл

```
for(i=0;i<9;i++)
{       ...      }
```

- *for* – цикл властивостей об'єкта (змінних класу)

```
for(i in obj)
{       str = obj[i]      }
```

- *if..else* – умовний оператор

```
if(i>0)
```

```
{          ...          }
else
{          ...          }
```

- *while* – умовний цикл

```
while(j==k)
{          j++;          k--;          }
```

Зазначені оператори не містять повного переліку операторів JavaScript, але їх достатньо для виконання практичних завдань.

Тип Array дає можливість маніпулювання множинами як об'єктів сторінки, так і нових створених:

```
new_array = new Array();
new_array = new Array(5);
colors = new Array ("red","white","blue");
```

Розмірність масиву може змінюватися динамічно, наприклад,:

```
colors = new Array();
colors[5] = "red";
```

У цьому разі масив colors складається з 6 елементів (перший елемент масиву має індекс 0). Для масивів визначені три методи: join, reverse, sort. Наприклад,

```
colors = new Array("red","white","blue");
string = colors.join("+");
string = "red + white + blue";
```

Метод reverse змінює порядок елементів масиву на зворотний, sort – сортує елементи масиву в порядку зростання властивостей. Масив має дві властивості: length і prototype.

Поряд з масивами програми JavaScript використовують вбудовані масиви, наприклад, зображення (Images) чи гіпертекстові вказівки (Links).

Важливим елементом мови є події, які використовуються для виконання частин програмного коду скрипту, наприклад, onLoad="Scroll();". До найвживаніших можна зарахувати такі:

- onLoad – виконання скрипту чи функції під час завантаження;
- onChange – породжується під час зміни значення елементу форми;

- onClick – породжується під час вибору об'єкта (button, checkbox і т.п.);
- onSelect – породжується під час вибору текстового об'єкта (text, textarea);
- onSubmit – під час натискання на кнопку Submit;
- onUnload – під час переходу до іншої сторінки.

Розглянемо приклади використання вбудованих об'єктів.

```
new_image = new Image();new_image = new Image
(width,height);
```

Об'єкт Image має низку властивостей, зокрема, назва файла:

```
img_array[0].src = "image1.gif"
```

Усі операції в програмі на JavaScript описують дії над об'єктами – елементами робочої області броузера та контейнерами мови HTML. Об'єкти мають властивості та методи. Існують також інші функції, які дають змогу працювати зі стандартними математичними функціями та керувати процесом виконання програми. JavaScript має механізм опрацювання подій перегляду динамічних об'єктів та управління багатовіконним інтерфейсом. Об'єкти JavaScript беруть свій початок від класу Window. Ієрархію основних об'єктів подано на рис. 2.1.

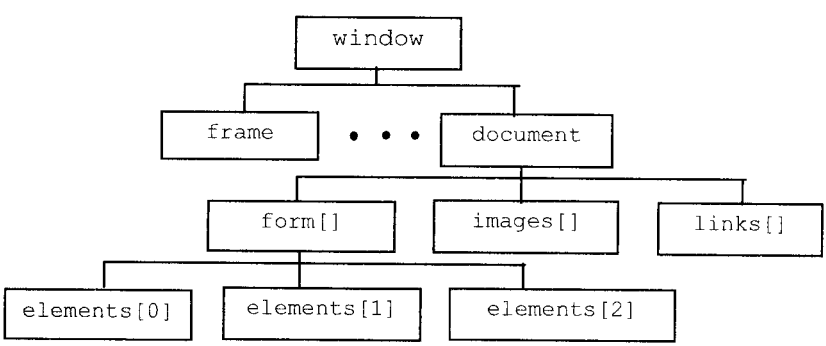


Рис. 2.1. Ієрархія об'єктів у мові JavaScript

Доступ до об'єктів та їх властивостей можливий за їх індексом або присвоєною властивості name назвою, наприклад:

```
place=document.forms[0].elements[0].value;
place=document.Myforms.myname.value;
```

де відповідним об'єктам присвоєні назви Myforms та myname.

Крім цих класів об'єктів, користувач може створювати і власні.

Об'єкт типу Image може бути названий як тег IMG HTML, після чого до нього можна звертатися на ім'я. Якщо Image використовується всередині форми, то він є властивістю цієї форми і для звертання до нього треба використовувати складні імена:

```
<img name="car" src="car.gif">
    document.car.src = "car1.gif"
```

Друга форма звертання до Image здійснюється за індексом, наприклад:

```
document.images[1].src = "car1.gif";
```

Розглянемо декілька прикладів написання функцій мовою програмування JavaScript.

Функція мультиплікації – об'єкти зображень:

```
function multipulti() {
  img_array = new Array();
  img_array[0] = new Image(50,100) ;
          ....
  img_array[99] = new Image(50,100) ;
  img_array[0].src = "image1.gif";
          ...
  img_array[99].src = "image100.gif";
  n=0;
          while(n==0;)
          {
          document.images[0].src =
          img_array[0].src;
          ...
          }
}
```

Функція прокрутки тексту у вікні – об'єкти вікна та документа. В цьому прикладі наведено повний текст HTML-

сторінки. JavaScript код знаходиться між тегами <SCRIPT LANGUAGE="JavaScript"> ..... </SCRIPT>, а звертання до нього відбувається на основі команди опрацювання події onLoad="Scroll();":

```
<html>
<head>
<title>A DHTML Scrolling Window</title>
<script language="JavaScript">
var pos=100;
function Scroll() {
  if (!document.getElementById("thetext")) return;
  obj=document.getElementById("thetext");
  pos -=1;
  if (pos < 0-obj.offsetHeight+130) return;
  obj.style.top=pos;
  window.setTimeout("Scroll();",30);
}
</script>
</head>
<body onLoad="Scroll();">
<h1>Приклад вікна з рухомим текстом</h1>
<div id="thewindow"
style="position:relative;width:180;height:150;
  overflow:hidden; border-width:2px; border-
style:solid; border-color:red">
<div id="thetext"
style="position:absolute;width:170;left:5;top:100">
<p>Перший параграф</p>
<p>Тут містяться тільки теги HTML, наприклад, <a
href="http://www.starlingtech.com/dhtml/">Посилання<
/a>.</p>
<p>Довільний параграф </p>
<ul><li>Приклад списку</li></ul>
<p>Текст Завершено </p>
<p><b>[<a href="javascript:pos=100;Scroll();">Почати
знову</a>]</b></p>
</div>
</div>
</body>
</html>
```

2.2. Додаткові можливості програмування на JavaScript

Розглянемо деякі не вказані раніше додаткові можливості мови програмування JavaScript, які полегшують опрацювання даних.

Змінні з назвами, чутливими до регістра:

```
var name = value;
var clientName = "Connie Client";
var age = 32;
var weight = 137.4;
```

Типи даних не вказуються, але підтримуються.

Змінні типу `String` :

```
var clientName = "Connie Client";
```

Тип володіє методами:

charAt (повертає String тип), indexOf, lastIndexOf, replace, split, substring, toLowerCase, toUpperCase

```
var fName = s.substring(0, s.indexOf(" "));
var cNlength=clientName.length ;
```

Можна використовувати лапки двох типів: " " або ' '.

Тип `String` має послідовності символів, як у Java

```
\' \" \& \n \t \\
```

Наведемо приклад перетворення на рядковий тип:

```
var s = String(myNum);
var s = count + " bananas,  ah!"
```

Числовий (Number) тип (зберігається у слові на 64 біти) :

```
var enrollment = 99;
var median142Grade = 2.8;
```

Перетворення `String в Number`:

```
var integerValue = parseInt("String");
var floatValue = parseFloat("String");
var integerValue = parseInt("123hello")//-повертає 123
var integerValue = parseInt("booyah");// повертає NaN
(не число)
```

Логічний тип (Boolean):

```
        var iLike190M = true;
if ("Marty is great") {    // true
    ...
}
```

0, NaN, "", null та невизначені типи є false

Перетворення значення до логічного типу в

```
var boolValue = Boolean(otherValue);
```

Ключові оператори слова Javascript:

+ – * / % ++ -- = += -= *=

/= %= == != > < >= <= && || !

== перевірка значення ("5.0" == 5 є true)

=== перевірка типу ("5" === 5 єfalse)

Приклади об'єктів:

Math

```
var rand1to10 = Math.floor(Math.random() * 10 + 1);
var three = Math.floor(Math.PI);
```

Об'єкт має методи :

```
abs, ceil, cos, floor, log, max, min, pow, random,
round, sin, sqrt, tan  Та властивості E, PI
```

Коментарі :

// рядковий коментар

/*

Багаторядковий коментар

*/

Функції (тип параметрів та поверненого значення не вказується)

```
function name(parameterName, ..., parameterName) {
    statements;
}
function quadratic(a, b, c) {
    return -b + Math.sqrt(b*b - 4*a*c) / (2*a);
}
```

Функції, що не повертають значень, повертають невизначені типи. Змінні у функціях мають локальний характер.

Виклик функцій

```
name(parameterValue, ..., parameterValue);
var root = quadratic(1, -3, 2);
```

Зайві параметри ігноруються, ті, яких бракує, доповнюються невизначеними типами.

Глобальні та локальні змінні:

```
var count = 1;
```

```
function f1() {
    var x = 999;
    count *= 10;
}
function f2() { count++; }

f2();
f1(); // 12
```

Змінна `count` глобальна, x – локальна

Вікна `Popup` :

```
alert("message");      // message
confirm("message");    // повертає true або false
prompt("message");     // повертає стрічку користувача
```

Об'єкти дати Date :

```
var today = new Date();                    // сьогодні
var midterm = new Date(2013, 4, 4);    // Квітень 4,
2013
```

Має методи :

```
getDate, getDay, getMonth, getFullYear, getHours,
getMinutes, getSeconds, getMilliseconds, getTime,
getTimezoneOffset, parse, setDate, setMonth,
setFullYear, setHours, setMinutes, setSeconds,
setMilliseconds, setTime, toString
```

Зауваження:

`getYear` повертає рік – дві цифри (getFullYear -4)

`getDay` повертає день тижня 0 (Неділя) – 6 (Субота)

`getDate` повертає день місяця

`Date` містить місяці 0-11 (не 1-12)

Масиви (Arrays) та способи їх ініціалізації:

```
var stooges = new Array();
stooges[0] = "Larry";
stooges[1] = "Moe";
stooges[2] = "Curly";
var stooges = new Array("Larry", "Moe", "Curly");
var stooges = ["Larry", "Moe", "Curly"];
```

Об'єкти масиву мають такі методи:

```
concat, join, pop, push, reverse, shift, slice,
sort, splice, toString, unshift
```

та властивість `length`

Приклад використання масиву як списку:

```
var a = new Array();
a.push("Morgan");           // Brian
a.push("Brian");            // Morgan,Brian
a.unshift("Kenneth");       // Kenneth,Morgan,Brian
a.push("Helene", "Jeff");   //
Kenneth,Morgan,Brian,Helene,Jeff
a.shift();                  //
Morgan,Brian,Helene,Jeff
a.pop();                    // Morgan,Brian,Helene
a.sort();                   // Brian,Helene,Morgan
```

`push pop add` – додати чи забрати елемент з кінця;

`unshift shift` – додати чи забрати елемент з початку.

Рядки і масиви можна розділяти чи об'єднувати:

var s = "the quick brown fox";

```
var a = s.split(" ");          //
[the,quick,brown,fox]
a.reverse();                   //
[fox,brown,quick,the]
s = a.join("!");               //
"fox!brown!quick!the"
```

`split` розділяє рядок на фрагменти, вставляючи розділювач;

`join` групує масив рядків у єдиний рядок, вставляючи розділювач.

Спеціальні значення: undefined: якщо не оголошено, null: якщо оголошено

```
var ned;
var benson = 9;
//   ned є null
//   benson є 9
```

Тип функції `typeof(value)`:

```
function foo() { alert("Hello"); }
var a = ["Huey", "Dewey", "Louie"];
The following statements are true:
typeof(3.14) == "number"
typeof("hello") == "string"
typeof(true) == "boolean"
typeof(foo) == "function"
typeof(a) == "object"
typeof(null) == "object"
```

```
typeof(undefined) == "undefined"
Timers: setTimeout, clearTimeout
function delayedMessage() {
var myTimer = setTimeout("alert('Booyah!');", 5000);
}
<h2 onclick="delayedMessage();">Click me now!</h2>
```

Функція `setTimeout` виконує код функції після заданої кількості мілісекунд повертає об'єкт таймера..

Вимкнути таймер можна функцією `clearTimeout(myTimer);`

```
function repeatedMessage() {
var myTimer = setInterval("alert('Rudy!');", 1000);
}
<h2 onclick="repeatedMessage();">Click me now!</h2>
```

Загальна помилка – це визначення таймера за локальною змінною:

```
function delayed(text) {
    var myTimer = setTimeout("alert(text);", 1000);
}
<h2 onclick="delayed('hello');">Click me now!</h2>
```

Масиви аргументів

```
function example() {
    for (var i = 0; i < arguments.length; i++) {
        alert(arguments[i]);
    }
}
```

Приклад виклику : `example("how", "are", "you");`

Масив як асоціативний (індекси – не цілі):

```
var map = new Array();
map[42] = "the answer";
map[3.14] = "pi";
map["champ"] = "suns";
...
```

Доступ до елементів асоціативного масиву за циклом `"for each":`

```
for (var name in arrayOrObject) {
    do something with arrayOrObject[name];
}
```

2.3. Опрацювання подій у Javascript

Мова HTML має елементи із спеціальними атрибутами – подіями. Функції Javascript використовують як обробники цих подій.

`onclick` – одна з основних подій HTML.

Обробники подій у старому стилі:

```
<body>
<button id="ok" onclick="okay();">Click me</button>
...
// викликається під час натискання кнопки OK
function okay() {
    var button = document.getElementById("ok");
    button.style.color = "red";
}
```

Обробник з елементами DOM

```
<body>
<button id="ok">Click me</button>
...
window.onload = initializeBody();     // глобальний код
// викликається під час завантаження сторінки, описує обробника події
function initializeBody() {
    document.getElementById("ok").onclick = okay;
}
function okay() {
    this.style.color = "red";
}
```

Javascript краще використовувати з розділенням змісту сторінки на три категорії:

- зміст (HTML),
- представлення (CSS),
- поведінка (Javascript).

Тоді сторінка не забивається кодом опрацювання та стилістикою.

Подія window.onload = name();
// встановлює обробника події

```
function name() {
    event handler
}
```

Схема опрацювання події така:

```
window.onload = initializeBody();
      function initializeBody() {
          var name = document.getElementById("ok");
          name.event = function;
          ...
      }
```

Події миші.

- Для клікання на мишу DOM об'єкти мають такі властивості:

onmousedown: натискання;

onmouseup: відпускання;

onclick: натискання – відпускання;

ondblclick: подвійне клікання.

- Під час руху:

onmouseover : курсор миші входить на елемент;

onmouseout : курсор миші виходить з елемента;

onmousemove : курсор миші рухається у межах об'єкта;

```
myElement.onmousemove = myFunction();
```

Приклад подій від миші

```
<div id="dare">Click me ... I dare you!</div>
function initializeBody() {
   document.getElementById("dare").onmousedown =
colorIt;
}
function colorIt() {
    this.style.backgroundColor = "red";
}
```

В обробнику ключове слово `this` позначає той самий елемент `document.getElementById()`

```
<div id="dare">Click me ... I dare you!</div>
function initializeBody() {
    var dareDiv = document.getElementById("dare");
    dareDiv.onmousedown = colorIt;
    dareDiv.onmouseup = uncolorIt;
}
  function colorIt() {
  this.style.backgroundColor = "red";
  }
```

```
function uncolorIt() {
this.style.backgroundColor = "transparent";
}
```

або

```
function colorIt(event) {
    this.style.backgroundColor = "red";
    this.innerHTML = "You clicked (" + event.screenX
+
        ", " + event.screenY + ")");
}
```

Властивості об'єкта події:

```
type :  "click" чи "mousedown" ;
same : назва корисна під час опрацювання багатьох
подій;
clientX, clientY : координати зверху наліво ;
screenX, screenY : координати зверху ліворуч на екрані;
```

Якщо події в браузерах несумісні, то можна використати елементи стандартизації:

```
function handleClick(event) {
    event = standardizeEvent(event);
    ...
}
// направити несумісність
function standardizeEvent(event) {
    var e = event || window.event;
    e.srcElement = e.srcElement || e.target;
    return e;
}
```

Розглянемо приклад:

```
<div id="gum"><img src="gum.png" alt="gum" />
Double your pleasure, <em>double your
fun</em>!</div>
function initializeBody() {
   document.getElementById("gum").ondblclick =
doBorder;
}
function doBorder(event) {
    event = standardizeEvent(event);
    event.srcElement.style.border = "2px dashed
blue";
}
```

Клавіатура події – властивості:

- onkeydown: натискання клавіші для елемента у фокусі;
- onkeyup: відпускання клавіші для елемента у фокусі;
- onkeypress: натискання і відпускання клавіші для елемента у фокусі;
- onfocus: елемент у фокус ставить клавіатуру;
- onblur: елемент втрачає фокус на клавіатуру.

Властивості такі: keyCode: ASCII значення клавіші, перетворення до букви: String.fromCharCode(event.keyCode)

Список значень клавіші

- altKey: true, якщо Alt key тримається;
- ctrlKey: true, якщо Ctrl key тримається;
- shiftKey: true, якщо Shift key тримається.

Приклад одержання занесених даних

```
    // обробник події  window.onload
document.getElementById("mytextbox").onkeypress =
keyPress;
function keyPress(event) {
    event = standardizeEvent(event);
    if (event.keyCode == 13) {
        // коде 13 -  Enter
        do something;
    }
}
```

Події Tex box для тегу <input type="text">, <textarea>

- onselect: текст text box вибрано;
- onchange: зміст у text box змінився.

Події браузера:

- *onload*: завантаження;
- *onunload*: сторінка існує;
- *onresize*: вікно браузера змінюється;
- *onerror*: під час завантаження документа чи зображення сталася помилка.

Під час програмування основними помилками є:

• Неправильні оператори та команди

```
window.onload = initalizeBody;    // помилка
...
function initializeBody() {
    var theDiv =
document.getElementById("puzzlearea");
    ...
}
```

• Неправильне розташування дужок

```
function foo() {
    ...
function bar() {
    ...
}
```

• () в обробнику подій (під час під'єднання обробника подій подається тільки його назва)

```
myObject.onclick = handleClick();    // неправильно!
myObject.onclick = handleClick;      // правильно
```

Поставивши дужки () після назви функції, її викликають відразу.

2.4. Використання JQuery для опрацювання змісту сторінки

`JQuery` – це бібліотека `JavaScript` функцій підтримки зміни змісту на web-сторінці: для опрацювання подій, анімації, модифікації змісту тегів HTML документів та спрощення взаємодії з сервером за технологією `Ajax`. Технологія `JQuery` спрощує програмування сценаріїв засобами мови `JavaScript`.

Синтаксис `JQuery` подібний до синтаксису стилів CSS (підтримуються всі селектори), а саме: складається з селектора (вказівника) на елемент HTML та функції для виконання певних дій над атрибутами цього елемента: `$(selector).action()`

Приклади:

`$(this).hide()` – заховати елемент,

`$("p").hide()` – заховати всі параграфи,

`$("p.test").hide()` – заховати параграфи класу (`class=«test»`),

`$("#test").hide()` – заховати елементи з `id="test"`.

Функція є бібліотечна або описана відразу після її оголошення. Функція може використовуватись для всього тіла сторінки без вказування конкретного селектора.

Наведемо приклад використання селектора та простої побудованої функції:

```
$(document).ready(function() {
     alert("document is ready");
     });
```

Ця інструкція використовується у разі завантаженого HTML-документа та в готовому DOM навіть при недовантажених графічних об'єктах. Вона позначає функцію для виконання над об'єктами сторінки. В цьому випадку видається повідомлення після завантаження сторінки.

Помилка виникає, якщо:

- елемент не існує,
- зображення не завантажено тощо.

Для підімкнення бібліотеки в HTML-коді необхідно використати теги:

- `<script type='text/javascript' src='jquery.js'></script>`
- `<script type='text/javascript' src='jquery.min.js'></script>`
- `<script type="text/javascript" src="/jsz/jqModal.js"></script>`
- `<script type="text/javascript" src="/jsz/cache/03565abe.js"></script>`
- `<script type="text/javascript" src="/jsz/adriver.core.2.js"></script>`

Усі селектори погруповані в бібліотеках методів: `Query Selectors, jQuery Events, jQuery Effects, jQuery Callback, jQuery HTML, jQuery CSS, jQuery AJAX.`

Розглянемо приклади представників селекторів з різних бібліотек jQuery, які є обов'язковими елементами інструкцій на виконання.

Синтаксис	Опис
$("*")	вибір усіх елементів
$("p")	вибір усіх <p> елементів
$("p.intro")	вибір усіх <p> елементів з class="intro"
$("p#intro")	вибір перших <p> елементів з id="intro"
$(":animated")	вибір усіх анімованих елементів
$(":button")	вибір усіх елементів <button>, зокрема. для <input>
$(":even")$(":odd")	вибір парних (непарних) елементів
$(this)	Вибирає поточний HTML-елемент
$("p#intro:first")	Вибір першого елемента <p> з id = "intro"
$(".intro")	Вибір усіх елементів з класом = "intro"
$("#intro")	Вибір першого елемента з id = "intro"
$("ul li:first")	Вибір першого елемента Перший
$("ul li:first-child")	Вибір першого елемента кожного
$("[href]")	Вибирає всі елементи з атрибутом HREF
$("[href$='.jpg']")	Вибирає всі елементи з атрибутом HREF, який закінчується «. jpg»
$("[href='#']")	Вибирає всі елементи з HREF, значення дорівнює «#»
$("[href!='#']")	Вибирає всі елементи з HREF, значення НЕ дорівнює «#»
$("div#intro .head")	Вибирає всі елементи з класом = «head» у елемент з id = "intro")

Нижче в таблиці наведемо головні методи з бібліотеки опрацювання подій jQuery Events:

Метод	Опис
`$(document).ready(function)`	Пов'язує функцію з готовим документом
`$(selector).click(function)`	Пов'язує функцію з подією у певному місці
`$(selector).dblclick(function)`	Пов'язує функцію з подією у певному місці (подвійне натискання)
`$(selector).focus(function)`	Пов'язує функцію з фокусом на елементі
`$(selector).mouseover(function)`	Пов'язує функцію з подією у певному місці (курсор на об'єкті)

Повний список методів можна знайти на ресурсі http://www.w3schools.com/jquery/jquery_ref_events.asp

Нижче в таблиці наведемо деякі головні методи бібліотеки jQuery Effects для створення графічних ефектів:

Функція	Опис
`$(selector).hide()`	Вибраний елемент ховається
`$(selector).show()`	Вибраний елемент з'являється
`$(selector).toggle()`	Перемикання стану вибраного елемента (між hide і show)
`$(selector).slideDown()`	Показ вибраного елемента згори донизу
`$(selector).slideUp()`	Закриття вибраного елемента знизу догори
`$(selector).slideToggle()`	Перемикання стану вибраного елемента (між slide-up і slide-down)
`$(selector).fadeIn()`	Закриття вибраного елемента знизу догори
`$(selector).fadeOut()`	Закриття вибраного елемента знизу догори
`$(selector).fadeTo()`	Закриття вибраного елемента до заданого рівня
`$(selector).animate()`	Анімувати вибраний елемент

Повний список методів можна знайти на ресурсі http://www.w3schools.com/jquery/jquery_ref_effects.asp .

Параметр «Speed» методів SlideDown(speed, callback), SlideUp(speed, callback), SlideToggle(speed, callback) приймає значення: slow, normal, fast, milliseconds (100, 500, 1000, etc).

Функція callback викликається після завершення роботи методу (назва функції).

Головні методи з бібліотеки jQuery HTML для маніпуляції з HTML-елементами наведено в таблиці.

Функція	Опис
$(selector).html(content)	Зміна змісту вибраного елемента (inner) HTML
$(selector).append(content)	Додавання змісту до вибраного елемента (inner) HTML
$(selector).after(content)	Додавання змісту після вибраного елемента HTML

Повний список методів для маніпуляції змістом тегів HTML можна знайти на ресурсі http://www.w3schools.com/jquery/jquery_ref_html.asp

Головні методи з бібліотеки jQuery CSS для маніпуляції з CSS елементами наведені в таблиці:

CSS Властивості	Опис
$(selector).css(name)	Отримати значення властивості стилю елемента
$(selector).css(name,value)	Встановити значення однієї властивості стилю для відповідних елементів
$(selector).css({properties})	Встановити кілька властивостей стилю для відповідних елементів
$(selector).height(value)	Встановити висоту відповідних елементів
$(selector).width(value)	Встановити ширину відповідних елементів

Повний список методів для маніпуляції змістом тегів HTML можна знайти на ресурсі
http://www.w3schools.com/jquery/jquery_ref_css.asp

Розглянемо приклад побудови функції – реакції на завантаження сторінки. Своєю чергою, реакція реалізована функцією заміни тексту параграфів під час натискання кнопки:

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="jquery.js"></script>
<script type="text/javascript">
$(document).ready(function(){
$("button").click(function(){
$("p").html("Welcome to jQuery");
  });
});
</script>
</head>
<body>
<h2>This is a heading</h2>
<p>This is a paragraph.</p>
<p>This is another paragraph.</p>
<button>Click me</button>
</body>
</html>
```

2.5. Розширення бібліотеки функцій jQuery засобами plugins

jQuery дає змогу доповнювати бібліотеку методів власними функціями для використання їх під час розроблення динамічних web-сторінок. Останні називаються вбудованими інструментами (plugin) для маніпуляцій над DOM-вузлами за допомогою ключового слова 'this' і так приєднуються функції.

Послідовність методів утворює ланцюжок методів:

```
$("div")
```

```
    .parent()
    .css("heigth", "10px")
    .fadeTo(0, 0.5)
    .addClass("divOwner");
```

Зазвичай, javascript-плагіни розташовуються в окремих файлах. Для файлів з плагінами jQuery існує угода щодо їх назви – вона має задовольняти формату jquery.plaginName.js. Наприклад, файл потрібно назвати jquery. reverseText.js.

Є два способи формування методів:

1. Якщо задати функцію через jQuery, то до неї можна звернутися тільки через глобальний об'єкт jQuery. У такому випадку контекст функції вказуватиме на глобальний об'єкт з об'єкта window:

```
    jQuery.sayHello = function() {
  alert('Привіт! Знайдено ' + this.length + 'елементів'
);
}
$('div').sayHello(); // тут не відбувається нічого
```

Нижче правильно :

```
    jQuery.sayHello = function(elem) {
  alert('Привіт! Знайдено ' + elem.length + 'елементів');
}
jQuery.sayHello($('div')); //подається повідомлення
```

2. Якщо задати функцію через jQuery.fn, то вона буде працювати з елементами, знайденими селектором і функцією $ (). Контекст цієї функції міститиме вибрані елементи:

```
    jQuery.fn.sayHello = function() {
  alert('Привіт! Знайдено ' + this.length + 'елементів');
}

$('div').sayHello(); //повідомлення : Привіт! Знайдено XX елементів,
```

Так, якщо потрібно написати плагін, який буде працювати тільки з вибраними об'єктами, то потрібно його створювати в jQuery.fn, а якщо неважливо, які елементи сторінки були обрані селектором, то краще створювати функцію через jQuery.

Поданий нижче плагін – це метод друку текстового рядка у зворотній послідовності. Розглянемо два способи оголошення такої функції.

З неявним позначенням параметрів, які подаються пізніше:

```
Query.fn.reverseText = function(params) { ... };
```

або з конкретними параметрами (наприклад, `minlength` та `maxlength`).

Щоб уникнути можливих конфліктів назв у плагіні, використовуючи знак $, код охопюється у конструкцію (`function ($) {}`) (`jQuery`) (подробиці див. в описі `jQuery.noConflict ()`):

```
(function($) {
$.fn.reverseText = function(minlength, maxlength) {
... };
})(jQuery);
```

Тоді виклик здійснюється так:

```
$("p").reverseText(0, 100);
```

view

У першому способі для опису параметрів використовується функція `jQuery extend` :

```
params = $.extend( {minlength: 0, maxlength: 99999},
 params);
```

Отже, `params.minlength` є 0 і `params.maxlength` є 99999 поки не перевизначені змінні.

Тепер подамо безпосередньо код функції подачі тексту у зворотній послідовності:

```
(function($) {
  $.fn.reverseText = function(params) {
  params = $.extend( {minlength: 0, maxlength: 99999},
params);
  this.each(function() {
  var $t = $(this);
  var origText = $t.text(), newText = '';
  if (origText.length >= params.minlength &&  origText.
length <= params.maxlength) {
  for (var i = origText.length-1; i >= 0; i--
) newText += origText.substr(i, 1);
    $t.text(newText);
     }
  }); return this;
  };
  })(jQuery);
```

Змінна this в тілі методу містить поточний об'єкт jQuery (той, на якому і був викликаний метод). Для того, щоб обійти всі вибрані елементи окремо, використовуємо метод each(). Для можливості продовжити ланцюжок методів метод повинен повернути поточний об'єкт jQuery.

Цей файл зберігається під назвою jquery.reversetext.js і викликається у HTML-сторінці після завантаження бібліотеки jQuery:

```
<!DOCTYPE html PUBLIC "-
//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/str
ict.dtd">
<html lang="en">
<head>
<title>jQuery plugin: reverseText demonstration</tit
le>  </head>
<body>
<h1>jQuery plugin: reverseText</h1>
<p>This jQuery plugin reverses all the text in the s
elected nodes.</p>
<ul>
<li>This text will be reversed</li>
<li>This text will not be reversed</li>
<li>reversed</li>
<li>not reversed</li>
</ul>
<script type="text/javascript" src="jquery-
1.3.2.min.js"></script>
<script type="text/javascript" src="jquery.reversete
xt.js"></script>
<script type="text/javascript">
// reverse even-numbered LI tag text
$("ul li:even").reverseText();
</script>
</body>
</html>
```

Перший і третій пункти списку будуть записані у зворотній послідовності.

Контрольні запитання

1. Які типи даних використовуються в JAVASCRIPT? Навести приклади.
2. Які оператори структурного програмування використовуються в JAVASCRIPT? Навести приклади.
3. Масив у JAVASCRIPT. Навести приклади.
4. Переваги та недоліки масивів у JAVASCRIPT. Навести приклади.
5. Які об'єкти є в JAVASCRIPT?
6. Порівняльний аналіз операторів циклів у JAVASCRIPT.
7. Які події опрацьовують програми JAVASCRIPT?
8. Події миші, клавіатури.
9. Порівняти JAVASCRIPT та інші подібні мови.
10. Мовою JAVASCRIPT створити програму контролю заповненості анкети форми.
11. Розробити програму доступу до елементів DOM засобами мови JAVASCRIPT.
12. Розробити програму корекції елементів вузлів дерева DOM засобами мови JAVASCRIPT.
13. Навести приклади об'єктів мови JAVASCRIPT.
14. Які бібліотеки є в JQUERY?
15. Приклади методів з бібліотек JQUERY HTML, Events, Effects.
16. Типи та оболонка вбудованих методів JQUERY.

Глава 3
ТЕХНОЛОГІЯ AJAX

3.1. Об'єкти Ajax та сценарії взаємодії з сервером

Зміна даних (завантаження їх з сервера) на сторінці без повторного її перезавантаження здійснюється за допомогою технології `Ajax - Asynchronous Javascript + XML` – способу використання коду мови JavaScript, що розташований на Web-сторінці. Файли даних викликаються з того самого сервера, де розміщена головна сторінка.

Зміни із зазначеного файла реалізуються у поточній web-сторінці за допомогою команд внесення змін до елементів DOM без перезавантаження сторінки. У синхронному режимі користувач чекає на нову сторінку, в асинхронному – працює зі сторінкою, фіксуючи в певні моменти оновлення даних на сторінці. Технологія `Ajax` зменшує потоки даних для трафіку обміну. Схема взаємодії елементів клієнта та сервера (мова, модель, об'єкти, методи та властивості) наведена на рис. 3.1.

Головним об'єктом `Ajax` для обміну клієнта з сервером є `XMLHttpRequest` (IE6 – ActiveXObject). Програміст працює з його методами:

`abort`,

`getAllResponseHeaders`,

`getResponseHeader`,

`open("GET", url, true)` - відкриває з'єднання зі сервером,

`send(data), send(null)` – надсилає запит (з даними або без) на сервер,

`setRequestHeader` - формує заголовок протоколу запиту,

`onreadystatechange` - вказує на зміну стану властивості `readyState`,

`readyState` - властивість, що містить статус об'єкта `XMLHttpRequest`:

0 – не ініціалізовано; 1 – встановлено з'єднання; 2 – відіслано запит; 3 – відповідь надходить; 4 – завершено приймання відповіді,

`responseText` - містить текстовий файл, отриманий від сервера,
`responseXML` - містить XML файл, отриманий від сервера,
`status` - 200 - успішне завершення, 404 – файл не знайдено,
`statusText` - текст повідомлення.

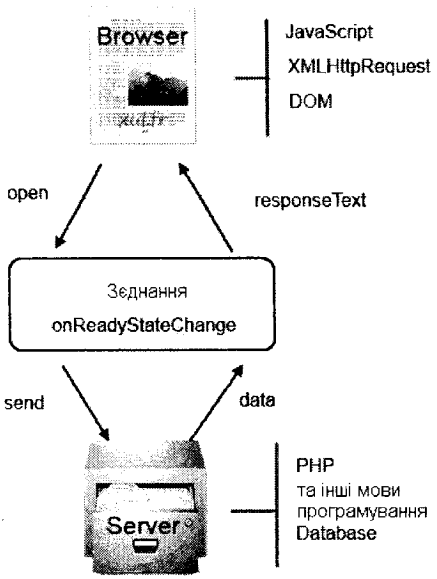


Рис. 3.1. Схема взаємодії клієнта з сервером за технологією `Ajax`

Наведемо приклад використання об'єкта `XMLHttpRequest`, що буде фрагментом обробника певної події

```
var url="http://www.xul.ua/somefile.xml';
var ajax = new XMLHttpRequest();
ajax.onreadystatechange = function_A();
ajax.open("GET", url, true);
ajax.send(null).
```

Обробник подій (`function_A`) прив'язується до властивості (події) `onreadystatechange`. Викликається, як тільки змінюється її значення. Вона містить код для виконання, коли запит завершено.

Схема запуску обробника така:

`readyState` змінюється → `onreadystatechange` → `handler` (це функція `function_A`) виконується.

Як правило, програміста цікавить стан `readyState` (значення 4 – завершення приймання файла). Web-сервер повертає код статусу (`status` = 200 при успішному завершенні).

Наведемо шаблон сценарію використання об'єкта `Ajax XMLHttpRequest` та його властивостей, зокрема, `readyState`:

```
var url="http:/www.xul.ua/somefile.xml";
var ajax = new XMLHttpRequest();
ajax.onreadystatechange = function {
    if (ajax.readyState == 4) {
        if (ajax.status == 200) {
```

Тут опрацювання даних, наприклад, `ajax.responseText`;

```
            var response = xmlHttp.responseText;
    document.getElementById("zipCode").value = response;
        } else {
```

Код опрацювання помилки

```
        }
};
ajax.open("GET", url, true);
ajax.send(null);
```

Більшість коду Ajax використовує анонімні функції для опрацювання подій. Тобто функція описана всередині іншої і не має назви. Ця корисна властивість робить можливим доступ до локальних змінних.

Під час використання об'єкта `XMLHttpRequest` застосовуються такі елементи безпеки:

- не виконується з web-сторінки, розміщеної на жорсткому диску;
- виконується на web-сторінці, розміщеній на web-сервері;
- викликаються файли з того самого сайта, де розташована web-сторінка (наприклад, файл `www.f1.com/a/b/c.html` можна викликати з `www.f1.com`).
- Під час формування запиту з параметрами до виконавчого файла на сервері використовується метод POST.

Наведемо приклад формування запиту з даними:

```
var url="http:/www.xul.ua';
var ajax = new XMLHttpRequest();
var data = "file=" + url + "&content=" + content;
 ajax.open("POST", "ajax_post_text.php", true);
ajax.setRequestHeader("Content-Type",
"application/x-www-form-urlencoded");
ajax.send(data);
```

3.2. Обмін форматованими даними

Поширеною мовою опису даних є XML. Мова XML призначена для опису та зберігання даних ієрархічного типу. Вона використовується для зберігання та обміну форматованими даними між різними програмними системами:

- дані web-серверів;
- дані результатів запитів до серверів баз даних;
- дані web-сервісів;
- новини RSS-блоків.

Мова XHTML є підмножиною XML. Структура документа XML така:

заголовок `<?xml version="1.0" encoding="UTF-8"?>`, потім тег, що може містити ще багато тегів.

Синтаксис тегу:

```
<element attributes>
    Текст чи теги
</element>
```

Тег без внутрішніх тегів/змісту закінчується символами />

Синтаксис атрибутів:

```
name="value"
```

Коментарі:

```
<!-- comment -->
```

Наведемо приклад файла XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
<book category="cooking">
  <title lang="en">Everyday Italian</title>
```

```
  <author>Giada De Laurentiis</author>
  <year>2005</year><price>30.00</price>
</book>
<book category="computers">
  <title lang="en">XQuery Kick Start</title>
  <author>James McGovern</author>
  <year>2003</year><price>49.99</price>
</book>
<book category="children">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year><price>29.99</price>
</book>
<book category="computers">
  <title lang="en">Learning XML</title>
  <author>Erik T. Ray</author>
  <year>2003</year><price>39.95</price>
</book></bookstore>
```

Наведені дані про книги можна відобразити ієрархічною структурою (рис. 3.2).

Повернений XML-файл записується у властивість (об'єкт) `responseXML`. Доступ до елементів даних та їх атрибутів здійснюється алгоритмом обходу дерева з використанням методів `getElementsByTagName()` та `getAttribute()`. Це проілюстровано наступним фрагментом програми JavaScript:

```
var xmlDoc = ajax.responseXML;
var books = xmlDoc.getElementsByTagName("book");
for (var i = 0; i < books.length; i++) {
    var category =
books[i].getAttribute("category");
    if (category == "computers") {
     var title =
books[i].getElementsByTagName("title")[0].firstChild
.nodeValue;
     var author =
books[i].getElementsByTagName("author")[0].firstChil
d.nodeValue;
        var p = document.createElement("p");
        p.innerHTML = title + ", by " + author;
        document.body.appendChild(p);
    }
}
```

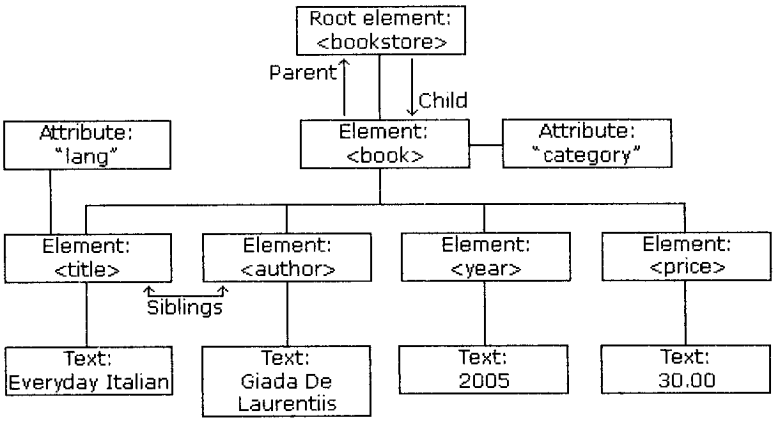


Рис. 3.2. Дерево ієрархічних даних

Фрагмент формує параграф з автором та назвами книг, наприклад,

```
"XQuery Kick Start, by James McGovern".
```

3.3. Використання jQuery для застосування Ajax

Значно простіше можна реалізувати відправлення асинхроних запитів від клієнта і отримання відповіді від сервера за допомогою методів технології `jQuery`, які автоматизують керування процесами в `Ajax`. Основні методи та їх функційні можливості описано в наведеній нижче таблиці.

Метод	Опис
1	2
`$.ajax({name:value, name:value, ... })`	Виконує запит `Ajax`
`$(selector).ajaxComplete(function (event,xhr,options))`	Визначає функцію для запуску, коли запит `Ajax` завершується

Продовження таблиці

1	2
`$(selector).ajaxError(function (event,xhr,options,exc))`	Визначає функцію для запуску, коли запит Ajax завершується з помилкою
`$(selector).ajaxSend(function (event,xhr,options))`	Визначає функцію для запуску, перш ніж запит Ajax надсилається
`$.ajaxSetup({name:value, name:value, ... })`	Встановлює значення за замовчуванням для наступних запитів Ajax
`$(selector).ajaxStart(function())`	Визначає функції при першому запиті Ajax
`$(selector).ajaxStop(function())`	Визначає функцію , коли всі Ajax запити завершились
`$(selector).ajaxSuccess( function(event,xhr,options))`	Визначає функцію, якщо запит Ajax успішно завершений
`$(selector).get(url,data, success(response,status,xhr), dataType)`	Завантаження даних з сервера за допомогою Ajax, HTTP GET-запиту
`$(selector).getJSON(url,data, success(data,status,xhr))`	Завантаження JSON-кодованих даних від сервера, використовуючи HTTP GET-запит
`$(selector).getScript(url, success(response,status))`	Завантаження (і виконання) JavaScript з сервера за допомогою Ajax, HTTP GET-запиту

Закінчення таблиці

1	2
`$(selector).load(url,data,callback)`	Завантаження HTML даних з сервера і розміщення їх у вибраний елемент
`$.param(object,trad)`	Серіалізоване подання масиву або об'єкта (може бути використано як URL-рядок запиту для запитів `Ajax`)
`$(selector).post(url,data, success(response,status,xhr), dataType)`	Завантаження даних з сервера, використовуючи HTTP POST `Ajax` запит
`$(selector).serialize()`	Кодування набору елементів форми у вигляді рядка для пересилання
`$(selector).serializeArray()`	Кодування набору елементів форми у вигляді масиву пар: назва і значення

У таблиці використано такі позначення:

- `url` – адреса сервера, тип (формат) даних, дані та назва функції, що прийме на опрацювання дані,
- `success(response,status,xhr)` – функція для виконання успішного завершення запиту,
- `response` – дані відповіді,
- `status` – статус запиту (`"success"`, `"notmodified"`, `"error"`, `"timeout"`, або `"parsererror"`),
- `xhr` – містить об'єкт XMLHttpRequest,

- dataType – це тип даних (xml, html, script, json, text, _default) з сервера (необов'язковий параметр).

Наведемо приклад виклику основного методу $.ajax(), з такими параметрами:

```
$.ajax({
  url: url,
  data: data,
  success: callback,
 dataType: 'json'
});
```

Аналогічні функції виконують методи

- $(selector).get(),
- $(selector).getJSON(),
- $(selector).post().

Наступний приклад надсилає запит, отримує відповідь з файлом та заносить окремі значення ключових слів у відповідні елементи сторінки. Реалізовано це ієрархією чотирьох функцій: **функція дії на завантажену сторінку (функція дії на подію (функція дії на опрацювання XML-файла (функція дії на знаходження та занесення значень відповідних ключів)))).**

```
$(document).ready(function(){              // після завантаження
 $('#example').click(function(){   // ф-я для події id =
example
  $.post('ajax/example.xml', {}, function(xml){  / дані
з файла
    $('#example').html('');
    $(xml).find('note').each(function(){   // заповнення
    $('#example').append('To: ' +
$(this).find('to').text() + '<br/>')
                .append('From: ' +
    $(this).find('from').text() + '<br/>')
                .append('<b>'  +
    $(this).find('heading').text() + '</b><br/>')
                .append(
    $(this).find('body').text() + '<br/>');
          });
```

```
        }, 'xml');              // тип даних
    })
});
```

Тут використовуються раніше наведені методи доступу до XML-тексту та знаходження значень та доповнення елементів сторінки:

- `$('#example').html('')` - Зміна змісту елемента (inner) HTML
- `$(xml).find('note').each(function(){ }` – пошук ідентифікатора і для кожного виконати функцію
- `$(this).find('to').text()` – пошук ключа і занесення значення у `text`
- `$('#example').append()` – Додавання змісту до елемента (inner) HTML

Файл `example.xml` має вигляд:

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
    <to>Peter</to>
    <from>Nick</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
</note>
```

Подібно наступний приклад надсилає запит, отримує відповідь з JSON текстом у файлі та заносить окремі значення ключових слів у відповідні елементи сторінки. Тут є тільки три функції, оскільки не потрібно шукати ключові слова у JSON-файлі (на відміну від XML-файла), а заносити дані згідно з назвою змінних, які відповідають назвам ключів у вхідному файлі.

```
$(document).ready(function(){      // після завантаження
 $('#example-4').click(function(){     // функція для події
id = example-4
 $.getJSON('ajax/example.json', {}, function(json){
// завантаження JSON даних з файла example.json
   $('#example-4').html(''); // заповнюємо DOM елемент
даними JSON
```

```
    $('#example-4').append('To: ' + json.note.to +
'<br/>')
                                .append('From: ' +
json.note.from + '<br/>')
                                .append('<b>' +
json.note.heading + '</b><br/>')
                                .append(
json.note.body + '<br/>');
            });
        })
});
```

Сам файл `example.json` має вигляд подібний до асоціативного масиву в мовах `JavaScript` чи `PHP`:

```
{
   note:{
       to:'Peter',
       from:'Nick',
       heading:'Reminder',
       body:'Don\'t forget to bring me my book!'
   }
}
```

Контрольні запитання

1. Які об'єкти є в AJAX?
2. Чому AJAX є асинхронним?
3. Методи основного об'єкта AJAX. Навести приклади.
4. Властивості основного об'єкта AJAX. Навести приклади.
5. Що потрібно для реалізації AJAX-технології?
6. Чому мова XML є основою технології AJAX?
7. Як прочитати дані з XML файла?
8. Які події опрацьовують програми в AJAX.
9. Які основні методи для відсилання та приймання даних?
10. Порівняти AJAX та інші подібні технології.
11. Засобами AJAX створити програму формування запиту.

12. Розробити програму засобами AJAX опрацювання відповіді.

13. Розробити програму модифікації вузлів дерева DOM засобами AJAX.

14. Навести приклади об'єктів мови AJAX.

15. Навести приклади функцій мови JQURY.

16. Які функції JQUERY надсилають запит на сервер?

17. Які функції JQUERY читають дані файла з сервера?

18. Навести приклад даних у форматі XML.

19. Навести приклади даних у форматі JSON.

20. Які функції JQUERY отримують дані з сервера?

Глава 4
ПІДТРИМКА СТИЛІВ ЗА ДОПОМОГОЮ W3.CSS

4.1. Під'єднання W3.CSS та основні елементи

W3.CSS – це бібліотека JavaScript функцій підтримки стилів на Web-сторінці: кольорів для різних типів тегів, анімації, шрифтів текстів, таблиць, тегів HTML, документів. Технологія W3.CSS спрощує розроблення дизайну сторінок.

Для використання бібліотеки необхідно на сторінці вказати адресу її розміщення, наприклад, за допомогою тегів:

```
<!DOCTYPE html>
<html>
<meta name="viewport" content="width=device-width,
initial-scale=1">
<link
rel="stylesheet"href="http://www.w3schools.com/lib/w
3.css">
```

Або завнтажити з ресурсу *w3css_downloads*, а назву бібліотеки (директорії) на власному комп'ютері подати в атрибуті href:

```
<link rel="stylesheet" href="/lib/w3.css">
```

Синтаксис W3.CSS подібний до синтаксису стилів CSS складається з атрибута *class* HTML-елемента та його значення, що вказує на тип та параметри функції. Вона виконується для реалізації представлення тегу. Наприклад, у наступному тегу *div* атрибут *class="w3=container w3-teal"* вказує на тип функції *container* і значення його кольору *teal*

```
<div class="w3-container w3-teal">
 <h1>My Header</h1>
</div>
```

Усі селектори умовно погруповані в бібліотеках класів: (W3.CSS) Containers, Panels, Borders, Cards, Fonts, Text, Round, Padding, Margins, Display, Buttons, Notes, Quotes, Alerts, Tables, Lists, Images,

Inputs, Badges, Tags, Icons, Responsive , Layout, Animations, Effects, Bars, Dropdowns, Accordions, Navigation, Sidebar, Tabs, Pagination, Progress Bars, Slideshow, Modal, Tooltips, Grid, Code, Filters, Trends, Case, Material, Validation, Versions, Pro, Mobile.

Особливо значне різноманіття мають бібліотеки для опрацювання кольорів різних тегів: Color Classes, Color Schemes, Color Flat UI, Color Libraries, Color Themes, Color Generator.

Послідовно розглянемо дію функцій на представлення тегів на сторінці.

Клас *w3-container* додає зліва і справа від змісту кожного тегу HTML прогалини розміром 16px. Він найкраще підходить до використання «контейнерних» тегів типу , <article>, <section>, <header>, <footer>, <form> та інших. З цим класифікатором елементи матимуть: спільні поля, спільні прогалини, спільне вирівнювання, спільні шрифти, спільні кольори.

Типи Panels, Notes, Quotes, Alerts реалізуються класом w3-panel, який додає прогалини розміром 16px на поля зверху і знизу до змісту зазначеного тегу HTML.

```
<div class="w3-panel w3-red">
 <p>I am a panel.</p>
</div>
```

Cards. Клас w3-card застосовують до зображень і записів, щоб подати їх у рамках.

У таблиці наведено типи класів w3-card.

Клас	Пояснення
w3-card	Те , що і для w3-card-2
w3-card-2	Контейнер для HTML змісту(рамка шириною 2px)
w3-card-4	Контейнер для HTML змісту(рамка шириною 4px)

Tables Інструкція w3-table призначена для полегшення роботи з HTML тегом table.

Клас	Пояснення
w3-table	Таблиця
w3-striped	таблиця смужками
w3-border	Таблиця з границями
w3-bordered	Позначені лінії
w3-centered	Центрований зміст
w3-hoverable	Зміна кольору під курсором
w3-table-all	Всі властивості

Lists. Інструкція w3-ul призначена для створення списків різних типів

```
<ul class="w3-ul">
 <li>Jill</li>
 <li>Eve</li>
 <li>Adam</li>
</ul>
```

Buttons. Іструкції w3-button, w3-btn призначені для керування типом кнопок.

Клас	Пояснення
w3-btn	Прямокутна кнопка з ефектом зміни кольору під курсором. За замовченням чорного кольору
w3-button	Прямокутна кнопка з ефектом зміни кольору під курсором. За замовченням сірого кольору
w3-bar	Горизонтальна лінійка кнопок (для меню)
w3-block	Кнопка на 100 % ширини.
w3-circle	Кругла кнопка
w3-ripple	Для творення пульсувального ефекту.

```
<div class="w3-show-inline-block">
<div class="w3-bar">
 <button class="w3-btn">Button</button>
```

```
 <button class="w3-btn w3-teal">Button</button>
 <button class="w3-btn w3-disabled">Button</button>
</div>
</div>
```

Tags, Labels, Badges, Signs. Інструкції w3-tag ,w3-label і w3-badge призначені для формування надписів, спеціальних знаків тощо

```
<p>Status: <span class="w3-tag">Done</span></p>
<div class="w3-panel w3-red">
 <p>I am a panel.</p>
</div>
<p>Updates <span class="w3-badge">9</span></p>
```

4.2. Адаптивність у W3.CSS

Адаптивність. Інструкція responsive grid забезпечує адаптацію подання елементів сторінки для всіх типів пристроїв: PC, лаптоп, таблетки та мобільні телефони. Підтримується сітка на 12 колонок для маленьких, середніх та великих класів.

Клас	Пояснення
w3-half	займає 1/2 вікна (для середніх та великих екранів)
w3-third	займає 1/3 вікна (для середніх та великих екранів)
w3-twothird	займає 2/3 вікна (для середніх та великих екранів
w3-quarter	займає 1/4 вікна (для середніх та великих екранів)
w3-threequarter	займає 3/4 вікна (для середніх та великих екранів)
w3-rest	займає решту колонок ширини вікна
w3-col	одна колонка серед 12 всієї сітки
w3-mobile	мобільна адаптивність до клітинки (колонки). Показує елементи блоком на мобільному

responsive класи мусять бути розміщені всередині класів w3-row (або w3-row-padding) щоб бути повністю адаптивними.

Клас	Пояснення
w3-row	Контейнер для адаптивних класів без прогалин
w3-row-padding	Контейнер для адаптивних класів з прогалинами з 8 px ліворуч і праворуч
w3-content	Контейнер для центрованого змісту фіксованого розміру
w3-hide-small	Ховати зміст на маленькому екрані (менше, ніж 601px)
w3-hide-medium	ховати зміст на середньому екрані
w3-hide-large	ховати зміст на великому екрані (більше, ніж 992px)
l1 - l12	адаптивні розміри для великих екранів
m1 - m12	адаптивні розміри для середніх екранів
s1 - s12	адаптивні розміри для малих екранів

Display. Інструкція w3-display-classes дає змогу показувати HTML-елементи на певних позиціях. Класи цього типу наведено в таблиці.

Клас	Пояснення
w3-display-container	контейнер для класів w3-display
w3-display-topleft	показує зміст верхнього лівого кута display-container
w3-display-topright	показує зміст верхнього правого кута w3-display-container
w3-display-bottomleft	показує зміст нижнього лівого кута w3-display-container
w3-display-bottomright	показує зміст верхнього правого кута w3-display-container

W3.CSS Modals. Інструкція w3-modal class дає змогу утворити умовні модулі мовою HTML, які як вікна можуть бути реалізовані або заховані. Зміст модуля формується будь-якими HTML-елементами (divs, headings, paragraphs, images, etc.).

Є два типи класів:

Клас	Пояснення
w3-modal	контейнер модуля (вікна)
w3-modal-content	модуль змісту

Images. Класи цього типу дають можливість оформляти границі зображення як округлені (Rounded), кола (Circle), позначені (Bordered) і з текстом на зображення (Text).

```
<img src="img_fjords.jpg" class="w3-round"
alt="Norway">
```

Progress Bars. HTML-елемент використовується для позначення прямокутника прогресу виконання. А властивості width і height вказують на розміри прямокутника.

```
<div class="w3-border">
 <div class="w3-grey"
style="height:24px;width:20%"></div>
</div>
```

Dropdowns. Клас w3-dropdown забезпечує реалізацію меню, що розгортається. Можливі такі модифікації:

Клас	Пояснення
w3-dropdown-hover	Елемент, що змінює колір під курсором
w3-dropdown-content	Зміст для висвітлення
w3-dropdown-click	Елемент для натискання

```
<div class="w3-dropdown-hover">
 <button class="w3-button">Hover Over Me!</button>
 <div class="w3-dropdown-content w3-bar-block w3-
border">
```

```
  <a href="#" class="w3-bar-item w3-button">Link
1</a>
  <a href="#" class="w3-bar-item w3-button">Link
2</a>
  <a href="#" class="w3-bar-item w3-button">Link
3</a>
 </div>
</div>
```

Accordions. Класи для відкриття чи захования змісту HTML. Клас `w3-hide` ховає зміст. Кнопку використовують для керування відкриттям чи хованням змісту:

```
<button onclick="myFunction('Demo1')" class="w3-
button w3-block w3-left-align">
Open Section 1</button>
<div id="Demo1" class="w3-container w3-hide">
 <p>Some text..</p>
</div>
<script>
function myFunction(id) {
  var x = document.getElementById(id);
  if (x.className.indexOf("w3-show") == -1) {
    x.className += " w3-show";
  } else {
    x.className = x.className.replace(" w3-show",
"");
  }
}
</script>
```

Tabs. Використовують для навігації серед змісту сторінки. Для керування навігацією використовують кнопки. У прикладі нижче елементи змісту містять клас "city", а стиль представлення змінюється від display:none до display:block:

```
<div id="London" class="city">
 <h2>London</h2>
 <p>London is the capital of England.</p>
</div>
<div id="Paris" class="city" style="display:none">
 <h2>Paris</h2>
 <p>Paris is the capital of France.</p>
```

```
</div>
<div id="Tokyo" class="city" style="display:none">
 <h2>Tokyo</h2>
 <p>Tokyo is the capital of Japan.</p>
</div>
```

Табулювання записують кнопками меню:

```
<div class="w3-bar w3-black">
 <button class="w3-bar-item w3-
button"onclick="openCity('London')">London</button>
 <button class="w3-bar-item w3-
button"onclick="openCity('Paris')">Paris</button>
 <button class="w3-bar-item w3-
button"onclick="openCity('Tokyo')">Tokyo</button>
</div>
```

А керування кодом JavaScript:

```
function openCity(cityName) {
  var i;
  var x = document.getElementsByClassName("city");
  for (i = 0; i < x.length; i++) {
    x[i].style.display = "none";
  }
document.getElementById(cityName).style.display =
"block";
}
```

Тут `openCity()` виконується в разі вибору кнопки в меню. Функція ховає елементи класу `class= "city" (display="none")`, або висвітлює елемент з тією самою назвою `(display="block")`;

Висновки та контрольні запитання

W3.CSS – це сучасні засоби підтримки адаптивності CSS до екранів різних розмірів для дизайну інтерфейсу. Вона компактна і швидкодіюча. Проста для освоєння, не залежить від інших бібліотек. Функції прискорюють і спрощують розроблення веб-застосувань. Бібліотеку W3.CSS підтримують основні браузери. Chrome, Firefox,

IE, Safari та інші для всіх пристроїв: ПК, ноутбук, планшет і мобільний телефон.

1. Які завантажити бібліотеки W3.CSS? Навести приклади.
2. Де розмістити виклик бібліотек?
3. Навести три приклади класів для керування стилями.
4. Які інструменти для забезпечення адаптивності до розмірів екранів?
5. Які відмінності від JQuery?
6. Які відмінності від стандартного CSS?
7. Які особливості зображень у W3.CSS?
8. Порівняти W3.CSS та інші подібні інструменти.
9. Створити адаптивне меню
10. Створити адаптивну таблицю.

Глава 5
БІБЛІОТЕКА BOOTSTRAP

5.1. Під'єднання Bootstrap та основні елементи

Bootstrap – це фреймворк для розроблення клієнтських застосувань (front end). Bootstrap містить шаблони на базі HTML та CSS з текстом, формами, кнопками, таблицями, навігацією, модулями, каруселями зображень та багато інших сучасних елементів веб-сторінок, а також вбудовані засоби JavaScript (плагіни). Bootstrap дає змогу створювати адаптивні сторінки у телефонах, планшетах та ноутбуках. Bootstrap 3 підтримує реалізацію для мобільних телефонів. Bootstrap підтримується у сучасних браузерах (Chrome, Firefox, Internet Explorer, Safari, Opera). Версія 4.0 містить препроцесор CSS (SASS) та підтримку `flex-box`.

Для створення сторінки необхідно завантажити бібліотеки підтримки `CSS Bootstrap, JavaScript та jQuery`:

```
<!-- Latest compiled and minified CSS -->
<link
rel="stylesheet"href="https://maxcdn.bootstrapcdn.co
m/bootstrap/3.3.7/css/bootstrap.min.css">

<!-- jQuery library -->
<scriptsrc="https://ajax.googleapis.com/ajax/libs/jq
uery/1.12.4/jquery.min.js"></script>

<!-- Latest compiled JavaScript -->
<scriptsrc="https://maxcdn.bootstrapcdn.com/bootstra
p/3.3.7/js/bootstrap.min.js"></script>
```

Розглянемо етапи розроблення веб-сторінки за допомогою Bootstrap

1. На першому етапі формується заголовок у стилі HTML5 `doctype`, оскільки Bootstrap потребує саме його (включно з атрибутами (`lang i charset`):

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="utf-8">
 </head>
</html>
```

2. **Bootstrap** 3 is mobile-first. Засоби Bootstrap 3 дають можливість розробляти сторінки, адаптивні до мобільних пристроїв. Налаштування сторінки під розмір екрана пристрою забезпечується атрибутами в тегу `<meta>` всередині тегу `<head>`:

```
<meta name="viewport" content="width=device-width,
initial-scale=1">,
```

де `width=device-width` для керування динамічним налаштуванням. `initial-scale=1` вказує на початковий рівень зміни розміру сторінки за першого завантаження.

3. **Containers**. Bootstrap також використовує два елементи (контейнери) для зберігання змісту сторінки: `.container` – клас фіксованої довжини, `.container-fluid` – клас повної ширини з розширенням до повного екрана.

Ці елементи є кінцевими вузлами дерева. Приклад тексту сторінки з використанням зазначених елементів наведено нижче:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <title>Bootstrap Example</title>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width,
initial-scale=1">
 <link
rel="stylesheet"href="https://maxcdn.bootstrapcdn.co
m/bootstrap/3.3.7/css/bootstrap.min.css">
<scriptsrc="https://ajax.googleapis.com/ajax/libs/jq
uery/1.12.4/jquery.min.js"></script>
<scriptsrc="https://maxcdn.bootstrapcdn.com/bootstra
p/3.3.7/js/bootstrap.min.js"></script>
</head>
<body>
<div class="container"> або <div class="container-
fluid">
```

```
  <h1>My First Bootstrap Page</h1>
 <p>This is some text.</p>
</div>
</body>
</html>
```

Атрибути Bootstrap погруповані в групи подібно W3.CSS. Назва групи вказує на призначення: `Grid Basic, Typography, Tables, Images, Jumbotron, Wells, Alerts, Buttons, Button Groups, Glyphicons, Badges/Labels, Progress Bars, Pagination, Pager, List Groups, Panels, Dropdowns, Collapse, Tabs/Pills, Navbar. Forms, Inputs, Input Sizing, Media Objects, Carousel, Modal, Tooltip, Popover, Scrollspy, Affix.`

Коротко зупинимось на головних атрибутах різних груп класів. `Typography` – стилі шрифтів керуються класами Bootstrap і мають незначні відмінності від стилів браузерів за замовчуванням.

Таблиці – стилі таблиць підтримуються класами `.table`, `.table-striped` – почергові рядкі з фоном, `.table-bordered` – з окантуванням таблиці, `.table-hover` – сірий фон на рядки таблиці, `.table-condensed` – компактна форма таблиці.

Контекстні класи можна використати для спеціального позначення рядків (`<tr>`) чи клітинок таблиці (`<td>`):

`.active` – колір під курсором у рядку чи клітинці;
`.success` – вказує на успіх або позитивну дію;
`.info` – вказує на інформацію щодо змін чи дії;
`.warning` – попередження для уваги;
`.danger` – вказує на небезпечні дії.

Клас `.table-responsive` забезпечує адаптивну таблицю:

```
<div class="table-responsive">
 <table class="table">
  ...
 </table>
</div>
```

Класи зображень. .img-rounded заокруглені кути .img-circle – зображення в колі <img src="cinqueterre.jpg" class="img-circle" alt="Cinque Terre"width="304" height="236">

Клас .img-thumbnail забезпечує мінітюаризацію зображень:

```
<img src="cinqueterre.jpg" class="img-thumbnail" alt="Cinque Terre"width="304" height="236">
```

Створити адаптивні зображення дає змогу клас .img-responsive , що додається до тегу <img>.

До класу .img-responsive можна застосувати властивості display: block; max-width: 100%; height: auto;.

Створити адаптивні вбудовані об'єкти (відео) дає можливість клас .embed-responsive-item:

```
<div class="embed-responsive embed-responsive-16by9">
 <iframe class="embed-responsive-item" src="..."></iframe>
</div>
```

Клас .well додає округлену границю, сірий фон та доповнення прогалинами:

```
<div class="well">Basic Well</div>
```

Розміри керуються відповідними класам від слів Small, Large: .well-sm .well-lg :

```
<div class="well well-sm">Small Well</div>
<div class="well well-lg">Large Well</div>
```

Для кнопок Bootstrap передбачає велике різноманіття стилів: Basic. Default. Primary. Success. Info. Warning Danger. LinkTo, для яких використовують такі класи: .btn .btn-default .btn-primary, .btn-success, .btn-info, .btn-warning, .btn-danger, .btn-link

```
<button type="button" class="btn">Basic</button>
<button type="button" class="btn btn-default">Default</button>
```

Кнопки можна використовувати в елементах <a>, <button> та <input>.

```
<input type="button" class="btn btn-info" value="Input Button">
```

```
<input type="submit" class="btn btn-info"
value="Submit Button">
```

Розміри кнопок керуються класами: .btn-lg, .btn-md, .btn-sm, .btn-xs

```
<button type="button" class="btn btn-primary btn-
xs">XSmall</button>
```

Стан кнопки задається класами .active та .disabled:

```
<button type="button" class="btn btn-primary
disabled">Disabled Primary</button>
```

Стрічка прогресу (Progress Bar). Bootstrap підтримкує декілька типів класом .progress в елементі :

```
<div class="progress">
 <div class="progress-bar" role="progressbar" aria-
valuenow="70"
 aria-valuemin="0" aria-valuemax="100"
style="width:70%">
  <span class="sr-only">70% Complete</span>
 </div>
</div>
```

Позначення посилань сторінок відбувається класом .pagination у елементі <ul> :

```
<ul class="pagination">
 <li><a href="#">1</a></li>
 <li><a href="#">2</a></li>
 <li><a href="#">3</a></li>
</ul>
```

Позиції можуть бути в станах .active або .disabled

```
<li  class="disabled"><a href="#">4</a></li>
```

А розміри блоків керуються класми .pagination-lg та .pagination-sm .

Клас .breadcrumb забезпечує створення менб-сторінок:

```
<ul class="breadcrumb">
 <li><a href="#">Home</a></li>
 <li><a href="#">Private</a></li>
 <li><a href="#">Pictures</a></li>
 <li class="active">Vacation</li>
</ul>
```

Вказівник на наступну і попередню сторінки .pager на <ul> елемент з або без класів .previous, .next для вирівнювання кнопок:

```
<ul class="pager">
 <li class="previous"><a href="#">Previous</a></li>
 <li class="next"><a href="#">Next</a></li>
</ul>
```

Для панелей використовують класи .panel, .panel-body , .panel-heading , .panel-footer:

```
<div class="panel panel-default">
 <div class="panel-body">Panel Content</div>
 <div class="panel-footer">Panel Footer</div>
</div>
```

Меню, що розгортається дає змогу вибрати позицію з їхнього списку, для цього використовують класи dropdown dropdown-menu dropdown-toggle .divider , .dropdown-header , .dropdown-menu-right, "dropup", та атрибути role і aria-*:

```
<div class="dropdown">
 <button class="btn btn-default dropdown-toggle"
type="button"id="menu1" data-
toggle="dropdown">Tutorials
 <span class="caret"></span></button>
 <ul class="dropdown-menu" role="menu" aria-
labelledby="menu1">
  <li role="presentation"><a role="menuitem"
href="#">HTML</a></li>
  <li role="presentation"><a role="menuitem"
href="#">CSS</a></li>
  <li role="presentation"><a
role="menuitem"href="#">JavaScript</a></li>
  <li role="presentation" class="divider"></li>
  <li role="presentation"><a role="menuitem"
href="#">About Us</a></li>
 </ul>
</div>
```

Клас .list-inline використовується до тегу <ul>: <ul class="nav nav-tabs">:

```
<ul class="nav nav-tabs">
```

```
 <li class="active"><a href="#">Home</a></li>
 <li><a href="#">Menu 1</a></li>
 <li><a href="#">Menu 2</a></li>
 <li><a href="#">Menu 3</a></li>
</ul>
```

Планування розташування елементів на формі, можливими є три розташування: вертикальне (за замовчування), горизонтальне, та в рядок. Усі елементи управління форми розміщаються всередині тегу з класом `<div class="form-group">`, а клас `.form-control` додається до всіх елементів управління типу `<input>`, `<textarea>`, та `<select>`, наприклад вертикального розташування:

```
<form>
 <div class="form-group">
  <label for="email">Email address:</label>
  <input type="email" class="form-control"
id="email">
 </div>
 <div class="form-group">
  <label for="pwd">Password:</label>
  <input type="password" class="form-control"
id="pwd">
 </div>
 <div class="checkbox">
  <label><input type="checkbox"> Remember me</label>
 </div>
 <button type="submit" class="btn btn-
default">Submit</button>
</form>
```

Замінивши класи на `.form-inline`, отримаємо розташування в рядок, з `.form-horizontal` та `.control-label` у всіх елементах `<label>` отримаємо горизонтальне розташування.

Bootstrap підтримує елементи керування: `input`, `textarea`, `checkbox`, `radio`, `select` вхідних даних HTML5: тексти, паролі, дати, час тощо:

```
<div class="form-group">
 <label for="usr">Name:</label>
 <input type="text" class="form-control" id="usr">
```

```
</div>
<div class="form-group">
 <label for="pwd">Password:</label>
 <input type="password" class="form-control"
id="pwd">
</div>
```

5.2. Адаптивність у Bootstrap

Bootstrap підтримує покриття екрана пристрою умовною сіткою за допомогою класів: `xs` (для телефонів), `sm` (для планшетів), `md` (дя ноутбуків), `lg` (для великих екранів).

Класи можуть комбінуватись. Усі класи масштабовані.

Під час використання сітки Bootstrap треба дотримуватись певних правил:

- рядки розміщують у межах класів `.container` або `.container-fluid`;
- рядки використовують для формування горизонтальних груп колонок;
- зміст розміщують у колонках і тільки колонки породжують від рядків;
- передвизначені класи типу `.row` і `.col-sm-4` призначені для планування розміщення;
- колонки створюють проміжки між колонками змісту зміною кількості прогалин. Ці прогалини появляються в рядках, в першій і останній колонці через від'ємне значення поля в класі `.rows`;
- сітку створюють у разі задавання числа колонок (можливо 12). Наприклад, три рівні колонки використовують за класом `.col-sm-4`

Приклад створення сітки наведено нижче:

```
<div class="container">
 <div class="row">
  <div class="col-*-*"></div>
 </div>
```

```
<div class="row">
 <div class="col-*-*"></div>
 <div class="col-*-*"></div>
 <div class="col-*-*"></div>
</div>
<div class="row">
 ...
</div>
</div>
```

У таблиці наведено властивості сіток Bootstrap для різних пристроїв:

	Дуже маленькі (<768px)	Малі планшети (>=768px)	Середні екрани (>=992px)	Великі екрани (>=1200px)
Grid behaviour	Горизонтальні завжди	Згортається, щоб почати, горизонтально вище від точки зупинки	Згортається, щоб почати, горизонтально вище від точки зупинки	Згортається, щоб почати, горизонтально вище від точки зупинки
Ширина контейнера	Авто	750px	970px	1170px
Клас	.col-xs-	.col-sm-	.col-md-	.col-lg-
Кількість колонок	12	12	12	12
Ширина колонки	Auto	~62px	~81px	~97px
Ширина прогалини	30px (15px з кожної стоони колонки)	30px (15px з кожної стоони колонки)	30px (15px з кожної стоони колонки)	30px (15px з кожної стоони колонки)
Гніздування	Так	Так	Так	Так
Зсува	Так	Так	Так	Так
Впорядкування колонок	Так	Так	Так	Так

Висновки та конрольні запитанння

Bootsrap – це сучасні засоби підтримки адаптивності CSS до екранів різних розмірів для дизайну інтерфейсу. Вона компактна і швидкодіюча, проста для освоєння, не залежить від інших бібліотек. Вони прискорюють і спрощують розроблення веб-застосувань. Бібліотеку Bootstrap підтримують основні браузери. Chrome, Firefox, IE, Safari та інші для всіх пристроїв: ПК, ноутбук, планшет і мобільний телефон.

1. Як завантажити бібліотеки Bootstrap? Навести приклади.
2. Де розмістити виклик бібліотек Bootstrap?
3. Навести три приклади класів для керування стилями Bootstrap.
4. Які інструменти для забезпечення адаптивності до розмірів екранів Bootstrap?
5. Які відмінності від JQuery та W3.CSS?
6. Які відмінності від стандартного CSS?
7. Які особливості зображень у Bootstrap?
8. Порівняти Bootstrap та інші подібні інструменти.
9. Створити адаптивне меню.
10. Створити адаптивну таблицю.

ГЛАВА 6
МОДЕЛЬ MVC В ANGULARJS

6.1. Дані та вирази

AngularJS версії 1.0 представлено у 2012 році. На відміну від виразів JavaScript AngularJS вирази записують безпосередньо в HTML-текст. AngularJS-вирази не підтримують розгалужень, циклів, виняткових ситуацій. Додатково AngularJS-вирази підтримують фільтри, дають змогу приєднати дані до HTML.

AngularJS вирази записуються у подвійних дужках : {{ *expression* }} або в межах директиви ng-bind="*expression*". AngularJS розкриє вираз і поверне результат на місці виразу. Вираз містить літерали, оператори і змінні : {{ 5 + 5 }} або {{ firstName + " " + lastName }}.

Для роботи з AngularJS необхідно завантажити сам фреймворк:

```
<!DOCTYPE html>
<html>
<scriptsrc="https://ajax.googleapis.com/ajax/libs/an
gularjs/1.6.4/angular.min.js"></script>
<body>
<div ng-app="">
 <p>My expression: {{ 7 + 7 }}</p>
</div>
</body>
</html>
```

Без директиви ng-app HTML видрукує зміст параграфа без обчислення виразу.

Числа вAngularJS такі самі, як у JavaScript:

```
<div ng-app="" ng-init="quantity=1;cost=5">
<p>Total in dollar: {{ quantity * cost }}</p>
</div>
```

Рядки в AngularJS такі самі, як у JavaScript:

```
<div ng-app="" ng-
init="firstName='John';lastName='Doe'">
<p>The name is {{ firstName + " " + lastName }}</p>
</div>
```

З використання директиви ng-bind

```
<div ng-app="" ng-
init="firstName='John';lastName='Doe'">
<p>The name is <span ng-bind="firstName + ' ' +
lastName"></span></p>
</div>
```

Об'єкти в AngularJS такі самі, як у JavaScript:

```
<div ng-app="" ng-
init="person={firstName:'John',lastName:'Doe'}">
<p>The name is {{ person.lastName }}</p>
</div>
```

Або через ng-bind:

```
<div ng-app="" ng-
init="person={firstName:'John',lastName:'Doe'}">
<p>The name is <span ng-
bind="person.lastName"></span></p>
</div>
```

Масиви в AngularJS такі самі, як у JavaScript:

```
<div ng-app="" ng-init="points=[1,15,19,2,40]">
<p>The third result is {{ points[2] }}</p>
</div>
```

Або через ng-bind:

```
<div ng-app="" ng-
init="points=[1,15,19,2,40]">
<p>The third result is <span ng-
bind="points[2]"></span></p>
</div>
```

Наведемо приклад зміни значення атрибуту CSS , наприклад, кольору фону поля введення даних:

```
<div ng-app="" ng-init="myCol='lightblue'">
<input style="background-color:{{myCol}}" ng-
model="myCol" value="{{myCol}}">
</div>
```

6.2. Модулі

Модулі в AngularJS позначають певні програми. Модуль також може містити частини програми. Модуль може містити керуючі елементи програми (контролери), які завжди мають

бути у модулі. Останні створюють за допомогою функції `angular.module`:

```
<div ng-app="myApp">...</div>
<script>
var app = angular.module("myApp", []);
</script>
```

"myApp" вказує на елемент HTML, у якому виконується модуль. У цей модуль тепер можна додавати контролери, директиви, фільтри та інши елементи `AngularJS`.

Розглянемо приклад додавання контролера директивою `ng-controller`:

```
<div ng-app="myApp" ng-controller="myCtrl">
{{ firstName + " " + lastName }}
</div>
<script>
var app = angular.module("myApp", []);
app.controller("myCtrl", function($scope) {
  $scope.firstName = "John";
  $scope.lastName = "Doe";
});
</script>
```

Модулі і контролери у файлі. Як і в JavaScript, модулі і контролери можна зберігати на дисковій пам'яті. У прикладі "myApp.js" є модулем , а "myCtrl.js" містить контролер:

```
<!DOCTYPE html>
<html>
<scriptsrc="https://ajax.googleapis.com/ajax/libs/an
gularjs/1.6.4/angular.min.js"></script>
<body>

<div ng-app="myApp" ng-controller="myCtrl">
{{ firstName + " " + lastName }}
</div>
<script src="myApp.js"></script>
<script src="myCtrl.js"></script>
</body>
</html>
```

`var app = angular.module("myApp", []);` – дужки вказують на можливість створення залежних модулів

```
app.controller("myCtrl", function($scope) {
    $scope.firstName = "John";
    $scope.lastName= "Doe";
});
```

Функції є локальними для модулів

```
<!DOCTYPE html>
<html>
<body>
<scriptsrc="https://ajax.googleapis.com/ajax/libs/an
gularjs/1.6.4/angular.min.js"></script>

<div ng-app="myApp" ng-controller="myCtrl">
{{ firstName + " " + lastName }}
</div>

<script>
var app = angular.module("myApp", []);
app.controller("myCtrl", function($scope) {
  $scope.firstName = "John";
  $scope.lastName = "Doe";
});
</script>

</body>
</html>
```

6.3. Директиви AngularJS

AngularJS дає змогу розширити HTML новими атрибутами – директивами. Вони розширюють функціональність опрацювання даних, можна розробити і власні директиви.

Приклади директив з префіксом `ng-` :

`ng-app` – ініціалізація програми, `ng-init` – ініціалізація даних, `ng-model` зв'язує дані програми зі значеннями, що вводяться елементами керування HTML (`input, select, textarea`), наприклад:

```
<div ng-app="" ng-init="firstName='John'">
<p>Name: <input type="text" ng-
model="firstName"></p>
```

```
<p>You wrote: {{ firstName }}</p>
</div>
```

Директива ng-app вказує для AngularJS, що є власником AngularJS-програми.

{{ firstName }} – вираз зв'язується з даними у ng-model="firstName".

У наступному прикладі дані виразу (представлення зв'язані моделлю ng-model, що ввводить дані:

```
<div ng-app="" ng-init="quantity=1;price=5">
Quantity: <input type="number" ng-model="quantity">
Costs:  <input type="number" ng-model="price">
Total in dollar: {{ quantity * price }}
</div>
```

Директива ng-repeat слугує повторенню елементів HTML з масивів, об'єктів тощо, наприклад масиву об'єктів:

```
<div ng-app="" ng-init="names=[
{name:'Jani',country:'Norway'},
{name:'Hege',country:'Sweden'},
{name:'Kai',country:'Denmark'}]">
<ul>
 <li ng-repeat="x in names">
  {{ x.name + ', ' + x.country }}
 </li>
</ul>
</div>
```

Додавання директив. Розширити функціональні можливості можна розроблення власних директив:

```
<div ng-app="myApp" w3-test-directive></div>
<script>
var app = angular.module("myApp", []);
app.directive("w3TestDirective", function() {
  return {
    template : "directive constructor"
  };
});
</script>
```

Розроблення нових директив. Для створення нових директив використовують функцію .directive . Створюючи директиву, пишемо її назву разом w3TestDirective, але в разі виклику

окремо: w3-test-directive. Виклик нової директиви здійснюють елементом HTML з назвою тега, що відповідає назві нової директиви. Виклик здійснюють одним з способів: назвою елемента, атрибутом, класом, коментарем:

```
Або <w3-test-directive></w3-test-directive>
або <div w3-test-directive></div>
або  <div class="w3-test-directive"></div>
або <!-- directive: w3-test-directive -->
```

```
<body ng-app="myApp">
<w3-test-directive></w3-test-directive>
<script>
var app = angular.module("myApp", []);
app.directive("w3TestDirective", function() {
  return {
    template : "<h1>Made by a directive!</h1>"
  };
});
</script>
</body>
```

Застосування директиви можна обмежити властивістю restrict до певних елементів.

```
var app = angular.module("myApp", []);
app.directive("w3TestDirective", function() {
  return {
    restrict : "A",
    template : "<h1>Made by a directive!</h1>"
  };
});
```

Обмеження : E для елемента, A – для атрибута, C – для класу, M – для коментарів, за замовчуванням EA.

Директива ng-model призначена для зв'язку значення поля введення з змінною AngularJS.

```
<div ng-app="myApp" ng-controller="myCtrl">
  Name: <input ng-model="name">
</div>

<script>
var app = angular.module('myApp', []);
```

```
app.controller('myCtrl', function($scope) {
  $scope.name = "John Doe";
});
</script>
```

Директива ng-model забезпечує валідацію даних: (number, e-mail, required):

```
<form ng-app="" name="myForm">
  Email:
  <input type="email" name="myAddress" ng-
model="text">
  <span ng-show="myForm.myAddress.$error.email">Not
a valid e-mail address</span>
</form>
```

Звязування даних. AngularJS програма має певну модель даних.

```
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
  $scope.firstname = "John";
  $scope.lastname = "Doe";
});
```

HTML-представлення. Контейнер HTML, у якому AngularJS програма видає результати називається представленням (view) Є декілька способі представлення даних.

Це директива ng-bind для зв'язування innerHTML з властивістю моделі:

```
<p ng-bind="firstname"></p>
```

Або подвійні дужки:

```
<p>First name: {{firstname}}</p>
```

Або директива ng-model у керуючих елементах HTML для зв'язку моделі та представлення:

```
<input ng-model="firstname">
```

Або

```
<div ng-app="myApp" ng-controller="myCtrl">
  Name: <input ng-model="firstname">
  <h1>{{firstname}}</h1>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
```

```
    $scope.firstname = "John";
    $scope.lastname = "Doe";
});
</script>
```

Об'єкт scope призначений для зв'язування представлення HTML (view) з командами JavaScript (controller). Об'єкт має властивості та методи. Він видимий у представленні та контролері. Об'єкт $scope передається контролеру як аргумент, а властивості контролера видимі у представленні (без вказування префікса `$scope`:

```
<div ng-app="myApp" ng-controller="myCtrl">
<h1>{{carname}}</h1>
</div>
<script>
var app = angular.module('myApp', []);

app.controller('myCtrl', function($scope) {
  $scope.carname = "Volvo";
});
</script>
```

scope – це модель (Model). Можливо мати декілька об'єктів. Директива `ng-repeat` дає змогу доступитись до рухомого об'єкта.

```
<div ng-app="myApp" ng-controller="myCtrl">
<ul>
  <li ng-repeat="x in names">{{x}}</li>
</ul>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
  $scope.names = ["Emil", "Tobias", "Linus"];
});
```

Усі програми мають `$rootScope` – об'єкт, створений для елемента HTML з директивою `ng-app`. Він доступний для всіх програм. Для однакових імен перевага віддається рухомому об'єкту. Нехай змінна "`color`"" існує в контролері та `rootScope`:

```
<body ng-app="myApp">
<p>The rootScope's favorite color:</p>
<h1>{{color}}</h1>
```

```
<div ng-controller="myCtrl">
  <p>The scope of the controller's favorite
color:</p>
  <h1>{{color}}</h1>
</div>
<p>The rootScope's favorite color is still:</p>
<h1>{{color}}</h1>
<script>
var app = angular.module('myApp', []);
app.run(function($rootScope) {
  $rootScope.color = 'blue';
});
app.controller('myCtrl', function($scope) {
  $scope.color = "red";
});
</script>
</body>
```

У прикладі буде надруковано -- `blue red blue`.

6.4. Контролери в AngularJS

Програмами в `AngularJS` управляють контролери. Вони відділені від представлень і працюють з даними моделі (зміни відображають негайно). Контролери є об'єктами `JavaScript`. Директива `ng-controller` визначає контролер.

```
<div ng-app="myApp" ng-controller="myCtrl">
  <h1 ng-click="changeName()">{{firstname}}</h1>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
  $scope.firstname = "John";
  $scope.changeName = function() {
    $scope.firstname = "Nelly";
  }
});
```

```
</script>
```
або
```
<div ng-app="myApp" ng-controller="myCtrl">

First Name: <input type="text" ng-
model="firstName"><br>
Last Name: <input type="text" ng-
model="lastName"><br>
<br>
Full Name: {{firstName + " " + lastName}}
</div>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
  $scope.firstName = "John";
  $scope.lastName = "Doe";
});
</script>
```

У прикладах: Програма – `ng-app="myApp"`, що працює в межах `<div>`. `ng-controller="myCtrl"` директива визначає контролер. `myCtrl` - `JavaScript` функція.
AngularJS викликає контролер з об'єктом $scope – власником змінних та функцій програми. У цьому прикладі контролер створює дві властивості у scope: (`firstName and lastName`).

Контролер може мати методи

```
<div ng-app="myApp" ng-controller="personCtrl">
First Name: <input type="text" ng-
model="firstName"><br>
Last Name: <input type="text" ng-
model="lastName"><br>
<br>
Full Name: {{fullName()}}
</div>
<script>
var app = angular.module('myApp', []);
app.controller('personCtrl', function($scope) {
  $scope.firstName = "John";
  $scope.lastName = "Doe";
  $scope.fullName = function() {
```

```
    return $scope.firstName + " " +
$scope.lastName;
  };
});
</script>
```

Контролери можуть бути у зовнішній пам'яті

```
<div ng-app="myApp" ng-controller="personCtrl">
First Name: <input type="text" ng-
model="firstName"><br>
Last Name: <input type="text" ng-
model="lastName"><br>
<br>
Full Name: {{fullName()}}
</div>
<script src="personController.js"></script>
```

Напишемо файл контролера `namesController.js`:

```
('namesCtrl', function($scope) {
  $scope.names = [
    {name:'Jani',country:'Norway'},
    {name:'Hege',country:'Sweden'},
    {name:'Kai',country:'Denmark'}
  ];
});
```

І використаємо його у програмі:

```
<div ng-app="myApp" ng-controller="namesCtrl">

<ul>
 <li ng-repeat="x in names">
  {{ x.name + ', ' + x.country }}
 </li>
</ul>
</div>
<script src="namesController.js"></script>
```

6.5. Засоби опрацювання даних

AngularJS забезпечує фільтри для перетворення даних: `currency` – формат для валют, `date` – специфічний формат даних, `filter` – вибір значень з масиву, `json` – перетворення до JSON

стрічок, limitTo – обмеження масиву/стрічки у певну кількість елементів символів, lowercase – переведення на нижній регістр, number – число у стрічку, orderBy – впорядкування масиву з виразом, uppercase – переведення у верхній регістр.

Наведемо декілька прикладів використання (без контролера):

uppercase

```
<div ng-app="myApp" ng-controller="personCtrl">
<p>The name is {{ lastName | uppercase }}</p>
</div>
```

orderBy :

```
<div ng-app="myApp" ng-controller="namesCtrl">
<ul>
 <li ng-repeat="x in names | orderBy:'country'">
  {{ x.name + ', ' + x.country }}
 </li>
</ul>
</div>
```

Currency :

```
<div ng-app="myApp" ng-controller="costCtrl">
<h1>Price: {{ price | currency }}</h1>
</div>

<div ng-app="myApp" ng-controller="namesCtrl">
filter :
<ul>
 <li ng-repeat="x in names | filter : 'i'"> // імя
містить і
  {{ x }}
 </li>
</ul>
</div>
```

Збільшення списку введенням імен:

```
<div ng-app="myApp" ng-controller="namesCtrl">
<p><input type="text" ng-model="test"></p>
<ul>
 <li ng-repeat="x in names | filter : test">
  {{ x }}
 </li>
</ul>

</div>
```

Сортування масиву у таблиці (директива ng-click у заголовку таблиці):

```
<div ng-app="myApp" ng-controller="namesCtrl">
<table border="1" width="100%">
 <tr>
  <th ng-click="orderByMe('name')">Name</th>
  <th ng-click="orderByMe('country')">Country</th>
 </tr>
 <tr ng-repeat="x in names | orderBy:myOrderBy">
  <td>{{x.name}}</td>
  <td>{{x.country}}</td>
 </tr>
</table>
</div>
<script>
angular.module('myApp', []).controller('namesCtrl',
function($scope) {
 $scope.names = [
  {name:'Jani',country:'Norway'},
  {name:'Carl',country:'Sweden'},
  {name:'Margareth',country:'England'},
  {name:'Hege',country:'Norway'},
  {name:'Joe',country:'Denmark'},
  {name:'Gustav',country:'Sweden'},
  {name:'Birgit',country:'Denmark'},
  {name:'Mary',country:'England'},
  {name:'Kai',country:'Norway'}
 ];
 $scope.orderByMe = function(x) {
  $scope.myOrderBy = x;
 }
});
</script>
```

6.6. Службові засоби – сервіси

В AngularJS, сервіс – це функція або об'єкт для використання у програмах. .AngularJS має близько 30 вбудованих сервісів. Наприклад `$location` повертає інформацію про розташування рухомої сторінки:

```
var app = angular.module('myApp', []);
```

```
app.controller('customersCtrl', function($scope,
$location) {
  $scope.myUrl = $location.absUrl();
});
```

`$http` сервіс – часто вживаний в AngularJS-програмах. Він робить запит до сервера і дає змогу опрацювати отримані дані з сервера:

```
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $http) {
  $http.get("welcome.htm").then(function (response)
{
    $scope.myWelcome = response.data;
  });
});
```

`$timeout` сервіс в AngularJS є аналогом функції `window.setTimeout`:

```
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $timeout)
{
  $scope.myHeader = "Hello World!";
  $timeout(function () {
    $scope.myHeader = "How are you today?";
  }, 2000);
});
```

Сервіс `$interval` в AngularJS є аналогом функції `window.setInterval`.

```
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope,
$interval) {
  $scope.theTime = new Date().toLocaleTimeString();
  $interval(function () {
    $scope.theTime = new
Date().toLocaleTimeString();
  }, 1000);
});
```

Власний сервіс. Розробити свій сервіс можливо за допомогою модуля:

```
app.service('hexafy', function() {
  this.myFunc = function (x) {
    return x.toString(16);
```

```
  }
});

app.controller('myCtrl', function($scope, hexafy) {
  $scope.hex = hexafy.myFunc(255);
});
```

Власний сервіс у фільтрі. Можна розробити власний сервіс та використати у модулі, контролері, директиві чи інших сервісах. Маємо сервіс

```
<ul>
<li ng-repeat="x in counts">{{x | myFormat}}</li>
</ul
```

Проілюструємо сервіс у фільтрі:

```
app.filter('myFormat',['hexafy', function(hexafy) {
  return function(x) {
    return hexafy.myFunc(x);
  };
}]);
```

Розглянемо приклад сервісу AngularJS $http:

```
<div ng-app="myApp" ng-controller="myCtrl">
<p>Today's welcome message is:</p>
<h1>{{myWelcome}}</h1>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $http) {
  $http.get("welcome.htm")
  .then(function(response) {
    $scope.myWelcome = response.data;
  });
});
</script>
```

Сервіс має методи: .delete(), .get(), .head(), .jsonp(), .patch(), .post(), .put().

```
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $http) {
  $http({
```

```
    method : "GET",
    url : "welcome.htm"
  }).then(function mySuccess(response) {
    $scope.myWelcome = response.data;
  }, function myError(response) {
    $scope.myWelcome = response.statusText;
  });
});
```

Сервіс має властивості : `.config`, `.data`, `.headers` – функція для отримання даних зі заголовку протокола, `.status` – число р щодо HTTP статтусу, `.statusText` – стрічка статусу.

```
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $http) {
  $http.get("welcome.htm")
  .then(function(response) {
    $scope.content = response.data;
    $scope.statuscode = response.status;
    $scope.statustext = response.statusText;
  });
});
```

Для опрацювання помилок додається функція до методу `.then` :

```
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $http) {
  $http.get("wrongfilename.htm")
  .then(function(response) {
    //First function handles success
    $scope.content = response.data;
  }, function(response) {
    //Second function handles error
    $scope.content = "Something went wrong";
  });
});
```

Якщо дані очікуються у JSON форматі, необхідно знати їх наповнення, щоб організувати прочитання, як у прикладі:

```
<div ng-app="myApp" ng-controller="customersCtrl">
<ul>
 <li ng-repeat="x in myData">
  {{ x.Name + ', ' + x.Country }}
 </li>
</ul>
```

```
</div>
<script>
var app = angular.module('myApp', []);
app.controller('customersCtrl', function($scope,
$http) {
$http.get("customers.php").then(function(response) {
    $scope.myData = response.data.records;
  });
});

</script>
```

Подібний приклад ілюструє прочитання даних з таблиці, надісланою рядками.

```
<div ng-app="myApp" ng-controller="customersCtrl">
<table>
 <tr ng-repeat="x in names">
  <td>{{ x.Name }}</td>
  <td>{{ x.Country }}</td>
 </tr>
</table>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('customersCtrl', function($scope,
$http) {
  $http.get("customers.php")
  .then(function (response) {$scope.names =
response.data.records;});
});
</script>
```

Висновки та контрольні запитання

AngularJS – це потужні сучасні засоби підтримки адаптивності CSS до екранів різних розмірів для дизайну інтерфейсу. Вона підтримує динамічні зміни на сторінці, компактна і швидкодіюча. Використана модель MVC, тобто: розділення представлення даних, опрацювання їх та управління ними. Проста

для освоєння, не залежить від інших бібліотек. Вони прискорюють і спрощують розроблення веб-застосувань. Бібліотеку AngularJS підтримують основні браузери. Chrome, Firefox, IE, Safari та інші для всіх пристроїв: ПК, ноутбук, планшет і мобільний телефон.

1. Які завантажити бібліотеки AngularJS? Навести приклади.
2. Де розмістити виклик бібліотек AngularJS?
3. Навести три приклади класів для керування стилями. AngularJS.
4. Що таке модуль ?
5. Що є директива?
6. Що є контролер?
7. Навести типи представлень.
8. Навести приклади моделей.
9. Навести приклади scope.
10. Які інструменти для забезпечення адаптивності до розмірів екранів AngularJS?
11. Які відмінності від JQuery, Bootstrap та W3.CSS?
12. Які відмінності від стандартного CSS?
13. Які особливості зображень в AngularJS?
14. Порівняти Bootstrap та інші подібні інструменти.
15. Створити адаптивне меню.
16. Створити адаптивну таблицю.

Частина II

БЕК-ЕНД ЗАСОБАМИ PHP, ASP.NET, JAVA, JSP

Глава 7
ТЕХНОЛОГІЯ PHP

7.1. PHP як засіб написання сценаріїв

PHP (Препроцесор Гіпертексту) є мовою сценаріїв загального призначення з відкритим вихідним кодом. PHP використовується для ведення web-розробок і може записуватись безпосередньо в HTML-коді.

Синтаксис мови бере початок з C, Java, Perl і є легким для вивчення. Переважним призначенням PHP є надання web-розробникам можливості швидкого створення динамічно генерованих web-сторінок, однак сфера застосування PHP лише цим не обмежується.

В офіційній документації мова PHP подається як HTML – вбудована мова програмування (скриптів) на стороні сервера. Це означає, що:

- оброблення PHP-коду відбувається на сервері ще до того, як web-сторінка буде передана браузеру;
- PHP-код може бути вбудований безпосередньо в HTML-код сторінки. Цим він відрізняється від Perl;
- PHP не є ні компілятором, ні інтерпретатором, а чимось середнім між ними. PHP обробляє сценарії, які подаються на вхід. Він транслює його в спеціальний байт-код (внутрішнє представлення). Після цього PHP виконує байт-код. У такому разі виконавчий файл не створюється. Байт-код значно компактніший, ніж звичайний код програми. Тому він швидше інтерпретується та виконується. Тому PHP більше інтерпретатор, ніж компілятор.

Використання інтерпретатора (а, значить, і PHP) має переваги:

- не потрібно дбати про звільнення виділеної пам'яті, закривати файл після закінчення роботи з ним – це здійснює браузер, оскільки програма виконується під його контролем;
- не потрібно визначати типи змінних та оголошувати змінну до її першого використання;

• відлагоджування програми та виявлення помилок істотно полегшуються – інтерпретатор повідомить про це;

• існує можливість створення програми, яка напише іншу програму, а потім введе в себе код щойно написаної програми і виконає його.

Виконання PHP-програм відбувається за схемою (рис. 7.1).

Рис. 7.1. Виконання програм PHP

Перед надсиланням сервером браузеру файла його переглядає препроцес-інтерпретатор. Файли, що піддаються обробленню препроцесором, повинні мати розширення `.phtml` або `.php` і містити код для препроцесора. Перед надсиланням сторінки PHP-код виконується сервером `Apache` і браузеру видається результат у вигляді HTML-сторінки (вона відрізняється від тієї, що зберігається на сервері). Сторінки з розширенням `.html/.htm` web-сервер відправляє браузеру без будь-якого опрацювання.

Основна відмінність від CGI-скриптів, написаних мовами Perl або C, полягає в тому, що в CGI-програмах треба писати виведений HTML-код, а PHP – дає змогу вносити програму – скрипт у код HTML-сторінки (обмежену тегами <? і ?>).

Переваги PHP такі:

продуктивність – сервер може обслуговувати мільйони звернень на день; *інтеграція з базами даних* – PHP володіє вбудованим зв'язком з багатьма системами баз даних (`MySQL`, `PostgreSQL`, `mSQL`, `Oracle`, `dbm`, `Hyperware`, `Informix`, `InterBase`, і `Sybase` та інших на основі ODBC-драйверів);

простота для вивчення – синтаксис PHP оснований на мовах програмування C та `Perl`;

можливість перенесення – пакет PHP можна використовувати під керуванням різних операційних систем (Linux, FreeBSD, Unix, Windows);

наявність вбудованих бібліотек – можна генерувати Gif-зображення, під'єднатись до багатьох мережевих служб, відправляти повідомлення електронною поштою, працювати з cookie-наборами і генерувати PDF-документи;

вартість – пакет PHP є безкоштовний.

Розглянемо приклади використання PHP-коду в тексті HTML-сторінки та самостійного PHP коду:

- в тексті сторінки код роздруковує повідомлення:

```
<!DOCTYPE html>
<html>
<head>
<title>Hello world</title>
</head>
<body>
<?php
print("Hello, World");
?>
</body>
</html>
```

- самостійний код – підрахунок кількості відвідувачів сторінки:

```
<?
if(file_exists("count.dat"))
{
  $exist_file = fopen("count.dat", "r");
  $new_count = fgets($exist_file, 255);
  $new_count++;
  fclose($exist_file);
  //
  print("$new_count people have visited this page");
  $exist_count = fopen("count.dat", "w");
  fputs($exist_count, $new_count);
  fclose($exist_count);
}
else
{
$new_file = fopen("count.dat", "w");
```

```
fputs($new_file, "1");
fclose($new_file);
}
?>
```

Для ввімкнення коду в текст сторінки використовуємо оператор `require`:

```
<? require  ("requiredfile.php"); ?>
```

Цю саму функцію виконує оператор `include()`:

```
include("filename");
include("header.php");
require(. . . . . );
```

Для внесення тексту в код сторінки використовуються команди `print()` або `echo()`:

```
print("text");
print("Hello, World!");
print("Escape \"chars\" are the SAME as in
Java!\n");
```

7.2. Основи програмування PHP

PHP-скрипти утворюються групами операторів. Це присвоєння, виклик функції, цикл, умовний оператор чи пустий оператор. Оператор завершується крапкою з комою. Додаткове групування операторів здійснюється за допомогою фігурних дужок {}. Група операторів (блок) є оператором.

Мова PHP підтримує 8 примітивних типів: 4 скалярних та 4 складних типи:

`boolean` – булевий літерал задається ключовими словами TRUE або FALSE (нечутливих до регістру символів):

```
$foo = True;
```

`integer` – цілі числа приймають значення з множини Z = {..., -2, -1, 0, 1, 2, ...}. Вони можуть бути десятковими (основа 10), 16-ковими (основа 16) першими символами числа є `0x`, або 8-вими (основа 8) – перший символ числа є 0 (нуль), з необов'язковими знаками - або +.

Наведемо приклад цілих чисел:

```
$a = -123;
$a = 0123; # 8-кове число
$a = 0x1A; # 16-кове число
```

Довжина *integer* залежить від розрядності.

`float` – числа з плаваючою комою (крапкою) (*float, double, real*) можна утворити такими присвоюваннями:

```
$a = 1.234;
$a = 1.2e3;
```

`string` – це рядок символів. На їх довжину в PHP немає обмежень. Літерал можна утворити трьома способами: одинарними лапками, подвійними лапками, `heredoc`-синтаксисом.

Перший спосіб є найпростіший (символ '), оскільки стрічка не опрацьовується додатково. Коса зворотна ставиться, якщо службові символи не треба чи треба виводити на друк:

```
echo 'Це проста стрічка.';
echo 'Arnold once said: "I\'ll be back"';//  ...."I'll be back"
echo 'Are you sure you want to delete C:\\*.*?'; //... C:\*.*?
```

Якщо стрічка обмежена подвійними лапками ("), PHP опрацьовує escape-послідовності (мнемонік) спеціальних символів (змінні розгортаються):

`\n` – прогон стрічки,

`\r` – повернення каретки ,

`\t` – горизонтальна табуляція,

`\\` – зворотна нахилена риска,

`\$` – знак долара,

`\"` – подвійні лапки,

`\[0-7]{1,3}` – послідовність символів, які збігаються з регулярним виразом, символи в 8-ковій нотації,

`\x[0-9A-Fa-f]{1,2}` – послідовність символів, які збігаються з регулярним виразом, символи в 16-ковій нотації.

Важливим способом обмеження рядка є синтаксис heredoc ("<<<"). Після <<< ставиться ідентифікатор, рядок і знову цей же ідентифікатор для закриття рядка (він мусить починатись з першого стовпчика рядка). Ідентифікатор формується за правилами PHP, починається з не-цифри чи символа підкреслення. Рядок не містить більше ніяких символів, крім ; . Це означає, що ідентифікатор не може вводитися з відступом і що не може бути ніяких прогалин та знаків табуляції до та після крапки з комою. Heredoc текст подібний до рядка з подвійними лапками.

Простий приклад:

```
<?php
$str = <<<EOD Приклад стрічки синтаксису heredoc.
EOD;
```

Розглянемо деякі приклади:

```
$great = 'fantastic';
echo "This is { $great}"; //вивід: This is {
fantastic}
echo "This is {$great}"; // вивід: This is fantastic
```

Розглянемо приклади роботи з символами :

Операція конкатенації:

```
<?php
$str = "This is a string";
$str = $str . " with some text"; // Конкатенація
```

або

```
$str .= " and a newline at the end.\n";
```

Різні типи лапок:

```
$num = 9;
$str = "<p>Number: $num</p>"; // <p>Number: 9</p>
```

або

```
$num = 9;
$str = '<p>Number: $num</p>';// <p>Number: $num</p>
```

Доступ до символів за зміщенням:

```
$str = 'This is a test.';
$first = $str{0};      //перший символ рядка
$last = $str{strlen($str)-1}; //останній символ рядка
?>
```

Перетворення типів або конвертація рядків здійснюється за правилами:

Призначається тип `float`, якщо стрічка містить один з символів `'.', 'e' чи 'E'`. В інших випадках тип є `integer`.

```
$foo = 1 + "-1.3e3";                // $foo – це float (-1299)
$foo = 1 + "bob-1.3e3";             // $foo – це integer (1)
$foo = 1 + "bob3";                  // $foo – це integer (1)
$foo = 1 + "10 Small Pigs";         // $foo – це integer (11)
$foo = 4 + "10.2 Little Piggies"; // $foo – це float (14.2)
$foo = "10.0 pigs " + 1;            // $foo – це float (11)
$foo = "10.0 pigs " + 1.0;          // $foo – це float (11)
```

Для тестування прикладів треба використати таку команду:

```
echo "\$foo==$foo; type is " . gettype ($foo) .
"<br />\n";
```

Конкатенація рядків здійснюється операцією '.' . Для роботи з рядками є потужній апарат функцій, зокрема: рядкових, для їх кодування/декодування (`mcrypt и mhash`), для роботи з URL-рядками тощо.

Адресний тип призначений для передачі змінних. Наприклад, локальна змінна у функції та змінна в області вказують на одне місце:

```
function foo (&$var)
{
    $var++;
}
$a=5;
foo ($a);
```

Змінній `$a` відповідає значення 6.

Адресний тип надає можливість двом змінним вказувати на одне значення:

```
$a =&$b
```

`$a i $b` вказують на одне місце.

Константи позначаються за допомогою функції `define()`. Це можуть бути скалярні величини типів `boolean, integer, float, string`. В імені константи не використовується знак `$`. Функція `constant()` для читання значень констант, якщо ім'я кон-

станти задано динамічним чином. Функція `get_defined_constants()` дає змогу отримати список усіх констант.

```
<?php
define("CONSTANT", "Hello world.");
echo CONSTANT; // "Hello world."
?>
```

Масив у PHP використовуються різними способами: реальний масив, список (вектор), хеш-таблиця, словник, колекція, стек, черга тощо. Масив створюється за допомогою ключового слова `array()`. Він заповнюється певною кількістю розділених комою пар `key => value` (ключ-значення):

```
array( [key =>] value
     , ...
     )
```

де `value` – довільна змінна. Ключ має тип `integer` або `string`. Значення довільне. За відсутності ключа, береться максимальний індекс, а новий `key` приймається збільшеним на 1. Це діє і для від'ємних значень, наприклад, якщо максимальний індекс = – 6, то новий ключ = -5. За відсутності цілочисельного індексу `key` = *0*. Специфікація заповненого ключа змінює його значення. `true` як ключ дає `integer 1`, `false` - `integer 0`, `NULL` – порожня стрічка. Об'єкт не може бути ключем.

Значення змінюються присвоюванням. Пара `key/value` знищується функцією `unset()`:

```
$a = array( 1 => 'one', 2 => 'two', 3 => 'three' );
unset( $a[2] ); // масив $a = array( 1=>'one',
3=>'three');
```

Розглянемо приклади різного запису однакових масивів:

```
     $a = array( 'color' => 'red'    //однією
командою
     , 'name'  => 'apple'
     ,            4        // key = 0
     );
$a['color'] = 'red'; //декількома командами
$a['name'] = 'apple';
```

```
$a[]          = 4;          // key = 0
```

Інші приклади ілюструють відсутність ключів:

```
$b[] = 'a';
$b[] = 'b';
$b[] = 'c'; // масив array( 0 => 'a' , 1 => 'b' , 2
=> 'c' ),
```

або

```
$switching = array(10              // key = 0
,5     =>  6
,3     =>  7
,'a'   =>  4
,11                          // key = 6  (бо максимальний
індекс був 5)
,'8'   =>  2           // key = 8
,'02' => 77            // key = '02'
, 0     => 12          //  значення 10 змінюється на 12 (див.
першу пару)
);
$empty = array();      // пустий масив
```

Об'єкти ініціалізує оператор *new*:

```
<?php
class foo
{
    function do_foo()
    {
        echo "Doing foo.";
    }
}
$bar = new foo;
$bar->do_foo();
?>
```

Ресурс – це спеціальна змінна типу resource, яка містить посилання на зовнішній ресурс. Вони створюються спеціальними функціями (PHP 4). Звільнення ресурсів автоматизоване. Вказівники на БД не знищуються збирачем сміття.

NULL змінна типу NULL. Змінна має значення NULL, якщо їй присвоєна константа NULL, не присвоєне ніяке значення, вона була unset()

Область видимості змінних. Усі змінні PHP поза функціями мають єдину область видимості. Ця область охоплює також include- і required-файли. Наприклад:

```
$a = 1;
include "b.inc";// Змінна $a буде доступна всередині команд
```

файла *b.inc*.

Всередині функцій є локальна область видимості. Наприклад:

```
$a = 1; // глобальна область
function Test()
{
    echo $a; // змінна локальної області
}
Test();//оператор echo стосується локальної змінної
```

У PHP глобальні змінні позначаються у функції, якщо вони використовуються в цій функції. Наприклад:

```
$a = 1;
$b = 2;
function Sum()
{
    global $a, $b;
    $b = $a + $b;
}
Sum();
echo $b;                    // результат "3".
```

Інший спосіб доступу до змінних глобальної області – використати масив $GLOBALS. Масив $GLOBALS є асоціативним. Елемент масиву – назва змінної як ключ (key) та її значення (value):

```
$a = 1;
$b = 2;
function Sum()
{
    $GLOBALS["b"] = $GLOBALS["a"] + $GLOBALS["b"];
}
Sum();
echo $b;  //  3
```

Для керування видимістю змінних використовується ключове слово static. Статична змінна існує тільки в локальній області

видимості функції, вона не втрачає значення, виходячи з області видимості. Розглянемо два приклади:

```
function Test ()
{
    $a = 0;
    echo $a;
    $a++;
}
```

Ця функція друкує "0". У разі виходу з неї змінна `$a` губиться:

```
function Test()
{
    static $a = 0;
    echo $a;
    $a++;
}
```

У другому випадку значення `$a` поновлюється для кожного виконання функції. `Static`-змінні використовуються для зупинки рекурсивних функцій:

```
function Test()
{
    static $count = 0;

    $count++;
    echo $count;
    if ($count < 10) {
        Test ();
    }
    $count--;
}
    ?>
```

Елементи структурного програмування. Конструкція структурного програмування в мові `PHP` є аналогічною до мови С.

Перерахуємо основні структури:

Розгалуження:

- `if (expr) statement`
- `if (expr) statement1 else statement2`

- `if (expr1)statem1 else if(expr2) statement2 else statement3`

Оператори повторення:

- `while (expr) statement`
- `while (expr): statement ... endwhile;`
- `do statement while (expr)`
- `for (expr1; expr2; expr3) statement`
- `for (expr1; expr2; expr3): statement; ...; endfor;`
- `oreach(array_expression as $value) statement`
- `foreach(array_expression as $key => $value) statement`

Мова PHP має **альтернативний синтаксис** для деяких структур керування: `if`, `while`, `for`, `foreach` і `switch`. У кожному випадку базова форма синтаксису змінюється – фігурні дужки замінюються: початкова на двокрапки ":", а закриваюча – на `endif;`, `endwhile`, `endfor;` `endforeach` чи `endswitch`, відповідно.

```
<?php if ($a == 5): ?>
A is equal to 5
<?php endif; ?>
```

Функції:

```
function name(parameterName, ..., parameterName) {
    statements;
}
function quadratic($a, $b, $c) {
    return -$b + sqrt($b*$b - 4*$a*$c) / (2*$a);
}
```

Тип параметрів та поверненого значення не описується.

Виклик функцій:

```
name(parameterValue, ..., parameterValue);
$root = quadratic(1, $x, $a + 3);
```

Для **створення класів** використовується ключове слово `class`. Клас є шаблоном для створення об'єктів. Він містить дані і

методи для їх опрацювання. Для позначення наслідування використовується ключове слово `extends`.

Розглянемо приклад класу:

```
<?php
class Cart
{
var $items;  //предмети
function add_item($artnr, $num) //Додати $num предметів
    {
        $this->items[$artnr] += $num;
    }
    function remove_item ($artnr, $num) // забрати предмети
    {
if ($this->items[$artnr] > $num) {
$this->items[$artnr]= $num;
return true;
        } else {
          return false;
        }
    }
}
?>
```

Клас `Cart` містить асоціативний масив товарів та дві функції. Для доступу до змінних класу і функцій всередині класу використовується псевдозмінна `$this`.

Об'єкти (змінні) класу створюються оператором `new`, а елементи масиву заносяться функцією:

```
<?php
$cart = new Cart;
$cart->add_item("10", 1);
// АБО $cart->items = array("10" => 1);
?>
```

7.3. Передача параметрів між HTML- та PHP-сторінками

У PHP-технології виділяємо 4 способи передачі даних між сторінками: через файли та бази даних, за допомогою глобальних масивів `$_GET[..]` та `$_POST[..]`, за допомогою глобального масиву сесій `$_SESSION['..']`та за допомогою глобального масиву обліку повідомлень `$_COOKIE[..]`.

PHP-скрипт отримує параметри запиту з HTML-форми web-браузера для опрацювання і підготовки відповіді за допомогою глобальних асоціативних масивів $_GET та $_POST, наприклад,

```
$cc = $_GET["creditcard"];        # у запиті метод GET
$username = $_POST["username"]; # у запиті метод POST
```

Асоціативні масиви $_GET $_POST називаються картами і вони є частиною всіх глобальних масивів сервера: `$_COOKIE`, `$_SERVER`, `$_FILES`, `$_ENV`, `$_REQUEST`, `$_SESSION`. Кожен з масивів має призначення для зберігання спільних параметрів клієнта та сервера (частина розглядається пізніше).

Для перевірки наявності відповідних параметрів використовується функція перевірки наявності ключа в масиві:

```
if (array_key_exists("creditcard", $_GET)) {
    $cc = $_GET["creditcard"];
    ...
} else {
    print("Error, you did not submit a credit card
number.");
    ...
    return;
}
```

Нижче наведені приклади функцій використання глобальних масивів для зберігання, що характеризують процеси з'єднання, підтримки сесії, даних запиту чи сервера тощо:

```
function global_session($var){
    if(!array_key_exists($var,$_SESSION))
        $_SESSION[$var]='';
```

```
        $GLOBALS[$var]=&$_SESSION[$var];
    }

    function global_cookie($var){
        if(!array_key_exists($var,$_COOKIE))
            $_COOKIE[$var]='';
        $GLOBALS[$var]=&$_COOKIE[$var];
    }

    function global_server($var){
        if(!array_key_exists($var,$_SERVER))
            $_SERVER[$var]='';
        $GLOBALS[$var]=&$_SERVER[$var];
    }

    function global_files($var){
        if(!array_key_exists($var,$_FILES))
            $_FILES[$var]='';
        $GLOBALS[$var]=&$_FILES[$var];
    }

    function global_env($var){
        if(!array_key_exists($var,$_ENV))
            $_ENV[$var]='';
        $GLOBALS[$var]=&$_ENV[$var];
    }

    function global_request($var){
        if(!array_key_exists($var,$_REQUEST))
            $_REQUEST[$var]='';
        $GLOBALS[$var]=&$_REQUEST[$var];
    }
?>
```

Для підтримки сеансів роботи серверних програм з одним і тим самим клієнтом у мові PHP є апарат зберігання інформації про поточні сесії та ідентифікації користувача певним кодом у змінній PHPSESSID. Сервер генеруєкод, передає його клієнтові, останній себе ним ідентифікує, передаючи цей код у стрічці параметрів запиту.

Перед основним текстом сторінки (головної сторінки) виконується функція:

```
<?php session_start(); ?>
```

Цей код реєструє сесію користувача, який викликає сторінку. Далі у глобальному масиві $_SESSION необхідно призначити унікальний ідентифікатор сесії:

```
<?php
session_start();
// зберегти дані сесії
$_SESSION['views']=1;
?>
```

Цей ідентифікатор, відповідно, може бути прочитаний:

```
<html>
<body>
<?php
//вивести дані
echo "Pageviews=". $_SESSION['views'];
?>
</body>
</html>
```

У разі виклику сторінки іншим користувачем ідентифікатор збільшується:

```
<?php
session_start();
if(isset($_SESSION['views']))
$_SESSION['views']=$_SESSION['views']+1;
else
$_SESSION['views']=1;
echo "Views=". $_SESSION['views'];
?>
```

Для знищення даних про сесію використовуються дві функції: session_destroy() - знищується сесія, unset() - для звільнення змінних сесії:

```
<?php
unset($_SESSION['views']);
?>
```

```
<?php
session_destroy();
?>
```

Другий механізм ідентифікації користувача – це створення повідомлень Cookie за допомогою функції setcookie(name, value, expire, path, domain), яка викликається перед відправлення HTML-коду до користувача.

Параметри мають таке навантаження: name – назва повідомлення, value – значення (автоматично кодується URLencode під час надсилання і розкодовується під час отримання), expire – часова межа існування (с).

```
<?php
setcookie("user", "Alex Porter", time()+3600);
?>
```

Значення повідомлення отримується в глобальному масиві $_COOKIE :

```
<?php
echo $_COOKIE["user"];
print_r($_COOKIE);
?>
```

У наступному прикладі виводиться значення повідомлення та функція його знищення:

```
<html>
<body>
<?php
if (isset($_COOKIE["user"]))
  echo "Welcome " . $_COOKIE["user"] . "!<br />";
else
  echo "Welcome guest!<br />";
setcookie("user", "", time()-3600);
?>
</body>
</html>
```

7.4. Додаткові можливості

Наведемо приклади найуживаніших функцій PHP для полегшення опрацювання даних.

Функції роботи з рядками : `explode, implode, strlen, strcmp, strpos, substr, strtolower, strtoupper, trim`

```
$length = strlen($favoriteFood);
$cmp = strcmp($name, "Jeff Prouty"); # > 0
$index = strpos($name, "e");    # 1
$first = substr($name, 8, 4);   # "Kuan"
$upper = strtoupper($name);      # "KENNETH KUAN"
```

Функції роботи з числами: `abs, ceil, floor, max, min, rand, round, srand...`

```
$piApprox = 355/113; # double: 3.141592920354
(int) $piApprox;     # int: 3
round($piApprox);    # double: 3
```

Перетворення рядка на число: intval()

Математичні константи : M_PI, M_E, M_LN2

Читання каталогів:

```
$DIR = "awesomeFiles";
$dh = opendir($DIR);
while ($file = readdir($dh)) {
    print("$file<br>\n");
}
closedir($dh);
```

- `opendir()` – читання каталога та повернення вказівника на нього;
- `readdir()` – читання назви одного файла;
- `closedir()` – зупинка читання.

Читання файлів:

```
$text = file_get_contents("filename");
$lines = preg_split("/\n/", $text);
foreach ($lines as $line) {
    do something with $line;
}
```

- `file_get_contents` повертає рядком цей файл;
- `file_set_contents` записує рядок у файл

Приклад підрахунку порожніх рядків

```
function count_blank_lines($file_name) {
    $text = file_get_contents($file_name);
    $lines = preg_split("/\n/", $text);
    $count = 0;
    foreach ($lines as $line) {
        if (strlen(trim($line)) == 0) {
            $count++;
        }
    }
    return $count;
}
...
print(count_blank_lines("15-php.html"));
```

Побудова рядка запиту за допомогою функції urlencode

```
$str = "Marty's cool!?";
$str2 = urlencode($str);    # "Marty%27s+cool%3F%21"
$url = "http://example.com/thing.php?param=$str2";
$str3 = urldecode($str2);  # "Marty's cool?!"
```

Функція urldecode декодує рядок запиту

Функції формування заголовка:

```
header("Content-type: text/plain");
header("Content-type: application/xml");
header("HTTP/1.1 400 Invalid Request");
header("HTTP/1.1 404 File Not Found");
header("HTTP/1.1 500 Server Error").
```

Функції асоціативних масивів:

- array_key_exists: чи існує значення для цього ключа;
- array_keys, array_values: повертають усі ключі чи значення масивам;
- asort, arsort: сортують масиви за значеннями;
- ksort, krsort: сортують масиви за ключами;
- print_r: друкує ключі та їхнє значення;
- var_dump: друкує ключі та їхні значення, а також змінні types/sizes.

7.5. Формування файлів програми на сервері

Розглянемо приклад PHP програми для авторизації користувача ресурсу. Традиційне вікно для заповнення імені та пароля користувача має вигляд, як на рис. 7.2.

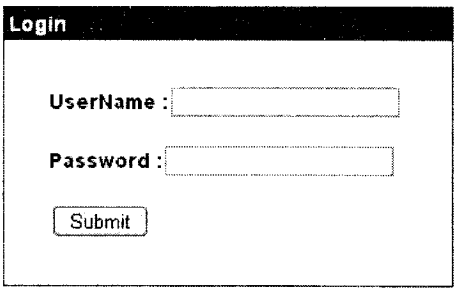


Рис. 7.2. Форма для заповнення даних користувача

Основною частиною HTML-коду цієї сторінки є тег форми з відповідними етикетками та елементами керування (введення даних і кнопка надсилання):

```
<form action="" method="post">
<label>UserName :</label>
<input type="text" name="username"/><br />
<label>Password :</label>
<input type="password" name="password"/><br/>
<input type="submit" value=" Submit "/><br />.
</form>
```

Програма авторизації на сервері складається з двох частин по два файли (рис. 7.3):

Рис. 7.3. Форма для заповнення даних користувача

Скрипт Login.php читання введених текстових даних, пошуку аналогічних даних у базі даних (таблиці) та видачі повідомлення про відсутність реєстрації або про допуск до ресурсу. Скрипт Config.php призначений для авторизації користувача бази даних та під єднання до неї.

Друга частина – це скрипт видачі повідомлення та верифікації користувача. Попередньо необхідно створити відповідну таблицю в базі даних (приклад для MySQL):

Database

```
MySQL таблиця admin з полями id, username, passcode.
CREATE TABLE admin
(
id INT PRIMARY KEY AUTO_INCREMENT,
username VARCHAR(30) UNIQUE,
passcode VARCHAR(30)
);
```

Config.php

```
<?php
$mysqli_hostname = "hostname";
$mysqli_user = "username";
$mysqli_password = "password";
$mysqli_database = "database";
$bd = mysqli_connect($mysqli_hostname, $mysqli_user,
$mysqli_password)
or die("Opps some thing went wrong");
mysqli_select_db($mysqli_database, $bd) or die("Opps some
thing went wrong");
?>
```

Login.php

```
PHP- та HTML-коди.
>?php
include("config.php");
session_start();
if($_SERVER["REQUEST_METHOD"] == "POST")
{
// username та password надіслані з форми
```

```
$myusername=addslashes($_POST['username']);
$mypassword=addslashes($_POST['password']);

$sql="SELECT id FROM admin WHERE username='$myusername'
and passcode='$mypassword'";
$result=mysqli_query($sql);
$row=mysqli_fetch_array($result);
$active=$row['active'];
$count=mysqli_num_rows($result);
// для знайдених $myusername, $mypassword кількість рядків = 1
if($count==1)
{
session_register("myusername");
$_SESSION['login_user']=$myusername;

header("location: welcome.php");
}
else
{
$error="Your Login Name or Password is invalid";
}
}
?>
```

lock.php

```
<?php
include('config.php');
session_start();
$user_check=$_SESSION['login_user'];

$ses_sql=mysqli_query("select username from admin where
username='$user_check' ");

$row=mysqli_fetch_array($ses_sql);

$login_session=$row['username'];
if(!isset($login_session))
{
header("Location: login.php");
}
?>
```

welcome.php

```
<?php
include('lock.php');
?>
<body>
<h1>Welcome <?php echo $login_session; ?></h1>
</body>
```

logout.php

Закриття сесії.

```
<?php
session_start();
if(session_destroy())
{
header("Location: login.php");
}
?>
```

7.6. Основні типи та функції доступу до БД

Для використання MySQL-сервера баз даних необхідно компілювати php програми з відповідною підтримкою (опція --with-mysql). Якщо шлях до mysql не вказується, php використовуватиме вбудовані в mysql бібліотеки клієнта. Специфікація шляху до mysql: --with-mysql=/path/to/mysql. Тоді php використовує відповідні бібліотеки.

Для використання MySQL-сервера баз даних необхідно компілювати php програми з відповідною підтримкою (опція --with-mysql). Якщо шлях до mysql не вказується, php використовуватиме вбудовані в mysql бібліотеки клієнта. Специфікація шляху до mysql: --with-mysql=/path/to/mysql. Тоді php використовує відповідні бібліотеки.

Сучасна версія PHP 5 підтримує доступ до бази даних MySQL командами, які в своїй назві використовую термін MySQLi для позначення об'єктно-орієнтованого та процедурного варіантів використання. Таке позначення використовується якщо реалізується

доступ тільки до MySQL і не передбачається переналаштування на іншу базу даних. Якщо ж в програмі можливі зміни джерела даних, раціонально використати об'єктно-орієнтований інтерфейс доступу PDO (PHP Data Objects), який передбачає підключення до 12 різних баз даних.

Продемонструємо на невеличких прикладах команди різних інтерфейсів :

Об'єктно-орієнтований підхід – під'єднання до бази даних :

```
<?php
.....
$conn = new mysqli($servername, $username,
$password);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn-
>connect_error);
}
echo "Connected successfully";
?>
```

Процедурний підхід – під'єднання до бази даних

```
<?php
.....
$conn = mysqli_connect($servername, $username,
$password);
// Check connection
if (!$conn) {
    die("Connection failed: " .
mysqli_connect_error());
}
echo "Connected successfully";
?>
```

Під'єднання до бази даних у PDO:

```
<?php
. . . . . .
try {
    $conn
= new PDO("mysql:host=$servername;dbname=myDB",
$username, $password);
    $conn->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
```

```
    echo "Connected successfully";
    }
catch(PDOException $e)
    {
    echo "Connection failed: " . $e->getMessage();
    }
?>
```

В такому ж порядку подамо приклади створення бази даних :

ООП :

```
<?php
. . . . . .
$sql = "CREATE DATABASE myDB";
if ($conn->query($sql) === TRUE) {
    echo "Database created successfully";
} else {
    echo "Error creating database: " . $conn-
>error;
}
$conn->close();
?>
```

ПП.:

```
<?php
. . . . . .
$sql = "CREATE DATABASE myDB";
if (mysqli_query($conn, $sql)) {
    echo "Database created successfully";
} else {
    echo "Error creating database: " .
mysqli_error($conn);
}
mysqli_close($conn);
?>
```

PDO :

```
<?php
. . . . . .
try {
    . . . . . .
    $sql = "CREATE DATABASE myDBPDO";
    $conn->exec($sql);
    echo "Database created successfully<br>";
    }
```

```
catch(PDOException $e)
    {
    echo $sql . "<br>" . $e->getMessage();
    }
$conn = null;
?>
```

Тепер наведемо приклади формування таблиць у різних інтерфейсах доступу, Спільними є команда SQL для створення таблиці :

```
CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP
)
```

ООП :

```
<?php
. . . . . .
$dbname = "myDB";
$sql = "CREATE TABLE MyGuests (
. . . . . .
)";
if ($conn->query($sql) === TRUE) {
    echo "Table MyGuests created successfully";
} else {
    echo "Error creating table: " . $conn->error;
}
$conn->close();
?>
```

ПП. :

```
<?php
. . . . . .
$dbname = "myDB";
. . . . . .
$sql = "CREATE TABLE MyGuests (
. . . . . .
)";
if (mysqli_query($conn, $sql)) {
    echo "Table MyGuests created successfully";
} else {
```

```
        echo "Error creating table: " .
    mysqli_error($conn);
    }
    mysqli_close($conn);
    ?>
```

PDO:

```
    <?php
    ..............
     $dbname = "myDBPDO";
     try {
     ..... . . . . . . . . .
     // sql to create table
     $sql = "CREATE TABLE MyGuests (
          . . . . . . . .
    )";
    $conn->exec($sql);
    echo "Table MyGuests created successfully";
    }
 catch(PDOException $e)
    {
    echo $sql . "<br>" . $e->getMessage();
    }
 $conn = null;
 ?>
```

Доступ до даних MySQL-сервера з PHP-програм реалізується функціями PHP у декілька етапів: 1) з'єднання з сервером баз даних, 2) вибір бази даних, 3) доступ до таблиці, 4) доступ до запису, 5) доступ до елемента стопчика (поля).

У такій послідовності розглянемо реалізацію доступу функціями PHP. Для того, щоб вибрати дані, треба спочатку створити базу даних, відповідні таблиці та занести у них записи, що складаються з полів певних форматів.

Створюємо базу даних "my_db":

```
<?php
$con = mysqli_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysqli_error());
  }
if (mysqli_query("CREATE DATABASE my_db",$con))
```

```
  {  echo "Database created"; }
else
  { echo "Error creating database: " .
mysqli_error();  }
// Create table
mysqli_select_db("my_db", $con);
$sql = "CREATE TABLE Persons
(
personID int NOT NULL AUTO_INCREMENT,
PRIMARY KEY(personID),
FirstName varchar(15),
LastName varchar(15),
Age int
)";
// Execute query
mysqli_query($sql,$con);

mysqli_close($con);
?>
```

У поданому фрагменті використано функції: з'єднання з сервером, вибору бази даних, виконання запиту та закриття з'єднання:

```
$con = mysqli_connect("localhost","peter","abc123")
mysqli_select_db("my_db", $con)
mysqli_query($sql,$con)
mysqli_close($con)
```

Для виконання запитів програмісту необхідно знати мову запитів SQL (у прикладі дві команди створення бази і таблиці виділені). Простий зміст фрагмента вказує, що створена таблиця `Persons` на три стовпчики: два по 15 символів на ім'я і прізвище, третій – для цілого значення віку.

Наступним прикладом (без команд з попереднього прикладу) демонструється занесення даних у створену таблицю (виділений текст представляє команду мови SQL):

```
<?php
. . . . . .
mysqli_select_db("my_db", $con);
mysqli_query("INSERT INTO Persons (FirstName,
LastName, Age)
VALUES ('Peter', 'Griffin',35)");
mysqli_query("INSERT INTO Persons (FirstName,
```

```
LastName, Age)
VALUES ('Glenn', 'Quagmire',33)");
mysqli_close($con);
?>
```

Головним завданням серверної програми в роботі з базою даних є занесення даних з HTML-форми у таблицю. В нижче наведеній формі дані вводяться за допомогою тегів `input`:

```
<html>
<body>
<form action="insert.php" method="post">
Firstname: <input type="text" name="firstname" />
Lastname: <input type="text" name="lastname" />
Age: <input type="text" name="age" />
<input type="submit" />
</form>
</body>
</html>
```

Серверна PHP програма таким кодом заносить надіслані в глобальний масив параметри

```
<?php
. . . . . .
$sql="INSERT INTO Persons (FirstName, LastName, Age)
VALUES
('$_POST[firstname]','$_POST[lastname]','$_POST[age]
')";

if (!mysqli_query($sql,$con))
  {
  die('Error: ' . mysqli_error());
  }
echo "1 record added";
mysqli_close($con);
?>
```

Занесені дані за необхідності виводяться у код HTML-сторінки. Приклад реалізації виведення даних у таблицю наведено нижче. Як і у попередніх прикладах з'єднання і вибір бази даних пропущено.

```
<?php
. . . . . .
$result = mysqli_query("SELECT * FROM Persons");
```

```
echo "<table border='1'>
<tr>
<th>Firstname</th>
<th>Lastname</th>
</tr>";

while($row = mysqli_fetch_array($result))
  {
  echo "<tr>";
  echo "<td>" . $row['FirstName'] . "</td>";
  echo "<td>" . $row['LastName'] . "</td>";
  echo "</tr>";
  }
echo "</table>";
mysqli_close($con);
?>
```

Наведемо основні функції MySQL-сервера, що підтримують доступ до баз даних з PHP-програм з урахуванням параметрів, які вони приймають та повертають:

• `int` **`mysqli_close`** `([int link_identifier])` - закриває з'єднання з MySQL-сервером (пов'язаним з іменем користувача). Без вказування `link_identifier` використовується відносно останнього. **`mysqli_close()`** не закриває з'єднань, утворених командою **`mysqli_pconnect()`**.

• `int` **`mysqli_connect`** `([string hostname [:port] [:/path/to/socket] [, string username [, string password]]])` - повертає додатний ідентифікатор з'єднання з MySQL у разі успіху і повідомлення про помилку в разі невдачі. **`mysqli_connect()`** встановлює з'єднання з MySQL-сервером. За відсутності параметрів за замовчуванням приймаються: *`host:port`* = 'localhost:3306', *`username`* = назва користувача, що володіє процесом на сервері, *`password`* = пусте.

Рядок назви хоста може містити номер порта, тобто "hostname:port" або шлях до сокета, тобто ":/path/to/socket" для локального хосту. Знак '@' перед функцією блока є повідомлення про помилки. Повторний виклик **`mysqli_connect()`** з тими самими аргументами не утворює нового з'єднання, а повертає

ідентифікатор попереднього з'єднання. З'єднання до сервера закривається по завершенні коду програми або функцією **mysqli_close()**.(див. попередній приклад),

• int **mysqli_create_db** (string database name [, int link_identifier])- для створення нової бази даних на сервері для вказаного ідентифікатора і:

• int **mysqli_data_seek** (int result_identifier, int row_number) - переводить вказівник рядка об'єкта виведення на вказаний номер рядка(запису). Функція **mysqli_fetch_row()** поверне вказівник назад. Наведемо приклад роздруку двох полів таблиці у зворотному порядку (окремі функції пояснюються нижче):

• int **mysqli_db_name** (int result, int row [, mixed field]) - бере як перший параметр вказівник об'єкта результату (список баз даних) з виклику функції **mysqli_list_dbs()** і повертає назву бази даних на основі порядкового номера. У разі помилки повертається FALSE. Функції **mysqli_errno()** та **mysqli_error()** подають природу помилок. Наведемо приклад видруку списку баз даних:

• int **mysqli_db_query** (string database, string query [, int link_identifier])- виконує запит у базі даних. Якщо ідентифікатор з'єднання не вказано, функція шукає відкрите з'єднання з MySQL-сервером. За відсутності такого зв'язку вона намагається його встановити (аналогічно функції **mysqli_connect()** без аргументів).

• int **mysqli_drop_db** (string database_name [, int link_identifier]) - знищує базу даних на сервері на основі ідентифікатора з'єднання.

• int **mysqli_errno** ([int link_identifier]) - повертає номер останньої помилки або 0 , якщо помилки не було.

• string **mysqli_error** ([int link_identifier]) - повертає текст помилки або порожній рядок, якщо помилки не було:

• array **mysqli_fetch_array** (int result [, int result_type]) - повертає масив позначеного рядка (запису) вбо false, якщо рядків більше нема. Функція є розширеною версією функції **mysqli_fetch_row()**. Додатково до розміщення рядка у пронумерованих елементах масиву функція розміщує дані в асоційованих індексах, використовуючи назви полів як ключі.

Якщо два і більше стовпчики мають однакову назву полів, останній стовпчик має пріоритет. Доступ до інших стовпчиків з тією самою назвою можливий за індексом стовпчика або його псевдонімом. Необов'язковий другий аргумент result_type у функції **mysqli_fetch_array()** є константою і набуває таких значень: MYSQL_ASSOC, MYSQL_NUM, або MYSQL_BOTH для характеристики типу масиву. Повертає асоційований масив позначеного рядка або false за відсутності рядків. Функція еквівалентна виклику функції **mysqli_fetch_array()** з другим параметром MYSQL_ASSOC. Повертає асоційований масив. Для отримання числових індексів та асоційованих ключів треба використовувати функцію **mysqli_fetch_array()**.

• object **mysqli_fetch_field** (int result [, int field_offset]) - повертає об'єкт з інформацією про поля, назви яких містить запит. Якщо не задано зміщення розташування (порядок) поля, видається наступне ще не видане цією функцією.

Контрольні запитання та вправи

1. Які типи структур даних використовуються у PHP? Навести приклади.
2. Які оператори структурного програмування використовуються у PHP? Навести приклади.
3. Багатовимірні масиви у PHP. Навести приклади.
4. Переваги та недоліки хеш-масивів у PHP. Навести приклади.

5. Створити фрагмент програми для читання файла у хеш.
6. Особливості проектування класів у PHP.
7. Порівняльний аналіз операторів циклів у PHP.
8. Створити схему програми для гри "хрестики–нулики".
9. Яким способом керувати типом змінних у PHP?
10. Створити фрагмент програми для записування хешу у файлі.
11. Створити об'єкти для їхнього використання у програмі "хрестики–нулики".
12. Порівняти класи в PHP та в іншій мові ООП.
13. Мовою PHP створити програму побудови частотного словника тексту.
14. Розробити принципи побудови пошукового сервера засобами мови PHP.
15. Розробити деревоподібну структуру даних за допомогою асоціативного масиву.
16. Подати структуру файлів доступу до даних.
17. Навести приклади функцій читання даних з таблиць.
18. Сформувати таблиці у базі даних.
19. Модифікувати дані у таблиці за заданими ключами полів. Вивести дані з таблиці у поля web-таблиці.

Глава 8
РОЗРОБКА WEB-СТОРІНОК ЗАСОБАМИ ASP.NET

8.1. Форми-засоби зворотного зв'язку в HTML

Web-сервер – це програма, яка призначена для зберігання, передачі гіпертекстової інформації (сформованої на основі HTML чи ASP), модифікації її змісту. Відображається ця інформація за допомогою спеціальної програми – навігатора або броузера. Основна відмінність гіпертекстової інформації від звичайної текстової полягає в тому, що гіпертекстовий документ містить посилання на інший документ (або інший файл). Документ, на який посилаються, може бути розміщений на іншому сервері. Документ містить посилання на інші документи.

Основна операція web-сервера проілюстрована на рис. 8.1. Ця система складається з двох об'єктів: броузера і web-сервера. Між ними існує канал зв'язку. Броузер посилає запит на сервер, сервер відсилає відповідь назад. Ця архітектура є характерною для сервера, який надсилає звичайні статичні сторінки.

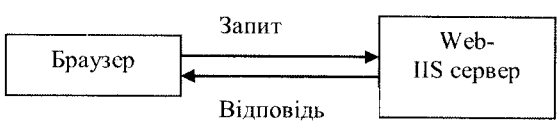


Рис. 8.1. Відношення типу клієнт/сервер між браузером і web-сервером

У випадку підтримки динамічних змін на web-сторінці на стороні сервера взаємодія програм-учасників динамічного сценарію проілюстрована на рис. 8.2, де сервер СУБД є SQL-сервер, а скрипти оновлення змісту сторінок написані мовою ASP (C#).

Для взаємозв'язку клієнта і сервера використовуються форми HTML. Тому для проектування модулів на стороні сервера

програміст повинен володіти знаннями про основні елементи мов HTML ASP, C# тощо – інструментів створення web-сторінок і особливо ланки для зв'язку сторінок з програмами, розташованими на сервері.

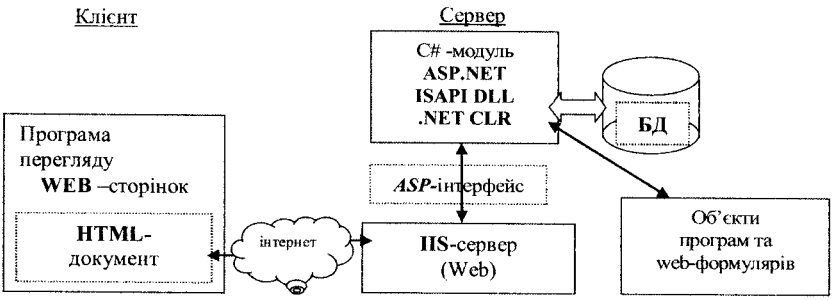


Рис. 8.2. Схема взаємодії клієнта і сервера

Середовище Visual Studio .NET дає змогу проектувати web-програми за допомогою певних компонент ASP.NET, що не потребує надавати багато уваги кодуванню мовою HTML. Компоненти забезпечують створення відповідних елементів управління для інтерфейсу користувача програми.

Насправді ASP.NET є бібліотекою динамічних компонент DLL ISAPI (Internet services Application Interface), що інкапсульована в CLR (Common Language Runtime) платформи .NET і дає змогу використовувати компоненти під час створення програм (рис. 8.2).

Використання Internet-форм забезпечує активну участь відвідувача у роботі сайту. Заповнивши форму, користувач може підписатися на додаткову інформацію, покритикувати сайт та інше. Крім того, за допомогою деяких елементів HTML-форм можна розширювати можливості web-сторінок.

Користувачеві за допомогою форми пропонується представити інформацію, вибравши один варіант з декількох або просто ввівши текст. Після заповнення форми подана користувачем інформація передається на сервер, де програма сортує її та

модифікує базу даних. Після завершення операції користувач отримує підтвердження про занесення даних.

Протокол HTTP [HyperText Transfer Protocol] запитів та відповідей слугує для встановлення зв'язку в момент доступу до сторінки.

Вказуючи адресу сторінки web-навігатор встановлює TCP-з'єднання з сервером (наприклад, через порт 80). Клієнт надсилає серверу запит і отримує відповідь.

Вони формуються за форматом "ключ : значення":

```
GET http://polynet.lviv.ua/index.html
If-Modified-Since: Thu, 31 Oct 2002 19:41:00 GMT
```

Відповідь сервера має вигляд:

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Sun, 17 Aug 2003 10:35:30 GMT
Content-Type: text/html
Last-Modified: Tue, 27 Mar 2001 10:34:52 GMT
Content-Length: XXX
<html>
  --- Тут знаходиться текст сторінки мовою HTML
</html>
```

Три цифри означають характер повідомлення:

- 1xx – інформаційне повідомлення,
- 2xx – вихід (200 OK),
- 3xx – перенаправлення (302 ресурси тимчасово відсутні),
- 4xx – помилки клієнта (400 помилковий запит, 401 неавторизовано, 403 заборонено),
- 5xx – помилки сервера (500).

Мова HTML дає можливість користувачеві надсилати інформацію на сервер засобами форм. Як приклад розглянемо таку форму:

```
<FORM METHOD="POST" ACTION="mailto:roman@acm.org">
  <INPUT TYPE="text" NAME="NAME" SIZE=30
MAXLENGTH=40>
  <TEXTAREA NAME="COMMENTS" ROWS=6 COLS=40>
</TEXTAREA>
```

```
<INPUT TYPE="submit" VALUE="Send information by e-
mail">
</FORM>
```

Вона має два поля типу (<INPUT TYPE="TEXT"...>)(<TEXTAREA ...> ...</TEXTAREA>).. для введення тексту та одну кнопку відсилання запиту з даними (<INPUT TYPE="SUBMIT"...>).

Крім того, можна використати : поле для введення пароля (<INPUT TYPE="PASSWORD"...>);

- вибрані дані з певного набору (<INPUT TYPE="RADIO"...>);
- списки для вибору значень (<SELECT> <OPTION> <OPTGROUP>);
- приєднані файли (<INPUT TYPE="FILE"...>);
- сховані параметри (<INPUT TYPE="HIDDEN"...>);
- адресат отримання інформації позначений у (URL) FORM ACTION.

Для надсилання запиту використовують кнопку або картинку (<INPUT TYPE="IMAGE"...>), результат від натискання на які однаковий.

8.2. Формуляри web

Для створення динамічних web-сторінок платформа .NET пропонує простір назв System.Web, який містить класи для створення засобів спілкування між навігатором та web-сервером (запити, відповіді, записки (кукі) тощо). Зокрема, похідні простори

- System.Web.UI,
- System.Web.UI.HtmlControls,
- System.Web.UI.WebControls

надають різні типи елементів управління для побудови інтерфейсу користувача web-програм.

Web-програми ASP.NET формуються на основі web-формулярів, які поділяються на файли типу .aspx (інтерфейс формуляра) та окремий файл коду (логіка програми). Для виконання їх треба помістити у каталог, доступний серверу IIS (наприклад, кореневий каталог wwwroot). Код програми компілюється автоматично і створюється збірка в кеш-пам'яті CLR. Якщо текст сторінки змінюється, то код логіки перекомпільовується автоматично. Якщо текст не змінюється, то запити використовують скомпільовану версію, що розташована в кеші. Цим механізмом підвищується ефективність web-програми з ASP.NET порівняно з попередніми версіями ASP.

Розглянемо сторінку, що складається з файлів двох типів:

1) інтерфейс формуляра-розширення .aspx

```
<% @Page Language="C#" Inherits="TodayPage"
Src="Today.cs" %>
<html>
   <body>
            <h1 align="center">
             Today we have <% OutputDay(); %>
            </h1>
   </body>
</html>
```

2) текст програми на C# –today.cs

```
using System;
using System.Web.UI;
public class TodayPage:Page
{
  protected void OutputDay()
  {
    Response.Write(DateTime.Now.ToString("D"));
  }
}
```

Можливий другий тип сторінки, коли файл .aspx містить і інтерфейс, і код опрацювання подій разом.

Сторінка .aspx успадковує клас, визначений у файлі коду. Останній успадковує клас System.Web.UI.Page (рис. 8.3). Методи класу позначаються як protected для дозволу доступу до них з файла .aspx.

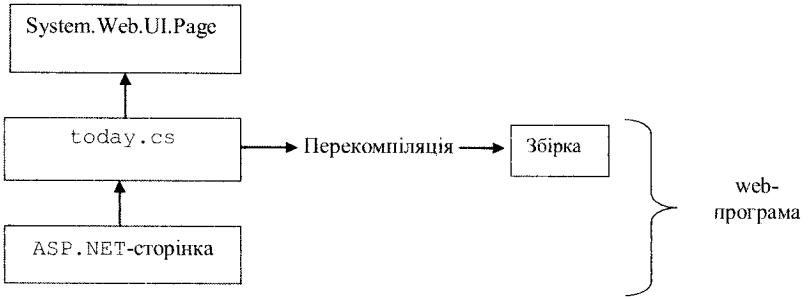


Рис. 8.3. Схема формування web-програми

Загалом під час побудови web-програм на основі ASP.NET проект містить, крім двох названих типів (сторінки .aspx і коду логіки .cs) файлів, конфігураційний файл web.config, файл видачі результатів аутентифікації login.aspx, файл аутентифікації login.aspx.cs (написані певною мовою програмування) та певних зовнішніх ресурсів (картинки, аудіо тощо).

8.3. Елементи управління web-сторінки

8.3.1. Типи елементів управлінння

Програми повинні містити коди відповідей на кожну подію, генеровану інтерфейсом під час доступу до елементів керування ASP.NET. Web-сервер інтерпретує етикетки та генерує відповідні візуальні елементи за допомогою HTML-коду.

Вирізняють три типи елементів управління ASP.NET:

- елементи управління HTML (традиційні HTML теги);
- web-елементи управління (етикетки ASP.NET);
- елементи управління перевірки цілості та наявності даних.

Елементи управління HTML трактуються сервером як текст і надсилаються до клієнта. Щоб зробити їх програмованими, треба використати атрибут runat="server".

У Visual Studio .NET можна використовувати елементи управління HTML, вказавши відповідну опцію меню під час проектування

web-формуляра "виконувати як елемент управління сервера". Додатково ASP.NET потребує, щоб усі елементи HTML були акуратно відкритими та закритими (як у XML-документах).

Усі елементи управління сторінки ASP.NET повинні міститися в межах тегу <form> з атрибутом runat="server":

```
<form runat="server">
  ... елементи HTML та елементи ASP.NET ...
</form>
```

Наступний приклад демонструє, як HTML-посиланнями (a - HtmlAnchor) можна динамічно завантажити сторінку з відповідною URL-адресою. Для цього треба присвоїти відповідне значення властивості HRef у коді опрацювання події, яка відбудеться на сторінці ASP.NET:

На сторінці ASP.NET маємо файл HTMLConrol.aspx:

```
<html>
  ...
  <body>
    <form id="HTMLControl" method="post"
runat="server">
      <a id="link" runat="server"> Visit our page!</a>
    </form>
  </body>
<html>
```

У наведеному файлі елементи HTMLControl та link є змінними файла коду реакції HTMLConrol.cs , з якими працюватиме сервер. Цей файл під час завантаження сторінки має вигляд:

```
public class HTMLControl : System.Web.UI.Page
  {
protected System.Web.UI.HtmlControls.HtmlAnchor link;
private void Page_Load(object sender, System.EventArgs e)
          {
          link.HRef = "http://daily.lviv.ua/";
          }
override protected void OnInit(EventArgs e)
          {
          this.Load += new
System.EventHandler(this.Page_Load);
          base.OnInit(e);
          }
  }
```

Програміст ініціалізує змінні сторінки за замовчуванням. Перед видачею клієнтові ASP.NET-сторінки сервер виконує метод `OnInit`, обробляє подію `Load`, пов'язану з завантаженням цієї сторінки, не генеруючи ніяких додаткових даних.

Перевантажений метод `OnInit` заносить у подію `Load` делегатом `EventHandler` метод `Page_Load`, що має виконуватись під час настання події – завантаження сторінки.

Метод `Page_Load` виконується щоразу під час виклику сторінки. Властивість `Page.IsPostBack` є `false`, якщо користувач викликає сторінку перший раз, і `true`, коли сервер повторяє реалізацію виклику. В наведеному прикладі під час завантаження сторінки `HTMLConrol.aspx` завантажиться сторінка `http://daily.lviv.ua;`

Основні елементи HTML-сторінки (значення `id` в коді сторінки та назви класів у бібліотеках `.NET` платформі), позначення їх назв (етикеток) у HTML-коді та пояснення їх функцій наведено в такій таблиці:

Елемент HTML (назва класу)	Етикетка HTML (назва в HTML-коді)	Опис
`HtmlAnchor`	`<a>`	Посилання на документи
`HtmlButton`	`<button>`	Кнопка
`HtmlForm`	`<form>`	Форма зворотного зв'язку
`HtmlGeneric`		Етикетка HTML не має назви
`HtmlImage`	`<image>`	Зображення
`HtmlInput...`	`<input type="...">`	Для введення даних: кнопки "button", "submit" та "reset", текст "text" та "password"), опції "checkbox" і "radio", зображення "image", файли "file", захований ввід "hidden".
`HtmlSelect`	`<select>`	Вибір певного значення
`HtmlTable...`	`<table> <tr> <td>`	Таблиці, рядки і стовпці
`HtmlTextArea`	`<textarea>`	Область тексту

Web-елементи управління. Другою складовою ASP.NET-сторінок є web-елементи управління ASP.NET, які потребують атрибута `runat="server"` і описуються у файлі `.aspx` за допомогою етикеток за формою

```
<asp:control id="identificator" runat="server" />
```

де `control` – тип web-елемента управління (етикетка, кнопка, список тощо), `identificator` – його назва – змінна доступу до елемента. Деякі елементи мають закриваючу дужку типу `</asp:control>`.

Наведемо конкретний приклад елемента управління – кнопки Button, для якої описується тип, назва, напис та назва функції для виконання після натискання кнопки:

```
<asp:Button ID="btnFillData" runat="server"
Text="Fill Grid" OnClick="btnFillData_Click" />
```

Щоб кнопка працювала, необхідно створити блок коду обробника подій на стороні сервера після натискання кнопки. Зауважимо, що параметри методу є метою делегата типу `System.EventHandler`:

```
<script runat="server">
protected void btnFillData_Click(object sender,
EventArgs e)
{
Button.Text = "The button was pressed";
}
</script>
```

Код цього скрипту записується безпосередньо в тілі сторінки файла .aspx.

У наступному прикладі позначимо файл елементів управління як `WebControl.aspx`. У секції `form` сторінки використаємо елемент – кнопку `Button`, під час натискання якої має бути видруковано текст на сторінці ASP.NET:

```
...
<form id="WebControl" method="post"
runat="server">
```

```
        <asp:Button id="Button" runat="server"
Text="Press the button"></asp:Button>
      </form>
      ...
```

Натискання на кнопку відстежується фрагментом тексту класу `WebControl (WebControl.cs)`, який розташований окремо від файла `WebControl.aspx`:

```
    public class WebControl : System.Web.UI.WebControls
    {
    private void Button_Click(object sender,
System.EventArgs e)
      {
        Button.Text = "The button was pressed";
      }
    }
```

Під час натискання кнопки спрацює реакція та зміниться текст на кнопці.

Елементи управління ASP.NET застосовуються подібно до того, як вони застосовуються у Windows-програмах. Набір таких елементів наведено у таблиці:

Web-елементи управління (назва класу)	Опис
1	2
`AdRotator`	Послідовність зображень (наприклад, банер)
`Button`	Кнопка
`Calendar`	Календар місяця
`CheckBox`	Вибір поля
`CheckBoxList`	Група елементів вибору
`DataGrid`	Таблиця даних
`DataList`	Список за шаблоном
`DropDownList`	Список, який розкривається
`HyperLink`	Посилання
`Image`	Зображення
`ImageButton`	Кнопка-зображення
`Label`	Етикетка
`LinkButton`	Кнопка-посилання

Продовження таблиці

1	2
ListBox	Список
Literal	Статичний текст
Panel	Панель-контейнер
PlaceHolder	Зберегти місце для динамічного додавання елементів
RadioButton	Радіо-кнопка
RadioButtonList	Група радіо-кнопок
Repeater	Дає змогу проглядати список елементів управління
Table	Таблиця
TextBox	Поле редагування
Xml	Демонстрація файла XML або XSL-перетворення

Елементи управління перевірки цілості призначені для видач повідомлення, якщо введені дані не відповідають наперед поставленим вимогам. Перевірка здійснюється автоматично після натискання кнопок Button, ImageButton, посилань LinkButton. Скасувати перевірку можна присвоєнням властивості CausesValidation значення false.

Для елементів контролю необхідно заповнити відповідні властивості, наприклад, елемент RangeValidator контролює область введених значень за допомогою властивості ControlToValidate (значення), повідомлення про помилку (ErrorMessage) та умови перевірки (тип Type=Integer, дозволена область значень між MinimumValue та Maximum Value).

Елементи контролю цілості даних наведено в такій таблиці:

Web-елементи керування з перевіркою	Опис
1	2
CompareValidator	Порівняння значень
CustomValidator	Дає змогу використовувати метод контролю введених значень

Продовження таблиці

1	2
RangeValidator	Перевірка перебування значення в межах
RegularExpressionValidator	Перевірка значення як компоненти регулярного виразу
RequiredFieldValidator	Обов'язкове введене значення
ValidationSummary	Сумарна інформація про всі помилки на сторінці

8.3.2. Шаблони фрагментів

За допомогою користувацьких елементів управління (user controls) в ASP.NET можна створити певні шаблони (фрагменти HTML-коду), що дають змогу швидко вносити зміни в раніше створену aspx-сторінку. Код шаблону також формується в ASP.NET і зберігається в .ascx-файлі. На основі коду створюється html-код для елемента управління, який вставляється у html-код сторінки під час її генерування.

Основним тегом файла є @ Control. Його параметри подібні до тега @Page. Заголовок шаблону містить такі атрибути: ClassName – клас для елемента управління; Language – мова програмування inline-коду сторінки; Inherits – базовий клас елемента управління.

Для використання шаблону на web-сторінці треба:

1) його зареєструвати (@ Register):

а) індивідуальна реєстрація :

```
<%@ Register TagPrefix="префікс" TagName="тег"
Src="текст_шаблону" %>
```

б) реєстрація замовного шаблона:

```
<% @ Register TagPrefix="префікс"
Namespace="простір_назв"
Assembly="назва_збірки" %>
```

2) вказати елемент у сторінці форматом

```
<префікс: тег>,
```

де префікс з тегу Register, а тег – це назва класу або задана назва.

Для відображення елементів управління, зображених на рис. 8.4, наведемо як приклад файл шаблону Calculator.ascx:

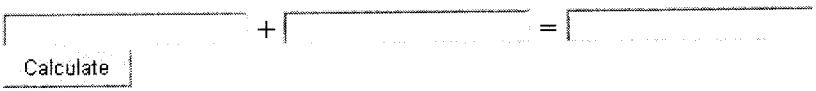


Рис. 8.4. Елементи калькулятора

```
<%@ Control Description="A simple calculator" %>
<asp:TextBox ID="Op1" runat=server/> +
<asp:TextBox ID="Op2" runat=server/> =
<asp:TextBox ID="Res" runat=server/>
<br/>
<asp:Button Text="Calculate"
            OnClick="OnCalculate" runat=server/>
<script language="C#" runat=server>
  private void OnCalculate(Object sender, EventArgs
e)
  {
    int res = Convert.ToInt32(Op1.Text) +
              Convert.ToInt32(Op2.Text);
    Res.Text = res.ToString();
  }
  public int Result
  {
    get { return Convert.ToInt32(Res.Text); }
  }
</script>
```

Тоді сама web-сторінка формується таким описом у файлі Calculator.aspx, що містить шаблон з керуючими елементами:

```
<%@ Page Language="C#" %>
<%@ Register TagPrefix="uc1" TagName="Calc"
    Src="Calculator.ascx" %>
<html>
  <body>
```

```
<form runat=server>
  <uc1:Calc id="UC1" runat=server/>
</form>
</body>
</html>
```

8.4. Зв'язування даних

Одним з елементів, що роблять Web-сторінки динамічними, є елементи управління інтерфейсу, які призначені для виконання операцій введення–виведення даних. Дані необхідно занести у відповідні змінні, масиви чи об'єкти. Можливі три способи прив'язування даних до відповідних об'єктів сторінки чи програми:

1. Описати дані в змінних, асоційованих з елементами управління (<asp:Button...Text= . . . />) – спосіб статичний;

2. Виконувати фрагменти програм з об'єктами, асоційованими з елементами управління форм. Цією формою звично користуються під час введення даних у прості списки та рядки, зокрема, під час опрацювання події завантаження web-сторінки процедурою Form_Load.

3. Використати спеціальні механізми зв'язування даних ("data binding") створенням об'єктів для даних (DataSet, Array тощо) і призначенням їх елементам контролю.

В останньому способі код відділено від інтерфейсу.

Можна підімкнути широкий спектр джерел даних: колекцій (Array, Hashtable тощо), класи даних (DataSet, DataTable, DataView, DataReader), файли XML, властивості, вирази, виклики методів тощо.

На сторінках ASP.NET можна включати вирази опрацювання даних у формі включень (аналогічно до включень JSP у мові Java):

<%# expresion %>.

Розглянемо приклади зв'язування даних згідно з вказаними способами.

1. До ASP.NET web-елементів керування `asp:RadioButtonList` (рис. 8.5), `asp:Listbox`, `asp:CheckBoxList`, `asp:DropDownList` відповідні текстові дані можна внести за допомогою компоненти `asp:ListItem`:

```
<html>
 <body>
  <form runat="server">
    <asp:RadioButtonList                    id="continent"
  runat="server">
      <asp:ListItem value="EU" text="EUROPE" />
      <asp:ListItem value="AM" text="AMERICA" />
      <asp:ListItem value="AF" text="AFRICA" />
    </asp:RadioButtonList>
  </form>
 </body>
</html>
```

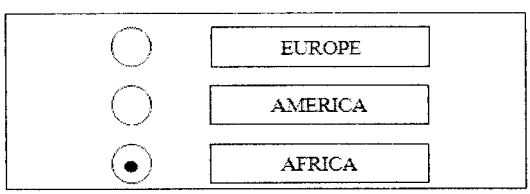


Рис. 8.5. Форма для заповнення параметрів користувача

2. Використання джерел даних незалежно від HTML-коду є зручнішим та швидшим для реалізації та модифікації коду. До одних з таких незалежних джерел належить і колекція `ArrayList`. Вона містить список об'єктів, якими можна заповнити список інтерфейсу. Для заповнення сторінки використовується метод `Add()`. Дані можна посортувати методами `Sort()` або `Reverse()`. Зв'язуємо дані колекції `ArrayList` з елементом управління `continent` на web-сторінці (попередній приклад) через властивість цього елемента `DataSource` його методом зв'язування `DataBind()`:

```
    private void Page_Load(object sender,
System.EventArgs e)
    {
      if (!Page.IsPostBack) {
         ArrayList list = new ArrayList();
         list.Add("EUROPE");
         list.Add("AMERICA");
         list.Add("AFRICA");
         list.Sort();
         continent.DataSource = list;
         continent.DataBind();
      }
    }
```

У цьому випадку елементи масиву використовуються як текст (Text) і як значення (Value), асоційоване з певними елементами списку інтерфейсу.

3. Для формування асоційованого масиву (ключ та його значення) можна використати колекції типів `Hashtable` чи `SortedList`. Для попереднього прикладу використаємо властивості: поле `DataTextField` для ключа і поле `DataValueField` для самого значення:

```
    private void Page_Load(object sender,
System.EventArgs e)
    {
      if (!Page.IsPostBack) {
         Hashtable table = new Hashtable();
         table.Add("EU", "EUROPE");
         table.Add("AM", "AMERICA");
         table.Add("AF", "AFRICA");

         continent.DataSource = table;
         continent.DataValueField="Key";
         continent.DataTextField="Value";
         continent.DataBind();
      }
    }
```

Незручністю таблиці `Hashtable` є неможливість сортування елементів списку. Для цієї мети можна використати колекцію

SortedList (заповнюється аналогічно до Hashtable і сортується автоматично).

Наведемо ще чотири типи елементів вибору та занесення даних вибору (на рис. 8.6 – Select, <asp:DropDownLis, <asp:CheckBoxList , <asp:ListBox).

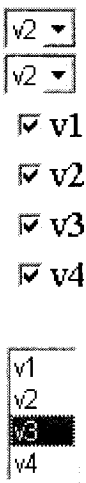


Рис. 8.6. Типи керуючих елементів вибору

Файл ArrayListBind.aspx для зв'язування даних має такий вид:

```
<%@ Page Language="C#" %>
<html>
  <body>
   <head>
<script runat=server>
void Page_Load(Object sender, EventArgs e)
{
  ArrayList vals = new ArrayList();
  vals.Add("v1");
  vals.Add("v2");
  vals.Add("v3");
  vals.Add("v4");

  s1.DataSource  = vals;
  cbl1.DataSource= vals;
  dd1.DataSource = vals;
  lb1.DataSource = vals;

  s1.DataBind();
}
</script>
</head>
 <form runat=server>
  <Select id="s1" runat=server /> <br/>
  <asp:DropDownList id="dd1" runat=server /> <br/>
  <asp:CheckBoxList id="cbl1" runat=server /> <br/>
  <asp:ListBox id="lb1" runat=server /> <br/>
  </form>
 </body>
</html>
```

8.5. Використання даних з файлів XML

Наведені в попередньому параграфі дані можна описати для інтерфейсу і за допомогою файла XML:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<continents>
  <continent>
       <id>EU</id>
       <name>EUROPE</name>
  </continent>
         <continent>
               <id>AM</id>
               <name>AMERICA</name>
  </continent>
         <continent>
               <id>AF</id>
               <name>AFRICA</name>
          </continent>
</continents>
```

З наведеного XML файла можна перенести дані у об'єкт класу `DataSet` переносом заголовків `id` та `name` у відповідні поля та зв'язуючи наповнений об'єкт з елементом управління інтерфейсу:

```
private void Page_Load(object sender, System.EventArgs e)
        {
        if (!Page.IsPostBack) {
            DataSet dataset = new DataSet();
            dataset.ReadXml(MapPath("continents.xml"));

            continent.DataSource = dataset;
            continent.DataValueField="id";
            continent.DataTextField="name";
            continent.DataBind();
           }
        }
```

Властивість `SelectedItem` списку вказує на вибрану користувачем опцію, а подія `SelectedItemChanged` генерується, коли користувач вибирає одну з опцій списку.

Кнопка asp:Repeater слугує для демонстрації всіх даних. Розташувавши її на web-формулярі та заповнивши відповідний об'єкт DataSet даними з файла XML, можна зв'язати asp:Repeater з даними таким кодом:

```
protected System.Web.UI.WebControls.Repeater
continent;
...
private void Page_Load(object sender,
System.EventArgs e)
{
   if (!Page.IsPostBack) {
      DataSet dataset = new DataSet();
      dataset.ReadXml(MapPath("continents.xml"));
      continent.DataSource = dataset;
      continent.DataBind();
   }
}
```

Для візуалізації таблиці від елемента web-сторінки asp:Repeater необхідно створити шаблон таблиці, призначити поля відповідним стовпцям за допомогою тегів HTML:

```
<asp:Repeater id="continent" runat="server">
  <HeaderTemplate>
      <table border="3" cellpadding="5">
          <tr>
              <th>
                  Code</th>
              <th>
                  Name</th>
          </tr>
  </HeaderTemplate>
  <ItemTemplate>
     <tr>
       <td>
         <%#
DataBinder.Eval(Container.DataItem, "id") %>
       </td>
       <td>
         <%#
DataBinder.Eval(Container.DataItem, "name")%>
```

```
            </td>
          </tr>
        </ItemTemplate>
        <FooterTemplate>
            </table>
        </FooterTemplate>
    </asp:Repeater>
```

Тег <HeaderTemplate> слугує для опису формату таблиці виведення даних з елемента asp:Repeater.

Для кожного запису даних використовується тег <ItemTemplate>. Для візуалізації значень даних шаблон заповнюється кодом типу

<%# DataBinder.Eval(Container.DataItem, "field") %>

Тег <FooterTemplate> закриває блок виведення елемента asp:Repeater.

Модифікувати попередні коди з внесенням кольорів та розділювачів. <AlternatingItemTemplate>. Елемент asp:Repeater має можливість увімкнути тег <SeparatorTemplate> для опису розділювачів між записами.

Крім System.Web.UI.WebControls.Repeater, ASP.NET містить інші елементи управління для візуалізації та модифікації даних. Наприклад, елементи

System.Web.UI.WebControls.DataList,
System.Web.UI.WebControls.DataGrid

використовуються для подання списків, таблиць і мають широке застосування для web-програм.

Елемент asp:DataList за своїми властивостями подібний до asp:Repeater і за замовчуванням містить таблицю для даних і використовується під час написання програм у Visual Studio .NET. Тегами заповнюються властивості для візуалізації даних у потрібній формі (зокрема стилям CSS).

Об'єкт asp:DataGrid є складнішим, але допомагає візуально проектувати програму.

8.6. Програмування в ASP.NET MVC

Нові версії ASP.NET дають змогу під час програмування розділити функціональний код основної програми на три частини: керування введенням, виведенням, сценарієм на сторінці тощо (C – controller); модель опрацювання даних (M – model); представленням даних на сторінці (V – view). Схему взаємодії розробленої програми з клієнтом у цій технології розроблення подано на рис. 8.7.

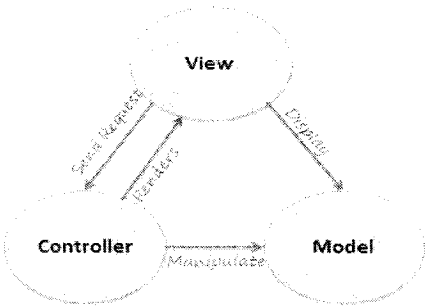


Рис. 8.7. Зв'язки у в програмі ASP.Net MVC між компонентами

Visual Studio пропонує розробнику в каталозі проекту бібліотеки шаблонів у відповідних назвах директорій (рис. 8.8).

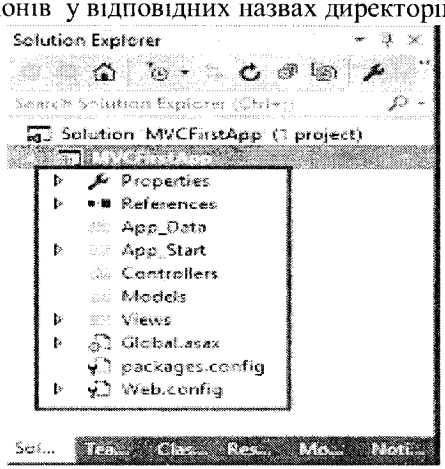


Рис. 8.8. Склад файлів та каталогів проекту ASP.Net MVC

Розробник, знаючи про зміст наповнення всіх інших файлів та директорій, основну увагу звертає на наповнення кодом класів трьох каталогів: Controllers, Models та Views.

Запит від користувача приходить до контролера. Отже, перше, що він повинен зробити – це дати команду на виведення сторінки для пересилання її клієнту. Крім того, контролер приймає запити від представлення на передачу команди для опрацювання даних у моделі. З погляду MVC для функціонування програми контролер – це керування сценарієм у межах проекту. Наведемо основні завдання контролера:

1) маршрутизатор MVC надсилає запит відповідному контролеру та методу Action на основі URL-адреси та налаштованих маршрутів. Контролер обробляє вхідні URL-адреси запитів;

2) усі публічні методи в класі Controller називаються методами Action;

3) клас контролера (Controller) повинен успадковувати клас System.Web.Mvc.Controller;

4) ім'я класу контролера повинно закінчуватися словом "Controller".

5) новий контролер можна створити за допомогою різних шаблонів. Можна створити власний шаблон.

6) actionResult – це базовий клас всього типу результату, який повертається з методу Action.

7) клас базового контролера містить методи, які повертають відповідний тип результату, наприклад: View(), Content(), File(), JavaScript () тощо.

8) метод дії може містити параметри типу Nullable.

Розглянемо простий приклад контролера, що має два методи: перший (виконується під час завантаження сторінки, назва Index) повертає представлення сторінки та метод внесення змін у список студентів (Update).

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.IO;
```

```
using System.Web.Mvc;
using ASP.NET_MVC_Application.Models;
using Newtonsoft.Json;

namespace ASP.NET_MVC_Application.Controllers
{
    public class StudentController: Controller
    {
        // GET: Student
        public ActionResult Index()
        {
            // Load the data for the student from Model
            var students = Student.GetStudents();
            // Return the view
            return View(students);
        }

    public ActionResult Update(int id)
      {
            var name = Request.Form["name"];
            var address = Request.Form["address"];
           // Get all of the students
            var students = Student.GetStudents();

            foreach (Student student in students)
            {
                // Find the student
                if (student.ID == id)
                {
                    // update his properties and save
it
                    student.Name = name;
                    student.Address = address;
                    // Break through the loop
                    break;
                }
            }
            // Update the students in the disk

System.IO.File.WriteAllText(Student.StudentFile,
JsonConvert.SerializeObject(students));
```

```
            // Add the details to the View
        Response.Redirect("~/Student/Index?Message=Student_U
        pdated");
                }
            }
```

Як ми бачимо з коду двох методів контролера, вони повертають представлення сторінки, вони мають доступ до полів сторінки (як і до всіх решти елементів сторінки), вони можуть записувати дані на диск тощо.

MVC містить різні класи результатів, які можна повернути з методів дії. Класи результатів є різних типів, таких як html, file, string, json, javascript тощо. У наведеній нижче таблиці перераховані всі класи результатів, наявні в ASP.NET MVC.

Result Class	Description
ViewResult	Представляє HTML і розмітку.
EmptyResult	Представляє відсутність відповіді.
ContentResult	Представляє рядок літерал
FileContentResult/ FilePathResult/ FileStreamResult	Представляє вміст файла
JavaScriptResult	JavaScript script.
JsonResult	JSON
RedirectResult	Представляє перенапрямлення до нової URL-адреси
RedirectToRouteResult	Представляти ще одну дію того чи іншого контролера
PartialViewResult	Повертає HTML з Partial view Returns HTML from Partial view
HttpUnauthorizedResult	Повертає статус HTTP 403Returns HTTP 403 status

Тепер розглянемо формування представлення (View) .. Зазначимо, що:

1) view – це інтерфейс користувача, який відображає дані та обробляє взаємодію користувачів;

2) папка переглядів містить окрему папку для кожного контролера;

3) ASP.NET MVC підтримує двигун перегляду Razor, крім традиційного движка .aspx;

4) файли перегляду View мають розширення .cshtml (.vbhtml)

Під час завантаження сторінка в цьому проекті має такий вигляд

```
@model IEnumerable<ASPNET_MVC_Application.Models.Student>

@{
   ViewBag.Title = "All Students";
   Layout = "~/Views/Shared/_Layout.cshtml";
}

<h3>table shows the Students detail</h3>
<p>You will find their details and links to other actions
operated over the student objects. </p>
<table class="students-table">
   <tr>
   <th>ID</th>
   <th>Name</th>
   <th>Address</th>
  </tr>

 @if(Model != null && Model.Count() > 0)
 {
   foreach (var student in Model)
   {
     // Create the list of these students
     <tr>
       <td>@student.ID</td>
       <td>@student.Name</td>
       <td>@student.Address</td>
        <td>
          <a
href="~/Student/Update/@student.ID">Update</a>
          </td>
     </tr>
   }
 }
      </table>.
```

Під час формування представлення View використовують мову Razor для ввімкнення даних та програмного коду. Синтаксис мови дуже простий і має такі пункти:

1) використовує @, щоб написати код на стороні сервера.

2) блок коду на стороні сервера починається з @ (* code *)

3) використовує @: або <text> </ <text>, щоб відобразити текст із коду блоку;

4) якщо умова починається з @if {};

5) для циклу починається з @for;

6) @model дає змогу використовувати об'єкт моделі в будь-якому місці View.

Як елементи управління використовують так звані HtmlHelpers. Основні з них наведено в таблиці

HtmlHelper	Типізовані HtmlHelpers	Html Control
Html.ActionLink		Anchor link
Html.TextBox	Html.TextBoxFor	Textbox
Html.TextArea	Html.TextAreaFor	TextArea
Html.CheckBox	Html.CheckBoxFor	Checkbox
Html.RadioButton	Html.RadioButtonFor	Radio button
Html.DropDownList	Html.DropDownListFor	Dropdown, combobox
Html.ListBox	Html.ListBoxFor	multi-select list box
Html.Hidden	Html.HiddenFor	Hidden field
Password	Html.PasswordFor	Password textbox
Html.Display	Html.DisplayFor	Html text
Html.Label	Html.LabelFor	Label
Html.Editor	Html.EditorFor	Генерування Html controls на основі даних

Приклад з елементами управління наведено нижче:

```
@model
IEnumerable<MVC_BasicTutorials.Models.Student>

@{
```

```
    ViewBag.Title = "Index";
    Layout = "~/Views/Shared/_Layout.cshtml";
}
<h2>Index</h2>
<p>
    @Html.ActionLink("Create New", "Create")
</p>
<table class="table">
    <tr>
        <th>
            @Html.DisplayNameFor(model =>
model.StudentName)
        </th>
        <th>
            @Html.DisplayNameFor(model =>
model.Age)
        </th>
        <th></th>
    </tr>

@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem =>
item.StudentName)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Age)
        </td>
        <td>
            @Html.ActionLink("Edit", "Edit", new {
id=item.StudentId }) |
            @Html.ActionLink("Details", "Details",
new { id=item.StudentId  }) |
            @Html.ActionLink("Delete", "Delete",
new { id = item.StudentId })
        </td>
```

```
    </tr>
}
</table>
```

Останньою компонентою є код Model, у якому дані опрацьовуються, щоб бути поданими для представлення View за допомогою контролера Controller.

```
using System;
using System.IO;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using Newtonsoft.Json;
namespace ASP.NET_MVC_Application.Models
{
   public class Student
   {
     public static string StudentFile =
HttpContext.Current.Server.MapPath("~/App_Data/Students.j
son");
     public int ID { get; set; }
     public string Name { get; set; }
     public string Address { get; set; }
     public static List<Student> GetStudents()
     {
        List<Student> students = new List<Student>();
        if (File.Exists(StudentFile))
        {
           // File exists..
           string content =
File.ReadAllText(StudentFile);
           // Deserialize the objects
           students =
JsonConvert.DeserializeObject<List<Student>>(content);
           // Returns the students, either empty list or
containing the Student(s).
           return students;
        }
        else
        {
           // Create the file
```

```
            File.Create(StudentFile).Close();
            // Write data to it; [] means an array,
            // List<Student> would throw error if [] is
not wrapping text
            File.WriteAllText(StudentFile, "[]");
           // Re run the function
            GetStudents();
        }
        return students;
      }
   }

    }
```

Метод цього класу переносить атрибути студентів з Json-файла у список студентів. Список студентів за допомогою контролера (Controller) переноситься у представлення сторінки View (циклом розноситься по рядках таблиці).

Важливою особливістю ASP.Net MVC є наявність двигуна маршрутизації:

1) маршрутизація відіграє важливу роль у межах MVC. Маршрутизація відображає URL-адресу до фізичного файла або класу (клас контролера в MVC);

2) маршрут містить шаблон URL-адреси та інформацію про обробника. Шаблон URL починається після імені домена;

3) маршрути можна налаштувати в класі RouteConfig. Водночас можна налаштувати декілька користувацьких маршрутів;

4) на маршрутних обмеженнях застосовуються обмеження на значення параметрів;

5) маршрут потрібно зареєструвати у події Application_Start у файлі Global.ascx.cs.

Процедуру формування маршрутів показано на рис. 8.9. А сам шлях формується у класі конфігурації шляху:

```
public class RouteConfig
{
    public static void
RegisterRoutes(RouteCollection routes)
    {
```

```
routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
        routes.MapRoute(
            name: "Student",
            url: "students/{id}",
            defaults: new { controller = "Student",
action = "Index"}
        );
        routes.MapRoute(
            name: "Default",
            url: "{controller}/{action}/{id}",
            defaults: new { controller = "Home",
action = "Index", id = UrlParameter.Optional }
        );
    }
}
```

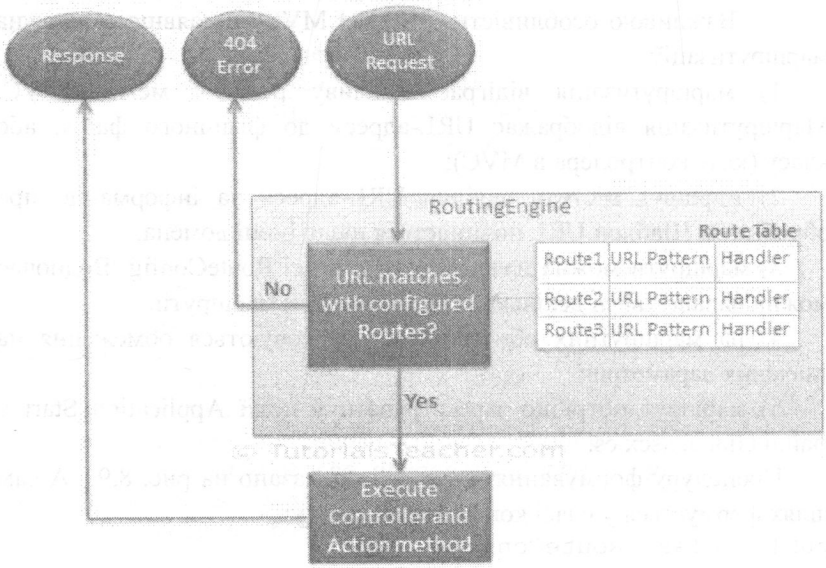


Рис. 8.9. Робота маршрутизатора ASP.Net MVC

Контрольні запитання та вправи

1. Пояснити, які змінні та об'єкти використовують у ASP.NET-сторінках?
2. Навести приклади HTML-елементів управління.
3. Навести приклади web-елементів управління.
4. Які атрибути елементів управління і де вони розміщуються?
5. Які класи успадковуються під час створення ASP.NET-сторінок?
6. Які типи файлів використовують в ASP.NET?
7. Навести правила створення шаблонів.
8. Пояснити різницю між файлами .aspx, .ascx, .ashx?
9. Які типи елементів для зв'язування даних?
10. Які класи для керування сесіями в ASP.NET?
11. Які складові компоненти засобів аутентифікації? Яка їхня функція?
12. Які функції покладаються на конфігураційний файл сервера?
13. Яка складові MVC ASP.NET?
14. Чим керує controller?
15. Які є головні об'єкти в механізмі MVC ASP.NET?
16. Які функції класів у model?

Глава 9
ДОСТУП ДО БАЗ ДАНИХ З ADO.NET

9.1. Інтерфейси доступу до баз даних

У стандартах Microsoft для роботи з даними можна зазначити:

- ODBC (Open Database Connectivity) – стандартне API для різних СУБД на основі SQL для доступу до даних;
- DAO (Data Access Objects) – інтерфейс для програмування баз даних JET/ISAM, використовує технології OLE та ActiveX;
- RDO (Remote Data Objects) – базовані на ODBC, орієнтовані на програмування в архітектурі клієнт-сервер;
- OLE DB – надбудова над COM, що уможливлює доступ до реляційних та нереляційних баз даних. Можуть використовуватись драйвери до ODBC та пропонується інтерфейс на рівні C++;
- ADO (ActiveX Data Objects) – об'єктно-орієнтований інтерфейс та модель програмування для OLE DB (рис. 9.1), доступних з різних мов програмування (наприклад, Visual Basic).

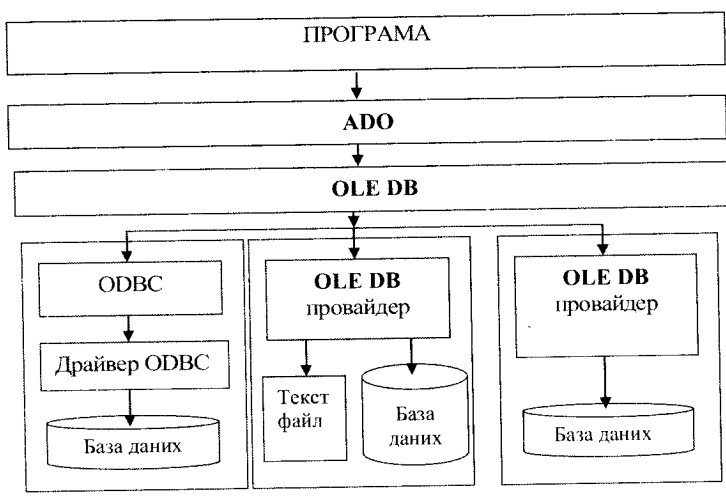


Рис. 9.1. Доступ до баз даних через ADO

ADO використовується в архітектурі клієнт-сервер для реляційних баз даних (не ієрархічних як у випадку XML). ADO не проектувалась як архітектура для розподілених обчислень.

Складовою платформи .NET для забезпечення доступу до СУБД є засоби ADO .NET-зібрання класів, інтерфейсів, структур та типів, що дають змогу реалізувати доступ до даних. Її можна розглядати як покращену версію ADO.

ADO .NET об'єднує можливості ADO та OLE DB у єдиній системі управління послугами доступу до даних (рис. 9.2). Кожна послуга містить класи для реалізації інтерфейсу до різних джерел даних. Наприклад, ADO Managed Provider (до джерел даних OLE DB), SQL Server Managed Provider (доступ до DBMS Microsoft), Exchange Managed Provider (дані з Microsoft Exchange) тощо. ADO .NET підтримує використання документів у XML форматі. Це спрощує доступ до даних за протоколом HTTP.

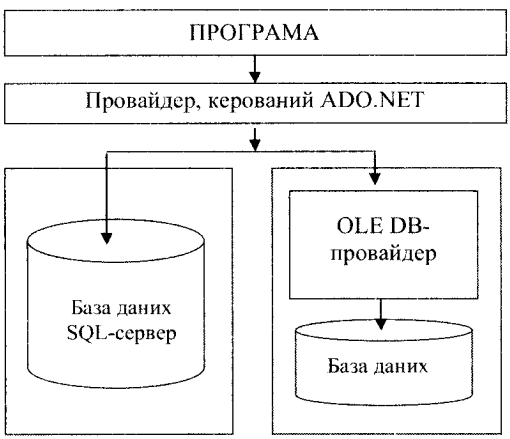


Рис. 9.2. Доступ до баз даних через ADO.NET

9.2. Архітектура ADO.NET

ADO.NET є технологією доступу до баз даних у платформі .NET (Dot Net), яка ґрунтується на технології Microsoft ActiveX®

Data Objects (ADO). ADO є незалежною від мови програмування технологією об'єктних моделей. ADO.NET є інтегральною частиною засобів .NET Compact Framework і підтримує доступ до реляційних баз даних, до XML документів та даних програм. На її основі можна розробити як клієнтські, так і серверні частини проектів широкого спектра задач.

Основні класи та інтерфейси ADO .NET наведено в цій таблиці:

Об'єкти	Базові класи	Інтерфейси	Призначення
`Connection`	`DbConnection`	`IDbConnection`	Відкриття та закриття з'єднання до джерел даних
`Command`	`DbCommand`	`IDbCommand`	Команди SQL запитів та назви процедур
`DataReader`	`DbDataReader`	`IDataReader`	Доступ до читання (тільки) даних
`DataAdapter`	`DbDataAdapter`	`IDataAdapter`	Посередник між користувачем та місцем зберігання даних. Містить 4 об'єкти команд (`select`, `insert`, `update`, `delete`)
`Parameter`	`DbParameter`	`IDataParameter`	Параметри для запитів
`Transaction`	`DbTransaction`	`IDbTransaction`	Здійснення транзакцій

ADO .NET визначає низку інтерфейсів, які пропонують базову функційність доступу до джерел даних. Реалізація інтерфейсів на

основі методів і властивостей створює конкретні програмні засоби доступу до джерел даних:

Інтерфейс `IDbConnection` – встановлює сесію з джерелом даних. Дає змогу відкривати та закривати з'єднання, починати та завершувати трансакції (методи `Commit` та `Rollback`. Класи `SqlDbConnection` та `OleDbConnection` реалізують інтерфейс `IDbConnection`.

Інтерфейс `IDbCommand` представляє команди, що надсилаються до джерела даних (як правило, в SQL). Класи `SqlDbCommand` та `OleDbCommand` реалізують інтерфейс `IDbCommand`.

Інтерфейс `IDataReader` описує доступ до джерел даних для читання. Класи `SqlDataReader` та `OleDbDataReader` реалізують інтерфейс `IDataReader`. Метод `ExecuteReader` повертає групу **стовців**, а метод `ExecuteScalar` повертає єдине значення, метод `ExecuteNonQuery` не повертає нічого (знищення та відновлення).

Інтерфейс `IDataAdapter` реалізований у класах `OleDbDataAdapter` та `SqlDataAdapter` використовується для побудови групи даних. З ними працюють у асинхронному режимі, коли не підтримується відкритим з'єднання з базою даних. Тут можна реалізувати операції вибірки (`select`), вставки (`insert`), оновлення (`update`) та знищення (`delete`).

В ADO.NET межі транзакцій вказують програмістом. Об'єкти класу `Connection` мають метод `BeginTransaction`, який повертає трансакцію (об'єкт типу `Transaction`). Трансакція виконується під час виклику методу `Commit` та переривається методом `Rollback` об'єкта `Transaction` поверненого методом `BeginTransaction`.

ADO .NET має дві форми доступу до даних:

Доступ **без з'єднання**: Програма виконує запит і розміщує результати виконання запиту в об'єкті типу `DataSet`, щоб пізніше їх перенести в задані масиви тощо. У цьому разі мінімізується час підтримки відкритого з'єднання до бази даних.

Доступ **через з'єднання**: читання з безадресним курсором ("firehose cursors"). Програма здійснює запит і читає дані за допомогою об'єкта `DataReader`.

ADO.NET визначає об'єкти типу `DataSet` та `DataTable`, які оптимізують перенесення даних по мережах Інтранету та Інтернету (зокрема крізь firewalls). Технологія надає також традиційні об'єкти `Connection` і `Command`.

За допомогою ADO.NET та .NET Framework можна організувати доступ до таких джерел даних:

- SQL Server (`System.Data.SqlClient`).
- OLEDB (`System.Data.OleDb`).
- ODBC (`System.Data.Odbc`).
- Oracle (`System.Data.OracleClient`).

Двома основними компонентами в `ADO.NET` є класи: `DataSet` та .NET Framework `Data Provider` (рис. 9.3).

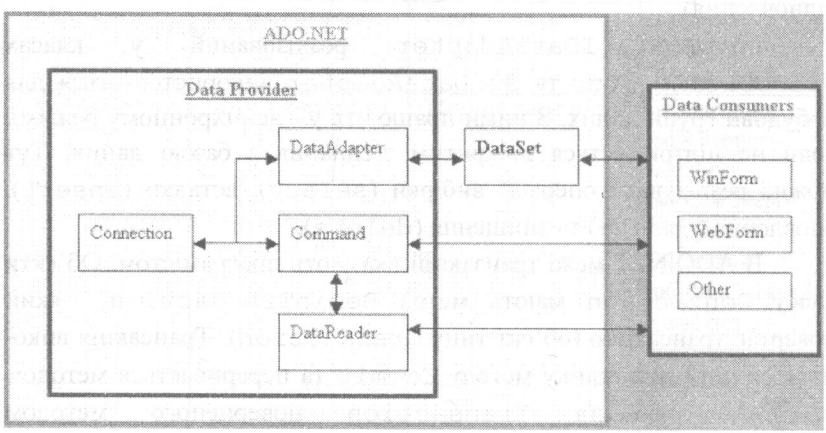


Рис. 9.3. Структура засобів ADO.NET

Компонента джерела даних (`Data Provider`) за призначенням подібна до диспетчерів ODBC чи JDBC та містить такі об'єкти:

- `Connection` (`SqlConnection`, `OleDbConnection`, `OdbcConnection`, `OracleConnection`);

• Command (SqlCommand, OleDbCommand, OdbcCommand, OracleCommand) – SQL запити для роботи з даними баз даних: select, insert, delete та update;

• DataReader (SqlDataReader, OleDbDataReader, OdbcDataReader, OracleDataReader) – для читання даних великих об'ємів, що не розміщуються в пам'яті, DataAdapter використовує його для з'єднання з базою даних;

• DataAdapter (SqlDataAdapter, OleDbDataAdapter, OdbcDataAdapter, OracleDataAdapter) – для з'єднання з базою даних контейнера даних .

Об'єкт DataSet – компонента, що містить результати запиту (містить низку таблиць). За характером подібний до роз'єднаного кешу даних – місця в пам'яті. Складовими якого є об'єкти DataTables та DataRelations (форма для встановлення зв'язків між таблицями) – зберігають результати виконання команд. DataAdapter – слугує зв'язком між контейнером даних DataSet та джерелом (базою) даних Data Provider.

Етапи функціонування ADO.NET є такі:

1. Створення об'єкта Connection з вказуванням для нього параметрів з'єднання.
2. Створення DataAdapter.
3. Створення об'єкта Command, асоційованого з DataAdapter.
4. Відкриття DataSet для розміщення даних.
5. Відкриття з'єднання.
6. Наповнення DataSet даними за допомогою DataAdapter.
7. Закриття з'єднання.
8. Опрацювання даних, що зберігаються в DataSet.

9.3. Операції доступу до даних

Залежно від СУБД, що є джерелом даних, до програми залучаються такі бібліотеки класів (простори назв):

```
// СУБД MS Access
using System.Data.OleDb;
```

```
// СУБД MS SQL Server
using System.Data.SqlClient;
```

Для під'єднання до відповідної бази даних конструктор об'єкта Connection вимагає параметрів, які вказують на комп'ютер джерела даних (сервер, IP-адреса тощо), шлях до каталога з базою даних, спосіб забезпечення безпеки передачі (SSPI), ім'я користувача, зареєстрованого в СУБД, а також відповідний пароль.

Наприклад, необхідно вказати назву бази даних СУБД MS Access для під'єднання до "*PersonDatabase.mdb*" разом з шляхом до файлів:

```
public string
    conString=@"Provider=Microsoft.Jet.OLEDB.4.0;" +
       @" DataSource=..\\..\\PersonDatabase.mdb";
```

Конструктором з параметром – рядком адреси створюється об'єкт для відкриття з'єднання з СУБД:

```
. . . . . .
OleDbConnection con = new
OleDbConnection(conString);
// відкрити з'єднання з СУБД
 con.Open();
. . . . . .
```

Відкриття з'єдання займає певні ресурси сервера, які він розділяє між всіма користувачами, що мають доступ до його даних. Тому під час написання вищезазначених операцій треба користуватись правилами:

- Відкривати з'єднання тільки за необхідності, тобто перед безпосередніми операціями маніпуляцій з даними;
- Закривати з'єднання відразу після завершення операцій доступу.

Наведемо фрагмент програми доступу до MySQL-сервера, що має команди створення об'єкту, відкриття з'єднання, формування та виконання SQL-запиту, закриття з'єднання та опрацювання виняткових ситуацій:

```
. . . . . .
private SqlConnection con = null;
private string constr ="Integrated
Security=SSPI;" +
```

```
        "Initial Catalog=Northwind;" +
        "Data Source=SONY\\MYSQLSERVER;";
          private void fnGetConnection()
         {
           try
           {
                 // під'єднання
                 con = new SqlConnection(constr);
                 con.Open();
                 // формування та виконання SQL-запиту
            string sqlInsert = "INSERT INTO
                Department(DepartmentName) VALUES
             (@DepartmentName)";
           SqlCommand insertCommand = new
    SqlCommand(sqlInsert, con);
           SqlParameter                    param          =
    insertCommand.Parameters.Add                          (new
    SqlParameter("@DepartmentName",    SqlDbType.VarChar,
    100));
            param.Value = "Arsen";
        }
        catch (Exception ex)
                {
                 MessageBox.Show("Error in connection :
"+ex.Message);
                }
        finally
        {
          // закрити з'єднання
          if (con != null)
            con.Close();
        } // кінець блока finally
    }
  }
```

Під час розміщення даних у пам'яті і роботі з ними (внесення змін, знищення чи поновлення) без з'єднання методи `Update (DataAdapter)`, `AcceptChanges (DataSet)` та `RejectChanges` слугують для фіксації чи заборони проведення змін у базах даних.

9.4. Складові класу DataSet

Клас DataSet забезпечує доступ до даних без з'єднання. Він містить копію прочитаних даних, яка завантажуються у системі клієнта.

Об'єкт DataSet складається з двох об'єктів (рис. 9.4):

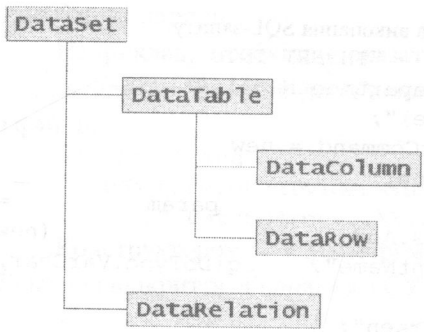


Рис. 9.4. Схема класу DataSet

- DataTable – містить null або багато об'єктів типу DataTable, Columns, Rows, Constraints;
- DataRelation – містить null або багато об'єктів типу DataRelation, між якими встановлені відношення батьки/ діти для пари об'єктів типу DataTable.

Об'єкт DataSet інкапсулює таблиці і підтримує відношення між ними. Зміст DataSet можна серіалізувати (представити послідовністю символів) у форматі XML. У ньому можливо проводити динамічні зміни даних та метаданих.

Клас DataTable представляє таблиці у пам'яті. Схема таблиці описується через стовпці Columns. Цілісність даних забезпечується об'єктами забезпечення обмежень (Constraint), які містять події на реакції операцій над таблицею.

Клас DataColumn призначений для опису стовпців на основі властивості DataType і містить обмеження Constraints та відношення Relations. Володіє властивостями AllowNull, Unique та ReadOnly.

Клас DataRow описує дані рядка таблиці (розміщені в колекції Rows об'єкта DataTable.

Клас DataRelation пов'язує DataTable з DataColumn та призначений для підтримки обмежень на цілісність. Для доступу до даних, пов'язаних з конкретним записом, треба виконати метод GetChildRecords з відповідного стовпця DataRow.

Розглянемо фрагменти коду для створення таблиць, стовпців та первинних ключів у базі даних, що на рис. 9.5:

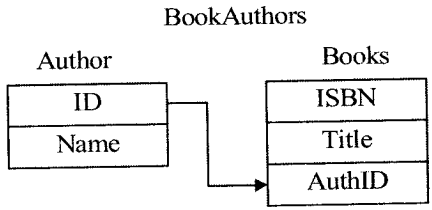


Рис. 9.5. Приклад бази даних

```
DataTable authors = new DataTable("Author");
DataTable books = new DataTable("Book");
DataColumn id = authors.Columns.Add("ID", typeof(Int32));
DataColumn name = new
authors.Columns.Add("Name",typeof(String));
authors.PrimaryKey = new DataColumn[] {id};
id.AutoIncrement = true;
DataColumn isbn  = books.Columns.Add("ISBN",
typeof(String));
DataColumn title  = books.Columns.Add("Title",
typeof(String));
DataColumn authid =
books.Columns.Add("AuthID",typeof(Int32));
books.PrimaryKey = new DataColumn[] {isbn};
DataColumn[] secondkey = new DataColumn[] {authid};
DataRelation bookauth = new DataRelation("BookAuthors",
        authors.PrimaryKey, secondkey);
```

На основі створених об'єктів таблиць формується об'єкт DataSet:

```
DataSet dataset = new DataSet();
dataset.DataSetName = "BookAuthors";
dataset.Tables.Add (authors);
dataset.Tables.Add (books);
dataset.Relations.Add (bookauth);
```

Наступним фрагментом вносяться дані у відповідні поля таблиці:

```
DataRow shkspr = authors.NewRow();
shkspr["Name"] = "William Shakespeare";
authors.Rows.Add(shkspr);

DataRow row = books.NewRow();
row["AuthID"] = shkspr["ID"];
row["ISBN"] = "1000-XYZ";
row["Title"] = "MacBeth";
books.Rows.Add(row);
// Виконати внесені зміни
dataset.AcceptChanges();
```

Можливі два типи класу `DataSet`: типізовані та нетипізовані. Перші утворюються з базового класу `DataSet` та для генерації нового класу використовують інформацію з XML схемних файлів (*.xsd* file). Ця інформація (таблиці, стовпці тощо) генерується та компілюється в новий клас `DataSet` як ряд об'єктів та властивостей першого класу. Типізовані класи підтримуються засобами IntelliSense у редакторі середовища Visual Studio.

Наступний приклад ілюструє доступ до першого рядка стовпця `CustomerID` у таблиці `Customers`:

```
DataSet dset = new DataSet();
string str;
str=dset.Customers[0].CustomerID;
```

Для створення типізованого `DataSet` необхідно

- Викликати командний prompt (*cmd*) на місці розташування схемного XSD-файла;
- Виконати утиліту *XSD.EXE* для створення типізованого класу `DataSet`:

```
xsd.exe /d /l:cs mydataset.xsd /n:mynamespace
```

Параметри вказують

`/d` : створити `DataSet`.

`/l:cs` – встановити мову `C#`.

`/n:mynamespace` – простір назв "`mynamespace`" для класу.

Результатом роботи *XSD.EXE* з даними аргументами буде файл *.cs* класу (*mydataset.cs*).

Для компіляції класу треба застосувати *csc.exe* :

```
csc.exe /t:library mydataset.cs /r:System.dll
/r:System.Data.dll
  /r:System.XML.dll /out:bin/mydataset.dll
/t:library
```

Компіляція створить бібліотечну компоненті типу DLL, а параметри вказують:

- /r: – на збірку, на яку формується посилання;
- /out: – на місце зберігання – підкаталог *bin* поточного каталога.

Нетипізований клас **DataSet** не визначається схемою, а формується програмно на основі таблиць та їх елементів.

Для нетипізованого типу DataSe*t* наведений вище код еквівалентний такому:

```
DataSet dset = new DataSet();
 string str;
 str=(string)dset.Tables["Customers"].Row[0].["Cus
tomerID"];
```

Об'єкт DataSet є контейнером, який можна наповнювати даними різними способами:

1. За допомогою об'єкта DataAdapter та його методу Fill :

```
string strCon = @"Data Source=SONY\MYSQLSERVER;" +
  "Initial              Catalog=Northwind;Integrated
Security=SSPI";
  string strSql="select * from customers";
  SqlConnection con=new SqlConnection(strCon);
  con.Open();
// створення об'єкта DataAdapter
  SqlDataAdapter dadapter=new SqlDataAdapter();
  dadapter.SelectCommand=new
SqlCommand(strSql,con);
  DataSet dset=new DataSet();
  dadapter.Fill(dset);
  con.Close();
// прив'язування даних та таблиці
  this.dataGrid1.DataSource=dset;
```

2. Створення програмних об'єктів DataTable, DataColumn та DataRow:

```
DataSet dset;
DataTable dtbl;
DataRow drow;
// створити рядок
drow=dtbl.NewRow();
// за назвою поля або індексом занести дані
drow["LastName"]="Altindag";
drow[1]="Altindag";
// додати новий рядок методом Add до DataRow
dtbl.Rows.Add(drow);
// або додати рядок методом Add через масив типу Object
dtbl.Rows.Add(new object[] {1, "Altindag"});
```

3. Перенести XML-документ чи потік у `DataSet`:

З бази даних `Pubs` SQL-запитом отримуються дані таблиці `Authors` у вигляді XML-документа і створюється об'єкт типу `SqlCommand`. Після створення об'єкта типу `DataSet`, метод `ExecuteXmlReader` передає дані об'єкта `DataSet` методом ReadXml.

Далі встановлюється властивість `DocumentContent` методом `GetXml` об'єкта `DataSet`. Для виведення документа використовується документ XSL Transformation (файл *authors.xsl* введений у проект):

```
protected System.Web.UI.WebControls.Xml XmlDisplay;
    string strCon = @"Data Source=SONY\MYSQLSERVER;" +
    "Initial Catalog=pubs;Integrated Security=SSPI";
    SqlConnection con=new SqlConnection(strCon);
    con.Open();
    try
    {
      string strSql="select * from FROM authors FOR XML
    AUTO, XMLDATA";
      SqlCommand cmd=new SqlCommand(strSql, con);
      DataSet dset=new DataSet();

dset.ReadXml(cmd.ExecuteXmlReader(),XmlReadMode.Fragme
nt);
  XmlDisplay.DocumentContent = dset.GetXml();
} finally {
  con.Close();
}
```

4. Об'єднанням (копіюванням) змісту іншого об'єкта DataSet методом Merge: якщо два об'єкти DataSet мають подібні схеми, то можна в один з них скопіювати інший, у якому, наприклад, були проведені певні зміни:

```
dataset1.Merge(dataset2);
```

9.5. Методи класу DataAdapter

Об'єкт типу DataAdapter є подібний до мосту, що з'єднує базу даних і об'єкт Connection з керованим засобами **ADO.NET** об'єктом DataSet за допомогою команд запитів. Командами вказується, які дані записати в DataSet, а які прочитати. Крім команд, це можна також здійснити раніше сформованими процедурами.

Назви об'єктів DataAdapter залежать від типу СУБД, до яких здійснюється доступ, наприклад, OLEDbDataAdapter (MS ACCESS) чи SQLDataAdapter (SQL SERVER)

Об'єкт DataAdapter має чотири властивості, що дають змогу контролювати проведені зміни на сервері: SelectCommand, UpdateCommand, InsertCommand, DeleteCommand. Ці властивості переносяться до об'єкта Command, що використовується в операціях маніпуляції з даними.

Головними методами об'єкта DataAdapter є: Fill – наповнює даними DataSet; FillSchema – запит на інформацію про схему, що модифікується; Update – змінити базу даних. Об'єкт DataAdapter має властивості DeleteCommand, InsertCommand і UpdateCommand.

Розглянемо приклад блока команд з'єднання, запиту та читання:

```
try {
 // Створити екземпляр
  con =new OleDbConnection(conString);
 dAdapter=new  OleDbDataAdapter("select  *  from
PersonTable", con);
 // Створити екземпляр
        dSet=new DataSet();
 // оновлення рядків у DataSet
```

```
        dAdapter.Fill(dSet,"PersonTable");
      }catch(Exception ex) {
         MessageBox.Show("Error : "+ex.Message);
        // з'єднання не відбулось
        return false;
      }// кінець блока try-catch
      // з'єднання відбулось
      . . . . . .
```

Для закріплення матеріалу наведемо приклад повного тексту програми з відповідним інтерфейсом користувача, що запитує певні дані та виводить їх у таблицю на екран:

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Windows.Forms;

class DisconnectedDataForm : Form
{
   private SqlConnection  conn;
   private SqlDataAdapter daCustomers;
   private DataSet  dsCustomers;
   private DataGrid dgCustomers;
   private const string TableName = "Customers";
   // Ініціалізувати форму з DataGrid та Button
   public DisconnectedDataForm()
   {
            // наповнити dataset
            InitData();
            // створити об'єкт datagrid
            dgCustomers = new DataGrid();
            dgCustomers.Location = new Point(5, 5);
            dgCustomers.Size = new Size(
            this.ClientRectangle.Size.Width - 10,
            this.ClientRectangle.Height - 50);
            dgCustomers.DataSource = dsCustomers;
            dgCustomers.DataMember = TableName;
            // створити кнопку з написом update
```

```
                Button btnUpdate = new Button();
                btnUpdate.Text = "Update";
                btnUpdate.Location = new Point(
                  this.ClientRectangle.Width/2 -
btnUpdate.Width/2,
                  this.ClientRectangle.Height -
(btnUpdate.Height+10));
                btnUpdate.Click += new
EventHandler(btnUpdateClicked);
                // перевірити елементи управління на формі
        Controls.AddRange(new Control[] { dgCustomers,
btnUpdate });
        }
        // створити об'єкти ADO.NET
        public void InitData()
        {
                // встановити з'єднання
                conn = new SqlConnection(

        "Server=(local);DataBase=Northwind;Integrated
                        Security=SSPI");

        // 1. утворити екземпляр DataSet
                dsCustomers = new DataSet();

        // 2. задати об'єкту SqlDataAdapter команду select та
з'єднання
              daCustomers = new SqlDataAdapter(
              "select CustomerID, CompanyName from
Customers", conn);

        // 3. Наповнити командами insert, update, and delete
commands
                SqlCommandBuilder cmdBldr = new
                        SqlCommandBuilder(daCustomers);

        // 4. наповнити dataset
                daCustomers.Fill(dsCustomers,
TableName);
        }
```

```
        // Реакція на кнопку Update
        public void btnUpdateClicked(object sender,
EventArgs e)
        {
                // запис змін у DataBase
                daCustomers.Update(dsCustomers, TableName);
        }

        // стартувати форму Windows
        static void Main()
        {
                Application.Run(new
DisconnectedDataForm());
        }
 }
```

Метод `InitData` містить методи, необхідні для створення об'єктів `SqlDataAdapter` та `DataSet`. Властивість `DataSource` об'єкта `DataGrid` встановлюється в конструкторі. У разі натискання кнопки `Update` викликається метод `Update` в обробнику подій `btnUpdateClicked` і зміни в даних здійснюються у базі даних.

9.6. Читання даних у ввімкненому стані

9.6.1. Доступ до даниха з клієнта

Клас `SqlDataReader` призначений для читання даних потоком. Дані читаються послідовно і для повторного використання їх необхідно зберігати. Екземпляр створюється методом `ExecuteReader()` класу `Command` (об'єкт `SqlCommand` створює посилання на з'єднання та SQL-команду):

```
. . . . . .
SqlDataReader rdr = cmd.ExecuteReader();
. . . . . .
```

`SqlDataReader` повертає дані в послідовному потоці як таблицю рядок за рядком. Щоб прочитати попередні рядки треба знову створювати екземпляр `SqlDataReader` і читати дані в

потоці знову. Для читання рядків необхідно організувати цикл while:

```
while (rdr.Read())
{
    // отримати результати кожного стовпчика
    string contact = (string)rdr["ContactName"];
    string company = (string)rdr["CompanyName"];
    string city    = (string)rdr["City"];

    // друк результатів
    Console.Write("{0,-25}", contact);
    Console.Write("{0,-20}", city);
    Console.Write("{0,-25}", company);
    Console.WriteLine();
}
```

Метод Read() повертає логічний тип true, якщо є ще записи для читання. Якщо прочитаний останній запис повернене значення приймає false.

Читання стовпців здійснюється за допомогою числа-індекса об'єкта rdr[0]або за назвою стовпця rdr["City"].Індексатор повертає типи об'єктів, які в прикладі перетворюються на рядки.

Як правило, оператори доступу розміщуються у блоці try. Після завершення читання об'єкт необхідно закрити методом Close():

```
try
{
        // оператори доступу до бази даних
}
finally
{
// закрити читача
        if (rdr != null)
        {
            rdr.Close();
        }

// закрити з'єднання
}
```

Наступний приклад ілюструє роботу читача SqlDataReader.

```
using System;
using System.Data;
using System.Data.SqlClient;

namespace DemoSqlDataReader
{
  class ReaderDemo
  {
          static void Main()
          {
                  ReaderDemo rd = new ReaderDemo();
                  rd.SimpleRead();
          }

          public void SimpleRead()
          {
                  // оголошення SqlDataReader
                  SqlDataReader rdr = null;

                  // створити об'єкт для з'єднання
                  SqlConnection conn = new
SqlConnection(
        "Data Source=(local);Initial
        Catalog=Northwind;Integrated
        Security=SSPI");
                  // створити об'єкт команд
                  SqlCommand cmd  = new SqlCommand(
                          "select * from
Customers", conn);
                  try
                  {
                  // відкрити з'єднання
                  conn.Open();
   // 1. отримати екземпляр SqlDataReader
                  rdr = cmd.ExecuteReader();
   // друк заголовків стовців
   Console.WriteLine(
"Contact Name              City
Company Name");
   Console.WriteLine(
"-------------              ------------       ------
------");
```

```
    // 2. друк запису (три стовці)
                        while (rdr.Read())
                        {
    // елемент – дані з кожного стовпця
                string contact =
(string)rdr["ContactName"];
                string company =
(string)rdr["CompanyName"];
                string city    = (string)rdr["City"];

                // друк елементів запису
                Console.Write("{0,-25}", contact);
                Console.Write("{0,-20}", city);
                Console.Write("{0,-25}", company);
                Console.WriteLine();
                }
    }
    finally
    {
    // 3. закрити читача
                if (rdr != null)
                {
                rdr.Close();
                }

    // закрити з'єднання
                if (conn != null)
                {
                conn.Close();
                                                }
                                        }
                                }
                }
    }
```

На завершення параграфа наведемо приклади деяких SQL-команд на заповнення бази даних у СУБД MS SQL-Server. Повний набір команд цього програмного продукту як і інших СУБД наведено в спеціалізованій літературі.

Запити до сервера баз даних MS SQL-Server мають інструкції мови запитів SQL: 1) на створення таблиці даних

```
CREATE TABLE [dbo].[Department] (
  [DepartmentID] [int] IDENTITY (1, 1) NOT NULL ,
  [DepartmentName] [varchar] (100),
  [CreationDate] [datetime] NULL
) ON [PRIMARY]
GO
```

2) на зміну таблиці даних

```
ALTER TABLE [dbo].[Department] WITH NOCHECK ADD
   CONSTRAINT [PK_Department] PRIMARY KEY CLUSTERED
   (
         [DepartmentID]
   )  ON [PRIMARY]
GO
```

3) на формування процедури заповнення полів відповідної таблиці

```
CREATE PROCEDURE dbo.CreateDepartment
@DepartmentName varchar(100),
AS
   INSERT   INTO   Department   (DepartmentName,
CreationDate)
   VALUES (@DepartmentName, GetDate())
   RETURN scope_identity()
GO
```

9.6.2. Доступ до даних з сервера

Для доступу до даних з сервера застосовуються ті самі бібліотеки класів та інтерфейсів, що використовуються для створення клієнтських застосувань. Проілюструємо це прикладом ASP.NET сторінки. На сторінці розташовано asp-елемент `Repeater (rp1)`, в поля якого методом `Page_Load` заносяться дані з таблиці `Dogs`. Атрибути доступу до бази даних знаходяться у конфігураційному файлі «testdsn». Таблиця переноситься потоком об'єктом типу `SqlDataReader`.

```
<%@ Page language=C# Debug=true%>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<html>
<script runat="server">
protected void Page_Load(Object src, EventArgs e)
```

```
{
  if (!IsPostBack)
  {
    string dsn =
ConfigurationSettings.AppSettings["testdsn"];
    SqlConnection conn = new SqlConnection(dsn);
    SqlCommand cmd = new SqlCommand("SELECT * FROM
Dogs", conn);
    try
    {
      conn.Open();
      SqlDataReader reader = cmd.ExecuteReader();
     rpl.DataSource = reader;
      DataBind();
    }
    catch (Exception ex)
    { Response.Write(ex.ToString()); }
    finally
    { conn.Dispose(); }
  }
}
</script>
<body>
<form runat=server>
<asp:Repeater id="rpl" runat=server>
  <ItemTemplate>
    <%# DataBinder.Eval(Container.DataItem, "Name")
%> is a
    <%# DataBinder.Eval(Container.DataItem, "Breed")
%>
    and looks like this:
    <asp:Image runat=server
         ImageUrl=<%#
DataBinder.Eval(Container.DataItem, "Image") %> />
  </ItemTemplate>
  <SeparatorTemplate>
    <hr/>
  </SeparatorTemplate>
</asp:Repeater>
</form>
</body>
</html>
```

Контрольні запитання та вправи

1. Що є база даних, а що таблиця даних?
2. Які функції покладені на драйвера та диспетчера СУБД?
3. Як здійснити доступ до СУБД різних виробників?
4. Які особливості доступу до баз даних у платформі .NET?
5. Мова SQL, її призначення та складові?
6. Як формується запит у мові програмування .NET?
7. Які об'єкти можна утворити методами класу `DataAdapter`?
8. Які об'єкти можна утворити методами класу `Command`?
9. Які методи класу `Connection` найчастіше використовуються програмістом?
10. Яким інструментарієм повинен володіти програміст для реалізації віддаленого доступу до баз даних .NET?
11. Як клас `DataSet` зберігає таблиці?
12. Які функції класу `DataReader`?
13. Які положення ASP.NET повинен знати програміст, щоб проектувати та програмувати виведення результатів з баз даних?
14. У чому полягає модульність програм доступу до баз даних?
15. Чому при доступі до баз даних необхідно використовувати блок `try`?
16. Як пов'язати об'єкти, отримані в результаті реалізації запитів зі сторінками, які формуються для відсилання до клієнта?
17. Написати фрагменти програм доступу до СУБД.
18. Формування запитів, отримання таблиць-результатів запитів.
19. Перенесення результатів у поля web-сторінки.
20. Скласти проект системи клієнт-сервер для таблиць даних.

Глава 10
JAVA-СЕРВЛЕТИ ТА JSP-ТЕХНОЛОГІЯ

10.1. Функції сервлетів

Інтерактивний інтерфейс користувача – це система для забезпечення взаємодії користувача і відповідної програми. Для технології WWW (World Wide Web) інтерактивний інтерфейс складається з послідовності HTML-документів і може бути класифікований як статичний або динамічний. У першому випадку документ залишається незмінним протягом його використання. У другому випадку джерелом інтерфейсу є HTML-документ, згенерований так званим cgi-модулем (від терміну CGI – Common Gate Interface). Отримання даних від клієнта до сервера, опрацювання їх на сервері та повернення відповіді до клієнта – всі ці завдання можливо вирішувати на основі зазначеного інтерфейсу. Серед них найцікавішими є: доступ до баз даних чи SQL-сервера, отримання інформації від периферійних пристроїв, створення клієнтських робочих місць тощо. Схему взаємодії клієнта і сервера через CGI-інтерфейс подано на рис. 10.1.

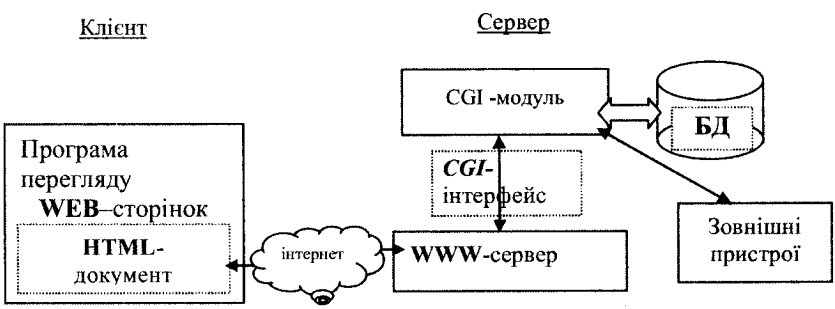


Рис. 10.1. Схема взаємодії клієнта і сервера через CGI

Спосіб взаємодії (обміну запитів та відповідей) між програмою-клієнтом і програмою-сервером у межах технології WWW

здійснюється за HTTP-протоколом. Щоб визначити мінімально необхідні операції cgi-модулів чи сервлетів під час читання чи формування потоків наведемо поля, які містять протоколи запитів та відповідей (з відповідними поясненнями):

Запит

GET http:/testserver/ HTTP/1.0 - метод, адреса та версія протоколу
Connection :Keep-Alive – опція на підтримку тримати з'єднання відкритим
Accept: image/x-xbitmap - типи даних (*Accept: */** - всі види)
Accept-Charset: iso-8859-1,8, utf-8 - кодування символів
Accept: encoding: gzip - кодування та стискування
Accept-language: en, fr - мови броузера
Host:testserver:80 - сервер, що обслуговує web-вузол
User-agent: Mozilla/1. 1N (Windows; 16bit) -ідентифікатор броузера

Відповідь сервера

GET http:/testserver/ .200 OK -код повернення
Date: Thursday, 12-Oct-95 02:51:33GMT - дата
Server: WinHttpd/1. 4a (Shareware Non-Commercial License)- сервер
Content-length: 2222 - довжина відповіді в байтах
Content-type: image/gif - типи даних
Last-modified: Monday, 20-Feb-95 13:48:10 GMT - дата внесення змін

Для проектування CGI-модулів та сервлетів програміст повинен володіти знаннями про основні елементи мови HTML – інструменту створення web-сторінок і особливо ланки для зв'язку сторінок з програмами, розташованими на сервері. Коротко нагадаємо елементи форм HTML, функція яких полягає у формуванні параметрів для передачі від клієнта до програмних засобів, розташованих на сервері. Опускаючи теги заголовків, вирінювання тощо, наведемо тег форми HTML, який має п'ять дескрипторів введення параметрів та команд: `input type="TEXT"` – для введення прізвища, `TEXTAREA` – для введення тексту опису, `input type="radio"` –

для вибору одного з можливих слів, input type="SUBMIT" – для подання команди на пересилання запиту з набраними даними, <input type="reset"> – для відновлення початкового стану полів.

```
    <form action="http://localhost:8080/MyProgram"
method="POST">
       Прізвище  <input type="TEXT" name="name" >
    <BR>
     <TEXTAREA name="description" row=5 cols=40>
      </TEXTAREA>
    <input type="radio" name="sex" value="male"> чол.
     <input type="radio" name="sex" value="female"> жін.
     <BR>
     <input type="SUBMIT" value="Запит">
     <input type="reset">
    </form>
```

Після натискання кнопки пересилання запиту броузер групує параметри і методом, на який вказує дескриптор method (за замовчуванням GET), формує запит і надсилає його до сервера. Доступ до надісланих параметрів програми сервера здійснюють відповідними методами.

У WWW-серверах прикладні програми або cgi-модулі для опрацювання потоків даних від клієнтів та формування відповідей для них можуть бути написані на мовах різних рівнів, зокрема, C/C++. Pascal, Perl, Visual Basic тощо. Характерною властивістю інтерфейсу є створення нових процесів на сервері у разі надходження кожного нового запиту (рис. 6.2, *а*). Мова Java і WWW –сервери, що підтримують її, також дають змогу також проектувати динамічний інтерфейс користувача для доступу до баз даних, пристроїв тощо. У цьому разі на сервері розташовуються так звані сервлети – модулі опрацювання інформації типу "запит-відповідь" для Web-серверів. Сервери не вимагають створення нових процесів для кожного нового запиту. Багато сервлетів виконуються паралельно в межах одного процесу на сервері (рис. 10.2) без перезавантаження.

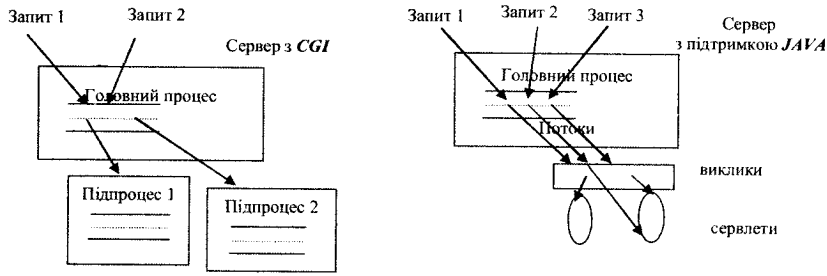


Рис. 10.2. Схеми реакції різних серверів на запит клієнта

Сервлет відповідає за опрацювання інформації з HTML-форм (web-сторінок) для оновлення баз даних та для генерації динамічних документів. Функції сервлета на сервері подібні до функцій аплета на броузері. Сервлети не мають графічного інтерфейсу. Сервлети є вбудовані у різні сервери, оскільки інтерфейс, використаний для написання сервлетів, не пов'язаний з середовищем сервера та протоколами зв'язку. Сервлети є поширені серед HTTP-серверів. Web-сервери підтримують інтерфейс сервлетів. Сервлети обходять проблему програмування серверів з платформозалежними інтерфейсами прикладного програмування, оскільки вони розробляються на стандартному Java-розширенні - Java Servlet API.

Основними завданнями сервлетів є:

• Організація співпраці між людьми. Сервлет підтримує багато запитів одночасно та їх синхронізує. Це дає їм змогу підтримувати on-line-конференції, -магазини, -ярмарки тощо. Опрацювання форм-замовлень (кредитних карток) є частиною системи прийому та опрацювання замовлень до баз даних товарів чи послуг системою оплати on-line.

• Перенапрямлення запитів. Сервлети перенаправляють запити до інших серверів або сервлетів. Так врівноважується навантаження між декількома серверами, що містять ресурси однакового змісту, розподіляється один логічний сервіс на декілька серверів відповідно до типу задачі або інших організаційних обмежень.

10.2. Будова та основні методи сервлетів

Головним батьком усіх сервлетів є інтерфейс `Servlet`. Він оголошує, але не реалізує методи, які управляють роботою сервлета та його спілкуванням з клієнтами. Під час написання сервлета розробник має реалізувати деякі або всі методи інтерфейсу.

Наступний рівень ієрархії – абстрактний клас `GenericServlet`, у якому частково реалізовані методи `Servlet` і які є базою для створення специфічних сервлетів. Як базовий приймається абстрактний клас `HttpServlet`, який доповнений методами, характерними для протоколу `HTTP`. Для побудови працездатного сервлета необхідно створити клас, що успадковує HttpServlet, і в якому потрібно формувати логіку його роботи (рис. 10.3).

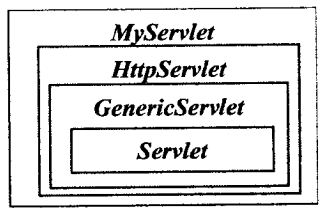


Рис. 10.3. Ієрархія побудови сервлетів

Інтерфейсом `Servlet` передбачена реалізація двох методів: `getServletConfig` і `getServletInfo`. Перший повертає об'єкт типу `ServletConfig`, що містить параметри конфігурації сервлета, а другий – рядок, який описує призначення сервлета та іншу інформацію. Сервлет `HttpServlet` відповідає за опрацювання запитів `HTTP`-протоколу. В ньому є реалізований метод `service`, який слугує диспетчером інших методів для опрацювання доступу до ресурсів. Специфікація `HTML` визначає такі методи: `GET` – універсальний запит ресурсу за його універсальною адресою `(URL)`, `POST` – метод передачі даних (введених користувачем у поля інтерактивних `Web`-сторінок) на сервер, `DELETE` – знищення даних, які знаходяться за адресою `URI`, `PUT` – розміщення даних за вказаним `URI`, `OPTIONS` – запит інформації про комунікаційні параметри сервера, `TRACE` – вимога на повернення тіла запиту без змін (для відлагодження).

Метод service (HttpServlet) визначає метод доступу до ресурсів у запиті і викликає відповідний метод, ім'я якого відповідає назві методу доступу до ресурсів, але з префіксом do: doGet, doPost, doPut, doDelete, doOptions і doTrace. Розробнику необхідно перевизначити потрібний метод (найчастіше це doGet), розмістивши в ньому функціональну логіку програми.

Взаємодія з клієнтом. Приймаючи запит від клієнта, сервлет отримує два об'єкти класів пакета javax.servlet.Http:

- Об'єкт класу HttpServletRequest призначений для встановлення зв'язку від клієнта до сервера. Він надає сервлета доступ до: 1) імен параметрів, що надсилаються клієнтом; 2) протоколу, що використовується клієнтом; імен віддаленого клієнта, що робить запит; 4) сервера, який його отримав; 5) вхідного потоку ServletInputStream. Вхідний потік містить дані від клієнтів, які використовують протоколи аплікацій для методів POST і PUT. Клас HttpServletRequest успадковує ServletRequest і дає змогу сервлетам отримувати інформацію про HTTP заголовки.
- Об'єкт класу HttpServletResponse призначений для встановлення зв'язку від сервлета зворотно до клієнта. Він містить методи для надсилання повідомлень клієнтові, а саме: 1) дає змогу сервлета встановити довжину змісту і тип MIME відповіді; 2) створює об'єкт вихідного потоку ServletOutputStream. За допомогою об'єкта класу PrintWriter сервлет відправляє відповідь. Клас HttpServletResponse, що успадковує ServletResponse, містить методи для маніпуляції інформацією HTTP-заголовків.

Послідовність роботи сервлета є такою: ініціалізація, обслуговування запитів і завершення існування. У кожній з цих фаз сервер, на якому виконується сервлет, викликає відповідний метод інтерфейсу Servlet. Першим викликається метод init. Він дає сервлета можливість ініціалізації даних і підготовки для опрацювання запитів. Найчастіше цим методом програмісти зберігають вхідний потік даних і початкові значення для обчислень.

Пізніше сервер знаходиться в стані очікування запитів від клієнтів. Запит спричинює виклик методу `service` сервлета. Усі параметри запиту, розміщені в об'єкті класу `ServletRequest`, передаються через перший параметр для методу `service`. Другим параметром методу є об'єкт класу `ServletResponse`, в якому розміщаються вихідні дані відповіді, які формуються для клієнта. Кожний новий запит спричинює новий виклик методу `service`. Відповідно до специфікації JSDK, метод `service` може опрацьовувати відразу кілька запитів, тобто бути синхронізованим для виконання у багатопоточних середовищах. Це особливо критично, якщо звертання в ньому відбувається до спільних даних. За необхідності уникнути множинних запитів сервлет повинен реалізувати інтерфейс `SingleThreadModel`. Останній не містить жодного методу і слугує лише міткою для повідомлення серверу про однопоточну природу сервлета. Під час звертання до такого сервлета кожен новий запит буде розміщений у черзі для очікування завершення процесу опрацювання попереднього запиту.

Після завершення виконання сервлета сервер викликає метод `destroy`, припускаючи, що цим методом сервлет "звільнить" зайняті раніше ресурси. Під час програмування цього методу розробник повинен пам'ятати, що в багатопоточних середовищах сервер може викликати метод `destroy` в будь-який момент часу, навіть, якщо не завершився метод `service`. Тому важливою є обов'язкова синхронізація доступу до спільних ресурсів.

Параметри ініціалізації сервлета знаходяться у дескрипторі розміщення (так званому web.xml файлі). Нижче наводиться приклад для зберігання двох початкових параметрів:

```
<web-app>
      <servlet>
            <servlet-name>MyServletName</servlet-
      name>
            <servlet-
      class>com.mycompany.MyServlet</servlet-class>
            <init-param>
                  <param-name> param1 </param-name>
                  <param-value> value1 </param-value>
```

```
            </init-param>
            <init-param>
                  <param-name> param2 </param-name>
                  <param-value> value2 </param-value>
            </init-param>
         ...

      </servlet>

   ...
</web-app>
```

Розглянемо приклад найпростішого сервлета, який методом `init` читає параметри ініціалізації сервлета різними способами:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class MyServlet extends HttpServlet {
      public void init() throws ServletException {
      // доступ до значення параметра з іменем param1
      getServletContext().log("getinit  init");      //
параметр ініціалізації
      String value =
      getServletConfig().getInitParameter("param1");
      // або доступ до всіх поіменованих параметрів
      java.util.Enumeration enum =          // Отримати всі
початкові                                  параметри
      getServletConfig().getInitParameterNames();
      for (; enum.hasMoreElements(); ) {
      String name = (String)enum.nextElement(); //
назви параметрів
      value                                          =
getServletConfig().getInitParameter(name);
            }
      // або доступ до значення параметра з іменем param1 іншим
методом
      value                                          =
getServletContext().getInitParameter("param1");
      // код методу init
            }
      }
```

```
    public void doGet(HttpServletRequest req,
    HttpServletResponse resp) throws IOException {
        PrintWriter out = resp.getWriter(); // потік
    виводу сторінки
        out.println("<html><head><title>ASimple
        Servlet</title></head><body>");
        out.println("Today        is        "+(new
    java.util.Date()));
        out.println("</body></html>");
        out.close();
    }
    public void destroy(){
  // код методу
    }
}
```

У наведеному прикладі використовуються методи класу `ServletConfig` для отримання інформації про параметри ініціалізації чи середовища. Імена цих параметрів можна одержати через енумератор, що повертається методом `getInitParameter Names`. Значення ж конкретного параметра одержують викликом `getInitParameter`. Метод одержання контексту сервлета `getServletContext` дає змогу отримати інформацію про середовище, у якому виконується сервлет. Метод `log` записує необхідні текстові дані до протоколу роботи сервлетів.

Запит типу `GET` від клієнта містить параметри в стрічці запиту. Вони відділені від адреси `URL` знаком питання, наприклад, `http://hostname.com?p1=v1&p2=v2` містить два параметри запиту – `p1 i p2`. Використовуючи запит типу POST, параметри беруться зі стрічки та з закодованого тіла запиту. Наведемо приклад методу сервлета, універсального щодо вибору значень параметрів у різних типах запитів:

```
public void doGet(HttpServletRequest req,
HttpServletResponse resp) throws IOException { //
опрацювання Get запиту
  doGetOrPost(req, resp);              // виклик спільного
методу
}
```

```
public void doPost(HttpServletRequest req,
HttpServletResponse resp) throws IOException { //
опрацювання POST
   doGetOrPost(req, resp);
      }
private void doGetOrPost(HttpServletRequest req,
HttpServletResponse resp) throws IOException {
      String name = "param";
         // назва параметра
      String value = req.getParameter(name);      //
значення парметра
      if (value == null) {// слова 'param' немає, тобто
      http://hostname.com?a=b
      } else if ("".equals(value)) { // слово 'param' є, але
      без значення // тобто http://hostname.com?param=&a=b
      }
      PrintWriter out = resp.getWriter();          //
вихідний потік
      resp.setContentType("text/plain"); // генерація
типу сторінки
      Enumeration enum = req.getParameterNames();
      for (; enum.hasMoreElements(); ) {
      name = (String)enum.nextElement(); // назва
параметра
      out.println(name);
      value = req.getParameter(name); // значення
параметра в запиті
            // або якщо більше параметрів мають однакову назву
      String[] values = req.getParameterValues(name);
      for (int i=0; i<values.length; i++) {
      out.println("    "+values[i]);
          }
      }
      out.close();    // закриття потоку відповіді
}
```

У наведеному прикладі для управління HTTP запитами сервлет успадковує клас HttpServlet і в ньому перевизначаються відповідні методи, а саме: doGet і doPost викликають один спільний метод doPostdoGetOrPost незалежно від типу методу

надсилання запиту. Метод повертає клієнтові HTML-сторінку. Запит користувача представлений об'єктом класу HttpServletRequest, відповідь – об'єктом класу HttpServletResponse. Відповідь є текстовою інформацією, що формується в об'єкті out (клас PrintWriter), отриманого методом getWriter класу HttpServletResponse.

Перевизначений метод *doGet* класу HttpServlet викликається, якщо клієнт здійснює GET-запит (він є за замовчуванням). Управління POST-запитами здійснюється також перевизненим методом. Приклад містить низку методів доступу до об'єктів запиту: getParameter, getParameterNames, getParameterValues та відповіді: setContentType. Більше методів опрацювання запитів та формування відповіді наведено в наступному параграфі.

10.3. Додаткові можливості сервлетів

Сервлет володіє деякими додатковими об'єктами, які призначені для відстеження сесії контакту. Статус з'єднання між сервлетом і клієнтом може зберігатися протягом певного періоду часу багатократних з'єднань. Для створення ефективних сервлетів необхідно знати можливості допоміжних класів пакету. Зокрема, метод getCharacterEncoding визначає символьне кодування запиту, а методи getContentType і getProtocol - MIME-тип запиту, назву і версію протоколу. Інформацію про ім'я сервера, що приймає запит, і порт сервера, на який запит надійшов, надають методи getServerName і getServerPort. IP-адреса клієнта повертається методом getRemoteAddr, а його ім'я – методом getRemoteHost:

```
String addr = req.getRemoteAddr();      // 123.123.123.123
String host = req.getRemoteHost();      //hostname.com
```

Прямий доступ до змісту запиту реалізується методом getInputStream чи getReader. Перший повертає вказівник на

об'єкт класу ServletInputStream, а другий – на BufferedReader. Після цього доступ до байтів отриманого запиту реалізується методами роботи з потоками мови Java. Клас ServletResponse має кілька методів: getOutputStream повертає вказівник на потік ServletOutputStream, getWriter повертає вказівник на потік типу PrintWriter. Ці методи наведено у прикладах сервлетів.

Сервлет виділяє параметри запиту, сформовані і надіслані клієнтом. Прикладом є електронна анкета, виконана у вигляді форм введення, значень полів і кнопок на web-сторінці. Вони автоматично перетворюються на параметри URL. Три спеціальних методи класу ServletRequest займаються поділом параметрів та вилученням їх значень. Метод getParameter повертає значення параметра за його ім'ям чи null, якщо параметра з таким ім'ям немає. Метод getParameterValues повертає масив рядків, якщо параметр є складний, наприклад, значення полів форми. Метод getParameterNames повертає імена усіх надісланих параметрів.

Розглянемо приклади використання вказаних методів сервлета на прикладі метода під час надсилання зображення до клієнта:

```
public void doGet(HttpServletRequest req,
HttpServletResponse resp) throws IOException {
        ServletContext sc = getServletContext();
        String filename = sc.getRealPath("image.gif");//
назва файла
        String mimeType = sc.getMimeType(filename);
 // MIME тип
        if (mimeType == null) {
            sc.log("Не знайдено MIME тип для
        "+filename);
resp.setStatus(HttpServletResponse.SC_INTERNAL_SERVE
R_ERROR);
            return;
        }
        resp.setContentType(mimeType); // встановити тип
        зображення
        File file = new File(filename);       // створити
        файл
```

```
        resp.setContentLength((int)file.length());
        FileInputStream in = new FileInputStream(file);
        // вих. потік
        OutputStream out = resp.getOutputStream();
        byte[] buf = new byte[1024]; // підготовка до
копіювання файла
        int count = 0;
        while ((count = in.read(buf)) >= 0) {
              out.write(buf, 0, count);       // копіювання
        файла
                    }
        in.close();
        out.close();
}
```

Розглянемо приклад сервлета, який виводить у вікні броузера список файлів визначеного каталога комп'ютера-сервера, показує кількість файлів та їх місце розташування.

```
 public class SampleServlet extends HttpServlet {
        public void doGet( HttpServletRequest req,
        HttpServletResponse  res)    throws
        ServletException, IOException {
        String param;
        PrintWriter w = res.getWriter(); // потік відповіді
для сервера
        res.setContentType("text/html");      // тип HTML-
сторінки
        generateHeader("Dir. Viewer Servlet",w); //
генерація заголовка
        if((param=req.getParameter("dirToShow"))==null)
        if((param = getInitParameter("dirToShow"))
        ==null){ // розшифрування параметра властивостей
        w.println("<H1><FONT COLOR=RED>" +    // у потік-
повідомлення
  "The <EM>dirToShow</EM> parameter
required!</FONT></H1>");
      return;              // опрацювання запиту переривається –
відсутній параметр
        }
        File root = new File(param);
```

```
        if(!root.isDirectory())                    // чи існує
    каталог з таким ім'ям
        {
        w.println("<H1><FONT COLOR=RED>" + // повідомлення
    для клієнта
        "The parameter is not a directory or not
        exist!</FONT></H1>");
        return; // запит переривається, відсутній парметр у заголовку
        та властивостях
        }
        File[] fileList = root.listFiles(); // список
    файлів каталогу
        w.println("<H2><FONT COLOR=TEAL>" +  //
    повідомлення для клієнта
        "Total number of files in the choosen directory
    - " +
        fileList.length + "</FONT></H2>"); //загальна
    кількість у каталозі
        w.println("<H3><FONT COLOR=PURPLE>" +
        "Directory path - " + param +
    "</FONT></H3><HR>"); // каталог
        w.println("<TABLE BORDER=0 CELLSPACING=5>");//
    форматування
        for(int i = 0; i < fileList.length; i++)
        printName(fileList[i], w); //виводяться імена файлів у
    таблиці
        w.println("</TABLE><HR>");
        generateFooter(w);     // генерація закінчення
    HTML-сторінки
                                    // допоміжні методи
    private void printName(File name, PrintWriter
output){
        String type = name.isDirectory()
        ? " (Directory)" : " (File)";   // чи є виведене ім'я
        каталогом чи простим файлом
        output.println("<TR><TD>"
        type+"</TD><TD><FONTCOLOR=BLUE>" i +
        name.getName() + "</FONT></TD></TR>");// тип файла,
        його ім'я
        }
```

```
    private void generateHeader(String title,
PrintWriter
          output) {
          output.println("<HTML>\n<HEAD>\n<TITLE>" +
          title + "</TITLE>\n</HEAD>\n<BODY>");
          }
    private void generateFooter(PrintWriter output  {
          output.println("</BODY>\n</HTML>");
          output.flush();
          output.close();
          }
      public String getServletInfo()
          {
          return "This servlet shows a content of a
    directory" +
           "mentioned in dirToShow parameter or
property.";
          }
    }
```

Наведений приклад сервлета містить перевизначений метод `doGet`, яким формується HTML-сторінка у вигляді таблички з назвами файлів. Сервлет має низку допоміжних методів: формування заголовку та закінчення сторінки, друку назви файла та службового повідомлення. Приклад ілюструє необхідність знання мови HTML для формування відповідей для клієнта.

Визначити метод доступу до ресурсів (на основі якого побудований запит) можна за допомогою виклику `getMethod`. Рядок HTTP-запиту можна одержати методом `getQueryString`. Ім'я користувача, від якого надіслано запит, надає метод `getRemoteUser`.

Контекст виконання сервлета дає певні засоби для спілкування із сервером. Якщо для виконання задачі необхідно знати MIME-тип файла, то викликається метод `getMimeType`.

Метод `getRealPath` дає можливість отримати шлях до файла стосовно каталога. Інформація щодо сервера надається методом `getServerInfo`.

Наведемо ще низку методів класу HttpServletResponse: sendError повідомляє код помилки та текст попередження, setDateHeadery додає параметри до заголовку відповіді.

Метод getServlet завантажує сервлет за його ім'ям, а метод getServletNames повертає енумератор з іменами усіх встановлених сервлетів.

Сервлети мають об'єкти, які працюють із закладками cookie. Цей механізм призначений для зберігання певної інформації про клієнта і надсилання їх на сервер. Надіслати cookie на клієнтську станцію можна методом addCookie. Викликати закладку cookie (отриманий із запитом) можна методом getCookies.

Більшість зазначених класів описані у складі пакету Java Servlet Development Kit.

10.4. Запуск та налаштування сервлетів

Для налаштування сервлетів необхідно розглянути файл властивостей – servlet.properties. У ньому в парах "ключ-значення" зберігаються дві властивості, що використовуються для конфігурації, створення та ініціалізації сервлетів. Першою є servlet.<ім'я сервлета>.code, що визначає ім'я сервлета і ставить його у відповідність двійковому class-файла сервлета. Наприклад, якщо скомпільовано клас сервлета з ім'ям MyServletClassName, то в результаті компіляції файла з ім'ям MyServletClassName.class можна надати коротке ім'я, наприклад, myservlet таким способом:

```
servlet.myservlet.code=MyServletClassName
```

Під час звертання до сервлета з ім'ям myservlet сервер знайде цей рядок у файлі властивостей і завантажить клас MyServletClassName, ініціалізує його та передасть йому запит. Ім'я class-файла повинне задаватися цілком, враховуючи ім'я пакета, у якому визначений клас.

Другою властивістю є servlet.<ім'я сервлета>.initargs, що визначає передані сервлета параметри ініціалізації. Значення параметрів можуть бути отримані методом getInitParameter. Якщо параметрів мало, вони відокремлюються комами. Приклад завдання параметрів такий:

```
servlet.myservlet.initargs= \
parameterName1=sValue1, parameterName2=sValue2
```

Файл servlet.properties розміщають у визначений каталог на сервері, де зберігаються class-файли. Різні сервери допускають альтернативне місце розташування файла властивостей (на розгляд адміністратора).

Наведемо текст файла властивостей servlet.properties для наведеного прикладу сервлета перегляду назв файлів каталогу:

```
# dirlist servlet
servlet.dirviewer.code=SampleServlet
servlet.dirviewer.initArgs=\
dirToShow=D:\\SUN\\SDK\\examples
```

Символ "зворотня коса" у шляху до каталогу задається за правилами мови Java, тобто подвійними "зворотними косими".

Для роботи із сервлетами призначена утиліта servletrunner. Після запуску вона опитує порт 8080, і у разі надходження запиту звертається до сервлета, отримує від нього відповідь і пересилає його до програми-клієнта. Командний рядок servletrunner має багато опцій, серед яких корисними можуть виявитися такі:

- p (port) – "порт, що прослуховується" під час очікування запиту;
- t (timeout) – час тайм-ауту в мс;
- d (dir) – каталог, де зберігаються сервлети;
- r (root) – кореневий каталог, у якому зберігаються документи;
- s (filename) – альтернативне ім'я файла властивостей.
- v – відображати дані, виведені в стандартні потоки.

Після запуску утиліти servletrunner (як найпростіший web-сервер) до неї можна звертатися за допомогою браузера чи

іншої клієнтської програми. Утиліта servletrunner має знайти необхідні сервлети.

Запускається сервлет таким рядком:

```
Servletrunner -p 80 -d E:\PROJECTS\Java\Servlet -r
E:\PROJECTS\Java\Servlet
```

Відлагоджується сервлет під час перегляду тексту сторінки броузером. Помилки висвітлюються. З web-броузера можна звернутися до сервлета за допомогою відправлення запиту на адресу URL, яка складається з декількох частин: імені комп'ютера, номера порта, каталогу servlet, імені сервлета і списку параметрів. Наприклад, звертання до сервлета myservlet, що знаходиться на локальному комп'ютері, має вигляд:

```
http://localhost:8080/servlet/myservlet?param1=svalue1
```

Наведений запит звертається до сервлета myservlet, передаючи йому параметр param1 зі значенням svalue1.

Звертання до сервлета з попереднього параграфа має вигляд:

```
http://mitrich/servlet/dirviewer?dirToShow=D:/Sun/SDK/exa
mples
```

Без професійних серверів, які підтримують Java-технологію (Tomcat, JSWDK, Java Web Server, тощо), налагодити складні сервлетні класи важко, оскільки утиліта servletrunner має найпростіші можливості для їх запуску.

10.5. Основні інструменти JSP-технології

Розширення динамічних властивостей web-сторінок разом зі скриптами та аплетами можливе за допомогою технології JSP (Java Server Pages), яка полягає у внесенні в тіло web-сторінки фрагментів java-програм (скриплетів). Останні перетворюються на сервлети, компіюються та виконуються сервером. Згідно з виконаними операціями формується чи модифікується вигляд сторінки для перегляду на броузері. Технологія JSP має низку переваг над:

- статичними HTML-сторінками, які в чистому вигляді не можуть створити динамічних сценаріїв, відображаючи наперед приготовані тексти та графічні зображення;
- сторінками з увімкненим кодом на JavaScript, який може маніпулювати з даними, розташованими на комп'ютері клієнта;
- ввімкненнями SSI (Server-Side Includes), які дають змогу робити найпростіші динамічні вставки в текст сторінки з боку сервера і потребують додаткових програм на сервері;
- сервлетами, які для внесення змін на сторінці повинні мати відповідні коди виводу у вихідний потік цих змін, тоді як JSP дає змогу писати безпосередньо в тексті сторінки коди змін;
- технологією ASP (Active Server Pages) від Microsoft, у якій коди записуються мовою Visual Basic або іншими мовами від MS, і виконуються на серверах, що підтримують програмне забезпечення від MS.

Технологія JSP дає змогу формувати такі типи включень у HTML-сторінку:

- прихований коментар:

```
<%-- JSP прихований коментар--%>
```

- видимий на сторінці коментар:

```
<!—JSP коментар, видимий на сторінці →
```

- директорії

```
<%@ page import=»com.mycompany.*» %>
```

Java–вирази, які під час завантажування сторінки виводять текстовий результат їх дії, наприклад:

```
Значення  пі є: <%= Math.PI %>
Випадкове число в діапазоні від 1 до 100 є:
<%= (int)(Math.random()*100)+1 %>
Значення параметра «р» в запиті є:
<%= request.getParameter(«р») %>
Сьогодні маємо дату
<%= new
java.text.SimpleDateFormat(«EEEE»).format(new
java.util.Date()) %>
```

- описи змінних, функцій тощо:

```
<%!
```

```
        String getMsg() {
            return «try again»;
      }
    %>
```

- скриплети, які генерують текст у вихідний потік:

```
    <%
      if (Math.random() > .5) {
          out.println(«<b>You win!</b>»);
      } else {
          out.println(«Sorry, you lose. Try again.»);
      }
     %>
```

Розглянемо приклад сторінки з простим фрагментом скриплету:

```
       <html>
    <head><title>Простий приклад JSP
сторінки</title></head>
    <body>
    <pre>
        На даній сторінці наведено приклад простого
скриплету
        <hr>
      <%-- Це коментар, що не виводиться на сторінці --%>
     <%!
        String getMsg() {
            double rand = Math.random();
            if (rand < .1) {
                return «you win!»;
            } else {
                return «try again»;
                          }
              }
             %>
      <%     out.println(getMsg());     %>
        /pre>
    </body>
    </html>
```

Розглянемо фрагменти скриплетів, які використовують структури вибору текстів для виводу на сторінці. Використовуються

функції сервлетів, описані в розділі про сервлет. Код переривається текстом HTML-сторінки:

```
    <% if
(«sValue».equals(request.getParameter(«param1»))) { %>
     Оголошення номер 1 ( param1 має значення sValue)
   <% } else { %>
     Оголошення номер 2
   <% } %>
```

або

```
   <%
switch(Integer.parseInt(request.getParameter(«param2»)))
{
       case 0: %>
       Оголошення номер 1 ( param2 = 0)
   <%     break;
       case 1: %>
       Оголошення номер 2   (якщо param2 = 1)
   <%     break;
       default: %>
      Оголошення номер 3 (param2 має інші значення)
   <% } %>
```

За допомогою методів сервлетів JSP-сторінка має доступ до заголовку запиту, що проілюстровано таким прикладом:

```
Метод запиту є <%= request.getMethod() %>
    URI запиту наступний <%= request.getRequestURI() %>
    Протокол запиту є <%= request.getProtocol() %>
    Використано браузер <%= request.getHeader(«user-
agent») %>
```

10.6. Бібліотека стандартних тегів

Розширення можливостей програмування JSP сторінок досягається використанням тегів (і відповідних методів) з бібліотеки стандартних тегів (The JavaServer Pages Standard Tag Library – JSTL).

Теги біліотеки JSTL класифіковані згідно з їхніми функціями у групи:

- Теги ядра (Core Tags)
- Форматування (Formatting tags)
- Теги SQL
- Теги XML
- Функції JSTL.

Для встановлення бібліотеки необхідно переписати та розрахувати відповідний дистрибутивний файл з ресурсу (Apache Tomcat) Apache Standard Taglib. Або скопіювати дитрибутив Jakarta Taglibs (JAR) до каталогу 'lib' \ROOT\WEB-INF\lib. Відповідно вказати назву каталогу для під'єднання у директиві <taglib> для кожної JSP-сторінки.

Нижче наведено приклад директиви під'єднання бібліотеки Core Tags – найчастіше вживаних тегів:

```
<%@ taglib prefix=«c»
          uri=«http://java.sun.com/jsp/jstl/core» %>
```

Всі теги бібліотеки Core JSTL наведені в таблиці:

Тег	Опис
1	2
<c:out >	Як <% = ... >, Але для виразів
<c:set >	Встановлює результат обчислення виразу в «scope»
<c:remove >	Видаляє область дії змінної (від конкретної області, якщо вказана)
<c:catch>	Перехоплення виняткової ситуації та повідомлення про неї
<c:if>	Умовний оператор
<c:choose>	Оператор вибору
<c:when>	Оператор вибору
<c:otherwise >	Оператор вибору
<c:import>	Повертає абсолютну або відносну URL-адресу

Продовження таблиці

1	2
<c:forEach >	Основний тег ітерації.
<c:forTokens>	Обробник речень
<c:param>	Додавання параметра в URL для тега 'import'
<c:redirect >	Перенапрямлення на новий URL
<c:url>	Створює URL з додатковими параметрами запиту

Для форматування тексту, дати, часу, міжнародних сторінок використовуються теги форматування (Formatting):

```
<%@ taglib prefix="fmt"
          uri="http://java.sun.com/jsp/jstl/fmt" %>
```

Усі теги форматування бібліотеки JSTL наведно в таблиці:

Тег	Опис
<fmt:formatNumber>	Для відображення числового значення з конкретною точністю або формату
<fmt:parseNumber>	Аналізує рядок подання числа
<fmt:formatDate>	Формати дати і / або часу
<fmt:parseDate>	Розбір строкового представлення дати і / або часу
<fmt:bundle>	Завантажує набір ресурсів
<fmt:setLocale>	Збереження даної місцевості в змінній конфігурації місцевості.
<fmt:setBundle>	Завантаження набору ресурсів
<fmt:timeZone>	Визначає часовий пояс
<fmt:setTimeZone>	Збереження цього часового поясу в змінній конфігурації часового поясу
<fmt:message>	Для відображення різних повідомлень
<fmt:requestEncoding>	Встановлює кодування запиту

JSTL бібліотека SQL-тегів забезпечує теги для взаємодії з реляційними базами даних (СКБД), такі як Oracle, MySQL або Microsoft SQL Server. Нижче наводено синтаксис, щоб ввімкнути JSTL SQL бібліотеку в вашому JSP:

```
<%@ taglib prefix="sql"
           uri="http://java.sun.com/jsp/jstl/sql" %>
```

Список SQL тегів JSTL наведено в таблиці:

Тег	Опис
<sql:setDataSource>	Створення простого DataSource
<sql:query>	Виконує SQL запит
<sql:update>	Виконує SQL оновлення,
<sql:param>	Встановлює параметр у SQL запит
<sql:dateParam>	Встановлює параметр дати в SQL запит java.util.Date.
<sql:transaction >	Виконання усіх операторів в одній транзакції.

JSTL містить низку стандартних функцій, більшість з яких є загальні функції роботи з рядками. Нижче наводиться синтаксис ввімкнення JSTL-бібліотеки функцій у JSP:

```
<%@ taglib prefix="fn"
           uri="http://java.sun.com/jsp/jstl/functions"
%>
```

Список функцій JSTL наведено в таблиці.

Функція	Опис
1	2
fn:contains()	Чи вхідний рядок містить підрядок.
fn:containsIgnoreCase()	Чи вхідний рядок містить підрядок (нечутливий до регістру)
fn:endsWith()	Чи вхідний рядок закінчується зазначеним суфіксом

Продовження таблиці

1	2
fn:escapeXml()	Вилучення символів XML-розмітки
fn:indexOf()	Повертає індекс першого входження підрядка
fn:join()	Об'єднує всі елементи масиву в рядок
fn:length()	Повертає кількість елементів у колекції або кількість символів у рядку
fn:replace()	Повертає рядок у результаті заміни в рядку введення всіх входжень з даним рядком
fn:split()	Розбиває рядок на масив підрядків
fn:startsWith()	Чи вхідний рядок починається з вказаного префікса
fn:substring()	Повертає підмножину рядків
fn:substringAfter()	Повертає підмножину рядків після певного підрядка
fn:substringBefore()	Повертає підмножину рядків до певного підрядка
fn:toLowerCase()	Перетворює всі символи рядка на нижній регістр
fn:toUpperCase()	Перетворює всі символи рядка на верхній регістр
fn:trim()	Видаляє пробіли з обох кінців рядка

Контрольні запитання та вправи

1. Чи є відмінність для програміста під час написання сервлета використовувати методи `GET` та `POST`?

2. Чи забезпечують безпеку передачі параметрів HTML-форми?

3. Які завдання виконують сервлети? Яке призначення основних методів сервлета?

4. Які об'єктами обмінюється сервлет з сервером? Що містять дані об'єкти? За яким протоколом працюють сервлети? Яке призначення методів `doGet`, doPost? Якими методами оперує програміст під час перевизначення зазначених методів? Що повинен отримати програміст за допомогою цих методів?

5. Що поверне метод `getParameter`, якщо немає параметрів?

6. Яку інформацію допоможуть отримати методи класів сервлетів?

7. Знаннями про яку мову (її основні положення) повинен володіти програміст, щоб проектувати та програмувати сервлети? Що є теги? Які ще знаєте механізми створення динамічних web-сторінок?

8. Де зберігаються параметри ініціалізації сервлета і як можна отримати доступ до їх значень?

9. Що розуміємо під термінами Header та Footer?

Глава 11
ДОСТУП ДО ВІДДАЛЕНИХ БАЗ ДАНИХ

11.1. Завантаження драйверів баз даних

Значний відсоток програмного забезпечення створюється для роботи з базами даних. Основні мови програмування мають розвинені механізми вибору, збереження та корекції інформації в базах даних. Завдяки пакету JDBC (Java Database Connectivity) з Java-програми можна під'єднуватись до системи баз даних та взаємодіяти з нею за допомогою мови керування базами даних – SQL (Structured Query Language). JDBC є інтерфейсом рівня SQL-виклику (доступу). Сам JDBC не може працювати, а використовує основні абстракції і методи ODBC (Open Database Connectivity) – загального інтерфейсу доступу до найрізноманітніших баз даних. На відміну від інтерфейсу ODBC, JDBC організований простіше. Головною його частиною є драйвер для доступу з JDBC до джерел даних. Цей драйвер є найвищим в ієрархії класів JDBC і називається DriverManager. На рис. 11.1 наведено схему під'єднання Java-програми до бази даних. Менеджером є клас DriverManager, який має структуру даних для зберігання самих драйверів – об'єктів типу Driver – та інформації про них.

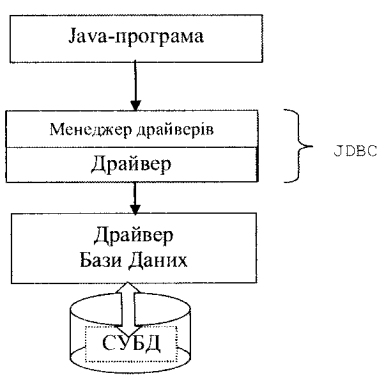


Рис. 11.1. Доступ до бази даних за різними схемами

Драйвери JDBC створюють постачальники СУБД. Їх робота полягає в опрацюванні під'єднань та команд від Java-програми для генерації машинно-залежних викликів щодо бази даних. За відсутності драйверів JDBC (а драйвери ODBC постачаються майже завжди) варіантом доступу є так званий JDBC-ODBC-Міст.

ODBC був використаний як основа JDBC через його популярність серед незалежних розробників програмного забезпечення і користувачів. Крім того, шлях через JDBC-ODBC-Міст може виявитися коротшим.

Спеціалісти фірми Sun розробили спеціалізований пакет java.sql, який дає змогу мові Java здійснювати доступ до системи управління реляційними базами даних. Якість доступу оцінюються можливостями засобів доступу до баз даних, зручності та повноти інтерфейсів. Наприклад, пакет java.sql версії JDK1.1 містить класи, які забезпечують більшість функцій, характерних для ODBC-програм, наприклад:

java.sql.CallableStatement – забезпечує виконання процедур, що зберігаються;

java.sql.DatabaseMetaData – створює можливості дослідження властивостей бази даних щодо її реляційної повноти, цілісності, отримання даних про типи та зміст таблиць, колонок, індексів, ключів тощо;

java.sql.ResultSetMetaData – методи класу дають змогу виводити інформацію з таблиць бази даних (друкувати назви таблиць і колонок).

Відмінність Java від інших мов програмування полягає в тому, що функції доступу до баз даних можна організувати гнучко, використовуючи переваги сучасних об'єктно-орієнтованих технологій, WWW і Intranet/Internet. Розглянемо варіанти використання Java-програм під час взаємодії з базами даних.

Щоб приступити до проектування баз даних насамперед необхідно завантажити бажані драйвери до тих СУБД, форматами яких буде користуватись програміст.

Інформацію про драйвери відповідних виробників можна отримати на сайтах:

http://java.sun.com/products/jdbc - Sun
http://otn.oracle.com/software/content.html - Oracle
http://mmMySQL.sourceforge.net - MySQL database
http://www.j-netdirect.com - SQL Server
http://www.datadirect-technologies.com
http://www.freetds.org

Завантажити необхідні драйвери (вони будуть автоматично зареєстровані системою) треба перед під'єднанням до відповідних СУБД такою командою з командної стрічки:

```
> java -
Djdbc.drivers=com.company1.Driver:com.company2.Driver MyApp
```

або конкретно

```
> java -
Djdbc.drivers=sun.jdbc.odbc.JdbcOdbcDriver:Oracle.jd
bc.driver.OracleDriver MyJdbcApp
```

Другим рекомендованим способом є виклик методу `Class.forName()` в межах коду:

```
try {
      String driverName ="org.gjt.mm.mysql.Driver";

      Class.forName(driverName);
      } catch (ClassNotFoundException e) {
      }
```

Після завантаження драйверів є корисною інформація про наявні завантажені JDBC драйвери. Приклад фрагменту програми наведений нижче:

```
List drivers =
Collections.list(DriverManager.getDrivers());
for (int i=0; i<drivers.size(); i++) {// цикл за числом
драйверів
Driver driver = (Driver)drivers.get(i);
String name = driver.getClass().getName();    // назва
драйвера
int majorVersion = driver.getMajorVersion(); // версія
  int minorVersion = driver.getMinorVersion();
boolean isJdbcCompliant = driver.jdbcCompliant();
}
```

Після реєстрації драйвера за допомогою диспетчера драйверів його можна застосувати для під'єднання до бази даних. Про створення нового під'єднання треба повідомити диспетчера. Отримавши повідомлення, диспетчер викличе відповідний драйвер і поверне вказівник на встановлене під'єднання. Для створення під'єднання необхідно вказати місце розташування бази даних, ім'я користувача та пароль. Створення під'єднання відбувається за допомогою об'єкта класу `Connection`:

```
Connection
connection=DriverManager.getConnection(url,
username, password);
```

Згідно з встановленими правилами Інтернету, база даних і засоби її обслуговування ідентифікуються за допомогою URL, який задається у форматі:

```
jdbc:дод-протокол:дод-ім'я
```

де "`дод-протокол`" – назва драйвера, їх набору або механізму встановлення з'єднання з базою даних; "`дод-ім'я`" – додаткова інформація для вказаного набору. Наприклад, у разі застосування `ODBC` в `URL`-рядок підставляється ця назва та `DSN`-назва (`Data Source Name`) джерела з `ODBC.INI` файла: `jdbc:odbc:dBase`

Якщо замість `ODBC` використовується ім'я мережевого сервісу до бази даних, то до адреси `URL` входить назва сервера, порт, назва директорії, файла тощо, як це зображено у прикладі завантаження драйвера СУБД `SQLServer` (NetDirect JDBC driver) та встановлення під'єднання до відповідної бази даних:

```
Connection connection = null;
     try {
     String driverName =
"com.jnetdirect.jsql.JSQLDriver";
     String serverName = "127.0.0.1";
     String portNumber = "1433";
     String mydatabase = serverName + ":" +
portNumber;
        String url = "jdbc:JSQLConnect://" +
mydatabase;
     String username = "username";
```

```
String password = "password";
Class.forName(driverName);     // завантажити драйвер
connection = DriverManager.getConnection(url,
username, password);
    // під'єднання до бази
} catch (ClassNotFoundException e) {
                    // неможливо знайти драйвер
} catch (SQLException e) {
                    // неможливо під'єднатися
}
```

У наведеномо прикладі `url` немає останнього параметра – ідентифікатора. Під час завантаження драйвера і під'єднання до СУБД Oracle останній параметр `(sid = "mydatabase";)` вказується:

```
    Connection connection = null;
   try {
       String driverName =
"oracle.jdbc.driver.OracleDriver";
       Class.forName(driverName);
       String serverName = "127.0.0.1";
       String portNumber = "1521";
       String sid = "mydatabase";
       String url = "jdbc:oracle:thin:@" + serverName +
":" +
          portNumber + ":" + sid;
          String username = "username";
          String password = "password";
          connection = DriverManager.getConnection(url,
          username, password);
          } catch (ClassNotFoundException e) {
          // неможливо знайти драйвер
          } catch (SQLException e) {
                // неможливо під'єднатися
          }
```

Визначити назву драйвера, який бере участь у під'єднанні можна за допомогою методу `getDriver(url)` :

```
try {
      Connection conn = DriverManager.
      getConnection(url, username, password);
```

```
        Driver driver = DriverManager.getDriver(url); //
драйвер з URL
            } catch (SQLException e) {
            }
```

Завдяки універсальності і доступності механізм JDBC-ODBC-Міст використовується частіше. Він реалізований у вигляді JdbcOdbc.class (для платформи Windows JdbcOdbc.dll) і входить у різні версії JDK. Крім JdbcOdbc-бібліотек, існують спеціальні драйвери, які реалізують безпосередній доступ до баз даних через стандартний інтерфейс ODBC. Як правило, вони описуються у файлі ODBC.INI. На внутрішньому рівні JDBC-ODBC-Міст відображає методи Java у виклики ODBC і цим дає можливість використати будь-які існуючі драйвери ODBC.

11.2. Виконання SQL-операторів з Java-програми

Для доступу до баз даних і роботи з ними в прикладних програмах використовується мова SQL – структурована мова запитів. Аргументами операторів мови є таблиці та їх складові: рядки, стовці та клітинки. В цьому параграфі використовуються два найвживаніші оператори: SELECT – отримання вибірки з таблиці та INSERT – внесення значень у відповідні клітинки таблиці. Для ознайомлення з усіма можливостями SQL треба використати відповідну літературу.

Доступ до таблиць баз даних з Java-програми здійснюється за допомогою методів класу Statement. Об'єкти цього класу призначені для зберігання SQL-команд і вони створюються методом createStatement класу Connection. Після пересилання об'єкта типу Statement до бази даних за допомогою встановленого під'єднання СУБД виконає отриману SQL-команду і поверне результат її виконання у вигляді об'єкта класу ResultSet (рис. 11.2).

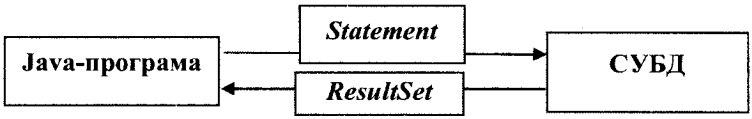


Рис. 11.2. Класи взаємодії програми з СУБД

Методи класу `Statement` повинні завжди виконуватись у межах блока `try{ } catch`. Розглянемо приклад встановлення вказівника на номер запису (рядка) в таблиці перегляду під час виконання SQL-запиту. Драйвер фізично копіює з бази даних запис з цим номером (для клієнта – *fetch size*). Цей параметр призначений для підвищення ефективності програми перегляду таблиць. Вказівник можна визначити методом `getFetchSize`, задавати методом `setFetchSize(int fetchSize)` і ним же перевизначити у вихідному об'єкті результатів класу `ResultSet`. Розглянемо приклад роботи з вказівником:

```
try {
      Statement stmt = connection.createStatement();
      int fetchSize = stmt.getFetchSize();// вказівник на
біжучий запис
      stmt.setFetchSize(100);
         // вказівник на 100-й  запис        ResultSet
      resultSet = stmt.executeQuery("SELECT * FROM
      my_table");   // об'єкт результату як результат виконання
      executeQuery
      resultSet.setFetchSize(100);                    //
вказівник на запис 100
      } catch (SQLException e) {
      }
```

Розглянемо тепер приклад доступу до конкретних даних (рядків, стовпців, клітинок) з о об'єкту класу `ResultSet`, надісланого драйвером. Доступ до полів рядка є можливий, якщо останній є біжучим. Рядок, означений вказівником, вважається біжучим. Початкове положення вказівника на запис знаходиться перед першим рядком. Після виклику методу `next` вказівник встанов-

люється на перший рядок, який стає біжучим. Можливими є два способи отримати дані з біжучого рядка. Перший використовує індекс стовпчика, починаючи з першого. Другий використовує назву стовпця. Наприклад, для запиту `SELECT col1, col2 FROM my_table` значення із стовпця `col2` можна отримати за індексом стовпця 2 або за його назвою `col2`. Наведений приклад демонструє ці два способи.

```
try {
     Statement stmt = connection.createStatement();
     ResultSet rs = stmt.executeQuery("SELECT * FROM
     my_table");
     while (rs.next()) {      // зміна номеру запису
      String s = rs.getString(2);// дані з клітинки (за
індексом стовпця)
       s = rs.getString(" col2 ");// дані з рядка за назвою
стовпця
          }
     } catch (SQLException e) {
          }
```

Кожна база даних характеризується типами даних, які можна зберігати у таблицях. Нижче наведено приклад виділення даних з типами СУБД `MySQL` (назви типів вказані під час опису змінних та об'єктів, як і в назвах відповідних методів).

```
try {
    Statement stmt = connection.createStatement();
    ResultSet rs = stmt.executeQuery("SELECT * FROM
    mysqli_all_table");
    while (rs.next()) {
      boolean bool = rs.getBoolean("col_boolean");
      byte b = rs.getByte("col_byte");
      short s = rs.getShort("col_short");
      int i = rs.getInt("col_int");
      long l = rs.getLong("col_long");
      float f = rs.getFloat("col_float");
      double d = rs.getDouble("col_double");
      BigDecimal bd =
      rs.getBigDecimal("col_bigdecimal");
```

```
        String str = rs.getString("col_string");
        Date date = rs.getDate("col_date");
        Time t = rs.getTime("col_time");
        Timestamp ts =
        rs.getTimestamp("col_timestamp");
        InputStream ais =
        rs.getAsciiStream("col_asciistream");
      InputStream bis =
      rs.getBinaryStream("col_binarystream");
      Blob blob = rs.getBlob("col_blob");
              }
      } catch (SQLException e) {
}
```

На рівні з вибором даних з таблиць використовуються об'єкти класу Statemen для внесення змін до відповідних таблиць, наприклад під час внесення змін до рядка (треба пам'ятати про стан вказівника на рядки таблиці):

```
try {
      Statement stmt = connection.createStatement();
      String sql = "INSERT INTO my_table (col_string)
      VALUES('a string')";
      stmt.executeUpdate(sql);          // внести зміни до таблиці
            } catch (SQLException e) {
      }
```

Якщо програма містить SQL-запит, який має виконуватись декілька разів, але з різними значеннями, то для підвищення ефективності роботи програми потрібно використати апарат "попередньо підготовлених команд". Наприклад, якщо з web-сторінки за допомогою запиту ведеться пошук інформації про товари на основі їх ідентифікаторів, то у цьому разі має використовуватись попередньо підготовлена команда. Для того, щоб інтерпретація SQL-команди здійснювалась тільки один раз, диспетчери баз даних підтримують концепцію "попередньо скомпільованої команди". Пакет java.sql для підвищення ефективності роботи з базами даних передбачає використання класу PrepareStatement. Розглянемо спосіб його використання на прикладі занесення рядка у таблицю бази даних.

Запит у попередньо підготовленій команді містить символи '?' замість потрібних значень полів. Необхідно спочатку занести значення, а потім виконати команду:

```
try {
      String sql = "INSERT INTO my_table (col_string)
      VALUES(?)";
      PreparedStatement pstmt =
      connection.prepareStatement(sql);
      for (int i=0; i<10; i++) {                    //
занести 10 рядків
            pstmt.setString(1, "row "+i); // 1-індекс,
      значення
            pstmt.executeUpdate();                  //
      занести рядок
            }
      } catch (SQLException e) {
      }
```

Ще один приклад показує застосування попередньо підготовленої команди для внесення змін до таблиці СУБД MySQL

```
try {
      String sql = "INSERT INTO mysqli_all_table("
      + "col_boolean,"
      + "col_byte,"
      + "col_short,"
      + "col_int,"
      + "col_long,"
      + "col_float,"
      + "col_double,"
      + "col_bigdecimal,"
      + "col_string,"
      + "col_date,"
      + "col_time,"
      + "col_timestamp,"
      + "col_asciistream,"
      + "col_binarystream,"
      + "col_blob) "
      + "VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)";
```

```
    PreparedStatement pstmt =
connection.prepareStatement(sql);
          pstmt.setBoolean(1, true);    // занести відповідні
    значення
          pstmt.setByte(2, (byte)123);
          pstmt.setShort(3, (short)123);
          pstmt.setInt(4, 123);
          pstmt.setLong(5, 123L);
          pstmt.setFloat(6, 1.23F);
          pstmt.setDouble(7, 1.23D);
          pstmt.setBigDecimal(8, new BigDecimal(1.23));
          pstmt.setString(9, "a string");
          pstmt.setDate(10, new
          java.sql.Date(System.currentTimeMillis()));
          pstmt.setTime(11, new
          Time(System.currentTimeMillis()));
          pstmt.setTimestamp(12, new
          Timestamp(System.currentTimeMillis()));

    File file = new File("infilename1"); // потік ascii
    FileInputStream is = new FileInputStream(file);
    pstmt.setAsciiStream(13, is, (int)file.length());

    file = new File("infilename2");          // потік
    двійкових значень
    is = new FileInputStream(file);
    pstmt.setBinaryStream(14, is, (int)file.length());


    file = new File("infilename3");    // потік blob
    is = new FileInputStream(file);
    pstmt.setBinaryStream(15, is, (int)file.length());

          pstmt.executeUpdate(); // занести рядок
          } catch (SQLException e) {
          } catch (FileNotFoundException e) {
          }
```

Під час завершення пересилання запитів та отримання результатів відповіді об'єкти передбачають виконання методу закриття – `close`.

Java інтерфейс до баз даних містить низку допоміжних рідше використовуваних класів, за допомогою яких користувач може визначити для баз даних і таблиць перегляду назви стовців, їх кількість, властивості тощо. До таких класів належать `DatabaseMetaData` та `ResultSetMetaData`. Розглянемо приклад використання:

1) визначити чи в базі даних підтримується механізм транзакцій (оптимізації одночасних запитів до спільних даних):

```
try {
     DatabaseMetaData dmd =
connection.getMetaData();
     if (dmd.supportsTransactions()) {
               // транзакції підтримуються
          }  else {
          // транзакції  не підтримуються
               }
     } catch (SQLException e) {    }
```

2) отримати назву драйвера, що бере участь в доступі до баз даних

```
try {
     DatabaseMetaData dmd =
connection.getMetaData();
     String driverName = dmd.getDriverName();
     } catch (SQLException e) {
     }
```

3) отримати кількість стопців таблиці перегляду (результату)

```
try {
     ResultSetMetaData rsmd =
ResultSet.getMetaData();
     int columnCount = rsmd.getColumnCount();  //
кількість стовпців
     } catch (SQLException e) {
     }
```

Розглянемо тепер процес побудови системи Java-програм (без розкриття методів класів), яка складається з серверної та клієнтської частин. Серверна частина – це сервлет з реалізованим доступом до баз даних, а клієнт надсилає запити з форм web-сторінки (рис. 11.3) для отримання даних з таблиць.

Контрольні запитання та вправи

1. Що є база даних, а що таблиця даних?
2. Які функції драйвера та диспетчера DriverManager у СУБД? Як здійснити доступ до СУБД різних виробників?
3. Які особливості доступу до баз даних в Java та інших мовах програмування?
4. Мова SQL, її призначення та складові?
5. Як формується запит у мові програмування ***Java***? Яке призначення класу Statement ?
6. Які об'єкти можна утворити методами класу Statement?
7. Які методи класу Connection найчастіше використовує програміст?
8. Яким інструментарієм повинен володіти програміст для реалізації віддаленого доступу до баз даних, крім основ програмування на Java?
9. Які класи відповідають за формування результівних таблиць?
10. Які положення мови HTML повинен знати програміст, щоб проектувати та програмувати сервлети доступу до баз даних та виведення результатів?
11. Що дозволяє механізм форм?
12. У чому полягає модульність програм доступу до баз даних?
13. Чому при доступі до баз даних необхідно використовувати блок try та атрибут synchronized?
14. Чи можуть мати доступ до баз даних інші програмні продукти, крім сервлетів?

15. Які типи виняткових ситуацій можуть генерувати сервлети доступу до баз даних та виведення таблиць? У чому фізична причина цих виняткових ситуацій?

16. Як пов'язати об'єкти отримані в результаті реалізації запитів та сторінками, які формуються для відсилання до клієнта?

17. Написати фрагменти програм доступу до СУБД, формування запитів, отримання таблиць – результатів запитів, перенесення результатів у поля web-сторінки.

18. Скласти проект системи клієнт-сервера (без сервлетів), ці функції останнього входять до формування бажаних таблиць для кожного клієнта.

СПИСОК ЛІТЕРАТУРИ

1. Томсон Л. Разработка WEB-приложений на PHP и MySQL / Л. Томсон, Л. Веллинг. – М., СПб., К. : ДиаСофт, 2001. – 655 с.
2. Дюбуа П. Применение Perl и MySQL в WEB-приложениях / П. Дюбуа. – М., СПб., К. : Вильямс, 2003. – 474 с.
3. Котеров Д. Самоучитель PHP 4 / Д. Котеров. – К. : BHV, 2002. – 572 с.
4. Дюбуа П. MySQL: использование и администрирование / П. Дюбуа. – М. : Питер, 2011. – 368 с.
5. Дейтел Х. М. Как программировать на XML с использованием Java 2, Perl/CGI, Active Server Pages / Х. М. Дейтел, П. Дж. Дейтел, Т. Р. Нието, Т. М. Лин, П. Садху. – М. : Бином, 2001. – 930 с.
6. http://htmlbook.at.ua/news/tutorial_html/1-0-1
7. http://www.w3schools.com/tags/default.asp
8. http://www.w3.org/TR/html-markup/references.html
9. http://dev.w3.org/html5/html-author/
10. http://www.w3schools.com/jsref/default.asp
11. www.php.net.
12. www.phpwizard.net.
13. www.mysql.com.
14. www.apache.org.
15. Хаммонд М. Программирование на платформе .NET / М. Хаммонд, Д. Уоткинз, Б. Ейбрамс. – М. : Вільямс, 2003. – 368 с.
16. Рихтер Д. Программирование на платформе .NET FRAMEWORK / Д. Рихтер. – М. : Русская редакция, 2002. – 480 с.
17. Бучек Г. ASP.NET. Программирование. – СПб. : Питер, 2002. – 510 с.
18. http://www.aspnet.15seconds.com
19. http://www.csharp-station.com
20. http://msdn.microsoft.com/en-us/library/9k6k3k4a (v=vs.80).aspx
21. http://www.w3schools.com/aspnet/webpages_ref_classes.asp
22. http://www.asp.net/web-pages/overview/more-resources/asp-net-web-pages-api-reference
23. http://tomcat.apache.org/tomcat-5.5-doc/jndi-resources-howto.html.
24. http://www.oracle.com/technetwork/articles/javase/servlets-jsp-140445.html

ЗМІСТ

Книги для навчання і роботи!

Левус Є. В., Мельник Н. Б.

ВСТУП ДО ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Навчальний посібник. – 2018. – 248 с.
ISBN 978-966-941-131-0

Посібник призначений для вивчення навчальної дисципліни "Вступ до інженерії програмного забезпечення". Містить теоретичні відомості, що стосуються означення інженерії програмного забезпечення, життєвого циклу програмного забезпечення, його моделей. Також подано матеріали для самостійного вивчення розділу "Групова динаміка і комунікації" навчальної дисципліни. Важливим засобом підготовки студентів до поточного й семестрового контролю є наведені у посібнику тестові завдання.

Видання призначене для студентів спеціальності "Інженерія програмного забезпечення", а також може використовуватися для інших спеціальностей галузі знань "Інформаційні технології" для вивчення дисциплін, пов'язаних з розробленням програмного забезпечення.

Левус Є. В. та ін.

ЖИТТЄВИЙ ЦИКЛ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Навчальний посібник. – 2017. – 208 с.
ISBN 978-966-941-098-6

Навчальний посібник призначений для вивчення розділу "Життєвий цикл програмного забезпечення" дисципліни "Вступ до інженерії програмного забезпечення". У посібнику, крім теоретичних відомостей, наведено чотири лабораторні роботи за основними етапами життєвого циклу: аналізу і визначення вимог, проектування й програмної реалізації, тестування, супроводу. Понад 100 тестових завдань, наведених у посібнику, є важливим засобом підготовки студентів до поточного й семестрового контролю.

Видання призначено для студентів спеціальності "Інженерія програмного забезпечення", а також може використовуватися для інших спеціальностей галузі знань "Інформаційні технології" для вивчення дисциплін, пов'язаних із розробленням програмного забезпечення.

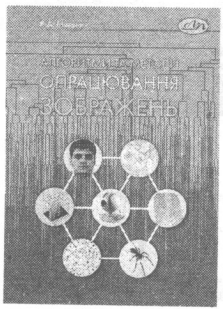
Тимовчак-Максимець О. Ю. та ін.

ПОШУКОВІ ТЕХНОЛОГІЇ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

Навчальний посібник. – Львів. – 2017. – 124 с
ISBN 978-617-607-495-3 (серія)
ISBN 978-966-941-036-8 (вип. 11)

Викладено основні поняття та види інформаційного пошуку, інформаційні потреби та їхній вплив на інформаційний пошук, розглянуто інформаційно-пошукові мови та лінгвістичне забезпечення інформаційного пошуку, охарактеризовано види пошукових систем WWW та їх функції, що широко використовують у практиці для створення пошукових запитів.

Для студентів вищих навчальних закладів – майбутніх фахівців у галузі інформаційно-пошукової діяльності, а також студентів та викладачів гуманітарних напрямів підготовки.



Мельник Р. А.

АЛГОРИТМИ ТА МЕТОДИ ОПРАЦЮВАННЯ ЗОБРАЖЕНЬ

Навчальний посібник. – 2017. – 220 с.
ISBN 978-966-941-025-2

У посібнику викладено матеріал, який стосується алгоритмів та методів опрацювання зображень з метою їхнього аналізу, класифікації та пошуку. Подано принципи комп'ютерного зору, методи сегментування та покращення зображень, алгоритми обчислення ознак зображень для їх аналізу та класифікації, методи кластерного аналізу для пошуку зображень, методи стиснення зображень.

Для студентів вишів, які вивчають сучасні технології опрацювання зображень для створення програмного забезпечення та застосувань, об'єктами яких є зображення.

Журавчак Л. М., Нитребич О. О.

ДОСЛІДЖЕННЯ ОПЕРАЦІЙ. ЛАБОРАТОРНИЙ ПРАКТИКУМ

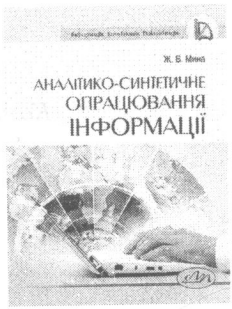
Навчальний посібник. – 2016. – 112 с.
ISBN 978-966-941-003-0

Практикум містить лабораторні роботи з дисципліни "Дослідження операцій", а саме: математичну основу для розв'язання задач оптимального управління, які можуть бути зведені до задач лінійного програмування, потокових, цілочислового програмування та теорії ігор. Виконання лабораторних робіт дає змогу студентам розширити і закріпити лекційний матеріал, отримати практичні навички розв'язання задач за різними методами, а також розроблення та відлагодження відповідних програм.

Для студентів спеціальності "Інженерія програмного забезпечення", а також для всіх, хто бажає самостійно опанувати різні методи розв'язання задач із дисципліни "Дослідження операцій". Може бути корисним для фахівців-початківців, що працюють в галузі дослідження операцій, аспірантів і науковців, а також розробників програмного забезпечення.

Мина Ж. В.

АНАЛІТИКО-СИНТЕТИЧНЕ ОПРАЦЮВАННЯ ІНФОРМАЦІЇ



Навчальний посібник. – 2016. – 196 с.
ISBN 978-617-607-495-3 (серія)
ISBN 978-617-607-926-2 (випуск 9)

У навчальному посібнику "Аналітико-синтетичне опрацювання інформації" розглянуто найважливіші теоретичні та практичні питання інформаційно-аналітичної діяльності, методику організації інформаційно-аналітичної діяльності. Особливу увагу звернено на найпоширеніші види згортання інформації: анотування, реферування та створення оглядових документів.

Для студентів вищих навчальних закладів, викладачів, спеціалістів з інформаційної діяльності та широкого кола читачів.

НАВЧАЛЬНЕ ВИДАННЯ

Мельник Роман Андрійович

ПРОГРАМУВАННЯ ВЕБ-ЗАСТОСУВАНЬ (ФРОНТ-ЕНД ТА БЕК-ЕНД)

Навчальний посібник

Редактор Олеся Косик
Технічний редактор Лілія Саламін
Комп'ютерне верстання Марти Гарасимів
Художник-дизайнер Анна Христонько

Здано у видавництво 21.02.2018. Підписано до друку 25.06.2018.
Формат 60×84 $^{1}/_{16}$. Папір офсетний. Друк офсетний.
Умовн. друк. арк. 14,41. Обл.-вид. арк. 8,3.
Наклад 150 прим. Зам. 180242.

Видавець і виготівник: Видавництво Львівської політехніки
Свідоцтво суб'єкта видавничої справи ДК № 4459 від 27.12.2012 р.

вул. Ф. Колесси, 4, Львів, 79013
тел. +380 32 2582146, факс +380 32 2582136
vlp.com.ua, ел. пошта: vmr@vlp.com.ua