МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДНІПРОДЗЕРЖИНСЬКИЙ ДЕРЖАВНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

КОНСПЕКТ ЛЕКЦІЙ

з дисципліни
„Технології та засоби створення програмного забезпечення АСУ”
(4 курс 7 семестр)
для студентів
напряму 6.050103 „Програмна інженерія”

Затверджено редакційно-видавничою
Секцією науково методичної ради ДДТУ
протокол № _9_ від 23.05.13 р.,

Дніпродзержинськ
2013

Конспект лекцій з дисципліни „Технології та засоби створення програмного забезпечення АСУ” 4 курс 7 семестр для студентів напряму 6.050103 „Програмна інженерія” / К.М. Ялова, В.В. Завгородній // Дніпродзержинськ: ДДТУ, 2013. – 176 с.

Укладачі: к.т.н., ст.викл. Ялова К.М.
ст. викл. Завгородній В.В.

Рецензент: д.т.н., проф. Самохвалов С.Є.
(Дніпродзержинський державний технічний університет)

Затверджено на засіданні кафедри ПЗС
(протокол № 13 від 24.04.13)

Наведені основні теоретичні відомості стосовно технології ASP.NET та принципи розробки веб-додатків із застосуванням мови програмування C# та технології ASP.NET. Представлено основні механізми створення, конфігурування, оптимізації та розгортання веб-додатків ASP.NET.

ЗМІСТ

ВСТУП

Даний конспект лекцій відповідає матеріалам Модулю 3 дисципліни „Технології та засоби створення програмного забезпечення АСУ”.

Мета дисципліни „Технології та засоби створення програмного забезпечення АСУ” **–** формування представлень про методи проектування і розробки програмного забезпечення (ПЗ) АСУ, принципів побудови, структури і прийомів роботи з інструментальними засобами, що підтримують створення ПЗ АСУ; методів організації роботи в колективах розробників ПЗ. На формування навиків проектування, реалізації оцінки якості і аналізу ефективності програмного забезпечення при використанні мови програмування C#, програмної платформи Silverlight, технології ASP.NET та фреймворка ASP.NET MVC.

Завдання дисципліни – формування теоретичних знань та практичних навичок у майбутніх фахівців відповідно до поставленої мети.

У результаті вивчення навчальної дисципліни студент повинен **знати:**

- принципи побудови, структури і прийоми роботи з інструментальними засобами, що підтримують створення ПЗ;
- принципи роботи і структуру програмних додатків на основі розглядуваних мов програмування, програмних платформ та технологій;
- принципи розробки користувацького інтерфейсу програмних додатків, розроблених за допомогою мови програмування C#, в рамках програмної платформи Silverlight, технології ASP.NET та фреймворка ASP.NET MVC;
- способи персоналізації та створення стилю програмних додатків, розроблених за допомогою мови програмування C#, в рамках програмної платформи Silverlight, технології ASP.NET та фреймворка ASP.NET MVC;
- принципи, способи та механізми організації роботи програмних додатків із базою даних;
- механізми оптимізації програмних додатків;
- принципи тестування, відлагодження та розгортання програмних додатків, розроблених за допомогою мови програмування C#, в рамках програмної платформи Silverlight, технології ASP.NET та фреймворка ASP.NET MVC.

Вміти із застосуванням мови програмування C#, в рамках програмної платформи Silverlight, технології ASP.NET та фреймворка ASP.NET MVC:

- застосовувати надбані навички розробки програмних додатки
- розробляти користувацький інтерфейс програмних додатків
- створювати дизайн та персональних стиль програмних додатків
- розробляти програмні додатки, що забезпечують здійснення операцій додавання, видалення, модифікацію та пошук даних із БД.
- застосовувати програмні механізми оптимізації програмних додатків
- тестувати та підлагоджувати програмні додатки;
- комбінувати різні мови і системи програмування, а також методи проектування з метою оптимального вирішення поставлених завдань.

Тема 29. Загальні принципи роботи та структура веб-додатків на основі ASP.NET 5

Розробка інформаційних систем, орієнтованих на роботу в середовищі Інтернет останнім часом стає все більш актуальною. Сучасні технології дозволяють створювати потужні, масштабовані Web-програми, при цьому, сам процес створення таких додатків стає все більш простим і доступним практично будь-якому фахівцю в галузі інформаційних систем. Спостерігається інтеграція існуючих технологій створення традиційних додатків, що працюють на комп'ютері клієнта, мережевих програм, орієнтованих на виконання потужними серверами, обслуговуючими тисячі клієнтів і засобів проектування додатків.

Технології розробки додатків, створені корпорацією Microsoft по праву вважаються одними з найбільш сучасних і передових. Вони відповідають всім сучасним вимогам, що пред'являються сьогодні до засобів розробки професійних додатків. Однією з найбільш сучасних та актуальних технологій такого роду є технологія розробки Інтернет додатків ASP.NET. Її перевагою можна вважати гнучку архітектуру, простоту, використання широко поширених мов програмування та уніфікованої технології доступу до даних.

29.1. Архітектура Web додатків

Web-програми являють собою особливий тип програм, побудованих за архітектурою „клієнт-сервер". Особливість полягає в тому, що сам Web-додаток знаходиться і виконується на сервері, клієнт при цьому отримує тільки результати роботи. Робота програми ґрунтується на отриманні запитів від користувача (клієнта), їх обробці та видачі результату. Передача запитів і результатів їх обробки відбувається через Інтернет (рис.29.1).

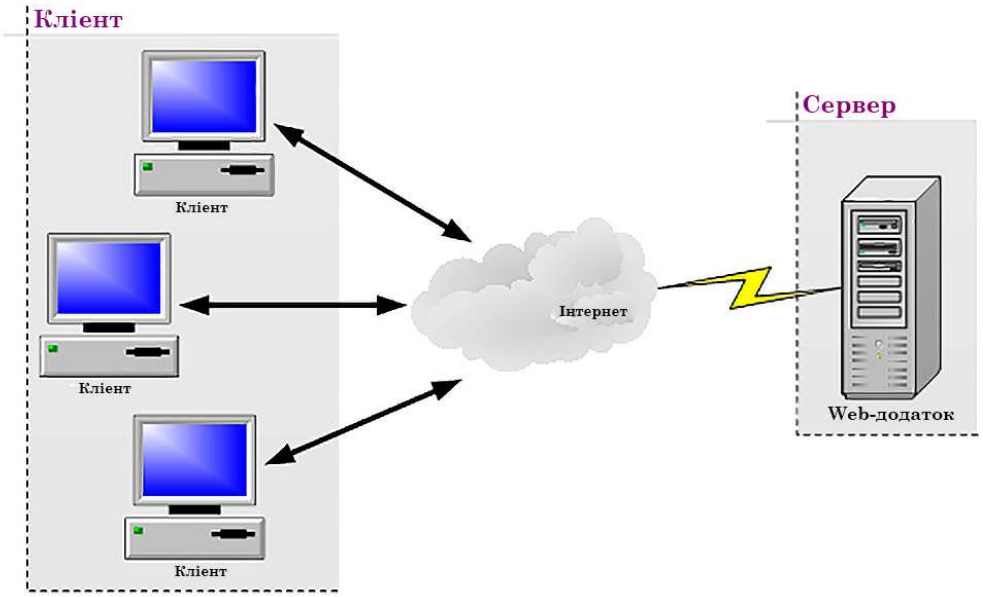

Рисунок 29.1 – Архітектура Web додатків

Відображенням результатів запитів, а також прийомом даних від клієнта та їх передачею на сервер зазвичай займається спеціальний додаток-браузер (Internet Expolrer, Mozilla, Opera і т.д.). Як відомо, однією з функцій

браузера є відображення даних, отриманих з Інтернету у вигляді сторінки, описаної мовою HTML, отже, результат, який передається сервером клієнтові повинен бути представлений на цій мові.

На стороні сервера Web-додаток виконується спеціальним програмним забезпеченням (Web-сервером), який і приймає запити клієнтів, обробляє їх, формує відповідь у вигляді сторінки, описаною мовою HTML і передає його клієнту. Одним з таких Web-серверів є Internet Information Services (IIS) компанії Microsoft. Це єдиний Web-сервер, здатний виконувати Web-додатки, створені з використанням технології ASP.NET.

У процесі обробки запиту користувача, Web-додаток компонує відповідь на основі виконання програмного коду, що працює на стороні сервера, Web-форми, сторінки HTML, іншого вмісту включаючи графічні файли. У результаті, як вже було сказано, формується HTML-сторінка, яка і відправляється клієнту. Виходить, що результат роботи Web-програми ідентичний результату запиту до традиційного Web-сайту, проте, на відміну від нього, Web-додаток генерує HTML-код залежно від запиту користувача, а не просто передає його клієнту в тому вигляді, в якому цей код зберігається у файлі на стороні сервера. Тобто, Web-додаток динамічно формує відповідь за допомогою виконуваного коду, так званої виконуваної частини.

За рахунок наявності виконуваної частини, Web-програми здатні виконувати практично ті ж операції, що й звичайні Windows-додатки, з тим лише обмеженням, що код виконується на сервері, в якості інтерфейсу системи виступає браузер, а в якості середовища, за допомогою якого відбувається обмін даними – Інтернет. До найбільш типових операцій, що виконуються Web-додатками відносяться:

- прийом даних від користувача і збереження їх на сервері;
- виконання різних дій за запитом користувача: вилучення даних з бази даних (БД), додавання, видалення, зміна даних в БД, проведення складних обчислень;
- автентифікувати користувача і відображати інтерфейс системи, відповідний даному користувачеві;
- відображати оперативну постійно змінюючу інформацію і т.д.

29.2. Короткий опис архітектури ASP.NET і .NET Framework

ASP.NET – це платформа для створення Web-додатків і Web-сервісів, що працюють під управлінням IIS. Сьогодні існують інші технології, що дозволяють створювати Web-додатки. До них відносяться перш за все, дуже популярні сьогодні мови PHP і PERL, старіша і менш популярна сьогодні технологія CGI і т.д. Однак, ASP.NET відрізняється від них високим ступенем інтеграції з серверними продуктами, а також з інструментами Microsoft для розробки, доступу до даних і забезпечення безпеки. Крім того, використання ASP.NET дозволяє розробляти Web і Windows-додатки, використовуючи дуже схожі технологічні ланцюжки, однакові мови програмування, технології доступу до даних і т.д. Більш того, базові мови програмування, за допомогою яких сьогодні можлива розробка Web-додатків, є повністю об’єктно-

орієнтованими, що робить розробку здійсненної частини, а також її модифікацію, обслуговування, налагодження і повторне використання набагато більш простим заняттям, ніж в інших технологіях. Існує досить великий перелік сильних сторін використання ASP.NET для створення складних Web додатків.

Зауважимо лише, що ASP.NET функціонує виключно на серверах Windows, так як вимагає наявності IIS. Для створення Web-додатків, що не вимагають IIS, а використовують, наприклад, Web сервер Apache, і працюють на серверах під управлінням операційних систем, відмінних від Windows, застосовуються інші технології.

Важливим моментом у розумінні архітектури ASP.NET є той факт, що вона є частиною інфраструктури .NET Framework.

Платформа .NET Framework є надбудовою над операційною системою, в якості якої може виступати будь-яка версія Windows. На сьогоднішній день платформа. NET Framework включає в себе:

- чотири офіційні мови: C#, VB.NET, Managed C++ (керований C++) и JScript .NET;
- об'єктно-орієнтоване середовище CLR (Common Language Runtime), спільно використовуване цими мовами для створення додатків під Windows і для Internet;
- ряд пов'язаних між собою бібліотек класів під загальним ім'ям FCL (Framework Class Library).

Відносини з концептуальної точки зору архітектурних компонентів платформи. NET Framework представлені на рис. 29.2.

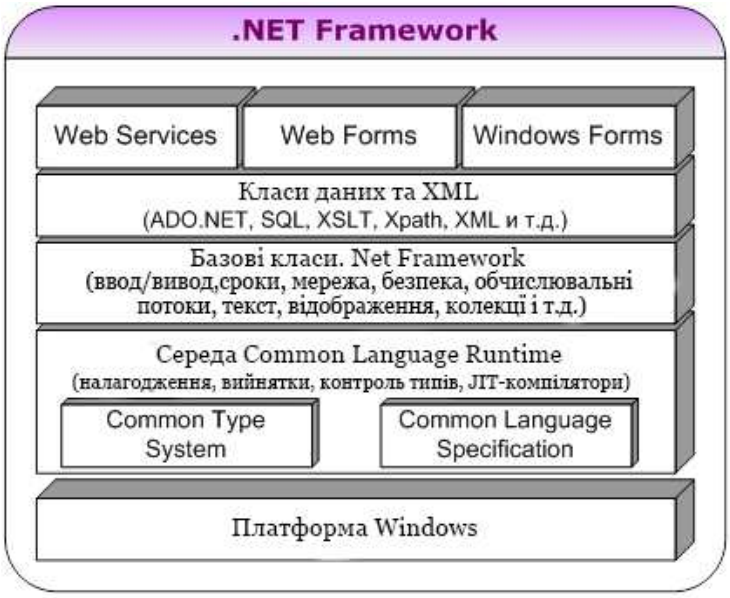

Рисунок 29.2 – Архітектура .NET Framework

Кожний Web-додаток, що розробляється на основі ASP.NET, складається з інформаційної частини, програмного коду, і відомостей про конфігурацію.

Інформаційна частина містить статичні і динамічні елементи сторінки і реалізуються у вигляді Web-форм. Статичні елементи являють собою типові

елементи мови HTML, динамічні же компонуються програмним кодом програми під час його виконання (наприклад, запити до бази даних).

Програмний код реалізує логіку, певну в процедурах обробки даних, що визначають реакцію програми на запити користувача. Програмний код виповнюється сервером і взаємодіє з динамічними елементами інформаційної частини для формування відгуку програми.

Відомості про конфігурацію являють собою файли, що містять параметри, що визначають спосіб виконання програми на сервері, параметри безпеки, реакцію програми на виникаючі помилки і т.д.

Основним елементом Web-програми є Web-форма (або Web-сторінка), яка з одного боку схожа на Windows форму, тому що дозволяє розміщувати всередині себе різні елементи управління, здатні відображати дані і реагувати на дії користувача, а з іншого – являє собою HTML сторінку, тому що містить всі її атрибути. Описи елементів управління, згаданих раніше представляються в коді HTML сторінки у вигляді спеціальних тегів.

У другому випадку кожна Web-сторінка розділяється на дві частини: Web-форму і файл, що містить програмний код. При цьому, форма, як і в першому випадку, зберігається у файлі з розширенням .aspx, а програмний код – з розширенням .cs. Така модель забезпечує кращу організацію елементів Web-програми за рахунок відділення для користувача інтерфейсу від програмної логіки.

Вивчивши цей приклад, можна описати типовий сценарій взаємодії елементів Web програми один з одним і з клієнтом, що здійснює запит форми цього додатка (рис. 29.3).

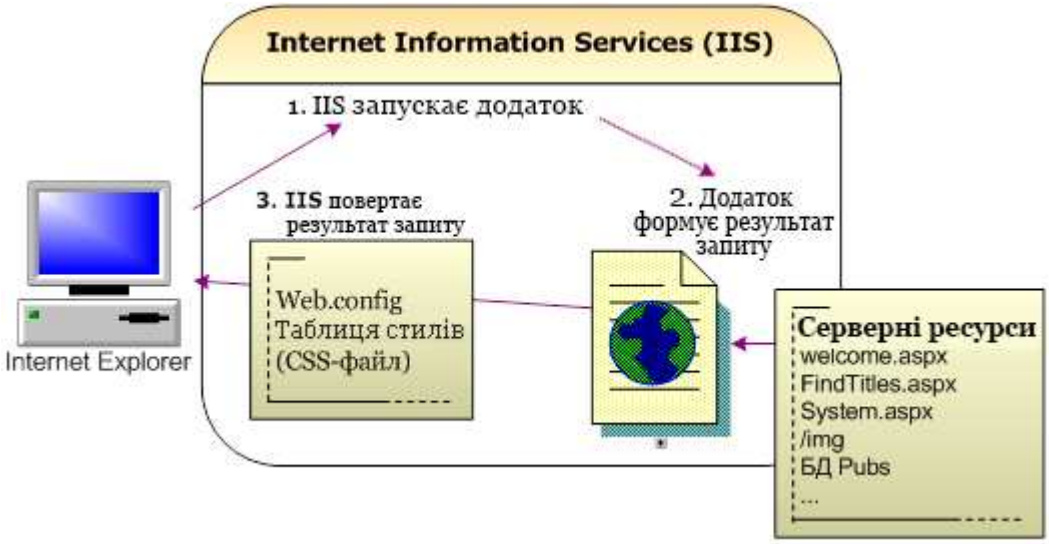

Рисунок 29.3 – Типовий сценарій взаємодії елементів Web-програми з клієнтом

Як видно з рис. 29.3, під час звернення клієнта до Web-додатка, останній запускається на сервері IIS. Запущене застосування формує відгук. Для цього на сервері створюється екземпляр запитаної Web-форми, яка генерує HTML-текст відгуку, який і передається браузеру клієнта. Відразу після цього примірник Web-форми знищується. Користувач, отримавши HTML-сторінку, згенеровану додатком, має можливість заповнювати різні поля фо-

рми (тестові поля, перемикачі тощо). Після заповнення всіх необхідних полів форми, користувач ініціює відправку даних, введених ним в сторінку назад на сервер. Це відбувається за рахунок використання технології зворотного відсилання, яка викликається при виконанні певних дій (наприклад, натискання на кнопку). Отримавши дані від користувача, сервер створює новий екземпляр Web-форми, заповнює його отриманими даними і обробляє всі необхідні події. Після закінчення обробки, сервер формує HTML-код відповіді і відправляє його клієнту, а потім знищує примірник Web-форми. Більш докладно описаний сценарій зображений на рис. 29.4 і 29.5.

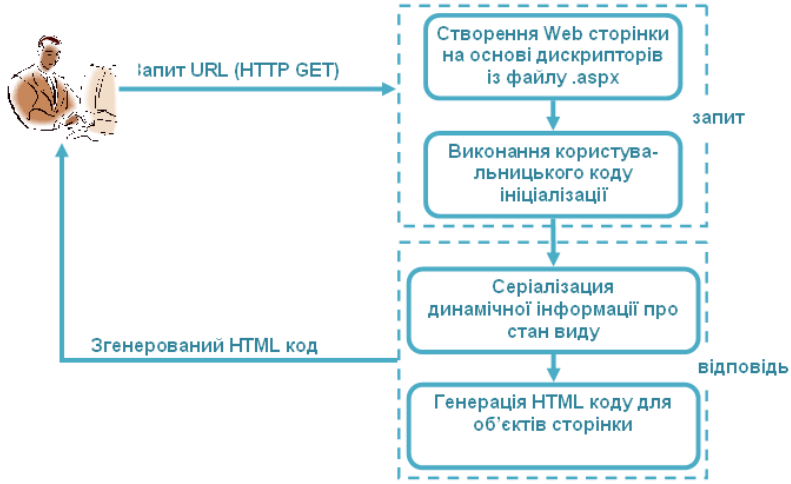

Рисунок 29.4 – Докладний сценарій взаємодії елементів Web-додатків з клієнтом при першому запиті

Рисунок 29.5 – Докладний сценарій взаємодії елементів Web-додатків з клієнтом при запиті зворотного відсилання

У момент закінчення роботи з Web-додатком, користувач або закриває браузер, або переходить на іншу інтернет сторінку. У цей момент завершується сеанс роботи користувача з даним додатком, проте сам додаток може бути завершено сервером не відразу після закінчення останнього сеансу роботи користувача. Це пов'язано з управлінням розподілом пам'яті платформою. NET Framework, яка заснована на періодичній перевірці посилань об'єктів. Якщо в результаті такої перевірки виявиться, що об'єкт більше не використовується, сервер знищує його, звільняючи таки чином займану їм пам'ять. Таким чином, не можна точно сказати коли саме настане подія Application_End для даного Web-додатку.

Такий принцип організації виконання Web-додатків добре підходить для масштабованих додатків з інтенсивним мережевим обміном. Однак у нього є і недоліки. Зокрема, виявляється неможливим зберігати дані, що належать формі навіть протягом роботи користувача з додатком. Тобто якщо ми захочемо створити якусь змінну, що зберігає, наприклад ідентифікатор замовлення, з яким ми в даний момент працюємо, зробити це буде неможливо, тому що форма після відправки клієнтові відразу ж знищується. Щоб обійти цей недолік, ASP.NET використовує спеціальний механізм для збереження даних, введених в елементи управління Web-форми. Згідно з цим принципом, в рамках кожного запиту, на сервер відправляються всі дані, які були введені в елементи управління. При цьому, як уже згадувалося вище, на сервері виникає подія Page_Init, метою якої є створення Web-форми і її ініціалізація. У процесі ініціалізації, в елементи управління створеної форми записуються передані від клієнта дані. Тепер ці дані стають доступні додатком допомогою обробки події Page_Load, що виникає при кожному зверненні до сторінки.

Взагалі, для кращого розуміння процесу взаємодії користувача з Web-додатком, розглянемо послідовність подій, що виникає при зверненні клієнта до сторінки додатка.

Отже, при запиті сторінки, перш за все, ініціюється подія Page_Init, яка виробляє початкову ініціалізацію сторінки та її об'єкта. У рамках обробки даної події програміст може розмістити код, який здійснює початкову ініціалізацію сторінки. Тим не менш, цю подію не можна використовувати для ініціалізації елементів управління, розміщених на сторінці, тому що вони ще не створені.

Після цього ініціюється подія Page_Load. Більшість Web-сторінок використовують цю подію для виконання ініціалізації, наприклад заповнення полів даними, установка початкових значень для елементів управління і т.д. Крім того, в процедурі обробки даної події можливе визначення того чи була завантажена сторінка вперше, або звернення до неї здійснюється повторно в рамках технології зворотного відсилання, яка сталася внаслідок натиснення користувачем кнопки, або іншого елемента управління, розміщеного на сторінці. В англійській термінології зворотне відсилання даних на сервер називається post back. Для визначення поточного стану сторінки необхідно перевірити властивість Page.IsPostBack, яка буде мати значення false при першому запуску сторінки. Визначення того, чи виробляється перше звернення до даної сторінки, або повторне важливо, оскільки дозволяє проводити ініціалі-

зацію тільки в тому випадку, коли сторінка запитується вперше. Так, наприклад, при зворотної відсилання даних на сервер не тільки немає необхідності проводити ініціалізацію, встановлюючи початкові значення елементів управління, це може бути помилкою, тому що ці елементи управління повинні отримати значення, передані їм від користувача. Надалі, у разі, якщо для сторінки було проведено зворотнє відсилання, викликаються події елементів управління, розміщені на сторінці. Ці події запам'ятовуються в той момент, коли користувач виробляв дії з елементами управління у вікні браузера, а при передачі даних на сервер виконуються по порядку.

Після виклику події Page_Load відбувається так звана перевірка достовірності сторінки. Необхідність такої перевірки виникає тоді, коли користувач ввів в елементи управління, розташовані на сторінці дані, які згодом необхідно зберегти або використовувати для обробки. В ідеалі перевірка достовірності повинна відбуватися на стороні клієнта для того, щоб користувач був проінформований про проблеми з введенням даних перед їх відправкою на сервер, оскільки це дозволяє зменшити обсяг інформації, що передається по мережі і прискорити процес обміну даними з сервером. Однак, незалежно від того, чи була проведена перевірка достовірності даних на стороні клієнта чи ні, її необхідно здійснювати і на стороні сервера. Здійснення перевірки достовірності досить складне завдання. Складність ця обумовлена відмінністю моделей клієнтського і серверного програмування. У ASP.NET існує кілька елементів управління перевіркою достовірності. Ці елементи управління виконують автоматичну клієнтську і серверну перевірку достовірності. У разі, якщо перевірка достовірності виявила помилки у введених даних, ASP.NET повідомить про це користувача і не дозволить здійснювати подальшу роботу зі сторінкою до усунення помилок.

Наступним кроком обробки Web-форми є обробка всіх подій, ініційованих користувачем з моменту останньої зворотного відсилання. Для ілюстрації цього, наведемо приклад.

Нехай у нас існує сторінка з кнопкою (Button) „Відправити" і текстовим полем (TextBox) без автоматичного зворотного посилання. При зміні тексту в текстовому полі і натисканні на кнопці „Відправити" ініціюється зворотна відправка даних сторінки на сервер (цього не сталося при зміні тексту в текстовому полі, тому що відповідна опція цього елемента управління AutoPostBack встановлена в false). У момент зворотної відправки сторінки на сервер ASP.NET запускає наступні події:

Page.Init
Page.Load
TextBox.TextChanged
Button.Click
Page.PreRender
Page.Unload

У результаті обробки всіх ініційованих подій, генерується HTML-код сторінки, який і передається клієнту, після чого виконується Очищення, в рамках якої, ініціюється подія Page_Unload. Вона призначена для звільнення

ресурсів, зайнятих даною сторінкою. Подія Page.PreRender ініціюється після того, як сервер обробив всі події сторінки, але генерація її HTML-коду ще не відбулася. Зазвичай це подія використовується ASP.NET для прив'язки елементів управління до джерел даних безпосередньо перед створенням HTML-коду і відправкою його клієнту.

Описана послідовність подій дозволяє створити опис життєвого циклу Web-сторінки, що зображений на рис. 29.6.

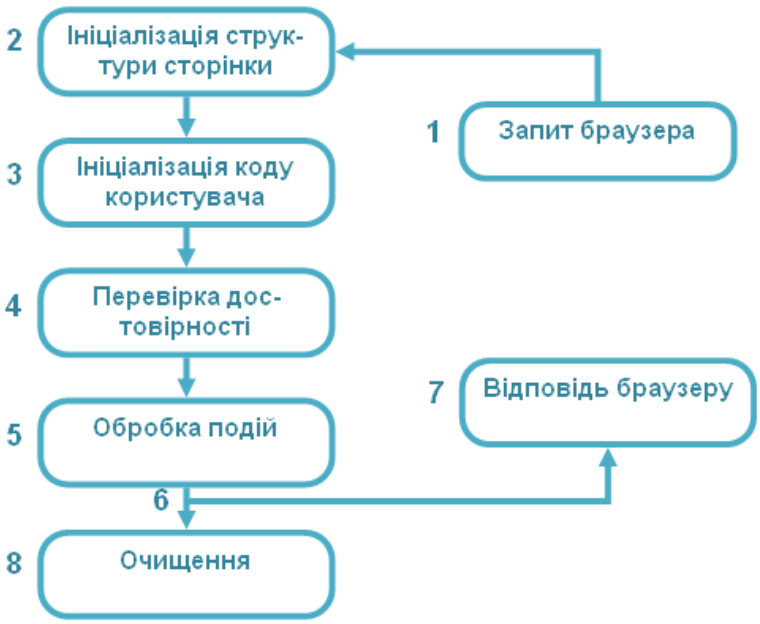

Рисунок 29.6 – Життєвий цикл сторінки ASP.NET

Файл Default.aspx.cs містить програмний код, прив'язаний до сторінки. Організація цього файлу практично повністю повторює організацію аналогічного файлу для проекту Windows-додатка.

Файл починається з підключення різних просторів імен, що містять описи тих класів .NET Framework, які необхідно використовувати в даному модулі. Після чого йде опис класу сторінки, що складається з різних функцій, в тому числі прив'язаних до обробки подій даної форми. За замовчуванням, створена заготівля функції – обробника події відкриття сторінки Page_Load.

Принцип розробки додатку в ASP.NET повністю відповідає об'єктно-орієнтованому підходу. Програміст в процесі створення Web-програми оперує класами, визначаючи їх атрибути та значення, а також методи, призначені для виконання об'єктами класу дій, прив'язаних до подій сторінки.

29.3. Вікно Solution Explorer. Структура ASP.NET-додатків

Visual Studio впорядковує додатки за допомогою проектів і рішень.

Проект (project) – це набір файлів, з яких у підсумку компонується виконуваний файл. Рішення (solution) – це група проектів, що утворюють функціональну одиницю. Файли, складові рішення, можна переглядати у вікні Solution Explorer, показаному на рис. 29.7.

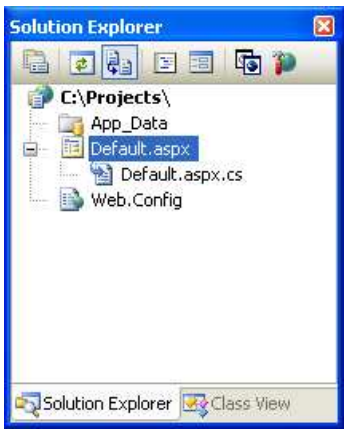

Рисунок 29.7 – Вікно Solution Explorer

Стартовий проект (start-up project) запускається першим по клацанню кнопки Start в Visual Sludio. Якщо рішення складається з декількох проектів, то стартовий проект зазвичай викликає інші проекти цього рішення.

Відомості про рішення зберігаються у файлі рішення, який має розширення. Sln і за замовчуванням розміщується в папці Мої документи. Ці файли дозволяють відкривати рішення, а файли проектів (файли з розширенням .vbproj або .csproj) дозволяють безпосередньо відкривати проекти, що розташовані у відповідних папках. При збереженні проекту, відкритого таким чином, Visual Studio створить новий файл рішення.

Розглянемо структуру створеного проекту, відображеного у вікні Solution Explorer.

Після створення нового проекту, Visual Studio створює наступні файли: Default.aspx і Default.aspx.cs. Крім того, автоматично створюється каталог App_Data, призначений для зберігання файлів бази даних, використовуваної ASP.NET. Це може бути БД Access, SQL Server, XML або файл будь-якого іншого формату доступного в ASP.NET додатку. Крім каталогу App_Data в проекті Web-програми можуть створюватися і інші каталоги. Частина з них грає зумовлене значення і займає особливе місце в системі, їх імена зумовлені і за ними закріплені певні функції. Такі каталоги часто називають віртуальними каталогами додатку. Іншу частину складають каталоги, що створюються самими користувачами. За такими каталогами не закріплені певні функції, а їх призначення в системі визначає сам користувач. Файл Default.aspx являє собою файл, що містить опис Web-сторінки в форматі HTML.

Таким чином, згідно такої моделі організації проекту, ASP.NET дотримується принцип розділення вихідного коду і інтерфейсу системи. Інтерфейс системи описується в файлах з розширенням .Aspx, а вихідний код розміщується в файлах з розширенням .cs, у разі, якщо в якості мови програмування використовується C#, і .vb – якщо Visual Basic.

Ще одним важливим файлом проекту ASP.NET-додатку є файл Web.Config. При початковому створенні проекту, Visual Studio не створює цей файл. Однак, при першому запуску програми (за допомогою натискання клавіші F5), користувачеві пропонується створити цей файл і встановити в ньому опцію, роздільну налагодження даного додатка, як було показано вище.

Файл Web.Config є конфігураційним файлом, створеному на базі формату XML. Фрагмент такого файлу, зображений на рисрисунок 29.. Він містить велику кількість параметрів настройки на рівні додатку, які конфігурують всі аспекти, починаючи з безпеки, і закінчуючи налагодженням, підключенням до джерел даних та управлінням стану.

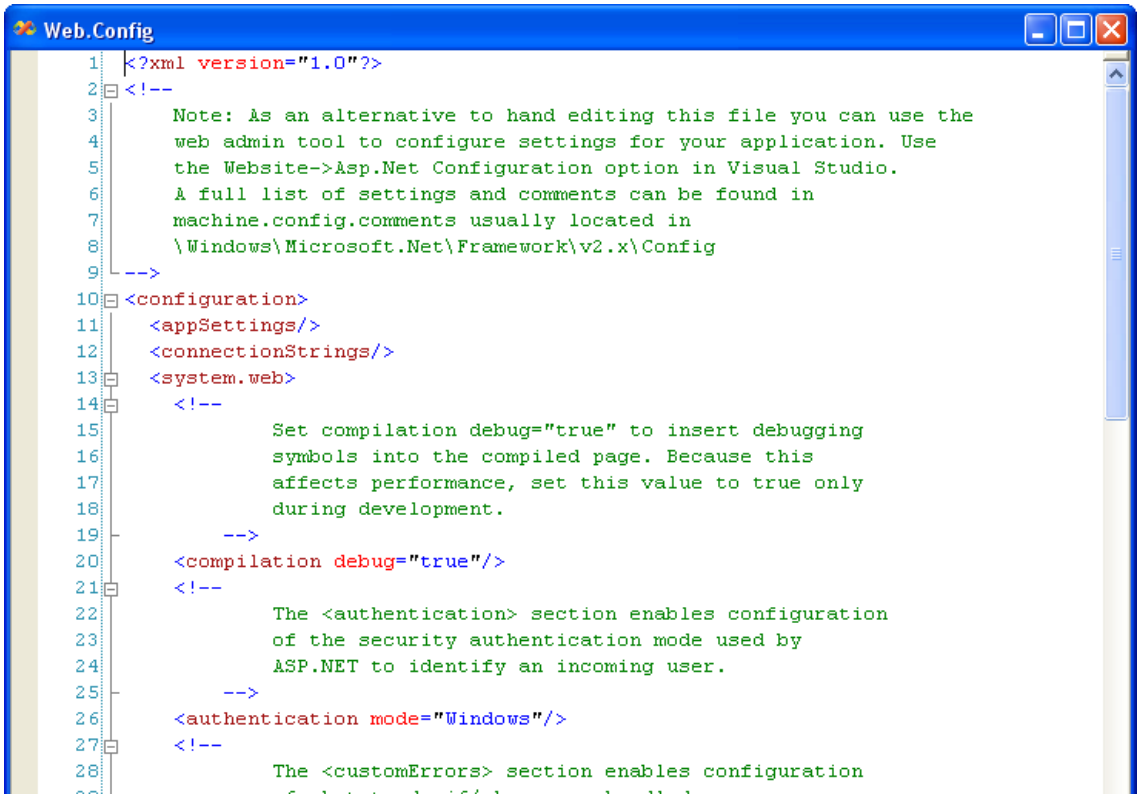

Рисунок 29.8 – Приклад файлу Web.Config.

Однак, крім перерахованих вище файлів, додаток ASP.NET може містити й інші елементи, що грають свою роль і виконують свої завдання. Всі ці елементи можуть бути додані в проект по ходу його розвитку. Для більш глибокого розуміння принципів організації та створення Web-програми на основі ASP.NET, розглянемо деякі з них більш докладно.

29.4. Перший проект

У проекті буде створена сторінка default.aspx. Виберіть її, і з’явиться вікно редагування із закладками Design і Source. Не змінюючи нічого, клацніть на кнопці зі стрілкою, щоб переглянути сторінку в браузері. З’явиться вікно, в якому запитується, чи потрібно додати в файл web.config можливість налагодження. Натисніть "OK". На панелі завдань повинен з’явитися значок Web-сервера. Відкриється браузер, що показує сторінку за адресою http://localhost:номер_порта/Website1/default.aspx. "Localhost" позначає сервер, що працює на вашому комп’ютері. Поки що сторінка в браузері порожня. Але вихідний код цієї сторінки не порожній. Програма згенерувала код для вас.

<%@ Page Language="C#" AutoEventWireup="true";
CodeFile="Default.aspx.cs" Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN";

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
<title>Untitled Page</title>
</head>
<body>
<form id="form1" runat="server">
* *
* *
</form>
</body>
</html>

Розберемо цю сторінку.

<%@ Page Language="C#" %>. Тег <% завжди призначається для інтерпретації ASP-коду. Директива Page завжди присутня на сторінці aspx. Її атрибут Language – це вказівка, що в скриптах даної сторінки використовуватиметься C #, а могли б VB, C + + або J #. CodeFile – ім'я файлу з відокремленим кодом (code-behind). Inherits – клас, певний в тому файлі, від якого успадковується клас сторінки.

Одночасно буде створений файл Default.aspx.cs.

Це технологія розділення коду, про яку ми вже говорили. Сама форма знаходиться у файлі Default.aspx, а у файлі Default.aspx.cs знаходиться клас сторінки на мові C#. Таким чином, дизайн сторінки може бути змінений не зачіпаючи код сторінки, що дозволяє розділити відповідальність за зовнішній вигляд і роботу сторінки між дизайнером і програмістом.

<form runat="server">

Цей тег дає вказівку компілятору обробляти елементи управління сторінки. Зверніть увагу на те, що даний тег має властивість runat, для якого встановлено значення "server" (інших значень не буває). При використанні цієї властивості елемент управління обробляється компілятором, а не передається браузеру "як є".

Вставте в Default.aspx між тегами <form> і </form> тег, що задає елемент управління.

<asp:Label id="Time" runat="server"
Text="Поточний час: "
/>

Серверний елемент управління Label є засобом розміщення на сторінці тексту, який може містити теги HTML. Змінюючи значення властивостей цього елемента керування в коді, ви можете динамічно змінювати текст на сторінці. У asp: Label компілятору повідомляється, з яким об'єктом ведеться робота (у розглянутому випадку – з елементом управління Label).

Далі задаються різні властивості елемента керування. У першу чергу визначається його ім'я id = "Time" і атрибут "runat", а також текст.

В файлі Default.aspx.cs повинен міститися такий текст:

using System;

```
.....
public partial class _Default : System.Web.UI.Page
{
   protected void Page_Load(object sender, EventArgs e)
   {
    }
}
```

Ключове слово partial з'явилося в C# 2.0, і дозволяє розбити текст визначення класу між різними файлами. System.Web.UI.Page – клас, базовий для всіх сторінок ASP.NET. Отримане значення привласнюється властивості Text об'єкта Time. Це елемент управління типу Label (мітка), який ми вставили. Час на годиннику клієнта і сервера може не збігатися, якщо вони знаходяться в різних точках земної кулі. Page_Load схожий на звичайний обробник події форми. Як можна легко здогадатися, ця функція викликається кожного разу, коли завантажується форма. Запустіть сторінку на перегляд кнопкою F5 або натиснувши на кнопку зі стрілкою на панелі інструментів. У браузері повинна відкритися сторінка, на якій буде написано поточний час. Відкрийте вихідний текст сторінки. Ніякого коду на C # або елементів управління ASP.NET там не буде:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head><title>Час, вперед</title>
</head>
<body>
<form name="form1" method="post" action="Default.aspx"
id="form1">
<div>
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
value="/wEPDwUJODExMDE5NzY5D2QWAgIDD2QWAgIBDw8WAh4EVGV4
dAUSMDguMDYuMjAwNiA0OjU2OjQ3ZGRkkEMgqXmKC0v9vwAwh999lefuIOw=" />
</div>
<div>
   <span id="Time">Поточний час:
08.06.2006 4:56:47</span>
  </div>
  </form>
</body>
</html>
```

Обновіть сторінку. Ви побачите нове значення часу.

29.5. Властивості сторінки

Сторінка – це основа всього в web-додатку. Клас System.Web.UI.Page інкапсулює функціональність, необхідну для створення й обробки сторінок ASP.NET. Кожна сторінка ASP.NET – це об'єкт класу, який автоматично ге-

нерується ядром ASP.NET. Клас успадковується від асоційованого зі сторінкою класу, якщо ми використовуємо відокремлений код, або прямо успадковується від System.Web.UI.Page, якщо код на C# вбудований в сторінку. Середовище також створює конструктор за замовчуванням.

Щоб переконатись в цьому, можемо створити сторінку "PageType.aspx":

```
<%@ Page Language="C#" %>
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
   <title>Тип сторінки</title>
</head>
<body>
   <div>
   <% Response.Output.Write("Тип даної сторінки
{0}",this.GetType()); %>
   </div>
    <div>
   <% Response.Output.Write("Базовий тип даної сторінки
{0}",this.GetType().BaseType); %>
   </div>
</body>
</html>
```

Ось результат (рис. 29.9):

Тип даної сторінки – ASP.pagetype_aspx. Базовий тип даної сторінки — System.Web.UI.Page

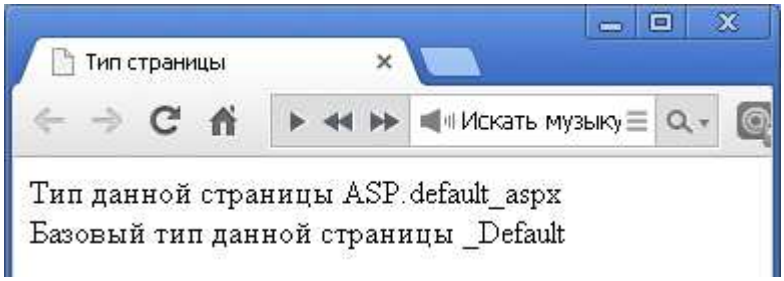

Рисунок 29.9 – Така ж сторінка, створена за технологією розділення коду

До поточного об’єкту сторінки можна звертатися як до змінної Page. Page – це контейнер елементів управління даної сторінки, тому містить в собі колекцію Controls. Якщо в тезі <head> присутній атрибут runat = "server", то в Page міститься і поле Header, через яке можна управляти заголовком сторінки – наприклад, поміняти назву сторінки в заголовку браузера, призначити файл каскадних таблиць стилів:

```
<script runat="server">
```

У сторінки є дві важливі властивості – Response і Request. Властивість Response має тип HttpResponse. Response сторінки можна сприймати як вихідний потік. Весь HTML-код генерується сторінки в принципі може бути виведений через запис в цей потік. Це був звичайний спосіб роботи розробників asp. Функція Response.Redirect перенаправляє браузер на іншу сторінку:

Response.Redirect ("NavigationTarget.aspx? Name =" +
System.Web.HttpUtility.UrlEncode (Name.Text));

Тут формується командний рядок з параметрами QueryString, які цільова сторінка може прочитати.

Аналогічно, властивість Request – це запит, переданий на сервер для виведення потрібної сторінки. Він має тип HttpRequest. У ньому зберігається все про клієнта, включаючи налаштування його браузера, файли-cookie і дані, введені ним в форму:

NameLabel.Text = Server.HtmlEncode(Request.QueryString["Name"]);

29.6. Події сторінки

Робота середовища ASP. NET зі сторінкою починається з отримання і обробки web-сервером IIS-запиту до даної сторінки і передачі цього запиту середовищі виконання ASP. NET. Середовище виконання аналізує, чи потрібно компілювати сторінку або можна видати як відповідь сторінку з кеша.

Потім починається життєвий цикл сторінки. Він починається з етапу PreInit. Після отримання запиту середовище виконання завантажує клас спричиненої сторінки, встановлює властивості класу сторінки, вибудовує дерево елементів, заповнює властивості Request, Response і UICulture і викликає метод IHttpHandler.ProcessRequest. Після цього середовище виконання перевіряє, яким чином була викликана ця сторінка, і якщо сторінка викликана шляхом передачі даних з іншої сторінки, про що буде розказано далі, то середу виконання встановлює властивість PreviousPage.

На цьому етапі встановлюється також властивість IsPostback об’єкта Page, яка дозволяє дізнатися, чи в перший раз завантажується форма або вона повинна формуватися як результат обробки даних, введених користувачем. У обробниках подій сторінки можна перевірити цю властивість:

```
if (!Page.IsPostBack)
   {
      // обробляти
   }
```

Далі відбувається ініціалізація сторінки – подія Init. Під час ініціалізації створюються дочірні користувача елементи управління і їм встановлюються властивості id. У цей же час до сторінці застосовуються теми оформлення. Якщо сторінка викликана в результаті постбека, то на цьому етапі дані, відправлені на сервер, ще не завантажені в властивості елементів управління. Програміст може ініціалізувати їх властивості.

Завантаження. Під час події Load встановлюються властивості елементів управління на підставі інформації про стан, якщо сторінка створюється внаслідок відправки даних форми.

Якщо на сторінці існують валідатори, то для них викликається метод Validate(). Потім викликаються обробники подій (за умови, що сторінка генерується у відповідь на дії користувача).

У методі Render генерується сам HTML-код виведеної сторінки. При цьому сторінка викликає відповідні методи дочірніх елементів, а ті в свою чергу – методи своїх дочірніх елементів.

У методі Render код виводиться в Response.OutputStream. Сама сторінка теж вважається елементом управління – клас Page є спадкоємцем класу Control. Якщо на сторінці є блоки відображення, вони стають частиною функції профарбовки (rendering) .

Нарешті, сторінка вивантажується з пам’яті сервера і відбувається подія Unload.

Під час життєвого циклу сторінки відбуваються різні події (рис. 29.10). Можна включити трасування сторінки, щоб подивитися порядок їх виникнення:

<%@ Page Language="C#" Trace ="true" TraceMode="SortByTime" %>

Trace Information

Category	Message	From First(s)	From Last(s)
aspx.page	Begin PreInit		
aspx.page	End PreInit	1,14336933884055	1,143369
aspx.page	Begin Init	1,14477789774957	0,001409
aspx.page	End Init	1,20637455319042	0,061597
aspx.page	Begin InitComplete	1,20649412145957	0,000120
aspx.page	End InitComplete	1,20710006439366	0,000606
aspx.page	Begin PreLoad	1,20798704863328	0,000887
aspx.page	End PreLoad	1,21330616041983	0,005319
aspx.page	Begin Load	1,21343271281685	0,000127
aspx.page	End Load	1,26233529680448	0,048903
aspx.page	Begin LoadComplete	1,26243614761094	0,000101
aspx.page	End LoadComplete	1,26813100547695	0,005695
aspx.page	Begin PreRender	1,26871404047162	0,000583
aspx.page	End PreRender	5,70876996936761	4,440056
aspx.page	Begin PreRenderComplete	5,70993101078489	0,001161
aspx.page	End PreRenderComplete	5,71078558867119	0,000855
aspx.page	Begin SaveState	5,90122683190182	0,190441
aspx.page	End SaveState	6,01436635103065	0,113140
aspx.page	Begin SaveStateComplete	6,01451162089036	0,000145
aspx.page	End SaveStateComplete	6,01518740510316	0,000676
aspx.page	Begin Render	6,01525976066791	0,000072
aspx.page	End Render	6,78199296279276	0,766733

Рисунок 29.10 – Трасування додатку

Під час трасування не виводиться подія Unload, тому що воно відбувається, коли весь код вже виведений. Під час обробки цієї події потрібно звільнити ресурси, наприклад, з’єднання з базами даних або відкриті файли.

Повний список подій сторінки, які можна перевизначити в класі сторінки:

1. *PreInit*
2. *Init*
3. *InitComplete*
4. *PreLoad*
5. *Load*
6. *LoadComplete*
7. *PreRender*
8. *PreRenderComplete*
9. *Unload*

Для всіх подій визначені обробники – віртуальні функції OnInit, OnLoad. Коли AutoEventWireup одно true, в класі автоматично оголошуються функції-обробники подій з префіксом Page – Page_Load, Page_Init і так далі. Одне з найпопулярніших подій – це Page_Load. Генеруючи нову сторінку,

Visual Studio створює обробник цієї події. Тут можна змінити зовнішній вигляд елементів і задати нові. Можна встановити AutoEventWireup в false. У такому випадку треба писати перевантажені версії віртуальних функцій:

protected override void OnInit(EventArgs e)

{ }

Так можна добитися прискорення роботи сторінки.

29.7. Способи впровадження коду ASP. NET в сторінку

Є три способи впровадити код на програмному мовою в сторінку aspx. Блок <script runat="server"> </script> називається блоком оголошення коду. Тег <script> схожий на той, яким запроваджуються скрипти JavaScript, але з великим відмінністю – скрипт виконується на сервері. Тому необхідно задавати атрибут runat = "server". Атрибут language у нього може приймати значення C#, VB, J#. У сторінках з відокремленим кодом можна писати і на C++. За замовчуванням приймається значення мови Visual Basic, тому не треба забувати вказувати мову, коли пишете на C#. Але немає необхідності це робити, якщо мова визначена в директиві Page. Можна писати на різних мовах в одному додатку, але не можна змішувати різні мови на одній сторінці.

Усередині блоку можна оголошувати змінні, константи і функції. Насправді в C# немає глобальних змінних, так що це будуть члени класу сторінки. Але вони виглядають глобальними, тому що клас не описаний програмістом, його генерує ASP.NET. Тому будемо називати їх змінними сторінками. Тут можна перевизначити віртуальні методи класу сторінки. У блоці також можна оголошувати класи, але вони будуть внутрішніми по відношенню до класу сторінки.

29.8. Блоки відображення

Будь-який код, впроваджений за допомогою <% і %>, обробляється під час події Render як частина сторінки. У тілі блоку <% %> допустимо оголошувати змінні (тоді вони будуть локальними для того методу, в якому даний блок буде реалізований), але не можна оголошувати методи або типи.

Більш сучасний спосіб – використання серверних елементів управління. Вони описані в тілі сторінки подібно до звичайних елементам розмітки, є членами класу сторінки. До них можливе звернення через ідентифікатор. Наприклад, замість того щоб виводити текст через Response.Write, можна встановити текст елемента керування. Об’єкт будь-якого класу створюється за допомогою синтаксису „тег об’єкта”:

<object id="items" class="System.Collections.ArrayList"

runat="server"/>

Контрольні питання

1. Які бувають найбільш типові операції, що виконуються Web-додатком?
2. Що таке ASP.NET?
3. Які переваги має платформа ASP.NET?
4. З чого складається Web-додаток?

5. Яка подія ініціюється перш за все при запиті сторінки?
6. Для чого використовується подія Page.Load?
7. Які події ASP.NET запускає у момент зворотної відправки сторінки на сервер?
8. Що містять в собі файли з розширенням .aspx?
9. Як називається файл параметрів настройки на рівні додатку?
10. Яка подія відповідає за поточний стан сторінки Web-додатку?

Тема 30. Серверні елементи управління

Одна з найважливіших задач web-розробника – отримання та обробка даних, введених користувачем. Інформація надсилається серверу через форму. Форма містить елементи управління, які дозволяють різними способами вводити інформацію.

Форми застосовуються в більшості сайтів. Наприклад, якщо ви пишете лист в web-інтерфейсі, з'являється форма з текстовими полями, відповідними адресату, темі і тексту листа. Натисканням на кнопку можна додати доданий файл і остаточно надіслати листа кнопкою Send.

Форма HTML містить теги, такі як текстове поле, що випадає, перемикачі (radiobuttons) і прапорці (checkbox), кнопки.

Форми ASP. NET відрізняються від звичайних форм наявністю властивості runat = "server". Вони обробляються ASP.NET на сервері. Форми є одним з полів сторінки. На сторінці знаходяться елементи управління. Багато хто з них повинні бути розташовані всередині форми. ASP.NET дозволяє запам'ятовувати стан елементів управління, тобто текст, який був введений в текстове поле, або вибраний перемикач, передавати його на сервер і назад на сторінку після її поновлення:

<form ID="FormVote" runat="server"></form>

Всі форми обробляються методом POST. Атрибут Action можна задавати, але не обов'язково. За умовчанням це поточна сторінка.

У елементів управління ASP.NET теж є властивість runat = "server". Другий обов'язковий атрибут – це його ідентифікатор, або ID. Він потрібен, щоб звертатися до елементу в програмі, тобто це ім'я члена сторінки, за яким ми можемо його ідентифікувати.

Перерахуємо групи елементів управління:

- Елементи управління HTML;
- Стандартні або серверні елементи управління;
- Елементи перевірки даних;
- Елементи-джерела та елементи-споживачі даних;
- Елементи навігації по сайту;
- Елементи логінів та паролів;
- Користувацькі елементи.

Елементи управління HTML є спадкоємцями класу System.Web.UI.-HtmlControls.HtmlControl. Вони безпосередньо відображаються у вигляді елементів розмітки HTML. Їх відображення не залежить від типу браузера. Властивості таких елементів повністю відповідають атрибутам тегів HTML.

Порівняйте звичайний тег

<input id="Reset1" type="reset" value="reset" />

З елементами управління HTML

<input id="Reset1" runat="server" type="reset" value="reset" />

Різниця полягає тільки в наявності атрибута runat = "server". Але він дає колосальну різницю. Теги сервер відображає як є, а елементом управління можна маніпулювати в коді. Тільки в іншому випадку у функції-методі сторінки можна написати

Reset1.Value = "АСП";

Що рівносильно

this.Reset1.Value = "АСП";

Отже, Reset1 стає одним з членів класу сторінки. Ці класи використовуються, якщо необхідно отримати певні теги HTML або якщо потрібно конвертувати старі сторінки asp. Елементи управління HTML можна розміщувати на одній сторінці упереміш з серверними елементами.

Стандартні або серверні елементи управління потужніщі, тому що вони прив'язані не до розмітки, а до функціональності, яку потрібно забезпечити. Багато елементів не мають аналогів в HTML, наприклад, календар. Їх профарбовка повністю контролюється ASP.NET. Перехоплюючи події PreRender, Init, Load, можна втрутитися в цей процес. Оголошення серверного елемента управління починаються з блоку <asp:тип> і закінчуються </asp:тип>

Наприклад:

<asp:Label ID="Label1" runat="server" Text="Hello World"></asp:Label>

Можливо також закрити оголошення тегом />, якщо всередині блоку немає тексту:

<asp:Label ID="Label1" Runat="server" Text="Hello World" />

Властивості цих елементів строго типізовані, на відміну від HTML-елементів. У таблиці 30.1 наведено елементи управління, які мають пару серед тегів HTML. Взагалі їх набагато більше. Деякі елементи генерують не тільки HTML-код, а й JavaScript. У ASP.NET 2.0 було додано безліч нових складних елементів управління, наприклад, MultiView, TreeView, Wizard, GridView. Про можливості одного тільки GridView можна написати цілу статтю.

Таблиця 30.1. Таблиця відповідності деяких серверних елементів управління тегами HTML

Елемент управління ASP. NET	Відповідний тег HTML	Призначення
<asp:Label>	<Span>	Відобразити текст
<asp:ListBox>	<Select>	Список вибору
<asp:DropDownList>	<Select>	Список, що випадає
<asp:TextBox>	<Input Type="Text"> <Input Type="Password">	Рядок редагування Поле редагування паролю
<asp:HiddenField>	<Input Type="Hidden">	Невидиме поле
<asp:RadioButton>	<Input Type="Radio">	Перемикач
<asp:RadioButtonList>	<Input Type="Radio">	Список перемикачів
<asp:CheckBox>	<Input Type="CheckBox">	Прапорець

Продовження табл. 30.1

<asp:CheckBoxList>	<Input Type="CheckBox">	Список прапорців
<asp:Button>	<Input Type="button"> <Input Type="submit">	Користувацька кнопка Командна кнопка
<asp:Image>	<img>	Зображення
<asp:ImageButton>	<input type="image">	Кнопка-зображення
<asp:Table>	<Table>	Таблиця
<asp:Panel>	<Div>	Контейнер
<asp:BulletedList>	,	Маркерований список
<asp:HyperLink>	<A Href>	Гіперпосилання

Сервер не обов'язково генерує ті ж самі теги HTML для серверних елементів управління. Все залежить від типу браузера, який використовує клієнт.

Всі серверні елементи управління знаходяться в просторі імен System.Web.UI.Control і успадковуються від класу System.Web.UI.Web Controls.WebControl.

Опис та характеристики інших елементів будуть розглядатися в подальших лекціях. Всі існуючі класи ви можете переглянути за допомогою Class Browser. У всіх інтегрованих середовищах розробки є можливість додавати елементи керування за допомогою простого перетягування мишею.

30.1. Label

Цей елемент управління дозволяє виводити відформатований, аналогічно узагальненому строчному елементу розмітки <SPAN>. Всіма властивостями цього об'єкта можна управляти з вашої програми ASP.NET. Більшість методів і властивостей успадковані від WebControl. Головна власна властивість – це, звичайно, його зміст – Text.

Нижче (табл. 30.2) перераховані властивості, що керують зовнішнім виглядом елемента, присутні в класі WebControl і, отже, застосовні до всіх елементів-спадкоємців WebControl, а не тільки до Label.

Таблиця 30.2. Властивості елемента Label

BackColor	Колір фону
BorderColor	Колір меж елементу управління
BorderStyle	Стиль кордону – суцільний, пунктир, точковий, подвійний і інші
BorderWidth	Ширина кордону
Enabled	Активність. Якщо false, не можна вводити дані, не отримує фокус
Font	Шрифт, складається з декількох атрибутів
ForeColor	Колір, яким відображається текст
Height	Висота елемента
Width	Ширина елемента
Visible	Чи видно елемент управління
TabIndex	Індекс табуляції, в порядку номерів яких у формі переміщається фокус при натисканні на клавішу Tab
ToolTip	Текст вікна підказки

У версії 2.0 з’явилася можливість задавати для елементів управління „гарячі” клавіші, або клавіші швидкого доступу. Властивість AccessKey визначає послідовність натиснутих клавіш, яка призводить до установки фокусу на даному елементі. Наприклад, AccessKey = "N" означає, що для виклику функціональності треба натиснути на Alt + N. Встановити фокус в Label неможливо, тому встановлюємо властивість AssociatedControlId, яка вказує на інший елемент. Якщо це TextBox, то фокус встановлюється в нього.

30.2. Елемент управління TextBox

Він замінює елементи розмітки <textbox> і <textarea>, його основні характеристики наведені в таблиці 30.3.

Таблиця 30.3. Властивості елемента Textbox

Властивість	Визначення
Textmode	при властивості textmode, яка дорівнює MultiLine, вийде багаторядкове поле введення, а при SingleLine – однорядкове. Якщо textmode дорівнює Password, введені дані замінюються зірочками, як при <Input Type = "Password">. Природно, це потрібно в основному для введення пароля.
Rows	задається при textmode, встановленому в MultiLine, і задає кількість рядків для введення. Аналогічно функціонує властивість columns – кількість символів в рядку.
Wrap	Якщо властивість Wrap встановлена, то текст переходить на новий рядок, щоб повністю поміщатися у вікні.

У всіх класів, успадкованих від WebControl, в ASP. NET 2.0 з’явився метод Focus(). Він встановлює фокус на елемент керування. Найчастіше застосовується саме до TextBox. Додайте txtName. Focus() в подію Page_Load(), і курсор при завантаженні сторінки вже буде встановлений в потрібному рядку введення.

30.3. Елемент управління RadioButton

Перемикачі не ходять поодинці. Один перемикач на сторінці не має сенсу. Потрібні хоча б два. Типовий набір перемикачів визначається так:

<asp:RadioButton ID="RadioButton1" Runat="server" Text="Yes" GroupName="Set1" />

<asp:RadioButton ID="RadioButton2" Runat="server" Text="No" GroupName="Set1"/>

Атрибут Text виводиться праворуч від перемикача. У цьому прикладі важливо, що у обох перемикачів збігається властивість GroupName. Це дозволяє їм працювати як одне ціле. Перевага окремих перемикачів над RadioButtonList в тому, що між перемикачами можна помістити будь-який текст, картинки та інші елементи. У RadioButton є подія CheckedChanged, яка викликається, коли користувач вибирає один з перемикачів групи. Щоб обробник цієї події викликався, необхідно встановити властивість AutoPostBack. Група перемикачів працює так, що вибір одного елемента з групи знімає вибір з інших. Цей елемент управління підходить для вибору

однієї правильної відповіді з декількох. Властивість radSample.SelectedItem.Value показує вибраний елемент.

30.4. CheckBoxList

Загальний формат створення набору прапорців:

```
<asp:CheckBoxList ID="CheckBoxList1" runat="server">
    <asp:ListItem> текст 1 .</asp:ListItem>
    <asp:ListItem> текст 2 .</asp:ListItem>
</asp:CheckBoxList>
```

Якщо в CheckBoxList безліч варіантів, то можна їх розташувати в кілька стовпців. При цьому можна рухатися зверху вниз – справа наліво, або навпаки. Це залежить від RepeatDirection – Horizontal або Vertical. Текст може бути розташований праворуч або ліворуч від прапорця.

30.5. DropDownList

Загальний формат створення випадаючого списку:

```
<asp:DropDownList id="Вибір" runat="server">
  <asp:ListItem> текст1 </asp:ListItem >
  <asp:ListItem >текст 2</asp:ListItem >
</asp:DropDownList >
```

Тоді:

```
void Page_Load()
{
  if (Page.IsPostBack)
    lblMessage.Text = "Ви обрали" + Вибір.SelectedItem.Value;
}
```

Властивість Items елемента управління DropDownList має кілька методів для додавання і видалення рядків. Використовуючи методи Add і Insert, можна додати елемент або вставити його в зазначену позицію в DropDownList; AddRange дозволяє додати масив елементів в DropDownList; метод Clear видаляє всі елементи; методи Remove і RemoveAt видаляють зазначений елемент або елемент, що знаходиться у зазначеній позиції відповідно. Наприклад, так можна програмно створити DropDownList у функції Page_Load:

```
Category = new DropDownList();
Category.Items.Add("Комп'ютери");
Category.Items.Add("Принтери");
Category.Items.Add("Комплектуючі");
ListItem selItem = new ListItem("Монітори", "монітори");
Category.Items.Add(selItem);
Category.Items.Add(new ListItem("Компакт-диски"));
Category.SelectedIndex = 3;
form1.Controls.Add(Category);
```

Щоб очистити вибір на елементі DropDownList, встановіть SelectedIndex в (–1). Якщо встановити у ListItem властивість Enable в false, то його буде не видно в списку, проте з ним можна працювати з програми. Подія SelectedIndexChanged запускається, коли користувач обирає інший елемент.

30.6. ListBox

Загальний формат створення списку

<asp:ListBox>

Елемент керування дозволяє вибрати кілька пунктів списку одночасно. Для цього треба встановити його властивість SelectionMode:SelectionMode = „multiple”.

Властивість Rows встановлює кількість елементів, які видно в листі. Якщо елементів більше, то з’являється смуга прокрутки.

Властивість Items повертає колекцію елементів ListItem, які знаходяться в списку. Вона дозволяє визначити вибрані пункти.

30.7. Panel

Часто буває потрібно вставити елемент керування в точно визначене місце сторінки. У HTML для цього використовують елемент розмітки <DIV> – стандартний контейнер. Його аналог в ASP. NET:

<asp:Panel>

Щоб змусити мітку відображатися перед списком, необхідно помістити перед DropDownList об’єкт Panel:

*<asp:Panel ID="Panel1" runat="server"> </ asp: Panel>
*

після чого викликати метод Controls.Add (...) від цього об’єкта:

Panel1.Controls.Add (ShopNews);

Властивість HorizontalAlign елемента Panel корисна, якщо потрібно встановити вирівнювання елементів управління, які в ньому містяться. Поміняємо код в попередньому прикладі:

<asp: Panel ID = "Panel1" runat = "server" HorizontalAlign="Center" width=500 />

Текст тепер розміщується в центрі мітки. Булева властивість Wrap елемента Panel змушує переносити текст на новий рядок, якщо вона встановлена, або розширювати розділ, коли текст не вміщується в один рядок, якщо вона не встановлена. Якщо в програмі встановити властивість Visible панелі в False, можна зробити невидимими всі елементи, які в ній знаходяться. Стилі, встановлені в панелі, успадковуються всіма вкладеними елементами.

30.8. Button

Button – це командна кнопка, натискання на яку часто призводить до відправки даних на сервер. Можна створювати кнопки двох типів: для передачі даних форми (submit button) або командні кнопки для виконання різних функцій, пов’язаних з даною кнопкою. Якщо на формі є кілька кнопок, властивість CommandName дозволяє дізнатися, яка саме кнопка була натиснута.

ASP.NET підтримує 3 вида подій:

- Події, які відбуваються в браузері клієнта і обробляються кодом на Javascript.
- Події завантаження сторінки.
- Події елементів управління.

Наприклад, щоб обробити клацання на кнопці, ми перевизначаємо події Click.

```
protected void Button1_Click(object sender, EventArgs e)
  {
  }
```

Події можна визначити через вкладку подій у вікні властивостей. Другий аргумент всіх обробників подій має тип EventArgs або який-небудь успадкований від нього.

30.9. Image

Елемент управління asp:image відповідає тегу img мови HTML. Його можна використовувати для динамічного додавання на сторінку нових зображень. Повернемося до нашого туристичного агентства. Ми вирішили, що, коли клієнт обирає міста, на сторінку автоматично повинна виводитися карта відповідного міста. Залишаємо це, як вправу. Карти міст можна знайти через пошукову систему Яндекс.

<asp:Image> має властивості AlternateText, ImageUrl, ImageAlign.

Таблиця 30.4. Властивості елемента Image

AlternateText	Відповідає атрибуту ALT тега IMG. Відображається, якщо показ картинок відключений або картинки неможливо знайти
ImageUrl	Відповідає атрибуту SRC тега IMG
ImageAlign	Відповідає атрибуту ALIGN тега IMG

Як завжди, властивості можна змінювати з програми. Наприклад, змінюючи значення ImageUrl, можна організувати перегляд безлічі картинок у вигляді слайд-шоу.

30.10. ImageButton

Елемент управління ImageButton являє собою комбінацію елементів Image і Button. Його можна використовувати для створення зображень, чутливих до кліку мишки. Клацання є подією, при настанні якого виконується деякий код:

```
<asp:ImageButton id="imgButton" OnCommand ="SubmitPartl" runat= "server" />
```

ImageButton дозволяє досягти ефекту, аналогічного карті зображення. Подія Click дозволяє дізнатися координати клацання миші і реагувати відповідно регіону, в якому була натиснута миша. Обробник події повинен приймати аргумент типу ImageClickEventArgs – спадкоємця System.EventArgs. У нього є додаткові поля X і Y – координати кліка мишки:

```
protected void ImageButton1_Click(object sender,
System.Web.UI.WebControls.ImageClickEventArgs e)
{
// обробка подій
}
```

30.11. HyperLink і LinkButton

HyperLink – гіперпосилання звичайне або з картинкою. Воно дозволяє пересуватися по сайту або давати посилання на інші сайти:
<asp: HyperLink ID = "HyperLink1" runat = "server" NavigateUrl = "~/Customer.aspx"> HyperLink </ asp: HyperLink>.

Знак ~ позначає кореневий каталог поточного сайту.

LinkButton – це кнопка, яка виглядає як гіперпосилання. Натискання на неї приводить до перезавантаження сторінки. У властивості PostBackUrl можна задати адресу сторінки, яка буде обробляти поточну.

30.12. Calendar

Цей клас не має аналогів в HTML. Визначивши єдиний елемент управління, можна створити і надати в розпорядження відвідувачів повноцінний календар, де вони зможуть прокручувати місяці, вибирати день або тиждень. Зовнішній вигляд цього елемента управління може бути найрізноманітнішим. І все це реалізується засобами HTML. Раніше це було можливо тільки за допомогою ActiveX – контролів, які потрібно завантажувати з сервера, реєструвати в системі і перевіряти на безпеку. Calendar має безліч властивостей, деякі з них наведені в табл. 30.5.

Таблиця 30.5. Властивості елементу управління Calendar

CellPadding	„Набивання” (відстань між кордонами клітинки і її вмістом)
CellSpacing	Відстань між клітинами
DayNameFormat	Спосіб написання назв днів тижня. Може приймати значення FirstLetter, FirstTwoLetters, Full, Short
FirstDayOfWeek	Для завдання першого дня тижня, Default – установки, прийняті в системі
NextPrevFormat	Показ назв попереднього і наступного місяців. FullMonth – повна назва, ShortMonth – перші 3 літери, CustomText – будь-який текст, визначений програмістом
SelectionMode	Спосіб вибору дати. Доступні Day, DayWeek, DayWeekMonth і None
ShowDayHeader	Чи показувати назви днів тижня (так за замовчуванням)
ShowGridLines	Чи показувати сітку (ні за замовчуванням)
ShowTitle	Чи показувати заголовок (ні за замовчуванням)
TitleFormat	MonthYear, Month
TodaysDate	Яка дата буде обрана поточною. За замовчуванням – дата на сервері
VisibleDate	Місяць, який буде показаний в календарі

Calendar підтримує різні календарні системи – не тільки звичний григоріанський, але і юліанський, іудейський, мусульманський, буддистський. Це можна зробити, змінюючи культурну інформацію сторінки. Властивість SelectionMode, рівне DayWeekMonth, дозволяє вибрати як конкретний день, так і тиждень або місяць. Як же отримати вибраний діапазон дат? SelectedDate повертає тільки перший день діапазону. Для цього існує колекція Calendar1.SelectedDates. А ще елегантніше буде написати так:

```
foreach ( DateTime i in calVoyage.SelectedDates)
        {  TextToUser.Text += i.ToShortDateString() + "<br>";
```

}

Можна заборонити користувачеві вибирати минулу дату. Для цього можна скористатися властивістю IsSelectable:

```
if (e.Day.Date < DateTime.Now)
{ e.Day.IsSelectable = false;
}
```

30.13. Відправка даних іншій сторінці

У ASP. NET не розв'язується відправка даних між сторінками. У ASP. NET елементи управління мають властивість PostBackUrl, де можна вказати, якій сторінці система повинна передати Web-форму, якщо відправлення даних на сервер ініційоване цим елементом управління. Через властивість PreviousPage сторінки можна з'ясувати, яка сторінка була джерелом постбека нашої сторінки.

30.13. 1. AutoPostBack. Прив'язка до даних. Колекції. Перевірка правильності даних, що вводяться

Програмування в ASP.NET орієнтоване на події. Події на сторінці (наприклад натискання на кнопку) обробляються на сервері. Зміни в тексті поля редагування, вибору опції в списку, натиснення на прапорець або перемикач не викликають негайної відправки на сервер. Цього можна домогтися, якщо встановити властивість AutoPostBack для цих елементів.

Якщо AutoPostBack встановлений для елемента керування TextBox, то для нього буде викликатися подія TextChanged, як тільки поле втратить фокус або буде натиснута клавіша Enter. Щоб ця властивість працювала, браузер повинен підтримувати ECMAScript (стандарт JavaScript, прийнятий Європейською асоціацією виробників комп'ютерів).

Джерелом даних для елементів управління можуть служити таблиці даних. Давайте розберемо приклад, що входить до складу Visual Studio – CarSelectorSample. Дія відбувається в електронному магазині автомобілів (табл. 30.6). Є різні марки машин, причому для кожної марки є декілька моделей. При виборі марки машини в першому списку в другий список автоматично вантажаться відповідні моделі:

Таблиця 30.6. Приклад електронного магазину автомобілей

Brand	Buick	Chevrolet	Pontiac	Toyota	Mileage	Featurs
Buick	Century	Impala	Grand Am	Avalon	10– 10 000	Leatherseat
Chevrolet	LeSabre	Malibu	Grand Prix	Camry	10000– 20000	Sun roof
Pontiac	Park Avenue	Metro	Montana	Camry Solara	20000– 30000	CD player
Toyota	Regal	Prizm	Sunfire	Celica	30000 and more	ABS

Всі дані, які використовуються на цій сторінці, зібрані в таблицю. Для зберігання такої таблиці існує клас DataTable. Таблиця складається із стовп-

ців – DataColumn і рядків DataRow. Клас DataView дозволяє створювати різні представлення даних таблиці. Перший стовпець служить джерелом даних списку марок. Залежно від обраної моделі, в список моделей завантажується одна з 2–5 колонок.

Спочатку створюється таблиця:

Cars = new DataTable();

Cars.Columns.Add(new DataColumn("Brand", typeof(string)));

Тут викликається один з конструкторів DataColumn. Перший аргумент – назва колонки, другий – тип:

CarRow = Cars.NewRow();

Створюється новий рядок таблиці. Чарунка таблиці задається за допомогою індексу рядка:

CarRow[6]= "Power seat";

І рядок додається в таблицю:

Cars.Rows.Add(CarRow);

У списку марок, що випадає, встановлено властивість AutoPostBack. Це означає, що сторінка автоматично подається на сервер, коли в цьому списку змінюється вибраний елемент. В обробнику вибору нового елементу спочатку з'ясовується, обраний елемент:

string selected = DropDownList1.SelectedItem.Value;

У операторі switch відбувається перемикання другого списку на один із стовпців таблиці завданням властивостей DataTextField і DataValueField, де DataTextField – текст, що відображається у списку, а DataValueField – вибране значення. В даному випадку, як часто буває, вони однакові.

30.13.2. Прив'язка до даних

Деякі елементи управління: списки, таблиці та інші – мають властивість DataSource, яка відповідає за прив'язку до даних. Тип цієї властивості – object, тобто він може бути будь-яким, але повинен реалізовувати інтерфейс IEnumerable. Часто значеннями цієї властивості призначають колекції. У такому випадку немає потреби додавати значення вручну. Властивість DataSource може бути прив'язане до колекцій, що підтримують інтерфейси IEnumerable, ICollection або IListSource. Джерелом даних також можуть бути XML – файли, бази даних. Викликом методу DataBind дані прив'язуються до елементу управління. Метод Page.DataBind викликає прив'язку даних у всіх елементів на сторінці.

Наведений нижче список, що випадає, допомагає вибрати континент для подорожі. Джерело даних – динамічний масив ArrayList. Використовуйте його, якщо в програмі відбувається багато вставок і вилучень:

```
void Page_Load()
   {
      ArrayList ContinentArrayList = new ArrayList();
      ContinentArrayList.Add("Worldwide");
      ContinentArrayList.Add("America");
      ContinentArrayList.Add("Africa");
```

```
        ContinentArrayList.Insert(1, "Asia-Pacific");
        ContinentDropDownList.DataSource = ContinentArrayList;
        ContinentDropDownList.DataBind();
    } //End Page_Load()
....
<asp:DropDownList id="ContinentDropDownList" runat="server" />
```

Контрольні запитання

1. Чим відрізняються форми ASP.NET від звичайних?
2. Яким методом обробляються всі форми Web-додатку?
3. Для чого потрібен атрибут ID у елементах управління?
4. В якому просторі імен знаходяться серверні елементи управління?
5. Яка подія RadioButton викликається коли користувач обирає один з перемикачів групи?
6. Яка властивість списку ListBox встановлює кількість елементів, що буде видно в листі?
7. Яка властивість Button дозволяє дізнатися яка кнопка була натиснута, якщо на формі Web-додатку їх декілька?
8. Які властивості має елемент управління asp:image?
9. За допомогою якої властивості Calendar можна заборонити користувачеві обирати минулу дату?
10. За що відповідає властивість DataSourse?

Тема 31. Валідація даних

31.1. Класи перевірки даних (валідатори)

Переважна більшість сторінок Web додатків використовуються для введення даних користувачем, які потім можуть зберігатися в допоміжних файлах, базі даних і т.п. Одним з найважливіших етапів отримання даних від користувача є їх перевірка, необхідна для того, щоб виключити даремну, неінформативну, або суперечливу інформацію.

Критерії перевірки можуть бути найрізноманітнішими, починаючи з того, вводилися дані взагалі і закінчуючи перевіркою типу даних.

В ідеалі перевірка даних, що вводяться, повинна здійснюватися на стороні клієнта, т.я. в цьому випадку він може відразу ж, ще до відправки даних на сервер, отримати повідомлення про помилки, які там містяться. Однак, незалежно від того, чи здійснювалася перевірка даних, що вводяться на стороні клієнта, необхідно здійснювати перевірку і на стороні сервера.

Клієнтська перевірка даних, що вводяться, зазвичай здійснюється за рахунок використання програмного коду, написаного мовою JavaScript, при цьому виникає ряд складнощів, які пов’язані з відмінностями клієнтського програмування і серверного.

У ASP.NET реалізований цілий ряд елементів управління, призначених для перевірки введених даних, так званих верифікаторів. Ці елементи можна прив’язати до будь-якого елементу управління вводом. Після прив’язки, верифікатор виконує автоматичну клієнтську і серверну перевірку даних, що

вводяться. Якщо дані, введені в елемент введення даних, не задовольняють умові верифікатора, останній перешкоджає відправці сторінки на сервер.

Всього в ASP.NET 2.0 існує шість елементів управління, призначених для здійснення перевірки введених даних (табл. 31.1).

Таблиця 31.1. Елементи управління для перевірки введених даних

RequiredFieldValidator	Контролює наявність введених даних в елемент управління
RangeValidator	Перевіряє знаходження значення елемента управління в межах заданого діапазону
RegularExpressionValidator	Визначає відповідність значення даного елемента управління певному регулярному виразу
CompareValidator	Порівнює значення поточного елемента управління з константою або значенням іншого елемента управління
CustomValidator	Виконує задану операцію перевірки достовірності на стороні клієнта або на стороні сервера для реалізації власної логіки перевірки введених даних
ValidationSummary	Відображає інформацію на сторінці, або у спливаючому вікні з повідомленнями про помилки для кожного елемента управління, перевірка якого завершилася помилкою

Допускається використання декількох елементів управління перевіркою введення даних, пов’язаних з одним елементом введення даних. За допомогою верифікаторів можливо перевірити дані, що вводяться в такі елементи управління як TextBox, ListBox, DropDownList, RadioButtonList, HtmlInputText, HtmlTextArea, HtmlSelect.

При використанні валідаторів можливе використання автоматичної перевірки сторінки під час її надсилання не сервер, або ж вручну в коді. Найбільш поширений перший підхід, при якому користувач бачить звичайну сторінку з елементами введення даних, заповнює їх і ініціює відправку сторінки на сервер. Якщо відправлення ініціюється натисканням на кнопку (що буває в більшості випадків), перевірка введених даних залежить від значення властивості CausesValidation кнопки.

1 Якщо властивість CausesValidation = false, елементи управління перевіркою даних, що вводяться, просто ігноруються, а сторінка відправляється на сервер. При цьому, всі обробники подій будуть виконані.

2. Якщо властивість CausesValidation = true, здійснюється автоматична перевірка кожного елемента керування, розташованого на сторінці. Якщо при перевірці елемента управління виникає помилка, ASP.NET повертає сторінку з повідомленнями про помилку в залежності від налаштувань верифікаторів. При цьому обробник події натискання на кнопці, який ініціював відправку сторінки на сервер може виконатися, але може і не виконатися. Для перевірки цього необхідно здійснити перевірку достовірності сторінки.

Перевірка даних, що вводяться, здійснюється під час клацання користувачем на певних кнопках, наприклад, при виникненні події зміни значення поточного елементу списку, такого не відбувається. Проте, верифікатори до-

дають код для перевірки введених даних на стороні клієнта. У цьому випадку, практично вся перевірка може здійснюватися на стороні клієнта, що для клієнта буде виглядати ідентично результатами перевірки даних на стороні сервера. Проте, при успішній перевірці даних на стороні клієнта, на стороні сервера подібна перевірка буде здійснена повторно.

Всі валідатори утворені від класу BaseValidator і мають наступні основні загальні для всіх властивості, які наведені в таблиці 31.2.

Таблиця 31.2. Властивості валідаторів, що утворені від класу BaseValidator

ControlToValidate	Містить ідентифікатор елемента керування введення даних, що підлягає перевірці.
Display	Визначає спосіб відображення повідомлення про помилку. Значення static означає, що при створенні сторінки буде заздалегідь підраховано і зарезервовано місце на сторінці, необхідне для відображення повідомлення про помилку. Значення dynamic дозволяє змінювати сторінку в процесі виконання, додаючи повідомлення про помилку при виникненні необхідності.
Enabled	Визначає стан валідатора. При значенні false елемент є відключеним і не здійснює перевірку даних, що вводяться.
EnableClientScript	Визначає чи буде виконуватися перевірка даних, що вводяться на стороні клієнта.
ErrorMessage	Містить рядок помилки, яка буде відображатися в підсумковій інформації про помилки елемента ValidationSummary.
Text	Містить рядок тексту помилки, яка відображається валідатором в разі виникнення помилки.
IsValid	Містить значення поточного результату перевірки вводу даних. Ця властивість має сенс перевіряти тільки в тому випадку, якщо перевірка на стороні клієнта не здійснювалася.
SetFocusOnError	Дозволяє встановити фокус введення на той елемент управління, який викликав помилку при перевірці даних, що вводяться. Для включення цього режиму необхідно встановити значення властивості рівним true.
ValidationGroup	Визначає групу, до якої можливе об’єднання кількох валідаторів для виконання незалежної перевірки.

Отримати доступ до валідатора просто – розкрийте в Toolbox вкладку „Validation”. Класи валідаторів утворюють ієрархію, на чолі якої стоїть абстрактний клас Base Validator, як показано на рис. 31.1.

Базовий клас валідаторів сам спадкоємець класу Label, так що по суті всі валідатори – мітки, текст в яких стає видимим, коли не виконуються задані нами умови перевірки. За умовчанням текст в валідаторах – червоний (згадайте школу та зауваження вчительки в зошиті). Але, звичайно ж, цей колір можна змінити на більш приємний. Всі валідатори мають властивість ControlToValidate. Вона задає той елемент управління, дані в якому перевіряються даним валідатором. Цей елемент повинен знаходитися в одному контейнері з валідатором. Найбільш часто застосовані властивості наведені в табл. 31.3.

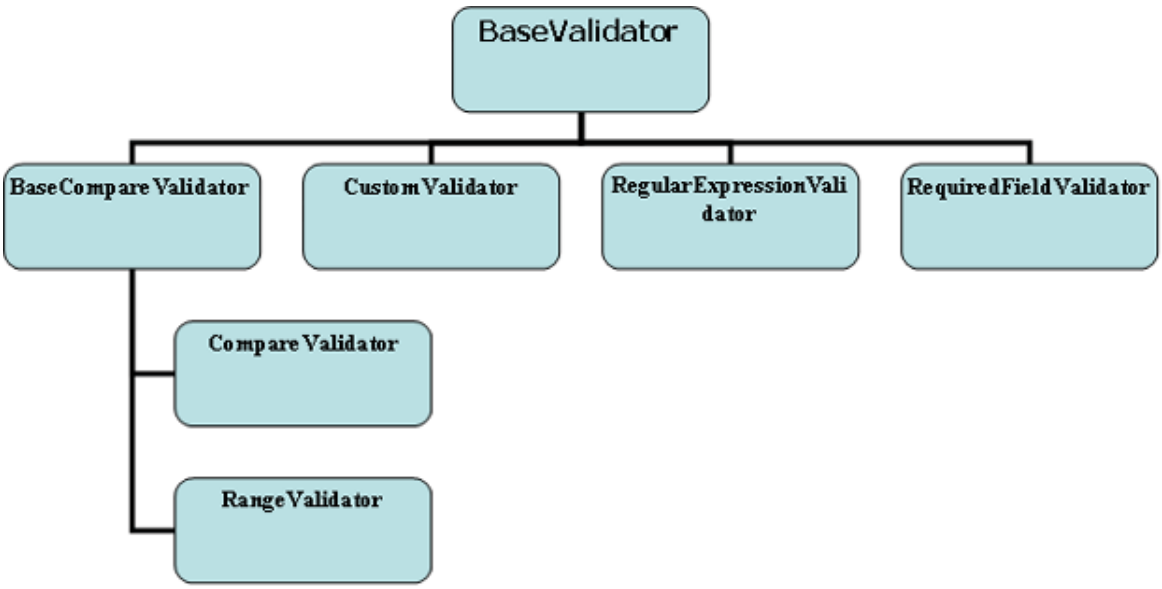

Рисунок 31.1 – Ієрархія класів валідаторів

Таблиця 31.3. Загальні властивості валідаторів

Display	Чи надавати місце статично або динамічно
EnableClientScript	Чи дозволяти генерувати клієнтський код
ErrorMessage	Текст повідомлення про помилку
IsValid	Чи пройшов валідацію пов’язаний з валідатором елемент управління

31.2. Ініціація перевірки даних

Перевірка завжди ініціюється якою-небудь подією. Зазвичай це клацання на кнопках Button, ImageButton, LinkButton, в які за умовчанням властивість CausesValidation встановлена в True. Можна прибрати цю властивість для деяких кнопок, яким вона не потрібна, наприклад, для кнопки Cancel.

У прикладі з автосалоном на сторінці є декілька валідаторів:

<asp:requiredfieldvalidator id="RequiredFieldValidator2" runat="server" ErrorMessage="Required" ControlToValidate="DropDownList1">

</asp:requiredfieldvalidator>

Клас RequiredFieldValidator перевіряє, чи було змінено значення у зв’язаному з ним елементі управління. Якщо, як в даному випадку, це випадає список – спочатку вибрано порожнє значення, але потрібно, щоб користувач вибрав конкретну марку. Якщо вибір не був зроблений, але кнопка submit була натиснута, валідація провалюється і виводиться текст, заданий в ErrorMessage або в Text. Валідатори відображають текст, вказаний у властивості "Text", завжди, коли вона не порожня, а текст, встановлений у властивості "ErrorMessage" – тоді, коли властивість "Text" дорівнює " ". Первинне значення задається властивістю InitialValue. Якщо ця властивість не задана, то перевірка проводиться на відсутність значення (наприклад, порожній Textbox). Для перевірки коректності введення електронної адреси використовується клас RegularExpressionValidator:

Your Email

(Required)

<asp:textbox id="TextBox1" runat="server"></asp:textbox>

<asp:RegularExpressionValidator id="RegularExpressionValidator1" runat ="server" ControlToValidate="TextBox1" ErrorMessage="Not a valid Email" ValidationExpression="\w+([–+.]\w+)@\w+([–.]\w+)*\.\w+([–.]\w+)*">*
</asp:RegularExpressionValidator>

<asp:RequiredFieldValidator id="RequiredFieldValidator1" runat="server" ControlToValidate="TextBox1" ErrorMessage=""> </asp:RequiredFieldValidator>*

**

ValidationExpression – регулярний вираз, на відповідність якому проходить перевірку значення текстового поля. У Visual Studio 2005 надає кілька готових шаблонів регулярних виразів, які можна вибрати у вікні властивостей, – телефонних номерів різних країн, адрес, і, найкорисніше, шаблони електронної пошти і адреси в Інтернеті.

З одним елементом управління може бути пов’язано кілька валідаторів. Наприклад, електронна адреса перевіряється і на відповідність шаблону, і на обов’язкове заповнення.

Властивість Page.IsValid дозволяє визначити, чи пройшла вся сторінка валідацію. Для браузерів, які підтримують DHTML, перевірка відбувається на стороні клієнта. Для цього автоматично генерується JavaScript-код. Таким чином економляться ресурси сервера і трафік, які б довелося витратити на передачу неправильних даних.

31.3. Валідатори порівняння

CompareValidator порівнює значення із значенням в іншому елементі управління або з константою. Також можна перевірити, чи можна конвертувати значення в пов’язаному з ним елементі управління в який-небудь тип. Властивість Operator дозволяє встановити операцію, за допомогою якої відбувається порівняння: Equal, NotEqual, GreaterThan, GreaterThanEqual, LessThan , LessThanEqual. Значення DataTypeCheck означає перевірку на відповідність типу.

Властивість Type може приймати значення String, Integer, Date, Double і Currency. Властивість ControlToCompare задає елемент управління, з яким відбувається порівняння. ValueToCompare задає значення. З цих двох властивостей встановленим може бути тільки одне.

RangeValidator перевіряє відповідність введеного значення діапазону, заданому властивостями MinimumValue і MaximumValue:

<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>

<asp:RangeValidator ID="RangeValidator1" runat="server" ControlToValidate="TextBox1" ErrorMessage="Не більше 10 бутилок пива в одні руки" MaximumValue="10" MinimumValue="1" Type="Integer">

</asp:RangeValidator>

31.4. ValidationSummary

Клас ValidationSummary дозволяє вивести підсумкову інформацію про всіх валідаторів на сторінці. Вона може бути виведена в різній формі:

- BulletList – список зі значками;
- List – звичайний список;
- SingleParagraph – простий параграф.

Інформацію можна виводити на сторінці, а можна в інформаційному вікні, якщо ShowMessage поставити в True. Для всіх валідаторів виводиться властивість Error Message, а не текст. Text виводиться в самому валідаторі.

31.5. CustomValidator

Якщо потрібно зробити таку перевірку, яку не виходить здійснити за допомогою стандартних валідаторів, в гру вступає Custom Validator. У класі CustomValidator можна написати будь-яку функцію, яка перевірятиме значення як на стороні сервера, так і у клієнта. Класичний приклад – перевірка числа на парність.

Елемент управління RangeValidator використовується для перевірки попадання введеного значення в діапазон значень валідатора. Основними властивостями даного валидатора є MinimumValue, MaximumValue і Type. Перші два визначають діапазон допустимих значень, type – тип даних, що вводяться в елемент керування.

Як приклад, додамо до попередньої сторінки елемент введення даних про вік користувача, який повинен бути в межах від 18 до 70 років, а також верифікатор RangeValidator, який контролюватиме значення числа, що вводиться.

Варто відзначити, що у валідаторів існує можливість використання графічних об’єктів замість тексту, що виводяться на екран в разі виникнення помилки. Так, в даному прикладі можливе використання зображення замість символу зірочка. Для цього необхідно ввести у властивість Text валідатору тег <img>, із зазначенням графічного файлу, який необхідно відображати у разі виникнення помилки.

31.6.Групи валідації

На складних сторінках введення даних може існувати багато елементів для введення даних. Залежно від їх призначення, буває зручно об’єднати їх в логічні групи, в яких була б реалізована своя логіка перевірки даних. Для цього доцільно згрупувати валідатори. Це буває доцільно у випадку, коли на сторінці розташовано кілька панелей, усередині яких знаходяться окремі поля введення даних і кнопки, які ініціює подія postback. У таких ситуаціях необхідно, щоб усередині кожної панелі реалізовувалася окрема логіка перевірки введених даних.

У ASP.NET таке можливо завдяки групам перевірки. Для створення групи перевірки введених даних необхідно помістити елементи введення і кнопку в одну групу перевірки даних. Для цього необхідно встановити однакове значення властивості ValidationGroup всіх елементів управління, що входять в цю групу. Як значення властивості ValidationGroup використовується рядок, наприклад Login, Registration і т.п.

Для відображення підсумкової інформації про результати всіх перевірок інформації, що вводиться, необхідно використовувати елемент управління ValidationSummary. Він виводить значення ErrorMessage кожного валіда-

тора, для якого ця перевірка завершилася невдачею. Підсумкова інформація може відображатися або на сторінці, або в окремому вікні. Для вказівки даного режиму необхідно встановити властивості ShowMessageBox і ShowSummary. При значенні властивості ShowMessageBox = true, повідомлення виводиться в окремому вікні, якщо ж значення властивості ShowSummary=true – на сторінці. При відображенні підсумкової інформації на сторінці можливо встановити деякі додаткові параметри за допомогою властивості DisplayMode, а також задати заголовок для підсумкової інформації за допомогою властивості HeaderText.

Доступ до верифікаторів можливий з програмного коду. Всі валідатори, при їх розміщенні на сторінці поміщаються в колекцію Validators. Для послідовного звернення до всіх валідаторів можна організувати циклічний перебір елементів цієї колекції.

31.7. Комбінування верифікаторів

Дуже часто необхідно здійснювати кілька перевірок, що вводяться. Зробити це можна різними способами, в тому числі і за рахунок створення користувальницької логіки перевірки введених даних. Однак, більшість завдань можуть бути вирішені шляхом комбінування верифікаторів. Для інтерпретації повідомлень про помилки, що генеруються, при використанні декількох верифікаторів з одним елементом управління, необхідно розуміти правила перевірки введених даних, прийняті в ASP.NET. Так, всі верифікатори за винятком RequiredFieldValidator прирівнюють відсутність введеного значення до введення правильного значення. Таким чином, за відсутності введених даних генеруються одні повідомлення (це перевіряє RequireFieldValidator) Та інші, якщо введене значення знаходиться поза допустимого діапазону або має невірний формат.

Для додавання ще одного верифікатора і зв’язування його з поточним полем введення, необхідно додати його на сторінку і встановити для нього властивості Text, ErrorMessage, ControlToValidate, ValidationGroup.

Контрольні запитання

1. Як називаються елементи управління, що призначені для перевірки введених даних в ASP.NET?
2. Для чого потрібна властивість валідаторів ControlToValidate?
3. Що таке ValidationExpression?
4. Яка властивість дозволяє визначити, чи пройшла вся сторінка валідацію?
5. Як називається валідатор, що порівняє значення із значенням в іншому елементі управління?
6. Яка властивість дозволяє встановити операцію, за допомогою якої відбувається порівняння?
7. За допомогою якого класу можна вивести підсумкову інформацію про всіх валідаторів на сторінці?
8. Скільки всього існує елементів управління, що призначені для перевірки введених даних?

9. Яка властивість перевіряє відповідність введеного значення діапазону, заданому властивостями MinimumValue і MaximumValue?

10. Для чого використовується елемент управління RangeValidator?

Тема 32. Шаблони дизайна сторінок MasterPage. Створення користувацьких елементів управління UserControl

Для простого користувача відмінність одного сайту від іншого – в різноманітному дизайні сторінок. Більшість web-сайтів сьогодні мають впізнаваний дизайн, який досягається використанням одних і тих же елементів у тих же самих місцях у різних сторінках сайту. Тому дизайн сторінок є навряд чи менш важливим, ніж загальна функціональність.

Деякі розробники копіюють і вставляють повторювані елементи у всіх сторінках. Це неефективно, адже якщо потрібно змінити одну деталь в цих загальних елементах, зміни доведеться вводити у всіх сторінках. Можна поміщати повторювані шматки коду в файли, що включаються за допомогою команди HTML include. Але так важко побачити остаточний вигляд сторінки в середовищі розробки!

Кращий спосіб створення та повторного використання сторінок повинен задовольняти трьом вимогам:

1) легкість в модифікуванні сторінки

2) внесення змін не повинно вимагати обширної перекомпіляції і численних виправлень у вихідному коді

3) будь-яка зміна має мінімально впливати на загальну продуктивність програми.

Одним із засобів вирішення подібних завдань є майстер сторінки (master pages). Він реалізує просту модель створення шаблонів форм з можливістю їх повторного використання.

За допомогою шаблонів сторінок можна визначити деякий загальний зміст і помістити його в сторінку з розширенням .master. Природно, таких сторінок в додатку може бути кілька. Цей шаблон можуть використовувати будь-яку кількість дочірніх сторінок, які, як і звичайні сторінки, мають розширення aspx. Для реалізації даного механізму, в ASP.NET введені такі типи сторінок, як майстер-сторінки (master pages) і сторінки вмісту (content pages). Майстер-сторінка (еталонна сторінка) являє собою шаблон сторінки, при цьому, вона може містити будь-які елементи, допустимі для звичайної сторінки, а також програмний код. Сторінка вмісту містить допустимі елементи управління, за допомогою яких визначає вміст, яким заповнюються спеціальні області майстер-сторінок. Зазвичай, майстер-сторінка містить фіксовані елементи, однакові для всіх сторінок, і заповнювач вмісту для іншої частини сторінки. Найбільш типовими фіксованими елементами є верхній і нижній колонтитули, панель навігації, панель меню і т.д. Сторінка вмісту отримує від майстер-сторінки фіксовані елементи і надає додатковий вміст.

Для кінцевого користувача еталонні сторінки абсолютно прозорі. При роботі з додатком з боку клієнта видно тільки адреси сторінок контенту. При

запиті сторінки виконуюче середовище застосовує різні алгоритми компіляції і створює динамічний клас, об'єднуючи вміст еталонної і контентної сторінок.

Для розробки веб-сторінок на основі еталона треба почати зі створення самого еталона. Еталонна сторінка – це файл з розширенням. master, чий синтаксис не надто відрізняється від застосовуваного в звичайній сторінці. aspx. У еталонної сторінки є 2 основні ознаки:

1) директива @Master, що замінює директиву @Page

2) один або кілька дочірніх елементів управління ContentPlaceHolder.

Кожен вбудований елемент ContentPlaceHolder вказує область розмітки, чий реальний вміст буде визначатися сторінками контенту в період виконання.

Тіло еталонної сторінки може містити будь-яку комбінацію серверних елементів управління, звичайного тексту, зображень, HTML-елементів і керованого коду.

32.1. Створення еталонної сторінки Master page

У діалозі Add New Item виберіть тип сторінки Master Page. Як і звичайні сторінки, їх можна створювати з відокремленим кодом або кодом, вбудованим в сторінку. Це ніяк не впливає на модель поділу коду дочірніх сторінок. Крім того, головна і дочірня сторінки можуть розроблятися на різних мовах.

При створенні майстер сторінки, вона повинна мати хоча б один елемент ContentPlaceHolder – контейнер, за допомогою якого відзначаються ті місця, куди дочірні сторінки можуть поміщати свій власний контент. Поле підстановки контенту визначається унікальним ідентифікатором:

<asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">

Фактично, дана майстер-сторінка готова для використання, тобто в ній присутні всі необхідні елементи для того, щоб відображати вміст сторінок вмісту (фактично, для цього потрібен тільки елемент управління ContentPlaceHolder) і необхідний мінімум дизайну. Однак, сама по собі майстер-сторінка не може бути відображена у вікні браузера, тому що вона надає необхідні елементи дизайну і розташування елементів управління, що входять в шаблон сторінки, сторінки вмісту, яка і містить ті дані, які мають бути відображені у вікні браузера. Тому, для перевірки результатів роботи, необхідно створити хоча б одну сторінку вмісту.

32.2. Створення сторінки контенту

В якості сторінки – спадкоємиці (нащадка) шаблону може використовуватися будь-яка сторінка, додана в додаток або створена за замовчуванням default.aspx.

1) Додавання в додаток. Сторінка створюється, як зазвичай, тільки відзначається прапорець з написом Select Master Page. З'являється діалог, в якому необхідно вибрати шаблон сторінки:

<%@ Page Language="C#" MasterPageFile="~/MasterPage.master" AutoEventWireup="true" CodeFile="MainSchool.aspx.cs" Inherits="MainSchool" Title="Untitled Page" %>

Атрибут MasterPage директиви Page визначає шаблон дизайну, або еталонну сторінку даної сторінки.

2) В якості сторінки вмісту використовуємо сторінку Default.aspx, додану до проекту Web-додатку за замовчуванням. Для того, щоб перетворити звичайну сторінку в сторінку вмісту, необхідно в якості значення властивості MasterPageFile сторінки, вказати ім’я майстер сторінки, а також додати на сторінку елемент управління <asp:Content>. Даний елемент керування відсутній на панелі елементів управління Toolbox. Для цього необхідно перейти в режим редагування дизайну сторінки вмісту та виконати команду контекстного меню Create Custom Content елемента керування ContentPlaceHolder При цьому активізується вміст даного елемента управління, внаслідок чого стає можливим вводити текст, а також додавати інші елементи управління в нього. Фактично, при цьому, всі дані що вводяться розміщуються всередині сторінки Default.aspx.

Вихідний код сторінкиDefault.aspx в результаті виконаних операцій, а також введення тексту „це початкова сторінка Default.aspx”, буде виглядати наступним чином:

<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default" MasterPageFile="~/MainMasterPage.master"%>

<asp:Content ID="Content1" runat="server" ContentPlaceHolderID= "ContentPlaceHolder1">Це початкова сторінка Default.aspx</asp:Content>

Можливо налаштувати додаток так, щоб всі сторінки успадковували одну сторінку шаблону дизайну. У файлі конфігурації в секцію System.web потрібно вставити елемент:

<pages masterPageFile="~/ MasterPage.master " />

Але і в цьому випадку призначення головної сторінки в директиві Page має пріоритет над призначенням на рівні додатку. Установка web.config діє на тих сторінках, в яких masterPageFile не вказаний, але визначені елементи управління Content. Ця установка не діє на звичайні aspx-сторінки.

Сторінка, пов’язана з еталоном, автоматично успадковує всі його вміст і отримує можливість приєднати власну розмітку і серверні елементи управління до всіх заданих полях підстановки. а дочірня сторінка редагується в середовищі розробки, на вкладці Design видна повна сторінка разом з елементами з шаблону, але вони показані сірим кольором. Їх редагувати не можна. Можна редагувати те, що знаходиться в елементах Content.

Керуючий елемент Content – це основна складова сторінки контенту – це контейнер для інших керівників елементів, розміщених на сторінці контенту. Він не є автономних і використовується тільки в зв’язці відповідним елементів ContentPlaceHolder. На сторінці-спадкоємиці шаблону можуть бути тільки елементи типу Content, кожен з яких відповідає одному елементу ContentPlaceHolder шаблону. Не можна вставляти вміст поза цих елементів, інакше ASP. NET не зможе об’єднати головну сторінку зі сторінкою вмісту. Ідентифікатор ContentPlaceHolder повинен збігатися з атрибутом ContentPlaceHolderID відповідного елемента Content:

<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server"> </asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder2" Runat="Server"> </asp:Content>

Програма створила дочірню сторінку з двома елементами управління Content. Якщо переключитися на вкладку Design, ці два елементи Content будуть показані у вигляді порожніх прямокутників, розташованих поруч один з одним, так як у шаблоні вони знаходяться в двох сусідніх елементах таблиці.

Після того, як сторінка вмісту створена, можна запустити Web-додаток. Тепер при натиску на посилання, в області ContentPlaceHolder буде завантажуватися відповідна сторінка.

Використання майстер-сторінок є дуже потужним інструментом, що дозволяє значно спростити створення Web-програми, сторінки якого оформлені в єдиному стилі, і, що також важливо, підтримання даного додатку (оновлення, додавання елементів, зміна дизайну і т.д.). Проте, виникають завдання, вирішення яких при використанні майстер-сторінок викликає труднощі. До таких питань належать питання пов'язані з доступом до майстер-сторінки з коду сторінки вмісту, вкладенням майстер-сторінок один в одного і динамічне завдання майстер-сторінки.

Творці Web-додатків знають, що достатньо важливе значення в цій розробці грають заголовки сторінок, а також рядки метадискрипторів, від яких залежить індексація сторінки пошуковими роботами. Яким же чином можливо задавати заголовки сторінок вмісту, адже вони фактично вкладаються у майстер сторінку? Зробити це можна двома способами. Задавши заголовок сторінки в атрибуті Title директиви Page сторінки вмісту, як показано нижче.

<%@ Page Language="C#" AutoEventWireup="true" CodeFile= "Default.aspx.cs" Inherits="_Default" MasterPageFile="~/MasterPages/ Main-MasterPage.master" Title="Перша сторінка сайту" %>

Для цього необхідно отримати посилання на об'єкт Page для майстер-сторінки, до якої можливо звертатися з процедури обробки завантаження поточної сторінки таким чином:

Page masterPage = base.Master.Page;

masterPage.Header.Title = "заголовок встановлений програмно";

При побудові досить складного Web-додатку, може знадобитися зміна майстер-сторінки під час виконання. Це особливо актуально у випадку, коли зовнішній вигляд сторінки повинен змінюватися в залежності від дій користувача, наприклад, залежно від того, який користувач пройшов авторизацію в системі, або якщо необхідно передбачити можливість налаштування зовнішнього вигляду додатка в залежності від уподобань користувача. У будь-якому з цих випадків, для реалізації даної функціональності, можна створити кілька майстер-сторінок, які потім перемикати залежно від дій користувача.

Для зміни поточної майстер-сторінки програмним шляхом необхідно задати ім'я файлу нової майстер сторінки у властивості Page.MasterPageFile. Робити це необхідно у події Page.PreInit сторінки вмісту. При цьому, необхідно врахувати наступне. Майстер сторінки, на які будуть перемикатися сторінки вмісту, мають бути „сумісні" один з одним. Тобто вони повинні містити однакові елементи керування (однакового типу, з однаковими значеннями ID), в іншому випадку, станеться помилка при спробі звернення до якогось із них. Крім того, якщо сторінка вмісту, для якої передбачається динамі-

чна зміна майстер сторінки, повинна програмно звертатися до елементів майстер сторінки, це звернення необхідно здійснювати безпосередньо, без використання строго типізованого підходу на основі створення властивостей майстер сторінки, як було показано раніше. При цьому, використовувати директиву <%@ MasterType %> небажано.

Для того щоб зі сторінки вмісту звернутися до елементів управління головної сторінки, можна скористатися функцією FindControl. Безпосередньо звернутися до них не можна, так як це захищені члени.

```
Label mpLabel = (Label) Master.FindControl("masterPageLabel");
if(mpLabel != null)
{   //Set content page title to master page control
  Title.Text = mpLabel.Text
}
```

32.3. Користувальницькі і власні серверні елементи управління

Хоча набір стандартних елементів великий – завжди може знадобитися такий елемент, якого в стандартному постачанні немає. Або є сторінка з такою функціональністю, яку хочеться використовувати і на інших сторінках.

Користувальницькі елементи управління інкапсулюють кілька готових елементів в одному контейнері, який можна повторно використовувати в проекті. Вони успадковуються від класу UserControl – спадкоємця класу Control. Користувальницькі елементи компілюються точно так само, як і сторінки aspx.

Серверні елементи, крім цього, реалізують власну поведінку і самостійно виводять HTML-код, який їх відображає. Вони можуть успадковуватися від WebControl або одного з класів стандартних елементів управління. Їх можна використовувати в будь-яких проектах і поширювати у вигляді скомпільованої PE (Portable Executable) збірки.

Ми знаємо, що клас Page успадковує клас Control, як і всі елементи управління, деякі прямо, а інші через клас WebControl або HtmlControl. Отже, між написанням сторінки і розробкою власного елемента управління повинно бути багато спільного. У них теж є свій життєвий цикл. У класі Control визначені події Init, Load, DataBinding, PreRender, Unload, Disposed. Властивості, які Control надає своїм спадкоємцям, включають EnableViewState, ID, UniqueID, Page, Parent, SkinID, ViewState і Controls – колекція дочірніх елементів управління.

Клас Control надає можливість поміщати елемент управління в дерево елементів управління, які відображаються на сторінці .aspx. Клас Control також реалізує інтерфейс System.ComponentModel.IComponent, який робить компонент конструктивним. Конструктивний компонент може бути доданий в панель Toolboox візуального дизайнера, може бути поміщений на сторінку що розробляється методом drag-and-drop, може відображати властивості у вікні властивостей і забезпечувати інші види підтримки режиму розробки (у тому числі Smart Tags).

Користувальницькі елементи управління можна створювати у візуальному редакторі за тією ж моделлю, що і сторінки aspx. Як завжди, відкриємо діалог NewFile і виберемо тип сторінки Web User Control. Розширення файлу

з дизайном елемента – ascx, а з кодом класу – ascx.cs. На відміну від сторінок aspx, сам по собі користувальницький елемент не можна побачити в браузері, для цього він повинен знаходитися на якій-небудь сторінці:

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeFile="WebUserControl.ascx.cs" Inherits="WebUserControl" %>
```

Клас користувальницького елементу управління – спадкоємець System.Web.UI.UserControl. В іншому він нічим не відрізняється від файлу з класом сторінки:

```
public partial class WebUserControl : System.Web.UI.UserControl
{   }
```

Можна додати в нього будь-які елементи управління і HTML-код:

```
<h1><%= Greeting %>, <%= Name %>!</h1>
<asp:TextBox ID="txtName" runat="server"></asp:TextBox><br />
<asp:Button ID="btnClick" runat="server" Text="Button" />
```

У класі елемента управління визначимо його властивості:

```
string name;
   string greeting;
   public string Greeting
   {       get
      {          return greeting;       }
      set      {          greeting = value;       }
   }
   public string Name
   {       get      {          return name;       }
      set      {          name = value;       }
   }
  protected void Page_Init(object sender, EventArgs e)
  {
   btnClick.Text = "Enter your name and click";
  }
```

При натисканні на кнопку властивості елемента заповнюються даними з текстового поля:

```
protected void btnClick_Click(object sender, EventArgs e)
  {
   Name = txtName.Text;
  }
```

Тепер перетягніть назву елемента зі Solution Explorer на будь-яку сторінку.

Щоб використовувати користувальницький елемент на сторінці, його треба зареєструвати. Директива Register з’являється автоматично:

```
<%@ Register Src=" WebUserControl.ascx" TagPrefix="User"
TagName="GreetingControl" %>
```

Атрибут TagPrefix директиви Register задає префікс, за допомогою якого даний користувальницький елемент можна створювати на сторінках aspx. Його значення може бути будь-яким, крім asp, яке зарезервоване для вбудованих елементів управління ASP .NET. TagName — це ім’я елемента, що йде

після префікса; атрибут Src визначає шлях до файлу користувальницького елементу управління.

Тепер новий користувальницький елемент управління можна описати так:

<User:GreetingControl id="Hello" runat="server" Name="Heinrich" Greeting="Guten Tag"/>

Властивості, описані в класі, можна встановлювати в описі на сторінці, причому вони навіть будуть видні у вікні властивостей дизайнера.

Користувальницькі елементи повністю беруть участь у відображенні сторінки, і вставлені в нього елементи поводяться як звичайно. Під час життєвого циклу сторінки викликаються події вбудованого в неї елемента керування.

У коді сторінки можна маніпулювати його властивостями:

```
protected void Page_Load(object sender, EventArgs e)
 {
  Hello.Greeting = "Привет";
 }
```

Користувальницький елемент може отримати доступ до сторінки, в якій знаходиться, через властивість Parent. Якщо в батьківську форму додати TextBox, він може прочитати його значення і використовувати його:

```
<asp:TextBox ID="txtGreeting" runat="server"></asp:TextBox><br />
<User:GreetingControl id="Hello2" runat="server" Name="Heinrich" Greeting="Guten Tag" OnLoad="Hello2_Load"/>
 protected void btnClick_Click(object sender, EventArgs e)
 {
  Name = txtName.Text;
  TextBox tb=Parent.FindControl("txtGreeting") as TextBox;
  if ( tb!= null)
  {     Greeting = tb.Text;    }
 }
```

Тільки що створений елемент управління можна навіть додати в панель інструментів, і перетягувати звідти на будь-яку сторінку. Але директиву Register доведеться додавати самим.

Можна спробувати створити WebUserControl у події Page_Load, але це не дасть результату. Причина в тому, що клас оголошений лише частково у файлі відокремленого коду, в ньому не вистачає функції отрисовки, яка з’явиться при обробці сторінки ascx ASP.NET.

Щоб програмно створити екземпляр користувальницького елементу управління, потрібно викликати функцію LoadControl, який поверне екземпляр класу System.Web.UI.Control, що містить завантажуваний елемент управління. Щоб мати доступ до властивостей класу WebUserControl, потрібно перетворення типу. І, як і будь-який інший елемент керування, завантажений користувальницький елемент управління може бути доданий в колекцію елементів управління web-форми. Його не можна додати в сторінку, так як його складовою частиною є кнопка, яка може перебувати тільки у формі:

```
WebUserControl wuc = (WebUserControl)Page.LoadControl("WebUserControl.ascx");
```

wuc.Greeting = "Здоровенькі були";
wuc.Name = "Тарас";
form1.Controls.AddAt(0, wuc);

Більш корисний користувальницький елемент управління – нижній колонтитул будь-якої сторінки. Його можна помістити на шаблон дизайну. Він може відобразити юридичну інформацію, адреса web-майстра і дату останнього оновлення:

<%@ Control Language="C#" AutoEventWireup="true" CodeFile= "Footerl.ascx.cs" Inherits="Footerl" %>Copyright ©
*<asp:Label ID="lblYear" runat="server" /> by Your Company Name.
*
Зауваження, коментарі, проблеми? Зв’яжіться з web-майстром
*<asp:Label ID="lblEmail" runat="server" />
*
Дата останньої модифікації цієї сторінки: <asp:Label
*ID="lblLastMod" runat="server" />
*
Текст в елементах Label змінюється в оброблювачі Page_Load.
protected void Page_Load(object sender, EventArgs e)
{
lblYear.Text = DateTime.Now.Year.ToString();
lblEmail.Text = "webmaster";
lblLastMod.Text = System.IO.File.GetLastWriteTime(Server.MapPath (Request.Url.LocalPath)).ToLongDateString();
}

32.4. Серверні елементи управління

Серверні елементи управління, успадковані від WebControl або Control, складніше створювати, але у них ще більше можливостей. Такі елементи мають власні методи генерації HTML-коду. Складність у тому, що тут клас повністю описується програмістом, без візуального дизайну і файлу ascx. У класі WebControl визначені візуальні властивості, такі як BackColor, Font, ToolTip. У них можна визначити складну логіку користувача інтерфейсу. Якщо елемент керування не потребує таких властивостей, його потрібно наслідувати від Control. При цьому він генерує HTML-код, наприклад теги <meta> або приховані елементи.

Серверні елементи управління можуть бути побудовані по-різному. По-перше, вони можуть просто перевизначати метод RenderContents, так що під час виконання на їх місці з’явиться шматок коду HTML. По-друге, можуть створювати складні елементи, які служать контейнерами. Також можна наслідувати наявні елементи управління і додавати до них нову функціональність. За підтримки деяких інтерфейсів серверні елементи управління підтримують зв’язування з даними і створення шаблонів.

Серверні елементи управління розміщуються в бібліотеці WebControls. Щоб створити бібліотеку, в меню File виберіть New Project, і в діалозі, що з’явився – тип проекту Web Control Library (він знаходиться у вузлі Visual C # – Windows). У проекті вже створений найпростіший серверний елемент

управління. Бібліотеку можна створити тільки в Visual Studio, а в VWD – тільки клас в папці App_Code:

```
namespace WebControlLibrary1
{
 [DefaultProperty("Text")]
 [ToolboxData("<{0}:WebCustomControl1 runat=server> </{0}:WebCustomControl1>")]
 public class WebCustomControl1 : WebControl
 {
  [Bindable(true)]
  [Category("Appearance")]
  [DefaultValue("")]
  [Localizable(true)]
  public string Text
  {   get   {
     String s = (String)ViewState["Text"];
     return ((s == null) ? String.Empty : s);   }
    set   {   ViewState["Text"] = value;   }
  }
  protected override void RenderContents(HtmlTextWriter output)
 {   output.Write(Text);   }
 }
}
```

У цього елемента всього одна властивість Text, і він просто записує в потік виводу сторінки HTML значення цієї властивості.

Якщо в рішенні є проект з бібліотекою користувальницьких елементів, вони автоматично додаються в інструментальну панель (Toolbox). Для цього достатньо всього лише скомпілювати проект. У папці Bin з'являється WebControlLibrary1.dll. Це збірка, в якій знаходяться всі елементи управління бібліотеки.

Якщо ви працюєте з Visual Studio, все одно можна відкомпілювати класи в збірку. Dll з командного рядка1.

*csc /t:library /out: WebControlLibrary1.dll /r:System.dll /r:System.Web.dll *.cs*

В панелі інструментів з'явиться нова секція зі значками-шестерінками, і елементи управління можна перетягувати звідти на сторінки.

Директива Register, яка автоматично додається, буде містити назву цієї збірки і простір імен, в якому знаходиться елемент управління:

<%@ Register Assembly="WebControlLibrary1" Namespace= "WebControlLibrary1" TagPrefix="cc1" %>

Щоб не писати одну і ту ж директиву на багатьох сторінках, бібліотеку можна зареєструвати у файлі web.config.

У створеного елемента, крім властивості Text, є всі властивості зовнішнього вигляду і поведінки, як у стандартних елементів управління, як ви можете переконатися, відкривши його вікно властивостей. Він дуже схожий на елемент Label.

Доступ до збірки WebControlLibrary1.dll можна надати всім додаткам, якщо помістити її в глобальний кеш збірок.

32.5. Атрибути

Наявність атрибутів – важлива властивість мов .NET. З їх допомогою в метадані класу в збірку додається інформація, яка використовується різним чином. Атрибути – це класи, які успадковують System.Attribute і застосовуються до просторів імен, класам, властивостей і методів. Синтаксис застосування атрибута в C#:

[CustomAttr(Update:=true, Keep=false)]

Конструктор атрибута вказується у квадратних дужках із параметрами, які визначені в декларації класу атрибута.

В ASP.NET атрибути в тому числі визначають поведінку користувальницьких елементів управління (а також, наприклад, використовуються в описі web-сервісів). В описі WebCustomControl1 атрибути застосовані і до самого класу, і до властивості Text. Атрибути класу визначають його розміщення в панелі Toolbox і в дизайнері сторінок. Їх можна поділити на 3 категорії: атрибути, що допомагають середовищі розробки працювати з елементом управління в режимі дизайну; атрибути, керуючі висновком дочірніх елементів; атрибути, що визначають його поведінку в панелі інструментів.

DefaultProperty визначає властивість за замовчуванням. При відкритті вікна властивостей елемента керування в дизайнері позначена цим атрибутом властивість буде виділена бежевим.

ToolboxData задає форматуючий рядок. Наприклад,

ToolboxData("<{0}:WebCustomControl1 runat=server></{0}:WebCustomControl1>")

Рядок з формальним параметром, як значення якого підставляється атрибут TagPrefix директиви Register, яка реєструє даний елемент на конкретній сторінці.

Коли в дизайнері сторінки відбувається подвійний клік мишки на елементі, середовище розробки створює обробник події, наприклад, SelectedIndexChanged для списку, що випадає DropDownList. Яка саме подія, визначається атрибутом DefaultEvent.

Атрибути застосовуються також до властивостей і подій елемента.

Bindable вказує, що властивість можна пов’язати з джерелом даних. Category означає категорію у вікні властивостей елемента. Властивості розбиваються на категорії, якщо вибрати подання Categorized вікна властивостей.

Атрибут Themable визначає, чи може дана властивість визначатися у файлі скіна. За замовчуванням ASP. NET дозволяє всім властивостям написаних нами елементів управління бути описаними у файлах скінів. Це не завжди зручно, якщо властивість не відноситься до зовнішнього вигляду елемента. У такому випадку атрибут створюється з параметром false:

```
[Themeable(false)]
  public string Text
```

Browsable вказує, чи буде відображатися властивість у вікні властивостей, і EditorBrowsable – чи можливо буде його редагувати.

Існують і інші атрибути.

32.6. Фарбування (Rendering) елемента управління

У цьому прикладі побудований елемент управління, наслідуючий від Label. Він розфарбовує текст у випадкові кольори. Властивість EnableRainbowMode можна відключити, тоді він буде поводити себе як звичайна мітка:

```
[ToolboxData("<{0}:RainbowLabel runat=server></{0}:RainbowLabel>")]
 public class RainbowLabel : Label
 {
  [Bindable(true)]
  [Category("Appearance")]
  [DefaultValue("true")]
  [Localizable(true)]
  public bool EnableRainbowMode
  {     get     {
     if (ViewState["EnableRainbowMode"] == null)
      return true;
     else
      return
bool.Parse(ViewState["EnableRainbowMode"].ToString());     }
    set     {
     ViewState["EnableRainbowMode"] = value;     }
  }
  protected override void RenderContents(HtmlTextWriter output)
  {     if(EnableRainbowMode)
     output.Write(ColorizeString(Text));
   else
     output.Write(Text);
  }
  private string ColorizeString(string input)
  {     StringBuilder output = new StringBuilder(input.Length);
   Random rand = new Random(DateTime.Now.Millisecond);
   for (int i = 0; i < input.Length; i++)
   {     int color = rand.Next(0xffffff);
  string strColor = string.Format(@"<span style=""color: #{0:x}"">", color);
  output.Append(strColor);
  output.Append(input.Substring(i, 1));
  output.Append("</span>");     }
   return output.ToString();
  }
 }
```

32.7. Складові елементи управління

Складові елементи управління успадковуються від класу Composite Control. Цей елемент являє собою об’єднання текстового рядка з валідатором, який перевіряє його значення на відповідність шаблону адреси електронної по-

шти. EnsureChildControls – це метод, який перевіряє, чи існують вкладені елементи. Якщо ні, викликається метод CreateChildControl: [DefaultProperty ("Text")]

```
[ToolboxData("<{0}:EmailTextBox runat=server></{0}:EmailTextBox>")]
public class EmailTextBox : CompositeControl, INamingContainer
{   private TextBox textBox;
 private RegularExpressionValidator validator;
 [Bindable(true)]
 [Category("Appearance")]
 [DefaultValue("")]
 [Localizable(true)]
 [Themeable(false)]
 public string Text
 {    get    {
   EnsureChildControls();
   return textBox.Text;     }
  set    {
   EnsureChildControls();
   textBox.Text = value;     }
 }
 [Themeable(false)]
 public string ErrorMessage
 {    get    {
   EnsureChildControls();
   return validator.ErrorMessage;     }
  set
  {      EnsureChildControls();
   validator.ErrorMessage = value;     }
 }
 public override ControlCollection Controls
 {    get    {
   EnsureChildControls();
   return base.Controls;     }
 }
 protected override void CreateChildControls()
 {
  Controls.Clear();
  textBox = new TextBox();
  validator = new RegularExpressionValidator();
  Controls.Add(validator);
  Controls.Add(textBox);
  textBox.ID = "Email1";
  validator.ControlToValidate = textBox.ID;
validator.ValidationExpression=@"\w+([–+.']\w+)*@\w+([–.]\w+)*\.\w+([–.]\w+)*";
 }
}
```

У елемента управління EmailTextBox маються властивості Text і ErrorMessage, які можна визначати на сторінках aspx.

<cc1:EmailTextBox ID="EmailTextBox1" runat="server" Text="Hello" ErrorMessage="Адреса E-mail невірна!"/>

Контрольні запитання

1. Які існують основні вимоги щодо повторного використання сторінок?
2. Що являє собою еталонна сторінка?
3. Які основні ознаки еталонної сторінки?
4. Як використовується контейнер ContentPlaceHolder?
5. Що треба зробити для того, щоб перетворити звичайну сторінку Web-додатку в сторінку вмісту?
6. Для чого потрібен керуючий елемент Content?
7. У якій бібліотеці розміщуються серверні елементи управління?
8. Що таке атрибути?
9. Для чого потрібні складові елементи управління?
10. Якими способами можуть бути побудовані серверні елементи управління?

Тема 33. Навігація по веб-додатку. Персоналізація

Будь-який Web-додаток являє собою досить складну сукупність взаємопов’язаних сторінок. Добре продуманий і спроектований Web-додаток володіє не тільки красивим і ергономічним дизайном, а й хорошою системною навігацією, що дозволяє легко переходити від одного розділу до іншого, а також вирішувати інші завдання. ASP.NET має досить великими можливостями, що дозволяють реалізовувати складні системи навігації. Для реалізації систем навігації використовується цілий ряд елементів управління, призначених для вирішення окремих завдань. Розглянемо найбільш важливі і найпоширеніші з них.

Інфраструктура навігації по додатках має три основних компоненти:

- Карта сайту. Загальний програмний інтерфейс для надання ієрархічної структури веб-сайту. Цей об’єкт є доступним через об’єкт карти сайту. Він заповнюється інформацією, прочитаною з сховища даних, яка передається через компоненти провайдера карти сайту. За замовчуванням таким сховищем даних є XML-файл кореневого рівня з ім’ям app.sitemap.

- Навігація по сайту. Кожен визначений у карті сайт і вузол, пов’язаний з деяким, і розробники сторінки можуть легко отримати і використовувати в період виконання будь-якого розділу.

- Відображення структури сайту. Цей компонент візуалізує інтерфейс користувача навігації по сайту.

33.1. Карта сайту

Карта сайту – це логічний контейнер і ієрархічним набором вузлів, кожен з яких представляє сторінку додатка. Окремі сторінки зібрані в папки, яким можуть відповідати фізичні папки.

Карта сайту служить джерелом інформації для всіх елементів управління групи Navigation. З нею можна працювати програмно за допомогою класу SiteMap або через елемент управління – джерело даних SiteMapDataSource.

Вузли siteMapNode можуть вкладатися один в одного, створюючи ієрархію. Логіка вкладеності вузлів ніяк не пов'язана з фізичним розташуванням файлів. Кожен атрибут url у файлі .sitemap повинен бути унікальним.

Карти сайту доцільно використовувати у випадку, коли Web-додаток містить велику кількість сторінок. Карти сайту пропонують зручний механізм визначення структури сторінок додатку, а також її відображення за допомогою декількох елементів управління. Ці елементи управління розташовані в розділі Navigation панелі Toolbox. Основних елементів управління три – SiteMapPath, Menu, TreeView. Всі ці елементи призначені для вирішення однієї і тієї ж задачі – надання можливості користувачеві Web-програми здійснювати навігацію по сторінках. Різниця між ними полягає в способах відображення посилань на відповідні сторінки. Для використання будь-якого їх перерахованих компонентів, необхідно визначити структуру додатку, яка зберігається окремо.

Для додавання навігації по сторінках Web-додатку, насамперед, необхідно додати елемент Site Map в проект програми. При цьому автоматично створиться код карти сайту:

```
<?xml version="1.0" encoding="utf–8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap–File-1.0" >
   <siteMapNode url="" title=""  description="">
      <siteMapNode url="" title=""  description="" />
      <siteMapNode url="" title=""  description="" />
   </siteMapNode>
</siteMap>
```

Як видно з вихідного коду, карта сайту повинна починатися з кореневого вузла <siteMap>. Елементи структури описуються в тегах <siteMapNode>. За допомогою цих тегів можна вказувати ієрархію елементів Web-додатку. Для вказівки вкладених елементів їх просто необхідно розташувати всередині відповідного тега <siteMapNode>. Властивості кожного тега необхідні для задання відповідних значень. У прикладі вище видно, що кожному елементу відповідає три властивості: url, title, description. Їх призначення очевидно – url використовується для вказівки інтернет-адреси сторінки, якій відповідає цей елемент, title задає найменування елемента, що наведено елементом управління. Він використовується як текст гіперпосилання, створюваної в TreeView або Menu, description – опис елемента, яке відображається у вигляді спливаючої підказки (Tooltip) при наведенні покажчика миші на відповідний елемент.

Для сторінок в кореневій директорії досить вказати їх назву. Якщо сторінка знаходиться в піддиректорії, шлях вказується за допомогою прямого слеша.

Об'єкт siteMap завантажується зареєстрованим провайдером карти сайту. Провайдером за замовчуванням є клас XmlSiteMapProvider, який працює з конфігураційним Xml – файлом web.sitemap. Провайдер карти сайту – це будь-який клас, який реалізує інтерфейс ISiteMapProvider. Деякі властивості об'єкта siteMap наведено в таблиці 33.1.

Таблиця 33.1. Статичні властивості об’єкта siteMap

Властивість	Опис
CurrentNode	Отримує siteMapNode, що надає поточну потрібну сторінку
Provider	Повертає об’єкт провайдера сайту для поточної карти сайту
Providers	Повертає набір іменованих об’єктів провайдерів сайту для об’єкта siteMap.
RootNode	Отримує siteMapNode, що представляє сторінку верхнього рівня в стурктурі навігації по сайту.

Джерело даних SiteMapDataSource працює за допомогою провайдера SiteMapProvider. Це джерело даних не підтримує кешування, як інші. Він може бути пов’язаний тільки з файлом карти сайту.

У найпростішому вигляді оголошується так:

<asp:SiteMapDataSource ID="SiteMapDataSource1" Runat="server" />

Властивість ShowStartingNode визначає, чи буде елемент читати кореневий вузол карти сайту. Якщо властивість не встановлено, то в колекцію вузлів потраплять тільки дочірні елементи кореневого вузла.

StartFromCurrentNode = False задає можливість читати тільки вузли, починаючи з поточної сторінки.

Властивість FlatDepth задає кількість рівнів вкладеності, яке читається з карти сайту. За умовчанням це – 1, тобто читаються всі доступні рівні.

SiteMapProvider може бути корисним при написанні власних провайдерів карти сайту.

SiteMapViewType визначає форму подання вузлів. За умовчанням це Tree. Якщо значення дорівнює Path, то буде читатися шлях між кореневим вузлом і поточним, як в елементі управління SiteMapPath.

Після того, як карта сайту визначена, стає можливим використання елементів управління, пов’язаних з нею для відображення даних про структуру програми. Для цього можливе використання таких елементів управління як TreeView, Menu, SiteMapPath. Елементи TreeView і Menu мають однакову функціональність і призначені для виведення елементів структури програми на екран. Різниця полягає у формі подання даних. Елемент SiteMapPath призначений для відображення поточного становища користувача в ієрархії Web програми і дозволяє йому переходити вгору по ієрархії на більш високий рівень. Розглянемо приклади використання все трьох елементів управління.

33.2. Використання TreeView

Дані карти сайту організовані ієрархічно і, отже, можуть автоматично оброблятися ієрархічними елементами управління, такими як TreeView. Він може черпати інформацію як з будь-якого XML-файла через XmlDataSource, так і з карти сайту з допомогою SiteMapDataSource. Як випливає з його назви, TreeView показує дані у вигляді дерева, причому його вузли можна розкривати і закривати, вибирати окремі "листя". При цьому будуть запускатися події, які можна обробити.

TreeView складається з вузлів, які з’єднані між собою відносинами „батько-нащадок”. У одного батька може бути один або декілька нащадків. Вуз-

ли, у яких немає батьків, називаються кореневими. Їх в елементі управління може бути декілька. Вузли, у яких немає нащадків, називаються листям.

При декларації TreeView на сторінці вузли описуються тегами TreeNode. Допускається будь-який рівень вкладеності вузлів один в одного. Вузли елемента управління можна редагувати візуально, а деякі властивості TreeView наведені в таблиці 33.2

Таблиця 33.2. Статичні властивості об’єкта TreeView

Властивість	Опис
PopulateOnDemand	Якщо потрібно програмно додавати дочірні вузли, властивість PopulateOnDemand потрібно встановити в True.
ShowLines	ShowLines = "True" включає показ ліній, що з’єднують вузли
ImageUrl.	Замість тексту у вузлах можна показувати зображення, задані властивістю ImageUrl.
ImageSet	Властивість має набір визначених значків для різних типів вузлів

Якщо джерелом даних служить XmlDataSource, то його вузли можна прив’язати до елементу TreeView. За замовчуванням TreeView показуватиме назви вузлів, а не їх внутрішній вміст. У SmartTag виберіть пункт Edit TreeNode Databindings. Значення TextField використовується, якщо потрібно показати значення атрибутів вузла у вихідному файлі XML, а #InnerText вказує текст між відкриває і закриває тегами вузла.

Якщо вибираємо джерелом даних SiteMap, то на сторінці створюється ще один елемент управління – SiteMapDataSource. Змістовна частина цього меню знаходиться у файлі карти сайту, а форматування проводиться у властивостях TreeView. Властивість AutoGenerateDataBindings дозволяє автоматично заповнювати вузли інформацією з карти сайту.

33.2.1. Програмне управління TreeView

У TreeView є безліч подій. Подія SelectedNodeChanged запускається, коли користувач обирає вузол:

```
protected void TreeLibrary_SelectedNodeChanged(object sender,
EventArgs e)
   {
Label1.Text = "Ви обрали категорію " + TreeLibrary.SelectedNode.Text;
   }
```

Можна програмно розкривати і закривати вузли:

```
TreeView1.ExpandAll();
TreeView1.CollapseAll();
```

Подія TreeNodePopulate дозволяє динамічно заповнювати вузли, при цьому можна економити пам’ять, якщо заповнювати вузли тільки на вимогу після розкриття батьківського вузла. Подія TreeNodePopulate викликається, якщо дію (наприклад, розкриття) проведено з вузлом, у якого SelectAction налаштований на цю дію.

TreeView дозволяє не тільки показувати інформацію, але ставити прапорці поруч з вузлами. Це корисно, якщо в ньому міститься інформація про товари і користувач може обрати деякі з них. Властивість ShowCheckBoxes допускає 5 значень: None, Root, Parent, Leaf, All:

ShowCheckBoxes="Leaf"

При цьому поряд з вузлами-листям з'являються прапорці. Значення прапорців можна прочитати програмно.

33.3. Елемент управління Menu

Меню, що випадає можна створити засобами одного лише css. Це гарне рішення, але вимагає великого обсягу коду. Також необхідно передбачити можливість перегляду різними браузерами. Більшість розробників створюють меню за допомогою JavaScript. У ASP.NET 2.0 створення випадаючого меню будь-якого рівня вкладеності вимагає всього двох рядків:

```
<asp:SiteMapDataSource  ID="SiteMapDataSource1" runat="server" />
<asp:Menu ID="Menu1" runat="server"
DataSourceID="SiteMapDataSource1"></asp:Menu>
```

Menu ідеальний для відображення великої кількості ієрархічної інформації. Займаючи мало місця, вузли меню розкриваються при наведенні курсору миші. Стрілка поруч з пунктом меню означає, що в ньому є підменю. Замість стрілки можна використовувати зображення, задавши властивість DynamicPopOutImageUrl. Зображення також можна задіяти як роздільники між пунктами меню.

Menu допускає горизонтальну і вертикальну орієнтацію, яка задається властивістю Orientation. При горизонтальній орієнтації можна отримати смужку меню. Меню складається з статичної та динамічної частин, кожен – зі своїм набором стилів.

Властивість StaticDisplayLevels за замовчуванням дорівнює 1, тобто показуються тільки головні пункти меню, а інші з'являються в момент наведення мишки. Якщо це значення змінити, отримаємо статичне багаторівневе меню. На сторінці різні меню можна прив'язати до різних джерел SiteMapDataSource, тоді при одній карті сайту вони будуть показувати його різні підмножини.

Коли елемент управління прив'язаний до карти сайту, то пункти меню являють собою гіперпосилання на сторінки. Подія MenuItemClick дозволяє визначити поведінку сторінки при виборі пунктів меню, коли він заповнюється іншими способами, наприклад через XmlDataSource. У параметрі MenuEventArgs знаходиться інформація і про обраний пункті, і про батьківський сайт:

```
<script runat="server">
  protected void Menu1_MenuItemClick(object sender, MenuEventArgs e)
  {
    Listbox1.Items.Add(e.Item.Parent.Value + " : " + e.Item.Value);
  }
</script>
```

33.4. Елемент управління SiteMapPath

У ASP.NET надається готовий елемент керування для виведення ієрархії сторінок сайту. Налаштувавши провайдер карти сайту та визначивши реальну карту, необхідно додати на сторінку елемент управління SiteMapPath. Він складається з послідовності гіперпосилань на всі вищестоячі вузли сайту. Поточна сторінка відображена простим текстом. Це налаштування можна змінити, встановивши властивість RenderCurrentNodeAsLink в True.

Для того щоб на сторінці працював цей елемент, навіть не потрібно джерела даних. Він автоматично читає карту сайту з файлу Web.sitemap. Досить просто перетягнути його на сторінку. За замовчуванням елемент управління шляху карти сайту повертає рядок з усіма вузлами карти сайту та фізичними шляхами, розділеними знаком >. Кожен елемент візуалізується як гіперпосилання, яка вказує на відповідну сторінку. Зовнішній вигляд елемента керування можна налаштовувати 4 властивостями стилю (табл. 33.3), кожен з яких задається окремо: для кореневого елемента RootNodeStyle, для роздільника HoverNodeStyle, звичайного вузла NodeStyle та поточного вузла CurrentNodeStyle.

Таблиця 33.3. Статичні властивості об’єкта SiteMapPath

Властивість	Опис
PathDirection	дозволяє змінити напрямок від кореня до поточної сторінки на зворотний.
PathSeparator	задає роздільник між вузлами
ParentLevelsDisplayed	дозволяє обмежити кількість батьківських вузлів що відображаються. Якщо воно дорівнює – 1 (за замовчуванням), то показуються всі вузли.
Description	Якщо потримати курсор мишки над елементом, з’явиться підказка, текст якої береться з атрибуту description відповідного вузла карти сайту. Відключити відображення підказки можна за допомогою властивості ShowToolTips="false".

Зовнішній вигляд сайту дуже важливий. А він складається з зовнішнього вигляду окремих елементів. Але нам довелося б важко, якби не було іншого способу домогтися єдиного оформлення сторінок, окрім як призначаючи властивості, пов’язані із зовнішнім виглядом, для всіх елементів окремо. Але, на щастя, творці ASP.NET подбали про це, створивши функціонал тем.

33.5. Стилі елементів управління

За замовчуванням стиль елементів ASP.NET дуже простий – чорні букви на білому фоні. Щоб домогтися гарних дизайнерських ефектів, можна використовувати ті ж способи, що і при дизайні HTML-сторінок. Наприклад, форматувати текст за допомогою тегів <i>, <b> і так далі. Але так буде важко зберегти єдине стильове рішення на всіх сторінках великого сайту. Налаштування шрифтів за допомогою тега <font> теж вважається застарілим. Повсюди застосовуються CSS (каскадні таблиці стилів).

CSS (Cascading Style Sheets, каскадні таблиці стилів) – це набір параметрів форматування, що застосовується до елементів web-сторінки для управління їх видом і положенням.

При використанні таблиці пов’язаних стилів опис селекторів та їх властивостей розташовується в окремому файлі, як правило, з розширенням css, а для зв’язування документа з цим файлом застосовується тег <LINK>. Якщо визначити клас в заголовку через тег <style> або зовнішній файл, то стиль елемента керування легко можна змінити через властивість CssClass.

У Visual Studio є Style Builder, за допомогою якого можна легко створити бажаний стиль. З ним зручно працювати навіть новачкам. Стилі можна розробляти як для конкретного елемента, так і створювати для подальшого використання. У режимі Design оберіть елемент управління і з контекстного меню оберіть Style.

33.6. Зовнішні файли стилю

Стилі, як правило, зберігаються в одному або декількох зовнішніх файлах, посилання на які прописані у всіх документах сайту, або зберігаються разом з темою. Завдяки цьому зручно правити стиль в одному місці – при цьому оформлення елементів автоматично змінюється на всіх сторінках, які пов’язані із зазначеним файлом. Замість того, щоб модифікувати десятки HTML-файлів, досить відредагувати один файл зі стилем, і оформлення потрібних документів відразу ж зміниться.

Для того, щоб створити визначення стилю у зовнішньому файлі, додайте в проект новий файл. У діалозі "Новий файл" оберіть тип файлу StyleSheet. За замовчуванням він називається StyleSheet.css.

Елементи стилю можна визначати і за допомогою можливості IntelliSense. Вставте курсор всередині фігурних дужок і натисніть на Enter. З’являється меню, що випадає зі списком усіх можливих атрибутів стилю.

Готовий файл стилю можна призначити сторінці через вікно властивостей. Оберіть DOCUMENT з верхнього списку і призначте властивість StyleSheet. У полі цієї властивості має бути кнопка з трьома крапками. Натискання на цю кнопку приводить до діалогу, де можна вибрати будь-який з наявних у проекті файлів стилів. У код сторінки автоматично додається потрібне визначення:

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

Файл можна і просто перетягнути зі Solution Explorer.

33.7. Теми

Теми схожі на CSS тим, що теж дозволяють визначати зовнішній вигляд сторінок. Але теми можуть зробити набагато більше них. Таблиці стилів визначаються для тегів HTML, а скіни, які входять в тему, – для елементів управління. Скіни застосовуються на стороні сервера, тому можуть використовуватися для установки специфічних властивостей серверних контролів. Наприклад, для елемента керування Calendar такою властивістю є DayNameFormat. CSS ніяк не дозволяє оперувати такими властивостями. Теми можна застосовувати до сторінок, до сайту або окремих елементах управління.

Файли тем знаходяться в папці з зарезервованою назвою App_Themes. Цю папку можна створити, якщо в контекстному меню проекту вибрати "Add ASP.NET Folder". У папці App_Themes можна створювати теми, наприклад, для різних пір року. У кожної теми буде свій каталог, в якому будуть знаходитися відносяться до неї файли. На багатьох сайтах користувач може обрати тему. Вкладення тем один в одного не допускається.

Кожна тема зазвичай складається з одного або декількох файлів скінів з розширенням ".Skin", а також інших, необхідних для завдання зовнішнього вигляду сайту файлів, таких як файли каскадних таблиць стилів, картинок, XSL-перетворень і так далі, які також можуть бути впорядковані в піддиректоріях кореневої директорії теми. Файли скінів і таблиць стилів зазвичай розташовані в корені теми, а картинки – в піддиректорії Images.

Теми можна застосувати до сторінки з допомогою атрибуту Theme директиви Page. Тему можна поміняти програмно. Тому можна дати можливість користувачеві обрати тему. Тему можна встановити до або під час події PreInit:

```
protected void Page_PreInit(object sender, EventArgs e)
  {
     Page.Theme = "Black";
  }
```

Тему можна застосувати до всіх сторінок додатку, якщо у файлі web.config вставити цю директиву:

```
<configuration>
<system.web>
<pages theme=" Black " />
</system.web>
</configuration>
```

Якщо тема встановлена в сторінці, вона має перевагу перед глобальною темою. Теми сторінки перевизначають властивості елементів управління. Якщо потрібно, щоб теми не застосовувалися до елементу, потрібно встановити його властивість EnableTheming в False.

Якщо потрібно скасувати застосування теми до групи елементів, можна помістити їх в Panel і встановити його властивість EnableTheming в False.

Властивість EnableTheming можна міняти і на рівні сторінки:

```
<%@ Page Language="C#" AutoEventWireup="true" EnableTheming="False"%>
```

Скін – це шаблон елемента управління з визначенням набору візуальних властивостей, які будуть використовуватися для генерації елементів управління даної теми. Скіни можуть працювати разом з картинками і таблицями стилів. Один ".Skin"-файл може зберігати безліч різних елементів управління.

Контрольні запитання

1. Які три основні компоненти навігації по додатках?
2. Що таке „карта сайту”?
3. Для чого призначена властивість CurrentNode?
4. Яка властивість TreeView дозволяє автоматично заповнювати вузли інформацією з карти сайту?

5. За допомогою якої властивості Menu можна змінити відображення пунктів меню?

6. Що таке CSS?

7. За допомогою яких властивостей SiteMapPath можна змінити зовнішній вигляд елементу управління?

8. Як застосувати тему до сторінки Web-сайту?

9. Як створити визначення стилю у зовнішньому файлі?

10. Що таке скін?

Тема 34. Елементи-споживачі та постачальники даних

34.1. Елементи-споживачі даних (Data-Bound Controls). Синтаксис динамічного зв'язування

Прив'язка даних є дуже потужним засобом організації автоматичного відображення даних в елементі управління без використання програмування. Більшість елементів управління ASP.NET підтримують прив'язку даних. При цьому прив'язка даних може бути як даних з одним значенням, так із множинними значеннями. Прив'язка з одним значенням означає, що елемент управління може відображати єдине значення, що витягають із джерела даних. Такі принципи використовуються елементами управління TextBox, LinkButton, Image, HyperLink. Прив'язка з множинними значеннями означає, що елемент управління може відображати кілька значень, що витягують із джерела даних. Елементи управління, що підтримують прив'язку з множинними значеннями будуються на основі списків та електронних таблиць. Типовими представниками таких елементів управління є ListBox і GridView.

Для прив'язки елементів управління до джерела даних необхідно встановити значення властивості DataSource, в якому вказати найменування об'єкта, що містять дані необхідні для відображення. При установці властивості DataSource створюється логічний зв'язок між елементом управління, здатним відображати дані і об'єктом даних, що підлягають відображенню. Після того, як джерело даних визначено, необхідно заповнити його даними. Для цього використовується метод DataBind() елемента управління, що заповнюється даними, який витягає дані з джерела, проходить в циклі по джерелу, визначеному в DataSource і оновлює сторінку. Прив'язка з множинними значеннями є найбільш потужним типом прив'язки, так як в цьому випадку не потрібно програмувати циклічний перебір значень джерела даних і виведення їх на екран, ця логіка вже реалізована в елементі управління, що підтримує множинну прив'язку.

Можна отримувати дані, пов'язані з елементом управління, в його декларації на сторінці. Це робиться за допомогою різновиду блоку відображення. У ранніх версіях ASP. NET за допомогою такого механізму можна було тільки читати, але тепер можливо і двостороннє зв'язування. Особливо це важливо в елементах управління, що використовують шаблони. Хоча DataGrid і GridView автоматично відображають дані, але і в них для створення потрібних ефектів використовуються стовпці-шаблони.

34.2. Прив’язка з одним значенням

Елементи управління, що підтримують прив’язку даних з одним значенням дозволяють здійснити прив’язку деяких з їхніх властивостей до даних за допомогою виразу прив’язки даних. Вираз прив’язки даних вводиться в тексті сторінки. Aspx полягає всередині обмежувачів <% #%>.

У якості вираження прив’язки даних може бути використана загальнодоступна, або захищена змінна, а також будь-яке інше вираження, що може бути обчислено в момент виконання сторінки. Так, в якості вираження прив’язки даних допустимо використовувати функції, властивості, об’єкти, визначені в класі сторінки.

Після визначення і налаштування джерел даних для об’єктів на сторінці ASPX необхідно прив’язати дані до цих джерел. Для прив’язки даних до джерел даних можна використовувати методи Page.DataBind і Control.DataBind.

Обидва методи працюють схожим чином. Основна відмінність полягає в тому, що всі джерела даних прив’язуються до серверних елементів управління після виклику методу Page.DataBind. Поки не буде явним чином викликаний метод DataBind елемента керування або поки не буде викликаний метод сторінкового рівня Page.DataBind, ніякі дані передаватися елементу управління не будуть. Як правило, метод Page.DataBind (або метод DataBind) викликається з події Page_Load.

34.3. Прив’язка з множинним значенням

Для відображення набору значень необхідно використовувати прив’язку з множинним значенням. Прив’язаний набір значень повинен міститися в об’єкті, здатному зберігати колекції значень (наприклад об’єкти ArrayList, Hashtable та ін.).

Принципи використання всіх списових елементів, здатних відображати пов’язані дані одинакові. Для прив’язки такого елемента до джерела даних, необхідно встановити значення властивостей DataSource і DataTextField.

Якщо у функції Page_Load ми писали

ContinentDropDownList.DataSource = ContinentArrayList;

то на сторінці це можна зробити за допомогою

<asp: DropDownList id = "ContinentDropDownList" datasource = ‘<% #ContinentArrayList%> ‘runat = "server">

У ASP. NET, якщо встановлена властивість DataSourceID, то подія DataBinding викликається автоматично, властивості наведені в табл. 34.1.

Таблиця 34.1. Властивості об’єктів споживачів даних

DataSource	Об’єкт, що містить дані, які прив’язуються.
DataSourceID	Властивість, що містить ідентифікатор об’єкта, що використовується для підключення до джерела даних. На відміну від DataSource, значення DataSourceID можна встановити в режимі дизайну на етапі розробки програми.
DataTextField	Містить найменування стовпця (для рядка) таблиці або властивість (для об’єкта) елемента даних, що містить та відображається на екрані значення.

Продовження таблиці 34.1

DataTextFormatString	Містить необов'язковий рядок форматування, застосовувану для відображення даних.
DataValueField	Містить унікальний ідентифікатор, або поле первинного ключа, що використовується для вилучення даних, пов'язаних з вибираним користувачем елементом.

При цьому в локальному кеші створюється копія прочитаних даних. Для елементів Repeater, DataList, DataGrid синтаксис прив'язки даних розбирається у події ItemDataBound, для GridView – в RowDataBound. Ці події викликаються стільки разів, скільки записів у джерелі даних. Кожен раз, коли ASP.NET зустрічає ці роздільники в шаблоні, всередині елемента Item (RepeaterItem, DataListItem, DataGridItem) створюється елемент типу DataBoundLiteralControl, усередині якого записується вираз всередині роздільників. У обробнику події DataBinding цього елемента визначена змінна Container, яка вказує на цей самий Item, тобто секцію елемента. Класи Item зберігають дані у властивості DataItem. Тому в шаблонах доступ до даних відбувається за допомогою синтаксису Container.DataItem.

Споживачі даних відображають дані, отримані з класів-джерел даних. Вони надають багато корисних функцій. Наприклад, елемент управління GridView може не тільки показувати дані, а й сортувати, вибирати, редагувати їх. Якщо цієї функціональності недостатньо, її можна розширити, написавши власні обробники подій.

Елементи, які можуть бути пов'язані з елементами-джерел даних, різноманітні. По-перше, це вже добре знайомі DropDownList, ListBox, CheckBoxList, RadioButtonList, BulletedList. Однак для всіх них необхідно, як джерело даних вказувати не DataSource, а DataSourceID. Всі ці елементи відображати може тільки одне поле, вказане в DataTextField, з можливістю завдання другого як індексного у властивості DataValueField:

```
<asp:SqlDataSource ID="SqlDataSource3" runat="server"
ConnectionString="<%$ ConnectionStrings:NorthwindConnectionString %>"
SelectCommand="SELECT [CategoryName], [CategoryID] FROM [Categories]">
</asp:SqlDataSource>
<asp:DropDownList ID="DropDownList1" runat="server" DataSourceID="SqlDataSource3" DataTextField="CategoryName">
</asp:DropDownList>
```

AppendDataBoundItems – це нова властивість. Вона дозволяє комбінувати дані з елемента-джерела з даними, статично оголошеними на сторінці.

34.4. Елемент управління Repeater

Repeater в перекладі означає „той, хто повторює" – елемент-повторювач заданого шаблону для всіх полів джерела даних.

Шаблон – це безліч тегів HTML і серверних елементів управління, які задають зразок для відображення складової частини складного елементу

управління. DataGrid може використовувати шаблони чи ні, але Repeater без них існувати не може – сам по собі він не має візуального представлення. Таким чином, програміст сам визначає його зовнішній вигляд. Крім DataSourceID і DataMember, власних властивостей у нього немає. Тому у програміста є повний контроль над тим, як виводиться Repeater.

Як мінімум, повинен бути описаний шаблон ItemTemplate. HeaderTemplate який відображається один раз на початку відмальовки репітера, FooterTemplate наприкінці, SeparatorTemplate між відображенням кожного пункту, AlternatingItemTemplate – для парних пунктів. Всі серверні елементи управління в шаблон поміщаються цілком для того, щоб отримати таблицю, використовують прості теги HTML. Властивості елемента представлені в табл. 34.2.

Таблиця 34.2. Властивості елемента Repeater

Властивість	Опис
AltematingItemTemplate	Шаблон, використаний для рендеринга парних елементів DataMember
DataMember	Таблиця джерела даних, зв'язані з елементом управлінн
DataSourse	Джерело даних типу IEnumerable для заповнення списку
DataSourseID	Ім'я елемента управління представляє джерело даних для заповнення списку
FooterTemplate	Шаблон для рендеринга нижнього колонтитулу
Header Template	Шаблон для рендеринга верхнього колонтитулу
Items	Об'єкт RepeaterItemCollection – колекція об'єкту RepeaterItem, кожний із яких представляє один рядок даних
ItemTemplate	Шаблон для рендеринга елементів даних
SeparatorTemplate	Шаблон для рендеринга розд даних

Елемент управління Repeater заповнює вказаний список об'єктами даних, що містяться у зв'язаному з ним джерелі даних. Кожному об'єкту з цього джерела, ставиться у відповідність об'єкт RepeaterItem, який і додається в колекцію.

Клас RepeaterItem дозволяє представити будь-яку зі складових елемента управління (верхній або нижній колонтитул, роздільник), його застосування не обмежена тільки представленням елементів даних. Інтерфейс програмування RepeaterItem складають властивості, перераховані в таблиці 34.3.

Таблиця 34.3. Інтерфейс програмування RepeaterItem

Властивість	Опис
DataItem	Повертає об'єкт даних зі зв'язаного джерела даних, відповідно відображеному елементу. У елементу RepeaterItem, не зв'язаних з даними, таких як нижній і верхній колонтитули, ця властивість містить значення null.
ItemIndex	Повертає 0–базовий індекс виведеного елемента
ItemTipe	Повертає значення перечисленого типу ListItemTipe, який визначає тип виведеного елемента. Допустимі значення Header, Footer, Item, AltematingItem і Separator

Подібно будь-якому іншому елемента керування ASP.NET, пов’язаному з даними, Repeater генерує користувацький інтерфейс тільки у випадку, коли викликається його метод DataBind, а в ASP.NET ще й тоді, коли він пов’язаний з джерелом даних. DataBind – єдиний власний метод інтерфейсу класу Repeater, якщо не враховувати тих, які він успадковує від батьківських класів. Виклик даного методу змушує елемент управління перебудувати свою ієрархію елементів.

Виклик методу DataBind необхідний для генерування HTML-коду елемента управління і запису його у вихідний потік. Крім подій, визначених у батьківському класі, елемент управління Repeater генерує три додаткові події: ItemCreated, ItemCommand і ItemDataBound Обробника подій ItemCreated і ItemDataBound передаються посилання на створений об’єкт RepeaterItem. Будь-яка внесена в цей об’єкт зміна відображається на підсумковому висновку елемента управління Repeater.

34.5. Елемент управління DataList

DataList має ті ж риси, що й Repeater, тобто виводить дані згідно шаблонам. Однак це більш багатий елемент управління. По-перше, він підтримує вибір, редагування, видалення і вставку. Тому список шаблонів поповнюється SelectedItemTemplate і EditItemTemplate. Крім того, у нього є верхній і нижній колонтитули зі стилями HeaderStyle і FooterStyle.

По-друге, можна змінити способи відображення. За замовчуванням DataList виводить дані поколонно у таблицю. Властивість RepeatLayout, встановлена як Flow, прибирає табличні теги з вихідного потоку. RepeatDirection змінює напрямок виведення з вертикального на горизонтальний. RepeatColumns задає кількість стовпців таблиці, за замовчуванням дорівнює 1.

DataList – спадкоємець абстрактного класу BaseDataList, який успадковує WebControl. Тому у нього, на відміну від Repeater, маються візуальні властивості. При відображенні він являє собою таблицю, тому присутні властивості CellPadding і CellSpacing.

У DataList є шаблон за замовчуванням, який створює його у вигляді вертикально розташованих міток для кожного поля, а ліворуч від них поміщають текст з назвою поля. Щоб увійти в режим редагування шаблону, потрібно скористатися можливістю SmartTag – Edit Templates. Після того, як редагування закінчено, не забудьте вийти з режиму – End Template Editing.

Можна спроектувати цей елемент так, щоб у звичайному стані відображалася коротка інформація, а в обраному стані – більш докладна. Щоб реалізувати редагування, також треба обробити подію. Тому в ASP.NET DataList краще застосовувати для показу даних без редагування, а якщо редагування все ж потрібно – використовувати елемент управління FormView.

Властивість DataKeyField є й у DataGrid, і у DataList. За її допомогою відбувається зв’язування з ключовим полем таблиці даних.

34.6. Табличний і потоковий вивід

Властивість RepeatLayout приймає значення перечисленого типу RepeatLayout: Table (за замовчуванням) і Flow. Коли воно має значення Flow,

елемент управління DataList укладає шаблонний елемент в тег , а потім автоматично додає тег
, якщо елементи виводяться по вертикалі. У разі ж горизонтального виведення текст просто додається в кінець HTML-рядки. Далі наведено приклад виведення елемента керування для вертикального рендеринга з установкою Flow.

```
<span id="DataList1">
<span> шаблон верхнього колонтитула </span>
<br>
<span> шаблон елемента </span>
<br>
<span> шаблон роздільника </span>
<span> шаблон нижнього колонтитула </span>
</span>
```

Режим Table позбавляє вас від необхідності HTML-кодування табличної розмітки. Ось як виглядає висновок елемента управління DataList в цьому режимі:

```
<table id="DataList1">
<tr>
<td> шаблон верхнього колонтитула </td>
</tr>
<tr>
<td> шаблон елемента </td>
</tr>
<tr>
<td> шаблон роздільника </td>
</tr>
```

Вивід елемента управління DataList може бути відформатований у вигляді декількох стовпців – їх кількість задається у властивості RepeatColumns. За умовчанням текст виводиться в один стовпець. Для випадків коли їх має бути кілька, підходить і табличний, і потоковий рендеринг, але табличний, звичайно зручніше.

```
<asp:datalist runat="server" id="list" RepeatColumns="3"> </asp:datalist>
```

34.7. Елемент управління DataGrid

Це дуже популярний елемент управління, і не дивно. DataGrid робить дуже легким уявлення табличній інформації, яка міститься в базах даних, файлах XML або створюється вручну. Досить створити DataGrid, встановити властивість DataSource і отримати готову таблицю на сторінці. Формат таблиці можна міняти незалежно від даних. Дані можна сортувати, вибирати, редагувати.

У найпростішому варіанті потрібно встановити тільки властивість DataSource, її значенням може бути об’єкт, який реалізує інтерфейс IEnumerable, наприклад SqlDataReader, DataTable. При цьому на сторінці виводиться таблиця, де рядкам відповідають записи, а стовпцям – поля.

За замовчуванням елемент DataGrid сам визначає кількість полів у джерелі даних і генерує колонки таблиці. Це визначається властивістю AutoGenerateColumns. З елементом управління DataGrid можуть бути

пов’язані не всі типи даних. Підтримуються примітивні типи, рядки, DataTime і Decimal. Якщо в полі не підтримуваний тип, стовпець не буде створений. Якщо жодного відповідного поля немає, буде викинуто виняток.

DataGrid має заголовок (Header), який за замовчуванням видно, і нижній колонтитул (Footer). Під час автоматичної генерації в заголовку кожного стовпця виводиться назва поля.

Якщо AutoGenerateColumns встановити в False, можна самим управляти колонками і визначати більш складний його вигляд. У такому випадку треба включити в DataGrid елементи BoundColumn. Деякі властивості BoundColumn:

- DataField визначає поле джерела даних;
- DataFormatString задає формат виведення даних;
- ReadOnly робить поле недоступним для редагування.

У заголовку і нижньому колонтитулі можна встановити будь-який текст, а в заголовку – ще й картинку (HeaderText, FooterText, HeaderImageUrl).

У чарунці таблиці вставляється LiteralControl, текст якого береться з джерела даних і форматується відповідно до DataFormatString. Для рядка, що редагується в осередку, з’являється TextBox. Є й інші типи колонок. ButtonColumn відображає в кожному рядку командну кнопку. Якщо пов’язати її з полем, на кнопках будуть написи з цього поля. EditCommandColumn показує кнопки для редагування. HyperLinkColumn перетворює текст в гіперпосилання. Наприклад, поле PhotoPath можна показати в такій колонці, і тоді клацання по посиланню покаже фотографію. TemplateColumn дозволяє визначити шаблон відображення, як у DataList.

При бажанні можна програмно приховувати і показувати колонки, наприклад:

DataGrid1.Columns [1]. Visible =! (DataGrid1.Columns [1]. Visible);

У елемента DataGrid є 7 властивостей, які задають стилі різних його частин або типів рядків. Всі вони мають тип TableItemStyle. Це AlternatingItemStyle, EditItemStyle, FooterStyle, HeaderStyle, ItemStyle, PagerStyle і SelectedItemStyle. Стилі утворюють ієрархію, тобто атрибут „Стиль”, який вище в ієрархії, успадковує ті, які нижче, якщо він його не перевизначає. Порядок в ній такий:

1. EditItemStyle – стиль рядка, що редагується;
2. SelectedItemStyle – стиль вибраного рядка;
3. AlternatingItemStyle – стиль кожного другого рядка;
4. ItemStyle – стиль рядка за умовчанням;
5. ControlStyle – всі властивості, які впливають на зовнішній вигляд елемента, наприклад BackColor. PagerStyle, FooterStyle, HeaderStyle теж його успадковують.
6. PagerStyle – стиль пейджера, тобто номерів сторінок – гіперпосилань, при виборі яких таблиця перегортається. Щоб пейджер з’явився, повинен бути встановлений атрибут AllowPaging і кількість записів має бути більше PageSize. Всі ці властивості зручно встановлювати за допомогою PropertyBuilder.

34.8. Елемент управління GridView

Елемент управління GridView є вдосконаленим елементом, покликаним замінити DataGrid. Все сказане про DataGrid відноситься до GridView, але з трохи іншими назвами. Так, замість BoundColumn вживається BoundField, а в назвах стилів замість Item знаходиться Row. Таким чином, будь DataGrid можна перетворити в GridView, але не навпаки. Хоча в найпростішому варіанті GridView відображає таку ж таблицю, спадкоємець не DataGrid, а CompositeDataBoundControl.

Головна перевага GridView – автоматичне зв'язування з даними, завдяки чому немає необхідності писати обробники подій, щоб забезпечити функціональність, таку як видалення, редагування, сортування, розбиття на сторінки за умови зв'язування з елементами-джерелами даних. Він забезпечує стандартну обробку подібних подій, але її завжди можна розширити, щоб забезпечити додаткові можливості.

GridView разом з SqlDataSource з'явиться простим перетягуванням таблиці Users на форму.

Установка властивості AllowSorting створює в заголовку гіперпосилання, при натисканні на таблиця яка буде сортуватися по обраному полю. У цьому виявляється перевага перед DataGrid, в якому для сортування необхідно перевизначити подію SortCommand.

Після повторного натискання на заголовок таблиця сортується по цьому полю в спадному порядку.

Іноді значення в якому-небудь полі можуть повторюватися, і потрібно відсортувати значення спочатку по першому полю, потім по другому. Але сортування за замовчуванням сортує тільки по одному полю. Якщо зараз натиснути мишкою на Comments, сортування по імені пропаде.

Елементи управління, які описані в цій лекції, мають по дві події для кожної операції. Одна з них виникає до початку виконання операції, друга – після. Наприклад, RowUpdated – коли користувач оновлює запис, відбувається подія RowUpdating, і виконується команда Update джерела даних і потім відбувається подія RowUpdated.

У RowUpdating ми можемо дізнатися, яка інформація надсилається джерелам даних і змінити її, а можливо і скасувати оновлення.

У обробнику події RowUpdating передається аргумент типу GridViewUpdateEventArgs. Його властивість RowIndex визначає індекс поточного рядка. Щоб скасувати операцію, встановіть властивість Cancel в true.

У колекціях OldValues і NewValues містяться старі і нові значення полів. У цьому прикладі перед оновленням, дані кодуються з метою захисту від хакерів.

```
protected void GridView1_RowUpdating(Object sender,
GridViewUpdateEventArgs e)
   {       foreach (DictionaryEntry entry in e.NewValues)
      {          if(entry.Value!=null)
           e.NewValues[entry.Key] =
Server.HtmlEncode(entry.Value.ToString());       }
   }
```

Подію RowUpdated можна використовувати щоб обробляти помилки, при введенні даних. У таблиці Customers поле CompanyName не допускає порожнього значення. Якщо користувач його не введе, буде викинуто виняток.

У програмному режимі можна маніпулювати колекцією Rows, яка складається з об'єктів типу GridViewRow. Рядки GridView мають властивості RowState и RowType. RowState може приймати значення з перерахування: Normal, Edit, Alternate, Insert и Selected.

Дані, які зберігаються в рядку, можна отримати з колекції DataItem, яка індексується назвою полів.

Можна пов'язати GridView із представленням (View), але уявлення не підтримується редагуванням.

34.9. Елемент управління DetailsView

Він показує кожен раз один запис з джерела даних в таблиці з двома стовпцями, де зліва відображається назва поля, а праворуч – значення. Так само як і GridView, DetailsView дозволяє виробляти розбивку на сторінки, редагування і видалення з автоматичним зв'язуванням з джерелом даних. Він також є спадкоємцем CompositeDataBoundControl.

На відміну від GridView, DetailsView дозволяє вставляти записи. Для цього потрібно встановити значення властивості AutoGenerateInsert Button = "True". При відображенні з'явиться кнопка New. Її натискання переводить елемент в режим вставки, за замовчуванням для кожного поля генеруються TextBox.

Якщо джерело даних для DetailsView – SqlDataSource, то у нього повинні бути визначені властивості InsertCommand і набір параметрів.

У DetailsView є пари подій, які відбуваються при зв'язуванні з даними, при переході з режиму перегляду в режим вставки, при гортанні сторінки.

34.10. Елемент управління FormView

Ще один новий елемент FormView схожий на DetailsView, але відрізняється від нього тим, що потребує у шаблоні для свого представлення.

34.11. Елементи – джерела даних (Data Source Controls)

Для роботи з даними в ASP.NET існують дві групи спеціальних елементів управління. Перша призначена для того, щоб здійснювати зв'язок з джерелами даних. Друга група служить для відображення даних.

Об'єкти даних DataAdapter і DataConnection безпосередньо зв'язуються формою. Зараз це теж можливо, але технологія змінилася. Введена нова об'єктна модель джерел даних. Класи-джерела даних забезпечують кращу абстрактізацію, ніж використання класів ADO.

Ці елементи полегшують роботу з ADO. NET, інкапсулюючи роботу із з'єднаннями, командами і адаптерами. Вони реалізують інтерфейс IDataSource, в якому визначений базовий набір можливостей роботи з джерелами даних. Більшість цих класів надають функціональність для читання і запису. Вони є обгортками об'єктів ADO. NET. У попередніх версіях треба було створювати об'єкти ADO самим і пов'язувати елементи – управління з ними за допомогою команди DataBind. Наприклад:

```
<asp:BulletedList ID="BulletedList1" runat="server"
BulletStyle="Square" DataTextField="CategoryName"
DataValueField="CategoryID">
</asp:BulletedList>
protected void Page_Load(object sender, EventArgs e)
{
   SqlConnection conn = new SqlConnection(@"Data
Source=(local)\sqlexpress;Initial Catalog=Northwind;Integrated
Security=True");
   SqlCommand cmd = new SqlCommand("SELECT CategoryID,
CategoryName FROM Categories", conn);
   SqlDataAdapter da = new SqlDataAdapter(cmd);
   DataSet ds = new DataSet();
   da.Fill(ds);
   BulletedList1.DataSource = ds;
   BulletedList1.DataBind();
}
```

Тепер елементи управління зв'язуються з елементом-джерелом допомогою властивості DataSourceID. Будь-який клас-джерело даних може бути пов'язаний майже з будь-яким класом для відображення даних, і це надає велику гнучкість.

Всього в ASP. NET 5 елементів-джерел даних:

1) SqlDataSource, SqlDataSource дозволяє з'єднуватися з більшістю реляційних СУБД. Sql в назві класу означає, що служить для з'єднання з базами, які розуміють мову запитів Sql, а не тільки з MS SQL Server.

2) AccessDataSource. AccessDataSource птимізований для роботи з базами Access. Наприклад:

```
<asp:AccessDataSource ID="AccessDataSource1" runat="server"
DataFile="~/App_Data/guestbook.mdb" SelectCommand="SELECT [WriteDate],
[UserName], [UserMail], [Message] FROM [guestbook]"> </asp:AccessDataSource>
```

3) ObjectDataSource для роботи з табличними джерелами даних.

4) XmlDataSource і SiteMapDataSource – для роботи з ієрархічними даними. Ця спеціалізація XmlDataSource, працює з файлами навігації по сайту і служить джерелом даних для елементів управління навігації.

Елементи-джерела даних призначені для двостороннього обміну даними, тобто як для читання, так і для запису. Самі по собі вони нічого не відображають. Дані будуть доступні підключеним до них елементів управління.

34.11.1. SqlDataSource

SqlDataSource об'єднує в собі можливості SqlConnection і SqlDataAdapter (плюс додаткові). У властивості DataSourceMode SqlDataSource задається, за допомогою DataReader або DataSet виходять дані. Під час читання за допомогою DataReader деякі можливості не підтримуються.

Отримання даних пов'язано з властивостями, схожими на властивості SqlDataAdapter:SelectCommand, SelectCommandType, DeleteCommand, Delete-

CommandType і так далі. SelectCommandType може бути 2 типів – Text і StoredProcedure. Команди виконуються, коли викликаються відповідні методи.

Метод Select викликається з параметром типу DataSourceSelectArguments і повертає DataSet або IDataReader залежно від значення властивості DataSourceMode, інші ж методи викликаються без параметрів і повертають кількість оброблених рядків. Метод Select немає необхідності викликати явно. Він викликається автоматично, коли пов'язаному з SqlDataSource елементу потрібні дані для відображення.

Процедури сервера зазвичай мають параметри. Тому передбачені також колекції параметрів. Самі параметри можуть бути звичайними або пов'язаними з елементами управління.

Значення параметрів можна отримувати і з інших різноманітних джерел. Є кілька класів параметрів-спадкоємців класу Parameter:CookieParameter використовує значення ключа файлу cookie, FormParameter – змінних форми, QuerystringParameter – адресного рядка, ProfileParameter – профілю користувача і SessionParameter – змінної сесії:

<asp:Parameter Name="UID" Type="Int32" DefaultValue="0" />

Такий тип параметрів застосовується, якщо SqlDataSource використовується як джерело даних для елементів з автоматичним зв'язуванням – GridView, FormView, DetailsView. Значення параметра передається у всередині цих елементів.

В інших випадках задіюється ControlParameter, тобто значення параметра береться з елемента керування. Також задається властивість, звідки і береться значення. Хоча якщо це Text, його можна не писати. Властивість CancelSelectOnNullParameter визначає, чи буде перерваний запит, якщо значення будь-якого параметра Null.

34.11.2. Вказівка джерела даних

Для роботи з джерелом даних за допомогою керованих провайдерів даних контрол SqlDataSource використовує нововведення від ADO.NET під назвою „фабрика керованих провайдерів даних", призначене для зручності створення додатків, що не залежать то баз даних. Але для цього окрім самого рядка підключення контрол також повинен знати і використовуваний керований провайдер доступу до даних. Для вказівки цього провайдера використовується властивість Provider. Ну а рядок підключення вже за звичкою пишеться у властивість ConnectionString.

<asp:SqlDataSource ID="SqlDataSource1" runat="server" ConnectionString="Data Source=DIMONSERVER; InitialCatalog=Northwind; Persist Security Info=True; UserID=sa;Password=" ProviderName= "System.Data.SqlClient"> </asp:SqlDataSource>

Крім того рядок підключення до джерела даних можна задати з використанням ще одного нововведення. NET 2 – секції connectionStrings файлу web.config. При цьому при завданні рядки підключення в web.config з самим рядком можна задати і використовуваного провайдера, що автоматично звільняє нас від завдання провайдера при визначенні SqlDataSource:

<connectionStrings>
<add name="NorthwindConnectionString" connectionString="DataSource= DIMONSERVER;Initial Catalog=Northwind;Persist Security Info=True; UserID= sa; Password=" providerName="System.Data.SqlClient" /> </connectionStrings>
<asp:SqlDataSource ID="SqlDataSource1" runat="server" Connection-String="<%$ ConnectionStrings:NorthwindConnectionString %>">
</asp:SqlDataSource>

При цьому використовується ще одне нововведення ASP.NET 2 – динамічні вирази. Але успішне підключення до джерела даних – це тільки лише необхідне, але ніяк не достатня умова для отримання потрібних даних.

34.11.3. Отримання даних в SqlDataSource

Для завдання запиту для отримання даних у ньому властивість SelectCommand разом із супутніми їй властивостями SelectCommandType і SelectParameter. Тип набору даних, що повертаються (і внутрішній клас, який використовується для зберігання даних) визначає властивість DataSourceMode, що приймає значення DataSet або DataReader. Почнемо природно з найпростішого – з одержання повного списку клієнтів компанії. Для цього нам всього лише потрібно вказати соотвествующий запит у властивості SelectCommand.

<asp:SqlDataSource ID="SqlDataSource1" runat="server" ConnectionString="<%$ ConnectionStrings:NorthwindConnectionString %>" SelectCommand="SELECT [CustomerID], [CompanyName], [ContactName], [ContactTitle], [City], [Country] FROM [Customers]"> </asp:SqlDataSource>

Аналогічно у випадку, якщо у нас вже є збережена процедура (наприклад, ListCustomers), що повертає результат цього запиту, то досить задати її назва у властивості SelectCommand і встановити властивість SelectCommandType в значення „StoredProcedure”.

<asp:SqlDataSource ID="SqlDataSource1" runat="server" Connection-String="<%$ ConnectionStrings:NorthwindConnectionString %>" SelectCommand="ListCustomers" SelectCommandType="StoredProcedure">
</asp:SqlDataSource>

В принципі все досить тривіально і нічого в цьому складного немає. Тож трохи ускладнимо завдання і ознайомимося з завданням параметрів для запиту. Припустимо, що нам необхідно виводити на сторінці клієнтів тільки з певної країни. Відповідно нам потрібно змінити запит, додавши до нього параметр @Country.

<asp:SqlDataSource ID="SqlDataSource1" runat="server" Connection-String="<%$ ConnectionStrings:NorthwindConnectionString %>" SelectCommand="SELECT [CustomerID], [CompanyName], [ContactName], [ContactTitle], [City], [Country] FROM [Customers] where Country = @Coutry"> <SelectParameters> <asp:Parameter Name="Coutry" /> </SelectParameters>
</asp:SqlDataSource>

Але просто вказати параметр недостатньо – для отримання даних необхідно також вказувати його значення. А значення цей параметр може приймати з найрізноманітніших джерел – з адресного рядка, куки, сесії, профілю користувача, змінних форми і серверних контролов. Наприклад, у разі отримання значення з адресного рядка оголошення параметра буде таким:

<asp:QueryStringParameter Name="Coutry" QueryStringField="Country" />

А для отримання значення із контрола TextBox форми потрібно написати приблизно так:

<asp:ControlParameter ControlID="TextBox1" Name="Coutry" PropertyName="Text" />

При цьому при завданні параметра можна вказати додаткові властивості – DefaultValue для вказівки значення за замовчуванням, Type – для вказівки типу даних значення і т.д. Крім того, не менш важливу роль при складанні запиту для отримання даних відіграє властивість SqlDataSource.CancelSelectOnNullParameter, яка визначає переривати чи виконання запиту в разі, коли якийсь з параметрів дорівнює null, а також властивість Parameter.ConvertEmptyStringToNull, яка вказує на необхідність конвертації порожніх значень параметрів у null. Наприклад, якщо ми хочемо мати можливість у попередньому прикладі отримувати також список всіх клієнтів компанії незалежно від їхньої країни, то необхідно переробити опис SqlDataSource на ось таке:

<asp:SqlDataSource ID="SqlDataSource1" runat="server" CancelSelectOnNullParameter="False" ConnectionString="<%$ ConnectionStrings: NorthwindConnectionString %>" SelectCommand="SELECT [CustomerID], [CompanyName], [ContactName], [ContactTitle], [City], [Country] FROM [Customers] where @Country is null or Country = @Country">

<SelectParameters>

<asp:QueryStringParameter Name="Country" QueryStringField= "Country" ConvertEmptyStringToNull="True" />

</SelectParameters>

</asp:SqlDataSource>

І тепер запит сторінки без вказівки параметра Country або із зазначенням порожнього значення повертатиме список всіх клієнтів. Як бачите, отримати дані з бази за допомогою контрола SqlDataSource дуже просто. Але що робити в ситуації, коли через велику кількість однакових запитів неприпустимо виростає навантаження на базу даних? У цьому випадку на допомогу приходить кешування.

Кешування потрібно для збільшення ефективності роботи з даними. При включеному кешуванні SqlDataSource „запам’ятовує” велику кількість записів на заданий час, навіть якщо всі дані не відображаються відразу, але можуть знадобитися при гортанні. Наприклад, студент читає першу сторінку лекції, логічно, що він незабаром перейде до другої. Якщо сервер зберігає в оперативній пам’яті всі сторінки лекції, він їх оперативно видасть, не звертаючись до бази.

При включеному кешуванні дані читаються „великим шматком”, а вибірка записів відбувається за допомогою фільтрування.

У SqlDataSource кешування і сортування можливі тільки при отриманні даних через DataSet. Якщо DataSourceMode одно DataReader, а EnableCaching – True, буде викинуто виключення NonSupportedException.

Тривалість кешування можна задати у властивості CacheDuration, це може бути певна кількість секунд або Infinite, тобто дані ніколи не оновлюються. Поведінка кешування залежить від поєднання властивостей CacheDuration і CacheExpirationPolicy. Якщо значення CacheExpiration Policy Absolute, то елемент запитує інформацію через проміжки часу, визначені в CacheDuration, а стару стирає з пам'яті. Якщо CacheExpirationPolicy дорівнює значенню Sliding, то SqlDataSource починає відлік часу після кожного запиту до нього. Дані з кешу застарівають, якщо протягом часу CacheDuration не було жодного Select-запиту.

Нам же цікавіше те, що контрол SqlDataSource кешує дані з урахуванням параметрів, тобто для наведеного вище мною прикладу виведення клієнтів по країнах для кожного значення параметра Country в кеші буде створена окрема запис. У деяких випадках це зручно, але в даному випадку було б більш цікаво закешувати в пам'яті весь набір даних і вже з закешованого набору вибирати тільки потрібні дані. І це теж можна зробити за допомогою властивості FilterExpression, що задає вираження для фільтрації отриманих даних і колекція FilterParameters, що задає, як і колекція ControlParameters, значення для фільтра.

Але тут є і свої особливості. Вираз в FilterExpression задається аналогічно висловом DataView.RowFilter (насправді в глибинах SqlDataSource для фільтрації як раз таки використовується DataView), відповідно в задаваємій в цій властивості рядку не може бути й мови ні про які SQL-подібних параметрах. А для завдання параметрів потрібно використовувати той же принцип, який застосовується при форматуванні рядків – вирази типу {0}, {1} і т.д. І повертаючись до нашого завдання, значення властивості FilterExpression прийме ось такий вигляд – FilterExpression ="Country = '{0}'". А значення параметрів в цей рядок підставлятимуться з урахуванням порядку їх оголошення в колекції FilterParameters.

Ну, а повний опис контрола SqlDataSource, кешируючого одержуваний список клієнтів на 60 секунд і використовує фільтрацію по країнам буде таким:

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server" ConnectionString="<%$ ConnectionStrings:NorthwindConnectionString %>" SelectCommand="SELECT [CustomerID], [CompanyName], [ContactName], [ContactTitle], [City], [Country] FROM [Customers]" CacheDuration="60" EnableCaching="True" FilterExpression="Country = '{0}'">
<FilterParameters>
<asp:ControlParameter ControlID="TextBox1" DefaultValue="USA" Name="Coutry" PropertyName="Text" />
</FilterParameters>
</asp:SqlDataSource>
<asp:GridView ID="GridView1" runat="server" AllowSorting="True" AutoGenerateColumns="False" DataSourceID="SqlDataSource1">
```

<Columns>

<asp:BoundField DataField="CustomerID" HeaderText="CustomerID" ReadOnly="True" SortExpression="CustomerID" />

<asp:BoundField DataField="CompanyName" HeaderText="CompanyName" SortExpression="CompanyName" />

<asp:BoundField DataField="ContactName" HeaderText="ContactName" SortExpression="ContactName" />

<asp:BoundField DataField="ContactTitle" HeaderText="ContactTitle" SortExpression="ContactTitle" />

<asp:BoundField DataField="City" HeaderText="City" SortExpression= "City" />

<asp:BoundField DataField="Country" HeaderText="Country" SortExpression="Country" />

</Columns>

</asp:GridView>

<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>

Тепер якщо відкрити сторінку з цим кодом і запустити для контрола SQL Profiler, то ми побачимо, що запити до бази для отримання списку клієнтів будуть виконуватися тільки в разі спорожнілого кеша (тобто не частіше, ніж раз на хвилину).

34.11.4. Зміна даних в SqlDataSource

Контрол SqlDataSource вміє не тільки отримувати дані з джерела даних, а й уміє оновлювати ці дані у джерелі даних. Він використовує для цієї властивості UpdateCommand, DeleteCommand і InsertCommand, настройка яких в принципі аналогічна настройці SelectCommand для отримання даних. Але є в цій настройці і свої нюанси.

Для оновлення даних у джерелі даних (як і для їх отримання) контрол SqlDataSource використовує параметризовані запити або збережені процедури. Але при цьому в колекцію параметрів для update і delete запитів додаються також параметри з оригінальними значеннями оновлюваних даних у зазначеному у властивості OriginalValuesParameterFormatString форматі (за умовчанням цей формат має вигляд "original_{0}", тобто параметри з оригінальними значеннями мають префікс original_). Які саме параметри потрапляють в цей список визначає значення властивості ConflictDetection – при CompareAllValues в запити передаються оригінальні значення всіх оновлюваних полів, а при OverwriteChanges – тільки ключових. Ознака того, що поле ключове визначається поза контрола SqlDataSource (у разі прив’язки контрола SqlDataSource до нових табличним Контролем типу GridView список ключових полів визначається значенням властивості GridView.DataKeys).

Початок опису контрола SqlDataSource вже нам знайомий:

<asp:SqlDataSource ID="SqlDataSource1" runat="server" ConnectionString="<%$ ConnectionStrings:NorthwindConnectionString %>"

SelectCommand="SELECT [RegionID], [RegionDescription] FROM [Region]"

Так як GridView підтримує редагування і видалення даних, але не підтримує їх додавання необхідно визначити тільки Update і Delete команди:

DeleteCommand="DELETE FROM [Region] WHERE [RegionID] = @original_RegionID"

UpdateCommand="UPDATE [Region] SET [RegionDescription] = @RegionDescription WHERE [RegionID] = @original_RegionID"

Ну і задати додатково описані вище властивості для передачі оригінальних параметрів (в даному випадку оновлення даних відбувається тільки за значенням ключового поля RegionID):

ConflictDetection="OverwriteChanges"

OldValuesParameterFormatString="original_{0}" >

Тепер в самому SqlDataSource залишилося описати лише колекції параметрів для команд оновлення та видалення

<DeleteParameters>

<asp:Parameter Name="original_RegionID" Type="Int32" />

</DeleteParameters>

<UpdateParameters>

<asp:Parameter Name="RegionDescription" Type="String" />

<asp:Parameter Name="original_RegionID" Type="Int32" />

</UpdateParameters>

</asp:SqlDataSource>

А так же описати GridView, в якому ми зможемо поспостерігати за результатами:

<asp:GridView ID="GridView1" runat="server" AutoGenerateColumns= "False" DataKeyNames="RegionID" DataSourceID="SqlDataSource1">

<Columns>

<asp:BoundField DataField="RegionID" HeaderText="RegionID" ReadOnly="True" />

<asp:BoundField DataField="RegionDescription" HeaderText="RegionDescription" />

<asp:CommandField ShowDeleteButton="True" ShowEditButton="True" />

</Columns>

</asp:GridView>

34.11.5. Методи і події SqlDataSource

Окрім можливості декларативної взаємодії з візуальним контролем SqlDataSource надає також програмну можливість виклику команд за допомогою відповідних методів – Select, Insert, Update і Delete. Метод Select викликається з параметром типу DataSourceSelectArguments і повертає DataSet або IDataReader залежно від значення властивості DataSourceMode, інші ж методи викликаються без параметрів і повертають кількість оброблених рядків. Наприклад в попередньому прикладі ми не використали властивість InsertCommand так як контрол GridView не вміє додавати дані. Виправимо тепер це упущення за допомогою додаткових контролів.

Для початку додамо в код форми 2 текстбокса для введення значень RegionID і RegionDescription і кнопку для додавання введених даних у базу.

* Add New Region: *

Region ID: <asp:TextBox ID="txtRegionID" runat="server"> </asp:TextBox>, Decription: <asp:TextBox ID="txtRegionDescription" runat="server"></asp:TextBox> <asp:Button ID="Button1" runat="server" Text="Add" />

Тепер додамо в попередній опис SqlDataSource код для вставки запису

InsertCommand="INSERT INTO [Region] ([RegionID], [RegionDescription]) VALUES (@RegionID, @RegionDescription)"

<InsertParameters>

<asp:ControlParameter ControlID="txtRegionID" Name="RegionID" PropertyName="Text" Type="Int32" />

<asp:ControlParameter ControlID="txtRegionDescription" Name="RegionDescription" PropertyName="Text" Type="String" />

</InsertParameters>

Як бачите в описі параметрів в даному випадку використовується ControlParameter для вказівки того, що значення параметрів потрібно отримувати з контролів. Тепер все, що нам потрібно зробити, це в обробнику події Click кнопки написати наступний рядок:

SqlDataSource1.Insert ();

Все, тепер при натисканні на кнопку Add в базу додасться нова запис, що містить введені в текстбокси значення. Аналогічним чином можна викликати і інші методи зміни даних у джерелі даних.

Контрольні запитання

1. Які методи використовуються для прив'язки даних до джерел даних?
2. Яка властивість дозволяє комбінувати дані з елемента-джерела з даними?
3. Що таке шаблон?
4. Для чого необхідний виклик методу DataBind?
5. Яка властивість задає формат виведення даних у DataGrid?
6. Яка головна перевага GridView?
7. Яка властивість використовується для того, щоб вказати провайдер доступу до даних?
8. Для чого потрібне кешування?
9. Які в ASP.NET елементи-джерела даних?
10. Яка властивість SQLDataSource визначає тип набору даних, що повертаються?

Тема 35. Використання бази даних у веб-додатках

Діяльність більшості web-додатків зосереджена на вилученні, відображенні та модифікації даних. Дані завдання здаються досить прямолінійними, але за останнє десятиліття способи використання даних постійно змінювалися. Розробники перейшли від простих клієнтських додатків з локальними базами даних до розподільних систем, заснованих на централізованих базах даних та виділених серверах. У той же час розвивалися технології доступу до даних. Сьогодні понад 95% всіх інформаційних систем так чи інакше використовують бази даних. Ось чому питання організації взаємодії додатку з базою даних є актуальними, а увага провідних розробників програмного забезпе-

чення прикута до цієї проблеми. Дана лекція присвячена вивченню питань, пов'язаних з організацією взаємодії web-додатків з базами даних.

Однією з основних ідей, що лежать в основі ADO.NET є наявність постачальників даних. Постачальник даних – це набір класів, призначених для взаємодії із сховищем даних певного типу. За рахунок цього, модель ADO.NET є надзвичайно гнучкою і розширюваною. Розглянемо рівні моделі постачальників ADO.NET (рис.35.1).

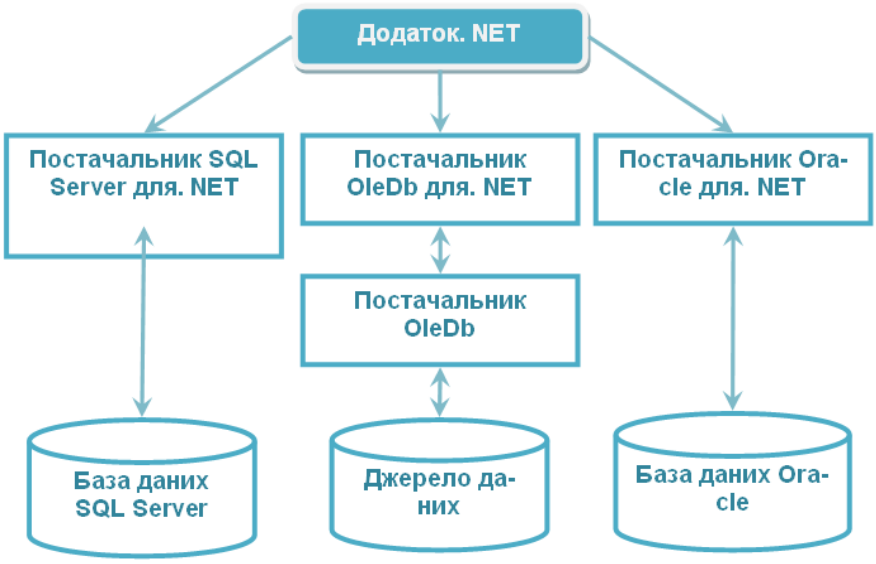

Рисунок 35.1 – Рівні моделей постачальників ADO.NET

Як видно з рис. 35.1, додаток .NET взаємодіє з базою даних за допомогою постачальників даних. Кожен постачальник даних може забезпечувати доступ тільки до бази даних певного формату. Так, для доступу до БД Microsoft SQL Server використовується постачальник SQL Server для .NET, для доступу до БД Oracle – постачальник Oracle для .NET і т.д. Названі бази даних є одними з найбільш поширених у всьому світі, тому постачальники даних для них виділені в окремі елементи моделі ADO.NET, тим не менш, існує безліч баз даних іншого формату, до яких необхідно здійснювати доступ з програми .NET. Для цього може застосовуватися постачальник даних OleDb для .NET або ODBC для .NET, які забезпечують доступ до будь-якими даними, для яких існує драйвер OleDb, або ODBC відповідно.

При побудові програми, що використовує доступ до даних, необхідно спершу спробувати знайти „рідного" постачальника даних для .NET. Якщо такого не існує, можна скористатися OleDb, якщо існує відповідний драйвер для джерела даних, до якого встановлюється підключення. Т.я. технологія OleDb існує досить давно, існує багато драйверів для різних джерел даних, що підтримують її. Якщо в системі не встановлений відповідний драйвер OleDb, його можна спробувати знайти на сайті розробника тієї бази даних, до якої здійснюється доступ, або на сайті Microsoft. Те ж саме справедливо і для технологи ODBC. У ряді випадків, існує кілька альтернативних варіантів організації доступу до даних певного формату. Наприклад, доступ до джерела даних на основі SQL Server можна організувати використовуючи або постачальник SQL Server

для .NET, або постачальник OleDb. Проте, завжди краще використовувати той постачальник даних, який спеціально призначений для забезпечення доступу до даного джерела даних, тому що він враховує його особливості.

Кожен постачальник. NET реалізує однакові базові класи – Connection, Transaction, DataAdapter, Command, Parameter, DataReader імена яких залежать від постачальника. Наприклад, у постачальника SQL Server існує об'єкт SqlDataAdapter, у постачальника OleDb – OleDbDataAdapter и т.д.

У кожного постачальника даних існує власний простір імен. Хоча всі постачальники відносяться до простору імен System.Data, кожен з них містить свій підрозділ цього простору, який містить об'єкти, специфічні для даного постачальника. Наприклад, об'єкт SqlDataAdapter знаходиться в просторі імен System.Data.SqlClient.

Усі постачальники даних .NET реалізують однакові базові функції, тому код створюваний для доступу до даних виглядає приблизно однаково незалежно від постачальника. Це означає, що розглядаючи використання інтерфейсів окремого постачальника, ми фактично розглядаємо можливість використання цих же інтерфейсів стосовно і до інших постачальників даних.

При організації доступу до даних за допомогою ADO.NET виключно важливу роль відіграють об'єкти. Розглянемо їх більш докладно.

Об'єкт Connection представляє з'єднання з джерелом даних. З його допомогою можна задати тип джерела даних, його місцезнаходження, параметри доступу і ряд інших атрибутів. Перед тим, як почати взаємодію з джерелом даних, необхідно встановити підключення до нього за допомогою об'єкта Connection.

Об'єкт Command представляє запит до джерела даних, виклик збереженої процедури або прямий запит на повернення вмісту конкретної таблиці. Як відомо, існує кілька типів запитів. Частина з них повертають дані, які добуваються з джерела даних, інші – змінюють записи, треті – управляють структурою БД. За допомогою об'єкта Command можливо виконати будь-який з перерахованих типів запитів. Відмінності в поведінці об'єкта Command починають проявлятися тоді, коли необхідно виконати той чи інший запит. Так, наприклад, при необхідності виконання запиту, не повертаються записи, необхідно використовувати метод ExecuteNonQuery об'єкта Command. При отриманні даних з БД – методExecuteReader, який в свою чергу повертає об'єкт DataReader, що дозволяє переглядати отримані в результаті запиту записи.

Об'єкт DataReader призначений для максимально швидкої вибірки та перегляду повернутих запитом записів. Однак, він дозволяє переглядати весь результуючий набір записів шляхом переміщення від одного запису до іншої, і, таким чином, переглядати тільки один запис за раз. Крім того, DataReader не має можливості для оновлення значень записів, внаслідок чого має можливість працювати в режимі тільки для читання, за рахунок чого має високу продуктивність.

Об'єкт Transaction дозволяє здійснювати угруповання записів в логічну одиницю роботи, званою транзакцією. Транзакція логічно об'єднує кілька різних дій, пов'язаних з маніпулюванням даними в єдине ціле. У процесі виконання дій, здійснюваних у рамках транзакції, СУБД зазвичай кешує зміни, що вносяться цими діями в дані до моменту завершення транзакції. Це до-

зволяє проводити скасування будь-яких змін, виконаних в рамках транзакції у випадку, якщо хоча б одне з дій транзакції завершилося невдало.

Об'єкт Parameter дозволяє вводити в запит елемент, значення якого може бути задане безпосередньо перед виконанням запиту. За рахунок цього відпадає необхідність щоразу змінювати текст самого запиту.

Об'єкт DataAdapter (табл. 35.1) є сполучною ланкою між від'єднаними об'єктами ADO.NET і базою даних. З його допомогою здійснюється заповнення таких об'єктів як DataSet чи DataTable значеннями, отриманими в результаті виконання запиту до бази даних для подальшої автономної роботи з ними. Крім цього, DataAdapter реалізує ефективний механізм виконання оновлення даних, що зберігаються в базі даних змінами, внесеними дані об'єктів DataSet і DataTable.

Таблиця 35.1. Основні методи DataAdapter

Fill	Виконання запиту типу Select, визначеного у властивості SelectCommand і додавання таблиці, одержуваної в результаті даного запиту в DataSet.
FillSchema()	Виконання запиту типу Select, текст якого розташований у властивості SelectCommand і додавання таблиці, яка містить тільки структуру даних отриманих в результаті виконання запиту в DataSet.
Update()	Застосовує всі зміни, внесені до DataTable до джерела даних. При цьому виконуються команди вставки, оновлення та видалення, розташовані у властивостях InsertCommand, UpdateCommand, DeleteCommand.

Метод Fill об'єкту DataAdapter, в якості параметрів використовує ім'я об'єкта DataSet, в який необхідно помістити дані, що повертаються в результаті виконання запиту, визначеного в рядку sqlString. У другому параметрі можна вказати ім'я, яке буде зіставлено з таблицею, створеної в DataSet. З прикладу видно, що в даному випадку не використовується явний виклик методу Open об'єкту Connection. Це пояснюється тим, що даний метод викликається неявно при виконанні методу Fill. Таким чином, DataAdapter спершу відкриває з'єднання з БД, потім виконує необхідні маніпуляції з даними, після чого закриває відкрите раніше з'єднання. Якщо з якихось причин такий алгоритм взаємодії з базою даних потрібно змінити, необхідно до виклику методу Fill відкрити з'єднання з БД. У цьому випадку DataAdapter буде використовувати вже існуюче з'єднання.

Після того, як необхідний набір даних був витягнутий з бази даних, його необхідно відобразити на Web-сторінці. Для цього можна скористатися способом, описаним раніше, при розгляді об'єкта DataReader. Однак, в даному випадку, набагато більш ефективного використання можна домогтися за допомогою використання можливостей прив'язки даних до візуальних об'єктів, здатним виводити дані на екран. Основна ідея, що лежить в основі прив'язки даних, полягає у створенні зв'язку між об'єктом даних і елементом управління. Всю подальшу роботу, пов'язану з витяганням даних і виведенням їх на екран виконує ASP.NET.

Об'єкт DataTable дозволяє переглядати дані у вигляді наборів записів і стовпців. Фактично, він являє собою аналог таблиці БД, розміщений в

пам’яті. Перевагою такої організації є можливість автономної роботи з даними. Це означає, що після встановлення з’єднання з базою даних, читання даних і заповнення ними об’єкта DataTable можна відключитися від джерела даних і продовжувати працювати з ним в автономному режимі. Така можливість надзвичайно корисна при організації Web-додатків, які повинні бути добре масштабованими і орієнтовані на багатокористувацький режим роботи. При цьому, однак, виникають і побічні ефекти, один з яких пов’язаний з тим, що людина, що працює з автономним набором даних не може побачити змін, що вносяться до дані в цей момент іншими користувачами.

Об’єкт DataColumn являє собою стовпчик об’єкта DataTable. Набір же усіх стовпчиків об’єкта DataTable являє собою колекцію Columns. Постредством цього об’єкта можна отримати доступ до значення комірки відповідного стовпчика.

Об’єкт DataRow являє собою рядок об’єкта DataTable. Набір всіх рядків цього об’єкта являє собою колекцію Rows. DataRow дуже часто використовується для доступу до значення конкретного поля певного запису. При цьому застосовується властивість Item.

Об’єкт DataSet являє собою від’єднаний набір даних, який може розглядатися як контейнер для об’єктів DataTable. DataSet дозволяє організовувати всередині себе структуру, яка повністю відповідає реальній структурі таблиць і зв’язків між ними в БД. Це зручно в тому випадку, коли при роботі з базою даних необхідні дані з різних таблиць. У цьому випадку, замість того, щоб багато разів звертатися до сервера і вибирати дані з однієї таблиці за раз, можна помістити всі дані в об’єкт DataSet, а потім передати його клієнтського додатку. DataSet є дуже потужним інструментом для роботи з від’єднаними наборами даних. Всі зміни, що вносяться до дані, що зберігаються в DataSet ешируются в об’єктах DataRow. Коли виникає необхідність передачі змін з DataSet в БД можливо здійснити передачу тільки змінених даних, замість того, щоб передавати всі дані об’єкта, що значно знижує обсяг даних, переданих між клієнтським комп’ютером і сервером. Дані в DataSet від’єднані від БД. Всі зміни даних кешируются в об’єктах DataRow. При виникненні необхідності передачі змін до даних об’єкта DataSet існує можливість передачі тільки що змінилася частини даних, що дозволяє значно економити ресурси каналу зв’язку, т.к. в цьому випадку передається набагато менший обсяг даних. Узагальнюючи все вищесказане можна зробити висновок про те, що використання об’єкта DataSet в ряді випадків виявляється більш ефективним, ніж DataReader. Найбільш типовими ситуаціями, в яких рекомендується застосовувати об’єкт DataSet є:

1. Необхідність реалізації серіализація даних на диск. DataSet дозволяє легко зберігати дані у файлі XML. При цьому можливі варіанти збереження тільки даних, тільки структури даних, або і того й іншого.

2. Необхідність організації навігації по набору даних у двох напрямках. Як уже згадувалося, DataReader забезпечує переміщення по набору тільки вперед. За допомогою DataSet можлива організація посторінкового перегляду даних.

3. Необхідна прив’язка декількох елементів управління до одному набору даних. DataSet, на відміну від DataReader містить засоби організації сортування та фільтрації даних.

Для вилучення даних з бази даних і наповнення ними об’єкта DataSet необхідно використовувати ще один об’єкт DataAdapter, який крім іншого дозволяє оновлювати дані БД на основі внесених до DataSet змін.

Важливо! Більшість Web-додатків використовує DataSet для отримання даних з бази даних і показу їх на сторінці. Для оновлення даних в БД використовуються прямі команди.

Так як об’єкт DataSet може містити багато таблиць, у властивості DataSource необхідно вказати ім’я таблиці, до якої необхідно здійснити прив’язку. Властивість DataTextField містить найменування стовпця таблиці, значення якого будуть відображатися елементом управління. Властивість DataValueField є необов’язковою і позначає те, що при виборі значення в елементі управління, значення властивості SelectedItem цього елемента буде встановлено рівним значенню поля, зазначеного в DataValueField, відповідного вибраному запису.

Об’єкт DataRelation являє собою опис зв’язків між таблицями реляційної бази даних. Він надається об’єктом DataSet і дозволяє організовувати взаємозв’язки між таблицями відокремленого набору даних об’єкта DataSet. Об’єкт DataRelation виконує функцію аналогічну тій, яку виконують зв’язку, які визначаються в СУБД між таблицями при створенні структур даних. Це стосується і дотримання принципів посилальної цілісності інформації. Наприклад, DataRelation можна налаштувати таким чином, щоб зміна значення первинного ключа батьківської таблиці автоматично передавалися (каскадувати) дочірнім записам, а при видаленні запису в батьківській таблиці, автоматично віддалялися записи в дочірніх таблицях, пов’язаних з нею.

Об’єкт DataView призначений для організації можливості перегляду вмісту DataTable. Це відноситься до таких операцій, як сортування і фільтрація записів. За допомогою DataView допускається переглядати вміст одного об’єкта DataTable з різними установками фільтрації і сортування. Для цього необхідно використовувати два різних об’єкта DataView, пов’язаних з одним об’єктом DataTable. ака можливість виключає необхідність зберігання одного набору даних у двох різних структурах.

Більш докладно використання всіх об’єктів моделі ADO.NET буде розглянуто нижче.

35.1. Організація взаємодії з БД

Для того, щоб додаток .NET міг здійснювати взаємодію з джерелом даних, необхідно встановити з’єднання з ним. Найбільш типовим сценарієм роботи Web-програми є наступним:

1. Встановлюється з’єднання, відкривається підключення до бази даних.
2. Виконується один або декілька запитів, які здійснюють внесення змін до наборів даних джерела даних, а також вибірку даних з БД.

3. Здійснюється відключення від джерела даних. При цьому користувач працює з видаленим набором даних, переглядаючи його, виконуючи фільтрацію, вносячи зміни і т.д.

4. При необхідності перенесення змін, з від'єднаного набору даних в БД, а також при необхідності перегляду змін, внесених до БД іншими користувачами, здійснюється підключення до джерела даних, виконуються необхідні дії, після чого проводиться відключення від БД.

Розглянемо описані вище етапи більш детально.

35.2. Підключення до БД

Як вже говорилося вище, для підключення до джерела даних з використанням ADO.NET необхідно скористатися об'єктом Connection. Для розгляду прикладів підключень до джерел даних, перш за все, необхідно вибрати постачальника даних, з яким працюватиме додаток. Потім необхідно підключити відповідні простори імен, що містять визначення об'єктів обраного постачальника. Як приклади підключення до джерел даних будемо розглядати бази даних Access і SQL Server Express Edition. Такий вибір продиктований насамперед тим, що Access дуже добре підходить для побудови невеликих інформаційних систем, невимогливий до ресурсів комп'ютера, не вимагає установки спеціального програмного забезпечення. SQL Server Express є вільно поширюваною СУБД, що володіє всіма достоїнствами комерційного SQL Server і володіє низкою обмежень, в тому числі на максимальний обсяг бази даних (не більше 4 Гб). Всі прийоми роботи з базами даних, що описуються нижче можуть бути використані також для роботи з комерційними версіями продуктів Microsoft, та іншими СУБД.

Отже, для того, щоб підключитися до бази даних під час виконання програми, необхідно створити об'єкт Connection, а також задати його властивості, що визначають поточні параметри підключення. Основним параметром, що встановлює необхідні опції для підключення до БД є рядок з'єднання, яка являє собою набір пар „ім'я – значення", розділених крапкою з комою. Порядок проходження значень цих параметрів, а також їх регістр не важливі. Рядок з'єднання залежить від СУБД, до якої здійснюється підключення, а також від використовуваного постачальника даних. Тим не менш, існує кілька фрагментів інформації, що вказується в рядку підключення, необхідних практично завжди. Перерахуємо їх і прокоментуємо їх призначення:

1. Сервер, на якому знаходиться база даних. Якщо СУБД, до якої здійснюється підключення розташована на клієнтському комп'ютері (так буде у всіх прикладах, що розглядаються в рамках даного курсу), замість імені сервера необхідно вказувати ім'я localhost, або IP адреса 127.0.0.1.

2. Ім'я бази даних, до якої здійснюється підключення.

3. Спосіб аутентифікації користувача. Існуючі клієнт-серверні СУБД (до яких відноситься SQL Server, Oracle і ряд інших) дозволяють вказувати в рядку підключення ім'я користувача та пароль, які будуть використовуватися для перевірки можливості доступу до бази даних, або використовувати параметри поточного користувача.

Для підключення до БД використовується об’єкт Connection з простору імен System.Data.SqlClient у випадку з SQL Server і System.Data.OleDb у випадку з іншими джерелами даних, наприклад Access. Рядки з’єднання з базами даних при цьому будуть виглядати наступним чином.

string strSqlConnection = "Data Source=localhost\\sqlexpress; InitialCatalog=TEST_DB;Integrated Security=SSPI";

string strOleDbConnection = "Provider=Microsoft.Jet.OLEDB.4.0; DataSource=C:\\Projects\\Ex_Db\\App_Data\\TEST_DB.mdb";

Для відкриття з’єднання з базою даних необхідно викликати метод Open об’єкта Connection. При цьому, рядок з’єднання з БД можна передати як як параметр конструктора об’єкта, так і потім за допомогою установки відповідної властивості.

```
SqlConnection sqlCon=new SqlConnection(strSqlConnection);
sqlCon.Open();
OleDbConnection oleDbCon=new OleDbConnection();
oleDbCon.ConnectionString=strOleDbConnection;
oleDbCon.Open();
```

Рядок з’єднання з базою даних можна жорстко прописати у вихідному коді програми, однак, це не найкращий варіант, тому що при зміні шляху до бази даних, або інших параметрів з’єднання доведеться вносити зміни у вихідний код програми та перекомпілювати його. У зв’язку з цим, найкраще використовувати рядок з’єднання, збережену в файлі web.config.

```
<connectionStrings>
<add name="TEST_DBConnectionString" connectionString="Provider= Microsoft.Jet.OLEDB.4.0; DataSource=C:\Projects\Ex_Db\App_Data\ TEST_DB.mdb"/>
</connectionStrings>
```

У наслідку цей рядок можна витягти з файлу web.config.

```
string strOleDbConnection = WebConfigurationManager.ConnectionStrings ["TEST_DBConnectionString"].ConnectionString;
```

Управління з’єднанням здійснюється дуже легко. Методи Open і Close об’єкта Connection виконують всю роботу. Однак, слід враховувати, що при підключенні до бази даних може відбутися збій, в результаті якого встановити з’єднання з нею виявиться неможливо. Це може бути особливо актуальним при розміщенні бази даних на іншому сервері, який у момент підключення може виявитися недоступний. Для того, щоб невдала спроба з’єднання з базою даних не приводила до фатальних наслідків при роботі додатка, необхідно використовувати конструкції try catch, що дозволяють адекватно реагувати на виниклу помилку. Наступний приклад демонструє можливість використання такого підходу.

```
try
{sqlCon.Open();
lbl_DB.Text = "<b>Сервер:</b>"+sqlCon.ServerVersion;
lbl_DB.Text += "</br><b>Соединение:</b>" + sqlCon.ToString();}
catch(Exception ex)
{lbl_DB.Text = "При соединении с БД произошла ошибка ";
```

```
lbl_DB.Text += ex.Message;}
finally
{sqlCon.Close();
lbl_DB.Text += "</br><b>Соединение:</b>";
lbl_DB.Text += sqlCon.State.ToString();
}
```

Результатом роботи цього фрагмента коду в разі успішного встановлення з'єднання буде повідомлення, зображене на рис. 35.2.

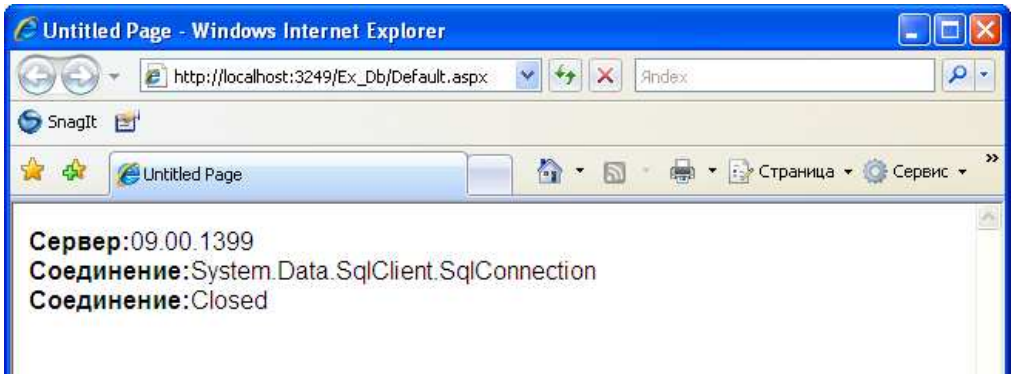

Рисунок 35.2 – Вікно повідомлення про параметри і стан поточного з'єднання з БД

На рис. 35.3. зображено вікно, що містить повідомлення про помилку встановленого з'єднання.

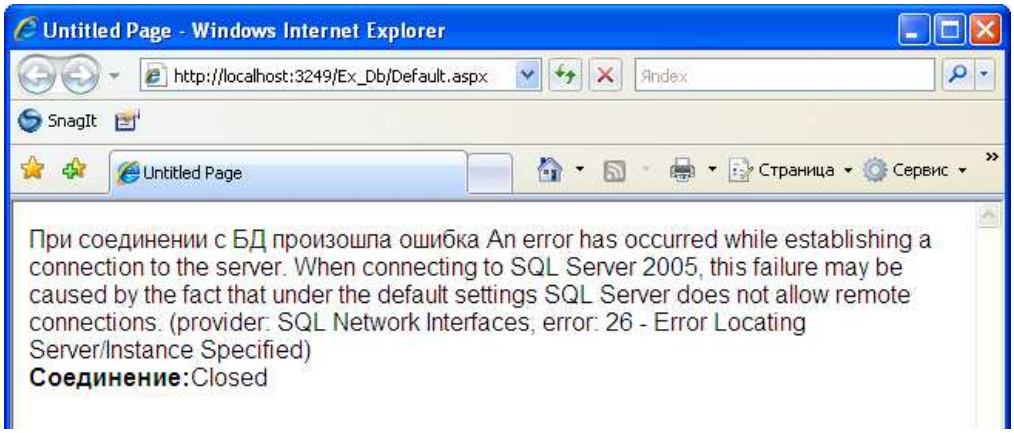

Рисунок 35.3 – Вікно повідомлення про помилку в момент підключення до БД

Для обслуговування з'єднання з базою даних витрачаються ресурси комп'ютера, на якому виконується Web-додаток. У зв'язку з цим, рекомендується відкривати з'єднання якомога пізніше, а закривати якомога раніше, відразу після того, як всі необхідні дії з об'єктами бази даних були виконані. Крім того, бажано будувати програмний код так, щоб з'єднання з базою даних закривалося при будь-якому результаті з'єднання з нею. У попередньому прикладі, код, розташований в блоці finally виконається при будь-якому результаті спроби підключення до неї, що гарантує звільнення ресурсів сервера не чекаючи моменту, коли пам'ять буде звільнена збирачем сміття.

Контрольні запитання

1. Що таке постачальник даних?
2. За допомогою якого об’єкту можна задати тип джерела даних?
3. Для чого використовується об’єкт DataAdapter?
4. Для чого призначений об’єкт DataView?
5. Що являє собою об’єкт DataSet?
6. Який об’єкт дозволяє переглядати дані у вигляді наборів записів і стовпців?
7. Які найбільш типові ситуації, в яких рекомендується застосовувати об’єкт DataSet?
8. Який об’єкт дозволяє здійснювати транзакцію?
9. Що являє собою об’єкт DataRelation?
10. Для чого потрібен об’єкт Parameter?

Тема 36. Трирівнева архітектура доступу до даних ASP.NET

Логіка взаємодії складних інформаційних систем з базами даних зазвичай буває досить складною. Складність ця багаторазово збільшується разом із збільшенням складності інформаційної системи. Якщо не забезпечити уніфіковані механізми і інтерфейси доступу до даних, підтримку і розвиток інформаційної системи стає дуже трудомістким, а часом і просто неможливим. Виходом з цієї ситуації є створення проміжного шару, що забезпечує взаємодію додатки з базою даних на основі принципів трирівневої архітектури баз даних.

У складних професійних додатках код доступу до бази даних не вбудовується в сам додаток, а інкапсулюється в окремий виділений клас. Для виконання операції з базою даних додатку необхідно створити екземпляр цього класу і викликати відповідний метод. При створенні такого класу необхідно дотримуватися ряду рекомендацій:

1. Необхідно відкрити з’єднання з базою даних перед виконанням операції з нею і закривати відразу після завершення цієї операції.

2. Необхідно реалізовувати обробку помилок для гарантованого закриття підключення навіть якщо в процесі виконання операції виникла виняткова ситуація.

3. Так як реалізація логіки взаємодії з базою даних описана в методах класу, необхідно передати вихідні дані методам за допомогою параметрів, а результати повернути за засобом повернутого значення.

4. Не можна дозволяти клієнту вказувати і змінювати рядок з’єднання з базою даних, тому що це може стати причиною порушення безпеки програми.

5. Не можна підключатися до джерела даних з використанням клієнтського ідентифікатора користувача. Для більш ефективної організації взаємодії додатку з БД краще використовувати систему безпеки на основі ролей, чи білетів.

6. При виборі даних з БД необхідно вибирати лише ті рядки, які дійсно необхідні додатком для роботи. Якщо не дотримуватися даного принципу, швидкодія додатку може помітно падати у міру зростання бази даних, тому що обсяги переданої по мережі інформації можуть значно збільшуватися.

Сучасні програми створюються на основі об’єктно-орієнтованого підходу. Основним будівельним блоком додатку в цьому підході є клас. Всі оброблювані додатком дані зберігаються в об’єктах, які також містять і методи для їх обробки. Проте, більшість використовуваних СУБД сьогодні побудовані на принципах реляційної моделі даних. Принципи обробки даних в об’єктних і реляційних моделях відрізняються, це необхідно враховувати при організації взаємодії з базою даних. Нижче розглядаються деякі питання організації такої взаємодії.

У добре організованому додатку створюється окремий клас для кожної таблиці, або логічно взаємопов’язаної групи таблиць бази даних. Цей клас використовується для організації взаємодії додатку з базою даних, реалізуючи всі необхідні операції маніпуляції цими даними. Сукупність таких класів являє собою як би проміжний програмний шар, що знаходиться між класами, що містить дані і методи для їх обробки і даними, збереженими в базі даних. Набір таких класів, що реалізують логіку взаємодії додатку з базою даних, може бути реалізований у вигляді окремої збірки, яка може бути відкомпільована. Це доцільно в тому випадку, коли аналогічну логіку необхідно реалізувати в декількох додатках.

Насамперед, створимо клас, що містить поля, що відповідають полям таблиці. Доступ до цих полів повинен здійснюватися за допомогою відповідних властивостей. У типовому об’єктно-орієнтованому додатку такі класи виконують роль сховища даних, доступ до яких надають методам, які здійснюють їх обробку. У досить великому і складному додатку класи бажано розміщувати окремо від основного коду програми. Це дозволяє легко відокремлювати код основної програми від коду, який може бути перенесений в інші додатки і повторно використовуватися для полегшення у прискорення створення додатку. Для створення класу рекомендується виконати команду Website → Add New Item. У діалоговому вікні вибрати в якості типу створюваного елемента Class, ввести ім’я класу і натиснути кнопку Add. З’явлене діалогове вікно, повідомляє, що класи використовувані додатком, повинні бути поміщені в папку App_Code. Рекомендується помістити створюваний клас в цю папку.

Так, для таблиці tovar БД (рис. 36.1) необхідно створити клас, наприклад Ptoduct.

Рисунок 36.1 – Таблиця tovar бази даних

Текст класу Product наведений нижче:

```
public class Product
{   private int kod_tov;
   private string nazv;
   private string gost;
   public int Kod_tov
   {  get { return kod_tov; }
      set { kod_tov = value; }
   }
   public string Nazv
   {   get { return nazv; }
      set {nazv = value; }
   }
   public string Gost
   {   get { return gost; }
      set { gost = value; }
   }
   public Product(int kod_tov, string nazv, string gost)
   {   this.kod_tov = kod_tov;
      this.nazv = nazv;
      this.gost = gost;
   }
}
```

Вся обробка даних про товари додатком має проводитися з використанням класу Product. Для цього необхідно спочатку завантажити в цей клас необхідну інформацію з бази даних, зробити необхідні дії, а потім оновити інформацію про товари в базі даних. Звичайно, питання організації універсальних способів взаємодії об'єктів докладання з базою даних досить складні і виходять за рамки даного курсу, тому обмежимося лише найбільш загальними принципами створення такого механізму.

Створимо клас, призначений для організації взаємодії додатку з таблицею Товари бази даних. Цей клас повинен містити необхідні методи для виконання всіх тих операцій, які необхідно виконувати клієнтському додатку з інформацією розташованої в базі даних. У даному прикладі, створимо клас, який „уміє" витягувати інформацію про товари з таблиці Товари, а також додавати, видаляти і оновлювати цю інформацію. Додавання та видалення даних є найпростішими процедурами, тому почнемо з їх реалізації.

Створимо клас ProductsDB як було описано вище. Визначимо в класі закриту змінну connectionString, що містить рядок підключення, який у момент створення класу (в конструкторі) зчитується з файлу web.config і поміщається в connectionString. У web.config параметр, що містить рядок підключення може називатися по-різному, тому можна передбачити два конструктора: один – налаштований на витяг рядка підключення з жорстко заданого параметра, другий – приймає ім'я даного параметра і використовує його для читання рядка підключення.

```
public class ProductsDB
{   private string connectionString;
   public ProductsDB()
   {connectionString = WebConfigurationManager.ConnectionStrings["con-
nectionString"].ConnectionString;
   }
   public ProductsDB(string conString)
   {
connectionString=WebConfigurationManager.ConnectionStrings[conString
].ConnectionString;
   }
}
```

При цьому у файлі web.config прописується рядок підключення до необхідної БД.

```
<connectionStrings>
<add name="connectionString"
connectionString="Data Source=KATYAN\SQLEXPRESS;
InitialCatalog=tovar; Integrated Security=True"
providerName="System.Data.SqlClient" />
</connectionStrings>
```

Визначимо метод, призначений для додавання нового запису в таблицю Товари. Для цього створимо об’єкт з’єднання з базою даних. У даному прикладі використовується СУБД SQL Server Express 2005, але аналогічний код (з невеликими змінами) можна використовувати для підключення практично до будь якої СУБД. Створимо запит з параметрами, заповнимо ці параметри значеннями, витягнутими з полів об’єкта Product, переданого в даний метод як параметр і виконаємо запит. Текст методу AddProduct, що виконує вище описані дії, представлений нижче.

```
public int AddProduct(Product prod)
{
SqlConnection con = new SqlConnection(connectionString);
string query = "INSERT INTO Товари (КодТовара,Найменування това-
ру,Ціна) VALUES (@id,@name,@cost)";
SqlCommand cmd = new SqlCommand(query, con);
cmd.CommandType = CommandType.Text;
cmd.Parameters.Add(new SqlParameter("@id", SqlDbType.Int, 4));
cmd.Parameters.Add(new SqlParameter("@name", SqlDbType.VarChar, 50));
cmd.Parameters.Add(new SqlParameter("@cost", SqlDbType.Float, 8));
cmd.Parameters["@id"].Value = prod.ProductID;
cmd.Parameters["@name"].Value = prod.ProductName;
cmd.Parameters["@cost"].Value = prod.ProductCost;
try
      {        con.Open();
         return cmd.ExecuteNonQuery();
      }
```

```
        catch (SqlException e)
        {
    throw new ApplicationException("Помилка додавання нового товару в
таблицю Товари");
        }
        finally
        {       con.Close();       }
    }
```

Тут варто звернути увагу на те, що відкриття з'єднання з базою даних, а також виконання запиту виконуються в блоці try, що дозволяє обробляти виняткові ситуації, що виникають, і гарантує, що яким би не був результат виконання операції, буде закрито. У разі виникнення помилки формується виняткова ситуація, що виводить повідомлення про помилку. У формулюванні цього повідомлення слід уникати зайвих подробиць про причини виникнення помилки, тому що деякі користувачі можуть використовувати дану інформацію для пошуку проломів в захисті інформаційної системи та використання їх для порушення безпеки системи.

Аналогічним чином створюються інші методи роботи з базою даних. Створимо метод для видалення товару з таблиці Товари. Код методу DeleteProduct, що здійснює цю операцію наведено нижче.

```
public void DeleteProduct(int productID)
  {
    SqlConnection con = new SqlConnection(connectionString);
    string query = "DELETE FROM Товари WHERE КодТовара=@ID";
    SqlCommand cmd = new SqlCommand(query, con);
    cmd.CommandType = CommandType.Text;
    cmd.Parameters.Add("@ID", SqlDbType.Int, 4);
    cmd.Parameters["@ID"].Value = productID;
    try
    {       con.Open();
      cmd.ExecuteNonQuery();       }
    catch (SqlException e)
    {
      throw new ApplicationException("Помилка видалення товару з
таблиці Товари");
    }
    finally
    {       con.Close();       }
  }
```

Відмінність даного методу від попереднього полягає в тому, що виконується запит на видалення запису, а в якості параметра передається номер (код) товару, який треба видалити.

При створенні методів , що витягають дані з БД необхідно враховувати для чого будуть застосовуватися дані методи. Справа в тому, що може існувати безліч різних потреб у витягуваних із БД даних. Відмінності в них зво-

дяться до уявлення витягнутих даних у різних форматах. Наприклад, якщо необхідно організовувати вивід інформації з використанням таких елементів як GridView, найзручніше зробити так, щоб метод повернув таку структуру, яку можна використовувати для прив'язки даних до цього елемента. До таких структур, як вже було сказано вище, відносяться DataReader, DataTable, DataSet і т.д. Якщо необхідно витягнути дані про один товар, необхідно, щоб метод повертав об'єкт Product, якщо це необхідно, щоб метод витягував список існуючих товарів, представлених в БД і необхідних для їх обробки, зручніше, щоб метод повернув масив об'єктів Product. У реальних додатках може знадобитися кілька методів, призначених для реалізації різних режимів роботи з БД. Наприклад, для відображення викликається один метод, який видобуває дані з БД, а для їх редагування – інший.

Створимо метод GetProductsTable(), призначений для вилучення даних про товари з таблиці Товари та приміщення їх в об'єкт DataTable. Вихідний код цього методу наведено нижче.

```
public DataTable GetProductsTable()
   {
      SqlConnection con = new SqlConnection(connectionString);
string query = "SELECT КодТовара,Найменування Товара,Ціна FROM Товари";
      SqlCommand cmd = new SqlCommand(query, con);
      cmd.CommandType = CommandType.Text;
      SqlDataAdapter da = new SqlDataAdapter(query, con);
      DataTable dt = new DataTable("Product");
      try
      {        con.Open();
         da.Fill(dt);
         return dt;      }
      catch (SqlException e)
      {
         throw new ApplicationException("Помилка читання списку товарів з таблиці Товари");
      }
      finally
      {        con.Close();        }
   }
```

Надалі отриманий в результаті такої операції результат, може бути використаний для відображення даних в об'єкті GridView.

Описані вище методи дозволяють отримувати дані про товари з БД у форматі, у найкраще пристосованому для їх відображення на екрані, але погано відповідного для редагування даних.

Для демонстрації використання створеного шару доступу до даних помістимо на форму об'єкт ProductsView класу GridView, а також створимо наступний код, що використовує можливості класу ProductsDB.

```
   protected void Page_Load(object sender, EventArgs e)
   {
```

```
        //Створення нового об`єкта доступу до даних
        ProductsDB prodDB = new ProductsDB("connectionString");
        if (!Page.IsPostBack)
        {          // Створення нового об`єкта Product
           Product prod = new Product(31, "Товар31","ГОСТ Товара31" );
           //Додавання створеного об`єкта Product у таблицю БД
           prodDB.AddProduct(prod);
        }
        //Встановити джерело даних та здійснити їх прив`язку
        //для елемента GridView
        ProductsView.DataSource = prodDB.GetProductsTable();
        Page.DataBind();
    }
```

Для демонстрації можливості видалення даних з таблиці Товари помістимо на форму кнопку і створимо наступний обробник події натисканням на неї.

```
protected void Button1_Click(object sender, EventArgs e)
    {       ProductsDB prodDB = new ProductsDB("connectionString ");
        prodDB.DeleteProduct(31);
        ProductsView.DataSource = prodDB.GetProductsTable();
        Page.DataBind();
    }
```

У цьому методі знову здійснюється установка джерела даних для об’єкта GridView, а також їх прив’язка. Це необхідно для того, щоб відобразити змінення в даних, вироблених раніше виконаними діями.

Контрольні питання

1. Яких рекомендацій треба дотримуватись при створенні екземпляру класу?
2. Що таке трирівнева архітектура веб-додатку?
3. Яким чином здійснюється перехід від реляційної моделі бази даних до об’єктно-орієнтованого представлення у веб-додатку?
4. Назвіть правила побудови трирівневої архітектури веб-додатку
5. Для збереження даних із таблиць БД у веб-додатку необхідно створити...
6. Яким чином можна задати рядок підключення до БД у веб-додатку?

Тема 37. Використання кешування даних в ASP.NET

Web-програми доцільно створювати в тих випадках, коли необхідні хороші показники його масштабованості та продуктивності. Кількість користувачів таких додатків може досягати декількох десятків і сотень тисяч. Кожне звернення користувача може призводити до необхідності читання і запису яких-небудь даних на жорсткі диски сервера. У тих випадках, коли деяка частина інформації є більш затребуваною, ніж її інші частини, доцільно використовувати кешування даних. Як відомо, кешування здатне значно прискорити роботу як апаратних, так і програмних частин комп’ютера. Практично всі сучасні пристрої зберігання і передачі даних використовують кеш пам’ять для прискорення роботи за рахунок мінімізації затримок, пов’язаних з очіку-

ванням закінчення запису, читання або передачі даних. Наприклад, можна кешувати результати складного запиту, так що для повторного його виконання взагалі не потрібно звертатися до бази даних. Замість цього відповідний об'єкт буде витягнутий безпосередньо з пам'яті сервера, що набагато швидше. Витонченість кешування в тому, що на відміну від багатьох інших прийомів підвищення продуктивності, воно збільшує як продуктивність, так і масштабованість. Продуктивність зростає завдяки значному зниженню часу, необхідного на вилучення інформації. Масштабованість ж стає краще завдяки тому, що обходяться вузькі місця – начебто з'єднанні з базою даних. В результаті додаток може обслужити більше запитів сторінок одночасно при меншій кількості операцій з базою даних.

Наприклад, можна кешувати результати складного запиту, так щоб для повторного його виконання взагалі не потрібно звертатися до бази даних. Замість цього відповідний об'єкт буде витягнутий безпосередньо з пам'яті сервера, що набагато швидше. Витонченість кешування в тому, що на відміну від багатьох інших прийомів підвищення продуктивності, воно збільшує як продуктивність, так і масштабованість. Продуктивність зростає завдяки значному зниженню часу, необхідного на вилучення інформації Масштабованість ж стає краще завдяки тому, що обходяться вузькі місця – начебто з'єднанні з базою даних. В результаті, додаток може обслужити більше запитів сторінок одночасно при меншій кількості операцій з базою даних.

У ASP.NET кешування – це спосіб прискорення роботи за рахунок організації зберігання в оперативній пам'яті сервера копій інформації, створення якої пов'язане з великими накладними витратами. Так, якщо користувачі постійно запитують інформацію для отримання якої необхідно виконати досить громіздку, складну і тривалу обчислювальну процедуру, доцільно буде виконати її один раз, записати результат її виконання в кеш, і при зверненні користувачів відразу видавати результат. Зрозуміло, що такий підхід значно прискорить процес отримання користувачами результату.

Використання кешування призводить не тільки до підвищення продуктивності і масштабованості, а й створює ряд проблем, які необхідно знати і вміти їх обходити. Одна з таких проблем це те, що кешування задіє оперативну пам'ять, яка ніколи не буває зайвою. Якщо спробувати зберегти в оперативній пам'яті занадто багато даних, операційна система скидає зайві дані на диск, що може призвести до уповільнення роботи всієї системи в цілому. Для управління цим процесом в ASP.NET реалізований інтелектуальний механізм визначення переповнення кешу, який заснований на тому, що при спробі запису в кеш даних, обсяг яких перевищує доступний для використання кеша обсяг оперативної пам'яті, ASP.NET вибірково видалить з кешу частину даних для забезпечення максимальної загальної продуктивності системи. Крім того, при використанні механізмів кешування, необхідно пам'ятати, що повинен бути якийсь принцип, згідно з яким буде здійснюватися перевірка оновлення інформації, на основі якої будується кеш. Якщо дані таблиці бази даних або файлу змінилися, це може означати, що дані кеша застаріли і їх необхідно оновити.

ASP.NET надає вдосконалений механізм управління кешуванням, реалізуючи всі необхідні елементи управління політикою кешування. Зокрема, він дозволяє управляти заміщенням інформації, що знаходиться в кеші, управляти профілями кеш, за допомогою яких можна визначити налаштування кешування для групи сторінок, керувати зберіганням вмісту кеша в оперативній пам'яті і на жорстких дисках, відстеження змін у вихідних даних і видаляти, або оголошувати недійсними елементи, що кешуються.

Багато розробників вивчаючи кешування, сприймають його як щось зайве, проте це величезна помилка. Розумне застосування кешування забезпечує багаторазове підвищення продуктивності – за рахунок утримання в пам'яті важливої інформації навіть на короткий період часу.

ASP.NET підтримує два типи кешування: кешування даних і кешування введення. Рекомендується використовувати обидва ці типи кешування, т.к. це здатне значно підвищити продуктивність програми.

Кешування даних – управляється безпосередньо з коду ASP.NET додатки, в якому розробник сам визначає яку інформацію необхідно помістити в кеш. Сторінки, до яких звертаються користувачі, можуть перевірити існування зацікавленої їх інформації в кеші перш ніж виконати кроки, необхідні для його отримання. У цьому сенсі кешування можна порівняти зі станом додатки з тією лише різницею, що при неможливості збереження даних в кеші вони видаляються, крім того можна налаштувати автоматичне видалення даних з кешу після закінчення певного часу.

Кешування виводу – найпростіший вид кешування, який дозволяє зберегти копію з генерованої HTML-сторінки, відправленої клієнту. Таким чином, якщо один, або декілька клієнтів запитають у сервера ту ж саму сторінку, вони отримають її копію з кешу.

Згадані види кешування стали основою для створення ще двох різновидів:

• кешування фрагментів – дозволяє зберігати в кеші лише частину готового HTML коду сторінки. В основному це відноситься до користувальницьких елементів управління. Наступного разу, коли виконується дана сторінка, генеруються ті ж події, але код відповідних елементів управління вже не виконується;

• кешування джерел даних – вбудований в об'єкти доступу до даних механізм кешування даних в тому числі SqlDataSource, XmlDataSource, який використовується автоматично при використанні відповідного елемента.

Розглянемо основні аспекти застосування кешування в ASP.NET.

37.1. Кешування виведення

При використанні даного виду кешування, згенерований у результаті виконання програми HTML-код зберігається в пам'яті, і при повторному запиті цієї ж сторінки, клієнту передається вже згенерований раніше HTML-код. Коли та ж сама сторінка запитується знову, об'єкти елементів управління не створюються заново, життєвий цикл сторінки не запускається і нічого з коду не виконується. Замість цього користувачеві доставляється кешування HTML-розмітки.

Існує два способи додавання сторінки до кеш виводу. Найбільш поширений полягає у вставці директиви OutputCache в початок файлу. Aspx, безпосередньо під директивою Page.

Атрибут Duration задає кількість секунд, протягом яких необхідно зберігати сторінку в кеші. Параметр VaryByParam що дорівнює значенню None, встановлює режим кешування, при якому, не залежно від додаткових параметрів, в кеші зберігатиметься тільки одна копія даної сторінки. Якщо додати фрагмент коду:

<%@ OutputCache Duration="10" VaryByParam="None" %>

Тепер при спробі оновлення сторінки менш ніж через 10 секунд, ніяких змін поточних значень відбуватися не буде. При спробі оновлення сторінки після закінчення 10 секунд, сторінка оновиться насправді.

Незважаючи на явне зазначення часу, протягом якого сторінка повинна знаходитися в кеші, вона може бути видалена з нього раніше. Це може відбутися в тому випадку, якщо для приміщення в кеші для нового елемента не вистачає місця. Завдяки такому механізму, реалізованому в ASP.NET можна не піклуватися про необхідність очищення пам’яті кеша.

37.2. Профілі кешування

Одна з проблем кешування виведення пов’язана з необхідністю вбудовування інструкції в сторінку – або в частину розмітки .Aspx, або в код класу. Хоча перший варіант (використання OutputCache) щодо ясний, все ж він породжує проблеми управління, якщо створюються десятки кешованих сторінок. Якщо ви хочете змінити кешування для всіх цих сторінок (наприклад, змінивши час знаходження об’єктів в кеші з 30 до 60 секунд), то доведеться модифікувати кожну сторінку. Крім того, ASP.NET знадобиться їх всіх перекомпілювати.

ASP.NET також дозволяє застосувати одні й ті ж налаштування кешування до цілої групи сторінок за допомогою засобу, званого профілями кеша. Профілі кешу дозволяють описувати налаштування кеша у файлі web.config. асоціювати ім’я з цими настройками і потім застосовувати їх до багатьох сторінок відразу, вказуючи це ім’я. Таким чином, ви отримуєте свободу модифікувати всі пов’язані сторінки за один раз – просто змінюючи відповідний профіль кешу у файлі web.config.

Для визначення профілю кеша використовується дескриптор <add> в розділі <outputCacheProfiles>. Тут профілем призначається ім’я та тривалість утримання об’єктів у кеші.

У розділі <outputCache> задаються параметри кешування сторінок, званого також кешуванням виводу (output caching). Ось його схема з використовуваними за замовчуванням значеннями:

<outputCache enableOutputCache = "true"
enableFragmentCache = "true"
sendCacheControlHeader = "true"
comitVaryStar = "false">
</outputCache>

Якщо кешування сторінок або кешування фрагментів (fragment caching) декларативно відключено, ні сторінки, ні елементи керування користувача не

кешуються, якими б не були програмні установки. Атрибут sendCacheControlHeader вказує, чи повинен модуль кешування сторінок за замовчуванням відправляти у відповіді браузеру заголовок cache-control:private; атрибут omitVaryStar включає і відключає відправку заголовка Vary:*.

У розділі <outputCacheSettings> містяться групи параметрів кешування, застосовуваних до сторінок за допомогою директиви @ OutputCache. Тут міститься тільки один дочірній розділ – <outputCacheProfiles>, де визначаються так звані профілі кешування сторінок (output cache profile), що представляють собою просто іменовані групи установок. Ось приклад:

<outputCacheSettings>
<outputCacheProfiles>
<add name="ServerOnly" duration="60" varyByCustom="browser" />
</outputCacheProfiles>
</outputCacheSettings>

Профіль ServerOnly визначає, що сторінку потрібно кешувати протягом 60 с і дозволяє зберігати різні версії сторінки для браузерів різних типів. Ось повна схема розділу <outputCacheProfiles>:

<outputCacheProfiles>
<add name = ""
enabled = "true"
duration = "–1"
location = ""
sqlDependency = ""
varyByCustom = ""
varyByControl = ""
varyByHeader = ""
varyByParam = ""
noStore = "false"/>
</outputCacheProfiles>

Нарешті, в розділі <sqlCacheDependency> містяться параметри, використовувані класом SqlCacheDependency при кешуванні інформації з баз даних SQL Server. Даний клас реалізує в ASP.NET функцію залежності від бази даних – користувальницької залежності, згідно з якою дані таблиць бази даних, що кешуються, видаляються, коли в ці таблиці вносяться зміни. З певною періодичністю він опитує таблиці, щоб дізнатися, які з них були змінені.

<sqlCacheDependency enabled="true" pollTime="1000">
<databases>
<add name="Northwind" connectionStringName="LocalNWind" />
</databases>
</sqlCacheDependency>

Атрибут pollTime задає період опитування (в мілісекундах). У наведеному прикладі таблиці, підлягають моніторингу, які повинні оптуватися кожну секунду. Посилання на базу даних, що підлягає моніторингу, міститься в розділі <databases>. Його атрибут name – це не ім’я бази даних, а всього лише ім’я залежності. Базу даних, а точніше рядок підключення до неї, ідентифікує

атрибут connectionStringName, який вказує на запис з розділу <connection-Strings> файлу web.config. Що стосується переліку таблиць, які підлягають моніторингу, то він залежить від результату роботи утиліти aspnet_regsql.exe, про яку мова піде далі в цьому розділі.

Установки, задані в розділі <sqlCacheDependency>, не виробляють ніякого ефекту, якщо ви використовуєте клас SqlCacheDependency спільно з механізмом повідомлення про зміну вихідних даних запиту (query notifications) з SQL Server.

37.3. Кешування і рядок запиту

Зрозуміло, іноді інформація повинна бути динамічною. Прикладом може служити ситуація, коли сторінка використовує інформацію поточного користувальницького сеансу для оформлення інтерфейсу. У цьому випадку повне кешування сторінки не підходить (хоча часткове може виявитися корисним). Інший приклад – коли сторінка отримує інформацію від іншої сторінки через рядок запиту. При цьому сторінка надто динамічна для того, щоб її можна було кешувати.

У розглянутому прикладі атрибут VaryByParam встановлюється в None. Це повідомляє ASP.NET, що ми хочемо зберігати тільки одну копію сторінки, що кешується, і яка підходить для всіх сценаріїв. Якщо запит до цієї сторінки буде додавати рядок аргументів до URL, це не має значення – ASP.NET весь час буде використовувати той самий висновок, поки він не застаріє. Ви можете перевірити це, додаючи параметр рядка запиту вручну у вікні браузера (як наприклад, A = b).

Вище вже відзначалися деякі потенційні проблеми, пов’язані з кешуванням сторінки виводу. Одним з таких випадків є використання даних, переданих сторінці через рядок запиту. При встановленому режимі кешування, розглянутому вище, незважаючи на введення в рядок адреси значень змінних, сторінка зберігається в кеші задані 10 секунд. Однак, таку поведінку можна змінити якщо вказати як значення параметра VaryByParam значення «*», яке встановлює режим кешування сторінок, що містять рядок запиту. При цьому, ASP.NET починає кешувати окремі копії сторінки для різних значень аргументів, зазначених у рядку запиту. Використання значення «*» в якості значення параметра VaryByParam в деяких випадках здатне викликати додаткові проблеми.

Справа в тому, що в деяких типах додатків досить часто використовується передача безлічі параметрів в рядку запиту. Якщо значення хоча б одного з переданих параметрів буде відрізнятися від значення такого ж параметра кешованої сторінки, запитувана сторінка буде кешувати знову. Наприклад, якщо в рядку запиту передається інформація про клієнта, а також про обраний ним товар, зрозуміло, що кількість комбінацій клієнта і товару дуже багато, а отже практично всі запитувані сторінки будуть поміщені в кеш, причому їх повторне використання буде прагнути до 0. У цьому випадку доцільно виявити ті параметри, які найбільш часто використовуються для передачі параметрів, з високим ступенем повторного використання і вказати їх як значення параметра VaryByParam. Наприклад, наступний рядок дає команду

ASP.NET для кешування тільки тих сторінок, які містять параметр ClientID, значення якого, в свою чергу, відрізняється від уже збереженого в кеші.

<%@ OutputCache Duration="30" VaryByParam="ClientName" %>

37.4. Фрагментне кешування

У ряді випадків буває недоцільно з якихось причин кешувати всю сторінку цілком. Це може бути в тому випадку, коли на сторінці знаходиться динамічний вміст, яке необхідно оновлювати постійно. У цьому випадку можна скористатися можливістю кешування фрагментів.

Реалізувати цей різновид кешування можна двома способами:

1. Створити користувальницький елемент управління, та помістити в нього ту частину сторінки, яку необхідно кешувати, та налаштувати для даного елемента кешування. При цьому для сторінки налаштовувати кешування не потрібно.

2. Створити свій метод, який повинен повертати деяке значення. За допомогою об'єкта Substitution вказати на необхідність виклику даного методу при зверненні користувача до Web-сторінці. При цьому сама сторінка може кешуватися, однак, даний метод буде виконуватися в будь-якому випадку, повертаючи фактично динамічний вміст, який буде вставлено в фрагмент сторінки, що кешується. Причому такий сценарій обробки буде застосовуватися завжди. Фактично, виходить реалізація, зворотна описаній в першому пункті.

37.5. Використання користувача елементу управління для реалізації фрагментного кешування

Щоб реалізувати кешування фрагментів, потрібно створити користувальницький елемент управління для частини сторінки, яку планується кешувати. Потім до цього елементу управління можна додати директиву OutputCache. У результаті вся сторінка не буде кешуватися, а для користувача елемент управління – буде.

Концептуальне кешування фрагментів – те ж саме, що і кешування сторінок. Є тільки одна пастка: якщо сторінка витягує кешовану версію користувача елементу управління, вона не може взаємодіяти з ним в коді. Наприклад, якщо елемент управління має властивості, то код веб-сторінки не може до них звертатися або модифікувати їх. Коли застосовується кешована версія користувача елементу управління, в сторінку просто вставляється блок HTML-розмітки. Відповідний об'єкт елемента керування не доступний.

37.6. Використання елемента керування Substitution для реалізації фрагментного кешування

Розглядаючи приклади, наведені вище неважко помітити, що параметри кешування для сторінки, на яку поміщаються елементи управління не встановлювалися. Це призводило до того, що кешуватися тільки ті елементи, для яких були встановлені відповідні параметри. Іноді буває корисніше вчинити навпаки, встановити параметри кешування для всієї сторінки, а для вмісту деяких елементів зробити можливим відсутність кешування. У цьому випадку необхідно скористатися елементом Substitution. Для роботи цього елементу управління необхідно назву статичного методу, що повертає динаміч-

ний вміст, який відображається на сторінці. Елемент Substitution може бути розміщений на сторінці так само, як і будь-які інші елементи управління, що дозволяє контролювати розташування його вмісту всередині сторінки.

37.7. Кешування даних

Кешування даних – найбільш гнучкий тип кешування, однак для своєї реалізації він вимагає виконання в коді ряду додаткових кроків. Базовий принцип кешування даних полягає в додаванні елементів, створення яких обходиться дорого, в спеціальний вбудований об'єкт колекції (званий Cache). Цей об'єкт працює подібно об'єкту Application. Він доступний глобально всім запитам від всіх клієнтів у додатку. Однак існує кілька ключових відмінностей:

- Об'єкт Cache є безпечним щодо потоків. Це означає, що не потрібно явно блокувати і розблокувати колекцію Cache перед додаванням або видаленням елемента. Проте, об'єкти в колекції Cache самі по собі повинні бути безпечними до потоків. Наприклад, створений користувальницький бізнес-об'єкт можуть спробувати використати більше одного клієнта одночасно, а це може привести до псування даних. Для обходу цього обмеження існують різні способи. Найпростіший спосіб, який буде продемонстрований в цьому розділі, полягає просто в створенні дублюючої копії об'єкта, якщо потрібно працювати з ним на веб-сторінці. Елементи з кешу видаляються автоматично ASP.NET видаляє елемент з кешу, якщо він застарів (минув його час існування), якщо змінюються об'єкти або файли, від яких він залежить, або ж якщо серверу не вистачає вільної пам'яті. Це означає, що кеш можна вільно використовувати, не піклуючись про витрату пам'яті сервера, тому що при необхідності ASP.NET видалить елементи. Але оскільки кешований об'єкт перш ніж намагатимуться використовувати його, в іншому випадку виникне виключення NullReferenceException.

- Елементи кеша підтримують залежності. Кешований об'єкт можна прив'язати до файлу, таблиці бази даних або до ресурсу іншого типу. Якщо цей ресурс змінюється, кешований об'єкт автоматично вважається недійсним і знищується. Як і стан додатка, кешований об'єкт приєднаний до процесу. Це означає, що він не повинен існувати після перезапуску домену програми, і не може бути розділений між комп'ютерами в веб-серверному кластері. Така поведінка закладена при проектуванні, тому що ціна яку довелося б заплатити за надання можливості багатьом комп'ютерам взаємодіяти з поза процесним кешем, звела б нанівець виграш в продуктивності. Тому більше сенсу дозволити кожному веб-серверу мати власний кеш.

37.7.1. Додавання елементів в кеш

Як і у випадку з колекціями Application і Session, додавати елемент у колекцію Cache можна простим привласненням нового імені ключа:

Cache ["key"] = item;

Однак такий підхід зазвичай не застосовується, тому що він не дозволяє отримати контроль над часом знаходження об'єкта в кеші. Більше кра-

щий підхід полягає в застосуванні методу Insert(). Чотири версії цього методу описані в таблиці 37.1.

Таблиця 37.1. Версії методу Insert

Перезавантаження	Опис
Cache.Insert (key, value)	Вставляє елемент у кеш під зазначеним ключовим іменем, використовуючи пріоритет і час існування за замовчуванням. Це еквівалентно застосуванню синтаксису колекції на основі індексу присвоюванню нового ключового імені
Cache.Insert (key, value, dependencies)	Вставляє елемент у кеш під зазначеним ключовим іменем, використовуючи пріоритет і час існування за замовчуванням. Останній параметр містить об'єкт Cache Dependency, пов'язаний з іншими файлами або кешувальними елементами і дозволяє оголошувати даний елемент недійсним у разі їх зміни
Cache.Insert (key, value, dependencies, absoluteExpiration, slidingExpiration)	Вставляє елемент у кеш під зазначеним ключовим іменем, використовуючи пріоритет і зазначену політику застаріння (одну із двох). Ця версія методу Insert() використовується найбільш часто
Cache.Insert (key, value, dependencies, absoluteExpiration, slidingExpiration, priority,onRemoveCallback)	Дозволяє конфігурувати всі аспекти політики кешування елемента, включаючи час існування, залежності і пріоритет. До того ж можна передати делегат, який вказує на метод, який повинен бути викликаний при видаленні елемента з кеш

37.8. Кешування за допомогою елементів управління джерелами даних

Ми витратили досить багато часу, розбираючись з елементами управління джерел даних. Всі елементи – SqlDataSource, ObjectDataSource і XmlDataSource – підтримують вбудоване кешування даних. Застосування кешування з цими елементами управління настійно рекомендується, так як елементи управління джерелами даних часто генерують додаткові запити. Наприклад, вони генерують повторний запит після кожної зворотної відправки, при зміні параметрів і виконують окремий запит для кожного прив'язаного елемента керування, навіть якщо в них використовуються однакові команди. Навіть невелике кешування може знизити ці накладні витрати.

На замітку! Хоча багато елементів управління джерелами даних підтримують кешування, це не обов'язкова властивість будь-якого елементу управління джерелом даних. Ви зустрінете такі елементи управління джерелами даних, які кешування взагалі не підтримують, причому для них це має сенс (наприклад SiteMapSource).

Для підтримки кешування всі елементи управління джерелами даних застосовують одні й ті самі властивості, які перераховані в таблиці 37.2.

Таблиця 37.2. Властивості елементів управління джерелами даних, що пов'язані з кешуванням

Властивість	Опис
EnableCaching	Якщо дорівнює true, то кешування включено. Значення за умовчанням є false

Продовження табл. 37.2

CacheExpirationPolicy	Використовує значення з перерахування DataSourceCache Expirity: Absolute – для абсолютного застаріння (коли вказується фіксований час перебування об'єкта в кеші) або Sliding – для ковзаючого застаріння (коли тимчасове вікно скидається при кожному витяганні об'єкта з кешу)
CacheDuration	Час в секундах перебування об'єкта в кеші. Якщо використовується ковзне застаріння, тимчасова межа скидається кожен раз, коли об'єкт витягується з кеша. Значення за замовчуванням – 0 (або Infinite) – дозволяє зберігати кешовані елементи нескінченно
CacheKey Dependency і SqlCacheDependency	Дозволяє встановити залежність одного кешованого елемента від іншого (CacheKey Dependency) або від таблиці в базі даних (SqlCacheDependency).

37.9. Кешування за допомогою SqlDataSource

Коли ви вмикаєте кешування для елемента керування SqlDataSource, то кешуєте результати SelectQuery. Однак якщо ви створюєте запит, який приймає параметри, то SqlDataSource буде кешувати результати окремо для кожного набору значень цих параметрів.

Наприклад, припустимо, що створена сторінка, яка дозволяє переглядати співробітників по містах. Користувач обирає потрібне місто у вікні списку, і ви застосовуєте елемент SqlDataSource для заповнення екранної сітки відповідними записами про співробітників. Для наповнення сітки використовується наступний елемент SqlDataSource:

<asp:SqlDataSource ID="sourceEmployees" runat="server" ProviderName="System.Data.SqlClient" ConnecntionString="<%$ ConnectionStrings: Northwind $>"

SelectCommand="SELECT EmployeeID, FirstName, LastName, Title, City FROM Employees WHERE City=@City">

<SelectParameters>

<asp:ControlParameter ControlID="lstCities" Name="City" PropertyName="SelectedValue"/>

</SelectParameters>

</asp:SqlDataSource>

У цьому прикладі кожен раз, коли обирається місто, виконується окремий запит, щоб отримати записи про співробітників тільки з цього міста. Запит застосовується для наповнення DataSet, який потім кешується. Якщо обирається інше місто, процес повторюється, і новий DataSet кешується окремо. Однак якщо ви вкажете місто, яке було раніше обраним вами або іншим користувачем, то відповідний DataSet витягується з кешу (якщо тільки він не був видалений з причини закінчення відведеного часу).

На замітку! Кешування SqlDataSource працює, тільки коли властивість DataSource Mode встановлено в DataSet (за замовчуванням так і є). Воно не працює, коли встановлено режим DataReader, тому що об'єкт DataReader підтримує активне з'єднання з базою даних і не може ефективно кешуватися.

Кешування окремих результатів для різних значень параметрів працює добре, якщо деякі значення параметрів використовуються частіше за інших. Наприклад, якщо результати по Лондону запитуються набагато частіше, ніж результати по Редмонду, це гарантує, що результати по Лондону залишатимуться в кеші, навіть коли об'єкт DataSet для Редмонда буде видалений з пам'яті. Припускаючи, що повний набір результатів надзвичайно великий, це може виявитися найбільш ефективним підходом.

З іншого боку, якщо всі значення параметрів використовуються приблизно з однаковою частотою, цей підхід не такий гарний. Одна з проблем, які він породжує, складе в тому, що коли елементи в кеші застаріють і будуть видалені, потрібно безліч запитів до бази даних, щоб заново наповнити кеш (по одному на кожне значення параметра), що не так ефективно, як отримання комбінованого результату в єдиному запиті.

Якщо ви зіштовхуєтеся з іншою ситуацією, то може змінити SqlDataSource так, щоб він витягав DataSet з повним списком всіх співробітників і поміщав його в кеш. Потім SqlDataSource може витягувати тільки потрібні записи для задоволення кожного параметризованого запиту до DataSet. У цьому випадку єдиний DataSet з повним набором всіх записів буде поміщений в кеш і зможе задовольнити запит з будь-яким значенням параметра. Щоб застосувати цей прийом, знадобиться переписати SqlDataSource для використання фільтрації. По-перше, запит повинен повертати всі рядки і оператор SELECT не повинен мати параметрів:

<asp:SqlDataSource ID="sourceEmployees" runat="server"

SelectCommand="SELECT EmployeeID, FirstName, LastName, Title, City FROM Employees" …>

</asp:SqlDataSource>

По-друге, необхідно визначити вираз фільтра. Це та частина, яка потрапляє в конструкцію WHERE звичайного SQL-запиту, і вона записується аналогічно тому, як це робилося у властивості DataView.RowFilter. (В дійсності SqlDataSource використовує "за лаштунками" фільтрацію рядків DataView.) Однак тут присутня пастка – якщо значення фільтра виходить з іншого джерела (такого як елемент управління), знадобиться визначити один або більше наповнювачів, використовуючи синтаксис {0} для першого з них, {1} – для другого і т.п. Потім у розділі <FilterParameters> передаються значення фільтра – майже таким же чином, як вказувалися параметри вибірки в першій версії:

Ось так виглядає повний дескриптор об'єкта SqlDataSource:

<asp:SqlDataSource ID="sourceEmployees" runat="server" ProviderName="System.Data.SqlClient" ConnectionString="<%$ ConnectionStrings:Northwind $>"

SelectCommand="SELECT EmployeeID, FirstName, LastName, Title, City FROM Employees" FilterExpression="City=`{0}`" EnableCaching="True">

<FilterParameters>

<asp:ControlParameter ControlID="lstCities" Name="City" PropertyName="SelectedValue"/>

</FilterParameters>

</asp:SqlDataSource>

Контрольні запитання

1. Що таке кешування?
2. Які переваги використання кешування даних в ASP.NET?
3. Які типи кешування підтримує ASP.NET?
4. Для чого потрібні профілі кешу?
5. Якими способами можна реалізувати фрагментне кешування?
6. Що таке концептуальне кешування фрагментів?
7. Яка властивість задає час перебування об’єкта в кеші в секундах?
8. Що таке кешування даних?
9. Які існують відмінності між Cache і Application?
10. Для чого використовується директива OutputCache?

Тема 38. Робота з XML

Абревіатура XML розшифровується як Extensible Markup Language, в перекладі „розширювана мова розмітки”. Як і мова HTML, вона є підмножиною SGML (Standard General Markup Language) – „дідуся” мов розмітки. Ми вже не раз стикалися з форматом XML. Такий формат конфігураційних файлів, файлу опису об’єктних джерел даних.

XML – це універсальний, незалежний від платформи стандарт опису інформації, який можна використовувати для представлення ієрархічних даних та уніфікації переданої інформації. Без його знання неможливе розуміння SOAP і отже, веб-сервісів. XML став де-факто стандартом передачі даних в мережі Інтернет. Стандарт XML і пов’язаних з ним форматів визначається консорціумом W3C (World Wide Web Consortium). Наприклад, ми створюємо aspx сторінки у форматі XHTML – перехідному між HTML і XML, стандарт якого теж визначений W3C. Стандарт XHTML накладає більш суворі правила на правильне формування документа, аналогічні правилам XML.

Давайте зрозуміємо головну відмінність XML від HTML. XML створений для опису даних і фокусується на тому, що саме вони з себе представляють. HTML створений для демонстрації даних і фокусується на тому, як дані виглядають. Якщо в традиційному HTML поняття „уявлення” і „візуалізація” часто змішуються, то під час роботи з XML ми чітко поділяємо ці поняття. Теги XML не призначені творцями мови, на відміну від тегів HTML. Кожен автор документа сам визначає власні теги.

Стандарт вимагає, щоб програма, яка обробляє XML-документ, була повинна зупинити роботу, якщо виявила помилку. А якщо браузер виявить незрозумілий тег в HTML, або відсутність тега, що закривається, він це просто ігнорує.

На початку XML-документа обов’язково з’являється його декларація, або пролог. У ньому вказується версія стандарту XML, якому він відповідає.

<?xml version="1.0" encoding="utf–8" ?>

Декларація не є частиною XML-документа і не має закриває тега. У тексті XML-файла можуть перебувати коментарі в стилі HTML – <!–Text–>.

XML-документ може мати тільки один кореневий елемент. У нього можуть бути вкладені інші вузли, а в них, у свою чергу – інші. Кожному відкриваючому тегу XML повинен відповідати тег, що закривається. Після за-

вершального тега кореневого елемента не може бути інших тегів. Теги XML чутливі до регістру (case-sensitive). Теги повинні бути цілком вкладені один в одного, тому, код, допустимий в HTML:

Какой-то текст

є помилкою в XML.

У тегів можуть бути атрибути. Значення атрибутів повинні бути укладені в лапки. Порядок атрибутів значення не має. Між тегами, що відкриваються і закриваються, може знаходиться текст. У XML зберігаються всі прогалини, що знаходяться в тексті. Якщо тексту немає, можна застосувати скорочену форму запису. Приклад тега XML:

<PROPERTY Label="ogl_extension" Value="4520" Itemtype= "predefined" />

Це коротка форма тега:

<PROPERTY Label="ogl_extension" Value="4520" Itemtype= "predefined"></PROPERTY>

Вам це нічого не нагадує? Правила опису елементів ASP.NET точно такі ж.

Існує атрибут xmlns, який визначає простір імен. Значним його може бути будь унікальне ім'я. Існує домовленість використовувати URL, так як вони унікальні. Простору імен мають сенс, аналогічний їх застосування в .NET Framework – щоб не змішувати однакові імена, використовувані різними розробниками. Назва простору імен відділяється від імені двокрапкою.

XML-файли являють ієрархічну інформацію, яку можна представити у вигляді дерева з одним коренем.

Документи XML, що задовольняють всім вимогам синтаксису, називають правильними (well-formed). Для опису даних XML використовує DTD (Document Type Definition) – визначення типу документа. Якщо файл відповідає DTD, він вважається дійсним (valid).

Браузери IE 6.0, FireFox 1.5 відображають XML-файли з виділенням синтаксису. Батьківські вузли можна розкривати і закривати. Наприклад, в закритому вигляді кореневий вузол файлу BirthDay.xml виглядає так:

<BD>

Якщо його розкрити, побачимо:

```
<BD>
  <Item Type="Plugin">
   <LinkText> Відправити вітальну листівку </LinkText>
   <PluginID>Friendship</PluginID>
   <InitData />
   </Item>
  <Item Type="URL">
   <AdditionalText> Відправити вітальну листівку </AdditionalText>
   <URL>www.icq.com</URL>
  </Item>
</BD>
```

Середовища розробки Visual Studio і VWD Express перевіряють правильність xml-документів прямо під час редагування.

38.1. Введення

Розширювана мова розмітки (XML) спочатку була задумана як мова для опису нових форматів документів World Wide Web. XML походить від Стандартної узагальненої мови розмітки (Standard Generalized Markup Language – SGML) і може вважатися метамовою: мовою для визначення мов розмітки. SGML і XML – це орієнтовані на текст формати, які забезпечують механізми опису структур документів за допомогою тегів розмітки (слів, узятих в кутові дужки ‘<‘i‘>‘). Web-розробники можуть помітити деяку схожість між HTML і XML, обумовлену тим фактом, що вони обидва походять від SGML.

Оскільки застосування XML зросла, зараз загальноприйнято вважати, що XML корисний не тільки при описі нових форматів документів для Web, але також підходить для опису структурованих даних. Приклади структурованих даних включають інформацію, яка зазвичай міститься в великоформатних таблицях, файлах конфігурації програми і мережевих протоколах.

XML є кращим для дотеперішніх форматів даних, тому що XML може запросто уявити і табличні дані (такі як реляційні дані з бази даних або великих таблиць), і псевдоструктурірованние дані (такі як Web-сторінки або ділові документи). Популярні ранні формати, такі як файли з розділяються комою значеннями (CSV), або підходять для табличних даних і погано описують псевдоструктурірованние дані, або, як RTF, занадто спеціалізовані для псевдоструктурірованних текстових документів. Це призвело до широкого поширення XML як мови для обміну інформацією.

38.2. XML всюди

Крім здатності представляти і структуровані, і псевдоструктурірованние дані, XML має кілька характеристик, які зумовили його широке використання в якості формату представлення даних. XML – розширювана платформа – незалежна і що підтримує локалізацію, т.я. повністю сумісна з Unicode. Той факт, що XML – текстовий формат, означає, що при виникненні необхідності XML – документи можна читати і редагувати, використовуючи стандартні інструменти редагування текстів.

Розширюваність XML проявляється багатьма способами. Насамперед, на відміну від HTML, у нього немає фіксованого словника. З XML кожен може визначити спеціальні словники для конкретних програм або різних галузей промисловості. По-друге, додатки, що обробляють або використовують формати XML, більш стійкі до змін у структурі пропонованого ним XML, ніж додатки, які використовують інші формати. Наприклад, додаток, який залежить від обробки елемента <Customer> з атрибутом customer – id, зазвичай не повинно перериватися, якщо інший атрибут, такий як last – purchase – date, був доданий в елемент <Customer>. Така гнучкість невластива іншим форматам даних і є істотною перевагою використання XML.

XML не прив’язаний ні до однієї мови програмування, операційній системі або постачальнику програмного забезпечення. До речі, створювати або споживати XML, використовуючи різні мови програмування – занадто прямолінійно. Незалежність від платформ робить XML дуже корисним в якості

засобу досягнення можливості взаємодіяти між різними платформами програмування та операційними системами.

Переваги подання даних у вигляді XML були визнані багатьма і призвели до поширення XML-джерел даних. Ділові документи, бази даних і межделовое спілкування – все це приклади інформаційних джерел, які переходять або перейшли до використання XML як формату представлення. Такі продукти Microsoft як Microsoft Office, Microsoft SQL Server і Microsoft.NET Framework дають можливість кінцевим користувачам і розробникам створювати і використовувати документи, мережеві повідомлення та інші дані у вигляді XML.

38.2. Порівняння XML і HTML

І HTML, і XML документи складаються з елементів, кожен з яких включає „початковий тег" (<order>), „кінцевий тег" (</order>) і інформацію, укладену між цими двома тегами (яка називається вмістом елемента). Елементи можуть бути анотовані атрибутами, що містять метадані про елемент і його вміст.

Однак між HTML і XML є істотні відмінності. XML чутливий до регістру, в той час як HTML – ні. Це означає, що в XML початкові теги <Table> і <table> різні, тоді як в HTML – це одне і те ж. Інша відмінність між HTML і XML у тому, що XML представляє концепцію правильної побудови. Правила побудови XML усувають деяку невизначеність, притаманну обробці таких мов розмітки як HTML, вводячи такі постулати, як те, що всі значення атрибутів повинні бути укладені в лапки, і що у всіх елементів повинні бути або початковий і кінцевий теги, або явне вказівку того, що це порожні елементи. Короткий опис правил побудови дається в XML FAQ в розділі D.2.

Найсуттєвіша відмінність між HTML і XML у тому, що в HTML є зумовлені елементи і атрибути, поведінка яких зумовлена, в той час як в XML такого немає. Замість цього, автори документа можуть створювати власні XML-словники, призначені саме для їх застосування або ділових потреб. В даний час існують XML-словники для величезної кількості галузей промисловості і додатків: від фінансових картотек (XBRL) і фінансових операцій (FpML) до Web- документів (XHTML) і мережевих протоколів (SOAP). Відсутність визначених елементів і атрибутів, які визначають, як формується або відображається XML-документ, дає можливість авторам зосередитися на створенні документів, які містять тільки істотну семантичну інформацію в їх конкретної предметної області. Відділення вмісту від уявлення, що стало можливим з XML-словниками, істотно збільшує можливості повторного використання інформації та перенацілювання вмісту.

38.3. Анатомія XML-документа

Нижче наведено приклад XML – документа, який представляє замовлення покупця в магазині музичних CD. На що треба звернути увагу – це на те, як просто документ представляє і строго структуровані дані, які описують інформацію про компакт дисках, і псевдоструктуровані дані, що містять спеціальні інструкції і коментарі по визначеному покупцеві.

<?xml version="1.0" encoding="iso–8859–1" ?>

```
<?xml-stylesheet href="orders.xsl"?>
<order id="ord123456">
 <customer id="cust0921">
  <first-name>Dare</first-name>
  <last-name>Obasanjo</last-name>
  <address>
   <street>One Microsoft Way</street>
   <city>Redmond</city>
   <state>WA</state>
   <zip>98052</zip>
  </address>
 </customer>
 <items>
  <compact-disc>
   <price>16.95</price>
   <artist>Nelly</artist>
   <title>Nellyville</title>
  </compact-disc>
  <compact-disc>
   <price>17.55</price>
    <artist>Baby D</artist>
    <title>Lil Chopper Toy</title>
  </compact-disc>
 </items>
<! - Доставка повинна бути проведена в будь-якому випадку ->
<special-instructions xmlns:html="http://www.w3.org/1999/xhtml/">
<html:p> Якщо замовника немає за вказаною адресою, тоді спробуйте
залишити пакет в одному з наступних місць, список яких наведено в порядку
пріоритетності доставки
<html:ol>
<html:li> Сусідня двері </ html : li >
<html:li> У консьєржа </ html : li >
<html:li> На порозі </ html : li >
</ html : ol >
<html:b> Примітка </ html : b > Не забудьте залишити записку з точ-
ним зазначенням, де знаходиться пакет.
</ html : p >
</ special - instructions >
</ order >
```

Документ починається з необов'язкового опису XML, у якому вказується, яка версія XML та кодування символом використовуються. Далі слід інструкція обробки xml-stylesheet, яка використовується для зв'язування таблиці стилів, що містить інструкції щодо форматування, з XML-документом.

Таблиця стилів використовується для формування привабливого зовнішнього вигляду документа в користувальницьких додатках, таких як Web-

браузери. Інструкції обробки зазвичай використовуються для введення інформації про програму в XML-документ. Наприклад, більшість програм, що обробляють вміст наведеного вище документа, ймовірно, проігнорують інструкцію обробки xml-stylesheet. З іншого боку, додатки, що використовуються для відображення XML – документа, такі як Web-браузер, могли б використовувати інформацію інструкції обробки для того, щоб визначити, де розташовується таблиця стилів, яка містить спеціальні інструкції для відображення документа.

38.4. AdRotator

Елемент управління AdRotator дозволяє показувати рекламні банери і автоматично замінювати їх на інші. Самі банери описані у файлі XML або іншому джерелі даних. Реклама оновлюється кожного разу при оновленні сторінки. У властивості AdvertismentFile задається ім’я XML-файла. Скелет XML-файла такий:

```
<?xml version="1.0" encoding="utf–8" ?>
 <Advertisements
 xmlns="http://schemas.microsoft.com/AspNet/AdRotator-Schedule-File">
 </Advertisements>
```

Усередині вузла Advertisements розташовуються вузли <Ad> </ Ad> У цих вузлів є 5 атрибутів, всі вони необов’язкові (табл. 38.1).

Таблиця 38.1. Властивості вузла Advertisements

ImageUrl	Картинка, яка буде демонструватися при виборі даного оголошення
NavigateUrl	Адреса, за якою буде здійснений перехід при натисканні на картинку
AlternateText	Альтернативний текст, якщо показ зображень вимкнений
Impressions	Всі значення Impressions сумуються. Ймовірність показу реклами дорівнює значенню Impressions, поділеному на цю суму.
Keyword	Ключове слово-категорія реклами, дозволяє фільтрувати оголошення

Приклад файлу AdvertismentFile, який називається ads.xml:

```
<?xml version="1.0" encoding="utf–8" ?>
<Advertisements
xmlns="http://schemas.microsoft.com/AspNet/AdRotator-Schedule-File">
<Ad>
<ImageUrl>fixed.gif</ImageUrl>
<NavigateUrl>http://www.im.am</NavigateUrl>
<AlternateText>  Безкоштовний хостинг </AlternateText>
<Impressions>40</Impressions>
<Keyword>Хостинг</Keyword>
</Ad>
<Ad>
<ImageUrl>logo2.jpg</ImageUrl>
<NavigateUrl>http://www.nv.am</NavigateUrl>
<AlternateText>Газета "Новий час"</AlternateText>
<Impressions>50</Impressions>
<Keyword>Новини</Keyword>
```

</Ad>
<Ad>
<ImageUrl>summer.jpg</ImageUrl>
<NavigateUrl>http://www.utro.ru</NavigateUrl>
<AlternateText>Співачку Жасмін побив чоловік!</AlternateText>
<Impressions>100</Impressions>
<Keyword>Жовті новини</Keyword>
</Ad>
</Advertisements>

На сторінку поміщений елемент управління. Його властивість AdvertisementFile вказує на цей файл.

<asp:AdRotator ID="AdRotator1" runat="server" AdvertisementFile= "ads.xml" Height="164px" Width="574px" />

Якщо встановлено властивість Keyword, то елемент управління показує тільки ту рекламу, яка відповідає його змісту. Так як його можна міняти динамічно, можна підлаштовувати рекламу під потреби користувача. Keyword повинен зустрічатися хоча б раз у файлі оголошень, інакше замість реклами буде порожній прямокутник.

У попередніх версіях ASP.NET можна було працювати тільки з файлами XML. Тепер можна використовувати будь-яке джерело даних, зв'язавшись з елементом управління – джерелом даних. У такому випадку необхідно вказати як мінімум 3 поля джерела у властивостях ImageUrlField, NavigateUrlField і AlternateTextField.

<asp:AdRotator ID="AdRotator2" runat="server" DataSourceId= "SqlDataSource1" AlternateTextField="Alternate" ImageUrlField="Image" NavigateUrlField="NavigateUrl" />

38.5. Файли перетворення документа

Відомо, що для форматування HTML-файлів часто використовуються CSS (Cascading Stylesheets), хоча це необов'язково, оскільки браузери співвідносять з усіма тегами певний зовнішній вигляд. Елемент <p> задає параграф, <B> – напівжирний шрифт – браузер знає, як їх показувати.

Оскільки XML не використовує спочатку задані теги, їх значення може бути яким завгодно: <table> може означати таблицю HTML, а може і дерев'яний стіл. Тому браузери показують XML-документи „як є". Можна задати CSS файли і для XML-документів, але це не рекомендується.

Для того, щоб задати формат відображення XML-документів, використовуються таблиці стилів XSL. XSL – розширювана мова стилів (Extensible Stylesheet Language) набагато більш багатий можливостями, ніж CSS. XSL – більше, ніж просто таблиця стилів.

Один і той же файл XML можна пов'язати з різними таблицями XSL, в тому числі програмно.

XSL складається з 3 частин:

1. XSLT – методу перетворення XML-документів
2. XPath – методу завдання частин і шляхів до елементів XML

3. XSL Formatting Objects – методу форматування XML-документів.

Найважливіша частина XSL – це мова перетворень XSLT (XSL Transformation). Вона застосовується для перетворення XSL-документів в інші типи документів або інші XSL-документи. Часто XSLT використовується для перетворення XSL-документа у формат HTML.

Для того, щоб створити XSLT-документ, оберіть у діалозі створення файлу XSLT file. VS 2005 створює каркас таблиці стилів. Так як таблиця стилів сама по собі є XML-документом, вона починається з декларації XML.

<?xml version="1.0" encoding="utf–8"?>

Тег xsl: stylesheet задає початок таблиці стилів.

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

</xsl:stylesheet>

Якщо ви вивчали CSS, то знаєте, що для завдання стилів використовуються правила. Правило складається з селектора і опису стилю в фігурних дужках.

```
a {
font-size:medium;
color:Fuchsia;
}
```

Це правило визначає стиль виводу гіперпосилань середнього розміру шрифтом фіолетовим кольором.

У XSL використовуються шаблони. Для зв’язування шаблону з XML-елементом використовується атрибут відповідності.

Тег xsl: template задає початок шаблону. Атрибут шаблону match = "/"> пов’язує шаблон і кореневий елемент вихідного XML-документа.

<xsl:template match="/">

</xsl:template>

У цей тег вкладений шаблон HTML-файла. Коментар нагадує про те, що туди потрібно вставити XSL-елементи. Створення файлів XSLT розглянемо на прикладі. Створіть у папці App_Data файл XML "Quotes.xml"

```
<?xml version="1.0" encoding="utf–8" ?>
<!- цитати великих людей -->
<Quote>
<Text> Хотіли як краще, а вийшло як завжди.</Text>
<Author> Віктор Черномирдін </Author>
</Quote>
<Quote>
<Text> Америка - континент, названий так тому, що його відкрив Колумб.</Text>
<Author>Жорж Елгозі</Author>
</Quote>
<Quote>
<Text> Я шаленію від однієї думки про те, скільки б я всього дізнався, якби не ходив до школи.</Text>
<Author>Джордж Бернард Шоу</Author>
</Quote>
```

<Quote>
<Text> Багато чого придумано для того, щоб не думати.</Text>
<Author>Карел Чапек</Author>
</Quote>
<Quote>
<Text> Якщо скажеш правду, все одно рано чи пізно потрапиш.</Text>
<Author>Оскар Уайльд</Author>
</Quote>
<Quote>
<Text> Бути йому президентом, якщо його до того часу не повісять.</Text>
<Author>Марк Твен</Author>
</Quote>
</Quotes>

Щоб внести у вихідний потік XSLT-перетворення кожен XML-елемент, застосовується тег XSL xsl: for-each. Елемент: for-each визначає місце розташування елементів в XML-документі і повторює шаблон для кожного з них.

<xsl:for-each select="Quotes/Quote"> </xsl:for-each>

Все, що знаходиться в шаблоні, буде виводитися стільки разів, скільки в першому дкументі встрітиться елемент Quote, заключений в тег Quotes.
Для того щоб внести в вихідний потік XSLT-перетворення значень XML-елемента, використовується тег XSL xsl:value-of:

<xsl:value-of select="Text"/>
<hr/>

Дані можна відсортувати за допомогою тега xsl: sort, який повинен знаходитися всередині елемента xsl: for-each:

<xsl:sort select="Author" />

XSL може застосовувати умови для показу і форматування інформації залежно від значень елементів. „Умовний оператор” має вигляд <xsl:choose>, в який вкладено елементи <xsl:when> і можливо, <xsl:otherwise>. Умова задається в елементі <xsl:when> за допомогою парамерта test:

<xsl:choose>
<xsl:when test="Author=‘Марк Твен’">
<img src="http://www.tvkultura.ru/p/q_14406.jpg"></img>
</xsl:when>

Остаточний вигляд файлу трансформації:

```
<?xml version="1.0" encoding="utf–8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 version="1.0">
 <xsl:template match="/">
  <html>
   <body>
    <h1 style="background-color: RoyalBlue; color: white;
     font-size: 24pt; text-align: center; letter-spacing: 1.0em">
```

```
    відомі цитати
   </h1>
   <table border="0">
    <tr style="font-size: 12pt; font-family: verdana;
     font-weight: bold">
     <td style="text-align: center">Цитата</td>
     <td style="text-align: center">Автор</td>
    </tr>
    <xsl:for-each select="Quotes/Quote">
     <xsl:sort select="Author" />
     <tr style="font-size: 10pt; font-family: verdana">
      <td>
       <xsl:value-of select="Text"/></td>
      <td>
       <xsl:choose>
        <xsl:when test="Author='Марк Твен'">
         <img src="http://www.tvkultura.ru/p/q_14406.jpg"></img>
        </xsl:when>
        <xsl:otherwise>
        <i><xsl:value-of select="Author"/></i>
        </xsl:otherwise>
       </xsl:choose>
      </td>
     </tr>
    </xsl:for-each>
   </table>
  </body>
 </html>
</xsl:template>
</xsl:stylesheet>
```

Якщо хочете побачити результат перетворення документа в браузері, включіть після XML-декларації оголошення

<?xml-stylesheet type="text/xsl" href="XSLTFile.xsl"?>,

або оберіть у меню XML пункт Show XML Output і визначте файл перетворення.Той же самий XML-документ можна перетворити за допомогою іншого XSL-файлу:

```
<?xml version="1.0" encoding="utf-8"?>
 <xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="/">
    <html>
    <body>
     <h1 style="background-color: Brown; color: white;
        font-size: 24pt; text-align: center; letter-spacing: 1.0em">
      Майстри Афоризму
```

```
</h1>
<xsl:for-each select="Quotes/Quote">
  <xsl:value-of select="Text"/>
 <br/>
  <xsl:value-of select="Author"/>
 <hr width="70%"/>  </xsl:for-each>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

У результаті будуть виводитися цитати, розділені горизонтальною лінією.

38.6. Файли визначення схеми документа

Відповідно до сучасного стандарту, пройшов стандартизацію документ повинен відповідати пов’язаному з ним файлу XSD (XML Schema Definition) - файлу визначення схеми XML, який визначає конкретну мову, тобто описує, які елементи і типи можуть з’являтися в документі. Схеми XSD покликані замінити DTD (Document Type Definition), різниця між ними полягає в тому, що файли XSD самі теж використовують синтаксис XML. Схеми XSD дозволяють визначити, які теги дозволені, обов’язкові вони чи ні, чи можуть повторюватися в документі і так далі. Таким чином, XML описує дані, а XSD - структуру цих даних, або метадані. У термінах програмування, XSD – опис типів, в той час як в XML-файлі описані об’єкти цих типів. За адресою http://www.w3.org/TR/2003/WD-xmlschema-11-req-20030121/ знаходиться робочий проект стандарту XSD 1.1.

Файл опису схеми починається з опису префікса простору імен, який включається потім у всі елементи цього файлу. Адреса http://tempuri.org призначається для завдання URI для просторів імен ASP.NET.

```
<xs:schema id="XMLSchema2" targetNamespace="http://tempuri.org/XMLSchema2.xsd" elementFormDefault="qualified"
xmlns="http://tempuri.org/XMLSchema2.xsd"
xmlns:mstns="http://tempuri.org/XMLSchema2.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

Створюючи схеми XSD, можна:

1. Декларувати елементи і атрибути

```
<xs:element name="Author" type="xs:string" default="Пушкин" minOccurs="1" maxOccurs="1" />
```

Наприклад, це визначення задає, що елемент "Author" строкового типу, повинен з’являтися один і тільки один і раз, і якщо він не вказаний, то приймає значення "Пушкін".

```
<xs:element name="Child" type=" xs:string" maxOccurs="unbounded"/>
```

Параметр maxOccurs = "unbounded" вказує, що елемент може зустрічатися будь-яку кількість разів.

Параметр ref дозволяє посилатися на вже описаний в даному файлі глобальний елемент або атрибут, щоб уникнути повторного опису одних і тих же елементів.

2. Визначати прості і складні типи.

У XSD є зумовлені типи – приблизно такі ж, як в .NET. Під час роботи програми вони перетворюються на типи .NET. На їх підставі можна будувати складні типи, схожі на структури мов програмування. Складний тип складається з послідовності описів елементів. Визначимо складний тип:

```
<xs:complexType name="Quote">
<xs:sequence>
<xs:element name="Text" type="xs:string" minOccurs="1" maxOccurs="1" />
<xs:element name="Author" type="xs:string" default="Пушкин" minOccurs="1" maxOccurs="1" />
</xs:sequence>
</xs:complexType>
```

Тег <xs:sequence> визначає, що елементи в даному типі повинні з’являтися в заданому порядку. Якби використовувався тег <xs:all>, то елементи могли б з’являтися в будь-якому порядку.

Тег <xs:choice> схожий на структуру з варіантами. Він визначає, що в елементі данног типу повинен бути тільки один з вкладених елементів.

```
<xs:complexType name="StateProvinceType">
 <xs:choice>
  <xs:element name="State" type="xs:string"/>
  <xs:element name="Province" type="xs:string"/>
 </xs:choice>
</xs:complexType>
```

Прості типи теж будуються на основі стандартних типів, накладаючи різні обмеження. Типи можуть бути глобальними або вкладеними у визначення елементів. У попередньому прикладі визначений глобальний складний тип Quote.

Глобальний тип можна використовувати у визначенні елементів.

```
<xs:element name="Quote" type="Quote" maxOccurs="unbounded" />
```

У наступному прикладі визначений простий тип, вкладений у визначення елемента MyValue.

```
<xs:element name="MyValue" type="MyInteger"/>
  <xs:simpleType name="MyInteger">
   <xs:restriction base="xs:positiveInteger">
    <xs:minInclusive value="1"/>
    <xs:maxInclusive value="10"/>
   </xs:restriction>
  </xs:simpleType>
```

Значеннями цього типу можуть бути цілі позитивні числа від 1 до 10.

Простий тип може бути перерахуванням:

```
  <xs:simpleType name="Answers">
   <xs:restriction base="xs:string">
```

```
<xs:enumeration value="yes">
</xs:enumeration>
<xs:enumeration value="no" />
<xs:enumeration value="don’t know" />
</xs:restriction>
</xs:simpleType>
```

3. Додавати нові групи і групи атрибутів

У визначенні складного типу можуть фігурувати атрибути. Припустимо, ми хочемо побудувати схему такого файлу:

```
<?xml version="1.0" encoding="utf–8" ?>
<FilmChoices>
<Film Title=‘ Броненосець "Потьомкін"‘>
<Year>1925</Year>
<Director>Эйзенштейн</Director></Film>
<Film Title="Війна та мир">
<Year>1967</Year>
<Director>Сергій Бондарчук</Director> </Film>
<Film Title ="П’ята рота">
<Year>2005</Year>
<Director>Федір Бондарчук</Director> </Film>
</FilmChoices>
```

Необхідно вимагати наявність атрибута Title:

```
<xs:attribute name="Title" type="xs:string"  use="required"/>
```

Атрибути можуть бути тільки простих типів.

4. Додавати анотації.

Анотації дозволяють вставляти опис існуючих елементів, таким чином у файл додається документація.

```
<xs:complexType name="Quote">
<xs:annotation>
<xs:documentation>
Цитати різних авторів
</xs:documentation>
</xs:annotation>
```

<xs:documentation> призначається для читачів файлу, а <xs:appinfo> для обробних файл програм.

Повний опис синтаксису XSD можна прочитати за адресою http://www.w3.org/2001/XMLSchema.xsd

Редагувати XSD-файли в Visual Studio 2005 можна і через вихідний код, і за допомогою дизайнера. Для XML-документа можна автоматично згенерувати соответсвующую йому схему. У вікні властивостей XML-документа можна задати як файл схеми, так і файл перетворення. У такому випадку студія автоматично перевіряє файл на відповідність схемі, і навіть IntelliSense підставляє теги з цього файлу.

38.7. Клас XmlReader

За допомогою класу XmlReader можна швидше, ніж іншими методами, отримати дані з XML-документів.

XmlReader – це абстрактний клас. Щоб почати читання, в статичний метод Create передається об’єкт класу XmlReaderSettings. Ця функція підраховує кількість вузлів в документі.

```
using System.Xml;
using System.IO;
   private int CountNodes(string xmlFile)
   {
      int NodesCount=0;
      XmlReaderSettings settings = new XmlReaderSettings();
      settings.IgnoreWhitespace = true;
      settings.IgnoreComments = true;
      using (XmlReader reader = XmlReader.Create(xmlFile, settings))
      {          while (reader.Read())
        {              if (reader.NodeType == XmlNodeType.Element)
          {                NodesCount++;            }
        }
      }
      return NodesCount;
   }
```

Клас XmlReader дозволяє витягувати з документа класи CLR. Нехай у нас є меню ресторану.

```
<?xml version="1.0"?>
<pizza_menu>
 <food name="Піца грандіозо ">
  <price>450.00</price>
  <description>  Гриби, бекон, салямі, шинка,
   баварські сосиски, артишоки,
   висушені на сонці помідори,
   сир Пармезан </description>
  <calories>700</calories>
 </food>
 <food name="Прего піца">
  <price>306.00</price>
  <description>
    Шматочки ніжною курячої грудки в соусі Песто,
     червоний солодкий перець, гриби,
     кукурудза, сир Пармезан
  </description>
  <calories>650</calories>
 </food>
 <food name="Піца Маргарита">
  <price>126.00</price>
```

```
      <description>
       Класична італійська піца
        подається на вибір з базиліком
        або без базиліка      </description>
      <calories>600</calories>
    </food>
  </pizza_menu>
```

Напишемо функцію, яка порахує суму цін і кількості калорій в меню.

```
    protected void Page_Load(object sender, EventArgs e)
    {
        int ItemsCount = 0;
        decimal DishesTotal = 0;
        UInt16 CaloriesTotal = 0;
        XmlReaderSettings settings = new XmlReaderSettings();
        settings.IgnoreWhitespace = true;
        NameTable nt = new NameTable();
        object food = nt.Add("food");
        object price = nt.Add("price");
        object calories = nt.Add("calories");
       settings.NameTable = nt;
string MenuFile = Path.Combine(Request.PhysicalApplicationPath, "menu.xml");
        using (XmlReader reader = XmlReader.Create(MenuFile, settings))
        {           while (reader.Read())
          {               if (reader.NodeType == XmlNodeType.Element &&
              food.Equals(reader.LocalName))
            {                 ItemsCount++;              }
            if (reader.NodeType == XmlNodeType.Element &&
              price.Equals(reader.LocalName))
            {  DishesTotal +=
              (UInt16)reader.ReadElementContentAsDecimal();  }
            if (reader.NodeType == XmlNodeType.Element &&
            calories.Equals(reader.LocalName))
            {  CaloriesTotal +=
              (UInt16)reader.ReadElementContentAsInt();   }
          }
        }
  Response.Write(String.Format("Ви замовили {0} страви на суму {1:C}, {2} калорій",
        ItemsCount, DishesTotal, CaloriesTotal));
    }
```

38.8. Клас XPathDocument

Клас забезпечує читання і збереження в пам’яті XML-документів для трансформацій за допомогою XSL. За документом можна переміщатися в будь-якому напрямку і отримувати довільний доступ до будь-якого елементу, використовуючи вирази XPath.

Візьмемо XML-документ "Quotes.xml" і файл трансформації XSL "Quotes.xsl". У вихідний потік сторінки буде направлений результат перетворення XML-документа.

```
<% XPathDocument doc =
    new XPathDocument(Server.MapPath("App_Data\\Quotes.xml"));
      XslCompiledTransform xsl = new XslCompiledTransform();
      xsl.Load(Server.MapPath("App_Data\\Quotes.xsl"));
          xsl.Transform(doc, null, Response.OutputStream); %>
```

Завдяки тому, що у файлі трансформації визначені табличні теги, на сторінці з'явиться таблиця з потрібною інформацією.

38.9. Елемент управління XML

Елемент управління XML надає спосіб перетворити XML-документ, використовуючи таблицю стилів XSL. Властивість DocumentSource дозволяють задати XML-файл, в якому знаходяться дані, TransformSource - файл трансформації XSLT.

У попередньому прикладі того ж результату можна досягти, якщо поставити на сторінці елемент управління XML.

```
<asp:Xml ID="Xml1" runat="server" DocumentSource=
"~/App_Code/Quotes.xml" TransformSource="~/App_Data /Quotes.xsl"> </asp:Xml>
```

38.10. XMLDataSource

Елемент – джерело даних XMLDataSource забезпечує простий спосіб підключення XML-документів як джерел даних до елементів, що відображає інформацію. Також можна задати запит XPath для того, щоб відфільтрувати дані. Як і SqlDataSource, він дозволяє редагувати, видаляти, додавати запису даних. Для цього потрібно отримати доступ до що знаходиться в ньому об'єкту XmlDataDocument за допомогою виклику методу GetXmlDocument. Після редагування документ зберігається за допомогою методу Save.

На відміну від табличних даних в СУБД, дані в XML-файлах ієрархічні, тому XMLDataSource зручно прив'язувати до ієрархічними елементам управління, наприклад Menu.

38.11. Синтаксис прив'язки до даних XML

Так як в додатках XML – дані використовуються все частіше і частіше, був введений метод прив'язки даних, отриманих з XMLDataSource.

Ці методи працюють так само, як Bind і Eval.

```
<% XPathBinder.Eval(Container.DataItem, "name"); %>
```

Як і при зв'язуванні з допомогою SQLDataSource, можна скорочено писати:

```
<%# XPath("name")%>
```

Так само, як і у DataBinder, метод Eval класу XPathBinder підтримує рядки форматування:

```
<% XPath("employees/employee/HireDate", "{0:mm dd yyyy}") %>
```

Застосуємо цей синтаксис в елементі DataList, який отримує дані з джерела даних XmlDataSource:

```
<asp:XmlDataSource ID="XmlDataSource1" runat="server"
DataFile="~/nobel.xml" XPath="//nobel/literature/writer">
</asp:XmlDataSource>
<asp:DataList ID="DataList1" DataSourceID="XmlDataSource1"
runat="server">
<ItemTemplate>
<p>
 <%# XPath("name")%>  получил премию по литературе в
<%# XPath("winningdate")%> за произведение <%# XPath("work")%></b>
</p>
</ItemTemplate>
</asp:DataList>
```

Контрольні запитання

1. Що таке XML?
2. Яка основна відмінність XML від HTML?
3. Для чого призначений елемент управління AdRotator?
4. Як задати формат відображення XML–документів?
5. З яких частин складається XSL?
6. Що таке XSD?
7. Для чого використовується клас XmlReader?
8. Для чого застосовується мова перетворень XSLT?
9. Який атрибут використовується для зв’язування шаблону з XML?
10. Для чого використовується клас XPathDocument?

Тема 39.Розробка веб-сервісів

Web-сервіси забезпечують просту, гнучку, засновану на відкритих стандартах модель для зв’язування додатків через Інтернет. Web-сервіси дозволяють пов’язувати існуючі програми незалежно від їх платформ, мов програмування, об’єктних моделей, які використовуються для їх реалізації.

Web-сервіси призначені для прямої взаємодії з іншими додатками через Інтернет і з цієї причини для них не передбачається користувача інтерфейсу. Замість користувача інтерфейсу Web-сервіси реалізують стандартні програмні інтерфейси, які називаються контрактами.

Web-сервіси:

• незалежні від мови програмування. Самі Web-сервіси можна реалізувати на будь-якій мові програмування, і програми, які звертаються до Web-сервісову, також можна реалізувати на будь-якій мові програмування;

• засновані на відкритих протоколах і стандартах Інтернету (HTTP, XML, SOAP);

• чи не залежать від платформи, на якій вони (або програми) реалізовані;

• за замовчуванням використовують stateless service architecture (архітектуру роботи зі сервісами без збереження стану), яка є більш масштабованої. У ній кожну відповідь Web-сервіси – це новий об’єкт з новим станом.

• Web-сервіси працюють в асинхронному режимі. В результаті і Web-сервіси, і додатки-клієнти можуть виконувати інші дії, поки, наприклад, готується відповідь Web-сервіси.

У найпростішому варіанті Web-сервіс можна представити як компонент, методи якого можна викликати через Web.

Для чого зазвичай використовуються Web-сервіси:

• запити аутентифікації (приклад такої відкритої в Інтернеті Web-сервіси-Microsoft Passport);

• сервіси, які надають інформацію про прогнози погоди для Web-сайтів;

• інформація з курсів валют;

• інформація про ціни на квитки (наприклад, авіа);

• ціни на акції;

• бронювання місць у готелях;

• відстеження проходження замовлення;

• передача заголовків новин і багато іншого.

Насправді через Web-сервіси можна зручно реалізувати будь-який обмін даними – захищений, заснований на відкритих стандартах, що працює без проблем через брандмауери і т.п.

Web-сервіс XML – це компонент, який реалізує певний алгоритм,що представляє певну функціональність різних програм, які отримують до неї доступ за стандартними протоколами, таким як HTTP, XML і SOAP (Simple Object Access Protocol – простий протокол доступу до об’єктів). SOAP грає роль універсального протоколу зв’язку між Web сервісами та клієнтськими додатками. SOAP – це ядро архітектури Web-сервісів, він покликаний замінити DCOM, RMI, IIOP та інші стандартні протоколи при роботі в гетерогенному середовищі.

Для обміну даними Web-сервіси XML використовують повідомлення у форматі XML, що дозволяє їм взаємодіяти з самими різними додатками. Web – сервіси XML застосовуються для інтеграції додатків, написаних на різних мовах програмування, які працюють на різних платформах. Вони придатні для розгортання та в Інтернеті, де вони полегшують доступ до ресурсів організації, і в інтрамережі для інтеграції корпоративного ПЗ.

Важливою особливістю моделі обчислень, заснованої на Web-сервісах XML, є те, що їх клієнтам не потрібно знати мову, на якому написаний Web-сервіс, досить знати адресу Web-сервісу та методи, які він підтримує.

Зрозуміти, в яких випадках доречно застосування рішень на основі Web-сервісів, допоможуть наступні приклади. Найпростіший Web-сервіс XML складається з коду, що реалізує деяку функціональність, наприклад розрахунок суми податку на прибуток. Такому Web-сервісу потрібен клієнтський додаток, що надає вхідні дані (розмір річного прибутку, витрат і вирахувань). Клієнтський додаток здатний викликати метод цього сервісу, передаючи йому необхідні дані як аргументи. Виклик разом з аргументами у форматі XML передається Web-сервісу через HTTP-канал з використанням протоколу SOAP. Відповідь повертається аналогічним чином. Інший приклад ілюструє застосування Web-сервісів XML для інтеграції додатків. За допомогою такого сервісу

можна наділити додаток для формування платіжних відомостей, написане однією мовою (скажімо, на мові COBOL), здатністю пересилати дані компоненту, написаному на іншій мові (припустимо, на Visual Basic).

Інша важлива властивість моделі обчислень, заснованої на Web-сервісах XML, у тому, що ні клієнт, ні сам сервіс не „знають”, як реалізований компонент, з яким доводиться взаємодіяти.

Загальний принцип роботи з Web-сервісом виглядає так (рис. 39.1):

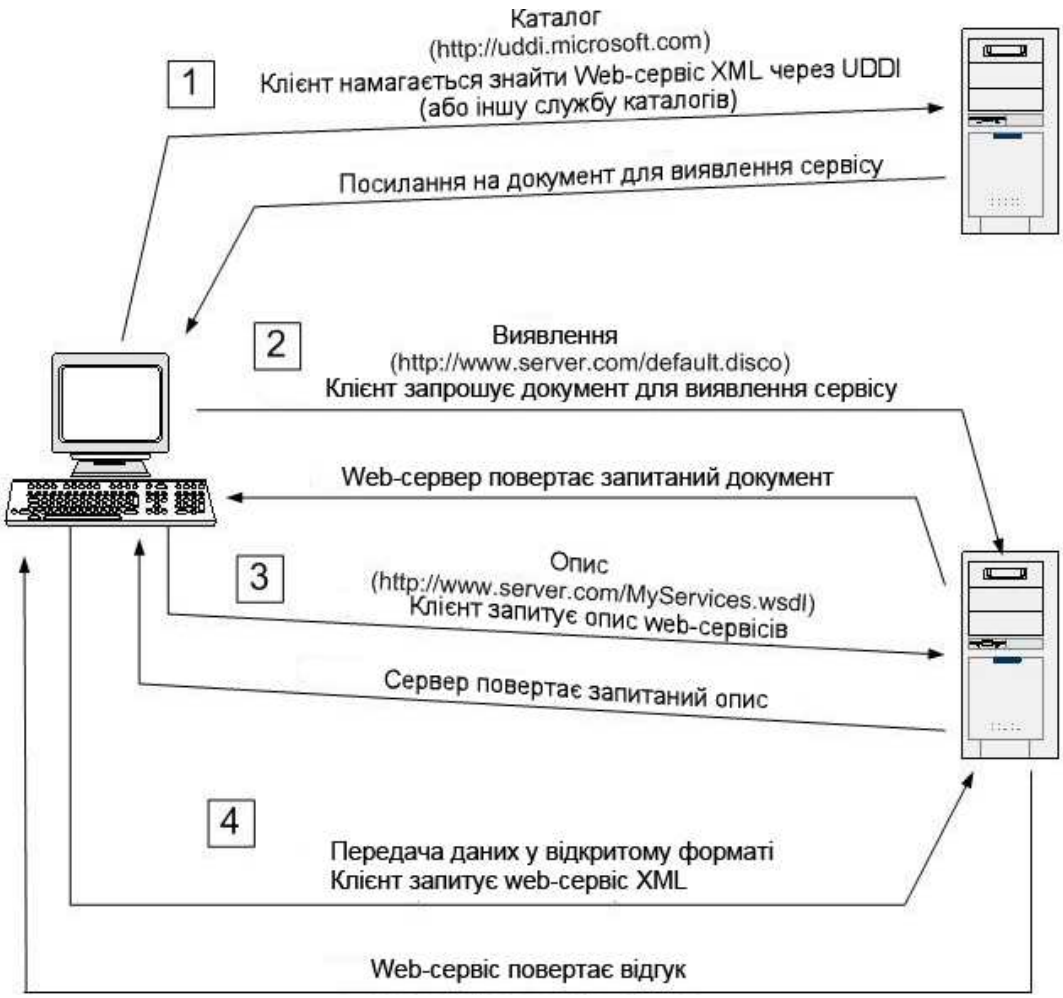

Рисунок 39.1 – Загальний принцип роботи з Web-сервісом

• розробник створює Web-сервіс;

• розробник поміщає опис Web-сервісу в стандартному форматі на спеціальному Web-сайті UDDI (Universal Description, Discovery and Integration);

• розробник клієнтського Web-додатки запрошувати Web-сайт UDDI на предмет доступних Web-сервісів. У відповідь повертається список доступних Web-сервісів, у форматі discovery files (DISCO)

• він же вибирає потрібний йому Web-сайт і через файл DISCO отримує інформацію про URL Web-сервіси та місцезнаходження документа Web Services Description Language (WSDL), в якому міститься вся необхідна інформація про взаємодію з сервісом;

• на основі отриманого документа WSDL створюється проксі-об’єкт для Web-сервіси. Цей об’єкт дозволяє звертатися до Web-сервісах як до звичайного COM-компоненту;

• проксі об’єкт прив’язується до Web-сервісу;

• потім цей проксі-об’єкт можна використовувати звичайним чином, наприклад, на Web-формі.

Детальніше про деякі моменти. UDDI – спеціальна специфікації на публікацію інформації про Web-сервісі. Інформацію про цю специфікації і списки доступних в Інтернеті Web-сервісів доступні за адресами www.uddi.org і uddi.microsoft.com.

Файли виявлення (discovery files, DISCO) – це файли у форматі XML, які містять посилання (URL) на ресурси, що забезпечують інформацію про Web-сервісову. Ці файли забезпечують можливість програмного виявлення Web-сервісів.

Файл WSDL – ще один XML-сумісний файл, який використовується програмно при створенні проксі-класу для взаємодії з Web-сервісом. Він містить інформацію про:

• шляхи URL до Web-сервісу

• методи і властивості Web-сервіси;

• використовувані типи даних;

• підтримувані протоколи взаємодії.

39.1. Взаємодія клієнтів і Web-сервісів XML

Взаємодія між клієнтом і Web-сервісом XML нагадує процес виклику видалених процедур (remote procedure call, RFC). Для виклику методів Web-сервісу XML клієнт використовує об’єкт проксі.

Взаємодія клієнта з Web-сервісом XML протікає в декілька фаз і включає наступні дії (рис. 39.2):

1. На локальному комп’ютері на базі WSDL-опису сервісу створюється об’єкт класу проксі-клієнтський заступник Web-сервісу XML. Клієнтський заступник має такий же інтерфейс, як і у Web-сервісу. і створює ілюзію, що ми працюємо з локальним, а не віддаленим об’єктом Web-сервісу XML.

2. Клієнт викликає метод об’єкта проксі.

3. Інфраструктура Web-сервісів XML на клієнтському комп’ютері серіалізуются виклик разом з аргументами і відправляє результуюче повідомлення SOAP по мережі Web-сервісу XML.

4. На сервері, де працює цей Web-сервіс, інфраструктура Web-сервісів XML десеріалізует отримане повідомлення SOAP, створює екземпляр Web-сервісу XML і ви-викликають метод з аргументами, переданими клієнтом.

5. Web-сервіс XML виконує метод і повертає інфраструктурі результат з вихідвими параметрами (якщо вони є).

6. Інфраструктура серіалізуются повернене значення з параметрами і відправляє створене повідомлення SOAP по мережі клієнтові.

7. Інфраструктура Web-сервісів XML на клієнтському комп’ютері реалізує про отримане повідомлення SOAP і передає результати об’єкту проксі.

8. Об’єкт-проксі передає клієнту возвращением методом значення і вихідні параметри.

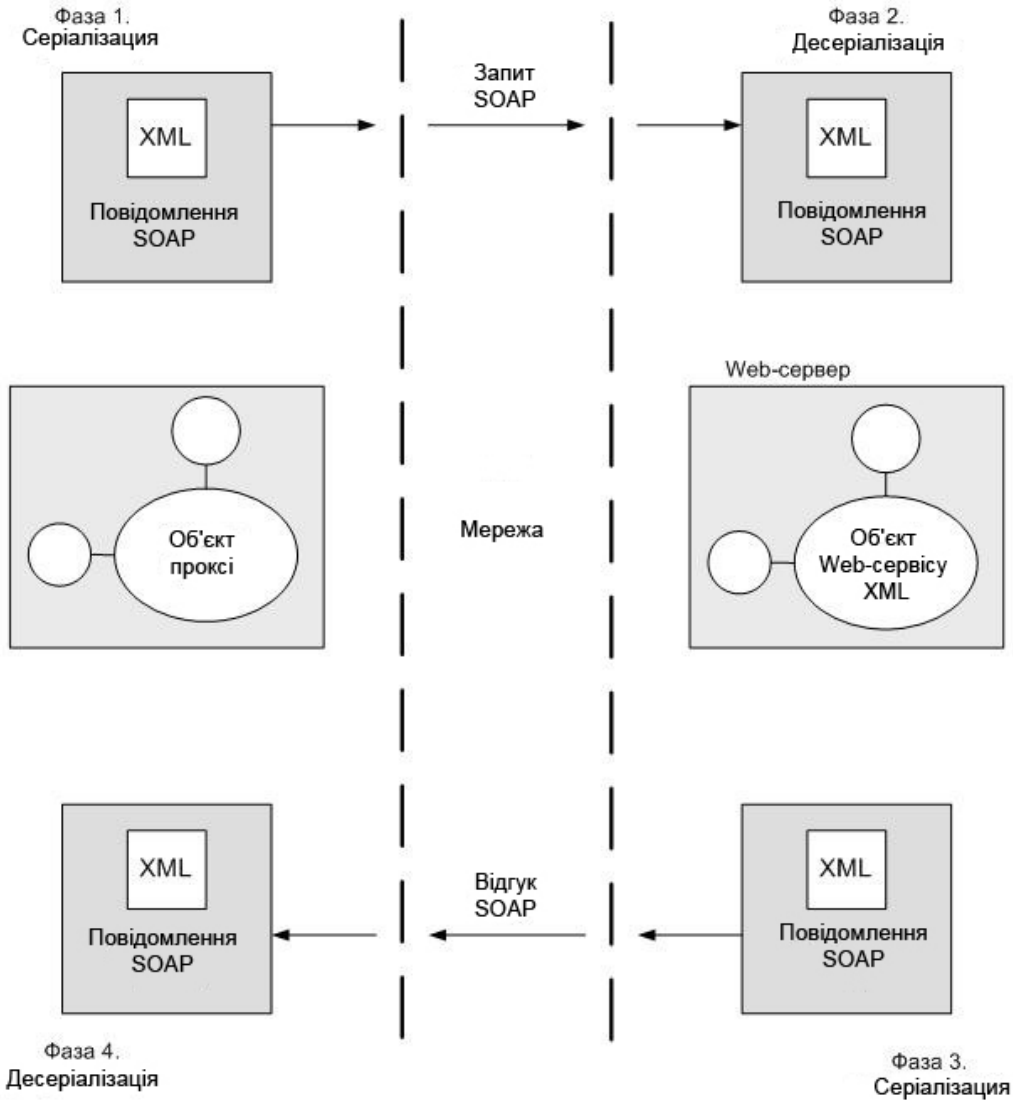

Рисунок 39.2 – Схема взаємодії клієнта і Web-сервісу XML

Загальний план роботи з Web-сервісами:

1) Створюємо файл. Asmx і визначаємо в ньому простору імен, класи, властивості і Web-методи Web-сервіси;

2) Вказуємо, які методи і властивості будуть доступні користувачам через Інтернет.

Для перевірки функціональності Web-сервіси можна звернутися до нього безпосередньо з Web-браузера. Це – не нормальний режим роботи Web-сервіси і застосовувати його слід тільки для цілей тестування.

Якщо звернутися з Web-браузера до сервісову безпосередньо, то повернеться сторінка опису Web-сервіси у форматі XML, на якій буде інформація про всі методи цієї Web-сервіси.

Якщо викликати з Web-браузера метод Web-сервіси, то повернуться повертаються цим методом дані (також у форматі XML).

Але найчастіше Web-сервіс використовують не безпосередньо, а опосередковано, через Web-форму. Для цього:

1) У проекті створюють Web-reference для Web-сервіси. У результаті створюється файл *.Vb або *. Cs з вихідним кодом для проксі-об'єкта, через який вироблятиметься робота з Web-сервісову.

2) Проект компілюється. Одночасно компілюється і проксі клас і у вигляді модуля DLL поміщається в каталог /bin.

3) В коді Web-форми створюється об'єкт WebReference.

4) Викликаються методи Web-сервіси і які повертаються ними дані використовуються на Web-формі.

Незважаючи на те, що звернення до Web-сервісів безпосередньо в підсумку в додатках не рекомендується, під час розробки воно використовується дуже часто (наприклад, для цілей тестування). Про це зараз і буде розказано.

При зверненні на Web-сервіс безпосередньо з броузера звичайно спочатку треба звернутися на початкову сторінку Web-сервіси (*.Asmx). Вам буде згенерована сторінка з списком методів і ще одним посиланням-на сторінку з формальним описом Web-сервіси-WSDL contract (призначеної для обробки програмним способом). Потрібно клацнути по посиланню з ім'ям потрібного методу. Відкриється ще одна автоматично згенерувала сторінка зі списком параметрів даного методу і полями для введення значень цих параметрів. Після того, як всі значення будуть введені, можна натиснути на кнопку Invoke. У відповідь вам буде переданий результат виконання цього методу (у форматі XML).

А тепер про програмний виклик Web-сервісів через проксі-клас.

Про створення проксі-класу програмним чином вам піклуватися особливо не потрібно. Досить з графічного інтерфейсу Solution Explorer додати Web Reference і потрібний проксі-клас буде створений автоматично. Для цього потрібно натиснути правою кнопкою миші по проекту і в контекстному меню вибрати Add->Web Reference. Відкриється вікно, в якому вам буде запропоновано ввести URL сторінки. Asmx або wsdl. Після того, як ви введете цю адресу, VS.NET автоматично виявить всі доступні Web-сервіси і запропонує вам вибрати потрібну.

Далі можна в коді Web-форми (наприклад, в подієвої процедурі для події Page_Load) просто створити необхідний об'єкт проксі-класу.

(Можливо, спочатку потрібно буде натиснути на F5, щоб проксі-клас скомпілірувався в збірку).

А потім викликаємо властивості Web-сервіси, як звичайні методи цього об'єкта і, при необхідності, передаємо їм параметри.

Для кожного методу Web-сервіси автоматично генеруються три методи проксі класу:

• метод, по імені просто співпадає з методом Web-сервіси

• метод, Begin_ім'я_метода

• метод End_ім'я_метода

Обидві останніх різновиди призначені для виклику/завершення роботи методу в асинхронному режимі.

При роботі з Web-сервісом цілком ймовірні помилки наступних видів:

• сервіс недоступний;

• відповіді від Web-сервісу доводиться чекати дуже довго;

• в Web-сервісі сталася внутрішня помилка (і замість очікуваних даних з Web-сервісу повернулося повідомлення про помилку).

Звичайно ж, потрібно налаштовувати обробник помилок для цих Web-сервісів.

39.2. Створення веб-сервісу

Створення web-сервісу деяким відрізняється від створення звичайної сторінки. Є два варіанти: можна створити окремий проект або вставити web-сервіс в існуючий проект. У першому випадку інші проекти повинні створювати web-посилання, щоб звертатися до сервісів цього проекту. Файл web-сервісу має розширення asmx. Файл web-сервісу повинен починатися з директиви WebService. Клас web-сервісу може бути нащадком класу System.Web.Services.WebService.

Якщо при оголошенні web-сервісу ви породили його від класу System.Web.Services.WebService, то ви автоматично отримуєте доступ до глобальних об’єктах докладання ASP.NET Application, Context, Session, Server і User. Якщо ж ви створювали клас web-сервісу якось інакше-нічого страшного. Ви все одно можете отримати доступ до вищепереліченим властивостям за допомогою відповідних властивостей статичного HttpContext.Current.

Методи, які сервіс надає для дзвінка за допомогою SOAP-запитів, повинні мати атрибут WebMethod. У атрибута WebMethod існує 6 властивостей, що впливають на роботу методу.

Принципово робота по створенню Web-сервісу виглядає так:

1) Створюємо в VS.NET новий проект на основі шаблону Web Service. Інтегроване середовище розробки (IDE) Visual Studio. NET 2.0 надає інструментарій для створення, розгортання, публікації і запуску Web-сервісів XML, створених із застосуванням ASP.NET. Для того, щоб створити Web-сервіс XML, виберемо в меню File IDE-середовища послідовність опцій: New-> Web Site (рис. 39.3).

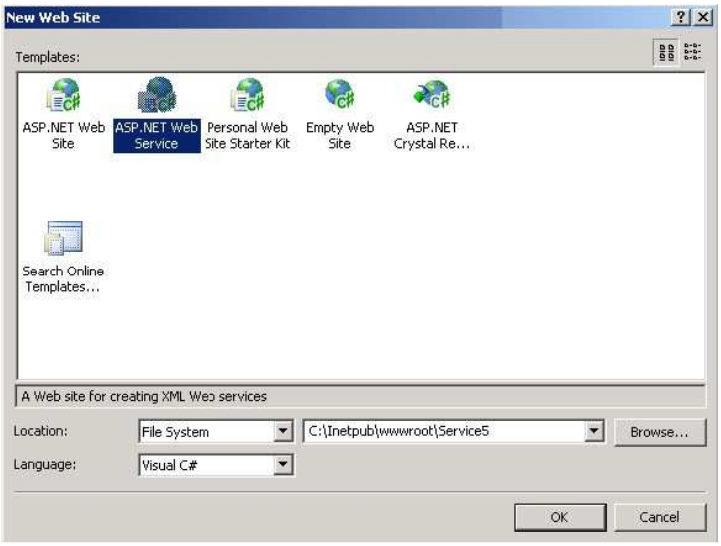

Рисунок 39.3 – Діалогове вікно New Web Site

У діалоговому вікні New Web Site (рис.39.3) на панелі Templates виберемо ASP.NET Web Service, вкажемо використовувану мову –Visual C# і визначимо місце розташування файлів проекту Web-сервісу (File System). Тут же необхідно вказати фізичну адресу проекту Web-сервісу. Після натискання на кнопку OK в каталозі Service5 будуть створені підкаталоги App_Code і App_Data і два вихідних файлу проекту: Service.asmx і Service.cs. Для того, щоб імена файлів відповідали імені проекту і, відповідно, імені Web-сервісу, необхідно перейти у вікно Solution і перейменувати імена файлів проекту. Після створення проекту Web-сервісу в проект автоматично розміщуються:

Service.asmx – складає інтерфейс Web-сервісу XML. Service.asmx містить директиву обробки Web-сервісу XML і служить його вхідною точкою.

Файл Service.asmx.vb (Servicel.asmx.cs) – прихований файл, залежний від WebService.asmx; містить клас відокремленого коду Web-сервісу XML. Визначають основну функціональність Web-сервісу XML. У ASP.NET 2.0 замість файлів Servicel.asmx.vb і Servicel.asmx.cs генеруються файли Service.vb і Service.cs.

Service.vsdisco (або файл виявлення) – XML-файл з посиланнями на URL ресурси, необхідні для виявлення Web-сервісу XML;

Web.config – містить відомості про конфігурацію Web-проекту, наприклад опис режиму налагодження і способу аутентифікації, а також визначає нестандартні повідомлення про помилки для даного проекту. У Web.config можна зберігати відомості про конфігурацію Web-сервісів XML.

2) Додаємо в нього функції, які будуть доступні через Web (одна така функція HelloWorld() вже поміщена в шаблон для прикладу. Щоб вона запрацювала, достатньо її розкоментувати. Кожну функцію, яка стане відкритим для доступу методом Web-сервісу, потрібно помітити тегом <WebMethod()>.

Інші аргументи атрибути WebMethod:

1. CacheDuration визначає проміжок часу в секундах, на який кешується web-сервіс. За замовчуванням дорівнює 0, що означає, що кешування відключено.

У цьому оголошенні значення, що повертається методом GetCustOrders, кешируєтся на 10 хвилин.

2. Description – опис методу, який виводиться на сторінці сервісу під посиланням на сторінку методу.

3. EnableSession дозволяє включити підтримку сесій. За замовчуванням підтримка сесій в web-сервісах відключена. Щоб включити її, визначте web-метод таким чином:

[WebMethod (EnableSession = true)]

4. MessageName визначає унікальне ім’я методу. Дозволяє використовувати перевантажені функції з одним ім’ям, але різними сигнатурами.

5. TransactionOption управляє підтримкою транзакцій. За замовчуванням вона відключена. Web-сервіси обмежено підтримують транзакції, тобто веб-сервіс може породжувати транзакцію, але при цьому не може бути учасником іншої транзакції. Можливі значення аргументу – Disabled, NotSupported, Supported, Required, і RequiresNew. Якщо викликається web-метод з TransactionOption, встановленим у Required або RequiresNew, а в ньому ви-

кликається інший web-метод з такими ж установками, кожен з цих методів ініціює свою транзакцію.

3) Компіляція проекту та запуск Web-сервісу ASP.NET. Оскільки до складу Web-сервісу XML входить код на мові C#, то перед використанням Web-сервісу XML виконаємо його компіляцію. Для компіляції проекту Web-сервісу необхідно в меню Build вибрати опцію Build Web Site або Rebuild Web Site. Протокол компіляції буде виданий у вікно Output. Відзначимо, що середовище Visual Studio. NET має вбудований Web-сервер (ASP.NET Development Server-сервер розробника додатків ASP.NET), що дозволяє виконувати запуск і тестування створюваного Web-сервісу без використання Web-сервера IIS. Це дозволяє виконувати налагодження Web-сервісу без його публікації на Web-сервері IIS (у розробника Web-сервісу можуть бути відсутні права на створення віртуального каталогу в цьому сервері або йому взагалі буде заборонено використовувати Web-сервер IIS для цілей налагодження). Тому при запуску виконання Web-сервісу (наприклад, після натискання клавіші Ctrl/F5 при запуску без відладчика) у вікні диспетчера задач з’явиться ярличок з адресою Web-сервісу. При використанні Web-сервера IIS (Internet Information Server) фірми Microsoft доступ до Web-сервісів зазвичай проводиться через порт з номером 80. Цей порт не вказується в URL, оскільки мається на увазі за умовчанням. Web-сервер ASP.NET Development Server використовує інші номери портів, причому для кожного Web-сервісу номер порту формується динамічно.

4) Інтерактивне тестування Web-сервісів XML. Тестування веб-сервісів можна виконувати без використання клієнтського додатку. Для цього після створення веб-сервісу необхідно або запустити його як додаток (наприклад, по Ctrl/F5), або у вікні Solution Explorer виділити вхідний файл сервісу, по правій клавіші миші відкрити контекстне меню і вибрати пункт View in Browser. У будь-якому випадку на екрані з’явиться вікно браузера Internet Explorer з переліком веб-методів, доступних з даного веб-сервісу.

Контрольні запитання

1. Що таке Web-сервіси?
2. Для чого зазвичай використовуються Web-сервіси?
3. Що таке файли виявлення?
4. Які види помилок бувають під час роботи з Web-сервісом?
5. Який атрибут WebMethod визначає проміжок часу, на який кешується Web-сервіс?
6. Який метод WebMethod дозволяє включити підтримку сесій?
7. Для чого використовується атрибут Description?
8. Який файл складає інтерфейс Web-сервісу?
9. Яка існує важлива особливість моделі обчислень, що заснована на Web-сервісах XML?
10. Яку інформацію містить в собі файл WSDL?

Тема 40. Управління станом

Поняття програми лише тоді можна застосувати до проекту, коли всі його сторінки працюють із загальною інформацією. Коли користувач реєструється на сайті, всі сторінки підлаштовуються під його налаштування. Наприклад, якщо додаток – електронний магазин, то вибрані товари поміщаються в „кошик”, яка „подорожує” разом з користувачем і дозволяє додавати в неї нові товари.

Однак відомо, що протокол HTTP спочатку не підтримує сесії. Зазвичай сервер посилає сторінку у відповідь на запит, і на цьому з’єднання обривається. Хоча HTTP1.1 підтримує режим keep-alive, сервер не в змозі визначити, що запити йдуть з одного і того ж клієнта. У ASP.NET при кожному з’єднанні на сервері створюється сесія, ідентифікатор якої зазвичай зберігається у файлі-cookie (або передається в командному рядку, якщо це неможливо).

До цих пір ми розглядали роботу ASP.NET в межах однієї сторінки. Натиснули кнопку – отримали результат. На практиці при роботі з web-додатками до мети ведуть десятки взаємопов’язаних запитів, робляться на різних сторінках. Що об’єднує сторінки додатка в одне ціле?

У ASP.NET підтримується два типи управління інформацією про стан: серверний тип і клієнтський тип. Серверний тип забезпечує максимальну захищеність, але при цьому витрачає додаткові ресурси сервера. Клієнтський тип менш ресурсномісткий з точки зору серверних ресурсів, проте його захищеність набагато нижче.

При використанні серверного типу у вашому розпорядженні:

- Application state (інформація додатки, загальна для всіх користувачів, наприклад, інформація про кількість відвідувачів Web-сайту);
- Session state (інформація конкретного користувача, доступна тільки в його сеансі. Наприклад, інформація про обраної ним колірній схемі);
- Database state server – якщо інформації стану багато, її можна зберігати в базі (на SQL Server або інший). Цей спосіб можна використовувати разом з session state або cookies .
- Об’єкт Cache – ще одна можливість працювати з інформацією стану на рівні додатку.

При використанні клієнтського типу вам доступні:

- Кукі – маленькі текстові файли, які створюються на комп’ютері користувача для зберігання інформації стану;
- Властивість ViewState – вбудована структура, яка забезпечує зберігання значень при повторному зверненні до сторінки. Фізично є приховане поле в сторінці;
- Query strings – інформація стану поступово накопичується в рядку запиту і передається при запиті кожної нової сторінки.

40.1. Колекція ViewState

Для того, щоб зберігати значення елементів управління між зверненнями до сторінці додатка можна використовувати кілька механізмів, що розглядаються нижче.

Перш за все, необхідно розглянути можливість використання стану виду (ViewState). Його доцільно використовувати в тому випадку, коли необхідно організувати зберігання даних в межах однієї сторінки. Всі елементи управління використовують стан виду за замовчуванням для збереження значень властивостей між операціями зворотного відсилання даних. Тут же можливо організувати зберігання своїх власних даних, що складаються з простих типів і спеціальних об'єктів. Приховане поле ViewState є у згенерованих ASP. NET сторінок, там зберігається інформація про стан відображення сторінки, яка передається на сервер і назад, щоб після перезавантаження на ній зберігалися її динамічні зміни. Там не зберігаються введені користувачем дані, вони відправляються, як звичайно, в тілі запиту POST.

Розуміння, що таке стан відображення і як воно працює, дуже важливо, особливо для розробників, які пишуть власні елементи управління.

У HTML було введено приховане поле <input type=hidden> саме для того, щоб передавати потрібну для програміста, але не треба чи не цікаву для користувача інформацію на сервер. ASP.NET користується цими полями, при цьому шифруючи інформацію (хоча досвідчений користувач зможе її розшифрувати). Стан відображення зберігається в прихованому полі форми з ідентифікатором __VIEWSTATE.

Яким же чином генерується це поле, що в ньому зберігається і коли воно читається при поверненні форми? Щоб це зрозуміти, треба більш детально розглянути життєвий цикл сторінки (рис. 40.1). Кожен раз, коли на сервері генерується сторінка, в перший раз або після постбека, заново створюється об'єкт Page і всі його елементи. Якщо властивість елемента керування задекларовано на сторінці або змінюється під час події ініціалізації, воно буде таким завжди.

З рисунка видно, що між подіями Init і Load при постбеку відбуваються ще дві події: завантаження стану відображення та завантаження даних постбека. І ті, й інші читаються з тіла запиту POST. Перед подією Render стан відображення знову записується за допомогою функції SaveViewState. Функція SaveViewState об'єкта рекурсивно викликає її ж вкладених в неї елементів. При цьому ця інформація кодується за алгоритмом base-64. Під час отрисовки сторінки створюється прихований елемент __VIEWSTATE. Отже , під час події Load або в обробниках подій елементів керування можна змінювати властивості сторінки або будь-яких її елементів.

Властивість ViewState є як у сторінки, так і у всіх елементів управління на ній. ViewState організовано за принципом колекції, яка в свою чергу має тип словника. Це означає, що дані зберігаються у форматі ім'я – значення. Кожен елемент, при цьому індексується за допомогою унікального строкового імені. Наступний приклад додає в колекцію ViewState елемент з ім'ям Name і привласнює йому значення Іван.

ViewState [" Name "] = "Іван" ;

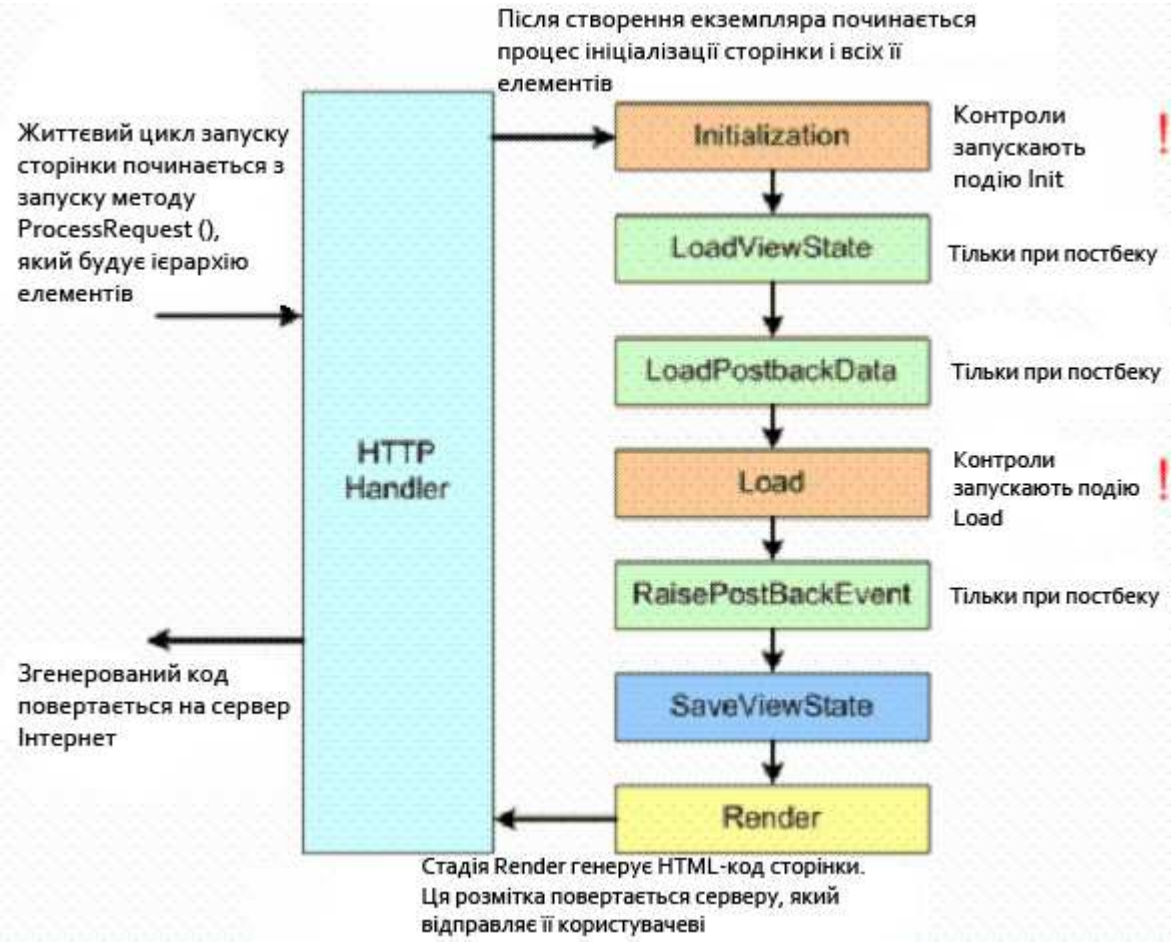

Рисунок 40.1 – Життєвий цикл сторінки

При цьому, якщо в колекції до цього не існувало елемента з ім’ям Name, то він додається, якщо ж такий елемент був, його значення замінюється новим. Для вилучення елемента з колекції необхідно використовувати ім’я елемента. Крім того, т.к. колекція ViewState дозволяє зберігати дані складаються не тільки з простих типів, а й спеціальні об’єкти (у загальному випадку будь-які об’єкти), під час витягу значення елемента, необхідно перетворити його тип до того, який буде вилучатись. Наступний приклад дозволяє витягти значення елемента Name і перетворити його в рядок:

string name ;
if (ViewState [" Name "] ! = null)
name = (string) ViewState [" Name "] ;

Перевірка на наявність елемента колекції необхідна, тому при зверненні до неіснуючого елементу колекції виникає виняток NullReferenceException.

Стан сторінки є програмісту, і в нього можна додавати додаткові дані. Наприклад, коли потрібно було сортувати DataGrid, в змінну ViewState ["sort "] записувалося вираз сортування.

ViewState являє собою колекцію пар „ключ/значення”, подібно об’єктах докладання і сесії. У нього можна поміщати тільки об’єкти з атрибутом Serializable. Властивість ViewState має тип System.Web.UI.StateBag, і синтаксис доступу до даних схожий на доступ до значень Hashtable.

Поле __VIEWSTATE може стати досить великим (порядку десятків кілобайт), наприклад, коли в формі присутній DataGrid або GridView. Ці елементи управління зберігають у стані відображення весь свій вміст. І вся ця інформація подорожує з клієнта на сервер і назад. Збільшує це поле і елемент Wizard, і TreeView. Загалом, чим більше можливостей і властивостей, тим дорожче доводиться за це платити. Тому розглянемо методи скорочення розміру поля __VIEWSTATE. У кожного елемента управління є властивість EnableViewState, яке за замовчуванням дорівнює True. Якщо встановити його в False, то стан даного елемента керування не буде записано в об'єкт ViewState. Це властивість можна відключити і у всієї сторінки – за допомогою атрибуту EnableViewState директиви Page або програмно.

Хоча відключення EnableViewState дозволяє зменшити кількість переданої інформації, не завжди це можна робити безкарно. Якщо на сторінці є DataGrid, який заповнюється під час події Page_Load за умови if (!Page.IsPostBack), тобто при першому завантаженні сторінки, то після постбека він просто зникне зі сторінки. Отже, при вимкненому EnableViewState кожного разу необхідно читати дані з джерела заново. Що вибрати, залежить від ситуації. В одних випадках буває важливіше заощадити на часі завантаження сторінки, в інших – на з'єднанні з базою даних.

Як вже говорилося, стан виду дозволяє зберігати крім простих типів об'єкти. Для того, щоб елемент міг бути збережений в стан виду, він повинен бути спершу перетворений в послідовність байтів. Такий процес називається серіалізацією. При створенні нового класу, за замовчуванням, його об'єкти не піддаються серіалізації. При спробі помістити їх у стан виду, з'являється повідомлення про помилку. Для того, щоб створити об'єкт, що піддається серіалізації, необхідно перед оголошенням класу даного об'єкта розмістити атрибут Serializable.

Використання стан виду є хорошим варіантом збереження значень змінних і полів форм між викликами, тому що передбачає використання ресурсів пам'яті сервера. Тим не менше, існують декілька негативних сторін використання ViewState.

Дані ViewState зберігаються в прихованих полях сторінки і передаються на сторону клієнта. У цьому можна переконатися переглянувши вихідних код такої сторінки. Звичайно, ці дані не зберігаються у відкритому вигляді, вони представлені у форматі Base64, проте їх легко можна перетворити в масив байт, що представляють символи ASCII. Для цього досить використовувати функцію FromBase64String.

В результаті цих дій в рядку decodedString міститиметься рядок, отримана в результаті перетворення Base64 рядки в набір символів ASCII, які можуть бути виведені на екран і легко прочитані.

Все це змушує зробити два важливих висновки. По-перше, кількість інформації, що зберігається в стан виду не повинно бути великим, тому що це призводить до збільшення обсягу переданих даних від сервера до клієнта і навпаки. У разі необхідності збереження великої кількості даних краще скористатися засобами бази даних, або використати стан сеансу. По-друге, в ViewState не можна зберігати критичні дані (дані, доступ користувача до яких необхідно

заборонити), тому що вони легко можуть бути декодовані і прочитані, більше того, досвідчений користувач зможе змінити ці дані при здійсненні запиту на зворотний відсилання. У цьому випадку краще скористатися станом сеансу.

ViewState не дозволяє зберігати інформацію, яка використовуватиметься кількома сторінками. У цьому випадку краще скористатися станом сеансу, наборами cookie, або рядком запиту.

40.2. Об’єкт Session

Інформація session state зберігається за допомогою об’єкта HttpSessionState. Цей об’єкт автоматично створюється для кожного активного сеансу Web-додатку. По суті це – такий же набір ключів/значень, однак у порівнянні з HttpApplicationState для цього об’єкта передбачено безліч специфічних властивостей, за допомогою яких можна отримати інформацію про сеанс або налаштувати його властивості. При бажанні інформацію HttpSessionState можна зберігати в базі даних або на state server.

Набір комбінацій ключ/сеанс, який використовується в цих об’єктах, називається hashtable (структура, дуже схожа на словник).

Кожен активний сеанс Web-додатку ідентифікується і відстежується за унікальним 120-бітним Session ID, який складається виключно з символів ASCII, дозволених в URL. Далі Session ID використовується в куки або прописується прямо в рядок запиту URI (такий підхід називається Cookieless Session ID) і по ньому і ідентифікується сеанс користувача.

Сеансом є період часу, коли користувач знаходиться на сайті. Він починається, коли відвідувач вперше заходить на сайт. Користувач може закрити браузер, і сервер не буде знати про це. Тому в сесії існує таймаут, який за замовчуванням дорівнює 20 хвилинам. Якщо протягом цього часу користувач не здійснював активних дій на сайті, сесія вважається закритою. Для додатків, де втрата даних критична (наприклад, фінансових), зазвичай таймаут зменшують. Об’єкт Session призначений для реалізації механізму стану сеансу, використовуваного для зберігання будь-якого типу даних користувача, які необхідно зберігати між запитами Web-сторінок. Користувача дані при цьому зберігаються у форматі ім’я-значення. Такий механізм, зокрема, можна використовувати при створенні інтернет-магазину, у якому покупець перед покупкою складає товари у віртуальну корзину, яка після завершення сеансу (переходу на іншу сторінку, або завершення роботи з браузером) повинна бути вилучена. Для збереження даних про обрані товари можна скористатися об’єктом Session.

При підключенні користувача до додатка, створюється окремий сеанс і окрема колекція даних. Такі можливості мають і свою ціну – дані , що зберігаються в Session зберігаються в оперативній пам’яті сервера. Навіть при невеликому обсязі цих даних, їх використання може загрожувати продуктивності додатка в тому випадку, якщо до сайту почнуть отримувати сотні і тисячі клієнтів.

Робота зі станом сеансу практично аналогічно роботі із станом виду, за винятком того, що замість ключового слова ViewState використовується Session.

Серверна робота зі станом вимагає обов’язкової передачі Session ID за допомогою куки на клієнтському комп’ютері. Для тривалого зберігання ін-

формації сеансу (наприклад, даних, які користувач ввів при реєстрації) найзручніше цю інформацію зберігати на джерелі.

Приклад :

Session [" user "] = user ;

Для відновлення збереженого об’єкта user необхідно скористатися наступним кодом.

user = (User) Session [" user "] ;

Стан сеансу знищується в наступних випадках:

1) якщо користувач закриває браузер;

2) після закінчення 20 хвилин з моменту останньої активності користувача;

3) при явному завершенні сеансу з програмного коду за допомогою виклику методу Session.Abandon();

У сесії можуть зберігатися змінні будь-яких типів, у тому числі призначені для користувача. Запит до даних сесії повертає тип object, тому після отримання змінної необхідно привести його до потрібного типу:

Username = Session [" Username"].ToString();

40.3. Об’єкт Application

Робота з application state забезпечується класом HttpApplicationState, який автоматично створюється для кожного активного додатка ASP.NET. Сам по собі цей об’єкт – всього лише колекція пар ім’я ключа/значення. У нього безліч властивостей і методів, які чомусь не підказуються (можна отримати інформацію тільки через документацію). Фактично це – глобальна структура зберігання, яка доступна з усіх сторінок Web-додатку. У цю структуру можна додавати значення ключ/значення на рівні всього програми, а потім витягувати їх. Зазвичай використовується для зберігання даних, загальних для всіх сеансів, які змінюються не часто.

Об’єкт Application багато в чому аналогічний Session. Різниця полягає тільки в тому, що дані, що зберігаються в ньому глобальні для всього програми. Це так званий стан додатка, доступ до даних якого може отримати будь-який клієнт.

Стан додатки схоже на стан сеансу, тому що зберігає інформацію на сервері, дозволяє зберігати об’єкти такого ж типу і використовує формат ім’я-значення для зберігання даних.

За допомогою стану програми можна створити лічильник, за допомогою якого відстежувати інформацію про кількість раз, яке ту чи іншу дію виконувалося всіма клієнтами даного додатка.

Час життя елементів стану програми ніколи не закінчується, вони існують до тих пір , поки додаток або сервер не будуть перезапущени.

У сучасній практиці стан додатка застосовується вкрай рідко, тому що не є ефективним.

40.4. Cookie

У більшості Web-додатків для роботи із станом використовуються куки. Кукі – це невеликий набір даних, який зберігається або в текстовому файлі на клієнтському комп’ютері, або просто в оперативній пам’яті там же. Кукі вико-

ристовуються для зберігання інформації про конкретного клієнта, сеанса або додатка. При зверненні клієнта на Web-сервер браузер посилає інформацію куки разом із запитом. Для кожного кукі визначається домен, до якого цей куки призначений. Для одного домену може бути використано одночасно кілька кукі.

Кукі бувають двох видів:

- Тимчасові – живуть тільки до закриття вікна браузера;

- Постійні – зберігаються на жорсткому диску комп'ютера і можуть жити місяцями і роками (як їм призначено). Internet Explorer зберігає їх у вигляді файлів з іменами username@domainname.txt.

Користувач завжди може почистити кукі на своєму комп'ютері, так що таку можливість слід мати на увазі.

Кукі – не дуже безпечне рішення (особливо в порівнянні з застосуванням HttpSessionState), користувач легко може змінити значення кукі на своєму комп'ютері.

У кожному кукі може зберігатися не більше 4 Кбайт інформації. Кукі створюються за допомогою властивості Cookies об'єкта Response і класу Request. Властивість Cookies представляє колекцію Cookies (екземпляр класу HttpCookieCollection).

40.5. Файл Global.asax , об'єкти докладання і сеансу

Ще одним глобальним поняттям є обробка подій рівня програми. Це події на кшталт: „на одній з сторінок додатку почалася обробка запиту" або „на якійсь сторінці сталася помилка". Такі події обробляються в коді файлу global.asax.

Додайте в проект новий файл типу Global Application Class. Це можна зробити через меню Website, або клацнувши правою кнопкою миші на назві проекту в Visual Studio, або через меню File -> New в WebMatrix (рис. 40.2).

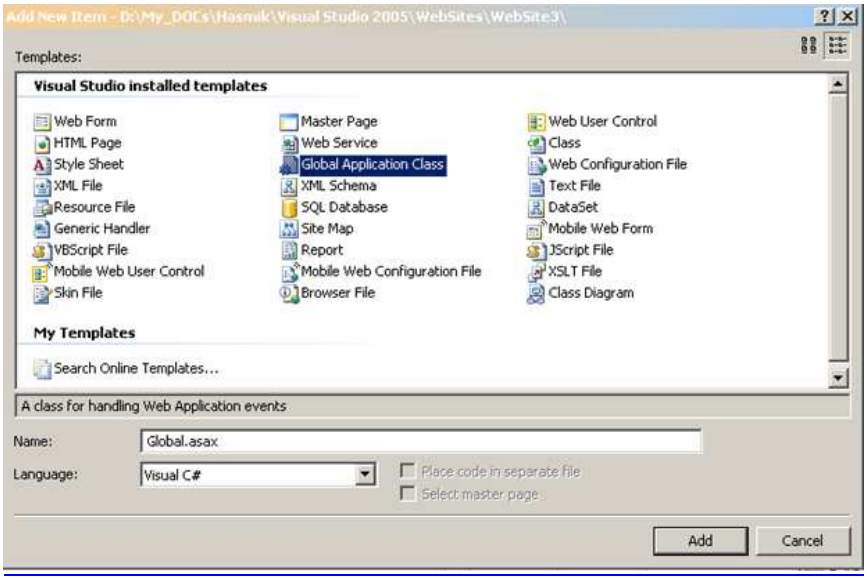

Рисунок 40.2 – Форма додавання файлу global.asax до веб-додатку

Файл Global.asax – це текстовий файл, який зберігається в кореневій теці програми ASP. NET. У проекті може бути тільки один Global.asax. Він

містить об’єкти, події, змінні рівня програми. При створенні файлу в ньому знаходяться функції, які повинні виконуватися при настанні будь-якого з п’яти подій, описаних в таблиці 40.1.

Таблиця 40.1. Події для файлу Global.asax

Події	Умови наступу
Application_Start	Перша сторінка додатку відкривається будь-яким користувачем
Application_End	Робота додатку завершується
Session_Start	Відвідувач активізує додаток
Session_End	Користувач залишає додаток або не запитує сторінку на протязі деякого періоду часу
Application_Error	Під час виконання додатку з’являється необроблена помилка

Дані події призначені для виконання коду або створення змінних, які існують доти, поки існує програма або поки відвідувач перебуває на вузлі. За рахунок цього можна зберігати реєстраційну інформацію, незмінні дані для web-вузла або постійні з’єднання з джерелами даних.

40.6. Використання рядка запиту

Однією з часто виникаючих проблем при розробці Web-додатків є передача інформації від однієї сторінки до іншої і від одного додатка до іншого додатку. Існує кілька способів вирішення цієї проблеми, одним з яких є використання рядка запиту, при якому дані передаються в URL адресі.

Цей підхід дуже часто застосовується в пошукових системах. Наприклад, таким чином виглядають URL найбільш популярний пошукових систем при пошуку рядка „Web design”.

Пошукова система Yandex

http://www.yandex.ru/yandsearch?text=Web+design

Пошукова система Rambler

http://www.rambler.ru/srch?words=Web+design

Пошукова система Google

http://www.google.ru/search?hl=ru&newwindow=1&q=Web+design&lr =

З представлених прикладів видно, що ключові слова, складові запит вказані в рядку URL після знака питання. Саме ця частина URL називається рядком запиту. Таким чином можливо організувати передачу значень параметрів прямо в рядку запиту. Перевага такого підходу полягає в тому, що рядок запиту проста за своєю структурою і не викликає навантаження на сервер, за допомогою такого механізму можна легко переносити інформацію з однієї сторінки на іншу. Недолік полягає в тому, що за допомогою рядка запиту можливо передавати тільки інформацію, представлену у вигляді простих рядків, що містять символи, які допускається використовувати в URL адресі.

Для передачі інформації в рядок запиту, його необхідно помістити в URL-адресу сторінки, до якої повинен відбутися перехід. Це можливо зробити використовуючи елемент управління HyperLink, або скористатися оператором Response.Redirect( ).

Наприклад, для того, щоб перейти на сторінку login.aspx і передати в рядку запиту змінну username необхідно виконати наступний код.

String FirstName = "Іван";

String LastName = "Іванов";

Response.Redirect ("login.aspx?Username="+FirstName + "" + LastName);

Для передачі декількох параметрів в рядку запиту, параметри необхідно розділяти знаком амперсанд – "&". З урахуванням цього, попередній приклад можна переробити так, щоб ім'я та прізвище користувача передавалися окремо. Для цього змінимо рядок Response.Redirect таким чином:

Response.Redirect ("login.aspx ? Firstname ="+F_Name + "&lastname ="+ L_Name);

Для вилучення рядка запиту, необхідно використовувати метод QueryString об'єкта Request. Для вилучення значень параметрів, переданий в попередній прикладах, необхідно використовувати наступний код.

string FN = Request.QueryString ["firstname"];

string LN = Request.QueryString ["lastname"];

При використанні рядка запиту слід пам'ятати, що вона передається завжди у відкритому вигляді, тому є дуже вразливою. Для захисту інформації, переданої в рядку запиту можливо використовувати шифрування.

Контрольні запитання

1. Які типи управління інформацією підтримуються у ASP.NET?
2. Що таке HashTable?
3. Для чого призначений об'єкт Session?
4. В яких випадках знищується стан сеансу?
5. Яка різниця між Application і Session?
6. Що таке кукі?
7. Які види кукі існують?
8. Що містить в собі файл Global.asax?
9. Що таке рядок запиту?
10. Для чого потрібен елемент управління HyperLink?

Тема 41. Аутентифікація користувачів. Локалізація додатку

Створення багатомовних web-сайтів має особливо велике значення в неангломовних країнах. Спочатку ASP.NET була налаштована на англійську мову, причому на його американський різновид. Причини цього очевидні. Але платформа .NET підтримує концепцію інформації про культуру, а рядки зберігаються у форматі Unicode, що дозволяє писати їх на багатьох мовах. Глобалізація – це створення додатків, здатних працювати в різних культурних середовищах. Локалізація – створення ресурсів для роботи з конкретною культурою. Ресурси мають бути відокремлені від програмного коду.

Класи для роботи з інформацією про культури укладені в просторі імен Globalization. Клас CultureInfo містить властивість CurrentCulture, яка дозволяє дізнатися всі дані про поточну культуру – формати відображення, календар, кодову сторінку і інші.

Файли ресурсу містять рядки, які можуть бути написані на різних мовах для різних культурних середовищ. Формат цих файлів – XML, слідуючи

спеціальній схемі Microsoft ResX. Файли .resx автоматично включаються в збірку для використання на сторінках. Крім рядків, файли ресурсу можуть містити зображення та інші файли. Їх можна використовувати для створення багатомовних додатків. На відміну від попередніх версій, ресурси не потрібно компілювати вручну в збірку-сателіт – ASP. NET робить це сама.

У папці App_GlobalResources зберігаються файли ресурсів, назви яких відповідають культурній схемі. Наприклад, ресурс для російської культури називається Resource.ru–ru.resx, для фінської – Resource.fi–FI.resx. Ресурс з нейтральною культурою називається просто Resource.resx. Ресурси доступні для всіх сторінок і користувальницьких елементів управління. Проміжне розширення слід стандарту. NET на регіональні стандарти – складається з головного і допоміжного тегів.

У папці \App_LocalResources зберігаються локальні файли ресурсів для конкретних сторінок. Назва файлу ресурсу формується з імені сторінки, коду культурного середовища та розширення .resx. Наприклад, default.aspx.de.resx – це файл ресурсу німецькою мовою для сторінки default.aspx.

Протокол HTTP дозволяє браузерам посилати список бажаних мов на сервер. У браузері можна налаштувати переважні мови web-сторінок. ASP.NET дозволяє автоматично модифікувати культуру сторінки в залежності від першої мови у списку. Для цього атрибуту Culture директиви Page потрібно привласнити значення auto. Так само йде справа у атрибута UICulture – він визначає культуру користувача інтерфейсу. Менеджер ресурсів шукає рядки та інші ресурси у файлі з тим розширенням, яке визначено в атрибуті UICulture. Формат відображення дат, чисел і грошової інформації визначається атрибутом Culture.

Якщо помістити на сторінку елемент управління Calendar, він буде в тому форматі, який відповідає культурній інформації, пов’язаною з даною мовою. Числа і грошові одиниці теж будуть виводитися в форматі цієї культури.

Властивості Culture і UICulture сторінки – рядкові, і приймають значення в довгому форматі, наприклад, "English (United States)":

<%@ Page Language="C#"Culture="Auto" UICulture="Auto" %>

Якщо перша мова у списку відповідає культурі, яку підтримує ASP.NET, на самому початку життєвого циклу сторінки ця культура стає поточною. Якщо є файли ресурсу на цій мові, за допомогою класу Resource можна отримати доступ до рядків файлу ресурсів з відповідним розширенням, інакше ResourceManager читатиме з файлів ресурсів за замовчуванням.

Завантажити рядок з файлу ресурсів можна по-різному. Перший спосіб – використовувати клас Resource. Попередньо в App_Global Resources потрібно створити файли Resource.resx (табл. 41.1) і Resource.ru–RU.resx з рядками (табл. 41.2).

Таблиця 41.1. Resource.resx

Назва	Значення
Answer	Good morning,
PageTitle	Sample Globalization Page
Question	What is your name?

Таблиця 41.2. Resource.ru–RU.resx:

Назва	Значення
Answer	Привіт
PageTitle	Приклад глобалізації ASP.NET
Question	Як Вас звати?

```
<%@ Page Language="C#"Culture="Auto" UICulture="RU–ru" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server">
protected void Page_Load(object sender, System.EventArgs e)
   {       Page.Title = Resources.Resource.PageTitle;
   }
protected void Button1_Click(object sender, System.EventArgs  e)
   {  Localize1.Text = Resources.Resource.Answer + ", " + Textbox1.Text;
   }
</script>
<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
<title></title>
</head>
<body>
   <form id="Form1" runat="server">
      <p><%= Resources.Resource.Question %></p><br />
     <asp:TextBox ID="Textbox1" Runat="server"></asp:TextBox><br />
<asp:Button ID="Button1" Runat="server" Text="Submit" OnClick="Button1_Click" />
      <p>
        <asp:Localize ID="Localize1" runat="server"></asp:Localize>
        </p>
   </form>
</body>
</html>
```

Другий спосіб дозволяє статично виводити інформацію з локальних файлів ресурсів:

```
<%@ Page Language="C#" AutoEventWireup="true"  Culture="auto:en-
US" meta:resourcekey="PageResource1" UICulture="auto:en-US" %>
<asp:Button  ID="ConvertButton"  runat="server"  Text="Convert"
OnClick="ConvertButton_Click" meta:resourcekey="ButtonResource1"/>
```

У LocalizeImp_cs.aspx.resx міститься рядок ButtonResource1.Text зі значенням конвертувати, в LocalizeImp_cs.aspx.de.resx зі значенням Konvertieren, в файлах ресурсів для інших мов – переклад слова на ці мови.

41.1. Аутентифікація і авторизація

Важлива частина багатьох web-додатків – можливість контролювати доступ до ресурсів. Безпека проекту обертається навколо двох концепцій –

аутентифікації та авторизації. Аутентифікація – процес визначення особи користувача. Авторизація – процес визначення прав користувача на доступ до певних ресурсів, на перегляд деяких розділів, на можливість редагування інформації і так далі.

Для авторизації користувач вводить ім’я користувача, під яким зареєстрований(а), і пароль. Після цього додаток визначає можливість доступу до ресурсів, а також може видозмінювати як зовнішній вигляд за допомогою тем, так і зміст сторінок, що генеруються. Наприклад, у форумі записи можуть показуватися у вигляді дерева або лінійно.

У ASP. NET аутентифікацією керують за допомогою служби Membership, яка дозволяє визначити різні види членства на сайті. Інформацію про членів можна зберігати в різних місцях – в базах даних, текстових файлах або навіть в облікових записах Windows. Конфігурувати членство можна індивідуально для кожного користувача або на основі ролей за допомогою сервісу Role Manager. Ролі полегшують конфігурування, так як можна створювати ролі і потім додавати користувачів до готових ролей. Будь-якому користувачеві може належати будь-яка кількість ролей.

За замовчуванням служби використовують провайдера AspNetSqlProvider. У такому випадку ASP. NET автоматично створює базу даних ASPNETDB.MDF в директорії проекту App_Data, коли запускається команда ASP.NET Configuration – або програмно, або за допомогою елементів управління групи Login задаються служби Membership або Role Manager. Служба Membership також забезпечена провайдером, що працюють з Active Directory. Role Manager може працювати з провайдером, пов’язаним з Authorization Manager операційної системи Widnows 2003. API, пов’язане зі службою, дозволяє налаштувати її і на використання користувальницького провайдера.

Ролі мають певні права. Наприклад, адміністратор сайту може змінювати налаштування, редагувати членство інших користувачів. Звичайний користувач з роллю Friend може переглянути приватні альбоми адміністратора, а неавторизований користувач – тільки публічні. Ця модель реалізована в додатку Starter Kit. У цьому додатку ролі створюються під час першого запуску:

```
void Application_Start(object sender, EventArgs e)
 {
SiteMap.SiteMapResolve += new SiteMapResolveEventHandler(AppendQueryString);
   if (!Roles.RoleExists("Administrators"))
Roles.CreateRole("Administrators");
   if (!Roles.RoleExists("Friends"))
Roles.CreateRole("Friends");
}
```

З цього прикладу видно, що доступ до ролей можна отримати за допомогою класу Roles. Це не глобальна змінна, а статичний клас, доступний з будь-якого файлу проекту. Створення ролі виражається в тому, що в базі ASPNETDB.MDF в таблиці Roles з’являється новий запис.

Створити користувачів і призначити їм ролі можна у вбудованому додатку ASP.NET Configuration. Інформація про користувачів зберігається в

таблицях aspnet_Users, aspnet_UsersInroles, aspnet_Membership. Паролі зберігаються в зашифрованому за допомогою хеш-функції вигляді. Навіть адміністратор не може його підглянути. Пароль, який вводиться в момент аутентифікації, теж хешируется і порівнюється зі значенням в базі. Зберігання в незашифрованому вигляді – це загроза безпеці.

Будь-який користувач може зареєструватися, але при цьому не отримає ролі і у нього не буде нових прав, поки адміністратор не призначить йому ролі.

У цьому додатку можна також створювати правила доступу.

Налаштування конфігурації служб Membership і Role Manager, звичайно ж, записуються у файл Web.config:

<roleManager enabled="true" />

Таким чином включається служба Role Manager.

Елемент authentication mode визначає спосіб аутентифікації. Якщо це Forms, то свої ім'я та пароль користувач вводить у формі. У локальній мережі (інтернет) можна аутентифікувати користувачів по їх обліковому запису, тоді його значення ставиться як Windows.

За допомогою елемента authentication запускається служба Membership:

```
<authentication mode="Forms">
    <forms loginUrl ="Login.aspx"/>
  </authentication>
```

Інші доступні значення – Passport і None. При значенні Passport користувачі авторизуються за допомогою паспорта від Microsoft.

При аутентифікації за допомогою форм користувачі отримують доступ до сторінок в залежності від даних, введених на формі. До входу на сайт користувач вважається анонімним і використовується анонімна аутентифікація. Після того, як він пройшов аутентифікацію, користувач отримує файл-cookie, який використовується для подальших запитів.

Всього у елемента forms 8 можливих атрибутів, які розглянуті в табл. 41.3.

Таблиця 41.3. Атрибути елемента Forms

name	Визначає ім'я файлу-cookie, який надсилається користувачам після аутентифікації. За замовчуванням він називаєть- ся. ASPXAUTH
loginUrl:	URL сторінки, з якої можна увійти в систему. За умовчанням це Login.aspx
protection:	Рівень захисту файла-cookie на користувальницькій машині. Можливі значення – All, None, Encryption, і Validation
timeout:	Час, після закінчення якого cookie застаріває. Значення за замовчуванням – 30 хвилин
path:	Шлях до файлів cookie
requireSSL:	Чи потрібно шифрувати дані по протоколу SSL
slidingExpiration	Якщо True, то cookie застаріває через період часу timeout після останнього запиту. Інакше – через період часу timeout після створення
cookieless:	Місце зберігання ідентифікатора користувача. Значення за замовчуванням useDeviceProfile

За замовчуванням всім користувачам доступні всі сторінки програми. Якщо потрібно обмежити доступ користувачів до якої-небудь сторінці, в Web.config вводиться запис:

```
<location path="Admin.aspx">
  <system.web>
   <authorization>
    <allow roles="Admin" />
    <deny users="*" />
   </authorization>
  </system.web>
 </location>
```

Значення "?" позначає анонімного користувача, а "*" – всіх користувачів:

```
<allow users="?" />
```

В елементах allow і deny користувачі і ролі можуть бути задані перерахуванням:

```
<allow users="Alex, Dave" />
```

Елемент location визначає частину сайту, доступ до якої потрібно обмежити. В даному випадку це одна сторінка Admin.aspx. Перша частина авторизації дозволяє доступ до неї користувачам в ролі адміністратора Admin. Друга забороняє доступ всім іншим користувачам.

Якщо елемент system.web знаходиться в кореневому вузлі файлу, то вкладений у нього вузол authorization визначає налаштування доступу до всього сайту.

Коли неавторизований користувач намагається отримати доступ до сторінки, відкривається форма, визначена в атрибуті forms loginUrl. Якщо він введе ім’я і пароль користувача, який має доступ до сторінки, то вона відкриється, інакше знову йому або їй буде показана форма логіна.

Часто буває потрібно конфігурувати з точки зору безпеки систему навігації. Це означає, що меню не має показувати ті сторінки, для перегляду яких користувач не авторизований:

```
<siteMap defaultProvider="AspXmlSiteMapProvider" enabled="true">
    <providers>
     <clear/>
     <add name="AspXmlSiteMapProvider" type="System.Web.XmlSiteMapProvider, System.Web, Version=2.0.3600.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" siteMapFile="web.sitemap" securityTrimmingEnabled="true"/>
    </providers>
  </siteMap>
```

Тут атрибут securityTrimmingEnabled, встановлений в true, змушує провайдера карти сайту фільтрувати вузли залежно від ролі користувача. Це означає, що в елементах навігації будуть видні тільки доступні сторінки.

Член User сторінки дозволяє отримати доступ до всієї інформації, що стосується поточного користувача.

Метод IsInRole визначає, чи належить користувач до ролі:

```
if (User.IsInRole("Admin"))
    Page.Title = "Hello, Administrator!";
```

Властивість Identity дає інформацію про аутентифікації користувача:

```
if (User.Identity.IsAuthenticated)
//виконати код, якщо користувач легальний
```

User.Identity.AuthenticationType показує спосіб аутентифікації, встановлений в Web.config.

Можна обмежити доступ не тільки до сторінок, але і до частин сторінок.

41.2. Елементи управління групи Login

Ми вже створювали форми реєстрації. Це було тільки вправа, так як елементи управління цієї групи можуть брати на себе реєстрацію та авторизацію користувачів, відновлення пароля та інші функції, взаємодіючи при цьому з системою Membership і Roles. Група Login знаходиться в панелі інструментів.

Елемент управління LoginName дозволяє показати ім'я користувача:

```
Заходьте ще, <asp:LoginName ID="FCLoginName" Runat="server" />
```

Текст, що відображається можна форматувати за допомогою FormatString:

```
<asp:LoginName ID="HelloLoginName" Runat="server" FormatString=
"Hello, {0}" />.</p>
```

LoginStatus – це розумний елемент управління. Якщо користувач не аутентифікований, відображається гіперпосилання Login на сторінку входу на сайт. Якщо користувач авторизований, це ImageButton з командою Logout. Якщо її натиснути, то користувач виходить з системи. Текст посилань можна міняти у властивостях LoginText і LogoutText або використовувати зображення:

```
<asp:LoginStatus ID="LoginStatus1" runat="server"
LoginText="Вхід" LogoutText="Вихід" />
  <asp:LoginStatus ID="LoginStatus2" runat="server"
LogoutAction="Redirect" LogoutPageUrl="~/Default.aspx"
LoginImageUrl="~/Images/arrow_next.gif"
LogoutImageUrl="Images/arrow_prev.gif" />
```

Натискання на посилання Logout в цьому випадку перенаправляє користувача на сторінку Default.aspx.

Елемент управління LoginView складається з двох шаблонів: AnonymousTemplate і LoggedInTemplate. Який з них використовується для відображення, залежить від того, переглядає сторінку анонімний користувач або авторизований. На сторінці default.aspx Starter Kit це єдиний елемент на сторінці Registration.aspx:

```
<asp:loginview id="LoginArea" runat="server">
  <AnonymousTemplate>
    <asp:login id="Login1" runat="server">
      <layouttemplate>
        <div class="login">
          <h4>Login to Site</h4>
```

```
<asp:label runat="server" id="UserNameLabel" CssClass="label" associatedcontrolid="UserName">User Name</asp:label>
<asp:textbox runat="server" id="UserName" cssclass="textbox" accesskey="u" />
<asp:requiredfieldvalidator runat="server" id="UserNameRequired" controltovalidate="UserName" validationgroup="Login1" errormessage="User Name is required." tooltip="User Name is required." >* </asp:requiredfieldvalidator>
<asp:label runat="server" id="PasswordLabel" CssClass="label" associatedcontrolid="Password">Password</asp:label>
<asp:textbox runat="server" id="Password" textmode="Password" cssclass="textbox" accesskey="p" />
<asp:requiredfieldvalidator runat="server" id="PasswordRequired" controltovalidate="Password" validationgroup="Login1" tooltip="Password is required." >*</asp:requiredfieldvalidator>
<div><asp:checkbox runat="server" id="RememberMe" text="Remember me next time"/></div>
<asp:imagebutton runat="server" id="LoginButton" CommandName="Login" AlternateText="login" skinid="login" CssClass="button"/> or
<a href="register.aspx" class="button"><asp:image id="Image1" runat="server" AlternateText="create a new account" skinid="create"/></a>
<p><asp:literal runat="server" id="FailureText" enableviewstate="False"> </asp:literal></p>
                </div>
            </layouttemplate>
        </asp:login>
    </anonymoustemplate>
    <LoggedInTemplate>
        <h4><asp:loginname id="LoginName1" runat="server" formatstring="Welcome {0}!" /></h4>
    </LoggedInTemplate>
    </asp:loginview>
```

Анонімному користувачу надається можливість зайти або зареєструватися, а авторизований бачить вітання зі своїм ім’ям.

Крім того, за допомогою <RoleGroups> можна створювати шаблони, які будуть показані користувачам з певними ролями:

```
<RoleGroups>
    </asp:RoleGroup>
    <asp:RoleGroup Roles="Moderator">
        <ContentTemplate>
Ви можете видаляти чужі повідомлення.
        </ContentTemplate>
    </asp:RoleGroup>
    <asp:RoleGroup Roles="ClubMember">
        <ContentTemplate>
Ви можете заходити до клубного форуму.
```

```
        </ContentTemplate>
    </RoleGroups>
```

Шаблони груп проглядаються по порядку, і показується той з них, що знайдено першим з груп, в які входить користувач. Наприклад, якщо він належить до груп Moderator і ClubMember, буде показаний шаблон Moderator. Ролі можна перераховувати через кому, тоді шаблон застосуємо до всіх зазначених груп:

```
<asp:RoleGroup Roles="Moderator, Administrator">
```

Інші елементи управління цієї групи – форми і майстри – Login, PasswordRecovery, ChangePassword.

CreateUserWizard дозволяє створювати користувачів, використовуючи службу Membership. Звісно, у ньому відбувається валідація введених даних. Наприклад, довжина пароля повинна бути не менше 7 знаків і в ньому повинен бути присутнім хоча б один символ – не буква і не цифра.

41.3. Персоналізація

Чимало сайтів збирають інформацію про користувачів, щоб підлаштувати показ інформації під їх особисті смаки. Часто для цього використовують файли-cookie, об’єкти додатку і сесії.

У ASP. NET 2.0 з’явилися нові зручні способи зберігати інформацію користувача. Це функція персоналізації. Механізм персоналізації дозволяє встановити прямий зв’язок між користувачем і всіма даними, що відносяться до нього. При цьому його налаштування зберігаються не в файлах-cookie на комп’ютері користувача, які він може стерти, а на сервері. Їх можна помістити у будь-яке сховище даних.

Модель персоналізації проста і розширювана.

У файлі web.config міститься інформація про те, які дані про користувача необхідно зберігати. Вона записується у секції <configuration> <system.web> перед секцією authentication:

```
<profile>
    <properties>
      <add name="FirstName" />
      <add name="LastName" />
      <add name="LastVisited" />
      <add name="Age" />
      <add name="Member" />
    </properties>
  </profile>
```

Профіль персоналізації може зберігати дані про авторизованому користувача, але може обслуговувати і анонімного користувача. За замовчуванням анонімна персоналізація вимкнена. Щоб її увімкнути, потрібно у файлі web.config створити запис:

```
<anonymousIdentification enabled="true" />
```

(також у секції <system.web>)

Коли анонімна персоналізація включена, ASP. NET зберігає унікальний ідентифікатор для кожного анонімного користувача. Він надсилається разом з будь-яким запитом. Ідентифікатор зберігається у файлі-cookie користувача, а додаткові дані, які вдалося зібрати про його переваги, – на сервері. За замовчуванням ім'я файлу-cookie – .ASPXANONYMOUS. Його можна змінити за допомогою атрибуту cookieName елемента anonymousIdentification:

<anonymousIdentification enabled="true" cookieName=".intuit"/>

Час зберігання файла-cookie у хвилинах визначається атрибутом cookieTimeout.

За замовчуванням воно дорівнює 100 000 хвилинам (69,4 доби).

Від використання cookie можна відмовитися, наприклад, написавши:

cookieless="UseUri"

У такому випадку ідентифікатор передається через рядок URL:

cookieless="AutoDetect"

У цьому випадку визначаються налаштування користувача. Якщо можливість зберігати cookie вимкнена, використовується рядок URL.

Анонімний ідентифікатор за своєю суттю являє собою GUID (32-байтное число, алгоритм генерації якого гарантує глобальну унікальність). Властивість AnonymousId об'єкта Request сторінки дозволяє отримати до нього доступ.

Для того щоб визначити, які дані можна зберігати для анонімного користувача, в елемент add name визначають атрибут allowAnonymous:

<add name="LastVisited" allowAnonymous="true"/>

Контрольні запитання

1. Що таке глобалізація та локалізація?
2. В якому просторі імен укладені класи для роботи з інформацією про культури?
3. Що таке аутентифікація та авторизація?
4. Який елемент визначає спосіб аутентифікації?
5. Який метод визначає чи належить користувач до ролі?
6. Який елемент визначає частину сайту, доступ до якої потрібно обмежити?
7. Для чого використовується елемент управління LoginStatus?
8. Які функції механізму персоналізації?
9. Який елемент управління дозволяє показати ім'я користувача?
10. Який атрибут Forms вказує час, після закінчення якого cookie застаріває?

Тема 42. Конфігурування, оптимізація та розгортання веб-додатку ASP.NET

42.1. Установка і конфігурація IIS

Компонент IIS включений як частина установки Windows (як для сервера, так і для робочих станцій) і вимагає активізації і конфігурації. Найпростіший спосіб активізації і конфігурації IS 7 в будь-якій версії Windows – використання програми установки веб-платформи (Web Platform Installer – WebPI). WebPI надає стандартний інтерфейс і каталог підключаються компо-

нентів, які розширюють набір функціональних можливостей IIS. У цьому розділі ми розглянемо застосування WebPI для приведення IIS в дію. У міру розгляду різних варіантів розгортання ми будемо повертатися до WebPI для установки додаткових засобів.

На замітку! У цій лекції передбачається, що ви входите в систему сервера з обліковим записом адміністратора. Це дозволить встановити IIS 7 і управляти його роботою.

Щоб запустити WebPI, відкрийте браузер на сервері і зверніться до сторінки http://www.microsoft.com/web/downloads/platform.aspx. Відобразиться посилання для завантаження програми установки. Залежно від операційної системи і налаштувань браузера будуть виведені попередження, пов'язані з безпекою. Завантажте та запустіть програму установки. Це призведе до встановлення WebPI, а не IIS. По завершенні установки WebPI запуститься і відкриється вікно з What's New? (Нові можливості) WebPI, показане на рис. 42.1.

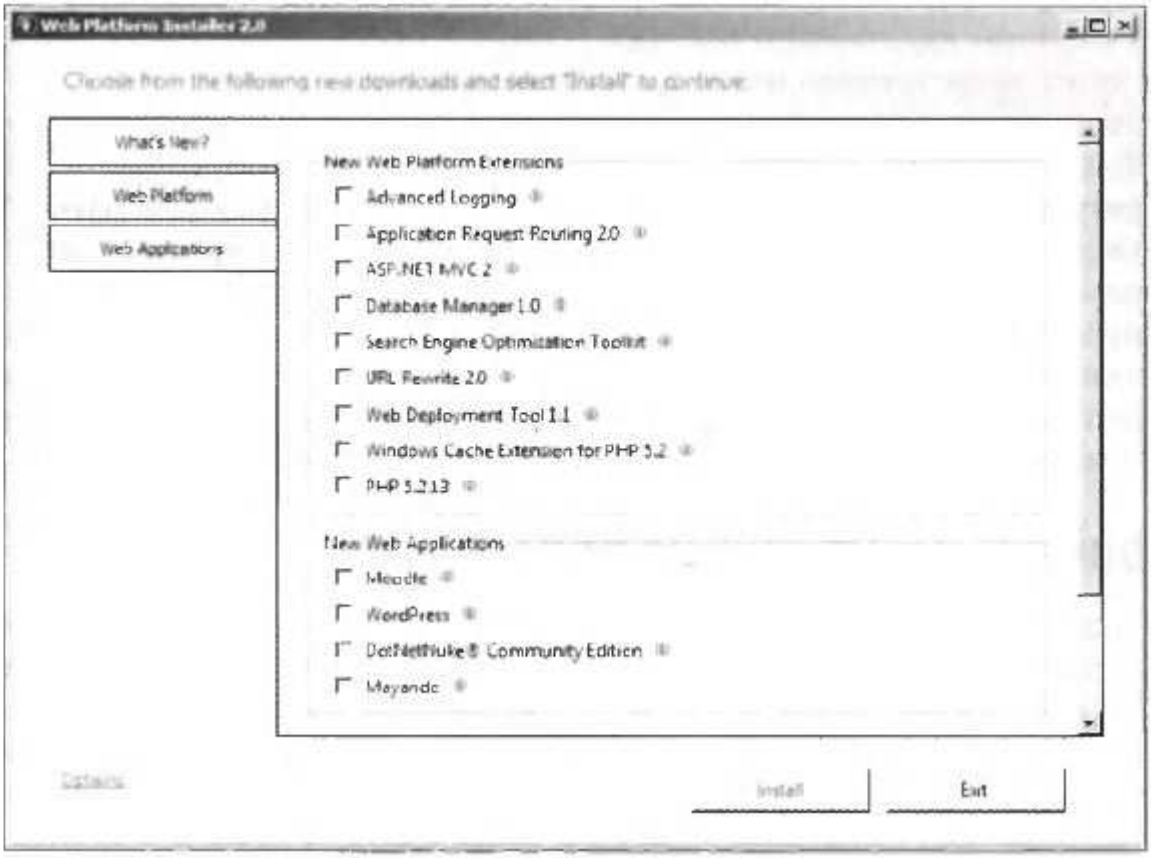

Рисунок 42.1 – Екран What's New

Клацніть на вкладці Web Platform (Веб-платформа). Як легко переконатися, WebPI дозволяє встановлювати не тільки IIS 7. Можна встановити також SQL Server Експрес, різні платформи побудови веб-додатків і ряд інструментів підтримки. Для простоти ми встановимо тільки ті елементи, які потрібні для вивчення матеріалу цієї глави.

Спочатку виберіть потрібні компоненти IIS 7. Клацніть на посиланні Click to include the recommended products (Клацніть, щоб додати рекомендовані продукти в розділі WebServer (Веб-сервер). З'явиться зелена мітка пра-

порця. Клацніть на посиланні Customize (Налаштувати) у розділі Web Server. В результаті відобразиться список окремих доступних функціональних засобів IIS 7 – оберіть варіант ASP.NET.

Поверніться на вкладку Web Platform і виберіть опцію .NET Framework 4.0 в розділі Frameworks and Runtimes (Платформи та виконуючі середовища). Поки нічого більше не потрібно, тому клацніть на кнопці Install (Встановити). З'явиться список компонентів, які будуть встановлені, і пов'язані з ними умови ліцензійної угоди. Прийміть умови угоди, щоб почати завантаження і установку.

Значна частина необхідних програмних компонентів вже входить до складу установки Windows. Програма WebPI завантажить будь-які відсутні компоненти і сконфігурує IIS. По завершенні установки і конфігурації відкриється екран з оглядом результатів установки. Вийдіть з WebPI.

Щоб перевірити правильність роботи всіх встановлених компонентів, відкрийте веб-браузер на сервері і перейдіть до локального вузла (localhost). Відкриється використовувана за замовчуванням сторінка вітання IIS 7.

42.2. Управління IIS 7

Після того як компонент IIS 7 встановлений і запущений, ним можна управляти. Для цього призначений інструмент IIS Manager (Диспетчер служб IIS), встановлений на сервері програмою WebPI. Його можна знайти в меню Start (Пуск). Конкретне розташування може залежати від використовуваної версії Windows. Ярлик програми буде розташовуватися в розділі Programs (Програми) або Administrative Tools (Адміністрування). Початкова сторінка IIS Manager показана на рис. 42.2.

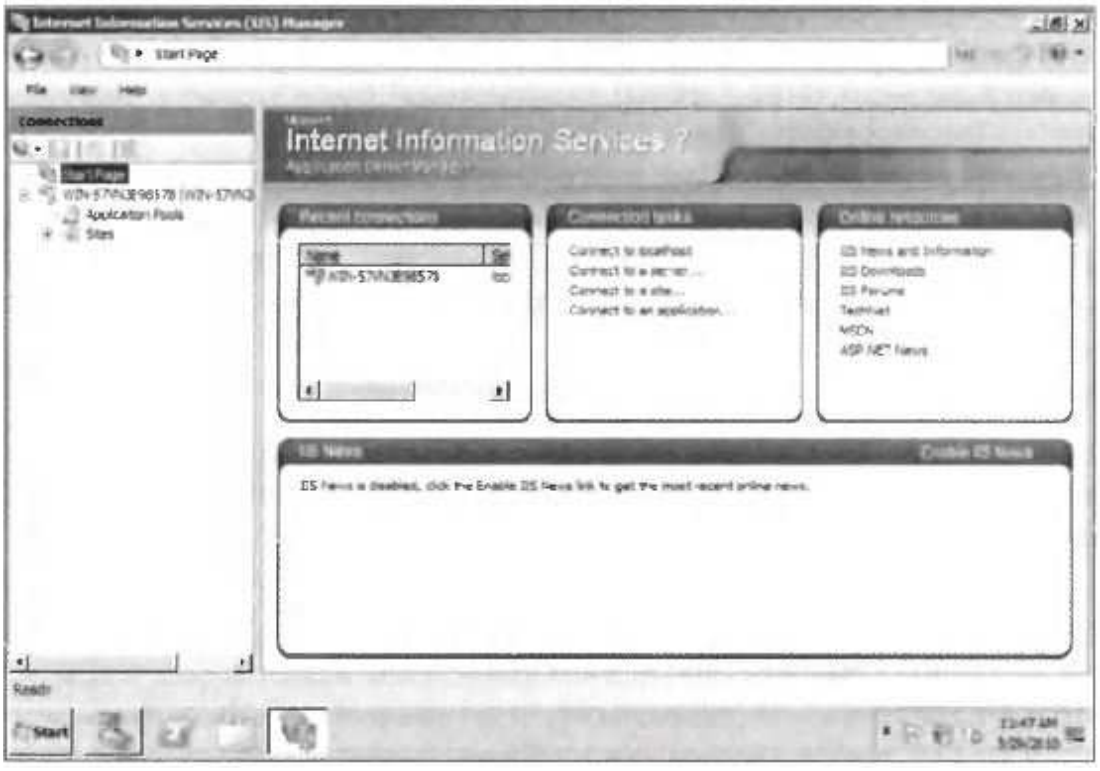

Рисунок 42.2 – Головна сторінка IIS Manager

Тепер потрібно ознайомитися з рядом термінів, які використовуються в IIS. У лівій частині вікна IIS Manager відображаються запис StartPage (Головна сторінка) і запис з ім'ям використовуваного сервера. Наш сервер має важкий для запам'ятовування ім'я WIN–57VN3E98578, сгенероване за замовчуванням Windows Server, яке буде використовуватися в більшості прикладів цієї глави. Якщо клацнути на імені, відкриється уявлення сервера, показане на рис. 42.3.

Рисунок 42.3 – Уявлення сервера IIS Manager

Це уявлення відображає набір значків, які дозволяють конфігурувати параметри сервера. У правій частині екрана розташований список доступних дій. Наприклад, в цьому поданні можна запускати, зупиняти і перезапускати сервер. Якщо розгорнути елемент сервера в деревовидному уявленні в лівій частині екрана, відобразиться елемент Sites (Сайти), що містить єдину запис Default Web Site (Веб-сайт за замовчуванням). Сайт – це колекція файлів і каталогів, що утворюють веб-сайт. На одному сервері IIS може підтримувати кілька сайтів, як правило, на різних портах TCP/ IP (за замовчуванням використовується порт 80). Поєднання імені сервера і порту сайту утворює першу частину URL-адреси. Наприклад, при використанні сервера my web server з сайтом, підключеним до порту 80, URL-адреса виглядає наступним чином:

http://mywebserver:80

Кожен сайт може містити безліч файлів і каталогів. Кожен з них утворює частина URL-адреси. Так, URL-адресу статичної сторінки mypage.html, розташованої в каталозі myfiles, буде наступним:

http://mywebserver:80/myfiles/mypage.html

На замітку! У деяких ситуаціях ім'я, під яким сервер відомий вам, і ім'я, яке клієнти використовують для отримання вмісту, будуть відрізнятися. Ми залишимо цей нюанс без уваги, але адміністратор сервера або компанія, що надає послуги хостингу, нададуть необхідні відомості, якщо це важливо для конкретного сервера.

Отже, кожен сервер може підтримувати безліч сайтів, кожен з яких працює на іншому порту або з іншою IP-адресою. Кожен сайт може мати безліч файлів і каталогів, і поєднання цих елементів надає інформацію про URL- адресу. Ми повернемося до URL-адрес та використання IIS Manager під час розгляду кожного з підходів до розгортання.

42.3. Розгортання веб-сайту

Після того як IIS 7 запущений і ви навчилися їм управляти, можна приступати до розгортання веб-сайтів. У найпростішому випадку розгортання веб-додатків ASP. NET зводиться до копіювання структури каталогів додатку на цільовий комп'ютер та налаштування середовища. Для простих додатків це майже завжди так. Однак якщо додаток використовує бази даних або звертається до інших ресурсів, доведеться виконати ряд додаткових кроків. Нижче перераховані звичайні фактори, що вимагають додаткових дій по конфігурації.

- Скопіюйте всі необхідні файли програми на цільовий комп'ютер. Понад це робити нічого не знадобиться. Однак при використанні глобальних збірок, доступ до яких здійснюється через GAC, необхідно упевнитися в їх існуванні. Якщо вони не існують, їх необхідно встановити за допомогою утиліти командного рядка gacutil.exe, що входить до складу .NET Framework.
- Створіть і налаштуйте базу даних для програми. Важливо не тільки створити базу даних та її таблиці, але і налаштувати облікові записи для входу на сервер бази даних і користувачів бази даних. Не забувайте, що у разі застосування вбудованої аутентифікації для підключення до бази даних SQL Server обліковий запис, під яким виконується ASP.NET (обліковий запис пулу додатків або обліковий запис aspnet_wp.exe), буде потрібно конфігурувати в якості користувача бази даних програми. Підхід веб-розгортання може спростити розгортання бази даних (див. Розділ "Використання веб-розгортання" далі в главі). Якщо ж застосовується інший підхід, конфігурування і заповнення баз даних доведеться виконувати вручну.
- Налаштуйте IIS так, як того вимагає додаток. Створіть необхідні пули додатків, зробіть каталог додатка доступним для спільного використання в якості віртуального каталогу і належним чином налаштуйте віртуальний каталог. Більш докладно це описано в розділі "Використання пулів додатків" далі у розділі.
- Встановіть права облікового запису Windows для користувача робочого процесу. Користувач, від імені якого запускається робочий процес (w3wp.exe), потребує доступу для читання до каталогів додатка. Якщо додаток звертається до інших ресурсів, наприклад, до системного реєстру або до журналу подій, для облікового запису робочого процесу знадобиться налаштувати дозвіл на доступ до цих ресурсів.

- Якщо необхідно обробляти будь-які URL-адреси з розширеннями імен файлів, які відрізняються від розширень, зареєстрованих під час установки ASP.NET за замовчуванням, додайте відображення файлів IIS.
- Налаштуйте ASP.NET (і налаштування IIS 7.0, специфічні для програми) у файлі web.config для виробничих середовищ. Іншими словами, додайте (або модифікуйте) будь-які потрібні рядки з'єднання і параметри налаштування додатку, а також параметри безпеки та авторизації, параметри налаштування стану сеансу і параметри налаштування глобалізації.
- У деяких випадках необхідно також модифікувати файл machine.config. Наприклад, якщо робота виконується в середовищі веб-хостингу, а в цілях балансування навантаження додаток працює на безлічі веб-серверів, необхідно синхронізувати будь-які ключі шифрування, що використовуються для шифрування мандатів аутентифікації за допомогою форм або стану уявлення на всіх цих комп'ютерах. Ці ключі зберігаються у файлі machine.config і повинні бути однаковими на кожному комп'ютері вебформи з тим, щоб один комп'ютер міг розшифрувати інформацію, зашифровану іншим комп'ютером, що раніше обробляв запит.

Основне завдання пов'язане з розгортанням вмісту додатка на сервері IIS 7. У наступних розділах описані три найбільш часто вживаних підходу до розгортання. На замітку! У цій лекції передбачається наявність у вас прав адміністратора і вхід в систему сервера від імені облікового запису адміністратора. Якщо використовується сервер, що розділяє, або постачальник послуг хостингу, то постачальник надішле відомості про обліковий запис, який повинен застосовуватися під час розгортання веб-сайту. Для простоти ми будемо вважати, що робота виконується з обліковим записом адміністратора, але при управлінні власним сервером слід подумати про використання менш привілейованого облікового запису.

42.3.1. Розгортання за допомогою копіювання файлів

Найпростіший спосіб розгортання веб-сайту передбачає копіювання файлів з робочої станції на сервер. Незважаючи на простоту, цей підхід вимагає наявності безпосереднього доступу до сервера. Саме з цієї причини деякі IT-підрозділи і компанії, що надають послуги хостингу, не підтримують цю опцію. Але вона може бути найпростішою під час управління власним сервером або за наявності особливих домовленостей з компанією хостингу.

На замітку! Ця методика розгортання працює для всіх версій IIS 7, що функціонують під управлінням всіх версій Windows.

Перш ніж розгортати веб-сайт, потрібно підготувати IIS. Головне рішення, яке потрібно прийняти, стосується місця розміщення вмісту і його впливу на кінцеву URL-адресу. Почнемо з очевидного підходу – припустимо, що необхідно, щоб URL- адреса для вмісту даного прикладу була наступною:

http://<им'я_сервера>:80/WebsiteDeployment/FileCopy

IIS потрібно підготувати так, щоб було куди скопіювати наш файл. В IIS Manager оберіть елемент Default Web Site. Як видно з його імені, це сайт за замовчуванням на сервері. Клацніть на ньому правою кнопкою миші і в

контекстному меню оберіть Explore (Провідник), щоб відрити вікно провідника Windows для заданого за замовчуванням каталогу IIS, яким є inetpub\wwwroot на системному томі (як правило, C:\). Створіть каталог WebsiteDeployment, а в ньому – каталог FileCopy (щоб забезпечити існування шляху inetpub\wwwroot\WebsiteDeployment\FileCopy). Закрийте вікно провідника, щоб повернутися в IIS Manager. Клацніть правою кнопкою на записі Default Web Site і в контекстному меню оберіть пункт Refresh (Оновити), щоб побачити новий каталог.

Для демонстрації цієї методики розгортання ми створили дуже простий веб-сайт. Цей проект містить єдину форму ASP.NET з однією міткою, як показано на рис. 42.4.

Рисунок 42.4 – Форма ASP.NET для прикладу веб-сайту

У коді цієї форми визначений текст мітки, що відображає версію .NET Framework, яка використовується для обслуговування даного сайту:

```
using System;
    public partial class Default : System.Web.UI.Page
    { protected void Page_Load(object sender, EventArgs e)
        {   Labell.Text = System.Environment.Version.Major.ToString();}
    }
```

Під час використання цієї методики розгортання потрібно просто скопіювати файли в створений каталог. Перемістіть файли веб-сайту на сервер будь-яким відповідним способом – за допомогою загального мережевого диска, зйомного диску USB, диску DVD і т.п. – і скопіюйте файли Default.aspx і Default.aspx.cs в каталог FileCopy, створений на сервері. Коли файли будуть скопійовані, поверніться у вікно IIS Manager на сервері, натисніть правою кнопкою на папці FileCopy в деревовидному вигляді та в контекстному меню оберіть пункт Refresh. У нижній частині екрана натисніть на кнопку Content View (Перегляд вмісту). У центральній частині вікна повинні відобразитися два файли веб-сайту, як показано на рис. 42.5.

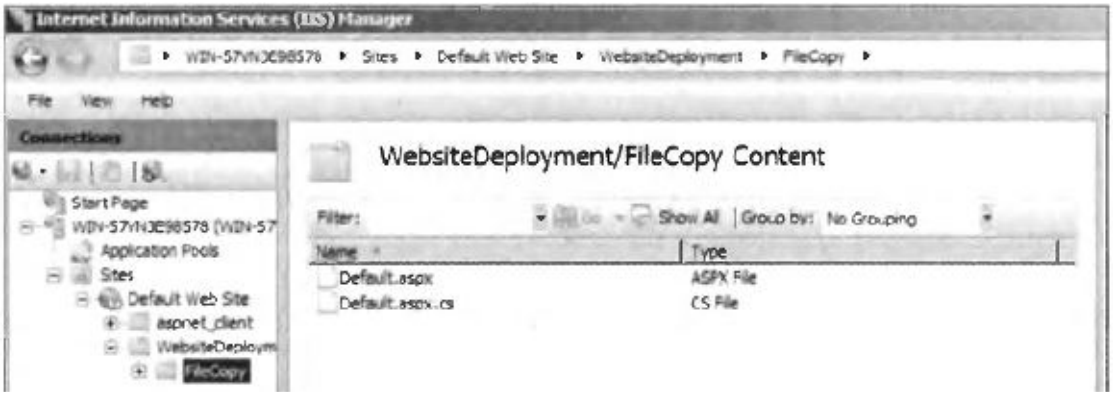

Рисунок 42.5 – Елементи розгорнутого сайту

У цьому полягає найбільш важлива частина цієї технології розгортання – спочатку повинна бути створена структура каталогів, яка представляє необхідну URL-адресу, а потім у готові каталоги копіюються файли веб-сайту. Подивимося, як це виглядає на практиці. Для цього оберіть File Copy в IIS Manager і клацніть на посиланні Browse (Огляд) у правій частині вікна. Відкриється веб-браузер з завантаженою URL-адресою створеної папки. Вікно браузера має виглядати подібно показаному на рис. 42.6.

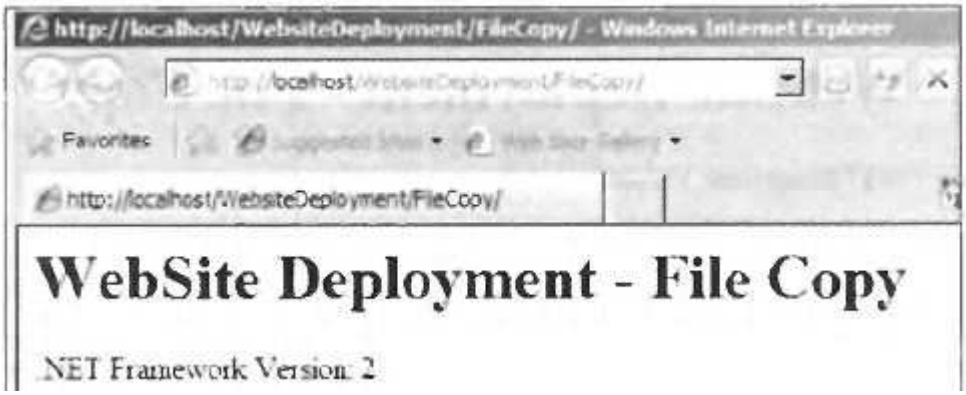

Рисунок 42.6 – Перегляд розгорнутого веб-сайту

Поглянувши на URL-адреса, можна впевнитися в тому, що був отриманий бажаний результат. Браузер завантажив веб-сайт з наступної адреси:

http://localhost/WebsiteDeployment/FileCopy/

Згадайте, що local host – це спеціальне ім'я, що позначає поточний комп'ютер, а URL-адреса, в якій не вказано порт, буде використовувати порт 80. Можете перевірити це, направивши браузер за наступною URL-адресою:

http://win-57vn3e9857:80/WebsiteDeployment/FileCopy/

Результат буде повністю аналогічний попередньому. Можливо, ви звернули увагу, що версією .NET Framework, про яку повідомляється на рис. 42.6, є 2. Це спричинить виникнення проблем для будь-якого веб-сайту, який залежить від функціональності ASP.NET 4. Робота поки не завершена – розгорнутий веб-сайт повинен бути налаштований так, щоб він використовував версію .NET Framework 4. Для цього знадобиться змінити параметри налаштування використовуваного за замовчуванням пулу додатків. Поки не замислюйтесь про те, що собою являють пули додатків – вони будуть детально розглядатися в розділі „Управління веб-сайтом” далі у розділі. Поки досить зробити так, щоб веб-сайт використовував версію .NET Framework 4.

У вікні IIS Manager розгорніть елемент сервера і клацніть на елементі Application Pools (Пули додатків). Клацніть на посиланні Set Application Pool Defaults (Визначити значення за замовчуванням для пулу додатків). У діалоговому вікні Application Pool Defaults (Значення за замовчуванням для пулу додатків) змініть значення налаштування .NET Framework Version (Версія середовища .NET Framework) на v4.0, як показано на рис. 42.7.

Поверніться в браузер і перезавантажте сторінку. Тепер версією .NET Framework, повідомленої сторінки веб-сайту повинна бути 4 (рис. 42.8).

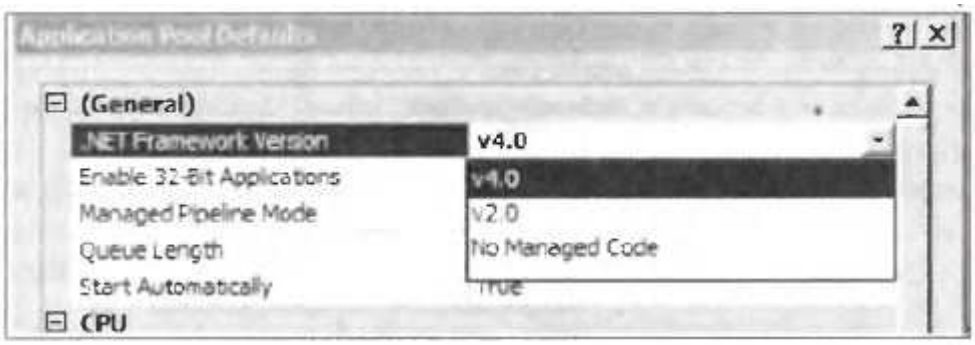

Рисунок 42.7 – Установка версії .NET Framework в діалоговому вікні Application Pool Defaults

Рисунок 42.8 – Веб-сайт, що функціонує під управлінням версії .NET Framework 4

IIS знадобиться також вказати, що розгорнутий сайт є додатком. Це не обов'язково, але під час розгортання додатків ASP.NET майже завжди буде бажаним – активізується стан сеансу та інші функціональні засоби ASP.NET. Клацніть правою кнопкою миші на папці FileCopy в області Connections (Підключення) і в контекстному меню оберіть Convert to Application (Перетворити в додаток), як показано на рис. 42.9.

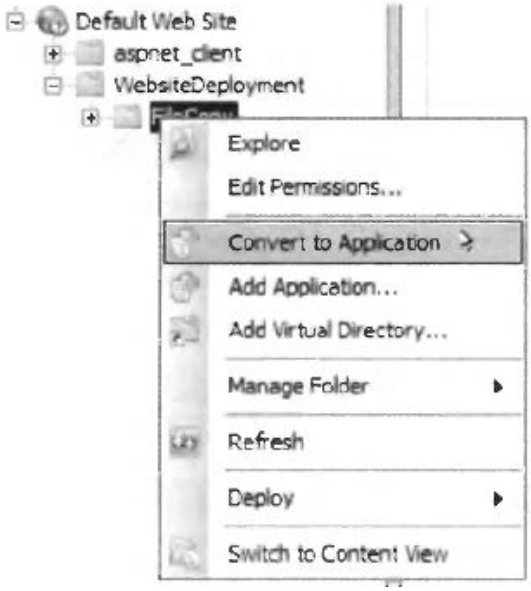

Рисунок 42.9 – Перетворення в додаток

Відкриється діалогове вікно Add Application (Додавання програми).Пул додатків, що використовується, можна змінити, натиснувши на кнопку Select (Обрати). Детальніше пули додатків розглядаються далі у розділі. Налаштувати обліковий запис користувача, який IIS буде застосовувати для доступу до вмісту сайту, можна за допомогою кнопок Connect as... (Підкл. Як...) і Test Settings... (Тест налаштувань...). Поки що просто клацніть на кнопці ОК. Можливо, доведеться обрати пункт Refresh (Оновити) в меню View (Вид) (або, як це часто має місце, закрити і знову відкрити IIS Manager), але тепер значок записи FileCopy в деревовидному поданні повинен змінитися

42.3.2. Використання веб-розгортання

Веб-розгортання дозволяє розгортати веб-сайт безпосередньо з середовища Visual Studio 2010, використовуючи HTTP – але тільки за наявності створеного проекту, а не у беспроектному варіанті. Не плутайте веб-розгортання з серверними розширеннями Front Page (Front Page Server Extensions – FPSE). Розширення FPSE застаріли, хоча їх можна завантажити та встановити для IIS 7.0 в Windows Server 2008 і Windows Vista.

На замітку! Ця технологія розгортання працює тільки в серверних версіях Windows.

Щоб можна було використовувати функцію веб-розгортання, в IIS знадобиться додати два компоненти. Для цього доведеться знову скористатися програмою WebPI, яка була встановлена на комп'ютер під час установки і конфігурації IIS 7 на початку лекції. Запустіть WebPI через меню Start і в розділі Web Server (Веб-сервер) вкладки Web Platform (Веб-платформа) позначте прапорець Customize (Налаштувати).

У розділі Deployment and Publishing (Розгортання та публікація) позначте прапорець Web Deployment Tool (Засіб веб-розгортання). На момент написання цієї книги поточною версією була 1.1. У розділі Management (Управління) позначте прапорець Management Service (Служба управління). Щоб цей підхід до розгортання працював правильно, важливо встановити обидва ці компоненти.

Після завершення установки запустіть IIS Manager і оберіть елемент сервера в дереві Connections (Підключення). Прокрутіть до нижньої частини екрана. Якщо все було встановлено правильно, повинні відобразитися два нових значка – Management Service (Служба управління) та Management Service Delegation (Делегування служби управління).

Тепер потрібно налаштувати і запустити службу Windows, яка буде прослуховувати запити віддаленого управління IIS. Двічі клацніть на значку Management Service, щоб відкрити сторінку властивостей Management Service.

Позначте прапорець Enable remote connections (Дозволити віддалені підключення), клацніть на кнопку Apply (Застосувати), а потім на кнопку Start (Запустити).

Дозвіл віддалених підключень дозволяє іншим комп'ютерам виконувати операції з управління IIS. У цьому розділі даний засіб використовується для публікації безпосередньо з середовища Visual Studio, а в наступному розділі – для публікації пакета з середовища IIS Manager. Ці функціональні мо-

жливості демонструються з використанням облікового запису Administrator (Адміністратор), але значок Management Service Delegation дозволяє вказувати інші облікові записи і точно конфігурувати дозволені дії.

Порада. За замовчуванням служба Management Service не працює автоматично у початковому завантаженні сервера. Щоб вона запускалася автоматично, змініть настройку служби Web Management Service (служба веб-управління) у вкладці Services (Служби).

Веб-розгортання можна застосовувати з середовища Visual Studio 2010 для проектів додатків ASP.NET, але не для веб-сайтів, створених без використання проектів. Щоб продемонструвати цей вид розгортання, ми створили проект, що містить єдину сторінку, яка повідомляє версію .NET Framework, подібно попередньому прикладу. Єдина відмінність між веб-сайтом, створеним для цього прикладу, і попереднім полягає у використанні проекту Visual Studio.

Щоб створити цей приклад самостійно, створіть новий проект Visual Studio, скориставшись шаблоном ASP.NET Empty Web Application (Порожній веб-додаток ASP.NET), і додайте в проект нову веб-форму Default.aspx. У цю форму можна помістити будь-які компоненти – ми використовували той самий формат, що і раніше, як показано на рис. 42.10.

Рисунок 42.10 – Приклад веб-сайту

Оскільки потрібна впевненість у тому, що наш сайт працює під управлінням необхідної версії .NET, а це дасть можливість користуватися всіма засобами ASP.NET 4, ми додали код для відображення старшого номера версії платформи:

```
using System;
namespace Web_Deploy {
    public partial class Default: System.Web.UI.Page {
        protected void Page_Load (object sender, EventArgs e) {
            Labell.Text =
        System.Environment.Version.Major.ToString ();
        }
    }
}
```

На шляху від розробки до тестування і введення в дію багато проектів проходять через ряд етапів розгортання, кожен з яких передбачає розгортання

на проміжних серверах з різними конфігураціями. Модель веб-розгортання включає в себе корисний засіб спрощення цього процесу за рахунок трансформації файлу web.config відповідно до кожного етапу процесу розгортання.

Якщо розкрити web.config у вікні Solution Explorer (Провідник рішень), відобразяться елементи для кожної конфігурації збірки, визначеної в проекті – файли трансформації web.config. За замовчуванням Visual Studio 2010 створює конфігурації Debug (Налагодження) і Release (Редакція), але за допомогою диспетчера конфігурацій (Configuration Manager) можна додавати нестандартні конфігурації.

Під час веб-розгортання XML-оператори у файлі трансформації для активної збірки застосовуються до вихідного файлу web.config проекту, забезпечуючи додавання, зміну та видалення параметрів конфігурації. Під час розгортання проекту трансформовані параметри включаються в нього – тобто, наприклад, не знадобиться вручну змінювати рядки підключення, щоб працювати з базами даних, розташованими в області іншого етапу. Це не тільки зручно, але й знижує ймовірність допущення помилок конфігурації, що руйнують середу під час розгортання. Для кожного прикладу ми будемо використовувати один і той самий вихідний файл web.config з наступним вмістом:

```
<?xml version="l.0"?>
<configuration>
<connectionStrings>
<add name="NorthwindConnectionString" connectionstring="Data Source=. \SQLEXPRESS; AttachDbFilename=|DataDirectory| \Northwind.mdf; Integrated Security=True;User Instance=True" providerName="System.Data.SqlClient" />
</connectionStrings>
<system.web>
<compilation debug="true" targetFramework="4.0" />
</system.web>
</configuration>
```

Цей код містить рядок підключення до бази даних Northwind, розміщений на робочій станції, і в якості цільового середовища вказує .NET 4 із застосуванням налагоджувальних символів.

Найбільш часто вживаною трансформацією є зміна атрибуту, який повинен бути встановлений у вихідному файлі web.config – наприклад, зміна рядка підключення до бази даних. Ось як виглядає файл трансформації web.release.config, який змінює рядок підключення, щоб він вказував на машину з Windows Server 2008 R2:

```
<?xml version="l.0"?>
<configuration xmlns:xdt="http://schemas.microsoft.com/XML-Document-Transform">
<connectionStrings>
<add name="NorthwindConnectionString; connectionString= "DataSource=WIN-57VN3E98578; InitiaCatalog=Northwind; IntegratedSecurity= True" xdt:Transform="SetAttributes" xdt:Locator*"Match(name)" />
</connectionStrings>
```

</configuration>

Для установки існуючого атрибуту застосовується теж саме визначення, що і у вихідному файлі (в даному випадку, додавання іменованого елемента в connectionStrings), з додаванням елементів Transform і Locator. Установка значення елемента Transform в SetAttributes заміщає відповідні елементи перетвореними даними – в цьому прикладі, зміненим рядком підключення. Елемент Locator вказує, що порівняння елемента з початковим повинно здійснюватися за атрибутом імені, в даному випадку – NorthwindConnectionString. Можливість вказівки того, як має виконуватися порівняння елементу, корисна при існуванні кількох атрибутів одного типу.

У вихідний код, що завантажується, включений приклад проекту, який містить цю та наступні трансформації. Під час розгортання проекту з використанням конфігурації Release застосовується відповідна трансформація, і файл web.config, встановлений в IIS, міститиме нове значення рядка підключення, як показано в наступному прикладі:

```
<?xml version="l.0"?>
<configuration>
    <connectionStrings>
<add name="NorthwindConnectionString" connectionString= "Data Source=WIN-57VN3E98578;Initial Catalog=Northwind; IntegratedSecurity= „True" providerName="System.Data.SqlClient" />
    </connectionStrings>
    <system.web>
        <compilation debug="true" targetFramework="4.0" />
    </system.web>
</configuration>
```

Як бачите, механізм трансформації замінив рядок підключення етапу розробки (який використовує вкладений файл SQL Server Express) рядком підключення етапу розгортання (який виконує підключення до сервера). Трансформація не застосовується до тих пір, поки не буде виконано розгортання, тому навіть під час перемикання на конфігурацію збірки і перекомпонуванні проекту значення у файлі web.config проекту як і раніше будуть використовуватися на робочій станції.

Щоб вставити елемент в файл конфігурації, для атрибуту Transform необхідно вказати значення Insert. Додаткова трансформація, яка додає другий рядок підключення, виділена напівжирним:

```
<?xml version="l.0"?>
<configuration xmlns:xdt="http://schemas.microsoft.com/XML-Document-Transform">
    <connectionStrings>
<add name="NorthwindConnectionString" connectionString="DataSource= WIN-57VN3E98578; InitialCatalog=Northwind;Integrated Security=True" xdt:Transform="SetAttributes" xdt:Locator="Match(name)"/> й
    </connectionStrings>
    <connectionStrings>
```

```
<add name="Connection2" connectionString="Data Source=MyServer;
InitialCatalog=MyDB; Integrated Security=True" xdt:Transforms"Insert"/>
</connectionStrings>
</configuration>
```

Перший приклад було залишено для того, щоб продемонструвати формат вказівки кількох трансформацій. Під час розгортання проекту перетворений файл web.config набуває наступний вигляд (додані рядки виділені напівжирним шрифтом):

```
<?xml version="l.0"?>
<configuration>
<connectionStrings>
<add name="NorthwindConnectionString",connectionString=" Data
Source=.\SQLEXPRESS;
AttachDbFilename=|DataDirectory|\Northwind.mdf;
Integrated Security=True; UserInstance=True" providerName=
"System.Data.SqlClient" />
<add name= "Connection2" connectionString="Data Source=MyServer;
InitialCatalog=MyDB; Integrated Security=True" />
</connectionStrings>
<system.web>
<compilation debug="true" targetFramework="4.0" />
</system.web>
</configuration>
```

Щоб замінити весь розділ файлу web.config, потрібно використати значення Replace атрибута Transform, як показано в наступному прикладі:

```
<?xml version="1.0"?>
<configuration xmlns:xdt="http://schemas.microsoft.com/XML-Document-Transform">
        <system.web xdt:Transform="Replace">
<customErrors defaultRedirect="GenericError.htm" mode="RemoteOnly">
            <error statusCode="500" redirect="InternalError.htm"/>
            </customErrors>
        </system.web>
</configuration>
```

Ця трансформація вказує, що весь блок system.web файлу web.config повинен бути замінений вмістом з файлу трансформації. Результат, отриманий під час розгортання проекту, має наступний вигляд:

```
<?xml version="l.0"?>
<configuration>
<connectionStrings>
<add name="NorthwindConnectionString" connectionString="Data Source=.\SQLEXPRESS;AttachDbFilename=|DataDirectory|\Northwind.mdf; Integrated-Security=True; UserInstance=True" providerName= "System.Data.SqlClient" />
</connectionStrings>
<system.web>
<customErrors defaultRedirect="GenericError.htm" mode="RemoteOnly">
```

```
<error statusCode="500" redirect="InternalError.htm" />
</customErrors>
</system.web>
</configuration>
```

Рядок підключення, використаний на етапі розробки, залишився незмінним, але цільова версія платформи була замінена.

Щоб видалити розділ файлу конфігурації, його потрібно оголосити з використанням значення Remove атрибута Transform, наприклад:

```
<?xml version="1.0"?>
<configuration xmlns:xdt="http://schemas.microsoft.com/XML-Document-Transform">
<connectionStrings xdt:Transforms"Remove"/>
</configuration>
```

Ця трансформація видаляє розділ connectionStrings, що під час розгортання призводить до наступного результату:

```
<?xml version="1.0"?>
<configuration>
    <system.web>
        <compilation debug="true" targetFramework="4.0" />
    </system.web>
</configuration>
```

Окремі атрибути можна видаляти за допомогою значення RemoveAttributes, вказуючи атрибути, що видаляться, в розділеному комами списку:

```
<?xml version="1.0"?>
<configuration xmlns:xdt="http://schemas.microsoft.com/XML-DocumentTransform">
    <system.web>
<compilation xdt:Transform= "RemoveAttributes(debug, targetFramework)"/>
    </system.web>
</configuration>
```

Ця трансформація видаляє атрибути debug і target Framework з елемента compilation, даючи наступний результат під час розгортання:

```
<?xml version="1.0"?>
<configuration>
    <connectionStrings>
<add name="NorthwindConnectionString" connectionString="Data Source=
.\SQLEXPRESS; AttachDbFilename=|DataDirectory|\Northwind.mdf; IntegratedSe-
curity=True; User Instance=True" providerName= "System.Data.SqlClient"/>
    </connectionStrings>
    csystem.web>
        <compilation />
    </system.web>
</configuration>
```

Ще одним корисним засобом веб-розгортання є публікація бази даних під час публікації проекту. Сценарій SQL, що містить схему бази даних або схему і дані, генерується в ході процесу розгортання і потім використовується для заповнення середовища розгортання. Цю функціональну можливість

слід застосовувати з певною обережністю, тому що виробничу базу даних легко замінити тестовими даними.

Конфігурування публікації бази даних – двокроковий процес. Щоб активізувати цей засіб, позначте прапорець Include all databases configured in the Package/Publish SQL tab (Включити всі бази даних, сконфігуровані на вкладці "Пакет/Публікація SQL") на вкладці Package/Publish Web (Пакет/Веб-публікація) властивостей проекту.

Щоб задати деталі, перейдіть на вкладку Package/Publish SQL (Пакет/Публікація SQL). Таблиця Database Entries (Записи баз даних) містить по одному запису для кожної бази даних, яка буде опублікована. Клацнувши на кнопці Import from web.config (Імпортувати з файлу web.config), можна використовувати рядки підключень, які вже визначені в проекті. Під час імпорту рядків підключення багато елементів форми будуть заповнені автоматично, і залишиться додати тільки рядок підключення до цільової базі даних. Якщо рядок цільового підключення залишити порожнім, використовуватиметься значення з файлу web.config. Якщо ж вказати трансформацію для цього рядка підключення, використовуватиметься перетворений результат. Нюанси трансформацій файлу web.config описані в попередньому розділі. За наявності декількох баз даних порядок їх публікації можна змінити за допомогою стрілок переміщення вгору і вниз в таблиці Database Entries. Можна також вказати, що саме буде містити сценарій SQL – тільки схему бази даних або схему і дані.

Корисна можливість – включення в сценарій SQL операторів, які будуть видаляти існуючу таблицю і дані під час розгортання. Для цього файл проекту повинен бути активізований безпосередньо – Visual Studio не пропонує для цього ніякого користувача інтерфейсу. Відкрийте файл з розширенням .csproj і знайдіть розділ PublishDatabaseSettings. Відшукайте дескриптор PreSource і додайте до цього рядка атрибут ScriptData = "False" (в наступному прикладі він виділений напівжирним):

<PublishDatabaseSettings>
<Objects>
<ObjectGroupName="NorthwindConnectionString-Deployment" Order="1">
<Destination Path= DataSource=WIN-57VN3E98578\SQLEXPRESS InitialCatalog=NorthwindPersist SecurityInfo=True; UserID= sa_deploy; Password=sa" />
<Object Type="dbFullSql">
<PreSource Path="Data Source=.\sqlexpress lnitialCatalog= NorthwinsPersist SecurityInfo=True UserID=sa_deploy Password=sa" ScriptDropsFirst=”true” ScriptSchema=True" ScriptData="False" CopyAllFull-TextCatalogs=False" />
<Source Path="obj\Debug\AutoScripts\NorthwindConnectionStringDeployment_SchemaOnly.sql" Transacted="True" />
</Object>
</ObjectGroup>
</Objects>
</PublishDatabaseSettings>

Visual Studio запропонує перезавантажити файл проекту, щоб нове налаштування вступило в дію. Після того як налаштування публікації сконфігуровані, розгортання проекту виконується, як описано в наступних розділах.

Застосовуючи ці функціональні можливості і опції, потрібно дотримуватися особливої обережності – недбалий вибір може вимагати відновлення бази даних з резервної копії. Мало що може бути настільки ж неприємним, як виявлення того, що виробнича база даних була стерта.

Щоб розгорнути веб-сайт з середовища Visual Studio, відкрийте проект веб-сайту і оберіть у меню Build (Збірка) пункт Publish (Публікувати). Точне найменування пункту меню буде залежати від імені, призначеного проекту. У розглянутому прикладі проекту пункт буде називатися Publish Web_Deploy.

При виборі цього пункту меню відкриється діалогове вікно Publish Web (Веб-публікація), показане на рис. 42.11. Воно є основним вікном засобу Web Deploy. У списку Publish method (Метод публікації) оберіть Web Deploy (Веб-розгортання). У полі Service URL (URL-адреса служби) введіть ім'я сервера, на якому буде виконуватися публікація – сервер Windows Server 2008 R2, що застосовується у прикладі, має ім'я WIN-57VN3E98578.

Рисунок 42.11 – Діалогове вікно Publish Web

Поле Site/application (Сайт/додаток) дозволяє вказати місце розгортання сайту і, як вже було сказано, воно утворює URL-адресу, яку будуть використовувати клієнти. У розглянутому прикладі URL-адреса має бути такою:

http://<servername>/WebsiteDeployment/WebDeploy/

Ця URL-адреса помістить приклад поряд з тим, який ми розгорнули копіюванням файлів, створених в попередньому розділі. Для цього в полі Site/application поміщено значення Default Web Site/WebsiteDeployment/ WebDeploy.

Позначте прапорець Mark as IIS application on destination (Помітити у місці призначення як додаток IIS). Це еквівалентно процесу розгортання вручну, використаному в попередній методиці.

У розділі Credentials (Мандати) відзначте прапорець Allow untrusted certificate (Дозволити неперевірені сертифікати), якщо в IIS 7 не був встановлений сертифікат, виданий центром сертифікації. На нашому сервері такий сертифікат відсутній, тому прапорець необхідно відзначити, щоб можна було виконати публікацію з використанням веб-розгортання. І, нарешті, введіть ім'я користувача та пароль для облікового запису, який повинен застосовуватися для публікації – в прикладі вказаний обліковий запис Administrator. Компанія, що надає послуги хостингу, або адміністратор сервера надасть відомості про обліковий запис, який потрібно буде використати.

Ось і все – тепер сайт можна розгорнути, клацнувши на кнопці Publish. Якщо оновити (або перезапустити) IIS Manager, відобразиться новий розгорнутий додаток (рис. 42.12). Будь-які виконані трансформації web.config і будь-які сконфігуровані бази даних будуть опубліковані.

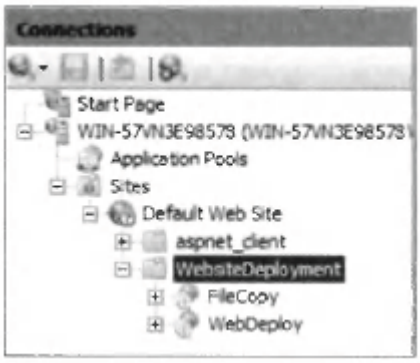

Рисунок 42.12 – Сайт, опублікований за допомогою веб-розгортання

Наприкінці можна протестувати розгорнутий додаток, щоб перевірити, чи працює створений приклад програми. Перейдіть в веб-браузері за наступною URL-адресою:

http://WIN-57VN3E98578/WebsiteDeployment/WebDeploy

Якщо всі дії, описані при створенні цього прикладу, були виконані правильно, результат повинен виглядати, як показано на рис. 42.13.

Рисунок 42.13 – Тестування розгортання

42.3.3. Використання FTP-розгортання

FTP-розгортання здійснює розгортання проекту на сервері за допомогою протоколу передачі файлів (File Transfer Protocol – FTP). Основна перевага FTP– розгортання – більш широкий діапазон підтримуваних платформ. Основний недолік цього методу полягає в тому, що при такому розгортанні виникає більше проблем, пов'язаних з брандмауером, ніж при веб-розгортанні.

На замітку! Цей підхід до розгортання підтримується у всіх версіях IIS 7 для всіх версій Windows.

Найпростіший спосіб установки і конфігурації FTP-розгортання передбачає застосування програми Web Platform Installer. Запустіть Web PI, перейдіть на вкладку Web Platform (Веб-платформа) і оберіть пункт Customize (Налаштувати) у розділі Web Server (Веб-сервер). Перейдіть та оберіть пункт FTP Publishing Service (Служба FTP-публікації). Клацніть на кнопці Install (Встановити), прийміть умови ліцензійної угоди і почніть установку. Служби IIS не виявлять зміни, якщо вони функціонували під час установки, тому при необхідності зупиніть і знову запустіть IIS.

Спочатку FTP-розгортання повинно бути включено на сайті IIS. Розгорніть дерево Connections у вікні IIS Manager, клацніть правою кнопкою миші на гілці Default Web Site (Веб-сайт за замовчуванням) і в контекстному меню оберіть пункт Add FTP Publishing (Додати FTP-публікацію). Відкриється перша сторінка майстра Add FTP Site Publishing Wizard (Майстер додавання FTP-публікації сайту), показана на рис. 42.14. Якщо сервер має більше одного мережевого інтерфейсу, в розділі Binding (Зв'язування) можна вибрати той з них, який буде прослуховуватися на предмет запитів розгортання. Якщо ж сервер має тільки один інтерфейс або необхідно прослуховувати кожен з встановлених інтерфейсів на предмет запитів, залиште в полі IP Address (IP-адреса) значення All Unassigned (Всі непризначені). Значенням за замовчуванням в полі Port (Порт) є 21 – стандартний TCP-порт для FTP.

У розділі SSL можна вимагати шифрування для захисту мережевого трафіку FTP- розгортання. Тестовий сервер не має встановленого сертифіката SSL, тому обрана опція No SSL (Без SSL). Клацніть на кнопці Next (Далі), щоб перейти до наступного екрану, який дозволяє вказати тих, кому дозволено користуватися FTP- розгортанням. Необхідно дозволити доступ тільки для облікового запису Administrator, тому включається аутентифікація за рахунок вибору

перемикача Basic (Базова), потім перемикача Specified users (Зазначені користувачі) і введення Administrator в полі імені облікового запису.

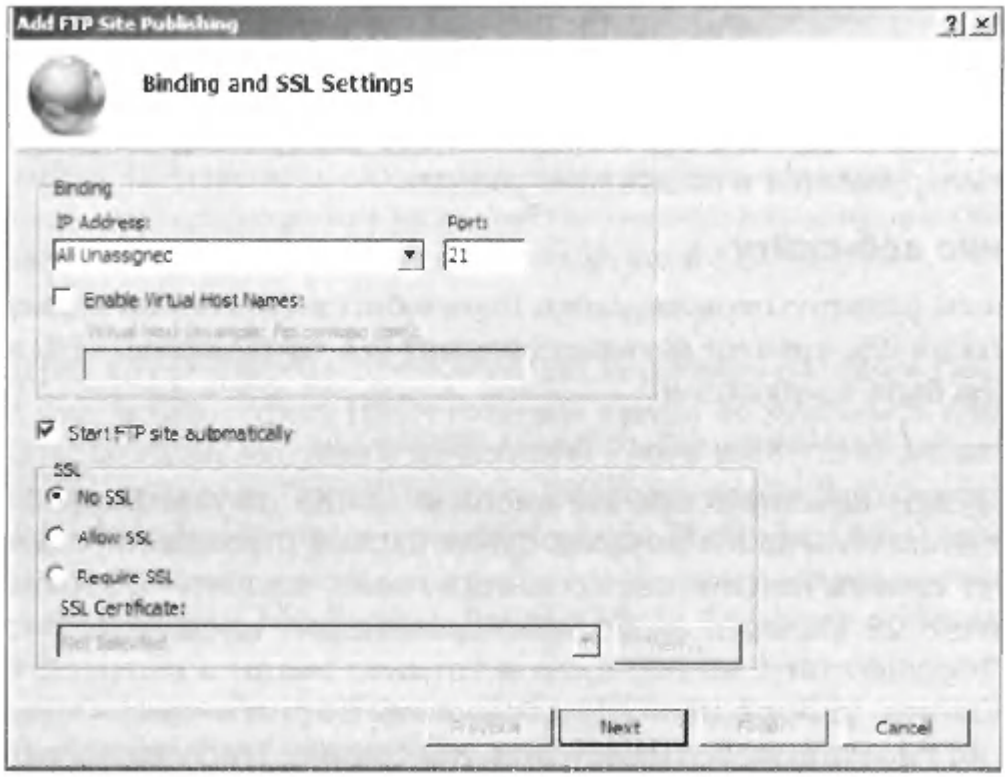

Рисунок 42.14 – Перша сторінка майстра Add FTP Site Publishing Wizard

Під час розгортання потрібно мати доступ для читання і запису, тому відзначені відповідні прапорці. Ваша конфігурація може відрізнятися залежно від політики адміністрування сервера. Для завершення конфігурування клацніть на кнопці Finish (Готово).

При першому розгортанні додатка повинен бути створений цільовий каталог. Причина в тому, що інакше не вийде вказати, що каталог повинен трактуватися як додаток, як це було під час веб-розгортання.

Веб-сайт, який буде використовуватися в прикладі застосування цієї методики розгортання, ідентичний описаному для веб-розгортання, за винятком того, що текст сторінки (єдиної) змінений, як показано на рис. 42.15.

Рисунок 42.15 – Веб-сайт FTP Deploy

Деталі використання FTP для розгортання беспроектного веб-сайту розглядаються в наступному розділі.

Перед першим розгортанням сайту знадобиться створити на сервері каталог призначення і вказати IIS, що цей каталог містить веб-додаток. URL-адреса для цього прикладу повинна бути наступною:

http: //<им'я_сервера>/WebsiteDeploymeint/FTPDeploy

Для цього потрібно натиснути правою кнопкою миші на вузлі Default Web Site в IIS Manager і в контекстному меню обрати пункт Explore (Провідник). Потім у вікні провідника слід створити каталог WebsiteDeployment, а в ньому – FTPDeploy.

Оновіть вікно IIS Manager, розгорніть деревоподібне подання, щоб знайти папку FTPDeploy, клацніть на ній правою кнопкою миші і в контекстному меню виберіть Convert to Application (Перетворити в додаток). Залиште значення, що використовуються за замовчуванням, без зміни. Ми створили папку призначення для розгортання і вказали IIS, що вона буде містити додаток.

Щоб виконати розгортання, відкрийте приклад проекту в Visual Studio 2010 і оберіть пункт Publish FTP_Deploy (Опублікувати FTP_Deploy) з меню Build. Відкриється те ж саме діалогове вікно, яке застосовувалося під час веб-розгортання. У списку Publish Method (Метод публікації) вкажіть FTP. Вид діалогового вікна зміниться, як показано на рис. 42.16.

Формат поля Target Location (Цільове місце розташування) для FTP-методу дещо відрізняється від попереднього прикладу. Він має таку форму:

ftp://<им'я сервера>/цільове_місце_розташування

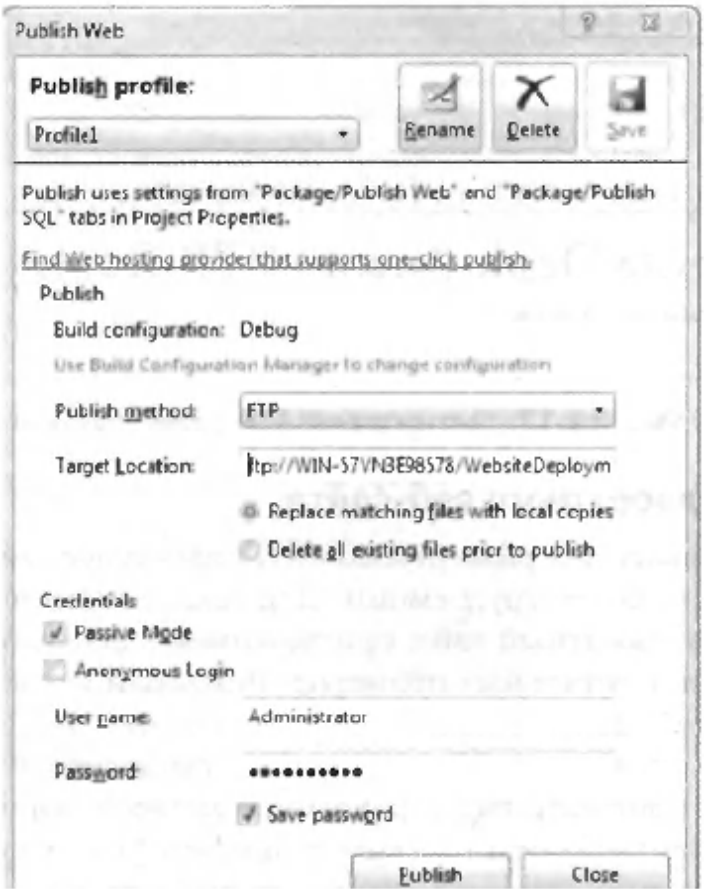

Рисунок 42.16 – Діалогове вікно Publish Web

Елементи ftp і ім'я_сервера очевидні, але важливо відзначити, що ця URL-адреса не містить ім'я сайту. IIS відомо, для якого сайту активізовано FTP-розгортання, і тому цільове_місце_розташування вказується щодо кореневого каталогу цього сайту. Як видно на малюнку, була вказана наступна URL-адреса:

ftp://WIN-57VN3E98578/WebsiteDeployment/FTPDeploy

Ця URL-адреса є відносною для кореневого каталогу сайту DefaultWeb Site та відповідає і URL-адресі HTTP, яку потрібно використовувати для вмісту, та каталогам, які були створені на початку цього розділу. Будьте уважні, обираючи перемикач в розділі Target Location. Швидше за все, потрібно буде вибрати перемикач Replace (Замінити). Вибір перемикача Delete (Видалити) призводить до видалення будь-яких файлів, що існують на сервері, але не є частиною проекту.

Як бачите, був встановлений прапорець Passive Mode (Пасивний режим). Можливо, для своїх розгортувань ви вчините так само, оскільки це підвищує ймовірність успішного розгортання без необхідності зміни конфігурації брандмауерів. І, нарешті, для сервера був вказаний обліковий запис Administrator з відповідним паролем. Тут можна задати й інший обліковий запис, який буде використовуватися під час розгортання сайту. Клацніть на кнопці Publish, щоб приступити до розгортання.

Порада. Одна з найбільш поширених причин виникнення проблем під час використання FTP-розгортання – конфігурація брандмауера. При виникненні проблем перевірте конфігурацію брандмауера Windows на комп'ютерах сервера і клієнта, а також конфігурацію будь-яких фізичних брандмауерів в інфраструктурі, що використовується.

По завершенні розгортання сайт можна перевірити, перейшовши за відповідною URL-адресою. Результат показаний на рис. 42.17.

Рисунок 42.17 – Тестування FTP-розгортання

42.3.4. Розгортання беспроектного веб-сайту

FTP можна використовувати для розгортання беспроектного веб-сайту, хоча цей процес є дещо більш трудомістким. Для демонстрації цього підходу ми створили дуже простий беспроектний сайт, дотримуючись послідовності дій, яка використовувалася в інших прикладах.

Файл Default.aspx.cs містить код, який встановлює текст мітки, відповідної версії .NET Framework, під управлінням якої виконується додаток.

Перед першим розгортанням на сервері повинен бути створений цільовий каталог і перетворений в додаток – так само, як раніше це робилося для FTP-розгортання на основі проекту. Інструкції були дані в попередньому розділі. Для цього прикладу створено каталог FTPProjectlessDeploy всередині каталогу WebsiteDeployment.

Після того як всі описані дії виконані, оберіть пункт Copy Web Site... (Копіювати веб-сайт...) в меню Website, в результаті чого відкриється вкладка Copy Web (Копіювання веб-сайту), показана на рис. 42.18. У лівій частині відображаються файли, що утворюють сайт – незважаючи на наявність всього двох файлів, це вікно дає уявлення про те, як воно може виглядати для більш складного сайту.

Рисунок 42.18 – Вкладка Copy Web

Наступний крок полягає в підключенні до IIS. Клацніть на кнопці Connect (Підключитися) у верхній частині вкладки Copy Web. В результаті відкриється діалогове вікно Open Web Site (Відкриття веб-сайту), наведене на рис. 42.19. Клацнувши на кнопці FTP Site (FTP-сайт) в лівій частині вікна.

Під час заповнення полів використовується та ж інформація, що і під час розгортання сайту на основі проекту. У полі Server (Сервер) потрібно ввести ім'я сервера, в якому буде виконуватися розгортання. За замовчуванням використовується порт 21, що зазвичай встановлюється для FTP. Поле Directory (Каталог) визначає місцеположення, в якому повинно бути розгорнуто вміст, який визначається щодо кореневого каталогу сайту. У нашому випадку розгортання необхідно виконати в каталог WebsiteDeployment/ FTPProjectlessDeploy. Прапорець Passive Mode (Пасивний режим) відзначений за замовчуванням. Рекомендується залишити це без змін, щоб знизити ймовірність виникнення проблем при наявності брандмауерів між робочою станцією і сервером. Зніміть позначч-

ку з прапорця Anonymous Login (Анонімний вхід) і введіть ім'я облікового запису та пароль, що використовуються для розгортання.

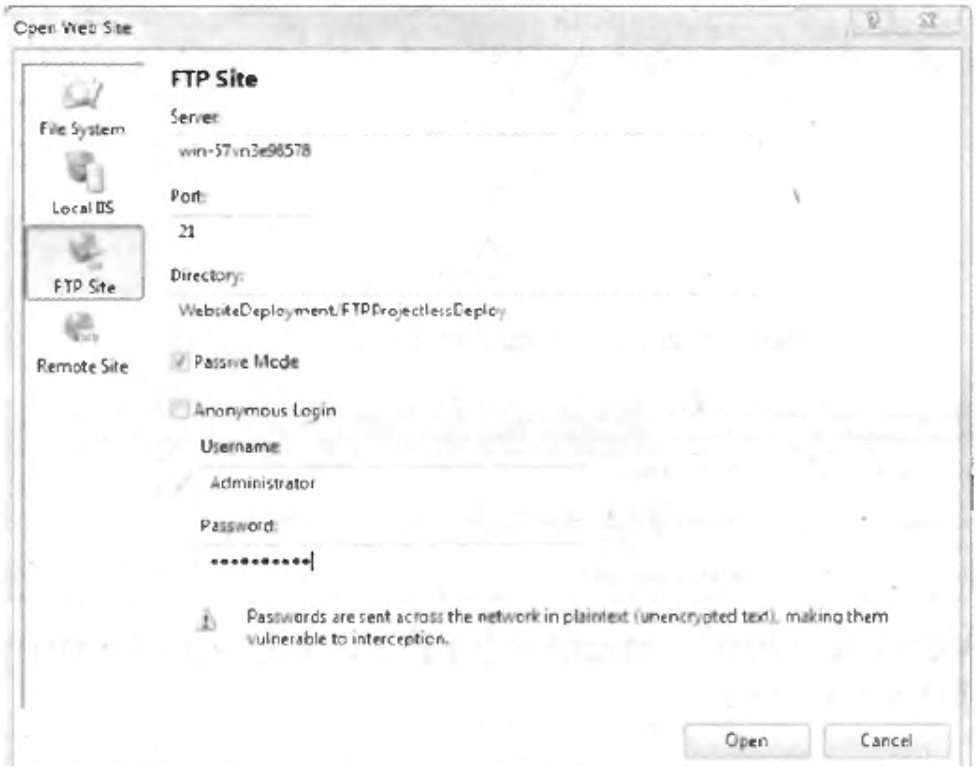

Рисунок 42.19 – Діалогове вікно Open Web Site

Як і в попередніх прикладах, ми використовували обліковий запис Administrator. Після заповнення всіх цих відомостей клацніть на кнопці Open (Відкрити). Тепер права частина вкладки Copy Web буде відображати вміст каталогу, створеного на сервері. Оскільки це розгортання є першим, сервер не містить файлів і права частина вікна порожня.

Оберіть обидва файли в лівій частині вікна. Це призведе до активізації декількох кнопок в центрі вкладки (рис. 42.20).

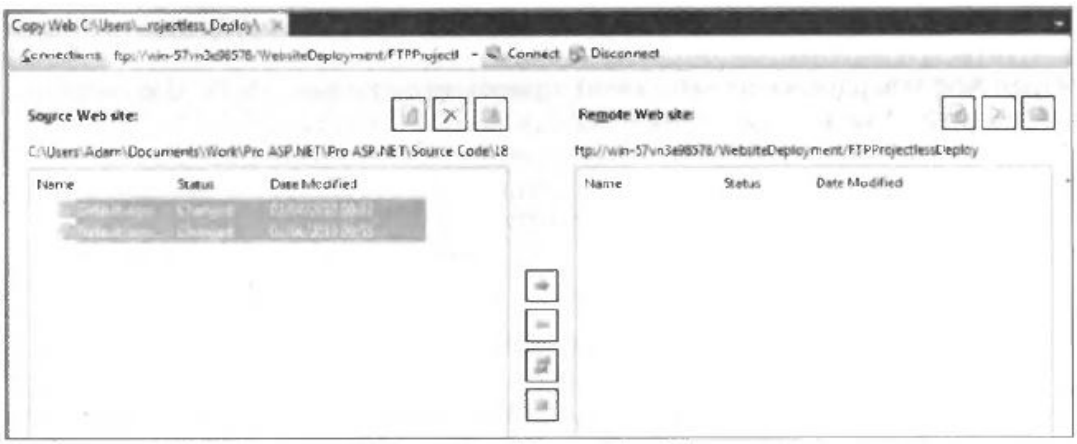

Рисунок 42.20 – Вибір файлів для розгортання

Цей етап потребує особливої уваги – будуть розгорнуті тільки вибрані файли. Пропущений файл не переміщується на сервер, і поведінка сайту буде некоректною.

Клацніть на найвищій кнопці в центрі вкладки – на ту, яка містить стрілку, що вказує праворуч. Це призведе до копіювання файлів на сервер, і вони відобразяться в правій частині вікна. Розгортання можна протестувати, відкривши в браузері цільову URL-адресу. Результат показаний на рис. 42.21.

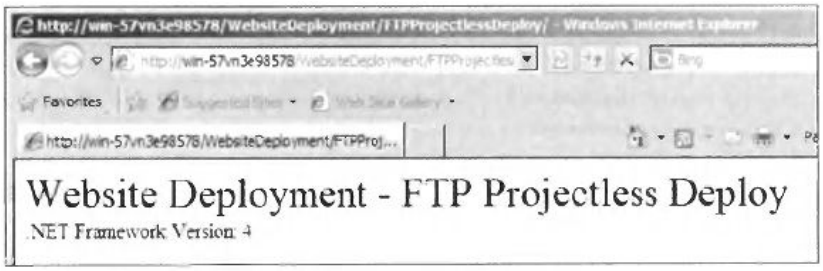

Рисунок 42.21 – Тестування розгортання

Хоча вибір файлів для розгортання вимагає уваги, Visual Studio надає набір корисних підказок. Якщо змінити один з файлів сайту, а потім знову вибрати пункт Copy Web Site ..., зміни будуть помічені стрілкою, як показано на рис. 42.22. Це дуже зручно для синхронізації файлів.

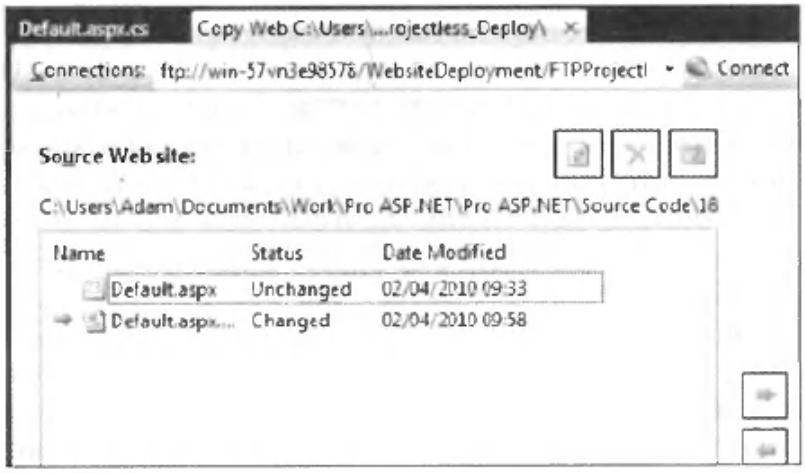

Рисунок 42.22 – Виділені зміни на вкладці Copy Wet

Після розгортання сайтом можна управляти із застосуванням засобів IIS. Нижче розглядаються найбільш корисні опції конфігурації і способи їх використання.

42.4. Створення нового сайту

IIS 7 може підтримувати безліч сайтів на одному сервері. У розглянутих прикладах розгортання вміст додавався до сайту за замовчуванням, а в цьому розділі буде показано, як створити абсолютно новий сайт. Розгорніть деревоподібне подання в IIS Manager, клацніть правою кнопкою миші на вузлі Sites (Сайти) і в контекстному меню оберіть пункт Add Web Site ... (Додати веб-сайт ...). Відкриється діалогове вікно Add Web Site (Додавання веб-сайту), показане на рис. 42.23.

Поле Site name (Ім'я сайту) повинно містити що-небудь значуще. Воно використовується для ідентифікації сайту в середовищі IIS Manager, але не впливає на вміст сайту. У цьому прикладі пул додатків був залишений без

змін (пули додатків розглядаються далі в главі). Поле Physical path (Фізичний шлях) визначає місцеположення, в якому IIS 7 буде шукати вміст для запитів на обслуговування, адресованих новому сайту. У цьому прикладі на сервері був створений новий каталог C:\FileCopySite.

Кнопки Connect as ... (Дод. Як ...) і Test Settings ... (Тест налаштувань ...) дозволяють вказати інші облікові дані користувача для доступу до вмісту сайту.

Розділ Bindings (Прив'язка) дозволяє вказати, як IIS 7 буде прослуховувати запити, що надходять від клієнтів. IIS 7 підтримує безліч протоколів, але ми зосередимо увагу на HTTP, оскільки він використовується найбільш широко. Для цього в списку Type (Тип) оберемо опцію http. Меню IP address (IP-адреса) дозволяє вибрати мережевий інтерфейс, який сервер буде прослуховувати на предмет запитів. Для цього параметра було залишено значення All Unassigned (Всі непризначені) – тобто, IIS буде прослуховувати всі інтерфейси за винятком тих, де інший сайт повинен обслуговуватися через той самий порт TCP. Значення Port (Порт) дозволяє вказати порт TCP, на якому IIS 7 буде прослуховувати запити клієнтів. У загальному випадку кожен сайт повинен обслуговуватися через унікальний порт, тому, щоб уникнути конфліктів з підключеним до порту 80 веб-сайтом за замовчуванням, ми вибрали порт 81.

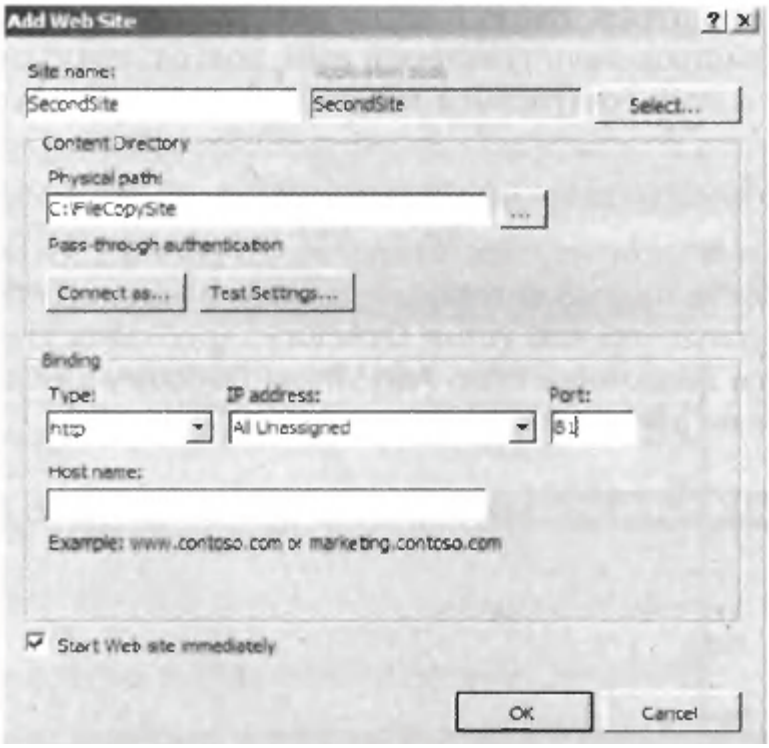

Рисунок 42.23 – Діалогове вікно Add Web Site

Крім того, відмічений прапорець Start Website immediately (Запустити веб-сайт негайно) – тобто відразу після клацання на кнопці ОК сервер IIS створить веб-сайт і почне прослуховувати запити. Більше конфігурувати нічого, тому клацніть на кнопці ОК, щоб створити і запустити веб-сайт. Кожен з розглянутих у цій главі варіантів розгортання дозволяє вказувати сайт для розгортання – пам'ятайте, що під час розгортання сайти розрізняються по іменах і використовують зазначені номери портів.

При установці місця призначення для прикладів веб-сайтів вміст поміщається в каталог, в якому IIS 7 шукає вміст за замовчуванням. Але вміст можна було б розмістити десь в іншому місці, а потім використовувати віртуальний каталог для посилань на нього. Щоб продемонструвати цей підхід, створимо на сервері новий каталог і скопіюємо в нього вміст сайту. Шлях до нового каталогу виглядає наступним чином:

C:\WebsiteDeployment\VirtualDirectory

Щоб зв'язати новий каталог з IIS, відкрийте IIS Manager, розгорніть деревоподібне уявлення, клацніть правою кнопкою миші на елементі Default Web Site і в контекстному меню оберіть пункт Add Virtual Directory...(Додати віртуальний каталог...). В результаті відкриється діалогове вікно Add Virtual Directory (Додавання віртуального каталогу), показане на рис. 42.24.

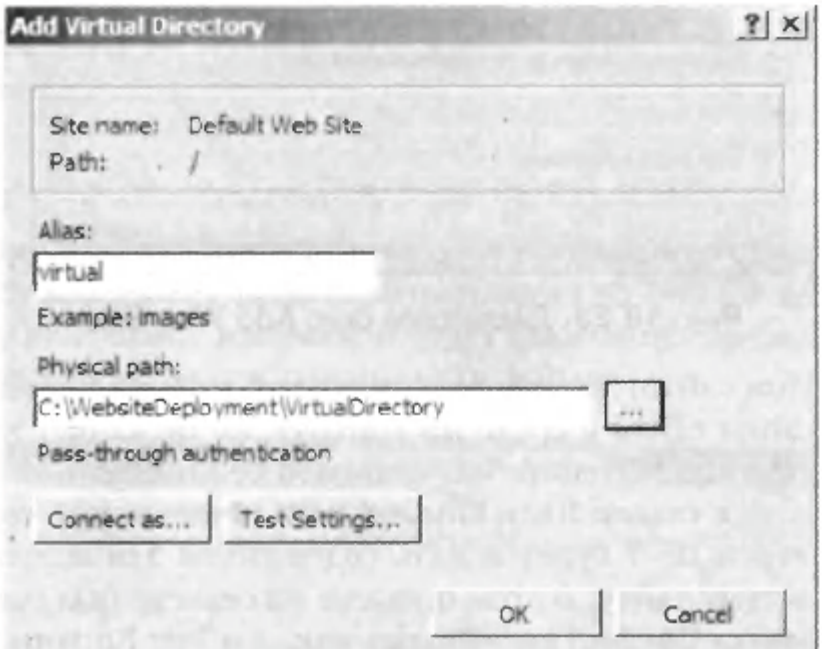

Рисунок 42.24 – Діалогове вікно Add Virtual Directory

URL-адреса цього веб-сайту повинна бути такою:

http://<ім'я_сервера>/virtual

У полі Alias (Псевдонім) введіть virtual. Для шляху був обраний кореневий каталог веб-сайту за замовчуванням – тобто, будь-яке ім'я, введене в поле Alias (Псевдонім), буде додаватися в URL-адресу безпосередньо після імені сервера. У полі Physical path (Фізичний шлях) введіть шлях до одного з створених раніше каталогів розгортання. Клацніть на кнопці ОК, щоб створити віртуальний каталог. Щоб протестувати його, відкрийте браузер на сервері і направте його на URL-адресу http: // localhost / virtual. Як і раніше, відкриється створений нами простий веб-сайт, але на цей раз вміст буде вилучатись з нового каталогу, а доступ до нього буде здійснюватися за допомогою зазначеної спеціальної URL-адреси.

Клас VirtualPathProvider пропонує альтернативний підхід до роботи з віртуальними каталогами, під час якого вміст може генеруватися програмно або з бази даних, а не вилучатись з файлової системи.

Найкращий спосіб зрозуміти можливості класу VirtualPathProvider – розглянути невеликий приклад. Ми продемонструємо, як створити простий клас VirtualPathProvider, який може читати файли ASPX з таблиці SQL Server.

Для цього необхідна таблиця бази даних, подібна показаній на рис. 42.25, яка зберігає три сторінки.

Як бачите, таблиця включає ім'я файлу (яке є первинним ключем) і вміст. Тип коду вмісту може бути будь-яким, зрозумілим ASP.NET; стосовно до розглянутого прикладу, оскільки ми маємо намір обслуговувати прості сторінки, вміст може бути будь-яким кодом, який скомпілюється аналізатором сторінок.

Клас VirtualPathProvider визначений у просторі імен System.Web.Hosting. Просто додайте новий клас в каталог App_Code і успадкує його від класу VirtualPathProvider.

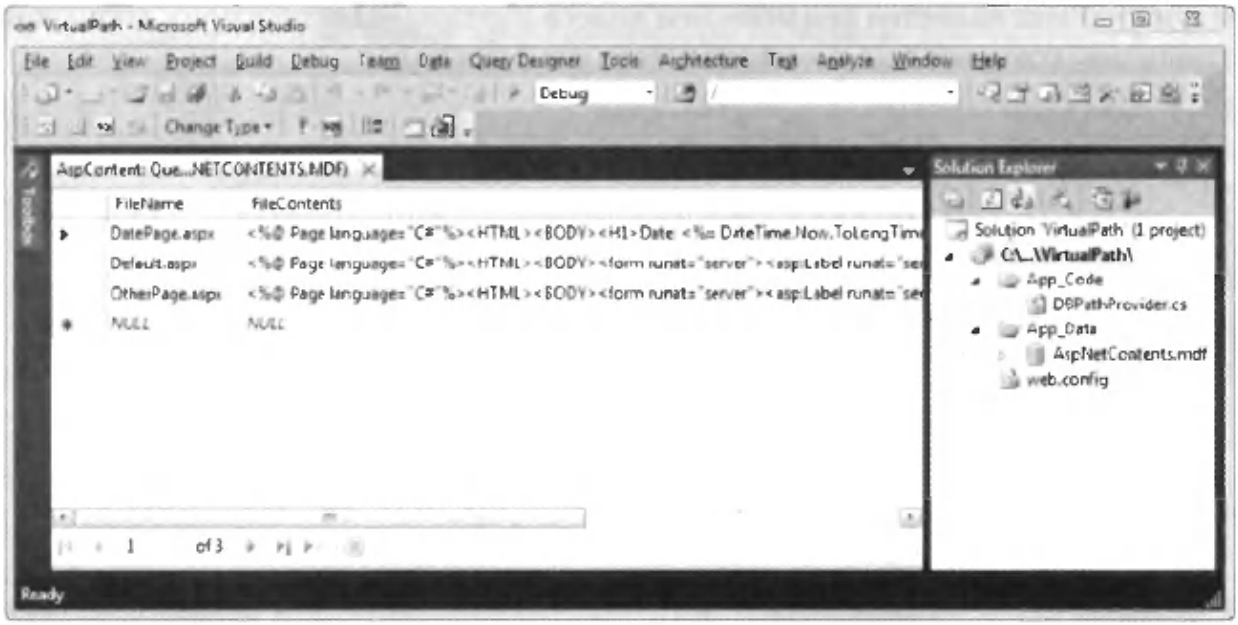

Рисунок 42.25 – База даних SQL Server, що використовується для класу VirtualPathProvider

Клас повинен реалізувати як мінімум такі методи:

```
using System;
using System.Data.SqlClient;
using System.10;
using System.Text;
using System.Web.Hosting;
public class DBPathProvider : VirtualPathProvider {
public static void Applnitialize()
{
HostingEnvironment.RegisterVirtualPathProvider(new DBPathProvider()) ;
}
public override bool FileExists(string virtualPath)
{ throw new Exception("The method or operation is not implemented.");
// Метод або операція не реалізована.
}
```

```
public override VirtualFile GetFile (string virtualPath) {
throw new Exception ("The method or operation is not implemented.");
// Метод або операція не реалізована
}
}
```

Постачальник повинен реалізувати статичний метод Applnitialize, в якому потрібно створити екземпляр класу і зареєструвати його всередині платформи в якості постачальника. Метод FileExists служить для перевірки того, чи може шлях обслуговуватися постачальником, а метод GetFile викликається для отримання вмісту для шляху, повертаючи екземпляр абстрактного класу VirtualFile – ніяких конкретних реалізацій класу VirtualFile не існує. Іншими словами, для забезпечення підтримки постачальника знадобиться розширити абстрактний клас.

Наступний код є реалізацією класу VirtualFile, що супроводжує нашого постачальника, який був поміщений в цей же файл коду:

```
public class DBVirtualFile : VirtualFile {
private string _FileContent;
public DBVirtualFile(string virtualPath, string fileContent): base(virtualPath)
{ FileContent=fileContent;
}
public override Stream Open() { Stream stream=new MemorvStream();
StreamWriter writer = new StreamWriter(stream, Encoding.Unicode);
writer.Write(_FileContent);
writer.Flush ();
stream.Seek(0, SeekOrigin.Begin);
return stream;
}
}
```

Конструктор класу отримує віртуальний шлях і вміст файлу. У методі Open рядок вмісту зберігається в об'єкті MemoryStream, який повертається як результат методу. ASP.NET використовує потік для читання вмісту так, як якщо б воно було прочитано з файлової системи – це можливо завдяки абстрагування байтів за допомогою класів Stream.

Щоб завершити клас свого постачальника, в нього необхідно додати підтримку для вилучення вмісту з бази даних. Якщо файл не існує в базі даних постачальник просто направляє запит попередньому постачальнику (який був обраний інфраструктурою під час реєстрації в статичному методі Applnitialize). Додайте метод для отримання вмісту з бази даних в клас DBPathProvider:

```
private string GetFileFromDB (string virtualPath) {
string contents;
string fileName = virtualPath.Substring (virtualPath.IndexOf ('/', 1) +1);
//Читати файл з бази даних.
SqlConnection conn = new SqlConnection ();
```

```
conn.ConnectionString = "Data Source = .WSQLEXPRESS; Initial Catalog = \"ASPNETCONTENTS\"; Integrated Security = True"; conn.Open ();
 try {
SqlCommand cmd = new SqlCommand ("SELECT FileContents FROM AspContent " +"WHERE FileName = @ fn", conn);
cmd.Parameters.AddWithValue ("@ fn", fileName);
contents = cmd.ExecuteScalar () as string;
if (contents == null )
contents = string.Empty;
} catch {
contents = string.Empty;
} finally {
conn.Close ();
}
return contents;
}
```

Функція GetFileFromDB витягує ім'я файлу з віртуального шляху, а потім зчитує відповідний вміст з бази даних. Потім цей метод використовується обома методами FileExists і Get File, як показано в наступному фрагменті коду:

```
public override bool FileExists (string virtualPath) {
string contents = this.GetFileFromDB (virtualPath);
if (contents.Equals (string.Empty)) {
return false;
} else {
return true;
}}
public override VirtualFile GetFile (string virtualPath) {
string contents = this.GetFileFromDB (virtualPath);
if (contents.Equals (string.Empty)) {
return Previous.GetFile (virtualPath);
} else {
return new DBVirtualFile (virtualPath, contents);
}
}
```

В постачальнику можна реалізувати кілька додаткових методів, які можуть бути корисними для більш складних моделей, таких як перевірка існування каталогу (DirectoryExists), обчислення хешу файлу (GetFileHash) і виконання перевірки кеша (GetCacheDependency). Після того, як основні функції визначені, постачальника можна протестувати. На рис. 42.26 показані три вікна браузера, в кожному з яких відображається одна зі сторінок, отримана з бази даних.

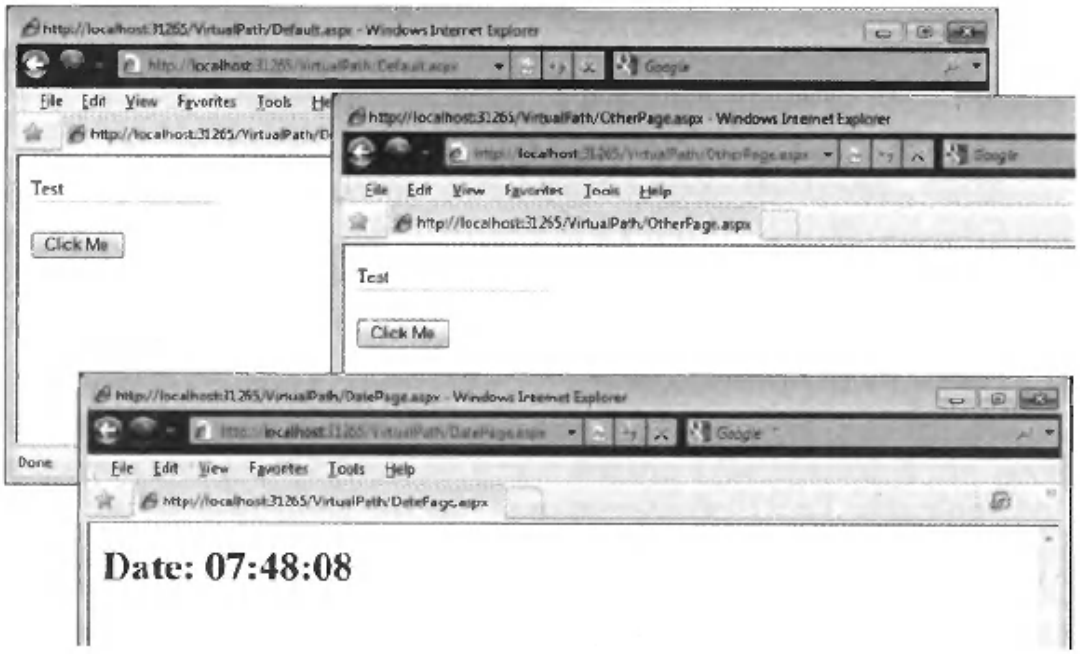

Рисунок 42.26 – Клас VirtualPathProvider в дії

42.5. Використання пулів додатків

Пули додатків дозволяють для спрощення конфігурації і управління групувати разом аналогічні або пов'язані додатки. При цьому додатки, які включені в пули додатків, ізолюються, в результаті чого проблеми, що виникають в одному пулі, не роблять впливу на додатки з інших пулів. Не існує ніяких жорстких і коротких правил призначення додатків в пули. Додатки можна групувати за принципом схожості профілів робочих характеристик, приналежності одному підрозділу або будь-яким іншим принципом, виправданому в конкретному середовищі. Як буде показано, однією з найбільш корисних засобів є можливість наявності різних пулів додатків, які використовують різні версії .NET Framework. У цьому розділі ми розглянемо, як створювати, конфігурувати і призначати додатки в пули додатків.

IIS 7 автоматично створює набір пулів додатків, у тому числі пул, що використовується за замовчуванням під час створення нового додатку. Переглядати і управляти пулами додатків можна за допомогою IIS Manager – для цього достатньо розгорнути елемент сервера в деревовидному уявленні і клацнути на елементі Application Pools (Пули додатків). В результаті відобразяться пули, визначені на сервері. Пули, визначені на нашому комп'ютері Windows Server 2008 R2, показані на рис. 42.27.

Пули додатків перераховані в таблиці в середині екрана. Стовпці цієї таблиці містять найбільш важливі характеристики пулів, описані в табл. 42.1. Під час розгортання додатка воно призначається в пул додатків, що використовується за замовчуванням. Під час розгортання програми в попередніх прикладах налаштування пулу додатків за умовчанням були змінені так, щоб він використовував .NET 4.

Рисунок 42.27 – Стандартні пули додатків

42.5.1. Створення нового пулу додатків

Нестандартний пул додатків можна створити, клацнувши на дії Add Application Pool (Додати пул додатків) в правій частині екрана IIS Manager. Відкриється діалогове вікно Add Application Pool (Додавання пулу додатків), показане на рис. 42.28. Введіть ім'я нового пулу (у прикладі використано ім'я CustomAppPool), оберіть версію .NET Framework, яка буде застосовуватися для запуску додатків, призначених у пул, і необхідний режим керованого конвеєра (режим Classic (Класичний) призначений для успадкованих додатків; якщо впевненості немає, оберіть режим Integrated (Вбудований)).

Таблиця 42.1. Характеристики пулів додатків, що відображаються в головному вікні IIS Manager

Стовпчик	**Опис**
Name (Ім’я)	Визначає ім'я пулу додатків. Після того як пул створений, його ім'я змінити не можна
Status (Стан)	Цей стовпець показує, чи виконується пул додатків, тобто чи буде генеруватися відповідь на запити до додатків, призначеним в пул. Детальніше це описано в розділі "Запуск і зупинка пулу додатків"
.NET Framework Version (Версія .NET Framework)	Версія .NET Framework, яка буде використовуватися для виконання керованого коду – в прикладах розгортання ця настройка пулу додатків за умовчанням була змінена. Більш докладно вказівка версій платформи описана в розділі „Використання паралельного виконання”
Managed Pipeline Mode (Режим керованого конвеєра)	IIS 7 підтримує два режими конвеєра для обробки запитів Integrated (Вбудований) і Classic (Класичний). Розширення конвеєра докладніше розглядається в розділі "Розширення вбудованого конвеєра"
Identity (Ідентифікація)	Обліковий запис Windows, що використовується для запуску додатків пулу
Applications (Кількість додатків)	Кількість додатків, призначених у пул; на рис. 42.28 видно, що пул DefaultAppPoo містить три додатки

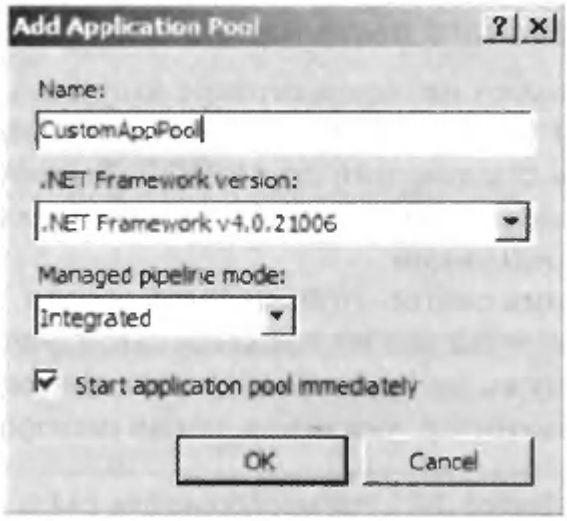

Рисунок 42. 28 Діалогове вікно Add Application Pool

Клацніть на кнопці ОК, і новий пул буде створений і доданий в список IIS Manager. Клацання на дії Advanced Settings... (Додаткові параметри...) дозволить конфігурувати деталі, пов'язані з пулом.

42.5.2. Призначення програми в пул додатків

Щоб призначити додаток в пул додатків, оберіть додаток у вікні IIS Manager і клацніть на дії Basic Settings (Основні налаштування) в правій частині екрана. Відкриється діалогове вікно Edit Application (Зміна додатка). Клацніть на кнопці Select (Вибрати) і оберіть пул додатків із списку, як показано на рис. 42.29. Ми обрали спеціальний пул додатків, створений в попередньому розділі.

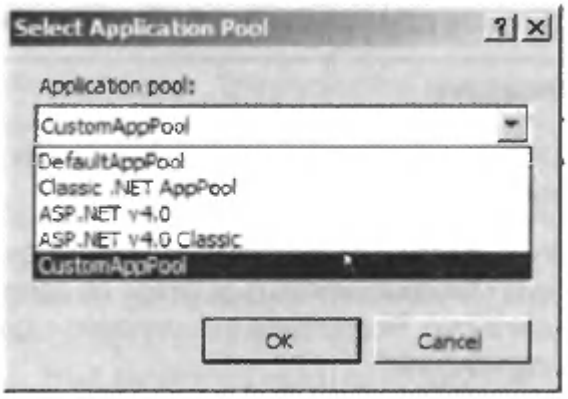

Рисунок 42.29 – Діалогове вікно Select Application Pool

Клацніть на кнопці ОК. Клацніть на елементі Application Pools в IIS Manager – кількість додатків пулу CustomAppPool в стовпці Applications стане рівним 1, а кількість додатків пулу DefaultAppPool зменшиться на одиницю.

42.5.3. Запуск і зупинка пулу додатків

Після клацання на пулі додатків в правій частині вікна IIS Manager в розділі Application Pool Tasks (Завдання пулу додатків) відобразяться три дії. Дії Start (Початок) і Stop (Зупинити) визначають те, чи обслуговуються запити, адресовані призначеним в пул додатком. Якщо пул зупинений, клієнти

будуть отримувати повідомлення про помилку. Дія Recycle (Перезапуск) переустановлює пул додатків. Це корисно для усунення важких для діагностування проблем, що поступово накопичуються.

Контрольні запитання

1. Що таке local host?
2. Як розгорнути веб-сайт з середовища Visual Studio?
3. Який найпростіший спосіб установки конфігурації FTP-розгортання?
4. Для чого використовуються пули додатків?
5. Що таке сайт?
6. Для чого застосовується елемент Locator?
7. Яка основна перевага FTP-розгортання?
8. Які можливості класу VirtualPathProvider?
9. Для чого використовується функція GetFileFromDB?
10. Який найпростіший спосіб підготовки IIS до застосування розігріву додатків?

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Культин Н. Б. C# в задачах и примерах. – СПб.: БХВ-Петербург, 2009. – 320 с.

2. Клімов Л. П. С #. Поради програмістам. - СПб.: БХВ-Петербург, 2008. – 544 с

3. М. Мак-Дональд, А. Фример, М. Шпушта. Microsoft ASP.NET 4 с примерами на C# 2010 для профессионалов, 4-е изд. – М.: Вильямс, 2011. – 1424с.

4. Д. В. Столбовский. Основы разработки Интернет и веб-приложений, М.: Вильямс, 2010. – 256с.

5. М. Беллиньясо. Разработка Веб-приложений в среде ASP.NET. – М.: Вильямс, 2007. – 640с.

6. Эспозито Д. Microsoft ASP.NET 2.0. Углубленное изучение. / Пер. с англ. — М.: Издательство «Русская Редакция»; СПб.:Питер, 2008. — 592 с.

7. Основы ASP.NET. ИНТУИТ, 2007. – 238 с.

Навчальне видання

Конспект лекцій з дисципліни „Технології та засоби створення програмного забезпечення АСУ” 4 курс 7 семестр для студентів напряму 6.050103 „Програмна інженерія” / К.М. Ялова, В.В. Завгородній // Дніпродзержинськ: ДДТУ, 2013. – 176 с.

Укладачі: к.т.н., ст.викл. Ялова К.М.
ст. викл. Завгородній В.В.

Підписано до друку
Формат Обсяг др. ар.
Тираж екз. Заказ №

м. Дніпродзержинськ,
вул. Дніпробудівська ,2.