

Учреждение образования  
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ  
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

**Кафедра информационных систем и технологий**

# **ОРГАНИЗАЦИЯ WEB-ПОРТАЛА И АДМИНИСТРИРОВАНИЕ РЕСУРСОВ В WEB**

**Методические указания к выполнению курсовых работ  
для студентов специальности 1-40 01 02-03  
«Информационные системы и технологии  
(издательско-полиграфический комплекс)»  
заочной формы обучения**

Минск 2012

УДК 004.436.4(075.8)

ББК 32.973–01я73

О-64

Рассмотрены и рекомендованы редакционно-издательским советом университета

Составители:

*Д. М. Романенко, Ю. О. Булова*

Рецензент

кандидат технических наук, доцент, заведующий кафедрой полиграфического оборудования и систем обработки информации  
Белорусского государственного технологического университета

*М. С. Шмаков*

По тематическому плану изданий учебно-методической литературы университета на 2012 год. Поз. 209.

Для студентов специальности 1-40 01 02-03 «Информационные системы и технологии (издательско-полиграфический комплекс)» заочной формы обучения.

© УО «Белорусский государственный технологический университет», 2012

# ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ .....	5
1. ОРГАНИЗАЦИЯ ВЫПОЛНЕНИЯ КУРСОВОЙ РАБОТЫ.....	6
1.1. Этапы и сроки выполнения курсовой работы.....	6
1.2. Элементы и содержание курсовой работы.....	6
1.3. Темы курсовой работы .....	9
2. СТРУКТУРА И ПОСЛЕДОВАТЕЛЬНОСТЬ ВЫПОЛНЕНИЯ КУРСОВОЙ РАБОТЫ.....	10
2.1. Цель работы .....	10
2.2. Состав и последовательность работ.....	10
2.3. Требования к web-сайту .....	11
3. РУКОВОДСТВО ПО РАЗРАБОТКЕ ДИНАМИЧЕСКИХ САЙТОВ.....	12
3.1. Архитектура ASP.NET.....	12
3.2. Создание страницы web-приложения .....	15
3.3. Взаимодействие элементов web-приложения с клиентом.....	19
3.4. Элементы управления и их свойства .....	21
3.4.1. Элемент управления TextBox .....	21
3.4.2. Элемент управления Label .....	23
3.4.3. Элемент управления HyperLink.....	24
3.4.4. Элемент управления Image .....	24
3.4.5. Элемент управления CheckBox .....	25
3.4.6. Элемент управления RadioButton.....	26
3.4.7. Элемент управления Table .....	26
3.4.8. Элемент управления Panel .....	28
3.4.9. Кнопки.....	28
3.4.10. Списки.....	29
3.4.11. Динамическое создание элементов управления .....	32
3.5. Проверка вводимых данных .....	35
3.6. Работа с базами данных.....	39
3.7. Дизайн страниц.....	43
4. ПУБЛИКАЦИЯ САЙТОВ.....	46
4.1. Установка Microsoft IIS 6.0 .....	46
4.2. Публикация сайта, разработанного на платформе ASP с использованием языка C#.....	50
5. ТЕСТИРОВАНИЕ WEB-ПРИЛОЖЕНИЯ.....	57

5.1. Основные типы тестов в Visual Studio .....	57
5.2. Создание теста web-производительности.....	58
5.3. Нагрузочное тестирование .....	61
5.4. Запуск и анализ результатов выполнения нагрузочных тестов .....	69
6. ТРЕБОВАНИЯ К ОФОРМЛЕНИЮ ПОЯСНИТЕЛЬНОЙ ЗАПИСКИ .....	71
ЗАКЛЮЧЕНИЕ .....	77
СПИСОК РЕКОМЕНДУЕМЫХ ИСТОЧНИКОВ .....	78
ПРИЛОЖЕНИЕ 1 .....	79
ПРИЛОЖЕНИЕ 2 .....	80

# ПРЕДИСЛОВИЕ

World Wide Web – распределенная система, предоставляющая доступ к связанным между собой документам, расположенным на различных компьютерах, подключенных к Интернету. Всемирную паутину образуют миллионы web-серверов. Большинство ресурсов сети Интернет представляет собой гипертекст. Гипертекстовые документы, размещаемые во всемирной паутине, называются web-страницами. Несколько web-страниц, объединенных общей темой, дизайном, а также связанных между собой ссылками и обычно находящихся на одном и том же web-сервере, называются web-сайтом. Для загрузки и просмотра web-страниц используются специальные программы – браузеры.

Динамический web-сайт – это сайт, страницы которого основаны на шаблонной странице, в которую вставляется постоянно меняющееся информационное наполнение, которое обычно хранится в базе данных. Когда пользователь запрашивает страницу, соответствующая информация извлекается из базы, вставляется в шаблон, образуя новую web-страницу, и пересылается web-сервером в пользовательский браузер, который и отображает ее должным образом. Кроме информационного наполнения, динамически могут создаваться и элементы навигации по web-сайту.

Редактирование страницы-шаблона и содержимого может производиться как средствами самого сайта, так и с применением стороннего программного обеспечения. Возможность править все страницы предоставляется только определенной категории пользователей, например администраторам или зарегистрированным пользователям.

В издании представлен для изучения студентами развивающийся раздел web-программирования, ориентированный на разработку динамических web-сайтов. Методические указания предназначены для обучения студентов методам и технологиям разработки динамических web-сайтов, активно использующих базы данных. Программа построена таким образом, что обучение базируется на знаниях по созданию статических HTML-сайтов, полученных в рамках курса «Информационные технологии». По окончании обучения создание и программирование сайтов не составит больших сложностей.

# 1. ОРГАНИЗАЦИЯ ВЫПОЛНЕНИЯ КУРСОВОЙ РАБОТЫ

## 1.1. Этапы и сроки выполнения курсовой работы

Тематика курсовых работ (КР) представлена отдельным списком (см. подраздел 1.3). Курсовая работа разрабатывается студентами заочной формы обучения в сроки, предусмотренные графиком учебного процесса. Законченную курсовую работу (web-сайт и пояснительную записку) студент должен **представить на кафедру до начала сессии**.

После проверки преподавателем, при условии отсутствия замечаний, требующих устранения, КР до защиты остается на кафедре. Если преподаватель указал замечания, требующие устранения, студент должен устранить их до защиты работы.

## 1.2. Элементы и содержание курсовой работы

Общий объем работы 30–60 страниц машинописного текста (без приложений). Приложения формируются при необходимости, если имеющиеся таблицы, рисунки, расчеты затрудняют восприятие материала курсовой работы.

Структурными элементами курсовой работы являются:

- титульный лист;
- задание на курсовую работу;
- реферат;
- содержание;
- введение;
- основная часть;
- заключение;
- библиографический список;
- приложение.

**Титульный лист** является первой страницей курсовой работы, оформляется в соответствии с требованиями кафедры и не нумеруется.

**Содержание** включает введение, наименование всех разделов, подразделов, пунктов (если они имеют наименование) основной части, заключение, библиографический список и приложения с указанием номеров страниц, с которых начинаются эти элементы работы.

**Введение** должно содержать оценку современного состояния предметной области, с которой связано выполнение курсовой работы. При раскрытии содержания проблем предметной области, как правило, используются соответствующие государственные целевые программы и концепции социально-экономического развития страны. Указываются методы научного поиска, дается обзор использованных источников и литературы.

Затем обосновывается актуальность темы и формулируется цель курсовой работы, а также комплекс взаимосвязанных задач, подлежащих решению.

**Основная часть** включает четыре раздела:

1. Теоретические основы предметной области.
2. Web-дизайн сайта. Разработка базы данных.
3. Программная реализация web-сайта.
4. Тестирование и публикация web-сайта.

В зависимости от особенностей работы основную часть излагают в виде сочетания текста, таблиц, формул, иллюстраций и листингов программного кода разработанного сайта. Основную часть следует делить на разделы, подразделы и пункты (при необходимости).

*Первый раздел* работы носит теоретический характер. В нем рассматриваются платформы, а также их инструментарии для создания web-сайтов. Описываются их сильные и слабые стороны. Обосновывается выбор одной из них, позволяющей эффективно реализовать поставленные задачи.

*Во втором разделе* излагается web-дизайн сайта, который состоит из нескольких этапов.

*Первый этап* создания сайта – аналитический. На этом этапе анализируются задачи, которые следует учитывать при создании web-дизайна, изучаются потенциальные посетители web-сайта. Затем посетители сайта делятся на различные категории, определяются цели и сценарии посещения сайта каждой группой посетителей. Это крайне важный этап, без которого невозможна дальнейшая работа по созданию сайта.

*Второй этап.* На основе данных аналитики, полученных на первом этапе работ, начинается разработка концепции и структуры ресурса – основа дальнейших работ по созданию web-дизайна и текстового наполнения сайта. При разработке сайта очень важным моментом является разработка правильной концепции. Концепция сайта представляет собой совокупность обозначенных целей, сформулированных в емкой, интересной форме, удобной для изучения пользователем. Эффективная концепция – залог того, что создание сайта в це-

лом и web-дизайна в частности будет корректным с точки зрения предстоящих задач. Структура ложится в основу дизайна web-сайта и должна учитывать потребности и предпочтения целевой аудитории, следовать ее поведенческим мотивациям и сценариям для каждой группы пользователей. Логически обоснованная структура – необходимый элемент, без которого невозможно разработать качественный дизайн и текстовое наполнение web-сайтов. Дизайн для web-сайтов, при разработке которого не учитываются логические связки в структуре, в дальнейшем имеет проблемы с количеством пользователей и качеством навигации.

*Третий этап* создания сайта – непосредственно web-дизайн. Предполагает соблюдение ряда обязательных требований:

1. Максимально удобный интерфейс: логика, структура и система навигации сайта.

2. Уникальное, яркое графическое решение: при этом web-дизайн сайта должен быть разработан с ориентацией не на эстетические взгляды создателей, а на вкусовые предпочтения целевой аудитории.

3. Web-дизайн не должен мешать восприятию и читабельности текста.

4. Web-дизайн должен быть разработан таким образом, чтобы, несмотря на все красоты, не увеличивать скорость загрузки страниц сайта.

Без соблюдения этих правил невозможна разработка эффективного web-дизайна.

*Третий раздел* представляет собой описание программной реализации web-сайта. В силу того что описать весь сайт не представляется возможным, необходимо уделить внимание, прежде всего, описанию программной реализации ключевых элементов.

*Четвертый раздел* должен быть посвящен тестированию сайта. В ходе тестирования проверяются программные модули, интерфейс, текстовое наполнение и web-дизайн, все те компоненты, от которых зависит эффективность работы ресурса. Создание сайтов – многоступенчатый и многокомпонентный процесс, и тестирование помогает проверить, все ли этапы были качественно реализованы. В случае выявления слабых мест, неудобных логических связок, программных сбоев проводится корректировка, меняется текст, web-дизайн и т. д. Успешное тестирование – необходимый показатель, без которого невозможно оценить, насколько грамотно было выполнено создание сайта.

В **заключении** формулируются краткие выводы по проделанной работе, дается оценка степени выполнения поставленных задач, полученных проектных разработок.

**Библиографический список** должен содержать сведения об источниках, использованных при выполнении КР. Порядок составления и оформления библиографического списка приведен далее.

В **приложение** рекомендуется включать материалы, связанные с выполнением курсовой работы (таблицы, графики, рисунки), если они затрудняют восприятие материала, а также листинг кода разрабатываемого web-сайта.

Рекомендации по разработке сайта, его информационное обеспечение подробно описаны в разделах 2 и 3 методических указаний к выполнению курсовых работ по дисциплине «Информационные технологии» (Информационные технологии. Разработка web-сайта на основе HTML с использованием JavaScript. Сост. Н. А. Жиляк). Создание статических web-сайтов на языке HTML также рассмотрено в вышеупомянутом пособии (раздел 4). Поэтому в дальнейшем рассмотрим лишь особенности создания динамических сайтов с использованием технологии ASP, а также языка C#.

### **1.3. Темы курсовой работы**

Примерный перечень основных тем курсовой работы.

1. Разработка сайта интернет-магазина.
2. Разработка сайта-визитки компании.
3. Разработка информационного web-портала.
4. Разработка сайта-файлообменника.
5. Организация форума.
6. Разработка поискового портала.
7. Разработка фотосайта.
8. Организация музыкального сайта.
9. Организация мультимедийного сайта.
10. Организация электронной библиотеки.
11. Организация игрового web-сайта.
12. Разработка файлообменного сайта типа «torrent».
13. Разработка промо-сайта.
14. Разработка тематического сайта для организации.
15. Разработка сайта новостей.

Необходимо отметить, что любая из тем может быть уточнена в соответствии с содержательной составляющей web-сайта.

## **2. СТРУКТУРА И ПОСЛЕДОВАТЕЛЬНОСТЬ ВЫПОЛНЕНИЯ КУРСОВОЙ РАБОТЫ**

### **2.1. Цель работы**

Целью работы является изучение основ создания динамических сайтов на платформе ASP (язык C#), отработка процесса опубликования сайта с использованием web-сервера IIS. В процессе выполнения курсовой работы студенты должны разработать структуру web-сайта (в зависимости от тематики структура сайта может отличаться), наполнить его содержательной информацией, создать соответствующий программный модуль с использованием современных инструментальных средств, реализовать поиск данной информации в рамках сайта. Вся ключевая информация должна храниться в базе данных.

### **2.2. Состав и последовательность работ**

Состав и последовательность работ следующие:

- изучение темы работы и назначения web-сайта, предметной области;
- сбор содержательной информации по теме курсовой работы;
- анализ и выбор инструментальных средств реализации web-сайта;
- разработка дизайн-макета главной и остальных страниц сайта, решение вопросов о применении фреймовых структур, графических средствах, мультимедийных возможностях.
- разработка состава и структуры web-сайта, разработка структуры базы данных, определение содержания каждой страницы, проектирование системы навигации (взаимосвязь информационных блоков и гиперссылок);
- определение порядка сопровождения и обновления страниц;
- создание web-сайта с помощью выбранного инструментального средства, заполнение базы данных соответствующей информацией;
- разработка подсистемы поиска информации в рамках сайта (с использованием запросов к базе данных);
- изучение исходного кода страниц, написание комментариев;
- публикация web-сайта;
- тестирование web-сайта;
- оформление курсовой работы в соответствии с подразделом 1.2, защита работы с демонстрацией результатов работы.

### 2.3. Требования к web-сайту

К web-сайту предъявляются следующие требования:

1. Web-сайт должен быть выполнен в едином стиле.
2. Web-сайт должен корректно работать в браузерах Opera, Mozilla Firefox, Google Chrome и Internet Explorer.
3. Время загрузки каждой страницы должно быть минимальным.
4. Каждая страница должна содержать удобные и понятные средства навигации по сайту в различных направлениях.
5. Применение фреймов, списков, таблиц, графики, средств мультимедиа должно быть разумным и соответствовать тематическому направлению сайта.
6. На сайте должна присутствовать развитая подсистема поиска информации по сайту.
7. На сайте может быть реализована возможность регистрации и авторизации пользователей, разграничение прав между гостевыми посетителями и зарегистрированными пользователями.
8. Работоспособность web-сайта должна быть протестирована одним из известных способов.

## 3. РУКОВОДСТВО ПО РАЗРАБОТКЕ ДИНАМИЧЕСКИХ САЙТОВ

В данном разделе будут рассмотрены основы создания сайтов на языке С# (для платформы ASP) и PHP. Отметим, что предполагается наличие у студентов базовых знаний по созданию сайтов на языке разметки HTML. В динамическом web-сайте обязательным является активное использование баз данных для хранения основного контента сайта. Также должен быть предусмотрен поиск информации в пределах сайта.

ASP.NET – это платформа, которая является частью инфраструктуры .NET Framework и предназначена для создания web-приложений и web-сервисов, работающих под управлением IIS. Web-приложения, созданные с использованием ASP.NET, построены на архитектуре «клиент – сервер»: приложение находится и выполняется на сервере IIS, который принимает и обрабатывает запросы клиентов, формирует страницу на языке HTML и передает ее клиенту.

В файлах ASP.NET используется код на таких языках программирования, как С#, JScript.NET, VisualBasic.NET, что позволяет применять непосредственно в web-приложениях возможности объектно-ориентированного программирования.

### 3.1. Архитектура ASP.NET

Для рассмотрения архитектуры web-приложения, созданного на ASP.NET, создадим простейшее приложение. Для этого у Вас должен быть установлен Microsoft Visual Studio 2010 или Microsoft Visual Web Developer 2010 Express (принцип работы в обоих приложениях одинаков).

При запуске Microsoft Visual Web Developer 2010 появляется начальная страница (рис. 3.1).

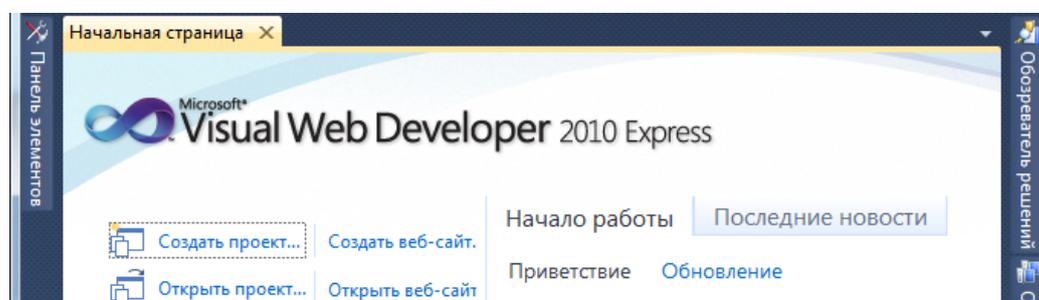


Рис. 3.1. Начальная страница Microsoft Visual Web Developer 2010

Для создания проекта на начальной странице необходимо перейти по ссылке **Создать проект** (Create Project) либо выполнить команду **Файл → Создать проект** (File → Create Project), после чего появится окно создания проекта (рис. 3.2).

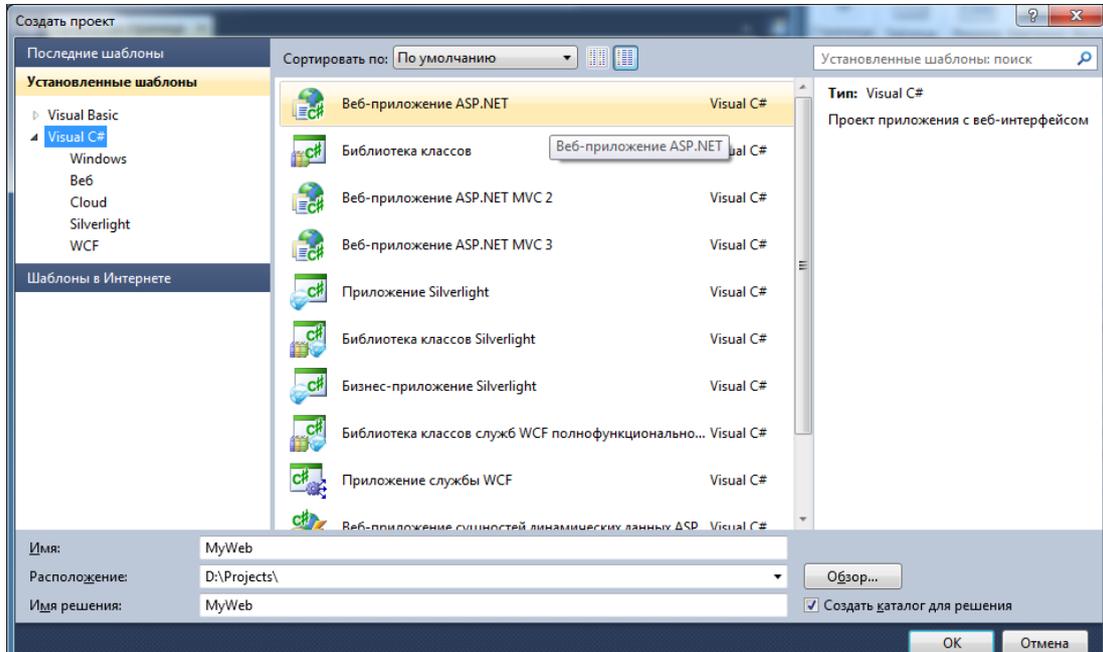


Рис. 3.2. Окно создания проекта

Из предложенных шаблонов необходимо выбрать шаблон **Visual C# → Веб-приложение ASP.NET** (Visual C# → Web Application ASP.NET), указать название проекта (в нашем случае MyWeb) и папку, в которой следует его расположить, и нажать **ОК**.

После выполнения вышеуказанных действий Visual Web Developer создаст новое web-приложение. По умолчанию начальной страницей нового web-сайта является **Default.aspx** (рис. 3.3).

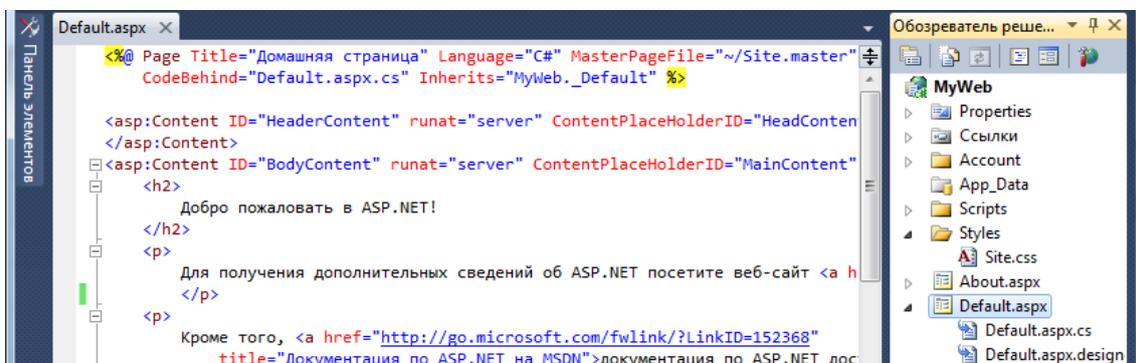


Рис. 3.3. Начальная страница web-приложения

В основу разработки web-приложений на ASP.NET положена модель разделения кода представления и кода реализации. Каждое web-приложение, разрабатываемое на основе ASP.NET, состоит из информационной части, программного кода и сведений о конфигурации.

**Информационная часть** содержит статические и динамические элементы страницы и реализуется в виде web-форм. Статические элементы представляют собой типичные элементы языка HTML, динамические же компонуются программным кодом приложения во время его выполнения (например, запросы к базе данных).

**Программный код** реализует логику, описанную в процедурах обработки данных, определяющих реакцию приложения на запросы пользователя. Программный код исполняется сервером и взаимодействует с динамическими элементами информационной части для формирования отклика приложения.

**Сведения о конфигурации** представляют собой файлы, содержащие параметры, определяющие способ исполнения приложения на сервере, параметры безопасности, реакцию приложения на возникающие ошибки и т. д.

Рассмотрим основные файлы созданного нами web-приложения. Начальная страница **Default.aspx** содержит описание визуального представления web-страницы. Файл состоит из структуры HTML-документа и директив ASP.NET, где указана информация об автоматическом связывании событий данной страницы с обработчиками этих событий. Из кода, приведенного выше, видно, что, во-первых, для создания кода HTML, возвращаемого браузеру, будет использован язык C# (атрибут Language). Во-вторых, код C# содержится в отдельном файле, который будет выполняться на web-сервере (атрибут CodeBehind). И, наконец, атрибут Inherits указывает на имя класса, определенного в CodeBehind. Атрибут runat="server", размещенный в дескрипторе <form>, означает, что данный элемент должен быть обработан средой выполнения ASP.NET.

**Default.aspx.cs** содержит код, обрабатывающий события данной web-формы (расширение .cs указывает на то, что используется язык программирования C#).

**Web.config** содержит параметры конфигурации web-сервера для обслуживания этого проекта.

**Global.asax** содержит обработчики событий, которые реагируют на глобальные события приложения.

**Site.css** определяет стили оформления элементов управления, web-форм и т. д., генерируемые для проекта.

Для того чтобы разобраться, как взаимодействуют компоненты между собой, создадим новую форму.

### 3.2. Создание страницы web-приложения

Для добавления новой формы (страницы) в проект, необходимо щелкнуть правой кнопкой мыши по названию проекта в обозревателе и выбрать пункты **Добавить** → **Создать элемент** (Add → Create Element), после чего появится окно добавления нового элемента (рис. 3.4).

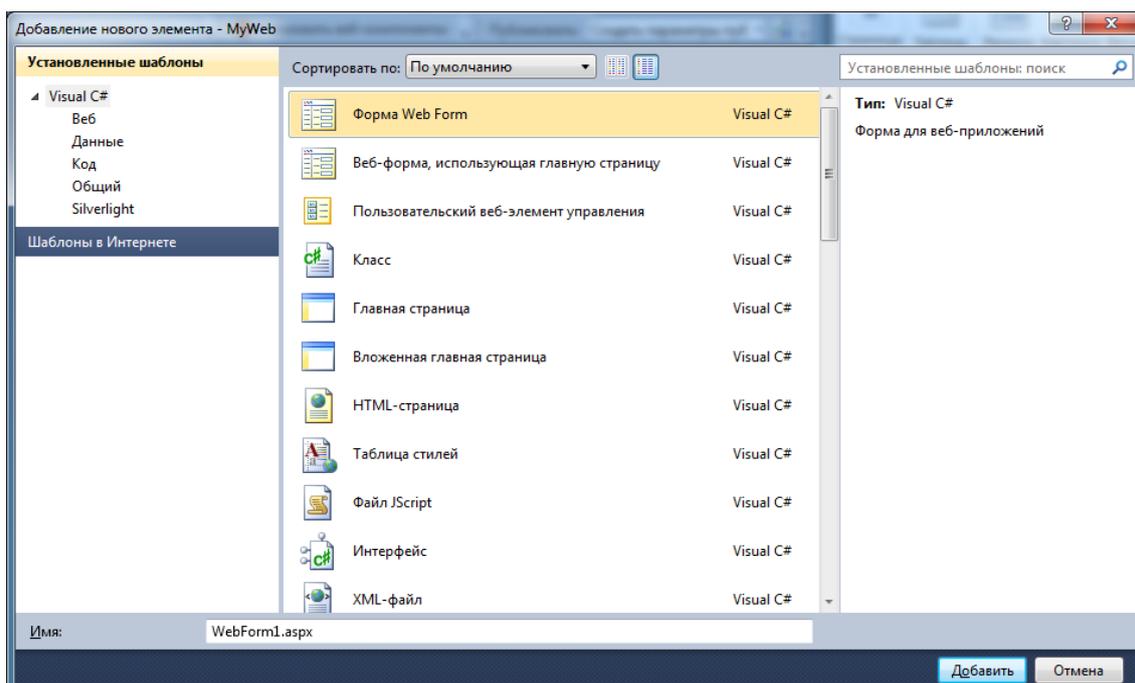


Рис. 3.4. Окно добавления нового элемента

Из предложенных шаблонов необходимо выбрать **Форма Web Form**, задать ей имя и нажать **Добавить** (Add). Таким образом, создается новая web-форма WebForm1.aspx и автоматически генерируется связанный файл с расширением .cs, содержащий код этой страницы.

Редактировать web-форму WebForm1.aspx можно в графическом либо текстовом режиме. Для переключения между режимами достаточно щелкнуть ярлычки **Конструктор** (Design) и **Исходный код** (Source), расположенные внизу окна документа (рис. 3.5)

Web-форма не только представляет собой HTML-страницу со всеми присущими ей атрибутами, но и как Windows-форма позволяет размещать внутри себя различные элементы управления, способные отображать данные и реагировать на действия пользователя.

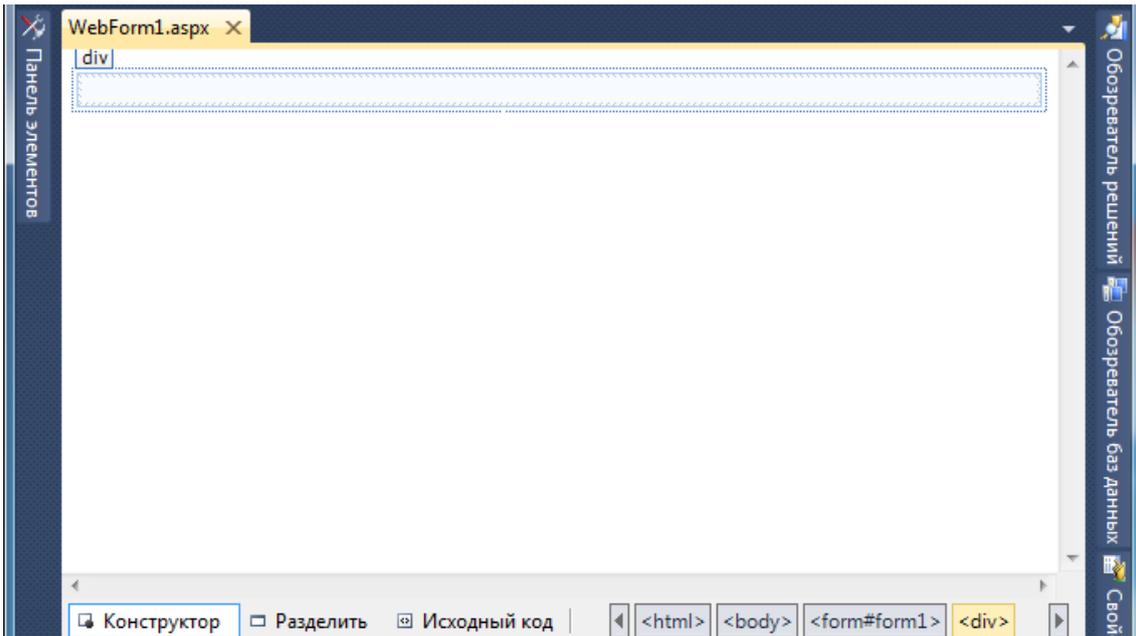


Рис. 3.5. Редактирование web-формы в графическом режиме

Добавим на форму элемент управления **Button**. Для этого откроем **Панель элементов** (ToolBox) и, используя технологию drag-and-drop, перетащим на область формы выбранный элемент **Button** (рис. 3.6). Такого же эффекта можно добиться двойным щелчком по объекту, предварительно установив указатель в нужную область.

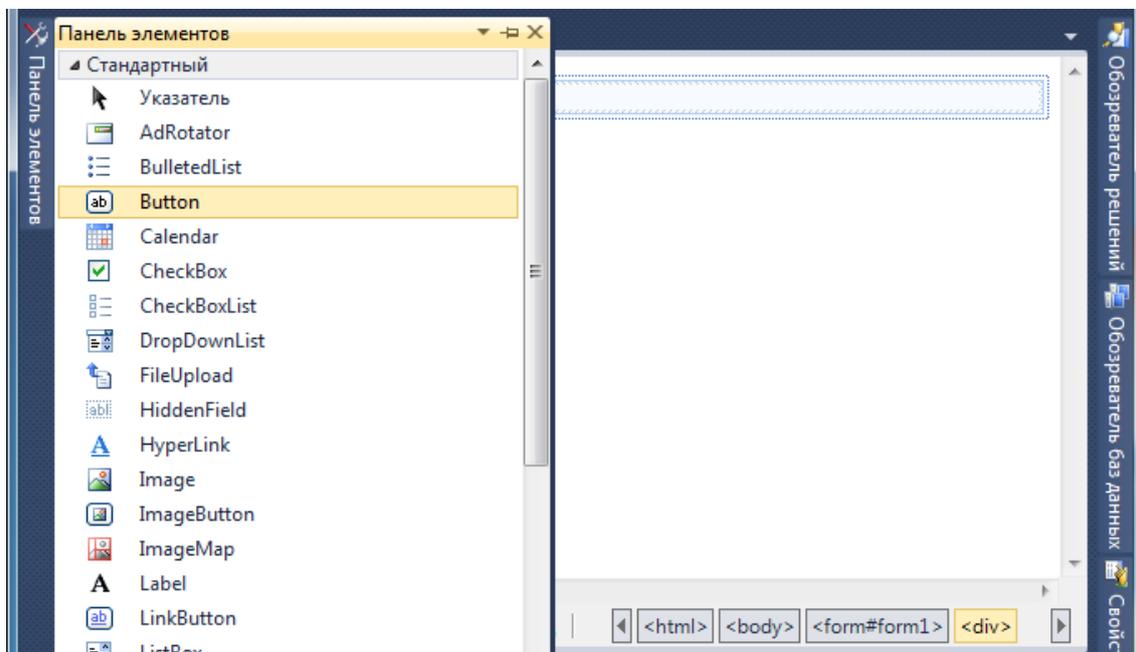


Рис. 3.6. Добавление элемента управления на форму

Описания элементов управления представляются в коде HTML-страницы в виде специальных дескрипторов. Перейдя в режим **Исходный код** или **Разделить**, можно заметить появившуюся строку кода:

```
<asp:Button ID="Button1" runat="server" Text="Button" />
```

Так как основу файла .aspx составляет HTML-код страницы, элементы управления необходимо размещать внутри HTML-элемента **form** (рис. 3.7).

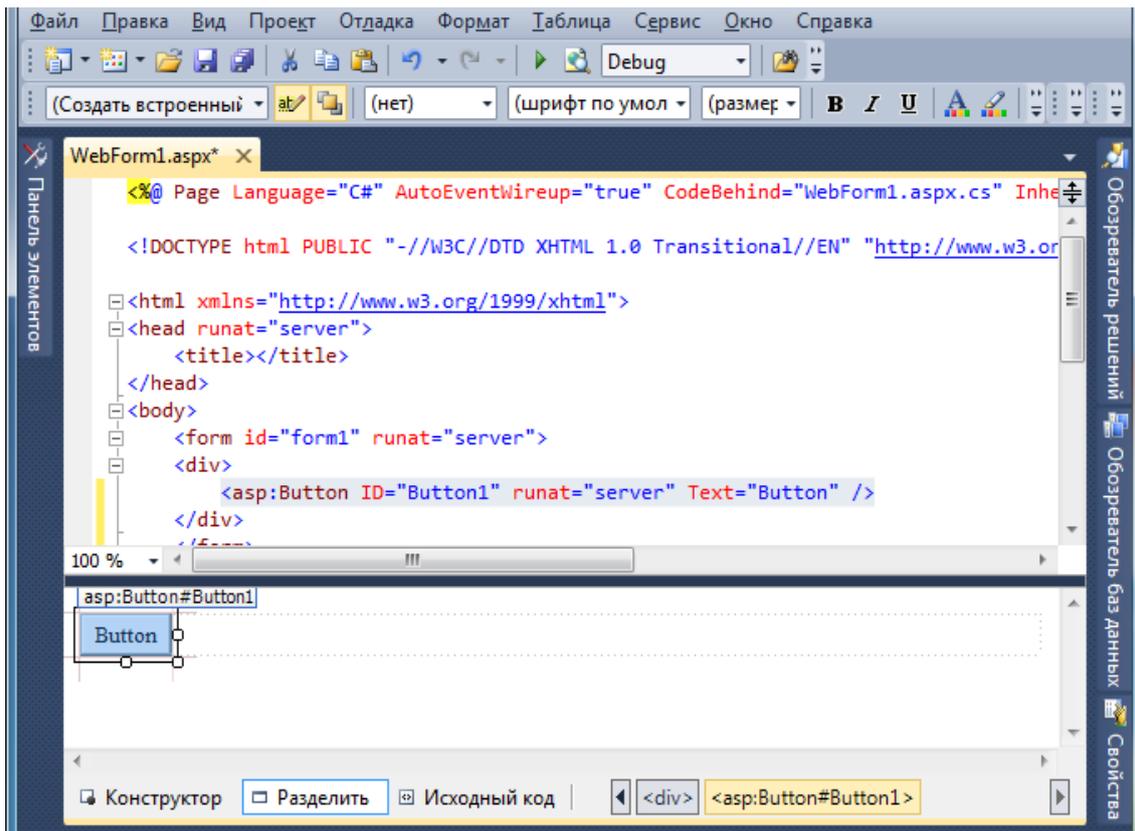


Рис. 3.7. Положение элемента управления Button в коде web-страницы

Следующим этапом является создание обработчика события нажатия кнопки. Пусть после нажатия на кнопку в окно формы выведется сообщение «Это простейшее приложение ASP.NET».

Для создания обработчика события нажатия на кнопку необходимо переключиться в режим дизайна, выделить кнопку, открыть окно свойств (Properties) выделенного объекта, переключиться в режим показа событий, нажав кнопку , и произвести двойной щелчок левой кнопкой мыши по событию, обработчик которого нужно создать, в нашем примере это Click (рис. 3.8).

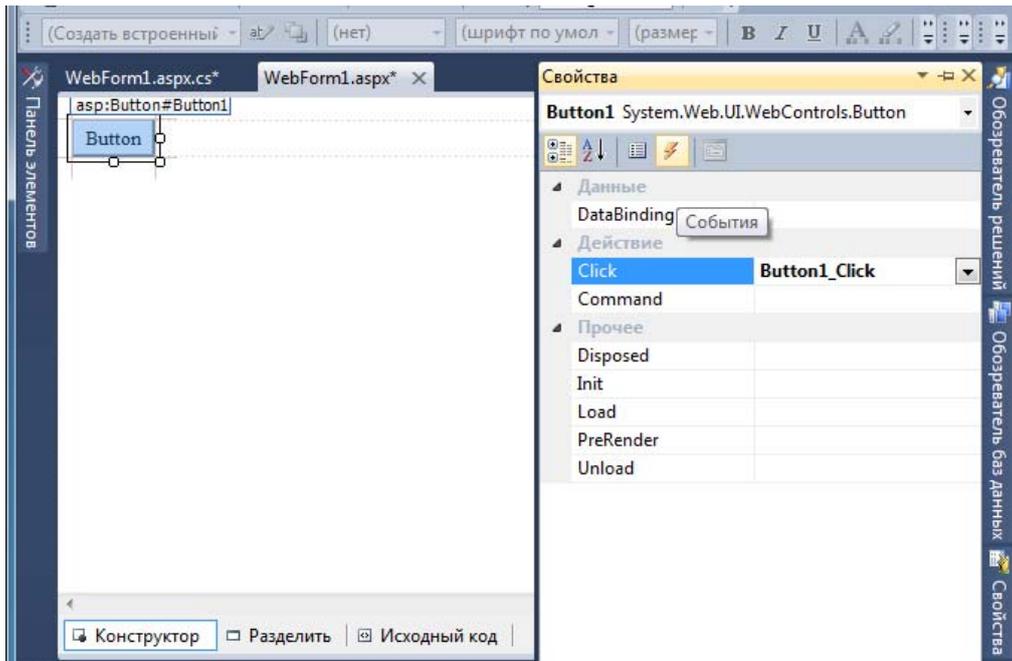


Рис. 3.8. Добавление обработчика события выделенному элементу в окне свойств

В результате создается процедура **Button1\_Click**, связанная с событием **Click** элемента управления **Button**. Эта процедура помещается в файл **WebForm1.aspx.cs**. Для вывода информации в окно браузера можно использовать класс **Response**, который применяется для формирования отклика сервера, пересылаемого браузеру клиента (рис. 3.9).

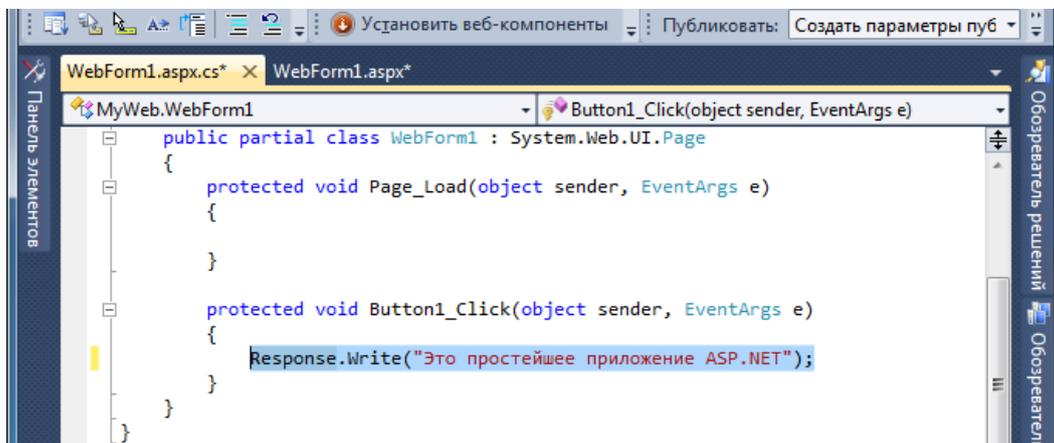


Рис. 3.9. Редактирование кода обработчика события Button1\_Click

Запустить приложение можно нажатием клавиши **F5** (либо щелчком по кнопке  на панели инструментов), после чего запустится отладка приложения и в браузере, используемом по умолчанию, отобразится страница **Default.aspx** (рис. 3.10). Следует отметить, что по-

сле запуска приложения Visual Web Developer запускает свой локальный сервер.



Рис. 3.10. Отображение страницы web-приложения в окне браузера

После нажатия на кнопку на экране отобразится строка «Это простейшее приложение ASP.NET» (рис. 3.11).

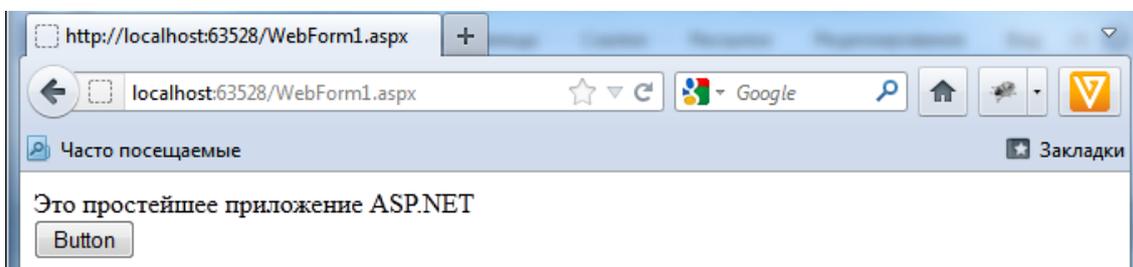


Рис. 3.11. Результат отображения страницы после нажатия кнопки Button1

### 3.3. Взаимодействие элементов web-приложения с клиентом

Взаимодействие пользователя с web-приложением, реализованным на ASP.NET, включает в себя следующие процессы:

1. При запросе ASPX-страницы инициируется событие Page\_Init, производящее начальную инициализацию страницы и ее объекта.
2. Иницируется событие Page\_Load, которое может быть использовано, например, для установки начальных значений для элементов управления. При этом также можно определить, была ли страница загружена впервые или обращение к ней осуществляется повторно в рамках обратной отсылки в ответ на события, связанные с элементами управления, размещенными на странице; т. е. проверить свойство Page.IsPostBack.
3. Выполняется проверка валидности элементов страницы с точки зрения корректности введенных пользователем данных.
4. Осуществляется обработка всех событий, связанных с действиями пользователя с момента последней обратной отсылки.

При обращении клиента к web-приложению сервер IIS перенаправляет запрос модулю ASP.NET на сервере. Модуль ASP.NET читает файл построчно и выполняет коды сценариев, содержащиеся в файле. После этого браузеру возвращается файл ASP.NET, но уже в виде HTML-документа. После отправки экземпляра web-формы уничтожается. Пользователь, получив HTML-страницу, сгенерированную приложением, имеет возможность заполнять различные поля формы. При выполнении определенных действий (например, нажатия на кнопку) пользователь инициирует отправку данных обратно на сервер. Получив данные от пользователя, сервер создает новый экземпляр web-формы, заполняет его полученными данными и обрабатывает все необходимые события. По окончании обработки сервер формирует HTML-код ответа и отправляет его клиенту, а затем уничтожает экземпляр web-формы.

Закончив работу с web-приложением, пользователь либо закрывает браузер, либо переходит на другую интернет-страницу. В этот момент завершается сеанс работы пользователя с данным приложением, однако сервер может прекратить выполнение самого приложения не сразу после окончания последнего сеанса работы пользователя. Это связано с управлением распределением памяти платформой .NET Framework, которая основана на периодической проверке ссылок объектов. Если в результате такой проверки обнаружится, что объект больше не используется, сервер уничтожит его, освободив тем самым занимаемую им память. Таким образом, нельзя точно сказать, когда именно наступит событие **Application\_End** для данного web-приложения.

Однако часто требуется сохранять данные, принадлежащие форме, в течение работы пользователя с приложением. Для этого ASP.NET использует специальный механизм для сохранения данных, введенных в элементы управления web-формы. Согласно этому принципу, в рамках каждого запроса на сервер отправляются все данные, которые были введены в элементы управления. При этом на сервере возникает событие **Page\_Init**, целью которого является создание web-формы и ее инициализация. В процессе инициализации в элементы управления созданной формы записываются переданные от клиента данные. Теперь эти данные становятся доступны приложению посредством обработки события **Page\_Load**, возникающего при каждом обращении к странице.

Для более глубокого понимания кода, созданного ASP.NET в процессе обработки события нажатия на кнопку, необходимо просмотреть исходный код страницы, отображенной в браузере (контекстное меню → Просмотр HTML-кода) (рис. 3.12).

```

<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>

</title></head>
<body>
  <form method="post" action="WebForm1.aspx" id="form1">
<div class="aspNetHidden">
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
value="/wEPDwUKMjA0OTM4MTAwNGRkjYmkUaUJhooRrpi+JqD29jP1Sjlfa0BIg7QdVUBfDdE=" />
</div>

<div class="aspNetHidden">

  <input type="hidden" name="__EVENTVALIDATION" id="__EVENTVALIDATION"
value="/wEWAglcmbTPCwKM54rGBkkuZwH0v4wr0GXmnsO/PbtMG40Iv+4PzTN8ZZLrdUGU" />
</div>
  <div>
    <input type="submit" name="Button1" value="Button" id="Button1" />
  </div>
</form>
</body>
</html>

```

Рис. 3.12. Исходный код страницы, отображенной в браузере

Для возврата информации серверу используется следующая строка:

```
<input type="hidden" name="" id="" value="" />
```

Значение состояния вида (ViewState) представляет собой строку, заданную в формате Base64. Состояние отображения web-формы доступно только внутри этой web-формы. Если необходимо сделать данные, введенные в данную web-форму, доступными другим web-формам одного и того же приложения, их следует сохранить в объектах с более глобальной областью видимости, которые называют переменными состояния: Application и Session. Переменные состояния Application доступны всем пользователям приложения и могут рассматриваться как глобальные переменные, обращение к которым возможно из любых сеансов. Переменные состояния Session доступны только в рамках одного сеанса, следствием чего является то, что они оказываются доступными только одному пользователю. В переменных состояния можно хранить данные любого типа.

### 3.4. Элементы управления и их свойства

#### 3.4.1. Элемент управления *TextBox*

Элемент управления TextBox предназначен для создания на web-форме полей ввода текста. Например:

```
<asp:TextBox ID="Name" runat="server" />
```

В данном примере на web-форме создается поле ввода текста и назначается ему программный идентификатор «Name».

Элемент управления TextBox имеет ряд свойств, которые приведены в табл. 3.1.

Таблица 3.1

**Свойства элемента управления TextBox**

Свойство	Описание
Text	Используется для декларативного добавления текста в TextBox
Rows	Задаёт число строк в TextBox
Columns	Задаёт количество столбцов, которые могут отображаться в TextBox
MaxLength	Ограничивает максимальное число символов, которые могут быть введены
ReadOnly	Значение true запрещает изменение текста пользователем
Wrap	Управляет активизацией режима переноса строк в многострочном TextBox
TextMode	Может быть равно SingleLine (по умолчанию) для создания однострочных полей, MultiLine для создания многострочных полей или Password для создания полей ввода пароля, которые вместо вводимых пользователем символов отображают звездочки или иной символ
AutoPostBack	При установке свойства AutoPostBack элемента управления TextBox в true возвраты формы будут происходить (генерируется событие TextChanged) в момент изменения текста в поле ввода. В противном случае, по умолчанию, элементы управления TextBox сами не инициируют возвраты формы и, таким образом, генерируют событие TextChanged, если возврат формы вызван другим элементом управления

Рассмотрим несколько примеров.

Следующий оператор создает TextBox, первоначально содержащий строку «Добрый день!»:

```
<asp:TextBox ID="Name" runat="server" Text="Добрый день!" />
```

Для считывания содержимого элемента управления TextBox необходимо использовать сценарий вида:

```
string name = Name.Text;
```

Следующий оператор создает поле ввода пароля с именем «Password»:

```
<asp:TextBox ID="Password" runat="server" TextMode="Password" />
```

Для создания многострочного поля с числом строк 10 необходимо установить следующие параметры:

```
<asp:TextBox ID="Comments" runat="server" Rows="10" TextMode="MultiLine" />
```

Если текст в поле TextBox изменился, то после возврата формы элемент управления генерирует событие TextChanged. Обработчик события задается атрибутом OnTextChanged дескриптора <asp:TextBox>:

```
<asp:TextBox ID="Name" runat="server"
OnTextChanged="Name_TextChanged" />
protected void Name_TextChanged(object sender, EventArgs e)
{
    // Имя изменилось, считать его из TextBox.
    string name = Name.Text;
}
```

В следующем примере в ответ на каждый символ, вводимый пользователем, элементы управления TextBox в режиме AutoPostBack генерируют события TextChanged (при переходе к другому элементу управления формы):

```
<asp:TextBox ID="Name" runat="server"
OnTextChanged="Name_TextChanged" AutoPostBack="True" />
```

### **3.4.2. Элемент управления Label**

Label – один из простейших web-элементов управления – позволяет создавать на web-формах программно-управляемые надписи. Текст элемента управления доступен через свойство Text. Следующий оператор добавляет на web-страницу строку «Привет»:

```
<asp:Label ID="Label_Hello" runat="server" Text="Привет" />
```

Часто элементы управления Label применяют, чтобы зарезервировать место для информации, выводимой серверными сценариями. Следующий оператор объявляет пустой элемент управления Label с программным идентификатором «Output»:

```
<asp:Label ID="Output" runat="server">
```

Для помещения строки «Привет» в то место web-страницы, где расположен элемент управления Label, необходимо использовать следующий сценарий:

```
Output.Text = "Привет";
```

### 3.4.3. Элемент управления *HyperLink*

Элемент управления *HyperLink* добавляет к web-форме гиперссылку в виде текста или изображения.

Следующий оператор создает гиперссылку, которая представляется на web-странице в виде текстовой строки и ссылается на [www.it.bstu.unibel.by](http://www.it.bstu.unibel.by).

Пример гиперссылки в виде текста:

```
<asp:HyperLink ID="HyperLink_Isit" runat="server"
NavigateUrl="http://it.bstu.unibel.by" Text="Сайт кафедры ИСиТ БГТУ" />
```

Пример гиперссылки в виде изображения:

```
<asp:HyperLink ID="HyperLink_Isit" runat="server"
NavigateUrl="http://it.bstu.unibel.by" ImageUrl="~/isit.gif" Target="_blank" />
```

Свойство *Target* позволяет управлять тем, как будет отображаться целевая web-страница. В примере, описанном выше, оператор открывает страницу в новом окне.

Надо отметить, что элементы управления имеют смысл только тогда, когда необходимо изменять свойства элемента управления динамически.

Следующий код инициализирует целевой адрес гиперссылки во время загрузки страницы:

```
<asp:HyperLink ID="HyperLink_Isit" runat="server" Text="Сайт кафедры
ИСиТ БГТУ" />
```

```
protected void Page_Load(object sender, EventArgs e)
{
    HyperLink_Isit.NavigateUrl = "http://it.bstu.unibel.by";}
```

### 3.4.4. Элемент управления *Image*

Элемент управления *Image* добавляет к web-форме картинки, генерируя дескрипторы `<img>`. Самые популярные свойства *Image* – *ImageUrl* (определяет URL картинки, которая будет отображаться), *ImageAlign* (управляет выравниванием картинки), *AlternateText* (задает альтернативный текст для картинки). Альтернативный текст отображается вместо картинки в браузерах, которые работают в текстовом режиме.

Следующий оператор объявляет элемент управления *Image* на web-форме:

```
<asp:Image ID="Image_Logo" runat="server" ImageUrl="~/Logo.gif"
AlternateText="Логотип" />
```

### 3.4.5. Элемент управления *CheckBox*

Элемент управления *CheckBox* создает на web-форме поле флажков. Свойство *Checked* определяет, установлен ли флажок (*true*) или сброшен (*false*), а *Text* определяет текст, отображаемый рядом. Следующий оператор объявляет элемент управления *CheckBox* на web-форме:

```
<asp:CheckBox ID="Confirm" runat="server"
Text="Подтвердите свой выбор" />
```

А этот серверный сценарий определяет состояние флажка, когда форма возвращается на сервер:

```
if (Confirm.Checked) {
    // Флажок установлен.
}
else {
    // Флажок сброшен.
}
```

В нестандартном случае, когда нужно поменять местами флажок и текст, обычно отображаемый правее, используется атрибут *TextAlign="Left"* в дескрипторе элемента управления.

При установке и сбросе флажка элементы управления *CheckBox* генерируют события *CheckedChanged*. По умолчанию событие *CheckedChanged* не генерируется сразу же при щелчке флажка, а откладывается до возврата формы. Чтобы реагировать на изменения состояния флажка сразу, необходимо установить в *true* свойство *AutoPostBack* для принудительного возврата формы:

```
<asp:CheckBox ID="Confirm" runat="server" Text="Подтвердите свой выбор"
AutoPostBack="True" OnCheckedChanged="Confirm_CheckedChanged" />
```

```
protected void Confirm_CheckedChanged(object sender, EventArgs e)
{
    // Флажок только что установлен или сброшен.
    // Выполнить нужные действия.
}
```

Однако надо отметить, что не стоит везде устанавливать *AutoPostBack* в *true*, так как это приводит к большой нагрузке на сервер. Целесообразно использовать это только тогда, когда необходимо динамическое изменение содержимого страницы при всяком изменении состояния элемента управления.

### 3.4.6. Элемент управления *RadioButton*

Элемент управления `RadioButton` создает на web-форме кнопки-переключатели, которые отображают список взаимоисключающих вариантов. `RadioButton` является производным от `CheckBox` и, таким образом, поддерживает те же свойства и события. Кроме того, он имеет дополнительное свойство `GroupName` для указания группы, к которой относится переключатель.

В следующем примере объявляется пять элементов управления `RadioButton`, разделенных на две группы: из трех и из двух кнопок. Свойство `RadioButton.Checked` включает первую кнопку каждой группы:

```
<asp:RadioButton ID="Green" runat="server" GroupName="Color"
Text="Зеленый" Checked="True" /><br />
<asp:RadioButton ID="Red" runat="server" GroupName="Color"
Text="Красный" /><br />
<asp:RadioButton ID="Blue" runat="server" GroupName="Color"
Text="Синий" />

<asp:RadioButton ID="Circle" runat="server" GroupName="Shape"
Text="Круг" Checked="True" /><br />
<asp:RadioButton ID="Square" runat="server" GroupName="Shape"
Text="Квадрат" />
```

Группировка элементов управления данного типа с помощью атрибута `GroupName` сообщает браузеру о том, какие переключатели нужно отключить при включении данного переключателя.

Чтобы в серверном сценарии определить, какой переключатель из группы был включен, нужно проверить значение свойства `Checked` у каждой кнопки группы. Удобнее для добавления переключателей к web-странице использовать `RadioButtonList`. Его свойство `SelectedIndex` соответствует включенной кнопке. `RadioButtonList` также упрощает задачу выравнивания кнопок-переключателей на странице.

### 3.4.7. Элемент управления *Table*

Код для задания элемента управления `Table`:

```
<asp:Table ID="MyTable" runat="server">
  <asp:TableRow runat="server">
    <asp:TableCell runat="server">Row 1, Column
1</asp:TableCell>
    <asp:TableCell runat="server">Row 1, Column
2</asp:TableCell>
```

```

</asp:TableRow>
<asp:TableRow runat="server">
    <asp:TableCell runat="server">Row 2, Column
1</asp:TableCell>
    <asp:TableCell runat="server">Row 2, Column
2</asp:TableCell>
</asp:TableRow>
</asp:Table>

```

Элементы управления Table полезны, когда содержимое таблицы нужно изменять динамически. Например, в следующем серверном сценарии изменяется текст во всех ячейках таблицы:

```

MyTable.Rows[0].Cells[0].Text = "Cell 1";
MyTable.Rows[0].Cells[1].Text = "Cell 2";
MyTable.Rows[1].Cells[0].Text = "Cell 3";
MyTable.Rows[1].Cells[1].Text = "Cell 4";

```

А этот сценарий создает во время исполнения всю таблицу:

```

<asp:Table ID="My_Table" runat="server">
</asp:Table>
for (int i = 0; i < 2; i++)
{
    TableRow row = new TableRow();
    for (int j = 0; j < 2; j++)
    {
        TableCell cell = new TableCell();
        cell.Text = String.Format("Row {0}, Column {1}", i + 1, j + 1);
        row.Cells.Add(cell);
    }
    My_Table.Rows.Add(row);
}

```

Эти сценарии работают благодаря тому, что строки, содержащиеся в объекте Table, доступны через свойство Rows. Каждая строка набора Rows – это экземпляр класса TableRow. Внутри строки каждая ячейка представлена объектом TableCell, доступ к которому осуществляется посредством набора Cells объекта-строки. Вызов Add для набора Rows или Cells позволяет программно добавить строку к таблице или ячейку к строке.

По умолчанию рамки элементов управления Table невидимы. Изменить это можно, установив свойство GridLines в Horizontal, Vertical или Both. Другие свойства Table – CellPadding и CellSpacing, – как и одноименные HTML-атрибуты, управляют промежутками внутри ячеек, а BackImageUrl задает фоновую картинку. Таблицы часто используются web-страницами для отображения цветного фона. Изменить цвет фона для Table позволяет его свойство BackColor, унаследованное от WebControl.

### 3.4.8. Элемент управления Panel

Элемент управления Panel служит контейнером для других элементов управления.

Следующий программный код позволяет отображать и скрывать два элемента управления Label, устанавливая свойство Visible элемента управления Panel в true или false при всяком щелчке флажка:

```
<asp:Panel ID="MyPanel" runat="server" Height="50px" Width="125px">  
<asp:Label ID="surname" runat="server" Text="Иванов"></asp:Label><br />  
<asp:Label ID="name" runat="server" Text="Иван"></asp:Label><br />  
</asp:Panel>  
  
<asp:CheckBox ID="Toggle" runat="server" AutoPostBack="True"  
OnCheckedChanged="Toggle_CheckedChanged" Text="Показать надписи" />
```

```
protected void Toggle_CheckedChanged(object sender, EventArgs e)  
{  
    MyPanel.Visible = Toggle.Checked;  
}
```

### 3.4.9. Кнопки

Семейство web-элементов управления включает три типа кнопок: Button, LinkButton и ImageButton. Все три имеют общее функциональное назначение: возврат содержащей их формы на сервер. Отличия: Button выглядит как командная кнопка, LinkButton – как гиперссылка, а ImageButton отображает заданную картинку.

Ниже приведены примеры объявления экземпляров кнопок каждого типа:

```
<asp:Button ID="Button1" runat="server" Text="Кнопка"  
OnClick="Button1_Click" />
```

```
<asp:LinkButton ID="LinkButton1" runat="server" Text="Кнопка-ссылка" />
```

```
<asp:ImageButton ID="ImageButton1" runat="server" ImageUrl="~/Logo.gif"
AlternateText="Кнопка-изображение" OnClick="ImageButton1_Click" />
```

Свойство `Text` задает текст, отображаемый поверх элемента управления `Button` или `LinkButton`. `ImageUrl` задает картинку, отображаемую элементом управления `LinkButton`. Все типы кнопок генерируют по щелчку два вида событий: `Click` и `Command`. Атрибут `OnClick` в дескрипторе элемента управления связывает с кнопкой обработчик `Click`. Обработчики `Click` для `Button` и `LinkButton` имеют следующий прототип:

```
protected void Button1_Click(object sender, EventArgs e)
{
    // Здесь выполняется обработка события,
}
```

Прототип обработчиков `Click` для элементов `ImageButton` отличается:

```
protected void ImageButton1_Click(object sender, ImageClickEventArgs e)
{
    // Получить координаты щелчка,
    int x = e.X;
    int y = e.Y;
}
```

Объект `ImageClickEventArgs`, передаваемый обработчику `Click` объекта `ImageButton`, содержит открытые поля `X` и `Y`, указывающие положение курсора мыши в момент щелчка. `X` и `Y` измеряются в пикселах и соответствуют расстоянию от левого верхнего угла картинки.

### 3.4.10. Списки

В семейство со списками входят следующие четыре элемента управления: `ListBox`, `DropDownList`, `CheckBoxList` и `RadioButtonList`.

Элемент управления **`DropDownList`** отображает элементы в выпадающем списке.

Следующий код создает список из трех цветов и отображает выбор пользователя на web-странице:

```
<asp:DropDownList ID="ColorList" runat="server">
    <asp:ListItem Text="Зеленый"></asp:ListItem>
    <asp:ListItem Text="Красный"></asp:ListItem>
```

```

        <asp:ListItem Text="Синий"></asp:ListItem>
</asp:DropDownList>
        <asp:Button ID="Select_Color" runat="server"
OnClick="Select_Color_Click" Text="Выберитецвет" />
        <asp:TextBox ID="Color" runat="server"></asp:TextBox>

```

```

protected void Select_Color_Click(object sender, EventArgs e)
{
    Color.Text = ColorList.SelectedItem.Text;
}

```

Свойство SelectedItem позволяет определить, какой элемент из списка выбран.

Элемент управления **ListBox** отличается от DropDownList тем, что отображает элементы в статичном виде.

```

<asp:ListBox ID="ColorList" runat="server" Rows="3">
    <asp:ListItem Text="Красный"></asp:ListItem>
    <asp:ListItem Text="Оранжевый"></asp:ListItem>
    <asp:ListItem Text="Желтый"></asp:ListItem>
</asp:ListBox>
<asp:Button ID="Select_Color" runat="server" OnClick="Select_Color_Click"
Text="Выберите цвет"/>
<asp:TextBox ID="Color" runat="server"></asp:TextBox>

```

```

protected void Select_Color_Click(object sender, EventArgs e)
{
    Color.Text = ColorList.SelectedItem.Text;
}

```

Свойство Rows определяет высоту списка.

Атрибут SelectionMode="Multiple" в дескрипторе элемента управления создает ListBox с множественным выбором, однако при этом придется перебирать все элементы списка по одному, проверяя свойство Selected, например:

```

<asp:ListBox ID="ColorList" runat="server" Rows="3" SelectionMode=
"Multiple">

```

Следующий метод принимает ссылку на ListBox как входной параметр и возвращает массив целых чисел, содержащий индексы всех выбранных элементов, начиная с нуля:

```

int[] GetSelectedIndices(ListBox lb)
{
    ArrayList a = new ArrayList();
    for (int i = 0; i < lb.Items.Count; i++)
    {
        if (lb.Items[i].Selected)
            a.Add(i);
    }
    int [] indices = new int[a.Count];
    a.CopyTo (indices);
    return indices;}

```

Элемент управления **CheckBoxList** создает массивы флажков, например:

```

<asp:CheckBoxList ID="ColorList" runat="server">
    <asp:ListItem Text="Зеленый" Selected="True"></asp:ListItem>
    <asp:ListItem Text="Красный"></asp:ListItem>
    <asp:ListItem Text="Синий"></asp:ListItem>
</asp:CheckBoxList>

```

Чтобы в серверном сценарии определить, установлен ли данный флажок, необходимо обратиться к значению его свойства Selected:

```

if (ColorList.Items[2].Selected)
{
    // Флажок установлен.
}
else
{
    // Флажок сброшен.
}

```

Элемент управления **RadioButtonList** упрощает создание групп переключателей и определение выбранного переключателя.

Следующий программный код создает столбец переключателей и включает первый из них:

```

<asp:RadioButtonList ID="ColorList" runat="server">
    <asp:ListItem Text="Зеленый" Selected="True"></asp:ListItem>
    <asp:ListItem Text="Красный"></asp:ListItem>

```

```
<asp:ListItem Text="Синий"></asp:ListItem>  
</asp:RadioButtonList>
```

Для определения выбранного переключателя используется следующий сценарий:

```
int index = ColorList.SelectedIndex;
```

### **3.4.11. Динамическое создание элементов управления**

Иногда необходимо, чтобы интерфейс приложения менялся в зависимости от действий пользователя. В этом случае удобным инструментом реализации динамического интерфейса становится динамическое создание и отображение на странице элементов управления. Для создания элемента управления нужно создать объект соответствующего класса, присвоить его атрибутам необходимые значения и добавить его к коллекции элементов управления страницы. Но так как в момент исполнения программного кода, создающего объект и добавляющего его к коллекции элементов управления, страница уже создана, то элемент будет добавляться после последнего элемента управления страницы.

Для большего контроля над расположением динамически создаваемого элемента управления можно воспользоваться специально предназначенным для этого элементом `PlaceHolder`. Данный элемент управления служит для размещения внутри него других элементов и может быть расположен в любом месте страницы. При этом если внутри `PlaceHolder` отсутствует содержимое, это никак не повлияет на содержимое страницы.

При использовании динамически создаваемых элементов управления необходимо помнить, что они существуют только до очередной обратной отсылки. Если после обратной отсылки динамически созданный элемент управления нужно по-прежнему отображать на странице, необходимо создавать его в обработчике события `Page.Load`. При создании элемента управления в обработчике события `Page.Load` ASP.NET использует любую информацию о состоянии вида по завершении этого обработчика события.

Для возможности программного взаимодействия с динамически созданным элементом управления последнему необходимо присвоить уникальный идентификатор (ID), являющийся аналогом ID любого другого элемента управления. В дальнейшем этот идентификатор можно использовать для того, чтобы найти данный элемент управления в коллекции элементов управления страницы с помощью метода `Page.FindControl()` и использовать в программном коде.

Проиллюстрируем все вышесказанное на следующем примере. Предположим, в рассматриваемом выше примере необходимо ввести возможность ввода отчества по желанию пользователя. При этом не желательно, чтобы поле для ввода отчества появлялось на экране, если пользователь включит соответствующий режим. Реализовать это можно следующим образом.

Изменим предыдущий макет. Для кнопки укажем в свойствах ID=btn\_Submit. В режиме дизайна страницы поместим перед кнопкой два элемента Label и два элемента TextBox, предназначенных для ввода фамилии и имени. Зададим для элементов Label идентификаторы lbl\_LastName и lbl\_FirstName, а для элементов TextBox – соответственно tb\_LastName и tb\_FirstName.

После этого разместим на форме элемент управления CheckBox и установим свойство AutoPostBack этого элемента равным True. Расположим на форме элемент управления Placeholder, а также поместим элемент Label для вывода приветствия и зададим ему ID=lbl\_Result. В результате этих действий страница в режиме дизайна будет выглядеть следующим образом (рис. 3.13).

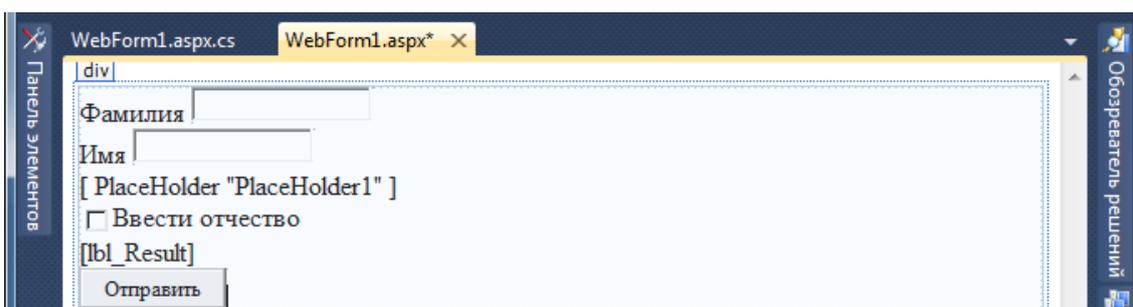


Рис. 3.13. Вид web-формы в режиме дизайна

Модифицируем исходный код приложения. В обработчик события Page.Load добавим следующий код, создающий объекты типа Label и TextBox и добавляющий их к коллекции Controls элемента Placeholder:

```
if (CheckBox1.Checked)
{
    Label lblSName = new Label();
    lblSName.ID = "lbl_SName";
    lblSName.Text = "Отчество";
    Placeholder1.Controls.Add(lblSName);
    TextBox tbSName = new TextBox();
```

```

tbSName.ID = "tb_SName";
PlaceHolder1.Controls.Add(tbSName);
}

```

Внесем изменения в программный код обработчика события нажатия на кнопку «Отправить». Полный текст программного кода этого обработчика представлен ниже.

```

string sname=string.Empty;
    TextBox tbSname=(TextBox)Page.FindControl("tb_SName");
if (tbSname!=null)
{
    sname = tbSname.Text;
}
lbl_Result.Text = "Здравствуйте, "+tb_FirstName.Text+" "+sname+" "+
    tb_LastName.Text+"! Добро пожаловать в приложение
    ASP.NET";
lbl_Result.ForeColor = Color.Red;

```

Из примера видно, что с помощью метода Page.FindControl происходит поиск элемента управления tb\_SName. В случае, если он будет найден на странице, значение свойства Text данного элемента записывается в переменную sname, которая затем используется для вывода информации на экран.

Для подключения к данному проекту области имен (namespace), содержащей определение типа Color, ввести в начале .cs-файла следующую строку:

```
using System.Drawing;
```

Результат работы программы изображен на рис. 3.14 и 3.15.

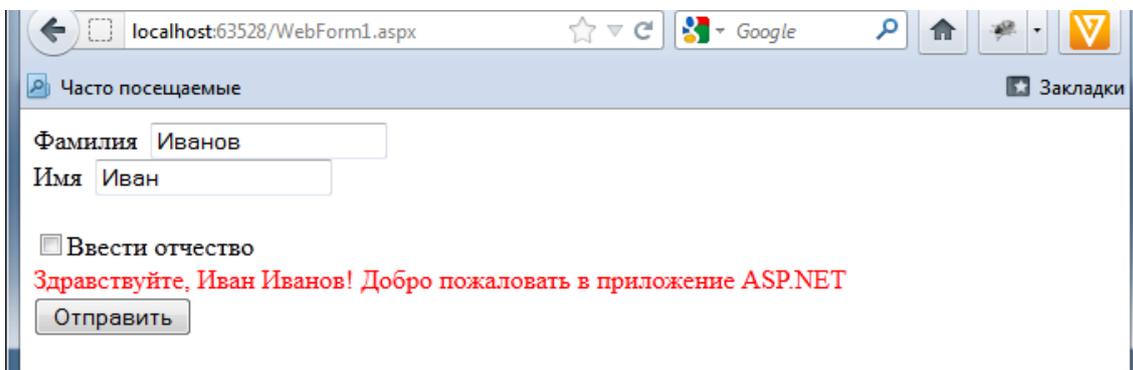


Рис. 3.14. Отображение примера в окне браузера (при CheckBox1.Checked=False)

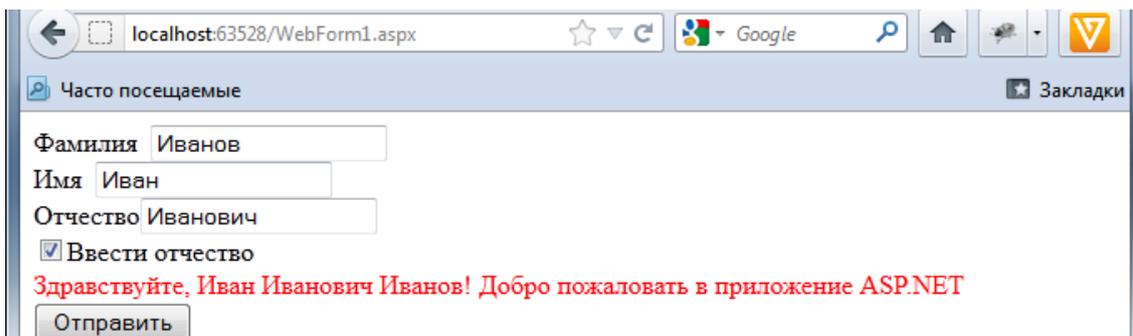


Рис. 3.15. Отображение примера в окне браузера (при CheckBox1.Checked=True)

Динамически созданные элементы управления могут обрабатывать события. Для реализации этой возможности необходимо подключить обработчик события с помощью кода-делегата в событии Page.Load. Так как все события элементов управления генерируются после Page.Load, подключение обработчика события динамического элемента управления за пределами Page.Load приведет к тому, что событие элемента управления обрабатываться не будет.

Для подключения обработчика события изменения текста в динамически созданном элементе управления необходимо выполнить следующие действия.

1. Добавить процедуру обработчика события.

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

2. В событии Page.Load привязать этот обработчик к соответствующему событию элемента управления.

```
tbSName.TextChanged += new System.EventHandler
(this.tb_SName_TextChanged);
```

### 3.5. Проверка вводимых данных

Вводимые пользователем данные могут сохраняться во вспомогательных файлах, базе данных и т. п. и влиять на дальнейшую работу сайта и предоставляемый пользователю функционал. Валидация – проверка вводимых данных – позволяет исключить ошибочную, бесполезную или вредоносную информацию.

Критерии проверки могут быть различными, начиная с того, ввелись ли данные вообще, и заканчивая проверкой типа данных, соответствием конкретному шаблону или диапазону значений. Проверять

введенную информацию необходимо на стороне сервера, но целесообразно начинать проверку еще до отправки пользователем данных (обычно осуществляется программой, написанной на JavaScript).

В ASP.NET реализовано шесть элементов управления, производящих проверку вводимых данных, так называемых верификаторов:

- **RequiredFieldValidator** – проверяет наличие введенных данных;
- **RangeValidator** – проверяет принадлежность определенному диапазону;
- **RegularExpressionValidator** – проверяет соответствие регулярному выражению;
- **CompareValidator** – сравнивает с константой или другим элементом управления;
- **CustomValidator** – используется для нетипичной проверки с помощью собственной функции, например проверка числа на четность;
- **ValidationSummary** – позволяет вывести итоговую информацию по всем валидаторам на странице. Форма вывода может быть различной.

После назначения верификатора выполняется автоматическая клиентская и серверная проверка данных. Если отсылаемая информация не удовлетворяет условию, верификатор препятствует отправке страницы на сервер.

Рассмотрим небольшой пример работы верификаторов.

Для добавления верификатора необходимо указать место на странице, где он будет размещен, в панели элементов найти необходимый валидатор и двойным нажатием добавить его на страницу (рис. 3.16).

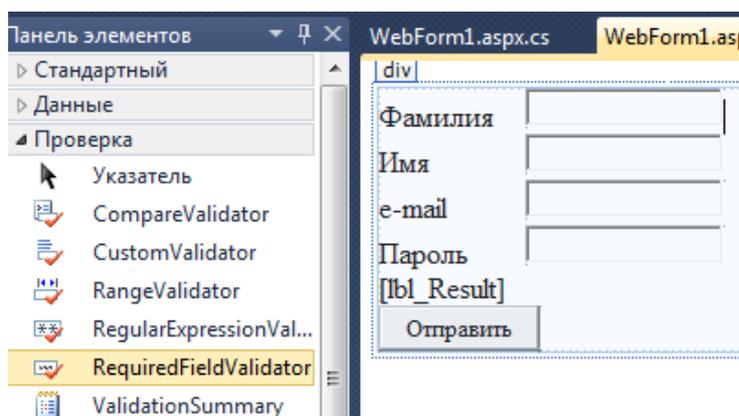


Рис. 3.16. Добавление верификатора

После добавления валидатора в его свойствах необходимо указать, с каким элементом он будет взаимодействовать (**ControlToValidate** – рис. 3.17).

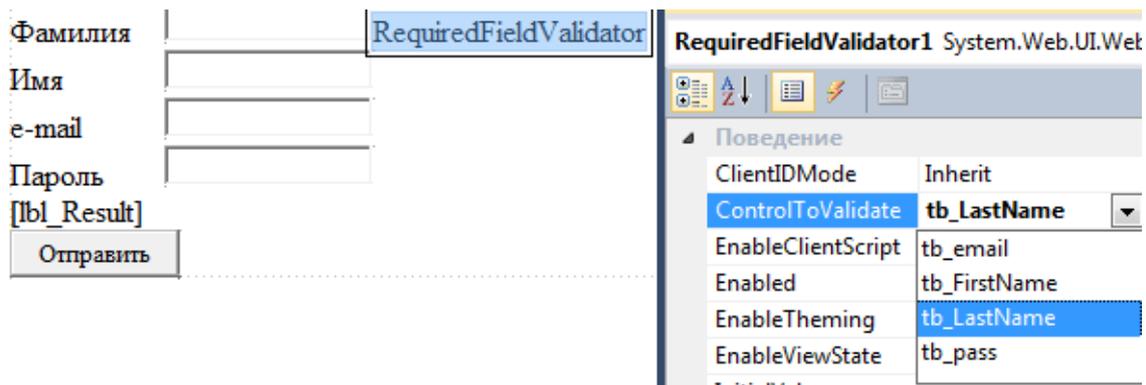


Рис. 3.17. Окно свойств валидатора RequiredFieldValidator

Второе, что необходимо установить в свойствах валидатора (поле **ErrorMessage**), это сообщение, которое должно выводиться в случае его срабатывания, например «\*».

По аналогии RequiredFieldValidator необходимо добавить на остальные поля, обязательные к заполнению. В случае, если поле не будет заполнено, валидатор выведет сообщение, указанное в ErrorMessage. Однако отсутствие значения в поле – не единственная ошибка, которая может возникнуть в форме регистрации. Например, поле e-mail может содержать массу ошибок различного рода. Для контроля правильности его написания необходимо использовать верификатор RegularExpressionValidator, в свойствах которого нужно выставить дополнительный параметр ValidationExpression и указать в нем регулярное выражение для проверки e-mail (рис. 3.18).

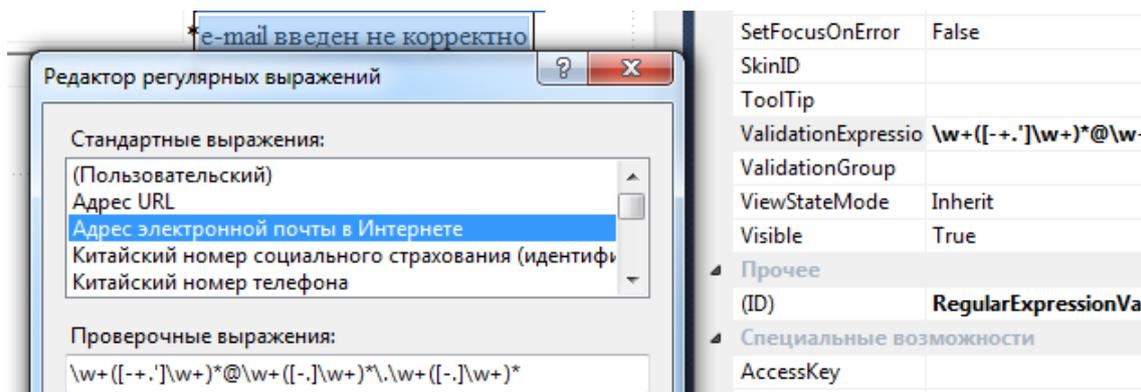


Рис. 3.18. Окно свойств валидатора RegularExpressionValidator

Подобного рода проверками можно исключить вероятность ошибки для любого поля.

Необходимо учесть ситуацию, в которой пользователь в обход ограничению может отправить данные на сервер. Несомненно, сервер

также сделает проверку и выдаст сообщения. При этом добросовестно продолжит работать с неправильными данными, как если бы проверки не было. Во избежание этого в обработчике события для кнопки необходимо написать соответствующий код:

```
if (!Page.IsValid) return; //Если страница не валидна, прекратить выполнение скрипта
```

С помощью `ValidationSummary` можно подвести своеобразный итог по ошибкам, обнаруженным другими валидаторами. Его необходимо разместить в области для вывода списка ошибок (рис. 3.19).

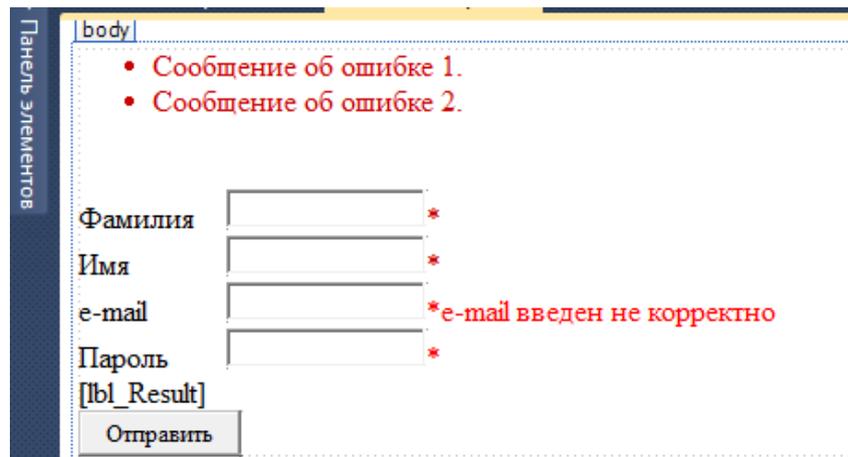


Рис. 3.19. Добавление валидатора `ValidationSummary`

`ValidationSummary` отображает список сообщений в поле `ErrorMessage`, полученных от других валидаторов. Исходя из этого, в свойствах ранее созданных элементов необходимо указать соответствующие значения. При этом значение, заданное в поле «Text», будет отображаться на месте расположения сработавшего валидатора, а `ErrorMessage` – в `ValidationSummary` (рис. 3.20).

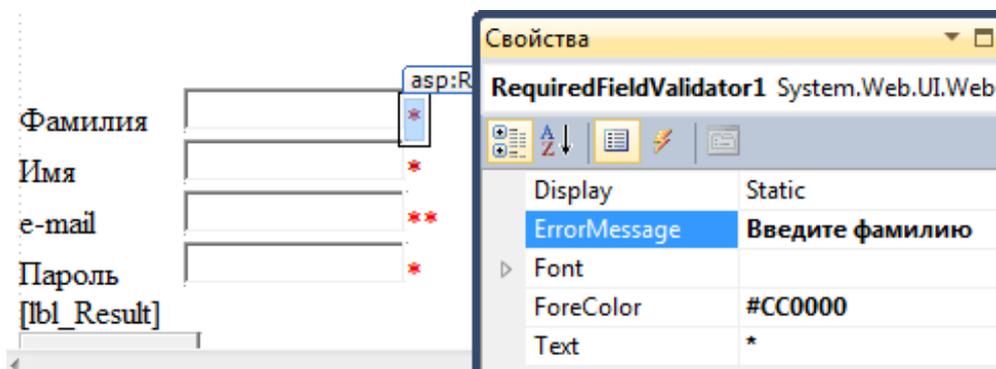


Рис. 3.20. Изменение свойств валидаторов

После компиляции примера получаем следующий результат (рис. 3.21).

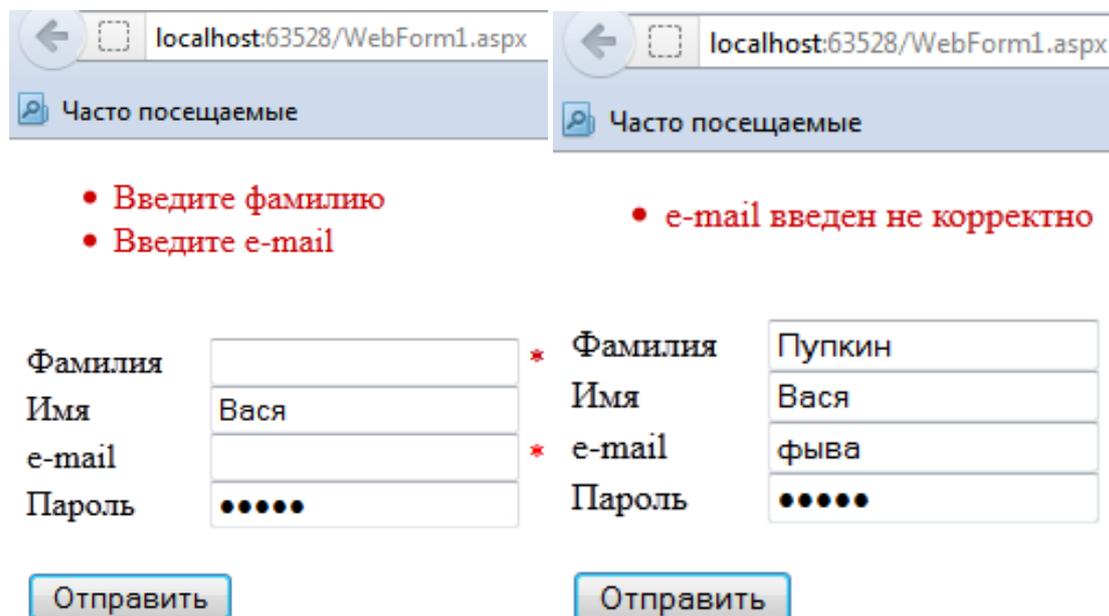


Рис. 3.21. Действие верификаторов при некорректном вводе данных

### 3.6. Работа с базами данных

Наиболее типичным сценарием взаимодействия web-приложения с базой данных является следующий:

1. Устанавливается соединение с базой данных (БД).
2. Выполняется один или несколько запросов, осуществляющих внесение изменений или выборку данных из БД.
3. Осуществляется отключение от источника данных. При этом пользователь продолжает работать с отсоединенным набором данных, просматривая его, выполняя фильтрацию, внося изменения и т. д.

При необходимости переноса изменений из отсоединенного набора данных в БД, а также при необходимости просмотра изменений, внесенных в БД другими пользователями, нужно произвести действия, описанные в пунктах 1–3.

Рассмотрим приведенные выше этапы более подробно.

Для того чтобы подключиться к базе данных, необходимо создать объект `Connection`, а также задать его свойства, определяющие текущие параметры подключения. Основным параметром, устанавливающим необходимые для подключения к БД опции, является строка соединения, представляющая собой набор пар «имя – значение», разде-

ленных точкой с запятой. Строка соединения зависит от СУБД, к которой осуществляется подключение. Тем не менее существуют параметры, необходимые практически всегда:

1. Адрес сервера, на котором находится база данных. Если она расположена на клиентском компьютере, необходимо указать «localhost» либо IP-адрес 127.0.0.1.

2. Имя базы данных, к которой производится подключение.

3. Имя пользователя и пароль, которые используются для проверки прав БД.

Управление соединением осуществляется с помощью методов Open и Close объекта Connection. Следует учитывать, что при подключении к базе данных может произойти сбой, в результате которого установить соединение с ней окажется невозможно – типичная ситуация при размещении базы данных на другом сервере. Чтобы неудавшаяся попытка соединения с базой данных не приводила к фатальным последствиям, необходимо использовать конструкции try catch, позволяющие адекватно реагировать на возникшую ошибку:

```
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();
}
```

Для обслуживания соединения с базой данных расходуются ресурсы web-сервера. Поэтому рекомендуется открывать соединение как можно реже и позже, а закрывать как можно раньше – сразу после того, как все необходимые действия с БД были выполнены. Кроме того, желательно строить программный код так, чтобы соединение с базой данных закрывалось при любых событиях. В предыдущем примере код, расположенный в блоке `finally`, выполнится при любом исходе попытки подключения к БД, что гарантирует освобождение ресурсов сервера до того, как память будет освобождена сборщиком мусора.

Подключение необходимо для того, чтобы выполнять определенные команды над элементами БД. Одним из основных классов ADO.NET, способным выполнять любой SQL-оператор, является класс `Command`. Существует три типа команд класса `Command`:

- `CommandType.Text` – выполнение прямого оператора SQL, текст которого устанавливается в свойстве `CommandText`. Это значение по умолчанию;

- `CommandType.StoredProcedure` – выполнение хранимой процедуры, имя которой установлено в свойстве `CommandText`;

- `CommandType.TableDirect` – выполнение опроса всей таблицы базы данных, имя которой задается в свойстве `CommandText`. Этот тип команд используется для обратной совместимости с некоторыми драйверами OLE DB и не поддерживается поставщиком данных SQL Server.

Для выполнения созданной команды необходимо использовать один из следующих методов:

- `ExecuteReader()` – выполнение запроса SQL и возврат объекта `DataReader`, представляющего однонаправленный курсор, с доступом только для чтения;

- `ExecuteNonQuery()` – выполнение SQL-команд, предназначенных для вставки, изменения, удаления записей БД. Результатом работы команды является количество строк, обработанных командой;

- `ExecuteScalar()` – выполнение SQL-команды и возврат первой строки результата запроса. Обычно используется для выполнения команды, содержащей агрегирующие функции типа `COUNT()`, `MAX()` и т. д.

Реализуем самый популярный сценарий работы с базой данных (получение данных, просмотр с сортировкой, листанием и редактированием данных), не написав при этом ни строчки кода.

В режиме визуального редактирования, в панели элементов, необходимо найти элемент `SqlDataSource`, добавить его на страницу и указать источник данных (рис. 3.22).

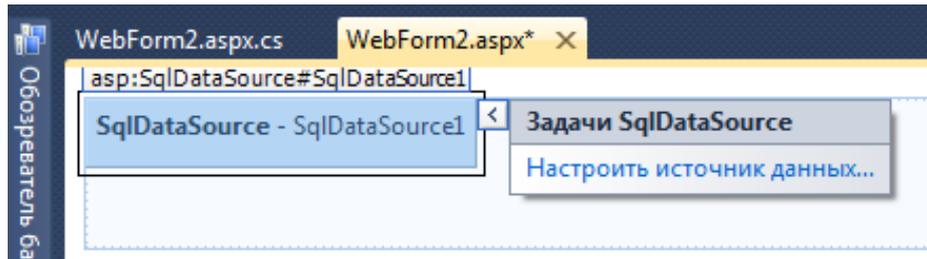


Рис. 3.22. Добавление элемента SqlDataSource

Необходимо выбрать нужную базу данных, указать таблицу и поля, с которыми должен работать элемент, указать генерацию инструкции по вставке, обновлению и удалению записей (рис. 3.23).

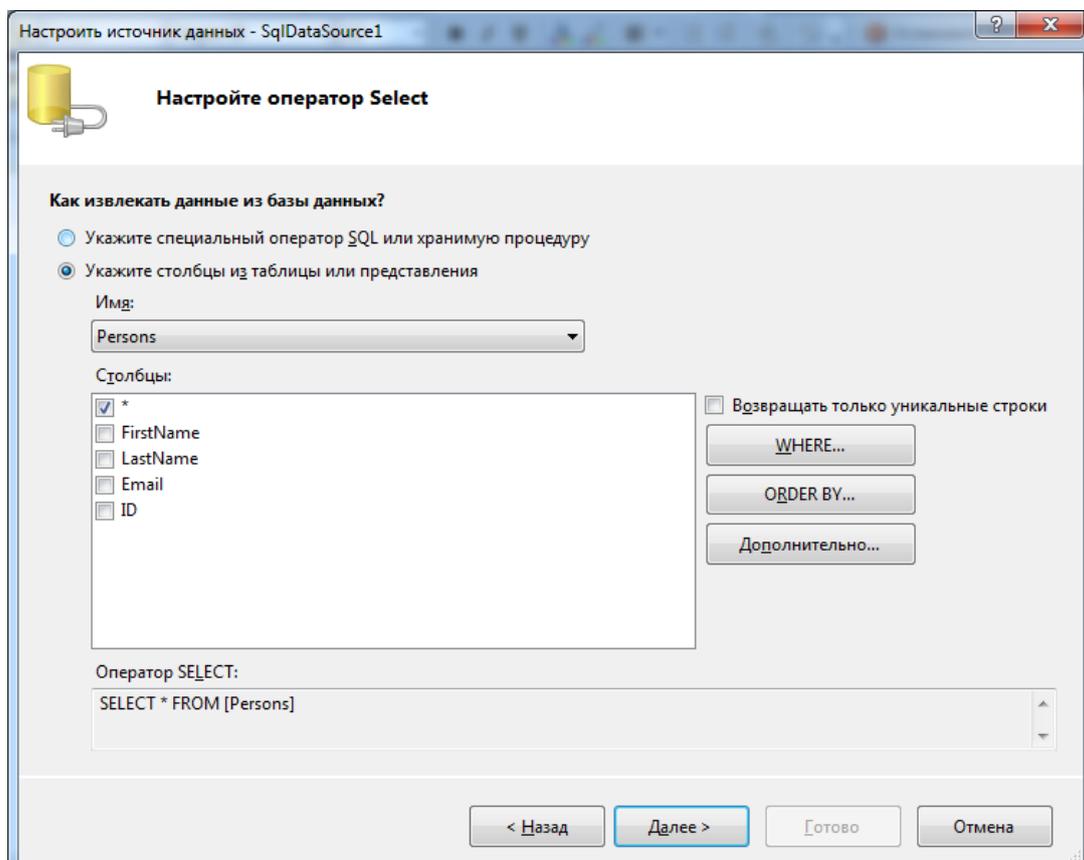


Рис. 3.23. Указание генерации инструкции по вставке, обновлению и удалению записей

С помощью этого элемента мы будем связываться с базой данных. Теперь необходимо указать элемент, в котором полученная информация будет отображаться, например GridView. Проассоциировать его с базой данных и включить опции редактирования, удаления, сортировки и разбиения на страницы (рис. 3.24).

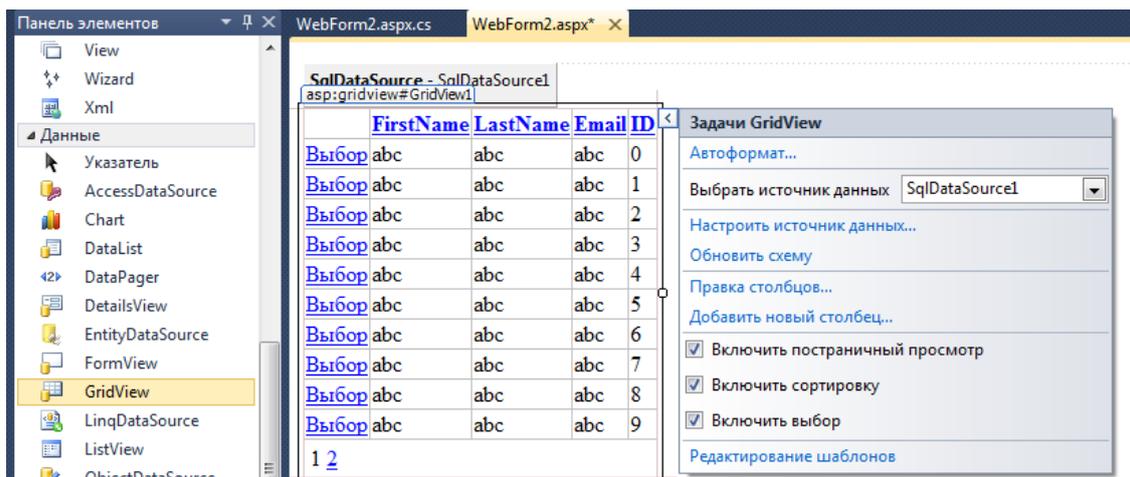


Рис. 3.24. Добавление элемента GridView

После запуска приложения появится таблица, позволяющая создавать и редактировать записи, сортировать данные по полям, переходить по страницам (рис. 3.25).

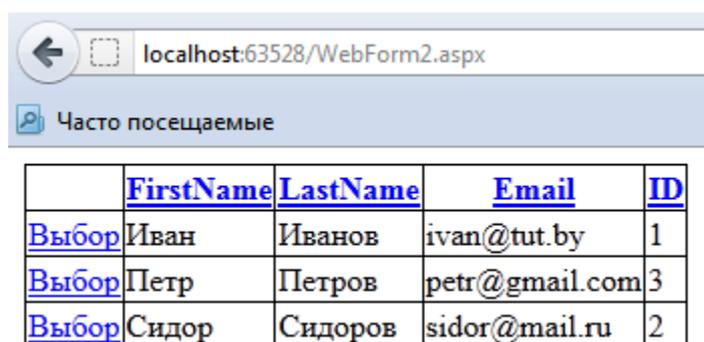


Рис. 3.25. Результат запуска приложения

### 3.7. Дизайн страниц

Шаблоны страниц – это мощное средство, помогающее быстро и качественно оформить внешний вид сайта, а также впоследствии легко изменять его. VWD предоставляет в распоряжение web-разработчика мастер-страницы, таблицы стилей и темы. Их использование позволяет быстро создать заготовку типового сайта.

Мастер-страницы служат шаблоном для отображения других страниц. Для этого на мастер-странице выделяются области, не подлежащие изменению, и области, где будет отображаться информация связанных страниц (страниц контента).

Контент-страница – любая страница, которая использует мастер-страницу. Каждый раз, когда посетитель запрашивает контент-

страницу, ASP.NET загружает мастер-страницу, производит слияние с контент-страницей и посылает объединенный результат пользователю. Слияние мастер-страницы и контент-страницы на лету имеет два важных последствия:

1) посетитель всегда получает текущие версии шаблона и его содержания;

2) полученная после слияния страница обладает всеми возможностями обычной ASP-страницы. Например, мастер-страница, как и контент-страница, может содержать любые элементы управления или фрагменты кода. Лучше всего разрабатывать мастер-страницу на этапе планирования сайта.

Чтобы создать сайт, использующий мастер-страницы, нужно выполнить следующие действия:

1. Создать новый сайт.

2. Удалить любые web-страницы, которые содержит сайт. Например, удалить страницу default.aspx, которая создается по умолчанию.

3. В окне Solution Explorer щелкнуть правой кнопкой мыши по папке сайта и выбрать Add New Item из контекстного меню.

4. Когда диалоговое окно Add New Item появится, выполнить следующие действия:

- выбрать из всех элементов списка Master Page;
  - оставить название страницы по умолчанию MasterPage.master;
  - выбрать язык программирования (по умолчанию C#);
  - поставить галочку напротив Place code in separate file;
5. Нажать Add, чтобы создать мастер-страницу.

Созданная мастер-страница доступна для редактирования за исключением одного элемента управления – ContentPlaceHolder, который создается на ней по умолчанию. Этот элемент резервирует пространство для содержимого контент-страницы. Редактирование мастер-страницы такое же, как и редактирование обычной ASP-страницы. Поэтому к ней можно применить все элементы управления, рассмотренные ранее.

Самый удобный способ использования мастер-страницы состоит в том, чтобы привязать ее к новой странице во время ее создания, что может быть выполнено следующим образом:

1. В окне Solution Explorer щелкнуть правой кнопкой мыши по папке, где необходимо разместить новую контент-страницу, затем выбрать Add New Item из контекстного меню.

2. В появившемся диалоговом окне Add New Item выбрать шаблон документа Web Form и поставить галочку напротив Select Master Page.

3. Щелкнуть на кнопку Add. В диалоговом окне Master Page выбрать мастер-страницу.

4. Чтобы создать страницу, нажмите кнопку ОК.

При работе с контент-страницей в режиме конструктора, части, которые добавились из мастер-страницы, будут недоступны для редактирования. Чтобы изменить что-нибудь в них, необходимо открыть и отредактировать мастер-страницу.

Чтобы применить CSS для определенного элемента страницы, необходимо выделить его в режиме конструктора и затем выбрать пункт Style из меню Format или в окне Properties. Свойства CSS сгруппированы в восемь категорий, которые расположены по левому краю диалогового окна. Применение свойств CSS к индивидуальным элементам страницы обеспечивает большую гибкость в оформлении.

При разработке дизайна предпочтительно создавать файлы таблиц стилей. Для создания файла таблицы стилей необходимо выбрать New File из меню File или щелкнуть правой кнопкой мыши по папке в Solution Explorer, выбрать пункт Add New Item. В появившемся диалоговом окне выбрать шаблон Style Sheet, а затем нажать Add.

В файле таблицы стилей содержатся правила, каждое из которых состоит из названия, сопровождаемого списком свойств CSS и значений. Есть два способа добавить новое правило стиля к файлу таблицы стилей (если имеется опыт) – напечатать самостоятельно или выбрать пункт Add Style Rule из меню Styles, что приведет к появлению диалогового окна Add Style Rule.

При использовании Add Style Rule необходимо выбрать тип стиля: Element, Class Name или Element ID. Выбрать или напечатать название и, если создается правило класса, ограничить его применение определенным названием дескриптора. Например, правило стиля по имени TABLE.test применяется только к дескрипторам, закодированным как <TABLE class = "test">.

Как только правило стиля создано, можно добавить к нему свойства одним из трех способов:

- 1) напечатав их самостоятельно или с помощью IntelliSense;
- 2) выбрав правило стиля и выбрав Build Style из меню Styles;
- 3) щелкнув правой кнопкой мыши по стилю и выбрав Build Style из контекстного меню.

Чтобы использовать созданный файл CSS, нужно связать его с web-страницей, открыв ее в режиме Design, и из окна Solution Explorer поместить на нее CSS.

## 4. ПУБЛИКАЦИЯ САЙТОВ

Для публикации сайтов используются web-серверы. В рамках курса «Организация web-портала и администрирование ресурсов в web» изучается практическое использование web-сервера IIS (Internet Information Service). Рассмотрим основные моменты публикации динамических сайтов с помощью данного web-сервера.

### 4.1. Установка Microsoft IIS 6.0

Установить IIS можно с помощью *Мастера настройки сервера (Manage Your Server)*. Его можно запустить следующим образом: в меню *Пуск* выберите *Все программы*, затем *Администрирование* и *Мастер настройки сервера* (рис. 4.1).

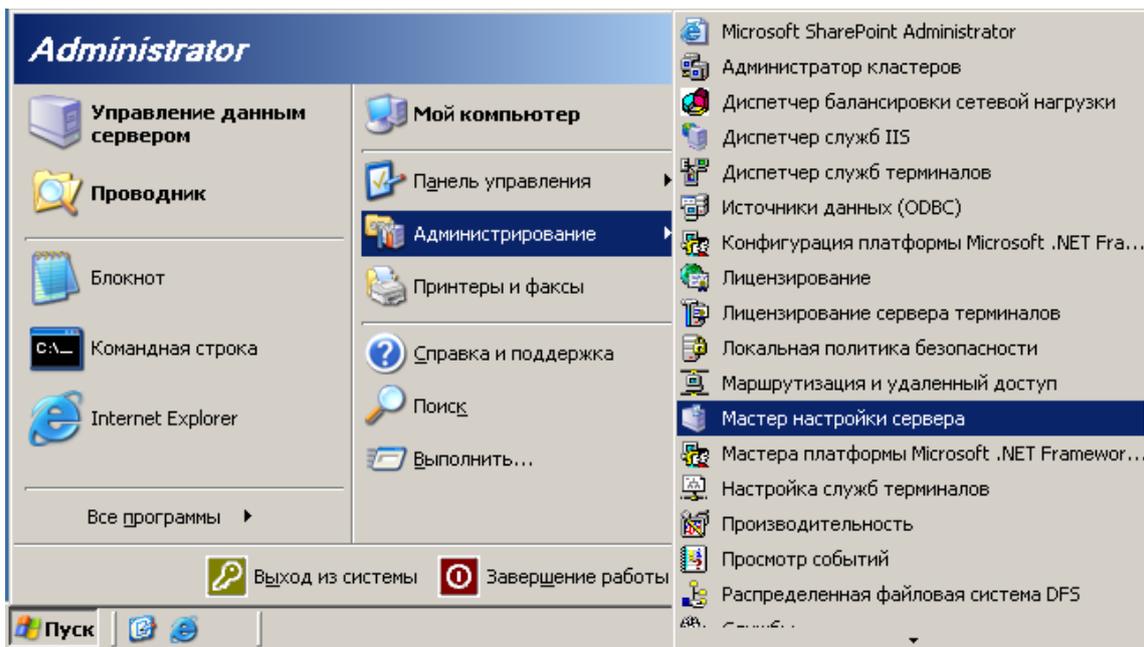


Рис. 4.1. Пример запуска *Мастера настройки сервера*

В этом окне можно добавить новую роль серверу, что позволит настроить его на выполнение определенной задачи. Также с помощью этого окна можно осуществлять управление текущей ролью сервера. Мастер установки протестирует ваши доступные и активированные сетевые подключения и выведет окно *Роль сервера* (рис. 4.2).

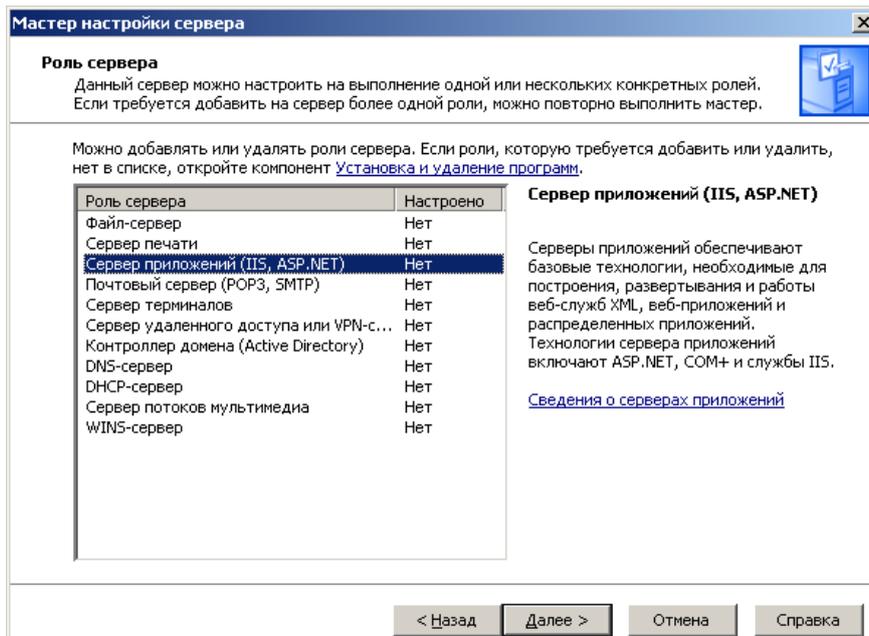


Рис. 4.2. Выбор устанавливаемой роли сервера

Итак, в окне *Мастер настройки сервера* необходимо выбрать роль *Сервер приложений* (web-сервер IIS) и щелкнуть кнопку *Далее* для продолжения. Для роли *Сервер приложений* следующим окном будет страница *Параметры сервера приложений* (рис. 4.3), позволяющая выбрать (или не выбирать) дополнительные инструменты, такие как серверные расширения FrontPage или ASP.NET, активировав соответствующую опцию перед тем, как продолжить.

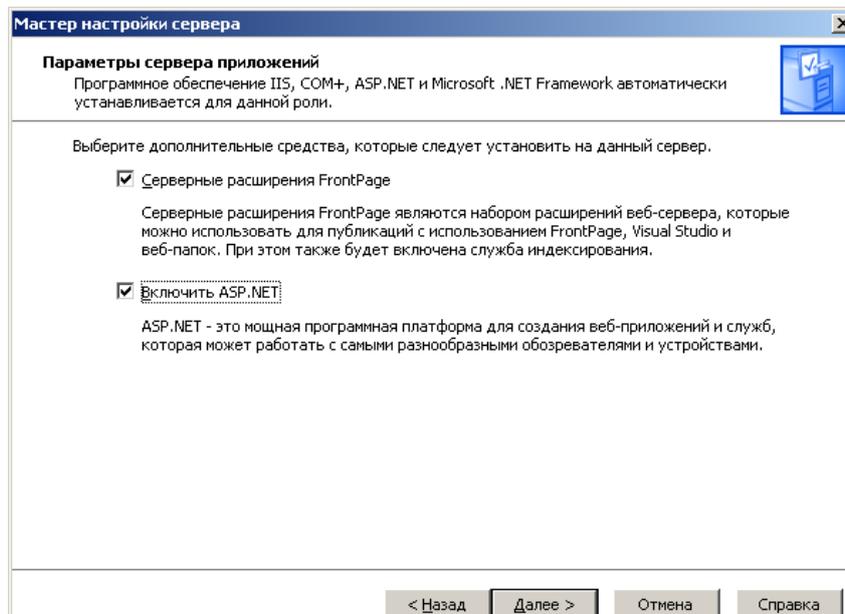


Рис. 4.3. Выбор параметров сервера приложений

Следующее окно *Сводка выбранных параметров* (рис. 4.4) показывает те опции, которые были выбраны для установки. Как видно из рисунка, не было предоставлено возможности выбора установки таких дополнительных служб, как FTP, NNTP или SMTP, и они не устанавливаются по умолчанию при назначении этой серверной роли данным способом.

Для того чтобы определить дополнительные службы или другие индивидуальные настройки, необходимо запустить полную установку с помощью программы Add or Remove Programs из панели управления.

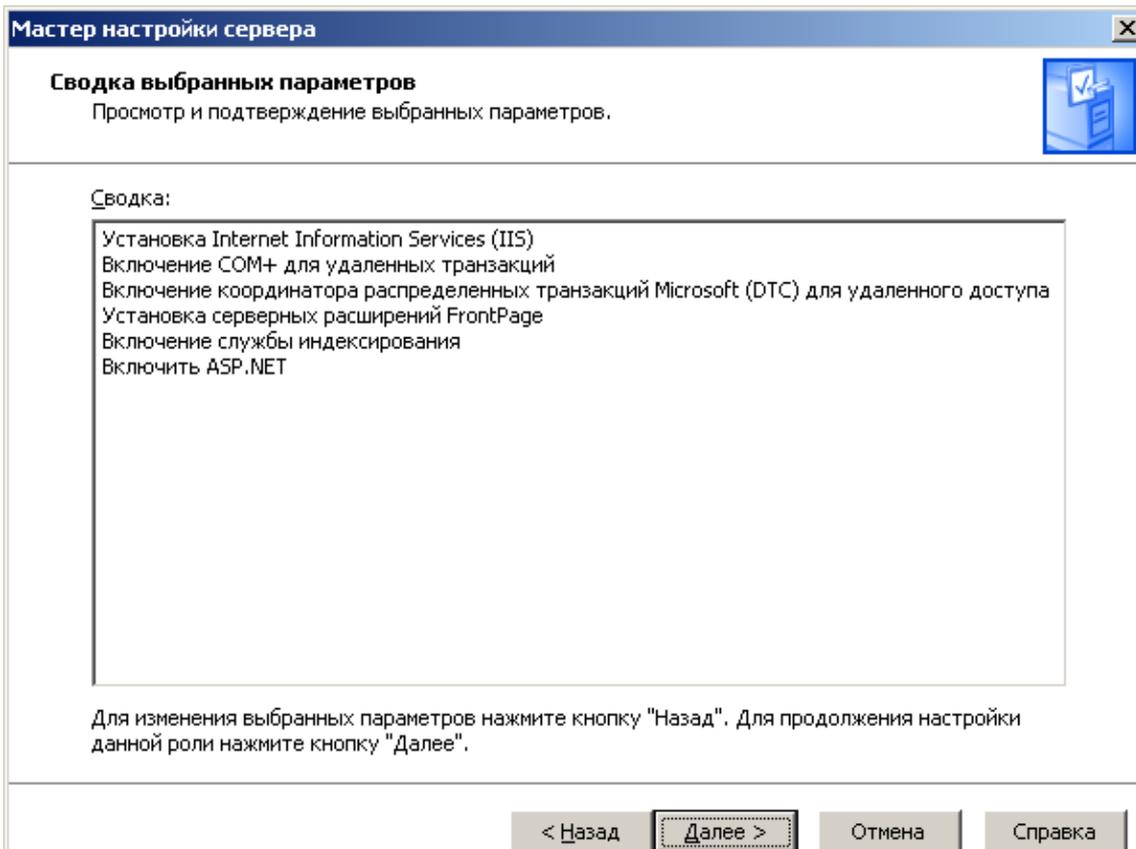


Рис. 4.4. Окно с информацией о выбранных параметрах

Затем IIS 6.0 устанавливается и конфигурируется автоматически *Мастером настройки сервера* без какого-либо дальнейшего участия пользователя. При необходимости, предоставьте системе доступ к дистрибутиву операционной системы, с которого она была установлена. После того как процесс будет завершен, заключительное окно *Мастера настройки сервера* сообщит, что "Данный сервер теперь является сервером приложений" (рис. 4.5).

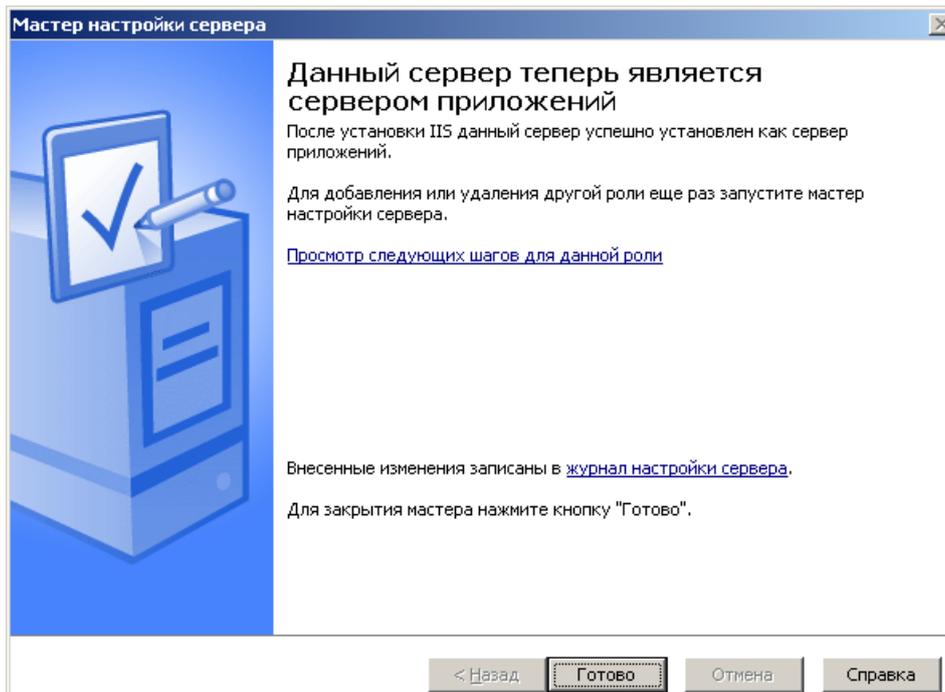


Рис. 4.5. Завершение настройки сервера

Запуск web-сервера IIS осуществляется следующим образом (рис. 4.6).

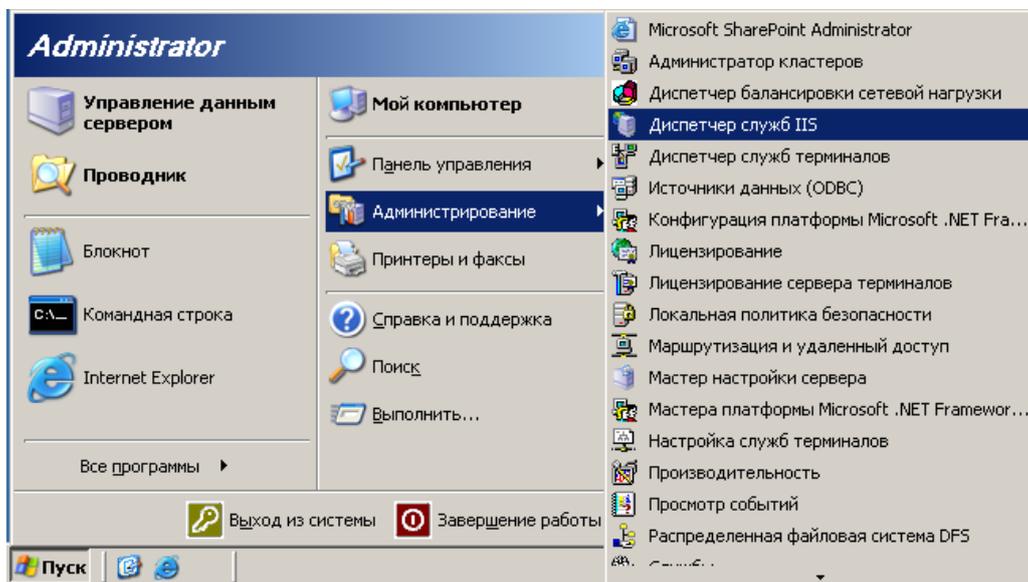


Рис. 4.6. Пример запуска web-сервера IIS

Легко убедиться, что установлена только одна служба – World Wide Web (рис. 4.7). FTP, NNTP и SMTP не добавляются по умолчанию при установке серверной роли данным способом.

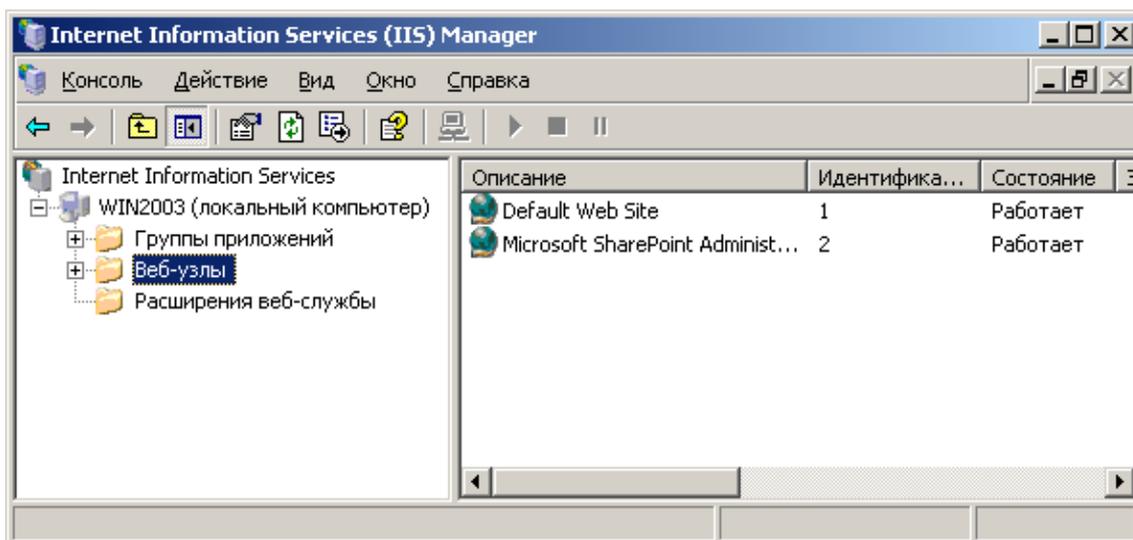


Рис. 4.7. Консоль управления web-сервера IIS

## 4.2. Публикация сайта, разработанного на платформе ASP с использованием языка C#

Прежде чем воспользоваться *Мастером создания web-узлов*, необходимо решить, какое имя в сети будет иметь этот сайт. При использовании web-браузера (например, Internet Explorer) для соединения с web-узлом и просмотра его домашней страницы можно задать URL сайта различными способами, например при помощи IP-адреса, или NetBIOS-имени, или полностью определенного DNS-имени.

Для web-сайтов рекомендуется использовать альтернативные IP-адреса сетевой интерфейсной платы сервера Win2003. В дальнейшем необходимо установить соответствие между вновь создаваемым web-узлом и добавляемым новым альтернативным IP-адресом. Чтобы добавить альтернативный IP-адрес (например, 192.168.0.30), необходимо выполнить следующие действия:

- нажмите кнопку *Пуск*, выберите меню *Подключение* → *Отобразить все подключения*;
- в окне *Сетевые подключения* щелкните правой кнопкой мыши значок *Подключение по локальной сети* и выберите *Свойства* в контекстном меню (или дважды щелкните этот значок и далее нажмите кнопку *Свойства*);
- в окне *Подключение по локальной сети* дважды щелкните *Протокол Интернета (TCP/IP)*;
- в окне *Свойства: Протокол Интернета (TCP/IP)* щелкните кнопку *Дополнительно*;

– в разделе *IP-адреса* диалогового окна *Дополнительные параметры TCP/IP* щелкните кнопку *Добавить*, задайте дополнительный IP-адрес и маску подсети и щелкните кнопку *Добавить*;

– закройте все диалоговые окна, нажимая кнопку *ОК*.

Далее создайте новый web-узел с именем MySite, выполнив следующие действия.

1. Запустите консоль Internet Information Services на сервере Win2003, а затем выберите имя сервера в дереве консоли.

2. Щелкните кнопку *Действие* на панели инструментов, нажмите *Создать* и выберите *Web-узел* в раскрывающемся меню (рис. 4.8).

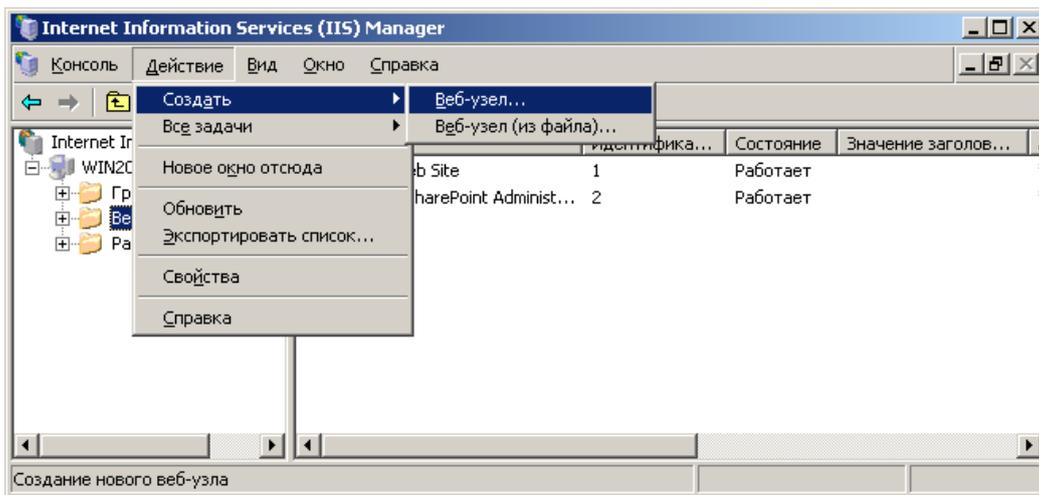


Рис. 4.8. Вызов *Мастера создания web-узлов*

Откроется окно *Мастера создания web-узлов*. Щелкните кнопку *Далее* (рис. 4.9).

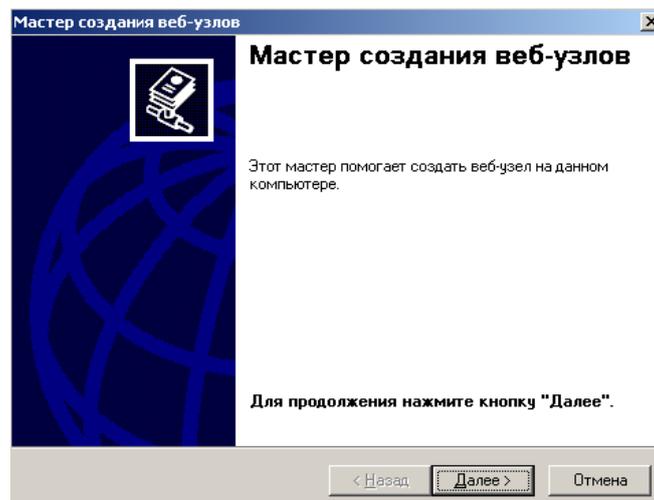


Рис. 4.9. Начало работы *Мастера создания web-узлов*

3. Чтобы задать обозначение узла, наберите название **MySite** (рис. 4.10). Это имя будет отображаться в консоли Internet Information Services и будет служить обозначением нового web-узла для администратора. Затем щелкните кнопку *Далее*.

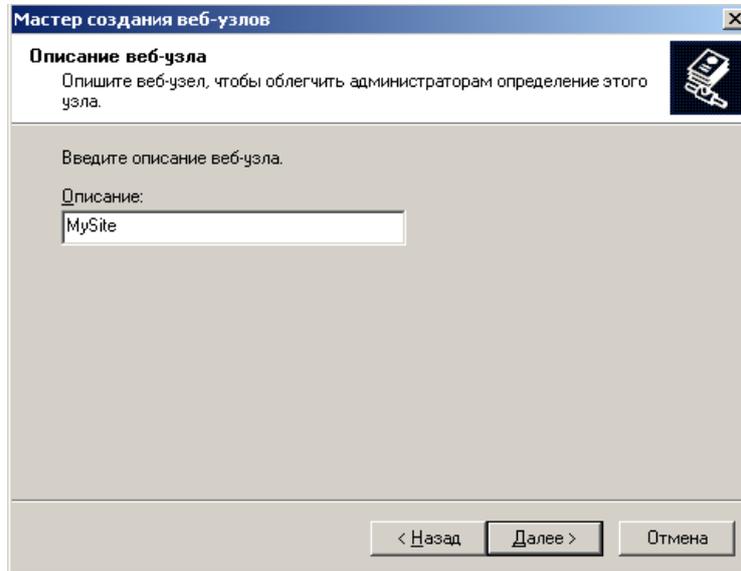


Рис. 4.10. Присвоение названия web-узлу

4. В поле *Введите IP-адрес для веб-узла* задайте любой вторичный IP-адрес (например, 192.168.0.30) в качестве IP-адреса для данного сайта, не изменяя других настроек (рис. 4.11). Затем щелкните кнопку *Далее*.

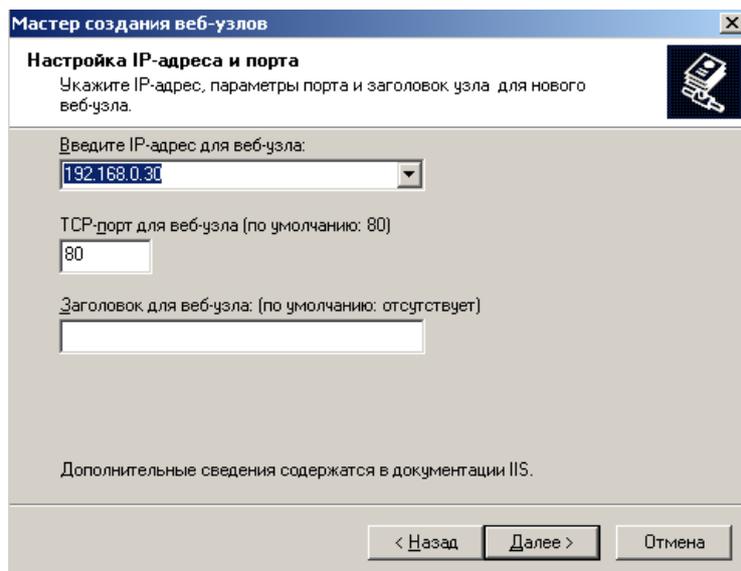


Рис. 4.11. Присвоение IP-адреса web-узлу

5. На следующем шаге мастера задайте путь к домашнему каталогу данного web-узла как C:\MySite (удобнее всего воспользоваться кнопкой *Обзор*) (рис. 4.12). Обратите внимание, что домашний каталог сайта может быть как локальным каталогом, так и сетевым разделяемым ресурсом. Флажок *Разрешить к веб-узлу анонимный доступ* оставьте установленным. В выбранном каталоге должны размещаться все необходимые файлы сайта (в примере это один файл Default.htm). Щелкните кнопку *Далее*.

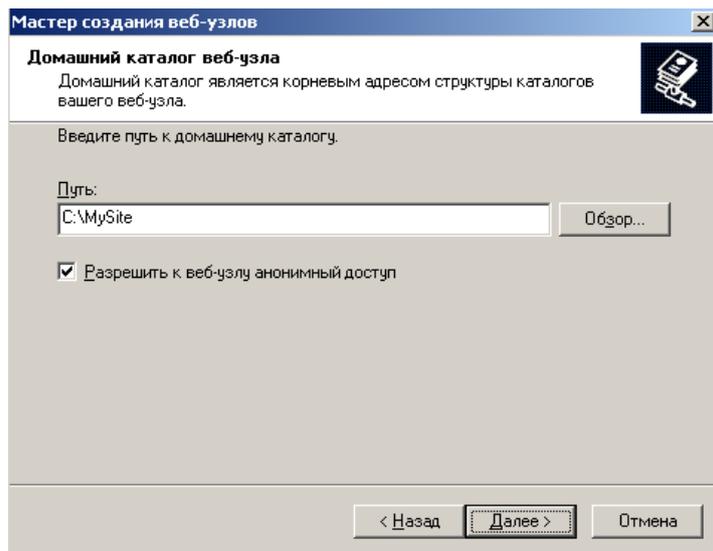


Рис. 4.12. Задание домашнего каталога сайта

6. Задайте настройки полномочий доступа, как показано на рис. 4.13, и щелкните кнопку *Далее*.

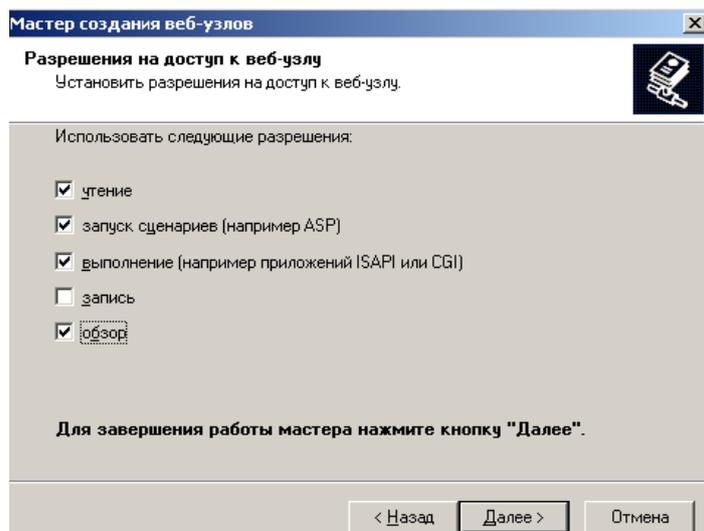


Рис. 4.13. Определение разрешений на доступ к web-узлу

7. Чтобы завершить работу мастера, щелкните кнопку *Готово* (рис. 4.14).

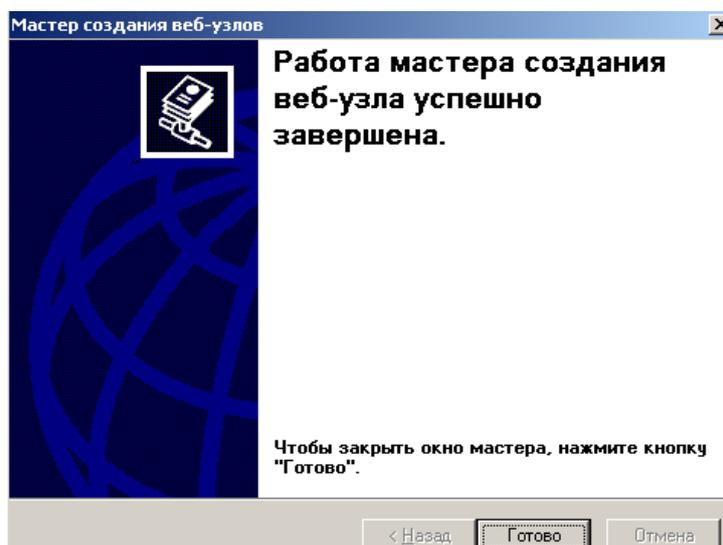


Рис. 4.14. Завершение работы *Мастера создания web-узлов*

Новый web-узел **MySite** появится в качестве узла в окне консоли *Internet Information Services* под обозначением сервера Win2003, и в нем будет только одна web-страница – *Default.htm* (рис. 4.15).

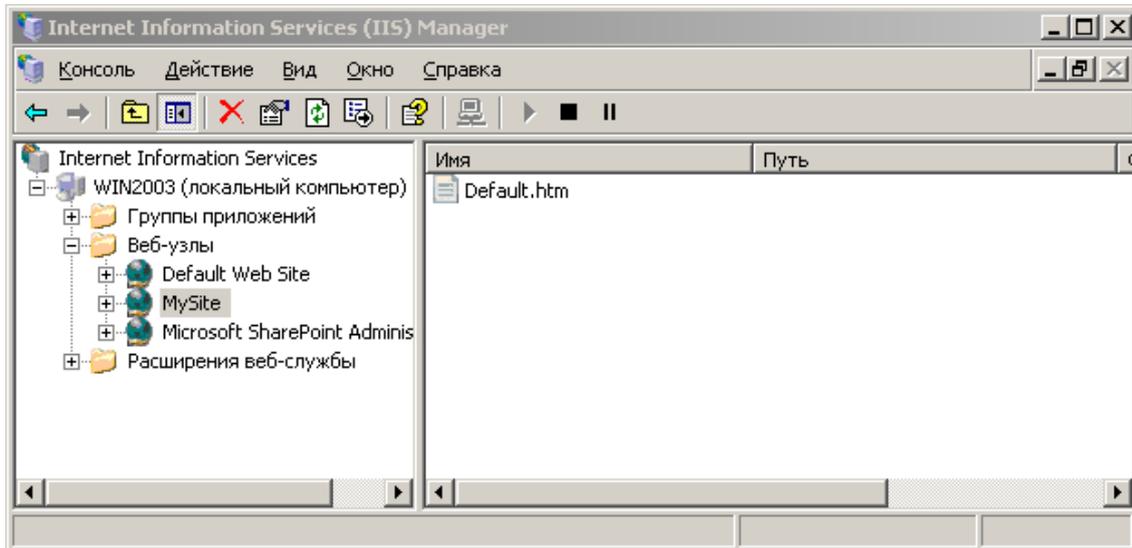


Рис. 4.15. Пример консоли IIS с установленным web-узлом MySite

8. Чтобы проверить работу нового web-узла, запустите браузер и введите `http://192.168.100.30` в адресную строку. В окне браузера должна загрузиться страница *Default.htm* (рис. 4.16). Если для исполь-

зуюемого IP-адреса определено DNS-имя, то для обращения к данному web-узлу можно вместо IP-адреса использовать это имя.

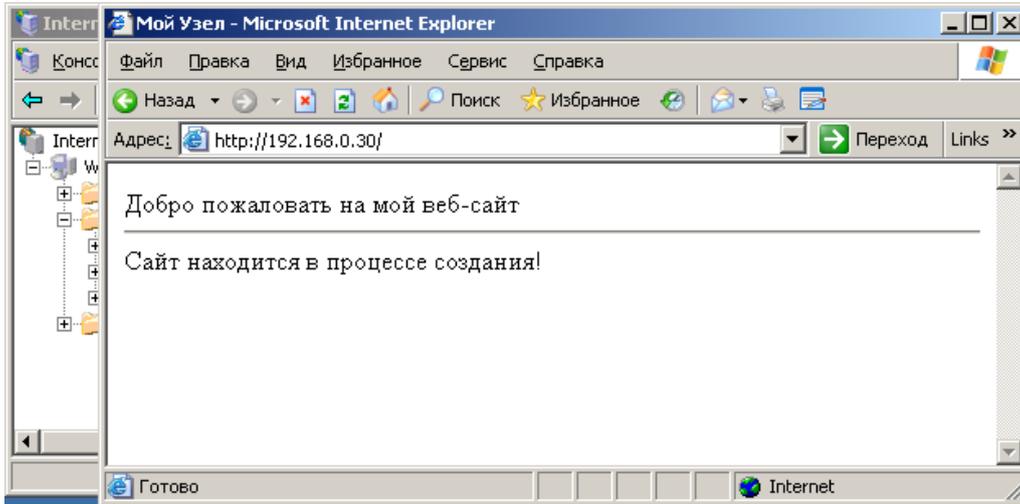


Рис. 4.16. Пример работы web-сайта

9. Для корректной работы сайта, разработанного на платформе ASP, в настройках созданного web-узла укажите в качестве стартовой соответствующую страницу, как, например, показано на рис. 4.17, а также разрешите выполнение расширений ASP.NET, как показано на рис. 4.18 (отмечено выделением).

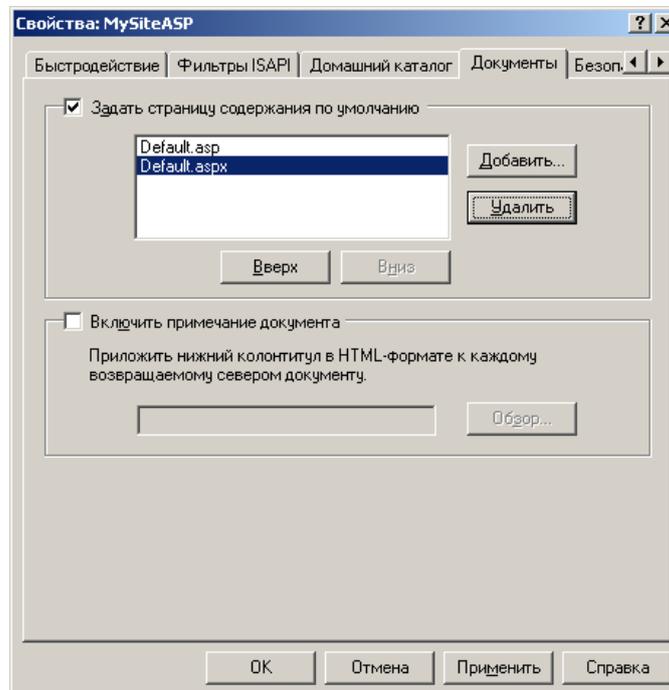


Рис. 4.17. Задание стартовой страницы

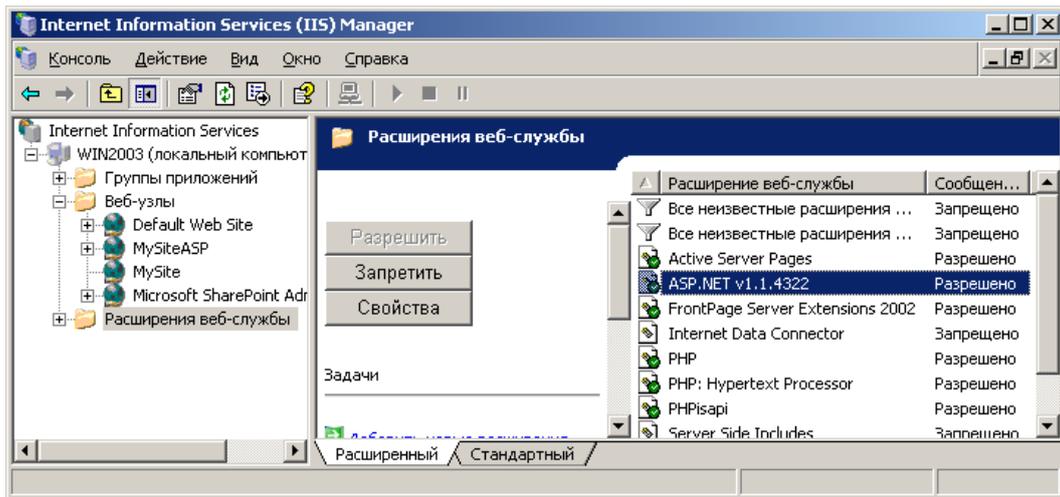


Рис. 4.18. Разрешение использования ASP.NET

10. В итоге получим опубликованный сайт, как показано на рис. 4.19.

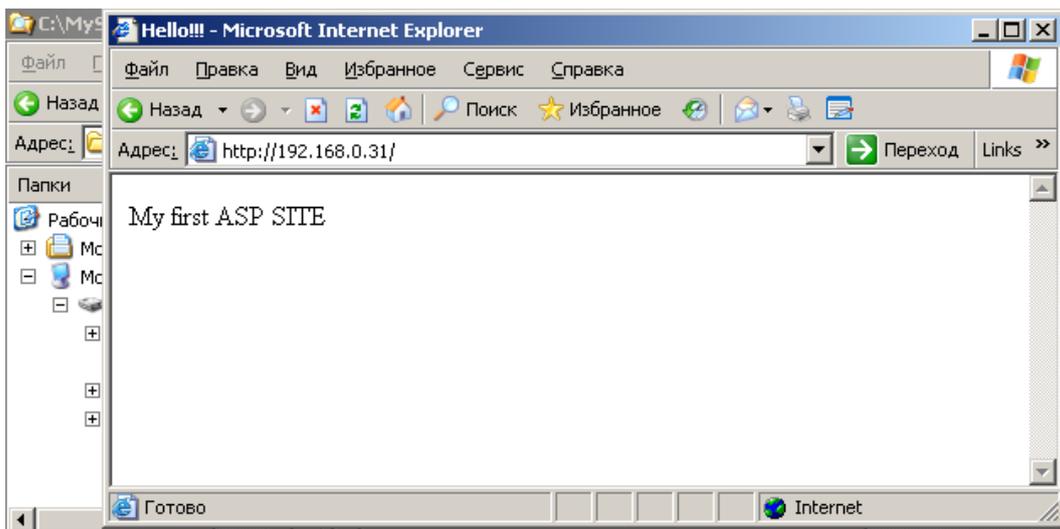


Рис. 4.19. Пример работы web-сайта на ASP

Для сайта на платформе ASP аналогичным образом был создан web-узел с названием MySiteASP, которому был присвоен IP-адрес 192.168.0.31.

## 5. ТЕСТИРОВАНИЕ WEB-ПРИЛОЖЕНИЯ

Инструмент разработки программного обеспечения Microsoft Visual Studio 2010 предоставляет множество способов тестирования приложений с целью проверки их функциональности и производительности.

### 5.1. Основные типы тестов в Visual Studio

К основным типам тестов, реализованных в Microsoft Visual Studio 2010, относятся:

1) Basic Unit Test – модульный программный тест, при выполнении которого вызываются методы класса и проверяются возвращаемые ими значения;

2) Unit Test – то же самое, что и Basic Unit Test. Разница между Basic Unit Test и Unit Test заключается в том, что Unit Test предоставляет разработчику более сложный конструктор для создания модульного теста;

3) Unit Test Wizard – мастер создания модульного теста. Позволяет быстро сгенерировать тест с помощью мастера, затем полученный код можно редактировать;

4) Coded Ui Test – данный вид тестов предназначен для автоматизации функциональных тестов и тестирования пользовательского интерфейса;

5) Generic Test – специальный вид тестов, который позволяет выполнять внешние тестирующие приложения;

6) Web Performance Test – функциональное тестирование web-приложений в рамках организации нагрузочных тестов (Load Test);

7) Load Test – тесты для проведения нагрузочного тестирования web-приложений;

8) Ordered Test – тесты, которые позволяют организовать выполнение всех автоматических тестов в определенном порядке;

9) Database Unit Test – модульные тесты для тестирования баз данных.

Для тестирования web-сайтов наибольший интерес представляют Web Performance Test и Load Test. В случае, если возможностей Visual Studio не будет хватать для создания автоматизированных тестов, можно выполнять внешние тестирующие приложения с помощью Generic Test.

Для создания нового теста необходимо в главном меню выбрать из меню *Test* команду *New Test...*, после этого появится форма мастера создания нового теста. На данной форме отображается перечень доступных типов тестов для проекта. Сперва необходимо выбрать требуемый вид теста из списка, затем новый тест нужно связать с существующим проектом или создать для него новый проект тестирования кода, написанного на языке C++, C# или Microsoft Visual Basic.

## 5.2. Создание теста web-производительности

Тест web-производительности генерирует HTTP-запросы и фиксирует время ожидания ответа для каждого из них.

1. Откройте Microsoft Visual Studio, расположенную в Start | All Programs | Microsoft Visual Studio 2010.

2. Создайте новый проект, воспользовавшись меню (File | New | Project...).

3. В открывшемся окне New Project мастера создания новых проектов в меню Visual C# | Test выберите тип проекта Test Project и нажмите ОК (рис. 5.1).

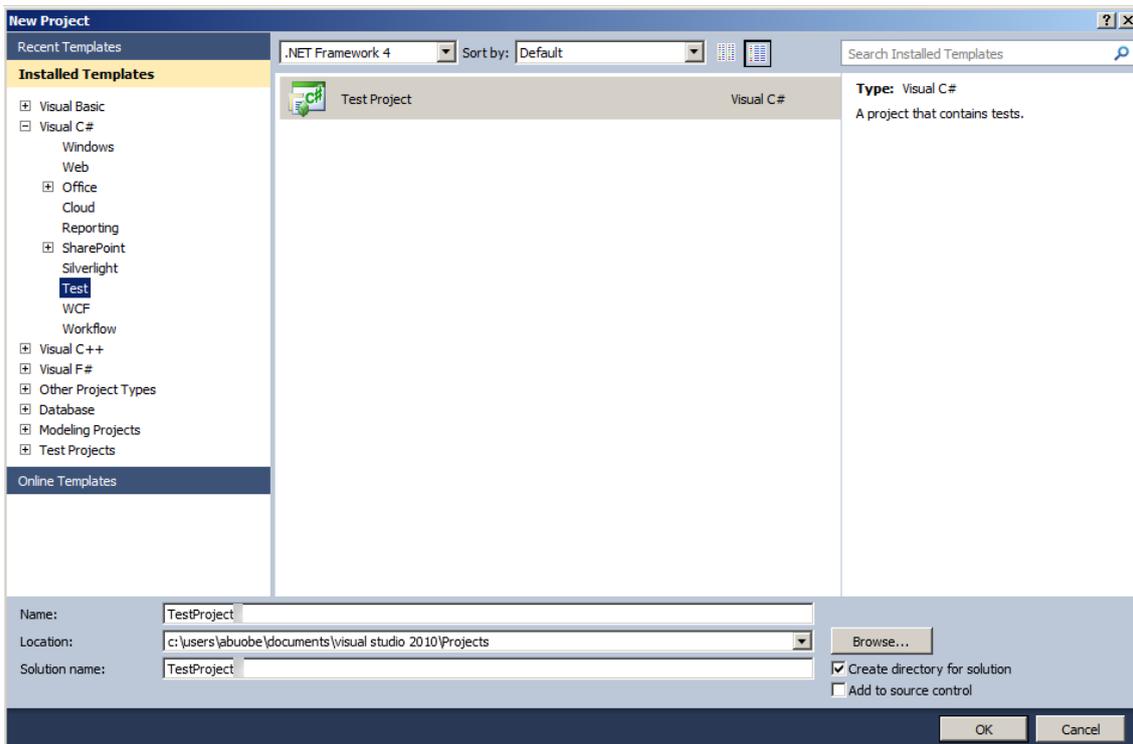


Рис. 5.1. Создание тестового проекта

4. Закройте отобразившуюся вкладку UnitTest1.cs.

5. В Solution Explorer на элементе Test Project дерева решения щелкните правой кнопкой мыши и в контекстном меню выберите Add | Web Performance Test... (рис. 5.2). В результате в дерево проекта добавится файл WebTest1.webtest, содержащий описание теста web-производительности, при этом произойдет автоматический запуск браузера Internet Explorer с интегрированным в него Web Test Recorder.

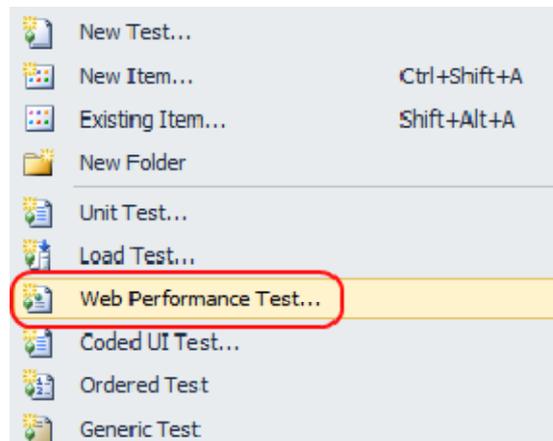


Рис. 5.2. Добавление в проект нового теста web-производительности

Web Test Recorder запускается с Internet Explorer в качестве Browser Helper Object (BHO).

6. Web Test Recorder запустится в режиме записи действий. Убедиться в этом вы можете, посмотрев на панель инструментов, расположенную в левой верхней части окна браузера (рис. 5.3). В любой момент вы можете приостановить (Pause) или остановить (Stop) процесс записи, нажав соответствующую кнопку.

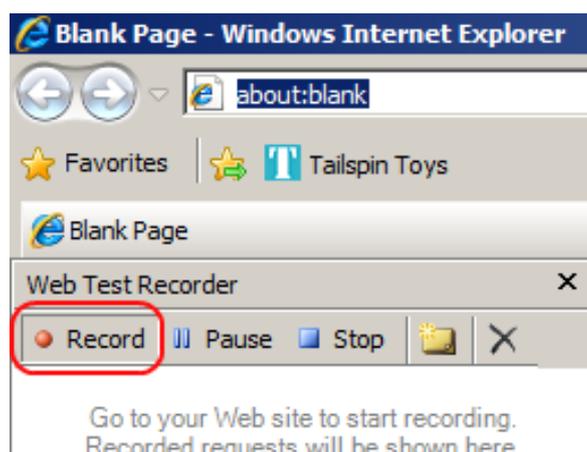


Рис. 5.3. Web Test Recorder, запущенный по умолчанию в режиме записи действий

Необходимо отметить, что пока Web Test Recorder находится в состоянии Record, все задержки между действиями пользователя будут интерпретироваться как «время ожидания». Например, записывая предполагаемый сценарий взаимодействия пользователя с сайтом, на странице продуктов сделайте паузу, подождите несколько секунд, как будто бы вы проверяете данные. Данный тест необходимо создавать не только для замера web-производительности – он будет являться элементом нагрузочного теста, который будет создан далее.

7. Чтобы загрузить web-сайт, в Internet Explorer в панели Избранное выберите закладку с названием сайта. Обратите внимание, что Web Test Recorder зафиксировал запрос.

8. Осуществите навигацию по сайту, представив себя в роли пользователя сайта.

9. На панели инструментов Web Test Recorder нажмите кнопку Stop (рис. 5.4).



Рис. 5.4. Остановка работы Web Test Recorder

10. После остановки записи web-тестов Visual Studio запускает две задачи обработки сгенерированных данных: обработка и сохранение динамических параметров (Dynamic parameters).

Под динамическими параметрами понимаются параметры, значения которых определяются только в процессе записи взаимодействий с web-сайтом. В качестве примера таких параметров можно привести: ASP.NET view state, GET/POST-параметры.

Сохраненные динамические параметры используются для воспроизведения первоначально записанного сценария при повторных запусках теста. Далее тест запускается на выполнение и формирование отчета.

11. Для просмотра детальной информации о тесте щелкните правой кнопкой мыши на записи в окне Test Results и в контекстном меню выберите View Test Results Details (рис. 5.5).

Получить детальную информацию о выполнении теста вы также можете, дважды нажав на запись в окне Test Results.

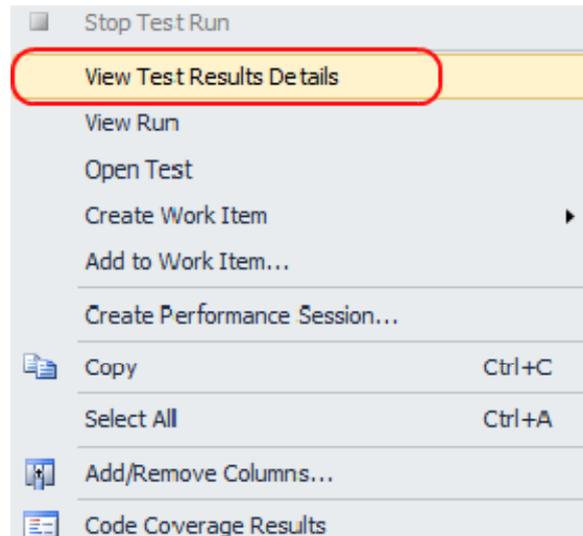


Рис. 5.5. Просмотр детальной информации о результатах теста

12. В окне Test Results Details приводится последовательность HTTP-запросов, ответов в рамках теста Visual Studio и другие детали теста (рис. 5.6). По умолчанию выделяется запись, характеризующая первый запрос/ответ в рамках данного теста.

Passed		<a href="#">Click here to run again</a>	Internet Explorer 7.0 LAN	<a href="#">Edit run settings</a>
	Request			Status
✓	<a href="http://win-gs9gmujts8:8000/">http://win-gs9gmujts8:8000/</a>			200 OK
✓	<a href="http://win-gs9gmujts8:8000/">http://win-gs9gmujts8:8000/</a>			200 OK
✓	<a href="http://win-gs9gmujts8:8000/home/show">http://win-gs9gmujts8:8000/home/show</a>			200 OK
✓	<a href="http://win-gs9gmujts8:8000/Cart/AddItem">http://win-gs9gmujts8:8000/Cart/AddItem</a>			302 Found
✓	<a href="http://win-gs9gmujts8:8000/Cart/Show">http://win-gs9gmujts8:8000/Cart/Show</a>			200 OK
✓	<a href="http://win-gs9gmujts8:8000/Order/Checkout">http://win-gs9gmujts8:8000/Order/Checkout</a>			200 OK
✓	<a href="http://win-gs9gmujts8:8000/Order/Address">http://win-gs9gmujts8:8000/Order/Address</a>			302 Found
✓	<a href="http://win-gs9gmujts8:8000/Order/Finalize">http://win-gs9gmujts8:8000/Order/Finalize</a>			200 OK
✓	<a href="http://win-gs9gmujts8:8000/Order/Payment">http://win-gs9gmujts8:8000/Order/Payment</a>			302 Found

Рис. 5.6. Окно Test Result Details, содержащее детальную информацию о запуске теста

### 5.3. Нагрузочное тестирование

Тесты web-производительности, наиболее полно описывающие сценарии поведения пользователей, являются основой для проведения качественного нагрузочного тестирования. Отметим, что базой для данного теста будет выступать тест web-производительности, созданный ранее. Для создания нагрузочного теста необходимо будет определить пользовательскую нагрузку, выбрать перечень тестов web-

производительности, указать тип сети, эмулируемый браузер, счетчики производительности и другие показатели, статистику по которым нужно собрать в ходе выполнения теста.

1. Чтобы запустить мастер создания нагрузочного теста, в Solution Explorer щелкните правой кнопкой мыши на проекте Web Test и в контекстном меню выберите Add | Load Test... (рис. 5.7).

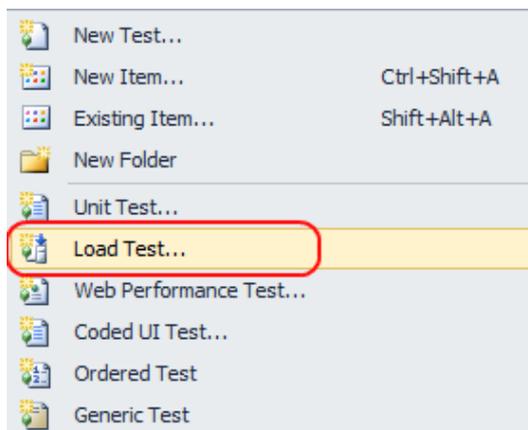


Рис. 5.7. Создание нагрузочного теста

2. Первый экран мастера создания нагрузочного теста является экраном приветствия. Для перехода к поэтапной настройке сценария нагрузочного теста нажмите кнопку Next (рис. 5.8).

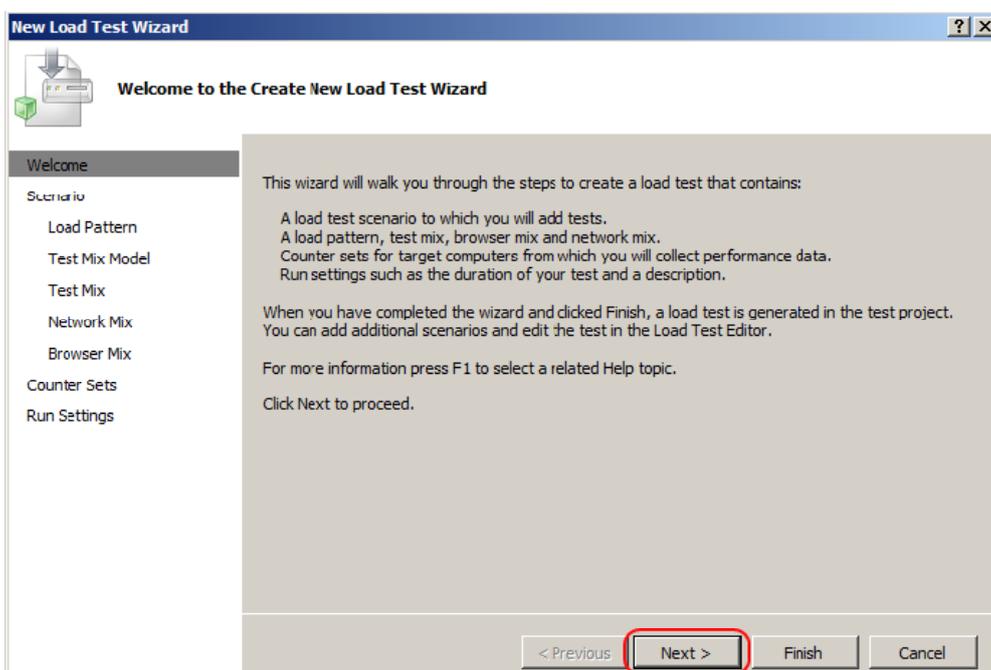


Рис. 5.8. Мастер создания нагрузочного теста с поэтапной настройкой

3. Введите название, наиболее точно отражающее суть данного нагрузочного теста, при этом профиль ожидания следующего действия оставьте в том положении, которое было установлено по умолчанию. Обратите внимание, что по умолчанию в качестве времени задержки выполнения запросов используются временные интервалы, которые определяются как среднее значение интервалов на предыдущей итерации с добавлением случайной величины, сгенерированной по нормальному закону распределения.

4. Чтобы перейти к этапу определения Load Pattern, нажмите кнопку Next.

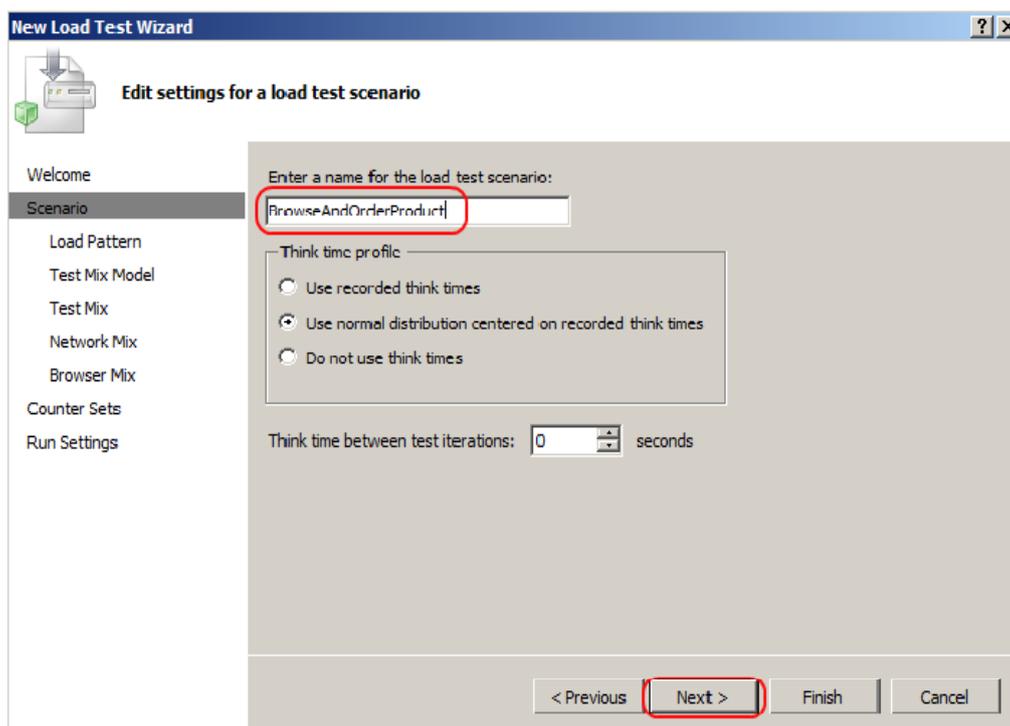


Рис. 5.9. Начальный этап настройки нагрузочного тестирования

5. В рамках данного нагрузочного теста будем использовать постоянную нагрузку (как было установлено по умолчанию), но изменим количество пользователей на 5 (Visual Studio 2010 Ultimate предоставляет возможность использования до 250 виртуальных пользователей в рамках только одного ядра процессора). Важно сбалансировать моделируемое количество пользователей так, чтобы машина имела достаточно ресурсов на запуск IIS и обеспечение нагрузочного тестирования на одной и той же машине.

6. Для перехода к этапу Test Mix Model для определения стратегии смешивания тестов нажмите кнопку Next (рис. 5.10).

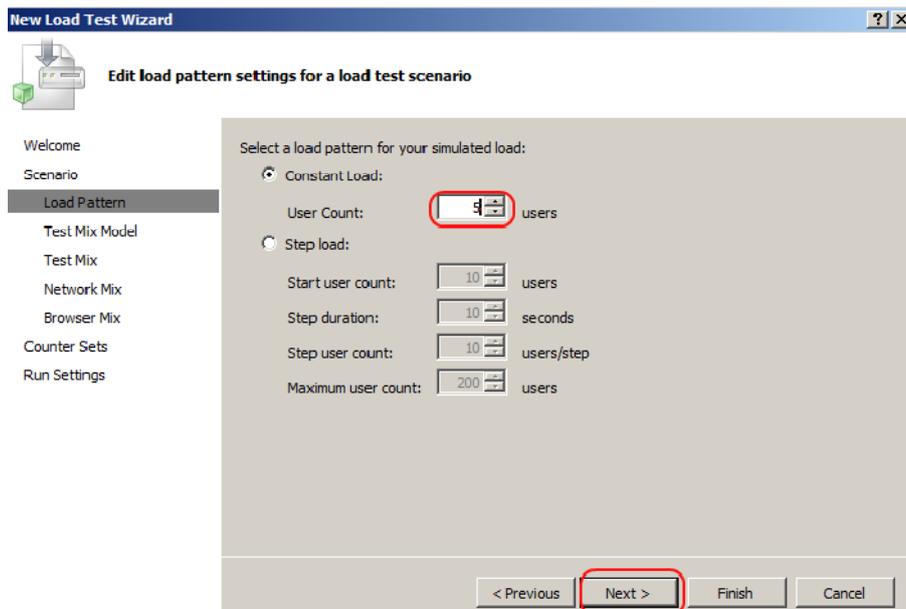


Рис. 5.10. Настройка числа виртуальных пользователей, которые будут генерировать нагрузку в течение проведения тестирования

7. Ознакомьтесь с описанием каждой стратегии смешивания тестов. Для этого последовательно выбирайте стратегию, и в правой части окна мастера будет появляться ее описание (рис. 5.11).

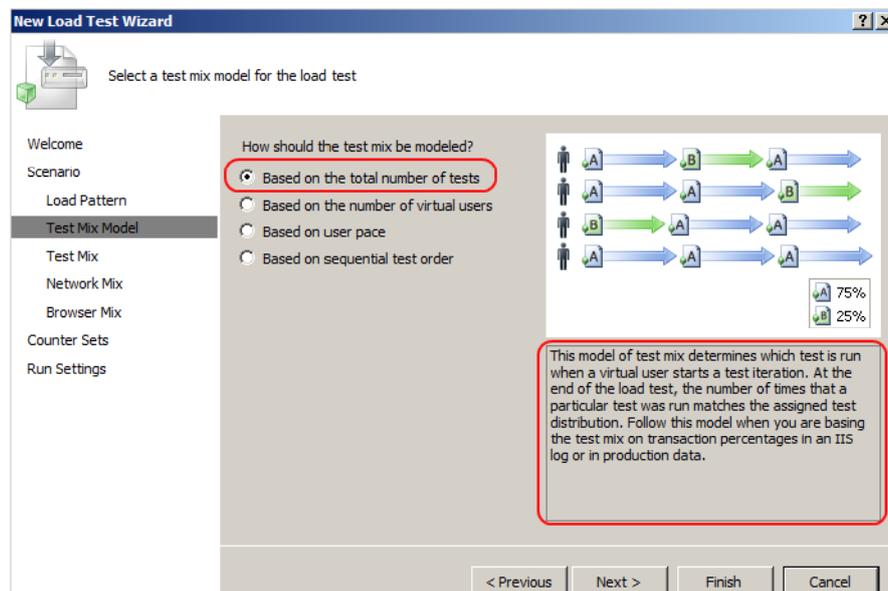


Рис. 5.11. Выбор стратегии смешивания тестов

8. Выберите первую стратегию смешивания тестов Based on the total number of tests и нажмите на кнопку Next. После чего появится интерфейс с выбором тестов для нагрузочного тестирования.

9. Для того чтобы открыть окно Add Test, нажмите кнопку Add.

10. Для включения теста в список тестов, которые будут выполняться в рамках данного нагрузочного тестирования, дважды щелкните на него. В списке доступных тестов может находиться тест, сгенерированный средой разработки при создании тестового проекта.

11. Для завершения процесса добавления тестов нажмите кнопку ОК (рис. 5.12).

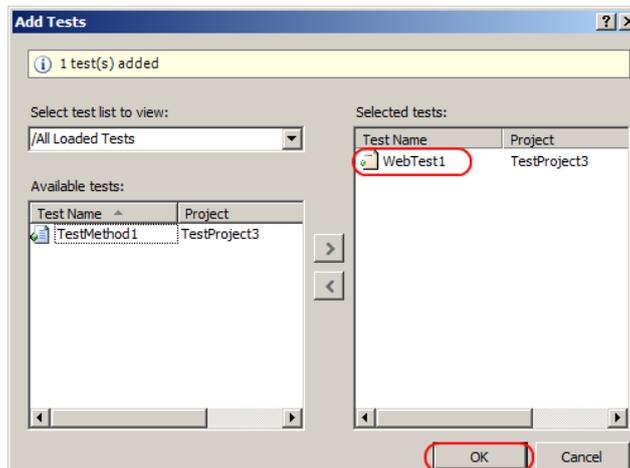


Рис. 5.12. Выбор тестов web-производительности для нагрузочного тестирования

12. Экран этапа настройки стратегии смешивания теперь содержит web-тест, который вы добавили на предыдущем шаге (рис. 5.13). Если было бы выбрано несколько тестов, то была бы возможность задать процентное соотношение их использования в течение нагрузочного тестирования.

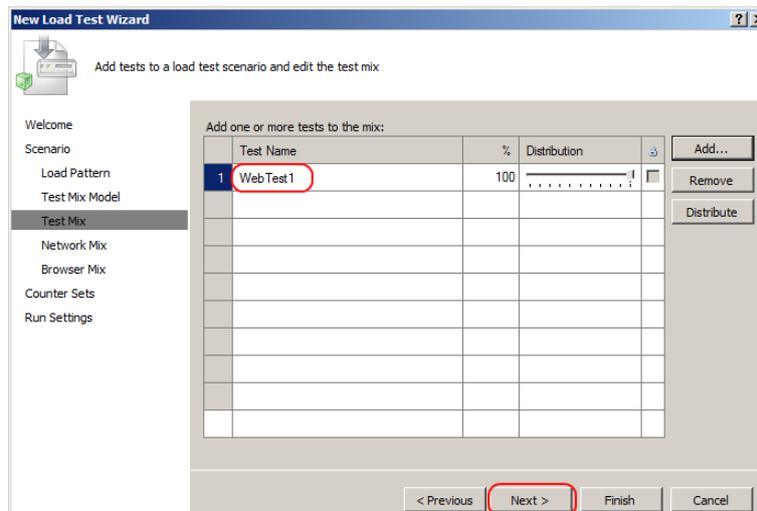


Рис. 5.13. Процентное распределение использования тестов

13. Для перехода к этапу настройки сети нажмите кнопку Next.

14. Этап Network Mix позволяет выбрать, какой тип сети мы планируем использовать в рамках нагрузочного тестирования. Мы можем добавить несколько типов сетей и задать процентное соотношение между ними. Предположим, что 75% пользователей нашего ресурса используют кабельное подключение с пропускной способностью 384 Kbps, оставшиеся 25% используют беспроводное соединение 3G. Для этого добавьте два типа сетей и установите процентное соотношение между ними, как показано на рис. 5.14.

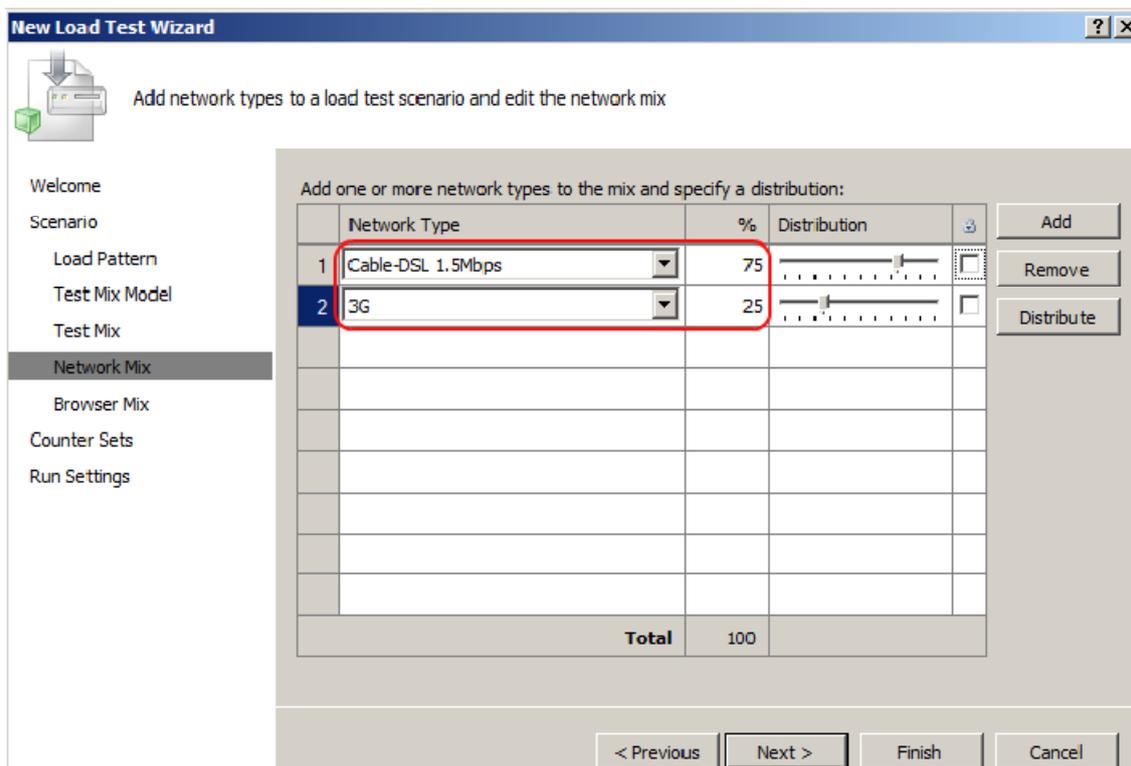


Рис. 5.14. Определение параметров сети и установка процентного распределения использования той или иной конфигурации сети

Отметим, что Visual Studio 2010 позволяет более реалистично эмулировать сетевые характеристики, чем предыдущие версии. Данное преимущество позволяет создавать наиболее реалистичные модели взаимодействия пользователей с тестируемым приложением.

15. Для перехода к этапу настройки эмулируемых браузеров нажмите кнопку Next.

16. Этап настройки браузеров позволяет выбрать один или несколько типов браузеров и указать процентное распределение между

ними (рис. 5.15). Возможность настройки типов используемых браузеров позволяет наиболее точно описывать характеристики аудитории данного ресурса.

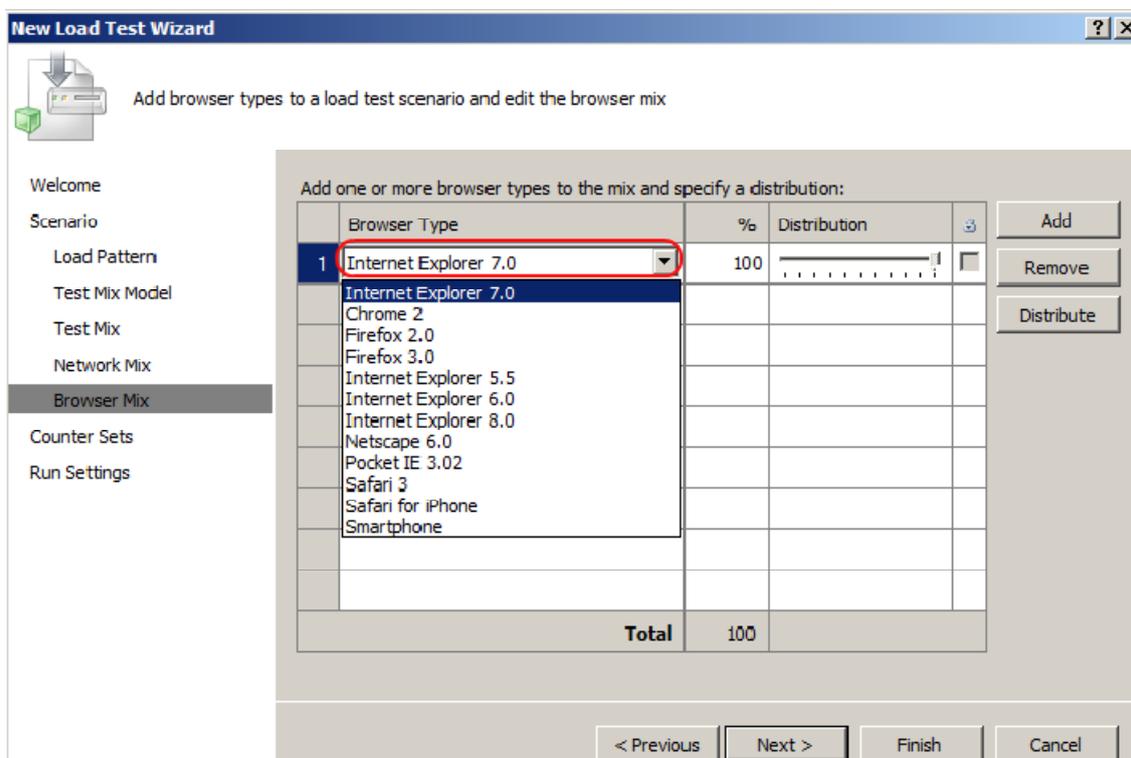


Рис. 5.15. Определение параметров браузеров и их процентное распределение

Выбор типа браузера приведет к замене заголовков в HTTP-запросах, которые будут отправляться web-серверу в рамках тестов web-производительности. Существует возможность добавления других типов браузеров, достаточно просто добавить специально созданный XML-файл с расширением *.browser* в директорию `\Common7\IDE\Templates\LoadTest\Browsers`.

17. Для перехода к этапу настройки статистики нажмите кнопку Next.

18. Для выбора и настройки счетчиков производительности, которые будут использоваться в процессе нагрузочного тестирования и по которым будет осуществляться подсчет статистики, сначала добавьте компьютер, нажав Add Computer. Затем выберите счетчик производительности ASP.NET (рис. 5.16). Обратите внимание, что компьютеры Controller Computer и Agent Computers уже созданы со своими наборами счетчиков (по умолчанию), при этом все они, включая созданный на предыдущем шаге, в данном случае представляют собой одну физическую машину.

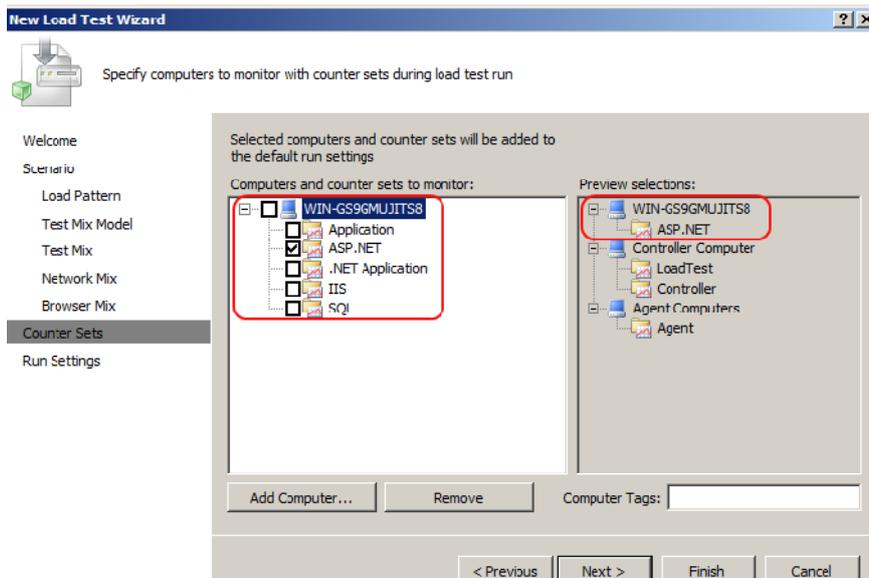


Рис. 5.16. Добавление необходимых счетчиков производительности для нагрузочного теста

19. Для перехода к настройке параметров запуска нажмите кнопку Next.

20. Параметры запуска нагрузочного теста позволяют определить длительность выполнения нагрузочного теста, при этом его продолжительность можно задать как во временных единицах, так и с использованием числа итераций (рис. 5.17).

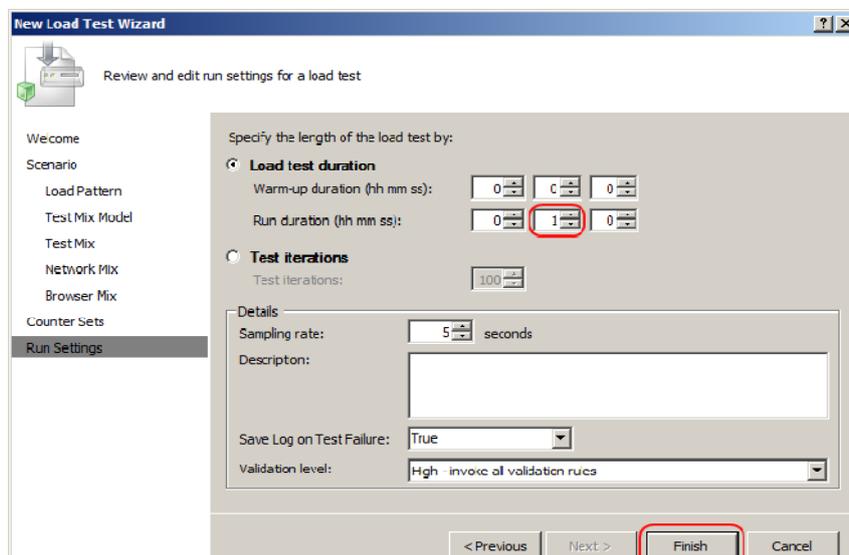


Рис. 5.17. Определение продолжительности нагрузочного теста

Для сохранения конфигурации нагрузочного теста нажмите кнопку Finish.

## 5.4. Запуск и анализ результатов выполнения нагрузочных тестов

Чтобы перейти к анализу полученных результатов тестирования, откройте структуру сгенерированного нагрузочного теста, если она еще не открыта, дважды щелкнув на тесте в Solution Explorer (рис. 5.18). Файл определения нагрузочного теста имеет расширение .loadtest.

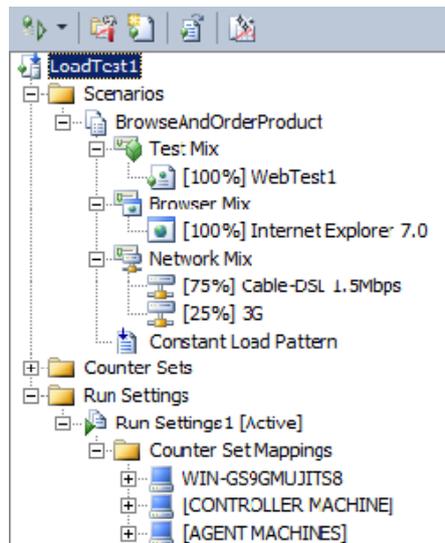


Рис. 5.18. Структура нагрузочного теста

Для запуска нагрузочного теста воспользуйтесь кнопкой Run Test (рис. 5.19).



Рис. 5.19. Расположение кнопки Run Test на панели инструментов

После чего тест запустится и будет выполняться в течение нескольких минут, при этом в окне отображения результатов вы сможете получать оперативную информацию о результатах выполнения тестов, как в виде текстовой информации, так и в виде графиков. По умолчанию окно результатов представляет собой четыре панели, на которых представлены ключевые счетчики производительности (рис. 5.20). При этом задержка обновления информации составляет установленный по умолчанию 5-секундный интервал, но данное значение может быть изменено в настройках нагрузочного теста.

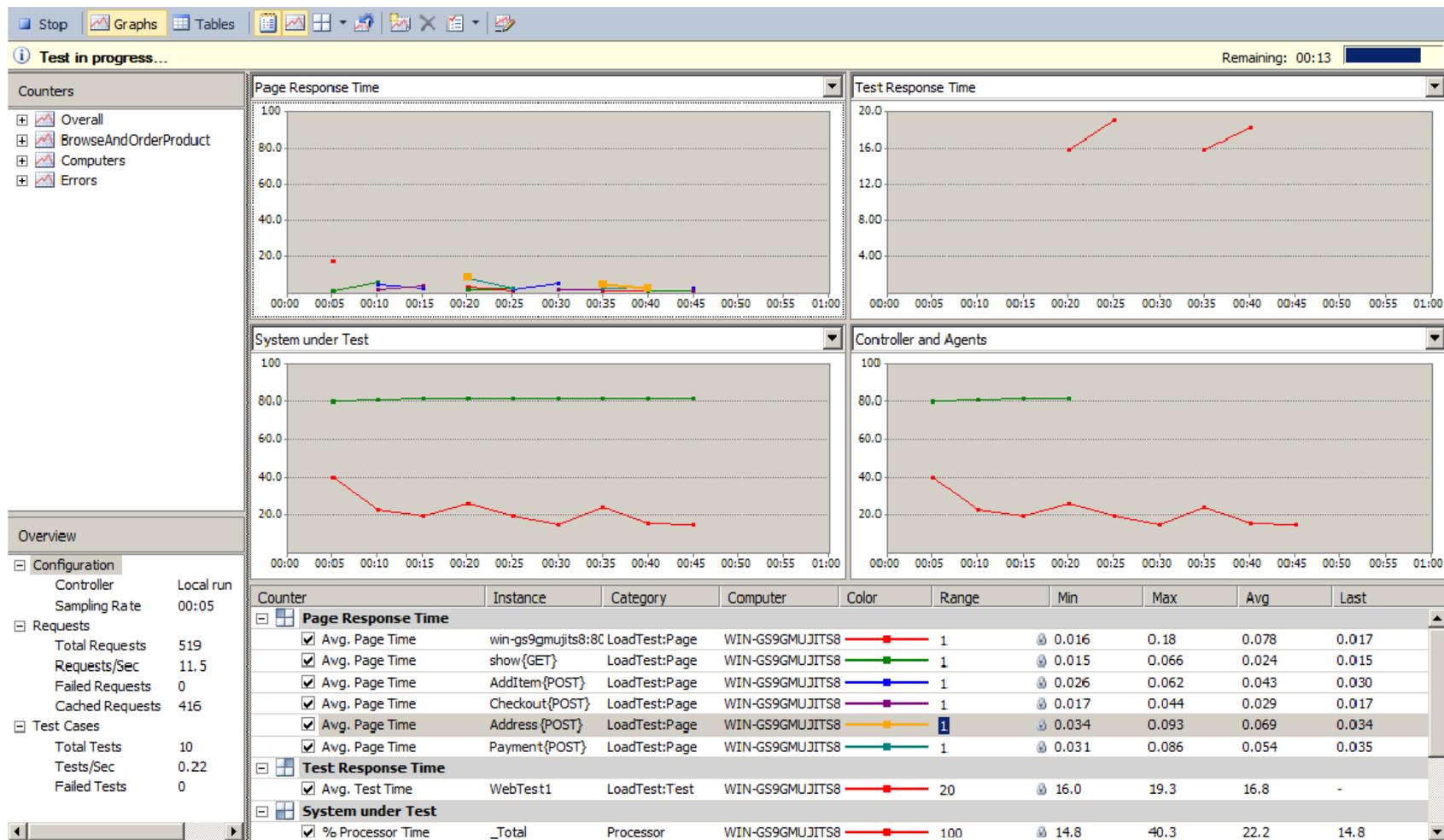


Рис. 5.20. Результаты нагрузочного тестирования, предоставляющие ключевые показатели эффективности во время выполнения нагрузочного тестирования

## **6. ТРЕБОВАНИЯ К ОФОРМЛЕНИЮ ПОЯСНИТЕЛЬНОЙ ЗАПИСКИ**

Примерный объем пояснительной записки должен составлять 30–60 страниц печатного текста, включая приложения.

К оформлению основного текста пояснительной записки предъявляются следующие требования.

### **Титульный лист**

Титульный лист следует оформлять в соответствии с приложением 1. Титульный лист не нумеруется, но в общую нумерацию страниц он входит.

### **Содержание**

Содержание размещается на новой странице и включает введение, наименования разделов, подразделов и пунктов, заключение, список использованных источников и приложения с указанием номеров страниц, на которых они размещены. Слово «Содержание» записывают в виде заголовка, размещенного по центру.

Заголовки элементов пояснительной записки в содержании соединяют отточием с номером страницы, на которой расположен заголовок. Номера страниц проставляют арабскими цифрами вплотную к правому полю без знаков препинания. Содержание должно быть сформировано автоматически.

### **Параметры страницы**

Пояснительная записка оформляется на бумаге формата А4 (297×210 мм) на одной стороне листа. Поля страницы: правое – 10 мм; верхнее – 20 мм; левое – 23 мм; нижнее – 15 мм. Цвет шрифта – черный. Текст пояснительной записки следует печатать шрифтом Times New Roman размером 14 пт, через одинарный межстрочный интервал. В случае вставки в строку формул допускается увеличение межстрочного интервала. Размер шрифта символов в формулах и уравнениях, заголовках разделов, названиях, надписях и подрисуночных подписях иллюстраций, названиях и тексте таблиц должен соответствовать размерам шрифта основного текста.

Допускается использование сокращений и аббревиатур.

Текст пояснительной записки набирается с абзацного отступа, равного 15 мм. Размеры полей и абзацных отступов должны быть одинаковыми на протяжении всего текста пояснительной записки.

Страницы пояснительной записки следует нумеровать арабскими цифрами, соблюдая сквозную нумерацию по всему тексту пояснительной записки. Номера ставятся в правом нижнем углу без точки.

В общую нумерацию страниц включают титульный лист, бланк задания, все листы работы, иллюстрации и таблицы, расположенные на отдельных листах, а также приложения. На титульном листе номер страницы не проставляют.

### **Оформление заголовков**

Каждый раздел начинается с новой страницы. Заголовки элементов записки «Содержание», «Введение», «Заключение», «Список использованных источников», «Приложение» следует записывать в начале соответствующих страниц строчными буквами, кроме первой прописной, полужирным шрифтом с выравниванием по центру.

Заголовки структурных элементов и разделов основной части следует располагать с абзацного отступа, набирать полужирным шрифтом, строчными буквами, кроме первой прописной.

В конце заголовка точку не ставят. Перенос слов в заголовках запрещен. Если заголовок занимает более одной строки, то последующие его строки должны быть записаны без абзацного отступа. Если заголовок состоит из двух предложений, то их разделяют точкой.

Нумерация заголовков выполняется арабскими цифрами. Подразделы должны иметь порядковую нумерацию внутри раздела, номер подраздела состоит из номеров раздела и подраздела, разделенных точкой. В конце точка не ставится (например, 6.1). Номер пункта состоит из номеров раздела, подраздела и пункта, разделенных точками (например, 6.1.1).

Заголовки разделов должны быть отделены от текста интервалом в 18 пт, заголовки подразделов и пунктов сверху – интервалом 18 пт, снизу – интервалом 12 пт. Соседние, последовательно записанные заголовки раздела и подраздела следует отделять друг от друга интервалом 12 пт, а подраздела и пункта – интервалом 6 пт.

Запрещено переносить заголовки подразделов и пунктов с листа на лист, а также записывать их в конце текста, если после указанных заголовков на листе размещается меньше двух строк текста.

## **Изложение текста**

В тексте не допускается употреблять обороты разговорной речи, применять для одного и того же понятия различные термины, сокращения слов, кроме установленных правилами орфографии. Перечень допускаемых сокращений русских слов оговорен в ГОСТ 2.316 и 7.12, белорусских – в СТБ 7.12.

Наименование команд, режимов следует выделять кавычками и курсивом, например «*Включить*».

В тексте перед обозначением параметра необходимо дать пояснение, например «...среднеквадратическое отклонение  $\sigma$ ...».

## **Перечисления**

Пункты перечисления записывают после двоеточия с абзацного отступа, после каждого пункта ставится точка с запятой, после последнего – точка. Перед каждым пунктом перечисления следует поставить тире. Ниже приведен пример простого перечисления.

В цикле необходимо обрабатывать исключения нескольких типов:

- отсутствие прав доступа к файлу/каталогу;
- принудительное завершение потока извне;
- слишком длинный путь (более 255 символов).

В сложных перечислениях для детализации используют строчные буквы и арабские цифры.

## **Таблицы**

Таблицу следует располагать в записке непосредственно после текста, в котором она упоминается. На все таблицы документа в тексте должны быть приведены ссылки, например «...в таблице 8.1».

Таблица при необходимости может иметь заголовок, который выполняется строчными буквами (кроме первой прописной) и помещается на одной строке через тире после слова «Таблица».

При переносе части таблицы на другую страницу слово «Таблица» и ее название помещают только над первой частью таблицы, над последующими частями таблицы пишут слева «Продолжение таблицы» с указанием ее номера, например «Продолжение таблицы 6.1».

Таблицы нумеруются арабскими цифрами сквозной нумерацией или в пределах раздела.

Ниже приведен пример оформления таблицы.

Таблица 6.1 – Соответствие классов-форм и реализованных модулей

Тип сайта	Назначение
Сайт-визитка	Содержит самые общие данные о компании или владельце: вид деятельности, история, прайс-лист, контактная информация, реквизиты, схема проезда и т. д.
Интернет-магазин	Сайт с каталогом продукции, с помощью которого клиент может заказать нужные ему товары. Используются различные системы расчетов
Промо-сайт	Сайт о конкретной торговой марке или продукте. На таких сайтах размещается исчерпывающая информация о бренде, различных рекламных акциях (конкурсы, викторины, игры и т. п.)

В таблице допускается применять размер шрифта на 1–2 пт меньше, чем в тексте.

### Формулы и уравнения

В формулах и уравнениях в качестве символов следует применять обозначения, установленные стандартами или принятые в данной отрасли. Пояснения символов и числовых коэффициентов, входящих в формулу, если они не пояснены ранее в тексте, должны быть приведены непосредственно под формулой. Первая строка пояснения должна начинаться со слова «где» без двоеточия после него.

Например: Математическое ожидание непрерывной случайной величины рассчитывается по следующей формуле:

$$M[X] = \int_{-\infty}^{+\infty} xf_X(x)dx, \quad (6.1)$$

где  $x$  – непрерывная случайная величина;

$f_X(x)$  – плотность распределения величины  $X$ .

Формулы и уравнения выравниваются по центру.

Все формулы и уравнения нумеруются арабскими цифрами сквозной нумерацией или в пределах раздела. В случае нумерации в пределах раздела номер формулы состоит из номера раздела и порядкового номера формулы, разделенных точкой. Номер указывают в круглых скобках с правой стороны листа на уровне формулы. Ссылки в тексте на порядковые номера формул дают в скобках, например «...в формуле (6.1)...».

## Иллюстрации

Иллюстрации (фотографии, рисунки, чертежи, схемы, диаграммы, графики, карты и др.) служат для наглядного представления характеристик объектов исследования, полученных теоретических и (или) экспериментальных данных и выявленных закономерностей. Не допускается одни и те же результаты представлять в виде иллюстрации и таблицы.

Иллюстрации следует располагать непосредственно после абзаца, содержащего ссылку на них. Если для размещения иллюстрации недостаточно места на соответствующей странице, необходимо разместить ее в начале следующей страницы. На все иллюстрации должны быть даны ссылки в тексте.

Иллюстрации следует нумеровать сквозной нумерацией или в пределах каждого раздела арабскими цифрами. При нумерации в пределах раздела номер включает в свой состав номер раздела и порядковый номер рисунка по разделу, разделенные точкой, например «Рисунок 6.1». Иллюстрации отделяют от текста интервалом 14 пт. Ниже дается пример оформления рисунка.

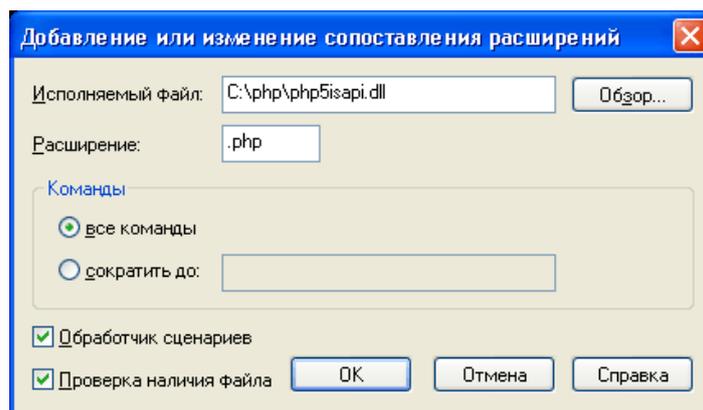


Рисунок 6.1 – Добавление серверных расширений

## Ссылки

Ссылки на источники в тексте выполняют с указанием порядкового номера в соответствии с библиографическим списком. Номер источника по списку заключается в квадратные скобки, например [4].

## Приложения

Приложения должны иметь общую с остальной частью записки сквозную нумерацию страниц. В тексте документа на все приложения

должны быть ссылки. Приложения располагают в порядке ссылок на них. Все приложения должны быть перечислены в содержании документа с указанием их номера и заголовка.

Каждое приложение следует начинать с нового листа с указанием в правом верхнем углу слова «ПРИЛОЖЕНИЕ», напечатанного прописными буквами. Приложение должно иметь содержательный заголовок, который размещается с новой строки по центру листа с прописной буквы.

Приложения обозначают заглавными буквами русского алфавита, начиная с А (за исключением Ё, З, Й, О, Ч, Ъ, Ы, Ь); например ПРИЛОЖЕНИЕ А, ПРИЛОЖЕНИЕ Б. Допускается обозначать приложения буквами латинского алфавита, за исключением букв I и O.

### **Пример оформления списка использованных источников**

1. Брауде, Э. Дж. Технология разработки программного обеспечения / Э. Дж. Брауде. – СПб.: Питер, 2004. – 656 с.

2. Сергиевский, Г. М. Функциональное и логическое программирование / Г. М. Сергиевский, Н. Г. Волченков. – М.: Академия, 2010. – 320 с.

3. Компьютерная геометрия / А. О. Иванов [и др.]. – М.: Интернет-университет информационных технологий, 2010. – 392 с.

4. Способ устройство кодирования: пат. 432848 Респ. Беларусь, МПК7 С 08/ Н 8/90 / Л. М. Урбанович, Н. Г. Волочкова; заявитель Полоц. гос. ун-т. – № а 9897644; заявл. 09.07.2006; опубл. 23.09.2010 // Афіцыйны бюл. / Нац. цэнтр інтэлектуал. уласнасці. – 2010. – № 3. – С. 156.

5. Электронная газета ArcReview. – 2007. – № 1 // Библиотека электронных ресурсов компании DATA+ [Электронный ресурс]. – Режим доступа: [http://www.dataplus.ru/Arcrev/Number\\_40/Index.html](http://www.dataplus.ru/Arcrev/Number_40/Index.html). – Дата доступа: 25.01.2010.

6. Proceeding of minisymposium on biological nomenclature in the 21<sup>st</sup> centry [Electronic resource] / Ed. J. L. Reveal. – College Park M.D., 1996. – Mode of access: <http://www.inform.ind.edu/PBIO/brum.html>. – Date of access: 14.09.2005.

## ЗАКЛЮЧЕНИЕ

В методических указаниях к выполнению курсовых работ по дисциплине «Организация web-портала и администрирование ресурсов в web» для студентов специальности 1-40 01 02-03 «Информационные системы и технологии (издательско-полиграфический комплекс)» заочной формы обучения были представлены:

1. Структура курсовой работы, ее содержание по разделам, примеры тем курсовых работ.

2. Рекомендации по созданию динамических сайтов на платформе ASP (язык C#). На примерах рассмотрены основные принципы создания, а также элементы сайтов, методы взаимодействия web-сайта с базой данных, принципы организации поиска информации в базе данных, методы вывода найденной информации с помощью элементов сайта.

3. Рекомендации по опубликованию web-сайтов при помощи web-сервера IIS. Также описано тестирование web-приложений в системе Visual Studio 2010: подробно рассмотрено на примерах использование двух тестов – Web Performance Test и Load Test.

4. Требования к оформлению пояснительной записки, примеры оформления рисунков, таблиц, списков, листингов программного кода, литературных источников, формул и т. д.

## СПИСОК РЕКОМЕНДУЕМЫХ ИСТОЧНИКОВ

1. Хенриксон, Х. IS 6. Полное руководство / Х. Хенриксон, С. Хофманн. – СПб.: Эконом, 2004. – 864 с.
2. Купцевич, Ю. Е. Альманах программиста. Платформа 2003: Microsoft Windows Server 2003, Microsoft Internet Information Services 6.0, Microsoft Office System / Ю. Е. Купцевич. – М.: Русская редакция, 2003. – 320 с.
3. Рейли, Д. Создание приложений на ASP.NET / Д. Рейли – М.: Русская редакция, 2002. – 480 с.
4. Айзекс, А. Dynamic HTML / А. Айзекс. – СПб.: BHV-Петербург, 1998. – 421 с.
5. Гончаров, А. Самоучитель HTML / А. Гончаров. – СПб.: Питер, 2000. – 356 с.
6. HTML 4. Энциклопедия пользователя / Р. Дарнелл [и др.]. – Киев: ДиаСофт, 1999. – 688 с.
7. Денисов, А. Internet Explorer 5: справочник / А. Денисов. – СПб.: Питер, 1999. – 448 с.
8. Хоумер, А. Dynamic HTML: справочник / А. Хоумер, К. Улмен. – СПб.: Питер, 1999. – 512 с.
9. Петюшкин, А. В. HTML. Экспресс-курс / А. В. Петюшкин. – СПб.: БХВ-Петербург, 2003. – 256 с.
10. Кингсли-Хью, Э. JavaScript: учебный курс / Э. Кингсли-Хью, К. Кингсли-Хью. – СПб.: Питер, 2002. – 272 с.

# ПРИЛОЖЕНИЕ 1

## ПРИМЕР ОФОРМЛЕНИЯ ТИТУЛЬНОГО ЛИСТА ПОЯСНИТЕЛЬНОЙ ЗАПИСКИ КУРСОВОЙ РАБОТЫ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ  
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет издательского дела и полиграфии  
Кафедра информационных систем и технологий  
Специальность 1-40 01 02 «Информационные системы и технологии»  
Специализация «Издательско-полиграфический комплекс»

### ПОЯСНИТЕЛЬНАЯ ЗАПИСКА КУРСОВОЙ РАБОТЫ

по дисциплине «Организация web-портала и администрирование  
ресурсов в web»

Тема \_\_\_\_\_  
\_\_\_\_\_

Исполнитель  
студент(ка) 2 курса группы 8 \_\_\_\_\_ И. А. Иванов  
подпись, дата

Руководитель  
доцент, к.т.н. \_\_\_\_\_ Д. М. Романенко  
подпись, дата

Курсовая работа защищена с оценкой \_\_\_\_\_  
Руководитель \_\_\_\_\_  
подпись

Д. М. Романенко

Минск 2012



4. Содержание пояснительной записки курсовой работы (перечень вопросов, подлежащих разработке)

---

---

---

---

---

---

---

---

5. Перечень графического, иллюстрационного материала

---

---

---

---

6. Консультанты (с указанием разделов)

---

---

---

---

7. Календарный график работы

---

---

---

8. Дата выдачи задания \_\_\_\_\_

Руководитель \_\_\_\_\_  
подпись

Д. М. Романенко

Задание принял(а) к исполнению \_\_\_\_\_  
дата и подпись студента(ки)

# **ОРГАНИЗАЦИЯ WEB-ПОРТАЛА И АДМИНИСТРИРОВАНИЕ РЕСУРСОВ В WEB**

Составители: **Романенко** Дмитрий Михайлович  
**Булова** Юлия Олеговна

Редактор *О. А. Семенец*  
Компьютерная верстка *Е. Ю. Орлова*  
Корректор *О. А. Семенец*

Издатель:  
УО «Белорусский государственный технологический университет».  
ЛИ № 02330/0549423 от 08.04.2009.  
ЛП № 02330/0150477 от 16.01.2009.  
Ул. Свердлова, 13а, 220006, г. Минск.