МЕЖГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Автоматизированные системы управления»

РАСПРЕДЕЛЕННЫЕ СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ

Методические рекомендации к лабораторным работам для студентов специальности 1-28 01 02 «Электронный маркетинг» очной и заочной форм обучения

Могилев 2022

УДК 004.43
ББК 32.973.2
Р24

Рекомендовано к изданию
учебно-методическим отделом
Белорусско-Российского университета

Одобрено кафедрой «Автоматизированные системы управления» «08» декабря 2021 г., протокол № 5

Составитель ст. преподаватель Д. А. Денисевич

Рецензент канд. техн. наук, доц. В. В. Кутузов

Методические рекомендации к лабораторным работам предназначены для студентов специальности 1-28 01 02 «Электронный маркетинг» очной и заочной форм обучения.

Учебно-методическое издание

РАСПРЕДЕЛЕННЫЕ СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ

Ответственный за выпуск А. И. Якимов

Корректор И. В. Голубцова

Компьютерная верстка Е. В. Ковалевская

Подписано в печать . Формат 60 × 84/16. Бумага офсетная. Гарнитура Таймс.
Печать трафаретная. Усл. печ. л. . Уч.-изд. л. . Тираж 26 экз. Заказ №

Издатель и полиграфическое исполнение:
Межгосударственное образовательное учреждение высшего образования
«Белорусско-Российский университет».
Свидетельство о государственной регистрации издателя,
изготовителя, распространителя печатных изданий
№ 1/156 от 07.03.2019.
Пр-т Мира, 43, 212022, г. Могилев.

Содержание

Введение

Дисциплина «Распределенные системы обработки информации» предоставляет базовые знания для моделирования, проектирования и программной разработки распределенных задач на основе современных методов, лучших решений и современных технологий.

Полученные в ходе выполнения лабораторных работ знания, умения и навыки предоставляют базу для профессиональной разработки архитектурных и проектных решений, приложений клиент-серверной архитектуры и web-распределенных вычислений в корпоративных информационных системах.

Рассматриваются расширенные возможности и перспективы применения языка Java в распределенных корпоративных системах и web-сервисах. Изучаются также основы архитектуры и программирования в .NET с применением языка C# и современных технологий на основе .NET.

При выполнении лабораторных работ используется следующее программное обеспечение: операционная система Microsoft Windows; пакет jdk; среды разработки NetBeans IDE или IntelliJ IDEA; СУБД MySQL.

1 Лабораторная работа № 1. Работа с web-ресурсами

Цель работы: ознакомиться с возможностями поиска информации в удаленных URL-ресурсах.

Порядок выполнения работы.

1 Изучить теоретические сведения, постановку задачи и порядок выполнения работы.

2 Выполнить практическое задание.

3 Сделать выводы по полученным результатам.

4 Ответить на контрольные вопросы.

5 Оформить отчет.

Теоретические сведения.

Работа с web-ресурсами [1, с. 73–84].

Постановка задачи.

Следует создать небольшое приложение, которое позволяет ввести ключевые слова и найти любой (один) html-документ, соответствующий (или, как говорят, релевантный) введенному набору ключевых слов.

Создаем исходную форму для поиска html-документов по ключевым словам согласно образцу (рисунок 1.1).

Рисунок 1.1 – Окно приложения

В текстовой области вводим ключевые слова. Нажимаем кнопку Search. Выполняется поиск по документам, записанным в пакете source_html (рисунок 1.2).

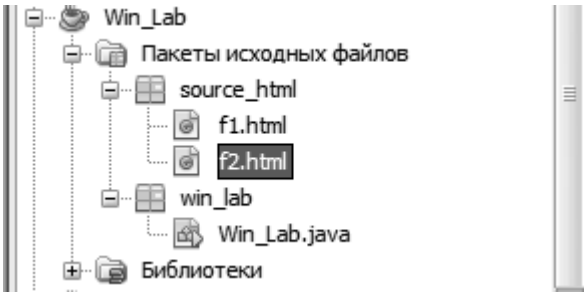

Рисунок 1.2 – Определение области поиска

В конструкторе создаются сама форма и ее визуальные компоненты (кнопки и текстовая область). Нас в первую очередь интересует кнопка поиска (sea). Ее обработчик такой:

```
      if (ae.getSource()==sea)
  {
     String [] keywords=txa.getText().split(",");
     for (int j=0;j<keywords.length;j++)
     {
        System.out.println(keywords[j]);

     }
     File f = new File("e:/work5/Win_lab/src/source_html");
     ArrayList<File> files =
 new ArrayList<File>(Arrays.asList(f.listFiles()));
     txa.setText("");
     for (File elem : files)
     {
      int zcoincidence = test_url(elem,keywords);
      txa.append("\n"+elem+"  :"+zcoincidence);
     }}
```

Сначала формируем массив ключевых слов (которые должны разделяться запятыми):

```
String [] keywords=txa.getText().split(",");
```

Затем формируем список файлов в указываемой директории:

```
  File f = new File("e:/work5/Win_lab/src/source_html");
 ArrayList<File> files =
 new ArrayList<File>(Arrays.asList(f.listFiles()));
```

После этого для каждого файла в цикле получаем с помощью метода test_url число совпадений. Если вышепредставленную «заготовку» программы выполнить с ключевыми словами students, like, travelling, то получим следующий результат в окне программы (рисунок 1.3).

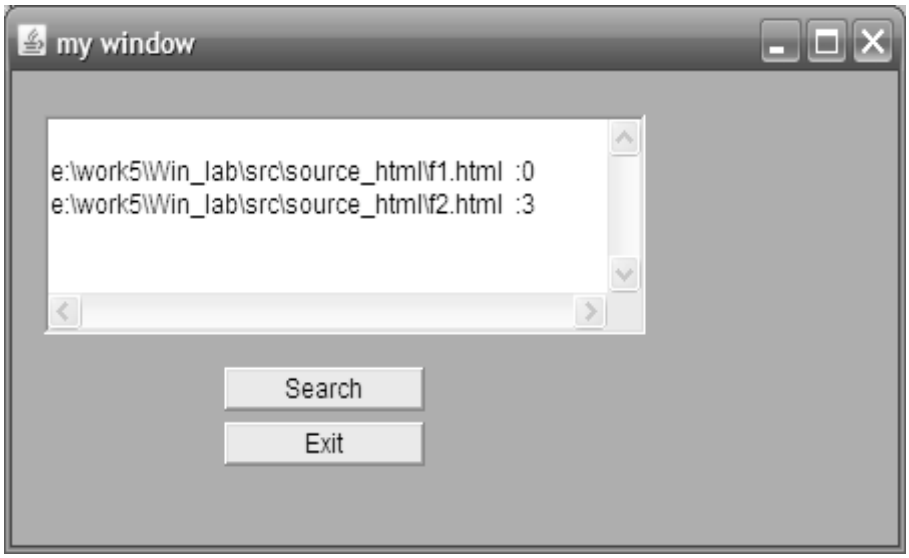

Рисунок 1.3 – Результат работы программы

Напротив имени каждого файла указано число совпадений с ключевыми словами.

Задания для самостоятельной работы.

1 Подготовить собственный набор html-документов для реализации поиска в них количества совпадений введённых ключевых слов.

2 Завершить заготовку программы так, чтобы файл с наибольшим числом совпадений открывался браузером.

Содержание отчета.

1 Титульный лист.
2 Цель работы.
3 Исходные данные.
4 Ход выполнения работы.
5 Выводы.

Контрольные вопросы

1 Что такое URL?
2 Как программно получить список всех файлов в папке?
3 Как программно запустить браузер и открыть html-файл?
4 Объясните использование потоковых классов в вашем приложении.

2 Лабораторная работа № 2. Работа на основе сокетных соединений

Цель работы: приобрести навыки проектирования и разработки приложений в архитектуре клиент-сервер.

Порядок выполнения работы.

1 Изучить теоретические сведения.
2 Выполнить практическое задание.
3 Сделать выводы по полученным результатам.
4 Ответить на контрольные вопросы.
5 Оформить отчет.

Теоретические сведения.

Простое клиент-серверное приложение на основе сокетов [1, с. 30–37].

Клиент-сервер (от англ. client-server) – вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг (сервисов), называемыми серверами, и заказчиками услуг, называемыми клиентами.

Таким образом, работа сервера состоит в прослушивании соединения, она выполняется с помощью специального объекта, который создается пользователем. Работа клиента состоит в попытке создать соединение с сервером, и это выполняется с помощью специального клиентского объекта. Как только соединение установлено, то и клиентская, и серверная сторона соединения превращается в потоковый объект ввода/вывода, таким образом, можно трактовать соединение, как будто пользователь читает и пишет файл.

IP-адреса не достаточно для уникальной идентификации сервера, т. к. на одной машине может существовать несколько серверов. Каждая IP-машина также содержит порты, и когда устанавливается клиент или сервер, тогда необходимо выбрать порт, через который и клиент, и сервер согласны соединиться.

Порт – это программная абстракция. Клиентская программа знает, как соединиться с машиной через ее IP-адрес, но она не может соединиться с определенной службой на этой машине. Таким образом, номер порта стал вторым уровнем адресации. Идея состоит в том, что при запросе определенного порта запрашивается служба, ассоциированная с этим номером порта. Обычно каждая служба ассоциируется с уникальным номером порта на определенной серверной машине. Клиент должен предварительно знать, на каком порту запущена нужная ему служба.

Сокет – это программный интерфейс, предназначенный для передачи данных между приложениями. Что же касается типов сокетов, то их два – потоковые и датаграммные.

В Java создается сокет, чтобы создать соединение с другой машиной, затем получается InputStream и OutputStream (или с соответствующими конверторами

Reader и Writer) из сокета, чтобы получить возможность трактовать соединение как объект потока ввода/вывода. Существуют два класса сокетов, основанных на потоках: ServerSocket, который использует сервер для «прослушивания» входящих соединений, и Socket, который использует клиент для инициализации соединения. Как только клиент создаст сокетное соединение, Server-Socket возвратит (посредством метода accept()) соответствующий Socket, через который может происходить коммуникация на стороне сервера. На этой стадии используются методы getInputStream( ) и getOutputStream( ) для получения соответствующих объектов InputStream'a и outputstream'a для каждого сокета. Онидолжны быть обернуты внутрь буферных и форматирующих классов точно так же, как и другие объекты потоков.

Когда создается ServerSocket, то ему дается только номер порта и не дается IP-адрес, поскольку он уже есть на той машине, на которой он установлен. Однако когда создается Socket, то необходимо передать ему и IP-адрес, и номер порта, с которым необходимо соединиться. Socket, который возвращается из метода ServerSocket.accept( ), уже содержит всю эту информацию.

Представленный далее пример покажет простейшее использование серверного и клиентского сокета. Все, что делает сервер, – это ожидает соединения, затем использует сокет, полученный при соединении, для создания InputStream'аи OutputStream'a. Они конвертируются в Reader и Writer, которые оборачиваются в BufferedReader и PrintWriter. После этого все, что будет прочитано из BufferedReader'a, будет переправлено в PrintWriter, пока не будет получена строка «END», означающая, что пришло время закрыть соединение.

Пример сервера, который просто отсылает назад все, что посылает клиент:

```
Import java.io.*; import java.net.*;
public class ExampleServer {
// Выбираем порт вне пределов 1–1024:
public static final int PORT = 8080;
public static void main(String[] args) throws IOException {
ServerSocket s = new ServerSocket(PORT); System.out.println("Started: " + s);
try { /* Блокирует до тех пор, пока не возникнет соединение:*
/ Socket socket = s.accept();
try { System.out.println("Connection accepted:" + socket);
BufferedReader in = new BufferedReader(new InputStreamReader(socket.getlnputStream()));
 /* Вывод автоматически выталкивается из буфера PrintWriter'ом*/
PrintWriter out = new PrintWriterlBufferedWriter (new OutputStreamWriter (socket.
getOutputStream())), true);
while (true) { String str = in.readLine();
if (str.equals("END")) break;
System, out.println ("Echoing: " + str); out.println(str);
}
/* Всегда закрываем два сокета...*/
}
```

```
finally {
.println( System.out.println("closing..."); socket.close();
}
finally {
s. close (); } } }
```

Таким образом, для ServerSocket'a необходим только номер порта, а не IP-адрес (т. к. он запускается на локальной машине!). Когда вызывается accept(), метод блокирует выполнение до тех пор, пока клиент не попробует подсоединиться к серверу. То есть сервер ожидает соединения, но другой процесс может выполняться. Когда соединение установлено, метод accept() возвращает объект Socket, представляющий это соединение.

Задания для самостоятельной работы.

1 Создать на основе сокетов клиент-серверное приложение.

2 Клиент посылает через сервер сообщение другому клиенту.

3 Клиент посылает через сервер сообщение другому клиенту, выбранному из списка.

4 Чат. Клиент посылает через сервер сообщение, которое получают все клиенты.

5 Клиент при обращении к серверу получает случайно выбранный сонет Шекспира из файла.

6 Сервер рассылает сообщения выбранным из списка клиентам.

7 Сервер рассылает сообщения только тем клиентам, которые в настоящий момент находятся online.

8 Чат. Сервер рассылает всем клиентам информацию о клиентах, вошедших в чат и покинувших его.

9 Клиент выбирает изображение из списка и пересылает его другому клиенту через сервер.

Содержание отчета.

1 Титульный лист.

2 Цель работы.

3 Задание.

4 Ход выполнения работы.

5 Выводы.

Контрольные вопросы

1 Что такое архитектура клиент-сервер?

2 Назначение порта, зарезервированные номера портов.

3 Что такое сокет? Назначение сокета.

4 Опишите механизм создания входного и выходного потоков.

5 Опишите механизм работы метода accept().

3 Лабораторная работа № 3. Работа с серверными классами

Цель работы: ознакомиться с техникой использования сервлетов – классов Java на стороне сервера.

Порядок выполнения работы.

1 Изучить основные теоретические сведения.
2 Выполнить практическое задание.
3 Сделать выводы по полученным результатам.
4 Ответить на контрольные вопросы.
5 Оформить отчет.

Основные теоретические сведения.

Сервлеты и jsp-страницы [1, с. 84–97].

Постановка задачи.

Нас интересует поиск различных информационных объектов на стороне сервера. Клиентская часть обычно представляется сайтом (документом Html). В качестве примера возьмем следующий сайт:

```
<!DOCTYPE html>
<html>
     <head>
       <title></title>
       <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
   </head>
<body bgcolor="#aaccff">
  <Font color="green" size="10">Форма для работы со словарем </Font>
       <br>
       <br>
 <form name="frm" method="Get" action="MyServlet">
           <Font color="blue" size="6"> Введите русское слово:</Font>
         <Input type="Text" name="txt" value=""/>
         <br>
         <br>
  <Font color="blue" size="6">Перевод: </Font>
<input type="text" name ="trans" value="     "/><br>
       <h4>Кликни здесь для получения перевода :
                 <Input type="submit" value="Перевести"/>
       </h4>
  </form>
</body>
</html>
```

В тексте сайта выполняется вызов сервлета с именем MyServlet. Это имя указывается в атрибуте action="MyServlet". Вызов сервлета выполняется по нажатию на кнопку с типом type="submit". При открытии сайт выглядит следующим образом (рисунок 3.1).

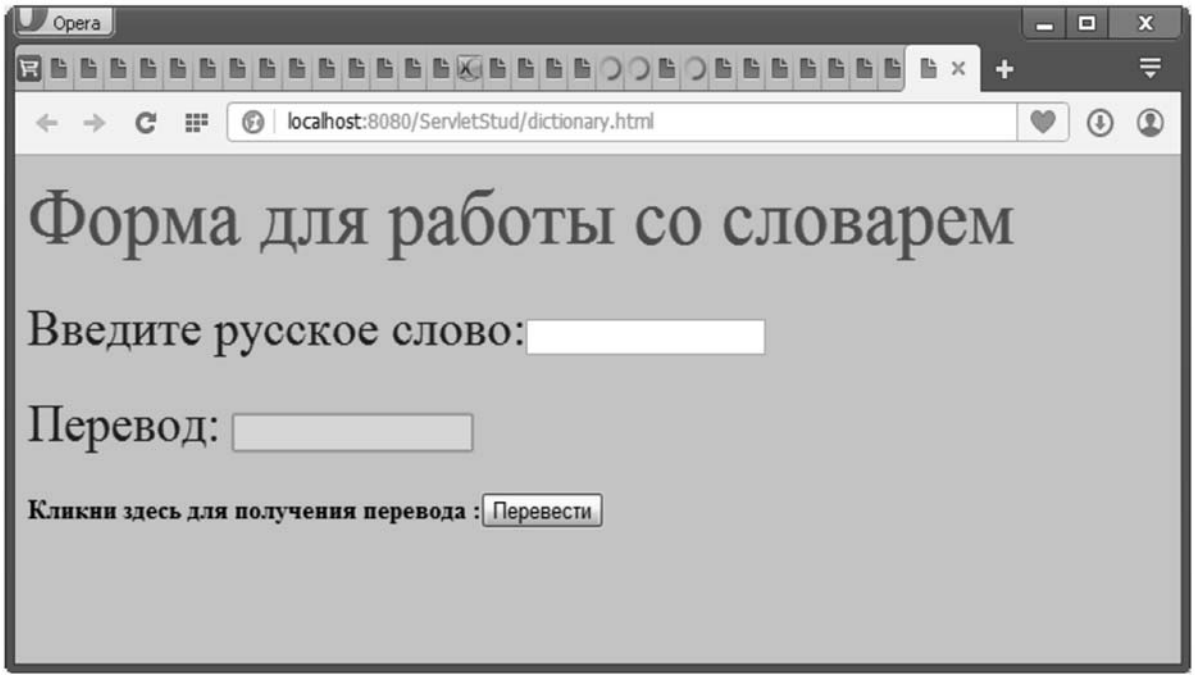

Рисунок 3.1 – Страница сайта

Нужно создать сервлет, обрабатывающий данную форму. Для построения приложения подобного типа следует использовать шаблон Java Web. Структура такого проекта состоит из следующих узлов (рисунок 3.2).

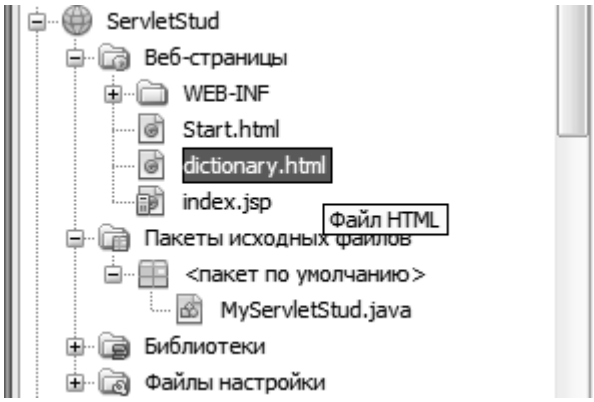

Рисунок 3.2 – Структура проекта

В узле WEB-INF помещают файлы jsp и html. В узле Пакеты исходных файлов помещают сервлеты и классы Java. В узел Библиотеки добавляют jar-файлы и библиотеки с требуемыми классами Java (если необходимо). Нам также понадобится изменить конфигурационный файл в узле Файлы и настройки.

Будем считать, что документ dictionary.html (текст которого приведен ранее) создан. Создаем теперь класс сервлета. Для этого открываем контекстное меню щелчком правой кнопки мыши на узле <пакет по умолчанию>, выбираем пункт Создать и опцию Сервлет. Указываем имя класса сервлета.

Класс сервлета содержит два важнейших метода – doGet и doPost. Оба предназначены для обработки данных от клиентской формы. Эти методы вызываются в зависимости от атрибута method тега form документа html:

```
<form name="frm" method="Get" action="MyServlet">
```

В методе Get данные клиентской формы передаются одним сообщением вместе со служебной информацией (заголовком). В методе Post данные разбиваются на несколько пакетов. Оба этих метода «перенаправлены» на один метод processRequest. Именно в последнем методе мы и «сосредоточим» всю необходимую обработку:

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException
{
      response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();
      String rus_word=""+request.getParameter("txt");
```

Здесь приведена стартовая часть метода processRequest. У метода есть два встроенных объекта – request, response, с помощью которых можно получить значения элементов формы клиентского сайта и направить их обратно в клиентский сайт. Видим, как читается значение параметра:

```
String rus_word=""+request.getParameter("txt");
```

Отправка значений обратно клиенту поясняется следующим фрагментом:

```
PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet MyServletStud</title>");
        out.println("</head>");
        out.println("<body bgcolor='#aaccff'");
        out.println("<form>");
        out.println("<h2> Привет клиенту!!!</h2><br><br>");
        out.println("</form>");
        out.println("</body>");
        out.println("</html>");
```

Создаем потоковый объект PrintWriter out и используем его метод println. Теперь скорректируем конфигурационный файл web.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
    <servlet>
```

```
    <servlet-name>MyServlet</servlet-name>
    <servlet-class>MyServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>MyServlet</servlet-name>
    <url-pattern>/MyServlet</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>
      Dictionary.html
     </welcome-file>
   </welcome-file-list>
   <session-config>
    <session-timeout>
      30
    </session-timeout>
  </session-config>
</web-app>
```

Чтобы этот файл появился в проекте, нужно открыть контекстное меню на имени проекта и выполнить последовательно пункты Очистить и Построить и Развернуть.

Задания для самостоятельной работы.

1 Завершить приложение работоспособной программой, в которой используется база данных, содержащая переводы слов.

2 Добавить в программу обратный перевод.

Содержание отчета.

1 Титульный лист.

2 Цель работы.

3 Задание.

4 Ход выполнения работы.

5 Выводы.

Контрольные вопросы

1 Что такое сервлет, какие основные объекты и методы он предоставляет?

2 Как сервлет определяется в клиентском сайте?

3 Для чего служит конфигурационный файл web.xml? Поясните его структуру.

4 Как вернуть результаты, полученные в сервлете, обратно на сторону клиента?

4 Лабораторная работа № 4. Web-сервисы: взаимодействия, реализованные через удаленный вызов процедур RPC

Цель работы: научиться работать с технологией web-сервисов, используя RPC-ориентированный подход.

Порядок выполнения работы.

1 Изучить основные теоретические сведения.
2 Выполнить практическое задание.
3 Сделать выводы по полученным результатам.
4 Ответить на контрольные вопросы.
5 Оформить отчет.

Основные теоретические сведения.

Разработка web-сервиса, реализующего RPC-ориентированное взаимодействие [9, с. 5–10].

Web-сервисы – это часть бизнес-логики приложения, размещенной в интернете, которая обеспечивает доступ по стандартным интернет-протоколам, таким как HTTP или SMTP.

***Главные технологии web-сервисов*:**

– SOAP (Simple Object Access Protocol) – стандарт «упаковки» XML-документов для транспортировки по протоколам SMTP, HTTP или FTP;

– WSDL (Web Service Description Language) – это XML-технология, описывающая интерфейсы web-сервисов. Помогает клиентам автоматически понять, как работать с сервисом;

– UDDI (Universal Description, Discovery and Integration) – обеспечивает всемирную регистрацию web-сервисов. Используется для обнаружения сервиса путем поиска по имени, категории и др.

***Взаимодействие между технологиями*:**

– клиент запрашивает у репозитория UDDI сервис по его имени или идентификатору;

– клиент получает информацию о размещении WSDL-документа от UDDI-репозитория. Он содержит информацию о том, как связаться с сервисом, и формат запроса в XML;

– согласно найденной в WSDL информации клиент создает SOAP-сообщение и посылает на хост сервиса.

Провайдеры публикуют информацию (метаданные) о сервисе в репозитории. Согласно документу Web Services Conceptual Architecture компании IBM для публикации используются несколько различных механизмов.

***Механизм публикации прямой*.**

Инициатор запроса ищет описание сервиса прямо у провайдера сервиса, используя e-mail, FTP, CD. Провайдер доставляет описание сервиса и одновременно делает сервис доступным для инициатора запроса. Как такового репозитория (брокера) здесь нет.

Apache Axis **(*Apache eXtensible Interaction System*)** – система для конструирования SOAP-процессоров, таких как клиенты, серверы, шлюзы и др. SOAP – это механизм для коммуникации приложений посредством интернета.

Задания для самостоятельной работы.

Написать web-сервис, реализующий функционал приложения в соответствии с вариантом предметной области, посредством которого клиент сможет выполнять операции просмотра, добавления, удаления и поиска информации. Данные должны храниться в массиве на сервере.

1 Разработать приложение, реализующее учет продажи телевизоров.

2 Разработать приложение, реализующее учет закупки компьютеров университетом.

3 Разработать приложение, реализующее учет посещаемости студентами лекций.

4 Разработать приложение, реализующее учет покупок мороженого в магазине.

5 Разработать приложение, реализующее учет количества сотрудников в подразделениях предприятия.

6 Разработать приложение, реализующее учет зверей в зоопарке.

7 Разработать приложение, реализующее учет машин в автосалоне.

8 Разработать приложение, реализующее учет продажи конфет кондитерской фабрикой.

9 Разработать приложение, реализующее учет продажи канцелярских товаров в магазине.

10 Разработать приложение, реализующее учет отгрузки овощей на торговом складе.

11 Разработать приложение, реализующее учет поставок топлива на автозаправки.

12 Разработать приложение, реализующее учет поставок учебников в школы.

13 Разработать приложение, реализующее учет нагрузки по предметам.

14 Разработать приложение, реализующее учет успеваемости студентов.

Контрольные вопросы

1 Как происходит взаимодействие между технологиями web-сервисов?

2 Для чего предназначена SOAP-технология?

3 Какие механизмы используются для публикации информации о сервисе в регистре?

4 Когда функция main выбрасывает исключения?

5 Для чего предназначена функция invoke()?

5 Лабораторная работа № 5. Web-сервисы: реализация документоориентированного взаимодействия

Цель работы: научиться работать с технологией web-сервисов, используя документоориентированный подход.

Порядок выполнения работы.

1 Изучить основные теоретические сведения.
2 Выполнить практическое задание.
3 Сделать выводы по полученным результатам.
4 Ответить на контрольные вопросы.
5 Оформить отчет.

Основные теоретические сведения.

Разработка web-сервиса, реализующего документоориентированное взаимодействие [9, с. 12–28].

В лабораторной работе № 4 мы создали web-сервис, используя модель RPC. В данной лабораторной работе документы XML будут являться основным инструментом взаимодействия между вызывающим объектом и web-сервисом. Этот тип SOAP-разработки называется документоориентированным (*document-style*) подходом.

В случае документоориентированного программирования клиент передает web-сервису документ XML, который обрабатывается на сервере. Здесь также (как и в лабораторной работе № 4) на макроуровне клиент передает запрос SOAP и получает ответ SOAP (рисунок 5.1).

Рисунок 5.1 – Модель взаимодействия клиента и сервиса при документоориентированном подходе

В документоориентированном SOAP-программировании клиенту не нужно сериализовать вызов Java и его аргументы в документ XML. И наоборот, серверу

не нужно десериализовать документ XML в вызов Java и типы данных. В RPC SOAP-программировании клиент и сервер должны придерживаться строго определенной программной модели: клиент вызывает метод с возможными аргументами, а сервер в ответ возвращает одно значение. В документоориентированном программировании в обмене участвует документ XML и значение каждого элемента интерпретируется участвующими во взаимодействии сторонами. Вдобавок к этому клиент и web-сервис могут объединить в запрос и ответ несколько документов. Так как процесс сериализации/десериализации отсутствует, приложения, разработанные посредством использования документоориентированного программирования, должны быть более быстрыми, чем их RPC-аналоги. Тем не менее это не обязательно будет так. Это можно аргументировать тем, что при документоориентированном программировании сериализация зависит от разработчика, т. к. в некоторый момент нам может понадобиться преобразовать наши данные Java в XML и наоборот. А также задействованные документы XML потенциально могут быть намного больше, чем простые типы, используемые в RPC-модели.

Документоориентированное SOAP-программирование подходит, когда мы хотим осуществлять обмен данными между двумя и более сторонами. Это особенно справедливо в случае, когда у нас уже есть документ XML, представляющий данные, т. к. мы можем обмениваться самим документом XML в исходном виде без необходимости преобразовывать его структуры данных Java. Отсутствие шагов сериализации/десериализации упрощает разработку и ускоряет обработку данных.

RPC-метод разработки SOAP срабатывает очень хорошо, если проблему можно легко смоделировать с помощью вызовов методов или если у нас уже есть функциональный API. Для приложений, которые манипулируют документами XML с помощью XSLT, документ XML более предпочтителен, чем структуры данных Java, т. к. процессор XSLT может автоматически преобразовывать этот документ.

Задания для самостоятельной работы.

1 Информация о телевизорах хранится на сервере в виде XML-файла. Написать web-сервис, возвращающий содержимое XML-файла клиентскому приложению.

2 Информация о музыкальных форматах хранится на сервере в виде XML-файла. Написать web-сервис, возвращающий содержимое XML-файла клиентскому приложению.

3 Информация о компьютерах хранится на сервере в виде XML-файла. Написать web-сервис, возвращающий содержимое XML-файла клиентскому приложению.

4 Информация о мобильных телефонах хранится на сервере в виде XML-файла. Написать web-сервис, возвращающий содержимое XML-файла клиентскому приложению.

5 Информация о сотрудниках хранится на сервере в виде XML-файла. Написать web-сервис, возвращающий содержимое XML-файла клиентскому приложению.

6 Информация о книгах хранится на сервере в виде XML-файла. Написать web-сервис, возвращающий содержимое XML-файла клиентскому приложению.

7 Информация о гостиницах хранится на сервере в виде XML-файла. Написать web-сервис, возвращающий содержимое XML-файла клиентскому приложению.

8 Информация о курсах валют хранится на сервере в виде XML-файла. Написать web-сервис, возвращающий содержимое XML-файла клиентскому приложению.

9 Информация о туристических направлениях хранится на сервере в виде XML-файла. Написать web-сервис, возвращающий содержимое XML-файла клиентскому приложению.

10 Информация о предприятии хранится на сервере в виде XML-файла. Написать web-сервис, возвращающий содержимое XML-файла клиентскому приложению.

11 Информация об оценках студентов хранится на сервере в виде XML-файла. Написать web-сервис, возвращающий содержимое XML-файла клиентскому приложению.

12 Информация о вокзале хранится на сервере в виде XML-файла. Написать web-сервис, возвращающий содержимое XML-файла клиентскому приложению.

13 Информация о больнице хранится на сервере в виде XML-файла. Написать web-сервис, возвращающий содержимое XML-файла клиентскому приложению.

14 Информация о зоопарке хранится на сервере в виде XML-файла. Написать web-сервис, возвращающий содержимое XML-файла клиентскому приложению.

15 Информация об автомобилях хранится на сервере в виде XML-файла. Написать web-сервис, возвращающий содержимое XML-файла клиентскому приложению.

Контрольные вопросы

1 Что является основным инструментом взаимодействия при использовании документоориентированного подхода?

2 Как работает документоориентированное взаимодействие?

3 В чем его различие с RPC-взаимодействием?

4 Почему при документоориентированном программировании не требуется сериализация/десериализация объектов?

5 Когда лучше применять RPC-модель, а когда документную модель?

6 Объясните структуру SOAP-сообщения.

6 Лабораторная работа № 6. Реализация взаимодействия с EJB-компонентом

Цель работы: ознакомиться с компонентной технологией EJB.

Порядок выполнения работы.

1 Изучить теоретические сведения, постановку задачи и порядок выполнения работы.

2 Выполнить практическое задание.

3 Сделать выводы по полученным результатам.

4 Ответить на контрольные вопросы.

5 Оформить отчет.

Теоретические сведения.

Создание EJB-компонентов [1, с. 196–205].

Задание для самостоятельной работы.

Приведенное приложение в [1, с. 196–205] следует усложнить, включив в интерфейс методы

```
public String addStudent(),
public String deleteStud(Student em),
```

а также методы для вывода списка всех студентов, подсчета числа студентов в указанной группе, вывода списка студентов, обучающихся на указанном факультете (department).

Содержание отчета.

1 Титульный лист.

2 Цель работы.

3 Задание.

4 Ход выполнения задания.

5 Выводы.

Контрольные вопросы

1 Как вы понимаете назначение и принципы работы сеансового компонента?

2 Как реализуется удаленный сеансовый компонент?

3 Как добавить в компонент новый бизнес-метод?

4 Как осуществляется доступ к методам компонента из клиента?

7 Лабораторная работа № 7. Разработка ASP-приложений на базе платформы .Net Framework

Цель работы: изучить различные элементы управления в приложениях ASP.NET. Разработать простое приложение со стандартными элементами управления на базе технологии ASP.NET.

Порядок выполнения работы.

1 Изучить краткие теоретические сведения.
2 Выполнить задание.
3 Сделать выводы по полученным результатам.
4 Ответить на контрольные вопросы.
5 Оформить отчет.

Краткие теоретические сведения.

Одна из самых важных задач web-разработчика – получение и обработка данных, введенных пользователем. Информация посылается серверу через форму. Форма содержит элементы управления, которые позволяют различными способами вводить информацию.

Формы ASP.NET отличаются от обычных форм наличием свойства *runat = "server"*. Они обрабатываются ASP.NET на сервере. На странице находятся элементы управления. ASP.NET позволяет запоминать состояние элементов управления, т. е. текст, который был введен, или выбранный переключатель, передавать его на сервер и обратно на страницу после ее обновления.

Элемент управления, для которого использован атрибут runat = "server", становится доступным из программного кода в файле codebehind, а на события этого элемента управления реагирует сервер. Если данный атрибут убрать, элемент управления станет обычным элементом управления HTML.

Второй обязательный атрибут – это его идентификатор, или ID. Он нужен, чтобы обращаться к элементу в программе.

Существуют две группы элементов управления.

1 ***Элементы управления HTML.***

Элементы управления HTML являются наследниками класса System.Web.UI.HtmlControls.HtmlControl. Они непосредственно отображаются в виде элементов разметки HTML. Их отображение не зависит от типа браузера. Свойства таких элементов полностью соответствуют атрибутам тегов HTML.

2 ***Стандартные или серверные элементы управления.***

Серверные элементы мощнее, они привязаны не к разметке, а к функциональности, которую нужно обеспечить. Многие элементы не имеют аналогов в HTML, например, календарь. Их отрисовка полностью контролируется ASP.NET. Перехватывая события PreRender, Init, Load, можно вмешаться в этот процесс. Объявления серверного элемента управления начинаются с блока <asp:тип> и заканчиваются </asp:тип>.

В таблице 7.1 приведены элементы управления, которые имеют пару среди тегов HTML. Некоторые элементы генерируют не только HTML-код, но и JavaScript.

Таблица 7.1 – Соответствие некоторых серверных элементов управления тегам HTML

Элемент управления ASP.NET	Соответствующий тег HTML	Назначение
<asp:Label>	<Span>	Отобразить текст
<asp:ListBox>	<Select>	Список выбора
<asp:DropDownList>	<Select>	Выпадающий список
<asp:TextBox>	<Input Type="Text"> <Input Type="Password"> <Textarea>	Строка редактирования Поле редактирования
<asp:HiddenField>	<Input Type="Hidden">	Невидимое поле
<asp:RadioButton>,		
<asp:RadioButtonList>	<Input Type="Radio">	Переключатель, список переключателей
<asp:CheckBox>		
<asp:CheckBoxList>	<Input Type="CheckBox">	Флажок, список флажков
<asp:Button>	<Input Type="button"> <Input Type="submit">	Командная кнопка
<asp:Image>	<img>	Изображение
<asp:ImageButton>	<input type="image">	Кнопка-изображение
<asp:Table>	<Table>	Таблица
<asp:Panel>	<Div>	Контейнер
<asp:HyperLink>	<A href>	Гиперссылка

Задание.

Создать web-форму с элементами управления. При нажатии на кнопку «Посчитать» должны рассчитываться проценты, которые должны быть уплачены по кредиту. В данном случае будем считать, что проценты рассчитываются по формуле (7.1):

$$\textit{Сумма проц.} = \textit{Сумма кредита} \cdot \left(\frac{\textit{Проц. Ставка}}{100}\right) \cdot \left(\frac{\textit{Кол. дн.}}{360}\right). \qquad (7.1)$$

Пользователь должен выбирать только из стандартных ставок по кредиту в 12 %, 15 % и 20 % годовых (в ниспадающем списке).

Контрольные вопросы

1 Чем отличаются серверные элементы управления от стандартных?
2 Чем отличаются приложения Web Forms и Windows Forms?
3 Какие события жизненного цикла web-приложения вы знаете?
4 Какие существуют три типа событий серверных элементов управления?
5 Какие этапы можно выделить в жизненном цикле страницы ASP.NET?

8 Лабораторная работа № 8. Разработка многопоточных приложений

Цель работы: изучить средства организации многопоточной обработки информации в платформе Microsoft.NET.

Порядок выполнения работы.

1 Изучить теоретические сведения.
2 Выполнить задание.
3 Сделать выводы по полученным результатам.
4 Ответить на контрольные вопросы.
5 Оформить отчет.

Теоретические сведения.

Понятие потоков, возможности многопоточного программирования.

Развитие аппаратных средств и режимов их работы привело к появлению новых возможностей по организации эффективной работы вычислительных ресурсов. Новые средства мультипроцессирования позволяют решить сложные задачи организации вычислительного процесса.

1 Приложение должно выполнять длительные операции обработки данных или ожидать освобождения некоторого ресурса.

2 Приложение должно выполнять операции в фоновом режиме, одновременно обслуживая пользовательский интерфейс.

3 Нужен механизм, позволяющий нескольким операциям выполняться одновременно с целью повышения степени использования компьютера и ускорения вычислений.

Поток является единицей диспетчеризации при обработке данных, т. е. его исполнение подвергается планированию со стороны операционной системы. Многозадачность – это одновременное исполнение нескольких потоков. Существуют два вида многозадачности – кооперативная (cooperative) и вытесняющая (preemptive). Платформа .NET работает только в системах с вытесняющей многозадачностью.

При вытесняющей многозадачности процессор отвечает за выделение каждому потоку определенного количества времени, в течение которого поток может выполняться, кванта времени (timeslice). Далее процессор переключается между разными потоками, выдавая каждому потоку его квант времени, а программист не заботится о том, как и когда возвращать управление, в результате чего могут работать и другие потоки. В случае вытесняющей многозадачности на однопроцессорной машине в любой момент времени реально будет исполняться только один поток. Поскольку интервалы между переключениями процессора от процесса к процессу очень малы (в пределах микросекунд), возникает иллюзия многозадачности. Чтобы несколько потоков на самом деле работали одновременно, потребуется многопроцессорная машина.

Преимущества использования потоков состоят в повышении производительности, ускорении вычислительного процесса при решении задачи, обеспечении интерактивного режима работы, а также режимов реального времени. Использование потоков дает возможность эквивалентного разделения вычислительной мощности процессора между приложениями, возможность сокращения простоев и стоимости обработки информации.

Создание потоков, свойства потоков.

Потоки выполнения в платформе Microsoft.NET создаются на основе системного класса System.Thread, экземпляры которого соответствуют реальным потокам исполнения в рамках операционной системы. Рассмотрим пример создания потоков исполнения:

```
using System;
using System.Threading;

class SimpleThreadApp {
public static void WorkerThreadMethodQ
{
Console.WriteLine("Запущен рабочий поток");
}

public static void Main() {
ThreadStart worker = new ThreadStart(WorkerThreadMethod); Console.WriteLine("Main – Создаем рабочий поток"); Thread t = new Thread(worker); t.StartQ; Console.WriteLine("Main – Рабочий поток запущен");
}}
```

Скомпилировав и запустив это приложение, можно заметить, что сообщение метода Main выводится перед сообщением рабочего потока. Это доказывает, что рабочий поток действительно работает асинхронно. Пространство имен System. Threading содержит классы, необходимые для организации потоков в среде .NET.

```
ThreadStart WorkerThreadMethod = new ThreadStart(WorkerThreadMethod);
```

ThreadStart – это делегат. Он должен быть задействован при создании нового потока. Он используется, чтобы задать метод, который должен вызываться как метод потока. Здесь создаётся экземпляр объекта Thread, при этом конструктор принимает в качестве аргумента только делегат ThreadStart:

```
Thread t = new Thread(worker);
```

После этого вызывается метод Start объекта Thread, в результате чего вызывается метод WorkerThreadMethod.

Поток начинает существование в состоянии Start. В процессе работы он может самостоятельно или под влиянием других потоков перейти в состояние приостановки (Suspend, Sleep, WaitX), завершиться (Abort) или слиться (Join) с

другим потоком. Из этих состояний он может быть выведен по требованию программы (Resume), по истечении времени (Expire Time), по прерыванию (Interrupt) или по сигналу (Notified). После завершения работы поток прекращается (All Done).

Каждый поток обладает набором свойств, позволяющим определить его состояние или отличить его от других потоков:

IsAlive – указывает, что поток существует;

IsBackground – указывает, что это фоновый поток;

IsThreadPoolThread – указывает, что поток относится к пулу потоков;

Name – указывает имя потока;

Priority – указывает и устанавливает приоритет потока;

ThreadState – указывает состояние потока.

Планирование потоков, методы синхронизации.

При переключении процессора по окончании выделенного потоку кванта времени процесс выбора следующего потока, предназначенного для исполнения, далеко не произволен. У каждого потока есть приоритет, указывающий процессору, как должно планироваться выполнение этого потока по отношению к другим потокам системы. Для потоков, создаваемых в период выполнения, уровень приоритета по умолчанию равен Normal. Для просмотра и установления этого значения служит свойство Thread.Priority. Установщик свойства Thread.Priority принимает аргумент типа Thread.ThreadPriority, который представляет собой перечислимый тип, имеющий значения Highest, AboveNormal, Normal, BelowNormal и Lowest.

Чтобы проиллюстрировать, как приоритет влияет на код, рассмотрим пример, в котором один поток считает от 1 до 10, а другой – от 11 до 20.

```
using System;
using System.Threading;
class ThreadSchedulelApp
{
public static void WorkerThreadMethod1(object param)
{
int start = (int)param; Console.WriteLine("Worker thread started");
Console.WriteLine ("Worker thread - counting slowly from 1 to 10"); for (int i = start; i < start
+ 10; i++) {
// Код, который имитирует работу Console.Write("."); Thread.Sleep(500);
Console.Write("{0}", i);
}
Console.WriteLine("Worker thread finished");
}

public static void Main()
{
Thread worker1 = new Thread(WorkerThreadMethod1); Thread worker2 = new
Thread(WorkerThreadMethod1); Console.WriteLine("Main - Creating worker threads");
worker1.Start(1); worker2.Start(11);
}}
```

Обратите внимание на то, что метод, реализующий работу потока, принимает параметр param, позволяющий передать в поток параметр, на основе которого поток строит свое выполнение. В данном случае параметр задает начальное число для счетчика. Примерный результат работы программы приведен далее:

```
Main - Creating worker threads Worker thread started
Worker thread started
.1.11.12.2.13.3.14.4.15.5.16.6.17.7.8.18.19.9.20.10
Worker thread finished Worker thread finished
```

В зависимости от поведения диспетчера порядок цифр может изменяться. Теперь изменим свойство Priority каждого потока, как это сделано в следующем коде. Результаты будут во многом отличаться от предыдущих:

```
Main - Creating worker threads Worker thread started
Worker thread started
.1.11.2.12.3.13.4.14.5.15.6.16.7.17.8.18.9.19.10.20
Worker thread finished Worker thread finished
```

В данном случае первый поток имел наивысший приоритет, поэтому он всегда срабатывал первым.

Рассмотрим механизмы явной синхронизации, позволяющие управлять порядком выполнения потоков. Пусть у нас есть программа имитации доступа к банковскому счету, которая позволяет снимать со счета деньги:

```
using System;
using System.Threading;
class Account
{
public int mBalance; public void Withdraw100()
{
int oldBalance = mBalance; Console.WriteLine("Withdrawing 100..."); Thread.Sleep(100);
int newBalance = oldBalance - 100;   mBalance = newBalance;
}

static void Main(string[] args)
{
Account acc = new Account(); acc.mBalance = 500;
Console.WriteLine("Account balance = {0}", acc.mBalance);
Thread user1 = new Thread(
new ThreadStart(acc.Withdraw100)); Thread user2 = new Thread(
new ThreadStart(acc.Withdraw100)); user1.Start();
user2.Start(); user1.Join(); user2.Join();
Console.WriteLine("Account balance = {0}", acc.mBalance);
}}
```

Результат работы этой программы без синхронизации показан далее: со счета сняли 200 ед. за 2 транзакции, но баланс скорректировался только на 100.

```
Account balance = 500 Withdrawing 100...
Withdrawing 100... Account balance = 400
```

Как избежать подобных непредсказуемых состояний? На самом деле, как это обычно бывает в программировании, существует несколько способов решения этой хорошо известной проблемы. Стандартное средство – синхронизация. Синхронизация позволяет задавать критические секции (critical sections) кода, в которые в каждый отдельный момент может входить только один поток, гарантируя, что любые временные недействительные состояния вашего объекта будут невидимы его клиентам. Рассмотрим средство определения критических секций – класс .NET Monitor. Он позволяет оградить критическую секцию кода, гарантируя ее атомарное исполнение. Пример его использования в функции Withdraw100 приведен далее:

```
public void Withdraw100()
{
Monitor.Enter(this); try
{
int oldBalance = mBalance; Console.WriteLine("Withdrawing 100..."); Thread.Sleep(100);
int newBalance = oldBalance - 100; mBalance = newBalance;
}
finally
{
Monitor.Exit(this); }}
```

Теперь действия в границах монитора будут выполняться последовательно для всех потоков, и результат работы программы будет таким:

```
Account balance = 500 Withdrawing 100...
Withdrawing 100... Account balance = 300
```

Задания для самостоятельной работы.

Создать приложение с элементарным графическим интерфейсом. Действия по варианту выполнять во вторичном потоке.

1 Копирование файла (* с прогрессбаром).

2 Поиск вхождения строки в большой файл (* с прогрессбаром).

3 Подсчет размера каталога (со всеми подкаталогами, файлами).

4 Поиск файла по шаблону (? – символ, * – множество символов) в папке.

Контрольные вопросы

1 Что такое потоки, для чего их применяют?

2 Какие состояния имеет поток при своем существовании? Чем они отличаются?

3 Для чего нужен приоритет потока, как регулируется приоритет?

4 Что такое синхронизация потоков, какими средствами она выполняется?

9 Лабораторная работа № 9. Разработка многопоточных приложений оптимизации ресурсов видовой логистики

Цель работы: получить навыки реализации многопоточной обработки информации в платформе Microsoft.NET.

Порядок выполнения работы.

1 Изучить основные теоретические сведения.
2 Выполнить практическое задание.
3 Сделать выводы по полученным результатам.
4 Ответить на контрольные вопросы.
5 Оформить отчет.

Основные теоретические сведения.

Платформа разработки .NET Framework позволяет создавать web-сервисы, приложения Web Forms, Win32 GUI, Win32 GUI (с консольным интерфейсом пользователя), службы (управляемые Service Control Manager), утилиты и автономные компоненты. .NET Framework – это управляемая среда для разработки и исполнения приложений, обеспечивающая контроль типов. Эта среда управляет выполнением программы, она: выделяет память под данные и команды; назначает разрешения программе или отказывает в их предоставлении; начинает исполнение приложения и управляет его ходом; отвечает за освобождение и повторное использование памяти, занятой ресурсами, более не нужными программе. .NET Framework состоит из двух основных компонентов: общеязыковой исполняющей среды (CLR) и библиотеки классов Framework Class Library (FCL).

CLR можно рассматривать как среду, управляющую исполнением кода и предоставляющую ключевые функции, такие как компиляция кода, выделение памяти, управление потоками и сбор мусора. Благодаря использованию общей системы типов (common type system, CTS) CLR выполняет строгую проверку типов, а защита по правам доступа к коду позволяет гарантировать исполнение кода в защищенном окружении.

Библиотека классов .NET Framework содержит набор полезных типов, разработанных специально для CLR и доступных для многократного использования. Типы, поддерживаемые .NET Framework, являются объектно-ориентированными, полностью расширяемыми и обеспечивают эффективную интеграцию приложений с .NET Framework.

При компиляции кода для .NET Framework компилятор генерирует код на общем промежуточном языке (common intermediate language, CIL), а не традиционный код, состоящий из процессорных команд (рисунок 9.1).

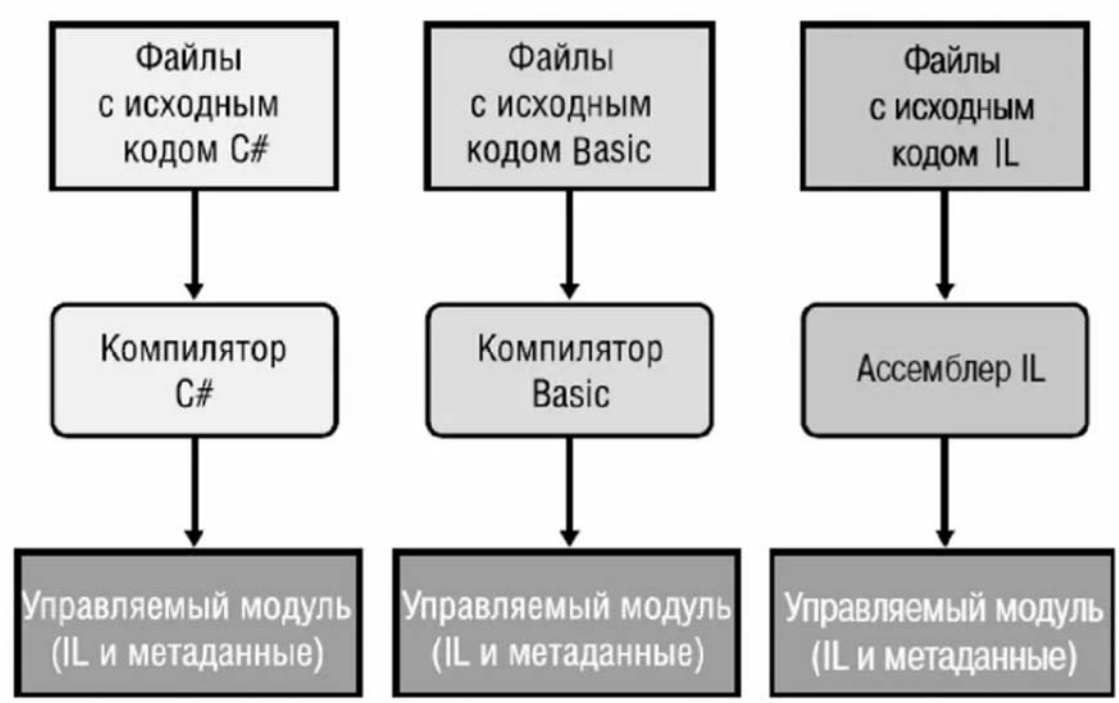

Рисунок 9.1 – Компиляция исходных файлов

При исполнении CLR транслирует CIL в команды процессора (отличие от Java).

Общеязыковая спецификация **(*Common Language Specification*, *CLS*)** – стандарт, который определяет минимальный набор правил, для разработчиков компиляторов, чтобы их языки интегрировались с другими. Microsoft разработала ряд таких языков: C++ с управляемыми расширениями, C #, Visual Basic .NET (сюда относятся и Visual Basic Scripting Edition, или VBScript, и Visual Basic for Applications, или VBA), а также JScript и др. (см. рисунок 9.1).

Все эти механизмы позволяют создавать собственные классы, упрощают установку приложений, предоставляют возможность работы на нескольких платформах, предоставляют возможность интеграции языков программирования и, соответственно, сервис сторонним приложениям; вследствие этого упрощается повторное использование кода и создается большой рынок готовых компонентов, реализуется автоматическое управление памятью (сбор мусора), гарантируется корректное обращение к существующим типам, единая структура обработки исключений для восстановления основного алгоритма, обеспечивается взаимодействие с существующим кодом.

Благодаря использованию общеязыковой исполняющей среды межъязыковая совместимость позволяет компонентам, реализованным с применением .NET Framework, взаимодействовать друг с другом независимо от языка, на котором они написаны. Так, приложение на Visual Basic .NET может обращаться к DLL, написанной на C#, а та, в свою очередь, способна вызвать ресурсы, созданные на управляемом C++ или любом другом .NET-совместимом языке. Межъязыковая совместимость поддерживается и для наследования в ООП

(объектно-ориентированном программировании), например, на основе C# можно объявлять классы в программах на Visual Basic .NET и наоборот.

Основные отличия и преимущества платформы .NET.

1 Единая программная модель (представляется единая общая объектно-ориентированная программная модель подключения функций ОС).

2 Упрощенная модель программирования (разработчику не нужно разбираться с реестром, глобально-уникальными идентификаторами (QUID), lUnknown, AddRef, Release, HRESUIT и т. д. Но в случае, если пишется приложение .NET Framework, которое взаимодействует с существующим не-.NET кодом, нужно разбираться во всех этих концепциях.

3 Отсутствие проблем с версиями. Архитектура .NET Framework позволяет изолировать прикладные компоненты, так что приложение всегда загружает версии компонент, с которыми оно строилось и тестировалось. Если приложение работает после начальной установки, оно будет работать всегда (в отличие от сложностей с версиями DLL).

4 Упрощенная разработка. Компоненты .NET Framework (их называют просто типами) теперь не связаны с реестром. По сути, установка приложений NET Framework сводится лишь к копированию файлов в нужные каталоги и установке ярлыков в меню Start, на рабочем столе или на панели быстрого запуска задач. Удаление же приложений сводится к удалению файлов.

5 Работа на нескольких платформах. При компиляции кода для .NET Framework компилятор генерирует код на общем промежуточном языке (common intermediate language, CIL), а не традиционный код, состоящий из процессорных команд конкретного процессора. Это значит, что можете развертывать приложение для .NET Framework на любой машине, где работают версии CLR и FCL.

6 Интеграция языков программирования. COM позволяет разным языкам *взаимодействовать*. .NET Framework позволяет разным языкам *интегрироваться*, т. е. одному языку использовать типы, созданные на других языках. На основе предоставляемой CLR общей системы типов (Common Type System, CTS).

7 Упрощенное повторное использование кода .NET Framework позволяет создавать собственные классы, предоставляющие сервис сторонним приложениям для многократного использования кода.

8 Автоматическое управление памятью (сбор мусора). CLR автоматически отслеживает использование ресурсов, гарантируя, что не произойдет их утечки. По сути, она исключает возможность явного «освобождения» памяти, используя сборщик мусора.

9 Проверка безопасности типов. CLR может проверять безопасность использования типов в коде, что гарантирует корректное обращение к существующим типам. Если входной параметр метода объявлен как 4-байтное значение, CLR обнаружит и предотвратит применение 8-байтного значения для этого параметра. Безопасность типов также означает, что управление может передаваться только в определенные точки (точки входа методов). Невозможно

указать произвольный адрес и заставить программу исполняться, начиная с этого адреса. Совокупность всех этих защитных мер избавляет от многих распространенных программных ошибок.

10 Развитая поддержка отладки. Поскольку CLR используется для многих языков, можно написать отдельный фрагмент программы на языке, наиболее подходящем для конкретной задачи, – CLR полностью поддерживает отладку многоязыковых приложений.

11 Единый принцип обработки сбоев. В CLR обо всех сбоях сообщается через исключения, которые позволяют отделить код, необходимый для восстановления после сбоя, от основного алгоритма. Такое разделение облегчает написание, чтение и сопровождение программ. Кроме того, исключения работают в многомодульных и многоязыковых приложениях. CLR также предоставляет встроенные средства анализа стека, заметно упрощающие поиск фрагментов, вызывающих сбои.

12 Безопасность. Используется «кодоцентрический» способ контроля за поведением приложений. Такой подход реализован в модели безопасности доступа к коду.

Практическая часть рассмотрена в [9, с. 30–37].

Задание для самостоятельной работы.

Разработать приложение согласно варианту, описанному в лабораторной работе № 4. Приложение должно предусматривать хранение исходных данных в файле, возможность добавления, изменения, удаления просмотра, поиска и сортировки информации.

Содержание отчета.

1 Титульный лист.
2 Цель работы.
3 Задание.
4 Ход выполнения задания.
5 Выводы.

Контрольные вопросы

1 Чем обусловлена упрощенная модель программирования в среде .NET?

2 За счет чего обеспечивается отсутствие проблем с версиями программного обеспечения, используемого при разработке приложений на платформе .NET?

3 Каким образом реализуется взаимодействие с существующим кодом?

4 Чем обусловлена мультиплатформенность приложений, создаваемых с помощью платформы .NET?

5 Что такое общеязыковая спецификация?

6 Какие приложения можно создавать с помощью платформы .NET и какие объектно-ориентированные языки при этом используются?

10 Лабораторная работа № 10. Разработка ASP-приложений на базе платформы .NET Framework

Цель работы: изучить принципы и средства создания приложений на платформе Microsoft ASP.NET, основные элементы приложений, методы их развертывания и исполнения.

Порядок выполнения работы.

1 Изучить теоретические сведения.
2 Выполнить задание.
3 Сделать выводы по полученным результатам.
4 Ответить на контрольные вопросы.
5 Оформить отчет.

Теоретические сведения.

Создание web-приложений на базе ASP.NET [9, с. 23–29].

ASP.NET – это часть технологии .NET, используемая для написания мощных клиент-серверных интернет-приложений. Она позволяет создавать динамические страницы HTML на основе множества готовых элементов управления, применяя которые, можно быстро создавать интерактивные web-сайты.

Формы ASP.NET.

Основным средством взаимодействия web-приложения с пользователем являются web-формы, представляющие собой аналог оконных форм Windows.

При создании формы программист вначале рисует форму в визуальном редакторе с помощью набора элементов управления, а затем настраивает обработку событий для формы. Обработчики могут относиться как к уровню страницы, так и к отдельному компоненту формы.

Рассмотрим простой пример формы:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default2.aspx.cs" Inherits="Default2" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
</head>
<body>
<form id="form1" runat="server">
<div>
<asp:Label ID="Label1" runat="server" Text="Введите свое имя"></asp:Label>
<asp:TextBox ID="TextBox1" runat="server" Height="46px" Width="257px"></asp:TextBox>
<asp:Button ID="Button1" runat="server" Text="Отправить" onclick="Button1_Click" />
```

```
</div>
<p>
<asp:Label ID="Label2" runat="server" Text=""></asp:Label>
</p>
</form>
</body>
</html>
```

В данной форме присутствуют четыре компонента: метка Label1, содержащая текст «Введите свое имя»; поле ввода TextBox1 для ввода имени; кнопка Button1; метка Label2. Вначале она пуста, а в процессе работы в нее выводится сообщение. В параметрах кнопки указан обработчик нажатия Button1_Click. При нажатии на кнопку введенное в поле ввода имя внедряется в сообщение, которое выводится в метке Label2.

Код обработки данной формы приведен далее.

```
using System;
using System.Collections.Generic; using System.Linq;
using System.Web; using System.Web.UI;
using System.Web.UI.WebControls;

public partial class Default2 : System.Web.UI.Page
{
protected void Page_Load(object sender, EventArgs e)
{
}

protected void Button1_Click(object sender, EventArgs e)
{
Label2.Text = "Привет, " + TextBox1.Text + "!"; }}
```

Задания для самостоятельной работы.

Реализовать на платформе ASP.NET графическое приложение для работы с данными, представленными коллекцией элементов. Приложение должно реализовать добавление, удаление, просмотр и поиск сведений. В качестве вариантов задания использовать:

1) телефонный справочник;
2) адресную книгу;
3) словарь;
4) интернет-магазин;
5) склад комплектующих.

Контрольные вопросы

1 Опишите технологию работы приложений ASP.NET.
2 Для чего используется разделение кода?
3 Что такое динамическая страница HTML?
4 Какие события страницы можно обрабатывать в ASP.NET?

11 Лабораторная работа № 11. Разработка GUI-приложения на C# в архитектуре клиент-сервер. Взаимодействие между клиентами реализовать через сокеты

Цель работы: приобрести навыки разработки GUI-приложений на языке C# в архитектуре клиент-сервер для платформы .NET.

Порядок выполнения работы.

1 Изучить теоретические сведения.
2 Выполнить задание.
3 Сделать выводы по полученным результатам.
4 Ответить на контрольные вопросы.
5 Оформить отчет.

Теоретические сведения.

Разработка GUI-приложения в архитектуре клиент-сервер на языке C# [9, с. 39–47].

Сокеты появились тогда, когда начали создавать первые системы распределенных вычислений. Их любят многие за гибкость и простоту в использовании. На их основе можно очень быстро и просто сделать первый прототип любого распределенного приложения. Присутствуют они и в .NET (System.Net.Sockets). В следующем листинге представлен простейший пример взаимодействия клиент-сервер:

```
private static void OnBeginAccept(IAsyncResult ar)
{
Socket listener = (Socket)ar.AsyncState
using (Socket client = listener.EndAccept(ar))
{
byte[] buffer = new byte[_bufferSize];
// Получаем данные от клиента.
client.Receive(buffer);
buffer = Encoding.UTF8.GetBytes(string.Format("Hello, {0}",
Encoding.UTF8.GetString(buffer)));
// Отправляем клиенту ответ.
client.Send(buffer); }}

static void Main(string[] args)
{
EndPoint endPoint = new IPEndPoint(IPAddress.Loopback, 8320);
using (Socket listener = new Socket(endPoint.AddressFamily,
SocketType.Stream, ProtocolType.Tcp))
using (Socket client = new Socket(endPoint.AddressFamily,
SocketType.Stream, ProtocolType.Tcp))
{ // Сокет будет слушать порт 8320 на локальном адресе
listener.Bind(endPoint);
// Максимальный размер очереди подключений.
```

```
// Нам для теста достаточно одного.
listener.Listen(1);
// Запускаем поток, который ждет подключения клиента.
listener.BeginAccept(new AsyncCallback(OnBeginAccept), listener);
// Подсоединяемся клиентом.
client.Connect(endPoint);
Console.Write("Введите сообщение: ");
string request = Console.ReadLine();
// Отсылаем данные на сервер.
int count = client.Send(Encoding.UTF8.GetBytes(request));
byte[] buffer = new byte[_bufferSize];
// Получаем данные с сервера.
client.Receive(buffer);
string response = Encoding.UTF8.GetString(buffer);
int index = response.IndexOf('\0');
Console.WriteLine(string.Format("Ответ сервера: {0}",
response.Remove(index))); } }
```

В .NET есть и более высокоуровневые надстройки над сокетами. Классы TcpClient и TcpListener предоставляют функциональность, осуществляющую коммуникации не на уровне сокетов, а на уровне потока ввода/вывода. Для этого используется класс NetworkStream. Данный класс является наследником System.IO.Stream, но переопределенные им методы чтения и записи не записывают данные ни в какие внутренние хранилища (файлы, массивы байт), а передают прямо в сокет на сервер.

Задания для самостоятельной работы.

В соответствии с заданием, полученным в лабораторной работе № 4, разработать графическое приложение на языке C# в архитектуре клиент-сервер. В разрабатываемом приложении предусмотреть многопоточность, хранение информации в файле на серверной стороне, а со стороны клиентской части приложения предусмотреть возможность просмотра, добавления, удаления, редактирования и поиска информации, хранящейся на сервере.

Контрольные вопросы

1 Что такое форма в .NET? Особенности ее создания.
2 Для чего предназначен и как используется элемент управления RadioBox?
3 Для чего предназначен класс TCP_Client?
4 Что указывается в параметрах создаваемого экземпляра объекта TCP_Client?
5 Какова последовательность действий при создании GUI-приложения?
6 Для чего предназначен и как используется контрол ListBox?
7 Каким образом организуется работа контролов CheckBox и RadioBox?
8 Для чего предназначен класс TCP_Listener?

12 Лабораторная работа № 12. Разработка GUI-приложения на C# , используя ADO.NET для доступа и работы с данными

Цель работы: приобрести навыки разработки приложений с использованием технологии ADO.NET.

Порядок выполнения работы.

1 Изучить теоретические сведения.
2 Выполнить задание.
3 Сделать выводы по полученным результатам.
4 Ответить на контрольные вопросы.
5 Оформить отчет.

Теоретические сведения.

Разработка приложения, использующего технологию ADO.NET [9, с. 49–60].

Преимущества и нововведения в ADO.NET.

Использование разъединенной модели доступа к данным.

В клиент-серверных приложениях традиционно используется технология доступа к источнику данных, при которой соединение с базой поддерживается постоянно. Однако после широкого распространения приложений, ориентированных на интернет, выявились некоторые недостатки такого подхода. Попробуем выявить некоторые из них.

Соединения с базой данных требуют выделения системных ресурсов, что может быть критично при большой нагрузке сервера. Хотя постоянное соединение позволяет несколько ускорить работу приложения, общий убыток от растраты системных ресурсов сводит преимущество на нет.

Специфика web-приложений не позволяет серверу в каждый момент времени знать, что необходимо пользователю. То есть до следующего запроса сервер не имеет представления, нужно ли еще поддерживать соединение.

Опыт разработчиков показал, что приложения с постоянным соединением с источником данных чрезвычайно трудно поддаются масштабированию.

Хотя существуют и другие недостатки, приведенные наиболее существенны. Все эти проблемы порождаются постоянным соединением с базой данных и решаются в ADO.NET, где используется другая модель доступа. Теперь соединение устанавливается лишь на то короткое время, когда необходимо проводить операции над базой данных.

В основе ADO.NET лежит новая объектная модель, в основе которой, в свою очередь, два разных подхода к работе с данными: присоединенный и отсоединенный.

Присоединенные объекты (такие как Connection, Transaction, DataReader, Command) предназначены для управления соединением, транзакциями, выборки данных и передачи изменений в БД. Отсоединенные объекты удобны тем, что

они позволяют работать с данными автономно (и манипулировать данными в произвольном порядке).

Хранение данных в объектах DataSet.

При работе с базой данных нам чаще всего приходится работать не с одной, а несколькими записями. Более того, данные эти могут собираться из различных таблиц. В разъединенной модели доступа к базе данных не имеет смысла соединяться с источником данных при каждом обращении. Исходя из этого, представляется логичным хранить несколько строк и обращаться к ним при необходимости. Для этих целей и используется DataSet.

DataSet представляет собой, по сути, упрощенную реляционную базу данных и может выполнять наиболее типичные для таких баз данных операции. Теперь, в отличие от Recordset, мы можем хранить в одном DataSet сразу несколько таблиц, связи между ними, выполнять операции выборки, удаления и обновления данных. Безусловно, разъединенная модель не позволяет постоянно отслеживать изменения в базе данных, производимые другими пользователями. Это может привести к ошибкам в таких приложениях, где информация должна обновляться каждый момент, – заказ билетов или продажа ценных бумаг. Однако в любую секунду может быть получена свежая информация из базы данных через вызов метода *FillDataSet*. Таким образом, DataSet остается чрезвычайно удобным для самого широкого класса приложений, когда необходимо получить данные из базы и как-либо обработать их.

Глубокая интеграция с XML.

Все более широко распространяющийся язык XML играет важнейшую роль в технологии ADO.NET и приносит еще несколько преимуществ по сравнению с традиционным подходом.

Заметим для начала, что практически любой XML-файл может быть использован как источник данных и на его основе может быть создан DataSet. Точно так же при передаче данных между компонентами или сохранении их в файл используется XML.

Программист, работающий с ADO.NET, не обязательно должен иметь опыт работы с XML или познания в этом языке. Все операции остаются прозрачными для разработчика.

Так как XML имеет текстовое представление, это позволяет передавать его по протоколам типа HTTP через брандмауэры. Дело в том, что системы защиты обычно настроены на фильтрацию двоичной информации, текстовая же информация легко пропускается, что облегчает создание распределенных приложений. XML представляет собой промышленный стандарт, поддерживаемый практически любой современной платформой, что позволяет передавать данные любому компоненту, умеющему работать с XML и выполняющемуся под любой операционной системой.

При передаче больших объемов информации при использовании технологии COM возникает проблема приведения типов данных, т. к. COM поддерживает лишь ограниченный их набор. Действительно, COM-маршаллинг может требовать длительной обработки, что негативно сказывается на

производительности приложения. XML же поддерживает неограниченное число типов и не требует их конверсии, что позволит ускорить процесс передачи данных.

Задания для самостоятельной работы.

1 Разработать приложение учета продажи авиабилетов в аэропорту.

2 Разработать приложение учета продаж компьютеров.

3 Разработать приложение учета посадок деревьев в районах города Минска.

4 Разработать приложение учета продажи лотерейных билетов различных лотерей.

5 Разработать приложение учета отгрузки стройматериалов.

6 Разработать приложение учета закупок оргтехники подразделениями университета.

7 Разработать приложение учета закупки канцелярских товаров кафедрами.

8 Разработать приложение учета продажи компакт-дисков.

9 Разработать приложение учета сдачи студентами лабораторных работ.

10 Разработать приложение учета посещаемости лекций студентами.

11 Разработать приложение учета сдачи спортивных нормативов студентами.

12 Разработать приложение учета регистрации абитуриентов.

13 Разработать приложение учета регистрации книг в библиотеке.

14 Разработать приложение учета отгрузки товаров на складе.

15 Разработать приложение учета отправки писем адресатам в организации.

Контрольные вопросы

1 Особенности архитектуры ADO.NET .

2 Отсоединенные объекты.

3 Для чего предназначен объект DataTable?

4 Для чего предназначены методы Fill и Update и что они получают в качестве параметров?

5 Для чего предназначен объект Parameter?

6 Для чего предназначен объект DataSet?

7 Для чего предназначен объект Connection?

8 Каких поставщиков данных можно использовать в ADO.NET?

9 В чем заключается различие между поставщиками данных OLE DB и ODBC?

10 С помощью какого элемента управления в форме отображаются данные, хранящиеся в таблице базы данных?

13 Лабораторная работа № 13. Разработка ASP-приложений на платформе .Net Framework

Цель работы: приобрести навыки разработки web-приложений с использованием технологии ASP.NET.

Порядок выполнения работы.

1 Изучить теоретические сведения.
2 Выполнить задание.
3 Сделать выводы по полученным результатам.
4 Ответить на контрольные вопросы.
5 Оформить отчет.

Теоретические сведения.

Разработка web-приложения, использующего технологию ASP .NET [9, с. 62–71].

Структура проекта и ASPX-файла.

ASP.NET-проект, создаваемый в Microsoft Visual Studio, состоит из нескольких файлов.

ASPX-файл является, по сути, обычным HTML, в котором можно использовать специальные теги. Дополнительные теги делятся на две категории: служебные, задающие параметры страницы и позволяющие внедрять код, и теги контролов, которые представляют собой новые интерфейсные конструкции, настройка которых инкапсулирована в тег.

Первому типу принадлежат серверные директивы, теги <%= %>, <% %>,<%# %>, <script runat="server">...</script>. Сначала рассмотрим директивы. Их синтаксис имеет вид:

```
<%@ Directive %>
```

Базовой директивой является @Page, которая описывает основные параметры страницы, такие как файл с исходным текстом, язык кода, параметры трассировки. Также используются директивы @Import, @Assembly, @OutputCache и др.

Теги <%= %> используются для выставки вычисляемых значений. Например, конструкция <%= System.DateTime.Now %> после компиляции будет вместо себя подставлять текущее время.

Теги <% %> позволяют вставлять код в страницу – то, что в основном используется в ASP и JSP. Например:

```
<%if (User.Identity.Name == "Admin")
{
%>
<a href="adminpage.htm">Перейти к странице администрирования<a>
<% }
%>
```
– отображает ссылку только для пользователя Admin.

Конструкция <%= %> является аналогом <%= %> для компонент с привязкой к данным. Они подставляют вместо себя значение, зависящее от текущего контекста данных. Приведем пример, который, однако, здесь подробно разобран не будет: привязка к данным будет обсуждена в отдельном разделе.

```
<asp:Repeater id="lstData" runat="server" >
<ItemTemplate>
<%# (Container.DataItem as DateTime).ToShortDateString() %>
</ItemTemplate>
</asp:Repeater>
```

Также в страницу можно вставлять дополнительные методы с помощью тега <script runat="server">:

```
<script language="C#" runat="server" >
void SubmitBtn_Click( object sender, EventArgs e )
{
txtMessage.Text = "Hello, world";
}
</script>
```

Параметр language тега script задает язык, на котором он написан.

Теперь рассмотрим теги контролов – они позволяют вставлять в страницу различные элементы управления. Многие из вас сразу вспомнят про стандартные элементы управления вроде кнопок или полей ввода. На самом деле, контролы ASP.NET могут представлять не только стандартные элементы, кодируемые одним HTML-тегом, но и довольно сложные DHTML-конструкции. При этом если при написании таких конструкций в обычном HTML вы получаете громоздкий и нерасширяемый код, то при использовании ASP.NET все это для вас инкапсулировано в одном теге. При этом код, который будет подставляться, генерируется на сервере, что позволяет изменять его в зависимости, например, от типа браузера.

Теги серверных контролов допустимы только внутри серверных форм – конструкций вида <form runat=«server»> ... </form>. Сами теги контролов пишутся в XML-стиле: с обязательным закрыванием и пространствами имен. Общий вид такого тега:

```
<пространство_имен:имя_тега runat="server" >
 </пространство_имен:имя_тега >
```

Пространства имен используются для того, чтобы случайно не совпали имена тегов. Стандартные элементы управления от Microsoft находятся в пространстве имен asp: <asp:Button />, <asp:TextBox /> и т. п.

Практическая часть.

Приступим к созданию нашего первого приложения на платформе ASP.NET. Запустим Visual Studio и выберем в меню File пункт New Web Site.

В появившемся диалоге выберем тип проекта ASP.NET Web Site. В поле Location выбираем File System, что означает, что наш сайт будет храниться на жестком диске компьютера (если на вашей машине установлен сервер IIS, то в качестве месторасположения создаваемого сайта можно указать его). В качестве языка программирования в поле Language выбираем C#.

Visual Studio создаст проект по умолчанию. Его структура будет видна в окне Solution Explorer.

Здесь можно увидеть два файла:

1) Default.aspx – экранная форма web-страницы;
2) Default.aspx.cs – исходный код формы.

Добавим на нашу форму поле ввода, метку и кнопку. Чтобы это сделать, откроем панель Toolbox и добавим поле ввода. Для этого перетащим элемент TextBox с панели на нашу страницу. В панели свойств установим для нашего поля ввода идентификатор *txtName*.

Таким же способом добавим на страницу Label и кнопку Button. Присвоим метке ID – l*blMessage*, Text – пустую строку, а Button – *btnHello* и «Поздороваться!» соответственно.

Теперь необходимо добавить некоторую логику. Наша страница будет здороваться с пользователем по имени, после того как он введет имя в поле ввода и нажмет кнопку. Для этого нужно написать обработчик события «пользователь нажал на кнопку» и вывести в метке соответствующий текст. Для создания обработчика дважды щелкните мышкой на кнопке. Visual Studio откроет файл с именем *WebForm1.aspx.cs*. К каждой форме невидимо привязываются файлы с *.aspx.cs* и *.aspx.resx*, но их не показывают в проекте, чтобы не нагружать его. В файле *.cs* содержится код страницы. После того как вы открыли его двойным щелчком на кнопке, в него добавился соответствующий обработчик. Введите в него следующий код:

```
protected void btnHello_Click(object sender, EventArgs e)
{
lbMessage.Text = "Добрый день," + txtName.Text + " добро пожаловать в ASP.NET!";
}
```

Теперь нужно скомпилировать проект. Это делается комбинацией клавиш Ctrl+Shift+B или командой меню Build -> Build WebSite.

После запуска приложения перед нами откроется окно web-браузера.

Задания для самостоятельной работы.

1 Разработать web-страницу с функциями калькулятора.

2 Разработать web-страницу, реализующую простейший учет продаж компьютеров. Данные хранятся в форме.

3 Разработать web-страницу, реализующую конвертер валют.

4 Разработать web-страницу, реализующую подсчет скидок по товарам.

5 Разработать web-страницу, реализующую расчет состояния автомобиля в зависимости от пробега.

6 Разработать web-страницу, реализующую функцию учета посещения студентами занятий. Предусмотреть использование DataGridView и подключение к источнику данных.

7 Разработать web-страницу, реализующую функцию учета закупки компьютеров кафедрами университета. Предусмотреть использование DataGridView и подключение к источнику данных.

8 Разработать web-страницу, реализующую функцию учета продажи сотовых телефонов в салоне связи. Предусмотреть использование Data-GridView и подключение к источнику данных.

Контрольные вопросы

1 Что такое платформа ASP.NET?

2 Какие функции выполняет платформа ASP.NET?

3 Как происходит компиляция кода в ASP.NET?

4 Что такое web-форма?

5 Для чего предназначены web-формы и какие функции они выполняют?

6 Какие основные элементы управления используются на web-формах?

7 Какую структуру имеет ASPX-файл?

8 Для чего предназначена конструкция <%= %>?

9 На каких языках программирования возможно написание приложений для ASP.NET-платформы?

10 Назовите основные файлы, содержащиеся в создаваемом проекте ASP.NET.

14 Лабораторная работа № 14. Разработка основных проектных решений на основе статических и динамических диаграмм и паттернов проектирования (MVC)

Цель работы: изучить средства организации паттерна MVC и ознакомиться с созданием приложения ASP.NET MVC 5.

Порядок выполнения работы.

1 Изучить теоретические сведения.
2 Выполнить задание.
3 Сделать выводы по полученным результатам.
4 Ответить на контрольные вопросы.
5 Оформить отчет.

Теоретические сведения.

Создание проекта.

MVC – это шаблон программирования, который позволяет разделить логику приложения на три части:

1) Model (модель). Получает данные от контроллера, выполняет необходимые операции и передаёт их в вид;
2) View (вид или представление). Получает данные от модели и выводит их для пользователя;
3) Controller (контроллер). Обрабатывает действия пользователя, проверяет полученные данные и передаёт их модели.

Создадим первое приложение с ASP.NET MVC, цель которого – дать некоторое начальное понимание работы. Приложение будет эмулировать работу книжного магазина со следующим функционалом: предоставление выбора книг; формирование покупки.

Создание проекта.

Откроем Visual Studio и создадим новый проект, выбрав **File -> New Project.** Назовем новый проект, например, BookStore_FIO, где FIO –ФамилияИО. Затем в окне создания нового проекта выберем MVC. В правой части окна изменим тип аутентификации приложения на No Authentication (рисунок 14.1).

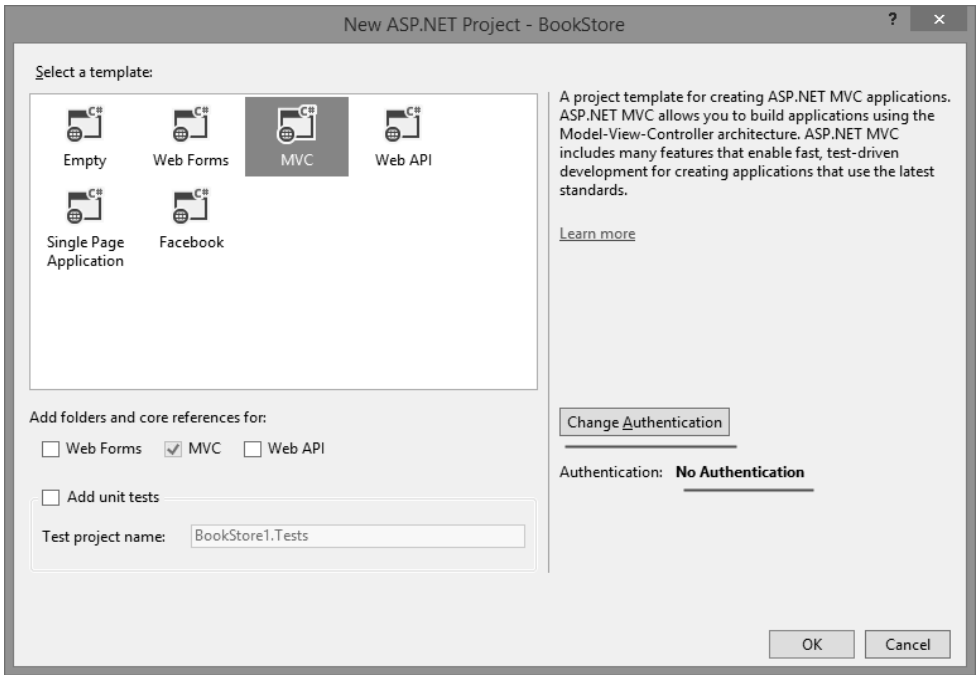

Рисунок 14.1 – Создание нового проекта ASP.NET MVC

После этого будет создан проект, который практически не обладает никакой функциональностью, хотя уже имеет базовую структуру.

Определим модели данных нашего приложения. Поскольку речь идет о книжном магазине, то такими моделями могут быть модель книги и модель покупки книги.

В проекте уже по умолчанию определена папка Models. В ней будут находиться наши модели. Нажмем на эту папку правой кнопкой мыши и в появившемся меню выберем **Add->Class**. Назовем первый новый класс Book и добавим в него код, описывающий модель книги:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace BookStore_ErinaMA.Models
{
    public class Book
    {
        // ID книги
        public int Id { get; set; }
        // название книги
        public string Name { get; set; }
        // автор книги
        public string Author { get; set; }
        // цена
        public int Price { get; set; }
    }
}
```

Подобным образом создадим второй класс с названием модели Purchase, который будет отвечать за отдельную совершаемую покупку книги.

EntityFramework.

Для работы с данными в ASP.NET MVC рекомендуется использовать фреймворк Entity Framework, хотя его использование необязательно и всецело зависит от предпочтений разработчика. Преимущество этого фреймворка состоит в том, что он позволяет абстрагироваться от структуры конкретной базы данных и вести все операции с данными через модель.

Сейчас наш проект не содержит библиотек EntityFramework. И чтобы их добавить в проект, воспользуемся пакетным менеджером NuGet. В окне Solution Explorer (Обозреватель решений) нажмем правой кнопкой мыши в структуре проекта на узел References и в появившемся меню выберем Manage NuGet Packages.

В окне управления пакетами NuGet в правом верхнем углу введите в поле поиска EntityFramework и нажмите Enter. После этого в среднем столбце будут отображены все найденные пакеты, которые имеют отношение к запросу, а самым первым будет пакет самого фреймворка EntityFramework, который нам и необходимо установить (рисунок 14.2).

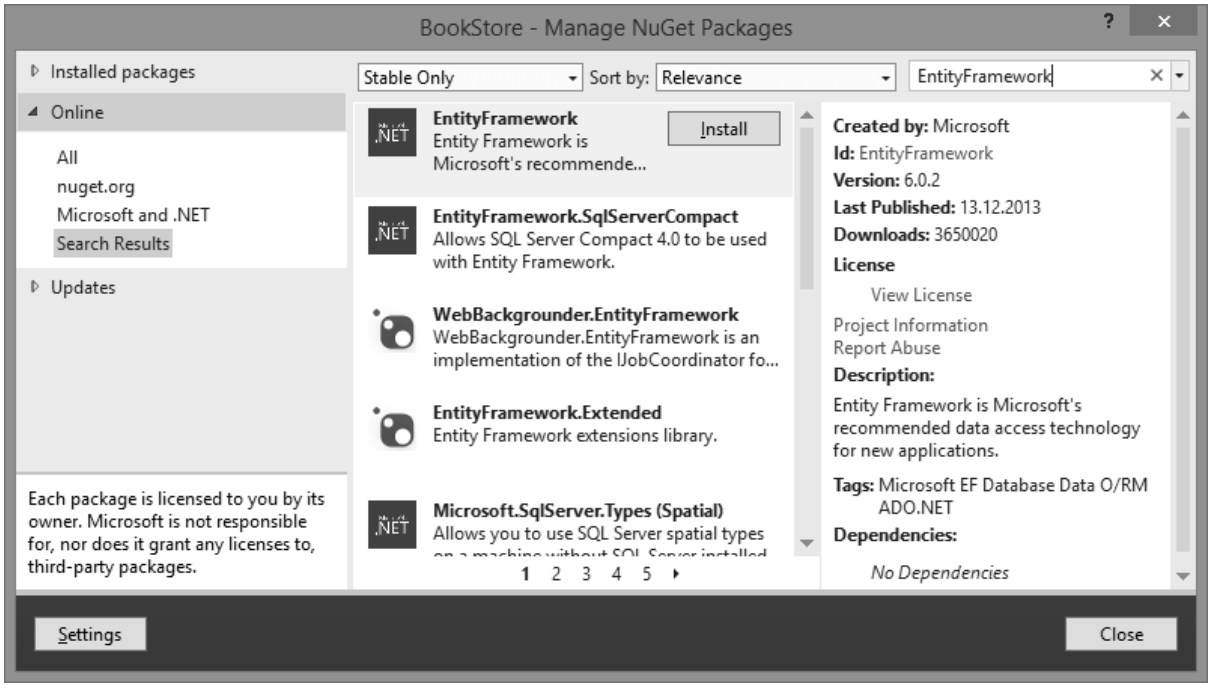

Рисунок 14.2 – Установка фреймворка EntityFramework

Запустим процесс установки пакета, нажав на кнопку Install.

Создание контекста данных.

После завершения установки создадим контекст данных. Контекст данных использует EntityFramework для доступа к БД на основе некоторой модели. Добавим в папку Models новый класс BookContext.

Для использования класса DbContext подключите библиотеку System.Data.Entity.

Создание контроллера и представлений.

Для контроллеров предназначена папка Controllers. По умолчанию при создании проекта в нее добавляется контроллер HomeController, который практически не имеет никакой функциональности.

В контроллере определены по умолчанию три метода: Index, About и Contact. Далее подключаем пространство имен моделей, после чего создается объект контекста данных, через который мы будем взаимодействовать с БД:

```
BookContext db = new BookContext();
```

Используя свойство db.Books, получаем из базы данных набор объектов Book. Для этого необходимо передать этот набор в представление.

Создание представления.

Для передачи списка объектов Book в представление используем объект ViewBag. ViewBag – объект, позволяющий определить любую переменную и передать ей некоторое значение.

Для представлений в проекте предназначена папка Views. По умолчанию в этой папке уже есть подкаталог для представлений контроллера Home, в котором три представления: About.cshtml, Contact.cshtml и Index.chtml.

Данные для моделей по умолчанию.

Воспользуемся специальным классом, который добавляет начальные данные в БД автоматически. Для этого в папку Models добавим новый класс BookDbInitializer и приведем код следующим образом:

```
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;
using System.Web;

namespace BookStore_ErinaMA.Models
{
    public class BookDbInitializer : DropCreateDatabaseAlways<BookContext>
    {
        protected override void Seed(BookContext db)
        {
            db.Books.Add(new Book { Name = "Война и мир", Author = "Л. Толстой", Price = 220 });
            db.Books.Add(new Book { Name = "Отцы и дети", Author = "И. Тургенев", Price = 180 });
            db.Books.Add(new Book { Name = "Чайка", Author = "А. Чехов", Price = 150 });

            base.Seed(db);
        }
    }
}
```

Класс DropCreateDatabaseAlways позволяет при каждом новом запуске заполнять базу данных заново некоторыми начальными данными. В качестве таких начальных значений создаются три объекта Book. Используя метод db.Books.Add, мы добавляем каждый такой объект в базу данных.

Если открыть папку проекта на жестком диске и перейти к каталогу App_Data, то увидим только что созданную базу данных Bookstore.mdf, которая и хранит данные по умолчанию.

Запуская проект на выполнение, отображается web-страница в браузере с данными по умолчанию (рисунок 14.3).

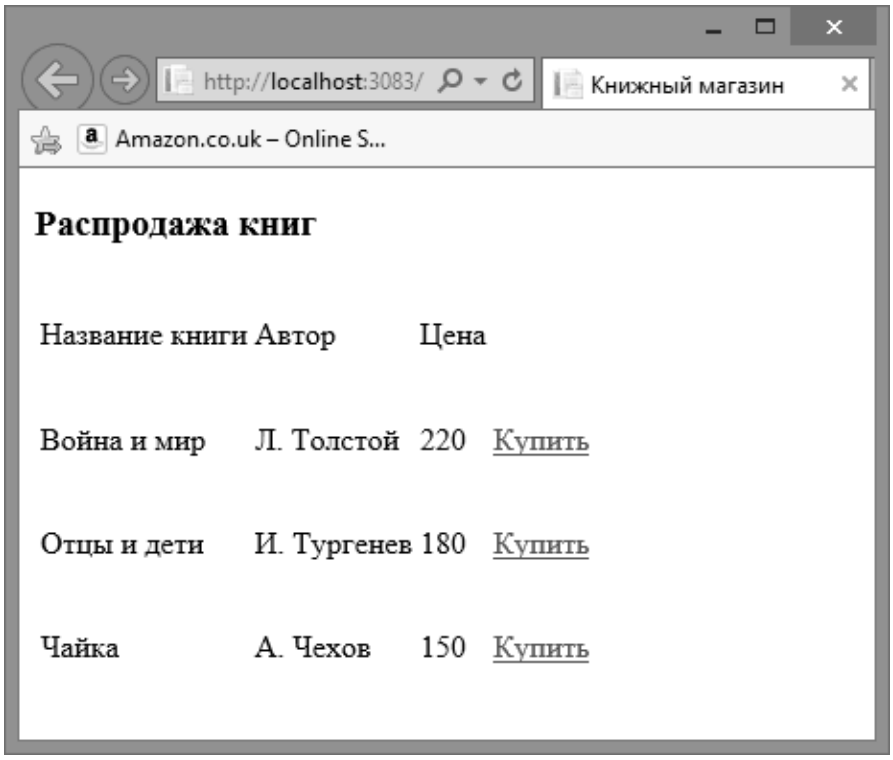

Рисунок 14.3 – Net-страница приложения с данными по умолчанию

Индивидуальное задание.

В соответствии с заданием, полученным в лабораторной работе № 13, разработать приложение на языке C# с использованием технологии MVC. В разрабатываемом приложении реализовать стилизацию страницы.

Контрольные вопросы

1 Как в Visual Studio создать проект ASP.NET MVC?

2 Что такое контроллер?

3 Что такое вид (view)?

4 ASP.NET MVC, позволяющий добавлять, редактировать и удалять данные.

5 Как реализовать контроллер и вид, формирующий выборку данных из модели данных Entity Framework в виде HTML-таблицы?

Список литературы

1 **Герман, О. В.** Администрирование и программирование распределенных приложений: учебно-методическое пособие / О. В. Герман, Ю. О. Герман. – Минск: БГУИР, 2016. – 240 с.

2 **Арлоу, Д.** UML и унифицированный процесс. Практический объектно-ориентированный анализ и проектирование / Д. Арлоу, А. Нейштадт. – Москва: Символ, 2016. – 624 с.

3 **ГОСТ 2.105–95 ЕСКД**. Общие требования к текстовым документам. – Минск, 1995. – 37 с.

4 **Гробов, И. Д.** ASP.NET 2.0: теория и практика / И. Д. Гробов. – Москва: Диалог-МИФИ, 2007. – 608 с.

5 **Гуриков, С. Р.** Информатика: учебник / С. Р. Гуриков. – Москва: ФОРУМ; ИНФРА-М, 2018. – 463 с.

6 **Дадян, Э. Г.** Методы, модели, средства хранения и обработки данных: учебник / Э. Г. Дадян, Ю. А. Зеленков. – Москва: Вузовский учебник; ИНФРА-М, 2017. – 168 с.

7 **Курняван, Б**. Программирование web-приложений на языке Java: пер. с англ. / Б. Курняван. – Москва: Лори, 2014. – 880 с.

8 **Отвагин, А. В.** Платформа Microsoft.NET: лабораторный практикум по дисциплине «Объектно-ориентированное проектирование и программирование» для студентов специальности 1-40 02 01 «Вычислительные машины, системы и сети» всех форм обучения / А. В. Отвагин, Н. А. Павлёнок. – Минск: БГУИР, 2011. – 46 с.

9 Распределенные информационные системы: лабораторный практикум для студентов специальности 1-40 01 02 «Информационные системы и технологии (в экономике)» всех форм обучения / В. Н. Комличенко [и др.]. – Минск: БГУИР, 2012. – 74 с.

10 **Флэнаган, Д.** JavaScript: карманный справочник: пер. с англ. / Д. Флэнаган. – 3-е изд. – Москва: Вильямс, 2015. – 314 с.

11 **Шилдт, Г.** Java 8. Полное руководство: пер. с англ. / Г. Шилдт. – Москва: Вильямс, 2018. – 1375 с.