ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - ВАРНА

КАТЕДРА “КОМПЮТЪРНИ НАУКИ И ТЕХНОЛОГИИ”

доц. д-р инж. Огнян Железов

Microsoft интернет технологии

(ASP.NET, ADO.NET, Web Services)

Настоящият учебник е разработен в съответствие с учебната програма по дисциплината “Microsoft Интернет технологии”, която се изучава в ОКС “Магистър” на специалност “Компютърни системи и технологии” в Технически университет – Варна.

Програмният код на всички примери е написан на C#. Освен C# желателно е читателят да има познания по HTML, тъй като в много случаи се генерира програмно HTML код, който се изпраща към потребителския браузър.

Авторът се е старал да дава изчерпателно описание на разглежданите обекти, атрибути и методи, така, че учебникът да може да предоставя и необходимите справочни данни на разработчиците на ASP.NET приложения.

Авторът изказва благодарност на рецензента, доц. Н. Николов за ценните препоръки по оформянето на учебника.

Съдържание

ASP.NET

Активните електронни страници (ASP) са технология, която широко се използува за създаване на динамични страници и Интернет приложения. Въпреки безспорните си предимства в сравнение с по-старите технологии (напр. CGI) ASP технологията има и някои недостатъци, като необходимостта от създаване на значителен по обем код за сравнително прости задачи и необходимостта от инсталиране на допълнителни разширения за някои често използвани функции, например за операции с файлове. За преодоляването на тези ограничения на ASP фирмата Microsoft разработи нова технология, наречена Microsoft ASP.NET, която е част от Microsoft .NET стратегията за развитие на WEB.

1. Предимства на ASP.NET

1.1. Независимост от езика за програмиране

Приложенията, създадени с .NET програмни езици се компилират до междинен език (IL) и в тоя вид се изпълняват от Common Language Runtime (CLR) – ядрото на NET Framework, с което на практика се губи разликата между използването на различни езици за програмиране.

ASP.NET поддържа голям брой .NET езици за програмиране. Microsoft предоставя компилатори за VB.NET, C#, Microsoft Visual C++, Microsoft Jscript.NET и Microsoft J#. Други производители са разработили .NET компилатори за Cobol, Pascal, Perl, Smalltalk и др.

1.2. Увеличена производителност

ASP.NET кодът се компилира еднократно при създаване или променяне на кода на ASP.NET модула. Когато една страница се заяви за пръв път, средата компилира кода и съхранява копие в буферната памет. Когато страницата се заяви отново се използва компилираното копие. Това значително ускорява изпълнението на ASP.NET приложенията.

Използвайки опита, натрупан от класическата ASP технология, фирмата Microsoft е взела мерки за повишаване на производителността на ASP.NET технологията. За целта се използват средства, като съхраняване в буферната (кеш) памет на компилирани копия на ASP.NET страниците и други (по принцип известни) технологии.

1.3. Сървърни контроли

Сървърните контроли са нови обекти, които нямат аналог в класическата ASP технология. Те предоставят функционалност (атрибути, методи и събития), коато е достъпна за кода, който се изпълнява от страна на сървъра, и същевремено изпращат към клиента (Web браузъра) HTML код, благодарение на което се визуализират в ASP.NET страницата. Това разширява значително възможностите на разработчика в сравнение с ASP технологията, в която той създава интерфейса на модулите само с класическите HTML контроли и COM компоненти, чийто код се изпълнява само от страна на Web браузъра.

1.4. Изпълняване на ASP.NET приложенията в отделен процес

За разлика от класическата ASP технология, при която ASP приложенията се изпълняват в същия процес и същото адресно пространство, в което се изпълнява и Internet Information Server (IIS), при ASP.NET технологията приложенията се изпълняват в отделен процес aspnet_wp.exe. Това предоставя по-голяма гъвкавост, стабилност и производителност, като не дава възможност на ASP.NET приложение да блокира работата на сървъра. Параметрите на процеса могат да се управляват чрез конфигурационния файл machine.config и се прилагат за целия сървър.

Този нов процес е независим от процеса, в който се изпълнява IIS. Той просто използва IIS за да получава заявки и да изпраща отговори. От това следва, че процесът на ASP.NET може да бъде създаван и унищожаван независимо от IIS. Например когато разработчикът тества ASP.NET приложение на своя компютър, процесът aspnet_wp.exe се стартира едва при първото заявяване на модул от приложението, независимо от това, че по правило IIS се стартира автоматично при зареждане на операционната система.

2. Структура на ASP.NET страницата

ASP.NET страниците могат да съдържат пет типа тагове, съдържащи програмен код, който се изпълнява от страна на сървъра:

1) Сървърни директиви
2) Скрипт блокове
3) Процедурни блокове
4) Блокове за извеждане (Write Blocks)
5) Директиви за вмъкване от страна на сървъра

2.1. Сървърни директиви

Синтаксис: **<%@ *server directives* %>**

Сървърните директиви задават стойност на ASP опции за Internet Service Manager. Тези блокове могат да се съдържат само в началото на ASP файла и то само веднъж. Например ако използвате за програмния код на страната на сървъра не подразбиращия се VB.NET а друг програмен език, например C#, в началото на ASP страницата трябва да се включи директивата

```
<%@ Page Language="C#" %>
```

За включване в в ASP.NET страницата на пространство от имена (съдържащо декларации на класове, необходими за кода на страницата) се използва сървърната директива Import, например

```
<%@ Import Namespace="System.Web.Mail" %>
```

2.2. Скрипт блокове

Скрипт блоковете са най-често използваните блокове за включване на програмен код директно в .aspx страницата. Скрипт блоковете, съдържащи кода на ASP.NET страницата, който се изпълнява от страна на сървъра трябва задължително да имат атрибут runat=”server”. Те могат да съдържат декларации на събитийни процедури на класа Page и на сървърните контроли, декларации на атрибути и методи и всеки друг код, който може да се включи в декларацията на класа, който представя ASP.NET модула. По време на изпълнение самостоятелната ASP.NET страница (т.е. която не е свързана с кодова страница) се третира като

клас, наследник на класа *Page. ASP.NET страницата не може да съдържа декларации на други класове.. При компилиране на страницата се генерира клас, който съдържа всички атрибути и методи и сървърни контроли като членове на класа. Кода на ASP.NET страницата, който се изпълнява от страна на сървъра става част от кода на класа, напр. събитийните процедури стават методи на класа. По-долу е даден пример за самостоятелна ASP.NET страница, която съдържа блок скрипт със събитийна процедура Page_Load. Тази събитийна процедура се изпълнява при всяко заявяване на страницата и извежда в прозореца на браузъра низа “Hello ASP.NET”

```
<%@ Page Language="C#" %>
<HTML><HEAD>
<TITLE>Hello page</TITLE>
<script runat="server">
  void Page_Load(object sender, EventArgs e)
  {
    Response.Write("Hello ASP.NET");
  }
</script>
</HEAD>
<BODY>
</BODY></HTML>
```

Пример 1 Самостоятелна ASP.NET страница със скрипт блок

1) Блокове за запис:

Синтаксис: **<%= *value* %>,**

където: *value* може да бъде променлива или израз.

Описание: При компилиране на сценария (кода на страницата) този блок се преобразува в оператор **Response.Write(*value*).** При изпълнение на сценария, като резултат от изпълнението на блока, към потребителския браузър се изпраща стойността на ***value***.

2.3. Процедурни блокове

Процедурните блокове са наследени от “класическата” ASP технология. Те се включват в HTML кода на страницата и могат да съдържат блок от инструкции, който се изпълнява от страна на сървъра.

---

* Класът Page е разгледан в т. 5

Използваният програмен език се определя от сървърната директива @Page.

Синтаксис: **<% *procedural script* %>**

където: *procedural script* може да бъде една инструкция или блок от инструкции, записани на един или няколко програмни реда.

2.4. Блокове за извеждане (Write Blocks)

Блоковете за извеждане също са наследени от ASP технологията. Те могат да съдържат само израз, предшестван от оператор за присвояване.

Синтаксис: **<% =*expression* %>**

При компилиране този блок се компилира като инструкция Response.Write(expression) , чрез която изчисленият при изпълнение израз expression се изпраща в изходния поток към потребителския браузър.

```
<%@ Page Language="C#" %>
<HTML><HEAD>
<TITLE>Hello page</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF" TEXT="#000000" LINK="#0000FF"
VLINK="#800080">
<%="Hello ASP.NET" %>
<HR>The current date and time is
<% Response.Write(DateTime.Now);%>
</BODY></HTML>
```

Пример 2 Самостоятелна ASP.NET страница, която съдържа процедурен блок и блок за извеждане

В примера блокът за извеждане съдържа като израз само текстова константа, която ще се изведе в прозореца на браузъра, Процедурният блок извежда тек. дата и време чрез метода Write на обект Response.

2.5. Директиви за вмъкване от страна на сървъра

Директивите за вмъкване от страна на сървъра дават възможност за вмъкване в няколко ASP.NET страници на един и същ код. При изпращане на документа към потребителския браузър, блокът за вмъкване се замества със съдържанието на зададения файл.

Синтаксис: **<!--#include virtual="*virtual_path_to_file*"-->**

или: **<!--#include file="*relative_physical_path_to_file*"-->**

където:

virtual_path_to_file е относителен път до файл. като за начална директория се счита кореновата (root) директория на сървъра.

relative_physical_path_to_file е файловата спецификация като абсолютен или относителен (спрямо текущата директория) път до файла

Ако сървърът е Apache Web server, използувайте варианта с опция **Virtual**. За Apache Web server тагът **include** с опция **Virtual** се изпълнява винаги, а с опция **File** – само в определени случаи. Ако сървърът е Microsoft IIS server, можете да използувате и двете опции. За Microsoft IIS server обаче е в сила ограничението, че с тагът **include** с опция **File** се изпълнява само за файлове, които се съдържат в текущата директория или нейни поддиректории. За вмъкването на файлове от директории над текущата директория в дървото на директориите е необходимо допълнително конфигуриране от системния администратор. Най-кратка е директивата **include** с опция **File** когато файлът, който се вмъква е в същата директория, в която е ASP.NET файла. В дадения по-долу пример в ASP.NET страницата се вмъква съдържанието на ASP.NET файла Show_Date.aspx.

```
<HR>The current date and time is
<% Response.Write(DateTime.Now);%>
```

Съдържание на файла Show_Date.aspx.

Вмъкнатият с директивата код се изпълнява в контекста на текущата ASP.NET страница. Така ако Show_Date.aspx се заяви самостоятелно ще се генерира грешка, защото подразбиращият се език на ASP.NET модулите е VB.NET, а кода в нея е написан на C#. При вмъкване в дадената по-долу ASP.NET страница грешка не възниква, защото за нея със сървърната директива @Page е зададен език C#.

```
<%@ Page Language="C#" %>
<HTML><HEAD> <TITLE> Include Example </TITLE> </HEAD>
<BODY BGCOLOR=#FFFF0>
<%="Hello ASP.NET" %>
<!--#include file="Menu_include.txt" -->
</BODY></HTML>
```

Пример 3 ASP страница, съдържаща директива за вмъкване от страна на сървъра

Важно е да се отбележи, че блоковете за вмъкване се изпълняват ако са включени в модул от интернет приложение (напр. ASP.NET страница), но няма да се изпълнят, ако са включени в HTML страница, тъй като сървърът изпраща HTML документите към потребителския браузър без никаква обработка. Ако искате да използвате блокове за вмъкване от страна на сървъра (SSI) в HTML страница, трябва да замените разширението на HTML файла от **htm** или **html** на **shtml**, и, което е не по-малко важно, да получите информация от системния администратор на сървъра дали той поддържа тази опция.

3. Отделяне на кода от потребителския интерфейс

ASP страниците съдържат в един файл HTML и скрипт, който се изпълнява от страната на сървъра, което прави страницата трудна за четене, настройка и поддържане. ASP.NET предлага решение на този проблем, като предоставя няколко възможности за отделяне на кода от потребителския HTML интерфейс на ASP.NET модулите.

a) С отделяне на кода в отделна кодова страница (code behind page), която може да бъде написана на кой да е от езиците, поддържани от .NET платформата.

b) Със създаване на потребителски контроли.

c) С включване на код за специфични функции в компоненти, които се изпълняват на сървъра и могат да се извикват от код, който се изпълнява на сървъра.

4. Web формуляри

Основното средство на Visual Studio NET за отделяне на кода от съдържанието е модела на WEB формулярите. Всеки WEB формуляр има модул със същото име, който съдържа т.н. *код зад страницата*[*]. Този модул съдържа кода на програмата (основно събитийните процедури), докато .aspx файла съдържа визуалните компоненти.

[*] Програмния код в кодовите страници някои автори наричат *бизнес логика*. Тъй като бизнес логиката изобщо, и програмния код в частност, не могат да се считат за еквивалентни, авторът не използва този термин.

Visual Studio .NET предоставя дизайнер за създаване на ASP.NET страници, аналогичен по вид и функции с дизайнера за проектиране на Windows форми. Разликата между Windows фóрмите и Web формулярите от гледна точка на потребителя е в това, че Windows фóрмите се визуализират като Windows прозорци, докато Web формулярите изпращат данни към браузъра на потребителя, които се визуализират като HTML документ. Visual Studio .NET използва модел за създаване на Web формуляри, при който с .aspx файла се асоциира файл с код, който съдържа клас, наследяващ класа Page. Този файл (разширението му зависи от използвания програмен език) се нарича "стоящ отзад код" (code behind page) или просто "кодова страница". Например ако .aspx страницата е записана във файл с име "Page1.aspx" и програмният език е C#, то кодовата страница ще бъде записана във файл с име "Page1.aspx.cs".

5. Класът Page – основният клас, който наследяват ASP.NET модулите

Класът Page се съдържа в именното пространство System.Web.UI. Той се асоциира с файловете, които имат разширение .aspx. Тези файлове се компилират по време на изпълнение като обекти от клас Page и се записват (кешират) в буферната памет на сървъра. Класът Page (и съответно неговият наследник, деклариран в кодовата страница) е контейнер за сървърните контроли, които се съдържат в .aspx страницата, с изключение на тези, които реализират интерфейса InamingContainer, а също и техните дъщерни контроли. Той съдържа и събитийните процедури на сървърните контроли, както и на самия обект Page. При създаване на кодовата страница е необходимо да се вмъкнат именните пространства System и System.Web.UI. При необходимост се включват и други именни пространства, съдържащи класовете, които се използват в кодовата страница. По-долу е даден пример за aspx файл и съответстващата му кодова страница. Кодовата страница разширява класа Page и в нея са декларирани обектни променливи за контролите, съдържащи се в .aspx страницата.

```
using System;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;
```

```
public class MyCodeBehind : Page {

   public TextBox        Name;
   public HtmlGenericControl   MySpan;

   public void SubmitBtn_Click(Object sender, EventArgs e) {

         MySpan.InnerHtml = "Hello, " + Name.Text + ".";
  }
}
```

Пример 4 Кодова страница на WEB фо́рма.

```
<%@ Page Inherits="MyCodeBehind" Src="PageExample.cs" %>
<html>
<body>

<center><form runat="server">

         <table>
           <tr>
             <td> Name: </td>
             <td> <asp:textbox id="Name" runat="server"/> </td>
           </tr>
           <tr>
             <td></td>
                          <td><asp:button  id="MyButton"  text="Click  Here"
OnClick="SubmitBtn_Click" runat="server"/></td>
           </tr>
           <tr>
             <td></td>
             <td><span id="MySpan" runat="server" /></td>
           </tr>
         </table>
</form></center>
  </body>
</html>
```

Пример 5 .aspx файл на WEB фо́рма.

5.1. Атрибути на обект от клас Page

Обектът Page предоставя атрибути и методи и генерира събития, които са свързани с изпълнението на кода .aspx страницата и нейната кодова страница ако е използван модела Web фóрмите. Той предоставя също така атрибути, които са референции към обекти и колекции, асоциирани с обекта Page, които се генерират от ASP.NET. Дадената по-долу таблица съдържа кратко описание на атрибутите.

5.1.1. Атрибути, представящи параметри на обект Page

ClientID	(наследен от класа Control, за четене, тип String) Съдържа идентификатор на обекта (в случая Page), генериран от ASP.NET.
ClientTarget	(за запис и четене, тип String) Задава и връща символен низ, съдържащ параметри на браузъра. Задаването на стойност позволява на потребителя да надпише (т.е. да промени) автоматично получените параметри на браузъра и така да укаже как да се визуализират елементите на Web фóрмата.
EnableViewState	(за запис и четене, тип Boolean) Задава и връща стойност, която показва, дали Web фóрмата съхранява външния си вид (включително съдържанието на сървърните контроли) след всяко повторно зареждане.
ErrorPage	(за запис и четене, тип String) Задава и връща адрес на страница, която да бъде заредена при възникване на грешка при изпълнение на кода на Web фóрмата.
ID	(за запис и четене, тип String) Задава и връща символен низ, съдържащ идентификатор на обекта Page.
IsPostBack	(за четене, тип Boolean) Съдържа а стойност, която показва дали страницата е била заредена повторно в резултат на изпращане на данни (стойност **true**) , или е била заредена за пръв път (стойност **false**).

IsValid	(за четене, тип Boolean) Съдържа стойност, която показва, дали въведените данни в страницата са валидни (стойност **true**).
TemplateSource Directory	(наследен от класа Control, за четене, тип String) Съдържа виртуалната директория, в която се съдържа .aspx страницата.
UniqueID	(наследен от класа Control, за четене) Съдържа уникален идентификатор на обекта (в случая обекта Page), включващ и името на контейнера на обекта
Visible	(за четене и запис, тип Boolean) Задава стойност, която определя дали aspx страницата се визуализира.

Повечето от изброените атрибути се използват с подразбиращите се стойности. Атрибутът **IsPostBack** дава възможност да се определи дали aspx страницата се зарежда за пръв път (тогава атрибутът има стойност false) и при необходимост да се извършат инициализации на променливи. Код за инициализация може да се включи в събитийната процедура Page_Load, както е показано в следващия пример:

```
protected void Page_Load(Object Src, EventArgs E){
   if(!Page.IsPostBack){ //Проверка дали е първо извикване на .aspx страницата
        //Инициализиране на списъчен контрол ListBox1 при първо извикване
        ListBox1.Items.Add("Bulgaria");
        ListBox1.Items.Add("Sweden");
        ListBox1.Items.Add("France");
        ListBox1.Items.Add("Italy");
  }
}
```

Пример 6 Използване на атрибута IsPostBack на обект Page за инициализиране в събитийната процедура Page_Load.

5.1.2. Атрибути –референции към обекти (обект Page)

Тези атрибути, представени в дадената по-долу таблица съдържат обектни променливи за обекти, асоциирани с обект Page.

Application (за четене) Съдържа референция за обект HttpApplicationState за текущата Web заявка.

Cache (за четене) Съдържа референция за обект Cache, асоцииран с приложението, в което се съдържа WEB фóрмата.

NamingContainer (наследен от класа Control) (за четене) Съдържа референция към контейнера на сървърните контроли, включени във Web фóрмата.

Page (наследен от класа Control), (за четене) Съдържа референция към екземпляра на класа Page., който съдържа сървърните контроли.

Parent (наследен от класа Control), (за четене) Съдържа референция към родителския обект в йерархията от обекти.

Request (за четене) Съдържа референция към обекта HttpRequest, асоцииран с обекта Page. Този обект предоставя атрибути и методи, свързани с получаването на данни от HTTP заявка от клиента.

Response (за четене) Съдържа референция към обекта HttpResponse, асоцииран с обекта Page. Този обект предоставя атрибути и методи, свързани с изпращането на данни по HTTP протокол към клиента.

Server (за четене) Съдържа референция към обект Server (екземпляр на класа HttpServerUtility).

Session (за четене) Съдържа референция към текущия обект Session, създаден за потребителската сесия от ASP.NET.

Site (наследен от класа Control), (за четене) Съдържа референция към обект Site, който съдържа информация за Web сайта, в който се съдържа .aspx страницата.

Trace (за четене) Съдържа референция към обект TraceContex за текущата Web заявка.

User (за четене) Съдържа информация за потребителя.

ClientScript (за четене) Съдържа референция към обект от клас ClientScriptManager, който дава възможност за добавяне на скриптове към ASP.NET страницата.

5.1.2.1. Добавяне на JavaScript блок към ASP.NET страница чрез обект ClientScript

```
string scriptKey = "intoPopupMessage:" + this.UniqueID;
string scriptBlock =
@"<script language=""JavaScript"">
  <!--
    alert(""%%POPUP_MESSAGE%%"");
  // -->
  </script>";
scriptBlock = scriptBlock.Replace("%%POPUP_MESSAGE%%", "My Message");

ClientScript.RegisterStartupScript(GetType(), scriptKey, scriptBlock);
```

В примера се използва метода RegisterStartupScript на обект ClientScript за добавяне на блок с JavaScript код към ASP.NET страницата. Методът добавя блока в края на ASP.NET фòрмата. Като резултат се извежда диалогова кутия alert с текст "My Message".

5.1.3. Атрибути – референции към колекции (обект Page)

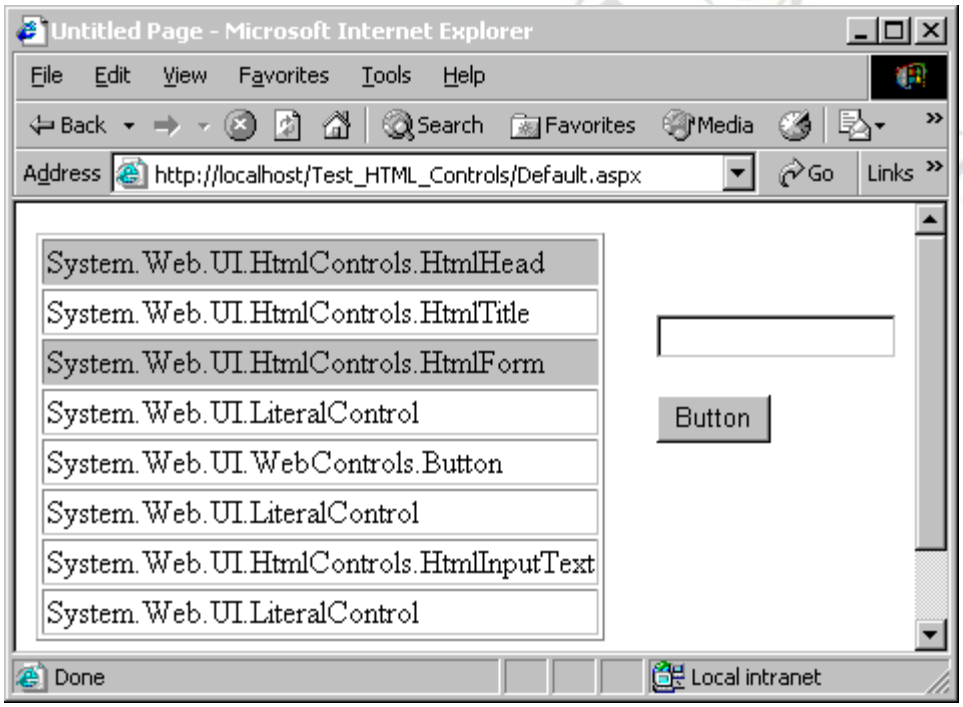
Тези атрибути, представени в дадената по-долу таблица съдържат референции към колекции, асоциирани с обект Page. Колекциите предоставят набор от атрибути и методи за операции с до елементите на колекцията

Controls	(наследен от класа Control), (за четене) Съдържа референция към обект от класControlCollection - колекция от обектни променливи (референции) за контролите, включени във Web фóрмата.
Validators	(за четене) Съдържа референция към колекцията от валидиращи контроли на Web фóрмата.

Колекцията Controls на обекта Page дава на програмиста достъп до сървърните контроли, съдържащи се в .aspx страницата. Даденият по-долу пример извършва итерация през елементите на колекцията Controls на обект Page и на вложените в него обекти.

```
   protected void ShowControls(object sender, EventArgs e)
   {
      string s = "<table border>";
      foreach (Control c in Page.Controls) //цикъл за колекцията Controls на
Page
      {
         if (c.Controls.Count > 0) //Проверка дали контролът има колекция
         {
            s += "<tr><td bgcolor='#ffffe0'>" + c.GetType().ToString() +
"</td></tr>";
            foreach (Control c2 in c.Controls) //цикъл за вложена колекция
            {
               s += "<tr><td>" + c2.GetType().ToString()+"</td></tr>";
            }
         }
      }
      s += "</table>";
      Response.Write(s); //иззвеждане на генерираната HTML таблица
   }
```

Пример 7 Извеждане на колекцията от контроли на обект Page и на вложените в него обекти.



Фиг. 1 Резултат от изпълнението на скрипта

В примера се генерира динамично код на HTML таблица с типовете контроли, които се съдържат в ASP.NET страницата. Външният цикъл foreach извършва итерация за контролите, вложени в обект Page, и добавя към кода на таблицата ред със сив цвят с текст, показващ типа на контрола. В случая обекта Page съдържа обект от тип System.Web.UI. HtmlControls.HtmlHead, представящ заглавната част на ASP.NET страницата, и обект от тип System.Web. UI.HtmlControls.HtmlForm

6. Директивата @Page

Директивата @Page служи за задаване на атрибути на ASP.NET страницата (.aspx файла) , които се използват при нейната обработка от парсера и компилатора. Директивата @Page е незадължителна, но по подразбиране се включва от Visual Studio.NET в началото на всяка ASP.NET страница. Тя има следния синтаксис, който включва името на директивата и двойки атрибут-стойност:

<%@ Page *attribute*="*value*" [*attribute*="*value*"…] %>

Пример:

```
<%@ Page Language="C#"  CodeFile="Default.aspx.cs" Inherits="_Default"
%>
```

Пример 8 Директива @Page, чрез която се задават програмният език, пътя до кодовата страница и името на класа – наследник на класа Page, който е деклариран в нея.

6.1. Атрибути на директивата @Page

AspCompat

Когато има стойност **true** разрешава изпълнението на страницата в режим на единична апартаментна нишковост (single-threaded apartment ‘STA’ thread). Това дава възможност на .aspx страницата да използва STA компоненти, например компоненти, разработени с Microsoft Visual Basic 6.0. Също така задаването на стойност **true** на този атрибут позволява използването на COM+ 1.0 компоненти, което предоставя достъп до обекти на класическите Active Server Pages (ASP). Обръщение към тях може да се осъществи чрез обекта ObjectContext или чрез метода OnStartPage. Подразбиращата се стойност на атрибута е **false**.

Задаването на стойност **true** на този атрибут може да доведе блокиране на изпълнението на ASP.NET страницата.

AutoEventWireup
Показва дали събитията на на .aspx. страницата се свързват автоматично (стойност **true**) със събитийни процедури, чиито имена са зададени по конвенцията *Page_eventname*, където *eventname* е името на съответното събитие.. При подразбиращата се стойност true не е необходимо програмистът да указва кое от събитията на страницата с коя събитийна процедура трябва да се свърже. Като неудобство за потребителя може да се посочи това, че в този случай той не може сам да определя името на събитийните процедури, а трябва да приеме имената, които им се дават по подразбиране, напр. Page_Init и Page_Load.

По-долу е даден пример за свързване на събитийна процедура със събитие за Visual Basic,NET и за C#

За Visual Basic,NET

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
```

За C#

```
private void InitializeComponent()
{
  this.Load += new System.EventHandler(this.Page_Load);
}
```

Buffer
Определя дали е разрешено буферирането на данните, изпращани от ASP.NET модула към потребителския браузър (стойност true) или е забранено (стойност false). За разлика от "класическото" ASP, където буферирането по подразбиране е забранено, в ASP.NET буферирането по подразбиране е разрешено. При разрешено буфериране даните, изпращани от ASP.NET модула се записват в буферна памет и се изпращат наведнъж към заявителя (WEB браузъра) след като завърши обработването на ASP.NET модула. Програмистът може да предизвика изпращане на натрупаните в буфера данни като изпълни метода Flush на обект Response.

CodeFile
Задава пътя до кодовата страница (code-behind file) на .aspx страницата. Този атрибут се използва с атриута Inherits за да асоциира кодовата страница на ASP.NET модула с .aspx файла. Пример:

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

Пример 9 Директива @Page с атрибути **CodeFile** и **Inherits**

CodePage
Атрибутът задава целочислена стойност - идентификатор на кодовата таблица, която се използва при кодиране на съдържанието на .aspx страницата, изпращана към Web браузъра. Алтернатива на този атрибут е атрибутът responseEncoding, на който може да се присвои стойност в конфигурационния файл Web.config. Списък от стойностите на CodePage е даден в документацията за атрибута на класа Encoding. За кодиране на кирилица стойността е 1251.

Explicit
Определя дали aspx страницата се компилира с опцията Explicit на Visual Basic.NET. Стойност **true** означава, че е задължително декларирането на всички променливи в кода на VB.NET с някоя от директивите Dim, Private, Public, или ReDim. При стойност false декларирането на променливите във VB.NET не е задължително. Използването на недекларирани променливи обаче се счита за лоша практика и води до трудно откриваеми грешки, поради това, ако програмирате на VB.NET, ви препоръчваме да включите опцията Explicit=true в директивата Page.

7. Клас HttpApplicationState и обект Application

Обектът Application се използва за поддържане на обща информация за цялото приложение, която да бъде достъпна до всички потребители в приложението. Обектът Application се създава като екземпляр на класа HttpApplicationState, когато първият потребител заяви ресурс от ASP.NET приложението за първи път от последното стартиране на Web сървъра. Обектът Application ще бъде достъпен за всички потребители, които посещават сайта до тогава, докато не изтече

контролното време (таймаут) на всички потребителски сесии или Web сървъра не бъде рестартиран. Обектът предоставя следните методи и колекции:

Методи:	Lock	Заключва променливите на приложението
	Unlock	Отключва променливите на приложението
	Remove	Унищожава променлива на приложението
Колекции	Contents	Връща референция към обект Application. Съдържа всички променливи на приложението
	StaticObjects	[*]Съдържа всички статични обекти обекти с област на действие в рамките на приложението.

7.1. Създаване на променливи на приложението

Към обекта Application динамично могат да бъдат добавяни свойства, които се разглеждат като глобални променливи, достъпни за всяка страница от приложението и всеки потребител на това приложение. Достъпът до променливите на приложението може да се осъществява непосредствено чрез обекта Application:

Application[“propertyname”] = value;

или чрез атрибута Contents, който връща референция към обекта Application за текущото приложение.

Application.Contents[“propertyname”] = value;

Създадената с тази инструкция променлива на приложението е достъпна от всяка страница на Web сайта в рамките на приложението.

[*] Добавянето на статични обекти към приложението е дадено в описанието на файла GLOBAL.asax

7.2. Унищожаване на променливи на приложението -метод Remove

Методът Remove премахва именуван обект от колекцията на обект Application.
Декларация (C#)
public void Remove(string name)

Например ако искаме да унищожим променливата на приложението с име “var1” можем да използваме следната инструкция

Application.Remove(“var1”);

7.3. Заключване на променливи на приложението -метод Lock

Методът Lock предизвиква временно заключване на всички променливи на приложението, така че те могат да бъдат променяни само от потребителя, който е извършил заключването. Така се избягва възникването на конфликт от едновременно променяне на съдържанието на една и съща променлива от няколко потребители.
Декларация (C#): public void Lock()

7.4. Отключване на променливи на приложението -метод Unlock

Методът Unlock премахва заключването на променливите на приложението, така че те да могат да бъдат променяни от всички потребители.
Декларация (C#): public void Unlock()

По-долу е даден пример за запис на нова стойност в променлива myvar като преди присвояването на стойността променливите се заключват преди записа и се отключват след изпълнението на командата за запис. Новата стойност на променливата се извежда на браузъра на потребителя с метода Write на обекта Response.

```
protected void Page_Load(object sender, EventArgs e)
{
   Application.Lock();
   Application( "myvar" ) = "Hello World!";
   Application.Unlock();
```

```
    Response.Write Application( "myvar" );
}
```

Пример 10 Създаване на променливи на приложението чрез обект Application.

7.5. Колекция Contents (обект Application).

Съдържа всички променливи на приложението. Колекцията на обект Application е от типа NameObjectCollection, което означава, че елементите имат ключ от тип string и стойност от базовия тип Object, следователно в променливите на приложението може да се записват стойности от всеки тип данни.

Например, ако искаме да определим колко потребителя са свързани логически в едно и също време, то може да инициализираме променлива на приложението VisitorNum в събитийната процедура Application_Start на модула Global.asax.

```
void Application_Start(object sender, EventArgs e)
{
  //Код, който се изпълнява при първо стартиране на приложението
  Application["VisitorNum"]=0;
}
```

След това, когато всеки потребител се свързва към ASP.NET приложението, тази глобална променлива се увеличава с единица и се извежда общият брой на текущите потребители. За целта в събитийната процедура **Session_OnStart,** включена във файла **Global.asax** трябва да се изпълни

```
void Session_Start(object sender, EventArgs e)
{
    Application["VisitorNum"]=
    Convert.ToInt32(Application["VisitorNum"]) + 1;
}
```

Аналогично когато потребителят се изключи, в събитийната процедура **Session_OnEnd** от файла **Global.asax** трябва да се изпълни:

```
   void Session_Start(object sender, EventArgs e)
   {
       Application["VisitorNum"]=
       Convert.ToInt32(Application["VisitorNum"]) + 1;
}
```

Така в променливата на приложението VisitorNum винаги ще се съдържа текущият брой на потребителите на сайта.

Контролни въпроси 1

a) Какво време на живот имат променливите, създадени чрез обект Application
b) Какъв е типът на създаваните променливи
c) Защо се налага да се използва заключване при работа с променливи, създадени чрез обект Application

8. Клас HttpSessionState и обект Session

Обектът Session представя потребителска сесия за Интернет приложение. Обектът Session се създава като екземпляр на класа HttpSessionState, когато нов потребител заяви ресурс от ASP.NET приложението и за него се стартира нова сесия Обектът предоставя следните методи, атрибути и колекции:

8.1. Атрибути на обект Session

Contents	Връща референция към обект Session за текущата сесия. Съдържа колекция от всички променливи на сесията
SessionID	Връща уникален идентификатор на сесията.
Timeout	Задава контролно време за прекратяване на сесията в минути.
IsCookieless	Връща стойност, която показва, дали идентификатора на сесията ID се изпраща добавен съм адреса (URL) или се записва в HTTP бисквидка.
IsNewSession	Връща стойност, която показва, дали сесията е

	била създадена в рамките на текущата заявка.
IsReadOnly	Връща стойност, която показва, дали сесията е само за четене (read-only).
IsSynchronized	Връща стойност, която показва, дали достъпа до колекцията от сесийни променливи е синхронизиран (thread safe).
LCID	Задава идентификатор за местоположението на потребителя
StaticObjects	*Връща коленция, която съдържа всички статични обекти с област на действие в рамките на сесията

Ако даден сайт има 200 потребителя в определен момент, то ще трябва да има стартирани 200 обекта Session. Обекта Session позволява да се създават потребителски свойства, които ще обхващат глобално конкретния потребител, който влиза в даденият Web сайт. Обектите Session са тези, които дават възможност да се осигурят уникални и персонални преживявания за потребителите на един Web сайт. Например може да се запита един потребител какви цветове предпочита и какво е неговото име и да се адаптира съдържанието на HTML страниците, която се изпраща към потребителя, според въведената информация. Като се използват свойствата на обекта Session, може да се създават ASP страници, които са оформени според предпочитанията на потребителя и може да се обръщаме към потребителя по име, докато трае неговото посещение на страницата.

Сесийни променливи

По подразбиране сесийните променливи се съхраняват в паметта на Web сървъра (InProc Mode). Достъпът до сесийните променливи може да се осъществява непосредствено чрез обекта Session:

```
Session[“propertyname”] = value;
```

или чрез атрибута Contents, който връща референция към обекта Session за текущата сесия

* Добавянето на статични обекти към приложението е дадено в описанието на файла GLOBAL.asax

Session.Contents[“propertyname”] = value;

Създадената с тази инструкция сесийна променлива е достъпна от всяка страница на Web сайта в рамките на сесията. Обектът Session се унищожава, когато потребителя напусне Web сайта, или когато е изминало определено време без активност от негова страна, (time-out) на сесията.

8.2. Идентификатор на сесията SessionID

Атрибутът SessionID се използва за да идентифицира потребителя и да го свърже със сесийните данни на сървъра. Стойността на SessionID се генерира от ASP.NET чрез генератор на случайни числа и се записва като сесийна бисквидка от потребителския браузър. При всяко следващо заявяване стойността на SessionID се изпраща в бисквидка към ASP.NET приложението или се предава като се добавя към адреса.

8.3. Конфигуриране на предаването на идентификатора на сесията

Разработчикът може да определи как ще се предава идентификатора на сесията SessionID чрез конфигурационния елемент sessionState на файла Web.config на приложението. Елементът sessionState притежава атрибут cookieless, стойността на който определя начина, по който ще се предава SessionID По-долу е дадено значението на стойностите, които могат да се задават на този атрибут:

AutoDetect	ASP.NET определя дали браузърът, изпратил заявката, поддържа бисквидки. Ако браузърът поддържа бисквидки, SessionID се изпраща чрез бисквидка; в противен случай се изпраща добавен към адреса Ако браузърът поддържа бисквидки но те са забранени от потребителя, въпреки това SessionID се изпраща чрез бисквидка.
UseCookies	SessionID се изпраща чрез бисквидка независимо от това дали браузърът поддържа бисквидки или не.
UseDeviceProfile	ASP.NET определя дали да използва бисквидки въз

	основа на стойността на HttpBrowserCapabilities. Ако настройката HttpBrowserCapabilities показва, че браузърът поддържа бисквидки, те се използват, в противен случай противен случай SessionID се изпраща добавен към адреса.
UseUri	SessionID се изпраща добавен към адреса независимо от това дали браузърът поддържа бисквидки или не.

По-долу е даден пример за конфигурационен файл Web.config, в който на атрибута cookieless на елемента sessionState е зададена стойност UseUri, в резултат на което SessionID се предава добавен към адреса на заявените модули от приложението. Освен това чрез атрибута timeout е зададено контролно време 10 минути за изтичане на сесията.

```
<?xml version="1.0"?>
<configuration>
   <appSettings/>
   <connectionStrings/>
   <system.web>
     <sessionState
       cookieless="UseUri"
       timeout="10">
     </sessionState>
       <compilation debug="false" />
       <authentication mode="Windows" />
   </system.web>
</configuration>
```

Пример 11 Конфигуриране на предаването на SessionID без използването на бисквидки

Ако се изведе стойността на SessionID в прозореца на браузъра, както е показано в следващия пример, се вижда, че стойността съвпада с добавения към адреса на модула низ

```
   protected void Page_Load(object sender, EventArgs e)
   {
      Response.Write("<BR>SessionID=" + this.Session.SessionID);
   }
```

Пример 12 Извеждане на идентификатора на сесията чрез събитийната процедура Page_Load

Ако например изведената стойност на SessionID е “cvwbspyavqdgwu55nd5z14uq”, то адреса, с който се заявява модула (ако се тества ASP.NET приложението на локално инсталирания IIS ще има вида:

```
http://localhost/DemoSession/(S(cvwbspyavqdgwu55nd5z14uq))/Default.aspx
```

Пример за добавяне на стойността на SessionID към адреса, с който се заявява модул от ASP.NET приложение.

Идентификаторът SessionID се изпраща между сървъра и клиента (Web браузъра) като обикновен (некриптиран) текст. В случай, че се изисква по-голяма сигурност при предаването на SessionID, и по този начин да се ограничи на възможността от неоторизиран достъп до сесийните данни, се препоръчва да се използва обмен по криптиран канал (SSL).

8.4. Прекратяване на сесия - метод Abandon

Методът **Abandon** прекъсва потребителската сесия и унищожава всички сесийни променливи, свързани с потребителя. При заявяване на нова страница се инициира началото на нова сесия.
Например ако записвате име и парола за текущия потребител в сесийни променливи, можете да използувате методът Abandon за да създадете страница за прекратяване на сесията (Log Out). Извикването на метода Abandon ще унищожи променливите за име и парола, записани като сесийни променливи на потребителския компютър

Важно е да се запомни, че при извикването на метода Abandon сесийните променливи ще се унищожат едва когато завърши обработката на текущата ASP страница.

Това се илюстрира от следния пример. Сесийната променлива "var1" в дадената по-долу .aspx страница запазва стойността си и след извикването на метода Abandon, но ако се предизвика повторно зареждане на страницата посредством щракване с мишката върху хипервръзката, се вижда, че сесийната променлива вече съдържа празен низ. Проверката за първо зареждане на .aspx страницата се прави чрез атрибута IsPostBack на обект Page.

```
protected void Page_Load(object sender, EventArgs e)
{
  if(!this.IsPostBack) //Проверка за първо зареждане
  {
    Session["var1"]="Hello";
    Session.Abandon();
    Response.Write("След Abandon() var1="+Session["var1"]);
  }else{
    Response.Write("След повторно извикване var1="+Session["var1"]);
  }
}
```

Пример 13 Използуване на метода Abandon на обекта Session за унищожаване на всички сесийни променливи.

8.5. Унищожаване на сесийна променлива -метод Remove

Методът Remove премахва именуван обект от колекцията на обект Session.

Декларация (C#)

public void Remove(string name)

Например ако искаме да унищожим сесийна променлива с име “var1” можем да използваме следната инструкция

Session.Remove(“var1”);

8.6. Колекция Contents (обект Session).

Съдържа всички променливи, създадени по време на сесията, Цялото съдържание на колекцията може да се изведе чрез цикъл foreach, както е показано в дадения по-долу пример:

```
protected void Button1_Click(object sender, EventArgs e)
{
 string s = "<table border>"; //Начален етикет на таблицата
 foreach (string key in Session.Contents)
 { //Добавяне на ред с име и стойност на сесийна променлива
     s += "<tr><td>" + key + "</td><td>" + Session[key] + "</td></tr>";
 }
 s += "</table>";
```

```
  Response.Write(s); //Извеждане на таблицата
}
```

Пример 14 Извеждане на всички променливи на сесията в HTML таблица посредством цикъл **foreach**

Колекцията на обект Session (както и на обект Application) е от типа NameObjectCollection, което означава, че елементите имат ключ от тип string и стойност от базовия тип Object, следователно в сесийните променливи може да се записват стойности от всеки тип данни.

Контролни въпроси 2

a) Как може да програмно да се предизвика унищожаването на всички сесийни променливи.
b) Защо не се налага да се използва заключване при работа със сесийни променливи.

9. Събитиен модел на ASP.NET

Изпълнението на ASP.NET приложенията се управлява от събития, които се генерират както от потребителя, така и от WEB браузъра и WEB сървъра по време на изпълнението на заявките за .aspx страници от приложението..NET Framework предоставя набор от типове и класове, които дават възможност на програмистите програмно да генерират събития и да създават процедури, които да се изпълняват при определени събития. Ще разгледаме накратко основните от типове и класове, свързани със събитийния модел на.NET Framework и ASP.NET и използването им в C#.

9.1. Делегати в C#

Делегатите са дефинирани като нов тип данни в C# - типово безопасен функционален указател. Общото между функционалните указатели в C и делегатите в C# е в тяхното предназначение – да служат за извикване на функции със зададен тип на връщания резултат и със зададен набор от параметри. Най-същественото различие между функционалните указатели на C и делегатите на C# е това, че функционалният указател е просто променлива, която съдържа адрес на

функция, а делегатът е обект, които предоставя атрибут за извикване на функция.

9.1.1. Деклариране на тип делегат

Декларацията на тип делегат дефинира клас, който се наследява от класа System.Delegate. В C# декларацията има следния синтаксис:

```
public delegate type_of_result name_of_delegate(list_of_parameters);
```

където:
type_of_result е типът на връщания от типа делегат резултат
name_of_delegate е името на декларирания тип делегат
list_of_parameters е списък от формални параметри на типа делегат

Важно е да се запомни, че типът делегат се създава за методи с точно зададен списък от параметри (със зададен тип за всеки параметър от списъка) и тип на връщания резултат.
Пример:

```
public delegate void MyDelegateType(string event_message);
```

Пример 15 Деклариране на тип делегат

В примера е деклaран тип делегат за събитийна процедура от тип void с един параметър от тип string.

9.1.2. Деклариране на променлива от типа делегат

Променливата от типа делегат се декларира в съответствие със синтаксиса на C# тип_на_променливата име_на_променливата

```
MyDelegateType MyDelegate;
```

Пример 16 Деклариране на променлива от типа делегат MyDelegateType

9.1.3. Създаване на обект от типа делегат с присвоен метод

За да използваме създадения тип делегат трябва да създадем обект от класа, който той представя. Конструкторът на класа трябва да получи като параметър името на метода, който ще се извиква с делегата.

Конструктовът връща обект от декларирания тип делегат, чрез който метода може да бъде извикван.

Тъй като класът MyDelegateType е динамично създаден, той не може да се използва за инициализиране на атрибути (полета) на класа, в който е създаден. Обектът от клас MyDelegateType трябва да бъде присвоен на локална променлива в метод на текущия клас, както е показано в дадения по-долу пример:

```
public void Page_Load(object sender, EventArgs e)
{
   MyDelegateType MyDelegate;
   MyDelegate = new MyDelegateType (MyMethod);
}
```

Пример 17 Създаване на обект делегат от клас **MyDelegateType**

В примера се създава делегат като обект MyDelegate от клас MyDelegateType като на конструктора на класа се предава параметър името на метод MyMethod, който да бъде извикан при генерирането на събитие от съответния тип. За да може да се абонира обектът-получател за уведомяване при възникване на събитие, обектът източник трябва да притежава съответен член от тип event за генериране на събития.

9.2. Събитията и типът event

Ключовата дума event (събитие) дава възможност за деклариране на специален тип членове на класа в NET Framework. Членовете от тип event позволяват на обектите от класа да уведомяват други обекти са настъпване на определени събития. На практика чрез метод на обект event се извикват събитийни процедури (по правило от други обекти) които са абонирани за събитието. Важно е да се запомни, че само клас, в който е деклариран член от тип event може да генерира събитие. C# предоставя два вида декларации на членове от тип event:

Декларация в стил поле на клас:

```
public event TheDelegateType TheEventName;
```

където:

TheDelegateType е тип делегат, който ще се използва се извикване на методи при генерирането на събитие

TheEventName е името на члена event

Декларация в стил атрибут на клас:

```
public event TheEventHandler MyEvent
{
  add { код за добавяне на делегат към списъка от манипулатори на
събитието}
  remove { код за премахване на делегат от списъка от манипулатори }
}
```

Този синтаксис дава по големи възможности за контрол на добавянето и премахването на "абонати" за събитието., но е по сложен за използване и освен това по подразбиране не е безопасен при използване на обекта в многонишково приложение. Поради това за ASP.NET приложения е по-добре да се използва първия синтаксис за деклариране на член от тип event, както е в дадения по-долу пример:

```
    public event MyDelegateType MyEvent;
```

Пример 18 Деклариране в клас-източник на събитие на член от тип event

В примера в класа-източник на събитие се декларира член от тип event с име MyEvent за делегати (т. е. функционални указатели) от клас MyDelegateType.

9.3. Абониране за уведомяване при възникване на събитие

Събитийния модел на.NET Framework често се онагледява като взаимодействие *издател-абонати* или *източник-приемници*. Издателят (източникът) генерира събитие, а абонатите (приемниците) го получават и изпълняват съответните методи (събитийни процедури). От тази схема с претенции за нагледност може да се направи погрешният извод, че генерирането на събитие е свързано с изпращане на някакъв вид съобщение, подобно на това, което например ОС Windows записва в опашката от съобщения на активния прозорец при натискане на бутон от клавиатурата. Всъщност нещата са доста по прости. В члена от тип event на обекта-източник на събитие, предварително трябва да се запишат функционалните указатели на всички методи, които трябва да се извикат при генериране на събитието. Тези типово безопасни функционални указатели са описаните по-горе делегати. Записът на

делегат в член от тип event става с оператора “+=” както е показано с дадения по-долу пример:

```
this.MyEvent += MyDelegate;
```

Пример 19 Добавяне на делегат към член от тип event

В примера към члена MyEvent от тип event се добавя делегатът MyDelegate. В случая MyEvent е член на текущия клас, но по правило тази инструкция се изпълнява в кода на класа-приемник на събитието, като в тоя случай вместо променливата this трябва да се използва обектна пременлива (референция) към класа-източник на събитието. Ако програмистът иска да има контрол върху стойността на аргумента в тази операция, е по-добре MyEvent да не се декларира с модификатор public (т.е. да не е видим извън класа) а записът в него да става чрез метод на класа, подобен на set методите за атрибути.

9.4. Генериране на събитие чрез обект event

Генерирането на събитие е всъщност изпълнение на инструкция, с която на члена от тип event се предават фактически патраметри за методите и той извиква тези методи като използва записаните в него функционални указатели. Синтаксисът на инструкцията за генериране на събитие е следния:

[object_variable.]member_of_type_event(list of parameters);

където:

object_variable е обектна променлива за обекта – източник на събитие

member_of_type_event е членът от тип event, който генерира събитието

list of parameters е списъкът от параметри, с които се извикват методите, абонирани за събитието (т.е. за които в списъка има асоциирани делегати

Пример

```
public void FireEvent(string event_message)
{
    this.MyEvent(event_message);
}
```

Пример 20 Генериране на събитие **MyEvent**

В примера е деклариран метод, който получава параметър от тип string и генерира събитие MyEvent, като му предава този параметър. В резултат от изпълнението на инструкцията this.MyEvent(event_message); ще се извикат за изпълнение методите, за които са асоциирани делегати в члена от тип evemt. Всички извикани за изпълнение методи ще получат параметъра event_message.

Да разгледаме един цялостен пример за ASP.NET модул, който декларира тип делегат и събитие, създава обект делегат , асоциира го със събитието и генерира събитието, в резултат на което се извиква за изпълнение метод на класа.

```
public partial class _Default : System.Web.UI.Page
{
  // Деклариране на тип делегат
   public delegate void MyDelegateType(string event_message);
   // Деклариране на член от тип event (събитие) за типа делегат
   public event MyDelegateType MyEvent;

   protected void Page_Load(object sender, EventArgs e)
   {
       //Деклариране на променлива от типа делегат
       MyDelegateType MyDelegate
       //Създаване на обект делегат от тип (клас) MyDelegateType
       MyDelegateType MyDelegate = new MyDelegateType(MyMethod);
       //Асоцииране на делегата MyDelegate със събитието MyEvent
       this.MyEvent += MyDelegate;
       // Генериране на събитието MyEvent
       this.MyEvent("Event has been fired");
   }
   // Метод, който се извиква чрез делегата MyDelegate
   // при генериране на събитие MyEvent
   protected void MyMethod(string event_message)
   {
      Response.Write("<BR>" + event_message); //Извеждане на
съобщение
   }
}
```

Пример 21 Код на ASP.NET модул, който генерира събитие

Примерът съдържа декларацията на класа _Default, наследник на класа Page. В декларацията на класа са включени декларация на тип делегат MyDelegateType, декларация на член на класа MyEvent от тип event за този делегат и декларацията на методите Page_Load и MyMethod.

В събитийната процедура Page_Load се създава обект делегат MyDelegate от клас MyDelegateType, който е функционален указател към метода MyMethod. Този делегат се записва в члена MyEvent от тип event, след което се генерира събитието MyEvent с инструкцията *this.MyEvent("Event has been fired");* Като резултат чрез записания в MyEvent делегат се извиква метода MyMethod, който извежда съобщението "Event has been fired", получено като стойност на параметъра event_message от MyEvent.

Примерът е умишлено опростен за да демонстрира по-разбираемо операциите, свързани с използването на делегати и събития в събитийния модел на.NET Framework и в частност на ASP.NET. Ако необходимо изпълнението на метода MyMethod от текущия клас, най-просто е да извършим непосредствено обръщение към него с инструкцията MyMethod("Event has been fired"). По правило събитията са средство за изпълнение на външна за даден клас (и обект от него) процедура. Така обектът капсулира определена функционалност, но дава възможност на програмиста, който го използва да включи свой код във външна процедура (т.н. събитийна процедура), която се изпълнява по инициатива на обекта, тогава, когато обектът генерира съответното събитие. Това е идеята за обратно извикване (callback), включена в езика C и развита в под различна форма в други програмни езици.

9.5. Видове събития в ASP.NET

ASP.NET технологията предоставя на разработчиците на интернет приложения възможност за изпълнение на събитийни процедури от страна на сървъра по модел, подобен на модела на събитията, които се изпълняват от страна на клиента. Ако използваме Visual Studio.NET за създаване на ASP.NET Web сайт и добавим бутон от групата Standard и след това щракнем двукратно с мишката върху него, ще попаднем в кодовата страница на ASP.NET модула и ще видим, че курсора на мишката е установен в събитийна процедура Button1_Click. По същия начин създаваме събитийна процедура за бутон и когато създаваме с

Visual Studio.NET проект Windows application. Така, че при използване на Visual Studio.NET и помощната програма (дизайнер) за ASP.NET приложения, събитийни процедури се създават както за локални Windows приложения.

Важно е обаче да се има предвид съществената разлика между събитията, които се генерират и изпълняват в клиентските приложения и тези,. които се генерират от сървърните контроли в .aspx страница и се изпълняват от страна на сървъра. В ASP.NET приложенията, когато се генерира събитие от страна на клиента, а трябва да се изпълни събитийна процедура от страна на сървъра, е необходимо да се използва механизъм за предаване на информация за събитията. Разработчиците на Microsoft не са имали много възможности за да създадат този механизъм. Известните начини за предаване на данни от HTML документ, изведен в прозорец на WEB браузъра, към модул от интернет приложение са два:

1. Чрез т.н. бисквидки – текстова информация, която се обменя между браузъра и WEB сървъра и се съхранява на потребителския компютър. На този механизъм за обмен на информация обаче не може да се разчита, тъй като потребителят може да забрани записа на бисквидки чрез функция на браузъра

2. Чрез скрит (от тип hidder) елемент в HTML формуляр в кода на .aspx страницата. Разработчиците на Microsoft са избрали този механизъм, тъй като той не притежава недостатъците на бисквидките и при използване на метода POST за изпращане на данни от HTML формуляра (както е във въведените от Microsoft WEB фóрми) е невидим за потребителите.

И така, събитийният модел на ASP.NET включва два вида събития:

1. Събития, които се генерират от страна на клиента и се обработват от страна на сървъра. ASP.NET предоставя два основни вида събития от този тип – Click и Changed, които се генерират съответно при щракване с мишката и при променяне на въведените или селектирани данни в контрола. Различните видове сървърни контроли поддържат различни събития. Например контролът Button поддържа събитие Click, но не поддържа събитие Changed, а контролът TextBox поддържа събитие Changed а не поддържа Click.
2. Събития, които се генерират и обработват от страна на сървъра. Жизненият цикъл на ASP.NET предоставя набор от такива

събития, , които включват събития, генерирани от обекта от клас Page, който представя .aspx страницата, и събития, генерирани от сървърните контроли. Класът Page и сървърните контроли са наследници на класа Component, и поради това основните събития, които генерира обектът Page, като например Init, Load, и PreRender, се генерират и от сървърните контроли. Различните видове сървърни контроли обаче могат да генерират и специфични събития, които класът Page на притежава.

Освен това, тъй като .aspx страницата, която се получава и извежда от WEB браузъра е HTML документ, то .aspx страницата може да използва и събития, които се генерират и обработват от страна на клиента, което на практика става с код на JavaScript, включен в .aspx страницата. Събитийният модел на HTML документите е предмет на курса по WEB дизайн и няма да се разглежда в този курс.

10. Жизнен цикъл на ASP.NET модулите

Кодът на ASP.NET модулите се изпълнява от WEB сървъра. При всяко извикване на съответната .aspx страница, при изпълнението на кода на съответния модул, се генерира HTML код на страницата и се изпраща на потребителския браузър. Жизненият цикъл на ASP.NET модула започва при заявяването на неговата .aspx страница и завършва тогава, когато .aspx страницата се изпрати като HTML документ към потребителския браузър. Всяко повторно заявяване на .aspx страницата (например при щракване с мишката върху бутон от WEB формуляр в страницата) предизвиква ново изпълнение на кода на ASP.NET модула, при което отново се генерира и изпраща към браузъра HTML страница. При това се генерират последователно следните събития

PreInit	Начало на инициализация
Init	Инициализацият
InitComplete	Край на инициализация
PreLoad	Начало на зареждане на обектите
Load	Извикване на методите OnLoad на обектите
Control events	събития Changed и Click на контролите
LoadComplete	Край на зареждане на обектите
PreRender	Начало на генериране на HTML код на страницата

SaveStateComplete	Запис на състоянието на контролите във ViewState
фаза Render	Извикване на методите Render на обектите
Unload	Край на изпращането на HTML кода

Важно е да се запомни, че при заявяване на .aspx страница потребителят получава чрез браузъра си HTML документ, който съдържа само HTML код и код на JavaScript, който се изпълнява от WEB браузъра. Част от кода на получения HTML документ е динамично генериран в резултат на изпълнение на кода на ASP.NET модула, и поради това видът и съдържанието на страницата могат да бъдат различни при всяко следващо заявяване.

Жизненият цикъл на ASP.NET модула включва няколко основни събития. ASP.NET технологията предоставя на програмистите възможност да създават собствени събитийни процедури за тези събития, и това ги прави полезен инструмент за разработчиците на интернет приложения. По долу са разгледани тези събития в последователността, в която те се генерират по време на жизнения цикъл на ASP.NET модула.

10.1. Инициализиране на ASP.NET модула. Сибития PreInit, Init и InitComplete

При извикване на .aspx страницата се генерират последователно събитията PreInit, Init и InitComplete на обекта от клас Page, който представя ASP.NET модула, и се изпълняват съответните събитийни процедури Page_PreInit Page_Init. и Page_InitComplete, ако са включени в кода на ASP.NET модула. Това са първите събития от страна на сървъра, за което програмистът може да изпълни своя събитийна процедура. Тези процедури могат да се използват за инициализаиране на променливи, но трябва да се има в предвид разликата между тях:

Page_PreInit	Достъпен е атрибута IsPostBack, може динамично да се създават контроли. Не са установени стойностите на атрибути на контроли, които са променени от потребителя при повторно извикване на страницата.
Page_Init	Всички контроли са инициализирани. Могат да бъдат променяни стойностите на техни атрибути.
Page_InitComplete	Всички инициализации на обекти и контроли са завършени.

10.2. Зареждане на ASP.NET модула Събития PreLoad, Load и LoadComplete

Събитията PreLoad, Load и LoadComplete се генерират всеки път, когато се извиква.aspx страницата, и се изпълняват събитийните процедури Page_PreLoad, Page_Load и Page_LoadComplete ако са включени в кода на ASP.NET модула. Събитията Load следват след събитията Init на ASP.NET модула и на сървърните контроли. Първо се генерира събитието PreLoad. Извикват се методите OnLoad и се генерират събития Load на обект Page и сървърните контроли. Тази последователност от събития завършва със събитието LoadComplete на обект Page.

Събитийната процедура Page_Load дава възможност за изпълнение на операции преди да се изпълнят събитйните процедури от тип Changed и Click. Преди нейното изпълнение, при повторно извикване, състоянието на сървърните контроли е вече актуализирано в съответствие с получените данни от WEB формуляра на .aspx страницата.

10.3. Събитийни процедури от тип Changed

Събитията от тип Changed се генерират при променяне на състоянието на сървърните контроли. Например след въвеждане на текст в текстова кутия TextBox1 се генерира събитието TextChanged и се изпълнява събитийна процедура TextBox1_TextChanged, при променяне на селекцията в списъчен обект ListBox1 се генерира събитието SelectedIndexChanged и се изпълнява събитийна процедура ListBox1_SelectedIndexChanged. Информацията за променяне на на състоянието на сървърните контроли се изпраща към ASP.NET модула тогава, когато се извиква повторно .aspx страноцата. Тогава се изпращат данни и информация за променяне на състоянието на контролите от WEB формуляра на страницата. Като резултат след събитийната процедура Page_Load се изпълняват (не по реда на променянето на контролите) събитийните процедури от тип Changed. По-подробно този въпрос е разгледан в главата “Сървърни контроли”.

10.4. Събитийни процедури от тип Click

Събитията от тип Click се генерират при щракване с мишката върху бутон от WEB формуляр. Например при щракване с мишката върху WEB сървърния контрол Button1 се генерира събитие Click и в ASP.NET модула се изпълнява събитийната процедура Button1_Click По подразбиране именно събитието Click предизвиква изпращане на данните от WEB формуляра на .aspx страницата към ASP.NET модула на сървъра. Събитийната процедура за събитие от тип Click се изпълнява след събитийните процедури Changed.

10.5. Начало на цикъла за извеждане на сървърните контроли. Сибитие PreRender

Събитието PreRender се генерира след изпълнението на методите DataBind на свързаните към източник на данни контроли, които имат зададена валидна стойност на атрибута DataSourceId. Събитието извиква изпълнението на събитийната процедура Page_PreRender. Както показва и името на събитието, след изпълнението му се изпълняват методите Render на обект Page и сървърните контроли в .aspx страницата. Преди това обаче се генерира събитието SaveStateComplete.

10.6. Запис на състоянието на сървърните контроли. Сибитие SaveStateComplete

Събитието SaveStateComplete се генерира след запис на състоянието на .aspx страницата и контролите в техните атрибути ViewState. Три изпращане на .aspx страницата като HTML документ към потребителския браузър стойностите на тези атрибути се изпращат като стойност на скрит (от тип hidden) HTML елемент na WEB формуляра. Измененията в сървърните контроли след генериране на събитието SaveStateComplete се игнорират.

10.7. Премахване от паметта на генерираната .aspx страница. Сибитие Unload

Събитието Unload се генерира всеки път, когато се приключва обработката на поредната заявка към .aspx страницата, като при това се изпълнява събитийната процедура Page_Unload на ASP.NET модула. Събитието Unload се генерира след като HTML кодът на .aspx страницата е получен, изпратен към потребителския браузър и може да бъде унищожен от страна на сървъра. В събитийната процедура Page_Unload вече са недостъпни атрибутите на обекта Page (и следователно и обекти като Request и Response).

Важно е да се отбележи, че събитието Unload се генерира от страна на сървъра и не е свързано по никакъв начин със затварянето на прозореца на браузъра, в който се визуализира .aspx страницата. Ако разработчикът иска да изпълни някакви операции при затваряне на прозореца на браузъра или зареждане на друга страница в него, той трябва да използва манипулатора onunload на обект window на JavaScript и код на JavaScript, който се изпълнява от браузъра (т.е, от страна на клиента, а не от страна на сървъра).

При използването на събитийните процедури от жизнения цикъл на .aspx страницата трябва да се има предвид, че те се свързват автоматично със събитията с дадените по-горе имена (по конвенцията име-на-обект_ime-на-събитие) само тогава, когато атрибутът AutoEventWireup na обекта от клас Page на ASP.NET модула има стойност true. По правило стойността на този атрибут се задава в директивата @Page в .aspx файла на страницата (вижте описанието на директивата в т. 6).

11. Сървърни контроли

Сървърните контроли са обекти на ASP.NET модулите, които предоставят функционалност, която включва код, изпълняван от страна на сървъра и графичен интерфейс, който дава възможност за интерактивно взаимодействие с потребителя. Ако търсим аналог на сървърните контроли в ASP технологията, можем да го открием в COM обектите, които могат да се включат във файла Global.asa чрез етикет <object> и включват атрибут runat="server". За разлика от COM обектите обаче, Web контролите изпращат към потребителския браузър HTML код и така се визуализират в .aspx страниците, т.е. те (в ASP.NET приложението) притежават графичен интерфейс.

Може да се направи следната класификация на Web контролите

- HTML сървърни контроли – това на практика са HTML контроли с атрибут runat="server", което дава възможност за изпълнение на събитийни процедури от страна на сървъра.
- Web сървърни контроли – това са нови контроли, които притежават разширена функционалност от страна на сървъра.
- Валидизиращи контроли – това са Web сървърни контроли, които дават възможност за проверка на валидността на въведени от потребителя данни
- Свързани контроли – това са Web сървърни контроли, които се свързват към източник и дават възможност за извеждане и актуализиране на данни.

По-долу ще бъдат разгледани по-подробно видовете Web контроли

12. Web сървърни контроли

Web сървърните контроли са дефинирани в именното пространство System.Web.UI. WebControls и се съдържат в компилиран вид във файла system.Web.dll. Не е необходимо в .aspx страниците да се включва с директива @Import това именовано пространство. WEB сървърните контроли се създават чрез именното пространство asp:

Web сървърните (intrinsic) ASP.NET контроли се използуват като алтернатива на HTML контролите. Най-съществената разлика в синтаксиса им в сравнение с HTML контролите е атрибута asp: , който се включва в началото на етикета на контрола преди името му.

Например за да се добави етикетът Label към WEB форма, в нея трябва да се включи следния код:

```
<asp:Label runat="server" Text="Label1"></asp:Label>
```

Ако включим този код в HTML документ и го изведем във Web браузър, ще видим, че от етикета Label няма да се изведе нищо. Това е така, защото това не е HTML етикет. Ако обаче включим този код в ASP.NET страница, при нейното извикване от Web сървъра контролът Label ще изпрати към браузъра следния HTML код

```
<span>Label1</span>
```

Всеки сървърен контрол притежава метод Render(), който се изпълнява когато се обработва ASP.NET страницата от сървъра. Методът Render() записва HTML кода, с който се визуализира контрола, в изходния HTML поток, който се изпраща към потребителския браузър. В случая сървърния контрол Label добавя към изходния HTML поток низа .” Label1”.

12.1. Предимства на Web сървърните контроли:

- Използуват унифицирана конвенция за именуване.
- Предоставят набор от общи атрибути за именуване за всички контроли.
- Предоставят уникални имена на атрибути, специфични за контролите
- Създават специфичен за браузъра HTML код

12.1.1. Унифицирана конвенция за именуване.

Един от недостатъците на HTML контролите е отсъствието на правило (конвенция) за именуване, което да съответствува на вида на контрола. Например контролите

```
<input type="radio">
<input type="checkbox">
<input type="submit">
```

имат един и същ етикет (<input>) но са с доста различна функционалност. Web сървърните ASP.NET контроли предоставят алтернативен начин за деклариране, в който след префикса asp следва уникален тип на контрола. Следните редове код декларират ASP.NET контроли за радиобутон, селектор и бутон за изпращане.

```
<asp:RadioButon runat=”server”>
<asp:CheckBox runat=”server”>
<asp:Button runat=”server”>
```

12.1.2. Общи атрибути за именуване

В ASP.NET общите (с едно и също значение) атрибути на контролите имат едни и същи имена. Например ако искате да зададете фонов цвят на контрол ще използвате един и същ атрибут независимо от вида на контрола. Общите атрибути на сървърните контроли се наследяват от класа WebControl, например атрибутите Text, BackColor и ForeColor. В примера, даден по-долу е показан код на два сървърни контрола – TextBox и Button, от който се вижда, че всички използвани атрибути имат едно и също наименование.

```
<asp:TextBox id=”t1” Text=”Peter” BackColor=”red” runat=”server”/>
<asp:Button id=”btnSubmit” Text=”Send” BackColor=”cyan”
runat=”server”/>
```

12.1.3. Атрибути, специфични за контролите

Web сървърните (intrinsic) контроли притежават и атрибути, които са специфични за определени контроли. Например контрола CheckBox има атрибут Checked, с който се указва, дали кутията за селекция е селектирана:

```
<asp:CheckBox Checked=”True” Text=”I agree” runat=”server”/>
```

ASP.NET контрол	HTML контрол
<asp:TextBox … />	<input type=”text” … >
<asp:Button … />	<input type=”Submit” … >
<asp:CheckBox … />	<input type=” CheckBox” … >
<asp:image />	<img src=”…” >
<asp:hiperlik> …</asp:hiperlik>	
<asp:panel> …</asp:panel>	…

<asp:label> … </asp:label>	…
<asp:dropdownlist runat=”server> <asp:listitem> … </asp: listitem> <asp:listitem> … </asp: listitem> </asp:dropdownlist>	<select> <option> … </option> <option> … </option> </select>

Таблица 1 Съответствия между HTML и сървърните ASP.NET контроли

12.2. Обработване на събития от сървърните контроли

Web сървърните контроли поддържат само събития от страна на сървъра. Изключение прави само контрола Button, който притежава манипулатор OnClientClick, с който, при щракване върху бутона, може да се изпълни код на JavaScript в потребителския браузър. Те имат ограничен набор от събития, които генерират, като всяко от тези събития предизвиква предаване на информация от клиента към сървъра. Сървърните контроли поддържат събития “Click”. Контролите, които приемат въвеждане от клавиатурата поддържат събития “Changed”, които индицират променяне на въведения в контрола текст. Някои сървърни контроли поддържат специални версии на събития “Changed” , например “SelectionChanged” , които се генерират когато промени селектирания елемент в списъчен контрол. . Събития, свързани с движение на курсора на мишката не се поддържат.

12.3. Използуване на събитийни процедури

За използуване на събитийни процедури е необходимо да се включи манипулатор на събитието в етикета на контрола, а след това да се добави кода на събитийната процедура.
Пример:

<asp:button id=”btn” runat=”server” onclick=”btn_click” />

Web контролът button има само манипулатор на събитие “onclick” и няма “onServerClick”

Събитийната процедура на бутона има вида:

```
void btn_click(object sender, EventArgs e)
{
  // код на събитийната процедура
}
```

Web контролите (с изключение на контрола Button) поддържат събития само от страна на сървъра , докато HTML контролите, които имат атрибут runat=”server” поддържат събития както от страна на клиента, така и от страна на сървъра.

Това твърдение, което може да бъде прочетено в документацията на Microsoft, определено се нуждае от доуточняване. Както е показано в таблица 1, всички Web контроли се изобразяват във Web браузъра като HTML контроли, а HTML контролите притежават манипулатори за събития от страна на клиента. Например ако .aspx страницата съдържа контрола TextBox той ще изпрати към Web браузъра код на HTML контрол – текстова кутия. Ако разработчикът добави в етикета на контрола манипулатори на събития, те ще се изпратят към Web браузъра в етикета на текстовата кутия. В дадения по-долу пример в кода на контрола TextBox1 са включени манипулаториOnFocus и OnBlur на събитията focus и blur (получаване на фокус и загуба на фокус) на текстовата кутия.

```
<asp:TextBox ID="TextBox1" runat="server"  OnFocus=
"this.style.backgroundColor='#FFFFA0' "
OnBlur="this.style.backgroundColor='white' "></asp:TextBox>
```

Пример 22 Код на контрол TextBox, който съдържа манипулатори за събития, които се изпълняват от Web браузъра

В примера чрез манипулаторите OnFocus и OnBlur се променя цвета на фона на текстовата кутия. Така при щракване с мишката в текстовата кутия тя ще се изобрази със светложълт фонов цвят, а при щракване извън нея - с бял фонов цвят. Обектната променлива this в примера представя JavaScript обекта , който генерира събитието.

```
<input name="TextBox1" type="text" id="TextBox1" OnFocus=
"this.style.backgroundColor='#FFFFA0' "
OnBlur="this.style.backgroundColor='white' " />
```

Пример 23 HTML код, с който контролът TextBox1 се изобразява във Web браузъра

12.4. Базовият клас на Web сървърните контроли – класът WebControl

Класът WebControl е базов клас на всички класове за контроли в именното пространство System.Web.UI.WebControls. Неговите атрибути, методи и събития се притежават от всички контроли, декларирани в това пространство.

Декларация (C#)

```
public class WebControl : Control, IAttributeAccessor
```

12.4.1. Атрибути на класа WebControl

Атрибути за достъп до контрола

Програмният достъп до контролите в .NET приложенията се извършва чрез техните идентификатори (тип string). Идентификаторът ID се задава от разработчика, а идентификаторите ClientID и UniqueID се генерират автоматично от .NET Framework.

ID	Задава и връща идентификатора на сървърния контрол.
ClientID	Задава и връща идентификатора на сървърния контрол, генериран от .NET Framework. Ако разработчикът е задал стойност на ID, стойността на ClientID съвпада с ID.
UniqueID	Връща уникален идентификатора на сървърния контрол, генериран от .NET Framework. Той се различава от ID тогава, когато контролът е включен като дъщерен в комплексен контрол и се използват няколко негови екземпляра (инстанции).

Атрибути за форматиране

BackColor	Задава и връща фонов цвят на сървърния контрол.
BorderColor	Задава и връща цвят на рамката.

BorderStyle	Задава и връща цвят на рамката на контрола.
BorderWidth	Задава и връща дебелина на линията за рамката на контрола..
Font	Задава и връща обект Font, който задава параметрите на текста, изобразяван в контрола.
ForeColor	Задава и връща цвета на текста на контрола.
Style	Задава и връща колекция от текстови атрибути, които задават стила (CSS) с който се изобразява контрола в прозореца на браузъра.
CssClass	Задава и връща CSS клас, с който се форматира контрола при изобразяването му в прозореца на браузъра.
Width	Задава и връща дължината на контрола.
Height	Задава и връща височината на контрола.

Атрибути, които връщат референции към свързани обекти

Атрибутите Page, Parent, NamingContainer, Controls, Site имат същото значение, както съответните атрибути на обект Page. Атрибутът NamingContainer е разгледан по-подробно в т. "Разширяване на WEB контроли".

12.4.2. Методи на класа WebControl

Класът WebControl притежава над 20 публични* метода, повечето от които са наследени от класа Control. Част от тези методи се използват от разработчиците на WEB контроли, а други са от значение само за определен тип контроли, напр. методът DataBind за свързаните към източник на данни контроли. Ще разгледаме само четири метода, които са най-често се използвани във WEB приложенията.

Получаване информация за типа на контрола – метод **GetType**

Методът връща обект от клас System.Type, който представя типа на контрола.

Декларация: (C#)

```
public Type GetType()
```

* т.е. с модификатор public – достъпни извън кода на контрола

В дадения по-долу пример метода GetType се използва за да се изведе в HTML таблица типа на контролите, съдържащи се във Web форма.

```
string s = "<TABLE BORDER>";
// Итерация с цикъл foreach на контролите във Web форма
foreach (Control c in this.form1.Controls)
{
   s += "<TR><TD>Type="+c.GetType().ToString()+ "</TD><TD>ID="+
c.ID + "</TD></TR>";
}
s+="</TABLE>";
Label1.Text=s; // Извеждане на таблицата
```

Пример 24 Извеждане на идентификатора и типа на контролите във Web форма

Установяване на фокус върху контрола – метод Focus

Методът установява фокуса за въвеждане върху контрола в прозореца на браузъра. Това означава на практика, че контролът приема данните, въведени от клавиатурата.

Декларация: (C#)

```
public virtual void Focus ()
```

Проверка за наличие на дъщерни контроли – метод HasControls

Методът връща логическа стойност, която показва, дали контролът е комплексен, т.е. контейнер на дъщерни контроли. Пример за конплексни контроли са свързаните контроли, които извеждат в обекти Label или TextBox полета от записите, получени от източник на данни.

Декларация: (C#)

```
public virtual bool HasControls()
```

Получаване на референция към контрол – метод FindControl

Методът търси дъщерен контрол в именния контейнер (NamingContainer) по зададен идентифокатор. Може да се използва ако контролът е контейнер на дъщерни контроли, за които създава уникално именно пространство. Такъв е например свързаният контрол GridView, който в режим Edit извежда като дъщерни контроли текстови кутии със съдържанието на полетата от източника на данни.

Декларация: (C#)

protected virtual Control FindControl (string id)

където id е идентификатор на търсения контрол

12.5. Списъчни контроли ListBox, RadioButtonList, DropDownList

WEB сървърните списъчни контроли дават възможност на потребителя да избере един или няколко елемента от зададен списък. Най-често използваният списъчен контрол е контролът ListBox. Той се различава както по външен вид, така и по възможности от контрола DropDownList по това, че може да извежда много елементи наведнъж и дава възможност (като опция) да се селектира повече от един елемент.

Размерите на контрола ListBox може да се контролират по два различни начина:

a) Чрез задаване на броя на извежданите редове с атрибута Rows
b) Чрез задаване на размерите на контрола с атрибутите Width и Height в брой пиксели или в проценти. Ако се зададе стойност на Height контролът игнорира зададения брой редове и се извежда със зададения ветрикален размер.

Размерите на контрола RadioButtonList може да се контролират с атрибутите Width и Height в брой пиксели или в проценти, но минималните размери се определят от съдържанието на контрола.

12.5.1. Атрибут Items – колекция от елементите на списъчния обект

Атрибутът връща колекция, която съдържа елементите в списъчния обект ListBox.

Декларация (C#):

public ObjectCollection Items { get; }

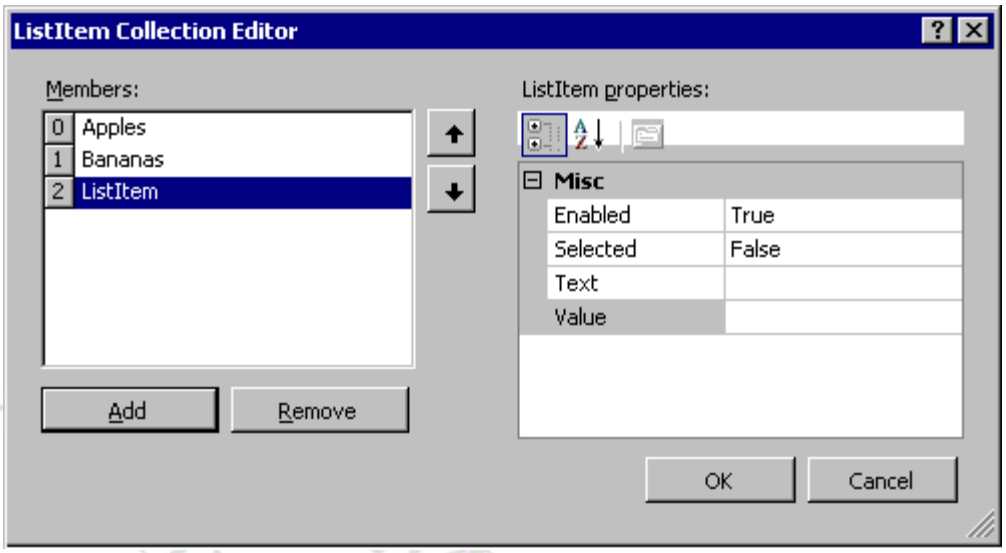
Колекцията Items предоставя възможност за операции с елементите на списъчния обект:

Добавяне на елемент посредством метода Add на обект Item.

```
ListBox1.Items.Add(new ListItem("Hello"));
```

Добавяне на елемент към списъчен контрол посредством метод Add на обект Item

Visual Studio.Net (VS.NET) предоставя в режим Design помощно средство ListItem Collection Editor за въвеждане и редактиране на елементите на списъчните обекти:



Фиг. 2 Графичен редактор на VS.NET за елементи на списъчните обекти

12.5.2. Обект Item – елемент от списъчен обект

Елементите на списъчните обекти се представят от обекти Item, достъпът до които се осъществява чрез колекцията Items. В графичния редактор за елементите Item, показан по-горе, се виждат атрибутите, които предоставя обект Item:

Text (тип string) Задава и връща текста, извеждан от елемента Item.

Value (тип string) Задава и връща стойността, асоциирана с елемента. Атрибутът Value дава възможност при селектиране на елемента да се изпраща стойност, различна от текста, извеждан от елемента. Например елементът Item с код

```
<asp:ListItem Value="1">First Item</asp:ListItem>
```

ще извежда текст “First Item” , а ше изпраща стойност “1”.

Selected (тип Boolean) Задава и връща стойност, която показва дали при начално извеждане на списъчния контрол елементът ще бъде селекторан или не. В контрола DropDownList, който извежда падащ списък, само един от елементите може да има стойност True на атрибута Selected, в противен случай се извежда съобщение за грешка при компилирането на .aspx страницата.

Enabled (тип Boolean) Задава и връща стойност, която показва дали елементът ще се извежда в списъчния контрол.

12.6. Използуване на атрибута AutoPostBack на Web контролите.

Във Web контролите по подразбиране само Click събитията предизвикват изпращане на данни от HTML форма към сървъра. Събитията “Change” се прихващат, но не предизвикват незабавно изпращане на данни към сървъра. Вместо това те се съхраняват от контрола в кеш-памет до изпращането на данни към сървъра.
След изпращането на данни от HTML форма към сървъра, от събитийните процедури първи се изпълняват процедурите за събитията “Change” в не-последователен на възникването им ред.След като се изпълнят всички събитийни процедури за възникнали събития “Change” се изпълнява и събитийната процедура за събитието “Click”, което е предизвикало изпращането на данните.

Програмистът може да разреши изпращането на данни от Web формата към сървъра при възникване на събитие “Change” (и така да предизвика незабавното му отработване) със задаване на стойност *true* на атрибута “AutoPostBack” за съответния ASP.NET контрол.

```
<script language="C#" runat="server">
protected void Page_Load(Object Src, EventArgs E){
        if(!Page.IsPostBack){
  System.Collections.ArrayList fruits =new
  System.Collections.ArrayList();
  fruits.Add("Apples");
  fruits.Add("Oranges");
  fruits.Add("Bananas");
  fruits.TrimToSize();
  rb.DataSource= fruits;
  rb.DataBind(); //добавяне на записи от източника
        }
  }
void displayMessage(Object Src,EventArgs E){
        lbl1.Text="Вашият любим плод е: " +
        rb.SelectedItem.Text;
}
```

```
</script>
<html>
<body>
<form runat="server">
<asp:RadioButtonList id="rb" runat="server"
AutoPostBack="True"
onSelectedIndexChanged="displayMessage" />
<p><asp:label id="lbl1" runat="server" /></p>
</form>

</body>
</html>
```

Пример 25 – използване на атрибута AutoPostPack на контрол RadioButtonList.

13. Контроли за валидизация(Input Validation Controls)

13.1. Предназначение

В ASP приложенията проверката на валидността на данните изисква доста програмен код. Необходимо е да се изпълни код за проверка на данните за валидност или от страна на клиента, или от страна на сървъра или и на двете места едновременно. При откриване на грешка е необходимо да се изведе съобщение, което да укаже на клиента (оператора) какви корекции трябва да направи във входните данни.

С въвеждането на контролите за валидизация в ASP.NET проверката на валидността на данните се улеснява значително. В повечето случаи програмистът може да извърши необходимите проверки на въведените данни без да пише код а само като задава критерии за проверка

13.2. Видове контроли за валидизация.

Контролите за валидизация се добавят към ASP.NET страницата като другите контроли. Има контроли за специфични типове валидизация -проверка за попадане на стойността в зададена област

(range validator), проверка за съответствие със зададен шаблон (pattern validator) и др.

Контролите за валидизация осъществяват контрол на въведените данни както от страна на клиента, така и от страна на сървъра. Проверката от страна на сървъра е необходима за да се избегне неоторизирания достъп до сървърното приложение чрез заобикаляне на проверките от страна на клиента.

С един WEB контрол за въвеждане на данни могат да се асоциират няколко контрола за валидизация

ASP.NET предоставя следните контроли за валидизация:

Тип на контрола	Функция
RequiredFieldValidator	Проверява дали е въведена стойност в контрола
CompareValidator	Сравнява съдържанието на един контрол за въвеждане със зададена стойност или със съдържанието на друг контрол.
RangeValidator	Проверява дали стойността, съдържаща се в един контрол за въвеждане е в зададен интервал от стойности или между стойностите, съдържащи се в други контроли.
RegularExpression Validator	Сравнява въведената стойност с шаблон, зададен като регулярен израз.
CustomValidator	Дава възможност за използване на потребителски код при извършване на проверките.
ValidationSummary	Извежда обща информация за грешките, генерирани от контролитеот всички контроли за валидизация.

13.3. Общи атрибути на контролите за валидизация

Всички контроли за валидизация наследяват класа **BaseValidator** и притежават следните общи (наследени от BaseValidator) атрибути:

1) Атрибут **errorMessage**

Задава и връща символен низ, който се извежда като съобщение в .aspx страницата когато контрола за валидизация генерира грешка.

2) Атрибут **ControlToValidate**
На този атрибут трябва да се присвои идентификаторът ID на съществуващ контрол за въвеждане на данни от контейнера (ASP.NET страницата). Ако ID не указва съществуващ контрол, се генерира грешка при визуализиране на (ASP.NET страницата.

3) Атрибут **Display**
Този атрибут определя поведението на контрола за валидизиране при извеждане на съобщение за грешка. Може да съдържа една от следните стойности:

None	Контролът за валидизация не извежда съобщение за грешка в областта, в която е разположен. Тази стойност се задава, когато искаме съобщението за грешка да се изведе само в общия за всички валидизиращи обекти контрол ValidationSummary.
Static	Заделя се статично област за извеждане на съобщение за грешка от дадения контрол за валидизация. В резултат на това при извеждане на съобщение за грешка не се променя разположението на елементите на ASP.NET страницата.
Dynamic	Областта за извеждане на съобщение за грешка се заделя динамично, което означава, че извеждането на съобщението може да предизвика разместване на елементите на ASP.NET страницата

4) Атрибут **EnableClientScript**
Задава и връща логическа стойност, която определя дали ще се извършва валидизация от страна на клиента. Ако потребителският браузър изпълнява Java Script, валидизацията може да бъде изпълнена и от страна на клиента, но ако се зададе стойност **false** на атрибута **EnableClientScript** валидизация ще се извършва само от страна на сървъра.

5) Атрибут **Enabled**
Задава и връща логическа стойност, която определя дали е разрешено функционирането на контрола за валидизация. Ако атрибута **Enabled** има стойност **false**, контролът няма да извършва валидизация. Същия резултат ще има и задаването на стойност **false** на атрибута **Visible,** с тази разлика, че в тоя случай контролът за валидизация няма да се визуализира изобщо, т.е. няма да се заделя място за него в ASP.NET страницата.

6) Атрибут **ForeColor**
Задава и връща стойност от тип **System.Drawing.Color**, която определя цвета на текста на извежданото от кнтрола за валидизация съобщение за грешка

7) Атрибут **Type** (изброим тип ValidationDataType)
Задава типа на данните, които контролът CompareValidator или RangeValidator сравнява. Възможните стойности са String, Integer, Double, Date, Currency. Стойностите на съответните атрибути се задават като низове, а при сравнението се преобразуват в зададеният тип. Ако не може да се извърши преобразуването, не се прави сравнение.

13.4. Контрол **RequiredFieldValidator**

Проверява дали е въведена стойност в контрола, указан с атрибута ControlToValidate. Ако не е въведена стойност се генерира грешка. Контролът RequiredFieldValidator притежава само общите атрибути на валидизиращите контроли, дадени по-горе. Даденият по-долу пример съдържа ASP.NET контрол за въвеждане и контрол за валидизация от тип RequiredFieldValidator, свързан с него.

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

Пример 26 – Използване на контрол RequiredFieldValidator.

13.5. Контрол **CompareValidator**

Сравнява съдържанието на един контрол за въвеждане със зададена стойност (ValueToCompare) или със съдържанието на друг контрол (ControlToCompare). Генерира се грешка по зададено условие.
Специфични атрибути:

1. Атрибут **ControlToCompare**
Задава и връща идентификатор на контрола, от който се взема стойност за сравняване.
2. Атрибут **ValueToCompare**

Задава и връща стойност за сравняване.
Ако са зададени стойности и на двата атрибута предимство се дава на ControlToCompare

13.6. Контрол **RegularExpressionValidator**

Сравнява въведената стойност с шаблон, зададен като регулярен израз. Генерира се грешка ако липсва съвпадение със шаблона.

Специфичен атрибут: ValidationExpression (тип string) - Задава и връща регулярен израз който задава шаблона, използван за валидизиране на въведената стойност в проверявания контрол.

Регулярните изрази са много мощни шаблони за проверка на валидността на въведените данни. Контролът RegularExpression Validator предоставя списък от готови шаблони (напр. за проверка на валидността на e-mail адрес, на интернет адрес, на телефонен номер) които, ако се реализират без използването на регулярен израз, биха изисквали многократни проверки с конструкции if-else.

14. HTML сървърни контроли

HTML контролите са дефинирани в именното пространство System.Web.UI. HTMLControls. За да бъде достъпен като контрол от страна на сървъра и за да може да генерира събития които да се отработват от страна на сървъра, е необходимо:

- в HTML етикета на контрола да се включи атрибут runat=”server”.
- HTML контролът да се включи във WEB формуляр (т.е. HTML формуляр с атрибут runat=”server”.
- HTML етикета на контрола да бъде правилно затворен с “/>” или със затваряща част, например
  <input id="Submit1" type="submit"…… /> или
  <input id="Submit1" type="submit"…… ></input>

Освен като сървърни контроли HTML контролите могат да генерират събития и като обикновени HTML контроли. Например даденият по-долу пример съдържа код на HTML бутон с атрибут runat=”server”., който съдържа манипулатор **onclick** за събитие, което се генерира от страна на браузъра, и манипулатор **onserverclick** за събитие, което се генерира от страна на сървъра.

```
<input id="Submit1" type="submit" value="submit" onclick="return
Submit1_onclick()" runat="server" onserverclick="Submit1_ServerClick" />
```

Пример 27 Код на HTML бутон с манипулатори за събития от страна на браузъра и от страна на сървъра.

15. ASP.NET конфигуриране

ASP.NET конфигурационната система дава възможност за задаване на конфигурационни параметри, които се зареждат при първото изпълнение на ASP.NET приложението и могат да бъдат променяни по всяко време без това да нарушава нормалното функциониране на Web приложенията и на Web сървъра.

Информацията за конфигурирането на ASP.NET приложенията се съхранява в текстови файлове в XML формат. Те могат да бъдат създадени с всеки текстов редактор. Във всяка директория на приложението може да се включи конфигурационен файл с име Web.config, като всеки от тези файлове прилага включените в него конфигурационни параметри към неговата собствена директория и всички нейни поддиректории. Конфигурационните файлове в дъщерните директории могат да надпишат параметрите, зададени във Web.config файла на родителската директория.

Освен конфигурационните файлове Web.config ASP.NET предоставя на администратора на Web сървъра конфигурационен файл Machine.config, с който могат да се зададат параметри на целия Web сървър. Спецификацията на файла е:

```
sdrive:\Microsoft.NET\Framework\versionNumber\CONFIG\Machine.config
```

където:
sdrive е системното дисково устройство
versionNumber е номерът на версията на NET.Framework

При изпълнение на заявки ASP.NET използва информацията от йерархичната структура от конфигурационни файлове като съхранява конфигурационните параметрите в ресурс за използването им при следващите заявки. Йерархичната структура се създава в съответствие с адресите на заявките (URL) а не в съответствие с файловата структура на Web сървъра.

ASP.NET открива измененията в конфигурационните файлове и автоматично прилага променените конфигурационни параметри. Това спестява необходимостта от рестартиране на сървъра при променяне в някой от конфигурационните файлове. Изключение от това правило са промените в секцията <processModel> на конфигурационния файл Machine.config, които изискват рестартиране на сървъра за да бъдат приложени. Конфигурационната система на ASP.NET е разширяема. Разработчикът може да добавя нови секции и конфигурационни параметри, като тези изменения също се отработват автоматично.

По подразбиране ASP.NET защитава конфигурационните файлове от външен достъп. При опит за отваряне на файл Web.config с потребителския браузър се генерира грешка “HTTP access error 403 (forbidden)”.

16. Сигурност на ASP.NET приложенията

16.1. Идентификация на потребител в ASP. NET.

Идентификацията на потребител (Authentication) за WEB приложенията е процес на получаване от потребителя на информация дали той е акредитиран да работи с приложението.

Одобряването на потребител (Authorization) е даване на разрешение на потребителя за достъп до ресурсите на приложението.

Както се вижда от дадените по-горе определения, тези операции са тясно свързани. Идентификацията предшества одобряването на потребителя. Дори когато приложението позволява анонимен достъп, въпреки това се прави идентификация на потребителя като анонимен. Разработчикът може да извършва идентификация със свой код или да я делегира на оторизиран идентификатор (напр. Microsoft Passport).

ASP.NET предоставя следните възможности за идентификация:

- Windows идентификация
- Паспортна идентификация – чрез централизирана услуга на Microsoft идентификация
- Идентификация чрез формуляр

Вида на идентификацията се задава в секцията <configuration> на конфигурационния файл Web.config

```
<configuration >
<system.Web>
```

```
  ...
  <authentication mode="authmode"/>
  ...
</system.Web>
</configuration >
```

Стойностите на параметъра "authmode" за различните видове идентификация са съответно Windows, Passport, Form или None, ако не се прави идентификация.Тъй като идентификацията и оторизирането са ключови елементи за сигурността на интернет приложенията, ще разгледаме по-подробно възможностите, които ни предоставят за извършването им ASP.NET и Internet Information Services (IIS).

16.2. Windows идентификация

Windows идентификацията в ASP.NET приложенията се изпълнява чрез Internet Information Server (IIS). IIS предоставя възможност за четири вида идентификация: анонимна (anonymous), базова (basic) с хеширане (digest) и интегрирана (windows integrated).

Анонимната идентификация е всъщност липса на идентификация. По подразбиране (без допълнително конфигуриране) IIS не прави никаква идентификация. ASP.NET приложението е достъпно за всички.

Базовата идентификация изисква потребителят да изпрати име и парола. Тази информация обаче се изпраща към IIS в некриптиран вид (като обикновен текст), което дава ниско ниво на сигурност на този тип идентификация.

Идентификацията с хеширане също изисква потребителят да изпрати име и парола. Тази информация обаче се изпраща към IIS в хеширан вид, което дава по-високо ниво на сигурност на този тип идентификация. Необходимо е потребителският браузър да поддържа хешрането (за Internet Explorer това означава да е поне пета версия) и да има достъпен акаунт на потребителя.

Интегрираната идентификация не изисква изпращането на парола. Идентификацията използва специализиран (Kerberos или challenge/ response) протокол за идентификация на потребителя.

16.3. Паспортна идентификация

Паспортна идентификация използва услугата Microsoft's passport service за да идентифицира потребителя на ASP.NET приложението. Ако потребителите на приложението са си регистрирали паспорт в тази услуга, и разработчикът конфигурира приложението за да използва паспортна идентификация, Web услугата поема всички операции по идентификацията.

Паспортът използва криптирана босквидка за да получи информация за идентифицирането потребителя. Ако потребителят притежава паспорт, когато той посети сайта, ще се счита за идентифицитан от ASP.NET. В противен случай той ще бъде пренасочен към услугата за регистриране. След успешно регистриране потребителят се пренасочва към сайта.

За да използва паспортната идентификация разработчикът трябва да изтегли Passport Software Development Kit (SDK) и да го инсталира на своя сървър. Адресът, от който продуктът може да бъде изтеглен е: http://msdn.microdoft.com/library/default.asp?url=/downloads/list/ Websrvpass.aps. Той предоставя всичко необходимо за използването на паспортна идентификация от ASP.NET приложение.

16.4. Идентификация чрез формуляр

Идентификацията чрез формуляр позволява на разработчика да запише информацията за идентификация (напр. име и парола) във файла Web.config, или да използва собствен метод. ASP.NET поема грижата да съхранява в рамките на сесията идентификационната информация за потребителя.

При идентификацията чрез формуляр когато потребителят заяви страница от приложението, ASP.NET проверява дали на компютъра на потребителя има записана определена сесийна бисквидка. Ако бисквидката съществува, ASP.NET счита потребителя за идентифициран и изпълнява заявката.

Ако бисквидката не съществува, ASP.NET пренасочва потребителя към формуляр за регистрация, указан от атрибута loginUrl на тага <form> в конфигурационния файл Web.config. При успешно регистриране ASP.NET приложението записва сесийна бисквидка за регистрацията на потребителския компютър и пренасочва потребителя към началната страница на приложението.

16.4.1. Конфигуриране на идентификация чрез формуляр

Файлът Web.config дава възможност за конфигуриране на два вида идентификация чрез формуляр:

За конфигуриране на идентификация чрез формуляр разработчикът може да използва файла Web.config, като включи в него имената и паролите на потребителите, за които е разрешен достъп до ASP.NET приложението, както е показано в следващия пример

```
<configuration>
   <system.Web>
      <customErrors mode="Off"/>
      <authentication mode="Forms">
         <forms name="AuthCookie"
            path="/"
            loginUrl="login.aspx"
            protection="All"
            timeout="30">
            <credentials passwordFormat="Clear">
               <user name="jeff" password="test" />
               <user name="mike" password="test" />
            </credentials>
         </forms>
      </authentication>
      <authorization>
         <deny users="?" />
      </authorization>
   </system.Web>
</configuration>
```

Пример 28 Съдържание на файл Web.config с параметри за идентификация чрез формуляр

Да разгледаме значението на използваните във Web.config параметри, свързани с идентификацията

<customErrors mode="Off"/>	Параметърът customErrors определя дали извеждането на съобщения за грешно въведени данни от потребителя се разрешава, забанява или се разрешава само изпращането им към клиента. Възможните стойности са Off. On и RemoteOnly
<forms name="AuthCookie" path="/" loginUrl="login.aspx" protection="All" timeout="30">	Секцията forms задава: - име на бисквидката: name="AuthCookie" - път и име на формуляра за идентификация path="/" loginUrl="login.aspx" - контролно време timeout="30"
<credentials passwordFormat="Clear">	Атрибутът passwordFormat в подсекцията credentials на секция forms задава криптиране на паролата. Възможните стойности са: - Clear – без криптиране - MD5 – криптиране с MD5 алгоритъм - SHA1 – криптиране с SHA1 алгоритъм
<user name="jeff" password="test" /> <user name="mike" password="test" />	Параметърите <user> в подсекцията credentials на секция forms задават име и парола на потребител. Ако чрез passwordFormat е зададено криптиране, стойностите на паролите трябва да са зададени в криптиран вид

Проверка дали потребителят е регистриран

Обектът Identity (достъп до него се осъществява чрез едноименния атрибут на обект User) предоставя информация за идентификацията на потребителя чрез следните атрибути:

IsAuthenticated	- Връща логическа стойност true ако потребителят успешно се е идентифицирал и

	false в противен случай
AuthenticationType	- Връща символен низ с типа на използваната идентификация
Name	- Връща символен низ с името на потребителя

Даденият по-долу пример съдържа събитийната процедура на Web контрола Login в модула за идентификация на потребител на ASP.NET приложение.Освен проверката чрез метода Authenticate потребителят може да включи и проверка със собствен код, като например сравнява въведените име и парола със записана в база от данни информация за регистрирани потребители,

```
protected void Login1_Authenticate(object sender, AuthenticateEventArgs e)
   {
      if (FormsAuthentication.Authenticate(Login1.UserName,
Login1.Password))
      {
         FormsAuthentication.RedirectFromLoginPage(Login1.UserName,
false);
      }
      else
      {
         Response.Write("<script>alert('Error!')</script>");
      }
}
```

Пример 29 Събитийна процедура на Web контрол Login за идентификация чрез формуляр по зададенн във Web.config имена и пароли

16.4.2. Класът FormsAuthentication

Класът FormsAuthentication предоставя атрибути и методи, които разработчикът може да използва за идентификация на потребители. Методът RedirectToLoginPage пренасочва браузъра към страницата за идентификация, чийто адрес е записан в LoginUrl на конфигурационния файл. Методът RedirectFromLoginPage пренасочва браузъра обратно към страницата, която е генерирала заявка за идентификация, или към подразбиращата се страница на ASP.NET приложението. Класът

предоставя също така методи за създаване и използване при необходимост на т.н. идентификационни билети (authentication tickets).

Идентификация по зададенн във Web.config имена и пароли – метод Authenticate

Методът сравнява име и парола на потребител, зададени като параметри, с имена и пароли на потребители, съхранени в конфигурационния файл Web.config. При откриване на съвпадение с някоя от съхранените двойките име/парола методът връща стойност true. Това дава възможност да се идентифицира потребителят и да му се предостави достъп до модула на ASP.NET приложението, което е генерирало заявка за идентификация чрез формуляр, както е показано в дадения по-горе пример.

Декларация (C#)

```
public static bool Authenticate (string name, string password)
```

Прекратяване на идентификацията – метод SignOut()

Методът прекратява идентификацията на потребителя като премахва идентификационния му билет от бисквидките или URL.

Декларация (C#)

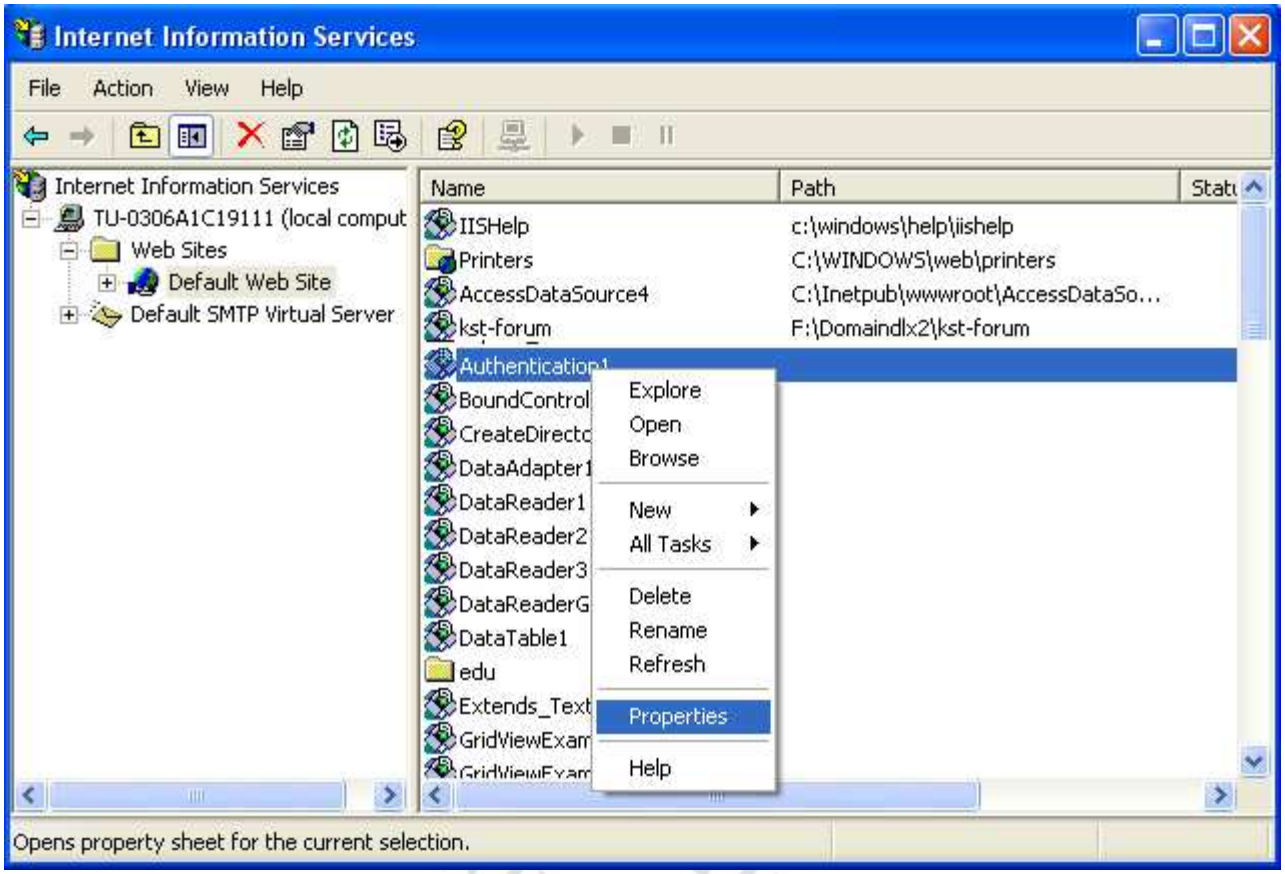
```
public static void SignOut()
```

Пример:

```
FormsAuthentication.SignOut(); //Прекратява идектификацията
FormsAuthentication.RedirectToLoginPage(); //Зарежда Login страницата
```

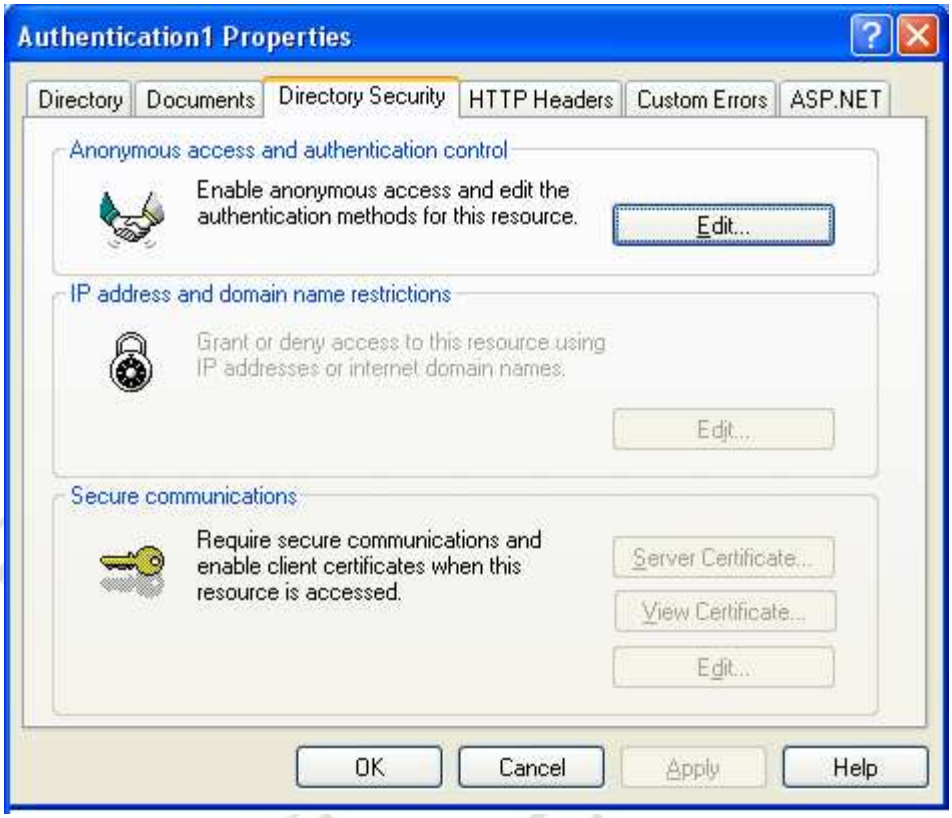
16.5. Конфигуриране на Windows идентификация

За конфигуриране на Windows идентификация за ASP.NET приложение се използва IIS Manager, с който трябва да се конфигурира виртуалната директория на приложението. Освен това трябва да се конфигурира и елементът <authentication> на файла Web.config на приложението.

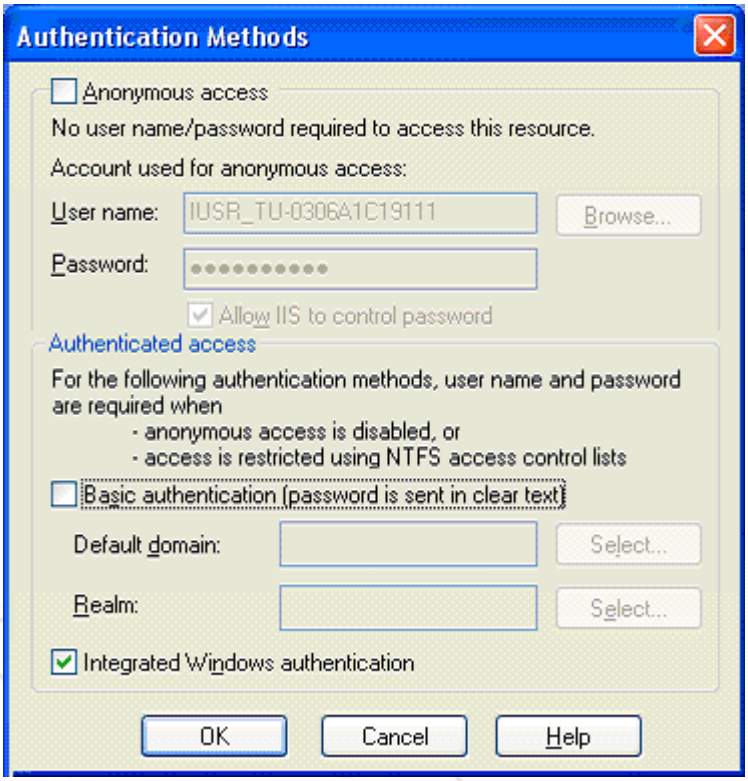


. Необходимо е да се изпълни следната последователност от операции:

1) Стартира се управляващата програма (IIS Manager) на Internet Information Services (IIS) от прозореца Control Panel→Administrative Tools→Internet Information Services.
2) Щраква се с десния бутон на мишката върху виртуалната директория на ASP.NET приложението и от изведеното меню се избира “Properties”.
3) От прозореца “Properties” се избира страницата Directory Security.

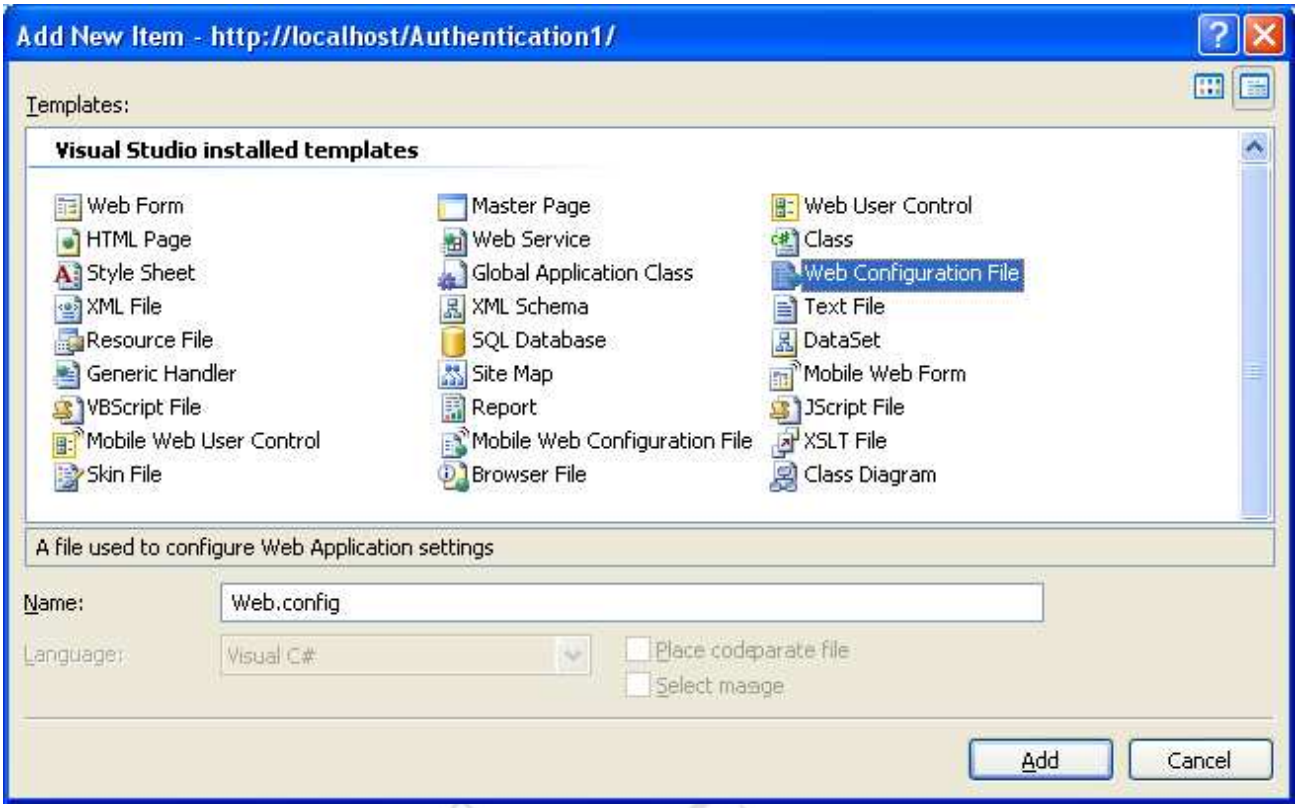


4) Щраква се върху бутона Edit под “Anonymous access and authentication control”.
5) Деселектира се кутията за маркиране Anonymous.
6) Селектира се кутията за маркиране Integrated Windows authentication.



7) Потвърждава се с бутона “OK” избрания метод за идентификация в текущия прозорец и в прозореца “Properties”

8) От менюто Website→Add New Item към приложението се добавя конфигурационен файл Web.config



9) В конфигурационния файл Web.config на приложението, на елемента authentication mode трябва да се зададе стойност "Windows" , както е показано в дадения по-долу пример.

```
<system.Web>
 ...
 <authentication mode="Windows"/>
 ...
</system.Web>
```

Пример 30 Задаване на Windows идентификация във файла Web.config.

Докато се разработва приложението, за изпълнението му в на режима за настройка (Debugging), е необходимо да се конфигурира Windows идентификация, както е описано по-горе, но може и да се запази анонимната идентификация за да се избегне необходимостта от въвеждане на име и парола при всяко стартиране.

16.6. Получаване на информация за потребителя при Windows идентификация

При използване на Windows идентификация разработчикът може да получи информация за името и паролата на потребителя чрез атрибута

User на обект HttpContext. Атрибутът връща обект principal представящ контекста за сигурност на потребителя, в който кодът се изпълнява. Този контекст включва интерфейса Iidentity към информацията за потребителя и ролите, които той притежава.
Декларация (C#)

```
public IPrincipal User { get; set; }
```

```
Response.Write("User=" + HttpContext.Current.User.Identity.Name);
```

Пример 31 Получаване на информация за потребител чрез атрибута User на обект HttpContext

Информация за идентификацията на потребителя може да се получи и от сървърните променливи, съдържащи се в колекцията ServerVariables на обект Request. По-долу е дадена информация за значението на тези сървърни променливи.

LOGON_USER	Windows NT акаунтът с който се е включил потребителят.
REMOTE_USER	Потребителско име, изпратено със заявката. Това е името, което в действителност се изпраща, в противоположност на това, което се получава след прилагането на някой идентификационен филтър, инсталиран на сървъра.
AUTH_USER	Необработено идентификационо име.
AUTH_PASSWORD	Стойност, въведена в клиентската идентификационна диалогова кутия. Тази променлива е достъпна само при използването на базова идентификация.
AUTH_TYPE	Име на идентификационния метод, който сървърът използва за да валидизира потребителите, когато те заявят защитен код.

Повече информация за идентификацията и одобряването на потребител можете да намерите на http://msdn.microsoft.com/en-us/library/ ms998358.aspx

Други online материали за конфигуриране и идентификация в ASP.NET
http://www.codersource.net/asp_net_Web_configuration_file.html

http://www.c-sharpcorner.com/UploadFile/lmoningi/AuthenticationAndAuthorizatio11252005233533PM/AuthenticationAndAuthorizatio.aspx
http://www.c-sharpcorner.com/UploadFile/sd_patel/WebConfigInASPNet11242005061608AM/WebConfigInASPNet.aspx

17. Съхраняване на данни в ASP. NET. Обект Cache

Обектът Cache на ASP.NET предоставя разширени възможности за съхраняване на обекти от различен тип в рамките на приложението. За всяко приложение на ASP.NET се създава отделен обект Cache, като времето на живот на обекта съвпада с времето на живот на приложението. Това означава например, че при рестартиране на WEB сървъра всички обекти Cache на приложенията ще бъдат унищожени и в последствие ще бъдат създадени отново за всяко приложение при първото му заявяване от някой клиент (WEB браузър). Обектът Cache предоставя прост интерфейс от тип речник*, който включва методи за запис на данни и за извличане на данни. Освен разгледаните по-долу методи обект Cache наследява от класа Object метода GetType( ), чрез който може да се получи типа на записан със зададен ключ елемент и метода ToString( ), който връща символно представяне на записания елемент.
Класът Cache се съдържа в именното пространство **System.Web.Caching.** Класът е деклариран като ненаследяем (**NotInheritable** за VB.NET , **sealed** за C#).
Пример за използване на обекта за запис и извличане на данни:

```
this.Cache.Insert("Send_Data","Hello from First Page");
```

Запис на данни (ключ и стойност) в кеш паметта с метод Insert

```
Label1.Text=Cache.Get("Send_Data").ToString();
```

Прочитане на записаните данни от кеш паметта посредством метод Get.

17.1. Атрибути на обект Cache

Count – (тип int) – връща броя на елементите, записани в обекта Cache

* Т.е. данните се записват като двойка ключ-стойност

Item – (тип string) – задава и връща стойност на елемент, записан в обекта Cache, по зададен ключ. Въпреки че по документация Item е атрибут, достъпът до елементите чрез него става по синтаксиса за достъп до елементите на масив.
Примери (C#)
Cache["ime"] = "Peter"; //запис на елемент с ключ "ime" и стойност "Peter"
String ime = Cache["ime"]; //Прочитане на стойността на елемент с ключ “име”

17.2. Методи на обект Cache

17.2.1. Запис на елемент в обект Cache. Методи Add и Insert

Методите Add и Insert дават възможност за добавяне на нов елемент към асоциативния масив на обект Cache. Разликата между двата метода е в това, че методът Add дава възможност за задаване на допълнителни атрибути на елемента и освен това връща обектна променлива (референция) за добавения елемент, докато метода Insert не връща стойност.

17.2.2. Метод Add (обект Cache)

Декларация (C#)
```
public Object Add (
        string key,
        Object value,
        CacheDependency dependencies,
        DateTime absoluteExpiration,
        TimeSpan slidingExpiration,
        CacheItemPriority priority,
        CacheItemRemovedCallback onRemoveCallback
)
```

където:
key е символен низ, който съдържа ключа на елемента.
value е параметър, който задава стойността, която се присвоява на елемента.

dependencies е файл или ключ на кеш елемент, който е свързан (dependencie) със създавания елемент.. Когато някой от свързаните елементи се промени, създавания елемент става невалиден или се премахва от кеш паметта. Ако елементът няма свързани с него елементи, атрибутът се присвоява стойност null.

absoluteExpiration задава моментът, когато обектът се премахва от паметта. Ако за задаване на интервала на валидност на елемента се използва параметъра slidingExpiration, тогава стойността на параметъра absoluteExpiration трябва да бъде NoAbsoluteExpiration.

slidingExpiration задава интервала от време, след изтичане на който елементът се счита с изтекла валидност и се унищожава. Ако за задаване на момента на изтичане на валидността на елемента се използва параметъра absoluteExpiration, тогава стойността на параметъра slidingExpiration parameter трябва да бъде NoSlidingExpiration.

Priority задава приоритет на обекта – ако е зададен, се отчита при премахване на обекти.

onRemoveCallback Делегат, който, ако бъде зададен, се извиква когато елементът се премахва от паметта. Може да се използва за да се изпълни потребителски код при премахване на елемента.

Връщана стойност: от тип Object, ако елементът е записан в кеш паметта в противен случай стойност null (Nothing за Visual Basic).

Пример:
Примерът създава метод AddItemToCache. Когато се извика методът, той задава на атрибута itemRemoved стойност false и регистрира метод onRemove като нова инстанция на делегата CacheItemRemovedCallback. Този делегат се присвоява на атрибута RemovedCallback на метода Add. Прави се проверка дали елемент с ключ Key1 е записан в обекта Cache. Ако Cache["Key1"] върне стойност null, се използва метода Add за да се запише елемент с ключ Key1, стойност "Value 1", абсолютно време за изтичане на валидност 60 секунди, и най-висок приоритет в кеш паметта. Създадения елемент се асоциира с метода onRemove чрез аргумента onRemoveCallback. Това дава възможност метода

RemovedCallback да бъде извикван когато елемента се премахва от паметта.
Пример:

```
public void AddItemToCache(Object sender, EventArgs e) {
   itemRemoved = false;

   onRemove = new CacheItemRemovedCallback(this.RemovedCallback);

   if (Cache["Key1"] == null)
     Cache.Add("Key1", "Value 1", null, DateTime.Now.AddSeconds(60),
TimeSpan.Zero, CacheItemPriority.High, onRemove);
}
```

Пример 32 Добавяне на елемент към обект Cache чрез метода Add.

На елемента се задава момент за изтичане на валидността чрез стойността DateTime.Now.AddSeconds(60) на параметъра absoluteExpiration. На параметъра **slidingExpiration** се задава стойност TimeSpan.Zero, което означава, че той в случая не се използва. На параметъра **priority** се задава стойност CacheItemPriority.High (висок приоритет), а на параметъра **onRemoveCallback** се присвоява събитийната процедура onRemove, която ще се изпълни когато се премахне елементът от обекта Cache.

17.2.3. Метод Insert (обект Cache)

Методът Insert, както и методът Add се използва за променяне на съществуващ елемент със същия ключ. За разлика от метода Add, методът Insert не връща стойност. Методът има няколко варианта с различен набор от параметри:

Cache.Insert (String, Object)
Cache.Insert (String, Object, CacheDependency)
Cache.Insert (String, Object, CacheDependency, DateTime, TimeSpan)
Cache.Insert (String, Object, CacheDependency, DateTime, TimeSpan, CacheItemPriority, CacheItemRemovedCallback)

В най-пълния вариант Insert се извиква със същите параметри както метода Add. В най-простия си вариант методът се извиква само с два параметъра – key и value и има следната декларация:

Декларация (C#)

```
public void Insert (
        string key,                    //ключ
        Object value,                  //стойност
)
```

Пример:

```
if(Cache["DSN"]!=null) Cache.Insert("DSN", connectionString);
```

Пример 33 Променяне на елемент в кеш паметта посредством метода Insert

В примера съдържанието на променливата connectionString ще се запише в елемент от обекта Cache с ключ DSN. Останалите параметри (които трябва задължително да се зададат ако се използва метода Add) запазват стойностите, с които са създадени. Ако искаме да зададем нова стойност на времето за валидност на елемента, можем да направим това, като извикаме метода Insert с четири параметъра.

```
Cache.Insert("MyData", Source, null, DateTime.MaxValue,
TimeSpan.FromMinutes(20));
```

Пример 34 Запис стойност на елемент в кеш паметта с интервал на валидност 20 минути

17.2.4. Четене на данни от обект Cache – метод Get

Методът извлича елемент от обект Cache по зададен ключ.

Декларация (C#)

```
public Object Get (
        string key
)
```

Да разгледаме един пример:

```
string ime=Cache.Get("Author");
```

Пример 35 Прочитане на елемент от кеш-паметта чрез метод Get.

В примера чрез метода Get се прочита съдържанието на елемент с ключ "Author" и се присвоява получената стойност на променлива ime.

17.2.5. Изтриване на данни от обект Cache – метод Remove

Методът изтрива елемент от обект Cache по зададен ключ. Методът връща елемента, който се премахва от обект Cache. Ако елемента липсва, се връща стойност null.
Декларация (C#)
public Object Remove (string key)

Пример (C#):

```
if(Cache["Key1"] != null){
Cache.Remove("Key1");
}
```

Пример 36 Изтриване на елемент от кеш паметта посредством метод Remove

В примера се прави проверка дали елемента съществува, и ако съществува, се премахва чрез метода Remove.

17.2.6. Итерация през всички елементи в обект Cache. Метод GetEnumerator.

Методът извлича колекция *dictionary enumerator*, която може да се използва за изпълнение на итерация през всички елементи в обект Cache, като итерацията се извършва по стойностите на ключовете на елементите.
Декларация (C#):
public IDictionaryEnumerator GetEnumerator( )

Интерфейсът IDictionaryEnumerator предоставя колекция от елементи (структури) DictionaryEntry, които притежават следните атрибути:

- Key (тип Object), връща ключът на елемента
- Value(тип Object), връща стойността на елемента

- Entry (тип DictionaryEntry), връща структура с атрибути Key и Value. Структурата Entry предоставя алтернативна възможност за достъп до ключа и стойността на елемента.

Пример:

```
using System.Web.Caching;
using System.Collections;
//……….
IDictionaryEnumerator CacheEnum = Cache.GetEnumerator();
string cacheItem =”<OL>”; //Начален етикет на номериран списък
while (CacheEnum.MoveNext())
{
cacheItem += “<LI>”+Server.HtmlEncode(CacheEnum.Key.ToString( ) +
"=" + CacheEnum.Value.ToString( ))+”</LI>”;  //Добавяне на елемент към
//HTML списъка
}
Label1.Text= cacheItem + ”</OL>”; //Извеждане на списъка в обект Label
```

Пример 37 Извеждане на всички елементи, съдържащи се в обект Cache чрез итерация по ключ.

В примера се използва конструкция while, която извършва итерация през елементите на колекцията докато метода MoveNext() връща стойност True. Ключът и стойността на текущия елемент се извличат чрез атрибутите Key и Value и се добавят към HTML кода на елемент от списък. Обърнете внимание на използването на метода HtmlEncode на обект Server за прекодиране на символните низове, получени от атрибутитеKey и Value. Прекодирането е необходимо за правилното извеждане на “специалните” за HTML символи, например ъгловите скоби “<” и “>”.

Итерация през елементите на колекция, която се получава чрез метода GetEnumerator и притежава метода MoveNext() може да се осъществи и с конструкцията на C# foreach. Важно е да се отбележи, че елементите, които се получават от колекцията при всяка итерация са достъпни само по стойност (т.е. като копие) и поради това не могат да се променят и предават по адрес. По-долу е даден пример, който дава същия резултат като предходния, но реализиран с конструкция foreach.

```
using System.Web.Caching;
using System.Collections;
//……….
```

```
string cacheItem =”<OL>”; //Начален етикет на номериран списък
      foreach (DictionaryEntry item in Cache)
{
cacheItem += “<LI>”+Server.HtmlEncode(item.Key.ToString() + "=" +
item.Value.ToString())+”</LI>”;  //Доб. на елемент към HTML списъка
}
Label1.Text= cacheItem + ”</OL>”; //Извеждане на списъка в обект Label
```

18. Получаване на данни от потребителския браузър. Класът HttpRequest

Класът HttpRequest се съдържа в именното пространство **System.Web.** Класът е деклариран като ненаследяем (**NotInheritable** за VB.NET , **sealed** за C#). При заявка за .aspx страница се създава асоцииран с нея обект Request – екземпляр на класа HPPTRequest. Достъп до асоциирания с .aspx страницата обект **Request** може да се получи посредством атрибута **Request** на обект **HttpContext** или посредством аналогичния атрибут на обект **Page** Обектът **Request** предоставя атрибути, методи и колекции, свързани с получаването на данни от HTTP заявката. в .aspx страницата. Най-често данните в тялото на заявката са двойки име-стойност, изпратени от HTML или WEB формуляр.

18.1. Колекции на обект Request

Обект Request предоставя няколко колекции, които са от клас NameValueCollection, деклариран в именното пространство пространство System.Collections.Specialized. Колекциите от този клас съдържат двойки ключ-стойност от тип string. Данните, изпратени от HTML или WEB формуляр по метода GET и по метода POST могат да бъдат извлечени от колекциите FORM и QueryString на обект Request. Освен това обектът Page предоставя колекции Cookies, която съдържа бисквидките, записани на потребителския компютър, колекцията от сървърни променливи ServerVariables. Две са новите (в сравнение с ASP технологията колекции - колекцията от заглавни части Headers и колекцията от прикачени файлове Files.

Form	Връща колекция от данни, изпратени по метода POST.
QueryString	Връща колекция от данни, изпратени по метода GET.
Cookies	Връща колекция от "бисквидки", изпратени от клиента.
ServerVariables	Връща колекция от променливи на Web сървъра.
Headers	Връща колекция от HTTP заглавни части (headers).
Params	Връща обща колекция, която съдържа данните от колекциите QueryString, Form, ServerVariables и Cookies.
Files	Връща колекция, съдържаща изпратени от потребитела файлове (Multipart MIME format).

Даденият по-долу пример извежда имената на WEB контролите от колекцията Form на обект Request. Атрибутът AllKeys на колекцията връща масив от тип string, който съдържа всички ключовете на всички елементи от колекцията, които за колекция Form са имената на контролите от HTML формуляра, от тойто са изпратени даннитео метода POST.

```
System.Collections.Specialized.NameValueCollection coll;

//Load Form variables into NameValueCollection variable.
coll=Request.Form;
// Get names of all forms into a string array.
String[] arr1 = coll.AllKeys; //Връща масив, съдържащ ключовете в coll
for (int loop1 = 0; loop1 < arr1.Length; loop1++)
{
	Response.Write("Form: " + arr1[loop1] + "<br>");
}
```

Пример 38 Извеждане на имената на всички контроли във формата, от която са изпратени данни по метода POST.

Итерация през всички елементи от колекция на обект Request можем да извършим и посредством конструкцията на C# foreach, както е показано в следващия пример.

```
string s="<TABLE>";
foreach(string key in this.Request.Form.AllKeys)
{// не извежда __VIEWSTATE и __EVENTVALIDATION
        if(key.Substring(0,2)!="__")
        s += "<TR><TD>"+key + "</TD><TD>"
        + this.Request.Form.Get(key) + "</TD></TR>";
}
Response.Write(s +"</TABLE>");
```

Пример 39 Извеждане на данните (двойки име-стойност) от колекцията Form на обект HTTPRequest.

18.2. Атрибути на обект от клас HTTPRequest

AcceptTypes	Връща масив от тип String, който съдържа низове-идентификатори на поддържаните от клиентския браузър типове извеждана информацияа (MIME accept types).
ApplicationPath	Връща виртуалния път на ASP.NET приложението спрямо главната директория на сървъра.
Browser	Връща информация за параметрите на браузъра, изпратил заявката.
ClientCertificate	Връща текущия сертификат за сигурност (security certificate) на клиентския браузър.
ContentEncoding	Връща номера на кодовата таблица за заявката.
ContentLength	Връща дължината в байтове на данните, изпратени от клиента.
ContentType	Връща типа на данните в заявката, изпратена от клиента (MIME type).
CurrentExecutionFilePath	Връща виртуалния път на приложението, от който е изпратена текущата заявка.
FilePath	Връща виртуалния път на документа, от който е изпратена текущата заявка.
Filter	Задава и връща филтър, който да се използва при четене на данни от текущия входния поток.
HttpMethod	Тип String - Връща името на HTTP метода, по който са изпратени данните от клиента (GET, POST или HEAD).

InputStream	Връща тялото (т.е. данните без заглавната част) на HTTP заявката.
IsAuthenticated	Тип bool - Връща стойност, която показва, дали потребителят е бил идентифициран.
IsSecureConnection	Тип bool - Връща стойност, която показва дали се използва криптирана HTTP (secure sockets), т.е. HTTPS протокол.
Path	Тип String - Връща виртуалния път на текущата заявка.
PathInfo	Тип String - Връща допълнителна информация за пътя до ресурс с интернет адрес (URL extension).
PhysicalApplicationPath	Тип String - Връща пълен физически път до главната директория на изпълняваното интернет приложение.
PhysicalPath	Тип String - Връща физическия път съответстващ на заявения интернет адрес (URL).
RawUrl	Връща пълен (необработен) URL на текущата заявка.
RequestType	Има същото значение както HttpMethod. Запазен е за съвместимост с ASP технологията.
TotalBytes	Връща броя байтове, изпратени от потребителския браузър.
Url	Клас Uri - Връща информация за интернет адреса (URL) на текущата заявка.
UrlReferrer	Клас Uri - Връща информация за интернет адреса (URL) от който е заявена текущата страница.
UserAgent	Тип String - Връща необработено съдържание на заялавната част user agent , която съдържа информация за браузъра.
UserHostAddress	Тип String - Връща IP host адрес на мрежов клиент.
UserHostName	Тип String - Връща име (DNS) на мрежов клиент.
UserLanguages	Тип String[] - Връща сортиран масив с информация за езиците, зададени за използване на потребителския браузър.

18.3. Методи на клас HTTPRequest

BinaryRead	Тип byte[] - Чете зададен брой байтове от текущия входен поток.

GetHashCode	Тип int - (наследен от класа Object) Функция за хеширане, подходяща за използване в алгоритми за хеширане и структури от данни, напр. хеш-таблици..
MapImageCoordinates	Тип int[] – Връща масив, който съдържа координати, получени от обект image на HTML или WEBформуляр. Трябва да получи като параметър стринг с името на обекта image.
MapPath	Тип String – Връща физически път по зададен виртуален път – параметър от тип String.
SaveAs	Тип void – Записва данните от HTTP заявката във файл. Трябва да получи като параметър символен низ със спецификацията на файла и логическа стойност, която показва дали да се записват заглавните части на заявката.

19. Изпращане на данни към потребителския браузър. Класът HttpResponse

Класът HttpResponse (както и класа HttpRequest) е деклариран в именното пространство **System.Web** като ненаследяем (**NotInheritable** за VB.NET , **sealed** за C#). При заявка за .aspx страница се създава асоцииран с нея обект Response – екземпляр на класа HttpResponse. Достъп до асоциирания с .aspx страницата обект **Response** може да се получи посредством атрибута **Response** на обект **HttpContext** или посредством аналогичния атрибут на обект **Page** Обектът **Response** предоставя атрибути, методи и колекции, свързани с изпращането на данни на данни от.aspx страницата към клиента (потребителския браузър.

19.1. Атрибути на обект Response

Output	Връща обект TextWriter, чрез метода Write на който може да се изпраща форматиран низ в изходния поток (по подразбиране към Web браузъра. Пример: Response.Output.Write("I am {0} at {1:d}", "here", DateTime.Now);
Cookies	Връща колекцията бисквидки, създадени на сървъра и

	изпратени на клиента със заглавната част Set-Cookie. Пример: Response.Cookies.Add(new HttpCookie("MyColor","Green")); //Създава сесийна бисквидка MyColor

19.2. Методи на обект Response

Методите на обект Response могат да се разделят на няколко групи: - методи, които записват данни в непосредствено в изходния поток, методи, които добавят, променят или изтриват данни в буфера на изходния поток и методи, които извършват операции с файлове.

19.2.1. Методи за запис на данни в изходния поток

Това са най-често използваните методи на обект Response. Те дават възможност за запис на символен низ и двоични данни (метод BinaryWrite) в изходния поток.

Write	Записва символен низ в изходния HTTP поток.
BinaryWrite	Записва низ от двоични данни в изходния HTTP поток.
AddHeader	Добавя заглавна част (HTTP header) към изходния поток. Този метод е добавен за съвместимост с класическия ASP.
AppendHeader	Добавя заглавна част (HTTP header) към изходния поток.
WriteFile	Записва съдържанието на зададен файл непосредствено в HTTP изходния поток

19.2.2. Методи които извършват операции с буфера на изходния поток

Буферирането на данните от изходния поток за ASP.NET е разрешено по подразбиране. Обектът Response предоставя методи за запис, добавяне и изтриване данни в буфера, посредством които програмистът може да актуализира съдържанието на буфера, преди то

да бъде изпратено към клиента (потребителския браузър). Част от методите са аналогични на методите на обект Response на класическия ASP, но освен тях има добавени нови методи, основно за променяне на определени части от съдържанието на буфера, които липсваха в класическия ASP.

Clear	Изчиства цялото съдържание на буфера.
ClearContent	Изчиства съдържанието на буфера без заглавните части на HTTP заявката.
ClearHeaders	Изчиства от буфера заглавните части на HTTP заявката.
Flush	Изпраща текущото съдържание на буфера към клиента.След изпращането съдържанието на буфера се изчиства.
End	Изпраща текущото съдържание на буфера към клиента, спира изпълнението на текущия модул и генерира събитие Application_EndRequest event.
Close	Прекратява връзката с клиента на ниво сокет.

19.2.3. Методи които извършват операции с кешираните данни

AddCacheItemDependencie s	Извършва валидизация на кеширан отговор, зависим от други обекти в кеш паметта.
AddCacheItemDependency	Извършва валидизация на кеширан отговор, зависим от друг обект в кеш паметта.
AddFileDependencies	Добавя група от имена на файлове, към колекцията от имена на файлове, от които е зависим текущият отговор.
AddFileDependency	Добавя име на файл към колекцията от имена на файлове, от които е зависим текущият отговор.
AppendToLog	Добавя потребителска информация към log файла на IIS.
ApplyAppPathModifier	Добавя идентификатор на сесия (session ID) към виртуалния път и връща получения низ.

Pics	Добавя HTTP заглавна част PICS-Label към изходния поток.
Redirect	Пренасочва клиента към нов URL.
RemoveOutputCacheItem	Статичен метод – премахва от кеш паметта всички кеширани обекти, асоциирани със зададения път.

Ще разгледаме по-детайлно само някои от методите.

19.2.4. Метод **Write**

Методът **Write** изпраща символен низ в изходния поток към браузъра. В ASP технологията той е основният метод за извеждане на информация от процедурен блок на активната електронна страница. В ASP.NET текстова информация по правило се извежда в сървърните контроли.

Синтаксис:

Response.Write <expression>

където <expression> е израз, който се изчислява и резултатът, преобразуван в символен низ, се изпраща към потребителя.
Даденият по-долу пример изпраща към потребителя символния низ “Current Time is: “ и стойността на текущото време, получена от вградената функция на VBScript Date().

```
<html><body>
<%
Response.Write(“Current Date is: ”);
Response.Write(Date());
%>
</body></html>
```

Пример 40 Изпращане на информация към браузъра посредством метода **Write**

19.2.5. Метод **Redirect**

Методът **Redirect** на обекта **Response** дава възможност от кода на ASP.NET модула да се предизвика зареждане на друг документ в прозореца на браузъра Например, при проверка на регистрацията на

потребител, ако той е регистриран, може да се предизвика зареждането на модул за достъп до функциите на интернет приложението, а ако не е регистриран – зареждането на модул за регистриране.
ако имаме Web страница, където наш потребител е прехвърлен в една входна регистрационна форма, която ще провери автентичността на неговия потребителски ID относно базата данни на Access. Ако ID идентификаторът съществува, потребителят се изпраща в главната страница. Ако няма такъв номер, потребителят се прехвърля към една нова входна страница, за регистрирането му като нов потребител. Може да се използва конструкция If…Then заедно с метода Response.Redirect,
както е показано в следният пример:

```
if(login==true){
        //Потребителя се е включил успешно
        // и се допуска към   главната страница
         Response.Redirect("welcome.html");
      }else{
         //Потребителя не се допуска до системата
         Response.Redirect("newmember.aspx");
}
```

Пример 41 Използуване на метода Redirect за зареждане на нов документ в браузъра.

20. Атрибути и методи на WEB сървъра - клас HttpServerUtility

Класът предоставя методи, свързани с обслужването на WEB заявки. При изпълнение на ASP.NET приложение се създава вътрешен (intrinsic) обект **Server** – екземпляр на класа HttpServerUtility. Посредством обекта **Server** програмистът получава на разположение атрибутите и методите на класа HttpServerUtility.

20.1. Атрибути на обект Server

MachineName – (тип string) връща низ, който съдържа името на компютъра, на който се изпълнява WEB сървъра.

ScriptTimeout (тип int) Заадава и връща максималното време в секунди, за което WEB сървърът обслужва една заявка. След изтичане на това време изпълнението на ASP.NET модула се прекратява. Така се избягва възможността WEB сървърът да се ангажира неограничено време с изпълнението на някой ASP.NET модул (например такъв, който съдържа безкраен цикъл). Ако очакваме, че по някакви причини кодът на нашия ASP модул ще се изпълнява за време, превишаващо подразбиращата се стойност на ScriptTimeout (10 секунди), е желателно да присвоим по-голяма стойност на атрибута.

20.2. Методи на обект Server

ClearError	Изчиства всички генерирани преди това изключения.
CreateObject	Създава екземпляр на COM обект.
CreateObjectFromClsid	Създава екземпляр на COM обект по зададен идентификатор на клас. (CLSID).
Execute	(надписан) Изпълнява заявка за WEB ресурс в контекста на текущата заявка.
GetHashCode	(наследен от класа Object) Използва се като хеш функция в алгоритми за кеширане.
GetLastError	Връща последно генерираното изключение.
GetType	(наследен от класа Object) Връща обект от клас Type, който представя типа на текущата инстанция..
HtmlDecode	(надписан).Декодира символен низ, кодиран с метода **HtmlEncode**.
HtmlEncode	(надписан).Кодира символен низ така, че да може да бъде извеждан от WEB браузър.”Специалните” символи се заменят със символни последователности, които започват с “&” и завършват с “;”. Наапример “>” се заменя с “<”.
MapPath	Връща пълната файлова спецификация до файл на WEB сървъра по зададен относителен път..
Transfer	(надписан). Прекратява изпълнението на текущата страница и стартира изпълнението

	на нова страница, като в браузъра HTML кода от новата страница се добавя към изведения.
UrlDecode	(надписан).Декодира символен низ, кодиран с метода **UrlEncode.**
UrlEncode	(надписан). Кодира символен низ така, че да може да бъде добавен към интернет адрес (URL). ”Специалните” символи се заменят със символни последователности от вида %XX, където XX е ASCII кода на символа в 16-тична бройна система., Например “=” се заменя с %3D”.
UrlPathEncode	Кодира частта path (пътят от файлова спецификация) на интернет адрес (URL) по същия начин, както това прави методът **UrlEncode** за какъв да е зададен символен низ. Такова кодиране е необходимо, защото във файловата спецификация могат да се съдържат символи, които не трябва да се включват в URL, например празни интервали

20.2.1. Метод **CreateObject**

Методът създава инстанция на COM обект по зададен програмен идентификатор (ProgID). Програмният идентификатор се задава като символен низ, който съдържа името на библиотеката и името на класа, от който се създава обекта.
Декларация
[C#] public object CreateObject(string);
Пример:

```
ADODB._Connection Con =
(ADODB._Connection)Server.CreateObject("ADODB.Connection");
```

Пример 42 Създаване на COM обект с метода **CreateObject** на обект Server

В примера се създава обект Connection от библиотеката ADODB. Тъй като обектите от библиотеката ADODB имат т.н. апартаментна нишковост, за тяхното използване в ASP.NET модул е необходимо да се даде стойност true на атрибута AspCompat в директивата Page на .aspx

страницата на модула. Освен това е необходимо в ASP.NET проекта в развойната среда на Visual Studio.NET да се включи референция към файла на библиотеката чрез функцията AddReference на менюто Website. При включване на референция към COM библиотека Visual Studio.NET автоматично преобразува COM типовете в метаданни, създава асембли с метаданните и го записва в поддиректорията bin на проекта.

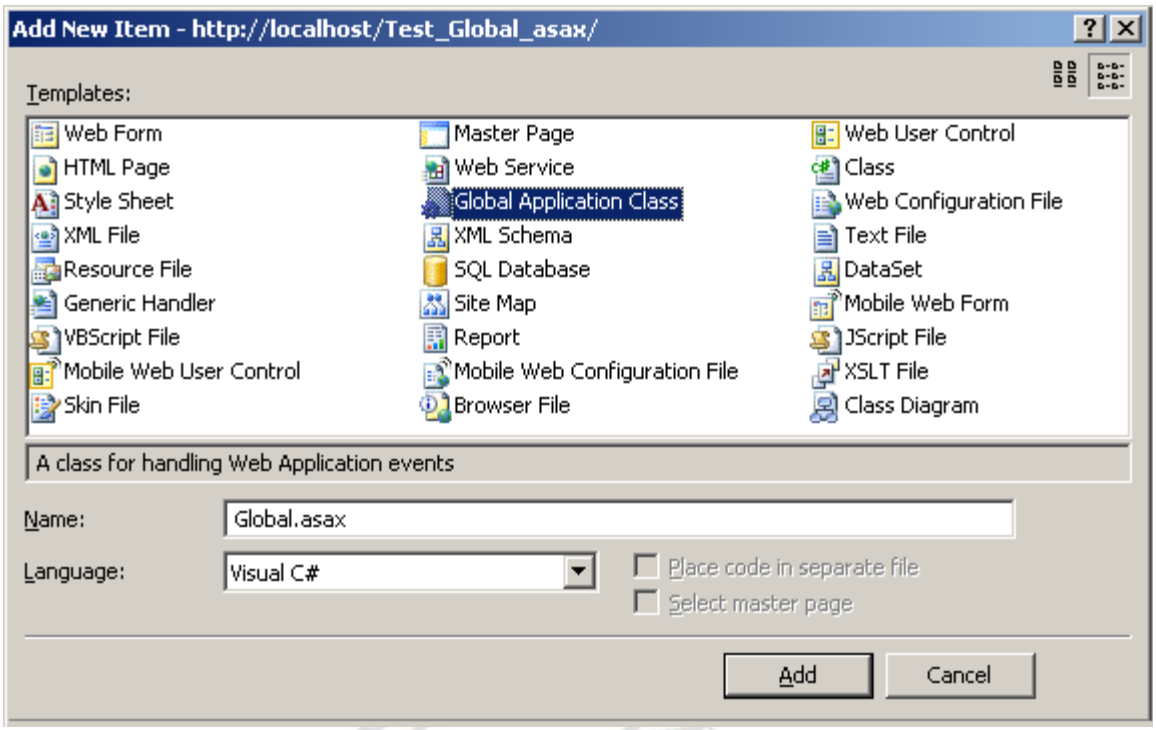
21. Файлът Global.asax

Файлът Global.asax е предоставя на програмиста възможност да използва събитийни процедури за начало и край на потребителска сесия и начало и край на изпълнението на ASP.NET приложение. Този файл не е задължителен, ако не се нуждаем от възможностите, които той ни дава можем просто да не го включваме в приложението. Ако се използва, Global.asax трябва да се запише в корена на нашата виртуална директория. При изпълнение на ASP.NET приложението Global.asax се парсва и компилира в динамично генериран клас на .NET Framework, който наследява базовия клас HttpApplication. Файлът Global.asax управлява събитията на обектите Application и Session и се използва също за създаване и унищожаване на обекти и променливи с област на действие в рамките на приложението и сесията. Всяко ASP.NET приложение може да съдържа само един файл Global.asax.

Файлът Global.asax file може да съдържа следното:

- Събитийни процедури на приложението (за общо 11 събития)
- Събитийни процедури на сесията Session_OnStart и Session_OnEnd.
- Декларации <object>
- Декларации TypeLibrary
- Директиви #include

21.1. Включване на файл Global.asax във WEB проект



21.2. Събитийни процедури на файла Global.asax

Обектите Application и Session имат събития, които настъпват в началото и края на създаването на обекта. В тях програмистът може да включи код, който да се изпълни при генериране на определени събития по време на изпълнението на заявка към ASP.NET приложението. Тези процедури могат да се използват например за инициализиране на обекти, за създаване на променливи с област на действие в рамките на приложението и сесията и т.н. По-долу е даден списък от имената на събитийните процедури и кратко описание на събитието, което предизвиква изпълнението им

Събитийна процедура	Събитие
Application_Start	Създаване на първи екземпляр на обект от клас HttpApplication class. Дава възможност за създаване на обекти, достъпни от всички екземпляри на HttpApplication.

Application_BeginRequest	Получаване на заявка за модул от ASP.NET приложение
Application_AuthenticateRequest	Заявката е готова за идентификация на потребителя
Application_AuthorizeRequest	Заявката е готова за идентификация на автора
Application_ResolveRequestCache	Обработката на заявката може да бъде прекратена ако има кеширана версия
Session_Start	Това събитие възниква само когато нов потребител заяви първата (за него) страница от ASP.NET приложението
Application_AcquireRequestState	Текущото състояние на приложението преди изпълнение на заявката е достъпно
Application_PreRequestHandlerExecute	Манипулаторът на заявката е готов за изпълнение

Дадените по-горе събитийни процедури (ако се декларирани в Global.asax) се изпълняват последователно при всяко заявяване на .aspx страница от ASP.NET приложението. Изключение прави само процедурата Session_Start, което се генерира само когато нов потребител заяви първата (за него) страница от ASP.NET приложението. Ако от тази страница (напр. чрез хипервръзка) се заяви друга страница от ASP.NET приложението, събитийната процедура Session_Start не се извиква отново.

След изпълнението на тези събитийни процедури се генерират събитията от жизнения цикъл* на заявения ASP.NET модул и се изпълняват съответните събитийни процедури (Page_Init, Page_Load ……) ако са декларирани в кода на модула. След последното събитие (Unload) от тоя жизнен цикъл се изпълняват последователно дадените по-долу събитийни процедури, ако са декларирани във файла Global.asax.

Събитийна процедура	Събитие
Application_PostRequestHandlerExecute	Завършено е изпълнението на

* Вижте т.8 Жизнен цикъл на .ASP.NET модулите

	заявката
Application_ReleaseRequestState	Завършено е съхраняването на състоянието на заявката
Application_UpdateRequestCache	Файла,получен при компилиране-то е готов за запис в кеш паметта
Application_EndRequest	Последно генерирано събитие когато изпълнението на приложението е приключило.

Освен това файлът Global.asax може да съдържа следните събитийни процедури, които не се изпълняват винаги в дадената по-горе последователност, а само при определени условия:

Session_OnEnd - Тази събитийна процедура се изпълнява когато някой потребител прекрати сесията си. Това става на практика когато потребителят не е изпратил заявка към ASP приложението до изтичането на определено време – по подразбиране това време е 20 мин. (може да бъде променено посредством атрибута timeout на обект Session)

Application_Error - Тази събитийна процедура се изпълнява когато се генерира изключение (run-time error) по време на изпълнение на кода на ASP.NET приложението. Събитийната процедура може да се използва за извеждане на извеждане на съобщение с информация за изключението, а също за пренасочване към друга страница (например чрез метода Redirect на обект Response)..

Ето един пример за използване събитийни процедури на файл Global.asax:

```
<%@ Application Language="C#" %>
<script runat="server">

   void Application_Start(object sender, EventArgs e)
   {
      Application(“visitors”)=0; //Инициализаиране на брояча
   }

   void Session_Start(object sender, EventArgs e)
   {
```

```
    // Code that runs when a new session is started
  Application("visitors")= Application("visitors")+1; //Отброяване
  }

  void Session_End(object sender, EventArgs e)
  {
    // Code that runs when a session ends.
  Application("visitors")= Application("visitors")-1; //Потребител
                                                      //напуска приложението
  }
</script>
```

Пример 43 Съдържание на файл Global.asax за отброяване на текущия брой посетители на ASP.NET приложението.

В даденият пример за отброяване на посетителите на сайта в даден момент се използва променлива на приложението *visitors*, на която се присвоява стойност 0 при първо стартиране на приложението, увеличава се с 1 при всяко начало на сесия и се намалява с единица при завършване на сесия. Тъй като променливите на приложението са достъпни от всяка ASP страница, то при включване на нов посетител чрез променливата *visitors* може да се изведе информация колко е броя включени посетители в момента. Както се вижда от горния пример в Global.asax файла не се използват ASP скриптови блокови ограничители (<% и %>). Кодът на процедурите се включва в контейнерния етикет <script> </script> , като чрез атрибут language се указва използваният скриптов език, а с параметър runat="server" се указва, че скриптът се изпълнява от страна на сървъра.

21.3. Декларации <object>

Декларациите <object> дават възможност за създаване на обекти с област на действие (scope) в рамките на сесията и в рамките на приложението. Етикетите <object> трябва да бъдат включени във файла Global.asax извън контейнерния етикет <script> </script>.

Синтаксис на етикета <object>:

```
<object runat="server" scope="scope" id="id"
{progid="progID"|classid="classID"}>
....
</object>
```

където:

scope - задава областта на действие на обекта (може да има стойност Session or Application)

id - задава уникален идентификатор на обекта

ProgID - Идентификатор, асоцииран с идентификатор на класа (class id). Задава се във формат [Vendor.]Component[.Version]

ClassID - Задава уникален идентификатор за COM класа на обекта.

Забележка: При деклариране на обект с етикета < object > трябва да бъде използван или атрибута ProgID or ClassID (но не и двата едновременно)

Примери за деклариране на обекти

```
<object runat="server" scope="session" id="MyAd"
progid="MSWC.AdRotator">
</object>
```

Пример 44 Деклариране на обект с област на действие в рамките на сесия

```
<object runat="server" scope="application" id="MyConnection"
classid="Clsid:8AD3067A-B3FC-11CF-A560-00A0C9081C21">
</object>
```

Пример 45 Деклариране на обект с област на действие в рамките на приложението

Декларираните обекти могат да се използват във всички модули на ASP приложението. При област на действие scope=”Session” ще бъдат създавани отделни екземпляри на обекти за всяка сесия

21.4. Декларации TypeLibrary

Декларациите TypeLibrary се включват във файла Global.asax посредством етикет METADATA . TypeLibrary се използва като контейнер на съдържанието на DLL файл, съответстващ на зададен COM обект. Включването на декларация TypeLibrary за даден обект във файла Global.asax дава възможност за достъп до константите на COM обекта, което може например да се използва от ASP приложението за

извеждане на по-детайлна информация за грешките, генерирани от този обект. От казаното следва, че има смисъл да се включва декларация TypeLibrary само COM обекти, които декларират собствени типове данни..

Синтаксис:

```
<!--METADATA TYPE="TypeLib"
file="filename"
uuid="typelibraryuuid"
version="versionnumber"
lcid="localeid"
-->
```

където:

File - Задава абсолютен път до библиотеката от типове. Не може да се използва съвместно с атрибут uuid.

uuid - Задава уникален идентификатор на библиотеката от типове. Не може да се използва съвместно с атрибут file .

version - (незадължителен). Използва се за задаване на версията. Ако зададената версия не бъде открита, се използва последната по номер.

localeid - (незадължителен). Задава локалният идентификатор, който ще бъде използван за библиотеката.

Забележка: Етикетите METADATA могат да бъдат включени навсякъде във файла Global.asax (и вътре и извън <script> блоковете). Препоръчва се обаче включването им в началото на файла.

22. Операции с обекти от файловата система

Файловата система (ФС) е структура, състояща се от директории и файлове (обекти на ФС), която организира съхраняването и достъпа до данните върху дисковите запомнящи устройства на компютъра. На практика цялото програмно обезпечение и всички локални данни, които се използват от компютъра се съхраняват във файловете на ФС. Поради това всеки език за програмиране и всяка среда за разработване на програмни продукти (с малки изключения*) предоставят средства за операции с обекти на ФС.

В .NET Framework именното пространство System.IO съдържа класовете File и Directory, чрез които .NET Framework предоставя възможност за операции с обекти от файловата система на компютъра. Тъй като методите на тези класове са статични, те се извикват непосредствено чрез името на класа, а не чрез обект, създаден от класа чрез конструктор. .NET Framework обаче предоставя и класове FileInfo и DirectoryInfo с аналогични по предназначение методи, които не са статични и се извикват чрез обект, създаден като екземпляр на съответния клас.

22.1. Класът File

Класът File се съдържа в именното пространство System.IO. Класът предоставя набор от статични методи за основните операции: (копиране, изтриване, преместване, отваряне) с файлове от локалната файлова система. Тъй като методите са статични, извикването им става чрез името на класа. .NET Framework предоставя и нестатичен клас FileInfo с аналогични по предназначение методи, достъпът до които се осъществява чрез обект, създаден с конструктора на класа.

Декларация:
public static class File

Методи на класа File

* Например в Java аплетите, от съображения за сигурност, по подразбиране е забранен достъпът до локалната файлова система.

Класът File предоставя голям набор от методи, който е подходящо да се разглеждат по групи, според тяхното предназначение

22.1.1. Методи за копиране, изтриване, преместване, преименуване

Copy	Копира съществуващ файл в нов файл по зададени спецификации. При overwrite=true се разрешава надписване. Декларация: (C#) public static void Copy(source,destination, overwrite)
Delete	Изтрива файл по зададена спецификация. Ако файлът не съществува, не се генерира изключение Декларация: (C#) public static void Delete(string path)
Move	Премества файл с възможност за променяне и на името на файла Декларация: (C#) public static void Move(source,destination)
Replace	Замества съдържанието на destination файл със съдържанието на source файл като записва резервно копие backup на source файла. Декларация: (C#) public static void Replace(source,destination, backup)

22.1.2. Методи за получаване на атрибути на файл

Класът File предоставя голям набор от Get методи, които връщат различни атрибути на файлове. Това са Get методите:

GetAttributes , GetCreationTime , GetCreationTimeUtc , GetLastAccessTime , GetLastAccessTimeUtc , GetLastWriteTime , GetLastWriteTimeUtc

и Set методите:

SetAccessControl , SetAttributes , SetCreationTime , SetCreationTimeUtc , SetLastAccessTime , SetLastAccessTimeUtc , SetLastWriteTime , SetLastWriteTimeUtc

Get методите ролучават като параметър спецификацията на файла и връщат стойността на параметъра на файла. Set методите ролучават като

първи параметър спецификацията на файла и като втори параметър новата стойност на параметъра на файла.

22.1.3. Методи за четене на съдържанието на файл

Отваряне на файл – метод Open

Методът отваря файл и връща обект FileStream, който предоставя методи за операции със съдържанието на файла.

Декларация: (C#)

```
public static FileStream Open(
   string path              // Спецификация на файла
   [,FileMode mode]     // Режим на отваряне (създаване,презапис)
   [,FileAccess access]  // Режим на достъп (четене, запис)
   [,FileShare share]      //Режим на поделяне с други нишки
)
```

Пример:

```
FileStream fs;
using (fs = File.Open(path, FileMode.Open, FileAccess.Read,
FileShare.None))
{
                Label1.Text=fs.ReadToEnd();
                fs.Close();
}
```

Пример 46 Отваряне на файл и извеждане на съдържанието му в обект Label

В примера файла се отваря за четене. Методът Open връща обект FileStream, чрез който се прочита съдържанието на файла и се присвоява на атрибута Text на обект Label1,

Отваряне на съществуващ файл за четене - метод OpenRead

Методът е подобен на Open, но само с един параметър (файловата спецификация), тъй като отваря файла само за четене.

Декларация: (C#)

```
public static FileStream OpenRead (string path)
```

Отваряне на текстов файл с UTF-8 кодиране за четене - Метод OpenText

Методът отваря текстов файл и връща обект StreamReader, който предоставя методи за операции със съдържанието на файла.
Декларация: (C#)
public static StreamReader OpenText (string path)

Четене на двоичен файл в масив – метод ReadAllBytes

Методът връща двоичен масив със съдържанието на файла и затваря файла след прочитането на съдържанието му.
Декларация: (C#)
public static byte[] ReadAllBytes (string path)

Четене на текстов файл в масив – метод ReadAllLines

Методът прочита текстов файл връща масив от тип string, в който последователно са записани прочетените текстови редове от файла. След прочитането файлът се затваря. Незадължителния параметър *encoding* задава използваното кодиране.
Декларация: (C#)
public static string[] ReadAllLines (string path [, Encoding encoding])

Четене на текстов файл във символен низ – метод ReadAllText

Методът е аналогичен на ReadAllLines с тази разлика, че връща низ (а не масив), в който е записано цялото съдържание на прочетения текстов файл.
Декларация: (C#)
public static string ReadAllText (string path [, Encoding encoding])
Пример:

```
using System.IO; //Добавяне на именното пространство System.IO

   protected void ReadFile_Click(object sender, EventArgs e)
   {
      string path = "C:\\Autoexec.bat";
      try
```

```
    {
       TextBox1.Text=File.ReadAllText(path);
    }
    catch (Exception ex)
    {
       Response.Write("<BR>Error=" + ex.Message);
    }

 }
```

Пример 47 Прочитане и извеждане в текстова кутия на съдържанието на текстов файл

В примера е показана събитийна процедура на бутон от ASP.NET модул, която прочита съдържанието на файла Autoexec.bat и го извежда с многотедовата текстова кутия TextBox1. Инструкцията за четене на файла, която използва метода ReadAllText на обекта File е включена в конструкция try-catch за прихващане на изключения. В примера се прихваща само базовото изключение Exception, но конструкцията try-catch дава възможност и за по-детайлизирано обработване на различните типове изключения. Например преди прихващане на Exception може да се прихваща FileNotFoundException, изключение, което се генерира ако файлът не съществува.

22.1.4. Методи за запис във файл

Отваряне на съществуващ файл за запис – метод OpenWrite

Методът отваря съществуващ файл за запис по зададена пълна или относителна (спрямо текущата работна директория) спецификация. Методът връща обект FileStream, който предоставя методи за запис във файла. Ако файлът не съществува се генерира изключение FileNotFoundException.
Декларация: (C#)
public static FileStream OpenWrite(string path)

Запис на двоични данни във файл – метод WriteAllBytes

Създава нов файл, записва в него съдържанието на зададен байтов масив и затваря файла. Ако файлът съществува, той се надписва.
Декларация: (C#)
public static void WriteAllBytes(string path, byte[] bytes)

Запис на символен масив във файл – метод WriteAllLines

Създава нов файл, записва в него елементите на символен масив като текстови редове и затваря файла. Ако файлът съществува, той се надписва. Незадължителният параметър encoding задава кодирането на текста.
Декларация: (C#)

```
public static void WriteAllLines(
   string path,              //Спецификация на файла
   string[] contents         //символен масив
   [,Encoding encoding]      //кодиране
)
```

Запис на символен низ във файл – метод WriteAllText

Методът е аналогичен на WriteAllLines с тази разлика, че записва във файла съдържанието на символен низ, а не на символен масив.
Декларация: (C#)

```
public static void WriteAllText (
   string path,              //Спецификация на файла
   string contents           //символен низ
   [,Encoding encoding]      //кодиране
)
```

Пример:

```
using System.IO; //Добавяне на именното пространство System.IO

   protected void Button1_Click(object sender, EventArgs e)
   {
      string path = Server.MapPath(Request.ApplicationPath);
      try
      {
         File.WriteAllText(path+"/proba.txt", TextBox1.Text);
      }
      catch (Exception ex)
```

```
        {
           Response.Write("<BR>Error=" + ex.Message);
        }
    }
```

Пример 48 Запис на съдържанието на текстова кутия във файл

В примера е показана събитийна процедура на бутон от ASP.NET модул, която записва съдържанието на текстова кутия TextBox1 във файл proba.txt в главната директория на ASP.NET приложението. Инструкцията за запис, която използва метода WriteAllText на обекта File е включена в конструкция try-catch за прихващане на изключения. Обърнете внимания на получаването в променливата path на спецификацията на директорията на ASP.NET модула. Атрибутът ApplicationPath на обект Request връща относителна спецификация, която се предава като параметър на метода MapPath на обект Server, който връща низ с пълната спецификация.

Добавяне на символен низ към текстов файл – метод AppendAllText

Методът е подобен на WriteAllText с тази разлика, че ако файлът съществува, добавя в края на файла съдържанието на символния низ. Ако файлът не съществува, той се създава и низът се записва в него, т.е. в тоя случай методът изпълнява просто запис както метода WriteAllText.

Декларация: (C#)

```
public static void AppendAllText (
   string path,              //Спецификация на файла
   string contents           //символен низ
   [,Encoding encoding]          //кодиране
)
```

22.2. Класът Directory

Класът предоставя набор от статични методи за основните операции: (създаване, изтриване, преместване) с директории от локалната файлова система. Класът не може да бъде наследяван.

Декларация: (C#)

```
public static class Directory
```

Методи на класа Directory

Класът Directory е деклариран в именното пространство System.IO. Той притежава значително по-малък набор от методи в сравнение с класа File, тъй като липсват методи за операции със съдържанието на директории, аналогични на тези за операции с файлове. По-долу са разгледани основните методи на класа. Тъй като методите са статични, извикването им става чрез името на класа. .NET Framework предоставя и нестатичен клас DirectoryInfo с аналогични по предназначение методи, достъпът до които се осъществява чрез обект, създаден с конструктора на класа.

22.2.1. Създаване на директории - метод CreateDirectory

Създава директория и поддиректории по зададен път. Методът връща обект от клас DirectoryInfo за създадената директория.
Декларация: (C#)
public static DirectoryInfo CreateDirectory (string path)
Пример:

```
using System.IO; //Добавяне на именното пространство System.IO


   protected void Button1_Click(object sender, EventArgs e)
   {
      try
      {
         Directory.CreateDirectory("C:\\D1\\D11");
      }
      catch (Exception ex)
      {
         Response.Write("Error=" + ex.Message);
      }
   }
```

Пример 49 Създаване на директория с метода CreateDirectory на клас Directory

В примера се създава поддиректория D11 на директория D1 на устройство C чрез метода CreateDirectory. Ако директория D1 не

съществува, тя също се създадава. Така с едно извикване на метода могат да бъдат създадени няколко вложени една в друга директории.

22.2.2. Изтриване на директории - метод Delete

Методът изтрива зададена директория по зададена спецификация в параметъра path. Спецификацията може да бъде пълна или относителна спрямо текущата работна директория. (може да бъде получена чрез метода GetCurrentDirectory) Ако незадължителният втори параметър recursive има стойност true, заедно с директорията се изтриват всички съдържащи се поддиректории и файлове.
Декларация: (C#)

```
public static void Delete(string path [, bool recursive])
```

23. Операции с потоци от данни. Клас Stream

Потокът в контекста на програмираните технологии се определя като последователност от байтове. Пример за поток са данните, които се четат или записват във файл, които се обменят при комуникация между процеси и др. Класът Stream е базовият клас в именното пространство System.IO. Той се наследява от производните класове (FileStream, StreamReader, StreamWriter), които предоставят атрибути и методи за операции с потоци от данни
Класовете за потоци предоставят три основни операции:

1) Запис от поток. Записът е прехвърляне на данни от поток в структура от данни, например в масив от байтове.
2) Четене в поток. Четенето в поток е прехвърляне на данни от структура от данни в поток.
3) Някои типове потоци поддържат търсене. Търсенето включва проверка за удовлетворяване на критерий и преместване на указателя на потока на съответната позиция. Търсенето е свързано с поддържането на информация за текущата позиция на курсора. Ако даден тип поток (напр. мрежовия поток) не поддържа концепцията за курсор, той съответно не притежава и операция търсене.

Базовият клас Stream притежава методи, които предоставят информация дали класът поддържа изброените по-горе операции. Това са методите CanRead, CanWrite и CanSeek. Тези методи се предефинират от класовете- наследници и предоставят информация кои от трите базови операции се поддържат от класа-наследник на Stream.

Методите Read и Write се предоставят във версии, с които може да се четат и записват данни в различни формати. Потоците, които поддържат търсене предоставят методи Seek и SetLength и атрибути Position и Length за променяне на текущата позиция на курсора и дължината на потока.

Някои видове потоци извършват буфериране на данните с цел повишаване на производителността на операциите. Потоците, извършващи буфериране, предоставят метод Flush, които изчиства буфера за данни и записва съдържащите се в него данни в приемника. При изпълнение на метода Close на обекта Stream автоматично се извиква метода Flush и освобождава системните ресурси, заделени за потока. Класът BufferedStream, който поддържа буфериране, може да се използва като обвивка (wrapper) на класове, които не поддържат буфериране, и по този начин им предоставя функционалността на буфериран поток.

Ако се нуждаете от поток, който не съхранява предистория, използвайте Null.

23.1. Четене на текстови данни. Клас StreamReader

Класът StreamReader (наследник на класа TextReader) предоставя методи за четене на символи от входно устройство или структура от данни в байтов поток в съответствие със зададено кодиране. Едно от основните му приложения е четене на данни от текстови файлове.

StreamReader по подразбиране използва UTF-8 кодиране ако не е зададено друго подразбиращо се кодиране в конкретната система. UTF-8 кодирането позволява четене на Unicode-кодирани символи съобразено с версията на операционната система.

По подразбиране StreamReader не е гарантирано безопасен при използване в паралелни нишки. При многонишкови приложения се препоръчва да се използва обвивката TextReader.Synchronized.

23.1.1. Как да създадем обект от клас StreamReader

Класът StreamReader предоставя 10 версии на конструктор. Обобщената декларация на конструктор с един задължителен и три незадължителни параметри има вида:

```
public StreamReader (
  string path OR Stream stream
  [,Encoding encoding]
  [,bool detectEncodingFromByteOrderMarks]
  [,int bufferSize]}
```

където:

path	Задава и връща спецификацията на файла, чието съдържание ще се чете чрез StreamReader.
stream	Задава и връща обект Stream, който ще се използва от StreamReader. Само един от двата параметъра path и stream трябва да бъде включен в конструктора.
encoding	Задава използваното кодиране.
Detect Encoding From ByteOrderMarks	Определя дали да се четат началните байтове от файла като маркер за използваното кодиране.
bufferSize	Задава минималния размер на използвания буфер в брой 16-битови кодове на символи. Ако зададената стойност е по-малка от минимално допустимата (128) се използва буфер минималният размер 128.

23.1.2. Атрибути на класа StreamReader

BaseStream	Връща обект FileStream, който представя базовия поток, над който се създава потока на StreamReader.
CurrentEncoding	Връща кодирането, което използва обекта StreamReader.
EndOfStream	(тип bool) Връща стойност, която показва дали указателя на StreamReader е в края на низа.

23.1.3. Методи на класа StreamReader

Close	Затваря потока, асоцииран със StreamReader и освобождава използваните ресурси.
DiscardBufferedData	Разрешава на обекта StreamReader да анулира данните в потока.
Peek	Прочита текущия символ, но без да премества указателя.
Read	Прочита зададен брой символи от потока.
ReadBlock	Прочита зададен брой символи от потока в буфер от зададен начален индекс. (Насលден от класа TextReader.)
ReadLine	Прочита символите до края на текущия ред.
ReadToEnd	Прочита цялото съдържание на символния поток.
Synchronized	Създава и връща нишково безопасна обвивка на класа TextReader. (методът се наследява от класа TextReader.)

Четене на последователност от символи - метод Read (клас StreamReader)

Методът чете следващия символ или последователност от символи от входния поток. Методът има две версии:

Read ()	Прочита един символ от входния поток и премества указателя за четене на следващия символ.
Read (Char[] buffer, Int32 number, Int32 buffer_index)	Прочита зададен брой символи от входния поток в буфер от зададен начален индекс буфера.

Даденият по-долу пример съдържа цикъл, чрез който се чете последователно по 5 символа от текстов файл, които се добавят към съдържанието на текстова кутия.

```
System.IO.StreamReader sr = new System.IO.StreamReader("C:\\
proba.txt")
        while(sr.Peek() >= 0){
            char[] c = new char[5];            //Масив за 5 символа
           sr.Read(c, 0, c.Length);            //Прочитане в масива
            TextBox1.Text+=new String(c);//Добавяне към
съдържанието на текстовата кутия
        }
        sr.Close();
```

Пример 50 Четене на текстов файл последователно по 5 символа

Четене по редове - метод ReadLine (клас StreamReader)

Методът ReadLine чете от потока символите до края на текущия ред и ги връща в символен низ.

Декларация: (C#)

public override string ReadLine ()

Даденият по-долу пример съдържа цикъл, чрез който се чете съдържанието на текстов файл ред по ред и добавя редовете към съдържанието на текстова кутия.

```
System.IO.StreamReader sr = new System.IO.StreamReader("C:\\proba.txt")
        while(sr.Peek() >= 0){
          TextBox1.Text+=sr.ReadLine();//Прочитане на ред от файла
        }
        sr.Close();
```

Пример 51 Четене на съдържанието на текстов файл ред по ред

Четене на цялото съдържание на текстов поток - Метод ReadToEnd (клас StreamReader)

Методът ReadToEnd чете всички символи от текущата позиция на курсора до края на текстовия поток. Ако потокът представя съдържанието на текстов файл, целият файл ще бъде прочетен.

Декларация (C#)

public override string ReadToEnd()

Даденият по-долу пример демонстрира използването на метода ReadToEnd за прочитане на съдържанието на текстов файл и извеждането му в текстова кутия:

```
    System.IO.StreamReader sr=new System.IO.StreamReader("C:\\
proba.txt");
    Label1.Text = sr.ReadToEnd();
    sr.Close();
```

Пример 52 Четене на съдържанието на текстов файл с метод ReadToEnd

Проверка за край на текстов поток - метод Peek и атрибут EndOfStream (клас StreamReader)

Методът Peek връща текущия символ от текстовия поток, но не премества указателя за четене. Методът връща -1 ако е бил достигнат края на файла. Това дава възможност за изпозлването на метода за проверяване на условие за достигане до края на потока при последователно четене на съдържанието. Това приложение на метода Peek е демонстрирано в дадените по-горе примери за методите Read и ReadLine.

Същия резултат можем да получим като използваме атрибута EndOfStream, който връща логичеста стойност true ако е достигнат края на потока и false в противен случай.

Декларация (C#)

```
public override int Peek()
public bool EndOfStream { get; }
```

```
System.IO.StreamReader sr = new System.IO.StreamReader("C:\\proba.txt")
      while(!sr.EndOfStream){
        TextBox1.Text+=sr.ReadLine();//Прочитане на ред от файла
      }
      sr.Close();
```

Пример 53 Използване на метода EndOfStream за проверка за достигане до края на текстов файл при последователно четене

23.2. Запис на текстови данни. Клас StreamWriter

Класът StreamWriter (наследник на класа TextWriter) предоставя методи за запис на символи в байтов поток към изходно устройство или структура от данни в съответствие със зададено кодиране. Едно от основните му приложения е запис на данни в текстови файлове.

StreamWriter по подразбиране използва UTF-8 кодиране ако не е зададено друго подразбиращо се кодиране в конкретната система. UTF-8 кодирането позволява четене на Unicode-кодирани символи съобразено с версията на операционната система.

По подразбиране StreamReader не е гарантирано безопасен при използване в паралелни нишки. При многонишкови приложения се препоръчва да се използва обвивката TextReader.Synchronized.

Как да създадем обект от клас StreamWriter

Класът StreamWriter предоставя 7 версии на конструктор. Обобщената декларация на конструктор с един задължителен и три незадължителни параметри има вида:

Декларация (C#)

public StreamWriter (string path OR Stream stream [,bool append [,Encoding encoding [,int bufferSize]]])

където:

path:	Задава и връща спецификацията на файла, в който ще се записва чрез StreamWriter.
stream	Задава и връща обект Stream, който ще се използва от StreamWriter. Само един от двата параметъра path и stream трябва да бъде включен в конструктора.
append:	Определя дали данните ще бъдат доавени към файла. Ако файлът съществува и append има стойност false, файлът се надписва. Ако файлът съществува и append има стойност true, данните се добавят към файла. Ако файлът не съществува, той се създава.
encoding	Задава използваното кодиране.

Атрибути на класа StreamWriter

AutoFlush	Задава и връща стойност, която опеделя дали StreamWriter ще изпраща цялото съдържание на буфера си след всяко извикване на метод Write.
BaseStream	Връща обект FileStream, който представя базовия поток, над който се създава потока на StreamWriter.
Encoding	Връща обект Encoding, който представя кодирането, използвано при запис.
FormatProvider	Връща обекта, който контролира форматирането. (наследен от клас TextWriter.)
NewLine	Задава и връща терминатор, който се записва за преминаване на нов ред (наследен от клас TextWriter.)

Методи на класа StreamWriter

Close	Затваря потока, асоцииран със StreamWriter и освобождава използваните ресурси.
Flush	Изчиства всички буфери за текущия поток за запис и записва всички съхранени данни в приемника.
Synchronized	Създава и връща нишково безопасна обвивка на класа TextWriter. (методът се наследява от класа TextWriter)
Write	Записва низ в изходния поток.
WriteLine	Записва низ в изходния поток като добавя терминатор за край на ред. (методът се наследява от класа TextWriter.)

Запис на низ в изходния поток - method Write

Методът Write записва низ в изходния текстов поток. Методът има повече от 20 версии и може да бъде използван да записва даннот променливи от различни типове: String, Char, Char[], Boolean, Decimal, Int32, Int64, Single, Double, Object, При записа в потока данните се преобразуват в последователност от символи.
Повечето от методите имат декларация от вида:
Декларация (C#)
public override void Write (Data_Type value)

където Data_Type е типа на данните

Даденият по-голу пример демонстрира използването на метода Write за запис на символни низове в текстов файл.

```
    // Създаване на екземпляр на StreamWriter за запис във файл.
    // Конструкцията using затваря StreamWriter след изпълнението на блока
    string path = Server.MapPath(Request.ApplicationPath);
    using (System.IO.StreamWriter sw = new System.IO.StreamWriter(path+"/TestFile.txt"))
        {
            // Запис на символни низове във файла
            sw.Write("This is the ");
            sw.WriteLine("header for the file.");
            sw.WriteLine("-------------------");
            // Обекти, които имат символно представяне също могат да се записват.
            sw.Write("The date is: ");
            sw.WriteLine(DateTime.Now);
        }
```

Пример 54 Използване на метода Write за запис в текстов файл

Файлът се записва в директорията на ASP.NET приложението. Спецификацията на директорията се получава в променливата path чрез метода MapPath на обект Server.

Запис на текстов ред в изходния поток - method WriteLine

Методът WriteLine записва низ в изходния текстов поток като добавя в края терминатор за преминаване на нов ред. Методът има същите версии като метода Write. Дадения по горе пример демонстрира използването на метода за запис на низ с текущата дата и време, който се получава от атрибута Now на клас DateTime.

Запис на буферираните данни в текстов поток - метод Flush

Методът Flush изчиства всички буфери за обекта StreamWriter и записва буферираните данни в изходния поток.

Декларация (C#)

```
public override void Flush()
```

Ако атрибутът AutoFlush на обекта StreamWriter има стойност true, това означава, че информацията за използваното кодиране няма да бъде премахната от буфера на потока за запис, Това дава възможност при запис на нов символен низ да се използва вече зададеното кодиране.

24. Операции с изображения с ASP.NET

ASP.NET и .NET Framework предоставят възможности за операции с изображения, аналогични с тези, които са на разположение на на програмиста при създаването на Windows приложения, които се изпълняват на потребителския компютър. С това се преодолява едно от ограниченията на "класическата" ASP, която принуждаваше програмистите да използват за целта допълнителни компоненти като ASPImage and ASPPicture, които трябваше да бъдат инсталирани на WEB сървъра. В ASP.NET, цялата необходима функционалност за генериране на изображения и операции с тях е включена в класовете на .NET Framework.

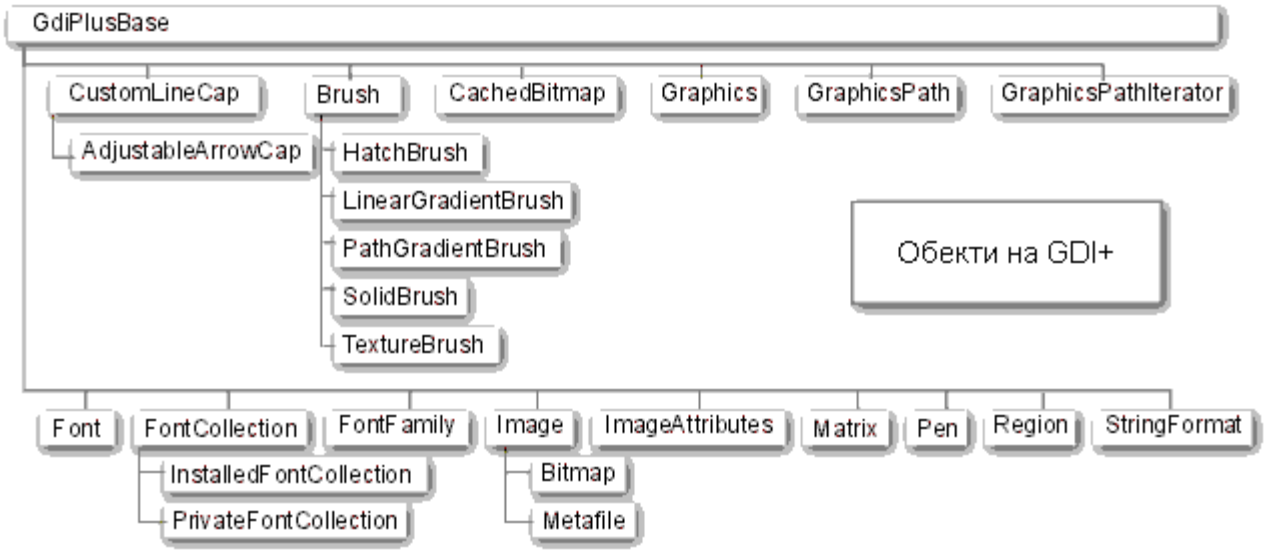
Най-често графичните възможности, които предоставя.NET Framework се използват в ASP.NET приложенията за динамично генериране на изображения, например:

- Генериране на графики и диаграми по данни, въведени от потребителя или записани в база от данни.
- Генериране на изображения със случаен текст (код за достъп) за включване в регистрационни страници, с което се предотвратява възможността за автоматична регистрация.

Пълното разглеждане на графичните инструменти на.NET Framework излиза извън рамките на настоящия курс. По-долу ще бъдат разгледани само някои от методите на основните класове, използвани при генериране на изображения и изчертаване на графични примитиви.

24.1. Именни пространства и класове на GDI+

Visual Studio.NET предоставя достъп до графичния интерфейс на Windows GDI+ посредством класове с управляем код, декларирани в именното пространство System.Drawing и негови подпространства. Именното пространство System.Drawing предоставя класове за достъп до базисните графични методи на интерфейса GDI+. Допълнтелни методи с разширена функционалност се предоставят от класовете на именните пространства System.Drawing.Drawing2D, System.Drawing.Imaging и System.Drawing.Text. По-долу е показана йерархията на класовете на GDI+.



Инструменти за изчертаване на графични примитиви

Основните инструменти за чертане се предоставят от именното пространство System.Drawing:
Brushes – четки за запълване на графични обекти с основните цветове. Класът предоставя атибути за голям брой цветове и чрез тях връща четка за избрания цвят. Например Brushes.Yellow връща четка с жълт цвят
SolidBrush – четка със зададен от потребителя цвят. Класът притежава конструктор с параметър обект от клас Color, с който се задава цвета на четката.
TextureBrush – четка за запълване с графично изображение (текстура).
LinearGradientBrush – четка за запълване с линейно изменение на цвета между два зададени цвята.
PathGradientBrush четка за запълване на сложен контур зададен с обект GraphicsPath с линейно изменение на цвета между два зададени цвята.

Pen - молив за изчертаване на линии. Класът притежава конструктор с параметър за цвят (Color или Brush) и параметър за ширина на линията.

Font – шрифт за извеждане на текст. В конструктора на класа се задават параметри на шрифта, например конструкторът Font("Arial", 12, FontStyle.Bold, GraphicsUnit.Point) задава име и размер на шрифта, стил Bold и графична единица Point.

24.2. Структури на GDI+ за параметри на графични елементи – Point, Rectangle, Size

Структурите на GDI+ представят параметри на графични елементи. Структурите могат да се използват като параметри на методи за изчертаване на геометричните фигури. GDI прдоставя следните класове за геометрични примитиви:

Point (точка) - представя подредена двойка от целочислени координати x и y на точка в двумерна област.

PointF (точка с реални координати) - представя подредена двойка от реални (тип float) координати x и y на точка в двумерна област.

Rectangle (правоъгълник) - представя подредена последователност от четири целочислени стойности, които задават началната координата и размера на правоъгълника.

RectangleF (правоъгълник с реални координати) - представя подредена последователност от четири реални (тип float) стойности, които задават началната координата и размера на правоъгълника.

Size - Съхранява подредена двойка от цели числа, които по правило представят размерите (дължина и височина) на правоъгълник.

SizeF - Съхранява подредена двойка от реални числа, които по правило представят размерите (дължина и височина) на правоъгълник.

24.3. Задаване на цвят в GDI+ - клас Color

Класът Color представя ARGB цвят в GDI+. Цветът се задава с четири компоненти:
A – непрозрачност (alpha канал)
R – яркост на червената съставяща на цвета.
G – яркост на зелената съставяща на цвета.
B – яркост на синята съставяща на цвета.
Стойността на всяка от съставящите се кодира с един байт, т.е. с целочислена стойност без знак в диапазона от 0 до 255.

Класът Color предоставя голям набор от атрибути, които връщат предварително дефинирани цветове. За създаване на цвят със зададени от потребителя стойности на компонентите A, R, G и B класът Color предоставя метода FromArgb. В дадения по-долу пример се създава

четка с бял непрозрачен цвят (напомняме, че компонентата Alpha задава ниво на непрозрачност от 0 до 255, като стойността 255 означава пълна непрозрачност).

```
SolidBrush whiteBr = new SolidBrush(Color.FromArgb(255, 255, 255,
255));
```

Пример 55 Създаване на непрозрачна четка с бял цвят чрез метода FromArgb

24.4. Създаване на битови карти – класове Bitmap и Image

Битовите карти представляват матрица от точки, която е основа на растерното изображение. VS.NET предоставя два класа за операции с растерни изображения – класът Image и класът Bitmap. Класовете Bitmap и Image предоставят набор от конструктори, посредством които могат да се създадат битмап карти със зададен размер и формат (брой битове на пиксел).Конструкторът (C#)

```
public Bitmap (
        int width,   //дължина
        int height,  //височина
        PixelFormat format      //формат на пикселите
)
```

създава празна битова карта със зададен размер и формат на представяне на пиксел.По-долу е даден пример за създаване на битова карта с формат 32 бита за пиксел

```
System.Drawing.Bitmap imgOutput = new System.Drawing.Bitmap(200,
  200, System.Drawing.Imaging.PixelFormat.Format32bppArgb);
```

Пример 56 Създаване на празна битова карта

Не трябва да се забравя, че създадената битова карта (обект Bitmap) е неинициализирана. Всички байтове в нея имат стойност 00, което за 32 битов ARGB формат означава черен цвят и пълна прозрачност за всички пиксели в картата. За да инициализираме битовата карта, можем изчертаем върху нея запълнен правоъгълник с нейния размер, като използваме четка с избран от нас цвят и прозрачност.

Класовете Bitmap и Image предоставят и метод, чрез който може да се създаде битова карта от графичен файл. Той може да ни бъде полезен когато искаме да дорисуваме предварително подготвено изображение-шаблон.

Декларация (C#)

```
public static Image FromFile (
      string filename //спецификация на файла
)
```

По-долу е даден пример за създаване на обект Image (който капсулира битова карта) от файл, записан в главната директория на ASP.NET приложението. Пълната спецификация на файла се получава чрез метода MapPath на обект Server.

```
System.Drawing.Image pattern_image =
System.Drawing.Image.FromFile(Server.MapPath("pattern.gif"));
```

Пример 57 Създаване на обект Image от графичен файл.

В примера е използван метода MapPath на обект Server за получаване на пълната файлова спецификация на файла pattern.gif.

24.5. Изчертаване на графични примитиви – клас Graphics

Един от основните класове в GDI+ е класът Graphics. Той предоставя повече от сто метода за изчертаване, графични операции и задаване на параметри на устройствата за извеждане. Може да се направи аналогия между класа Graphics и контекста на устройството за извеждане (Device Context) на по-старата версия - интерфейса GDI, и такава връзка действително съществува. От предоставените четири конструктора на класа Graphics два използват като параметър структура HDC. Основното различие е в променения програмен модел – използват се непосредствено методите на класа Graphics, а не не тези на класа CDC.

24.5.1. Създаване на обект Graphics

Класът Graphics предоставя няколко статични метода, които създават и връщат обект Graphics. Важно е да се запомни, че всеки обект Graphics се създава за определен обект, който извежда графика на дадено изходно устройство. Това е така, защото за изчертаването с методите на обекта Graphics е необходимо да се използват параметрите на изходното устройство. Така класът Graphics притежава методи, които създават обект Graphics чрез структура Hwnd за даден прозорец и

методи, които създават обект Graphics чрез структура Hdc (Handle to Device Context) на обект, който има графичен интерфейс. Тези методи не могат да се използват в ASP.NET, защото кодът на ASP.NET страниците се изпълнява не на потребителския компютър, а на WEB сървъра. В ASP.NET приложенията обект Graphics се създава за битова карта с метода FromImage. След изчертаването на изображението върху битовата карта, тя се изпраща в изходен поток към WEB браузъра, който я изобразява в прозореца си на потребителския компютър.
Декларация на метода FromImage (C#)

```
public static Graphics FromImage (
        Image image //битова карта, капсулирана в обект Image
)
```

След създаването на обект Graphics могат да се използват неговите методи за изчертаване.

24.5.2. Методи на обект Graphics за изчертаване на графични примитиви

Обектът Graphics притежава голям брой методи за изчертаване. Ще разгледаме само няколко от тях.

Изчертаване на линия (отсечка) – метод DrawLine

```
public void DrawLine (
        Pen pen,  //Молив за изчертаване (обект Pen)
        Point pt1,  //Обект Point за начална точка на отсечката
        Point pt2  // Обект Point за крайна точка на отсечката
)
```

Изчертаване на правоъгълник – метод DrawRectangle

```
public void DrawRectangle (
        Pen pen,  //Молив за изчертаване (обект Pen)
        Rectangle rect //Обект Rectangle, който задава координатите
                        //на правоъгълника
)
```

Изчертаване на текст – метод DrawRectangle

```
public void DrawString (
        string s,       //символен низ
        Font font,    //шрифт (обект Font)
        Brush brush,//четка, с която се изчертава текста
        PointF point //структура (обект) Point, която задава координатата на горния
```

//ляв връх на правоъгълника, в който се изчертава текста
)

Да разгледаме един пример на ASPX модул, който създава битова карта и обект Graphics за нея, изчертава правоъгълник и текст в него и изпраща изображението към потребителския браузър.

```
<script language="C#" runat="server">
protected void Page_Load(Object Src, EventArgs E){
// Създаване на битова карта (обект Bitmap)
   System.Drawing.Bitmap imgOutput = new System.Drawing.Bitmap(200,
60, System.Drawing.Imaging.PixelFormat.Format32bppArgb);
// Създаване на обект Graphics за битовата карта
   System.Drawing.Graphics g =
System.Drawing.Graphics.FromImage(imgOutput);
// Запълване на цялата битова карта с жълт цвят
   // Създаване на жълта четка.
   System.Drawing.SolidBrush yellowBrush = new
System.Drawing.SolidBrush(System.Drawing.Color.Yellow);
   // Създаване на структура Rectangle с размерите на Bitmap
   System.Drawing.Rectangle rect = new System.Drawing.Rectangle(0, 0,
200, 60);
   // Запълване на правоъгълника с цвят.
   g.FillRectangle(yellowBrush, rect);
   // Изчертаване на текст в правоъгълника с метода DrawString
   g.DrawString("Hello Drawing",
      new System.Drawing.Font("Arial", 12,
System.Drawing.FontStyle.Italic),
      new System.Drawing.SolidBrush(System.Drawing.Color.Blue),
      new System.Drawing.RectangleF(50, 20, 150, 20));
   Response.ContentType = "image/gif"; //задава тип на изпращаните
данни

imgOutput.Save(Response.OutputStream, ....System.Drawing.Imaging.Image
Format.Gif); //изпраща изображението
   //към потребителския браузър
}
</script>
```

Пример 58 Генериране на изображение от ASP.NET модул

Модулът съдържа само събитийната процедура Page_Load, включена в блок <script>. Чрез инструкцията Response.ContentType = "image/gif"; се задава тип на изпращаните данни: изображение в GIF формат. Изображението се изпраща към потребителския браузър чрез метода Save на обект Bitmap.

Ако модулът се заяви самостоятелно чрез името на .aspx файла, който съдържа кода на примера , в прозореца на браузъра ще се появи само генерираното изображение. Но освен това генерираното изображение може да се изведе в HTML документ или в друг .aspx файл, ако в него се включи HTML код за изображение, в който на атрибута SRC се присвои адреса на ASP.NET модула, както е показано в следващия пример:

```
<IMG SRC=”Draw_Image.aspx”>
```

Пример 59 Извеждане в HTML документ на изображение, генерирано от .ASP.NET модул

25. Потребителски контроли

Потребителският контрол (User control) е вид ASP.NET страница, която се използва като сървърен контрол от друга ASP.NET страница. Потребителските контроли предоставят на WEB приложенията прост и лесен за използване потребителски интерфейс (UI). За разлика от WEB сървърните контроли, които са компилирани в .dll файл и не могат да се редактират от разработчика на ASP.NET приложения, потребителските контроли могат да се променят по всяко време, тъй като се съдържат във файлове в самото приложение. Когато ASP.NET страницата се компилира, съдържащите се в нея потребителските контроли се компилират и се записват в кеш паметта на сървъра.

Предимства на потребителските контроли:

1. Потребителските контроли капсулират дефинициите на променливи и методи в собствено обектно пространство. Това премахва възможността от конфликти с дефиниции в ASP.NET страницата, в която е включен потребителския контрол.
2. Един потребителски контрол може да бъде използван многократно в една ASP.NET страница без това да предизвика конфликт.
3. Потребителският контрол може да бъде написан език за програмиране, различен от този, на който е написан кода на ASP.NET страницата.

Главното различие между кода, асоцииран с ASP.NET страницата и потребителските контроли е това, че освен код (декларация на клас и методи) потребителските контроли притежават и графичен интерфейс.

25.1. Създаване на потребителски контроли

Потребителския контрол е вид ASP.NET страница, която се използва като сървърен контрол. Файлът на потребителският контрол има .ascx разширение. Файловете с това разширение съдържат програми, които не се изпълняват като самостоятелни ASP.NET модули.

Потребителските контроли съдържат HTML и програмен код, но тъй като се включват като елементи на съществуващи страници, те не

съдържат <head> , <body> или <form> етикети. Тези тагове се съдържат в ASP.NET страницата, която е контейнер за контрола. Потребителските контроли участват в жизнения цикъл на ASP.NET страницата като генерират собствени събития и капсулират зададена част от кода, който се изпълнява. Например потребителският контрол може да генерира собствен отговор “postback” в неговата събитийна процедура Page_Load.

В среда на Visual Studio NET можете да добавите потребителски Web контрол към ASP.NET WEB сайт чрез функцията Website → Add New Item като от диалоговата кутия се избере Web User Control. Даденият по-долу код е HTML частта от потребителски контрол, който включва текстова кутия и бутон.

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeFile="WebUserControl.ascx.cs" Inherits="WebUserControl" %>
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
<asp:Button ID="Button1" runat="server" OnClick="Button1_Click"
Text="Button" />
```

Пример 60 Интерфейсна (.ascx) страница на потребителски контрол с директива **@ Control** за референция към кодова страница

За предоставяне на достъп до стойностите на контролите в страницата-контейнер, трябва да се създадат атрибути със спецификация public. Например дадената по-долу кодова страница на потребителския контрол от горния пример създава атрибут с име CtrlText, който има съдържанието на атрибута **Text** на текстовата кутия в контрола:

```
using System;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class WebUserControl : System.Web.UI.UserControl
{
   private string textbox_text; //Поле с локален достъп
   public string CtrlText //Атрибут, достъпен в контейнера на контрола
   {
      // Връща стойност от локалната променлива textbox_text.
```

```
        get
        {
            textbox_text = TextBox1.Text;
            return textbox_text;
        }
        // Записва стойност в textbox_text и променя текста в текстовата
кутия
        set
        {
            textbox_text = value;
            TextBox1.Text = textbox_text;
        }
    }
}
```

Пример 61 Кодова страница на потребителски контрол (C#) с декларация на атрибут.

Всички публични променливи, атрибути и методи на потребителските контроли са достъпни в страницата-контейнер.

25.2. Директивата @Control

Освен **.ascx** страница потребителските контроли могат да имат и кодова страница (code behind page). Директивата **@Control** се използва за създаване на референция към кодовата страница на потребителския контрол. Освен това директивата може за задава специфични атрибути на контрола, които се използват от парсера и компилатора на ASP.NET приложението. Тази директива може да бъде използвана само за потребителските контроли. Синтаксисът и е следния:

```
<% @Control име_на_атрибут=”стойност_на_атрибут” %>
```

като директивата може да съдържа повече от една двойка име=”стойност” с разделител празен интервал. Пример за директива @Control е даден по-горе в .ascx страницата на потребителски контрол

Директивата Control поддържа същите атрибути както директивата @Page с изключение на атрибутите AspCompat и Trace. За да се разреши трасирането трябва да се добави атрибут Trace в директивата @Page на

ASPX страницата, която е контейнер за контрола. В една интерфейсна (.ascx) страница може да се включи само една директива Control.

25.3. Използване на потребителски контроли в .aspx страница

Потребителските контроли се регистрират в .aspx страницата посредством директивата **@Register,** както е показано в следния пример:

```
<%@ Register Src="WebUserControl.ascx" TagName="WebUserControl"
    TagPrefix="uc1" %>
```

Пример 62 Директива @Register за регистриране на потребителски контрол

Директивата създава асоциация между страницата на потребителския контрол от една страна и префикса и етикета, с който контрола се включва в .aspx страница от друга. След регистрирането потребителския контрол в ASP.NET модула той може да бъде използван в него един или повече пъти, както обикновените сървърни контроли. В етикета, с който екземпляр на контрола се включва в .aspx страницата, както за сървърните контроли, трябва задължително да се включи атрибутът runat=”server”. Дадения по-долу пример съдържа код, с който в .aspx страница се включва екземпляр на потребителския контрол **WebUserControl**.

```
<uc1:WebUserControl ID="WebUserControl1" runat="server" />
```

Пример 63 Код за включване на потребителски контрол в .aspx страница

Атрибути на директивата **@Register.**

TagPrefix - определя уникално обектно пространство (namespace) за потребителския контрол., което дава възможност за диференцирането на различни потребителски контроли с едно и също име.

TagName – задава уникален етикет на потребителския контрол.

Src – задава виртуален път до .ascx файла на потребителския контрол.

Когато ASP.NET страницата-контейнер се зареди, компилаторът за реално време компилира файла на потребителския контрол и така прави възможно създаването на екземпляри на контрола в ASP.NET модула. Достъпът до атрибутите и методите на екземплярите на потребителските

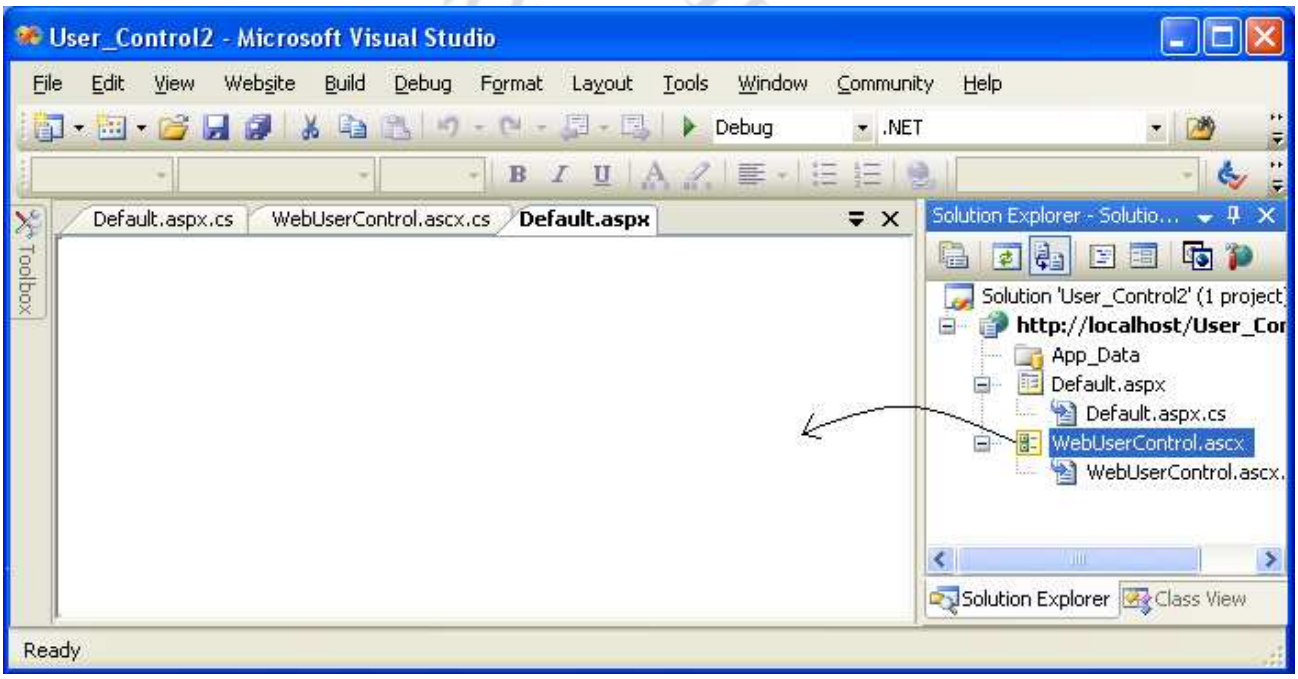
контроли се извършва (както за всички останали обекти в ASP.NET модула) посредством идентификатора на обекта, например

```
WebUserControl1.CtrlText = “Enter your email address here”;
```

Пример 64 Достъп до атрибут на потребителски контрол чрез идектификатора на контрола

25.4. Добавяне на контрол към .aspx страница посредством дизайнера на Visual Studio.NET

Добавянето на потребителски контрол към Web форма при използване на дизайнера на Visual Studio.NET се извършва чрез просто провлачване на иконата на контрола от прозореца Solution Explorer върху Web формата. Желателно е още при поставянето на контрола той да се включи в контейнерен обект на ASP.NET страницата, чрез който котролът да бъде позициониран, например панел или таблица.



Фиг. 3 Добавяне на потребителски контрол към .aspx страница чрез провлачване на иконата му от прозореца на Solution Explorer

Дизайнерът автоматично добавя директива @ Register и етикет за добавяне на контрола в страницата. Включеният по този начин контрол става част от страницата и се изобразява в нея. Също така атрибутите и

методите на контрола, декларирани с модификатор public, и събитията, които той генерира, стават програмно достъпни в кода на страницата, както е показано в предния пример.

25.5. Създаване на събития на потребителските конроли

Събитията на потребителските контроли дават възможност на ASP.NET модула, който ги използва, да реагира на събития, които възникват вътре в потребителския контрол. Например ако потребителски контрол съдържа бутон (както в дадения по-горе пример) то щракването върху бутона по подразбиране не генерира събитие в ASP.NET модула, в който е включен потребителския контрол. За да може ASP.NET модула да реагира на щракването върху бутона в потребителския контрол е необходимо събитието, което се генерира вътре в контрола и предизвиква изпълнението на събитийна процедура, включена в кода на контрола, да извиква събитийна процедура в ASP.NET модула. В описанието на събитийния модел на .NET Framework в т.7 беше обяснено, че генерирането на събитие, в резултат на което се изпълняват една или няколко събитийни процедури е всъщност извикване на тези функции (методи на обекти) чрез предварително записани в обекта от клас event функционални указатели (делегати). Така че, задачата за генерирането на събитие от бутон в потребителския контрол се свежда до:

- Деклариране в контрола на тип делегат (манипулатор на събитие) за събитийна процедура, която ще се извиква при генериране на събитието.
  Пример:

```
public delegate void EventHandler(Object obj, EventArgs e);
```

- Деклариране на събитие (атрибут от типа делегат с модификатор event) в потребителския контрол. Пример:

```
public event EventHandler ButtonClick;
```

- Създаване на обект (инстанция) на контрола.
- Запис в неговия атрибут event на обект делегат за метод (събитийна процедура), декларирана в ASP.NET модула. Конструкторът на класа делегат получава като параметър името на събитийната процедура.
  Пример:

(Обърнете внимание на това, че достъпът до типа EventHandler става чрез типа WebUserControl, а не чрез обектната променлива WebUserControl1 на потребителския контрол.)

```
WebUserControl1.ButtonClick +=
    new WebUserControl.EventHandler(WebUserControl1_Click);
```

- Извикването на тази процедура чрез инструкция (с използването на обект event), включена в събитийната процедура Click на бутона в кода на контрола.
  Пример:

```
ButtonClick(this, new EventArgs());
```

Както се вижда от това описание, последователността от операции е същата, както в дадения в т.7 пример за добавяне на събитие и събитийна процедура в ASP.NET модул. Разликата е в това, че инструкцията за запис (или добавяне) на делегат EventHandler към събитието ButtonClick се изпълнява в кода на ASP.NET модула, а не в кода на потребителския контрол. В тая инструкция достъпа до типа делегат EventHandler и събитието ButtonClick става с идентификатора WebUserControl1 на контрола. Така при генерирането на събитието с инструкцията ButtonClick(this, new EventArgs()); се извиква събитийна процедура, която не е в кода на потребителския контрол, а в неговия контейнер - ASP.NET модула. По-долу е даден пълния код на кодовата страница на потребителски контрол, който генерира събитие при щракване върху бутона Button1, и кода на ASP.NET модул, който изпълнява събитийна процедура генерирането на събитието в потребителския контрол.

```
using System;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class WebUserControl : System.Web.UI.UserControl
{
    //Деклариране на тип делегат (манипулатор на събитие) EventHandler
    public delegate void EventHandler(Object obj, EventArgs e);
    //Деклариране на събитие с делегати от тип EventHandler
    public event EventHandler ButtonClick;
```

```
//Събитийна процедура на бутона в потребителския контрол
   protected void Button1_Click(object sender, EventArgs e)
   {
      if (ButtonClick != null)
      {
         ButtonClick(this, new EventArgs()); //Генериране на събитие
      }
   } // край на събитийната процедура
} // край на декларацията на класа
```

Пример 65 Кодова страница на потребителския контрол, който генерира събитие **ButtonClick**

```
using System;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class _Default : System.Web.UI.Page
{
   protected void Page_Init(object sender, EventArgs e)
   {
      WebUserControl1.ButtonClick +=
      new WebUserControl.EventHandler(WebUserControl1_Click);

   }
   protected void WebUserControl1_Click(object sender, EventArgs e)
   {
      Label1.Text += "<BR>WebUserControl1 Button Clicked";
   }
}
```

Пример 66 Кодова страница на ASP.NET модул, който изпълнява събитийна процедура WebUserControl1_Click при щракване с мишката върху бутона в потребителския контрол.

За да тествате примера създайте проект за Web сайт и добавете към него потребителски контрол чрез функцията Add New Item от менюто Website. Изпълнете следната последователност от операции:

1. Изберете страницата WebUserControl.ascx на контрола. Изберете режим Design. От страницата Standard на прозореца Toolbox добавете бутон в страницата WebUserControl.ascx на контрола.
2. Изберете страницата Default.aspx на ASP.NET модула. Изберете режим Design. С провлачване на иконата на контрола от прозореца на Solution Explorer добавете инстанция на потребителския контрол WebUserControl в страницата Default.aspx на ASP.NET модула.
3. С провлачване от страницата Standard на прозореца Toolbox добавете в Default.aspx и обект Label.
4. Копирайте в кодовата страница на контрола WebUserControl.ascx.cs и в кодовата страница на ASP.NET модула дадените в примерите по-горе кодове.
5. Компилирайте и стартирайте изпълнението на ASP.NET приложението. В прозореца на браузъра ще се изведе страницата Default.aspx. При щракване с мишката върху бутона в страницата ще се изпълни събитийната процедура и ще се изведе съобщението “WebUserControl1 Button Clicked".

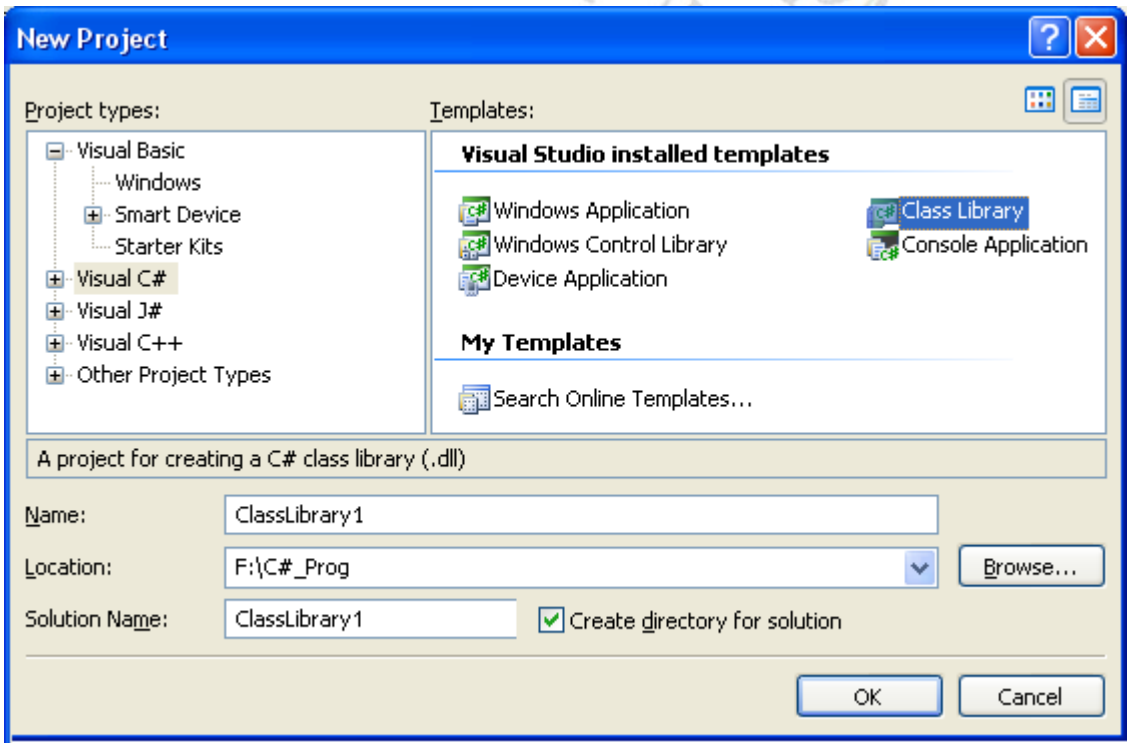
26. Създаване и използване на потребителски библиотеки

Динамично свързваните библиотеки .dll assembly (превежда се като “обединения”) са основните градивни елементи на приложенията в .NET Framework. За разлика от .dll библиотеките с неуправляем код обединенията се компилират до междинен език IL (идея, взаимствана от класовете на Java). Обединенията съдържат декларации на класове, от които приложението може да създаде обекти и използва функционалността, която те предоставят.

За динамично свързваните библиотеки (assembly) е полезно да се запомни следното:

- Обединенията са най-малките програмни елементи, които могат да бъдат заредени самостоятелно.
- Всички типове, декларирани в едно обединение имат една и съща версия
- Всички декларации на класове и типове в обединенията са включени в декларирани в него именни пространства.

По правило в едно обединение се декларира едно именно пространство, но се допуска и декларирането на повече от едно. Освен това едно обединение може да се съдържа в повече от един файл.



Фиг. 4 Създаване на проект за потребителска библиотека с Visual Studio .NET (от меню File→New→Project)

При създаване на проект за потребителска библиотека Visual Studio .NET генерира автоматично декларация на именно пространство и декларация на клас в него.

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ClassLibrary4
{
   public class Class1
   {
   }
}
```

Пример 67 Именно пространство и клас, декларирани автоматично от Visual Studio .NET

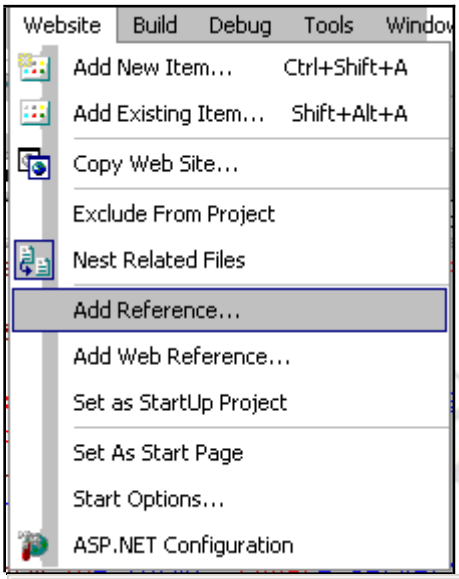
Програмистът може да добави необходимите му атрибути и методи към класа, както и други декларации на класове.

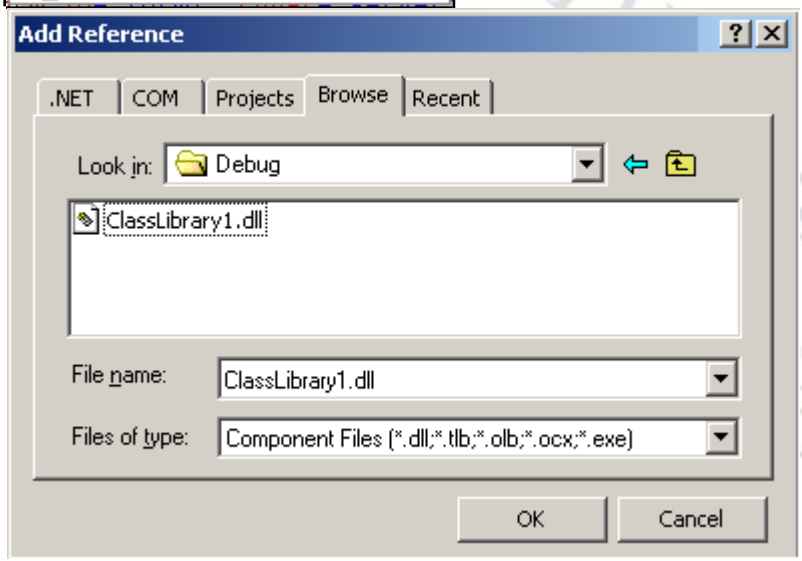
```
using System;
using System.Collections.Generic;
using System.Text;

namespace ClassLibrary4
{
   public class Class1
   {
        public string getAuthor( )
        {
                return “Peter”;
        }
   }
}
```

Пример 68 Деклариране метод в клас от потребителска библиотека

Включване на потребителска библиотека в ASP.NET приложение

За да се използва потребителска библиотека, тя трябва да бъде включена във Web сайта чрез функцията Reference на Visual Studio.NET. От менюто WebSite се избира функция Reference. Извежда се многостранична диалогова кутия, от която се избира





страницата “Browse”. Избира се файла на библиотеката (напр. ClassLibrary1.dll) и се потвърждава избора с бутона “OK”. Като резултат в прозореца “Solution Explorer”в директорията App Data се появява поддиректория Bin (ако вече не съществува), и в нея се добават файловете на библиотеката (напр. ClassLibrary1.dll и ClassLibrary1.pdb).

Библиотеките съдържат една или няколко декларации на класове, включени в общо пространство от имена (namespace). За по-удобно използване на класовете от библиотеката е желателно да включим именното пространство в кодовата страница на ASP.NET модула. По подразбиране името на файла на библиотеката съвпада с именното

пространство, така, че в нашия пример, ако използваме C# трябва да включим в началото на кодовата страница директивата

```
using ClassLibrary1;
```

За да използваме клас от библиотеката трябва да създадем обект – екземпляр на класа и чрез него можем да извикваме методите на класа. В нашия пример можем да изведем в прозореца на браузъра името на автора на библиотеката чрез следния код

```
Class1 myClass = new Class1();
Response.Write("Author " + myClass.getAuthor());
```

Регистриране на асембли в Global Assembly Cache

Learn how to register an assembly in the GAC.

1) Create an assembly key file

Use the sn.exe tool to create a key file:
sn -k StrongNameFile.snk

If your path environment variables aren't set, you'll have to go to the C:\ Program Files\Microsoft.NET\FrameworkSDK\Bin\ directory to run sn.exe)
The filename "StrongNameFile.snk" can be any name you want.

2) Edit your assembly
Now you have to add a tag which will link your assembly key to the assembly:

```
using System.Reflection;
[assembly:AssemblyKeyFile("StrongNameFile.snk")]
```

Normally this is done in the utility AssemblyInfo.cs (vb) file.

3)Add your assembly to the GAC

```
gacutil /i AssemblyFileName.dll
```

To uninstall this assembly from the GAC, use the command:

gacutil /u AssemblyFileName

4) (Opional) Add your assembly to machine.config

Locate the <assemblies> tag (for web apps,under <configuration>/<system.web>/<compilation>/<compilers>/<assemblies>)
Between the <assemblies> tags, enter:

```
<add assembly="AssemblyFileName, Version=0.0.0.0, Culture=neutral,
PublicKeyToken=5edf592a9c40680c" />
```

You can get the information for the assembly attribute, by running the gacutil /l command which will return a list of all the assemblies in the GAC. You will have to look for the one you just added and copy the entire line (less the Custom=XXX part at the end).

At this point, you will be able to place this directive in your aspx pages.

```
<%@Import Namespace="YourNamespace"%>
```

or, you can set a reference in the Visual Studio IDE as you would with other GAC Assemblies.

27. Разширяване на WEB контроли

.NET Framework дава възможност на разработчиците на ASP.NET приложения да разширяват функционалността на съществуващи WEB контроли, като им добавят допълнителни атрубути и методи. Този подход улеснява много проектирането на WEB контрол, тъй като новият контрол запазва цялата функционалност на базовия контрол, включително поддръжката в режим Дизайн,

За да се създаде контрол на базата на съществуващ WEB контрол е необходимо да се изпълни следната последователност от стъпки:

1) Създава се проект за потребителска библиотека WebControlLibrary (От менюто на VisualStudio File→New→Project→Class Library)..
2) Visual Studio.Net декларира именно пространство и клас на библиотеката. За да получим контрол, който разширява съществуващ WEB контрол, декларираме класа като наследник на WEB контрола.
3) Добавяме необходимите ни атрибути и методи.

По-долу е даден кода на контрол, който разширява WEB контрола TextBox

```
using System;
using System.Web.UI.WebControls;

namespace WebControlLibrary1
{
   /// <summary>
   /// Summary description for TextBoxA.
   /// </summary>
   public class TextBoxA : System.Web.UI.WebControls.TextBox
   {
      private string myprop;    //текстово поле
      public TextBoxA()         //конструктор на класа
      {

      }
// Допълнителен метод
      public void ToLower()
      {
         this.Text = this.Text.ToLower();
```

```
        }
// Допълнителен атрибут
        public string NewProp
        {
            get
            {
                if (myprop == null)
                    return "";
                else
                    return myprop;
            }
            set
            {
                myprop = value;
            }
        }
    } //end of class
}
```

Пример 69 Код на контрол, разширяващ WEB контрола TextBox

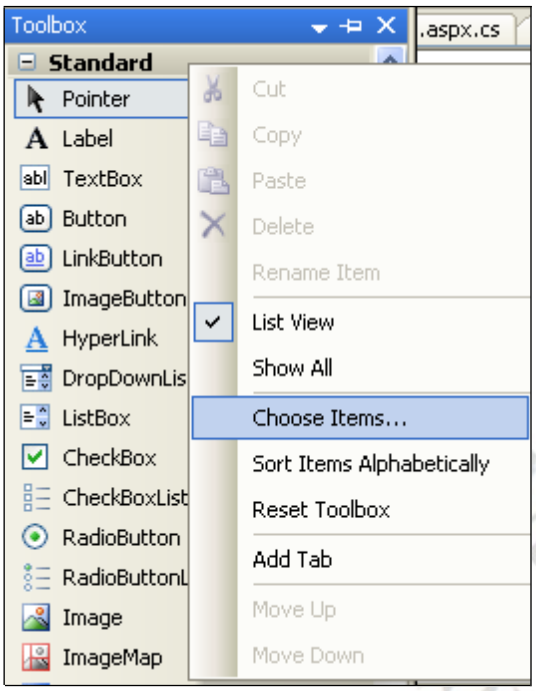
27.1. Добавяне на контрола към WEB страница

За да се добави разработения контрол към WEB страница е необходимо да се изпълнят няколко стъпки:

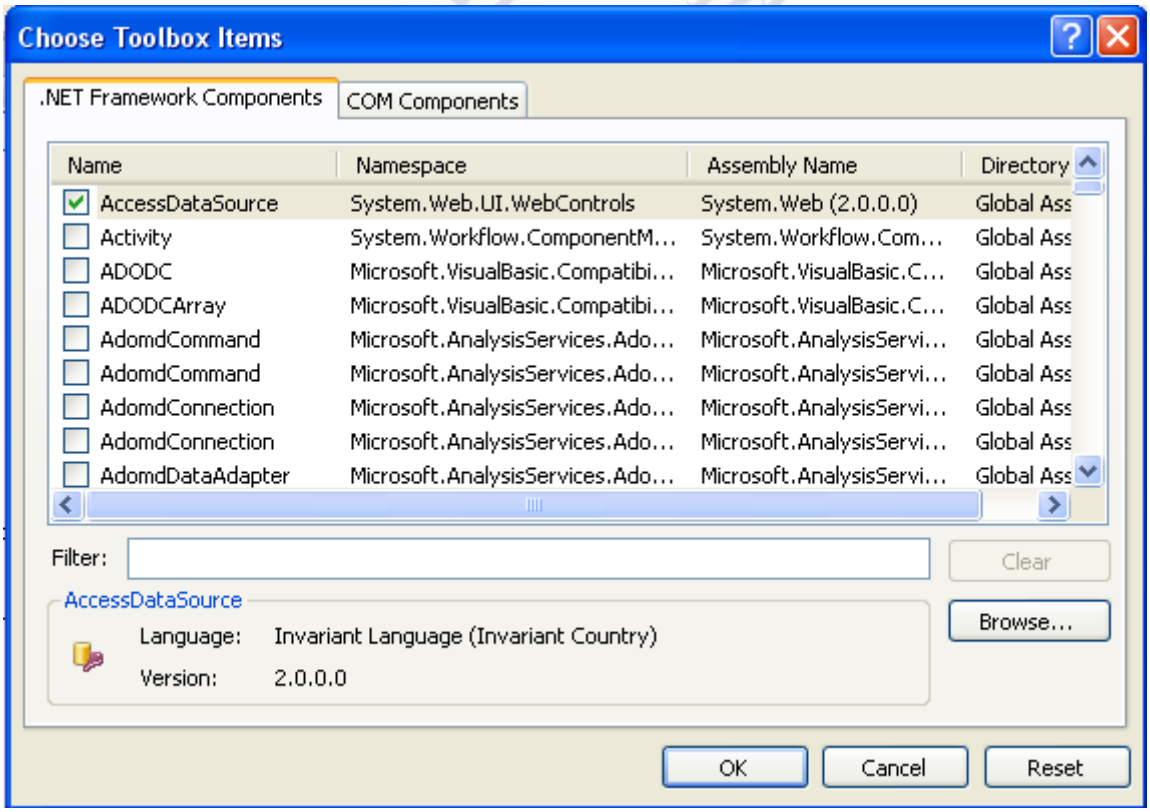
1) Да се включи референция към .dll файла на контрола, така, както това е необходимо да се направи за използване на библиотека от ASP.NET модул.
2) Да се добави сървърна директива **@Register** за регистриране на контрола, така, както това се прави за регистриране на потребителски контрол
3) Да се добави контрола в кода на .aspx страницата.

Visual Studio.Net дава възможност да се добави разработеният WEB контрол към прозореца Toolbox. Това дава възможност контролът да бъде включван в .aspx страницата чрез провлачване с мишката в режим Дизайн, така, както това се прави с останалите WEB контроли.

За да се добави WEB контрол към прозореца Toolbox трябва да се щракне десния бутон на мишката върху избрана група и от падащото меню да се избере Choose Items.



Извежда се диалогова кутия Choose Toolbox Items, в която с щракване върху бутона Browse се извежда диалогова кутия, чрез която се избира .dll файла на разработения WEB контрол (от bin директорията на проекта за контрола)



Фиг. 5 Добавяне на WEB контрол към прозореца Toolbox

27.2. Съхраняване на състоянието на добавените атрибути - атрибут **ViewState**

Web приложенията по подразбиране не съхраняват състоянието си. При всяко заявяване на модул от приложението се създава нов екземпляр на обект от клас Page за модула и обекти за включените в него сървърни контроли. Едно от предимствата на ASP.NET технологията е включеният в модела на WEB формите механизъм за съхраняване на състоянието на сървърните контроли при следващо заявяване на ASP.NET модула.

В разгледания по-горе жизнен цикъл на ASP.NET модула след зареждането на контролите състоянието им (което е съдържанието на всичките им атрибути) се записва в колекции StateBag чрез техните атрибути ViewState. След записа на състоянието на всички контроли се генерира събитието SaveStateComplete на ASP.NET модула.

Атрибутът ViewState на сървърните контроли предоставя достъп до колекцията от клас StateBag, която съдържа записаните състояния на атрибутите на контрола (по-нататък ще я наричаме просто колекция ViewState). За да се запази състоянието на контролите по време на цикъла на обмен на данни между ASP.NET модула и WEB браузъра, ASP.NET сървъра използва техните колекции ViewState. От състоянието на всички контроли се формира символен низ и се записва в обект от тип hidden във WEB формата, чрез който се предава към WEB браузъра.

Всеки контрол трябва сам да съхранява състоянието си като записва изменението на състоянието на атрибутите си в колекцията си ViewState. В даденият по-горе пример за контрол, рзширяващ WEB контрола TextBox, е добавен атрибут NewProp. Променянето на състоянието на атрибута обаче не се записва в колекцията ViewState на контрола. Резултатът е че стойността на атрибута не се запазва при следващо зареждане на .aspx страницата. За да се съхранява стойността на атрибута при следващо зареждане на страницата е необходимо при нейното променяне тя да се записва в колекцията ViewState, а при четене на атрибута стойността му да се извлича от ViewState. Даденият по-долу пример съдържа декларация на допълнителния атрибут NewProp, чиято стойност се съхранява и извлича чрез ViewState. Резултатът при заменянето на кода на атрибута е че стойността на атрибута се запазва при следващо зареждане на .aspx страницата.

```
// Допълнителен атрибут
      public string NewProp
      {
         get
         {
            if (ViewState["NewProp"] == null) return "";
            else return ViewState["NewProp"].ToString();
         }
         set
         {
            ViewState["NewProp"] = value;
         }
      }
```

Пример 70 Код на атрибут, чието съдържание се съхранява в колекцията ViewState

Повече информация за атрибута ViewState и значението му за съхраняването на състоянието на контролите в ASP.NET модулите можете да намерите в статията на Scott Mitchell “Understanding ASP.NET View State” на адрес http://msdn.microsoft.com/en-us/library/ms972976.aspx.

27.3. Извеждане на допълнителен HTML код преди кода на контрола – метод SetRenderMethodDelegate

Методът (наследен от класа Control) присвоява делегат за събитийна процедура, която се изпълнява в родителския контрол на WEB контрола преди да се изпълни метода Render на контрола. Чрез тази събитийна процедура разработчикът може да добави свой HTML код към това, което WEB контрола извежда в .aspx страницата.

Декларация: (C#)

```
public void SetRenderMethodDelegate (
      RenderMethod renderMethod //Делегат от тип RenderMethod
)
```

Делегатът от тип RenderMethod е метод, който трябва да бъде деклариран в родителския контрол на WEB контрола. Той има следната декларация:

```
public delegate void RenderMethod (HtmlTextWriter output, Control container)
```

където първия параметър е референция към обекта от клас HtmlTextWriter, който се използва за извеждане, а втория параметър е референция към родителския контрол на WEB контрола.

```
protected void Page_Load(object sender, EventArgs e)
{
   this.MyCustomControl.SetRenderMethodDelegate(this.RenderCustom);
}
```

Пример 71 Добавяне на делегат от тип RenderMethod към WEB контрол

По-долу е даден примерен код на събитийната процедура, която извежда текста “Caption of my control”

```
protected void RenderCustom(System.Web.UI.HtmlTextWriter writer,
  System.Web.UI.Control Container)
{
   writer.Write("<b>Caption of my control </b><br/>");
}
```

Пример 72 Събитийна процедура от тип RenderCustom

ADO.NET

ADO.NET е основният модел за операции с бази от данни за приложенията, базирани на Microsoft .NET. Компонентите на ADO.NET са включени в обвивката на .NET Framework; основно в именното пространство System.Data и подпространствата System.Data.OleDb и System.Data.SqlClient.

ADO.NET често се разглежда като еволюция на библиотеката ActiveX Data Objects (ADO) (включена във Visual Studio 6.0 и един от основните компоненти на ASP технологията) , поради което е полезно да се направи сравнение между двете технологии.

1. Предимства на ADO.NET

ADO.NET притежава някои съществени предимства в сравнение с ADO технологията:

1. Възможност за операции с данни без поддържане на постоянна връзка с източника:
   Един от основните компоненти на ADO.NET е обектът DataSet, който предоставя възможност за структурирано съхраняване на таблици с данни, получени от различни източници. След зареждане на данните в DataSet връзката с източника може да бъде прекъсната и да се извършват операции само със съхранените данни. Връзката е необходимо да се възстанови тогава, когато е необходимо да се актуализират данните в източника. В библиотеката ADO обектът с аналогично предназначение е Recordset, но той може да съхранява само една динамична таблица.
2. Възможност за обмен на данни в XML формат:
   Едно от ограниченията на ADO технологията беше обмена на данни в двоичен ADTG формат, поддържан в хомогенна среда от COM обекти. Компонентите на ADO.NET предоставят методи за обмен на данни в XML формат, което разширява възможностите за операции в хетерогенна среда. Това е особено важно за интернет приложения, които използват разпределени източници на данни.

Net Framework предоставя възможност за използване в .NET приложенията на обекти, разработени по COM технологията (включително и обекти от библиотеката ADO), но за целта те се включват в клас-обвивка (wrapper), което усложнява интерфейса за операции с тях. Освен това COM обектите се изпълняват в режим на единична апартаментна нишковост, което също създава проблеми при тяхното използване. Поради това за разработването на ASP.NET приложения се препоръчва операциите от данни да се реализират чрез ADO.NET.

2. Пространства от имена на ADO.NET

ADO.NET предоставя на разработчика набор от класове за основните операции с източници на данни. Те са включени в именното пространство System.Data и неговите подпространства. Класовете в подпространствата притежават определена специализация за определен вид източници на данни и определени операции:

System.Data.OleDb	Колекция от класове за операции с OLE DB източници на данни.
System.Data.SqlClient	Колекция от класове, оптимизирани за операции бази от данни на SQL Server 7.0 и следващи версии.
System.Data.Sql	Колекция от класове за специфични за SQL Server-операции, напр. получаване на инстанции на достъпните сървъри.
System.Data.SqlTypes	Предоставя класове за типове от данни, използвани в SQL Server 2005.
Microsoft.SqlServer.Server	Предоставя класове, интерфейси и колекции, използвани при интегрирането на CLR (common language runtime) в Microsoft SQL Server и неговото обкръжение.
System.Data.Odbc	Колекция от класове за операции с ODBC източници на данни.
System.Data.OracleClient	Колекция от класове, специализирани за операции бази от данни на Oracle
System.Transactions	Предоставя класове за създаване на транзакционни приложения.

3. Основен сценарий за операция с източник на данни

Основният сценарий за извършване на операция с източник на данни включва:

1) Създаване на връзка с източника на данни
2) Извличане на данни от източника. Тези данни могат да бъдат набор от записи във вид на динамична таблица или само информация за структурата на данните.
3) Изпълнение на команда за изтриване, добавяне или актуализиране на данни.
4) Прекратяване на връзката с източника на данни.

Както беше посочено по-горе, едно от предимствата на ADO.NET е това,че не е необходимо да се поддържа постоянно връзка с източника на данни. След извличането на данни те могат да бъдат съхранени в структуриран вид в обект DataSet, да се актуализират в него и връзката с източника да бъде възстановена тогава, когато приложението трябва да актуализира данните в източника.

Основните класове, които се използват при операции с бази от данни по този сценарий се съдържат в именните пространства System.Data, System.Data.OleDb и System.Data.SqlClient. Ще разгледаме последователно класовете, които те ни предоставят. В повечето случай пространствата System.Data.OleDb и System.Data.SqlClient предоставят аналогични по предназначение класове с техните атрибути и методи, така, че те ще бъдат разглеждани общо, като при необходимост ще се посочват различията

4. Създаване на връзка с източника на данни. Класове OleDbConnection и SqlConnection

Предоставят атрибути и методи за осъществяване на връзка с източника на данни. Ако системата за работа с БД е организирана по модела клиент-сървър, обектите от тези класове представят мрежова връзка със сървъра. Всички операции с БД, изпълнявани чрез ADO.NET обектите използват връзка , създадена чрез обекти Connection – екземпляри на OleDbConnection и SqlConnection. В зависимост от функционалността, предоставяна на клиента от доставчика на данни (data provider), някои колекции, атрибути или методи могат да не бъдат

достъпни. Класът SqlConnection е оптимизиран за създаване на връзка с SQL Server 7.0/2000, докато класът OleDbConnection използва интерфейса OLE DB

Класовете OleDbConnection и SqlConnection. предоставят следните атрибути, методи и събития:

4.1. Конструктори на класовете OleDbConnection и SqlConnection

Класовете предоставят конструктори без параметър и конструктори с един параметър – символен низ, чрез който се задават параметри на връзката с източника и доставчика на данни

```
public OleDbConnection([string]);
public SqlConnection([string]);
```

В дадените по-долу примери е показано използването на конструктор с параметри за създаване на обект Connection и използването на метода Open за създаване на връзка с източника.

```
public void CreateOleDbConnection()
{
  string myConnString = "Provider=SQLOLEDB;Data
     Source=localhost;Initial Catalog=Northwind;Integrated
     Security=SSPI;Connect Timeout=30";
  OleDbConnection myConnection = new
     OleDbConnection(myConnString);
  myConnection.Open();
}
```

ADO.NET Пример 1 Създаване на връзка с източник на данни посредством обект OleDbConnection.

```
public void CreateSqlConnection()
 {
   string myConnectString = "user id=sa;password=Jx$442pt;initial
     catalog=northwind;data source=mySQLServer;Connect Timeout=30";
   SqlConnection myConnection = new SqlConnection(myConnectString);
   myConnection.Open();
 }
```

ADO.NET Пример 2 Създаване на връзка с източник на данни посредством обект SqlConnection.

4.2. Атрибути на класовете OleDbConnection и SqlConnection

Класовете предоставят набор от атрибути, които представят параметри на връзката с източника и доставчика на данни:

ConnectionString	Тип **string** - Задава или връща низ от параметри, необходими за създаване на връзка.
ConnectionTimeout	Тип **int,** само за четене **-** Връща времето, което се изчаква за установяване на връзка.(в секунди).
Database	Тип **string,** само за четене - Връща името на текущата база от данни или на базата от данни, която ще се използва при създаване на връзка с източника.
DataSource	Тип **string,** само за четене - Връща пътя и името на източника на данни.
Provider	Тип **string,** само за четене - Връща името на доставчика на данни OLE DB provider.
ServerVersion	Тип **string,** само за четене - Връща низ, съдържащ версията на сървъра, към който клиентът се свързва.
State	Тип enum ConnectionState - Връща текущото състояние на връзката.

Както се вижда от таблицата, всички атрибути изброени са само за четене, с изключение на атрибута ConnectionString. За разлика от обекта Connection на библиотеката ADO, в ADO.NET само атрибута ConnectionString задава параметри на връзката с източника и доставчика на данни. Всички останали атрибути са само за четене и връщат информация за параметри на връзката. За потребителя особен интерес представлява атрибутът State, който връща информация за състоянието на връзката.

4.2.1. Атрибут ConnectionString

Тип **string** - Задава или връща низ от параметри, необходими за създаване на връзка. Може да съдържа набор от параметри като двойки име-стойност във формат ***параметър=стойност*** , разделени с “;”. Задължителен е параметърът Provider. За разлика от ADO, ADO.NET не поддържа доставчик MSDASQL за ODBC. За достъп до ODBC източници на данни е необходимо да се използва ODBC .NET доставчик

– той може да бъде изтеглен от http://msdn.microsoft.com/downloads. По-долу са дадени примери за съдържание на атрибут ConnectionString.

```
Provider=MSDAORA; Data Source=ORACLE8i7; User ID=OLEDB;
        Password=OLEDB
Provider=Microsoft.Jet.OLEDB.4.0; Data Source=c:\Members.mdb;
Provider=SQLOLEDB;Data Source=MySQLServer;Integrated
Security=SSPI;
```

ADO.NET Пример 3– Съдържание на атрибут **ConnectionString** за връзка с база от данни на Oracle, Access и SQL Server.

4.2.2. Атрибут State

Атрибутът е от изброим тип **ConnectionState,** само за четене.Той връща информация за състоянието на връзката. Може да съдържа една или няколко константи от тип ConnectionState, обединени с операция ИЛИ. Подразбиращата се стойност на атрибута е **Closed**. След успешно изпълнение на метода Open на обект Connection стойността на атрибута се променя на **Open**. Даденият по-долу пример използва атрибута State за да изведе във Web контрол Label състоянието да връзката след изпълнение на метода Open.

```
public void createOleDbConnection()
{
  OleDbConnection myConnection = new OleDbConnection();
  myConnection.ConnectionString = "Provider=SQLOLEDB;Data
Source=localhost;Initial Catalog=Northwind;Integrated Security=SSPI;";
  myConnection.Open();
  Label1.Text = "Connection State: " + myConnection.State.ToString();
  myConnection.Close();
}
```

Извеждане на състоянието на връзката с източник на данни посредством атрибут **State.**

Обектите Connection от клас OleDbConnection и SqlConnection генерират събитие **StateChange**, което програмистът може да използва за да изпълни своя процедура при промяна на състоянието на връзката.

4.3. Методи на класовете OleDbConnection и SqlConnection

Класовете OleDbConnection и SqlConnection предоставят набор от методи, които могат да се класифицират на осбствени и наследени от родителски класове. По-долу е дадено кратко описание на методите и по-детайлно описание на собствените методи на класовете.

4.3.1. Собствени методи

Open	Отваря връзка към базата от данни като използва параметрите, съдържащи се в атрибута ConnectionString.
Close	Затваря връзката към източника на данни. Това е предпочитания метод за затваряне на отворена връзка.
BeginTransaction	Задава начало на транзакция над базата от данни.
ChangeDatabase	Променя текущата база от данни за отворена връзка.
CreateCommand	Създава и връща обект Command, асоцииран с обекта Connection (съответно обект OleDbCommand за OleDbConnection или обект SqlCommand за SqlConnection)
GetOleDbSchemaTable (само за клас OleDbConnection)	Връща информация за схемата на базата от данни, като се отчитат зададените рестрикции.
ReleaseObjectPool	Показва, дали обединението на създадените връзки (connection pool) може да бъде изчистен когато се прекрати връзката със всички OLE DB доставчици.

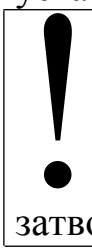
4.3.2. Метод, наследен от класа Component.

Dispose	Освобождава ресурсите, използвани от компонента.Може да се използва за затваряне на връзката вместо метода Close

4.3.3. Установяване на връзка с източника на данни - **Метод Open** -класове OleDbConnection, SqlConnection

Декларация (C#): public void Open();

Методът извлича връзка от обединението **connection pool** или (ако няма достъпна такава или използването на обединение е забранено) установява връзка с източника на данни.



Важно е да се запомни, че след излизането от областта на действие на обект Connection (например след прекратяване на обработката на .aspx страницата) връзката с източника не се затваря автоматично. Това означава, че програмистът трябва обязателно да затвори връзката чрез метода Close или чрез метода Dispose.

Даденият по-долу пример съдържа кода на процедура, която използва метода Open на класа OleDbConnection за установяване на връзка с източник на данни. Процедурата получава като параметър низ myConnString с параметри на връзката. След изпълнение на метода Open се извежда във Web контрол Label версията на сървъра и състоянието на връзката.

```
public void CreateOleDbConnection(string myConnString)
{
  OleDbConnection myConnection = new
OleDbConnection(myConnString);
  myConnection.Open();
  Label1.Text = "ServerVersion: " + myConnection.ServerVersion
    + "\nState: " + myConnection.State.ToString();
  myConnection.Close();
}
```

4.3.4. Прекратяване на връзката с източника на данни - Метод Close -класове OleDbConnection, SqlConnection

Декларация (C#): public void Close();

Методът превърта обратно всички непотвърдени транзакции, асоциирани с връзката. След това се премахва връзката от обединението connection pool или се прекратява връзката ако използването на обединението е забранено. Ако методът се извика при обработване на събитието StateChange, други събития от този тип повече не се генерират.

При повторно извикване на метода Close за вече прекратена връзка не се генерира грешка (exception).

4.3.5. Създаване на обект Command чрез обект Connection - Метод CreateCommand -класове OleDbConnection, SqlConnection

Декларации (C#) public OleDbCommand CreateCommand();
public SqlCommand CreateCommand();

Методът връща обект Command, който дава възможност за извършване операции над източника на данни. За разлика от библиотеката ADO, в библиотеката ADO.NET само обект Command дава възможност за актуализиране на данни, така че ако програмистът създава приложение, което да може да добавя, изтрива и променя записи в БД, той трябва да създаде обект Command чрез неговия конструктор или чрез метода CreateCommand на обект от клас **OleDbConnection** или **SqlConnection.** NET Framework предоставя на програмиста WEB контроли, които дават възможност за операции с БД. Тези контроли също използват обект Command, но той се създава неявно, като програмистът само задава необходимите параметри за създаване на връзка с БД.

4.3.6. Извличане на информация за структурата на БД - Метод GetOleDbSchemaTable -клас OleDbConnection

Декларация (C#):
public DataTable GetOleDbSchemaTable(Guid schema, object[] restrictions);

където:
schema – (изброим тип OleDbSchemaGuid) определя вида информация, която се извлича за базата от данни.
Restrictions (тип object[]) **–** задава критерий (филтър), който да се използва извличане на данните.

Методът получава от доставчика информация за базата от данни в съответствие със зададената схема в параметъра **schema** по зададен критерий в параметъра **restrictions**. Методът връща обект **DataTable**

(динамична таблица), от който може да се извлекат получените данни. На практика схемата определя какви полета ще има получената динамична таблица. Стойностите в масива **restrictions** съответстват на реда на стойностите от зададената схема**.** Ако на параметъра **restrictions** се присвои празен масив, се извличат всички данни от зададената схема.

Даденият по-долу пример извлича информация за таблиците, които се съдържат в базата от данни, за която е създаден обект Connection.

```
public DataTable GetTables(OleDbConnection conn)
{
  conn.Open();
  DataTable schemaTable =
  conn.GetOleDbSchemaTable(OleDbSchemaGuid.Tables,
                  new object[] {null, null, null, "TABLE"});
  conn.Close();
  return schemaTable;
}
```

ADO.NET Пример 4 Получаване на информация за таблиците в база от данни чрез метода. **GetOleDbSchemaTable.**

4.4. Събития на класовете OleDbConnection и SqlConnection

Обектите Connection – екземпляри на OleDbConnection и SqlConnection – генерират следните събития:

StateChange	Генерира се когато се промени състоянието на връзката.
InfoMessage	Генерира се когато доставчикът изпрати предупреждение или информационно съобщение.
Disposed	(наследено от класа Component) Добавя манипулатор за събитието Disposed на компонента, което възниква при унищожаването му.

Програмистът може да асоциира събитийни процедури с тези събития и така да предизвика изпълнението на свой код при настъпването им.

5. Изпълнение на команди над източника на данни - класове OleDbCommand и SqlCommand

Класовете са декларирани като ненаследяеми (sealed) в именните пространства съответно **System.Data.OleDb** и **System.Data.SqlClient.** Те предоставят атрибути и методи, свързани с изпълнението на команди (SQL команди и съхранени процедури) над източника на данни.

Конструктори: [C#]
public OleDbCommand([string [, OleDbConnection [, OleDbTransaction]]]);
public SqlCommand([string [, SqlConnection [, SqlTransaction]]]);
където:
string е низ, който съдържа SQL команда и съхранена процедура.
OleDbConnection, SqlConnection - обект Connection от съответния клас.
OleDbTransaction, SqlTransaction - обект Transaction от съответния клас.

Собствени атрибути: **CommandText, CommandTimeout, CommandType, Connection, DesignTimeVisible, Transaction, UpdatedRowSource.**

Атрибути, наследени от клас Component: Container, Site;

5.1. Методи на класовете OleDbCommand и SqlCommand

Методите на обект Command от класове OleDbCommand и SqlCommand дават възможност за изпълнение на команди над източника и доставчика на данни. Обектът предоставя три различни методи за изпълнение на команди – ExecuteReader, ExecuteScalar и ExecuteNonQuery, всеки от които е подходящ за определен тип команди;

5.1.1. Създаване на обект за извличанена записи в изходен поток– метод ExecuteReader

Методът ExecuteReader изпраща командата, зададена в CommandText към доставчика на данни като използва асоциирания обект Connection и създава и връща обект DataReader съответно от клас OleDbDataReader или SqlDataReader. Обектът DataReader дава възможност за последователно четене на извлечените записи и полетата в тях.

Метод Cancel

Декларация (C#): public override void Cancel()

Изпраща към доставчика команда за прекратяване на изпълнението на команда над източника на данни.

Метод CreateParameter

Декларация: public OleDbParameter CreateParameter()

Създава обект Parameter от класа съответно OleDbParameter или SQLParameter (в зависимост от класа на обект Command).

Метод ExecuteNonQuery

Декларация (C#): public OleDbParameter CreateParameter()
public override int ExecuteNonQuery()

Изпълнява SQL команда като използва обекта Connection, асоцииран с обект Command и връща броя записи, които са участвали при изпълнението на командата.

Пример:

```
SqlConnection connection = new SqlConnection(dbConnectionString);
SqlCommand comm = new SqlCommand("sp_AddUser", connection);
comm.CommandType = CommandType.StoredProcedure;
comm.Parameters.Add("@name", SqlDbType.VarChar).Value = "John";
comm.Parameters.Add("@email", SqlDbType.VarChar).Value ="j@abv.bg";
connection.Open();
comm.ExecuteNonQuery();
connection.Close();
```

Пример 73 В примера чрез метода ExecuteNonQuery се извиква за изпълнение съхранена процедура "sp_AddUser" с параметри @name и @email. Преди извикването на метода параметрите се добавят към колекцията Parameters на създадения обект от клас SqlCommand.

Метод ExecuteScalar

Декларация (C#): public override Object ExecuteScalar()

Изпълнява заявка и връща стойността на първото поле от първия запис от извлечената динамична таблица Останалите извлечени данни се игнорират.

Метод Prepare

Декларация (C#): public override void Prepare()

Създава обработена (или компилирана) версия на зададена команда за операции над източника на данни.

Метод ResetCommandTimeout

Декларация (C#): public void ResetCommandTimeout()

Записва подразбиращата се стойност на атрибута CommandTimeout – 30 секунди.

5.2. Атрибути на класовете OleDbCommand и SqlCommand

CommandText Задава и връща текст на SQL команда или име на съхранена процедура, която да се изпълни над източника на данни.

CommandTimeout Задава и връща времето за изчакване за изпълнение на командата (в секунди) след изтичане на което се генетира грешка.

CommandType Задава и връща стойност, която показва значението на атрибута CommandText.

Connection Задава и връща обект Connection (от клас съответно OleDbConnection или SqlConnection) използван от обекта Command.

DesignTimeVisible Задава и връща стойност, която показва дали обектът трябва да бъде видим в режим Дизайн.

Transaction Задава и връща транзакцията, в която OleDbCommand се изпълнява.

UpdatedRowSource Задава и връща начина, по който резултатът от изпълнение на командата се прилага към обектите DataRow когато се изпълнява метода Update на обект DbDataAdapter.

5.3. Колекция на класовете OleDbCommand и SqlCommand

Parameters Връща колекция от тип OleDbParameterCollection, която съдържа обекти Parameter, асоциирани с обекта Command.

6. Извличане на данни от източника в изходен поток. Класове OleDbDataReader и SqlDataReader

Класовете OleDbDataReader и SqlDataReader се съдържат съответно в именните пространства System.Data.OleDb и System.Data. SqlClient Те предоставят възможност за последователно четене на данни от източника в изходен поток. Това е операция, която натоварва възможно най-малко доставчика на данни и като обем на изчислителни операции и като ресурси (памет).

Обект от клас OleDbDataReader или SqlDataReader може да бъде създаден посредством метода ExecuteReader на обект Command съответно от клас OleDbCommand или SqlCommand. Класовете не предоставят публичен конструктор.

Когато се използва обект DataReader, асоциираната с него връзка и съответният обект Connection са ангажирани и не могат да се използват за други операции. Операциите с обект DataReader се прекратяват след изпълнението на метода му Close и след това могат се извършват други операции с обект Connection, например създаване на друг обект Command.

Измененията, извършвани над източника на данни от друг процес по принцип са видими за потребителя, който чете данни чрез обект DataReader, но резултата зависи от времевите съотношения на двата процеса.

След изпълнение на метода Close на обект DataReader са достъпни за четене само атрибутите IsClosed и RecordsAffected.

Даденият по-долу пример съдържа код на процедура, която демонстрира използването на обект DataReader. Процедурата прочита и изпраща към клиента (потребителския браузър) съдържанието на полетата OrderID и CustomerID от всички записи в таблицата Orders във вид на номериран HTML списък.

```
<script language="c#" runat="server">
```

```
public void ReadMyData(string myConnString) {
string mySelectQuery = "SELECT OrderID, CustomerID FROM Orders";
  OleDbConnection myConnection = new
      OleDbConnection(myConnString);
  OleDbCommand myCommand = new
      OleDbCommand(mySelectQuery,myConnection);
  myConnection.Open();
  OleDbDataReader myReader = myCommand.ExecuteReader();
  // Извикването на метод Read трябва да предхожда методите Get
  Response.Write("<OL>");
  while (myReader.Read()) {
  Response.Write("<LI>"+myReader.GetInt32(0) + ", " +
      myReader.GetString(1));
  }
  // След прочитането на даните трябва да се изпълни метода Close
  myReader.Close();
  // След края на операциите се затваря връзката
  myConnection.Close();
  Response.Write("</OL>");
}
</script>
```

ADO.NET Пример 5- Прочитане на данни с обект Reader

Класовете OleDbDataReader или SqlDataReader предоставят следните атрибути, методи и събития:

6.1. Атрибути на класовете OleDbDataReader и SqlDataReader

FieldCount (тип int) Връща броя на полетата в текущия запис.

HasRows (тип bool) Връща стойност, която показва дали обекта DataReader съдържа записи.

IsClosed (тип bool) Показва дали изходния поток на DataReader е затворен.

Item[int index|string name**]** Връща стойността на поле от текущия запис в оригиналния формат на данните. Полето може да се адресира с индекс или име на полето.

RecordsAffected (тип int) Връща броя на прочетените записи

6.2. Методи на обект от класове OleDbDataReader и SqlDataReader

6.2.1. Метод Read - OleDbDataReader , SqlDataReader

Методът прочита един ред от източника на данни, представен от асоциирания обект Command. След неговото изпълнение съдържанията на полетата (колони в динамичната таблица) са достъпни за четене с Get методите на обекта. При създаване на обект DataReader с метода ExecuteReader на обект Command указателят на обект DataReader е преди първия ред и за да могат да се прочетат данни с методите Get е необходимо да се изпълни първо метода Read.
Методът връща логическа стойност true ако има още редове, достъпни за четене и стойност false ако са прочетени всички редове от източника на данни са вече прочетени.

6.2.2. Метод Close - OleDbDataReader , SqlDataReader

Затваря изходния поток на обект DataReader от клас OleDbDataReader или SqlDataReader. Изпълнението на метода освобождава връзката с източника на данни, представена от асоциирания обект Connection и тя може да бъде използвана за изпълнение на други операции.

6.2.3. Get методи, които връщат единична стойност

Това са методите GetBoolean, GetByte, GetChar, GetDateTime, GetDecimal, GetDouble, GetFloat, GetGuid, GetInt16, GetInt32, GetInt64, GetString, GetTimeSpan. Те прочитат и връщат стойност от поле (от прочетен с метода Read ред) по зададен индекс на полето с нулева минимална стойност (нулево базиран). Методите не извършват

преобразуване на данни и за да се изпълнят без генериране на грешка, съответното поле трябва да съдържа данни от тип, съответстващ на метода. Например за да се изпълни метода GetInt32(1), то полето с индекс 1 от прочетения ред трябва да съдържа данни от тип ***signed int32.*** Ако полето съдържа данни от тип, който не съответства на метода се генерира изключение InvalidCastException. Желателно е преди изпълнението на Get метод да се прави проверка за липсваща или нулева стойност посредством метода **IsDBNull.**
Декларация (C#):
public data_type Get_data_type (int ordinal);
където:
data_type е типът на данните, които връща метода
ordinal - нулево базиран индекс на полето в прочетения ред.

6.2.4. Извличане на данните от поле в оригиналния им формат - Метод GetValue

Методът връща съдържанието на поле в оригиналния формат на данните по зададен нулево базиран индекс (ordinal). Ако полето съдържа липсваща или нулева стойност, методът връща стойност DBNull.
Декларация (C#):
public object GetValue(int ordinal);

6.2.5. Методи за прочитане на низ от стойности:

Метод GetBytes-Прочита поток от байтове от зададено поле в буфер.като записва данните в буфера от зададена начална позиция.

Декларация (C#):
```
public long GetBytes(
   int i,                  //индекс на колоната (полето) в реда - начален индекс 0
   long dataIndex,   //индекс в полето, от който започва четенето
   byte[] buffer,      //буфер, в който се записва прочетените данни
     int bufferIndex,  //начален индекс в буфера, от който се записват данните
   int length          //максимална дължина на записа в буфера
```

);

Методът GetBytes връща броя на достъпните байтове в полето. В повечето случаи това е дължината на цялото поле. Възможно е обаче това да е броя на оставащите байтове, ако GetBytes е вече извикван за да прочете част от байтовете, записани в полето.
Ако вместо име на масив на параметъра buffer се предаде стойност **null**, методът GetBytes връща общия брой байтове, записани в полето.
Не се извършва преобразуване на типа на данните, което означава, че данните, записани в полето трябва да са от тип byte, за да може да се изпълни метода.

Метод GetChars-Прочита поток от символи от зададено поле в буфер.като записва данните в буфера от зададена начална позиция.

Декларация (C#):
```
public long GetChars(
  int i,                // нулево базиран индекс на колоната (полето) в реда
  long dataIndex,   //индекс в полето, от който започва четенето
  char[] buffer,      //буфер, в който се записва прочетените данни
    int bufferIndex,  //начален индекс в буфера, от който се записват данните
  int length          //максимална дължина на записа в буфера
);
```
Методът е аналогичен по характеристики на метода GetBytes, като разликата е само в типа на прочитаните данни.

6.2.6. Прочитане на всички полета в масив. Метод GetValues.

Методът прочита съдържанието на всички полета в масив от тип Object. Както е известно, типът Object(деклариран като обект в System.Object) е родителски тип на всички типове променливи в.NET Framework и поради това в променлива, декларирана като Object може да се записват стойности от всеки друг тип.

Декларация (C#):
```
public int GetValues(
  object[] values //масив от тип Object, в който се записват прочетените //
                  стойности на полета
```

);
Пример:

```
Object[] values = new Object[reader.FieldCount];
int fieldCount = reader.GetValues(values);
```

ADO.NET Пример 6 Извличане на всички полета от текущия запис на DataReader в масив

6.2.7. Получаване на типа на данните - Метод GetDataTypeName

Методът връща низ с името на типа данни в зададено поле.
Декларация (C#):

```
public string GetDataTypeName(
  int index  //индекс на полето (начална стойност 0)
);
```

6.2.8. Извличане на типа на полето - Метод GetFieldType

Методът връща типа на полето в съответствие с декларираните типове данни в .Net Framework.
Декларация (C#):

```
public Type GetFieldType(
  int index  //индекс на полето (начална стойност 0)
);
```

6.2.9. Извличане на името на полето - Метод GetName

Методът връща низ с името на поле по зададен индекс. Можете да използвате този метод например при извеждане на заглавен ред с имената на полетата на таблица, която съдържа извлечени записи от базата от данни.
Декларация (C#):

```
public string GetName(
  int index  // нулево базиран индекс на полето
);
```

6.2.10. Извличане на информация за полетата - Метод GetSchemaTable

Методът връща таблица с информация (метаданни) за полетата в извадката. Декларация:
public override DataTable GetSchemaTable ()

6.2.11. Проверка за нулева стойност – метод IsDBNull

Връща логическа стойност true ако в полето няма записана стойност, и false в противен случай.

6.2.12. Прочитане на следващ резултат – метод NextResult

Премества DataReader на следващия резултат при пакетно изпълнение на SQL команди. Връща логическа стойност true ако резултатът е извлечена динамична таблица (resultset), , и false в противен случай.
Декларация:
public override bool NextResult()

7. Структурирано кеширане на данни - клас DataSet

Класът DataSet, който предоставя възможност за съхраняване (кеширане) в паметта на извадки от източници на данни със запазване на тяхната структура, е основен компонент от архитектурата на ADO.NET. Обектът DataSet (екземпляр от клас DataSet) може да съдържа колекция от обекти DataTable, които представят кеширани таблици. Обектите DataTable могат да се свързват с релации посредством обекти DataRelation. Задаване на проверка за интегритет на данните може да се осъществи посредством обекти UniqueConstraint и ForeignKeyConstraint. Данните в DataSet се съхраняват в XML формат, така че можем да си представяме DataSet като XML файл, съхраняван в оперативната памет. Освен структурираното съдържание обаче DataSet предоставя набор от атрибути и методи за операции с данните и обмен на данни с външни файлове и обекти.

Конструктор:

[C#] public DataSet([string]);

където незадължителният параметър string задава име на обекта

Даденият по-долу пример съдържа код на събитийната процедура Page_Load, с който се създава обект DataSet и се добавят към него две таблици. Съдържанието на таблиците се извежда в контроли GridView.

Обърнете внимание на създаването на авто-инкрементираща колона и първичен ключ на таблицата. Атрибута PrimaryKey на обект DataTable е от тип DataColumn[], т.е. масив от обекти DataColumn, тъй като първичен ключ може да бъде не само една колона но и набор от колони. За да може да се дефинират една или набор от колони като първичен ключ (PrimaryKey) те трябва вече да са добавени към таблицата.

```
private void Page_Load(object sender, System.EventArgs e)
{
   DataSet  myDataSet= new DataSet("aNewDataSet");
   //Create two DataTable objects using a function.
   DataTable table1 = MakeTable("idTable1", "thing1");
   DataTable table2 = MakeTable("idTable2", "thing2");
   myDataSet.Tables.Add(table1);
   myDataSet.Tables.Add(table2);
   GridView1.DataSource = myDataSet.Tables[0];
   GridView1.DataBind();
   GridView2.DataSource = myDataSet.Tables[1];
   GridView2.DataBind();
}
//Функция, която създава и връща таблица с две колони и два реда
DataTable MakeTable(string c1Name, string c2Name)
{
      DataTable myTable = new DataTable(c1Name);
      //Добавяне на две полета чрез обекти DataColumns
      DataColumn myColumn1 = new DataColumn(c1Name,
      System.Type.GetType("System.Int32"));
      // Колоната myColumn1се задава като авто-инкрементираща
      myColumn1.AutoIncrement = true;
      myColumn1.AutoIncrementSeed = 1; //начална стойност
      myTable.Columns.Add(myColumn1);
      // Колоната myColumn1се обявява за първичен ключ
      myTable.PrimaryKey = new DataColumn[]{ myColumn1 };
```

```
        DataColumn myColumn2 = new DataColumn(c2Name,
        System.Type.GetType("System.String"));
        myTable.Columns.Add(myColumn2);
        //Добавяне на редове към таблицата
        DataRow tRow = myTable.NewRow();
        tRow[c2Name] = "Test1"; //Запис на стойност в полето
        myTable.Rows.Add(tRow); //Добавяне на реда
        tRow = myTable.NewRow();
        tRow[c2Name] = "Test2";  //Запис на стойност в полето
        myTable.Rows.Add(tRow);
        return myTable;
}
```

7.1. Атрибути на обект DataSet

CaseSensitive (тип **bool**) - Задава и връща логическа стойност, която показва дали при сравнението на низове в обектите DataTable се прави разлика между малки и големи букви.

DataSetName (тип **string**) Задава и връща името на обекта DataSet.

DefaultViewManager (Клас DataViewManager) Задава обект от клас DataViewManager, който се използва за създаване на потребителски изглед на данните в DataSet с възможност за филтриране, търсене и навигация.

EnforceConstraints (тип **bool**) - Задава и връща стойност, която показва дали рестрикциите (constraint rules) се изпълняват при операции update.

ExtendedProperties - Връща колекция, съдържаща информация за потребителя..

HasErrors (тип **bool**) - Връща стойност, която показва дали има грешки в някой от редовете на някоя от таблиците в DataSet.

Locale (клас **CultureInfo**) Задава и връща информация за региона на потребителя, която се използва при сравняването на символни низове.

Namespace (тип **string**) - Задава и връща низ с именното пространство за DataSet.

Атрибутът Namespace се използва когато се чете или записва XML документ в DataSet с използване на методите ReadXml, WriteXml, ReadXmlSchema, или WriteXmlSchema.

Именното пространство на XML документ се използва като област на достъпност (scope) на XML атрибутите и елементите когато съдържанието му се чете в DataSet. Например, ако DataSet съдържа схема за четене с именно пространство "myCompany," то при опит за четене (с метод ReadXml) от документ с именно пространство "theirCompany," всички данни които не съответстват на схемата за четене ще бъдат игнорирани.

```
private void ReadData(DataSet thisDataSet){
  thisDataSet.Namespace = "CorporationA";
  thisDataSet.Prefix = "DivisionA";
  // Read schema and data.
  string filename = "CorporationA_Schema.xml";
  thisDataSet.ReadXmlSchema(filename);
  filename = "DivisionA_Report.xml";
  thisDataSet.ReadXml(filename);
}
```

Prefix (тип **string**) - Задава и връща XML префикс, който се свързва с именното пространство на DataSet.

Атрибути на DataSet, които връщат колекции

Relations Връща колекция от обекти DataRelation, които представят от релациите, които свързват таблиците и позволяват релация от родителски към дъщерни таблици.

Tables Връща колекция от обекти DataTable, които представят таблиците, съдържащи се в DataSet.

7.2. Методи на обект DataSet

Методи за копиране на структура и данни от обект DataSet

7.2.1. Създаване на пълно копие - Метод Copy

Копира структура и данни от обект DataSet. Методът връща обект DataSet, който е пълно копие на дадения обект.
Декларация (C#):
public: virtual DataSet Copy();

Даденият по-долу пример съдържа код на функция, която използва метода Copy на обект DataSet за да създаде пълно копие на обекта и да извърши операции с данните, които той съдържа

```
private void CopyDataSet(DataSet myDataSet){
  // Създаване на обектна променлива за обекта - копие.
  DataSet copyDataSet;
  copyDataSet = myDataSet.Copy();
  // Код за операции с обекта - копие.
}
```

ADO.NET Пример 7- Създаване на копие на обект DataSet с метода Copy

7.2.2. Копиране на структурата - Метод Clone

Методът копира цялата структура на обекта DataSet, в това число всички обекти DataTable, схеми, релации и рестрикции (constraints). Не копира никакви данни. Методът връща обект DataSet.
Декларация (C#):
public: virtual DataSet Clone();

7.2.3. Добавяне на обекти към DataSet - Метод Merge

Методът добавя към обекта DataSet съдържанието на зададен обект DataSet, DataTable или масив от обекти DataRow.
Декларация (C#):
public void Merge (DataSet dataSet| DataTable dataTable| DataRow[] dataRow)

7.2.4. Копиране на измененията в DataSet - метод GetChanges

Методът връща копие на DataSet, съдържащо само измененията, направени в обекта DataSet от неговото зареждане с данни или от последното изпълнение на метода AcceptChanges. Ако се зададе параметър DataRowState могат да се извлекат само определен вид изменения

Декларация (C#):

public DataSet GetChanges([DataRowState]);

където:

незадължителният параметър DataRowState задава измененията, които се записват в копието. DataRowState е изброим тип (Enumeration). Стойности:

DataRowState.Added	Задава копиране само на добавените записи
DataRowState.Deleted	Задава копиране само на изтритите записи
DataRowState.Modified	Задава копиране само на променените записи
DataRowState.Detached	

Таблица 2 Значение на стойностите на изброимия тип **DataRowState**

По-долу е даден пример, който използва метода GetChanges за да получи обект DataSet, който да съдържа в копираните таблици само редовете, в които са направени изменения. Променените редове се записват в базата от данни - източник посредством метода Update на OleDbDataAdapter.

```
private void UpdateDataSet(DataSet myDataSet){
  // Check for changes with the HasChanges method first.
  if(!myDataSet.HasChanges(DataRowState.Modified)) return;
  // Create temporary DataSet variable.
  DataSet xDataSet;
  // GetChanges for modified rows only.
  xDataSet = myDataSet.GetChanges(DataRowState.Modified);
  // Check the DataSet for errors.
  if(xDataSet.HasErrors){
    // Insert code to resolve errors.
  }
  // After fixing errors, update the DBMS with the DataAdapter
  // used to create the DataSet.
```

```
  myOleDbDataAdapter.Update(xDataSet);
}
```

ADO.NET Пример 8 – Запис на измененията в копие на обект DataSet чрез метод GetChanges**.**

7.2.5. Проверка за изменения в DataSet - метод HasChanges

Методът проверява за направени изменения в обект DataSet от неговото зареждане с данни или от последното изпълнение на метода AcceptChanges и връща логическа стойност true ако са направени изменения. Ако се зададе параметър DataRowState се прави проверка само за зададен вид изменения.
Декларация (C#):
public bool HasChanges([DataRowState]);
където: незадължителният параметър DataRowState задава измененията, за които се прави проверка. За значението на декларираните стойности на DataRowState вижте описанието на метода GetChanges.

7.2.6. Потвърждаване на изменения в DataSet - метод AcceptChanges

Потвърждава всички изменения, направени в обекта DataSet от неговото зареждане с данни или от последното изпълнение на метода AcceptChanges. Изпълнението на метода предизвиква изпълнението на аналогичния метод за всички таблици (DataTable) в обекта DataSet, което от своя страна предизвиква изпълнението на метода AcceptChanges за всички редове (DataRow) в таблиците.
Декларация (C#):
[Serializable] public void AcceptChanges();

7.2.7. Анулиране на изменения в DataSet - метод RejectChanges

Анулира всички изменения, направени в обекта DataSet от неговото зареждане с данни или от последното изпълнение на метода AcceptChanges. Изпълнението на метода предизвиква изпълнението на аналогичния метод за всички таблици (DataTable) в обекта DataSet, което от своя страна предизвиква изпълнението на метода RejectChanges за всички редове (DataRow) в таблиците.

Декларация (C#):
[Serializable] public void RejectChanges ();

7.2.8. Изтриване на данните в DataSet - метод Clear

Методът изтрива съдържанието на всички таблици DataSet като премахва всички редове в тях.
Декларация (C#):
public void Clear();

7.2.9. Възстановяване на оригиналните данни в DataSet - метод Reset

Методът възстановява първоначалните данни в DataSet от неговото създаване.
Декларация (C#):
public void Reset();

Методи за операции с XML представянето на данните в DataSet

7.2.10. Метод GetXml – Четене в символен низ на данните от обект DataSet в XML формат

Методът връща в символен низ съдържанието на обект DataSet в XML формат.
Декларация (C#):
public string GetXml()

7.2.11. Метод GetXmlSchema – Четене в символен низ на структурата на данните от обект DataSet в XML формат

Методът връща в символен низ структурата на данните от обект DataSet в XML формат.
Декларация (C#):
public string GetXmlSchema ()

7.2.12. Метод ReadXmlSchema – Четене на схемата на данните от XML файл в DataSet

Методът чете само схемата на XML данни от файл и ги записва в DataSet и, като опция, записва схемата на данните от файла в DataSet. Това означава, че ако XML файлът съдържа таблици с данни, в DataSet ще се запише само структурата на таблиците, но без данни в тях.
Декларация (C#):
public XmlReadMode ReadXml (string fileName)
където:
fileName задава спецификацията на файла

7.2.13. Метод WriteXml – Запис на данни от DataSet в XML файл

Методът записва XML данни и, като опция, схемата на данните в DataSet във файл.
Декларация (C#):
public void WriteXml (string fileName [, XmlWriteMode mode])
където:
fileName задава спецификацията на файла
а незадължителният параметър mode определя дали ще се записват само данните или данните и XML схемата на съдържанието на DataSet

7.2.14. Метод WriteXmlSchema – Запис на структурата на данните от DataSet в XML файл

Методът записва структурата на данните (структурата на таблиците и връзките между тях) от XML файл.
Декларация (C#):
public void WriteXml (string fileName)
където:
fileName задава спецификацията на файла

8. Прочитане и актуализиране на данни. Класове OleDbDataAdapter и SqlDataAdapter

Обектите DataAdapter от класове OleDbDataAdapter и SqlDataAdapter служат като посредник между обекта DataSet и източника на данни, като предоставят методи за извличане и актуализиране на данни. Методът Fill на обект DataAdapter извлича

данни от източника и ги зарежда в DataSet, а методът Update записва измененията, направени в DataSet обратно в източника на данни.

Когато обектът DataAdapter зарежда данни в DataSet, той създава необходимите таблици и полета в тях ако те вече не съществуват. Важно е да се запомни, че информацията за първичните ключове на таблиците ще бъде записана само ако атрибутът MissingSchemaAction има стойност AddWithKey. Класовете OleDbDataAdapter и SqlDataAdapter предоставят метод FillSchema, с който в DataSet може да се запише информацията за първичните ключове преди да бъдат записани данните. В случай, че самият доставчик на данни (напр. MSDataShape provider) не връща информация за първичните ключове, програмистът трябва да зададе със свой код стойности на атрибутите PrimaryKey за съответните полета от таблиците в DataSet.

За актуализирането на данните в DataSet класовете OleDbDataAdapter и SqlDataAdapter предоставят атрибути SelectCommand, InsertCommand, DeleteCommand, UpdateCommand, и TableMappings.

```
string mySelectText = "SELECT * FROM CUSTOMERS";
string myConnString = "Provider=SQLOLEDB;Data Source=localhost;" +
                      "Integrated Security=SSPI;Initial
Catalog=northwind";
OleDbDataAdapter custDA = new OleDbDataAdapter(mySelectText,
myConnString);
```

ADO.NET Пример 9- Създаване на обект **DataAdapter** с конструктор с два параметъра - низ, съдържащ SQL команда Select и низ, съдържащ параметри на връзката.

Конструктори:

a) Без параметри:
   public OleDbDataAdapter();
   public SqlDataAdapter();

b) С параметър - обект Command за SQL команда Select
   public OleDbDataAdapter(OleDbCommand);
   public SqlDataAdapter(OleDbCommand);

c) С параметри низ, съдържащ SQL команда Select и обект Connection
   public OleDbDataAdapter(string, OleDbConnection);

public SqlDataAdapter (string, OleDbConnection);

d) С параметри низ, съдържащ SQL команда Select и низ, съдържащ параметри на връзката
public OleDbDataAdapter(string, string);
public SqlDataAdapter (string, string);

8.1. Извличане на данни в DataSet. Метод Fill - класове OleDbDataAdapter и SqlDataAdapter

Методът добавя или актуализира съдържанието на записи в обект DataSet или в обект DataTable от DataSet.

Декларации: Методът има много декларации, които използват различни по брой и вид параметри. По-долу са дадени тези, които се използват най-често:

[C#] **public override int Fill(DataSet);** //попълва DataSet с данни, извлечени от източника

[C#] **public int Fill(DataTable);** //попълва таблица от DataSet с данни, извлечени от източника

[C#] **public int Fill(DataSet, string);** //попълва таблица от DataSet с данни, извлечени от източника по зададено име на таблицата (втори параметър)

[C#] **protected virtual int Fill(DataTable, IDataReader);** //попълва DataSet с данни, извлечени от източника чрез обект DataReader.

Методът Fill извлича данни от източника като използва команда SELECT. Обектът Connection за връзка с източника трябва да бъде създаден, но не е задължително връзката да бъде отворена. Ако връзката с източника не е отворена преди изпълнението на метода Fill, тя се отваря, и след извличането на данни се затваря отново. Ако връзката е била отворена преди изпълнението на метода Fill, тя остава отворена.

Ако възникне грешка при извличането на данните, то операцията се прекратява и в DataSet остават само данните, извлечени преди възникването на грешка.

Ако от източника не бъде извлечен нито един запис, то не се добавят таблици към DataSet, но също така не се генерира грешка.

Ако в извлечените записи има полета с повтарящи се имена, обектът DbDataAdapter добавя към дублиращите се имена пореден номер по схемата, "columnname1", "columnname2", "columnname3", и т.н.. Ако в записите има неименовани полета, те се записват в DataSet като се именуват по схемата "Column1", "Column2", ….

Когато изпълняваната заявка към БД връща няколко извадки (няколко динамични таблици), тогава всяка извадка се записва в DataSet като отделен обект DataTable. Имената на таблиците се получават, като към зададеното име на първата таблица се добавя пореден номер (например, "Table", "Table1", "Table2", и т.н.). За заявките, които не връщат извадки (например за SQL команда Insert) таблици не се създават.

Ако при изпълнение на SQL команда, която връща много извадки, се генерира грешка, изпълнението на командата се прекратява и останалите таблици не се записват в DataSet.

8.2. Запис на измененията от DataSet в източника на данни Метод Update - класове OleDbDataAdapter и SqlDataAdapter

Методът извиква съответните команди INSERT, UPDATE, or DELETE за всеки вмъкнат, променен или изтрит запис в DataSet.

Декларации

[C#] public int Update(DataRow[]); //Запис на измененията в масив от обекти DataRow

[C#] public override int Update(DataSet); //Запис на измененията в целия DataSet

[C#] public int Update(DataTable); //Запис на измененията в обект DataTable

[C#] public int Update(DataSet, string); //string съдържа име на таблица

Даденият по-долу пример съдържа код, с който се актуализира съдържанието на таблица в БД чрез обект DataAdapter.

```
try{
    DataGrid1.Update(); //записва измененията от DataGrid в DataSet
    sqlDataAdapter1.Update(DataSet1,"Address");
```

```
}catch(Exception ex ){
      Response.Write(“Error: “ + ex.ToString);
}
```

ADO.NET Пример 10 Запис на измененията от обект DataSet в източника на данни чрез метода Update на обект DataAdapter.

9. Генериране на команди и параметри за връзка със свързани контроли – контрол SqlDataSource

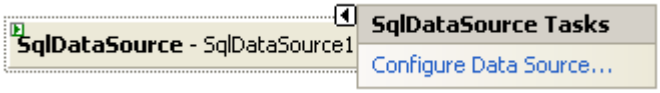
Контролът SqlDataSource се използва като посредник между SQL релационна база от данни и свързаните към източник на данни контроли (data-bound controls) GridView, DataList, DataListView, DetailsView и FormView. Той е удобен за използване тогава, когато още при проектирането на ASP.NET приложението е известен източника на данни. SqlDataSource предоставя в режим на проектиране помощни средства за генериране на параметри на връзката и SQL команди за основните операции над източника на данни. Комбинацията DataSource контрол и свързан контрол дава възможност за извеждане, сортиране и актуализиране на данни без да е необходимо да се въвежда програмен код. При база от данни на Acccess може да се използва и неговият наследник AcccessDataSource.

9.1. Задаване на вида на извлечените данни – атрибут DataSourceMode

Обектът SqlDataSource предоставя две възможности за извличане на записи от източника на данни – в обект DataSet и в обект DataReader. Режимът се задава чрез **DataSourceMode**. Атрибутът е от изброим тип SqlDataSourceMode, със стойности съответно DataSet и DataReader за двата режима на извличане на записите.

9.2. Свързване към източника на данни – атрибут ConnectionString

Задава и връща параметрите на връзката на SqlDataSource към източника и доставчика на данни. SqlDataSource може да използва ADO.NET доставчиците, SqlClient, OleDb, Odbc, и OracleClient (за Oracle 8.1.7 и следващи версии) , чрез които може да се свърже на практика към всеки един тип релационна база от данни.
Net Framework предоставя помощно средство (Wizard) за създаване на низ ConnectionString с параметри на връзка.



Фигура 1 Вид на SqlDataSource контрол при включването му в ASP.NET модул в режим Дизайн.

При конфигуриране на връзката, помощната програма предлага да съхрани с определено име низа ConnectionString в конфигурационния файл Web.Config. Препоръчително е да се съхраняват низовете с параметрите на връзка във Web.Config, защото при евентуално променяне на параметрите те ще трябва се коригират само във Web.Config, а не във всеки свързан контрол поотделно. Програмистът може да прочете съдържанието на низ, съхранен във файла Web.Config, чрез атрибута AppSettings на класа ConfigurationManager, както е показано в дадения по-долу пример

```
string strConnectionString = System.Configuration.ConfigurationManager
.AppSettings["SQLConnectionString"];
```

ADO.NET Пример 11 Прочитане на низ ConnectionString от конфигурационния файл Web.Config

9.3. Извличане на записи – атрибути SelectCommandType, SelectCommand, SelectQuery и метод Select (обект SqlDataSource)

9.3.1. Атрибут SelectCommandType (тип string)

Задава и връща символен низ, който определя типа на командата в атрибута SelectCommand. Валидните стойности са “Text” ако атрибута SelectCommand съдържа SQL команда, и “StoredProcedure” ако съдържа име на съхранена процедура.

9.3.2. Атрибут SelectCommand (тип string)

Задава и връща командата (като символен низ), чрез която се извличат записи от източника на данни.

9.3.3. Атрибут SelectQuery

Този атрибут е достъпен само в режим “Дизайн” чрез прозореца “Properties”. Стартира помощна програма (Wizard) за генериране на

ConnectionString, която дава възможност за избор на доставчик (сървър) и източник (база от данни).

9.3.4. Метод Select

Методът Select извлича записи от източника на данни, като използва зададената в атрибута **SelectCommand** команда и параметрите, зададени в колекцията SelectParameters. Методът се извиква автоматично при заявяване на ASP.NET модула ако са зададени стойности на атрибутите ConnectionString, SelectCommand и SelectCommandType. Преди изпълнението на метода контролът генерира събитие Selecting на контрола SqlDataSource, което програмистът може да използва, за да актуализира атрибутите и параметрите, свързани с извличането на записи. След изпълнението на метода се генерира събитието Selected.

Контролът SqlDataSource извлича данни от източника всеки път, когато се извиква методът Select на контрола. Този метод предоставя програмен достъп до метода, който се задава като стойност на атрибута SelectMethod. Методът Select се извиква автоматично от контролите, свързани с контрола SqlDataSource когато се извиква техният метод DataBind ако атрибутът DataSourceID на свързан контрол съдържа идентификатора ID на контрола SqlDataSource. Като алтернатива ID на SqlDataSource може да се запише в атрибута DataSource на свързания контрол, но тогава за зареждане на данните трябва да се изпълни метода DataBind на свързания контрол. Some examples of data-bound controls that can use SqlDataSource are DataGrid, DetailsView, DataList, and DropDownList. Методът Select може да се извиква програмно по всяко време за да се извличат данни от източника.

9.4. Атрибути, методи и колекции, свързани с актуализиране на записи

Обектът SqlDataSource предоставя атрибути и методи за основните операции със записи – добавяне, изтриване и актуализиране. Както се вижда от дадения по-долу списък, атрибутите са подобни по значение на тези, които са свързани с извличането на записи.

Атрибути	
DeleteCommand	(тип string) Задава и връща командата за

	изтриване на записи
InsertCommand	(тип string) Задава и връща командата за добавяне на записи
UpdateCommand	(тип string) Задава и връща командата за актуализиране на записи
DeleteCommandType	(изброим тип) Задава и връща вида на командата за изтриване на записи – текст или съхранена процедура.
InsertCommandType	(изброим тип) Задава и връща вида на командата за добавяне на записи – текст или съхранена процедура.
UpdateCommandType	(изброим тип) Задава и връща вида на командата за актуализиране на записи – текст или съхранена процедура.
DeleteQuery	(достъпен само в режим Дизайн) – стартира помощна програма за генериране на команда за изтриване на записи.
InsertQuery	достъпен само в режим Дизайн) – стартира помощна програма за генериране на команда за добавяне на записи.
UpdateQuery	достъпен само в режим Дизайн) – стартира помощна програма за генериране на команда за актуализиране на записи.

Методи	
Delete	Изтрива записи като използва командата, зададена в атрибута DeleteCommand
Insert	Добавя записи като използва командата, зададена в атрибута InsertCommand
Update	Актуализира записи като използва командата, зададена в атрибута UpdateCommand

Методите дават възможност за програмно изпълнение на основните операции над източника от данни. Методите използват генерираните от SqlDataSource параметрични команди DeleteCommand, InsertCommand и UpdateCommand.

Колекции	
DeleteParameters	Колекция от параметри за командата

	DeleteCommand
InsertParameters	Колекция от параметри за командата InsertCommand
UpdateParameters	Колекция от параметри за командата UpdateCommand

Колекциите на обект SqlDataSource предоставят достъп по параметрите на командите за изпълнение на операции над източника на данни. По правило SqlDataSource работи съвместно със свързан контрол, напр. GridView, който задава стойности на параметрите на командата (напр. UpdateCommand) за дадена операция и извиква съответния метод (напр. Update).

```
SqlDataSource1.UpdateParameters["id"].DefaultValue = user_id.Text;
SqlDataSource1. UpdateParameters ["email"].DefaultValue = email.Text;
SqlDataSource1.Update(); //Изпълнение на операция Update
```

ADO.NET Пример 12 Задаване на параметри за операция Update чрез колекцията UpdateParameters на SqlDataSource

9.5. Необходимост от корекция на текста на командата InsertCommand

Това е едно интересно недоглеждане на създателите на контрола SqlDataSource. Генерираната параметризирана команда за добавяне на запис и параметрите, свързани с нея, предполагат записване на стойност и на полето първичен ключ (primary key). Това води до грешка при опит за добавяте на запис, тъй като по правило първичният ключ е от тип, (напр. Autoincrement) при който му се генерира уникална стойност от сървъра за базата от данни. По-долу е даден пример за команда InsertCommand и параметри за нея за таблица от БД на Access, която има първичен ключ – поле id от тип AutoIncrement и две текстови полета ime и email. Този код е включен в кода на контрола SqlDataSource в .aspx страницата.

```
InsertCommand="INSERT INTO [Address] ([id], [ime], [email]) VALUES
(?, ?, ?)"
```

```
.....................................................................................................
..........
        <InsertParameters>
          <asp:Parameter Name="id" Type="Int32" />
          <asp:Parameter Name="ime" Type="String" />
          <asp:Parameter Name="email" Type="String" />
        </InsertParameters>
```

ADO.NET Пример 13 Команда и параметри за добавяне на запис, генерирани от контрол SqlDataSource.

При използване на тази команда, например от контрол FormView, се генерира грешка заради записа в полето id. За да се избегне това е необходимо да се отвори в режим “Source” .aspx страницата и ръчно да се изтрият от InsertCommand полето id и неговият параметър “?”, а също и съответният параметър от секцията InsertParameters. След тази корекция кодът ще добие вида:

```
InsertCommand="INSERT INTO [Address] ([ime], [email]) VALUES (?, ?)"
.....................................................................................................
..........
        <InsertParameters>
          <asp:Parameter Name="ime" Type="String" />
          <asp:Parameter Name="email" Type="String" />
        </InsertParameters>
```

ADO.NET Пример 14 Коригирани команда и параметри за добавяне на запис в кода на контрол SqlDataSource.

и командата може без проблеми да бъде използвана за добавяте на запис в таблицата от БД.

Пример:

```
using System.Text.RegularExpressions; //Именно пространство за
                                      //регулярни изрази
public partial class _Default : System.Web.UI.Page
{  //Регулярен израз за проверка на e-mail адрес
   Regex mail_pattern =
new Regex(@"\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*");
```

```
// Събитийна процедура на SqlDataSource, която се изпълнява преди
Update
   protected void SqlDataSource1_Updating(object sender,
SqlDataSourceCommandEventArgs e)
   {  string err_mess = "";
      try
      {  Label1.Text= err_mess; //Инициализация на съобщението за
грешка
         // Проверка за въведената стойност за полето ime
         err_mess = "Въведете име";
         string new_ime = e.Command.Parameters["ime"].Value.ToString();
         if (new_ime == "") {
            e.Cancel = true;
            Label1.Text = err_mess;
         }
         // Проверка за въведената стойност за полето email
         err_mess = "Въведете емейл";
         string new_email =
e.Command.Parameters["email"].Value.ToString();
         if (new_email == "") {
            e.Cancel = true;
            Label1.Text = err_mess;
         }
         // Проверка на въведената стойност за email с регулярен израз
         if(!mail_pattern.IsMatch(new_email)){
            e.Cancel = true;
            Label1.Text = "Въведете валиден е-мейл адрес";
         }
      }
      catch (Exception ex) {
         e.Cancel = true;
         Label1.Text = err_mess;
      }
   }
```

ADO.NET Пример 15 Проверка за валидност на актуализираните данни чрез събитийната процедура Updating на контрола SqlDataSource

9.6. Събитийни процедури на контрол SQLDataSource

Независимо от това, че контролът SQLDataSource няма графичен интерфейс (т.е. не се визуализира в .aspx страницата), той той генерира значителен брой събития, които обаче са свързани не с действия на оператора, а с операциите, които се изпълняват над източника и доставчика на данни чрез контрола. По-долу е дадено кратко описание на тези събития. Те са привързани към операциите, изпълняване от контрола (изпълняват се непосредствено преди или след определена операция) и могат да бъдат използвани за изпълнение на код, създаден от програмиста. Вторият параметър на събитийните процедури, които се изпълняват преди изпълнението на операция от SQLDataSource е обект от клас – наследник на класа CancelEventArgs и притежава атрибут Cancel. Задаването на стойност true на този атрибут отменя изпълнението на предстоящата операция

Събитиятята, които се генерират след изпълнение на операция от SQLDataSource са свързани със събитийни процедури, които предоставят втори параметър от тип SqlDataSourceStatusEventArgs, който предоставя атрибут AffectedRows, който връща броя на записите, които участват в операцията.

Вторият параметър на всички събитийни процедури притежава атрибут Command, който връща обект Command, представящ командата, която ще се изпълни или е изпълнена над източника на данни. Да си припомним, че обекта Command притежава колекция Parameters, чрез която получаваме достъп до параметрите на командата, в случай че използваме параметризирана заявка, или съхранена процедура с параметри.

DataBinding	Генерира се когато контролът се свързва с източника на данни. (наследен от класа Control.)
Deleted	Генерира се когато операцията изтриване (Delete) е изпълнена.
Deleting	Генерира се преди операция Delete.
Disposed	Генерира се когато сървърният контрол се премахва от паметта, което е последното събитие от жизнения цикъл на контрола, стартиран със заявяването на ASP.NET модула. (наследен от класа Control.)
Filtering	Генерира се before a filter operation.
Init	Генерира се когато сървърният контрол се инициализира, което е първата стъпка от жизнения цикъл на контрола.

	(наследен от класа Control)
Inserted	Генерира се когато операцията добавя insert operation has completed.
Inserting	Генерира се преди операция вмъкване на запис (Insert).
Load	Генерира се когато контролът се зарежда в .aspx страницата. (наследен от класа Control)
PreRender	Генерира се когато контролът се зарежда в .aspx страницата, но преди изобразяването му (наследен от класа Control)
Selected	Генерира се когато извличането на записи е завършено.
Selecting	Генерира се преди операция извличане на записи (Select).
Unload	Генерира се когато сървърният контрол се премахва от паметта. (наследен от класа Control)
Updated	Генерира се след като операцията update се изпълнява.
Updating	Генерира се преди изпълнението на операцията update.

10. Динамично генериране на SQL команди за актуализация на БД – класове OleDbCommandBuilder и SQLCommandBuilder

Обектите от класове OleDbCommandBuilder и SQLCommandBuilder предоставят възможност за динамично генериране на SQL командите DeleteCommand, InsertCommand, и UpdateCommand, които са необходими на обекта DataAdapter за актуализация на записите в база от данни. Генерирането на тези команди чрез обекти CommandBuilder може да бъде използвано само тогава, когато извлечените записи и съответно таблицата DataTable в обект DataSet са получени само от една таблица в базата от данни.

За да може обектът CommandBuilder да генерира командите DeleteCommand, InsertCommand, и UpdateCommand, трябва да бъде зададена SQL командата SELECT, чрез която да се извличат полетата, за които ще се задават стойности при изпълнение на SQL командите INSERT и UPDATE. За командата SELECT се създава обект Command, който се асоциира с обект DataAdapter, както едоказано в дадения по-долу пример.

```
adapter.SelectCommand = new OleDbCommand("Select id, ime from
Address", con); //Създаване на обект Command за извличане на записи
OleDbCommandBuilder builder = new OleDbCommandBuilder(adapter);
```

ADO.NET Пример 16 Създаване на обекти Command и CommandBuilder, асоциирани с DataAdapter

Важно е да се запомни също така, че командата SELECT трябва да извлича първичния ключ на записите (primary key) от таблицата, която ще се актуализира. Ако командата SELECT не извлича първичния ключ, при опит за получаване на команди от обект CommandBuilder, се генерира изключение InvalidOperationException. По-долу е даден код на събитийна процедура, който създава връзка към база от данни на Access, извлича записите от таблица Address в обект DataSet, променя съдържанието на полето "ime" в таблицата в DataSet и записва измененияята в БД чрез метод Update на обект DataAdapter, като използва динамично генерирана SQL команда Update от обект CommandBuilder.

```
protected void Button1_Click(object sender, EventArgs e)
{
   try
   {
      //Създаване на връзка
      OleDbConnection con = new
      OleDbConnection(@"Provider=Microsoft.Jet.OLEDB.4.0;Data
      Source=F:\Members.mdb");
      con.Open();
      Label1.Text += "<BR>Connection Successful";
      //
      //Създаване на обекти DataSet, DataAdapter и Command
      DataSet ds = new DataSet("ds"); //Създаване на обект DataSet
      OleDbDataAdapter adapter = new OleDbDataAdapter();
      adapter.SelectCommand = new OleDbCommand("Select id, ime from
      Address", con); //Създаване на обект Command за извличане на
записи
      //
```

```
      // Създаване на обект CommandBuilder, асоцииран с обекта
DataAdapter
      OleDbCommandBuilder builder = new
OleDbCommandBuilder(adapter);
      adapter.Fill(ds, "Address");      //Запис на таблицата в DataSet
         //
         //Променяне на поле от първи запис таблица Address
      DataRow row=ds.Tables["Address"].Rows[0];
      row[1] = TextBox1.Text;  //Запис на стойност, въведена в TextBox1
      //
      //Запис на измененията в БД
      DataSet new_ds = ds.GetChanges();  //Запис на измененията в нов
DataSet
         adapter.Update(new_ds, "Address");  //запис в БД чрез DataAdapter
         ds.AcceptChanges();
         con.Close(); //Затваряне на връзката
   }
   catch (Exception ex) //Прихващане на грешка
   {
      Label1.Text += "<BR>Error: " + ex.ToString(); //Извеждане на
грешката
   }
}
```

ADO.NET Пример 17 Актуализиране на запис в база от данни посредством динамично генерирана команда UPDATE от обект CommandBuilder.

Когато се асоциира с DataAdapter, обектът CommandBuilder автоматично генерира SQL команди, които се присвояват на атрибутите InsertCommand, UpdateCommand, и DeleteCommand на DataAdapter ако те съдържат стойност null. Атрибутите InsertCommand, UpdateCommand, и DeleteCommand, на които вече е била присвоена SQL команда запазват стойността си.

Изгледите (views) от базата от данни, които са създадени чрез обединяване на данни от няколко таблици в БД не се считат за една таблица и за тях не може да се генерират команди чрез обект CommandBuilder.

10.1. Атрибути на обект CommandBuilder.

10.1.1. Атрибут DataAdapter – задава и връща обект DataAdapter, със който се асоциира обектът CommandBuilder.Ако обектът CommandBuilder се създаде с празен конструктор, то той трябва да се асоциира с обект DataAdapter с този атрибут.

```
OleDbCommandBuilder builder = new OleDbCommandBuilder();
builder.DataAdapter = adapter;
```

ADO.NET Пример 18 Асоцииране на обект CommandBuilder с обект DataAdapter чрез атрибута *DataAdapter*

10.1.2. Атрибут SetAllValues – (тип bool) определя дали в генерираната команда UPDATE ще участват всички полета от командата SELECT (стойност true) или само тези, в които има промент (стойност false).

10.2. Методи на обект CommandBuilder.

10.2.1. Методи, които връщат генерираните команди - GetInsertCommand( ),GetDeleteCommand( ) и GetUpdateCommand( )

Методите връщат обект Command, който представя съответната генерирана команда от обект CommandBuilder. Ако се изпълнят с параметър от тип bool, при стойност true се генерират параметри на обекта Command с имена, получени от имената на полетата от командата (ако това е възможно), а при стойност false се генерират параметри с имена @p1, @p2, и т.н.

Можем да използваме методите за да променим някои от параметрите на генерираните от CommandBuilder обекти Command. Например можем да променим стойността на атрибута CommandTimeout на обекта Command, получен от метода GetUpdateCommand( ) и след това да присвоим променения обект на асоциирания с CommandBuilder обект DataAdapter.

11. Свързани WEB сървърни контроли

Свързаните Web сървърни контроли са контроли, които могат да бъдат свързани към контрол-източник и дават възможност за извеждане

и актуализиране на данни в ASP.NET приложенията. Свързаните контроли са съставни (контейнерни) контроли, които включват в себе си други Web контроли като Label и TextBox. Класовете за контролите са декларирани в именното пространство System.Web.UI.WebControls

Външния вид и съдържанието на свързаните контроли може да се променя посредством т.н. шаблони (темплейти). Освен това свързаните контроли предоставят голям набор от атрибути, методи и събития, които програмистът може да използва. По-долу са разгледани общите атрибути, методи и събития на свързаните контроли, класифицирани според предназначението им.

11.1. Създаване на връзка между свързан контрол и източник на данни. Атрибути DataSourceID, DataSource, DataMember и метод DataBind

Свързаните контроли могат да използват като източник на данни контроли-източници като ObjectDataSource или SqlDataSource. Контролът-източник се свързва към източника на данни, напр. база от данни или обект-източник на данни. Създаването на връзка между свързан контрол и контрол-източник става, като в атрибута **DataSourceID** на свързания контрол се запише стойността на идентификатора ID на контрола-източник. Данните, предоставяни от контрола-източник се извеждат в свързания контрол. Двойката свързан контрол/контрол-източник дава възможност за въвеждане и актуализиране на данните в повечето случаи без да е необходим допълнителен код или с минимален код от разработчика.

В ASP.NET версии 1.0 и 1.1, създаването на връзка между свързан контрол и контрол-източник става със запис в атрибута **DataSource** на стойността на обектна променлива на контрол-източник, и изпълнение на метода **DataBind()**. В ASP.NET версия 2.0 този алгоритъм за свързване (както и създадения преди код), могат да продължават да се използват, но се препоръчва свързване чрез атрибута DataSourceID.

Атрибутът **DataMember** задава и връща списък от данни, към който се свързва контролът. Необходимо е се зададе стойност на този атрибут тогава, когато източникът на данни съдържа повече от един списъчен обект с данни. Напромер когато източник на данни е контролът DataSource и той съдържа повече от една таблица, на атрибута DataMember трябва да се присвои низ с името на таблицата, към която да се свърже контрола.

Даденият по-долу пример съдържа код, с който контролът GridView се свързва към база от данни на Access. Данните се извличат от таблица Address и се записват в контрол DataSet чрез метода Fill на обект DataAdapter. Контролът GridView се свързва с DataSet чрез атрибута DataSource, а с таблицата Address – чрез атрибута DataMember. Извличането на данните става с метода DataBind.

```
DataSet ds = new DataSet();
OleDbConnection con = new
    OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0.;Data Source="
    + Server.MapPath("App_Data\\Members.mdb"));
OleDbDataAdapter da = new OleDbDataAdapter("Select * from Address",
    con);
da.Fill(ds, "Address"); //Fill data to DataSet table Address
GridView1.DataSource = ds;
GridView1.DataMember = "Address";
GridView1.DataBind();
```

ADO.NET Пример 19 Извличане на данни в контрол GridView чрез атрибутите DataSource и DataMember и метода DataBind

Когато низът с параметри за връзка към БД е записан в конфигурационния файл Web.Config (което е за предпочитане), в кода на свързания контрол в .aspx страницата, той може да се присвои на съответния параметър на свързания контрол с Write блок както следва:

```
ConnectionString="<%$ ConnectionStrings:MyConnectionString %>"
```

11.2. Общи атрибути на свързаните контроли

Свързаните контроли са наследници на класа WebControl и наследяват от него набор от атрибути основно имащи отношение към графичното представяне на контрола. По-долу ще разгледаме по-подробно само атрибутите, които са специфични за свързаните контроли.

11.2.1. Атрибути за форматиране

Атрибутите връщат референция към обект от клас TableItemStyle, които дава възможност за форматиране на ред с определено предназначение на свързания контрол.

RowStyle	Връща референция към обект , който дава възможност за форматиране на редовете в контрола GridView.
EditRowStyle	Връща референция към обект, който дава възможност за форматиране на реда за редактиране (функция Edit) на полета в текущия запис.
HeaderStyle	Връща референция към обект, който дава възможност за форматиране на заглавния ред (Header) на свързания контрол.
FooterStyle	Връща референция към обект, който дава възможност за форматиране на последния ред (Footer) на свързания контрол.
EmptyDataRowStyle	Връща референция към обект, който дава възможност за форматиране на празен (без данни) ред от свързания контрол.
GridLines	Задава и връща стил за вътрешната рамка на свързания контрол. Атрубутът е от изброим тип GridLines със стойности Both, Horizontal, Vertical, None.
HorizontalAlign	Задава и връща хоризонталното подравняване на свързания контрол.

Обектът TableItemStyle притежава атрибути, аналогични на атрибутите, с които се задава форматирането на WEB контролите. Стойностите на атрибутите на редовете могат да се променят както програмно, така и в режим Дизайн чрез прозореца Properties.

```
GridView1.HeaderStyle.BackColor = System.Drawing.Color.DarkBlue;
```

ADO.NET Пример 20 Програмно задаване на фонов цвят на заглавния ред на свързан контрол

В примера се използва атрибута BackColor на обект HeaderStyle за да се промени фоновия цвят на свързания контрол GridView1. По

правило стойности на атрибутите за форматиране се задават в режим на проектиране чрез прозореца Properties.

11.2.2. Атрибути за управление на елементи от потребителския интерфейс

Атрибутите задават и връщат логическа стойност, които определя ще бъдат създадени и изведени елементи (полета, бутони) от интерфейса на свързания контрол.

HeaderRow	Връща обект (от клас, наследник на TableRow) , който представя заглавния ред (Header) на свързания контрол. Типът на обекта се определя от типа на контрола.
FooterRow	Връща обект (от клас, наследник на TableRow) , който представя последния ред (Footer) на свързания контрол. Типът на обекта се определя от типа на контрола.
AllowPaging	Задава и връща стойност, която определя дали ще бъде достъпно извеждането на данните по страници.
Caption	Задава и връща текста, който се извежда като HTML заглавен надпис на свързания контрол.
CaptionAlign	Задава и връща хоризонталната и вертикална позиция на текста, който се извежда като HTML заглавен надпис на свързания контрол.
EmptyDataText	Задава и връща текст, който ще се изведе, ако източникът на данни на свързания контрол не съдържа записи.

11.2.3. Атрибути за колекции

Controls	Връща колекция от дъщерни контроли в свързания контрол.

11.2.4. Атрибути, свързани със страницирането

TopPagerRow	Връща обект от клас, наследник на TableRow (в зависимост от вида на контрола), който представя горен ред за превключване за превключване на страниците на свързания контрол. Редът се извежда, ако атрибута AllowPaging има стойност true.
BottomPagerRow	Връща обект от клас, наследник на TableRow, който представя долен ред за превключване на страниците на свързания контрол. Редът се извежда, ако атрибута AllowPaging има стойност true.
PageCount	Връща броя на страниците, необходими за извеждането на записите от източника на данни в свързания контрол.
PageIndex	Задава и връща индекса на текущо извежданата страница.
PagerSettings	Връща обект от клас PagerSettings, който предоставя достъп до атрибутите на бутоните в реда за превключване на страниците.
PagerStyle	Връща обект от клас TableItemStyle, който дава възможност за форматиране на реда за превключване на страниците.
PagerTemplate	Задава и връща потребителското съдържание на реда за превключване на страниците
PageSize	Задава и връща броя на записите, които се извеждат в една страница на свързания контрол.

11.2.5. Достъп до родителския WEB контрол – атрибут NamingContainer

Всяка .aspx страница в ASP.NET Web приложението съдържа йерархия от контроли. Тази йерархия не зависи от това дали контролът генерира графичен интерфейс, видим за потребителя. Именованият контейнер NamingContainer за даден WEB контрол е родителския контрол, който е над него в йерархията и поддържа интерфейса INamingContainer. WEB сървърният контрол, който поддържа този интерфейс, създава уникално именно пространство за атрибутите ID на неговите дъщерни сървърни контроли.

Създаването на уникално именно пространство за сървърни контроли е изключително важно когато свързваме данни към списъчен Web сървърен контрол, като сървърните контроли Repeater или DataList. Когато много записи от данни създават много инстанции на сървърен контрол , които е дъщерен на списъчния контрол, именованият контейнер дава възможност всяка инстанция на този дъщерен контрол да има стойност на атрибута UniqueID, която не предизвиква конфликт. За цялата .aspx страница именованият контейнер е обектът от клас Page, който се генерира, когато се заяви страницата.

Атрибутът NamingContainer не винаги връща същият обект както атрибутът Parent на WEB контрола. Например ако дъщерният контрол се съдържа в HTML таблица в темплейт на контрола GridView, тогава родителският контрол е клетката от таблицата (обект HtmlTableCell), но именованият контейнер е обект от клас GridViewRow, тъй като това е списъчния контрол, който създава именно пространство за дъщерните WEB контроли. Даденият по-долу пример съдържа събитийната процедура на бутон, включен в таблица, съдържаща се в темплейта EmptyDataTemplate.

```
protected void Button4_Click(object sender, EventArgs e)
{
      MessageLabel.Text  +=  ((Button)sender).Parent.GetType().ToString()
+"<BR>";
                                                MessageLabel.Text          +=
((Button)sender).NamingContainer.GetType().ToString()";
}
```

ADO.NET Пример 21 Използване на атрибута NamingContainer на WEB сървърен контрол.

Събитийната процедура извежда типа на обектите, получавани от атрибутите Parent и NamingContainer. Атрибутът Parent връща обект от тип System.Web.UI.WebControls.TableCell, а атрибутът NamingContainer – обект от тип System.Web.UI.WebControls.GridViewRow и следователно дава възмжност за достъп до реда от контрола GridView.

12. Извеждане и актуализиране на данни в табличен вид – свързан контрол GridView

Контролът GridView извежда данните от източника в табличен вид и предоставя възможност на оператора да сортира данните, да редактира и изтрива избран запис.

Контролът GridView е наследник на контрола DataGrid от първите версии на ASP.NET. Освен добавената възможност за използване на функционалността на контрола SqlDataSource, контролът GridView предлага и други нови възможности като като дефиниране на повече от едно поле като първичен ключ, използване на свързани полета и темплейти и нов модел за обработка на събития.

12.1. Атрибути на обект GridView

12.1.1. Атрибути за форматиране

Контролът GridView предоставя два специфични атрибута за форматиране, свързани с табличното извеждане на данните. Атрибутите връщат референция към обект от клас TableItemStyle, които дава възможност за форматиране на ред с определено предназначение на контрола GridView.

AlternatingRowStyle	Връща референция към обект, който дава възможност за форматиране на нечетните (при начален индекс 0) редове с данни в GridView контрола
SelectedRowStyle	Връща референция към обект, който дава възможност за форматиране на селектирания ред с данни в GridView контрола
RowHeaderCol	Задава и връща името на колоната, която трябва да се

umn	изведе като заглавна в контрола GridView. Текстът в колоната се извежда като удебелен и центриран.

12.1.2. Атрибути за управление на елементи от потребителския интерфейс

Атрибутите задават и връщат логическа стойност (тип bool), които определят дали ще бъдат създадени и изведени елементи (редове, полета, бутони) от интерфейса на на контрола GridView.

AutoGenerateColumns	Задава и връща стойност, която определя дали за полетата от източника на данни ще бъдат автомотично създадени свързани полета.
AutoGenerateDeleteButton	Задава и връща стойност, която определя дали ще бъде автоматично добавено поле CommandField column с бутон Delete към всеки ред от свързания контрол.
AutoGenerateEditButton	Задава и връща стойност, която определя дали ще бъде автоматично добавено поле CommandField column с бутон Edit към всеки ред от свързания контрол.
AutoGenerateSelectButton	Задава и връща стойност, която определя дали ще бъде автоматично добавено поле CommandField column с бутон Select към всеки ред от свързания контрол.
AllowSorting	Задава и връща стойност, която определя дали ще бъде включена фукцията за сортиране на контрола GridView.
ShowHeader	Задава и връща стойност, която определя дали ще бъде изведен заглавен ред (Header) на контрола GridView
ShowFooter	Задава и връща стойност, която определя дали ще бъде изведен последен ред (Footer) на контрола GridView

Инструкцията в дадения по-долу пример програмно разрешава извеждането на последен ред (Footer) на контрола GridView. При добавяне на контрола чрез дизайнера на Visual Studio .NET последният ред по подразбиране не се извежда, но може да бъде изведен чрез

задаване на стойност true на атрибута ShowFooter. Това може да се направи и в режим “Дизайн” чрез прозореца Properties.

```
GridView1 ShowFooter = true;
```

ADO.NET Пример 22 Извеждане на последен ред (Footer) на контрола GridView

Автоматичното генериране на елементи от потребителския интерфейс чрез атрибути AutoGenerate…….. има недостатъка, че те не могат да се форматират в режим “Дизайн”. Затова се предпочита добавянето на управляващи елементи (за операции Edit, Delete и Update) чрез дизайнера на Visual Studio .NET.

12.1.3. Атрибути за колекции

Columns	Връща колекция от обекти DataControlField, които представят колоните в контрола GridView.
DataKeys	Връща колекция от обекти DataKey които представят първичните ключове на всички даннови редове в контрола GridView.
DataKeyNames	Връща масив от тип string, който съдържа имената на полетата, които са първични ключове на записите, изведени в контрола GridView. Ако се извежда съдържанието на една таблица с един първичен ключ масивът ще съдържа само един елемент.
Rows	Връща колекция от обекти GridViewRow, които представят редовете в контрола GridView.

12.1.4. Събития и атрибути, свързани с бутоните на контрола GridView

При щракване с мишката върху бутон от контрола GridView се генерира събитието RowCommand. Вторият параметър на събитийната процедура за това събитие е от тип GridViewCommandEventArgs и предоставя на програмиста следните атрибути:

CommandArgument - наследен от CommandEventArgs	(тип string) Връща допълнителни параметри на командата (ако има такива. Например за командата за сортиране допълнителен параметър може да има стойност “Acceding” – в нарастващ ред..(Inherited from CommandEventArgs.)
CommandName - наследен от CommandEventArgs	(тип string) Връща името на командата, напр. "Sort", “Edit”, “Delete”, “Update”
CommandSource	(тип Object) Връща обектна променлива за контрола GridView, който съдържа бутона

12.1.5. Събития и атрибути за операция Select за контрола GridView

С операция Select (селектиране на ред) са свързани събитията:

SelectedIndex Changing	Генерира се при щракване върху бутона Select преди контролът GridView да изпълни операцията селектиране.
SelectedIndex Changed	Генерира се при щракване върху бутона Select след като контролът GridView изпълни операцията селектиране.

В събитийната процедура SelectedIndexChanged са достъпни следните атрибути:

SelectedDataKey	Връща обект DataKey, който представя първичния ключ на селектирания ред.
SelectedIndex	Връща индекса на селектирания ред.
SelectedRow	Връща обект GridViewRow, който представя селектирания ред.
SelectedValue	Връща стойността на ключа на избрания ред в контрола GridView.

```
protected void GridView1_SelectedIndexChanging(object sender,
GridViewSelectEventArgs e)
```

```
{
    for (int n = 0; n < GridView1.Rows.Count; n++)
    {  // Задаване на фонов цвят на всички редове на контрола
       GridView1.Rows[n].BackColor = System.Drawing.Color.White;
    }
    GridView1.Rows[e.NewSelectedIndex].BackColor =
    System.Drawing.Color.Yellow;  //Задаване на фонов цвят на сел. ред
}
```

ADO.NET Пример 23 Променяне на цвета на селектирания ред чрез атрибута NewSelectedIndex на обект GridViewSelectEventArgs

12.1.6. Събития и атрибути за режим Edit за контрола GridView

С операция Edit (редактиране на ред) е свързано събитието

и атрибутът:

EditIndex	Задава и връща индекса на реда, който се редактира. Атрибутът е достъпен след като обектът GridView е превключил в режим редактиране.

12.1.7. Събития и атрибути за операции Delete за контрола GridView

С операция Delete (изтриване на ред) са свързани събитията

RowDeleting	Генерира се при щракване върху бутона Delete преди контролът GridView да изтрие реда.
RowDeleted	Генерира се при щракване върху бутона Delete след като контролът GridView изтрие реда.

Събитийната процедура RowDeleting получава като втори параметър обект от клас GridViewDeleteEventArgs, които предоставя следните атрибути:

Cancel	Задава и връща стойност, която показва дали събитието ще бъде анулирано. При стойност true текущият ред не се изтрива.
Keys	Връща колекция от тип dictionary, съдържаща двойки ключ/стойност за първичните ключове на реда, който ще се изтрива.
RowIndex	Връща индекса на реда, който ще се изтрива.
Values	Връща колекция от тип dictionary, съдържаща двойки ключ/стойност за полетата (които не са първични ключове) на реда, който ще се изтрива.

12.1.8. Събития и атрибути за операция Update за контрола GridView

С операция Delete (изтриване на ред) са свързани събитията

RowUpdating	Генерира се при щракване върху бутона Update преди контролът GridView да актуализира реда.
RowUpdated	Генерира се при щракване върху бутона Update след като контролът GridView актуализира реда.

Събитийната процедура RowUpdating получава като втори параметър обект от клас GridView UpdateEventArgs, които предоставя следните атрибути:

Cancel	Задава и връща стойност, която показва дали събитието ще бъде анулирано. При стойност true текущият ред не се актуализира.
OldKeys	Връща колекция от тип dictionary, съдържаща оригиналните двойки ключ/стойност за първичните ключове на реда, който ще се актуализира.
NewKeys	Връща колекция от тип dictionary, съдържаща новите двойки ключ/стойност за първичните ключове на реда, който ще се актуализира.
RowIndex	Връща индекса на реда, който ще се актуализира.
OldValues	Връща колекция от тип dictionary, съдържаща оригиналните двойки ключ/стойност за полетата (които не са първични ключове) на реда, който ще се

	актуализира.
NewValues	Връща колекция от тип dictionary, съдържаща новите двойки ключ/стойност за полетата (които не са първични ключове) на реда, който ще се актуализира.

Както се вижда от таблицата, атрибутите съвпадата с тези на аргумента на събитийната процедура Deleting, с тази разлика, че за операция Update са достъпни по две двойки атрибути Keys и Values съответно за оригиналните (т. е. преди актуализирането) и новите стойности на полетата на записа.

13. Извеждане и актуализиране на данни в един запис – свързани контроли FormView и DetailsView

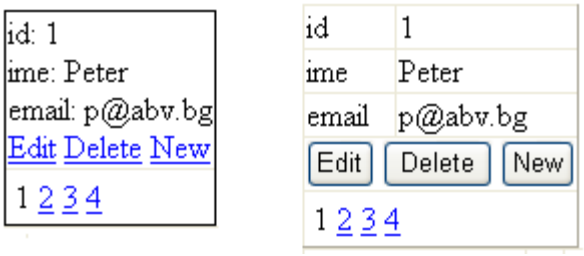
Свързаните контроли FormView и DetailsView извеждат единичен запис (FormView във вид на списък, а DetailsView – във вид на таблица) и предоставят възможност за преместване по записите, вмъкване, актуализиране и изтриване на запис. Те са по-подходящи от контрола GridView за актуализиране на записи в таблици с много полета, тъй като извеждат само един запис като списък с вертикално подреждане на елементите. При извеждането на записи с много полета в контрола GridView потребителят е принуден да скролира в хоризонтална посока страницата като във всеки момент вижда само част от полетата. Контролите FormView и DetailsView често се използват в сценарии “master-detail”, при които с master контрола (например GridView) се избира запис, който се извежда контрол FormView или DetailsView.

Съдържанието на контролите FormView и DetailsView се извеждат във WEB браузъра. Разликата между тях е, че контролът DetailsView извежда к всяко поле от записа в отделен ред от таблицата., докато контролът FormView извежда всички полета от записа се извеждат в една клетка.

13.1. Методи на контролите FormView и DetailsView

Контролите FormView и DetailsView предоставят набор от методи за основните операции над източника на данни – добавяне, изтриване и

актуализиране на записи. Контролите предоставят на потребителя на ASP.NET приложението управляващи елементи (бутони или хипервръзки) за избор на режима на функциониране – редактиране (Edit), изтриване (Delete) или добавяне (Insert).



Фиг. 6 Вид на контролите FormView и DetajlsView в режим ReadOnly

13.1.1. Променяне на режима на контрола – метод ChangeMode

Методът дава възможност за програмно променяне на режима на контрола. Той трябва да получи като параметър стойност от изброим тип (Enumeration) съответно FormViewMode или DetajlsViewMode.

```
FormView1.ChangeMode(FormViewMode.Edit);
```

ADO.NET Пример 24 Променяне на режима на контрол FormView чрез метода ChangeMode

13.1.2. Изпълнение на операции над източника на данни – методи UpdateItem, DeleteItem и InsertItem

Методите дават възможност за изпълнение на основните операции над източника на данни. По правило операциите Update, Delete и Insert се изпълняват когато контролът е в съответния режим (напр. Insert) и потребителят щракне с мишката върху бутона за съответната операция на контрола. Методите обаче са публични и могат да бъдат извиквани програмно.

Декларации (C#)

```
public virtual void UpdateItem (bool causesValidation)
public virtual void DeleteItem ()
public virtual void InsertItem (bool causesValidation)
```

където causesValidation при стойност true задава валидизиране на .aspx страницата преди изпълнение на метода. Той е от значение

тогава, когато разработчикът използва контроли за валидизация за проверка на валидността на въведените за актуализиране или добавяне данни.

```
FormView1.DeleteItem();
```

ADO.NET Пример 25 Изтриване на текущия запис в контрол FormView с метода DeleteItem

13.2. Събития на контролите FormView и DetailsView

Контролите FormView и DetailsView генерират двойки събития, свързани с изпълнението на операциите над източника на данни Update, Delete и Insert. За всяка от тези операции се генерира събитие преди нейното изпълнение и събитие след нейното изпълнение. Разработчикът може да

13.2.1. Събития, които се генерират преди изпълнението на операция над източника на данни

Събитията, които се генерират преди изпълнението на операция над източника на данни са:

ItemUpdating - Генерира се преди операция Update
ItemDeleting - Генерира се преди операция Delete
ItemInserting - Генерира се преди операция Insert

Събитийните процедури за тези събития имат като втори параметър обект от клас XxxViewUpdateEventArgs, XxxViewDeleteEventArgs или XxxViewInsertEventArgs, където Xxx е съответно Form за контрола FormView или Detajls за DetajlsView. Чрез този обект разработчикът получава атрибути, които дават възможност на програмиста да манипулира параметрите на операцията над източника на данни или да забрани нейното изпълнение.

13.2.2. Отменяне на операцията – атрибут Cancel

Вторият параметър на събитийните процедури е наследник на класа CancelEventArgs и притежава атрибут Cancel. Задаването на стойност true на този атрибут отменя изпълнението на предстоящата операция. В даденият по-долу пример атрибутът Cancel се използва за да се забрани изтриването на запис със стойност на първичния ключ id 1.

```
void FormView1_ItemDeleting(object sender, FormViewDeleteEventArgs e)
{
  if ((int)e.Keys["id"] == 1)
  {
    Response.Write("Not Allowed");
    e.Cancel = true; //Забраняване на изтриването на записа
  }
}
```

ADO.NET Пример 26 Използване на атрибута Cancel на FormViewDeleteEventArgs за забраняване на операцията Delete за id=1

Visual Studio .NET генерира автоматично код на събитийна процедура при щракване с мишката върху събитието в прозореца Properties.

13.2.3. Достъп до параметрите на операцията – атрибути Keys, Values, OldValues и NewValues

Атрибутите Keys, Values, OldValues и NewValues предоставят параметри на командата, която ще се изпълнява над източника на данни. Данните се предоставят във вид на индексирана колекция Key/Value (интерфейс IorderedDictionary) където Key и Value са от тип object. Разработчикът може да използва тези параметри в събитийните процедури за събитията, които се изпълняват преди изпълнението на операциите Delete, Update и Insert. Значението на атрибутите е както следва:

Keys - Връща колекция, съдържаща първичите ключове на текущия запис

Values - Връща колекция, съдържаща двойки име/стойност за полетата от текущия запис, който не са първични ключове.

OldValues - (само за операция Update) Връща колекция, съдържаща двойки име/първоначална_стойност за полетата от текущия запис, който не са първични ключове.

NewValues - (само за операция Update) Връща колекция, съдържаща

двойки име/нова_стойност за полетата от текущия запис. Тази колекция може да съдържа нови стойности за полетата, които са първични ключове, ако е разрешено тяхното променяне.

В даденият по-долу пример се използва колекцията Values за да се получи въведената стойност за полето ime при добавяне на нов запис с контрола FormView. Ако дължината на въведеният низ за ime е по-малка от 3 символа се отменя операцията Insert и се извежда съобщение в контрола MessageLabel

```
void FormView1_ItemInserting(object sender, FormViewInsertEventArgs e)
{
      string ime=(string)e.Values["ime"];
      //
      if (ime.Length < 3)
      {
         e.Cancel = true; // отменя операцията Insert
               MessageLabel.Text = "Името трябва да съдържа поне три символа";

      }
}
```

ADO.NET Пример 27 Проверка за валидност на въведени данни за операция Insert чрез колекцията Values

Обърнете внимание на кастинга (string) за преобразуване на типа на стойността на Values["ime"]. Той е задължителен, тъй като колекцията връща стойности от тип object, независимо от това, че полето ime в базата от данни е от символен тип.

13.2.4. Събития, които се генерират след изпълнението на операция над източника на данни

Събитията, които се генерират след изпълнението на операция над източника на данни са:

ItemUpdated - Генерира се след операция Update

ItemDeleted - Генерира се след операция Delete
ItemInserted - Генерира се след операция Insert

Събитийните процедури за тези събития имат като втори параметър обект от клас XxxViewUpdatedEventArgs, XxxViewDeletedEventArgs или XxxViewInsertedEventArgs, където Xxx е съответно Form за контрола FormView или Detajls за DetajlsView. Чрез този обект разработчикът получава атрибути, които дават възможност на програмиста да получи информация за изпълнението на операцията.

13.2.5. Получаване на броя на записите, участвали в операцията – атрибут AffectedRows

Атрибутът AffectedRows е вторият параметър на събитийните процедури на контролите FormView и DetajlsView за събитията, които се генерират след операциите Delete, Update и Insert. Той връща броя записи, участвали при изпълнението на операцията. Например за операция Delete този атрибут връща броя на изтритите записи, за операция Update – броя на актуализираните записи.

13.2.6. Проверка за грешка при изпълнение на операция над източника на данни – атрибут Exception

Атрибутът Exception на вторият параметър на събитийните процедури за събитията, които се генерират след операциите Delete, Update и Insert на контролите FormView и DetajlsView връща обект от клас Exception, ако е възникнала грешка. Обектът Exception предоставя информация за грешката, която програмистът може да използва. Ако не е възникнала грешка атрибутът връща стойност null (Nothing за VB.NET).

```
void FormView1_ItemDeleted(object sender, FormViewDeletedEventArgs
e)
{
    if (e.Exception == null)
    { //Извеждане на броя изтрити записи
        MessageLabel.Text = "Изтрити записи: " + e.AffectedRows;
    }
    else
```

```
    {//Извеждане на информация за изключението
        MessageLabel.Text = "Error=" + e.Exception.Message;
    }
}
```

ADO.NET Пример 28 Използване на атрибути на параметъра FormViewDeletedEventArgs на събитийна процедура ItemDeleted

Даденият по-горе пример съдържа код на събитийна процедура за събитието ItemDeleted на контрола FormView. Чрез атрибута Exception се проверява дали при изтриване на запис в контрола FormView е възникнала грешка. Ако операцията Delete е изпълнена, събитийната процедура извежда броя изтрити записи, който се съдържа в атрибута AffectedRows. Ако е възникнало изключение се извежда информация за него, получена от атрибута Message на обект Exception.

Web услуги

Една от основните задачи на Интернет приложенията е да улеснят комуникацията между фирмите чрез предоставяне на бърз достъп до информацията. В много случаи обаче, достъпът до бази от данни през интернет не удовлетворява адекватно бизнес потребностите на организациите. Програмируемите Web приложения, които предоставят не само обем информация, но и възможност за изпълнение на операции от доставчика по заявка на клиента са предпочитаното средство за обмен на информация в Интернет среда.

Web услугите са софтуерни системи, асоциирани с идентификатор на Интернет ресурс (URI), чийто публичен интерфейс и връзки са описани с XML. Тези системи могат да взаимодействат помежду си по начин, определен от техните достъпни дефиниции, като комуникацията е базирана на XML съобщения и Интернет протоколи.

Web услугите (Web Services) предоставят прост и гъвкав модел за създаване на приложения, който работят съвместно в интернет среда като използват вече създадената инфраструктура и приложения. Използването на локално разработени услуги от Web приложенията улеснява тяхното създаване като интегрира ресурси, реализирани на различни платформи и програмни среди.

1. Предимства на Web услугите

Web услугите притежават съществени предимства пред съществуващите преди това технологии за интегриране в мрежова среда на корпоративни приложения:

1.1. Възможност за комуникация на приложенията през интернет

Едно от големите предимства на Web услугите е това, че те са базирани на HTTP протокола. Така Web услугите се интегрират в постоянно разрастващата се мрежова среда от ресурси, публикувани в WWW.

1.2. Независимост от платформата и програмния език

Разработчиците могат да сглобяват приложения, които използват Web услугите съвместно с локални ресурси и програмен код. Например в едно приложение може да се използват съвместно Microsoft Passport service за идентификация на потребител, услуга на друг доставчик за пресонализация на потребителя и услуга за извършване на разплащания чрез кредитни карти. При това е без значение операционната система и програмния език, на който са реализирани тези услуги.

2. Основни компоненти на Web услугите

Web услугите са базирани на предаването на XML документи по HTTP протокола. Организацията на този обмен на документи се осъществява чрез езика за описание WSDL, протокола SOAP и стандарта за описание, публикуване и откриване на услугите UDDI. По-долу е дадено кратко описание на тези компоненти.

2.1. XML

XML е съкращение от eXtensible Markup Language. XML е разширяем език за създаване на документи, който е независим от платформа, езици за програмиране и операционна система. Тази необвързаност го прави много полезен при нужда от взаимодействие между хетерогенни програмни платформи и/или операционни системи. За обработка на съдържанието на XML документа получателят трябва да притежава информация за неговата структура. Таза информация може да бъде представена като схема на документа (XSD схема или DTD – Document Type Definition), в която е описано значението на използваните в него елементи (тагове) и пространство от имена (XML namespace) с уникално определени имена за елементите.

XML използва структурно маркиране. При тоя тип маркиране маркерите описват структурата на документа, а не неговото форматиране при извеждане. Структурното описание улеснява много електронната обработка на документите, тъй като дава възможност за лесно добавяне, извличане и променяне на елементи от документа. Да разгледаме един пример:

```
<?xml version="1.0" encoding="windows-1251"?>
<група>
<студент>
<име>Иван</име>
<презиме>Денев</презиме>
```

```
<фамилия>Иванов</фамилия>
<фак_номер>046630</фак_номер>
</студент>
<студент>
<име>Десислава</име>
<презиме>Милева</презиме>
<фамилия>Димитрова</фамилия>
<фак_номер>046632</фак_номер>
</студент>
</група>
```

Web услуги- Пример 1 Структурно маркиране

Структурното значение на маркерите </група>, <студент>, <име>, <презиме>, <фамилия> и <фак_номер> се вижда и без обяснения. Очевидни са също и възможностите за електронна обработка на документа. Например за да се изведат данните за всички студенти с фамилия “Иванов” е достатъчно да се проверява само съдържанието на маркера <фамилия> във всеки маркер <студент> и при съвпадение с търсената фамилия да се изведе в подходящ вид съдържанието на родителския маркер <студент>.

2.2. WSDL - Език за описание на Web услугите

Езикът WSDL (Web Service Description Language) е XML схема за описание на интерфейса на Web услуга. Тази схема се записва в един или няколко XML документа. WSDL съдържа описание на следните елементи на Web услугата:

- Дефиниции на типове данни – базират се на XML схеми
- Абстрактни операции
- Методи за обвързване на услуги – асоциират абстрактните съобщения и операции с конкретни транспортни протоколи (по правило с протокола SOAP)

WSDL е проектиран като разширяема структура, която дава възможност за адаптиране към

2.3. SOAP – Протокол за достъп до Web услугите

SOAP (Simple Object Access Protocol) е XML-базиран протокол за достъп до Web услуги. Той дефинира формат за изпращане на

съобщения, който е в основата на обмена на данни между доставчика и клиента на Web услугата. SOAP наследява предимствата на XML:

- независимост от платформата и програмните езици
- простота и разширяемост
- комуникация през защитни стени (Firewall)

SOAP изпраща съобщения чрез HTTP заявка и получава отговор чрез HTTP отговор. HTTP слушателят, който получава SOAP съобщение трябва да притежава програмно средство (SOAP процесор), с което да извлече необходимата му информация от XML документа, който съдържа съобщението.

2.3.1. Структура на SOAP съобщението

SOAP съобщението има следната структура:

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header>
...
</soap:Header>

<soap:Body>
...
  <soap:Fault>
  ...
  </soap:Fault>
</soap:Body>

</soap:Envelope>
```

Web услуги - Пример 2 Структура на SOAP съобщение

Всички елементи на SOAP съобщението са декларирани в подразбиращото се пространство от имена за елемента SOAP envelope

http://www.w3.org/2001/12/soap-envelope

Ще разгледаме накратко елементите, от които е съставено SOAP съобщението.

Елементът Envelope

Елементът Envelope е обвивката на SOAP съобщението. Обвивката е с различно пространство от имена за различните версии на SOAP. Версията на SOAP процесора на клиента на WEB услугата трябва да съответства на версията на пространството от имена на Envelope. Версиите на SOAP се стандартизират от W3C, като препоръките за версиите са публикувани в сайта на на http://www.w3.org/TR/soap/ .

Пространството от имена xmlns:soap

Елементът Envelope в дадения по-горе пример включва атрибута xmlns:soap, който съдържа адреса на пространството от имена за елементите на SOAP съобщението "http://www.w3.org/2001/12/soap-envelope". Това пространство от имена определя елемента Envelope като обвивка на SOAP съобщение. Ако атрибутът xmlns:soap не съдържа адрес на спецификация на пространство от имена за SOAP се генерира грешка и се игнорира съобщението.

Атрибутът encodingStyle

Атрибутът encodingStyle се използва за да дефинира типовете данни, използвани в документа. Този атрибут може да се включи във всеки SOAP елемент, и се прилага към съдържанието на елемента и неговите дъщерни елементи.

Елементът Header

Елементът Header е незадължителен за SOAP. В него се включва специфична за приложението информация (за иидентификация на потребителя, за фирмата-доставчик и др.). Ако елементът Header се съдържа, той трябва да бъде първият дъщерен елемент в елемента

Envelope. В едно SOAP съпбщение може да има няколко елемента Header.

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header>
  <m:Trans xmlns:m="http://www.w3schools.com/transaction/"
  soap:mustUnderstand="1">234
  </m:Trans>
</soap:Header>
.................................................
.................................................
</soap:Envelope>
```

Web услуги - Пример 3 Заглавна част на SOAP съобщение

Атрибутът mustUnderstand

Атрибутът mustUnderstand се използва като индикация дали дадена заглавна част е задължителна или опционална за обработка от получателя. Ако mustUnderstand има стойност "1", то дъщерният елемент на Header трябва да бъде разпознат от SOAP процесора. Ако обработващата програма не разпознае елемента това ще доведе до грешка при обработката на заглавната част Header. Ако mustUnderstand има стойност "0" дъщерният елемент има само информативно значение.

Атрибутът actor

SOAP съобщението може да се придвижва от изпращача към получателя преминавайки през няколко междинни точки. Атрибутът actor дава възможност дадена заглавна част да се насочи към точно определена точка от пътя на съобщението (не обязателно към получателя). В междинни точки могат да се извършват обработки като идентификация на потребителя или други проверки, свързани със сигурността.

Синтаксис:
soap:actor="URI"
където URI е идентификатор (адрес) на получателя

Елементът Body

Елементът Body съдържа данните, които се изпращат чрез SOAP съобщението. Всяко SOAP съобщение трябва задължително да съдържа само един елемент Body. Дъщерните елементи на елемента Body трябва да съдържат атрубут, указващ пространството от имена, което да определя тяхното значение.

Даденият по-долу пример е SOAP съобщение, с което се изпраща заявка за цена на монитори. Елементите m:GetPrice и Item са включени в тялото на съобщението като елементи, специфични за приложението и се определят от пространството от имена "http://www.w3schools.com/ prices" , докато елементите на SOAP се определят чрез пространството от имена http://www.w3.org/2001/12/soap-envelope.

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body>
  <m:GetPrice xmlns:m=" http://www.monitors.com/prices">
    <m:Item>Monitors</m:Item>
  </m:GetPrice>
</soap:Body>

</soap:Envelope>
```

Web услуги - Пример 4 SOAP съобщение с приложно-специфични елементи в секция Body

Отговорът на SOAP съобщението може да има вида:

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
```

```
<soap:Body>
  <m:GetPriceResponse xmlns:m="http://www.monitors.com/prices">
    <m:Price>190</m:Price>
  </m:GetPriceResponse>
</soap:Body>
</soap:Envelope>
```

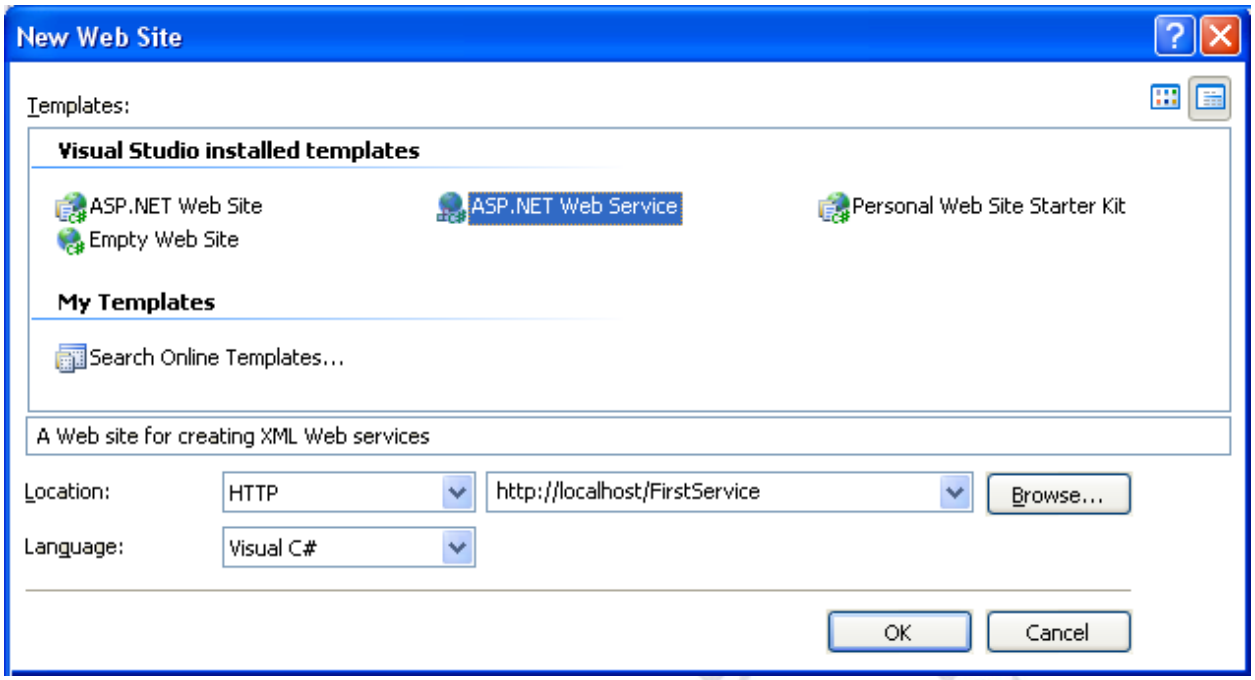
Елементът Fault

Незадължителният елемент Fault се използва за изпращане на съобщение за грешка. Всяко SOAP съобщение-отговор може да съдържа само един блок за грешка. Информацията за грешката се представя от дъщерните елементи на елемента Fault, които имат следното значение:

<faultcode	Код на грешката
<faultstring	Текст с информация за грешката
<faultactor	Индикатор (адрес) на SOAP процесора, генерирал грешката
<detail	Свързана с приложението информация за грешката, получена в случай че съобщението не е обработено успешно.

3. Създаване на Web услуга с Visual Studio .NET

За да създадем Web услуга с Visual Studio .NET избираме от менюто File→New→Web Site и в диалоговата кутия New Web Site избираме: Вид проект: ASP.NET Web Service, Location: HTTP Language: Visual C# и в текстовата кутия въвеждаме адрес http://localhost/ FirstService.



Фиг. 7 Създаване на проект за Web услуга

Visual Studio .NET създава файловете на проекта: Service.asmx, който съдържа само директивата

```
<%@ WebService Language="C#" CodeBehind="~/App_Code/Service.cs"
Class=" FirstService" %>
```

Web услуги - Пример 5 Директива WebService на . asmx страница в проект за Web услуга

и кодовата страница Service.cs, която съдържа декларацията на класа Service с конструктор без параметри Service() и метод HelloWorld().

```
using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
```

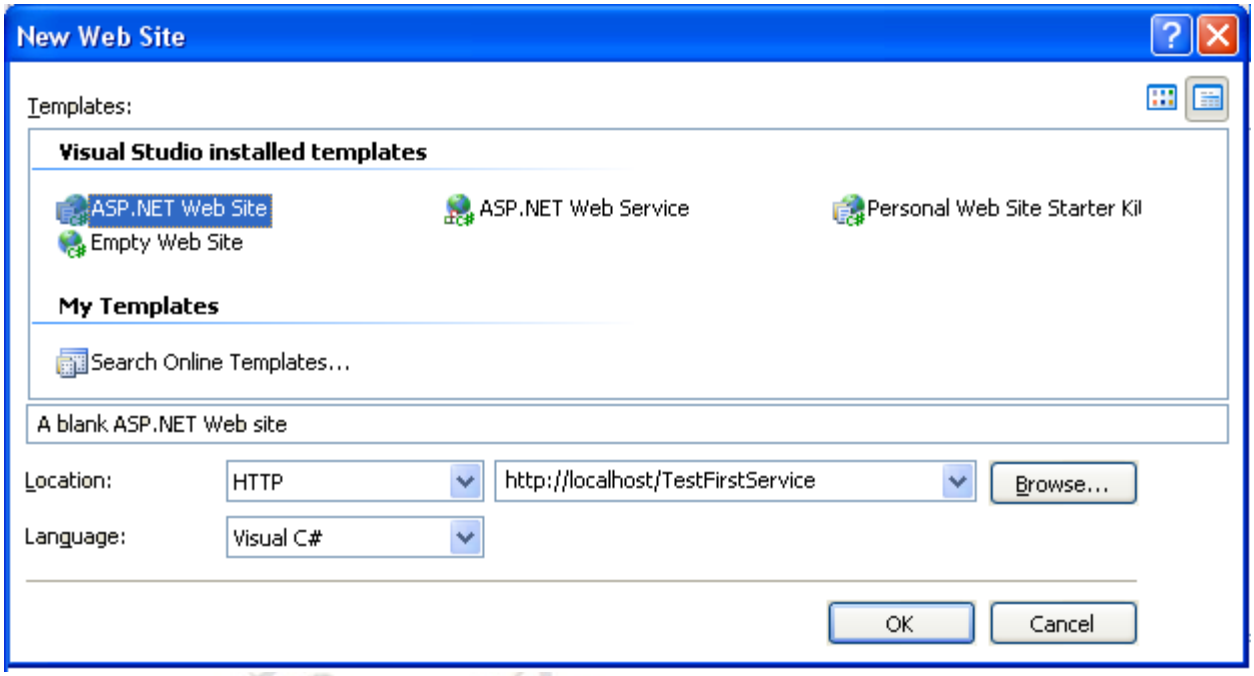
```
[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class FirstService: System.Web.Services.WebService
{
   public FirstService() {

      //Uncomment the following line if using designed components
      //InitializeComponent();
   }

   [WebMethod]
   public string HelloWorld() {
      return "Hello World";
   }
```

Web услуги - Пример 6 Кодова страница на Web услуга, генерирана от Visual Studio .NET

Преименуваме класа и конструктора му на FirstService. Променяме също стойността на атрибута Class в директивата WebService на новото име на класа FirstService. От менюто Build→ Build Solution. С това Web услугата е създадена.

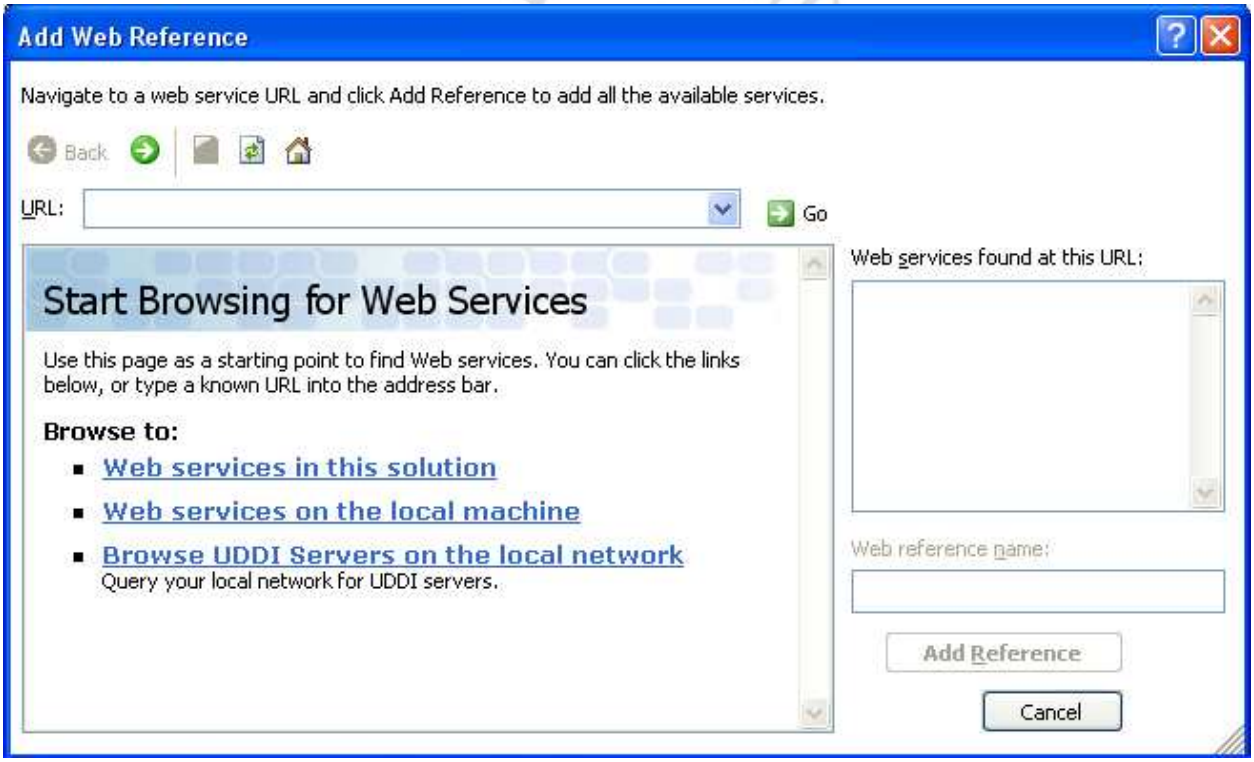
4. Създаване на Web сайт, използващ Web услуга с Visual Studio .NET

За да създадем Web сайт, който използва Web услуга с Visual Studio .NET избираме от менюто File→New→Web Site и в диалоговата кутия New Web Site избираме: Вид проект: ASP.NET Web Site, Location: HTTP Language: Visual C# и в текстовата кутия въвеждаме адрес http://localhost/TestFirstService.



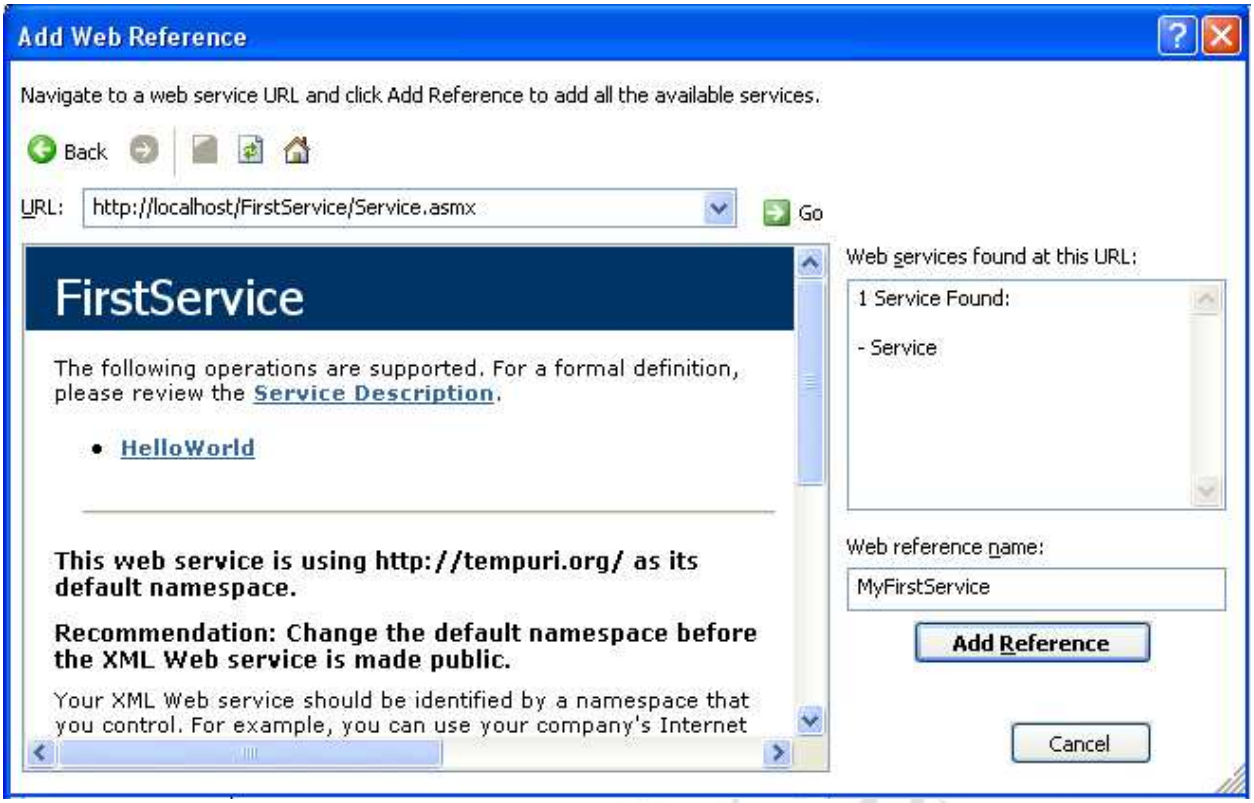
Фиг. 8 Създаване на Web сайт за тестване на Web услугата FirstService

За да получим достъп до Web услугата е необходимо да включим референция към нея чрез функцията Web Site→Add Reference.



Фиг. 9 Добавяне на референция за Web услуга към Web сайт

От диалоговата кутия избираме “Web Services on local machine” , и от списъка на услугите – услугата Service с адрес “http://localhost/ FirstService/Service.asmx”. В текстовата кутия “Web Reference Name” заменяме името на референцията от “localhost” на “MyFirstService”.



Фиг. 10 Променяне на името на референцията в диалоговата кутия Add Reference Name

За да извикаме метода на Web услугата да добавим към .aspx страницата на проекта един бутон и с двукратно щракване с мишката върху него да добавим за него към кодовата страница събитийната процедура Button1_Click.

Достъпа до Web услугата, след като е включена референция към нея, се извършва както достъп до потребителска библиотека. Използваме името на референцията към Web услугата като пространство от имена, създаваме обект от класа, деклариран във Web услугата и чрез него осъществяваме достъп до атрибутите и методите, които предоставя Web услугата. В случая тя съдържа само метода HelloWorld и можем да изведем с .aspx страницата на сайта символния низ, който той връща. За да не се налага всеки път преди името на класа да пишем пространството от имена MyFirstService е по-добре да го включим в кодовата страница с директива using.

```
using MyFirstService; //Пространство от имена на услугата

public partial class _Default : System.Web.UI.Page
{
   protected void Button1_Click(object sender, EventArgs e)
   { //Създаване на обект от класа на услугата
      FirstService oService = new FirstService();
      Response.Write("Call of Service's method " + oService.HelloWorld());
   }
}
```

Web услуги - Пример 7 Извикване на метод на Web услуга в ASP.NET приложение

Даденият по-горе пример съдържа кода на събитийна процедура Button1_Click на бутон от ASP.NET приложение, в която се извиква метод на Web услугата FirstService. Създава се обект oService от клас FirstService и чрез него се извиква метода HelloWorld на услугата. Символният низ, който връща метода се изпраща към прозореца на клиентския браузър посредством метода Write на обект Response.

```
   public FirstService () {
      string fspec = Server.MapPath("/") + "Request_data.txt";
      this.Context.Request.SaveAs(fspec, true); //Запис във файл

      //Uncomment the following line if using designed components
      //InitializeComponent();
   }
```

Web услуги - Пример 8 Запис във файл на получената от Web услугата заявка

5. Обмен на данни между Web услугата и клиента

Клиент на Web услуга може да бъде всяко приложение, което може да заявява и получава данни от Web сървър. Във Visual Studio.Net, както беше показано в предния пример, за да се свърже клиентското приложение към Web услуга, към него се добавя референция (Web

Reference) за услугата и пространство от имена за тази референция. Извикването на методи на услугата става чрез обект, създаден от класа на Web услугата по същия синтаксис, по който се създават обекти от класове, декларирани в приложението.

Зад сцената всъщност извикването на метод става чрез извикване на т.н. прокси обект, който извиква Web услугата. Базовият прокси клас предоставя необходимата функционалност за изпращане на SOAP заявка чрез която се извиква за изпълнение метода на Web услугата. Прокси обектът чете WSDL файла за да провери коректността на сигнатурата на метода и типа на данните, създава SOAP заявка и я изпраща на Web услугата за обработка. Прокси обектът добавя и необходимите заглавни части и изпраща HTTP заявка към Web сървъра, на който е публикувана Web услугата.

В дадения по-горе пример заявката, която изпраща ASP.NET приложението има следното съдържание:

```
POST /FirstService/Service.asmx HTTP/1.1
Connection: Keep-Alive
Content-Length: 288
Content-Type: text/xml; charset=utf-8
Expect: 100-continue
Host: localhost
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; MS Web Services Client
Protocol 2.0.50727.3053)
SOAPAction: "http://tempuri.org/HelloWorld"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<soap:Body>
<HelloWorld xmlns="http://tempuri.org/" />
</soap:Body>
</soap:Envelope>
```

Web услуги - Пример 9 SOAP заявка за изпълнение на метода HelloWorld от Web услугата FirstService

Както се вижда от примера, заявката включва заглавна част и SOAP съобщение, с което се заявява изпълнението на метода HelloWorld. Като

резултат от изпълнението на заявката се връща отговор, който също съдържа заглавна част и SOAP съобщение от вида:

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 85

<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
  <soap12:Body>
    <HelloWorldResponse xmlns="http://tempuri.org/">
      <HelloWorldResult>Hello World </HelloWorldResult>
    </HelloWorldResponse>
  </soap12:Body>
</soap12:Envelope>
```

Web услуги. Пример 10 SOAP отговор от изпълнение на метода HelloWorld от Web услугата FirstService.

Така, с този прост механизъм на обмен на XML документи Web услугите дават възможност за извикване на методи и получаване на резултати от тяхното изпълнение. Ако методът изисква входни параметри, тогава в SOAP заявката те се появяват като дъщерни елементи на елемента, с който се извиква метода.

Указател на примерите

Използвана литература

1. Microsoft. Introduction to Microsoft® ASP.NET, Trainer Pack, Material No: 2063CTP
2. О. Железов, Програмиране в мрежова среда, печ. база на ТУ-Варна, 2004г.
3. Джон Крейг, Джеф Уеб , Програмиране с VB.NET С. "СОФТПРЕС"., 2003г..
4. П. Харис , Програмиране с C# С. "Нисофт"., 2004г.
5. Ерик Нюкъмър, Web услуги XML, WSDL, SOAP, UDDL, , СофтПрес, 2004
6. Кит Франклин, VB.NET для разработчиков, М. "Вилямс", 2002

Информация в интернет сайтове:
http://www.15seconds.com/Issue/020220.htm Using Forms Authentication in ASP.NET By Jeff Gonzalez
Online ASP.NET учебни материали:
http://www.dotnet-guide.com. Справочник за ASP.NET
http://www.codeproject.com/ Сайт с учебни материали за Microsoft технологии
http://www.w3schools.com/ASPNET/default.asp Учебни материали за ASP.NET
http://www.w3schools.com/SOAP/default.asp Учебни материали за Web услуги

Web сайт с учебни материали, поддържан от автора:
http://rusalka-bg.com/dt/