

ASP.NET 教程

ASP.NET 是一个使用 HTML、CSS、JavaScript 和服务器脚本创建网页和网站的开发框架。

ASP.NET 支持三种不同的开发模式：

Web Pages（Web 页面）、MVC（Model View Controller 模型-视图-控制器）、Web Forms（Web 窗体）：

Web Pages 单页面模式	MVC 模型-视图-控制器	Web Forms 事件驱动模式
<div>最简单的 ASP.NET 模式。</div> <div>与 PHP 和经典 ASP 相似。</div> <div>内置了数据库、视频、图形、社交媒体等模板和帮助器。</div>	<div>MVC 将 Web 应用程序分成 3 个不同的组成部分：</div> <div>模型负责数据</div> <div>视图负责显示</div> <div>控制器负责输入</div>	<div>传统的 ASP.NET 事件驱动开发模式：</div> <div>带有服务器控件、服务器事件和服务器代码的网页。</div>

Web Pages 教程

如果您刚接触 **ASP.NET**，建议从 **Web Pages** 开始学习。

Web Pages 是开发 ASP.NET 网站最简单的开发模式。

在我们的 **Web Pages** 教程中，您将学习如何使用 VB (Visual Basic) 或者 C# (C sharp) 最新的 **Razor** 服务器标记语法将 HTML、CSS、JavaScript 和服务器代码结合起来。

您也可以学习如何使用具有可编程的 **Web Helpers**（包括数据库、视频、图形、社交媒体等等）来扩展您的网页。

[现在开始学习 ASP.NET Web Pages！](#)

MVC 教程

MVC 是一种使用 MVC（Model View Controller 模型-视图-控制器）设计创建 Web 应用程序的模式。

如果您想要一个替代传统的 **ASP.NET** 的轻量级的开发模式，可以从 **MVC** 开始学习。

在我们的 MVC 教程中，您将学到如何使用集成了现有的所有 ASP.NET 特性（比如 Master Pages、Security、Authentication 母版页、安全、验证）的轻量级的开发模式创建 Web 应用程序。

[现在开始学习 ASP.NET MVC！](#)

Web Forms 教程

Web Forms 是传统的基于事件驱动的 ASP.NET 模式。

多年来，开发者已经使用 ASP.NET Web Forms 创建了许多众所周知的大型网站。

如果您想学习在过去的 10 年中许多 Web 开发人员使用的设计模式，那么您可以从 **Web Forms** 开始学习。

[现在开始学习 ASP.NET Web Forms！](#)

谁适合阅读本教程？

本教程适合于任何想要学习在微软 ASP.NET 平台上创建网站的人员，从业余站点到最新的、现代化的、完全商业化的网络。

即使您是刚接触 Web 编程，您也可以学习本教程，如果对 HTML 和 CSS 有基本的了解将会有助于本教程的学习。

如果您对脚本语言如 JavaScript 或者 VB (Visual Basic) 有基本的了解，那将会对学习本教程很有帮助。

您是否偏爱 VB 胜过 C# (C sharp)？您是否想学习这两种语言？有个好消息：菜鸟教程提供的大多数代码实例都有这两种语言的版本。

如果您是一名有过 ASP.NET 开发经验的专业的 Web 开发人员，您仍然可以从本教程中学到很多东西，因为这些教程介绍了很多新的 ASP.NET 的概念，比如 HTML5、CSS3、jQuery 等等。

ASP.NET

经典 ASP - Active Server Pages（动态服务器页面）

ASP，全称 Active Server Pages（动态服务器页面），也被称为经典 ASP，是在1998年作为微软的第一个服务器端脚本引擎推出的。

ASP 是一种使得网页中的脚本在因特网服务器上被执行的技术。

ASP 页面的文件扩展名是 .asp，通常是用 VBScript 编写的。

如果您想学习经典 ASP，请访问我们的 [经典 ASP 教程](#)。

ASP.NET

ASP.NET 是新一代 ASP。它与经典 ASP 是不兼容的，但 ASP.NET 可能包括经典 ASP。

ASP.NET 页面是经过编译的，这使得它们的运行速度比经典 ASP 快。

ASP.NET 具有更好的语言支持，有一大套的用户控件和基于 XML 的组件，并集成了用户身份验证。

ASP.NET 页面的扩展名是 .aspx，通常是用 VB (Visual Basic) 或者 C# (C sharp) 编写。

在 ASP.NET 中的控件可以用不同的语言（包括 C++ 和 Java）编写。

当浏览器请求 ASP.NET 文件时，ASP.NET 引擎读取文件，编译和执行脚本文件，并将结果以普通的 HTML 页面返回给浏览器。

ASP.NET Razor

Razor 是一种将服务器代码嵌入到 ASP.NET 网页中的新的、简单的标记语法，很像经典 ASP。

Razor 具有传统的 ASP.NET 的功能，但更容易使用并且更容易学习。

ASP.NET 编程语言

本教程介绍了以下编程语言：

Visual Basic (VB.NET)

C# (发音：C sharp)

ASP.NET 服务器技术

本教程介绍了以下服务器技术

Web Pages（Razor 语法）

MVC（模型-视图-控制器）

Web Forms（传统的 ASP.NET）

ASP.NET 开发工具

ASP.NET 支持以下开发工具：

- WebMatrix
- Visual Web Developer
- Visual Studio

在本教程中，Web Pages 教程使用了 WebMatrix ， MVC 教程和 Web Forms 教程使用了 Visual Web Developer。

ASP.NET 文件扩展名

- 经典 ASP 文件的文件扩展名为 .asp
- ASP.NET 文件的文件扩展名为 .aspx
- Razor C# 语法的 ASP.NET 文件的文件扩展名为 .cshtml
- Razor VB 语法的 ASP.NET 文件的文件扩展名为 .vbhtml

[点我分享笔记](#)

反馈/建议



ASP.NET Web Pages - 教程

ASP.NET 是一个使用 HTML、CSS、JavaScript 和服务器脚本创建网页和网站的开发框架。

ASP.NET 支持三种不同的开发模式：

Web Pages（Web 页面）、MVC（Model View Controller 模型-视图-控制器）、Web Forms（Web 窗体）：

本教程介绍 **Web Pages**。

- Web Pages
- MVC
- Web Forms

从何入手？

多数开发人员学习一个新技术，是从查看运行实例开始的。

通过"运行实例"轻松学习

我们的"运行实例"工具让 Web Pages 变得更简单易学。

它在运行实例的同时显示 ASP.NET 代码和 HTML 输出。

点击"运行实例"按钮来看看它是如何工作的：

Web Pages 实例

<html>

```
<body>
<h1>Hello Web Pages</h1>
<p>The time is @DateTime.Now</p>
</body>
</html>
```

运行实例 »

什么是 Web Pages?

Web Pages 是三种创建 ASP.NET 网站和 Web 应用程序的编程模式中的一种。

其他两种编程模式是 Web Forms 和 MVC (Model View Controller 模型-视图-控制器)。

Web Pages 是开发 ASP.NET 网页最简单的开发模式。它提供了一种简单的方式来将 HTML、CSS、JavaScript 和服务器脚本结合起来:

- 容易学习, 容易理解, 容易使用

- 围绕着单一的网页创建

- 与 PHP 和经典 ASP 相似

- Visual Basic 或者 C# 的服务器脚本

- 全 HTML、CSS 和 JavaScript 控制

Web Pages 内置了数据库、视频、图形、社交媒体和其他更多的 Web Helpers, 因此很容易扩展。

Web Pages 教程

如果您刚接触 ASP.NET, 建议从 Web Pages 开始学习。

在我们的 Web Pages 教程中, 您将学习到如何使用 VB(Visual Basic) 或者 C#(C sharp) 最新的 Razor 服务器标记语法将 HTML、CSS、JavaScript 和服务器代码结合起来。

您也可以学习如何使用具有可编程的 Web Helpers (包括数据库、视频、图形、社交媒体等等) 来扩展您的网页。

Web Pages 实例

通过实例学习!

由于 ASP.NET 代码是在服务器上执行的, 您不能在您的浏览器中查看代码。您只能看到普通的 HTML 页面输出。

在菜鸟教程中, 每个实例都会把隐藏的 ASP.NET 代码显示出来, 这将让您更容易地理解它是如何工作的。

[Web Pages 实例](#)

Web Pages 参考手册

在本教程的最后, 您将看到一套完整的 ASP.NET 参考手册, 介绍了对象、组件、属性和方法。

[Web Pages 参考手册](#)

使用 WebMatrix

在本教程中, 我们使用了 WebMatrix 。

WebMatrix 是一个简单但功能强大的, 由微软专门为 Web Pages 量身定做的, 免费的 ASP.NET 开发工具。

WebMatrix 包含:

- Web Pages 实例和模板

- 一种 Web 服务器语言 (VB 或者 C# 的 Razor 服务器标记语法)

- 一种 Web 服务器 (IIS Express)

- 一种数据库服务器 (SQL Server Compact)

- 一个完整的 Web 开发框架 (ASP.NET)

通过使用 WebMatrix, 您可以从一个空的网站和一个空白页面开始开发, 或者您也可以使用"Web 应用程序库"中的开源应用程序进行二次开发。PHP 和 ASP.NET 应用程序很多都是开源的, 比如 Umbraco、DotNetNuke、Drupal、Joomla、WordPress 等等。WebMatrix 也有内置安全性、搜索引擎优化和网络出版工具。

使用 WebMatrix 开发的技术和代码可以无缝地转化为完全专业化的 ASP.NET 应用程序。

如果您想尝试使用 WebMatrix ， 请点击下面的链接进行安装：
<http://www.microsoft.com/web/gallery/install.aspx?appid=WebMatrix>

☐ ASP.NET

ASP.NET Web Pages Razor ☐

☐ 点我分享笔记

反馈/建议

Copyright © 2013-2018 菜鸟教程 runoob.com All Rights Reserved. 备案号： 闽ICP备15012807号-1



首页 HTML CSS JS 本地书签

☐ ASP.NET Web Pages 教程

ASP.NET Web Pages 布局 ☐

ASP.NET Web Pages - 添加 Razor 代码

在本教程中，我们将使用 C# 和 Visual Basic 代码的 Razor 标记。

什么是 Razor ？

- Razor 是一种将基于服务器的代码添加到网页中的标记语法
- Razor 具有传统 ASP.NET 标记的功能，但更容易使用并且更容易学习
- Razor 是一种服务器端标记语法，与 ASP 和 PHP 很像
- Razor 支持 C# 和 Visual Basic 编程语言

添加 Razor 代码

请记住上一章实例中的网页：

```
<!DOCTYPE html>

<html lang="en">
<head>
<meta charset="utf-8" />
<title>Web Pages Demo</title>
</head>
<body>
<h1>Hello Web Pages</h1>
</body>
</html>
```

现在向实例中添加一些 Razor 代码：

实例

```
<!DOCTYPE html>

<html lang="en">
<head>
<meta charset="utf-8" />
<title>Web Pages Demo</title>
</head>
<body>
<h1>Hello Web Pages</h1>
```

```
<p>The time is @DateTime.Now</p>
</body>
</html>
```

运行实例 »

该页面中包含普通的 HTML 标记，除此之外，还添加了一个 @ 标识的 Razor 代码。

Razor 代码能够在服务器上实时地完成多有的动作，并将结果显示出来。（您可以指定格式化选项，否则只会显示默认项。）

主要的 Razor C# 语法规则

Razor 代码块包含在 @{ ... } 中

内联表达式（变量和函数）以 @ 开头

代码语句用分号结束

变量使用 var 关键字声明

字符串用引号括起来

C# 代码区分大小写

C# 文件的扩展名是 .cshtml

C# 实例

```
<!-- Single statement block -->
@{ var myMessage = "Hello World"; }

<!-- Inline expression or variable -->
<p>The value of myMessage is: @myMessage</p>

<!-- Multi-statement block -->
@{
var greeting = "Welcome to our site!";
var weekDay = DateTime.Now.DayOfWeek;
var greetingMessage = greeting + " Today is: " + weekDay;
}
<p>The greeting is: @greetingMessage</p>
```

运行实例 »

主要的 Razor VB 语法规则

Razor 代码块包含在 @Code ... End Code 中

内联表达式（变量和函数）以 @ 开头

变量使用 Dim 关键字声明

字符串用引号括起来

VB 代码不区分大小写

VB 文件的扩展名是 .vbhtml

实例

```
<!-- Single statement block -->
@Code dim myMessage = "Hello World" End Code

<!-- Inline expression or variable -->
<p>The value of myMessage is: @myMessage</p>

<!-- Multi-statement block -->
@Code
dim greeting = "Welcome to our site!"
dim weekDay = DateTime.Now.DayOfWeek
```

```
dim greetingMessage = greeting & " Today is: " & weekDay
End Code
```

```
<p>The greeting is: @greetingMessage</p>
```

[运行实例 »](#)

更多关于 C# 和 Visual Basic

如果您想学习更多关于 Razor、C#、Visual Basic 编程语言，请查看本教程的 [Razor 部分](#)。

[ASP.NET Web Pages 教程](#)[ASP.NET Web Pages 布局](#)[点我分享笔记](#)[反馈/建议](#)

Copyright © 2013-2018 菜鸟教程 [runoob.com](#) All Rights Reserved. 备案号: 闽ICP备15012807号-1

[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)[ASP.NET Web Pages Razor](#)[ASP.NET Web Pages 文件夹](#)

ASP.NET Web Pages - 页面布局

通过 Web Pages，创建一个布局一致的网站是很容易的事。

一致的外观

在因特网上，您会发现很多网站都具有一致的外观和风格：

每个页面有相同的头部

每个页面有相同的底部

每个页面有相同的样式和布局

通过 Web Pages，您能非常高效地做到这点。您可以把重复使用的内容块（比如页面头部和底部）写在一个单独的文件中。

您还可以使用布局模板（布局文件）为站点的所有网页定义一致的布局。

Content Blocks（内容块）

许多网站都有一些内容是被显示在站点的每个页面中（比如页面头部和底部）。

通过 Web Pages，您可以使用 `@RenderPage()` 方法从不同的文件导入内容。

内容块（来自另一个文件）能被导入网页中的任何地方。内容块可以包含文本，标记和代码，就像任何普通的网页一样。

将共同的头部和底部写成单独的文件，这样会帮您节省大量的工作。您不必在每个页面中书写相同的内容，当内容有变动时，您只要修改头部或者底部文件，就可以看到站点中的每个页面的相应内容都已更新。

以下显示了它在代码中是如何呈现的：

实例

```
<html>
```

```
<body>
@RenderPage("header.cshtml")
<h1>Hello Web Pages</h1>
<p>This is a paragraph</p>
@RenderPage("footer.cshtml")
</body>
</html>
```

运行实例 »

Layout Page（布局页）

在上一部分，您看到了，想在多个网页中显示相同内容是非常容易的。

另一种创建一致外观的方法是使用布局页。一个布局页包含了网页的结构，而不是内容。当一个网页（内容页）链接到布局页，它会根据布局页（模板）的结构进行显示。

布局页中使用 **@RenderBody()** 方法嵌入内容页，除此之外，它与一个正常的网页没有什么差别。

每个内容页都必须以**布局指令**开始。

以下显示了它在代码中是如何呈现的：

布局页：

```
<html>
<body>
<p>This is header text</p>
@RenderBody()
<p>&copy; 2012 Runoob. All rights reserved.</p>
</body>
</html>
```

任何网页：

```
@{Layout="Layout.cshtml";}

<h1>Welcome to Runoob.com</h1>

<p>
Lorem ipsum dolor sit amet, consectetur adipisicing elit,sed do eiusmod tempor incididunt ut labore et dolore magn
a aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laborisnisi ut aliquip ex ea commodo consequa
t.
</p>
```

运行实例 »

D.R.Y. - Don't Repeat Yourself（不要自我重复）

通过 **Content Blocks**（内容块）和 **Layout Pages**（布局页）这两个 **ASP.NET** 工具，您可以让您的 **Web** 应用程序显示一致的外观。

这两个工具能帮您节省大量的工作，您不必再每个页面上重复相同的信息。集中的标记、样式和代码让您的 **Web** 应用程序更易于管理，更易于维护。

防止文件被浏览

在 **ASP.NET** 中，文件的名称以下划线开头，可以防止这些文件在网上被浏览。

如果您不想让您的内容块或者布局页被您的用户看到，可以重命名这些文件：

```
_header.cshtml
_footer.cshtml
_Layout.cshtml
```

隐藏敏感信息

在 **ASP.NET** 中，隐藏敏感信息（数据库密码、电子邮件密码等等）最通用的方法是将这些信息保存在一个名为"**_AppStart**"的单独的文件中。

_AppStart.cshtml


```
@{
WebMail.SmtpServer = "mailserver.example.com";
WebMail.EnableSsl = true;
WebMail.UserName = "username@example.com";
WebMail.Password = "your-password";
WebMail.From = "your-name-here@example.com";
}
```

[ASP.NET Web Pages Razor](#)

[ASP.NET Web Pages 文件夹](#)

[点我分享笔记](#)

[反馈/建议](#)

Copyright © 2013-2018 菜鸟教程 [runoob.com](#) All Rights Reserved. 备案号: 闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

[ASP.NET Web Pages 布局](#)

[ASP.NET Web Pages 全局文件](#)

ASP.NET Web Pages - 文件夹

本章介绍有关文件夹和文件夹路径的知识。

在本章中，您将学到：

逻辑文件夹结构和物理文件夹结构

虚拟名称和物理名称

Web URL 和 Web 路径

逻辑文件夹结构

下面是典型的 ASP.NET 网站文件夹结构：

Folders

"Account" 文件夹包含登录和安全文件

"App_Data" 文件夹包含数据库和数据文件

"Images" 文件夹包含图片

"Scripts" 文件夹包含浏览器脚本

"Shared" 文件夹包含公共的文件（比如布局和样式文件）

物理文件夹结构

在上述网站中的"Images"文件夹在计算机上的物理文件夹结构可能如下：

C:\Documents\MyWebSites\Demo\Images

虚拟名称和物理名称

以上面的例子为例：

网站图片的虚拟名称可能是"Images/pic31.jpg"。

对应的物理名称是"C:\Documents\MyWebSites\Demo\Images\pic31.jpg"。

URL 和路径

URL 是用来访问网站中的文件：<http://www.runoob.com/html/html-tutorial.html>

URL 对应于服务器上的物理文件：C:\MyWebSites\runoob\html\html-tutorial.html

虚拟路径是物理路径的一种简写表示。如果您使用虚拟路径，当您更改域名或者将您的网页移到其他服务器上时，您可以不用更新路径。

URL	http://www.runoob.com/html/html-tutorial.html
服务器名称	RUNOOB
虚拟路径	/html/html-tutorial.html
物理路径	C:\MyWebSites\runoob\html\html-tutorial.html

磁盘驱动器的根目录如下书写 C:，但是网站的根目录是 /（斜线）。

Web 文件夹的虚拟路径通常是与物理文件夹不相同。

在您的代码中，根据您的编码需要决定使用物理路径和和虚拟路径。

ASP.NET 文件夹路径有 3 种工具：~ 运算符、Server.MapPath 方法 和 Href 方法。

~ 运算符

使用 ~ 运算符，在编程代码中规定虚拟路径。

如果您使用 ~ 运算符，在您的站点迁移到其他不同的文件夹或者位置时，您可以不用更改您的任何代码：

```
var myImagesFolder = "~/images";
var myStyleSheet = "~/styles/StyleSheet.css";
```

Server.MapPath 方法

Server.MapPath 方法将虚拟路径 (/index.html) 转换成服务器能理解的物理路径 (C:\Documents\MyWebSites\Demo\default.html)。

当您需要打开服务器上的数据文件时，您可以使用这个方法（只有提供完整的物理路径才能访问数据文件）：

```
var pathName = "~/dataFile.txt";
var fileName = Server.MapPath(pathName);
```

在本教程的下一章中，您会学到更多关于读取（和写入）服务器上的数据文件的知识。

Href 方法

Href 方法将代码中的使用的路径转换成浏览器可以理解的路径（浏览器无法理解 ~ 运算符）。

您可以使用 Href 方法创建资源（比如图像文件 和 CSS 文件）的路径。

一般会在 HTML 中的 <a>、 和 <link> 元素中使用此方法：

```
@{var myStyleSheet = "~/Shared/Site.css";}

<!-- This creates a link to the CSS file. -->
<link rel="stylesheet" type="text/css" href="@Href(myStyleSheet)" />

<!-- Same as : -->
<link rel="stylesheet" type="text/css" href="/Shared/Site.css" />
```

Href 方法是 WebPage 对象的一种方法。

□ ASP.NET Web Pages 布局

ASP.NET Web Pages 全局文件 □

□ 点我分享笔记

反馈/建议



ASP.NET Web Pages - 全局页面

本章介绍全局页面 `AppStart` 和 `PageStart`。

在 Web 启动之前：_AppStart

大多数的服务器端代码是写在个人网页里边。例如，如果网页中包含输入表单，那么这个网页通常包含用来读取表单数据的服务器端代码。

然而，您可以通过在您的站点根目录下创建一个名为 `_AppStart` 的页面，这样在站点启动之前可以先启动代码执行。如果存在此页面，ASP.NET 会在站点中其它页面被请求时，优先运行这个页面。

`_AppStart` 的典型用途是启动代码和初始化全局数值（比如计数器和全局名称）。

注释 1: `_AppStart` 的文件扩展名与您的网页一致，比如：`_AppStart.cshtml`。

注释 2: `_AppStart` 有下划线前缀。因此，这些文件不可以直接浏览。

在每一个页面之前：_PageStart

就像 `_AppStart` 在您的站点启动之前就运行一样，您可以编写在每个文件夹中的任何页面之前运行的代码。

对于您网站中的每个文件夹，您可以添加一个名为 `_PageStart` 的文件。

`_PageStart` 的典型用途是为一个文件夹中的所有页面设置布局页面，或者在运行某个页面之前检查用户是否已经登录。

它是如何工作的？

下图显示了它是如何工作的：

`PageStart`

当接收到一个请求时，ASP.NET 会首先检查 `_AppStart` 是否存在。如果 `_AppStart` 存在且这是站点接收到的第一个请求，则运行 `_AppStart`。

然后 ASP.NET 检查 `_PageStart` 是否存在。如果 `_PageStart` 存在，则在其它被请求的页面运行之前先运行 `_PageStart`。

您可以在 `_PageStart` 中调用 `RunPage()` 来指定被请求页面的运行位置。否则，默认情况下，被请求页面是在 `_PageStart` 运行之后才被运行。



ASP.NET Web Pages - HTML 表单

表单是 HTML 文档中放置输入控件（文本框、复选框、单选按钮、下拉列表）的部分。

创建一个 HTML 输入页面

Razor 实例

```
<html>
<body>
@{
if (IsPost) {
string companyname = Request["companyname"];
string contactname = Request["contactname"];
<p>You entered: <br />
Company Name: @companyname <br />
Contact Name: @contactname </p>
}
else
{
<form method="post" action="">
Company Name:<br />
<input type="text" name="CompanyName" value="" /><br />
Contact Name:<br />
<input type="text" name="ContactName" value="" /><br /><br />
<input type="submit" value="Submit" class="submit" />
</form>
}
}
</body>
</html>
```

运行实例 »

Razor 实例 - 显示图像

假设在您的图像文件夹中有 3 张图像，您想根据用户的选择动态地显示图像。

这可以通过一段简单的 Razor 代码来实现。

如果在您的网站的图像文件夹中有一个名为 "Photo1.jpg" 的图像，您可以使用 HTML 的 元素来显示图像，如下所示：

```

```

下面的例子演示了如何显示用户从下列列表中选择图像：

Razor 实例

```
@{
var imagePath="";
if (Request["Choice"] != null)
{imagePath="images/" + Request["Choice"];}
}
<!DOCTYPE html>
<html>
<body>
<h1>Display Images</h1>
<form method="post" action="">
I want to see:
<select name="Choice">
<option value="Photo1.jpg">Photo 1</option>
<option value="Photo2.jpg">Photo 2</option>
<option value="Photo3.jpg">Photo 3</option>
</select>
<input type="submit" value="Submit" />
@if (imagePath != "")
{
<p>

</p>
}
</form>
</body>
```

</html>

运行实例 »

实例解释

服务器创建了一个叫 **imagePath** 的变量。

HTML 页面有一个名为 **Choice** 的下拉列表（<select> 元素）。它允许用户根据自己的意愿选择一个名称（如 **Photo 1**），当页面被提交到 Web 服务器时，则传递了一个文件名（如 **Photo1.jpg**）。

Razor 代码通过 **Request["Choice"]** 读取 Choice 的值。如果通过代码构建的图像路径（images/Photo1.jpg）有效，就把图像路径赋值给变量 **imagePath**。

在 HTML 页面中， 元素用来显示图像。当页面显示时，src 属性用来设置 imagePath 变量的值。

 元素是在一个 if 块中，这是为了防止显示没有名称的图像，比如页面第一次被加载显示的时候。

☐ ASP.NET Web Pages 全局文件

ASP.NET Web Pages 对象 ☐

☐ 点我分享笔记

反馈/建议

Copyright © 2013-2018 菜鸟教程 runoob.com All Rights Reserved. 备案号：闽ICP备15012807号-1

☐

首页 HTML CSS JS 本地书签

☐ ASP.NET Web Pages HTML 表单

ASP.NET Web Pages 文件 ☐

ASP.NET Web Pages - 对象

Web Pages 经常是跟对象有关的。

Page 对象

您已经看到了一些在使用的 Page 对象方法：

```
@RenderPage("header.cshtml")

@RenderBody()
```

在前面的章节中，您已经看到了两个 Page 对象属性（isPost 和 Request）：

```
If (isPost) {

    if (Request["Choice"] != null ) {
```

某些 Page 对象方法

方法	描述
href	使用指定的值创建 URL。
RenderBody()	呈现不在布局页命名区域的内容页的一部分。
RenderPage(page)	在另一个页面中呈现某一个页面的内容。

RenderSection(<i>section</i>)	呈现布局页命名区域的内容。
Write(<i>object</i>)	将对象作为 HTML 编码字符串写入。
WriteLiteral	写入对象时优先不使用 HTML 编码。

某些 Page 对象属性

属性	描述
isPost	如果客户端使用的 HTTP 数据传输方法是 POST 请求，则返回 true。
Layout	获取或者设置布局页面的路径。
Page	提供了对页面和布局页之间共享的数据的类似属性访问。
Request	为当前的 HTTP 请求获取 HttpRequest 对象。
Server	获取 HttpServerUtility 对象，该对象提供了网页处理方法。

Page 对象的 Page 属性

Page 对象的 Page 属性，提供了对页面和布局页之间共享的数据的类似属性访问。

您可以对 Page 属性使用（添加）您自己的属性：

- Page.Title
- Page.Version
- Page.anythingyoulike

页面属性是非常有用的。例如，在内容文件中设置页面标题，并在布局文件中使用：

Home.cshtml

```
@{
Layout="~/Shared/Layout.cshtml";
Page.Title="Home Page"
}

<h1>Welcome to runoob.com</h1>

<h2>Web Site Main Ingredients</h2>

<p>A Home Page (Default.cshtml)</p>
<p>A Layout File (Layout.cshtml)</p>
<p>A Style Sheet (Site.css)</p>
```

Layout.cshtml

```
<!DOCTYPE html>
<html>
<head>
<title>@Page.Title</title>
</head>
<body>
@RenderBody()
</body>
</html>
```



ASP.NET Web Pages - 文件

本章介绍有关使用文本文件的知识。

使用文本文件

在前面的章节中,我们已经了解到网页数据是存储在数据库中的。

您也可以把站点数据存储在文本文件中。

用来存储数据的文本文件通常被称为平面文件。常见的文本文件格式是 .txt、.xml 和 .csv (逗号分隔值)。

在本章中,您将学习到:

- 如何从文本文件中读取并显示数据

手动添加一个文本文件

在下面的例子中,您将需要一个文本文件。

在您的网站上,如果没有 **App_Data** 文件夹,请创建一个。在 **App_Data** 文件夹中,创建一个名为 **Persons.txt** 的文件。

添加以下内容到文件中:

Persons.txt

```
George, Lucas
Steven, Spielberg
Alfred, Hitchcock
```

显示文本文件中的数据

下面的实例演示了如何显示一个文本文件中的数据:

实例

```
@{
var dataFile = Server.MapPath("~/App_Data/Persons.txt");
Array userData = File.ReadAllLines(dataFile);
}

<!DOCTYPE html>
<html>
<body>

<h1>Reading Data from a File</h1>
@foreach (string dataLine in userData)
{
foreach (string dataItem in dataLine.Split(','))
```

```
{@dataItem <text>&nbsp;</text>}  
<br />  
}  
</body>  
</html>
```

[运行实例 »](#)

实例解释

使用 **Server.MapPath** 找到确切的文本文件的路径。

使用 **File.ReadAllLines** 打开文本文件，并读取文件中的所有行到一个数组中。

数组中的每个数据行中的数据项的数据被显示。

显示 Excel 文件中的数据

使用 **Microsoft Excel**，您可以将一个电子表格保存为一个逗号分隔的文本文件（.csv 文件）。此时，电子表格中的每一行保存为一个文本行，每个数据列由逗号分隔。

您可以使用上面的实例读取一个 **Excel .csv** 文件（只需将文件名改成相应的 **Excel** 文件的名称）。

[ASP.NET Web Pages 对象](#)

[ASP.NET Web Pages 帮助器](#)

[点我分享笔记](#)

[反馈/建议](#)

Copyright © 2013-2018 菜鸟教程 [runoob.com](#) All Rights Reserved. 备案号：闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

[ASP.NET Web Pages 文件](#)

[ASP.NET Web Pages WebGrid](#)

ASP.NET Web Pages - 帮助器

Web 帮助器大大简化了 Web 开发和常见的编程任务。

ASP.NET 帮助器

ASP.NET 帮助器是通过几行简单的 **Razor** 代码即可访问的组件。

您可以使用存放在 .cshtml 文件中的 **Razor** 语法构建自己的帮助器，或者使用内建的 **ASP.NET** 帮助器。

在本教程接下来的章节中，您将学到如何使用 **Razor** 帮助器。

下面是一些有用的 **Razor** 帮助器的简短说明：

WebGrid 帮助器

WebGrid 帮助器简化了显示数据的方式：

自动创建一个 **HTML** 表格来显示数据

支持不同的格式化选项

支持数据分页显示（第一页、下一页、上一页、最后一页）

Chart 帮助器

"Chart 帮助器" 能显示不同类型的带有多种格式化选项和标签的图表图像。

chart	chart
-------	-------

Chart 帮助器显示的数据来源可以是数组、数据库或者文件。

WebMail 帮助器

WebMail 帮助器提供了使用SMTP（Simple Mail Transfer Protocol 简单邮件传输协议）发送电子邮件的功能。

WebImage 帮助器

WebImage 帮助器提供了管理网页中图像的功能。

关键词：翻转、旋转、缩放、水印。

第三方帮助器

通过 **Razor**，您可以利用内建的或者第三方的帮助器来简化电子邮件、数据库、多媒体、社交网络以及很多其他问题（如导航和的网络安全）的使用。

安装帮助器

WebMatrix 已经包含了一些帮助器，您还可以手动安装其他的帮助器。

在 [runoob.com](#) 的 [WebPages 帮助器参考手册](#)中，您可以看到一个便捷的参考手册，包含了内建帮助器和其他可以通过手动安装附加到 ASP.NET Web Helpers Library 工具包中的帮助器。

如果您在 WebMatrix 中创建了一个网站，请按照下面的步骤安装帮助器：

1. 在 WebMatrix 中，打开 **Site** 工作区。
2. 点击 **Web Pages Administration**。
3. 使用密码 * 登录到 Web Pages Administration。
4. 使用 **搜索区** 搜索帮助器。
5. 点击 **Install** 安装您所需的帮助器。

（* 如果您是第一次使用 Web Pages Administration，会提示您创建一个密码。）

□

□ ASP.NET Web Pages 文件

ASP.NET Web Pages WebGrid □

□ [点我分享笔记](#)

反馈/建议

ASP.NET Web Pages - WebGrid 帮助器

WebGrid - 众多有用的 ASP.NET Web 帮助器之一。

自己写的 HTML

在前面的章节中，您使用 Razor 代码显示数据库数据，所有的 HTML 标记都是手写的：

数据库实例

```
@{
var db = Database.Open("SmallBakery");
var selectQueryString = "SELECT * FROM Product ORDER BY Name";
}
<html>
<body>
<h1>Small Bakery Products</h1>
<table>
<tr>
<th>Id</th>
<th>Product</th>
<th>Description</th>
<th>Price</th>
</tr>
@foreach(var row in db.Query(selectQueryString))
{
<tr>
<td>@row.Id</td>
<td>@row.Name</td>
<td>@row.Description</td>
<td align="right">@row.Price</td>
</tr>
}
</table>
</body>
</html>
```

运行实例 »

使用 WebGrid 帮助器

WebGrid 帮助器提供了一种更简单的显示数据的方法。

WebGrid 帮助器：

自动创建一个 HTML 表格来显示数据

支持不同的格式化选项

支持数据分页显示

支持通过点击列表标题进行排序

WebGrid 实例

```
@{
var db = Database.Open("SmallBakery") ;
var selectQueryString = "SELECT * FROM Product ORDER BY Id";
var data = db.Query(selectQueryString);
var grid = new WebGrid(data);
}
<html>
<head>
<title>Displaying Data Using the WebGrid Helper</title>
</head>
<body>
<h1>Small Bakery Products</h1>
<div id="grid">
@grid.GetHtml()
```

```
</div>
</body>
</html>
```

运行实例 »

ASP.NET Web Pages 帮助器

ASP.NET Web Pages 图表

点我分享笔记

反馈/建议

Copyright © 2013-2018 菜鸟教程 runoob.com All Rights Reserved. 备案号：闽ICP备15012807号-1

首页 HTML CSS JS 本地书签

ASP.NET Web Pages WebGrid

ASP.NET Web Pages Email

ASP.NET Web Pages - Chart 帮助器

Chart 帮助器 - 众多有用的 ASP.NET Web 帮助器之一。

Chart 帮助器

在前面的章节中，您已经学习了如何使用 ASP.NET 的 "帮助器"。

前面已经介绍了如何使用 "WebGrid 帮助器" 在网格中显示数据。

本章介绍如何使用 "Chart 帮助器" 以图形化的形式显示数据。

"Chart 帮助器" 可以创建不同类型的带有多种格式化选项和标签的图表图像。它可以创建面积图、条形图、柱形图、折线图、饼图等标准图表，也可以创建像股票图表这样的更专业的图表。

chart	chart
-------	-------

在图表中显示的数据可以是来自一个数组，一个数据库，或者一个文件中的数据。

根据数组创建图表

下面的实例显示了根据数组数据显示图表所需的代码：

实例

```
@{
var myChart = new Chart(width: 600, height: 400)
.AddTitle("Employees")
.AddSeries(chartType: "column",
xValue: new[] { "Peter", "Andrew", "Julie", "Mary", "Dave" },
yValues: new[] { "2", "6", "4", "5", "3" })
.Write();
}
```

运行实例 »

- **new Chart** 创建一个新的图表对象并且设置它的宽度和高度

- **AddTitle** 方法指定了图表的标题

- **AddSeries** 方法向图表中增加数据
- **chartType** 参数定义图表的类型
- **xValue** 参数定义 **x** 轴的名称
- **yValues** 参数定义 **y** 轴的名称
- **Write()** 方法显示图表

根据数据库创建图表

您可以执行一个数据库查询，然后使用查询结果中的数据来创建一个图表：

实例

```
@{
var db = Database.Open("SmallBakery");
var dbdata = db.Query("SELECT Name, Price FROM Product");
var myChart = new Chart(width: 600, height: 400)
.AddTitle("Product Sales")
.DataBindTable(dataSource: dbdata, xField: "Name")
.Write();
}
```

运行实例 »

- **var db = Database.Open** 打开数据库（将数据库对象赋值给变量 **db**）
- **var dbdata = db.Query** 执行数据库查询并保存结果在 **dbdata** 中
- **new Chart** 创建一个新的图表对象并且设置它的宽度和高度
- **AddTitle** 方法指定了图表的标题
- **DataBindTable** 方法将数据源绑定到图表
- **Write()** 方法显示图表

除了使用 **DataBindTable** 方法之外，另一种方法是使用 **AddSeries**（见前面的实例）。**DataBindTable** 更容易使用，但是 **AddSeries** 更加灵活，因为您可以更明确地指定图表和数据：

实例

```
@{
var db = Database.Open("SmallBakery");
var dbdata = db.Query("SELECT Name, Price FROM Product");
var myChart = new Chart(width: 600, height: 400)
.AddTitle("Product Sales")
.AddSeries(chartType: "Pie",
xValue: dbdata, xField: "Name",
yValues: dbdata, yFields: "Price")
.Write();
}
```

运行实例 »

根据 XML 数据创建图表

第三种创建图表的方法是使用 **XML** 文件作为图表的数据：

实例

```
@using System.Data;

@{
var dataSet = new DataSet();
dataSet.ReadXmlSchema(Server.MapPath("data.xsd"));
dataSet.ReadXml(Server.MapPath("data.xml"));
var dataView = new DataView(dataSet.Tables[0]);
var myChart = new Chart(width: 600, height: 400)
.AddTitle("Sales Per Employee")
.AddSeries("Default", chartType: "Pie",
xValue: dataView, xField: "Name",
```

```
yValues: dataView, yFields: "Sales")
.write();}
}
```

运行实例 »

ASP.NET Web Pages WebGrid

ASP.NET Web Pages Email

点我分享笔记

反馈/建议

Copyright © 2013-2018 菜鸟教程 runoob.com All Rights Reserved. 备案号: 闽ICP备15012807号-1



首页 HTML CSS JS 本地书签

ASP.NET Web Pages 图表

ASP.NET Web Pages PHP

ASP.NET Web Pages - WebMail 帮助器

WebMail 帮助器 - 众多有用的 ASP.NET Web 帮助器之一。

WebMail 帮助器

WebMail 帮助器让发送邮件变得更简单，它按照 SMTP（Simple Mail Transfer Protocol 简单邮件传输协议）从 Web 应用程序发送邮件。

前提：电子邮件支持

为了演示如何使用电子邮件，我们将创建一个输入页面，让用户提交一个页面到另一个页面，并发送一封关于支持问题的邮件。

第一：编辑您的 AppStart 页面

如果在本教程中您已经创建了 Demo 应用程序，那么您已经有一个名为 _AppStart.cshtml 的页面，内容如下：

_AppStart.cshtml

```
@{
WebSecurity.InitializeDatabaseConnection("Users", "UserProfile", "UserId", "Email", true);
}
```

要启动 WebMail 帮助器，向您的 AppStart 页面中增加如下所示的 WebMail 属性：

_AppStart.cshtml

```
@{
WebSecurity.InitializeDatabaseConnection("Users", "UserProfile", "UserId", "Email", true);
WebMail.SmtpServer = "smtp.example.com";
WebMail.SmtpPort = 25;
WebMail.EnableSsl = false;
WebMail.UserName = "support@example.com";
WebMail.Password = "password-goes-here";
WebMail.From = "john@example.com";
}
```

属性解释：

SmtpServer: 用于发送电子邮件的 SMTP 服务器的名称。

SmtpPort: 服务器用来发送 SMTP 事务（电子邮件）的端口。

EnableSsl: 如果服务器使用 SSL（Secure Socket Layer 安全套接层）加密，则值为 true。

UserName: 用于发送电子邮件的 SMTP 电子邮件账户的名称。

Password: SMTP 电子邮件账户的密码。

From: 在发件地址栏显示的电子邮件（通常与 UserName 相同）。

第二：创建一个电子邮件输入页面

接着创建一个输入页面，并将它命名为 Email_Input:

Email_Input.cshtml

```
<!DOCTYPE html>
<html>
<body>
<h1>Request for Assistance</h1>

<form method="post" action="EmailSend.cshtml">
<label>Username:</label>
<input type="text" name="customerEmail" />
<label>Details about the problem:</label>
<textarea name="customerRequest" cols="45" rows="4"></textarea>
<p><input type="submit" value="Submit" /></p>
</form>

</body>
</html>
```

输入页面的目的是手机信息，然后提交数据到可以将信息作为电子邮件发送的一个新的页面。

第三：创建一个电子邮件发送页面

接着创建一个用来发送电子邮件的页面，并将它命名为 Email_Send:

Email_Send.cshtml

```
@{ // Read input
var customerEmail = Request["customerEmail"];
var customerRequest = Request["customerRequest"];
try
{
// Send email
WebMail.Send(to:"someone@example.com", subject: "Help request from - " + customerEmail, body: customerRequest );
}
catch (Exception ex )
{
<text>@ex</text>
}
}
```

想了解更多关于 ASP.NET Web Pages 应用程序发送电子邮件的信息，请查阅：[WebMail 对象参考手册](#)。



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

☐ ASP.NET Web Pages Email

ASP.NET Web Pages – 发布 ☐

ASP.NET Web Pages - PHP

PHP 开发人员请注意，Web Pages 可以用 PHP 编写。

WebMatrix 支持 PHP

乍一看，认为 WebMatrix 只支持微软的技术。这是不正确的。在 WebMatrix 中，您能编写完整的 PHP 应用程序。

创建一个 PHP 站点

在[ASP.NET Web Pages - 创建一个网站](#)章节中，您已经创建了一个名为 "Demo" 的空网站，带有一个类型为 "CSHTML" 的空页面。

重复一遍创建的过程，创建一个名为 "Demo_PHP" 的空站点，勾选 "Enable PHP"（如下图所示），创建一个 PHP 类型的空白页，并将它命名 "index.php"，这样您就创建好了您的第一个 PHP 站点。

Screenshoot

创建一个 PHP 页面

将下面的代码复制到 "index.php" 文件中：

index.php

```
<!DOCTYPE html>
<html>
<body>

<?php
phpinfo();
?>

</body>
</html>
```

运行文件，看看 PHP 页面的演示。

☐ ASP.NET Web Pages Email

ASP.NET Web Pages – 发布 ☐

☐ 点我分享笔记

反馈/建议



ASP.NET Web Pages - 发布网站

学习如何在不使用 WebMatrix 的情况下发布 Web Pages 应用程序。

在不使用 WebMatrix 的情况下发布您的应用程序

通过在 WebMatrix（或者 Visual Studio）中使用发布命令，可以发布一个 ASP.NET Web Pages 应用程序到远程服务器上。

此功能会复制所有您的应用程序文件、cshtml 页面、图像以及用于 Web Pages、Razor、Helpers、SQL Server Compact（如果使用数据库）所有必需的 DLL 文件。

有时您不想使用 WebMatrix 发布您的应用程序。也许是因为您的托管服务提供商只支持 FTP，也许您已经有一个基于经典 ASP 的网站，也许您想自己复制所有的文件，也许您想使用 Front Page、Expression Web 等其他一些发布软件。

您会遇到问题吗？是的，会的。但是您有办法解决它。

要执行网站复制，您必须知道如何引用正确的文件，哪些 DLL 文件需要复制，并在何处存储它们。

请按照下列步骤操作：

1. 使用最新版本的 ASP.NET

在您继续操作之前，请确保您的主机运行的是最新版的 ASP.NET（4.0 或者 4.5）。

2. 复制 Web 文件夹

从您的开发计算机上复制您的网站（所有文件夹和内容）到远程主机（服务器）上的应用程序文件夹中。

☐ 如果您的应用程序中包含数据，不要复制数据（详见下面的第 4 点）。

3. 复制 DLL 文件

确保您的远程主机上的 bin 文件夹中包含了和您开发计算机上相同的 dll 文件。

复制 bin 文件夹之后，它应该包含以下文件：

Microsoft.Web.Infrastructure.dll

NuGet.Core.dll

System.Web.Helpers.dll

System.Web.Razor.dll

System.Web.WebPages.Administration.dll

System.Web.WebPages.Deployment.dll

System.Web.WebPages.dll

System.Web.WebPages.Razor.dll

WebMatrix.Data.dll

WebMatrix.WebData

4. 复制您的数据

如果您的应用程序包含数据或者数据库。例如 SQL Server Compact 数据库（在 App_Data 文件夹中的一个 .sdf 文件），请考虑以下几点：

您是否希望发布您的测试数据到远程服务器上？

大多数时候一般是不希望。

如果在您的开发计算机上有测试数据，它将覆盖您的远程主机上的生产数据。

如果您一定要复制 SQL 数据库（.sdf 文件），那么您应该删除数据库中的所有数据，然后从您的开发计算机上复制一个空的 .sdf 文件到服务器上。

就是这样。**GOOD LUCK!**



ASP.NET Web Pages - C# 和 VB 实例

通过 C# 和 Visual Basic 实例学习 ASP.NET Web Pages。

C# 实例	VB 实例
<p>基本的 Web Pages</p> <p>显示日期和时间</p> <p>可重复使用的头部和底部</p> <p>基本的 HTML 表单</p> <p>实例解释</p>	<p>基本的 Web Pages</p> <p>显示日期和时间</p> <p>可重复使用的头部和底部</p> <p>基本的 HTML 表单</p> <p>实例解释</p>
<p>基本的 C#</p> <p>For 循环</p> <p>For Each 循环</p> <p>While 循环</p> <p>Array 数组</p> <p>If 条件</p> <p>If Else 条件</p> <p>Else If 条件</p> <p>Switch 条件</p> <p>实例解释</p>	<p>基本的 VB</p> <p>For 循环</p> <p>For Each 循环</p> <p>While 循环</p> <p>Array 数组</p> <p>If 条件</p> <p>If Else 条件</p> <p>Else If 条件</p> <p>Select 条件</p> <p>实例解释</p>
<p>使用数据库</p> <p>显示数据库数据</p> <p>使用 WebGrid 显示数据</p> <p>实例解释</p>	<p>使用数据库</p> <p>显示数据库数据</p> <p>使用 WebGrid 显示数据</p> <p>实例解释</p>

使用 **Chart** 帮助器

[根据数组显示条形图](#)

[根据数据库显示条形图](#)

[根据数据库显示饼图](#)

[根据 XML 文件显示饼图](#)

[实例解释](#)

使用 **Chart** 帮助器

[根据数组显示条形图](#)

[根据数据库显示条形图](#)

[根据数据库显示饼图](#)

[根据 XML 文件显示饼图](#)

[实例解释](#)

[ASP.NET Web Pages – 发布](#)

ASP.NET Web Pages 类参考手册 [□](#)

[□ 点我分享笔记](#)

[反馈/建议](#)

Copyright © 2013-2018 菜鸟教程 [runoob.com](#) All Rights Reserved. 备案号：闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

[□ ASP.NET Web 的 C# 和 VB 实例](#)

ASP.NET Web Pages WebSecurity 参考手册 [□](#)

ASP.NET Web Pages - 类

ASP.NET 类参考手册

方法	描述
AsBool(), AsBool(true false)	转换字符串值为布尔值（true/false）。如果字符串不能转换为 true/false，则返回 false 或者其他规定的值。
AsDateTime(), AsDateTime(value)	转换字符串值为日期/时间。返回 DateTime。如果字符串不能转换为日期/时间，则返回 MinValue 或者其他规定的值。
AsDecimal(), AsDecimal(value)	转换字符串值为十进制值。如果字符串不能转换为十进制值，则返回 0.0 或者其他规定的值。
AsFloat(), AsFloat(value)	转换字符串值为浮点数。如果字符串不能转换为浮点数，则返回 0.0 或者其他规定的值。
AsInt(), AsInt(value)	转换字符串值为整数。如果字符串不能转换成整数，则返回 0 或者其他规定的值。
Href(path [, param1 [, param2]])	从带有可选的附加路径部分的本地文件路径创建一个浏览器兼容的 URL。
Html.Raw(value)	Renders value 呈现为 HTML 标记，而不是呈现为 HTML 编码输出。
IsBool(), IsDateTime(), IsDecimal(), IsFloat(), IsInt()	如果该值可以从字符串转换为指定的类型，则返回 true。
IsEmpty()	如果对象或者变量没有值，则返回 true。

IsPost	如果请求是 POST ，则返回 true 。（初始请求通常是 GET 。）
Layout	规定布局页面的路径应用到此页面。
PageData[key], PageData[index], Page	在当前请求的页面、布局页面、部分页面之间包含共享数据。您可以使用动态页面来对相同的数据进行属性访问。
RenderBody()	(Layout pages) 呈现没有在布局页面任何命名区域的内容页的内容 Renders the content of a content page that is not in any named sections.
RenderPage(path, values) RenderPage(path[, param1 [, param2]])	呈现使用了规定的路径和可选的额外数据的内容页。您可以通过 position （实例 1）或者 key （实例 2）从 PageData 获取额外参数的值。
RenderSection(sectionName [, required = true false])	(Layout pages) 呈现一个有名字的内容区域。设置 required 让一个区域为必需非可选的。
Request.Cookies[key]	获取或者设置 HTTP cookie 的值。
Request.Files[key]	Gets 在当前请求中上传的文件。
Request.Form[key]	获取在表单中 post 的数据（作为字符串）。 Request.Form 和 Request.QueryString 都要求[key] 检查。
Request.QueryString[key]	获取 URL 查询字符串中规定的数据。 Request.Form 和 Request.QueryString 都要求[key] 检查。
Request.Unvalidated(key) Request.Unvalidated().QueryString Form Cookies Headers[key]	有选择地禁用请求验证（表单元素、查询字符串值、 cookie 、 header 值）。请求验证默认是开启的，防止用户提交标记或者其他潜在的危险内容。
Response.AddHeader(name, value)	在应答中添加一个 HTTP 服务器响应头。
Response.OutputCache(seconds [, sliding] [, varyByParams])	Caches 在指定时间的页面输出缓存。设置 sliding 来重置每个页面的访问超时时间，设置 varyByParams 为请求页面的每个不同的查询字符串缓存不同版本的页面。
Response.Redirect(path)	重定向浏览器请求到一个新的位置。
Response.SetStatus(httpStatusCode)	设置 HTTP 状态代码发送到浏览器。
Response.WriteBinary(data [, mimeType])	写入 data 内容响应可选的 MIME 类型。
Response.WriteFile(file)	写入文件内容响应。
@section(sectionName) { content }	（布局页面）定义一个有名字的内容区域。
Server.HtmlDecode(htmlText)	解码一个 HTML 编码的字符串。
Server.HtmlEncode(text)	为呈现在 HTML 标记中的字符串编码。
Server.MapPath(virtualPath)	为指定的虚拟路径返回服务器的物理路径。
Server.UrlDecode(urlText)	解码 URL 文本。
Server.UrlEncode(text)	URL 文本编码。
Session[key]	获取或设置一个存在的值，直到用户关闭浏览器。

ToString()	显示一个用字符串表示的对象的值。
UrlData[index]	从 URL 获取额外的数据（例如， <i>/MyPage/ExtraData</i> ）。

[☐ ASP.NET Web 的 C# 和 VB 实例](#)

[ASP.NET Web Pages WebSecurity 参考手册](#) ☐

[☐ 点我分享笔记](#)

反馈/建议

Copyright © 2013-2018 菜鸟教程 [runoob.com](#) All Rights Reserved. 备案号：闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

[☐ ASP.NET Web Pages 类参考手册](#)

[ASP.NET Web Pages Database 参考手册](#) ☐

ASP.NET Web Pages - WebSecurity 对象

描述

WebSecurity 对象提供 ASP.NET Web Pages 应用程序的安全性和认证。

通过 WebSecurity 对象，您可以创建用户帐户，登录和注销用户，重置或者更改密码，以及其他更多与安全性相关的功能。

WebSecurity 对象参考手册 - 属性

属性	描述
CurrentUserId	获取当前登录用户的 ID。
CurrentUserName	获取当前登录用户的名称。
HasUserId	如果当前有用户 ID，则返回 true 。
IsAuthenticated	如果当前用户是登录的，则返回 true 。

WebSecurity 对象参考手册 - 方法

方法	描述
ChangePassword()	为指定的用户更改密码。
ConfirmAccount()	使用帐户确认令牌确认帐户。
CreateAccount()	创建一个新的用户帐户。
CreateUserAndAccount()	创建一个新的用户帐户。
GeneratePasswordResetToken()	生成一个密码重置令牌，可以在电子邮件中发送给用户以便用户可以重设密码。

GetCreateDate()	获取指定会员创建的时间。
GetPasswordChangeDate()	获取密码变更的日期和时间。
GetUserId()	根据用户名称获取用户 ID。
InitializeDatabaseConnection()	初始化 WebSecurity 系统（数据库）。
IsConfirmed()	检查用户是否已被确认。如果已确认，则返回 true 。（例如，可通过电子邮件进行确认。）
IsCurrentUser()	检查当前用户的名称是否与指定用户名匹配。如果匹配，则返回 true 。
Login()	设置身份验证令牌，登录用户。
Logout()	移除身份验证令牌，注销用户。
RequireAuthenticatedUser()	如果用户未通过身份验证，则设置 HTTP 状态为 401 （未经授权）。
RequireRoles()	如果当前用户不是指定角色的成员，则设置 HTTP 状态为 401 （未经授权）。
RequireUser()	如果当前用户不是指定用户名的用户，则设置 HTTP 状态为 401 （未经授权）。
ResetPassword()	如果密码重置令牌是有效的，改变用户的密码为新密码。
UserExists()	检查指定的用户是否存在。

技术数据

名称	值
Class	WebMatrix.WebData.WebSecurity
Namespace	WebMatrix.WebData
Assembly	WebMatrix.WebData.dll

初始化 **WebSecurity** 数据库

如果您想在您的代码中使用 **WebSecurity** 对象，首先您必须创建或者初始化 **WebSecurity** 数据库。

在您的 **Web** 根目录下，创建一个名为 **_AppStart.cshtml** 的页面（如果已存在，则直接编辑页面）。

将下面的代码复制到文件中：

_AppStart.cshtml

@{
WebSecurity.InitializeDatabaseConnection("Users", "UserProfile", "UserId", "Email", true);
}

上面的代码将在每次网站（应用程序）启动时运行。它初始化了 **WebSecurity** 数据库。

"Users" 是 **WebSecurity** 数据库（**Users.sdf**）的名称。

"UserProfile" 是包含用户配置信息的数据库表的名称。

"UserId" 是包含用户 **ID**（主键）的列的名称。

"Email" 是包含用户名的列的名称。

最后一个参数 **true** 是一个布尔值，表示如果用户配置表和会员表不存在，则会自动创建表。如果不想自动创建表，应设置参数为 **false**。

☐ 虽然 **true** 表示自动创建数据库 表，但是数据库不会被自动创建。所以数据库必须存在。

WebSecurity 数据库

UserProfile 表为每个用户创建保存一条记录，用户 ID（主键）和用户名字（email）：

Userld	Email
1	john@johnson.net
2	peter@peterson.com
3	lars@larson.eut

Membership 表包含会员信息，比如用户是什么时候创建的，该会员是否已认证，会员是什么时候认证的，等等。

具体如下所示（一些列不显示）：

User Id	Create Date	Confirmation Token	Is Confirmed	Last Password Failure	Password	Password Change
1	12.04.2012 16:12:17	NULL	True	NULL	AFNQhWfy....	12.04.2012 16:12:17

注释：如果您想看到所有的列和内容，请打开数据库，看看里边的每个表。

简单的会员配置

在您使用 WebSecurity 对象时，如果您的站点没有配置使用 ASP.NET Web Pages 会员系统 SimpleMembership，可能会报错。

如果托管服务提供商的服务器的配置与您本地服务器的配置不同，也可能会报错。为了解决这个问题，请在网站的 Web.config 文件中添加以下元素：

```
<appSettings>
<add key="enableSimpleMembership" value="true" />
</appSettings>
```



ASP.NET Web Pages - Database 对象

ASP.NET Database 对象参考手册

方法	描述
Database.Execute(SQLstatement [, parameters])	执行 SQL 语句 SQLstatement（带可选参数），比如 INSERT、DELETE 或者 UPDATE，并且返回受影响的记录统计。
Database.GetLastInsertId()	返回最近插入行的标识列。

Database.Open(<i>filename</i>) Database.Open(<i>connectionStringName</i>)	使用 <i>Web.config</i> 文件中的连接字符串打开指定的数据库文件或者指定的数据库。
Database.OpenConnectionString(<i>connectionString</i>)	使用连接字符串打开一个数据库。（与 Database.Open 的差异是，Database.Open 使用的是连接字符串的名称，连接字符串的值在其他地方配置。）
Database.Query(<i>SQLstatement</i> [, <i>parameters</i>])	使用 SQL 语句 <i>SQLstatement</i> （带可选参数）查询数据库，并返回结果集合。
Database.QuerySingle(<i>SQLstatement</i> [, <i>parameters</i>])	执行 SQL 语句 <i>SQLstatement</i> （带可选参数），并返回单条记录。
Database.QueryValue(<i>SQLstatement</i> [, <i>parameters</i>])	执行 SQL 语句 <i>SQLstatement</i> （带可选参数），并返回单个值。

[ASP.NET Web Pages WebSecurity 参考手册](#)

[ASP.NET Web Pages WebMail 参考手册](#)

[点我分享笔记](#)

反馈/建议

Copyright © 2013-2018 菜鸟教程 [runoob.com](#) All Rights Reserved. 备案号：闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

[ASP.NET Web Pages Database 参考手册](#)

[ASP.NET WebPages 帮助器参考手册](#)

ASP.NET Web Pages - WebMail 对象

通过 WebMail 对象，您可以很容易地从网页上发送电子邮件。

描述

WebMail 对象为 ASP.NET Web Pages 提供了使用 SMTP（Simple Mail Transfer Protocol 简单邮件传输协议）发送邮件的功能。

实例

请查看 [WebPages Email](#) 章节中的实例。

WebMail 对象参考手册 - 属性

属性	描述
SmtpServer	用于发送电子邮件的 SMTP 服务器的名称。
SmtpPort	服务器用来发送 SMTP 电子邮件的端口。
EnableSsl	如果服务器使用 SSL（Secure Socket Layer 安全套接层）加密，则值为 true。
UserName	用于发送电子邮件的 SMTP 电子邮件账户的名称。

Password	SMTP 电子邮件账户的密码。
From	在发件地址栏显示的电子邮件（通常与 UserName 相同）。

WebMail 对象参考手册 - 方法

方法	描述
Send()	向 SMTP 服务器发送需要传送的电子邮件信息。

Send() 方法有以下参数：

参数	类型	描述
to	String	收件人（用分号分隔）
subject	String	邮件主题
body	String	邮件正文

Send() 方法有以下可选参数：

参数	类型	描述
from	String	发件人
cc	String	需要抄送的电子邮件地址（用分号分隔）
filesToAttach	Collection	附件名
isBodyHtml	Boolean	如果邮件正文是 HTML 格式的，则为 true
additionalHeaders	Collection	附加的标题

技术数据

名称	值
Class	System.Web.Helpers.WebMail
Namespace	System.Web.Helpers
Assembly	System.Web.Helpers.dll

初始化 WebMail 帮助器

要使用 **WebMail** 帮助器，您必须能访问 **SMTP** 服务器。**SMTP** 是电子邮件的"输出"部分。如果您使用的是虚拟主机，您可能已经知道 **SMTP** 服务器的名称。如果您使用的是公司网络工作，您公司的 **IT** 部门会给您一个名称。如果您是在家工作，你也许可以使用普通的电子邮件服务提供商。为了发送一封电子邮件，您将需要：

SMTP 服务器的名称

端口号（通常是 25）

电子邮件的用户名

电子邮件的密码

在您的 **Web** 根目录下，创建一个名为 **_AppStart.cshtml** 的页面（如果已存在，则直接编辑页面）。

将下面的代码复制到文件中：

```
_AppStart.cshtml
@{
    WebMail.SmtpServer = "smtp.example.com";
}
```



```
WebMail.SmtpPort = 25;
WebMail.EnableSsl = false;
WebMail.UserName = "support@example.com";
WebMail.Password = "password";
WebMail.From = "john@example.com"
}
```

上面的代码将在每次网站（应用程序）启动时运行。它对 **WebMail** 对象赋了初始值。

请替换：

将 **smtp.example.com** 替换成您用来发送电子邮件的 SMTP 服务器的名称。

将 **25** 替换成服务器用来发送 SMTP 事务（电子邮件）的端口号。

如果服务器使用 SSL（Secure Socket Layer 安全套接层）加密，请将 **false** 替换成 **true**。

将 **support@example.com** 替换成用来发送电子邮件的 SMTP 电子邮件账户的名称。

将 **password** 替换成 SMTP 电子邮件账户的密码。

将 **john@example** 替换成显示在发件地址栏中的电子邮件。

☐ 在您的 **AppStart** 文件中，您不需要启动 **WebMail** 对象，但是在调用 **WebMail.Send()** 方法之前，您必须设置这些属性。

☐ ASP.NET Web Pages Database 参考手册

ASP.NET WebPages 帮助器参考手册 ☐

☐ 点我分享笔记

反馈/建议

Copyright © 2013-2018 菜鸟教程 [runoob.com](#) All Rights Reserved. 备案号：闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

☐ ASP.NET Web Pages WebMail 参考手册

ASP.NET Razor 标记 ☐

ASP.NET Web Pages - 更多帮助器

ASP.NET 帮助器 - 对象参考手册

Analytics 对象参考手册（Google）

Helper	描述
Analytics.GetGoogleHtml(<i>webPropertyId</i>)	为指定的 ID 呈现 Google Analytics JavaScript 代码。
Analytics.GetStatCounterHtml(<i>project</i> , <i>security</i>)	为指定的项目呈现 StatCounter Analytics JavaScript 代码。
Analytics.GetYahooHtml(<i>account</i>)	为指定的账号呈现 Yahoo Analytics JavaScript 代码。

Bing 对象参考手册

Helper	描述
--------	----

Bing.SearchBox(<i>[boxWidth]</i>)	给 Bing 传递搜索。您可以设置 Bing.SiteUrl 和 Bing.SiteTitle 属性来设定站点搜索和搜索框的标题，通常是在 <i>_AppStart</i> 页面设置这些属性。
Bing.AdvancedSearchBox(<i>[, boxWidth] [, resultWidth] [, resultHeight] [, themeColor] [, locale]</i>)	用可选的格式显示 Bing 搜索结果在页面上。您可以设置 Bing.SiteUrl 和 Bing.SiteTitle 属性来设定站点搜索和搜索框的标题，通常是在 <i>_AppStart</i> 页面设置这些属性。

Chart 对象参考手册

Helper	描述
Chart(<i>width, height [, template] [, templatePath]</i>)	初始化图表。
Chart.AddLegend(<i>[title] [, name]</i>)	给图表添加一个图例。
Chart.AddSeries(<i>[name] [, chartType] [, chartArea] [, axisLabel] [, legend] [, markerStep] [, xValue] [, xField] [, yValues] [, yFields] [, options]</i>)	给图表添加一系列数据。

Crypto 对象参考手册

Helper	描述
Crypto.Hash(<i>string [, algorithm]</i>) Crypto.Hash(<i>bytes [, algorithm]</i>)	返回指定数据的哈希。默认算法是 sha256。

Facebook 对象参考手册

Helper	描述
Facebook.LikeButton(<i>href [, buttonLayout] [, showFaces] [, width] [, height] [, action] [, font] [, colorScheme] [, refLabel]</i>)	让 Facebook 用户连接到网页。

FileUpload 对象参考手册

Helper	描述
FileUpload.GetHtml(<i>[initialNumberOfFiles] [, allowMoreFilesToBeAdded] [, includeFormTag] [, addText] [, uploadText]</i>)	为上传文件呈现 UI。

GamerCard 对象参考手册

Helper	描述
GamerCard.GetHtml(<i>gamerTag</i>)	呈现指定的 Xbox gamer 标签。

Gravatar 对象参考手册

Helper	描述
Gravatar.GetHtml(<i>email [, imageSize] [, defaultImage] [, rating] [, imageExtension] [, attributes]</i>)	为指定的电子邮件地址呈现 Gravatar 图像。

Json 对象参考手册

Helper	描述
Json.Encode(object)	用 JavaScript Object Notation (JSON) 把数据对象转换为字符串。
Json.Decode(string)	转换 JSON 编码的输入字符串为您指定的数据对象。

LinkShare 对象参考手册

Helper	描述
LinkShare.GetHtml(pageTitle [, pageLinkBack] [, twitterUserName] [, additionalTweetText] [, link Sites])	使用指定的标题和可选的 URL 呈现社会网络链接。

ModelState 对象参考手册

Helper	描述
ModelStateDictionary.AddError(key, errorMessage)	关联错误信息和一个表单域。使用 ModelState 帮助器访问成员。
ModelStateDictionary.AddFormError(errorMessage)	关联错误信息和一个表单。使用 ModelState 帮助器访问成员。
ModelStateDictionary.IsValid	如果没有验证错误，返回 true。使用 ModelState 帮助器访问成员。

ObjectInfo 对象参考手册

Helper	描述
ObjectInfo.Print(value [, depth] [, enumerationLength])	呈现一个对象和所有子对象的属性和值。

Recaptcha 对象参考手册

Helper	描述
Recaptcha.GetHtml([, publicKey] [, theme] [, language] [, tabIndex])	呈现 reCAPTCHA 验证测试。
ReCaptcha.PublicKey ReCaptcha.PrivateKey	设置 reCAPTCHA 服务的公共和私有密钥。通常是在 _AppStart 页面设置这些属性。
ReCaptcha.Validate([, privateKey])	返回 reCAPTCHA 测试结果。
ServerInfo.GetHtml()	Renders 呈现有关 ASP.NET Web Pages 的状态信息。

Twitter 对象参考手册

Helper	描述
Twitter.Profile(twitterUserName)	为指定的用户呈现 Twitter 流。
Twitter.Search(searchQuery)	为指定的搜索文本呈现 Twitter 流。

Video 对象参考手册

Helper	描述
Video.Flash(filename [, width, height])	为指定的文件呈现宽度和高度可选的 Flash 视频播放。
Video.MediaPlayer(filename [, width, height])	为指定的文件呈现宽度和高度可选的 Windows Media 播放器。

Video.Silverlight(<i>filename, width, height</i>)	为指定的 <i>.xap</i> 文件呈现所需的宽度和高度 的 Silverlight 播放器。
---	--

WebCache 对象参考手册

Helper	描述
WebCache.Get(<i>key</i>)	通过 <i>key</i> 返回指定的对象，如果对象未找到则返回 null 。
WebCache.Remove(<i>key</i>)	通过 <i>key</i> 从缓存中删除指定的对象。
WebCache.Set(<i>key, value [, minutesToCache] [, slidingExpiration]</i>)	通过 <i>key</i> 把 <i>value</i> 放置到指定名称的缓存中。

WebGrid 对象参考手册

Helper	描述
WebGrid(<i>data</i>)	Creates a 使用查询数据创建一个新的 WebGrid 对象。
WebGrid.GetHtml()	Renders markup 显示数据在 HTML 表格中。
WebGrid.Pager()	为 WebGrid 对象呈现一个页面。

WebImage 对象参考手册

Helper	描述
WebImage(<i>path</i>)	从指定的路径加载一个图像。
WebImage.AddImagesWatermark(<i>image</i>)	为指定图像加水印。
WebImage.AddTextWatermark(<i>text</i>)	为图像添加指定文本。
WebImage.FlipHorizontal() WebImage.FlipVertical()	水平/垂直翻转图像
WebImage.GetImageFromRequest()	当图像被传送到一个文件上传页面时，加载图像。
WebImage.Resize(<i>width, height</i>)	调整图像大小。
WebImage.RotateLeft() WebImage.RotateRight()	向左或向右旋转图像。
WebImage.Save(<i>path [, imageFormat]</i>)	保存图像到指定路径。

[☐ ASP.NET Web Pages WebMail 参考手册](#)

[ASP.NET Razor 标记 ☐](#)

[☐ 点我分享笔记](#)

反馈/建议

ASP.NET Razor - 标记

Razor 不是一种编程语言。它是服务器端的标记语言。

什么是 **Razor**?

Razor 是一种标记语法，可以让您将基于服务器的代码（**Visual Basic** 和 **C#**）嵌入到网页中。

基于服务器的代码可以在网页传送给浏览器时，创建动态 **Web** 内容。当一个网页被请求时，服务器在返回页面给浏览器之前先执行页面中的基于服务器的代码。通过服务器的运行，代码能执行复杂的任务，比如进入数据库。

Razor 是基于 **ASP.NET** 的，是为创建 **Web** 应用程序而设计的。它具有传统 **ASP.NET** 的功能，但更容易使用并且更容易学习。

Razor 语法

Razor 使用了与 **PHP** 和经典 **ASP** 相似的语法。

Razor:

```
<ul>
@for (int i = 0; i < 10; i++) {
<li>@i</li>
}
</ul>
```

PHP:

```
<ul>
<?php
for ($i = 0; $i < 10; $i++) {
echo("<li>$i</li>");
}
?>
</ul>
```

Web Forms（经典 **ASP**）:

```
<ul>
<% for (int i = 0; i < 10; i++) { %>
<li><% =i %></li>
<% } %>
</ul>
```

Razor 帮助器

ASP.NET 帮助器是通过几行简单的 **Razor** 代码即可访问的组件。

您可以使用 **Razor** 语法构建自己的帮助器，或者使用内建的 **ASP.NET** 帮助器。

下面是一些有用的 **Razor** 帮助器的简短说明：

[Web Grid](#)（**Web** 网格）

[Web Graphics](#)（**Web** 图形）

[Google Analytics](#)（**Google** 分析）

[Facebook Integration](#)（**Facebook** 集成）

[Twitter Integration](#)（**Twitter** 集成）

[Sending Email](#)（发送电子邮件）

[Validation](#)（验证）

Razor 编程语言

Razor 支持 **C#** (**C sharp**) 和 **VB** (**Visual Basic**)。

[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

ASP.NET Razor - C# 和 VB 代码语法

Razor 同时支持 C# (C sharp) 和 VB (Visual Basic)。

主要的 Razor C# 语法规则

Razor 代码块包含在 `@{ ... }` 中

内联表达式（变量和函数）以 `@` 开头

代码语句用分号结束

变量使用 `var` 关键字声明

字符串用引号括起来

C# 代码区分大小写

C# 文件的扩展名是 `.cshtml`

C# 实例

```
<!-- Single statement block -->
@{ var myMessage = "Hello World"; }

<!-- Inline expression or variable -->
<p>The value of myMessage is: @myMessage</p>

<!-- Multi-statement block -->
@{
var greeting = "Welcome to our site!";
var weekDay = DateTime.Now.DayOfWeek;
var greetingMessage = greeting + " Here in Huston it is: " + weekDay;
}
<p>The greeting is: @greetingMessage</p>
```

[运行实例 »](#)

主要的 Razor VB 语法规则

Razor 代码块包含在 `@Code ... End Code` 中

内联表达式（变量和函数）以 `@` 开头

变量使用 Dim 关键字声明

字符串用引号括起来

VB 代码不区分大小写

VB 文件的扩展名是 .vbhtml

实例

```
<!-- Single statement block -->
@Code dim myMessage = "Hello World" End Code

<!-- Inline expression or variable -->
<p>The value of myMessage is: @myMessage</p>

<!-- Multi-statement block -->
@Code
dim greeting = "Welcome to our site!"
dim weekDay = DateTime.Now.DayOfWeek
dim greetingMessage = greeting & " Here in Huston it is: " & weekDay
End Code

<p>The greeting is: @greetingMessage</p>
```

运行实例 »

它是如何工作的？

Razor 是一种将服务器代码嵌入在网页中的简单的编程语法。

Razor 语法是基于 ASP.NET 框架，专门用于创建 Web 应用程序的部分 Microsoft.NET 框架。

Razor 语法支持所有 ASP.NET 的功能，但是使用的是一种简化语法，对初学者而言更容易学习，对专家而言更有效率的。

Razor 网页可以被描述成带以下两种类型内容的 HTML 网页：HTML 内容和 Razor 代码。

当服务器读取页面时，它首先运行 Razor 代码，然后再发送 HTML 页面到浏览器。在服务器上执行的代码能够执行一些在浏览器上不能完成的任务，比如，访问服务器数据库。服务器代码能创建动态的 HTML 内容，然后发送到浏览器。从浏览器上看，服务器代码生成的 HTML 与静态的 HTML 内容没有什么不同。

带 Razor 语法的 ASP.NET 网页有特殊的文件扩展名 cshtml（Razor C#）或者 vbhtml（Razor VB）。

使用对象

服务器编码往往涉及到对象。

"Date" 对象是一个典型的内置的 ASP.NET 对象，但对象也可以是自定义的，一个网页，一个文本框，一个文件，一个数据库记录，等等。

对象有用于执行的方法。一个数据库记录可能有一个 "Save" 方法，一个图像对象可能有一个 "Rotate" 方法，一个电子邮件对象可能有一个 "Send" 方法，等等。

对象也有用于描述各自特点的属性。一个数据库记录可能有 FirstName 和 LastName 属性。

ASP.NET Date 对象有一个 Now 属性（写成 Date.Now），Now 属性有一个 Day 属性（写成 Date.Now.Day）。下面实例演示了如何访问 Date 对象的一些属性：

实例

```
<table border="1">
<tr>
<th width="100px">Name</th>
<td width="100px">Value</td>
</tr>
<tr>
<td>Day</td><td>@DateTime.Now.Day</td>
</tr>
<tr>
<td>Hour</td><td>@DateTime.Now.Hour</td>
</tr>
<tr>
<td>Minute</td><td>@DateTime.Now.Minute</td>
</tr>
```

```
<tr>
<td>Second</td><td>@DateTime.Now.Second</td>
</tr>
</td>
</table>
```

运行实例 »

If 和 Else 条件

动态网页的一个重要特点是，您可以根据条件决定做什么。

做到这一点的常用方法是使用 if ... else 语句：

实例

```
@{
var txt = "";
if(DateTime.Now.Hour > 12)
{txt = "Good Evening";}
else
{txt = "Good Morning";}
}
<html>
<body>
<p>The message is @txt</p>
</body>
</html>
```

运行实例 »

读取用户输入

动态网页的另一个重要特点是，您可以读取用户输入。

输入是通过 Request[] 功能读取的，并且传送输入数据是经过 IsPost 条件判断的：

实例

```
@{
var totalMessage = "";
if(IsPost)
{
var num1 = Request["text1"];
var num2 = Request["text2"];
var total = num1.AsInt() + num2.AsInt();
totalMessage = "Total = " + total;
}
}
<html>
<body style="background-color: beige; font-family: Verdana, Arial;">
<form action="" method="post">
<p><label for="text1">First Number:</label><br>
<input type="text" name="text1" /></p>
<p><label for="text2">Second Number:</label><br>
<input type="text" name="text2" /></p>
<p><input type="submit" value=" Add " /></p>
</form>
<p>@totalMessage</p>
</body>
</html>
```

运行实例 »



ASP.NET Razor - C# 变量

变量是用来存储数据的命名实体。

变量

变量是用来存储数据的。

一个变量的名称必须以字母字符开头，并且不能包含空格或者保留字符。

一个变量可以是一个指定的类型，表示它所存储的数据类型。`string` 变量存储字符串值（"Welcome to RUNOOB.COM"），`integer` 变量存储数字值（103），`date` 变量存储日期值，等等。

变量使用 `var` 关键字声明，或通过使用类型（如果您想声明类型）声明，但是 `ASP.NET` 通常能自动确定数据类型。

实例

```
// Using the var keyword:
var greeting = "Welcome to RUNOOB.COM";
var counter = 103;
var today = DateTime.Today;

// Using data types:
string greeting = "Welcome to RUNOOB.COM";
int counter = 103;
DateTime today = DateTime.Today;
```

数据类型

下面列出了常用的数据类型：

类型	描述	实例
int	整数（全数字）	103, 12, 5168
float	浮点数	3.14, 3.4e38
decimal	十进制数字（高精度）	1037.196543
bool	布尔值	true, false
string	字符串	"Hello RUNOOB.COM", "John"

运算符

运算符告诉 **ASP.NET** 在表达式中执行什么样的命令。

C# 语言支持多种运算符。下面列出了常用的运算符：

运算符	描述	实例
=	给一个变量赋值。	i=6
+	加上一个值或者一个变量。	i=5+5
-	减去一个值或者一个变量。	i=5-5
*	乘以一个值或者一个变量。	i=5*5
/	除以一个值或者一个变量。	i=5/5
+=	变量递增。	i += 1
-=	变量递减。	i -= 1
==	相等。如果值相等则返回 true 。	if (i==10)
!=	不等。如果值不等则返回 true 。	if (i!=10)
<	小于。	if (i<10)
>	大于。	if (i>10)
<=	小于等于。	if (i<=10)
>=	大于等于。	if (i>=10)
+	连接字符串（一系列互相关联的事物）。	"run" + "oob"
.	点号。分隔对象和方法。	DateTime.Hour
()	圆括号。将值进行分组。	(i+5)
()	圆括号。传递参数。	x=Add(i,5)
[]	中括号。访问数组或者集合的值。	name[3]
!	非。真/假取反。	if (!ready)
&&	逻辑与。	if (ready && clear)
	逻辑或。	if (ready clear)

转换数据类型

从一种数据类型转换到另一种数据类型，有时候是很有用的。

最常见的例子是将字符串输入转换为另一种类型，如整数或者日期。

一般规则下，都是将用户输入看做字符串处理，即使用户输入了数字。因此数值输入必须被转换成数字，然后才能将其用于计算。

下面列出了常用的转换方法：

方法	描述	实例
AsInt() IsInt()	转换字符串为整数。	if (myString.IsInt()) {myInt=myString.AsInt();}
AsFloat() IsFloat()	转换字符串为浮点数。	if (myString.IsFloat()) {myFloat=myString.AsFloat();}
AsDecimal() IsDecimal()	转换字符串为十进制数。	if (myString.IsDecimal()) {myDec=myString.AsDecimal();}

AsDateTime() IsDateTime()	转换字符串为 ASP.NET DateTime 类型。	myString="10/10/2012"; myDate=myString.AsDateTime();
AsBool() IsBool()	转换字符串为布尔值。	myString="True"; myBool=myString.AsBool();
ToString()	转换任何数据类型为字符串。	myInt=1234; myString=myInt.ToString();

[❏ ASP.NET Razor 语法](#)

[ASP.NET Razor C# 循环和数组](#) ❏

[❏ 点我分享笔记](#)

反馈/建议

Copyright © 2013-2018 菜鸟教程 [runoob.com](#) All Rights Reserved. 备案号: 闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

[❏ ASP.NET Razor C# 变量](#)

[ASP.NET Razor C# 逻辑](#) ❏

ASP.NET Razor - C# 循环和数组

语句在循环中会被重复执行。

For 循环

如果您需要重复执行相同的语句，您可以设定一个循环。

如果您知道要循环的次数，您可以使用 **for 循环**。这种类型的循环在向上计数或向下计数时特别有用：

实例

```
<html>
<body>
@for(var i = 10; i < 21; i++)
{<p>Line @i</p>}
</body>
</html>
```

[运行实例 »](#)

For Each 循环

如果您使用的是集合或者数组，您会经常用到 **for each 循环**。

集合是一组相似的对象，for each 循环可以遍历集合直到完成。

下面的实例中，遍历 ASP.NET Request.ServerVariables 集合。

实例

```
<html>
```

```
<body>
<ul>
@foreach (var x in Request.ServerVariables)
{<li>@x</li>}
</ul>
</body>
</html>
```

[运行实例 »](#)

While 循环

while 循环是一个通用的循环。

while 循环以 **while** 关键字开始，后面紧跟着括号，您可以在括号里规定循环将持续多久，然后是重复执行的代码块。

while 循环通常会设定一个递增或者递减的变量用来计数。

下面的实例中，**+=** 运算符在每执行一次循环时给变量 **i** 的值加 **1**。

实例

```
<html>
<body>
@{
var i = 0;
while (i < 5)
{
i += 1;
<p>Line @i</p>
}
}
</body>
</html>
```

[运行实例 »](#)

数组

当您存储多个相似变量但又不想为每个变量都创建一个独立的变量时，可以使用数组来存储：

实例

```
@{
string[] members = {"Jani", "Hege", "Kai", "Jim"};
int i = Array.IndexOf(members, "Kai")+1;
int len = members.Length;
string x = members[2-1];
}
<html>
<body>
<h3>Members</h3>
@foreach (var person in members)
{
<p>@person</p>
}
<p>The number of names in Members are @len</p>
<p>The person at position 2 is @x</p>
<p>Kai is now in position @i</p>
</body>
</html>
```

[运行实例 »](#)



ASP.NET Razor - C# 逻辑条件

编程逻辑：根据条件执行代码。

If 条件

C# 允许根据条件执行代码。

使用 **if** 语句来判断条件。根据判断结果，if 语句返回 **true** 或者 **false**：

if 语句开始一个代码块

条件写在括号里

如果条件为真，大括号内的代码被执行

实例

```
@{var price=50;}
<html>
<body>
@if (price>30)
{
<p>The price is too high.</p>
}
</body>
</html>
```

运行实例 »

Else 条件

if 语句可以包含 **else** 条件。

else 条件定义了当条件为假时被执行的代码。

实例

```
@{var price=20;}
<html>
<body>
@if (price>30)
{
<p>The price is too high.</p>
}
else
{
<p>The price is OK.</p>
}
```

```
}  
</body>  
</html>
```

运行实例 »

注释： 在上面的实例中，如果第一个条件为真，**if** 块的代码将会被执行。**else** 条件覆盖了除 **if** 条件之外的"其他所有情况"。

Else If 条件

多个条件判断可以使用 **else if** 条件：

实例

```
@{var price=25;}  
<html>  
<body>  
  @if (price>=30)  
  {  
    <p>The price is high.</p>  
  }  
  else if (price>20 && price<30)  
  {  
    <p>The price is OK.</p>  
  }  
  else  
  {  
    <p>The price is low.</p>  
  }  
</body>  
</html>
```

运行实例 »

在上面的实例中，如果第一个条件为真，**if** 块的代码将会被执行。

如果第一个条件不为真且第二个条件为真，**else if** 块的代码将会被执行。

else if 条件的数量不受限制。

如果 **if** 和 **else if** 条件都不为真，最后的 **else** 块（不带条件）覆盖了"其他所有情况"。

Switch 条件

switch 块可以用来测试一些单独的条件：

实例

```
@{  
  var weekday=DateTime.Now.DayOfWeek;  
  var day=weekday.ToString();  
  var message="";  
}  
<html>  
<body>  
  @switch(day)  
  {  
    case "Monday":  
      message="This is the first weekday.";  
      break;  
    case "Thursday":  
      message="Only one day before weekend.";  
      break;  
    case "Friday":  
      message="Tomorrow is weekend!";  
      break;  
    default:  
      message="Today is " + day;  
      break;  
  }  
<p>@message</p>  
</body>
```

</html>

运行实例 »

测试值（**day**）是写在括号中。每个单独的测试条件都有一个以分号结束的 **case** 值和以 **break** 语句结束的任意数量的代码行。如果测试值与 **case** 值相匹配，相应的代码行被执行。

switch 块有一个默认的情况（**default:**），当所有的指定的情况都不匹配时，它覆盖了"其他所有情况"。

[☐ ASP.NET Razor C# 循环和数组](#)

[ASP.NET Razor VB 变量](#) ☐

[☐ 点我分享笔记](#)

反馈/建议

Copyright © 2013-2018 菜鸟教程 [runoob.com](#) All Rights Reserved. 备案号：闽ICP备15012807号-1

☐

[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

[☐ ASP.NET Razor C# 逻辑](#)

[ASP.NET Razor VB 循环和数组](#) ☐

ASP.NET Razor - VB 变量

变量是用来存储数据的命名实体。

变量

变量是用来存储数据的。

一个变量的名称必须以字母字符开头，并且不能包含空格或者保留字符。

一个变量可以是一个指定的类型，表示它所存储的数据类型。**string** 变量存储字符串值（"Welcome to runoob.com"），**integer** 变量存储数字值（103），**date** 变量存储日期值，等等。

变量使用 **Dim** 关键字声明，或通过使用类型（如果您想声明类型）声明，但是 **ASP.NET** 通常能自动确定数据类型。

实例

```
// Using the Dim keyword:
Dim greeting = "Welcome to runoob.com"
Dim counter = 103
Dim today = DateTime.Today

// Using data types:
Dim greeting As String = "Welcome to runoob.com"
Dim counter As Integer = 103
Dim today As DateTime = DateTime.Today
```

数据类型

下面列出了常用的数据类型：

类型	描述	实例
----	----	----

integer	整数（全数字）	103, 12, 5168
double	64 位浮点数	3.14, 3.4e38
decimal	十进制数字（高精度）	1037.196543
boolean	布尔值	true, false
string	字符串	"Hello runoob.com", "John"

运算符

运算符告诉 **ASP.NET** 在表达式中执行什么样的命令。

VB 语言支持多种运算符。下面列出了常用的运算符：

运算符	描述	实例
=	给一个变量赋值。	i=6
+	加上一个值或者一个变量。	i=5+5
-	减去一个值或者一个变量。	i=5-5
*	乘以一个值或者一个变量。	i=5*5
/	除以一个值或者一个变量。	i=5/5
+=	变量递增。	i += 1
-=	变量递减。	i -= 1
=	相等。如果值相等则返回 true 。	if i=10
<>	不等。如果值不等则返回 true 。	if <>10
<	小于。	if i<10
>	大于。	if i>10
<=	小于等于。	if i<=10
>=	大于等于。	if i>=10
&	连接字符串（一系列互相关联的事物）。	"w3" & "schools"
.	点号。分隔对象和方法。	DateTime.Hour
()	圆括号。将值进行分组。	(i+5)
()	圆括号。传递参数。	x=Add(i,5)
()	圆括号。访问数组或者集合的值。	name(3)
Not	非。真/假取反。	if Not ready
And	逻辑与。	if ready And clear
OR	逻辑或。	if ready Or clear
AndAlso	扩展的逻辑与。	if ready AndAlso clear
OrElse	扩展的逻辑或。	if ready OrElse clear

转换数据类型

从一种数据类型转换到另一种数据类型，有时候是很有用的。

最常见的例子是将字符串输入转换为另一种类型，如整数或者日期。

一般规则下，都是将用户输入看做字符串处理，即使用户输入了数字。因此数值输入必须被转换成数字，然后才能将其用于计算。

下面列出了常用的转换方法：

方法	描述	实例
AsInt() IsInt()	转换字符串为整数。	if myString.IsInt() then myInt=myString.AsInt() end if
AsFloat() IsFloat()	转换字符串为浮点数。	if myString.IsFloat() then myFloat=myString.AsFloat() end if
AsDecimal() IsDecimal()	转换字符串为十进制数。	if myString.IsDecimal() then myDec=myString.AsDecimal() end if
AsDateTime() IsDateTime()	转换字符串为 ASP.NET DateTime 类型。	myString="10/10/2012" myDate=myString.AsDateTime()
AsBool() IsBool()	转换字符串为布尔值。	myString="True" myBool=myString.AsBool()
ToString()	转换任何数据类型为字符串。	myInt=1234 myString=myInt.ToString()

反馈/建议



ASP.NET Razor - VB 循环和数组

语句在循环中会被重复执行。

For 循环

如果您需要重复执行相同的语句，您可以设定一个循环。

如果您知道要循环的次数，您可以使用 **for 循环**。这种类型的循环在向上计数或向下计数时特别有用：

实例

```
<html>
<body>
@For i=10 To 21
@<p>Line #@i</p>
Next i
</body>
</html>
```

运行实例 »

For Each 循环

如果您使用的是集合或者数组，您会经常用到 **for each** 循环。

集合是一组相似的对象，**for each** 循环可以遍历集合直到完成。

下面的实例中，遍历 ASP.NET Request.ServerVariables 集合。

实例

```
<html>
<body>
<ul>
@For Each x In Request.ServerVariables
@<li>@x</li>
Next x
</ul>
</body>
</html>
```

运行实例 »

While 循环

while 循环是一个通用的循环。

while 循环以 **while** 关键字开始，后面紧跟着括号，您可以在括号里规定循环将持续多久，然后是重复执行的代码块。

while 循环通常会设定一个递增或者递减的变量用来计数。

下面的实例中，**+=** 运算符在每执行一次循环时给变量 **i** 的值加 1。

实例

```
<html>
<body>
@Code
Dim i=0
Do While i<5
i += 1
@<p>Line #@i</p>
Loop
End Code
</body>
</html>
```

运行实例 »

数组

当您要存储多个相似变量但又不想为每个变量都创建一个独立的变量时，可以使用数组来存储：

实例

```
@Code
Dim members As String()={"Jani","Hege","Kai","Jim"}
i=Array.IndexOf(members,"Kai")+1
len=members.Length
```

```
x=members(2-1)
end Code
<html>
<body>
<h3>Members</h3>
@For Each person In members
@<p>@person</p>
Next person
<p>The number of names in Members are @len</p>
<p>The person at position 2 is @x</p>
<p>Kai is now in position @i</p>
</body>
</html>
```

运行实例 »

□ ASP.NET Razor VB 变量

ASP.NET Razor VB 逻辑 □

□ 点我分享笔记

反馈/建议

Copyright © 2013-2018 菜鸟教程 runoob.com All Rights Reserved. 备案号: 闽ICP备15012807号-1

□

首页 HTML CSS JS 本地书签

□ ASP.NET Razor VB 循环和数组

ASP.NET MVC 简介 □

ASP.NET Razor - VB 逻辑条件

编程逻辑：根据条件执行代码。

If 条件

VB 允许根据条件执行代码。

使用 **if** 语句来判断条件。根据判断结果，**if** 语句返回 **true** 或者 **false**：

if 语句开始一个代码块

条件写在 **if** 和 **then** 之间

如果条件为真，**if ... then** 和 **end if** 之间的代码被执行

实例

```
@Code
Dim price=50
End Code
<html>
<body>
@If price>30 Then
@<p>The price is too high.</p>
End If
</body>
</html>
```

[运行实例 »](#)

Else 条件

if 语句可以包含 **else** 条件。

else 条件定义了当条件为假时被执行的代码。

实例

```
@Code
Dim price=20
End Code
<html>
<body>
  @if price>30 then
  @<p>The price is too high.</p>
  Else
  @<p>The price is OK.</p>
  End If
</body>
</html>
```

[运行实例 »](#)

注释：在上面的实例中，如果第一个条件为真，if 块的代码将会被执行。**else** 条件覆盖了除 if 条件之外的"其他所有情况"。

Elseif 条件

多个条件判断可以使用 **elseif** 条件：

实例

```
@Code
Dim price=25
End Code
<html>
<body>
  @If price>=30 Then
  @<p>The price is high.</p>
  ElseIf price>20 And price<30
  @<p>The price is OK.</p>
  Else
  @<p>The price is low.</p>
  End If
</body>
</html>
```

[运行实例 »](#)

在上面的实例中，如果第一个条件为真，if 块的代码将会被执行。

如果第一个条件不为真且第二个条件为真，**elseif** 块的代码将会被执行。

elseif 条件的数量不受限制。

如果 if 和 **elseif** 条件都不为真，最后的 **else** 块（不带条件）覆盖了"其他所有情况"。

Select 条件

select 块可以用来测试一些单独的条件：

实例

```
@Code
Dim weekday=DateTime.Now.DayOfWeek
Dim day=weekday.ToString()
Dim message=""
End Code
<html>
```

```
<body>
@Select Case day
Case "Monday"
message="This is the first weekday."
Case "Thursday"
message="Only one day before weekend."
Case "Friday"
message="Tomorrow is weekend!"
Case Else
message="Today is " & day
End Select
<p>@message</p>
</body>
</html>
```

运行实例 »

"Select Case" 后面紧跟着测试值（day）。每个单独的测试条件都有一个 case 值和任意数量的代码行。如果测试值与 case 值相匹配，相应的代码行被执行。

select 块有一个默认的情况（Case Else），当所有的指定的情况都不匹配时，它覆盖了"其他所有情况"。

❏ ASP.NET Razor VB 循环和数组

ASP.NET MVC 简介 ❏

❏ 点我分享笔记

反馈/建议

Copyright © 2013-2018 菜鸟教程 runoob.com All Rights Reserved. 备案号：闽ICP备15012807号-1

❏

首页 HTML CSS JS 本地书签

❏ ASP.NET Razor VB 逻辑

ASP.NET MVC Web 应用程序 ❏

ASP.NET MVC 教程

ASP.NET 是一个使用 HTML、CSS、JavaScript 和服务器脚本创建网页和网站的开发框架。

ASP.NET 支持三种不同的开发模式：

Web Pages（Web 页面）、MVC（Model View Controller 模型-视图-控制器）、Web Forms（Web 窗体）。

本教程介绍 **MVC**。

Web Pages

MVC

Web Forms

MVC 编程模式

MVC 是三种 ASP.NET 编程模式中的一种。

MVC 是一种使用 MVC（Model View Controller 模型-视图-控制器）设计创建 Web 应用程序的模式：

Model（模型）表示应用程序核心（比如数据库记录列表）。

View（视图）显示数据（数据库记录）。

Controller（控制器）处理输入（写入数据库记录）。

MVC 模式同时提供了对 HTML、CSS 和 JavaScript 的完全控制。

MVC

Model（模型） 是应用程序中用于处理应用程序数据逻辑的部分。

通常模型对象负责在数据库中存取数据。

View（视图） 是应用程序中处理数据显示的部分。

通常视图是依据模型数据创建的。

Controller（控制器） 是应用程序中处理用户交互的部分。

通常控制器负责从视图读取数据，控制用户输入，并向模型发送数据。

MVC 分层有助于管理复杂的应用程序，因为您可以在一个时间内专门关注一个方面。例如，您可以在不依赖业务逻辑的情况下专注于视图设计。同时也让应用程序的测试更加容易。

MVC 分层同时也简化了分组开发。不同的开发人员可同时开发视图、控制器逻辑和业务逻辑。

Web Forms 对比 MVC

MVC 编程模式是对传统 ASP.NET（Web Forms）的一种轻量级的替代方案。它是轻量级的、可测试性高的框架，同时整合了所有已有的 ASP.NET 特性，比如母版页、安全性和认证。

Visual Studio Express 2012/2010

Visual Studio Express 是 Microsoft Visual Studio 的免费版本。

Visual Studio Express 是为 MVC（和 Web Forms）量身定制的开发工具。

Visual Studio Express 包含：

MVC 和 Web Forms

拖拽 Web 控件和 Web 组件

Web 服务器语言（Razor 使用 VB 或者 C#）

Web 服务器（IIS Express）

数据库服务器（SQL Server Compact）

完整的 Web 开发框架（ASP.NET）

如果您已经安装了 Visual Studio Express，您将从本教程中学到更多。

如果您想安装 Visual Studio Express，请点击下列链接中的一个：

[Visual Web Developer 2012](#)（Windows 7 或者 Windows 8）

[Visual Web Developer 2010](#)（Windows Vista 或者 XP）

☐ 在您首次安装完 Visual Studio Express 之后，您可以通过再次运行安装程序来安装补丁和服务包，只需要再次点击链接即可。

ASP.NET MVC 参考手册

在本教程的最后，我们提供了完整的 ASP.NET MVC 参考手册供您查阅。

MVC 模式定义 Web 应用程序

带有三个逻辑层：

业务层（模型逻辑）

显示层（视图逻辑）

输入控制（控制器逻辑）

☐ ASP.NET Razor VB 逻辑

ASP.NET MVC Web 应用程序 ☐

☐ 点我分享笔记

反馈/建议

[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)[□ ASP.NET MVC 简介](#)[ASP.NET MVC 文件夹 □](#)

ASP.NET MVC - Internet 应用程序

为了学习 ASP.NET MVC，我们将构建一个 Internet 应用程序。

第 1 部分：创建应用程序。

我们将构建什么

我们将构建一个支持添加、编辑、删除和列出数据库存储信息的 Internet 应用程序。

我们将做什么

Visual Web Developer 提供了构建 Web 应用程序的不同模板。

我们将使用 Visual Web Developer 来创建一个带 HTML5 标记的空的 MVC Internet 应用程序。

当这个空白的 Internet 应用程序被创建之后，我们将逐步向该应用添加代码，直到全部完成。我们将使用 C# 作为编程语言，并使用最新的 Razor 服务器代码标记。

沿着这个思路，我们将讲解这个应用程序的内容、代码和所有组件。

创建 Web 应用程序

如果您已经安装了 Visual Web Developer，请启动 Visual Web Developer 并选择 New Project 来新建项目。否则您就只能通过阅读教程来学习了。

New Project

在 New Project 对话框中：

打开 Visual C# 模板

选择模板 ASP.NET MVC 3 Web Application

设置项目名称为 MvcDemo

设置磁盘位置，比如 c:\runoob_demo

点击 OK

当 New Project 对话框打开时：

选择 Internet Application 模板

选择 Razor Engine (Razor 引擎)

选择 HTML5 Markup (HTML5 标记)

点击 OK

Visual Studio Express 将创建一个如下所示的类似项目：

Mvc Explorer

我们将在本教程的下一章中探究有关文件和文件夹的内容。

[□ ASP.NET MVC 简介](#)[ASP.NET MVC 文件夹 □](#)[□ 点我分享笔记](#)

ASP.NET MVC - 应用程序文件夹

为了学习 ASP.NET MVC，我们将构建一个 Internet 应用程序。

第 2 部分：探究应用程序文件夹。

MVC 文件夹

一个典型的 ASP.NET MVC Web 应用程序的文件夹内容如下所示：

	应用程序信息
	Properties
	References
	应用程序文件夹
	App_Data 文件夹
	Content 文件夹
	Controllers 文件夹
	Models 文件夹
	Scripts 文件夹
	Views 文件夹
	配置文件
	Global.asax
	packages.config
	Web.config

所有的 MVC 应用程序的文件夹名称都是相同的。MVC 框架是基于默认的命名。控制器写在 **Controllers** 文件夹中，视图写在 **Views** 文件夹中，模型写在 **Models** 文件夹中。您不必再应用程序代码中使用文件夹名称。

标准化的命名减少了代码量，同时有利于开发人员对 MVC 项目的理解。

下面是对每个文件夹内容的简短概述：

App_Data 文件夹

App_Data 文件夹用于存储应用程序数据。

我们将在本教程后面的章节中介绍添加 SQL 数据库到 App_Data 文件夹。

Content 文件夹

Content 文件夹用于存放静态文件，比如样式表（CSS 文件）、图标和图像。

Visual Web Developer 会自动添加一个 **themes** 文件夹到 Content 文件夹中。themes 文件夹存放 jQuery 样式和图片。在项目中，您可以删除这个 themes 文件夹。

Visual Web Developer 同时也会添加一个标准的样式表文件到项目中：即 content 文件夹中的 **Site.css** 文件。这个样式表文件是您想要改变应用程序样式时需要编辑的文件。

Content

我们将在本教程的下一章中编辑这个样式表文件（Site.css）。

Controllers 文件夹

Controllers 文件夹包含负责处理用户输入和响应的控制器类。

MVC 要求所有控制器文件的名称以 **"Controller"** 结尾。

Visual Web Developer 已经创建好一个 **Home** 控制器（用于 **Home** 页面和 **About** 页面）和一个 **Account** 控制器（用于 **Login** 页面）：

Controllers

我们将在本教程后面的章节中创建更多的控制器。

Models 文件夹

Models 文件夹包含表示应用程序模型的类。模型控制并操作应用程序的数据。

我们将在本教程后面的章节中创建模型（类）。

Views 文件夹

Views 文件夹用于存储与应用程序的显示相关的 **HTML** 文件（用户界面）。

Views 文件夹中包含每个控制器对应的一个文件夹。

在 **Views** 文件夹中，**Visual Web Developer** 已经创建了一个 **Account** 文件夹、一个 **Home** 文件夹、一个 **Shared** 文件夹。

Account 文件夹包含用于用户账号注册和登录的页面。

Home 文件夹用于存储诸如 **home** 页和 **about** 页之类的应用程序页面。

Shared 文件夹用于存储控制器间分享的视图（母版页和布局页）。

Views

我们将在本教程的下一章中编辑这些布局文件。

Scripts 文件夹

Scripts 文件夹存储应用程序的 **JavaScript** 文件。

默认情况下，**Visual Web Developer** 在这个文件夹中存放标准的 **MVC**、**Ajax** 和 **jQuery** 文件：

Scripts

注释： 名为 **"modernizr"** 的文件时用于在应用程序中支持 **HTML5** 和 **CSS3** 的 **JavaScript** 文件。

[❏ ASP.NET MVC Web 应用程序](#) [ASP.NET MVC 页面和布局 ❏](#)

[❏ 点我分享笔记](#)

反馈/建议

Copyright © 2013-2018 菜鸟教程 [runoob.com](#) All Rights Reserved. 备案号：闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

[❏ ASP.NET MVC 文件夹](#) [ASP.NET MVC 控制器 ❏](#)

ASP.NET MVC - 样式和布局

为了学习 **ASP.NET MVC**，我们将构建一个 **Internet** 应用程序。

第 3 部分：添加样式和统一的外观（布局）。

添加布局

文件 `_Layout.cshtml` 表示应用程序中每个页面的布局。它位于 `Views` 文件夹中的 `Shared` 文件夹。

打开文件 `_Layout.cshtml`，把内容替换成：

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>@ViewBag.Title</title>
<link href="@Url.Content("~/Content/Site.css")" rel="stylesheet" type="text/css" />
<script src="@Url.Content("~/Scripts/jquery-1.5.1.min.js")"></script>
<script src="@Url.Content("~/Scripts/modernizr-1.7.min.js")"></script>
</head>
<body>
<ul id="menu">
<li>@Html.ActionLink("Home", "Index", "Home")</li>
<li>@Html.ActionLink("Movies", "Index", "Movies")</li>
<li>@Html.ActionLink("About", "About", "Home")</li>
</ul>
<section id="main">
@RenderBody()
<p>Copyright RUNOOB 2012. All Rights Reserved.</p>
</section>
</body>
</html>
```

HTML 帮助器

在上面的代码中，HTML 帮助器用于修改 HTML 输出：

`@Url.Content()` - URL 内容将在此处插入。

`@Html.ActionLink()` - HTML 链接将在此处插入。

在本教程后面的章节中，您将学到更多关于 HTML 帮助器的知识。

Razor 语法

在上面的代码中，红色标记的代码是使用 Razor 标记的 `C#`。

`@ViewBag.Title` - 页面标题将在此处插入。

`@RenderBody()` - 页面内容将在此处呈现。

您可以在我们的 [Razor 教程](#)中学习关于 `C#` 和 VB（Visual Basic）的 Razor 标记的知识。

添加样式

应用程序的样式表是 `Site.css`，位于 `Content` 文件夹中。

打开文件 `Site.css`，把内容替换成：

```
body
{
font: "Trebuchet MS", Verdana, sans-serif;
background-color: #5c87b2;
color: #696969;
}
h1
{
border-bottom: 3px solid #cc9900;
font: Georgia, serif;
color: #996600;
}
#main
{
padding: 20px;
background-color: #ffffff;
border-radius: 0 4px 4px 4px;
}
a
{
color: #034af3;
}
/* Menu Styles -----*/
ul#menu
{
padding: 0px;
position: relative;
margin: 0;
}
ul#menu li
```

```
{
display: inline;
}
ul#menu li a
{
background-color: #e8eef4;
padding: 10px 20px;
text-decoration: none;
line-height: 2.8em;
/*CSS3 properties*/
border-radius: 4px 4px 0 0;
}
ul#menu li a:hover
{
background-color: #ffffff;
}
/* Forms Styles -----*/
fieldset
{
padding-left: 12px;
}
fieldset label
{
display: block;
padding: 4px;
}
input[type="text"], input[type="password"]
{
width: 300px;
}
input[type="submit"]
{
padding: 4px;
}
/* Data Styles -----*/
table.data
{
background-color:#ffffff;
border:1px solid #c3c3c3;
border-collapse:collapse;
width:100%;
}
table.data th
{
background-color:#e8eef4;
border:1px solid #c3c3c3;
padding:3px;
}
table.data td
{
border:1px solid #c3c3c3;
padding:3px;
}
```

_ViewStart 文件

Shared 文件夹（位于 Views 文件夹内）中的 _ViewStart 文件包含如下内容：

```
@{Layout = "~/Views/Shared/_Layout.cshtml";}
```

这段代码被自动添加到由应用程序显示的所有视图。

如果您删除了这个文件，则必须向所有视图中添加这行代码。

在本教程后面的章节中，您将学到更多关于视图的知识。

[□ ASP.NET MVC 文件夹](#)

ASP.NET MVC 控制器 [□](#)

[□ 点我分享笔记](#)

[反馈/建议](#)

[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)[ASP.NET MVC 页面和布局](#)[ASP.NET MVC 视图](#)

ASP.NET MVC - 控制器

为了学习 ASP.NET MVC, 我们将构建一个 Internet 应用程序。

第 4 部分: 添加控制器。

Controllers 文件夹

Controllers 文件夹包含负责处理用户输入和响应的控制类。

MVC 要求所有控制器文件的名称以 "Controller" 结尾。

在我们的实例中, Visual Web Developer 已经创建好了以下文件: **HomeController.cs** (用于 Home 页面和 About 页面) 和 **AccountController.cs** (用于登录页面):

Controllers

Web 服务器通常会将进入的 URL 请求直接映射到服务器上的磁盘文件。例如: URL 请求 "http://www.w3cschool.cc/index.php" 将直接映射到服务器根目录上的文件 "index.php"。

MVC 框架的映射方式有所不同。MVC 将 URL 映射到方法。这些方法在类中被称为"控制器"。

控制器负责处理进入的请求, 处理输入, 保存数据, 并把响应发送回客户端。

Home 控制器

在我们应用程序中的控制器文件 **HomeController.cs**, 定义了两个控件 **Index** 和 **About**。

把 **HomeController.cs** 文件的内容替换成:

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

namespace MvcDemo.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {return View();}

        public ActionResult About()
        {return View();}
    }
}
```

Controller 视图

Views 文件夹中的文件 **Index.cshtml** 和 **About.cshtml** 定义了控制器中的 ActionResult 视图 Index() 和 About()。

[ASP.NET MVC 页面和布局](#)[ASP.NET MVC 视图](#)[点我分享笔记](#)

[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)[ASP.NET MVC 控制器](#)[ASP.NET MVC 数据库](#)

ASP.NET MVC - 视图

为了学习 ASP.NET MVC，我们将构建一个 Internet 应用程序。

第 5 部分：添加用于显示应用程序的视图。

Views 文件夹

Views 文件夹存储的是与应用程序显示（用户界面）相关的文件（HTML 文件）。根据所采用的语言内容，这些文件可能扩展名可能是 **html**、**asp**、**aspx**、**cshtml** 和 **vbhtml**。

Views 文件夹中包含每个控制器对应的一个文件夹。

在 **Views** 文件夹中，**Visual Web Developer** 已经创建了一个 **Account** 文件夹、一个 **Home** 文件夹、一个 **Shared** 文件夹。

Account 文件夹包含用于用户账号注册和登录的页面。

Home 文件夹用于存储诸如 **home** 页和 **about** 页之类的应用程序页面。

Shared 文件夹用于存储控制器间分享的视图（母版页和布局页）。

Views

ASP.NET 文件类型

在 **Views** 文件夹中可以看到以下 **HTML** 文件类型：

文件类型	扩展名
纯 HTML	.htm or .html
经典 ASP	.asp
经典 ASP.NET	.aspx
ASP.NET Razor C#	.cshtml
ASP.NET Razor VB	.vbhtml

Index 文件

文件 **Index.cshtml** 表示应用程序的 **Home** 页面。它是应用程序的默认文件（首页文件）。

在文件中写入以下内容：

```
@{ViewBag.Title = "Home Page";}

<h1>Welcome to runoob.com</h1>

<p>Put Home Page content here</p>
```

About 文件

文件 **About.cshtml** 表示应用程序的 **About** 页面。

在文件中写入以下内容：

```
@{ViewBag.Title = "About Us";}

<h1>About Us</h1>

<p>Put About Us content here</p>
```

运行应用程序

选择 **Debug**，从 **Visual Web Developer** 菜单中启动调试 **Start Debugging**（或者按 **F5**）。

您的应用程序将显示如下：

MVC Application

点击 **"Home"** 标签页和 **"About"** 标签页，看看它是如何运作的。

祝贺您

祝贺您。您已经创建好了您的第一个 **MVC** 应用程序。

注释：您暂时还不能点击 **"Movies"** 标签页。我们将在本教程的后面章节中为 **"Movies"** 标签页添加代码。

☐ ASP.NET MVC 控制器

ASP.NET MVC 数据库 ☐

☐ 点我分享笔记

反馈/建议

Copyright © 2013-2018 菜鸟教程 [runoob.com](#) All Rights Reserved. 备案号：闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

☐ ASP.NET MVC 视图

ASP.NET MVC 模型 ☐

ASP.NET MVC - SQL 数据库

为了学习 **ASP.NET MVC**，我们将构建一个 **Internet** 应用程序。

第 **6** 部分：添加数据库。

创建数据库

Visual Web Developer 带有一名 **SQL Server Compact** 免费的 **SQL** 数据库。

本教程所需的这个数据库可以通过以下几个简单的步骤来创建：

右击 **Solution Explorer** 窗口中的 **App_Data** 文件夹

选择 **Add, New Item**

选择 **SQL Server Compact Local Database ***

将数据库命名为 **Movies.sdf**

点击 **Add** 按钮

* 如果选项中没有 **SQL Server Compact Local Database**，则说明您尚未在计算机上安装 **SQL Server Compact**。请通过以下链接进行安装：[SQL Server Compact](#)

Visual Web Developer 会自动在 **App_Data** 文件夹中创建该数据库。

注释：在本教程中，需要您掌握一些关于 **SQL** 数据库的基础知识。如果您想先学习这个主题，请访问我们的 [SQL 教程](#)。

添加数据库表

双击 **App_Data** 文件夹中的 **Movies.sdf** 文件，将打开 **Database Explorer** 窗口。

如需在数据库中创建一个新的表，请右击 **Tables** 文件夹，然后选择 **Create Table**。

创建如下的列：

列	类型	是否允许为 Null
ID	int (primary key)	No
Title	nvarchar(100)	No
Director	nvarchar(100)	No
Date	datetime	No

对列的解释：

ID 是用于标识表中每条记录的整数（全数字）。

Title 是 100 个字符长度的文本列，用于存储影片的名称。

Director 是 100 个字符长度的文本列，用于存储导演的名字。

Date 是日期列，用于存储影片的发布日期。

在创建好上述列之后，您必须将 **ID** 列设置为表的主键（记录标识符）。要做到这点，请点击列名（ID），并选择 **Primary Key**。在 **Column Properties** 窗口中，设置 **Identity** 属性为 **True**：

DB Explorer

当您创建好表列后，保存表并命名为 **MovieDBs**。

注释：

我们特意把表命名为 "MovieDBs"（以 **s** 结尾）。在下一章中，您将看到用于数据模型的 "MovieDB"。这看起来有点奇怪，不过这种命名惯例能确保控制器连接上数据库表，您必须这么使用。

添加数据库记录

您可以使用 **Visual Web Developer** 向 **movie** 数据库中添加一些测试记录。

双击 **App_Data** 文件夹中的 **Movies.sdf** 文件。

右击 **Database Explorer** 窗口中的 **MovieDBs** 表，并选择 **Show Table Data**。

添加一些记录：

ID	Title	Director	Date
1	Psycho	Alfred Hitchcock	01.01.1960
2	La Dolce Vita	Federico Fellini	01.01.1960

注释： ID 列会自动更新，您可以不用编辑它。

添加连接字符串

向您的 **Web.config** 文件中的 **<connectionStrings>** 元素添加如下元素：

```
<add name="MovieDBContext"
connectionString="Data Source=|DataDirectory|Movies.sdf"
providerName="System.Data.SqlClient" />
```

[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)[ASP.NET MVC 数据库](#)[ASP.NET MVC 安全](#)

ASP.NET MVC - 模型

为了学习 ASP.NET MVC, 我们将构建一个 Internet 应用程序。

第 7 部分: 添加数据模型。

MVC 模型

MVC 模型包含了除纯视图和控制器逻辑以外的其他所有应用程序逻辑(业务逻辑、验证逻辑、数据访问逻辑)。

通过 MVC, 模型可以控制并操作应用程序数据。

Models 文件夹

Models 文件夹包含表示应用程序模型的类。

Visual Web Developer 自动创建一个 AccountModels.cs 文件, 该文件包含用于应用程序安全的模型。

AccountModels 包含 LogOnModel、ChangePasswordModel 和 RegisterModel。

添加数据库模型

本教程所需的数据库模型可以通过以下几个简单的步骤来创建:

在 **Solution Explorer** 窗口中, 右击 **Models** 文件夹, 并选择 **Add** 和 **Class**。

将类命名为 **MovieDB.cs**, 然后点击 **Add**。

编辑这个类:

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

namespace MvcDemo.Models
{
    public class MovieDB
    {
        public int ID { get; set; }
        public string Title { get; set; }
        public string Director { get; set; }
        public DateTime Date { get; set; }
    }

    public class MovieDBContext : DbContext
    {
        public DbSet<MovieDB> Movies { get; set; }
    }
}
```

注释:

我们特意把模型命名为 "MovieDB"。在上一章中, 您已经看到用于数据库表的 "MovieDBs" (以 s 结尾)。这看起来有点奇怪, 不过这种命名惯例能确保模型连接上数据库表, 您必须这么使用。

添加数据库控制器

本教程所需的数据库控制器可以通过以下几个简单的步骤来创建：

- 重建您的项目：选择 **Debug**，然后从菜单中选择 **Build MvcDemo**。
- 在 **Solution Explorer**（解决方案资源管理器）中，右击 **Controllers** 文件夹，选择 **Add** 和 **Controller**。
- 设置控制器名称为 **MoviesController**。
- 选择模板：**Controller with read/write actions and views, using Entity Framework**
- 选择模型类：**MovieDB (MvcDemo.Models)**
- 选择 data context 类：**MovieDbContext (MvcDemo.Models)**
- 选择视图 **Razor (CSHTML)**
- 点击 **Add**

Visual Web Developer 将创建以下文件：

- Controllers** 文件夹中的 **MoviesController.cs** 文件
- Views** 文件夹中的 **Movies** 文件夹

添加数据库视图

在 **Movies** 文件夹中，会自动创建以下文件：

- Create.cshtml
- Delete.cshtml
- Details.cshtml
- Edit.cshtml
- Index.cshtml

祝贺您

祝贺您。您已经向应用程序添加了您的第一个 MVC 数据模型。
现在您可以点击 "Movies" 标签页了。



ASP.NET MVC - 安全

为了学习 ASP.NET MVC，我们将构建一个 Internet 应用程序。

MVC 应用程序安全

Models 文件夹包含表示应用程序模型的类。

Visual Web Developer 自动创建 **AccountModels.cs** 文件，该文件包含用于应用程序认证的模型。

AccountModels 包含 **LogOnModel**、**ChangePasswordModel** 和 **RegisterModel**：

Model

Change Password 模型

```
public class ChangePasswordModel
{

    [Required]
    [DataType(DataType.Password)]
    [Display(Name = "Current password")]
    public string OldPassword { get; set; }

    [Required]
    [StringLength(100, ErrorMessage = "The {0} must be at least {2} characters long.", MinimumLength = 6)]
    [DataType(DataType.Password)]
    [Display(Name = "New password")]
    public string NewPassword { get; set; }

    [DataType(DataType.Password)]
    [Display(Name = "Confirm new password")]
    [Compare("NewPassword", ErrorMessage = "The new password and confirmation password do not match.")]
    public string ConfirmPassword { get; set; }

}
```

Logon 模型

```
public class LogOnModel
{

    [Required]
    [Display(Name = "User name")]
    public string UserName { get; set; }

    [Required]
    [DataType(DataType.Password)]
    [Display(Name = "Password")]
    public string Password { get; set; }

    [Display(Name = "Remember me?")]
    public bool RememberMe { get; set; }

}
```

Register 模型

```
public class RegisterModel
{

    [Required]
    [Display(Name = "User name")]
    public string UserName { get; set; }

    [Required]
    [DataType(DataType.EmailAddress)]
    [Display(Name = "Email address")]
    public string Email { get; set; }

    [Required]
    [StringLength(100, ErrorMessage = "The {0} must be at least {2} characters long.", MinimumLength = 6)]
    [DataType(DataType.Password)]
    [Display(Name = "Password")]
    public string Password { get; set; }

    [DataType(DataType.Password)]
    [Display(Name = "Confirm password")]
    [Compare("Password", ErrorMessage = "The password and confirmation password do not match.")]
}
```

```
public string ConfirmPassword { get; set; }

}
```

[☐ ASP.NET MVC 模型](#)

[ASP.NET MVC HTML 帮助器](#) ☐

[☐ 点我分享笔记](#)

[反馈/建议](#)

Copyright © 2013-2018 菜鸟教程 [runoob.com](#) All Rights Reserved. 备案号: 闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

[☐ ASP.NET MVC 安全](#)

[ASP.NET MVC – 发布](#) ☐

ASP.NET MVC - HTML 帮助器

HTML 帮助器用于修改 HTML 输出。

HTML 帮助器

通过 MVC，HTML 帮助器类似于传统的 ASP.NET Web Form 控件。

就像 ASP.NET 中的 Web Form 控件，HTML 帮助器用于修改 HTML。但是 HTML 帮助器是更轻量级的。与 Web Form 控件不同，HTML 帮助器没有事件模型和视图状态。

在大多数情况下，HTML 帮助器仅仅是一个返回字符串的方法。

通过 MVC，您可以创建您自己的帮助器，或者直接使用内建的 HTML 帮助器。

标准的 HTML 帮助器

MVC 包含了大多数常用的 HTML 元素类型的标准帮助器，比如 HTML 链接和 HTML 表单元素。

HTML 链接

呈现 HTML 链接的最简单的方法是使用 `Html.ActionLink()` 帮助器。

通过 MVC，`Html.ActionLink()` 不连接到视图。它创建一个连接到控制器操作。

Razor 语法:

```
@Html.ActionLink("About this Website", "About")
```

ASP 语法:

```
<%=Html.ActionLink("About this Website", "About")%>
```

第一个参数是链接文本，第二个参数是控制器操作的名称。

上面的 `Html.ActionLink()` 帮助器，输出以下的 HTML:

```
<a href="/Home/About">About this Website</a>
```

`Html.ActionLink()` 帮助器的一些属性:

属性	描述
<code>.linkText</code>	URL 文本（标签），定位点元素的内部文本。

.actionName	操作（action）的名称。
.routeValues	传递给操作（action）的值，是一个包含路由参数的对象。
.controllerName	控制器的名称。
.htmlAttributes	URL 的属性设置，是一个包含要为该元素设置的 HTML 特性的对象。
.protocol	URL 协议，如 "http" 或 "https"。
.hostname	URL 的主机名。
.fragment	URL 片段名称（定位点名称）。

注释：您可以向控制器操作传递值。例如，您可以向数据库 **Edit** 操作传递数据库记录的 **id**：

Razor 语法 C#:

```
@Html.ActionLink("Edit Record", "Edit", new {Id=3})
```

Razor 语法 VB:

```
@Html.ActionLink("Edit Record", "Edit", New With{.Id=3})
```

上面的 **Html.ActionLink()** 帮助器，输出以下的 **HTML**：

```
<a href="/Home/Edit/3">Edit Record</a>
```

HTML 表单元素

以下 **HTML** 帮助器可用于呈现（修改和输出）**HTML** 表单元素：

BeginForm()
EndForm()
TextArea()
TextBox()
CheckBox()
RadioButton()
ListBox()
DropDownList()
Hidden()
Password()

ASP.NET 语法 C#:

```
<%= Html.ValidationSummary("Create was unsuccessful. Please correct the errors and try again.") %>
<% using (Html.BeginForm()) {%>
<p>
<label for="FirstName">First Name:</label>
<%= Html.TextBox("FirstName") %>
<%= Html.ValidationMessage("FirstName", "") %>
</p>
<p>
<label for="LastName">Last Name:</label>
<%= Html.TextBox("LastName") %>
<%= Html.ValidationMessage("LastName", "") %>
</p>
<p>
<label for="Password">Password:</label>
<%= Html.Password("Password") %>
<%= Html.ValidationMessage("Password", "") %>
</p>
<p>
<label for="Password">Confirm Password:</label>
<%= Html.Password("ConfirmPassword") %>
<%= Html.ValidationMessage("ConfirmPassword", "") %>
</p>
<p>
<label for="Profile">Profile:</label>
```

```
<%= Html.TextArea("Profile", new {cols=60, rows=10})%>
</p>
<p>
<%= Html.CheckBox("ReceiveNewsletter") %>
<label for="ReceiveNewsletter" style="display:inline">Receive Newsletter?</label>
</p>
<p>
<input type="submit" value="Register" />
</p>
<%}%>
```

☐ ASP.NET MVC 安全

ASP.NET MVC – 发布 ☐

☐ 点我分享笔记

反馈/建议

Copyright © 2013-2018 菜鸟教程 runoob.com All Rights Reserved. 备案号：闽ICP备15012807号-1



首页 HTML CSS JS 本地书签

☐ ASP.NET MVC HTML 帮助器

ASP.NET MVC 参考手册 ☐

ASP.NET MVC - 发布网站

学习如何在不使用 Visual Web Developer 的情况下发布 MVC 应用程序。

在不使用 Visual Web Developer 的情况下发布您的应用程序

通过在 WebMatrix、Visual Web Developer 或 Visual Studio 中使用发布命令，可以发布一个 ASP.NET MVC 应用程序到远程服务器上。

此功能会复制所有您的应用程序文件、控制器、模型、图像以及用于 MVC、Web Pages、Razor、Helpers、SQL Server Compact（如果使用数据库）所有必需的 DLL 文件。

有时您不希望使用这些选项。或许您的主机提供商仅支持 FTP？或许您的网站基于经典 ASP？或许您希望亲自拷贝这些文件？又或许您希望使用 Front Page、Expression Web 等其他一些发布软件？

您会遇到问题吗？是的，会的。但是您有办法解决它。

要执行网站复制，您必须知道如何引用正确的文件，哪些 DLL 文件需要复制，并在何处存储它们。

请按照下列步骤操作：

1. 使用最新版本的 ASP.NET

在您继续操作之前，请确保您的主机运行的是最新版的 ASP.NET（4.0 或者 4.5）。

2. 复制 Web 文件夹

从您的开发计算机上复制您的网站（所有文件夹和内容）到远程主机（服务器）上的应用程序文件夹中。

如果您的 App_Data 文件夹中包含测试数据，请不要复制这个 App_Data 文件夹（详见下面的第 5 点）。

3. 复制 DLL 文件

在远程服务器上的应用程序根目录中创建 bin 文件夹。（如果您已经安装 Helpers，则 bin 文件夹已经存在）

复制下列文件夹中的所有文件：

C:\Program Files (x86)\Microsoft ASP.NET\ASP.NET Web Pages\v1.0\Assemblies

到您的远程服务器上的应用程序的 bin 文件夹中。

4. 复制 SQL Server Compact DLL 文件

如果您的应用程序使用了 SQL Server Compact 数据库（在 App_Data 文件夹中的一个 .sdf 文件），那么您必须复制 SQL Server Compact DLL 文件：

复制下列文件夹中的所有文件：

C:\Program Files (x86)\Microsoft SQL Server Compact Edition\v4.0\Private

到您的远程服务器上的应用程序的 bin 文件夹中。

创建（或者编辑）应用程序的 Web.config 文件：

实例 C#

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
<system.data>
<DbProviderFactories>
<remove invariant="System.Data.SqlServerCe.4.0" />

<add invariant="System.Data.SqlServerCe.4.0"
name="Microsoft SQL Server Compact 4.0"
description=".NET Framework Data Provider for Microsoft SQL Server Compact" type="System.Data.SqlServerCe.SqlCePro
viderFactory, System.Data.SqlServerCe, Version=4.0.0.1,Culture=neutral, PublicKeyToken=89845dcd8080cc91" />

</DbProviderFactories>
</system.data>
</configuration>
```

5. 复制 SQL Server Compact 数据

您的 App_Data 文件夹中有没有包含测试数据的 .sdf 文件？

您是否希望发布您的测试数据到远程服务器上？

大多数时候一般是不希望。

如果您一定要复制 SQL 数据文件（.sdf 文件），那么您应该删除数据库中的所有数据，然后从您的开发计算机上复制一个空的 .sdf 文件到服务器上。

就是这样。**GOOD LUCK!**

ASP.NET MVC - 参考手册

类

类	描述
AcceptVerbsAttribute	表示一个特性，该特性指定操作方法将响应的 HTTP 谓词。
ActionDescriptor	提供有关操作方法的信息，比如操作方法的名称、控制器、参数、特性和筛选器。
ActionExecutedContext	提供 ActionFilterAttribute 类的 ActionExecuted 方法的上下文。
ActionExecutingContext	提供 ActionFilterAttribute 类的 ActionExecuting 方法的上下文。
ActionFilterAttribute	表示筛选器特性的基类。
ActionMethodSelectorAttribute	表示一个用于影响操作方法选择的特性。
ActionNameAttribute	表示一个用于操作的名称的特性。
ActionNameSelectorAttribute	表示一个可影响操作方法选择的特性。
ActionResult	封装一个操作方法的结果并用于代表该操作方法执行框架级操作。
AdditionalMetadataAttribute	提供一个类，该类实现 IMetadataAware 接口以支持其他元数据。
AjaxHelper	表示支持在视图中呈现 AJAX 方案中的 HTML 。
AjaxHelper(TModel)	表示支持在强类型视图中呈现 AJAX 方案中的 HTML 。
AjaxRequestExtensions	表示一个类，该类对 HttpRequestBase 类进行了扩展，在其中添加了确定 HTTP 请求是否为 AJAX 请求的功能。
AllowHtmlAttribute	通过跳过属性的请求验证，允许请求在模型绑定过程中包含 HTML 标记。（强烈建议应用程序显式检查所有禁用请求验证的模型，以防止脚本攻击。）
AreaRegistration	提供在一个 ASP.NET MVC 应用程序内注册一个或多个区域的方式。
AreaRegistrationContext	对在 ASP.NET MVC 应用程序内注册某个区域时所需的信息进行封装。
AssociatedMetadataProvider	提供用于实现元数据提供程序的抽象类。
AssociatedValidatorProvider	为用于实现验证提供程序的类提供抽象类。
AsyncController	为异步控制器提供基类。
AsyncTimeoutAttribute	表示一个特性，该特性用于设置异步方法的超时值（以毫秒为单位）。
AuthorizationContext	对使用 AuthorizeAttribute 特性时所需的信息进行封装。
AuthorizeAttribute	表示一个特性，该特性用于限制调用方对操作方法的访问。
BindAttribute	表示一个特性，该特性用于提供有关应如何进行模型绑定到参数的详细信息。
BuildManagerCompiledView	表示在视图引擎呈现视图之前由 BuildManager 类编译的视图的基类。
BuildManagerViewEngine	为视图引擎提供基类。
ByteArrayModelBinder	映射浏览器请求到字节数组。
ChildActionOnlyAttribute	表示一个特性，该特性用于指示操作方法只应作为子操作进行调用。

ChildActionValueProvider	表示子操作中的值的值提供程序。
ChildActionValueProviderFactory	表示用于为子操作创建值提供程序对象的工厂。
ClientDataTypeModelValidatorProvider	返回客户端数据类型模型验证程序。
CompareAttribute	提供用于比较某个模型的两个属性的特性。
ContentResult	表示用户定义的内容类型，该类型是操作方法的结果。
Controller	提供用于响应对 ASP.NET MVC 网站所进行的 HTTP 请求的方法。
ControllerActionInvoker	表示一个类，该类负责调用控制器的操作方法。
ControllerBase	表示所有 MVC 控制器的基类。
ControllerBuilder	表示一个类，该类负责动态生成控制器。
ControllerContext	封装有关与指定的 RouteBase 和 ControllerBase 实例匹配的 HTTP 请求的信息。
ControllerDescriptor	封装描述控制器的信息，比如控制器的名称、类型和操作。
ControllerInstanceFilterProvider	将控制器添加到 FilterProviderCollection 实例。
CustomModelBinderAttribute	表示一个调用自定义模型联编程序的特性。
DataAnnotationsModelMetadata	为数据模型的公共元数据、DataAnnotationsModelMetadataProvider 类和 DataAnnotationsModelValidator 类提供容器。
DataAnnotationsModelMetadataProvider	实现 ASP.NET MVC 的默认模型元数据提供程序。
DataAnnotationsModelValidator	提供模型验证程序。
DataAnnotationsModelValidator(TAttribute)	为指定的验证类型提供模型验证程序。
DataAnnotationsModelValidatorProvider	实现 ASP.NET MVC 的默认验证提供程序。
DataErrorInfoModelValidatorProvider	为错误信息模型验证程序提供容器。
DefaultControllerFactory	表示默认情况下已注册的控制器工厂。
DefaultModelBinder	映射浏览器请求到数据对象。该类提供模型联编程序的具体实现。
DefaultViewLocationCache	表示视图位置的内存缓存。
DependencyResolver	为实现 IDependencyResolver 或公共服务定位器 IServiceLocator 接口的依赖关系解析程序提供一个注册点。
DependencyResolverExtensions	提供 GetService 和 GetServices 的类型安全实现。
DictionaryValueProvider(TValue)	表示值提供程序的基类，这些值提供程序的值来自实现 IDictionary(TKey, TValue) 接口的集合。
EmptyModelMetadataProvider	为不需要元数据的数据模型提供空的元数据提供程序。
EmptyModelValidatorProvider	为不需要验证程序的模型提供空的验证提供程序。
EmptyResult	表示一个不执行任何操作的结果，比如一个不返回任何内容的控制器操作方法。
ExceptionContext	P提供使用 HandleErrorAttribute 类的上下文。
ExpressionHelper	提供用于从表达式中获取模型名称的帮助器类。

FieldValidationMetadata	为客户端字段验证元数据提供容器。
FileContentResult	将二进制文件的内容发送到响应。
FilePathResult	将文件的内容发送到响应。
FileResult	表示一个用于将二进制文件内容发送到响应的基类。
FileStreamResult	使用 Stream 实例将二进制内容发送到响应。
Filter	表示一个元数据类型，它包含对一个或多个筛选器接口的实现、筛选器顺序和筛选器范围的引用。
FilterAttribute	表示操作和结果筛选器特性的基类。
FilterAttributeFilterProvider	定义筛选器特性的筛选器提供程序。
FilterInfo	封装有关可用的操作筛选器的信息。
FilterProviderCollection	表示应用程序的筛选器提供程序的集合。
FilterProviders	为筛选器提供一个注册点。
FormCollection	包含应用程序的表单值提供程序。
FormContext	对验证和处理 HTML 表单中的输入数据所需的信息进行封装。
FormValueProvider	表示 NameValueCollection 对象中包含的表单值的值提供程序。
FormValueProviderFactory	表示一个类，该类负责创建表单值提供程序对象的新实例。
GlobalFilterCollection	表示一个包含所有全局筛选器的类。
GlobalFilters	表示全局筛选器集合。
HandleErrorAttribute	表示一个特性，该特性用于处理由操作方法引发的异常。
HandleErrorInfo	封装有关处理由操作方法引发的错误的信息。
HiddenInputAttribute	表示一个特性，该特性用于指示是否应将属性值或字段值呈现为隐藏的 input 元素。
HtmlHelper	表示支持在视图中呈现 HTML 控件。
HtmlHelper(TModel)	表示支持在强类型视图中呈现 HTML 控件。
HttpDeleteAttribute	表示一个特性，该特性用于限制操作方法，以便该方法仅处理 HTTP DELETE 请求。
HttpFileCollectionValueProvider	表示要用于来自 HTTP 文件集合的值的值提供程序。
HttpFileCollectionValueProviderFactory	表示一个类，该类负责创建 HTTP 文件集合值提供程序对象的新实例。
HttpGetAttribute	表示一个特性，该特性用于限制操作方法，以便该方法仅处理 HTTP GET 请求。
HttpNotFoundResult	定义一个用于指示未找到所请求资源的对象。
HttpPostAttribute	表示一个特性，该特性用于限制操作方法，以便该方法仅处理 HTTP POST 请求。
HttpPostedFileBaseModelBinder	将模型绑定到已发布的文件。
HttpPutAttribute	表示一个特性，该特性用于限制操作方法，以便该方法仅处理 HTTP PUT 请求。
HttpRequestExtensions	扩展 HttpRequestBase 类，该类包含客户端在 Web 请求中发送的 HTTP 值。

HttpStatusCodeResult	提供一种用于返回带特定 HTTP 响应状态代码和说明的操作结果的方法。
HttpUnauthorizedResult	表示未经授权的 HTTP 请求的结果。
JavaScriptResult	将 JavaScript 内容发送到响应。
JsonResult	表示一个类，该类用于将 JSON 格式的内容发送到响应。
JsonValueProviderFactory	启用操作方法以发送和接收 JSON 格式的文本，并将 JSON 文本以模型绑定方式传递给操作方法的参数。
LinqBinaryModelBinder	映射浏览器请求到 LINQ Binary 对象。
ModelBinderAttribute	表示一个特性，该特性用于将模型类型关联到模型-生成器类型。
ModelBinderDictionary	表示一个类，该类包含应用程序的所有模型联编程序（按联编程序类型列出）。
ModelBinderProviderCollection	为模型联编程序提供程序提供一个容器。
ModelBinderProviders	为模型联编程序提供程序提供一个容器。
ModelBinders	提供对应用程序的模型联编程序的全局访问。
ModelBindingContext	提供运行模型联编程序的上下文。
ModelClientValidationEqualToRule	为发送到浏览器的相等验证规则提供一个容器。
ModelClientValidationRangeRule	为发送到浏览器的范围验证规则提供一个容器。
ModelClientValidationRegexRule	为发送到浏览器的正则表达式客户端验证规则提供一个容器。
ModelClientValidationRemoteRule	为发送到浏览器的远程验证规则提供一个容器。
ModelClientValidationRequiredRule	为必填字段的客户端验证提供一个容器。
ModelClientValidationRule	为发送到浏览器的客户端验证规则提供一个基类容器。
ModelClientValidationStringLengthRule	为发送到浏览器的字符串长度验证规则提供一个容器。
ModelError	表示在模型绑定期间发生的错误。
ModelErrorCollection	ModelError 实例的集合。
ModelMetadata	为数据模型的公共元数据、 ModelMetadataProvider 类和 ModelValidator 类提供容器。
ModelMetadataProvider	为自定义元数据提供程序提供抽象基类。
ModelMetadataProviders	为当前的 ModelMetadataProvider 实例提供容器。
ModelState	将模型绑定的状态封装到操作方法参数的一个属性或操作方法参数本身。
ModelStateDictionary	表示将已发表表单绑定到操作方法（其中包括验证信息）的尝试的状态。
ModelValidationResult	为验证结果提供容器。
ModelValidator	提供用于实现验证逻辑的基类。
ModelValidatorProvider	为模型提供验证程序的列表。
ModelValidatorProviderCollection	为验证提供程序的列表提供一个容器。
ModelValidatorProviders	为当前验证提供程序提供容器。

MultiSelectList	表示一个项列表，用户可从该列表中选择多个项。
MvcFilter	在派生类中实现时，提供一个元数据类型，它包含对一个或多个筛选器接口的实现、筛选器顺序和筛选器范围的引用。
MvcHandler	选择将处理 HTTP 请求的控制器。
MvcHtmlString	表示不应再次进行编码的 HTML 编码的字符串。
MvcHttpHandler	验证并处理 HTTP 请求。
MvcRouteHandler	创建一个实现 IHttpHandler 接口的对象并向该对象传递请求上下文。
MvcWebRazorHostFactory	创建 MvcWebPageRazorHost 文件的实例。
NameValueCollectionExtensions	扩展 NameValueCollection 对象，以便能够将集合复制到指定字典。
NameValueCollectionValueProvider	表示值提供程序的基类，这些值提供程序的值来自 NameValueCollection 对象。
NoAsyncTimeoutAttribute	为 AsyncTimeoutAttribute 特性提供便利包装。
NonActionAttribute	表示一个特性，该特性用于指示控制器方法不是操作方法。
OutputCacheAttribute	表示一个特性，该特性用于标记将缓存其输出的操作方法。
ParameterBindingInfo	封装与将操作方法参数绑定到数据模型相关的信息。
ParameterDescriptor	包含描述参数的信息。
PartialViewResult	表示一个用于将部分视图发送到响应的基类。
PreApplicationStartCode	为 ASP.NET Razor 应用程序预启动代码提供注册点。
QueryStringValueProvider	表示 NameValueCollection 对象中包含的查询字符串的值提供程序。
QueryStringValueProviderFactory	表示一个类，该类负责创建查询字符串值提供程序对象的新实例。
RangeAttributeAdapter	提供 RangeAttribute 特性的适配器。
RazorView	表示用于创建具有 Razor 语法的视图的类。
RazorViewEngine	表示一个用于呈现使用 ASP.NET Razor 语法的 Web 页面的视图引擎。
RedirectResult	通过重定向到指定的 URI 来控制对应用程序操作的处理。
RedirectToRouteResult	表示使用指定的路由值字典来执行重定向的结果。
ReflectedActionDescriptor	包含描述反射的操作方法的信息。
ReflectedControllerDescriptor	包含描述反射的控制器的信息。
ReflectedParameterDescriptor	包含描述反射的操作方法参数的信息。
RegularExpressionAttributeAdapter	提供 RegularExpressionAttribute 特性的适配器。
RemoteAttribute	提供使用 jQuery 验证插件远程验证程序的特性。
RequiredAttributeAdapter	提供 RequiredAttributeAttribute 特性的适配器。
RequireHttpsAttribute	表示一个特性，该特性用于强制通过 HTTPS 重新发送不安全的 HTTP 请求。
ResultExecutedContext	提供 ActionFilterAttribute 类的 OnResultExecuted 方法的上下文。

ResultExecutingContext	提供 ActionFilterAttribute 类的 OnResultExecuting 方法的上下文。
RouteCollectionExtensions	扩展 RouteCollection 对象以进行 MVC 路由。
RouteDataValueProvider	表示实现 IDictionary(TKey, TValue) 接口的对象中包含的路由数据的值提供程序。
RouteDataValueProviderFactory	表示用来创建路由数据值提供程序对象的工厂。
SelectList	表示一个列表，用户可从该列表中选择一个项。
SelectListItem	表示 SelectList 类的实例中的选定项。
SessionStateAttribute	指定控制器的会话状态。
SessionStateTempDataProvider	为当前 TempDataDictionary 对象提供会话状态数据。
StringLengthAttributeAdapter	提供 StringLengthAttribute 特性的适配器。
TempDataDictionary	表示仅从一个请求保持到下一个请求的数据集。
TemplateInfo	封装有关当前模板上下文的信息。
UrlHelper	包含用于为应用程序内的 ASP.NET MVC 生成 URL 的方法。
UrlParameter	表示路由过程中 MvcHandler 类使用的可选参数。
ValidatableObjectAdapter	提供可验证的对象适配器。
ValidateAntiForgeryTokenAttribute	表示用于阻止伪造请求的特性。
ValidateInputAttribute	表示一个特性，该特性用于标记必须验证其输入的操作方法。
ValueProviderCollection	表示应用程序的值提供程序对象的集合。
ValueProviderDictionary	已过时 。表示应用程序的值提供程序的字典。
ValueProviderFactories	表示值提供程序工厂对象的容器。
ValueProviderFactory	表示用来创建值提供程序对象的工厂。
ValueProviderFactoryCollection	表示应用程序的值提供程序工厂的集合。
ValueProviderResult	表示将一个值（如表单发送的值或查询字符串中的值）绑定到操作方法参数属性或绑定到该参数本身的结果。
ViewContext	封装与呈现视图相关的信息。
ViewDataDictionary	表示一个容器，该容器用于在控制器和视图之间传递数据。
ViewDataDictionary(TModel)	表示一个容器，该容器用于在控制器和视图之间传递强类型数据。
ViewDataInfo	对开发模板所使用的当前模板内容和与模板交互的 HTML 帮助器的相关信息进行封装。
ViewEngineCollection	表示对应用程序可用的视图引擎的集合。
ViewEngineResult	表示定位视图引擎的结果。
ViewEngines	表示对应用程序可用的视图引擎的集合。
ViewMasterPage	表示生成母版视图页所需的信息。
ViewMasterPage(TModel)	表示生成强类型母版视图页所需的信息。

ViewPage	表示将视图呈现为 Web Forms 页所需的属性和方法。
ViewPage(TModel)	表示将强类型视图呈现为 Web Forms 页所需的信息。
ViewResult	表示一个类，该类用于使用由 IViewEngine 对象返回的 IView 实例来呈现视图。
ViewResultBase	表示一个用于为视图提供模型并向响应呈现视图的基类。
ViewStartPage	提供可用于实现视图启动（母版）页的抽象类。
ViewTemplateUserControl	提供 TemplateInfo 对象的容器。
ViewTemplateUserControl(TModel)	提供 TemplateInfo 对象的容器。
ViewType	表示视图的类型。
ViewUserControl	表示生成用户控件所需的信息。
ViewUserControl(TModel)	表示生成强类型用户控件所需的信息。
VirtualPathProviderViewEngine	表示 IViewEngine 接口的抽象基类实现。
WebFormView	表示在 ASP.NET MVC 中生成 Web Forms 页时所需的信息。
WebFormViewEngine	表示一个用于向响应呈现 Web Forms 页的视图引擎。
WebViewPage	表示呈现使用 ASP.NET Razor 语法的视图所需的属性和方法。
WebViewPage(TModel)	表示呈现使用 ASP.NET Razor 语法的视图所需的属性和方法。

接口

接口	描述
IActionFilter	定义操作筛选器中使用的方法。
IActionInvoker	定义操作调用程序的协定，该调用程序用于调用一个操作以响应 HTTP 请求。
IAuthorizationFilter	定义授权筛选器所需的方法。
IClientValidatable	为 ASP.NET MVC 验证框架提供一种用于在运行时发现验证程序是否支持客户端验证的方法。
IController	定义控制器所需的方法。
IControllerActivator	对使用依赖关系注入来实例化控制器的方式进行精细控制。
IControllerFactory	定义控制器工厂所需的方法。
IDependencyResolver	定义可简化服务位置和依赖关系解析的方法。
IExceptionHandler	定义异常筛选器所需的方法。
IFilterProvider	提供用于查找筛选器的接口。
IMetadataAware	提供用于向 AssociatedMetadataProvider 类公开特性的接口。
IMoelBinder	定义模型联编程序所需的方法。
IMoelBinderProvider	定义用于为实现 IMoelBinder 接口的类动态实现模型绑定的方法。
IMcFilter	定义用于指定筛选器顺序以及是否允许多个筛选器的成员。
IResultFilter	定义结果筛选器所需的方法。

IRouteWithArea	将路由与 ASP.NET MVC 应用程序中的区域关联。
ITempDataProvider	定义临时数据提供程序的协定，这些临时数据提供程序用于存储在下一个请求中查看的数据。
IUnvalidatedValueProvider	表示一个可跳过请求验证的 IValueProvider 接口。
IValueProvider	定义 ASP.NET MVC 中的值提供程序所需的方法。
IView	定义视图所需的方法。
ViewDataContainer	定义视图数据字典所需的方法。
ViewEngine	定义视图引擎所需的方法。
ViewLocationCache	定义在内存中缓存视图位置所需的方法。
ViewPageActivator	对使用依赖关系注入创建视图页的方式进行精细控制。



ASP.NET Web Forms - 教程

ASP.NET 是一个使用 HTML、CSS、JavaScript 和服务端脚本创建网页和网站的开发框架。

ASP.NET 支持三种不同的开发模式：

Web Pages（Web 页面）、MVC（Model View Controller 模型-视图-控制器）、Web Forms（Web 窗体）：

本教程介绍 **Web Forms**。

Web Pages

MVC

Web Forms

从何入手？

多数开发人员学习一个新技术，是从查看运行实例开始的。

如果您想查看一个 Web Forms 运行实例，请查看以下的 [ASP.NET Web Forms 演示](#)。

什么是 Web Forms？

Web Forms 是三种创建 ASP.NET 网站和 Web 应用程序的编程模式中的一种。

其他两种编程模式是 Web Pages 和 MVC（Model View Controller 模型-视图-控制器）。

Web Forms 是最古老的 ASP.NET 编程模式，是整合了 HTML、服务器控件和服务端代码的事件驱动网页。

Web Forms 是在服务器上编译和执行的，再由服务器生成 HTML 显示为网页。

Web Forms 有数以百计的 Web 控件和 Web 组件用来创建带有数据访问的用户驱动网站。

Visual Studio Express 2012/2010

Visual Studio Express 是 Microsoft Visual Studio 的免费版本。

Visual Studio Express 是为 Web Forms（和 MVC）量身定制的开发工具。

Visual Studio Express 包含：

- MVC 和 Web Forms
- 拖拽 Web 控件和 Web 组件
- Web 服务器语言（Razor 使用 VB 或者 C#）
- Web 服务器（IIS Express）
- 数据库服务器（SQL Server Compact）
- 完整的 Web 开发框架（ASP.NET）

如果您已经安装了 Visual Studio Express，您将从本教程中学到更多。

如果您想安装 Visual Studio Express，请点击下列链接中的一个：

- [Visual Web Developer 2012](#)（Windows 7 或者 Windows 8）
- [Visual Web Developer 2010](#)（Windows Vista 或者 XP）

ASP.NET 参考手册

在本教程的最后，您将看到一套完整的 ASP.NET 参考手册，介绍了对象、组件、属性和方法。

[ASP.NET 参考手册](#)

[ASP.NET MVC 参考手册](#) [ASP.NET Web 页面](#)

[点我分享笔记](#)

反馈/建议



[ASP.NET Web Forms 教程](#) [ASP.NET 服务器控件](#)

ASP.NET Web Forms - HTML 页面

简单的 ASP.NET 页面看上去就像普通的 HTML 页面。

Hello RUNOOB.COM

在开始学习 ASP.NET 之前，我们先来构建一个简单的 HTML 页面，该页面将在浏览器中显示 "Hello RUNOOB.COM"：

Hello RUNOOB.COM!

用 HTML 编写的 Hello RUNOOB.COM

下面的代码将以 HTML 页面的形式显示实例：

```
<html>
<body bgcolor="yellow">
<center>
<h2>Hello RUNOOB.COM!</h2>
</center>
</body>
</html>
```

如果您想亲自尝试一下，请保存上面的代码到一个名为 "**firstpage.htm**" 的文件中，并创建一个到该文件的链接：[firstpage.htm](#)。

用 ASP.NET 编写的 Hello RUNOOB.COM

转换 HTML 页面为 ASP.NET 页面最简单的方法是，直接复制一个 HTML 文件，并把新文件的扩展名改成 **.aspx**。

下面的代码将以 ASP.NET 页面的形式显示实例：

```
<html>
<body bgcolor="yellow">
<center>
<h2>Hello RUNOOB.COM!</h2>
</center>
</body>
</html>
```

如果您想亲自尝试一下，请保存上面的代码到一个名为 "**firstpage.aspx**" 的文件中，并创建一个到该文件的链接：[firstpage.aspx](#)。

它是如何工作的？

从根本上讲，ASP.NET 页面与 HTML 是完全相同的。

HTML 页面的扩展名是 .htm。如果浏览器向服务器请求一个 HTML 页面，服务器可以不进行任何修改，就直接发送页面给浏览器。

ASP.NET 页面的扩展名是 .aspx。如果浏览器向服务器请求一个 ASP.NET 页面，服务器在将结果发回给浏览器之前，需要先处理页面中的可执行代码。

上面的 ASP.NET 页面不包含任何可执行的代码，所以没有执行任何东西。在下面的实例中，我们将添加一些可执行的代码到页面中，以便演示静态 HTML 页面和动态 ASP 页面的不同之处。

经典 ASP

Active Server Pages (ASP) 已经流行很多年了。通过 ASP，可以在 HTML 页面中放置可执行代码。

之前的 ASP 版本（在 ASP.NET 之前）通常被称为经典 ASP。

ASP.NET 不完全兼容经典 ASP，但是只需要经过少量的修改，大部分经典 ASP 页面就可以作为 ASP.NET 页面良好地运行。

如果您想学习更多关于经典 ASP 的知识，请访问我们的 [ASP 教程](#)。

用经典 ASP 编写的动态页面

为了演示 ASP 是如何显示包含动态内容的页面，我们将向上面的实例中添加一些可执行的代码（红色字体标识）：

```
<html>
<body bgcolor="yellow">
<center>
<h2>Hello RUNOOB.COM!</h2>
<p><%Response.Write(now())%></p>
</center>
</body>
</html>
```

<% -%> 标签内的代码是在服务器上执行的。

Response.Write 是用来向 HTML 输出流中写东西的 ASP 代码。

Now() 是一个返回服务器当前日期和时间的函数。

如果您想亲自尝试一下，请保存上面的代码到一个名为 "**dynpage.asp**" 的文件中，并创建一个到该文件的链接：[dynpage.asp](#)。

用 ASP.NET 编写的动态页面

下面的代码将以 ASP.NET 页面的形式显示实例：

```
<html>
<body bgcolor="yellow">
<center>
<h2>Hello RUNOOB.COM!</h2>
```



```
<p><%Response.Write(now())%></p>
</center>
</body>
</html>
```

如果您想亲自尝试一下，请保存上面的代码到一个名为 "dynpage.aspx" 的文件中，并创建一个到该文件的链接：[dynpage.aspx](#)。

ASP.NET 对比经典 ASP

上面的实例无法演示 ASP.NET 与经典 ASP 之间任何的不同之处。

正如最后的两个实例中，您看不出 ASP 页面和 ASP.NET 页面两者之间的不同之处。

在下一章中，您将看到服务器控件是如何让 ASP.NET 比经典 ASP 更强大的。

[❏ ASP.NET Web Forms 教程](#)

ASP.NET 服务器控件 [❏](#)

[❏ 点我分享笔记](#)

反馈/建议

Copyright © 2013-2018 菜鸟教程 [runoob.com](#) All Rights Reserved. 备案号：闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

[❏ ASP.NET Web 页面](#)

ASP.NET 事件句柄 [❏](#)

ASP.NET Web Forms - 服务器控件

服务器控件是服务器可理解的标签。

经典 ASP 的局限性

下面列出的代码是从上一章中复制的：

```
<html>
<body bgcolor="yellow">
<center>
<h2>Hello Runoob!</h2>
<p><%Response.Write(now())%></p>
</center>
</body>
</html>
```

上面的代码反映出经典 ASP 的局限性：代码块必须放置在您想要输出显示的位置。

通过经典 ASP，想要把可执行代码从 HTML 页面中分离出来是不可能的。这让页面变得难以阅读，也难以维护。

ASP.NET - 服务器控件

ASP.NET 通过服务器控件，已经解决了上述的"意大利面条式代码"问题。

服务器控件是服务器可理解的标签。

有三种类型的服务器控件：

HTML 服务器控件 - 创建的 HTML 标签

Web 服务器控件 - 新的 ASP.NET 标签

Validation 服务器控件 - 用于输入验证

ASP.NET - HTML 服务器控件

HTML 服务器控件是服务器可理解的 HTML 标签。

ASP.NET 文件中的 HTML 元素，默认是作为文本进行处理的。要想让这些元素可编程，需向 HTML 元素中添加 `runat="server"` 属性。这个属性表示，该元素将被作为服务器控件进行处理。同时需要添加 `id` 属性来标识服务器控件。`id` 引用可用于操作运行时的服务器控件。

注释：所有 HTML 服务器控件必须位于带有 `runat="server"` 属性的 `<form>` 标签内。`runat="server"` 属性表明了该表单必须在服务器上进行处理。同时也表明了包含在它内部的控件可被服务器脚本访问。

在下面的实例中，我们在 `.aspx` 文件中声明了一个 `HtmlAnchor` 服务器控件。然后我们在一个事件句柄（事件句柄是一种针对给定事件执行代码的子例程）中操作 `HtmlAnchor` 控件的 `HRef` 属性。`Page_Load` 事件是 ASP.NET 可理解的多种事件中的一种：

```
<script runat="server">
Sub Page_Load
link1.HRef="http://www.runoob.com"
End Sub
</script>

<html>
<body>

<form runat="server">
<a id="link1" runat="server">Visit RUNOOB!</a>
</form>

</body>
</html>
```

可执行代码本身已经被移到 HTML 之外了。

ASP.NET - Web 服务器控件

Web 服务器控件是服务器可理解的特殊 ASP.NET 标签。

就像 HTML 服务器控件，Web 服务器控件也是在服务器上创建的，它们同样需要 `runat="server"` 属性才能生效。然而，Web 服务器控件没有必要映射任何已存在的 HTML 元素，它们可以表示更复杂的元素。

创建 Web 服务器控件的语法是：

```
<asp:control_name id="some_id" runat="server" />
```

在下面的实例中，我们在 `.aspx` 文件中声明了一个 `Button` 服务器控件。然后我们为 `Click` 事件创建一个事件句柄，用来改变按钮上的文本：

```
<script runat="server">
Sub submit(Source As Object, e As EventArgs)
button1.Text="You clicked me!"
End Sub
</script>

<html>
<body>

<form runat="server">
<asp:Button id="button1" Text="Click me!"
runat="server" OnClick="submit"/>
</form>

</body>
</html>
```

ASP.NET - Validation 服务器控件

Validation 服务器控件是用来验证用户输入的。如果用户输入没有通过验证，将显示一条错误消息给用户。

每种 validation 控件执行一种指定类型的验证（比如验证某个指定的值或者某个范围的值）。

在默认情况下，当 `Button`、`ImageButton`、`LinkButton` 控件被点击时，会执行页面验证。您可以设置 `CausesValidation` 为 `false`，来阻止按钮控件被点击时进行验证。

创建 Validation 服务器控件的语法是：

```
<asp:control_name id="some_id" runat="server" />
```

在下面的实例中，我们在 `.aspx` 文件中声明了一个 `TextBox` 控件、一个 `Button` 控件、一个 `RangeValidator` 控件。如果验证失败，文本 `"The value must be from 1 to 100!"` 将会显示在 `RangeValidator` 控件中：

实例

```
<html>
<body>

<form runat="server">
<p>Enter a number from 1 to 100:
<asp:TextBox id="tbox1" runat="server" />
<br /><br />
<asp:Button Text="Submit" runat="server" />
</p>

<p>
<asp:RangeValidator
ControlToValidate="tbox1"
MinimumValue="1"
MaximumValue="100"
Type="Integer"
Text="The value must be from 1 to 100!"
runat="server" />
</p>
</form>

</body>
</html>
```

[☐ ASP.NET Web 页面](#)

ASP.NET 事件句柄 [☐](#)

[☐ 点我分享笔记](#)

反馈/建议

Copyright © 2013-2018 菜鸟教程 [runoob.com](#) All Rights Reserved. 备案号：闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

[☐ ASP.NET 服务器控件](#)

ASP.NET Web 表单 [☐](#)

ASP.NET Web Forms - 事件

事件句柄是一种针对给定事件来执行代码的子例程。

ASP.NET - 事件句柄

请看下面的代码：

```
<%
lbl1.Text="The date and time is " & now()
%>

<html>
<body>
<form runat="server">
<h3><asp:label id="lbl1" runat="server" /></h3>
</form>
</body>
</html>
```

上面的代码将在何时被执行？答案是：“不知道...”。

Page_Load 事件

Page_Load 事件是 ASP.NET 可理解的众多事件之一。Page_Load 事件会在页面加载时被触发，ASP.NET 将自动调用 Page_Load 子例程，并执行其中的代码：

实例

```
<script runat="server">
Sub Page_Load
lbl1.Text="The date and time is " & now()
End Sub
</script>

<html>
<body>
<form runat="server">
<h3><asp:label id="lbl1" runat="server" /></h3>
</form>
</body>
</html>
```

[演示实例 »](#)

注释：Page_Load 事件不包含对象引用或事件参数！

Page.IsPostBack 属性

Page_Load 子例程会在页面每次加载时运行。如果您只想在页面第一次加载时执行 Page_Load 子例程中的代码，那么您可以使用 Page.IsPostBack 属性。如果 Page.IsPostBack 属性设置为 false，则页面第一次被载入，如果设置为 true，则页面被传回到服务器（比如，通过点击表单上的按钮）：

实例

```
<script runat="server">
Sub Page_Load
if Not Page.IsPostBack then
lbl1.Text="The date and time is " & now()
end if
End Sub

Sub submit(s As Object, e As EventArgs)
lbl2.Text="Hello World!"
End Sub
</script>

<html>
<body>
<form runat="server">
<h3><asp:label id="lbl1" runat="server" /></h3>
<h3><asp:label id="lbl2" runat="server" /></h3>
<asp:button text="Submit" onclick="submit" runat="server" />
</form>
</body>
</html>
```

[演示实例 »](#)

上面的实例仅在页面第一次加载时显示 "The date and time is...." 消息。当用户点击 Submit 按钮是，submit 子例程将会第二个 label 中写入 "Hello World!"，但是第一个 label 中的日期和时间不会改变。

[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)[ASP.NET 事件句柄](#)ASP.NET ViewState [□](#)

ASP.NET Web Forms - HTML 表单

所有的服务器控件都必须出现在 `<form>` 标签中, `<form>` 标签必须包含 `runat="server"` 属性。

ASP.NET Web 表单

所有的服务器控件都必须出现在 `<form>` 标签中, `<form>` 标签必须包含 `runat="server"` 属性。`runat="server"` 属性表明该表单必须在服务器上进行处理。同时也表明了包含在它内部的控件可被服务器脚本访问:

```
<form runat="server">

...HTML + server controls

</form>
```

注释: 该表单总是被提交到自身页面。如果您指定了一个 `action` 属性, 它会被忽略。如果您省略了 `method` 属性, 它将会默认设置 `method="post"`。同时, 如果您没有指定 `name` 和 `id` 属性, 它们会由 ASP.NET 自动分配。

注释: 一个 .aspx 页面只能包含一个 `<form runat="server">` 控件!

如果您在一个包含不带有 `name`、`method`、`action` 或 `id` 属性的表单的 .aspx 页面中选择查看源代码, 您会看到 ASP.NET 添加这些属性到表单上了, 如下所示:

```
<form name="_ctl0" method="post" action="page.aspx" id="_ctl0">

...some code

</form>
```

提交表单

表单通常通过点击按钮来提交。ASP.NET 中的 `Button` 服务器控件的格式如下:

```
<asp:Button id="id" text="label" OnClick="sub" runat="server" />
```

`id` 属性为按钮定义了一个唯一的名称, `text` 属性为按钮分配了一个标签。`OnClick` 事件句柄规定了一个要执行的已命名的子例程。

在下面的例子中, 我们在一个 .aspx 文件中声明了一个按钮控件。一次鼠标单击就可以运行一个子例程, 可以更改该按钮上的文本。

[实例](#)

[ASP.NET 事件句柄](#)ASP.NET ViewState [□](#)[点我分享笔记](#)

ASP.NET Web Forms - 维持 ViewState

通过在您的 Web Form 中维持对象的 **ViewState**（视图状态），您可以省去大量的编码工作。

维持 ViewState（视图状态）

在经典 ASP 中，当一个表单被提交时，所有的表单值都会被清空。假设您提交了一个带有大量信息的表单，而服务器返回了一个错误。您不得不回到表单改正信息。您点击返回按钮，然后发生了什么.....所有表单值都被清空了，您不得不重新开始所有的一切！站点没有维持您的 **ViewState**。

在 ASP .NET 中，当一个表单被提交时，表单会连同表单值一起出现在浏览器窗口中。如何做到的呢？这是因为 ASP .NET 维持了您的 **ViewState**。**ViewState** 会在页面被提交到服务器时表明它的状态。这个状态是通过在带有 `<form runat="server">` 控件的每个页面上放置一个隐藏域定义的。源代码如下所示：

```
<form name="_ctl0" method="post" action="page.aspx" id="_ctl0">
<input type="hidden" name="__VIEWSTATE"
value="dDwtNTI0ODU5MDElOzs+ZBCF2ryjMpeVgUrY2eTj79HNl4Q=" />

.....some code

</form>
```

维持 **ViewState** 是 ASP.NET Web Forms 的默认设置。如果您想不维持 **ViewState**，请在 .aspx 页面顶部包含指令 `<%@ Page EnableViewState="false" %>`，或者向任意控件添加属性 `EnableViewState="false"`。

请看下面的 .aspx 文件。它演示了"老"的运行方式。当您点击提交按钮，表单值将会消失：

实例

```
<html>
<body>

<form action="demo_classicasp.aspx" method="post">
Your name: <input type="text" name="fname" size="20">
<input type="submit" value="Submit">
</form>
<%
dim fname
fname=Request.Form("fname")
If fname<>"" Then
Response.Write("Hello " & fname & "!")
End If
%>

</body>
</html>
```

[演示实例 »](#)

下面是新的 ASP .NET 方式。当您点击提交按钮，表单值不会消失：

实例

点击实例的右边框架中的查看源代码，您将看到 ASP .NET 已经在表单中添加了一个隐藏域来维持 **ViewState**。

```
<script runat="server">
Sub submit(sender As Object, e As EventArgs)
lbl1.Text="Hello " & txt1.Text & "!"
End Sub
</script>

<html>
<body>
```

```
<form runat="server">
Your name: <asp:TextBox id="txt1" runat="server" />
<asp:Button OnClick="submit" Text="Submit" runat="server" />
<p><asp:Label id="lbl1" runat="server" /></p>
</form>

</body>
</html>
```

演示实例 »

☐ ASP.NET Web 表单

ASP.NET TextBox 控件 ☐

☐ 点我分享笔记

反馈/建议

Copyright © 2013-2018 菜鸟教程 runoob.com All Rights Reserved. 备案号：闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

☐ ASP.NET ViewState

ASP.NET Button 控件 ☐

ASP.NET Web Forms - TextBox 控件

TextBox 控件用于创建用户可输入文本的文本框。

TextBox 控件

TextBox 控件用于创建用户可输入文本的文本框。

TextBox 控件的特性和属性列在我们的 [WebForms 控件参考手册页面](#)。

下面的实例演示了您可能会用到的 TextBox 控件的一些属性：

实例

```
<html>
<body>

<form runat="server">

A basic TextBox:
<asp:TextBox id="tb1" runat="server" />
<br /><br />

A password TextBox:
<asp:TextBox id="tb2" TextMode="password" runat="server" />
<br /><br />

A TextBox with text:
<asp:TextBox id="tb4" Text="Hello World!" runat="server" />
<br /><br />

A multiline TextBox:
<asp:TextBox id="tb3" TextMode="multiline" runat="server" />
<br /><br />
```

```
A TextBox with height:
<asp:TextBox id="tb6" rows="5" TextMode="multiline"
runat="server" />
<br /><br />
```

```
A TextBox with width:
<asp:TextBox id="tb5" columns="30" runat="server" />
```

```
</form>
```

```
</body>
```

```
</html>
```

[演示实例 »](#)

添加脚本

当表单被提交时，**TextBox** 控件的内容和设置可能会被服务器脚本修改。表单可通过点击一个按钮或当用户修改 **TextBox** 控件的值的时候进行提交。

在下面的实例中，我们在 **.aspx** 文件中声明了一个 **TextBox** 控件、一个 **Button** 控件和一个 **Label** 控件。当提交按钮被触发时，**submit** 子例程将被执行。**submit** 子例程将写入一行文本到 **Label** 控件中：

实例

```
<script runat="server">
Sub submit(sender As Object, e As EventArgs)
    lbl1.Text="Your name is " & txt1.Text
End Sub
</script>

<html>
<body>

<form runat="server">
Enter your name:
<asp:TextBox id="txt1" runat="server" />
<asp:Button OnClick="submit" Text="Submit" runat="server" />
<p><asp:Label id="lbl1" runat="server" /></p>
</form>

</body>
</html>
```

[演示实例 »](#)

在下面的实例中，我们在 **.aspx** 文件中声明了一个 **TextBox** 控件和一个 **Label** 控件。当您修改了 **TextBox** 中的值，并且在 **TextBox** 外部点击（或者按下了 **Tab** 键）时，**change** 子例程将会被执行。**change** 子例程将写入一行文本到 **Label** 控件中：

实例

```
<script runat="server">
Sub change(sender As Object, e As EventArgs)
    lbl1.Text="You changed text to " & txt1.Text
End Sub
</script>

<html>
<body>

<form runat="server">
Enter your name:
<asp:TextBox id="txt1" runat="server"
text="Hello World!"
ontextchanged="change" autopostback="true"/>
<p><asp:Label id="lbl1" runat="server" /></p>
</form>

</body>
</html>
```


演示实例 »

☐ ASP.NET ViewState

ASP.NET Button 控件 ☐

☐ 点我分享笔记

反馈/建议

Copyright © 2013-2018 菜鸟教程 runoob.com All Rights Reserved. 备案号: 闽ICP备15012807号-1

☐

[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

☐ ASP.NET TextBox 控件

ASP.NET 数据绑定 ☐

ASP.NET Web Forms - Button 控件

Button 控件用于显示一个下压按钮。

Button 控件

Button 控件用于显示一个下压按钮。下压按钮可能是一个提交按钮或者是一个命令按钮。在默认情况下，这个控件是提交按钮。

提交按钮没有命令名称，当它被点击时，它会把页面传回到服务器。您可以编写一些事件句柄，当提交按钮被点击时，用来控制动作的执行。

命令按钮有命令名称，并且允许您在页面上创建多个 Button 控件。您可以编写一些时间句柄，当命令按钮被点击时，用来控制动作的执行。

Button 控件的特性和属性列在我们的 [WebForms 控件参考手册页面](#)。

下面的实例演示了一个简单的 Button 控件：

```
<html>
<body>

<form runat="server">
<asp:Button id="b1" Text="Submit" runat="server" />
</form>

</body>
</html>
```

添加脚本

表单通常通过点击按钮进行提交。

在下面的实例中，我们在 .aspx 文件中声明了一个 TextBox 控件、一个 Button 控件和一个 Label 控件。当提交按钮被触发时，submit 子例程将被执行。submit 子例程将写入一行文本到 Label 控件中：

实例

```
<script runat="server">
Sub submit(sender As Object, e As EventArgs)
lb11.Text="Your name is " & txt1.Text
End Sub
</script>

<html>
<body>
```

```
<form runat="server">
Enter your name:
<asp:TextBox id="txt1" runat="server" />
<asp:Button OnClick="submit" Text="Submit" runat="server" />
<p><asp:Label id="lbl1" runat="server" /></p>
</form>

</body>
</html>
```

演示实例 »

☐ ASP.NET TextBox 控件

ASP.NET 数据绑定 ☐

☐ 点我分享笔记

反馈/建议

Copyright © 2013-2018 菜鸟教程 runoob.com All Rights Reserved. 备案号：闽ICP备15012807号-1

☐

[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

☐ ASP.NET Button 控件

ASP.NET ArrayList ☐

ASP.NET Web Forms - 数据绑定

我们可以使用数据绑定（Data Binding）来完成带可选项的列表，这些可选项来自某个导入的数据源，比如数据库、XML 文件或者脚本。

数据绑定

下面的控件是支持数据绑定的列表控件：

asp:RadioButtonList

asp:CheckBoxList

asp:DropDownList

asp:Listbox

以上每个控件的可选项通常是在一个或者多个 `asp:ListItem` 控件中定义，如下：

```
<html>
<body>

<form runat="server">
<asp:RadioButtonList id="countrylist" runat="server">
<asp:ListItem value="N" text="Norway" />
<asp:ListItem value="S" text="Sweden" />
<asp:ListItem value="F" text="France" />
<asp:ListItem value="I" text="Italy" />
</asp:RadioButtonList>
</form>

</body>
</html>
```

然而，我们可以使用某种独立的数据源进行数据绑定，比如数据库、XML 文件或者脚本，通过数据绑定来填充列表的可选项。

通过使用导入的数据源，数据从 HTML 中分离出来，并且对可选项的修改都是在独立的数据源中完成的。

在下面的三个章节中，我们将描述如何从脚本化的数据源中绑定数据。

☐ ASP.NET Button 控件

ASP.NET ArrayList ☐

☐ 点我分享笔记

反馈/建议

Copyright © 2013-2018 菜鸟教程 runoob.com All Rights Reserved. 备案号：闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

☐ ASP.NET 数据绑定

ASP.NET Hashtable ☐

ASP.NET Web Forms - ArrayList 对象

ArrayList 对象是包含单个数据值的项目的集合。

实例

尝试一下 - 实例

[ArrayList DropDownList](#)

[ArrayList RadioButtonList](#)

创建 ArrayList

ArrayList 对象是包含单个数据值的项目的集合。

通过 Add() 方法向 ArrayList 添加项目。

下面的代码创建了一个名为 mycountries 的 ArrayList 对象，并添加了四个项目：

```
<script runat="server">
Sub Page_Load
if Not Page.IsPostBack then
dim mycountries=New ArrayList
mycountries.Add("Norway")
mycountries.Add("Sweden")
mycountries.Add("France")
mycountries.Add("Italy")
end if
end sub
</script>
```

在默认情况下，一个 ArrayList 对象包含 16 个条目。可通过 TrimToSize() 方法把 ArrayList 调整为最终尺寸：

```
<script runat="server">
Sub Page_Load
if Not Page.IsPostBack then
dim mycountries=New ArrayList
mycountries.Add("Norway")
mycountries.Add("Sweden")
mycountries.Add("France")
mycountries.Add("Italy")
mycountries.TrimToSize()
end if
end sub
</script>
```

通过 Sort() 方法，ArrayList 也能够按照字母顺序或者数字顺序进行排序：

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

```
Sub Page_Load
if Not Page.IsPostBack then
dim mycountries=New ArrayList
mycountries.Add("Norway")
mycountries.Add("Sweden")
mycountries.Add("France")
mycountries.Add("Italy")
mycountries.TrimToSize()
mycountries.Sort()
end if
end sub
</script>
```

要实现反向排序，请在 **Sort()** 方法后应用 **Reverse()** 方法：

```
<script runat="server">
Sub Page_Load
if Not Page.IsPostBack then
dim mycountries=New ArrayList
mycountries.Add("Norway")
mycountries.Add("Sweden")
mycountries.Add("France")
mycountries.Add("Italy")
mycountries.TrimToSize()
mycountries.Sort()
mycountries.Reverse()
end if
end sub
</script>
```

绑定数据到 **ArrayList**

ArrayList 对象可为下列的控件自动生成文本和值：

asp:RadioButtonList

asp:CheckBoxList

asp:DropDownList

asp:Listbox

为了绑定数据到 **RadioButtonList** 控件，首先要在 **.aspx** 页面中创建一个 **RadioButtonList** 控件（不带任何 **asp:ListItem** 元素）：

```
<html>
<body>

<form runat="server">
<asp:RadioButtonList id="rb" runat="server" />
</form>

</body>
</html>
```

然后添加创建列表的脚本，并且绑定列表中的值到 **RadioButtonList** 控件：

实例

```
<script runat="server">
Sub Page_Load
if Not Page.IsPostBack then
dim mycountries=New ArrayList
mycountries.Add("Norway")
mycountries.Add("Sweden")
mycountries.Add("France")
mycountries.Add("Italy")
mycountries.TrimToSize()
mycountries.Sort()
rb.DataSource=mycountries
rb.DataBind()
end if
end sub
</script>

<html>
<body>

<form runat="server">
```

```
<asp:RadioButtonList id="rb" runat="server" />
</form>

</body>
</html>
```

演示实例 »

RadioButtonList 控件的 DataSource 属性被设置为该 ArrayList，它定义了这个 RadioButtonList 控件的数据源。RadioButtonList 控件的 DataBind() 方法把 RadioButtonList 控件与数据源绑定在一起。

注释：数据值作为控件的 Text 和 Value 属性来使用。如需添加不同于 Text 的 Value，请使用 Hashtable 对象或者 SortedList 对象。

☐ ASP.NET 数据绑定

ASP.NET Hashtable ☐

☐ 点我分享笔记

反馈/建议

Copyright © 2013-2018 菜鸟教程 runoob.com All Rights Reserved. 备案号：闽ICP备15012807号-1

☐

[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

☐ ASP.NET ArrayList

ASP.NET SortedList ☐

ASP.NET Web Forms - Hashtable 对象

Hashtable 对象包含用键/值对表示的项目。

尝试一下 - 实例

[Hashtable RadiobuttonList 1](#)

[Hashtable RadiobuttonList 2](#)

[Hashtable DropDownList](#)

创建 Hashtable

Hashtable 对象包含用键/值对表示的项目。键被用作索引，通过搜索键，可以实现对值的快速搜索。

通过 Add() 方法向 Hashtable 添加项目。

下面的代码创建了一个名为 mycountries 的 Hashtable 对象，并添加了四个元素：

```
<script runat="server">
Sub Page_Load
if Not Page.IsPostBack then
dim mycountries=New Hashtable
mycountries.Add("N","Norway")
mycountries.Add("S","Sweden")
mycountries.Add("F","France")
mycountries.Add("I","Italy")
end if
end sub
</script>
```

数据绑定

Hashtable 对象可为下列的控件自动生成文本和值:

asp:RadioButtonList

asp:CheckBoxList

asp:DropDownList

asp:Listbox

为了绑定数据到 **RadioButtonList** 控件, 首先要在 .aspx 页面中创建一个 **RadioButtonList** 控件 (不带任何 **asp:ListItem** 元素):

```
<html>
<body>

<form runat="server">
<asp:RadioButtonList id="rb" runat="server" AutoPostBack="True" />
</form>

</body>
</html>
```

然后添加创建列表的脚本, 并且绑定列表中的值到 **RadioButtonList** 控件:

```
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
dim mycountries=New Hashtable
mycountries.Add("N","Norway")
mycountries.Add("S","Sweden")
mycountries.Add("F","France")
mycountries.Add("I","Italy")
rb.DataSource=mycountries
rb.DataValueField="Key"
rb.DataTextField="Value"
rb.DataBind()
end if
end sub
</script>

<html>
<body>

<form runat="server">
<asp:RadioButtonList id="rb" runat="server" AutoPostBack="True" />
</form>

</body>
</html>
```

然后我们添加一个子例程, 当用户点击 **RadioButtonList** 控件中的某个项目时, 该子例程会被执行。当某个单选按钮被点击时, **label** 中会出现一行文本:

实例

```
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
dim mycountries=New Hashtable
mycountries.Add("N","Norway")
mycountries.Add("S","Sweden")
mycountries.Add("F","France")
mycountries.Add("I","Italy")
rb.DataSource=mycountries
rb.DataValueField="Key"
rb.DataTextField="Value"
rb.DataBind()
end if
end sub

sub displayMessage(s as Object,e As EventArgs)
lbl1.text="Your favorite country is: " & rb.SelectedItem.Text
end sub
</script>

<html>
```

```
<body>

<form runat="server">
<asp:RadioButtonList id="rb" runat="server"
AutoPostBack="True" onSelectedIndexChanged="displayMessage" />
<p><asp:label id="lb11" runat="server" /></p>
</form>

</body>
</html>
```

演示实例 »

注释：您无法选择添加到 **Hashtable** 的项目的排序方式。如需对项目进行字母排序或者数字排序，请使用 **SortedList** 对象。

[ASP.NET ArrayList](#)

[ASP.NET SortedList](#)

[点我分享笔记](#)

反馈/建议

Copyright © 2013-2018 菜鸟教程 [runoob.com](#) All Rights Reserved. 备案号：闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

[ASP.NET Hashtable](#)

[ASP.NET XML 数据绑定](#)

ASP.NET Web Forms - SortedList 对象

SortedList 对象结合了 **ArrayList** 对象和 **Hashtable** 对象的特性。

[尝试一下 - 实例](#)

[SortedList RadiobuttonList 1](#)

[SortedList RadiobuttonList 2](#)

[SortedList DropDownList](#)

SortedList 对象

SortedList 对象包含用键/值对表示的项目。**SortedList** 对象按照字母顺序或者数字顺序自动地对项目进行排序。

通过 **Add()** 方法向 **SortedList** 添加项目。通过 **TrimToSize()** 方法把 **SortedList** 调整为最终尺寸。

下面的代码创建了一个名为 **mycountries** 的 **SortedList** 对象，并添加了四个元素：

```
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
dim mycountries=New SortedList
mycountries.Add("N","Norway")
mycountries.Add("S","Sweden")
mycountries.Add("F","France")
mycountries.Add("I","Italy")
end if
end sub
</script>
```

数据绑定

`SortedList` 对象可为下列的控件自动生成文本和值：

`asp:RadioButtonList`

`asp:CheckBoxList`

`asp:DropDownList`

`asp:Listbox`

为了绑定数据到 `RadioButtonList` 控件，首先要在 `.aspx` 页面中创建一个 `RadioButtonList` 控件（不带任何 `asp:ListItem` 元素）：

```
<html>
<body>

<form runat="server">
<asp:RadioButtonList id="rb" runat="server" AutoPostBack="True" />
</form>

</body>
</html>
```

然后添加创建列表的脚本，并且绑定列表中的值到 `RadioButtonList` 控件：

```
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
dim mycountries=New SortedList
mycountries.Add("N","Norway")
mycountries.Add("S","Sweden")
mycountries.Add("F","France")
mycountries.Add("I","Italy")
rb.DataSource=mycountries
rb.DataValueField="Key"
rb.DataTextField="Value"
rb.DataBind()
end if
end sub
</script>

<html>
<body>

<form runat="server">
<asp:RadioButtonList id="rb" runat="server" AutoPostBack="True" />
</form>

</body>
</html>
```

然后我们添加一个子例程，当用户点击 `RadioButtonList` 控件中的某个项目时，该子例程会被执行。当某个单选按钮被点击时，`label` 中会出现一行文本：

实例

```
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
dim mycountries=New SortedList
mycountries.Add("N","Norway")
mycountries.Add("S","Sweden")
mycountries.Add("F","France")
mycountries.Add("I","Italy")
rb.DataSource=mycountries
rb.DataValueField="Key"
rb.DataTextField="Value"
rb.DataBind()
end if
end sub

sub displayMessage(s as Object,e As EventArgs)
lbl1.text="Your favorite country is: " & rb.SelectedItem.Text
end sub
</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>
```

演示实例 »

[ASP.NET Hashtable](#)

ASP.NET XML 数据绑定 [ASP.NET XML 数据绑定](#)

[点我分享笔记](#)

反馈/建议

Copyright © 2013-2018 菜鸟教程 [runoob.com](#) All Rights Reserved. 备案号: 闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

[ASP.NET SortedList](#)

ASP.NET Repeater 控件 [ASP.NET Repeater 控件](#)

ASP.NET Web Forms - XML 文件

我们可以绑定 XML 文件到列表控件。

一个 XML 文件

这里有一个名为 "countries.xml" 的 XML 文件:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<countries>

<country>
<text>Norway</text>
<value>N</value>
</country>

<country>
<text>Sweden</text>
<value>S</value>
</country>

<country>
<text>France</text>
<value>F</value>
</country>

<country>
<text>Italy</text>
<value>I</value>
</country>

</countries>
```

查看这个 XML 文件: [countries.xml](#)

绑定 DataSet 到 List 控件

首先, 导入 "System.Data" 命名空间。我们需要该命名空间与 DataSet 对象一起工作。把下面这条指令包含在 .aspx 页面的顶部:

```
<%@ Import Namespace="System.Data" %>
```

接着, 为 XML 文件创建一个 DataSet, 并在页面第一次加载时把这个 XML 文件载入 DataSet:

```
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
dim mycountries=New DataSet
mycountries.ReadXml(MapPath("countries.xml"))
end if
end sub
```

为了绑定数据到 RadioButtonList 控件, 首先要在 .aspx 页面中创建一个 RadioButtonList 控件 (不带任何 asp:ListItem 元素):

```
<html>
<body>

<form runat="server">
<asp:RadioButtonList id="rb" runat="server" AutoPostBack="True" />
</form>

</body>
</html>
```

然后添加创建 XML DataSet 的脚本, 并且绑定 XML DataSet 中的值到 RadioButtonList 控件:

```
<%@ Import Namespace="System.Data" %>

<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
dim mycountries=New DataSet
mycountries.ReadXml(MapPath("countries.xml"))
rb.DataSource=mycountries
rb.DataValueField="value"
rb.DataTextField="text"
rb.DataBind()
end if
end sub
</script>

<html>
<body>

<form runat="server">
<asp:RadioButtonList id="rb" runat="server"
AutoPostBack="True" onSelectedIndexChanged="displayMessage" />
</form>

</body>
</html>
```

然后我们添加一个子例程, 当用户点击 RadioButtonList 控件中的某个项目时, 该子例程会被执行。当某个单选按钮被点击时, label 中会出现一行文本:

实例

```
<%@ Import Namespace="System.Data" %>

<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
dim mycountries=New DataSet
mycountries.ReadXml(MapPath("countries.xml"))
rb.DataSource=mycountries
rb.DataValueField="value"
rb.DataTextField="text"
rb.DataBind()
end if
end sub

sub displayMessage(s as Object,e As EventArgs)
lbl1.text="Your favorite country is: " & rb.SelectedItem.Text
end sub
```

```
</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>
```

演示实例 »

[☐ ASP.NET SortedList](#)

ASP.NET Repeater 控件 [☐](#)

[☐ 点我分享笔记](#)

反馈/建议

Copyright © 2013-2018 菜鸟教程 [runoob.com](#) All Rights Reserved. 备案号: 闽ICP备15012807号-1

☐

[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

[☐ ASP.NET XML 数据绑定](#)

ASP.NET DataList 控件 [☐](#)

ASP.NET Web Forms - Repeater 控件

Repeater 控件用于显示被绑定在该控件上的项目的重复列表。

绑定 DataSet 到 Repeater 控件

Repeater 控件用于显示被绑定在该控件上的项目的重复列表。Repeater 控件可被绑定到数据库表、XML 文件或者其他项目列表。在这里，我们将演示如何绑定 XML 文件到 Repeater 控件。

在我们的实例中，我们将使用下面的 XML 文件 ("cdcatalog.xml")：

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<catalog>
<cd>
<title>Empire Burlesque</title>
<artist>Bob Dylan</artist>
<country>USA</country>
<company>Columbia</company>
<price>10.90</price>
<year>1985</year>
</cd>
<cd>
<title>Hide your heart</title>
<artist>Bonnie Tyler</artist>
<country>UK</country>
<company>CBS Records</company>
<price>9.90</price>
<year>1988</year>
</cd>
<cd>
```

```

<title>Greatest Hits</title>
<artist>Dolly Parton</artist>
<country>USA</country>
<company>RCA</company>
<price>9.90</price>
<year>1982</year>
</cd>
<cd>
<title>Still got the blues</title>
<artist>Gary Moore</artist>
<country>UK</country>
<company>Virgin records</company>
<price>10.20</price>
<year>1990</year>
</cd>
<cd>
<title>Eros</title>
<artist>Eros Ramazzotti</artist>
<country>EU</country>
<company>BMG</company>
<price>9.90</price>
<year>1997</year>
</cd>
</catalog>

```

查看这个 **XML** 文件: [cdcatalog.xml](#)

首先, 导入 **"System.Data"** 命名空间。我们需要该命名空间与 **DataSet** 对象一起工作。把下面这条指令包含在 **.aspx** 页面的顶部:

```
<%@ Import Namespace="System.Data" %>
```

接着, 为 **XML** 文件创建一个 **DataSet**, 并在页面第一次加载时把这个 **XML** 文件载入 **DataSet**:

```

<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
dim mycdcatalog=New DataSet
mycdcatalog.ReadXml (MapPath ("cdcatalog.xml"))
end if
end sub

```

然后我们在 **.aspx** 页面中创建一个 **Repeater** 控件。**<HeaderTemplate>** 元素中的内容被首先呈现, 并且在输出中仅出现一次, 而 **<ItemTemplate>** 元素中的内容会对应 **DataSet** 中的每条 **"record"** 重复出现, 最后, **<FooterTemplate>** 元素中的内容在输出中仅出现一次:

```

<html>
<body>

<form runat="server">
<asp:Repeater id="cdcatalog" runat="server">

<HeaderTemplate>
...
</HeaderTemplate>

<ItemTemplate>
...
</ItemTemplate>

<FooterTemplate>
...
</FooterTemplate>

</asp:Repeater>
</form>

</body>
</html>

```

然后我们添加创建 **DataSet** 的脚本, 并且绑定 **mycdcatalog DataSet** 到 **Repeater** 控件。然后 使用 **HTML** 标签来填充 **Repeater** 控件, 并通过 **<%#Container.DataItem("fieldname")%>** 绑定数据项目到 **<ItemTemplate>** 区域内的单元格中:

实例

```

<%@ Import Namespace="System.Data" %>

<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
dim mycdcatalog=New DataSet
mycdcatalog.ReadXml (MapPath ("cdcatalog.xml"))
cdcatalog.DataSource=mycdcatalog

```

```

cdcatalog.DataBind()
end if
end sub
</script>

<html>
<body>

<form runat="server">
<asp:Repeater id="cdcatalog" runat="server">

<HeaderTemplate>
<table border="1" width="100%">
<tr>
<th>Title</th>
<th>Artist</th>
<th>Country</th>
<th>Company</th>
<th>Price</th>
<th>Year</th>
</tr>
</HeaderTemplate>

<ItemTemplate>
<tr>
<td><%#Container.DataItem("title")%></td>
<td><%#Container.DataItem("artist")%></td>
<td><%#Container.DataItem("country")%></td>
<td><%#Container.DataItem("company")%></td>
<td><%#Container.DataItem("price")%></td>
<td><%#Container.DataItem("year")%></td>
</tr>
</ItemTemplate>

<FooterTemplate>
</table>
</FooterTemplate>

</asp:Repeater>
</form>

</body>
</html>

```

[演示实例 »](#)

使用 <AlternatingItemTemplate>

您可以在 <ItemTemplate> 元素后添加 <AlternatingItemTemplate> 元素，用来描述输出中交替行的外观。在下面的实例中，表格每隔一行就会显示为浅灰色的背景：

实例

```

<%@ Import Namespace="System.Data" %>

<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
dim mycdcatalog=New DataSet
mycdcatalog.ReadXml(MapPath("cdcatalog.xml"))
cdcatalog.DataSource=mycdcatalog
cdcatalog.DataBind()
end if
end sub
</script>

<html>
<body>

<form runat="server">
<asp:Repeater id="cdcatalog" runat="server">

```

```

<HeaderTemplate>
<table border="1" width="100%">
<tr>
<th>Title</th>
<th>Artist</th>
<th>Country</th>
<th>Company</th>
<th>Price</th>
<th>Year</th>
</tr>
</HeaderTemplate>

<ItemTemplate>
<tr>
<td><%#Container.DataItem("title")%></td>
<td><%#Container.DataItem("artist")%></td>
<td><%#Container.DataItem("country")%></td>
<td><%#Container.DataItem("company")%></td>
<td><%#Container.DataItem("price")%></td>
<td><%#Container.DataItem("year")%></td>
</tr>
</ItemTemplate>

<AlternatingItemTemplate>
<tr bgcolor="#e8e8e8">
<td><%#Container.DataItem("title")%></td>
<td><%#Container.DataItem("artist")%></td>
<td><%#Container.DataItem("country")%></td>
<td><%#Container.DataItem("company")%></td>
<td><%#Container.DataItem("price")%></td>
<td><%#Container.DataItem("year")%></td>
</tr>
</AlternatingItemTemplate>

<FooterTemplate>
</table>
</FooterTemplate>

</asp:Repeater>
</form>

</body>
</html>

```

[演示实例 »](#)

使用 <SeparatorTemplate>

<SeparatorTemplate> 元素用于描述每个记录之间的分隔符。在下面的实例中，每个表格行之间插入了一条水平线：

实例

```

<%@ Import Namespace="System.Data" %>

<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
dim mycdcatalog=New DataSet
mycdcatalog.ReadXml(MapPath("cdcatalg.xml"))
cdcatalg.DataSource=mycdcatalog
cdcatalg.DataBind()
end if
end sub
</script>

<html>
<body>

<form runat="server">
<asp:Repeater id="cdcatalg" runat="server">

```

```
<HeaderTemplate>
<table border="0" width="100%">
<tr>
<th>Title</th>
<th>Artist</th>
<th>Country</th>
<th>Company</th>
<th>Price</th>
<th>Year</th>
</tr>
</HeaderTemplate>

<ItemTemplate>
<tr>
<td><%#Container.DataItem("title")%></td>
<td><%#Container.DataItem("artist")%></td>
<td><%#Container.DataItem("country")%></td>
<td><%#Container.DataItem("company")%></td>
<td><%#Container.DataItem("price")%></td>
<td><%#Container.DataItem("year")%></td>
</tr>
</ItemTemplate>

<SeparatorTemplate>
<tr>
<td colspan="6"><hr /></td>
</tr>
</SeparatorTemplate>

<FooterTemplate>
</table>
</FooterTemplate>

</asp:Repeater>
</form>

</body>
</html>
```

[演示实例 »](#)

[ASP.NET XML 数据绑定](#)

[ASP.NET DataList 控件](#)

[点我分享笔记](#)

[反馈/建议](#)

Copyright © 2013-2018 菜鸟教程 [runoob.com](#) All Rights Reserved. 备案号: 闽ICP备15012807号-1

□

[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

[ASP.NET Repeater 控件](#)

[ASP.NET 数据库连接](#)

ASP.NET Web Forms - DataList 控件

DataList 控件，类似于 **Repeater** 控件，用于显示绑定在该控件上的项目的重复列表。不过，**DataList** 控件会默认地在数据项目上添加表格。

绑定 DataSet 到 DataList 控件

DataList 控件，类似于 **Repeater** 控件，用于显示绑定在该控件上的项目的重复列表。不过，**DataList** 控件会默认地在数据项目上添加表格。**DataList** 控件可被绑定到数据库表、XML 文件或者其他项目列表。在这里，我们将演示如何绑定 XML 文件到 **DataList** 控件。

在我们的实例中，我们将使用下面的 XML 文件 ("cdcatalog.xml")：

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<catalog>
<cd>
<title>Empire Burlesque</title>
<artist>Bob Dylan</artist>
<country>USA</country>
<company>Columbia</company>
<price>10.90</price>
<year>1985</year>
</cd>
<cd>
<title>Hide your heart</title>
<artist>Bonnie Tyler</artist>
<country>UK</country>
<company>CBS Records</company>
<price>9.90</price>
<year>1988</year>
</cd>
<cd>
<title>Greatest Hits</title>
<artist>Dolly Parton</artist>
<country>USA</country>
<company>RCA</company>
<price>9.90</price>
<year>1982</year>
</cd>
<cd>
<title>Still got the blues</title>
<artist>Gary Moore</artist>
<country>UK</country>
<company>Virgin records</company>
<price>10.20</price>
<year>1990</year>
</cd>
<cd>
<title>Eros</title>
<artist>Eros Ramazzotti</artist>
<country>EU</country>
<company>BMG</company>
<price>9.90</price>
<year>1997</year>
</cd>
</catalog>
```

查看这个 XML 文件：[cdcatalog.xml](#)

首先，导入 "System.Data" 命名空间。我们需要该命名空间与 **DataSet** 对象一起工作。把下面这条指令包含在 .aspx 页面的顶部：

```
<%@ Import Namespace="System.Data" %>
```

接着，为 XML 文件创建一个 **DataSet**，并在页面第一次加载时把这个 XML 文件载入 **DataSet**：

```
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
dim mycdcatalog=New DataSet
mycdcatalog.ReadXml (MapPath ("cdcatalog.xml"))
end if
end sub
```

然后我们在 .aspx 页面中创建一个 **DataList** 控件。**<HeaderTemplate>** 元素中的内容被首先呈现，并且在输出中仅出现一次，而 **<ItemTemplate>** 元素中的内容会对应 **DataSet** 中的每条 "record" 重复出现，最后，**<FooterTemplate>** 元素中的内容在输出中仅出现一次：

```
<html>
<body>

<form runat="server">
<asp:DataList id="cdcatalog" runat="server">

<HeaderTemplate>
...
</HeaderTemplate>
```



```
<ItemTemplate>
...
</ItemTemplate>

<FooterTemplate>
...
</FooterTemplate>

</asp:DataList>
</form>

</body>
</html>
```

然后我们添加创建 **DataSet** 的脚本，并且绑定 **mycdcatalog DataSet** 到 **DataList** 控件。然后 使用包含表头的 **<HeaderTemplate>**、包含要显示的数
据项的 **<ItemTemplate>** 和包含文本的 **<FooterTemplate>** 来填充 **DataList** 控件。请注意，可设置 **DataList** 的 **gridlines** 属性为 **"both"** 来显示表格边
框：

实例

```
<%@ Import Namespace="System.Data" %>

<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
dim mycdcatalog=New DataSet
mycdcatalog.ReadXml(MapPath("cdcatalog.xml"))
cdcatalog.DataSource=mycdcatalog
cdcatalog.DataBind()
end if
end sub
</script>

<html>
<body>

<form runat="server">
<asp:DataList id="cdcatalog"
gridlines="both" runat="server">

<HeaderTemplate>
My CD Catalog
</HeaderTemplate>

<ItemTemplate>
"<#Container.DataItem("title")%>" of
<#Container.DataItem("artist")%> -
$<#Container.DataItem("price")%>
</ItemTemplate>

<FooterTemplate>
Copyright Hege Refsnes
</FooterTemplate>

</asp:DataList>
</form>

</body>
</html>
```

演示实例 »

使用样式

您也可以向 **DataList** 控件添加样式，让输出更加花哨：

实例

```
<%@ Import Namespace="System.Data" %>

<script runat="server">
```

```

sub Page_Load
if Not Page.IsPostBack then
dim mycdcatalog=New DataSet
mycdcatalog.ReadXml(MapPath("cdcatalog.xml"))
cdcatalog.DataSource=mycdcatalog
cdcatalog.DataBind()
end if
end sub
</script>

```

```

<html>
<body>

<form runat="server">
<asp:DataList id="cdcatalog"
runat="server"
cellpadding="2"
cellspacing="2"
borderstyle="inset"
backcolor="#e8e8e8"
width="100%"
headerstyle-font-name="Verdana"
headerstyle-font-size="12pt"
headerstyle-horizontalalign="center"
headerstyle-font-bold="true"
itemstyle-backcolor="#778899"
itemstyle-forecolor="#ffffff"
footerstyle-font-size="9pt"
footerstyle-font-italic="true">

<HeaderTemplate>
My CD Catalog
</HeaderTemplate>

<ItemTemplate>
"<%=Container.DataItem("title")%>" of
<%=Container.DataItem("artist")%> -
$<%=Container.DataItem("price")%>
</ItemTemplate>

<FooterTemplate>
Copyright Hege Refsnes
</FooterTemplate>

</asp:DataList>
</form>

</body>
</html>

```

演示实例 »

使用 <AlternatingItemTemplate>

您可以在 <ItemTemplate> 元素后添加 <AlternatingItemTemplate> 元素，用来描述输出中交替行的外观。您可以在 DataList 控件内部对 <AlternatingItemTemplate> 区域的数据添加样式：

实例

```

<%@ Import Namespace="System.Data" %>

<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
dim mycdcatalog=New DataSet
mycdcatalog.ReadXml(MapPath("cdcatalog.xml"))
cdcatalog.DataSource=mycdcatalog
cdcatalog.DataBind()
end if
end sub
</script>

```

```
<html>
<body>

<form runat="server">
<asp:DataList id="cdcatalog"
runat="server"
cellpadding="2"
cellspacing="2"
borderstyle="inset"
backcolor="#e8e8e8"
width="100%"
headerstyle-font-name="Verdana"
headerstyle-font-size="12pt"
headerstyle-horizontalalign="center"
headerstyle-font-bold="True"
itemstyle-backcolor="#778899"
itemstyle-forecolor="ffffff"
alternatingitemstyle-backcolor="#e8e8e8"
alternatingitemstyle-forecolor="000000"
footerstyle-font-size="9pt"
footerstyle-font-italic="True">

<HeaderTemplate>
My CD Catalog
</HeaderTemplate>

<ItemTemplate>
"<#Container.DataItem("title")%" of
<#Container.DataItem("artist")%> -
$<#Container.DataItem("price")%>
</ItemTemplate>

<AlternatingItemTemplate>
"<#Container.DataItem("title")%" of
<#Container.DataItem("artist")%> -
$<#Container.DataItem("price")%>
</AlternatingItemTemplate>

<FooterTemplate>
&copy; Hege Refsnes
</FooterTemplate>

</asp:DataList>
</form>

</body>
</html>
```

[演示实例 »](#)

[ASP.NET Repeater 控件](#)

[ASP.NET 数据库连接](#)

[点我分享笔记](#)

[反馈/建议](#)

ASP.NET Web Forms - 数据库连接

ADO.NET 也是 .NET 框架的组成部分。ADO.NET 用于处理数据访问。通过 ADO.NET，您可以操作数据库。

[尝试一下 - 实例](#)

[数据库连接 - 绑定到 DataList 控件](#)

[数据库连接 - 绑定到 Repeater 控件](#)

什么是 ADO.NET？

ADO.NET 是 .NET 框架的组成部分

ADO.NET 由一系列用于处理数据访问的类组成

ADO.NET 完全基于 XML

ADO.NET 没有 Recordset 对象，这一点与 ADO 不同

创建数据库连接

在我们的实例中，我们将使用 Northwind 数据库。

首先，导入 "System.Data.OleDb" 命名空间。我们需要这个命名空间来操作 Microsoft Access 和其他 OLE DB 数据库提供商。我们将在 Page_Load 子例程中创建这个数据库的连接。我们创建一个 dbconn 变量，并为其赋值一个新的 OleDbConnection 类，这个类带有指示 OLE DB 提供商和数据库位置的连接字符串。然后我们打开数据库连接：

```
<%@ Import Namespace="System.Data.OleDb" %>

<script runat="server">
sub Page_Load
dim dbconn
dbconn=New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;
data source=" & server.mappath("northwind.mdb"))
dbconn.Open()
end sub
</script>
```

注释： 这个连接字符串必须是没有折行的连续字符串！

创建数据库命令

为了指定需从数据库取回的记录，我们将创建一个 dbcomm 变量，并为其赋值一个新的 OleDbCommand 类。这个 OleDbCommand 类用于发出针对数据库表的 SQL 查询：

```
<%@ Import Namespace="System.Data.OleDb" %>

<script runat="server">
sub Page_Load
dim dbconn,sql,dbcomm
dbconn=New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;
data source=" & server.mappath("northwind.mdb"))
dbconn.Open()
sql="SELECT * FROM customers"
dbcomm=New OleDbCommand(sql,dbconn)
end sub
</script>
```

创建 DataReader

OleDbDataReader 类用于从数据源中读取记录流。DataReader 是通过调用 OleDbCommand 对象的 ExecuteReader 方法来创建的：

```
<%@ Import Namespace="System.Data.OleDb" %>

<script runat="server">
```

```
sub Page_Load
dim dbconn,sql,dbcomm,dbread
dbconn=New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;
data source=" & server.mappath("northwind.mdb"))
dbconn.Open()
sql="SELECT * FROM customers"
dbcomm=New OleDbCommand(sql,dbconn)
dbread=dbcomm.ExecuteReader()
end sub
</script>
```

绑定到 Repeater 控件

然后，我们绑定 `DataReader` 到 `Repeater` 控件：

实例

```
<%@ Import Namespace="System.Data.OleDb" %>

<script runat="server">
sub Page_Load
dim dbconn,sql,dbcomm,dbread
dbconn=New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;
data source=" & server.mappath("northwind.mdb"))
dbconn.Open()
sql="SELECT * FROM customers"
dbcomm=New OleDbCommand(sql,dbconn)
dbread=dbcomm.ExecuteReader()
customers.DataSource=dbread
customers.DataBind()
dbread.Close()
dbconn.Close()
end sub
</script>

<html>
<body>

<form runat="server">
<asp:Repeater id="customers" runat="server">

<HeaderTemplate>
<table border="1" width="100%">
<tr>
<th>Companyname</th>
<th>Contactname</th>
<th>Address</th>
<th>City</th>
</tr>
</HeaderTemplate>

<ItemTemplate>
<tr>
<td><%#Container.DataItem("companyname")%></td>
<td><%#Container.DataItem("contactname")%></td>
<td><%#Container.DataItem("address")%></td>
<td><%#Container.DataItem("city")%></td>
</tr>
</ItemTemplate>

<FooterTemplate>
</table>
</FooterTemplate>

</asp:Repeater>
</form>

</body>
</html>
```

演示实例 »

关闭数据库连接

如果不再需要访问数据库，请记得关闭 **DataReader** 和数据库连接：

```
dbread.Close()  
dbconn.Close()
```

[❏ 点我分享笔记](#)

反馈/建议



ASP.NET Web Forms - 母版页

母版页为您的网站的其他页面提供模版。

母版页

母版页允许您为您的 **web** 应用程序中的所有页面（或页面组）创建一致的外观和行为。

母版页为其他页面提供模版，带有共享的布局和功能。母版页为内容定义了可被内容页覆盖的占位符。输出结果是母版页和内容页的组合。

内容页包含您想要显示的内容。

当用户请求内容页时，**ASP.NET** 会对页面进行合并以生成结合了母版页布局和内容页内容的输出。

母版页实例

```
<%@ Master %>  
  
<html>  
<body>  
<h1>Standard Header From Masterpage</h1>  
<asp:ContentPlaceHolder id="CPH1" runat="server">  
</asp:ContentPlaceHolder>  
</body>  
</html>
```

上面的母版页是一个为其他页面设计的普通 **HTML** 模版页。

@ Master 指令定义它为一个母版页。

母版页为单独的内容包含占位标签 **<asp:ContentPlaceHolder>**。

id="CPH1" 属性标识占位符，在相同母版页中允许多个占位符。

这个母版页被保存为 **"master1.master"**。

☐ 注释：母版页也能够包含代码，允许动态的内容。

内容页实例

```
<%@ Page MasterPageFile="master1.master" %>  
  
<asp:Content ContentPlaceHolderId="CPH1" runat="server">  
<h2>Individual Content</h2>
```

```
<p>Paragraph 1</p>
<p>Paragraph 2</p>
</asp:Content>
```

上面的内容页是站点中独立的内容页中的一个。

@ Page 指令定义它为一个标准的内容页。

内容页包含内容标签 **<asp:Content>**，该标签引用了母版页（ContentPlaceHolderId="CPH1"）。

这个内容页被保存为 **"mypage1.aspx"**。

当用户请求该页面时，ASP.NET 就会将母版页与内容页进行合并。

[点击这里显示 mypage1.aspx](#)

☐ 注释：内容文本必须位于 **<asp:Content>** 标签内部。标签外的内容文本是不允许的。

带控件的内容页

```
<%@ Page MasterPageFile="master1.master" %>

<asp:Content ContentPlaceHolderId="CPH1" runat="server">
<h2>RUNOOB</h2>
<form runat="server">
<asp:TextBox id="textbox1" runat="server" />
<asp:Button id="button1" runat="server" text="Button" />
</form>
</asp:Content>
```

上面的内容页演示了如何把 .NET 控件插入内容页，就像插入一个普通的页面中。

[点击这里显示 mypage2.aspx](#)

☐ ASP.NET 数据库连接

ASP.NET 导航 ☐

☐ 点我分享笔记

反馈/建议

Copyright © 2013-2018 菜鸟教程 [runoob.com](#) All Rights Reserved. 备案号：闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

☐ ASP.NET 母版页

ASP.NET 实例 ☐

ASP.NET Web Forms - 导航

ASP.NET 带有内建的导航控件。

网站导航

维护大型网站的菜单是困难而且费时的。

在 ASP.NET 中，菜单可存储的文件中，这样易于维护。文件通常名为 **web.sitemap**，并且被存放在网站的根目录下。

此外，ASP.NET 有三个心的导航控件：

Dynamic menus

TreeViews

Site Map Path


Sitemap 文件

在本教程中，使用下面的 **sitemap** 文件：

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<siteMap>
<siteMapNode title="Home" url="/aspnet/w3home.aspx">
<siteMapNode title="Services" url="/aspnet/w3services.aspx">
<siteMapNode title="Training" url="/aspnet/w3training.aspx"/>
<siteMapNode title="Support" url="/aspnet/w3support.aspx"/>
</siteMapNode>
</siteMapNode>
</siteMap>
```

创建 **sitemap** 文件的规则：

- XML 文件必须包含 围绕内容的 **<siteMap>** 标签
- <siteMap>** 标签只能有一个 **<siteMapNode>** 子节点（"home" 页面）
- 每个 **<siteMapNode>** 可以有多个子节点（网页）
- 每个 **<siteMapNode>** 带有定义页面标题和 URL 的属性

 **注释：** **sitemap** 文件必须位于站点根目录下，URL 属性必须相对于该根目录。

动态菜单

<asp:Menu> 控件可显示标准的网站导航菜单。

代码实例：

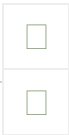
```
<asp:SiteMapDataSource id="nav1" runat="server" />

<form runat="server">
<asp:Menu runat="server" DataSourceId="nav1" />
</form>
```

上面实例中的 **<asp:Menu>** 控件是一个供服务器创建导航菜单的占位符。

控件的数据源由 **DataSourceId** 属性定义。**id="nav1"** 把数据源连接到 **<asp:SiteMapDataSource>** 控件。

<asp:SiteMapDataSource> 控件自动连接默认的 **sitemap** 文件（**web.sitemap**）。



TreeView

<asp:TreeView> 控件可显示多级导航菜单。

这种菜单看上去像一棵带有枝叶的树，可通过 **+** 或 **-** 符号来打开或关闭。

代码实例：

```
<asp:SiteMapDataSource id="nav1" runat="server" />

<form runat="server">
<asp:TreeView runat="server" DataSourceId="nav1" />
</form>
```

上面实例中的 **<asp:TreeView>** 控件是一个供服务器创建导航菜单的占位符。


控件的数据源由 **DataSourceId** 属性定义。**id="nav1"** 把数据源连接到 **<asp:SiteMapDataSource>** 控件。

<asp:SiteMapDataSource> 控件自动连接默认的 **sitemap** 文件（**web.sitemap**）。

SiteMapPath

SiteMapPath 控件可显示指向当前页面的指针（导航路径）。该路径显示为指向上级页面的可点击链接。

与 **TreeView** 和 **Menu** 控件不同，**SiteMapPath** 控件不使用 **SiteMapDataSource**。**SiteMapPath** 控件默认使用 **web.sitemap** 文件。

 **提示：** 如果 **SiteMapPath** 没有正确显示，很可能是由于 **web.sitemap** 文件中存在 URL 错误（打印错误）。

代码实例：

```
<form runat="server">
<asp:SiteMapPath runat="server" />
</form>
```

上面实例中的 **<asp:SiteMapPath>** 控件是一个供服务器创建导航菜单的占位符。



ASP.NET Web Forms - 实例

ASP.NET HTML 控件

[HTML Anchor](#)

[HTML Button](#)

[HTML Image](#)

[HTML Image 2](#)

[HTML Inputbutton](#)

[HTML InputCheckbox](#)

[HTML InputHidden](#)

[HTML InputImage](#)

[HTML InputRadiobutton](#)

[HTML Table](#)

[HTML Table 2](#)

[HTML Textarea](#)

ASP.NET Web 控件

[AdRotator](#)

[Button](#)

[Button 2](#)

[Calendar](#)

[Calendar 2](#)

[Calendar 3](#)

[Checkbox](#)

[CheckboxList](#)

[DataList](#)

[用 styles的DataList](#)

[用 <AlternatingItemTemplate>的DataList](#)

[DropDownList](#)

[Hyperlink](#)

[Image](#)

[ImageButton](#)

[Label](#)

[LinkButton](#)

[Listbox](#)

[Literal](#)

[Literal 2](#)

[Panel](#)

[Radiobutton](#)

[RadiobuttonList](#)

[Repeater](#)

[用 <AlternatingItemTemplate> 重复](#)

[用 <SeparatorTemplate>重复](#)

[Table](#)

[Table 2](#)

[Textbox](#)

[Textbox 2](#)

[Textbox 3](#)

[XML](#)

ASP.NET Validation 控件

[CompareValidator](#)

[CompareValidator 2](#)

[CustomValidator](#)

[RangeValidator](#)

[RangeValidator 2](#)

[RegularExpressionValidator](#)

[RequiredFieldValidator](#)

[Validationsummary](#)

[Validationsummary 2](#)

ASP.NET 事件

[Page_Load](#)

[Page.IsPostBack](#)

ASP.NET 数据绑定

[ArrayList RadioButtonList](#)

[ArrayList DropDownList](#)

[Hashtable RadioButtonList 1](#)

[Hashtable RadiobuttonList 2](#)

[Hashtable DropDownList](#)

[SortedList RadioButtonList 1](#)

[SortedList RadiobuttonList 2](#)

[SortedList DropDownList](#)

[XML RadiobuttonList](#)

ASP.NET 数据库

[数据库链接 - 绑定一个 Repeater 控件](#)

[数据库链接 - 绑定一个 DataList 控件](#)

[☐ ASP.NET 导航](#)

ASP.NET HtmlAnchor 控件 [☐](#)

[☐ 点我分享笔记](#)

[反馈/建议](#)



ASP.NET Web Forms - HTML 服务器控件

HTML 服务器控件是服务器可理解的 HTML 标签。

HTML 服务器控件

ASP.NET 文件中的 HTML 元素，默认是作为文本进行处理的。要想让这些元素可编程，需向 HTML 元素中添加 `runat="server"` 属性。这个属性表示，该元素将被作为服务器控件进行处理。

注释：所有 HTML 服务器控件必须位于带有 `runat="server"` 属性的 `<form>` 标签内！

注释：ASP.NET 要求所有 HTML 元素必须正确关闭和正确嵌套。

HTML 服务器控件	描述
HtmlAnchor	控制 <code><a></code> HTML 元素
HtmlButton	控制 <code><button></code> HTML 元素
HtmlForm	控制 <code><form></code> HTML 元素
HtmlGeneric	控制其他未被具体的 HTML 服务器控件规定的 HTML 元素，比如 <code><body></code> 、 <code><div></code> 、 <code></code> 等。
HtmlImage	控制 <code><image></code> HTML 元素
HtmlInputButton	控制 <code><input type="button"></code> 、 <code><input type="submit"></code> 和 <code><input type="reset"></code> HTML 元素
HtmlInputCheckBox	控制 <code><input type="checkbox"></code> HTML 元素
HtmlInputFile	控制 <code><input type="file"></code> HTML 元素
HtmlInputHidden	控制 <code><input type="hidden"></code> HTML 元素
HtmlInputImage	控制 <code><input type="image"></code> HTML 元素
HtmlInputRadioButton	控制 <code><input type="radio"></code> HTML 元素
HtmlInputText	控制 <code><input type="text"></code> 和 <code><input type="password"></code> HTML 元素
HtmlSelect	控制 <code><select></code> HTML 元素
HtmlTable	控制 <code><table></code> HTML 元素
HtmlTableCell	控制 <code><td></code> 和 <code><th></code> HTML 元素
HtmlTableRow	控制 <code><tr></code> HTML 元素
HtmlTextArea	控制 <code><textarea></code> HTML 元素



ASP.NET Web Forms - Web 服务器控件

Web 服务器控件是服务器可理解的特殊 ASP.NET 标签。

Web 服务器控件

就像 HTML 服务器控件，Web 服务器控件也是在服务器上创建的，它们同样需要 `runat="server"` 属性才能生效。然而，Web 服务器控件没有必要映射任何已存在的 HTML 元素，它们可以表示更复杂的元素。

创建 Web 服务器控件的语法是：

```
<asp:control_name id="some_id" runat="server" />
```

Web 服务器控件	描述
AdRotator	显示一个图形序列
Button	显示下压按钮
Calendar	显示日历
CalendarDay	calendar 控件中的一天
CheckBox	显示复选框
CheckBoxList	创建多选的复选框组
DataGrid	显示 grid 中数据源的字段
DataList	通过使用模版显示数据源中的项目
DropDownList	创建下拉列表
HyperLink	创建超链接
Image	显示图像
ImageButton	显示可点击的图像
Label	显示可编程的静态内容（使您对其内容应用样式）
LinkButton	创建超链接按钮
ListBox	创建单选或多选的下拉列表

ListItem	创建列表中的一个项目
Literal	显示可编程的静态内容（无法使您对其内容应用样式）
Panel	为其他控件提供容器
PlaceHolder	为由代码添加的控件预留空间
RadioButton	创建单选按钮
RadioButtonList	创建单选按钮组
BulletedList	创建项目符号格式的列表
Repeater	显示绑定到控件的项目的重复列表
Style	设置控件的样式
Table	创建表格
TableCell	创建表格单元格
TableRow	创建表格行
TextBox	创建文本框
Xml	显示 XML 文件或 XSL 转换的结果

[❏ ASP.NET XML 控件](#)

[ASP.NET CompareValidator 控件](#) ❏

[❏ 点我分享笔记](#)

[反馈/建议](#)

Copyright © 2013-2018 菜鸟教程 [runoob.com](#) All Rights Reserved. 备案号：闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

[❏ ASP.NET ValidationSummary 控件](#)

[WebSecurity CurrentUserId 属性](#) ❏

ASP.NET Web Forms - Validation 服务器控件

Validation 服务器控件是用来验证用户输入的。

Validation 服务器控件

Validation 服务器控件用于验证输入控件的数据。如果数据未通过验证，则向用户显示错误消息。

创建 Validation 服务器控件的语法是：

```
<asp:control_name id="some_id" runat="server" />
```

Validation 服务器控件	描述
CompareValidator	把一个输入控件的值与另一个输入控件的值或一个固定的值进行对比
CustomValidator	允许您编写一个方法，来处理输入值的验证
RangeValidator	检查用户输入值是否介于两个值之间
RegularExpressionValidator	确保输入控件的值匹配指定的模式
RequiredFieldValidator	使输入控件成为必需（必填）的字段
ValidationSummary	显示网页中所有验证错误的报告

☐ ASP.NET ValidationSummary 控件

WebSecurity CurrentUserId 属性 ☐

☐ 点我分享笔记

反馈/建议