



# ASP.NET MVC随想录

---

极客学院出版

# 前言

---

ASP.NET MVC 框架提供了一个可以代替 ASP.NET WebForm 的基于 MVC 设计模式的应用。本书主要讲解 ASP.NET MVC 中新增加的那些功能。

## | 适用人群

适用于使用 MVC 框架开发的程序员的进阶学习。

## | 学习前提

学习本教程前，你需要了解 ASP.NET 这门语言。

鸣谢：<http://www.cnblogs.com/OceanEyes>

# 目录

---

前言 .....	1
第 1 章 作者简介 .....	5
版权说明 .....	7
第 2 章 ASP.NET MVC 随想录 (1) ——开始使用 Bootstrap .....	8
Bootstrap 结构介绍 .....	10
在 ASP.NET MVC 项目中添加 Bootstrap 文件 .....	11
为网站创建 Layout 布局页 .....	14
使用捆绑打包和压缩来提升网站性能 .....	16
在 Bootstrap 项目中使用捆绑打包 .....	17
测试打包和压缩 .....	19
小结 .....	20
第 3 章 ASP.NET MVC 随想录 (2) ——使用 Bootstrap CSS 和 HTML 元素 .....	21
Bootstrap 栅格 (Grid) 系统 .....	23
Bootstrap HTML 元素 .....	25
Bootstrap 验证样式 .....	35
ASP.NET MVC 创建包含 Bootstrap 样式编辑模板 .....	37
小结 .....	20
第 4 章 ASP.NET MVC 随想录 (3) ——使用 Bootstrap 组件 .....	40
Bootstrap 导航条 .....	42
列表组 .....	46
徽章 .....	47
媒体对象 .....	48
页头 .....	50
路径导航 .....	51

	分页 .....	52
	输入框组 .....	53
	按钮式下拉菜单 .....	55
	警告框 .....	56
	进度条 .....	58
	小结 .....	20
第 5 章	ASP.NET MVC 使用 Bootstrap 系列 (4) ——使用 JavaScript 插件 .....	61
	Data 属性 VS 编程 API .....	63
	下拉菜单 (dropdown.js) .....	64
	模态框 (modal.js) .....	65
	标签页 (tab.js) .....	67
	工具提示 (tooltip.js) .....	68
	弹出框 (popover.js) .....	69
	手风琴组件 (collapse.js) .....	70
	旋转木马组件 (carousel.js) .....	72
	小结 .....	20
第 6 章	ASP.NET MVC 随想录 (5) ——创建 ASP.NET MVC Bootstrap Helpers .....	75
	内置的HTML Helpers .....	77
	创建自定义的 Helpers .....	78
	使用静态方法创建 Helpers .....	79
	使用扩展方法创建Helpers .....	81
	创建 Fluent Helpers .....	82
	创建自动闭合的 Helpers .....	85
	小结 .....	20
第 7 章	ASP.NET MVC 随想录 (6) ——漫谈 OWIN .....	88
	什么是 OWIN .....	89
	为什么我们需要 OWIN .....	90

	OWIN 的规范 .....	93
	小结 .....	20
第 8 章	ASP.NET MVC 随想录 (7) ——锋利的 KATANA .....	96
	ASP.NET 发展历程 .....	98
	走进Katana的世界 .....	100
第 9 章	ASP.NET MVC 随想录 (8) ——创建自定义的 Middleware 中间件 .....	112
	何为 Middleware 中间件 .....	114
	使用 Inline 方式注册 Middleware .....	115
	使用 Inline+ AppFunc 方式注册 Middleware .....	116
	定义原生 Middleware 类的形式来注册 Middleware .....	118
	使用Katana Helper来注册Middleware .....	119
	Middleware 的执行顺序 .....	120
	小结 .....	20



作者简介



王滨，网名木宛城主，擅长.NET平台下各种技术，Microsoft MVP。

博客地址：<http://www.cnblogs.com/OceanEyes>

ASP.NET MVC 自从问世以来，极大的吸引了开发者。随着技术的更新，越来越多的新功能被加到了ASP.NET MVC中。基于此，我想写一套教程，面向广大的开发人员，讲解 ASP.NET MVC 中各个功能，故该系列名为 ASP.NET MVC 随想录。

## 版权说明

---

本书版权是归作者王滨所有，未经作者允许，不得转载。



2

# ASP.NET MVC 随想录（1）——开始使用 Bootstrap

作为一名 Web 开发者而言，如果不借助任何前端框架，从零开始使用 HTML 和 CSS 来构建友好的页面是非常困难的。特别是对于 Windows Form 的开发者而言，更是难上加难。正是由于这样的原因，Bootstrap 诞生了。Twitter Bootstrap 为开发者提供了丰富的 CSS 样式、组件、插件、响应式布局等。同时微软已经完全集成在 ASP.NET MVC 模板中。

## Bootstrap 结构介绍

---

你可以通过[来下载](#)最新版本的 Bootstrap。

解压文件夹后，可以看到 Bootstrap 的文件分布结构如下，包含 3 个文件夹：

css 文件夹中包含了 4 个 .css 文件和 2 个 .map 文件。我们只需要将 bootstrap.css 文件包含到项目里这样就能将 Bootstrap 应用到我们的页面中了。bootstrap.min.css 即为上述 css 的压缩版本。

.map 文件不必包含到项目里，你可以将其忽略。这些文件被用来作为调试符号（类似于 Visual Studio 中的 .pdb 文件），最终能让开发人员在线编辑预处理文件。

Bootstrap 使用 Font Awesome（一个字体文件包含了所有的字形图标，只为 Bootstrap 设计）来显示不同的图标和符号，fonts 文件夹包含了 4 类的不同格式的字体文件：

- Embedded OpenType (glyphicons-halflings-regular.eot)
- Scalable Vector Graphics (glyphicons-halflings-regular.svg)
- TrueType font (glyphicons-halflings-regular.ttf)
- Web Open Font Format (glyphicons-halflings-regular.woff)

建议将所有的字体文件包含在你的 Web 应用程序中，因为这能让你的站点在不同的浏览器中显示正确的字体。

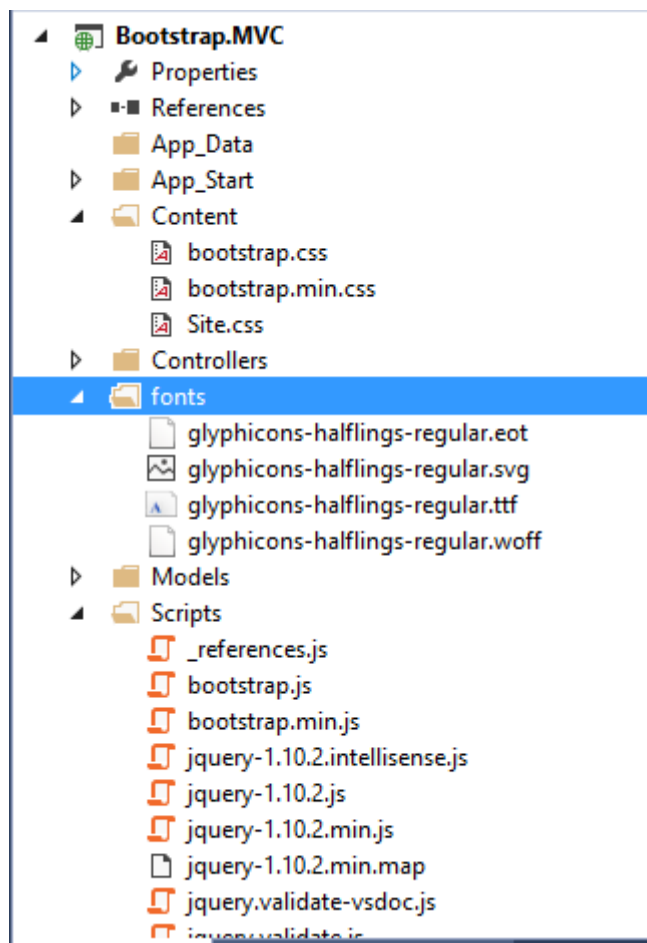
EOT 字体格式文件需要 IE9 及以上浏览器支持，TTF 是传统的旧字体格式文件，WOFF 是从 TTF 中压缩得到的字体格式文件。如果你只需要支持 IE8 之后的浏览器、iOS 4 以上版本、同时支持 Android，那么你只需要包含 WOFF 字体即可。

js 文件夹包含了 3 个文件，所有的 Bootstrap 插件被包含在 bootstrap.js 文件中，bootstrap.min.js 即上述 js 的压缩版本，npm.js 通过项目构建工具 Grunt 自动生成。

在引用 bootstrap.js 文件之前，请确保你已经引用了 JQuery 库因为所有的 Bootstrap 插件需要 JQuery。

## 在 ASP.NET MVC 项目中添加 Bootstrap 文件

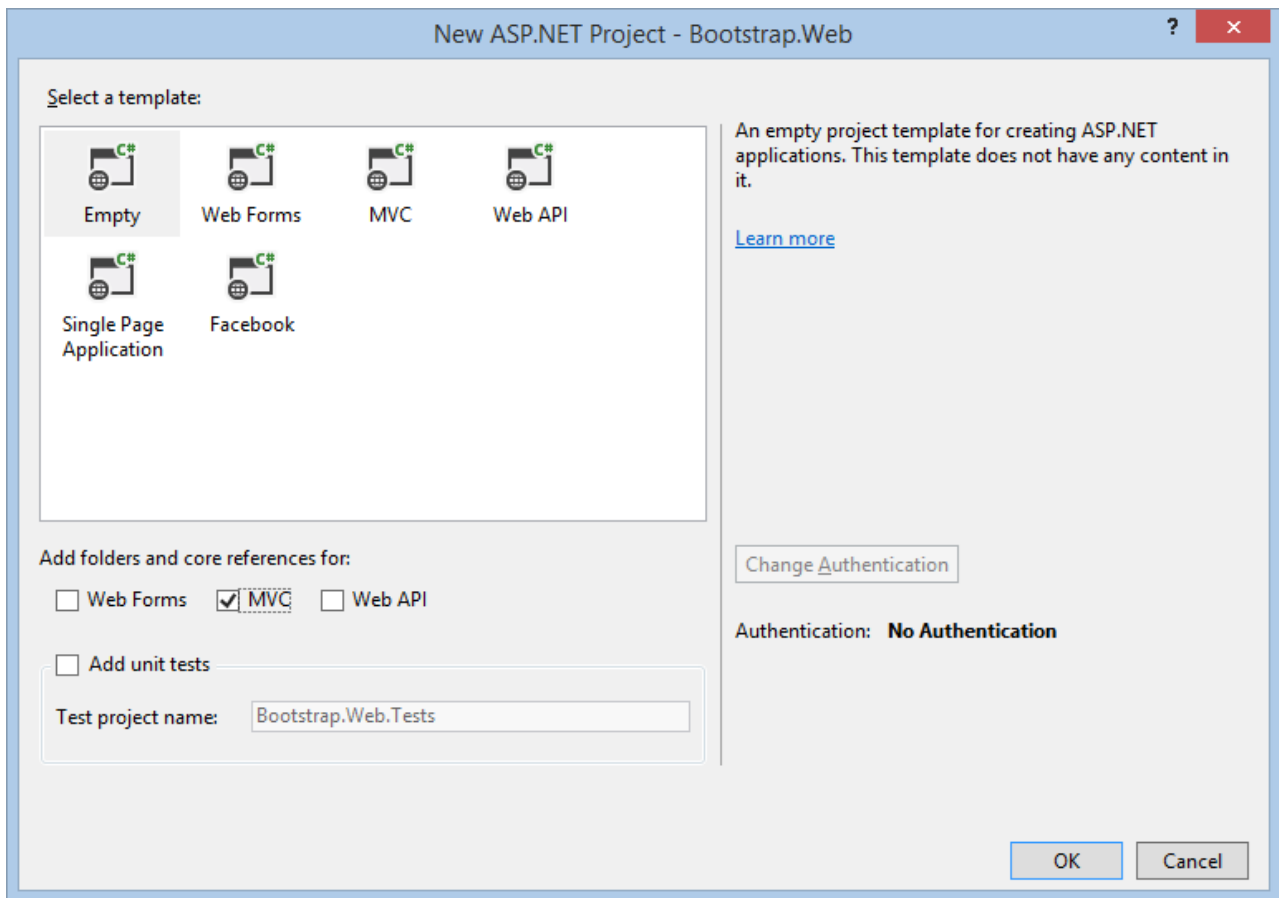
打开 Visual Studio 2013，创建标准的 ASP.NET MVC 项目，默认情况下已经自动添加了 Bootstrap 的所有文件，如下所示：



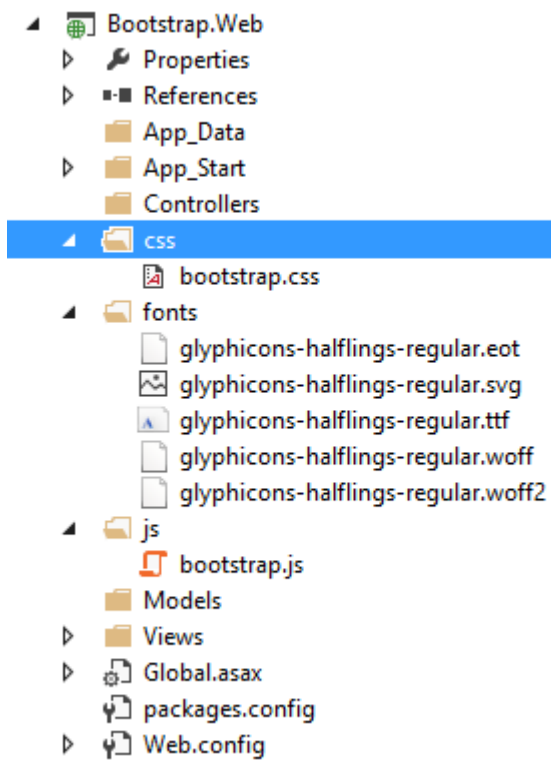
说明微软对于 Bootstrap 是非常认可的，高度集成在 Visual Studio 中。

值得注意的是，在 Scripts 文件中添加了一个名为 `_references.js` 的文件，这是一个非常有用的功能，当我们在使用 Bootstrap 等一些前端库时，它可以帮助 Visual Studio 启用智能提示。

当然我们也可以创建一个空的 ASP.NET MVC 项目手动去添加这些依赖文件，正如下图所示这样，选择空的模板：



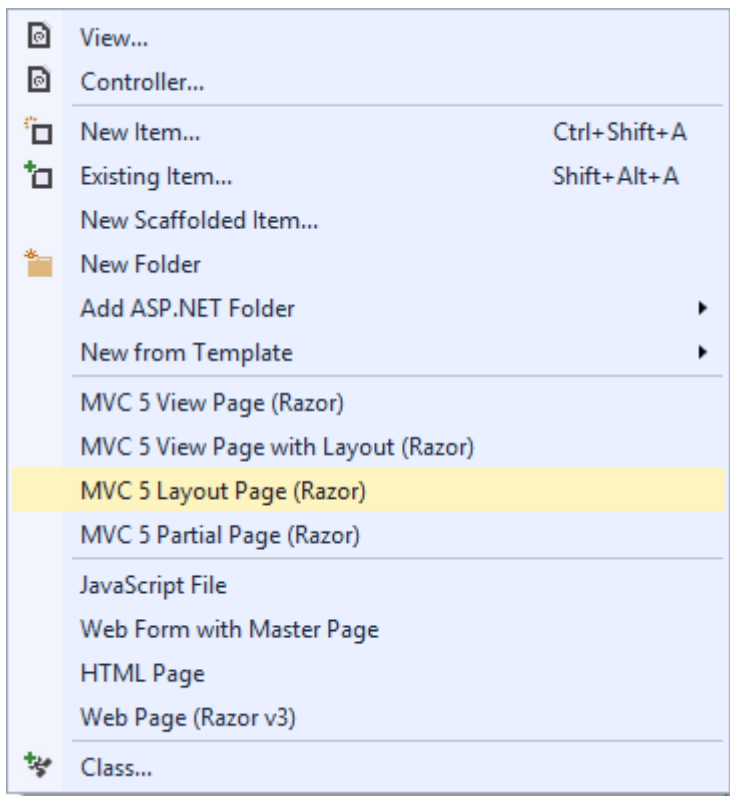
对于新创建的空白 ASP.NET MVC 项目来说, 没用 Content, Fonts, Scripts 文件夹——我们必须手动去创建他们, 如下所示:



当然，也可以用 Nuget 来自动添加 Bootstrap 资源文件。如果使用图形界面来添加 Bootstrap Nuget Package，则直接搜索 Bootstrap 即可；如果使用 Package Manager Console 来添加 Bootstrap Nuget Package，则输入 `Install-Package bootstrap`。

## 为网站创建 Layout 布局页

为了让我们的网站保持一致的风格，我将使用 Bootstrap 来构建 Layout 布局页。在 Views 文件夹创建 MVC Layout Page(Razor)布局文件，如下图所示：



在新创建的 Layout 布局页中，使用如下代码来引用 Bootstrap 资源文件。

```
<link href="@Url.Content("~/css/bootstrap.css")" rel="stylesheet">
<script src="@Url.Content("~/js/bootstrap.js")"></script>
```

其中使用 `@Url.Content` 会将虚拟或者相对路径转换为绝对路径，这样确保 Bootstrap 资源文件被引用。

新建一个名为 Home 的 Controller，并且添加默认 Index 的视图，使其套用上述的 Layout 布局页面，如下所示：

Add View

View name:

Index

Template:

Empty (without model)

Model class:

View options:

☐ Create as a partial view

☐ Reference script libraries

☒ Use a layout page:

~/Views/\_Layout.cshtml

...

(Leave empty if it is set in a Razor \_viewstart file)

Add

Cancel



## 使用捆绑打包和压缩来提升网站性能

---

捆绑打包(bundling)和压缩(minification)是 ASP.NET 中的一项新功能，允许你提升网站加载速度，这是通过限制请求 CSS 和 JavaScript 文件的次数来完成的。本质上是将这类文件结合到一个大文件以及删除所有不必要的字符（比如：注释、空格、换行）。

对于大多数现代浏览器访问一个主机名都有 6 个并发连接的极限，这意味着如果你在一张页面上引用了 6 个以上的 CSS、JavaScript 文件，浏览器一次只会下载 6 个文件。所以限制资源文件的个数是个好办法，真正意义上的使命必达，而不是浪费在加载资源上。

## 在 Bootstrap 项目中使用捆绑打包

---

因为我们创建的是空的 ASP.NET MVC 项目，所以并没有自动引用与打包相关的程序集。打开 Nuget Package Manager Console 来完成对 Package 的安装，使用如下 PowerShell 命令：

install-package Microsoft.AspNet.Web.Optimization 来安装 Microsoft.AspNet.Web.Optimization NuGet package 以及它依赖的 Package，如下所示：

```
PM> install-package Microsoft.AspNet.Web.Optimization
Attempting to resolve dependency 'Microsoft.Web.Infrastructure (≥ 1.0.0)'.
Attempting to resolve dependency 'WebGrease (≥ 1.5.2)'.
Attempting to resolve dependency 'Antlr (≥ 3.4.1.9004)'.
Attempting to resolve dependency 'Newtonsoft.Json (≥ 5.0.4)'.
Installing 'Microsoft.AspNet.Web.Optimization 1.1.3'.
```

在安装完成后，在 App\_Start 中添加 BundleConfig 类：

```
public static void RegisterBundles(BundleCollection bundles)
{
    bundles.Add(new ScriptBundle("~/bootstrap/js").Include(
        "~/js/bootstrap.js",
        "~/js/site.js"));
    bundles.Add(new StyleBundle("~/bootstrap/css").Include(
        "~/css/bootstrap.css",
        "~/css/site.css"));
}
```

ScriptBundle 和 StyleBundle 对象实例化时接受一个参数用来代表打包文件的虚拟路径，Include 顾名思义将需要的文件包含到其中。

然后在 Application\_Start 方法中注册它：

```
protected void Application_Start()
{
    AreaRegistration.RegisterAllAreas();
    RouteConfig.RegisterRoutes(RouteTable.Routes);
    BundleConfig.RegisterBundles(BundleTable.Bundles);
    BundleTable.EnableOptimizations = true;
}
```

记住，不要去包含 .min 类型的文件到打包文件中，比如 bootstrap.min.css、bootstrap.min.js，编译器会忽略这些文件因为他们已经被压缩过了。

在 ASP.NET MVC 布局页使用 `@Styles.Render("~/bootstrap/css")`、`@Scripts.Render("~/bootstrap/js")` 来添加对打包文件的引用。

如果 Visual Studio HTML 编辑器表明无法找到 Styles 和 Scripts 对象，那就意味着你缺少了命名空间的引用，你可以手动在布局页的顶部添加 `System.Web.Optimization` 命名空间，如下代码所示：

```
@using System.Web.Optimization
<!DOCTYPE html>
<html>
<head>
  <meta name="viewport" content="width=device-width" />
  <title>@ViewBag.Title</title>
  @Scripts.Render("~/bootstrap/js")
  @Styles.Render("~/bootstrap/css")
</head>
<body>
  <div>
    @*@RenderBody()*@
  </div>
</body>
</html>
```

当然为了通用性，最佳的实践是在 Views 文件夹的 `web.config` 中添加 `System.Web.Optimization` 名称空间的引用，如下所示：

```
<namespaces>
  <add namespace="System.Web.Mvc" />
  <add namespace="System.Web.Mvc.Ajax" />
  <add namespace="System.Web.Mvc.Html" />
  <add namespace="System.Web.Routing" />
  <add namespace="Bootstrap.Web" />
  <add namespace="System.Web.Optimization" />
</namespaces>
```

## 测试打包和压缩

---

为了使用打包和压缩，打开网站根目录下的 web.config 文件，并且更改 compilation 元素的 debug 属性为 false，即为 release。

当然你可以在 Application\_Start 方法中设置 BundleTable.EnableOptimizations = true 来同样达到上述效果（它会 override web.config 中的设置，即使 debug 属性为 true）。

最后浏览网页，查看源代码，可以清楚看到打包文件的路径是之前定义过的相对路径，点击这个链接，浏览器为我们打开了经过压缩处理过后的打包文件，如下图所示：

```
<!DOCTYPE html>

<html>
<head>
  <meta name="viewport" content="width=device-width" />
  <title>Index</title>
  <script src="/bootstrap/js?v=8a235NsDroe-Dh7OCFItn8uU0BFGmrGeSV55pNz8C1g1"></script>

  <link href="/bootstrap/css?v=rhbbWUaC6ikiRRqvQHmylCq7ouapevLIktyEy76p8vU1" rel="stylesheet"/>
</head>
<body>
  <div>

<h2>Index</h2>

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

## 小结

---

在这一章节中，简单为大家梳理了 Bootstrap 的体系结构，然后怎样在 ASP.NET MVC 项目中添加 Bootstrap，最后使用了打包和压缩技术来实现对资源文件的打包，从而提高了网站的性能。

3

## ASP.NET MVC 随想录（2）——使用 Bootstrap CSS 和 HTML 元素

Bootstrap 提供了一套丰富 CSS 设置、HTML 元素以及高级的栅格系统来帮助开发人员快速布局网页。所有的 CSS 样式和 HTML 元素与移动设备优先的流式栅格系统结合，能让开发人员快速轻松的构建直观的界面并且不用担心在较小的设备上响应的具体细节。

# Bootstrap 栅格 ( Grid ) 系统

在移动互联网的今天，越来越多的网站被手机设备访问，移动流量在近几年猛增。Bootstrap 提供了一套响应式、移动设备优先的流式栅格系统，随着屏幕或视口 ( viewport ) 尺寸的增加，系统会自动分为最多 12 列。

## 栅格参数

Bootstrap 3 提供了一系列的预定义 class 来指定列的尺寸，如下所示：

	超小屏幕 手机 (<768px)	小屏幕 平板 (≥768px)	中等屏幕 桌面显示器 (≥992px)	大屏幕 大桌面显示器 (≥1200px)
栅格系统行为	总是水平排列	开始是堆叠在一起的，当大于等于这些阈值时将变为水平排列		
.container 最大宽度	None ( 自动 )	750px	970px	1170px
类前缀	.col-xs-	.col-sm-	.col-md-	.col-lg-
列 ( column ) 数	12			
最大列 ( column ) 宽	自动	~62px	~81px	~97px
槽 ( gutter ) 宽	30px ( 每列左右均有 15px )			
可嵌套	是			
偏移 ( Offsets )	是			
列排序	是			

Bootstrap 栅格系统被分割为 12 列，当布局你的网页时，记住所有列的总和应该是 12。为了图示，请看如下 HTML 所示：

```
<div class="container">
  <div class="row">
    <div class="col-md-3" style="background-color: green;">
      <h3>green</h3>
    </div>
    <div class="col-md-6" style="background-color: red;">
      <h3>red</h3>
    </div>
    <div class="col-md-3" style="background-color: blue;">
      <h3>blue</h3>
    </div>
  </div>
</div>
```

注：Bootstrap 需要为页面内容和栅格系统包裹一个 .container 容器。



在上述代码中，我添加了一个 class 为 container 的 div 容器，并且包含了一个子的 div 元素 row（行）。row div 元素依次有 3 列。其中 2 列包含了 col-md-3 的 class、一列包含了 col-md-6 的 class。当他们组合在一起时，他们加起来总和是 12。但这段 HTML 代码只作用于显示器分辨率  $\geq 992$  的设备。所以为了更好的响应低分辨率的设备，我们需要结合不同的 CSS 栅格 class。故添加对平板、手机、低分辨率的 PC 的支持，需要加入如下 class：

```
<div class="container">
  <div class="row">
    <div class="col-xs-3 col-sm-3 col-md-3" style="background-color: green;">
      <h3>green</h3>
    </div>
    <div class="col-xs-6 col-sm-6 col-md-6" style="background-color: red;">
      <h3>red</h3>
    </div>
    <div class="col-xs-3 col-sm-3 col-md-3" style="background-color: blue;">
      <h3>blue</h3>
    </div>
  </div>
</div>
```

## Bootstrap HTML 元素

---

Bootstrap 已经为我们准备好了一大堆带有样式的 HTML 元素，如：

- Tables
- Buttons
- Forms
- Images

### Bootstrap Tables ( 表格 )

Bootstrap 为 HTML tables 提供了默认的样式和自定义他们布局和行为选项。为了更好的演示，我使用经典的 [Northwind](#) 示例数据库以及如下技术：

- 用 ASP.NET MVC 来作为 Web 应用应用程序
- Bootstrap 前端框架
- Entity Framework 来作为 ORM 框架
- StructureMap 执行我们项目的依赖注入和控制反转，使用 Nuget 来安装
- AutoMapper 自动映射 Domain Model 到 View Model，使用 Nuget 来安装

打开 Visual Studio，创建一个 ASP.NET MVC 的项目，默认情况下，VS 已经为我们添加了 Bootstrap 的文件。为了查看效果，按照如下的步骤去实施：

在 ASP.NET MVC 项目中的 Models 文件下添加一个 ProductViewModel

```
public class ProductViewModel
{
    public int ProductId { get; set; }
    public string ProductName { get; set; }
    public decimal? UnitPrice { get; set; }
    public int? UnitsInStock { get; set; }
    public bool Discontinued { get; set; }
    public string Status { get; set; }
}
```

在 APP\_Data 文件夹中添加 AutoMapperConfig 类, 通过 AutoMapper, 为 ProductViewModel 的 Status 属性创建了一个条件映射, 如果 Product 是 discontinued, 那么 Status 为 danger; 如果 UnitPrice 大于 50, 则设置 Status 属性为 info; 如果 UnitsInStock 小于 20, 那么设置 Status 为 warning。代码的逻辑如下:

```
Mapper.CreateMap<Product, ProductViewModel>()
    .ForMember(dest => dest.Status,
        opt => opt.MapFrom
            (src => src.Discontinued
                ? "danger"
                : src.UnitPrice > 50
                    ? "info"
                    : src.UnitsInStock < 20 ? "warning" : ""));
```

添加一个 ProductController 并且创建名为 Index 的 Action

```
public class ProductController : Controller
{
    //
    // GET: /Product/
    private readonly ApplicationDbContext _context;

    public ProductController(ApplicationDbContext context)
    {
        this._context = context;
    }
    public ActionResult Index()
    {
        var products = _context.Products.Project().To<productviewmodel>().ToArray();
        return View(products);
    }
}
```

上述代码使用依赖注入获取 Entity Framework DbContext 对象, Index Action 接受从数据库中返回 Products 集合然后使用 AutoMapper 映射到每一个 ProductViewModel 对象中, 最后为 View 返回数据。

最后, 在视图上使用 Bootstrap HTML table 来显示数据

```
<div class="container">
<h3>Products</h3>
<table class="table">
  <thead>
    <tr>
      <th>
        Product Name
      </th>
```

```
<th>
    Unit Price
</th>
<th>
    Units In Stock
</th>
<th>
    Discontinued
</th>
</tr>
</thead>
<tbody>
    @foreach (var item in Model)
    {
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.ProductName)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.UnitPrice)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.UnitsInStock)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Discontinued)
            </td>
        </tr>
    }
</tbody>
</table>
</div>
```

呈现的数据如下所示:

## Products

Product Name	Unit Price	Units In Stock	Discontinued
Chai	18.00	35	<input type="checkbox"/>
Chang	19.00	17	<input checked="" type="checkbox"/>
Aniseed Syrup	10.00	27	<input type="checkbox"/>
Chef Anton's Cajun Seasoning	61.00	53	<input type="checkbox"/>
Chef Anton's Gumbo Mix	21.35	89	<input type="checkbox"/>
Grandma's Boysenberry Spread	25.00	120	<input type="checkbox"/>
Uncle Bob's Organic Dried Pears	30.00	6	<input type="checkbox"/>
Northwoods Cranberry Sauce	40.00	21	<input type="checkbox"/>
Mishi Kobe Niku	97.00	29	<input checked="" type="checkbox"/>
Ikura	31.00	31	<input type="checkbox"/>
Queso Cabrales	21.00	22	<input type="checkbox"/>
Queso Manchego La Pastora	38.00	86	<input type="checkbox"/>
Konbu	6.00	24	<input type="checkbox"/>

### Bootstrap Tables 其余样式

Bootstrap 提供了额外的样式来修饰 table。比如使用 `table-bordered` 来显示边框，`table-striped` 显示奇偶数行间颜色不同（斑马条纹状），`table-hover` 顾名思义，当鼠标移动行时高亮，通过添加 `.table-condensed` 类可以让表格更加紧凑，单元格中的内补（padding）均会减半，修改后的代码如下所示：

```
<table class="table table-bordered table-striped table-hover">
</table>
```

显示的效果如下：

## Products

Product Name	Unit Price	Units In Stock	Discontinued
Chai	18.00	35	<input type="checkbox"/>
Chang	19.00	17	<input checked="" type="checkbox"/>
Aniseed Syrup	10.00	27	<input type="checkbox"/>
Chef Anton's Cajun Seasoning	61.00	53	<input type="checkbox"/>
Chef Anton's Gumbo Mix	21.35	89	<input type="checkbox"/>
Grandma's Boysenberry Spread	25.00	120	<input type="checkbox"/>
Uncle Bob's Organic Dried Pears	30.00	6	<input type="checkbox"/>
Northwoods Cranberry Sauce	40.00	21	<input type="checkbox"/>
Mishi Kobe Niku	97.00	29	<input checked="" type="checkbox"/>
Ikura	31.00	31	<input type="checkbox"/>
Queso Cabrales	21.00	22	<input type="checkbox"/>
Queso Manchego La Pastora	38.00	86	<input type="checkbox"/>
Konbu	6.00	24	<input type="checkbox"/>

## Bootstrap 上下文 Table 样式

Bootstrap 提供了额外的 class 能让我们修饰表的样式，提供的 class 如下：

- Active
- Success
- Info
- Warning
- Danger

修改上述代码，为动态添加样式：

```
@foreach (var item in Model)
{
<tr class="@item.Status">
  <td>
    @Html.DisplayFor(modelItem => item.ProductName)
  </td>
  <td>
    @Html.DisplayFor(modelItem => item.UnitPrice)
  </td>
  <td>
    @Html.DisplayFor(modelItem => item.UnitsInStock)
  </td>
  <td>
    @Html.DisplayFor(modelItem => item.Discontinued)
  </td>
</tr>
}
```

更新过后的效果如下所示：

## Products

Product Name	Unit Price	Units In Stock	Discontinued
Chai	18.00	35	<input type="checkbox"/>
Chang	19.00	17	<input checked="" type="checkbox"/>
Aniseed Syrup	10.00	27	<input type="checkbox"/>
Chef Anton's Cajun Seasoning	61.00	53	<input type="checkbox"/>
Chef Anton's Gumbo Mix	21.35	89	<input type="checkbox"/>
Grandma's Boysenberry Spread	25.00	120	<input type="checkbox"/>
Uncle Bob's Organic Dried Pears	30.00	6	<input type="checkbox"/>
Northwoods Cranberry Sauce	40.00	21	<input type="checkbox"/>
Mishi Kobe Niku	97.00	29	<input checked="" type="checkbox"/>
Ikura	31.00	31	<input type="checkbox"/>
Queso Cabrales	21.00	22	<input type="checkbox"/>
Queso Manchego La Pastora	38.00	86	<input type="checkbox"/>
Konbu	6.00	24	<input type="checkbox"/>

## Bootstrap Buttons

Bootstrap 提供了许多各种不同颜色和大小的大小，为核心的 buttons 提供 6 种颜色和 4 种尺寸可以选择，同样通过设置 class 属性来显示不同的风格：

- btn btn-primary btn-xs
- btn btn-default btn-sm
- btn btn-default
- btn btn-success btn-lg

可以为 Button 设置颜色的 class：

- btn-default
- btn-primary
- btn-success
- btn-info
- btn-warning
- btn-danger

所以可以使用如下代码来呈现效果：

```

<div class="row">
  <!-- default按钮 -->
  <button type="button" class="btn btn-default btn-xs">
    Default & Size=Mini
  </button>
  <button type="button" class="btn btn-default btn-sm">
    Default & Size=Small
  </button>
  <button type="button" class="btn btn-default">Default</button>
  <button type="button" class="btn btn-default btn-lg">
    Default & Size=Large
  </button>
</div>

```

显示效果如下：



## Bootstrap Form ( 表单 )

表单常见于大多数业务应用程序里，因此统一的样式有助于提高用户体验，Bootstrap 提供了许多不同的 CSS 样式来美化表单。

使用 ASP.NET MVC 的 `Html.BeginForm` 可以方便的创建一个表单，通过为

<

`form`>添加名为 `form-horizontal` 的 class 来创建一个 Bootstrap 水平显示表单。

```

@using (Html.BeginForm("Login", "Account", FormMethod.Post, new { @class = "form-horizontal", role = "form" }))
{

```



```

<div class="form-group">
    @Html.LabelFor(m => m.UserName, new { @class = "col-md-2 control-label" })
    <div class="col-md-10">
        @Html.TextBoxFor(m => m.UserName, new { @class = "form-control" })
        @Html.ValidationMessageFor(m => m.UserName)
    </div>
</div>
<div class="form-group">
    @Html.LabelFor(m => m.Password, new { @class = "col-md-2 control-label" })
    <div class="col-md-10">
        @Html.PasswordFor(m => m.Password, new { @class = "form-control" })
        @Html.ValidationMessageFor(m => m.Password)
    </div>
</div>
<div class="form-group">
    <div class="col-md-offset-2 col-md-10">
        <input type="submit" value="Log in" class="btn btn-default">
    </div>
</div>
}

```

上述代码中，使用 class 为 form-group 的

<

div>元素包裹了 2 个 Html 方法（Html.LabelFor、Html.TextBoxFor），这能让 Bootstrap 验证样式应用在 form 元素上，当然你也可以使用 Bootstrap 栅格 col- class 来指定 form 中元素的宽度，效果如下显示：

Bootstrap 基础表单默认情况下是垂直显示内容，在 Html.BeginForm 帮助方法里移除 class 为 form-horizontal 和 class col- 后，显示的效果如下：

## Log in.

User name

Password

Log in

内联表单表示所有的 form 元素一个接着一个水平排列，只适用于视口（viewport）至少在 768px 宽度时（视口宽度再小的话就会使表单折叠）。

记得一定要添加 **label** 标签，如果你没有为每个输入控件设置 label 标签，屏幕阅读器将无法正确识别。对于这些内联表单，你可以通过为 label 设置 **.sr-only** 类将其隐藏。详细代码如下：

```
@using (Html.BeginForm("Login", "Account", FormMethod.Post, new { @class = "form-inline", role = "form" }))
{
    <div class="form-group">
        @Html.LabelFor(m => m.UserName, new { @class = "sr-only" })
        @Html.TextBoxFor(m => m.UserName, new { @class = "form-control", placeholder = "Enter your username" })
        @Html.ValidationMessageFor(m => m.UserName)
    </div>
    <div class="form-group">
        @Html.LabelFor(m => m.Password, new { @class = "sr-only" })
        @Html.PasswordFor(m => m.Password, new { @class = "form-control", placeholder = "Enter your username" })
        @Html.ValidationMessageFor(m => m.Password)
    </div>
    <div class="form-group">
        <input type="submit" value="Log in" class="btn btn-default">
    </div>
}
```

显示效果如下：

## Log in.



Log in

## Bootstrap Image

在 Bootstrap 3.0 中, 通过为图片添加 `.img-responsive` 类可以让图片支持响应式布局。其实质是为图片设置了 `max-width: 100%;`、`height: auto;` 和 `display: block;` 属性, 从而让图片在其父元素中更好的缩放。

通过为 元素添加以下相应的类, 可以让图片呈现不同的形状。

- `img-rounded`
- `img-circle`
- `img-thumbnail`

请看如下代码:

```
<div class="row">
  <h3>Our Team</h3>
  @foreach (var item in Model)
  {
    <div class="col-md-4">
      
        @item.FirstName @item.LastName <small>@item.Title</small>
      </h3>
      <p>@item.Notes</p>
    </div>
  }
</div>
```



**Robert King** Sales Representative

Robert King served in the Peace Corps and traveled extensively before completing his degree in English at the University of Michigan in 1992.



**Laura Callahan** Inside Sales Coordinator

Laura received a BA in psychology from the University of Washington.



**Anne Dodsworth** Sales Representative

Anne has a BA degree in English from St. Lawrence College. She is fluent in French and German.

## Bootstrap 验证样式

默认情况下 ASP.NET MVC 项目模板支持 unobtrusive 验证并且会自动添加需要的 JavaScript 库到项目里。然而默认的验证不使用 Bootstrap 指定的 CSS。当一个 input 元素验证失败，JQuery validation 插件会为元素添加 input-validation-error class（存在 Site.css 中）。那怎样不修改 JQuery Validation 插件而且使用 Bootstrap 内置的错误样式呢？

Bootstrap 提供了 class 为 has-error 中的样式（label 字体变为红色，input 元素加上红色边框）来显示错误：

```
<div class="form-group has-error">
  @Html.LabelFor(m => m.UserName, new { @class = "col-md-2 control-label" })
  <div class="col-md-10">
    @Html.TextBoxFor(m => m.UserName, new { @class = "form-control" })
  </div>
</div>
```

所以，我需要动态来为

<

div class="form-group">元素动态绑定/移除 has-error。为了不修改 JQuery.validation 插件，我在 Scripts 文件夹中添加 jquery.validate.bootstrap 文件：

```
$.validator.setDefaults({
  highlight: function (element) {
    $(element).closest('.form-group').addClass('has-error');
  },
  unhighlight: function (element) {
    $(element).closest('.form-group').removeClass('has-error');
  },
});
```

这段脚本的通过调用 setDefaults 方法来修改默认的 JQuery validation 插件设置。看以看到我使用 highlight 和 unhighlight 方法来动态添加/移除 has-error class。

最后将它添加到打包文件中

```
bundles.Add(new ScriptBundle("~/bundles/jqueryval").Include(
  "~/Scripts/jquery.validate.js",
  "~/Scripts/jquery.validate.unobtrusive.js",
  "~/Scripts/jquery.validate.bootstrap.js"));
```

注：默认情况下，ASP.NET MVC 使用通配符来将 `jquery.validate` 文件打包到 `jqueryval` 文件中，如下所示：

```
bundles.Add(new ScriptBundle("~/bundles/jqueryval").Include(  
    "~/Scripts/jquery.validate*"));
```

但是 `jquery.validate.bootstrap.js` 必须在 `jquery validate` 插件后加载，所以我们只能显式的指定文件顺序来打包，因为默认情况下打包加载文件的顺序是按通配符代表的字母顺序排列的。

## ASP.NET MVC 创建包含 Bootstrap 样式编辑模板

编辑模板 (Editor Template) 指的是在 ASP.NET MVC 应用程序中, 基于对象属性的数据类型通过 Razor 视图渲染后, 自动产生表单 Input 元素。ASP.NET MVC 包含了若干的编辑模板, 当然我们也可以实现扩展。编辑模板类似于局部视图, 不同的是, 局部视图通过 name 来渲染, 而编辑模板通过类型来渲染。

举个例子, `@Html.EditorFor(model => model.Property)`, 如果 Property 类型为 string, 那么 `@Html.Editor` 会创建一个 Type=Text 的 Input 元素; 如果 Property 类型为 Password, 那么会创建一个 Type=Password 的 Input 元素。所以 EditorFor helper 是基于 model 属性的数据类型来渲染生成 HTML。

不过美中不足的是, 默认产生的 HTML 如下所示:

```
<input class="text-box single-line" data-val="true" data-val-required="The Category Name field is required." id="CategoryName" name="CategoryName" type="text" value>
```

可以看到 `class="text-box single-line"`, 但先前提到过, Bootstrap Form 元素 class 必须是 `form-control`。

所以, 为了让 Editor helper 生成 class 为 `form-control` 的表单元素, 我们需要创建一个自定义的编辑模板来重写旧的模板。你需要如下操作:

- 在 Shared 文件夹中创建名为 EditorTemplates (注意要一样的名称) 的文件夹
- 添加名为 string.cshtml (注意要一样的名称) 文件, 并添加如下代码:

```
@model string @Html.TextBox("", ViewData.TemplateInfo.FormattedModelValue, new { @class = "form-control" })
```

在上述代码中, 我们调用 `@Html.TextBox` 方法, 并且传递了一个空的字符串作为 textbox 的 name。这将会让 model 的属性名作为生成的 textbox 的 name, 并且 textbox 显示的内容是 model 的值, 最后追加了名为 `class` 的 attribute, 而且其值为 `"form-control"`。

重新生成项目, 发现新生成的 input 元素它的 class 已经改为 `"form-control"` 了。如下所示:

```
<input class="form-control" data-val="true" data-val-required="The Category Name field is required." id="CategoryName" name="CategoryName" type="text" value>
```

ASP.NET MVC 能让开发者创建根据自定义\*\*\*\*Data Type 的编辑模板, 比如自动生成多行文本框并且规定行数为 3 行, 也是同样的操作:

- 添加 MultilineText.Cshtml (注意名称相同) 文件到 EditorTemplates 中

- 添加如下代码：

```
@model string @Html.TextArea("", ViewData.TemplateInfo.FormattedModelValue. ToString(), new { @class = "form-control", rows = 3 })
```

- 为了让我们的 Model 的属性在渲染时采用 MultilineText.cshtml 编辑模板，我们需要为属性指定 Data Type attribute 为 MultilineText:

```
[DataType(DataType.MultilineText)] public string Description { get; set; }
```

最终显示如下所示：

```
<textarea class="form-control" cols="20" id="Description" name="Description" rows="3"></textarea>
```

## 小结

---

这篇文章为大家介绍了 Bootstrap 的响应式布局 ( grid )，并且简单介绍了 Bootstrap 中的 HTML 元素，包括 Table、Button、Form、Image…然后修改了 JQuery validate 插件默认的设置，使其友好支持 Bootstrap 中的错误提示样式。最后探索了 ASP.NET MVC 中的编辑模板，能让产生的 input 元素自动包含 form-control 样式，谢谢大家支持。





4

# ASP.NET MVC 随想录 (3) —— 使用 Bootstrap 组件

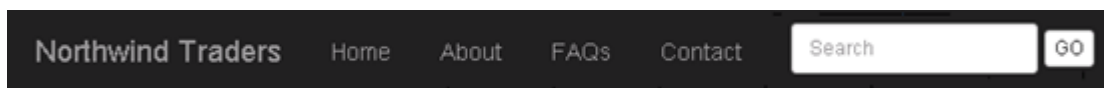


Bootstrap 为我们提供了十几种的可复用组件，包括字体图标、下拉菜单、导航、警告框、弹出框、输入框组等。在你的 Web Application 中使用这些组件，将为用户提供一致和简单易用的用户体验。

Bootstrap 组件本质上是结合了各种现有 Bootstrap 元素以及添加了一些独特 Class 来实现。Bootstrap 元素我在上一篇文章中涉及到。在这篇博客中，我将继续探索 Bootstrap 丰富的组件以及将它结合到 ASP.NET MVC 项目中。

## Bootstrap 导航条

Bootstrap 导航条作为“明星组件”之一，被使用在大多数基于 Bootstrap Framework 的网站上。为了更好的展示 Bootstrap 导航条，我在 ASP.NET MVC 的 \_Layout.cshtml 布局页创建一个 fixed-top 导航条，当然它是响应式的——在小尺寸、低分辨率的设备上打开时，它将会只展示一个按钮并带有 3 个子菜单，当点击按钮时垂直展示他们。在网页上显示如下：



在移动设备上显示如下：



在 ASP.NET MVC 默认的 \_Layouts.cshtml 布局页中已经帮我们实现了上述功能，打开它对其稍作修改，如下代码片段所示：

```
<div class="navbar navbar-inverse navbar-fixed-top">

    <div class="container">

        <div class="navbar-header">

            <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">

                <span class="icon-bar"></span>

                <span class="icon-bar"></span>

                <span class="icon-bar"></span>

            </button>

            @Html.ActionLink("Northwind Traders", "Index", "Home", null, new { @class = "navbar-brand" })

        </div>

        <div class="navbar-collapse collapse">

            @Html.Partial("_BackendMenuPartial")

        </div>

    </div>

</div>
```

```

        @Html.Partial("_LoginPartial")

    </div>

</div>

</div>

```

其中 class 为 .navbar-fixed-top 可以让导航条固定在顶部, 还可包含一个 .container 或 .container-fluid 容器, 从而让导航条居中, 并在两侧添加内补 (padding)

注意, 我使用了 2 个局部视图 (\_BackendMenuPartial 和 LoginPartial) 来生成余下的导航条 (使用 .navbar-collapse 类在低分辨率设备中折叠), 其中局部视图逻辑是基于当前访问的用户是否登陆来控制是否显示。

首先, 添加如下代码在 \_BackendMenuPartial 视图中, 这将会在导航条中产生一个搜索框:

```

@using (Html.BeginForm("Index", "Search", FormMethod.Post, new { @class = "navbar-form navbar-left", role = "search" })
{

    <div class="form-group">

        @Html.TextBox("searchquery", "", new { @id = "searchquery", @class = "form-control input-sm", placeholder = "Search" })

        @Html.Hidden("fromcontroller", @ViewContext.RouteData.Values["controller"], new { @id = "fromcontroller" })

    </div>

    <button type="submit" class="btn btn-default btn-xs">GO</button>

}

```

因为 Bootstrap 导航条作为整个网站的公共部分, 要实现快速搜索那么必须要知道当前所处于哪个 Controller, 这样才能提高检索效率。所以上述代码中, 增加了一个 Id 为 *fromcontroller* 隐藏字段, 代表当前访问的 Controller。

当点击搜索时, 发送 HTTP POST 请求到 Index Action 下。然后根据传递过来的 fromcontroller 来 switch 到具体的 Action 来执行搜索, 具体的搜索逻辑代码如下:

```

public ActionResult Index(string searchquery, string fromcontroller)
{

    switch (fromcontroller)
    {

```

```

        case "Products":

            return RedirectToAction("SearchProductsResult", new { query = searchquery });

        case "Customers":

            return RedirectToAction("SearchCustomersResult", new { query = searchquery });

        case "Employees":

            return RedirectToAction("SearchEmployeesResult", new { query = searchquery });

    }

    return View();
}

```

具体搜索的 Action 如下：

```

public ActionResult SearchProductsResult(string query)
{
    ViewBag.SearchQuery = query;

    var results = _context.Products.Where(p => p.ProductName.Contains(query)).ToList();

    return View(results);
}

public ActionResult SearchCustomersResult(string query)
{
    ViewBag.SearchQuery = query;

    var results = _context.Customers.Where(p => p.CompanyName.Contains(query)
        || p.ContactName.Contains(query)
        || p.City.Contains(query)
        || p.Country.Contains(query)).ToList();
}

```

```
return View(results);

}

public ActionResult SearchEmployeesResult(string query)
{
    ViewBag.SearchQuery = query;

    var results = _context.Employees.Where(p => p.FirstName.Contains(query)
    || p.LastName.Contains(query)
    || p.Notes.Contains(query)).ToList();

    return View(results);
}
```

## 列表组

---

列表组是灵活又强大的组件，不仅能用于显示一组简单的元素，还能结合其他元素创建一组复杂的定制内容。上面的搜索为我们重定向到 Result 视图，在此视图中，它为我们显示了搜索结果，为了更好的展示结果，我们可以使用列表组来显示搜索到的产品，视图中的代码如下所示：

```
@model IEnumerable<bootstrap.data.models.products>

@{
    ViewBag.Title = "搜索产品";
}

<div class="container">

    <div class="page-header">

        <h1>产品结果 <small>搜索条件: "@ViewBag.SearchQuery"</small></h1>

    </div>

    <ul class="list-group">

        @foreach (var item in Model)
        {

            <a href="@Url.Action("edit","products",new={ id=@item.ProductID})"class="list-group-item">@item.ProductName <s

        }

    </ul>

</div>
```

在上述代码中，为无序列表 ul 的 class 设置为 list-group，并且每一个 li 的 class 为 list-group-item，这是一个最简单的列表组。

## 徽章

徽章用来高亮条目，可以很醒目的显示新的或者未读的条目数量，为一个元素设置徽章仅仅只需要添加元素并设置它的 class 为 badge。所以，在上述代码的基础上稍作修改，添加徽章，表示库存个数，如下 HTML 所示：

```
<a href="@Url.Action("edit","products",new {id="@item.ProductID})" class="list-group-item">
    @item.ProductName <span class="badge">@item.UnitsInStock</span>
</a>
```

显示的结果为如下截图：

Aniseed Syrup	27
Grandma's Boysenberry Spread	120
Uncle Bob's Organic Dried Pears	6
Northwoods Cranberry Sauce	21



## 媒体对象

媒体对象组件被用来构建垂直风格的列表比如博客的回复或者推特。在 Northwind 数据库中包含一个字段 ReportTo 表示 Employee 向另一个 Employee Report。使用媒体对象可以直观来表示这种关系。在视图中的代码如下所示：

```
<div class="container">

<div class="page-header">

    <h1>员工搜索结果: <small>搜索条件: "@ViewBag.SearchQuery"</small></h1>

</div>

@foreach (var item in Model)
{
    <div class="media">

        <a class="pull-left" href="@Url.Action("edit", "employees", new { id="@item.EmployeeID" })">

        <div class="media-body">

            <h4 class="media-heading">@item.FirstName @item.LastName</h4>

            @item.Notes

            @foreach (var emp in @item.ReportingEmployees)
            {
                <div class="media">

                    <a href="#" class="pull-left">

```

```

</a>

<div class="media-body">

    <h4 class="media-heading">@emp.FirstName @emp.LastName</h4>

    @emp.Title

</div>

</div>

}

</div>

</div>

}





</div>

```

显示结果如下：

## 员工搜索结果： 搜索条件: "steven"

---

	<b>Steven Buchanan</b> Steven Buchanan graduated from St. Andrews University, Scotland, with a BSC degree in 1976.
	<b>Michael Suyama</b> Sales Representative
	<b>Robert King</b> Sales Representative
	<b>Anne Dodsworth</b> Sales Representative

---

可以看到，媒体对象组件是由一系列 class 为 media、media-heading、media-body、media-object 的元素组合而成，其中 media-object 用来表示诸如图片、视频、声音等媒体对象。

注：.pull-left 和 .pull-right 这两个类以前也曾经被用在了媒体组件上，但是，从 v3.3.0 版本开始，他们就不再被建议使用了。.media-left 和 .media-right 替代了他们，不同之处是，在 html 结构中，.media-right 应当放在 .media-body 的后面。

## 页头

---

当用户访问网页时, Bootstrap 页头可以为用户提供清晰的指示。Bootstrap 页头本质上是一个元素被封装在 `class=page-header` 的

<

`div>`元素中。当然你也可以利用元素来提供额外的关于页面的信息, 同时 Bootstrap 为页头添加了水平分隔线用于分隔页面, 如下 HTML 即为我们构建了页头:

```
<div class="page-header">

    <h1>员工搜索结果: <small>搜索条件: "@ViewBag.SearchQuery"</small></h1>

</div>
```

## 路径导航

---

路径导航（面包屑）在 Web 设计中用来表示用户在带有层次的导航结构中当前页面的位置。类似于 Windows 资源管理器。如下 HTML 所示：

```
<ol class="breadcrumb">

    <li>@Html.ActionLink("Home", "Index", "Home")</li>

    <li>@Html.ActionLink("Manage", "Index", "Manage")</li>

    <li class="active">Products</li>

</ol>
```

在上面 HTML 代码中，通过指定有序列表 ol 的 class 为 breadcrumb，每一个子路径用 li 来表示，其中通过设置 li 的 class 为 active 代表当前所处的位置。

各路径间的分隔符已经自动通过 CSS 的 :before 和 content 属性添加了。

## 分页

分页用来分隔列表内容，特别是显示大量数据时通过分页可以有效的减少服务器压力和提高用户体验，如下截图使用分页来显示产品列表：

Products Page 5 of 16

[Home](#) / [Manage](#) / [Products](#)

Product Name	Unit Price	Units In Stock
<a href="#">Gravad lax</a>	26	11
<a href="#">Guaraná Fantástica</a>	5	20
<a href="#">Gudbrandsdalsost</a>	36	26
<a href="#">Gula Malacca</a>	19	27
<a href="#">Gumbär Gummibärchen</a>	31	15

[««](#) [«](#) [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [...](#) [»](#) [»»](#)

要完成上述的分页，需要安装 PagedList.Mvc 程序包，在 NuGet 控制台中安装即可：Install-PackagePagedList.Mvc

然后修改 Action，它需要接受当然的页码，它是一个可空的整数类型变量，然后设置 PageSize 等于5，表示每页显示 5 条记录，如下代码所示：

```
public ActionResult Index(int? page)
{
    var models = _context.Products.Project().To<productviewmodel>().OrderBy(p => p.ProductName);

    int pageSize = 5;

    int pageNumber = (page ?? 1);

    return View(models.ToPagedList(pageNumber, pageSize));
}
```

## 输入框组

输入框组为用户在表单输入数据时可以提供更多的额外信息。Bootstrap 的输入框组为我们在 Input 元素的前面或者后面添加指定 class 的块，这些块可以是文字或者字体图标，如下所示：

```
<div class="form-group">

  <div class="col-sm-2 input-group">

    <span class="input-group-addon">

      <span class="glyphicon glyphicon-phone-alt"></span>

    </span>

    @Html.TextBox("txtPhone", "1194679215", new { @class = "form-control" })

  </div>

</div>
```

上面的输入框组合中，在 Textbox 的左边放置了一个带有字体图标 Phone 的灰色块，结果如下所示：



不仅可以使字体图标，还可以使用纯文本来显示信息，如下所示在 Textbox 右边放置了固定的邮箱域名：

```
<div class="form-group">


  <div class="col-sm-4 input-group">

    @Html.TextBox("txtEmail", "1194679215", new { @class = "form-control" })

    <span class="input-group-addon">@@qq.com</span>

  </div>

</div>
```



当然也可以在 Input 元素的两边同时加上块，如下代码所示：

```
<div class="form-group">

    <div class="col-sm-2 input-group">

        <span class="input-group-addon">¥</span>

        @Html.TextBox("txtMoney","100",new { @class = "form-control" })

        <span class="input-group-addon">.00</span>

    </div>

</div>
```

¥	100	.00
---	-----	-----

## 按钮式下拉菜单

按钮式下拉菜单顾名思义，一个按钮可以执行多种 action，比如既可以 Save，也可以 Save 之后再打开一个新的 Form 继续添加记录，如下所示：

```
<div class="form-group">

  <div class="col-sm-offset-2 col-sm-10">

    <div class="btn-group">

      <button type="submit" class="btn btn-primary btn-sm">Save</button>

      <button type="button" class="btn btn-primary btn-sm dropdown-toggle" data-toggle="dropdown">

        <span class="caret"></span>

        <span class="sr-only">Toggle Dropdown</span>

      </button>

      <ul class="dropdown-menu" role="menu">

        <li><a href="#" id="savenew">Save & New</a></li>

        <li class="divider"></li>

        <li><a href="#" id="duplicate">Duplicate</a></li>

      </ul>

    </div>

  </div>

</div>

</div>
```



## 警告框

---

Bootstrap 警告组件通常被用作给用户提供可视化的反馈，比如当用户 Save 成功后显示确认信息、错误时显示警告信息、以及其他的提示信息。

Bootstrap 提供了 4 种不同风格的警告，如下所示：

```
<div class="container">

  <div class="page-header">

    <h1>Alerts </h1>

  </div>

  <ol class="breadcrumb">

    <li>@Html.ActionLink("Home", "Index", "Home")</li>

    <li>Bootstrap</li>

    <li class="active">Alerts</li>

  </ol>

  <div class="alert alert-success"><strong>Success. </strong></div>

  <div class="alert alert-info"><strong>Info.</strong></div>

  <div class="alert alert-warning"><strong>Warning!</strong></div>

  <div class="alert alert-danger"><strong>Danger!</strong></div>

</div>
```

# Alerts

[Home](#) / [Bootstrap](#) / Alerts

**Success.**

**Info.**

**Warning!**

**Danger!**

可关闭的警告框可以让用户点击右上角的 X 来关闭，你可以使用 `alert-dismissible` 类：

```
<div class="alert alert-warning alert-dismissible" role="alert">

  <button type="button" class="close" data-dismiss="alert">

    <span aria-hidden="true"> × </span><span class="sr-only">Close</span>

  </button>

  <strong>Alert!</strong>这是可关闭的Alter

</div>
```

## 进度条

---

进度条在传统的桌面应用程序比较常见,当然也可以用在 Web 上。通过这些简单、灵活的进度条,可以为当前工作流程或动作提供实时反馈。Bootstrap 为我们提供了许多样式的进度条。

基本进度条是一种纯蓝色的进度条,添加一个 class 为 sr-only 的元素在进度条中是比较好的实践,这样能让屏幕更好的读取进度条的百分比。

```
<div class="row">

<h4>基本进度条</h4>

<div class="progress">

  <div class="progress-bar" role="progressbar" aria-valuenow="80" aria-valuemin="0" aria-valuemax="100" style="width: 80%;">

    <span class="sr-only">80%完成</span>

  </div>

</div>

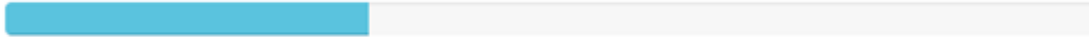
</div>
```

### 基本进度条



上下文情景变化进度条使用与按钮和警告框相同的类,根据不同情境展现相应的效果

- progress-bar-success
- progress-bar-info
- progress-bar-warning
- progress-bar-danger

**情景进度条(Success)****情景进度条(Info)****情景进度条(Warning)****情景进度条(Danger)**

条纹动画效果进度条，为了让进度条更加生动，可以为其添加条纹效果，在进度条 div 中添加 class 为 progress-striped。当然让进度条看起来有动画，可以再为其指定 active 的 class 在 div 上，如下所示：

```
<div class="progress progress-striped active">

<div class="progress-bar progress-bar-success" role="progressbar" aria-valuenow="40" aria-valuemin="0" aria-valuemax="100">

    <span class="sr-only">40% 完成 (success)</span>

</div>

</div>
```

**条纹进度条(Success)**

## 小结

---

在这篇文章中，探索了Bootstrap中丰富的组件，并将它结合到ASP.NET MVC项目中。通过实例可以发现，这类组件本质上是结合了各种现有Bootstrap元素以及添加了一些独特Class来实现。



5

## ASP.NET MVC 使用 Bootstrap 系列（4）——使用 Java aScript 插件



Bootstrap 的 JavaScript 插件是以 JQuery 为基础，提供了全新的功能并且还可以扩展现有的 Bootstrap 组件。通过添加 data attribute（data 属性）可以轻松的使用这些插件，当然你也可以使用编程方式的 API 来使用。

为了使用 Bootstrap 插件，我们需要添加 Bootstrap.js 或者 Bootstrap.min.js 文件到项目中。这两个文件包含了所有的 Bootstrap 插件，推荐引用 Bootstrap.min.js。当然你也可以包含指定的插件来定制化 Bootstrap.js，已达到更好的加载速度。

## Data 属性 VS 编程 API

---

Bootstrap 提供了完全通过 HTML 标记的方式来使用插件，这意味着，你可以不写任何 JavaScript 代码，事实上这也是 Bootstrap 推荐的使用方式。

举个例子，若要使 Alert 组件支持关闭功能，你可以通过添加 data-dismiss="alert" 属性到按钮(Button)或者链接(A)元素上，如下代码所示：

```
<div class="alert alert-danger">
  <button data-dismiss="alert" class="close" type="button">x</button>
  <strong>警告</strong>10秒敌人到达
</div>
```

当然，你也可以通过编程方式的 API 来实现同样的功能：

```
<div class="alert alert-danger" id="myalert">
  <button data-dismiss="alert" class="close" type="button" onclick="$('#myalert').alert('close')">x</button>
  <strong>警告</strong>10秒敌人到达
</div>
```



## 下拉菜单 (dropdown.js)

---

使用 dropdown 插件 (增强交互性)，你可以将下拉菜单添加到绝大多数的 Bootstrap 组件上，比如 navbar、tabs 等。注：将下拉菜单触发器和下拉菜单都包裹在 .dropdown 里，或者另一个声明了 position: relative; 的元素中。

如下是一个地域的菜单，通过 Razor 引擎动态绑定菜单元素：

```
<div class="form-group">
  @Html.LabelFor(model => model.TerritoryId, new { @class = "control-label col-md-2" })
  @Html.HiddenFor(model => model.TerritoryId)
  <div class="col-md-10">
    <div id="territorydropdown" class="dropdown">
      <button id="territorybutton" class="btn btn-sm btn-info" data-toggle="dropdown">@Model.Territory.TerritoryDe
      <ul id="territories" class="dropdown-menu">
        @foreach (var item in ViewBag.TerritoryId)
        {
          <li><a href="#" tabindex="-1" data-value="@item.Value">@item.Text</a></li>
        }
      </ul>
    </div>
  </div>
</div>
```

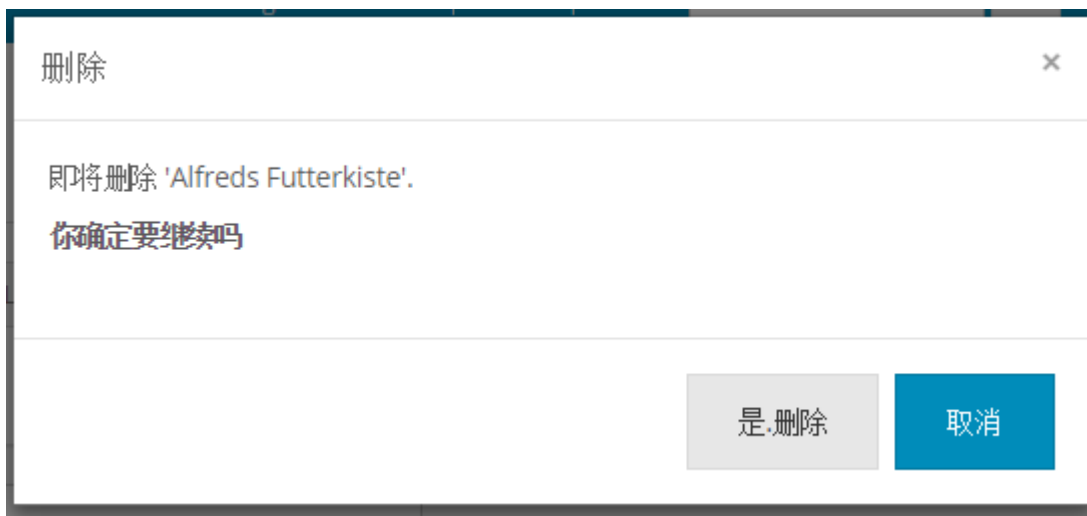
注意：通过添加 data 属性：data-toggle="dropdown" 到 Button 或者 Anchor 上，可以切换 dropdown 下拉菜单，增加了交互性。其中菜单的元素设置 tabindex=-1，这样做是为了防止元素成为 tab 次序的一部分。

## 模态框（modal.js）

模态框以弹出框的形式可以为用户提供信息亦或者在此之中完成表单提交功能。Bootstrap 的模态框本质上有 3 部分组成：模态框的头、体和一组放置于底部的按钮。你可以在模态框的 Body 中添加任何标准的 HTML 标记，包括 Text 或者嵌入的 Youtube 视频。

一般来说，务必将模态框的 HTML 代码放在文档的最高层级内（也就是说，尽量作为 body 标签的直接子元素），以避免其他组件影响模态框的展现和/或功能。

如下即为一个标准的模态框，包含 Header、Body、Footer:



将下面这段 HTML 标记放在视图的 Top 或者 Bottom:

```
<div class="modal fade" id="deleteConfirmationModal" tabindex="-1" role="dialog" aria-hidden="true">
<div class="modal-dialog">
  <div class="modal-content">
    <div class="modal-header">
      <button type="button" class="close" data-dismiss="modal" aria-hidden="true">&times;</button>
      <h4 class="modal-title">删除</h4>
    </div>
    <div class="modal-body">
      <p>即将删除 '@Model.CompanyName'. </p>
      <p>
        <strong>你确定要继续吗</strong>
      </p>
      @using (Html.BeginForm("Delete", "Customers", FormMethod.Post, new { @id = "delete-form", role = "form" }))
      {
        @Html.HiddenFor(m => m.CustomerId)
        @Html.AntiForgeryToken()
      }
    </div>
    <div class="modal-footer">
```

```

        <button type="button" class="btn btn-default" onclick="$('#delete-form').submit();">是.删除</button>
        <button type="button" class="btn btn-primary" data-dismiss="modal">取消</button>
    </div>
</div>
</div>
</div>

```

注意：通过在 Button 上添加 data 属性：data-dismiss="modal" 可以实现对模态框的关闭，当然你也可以使用编程方式 API 来完成：

```
<button type="button" class="btn btn-primary" onclick="$('#deleteConfirmationModal').modal('hide')">取消</button>
```

为了展示模态框，我们可以不写任何 JavaScript 代码，通过添加 data-toggle="modal" 属性到 Button 或者 Anchor 元素上即可，同时，为了表示展示哪一个模态框，需要通过 data-target 来指定模态框的 Id。

```
<a href="#" data-toggle="modal" data-target="#deleteConfirmationModal">Delete</a>
```

同样，也可以使用编程方式 API 来打开一个模态框：

```

$(document).ready(function () {

    $('#deleteConfirmationModal').modal('show');

});

```

## 标签页（tab.js）

Tabs 可以使用在大的表单上，通过 Tabs 分割成若干部分显示局部信息，比如在 Northwind 数据库中存在 Customer 顾客信息，它包含了基本信息和地址，可以通过 Tabs 进行 Split，如下所示：

要使用 Tabs 也是非常简单的：首先创建标准的无序列表元素，需要为它的 class 设置为 nav nav-tabs 或者 nav nav-pills。其中包含的元素即为 Tab 元素，需要设置其 data-toggle 为 tab 或者 pill 属性以及点击它指向的内容：

```
<ul id="customertab" class="nav nav-tabs">
  <li class="active"><a href="#info" data-toggle="tab">Customer Info</a></li>
  <li><a href="#address" data-toggle="tab">Address</a></li>
</ul>
```

为了设置 Tabs 的内容，需要创建一个父元素并设置 class 为 tab-content，在父的 div 容器中为每一个 Tab 内容创建 div 元素，并设置 class 为 tab-pane 和标识的 Id，通过添加 active 来激活哪一个 Tab 内容的显示。

```
<div class="tab-content well">
  <div class="tab-pane active" id="info">
    Customer Info
  </div>
  <div class="tab-pane" id="address">
    Customer Address
  </div>
</div>
```

当然也可以通过编程方式的 API 来激活：

```
$(document).ready(function () {
  $('nav-tabs a[href="#address"]').tab('show');
});
```

## 工具提示 ( tooltip.js )

---

Tooltip 能为用户提供额外的信息，Bootstrap Tooltip 能被用在各种元素上，比如 Anchor：

```
<a data-toggle="tooltip" data-placement="top" data-original-title="这是提示内容" href="#" >工具提示? </a>
```

你可以添加 data-toggle="tooltip" 来使用 tooltip，当然你也可以设置内容的显示位置，通过添加 data-placement 属性来实现，Bootstrap 为我们提供了 4 种位置：

- top
- bottom
- left
- right

最后，通过设置 data-original-title 设置了需要显示的 Title。

注意：为了性能的考虑，Tooltip 的 data-api 是可选的，这意味着你必须手动初始化 tooltip 插件：

```
<script type="text/javascript">
  $(document).ready(function () {
    $('[data-toggle="tooltip"]').tooltip();
  });
</script>
```

## 弹出框 ( popover.js )

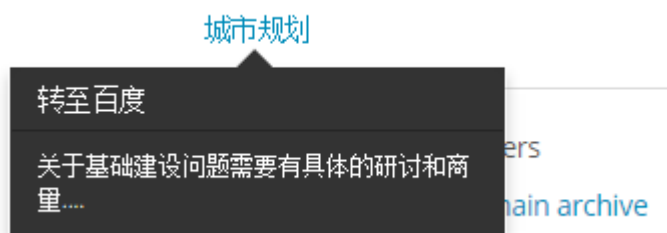
---

弹出框和 Tooltip 类似，都可以为用户提供额外的信息，但弹出框可以展示更多的信息，比如允许我们展示一个 Header 和 Content，如下所示：

```
<a data-content="关于基础建设问题需要有具体的研讨和商量...." data-placement="bottom" title="" data-toggle="popover" href="#">
```

和 Tooltip 一样，为了性能的考虑，data-api 是可选的，这意味着你必须手动初始化 popover 插件：

```
<script type="text/javascript">
$(document).ready(function () {
    $('[data-toggle="popover"]').popover();
});
</script>
```



## 手风琴组件（collapse.js）

手风琴组件有若干 panel groups 组成，每一个 panel group 依次包含了若干 header 和 content 元素,最常见的使用场景就是 FAQ，如下所示：

问题 1：什么是 Microsoft MVP 奖励？

问题 2：MVP 奖励存在的意义何在？

解答 2：我们相信，技术社区可以促进自由且客观的知识交流，因此可以构建出一个可靠、独立、真实且可使每个人获益的专业知识来源。Microsoft MVP 奖励旨在表彰那些能积极与其他技术社区成员分享自身知识的全球杰出技术社区领导者。

为了使用手风琴组件，需要对 Panel Heading 中的设置 data-toggle="collapse"和点击它展开的容器(Div)Id，具体如下所示:

```
<div class="row">
  <div class="panel-group" id="accordion">
    <div class="panel panel-default">
      <div class="panel-heading">
        <h4 class="panel-title">
          <a data-toggle="collapse" data-parent="#accordion"
            href="#questionOne">
            问题 1：什么是 Microsoft MVP 奖励？
          </a>
        </h4>
      </div>
      <div id="questionOne" class="panel-collapse collapse in">
        <div class="panel-body">
          解答 1：Microsoft 最有价值专家 (MVP) 奖励是一项针对 Microsoft 技术社区杰出成员的年度奖励，根据 Microsoft 技术
        </div>
      </div>
    </div>
    <div class="panel panel-default">
      <div class="panel-heading">
        <h4 class="panel-title">
          <a data-toggle="collapse" data-parent="#accordion"
            href="#questionTwo">
            问题 2：MVP 奖励存在的意义何在？
          </a>
        </h4>
      </div>
      <div id="questionTwo" class="panel-collapse collapse">
        <div class="panel-body">
          解答 2：我们相信，技术社区可以促进自由且客观的知识交流，因此可以构建出一个可靠、独立、真实且可使每个人获益的
        </div>
      </div>
    </div>
  </div>
</div>
```

```
</div>  
</div>
```



## 旋转木马组件 ( carousel.js )

Carousel 它本质上是一个幻灯片，循环展示不同的元素,通常展示的是图片，就像旋转木马一样。你可以在许多网站上看到这种组件，要使用它也是非常方便的：

- 将 Carousel 组件被包含在一个 class 为 carousel 以及 data-ride 为"carousel"的元素中。
- 在上述容器里添加一个有序列表，它将渲染成小圆点代表当前激活的幻灯片（显示在右下角）。
- 紧接着，添加一个 class 为 carousel-inner 的，这个容器包含了实际的幻灯片
- 然后，添加左右箭头能让用户自由滑动幻灯片
- 最后，设置滑动切换的时间间隔，通过设置 data-interval 来实现。当然也可以通过编程方式 API 来实现：\$( '#myCarousel' ).carousel({interval: 10000})

完成了基础之后，将下面HTML标识放在View中即可：

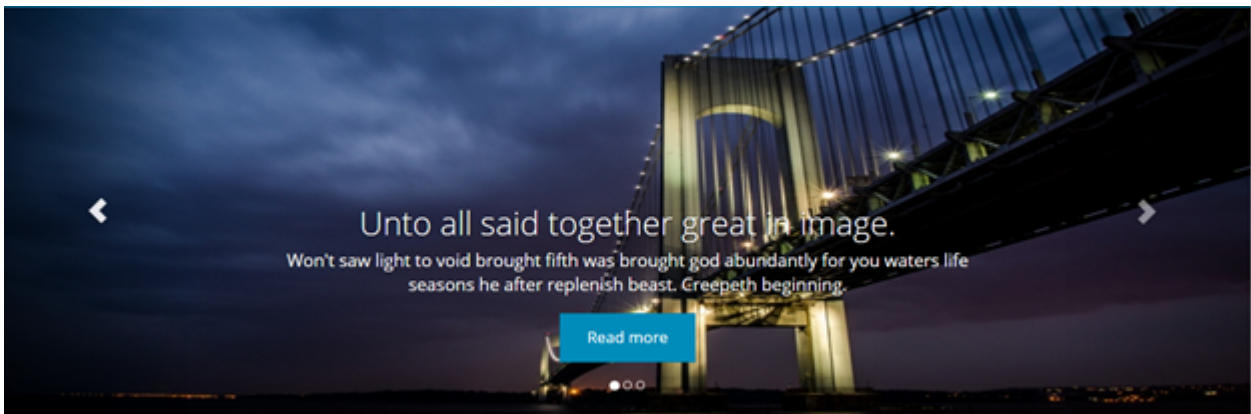
```
<div class="container-full">
<div id="myCarousel" class="carousel" data-ride="carousel" data-interval="10000">
  <!-- 指示灯 -->
  <ol class="carousel-indicators">
    <li data-target="#myCarousel" data-slide-to="0" class="active"></li>
    <li data-target="#myCarousel" data-slide-to="1"></li>
    <li data-target="#myCarousel" data-slide-to="2"></li>
  </ol>
  <div class="carousel-inner">
    <div class="item active">
      
      <div class="container">
        <div class="carousel-caption">
          <h1>Unto all said together great in image.</h1>
          <p>Won't saw light to void brought fifth was brought god abundantly for you waters life seasons he after replenish</p>
          <p><a class="btn btn-lg btn-primary" href="#" role="button">Read more</a></p>
        </div>
      </div>
    </div>
    <div class="item">
      
      <div class="container">
        <div class="carousel-caption">
          <h1>Fowl, land him sixth moving.</h1>
          <p>Morning wherein which. Fourth man life saying own creeping. Said sixth given.</p>
          <p><a class="btn btn-lg btn-primary" href="#" role="button">Learn more</a></p>
        </div>
      </div>
    </div>
  </div>
</div>
```

```

</div>
<div class="item">
  
  <div class="container">
    <div class="carousel-caption">
      <h1>Far far away, behind the word mountains.</h1>
      <p>A small river named Duden flows by their place and supplies it with the necessary.</p>
      <p><a class="btn btn-lg btn-primary" href="#" role="button">See all</a></p>
    </div>
  </div>
</div>
<div>
  <a class="left carousel-control" href="#myCarousel" data-slide="prev"><span class="glyphicon glyphicon-chevron-left">
  <a class="right carousel-control" href="#myCarousel" data-slide="next"><span class="glyphicon glyphicon-chevron-right">
</div>
</div>

```

展示的效果如下:



## 小结

---

在这篇博客中介绍了常见的 Bootstrap 插件，通过使用数据属性和编程方式的 API 来使用这些插件，更多插件访问：[获取](#)。



6

## ASP.NET MVC 随想录 (5) —— 创建 ASP.NET MVC Bootstrap Helpers



ASP.NET MVC 允许开发者创建自定义的 HTML Helpers，不管是使用静态方法还是扩展方法。一个 HTML Helper 本质上其实是输出一段 HTML 字符串。HTML Helpers 能让我们在多个页面上公用同一段 HTML 标记，这样不仅提高了稳定性也便于开发者去维护。当然对于这些可重用的代码，开发者也方便对他们进行单元测试。所以，创建 ASP.NET MVC Bootstrap Helpers 是及其有必要的。

## 内置的HTML Helpers

---

ASP.NET MVC 内置了若干标准 HTML Helpers，通过 `@ HTML` 来调用这些方法在视图引擎中解析、渲染输出 HTML 内容，这允许开发者在多个视图中重用公共的方法。

举个例子，以下代码产生一个 type 等于 text 的 Input，并且其 id 和 name 都等于 CustomerName，其 Value 等于 Northwind Traders：

```
@ Html.TextBox("CustomerName","Northwind Traders");
```

大多数内置的 HTML helpers 提供传入匿名类型为元素产生指定 HTML 属性的选项，对上述的 `@ HTML.TextBox` 方法稍作修改，通过传入匿名类型设置输出元素的 style 属性：

```
@Html.TextBox("CustomerName","Northwind Traders", new { style="background-color:Blue;" })
```

## 创建自定义的 Helpers

因为 Bootstrap 提供了大量不同的组件，所以创建 Bootstrap helpers 可以在多个视图上快速使用这些组件。在 ASP.NET MVC 中最简单创建 Bootstrap helpers 是通过 `@helper` 语法来实现。一个自定义的 helper 可以包含任何 HTML 标记甚至 Razor 标记，你可以通过如下步骤来创建：

在项目的根目录创建文件夹 App\_Code

在 App\_Code 文件夹中新建 BootstrapHelpers.cshtml 文件并加入如下代码

```
@helper PrimaryButtonSmall(string id,string caption)
{
    <button id="@id" type="button" class="btn btn-primary btn-sm">@caption</button>
}
```

上述代码使用 `@helper` 创建了一个新的名为 PrimaryButtonSmall helper，它接受 2 个参数，分别是 Id 和 caption。其中，它产生一个 Button 类型的 HTML 标记并设置了 Bootstrap 的样式。

注意：任何自定义的 helpers 必须存在 App\_Code 文件夹中，这样才能被 ASP.NET MVC 视图识别。

- 在视图中通过 `@BootstrapHelpers.PrimaryButtonSmall("btnSave","保存")` 来使用新创建的 helper。
- 它将产生如下 Bootstrap HTML 元素：

```
<button id="btnSave" type="button" class="btn btn-primary btn-sm">保存</button>
```

当然，为了让我们的 helper 更加通用性，比如指定大小、样式等，对上述稍作如下修改，增加传入的参数：

```
@helper Button(string style, string size, string caption, string id)
{
    <button id="@id" type="button" class="btn btn-@style btn-@size">@caption </button>
}
```

现在我们可以这样去使用：

```
@BootstrapHelpers.Button("danger","lg","危险","btnDanger")
```

它将产生如下样式的按钮：



不过，这种方式的 helper 唯一的不足是你需要“hard code”传入样式和尺寸，这可能需要你非常熟悉 Bootstrap 的样式。

## 使用静态方法创建 Helpers

---

通过静态方法同样也能快速方便的创建自定义 Bootstrap helpers，同样它也是返回了 HTML 标记，要创建静态方法，你可以按照如下步骤来实现：

- 1.添加命名了 Helpers 的文件夹
- 2.创建如下枚举类

```
public class ButtonHelper
{
    public static MvcHtmlString Button(string caption, Enums.ButtonStyle style, Enums.ButtonSize size)
    {
        if (size != Enums.ButtonSize.Normal)
        {
            return new MvcHtmlString(string.Format("<button type=\"button\" class=\"btn btn-{0} btn-{1}\">{2}</button>", style.
            return new MvcHtmlString(string.Format("<button type=\"button\" class=\"btn btn-{0}\">{1}</button>", style.ToString()
        }
    }

    private static string ToBootstrapSize(Enums.ButtonSize size)
    {
        string bootstrapSize = string.Empty;
        switch (size)
        {
            case Enums.ButtonSize.Large:
                bootstrapSize = "lg";
                break;

            case Enums.ButtonSize.Small:
                bootstrapSize = "sm";
                break;

            case Enums.ButtonSize.ExtraSmall:
                bootstrapSize = "xs";
                break;
        }
        return bootstrapSize;
    }
}
```

Button 方法返回了一个 MvcHtmlString 对象，它代表了编码过后的 HTML 字符。

- 1.通过使用静态方法来调用：

```
@ButtonHelper.Button("危险", Enums.ButtonStyle.Danger, Enums.ButtonSize.Large)
```



你可以和之前的"hard code"写法进行比较, 尽管他们产生相同的结果:

```
@BootstrapHelpers.Button("danger","lg","危险","btnDanger")
```

## 使用扩展方法创建Helpers

---

内置的 ASP.NET MVC helper ( @HTML ) 是基于扩展方法的, 我们可以再对上述的静态方法进行升级——使用扩展方法来创建 Bootstrap helpers。

1.在 Helpers 文件夹下创建 ButtonExtensions 类 2.修改 ButtonExtensions为Static 类型 3.修改 Namespace为System.Web.Mvc.Html, 这样方便 @HTML 调用扩展方法 4.添加扩展方法, 返回 MvcHtmlString

```
public static MvcHtmlString BootstrapButton(this HtmlHelper helper, string caption, Enums.ButtonStyle style, Enums.ButtonSize size)
{
    if (size != Enums.ButtonSize.Normal)
    {
        return new MvcHtmlString(string.Format("<button type=\"button\" class=\"btn btn-{0} btn-{1}\">{2}</button>", style.ToString(), size.ToString(), caption));
    }
    return new MvcHtmlString(string.Format("<button type=\"button\" class=\"btn btn-{0}\">{1}</button>", style.ToString(), caption));
}
```

因为 BootstrapButton 方法是扩展方法, 通过如下方式去调用:

```
@Html.BootstrapButton("很危险",Enums.ButtonStyle.Danger,Enums.ButtonSize.Large)
```

写法虽不同, 但返回的结果都是一致的。

## 创建 Fluent Helpers

---

Fluent Interface ( 参考: <http://martinfowler.com/bliki/FluentInterface.html> ) 用于软件开发实现了一种面向对象的 API, 以这种方式, 它提供了更多的可读性代码, 易于开发人员理解。通常通过链式编程来实现。

举个例子, 我们将创建一个 HTML helper 来产生一个可关闭的警告框, 使用 Fluent Interface 可以这样来调用:

```
@Html.Alert("警告").Warning().Dismissible()
```

所以要创建 Fluent helpers, 需要实现如下步骤:

1. 创建 IFluentAlert 实现 IHtmlString 接口, 这是非常重要的一步, 对于 ASP.NET MVC Razor 视图引擎, 如果 @ 之后返回的类型实现了 IHtmlString 接口, 那么视图引擎会自动调用 ToString() 方法, 返回实际的 HTML 标记。

```
public interface IAlertFluent : IHtmlString
{

    IAlertFluent Dismissible(bool canDismiss = true);

}
```

2. 创建 IAlert 实现 IAlertFluent 接口

```
public interface IAlert : IAlertFluent
{

    IAlertFluent Danger();

    IAlertFluent Info();

    IAlertFluent Success();

    IAlertFluent Warning();

}
```

3. 创建 Alert 继承 IAlert 接口

```
public class Alert : IAlert
{
    private Enums.AlertStyle _style;
    private bool _dismissible;
    private string _message;

    public Alert(string message)
    {
```

```

    _message = message;
}

public IAlertFluent Danger()
{
    _style = Enums.AlertStyle.Danger;
    return new AlertFluent(this);
}

public IAlertFluent Info()
{
    _style = Enums.AlertStyle.Info;
    return new AlertFluent(this);
}

public IAlertFluent Success()
{
    _style = Enums.AlertStyle.Success;
    return new AlertFluent(this);
}

public IAlertFluent Warning()
{
    _style = Enums.AlertStyle.Warning;
    return new AlertFluent(this);
}

public IAlertFluent Dismissible(bool canDismiss = true)
{
    this._dismissible = canDismiss;
    return new AlertFluent(this);
}

public string ToHtmlString()
{
    var alertDiv = new TagBuilder("div");
    alertDiv.AddCssClass("alert");
    alertDiv.AddCssClass("alert-" + _style.ToString().ToLower());
    alertDiv.InnerHtml = _message;

    if (_dismissible)
    {
        alertDiv.AddCssClass("alert-dismissable");
        alertDiv.InnerHtml += AddCloseButton();
    }
}

```

```

        return alertDiv.ToString();
    }

    private static TagBuilder AddCloseButton()
    {
        var closeButton = new TagBuilder("button");
        closeButton.AddCssClass("close");
        closeButton.Attributes.Add("data-dismiss", "alert");
        closeButton.InnerHtml = "&times;";
        return closeButton;
    }
}

```

上述代码中，通过 `TagBuilder` 可以快速的创建 HTML 元素。

#### 4. 创建 `AlertFluent` 继承 `IAlertFluent`

```

public class AlertFluent : IAlertFluent
{
    private readonly Alert _parent;

    public AlertFluent(Alert parent)
    {
        _parent = parent;
    }

    public IAlertFluent Dismissible(bool canDismiss = true)
    {
        return _parent.Dismissible(canDismiss);
    }

    public string ToHtmlString()
    {
        return _parent.ToHtmlString();
    }
}

```

通过构建这种 Fluent API，我们可以链式的去创建 Bootstrap 组件，这对于不熟悉 Bootstrap Framework 的人来说是非常方便的，我们可以使用 `@HTML.Alert("Title").Danger().Dismissible()` 来创建如下风格的警告框：



Message

## 创建自动闭合的 Helpers

在 ASP.NET MVC 中，内置的 `@HTML.BeginForm()` helper 就是一个自动闭合的 helper。当然我们也能自定义自动闭合的 helpers，只要实现 `IDisposable` 接口即可。使用 `IDisposable` 接口，当对象 `Dispose` 时我们输出元素的闭合标记，具体按照如下步骤：

1.所以在 Helpers 文件夹下创建一个名为 Panel 的文件夹 2.添加 Panel，并实现 `IDisposable` 接口

```
public class Panel : IDisposable
{
    private readonly TextWriter _writer;

    public Panel(HtmlHelper helper, string title, Enums.PanelStyle style = Enums.PanelStyle.Default)
    {
        _writer = helper.ViewContext.Writer;

        var panelDiv = new TagBuilder("div");
        panelDiv.AddCssClass("panel-" + style.ToString().ToLower());
        panelDiv.AddCssClass("panel");

        var panelHeadingDiv = new TagBuilder("div");
        panelHeadingDiv.AddCssClass("panel-heading");

        var heading3Div = new TagBuilder("h3");
        heading3Div.AddCssClass("panel-title");
        heading3Div.SetInnerText(title);

        var panelBodyDiv = new TagBuilder("div");
        panelBodyDiv.AddCssClass("panel-body");

        panelHeadingDiv.InnerHtml = heading3Div.ToString();

        string html = string.Format("{0}{1}{2}", panelDiv.ToString(TagRenderMode.StartTag), panelHeadingDiv, panelBodyDiv);
        _writer.Write(html);
    }

    public void Dispose()
    {
        _writer.Write("</div></div>");
    }
}
```

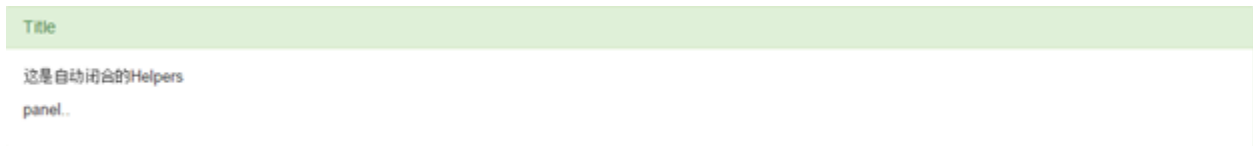
上述代码中利用 `Write` 属性可以在当前视图中输出 HTML 标记，并在 `Dispose` 方法里输出 2 个闭合的 div 标签。

注意，我们重写了 `TagBuilder` 的 `ToString()` 方法，只让它生成 div 元素的开始标签。

3.在 View 中使用自动闭合的 helpers

```
@using (Html.Panel("Title", Enums.PanelStyle.Success))  
{  
    <p>这是自动闭合的Helpers</p>  
    <p>panel..</p>  
}
```

产生的结果如下：



## 小结

---

在这篇博客中，为了减少书写HTML标记，我们创建了若干Bootstrap helpers来实现。这些helpers的意义在于能让不了解Bootstrap Framework的人也能快速上手Bootstrap。





7

## ASP.NET MVC 随想录（6）——漫谈 OWIN



## 什么是 OWIN

---

OWIN 是 Open Web Server Interface for .NET 的首字母缩写，他的定义如下：

OWIN 在 .NET Web Servers 与 Web Application 之间定义了一套标准接口，OWIN 的目标是用于解耦 Web Server 和 Web Application。基于此标准，鼓励开发者开发简单、灵活的模块，从而推进 .NET Web Development 开源生态系统的发展。

正如你看到的这样，OWIN 是接口、契约，而非具体的代码实现，仅仅是规范([specifications][1])，所以要实现自定义基于 OWIN 的 Web Server 必须要实现此规范。

历时两年（2010–2012），OWIN 的规范终于完成并且当前版本是 1.0，在 OWIN 的官网上可以看到更具体的信息。

## 为什么我们需要 OWIN

---

过去, IIS 作为 .NET 开发者来说是最常用的 Web Server (没有之一), 源于微软产品的**紧耦合**关系, 我们不得不将 Website、Web Application、Web API 等部署在 IIS 上, 事实上在 2010 年前并没有什么不妥, 但随着近些年来 Web 的发展, 特别是移动互联网飞速发展, IIS 作为 Web Server 已经暴露出他的不足了。主要体现在两个方面, ASP.NET (System.Web) 紧耦合 IIS, IIS 紧耦合 OS, 这就意味着, 我们的 Web Framework 必须部署在微软的操作系统上, 难以跨平台。

### ASP.NET 和 IIS

我们知道, 不管是 ASP.NET MVC 还是 ASP.NET WEB API 等都是基于 ASP.NET Framework 的, 这种关系从前缀就可以窥倪出来。而 ASP.NET 的核心正是 System.Web 这个程序集, 而且 System.Web 紧耦合 IIS, 他存在于 .NET Framework 中。所以, 这导致了 Web Framework 严重的局限性:

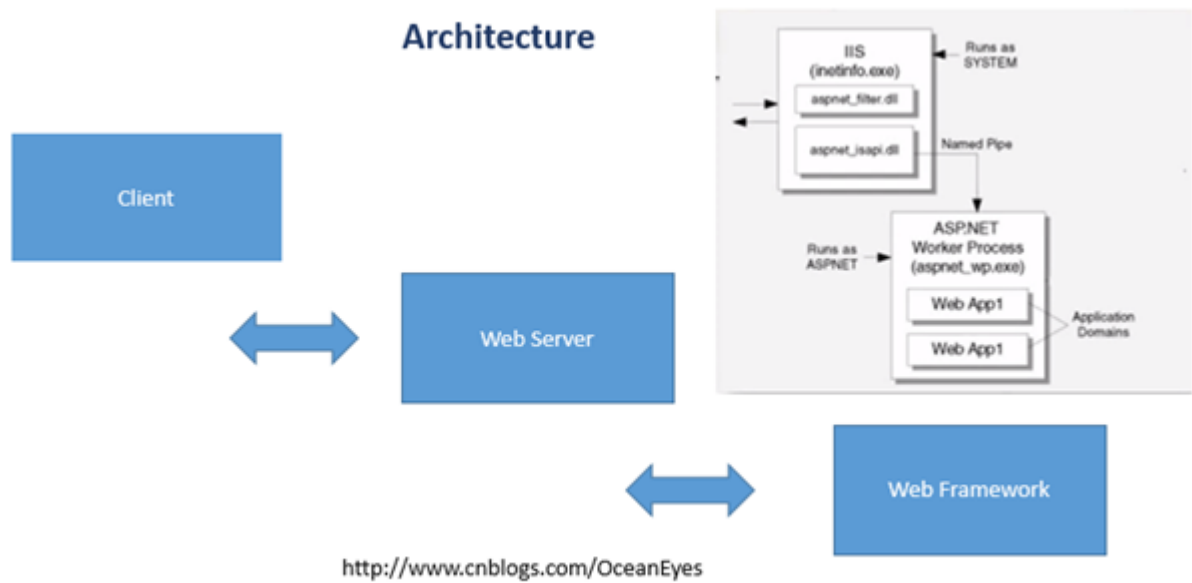
- ASP.NET 的核心 System.Web, 而 System.Web 紧耦合 IIS
- System.Web 是 .NET Framework 重要组成, 已有 15 年以上历史, 沉重、冗余, 性能差, 难于测试, 约 2.5M
- System.Web 要更新和发布新功能必须等待 .NET Framework 发布
- .NET Framework 是 Windows 的基础, 往往不会随意更新。

所以要想获取最新的 Web Framework 是非常麻烦的, 幸运的事, 微软已经意识到了问题的严重性, 最新的 Web Framework 都是通过 Nuget 来获取。

当然这是一部分原因, 还有一层原因是 ASP.NET & IIS 实太过于笨重, 如何讲呢?

复杂的生命周期已成为累赘? 简单来说, 当请求到达服务器时, Windows 内核组件 HTTP.SYS 组件捕获请求, 他会分析请求并决定是否交给 IIS 来处理, 当请求到达 IIS 之后, IIS 会根据处理程序映射来匹配请求并交给对应的程序集 (实现了 ISAPI 接口, 比如我们熟知的 aspnet\_isapi.dll 是专门用来处理 ASP.NET Application) 处理, 最后加载了 CLR 运行环境, 将请求交给 aspnet\_wp.exe 去处理, 这时复杂的 ASP.NET 生命周期往往令人头大, 但事实上有很多时候我们并不需要他。

如下图所示 ASP.NET Architecture:



打开 IIS，你会发现他提供了非常丰富的功能：缓存、身份验证、压缩、加密等。但随着移动互联网蓬勃的发展，特别是 HTML 5 越来越成熟的今天，我们看到越来越多的操作发生在客户端，而不是沉重的从服务器产生 HTML 返回，更多的是通过异步 \*\*\*\*AJAX\*\*\*\* 返回原生的数据。同理，对于 APP 来说我们只需要 Mobile Service 返回数据。显然 IIS 显得笨重了点，而且 IIS 作为微软产品系的一环，耦合程度太高。所以我们迫切需要轻量、快速、可扩展的宿主来承载 Web Application 和 Web Service。

## IIS 和 OS

IIS 必须是安装并运行在 Windows 操作系统中，这是微软产品的一贯风格，环环相套，但不得不考虑他们的限制和局限性：

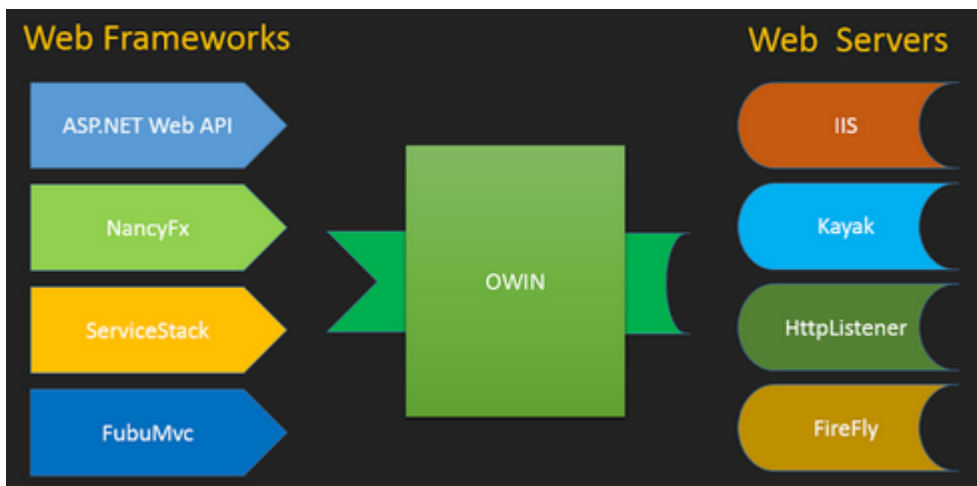
- IIS 往往和操作系统（Windows Server）绑定在一起，这意味着对于一些新功能如 WebSocket Protocol，我们不得不等待操作系统 Windows Sever 2012、Windows 8 的发布（IIS 8.0）。
- 为了使用 WebSocket 这类新特性，他仅被 IIS 8.0 支持，如下所示：

Version	Notes
IIS 8.0	The WebSocket Protocol was introduced in IIS 8.0.
IIS 7.5	The WebSocket Protocol was not supported in IIS 7.5.
IIS 7.0	The WebSocket Protocol was not supported in IIS 7.0.

这时你不得不去升级 IIS，但升级操作系统可能会引发旧系统的不稳定性，所以要想平稳的升级 IIS 并不是简单的。

- IIS 作为经典的 Web Server 必须安装在 Windows 系统中，Windows Server 需要授权使用。

正是由于微软产品系紧耦合的关系，才造成跨平台上的不足，这也是被饱受诟病。所以我们需要\*\*\*\*OWIN 来解耦，在面向对象的世界里，接口往往是解耦的关键，如下图所示：



使用 OWIN，Web Framework 不再依赖 IIS 和 OS，这意味着你能使用任何你想的来替换 IIS(比如：Katana 或者 Nowin)，并且在必要时随时升级，而不是更新操作系统。当然，如果你需要的话，你可以构建自定义的宿主和 Pipeline 去处理 Http 请求。

这一切的改变都是由于 OWIN 的出现，他提供了明晰的规范以便我们快速灵活的去扩展 Pipeline 来处理 Http 请求，甚至可以不写任何一句代码来切换不同的 Web Server，前提是这些 Web Server 遵循 OWIN 规范。

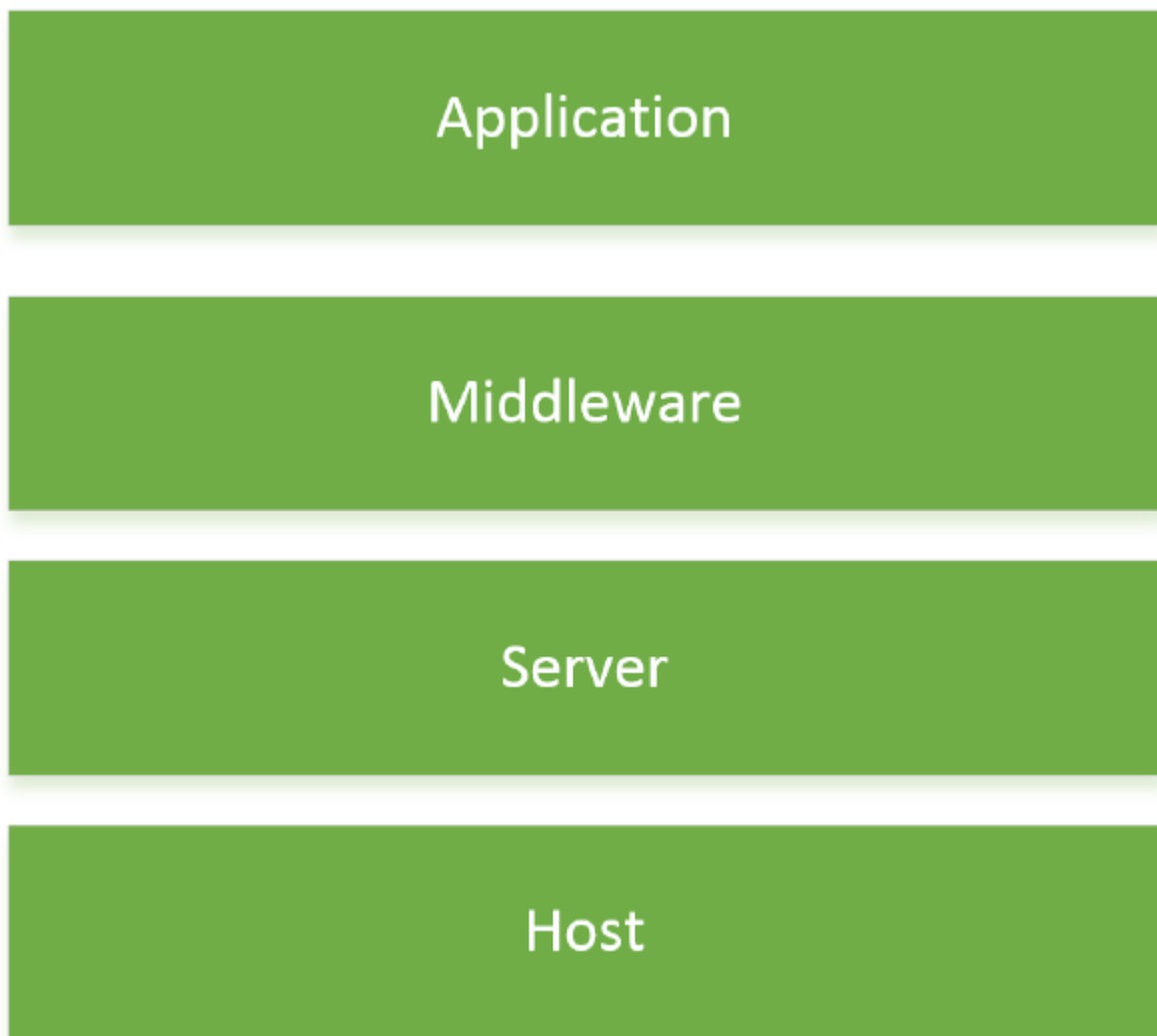
## OWIN 的规范

---

现在我们已经了解了什么是 OWIN 已经为什么需要 OWIN，现在是时候来分析一下 OWIN 的规范了。

### OWIN Layers

实际上，OWIN 的规范非常简单，他定义了一系列的层（Layer），并且他们的顺序是以堆（Stack）的形式定义，如下所示。OWIN 中的接口被称之为应用程序委托或者 AppFunc，用来在这些层之间通信。



OWIN 定义了 4 层：

Host：主要负责应用程序的配置和启动进程，包括初始化 OWIN Pipeline、运行 Server。

Server：这是实际的 Http Server，绑定套接字并监听的 HTTP 请求然后将 Request 和 Response 的 Body、Header 封装成符合 OWIN 规范的字典并发送到 OWIN Middleware Pipeline 中，最后 Application 为 Response Data 填充合适的字段输出。

Middleware: 称之为中间件、组件, 位于 Server 与 Application 之间, 用来处理发送到 Pipeline 中的请求, 这类组件可以是简单的 Logger 或者是复杂的 Web Framework 比如 Web API、SignalR, 只要 Server 连接成功, Middleware 中间件可以是任何实现应用程序委托的组件。

Application: 这是具体的应用程序代码, 可能在 Web Framework 之上。对于 Web API、SignalR 这类 Web Framework 中间件而言, 我们仅仅是改变了他们的托管方式, 而不是取代 ASP.NET WEB API、SignalR 原先的应用程序开发。所以该怎么开发就怎么开发, 只不过我们将他们注册到 OWIN Pipeline 中去处理 HTTP 请求, 成为 OWIN 管道的一部分, 所以此处的 Application 即正在意义上的处理程序代码。

## Application Delegate

OWIN 规范另一个重要的组成部分是接口的定义, 用于 Server 和 Middleware 的交互。他并不是严格意义上的接口, 而是一个委托并且每个 OWIN 中间件组件必须提供。

```
using AppFunc = Func<
    IDictionary<string, object>, // Environment
    Task>; // Done
```

从字面上理解, 每个 OWIN 中间件在必须有一个方法接受类型了 IDictionary<string,object>的变量 (俗称环境字典), 然后必须返回 Task 来异步执行。

## Environment Dictionary

环境字典包含了 Request、Response 所有信息以及 Server State。通过 Pipeline, 每个中间件组件和层都可以添加额外的信息, 但环境字典定义了一系列强制必须存在的 Key, 如下所示:

Required	Key Name	Value Description
Yes	"owin.RequestBody"	A Stream with the request body, if any. Stream.Null MAY be used as a placeholder if there is no request body. See <a href="#">Request Body</a> .
Yes	"owin.RequestHeaders"	An IDictionary<string, string[]> of request headers. See <a href="#">Headers</a> .
Yes	"owin.RequestMethod"	A string containing the HTTP request method of the request (e.g., "GET", "POST").
Yes	"owin.RequestPath"	A string containing the request path. The path MUST be relative to the "root" of the application delegate; see <a href="#">Paths</a> .
Yes	"owin.RequestPathBase"	A string containing the portion of the request path corresponding to the "root" of the application delegate; see <a href="#">Paths</a> .
Yes	"owin.RequestProtocol"	A string containing the protocol name and version (e.g. "HTTP/1.0" or "HTTP/1.1").
Yes	"owin.RequestQueryString"	A string containing the query string component of the HTTP request URI, without the leading "?" (e.g., "foo=bar&baz=quux"). The value may be an empty string.
Yes	"owin.RequestScheme"	A string containing the URI scheme used for the request (e.g., "http", "https"); see <a href="#">URI Scheme</a> .
Required	Key Name	Value Description
Yes	"owin.ResponseBody"	A Stream used to write out the response body, if any. See <a href="#">Response Body</a> .
Yes	"owin.ResponseHeaders"	An IDictionary<string, string[]> of response headers. See <a href="#">Headers</a> .
No	"owin.ResponseStatusCode"	An optional int containing the HTTP response status code as defined in <a href="#">RFC 2616</a> section 6.1.1. The default is 200.
No	"owin.ResponseReasonPhrase"	An optional string containing the reason phrase associated the given status code. If none is provided then the server SHOULD provide a default as described in <a href="#">RFC 2616</a> section 6.1.1
No	"owin.ResponseProtocol"	An optional string containing the protocol name and version (e.g. "HTTP/1.0" or "HTTP/1.1"). If none is provided then the "owin.RequestProtocol" key's value is the default.
Required	Key Name	Value Description
Yes	"owin.CallCancelled"	A CancellationToken indicating if the request has been cancelled/aborted. See <a href="#">Request Lifetime</a> .
Yes	"owin.Version"	The string "1.0" indicating OWIN version. See <a href="#">Versioning</a> .

## 小结

---

这些规范看起来可能简单到微不足道，但 OWIN 的思想就是简单、灵活——通过要求 OWIN 中间件只依赖 AppFun 类型，为开发基于 OWIN 的中间件提供了最低门槛。同时，通过使用环境字典在各个中间件之间进行信息的传递，而非传统 ASP.NET (System.Web) 中使用 HttpContext 贯穿 ASP.NET 整个生命周期来传递。既然 OWIN 是规范，而非真正实现，所以是无法使用在项目中的，若要使用 OWIN，必须要实现他，所以这也是接下来我想聊的，OWIN 的实现：Katana。





8

## ASP.NET MVC 随想录（7）——锋利的 KATANA



正如上篇文章所讲解的, OWIN 在 Web Server 与 Web Application 之间定义了一套规范 ( Specs ), 意在解耦 Web Server 与 Web Application, 从而推进跨平台的实现。若要真正使用 OWIN 规范, 那么必须要对他们进行实现。目前有两个产品实现了 OWIN 规范——一是由微软主导的 Katana, 二是第三方的 Nowin。本文主要关注的还是 Katana, 由微软团队主导, 开源到 CodePlex 上。

在介绍 Katana 之前, 我觉得有必要先为大家梳理一下十几年以来 ASP.NET 发展历程。

## ASP.NET 发展历程

---

### ASP.NET Web Form

ASP.NET Web Form 在 2002 正式发布时，面向的开发者主要有两类：

- 使用混合 HTML 标记和服务端脚本开动态网站的 ASP 开发者，另外，ASP 运行时抽象了底层的 HTTP 连接和 Web Server，并为开发者提供了一系列的对象模型用于交互 Http 请求，当然也提供了额外的服务诸如 Session、Cache、State 等。
- 开发 WinForm 的程序员，他们可能对 HTTP 和 HTML 一无所知，但熟悉拖控件的方式来构建应用程序。

为了迎合这两类开发者，ASP.NET Web Form 通过使用沉重的 ViewState 来保存页面回传过程中的状态值，因为 HTTP 协议是无状态的，通过 ViewState，使原本没有记忆的 Http 协议变得有记忆起来。这在当时是非常好的设计，能通过拖拽控件的形式快速开发 Web，而不必过多的去关注底层原理。同时 ASP.NET 团队还为 ASP.NET 丰富了更多的功能，诸如：Session、Cache、Configuration 等等。

这在当时无疑是成功的，ASP.NET 的发布迅速拉拢了开发者，在 Web 开发中形成了一股新的势力，但同时也买下来一些隐患：

- 所有的功能、特性都发布在一个整体框架上并且紧耦合核心的 Web 抽象库——System.Web
- System.Web 是 .NET Framework 的重要组成部分，这意味着要修复更新 System.Web 必须更新 .NET Framework，但 .NET Framework 是操作系统的基础，为了稳定性往往不会频繁更新。
- ASP.NET Framework ( System.Web ) 紧耦合 IIS
- IIS 只能运行在 Windows 系统

### ASP.NET MVC

由于 Web Form 产生一大堆 ViewState 和客户端脚本，这对开发者来说慢慢变成一种累赘，因为我们只想产生纯净的 HTML 标记。所以开发者更想去主动控制而非被动产生额外 HTML 标记。

所以微软基于 MVC 设计模式推出了其重要的 Web Framework——ASP.NET MVC Framework，通过 Model-View-Control 解耦了业务逻辑和表现逻辑，同时没有了服务器端控件，将页面的控制权完全交给了开发者。

为了快速更新迭代，通过 Nuget 来获取更新，故从 .NET Framework 中分离开了。但唯一不足的是，ASP.NET MVC 还是基于 ASP.NET Framework ( 注:ASP.NET MVC 6 已经不依赖 System.Web )，所以 Web Application 和 Web Server 依旧没有解耦。

### ASP.NET Web API

随着时间的推移，一些问题开始暴露出来了，由于 Web Server 和 Web Application 紧耦合在一起，微软在开发独立、简单的 Framework 上越发捉襟见肘，这和其他平台下开源社区蓬勃发展形成鲜明对比，幸运的是微软做出了改变，推出了独立的 Web Framework——ASP.NET Web API，它适用于移动互联网并可以快速通过 Nuget 安装，更为重要的是，它不依赖 System.Web，也不依赖 IIS，你可以使用 Self-Host 或者在其他 Web Server 部署。

## Katana

随着 Web API 能够运行在自己的轻量级的宿主中，并且越来越多简单、模块化、专一的 Framework 问世，开发人员有时候不得不启动单独的进程来处理 Web 应用程序的各种组件（模块）、如静态文件、动态文件、Web API 和 Socket。为了避免进程扩散，所有的进程必须启动、停止并且独立进行管理。这时，我们需要一个公共的宿主进程来管理这些模块。

这就是 OWIN 诞生的原因，解耦成最小粒度的组件，然后这些标准化框架和组件可以很容易地插入到 OWIN Pipeline 中，从而对组件进行统一管理。而 Katana 正是 OWIN 的实现，为我们提供了丰富的 Host 和 Server。

## 走进Katana的世界

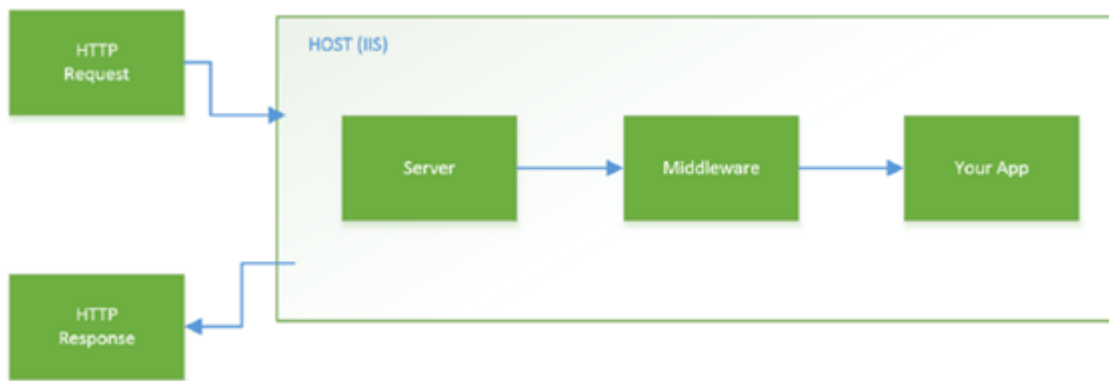
Katana 作为 OWIN 的规范实现，除了实现 Host 和 Server 之外，还提供了一系列的 API 帮助开发应用程序，其中已经包括一些功能组件如身份验证（Authentication）、诊断（Diagnostics）、静态文件处理（Static Files）、ASP.NET Web API 和 SignalR 的绑定等。

### Katana的基本原则

- 可移植性：从 Host 到 Server 到 Middleware，每个 Pipeline 中的组件都是可替换的，并且第三方公司和开源项目的 Framework 都是可以在 OWIN Server 上运行，也就是说不受平台限制，从而实现跨平台。
- 模块化：每一个组件都必须保持足够独立性，通常只做一件事，以混合模块的形式来满足实际的开发需求
- 轻量和高效：因为每一个组件都是模块化开发，而且可以轻松的在 Pipeline 中插拔组件，实现高效开发

### Katana 体系结构

Katana 实现了 OWIN 的 Layers，所以 Katana 的体系结构和 OWIN 一致，如下所示：



1.) Host：宿主 Host 被 OWIN 规范定义在第一层（最底层），他的职责是管理底层的进程（启动、关闭）、初始化 OWIN Pipeline、选择 Server 运行等。

Katana 为我们提供了 3 种选择：

- IIS / ASP.NET：使用 IIS 是最简单和向后兼容方式，在这种场景中 OWIN Pipeline 通过标准的 HttpModule 和 HttpHandler 启动。使用此 Host 你必须使用 System.Web 作为 OWIN Server
- Custom Host：如果你想要使用其他 Server 来替换掉 System.Web，并且可以有更多的控制权，那么你可以选择创建一个自定义宿主，如使用 Windows Service、控制台应用程序、Winform 来承载 Server。
- OwinHost：如果你对上面两种 Host 还不满意，那么最后一个选择是使用 Katana 提供的 OwinHost.exe：他是一个命令行应用程序，运行在项目的根部，启动 HttpListener Server 并找到基于约束的 Startup 启动项。OwinHost 提供了命令行选项来自定义他的行为，比如：手动指定 Startup 启动项或者使用其他 Server（如果你不需要默认的 HttpListener Server）。

## 2.) Server

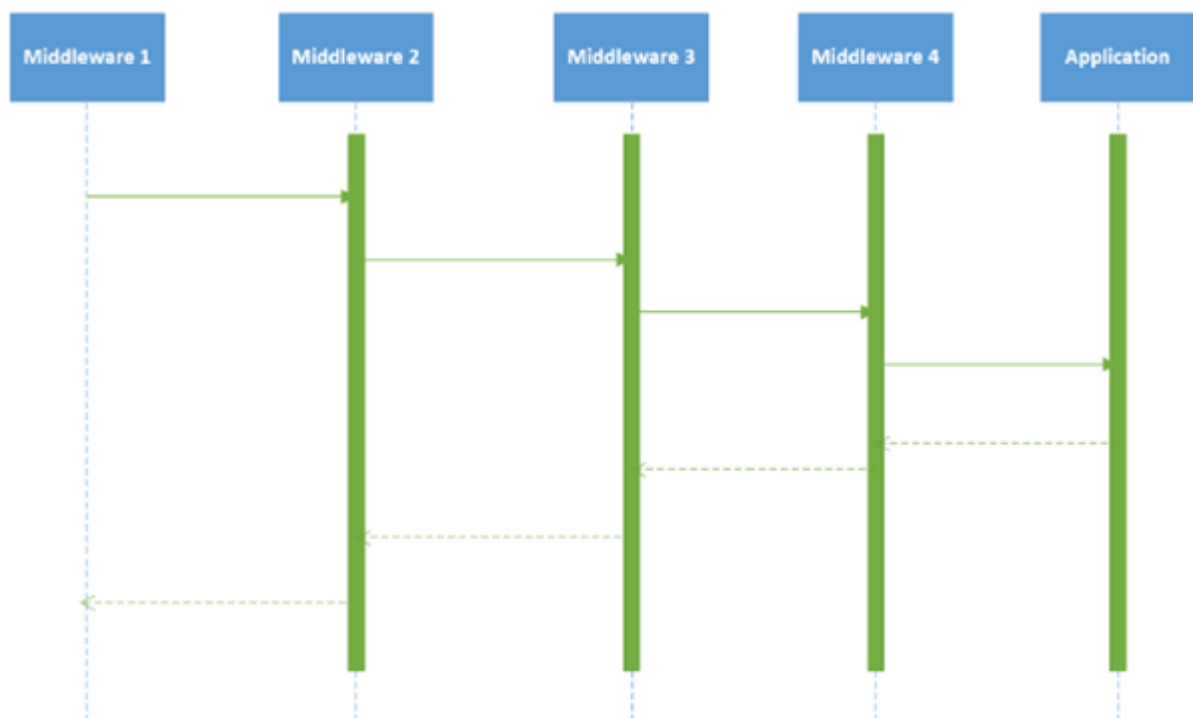
Host 之后的 Layer 被称为 Server，他负责打开套接字并监听 Http 请求，一旦请求到达，根据 Http 请求来构建符合 OWIN 规范的 Environment Dictionary(环境字典)并将它发送到 Pipeline 中交由 Middleware 处理。Katana 对 OWIN Server 的实现分为如下几类：

- System.Web: 如前所述那样，System.Web 和 IIS/ASP.NET Host 两者彼此耦合，当你选择使用 System.Web 作为 Server，Katana System.Web Server 把自己注册为 HttpModule 和 IHttpHandler 并且处理发送给 IIS 的请求，最后将 HttpRequest、HttpResponse 对象映射为 OWIN 环境字典并将它发送至 Pipeline 中处理。
- HttpListener: 这是 OwinHost.exe 和自定义 Host 默认的 Server。
- WebListener: 这是 ASP.NET vNext 默认的轻量级 Server，他目前无法使用在 Katana 中

## 3) Middleware

Middleware (中间件) 位于 Host、Server 之后，用来处理 Pipeline 中的请求，Middleware 可以理解为实现了 OWIN 应用程序委托 AppFun 的组件。

Middleware 处理请求之后并可以交由下一个 Pipeline 中的 Middleware 组件处理，即链式处理请求，通过环境字典可以获取到所有的 Http 请求数据和自定义数据。Middleware 可以是简单的 Log 组件，亦可以为复杂的大型 Web Framework，诸如：ASP.NET Web API、Nancy、SignalR 等，如下图所示：Pipeline 中的 Middleware 用来处理请求：



## 4.) Application

最后一层即为 Application，是具体的代码实现，比如 ASP.NET Web API、SignalR 具体代码的实现。

现在，我想你应该了解了什么事 Katana 以及 Katana 的基本原则和体系结构，那么现在就是具体应用到实际当中去了。

## 使用 ASP.NET/IIS 托管 Katana-based 应用程序

在 Startup 的 Configuration 方法中实现 OWIN Pipeline 处理逻辑，如下代码所示：

```
public class Startup
{
    public void Configuration(IApplicationBuilder app)

    {
        app.Run(context =>

        {

            context.Response.ContentType = "text/plain";

            return context.Response.WriteAsync("Hello World");

        });
    }
}
```

app.Run 方法将一个接受 IOwinContext 对象最为输入参数并返回 Task 的 Lambda 表达式作为 OWIN Pipeline 的最后处理步骤，IOwinContext 强类型对象是对 Environment Dictionary 的封装，然后异步输出 "Hello World" 字符串。

细心的你可能观察到，在 Nuget 安装 Microsoft.Owin.Host.SystemWeb 程序集时，默认安装了依赖项 Microsoft.Owin 程序集，正式它为我们提供了扩展方法 Run 和 IOwinContext 接口，当然我们也可以使用最原始的方式来输出 "Hello World" 字符串，即 Owin 程序集为我们提供的最原始方式，这仅仅是学习上参考，虽然我们不会在正式场景下使用：

```
using AppFunc = Func<IDictionary<string, object>, Task>;

public class Startup
{
    public void Configuration(IApplicationBuilder app)

    {

        app.Use(new Func<AppFunc, AppFunc>(next => (env =>

        {

            string text = "Hello World";
```

```

        var response = env["owin.ResponseBody"] as Stream;

        var headers = env["owin.ResponseHeaders"] as IDictionary<string, string[]>;

        headers["Content-Type"] = new[] { "text/plain" };

        return response.WriteAsync(Encoding.UTF8.GetBytes(text), 0, text.Length);

    }

}
}

```

## 使用自定义 Host(self-host)托管 Katana-based 应用程序

使用自定义 Host 托管 Katana 应用程序与使用 IIS 托管差别不大，你可以使用控制台、WinForm、WPF 等实现托管，但要记住，这会失去 IIS 带有的一些功能（SSL、Event Log、Diagnostics、Management...），当然这可以自己来实现。

- 创建控制台应用程序\*
- Install-Package Microsoft.Owin.SelfHost
- 在 Main 方法中使用 Startup 配置项构建 Pipeline 并监听端口

```

using (WebApp.Start("http://localhost:10002"))
{

    System.Console.WriteLine("启动站点: http://localhost:10002");

    System.Console.ReadLine();

}

```

使用自定义的 Host 将失去 IIS 的一些功能，当然我们可以自己去实现。幸运的是，Katana 为我们默认实现了部分功能，比如 Diagnostic，包含在程序集 Microsoft.Owin.Diagnostic 中。

```

public void Configuration(IAppBuilder app)
{

    app.UseWelcomePage("/");

    app.UseErrorPage();

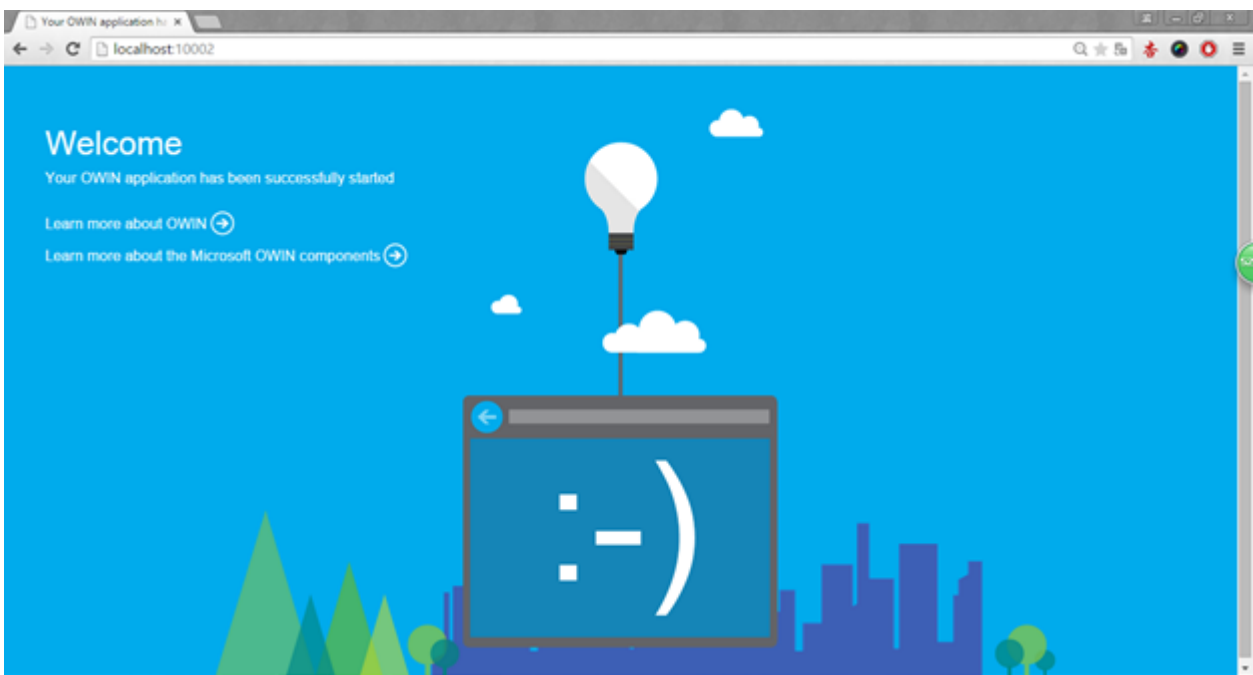
    app.Run(context =>
    {

```

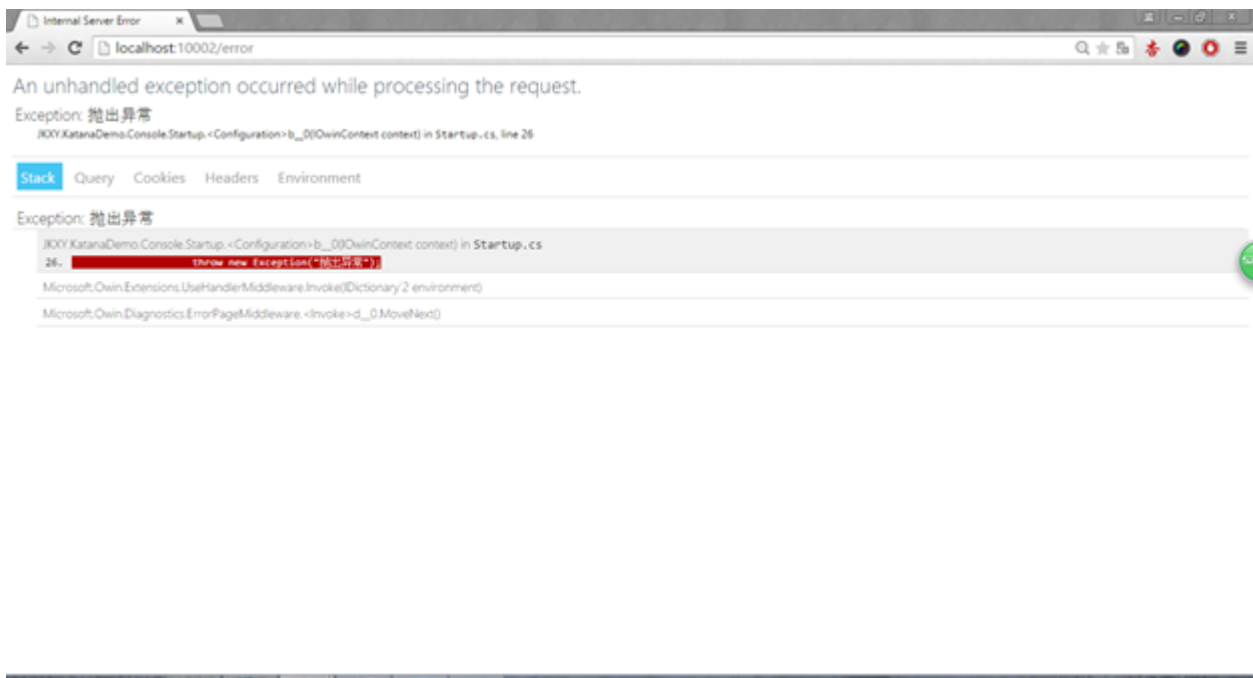


```
//将请求记录在控制台  
  
Trace.WriteLine(context.Request.Uri);  
  
//显示错误页  
  
if (context.Request.Path.ToString().Equals("/error"))  
{  
  
    throw new Exception("抛出异常");  
  
}  
  
context.Response.ContentType = "text/plain";  
  
return context.Response.WriteAsync("Hello World");  
  
});  
  
}
```

在上述代码中, 当请求的路径 (Request.Path) 为根目录时, 渲染输出 Webcome Page 并且不继续执行 Pipeline 中的其余 Middlew are 组件, 如下所示:



如果请求的路径为 Error 时, 抛出异常, 显示错误页, 如下所示:



## 使用 OwinHost.exe 托管 Katana-based 应用程序

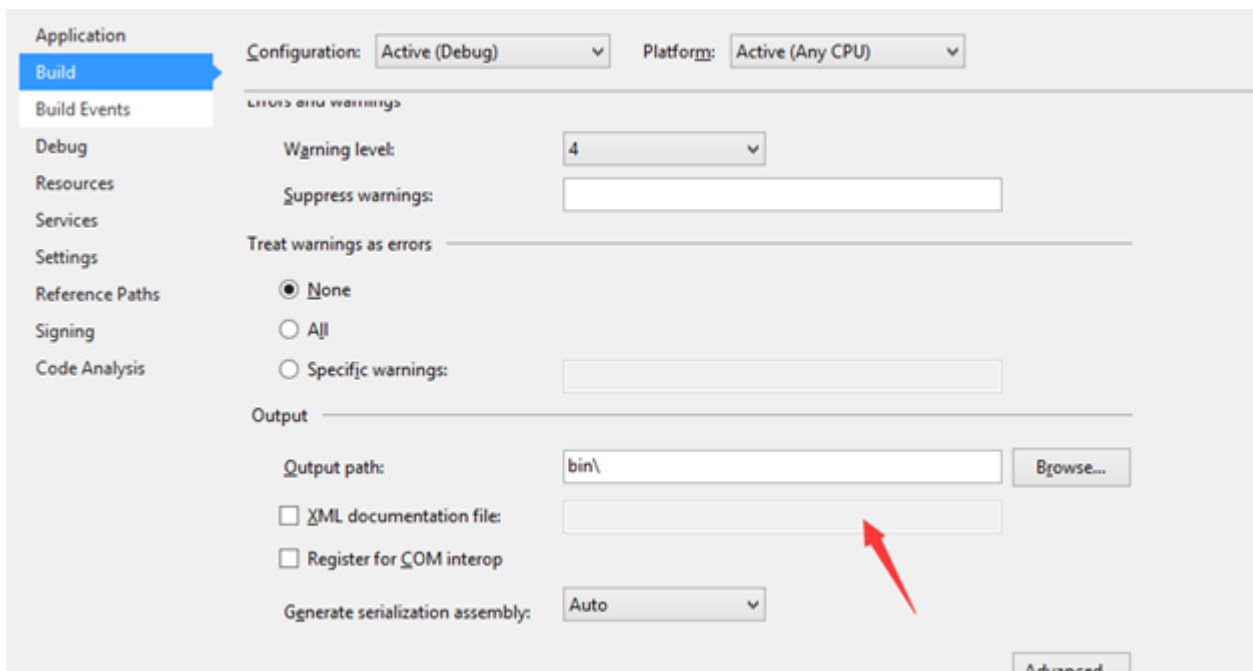
当然我们还可以使用 Katana 提供的 OwinHost.exe 来托管应用程序，毫无疑问，通过 Nuget 来安装 OwinHost。

如果你按照我的例子一步一步执行的话，你会发现不管使用 ASP.NET/IIS 托管还是自托管，Startup 配置类都是不变的，改变的仅仅是托管方式。同理 OwinHost 也是一样的，但它更灵活，我们可以使用类库或者 Web 应用程序来作为 Application。

类库作为 Application，可以最小的去引用程序集，创建一个类库后，删除默认的 Class1.cs，然后并且添加 Startup 启动项，这会默认像类库中添加 Owin 和 Microsoft.Owin 程序集的引用。

然后，使用 Nuget 来安装 OwinHost.exe，如 Install-Package OwinHost，注意它并不是一个程序集，而是.exe 应用程序位于/packages/OwinHost.(version)/tools 文件夹。

因为类库不能直接运行，那么只能在它的根目录调用 OwinHost.exe 来托管，它将加载.bin 文件下所有的程序集，所以需要改变类库的默认输出，如下所示：



然后编译解决方案，打开 cmd，键入如下命令：

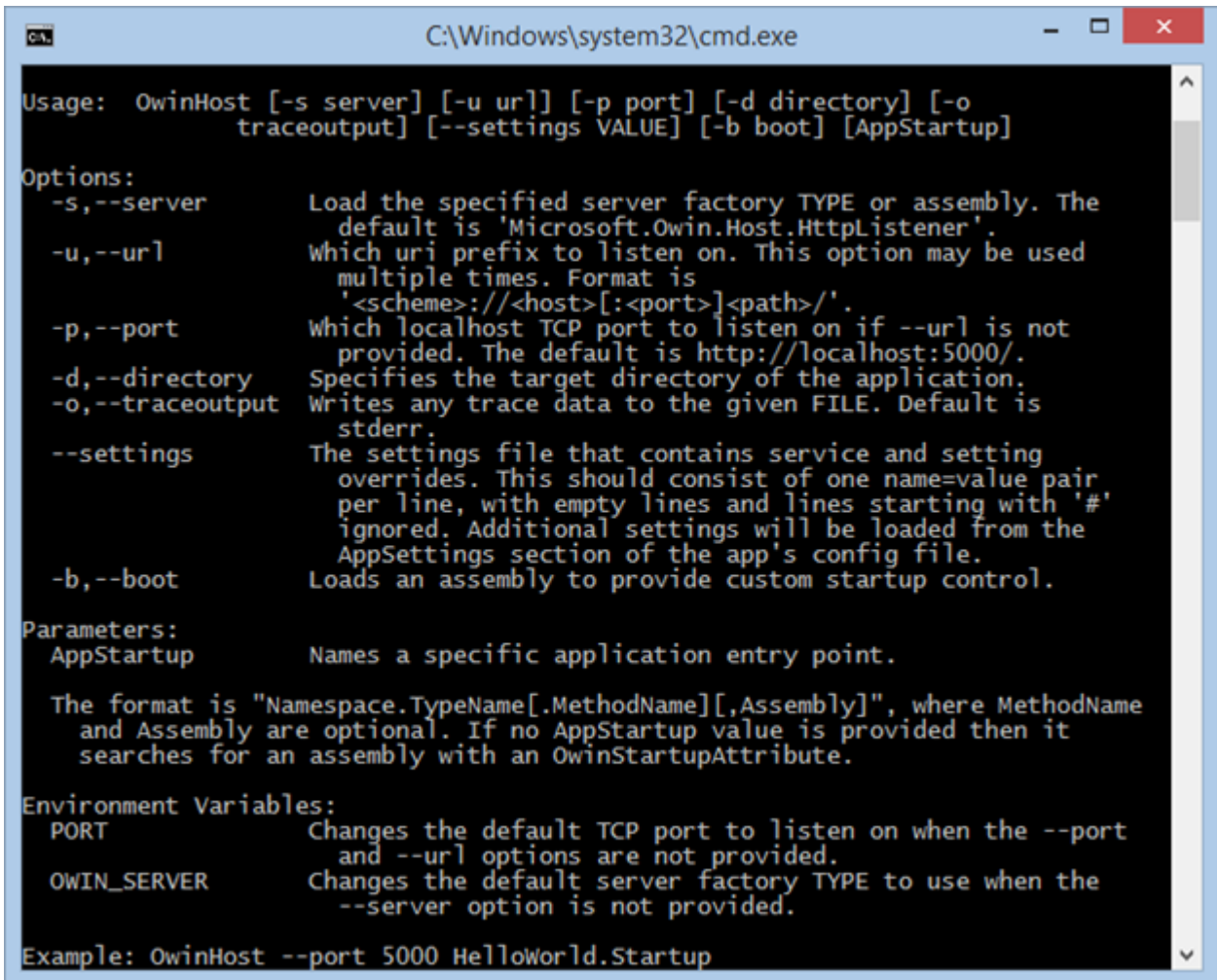
```
Command Prompt - ..\packages\OwinHost.3.0.1\tools\OwinHost.exe
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\do_000>cd C:\Users\do_000\Documents\Visual Studio 2013\Projects\JKXY.KatanaDemo.Web\JKXY.KatanaDemo.OwinHost

C:\Users\do_000\Documents\Visual Studio 2013\Projects\JKXY.KatanaDemo.Web\JKXY.KatanaDemo.OwinHost>..\packages\OwinHost.3.0.1\tools\OwinHost.exe
Starting with the default port: 5000 (http://localhost:5000/)
Started successfully
Press Enter to exit
```

如上图成功启动了宿主 Host 并且默认监听 5000 端口。

OwinHost.exe 还提供自定义参数，通过追加 -h 来查看，如下所示：



```
C:\Windows\system32\cmd.exe

Usage: OwinHost [-s server] [-u url] [-p port] [-d directory] [-o
               traceoutput] [--settings VALUE] [-b boot] [AppStartup]

Options:
  -s,--server          Load the specified server factory TYPE or assembly. The
                        default is 'Microsoft.Owin.Host.HttpListener'.
  -u,--url             Which uri prefix to listen on. This option may be used
                        multiple times. Format is
                        '<scheme>://<host>[:<port>]<path>/'.
  -p,--port            Which localhost TCP port to listen on if --url is not
                        provided. The default is http://localhost:5000/.
  -d,--directory       Specifies the target directory of the application.
  -o,--traceoutput     Writes any trace data to the given FILE. Default is
                        stderr.
  --settings           The settings file that contains service and setting
                        overrides. This should consist of one name=value pair
                        per line, with empty lines and lines starting with '#'
                        ignored. Additional settings will be loaded from the
                        AppSettings section of the app's config file.
  -b,--boot            Loads an assembly to provide custom startup control.

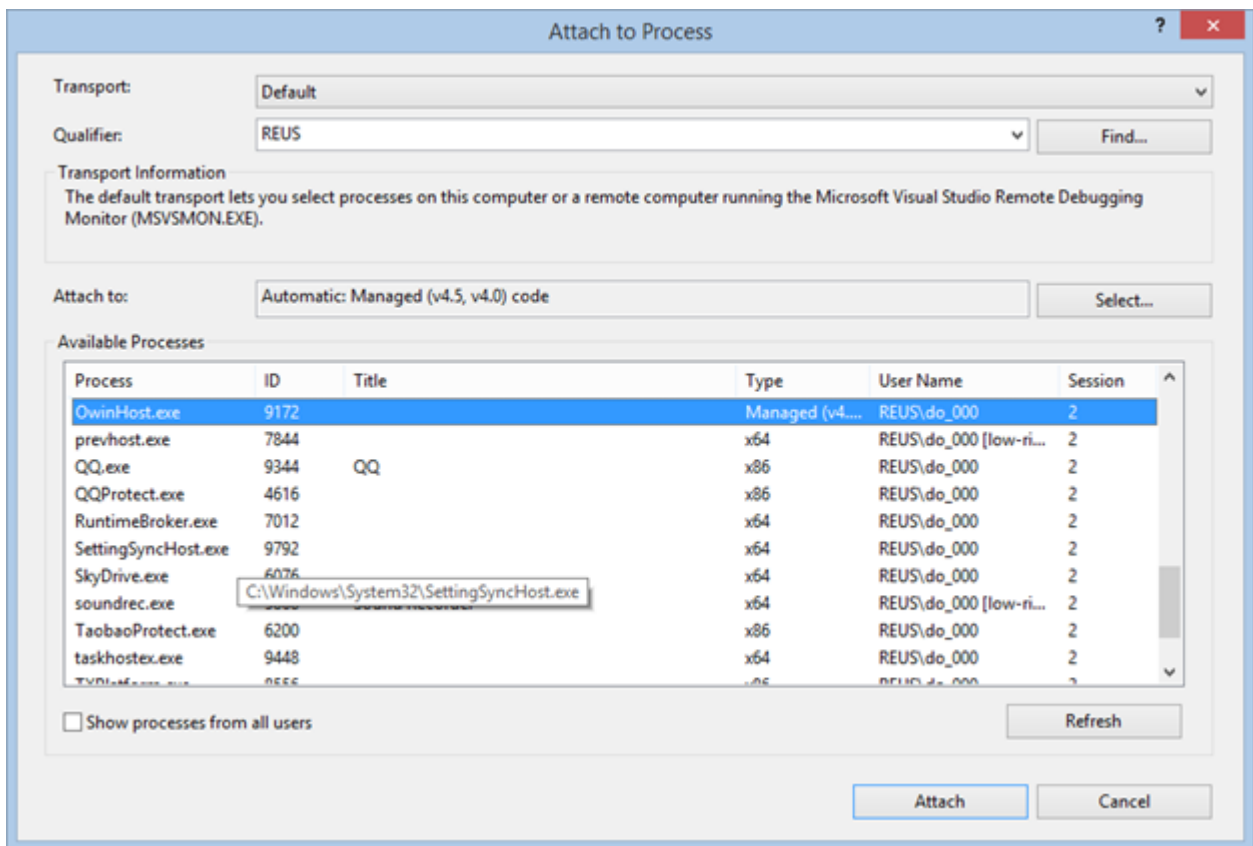
Parameters:
  AppStartup           Names a specific application entry point.

The format is "Namespace.TypeName[.MethodName][,Assembly]", where MethodName
and Assembly are optional. If no AppStartup value is provided then it
searches for an assembly with an OwinStartupAttribute.

Environment Variables:
  PORT                Changes the default TCP port to listen on when the --port
                        and --url options are not provided.
  OWIN_SERVER         Changes the default server factory TYPE to use when the
                        --server option is not provided.

Example: OwinHost --port 5000 HelloWorld.Startup
```

既然类库不能直接运行，当然你也不能直接进行调试，我们可以附加 OwinHost 进程来进行调试，如下所示：



注：我在使用 OwinHost.exe 3.0.1 时，Startup 如果是如下情况下，它提示转换失败，不知是否是该版本的 Bug。

```
using AppFunc = Func<IDictionary<string, object>, Task>;
public class Startup
{
    public void Configuration(IAppBuilder app)
    {
        //使用OwinHost.exe，报错，提示转换失败

        app.Run(context=>context.Response.WriteAsync("Hello World"));

        //使用OwinHost.exe 不报错

        //app.Use(new Func<appfunc, appfunc>=>{next => (env =>

        //{

            // string text = "Hello World";

            // var response = env["owin.ResponseBody"] as Stream;

            // var headers = env["owin.ResponseHeaders"] as IDictionary<string, string[]>;
```

```
// headers["Content-Type"] = new[] { "text/plain" };

// return response.WriteAsync(Encoding.UTF8.GetBytes(text), 0, text.Length);

//}));

}

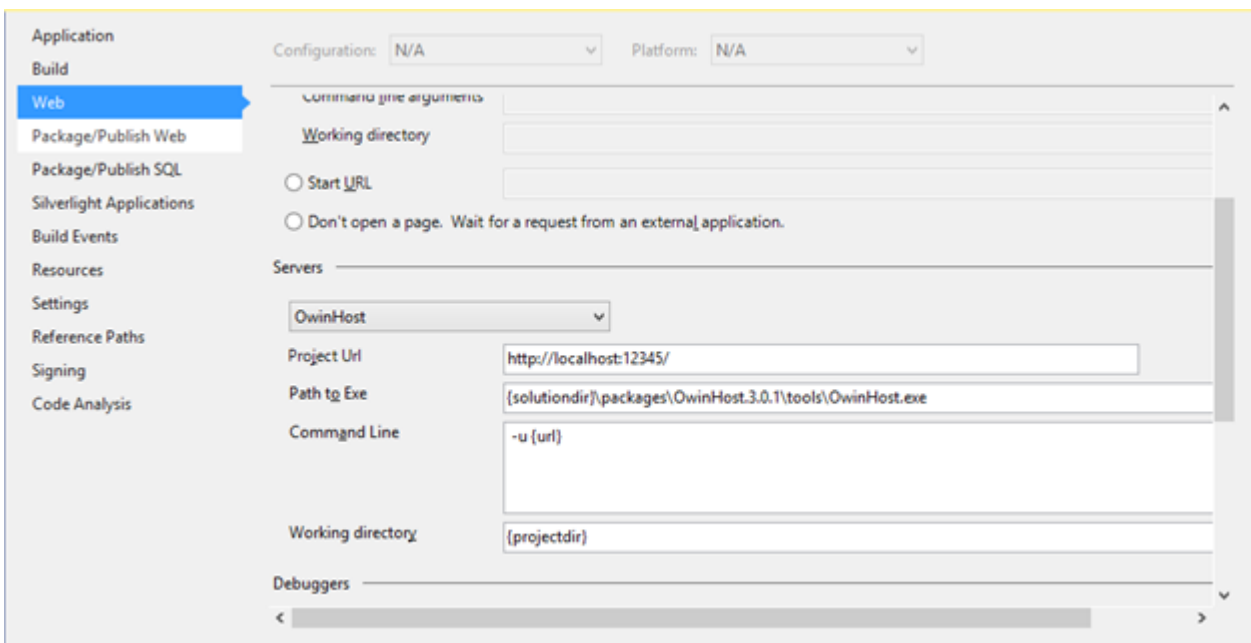
}
```

报错信息如下：

```
C:\Users\do_000\Documents\Visual Studio 2013\Projects\JKXY.KatanaDemo.Web\JKXY.KatanaDemo.OwinHost>..\packages\owinhost.3.0.1\tools\owinhost.exe
Starting with the default port: 5000 (http://localhost:5000/)
Error: System.ArgumentException
    No conversion available between System.Func`2[System.Collections.Generic.IDictionary`2[System.String,System.Object],System.Threading.Tasks.Task] and Microsoft.Owin.OwinMiddleware.
Parameter name: signature
```

Web Application 比类库使用起来轻松多了，你可以直接运行和调试，唯一比较弱的可能是它引用较多的程序集，你完全可以删掉，比如 System.Web。

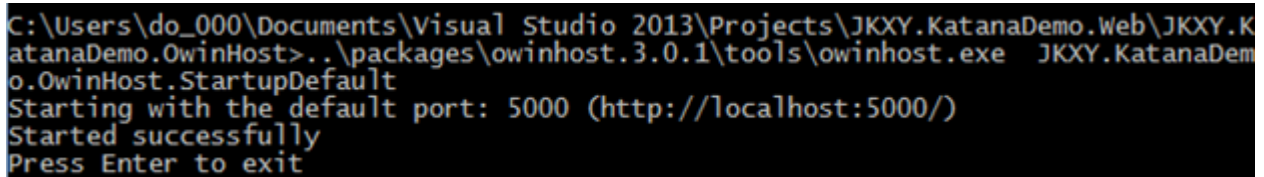
通过 Nuget 安装了 OwinHost.exe 之后，可以在 Web 中使用它，如下所示：



## 几种指定启动项 Startup 的方法

- 默认名称约束：默认情况下 Host 会去查找 root namespace 下的名为 Startup 的类作为启动项。
- OwinStartup Attribute：当创建 Owin Startup 类时，自动会加上 Attribute 如：[\*\*\*\*assembly\*\*: OwinStartup(typeof(JKXY.KatanaDemo.OwinHost.Startup))]\*\*

- 配置文件，如：
- 如果使用自定义 Host，那么可以通过 `WebApp.Start("http://localhost:10002")` 来设置启动项。
- 如果使用 OwinHost，那么可以通过命令行参数来实现，如下截图所示



```
C:\Users\do_000\Documents\Visual Studio 2013\Projects\JKXY.KatanaDemo.Web\JKXY.KatanaDemo.OwinHost>..\packages\owinhost.3.0.1\tools\owinhost.exe JKXY.KatanaDemo.OwinHost.StartupDefault
Starting with the default port: 5000 (http://localhost:5000/)
Started successfully
Press Enter to exit
```

## 启动项 Startup 的高级应用

启动项 Startup 支持 Friendly Name，通过 Friendly Name 我们可以动态切换 Startup。

比如在部署时，我们会有 UAT 环境、Production 环境，在不同的环境中我们可以动态切换 Startup 来执行不同的操作。

举个例子，我创建来两个带有 Friendly Name 的 Startup，如下所示：

```
[assembly: OwinStartup("Production", typeof(JKXY.KatanaDemo.Web.StartupProduction))]
namespace JKXY.KatanaDemo.Web
{
    using AppFunc = Func<IDictionary<string, object>?, Task>;
    public class StartupProduction
    {
        public void Configuration(IAppBuilder app)
        {
            app.Run(context => context.Response.WriteAsync("Production"));
        }
    }
}

[assembly: OwinStartup("UAT", typeof(JKXY.KatanaDemo.Web.StartupUAT))]

namespace JKXY.KatanaDemo.Web
{
    public class StartupUAT
    {
        public void Configuration(IAppBuilder app)
        {
            app.Run(context => context.Response.WriteAsync("UAT"));
        }
    }
}
```

\*\*\*\*根据 Friendly Name 使用配置文件或者 OwinHost 参数来切换 Startup\*\*\*\*

```
<appsettings>
  <add key="owin:appStartup" value="Production">
</add></appsettings>
```

## 小结

这篇博客为大家讲解了 Katana 的世界，那么接下来我将继续 OWIN & Katana 之旅，探索 Middleware 的创建，谢谢大家支持。





9

## ASP.NET MVC 随想录（8）——创建自定义的 Middleware 中间件



经过前 2 篇文章的介绍，相信大家已经对 OWIN 和 Katana 有了基本的了解，那么这篇文章我将继续OWIN 和 Katana 之旅——创建自定义的 Middleware 中间件。

## 何为 Middleware 中间件

---

Middleware 中间件从功能上可以理解为用来处理 Http 请求，当 Server 将 Http 请求封装成符合 OWIN 规范的字典后，交由 Middleware 去处理，一般情况下，Pipeline 中的 Middleware 以链式的形式处理 Http 请求，即每一个 Middleware 都是最小的模块化，彼此独立、高效。

从语法上理解 Middleware 的话，他是一个应用程序委托（`Func<IDictionary<string,object>, Task>`）的实例，通过使用 `IAppBuilder` 接口的 `Use` 或者 `Run` 方法将一个 Middleware 插入到 Pipeline 中，不同的是使用 `Run` 方法不需要引用下一个 Middleware，即他是 Pipeline 中最后的处理元素。

## 使用 Inline 方式注册 Middleware

---

使用 Use 方法可以将一个 Middleware 插入到 Pipeline 中，值得注意的是需要传入下一个 Middleware 的引用，代码如下所示：

```
app.Use(new Func<Func<IDictionary<string, object>, Task> /*Next*/,  
Func<IDictionary<string, object> /*Environment Dictionary*/, Task>>(next => async env =>  
{  
    string before = "Middleware1--Before(inline)" + Environment.NewLine;  
    string after = "Middleware1--After(inline)" + Environment.NewLine;  
    var response = env["owin.ResponseBody"] as Stream;  
    await response.WriteAsync(Encoding.UTF8.GetBytes(before), 0, before.Length);  
    await next.Invoke(env);  
    await response.WriteAsync(Encoding.UTF8.GetBytes(after), 0, after.Length);  
}));
```

上述代码中，实例化了一个委托，它需要传入下一个 Pipeline 中的 Middleware 引用同时返回一个新的 Middleware 并插入到 Pipeline 中。因为是异步的，所以别忘了 async、await 关键字。

## 使用 Inline+ AppFunc 方式注册 Middleware

---

为了简化书写, 我为应用程序委托 ( `Func<IDictionary<string,object>, Task>` ) 类型创建了别名 `AppFunc`:

```
using AppFunc=Func<IDictionary<string,object>/*Environment Dictionary*/,Task/*Task*/>;
```

所以又可以使用如下方式来讲 Middleware 添加到 Pipeline 中:

```
app.Use(new Func<AppFunc, AppFunc>(next => async env =>
{
    string before = "\tMiddleware2--Before(inline+AppFunc)" + Environment.NewLine;

    string after = "\tMiddleware2--After(inline+AppFunc)" + Environment.NewLine;

    var response = env["owin.ResponseBody"] as Stream;

    await response.WriteAsync(Encoding.UTF8.GetBytes(before), 0, before.Length);

    await next.Invoke(env);

    await response.WriteAsync(Encoding.UTF8.GetBytes(after), 0, after.Length);

}));
```

考虑到业务逻辑的增长, 有必要将 Lambda 表达式中的处理逻辑给分离开来, 所以对上述代码稍作修改, 提取到一个名为 `Invoke` 的方法内:

```
app.Use(new Func<AppFunc, AppFunc>(next => env => Invoke(next, env)));
private async Task Invoke(Func<IDictionary<string, object>, Task> next, IDictionary<string, object> env)
{
    var response = env["owin.ResponseBody"] as Stream;

    string pre = "\t\tMiddleware 3 - Before (inline+AppFunc+Invoke)" + Environment.NewLine;

    string post = "\t\tMiddleware 3 - After (inline+AppFunc+Invoke)" + Environment.NewLine;

    await response.WriteAsync(Encoding.UTF8.GetBytes(pre), 0, pre.Length);

    await next.Invoke(env);

    await response.WriteAsync(Encoding.UTF8.GetBytes(post), 0, post.Length);

}
```

虽然将业务逻辑抽取到一个方法中，但 Inline 这种模式对于复杂的 Middleware 还是显得不够简洁、易懂。我们更倾向于创建一个单独的类来表示。

## 定义原生 Middleware 类的形式来注册 Middleware

如果你只想简单的跟踪一下请求，使用 Inline 也是可行的，但对于复杂的 Middleware，我倾向于创建一个单独的类，如下所示：

```
public class RawMiddleware
{
    private readonly AppFunc _next;
    public RawMiddleware(AppFunc next)
    {
        this._next = next;
    }

    public async Task Invoke(IDictionary<string,object> env )
    {
        var response = env["owin.ResponseBody"] as Stream;

        string pre = "\t\t\tMiddleware 4 - Before (RawMiddleware)" + Environment.NewLine;

        string post = "\t\t\tMiddleware 4 - After (RawMiddleware)\r\n" + Environment.NewLine;

        await response.WriteAsync(Encoding.UTF8.GetBytes(pre), 0, pre.Length);

        await _next.Invoke(env);

        await response.WriteAsync(Encoding.UTF8.GetBytes(post), 0, post.Length);

    }
}
```

最后，依旧是通过 Use 方法来将 Middleware 添加到 Pipeline 中：

```
//两者方式皆可
//app.Use<rawmiddleware>();
app.Use(typeof (RawMiddleware));
```

上述代码中，IApplicationBuilder 实例的 Use 方法添加 Middleware 至 Pipeline 与 Inline 方式有很大不同，它接受一个 Type 而非 Lambda 表达式。在这种情形下，创建了一个 Middleware 类型的实例，并将 Pipeline 中下一个 Middleware 传递到构造函数中，最后当 Middleware 被执行时调用 Invoke 方法。

注意 Middleware 是基于约定的形式定义的，需要满足如下条件：

\* 构造函数的第一个参数必须是 Pipeline 中下一个 Middleware \* 必须包含一个 Invoke 方法，它接收 Owin 环境字典，并返回 Task

## 使用Katana Helper来注册Middleware

---

程序集 Microsoft.Owin 包含了 Katana 为我们提供的 Helper，通过他，可以简化我们的开发，比如 IOwinContext 封装了 Owin 的环境字典，强类型对象可以通过属性的形式获取相关数据，同时为 IApplicationBuilder 提供了丰富的扩展方法来简化 Middleware 的注册，如下所示：

```
app.Use(async (context, next) =>
{
    await context.Response.WriteAsync("\t\t\t\tMiddleware 5--Before(inline+katana helper)" + Environment.NewLine);

    await next();

    await context.Response.WriteAsync("\t\t\t\tMiddleware 5--After(inline+katana helper)" + Environment.NewLine);

});
```

当然我们也可以定义一个 Middleware 类并继承 OwinMiddleware，如下所示：

```
public class MyMiddleware : OwinMiddleware
{
    public MyMiddleware(OwinMiddleware next)
    : base(next)
    {

    }

    public override async Task Invoke(IOwinContext context)
    {
        await context.Response.WriteAsync("\t\t\t\tMiddleware 6 – Before (Katana helped middleware class)" + Environment.NewLine);

        await this.Next.Invoke(context);

        await context.Response.WriteAsync("\t\t\t\tMiddleware 6 – After (Katana helped middleware class)" + Environment.NewLine);

    }
}
```

然后将其添加到 Pipeline 中：

```
app.Use<mymiddleware>();
```

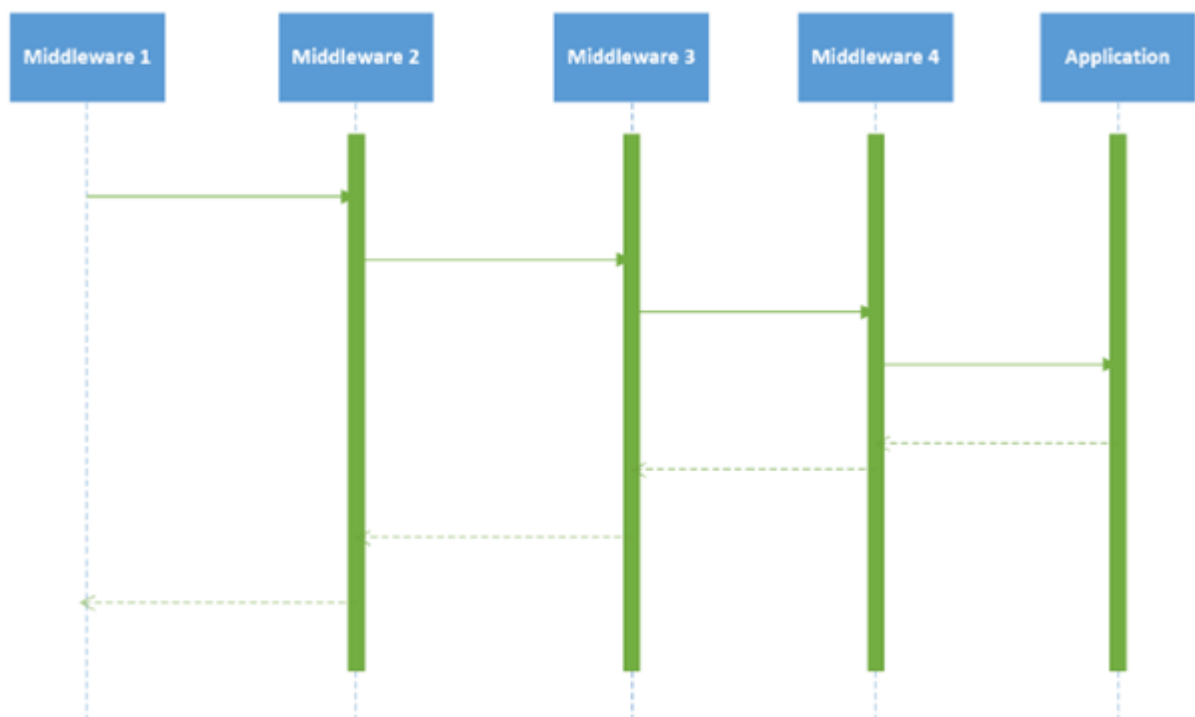


## Middleware 的执行顺序

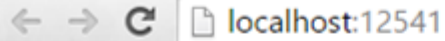
在完成上面 Middleware 注册之后，在 Configuration 方法的最后添加最后一个的 Middleware 中间件，注意它并不需要对下一个 Middleware 的引用了，我们可以使用 Run 方法来完成注册：

```
app.Run(context=>context.Response.WriteAsync("\t\t\t\t\tHello World"+Environment.NewLine));
```

值得注意的是，Pipeline 中 Middleware 处理 Http Request 顺序同注册顺序保持一致，即和 Configuration 方法中书写的顺序保持一致，Response 顺序则正好相反，如下图所示：



最后，运行程序，查看具体的输出结果是否和我们分析的保持一致：

A web browser address bar with navigation icons (back, forward, refresh) and a document icon, followed by the text "localhost:12541".

```
Middleware 1—Before(inline)
  Middleware 2—Before(inline+AppFunc)
    Middleware 3 - Before (inline+AppFunc+Invoke)
      Middleware 4 - Before (RawMiddleware)
        Middleware 5—Before(inline+katana helper)
          Middleware 6 - Before (Katana helped middleware class)
            Hello World
          Middleware 6 - After (Katana helped middleware class)
        Middleware 5—After(inline+katana helper)
      Middleware 4 - After (RawMiddleware)
    Middleware 3 - After (inline+AppFunc+Invoke)
  Middleware 2—After(inline+AppFunc)
Middleware 1—After(inline)
```

## 小结

---

在这篇文章中，我为大家讲解了有关自定义 Middleware 的创建的相关知识，Katana 为我们提供了非常多的方式来创建和注册 Middleware，在下一篇文章中，我将继续为大家讲解 OWIN 和 Katana 的其他知识，和大家一起探索 Katana 和其他 Web Framework 的集成。

# 极客学院

jikexueyuan.com

中国最大的IT职业在线教育平台



更多信息请访问 

<http://wiki.jikexueyuan.com/project/think-in-asp-net-mvc/>