第3章　ASP.NET技术架构

ASP.NET 4.0 应该说和以前的老版本比起来已经有很大的变化了。这不光是增加了新功能或者新的某些技术，而是其核心已经发生了变化。

本章将介绍有关 ASP.NET 4.0 技术体系下的代码模型、编译模型、可扩展性和缓存等。

3.1　代码模型

代码模型是了解 ASP.NET 4.0 工作原理的重要部分。作为代码模型的发展，ASP.NET 1.x 代码模型为新的改进提供了足够的实践调查和技术储备。在传统的 ASP.NET 1.x 代码模型中，开发人员可以选择直接在 ASPX 文件中编写代码，开发简单的小应用。同时也可以选择代码隐藏模型，将业务逻辑和事件处理代码写入一个只有代码的文件中，允许设计人员处理呈现文件而开发人员处理代码文件，加快实际项目的开发周期。

旧版本 ASP.NET 代码模型可以按照需求分为两部分：代码嵌入和代码隐藏模型。

代码嵌入方法是在 ASPX 页面中结合 HTML 和一种.NET 语言混合编程，界面页继承 Page 基类并在运行时被编译到零时文件 Temp.dll，如图 3-1 所示。

代码隐藏模型则支持开发人员把页面和代码分开编写，在部署和生成时进行隐藏代码的预编译使其成为引用型组件，在运行时编译页面到零时文件 Temp.dll，如图 3-2 所示。

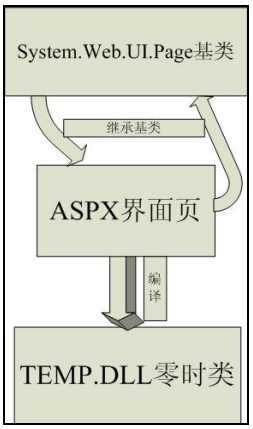

图 3-1　代码嵌入模型

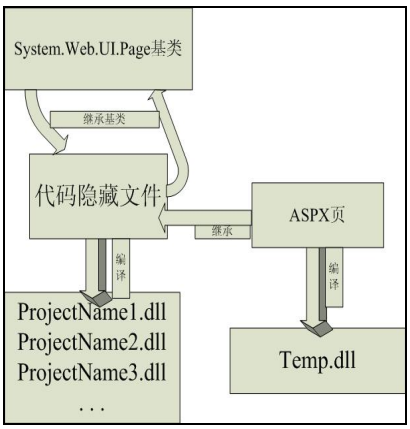

图 3-2　代码隐藏模型

（1）复杂的继承。在旧版本 ASP.NET 中不管是窗体页还是代码也都是一一继承的关系，但两个文件在实际使用中却通过一种更为复杂的关系连接在一起。

在 ASP.NET 1.x/2.x 中，开发人员可以使用 VS.NET 2003 或者 VS.NET 2005 将控件拖

放到 ASPX 页面，然后 IDE 自动在代码隐藏文件中生成声明代码。控件被添加到 ASPX 页时，为了使用和开发更为方便和规范，使用时必须把相关代码添加到代码隐藏文件。

（2）复杂的编译。在 ASP.NET 2.x 中，如果使用代码隐藏模型开发项目，那么所有的代码隐藏文件都编译到一个程序集并存储在 Web 应用程序的 bin 目录中。

系统在第一次运行时将全部代码编译并加载 bin 目录中的程序集。ASP.NET 运行库实际上将 ASPX 页编译到它自己的临时程序集中。

往往会出现很多部署后的问题，比如部署之后开发人员修改 ASPX 页的一个属性或者更改控件的类型，但并没有更新代码隐藏文件，也没有重新编译它们。这样操作后，由于代码和页面不匹配系统会出现各种错误。

新的 ASP.NET 4.0 代码模型。为了解决在上一个版本中出现的问题，新的版本做了根本性的改变。虽然 ASP.NET 4.0 继续提供代码内联和代码隐藏编码模型，但内联模型基本和 1.x 差不多，除了 VS2010 IDE 中的智能开发。

ASP.NET 4.0 同样也分为两个 2 个代码模型，代码嵌入式没有变化，对于代码隐藏模型，解决了代码隐藏模型的继承和编译问题。在 ASP.NET 4.0 中，代码隐藏文件不再是 System.Web.UI.Page 类的完整继承。整个代码隐藏文件属于局部类。局部类包含所有用户编写的代码，去掉了在旧版本 ASP.NET 中自动生成的基础结构和代码。

ASP.NET 4.0 的页面在被请求时，如果是包括代码隐藏文件的页面，ASP.NET 4.0 会将 ASPX 页和局部类合并为一个类，这样由于结构不统一，出现的错误基本就杜绝了。ASP.NET 4.0 的代码隐藏编码模型如图 3-3 所示。

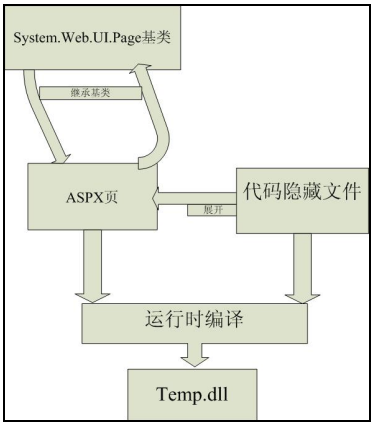

图 3-3　ASP.NET 4.0 代码隐藏编码模型

3.2　代码的结构

使用 VS2010 开发 ASP.NET 4.0 应用系统，具有的事件语法通过 Visual Studio 生成。获得的代码隐藏文件更为简短并且不受自动生成代码的影响。

以前版本出现的事件丢失现象能够避免了，这主要是由于 ASP.NET 运行库自动将代码隐藏中的事件连接到 ASPX 中的控件，不必通过 VS 的 IDE 显示。这种新代码隐藏模型大大降低了继承的复杂性。

代码隐藏文件将不会出现复杂的控件声明代码，因为 ASPX 页不直接继承代码隐藏文件，代码隐藏文件可以不需要定义和支持 ASPX 页的所有控件。作为开发人员，通过代码隐藏文件可以访问 ASPX 页上的任何控件，而不需要额外地声明代码。

ASP.NET 运行库编译块将该模式产生的声明和事件连接代码插入最终的已编译文件

中，保证代码开发人员和 Web 设计员工作同步。

比如新创建一个窗体页后简洁的代码如下。

```
namespace ASPNET40
{
    public partial class Webform1 : System.Web.UI.Page
    {
    void Page_Load(obje Page_Load(object ct rgs A sender, Event EventArgs
  e)
    {
    Label1.Text = " ASP.NET 4.0 简洁代码";
    }
    }
}
```

3.3 编译模型

作为编译模型，ASP.NET 4.0 也有了很大的改进。以往的 1.x 版本需要将开放人员编写的隐藏代码文件编译到一个程序集中并存储在 bin 目录中。该模式的编译过程将导致第一次请求 ASP.NET 页面时的响应速度比后续请求慢。同时 2.x 版本的窗体页 ASPX 必须要以明文的 HTML 形式部署到 Web 站点或者服务器。当开发人员使用代码内嵌模型时，将造成一些业务逻辑和代码也部署在生产服务器。这样的做法将有可能导致人为系统错误或者安全问题。2.x 版本的编译模型如图 3-4 所示。

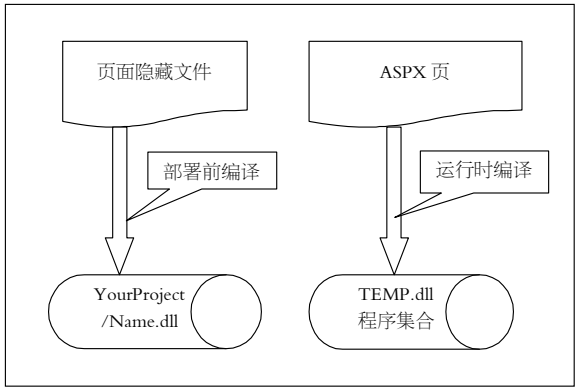

图 3-4　2.x 版本编译模型

ASP.NET 4.0 的编译模型有以下 4 种类型。

（1）普通编译。作为 ASP.NET 4.0 技术在编译程序时可以选择普通编译模式，该模式将代码隐藏文件编译到一个程序集并存储到 bin 目录中。

该模型对大多数 Web 站点运行并没有问题，效果不错。但需要注意的是，用该模式编译的系统，第一次请求页面速度比较缓慢。

（2）部署预编译。ASP.NET 4.0 允许在部署前对项目进行完整编译。在完整编译中，所有的代码隐藏文件、ASPX 页面、HTML、图形资源以及其他的后端代码都被编译到一个或多个可执行程序集中。

通过在部署之前预编译 Web 站点，安全性将大大提高，其他人员如果不对程序集进行反编译，将无法访问任何代码。为了增强保护，开发人员还可以进一步使用工具打乱生成的程序集，使部署的 Web 应用程序更安全。

部署预编译的主要缺点在于其严重的不可修改性，如果要进行更改，必须重新编译 Web 站点并重新部署。

（3）完整运行时编译。还有一种编译方式非常开放，ASP.NET 4.0 提供在运行时编译整个应用程序的新机制。

该方式主要是将未编译的代码隐藏文件和其他相关的代码放在代码目录\app_code 中，并通过 ASP.NET 4.0 创建并维护对程序集的引用。

这种选择以在服务器上存储未编译代码为代价，为更改已部署 Web 系统提供了最大的灵活性。

微软建议此模式适用于不需要大量支持代码的 ASP.NET 应用程序。对于简单的应用程序而言，快速部署和测试系统的能力要比高可靠性的编译方法更好。

（4）批编译。ASP.NET 4.0 中仍然可以选择进行批编译。批编译消除了第一次页面请求的延时，但造成了更长的启动周期。

批编译的优点在于，页面可以立即显示给第一个用户，并且可以在批编译过程中检测到 ASPX 页面中的任何错误。该编译模式必须内置在 web.config 文件中。

需要注意的是，在批编译过程中如果某个文件出现了问题，则该次批编译将跳过该文件。

批编译需要配置的 Web.config 的方法如下。

```
<compilation
batch=" batch="true|false"
batchTimeout="超时数"
maxBatchSize ="最大批编译数"
maxBatchGeneratedFileSize ="最大批编译文件大小"
</compilation>
```

3.4　扩展性与管道技术

ASP.NET 4.0 可以认为是一个可扩展的架构。ASP.NET 4.0 的许多模块和组件都可以扩展、修改或替换满足项目的需要。在 ASP.NET 4.0 中，重要的扩展手段就是使用诸如 HTTPHandlers 和 HTTPModules 等处理程序。

（1）请求管道。当一个请求表示需要访问某个站点的时候，IIS 接收到请求并将根据 IIS 设置将扩展名映射到 ISAPI 筛选器。比如，ASP.NET 4.0 的页面.aspx、.asmx、.asd 及

其他扩展名被映射到专用 ISAPI 筛选器 aspnet_isapi.dll。筛选器将启动 ASP.NET（CRL）运行库。

当请求遇到 ASP.NET 运行库，将在 HTTPApplication 对象上启动。HTTPApplication 对象将请求传递给一个或多个 HTTPModule 实例进行会话维护、验证或配置文件维护。如果是动词和路径则将请求传递给 HTTPHandler 处理。

处理模式和流程如图 3-5 所示。

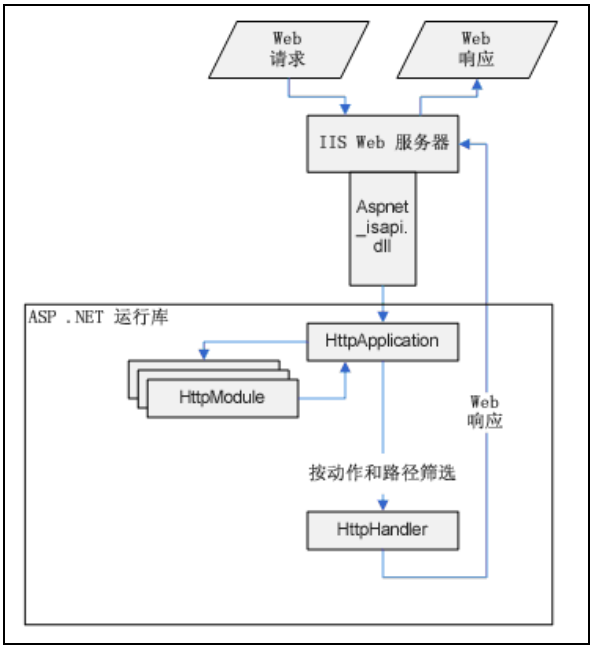

图 3-5　请求管道

（2）处理程序。ASP.NET 4.0 支持应用程序配置工具和其他新功能的处理程序。这些处理程序将允许开发人员配置 ASP.NET 用户和其他设置的管理工具。这些处理程序和功能如表 3-1 所示。

表 3-1　主要的处理程序

管理处理程序	描　　述
WebAdminHandler	管理 Web 站点的主页，该处理程序可以管理 ASP.NET 4.0 Web 应用程序
TraceHandler	跟踪处理信息程序
WebResourcesHandler	WebResourcesHandler 可以将 Web 资源配置为后部署
CachedImageServiceHandler	支持缓存图形组件信息处理
PrecompHandler	使用 PrecompHandler 可以对 ASP.NET 应用程序中的所有 ASPX 页面进行批编译
WebPartExportHandler	WebPartExportHandler 支持存储和传输 Web 部件布局信息
HTTPForbiddenHandler	禁止指定类型的文件被访问，比如母版页、外观文件及其他代码文件

3.5　缓存技术

当网页频繁被访问而且并发量很大时，有可能造成系统瘫痪。缓存技术在这种情况下显得非常重要。

在 ASP.NET4.0 中，缓存技术主要分三类，分别是页面级别、页面片段和编程缓存。很多人对于哪种情况该使用哪种缓存技术搞不清楚。一些人错误地认为编程缓存最好，可以缓存对象，很灵活，但有没有考虑资源消耗问题。频繁而不重复的操作可能导致大量缓存出现，拖垮服务器。比如带分页和排序的用户选择编程对象缓存方式就是非常错误的。

作为一般应用，最普遍的做法是将整个网页缓存以提高执行效率。这样做的优点是，当用户再次访问这个网页的时候，缓存页会被直接显示。页面级别缓存的一个最大问题是缓存的页面不可控，导致无用信息被存储，消耗系统资源。

在 ASP.NET 4.0 中，页面级别的缓存机制已得到扩展，开发人员现在可以使用缓存后替换功能，用刷新的内容替换缓存的部分内容。缓存后替换功能允许应用程序使用页面级别的缓存。

另外借助数据库缓存依赖关系，可以将缓存的页面绑定到 SQL Server 数据库中的特定表。如果表发生变化，缓存将自动过期并获取最新信息。

下面介绍页面组缓存的一些使用方法。

（1）OutputCache 指令。

指令 OutputCache 的声明能够实现页面输出缓存。OutputCache 指令声明需要在 ASP.NET 4.0 窗体页头部或者用户控件的头部完成，并且通过不同属性的设置，能够实现页面的缓存输出策略。

指令 OutputCache 可以使用的属性包括 10 个，即 CacheProfile、NoStore、Duration、Shared、Location、SqlDependency、VaryByControl、VaryByCustom、VaryByHeader 和 VaryByParam。这些属性将对缓存时间、缓存项的位置、SQL 数据缓存依赖等方面进行设置。

（2）使用 API 设置缓存。

使用 API 设置缓存页面的方法可谓一个新的选择，其效果和直接在页面声明 OutputCache 指令的效果一样。但是这种方法还可以进行更加详细的信息设置。

该方法的核心是调用 System.Web.HttpCachePolicy，该类主要用于设置缓存特定的 HTTP 标头和页面输出缓存的方式。

比如当需要设置该页面缓存的到期时间以及页面不随任何 GET 或 POST 参数改变的特性，需要编写如下的代码：

```
Response.Cache.SetExpires(Now.AddSeconds(60))
Response.Cache.SetCacheability(HttpCac heability.Public)
```

（3）数据库缓存依赖关系。

ASP.NET 4.0 支持数据库缓存依赖关系，当使用 SQL Server 2000 时可以使用表级别的

消息通知。假设该缓存页与某个表建立了依赖关系，当该表中数据发生变化，ASP.NET 4.0 将移除该缓存并根据变化建立新缓存，以保持信息准确显示。

数据库缓存依赖关系需要使用 SQL 数据库依赖属性 sqldependency。只要对目标表进行更改，缓存的页面就将过期。sqldependency 属性必须引用在配置文件 Web.config 的数据源 datasource 中。

比如需要对参数为“ID”的信息建立缓存依赖关系，使用的声明方式如下：

```
<%@ outputcache duration="3600"
varybyparam="ID"
sqldependency="MyDatabase:TableName" %>
```

需要强调的是，该方法只对微软的数据库系统 SQL Server 有效，如果使用其他数据库需要自定义依赖类 CacheDependency。

（4）管理 Output Cache。

如果项目启用了页面级缓存 Output Cache，默认情况下所有缓存数据会被存放到硬盘上。可以通过修改 DiskCacheable 的属性设置其是否缓存。声明方法如下：

```
<%@ OutputCache Duration="1000" VaryByParam="no" DiskCacheable="true" %>
```

在 web config 里可以配置缓存文件的大小（单位是兆），具体方法如下：

```
<caching>
  <outputCache>
   <diskCache enabled="true" maxSizePerApp="1000" />
  </outputCache>
</caching>
```

（5）缓存后替换功能。

在很多情况下使用页面级缓存可能需要实时的、更小局部的信息，比如时间、访问用户信息。按照以往情况，页面级缓存不易实现。ASP.NET 4.0 支持缓存后替换，也就是通知 ASP.NET 运行库是否应在向用户显示缓存页面之前重新判断该页面上的某个元素。

在使用缓存后替换功能时需要注意控件 Substitution 的方法属性 MethodName 必须指定。在对需要动态显示的信息进行回掉过程中需要保持其上下文，否则将提示无法找到该方法的错误信息。范例代码如下：

```
public partial class _Default : System.Web.UI.Page
{
   protected void Page_Load(object sender, EventArgs e)
   {
      Label1.Text = System.DateTime.Now.ToShortTimeString();

   }
   public static string GetTime(HttpContext context)
```

```
    {
        return System.DateTime.Now.ToShortTimeString();
    }
}
```

3.6　难点解析

相信读者看到这里已经对 ASP.NET 有一个初步的认识了。也许还是感觉脑子有点乱，不太明白到底 ASP.NET 技术包括哪些内容，ASP.NET 到底代表什么意思。

在正式学习具体开发技术之前，有必要简要概括地再次透彻解释一下 ASP.NET 技术的含义，让读者搞清楚它的本质。

其实 ASP.NET 是完全使用托管代码处理 Web 请求的一个成熟引擎平台。它不仅仅只是 WebForms 和 WebServices。

ASP.NET 是一个请求处理引擎。它获取客户端请求，然后通过它内置的管道，把请求传到一个终点。在这个终点，开发者可以添加处理这个请求的逻辑代码。实际上这个引擎和 HTTP 或者 Web Server 是完全分开的。事实上，HTTP 运行时是一个组件，可以把它宿主在 IIS 之外的应用程序上，甚至完全可以和其他的服务组合在一起。例如，可以把 HTTP 运行时宿主在 Windows 桌面应用程序里通过使用内置的管道路由请求，HTTP 运行时提供了一套复杂的但却很优雅的机制。在处理请求的每一个层面都牵涉许多对象，但大多数对象都可以通过派生或者事件接口扩展。所以，此框架具有非常高的可扩展性。通过这套机制，可以进入较低层次的接口，如缓存、身份验证、授权等是有可能的。可以在处理请求之前或之后过滤内容，或者仅仅把匹配指定签名的客户端请求直接路由到你的代码里或转向其他的 URL。针对同一件事情，可以通过不同的处理方法完成，而且实现代码都非常的直观。除此之外，在容易开发和性能之间，HTTP 运行时还提供了最佳的灵活性。

整个 ASP.NET 引擎完全构建在托管代码里，所有的扩展性功能都是通过托管代码的扩展提供。对于功能强大的.NET 框架而言，使用自己的东西，构建一个成熟的、高性能的引擎体系结构已经成为一个发展方向。尽管如此，重要的是，ASP.NET 给人印象最深的是其高瞻远瞩的设计，这使得在其之上的工作变得非常容易，并且提供了几乎可以钩住请求处理当中任意部分的能力。

使用 ASP.NET 可以完成一些任务，之前这些任务是使用 IIS 上的 ISAPI 扩展和过滤完成的。尽管还有一些限制，但与 ASP 相比，已经有了很大的进步。ISAPI 是底层 Win32 样式的 API，仅它的接口就有 1 兆，这对于大型程序开发是非常困难的。由于 ISAPI 是底层的接口，因此它的速度非常快。但对于企业级的程序开发是相当难于管理的。所以，在一定时间内，ISAPI 主要充当其他应用程序或平台的桥接口。但是无论如何，ISAPI 没有被废弃。事实上，微软平台上的 ASP.NET 和 IIS 的接口是通过宿主在.NET 里的 ISAPI 扩展来通信的，直达 ASP.NET 运行时。ISAPI 提供了与 Web Server 通信的核心接口，然后 ASP.NET

使用非托管代码获取请求以及对客户端请求发出响应。ISAPI 提供的内容经由公共对象类似于 HttpRequest 和 HttpResponse，通过一个设计优良的、可访问的接口，以托管对象的方式暴露非托管数据。

从浏览器到 ASP.NET，让我们从一个典型的 ASP.NET Web 请求的生命周期的起点开始。用户通过在浏览器中键入一个 URL，点击一个超链接，提交一个 HTML 表单（一个 post 请求），或者一个客户端程序调用基于 ASP.NET 的 WebService（通过 ASP.NET 提供服务）。在服务器端，IIS5 或者 IIS6 将会收到这个请求。ASP.NET 的底层通过 ISAPI 扩展与 IIS 通信，然后，通过 ASP.NET，这个请求通常被路由到一个带有.aspx 扩展名的页面。但是，这个处理过程如何工作，则完全依赖于 HTTP 处理器（handler）的执行。这个处理器将被安装用于处理指定的扩展。在 IIS 中，.aspx 经由“应用程序扩展”被映射到 ASP.NET ISAPI 的 dll 文件：aspnet_isapi.dll。每一个触发 ASP.NET 的请求，都必须经由一个已经注册，并且指向 aspnet_isapi.dll 的扩展名标识，这点读者应该结合本章其他小节仔细体会。

3.7 高手训练营

1. ASP.NET 分层式结构的优势和缺陷有哪些？
2. ASP.NET 的 MVC 模式是什么？说出他的优点和不足。
3. ASP.NET 的处理逻辑都包括哪些内容？
4. 用.NET 做 B/S 结构的系统，一般用几层结构来开发，说明每一层之间的关系以及为什么要这样分层。