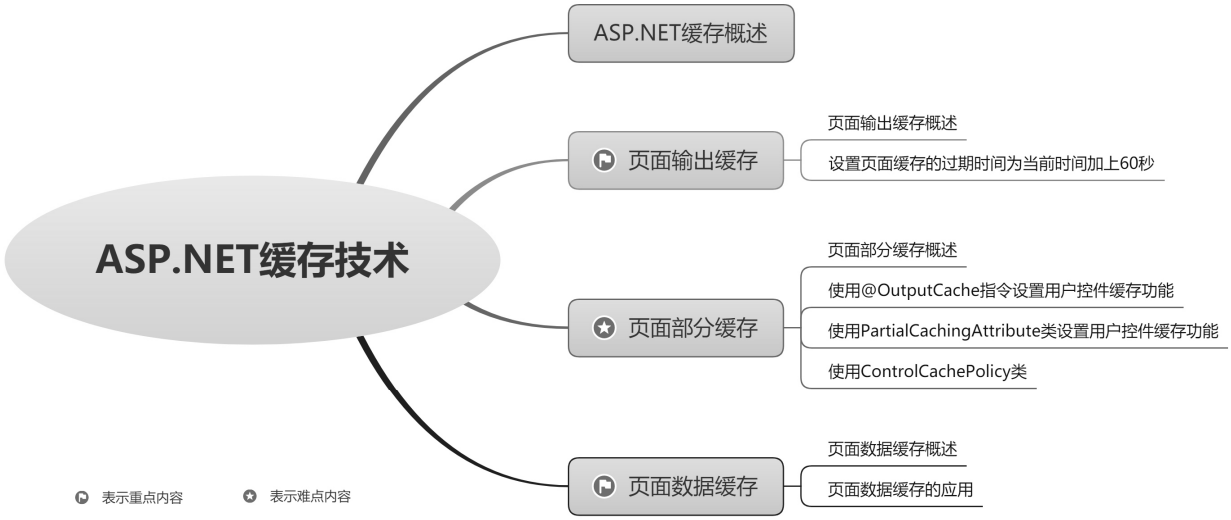
第 13 章

ASP.NET 缓存技术

缓存是系统或应用程序将频繁使用的数据保存到内存中，当系统或应用程序再次使用时，能够快速地获取数据。缓存技术是提高 Web 应用程序开发效率最常用的技术。在 ASP.NET 中，有 3 种 Web 应用程序可以使用缓存技术，即页面输出缓存、页面部分缓存和页面数据缓存，本章将分别进行介绍。

本章知识架构及重难点如下。



13.1 ASP.NET 缓存概述

缓存是 ASP.NET 中非常重要的一个特性，可以生成高性能的 Web 应用程序。生成高性能的 Web 应用程序最重要的因素之一，就是将那些频繁访问且不需要经常更新的数据存储在内存中，当客户端再一次访问这些数据时，可以避免重复获取满足先前请求的信息，以快速显示请求的 Web 页面。

ASP.NET 中有 3 种 Web 应用程序可以使用缓存技术，即页面输出缓存、页面部分缓存和页面数据缓存。ASP.NET 的缓存功能具有以下优点。

- ☑ 支持更为广泛和灵活的可开发特征。ASP.NET 2.0 及以上版本包含一些新增的缓存控件和 API，如自定义缓存依赖、Substitution 控件、页面输出缓存 API 等，这些特征能够明显改善开发人员对于缓存功能的控制。
- ☑ 增强可管理性。使用 ASP.NET 提供的配置和管理功能可以更加轻松地管理缓存。

☑ 提供更高的性能和可伸缩性。ASP.NET 提供了一些新的功能，如 SQL 数据缓存依赖等，这些功能将帮助开发人员创建高性能、伸缩性强的 Web 应用程序。

注意

缓存功能也有不足之处。例如，显示的内容可能不是最新、最准确的，为此必须设置合适的缓存策略。又如，缓存增加了系统的复杂性，并使其难于测试和调试。因此，建议在没有缓存的情况下开发和测试应用程序，然后在性能优化阶段启用缓存功能。

13.2　页面输出缓存

13.2.1　页面输出缓存概述

页面输出缓存是最为简单的缓存机制，该机制将整个 ASP.NET 页面内容保存在服务器内存中。当用户请求该页面时，系统从内存中输出相关数据，直到缓存数据过期。在这个过程中，将缓存内容直接发送给用户，而不必再次经过页面处理生命周期。通常情况下，页面输出缓存对于那些包含不需要经常修改内容，但需要大量处理才能编译完成的页面特别有用。另外，页面输出缓存是将页面全部内容都保存在内存中，并用于完成客户端请求。

页面输出缓存需要利用有效期来对缓存区中的页面进行管理。设置缓存的有效期可以使用@ OutputCache 指令。@ OutputCache 指令的格式如下：

```
<%@ OutputCache Duration="#ofseconds"
Location="Any | Client | Downstream | Server | None | ServerAndClient "
Shared="True | False"
VaryByControl="controlname"
VaryByCustom="browser | customstring"
VaryByHeader="headers"
VaryByParam="parametername"
%>
```

@ OutputCache 指令中的各个属性及其说明如表 13.1 所示。

表 13.1　@ OutputCache 指令中的各个属性及其说明

属　性	说　明
Duration	页或用户控件进行缓存的时间（以秒为单位）。该属性是必需的，在@ OutPutCache 指令中至少要包含该属性
Location	指定输出缓存可以使用的场所，默认值为 Any。用户控件中的@ OutPutCache 指令不支持此属性
Shared	确定用户控件输出是否可以由多个页共享，默认值为 false
VaryByControl	该属性使用一个用分号分隔的字符串列表来改变用户控件的部分输出缓存，这些字符串代表用户控件中声明的 ASP.NET 服务器控件的 ID 属性值。值得注意的是，除非已经包含了 VaryByParam 属性，否则在用户控件@ OutputCache 指令中必须包括该属性

续表

属　　性	说　　明
VaryByCustom	根据自定义的文本来改变缓存内容。如果赋予该属性的值为 browser，缓存将随浏览器名称和主要版本信息的不同而不同。如果值是 customstring，还必须重写 Global.asax 中的 GetVaryByCustomString 方法
VaryByHeader	根据 HTTP 头信息来改变缓存区内容，当有多重头信息时，输出缓存中会为每个指定的 HTTP 头信息保存不同的页面文档，该属性可以应用于缓存所有 HTTP 1.1 的缓存内容，而不仅限于 ASP.NET 缓存。页面部分缓存不支持此属性
VaryByParam	该属性使用一个用分号分隔的字符串列表使输出缓存发生变化。默认情况下，这些字符串与用 GET 或 POST 方法发送的查询字符串值对应。当将该属性设置为多个参数时，对于每个指定参数组合，输出缓存都包含一个不同版本的请求文档，可能的值包括 none、星号（*），以及任何有效的查询字符串或 POST 参数名称

13.2.2　设置页面缓存的过期时间为当前时间加上 60 秒

【例 13.1】设置页面缓存的过期时间为当前时间加上 60 秒。**（示例位置：mr\TM\13\01）**

本示例主要通过设置缓存的有效期指令@ OutputCache 中的 Duration 属性值，实现程序运行 60 秒内刷新页面，页面中的数据不发生变化，在 60 秒后刷新页面，页面中的数据发生变化，如图 13.1 和图 13.2 所示。

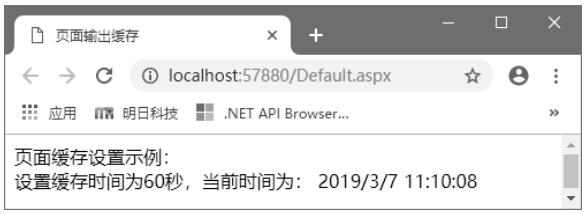


图 13.1　60 秒以内的页面缓存

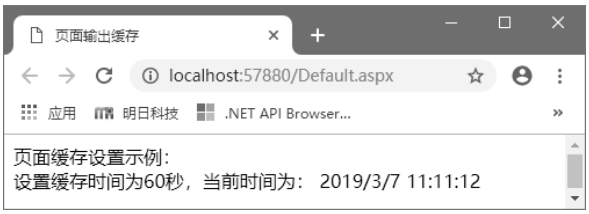


图 13.2　60 秒后的页面缓存

程序实现的主要步骤如下。

（1）新建一个网站，默认主页为 Default.aspx。

（2）将 Default.aspx 页面切换到 HTML 视图中，在<%@ Page>指令的下方添加如下代码，实现页面缓存的过期时间为当前时间加上 60 秒。

```
<%@ OutputCache Duration ="60" VaryByParam ="none"%>
```

（3）在节点<head>和<body>之间添加如下代码，输出当前系统时间，用于比较程序在 60 秒内和 60 秒后的运行状态。

```
<script language="C#" runat="server">
    void Page_Load(object sender, EventArgs e)
    {
        Response.Write("页面缓存设置示例：  <br> 设置缓存时间为 60 秒，当前时间为：  " +
DateTime.Now.ToString());
    }
    </script>
```

技巧

1. 通过 Response.Cache 以编程的方式设置网页输出缓存时间

设置网页输出缓存的持续时间可以采用编程的方式通过 Response.Cache 方法来实现。其使用方法如下：

```
Response.Cache.SetExpires(DateTime.Now.AddMinutes(10));        //10 秒后移除
Response.Cache.SetExpires(DateTime.Parse("3:00:00PM"));        //有效至下午 3 点
Response.Cache.SetMaxAge(new TimeSpan(0,0,10,0));              //有效 10 分钟
```

2. 设置网页缓存的位置

网页缓存位置可以根据缓存的内容来决定，如对于包含用户个人资料信息、安全性要求比较高的网页，最好缓存在 Web 服务器上，以保证数据无安全性问题；而对于普通的网页，最好允许它缓存在任何具备缓存功能的装置上，以充分使用资源来提高网页效率。

如果使用@ OupPutCache 指令进行网页输出缓存位置的设置，可以使用 Location 属性。例如，设置网页只能缓存在服务器的代码如下：

```
<%@ OutputCache Duration ="60" VaryByParam ="none" Location ="Server"%>
```

使用 HttpCachePolicy 类以程序控制方式进行网页输出缓存设置，代码如下：

```
Response.Cache.SetCacheability(HttpCacheability.Private);      //指定响应能存放在客户端，而不能由共
                                                               //享缓存（代理服务器）进行缓存
```

13.3　页面部分缓存

13.3.1　页面部分缓存概述

通常情况下，缓存整个页是不合理的，因为页的某些部分可能在每一次请求时都进行更改，在这种情况下，只能缓存页的一部分，即页面部分缓存。页面部分缓存是将页面部分内容保存在内存中以便响应用户请求，而页面其他部分内容则为动态内容。

页面部分缓存的实现包括控件缓存和缓存后替换两种方式。前者也可称为片段缓存，这种方式允许将需要缓存的信息包含在一个用户控件内，然后将该用户控件标记为可缓存的，以此来缓存页面输出的部分内容。例如，要开发一个股票交易的网页，每支股票价格是实时变动的，因此，整个页面必须是动态生成且不能缓存的，但其中有一小块用于放置过去一周的趋势图或成交量，它存储的是历史数据，这些数据早已是固定的事实，或者需要很长一段时间后才重新统计变动，将这部分缓存下来有很高的效益，可以不必为相同的内容做重复计算从而节省时间，这时就可以使用控件缓存。缓存后替换与用户控件缓存正好相反，用这种方式缓存整个页，页中的各段可以是动态的。

设置控件缓存的实质是对用户控件进行缓存配置，主要包括以下 3 种方法。

- ☑ 使用@ OutputCache 指令以声明方式为用户控件设置缓存功能。
- ☑ 在代码隐藏文件中使用 PartialCachingAttribute 类设置用户控件缓存。
- ☑ 使用 ControlCachePolicy 类以编程方式指定用户控件缓存设置。

说明

页面部分缓存可以分为控件缓存和缓存后替换。控件缓存是本章介绍的重点。缓存后替换是通过 AdRotator 控件或 Substitution 控件实现的，它是指控件区域内的数据不缓存，而此区域外的数据缓存。控件缓存和缓存后替换示意图如图 13.3 所示。

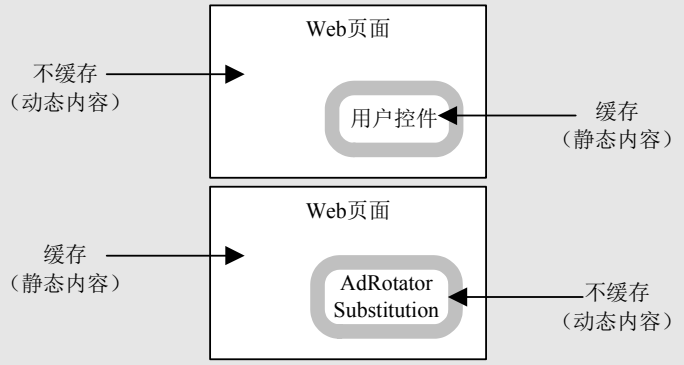


图 13.3　控件缓存和缓存后替换

13.3.2　使用@ OutputCache 指令设置用户控件缓存功能

@ OutputCache 指令以声明方式为用户控件设置缓存功能，用户控件缓存与页面输出缓存的@ OutputCache 指令设置方法基本相同，都在文件顶部设置@ OutputCache 指令，不同点包括如下两方面。

- ☑ 用户控件缓存的@ OutputCache 指令设置在用户控件文件中，而页面输出缓存的@ OutputCache 指令设置在普通 ASP.NET 文件中。
- ☑ 用户控件缓存的@ OutputCache 指令只能设置 6 个属性，即 Duration、Shared、SqlDependency、VaryByControl、VaryByCustom 和 VaryByParam，而在页面输出缓存的@ OutputCache 指令字符串中设置的属性多达 10 个。

用户控件中的@ OutputCache 指令设置源代码如下：

```
<%@ OutputCache Duration="60" VaryByParam="none" VaryByControl="ControlID" %>
```

以上代码为用户控件中的服务器控件设置缓存，其中缓存时间为 60 秒，ControlID 是服务器控件 ID 属性值。

误区警示

ASP.NET 页面和其中包含的用户控件都通过@ OutputCache 指令设置了缓存，应注意以下 3 点。

（1）ASP.NET 允许在页面和页面的用户控件中同时使用@ OutputCache 指令设置缓存，并且允许设置不同的缓存过期时间值。

（2）如果页面输出缓存过期时间长于用户控件输出缓存过期时间，则页面的输出缓存持续时间优先。例如，如果设置页面输出缓存为 100 秒，而设置用户控件的输出缓存为 50 秒，则包括用户控件在内的整个页将在输出缓存中存储 100 秒，而与用户控件较短的时间设置无关。

（3）如果页面输出缓存过期时间比用户控件的输出缓存过期时间短，则即使已为某个请求重新生成该页面的其余部分，也将一直缓存用户控件，直到其过期时间到期为止。例如，如果设置页面输出缓存为 50 秒，而设置用户控件输出缓存为 100 秒，则页面其余部分每到期两次，用户控件才到期一次。

13.3.3　使用 PartialCachingAttribute 类设置用户控件缓存功能

使用 PartialCachingAttribute 类可以在用户控件（.ascx 文件）中设置有关控件缓存的配置内容。PartialCachingAttribute 类包含 6 个常用属性和 4 种类构造函数，其中 6 个常用属性是 Duration、Shared、SqlDependency、VaryByControl、VaryByCustom 和 VaryByParam，与 13.3.2 节中的@ OutputCache 指令设置的 6 个属性完全相同，只是使用的方式不同，此处将不再对 6 个属性进行介绍。下面重点介绍 PartialCachingAttribute 类中的构造函数。PartialCachingAttribute 类中的 4 种构造函数及其说明如表 13.2 所示。

表 13.2　PartialCachingAttribute 类的构造函数及其说明

构 造 函 数	说　　明
PartialCachingAttribute(Int32)	使用分配给要缓存的用户控件的指定持续时间初始化 PartialCachingAttribute 类的新实例
PartialCachingAttribute(Int32, String, String, String)	初始化 PartialCachingAttribute 类的新实例，指定缓存持续时间、所有 GET 和 POST 值、控件名和用于改变缓存的自定义输出缓存要求
PartialCachingAttribute(Int32, String, String, String, Boolean)	初始化 PartialCachingAttribute 类的新实例，指定缓存持续时间、所有 GET 和 POST 值、控件名、用于改变缓存的自定义输出缓存要求，以及用户控件输出是否可在多页间共享
PartialCachingAttribute(Int32, String, String, String, String, Boolean)	初始化 PartialCachingAttribute 类的新实例，指定缓存持续时间、所有 GET 和 POST 值、控件名、用于改变缓存的自定义输出缓存要求、数据库依赖项，以及用户控件输出是否可在多页间共享

以上介绍了 PartialCachingAttribute 类的 6 个属性和 4 种构造函数，下面通过一个典型示例说明该类的具体应用方法。

【例 13.2】使用 PartialCachingAttribute 类实现设置用户控件缓存。**（示例位置：mr\TM\13\02）**

本示例主要通过使用 PartialCachingAttribute 类设置用户控件（WebUserControl.ascx 文件）的缓存有效时间为 20 秒。执行程序，示例运行结果如图 13.4 所示；示例运行 10 秒内刷新页面，运行结果如图 13.5 所示；示例运行 20 秒后刷新页面，运行结果如图 13.6 所示。

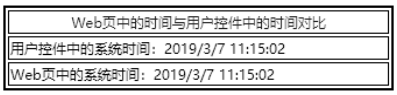


图 13.4　示例运行初期

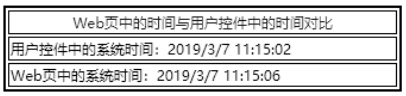


图 13.5　示例运行 10 秒内刷新页面

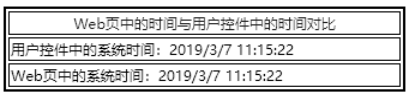


图 13.6　示例运行 20 秒后刷新页面

程序实现的主要步骤如下。

（1）新建一个网站，默认主页为 Default.aspx，并在该页中添加一个 Label 控件，用于显示当前系统时间。

（2）在该网站中添加一个用户控件，默认名为 WebUserControl.ascx，并在该用户控件中添加一个 Label 控件，用于显示当前系统时间。

（3）为了使用 PartialCachingAttribute 类设置用户控件（WebUserControl.ascx 文件）的缓存有效期时间为 20 秒，必须在用户控件类声明前设置[PartialCaching(20)]。代码如下：

```
using System;
using System.Collections;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Linq;
[PartialCaching(20)]                    //设置用户控件的缓存时间为 20 秒
public partial class WebUserControl : System.Web.UI.UserControl
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack)
        {
            this.Label1.Text = "用户控件中的系统时间：" + DateTime.Now.ToString();
        }
    }
}
```

说明

以上代码设置了缓存有效时间为 20 秒，这与在 WebUserControl.ascx 文件顶部设置@ OutputCache 指令的 Duration 属性值为 20 是完全一致的。

13.3.4 使用 ControlCachePolicy 类

ControlCachePolicy 是.NET Framework 中的类，主要用于提供对用户控件的输出缓存设置的编程访问。ControlCachePolicy 类包含 6 个属性，分别是 Cached、Dependency、Duration、SupportsCaching、VaryByControl 和 VaryByParams，如表 13.3 所示。

表 13.3　ControlCachePolicy 类的 6 个属性及其说明

属　　性	说　　明
Cached	用于获取或设置一个布尔值，该值指示是否为用户控件启用片段缓存
Dependency	获取或设置与缓存的用户控件输出关联的 CacheDependency 类的实例
Duration	获取或设置缓存的项将在输出缓存中保留的时间
SupportsCaching	获取一个值，该值指示用户控件是否支持缓存
VaryByControl	获取或设置要用来改变缓存输出的控件标识符列表
VaryByParams	获取或设置要用来改变缓存输出的 GET 或 POST 参数名称列表

下面通过一个典型示例说明该类的具体应用方法。

【例 13.3】使用 ControlCachePolicy 类实现设置用户控件缓存。（**示例位置：mr\TM\13\03**）

本示例主要演示如何在运行时动态加载用户控件、如何以编程方式设置用户控件缓存过期时间为 10 秒，以及如何使用绝对过期策略。执行程序，示例运行结果如图 13.7 所示；示例运行 5 秒内刷新页面，运行结果如图 13.8 所示；示例运行 10 秒后刷新页面，运行结果如图 13.9 所示。

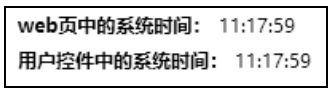


图 13.7　示例运行初期

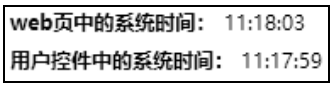


图 13.8　示例运行 5 秒内刷新页面图

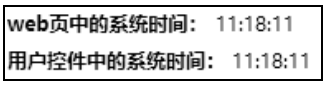


图 13.9　示例运行 10 秒后刷新页面

程序实现的主要步骤如下。

（1）新建一个网站，默认主页为 Default.aspx，并在该页中添加一个 Label 控件，用于显示当前系统时间。

（2）在该网站中添加一个用户控件，默认名为 WebUserControl.ascx，并在该用户控件中添加一个 Label 控件，用于显示当前系统时间。

（3）在用户控件（WebUserControl.ascx 文件）中，使用 PartialCachingAttribute 类设置用户控件的默认缓存有效时间为 100 秒，代码如下：

```
using System;
using System.Collections;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Linq;
//引入命名空间
using System.Data.SqlClient;
[PartialCaching(100)]
public partial class WebUserControl : System.Web.UI.UserControl
{
    protected void Page_Load (object sender, EventArgs e)
    {
        Label1.Text = DateTime.Now.ToLongTimeString();
    }
}
```

注意

使用 PartialCachingAttribute 类设置用户控件缓存过期时间的目的是实现使用 PartialCachingAttribute 类对用户控件类的包装，否则，在 ASP.NET 页中调用 CachePolicy 属性获取的 ControlCatchPolicy 实例是无效的。

（4）在 Default.aspx 页面的 Page_Init 事件下动态加载用户控件，并使用 SetSlidingExpiration 和

SetExpires 方法更改用户控件的缓存过期时间为 10 秒。Page_Init 事件的代码如下：

```
protected void Page_Init(object sender, EventArgs e)
    {
        this.Label1.Text = DateTime.Now.ToLongTimeString();
        //动态加载用户控件，并返回 PartialCachingControl 的实例对象
        PartialCachingControl pcc = LoadControl("WebUserControl.ascx") as PartialCachingControl;
        //如果用户控件的缓存时间大于 60 秒，那么重新设置缓存时间为 10 秒
        if (pcc.CachePolicy.Duration > TimeSpan.FromSeconds(60))
        {
            //设置用户控件过期时间
            pcc.CachePolicy.SetExpires(DateTime.Now.Add(TimeSpan.FromSeconds(10)));
            //设置缓存绝对过期
            pcc.CachePolicy.SetSlidingExpiration(false);
        }
        Controls.Add(pcc);  //将用户控件添加到页面中
    }
```

说明

（1）使用 TemplateControl.LoadControl 方法动态加载 WebUserControl.ascx 文件。由于用户控件 WebUserControl.ascx 已经为 PartialCachingAttribute 类包装，因此，LoadControl 方法的返回对象不是空引用，而是 PartialCachingControl 实例。

（2）使用 PartialCachingControl 实例对象的 CachePolicy 属性获取 ControlCachePolicy 实例对象。该对象主要用于对用户控件输出缓存进行设置，使用 SetExpires 方法和参数为 false 的 SetSlidingExpiration 方法，设置用户控件输出缓存有效期为 10 秒，并且设置缓存为绝对过期策略。

（3）利用 Controls 类的 Add 方法将设置好的用户控件添加到页面控件层次结构中。

13.4 页面数据缓存

13.4.1 页面数据缓存概述

页面数据缓存即应用程序数据缓存，它提供了一种编程方式，可通过键/值将任意数据存储在内存中。使用应用程序数据缓存与使用应用程序状态类似，但是与应用程序状态不同的是，应用程序数据缓存中的数据是很容易丢失的，即数据并不是在整个应用程序生命周期中都存储在内存中。应用程序数据缓存的优点是由 ASP.NET 管理缓存，它会在项过期、无效或内存不足时移除缓存中的项，还可以配置应用程序缓存，以便在移除项时通知应用程序。

ASP.NET 中提供了类似于 Session 的缓存机制，即页面数据缓存。利用数据缓存，可以在内存中存储各种与应用程序相关的对象。对于各个应用程序来说，数据缓存只是在应用程序内共享，并不能在应用程序间进行共享。Cache 类用于实现 Web 应用程序的缓存，在 Cache 中存储数据的最简单方法如下：

```
Cache["Key"]=Value;
```

从缓存中取数据时，需要先判断缓存中是否有内容，方法如下：

```
Value=(string)Cache["key"];
If(Value!=null)
{
//do something
}
```

误区警示

从 Cache 中得到的对象是一个 object 类型的对象，因此，在通常情况下，需要进行强制类型转换。

Cache 类有两个很重要的方法，即 Add 和 Insert 方法，其语法格式如下：

```
public Object Add[Insert] (
    string key,
    Object value,
    CacheDependency dependencies,
    DateTime absoluteExpiration,
    TimeSpan slidingExpiration,
    CacheItemPriority priority,
    CacheItemRemovedCallback onRemoveCallback
)
```

参数说明如下。

- ☑ key：用于引用该项的缓存键。
- ☑ value：要添加到缓存的项。
- ☑ dependencies：该项的文件依赖项或缓存键依赖项，当更改任何依赖项时，该对象即无效，并从缓存中移除，如果没有依赖项，则可将此参数设为 null。
- ☑ absoluteExpiration：过期的绝对时间。
- ☑ slidingExpiration：最后一次访问所添加对象时与该对象过期时之间的时间间隔。
- ☑ priority：缓存的优先级，由 CacheItemPriority 枚举表示。缓存的优先级共有 6 种，从大到小依次是 NotRemoveable、High、AboveNormal、Normal、BelowNormal 和 Low。
- ☑ onRemoveCallback：在从缓存中移除对象时所调用的委托（如果没有，可以为 null）。当从缓存中删除应用程序的对象时，它将会被调用。

Insert 方法声明与 Add 方法类似，但 Insert 方法为可重载方法，其结构如表 13.4 所示。

表 13.4　Insert 重载方法列表

重 载 方 法	说　明
Cache.Insert(String, Object)	向 Cache 对象插入项，该项带有一个缓存键引用其位置，并使用 CacheItemPriority 枚举提供的默认值
Cache.Insert(String, Object, CacheDependency)	向 Cache 中插入具有文件依赖项或键依赖项的对象
Cache.Insert(String, Object, CacheDependency, DateTime, TimeSpan)	向 Cache 中插入具有依赖项和过期策略的对象
Cache.Insert(String, Object, CacheDependency, DateTime, TimeSpan, CacheItemPriority, CacheItemRemovedCallback)	向 Cache 对象中插入对象，后者具有依赖项、过期和优先级策略以及一个委托（可用于在从 Cache 移除插入项时通知应用程序）

在 Insert 方法中，CacheDependency 是指依赖关系，DateTime 是有效时间，TimeSpan 是创建对象的时间间隔。

下面通过示例来讲解 Insert 方法的使用。

例如，将文件中的 XML 数据插入缓存，无须在以后请求时从文件中读取。CacheDependency 的作用是确保缓存在文件更改后立即到期，以便可以从文件中提取最新数据，重新进行缓存。如果缓存的数据来自若干个文件，还可以指定一个文件名的数组。代码如下：

```
Cache.Insert("key", myXMLFileData, new
System.Web.Caching.CacheDependency(Server.MapPath("users.xml")));
```

例如，插入键值为 key 的第 2 个数据块（取决于是否存在第 1 个数据块）。如果缓存中不存在名为 key 的键，或者与该键相关联的项已到期或被更新，那么 dependentkey 的缓存条目将到期。代码如下：

```
Cache.Insert("dependentkey", myDependentData, new
System.Web.Caching.CacheDependency(new string[] {}, new string[]
{"key"}));
```

下面是一个绝对到期的示例，此示例将对受时间影响的数据缓存 1 分钟，1 分钟过后，缓存将到期。

注意

绝对到期和滑动到期不能一起使用。

```
Cache.Insert("key", myTimeSensitiveData, null,
DateTime.Now.AddMinutes(1), TimeSpan.Zero);
```

下面是一个滑动到期的示例，此示例将缓存一些频繁使用的数据。数据将在缓存中一直保留，除非数据未被引用的时间达到了 1 分钟。

```
Cache.Insert("key", myFrequentlyAccessedData, null,
System.Web.Caching.Cache.NoAbsoluteExpiration,
TimeSpan.FromMinutes(1));
```

13.4.2 页面数据缓存的应用

【例 13.4】页面数据缓存的应用。**（示例位置：mr\TM\13\04）**

本示例主要演示如何利用 Cache 类实现应用程序数据缓存管理，包括添加、检索和删除应用程序数据缓存对象的方法。执行程序，并依次单击“添加”和“检索”按钮，示例运行结果如图 13.10 所示。

图 13.10　页面数据缓存的应用

程序实现的主要步骤如下。

（1）新建一个网站，默认主页为 Default.aspx。首先在该页中添加 3 个 Button 控件，分别用于执行添加数据缓存信息、检索数据缓存信息和移除数据缓存信息；然后添加 1 个 GridView 控件，用于显示数据信息；最后添加 2 个 Label 控件，分别用于显示缓存对象的个数和对缓存对象操作的信息。

（2）在 Default.aspx 页的 Page_Load 事件中，实现的主要功能是读取数据信息，并将其绑定到 GridView 控件。代码如下：

```
//判断缓存中是否包含关键字为 key 的数据，如果存在，读取缓存中的数据，否则，读取 XML 文件中的数据
protected void Page_Load(object sender, EventArgs e)
{
        DataSet ds = new DataSet();
        if (Cache["key"] == null)
        {
            ds.ReadXml(Server.MapPath("~/XMLFile.xml"));
            this.GridView1.DataSource = ds;
            this.GridView1.DataBind();
        }
        else
        {
            ds = (DataSet)Cache["key"];
            this.GridView1.DataSource = ds;
            this.GridView1.DataBind();
        }
}
```

（3）当用户单击“添加”按钮时，首先判断缓存中是否存在该数据，如果不存在，则将数据信息添加到缓存中。“添加”按钮的 Click 事件代码如下：

```
protected void Button3_Click(object sender, EventArgs e)
{
    if (Cache["key"] == null)          //判断缓存中是否存在该数据，如果不存在，则将数据信息添加到缓存中
    {
        DataSet ds = new DataSet();
        ds.ReadXml(Server.MapPath("~/XMLFile.xml"));
        Cache.Insert("key", ds, new System.Web.Caching.CacheDependency(Server.MapPath("XMLFile.xml")));
    }
    this.Label2.Text = "";
    DisplayCacheInfo();
}
```

（4）当用户单击“检索”按钮时，从缓存中检索指定的数据信息是否存在，并将检索的结果显示在界面中。“检索”按钮的 Click 事件代码如下：

```
protected void Button2_Click(object sender, EventArgs e)
{
    if (Cache["key"] != null)
    {
    this.Label2.Text ="已检索到缓存中包括该数据！";
    }
```

```
    else
    {
        this.Label2.Text = "未检索到缓存中包括该数据！";
    }
    DisplayCacheInfo();
}
```

（5）当用户单击“移除”按钮时，首先从缓存中判断指定的数据信息是否存在，如果存在，则从缓存中将指定的数据信息删除。“移除”按钮的 Click 事件代码如下：

```
protected void Button1_Click(object sender, EventArgs e)
{
    if (Cache["key"] == null) {
        this.Label2.Text = "未缓存该数据，无法删除！";
    }
    else{
        Cache.Remove("key");
        this.Label2.Text = "删除成功！";
    }
    DisplayCacheInfo();
}
```

13.5 实践与练习

尝试开发一个 ASP.NET 程序，要求缓存网页的不同版本，如缓存上海地区的客户信息、北京地区的客户信息等。