第六篇 性能优化

在计算机科学领域，广泛应用缓冲技术来提高系统的性能，它的原理是把经常存取的或者是比较重要的数据保存于内存中以减少系统的响应时间。对于 WEB 应用领域，缓冲技术主要是把 HTTP 请求的页面或数据保存于内存，以减少下次使用时重建它们的耗费。

ASP.NET 有两种用于 WEB 应用的缓冲技术：输出缓冲和数据缓冲。

输出缓冲指：把一次请求所产生的动态输出保存于内存中。

数据缓冲指：按照一定的策略把事先不确定的对象保存于内存中。

输出缓冲常用于把整个输出页面缓冲起来。对于一个存取繁忙的站点来说，把一些常用页面放入内存会带来性能上的极大提高。当一个页面被放入输出缓存，那么接下来的对该页面的请求将不再执行创建它的代码，而是从内存中直接返回该页面。

但实际上，保存整个输出页面的方法并不一定都行得通，因为有些页面的输出取决于客户端的不同请求，称之为“ 定制 ”。这时，采取的方法即找出不同中的相同，把一些并不需要经常重新创建的对象和数据识别出来，进行缓冲。一旦这些部分被识别，那么它们将被一次创建并在缓存中保持一定的时间。

选择缓存的时间是提高性能的关键。对一些部分来说，它们需要隔一定时间进行刷新，而另一些部分来说，可能仅仅只是需要保存一段时间。此种情况下，都可以设定“ 过期策略 ”来实现。一旦这些对象和数据到期，它们都将被从缓存中清除出去。当存取对象和数据的代码发现所要求的部分在内存中不存在时，将重建该对象或数据。

ASP.NET 支持文件和缓存关键字的依赖关系，它允许开发人员创建缓存依赖于一个外部文件或另一个缓存事物。利用这项技术可以更新一个缓存事物当其依赖的源文件发生改变时。

第一章 页面输出缓存

6.1.1 基本概念

页面输出缓存通过保存动态页面的输出内容，大大提高了服务器应用的能力。缺省情况下，输出缓存选项是被打开的，但并不是任意给定的输出响应都将被缓存，除非显示地指定页面应被缓存。

为使输出能够被缓存，输出响应至少应有一个有效的过期/有效策略以及公用 cache 的访问权限。当一个 GET 请求被送往页面，一个输出缓冲入口将被创建。接下来，对该页面的 GET 请求和 HEAD 的请求将直接从该缓冲入口中取出返回给用户，而对该页面的 POST 请求通常是显示地产生动态内容，却并非如同 GET 和 HEAD 请求一样从缓冲入口中取出。

输出缓存还支持带请求串的 GET 方法，把请求串作为页面识别的一部分。这就意味着带有相同键值但排列次序不同的请求串的 GET 请求，可能导致缓存中认为不存在该输出页面。

输出缓存需要知道页面缓存的过期/有效时间策略。如果一个页面在输出缓存中，而且又被指定为 60 分钟的页面过期时间，那么从它进入输出缓存开始，60 分钟后该页面将从输出缓存中被清除。如果恰在此时，有一个对该页面的请求到达，页面的代码将被执行，页面

输出又将重新进入输出缓冲。这种方式的过期策略称之为“ 强制过期 ”，页面只在一定时间内有效。

如下，我们可以用下面一条语句来显示的指出页面在输出缓冲中的保存时间。

<%@ OutputCache Duration=秒数 %>

6.1.2 实例

下面举一个简单的例子来证实 ASP.NET 中的页面缓存功能

在一个页加载时，我们显示它的时间，在页面过期时间（设为：10 秒）到达之前，我们把页面刷新（相当于重发 GET 请求），看一看显示的时间；然后，在过期时间到达之后，再看显示的时间。如果，第一次和第二次显示的时间相同，那么就证明了，系统存在有页面输出缓存功能，做为对比，当过期时间到达后，新的请求将导致重新执行页面代码，产生新的时间显示。

程序源程序

```
<!--文件名: performance\FormPageCache.aspx-->
<%@ OutputCache Duration="10" %>
<!--过期时间设为 10 秒-->
<html>
<head>
<title>
页面输出缓存测试
</title>
</head>
  <script language="VB" runat="server">
    Sub Page_Load(s As Object, E As EventArgs)
          lblTime.Text ="现在时间是:" &    DateTime.Now.ToString()
    End Sub
  </script>
  <body >
    <center>
    <h2><font face="Verdana">测试页面输出缓存实验</font></h2>
    <p><p><p>
    <hr>
    </center>
   <asp:label id="lblTime" runat="server"/>
  </body>
</html>
```

第一次输出效果：

第二次输出效果：

第三次输出效果：

实现页面缓存的另一种方法:

<%@ Import NameSpace=”System.Web.HttpCachePolicy” %>

‘指定页面输出缓存下一个 10 秒到期

Response.Cache.SetExpires(DateTime.Now.AddSeconds(10))

‘指定所有用户都有对缓存的访问权力

Response.Cache.SetCacheability(HttpCacheability.Public)

如果不希望进行页面缓存,可采用 Response.Cache.SetSlidingExpiration 方法,当其为 True 时，每次页面请求到达时，相当于页面过期时间到了，就要对页面输出重新刷新 。

‘当每次页面请求时,重置到期时间计数器，并且页面到期

Response.Cache.SetSlidingExpiration(True)

例如上面的例子可以改写成如下例子：

```
<!--文件名:performance\FormPageCache01.aspx-->
<%@ Import NameSpace="System.Web.HttpCachePolicy" %>
<html>
<head>
<title>
页面输出缓存测试 1
</title>
</head>
  <script language="VB" runat="server">
```

```
    Sub Page_Load(s As Object, E As EventArgs)
      response.cache.setexpires(DateTime.Now.addseconds(10))
      response.cache.setcacheability(Httpcacheability.Public)
      lblTime.Text ="现在时间是:" &    DateTime.Now.ToString()
    End Sub
  </script>
  <body >
    <center>
    <h2><font face="Verdana">测试页面输出缓存实验 1</font></h2>
    <p><p><p>
    <hr>
    </center>
   <asp:label id="lblTime" runat="server"/>
   </body>
</html>
```

输出的结果和上一例子大体相同，就不在重复了。

另外，在 ASP 中，对于上面的例子我们还可以写为：

```
Response.CacheControl = "Public"
Response.Expires = 10
```

从兼容性出发，ASP.NET 中依然具有相同的效果，但建议尽量使用 ASP.NET 的形式。

6.1.3 小结

本章介绍了页面输出缓存的基本概念，以及在 asp.net 中如何通过 page 命令和 response 对象在编程中实现页面输出缓存的实现方法。

第二章 页面数据缓存

6.2.1 基本概念

ASP.NET 提供了一个相当出色的缓存引擎机制，它允许页面保存和索引 HTTP 请求所要求的各种各样的对象。ASP.NET 的缓存对各个应用来说是私有的，是存储各种对象的存储器。缓存的生存周期取决于应用的生存周期，也就是说，当应用重新启动时，缓存实际上也已重建。

缓存提供了一个简单的字典接口，以便于开发者能够轻而易举放置对象到缓存中，并在以后使用。最简单的情况下，放置一个对象到缓存中，就如同对字典增加一个条目。

例如：

```
Cache(“myKey”)=MyValue
```

即是把 MyValue 放入缓存中一个叫 myKey 的缓存对象中，当需要引用 myKey 值时，可以采用下面的使用方式：

```
myValue1=Cache(“myKey”)
if myValue1 <> Null then
```

‘非空时的动作
…
end if

asp.net 提供了三种缓存替换的策略：

1.“ 腐烂搜索 ”（Scavenging）

比较类似于“ 最近最少使用 ”替换原则，当内存变得比教紧张时，缓存机制会找出最不常用和最不重要的对象，把它从内存中移出，以减轻系统压力。为控制“ 腐烂搜索 ”的具体行为，编程者必须在插入缓存对象时，指明插入它的耗费和多少时间内它必须被存取一次才能继续留在缓存中，以供替换时进行决择。

2.“ 到期控制 ”（Expiration）

编程者可以指定缓存对象的生存周期，这种指定的时间可以是绝对的也可以是相对的。例如绝对的时间（下午 6：00 到期），相对时间（该对象距最近一次存取它的时间满 10 分钟即到期）。当一个缓存对象到期后，它将从缓冲内存中移出，此时对该对象的索引将得到 null 值，除非又重新插入该对象。

3.“ 文件和键值依赖 ”

从外部文件或者是其他缓存键值是否改变，来决定本身键值是否有效。如果依赖发生改变，缓存对象将变得不可使用，并从缓存中移动出来。试想这样一种情况，应用从一个 XML 文件中读出金融信息，而该文件又被定期地修改。应用的作用是利用从该文件读出的信息构造一个图形对象以表示销售的情况。当应用读入文件时，它把数据插入缓存中，并记录下文件的依赖关系。当文件发生修改时，应用使开始产生的缓存对象无效并从内存中移出已经无用的数据，此后应用重新读入文件的数据，再把更新后的数据放入缓存，这样就完成了信息的更新，并返回给最终用户。

6.2.2 实例

例子：

从一个 XML 中读出用户信息并显示在页面上，页面提供了两个按钮，一个为增加用户，一个为刷新。我们在内存中建立了一个缓存对象“ DataCache5”,它与客户信息文件 ”custom1.xml”建立了依赖关系。当 custom1.xml 文件未发生变化时，我们按下“ 刷新 ”按钮，可以看到信息是从缓存中读出的；当我们输入客户相应的资料，增加了一个客户以后，再按下“ 刷新 “ 按钮后，我们可以看到客户信息变成了从文件读出。

1．ASPX 源程序（performance\FormDataCache.aspx）：

```
<!-- 文件名：FormDataCache.aspx -->
<%@ Import Namespace="System.IO" %>
<%@ Import Namespace="System.Data" %>
<html>
<script language="VB" runat="server">
sub LoadData1
'当缓存对象 DataCache5 有效时，从缓存中读出客户信息；无效时，从文件读出信息
dim dv1 as DataView
    dv1=Cache("DataCache5")
    if dv1 = Nothing
    dim ds as DataSet
```

```
        dim fs as FileStream
        dim sr as StreamReader

        ds=New DataSet
        fs=New FileStream(Server.MapPath("custom1.xml"),FileMode.Open,FileAccess.Read)
        sr=New StreamReader(fs)
        ds.ReadXml(sr)
        fs.Close()
        dv1=new DataView(ds.Tables(0))
        Cache.Insert("DataCache5",dv1,New cachedependency(Server.MapPath("custom1.xml")))
        lblMsg.text="数据从文件中读出..."
        Else
        lblmsg.text="数据从缓存中读出..."
        end if
'绑定到 DataGrid1 对象
        DataGrid1.datasource = dv1
        DataGrid1.databind()
end sub
sub Page_Load(s as object,e as eventargs)
'加载页面时，从文件中读出客户信息
        if Not IsPostBack
        LoadData1()
        end if
end sub
sub AddBtn_Click(s as object,e as eventargs)
'增加一个客户信息到文件中
dim FS as FileStream
dim Reader as StreamReader
dim DS as DataSet
dim dr1 as DataRow
dim tw1 as TextWriter
        if Not Page.IsValid
        lblMsg.text="还有域未曾填充..."
        else
        DS=New DataSet()
        FS=New
FileStream(Server.mappath("custom1.xml"),FileMode.Open,FileAccess.Read,FileShare.ReadWrit
e)
        Reader=New StreamReader(FS)
        DS.readxml(Reader)
                FS.Close()
        dr1=DS.tables(0).newrow()
                dr1("CustName")=txtName.text
        dr1("CustIdno")=txtIdno.text
```

```
        dr1("CustCard")=txtCard.text
        DS.tables(0).rows.add(dr1)

            FS=New
FileStream(Server.MapPath("custom1.xml"),FileMode.Create,FileAccess.ReadWrite,FileShare.Re
adWrite)
        tw1=New StreamWriter(FS)
        tw1=textwriter.synchronized(tw1)
        DS.writexml(tw1)
        tw1.close()
        LoadData1()
        end if
end sub
sub RefreshBtn_Click(s as object,e as eventargs)
        LoadData1()
end sub
</script>
<head>
<title>
数据缓冲实验
</title>
</head>
<body>
<center>
<form runat=server>
<h2>XML 文件缓存测试</h2>
<ASP:DataGrid id="DataGrid1" runat="server"
            Width="600"
            BorderColor="black"
            ShowFooter="false"
            CellPadding=3
            CellSpacing="0"
            Font-Name="Verdana"
            Font-Size="8pt"
                />
<hr>
<h3>添加一个客户信息</h3>
<table>
<tr>
<td>姓       名:</td>
<td><ASP:textbox id=txtName runat=server /></td>
<td><ASP:RequiredFieldValidator          ControlToValidate="txtName"          Display="static"
ErrorMessage="*" runat=server /> </td>
</tr>
```

```
<tr>
<td>身份证号:</td>
<td><ASP:textbox id=txtIdno runat=server /></td>
<td><ASP:RequiredFieldValidator ControlToValidate="txtIdno"
Display="static" ErrorMessage="*" runat=server /> </td>
</tr>
<tr>
<td>信用卡号:</td>
<td><ASP:textbox id=txtCard runat=server /></td>
<td><ASP:RequiredFieldValidator ControlToValidate="txtCard"
Display="static" ErrorMessage="*" runat=server /> </td>
</tr>
</table>
<p>
<asp:button text="增加" onclick="AddBtn_Click" runat=server />
<asp:button text="刷新" onclick="RefreshBtn_Click" runat=server />
<p><p><p>
<asp:label id=lblMsg runat=server />
</form>
</center>
</body>
</html>
```

2．初始运行效果

初始时，应用空间中无“ DataCache”缓存对象，故文件从 custom1.xml 中读出客户信息，开始时为空，只显示了字段名。

3．当添加一个用户信息后，文件 custom1.xml 发生改变，导致“ DataCache5”对象无效，

LoadData1 过程依然从文件 custom1.xml 中读取信息，并更新 DataCache5 对象。

4．由于 custom1.xml 文件未曾发生改变，所以当按下“ 刷新“ 按钮后，信息却是从缓存对象 DataCach5 中读出来。

6.2.3 小结

本章介绍了除页面输出缓存技术以外的另一种 asp.net 缓存技术，页面数据缓存。在我们的实践当中，页面数据缓存技术可能比页面输出缓存技术使用得更普遍一些。