

## Cache及(HttpRuntime.Cache与HttpContext.Current.Cache)

我们在.NET运用中经常用到缓存(Cache)对象。

除了System.Web.Caching下的Cache外，我们还可以用到HttpContext.Current.Cache以及HttpRuntime.Cache

那么，HttpContext.Current.Cache以及HttpRuntime.Cache有什么区别呢？

从MSDN上的解释可以看出，HttpRuntime.Cache是应用程序级别的，而HttpContext.Current.Cache是针对当前WEB上下文定义的。

然而，实际上，这二个都是调用的同一个对象，不同的是：HttpRuntime下的除了WEB中可以使用外，非WEB程序也可以使用。而HttpContext则只能用在WEB中。

因此，在可能的情况下，我们尽可能使用HttpRuntime(然而，在不同应用程序之间如何调用也是一个问题)。

=====

以下是摘录自：<http://www.cnblogs.com/caoxch/archive/2006/11/20/566236.html>

的对Cache的详细介绍

通常，应用程序可以将那些频繁访问的数据，以及那些需要大量处理时间来创建的数据存储在内存中，从而提高性能。例如，如果应用程序使用复杂的逻辑来处理大量数据，然后再将数据作为用户频繁访问的报表返回，避免在用户每次请求数据时重新创建报表可以提高效率。同样，如果应用程序包含一个处理复杂数据但不需要经常更新的页，则在每次请求时服务器都重新创建该页会使工作效率低下。

在这些情况下，为了帮助您提高应用程序的性能，ASP.NET 使用两种基本的缓存机制来提供缓存功能。第一种机制是应用程序缓存，它允许您缓存所生成的数据，如 **DataSet** 或自定义报表业务对象。第二种机制是页输出缓存，它保存页处理输出，并在用户再次请求该页时，重用所保存的输出，而不是再次处理该页。

### 应用程序缓存

应用程序缓存提供了一种编程方式，可通过键/值对将任意数据存储在内存中。使用应用程序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，应用程序缓存中的数据是易失的，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。使用应用程序缓存的优点是由 **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。有关更多信息，请参见[缓存应用程序数据](#)。

使用应用程序缓存的模式是，确定在访问某一项时该项是否存在于缓存中，如果存在，则使用。如果该项不存在，则可以重新创建该项，然后将其放回缓存中。这一模式可确保缓存中始终有最新的数据。

有关更多信息，请参见[如何：检索缓存项的值](#)。

### 页输出缓存

页输出缓存在内存中存储处理后的 **ASP.NET** 页的内容。这一机制允许 **ASP.NET** 向客户端发送页响应，而不必再次经过页处理生命周期。页输出缓存对于那些不经常更改，但需要大量处理才能创建的页特别有用。例如，如果创建大通信量的网页来显示不需要频繁更新的数据，页输出缓存则可以极大地提高该页的性能。可以分别为每个页配置页缓存，也可以在 **Web.config** 文件中创建缓存配置文件。利用缓存配置文件，只定义一次缓存设置就可以在多个页中使用这些设置。

页输出缓存提供了两种页缓存模型：整页缓存和部分页缓存。整页缓存允许将页的全部内容保存在内存中，并用于完成客户端请求。部分页缓存允许缓存页的部分内容，其他部分则为动态内容。有关更多信息，请参见[缓存 ASP.NET 页](#)。

部分页缓存可采用两种工作方式：控件缓存和缓存后替换。控件缓存有时也称为分段缓存，这种方式允许将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，以此来缓存页输出的部分内容。这一方式可缓存页中的特定内容，并不缓存整个页，因此每次都需重新创建整个页。例如，如果要创建一个显示大量动态内容（如股票信息）的页，其中有些部分为静态内容（如每周总结），这时可以将静态部分放在用户控件中，并允许缓存这些内容。

#### 公告

昵称: McJeremy&Fan  
园龄: 8年8个月  
粉丝: 44  
关注: 6  
[+ 加关注](#)

<	2008年12月			
日	一	二	三	
30	<u>1</u>	2	3	
7	<u>8</u>	<u>9</u>	10	
14	15	<u>16</u>	17	
21	22	23	24	
28	29	30	31	
4	5	6	7	

#### 搜索

#### 最新随笔

1. SQL中获取最近的N个数据
2. 【转】SQL FOR XML
3. JQuery之拖拽插件
4. SQL中SET和SELECT
5. JS定时保存表单数据
6. 求解：AJAX轮询跨域
7. ASP.NET Request 元素值）
8. Js下的StringBuilder
9. SharePoint判断页面

缓存后替换与控件缓存正好相反。这种方式缓存整个页，但页中的各段都是动态的。例如，如果要创建一个在规定时间内为静态的页，则可以将整个页设置为进行缓存。如果向页添加一个显示用户名的 **Label** 控件，则对于每次页刷新和每个用户而言，**Label** 的内容都将保持不变，始终显示缓存该页之前请求该页的用户的姓名。但是，使用缓存后替换机制，可以将页配置为进行缓存，但将页的个别部分标记为不可缓存。在此情况下，可以向不可缓存部分添加 **Label** 控件，这样将为每个用户和每次页请求动态创建这些控件。有关更多信息，请参见[缓存 ASP.NET 页的某些部分](#)。

根据请求参数缓存页

除缓存页的单一版本外，ASP.NET 页输出缓存还提供了一些功能，可以创建根据请求参数的不同而不同的页的多个版本。有关更多信息，请参见[缓存页的多个版本](#)。

自动移除数据

出于以下原因之一，ASP.NET 可以从缓存中移除数据：

- 由于服务器上的内存不足，开始一个称为“清理”的过程。
- 由于缓存中的项已过期。
- 由于项的依赖项发生了更改。

为了帮助管理缓存项，在将项从缓存中移除时，ASP.NET 会通知应用程序。

清理

清理是在内存不足时从缓存中删除项的过程。如果某些项在一段时间内未被访问，或是在添加到缓存中时被标记为低优先级，则这些项会被移除。ASP.NET 使用 **CacheItemPriority** 对象来确定要首先清理的项。有关更多信息，请参见[如何：将项添加到缓存中](#)。

过期

除了清理外，在缓存项过期时，ASP.NET 会自动从缓存中移除这些项。向缓存添加项时，可以按下表中的描述设置其过期时间。

过期类型	说明
可调过期	指定某项自上次被访问后多长时间过期。例如，可以将某项设置为自上次在缓存中被访问后 <b>20</b> 分钟过期。
绝对过期	指定某项在设定的时间过期，而不考虑访问频率。例如，可以将某项设置为在 <b>6:00 PM</b> 过期，或四小时后过期。

依赖项

可以将缓存中某一项的生存期配置为依赖于其他应用程序元素，如某个文件或数据库。当缓存项依赖的元素更改时，ASP.NET 将从缓存中移除该项。例如，如果您的网站显示一份报告，该报告是应用程序通过 XML 文件创建的，您可以将该报告放置在缓存中，并将其配置为依赖于该 XML 文件。当 XML 文件更改时，ASP.NET 会从缓存中移除该报告。当代码请求该报告时，代码会先确定该报告是否在缓存中，如果不在，代码会重新创建该报告。因此，始终都有最新版本的报告可用。

ASP.NET 缓存支持下表中描述的依赖项。

依 赖 项	说明
键依赖项	应用程序缓存中的项存储在键/值对中。键依赖项允许项依赖于应用程序缓存中另一项的键。如果移除了原始项，则具有键依赖关系的项也会被移除。例如，可以添加一个名为 ReportsValid 的缓存项，然后缓存若干个依赖于 ReportsValid 键的报告。当 ReportsValid 项被移除时，所有依赖于它的缓存报告同样也会从缓存中移除。
文件依赖项	缓存中的项依赖于外部文件。如果该文件被修改或删除，则缓存项也会被移除。
SQL 依赖项	缓存中的项依赖于 Microsoft SQL Server 2005、SQL Server 2000 或 SQL Server 7.0 数据库中表的更改。对于 SQL Server 2005，缓存中的项可依赖于表中的某一行。有关更多信息，请参见 <a href="#">使用 SqlCacheDependency 类在 ASP.NET 中缓存</a> 。
聚合依赖项	通过使用 <b>AggregateCacheDependency</b> 类缓存中的项依赖于多个元素。如果任何依赖项发生更改，该项都会从缓存中移除。
自定	

否处于编辑模式.

10. 有道JavaScript面试题(附一解决方案)

我的标签

SharePoint(16)

Javascript(15)

SQL(6)

asp.net(5)

Moss(4)

NVelocity(3)

Ajax(3)

HttpRequest(2)

Dom(2)

c#(2)

更多

随笔分类(156)

ASP.NET(C#)(72)

C++（vc++,vc++.ne

Effect C#(1)

Javascript(ajax)(33)

PHP(2)

SharePoint(18)

SQL(TSQL)(18)

XHTML+CSS(5)

天下杂侃(1)

随笔档案(114)

2015年12月 (1)

义依赖项	可以用您自己的代码创建的依赖关系来配置缓存中的项。例如，可以创建一个自定义 Web 服务缓存依赖项，当调用 Web 服务得到一个特定值时，该依赖项就会从缓存中移除数据。
------	--

## 应用程序缓存项移除通知

当项从应用程序缓存中移除时，您可以收到通知。例如，如果有一个需要大量处理时间才能创建的项，当从缓存中移除该项时，您会收到通知以便可以立即替换该项。这样，下次请求该项时，用户便不必等待处理该项。有关更多信息，请参见[如何：从缓存中移除项时通知应用程序](#)。

### ASP.NET 缓存中的新增功能

ASP.NET 2.0 版保留了 ASP.NET 1.1 版的所有缓存功能，同时添加了新功能并增强了现有功能。新功能包括缓存配置文件、自定义缓存依赖项、SQL 缓存依赖项以及在缓存页中创建动态内容（缓存后替换）。增强功能包括功能更强大的部分页（控件）缓存模型、增强的缓存配置以及输出缓存指令的改进。

## 新缓存功能

### 缓存配置文件

缓存配置文件使您能够在应用程序的 Web.config 文件中创建缓存设置，然后在单个页上引用这些设置。这使您能够将缓存设置同时应用于多页。例如，可以定义一个名为 **DailyUpdate** 的缓存配置文件，它将页的缓存持续时间设置为一天。然后可以配置各个页使用 **DailyUpdate** 缓存配置文件，并且这些页的缓存持续时间为一天。如果将 **DailyUpdate** 缓存配置文件更改为不使用缓存，将停止缓存这些页。有关更多信息，请参见 [ASP.NET 中的缓存配置](#)。

### 自定义缓存依赖项

在 ASP.NET 2.0 中，您可以根据应用程序特定情况创建自己的自定义缓存依赖项。若要创建自定义缓存依赖项，请创建从 **CacheDependency** 继承的类并在自定义类中实现您自己的依赖项方法。例如，您可以创建在 Web 服务中轮询数据的依赖项；当数据发生变化时，您可以使缓存数据无效。若要了解通过指定依赖项向缓存添加项目的信息，请参见[如何：将项添加到缓存中](#)。

### SqlCacheDependency

ASP.NET 2.0 引入了 **SqlCacheDependency** 类，它使您能够在缓存中配置一个项，以便在 Microsoft SQL Server 数据库中的表或行上拥有依赖项。当表中或特定行中发生更改时，具有依赖项的缓存项便会失效并从缓存中移除。ASP.NET 2.0 使您能够在 SQL Server 7.0、SQL Server 2000 和 SQL Server 2005 中设置表的依赖项。使用 SQL Server 2005 时，您还可以设置特定记录的依赖项。有关更多信息，请参见[使用 SqlCacheDependency 类在 ASP.NET 中缓存](#)。

### 缓存后替换

ASP.NET 2.0 现在支持缓存后替换，使您能够将页中的某一部分配置为不可缓存。因此，尽管缓存了该页，但在再次请求该页时，将重新处理它的部分内容。例如，您可以使用大多数静态内容（但不能使用在 **Label** 控件中显示用户名的内容）创建缓存页。如果不使用缓存后替换，用户名在所有请求中保持不变。如果使用缓存后替换，您可以将页标记为可缓存，然后将 **Label** 控件放置在标记为不可缓存的另一个控件中。此后每次请求该页时，都会刷新用户名。有关更多信息，请参见[缓存 ASP.NET 页的某些部分](#)。

## 缓存增强

### 控件缓存

在 ASP.NET 1.1 中，通过设置 **@ Control** 指令中的参数以声明方式配置用户控件缓存。在 ASP.NET 2.0 中，可以在运行时使用 **CachePolicy** 对象配置用户控件缓存设置。**CachePolicy** 对象使您能够按照以编程方式处理页输出缓存的相同方式处理用户控件缓存。有关更多信息，请参见[缓存 ASP.NET 页的某些部分](#)。

### 缓存配置增强

除了缓存配置文件外，ASP.NET 2.0 中还引入了新的缓存配置设置，可以在应用程序的 Web.config 文件中指定这些设置。这些设置增加了您对缓存的控制，如内存使用量和缓存清理行为。有关更多信息，请参见 [ASP.NET 中的缓存配置](#)。

### 输出缓存指令改进

ASP.NET 2.0 包括新的 **@ OutputCache** 指令选项以及对现有选项的增强。新功能和增强功能使您能够对输出缓存功能进行声明控制，而以前只能使用 **HttpCachePolicy** 类以编程方式实现此类控制。例如，现在可以用声明方式设置页 **@ OutputCache** 指令中的 **Duration** 属性和 **NoStore** 属性。有关更多信息，请参见[设置页的可缓存性](#)。

### 缓存应用程序数据

ASP.NET 为您提供了一个强大的、便于使用的缓存机制，用于将需要大量服务器资源来创建的对象存储在内存中。缓存这些类型的资源会大大改进应用程序的性能。

缓存是由 **Cache** 类实现的；缓存实例是每个应用程序专用的。缓存生存期依赖于应用程序的生存期；重新启动应用程序后，将重新创建 **Cache** 对象。

2013年8月 (1)
2011年4月 (1)
2011年3月 (3)
2011年2月 (1)
2010年10月 (1)
2010年7月 (1)
2010年6月 (3)
2010年5月 (2)
2010年3月 (13)
2009年12月 (1)
2009年11月 (4)
2009年9月 (3)
2009年8月 (3)
2009年7月 (12)
2009年6月 (2)
2009年5月 (1)
2009年4月 (9)
2009年3月 (1)
2009年1月 (5)
2008年12月 (6)
2008年11月 (4)
2008年10月 (5)
2008年9月 (8)
2008年8月 (1)
2008年7月 (1)
2008年6月 (10)

设计 **Cache** 类是为了便于使用。您可以将项放置在 **Cache** 中，并在以后使用简单的键/值对来检索这些项。有关如何执行此操作的示例，请参见[如何：将项添加到缓存中](#)和[如何：检索缓存项的值](#)。

**Cache** 类提供了强大的功能，允许您自定义如何缓存项以及将它们缓存多长时间。例如，当缺乏系统内存时，缓存会自动移除很少使用的或优先级较低的项以释放内存。该技术也称为清理，这是缓存确保过期数据不使用宝贵的服务器资源的方式之一。

当执行清理时，您可以指示 **Cache** 给予某些项比其他项更高的优先级。若要指示项的重要性，可以在使用 [Add](#) 或 [Insert](#) 方法添加项时指定一个 [CacheItemPriority](#) 枚举值。

当使用 [Add](#) 或 [Insert](#) 方法将项添加到缓存时，您还可以建立项的过期策略。您可以通过使用 [DateTime](#) 值指定项的确切过期时间（绝对过期时间），来定义项的生存期。也可以使用 [TimeSpan](#) 值指定一个弹性过期时间，弹性过期时间允许您根据项的上次访问时间来自定义该项过期之前的运行时间。一旦项过期，便将它从缓存中移除。试图检索它的值的行为将返回 **null**（在 Visual Basic 中为 **Nothing**），除非该项被重新添加到缓存中。

对于存储在缓存中的易失项（例如那些定期进行数据刷新的项或那些只在一段时间内有效的项），通常设置一种过期策略：只要这些项的数据保持为最新的，就将它们保留在缓存中。例如，如果您正在编写一个应用程序，该应用程序通过另一个网站获取数据来跟踪体育比赛的比分，那么只要源网站上比赛的比分不更改，就可以缓存这些比分。在此情况下，您可以根据其他网站更新比分的频率来设置过期策略。您可以编写代码来确定缓存中是否是最新的比分。如果该比分不是最新的，则代码可以从源网站读取比分并缓存新值。

最后，ASP.NET 允许您根据外部文件、目录（文件依赖项）或另一个缓存项（键依赖项）来定义缓存项的有效性。如果具有关联依赖项的项发生更改，缓存项便会失效并从缓存中移除。您可以使用该技术来在项的数据源更改时从缓存中移除这些项。例如，如果您编写一个处理 XML 文件中的财务数据的应用程序，则可以从该文件将数据插入缓存中并在此 XML 文件上保留一个依赖项。当该文件更新时，从缓存中移除该项，您的应用程序重新读取 XML 文件，然后将刷新后的数据放入缓存中。

注意

**Cache** 对象没有关于它所包含项的内容的信息。它只保留对这些对象的引用。它还提供跟踪它们的依赖项和设置到期策略的方法。

ASP.NET 中的缓存配置

ASP.NET 提供了许多可用于配置页面输出缓存和缓存 API 的选项。可以在处理了页面响应后使用页面输出缓存来缓存此页面响应。也可以通过编程的方式使用缓存 API 来缓存应用程序数据。有关更多信息，请参见 [ASP.NET 缓存概述](#)。

## 页面输出缓存配置

您可以在以下这些位置配置页面输出缓存：

- 配置文件 可以在应用程序配置层次结构的任何配置文件中配置页面输出缓存设置，包括 **Machine.config** 文件（用于设置计算机上所有的 Web 应用程序）和特定于应用程序的 **Web.config** 文件（用于设置单个应用程序）。
- 单个页面 可以在单个页面中以声明方式或编程方式设置缓存选项。还可将在配置文件中创建的缓存配置文件应用于单个页面。
- 用户控件 可以在单个用户控件中以声明方式或编程方式设置缓存。对于在其他情况下不缓存的页面内容来说，这是一种简便的缓存方法。

### Web.config 缓存配置设置

在 Web.config 文件中，有两个顶级配置节可用于页输出缓存：[OutputCacheSection](#) 和 [OutputCacheSettingsSection](#)。

**OutputCacheSection** 节用于配置应用程序范围的设置，例如是启用还是禁用页输出缓存。例如，您可以通过向 Web.config 文件中的 **OutputCacheSection** 添加 `enableOutputCache="false"` 来对整个应用程序禁用页输出缓存。由于配置文件中的设置要优先于单个页面中的缓存设置，因此，示例设置将导致不使用输出缓存。

**OutputCacheSettingsSection** 用于配置可由单个页使用的配置文件和依赖项。例如，下面的代码创建了一个名为 CacheProfile1 的 [OutputCacheProfile](#)，它将缓存实现页 60 秒：

```
<outputCacheSettings>  
  <outputCacheProfiles>  
    <add name="CacheProfile1" duration="60" />  
  </outputCacheProfiles>  
</outputCacheSettings>
```

### Machine.config 缓存配置设置

Machine.config 文件的配置节与 Web.config 文件的配置节基本相同，而只有一点区别：即可以锁定 Machine.config 文件中的配置设置，使任何级别的单个应用程序都无法重写这些配置设置。在宿主不希望单个应用程序修改缓存配置时，可能有必要在共享宿主方案中使用此功能。有关更多信息，请参见[如何：锁定 ASP.NET 配置设置](#)。

### 页面缓存配置设置

通过应用在配置文件中定义的缓存配置文件，可以配置单个页中的缓存。也可以在 [@ OutputCache](#) 指令中配置单个缓存属性 (property)，或者通过设置页的类定义中的属性 (attribute) 进行配置。有关更多信息，请参见 [@ OutputCache](#) 和 [设置页的可缓存性](#)。

2008年5月 (11)

最新评论

1. Re:Cache及(HttpRuntime.Cache与HttpContext.Current.Cache) - 博文要顶

2. Re:NET许可证及License - 好文，还的自己写一遍才能理解，还是不怎么了解证书

3. Re:前端面试总结 - mark

4. Re:jQuery之拖拽插件 - 猛拖拽会卡掉

5. Re:JavaScript抽象类 - 备忘  
心静如高山不动，心浮如

阅读排行榜

1. JQuery之拖拽插件(转载) - 博文要顶

2. 对SendMessage与Find - 解(21245)

3. SQL创建索引（转载） - 博文要顶

4. Cache及(HttpRuntime.Cache与HttpContext.Current.Cache) - 博文要顶

5. SQL中SET和SELECT - 830)

评论排行榜

1. 有道JavaScript监听 - 一解决方案)(27)

用户控件缓存配置设置

通过设置用户控件文件中的 [@ OutputCache](#) 指令，或设置控件类定义中的 [PartialCachingAttribute](#) 属性，可以对用户控件缓存进行配置。有关更多信息，请参见[缓存 ASP.NET 页的某些部分](#)。

缓存 API 配置设置

可以在 `Web.config` 文件中配置应用程序的缓存 API。对于页面输出缓存，应用程序宿主可以在 `Machine.config` 文件中设置配置属性，并锁定所有应用程序的缓存配置设置。应用程序缓存 API 在 [CacheSection](#) 中进行配置。例如，您可以使用下面的配置元素来禁用项过期：

```
<cache disableExpiration="true" />
```

还可以通过为属性（如配置文件的 **CacheSection** 中的 [DisableExpiration](#) 和 [DisableMemoryCollection](#) 属性）赋值的方式来指定其他应用程序缓存 API 配置设置。

如何：将项添加到缓存中

可以使用 [Cache](#) 对象访问应用程序缓存中的项。可以使用 **Cache** 对象的 [Insert](#) 方法向应用程序缓存添加项。该方法向缓存添加项，并且通过几次重载，您可以使用不同选项添加项，以设置依赖项、过期和移除通知。如果使用 **Insert** 方法向缓存添加项，并且已经存在与现有项同名的项，则缓存中的现有项将被替换。

还可以使用 [Add](#) 方法向缓存添加项。使用此方法，您可以设置与 **Insert** 方法相同的所有选项；然而，**Add** 方法将返回您添加到缓存中的对象。另外，如果使用 **Add** 方法，并且缓存中已经存在与现有项同名的项，则该方法不会替换该项，并且不会引发异常。

本主题中的过程阐释了向应用程序缓存添加项的如下方式：

- 通过键和值直接设置项，向缓存添加项。
- 使用 **Insert** 方法向缓存添加项。
- 向缓存添加项并添加依赖项，以便当该依赖项更改时，将该项从缓存中移除。可以基于其他缓存项、文件和多个对象设置依赖项。
- 将设有过期策略的项添加到缓存中。除了能设置项的依赖项以外，还可以设置项在一段时间以后（弹性过期）或在指定时间（绝对过期）过期。您可以定义绝对过期时间或弹性过期时间，但不能同时定义两者。
- 向缓存添加项，并定义缓存的项的相对优先级。相对优先级帮助 .NET Framework 确定要移除的缓存项；较低优先级的项比较高优先级的项先从缓存中移除。
- 通过调用 **Add** 方法添加项。

除了这里显示的依赖项，可以在 `SQL Server` 表上或基于自定义依赖项创建依赖项。有关更多信息，请参见 [ASP.NET 缓存概述和使用 SqlCacheDependency 类在 ASP.NET 中缓存](#)。

当从缓存中移除项时，还可以使用 [CacheItemRemovedCallback](#) 委托让应用程序缓存通知应用程序。有关完整示例，请参见[如何：从缓存中移除项时通知应用程序](#)。

通过键和值直接设置项向缓存添加项

- 通过指定项的键和值，像将项添加到字典中一样将其添加到缓存中。

下面的代码示例将名为 `CacheItem1` 的项添加到 **Cache** 对象中：

```
C#
复制代码
Cache["CacheItem1"] = "Cached Item 1";

Visual Basic
复制代码
Cache("CacheItem1") = "Cached Item 1"
```

通过使用 Insert 方法将项添加到缓存中

- 调用 **Insert** 方法，传递要添加的项的键和值。

下面的代码示例添加名为 `CacheItem2` 的字符串：

```
C#
复制代码
Cache.Insert("CacheItem2", "Cached Item 2");

Visual Basic
复制代码
Cache.Insert("CacheItem2", "Cached Item 2")
```

2. 读：<测试一下你解  
及算法能力>后(15)

3. 用javascript+asp.r  
的经验总结。(11)

4. C#中调用Outlook /

5. JQuery之拖拽插件(

推荐排行榜

1. Cache及(HttpRunt  
pContext.Current.Ca

2. Javascript监视变量

3. JavaScript布尔型数  
符。(3)

4. JQuery之拖拽插件(

5. JavaScript抽象类及  
忘(2)

## 通过指定依赖项向缓存添加项

- 调用 **Insert** 方法，将 **CacheDependency** 对象的一个实例传递给该方法

下面的代码示例添加名为 CacheItem3 的项，该项依赖于缓存中名为 CacheItem2 的另一个项：

C#

复制代码

```
string[] dependencies = { "CacheItem2" };
Cache.Insert("CacheItem3", "Cached Item 3",
    new System.Web.Caching.CacheDependency(null, dependencies));
```

Visual Basic

复制代码

```
Dim dependencies As String() = {"CacheItem2"}
Cache.Insert("CacheItem3", "Cached Item 3", _
    New System.Web.Caching.CacheDependency( _
    Nothing, dependencies))
```

下面的代码示例演示将名为 CacheItem4 的项添加到缓存中，并且在名为 XMLFile.xml 的文件上设置文件依赖项：

C#

复制代码

```
Cache.Insert("CacheItem4", "Cached Item 4",
    new System.Web.Caching.CacheDependency(
    Server.MapPath("XMLFile.xml")));
```

Visual Basic

复制代码

```
Cache.Insert("CacheItem4", "Cached Item 4", _
    New System.Web.Caching.CacheDependency( _
    Server.MapPath("XMLFile.xml")))
```

下面的代码示例演示如何创建多个依赖项。它向缓存中名为 CacheItem1 的另一个项添加键依赖项，向名为 XMLFile.xml 的文件添加文件依赖项。

C#

复制代码

```
System.Web.Caching.CacheDependency dep1 =
    new System.Web.Caching.CacheDependency(Server.MapPath("XMLFile.xml"));
string[] keyDependencies2 = { "CacheItem1" };
System.Web.Caching.CacheDependency dep2 =
    new System.Web.Caching.CacheDependency(null, keyDependencies2);
System.Web.Caching.AggregateCacheDependency aggDep =
    new System.Web.Caching.AggregateCacheDependency();
aggDep.Add(dep1);
aggDep.Add(dep2);
Cache.Insert("CacheItem5", "Cached Item 5", aggDep);
```

Visual Basic

复制代码

```
Dim dep1 As CacheDependency = _
    New CacheDependency(Server.MapPath("XMLFile.xml"))
Dim keyDependencies2 As String() = {"CacheItem1"}
Dim dep2 As CacheDependency = _
    New System.Web.Caching.CacheDependency(Nothing, _
    keyDependencies2)
Dim aggDep As AggregateCacheDependency = _
    New System.Web.Caching.AggregateCacheDependency()
aggDep.Add(dep1)
aggDep.Add(dep2)
Cache.Insert("CacheItem5", "Cached Item 5", aggDep)
```

## 将设有过期策略的项添加到缓存中

- 调用 **Insert** 方法，将绝对过期时间或弹性过期时间传递给该方法。

下面的代码示例将有一分钟绝对过期时间的项添加到缓存中：



C#

[复制代码](#)

```
Cache.Insert("CacheItem6", "Cached Item 6",  
    null, DateTime.Now.AddMinutes(1d),  
    System.Web.Caching.Cache.NoSlidingExpiration);
```

Visual Basic

[复制代码](#)

```
Cache.Insert("CacheItem6", "Cached Item 6", _  
    Nothing, DateTime.Now.AddMinutes(1.0), _  
    TimeSpan.Zero)
```

下面的代码示例将有 10 分钟弹性过期时间的项添加到缓存中：

C#

[复制代码](#)

```
Cache.Insert("CacheItem7", "Cached Item 7",  
    null, System.Web.Caching.Cache.NoAbsoluteExpiration,  
    new TimeSpan(0, 10, 0));
```

Visual Basic

[复制代码](#)

```
Cache.Insert("CacheItem7", "Cached Item 7", _  
    Nothing, System.Web.Caching.Cache.NoAbsoluteExpiration, _  
    New TimeSpan(0, 10, 0))
```

## 将设有优先级设置的项添加到缓存中

- 调用 **Insert** 方法，从 [CacheItemPriority](#) 枚举中指定一个值。

下面的代码示例将优先级值为 **High** 的项添加到缓存中：

C#

[复制代码](#)

```
Cache.Insert("CacheItem8", "Cached Item 8",  
    null, System.Web.Caching.Cache.NoAbsoluteExpiration,  
    System.Web.Caching.Cache.NoSlidingExpiration,  
    System.Web.Caching.CacheItemPriority.High, null);
```

Visual Basic

[复制代码](#)

```
Cache.Insert("CacheItem8", "Cached Item 8", _  
    Nothing, System.Web.Caching.Cache.NoAbsoluteExpiration, _  
    System.Web.Caching.Cache.NoSlidingExpiration, _  
    System.Web.Caching.CacheItemPriority.High, _  
    Nothing)
```

## 使用 **Add** 方法向缓存添加项

- 调用 **Add** 方法，它返回一个表示项的对象。

下面的代码示例向缓存添加名为 CacheItem9 的项，同时将变量 CachedItem9 的值设置为已添加的项。

C#

[复制代码](#)

```
string CachedItem9 = (string)Cache.Add("CacheItem9",  
    "Cached Item 9", null,  
    System.Web.Caching.Cache.NoAbsoluteExpiration,  
    System.Web.Caching.Cache.NoSlidingExpiration,  
    System.Web.Caching.CacheItemPriority.Default,  
    null);
```

Visual Basic

[复制代码](#)

```
Dim CachedItem9 As String = CStr(Cache.Add("CacheItem9", _  
    "Cached Item 9", Nothing, _  
    System.Web.Caching.Cache.NoAbsoluteExpiration, _  
    System.Web.Caching.Cache.NoSlidingExpiration, _  
    System.Web.Caching.CacheItemPriority.Default, _
```

Nothing))

如何：检索缓存项的值

要从缓存中检索数据，应指定存储缓存项的键。不过，由于缓存中所存储的信息为易失信息，即该信息可能由 ASP.NET 移除，因此建议的开发模式是首先确定该项是否在缓存中。如果不在，则应将它重新添加到缓存中，然后检索该项。

## 检索缓存项的值

- 通过在 **Cache** 对象中进行检查来确定该项是否不为 null（在 Visual Basic 中为 Nothing）。如果该项存在，则将它分配给变量。否则，重新创建该项，将它添加到缓存中，然后访问它。

下面的代码示例演示如何从缓存中检索名为 CacheItem 的项。代码将该项的内容分配给名为 cachedString 的变量。如果该项不在缓存中，则代码会将其添加到缓存中，然后将它分配给 cachedString。

C#

复制代码

```
string cachedString;
cachedString = (string)Cache["CacheItem"];
if (cachedString == null)
{
    cachedString = "Hello, World.";
    Cache.Insert("CacheItem", cachedString);
}
```

Visual Basic

复制代码

```
Dim cachedString As String
cachedString = CStr(Cache("CacheItem"))
If cachedString Is Nothing Then
    cachedString = "Hello, World."
    Cache.Insert("CacheItem", cachedString)
End If
```

如何：从缓存中移除项时通知应用程序

在大多数缓存方案中，当从缓存中移除项后，直到再次需要此项时，才需要将其放回缓存中。典型的开发模式是在使用项之前始终检查该项是否已在缓存中。如果项位于缓存中，则可以使用。如果不在缓存中，则应再次检索该项，然后将其添加回缓存。

但是，在某些情况下，如果从缓存中移除项时通知应用程序，可能非常有用。例如，您可能具有一个缓存的报告，创建该报告需花费大量的时间进行处理。当该报告从缓存中移除时，您希望重新生成该报告，并立即将其置于缓存中，以便下次请求该报告时，用户不必等待对此报告进行处理。

为了在从缓存中移除项时能够发出通知，ASP.NET 提供了 **CacheItemRemovedCallback** 委托。该委托定义编写事件处理程序时使用的签名，当对从缓存中移除项进行响应时会调用此事件处理程序。ASP.NET 还提供 **CacheItemRemovedReason** 枚举，用于指定移除缓存项的原因。

通常，通过在管理尝试检索的特定缓存数据的业务对象中创建处理程序，来实现回调。例如，您可能有一个 ReportManager 对象，该对象具有两种方法，即 GetReport 和 CacheReport。GetReport 报告方法检查缓存以查看报告是否已缓存；如果没有，该方法将重新生成报告并将其缓存。CacheReport 方法具有与 CacheItemRemovedCallback 委托相同的函数签名；从缓存中移除报告时，ASP.NET 会调用 CacheReport 方法，然后将报告重新添加到缓存中。

## 当从缓存中移除项时通知应用程序

1. 创建一个类，负责从缓存中检索项并处理回调方法，以将项添加回缓存中。
2. 在该类中，创建用于将项添加到缓存中的方法。
3. 在该类中，创建用于从缓存中获取项的方法。
4. 创建用于处理缓存项移除回调的方法。该方法必须具备与 CacheItemRemovedCallback 委托相同的函数签名。从缓存中删除项时，会在该方法中执行要运行的逻辑，如重新生成项并将其添加回缓存中。

## 测试缓存项回调

1. 创建一个 ASP.NET 网页，该网页将调用类中用于将项添加到缓存中的方法。

下面的代码示例演示如何调用 ReportManager 类的 GetReport 方法（在此过程后面的示例中定义）。然后将在使用页面的 Page\_Load 方法期间显示 **Label** 控件 Label1 中的报告。

C#

复制代码



```
protected void Page_Load(object sender, EventArgs e)
{
    this.Label1.Text = ReportManager.GetReport();
}

Visual Basic
复制代码
Protected Sub Page_Load(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Me.Load
    Me.Label1.Text = ReportManager.GetReport()
End Sub
```

2. 在浏览器中请求 ASP.NET 页并查看报告。
- 报告是在首次请求页时创建的，在缓存中的报告被移除之前，后续请求都将访问缓存中的报告。

## 示例

下面的代码示例演示一个名为 ReportManager 的、用于在从缓存中删除项时处理通知的完整类。该类管理字符串形式的报告，此报告表示一个长期运行的进程。

尽管该示例使用声明为 static（在 Visual Basic 中为 Shared）的类，但并不是必须使用静态类。不过，删除缓存项时，用于处理回调的方法必须存在。例如，不应在 ASP.NET 页中实现回调处理程序，因为在从缓存中删除项之前该页可能已被释放，因此用于处理回调的方法将不可用。为了确保从缓存中删除项时处理回调的方法仍然存在，请使用该方法静态类。但是，静态类的缺点是需要保证所有静态方法都是线程安全的。

警告

请不要在页面中将 CacheItemRemovedCallback 设置为一个方法。除了在释放页面后回调无法使用页面方法以外，将回调指向页面方法还会阻碍垃圾回收将页面使用的内存回收。由于回调包含对页面的引用，而垃圾回收器不会从内存中移除包含任何引用的项，因此会出现这种情况。在加载应用程序期间，这可能会导致内存很快被用光。

- 该示例类包括以下功能：
- 私有成员，用于跟踪报告是否已从缓存中移除。
  - 名为 CacheReport 的方法，用于将项以 MyReport 的名称添加到缓存中，并将该项设置为在添加到缓存中后一分钟过期。该方法还会将 ReportRemovedCallback 方法传递给 onRemoveCallback 参数，从而注册 ReportRemoveCallback 方法，以便在从缓存中删除项时进行调用。
  - 名为 GetReport 的方法，用于从缓存中获取项。该方法确定名为 MyReport 的项是否存在于缓存中。如果该项不存在，则该方法将调用 CacheReport，将该项添加到缓存中。
  - 名为 ReportRemovedCallback 的方法，用于处理缓存项移除回调。ReportRemovedCallback 具有与 CacheItemRemovedCallback 委托相同的函数签名。该方法将变量 \_reportRemovedFromCache 设置为 true，然后通过 CacheReport 方法将项添加回缓存中。

```
C#
复制代码
using System;
using System.Web;
using System.Web.Caching;

public static class ReportManager
{
    private static bool _reportRemovedFromCache = false;
    static ReportManager()
    { }

    public static String GetReport()
    {
        lock (typeof(ReportManager))
        {
            if (HttpContext.Current.Cache["MyReport"] != null)
                return (string)HttpRuntime.Cache["MyReport"];
            else
            {
                CacheReport();
                return (string)HttpRuntime.Cache["MyReport"];
            }
        }
    }
}
```

```

    }

    public static void CacheReport ()
    {
        lock (typeof(ReportManager))
        {
            HttpContext.Current.Cache.Add("MyReport",
                CreateReport(), null, DateTime.MaxValue,
                new TimeSpan(0, 1, 0),
                System.Web.Caching.CacheItemPriority.Default,
                ReportRemovedCallback);
        }
    }

    private static string CreateReport ()
    {
        System.Text.StringBuilder myReport =
            new System.Text.StringBuilder();
        myReport.Append("Sales Report<br />");
        myReport.Append("2005 Q2 Figures<br />");
        myReport.Append("Sales NE Region - $2 million<br />");
        myReport.Append("Sales NW Region - $4.5 million<br />");
        myReport.Append("Report Generated: " + DateTime.Now.ToString()
            + "<br />");
        myReport.Append("Report Removed From Cache: " +
            _reportRemovedFromCache.ToString());
        return myReport.ToString();
    }

    public static void ReportRemovedCallback(String key, object value,
        CacheItemRemovedReason removedReason)
    {
        _reportRemovedFromCache = true;
        CacheReport();
    }
}

```

Visual Basic

复制代码

```

Imports System
Imports System.Web
Imports System.Web.Caching
Public Class ReportManager
    Private Shared _reportRemovedFromCache As Boolean = False
    Shared Sub New()
    End Sub

    Private Sub New()
    End Sub

    Public Shared Function GetReport() As String
        SyncLock (GetType(ReportManager))
            If HttpContext.Current.Cache("MyReport") IsNot Nothing Then
                Return CStr(HttpRuntime.Cache("MyReport"))
            Else
                CacheReport()
                Return CStr(HttpRuntime.Cache("MyReport"))
            End If
        End SyncLock
    End Function

    Public Shared Sub CacheReport ()
        SyncLock (GetType(ReportManager))

```

```
HttpContext.Current.Cache.Add("MyReport", CreateReport(), _
    Nothing, DateTime.MaxValue, New TimeSpan(0, 1, 0), _
    System.Web.Caching.CacheItemPriority.Default, _
    AddressOf ReportRemovedCallback)

End SyncLock

End Sub

Private Shared Function CreateReport() As String
    Dim myReport As New System.Text.StringBuilder()
    myReport.Append("Sales Report<br />")
    myReport.Append("2005 Q2 Figures<br />")
    myReport.Append("Sales NE Region - $2 million<br />")
    myReport.Append("Sales NW Region - $4.5 million<br />")
    myReport.Append("Report Generated: " & _
        DateTime.Now.ToString() & "<br />")
    myReport.Append("Report Removed From Cache: " & _
        & _reportRemovedFromCache.ToString())
    Return myReport.ToString()
End Function

Public Shared Sub ReportRemovedCallback(ByVal key As String, _
    ByVal value As Object, ByVal removedReason _
    As CacheItemRemovedReason)
    _reportRemovedFromCache = True
    CacheReport()
End Sub

End Class
```

如何：从 ASP.NET 缓存中删除项


ASP.NET 缓存中的数据是易失的，即不能永久保存。由于以下任一原因，缓存中的数据可能会自动移除：

- 缓存已满。
- 该项已过期。
- 依赖项发生更改。

有关更多信息，请参见 [ASP.NET 缓存概述](#)。

从缓存中移除项的具体方法由用于向缓存添加项的代码确定。有关更多信息，请参见[如何：将项添加到缓存中](#)。项从缓存中移除时会向您发出通知。有关更多信息，请参见[如何：从缓存中移除项时通知应用程序](#)。

除了允许从缓存中自动移除项之外，还可以显式移除项。

 注意

如果调用 [Insert](#) 方法，并向缓存中添加与现有项同名的项，则将从缓存中删除该旧项。

## 从缓存中显式删除项

- 调用 [Remove](#) 方法，以传递要移除的项的键。

下面的示例演示如何移除键为 MyData1 的项。

```
Visual Basic
复制代码
Cache.Remove("MyData1")

C#
复制代码
Cache.Remove("MyData1");
```

使用 SqlCacheDependency 类在 ASP.NET 中缓存

ASP.NET 允许您使用 [SqlCacheDependency](#) 类创建依赖于数据库中表或行的缓存项。当表中或特定行中发生更改时，带有依赖项的项便会失效，并会从缓存中移除。可以在 Microsoft SQL Server 7.0、SQL Server 2000 和 SQL Server 2005 中设置表的依赖项。如果您使用 SQL Server 2005，还可以设置特定记录的依赖项。

在某些方案中，使用带有 SQL 依赖项的缓存可显著提高应用程序的性能。例如，假定您正在构建一个从数据库显示产品信息的电子商务应用程序。如果不进行缓存，则每当用户要查看产品时，应用程序都必须从数据库请求数据。您可以在某一时刻将产品信息缓存一天，由于产品信息已经在内存中，因此可确保较快的响应时间，但是，当产品信息发生变化时，缓存的产品信息就会失去与数据库中数据的同步，且不同步的时间最长可达一天。

使用 SQL 缓存依赖项可以缓存产品信息，并创建一个数据库表或行更改的依赖项。当且仅当数据更改时，基于该数据的缓存项便会失效并会从缓存中移除。下次从缓存中请求该项时，如果该项不在缓存中，便可以再次向缓存中添加更新后的版本，并且可确保具有最新的数据。

SQL 缓存依赖项还可用于页输出缓存。例如，可以创建一个名为 ViewProduct.aspx 的页，用于显示有关特定产品的信息。可以将该页的缓存策略设置为 SQL 依赖项，就如为手动添加到缓存中的项所设置的依赖项一样。该页便会一直存储在缓存中，直至所依赖的表或行发生更改为止。当数据发生更改时，便会重新创建页，并将新创建的页再次存储在输出缓存中。

有关更多信息，请参见 [ASP.NET 缓存概述](#)。

## 功能

ASP.NET SQL 缓存依赖项提供以下功能：

- SQL 缓存依赖项可用于应用程序缓存和页输出缓存。
- 可在 SQL Server 7.0 及更高版本中使用 SQL 缓存依赖项。
- 可以在网络园（一台服务器上存在多个处理器）或网络场（多台服务器运行同一应用程序）中使用 SQL 缓存依赖项。
- 与 SQL 缓存依赖项关联的数据库操作比较简单，因此不会给服务器带来很高的处理成本。
- 在应用程序和 SQL Server 中配置 SQL 缓存依赖项不需要很精深的 SQL 知识。ASP.NET 中包括可以自动执行此配置的工具。另外，还可以使用 [SqlCacheDependencyAdmin](#) 类以编程方式配置 SQL 缓存依赖项。

## SQL Server 7.0 和 SQL Server 2000 实现

ASP.NET 为 SQL Server 7.0 和 SQL Server 2000 的缓存依赖项实现了一个轮询模型。ASP.NET 进程内的一个线程会以指定的时间间隔轮询 SQL Server 数据库，以确定数据是否已更改。如果数据已更改，缓存依赖项便会失效，并从缓存中移除。可以在 Web.config 文件中以声明方式指定应用程序中的轮询间隔，也可以使用 SqlCacheDependency 类以编程方式指定此间隔。

对于 SQL Server 7.0 和 SQL Server 2000，SQL 缓存依赖项仅限于表级别的数据更改。可以将 ASP.NET 配置为轮询数据库来确定表中的更改，但不能确定特定行中的更改。

启用 SQL 缓存

为了在 SQL Server 7.0 和 SQL Server 2000 中使用 SQL 缓存依赖项，必须先将 SQL Server 配置为支持缓存依赖项。ASP.NET 提供了一些实用工具，可用于配置 SQL Server 上的 SQL 缓存，其中包括一个名为 Aspnet\_regsql.exe 的工具和 SqlCacheDependencyAdmin 类。有关如何在 SQL Server 上启用 SQL 缓存依赖项的更多信息，请参见[如何：使用缓存键依赖项缓存页输出](#)。

## SQL Server 2005 实现

SQL Server 2005 为缓存依赖项实现的模型不同于 SQL Server 7.0 和 SQL Server 2000 中的缓存依赖项模型。在 SQL Server 2005 中，不需要执行任何特殊的配置步骤来启用 SQL 缓存依赖项。此外，SQL Server 2005 还实现了一种更改通知模型，可以向订阅了通知的应用程序服务器发送通知，而不是依赖早期版本的 SQL Server 中必需的轮询模型。

SQL Server 2005 缓存依赖项在接收通知的更改类型方面更具灵活性。SQL Server 2005 监控对特定 SQL 命令的结果集的更改。如果数据库中发生了将修改该命令的结果集的更改，依赖项便会使缓存的项失效。此功能使得 SQL Server 2005 可以提供行级别的通知。

对于测试更改的查询有一些要求。必须提供完全限定的表名，其中包括所有者名称（例如 dbo.authors）。总之，SQL 2005 通知支持 Select 查询和存储过程，支持多个查询和嵌套查询，但不支持聚合操作（例如 COUNT(\*)）。有关 SQL Server 2005 支持哪些查询以及通知规则的更多信息，请参见“SQL Books Online”（SQL 联机丛书）中的主题“Creating a Query for Notification”（创建通知查询）。

## 在 ASP.NET 应用程序中配置 SQL 缓存

在对 SQL Server 7.0 或 SQL Server 2000 进行了缓存依赖项配置后，或是在 SQL Server 2005 中创建了适当的命令依赖项后，就可以配置您的应用程序来使用 SQL 缓存依赖项，如同配置任何其他缓存依赖项一样。例如，可以在 Web.config 文件中创建一个缓存配置文件，然后在应使用 SQL 缓存依赖项的每个页中引用该缓存配置文件。还可以在通过 SqlCacheDependency 类以编程方式启用 SQL 缓存依赖项后再使用它。有关更多信息，请参见[如何：使用缓存键依赖项缓存页输出](#)。

缓存 ASP.NET 页

ASP.NET 使您可以缓存 ASP.NET 页所生成的部分响应或所有响应，在 ASP.NET 中将这种技术称为输出缓存。可以在发出请求的浏览器、响应请求的 Web 服务器以及请求或响应流中任何其他具有缓存功能的设备（如代理服务器）上缓存页。缓存为您提供了一个强有力的方式来提高 Web 应用程序的性能。缓存功能允许利用缓存满足对页的后续请求，这样就不需要再次运行最初创建该页的代码。对站点中访问最频繁的页进行缓存可以充分地提高 Web 服务器的吞吐量（通常以每秒的请求数计算）。

可以在页或配置文件中以声明方式或者通过编程方式使用缓存 API 指定缓存设置。有关更多信息，请参见[设置页的可缓存性](#)。

可以根据查询字符串参数值或窗体变量值（控件值）缓存页。必须通过使用 `@ OutputCache` 指令的 `VarByParam` 属性，显式启用基于这些类型的值的缓存。有关更多信息，请参见[缓存页的多个版本](#)。

当用户请求某一缓存页时，ASP.NET 根据已经为该页定义的缓存策略确定其缓存输出是否仍有效。如果该输出有效，则将该缓存输出发送到客户端，并且不重新处理该页。ASP.NET 允许您在此验证检查期间运行代码，以便可以编写用于检查页是否有效的自定义逻辑。有关更多信息，请参见[如何：检查缓存页的有效性](#)。


有时，缓存整个页是不切实际的，因为在每次请求时可能需要更改页的某些部分。在这些情况下，可以缓存页的一部分。ASP.NET 提供了只缓存 ASP.NET 页的几部分的功能。有关更多信息，请参见[缓存 ASP.NET 页的某些部分](#)。

如何：以声明方式设置 ASP.NET 页的可缓存性

某页或用户控件的可缓存性指某页能否在其响应生命周期内缓存到某个设备上。这些设备包括发出请求的客户端（浏览器），响应请求的 Web 服务器，以及请求或响应流中任何具有缓存功能的设备（例如代理服务器）。

如果您在设计时知道某页需要什么样的可缓存性设置，您可以以声明方式设置可缓存性。该页将为所有请求使用相同的可缓存性设置。有关更多信息，请参见[设置页的可缓存性](#)。

## 以声明方式设置页的可缓存性

- 在页中包含 `@ OutputCache` 指令，并定义 `Duration` 和 `VarByParam` 属性。
  - 在 `@ OutputCache` 指令中包含 `Location` 属性，并将其值定义为 `OutputCacheLocation` 枚举中的下列值之一：[Any](#)、[Client](#)、[Downstream](#)、[Server](#)、[ServerAndClient](#) 或 [None](#)。
- 下面的代码演示如何将页的可缓存性设置为 60 秒：
- ```
<%@ OutputCache Duration="60" VarByParam="None"%>
```
- 注意

默认设置为 Any。如果未定义 Location 属性，则可以将页输出缓存在与响应有关的所有具有缓存功能的网络设备上。其中包括请求客户端、原服务器、以及响应通过的任何代理服务器。

## 使用缓存配置文件以声明方式设置页的可缓存性

- 在应用程序的 Web.config 文件中定义缓存配置文件，在配置文件中包括 `duration` 和 `varyByParam` 设置。
- 下面的 `< caching>` 配置元素定义名为 `Cache30Seconds` 的缓存配置文件，它将在服务器上将页缓存 30 秒之久。
- ```
<caching>
  <outputCacheSettings>
    <outputCacheProfiles>
      <add name="Cache30Seconds" duration="30"
        varyByParam="none" />
    </outputCacheProfiles>
  </outputCacheSettings>
</caching>
```
- 在使用配置文件的每个 ASP.NET 页中包含 `@ OutputCache` 指令，并将 `CacheProfile` 属性设置为 Web.config 文件中定义的缓存配置文件的名称。
- 下面的代码指定页应当使用名为 `Cache30Seconds` 的缓存配置文件：
- ```
<%@ OutputCache CacheProfile="Cache30Seconds" %>
```

如何：以编程方式设置页的可缓存性

页或用户控件的可缓存性指的是某一页是否能在该页的响应生命周期内缓存在某个设备上。缓存页的这些设备包括发出请求的浏览器，响应请求的 Web 服务器，以及请求或响应流中任何可执行缓存的设备，如代理服务器。

如果应用程序将根据运行时条件（如读取请求标头）确定可缓存性，您可以通过编程方式设置可缓存性。有关更多信息，请参见[设置页的可缓存性](#)。

## 以编程方式设置页的可缓存性

- 在页的代码中，调用 `Response` 对象的 `Cache` 属性的 `SetCacheability` 方法。
- 下面的代码将 Cache-Control HTTP 标头设置为 `Public`。

C#

[复制代码](#)

```
Response.Cache.SetCacheability(HttpCacheability.Public);
```

```
Response.Cache.SetCacheability(HttpCacheability.Public)
```

如果将 `NoCache` 或 `ServerAndNoCache` 传递到 `SetCacheability` 方法以防止请求的浏览器在它自己的历史记录文件夹中缓存某一页，那么任何时候当某个用户单击“后退”或“前进”按钮时，都会请求响应的新版本。通过调用 `Cache` 属性的 `SetAllowResponseInBrowserHistory` 方法，并且为 `allow` 参数传递 `true` 值，您可以按条件重写此行为。

如果将可缓存性设置为除 `NoCache` 或 `ServerAndNoCache` 之外的任何值，ASP.NET 将忽略由 `SetAllowResponseInBrowserHistory` 方法设置的值。

如何：设置 ASP.NET 页缓存的过期时间值

若要导致某一页添加到输出缓存中，需要为该页建立到期策略。这可以通过以声明方式或编程方式来实现。

## 以声明方式为页设置输出缓存到期时间

- 将 `@ OutputCache` 指令包括在您要缓存其响应的 ASP.NET 页（.aspx 文件）中。将 `Duration` 属性设置为一个正数值，将 `VaryByParam` 属性设置为一个值。

注意

默认情况下，`@ OutputCache` 指令将 `Cache-Control` 标头设置为 `Any`。

例如，下面的 `@ OutputCache` 指令将页的到期时间设置为 60 秒：

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

注意

在使用 `@ OutputCache` 指令时，必须包括一个 `VaryByParam` 属性，否则将出现分析器错误。如果不希望使用 `VaryByParam` 属性提供的功能，请将其值设置为“None”。有关更多信息，请参见[缓存页的多个版本](#)。

## 以编程方式为页设置输出缓存到期时间

- 在该页的代码中，在 `Response` 对象的 `Cache` 属性中设置该页的到期策略。

注意

如果以编程方式设置页的到期时间，则您还必须为缓存的页设置 `Cache-Control` 标头。为此，请调用 `SetCacheability` 方法并向其传递 `HttpCacheability` 枚举值 `Public`。

下面的代码示例设置与前面过程中的 `@ OutputCache` 指令相同的缓存策略。

C#

[复制代码](#)

```
Response.Cache.SetExpires(DateTime.Now.AddSeconds(60));
Response.Cache.SetCacheability(HttpCacheability.Public);
Response.Cache.SetValidUntilExpires(true);
```

Visual Basic

[复制代码](#)

```
Response.Cache.SetExpires(DateTime.Now.AddSeconds(60))
Response.Cache.SetCacheability(HttpCacheability.Public)
Response.Cache.SetValidUntilExpires(True)
```

当缓存页到期时，以后对该页的请求将导致动态生成的响应。会在指定的持续时间内缓存该响应页。

如何：检查缓存页的有效性

用户请求缓存页时，ASP.NET 根据您在该页中定义的缓存策略来确定缓存输出是否仍然有效。如果缓存输出有效，则将输出发送到客户端，并且不重新处理该页。但是，ASP.NET 提供了使用验证回调在该验证检查期间运行代码的功能，因此您可以编写自定义逻辑来检查该页是否有效。利用验证回调，可以使在使用缓存依赖项的正常进程之外的缓存页无效。

## 以编程方式检查缓存页的有效性



1. 定义 [HttpCacheValidateHandler](#) 类型的事件处理程序，并包括检查缓存页响应的有效性的代码。

验证处理程序必须返回下列 [HttpValidationStatus](#) 值之一：

- [Invalid](#) 指示缓存页无效，将从缓存中移除该页，并且该请求将被作为缓存未命中处理。
- [IgnoreThisRequest](#) 导致将请求视为缓存未命中处理。因此，将重新处理该页，但不会使缓存页无效。
- [Valid](#) 指示缓存页有效。

下面的代码示例阐释名为 `ValidateCacheOutput` 的验证处理程序，该处理程序确定查询字符串变量 `status` 包含值 “invalid” 还是 “ignore”。如果状态值为 “invalid”，则该方法返回 `Invalid`，并且使该页在缓存中无效。如果状态值为 “ignore”，则该方法返回 `IgnoreThisRequest`，并且该页仍保留在缓存中，但为该请求生成一个新响应。

C#

[复制代码](#)

```
public static void ValidateCacheOutput(HttpContext context, Object data,
    ref HttpValidationStatus status)
{
    if (context.Request.QueryString["Status"] != null)
    {
        string pageStatus = context.Request.QueryString["Status"];

        if (pageStatus == "invalid")
            status = HttpValidationStatus.Invalid;
        else if (pageStatus == "ignore")
            status = HttpValidationStatus.IgnoreThisRequest;
        else
            status = HttpValidationStatus.Valid;
    }
    else
        status = HttpValidationStatus.Valid;
}
```

Visual Basic

[复制代码](#)

```
Public Shared Sub ValidatePage(ByVal context As HttpContext, _
    ByVal data As [Object], ByRef status As HttpValidationStatus)
    If Not (context.Request.QueryString("Status") Is Nothing) Then
        Dim pageStatus As String = context.Request.QueryString("Status")

        If pageStatus = "invalid" Then
            status = HttpValidationStatus.Invalid
        ElseIf pageStatus = "ignore" Then
            status = HttpValidationStatus.IgnoreThisRequest
        Else
            status = HttpValidationStatus.Valid
        End If
    Else
        status = HttpValidationStatus.Valid
    End If
End Sub
```

2. 从其中一个页生命周期事件（如页的 [Load](#) 事件）中调用 [AddValidationCallback](#) 方法，将您在步骤 1 中定义的事件处理程序作为第一个参数传递。

下面的代码示例将 `ValidateCacheOutput` 方法设置为验证处理程序。

C#

[复制代码](#)

```
protected void Page_Load(object sender, EventArgs e)
{
    Response.Cache.AddValidationCallback(
        new HttpCacheValidateHandler(ValidateCacheOutput),
        null);
}
```

Visual Basic

[复制代码](#)

```
Protected Sub Page_Load(ByVal sender As Object, _  
    ByVal e As System.EventArgs) Handles Me.Load  
  
    Response.Cache.AddValidationCallback( _  
        New HttpCacheValidateHandler(AddressOf ValidatePage), Nothing)  
  
End Sub
```

缓存 ASP.NET 页的某些部分

有时缓存整个页是不现实的，因为页的某些部分可能在每次请求时都需要更改。在这些情况下，只能缓存页的一部分。执行此操作有两个选项：控件缓存和缓存后替换。

在控件缓存（也称为片段缓存）中，可以通过创建用户控件来包含缓存的内容，然后将用户控件标记为可缓存来缓存部分页输出。该选项允许缓存页中的特定内容，而在每次都重新创建整个页。例如，如果创建的页显示大量动态内容（如股票信息），但也有某些部分是静态的（如每周摘要），则可以在用户控件中创建这些静态部分并将用户控件配置为缓存。

缓存后替换与控件缓存正好相反。它对页进行缓存，但是页中的某些片段是动态的，因此不会缓存这些片段。例如，如果创建的页在设定的时间段内完全是静态的（例如新闻报道页），可以设置为缓存整个页。如果为缓存的页添加旋转广告横幅，则在页请求之间，广告横幅不会变化。然而，使用缓存后替换，可以对页进行缓存，但可以将特定部分标记为不可缓存。在本例中，将广告横幅标记为不可缓存。它们将在每次页请求时动态创建，并添加到缓存的页输出中。有关缓存后替换的更多信息，请参阅[动态更新缓存页的部分](#)。

## 控件缓存

通过创建用户控件来缓存内容，可以将页上需要花费宝贵的处理器时间来创建的某些部分（例如数据库查询）与页的其他部分分离开。只需占用很少服务器资源的部分可以在每次请求时动态生成。

在标识了要缓存的页的部分，并创建了用以包含这些部分中的每个部分的用户控件后，您必须确定这些用户控件的缓存策略。您可以使用 [@ OutputCache](#) 指令，或者在代码中使用 [PartialCachingAttribute](#) 类，以声明的方式为用户控件设置这些策略。


例如，如果在用户控件文件（.ascx 文件）的顶部包括下面的指令，则该控件的一个版本将在输出缓存中存储 120 秒。

```
<%@ OutputCache Duration="120" VaryByParam="None" %>
```

若要在代码中设置缓存参数，可以在用户控件的类声明中使用一个属性。例如，如果在类声明的元数据中包括下面的属性，则该内容的一个版本将在输出缓存中存储 120 秒：

```
C#  
复制代码  
[PartialCaching(120)]  
public partial class CachedControl : System.Web.UI.UserControl  
{  
    // Class Code  
}  
  
Visual Basic  
复制代码  
<PartialCaching(120)> _  
Partial Class CachedControl  
    Inherits System.Web.UI.UserControl  
    ' Class Code  
End Class
```

有关可在页输出中设置的属性的更多信息，请参阅 [@ OutputCache](#) 主题。有关如何开发用户控件的更多信息，请参见 [ASP.NET Web 服务器控件概述](#)。

 注意

由于可在页上嵌套用户控件，您还可以嵌套已放置到输出缓存中的用户控件。可以为页和嵌套的用户控件指定不同的缓存设置。

以编程方式引用缓存的用户控件

在以声明的方式创建可缓存的用户控件时，可以包括一个 [ID](#) 属性，以便以编程方式引用该用户控件实例。但是，在代码中引用用户控件之前，必须验证在输出缓存中是否存在该用户控件。缓存的用户控件只在首次请求时动态生成；在指定的时间到期之前，从输出缓存满足所有的后续请求。确定用户控件已实例化后，可以从包含页以编程方式操作该用户控件。例如，如果通过声明方式将 SampleUserControl 的 ID 分配给用户控件，则可以使用下面的代码检查它是否存在。

```
C#  
复制代码  
protected void Page_Load(object sender, EventArgs e)  
{  
    if (SampleUserControl != null)
```

```
// Place code manipulating SampleUserControl here.
}
Visual Basic
复制代码
Protected Sub Page_Load(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Me.Load
    If SampleUserControl <> Nothing Then
        ' Place code manipulating SampleUserControl here.
    End If
End Sub
```

## 以不同的持续时间缓存页和用户控件

可以为页和页上的用户控件设置不同的输出缓存持续时间值。如果页的输出缓存持续时间长于用户控件的输出缓存持续时间，则页的输出缓存持续时间优先。例如，如果页的输出缓存设置为 100 秒，而用户控件的输出缓存设置为 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。

下面的代码示例演示了当页的缓存持续时间长于用户控件的缓存持续时间时的效果。该页配置为缓存 100 秒。

```
C#
复制代码
<%@ Page language="C#" %>
<%@ Register tagprefix="SampleControl" tagname="Time" src="uc01.ascx" %>
<%@ OutputCache duration="100" varybyparam="none" %>

<SampleControl:Time runat="server" /><br /> <br /> <br />

This page was most recently generated at:<p>

<% DateTime t = DateTime.Now.ToString();
    Response.Write(t); %>
Visual Basic
复制代码
<%@ Page language="VB" %>
<%@ Register tagprefix="SampleControl" tagname="Time" src="uc01.ascx" %>
<%@ OutputCache duration="100" varybyparam="none" %>

<SampleControl:Time runat="server" /><br /> <br /> <br />

This page was most recently generated at:<p>
<% Dim t As DateTime = DateTime.Now.ToString()
Response.Write(t) %>
```

下面的代码示例演示了包括在页中的用户控件。控件的缓存持续时间设置为 50 秒。

```
C#
复制代码
<% @Control language="C#" %>
<% @OutputCache duration="50" varybyparam="none" %>

This user control was most recently generated at:<p>
<% DateTime t = DateTime.Now.ToString();
    Response.Write(t); %>
Visual Basic
复制代码
<% @Control language="VB" %>
<% @OutputCache duration="50" varybyparam="none" %>

This user control was most recently generated at:<p>
<% Dim t As DateTime = DateTime.Now.ToString()
Response.Write(t) %>
```

不过，如果页的输出缓存持续时间比用户控件的输出缓存持续时间短，则即使已为某个请求重新生成该页的其余部分，也将一直缓存用户控件直到其持续时间到期为止。例如，如果页的输出缓存设置为 50 秒，而用户控件的输出缓存设置为 100 秒，则页的其余部分每到期两次，用户控件才到期一次。

下面的代码演示了一个页的标记，该页中包含的用户控件的缓存持续时间长于该页的缓存持续时间。该页配置为缓存 50 秒。

C#

[复制代码](#)

```
<%@ Page language="C#" %>
<%@ Register tagprefix="SampleControl" tagname="Time" src="uc2.ascx" %>
<%@ OutputCache duration="50" varybyparam="none" %>

<SampleControl:Time runat="server" /><br /> <br /> <br />
```

This page was most recently generated at:<p>

```
<% DateTime t = DateTime.Now.ToString();
    Response.Write(t); %>
```

Visual Basic

[复制代码](#)

```
<%@ Page language="VB" %>
<%@ Register tagprefix="SampleControl" tagname="Time" src="Uc2.ascx" %>
<%@ OutputCache duration="50" varybyparam="none" %>

<SampleControl:Time runat="server" /><br /> <br /> <br />
```

This page was most recently generated at:<p>

```
<% Dim t As DateTime = DateTime.Now.ToString()
Response.Write(t) %>
```

下面的代码演示了包括在页中的用户控件。控件的缓存持续时间设置为 100 秒。

C#

[复制代码](#)

```
<% @Control language="C#" %>
<% @OutputCache duration="100" varybyparam="none" %>
```

This user control was most recently generated at:<p>

```
<% DateTime t = DateTime.Now.ToString();
    Response.Write(t); %>
```

Visual Basic

[复制代码](#)

```
<% @Control language="VB" %>
<% @OutputCache duration="100" varybyparam="none" %>
```

This user control was most recently generated at:<p>

```
<% Dim t As DateTime = DateTime.Now.ToString()
Response.Write(t) %>
```

## 缓存页的多个版本

有时，您可能希望缓存某页，但是会基于请求为该页创建不同的版本。例如，根据查询字符串中传递的值，该页可能具有不同的输出。

ASP.NET 允许在输出缓存中缓存同一页的多个版本。输出缓存可能会因下列因素而异：

- 初始请求（HTTP GET）中的查询字符串。
- 回发时传递的控制值（HTTP POST 值）。
- 随请求传递的 HTTP 标头。
- 发出请求的浏览器的主版本号。
- 该页中的自定义字符串。在这种情况下，可以在 Global.asax 文件中创建自定义代码以指定该页的缓存行为。

可以通过以下两种方法来缓存页输出的多个版本：使用 [@ OutputCache](#) 指令的属性以声明方式，或者使用 [HttpCachePolicy](#) 类的属性和方法以编程方式。

[@ OutputCache](#) 指令包括四个可用来缓存页输出的多个版本的属性：

- [VaryByParam](#) 属性可用来使缓存输出因查询字符串而异。

- [VaryByControl](#) 属性可用来使缓存输出因控制值而异。
- [VaryByHeader](#) 属性可用来使缓存输出因请求的 HTTP 标头而异。
- [VaryByCustom](#) 属性可用来使缓存输出因浏览器类型或您定义的自定义字符串而异。

📌注意

您必须在任何 `@OutputCache` 指令中包括 `VaryByParam` 属性或 `VaryByControl` 属性。但是，如果您不需要使缓存输出因控制值或参数而异，则可以定义值为 `None` 的 `VaryByParam`。

`HttpCachePolicy` 类提供两个属性和一个方法，您可以通过它们以编程方式指定与以声明方式所能设置的缓存配置相同的缓存配置。使用 [VaryByParams](#) 和 [VaryByHeaders](#) 属性可以分别指定查询字符串参数和标头名称作为缓存策略改变依据。使用 [SetVaryByCustom](#) 方法可以定义要作为输出缓存改变依据的自定义字符串。

如何：使用缓存键依赖项缓存页输出

有时，当缓存中的某一项被移除时，您可能需要从输出缓存中移除一页。例如，您可能使用一页来显示放置在应用程序缓存中并供多个页使用的占用大量进程的报告。当该报告已更改或已从缓存中移除时，您希望将页输出也从缓存中移除，因为该报告不再有效。为此，可使缓存的页输出依赖于其他缓存项。

📌注意

通过调用 [RemoveOutputCacheItem](#) 方法，可显式移除输出缓存中的任何页。可以从 `Global.asax` 文件、自定义 ASP.NET 服务器控件或页中执行此操作，具体取决于应用程序的需要。

## 使缓存的页输出依赖于另一缓存项

1. 在页中，以声明方式或编程方式指定缓存设置。有关更多信息，请参见[如何：设置 ASP.NET 页缓存的过期时间值、设置页的可缓存性和缓存页的多个版本](#)。
2. 在页代码中调用 [AddCacheItemDependency](#) 方法。将创建依赖项的缓存项的名称作为 `cacheKey` 参数传递。

下面的代码示例演示如何在名为 `ProcessIntensiveReport` 的项上设置依赖项。当此项被修改或移除时，将从缓存中移除页输出。

C#

[复制代码](#)

```
protected void Page_Load(object sender, EventArgs e)
{
    Response.AddCacheItemDependency("ProcessIntensiveReport");

    // Set additional properties to enable caching.
    Response.Cache.SetExpires(DateTime.Now.AddSeconds(60));
    Response.Cache.SetCacheability(HttpCacheability.Public);
    Response.Cache.SetValidUntilExpires(true);
}
```

Visual Basic

[复制代码](#)

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
    Response.AddCacheItemDependency("ProcessIntensiveReport")

    ' Set additional properties to enable caching.
    Response.Cache.SetExpires(DateTime.Now.AddSeconds(60))
    Response.Cache.SetCacheability(HttpCacheability.Public)
    Response.Cache.SetValidUntilExpires(True)
End Sub
```


注意

不能在 ASP.NET 用户控件中调用 `AddCacheItemDependency` 方法。不过，在指定 `@OutputCache` 指令的任何用户控件中，都可以创建描述缓存键依赖项的 [CacheDependency](#) 对象，并将其分配给 [UserControl](#) 对象的 [Dependency](#) 属性。

3.

如何：使用文件依赖项缓存页输出

有时候，您可能需要在文件发生更改时从输出缓存中移除某一页。例如，您可能有这样一页：该页从生成 XML 文件作为输出且占用大量进程的报告中获取其内容。仅当 XML 文件发生更改时，才需要重新处理该页。要将重新处理限制为仅在需要的时候进行，可以使页的缓存策略依赖于单个文件。如有必要，可以使缓存页依赖于多个文件。

注意

通过调用 [RemoveOutputCacheItem](#) 方法，可显式移除输出缓存中的任何页。可以从 Global.asax 文件、自定义 ASP.NET 服务器控件或页中执行此操作，具体取决于应用程序的需要。

## 使缓存页输出依赖于一个文件

1. 以声明方式或编程方式指定缓存页输出的设置。有关更多信息，请参见[如何：设置 ASP.NET 页缓存的过期时间值](#)、[设置页的可缓存性](#)和[缓存页的多个版本](#)。
2. 在页代码中调用 [AddFileDependency](#) 方法。将创建依赖项的文件的路径作为方法的 filename 参数传递。

下面的代码示例在 TextFile1.txt 文件上设置一个文件依赖项。当文件发生更改时，将从缓存中移除页输出。

C#

[复制代码](#)

```
protected void Page_Load(object sender, EventArgs e)
{
    string fileDependencyPath = Server.MapPath("TextFile1.txt");
    Response.AddFileDependency(fileDependencyPath);

    // Set additional properties to enable caching.
    Response.Cache.SetExpires(DateTime.Now.AddSeconds(60));
    Response.Cache.SetCacheability(HttpCacheability.Public);
    Response.Cache.SetValidUntilExpires(true);
}
```

Visual Basic

[复制代码](#)

```
Protected Sub Page_Load(ByVal sender As Object, _
    ByVal e As EventArgs) Handles Me.Load
    Dim fileDependencyPath As String = _
        Server.MapPath("TextFile1.txt")
    Response.AddFileDependency(fileDependencyPath)

    ' Set additional properties to enable caching.
    Response.Cache.SetExpires(DateTime.Now.AddSeconds(60))
    Response.Cache.SetCacheability(HttpCacheability.Public)
    Response.Cache.SetValidUntilExpires(True)
End Sub
```

注意

不能从 ASP.NET 用户控件使用这些方法。但是，在指定 [@ OutputCache](#) 指令的任何用户控件中，您都可以创建一个文件依赖项，并将其分配给 [Dependency](#) 属性。

## 使缓存页输出依赖于文件组

1. 以声明方式或编程方式指定缓存页输出的设置。有关更多信息，请参见[如何：设置 ASP.NET 页缓存的过期时间值](#)、[设置页的可缓存性](#)和[缓存页的多个版本](#)。
2. 在页代码中，创建一个包含该页所要依赖的文件的路径的 [String](#) 数组或 [ArrayList](#)。
3. 调用 [AddFileDependencies](#) 方法，并将数组作为 filenames 参数传递。

下面的代码示例创建包含 TextFile1.txt 和 XMLFile1.xml 文件的文件路径的字符串数组，并使页输出依赖于这两个文件。如果修改了其中任何一个文件，将从缓存中移除页输出。

C#

[复制代码](#)

```
protected void Page_Load(object sender, EventArgs e)
{
    string[] fileDependencies;
```



```

string fileDependency1 = Server.MapPath("TextFile1.txt");
string fileDependency2 = Server.MapPath("XMLFile1.xml");
fileDependencies = new String[] { fileDependency1,
    fileDependency2 };
Response.AddFileDependencies(fileDependencies);
}

```

Visual Basic

[复制代码](#)

```

Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load

    Dim fileDependencies() As String

    Dim fileDependency1 As String = Server.MapPath("TextFile1.txt")

    Dim fileDependency2 As String = Server.MapPath("XMLFile1.xml")

    fileDependencies = New String() {fileDependency1, _
        fileDependency2}

    Response.AddFileDependencies(fileDependencies)

End Sub

```

CacheItemRemovedCallback 委托

定义在从 [Cache](#) 移除缓存项时通知应用程序的回调方法。

命名空间: System.Web.Caching

程序集: System.Web (在 system.web.dll 中)

[语法](#)

Visual Basic (声明)

```

Public Delegate Sub CacheItemRemovedCallback ( _
    key As String, _
    value As Object, _
    reason As CacheItemRemovedReason _
)

```

Visual Basic (用法)

```
Dim instance As New CacheItemRemovedCallback(AddressOf HandlerMethod)
```

C#

```

public delegate void CacheItemRemovedCallback (
    string key,
    Object value,
    CacheItemRemovedReason reason
)

```

C++

```

public delegate void CacheItemRemovedCallback (
    String^ key,
    Object^ value,
    CacheItemRemovedReason reason
)

```

J#

```
/** @delegate */
```

```

public delegate void CacheItemRemovedCallback (
    String key,
    Object value,
    CacheItemRemovedReason reason
)

```

JScript

JScript 支持使用委托，但不支持进行新的声明。

参数

key

从缓存中移除的键。

value

与从缓存中移除的键关联的 [Object](#) 项。

reason

[CacheItemRemovedReason](#) 枚举指定的、从缓存移除项的原因。

#### 示例

下面的代码示例演示一个页面，该页面为用户显示缓存中某项的赋值，当该项从缓存中移除时将通知用户。它创建 `RemovedCallback` 方法，该方法使用 `CacheItemRemovedCallback` 委托的签名，以在缓存项被移除时通知用户并且使用 `CacheItemRemovedReason` 枚举告诉用户该项被移除的原因。此外，它使用 `Cache.Item` 属性将对象添加到缓存中并检索这些对象的值。在 `AddItemToCache` 方法中，它使用 `Cache.Add` 方法向缓存中添加项。若要使用 `CacheItemRemovedCallback` 委托，您必须使用此方法或者 `Cache.Insert` 方法向缓存中添加项，以便该项被移除时 ASP.NET 能自动调用正确的方法。自定义的 `RemoveItemFromCache` 方法使用 `Cache.Remove` 方法显式地从缓存中删除该项，这导致调用 `RemovedCallback` 方法。

Visual Basic

#### [复制代码](#)

```
<%@ Page Language="VB" %>

<html>
<Script runat=server>
    Shared itemRemoved As boolean = false
    Shared reason As CacheItemRemovedReason
    Dim onRemove As CacheItemRemovedCallback

    Public Sub RemovedCallback(k As String, v As Object, r As CacheItemRemovedReason)
        itemRemoved = true
        reason = r
    End Sub

    Public Sub AddItemToCache(sender As Object, e As EventArgs)
        itemRemoved = false

        onRemove = New CacheItemRemovedCallback(AddressOf Me.RemovedCallback)

        If (IsNothing(Cache("Key1"))) Then
            Cache.Add("Key1", "Value 1", Nothing, DateTime.Now.AddSeconds(60), TimeSpan.Zero, CacheItemPriority.High,
onRemove)
        End If
    End Sub

    Public Sub RemoveItemFromCache(sender As Object, e As EventArgs)
        If (Not IsNothing(Cache("Key1"))) Then
            Cache.Remove("Key1")
        End If
    End Sub
</Script>

<body>
<Form runat="server">
    <input type=submit OnServerClick="AddItemToCache" value="Add Item To Cache" runat="server"/>
    <input type=submit OnServerClick="RemoveItemFromCache" value="Remove Item From Cache" runat="server"/>
</Form>
<%
If (itemRemoved) Then
    Response.Write("RemovedCallback event raised.")
    Response.Write("<BR>")
    Response.Write("Reason: <B>" + reason.ToString() + "</B>")
Else
    Response.Write("Value of cache key: <B>" + Server.HtmlEncode(CType(Cache("Key1"),String)) + "</B>")
End If
%>
</body>
</html>

C#
复制代码
<html>
<Script runat=server language="C#">
```

```

static bool itemRemoved = false;
static CacheItemRemovedReason reason;
CacheItemRemovedCallback onRemove = null;

public void RemovedCallback(String k, Object v, CacheItemRemovedReason r) {
    itemRemoved = true;
    reason = r;
}

public void AddItemToCache(Object sender, EventArgs e) {
    itemRemoved = false;

    onRemove = new CacheItemRemovedCallback(this.RemovedCallback);

    if (Cache["Key1"] == null)
        Cache.Add("Key1", "Value 1", null, DateTime.Now.AddSeconds(60), TimeSpan.Zero, CacheItemPriority.High,
onRemove);
}

public void RemoveItemFromCache(Object sender, EventArgs e) {
    if(Cache["Key1"] != null)
        Cache.Remove("Key1");
}
</Script>
<body>
<Form runat="server">
<input type="submit" OnServerClick="AddItemToCache" value="Add Item To Cache" runat="server"/>
<input type="submit" OnServerClick="RemoveItemFromCache" value="Remove Item From Cache" runat="server"/>
</Form>
<% if (itemRemoved) {
    Response.Write("RemovedCallback event raised.");
    Response.Write("<BR>");
    Response.Write("Reason: <B>" + reason.ToString() + "</B>");
}
else {
    Response.Write("Value of cache key: <B>" + Server.HtmlEncode(Cache["Key1"] as string) + "</B>");
}
%>
</body>
</html>
JScript
复制代码
<html>
<Script runat=server language="JScript">

```

```

static var itemRemoved : boolean = false;
static var reason : CacheItemRemovedReason;
var onRemove : CacheItemRemovedCallback = null;

public function RemovedCallback(k : String, v : Object, r : CacheItemRemovedReason) {
    itemRemoved = true;
    reason = r;
}

public function AddItemToCache(sender : Object, e : EventArgs) {
    itemRemoved = false;

    onRemove = this.RemovedCallback;

    if (Cache["Key1"] == null)
        Cache.Add("Key1", "Value 1", null, DateTime.Now.AddSeconds(10), TimeSpan.Zero, CacheItemPriority.High,
onRemove);
}

```

```
    }

    public function RemoveItemFromCache(sender : Object, e : EventArgs) {
        if(Cache["Key1"] != null)
            Cache.Remove("Key1");
    }
</Script>

<body>
    <Form runat="server">
        <input type=submit OnServerClick="AddItemToCache" value="Add Item To Cache" runat="server"/>
        <input type=submit OnServerClick="RemoveItemFromCache" value="Remove Item From Cache" runat="server"/>
    </Form>
<%
if (itemRemoved) {
    Response.Write("RemovedCallback event raised.");
    Response.Write("<BR>");
    Response.Write("Reason: <B>" + reason + "</B>");
}
else {
    Response.Write("Value of cache key: <B>" + Server.HtmlEncode(Cache["Key1"].ToString()) + "</B>");
}
%>

</body>
</html>
```

CacheItemRemovedReason 枚举

指定从 [Cache](#) 对象移除项的原因。

命名空间: System.Web.Caching

程序集: System.Web (在 system.web.dll 中)

[语法](#)

Visual Basic (声明)

```
Public Enumeration CacheItemRemovedReason
```

Visual Basic (用法)

```
Dim instance As CacheItemRemovedReason
```

C#

```
public enum CacheItemRemovedReason
```

C++

```
public enum class CacheItemRemovedReason
```

J#

```
public enum CacheItemRemovedReason
```

JScript

```
public enum CacheItemRemovedReason
```

[成员](#)

成员名称	说明
DependencyChanged	从缓存移除该项的原因是与之关联的缓存依赖项已更改。
Expired	从缓存移除该项的原因是它已过期。
Removed	该项是通过指定相同键的 <a href="#">Insert</a> 方法调用或 <a href="#">Remove</a> 方法调用从缓存中移除的。
Underused	之所以从缓存中移除该项，是因为系统要通过移除该项来释放内存。

[备注](#)

[CacheItemRemovedCallback](#) 委托使用此枚举来通知 ASP.NET 应用程序从 [Cache](#) 移除对象的时间和原因。

Topic	Location
<a href="#">How to: Notify an Application When an Item Is Removed from the Cache</a>	<a href="#">Building ASP .NET Web Applications</a>
<a href="#">How to: Notify an Application When an Item Is Removed from the Cache</a>	<a href="#">Building ASP .NET Web Applications</a>
<a href="#">HOW TO: 當項目從快取移除時告知應用程式</a>	<a href="#">建置 ASP .NET Web 應用程式</a>

[示例](#)

下面的代码示例演示共享 Boolean 属性 itemRemoved、共享 CacheItemRemovedReason 枚举对象 reason 和 CacheItemRemovedCallback 委托 onRemove。后者可用在 [Insert](#) 或 [Add](#) 方法调用中。它还定义了方法 RemovedCallback，该方法的签名与 CacheItemRemovedCallback 委托匹配。当 RemovedCallback 方法被调用时，它将 itemRemoved 属性值更改为 true，并将 reason 属性值赋给由 CacheItemRemovedReason 枚举提供的原因。

Visual Basic

[复制代码](#)

```
Shared itemRemoved As boolean = false
Shared reason As CacheItemRemovedReason
Dim onRemove As CacheItemRemovedCallback

Public Sub RemovedCallback(k As String, v As Object, r As CacheItemRemovedReason)
    itemRemoved = true
    reason = r
End Sub
```

C#

[复制代码](#)

```
static bool itemRemoved = false;
static CacheItemRemovedReason reason;
CacheItemRemovedCallback onRemove = null;

public void RemovedCallback(String k, Object v, CacheItemRemovedReason r) {
    itemRemoved = true;
    reason = r;
}
```

JScript

[复制代码](#)

```
static var itemRemoved : boolean = false;
static var reason : CacheItemRemovedReason;
var onRemove : CacheItemRemovedCallback = null;

public function RemovedCallback(k : String, v : Object, r : CacheItemRemovedReason) {
    itemRemoved = true;
    reason = r;
}
```

CacheDependency 类

在存储于 ASP.NET 应用程序的 [Cache](#) 对象中的项与文件、缓存键、文件或缓存键的数组或另一个 CacheDependency 对象之间建立依附性关系。CacheDependency 类监视依附性关系，以便在任何这些对象更改时，该缓存项都会自动移除。

命名空间: System.Web.Caching

程序集: System.Web (在 system.web.dll 中)

[语法](#)

Visual Basic (声明)

```
Public Class CacheDependency
    Implements IDisposable
```

Visual Basic (用法)

```
Dim instance As CacheDependency
```

C#

```
public class CacheDependency : IDisposable
C++
```

```
public ref class CacheDependency : IDisposable
```

J#

```
public class CacheDependency implements IDisposable
```

JScript

```
public class CacheDependency implements IDisposable
```

[备注](#)

如果向缓存添加一个依赖于另一个对象（如文件或文件数组）的项，则在该对象更改时会自动从缓存中移除该依赖项。例如，假设您基于 XML 文件中的数据创建一个 [DataSet](#) 对象。可以利用使 DataSet 依赖于 XML 文件的 CacheDependency 对象将该 DataSet 添加到缓存中。如果该 XML 文件发生更改，则 DataSet 从缓存中移除。

可以用 [Add](#) 和 [System.Web.Caching.Cache.Insert](#) 方法向应用程序缓存中添加具有依赖项的项。不能使用 [Item](#) 属性向缓存中添加具有依赖项的项。

若要设置依赖项，请创建 `CacheDependency` 类的一个实例，指定项依赖的文件、键或目录，然后将该依赖项传递给 `Add` 或 `System.Web.Caching.Cache.Insert` 方法。`CacheDependency` 实例可以表示单个文件或目录、一组文件或目录，或者带有一系列缓存键的一组文件或目录（这些缓存键表示 `Cache` 对象中存储的其他项）。

#### 示例

下面的代码示例演示如何使用 [HasChanged](#) 属性来确定 `CacheDependency` 是否在对 `Cache` 中某项的上一个请求之后已更改。将传入 `start` 参数的 `dt` 值设置为 [DateTime.Now](#)。

#### Visual Basic

##### [复制代码](#)

```
' Insert the cache item.
Dim dep As New CacheDependency(fileName, dt)
myCache.Insert("key", "value", dep)

' Check whether CacheDependency.HasChanged is true.
If dep.HasChanged Then
    Response.Write("<p>The dependency has changed.")
Else
    Response.Write("<p>The dependency has not changed.")
End If
C#
```

##### [复制代码](#)

```
// Insert the cache item.
CacheDependency dep = new CacheDependency(fileName, dt);
cache.Insert("key", "value", dep);

// Check whether CacheDependency.HasChanged is true.
if (dep.HasChanged)
    Response.Write("<p>The dependency has changed.");
else Response.Write("<p>The dependency has not changed.");
J#
```

##### [复制代码](#)

```
// Insert the cache item.
CacheDependency dep = new CacheDependency(fileName, dt);
cache.Insert("key", "value", dep);

// Check whether CacheDependency.HasChanged is true.
if (dep.get_HasChanged()) {
    get_Response().Write("<p>The dependency has changed.");
}
else {
    get_Response().Write("<p>The dependency has not changed.");
}
```

#### AggregateCacheDependency 类

**注意：**此类在 .NET Framework 2.0 版中是新增的。

组合 ASP.NET 应用程序的 [Cache](#) 对象中存储的项和 [CacheDependency](#) 对象的数组之间的多个依赖项。无法继承此类。

命名空间: `System.Web.Caching`

程序集: `System.Web`（在 `system.web.dll` 中）

#### 语法

Visual Basic（声明）

```
Public NotInheritable Class AggregateCacheDependency
    Inherits CacheDependency
```

Visual Basic（用法）

```
Dim instance As AggregateCacheDependency
```

C#

```
public sealed class AggregateCacheDependency : CacheDependency
```

C++



```
public ref class AggregateCacheDependency sealed : public CacheDependency
J#
public final class AggregateCacheDependency extends CacheDependency
JScript
public final class AggregateCacheDependency extends CacheDependency
备注
```

AggregateCacheDependency 类监视依赖项对象的集合，以便在任何依赖项对象更改时，该缓存项都会自动移除。数组中的对象可以是 CacheDependency 对象、[SqlCacheDependency](#) 对象、从 CacheDependency 派生的自定义对象或这些对象的任意组合。

AggregateCacheDependency 类与 CacheDependency 类的不同之处在于前者允许您将不同类型的多个依赖项与单个缓存项关联。例如，如果您创建一个从 SQL Server 数据库表和 XML 文件导入数据的页，则可创建一个 SqlCacheDependency 对象来表示数据库表的依赖项，以及一个 CacheDependency 来表示 XML 文件的依赖项。可创建 AggregateCacheDependency 类的一个实例，将每个依赖项添加到该类中，而不是为每个依赖项调用 [Cache.Insert](#) 方法。然后，可使用单个 [Insert](#) 调用使该页依赖于 AggregateCacheDependency 实例。

#### 示例

下面的代码示例使用 AggregateCacheDependency 类将名为 XMLDataSet 的 [DataSet](#) 添加到依赖于文本文件和 XML 文件的缓存中。

Visual Basic

#### 复制代码

```
' When the page is loaded, use the
' AggregateCacheDependency class to make
' a cached item dependent on two files.
```

```
Sub Page_Load(sender As Object, e As EventArgs)
```

```
    Dim Source As DataView
```

```
    Source = Cache("XMLDataSet")
```

```
    If Source Is Nothing
```

```
        Dim DS As New DataSet
```

```
        Dim FS As FileStream
```

```
        Dim Reader As StreamReader
```

```
        Dim txtDep As CacheDependency
```

```
        Dim xmlDep As CacheDependency
```

```
        Dim aggDep As AggregateCacheDependency
```

```
        FS = New FileStream(Server.MapPath("authors.xml"), FileMode.Open, FileAccess.Read)
```

```
        Reader = New StreamReader (FS)
```

```
        DS.ReadXml (Reader)
```

```
        FS.Close()
```

```
        Source = new DataView(ds.Tables(0))
```

```
        ' Create two CacheDependency objects, one to a
```

```
        ' text file and the other to an XML file.
```

```
        ' Create a CacheDependency array with these
```

```
        ' two objects as items in the array.
```

```
        txtDep = New CacheDependency(Server.MapPath("Storage.txt"))
```

```
        xmlDep = New CacheDependency(Server.MapPath("authors.xml"))
```

```
        Dim DepArray() As CacheDependency = {txtDep, xmlDep}
```

```
        ' Create an AggregateCacheDependency object and
```

```
        ' use the Add method to add the array to it.
```

```
        aggDep = New AggregateCacheDependency()
```

```
        aggDep.Add(DepArray)
```

```
        ' Call the GetUniqueId method to generate
```

```
        ' an ID for each dependency in the array.
```

```
        msg1.Text = aggDep.GetUniqueId()
```

```
        ' Add the new data set to the cache with
```

```
        ' dependencies on both files in the array.
```

```
Cache.Insert("XMLDataSet", Source, aggDep)
If aggDep.HasChanged = True Then
    chngMsg.Text = "The dependency changed at: " & DateTime.Now

Else
    chngMsg.Text = "The dependency changed last at: " & aggDep.UtcLastModified.ToString()
End If

cacheMsg1.Text = "Dataset created explicitly"
Else
    cacheMsg1.Text = "Dataset retrieved from cache"
End If

MyLiteral.Text = Source.Table.TableName
MyDataGrid.DataSource = Source
MyDataGrid.DataBind()
End Sub

Public Sub btn_Click(sender As Object, e As EventArgs )

If (MyTextBox.Text = String.Empty) Then
    msg2.Text ="You have not changed the text file."
Else
    msg2.Text="You added " & MyTextBox.Text & "."

' Create an instance of the StreamWriter class
' to write text to a file.
Dim sw As StreamWriter
sw = File.CreateText(Server.MapPath("Storage.txt"))

' Add some text to the file.
sw.Write("You entered:")
sw.WriteLine(MyTextBox.Text)

' Write arbitrary objects to the file as needed.
sw.Write("Text added at:")
sw.WriteLine(DateTime.Now)
sw.WriteLine("-----")
sw.Close()

End If
End Sub
```

<h3>  
心静似高山流水不动，心清若巫峰雾气不沾。  
</h3>

分类： ASP.NET(C#)  
标签： Cache

好文要顶

关注我

收藏该文

McJeremy&Fan

关注 - 6

粉丝 - 44

+加关注

60

« 上一篇： 我的学习方向  
» 下一篇： 摘录一篇插件式开发的文章

posted @ 2008-12-01 09:31 McJeremy&Fan 阅读(17179) 评论(1) 编辑 收藏

评论列表

#1楼 2015-07-14 13:42 czzjw

好文要顶

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)，[访问](#) 网站首页。

访问页面被拦截

因为您没有权限访问该页面，  
如有疑问请联系您的网络管理

最新**IT**新闻：

· 这项专利告诉你，LG也要推出柔性可折叠设备了？

· 由前Google员工打造，AI驱动的“未来诊所”正式开张

· 徕卡发布了M系新旗舰，年入百万的摄影师们都沸腾了！

· 周鸿祎：光靠炒作、PPT发布会做不好品牌

· Linux基金会：LC3会议将首次在中国举办

» 更多新闻...

最新知识库文章：

· 「代码家」的学习过程和学习经验分享

· 写给未来的程序媛

· 高质量的工程代码为什么难写

· [循序渐进地代码重构](#)

· 技术的正宗与野路子

» 更多知识库文章...

Copyright ©2017 McJeremy&Fan

----- 心静似高山流水不动，心清若巫峰雾气不沾 -----