**Asp.Net MVC 缓存**

缓存是一种保存资源副本并在下次请求时直接使用该副本的技术。当 web 缓存发现请求的资源已经被存储，它会拦截请求，返回该资源的拷贝。

Web应用缓存技术大体上可以分为两类:服务端缓存和客户端缓存。两种目标都是减少重复性内容的生成和网络传输工作，因为缓存数据存储的位置不同，而分为服务端缓存和客户端缓存。

**服务端缓存**

服务端缓存技术关注于服务端数据查询，生成或者操作技术。主要就是减少处理请求的工作量，减少数据库查询次数和生成HTML数据的CPU周期--减少每个bit的数据。  
对于服务端缓存来说，不管是刷新页面，重新输入地址，还是Control+F5都不会规避缓存，如果缓存数据有效，一定是请求的缓存数据。

**输出缓存(Output Cache)**

输出缓存是Asp.Net下最常用的缓存机制。输出缓存，缓存服务端生成的HTML数据--缓存Action下返回数据（Html/Json）。这样，在每次调用相同的Action时，就不需要再次执行Action方法。(应该不同客户端浏览器，都共享这个缓存???试试)

**缓存位置（Location）**

OutputCache使缓存的内容一般放在三个位置上：服务端，代理服务器，浏览器客户端。通过Loaction属性可以设置缓存的位置。

Loaction属性有如下值：

* Any
* Client
* Downstream
* Server
* None
* ServerAndClient

*默认值为Any，就是在三个位置都会缓存。但是应该根据不同的情况使用不同的缓存位置。比如：要缓存的内容是针对特定用户的，每个用户都会不同。这样的话，该缓存就不能保存在服务器上。应该保存在浏览器客户端上。*

**使用Output Cache**

在Controller或者Action上添加[OutputCache]特性，使得被添加的Controller或Action可以缓存返回的数据。*(在Action添加会缓存当前的Action，在Controller会缓存该Controller下的所有Action)*

如下代码:当第一次方法该Action时，开始计时10秒，此10秒内所有访问该Action的请求都会请求缓存数据。当10秒结束后，再重新开始等待新一次请求，开始新的10秒缓存。就是**每隔10秒丢掉旧缓存，等待新的请求，更新缓存数据**。

using System.Web.Mvc;

using System.Web.UI；

namespace MvcApplication1.Controllers

{

public class HomeController : Controller

{

//缓存时间10秒，缓存变量为无，缓存位置为服务端

[OutputCache(Duration=10, VaryByParam="none"， Location = OutputCacheLocation.Server)]

public ActionResult Index()

{

return View();

}

}

}

View:

@{

ViewBag.Title = "Index";

}

<h2>Index</h2>

<p>@DateTime.Now.ToString()</p>

点击F12,查看请求

需要注意的是：

1. 该缓存时间是绝对时间。
2. 此缓存是对所有访问该页面的用户都有效。
3. 不能保证缓存一定有效。当内存资源不够时，缓存就会自动地将没用的或者优先级低缓存清除。

**客户端缓存**

除了服务端缓存外，客户端也可以缓存数据。它避免了向服务器重复提交获取重复数据的请求，把一些重复数据缓存到本地。服务端缓存是为了更快的处理客户端请求，而客户端缓存则是为了避免不必要的请求。

*（浏览器会自动把静态资源缓存到浏览器）*

MVC中指定Location值为OutputCacheLocation.Client 使缓存在浏览器客户端上。

using System.Web.Mvc;

using System.Web.UI;

namespace MvcApplication1.Controllers

{

public class BadUserController : Controller

{

//缓存时间为10秒，缓存参数为无，缓存位置为客户端

[OutputCache(Duration = 10, VaryByParam = "none",Location = OutputCacheLocation.Client)]

public ActionResult ClientCache()

{

return View();

}

}

}

@{

ViewBag.Title = "ClientCache";

}

<h2>ClientCache</h2>

<p>@DateTime.Now.ToString()</p>

**客户端缓存和服务端缓存不一样。**

刷新，重新输入地址，和Control+F5都有可能破坏客户端缓存，从服务端重新获取数据。

浏览器刷新，和重新输入地址会避免请求该URL页面的客户端缓存，只避免请求该URL页面的缓存。（如果该页面有其他URL是被客户端缓存的，这些资源或页面的缓存不会被避免）。

**那么什么情况下客户端缓存才有效？**

通过URL访问，客户端缓存才有效。

*（页面的静态资源）*

比如：

1. 页面A是客户端缓存，同时页面A有一个跳向页面B的链接。
2. 通过A到达页面B，同时页面B也有一个链接，这个链接跳向A。
3. 通过B再次访问A，此时页面A获取的数据就是客户端的缓存数据，并没有请求服务端，是没有请求服务端。不是304，status-code依然是200。

**Status-Code:304/200(from cache)**

* 304

只有当客户端和服务端同时都缓存了数据。且缓存没有更新的时候，才会有304。即这个缓存是要到服务端验证（根据ETag和If-Modify-Since），该缓存是否最新。如果要更新缓存，从服务端获取数据，status code：200，否则status code：304.

* 304 和200（from cache)区别

304是会到服务端去校验一次当前客户端缓存是否有效（根据ETag和If-Modify-Since）。而200（from cache）则没有向服务端校验，也没有向服务端请求，直接使用了客户端缓存。

有时我们又需要避免这种没有向服务端请求，直接使用缓存的情况。解决办法就是更改这个缓存的url，添加一个版本号或唯一值。这样因为url的更改使得在客户端没有对应的url缓存，就会从服务端重新获取，再缓存该URL的数据。

**不同内容的输出缓存**

之前的缓存都是Action返回相同的内容。如果Action每次返回的内容不同，那又该怎么缓存这些不同的内容呢？

使用OutputCache特性的VaryByParam属性来解决这个问题。当表单参数或查询字符串参数变化时，该属性能够创建同一个Action下不同的缓存。

如下代码：Master 获取列表。Details 获取列表中选择项的详细内容。通过使用VaryByParam来缓存不同的id的列表项的详细内容。

using System.Web.Mvc;

namespace MvcApplication1.Controllers

{

public class MoviesController : Controller

{

public MoviesController()

{

}

[OutputCache(Duration=int.MaxValue, VaryByParam="none")]

public ActionResult Master()

{

//获取列表

return View();

}

[OutputCache(Duration = int.MaxValue, VaryByParam = "id")]

public ActionResult Details(int id)

{

//根据参数id，从数据库中获取指定详细内容，并缓存该内容。不同的id会得到不同的内容，自然也会有缓存。

//但是如果设置VaryByParam="none"那么不管id是多少，都直接从缓存中获取数据，不执行该Action，这样就会只返回第一次选择项的数据。

return View();

}

}

}

Details()操作包括一个带有值“Id”的VaryByParam属性。当将Id参数的不同值传递给控制器操作时，将生成不同的缓存内容。

VaryByParam可以根据参数缓存不同的内容

* 当VaryByParam="\*"： 每当表单或查询字符串参数变化时，创建一个不同的缓存版本。
* 当VaryByParam="none"： 不创建不同的缓存内容,不根据参数缓存不同的内容，即只有一个内容的缓存。
* 当VaryByParam="参数列表"： 为不同的参数创建不同的缓存版本。

**缓存配置**

除了在OutputCache特性上直接配置缓存策略，可以在web.config文件中使用缓存配置文件，同一管理缓存的策略。使用配置文件相比直接使用属性有如下几点好处：

1. 可以实现一次定义，多处使用。
2. 可以修改web配置文件，而无需重新编译应用程序。(*如果想把已经部署到生产环境中的应用程序禁用缓存，可以修改web配置文件中定义的缓存配置。对web配置文件的任何更改都将被自动检测并应用。)*

例如，web.config部分定义了一个名为“cache1Hour”的缓存配置文件。使用该配置项时，只需指定CacheProfile=配置项名称即可。

<caching>

<outputCacheSettings>

<outputCacheProfiles>

<add name="Cache1Hour" duration="3600" varyByParam="none"/>

</outputCacheProfiles>

</outputCacheSettings>

</caching>

using System;

using System.Web.Mvc;

namespace MvcApplication1.Controllers

{

public class ProfileController : Controller

{

//配置文件中的缓存策略名称赋值给CacheProfile

[OutputCache(CacheProfile="Cache1Hour")]

public string Index()

{

return DateTime.Now.ToString();

}

}

}

简单介绍下Http缓存的头相关信息：

| **消息头** | **值** | **类型** | **说明** |
| --- | --- | --- | --- |
| Expires | Thu, 30 Nov 2017 08:21:14 GMT | 响应 | 过期时间，为格林威治时间 (GMT) |
| Pragma | no-cache | 响应 | 忽略浏览器缓存（Http1.1用Cache-Control代替） |
| Cache-Control | no-cache | 请求/响应 | 客户端缓存验证 |
| Cache-Control | no-store | 请求/响应 | 不在任何地方保存数据，不允许被缓存 |
| Cache-Control | max-age=[秒] | 请求/响应 | 设置浏览器缓存最长时间 |
| Cache-Control | public | 响应 | 缓存在任何地方 |
| Cache-Control | private | 响应 | 缓存该用户的浏览器 |
| Last-Modified | Thu, 30 Nov 2017 08:21:14 GMT | 响应 | 告诉浏览器服务端最后一次修改的时间 |
| If-Modified-Since | Thu, 30 Nov 2017 08:21:14 GMT | 请求 | 如果浏览器中Last-Modofied有值，在请求中把值给If-Modified-Since,提交给服务端 |
| ETag | 3df04c15b968d31:0 | 响应 | 该资源及其版本在服务端的唯一标识 |
| If-None-Match | 3df04c15b968d31:0 | 请求 | 把上次请求中获取到的ETag值，赋值给If-None-Match并提交给服务端 |
| Vary | Accept-Encoding | 响应 | 从多个缓存副本中选择匹配的版本 |

**有几个容易理解错误的点**

* no-cache: 使用no-cache 指令的目的是为了防止从缓存中使用过期的资源，所以每次使用缓存时都要到服务端去验证。从字面意思上很容易把no-cache误解成为不缓存，但事实上no-cache代表不缓存过期的资源，缓存会向源服务器进行有效期确认后处理资源。
* no-store: 不存储客户端相关请求或服务器响应的任何内容，即真正的不缓存。

如有不对，请多多指教。