

响应缓存与响应压缩

2018年11月28日 14:18

■ HTTP协议浏览器缓存

HTTP 1.1 缓存规范，通过 Cache-control 标头进行缓存控制。

详解 HTTP 的缓存机制与原理

来自 <<https://www.jianshu.com/p/f080181021cb>>

HTTP协议中的缓存

来自 <<http://www.cnblogs.com/TankXiao/archive/2012/11/28/2793365.html>>

彻底弄懂HTTP缓存机制及原理

来自 <<https://www.jianshu.com/p/dedb04225bc5>>

原始方法：Response.Headers[Microsoft.Net.Http.Headers.HeaderNames.CacheControl] = "public, max-age=600";

MVC中提供的方法：ResponseCacheAttribute

ResponseCache的设置	响应头
[ResponseCache(Duration = 600, Location = ResponseCacheLocation.Client)]	Cache-Control: private, max-age=600
[ResponseCache(Location = ResponseCacheLocation.None, NoStore = true)]	Cache-Control:no-cache, no-store
[ResponseCache(Duration = 60, VaryByHeader = "User-Agent")]	Cache-Control : public, max-age=60 Vary : User-Agent

同URL不同参数缓存不同副本：VaryByQueryKeys 和 VaryByHeader

NoStore 配置：当 NoStore 为 true 值，Cache-Control 为 no-store。

Location 配置：Location 为 None 值，Cache-Control 为 no-store 和 no-cache 标头，标头 Pragma 将被设置为 no-cache 值。

如果 NoStore 为 false，且 Location 为 None，Cache-Control 和 Pragma 都将设置为 no-cache 值。

除了直接给定参数，还支持策略配置：

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc(options =>
    {
        options.CacheProfiles.Add("Default",
            new CacheProfile()
            {
                Duration = 60
            });
        options.CacheProfiles.Add("Never",
            new CacheProfile()
            {
                Location = ResponseCacheLocation.None,
                NoStore = true
            });
    });
}
```

使用缓存策略配置：

```
[ResponseCache(Duration = 30)]
public class HomeController : Controller
```

```
{
    [ResponseCache(CacheProfileName = "Default")]
    public IActionResult Index()
    {
        return View();
    }
}
```

■ 在中间件中缓存

中间件缓存使用内存缓存机制，在服务器内存中进行。

Install-Package Microsoft.AspNetCore.ResponseCaching

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddResponseCaching();
}

public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    app.UseResponseCaching();
}
```

可以配置中间件缓存行为：URL区分大小写，响应最大不能超过1024字节。

```
services.AddResponseCaching(options =>
{
    options.UseCaseSensitivePaths = true;
    options.MaximumBodySize = 1024;
});
```

缓存条件：200状态，GET 或 HEAD 请求，必须为 public，不能有 Set-Cookie，Content-Length不能超标，头发无 Authorization 认证，等等，详细参阅官网。

■ 缓存标记帮助程序

两种方式：基于本机内存和分布式内存两种方式。

内存缓存标记写法：

```
<cache>@DateTime.Now</cache>
```

是否缓存，默认值为 true。如果设置为 false，则不会缓存呈现的输出。

```
<cache enabled="true">
    Current Time Inside Cache Tag Helper: @DateTime.Now
</cache>
```

绝对过期时间：

```
<cache expires-on="@new DateTime(2025, 1, 29, 17, 02, 0)">
    Current Time Inside Cache Tag Helper: @DateTime.Now
</cache>
```

绝对过期时间长度：

```
<cache expires-after="@(TimeSpan.FromSeconds(120))">
    Current Time Inside Cache Tag Helper: @DateTime.Now
</cache>
```

滑动过期时间长度:

```
<cache expires-sliding="@(TimeSpan.FromSeconds(60))">
    Current Time Inside Cache Tag Helper: @DateTime.Now
</cache>
```

标头值发生更改时触发缓存刷新:

```
<cache vary-by-header="User-Agent">
    Current Time Inside Cache Tag Helper: @DateTime.Now
</cache>
```

查询字符串发生更改时触发缓存刷新:

```
<cache vary-by-query="Make,Model">
    Current Time Inside Cache Tag Helper: @DateTime.Now
</cache>
```

路由数据参数值发生更改时触发缓存刷新:

```
routes.MapRoute( name: "default", template: "{controller=Home}/{action=Index}/{Make?}/{Model?}" );
```

```
<cache vary-by-route="Make,Model">
    Current Time Inside Cache Tag Helper: @DateTime.Now
</cache>
```

Cookie 值发生更改时触发缓存刷新

```
<cache vary-by-cookie=".AspNetCore.Identity.Application">
    Current Time Inside Cache Tag Helper: @DateTime.Now
</cache>
```

登录用户更改时触发缓存刷新, 也就是 User.Identity.Name 变化的时候。

```
<cache vary-by-user="true">
    Current Time Inside Cache Tag Helper: @DateTime.Now
</cache>
```

允许自定义缓存刷新方式, 当对象发生更改时。

```
public IActionResult Index(int myParam1, int myParam2)
{
    return View(viewName, myParam1 + myParam2);
}
```

```
<cache vary-by="@Model">
    Current Time Inside Cache Tag Helper: @DateTime.Now
</cache>
```

可设置缓存优先级, 缓存驱逐机制会根据优先级, 可能的值: High、Low、NeverRemove 和 Normal。

```
<cache priority="High">
    Current Time Inside Cache Tag Helper: @DateTime.Now
```

</cache>

分布式缓存标记帮助程序:

通过分布式接口 IDistributedCache 缓存数据, 修改缓存中的 KEY 键:

```
<distributed-cache name="my-distributed-cache-unique-key-101">
    Time Inside Cache Tag Helper: @DateTime.Now
</distributed-cache>
```

■ 静态文件缓存

```
app.UseStaticFiles(new StaticFileOptions
{
    OnPrepareResponse = context =>
    {
        context.Context.Response.GetTypedHeaders().CacheControl = new
        Microsoft.Net.Http.Headers.CacheControlHeaderValue
        {
            Public = true,
            MaxAge = System.TimeSpan.FromDays(1)
        };
    }
});
```

■ 基于 CDN 缓存

全称: Content Delivery Network 或 Content Ddistribute Network, 即内容分发网络。

CDN是什么? 使用CDN有什么优势?

来自 <<https://www.zhihu.com/question/36514327?rf=37353035>>

■ 响应内容压缩

网络带宽是有限的资源, 压缩可以网络传输资源的大小。

何时使用响应内容压缩: 1、无法使用反向代理压缩时; 2、直接托管, 未使用反向代理服务。

最佳实践: 不压缩本机压缩过的文件, 只压缩可压缩文件, 压缩小文件开销可能更大。

Install-Package Microsoft.AspNetCore.ResponseCompression

```
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddResponseCompression();
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        app.UseResponseCompression();
    }
}
```

Accept-Encoding header values	Middleware Supported	Description
br	Yes (default)	Brotli compressed data format
deflate	No	DEFLATE compressed data format
exi	No	W3C Efficient XML Interchange
gzip	Yes	Gzip file format
identity	Yes	"No encoding" identifier: The response must not be encoded.
pack200-gzip	No	Network Transfer Format for Java Archives
*	Yes	Any available content encoding not explicitly requested

测试工具：Fiddler、Firebug 或 Postman。设置 Accept-Encoding: br,gzip

不设置压缩提供程序时：优先 Brotli 算法，然后 Gzip 算法。

显式添加任何压缩提供程序，甚至自定义压缩提供程序：

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddResponseCompression(options =>
    {
        options.Providers.Add<BrotliCompressionProvider>();
        options.Providers.Add<GzipCompressionProvider>();
        options.Providers.Add<CustomCompressionProvider>();
    });
}
```

Brotli 压缩提供程序：BrotliCompressionProvider

可通过 BrotliCompressionProviderOptions 设置 Brotli 算法压缩级别。

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddResponseCompression();

    services.Configure<BrotliCompressionProviderOptions>(options =>
    {
        options.Level = CompressionLevel.Fastest;
    });
}
```

压缩级别	描述
CompressionLevel.Fastest	即使不以最佳方式压缩生成的输出，应尽可能快地完成压缩。
CompressionLevel.NoCompression	应该在不进行压缩。
CompressionLevel.Optimal	响应应以最佳方式压缩，即使压缩操作将耗费更多时间来完成。

Gzip 压缩提供程序：GzipCompressionProvider

Gzip 压缩提供程序默认情况下添加到与压缩提供程序的数组Brotli 压缩提供程序。

当客户端支持 Brotli 压缩的数据格式时，则将默认压缩为 Brotli 压缩。如果客户端不支持 Brotli 时客户端支持 Gzip 压缩, 压缩默认设置为 Gzip。

通过 GzipCompressionProviderOptions 也可设置压缩级别。

自定义压缩提供程序

```
public class CustomCompressionProvider : ICompressionProvider
{
    public string EncodingName => "myzip";
    public bool SupportsFlush => true;

    public Stream CreateStream(Stream outputStream)
    {
        // Create a custom compression stream wrapper here
        return outputStream;
    }
}
```

默认支持以下MIME类型压缩:

application/javascript、application/json、application/xml、text/css、text/html、text/json、text/plain 和 text/xml。

自定义 MIME 类型

```
options.MimeTypes = ResponseCompressionDefaults.MimeTypes.Concat(new[] { "image/svg+xml" });
```

关于反向代理服务器压缩配置

IIS配置响应内容压缩



Nginx Compression and Decompression

来自 <<https://docs.nginx.com/nginx/admin-guide/web-server/compression/>>