

深入探索HTTP协议的奥秘

北京理工大学计算机学院
金旭亮

熟练掌握几个探索HTTP奥秘的工具

Chrome
Developer
Tools

- 浏览器自带的开发工具，分析网页与HTTP请求和响应的利器

Fiddler

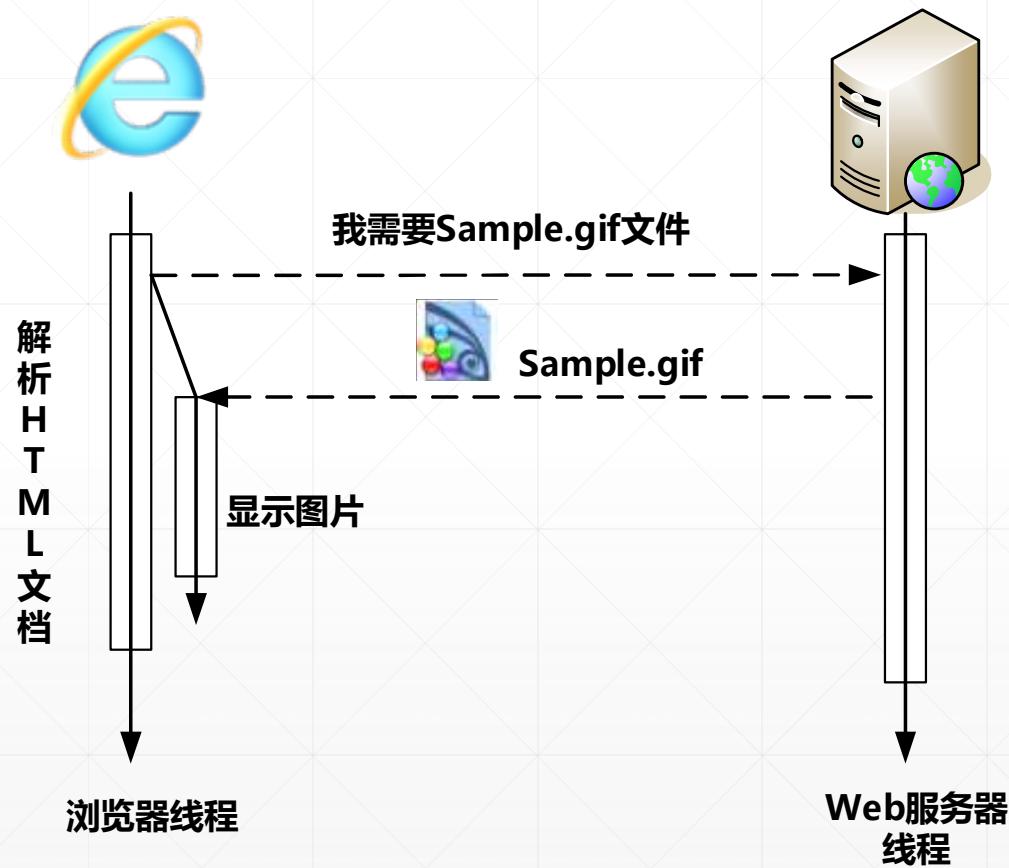
- 功能强大的通用HTTP开发与调试工具

HttpClient

- .NET所提供的简单易用的组件，可以方便地编程访问Web资源

Chrome Developer Tools

场景：浏览器是怎样解析HTML网页的？



显示一个网页需多次从Web Server提取数据

The screenshot shows a web browser window displaying a website for 'Jin Xiangliang's Online Education'. The browser's address bar shows the URL www.jinxuliang.com. The main content area displays a welcome message and three course categories: 'Java程序设计', 'C#编程语言与.NET技术基础', and '面向对象软件开发实践'. Below the main content, the browser's developer tools are open, specifically the Network tab. A red circle highlights the Network tab and the list of requests. The table below shows the details of the network requests made to load the page.

Name	Method	Status	Type	Initiator	Size	Time	Timeline
www.jinxuliang.com	GET	200	text/html	Other	24.7 KB	69 ms	
css?v=B_d5lajadZCW5ratubi99U73e6kzeYGr1p...J...	GET	304	text/css	www.jinxuliang.com:/7	169 B	10 ms	
modernizr?v=inCVuEf6J4Q07A0AcRsbJic_Ue5Mw...	GET	304	text/javascript	www.jinxuliang.com:/9	169 B	10 ms	
jquery?v=gkWytHptwkFjvHuNinBjchfwLwc_KbE-...	GET	304	text/javascript	www.jinxuliang.com:/279	169 B	11 ms	
bootstrap?v=NzP9D5jO6GVMzY8_4Kfk811W0Vrrh...	GET	304	text/javascript	www.jinxuliang.com:/281	169 B	32 ms	
angular.min.js	GET	304	application/x-javascript	www.jinxuliang.com:/283	211 B	11 ms	
ui-bootstrap-tpls.min.js	GET	304	application/x-javascript	www.jinxuliang.com:/284	211 B	32 ms	
angular.min.js.map	GET	304	text/plain	Other	93 B	42 ms	

使用浏览器配置的开发工具
(通常使用F12键调出来)，
可以直观地看到浏览器网页
的解析过程

浏览器在显示一个包容有
多种类型资源(图片、视
频、文字等)的复杂Web
网页时，通常需要向Web
Server发出多次HTTP请求。

试验：使用Chrome分析“浏览器端的数据缓冲”特性

The screenshot shows the Network tab in Chrome DevTools. It lists various resources loaded from 'jinxuliang.com' with their status codes. All status codes are 200, indicating they were loaded from the server. A red box highlights the 'Status' column.

Name	Method	Status
jinxuliang.com	GET	200
css?v=aWhffM8LdJyMUSTNvUufPVpmPmFLFaTdbzlz4K4...	GET	200
modernizr?v=inCVuEFe6J4Q07A0AcRsbJic_UE5MwpRMN...	GET	200
jquery?v=gkWyJthHPtwkFjvHuNinBjchlfwLwc_KbE-H26J2...	GET	200
bootstrap?v=OfX192nbUgK5NE8ftV4Ef6ToCtjUTli3wC5jn...	GET	200
angular.min.js	GET	200
ui-bootstrap-tpls.min.js	GET	200
favicon.ico	GET	200

第一次访问，状态码都是200

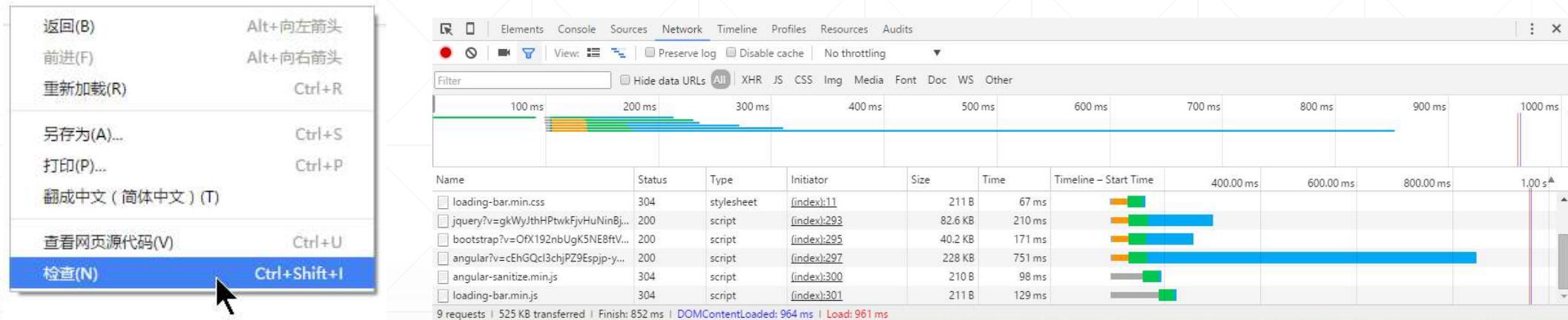
The screenshot shows the Network tab in Chrome DevTools after refreshing the page. While most resources still have a status code of 200, several have changed to 304, indicating they were loaded from the browser's cache. A red box highlights the 'Status' column.

Name	Method	Status
jinxuliang.com	GET	200
css?v=aWhffM8LdJyMUSTNvUufPVpmPmFLFaTdbzlz4K4...	GET	304
modernizr?v=inCVuEFe6J4Q07A0AcRsbJic_UE5MwpRMN...	GET	304
jquery?v=gkWyJthHPtwkFjvHuNinBjchlfwLwc_KbE-H26J2...	GET	304
bootstrap?v=OfX192nbUgK5NE8ftV4Ef6ToCtjUTli3wC5jn...	GET	304
angular.min.js	GET	304
ui-bootstrap-tpls.min.js	GET	304
favicon.ico	GET	200

刷新，不少状态码变为了304.....

利用数据缓存原理，人们在开发中广泛使用了“**CDN (Content Delivery Network)**”技术以减少不必要的数据传输，加快网页的呈现。

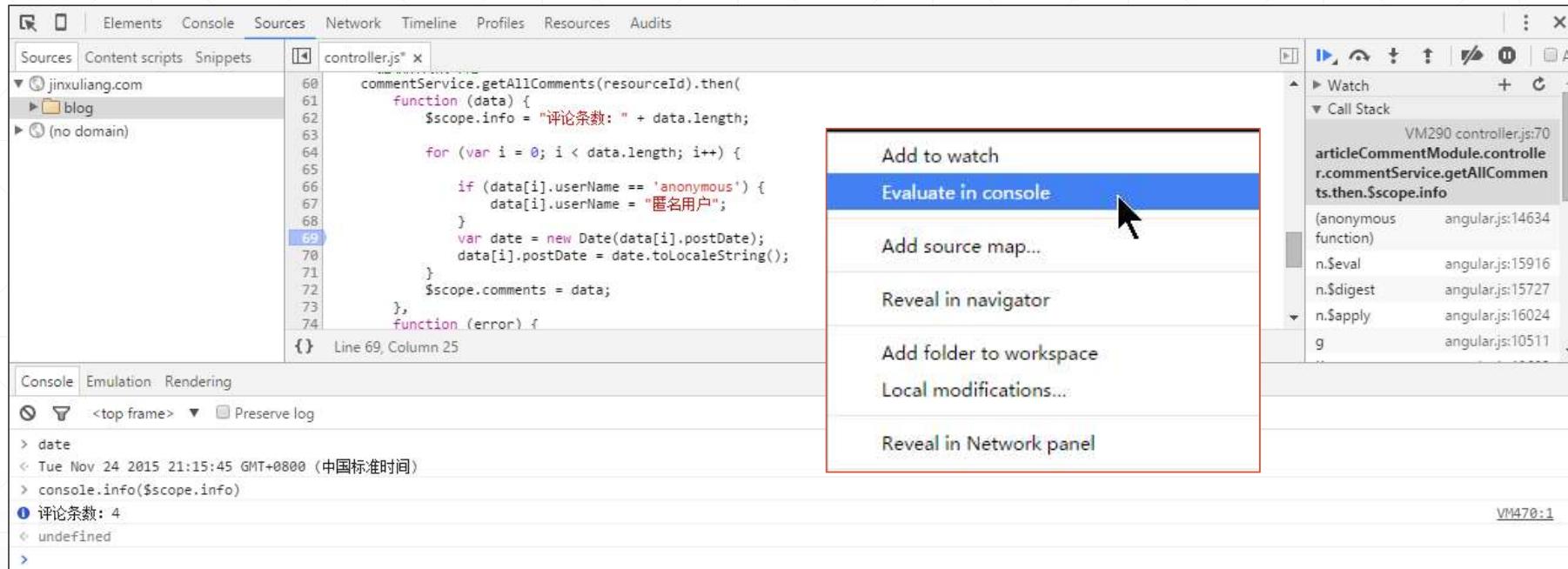
掌握Chrome Developer Tools的几招-1



快速定位并查看指定部分的HTML代码

Network选项卡：查看访问特定网页所发出的所有HTTP请求和响应，分析网页显示速度慢的瓶颈所在。

掌握Chrome Developer Tools的几招-2



使用Source选项卡，可以查看网页中的JavaScript代码，并且可以设置断点单步跟踪；选中部分代码，右击，选“Evaluate in console”，可以在Console面板中执行并显示结果。

在Console选项卡中使用“console.info(JS对象）”，可以查看到所有指定对象的相关属性及方法。

Fiddler

下载并安装Fiddler

<http://www.telerik.com/download/fiddler>

The screenshot shows the Telerik website with the URL www.telerik.com/download/fiddler. The main content area features a diagram illustrating Fiddler's role as a proxy between clients and servers. A red oval highlights the 'Free download' button, which is also pointed to by a red arrow from the adjacent 'Download Fiddler' page.

Fiddler
The free web debugging proxy for any browser, system or platform

Diagram illustrating Fiddler's role as a proxy between clients and servers.

Free download

Key Features

- Web Debugging**: Debug traffic from PC, Mac or Linux systems and mobile devices. Ensure the proper cookies, headers and cache directives are transferred between the
- Performance Testing**: Fiddler lets you see the "total page weight," HTTP caching and compression at a glance. Isolate performance bottlenecks with rules like "Flag any
- HTTP/HTTPS Traffic Recording**: Fiddler is a free web debugging proxy which logs all HTTP(s) traffic between your computer and the Internet. Use it to

Introducing a new Priority Support package for Fiddler. [Read blog post](#)

fiddler4setup.exe 显示所有下载内容...

Download Fiddler

Choose your download based on the version of the .NET Framework installed on your PC.
Users of Windows 8+ should choose Fiddler4.

Fiddler for .NET4
Version 4.6.15, ~1mb Signed EXE
Released on November 9, 2015

Fiddler for .NET2
Version 2.6.15, ~1mb Signed EXE
Released on November 9, 2015

How do you plan to use Fiddler? ▾
Your email
 Keep me informed about Fiddler and relevant offerings

Download Fiddler

可以使用Fiddler分析
浏览器访问特定网站
的信息交换过程

The screenshot shows the Fiddler Web Debugger interface. The main window displays a list of network sessions. Session 1 is selected, showing a request to `jinxuliang.com /`. The Request Headers tab shows the client's HTTP headers, including User-Agent (Mozilla/5.0). The Response tab shows the server's HTTP response, which includes the HTML content of the website.

#	Result	Protocol	Host	URL
1	200	HTTP	jinxuliang.com	/
2	304	HTTP	jinxuliang.com	/Content/css?v=aWhffM8...
3	304	HTTP	jinxuliang.com	/bundles/modernizr?v=InC...
4	304	HTTP	jinxuliang.com	/bundles/jquery?v=gkWY...
5	304	HTTP	jinxuliang.com	/bundles/bootstrap?v=Of...
6	304	HTTP	jinxuliang.com	/Scripts/angular.min.js
7	304	HTTP	jinxuliang.com	/Scripts/angular-ui/ui-boot...
8	200	HTTP	jinxuliang.com	/favicon.ico

Request Headers
GET / HTTP/1.1

Cache
Cache-Control: max-age=0

Client
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate, sdch
Accept-Language: zh-CN,zh;q=0.8
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/44.0.2403.130 Safari/537.36

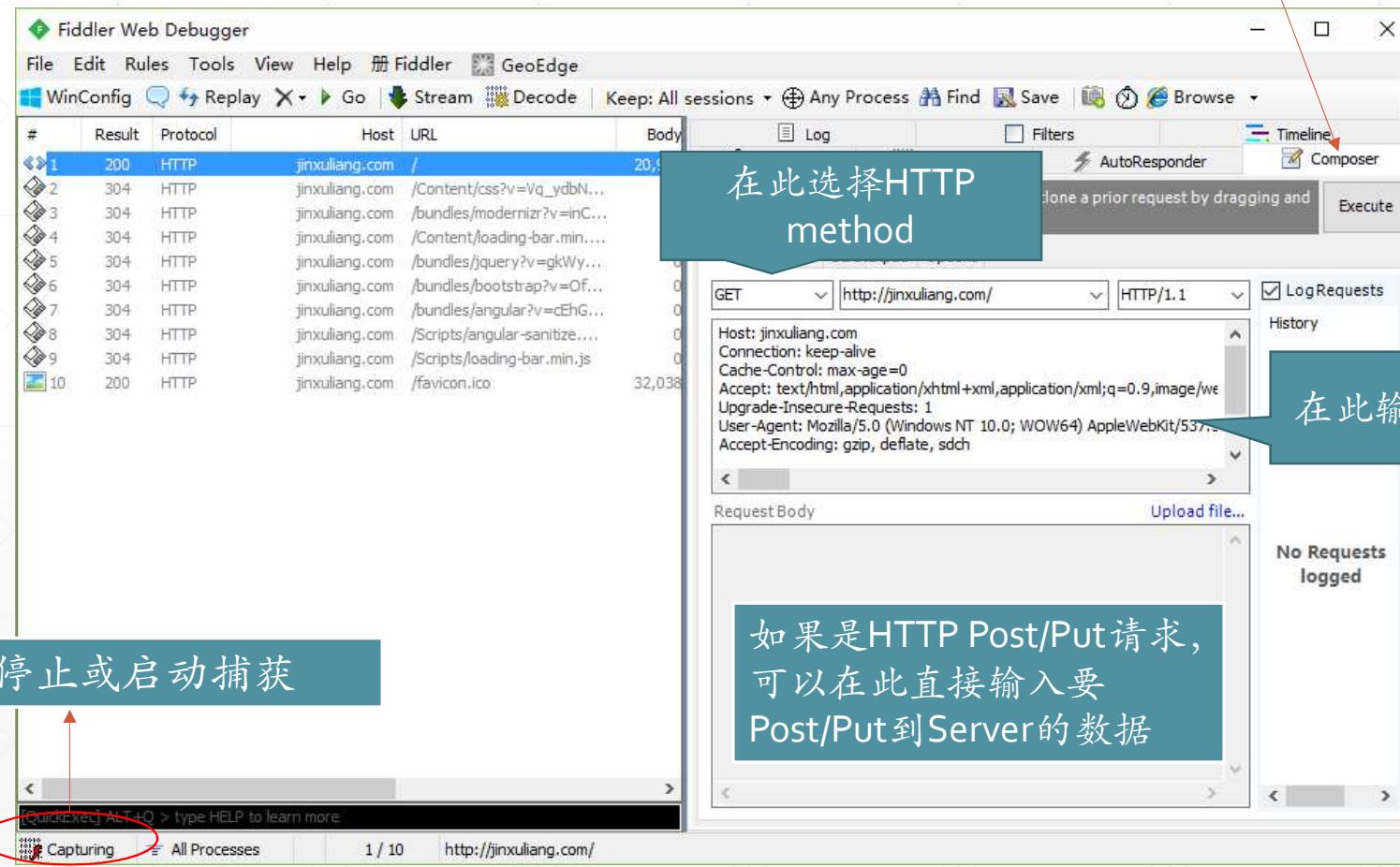
Transport
Connection: keep-alive
Host: jinxuliang.com

Response
HTTP/1.1 200 OK
Cache-Control: private
Content-Type: text/html; charset=utf-8
Server: Microsoft-IIS/7.5
X-AspNetMvc-Version: 5.2
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
Date: Sun, 23 Aug 2015 03:17:44 GMT
Content-Length: 25105

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>
```

Fiddler的基础使用技巧

使用Composer选项卡人工组合生成HTTP请求



HttpClient简介

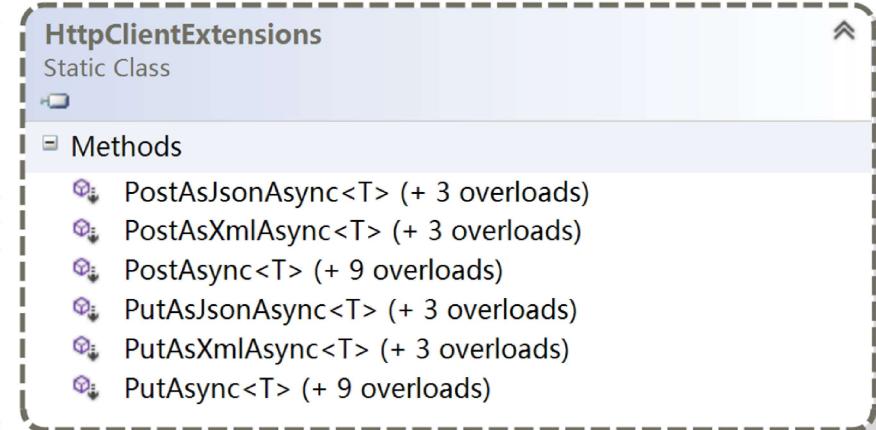
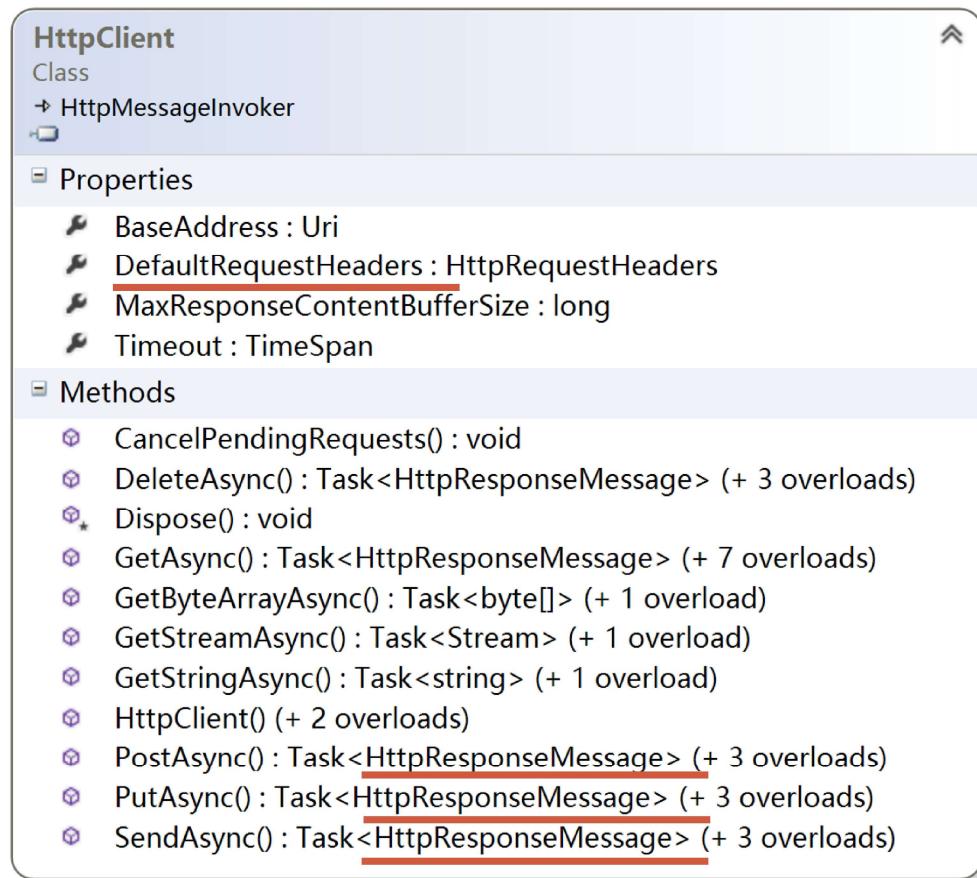
使用HttpClient

HttpClient可用于取代原先基类库中System.Net命名空间中的WebClient、WebRequest、WebResponse等组件，更简单易用。

HttpClient使用HttpRequestMessage/HttpResponseMessage封装请求与响应消息，支持async/await异步编程模式，可以同时向多个URI（甚至是跨域的）发出HTTP请求。

只要你了解一些HTTP的基础知识，你会发现HttpClient及相关类型的封装相当直观.....

HttpClient类所定义的属性与方法



有几个扩展方法，直接支持以Post和Put方式提交Json和Xml数据到服务端

HttpRequestHeaders

Sealed Class

↳ HttpHeaders

Properties

- Accept : `HttpHeaderValueCollection<MediaTypeWithQualityHeaderValue>`
- AcceptCharset : `HttpHeaderValueCollection<StringWithQualityHeaderValue>`
- AcceptEncoding : `HttpHeaderValueCollection<StringWithQualityHeaderValue>`
- AcceptLanguage : `HttpHeaderValueCollection<StringWithQualityHeaderValue>`
- Authorization : `AuthenticationHeaderValue`
- CacheControl : `CacheControlHeaderValue`
- Connection : `HttpHeaderValueCollection<string>`
- ConnectionClose : `bool?`
- Date : `DateTimeOffset?`
- Expect : `HttpHeaderValueCollection<NameValuePairWithParametersHeaderValue>`
- ExpectContinue : `bool?`
- From : `string`
- Host : `string`
- IfMatch : `HttpHeaderValueCollection<EntityTagHeaderValue>`
- IfModifiedSince : `DateTimeOffset?`
- IfNoneMatch : `HttpHeaderValueCollection<EntityTagHeaderValue>`
- IfRange : `RangeConditionHeaderValue`
- IfUnmodifiedSince : `DateTimeOffset?`
- MaxForwards : `int?`
- Pragma : `HttpHeaderValueCollection<NameValuePairHeaderValue>`
- ProxyAuthorization : `AuthenticationHeaderValue`
- Range : `RangeHeaderValue`
- Referrer : `Uri`
- TE : `HttpHeaderValueCollection<TransferCodingWithQualityHeaderValue>`
- Trailer : `HttpHeaderValueCollection<string>`
- TransferEncoding : `HttpHeaderValueCollection<TransferCodingHeaderValue>`
- TransferEncodingPreliminary : `bool?`
- Upgrade : `string`
- UserAgent : `string`
- Via : `HttpHeaderValueCollection`
- Warning : `HttpHeaderValueCollection`

这些属性，都直接对应着
HTTP协议中所规定的标准
HTTP Header。

HttpClient的DefaultRequestHeaders引用一个
HttpRequestHeaders对象，可以设定Http Header
值。

基类HttpHeaders可以添加和移除HTTP Header。

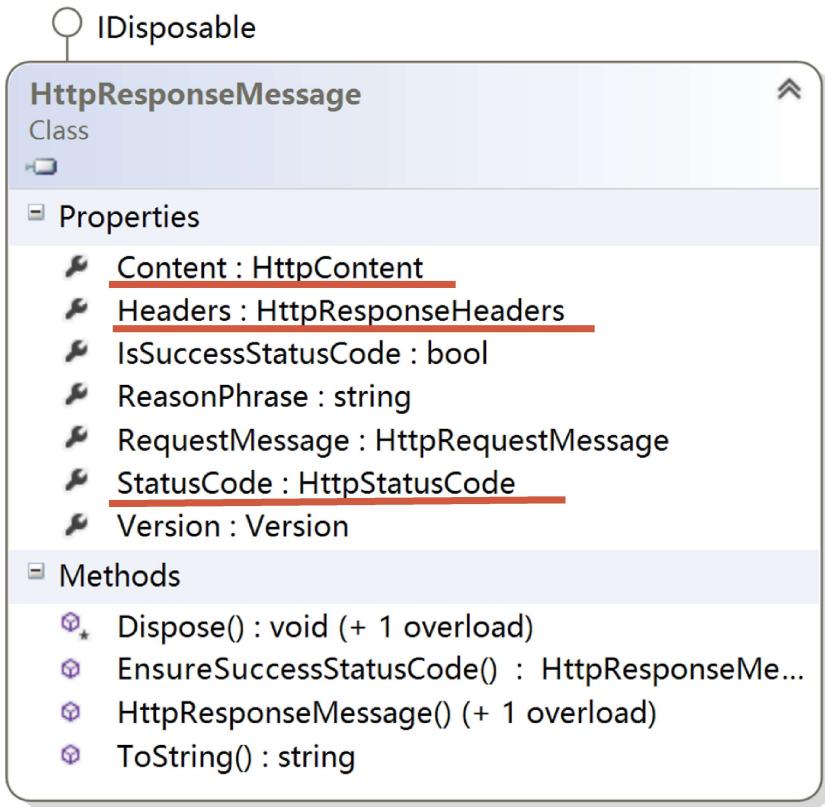
HttpHeaders

Abstract Class

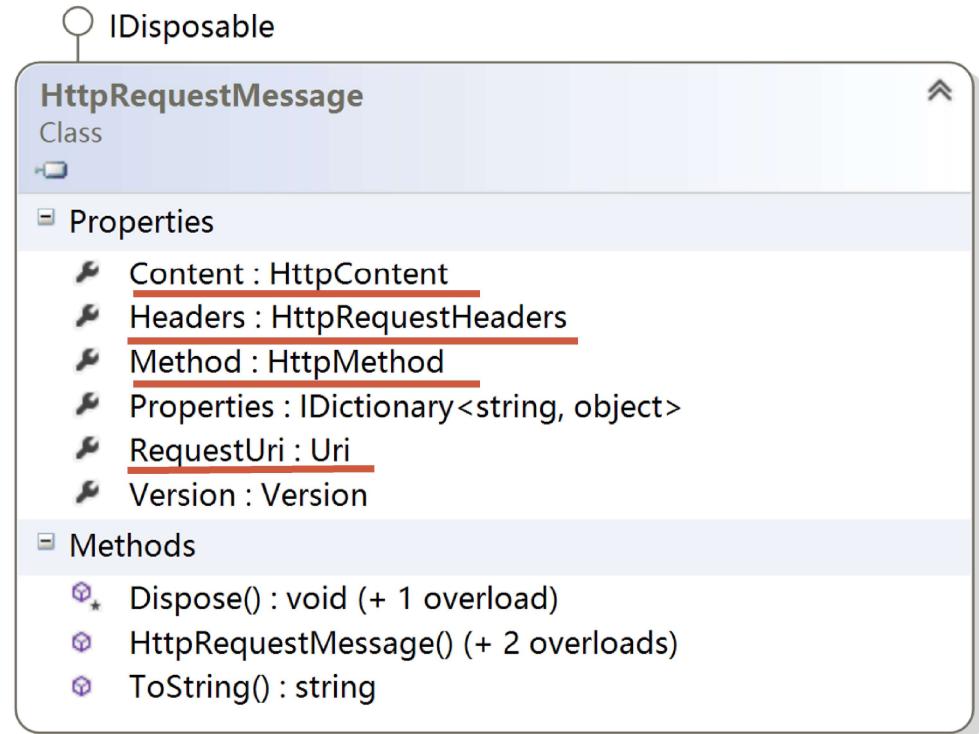
Methods

- `Add() : void (+ 1 overload)`
- `Clear() : void`
- `Contains() : bool`
- `GetEnumerator() : IEnumerator<KeyValuePair<string, IEnumerable<string>>>`
- `GetValues() : IEnumerable<string>`
- * `HttpHeaders()`
- `Remove() : bool`
- `ToString() : string`
- `TryAddWithoutValidation() : bool (+ 1 overload)`
- `TryGetValues() : bool`

HTTP请求和响应消息，被封装为**HttpResponseMessage**和**HttpRequestMessage**类

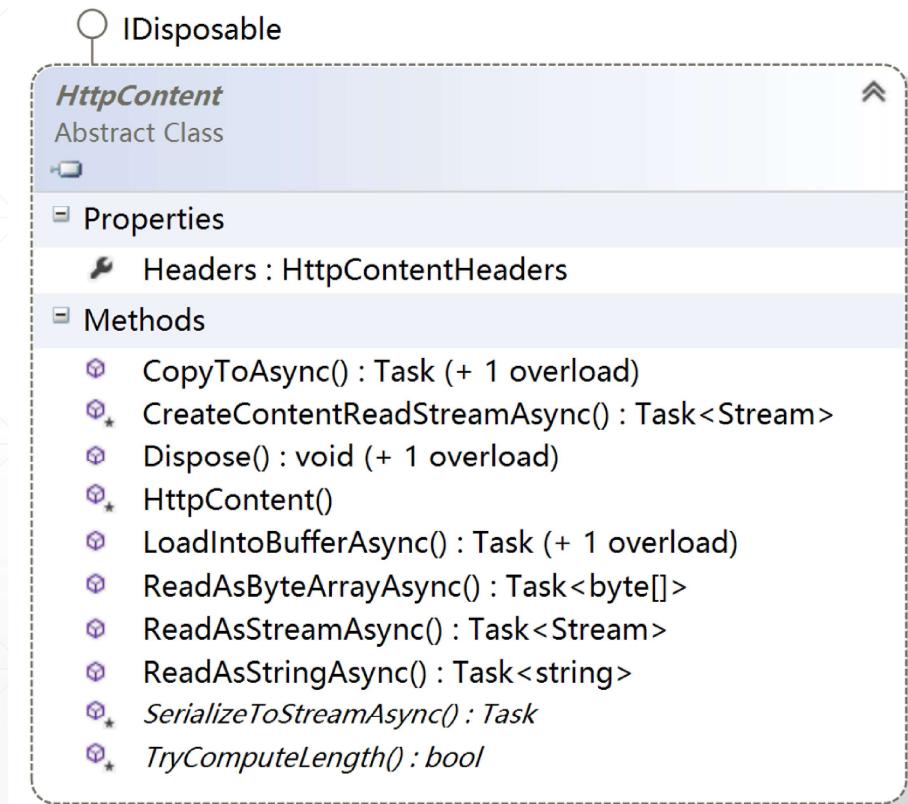


HttpClient的GetAsync()/PostAsync()等方法返回`HttpResponseMessage`对象。

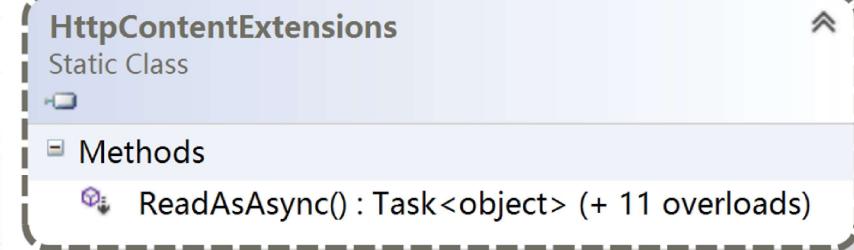


HttpClient的SendAsync()方法以`HttpRequestMessage`对象为方法参数

HTTP消息的主体抽象为HttpContent类型，并且定义了几个子类。



System.Object
System.Net.Http.HttpContent
System.Net.Http.ByteArrayContent
System.Net.Http.MultipartContent
System.Net.Http.StreamContent



HttpContent自己和一组扩展方法提供了读取HttpContent的便捷手段。

HttpClient基本编程技巧

使用HttpClient组件发出Get请求

使用HttpClient组件，向Web Server发出一个HTTP Get请求，并且通过指定“Accept:application/json”指明期望接收Json数据。

```
static HttpClient getHttpClient()
{
    var client = new HttpClient();
    client.BaseAddress = new Uri("http://localhost:12623");
    //设定HTTP Header，指定期望接收JSON字符串
    client.DefaultRequestHeaders.Add("Accept", "application/json ");
    return client;
}
```

```
static async Task TestGetFromWebServer()
{
    Console.WriteLine("向Web Server发出HTTP GET请求：\n");
    var client = getHttpClient();

    Console.WriteLine("\n收到服务器发回的数据：");
    string result = await client.GetStringAsync("api/myservice");
    Console.WriteLine(result);
}
```

HttpClient组件提供了一组异步方法，可以发出HTTP请求，其返回值则封装了从Web Server返回的数据

服务器响应HTTP请求

指明本方法只响应HTTP Get请求

```
[HttpGet]  
public IActionResult Get()  
{  
    Response.StatusCode = 200;  
    return Json(new MyClass()  
    {  
        Info = "Web Server发回的Json格式信息",  
        Id = 100  
    });  
}
```

设定响应码

将MyClass对象序列化为Json字符串
之后，再返回给客户端。

向Web Server发出HTTP GET请求。

收到服务器发回的数据：

{"id":100,"info":"Web Server发回的Json格式信息"}

测试查询字符串（客户端代码）

客户端发出包容查询字符串的HTTP请求

```
static async Task TestSearch()
{
    var client = getHttpClient();

    Console.WriteLine("HttpClient访问Web API服务器，通过查询字符串搜索数据。 \n");

    string searchWhat = "中国人 abc";
    string searchUrl = $"api/myservice/search?value={searchWhat}";

    string result = await client.GetStringAsync(searchUrl);

    Console.WriteLine($"已经将“{searchUrl}”发给服务器.....");

    Console.WriteLine($" \n收到服务器发回的数据:{result}");
}
```

测试查询字符串（服务端代码）

```
[HttpGet]  
[Route("[action]")]  
0 references  
public IActionResult Search(string value)  
{  
    return Json(new  
    {  
        message = "Web API Server : 您尝试着搜索\"" + value + "\""  
    });  
}
```

服务端参数value值，从URL中的查询字符串中按名字提取



HttpClient访问Web API服务器，通过查询字符串搜索数据。

已经将“api/myservice/search?value=中国人 abc”发给服务器……

收到服务器发回的数据：“message”:”Web API Server: 您尝试着搜索“中国人 abc””

添加定义HTTP Header

```
static async void TestHttpHeader()
{
    Console.WriteLine("测试向HTTP请求中添加自定义的Header。");
    var client = getHttpClient();
    client.DefaultRequestHeaders.Add("my-header", "my-header value");
    Console.WriteLine("Server端返回：");
    string result = await client.GetStringAsync("api/myservice/headers");
    Console.WriteLine(result);
}
```

客户端使用HttpClient向Web Server发送请求，添加了一个名为“my-header”的自定义HTTP Header。

```
[HttpGet]
[Route("headers")]
0 references
public IActionResult ShowHeaders()
{
    return Ok(Request.Headers);
}
```

Server端可以通过Request对象的Headers属性，提取出所有HTTP Header值。

Server端返回：

```
{"connection":["Keep-Alive"], "accept":["application/json"], "host":["localhost:12623"],
"my-header":["my-header value"], "mS-ASPNETCORE-TOKEN":["2ce4514d-007e-49ad-ba2b-e13d11
2e0964"], "x-Original-Proto":["http"], "x-Original-For":["127.0.0.1:60834"]}
```

客户端向服务端Post数据

```
static async Task TestPost()
{
    var client = getHttpClient();

    Console.WriteLine("向服务器Post一个MyClass对象。 \n");

    var data = new MyClass()
    {
        Id = new Random().Next(),
        Info = "新对象创建于" + DateTime.Now.ToShortTimeString()
    };

    try
    {
        var response = await client.PostAsJsonAsync("api/myService", data);
        response.EnsureSuccessStatusCode();
        Console.WriteLine("服务器返回处理结果");
        var result = await response.Content.ReadAsStringAsync();
        Console.WriteLine(result);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}
```

HttpClient组件提供了现成的方法将一个对象序列化为Json再发送给Server

服务端接收Post上来信息

```
[HttpPost]  
0 references  
public IActionResult Post([FromBody]MyClass obj)  
{  
    return Ok(obj);  
}
```

注意服务端的参数，需要添加
[FromBody]，指明从HTTP请求的
Body中提取数据并实例化。



敲任意键进行下一个演示：

向服务器Post一个MyClass对象。

服务器返回处理结果

```
{"message": "收到了你发送过来的MyClass对象", "receivedObj": {"id": 261249435, "info": "新对象创建于14:43"}}
```