

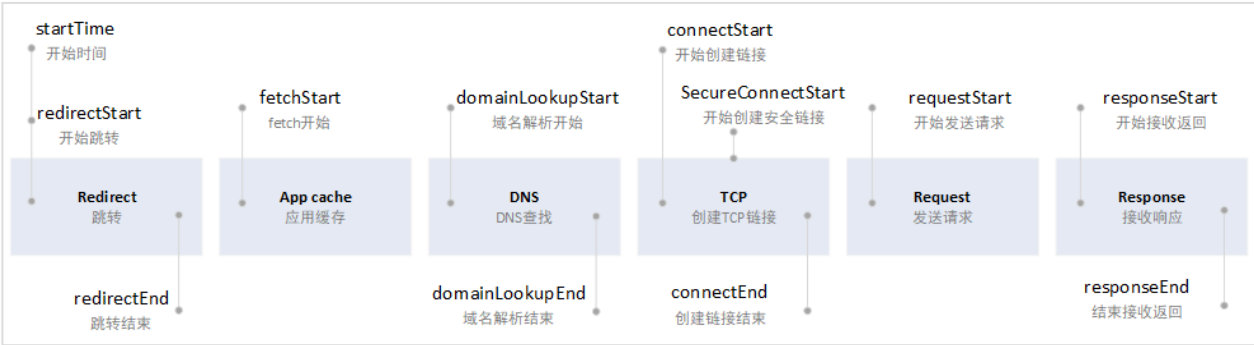


前端工程师，揭开HTTP的神秘面纱

📅 2018-07-03 | 📁 HTTP

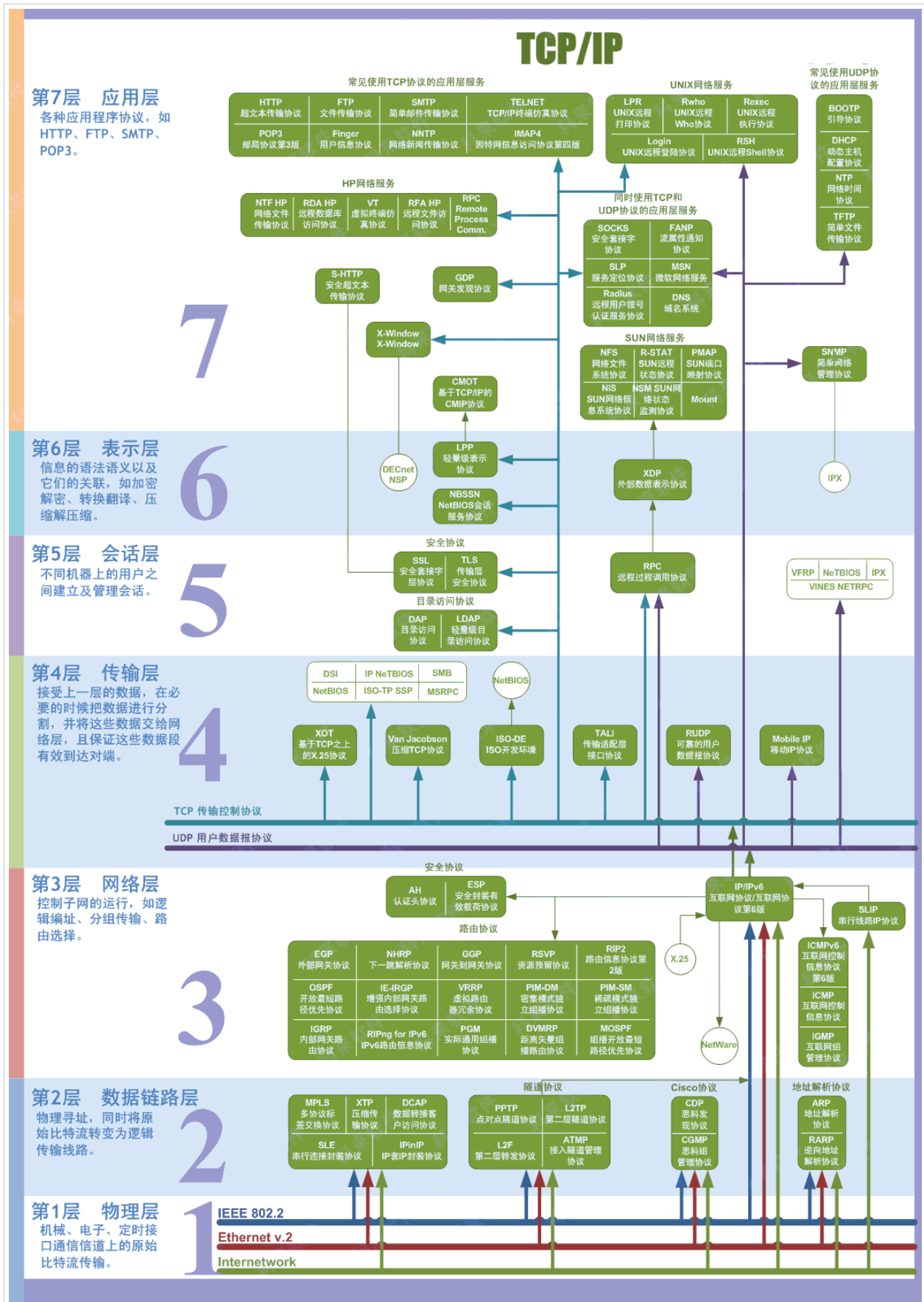
有关于网络协议、HTTP报文、跨域请求等http的基础知识的了解

浏览器输入URL后HTTP请求返回过程



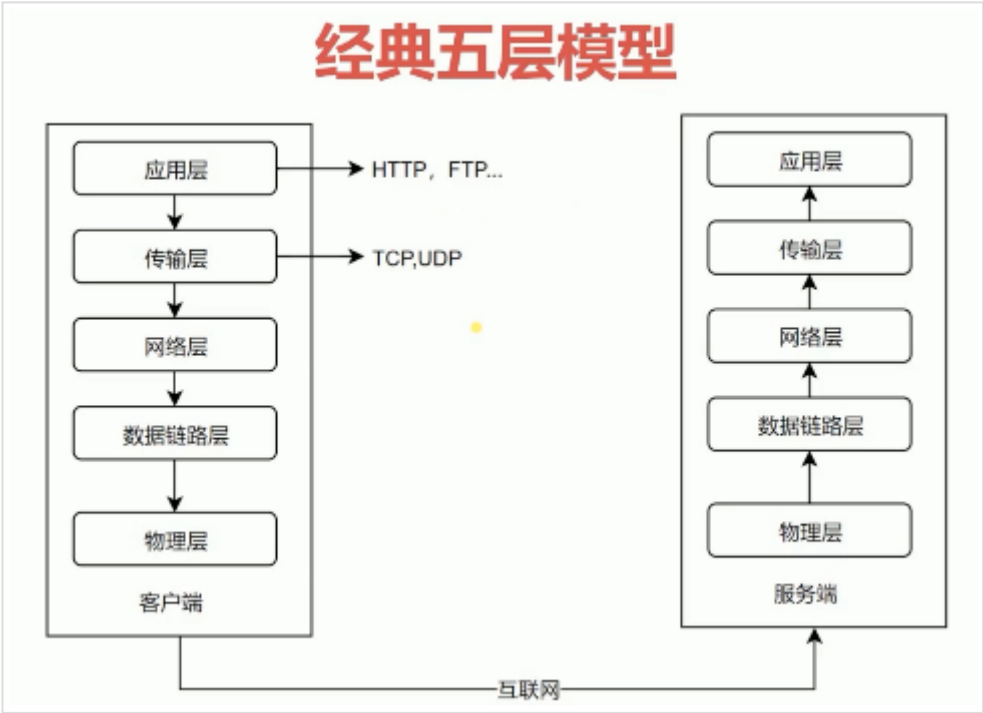
网络协议分层

OSI七层协议



五层协议

五层协议只是OSI和TCP/IP的综合，实际应用还是TCP/IP的四层结构。



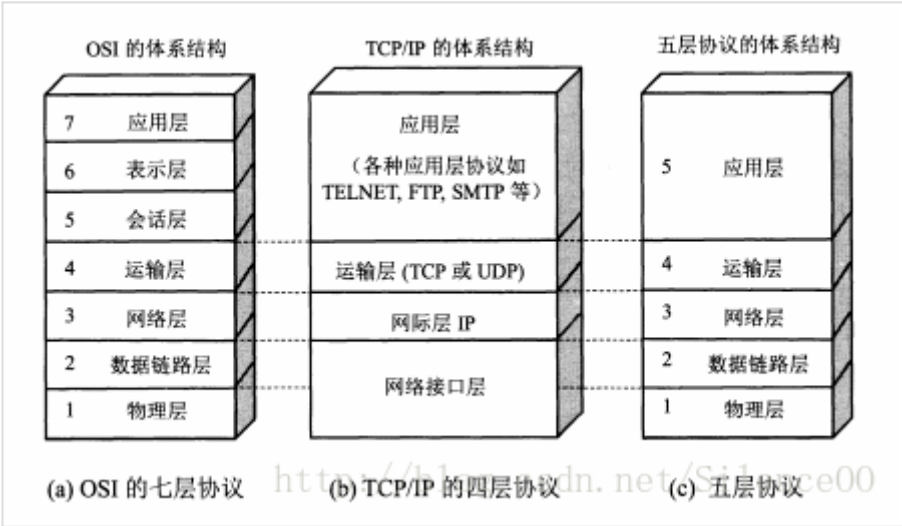
TCP/IP 协议

TCP(Transmission Control Protocol)传输控制协议

TCP/IP协议将应用层、表示层、会话层合并为应用层，物理层和数据链路层合并为网络接口层

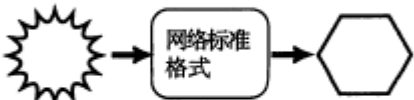
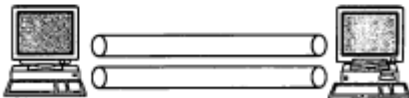
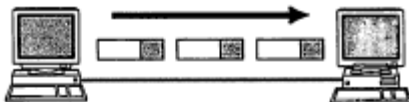

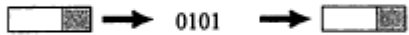
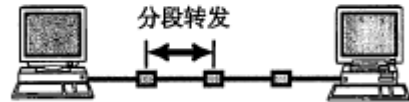
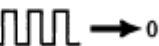

7 应用层	<应用层> TELNET, SSH, HTTP, SMTP, POP, SSL/TLS, FTP, MIME, HTML, SNMP, MIB, SIP, RTP ...
6 表示层	
5 会话层	
4 传输层	<传输层> TCP, UDP, UDP-Lite, SCTP, DCCP
3 网络层	<网络层> ARP, IPv4, IPv6, ICMP, IPsec
2 数据链路层	以太网、无线LAN、PPP..... (双绞线电缆、无线、光纤.....)
1 物理层	

三种模型结构



OSI七层网络模型	TCP/IP四层概念模型	对应网络协议
应用层（Application）	应用层	HTTP、TFTP、FTP、NFS、WAIS、SMTP
表示层（Presentation）		Telnet, Rlogin, SNMP, Gopher
会话层（Session）		SMTP, DNS
传输层（Transport）	传输层	TCP, UDP
网络层（Network）	网络层	IP, ICMP, ARP, RARP, AKP, UUCP
数据链路层（Data Link）	数据链路层	FDDI, Ethernet, Arpanet, PDN, SLIP, PPP
物理层（Physical）		IEEE 802.1A, IEEE 802.2到IEEE 802.11

各层的作用

	分层名称	功 能	每层功能概览
7	应用层	针对特定应用的协议。	针对每个应用的协议 电子邮件 ↔ 电子邮件协议 远程登录 ↔ 远程登录协议 文件传输 ↔ 文件传输协议
6	表示层	设备固有数据格式和网络标准数据格式的转换。	 接收不同表现形式的信息，如文字流、图像、声音等
5	会话层	通信管理。负责建立和断开通信连接（数据流动的逻辑通路）。管理传输层以下的分层。	何时建立连接，何时断开连接以及保持多久的连接？ 
4	传输层	管理两个节点之间的数据传输。负责可靠传输（确保数据被可靠地传送到目标地址）。	是否有数据丢失？ 
3	网络层	地址管理与路由选择。	经过哪个路由传递到目标地址？ 
2	数据链路层	互连设备之间传送和识别数据帧。	 数据帧与比特流之间的转换  分段转发
1	物理层	以“0”、“1”代表电压的高低、灯光的闪灭。界定连接器和网线的规格。	0101 →  → 0101 比特流与电子信号之间的切换  连接器与网线的规格

1. 物理层：

主要定义物理设备标准，如网线的接口类型、光纤的接口类型、各种传输介质的传输速率等。它的主要作用是传输比特流（就是由1、0转化为电流强弱来进行传输,到达目的地后在转化为1、0，也就是我们常说的数模转换与模数转换）。这一层的数据叫做比特。

2. 数据链路层：

定义了如何让格式化数据以进行传输，以及如何让控制对物理介质的访问。这一层通常还提供错误检测和纠正，以确保数据的可靠传输。

3. 网络层：

在位于不同地理位置的网络中的两个主机系统之间提供连接和路径选择。Internet的发展使得从世界各站点访问信息的用户数大大增加，而网络层正是管理这种连接的层。

4. 传输层：

定义了一些传输数据的协议和端口号（WWW端口80等），如：

TCP（transmission control protocol - 传输控制协议，传输效率低，可靠性强，用于传输可靠性要求高，数据量大的数据）

UDP（user datagram protocol - 用户数据报协议，与TCP特性恰恰相反，用于传输可靠性要求不高，数据量小的数据，如QQ聊天数据就是通过这种方式传输的）。主要是将从下层接收的数据进行分段和传输，到达目的地址后再进行重组。常常把这一层数据叫做段。

5. 会话层：

通过运输层（端口号：传输端口与接收端口）建立数据传输的通路。主要在你的系统之间发起会话或者接受会话请求（设备之间需要互相认识可以是IP也可以是MAC或者是主机名）

6. 表示层：

可确保一个系统的应用层所发送的信息可以被另一个系统的应用层读取。例如，PC程序与另一台计算机进行通信，其中一台计算机使用扩展二一十进制交换码（EBCDIC），而另一台则使用美国信息交换标准码（ASCII）来表示相同的字符。如有必要，表示层会通过使用一种通格式来实现多种数据格式之间的转换。

7. 应用层：

是最靠近用户的OSI层。这一层为用户的应用程序（例如电子邮件、文件传输和终端仿真）提供网络服务。

HTTP 发展历史

HTTP/0.9

- ☑ 只有一个命令GET
- ☑ 响应类型: 仅 超文本
- ☑ 没有header等描述数据的信息
- ☑ 服务器发送完毕，就关闭TCP连接

HTTP/1.0

- ☑ 增加了很多命令（post HESD）
- ☑ 增加status code 和 header
- ☑ 多字符集支持、多部分发送、权限、缓存等
- ☑ 响应：不再只限于超文本 (Content-Type 头部提供了传输 HTML 之外文件的能力 – 如脚本、样式或媒体文件)

HTTP/1.1

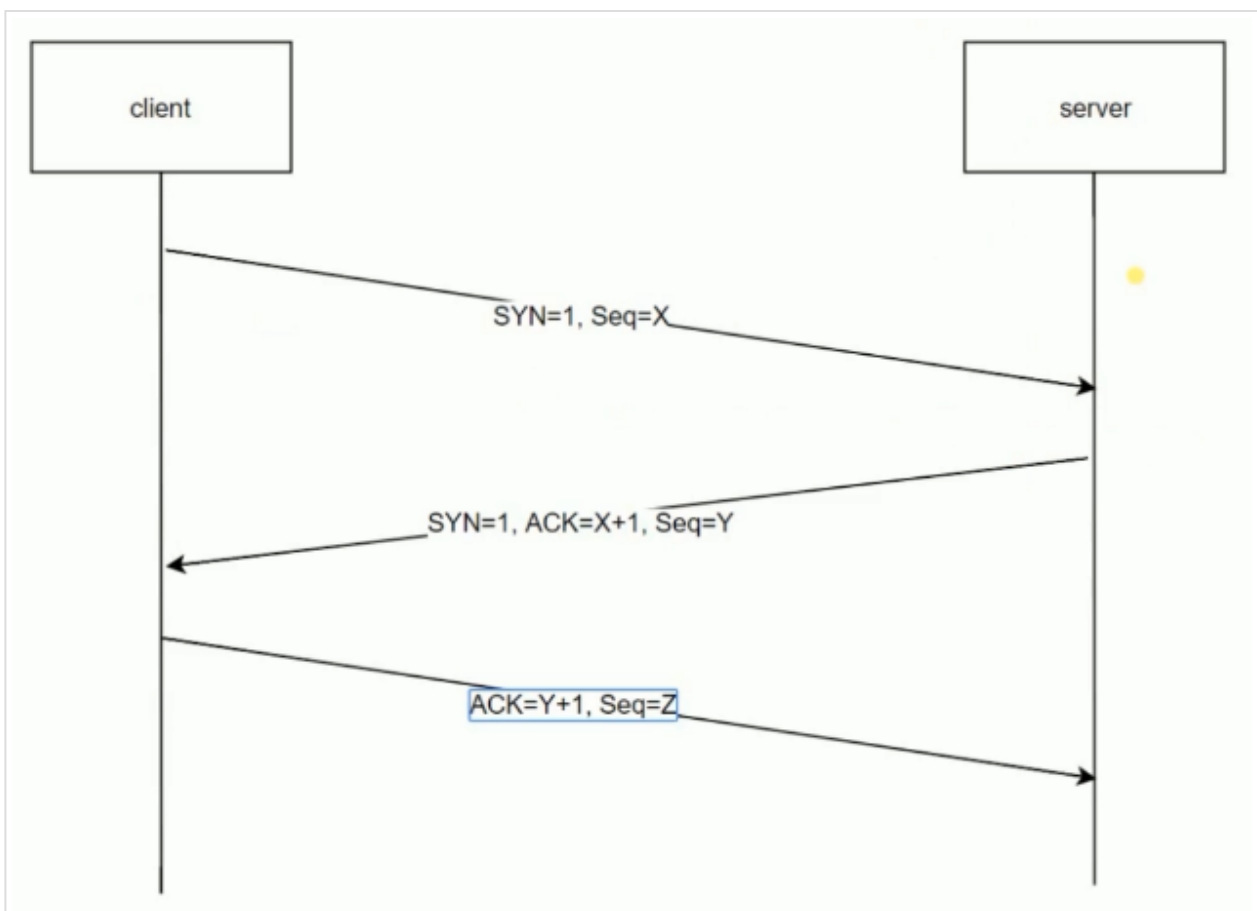
- ☑ 持久连接。TCP三次握手会在任何连接被建立之前发生一次。最终，当发送了所有数据之后，服务器发送一个消息，表示不会再有更多数据向客户端发送了；则客户端才会关闭连接（断开 TCP）
- ☑ 支持的方法: GET , HEAD , POST , PUT , DELETE , TRACE , OPTIONS
- ☑ 进行了重大的性能优化和特性增强，分块传输、压缩/解压、内容缓存磋商、虚拟主机（有单个IP地址的主机具有多个域名）、更快的响应，以及通过增加缓存节省了更多的带宽

HTTP2

- ☑ 所有数据以二进制传输。HTTP1.x是基于文本的，无法保证健壮性，HTTP2.0绝对使用新的二进制格式，方便且健壮
- ☑ 同一个连接里面发送多个请求不再需要按照顺序来
- ☑ 头信息压缩以及**推送**等提高效率的功能

三次握手

客户端和服务端在进行http请求和返回的工程中，需要创建一个 TCP connection（由客户端发起），http不存在连接这个概念，它只有请求和响应。请求和响应都是数据包，它们之间的传输通道就是 TCP connection。



位码即tcp标志位，有6种标示：SYN(synchronous建立联机) ACK(acknowledgement 确认) PSH(push传送) FIN(finish结束) RST(reset重置) URG(urgent紧急) Sequence number(顺序号码) Acknowledge number(确认号码)

第一次握手：主机A发送位码为syn = 1，随机产生seq number=1234567的数据包到服务器，主机B由SYN=1知道，A要求建立联机；

第二次握手：主机B收到请求后要确认联机信息，向A发送ack number=(主机A的seq+1)，syn=1，ack=1，随机产生seq=7654321的包；

第三次握手：主机A收到后检查ack number是否正确，即第一次发送的seq number+1，以及位码ack是否为1，若正确，主机A会再发送ack number=(主机B的seq+1)，ack=1，主机B收到后确认seq值与ack=1则连接建立成功

URI、URL、URN

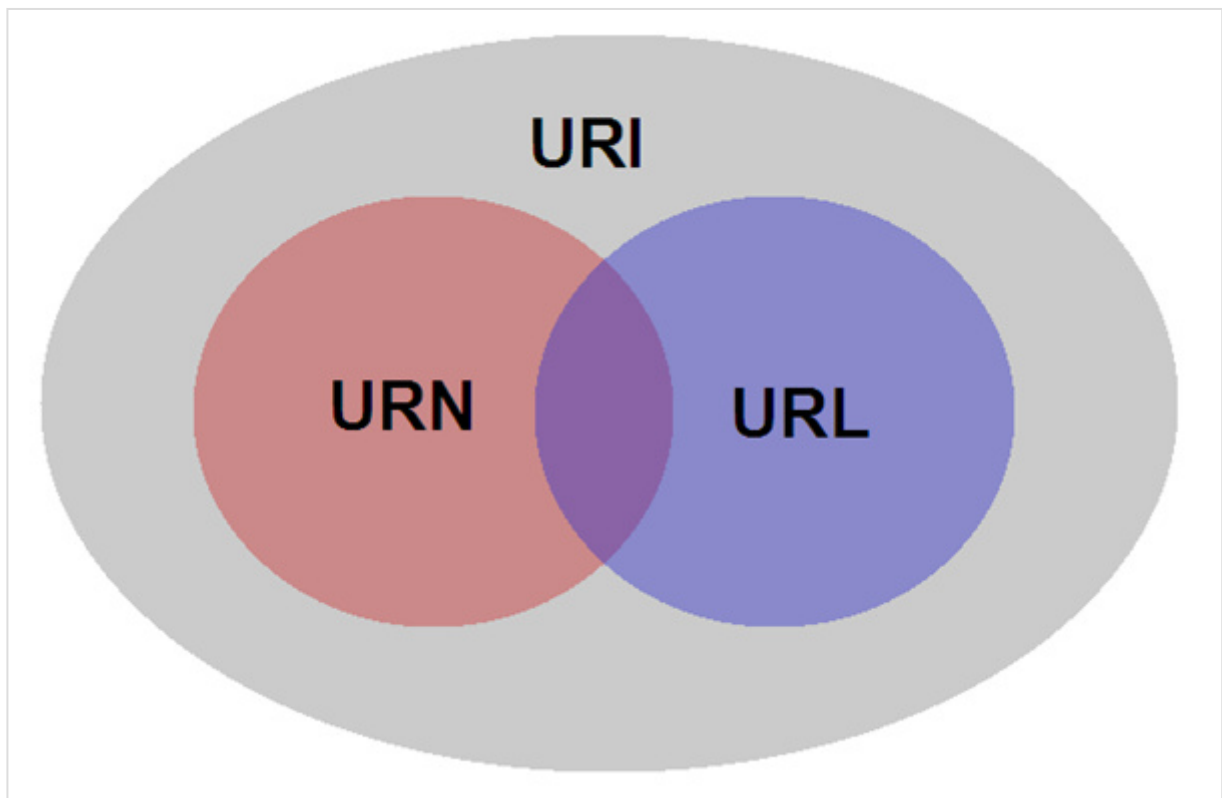
URI: Uniform Resource Identifier/统一资源标识符

URL: Uniform Resource Locator/统一资源定位器

URN: Uniform Resource Name/永久统一资源定位符

web上的各种资源（html、图片、视频、音频等）都由一个URI标识定位。URI相当于它们的详细“家庭住址”。

URI包含了URL和URN。



URL是URI的一种，不仅标识了Web 资源，还指定了操作或者获取方式，同时指出了主要访问机制和网络位置。

URN是URI的一种，用特定命名空间的名字标识资源。使用URN可以在不知道其网络位置及访问方式的情况下讨论资源。

网上的一个例子：

```
1 // 这是一个URI
2 http://bitpoetry.io/posts/hello.html#intro
3
```



```
-
4 // 资源访问方式
5 http://
6
7 // 资源存储位置
8 bitpoetry.io/posts/hello.html
9
10 #intro // 资源
11
12 // URL
13 http://bitpoetry.io/posts/hello.html
14
15 // URN
16 bitpoetry.io/posts/hello.html#intro
```

HTTP报文

请求报文：



响应报文：



HTTP 各种特性

curl

curl命令是一个利用URL规则在命令行下工作的文件传输工具。它支持文件的上传和下载，所以是综合传输工具，但按传统，习惯称curl为下载工具。作为一款强力工具，curl支持包括HTTP、HTTPS、ftp等众多协议，还支持POST、cookies、认证、从指定偏移处下载部分文件、用户代理字符串、限速、文件大小、进度条等特征。做网页处理流程和数据检索自动化，curl可以祝一臂之力。

更详细的CURL，[点这里](#)。

curl 访问 baidu.com :

```
FinGet@XMKT4W4TK6VHVII MINGW64 ~/Desktop
$ curl baidu.com
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
100   81  100   81    0     0    648      0  --:--:-- --:--:-- --:--:--   648<html>
<meta http-equiv="refresh" content="0;url=http://www.baidu.com/">
</html>
```

返回的内容中，html部分只有一个meta标签，<meta http-equiv="refresh"

content="0;url=http://www.baidu.com/">，这是因为我们访问的是 baidu.com，在浏览器中，浏览器会自动解析这个meta标签并重定向到 http://www.baidu.com/，然而命令行中并没有解析的功能。

curl 访问 www.baidu.com :

```
FinGet@XMKT4W4TK6VHVII MINGW64 ~/Desktop
$ curl www.baidu.com
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
100 2381  100 2381    0     0 25329      0  --:--:-- --:--:-- --:--:-- 25329<!DOCTYPE html>
<!--STATUS OK--><html> <head><meta http-equiv=content-type content=text/html; charset=utf-8><meta http-equiv=X-UA-Compatible content=IE=Edge><meta content=always name=referrer><link rel=stylesheet type=text/css href=http://s1.bdstatic.com/r/www/cache/bdorz/baidu.min.css><title>百度一下，你就知道</title></head>
<body link=#0000cc <div id=wrapper> <div id=head> <div class=head-wrapper> <div class=s_form> <div class=s_form_wrapper> <div id=lg> <img hidefocus=true src=//www.baidu.com/img/bd_logo1.png width=270 height=129> </div> <form id=form name=f action=//www.baidu.com/s class=fm> <input type=hidden name=bdorz_come value=1> <input type=hidden name=ie value=utf-8> <input type=hidden name=f value=8> <input type=hidden name=rsv_bp value=1> <input type=hidden name=rsv_idx value=1> <input type=hidden name=tn value=baidu> <span class="bg s_ipt_wr"> <input id=kwd name=wd class=s_ipt value maxlength=255 autocomplete=off autofocus> </span> <span class="bg s_btn_wr"> <input type=submit id=su value=百度一下 class="bg s_btn"> </span> </form> </div> <div id=u1> <a href=http://news.baidu.com name=tj_trnews class=mnav>新闻</a> <a href=http://www.hao123.com name=tj_trhao123 class=mnav>hao123</a> <a href=http://map.baidu.com name=tj_trmap class=mnav>地图</a> <a href=http://v.baidu.com name=tj_trvideo class=mnav>视频</a> <a href=http://tieba.baidu.com name=tj_trtieba class=mnav>贴吧</a> <noscript> <a href=http://www.baidu.com/bdorz/login.gif?login&tpl=mn&u=http%3A%2F%2Fwww.baidu.com%2F%3Fbdorz_come%3D1 name=tj_login class=lb>登录</a> </noscript> <script>document.write(' <a href="http://www.baidu.com/bdorz/login.gif?login&tpl=mn&u='+ encodeURIComponent(window.location.href+ (window.location.search === "" ? "?" : "&")+ "bdorz_come=1")+' name="tj_login" class="lb">登录</a>'); </script> <a href=//www.baidu.com/more/ name=tj_briicon class=bri style="display: block;">更多产品</a> </div> </div> <div id=ftCon> <div id=ftConw> <p id=1h> <a href=http://home.baidu.com>关于百度</a> <a href=http://ir.baidu.com>About Baidu</a> </p> <p id=cp> &copy; 2017 &nbsp; Baidu &nbsp; <a href=http://www.baidu.com/duty/>使用百度前必读</a> &nbsp; <a href=http://jiayi.baidu.com/ class=cp-feedback>意见反馈</a> &nbsp; &京ICP证030173号 &nbsp; <img src=//www.baidu.com/img/g.gif> </p> </div> </div> </div> </body> </html>
```

curl常用命令

-v 显示详细的请求信息

```

FinGet@XMKT4W4TK6VHVII MINGW64 ~/Desktop
$ curl -v www.baidu.com
* STATE: INIT => CONNECT handle 0x487e150; line 1392 (connection #-5000)
* Rebuilt URL to: www.baidu.com/
* Added connection 0. The cache now contains 1 members
* STATE: CONNECT => WAITRESOLVE handle 0x487e150; line 1428 (connection #0)
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
  0   0   0    0    0    0     0      0  --:--:-- --:--:-- --:--:--    0*   Trying 180.97.33.108.
..
* TCP_NODELAY set
* STATE: WAITRESOLVE => WAITCONNECT handle 0x487e150; line 1509 (connection #0)
* Connected to www.baidu.com (180.97.33.108) port 80 (#0)
* STATE: WAITCONNECT => SENDPROTOCONNECT handle 0x487e150; line 1561 (connection #0)
* Marked for [keep alive]: HTTP default
* STATE: SENDPROTOCONNECT => DO handle 0x487e150; line 1579 (connection #0)
> GET / HTTP/1.1
> Host: www.baidu.com
> User-Agent: curl/7.58.0
> Accept: */*
>
* STATE: DO => DO_DONE handle 0x487e150; line 1658 (connection #0)
* STATE: DO_DONE => WAITPERFORM handle 0x487e150; line 1783 (connection #0)
* STATE: WAITPERFORM => PERFORM handle 0x487e150; line 1799 (connection #0)
* HTTP 1.1 or later with persistent connection, pipelining supported
< HTTP/1.1 200 OK
< Accept-Ranges: bytes
< Cache-Control: private, no-cache, no-store, proxy-revalidate, no-transform
< Connection: Keep-Alive
< Content-Length: 2381
< Content-Type: text/html
< Date: Tue, 03 Jul 2018 03:35:30 GMT
< Etag: "588604f8-94d"
< Last-Modified: Mon, 23 Jan 2017 13:28:24 GMT
< Pragma: no-cache
* Server bfe/1.0.8.18 is not blacklisted
< Server: bfe/1.0.8.18
< Set-Cookie: BDORZ=27315; max-age=86400; domain=.baidu.com; path=/
<
{ [2381 bytes data]
* STATE: PERFORM => DONE handle 0x487e150; line 1968 (connection #0)
* multi_done
100 2381 100 2381 0 0 30525 0 --:--:-- --:--:-- --:--:-- 30525<!DOCTYPE html>
<!--STATUS OK--><html> <head><meta http-equiv=content-type content=text/html; charset=utf-8><meta http-equiv=X-UA-Compatible content=IE=Edge><meta content=always name=referrer><link rel=stylesheet type=text/css href=http://s1.bdstatic.com/r/www/cache/bdorz/baidu.min.css><title>百度一下, 你就知道</title></head>
> <body link=#0000cc> <div id=wrapper> <div id=head> <div class=head_wrapper> <div class=s_form> <div class=s_form_wrapper> <div id=lg> <img hidefocus=true src=//www.baidu.com/img/bd_logo1.png width=270 height=129> </div> <form id=form name=f action=//www.baidu.com/s class=fm> <input type=hidden name=bdorz_come value=1> <input type=hidden name=ie value=utf-8> <input type=hidden name=f value=8> <input type=hid

```

-X 指定请求方式

- 1 curl -X GET www.xxxx.com/xx/xx?xx=123
- 2
- 3 curl -X POST www.xxxx.com/xx/xx?xx=123

-o / -O 保存下载的文件

- 1 // 将文件下载到本地并命名为mygettext.html
- 2 curl -o mygettext.html http://www.gnu.org/software/gettext/manual/gettext.html
- 3
- 4 // 将文件保存到本地并命名为gettext.html
- 5 curl -O http://www.gnu.org/software/gettext/manual/gettext.html

CORS跨域请求的限制与解决

```
1 // server1.js
2 const http = require('http')
3 const fs = require('fs')
4
5 http.createServer(function (request, response) {
6   console.log('request come', request.url)
7
8   const html = fs.readFileSync('test.html', 'utf8')
9   response.writeHead(200, {
10     'Content-Type': 'text/html'
11   })
12   response.end(html)
13 }).listen(8888)
14
15 console.log('server listening on 8888')
```



```
1 // server2.js
2 const http = require('http')
3
4 http.createServer(function (request, response) {
5   console.log('request come', request.url)
6
7   response.end('123')
8 }).listen(8887)
9
10 console.log('server listening on 8887')
```



```
1 // test.html
2 <!DOCTYPE html>
3 <html lang="en">
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <meta http-equiv="X-UA-Compatible" content="ie=edge">
8   <title>Document</title>
9 </head>
10 <body>
11
12 </body>
13 <script>
14   fetch('http://127.0.0.1:8887');
15 </script>
16 </html>
```

❌ Failed to load <http://127.0.0.1:8887/>: No 'Access-Control-Allow-Origin' header is [\(index\):1](#) present on the requested resource. Origin '<http://localhost:8888>' is therefore not allowed access.

处理方法：

1.服务器端处理

```
1 // server2.js 服务器端设置允许跨域
2 response.writeHead(200, {
3   'Access-Control-Allow-Origin': '*' // * 表示任何域名下都可以访问这个服务,也可以指定域
4 })
```

2.jsonp

```
1 // test.html
2 <script src="http://127.0.0.1:8887"></script>
```

就算存在跨域，请求还是会发送，响应也会返回，只是浏览器端发现了存在跨域问题就将返回内容屏蔽了，并报错提示。

CORS 预请求

```
1 // test.html
2 <script>
3   fetch('http://127.0.0.1:8887',{
4     method: 'post',
5     headers: {
6       'X-Test-Cors': '123'
7     }
8   });
9 </script>
```

```
✖ Failed to load http://localhost:8887/: Request header field X-Test-Cors is not allowed by Access-Control-Allow-Headers in preflight response. (index):1
✖ Uncaught (in promise) TypeError: Failed to fetch (index):1
```

我们设置的请求头中 X-Test-Cors 在跨域请求的时候，不被允许。

虽然不允许跨域，但是请求仍然会发送，并返回成功。

× Headers Preview Response Timing

▼ General

Request URL: http://localhost:8887/
Request Method: OPTIONS
Status Code: ● 200 OK
Remote Address: [::1]:8887
Referrer Policy: no-referrer-when-downgrade

▼ Response Headers view source

Access-Control-Allow-Origin: *
Connection: keep-alive
Date: Sat, 19 May 2018 07:50:46 GMT
Transfer-Encoding: chunked

▼ Request Headers view source

Accept: */*

默认允许的请求方法：

- GET
- HEAD
- POST

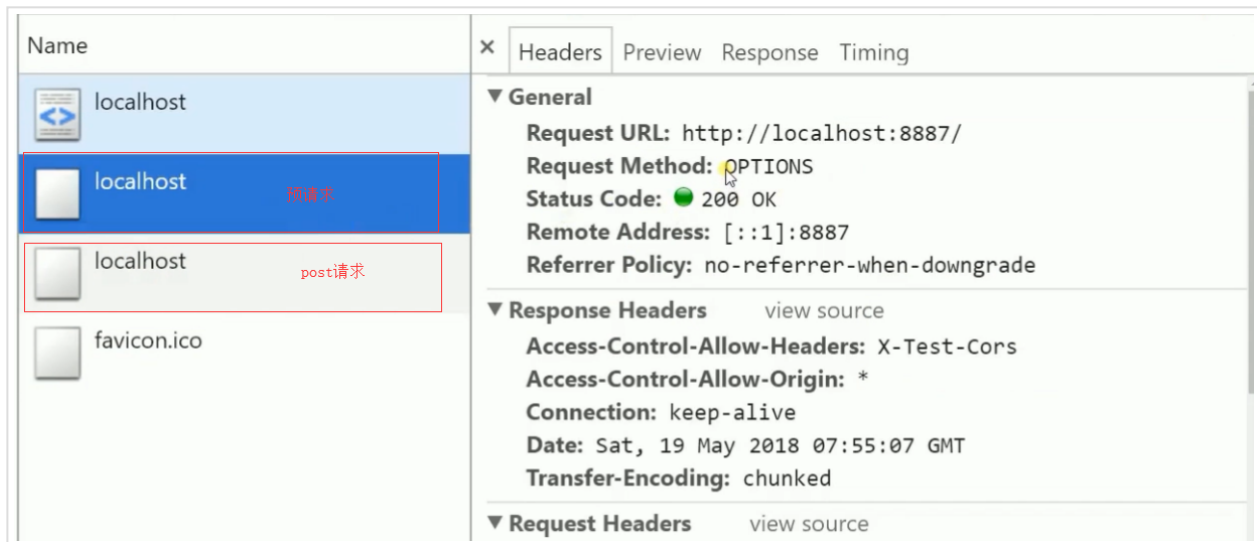
其他的方法(PUT、DELETE)都需要预请求验证的。

默认允许的 Content-Type :

- text/plain
- multipart/form-data
- application/x-www-form-urlencoded

怎样设置允许我们设置的请求头：

```
1 // server2.js
2 response.writeHead(200, {
3   'Access-Control-Allow-Origin': '*',
4   'Access-Control-Allow-Headers': 'X-Test-Cors' // 加上这个设置
5 })
```

首先发送一个预请求，预请求就是告诉浏览器接下来要发送的post请求是被允许的。

设置允许的请求方法：

```
1 // server2.js
2 response.writeHead(200, {
3   'Access-Control-Allow-Origin': '*',
4   'Access-Control-Allow-Headers': 'X-Test-Cors',
5   'Access-Control-Allow-Methods': 'POST, PUT, DELETE'
6 })
```

设置一个安全时间：

```
1 // server2.js
2 response.writeHead(200, {
3   'Access-Control-Allow-Origin': '*',
4   'Access-Control-Allow-Headers': 'X-Test-Cors',
5   'Access-Control-Allow-Methods': 'POST, PUT, DELETE',
6   'Access-Control-Max-Age': '1000'
7 })
```

Access-Control-Max-Age 的单位是秒，意思就是在多少秒以内，我们设置的这些允许的请求头，请求方法，是不需要发送预请求验证的，直接就可以通过，并发送。

缓存Cache-Control

常用值：

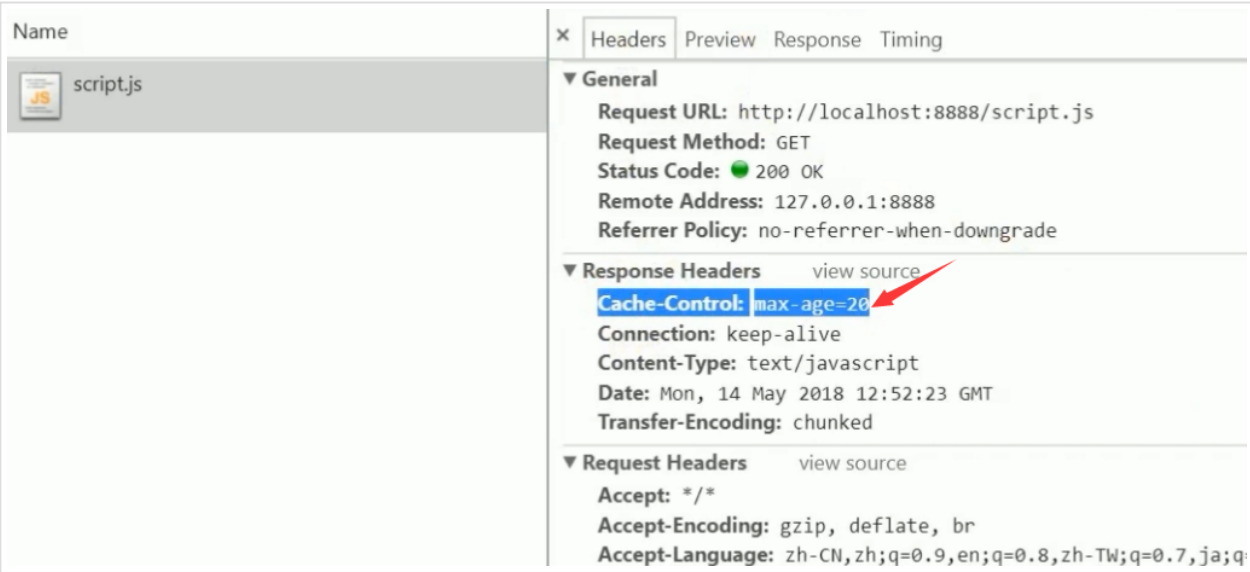
Cache-Control	说明
public	所有内容都将被缓存(客户端和代理服务器都可缓存)
private	内容只缓存到私有缓存中(仅客户端可以缓存，代理服务器不可缓存)
no-cache	必须先与服务器确认返回的响应是否被更改，然后才能使用该响应来满足后续对同一个网址的请求。因此，如果存在合适的验证令牌 (ETag)，no-cache 会发起往返通信来验证缓存的响应，如果资源未被更改，可以避免下载。
no-store	所有内容都不会被缓存到缓存或 Internet 临时文件中
must-revalidation/proxy-revalidation	如果缓存的内容失效，请求必须发送到服务器/代理以进行重新验证
max-age=xxx (xxx is numeric)	缓存的内容将在 xxx 秒后失效, 这个选项只在HTTP 1.1可用, 并如果和Last-Modified一起使用时, 优先级较高

```
1 // server.js
2 const http = require('http')
3 const fs = require('fs')
4
5 http.createServer(function (request, response) {
6   console.log('request come', request.url)
7
8   if (request.url === '/') {
9     const html = fs.readFileSync('test.html', 'utf8')
10    response.writeHead(200, {
11      'Content-Type': 'text/html'
12    })
13    response.end(html)
14  }
15
16  if (request.url === '/script.js') {
17    response.writeHead(200, {
18      'Content-Type': 'text/javascript',
19      'Cache-Control': 'max-age=20,public' // 缓存20s 多个值用逗号分开
20    })
21    response.end('console.log("script loaded")')
22  }
23 }).listen(8888)
24
25 console.log('server listening on 8888')
```



```
1 // test.html
2 <!DOCTYPE html>
3 <html lang="en">
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <meta http-equiv="X-UA-Compatible" content="ie=edge">
8   <title>Document</title>
9 </head>
10 <body>
```

```
11
12   </body>
13   <script src="/script.js"></script>
14 </html>
```

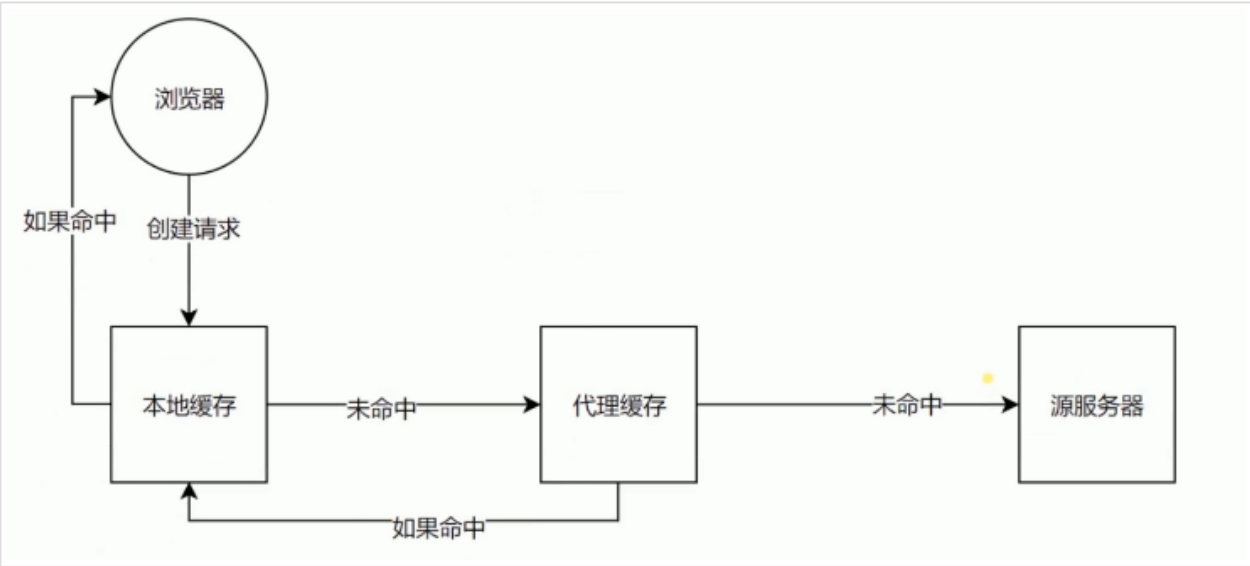


刷新会发现 script.js 是从缓存中获取的，请求时间也是0。

Name	Status	Type	Initiator	Size	Time	Waterfall	500.00 r
script.js	200 OK	script	(index) Parser	(from me...)	0 ms		
				(from memory cache)	0 ms		

我们希望浏览器缓存我们的图片，文件、js代码，但是服务器端代码更新了，浏览器端还是在缓存中获取的旧的文件。这就诞生了，webpack打包中出现的文件名后加上hash值，当文件改变时hash值也改变，这样浏览器就会发送新的请求到服务器端。

缓存验证



验证头:

- Last-Modified

上次修改时间

配合If-Modified-Since或者If-Unmodified-Since使用

对比上次修改时间以验证资源是否需要更新

- Etag

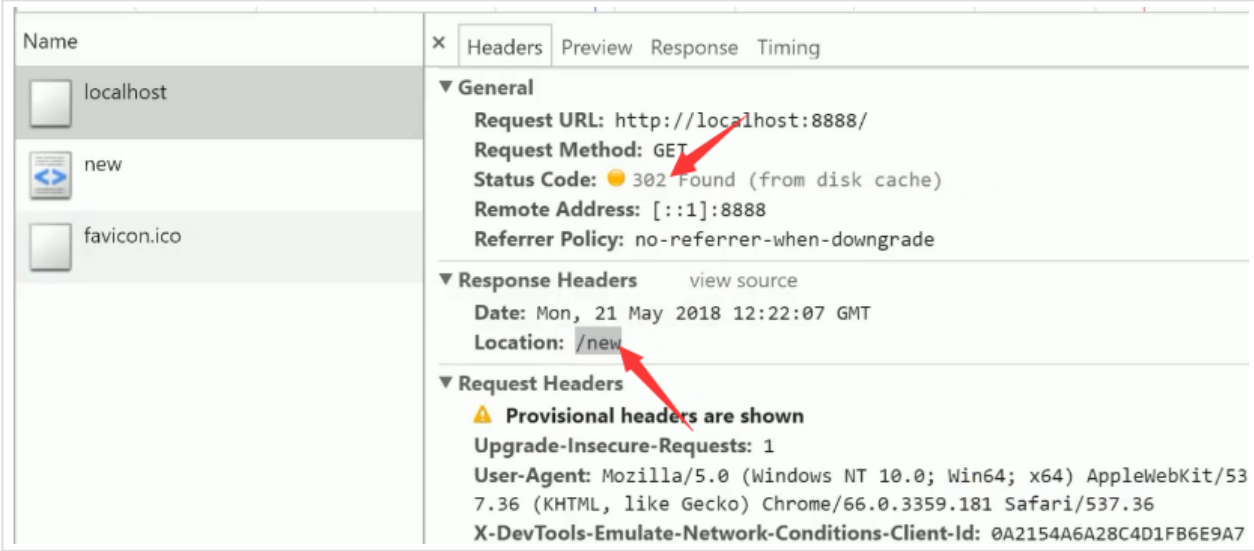
数据签名(内容修改, 签名就会改变)

配合If-Match或者If-Non-Match使用

对比资源的签名判断是否使用缓存

Redirect

```
1  const http = require('http')
2
3  http.createServer(function (request, response) {
4    console.log('request come', request.url)
5
6    if (request.url === '/') {
7      response.writeHead(302, { // or 301
8        'Location': '/new'
9      })
10     response.end()
11   }
12   if (request.url === '/new') {
13     response.writeHead(200, {
14       'Content-Type': 'text/html',
15     })
16     response.end('<div>this is content</div>')
17   }
18 }).listen(8888)
19
20 console.log('server listening on 8888')
```



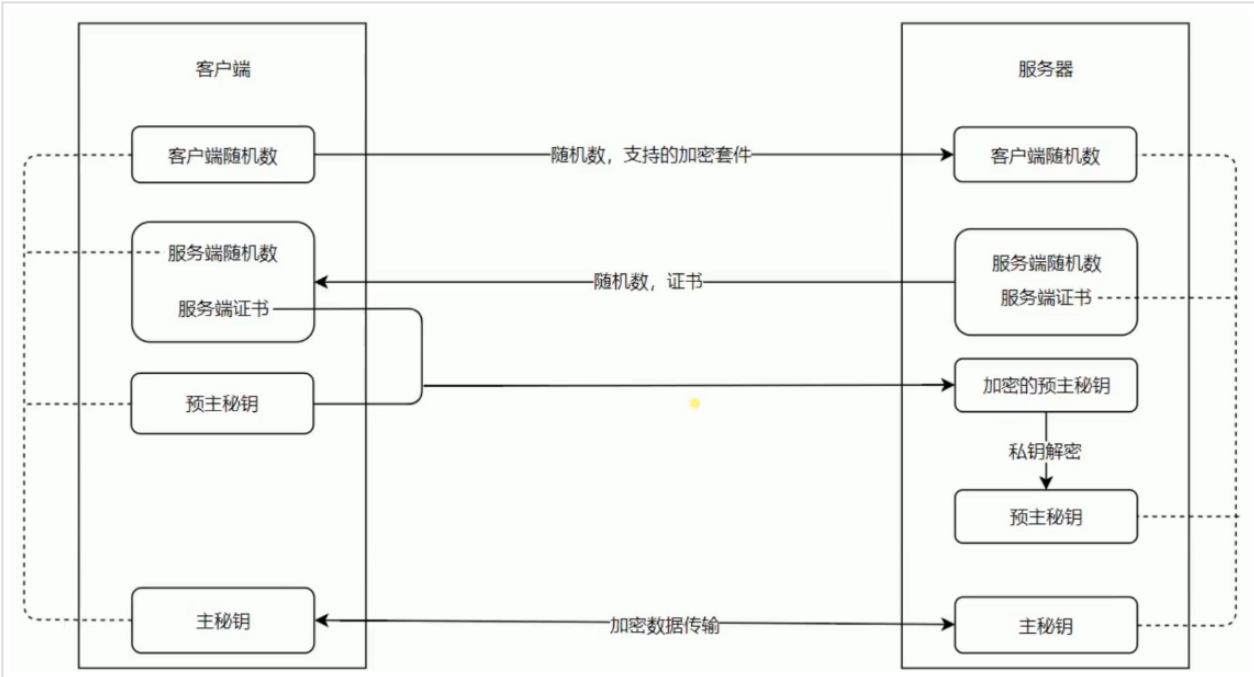
302临时跳转，301永久跳转，301从缓存种获取跳转，使用301之后，主动权就掌握在用户手里，如果用户不清理缓存，那就算服务器端改变了也没用。

Name	Status	Proto...	Type	Initiator	Size	Time	Connection ID	Waterfall
localhost	301 Move...	http/1.1		Other	(from disk...	41 ms 5 ms	0	
new	200 OK	http/1.1	docu...	(index) Redir...	170 B 26 B	35 ms 22 ms	8821	
favicon.ico	(pend...			Other	0 B 0 B	Pending	0	

Content Security Policy (网页安全政策)

阮一峰:Content Security Policy 入门教程

HTTPS



HTTPS和HTTP的区别主要为以下四点：

一、https协议需要到ca申请证书，一般免费证书很少，需要交费。


二、http是超文本传输协议，信息是明文传输，https 则是具有安全性的ssl加密传输协议。

三、http和https使用的是完全不同的连接方式，用的端口也不一样，前者是80，后者是443。

四、http的连接很简单，是无状态的；HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，比http协议安全。


最后

创建了一个前端学习交流群，感兴趣的朋友，一起来嗨呀！




H5前端


扫一扫二维码，加入该群。




微信小程序入门与实战 常用组件API开发技巧项目实战
初级 ▲ 13620 评价: 9.84分
学会使用小程序开发，一个使用组件与电影于一体的小程序实战。




React.js入门基础与案例开发
初级 ▲ 2557 评价: 9.48分
轻松入门 React 开发，React Router 4 与 Webpack 2 搭建开发环境。




全网稀缺Vue 2.0高级实战 独立开发专属音乐WebAPP
高级 ▲ 3261 评价: 9.95分
独一无二的Vue 2.0高级组件化开发实战课程，达到高级T4水平。




Vue.js 高仿饿了么外卖APP 2016 最火前端框架
高级 ▲ 6378 评价: 9.75分
【中阶】vue.js搭配angular.js和react.js模式，并解释它们的优缺点。




微信小程序商城构建全栈应用
中阶 ▲ 2425 评价: 9.82分
前后端分离，RESTful API标准接口+微信支付，掌握多平台的开发。




腾讯大厂亲授 Web 前后端架构分析与搭建技巧
中阶 ▲ 399 评价: 9.89分
借鉴每一行代码安全基础，突破前端开发瓶颈，全栈进阶。




ES6零基础教学 解析彩蛋项目
中阶 ▲ 1753 评价: 9.50分
ES6从零基础，量身设计的进阶课程，让你全面掌握ES6。




Node.js入门到企业Web开发中的应用
初级 ▲ 818 评价: 9.84分
Node.js在Web应用开发中的一个福音，特别适合于中小型系统的快速开发。




Get全栈技能点 Vue2.0/Node.js/MongoDB 打造商城系统
中阶 ▲ 1672 评价: 9.88分
【中阶实战】从前端入门全栈，让你快速掌握全栈。




开发微信全家桶项目Vue/Node/MongoDB高级技术栈全案
高级 ▲ 792 评价: 8.28分
Node/Vue SSR + Koa2 跨域开发微信公众号+小程序。




Node.js入门到企业Web开发中的应用
初级 ▲ 818 评价: 9.84分
Node.js在Web应用开发中的一个福音，特别适合于中小型系统的快速开发。



Redux+React Router+Node.js 全栈开发
高级 ▲ 1355 评价: 9.56分
全网唯一的React 15+Redux+React Router4 实战课程，学到手是你的资本。



四大堆栈解锁 Webpack 前端工程化
中阶 ▲ 664 评价: 10.00分
前端开发必备，灵活运用Webpack的使用可以极大的提高开发效率。



微信小程序入门与实战 搭建朋友圈
初级 ▲ 1631 评价: 9.84分
官方同步，第一时间掌握最新微信小程序开发核心技术。



全网首发mpvue小程序全栈开发
中阶 ▲ 821 评价: 9.87分
最新网络首发 学习mpvue+Koa+vue 全栈开发小程序。

更多教程资料，
加群@群主

如果我的文章对您有帮助！有钱的捧个钱场，没钱的捧个人场，谢谢您！

打赏

HTTP

Vue 全家桶，深入Vue 的世界

昵称

邮箱

网址(http://)


加上昵称更好玩。ヾ(≥▽≤)o来啊，快活啊!

Emoji | Preview

M↓

回复

4 评论




Anonymous

Chrome 67.0.3396.99 Windows 10.0

2018-07-14

回复

Click on expand




测试

Chrome 65.0.3325.181 Windows 10.0

2018-07-10

测试测试测试

回复



Anonymous


Chrome 67.0.3396.99 Windows 10.0

2018-07-06

整理的很不错，值得学习，我也是7/15满一年，大三实习。。。哈哈，经历类似，竟然连hexo的主题都一样！

醉了，不过我没有向你那么勤快，没有写那么多，应该向你学习！

回复



FinGet

Chrome 65.0.3325.181 Windows 10.0

2018-07-06

回复

@Anonymous

你博客放出来看看 😊



Anonymous

Chrome 67.0.3396.99

Windows 10.0

2018-07-06

回复

@FinGet

还没整好呢，5月份搭建的，6月份赶项目 🤖，
写了两篇笔记丢在那里，哪有你那么厉害呀。



Anonymous

Chrome 67.0.3396.99

Windows 10.0

2018-07-06

回复

@FinGet

请问一下：整理+撰写出来一篇，你大概需要多长时间呀？



FinGet

Chrome 65.0.3325.181

Windows 10.0

2018-07-06

回复

@Anonymous

我一般都是边学边记录（markdown），学完也就差不多能发布了。时间有长有短，至少都要两三天，长的一两周都有。毕竟上班还有工作要做。



王冰

Chrome 67.0.3396.99

Mac OS 10.12.3

2018-07-04

回复

感觉挺不错的，怎么人这么少呢



FinGet

Chrome 65.0.3325.181

Windows 10.0

2018-07-04

回复

@王冰

不知道朋友是从哪里找到这篇文章的

© 2018  FinGet—前端我一直在路上

 本站访客数: 4438 |  本站总访问量: 13921次