

HTTP 和 HTTPS

HTTP 的基本概念

http: 是互联网上应用最为广泛的一种网络协议，是一个客户端和服务端**请求和应答的标准（TCP）**，用于从 WWW 服务器传输超文本到本地浏览器的**超文本传输协议**。

HTTP工作原理

HTTP协议定义Web客户端如何从Web服务器请求Web页面，以及服务器如何把Web页面传送给客户端。客户端向服务器发送一个请求报文，服务器以一个状态行作为响应。

HTTP请求/响应的步骤

- 1.客户端连接到Web服务器
- 2.发送HTTP请求
- 3.服务器接受请求并返回HTTP响应
- 4.释放TCP连接
- 5.客户端（浏览器）解析HTML内容

记忆口诀：连接发送加响应，释放解析整过程。

HTTP 的 5 种方法

- GET---获取资源
- POST---传输资源
- PUT---更新资源
- DELETE---删除资源
- HEAD---获取报文首部

GET与POST的区别

1.浏览器回退表现不同 GET在浏览器回退时是无害的，而POST会再次提交请求 **2.浏览器对请求地址的处理不同** GET请求地址会被浏览器主动缓存，而POST不会，除非手动设置 **3.浏览器对响应的处理不同** GET请求参数会被完整的保留在浏览器历史记录里，而POST中的参数不会被保留 **4.参数大小不同**. GET请求在URL中传送的参数是有长度的限制，而POST没有限制 **5.安全性不同**. GET参数通过URL传递，会暴露，不安全；POST放在Request Body中，相对更安全 **6.针对数据操作的类型不同**.GET对数据进行查询，POST主要对数据进行增删改！简单说，GET是只读，POST是写。

HTTP报文的组成成分

请求报文{ 请求行、请求头、空行、请求体 } 请求行：{http方法、页面地址、http协议、http版本} 响应报文{ 状态行、响应头、空行、响应体 }

Request Header:

1. **GET /sample.Jsp HTTP/1.1** //请求行
2. **Host:** www.uuid.online/ //请求的目标域名和端口号

3. **Origin:** http://localhost:8081/ //请求的来源域名和端口号（跨域请求时，浏览器会自动带上这个头信息）
4. **Referer:** https://localhost:8081/link?query=xxxxx //请求资源的完整URI
5. **User-Agent:** Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/67.0.3396.99 Safari/537.36 //浏览器信息
6. **Cookie:** BAIDUID=FA89F036:FG=1; BD_HOME=1; sugstore=0 //当前域名下的Cookie
7. **Accept:** text/html,image/apng //代表客户端希望接受的数据类型是html或者是png图片类型
8. **Accept-Encoding:** gzip, deflate //代表客户端能支持gzip和deflate格式的压缩
9. **Accept-Language:** zh-CN,zh;q=0.9 //代表客户端可以支持语言zh-CN或者zh(值得一提的是q(0~1)是优先级权重的意思，不写默认为1，这里zh-CN是1，zh是0.9)
10. **Connection:** keep-alive //告诉服务器，客户端需要的tcp连接是一个长连接

Response Header:

1. **HTTP/1.1 200 OK** // 响应状态行
2. **Date:** Mon, 30 Jul 2018 02:50:55 GMT //服务端发送资源时的服务器时间
3. **Expires:** Wed, 31 Dec 1969 23:59:59 GMT //比较过时的一种验证缓存的方式，与浏览器（客户端）的时间比较，超过这个时间就不用缓存（不和服务进行验证），适合版本比较稳定的网页
4. **Cache-Control:** no-cache // 现在最多使用的控制缓存的方式，会和服务进行缓存验证，具体见博文“Cache-Control”
5. **etag:** "fb8ba2f80b1d324bb997cbe188f28187-ssl-df" // 一般是Nginx静态服务器发来的静态文件签名，浏览在没有“Disabled cache”情况下，接收到etag后，同一个url第二次请求就会自动带上“If-None-Match”
6. **Last-Modified:** Fri, 27 Jul 2018 11:04:55 GMT //是服务器发来的当前资源最后一次修改的时间，下次请求时，如果服务器上当前资源的修改时间大于这个时间，就返回新的资源内容
7. **Content-Type:** text/html; charset=utf-8 //如果返回是流式的数据，我们就必须告诉浏览器这个头，不然浏览器会下载这个页面，同时告诉浏览器是utf8编码，否则可能出现乱码
8. **Content-Encoding:** gzip //告诉客户端，应该采用gzip对资源进行解码
9. **Connection:** keep-alive //告诉客户端服务器的tcp连接也是一个长连接

https 的基本概念

https:是以安全为目标的 HTTP 通道，即 HTTP 下 加入 SSL 层进行加密。

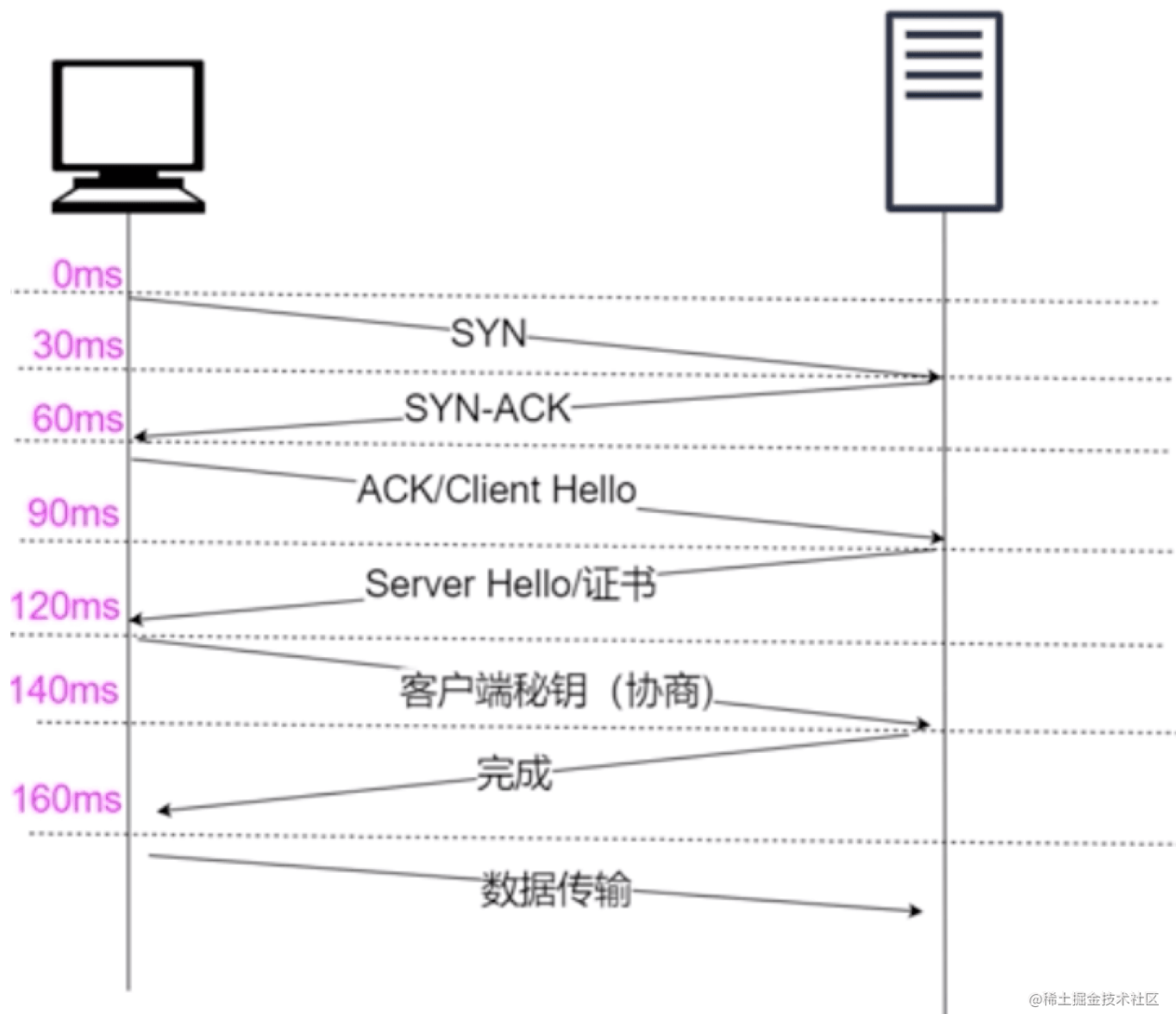
https 协议的作用：建立一个信息安全通道，来确保数据的传输，确保网站的真实性。

http 和 https 的区别？

- 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。
- Https 协议需要 ca 证书，费用较高。
- 使用不同的链接方式，端口也不同，一般，http 协议的端口为 80，https 的端口为 443。
- http 的连接很简单，是无状态的。

记忆口诀：明文传输超文本，安全等级各不同。CA证书费用高，无状连接端难同。

https 协议的工作原理



客户端在使用 HTTPS 方式与 Web 服务器通信时有以下几个步骤：

1. 客户端使用 https url 访问服务器，则要求 web 服务器建立 ssl 链接。
2. web 服务器接收到客户端的请求之后，会将网站的证书（证书中包含了公钥），传输给客户端。
3. 客户端和 web 服务器端开始协商 SSL 链接的安全等级，也就是加密等级。
4. 客户端浏览器通过双方协商一致的安全等级，建立会话密钥，然后通过网站的公钥来加密会话密钥，并传送给网站。
5. web 服务器通过自己的私钥解密出会话密钥。
6. web 服务器通过会话密钥加密与客户端之间的通信。

记忆口诀：一连二传三协商，四建五得六使用。

https 协议的优缺点

- HTTPS 协议要比 http 协议安全，可防止数据在传输过程中被窃取、改变，确保数据的完整性。
- https 握手阶段比较费时，会使页面加载时间延长 50%，增加 10%~20%的耗电。
- https 缓存不如 http 高效，会增加数据开销。
- SSL 证书也需要钱，功能越强大的证书费用越高。
- SSL 证书需要绑定 IP，不能再同一个 ip 上绑定多个域名，ipv4 资源支持不了这种消耗。

TCP/IP模型是互联网的基础，它是一系列网络协议的总称。这些协议可以划分为四层，分别为链路层、网络层、传输层和应用层。

- 链路层：负责封装和解封装IP报文，发送和接受ARP/RARP报文等。
- 网络层：负责路由以及把分组报文发送给目标网络或主机。
- 传输层：负责对报文进行分组和重组，并以TCP或UDP协议格式封装报文。
- 应用层：负责向用户提供应用程序，比如HTTP、FTP、Telnet、DNS、SMTP等。

OSI七层模型	TCP/IP概念层模型	功能	TCP/IP协议族
应用层	应用层	文件传输, 电子邮件, 文件服务, 虚拟终端	TFTP, HTTP, SNMP, FTP, SMTP, DNS, Telnet
表示层		数据格式化, 代码转换, 数据加密	没有协议
会话层		解除或建立与别的接点的联系	没有协议
传输层	传输层	提供端对端的接口	TCP, UDP
网络层	网络层	为数据包选择路由	IP, ICMP, RIP, OSPF, BGP, IGMP
数据链路层	链路层	传输有地址的帧以及错误检测功能	SLIP, CSLIP, PPP, ARP, RARP, MTU
物理层		以二进制数据形式在物理媒体上传输数据	ISO2110, IEEE802.1, IEEE802.2

The diagram illustrates the analogy between a human conversation and a TCP connection establishment. On the left, a conversation between Alice and Bob is shown. Alice asks, '你好! 我想和你聊一聊?' (Hello! I want to chat with you?). Bob responds, '好的, 我有空!' (Okay, I have time!). Alice then says, '好的, 我们开始吧' (Okay, let's start), which is accompanied by a red dot and a blue dot, symbolizing the start of data exchange. On the right, the corresponding TCP handshake is shown between a '客户端' (Client) and a '服务端' (Server). The client initiates with 'connect()' and 'SYN_SEND'. The server responds with 'listen()' and 'SYN-RECEIVED'. The client then sends 'ESTABLISHED'. The server responds with 'SYN-ACK'. The client then sends 'ACK'. Finally, the server sends 'ESTABLISHED'. A dashed arrow at the bottom indicates '开始交换数据' (Start exchanging data).

```
sequenceDiagram
    participant Alice
    participant Bob
    participant Client as 客户端
    participant Server as 服务端

    Alice->>Bob: 你好! 我想和你聊一聊?
    Bob-->>Alice: 好的, 我有空!
    Alice->>Bob: 好的, 我们开始吧
    Client->>Server: connect()
    Client->>Server: SYN_SEND
    Server->>Client: listen()
    Server->>Client: SYN-RECEIVED
    Client->>Server: ESTABLISHED
    Server->>Client: SYN-ACK
    Client->>Server: ACK
    Server->>Client: ESTABLISHED
    Note over Client, Server: 开始交换数据
```

1. 第一次握手：建立连接时，客户端发送syn包（syn=j）到服务器，并进入SYN_SENT状态，等待服务器确认；SYN：同步序列编号（Synchronize Sequence Numbers）。
2. 第二次握手：服务器收到syn包并确认客户的SYN（ack=j+1），同时也发送一个自己的SYN包（syn=k），即SYN+ACK包，此时服务器进入SYN_RECV状态；
3. 第三次握手：客户端收到服务器的SYN+ACK包，向服务器发送确认包ACK（ack=k+1），此包发送完毕，客户端和服务器进入ESTABLISHED（TCP连接成功）状态，完成三次握手。

握手过程中传送的包里不包含数据，三次握手完毕后，客户端与服务器才正式开始传送数据。

TCP 四次挥手

1. **客户端进程发出连接释放报文**，并且停止发送数据。释放数据报文首部，FIN=1，其序列号为seq=u（等于前面已经传送过来的数据的最后一个字节的序号加1），此时，客户端进入FIN-WAIT-1（终止等待1）状态。TCP规定，FIN报文段即使不携带数据，也要消耗一个序号。
- 2) **服务器收到连接释放报文**，**发出确认报文**，ACK=1，ack=u+1，并且带上自己的序列号seq=v，此时，服务端就进入了CLOSE-WAIT（关闭等待）状态。TCP服务器通知高层的应用进程，客户端向服务器的方向就释放了，这时候处于半关闭状态，即客户端已经没有数据要发送了，但是服务器若发送数据，客户端依然要接受。这个状态还要持续一段时间，也就是整个CLOSE-WAIT状态持续的时间。
- 3) 客户端收到服务器的确认请求后，此时，客户端就进入FIN-WAIT-2（终止等待2）状态，等待服务器发送连接释放报文（在这之前还需要接受服务器发送的最后的数 据）。
- 4) **服务器将最后的数据发送完毕后**，**就向客户端发送连接释放报文**，FIN=1，ack=u+1，由于在半关闭状态，服务器很可能又发送了一些数据，假定此时的序列号为seq=w，此时，服务器就进入了LAST-ACK（最后确认）状态，等待客户端的确认。
- 5) **客户端收到服务器的连接释放报文后**，**必须发出确认**，ACK=1，ack=w+1，而自己的序列号是seq=u+1，此时，客户端就进入了TIME-WAIT（时间等待）状态。注意此时TCP连接还没有释放，必须经过2**MSL（最长报文段寿命）的时间后，当客户端撤销相应的TCB后，才进入CLOSED状态。
- 6) 服务器只要收到了客户端发出的确认，立即进入CLOSED状态。同样，撤销TCB后，就结束了这次的TCP连接。可以看到，服务器结束TCP连接的时间要比客户端早一些。

TCP和UDP的区别

1. TCP是面向**连接**的，而UDP是面向无连接的。
2. TCP仅支持**单播传输**，UDP 提供了单播，多播，广播的功能。
3. TCP的三次握手保证了连接的**可靠性**；UDP是无连接的、不可靠的一种数据传输协议，首先不可靠性体现在无连接上，通信都不需要建立连接，对接收到的数据也不发送确认信号，发送端不知道数据是否会正确接收。
4. UDP的**头部开销**比TCP的更小，**数据传输速率更高**，**实时性更好**。

HTTP 请求跨域问题

1. 跨域的原理

跨域，是指浏览器不能执行其他网站的脚本。它是由浏览器的**同源策略**造成的。

同源策略是浏览器对 JavaScript 实施的安全限制，只要**协议**、**域名**、**端口**有任何一个不同，都被当作是不同的域。

跨域原理，即是通过各种方式，**避开浏览器的安全限制**。

2. 解决方案

最初做项目的时候，使用的是jsonp，但存在一些问题，使用get请求不安全，携带数据较小，后来也用过了iframe，但只有主域相同才行，也是存在些问题，后来通过了解和学习发现使用代理和proxy代理配合起来使用比较方便，就引导后台按这种方式做下服务器配置，在开发中使用proxy，在服务器上使用nginx代理，这样开发过程中彼此都方便，效率也高；现在h5新特性还有 window.postMessage()

- **JSONP:**

ajax 请求受同源策略影响，不允许进行跨域请求，而 script 标签 src 属性中的链接却可以访问跨域的 js 脚本，利用这个特性，服务端不再返回 JSON 格式的数据，而是返回一段调用某个函数的 js 代码，在 src 中进行了调用，这样实现了跨域。

步骤：

1. 去创建一个script标签
2. script的src属性设置接口地址
3. 接口参数，必须要带一个自定义函数名，要不然后台无法返回数据
4. 通过定义函数名去接受返回的数据

```
//动态创建 script
var script = document.createElement('script');

// 设置回调函数
function getData(data) {
    console.log(data);
}

//设置 script 的 src 属性，并设置请求地址
script.src = 'http://localhost:3000/?callback=getData';

// 让 script 生效
document.body.appendChild(script);
```

JSONP 的缺点:

JSON 只支持 get，因为 script 标签只能使用 get 请求；JSONP 需要后端配合返回指定格式的数据。

- **document.domain** 基础域名相同 子域名不同
- **window.name** 利用在一个浏览器窗口内，载入所有的域名都是共享一个 window.name
- **CORS** CORS(Cross-origin resource sharing)跨域资源共享 服务器设置对CORS的支持原理：服务器设置Access-Control-Allow-Origin HTTP响应头之后，浏览器将会允许跨域请求
- **proxy代理** 目前常用方式
- **window.postMessage()** 利用h5新特性 window.postMessage()
- **Websocket**

Cookie、sessionStorage、localStorage 的区别

相同点：

- 存储在客户端

不同点：

- cookie数据大小不能超过4k；sessionStorage和localStorage的存储比cookie大得多，可以达到5M+
- cookie设置的过期时间之前一直有效；localStorage永久存储，浏览器关闭后数据不丢失除非主动删除数据；sessionStorage数据在当前浏览器窗口关闭后自动删除
- cookie的数据会自动的传递到服务器；sessionStorage和localStorage数据保存在本地

HTTP状态码及常见状态码**HTTP状态码**

- 1xx：指示信息类，表示请求已接受，继续处理
- 2xx：指示成功类，表示请求已成功接受
- 3xx：指示重定向，表示要完成请求必须进行更进一步的的操作
- 4xx：指示客户端错误，请求有语法错误或请求无法实现
- 5xx：指示服务器错误，服务器未能实现合法的请求

常见状态码

- 200 OK：客户端请求成功
- 301 Moved Permanently：所请求的页面已经永久重定向至新的URL
- 302 Found：所请求的页面已经临时重定向至新的URL
- 304 Not Modified 未修改。
- 403 Forbidden：对请求页面的访问被禁止
- 404 Not Found：请求资源不存在
- 500 Internal Server Error：服务器发生不可预期的错误原来缓冲的文档还可以继续使用
- 503 Server Unavailable：请求未完成，服务器临时过载或宕机，一段时间后可恢复正常
- 1xx（临时响应）表示临时响应并需要请求者继续执行操作的状态码
 - 100 - 继续 请求者应当继续提出请求。服务器返回此代码表示已收到请求的第一部分，正在等待其余部分
 - 101 - 切换协议 请求者已要求服务器切换协议，服务器已确认并准备切换
- 2xx（成功）表示成功处理了请求的状态码
 - 200 - 成功 服务器已经成功处理了请求。通常，这表示服务器提供了请求的网页
 - 201 - 已创建 请求成功并且服务器创建了新的资源
 - 202 - 已接受 服务器已接受请求，但尚未处理
 - 203 - 非授权信息 服务器已经成功处理了请求，但返回的信息可能来自另一来源
 - 204 - 无内容 服务器成功处理了请求，但没有返回任何内容

- 205 - 重置内容 服务器成功处理了请求，但没有返回任何内容
- 3xx (重定向) 表示要完成请求，需要进一步操作；通常，这些状态代码用来重定向
 - 300 - 多种选择 针对请求，服务器可执行多种操作。服务器可根据请求者 (user agent) 选择一项操作，或提供操作列表供请求者选择
 - 301 - 永久移动 请求的网页已永久移动到新位置。服务器返回此响应 (对GET或HEAD请求的响应) 时，会自动将请求者转到新位置
 - 302 - 临时移动 服务器目前从不同位置的网页响应请求，但请求者应继续使用原有位置来进行以后的请求
 - 303 - 查看其它位置 请求者应当对不同的位置使用单独的GET请求来检索响应时，服务器返回此代码
 - 304 - 未修改 自上次请求后，请求的网页未修改过。服务器返回此响应，不会返回网页的内容
 - 305 - 使用代理 请求者只能使用代理访问请求的网页。如果服务器返回此响应，还表示请求者应使用代理
 - 307 - 临时性重定向 服务器目前从不同位置的网页响应请求，但请求者应继续使用原有的位置来进行以后的请求
- 4xx (请求错误) 这些状态码表示请求可能出错，妨碍了服务器的处理
 - 400 - 错误请求 服务器不理解请求的语法
 - 401 - 未授权 请求要求身份验证。对于需要登录的网页，服务器可能返回此响应
 - 403 - 禁止 服务器拒绝请求
 - 404 - 未找到 服务器找不到请求的网页
 - 405 - 方法禁用 禁用请求中指定的方法
 - 406 - 不接受 无法使用请求的内容特性响应请求的网页
 - 407 - 需要代理授权 此状态码与401 (未授权) 类似，但指定请求者应当授权使用代理
 - 408 - 请求超时 服务器等候请求时发生超时
 - 410 - 已删除 如果请求的资源已永久删除，服务器就会返回此响应
 - 413 - 请求实体过大 服务器无法处理请求，因为请求实体过大，超出了服务器的处理能力
 - 414 - 请求的URI过长 请求的URI (通常为网址) 过长，服务器无法处理
- 5xx (服务器错误) 这些状态码表示服务器在尝试处理请求时发生内部错误。这些错误可能是服务器本身的错误，而不是请求出错
 - 500 - 服务器内部错误 服务器遇到错误，无法完成请求
 - 501 - 尚未实施 服务器不具备完成请求的功能。例如，服务器无法识别请求方法时可能会返回此代码
 - 502 - 错误网关 服务器作为网关或代理，从上游服务器无法收到无效响应
 - 503 - 服务器不可用 服务器目前无法使用 (由于超载或者停机维护)。通常，这只是暂时状态
 - 504 - 网关超时 服务器作为网关代理，但是没有及时从上游服务器收到请求
 - 505 - HTTP版本不受支持 服务器不支持请求中所用的HTTP协议版本

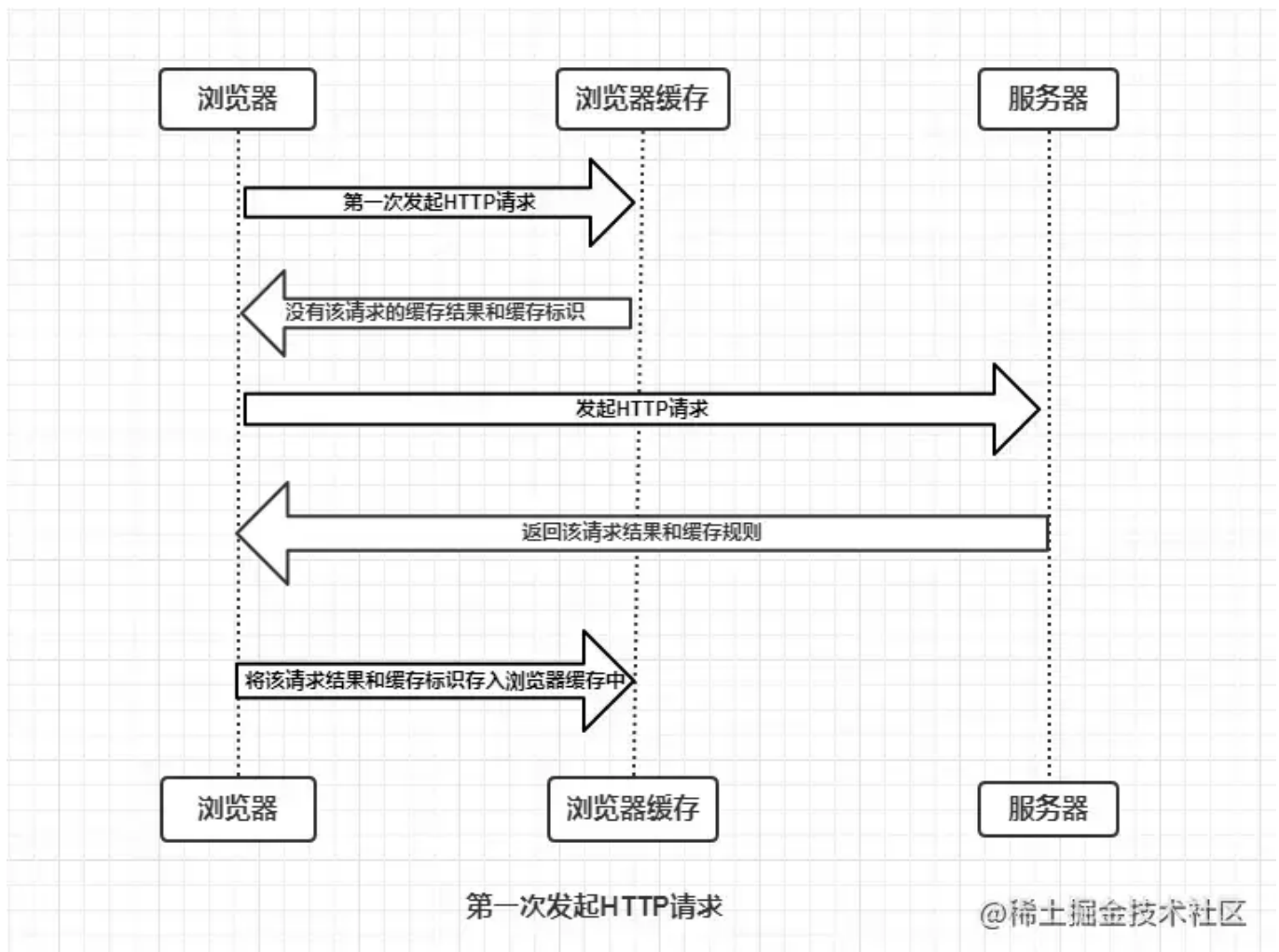
介绍下304过程

- a. 浏览器请求资源时首先命中资源的Expires 和 Cache-Control, Expires 受限与本地时间，如果修改了本地时间，可能会造成缓存失效，可以通过Cache-control: max-age指定最大生命周期，状态仍然返回200，但不会请求数据，在浏览器中能明显看到from cache字样。

- b. 强缓存失效，进入协商缓存阶段，首先验证ETagETag可以保证每一个资源是唯一的，资源变化都会导致ETag变化。服务器根据客户端上送的If-None-Match值来判断是否命中缓存。
- c. 协商缓存Last-Modify/If-Modify-Since阶段，客户端第一次请求资源时，服务器返回的header中会加上Last-Modify，Last-modify是一个时间标识该资源的最后修改时间。再次请求该资源时，request的请求头中会包含If-Modify-Since，该值为缓存之前返回的Last-Modify。服务器收到If-Modify-Since后，根据资源的最后修改时间判断是否命中缓存。

浏览器的缓存机制 强制缓存 && 协商缓存

浏览器与服务器通信的方式为应答模式，即是：浏览器发起HTTP请求 – 服务器响应该请求。那么浏览器第一次向服务器发起该请求后拿到请求结果，会根据响应报文中HTTP头的缓存标识，决定是否缓存结果，是则将请求结果和缓存标识存入浏览器缓存中，简单的过程如下图：



由上图我们可以知道：

- 浏览器每次发起请求，都会先在浏览器缓存中查找该请求的结果以及缓存标识
- 浏览器每次拿到返回的请求结果都会将该结果和缓存标识存入浏览器缓存中

以上两点结论就是浏览器缓存机制的关键，他确保了每个请求的缓存存入与读取，只要我们再理解浏览器缓存的使用规则，那么所有的问题就迎刃而解了。为了方便理解，这里根据是否需要向服务器重新发起HTTP请求将缓存过程分为两个部分，分别是强制缓存和协商缓存。

- 强制缓存

强制缓存就是向浏览器缓存查找该请求结果，并根据该结果的缓存规则来决定是否使用该缓存结果的过程。当浏览器向服务器发起请求时，服务器会将缓存规则放入HTTP响应报文的HTTP头中和请求结果一起返回给浏览器，控制强制缓存的字段分别是 **Expires** 和 **Cache-Control**，其中Cache-Control优先级比Expires高。

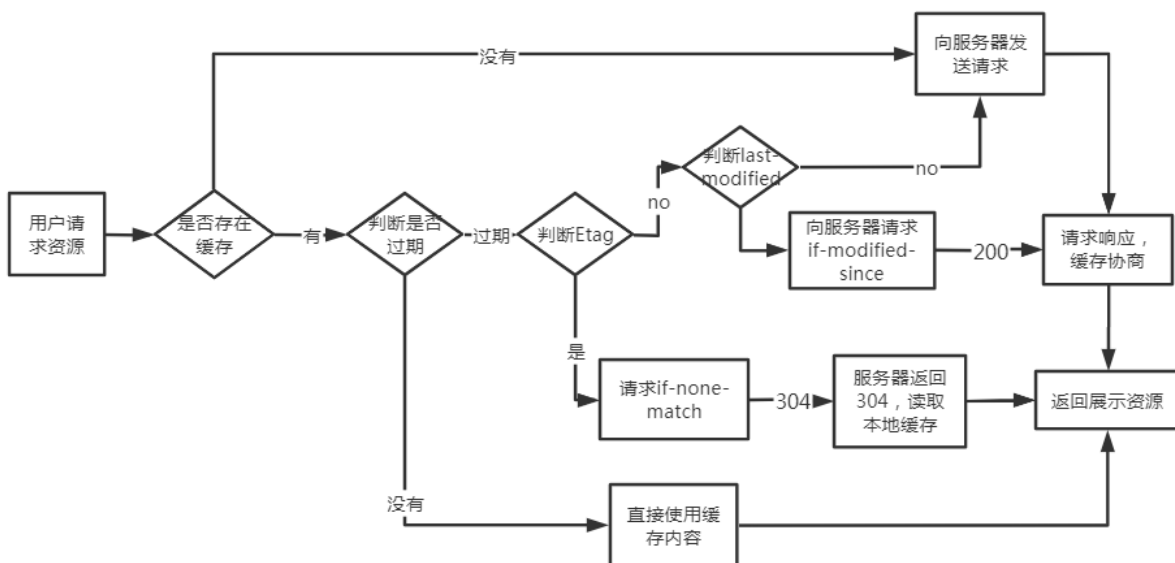
强制缓存的情况主要有三种(暂不分析协商缓存过程)，如下：

1. 不存在该缓存结果和缓存标识，强制缓存失效，则直接向服务器发起请求（跟第一次发起请求一致）。
2. 存在该缓存结果和缓存标识，但该结果已失效，强制缓存失效，则使用协商缓存。
3. 存在该缓存结果和缓存标识，且该结果尚未失效，强制缓存生效，直接返回该结果

• 协商缓存

协商缓存就是强制缓存失效后，浏览器携带缓存标识向服务器发起请求，由服务器根据缓存标识决定是否使用缓存的过程，同样，协商缓存的标识也是在响应报文的HTTP头中和请求结果一起返回给浏览器的，控制协商缓存的字段分别有：**Last-Modified / If-Modified-Since** 和 **Etag / If-None-Match**，其中Etag / If-None-Match的优先级比Last-Modified / If-Modified-Since高。协商缓存主要有以下两种情况：

1. 协商缓存生效，返回304
2. 协商缓存失效，返回200和请求结果结果



<https://segmentfault.com/a/1191000000000000> @稀土掘金技术社区

传送门 [# 彻底理解浏览器的缓存机制](#)

HTTP 请求跨域问题

1. 跨域的原理

跨域，是指浏览器不能执行其他网站的脚本。它是由浏览器的**同源策略**造成的。跨域访问是被各大浏览器所默认禁止的。

同源策略，是浏览器对 JavaScript 实施的安全限制，只要**协议**、**域名**、**端口**有任何一个不同，都被当作是不同的域。

跨域原理，即是通过各种方式，**避开浏览器的安全限制**。

2. 解决方案

最初做项目的时候，使用的是jsonp，但存在一些问题，使用get请求不安全，携带数据较小，后来也用过iframe，但只有主域相同才行，也是存在些问题，后来通过了解和学习发现使用代理和proxy代理配合起来使用比较方便，就引导后台按这种方式做下服务器配置，在开发中使用proxy，在服务器上使用nginx代理，这样开发过程中彼此都方便，效率也高；现在h5新特性还有 windows.postMessage()

◦ JSONP:

ajax 请求受同源策略影响，不允许进行跨域请求，而 script 标签 src 属性中的链接却可以访问跨域的 js 脚本，利用这个特性，服务端不再返回 JSON 格式的数据，而是 返回一段调用某个函数的 js 代码，在 src 中进行了调用，这样实现了跨域。

步骤：

1. 去创建一个script标签
2. script的src属性设置接口地址
3. 接口参数，必须要带一个自定义函数名，要不然后台无法返回数据
4. 通过定义函数名去接受返回的数据

```
//动态创建 script
var script = document.createElement('script');

// 设置回调函数
function getData(data) {
    console.log(data);
}

//设置 script 的 src 属性，并设置请求地址
script.src = 'http://localhost:3000/?callback=getData';

// 让 script 生效
document.body.appendChild(script);
```

JSONP 的缺点:

JSON 只支持 get，因为 script 标签只能使用 get 请求；JSONP 需要后端配合返回指定格式的数据。

- **document.domain** 基础域名相同 子域名不同
- **window.name** 利用在一个浏览器窗口内，载入所有的域名都是共享一个window.name
- **CORS** CORS(Cross-origin resource sharing)跨域资源共享 是一种机制，是目前主流的跨域解决方案，它使用额外的 HTTP 头来告诉浏览器 让运行在一个 origin (domain) 上的Web应用被准许访问来自不同源服务器上的指定的资源。服务器设置对CORS的支持原理：服务器设置Access-Control-Allow-Origin HTTP响应头之后，浏览器将会允许跨域请求

- 1.浏览器端会自动向请求头添加origin字段，表明当前请求来源。
- 2.服务器设置Access-Control-Allow-Origin、Access-Control-Allow-Methods、Access-Control-Allow-Headers等 HTTP响应头字段之后，浏览器将会允许跨域请求。

预检

但是还有复杂一点的请求，我们需要先发OPTIONS请求，a.com想请求b.com它需要发一个自定义的Headers: X-ABC和content-type，这个时候就不是简单请求了，a.com要给b.com发一个options请求，它其实在问b.com我用post行不行，还想在Headers中带X-ABC和content-type；并不是所有的headers都发这个OPTIONS请求，因为X-ABC是自定义的，所以需要发；b.com看到OPTIONS请求，先不会返回数据，先检查自己的策略，看看能不能支持这次请求，如果支持就返回200。

OPTIONS请求返回以下报文

HTTP/2.0 20 OK Access-Control-Allow-Origin:https://a.com Access-Control-Allow-Methods:POST,GET,OPTIONS Access-Control-Allow-Headers:X-ABC,Content-Type Access-Control-Max-Age:86400 // 告诉浏览器这个策略生效时间为一个小时，在一个小时之内发送类似的请求，不用在问服务端了，相当于缓存了 浏览器收到了OPTIONS的返回，会在发一次，这一次才是真正的请求数据，这次headers会带上X-ABC、contentType。

整体的过程cors将请求分为2种，简单请求和复杂请求，需不需要发送OPTIONS浏览器说的算，浏览器判断是简单请求还是复杂请求，cors是非常广泛的跨域手段 这里的缺点是OPTIONS请求也是一次请求，消耗带宽，真正的请求也会延迟。

- 最方便的跨域方案 **proxy代理+ Nginx** nginx是一款极其强大的web服务器，其优点就是轻量级、启动快、高并发。

跨域问题的产生是因为浏览器的同源政策造成的，但是服务器与服务器之间的数据交换是没有这个限制。

反向代理就是采用这种方式，建立一个虚拟的代理服务器来接收 internet 上的链接请求，然后转发给内部网络上的服务器，并将从服务器上得到的结果，返回给 internet 上请求链接的客户端。现在的新项目中nginx几乎是首选，我们用node或者java开发的服务通常都需要经过nginx的反向代理。

- **window.postMessage()** 利用h5新特性window.postMessage()

跨域传送门  # 跨域，不可不知的基础概念

粘包问题分析与对策

TCP粘包是指发送方发送的若干包数据到接收方接收时粘成一包，从接收缓冲区看，后一包数据的头紧接着前一包数据的尾。

粘包出现原因

简单得说，在流传输中出现，UDP不会出现粘包，因为它有**消息边界**

粘包情况有两种，一种是**粘在一起的包都是完整的数据包**，另一种情况是**粘在一起的包有不完整的包**。

为了**避免粘包**现象，可采取以下几种措施：

(1) 对于发送方引起的粘包现象，用户可通过编程设置来避免，**TCP提供了强制数据立即传送的操作指令push**，TCP软件收到该操作指令后，就立即将本段数据发送出去，而不必等待发送缓冲区满；

(2) 对于接收方引起的粘包, 则可通过优化程序设计、精简接收进程工作量、**提高接收进程优先级等措施**, 使其及时接收数据, 从而尽量避免出现粘包现象;

(3) 由接收方控制, 将一包数据按结构字段, 人为控制分多次接收, 然后合并, 通过这种手段来避免粘包。**分包多发**。

以上提到的三种措施, 都有其不足之处。

(1) 第一种编程设置方法虽然可以避免发送方引起的粘包, 但它关闭了优化算法, 降低了网络发送效率, 影响应用程序的性能, 一般不建议使用。

(2) 第二种方法只能减少出现粘包的可能性, 但并不能完全避免粘包, 当发送频率较高时, 或由于网络突发可能使某个时间段数据包到达接收方较快, 接收方还是有可能来不及接收, 从而导致粘包。

(3) 第三种方法虽然避免了粘包, 但应用程序的效率较低, 对实时应用的场合不适合。

一种比较周全的对策是: 接收方创建一预处理线程, 对接收到的数据包进行预处理, 将粘连的包分开。实验证明这种方法是高效可行的。

客户端与服务端长连接的几种方式

1. **ajax 轮询 实现原理**: ajax 轮询指客户端每间隔一段时间向服务端发起请求, 保持数据的同步。

优点: 可实现基础 (指间隔时间较短) 的数据更新。

缺点: 这种方法也只是尽量的模拟即时传输, 但并非真正意义上的即时通讯, 很有可能出现客户端请求时, 服务端数据并未更新。或者服务端数据已更新, 但客户端未发起请求。导致多次请求资源浪费, 效率低下。【**数据更新不及时, 效率低下**】

2. long poll 长轮询

实现原理: long poll 指的是客户端发送请求之后, 如果没有数据返回, 服务端会将请求挂起放入队列 (不断开连接) 处理其他请求, 直到有数据返回给客户端。然后客户端再次发起请求, 以此轮询。在 HTTP1.0 中客户端可以设置请求头 Connection:keep-alive, 服务端收到该请求头之后知道这是一个长连接, 在响应报文头中也添加 Connection:keep-alive。客户端收到之后表示长连接建立完成, 可以继续发送其他的请求。在 HTTP1.1 中默认使用了 Connection:keep-alive 长连接。

优点: 减少客户端的请求, 降低无效的网络传输, 保证每次请求都有数据返回, 不会一直占用线程。

缺点: 无法处理高并发, 当客户端请求量大, 请求频繁时对服务器的处理能力要求较高。服务器一直保持连接会消耗资源, 需要同时维护多个线程, 服务器所能承载的 TCP 连接数是有上限的, 这种轮询很容易把连接数顶满。每次通讯都需要客户端发起, 服务端不能主动推送。【**无法处理高并发, 消耗服务器资源严重, 服务端不能主动推送**】

3. iframe 长连接

实现原理:

在网页上嵌入一个 iframe 标签, 该标签的 src 属性指向一个长连接请求。这样服务端就可以源源不断地给客户端传输信息。保障信息实时更新。

优点: 消息及时传输。

缺点: **消耗服务器资源**。

4. WebSocket

实现原理： Websocket 实现了客户端与服务端的双向通信，只需要连接一次，就可以相互传输数据，很适合实时通讯、数据实时更新等场景。

Websocket 协议与 HTTP 协议没有关系，它是一个建立在 TCP 协议上的全新协议，为了兼容 HTTP 握手规范，在握手阶段依然使用 HTTP 协议，握手完成之后，数据通过 TCP 通道进行传输。

Websocket 数据传输是通过 frame 形式，一个消息可以分成几个片段传输。这样大数据可以分成一些小片段进行传输，不用考虑由于数据量大导致标志位不够的情况。也可以边生成数据边传递消息，提高传输效率。

优点： 双向通信。客户端和服务端双方都可以主动发起通讯。没有同源限制。客户端可以与任意服务端通信，不存在跨域问题。数据量轻。第一次连接时需要携带请求头，后面数据通信都不需要带请求头，减少了请求头的负荷。传输效率高。因为只需要一次连接，所以数据传输效率高。

缺点： 长连接需要后端处理业务的代码更稳定，推送消息相对复杂；

长连接受网络限制比较大，需要处理好重连。

兼容性，WebSocket 只支持 IE10 及其以上版本。

服务器长期维护长连接需要一定的成本，各个浏览器支持程度不一；

成熟的 HTTP 生态下有大量的组件可以复用，WebSocket 则没有，遇到异常问题难以快速定位快速解决。【需要后端代码稳定，受网络限制大，兼容性差，维护成本高，生态圈小】

利用Socket建立网络连接步骤

建立Socket连接至少需要一对套接字，其中一个运行于客户端，称为ClientSocket，另一个运行于服务器端，称为ServerSocket。

套接字之间的连接过程分为三个步骤：服务器监听，客户端请求，连接确认。

1、服务器监听：服务器端套接字并不定位具体的客户端套接字，而是处于等待连接的状态，实时监控网络状态，等待客户端的连接请求。

2、客户端请求：指客户端的套接字提出连接请求，要连接的目标是服务器端的套接字。

为此，客户端的套接字必须首先描述它要连接的服务器的套接字，指出服务器端套接字的地址和端口号，然后就向服务器端套接字提出连接请求。

3、连接确认：当服务器端套接字监听到或者说接收到客户端套接字的连接请求时，就响应客户端套接字的请求，建立一个新的线程，把服务器端套接字的描述发给客户端，一旦客户端确认了此描述，双方就正式建立连接。

而服务器端套接字继续处于监听状态，继续接收其他客户端套接字的连接请求。

非对称加密RSA

简介：

1. 对称加密算法又称现代加密算法。
2. 非对称加密是计算机通信安全的基石，保证了加密数据不会被破解。
3. 非对称加密算法需要两个密钥：公开密钥(publickey) 和私有密(privatekey)
4. 公开密钥和私有密钥是一对

如果用公开密钥对数据进行加密，只有用对应的私有密钥才能解密。 如果用私有密钥对数据进行加密，只有用对应的公开密钥才能解密。

特点： 算法强度复杂，安全性依赖于算法与密钥。 加密解密速度慢。

与对称加密算法的对比： 对称加密只有一种密钥，并且是非公开的，如果要解密就得让对方知道密钥。 非对称加密有两种密钥，其中一个是公开的。

RSA应用场景： 由于RSA算法的加密解密速度要比对称算法速度慢很多，在实际应用中，通常采取 数据本身的加密和解密使用对称加密算法(AES)。 用RSA算法加密并传输对称算法所需的密钥。

HTTP1、HTTP2、HTTP3

HTTP/2 相比于 HTTP/1.1，可以说是大幅度提高了网页的性能，只需要升级到该协议就可以减少很多之前需要做的性能优化工作，虽如此但HTTP/2并非完美的，HTTP/3 就是为了解决 HTTP/2 所存在的一些问题而被推出来的。

HTTP1.1 的缺陷

1. 高延迟 — 队头阻塞(Head-Of-Line Blocking)

队头阻塞是指当顺序发送的请求序列中的一个请求因为某种原因被阻塞时，在后面排队的所有请求也一并被阻塞，会导致客户端迟迟收不到数据。

针对队头阻塞的解决办法:

- 将同一页面的资源分散到不同域名下，提升连接上限。
- 合并小文件减少资源数，使用精灵图。
- **内联(Inlining)**资源是另外一种防止发送很多小图请求的技巧，将图片的原始数据嵌入在CSS文件里面的URL里，减少网络请求次数。
- 减少请求数量，合并文件。

2. 无状态特性 — 阻碍交互

无状态是指协议对于连接状态没有记忆能力。纯净的 HTTP 是没有 cookie 等机制的，每一个连接都是一个新的连接。

Header里携带的内容过大，在一定程度上增加了传输的成本。且请求响应报文里有大量字段值都是重复的。

3. 明文传输 — 不安全性

HTTP/1.1在传输数据时，所有**传输的内容都是明文**，客户端和服务端都无法验证对方的身份，无法保证数据的安全性。

4. 不支持服务端推送

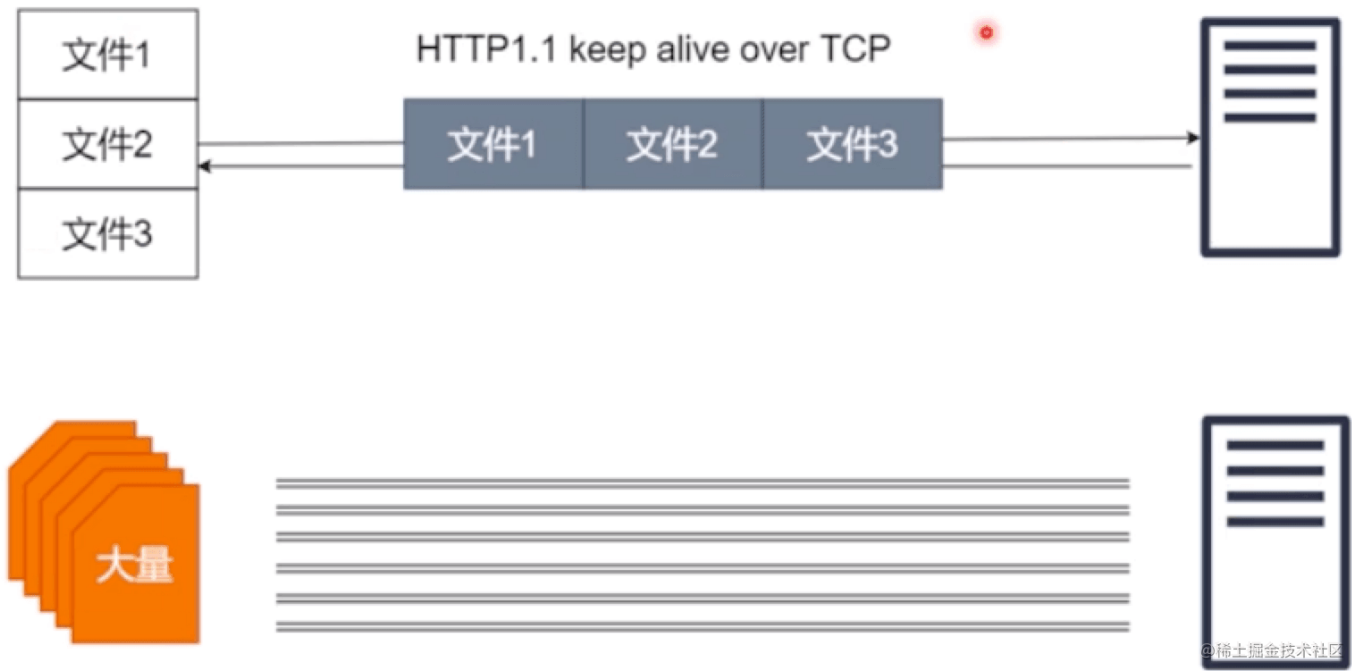
记忆口诀：队头阻塞高延迟，无状态阻交互，明文传输不安全，服务推送不支持。

HTTP 1.1 排队问题

HTTP 1.1多个文件共用一个TCP，这样可以减少tcp握手，这样3个文件就不用握手9次了，不过这样请求文件需要排队，请求和返回都需要排队，如果第一个文件响应慢，会阻塞后面的文件，这样就产生了对头的等待问

题。

有的网站可能会有很多文件，浏览器处于对机器性能的考虑，它不可能让你无限制的发请求建连接，因为建立连接需要占用资源，浏览器不想把用户的网络资源都占用了，所以浏览器最多会建立6个tcp连接；如果有上百个文件可能都需要排队，http2.0正在解决这个问题。



SPDY 协议与 HTTP/2 简介

1、HTTP/2 简介

HTTP/2是现行HTTP协议（HTTP/1.x）的替代，但它不是重写。**HTTP/2基于SPDY，专注于性能，最大的一个目标是在用户和网站间只用一个连接（connection）。**

2、HTTP/2 新特性

1、二进制传输

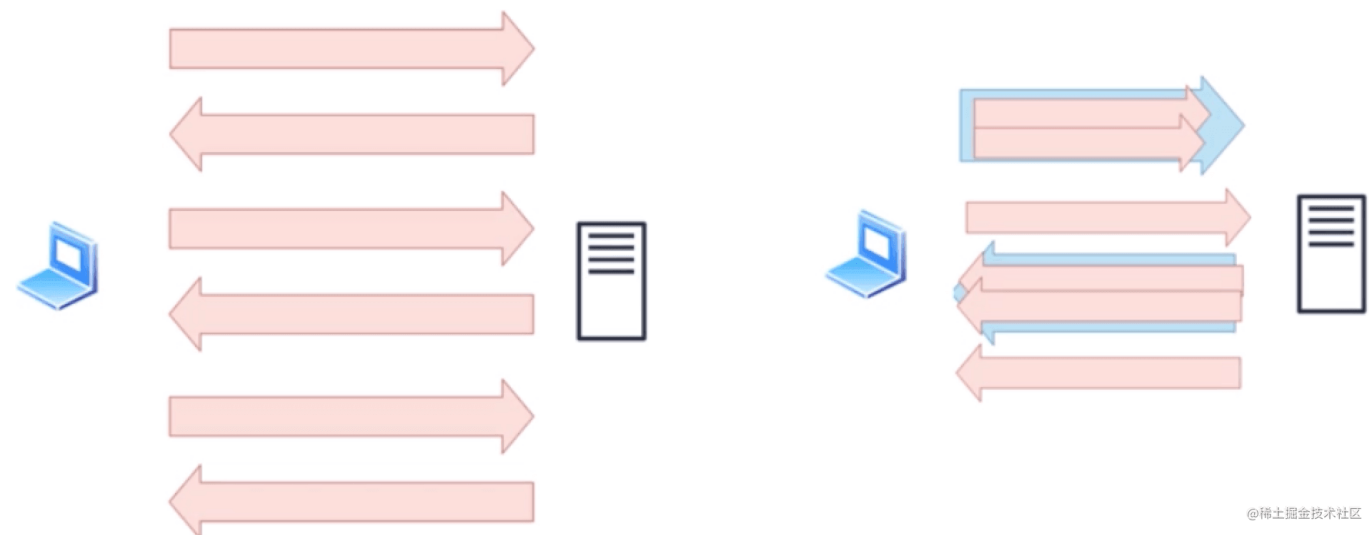
HTTP/2传输数据量的大幅减少,主要有两个原因:以二进制方式传输和Header 压缩。我们先来介绍二进制传输,HTTP/2 采用二进制格式传输数据，而非HTTP/1.x 里纯文本形式的报文，二进制协议解析起来更高效。HTTP/2 将请求和响应数据分割为更小的帧，并且它们采用二进制编码。

2、Header 压缩

HTTP/2并没有使用传统的压缩算法，而是开发了专门的“HPACK”算法，在客户端和服务端两端建立“字典”，用索引号表示重复的字符串，还采用哈夫曼编码来压缩整数和字符串，可以达到50%~90%的高压缩率。

3、多路复用

在 HTTP/2 中引入了多路复用的技术。多路复用很好的解决了浏览器限制同一个域名下的请求数量的问题，同时也更容易实现全速传输。



4、Server Push

HTTP2还在一定程度上改变了传统的“请求-应答”工作模式，服务器不再是完全被动地响应请求，也可以新建“流”主动向客户端发送消息。减少等待的延迟，这被称为“服务器推送”（Server Push，也叫 Cache push）

5、提高安全性

出于兼容的考虑，HTTP/2延续了HTTP/1的“明文”特点，可以像以前一样使用明文传输数据，不强制使用加密通信，不过格式还是二进制，只是不需要解密。

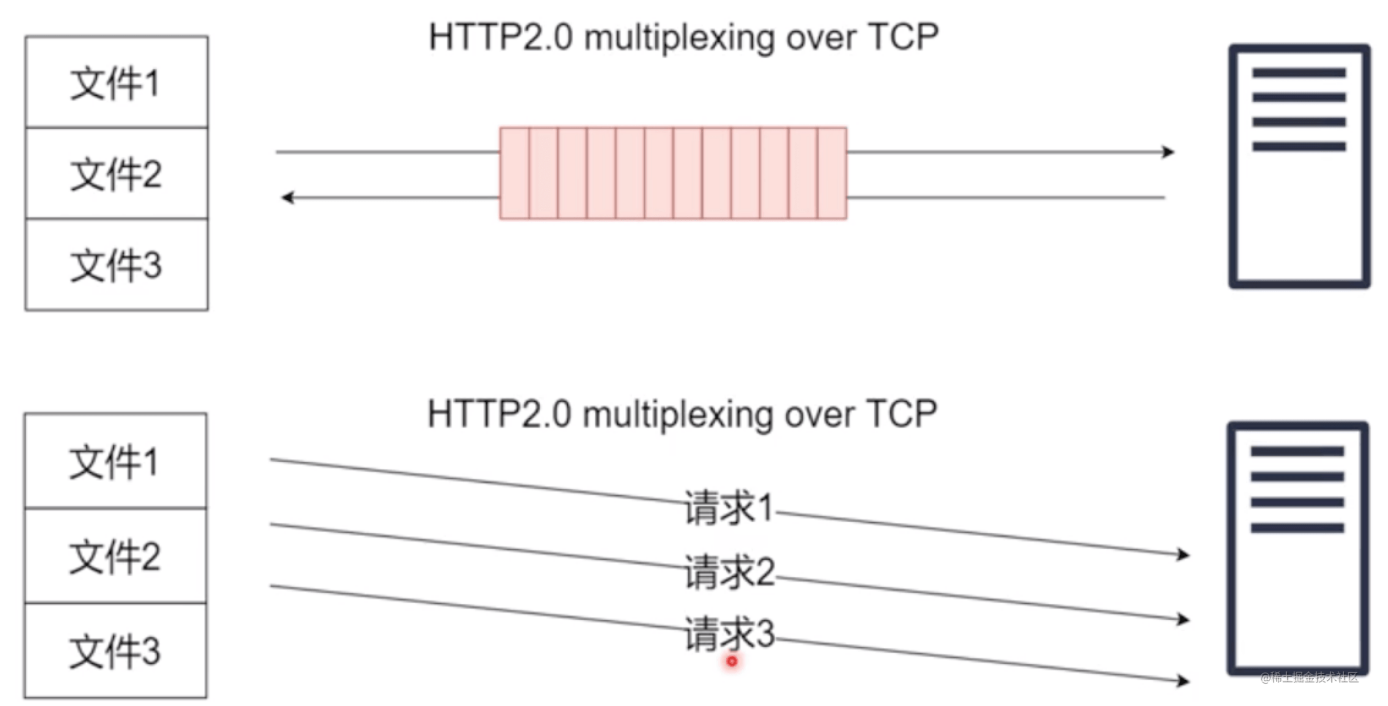
但由于HTTPS已经是大势所趋，而且主流的浏览器Chrome、Firefox等都公开宣布只支持加密的HTTP/2，所以“事实上”的HTTP/2是加密的。也就是说，互联网上通常所能见到的HTTP/2都是使用“https”协议名，跑在TLS上面。HTTP/2协议定义了两个字符串标识符：“h2”表示加密的HTTP/2，“h2c”表示明文的HTTP/2。

6、防止对头阻塞

http1.1如果第一个文件阻塞，第二个文件也就阻塞了。



http2.0的解决，把3个请求打包成一个小块发送过去，即使第一个阻塞了，后面2个也可以回来；相当于3个文件同时请求，就看谁先回来谁后回来，阻塞的可能就后回来，对带宽的利用是最高的；但没有解决TCP的对头阻塞，如果TCP发过去的一个分包发丢了，他会重新发一次；http2.0的解决了大文件的阻塞。



一个分包请求3个文件，即使第一个阻塞了，第二个也能返回

HTTP/2 的缺点

虽然 HTTP/2 解决了很多之前旧版本的问题，但它还是存在一个巨大的问题，主要是底层支撑的 TCP 协议造成的。HTTP/2的缺点主要有以下几点：

- 1. TCP 以及 TCP+TLS 建立连接时延时
- 2. TCP 的队头阻塞并没有彻底解决
- 3. 多路复用导致服务器压力上升也容易 Timeout

HTTP/3 新特性

1、HTTP/3简介

Google 在推SPDY的时候就搞了个基于 UDP 协议的“QUIC”协议，让HTTP跑在QUIC上而不是TCP上。而“HTTP over QUIC”就是HTTP/3，真正“完美”地解决了“队头阻塞”问题。

QUIC 虽然基于 UDP，但是在原本的基础上新增了很多功能，接下来我们重点介绍几个QUIC新功能。

2、QUIC新功能

QUIC基于UDP，而UDP是“无连接”的，根本就不需要“握手”和“挥手”，所以就比TCP来得快。此外QUIC也实现了可靠传输，保证数据一定能够抵达目的地。它还引入了类似HTTP/2的“流”和“多路复用”，单个“流”是有序的，可能会因为丢包而阻塞，但其他“流”不会受到影响。具体来说QUIC协议有以下特点：

- 实现了类似TCP的流量控制、传输可靠性的功能

虽然UDP不提供可靠性的传输，但QUIC在UDP的基础之上增加了一层来保证数据可靠性传输。它提供了数据包重传、拥塞控制以及其他一些TCP中存在的特性。

- 实现了快速握手功能

由于QUIC是基于UDP的，所以QUIC可以实现使用0-RTT或者1-RTT来建立连接，这意味着QUIC可以用最快的速度来发送和接收数据，这样可以大大提升首次打开页面的速度。**0RTT 建连可以说是 QUIC 相比 HTTP2 最大的性能优势。**

- **集成了TLS加密功能**
- **多路复用，彻底解决TCP中队头阻塞的问题**

和TCP不同，QUIC实现了在同一物理连接上可以有多个独立的逻辑数据流。实现了数据流的单独传输，就解决了TCP中队头阻塞的问题。

- **连接迁移**

TCP 是按照 4 要素（客户端 IP、端口, 服务器 IP、端口）确定一个连接的。而 QUIC 则是让客户端生成一个 Connection ID （64 位）来区别不同连接。只要 Connection ID 不变，连接就不需要重新建立，即便是客户端的网络发生变化。由于迁移客户端继续使用相同的会话密钥来加密和解密数据包，QUIC 还提供了迁移客户端的自动加密验证。

总结

- HTTP/1.1有两个主要的缺点：安全不足和性能不高。
- HTTP/2完全兼容HTTP/1，是“更安全的HTTP、更快的HTTPS”，二进制传输、头部压缩、多路复用、服务器推送等技术可以充分利用带宽，降低延迟，从而大幅度提高上网体验；
- QUIC 基于 UDP 实现，是 HTTP/3 中的底层支撑协议，该协议基于 UDP，又取了 TCP 中的精华，实现了即快又可靠的协议。

理解xss, csrf, ddos攻击原理以及避免方式

XSS(Cross-Site Scripting, 跨站脚本攻击)是一种代码注入攻击。攻击者在目标网站上注入恶意代码，当被攻击者登陆网站时就会执行这些恶意代码，这些脚本可以读取 `cookie`, `session tokens`，或者其它敏感的网站信息，对用户进行钓鱼欺诈，甚至发起蠕虫攻击等。

CSRF (Cross-site request forgery) 跨站请求伪造：攻击者诱导受害者进入第三方网站，在第三方网站中，向被攻击网站发送跨站请求。利用受害者在被攻击网站已经获取的注册凭证，绕过后台的用户验证，达到冒充用户对被攻击的网站执行某项操作的目的。

XSS避免方式：

1. url参数使用`encodeURIComponent`方法转义
2. 尽量不是有`innerHTML`插入HTML内容
3. 使用特殊符号、标签转义符。

CSRF避免方式：

1. 添加验证码
2. 使用token
 - 服务端给用户生成一个token，加密后传递给用户
 - 用户在提交请求时，需要携带这个token
 - 服务端验证token是否正确

DDoS又叫分布式拒绝服务，全称 **Distributed Denial of Service**，其原理就是利用大量的请求造成资源过载，导致服务不可用。

DDos避免方式：

1. 限制单IP请求频率。
2. 防火墙等防护设置禁止**ICMP**包等
3. 检查特权端口的开放

360技术：嗨，送你一张Web性能优化地图