Contents

[从输入URL到页面加载完毕发生了什么 2](#_Toc36573572)

[HTTP和HTTPS的区别 4](#_Toc36573573)

[HTTP报文 5](#_Toc36573574)

[CORS跨域资源共享 7](#_Toc36573575)

[Cookie 8](#_Toc36573576)

[Cookie的作用 8](#_Toc36573577)

[Cookie存在的问题 8](#_Toc36573578)

[Session 8](#_Toc36573579)

[LocalStorage 9](#_Toc36573580)

[SessionStorage 9](#_Toc36573581)

[Cache-control（强缓存） 9](#_Toc36573582)

[Etag（协商缓存） 9](#_Toc36573583)

[基于token的身份验证 10](#_Toc36573584)

[XSS跨站脚本攻击 10](#_Toc36573585)

[重定向状态码3XX 11](#_Toc36573586)

[1XX信息 11](#_Toc36573587)

[2XX成功 11](#_Toc36573588)

[4XX客户端错误 11](#_Toc36573589)

[5XX服务器错误 12](#_Toc36573590)

[HTTP版本对比 12](#_Toc36573591)

[三次握手和四次挥手 13](#_Toc36573592)

# 从输入URL到页面加载完毕发生了什么

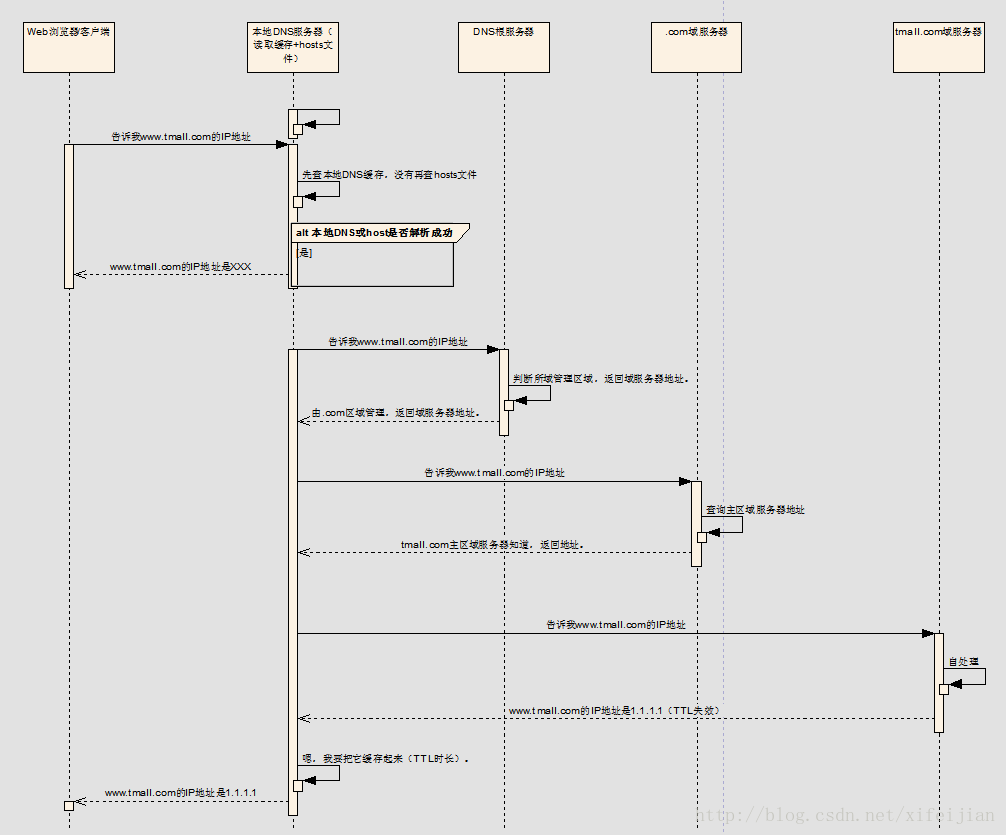
* **操作系统GUI将输入事件传递到浏览器**

在输入的过程中浏览器可能已经在做一些预处理，比如url的模糊匹配，如果直接键入回车则会直接跳到建立TCP连接的步骤甚至直接用缓存开始渲染。如Chrome

* **浏览器到浏览器内核**

浏览器会判断协议和安全检查然后交由浏览器内核开始网络请求

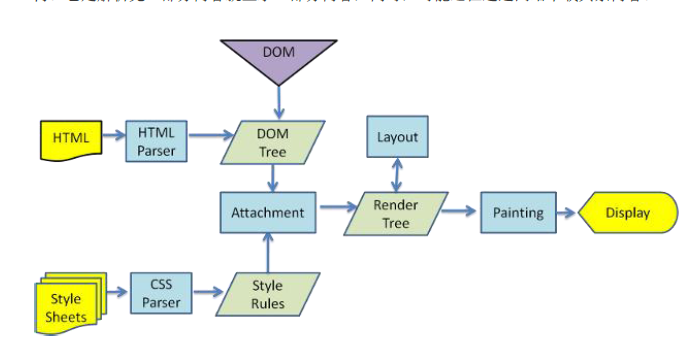
* **DNS解析**，将域名迭代解析为IP地址（IPV4），如lyp970805.com->116.62.181.32。



* + 这个过程中会先在本地域名服务器（先DNS高速缓存后hosts文件）中找，如果解析成功了直接就可以获得IP地址，否则去IPS DNS服务器
  + 互联网服务提供商（中国电信）提供查询如著名114.114.114.114，否则去根服务器
  + 根服务器判断域管理区域返回服务器地址（顶级域名阶段如.com），再去顶级域服务器问IP
  + 顶级与服务器给出主区域服务器地址，去主区域服务器地址
  + 自处理完成余下的域名分析返回一个IP，本地发现延迟过长就会缓存下来，并设置一个TTL（Time to Live），注意浏览器中的TTL与本地设置无关，一般每种浏览器有一个自己的固定值
    - DNS缓存：DNS存在多级缓存，从离浏览器距离排序依次是浏览器缓存，系统缓存，路由器缓存，IPS服务器缓存，根域名服务器缓存，顶级域名服务器缓存，主域名服务器缓存
    - DNS负载均衡：DNS服务器会根据每台服务器的负载机器和用户地理位置的距离等等因素动态分配IP，这就是DNS重定向。CDN（内容分发网络）就利用了这个技术。
* **建立TCP连接、发送HTTP请求**

拿到域名对应的IP地址后浏览器会以一个随机端口向服务器的WEB程序（httpd，nginx）80端口发起TCP连接请求，途径各种路由设备，到网卡，经过防火墙过滤，进入内核的TCP/IP协议栈，最终建立连接。Apache会创建一个的独立进程来处理

* **从服务端拿回数据到浏览器进行渲染**



浏览器是一个边解析边渲染的过程。首先浏览器解析HTML文件构建DOM树，然后解析CSS文件构建渲染树，等到渲染树构建完成后，浏览器开始布局渲染树并将其绘制到屏幕上。这个过程比较复杂，涉及到两个概念: reflow(回流)和repain(重绘)。DOM节点中的各个元素都是以盒模型的形式存在，这些都需要浏览器去计算其位置和大小等，这个过程称为relow;当盒模型的位置,大小以及其他属性，如颜色,字体,等确定下来之后，浏览器便开始绘制内容，这个过程称为repain。页面在首次加载时必然会经历reflow和repain。reflow和repain过程是非常消耗性能的，尤其是在移动设备上，它会破坏用户体验，有时会造成页面卡顿。所以我们应该尽可能少的减少reflow和repain。

注意文档加载过程中遇到js文件html文档会挂起渲染的线程要等待js文件加载解析执行完毕才可以恢复html的渲染线程。

# HTTP和HTTPS的区别

都是用于在web和服务器间传递信息，但http是明文传输的，可以直接通过调试工具和其他抓包方法获取到传输报文，对于一些机密信息显然是不适合的。HTTPS在HTTP的基础上添加了SSL协议，依靠证书来验证身份并为浏览器和服务器之间的通信加密。

二者有一些明显区别

* + HTTP明文，HTTPS非对称RSA加密
  + HTTPS需要证书
  + 连接方式不同，端口也不同HTTP80 HTTPS443

# HTTP报文

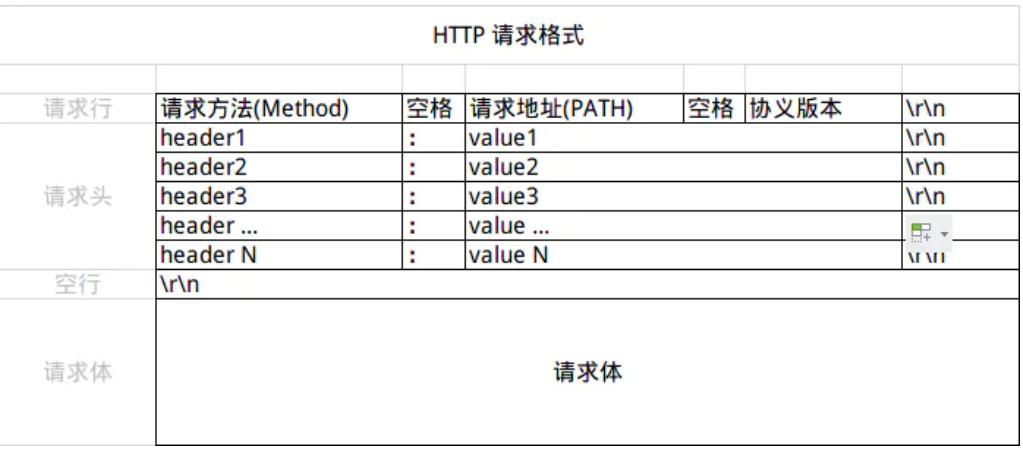
* **请求行**

请求方法+请求地址+协议及版本，以CRLF(\r\n结束)

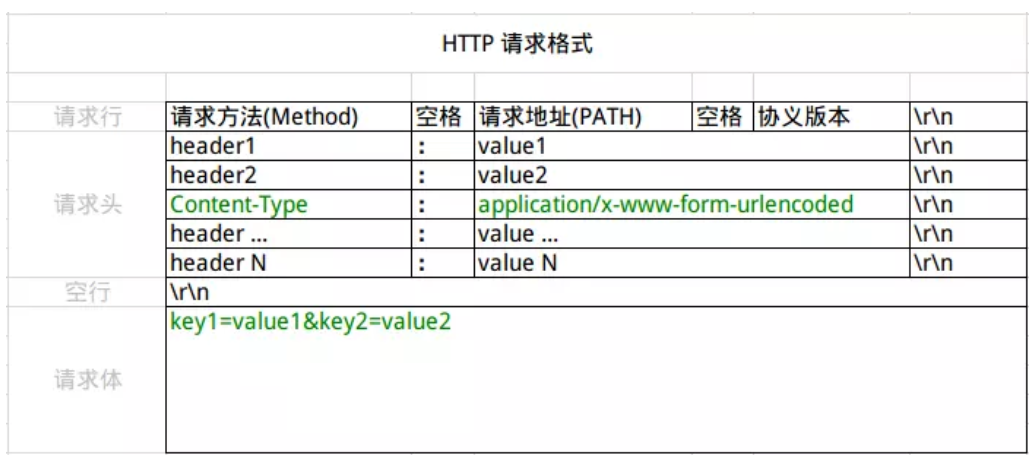
* **响应行**

协议及版本+状态码+状态，以CRLF(\r\n结束)

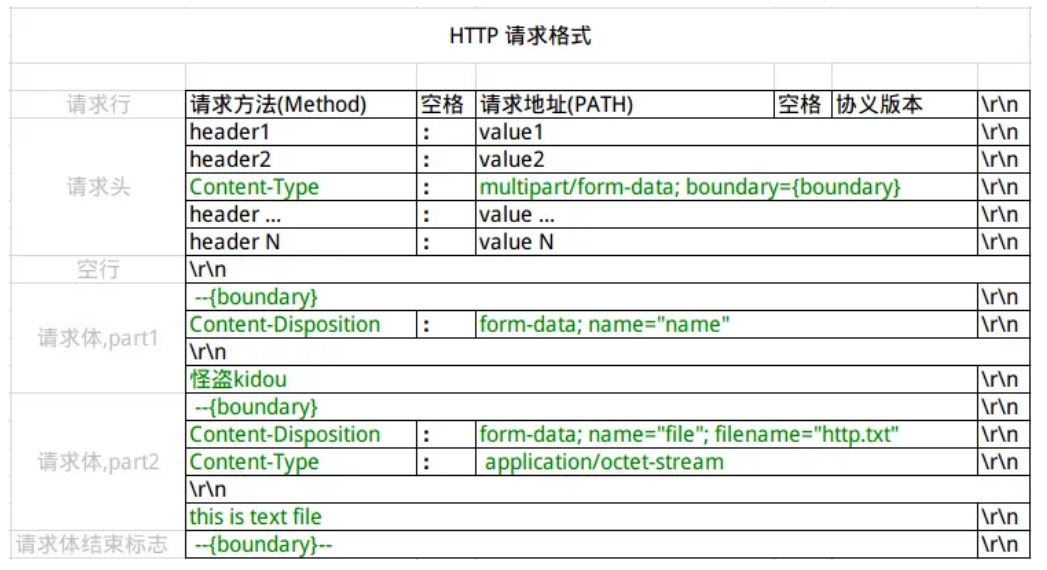
* **Header**
  + **通用**
    - Accept:说明接受的类型，可以多个值
    - Content-Type:请求体/响应体类型
      * application/x-www-form-urlencoded
      * multipart/form-data
      * multipart/json
      * text/plain
    - Content-Length:请求体/响应体长度（单位byte）
    - Content-Encoding:请求体/响应体的编码格式（gzip，defalte）
  + **请求**
    - User-Agent:用户标识（操作系统+浏览器）
    - Host：请求的主机和端口号
    - Origin：请求发起方（域名），跨域请求时会带上
    - Referer：Origin基础上会带上路由信息，防止盗链
  + **响应**
    - Location:重定向到另一个URL
    - Access-allow-origin：对应跨域请求
    - Server：后台服务器
* 请求体的三种形式
  + 请求体为任意类型，服务器不会解析请求体如POST



* + URL中的Query String格式要求 多个键值对之间用&连接，键与值之间用=连接



* + 请求体被分成多个部分，文件上传时常用，每个字段被boundary(content-type指定)分成独立的段，每段--加boundary开头，然后是该段的描述头，描述头之后CRLF，接内容，请求接数标志位boundary后--

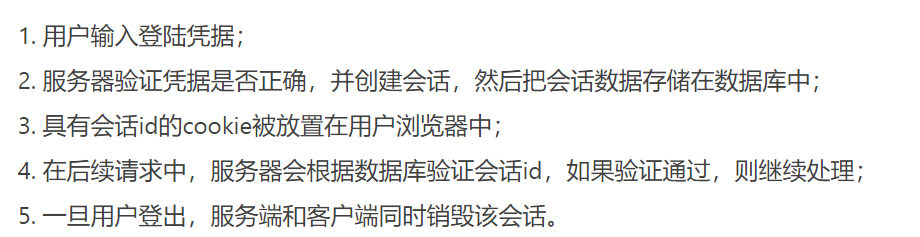


# CORS跨域资源共享

# Cookie

Cookie是服务器发送到用户浏览器并保存在本地的一小块数据，他会在浏览器下次向同一服务器发起请求时被携带并发送到服务器

Cookie本质上是HTTP的一个请求头



### Cookie的作用

* 识别用户身份

服务器回返给浏览器一个身份（cookie），当浏览器再次访问服务器会强制要求带上身份信息，服务器则可以利用这个身份信息（cookie）来识别浏览器

* 记录历史

Js可以修改储存在本地的cookie，下次打开网站时可以知道之前的历史操作

### Cookie存在的问题

* 基于浏览器储存，不同浏览器不共享
* 储存大小有限制，一般4KB左右（chrome 4KB）
* Cookie可以被用户手动修改
* 有效期默认20分钟不过可以被后端设置（如自动登录）
* Cookie的同源策略，cookie只需要是同一父级域名即可，如访问v.qq.com不仅有v.qq.com的cookie还会带上qq.com的cookie。注意浏览器中[www.qq.com和qq.com](http://www.qq.com和qq.com)是不同的域名。
* Cookie是明文保存的，存在风险，session是解决安全问题的方法

# Session

Client第一次访问server时server会生成一个sessionID放在响应头以coockie形式给client，server将要保存的数据保存在对应的sessionID下，服务器有多种方式保存session如内存数据库文件等等，下次client带着sessionID访问即可直接在内存中访问数据。

* Session在服务器 cookie在客户端
* Session用户不可改不可查看 cookie可以
* Session可以理解为一种基于cookie的数据存储方式

# LocalStorage

* HTML5提供的一个API，实质是一个哈希表，存在浏览器上。
* 适用于一些和后端毫无干系的用户行为记录或不敏感信息，如“不再提示”。
* Localstorage和http没有任何关系。
* 同源策略同cookie，有相同域名的页面可以相互读取localstorage
* 不同浏览器对每个域名localstorage的存储量不同，超出会被回绝，chrome10MB左右。

# SessionStorage

几乎同loacalstroage，不同在于有效期是页面会话持续，如果页面接数sessionstorage就会消失。

# Cache-control（强缓存）

第一次请求资源时将资源缓存，浏览器再次请求该资源直接使用缓存。使用缓存机制缩短二次访问的响应时间，提高页面响应性能，实现web性能优化。

* 首页一般不设置cacheconcontrol防止全部设置缓存用户访问页面不会再向服务器请求而导致页面不更新。
* 缓存更新是基于URL的，改动资源的URL浏览器就会重新请求资源，真正开发中资源版本号一般都是使用摘要算法生成的字符串。

# Etag（协商缓存）

Etag是http的内容，通过匹配标识符来判断资源是否需要下载

* 浏览器第一次访问服务器资源时，服务器端返回一个Etag响应头，值为资源MD5摘要的值
* 浏览器再次请求资源会把Etag响应头放在if-none-match请求头中，发送给服务器
* 服务器把这个值和请求文件的md5做匹配，如果两者匹配则返回304代表资源没有改变，不用再次下载

# 基于token的身份验证

讨论基于token的身份验证一般都是说JSON WEB TOKENS（JWT）与cookie的流程相似，但是用户登出后客户端会销毁token



* Token相对cookie的优势

基于token的验证是无状态的，后端不需要记录token，每个令牌都是独立的，服务器唯一的工作就是在成功的登录请求上签署token，并验证传入的token是否有效

Cookie是绑定到单个域的而token不是，对于需要向多个服务获取授权的单页面应用尤其有用。

# XSS跨站脚本攻击

# CSRF跨站请求伪造

# 重定向状态码3XX

* 301 Moved Permanently

永久性定向，请求的资源已经被分配了新的URL，以后应该使用新的URL。SEO友好

* 302 FOUND

临时性重定向，请求的资源被分配了新的URL，希望用户本次能使用新的URL访问，即表示URL可能会发生变化，书签也不会像301一样更新

* 303 See Other

与302接近，但要求使用GET方法获取请求的资源

* 307 Temporary Redirect

与302接近，不允许浏览器将原本为post的请求重定向到get请求上

301/302/303返回时，几乎所有浏览器都会把post改成get并删除请求报文内的主题，之后请求会自动再次发送。

**历史问题：**本来只有302，但是一些浏览器遇到302会改写方法，后来新的RFC标准中修改了302的定义然后引入了新的303和307

**URL劫持：**A重定向到B，显示B但是流量归A，而在搜索排名中靠前

# 1XX信息

* 100 continue：目前正常，客户端可以继续发送请求或者忽略这个响应
* 101 协议升级：请求者要求服务器切换协议，服务器确认并准备切换

# 2XX成功

* 200OK 最常用 表示请求的数据已经放在响应体返回了
* 204 NO content：请求已经成功处理但是返回的响应报文不包含实体的主体部分，一般在只需要客户端往服务器发送消息而不需要返回数据时使用

# 4XX客户端错误

* 400BadRequest 请求报文中存在语法错误
* 401unauthorized 表示发送的请求需要有认证信息
* 403 请求被拒绝 服务器端没有必要给出拒绝的详细理由
* 404 NOT Found
* 405 method not allowed请求方式和后台规定不符合
* 408请求超时

# 5XX服务器错误

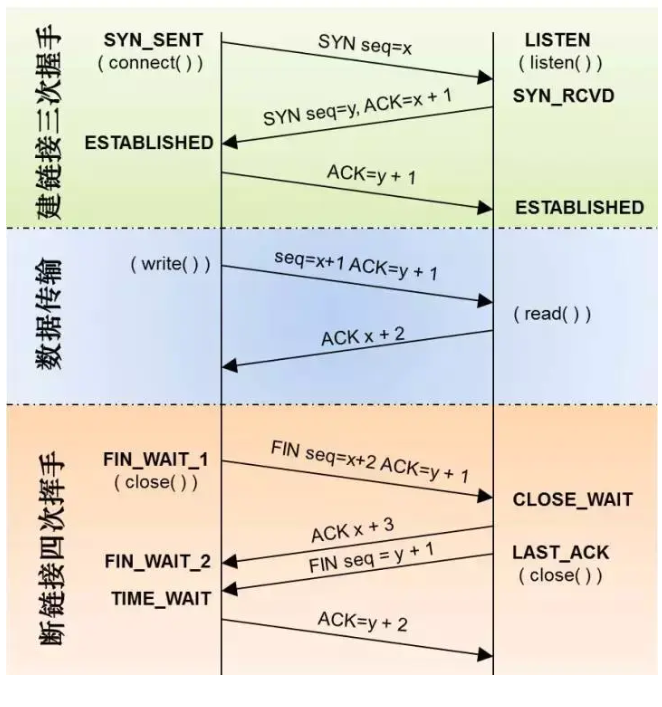
* 500服务器正在执行请求时发生错误
* 502 Bad Gateway
* 503 服务器暂时处于超负载
* 504 进程阻塞事件过长
* 505 不支持该http版本

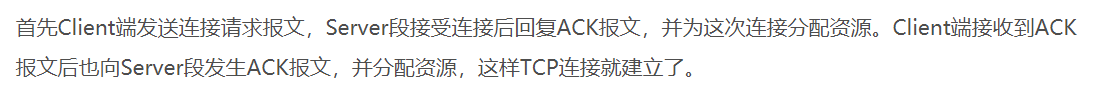
# HTTP版本对比

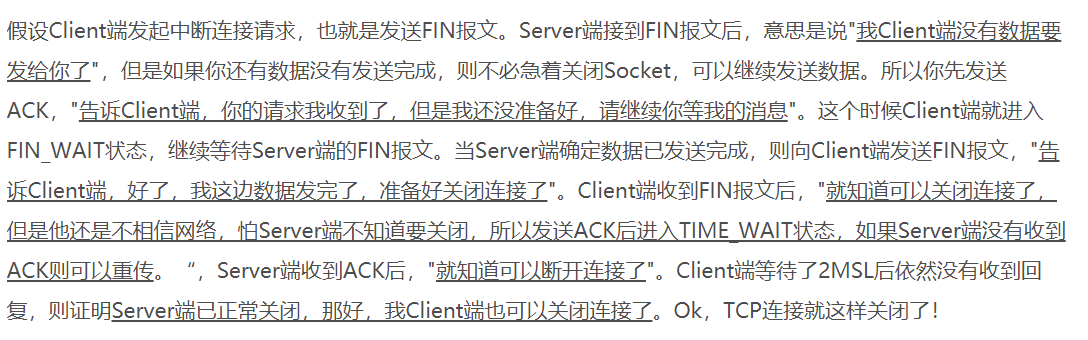
HTTP1.1支持长连接和管线化，在一个TCP连接上可以传送多个HTTP请求和响应，减少了建立和关闭连接的耗时。请求头中必须包含HOST域 否则报400错而1.0不需要带。1.1引入了更多的缓存控制策略。

HTTP2二进制分帧，多路复用

# 三次握手和四次挥手







# Get和post的区别



