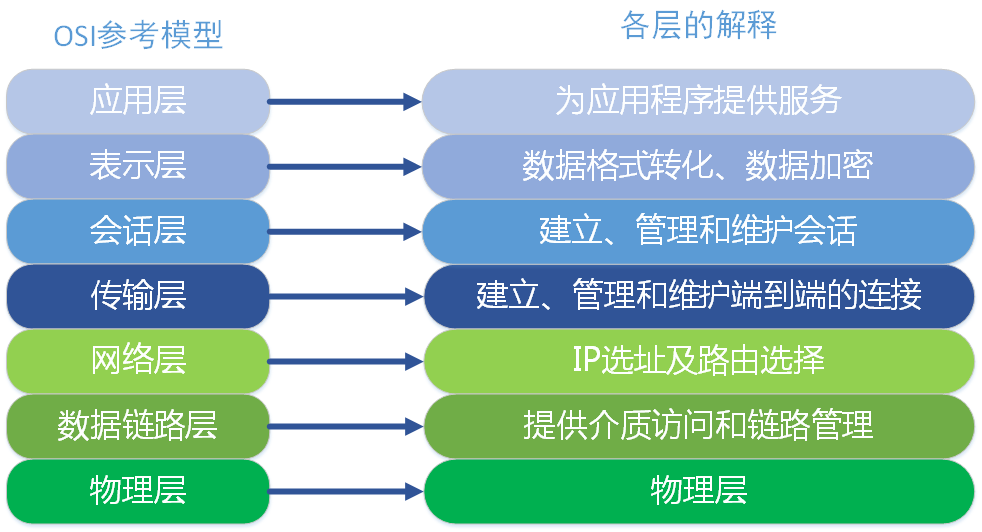
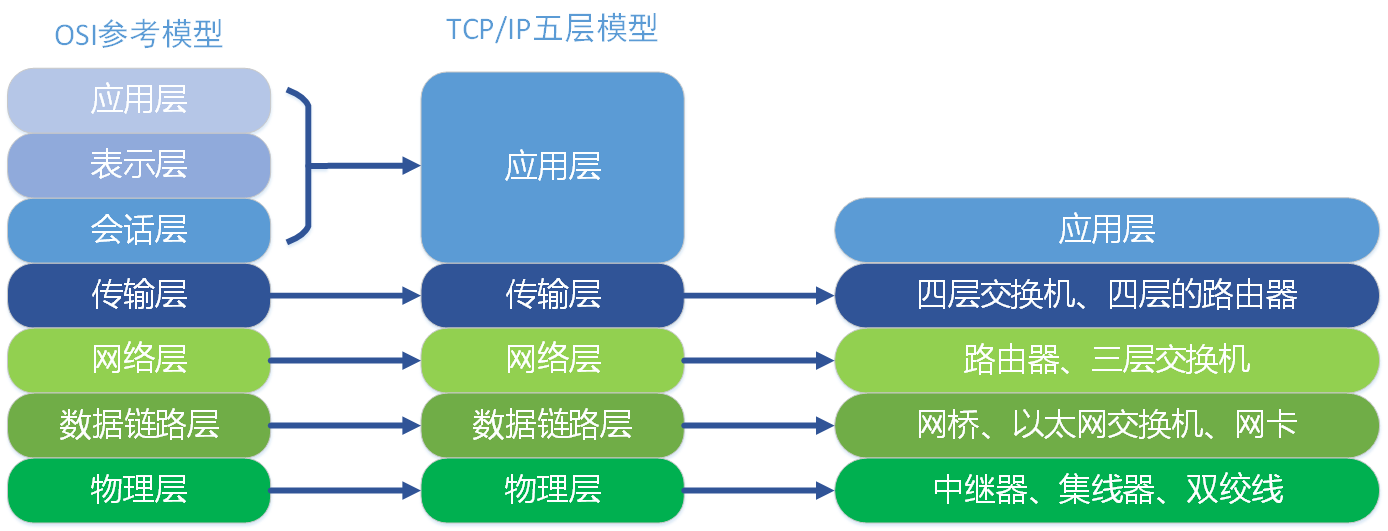
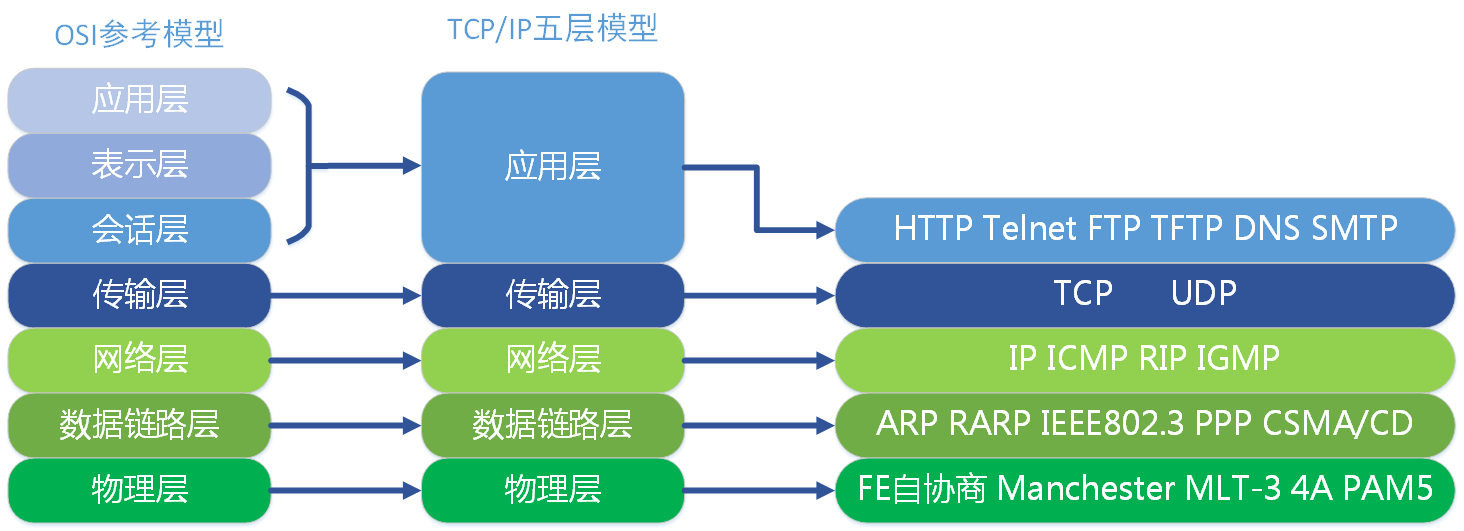
1. **OSI模型，HTTP，TCP，UDP分布在那些层**

参考：<https://www.cnblogs.com/qishui/p/5428938.html>







1. **从输入网址到显示网页的过程分析**

1、应用层DNS解析域名：客户端先检查本地是否有对应的IP地址，若找到则返回响应的IP地址。若没找到则请求上级DNS服务器，直至找到或到根节点。

2、浏览器与服务器建立TCP连接（默认端口80）(详细点可以说下三次握手的过程)

3、应用层客户端发送HTTP请求。

4、服务器响应请求：查找客户端请求的资源，并返回响应报文，响应报文中包括一个重要的信息——状态码（200-300，成功；304使用缓存）。

5、服务器返回相应文件给浏览器。

6、Tcp连接释放(可以说下四次挥手的过程)。

7、浏览器对HTML文件进行解析构建DOM树 ，构建渲染树 ，js根据DomAPI操作执行绑定事件等，页面显示完成。



TCP三次握手：

第一次握手：建立连接。客户端发送连接请求报文段，将SYN位置为1，Sequence Number为x；然后，客户端进入SYN\_SEND状态，等待服务器的确认；

第二次握手：服务器收到SYN报文段。服务器收到客户端的SYN报文段，需要对这个SYN报文段进行确认，设置Acknowledgment Number为x+1(Sequence Number+1)；同时，自己自己还要发送SYN请求信息，将SYN位置为1，Sequence Number为y；服务器端将上述所有信息放到一个报文段（即SYN+ACK报文段）中，一并发送给客户端，此时服务器进入SYN\_RECV状态；

第三次握手：客户端收到服务器的SYN+ACK报文段。然后将Acknowledgment Number设置为y+1，向服务器发送ACK报文段，这个报文段发送完毕以后，客户端和服务器端都进入ESTABLISHED状态，完成TCP三次握手。

TCP四次挥手：

（1）客户端发起连接中断请求，即向服务器端发送FIN报文；

（2）服务端收到后继续发未传完的数据，然后回复ACK报文，客户端收到ACK报文后进入等待状态；

（3）服务端确认发送完数据后，发送FIN报文；

（4）客户端收到FIN报文后，发送ACK报文，进入TIME\_WAIT状态。服务端收到ACK报文就断开连接。客户端在等待2ms后没有收到回复，可以确认服务端收到了ACK报文，断开连接。



1. 构建DOM树：按照顺序解析HTML文件，构建DOM树，在解析到外部的CSS和js文件时，向服务器请求下载资源。Css文件下载的同时可以继续解析html，js下载和执行时，解析器会停止对html的解析，这就是js阻塞。
2. 构建渲染树：浏览器解析css，形成CSSOM树，浏览器引擎通过DOM树和CSSOM树构造渲染树。
3. 布局渲染树：通过计算得到每个渲染对象在可视区域中的具体位置信息（大小和位置）
4. 绘制：将计算好的每个像素点信息绘制在屏幕上

## HTTP请求的cookie

cookie是用于维持服务端会话状态的，通常由服务端写入，保存在浏览器端，在后续请求中附加在请求头中，供服务端读取。每个cookie的大小一般不超过4KB。



HTTP请求，cookie的使用过程：

1. server通过HTTP Response中的“Set-Cookie: cookie内容”把cookie发送给client
2. client把cookie通过HTTP Request中的“Cookie：cookie内容”发送给server
3. 每次HTTP请求，cookie都会发送

HTTP请求发送cookies的条件：

1. 本地已经缓存有cookies
2. 根据请求的URL来匹配cookies的domain、path属性，如果符合才会发送。例如：访问[www.baidu.com](http://www.baidu.com)时，就不发送[www.qq.com的cookies](http://www.qq.com的cookies)

Cookie主要用来分辨两个请求是否来自同一个浏览器，以及用来保存一些状态信息。使用场合：

* + - * 1. 对话（session）管理：保存登录、购物车等记录信息
        2. 个性化：保存用户的偏好，比如网页的字体大小、背景色等
        3. 追踪：记录和分析用户行为

可以通过window.navigator.cookieEnabled=false在浏览器端设置不接收cookie，不向服务器端发送cookie

#### cookie的属性（服务器端写cookie时设置）

1. Expires，Max-age
   1. Expires：具体的到期时间，到时间后，浏览器不再保留这个cookie。Date.prototype.toGMTString()将时间转换成GMT格式
   2. Max-Age：从现在开始Cookie存在的秒数。Max-Age的值优先生效

如果Set-Cookie字段没有指定Expires或Max-Age属性，该Cookie 就是 Session Cookie，即它只在本次对话存在，一旦用户关闭浏览器，浏览器就不会再保留这个 Cookie，即使关闭网页，会话也未结束。

1. Domain，Path
   1. Domain：浏览器向该域名下的地址发出HTTP请求时，要附带这个Cookie。默认为当前URL的一级域名。如果服务器在Set-Cookie字段指定的域名不属于当前域名，浏览器拒绝这个Cookie
   2. Path：浏览器向指定Path指定的路径以及子路径发送HTTP请求时，要附带这个Cookie。
2. Secure，HttpOnly
   1. Secure：只有在https下才能将cookie发送到服务器。该属性只是一个**开关**，不需要指定值。在https下通信，会自动添加Secure
   2. HttpOnly：指定该Cookie无法通过js脚本拿到，主要是document.cookie、XMLHttpRequest对象和 Request API 都拿不到该属性。只有浏览器发出 HTTP 请求时，才会带上该 Cookie。一定程度上可以阻止跨站脚本攻击XSS

网站的登录与App的自动登录原理：

用户在连接服务器时，服务器生成一个唯一的SessionID，而SessionID这一数据则是用cookie保存在客户端，用户提交页面时，会将SessionID提交到服务器端，来存取Session数据。

Cookie跨域解决：

1. document.domain
2. jsonp：A页面请求不同域的B页面的cookie，A向服务器发起请求（以回调函数名称作为参数），由服务器获取B页面的cookie，通过动态插入js代码的方式，使用回调函数包裹请求的数据cookie，A页面就获取到了cookie

## Cookie的读取，设置,以及删除





## 前端存在哪些安全问题？

### CSRF

CSRF（Cross-site request forgery），中文名称：跨站请求伪造，也被称为：one click attack/session riding，缩写为：CSRF/XSRF。

你这可以这么理解CSRF攻击：攻击者盗用了你的身份，以你的名义发送恶意请求。CSRF能够做的事情包括：以你名义发送邮件，发消息，盗取你的账号，甚至于购买商品，虚拟货币转账......造成的问题包括：个人隐私泄露以及财产安全。

1. 检查报头中的Referer参数确保请求发自正确的网站（但XHR请求可调用setRequestHeader方法来修改Referer报头）；

2. 对于任何重要的请求都需要重新验证用户的身份；

3. 创建一个唯一的令牌（Token），将其存在服务端的session中及客户端的cookie中，对任何请求，都检查二者是否一致。

### XSS

Cross Site Scripting(跨站脚本攻击)，通过**插入恶意脚本**，实现对用户游览器的控制。

（1）浏览器自身可以识别简单的XSS攻击字符串，从而阻止简单的XSS攻击；

（2）对内容进行转义和过滤。让用户的输入什么就是什么，不要变成程序

（3）csp（内容安全策略），如果一段代码变成了程序，我们是否应该运行它。或者更准确一点说，它实际上是定义页面上哪一些内容是可被信任的，哪一些内容是不被信任的。HTTP 头中输出 CSP 策略：Content-Security-Policy: <policy-directive>; <policy-directive>

## JSON、JSONP、AJAX

1. JSON是一种数据格式，描述结构化数据。不限于JavaScript范畴。可以表示三种类型的值
2. 简单值：字符串、数值、布尔值、null。不支持undefined
3. 数组：数组是一种复杂数据类型，表示一组有序的值的列表。数组的值可以使任意类型——简单值、数组、对象
4. 对象：对象是一种复杂数据类型，表示一组无序的键值对。每个键值对中的值可以使简单类型，也可以是复杂类型。

Var ajson={“name”:’mike’}

注意！！json中的对象要求给属性加双引号，JavaScript中字面量对象属性名不加引号

JSON.stringfy()：将对象序列化为JSON字符串

JSON.parse()：将JSON字符串解析为js对象

1. JSONP是一种非官方跨域数据交互协议。
2. AJAX直接请求普通文件存在跨域无法访问的问题，web页面上调用js文件时不受跨域的影响，凡是拥有“src”属性的标签都有跨域的能力。如果想通过纯web端跨域访问数据，就可以通过在远程服务器上把数据封装进js格式的文件里，供客户端调用和进一步处理。
3. JSONP的工作原理：客户端以回调函数名作为参数添加到URL中，请求服务器端。服务端根据URL中的callback参数动态生成js文件，在js文件中调用callback并把数据传入。客户端在callback函数中对数据进行处理。
4. AJAX和JSONP的异同
5. 两者都是请求一个url，然后把服务器返回的数据进行处理，因此jQuery和ext等框架都把jsonp作为AJAX的一种形式进行了封装
6. AJAX和jsonp本质不同。AJAX的核心是通过XMLHttpRequest获取非本页内容，而jsonp的核心是动态添加<scipt>标签来调用服务器提供的js脚本。
7. AJAX和jsonp的区别不在于是否跨域，AJAX通过服务端代理一样可以实现跨域，jsonp本身也不排斥同域的数据的获取
8. Jsonp是一种方式，不一定非要用json格式来传递数据。

## http1.0与http1.1

1. HTTP1.1支持长连接和请求的流水线处理
   1. HTTP1.0规定浏览器与服务器只保持短暂的连接，浏览器的每次请求都需要与服务器建立一个TCP连接，服务器完成请求处理后立即断开TCP连接，服务器不跟踪每个客户也不记录过去的请求。
   2. HTTP 1.1的持续连接，也需要增加新的请求头来帮助实现，例如，Connection请求头的值为Keep-Alive时，客户端通知服务器返回本次请求结果后保持连接；Connection请求头的值为close时，客户端通知服务器返回本次请求结果后关闭连接。HTTP 1.1还提供了与身份认证、状态管理和Cache缓存等机制相关的请求头和响应头。
2. HTTP 1.1增加host字段
   1. HTTP1.0中认为每台服务器都绑定一个唯一的IP地址，但是随着虚拟主机技术的发展，一台物理机上可能有多个虚拟主机，共享一个IP地址。这样可能存在DNS解析不同站点的请求时解析到同一个IP地址，为了区分不同站点的请求，就需要host来区分不同站点。
   2. HTTP1.1的请求消息和响应消息都支持host头域，若请求消息中没有host头域会报错（400 Bad Request）
3. 100(Continue) Status(节约带宽)

HTTP1.1加入了新状态码100,。客户端先发送一个只带头域的请求，若服务器因权限拒绝了请求，回送响应码401（unauthorized）；如果服务器收到请求后回送响应码100，客户端就可以继续发送带实体的完整请求了。

1. HTTP1.1引入Chunked transfer-coding，将消息分割成任意大小的数据块，每个数据块在发送时都会附上块的长度，最后用一个零长度的块作为消息结束的标志。此方法允许发送方只缓冲消息的一个片段，避免缓冲整个消息带来的过载。
2. HTTP1.1在1.0的基础上加入了一些cache的新特性，当缓存对象的Age超过Expire（生命时长）时变成stale对象，不需要抛弃stale对象，而是与源服务器重新激活

## http、https和http2

#### http

无状态协议，不保留请求的状态。应用层的协议。

缺点：

1. 通信使用明文（不加密），内容可能会被窃听
2. 不验证通信方身份，因此有可能遭遇伪装
3. 无法证明报文的完整性，所以可能已遭篡改

优点：

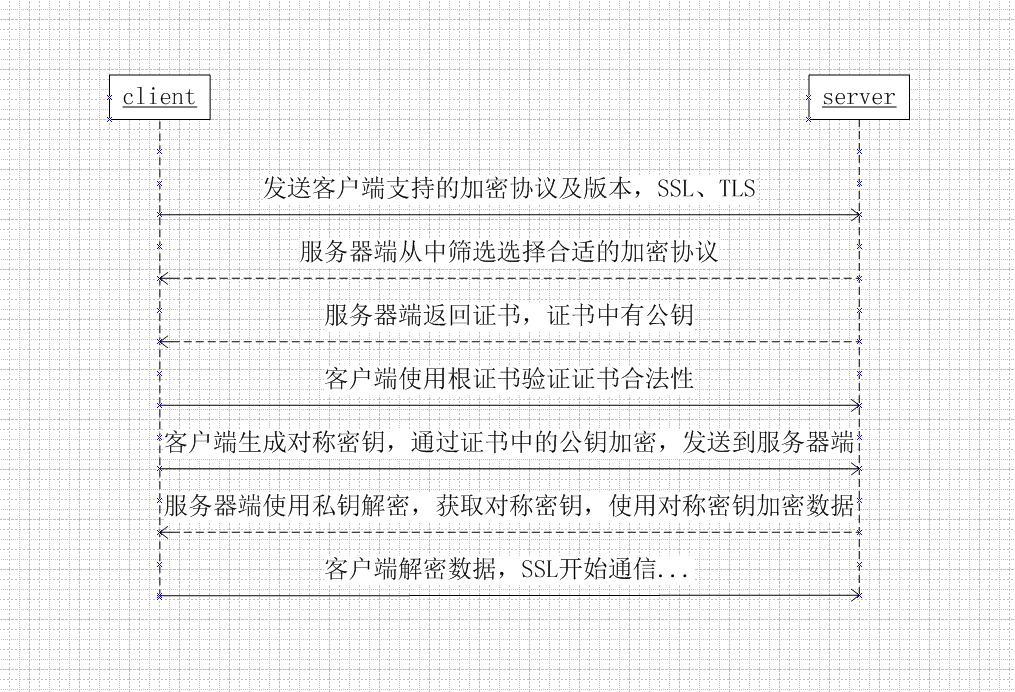
1. 传输速度快

#### https

https并非是应用层的一种新协议，只是http通信接口部分用ssl（安全套接字层）和tls（安全传输协议）代替而已。即添加了加密以及认证机制的HTTP称为https（HTTP Secure）。

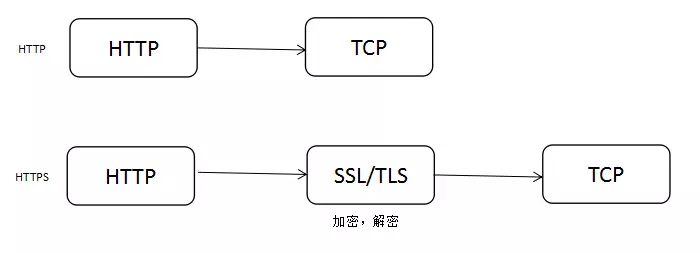
HTTP+加密+认证+完整性保护=HTTPS

https通信过程：



#### https与http的些区别

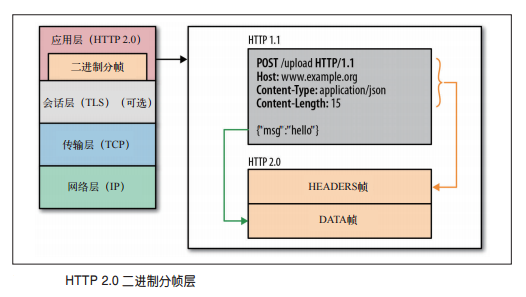
1. HTTPS协议需要到CA（证书的签发机构）申请证书，一般免费证书很少，需要交费。
2. HTTP协议运行在TCP之上，所有传输的内容都是明文，HTTPS运行在SSL/TLS之上，SSL/TLS运行在TCP之上，所有传输的内容都经过加密的。
3. HTTP和HTTPS使用的是完全不同的连接方式，用的端口也不一样，前者是80，后者是443。
4. HTTPS可以有效的防止运营商劫持，解决了防劫持的一个大问题。



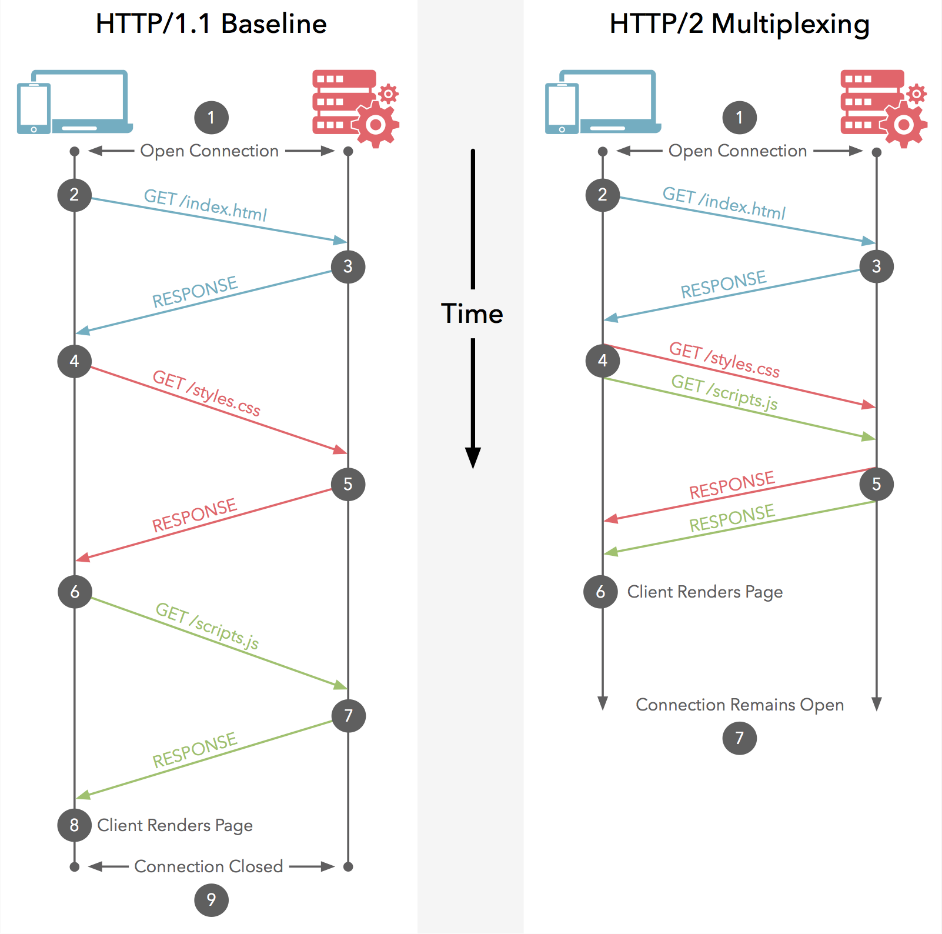
TLS/SSL中使用了非对称加密（RSA、DSA/DSS），对称加密（AES、3DES）以及HASH算法（MD5、SHA1、SHA256）

#### http2的新特性

1. 增加二进制分帧。HTTP2.0将所有传输的信息分割成更小的消息和帧，并对他们采用二进制格式的编码。http1.x的解析基于文本。二进制更健壮



1. 多路复用。所有HTTP2.0通信都在一个连接上完成，这个链接可以承载任意数量的双向数据流。每个数据流以消息的形式发送，而消息由一个或多个帧组成，这些帧可以乱序发送，再根据每个帧首部的流标识符重新组装。

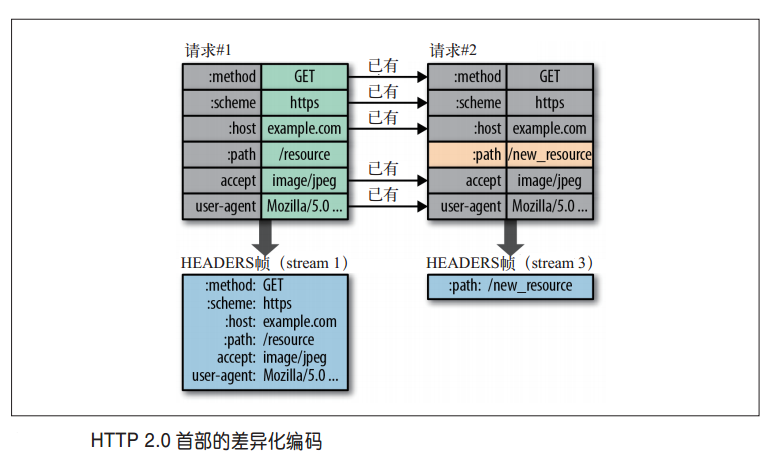


前端的一些性能优化方法：

1. CSS雪碧图合并-减少请求
2. 合并压缩css和js代码-减少请求
3. CSS代码放在header头里面，JS代码放到body结束之前，因为JS代码执行会阻塞。

对于HTTP1.1而言，浏览器通常有并行连接的限制，即最多几个并行连接，而多路复用通过单一的HTTP 2.0连接发起多重的请求-相应消息，这意味着HTTP 2.0的通信都在一个连接上完成了，这个连接可以承载任意数量的双向数据流。直观来说，就是上面我们所做的优化已经不需要了。

1. Header压缩。HTTP2.0在客户端和服务端使用首部表来跟踪和存储之间发送的键值对，对相同请求而言不需要再次发送和响应，通信期间几乎不会改变的通用键值只发送一次，相当于做了一层缓存。
   1. 如果请求不包含头部，对于同一资源的轮询请求，首部开销为0字节
   2. 如果首部发生变化，只需发送变化的数据在header帧里面，新增或修改的首部帧会被追加到首部表



1. 服务端推送。HTTP2.0新增服务器提示，先于客户端检测到将要请求的资源，提前通知客户端，服务器不发送所有资源的实体，只发送资源的URL，客户端接到提示后会进行验证缓存，如果真需要这些资源，则正式发起请求（服务器主动更新静态资源）。
2. 请求优先级。HTTP消息分解成多个独立帧后，就可以通过优化帧的交错和传输顺序，进一步提升性能。可以给每一个流带有一个31bit的优先值。

## CDN

CDN：Content Delivery Network，内容分发网络。基本思路是尽可能避开互联网上所有可能影响数据传输速度和稳定性的瓶颈和环节，使内容传输的更快、更稳定。

工作原理：通过在网络各处放置**节点服务器，**在现有互联网基础上构成一层智能**虚拟网络**，CDN能够实时的根据网络流量和各节点的连接、负载状况以及到用户的距离和响应时间等综合信息，将用户的请求重新导向到离用户最近的服务节点上。（通过权威服务器实现优质节点的选择，通过缓存减少源站的压力）

目的：使用户可以就近取得所需内容，解决网络拥挤的状况，提高用户访问网站的响应速度。

## HTML5的离线存储

使用 HTML5，通过创建**cache manifest**文件，可以轻松地创建 web 应用的离线版本。

HTML5引入了应用程序缓存，这意味着 web 应用可进行缓存，并可在没有因特网连接时进行访问。 应用程序缓存为应用带来三个优势：

* 离线浏览 – 用户可在应用离线时使用它们
* 速度 – 已缓存资源加载得更快
* 减少服务器负载 – 浏览器将只从服务器下载更新过或更改过的资源。

## 跨域资源共享CORS

CORS是一个W3C标准，“跨域资源共享”，允许浏览器向跨源服务器发出XMLHttpRequest请求，克服了AJAX只能同源（同协议、同域名、同端口）使用的限制。

思想：使用自定义的HTTP头部让浏览器和服务器沟通，从而决定请求或相应的成败

CORS需要浏览器和服务器同时支持。目前，所有浏览器都支持该功能，IE浏览器不能低于IE10。整个CORS通信过程由浏览器自动完成。对于开发者而言，CORS通信与同源的AJAX通信没有差别，代码完全一样。浏览器一旦发现AJAX请求跨源，就会添加一些附加的头信息，有时会多出一次附加请求，但是用户不会有感觉。

## 跨域的实现方式

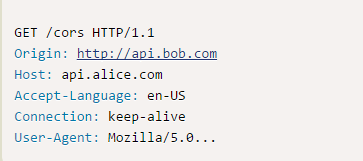
由于浏览器同源策略，凡是发送请求url的协议、域名、端口三者之间任意一个与当前页面地址不同即为**跨域**。

### CORS

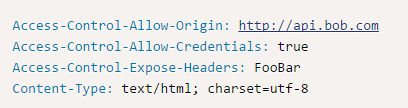
需要浏览器和服务器同时支持。CORS通信过程，是浏览器自动完成。浏览器会对AJAX跨域请求自动附加头信息。关键是服务器端实现CORS接口。

CORS请求分为两种：

* 1. 简单请求
     + 方法为：HEAD、GET、POST
     + HTTP头信息不超过：
       - Accept
       - Accept-Language
       - Content-Language
       - Last-Event-ID
       - Content-Type：只限于三个值application/x-www-form-urlencoded、multipart/form-data、text/plain
     + CORS流程：
       - 浏览器在HTTP头信息中添加Origin字段，说明请求来自的源（协议+域名+端口）



* + - * Origin指定的域名在许可范围内，服务端返回响应中包含：

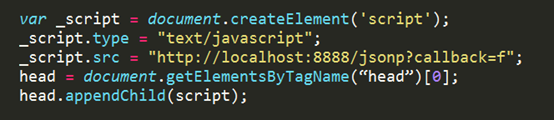
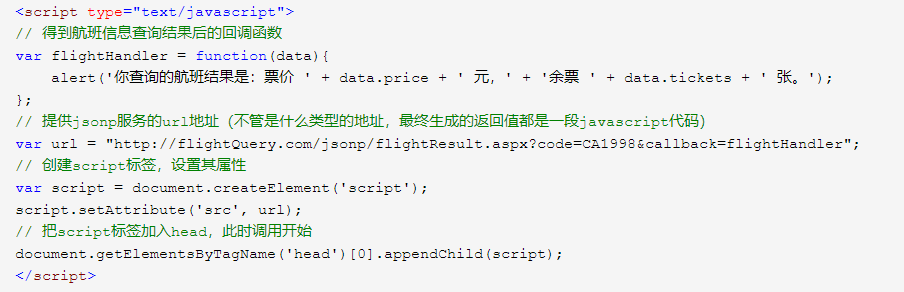


* + - * + Access-Control-Allow-Origin：必须。请求时的Origin字段，或者\*
        + Access-Control-Allow-Credentials：可选。是否允许发送Cookie，默认CORS请求不包含Cookie
        + Access-Control-Expose-Headers：可选。XHR处理6个基本字段外能获得的字段值。6个基本字段（Cache-Control、Content-Language、Content-Type、Expires、Last-Modified、Pragma）
  1. 非简单请求

在正式通信前，增加一次HTTP查询请求，浏览器先询问服务器，该网页的域名是否在服务器的许可名单。

### JSONP

这种方式主要是通过动态插入一个script标签。浏览器对script的资源引用没有同源限制，同时资源加载到页面后会立即执行（没有阻塞的情况下）。



缺点：

* 1. JSONP是从其他域中加载脚本，无法保证安全性
  2. 无法确定JSONP请求是否失败

### 通过iframe+document.domain来跨子域

同源策略的限制：

1. 不能通过ajax的方法去请求不同源中的文档。
2. 浏览器中不同域的框架之间是不能进行js的交互操作的。有一点需要说明，不同的框架之间（父子或同辈），是能够获取到彼此的window对象的，但是不能使用获取到的window对象的属性和方法(html5中的 postMessage方法是一个例外，还有些浏览器比如ie6也可以使用top、parent等少数几个属性)，总之，你可以当做是只能获取到一个几乎 无用的window对象。

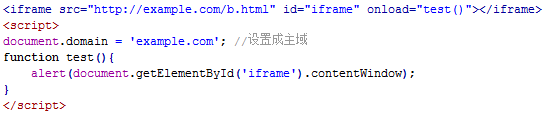
比如，有一个页面，它的地址是<http://www.example.com/a.html>  ， 在这个页面里面有一个iframe，它的src是<http://example.com/b.html>, 很显然，这个页面与它里面的iframe框架是不同域的，所以我们无法通过在页面中书写js代码来获取iframe中的东西



解决方法：

将http://www.example.com/a.html 和 http://example.com/b.html这两个页面的document.domain都设成相同的域名就可以了。只能把document.domain设置成自身或更高一级的父域，且主域必须相同。

首先，在页面 <http://www.example.com/a.html> 中设置document.domain:



在页面 <http://example.com/b.html> 中也设置document.domain，而且这也是必须的，虽然这个文档的domain就是example.com,但是还是必须显示的设置document.domain的值：



这样我们就可以通过js访问到iframe中的各种属性和对象了。

 但是在http://www.example.com/a.html 页面中通过ajax直接请求http://example.com/b.html 页面，即使你设置了相同的document.domain也还是不行的，所以修改document.domain的方法只适用于不同子域的框架间的交互。

若要通过ajax方法去与不同子域的页面交互，除了jsonp外，还可以使用一个隐藏的iframe做一个代理（C）。让C载入一个与ajax请求的目标页面同域的页面，所以C可以正常ajax请求b.html获取数据，再通过document.domain，js控制C去ajax请求b.html的数据，就可以获得数据了

### 使用HTML5中新引进的window.postMessage方法来跨域传送数据

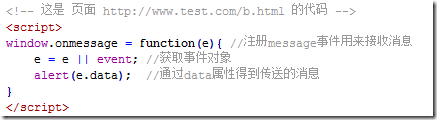
window.postMessage(message,targetOrigin)  方法是html5新引进的特性，可以使用它来向其它的window对象发送消息，无论这个window对象是属于同源或不同源，目前IE8+、 FireFox、Chrome、Opera等浏览器都已经支持window.postMessage方法。

window.postMessage（data，targetorigin），该window对象是指要发送消息的那一个window对象。参数：

1. data：要传递的数据。需要JSON.stringfy()对对象参数序列化；
2. targetOrigin：字符串参数。指定目标窗口的源，协议+主机+端口号[+URL]，URL会被忽略，所以可以不写，只是为了安全考虑，postMessage()方法只会将message传递给指定窗口，当然也可以将参数设置为"\*"，这样可以传递给任意窗口，如果要指定和当前窗口同源的话设置为"/"。

需要接收消息的window对象，可是通过监听自身的message事件来获取传过来的消息，消息内容储存在该事件对象的data属性中。





### 图片ping(单向的，只能发不能收)

图片可以从任何URL中加载，所以将img的src设置成其他域的URL，即可以实现简单的跨域，可以使用onload和onerror事件来确定是否接受到了响应。

使用图片ping跨域只能发送get请求，并且不能访问响应的文本，只能监听是否响应而已，可以用来追踪广告点击。

## HTTP POST和GET方法

### GET 方法

从指定的资源请求数据；

请注意，查询字符串（名称/值对）是在 GET 请求的 URL 中发送的：

/test/demo\_form.asp?name1=value1&name2=value2；

### POST 方法

向指定的资源提交要被处理的数据

请注意，查询字符串（名称/值对）是在 POST 请求的 HTTP 消息主体中发送的：

POST /test/demo\_form.asp HTTP/1.1；

Host: w3schools.com；             name1=value1&name2=value2；

### 比较 GET 与 POST

|  |  |  |
| --- | --- | --- |
|  | **GET** | **POST** |
| 后退按钮/刷新 | 无害 | 数据会被重新提交。 |
| 书签 | 可收藏为书签 | 不可收藏为书签 |
| 缓存 | 能被缓存 | 不能缓存 |
| 编码类型 | application/x-www-form-urlencoded | application/x-www-form-urlencoded 或 multipart/form-data。为二进制数据使用多重编码。 |
| 历史 | 参数保留在浏览器历史中。 | 参数不会保存在浏览器历史中。 |
| 对数据长度的限制 | 是的。当发送数据时，GET 方法向 URL 添加数据；URL 的长度是受限制的（URL 的最大长度是 2048 个字符）。 | 无限制。 |
| 对数据类型的限制 | 只允许 ASCII 字符。 | 没有限制。也允许二进制数据。 |
| 安全性 | 与 POST 相比，GET 的安全性较差，因为所发送的数据是 URL 的一部分。在发送密码或其他敏感信息时绝不要使用 GET ！ | POST 比 GET 更安全，因为参数不会被保存在浏览器历史或 web 服务器日志中。 |
| 可见性 | 数据在 URL 中对所有人都是可见的。 | 数据不会显示在 URL 中。 |

|  |  |
| --- | --- |
| **方法** | **描述** |
| HEAD | 与 GET 相同，但只返回 HTTP 报头，不返回文档主体。 |
| PUT | 上传指定的 URI 表示。 |
| DELETE | 删除指定资源。 |
| OPTIONS | 返回服务器支持的 HTTP 方法。 |
| CONNECT | 把请求连接转换到透明的 TCP/IP 通道。 |

## http请求头，请求体，cookie在哪个里面？url在哪里面？

参考：https://www.jianshu.com/p/80e25cb1d81a

### HTTP协议

Hyper Transfer Protocol（超文本传输协议）。用于从万维网（WWW:World Wide Web ）服务器传输超文本到本地浏览器的传送协议。

HTTP基于TCP/IP通信协议来传递数据（HTML 文件, 图片文件, 查询结果等）。

### HTTP协议原理

HTTP协议工作于客户端-服务端（C/S）架构上。浏览器作为HTTP客户端通过URL向HTTP服务端即Web服务器（Apache服务器，IIS服务器）发送所有请求。Web服务器根据收到的请求向客户端发送响应信息。HTTP默认端口为80，可自行修改。客户端与服务器端都是应用程序。

URI（Uniform Resource Identifiers，统一资源标识符）用来传输数据和建立连接。URL是一种特殊类型的URI，包含了用于查找某个资源的足够的信息

URL,全称是UniformResourceLocator, 中文叫统一资源定位符，是互联网上用来标识某一处资源的地址。

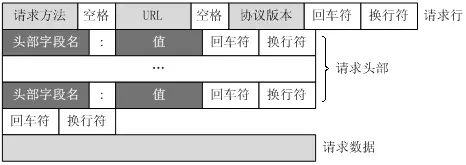
建立连接后，数据消息通过类似Internet邮件所使用的格式【RFC5322】和多用途Internet邮件扩展（MIME）【RFC2045】来传送

HTTP三点注意事项：

1. HTTP是无连接的：无连接是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间
2. HTTP是媒体独立的：任何类型的数据都可以通过HTTP传送，客户端以及服务器指定使用适合的MIME-type内容类型
3. HTTP是无状态协议：无状态是指协议对于事务处理没有记忆能力，如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大，但是，在服务器不需要先前信息时它的应答就较快

### HTTP请求报文

四部分：请求行、请求头、空行、请求数据，各占一行



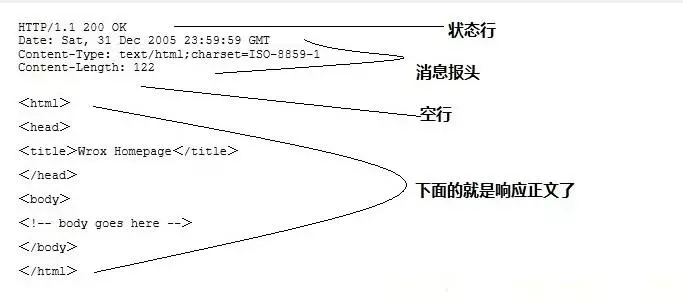
例子：





1. 请求行：
   1. 请求方法：
      1. HTTP1.0定义了三种请求方法： GET, POST 和 HEAD方法；
      2. HTTP1.1新增了五种请求方法：OPTIONS, PUT, DELETE, TRACE 和 CONNECT 方法
   2. 请求资源的URI路径
   3. HTTP版本号

### HTTP响应报文

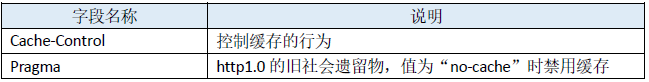


### HTTP首部字段

1. HTTP首部字段传递重要信息
2. 首部字段名：字段值
3. 4种HTTP首部字段类型
   1. 通用首部字段：请求报文和响应报文两方都会使用首部
   2. 请求首部字段：补充了请求的附加内容、客户端信息、响应内容相关优先级等信息。
   3. 响应首部字段：补充了响应的附加内容，也会要求客户端附加额外的内容信息。
   4. 实体首部字段：针对请求报文和响应报文的实体部分使用的首部。补充了资源的内容更新时间等与实体有关的信息。

## http中和缓存相关的字段

1. 通用首部字段（请求报文和响应报文都能用上的字段）



1. 请求首部字段

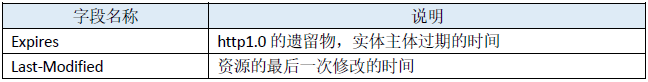


1. 响应首部字段

https://images2015.cnblogs.com/blog/561179/201604/561179-20160401161311394-1246877214.png

1. 实体首部字段

实体首部字段是包含在请求报文和响应报文中的的实体部分所使用的首部，用于补充内容的更新时间等实体相关的信息。



禁止缓存的方式：

* + 1. 客户端决定。HTML中使用meta标签，设置Pragma、expires、Cache-Control
       1. <meta http-equiv="Pragma" content="no-cache">仅有IE能识别，其他浏览器只能识别“Cache-Control:no-store”
       2. <meta http-equiv="expires" content="mon, 18 apr 2016 14:30:00 GMT">仅有IE能识别，只对页面有效，对页面上的资源无效
       3. Cache-Control: max-age=3600, must-revalidate
    2. 服务端决定。Last-Modified、ETag

## HTTP状态码知道哪些？

100 Continue 继续，一般在发送post请求时，已发送了http header之后服务端将返回此信息，表示确认，之后发送具体参数信息

　　200 OK 正常返回信息

　　201 Created 请求成功并且服务器创建了新的资源

202 Accepted 服务器已接受请求，但尚未处理

204 No Content 无内容

　　301 Moved Permanently 请求的网页已永久移动到新位置。

　　302 Found 临时性重定向。

　　303 See Other 临时性重定向，且总是使用 GET 请求新的 URI。

　　304 Not Modified 自从上次请求后，请求的网页未修改过，使用缓存的资源

　　400 Bad Request 服务器无法理解请求的格式，客户端不应当尝试再次使用相同的内容发起请求。

　　401 Unauthorized 请求未授权。

　　403 Forbidden 禁止访问。

　　404 Not Found 找不到如何与 URI 相匹配的资源。

　　500 Internal Server Error 最常见的服务器端错误。

503 Service Unavailable 服务器端暂时无法处理请求（可能是过载或维护）。

|  |  |
| --- | --- |
| **分类** | **分类描述** |
| 1\*\* | 信息，服务器收到请求，需要请求者继续执行操作 |
| 2\*\* | 成功，操作被成功接收并处理 |
| 3\*\* | 重定向，需要进一步的操作以完成请求 |
| 4\*\* | 客户端错误，请求包含语法错误或无法完成请求 |
| 5\*\* | 服务器错误，服务器在处理请求的过程中发生了错误 |

##### 301和302的区别

两者都表示重定向，浏览器在获取服务器返回的状态码后自动跳转到新的URL。

301：旧地址的资源被永久移除了，**搜索引擎在抓取新内容的同时将旧的网址替换为重定向之后的网址**。

302：暂时转移，旧地址A的资源还在（仍然可访问），**搜索引擎会抓取新的内容而保存旧的网址。**

**301和302的选择：**

当一个网站或者网页24—48小时内临时移动到一个新的位置，这时候就要进行302跳转，打个比方说，我有一套房子，但是最近走亲戚去亲戚家住了，过两天我还回来的。而使用301跳转的场景就是之前的网站因为某种原因需要移除掉，然后要到新的地址访问，是永久性的，就比如你的那套房子其实是租的，现在租期到了，你又在另一个地方找到了房子，之前租的房子不住了。

**尽量只用301跳转！！！——避免网址劫持**。

**网址URL劫持**：一个不道德的人在他自己的网址A做一个302重定向到你的网址B，出于某种原因， Google搜索结果所显示的仍然是网址A，但是所用的网页内容却是你的网址B上的内容。

**重定向**：通过各种方法将各种网络请求重新定个方向转到其他位置。

**何时进行重定向**：（1）网站调整（改变网页目录结构）（2）网页被移到一个新地址（3）网页扩展名改变（例如.php改成.html）。如果不做重定向，则用户收藏夹或搜索引擎数据库中旧地址只能让用户得到404的错误信息；注册了多个域名的网站，也需要重定向让让问这些域名的用户自动跳转到主站点。

##### 200和304的区别

200 OK：请求成功，第一次请求资源被发送回客户端，默认状态下200的响应可以被缓存。之后的请求时浏览器没有跟服务器确认，直接使用了浏览器的缓存。

304：内容没有变，可以使用缓存的内容。浏览器和服务器多确认了一次缓存有效性，再用的缓存。

触发时机：200是直接点击链接访问，输入网址回车也能触发；304是刷新页面，或是设置了长缓存、但Entity Tags没有移除时触发。

## Last-Modified与Etag

**Etag**：在静态资源文件后面添加一个唯一的参数（相当于查询字符串），由服务器端生成，并且随着文件的改变而改变，浏览器端只重新请求Etag发生变化的文件，减少浏览器端数据的流量，减轻服务器端的压力。

**Last-Modified**：在浏览器第一次请求一个URL时，服务器返回200，内容是请求的资源，同时有Last-Modified属性标记此文件在服务器端最后被修改的时间。

格式如下：

Last-Modified: Fri, 12 May 2006 18:53:33 GMT

　　客户端第二次请求此URL时，根据 HTTP 协议的规定，浏览器会向服务器传送 If-Modified-Since 报头，询问该时间之后文件是否有被修改过：

　　If-Modified-Since: Fri, 12 May 2006 18:53:33 GMT

如果服务器端的资源没有变化，则自动返回 HTTP 304 （Not Changed.）状态码，内容为空，这样就节省了传输数据量。

**有了Last-Modified，为什么还要用ETag？**

1. 因为如果在一秒钟之内对一个文件进行两次更改，Last-Modified就会不正确。
2. 某些服务器不能精确的得到文件的最后修改时间。
3. 一些文件也许会周期性的更改，但是他的内容并不改变(仅仅改变的修改时间)，这个时候我们并不希望客户端认为这个文件被修改了，而重新GET。

**有了Etag，为什么还要使用Last-Modified？**

1. 对某些静态资源的修改，例如图片，如果每次扫描内容生成ETag，比直接修改要慢。

Etag是通过对文件的索引节、大小和最后修改时间进行Hash后得到的。

## MVC模式

MVC：Module、View、Controller，是一种软件设计典范，用一种逻辑、数据、界面显示分离的方法组织代码。

**Module（模型）**是应用程序中用于处理应用程序数据逻辑的部分。通常模型对象负责在数据库中存取数据。

**View（视图）**是应用程序中处理数据显示的部分。通常视图是依据Module数据创建的。

**Controller（控制器）**是应用程序中处理用户交互的部分。通常控制器负责从View读取数据，控制用户输入，并向Module发送数据。

## 对浏览器内核的理解

分成两部分：渲染引擎（Layout engineer或Rendering Engine）和js引擎

**渲染引擎：**负责取得网页的内容（HTML、XML、图像等等）、整理讯息（例如加入CSS等），以及计算网页的显示方式，然后会输出至显示器或打印机。

浏览器内核的不同对于网页的语法解释会有不同，所以渲染的效果也不相同。所有网页浏览器、电子邮件客户端以及其他需要编辑、显示网络内容的应用程序都需要内核

**JS引擎：**解析和执行javascript来实现网页的动态效果。

最开始渲染引擎和JS引擎并没有区分的很明确，后来JS引擎越来越独立，内核就倾向于只指渲染引擎。

## 常见的浏览器内核

JavaScript引擎是SpiderMonkey。

1. Trident（[‘traɪd(ə)nt]）：IE6~IE10
2. Gecko([‘gekəʊ])：FireFox内核
3. Presto([‘prestəʊ])：Opera之前用
4. WebKit：遨游、Safari、Symbian、Android
5. Blink：Chrome、Opera、Yandex

移动端：

移动端的浏览器内核主要指系统内置浏览器的内核。

iOS平台：Webkit；

Android4.4之前：Webkit

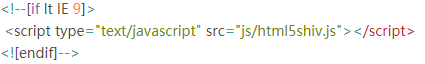
Android4.4：浏览器切换到了Chromium，内核是Webkit的分支Blink

Windows Phone8：Trident

## 谈一下浏览器兼容性

由于不同的浏览器内核对于HTML、CSS、JS的解析结果不同，导致页面在不同的浏览器上存在兼容性问题。在开发的时候会明确项目要兼容哪些浏览器的最低版本，在开发中就是要考虑到CSS样式和JavaScript在这些浏览器的兼容性了

1. **HTML**
   1. H5新标签在IE9以下的浏览器无法识别。
      1. 通过document.createElement(标签名)
      2. IE条件注释语句，添加html5shiv.js库



1. **CSS样式兼容性**
   1. css的hack问题：主要针对IE的不同版本，不同的浏览器的写法不同

IE的条件注释hack：

<!--[if IE 6]>此处内容只有IE6.0可见<![endif]-->

<!--[if IE 7]>此处内容只有IE7.0可见<![endif]-->

* 1. IE双边距问题：IE6在浮动后，又有横向的margin，此时，该元素的外边距是其值的2倍。

解决办法：display:block;

* 1. IE下图片的下方有空隙

解决办法：给img设置display:block;

* 1. IE下两个float之间会有3px

解决办法：给右边的元素也设置float：left

* 1. IE6下没有min-width的概念，其默认的width就是min-width
  2. IE6下在使用margin:0 auto;无法使其居中

解决办法：为其父容器设置text-align:center;

* 1. 被点击过后的超链接不再具有hover和active属性

解决办法:按lvha的顺序书写css样式，  
 ":**l**ink": a标签还未被访问的状态；  
 ":**v**isited": a标签已被访问过的状态；  
 ":**h**over": 鼠标悬停在a标签上的状态；  
 ":**a**ctive": a标签被鼠标按着时的状态；

* 1. 在使用绝对定位或者相对定位后，IE中设置z-index失效，原因是因为其元素依赖于父元素的z-index，但是父元素默认为0， 子高父低，所以不会改变显示的顺序
  2. IE6下无法设置1px的行高，原因是由其默认行高引起的

解决办法：为其设置overflow:hidden;或者line-height:1px

10）ul标签内外边距问题。ul标签在IE6\IE7中，有个默认的外边距，但是在IE8以上及其他浏览器中有个默认的内边距。解决方法：统一设置ul的内外边距为0

1. **JavaScript的兼容性**
   1. 标准的事件绑定方法函数为addEventListener，但IE下是attachEvent；
   2. 事件的捕获方式不一致，标准浏览器是由外至内，而IE是由内到外，标准是按照IE，由内向外
   3. window.event获取目标元素的方法不同，标准浏览器是event.target，而IE下是event.srcElement
   4. 在低版本的IE中获取的日期处理函数的值不是与1900的差值，但是在高版本的IE中和标准浏览器保持了一致，获取的值也是与1900的差值。

var year= new Date().getYear();

* 1. ajax的实现方式不同。获取XMLHttpRequest的不同，IE下是activeXObject
  2. IE中不能操作tr的innerHtml
  3. 获得DOM节点的父节点、子节点的方式不同

其他浏览器：parentNode parentNode.childNodes

IE：parentElement parentElement.children

## 性能优化的方法？

（1） 减少http请求次数：CSS Sprites, JS、CSS源码压缩、图片大小控制合适；网页Gzip，CDN托管，data缓存 ，图片服务器。

（2） 前端模板 JS+数据，减少由于HTML标签导致的带宽浪费，前端用变量保存ajax请求结果，每次操作本地变量，减少请求次数

（3） 用innerHTML代替DOM操作，减少DOM操作次数，优化javascript性能。

（4） 当需要设置的样式很多时设置className而不是直接操作style。

（5） 少用全局变量、缓存DOM节点查找的结果。减少IO读取操作。

（6） 避免使用CSS Expression，expression把Javascript脚本写放在css文件中，通过它来实现一些很方便的功能与效果。例如：#mydiv{left: expression(document.body.offsetWidth - 180 "px");}

（7） 图片预加载，将样式表放在顶部，将脚本放在底部 加上时间戳。

## 优雅降级和渐进增强

**优雅降级：**一开始就构建站点的完整功能，然后针对浏览器测试和修复。比如一开始使用 CSS3 的特性构建了一个应用，然后逐步针对各大浏览器进行 hack 使其可以在低版本浏览器上正常浏览

**渐进增强：**一开始就针对低版本浏览器进行构建页面，完成基本的功能，然后再针对高级浏览器进行效果、交互、追加功能达到更好的体验。

## 前端路由

**路由**是用来和后端服务器进行交互的一种方式，通过不同的路径请求不同的资源，请求不同的页面是路由的其中一种应用。

对用户来说，路由就是浏览器地址栏中的url与所见网页的对应关系。而对于web开发人员来说，路由更像是url与处理函数的对应关系。

**服务端路由**： 根据用户请求的不同网址，返回不同的网页内容。一是会造成服务器压力增加，二是每次都重新请求，响应较慢、用户体验下降。

**前端路由**：根据URL的改变，通过js对DOM操作实现不同UI之间的切换，不动态刷新页面或者跳转到新页面。只通过ajax向服务器端请求数据，对用户来说是无刷新的、即时响应，是更好的体验。

MVVM：Model View Module-View

## 前端路由的两种实现方式

###### History API

监听popState事件，用history.pushState 和 history.replaceState实现

1. **history.pushState和history.replaceState**

history.pushState()和history.replaceState()，用来在浏览历史中添加和修改记录，两者的的参数相同：

状态对象（state）：和创建的新历史记录相关联。

标题：给state传一个短标题。大多数浏览器会忽略

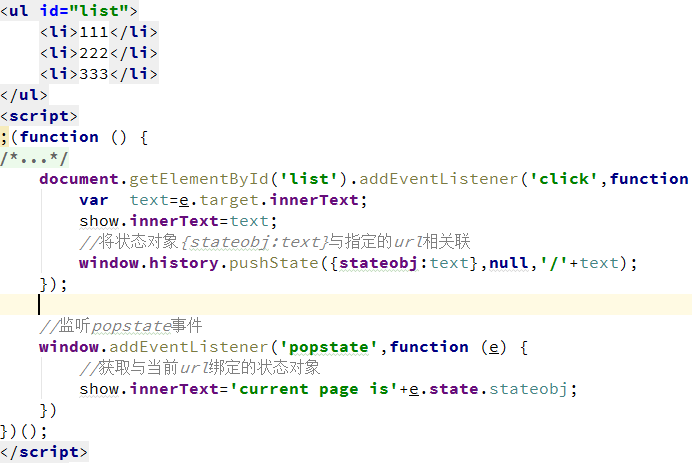
url：新的历史记录条目的地址。浏览器不会在调用pushState()方法之后加载该地址，但是当重启浏览器时会加载。若是相对路径，以当前url为基准；传入的url与当前url必须**同源**。该参数可选，不指定则是文档当前url。

两个API都是操作浏览器的历史记录，而不会引起页面的刷新。

pushState增加一条新的历史记录，replaceState会替换当前的历史记录。

**2、popState事件**

当同一个文档的浏览历史（即history对象）出现变化时，触发popState事件。pushState和replaceState方法不会触发，只有点击浏览器的倒退和前进按钮，或者使用js调用back、forward、go方法时才会触发。



###### Hash

改变hash值，监听hashchange事件。可以兼容低版本浏览器。

url中的#分为两种：

（1）锚点，例如回到顶部、页面中从某一部分到另一部分；

（2）路由中的hash。

Location.hash的值是#后的内容，例如http://www.163.com#netease此网址中，location.hash='#netease'。

**Hash的以下特性使其可以实现前端路由**：

1. url中hash值的变化不会重新加载页面。因为hash是用来指导浏览器行为的，对服务器端无用，不会包含在http请求中。
2. hash值的改变都会在浏览器的历史记录中增加一个记录，即能够通过浏览器的回退、前进按钮控制hash切换
3. 通过hashchange事件，监听到hash值变化，从而响应不同路径的逻辑处理

**触发hash值变化的方法**：

1. a标签的href属性。当标签点击后，地址栏会改变，同时触发hashchange事件

<a href="#kaola">to KAOLA</a>

1. 通过js直接赋值给location.hash，会改变url，触发hashchange事件

location.hash="#kaola"

###### 两种方式对比

1. 基于Hash的路由，兼容性更好，但是url必须以#开头；基于History API的路由，则更正式，可以设置与当前URL同源的任意URL，路径更直观。
2. 基于Hash的路由不需要对服务器做改动，基于History API的路由需要对服务器做一些改造，配置不同的路由都返回相同的页面。

## 前端路由和服务端路由的区别

**前端路由**：每跳转到不同的URL都是使用前端的hash路由，实际上只是JS根据URL的变化操作DOM元素，根据每个页面需要的去服务端请求数据，返回数据后和模板进行组合。

**服务端路由**：每跳转到不同的URL，都是重新访问服务端，然后服务端返回页面。页面也可以是服务端获取数据然后和模板组合，返回HTML；也可以是直接返回HTML模板，再由前端JS再去请求数据，使用前端模板和数据进行结合，生成想要的HTML

如何选择：

1. 前端路由：单页面应用，大部分页面结构不变，只改变部分内容的使用
2. 服务端路由：功能复杂的网站

## 前后端分离的意义

前后端分离：后台只需提供API接口，前端调用AJAX实现数据呈现

1. 解放前端。

前端不需要向后台提供模板或是后台在前端html中嵌入后台代码

1. 分工明确，提高工作效率。
2. 局部性能提升

通过前端路由的配置，可以实现页面的按需加载，不需要一开始加载首页便加载完网站的所有资源，服务器也不再需要解析前端页面，在页面交互以及用户体验上都有所提升。

1. 降低维护成本

通过目前主流的MVC框架，可以快速定位问题，代码重构及可维护性增强。

## 前端工程化

前端工程化是根据业务特点，将前端开发流程规范化，标准化，包括开发流程、技术选型、代码规范、构建发布等，用于提升前端工程师的开发效率和代码质量。

工程化：有个方法，让一大堆人，有组织有纪律地一起干活，目的是提高效率，保证质量。

可以从模块化、组件化、规范化、自动化四个方面展开：

* 1. **模块化**

将一个大文件拆分成相互依赖的小文件，再进行统一的拼装和加载。只有这样，才能多人协作。

* + - **JS的模块化**

ES6之前，JavaScript没有模块系统，模块加载方案CommonJS、AMD、CMD等

ES6规定了模块系统，可以取代现有的CommonJS、AMD、CMD规范

模块的打包和加载：

1. Webpack+Babel：将所有模块打包成一个文件同步加载，也可以打成多个chunk异步加载；
2. SystemJS+Babel主要是分模块异步加载；
3. 浏览器的<script type="module">加载

目前Webpack较流行。Safari支持第三种

* + - **CSS的模块化**

虽然SASS、LESS、Stylus等预处理器实现了CSS的文件拆分，但是没有解决CSS模块化带来的**选择器全局污染问题**。（导入一个CSS模块后，已存在的样式有被覆盖的风险）

工具：Shadow DOM、CSS in JS和CSS Modules

1. Shadow DOM：不兼容
2. CSS in JS：彻底抛弃CSS，使用JS或JSON写样式，处理伪类困难
3. CSS Modules：JS管理依赖，最大化的结合CSS生态和JS模块化能力。
   * + **资源的模块化**

Webpack的万能模块加载理念：所有的资源都可以且应该模块化

资源模块化的好处：

1. 依赖关系单一化。所有CSS和图片等资源的依赖关系统一走JS路线，无需额外处理CSS预处理器的依赖关系，也不需处理代码迁移时的图片合并、字体图片等路径问题；
2. 资源处理集成化。现在可以用loader对各种资源做各种事情，比如复杂的vue-loader等等。
3. 项目结构清晰化。使用Webpack后，你的项目结构总可以表示成这样的函数：dest = webpack(src, config)
   1. **组件化**
      * **组件化≠模块化。**

模块化是文件层面上对代码或资源拆分；组件化是设计层面，对UI（用户界面）的拆分。

组件：**每个包含模板(HTML)+样式(CSS)+逻辑(JS)功能完备的结构单元**

组件化实际上是一种按照模板(HTML)+样式(CSS)+逻辑(JS)三位一体的形式**对面向对象的进一步抽象**。

**除了封装组件本身，还要合理处理组件之间的关系，比如（逻辑）继承、（样式）扩展、（模板）嵌套和包含等，这些关系都可以归为依赖。**

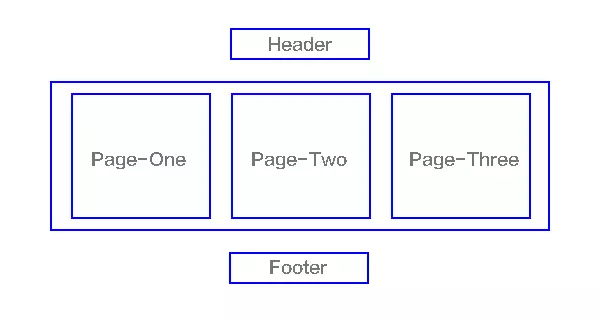
* 1. **规范化**

模块化和组件化确定了开发模型，而这些东西的实现就需要规范去落实。

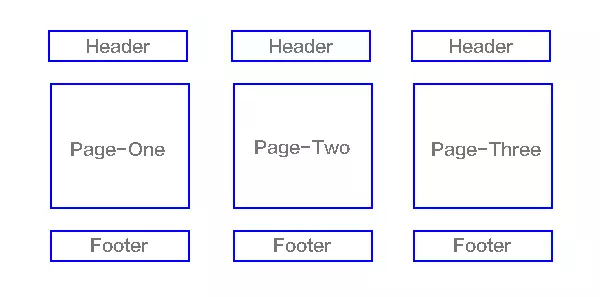
* 目录结构的制定
* 编码规范
* 前后端接口规范
* 文档规范
* 组件管理
* Git分支管理
* Commit描述规范
* 定期CodeReview
* 视觉图标规范
* ...
  1. **自动化**
* 图标合并
* 持续集成
* 自动化构建
* 自动化部署
* 自动化测试：使用测试框架，例如Karma + Mocha + Chai

## 26、单页面开发

单页面应用（SPA）：只有一张Web页面的应用，是一种从服务器加载的富客户端，单页面跳转仅刷新局部资源，公共资源（js、css等）仅需加载一次。



多页面应用（MPA）：多页面跳转刷新所有资源，每个公共资源（js、css等）需选择性重新加载。



单页面开发的优缺点：

优点：

* 1. 用户体验好、快，内容的改变不需要重新加载整个页面，避免了不必要的跳转和重复渲染。
  2. 可以缓存较多数据，减少服务器压力

缺点：

1. 不利于搜索引擎优化，爬虫看不到完整的源码（因为可能在某些事件触发后，才会向服务器端请求资源，修改DOM元素）
2. 初次加载页面更耗时
3. 前进、后退、地址栏等，需要程序控制
4. 页面复杂度更高

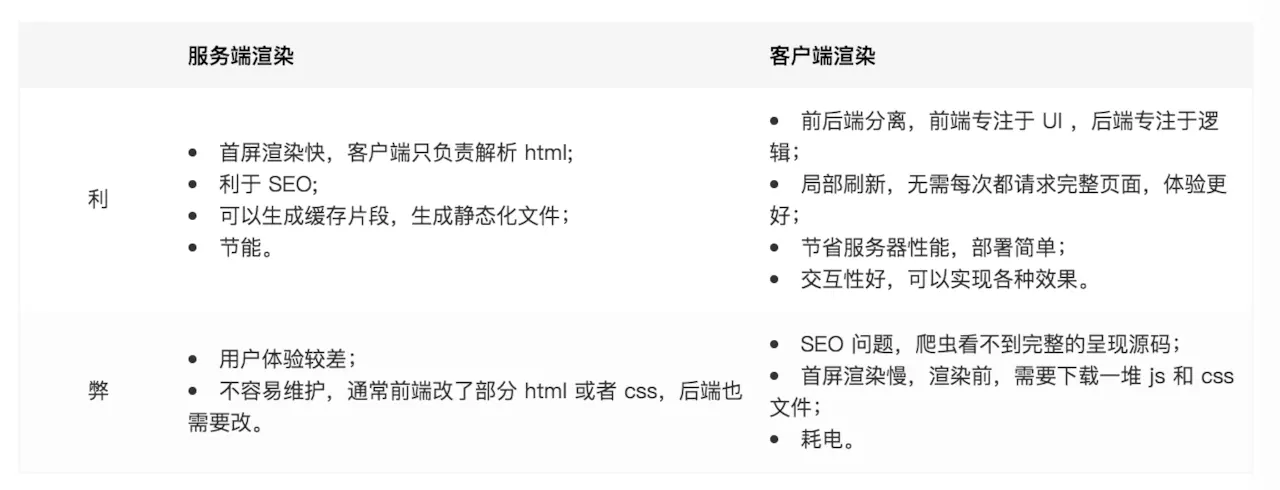
注：搜索引擎优化（SEO）：利用搜索引擎的搜索规则提高网站在搜索引擎的自然排名。



## 前端渲染vs后端渲染

**后端渲染**：后端的程序先把html页面上的特定区域，特定符号，用数据填充，再传给前端。

**前端渲染**：后端的html页面作为静态文件存在，前端请求时后端不对该文件做任何内容上的修改，直接以资源的方式返回给前端，前端拿到页面后，根据html页面上的js代码，对html的内容进行修改。



## 进程和线程的区别

**进程**

狭义：一段程序的。

广义：是具有独立功能的程序对某次数据集合的一次运行活动，是操作系统分配资源的基本单元。

1. 进程是一个实体。每个进程都有自己的地址空间，包括文本区域（存放程序代码）、数据区域（存储变量和进行执行期间使用的动态分配的内存）、堆栈（活动过程中调用的指令和本地变量）。
2. 进程是一个“执行中的程序”。程序是没有生命的实体，只有CPU赋予程序生命时，他才能称为一个活动实体，即进程。

三种状态：

1. 就绪：资源就绪，等待CPU
2. 运行：获取了CPU分配的资源
3. 阻塞：条件不够

通信：

1. 管道
2. 信号
3. 消息队列
4. 共享内存
5. 信号量
6. 套接字

**线程**

进程的一个实体，是被系统独立调度和分派的基本单位，线程不拥有系统资源，只拥有一点在运行中必不可少的资源(如程序计数器,一组寄存器和栈)，同属一个进程的所有线程共享进程的全部资源。一个进程包括一个或多个线程。

一个线程可以创建和撤销另一个线程，同一个进程中的多个线程可以并发执行，

同一进程的多个线程共享进程的全部系统资源，例如地址空间、文件描述符和信号处理等

但是多个线程有各自的调用栈、自己的寄存器环境、自己的线程本地存储

**区别**

1. 进程是系统分配资源的基本单位，线程是独立运行和独立调度的基本单位，比进程小
2. 进程拥有自己的地址空间，线程共享进程中的数据，使用相同的地址空间。CPU切换线程的花费小于进程
3. 线程之间通信更方便，通过共享全局变量、静态变量数据等；而进程之间需要以通信方式IPC（进程间通信）进行。
4. 多进程程序更健壮，一个进程死掉并不会影响其他进程；多线程只要有一个线程死掉，整个进程就死掉了。