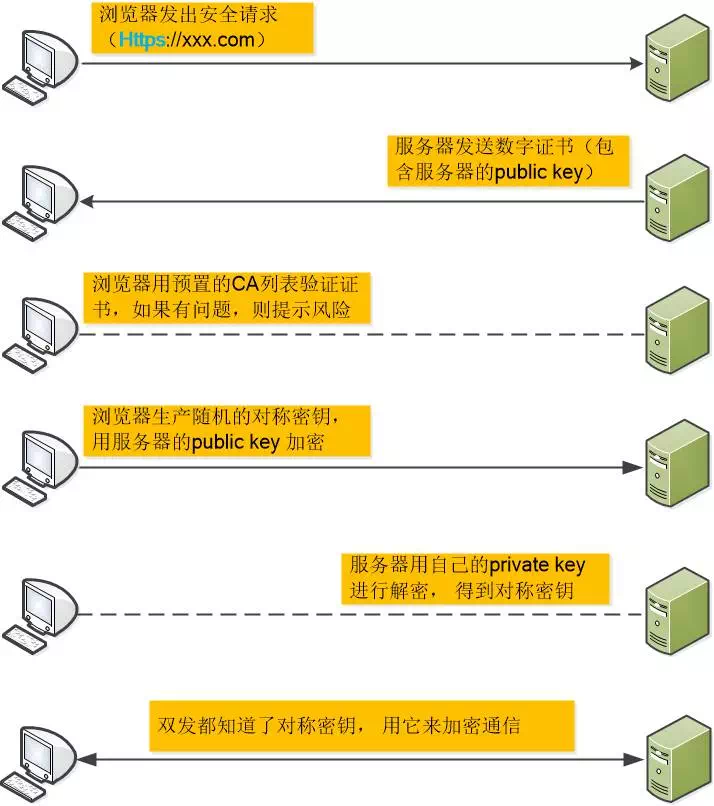
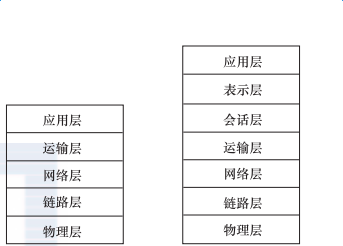
————————————————https————————————————————



https的ssl与tls的安全层协议位于应用层与传输层之间。

———————————————————五层协议————————————————



**网络攻击（CSRF、XSS）**

Csrf:中文名称：跨站请求伪造，也被称为：one click attack/session riding,

利用用户其他网站的cookie进行请求

防止：不用get用post，必要请求加token

Xss

恶意攻击者往Web页面里插入恶意Script代码，当用户浏览该页之时，嵌入其中Web里面的Script代码会被执行，从而达到恶意攻击用户的目的

防止：适当过滤

Ddos:泛洪攻击

http2.0/1.1/1.0

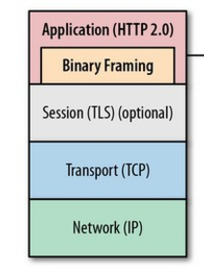
**2.0: 多路复用**允许同时通过单一的 HTTP/2 连接发起**多重**的请求-响应消息。 多流并行减少创建多个tcp链接。(提高单次传输的量（高带宽）降低延迟)

1.x: 在 HTTP/1.1 协议中 「浏览器客户端在同一时间，针对同一域名下的请求有一定数量限制。超过限制数目的请求会被阻塞」。(多个cdn的原因)

**2.0:消息头压缩**，减少ippackage，为了减少这块的开销并提升性能， HTTP/2会压缩这些首部： HTTP/2在客户端和服务器端使用“首部表”来跟踪和存储之前发送的键－值对，对于相同的数据，不再通过每次请求和响应发送； 首部表在HTTP/2的连接存续期内始终存在，由客户端和服务器共同渐进地更新; 每个新的首部键－值对要么被追加到当前表的末尾，要么替换表中之前的值。

**2.0：服务端推送**，将客户端需要的内容推送到客户端缓存，一次避免往返

***2.0：二进制分帧***，数据流以消息的形式发送，而**消息由一个或多个帧组成，帧可以在数据流上乱序发送，然后再根据每个帧首部的流标识符重新组装**。二进制分帧是HTTP/2的基石，其他优化都是在这一基础上来实现的。（原本每一个http请求都在一个tcp里，现在一个tcp多个http帧），http 与tcp之间加了分帧层，专门处理分帧，发送http请求的客户端不许考虑分帧。



（1.1/1.0在HTTP1.1中默认开启Connection： keep-alive，tcp复用）

server //nginx配置

{ listen 80; l

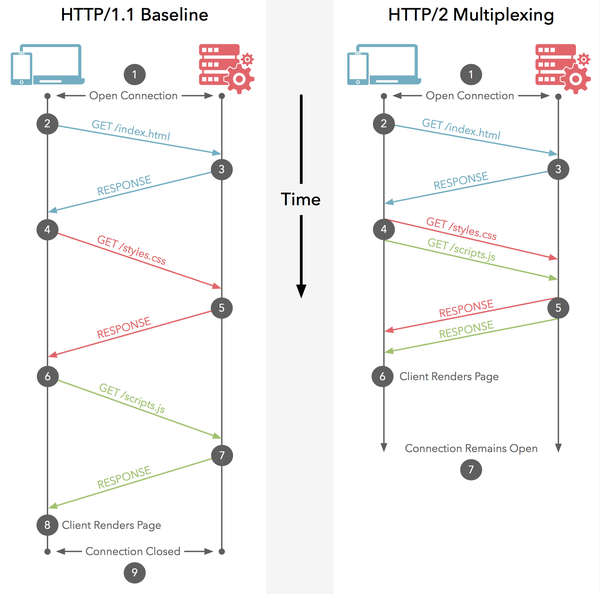
ocation / { return 301 https://$host$request\_uri; }

}

server {

listen 443 ssl http2;

}



**幂等性（规范）**

**多次请求返回相同结果，同一个请求，发送一次和发送N次效果是一样的。**

HTTP GET方法用于获取资源，不应有副作用，所以是幂等的。Get对数据类型和数据长度有限制。HTTP DELETE方法用于删除资源，有副作用，但它应该满足幂等性。HTTP POST方法用于创建资源，所对应的URI并非创建的资源本身，而是去执行创建动作的操作者，有副作用，不满足幂等性。HTTP PUT方法用于创建或更新操作，所对应的URI是要创建或更新的资源本身，有副作用，它应该满足幂等性。

有副作用的操作，需要有一个全局唯一标识（像作业帮内部id生成器），然后根据这个标识去check该操作是否已经被执行。比如支付页面就应该同步渲染id

get/post 明文/非明文

|  |  |  |  |
| --- | --- | --- | --- |
| 方法 | 幂等性 | 安全 |  |
| OPTIONS | yes | Yes//requesturlsupportmethod |  |
| GET | yes | Yes//获取 |  |
| HEAD | yes | Yes//请求头 |  |
| PUT | yes | No//和post类似，指定位置 |  |
| POST | no | No//提交数据（修改） |  |
| DELETE | yes | No//删除 |  |
| PATCH | no | no |  |

**安全方法（不会有副作用，安全的方法不会修改资源状态）会被缓存**

http

**requestheader**:

path:method:cookie:ua:referrer:path

**responseheader**:

content-type: date: etag: status:200 set-cookie

**端口：**

Http/https/:80/433

Vmstat

一个页面从输入URL到页面加载显示完成的详细过程

1. 浏览器根据url去查询dns缓存
2. 不存在则向本机操作系统请求解析url，查询到后返回浏览器
3. 浏览器三次握手与服务器建立链接
4. 浏览器在三次握手第三阶段发送http请求
5. 服务器接收到请求进行解析，返回响应报文。
6. 浏览器判断返回状态码，若状态码正常进入下一步
7. 浏览器接收到请求判断是否需要缓存并解析页面，如果有其他资源（css/js/图片/ajax）需要继续发送http请求，如果是http1.x，则串行的发起http请求。如果是2.0则并行发送请求
8. 浏览器接收到css，js去处理相应的操作，进行渲染执行相应的js

Cookie四部分：

1. 响应报文的cookie首部
2. 请求报文的cookie首部
3. 客户端存储的cookie

Session：

基于cookie实现，cookie存储一个sessionid，服务端根据sessionid解析出相应的session信息

若禁用cookie则url重写

状态码

2xx：成功

3xx：重定向

301 Moved Permanently新的永久性的URI 应当在响应的 Location 域中返回,

302 Move temporarily 与01类似，不自动缓存

304 Not Modified

4xx：请求错误

400 Bad Request 语义错误

401 Unauthorized

403 Forbidden

404 Not Found

5xx：服务器错误

500 Internal Server Error 代码错

501 Not Implemented 请求无法处理

502 Bad Gateway 作为网关或者代理工作的服务器尝试执行请求时

TCP与UDP

Udp：无连接、无链接状态、无拥塞控制

Tcp:三次握手建立链接维护链接状态，有拥塞控制、有慢启动，可靠数据传输（回退n帧，选择重传）

Ip：尽力而为的交付（转发/路由）

数据分片

Ipv4 2^32 ipv6 2^128

Netty

Linux内核select poll epoll

**Request-header**

**GET /xxx/xx**

Accept:

Accept-Encoding:

gzip, deflate

Accept-Language:

**Connection: keep-alive**

**Cookie:**

**Host:**

**Referer:**

**User-Agent**

**response**

**Cache-Contro**l:no-cache, must-revalidate

Content-Length:

Content-Type:

**Date**:Sun, 27 Aug 2017 17:43:31 GMT

**Expires**:

Server:cafe

Timing-Allow-Origin: