目录

[http 2](#_Toc4096950)

[1.1 2](#_Toc4096951)

[长连接 2](#_Toc4096952)

[断点续传 2](#_Toc4096953)

[请求方法 3](#_Toc4096954)

[host域 3](#_Toc4096955)

[2.0 3](#_Toc4096956)

[头信息压缩 3](#_Toc4096957)

[服务器推送 3](#_Toc4096958)

[多路复用 3](#_Toc4096959)

[二进制协议 3](#_Toc4096960)

[头信息 4](#_Toc4096961)

[https 4](#_Toc4096962)

[ssl/tls 4](#_Toc4096963)

[ca 5](#_Toc4096964)

[ca 5](#_Toc4096965)

[CA认证 5](#_Toc4096966)

[https 5](#_Toc4096967)

[功能 5](#_Toc4096968)

[认证过程 6](#_Toc4096969)

[1.客户端 6](#_Toc4096970)

[2.服务器 6](#_Toc4096971)

[3.客户端 6](#_Toc4096972)

[4.服务器 7](#_Toc4096973)

[5客户端 7](#_Toc4096974)

[其他 7](#_Toc4096975)

[tcp 7](#_Toc4096976)

[连接过程 7](#_Toc4096977)

[三次握手 8](#_Toc4096978)

[cookie 8](#_Toc4096979)

[session 8](#_Toc4096980)

# http

## 1.1

### 长连接

HTTP 1.0需要使用keep-alive参数来告知服务器端要建立一个长连接，而HTTP1.1默认支持长连接。

network.http.keep-alive（这种是http参数，不是header,即不能自己设置。） 默认是 true 是否同意持久连接。除非在响应中指定connection：close

network.http.keep-alive.timeout 默认是 300 持久连接同意的保持时间，这个调大了没意义，由于一般 server 设置的就是 300s。

network.http.max-connections-per-server 默认是 8 连接同一个server同意的最大连接数。

network.http.max-persistent-connections-per-server默认是 2 连接同一个server同意的最大持久连接数。

network.http.proxy.keep-alive 默认是 true 连接代理server是否同意持久连接

### 断点续传

增加了Range，content-length等头，通过Content-Range（重连后响应的范围）,Accept-Ranges（支持的断点续传格式，一般是bytes）,Range（重连后请求的文件范围）,Content-Length（响应的文件大小，重连后是剩余大小）这些实现断点续传。

### 请求方法

HTTP1.0定义了三种请求方法： GET, POST 和 HEAD方法。

HTTP1.1新增了五种请求方法：OPTIONS, PUT, DELETE, TRACE 和 CONNECT 方法。

### host域

增加host头，默认是请求的主机名（ip或者域名）。

http1.1中不能缺失host字段，但host字段可以是空值。

http响应头中不包含host字段

应用：比如一个ip多个域名，可以通过host判断用户访问的哪一个。

## 2.0

### 头信息压缩

一方面，头信息使用gzip或compress压缩后再发送；另一方面，客户端和服务器同时维护一张头信息表，所有字段都会存入这个表，生成一个索引号，以后就不发送同样字段了，只发送索引号，这样就提高速度了。

### 服务器推送

### 多路复用

### 二进制协议

HTTP/1.1 版的头信息肯定是文本（ASCII编码），数据体可以是文本，也可以是二进制。HTTP/2 则是一个彻底的二进制协议，头信息和数据体都是二进制，并且统称为"帧"（frame）：头信息帧和数据帧。

## 头信息

HTTP消息头支持自定义， 自定义的专用消息头一般会添加'X-'前缀。

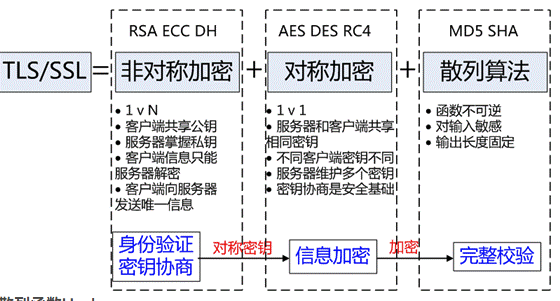
# https

咱们通常所说的 HTTPS 协议，说白了就是“HTTP 协议”和“SSL/TLS 协议”的组合。

## ssl/tls

SSL 是“Secure Sockets Layer”的缩写，中文叫做“安全套接层”。

TLS是“Transport Layer Security”的缩写，中文叫做“传输层安全协议”。



## ca

### ca

电子商务认证授权机构（CA, Certificate Authority），也称为电子商务认证中心，是负责发放和管理数字证书的权威机构，并作为电子商务交易中受信任的第三方，承担公钥体系中公钥的合法性检验的责任。

CA是证书的签发机构,它是PKI的核心。CA是负责签发证书、认证证书、管理已颁发证书的机关。它要制定政策和具体步骤来验证、识别用户身份，并对用户证书进行签名，以确保证书持有者的身份和公钥的拥有权。。

CA 也拥有一个证书（内含公钥）和私钥。网上的公众用户通过验证 CA 的签字从而信任 CA ，任何人都可以得到 CA 的证书（含公钥），用以验证它所签发的证书。

### CA认证

如果用户想得到一份属于自己的证书，他应先向 CA 提出申请。在 CA 判明申请者的身份后，便为他分配一个公钥，并且 CA 将该公钥与申请者的身份信息绑在一起，并为之签字后，便形成证书发给申请者。

如果一个用户想鉴别另一个证书的真伪，他就用 CA 的公钥对那个证书上的签字进行验证，一旦验证通过，该证书就被认为是有效的。

证书里包含了域名信息，过期时间

## https

### 功能

认证：伪装：无法验证身份，可能被伪装

加密：窃听：通信使用明文，内容可能被窃听

完整性保护：篡改：无法保证报文的完整性，内容可能被篡改

## 认证过程

一次完整的https请求大概是这样的。

### 1.客户端

向服务器发出请求。（携带：客户端支持的协议版本，加密算法，随机数（client random），压缩算法）

### 2.服务器

发送自己信息到客户端，包括服务器端的公钥（.car证书文件），协议版本，加密算法，压缩方法，随机数2(server random)。

### 3.客户端

#### 验证证书

根据服务器发回来的公钥去自己钥匙串里面匹配本地证书(公钥)，来判断证书是否过期，可用等。如果本地证书没有的话。就会弹出警告改证书不被信任，询问用户是否要安装这个证书。一旦安装，下次访问的时候再去自己的钥匙串里面去找的时候就能找到了。（所以如果这个证书没有内置，就可能在第一次建立连接时被中间人攻击）

匹配方法：先看CA的根证书是否在自己信任根证书列表中，在的话用CA的根证书提供的公钥来验证服务器证书中的数字签名，如果公钥可以解开签名，证明该证书确实被CA信任，再看看证书是否过期，访问的网站域名与证书绑定的域名是否一致，这些通过，说明证书是安全的。（解出的信息中的域名与访问域名不一致会发出警告：告诉用户此网站的安全证书有问题，可以防止中间人攻击）。

#### 随机数pre-master secret

生成一个随机数pre-master secret，然后使用证书中的公钥进行加密，然后传递给服务器端（这个随机值在TCP协议里面会自动帮我加。OSX内核部分，我们都无法接触到。这个值我们改不了）。

### 4.服务器

根据radom1、radom2、pre-master secret通过一定的算法得出session Key和MAC算法秘钥，作为后面交互过程中使用对称秘钥。（对称秘钥）。发给客户端。

### 5客户端

客户端也会使用radom1、radom2、pre-master secret，和同样的算法生成对称秘钥。和服务器发回来的秘钥匹配，一致则表示验证成功。像服务器发送正常的http请求。

## 其他

charles使用代理服务器的方式拦截https请求，它有自己的证书。

# tcp

## 连接过程

序号：seq序号，占32位，用来标识从TCP源端向目的端发送的字节流，发起方发送数据时对此进行标记。

确认序号：ack序号，占32位，只有ACK标志位为1时，确认序号字段才有效，ack=seq+1。

标志位：共6个，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)。

这些都不是存在报文里的，而是在tcp段的固定字节。

### 三次握手

1 客户端

发送SYN标志位，seq序号为随机值，没有ack序号

2 服务端

通过SYN标志位知道是建立连接，发送SYN，ACK标志位，seq序号为随机值，ack序号值为请求中的seq序号的值+1。

3客户端正式通信

发送ACK标志位，seq序号为响应中的ack序号，ack序号值为响应中的seq序号的值+1。同时发送正常的数据。

后面的都是这样， seq序号为发过来的ack，ack序号·是跟体积相关的随机数。

#### 四次挥手

# cookie

响应头里可以设置Set-Cookie(里面也是写的cookie),浏览器获取cookie后，会保存在本地，然后放在以后的请求的cookie中。

# session

保存在服务器端，但不是保存在内存中，而是保存在文件或数据库中