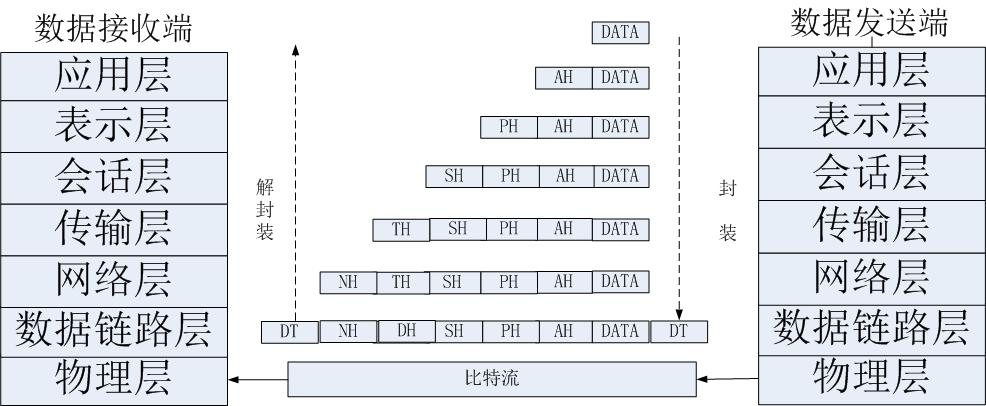
本篇文章主要讲解关于http协议的一些基础知识。主要内容如下：

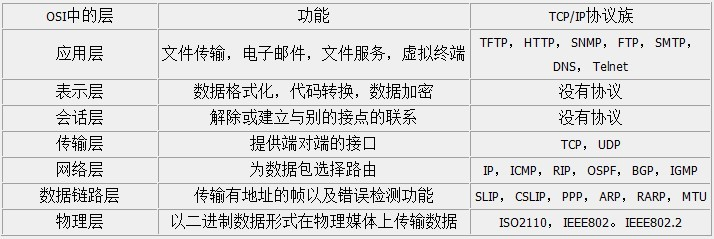
* 1、TCP
* 2、TCP的3次握手和4次挥手
* 3、HTTPS
* 4、SPDY
* 5、HTTP2.0
* 6、隧道
* 7、代理
* 8、InetAddress和InetSocketAddress

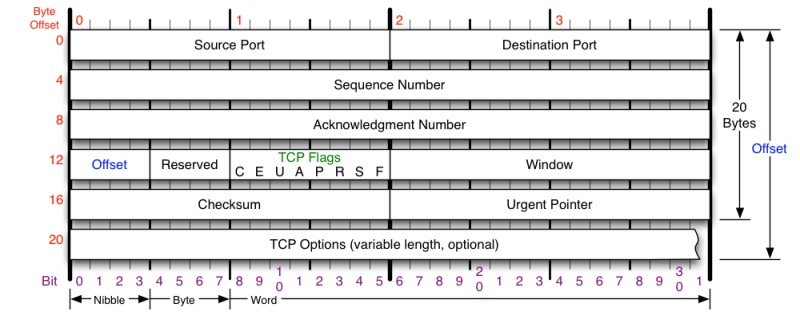
**一、TCP**

关于TCP的具体，我这里就不细说，如这是我们每个程序员的基本知识，我这里就简单说下，看下OSI的七层模型：



我们知道TCP工作在第四层Transport层(传输层)，IP在第三层Network层(网络层)，ARP在第二层Data Link层(数据链路层)；在第二层上的数据，我们把她叫做Frame，在第三层上的数据叫Packet，第四层的数据叫Segment。并且，我们需要知道，数据从应用层发下来，会在每一层都加上头部信息，进行封装，然后再发送到数据接收端。这个基本的流程你需要知道，就是每个数据都会经过数据的封装和解封装的过程。在OSI七层模型中，每一层的作用对应的协议如下：

TCP是一个协议，那这个协议是如何定义的，它的数据格式是什么样子的？那就要进行更深层次的剖析，就需要了解，甚至是熟记TCP协议中的每个字段的含义，如下图：

上面就是TCP协议头部的格式，由于它太重要了，所以下面就将每个字段的信息都详细说明下

* Source Port和Destination Port:分别占用16位，表示源端口号和目的端口号；用于区别主机中的不同进程，IP地址用来区分不同主机的，源端口号和目的端口号配合上IP首部中的源IP地址就能确定为一个TCP连接
* Sequenece Number:用来标识从TCP发送端向TCP接收端的数据字节流，他表示在这个报文中的第一个数据字节流在数据流中的序号；主要用来解决网络乱序的问题。
* Acknowledgment Number：32位确认序号包发送确认的一端所期望收到的下一个序号，因此，确认需要应该是上次已成功收到数据字节序号+1，不过只有当标志位中的ACK标志(下面介绍)为1时该确认序列号的字段才有效。主要用来解决不丢包的问题。
* Offset:给出首部中32bit字的数目，需要这个值是因为任选字段的长度是可变的。这个字段占4bit(最多能表示15个32bit的字，即4\*15=60个字节的首部长度)，因此TCP最多有60个字节的首部。然而，没有任选字段，正常的长度是20字节。
* TCP Flags:TCP首部中有6个比特，它们总的多个可同时被设置为1，主要是用于操控TCP的状态机，依次为URG，ACK，PSH，RST，SYN，FIN。
* Window:窗口大小，也就是有名的滑动窗口，用来进行流量控制；这是一个复杂的问题

TCP是主机对主机层的传输控制协议，提供可靠的连接服务，采用三次握手确认建立一个连接:位码即tcp标志位,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)Sequence number(顺序号码) Acknowledge number(确认号码)  
现在来介绍一下这些的标志位：

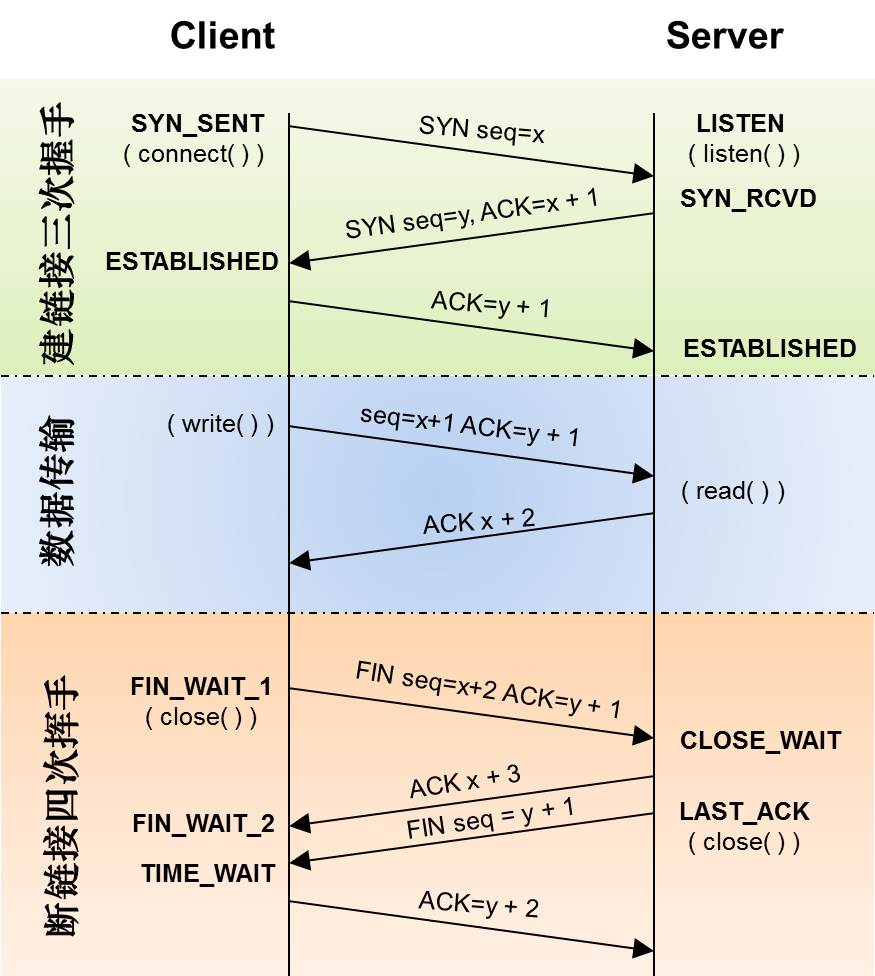
* URG:次标志表示TCP包的紧急指针域(后面马上就要说到)有效，用来保证TCP连接不被中断，并督促中间层设备要尽快处理这些数据。
* ACK:此标志表示应答域有效，就是说前面所说的TCP的应答将会包含在TCP数据包中；有两个取值：0和1，为1的时候表示应答域有效，反之为0。
* PSH:这个标志位表示Push操作。所谓Push操作就是是在数据包到达接受端以后，立即传送给应用程序，而不是在缓冲区排队。
* RST:这个标志位表示连接复位请求。用来复位哪些产生错误的链接，也被用来拒绝错误和非法的数据包。
* SYN:表示同步序号，用来建立连接。SYN标志位和ACK标志位搭配使用，当连接请求的时候，SYN=1，ACK=0；连接被响应的时候，SYN=1，ACK=1。这个标志的数据包常常被用来进行端口扫描。扫描者发送一个只有SYN的数据包，如果对方主机响应了一个数据包回来，就表明这台主机存在这个端口，但是由于这种扫描只是进行TCP三次握手的第一次握手，因此这种扫描的成功表明被扫描的机器很不安全，一台安全的主机将会强制要求一个连接严格的进行TCP的三次握手。
* FIN:表示发送端已经达到数据末尾，也就是说双方数据传送完成，没有数据可以传送了，发送FIN标志位的TCP数据包后，连接将断开。这个表示的数据包也经常被用于进行端口扫描。

讲完标志位后就要开始正式的连接。

**二、TCP3次握手和4次挥手**

**(一)、3次握手**

TCP是面向连接的，无论哪一方向另一方发送数据之前，都必须现在双方之间建立一条连接。在TCP/IP协议中，TCP协议提供可靠的连接服务，连接是通过三次握手进行初始化的。三次握手的目的是同步连接双方的序列号和确认号并交换TCP窗口大小的信息。这就是面试中经常被问到的TCP三次握手。那咱们就详细的了解下，看图述说：

多么清晰的一张图啊，可惜不是我的，我"借"来的

* 1、第一次握手:建立连接，客户端先发送连接请求报文，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；
* 2、第二次握手：服务器收到SYN报文。服务器收到客户端的SYN报文，需要对这个SYN报文进行确认，设置Acknowledgment Number为x+1(Sequence+1)；同时，自己还要送法SYN消息，将SYN位置为1，Sequence Number为y；服务器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。
* 3、第三次握手；客户端收到服务器的 SYN+ACK报文段。然后将Acknowlegment Number设为y+1,向服务器发送ACK报文段，这个报文段发送完毕后，客户端端服务器都进入ESTABLISHED状态，完成TCP三次握手。

实例:

IP 192.168.1.116.3337 > 192.168.1.123.7788: S 3626544836  
第一次握手：192.168.1.116发送位码syn＝1,随机产生seq number=3626544836的数据包到192.168.1.123,由SYN=1知道192.168.1.116要求建立联机;

IP 192.168.1.123.7788 > 192.168.1.116.3337: S 1739326486: ack 3626544837  
第二次握手：192.168.1.123收到请求后要确认联机信息，向192.168.1.116发送ack number=3626544837,syn=1,ack=1,随机产生seq=1739326486的包;

IP 192.168.1.116.3337 > 192.168.1.123.7788: ack 1739326487,ack 1  
第三次握手：192.168.1.116收到后检查ack number是否正确，即第一次发送的seq number+1,以及位码ack是否为1，若正确，192.168.1.116会再发送ack number=1739326487,ack=1，192.168.1.123收到后确认seq=seq+1,ack=1则连接建立成功。

完成了三次握手，客户端和服务器就可以开始传送数据了，以上就是TCP三次握手的总体介绍。

**(二)、4次回挥手**

当客户端和服务器进行三次握手简历TCP连接以后，当数据传送完毕，肯定要断开TCP连接。那对于TCP的断开，就是"4次挥手"。

* 1、客户端(也可以是服务器)，设置Sequence Number和Acknowledgment Number，向服务器发送一个FIN报文段。此时客户端进入FIN\_WAIT\_1状态；这表示客户端没有数据发送给主机了。
* 2、服务器收到客户端发来的FIN报文段，向客户端回一个ACK报文段，Acknowledgement Number为Sequence Number加1；客户端进入FIN\_WAIT\_2状态，服务器进入CLOSE\_WAIT状态；服务器告诉客户端，我同意你的"关闭"请求。
* 3、服务器向客户端发送FIN报文段，请求关闭连接，同时服务器进入LAST\_ACK状态。
* 4、客户端收到服务器发送的FIN报文段，向主机发送ACK报文段，然后客户端进入TIME\_WAIT状态，服务器收到客户端的ACK报文段以后，就关闭连接，此时，客户端等待2MSL后一次没有到收到回复，则证明Server端已正常关闭，那好，客户端也可以关闭连接了。

至此，TCP的四次分手就完成了。

为了让大家更好的理解3次握手和4次挥手，我准备了几道问题，让大家更好的理解3次握手和4次挥手。

**1、为什么握手要“3”次？**

一般同学会认为，握手为什么要3次，感觉2次就可以了。在谢希仁的<计算机网络>中是这样说的

为了防止已失效的连接请求报文段突然又传送到了服务端，因而产生错误。

在书中同时举了一个例子，如下:

“已失效的连接请求报文段”的产生在这样一种情况下：client发出的第一个连接请求报文段并没有丢失，而是在某个网络结点长时间的滞留了，以致延误到连接释放以后的某个时间才到达server。本来这是一个早已失效的报文段。但server收到此失效的连接请求报文段后，就误认为是client再次发出的一个新的连接请求。于是就向client发出确认报文段，同意建立连接。假设不采用“三次握手”，那么只要server发出确认，新的连接就建立了。由于现在client并没有发出建立连接的请求，因此不会理睬server的确认，也不会向server发送数据。但server却以为新的运输连接已经建立，并一直等待client发来数据。这样，server的很多资源就白白浪费掉了。采用“三次握手”的办法可以防止上述现象发生。例如刚才那种情况，client不会向server的确认发出确认。server由于收不到确认，就知道client并没有要求建立连接。”

这就很明白了，防止了服务器端的一直等待而浪费资源

**2、为什么握手要“3”次，而挥手要"4"次?**

TCP协议是一种面向连接的、可靠的、基于字节流的运输层通信协议。TCP是全双工模式，这就意味着，当客户端发起FIN报文段时，只表示客户端已经已经没有数据要发送了，客户端告诉服务器，它的数据已经全部发送完毕了，但是此时客户端还是可以接受服务器的数据；当服务器返回ACK报文段是时，表示它已经知道客户端没有数据发送了，但是服务器还是可以发送数据到客户端；当服务器也发送了FIN报文时，这时候就是表示服务器也没有数据要发送给客户端了，之后就可以愉快地中断这次TCP连接。

为了让大家更好的地理解四次挥手的原理，我们再讲解下四次挥手的状态：

* 1、FIN\_WAIT\_1:这个状态要好好解释一下，其实FIN\_WAIT\_1和FIN\_WAIT\_2状态的真正含义都是表示等待对方的FIN报文。而这两种状态的区别：FIN\_WAIT\_1状态实际上是当SOCKET在ESTABLISHED状态时，它想主动关闭连接，向对方发送了FIN报文，此时该SOCKET即进入FIN\_WAIT\_1状态。而当对方回应ACK报文后，则进入FIN\_WAIT\_2状态，当然在实际的正常情况下，无论对方何种情况下，都应该马上回应ACK报文，所以FIN\_WAIT\_1状态一般是比较难见到的，而FIN\_WAIT\_2状态还有时常常可看到。
* 2、FIN\_WAIT\_2:上面已经详细解释了这种状态，实际上 FIN\_WAIT\_2状态下的SOCKET，表示半连接，既有一方要求close连接，但另外还告诉对方，我暂时还有点数据需要传送给你(ACK信息)，稍后再关闭连接
* 3、CLOSE\_WAIT\_2:这种状态的含义其实表示在等待关闭。怎么理解？当对方close一个SOCKET后发送一个FIN报文给自己，服务器系统毫无疑问会回应一个ACK报文给对方，此时则进入到CLOSE\_WAIT状态。接下来呢，实际上你真正需要考虑的事情是查看你是否还有数据发送给对方，如果没有的话，那么你也就可以close这个这个SOCKET，发送FIN报文给对方，也即关闭连接。所以你在CLOSE\_WAIT状态下，需要完成的事情是等待你去关闭连接。
* 4、LAST\_ACK:这个状态还是比较容易好理解的，它是被动关闭一方在发送FIN报文后，最后等待对方的ACK报文。当收到ACK报文后，也可以进入CLOSED状态了。
* 5、CLOSED:表示连接中断
* 6、TIME\_WAIT:表示收到对方的FIN报文，并发出了ACK报文就等2MSL后即可会到CLOSED可用状态了。如果FIN\_WAIT\_1下，收到了对方同时带FIN标志和ACK标志的报文时，可以直接进入到TIME\_WAIT状态，而无须经过FIN\_WAIT\_2状态。

**3、为什么TIME\_WAIT状态需要经过2MSL(最大报文段生存时间)才能返回到CLOSE状态？**

虽然按道理，四个报文都发送完毕，我们可以直接进入CLOSE状态了，但是我们必须假设网络是不可靠的，一切都可能发生，比如有可能最后一个ACK丢失。所以TIME\_WAIT状态是用来重发可能丢失的ACK报文。

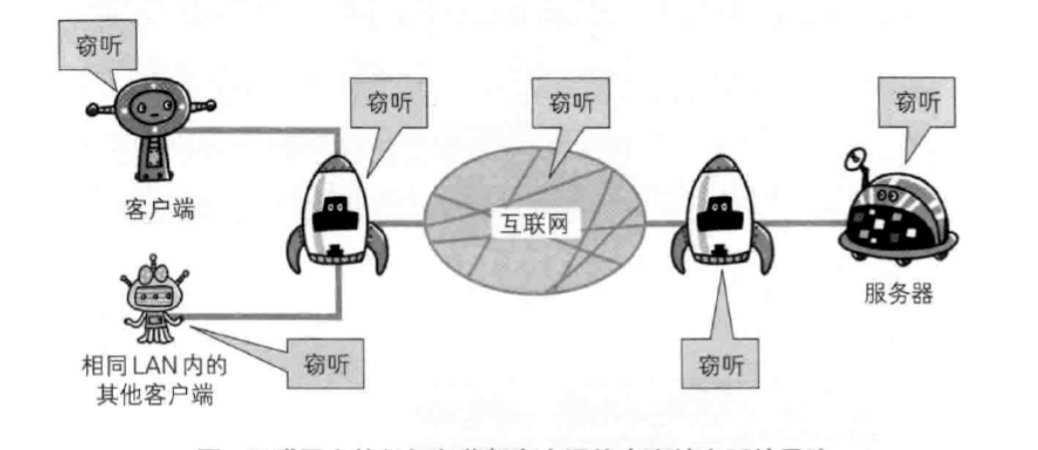
**三、HTTPS**

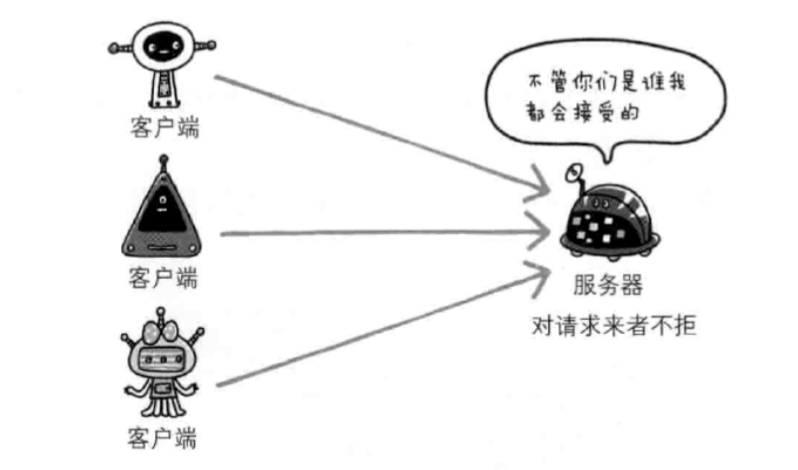
**(一)、什么是HTTPS?**

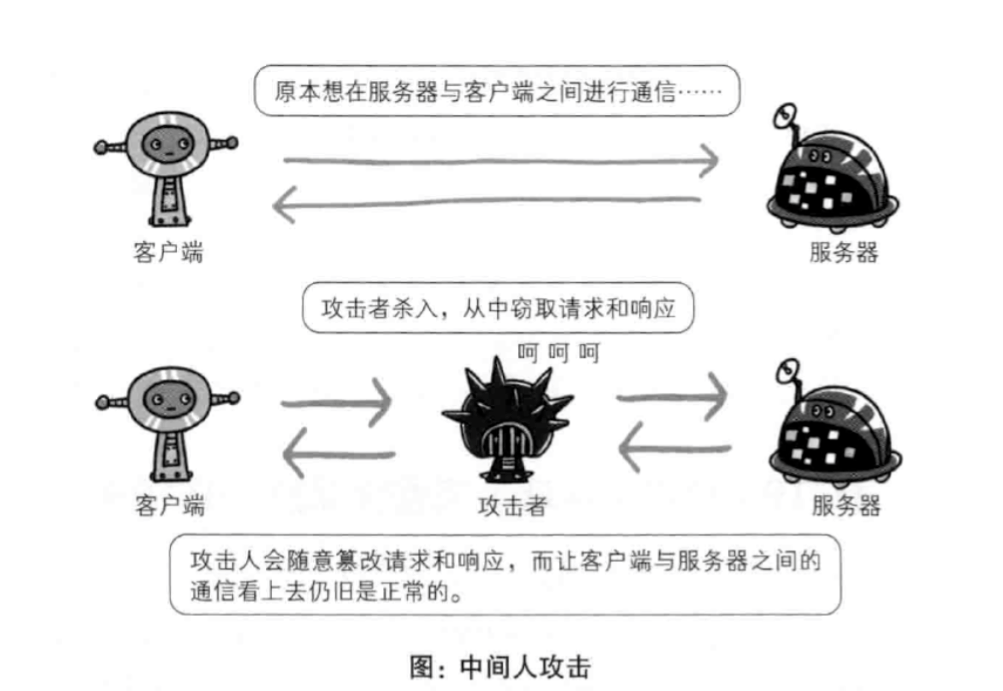
HTTPS(Hypertext Transfer Protocol over Secure Socket Layer/基于安全套接字层的超文本传输协议,或者也可以说HTTP OVER SSL)是网景公司开发的web协议。SSL的版本最高为3.0，后来的版本被称为TLS，现在所用的协议版本一般都是TLS，但是由于SSL出现的时间比较早，所以现在一般指的SSL一般也就是TLS，本文中后面都统一使用SSL来代替TLS。HTTPS是以安全为目标的HTT通道，简单讲就是HTTP的安全版，所以你可以理解HTTPS=HTTP+SSL。即HTTP下加入SSL层，HTTPS的安全基础是SSL，因此加密的详细内容是SSL。

**(二)、为什么要用HTTPS?**

在没有HTTPS之前使用HTTP的协议的，可见HTTPS肯定是弥补的HTTP的安全缺陷，那么咱们来看下HTTP协议的那些安全缺点：  
1、通信使用明文(不加密)，内容可能被窃听(抓包工具可以获取请求和响应内容)如下图：

2、不验证通讯方的身分，任何人都坑你发送请求，不管对方是谁都返回相应，如下图：

3、无法证明报文的完整性，可能会遭到篡改，即没有办法确认发出的请求/相应前后一致。

所以为了解决上述问题，需要在HTTP上加入加密处理和认证机制，我们把添加了加密和认证机制的http称之为http。即HTTP+加密+认证+完成性保护=HTTPS。  
这里先简单的说下HTTPS说下他的优势：

* 1、内容加密，建立一个信息的安全通道，来保证数据传输过程的安全性。
* 2、身份认证，确认网站的真是性。
* 3、数据完整性，防止内容被第三方冒充或者篡改。

是不是 刚好弥补了上面的缺陷。。。。  
补充一下:HTTPS并非应用层的一种新协议，只是HTTP通讯接口部分用SSL(secure socket layer)和TLS(transport layer security)协议代替，通常HTTP直接和TCP通信时，当使用SSL时，则演变成先和SSL通信，再由SSL和TCP通讯，因此所以HTTPS其实就是身披SSL保护外衣的HTTP

**(三)、HTTPS的缺点**

上帝给你关上一道门 同时给你打开一扇窗，所以万事万物都是各有利弊，那么HTTPS的缺点是什么那?  
由于要对数据进行加密，认证，所以注定他会比HTTP慢，当然现在也有很多优化。  
对数据进行加解密决定了它比http慢。  
PS:处于安全考虑，浏览器是不会再本地保存HTTPS缓存。实际上，只要在HTTP头中使用特定的命令，HTTPS是可以被缓存的。Firefox默认只在内存中缓存HTTS。但是，只要在请求头中有Cache-Control:Public，缓存就会被写到磁盘上，IE只要http头允许就可以缓存https内容，缓存策略与是否使用HTTPS协议无关。

**(四)、HTTP与HTTPS的相同和异同点**

1、HTTP与HTTPS的相同点：  
大多数情况下，HTTP和HTTPS是相同的，因此都采用同一个基础协议，作为HTTP或者HTTPS客户端--浏览器/app等，设置一个连接到web服务器指定的宽口。当服务器接收到请求，它会返回一个状态码以及消息，这个回应可能是请求信息、或者指示某个错误发送的错误信息。系统使用统一资源定位符URI，因此资源可以被唯一指定。在表面上HTTPS和HTTP唯一不同的只是一个协议头HTTPS的说明，其他都一样。

2、HTTP与HTTPS的不同点：

* 1、HTTPS需要用到CA申请证书。
* 2、HTTP是超文本传输协议，信息是明文的；HTTPS则是具有安全性的SSL加密传输协议。
* 3、HTTPS和HTTP使用的是完全不同的连接方式，用的端口也不一样，HTTP是80,HTTPS是443。
* 4、HTTP的连接很简单，是无状态的，HTTPS是HTTP+SSL协议构建的，可进行加密传输、身份认证的网络协议，比HTTP协议安全。

**(五)、简单说点加密和证书的事**

再详细说HTTPS之前先来了解下加密和证书的事情

**1、加密**

加密的2种技术：

* 1、对称加密(也叫私钥加密)，是指加密和解密使用相同的密钥的加密算法。有时又叫传统加密算法，就是加密密钥能够从解密密钥中推算出来，同时解密密钥也可以从加密密钥中推算出来。而在大多数的对称算法中，加密密钥和解密密钥是相同的，所以也称这种加密算法为秘密密钥或者单密钥算法，常见的对称加密有DES(Data Encryption Standard)、AES(Advanced Encryption Standard)、RC4、IDEA。
* 2、非对称加密，是指和甲酸算法不同，非对称加密算法有两个密钥：公开密钥(public key）和私有密钥(private key)；并且加密密钥和解密密钥是成对出现的。非对称加密算法在加密和解密过程使用了不同的密钥，非对称加密也称公钥加密，在密钥对中，其中一个密钥是对外公开的，所有都可以获取到，称为公钥，其中一个密钥是不公开的称为私钥。非对称加密算法对加密内容的长度有限制，不能超过公钥长度。常见的非对称加密RSA、DSA/DSS等。

**2、数字摘要**

数字摘要是采用单项Hash函数将需要加密的明文"摘要"成一串固定长度(128位)的密文，这一串密文又称为数字指纹，它有固定的长度，而且不同的明文摘要成密文，其结果总是不同的，而同样的明文摘要必定一定。“数字摘要”是https能确保数据完整性和防篡改的根本原因。常用的摘要主要有MD5、SHA1、SHA256等。

**3、数字签名**

数字签名技术就是对"非对称密钥加解密"和"数字摘要"两项技术的应用，它将摘要信息用发送者的私钥加密，与原文一起传送给接受者。接受者只有用发送者的公钥才能解密被加密的摘要信息，然后用HASH函数对收到的原因产生一个摘要信息，与解密的摘要信息对比。如果相同，则说明收到的信息是完整的，在传输的过程中没有被修改，否则说明信息被修改过，因此数字签名能够验证信息的完整性。  
数字签名的过程：

明文——>hash运算——>摘要——>私钥加密——>数字签名

数字签名的两种功效：

* 1、能确定消息确实由发送方签名并发出来的，因为别人假冒不了发送方的签名。
* 2、数字签名能确定消息的完整性。

注意:  
数字签名只能验证数据的完整性，数据本身是否加密不属于数字签名的控制范围

**4、数字证书**

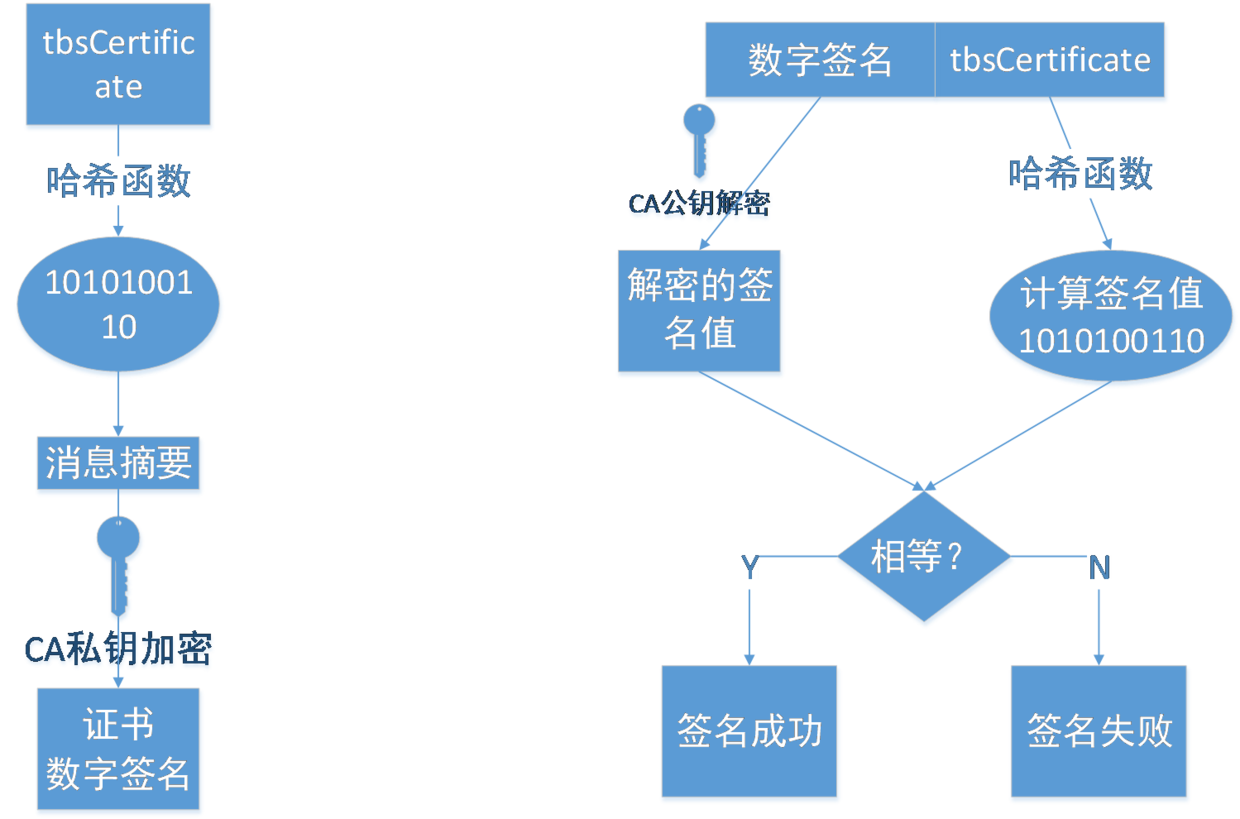
对于请求方来说，它怎么能确定它所得到的公钥一定是从目标主机哪里发送的，  
而且没有被篡改过呢?亦或者请求的目标主机本身就是从事窃取用户信息的不正当行为呢？这时候，我们需要一个权威的值得信赖的第三方机构(一般是由政府机构审核并授权的机构)来统一对外发送主机机构的公钥，只要请求方这种机构获取公钥，就避免了上述问题

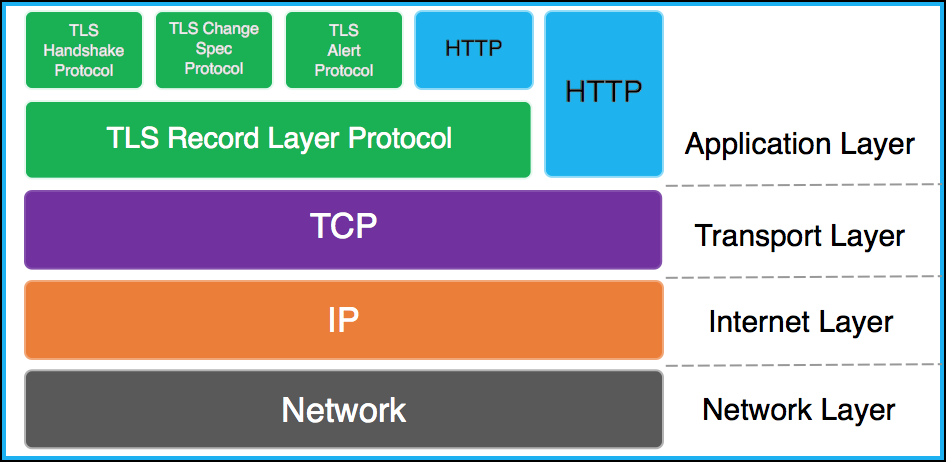
(1)数字证书的颁发过程:  
用户首先产生自己的密钥对，并将公共密钥及部分个人身份信息传送给认证中心。认证中心在核实身份后，将执行一些必要的步骤，以确信请求确实由用户发送而来，然后，认证中心将发送给用户一个数字证书，该证书内包含用户的人信息和他的公钥信息，同时还附有认证中心的签名信息(根证书私钥)签名。用户就可以使用自己的数字证书进行相关的各种活动。数字证书由独立的证书发行机构发布，数字证书各不相同，每种证书可提供不同级别的可信度。  
(2)证书包含哪些内容

* 1、证书颁发机构的名称
* 2、证书本身的数字签名
* 3、证书持有者的公钥
* 4、证书签名用到的Hash算法

(3)验证证书的有效性  
浏览器默认都是会内置CA跟证书，其中根证书包含了CA的公钥

* 防伪造证书1:如果证书颁发机构是伪造的，浏览器不认识，直接认为是危险证书
* 防伪造证书2:证书颁发机构是的确存在的，于是根据CA名，找到对应内置的CA根证书、CA的公钥。用CA的公钥，对伪造的证书的摘要进行解密，发现解密不了，认为是危险证书
* 防篡改:对于篡改的证书，使用CA的公钥对数字签名进行解密得到摘要A，然后再根据签名的Hash算法计算出证书的摘要B，对比A与B，若相等则正常，若不相等则是被篡改过的。
* 防过期失效验证:正式课在其过期前辈吊销，通常情况是该证书的私钥已经失密。较新的浏览器如果Chrome、Firefox、Opera和Internet Explored 都实现了在线证书的状态协议(OCSP)以排除这种情况:浏览器将网站提供的证书序列号通过OCSP发送给证书颁发机构，后者会告诉浏览器证书是否还是有效的。

 **(六)那么咱们来看下HTTPS的结构图**

  
由上图可以看到，HTTP是直接建立在TCP之上的，而HTTPS则又经过了TLS一系列协议。所以，https协议在建立连接之前，首先要和双方握手，身份认证，传输数据之前要进行加密

**(七)SSL和TLS**

**1、SSL(Secure Sokcet Layer，安全套接字层)**

SSL是Netscape(网景公司)所研发，用于保障在Internet上数据传输至安全，利用数据加密(Entryption)技术，可确保数据在网络值传输过程中不会被截取，当前版本为3.0。

SSL协议可分为两层:

* 1、SSL记录协议(SSL Record Protocol):它建立在可靠的传输协议(如TCP)之上，为高层协议提供数据封装、压缩、加密等基本功能的支持。
* 2、SSL握手协议(SSL Handshake Protocol):它建立在SSL记录协议之上，用于在实际的数据传输开始前，通讯双方进行身份认证、协商加密算法、交换加密密钥等。

**2、TLS(Transport Layer Security,传输层安全协议)**

用于两个应用程序之间提供保密性和数据完整性。  
TLS1.0是IETF(Internet Engineering Task Force,Internet工程任务组)制定的一种新的协议，它建立在SSL3.0协议规范之上，是SSL3.0的后续版本，可以理解为SSL3.1，它是写入RFC的，该协议由两层组成：TLS记录协议(TLS Record)和TLS握手协议(TLS Handshake)。较低的层为TLS记录协议，位于某个可靠的传输协议(例如TCP)上面

TLS 1.0是IETF（Internet Engineering Task Force，Internet工程任务组）制定的一种新的协议，它建立在SSL 3.0协议规范之上，是SSL 3.0的后续版本，可以理解为SSL 3.1，它是写入了 RFC 的。该协议由两层组成： TLS 记录协议（TLS Record）和 TLS 握手协议（TLS Handshake）。较低的层为 TLS 记录协议，位于某个可靠的传输协议（例如 TCP）上面。

**3、SSL/TLS协议作用：**

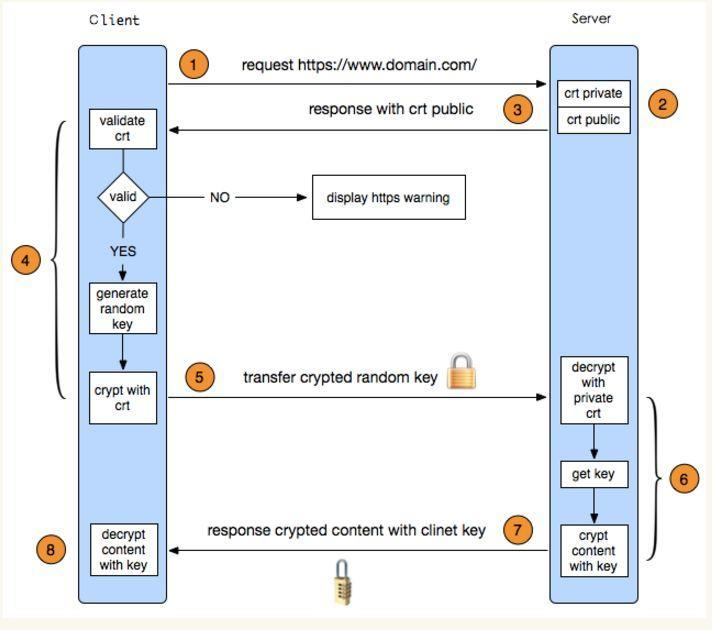
* 认证用户和服务器，确保数据发送到正确的客户端和服务器
* 加密数据以防止数据中途被窃取
* 维护数据的完整性，确保数据在传输过程中不被改变。

**4、TLS比SSL的优势**

* 1、对于消息认证使用密钥散列发：TLS使用"消息认证代码的密钥散列法"(HMAC),当记录在开放的网络(如因特网)上传时，该代码确保记录不会被变更。SSLv3.0还提供键控消息认证，但HMAC比SSLv3.0使用的(消息认证代码)MAC功能更安全
* 2、增强的伪随机功能(PRF):PRF生成密钥数据。在TLS中，HMAC定义PRF。PRF使用两种散列算法保证其安全性，如果任一算法暴露了，只要第二种算法未暴露，则数据仍然是安全的。
* 3、改进的已完成消息验证：TLS和SSLv3.0都对两个端点提供已完成的消息，该消息认证交换的消息没有被变更。然而，TLS将此已完成消息基于PRF和
* 4、一致证书处理：与SSLv3.0不同，TLS试图制定必须在TLS之间实现交互的证书类型。
* 5、特定警报消息：TLS提供更多的特定和附加警报，以指示任一会话端点检测到的问题。TLS还对核实应该发送某些警报进行记录。

**(八)HTTPS握手的过程**

HTTP握手是有“3次握手的”，HTTPS是有"8次握手的"，大体分为以下几个步骤：  
1 验证证书的有效性(是否被更改，是否合法)  
2 握手生成会话密钥  
3 利用会话密钥进行内容传输  
先上图

  
我去，都是英文的啊，不过自己看了以下，貌似还能看得懂，咱们就先详细说下：

**1、客户端首次发出请求**

由于客户端(如浏览器)对一些加解密算的支持程度不一样，但是在TLS协议传输过程中必须使用同一套加解密算法才能保证数据能够正常的加解密。在TLS握手阶段，客户端要首先告知服务器，自己支持哪些加密算法，所以客户端需要将本地支持的加密套件(Cipher Suit)的列表传送给服务器。除此之外，客户端还要产生一个随机数，这个随机数一方面需要在客户端保存，另一方面需要传给服务器，客户端的随机数需要跟服务器的随机数结合起来产生后面将要的Master Secret.  
客户端需要提供如下信息:

* 1、支持的协议版本，比如TLS 1.0版本
* 2、一个客户端生成的随机数，稍后用于生成"对话密钥"
* 3、支持的加密方法，比如RSA公钥加密
* 4、支持的压缩方法  
  PS：客户端发送的信息之中不包括服务器的域名，也就是说，理论上服务器只能包含一个网站，否则会分不清应该向客户端提供哪一个网站的提供的数字证书。这就是为什么通常一台服务器只能由一张数字证书的原因  
  对于虚拟主机的用户来说，这当然很不方便。2006年，TLS协议加入了Server Name Indication扩展，允许客户端向服务器提供它所请求的域名。

**2、服务器的配置**

采用HTTPS协议的服务器必须要有一套数字证书，可以是自己制作或者CA证书。区别就是自己办法的证书需要客户端验证通过，才可以继续访问，而使用CA证书则不会弹出提示页面。这套证书其实就是一堆公钥和私钥。公钥给别人加密使用，私钥给自己解密使用。服务器在接收到客户端的请求后，服务器需要确定加密协议的版本，以及加密的算法，然后也生成一个随机数。

**3、服务器首次回应**

把上个阶段生成的加密协议的版本，加密的算法，随机数以及自己的证书一起发送给客户端。这个随机数是整个过程的第二个随机数。  
返回的信息包括:

* 1、协议的版本，比如TLS1.0版本，如果浏览器与服务器支持的版本不一致，服务器关闭加密通信
* 2、加密的算法
* 3、随机数
* 4、服务器证书

除了上面这些信息，如果服务器需要确认客户端身份，就会再包含一项请求，要求客户端提供"客户端证书"。比如，金融机构往往只允许认证客户进入自己的网络，就会向正式客户提供USB密钥，里面包含了一张客户端证书。

**4、客户端验证证书**

客户端收到服务器回应以后，首先验证服务器证书。如果证书不是可信任机构颁布、或者证书中的域名和实际域名不一致、或者证书已经过期，就会向访问者显示一个警告，由其选择是否还要继续通信。  
PS:验证证书主要根据服务端发过来的证书名称，在本地寻找其低级证书，并一级一级直到根证书，验证各级证书的合法性。

**5、客户端传送加密信息**

验证证书通过后，客户端再次产生一个随机数(第三个随机数)，然后使用证书中的公钥进行加密，以及放一个ChangCipherSpec消息即编码改变的消息，还有整个前面所有消息的hash值，进行服务器验证，然后用新密钥加密一段数据一并发送到服务器，确保正式通信前无误。  
客户端使用前面的两个随机数以及刚刚新生成的新随机数(又称”pre-master key”)。  
简单的说下ChangeCipherSpec:  
ChangeCipherSpec是一个独立的协议，体现在数据包中就是一个字节的数据，用于告知服务器，客户端已经切换到之前协商好的加密套件(Cipher Suite)的状态，准备使用之前协商好的加密套件加密数据并传输了。  
这部分传送的是用证书加密后的随机值，目的就是让服务器得到这个随机值，以后客户端和服务其的通信就可以通过这个随机值来进行加密解密了。服务器与客户端之前的数据传输过程中是庸才对称加密方式加密的。

注意：  
此外，如果前一步，服务器要求客户端证书，客户端会在这一步发送证书及相关信息。

**6、生成会话密钥**

上面产生的随机数，是整个握手阶段的出现的第三个随机数，又称"pre-master key"。有了它之后，客户端和服务器同时有了三个随机数，接着双方就用事先协商的加密方法，各自生成本地会话所用的同一把"会话密钥"。  
PS:为什么一定要用三个随机数，来生成"会话密钥"？

答：不管是客户端还是服务器，都是下需要随机数，这样生成的密钥才不会每次都一样，由于SSL协议中证书是静态的，因此十分有必要引入一种随机因素来保证协商出的密钥的随机性。  
对于RSA密钥交换算法来说，pre-master-key本身就是一个随机数，再加上第一步、第三步消息中的随机数，三个随机数通过一个密钥导出器最终导出一个对称密钥。pre master 的存在在于SSL协议不信任每一个主机都能产生完全的随机数，如果随机数不随机，那么pre master secret就可能被猜出来，那么仅适用于pre master secret作为密钥就不合适了，因此必须引入新的随机因素，那么客户端和服务器加上pre master secret三个随机数一同生成的密钥就不容易被猜出了，一个伪随机数可能完全不随机，但是三个伪随机就十分接近随机了，每增加一个自由度，随机性增加的可不是一。

**7、服务器最后的回应**

服务器生成"会话密钥"后，向客户端最后发送下面信息：

* 1、编码改变通知，表示随后的信息都将用双方商定的加密方法和密钥发送。
* 2、服务器握手结束通知，表示服务器握手阶段已经结束。这一项同时也是前面所有内容的hash值，用来供客户端校验。

**8 客户端解密**

客户端用之前生成的私钥解密服务器传过来的信息，于是获取了解密后的内容

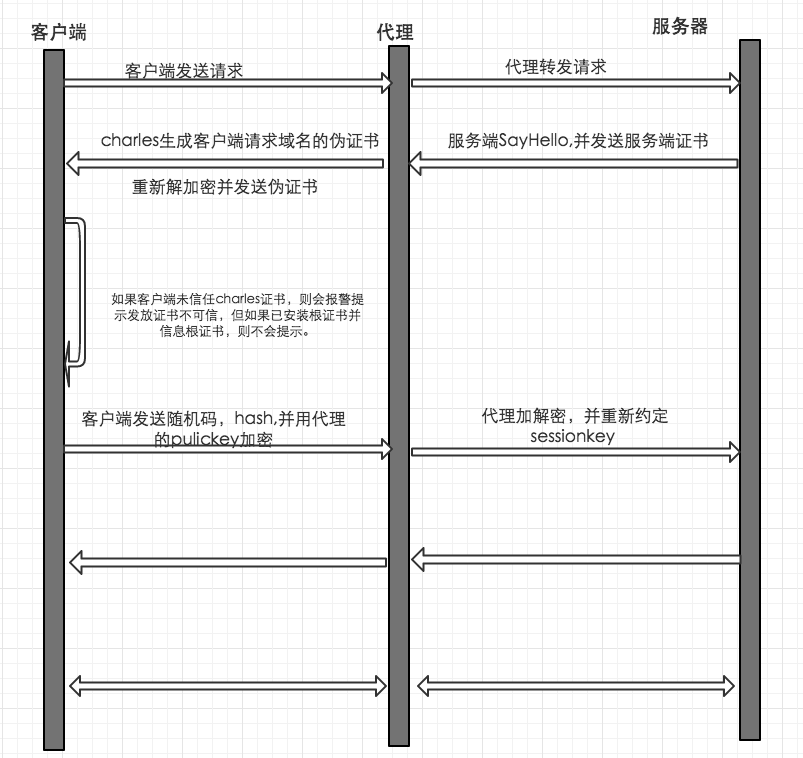
至此，整个握手阶段全部结束了。接下来，客户端与服务器进入加密通信，就完全是使用普通HTTP协议，只不过用"会话密钥"加密内容。

**注意事项**

SSL协议在握手阶段使用的是非对称加密，在传输阶段使用的是对称加密,也就是在说在SSL上传送数据是使用对称加密加密的!因为非对称加密的速度缓慢，耗费资源。其实当客户端和服务器使用非对称加密方式建立连接后，客户端和主机已经决定好了在传输过程使用的对称加密算法和关键的对称加密密钥，由于这个过程本身是安全可靠的，也即对称加密密钥是不可能被窃取盗用的，因此，保证了在传输过程中对数据进行对称加密也是阿安全可靠的！如果有人窃听通信，他可以知道双方选择的加密方法，以及三个随机数中的两个。整个通话的安全，只取决于第三个随机数(pre master secret)能不能被破解。

**(九)HTTPS与代理**

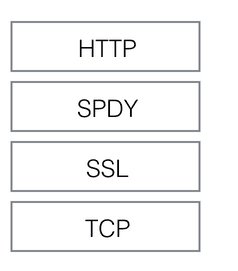
2.4 如何使用代理（charles）  
我们知道从HTTPS的整个原理可以知道，客户端和服务器进行通信的成果，客户端是能拿到数据的，代理也一定能拿到，包括公共密钥，证书，算法等，但代理无法获取服务器的私钥，所以无法获取5/6/7/8的会话密钥，也就无法得到数据传输的明文，所以默认的情况下，charles是无法抓https的。那如何让charles转包并获取明文？也就是让charles获取私钥，获取服务器的是不可能的，那只能在通信过程中使用charles自己的证书，并在通信的过程中主动为请求的域名发放证书，流程如下：

  
从上面可以看出，一旦信任代理的证书，代理在中间就做一个转换的角色，即担任客户端的服务器，又担任服务器的客户端，在中间传输的过程中做了加密解密转换。也就是说一旦信任代理，大力就可以干任何事。

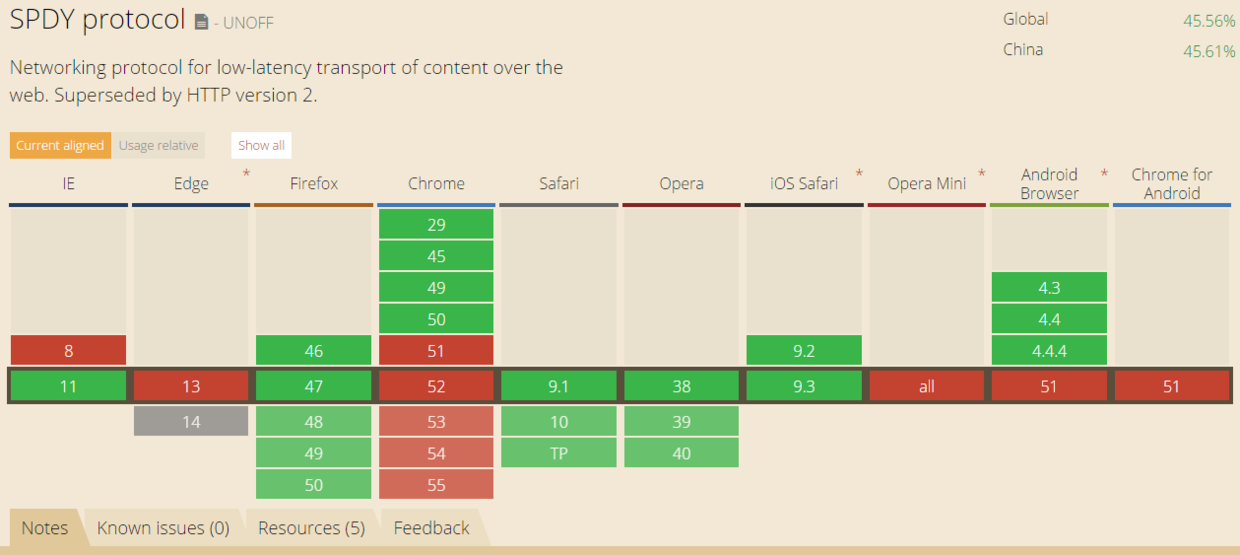
**四、SPDY**

2012年google如一声惊雷提出了SPDY的方案，大家猜开始从正面看待和解决老版本HTTP协议本身的问题，SPDY可以说是综合了HTTPS和HTTP两者有点于一体的传输协议，主要解决：

* 1、降低延迟:：针对HTTP高延迟的问题，SPDY优雅的采取了多路复用(multiplexing)。多路复用通过多个请求stream共享一个TCP连接的方式，解决了HOL blocking的问题，降低了延迟同事提高了带宽的利用率。
* 2、请求优先级：多路复用带来的一个新的问题是，在连接共享的基础上有可能会导致关键请求被阻塞。SPDY允许给每个request设置优先级，这样重要的请求就会优先得到相应。比如浏览器加载首页，首页的html内容应该优先展示，之后才是各种静态资源文件，脚本文件等加载，这样保证用户第一时间看到网页的内容。
* 3、header压缩：HTTP1.x的header很多时候都是重复多余的。选择和是的压缩算法可以减少包的大小和数量。
* 4、基于HTTPS的加密协议传输，大大提高了传输数据的可靠性。
* 5、服务端推送(server push)，采用SPDY的网页，例如一个网页有一个style.css请求，客户端在收到style.css数据的同事，服务端会将style.js文件推送给客户端，当客户端再次尝试获取style.js时就可以直接从缓存中获取到，不用再次发送请求了。SPDY构成图:



SPDY位于HTTP之下，TPC和SSL之上，这样可以轻松兼容老版本的HTTP协议，同时也可以使用已有的SSL功能。  
大家来看下SPDY的兼容性：

**五、HTTP2.0**

**(一)、HTTP2.0的前世今生**

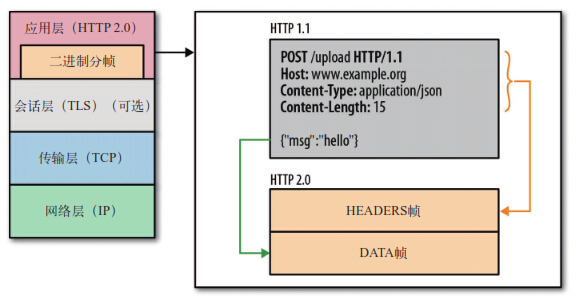
顾名思义有了HTTPS1.x，那么HTTP2.0也就顺理成章的出现了。  
在HTTP/1.x中，如果客户端想发起多个并行请求必须建立多个TCP连接，这无疑增大了网络开销。另外HTTP/1.x不会压缩请求和响应头，导致了不必要的网络流量，HTTP/1.x不支持资源优先级导致底层TCP连接利用率低下。而这些问题都是HTTP/2要着力解决的。HTTP2.0可以说是SPDY的升级版(其实也是基于SPDY设计的)，但是HTTP2.0跟SPDY仍有不同的地方，主要有以下两点：

* 1、HTTP2.0支持明文HTTP传输，而SPDY轻质使用HTTPS
* 2、HTTP2.0消息头的压缩算法采用HPACK，而非SPDY采用的DEFLATE

**(二)、 HTTP2.0的新特性**

* 1、新的二进制格式(Binary Format)：HTTP 1.x的解析是基于文本。基于文本洗衣的格式解析存在天然缺陷，文本的表现形式有多样性，要做到健壮性考虑的场景必然很多，二进制则不同，只认0和1的组合，基于这种考虑HTTP2.0的协议解析决定采用二进制分帧格式，实现方便且健壮。
* 2、多路复用(multiPlexing)：即连接共享，即每一个request都是用作连接共享机制的。一个request对应一个id，这样一个连接上可以有多个requst，每个连接的request可以随机的混杂在一起，接受方可以根据request的id将request再归属到各自不同的服务端请求里面，后面有一张多路复用原理图
* 3、请求优先级：把HTTP消息分解为很多独立的帧后，就可以通过优化这些帧的交错和传输顺序，进一步提供性能。为了做到这一点，每个流都有一个带有31比特的的优先值
* 4、header压缩：HTTP1.x的header带有大量的信息，而且每次都要重复发送，HTTP2.0使用encoder来减少需要传输的header大小，通讯双方各自cache一个header field表，既避免了重复的header传输，又减少了需要传输的大小。
* 5、服务端推送(server push)：同SPDY一样，HTTP2.0也具有server push功能。

HTTP2.0性能增强的核心，全在于新增的二进制分帧层，它定义了如何封装HTTP消息并在客户端与服务器之间传输

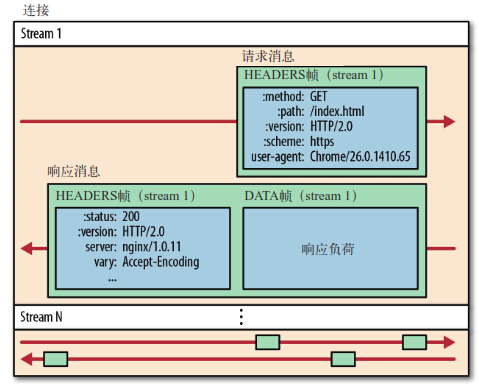
所以HTTP/2引入了三个新的概念：

* 1、数据流：基于TCP连接之上的逻辑双向字节流，对应一个请求及响应。客户端每发起一个请求就建立一个数据量就，后续该请求及其响应的所有数据都通过该数据流传输。
* 2、消息：一个请求或者响应对应的一系列数据帧
* 3、帧”：HTTP/2的最小数据切片单位，每个帧包含帧首部，至少也会标示出当前帧所属的流。

上述概念之间的逻辑关系：

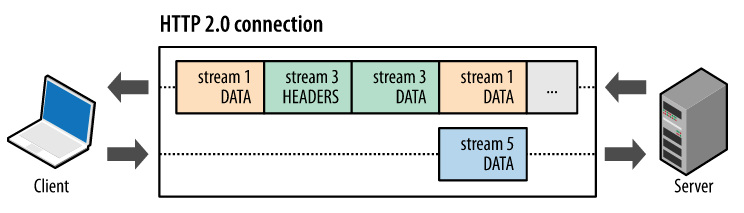
* 1、所有通信都在一个TCP连接上完成，此连接可以承载任意数量的双向数据流。
* 2、每个数据流都有一个唯一的标识符和可选的优先级信息，用于承载双向消息。
* 3、每条消息都是一条逻辑HTTP消息(例如请求或者响应)，包含一个或多个帧。
* 4、帧是最小的通信单位，承载着特定类型的数据，例如HTTP标头、消息负载等等。来自不同数据流的帧可以交错发送，然后再根据每个帧头的数据流标识符重新组装。
* 5、每个HTTP消息分分解为多个独立的帧后可以交错发送，从而在宏观上实现了多个请求或者响应并行传输的效果。这类似于多进程环境下的时间分片机制。

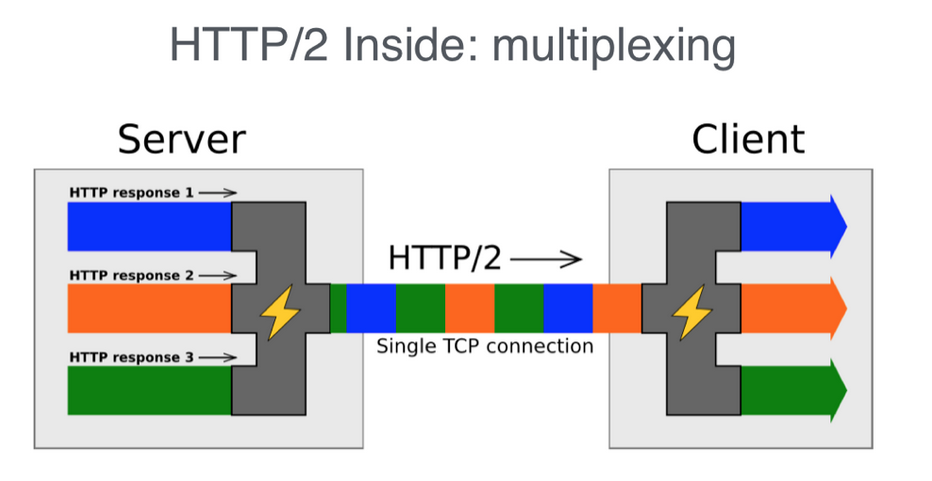
所有HTTP 2.0通信都在一个连接上完成，这个连接可以承载任意数量的双向数据流。每个数据流以消息的形式发送。而消息由一或多个帧组成，而这些帧可以乱序发送，然后再根据每个帧首部流标识符重新组装。

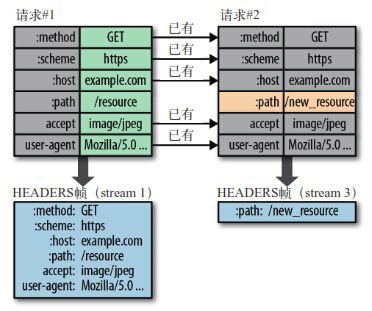


简而言之，HTTP2.0把HTTP协议通信的基本单位缩小为一个一个的帧，这些帧对应着逻辑流中的消息。相应地，很多流可以并行地在同一个TCP连接上交换消息。

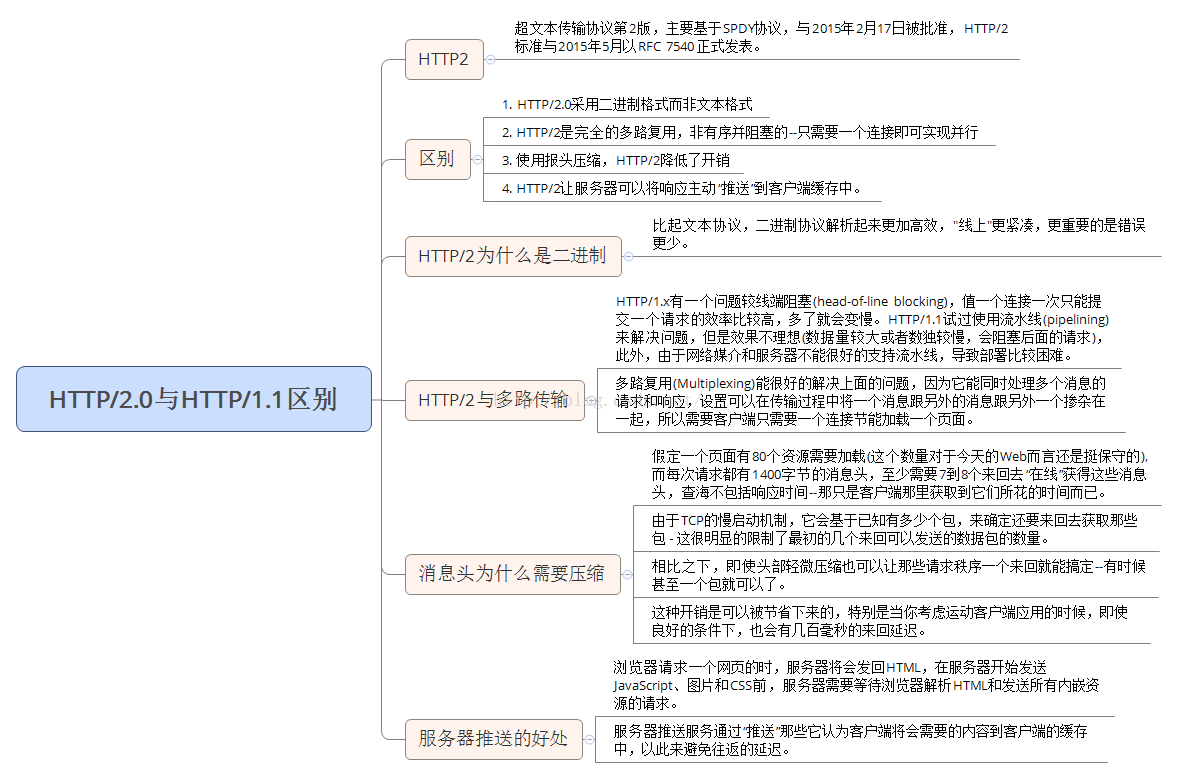
在HTTP/1.1中，如果客户端想发送多个平行的请求以及改进性能，必须使用多个TCP连接。HTTP2.0的二进制分帧层突破了限制；客户端和服务器可以把HTTP消息分解为互不依赖的帧，然后乱序发送，最后再把另一端把它们重新组合起来。如下图

下面这个图更形些

在HTTP/1.x中，头部的数据都是纯文本格式。通常会增加500-800字节的负荷。为了减少开销提高性能，HTTP/2压缩头部数据，由于HTTP2.0连接的两端都知道已经发送了哪些头部，这些头部的值是什么，从而可以针对之前的数据编码发送差异数据。如下图：



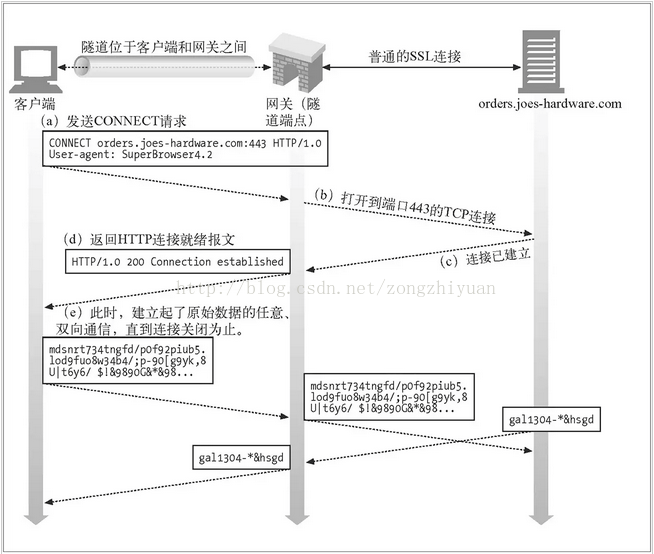
**(三)HTTP2.0与HTTP1.1的区别**

**六、隧道**

Web隧道(Web tunnel)是HTTP的另一种用法，可以通过HTTP应用程序访问非HTTP协议的应用程序，Web隧道允许用户允许用户通过HTTP连接发送非HTTP流量，这样就可以在HTTP上捎带其他协议数据了。使用Web隧道最常见的原因就是要在HTTP连接中嵌入非HTTP流量。这样这类流量就可以穿过只允许Web流量通过的防火墙了。

**(一) 建立隧道**

Web隧道是用HTTP的CONNECT方法建立起来的。CONNECT方法请求隧道网管创建一条到达任一目的服务器和端口的TCP连接，并对客户端和服务器职期间的后续数据进行盲转发。  
下图显示了CONNECT方法如何建立起一条到达网管的隧道，来自《HTTP 权威指南》



* 1、(a)是客户端相互发送了额一条CONNECT请求给隧道网关。客户端的CONNECT方法请求隧道网关打开一条TCP连接(在这里，打开的是到主机orders.joes-hardware.com的标准SSL端口443的连接)；
* 2、(b)和(c)中创建了TCP连接，一旦建立了TCP连接，网管就会发送一条HTTP200 Connection Established响应来通知客户端，此时，隧道就建立起来了。客户端通过HTTP隧道发送的所有数据都会被直接转发给输出TCP连接，服务器发送的所有数据都会通过HTTP隧道转发给客户端。

上图中的例子描述了一条SSL隧道，其中的SSL流量是在一条HTTP连接上发送的，但是通过CONNECT方法可以与使用任意协议的任意服务器建立TCP连接。

1、CONNECT请求  
除了起始行之外，CONNECT的语言与其他HTTP方法类似。一个后面跟着冒号和端口号的主机名取代了请求的URL。主机和端口都必须制定:

CONNECT home.netscape.com：443 HTTP/1.0

User-agent： Mozilla/4.0

和其他HTTP报文一样，起始行之后，有零个或多个HTTP请求首部字段。这些行照例以CRLF结尾，首部列表以一个空的CRLF结束。  
2、CONNECT响应  
发送了请求之后，客户端会等待来网管的响应，和普通HTTP报文一样，响应码表示成功。按照惯例，响应中的原因短语通常设为"Connection Established"。

HTTP/1.0 200 Connection Established

Proxy-agent: Netscape-Proxy/1.1

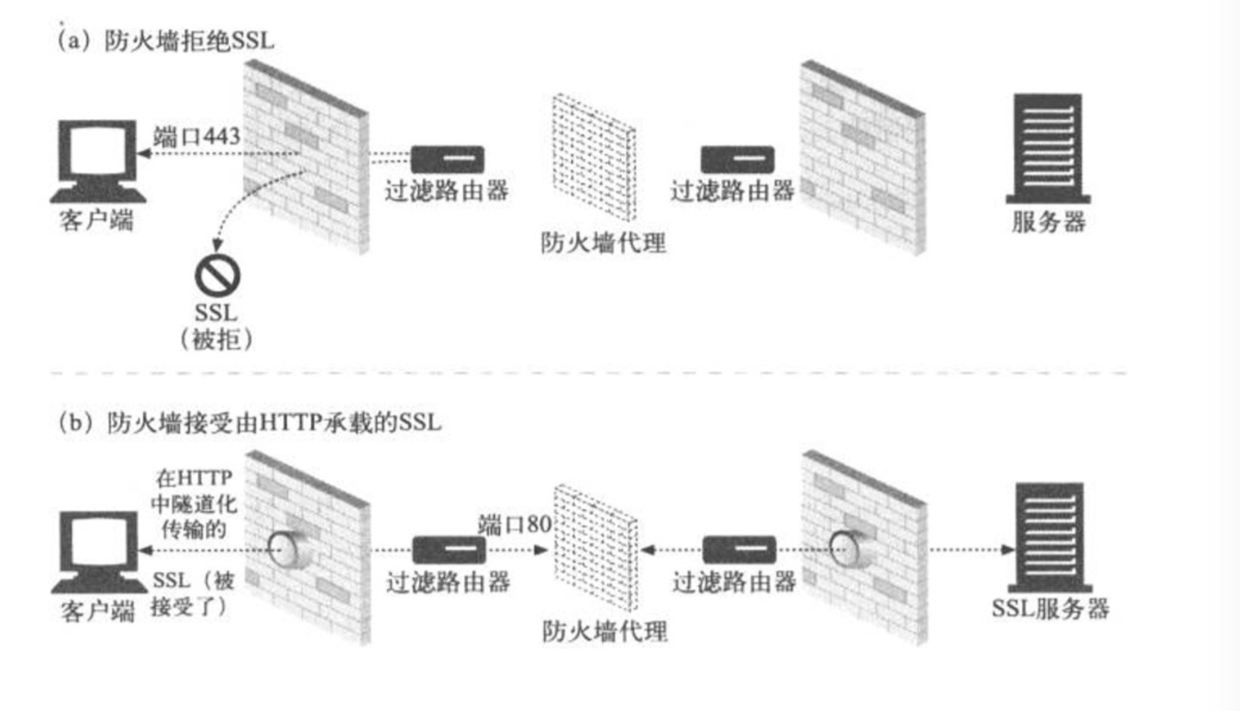
与普通HTTP响应不同，这个响应并不需要包含Content-Type首部。此时连接只是对原始字节进行转接，不再是报文的承载者，所以不需要使用内容类型。

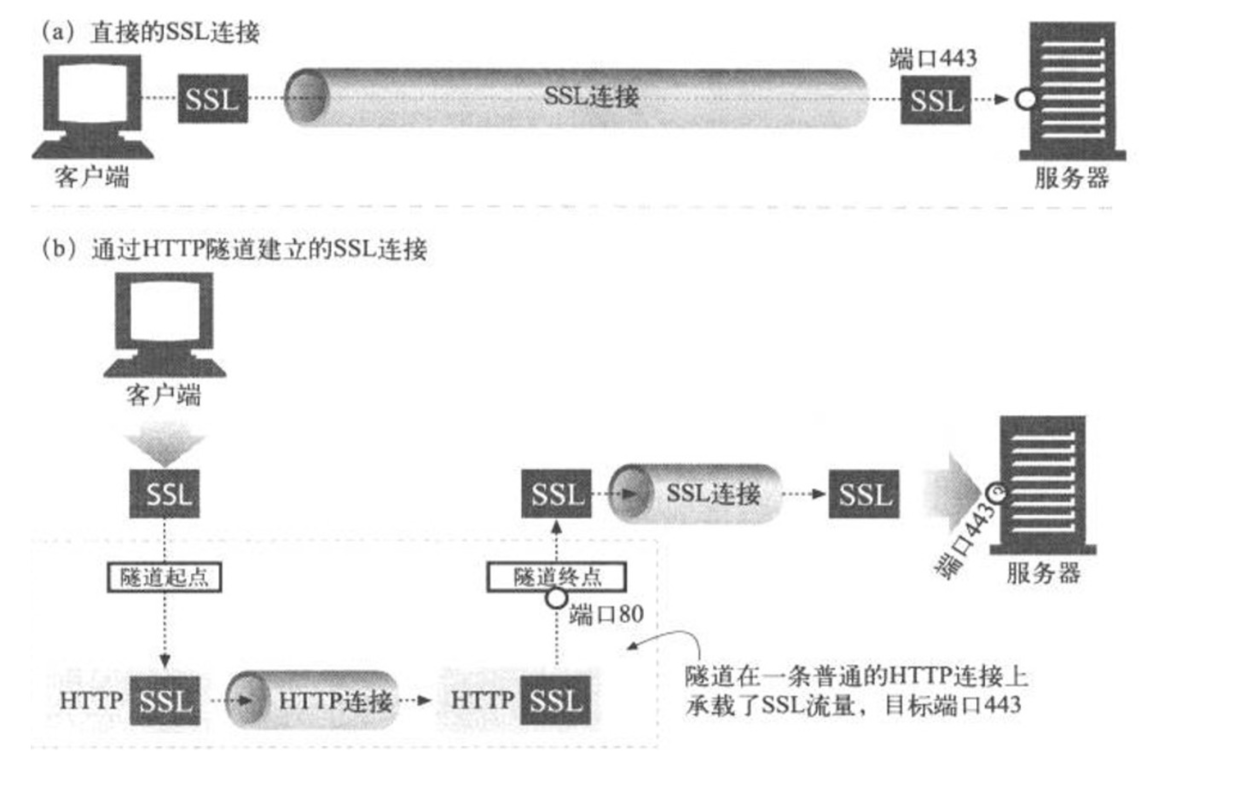
管道化数据对网管是不透明的，所以网管不能对分组的顺序和分组流做任何假设。一旦隧道建立起来了，数据就可以在任意时间流向任意方向了。

作为一种性能优化方法，允许客户端在发送了CONNECT请求之后，接受响应之前，发送隧道数据。这样可以更快的将数据发送给服务器，但这就意味着网管必须能够正确处理跟在请求之后的数据。尤其是，网管不能假设网络I/O请求只会返回首部数据，网管必须确保在连接准备就绪时，将与首部一同读进来的数据发送给服务器。在请求之后，或者其他非200但不致命的错误状态，就必须做好重发请求数据的准备。如果在任意时刻，隧道的任意一个端点断开连接，那个端点发出的所有未传输数据都会被传送给另一个端点，之后到另一个端点的链接也会被代理终止。如果还有数据要传输给关闭连接的端点，数据会被丢弃。

**(二)SSL隧道**

最初开发Web隧道是为了通过防火墙来传输加密的SSL流量。很多组织都会将所有流量通过分组过滤路由器和代理服务器以隧道方式传输，以提升安全性。但有些协议，比如加密SSL，其信息是加密的的，无法通过传统的代理服务器转发。隧道会通过一条HTTP连接来传输SSL流量，以穿过端口80的HTTP防火墙。

为了让SSL流量经现存的代理防火墙进行传输，HTTP中添加了一项隧道特性，在此特性中，可以将原始的加密数据放在HTTP报文中，通过普通的HTTP信道传送。



* a图代表一个SSL连接，SSL流量直接发给了一个(SSL端口443上的)安全Web服务器。
* b图代表了SSL流量被封装到一条HTTP报文中，并通过HTTP端口80的连接发送，最后解封装为普通的SSL连接。

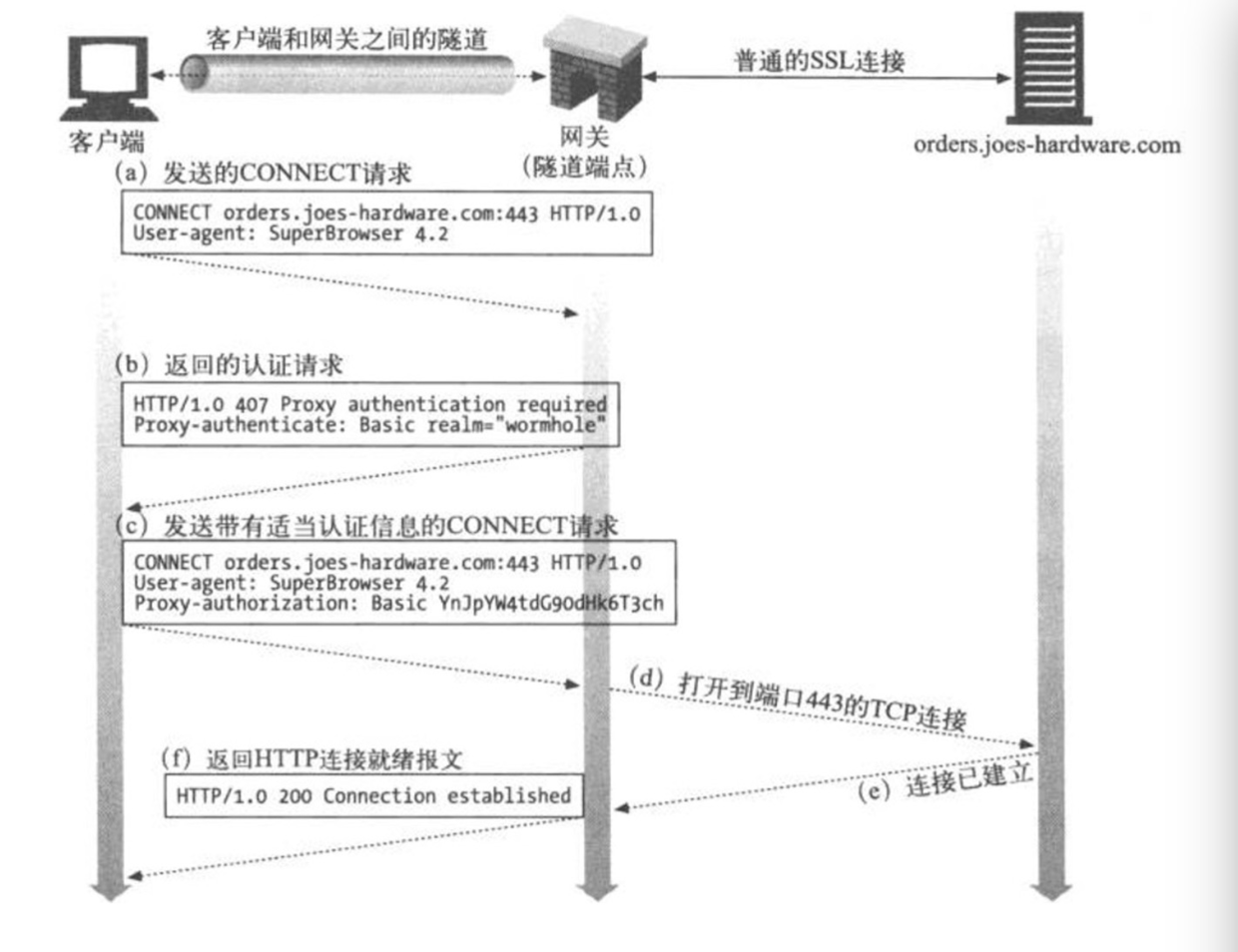
通常会用隧道将非HTTP流量传过端口过滤防火墙。这一点可以得到很好的利用。比如，通过防火墙传输完全SSL流量。但是，这项特性可能会被滥用，使得恶意协议通过HTTP隧道流入某个组织内部。  
可以像其他协议一样，对HTTPS协议(SSL上的HTTP)进行网管操作：由网关(而不是客户端)初始化与远端HTTPS服务器的SSL会话，然后代表客户端执行HTTPS事务。响应会由代理接受并解密，然后通过(不安全的)HTTP传送给客户端。这是网关处理FTP的方式。但这种方式有几个缺点：

* 1、客户端到网关之间的链接是普通的非安全HTTP；
* 2、尽管代理是已认证主体，但客户端无法对远端服务器执行SSL客户端认证(基于X509证书的认证)；网关要支持完整的SSL实现

　　可以像其他协议一样，对HTTPS协议(SSL上的HTTP)进行网关操作：由网关(而不是客户端)初始化与远端HTTPS服务器的SSL会话，然后代表客户端执行 HTTPS事务。响应会由代理接收并解密，然后通过(不安全的)HTTP传送给客户端。这是网关处理FTP的方式。但这种方式有几个缺点：客户端到网关之间的连接是普通的非安全HTTP；尽管代理是已认证主体，但客户端无法对远端服务器执行SSL客户端认证(基于X509证书的认证)；网关要支持完整的SSL实现。

对于SSL隧道机制来说，无需在代理中实现SSL。SSL会话是建立在产生请求的客户端和目的(安全的)Web服务器之间的，中间的代理服务器只是将加密数据经过隧道传输，并不会在安全事务中扮演其他的角色。

在适当的情况下，也可以将HTTP的其他特性与隧道配合使用。尤其是，可以将代理的认证支持与隧道配合使用，对客户端使用隧道的权利进行认证。

总的来说，隧道网管无法验证目前使用的协议是否就是它原本打算经过的隧道的协议。因此，比如说，一些修换捣乱的用户可能会通过本打算用于SSL的隧道，越过公司防火墙传递因特网的游戏流量，而恶意用户可能会用隧道打开Telnet会话，或用隧道绕过公司的E-mail扫描器来发送E-mail。为了降低对隧道的滥用，网关应该只为特定的知名端口，比如HTTPS端口443打开隧道。

**七、代理**

Web代理是一种存在于网络中间的实体，提供各式各样的功能。现在网络系统中，Web代理无处不在。

**(一)代理的作用**

* 1、提高访问速度。因为客户端要求的数据存于代理服务器的硬盘中，因此下次这个客户端或者其他要求相同的站点数据时，就会直接从代理服务器的硬盘中读取，代理服务器起到了缓存的作用，对热门站点有很多客户访问时，代理服务器的优势更为明显。
* 2、Proxy可以起到防火墙的作用，因为所有代理服务器的用户都必须通过代理服务器访问远程站点，因此在代理服务器上就可以设置相应的限制，以过滤或屏蔽掉某些信息。这是局域网网管对局域网用户访问限制最常用的办法，也是局域网用户为什么不能浏览某些网站的原因。拨号用户如果使用代理服务器，同样必须服从代理服务器的访问限制，除非你不使用这个代理服务器。
* 3、通过代理服务器访问一些不能直接访问的网站，互联网上有许多开放的代理服务器，客户在访问权限受到限制时，而这些代理服务器的访问权限是不受限制的，刚好代理服务器在客户的访问范围之内，那么客户通过代理服务器访问目标网站就能成为可能。国内高校的多使用教育网，不能出国，但通过代理服务器，就能实现访问因特网，这就是高校内代理服务器热的原因。
* 4、安全性得到提高。无论在上聊天室还是浏览网站，目标网站只知道你来自代理服务器，而你的真实IP就无法预测，这就使得使用者的安全性得以提高

**(二)、代理服务器工作流程**

* 1、当客户端A对Web服务器请求时，A的请求会首先发送到代理服务器。
* 2、代理服务器接收到客户端请求后，会检查缓存中是否存有客户端所需要的数据。
* 3、如果代理没有客户端A请求的数据，它将会向Web服务器提交请求
* 4、Web服务器响应请求的数据
* 5、代理服务器向客户端A转发Web服务器的数据
* 6、客户端B访问Web服务器，向代理服务发出请求。
* 7、代理服务器查找缓存记录，确认已经存在Web服务器相关数据。
* 8、代理服务器直接回应查询的信息，而不需要再去服务器进行查询，从而达到节约网络流量和提高访问速度的目的

**(三)、代理的形式**

HTTP代理存在两种形式，分别如下：

* 第一种是RFC 7230 -HTTP/1.1：Message Syntax and Routing(即修订后的RFC 2616，HTTP/1.1 协议的第一部分)描述的普通代理。这种代理扮演的是"中间人“的角色，对于连接到它的客户端来说，它是服务端，对于要连接的服务器来说，它是客户端。它就是负责在两端之间来回传送HTTP报文。
* 第二种是Tunneling TCP basedprotocols through Web proxy servers (通过Web代理服务器用隧道方式传输基于TCP协议)描述的隧道代理。通过HTTP协议正文部分(Body)完成通讯，以HTTP方式实现任意基于TCP应用层协议代理。这种代理使用HTTP的CONNECT方法建立连接，但是CONNECT 最开始并不是RFC 2616 -HTTP/1.1的一部分，直到2014年发布的HTTP/1.1修订版中，才增加了对CONNECT及隧道代理的描述，详见RFC 7231- HTTP/1.1:Semantics andContent。实际上这种代理早就被广泛实现。

PS:事实上，第一种代理，对应<HTTP权威指南>一书中第六章"代理"，第二种代理，对应第八章"集成点：网关、隧道及中继"中的8.5小节"隧道"。

**1、普通代理**

原理：HTTP客户端向代理服务器发送请求报文，代理服务器需要正确地处理请求和连接(例如正确处理(conntion:keep-alive)，同时向目标服务器发送请求，并将受到的响应装发给客户端。

* 假如我通过代理访问A网站，对A来说，它会把代理当做客户端，完全察觉不到真正的客户端的存在，这实现了隐藏客户端IP的目的，当然代理也可以修改HTTP请求头部，通过X-Forwarded-IP这杨的自定义头部告诉服务器真正的客户端IP。但服务器无法验证这个自定义头部真的是由代理添加，还是客户端修改了请求头，所以从HTTP头部字段取IP时，需要格外小心。
* 给浏览器显示的指定代理，需要手动修改浏览器或相关设置，或者指定PAC(Proxy Auto-Configuration，自动配置代理)文件自动设置，还有些浏览器支持WPAD(Web ProxyAutodiscovery Protocol ,Web代理自动发现协议)。显示指定浏览器代理这种方式一般称之为正向代理，浏览器齐总正向代理后，会对HTTP请求报文做一些修改，来规避老旧代理服务器的一些问题。
* 还有一种情况是访问A网站时，实际上访问的是代理，代理收到请求报文后，再向真正提供服务的服务器发起请求，并将响应转发给浏览器。这种情况一般被称为反向代理，它可以用来隐藏服务器IP及端口，一般使用反向代理后，需要通过修改DNS让域名解析到代理服务器IP，这是浏览器无法察觉真正服务器的存在，当然也就不需要修改配置了。反向代理是Web系统最为常见的部署方式。

**2、隧道代理**

原理：HTTP客户端通过HTTP的CONNECT方法请求隧道代理，创建一条到达任意目的服务器和端口的TCP连接，并对客户端和服务器之间的后继数据进行盲转发。  
具体请查看本片文章第五部分

这里将HTTP隧道分为两种：

* 1、不使用CONNECT的隧道  
  不实用CONNECT的隧道，实现了数据包的重组和转发。在Proxy收到来自客户端的HTTP请求后，会重新创建Request请求，并发送到目标服务器。当目标服务器返回Response给Proxy之后，Proxy会对Response进行解析，然后重新组装Resposne，发送给客户端。所以在不使用CONNECT方式建立的隧道，Proxy有机会对客户端与目标服务器之间的通信数据进行窥探，而且有机会对数据进行串改。
* 2、使用CONNECT的隧道:而对于使用CONNECT的隧道则不同。当客户端向Proxy发起HTTP CONNECT Method的时候，就是告诉Proxy，先在Proxy和目标服务器之间先建立起连接，在这个连接建立起来之后，目标服务器会返回一个回复给Proxy，Proxy将这个回复转发给客户端，这个Response是Proxy跟目标服务器连接建立的状态回复，而不是请求数据的Response。在此之后，客户端跟目标服务器所有的通信豆浆使用值前简建立来的连接。这种情况下的HTTP隧道，Proxy仅仅实现转发，而不会关心转发的数据，这也就是为什么在使用Proxy的时候，HTTPS请求必须首先使用HTTP CONNECT建立隧道。因为HTTPS的数据都是经过加密的，Proxy是无法对HTTPS的数据进行解密的，所只能使用CONNECT，仅仅对通信数据进行转发。

**3、与proxy有关的字段**

* X-Forwarded-For(XFF):是用来识别通过HTTP代理或者负载均衡方法连接到Web服务器的客户端最原始的IP地址的HTTP请求头字段。Squid:缓存代理服务器的开发人员最早引入了这一HTTP头字段，如果该没有XFF或者另外一个种相似的技术，所有通过代理服务器的连接只会显示代理服务器的IP地址(而非连接发起的原始IP地址)，这雅漾的代理服务器实际上充当了匿名服务提供者的角色，如果连接的原始IP地址不可得，恶意访问的检查与预防的难度将大大增加
* X-Forwarded-Host和X-Forwarded-Proto分别记录客户端最原始的主机和协议
* Proxy-Authorization:连接到Proxy的身份验证信息
* Proxy-connection:它不是标准协议的一部分，标准些协议中已经存在一种机制可以完成此协议头的功能，这就是Connection头域，与Proxy-Connection头相比，Connection协议头几乎提供了相同的功能，除了错误部分。而且，Connection协议头可用于任意连接之间，包括HTTP服务器，代理，客户端，而不是像Proxy-Connection一样，只能用于代理服务器和客户端之间。

**八、InetAddress类和InetSocketAddress类**

**(一)、InetAddress(IP地址)**

InetAddress是IP地址(也是主机地址)类封装计算机的IP地址和DNS，不包含端口。

**1、如何获取对象:**

创建主机地址对象要靠InetAddress类的几个静态工厂方法:

* InetAddress getByAddress(byte[] addr):仅根据IP地址的各个字节创建IP地址对象。
* InetAddress getLocalHost():返回本机IP地址对象
* InetAddress getByAddress(String host,byte[] addr): 根据主机名和IP地址的各个自己创建IP地址对象。要把IP地址的高字节放在addr的低索引处。
* InetAddress getByName(String host)：仅根据主机名创建IP地址对象。函数会访问DNS服务器来获取host对象的IP地址。

注：主机名既可以是域名，也可以是IP地址的字符串形式。例如，

InetAddress.getByName("www.163.com")

InetAddress.getByName("192.168.2.23")

**2、方法**

* String toString():返回"主机名/IP地址"字符串
* String getHostName() :返回此IP地址的主机名
* byte[] getAddress():返回此IP地址的各个字节，高字节放在地索引处
* String getHostAddress():返回此IP地址的字符串形式
* Boolea isReachable(int timeout):测试此IP地址的可达性，最多等待timeout毫秒。
* Boolean equal(Object obj)：如果此IP地址的getAddress()结果与obj.getAddress()不仅数组长度相同且每个元素也相同，则返回true。

**(二)、InetSocketAddress类**

InetSocketAddress是在InetAddress基础上封装了端口号。所以说InetSocketAddress是(IP地址+端口号)类型，也就是端口地址类型。  
它的基类是SocketAddress类，SocketAddress类里面什么都没有。

**1、如何获取对象:**

* InetSocketAddress(int port):创建IP地址为通配符地址，端口号为port的端口地址。
* InetSocketAddress(InetAddress addr,int port):创建IP地址addr，端口号为port的端口地址。
* InetSocketAddress(String host,int port)：根据主机名和端口号创建端口地址。函数会访问DNS服务器来解析出host和IP地址的，如果解析失败，则标记为"未解析"
* InetSocketAddress createUnresolved(String host,int port):同上，但是不会访问DNS服务器去解析主机名，而是直接标记为"未解析"。

**2、方法**

* boolean isUnsolved():如果尚未解析出IP地址，则返回true。
* String toString():返回"主机名/IP地址：端口地址"字符串
* InetAddress getAddress():返回此端口的IP地址。
* String getHostName():返回此端口的主机名。
* int getPort():返回此端口的端口号。
* boolean equals(Object obj):如果此端口与obj端口的IP地址和端口号都相等，则返回true。

**(三)、两者的却别**

关键就是在InetSocketAddress不基于任何协议，一般用于socket编程。

* 表面看InetSocketAddress多了一个端口号，端口的作用：一台拥有IP地址的主机可以提供许多服务，比如Web服务、FTP服务、SMTP服务等，这些服务完全可以通过1个IP地址来实现。
* 那么主机怎么区分不同的网络服务？显然不能只靠IP地址，因此IP地址与网络服务的关系是一对多的关系。
* 实际上是通过"IP地址+端口号"来区分不同的服务的。

转载至：http://www.jianshu.com/p/116ebf3034d9