#### 网络的字节序

大小端字节序：

小端字节序：低位存放在低地址

大端字节序：低位存放在高地址

网络字节序是大端字节序，有些系统的本机字节序是小端字节序，有些是大端字节序，所以为了保证传送顺序一致性，统一使用大端字节序

##### tcp三次握手、四次挥手

三次握手：

**第一次握手**：客户端发送SYN=1，ACK=0，seq=x的请求连接报文，客户端进入SYN-SEND状态；服务器端接收后，内核为这个连接维持一个半连接状态，并将半连接状态相关的数据放在一个未连接队列中。

**第二次握手**：通过第一次握手建立的未连接状态发送SYN=1，ACK=1，ack=x+1，seq=y的确认受到并请求连接报文，之后服务器端进入到SYN-RCVD状态；

**第三次握手**：客户端接收到第二次握手的报文后，发送第三次握手的ACK=1，seq=y+1，ack=x+1的确认连接报文，服务器接收到第三次握手报文后，将关于本次连接的处于未连接状态的数据从未连接队列中，转移到已连接队列。

最终分配套接字：三次握手后并没有直接分配套接字，而是将连接信息存储在已连接队列中，需要调用accept来从已连接队列的队列首取出节点，并为之分配地址和套接字，然后才开始发送数据，服务器接收到第三次握手报文后立刻进入到连接状态。

四次挥手：

**第一次挥手**：客户端与服务器端处于连接状态，客户端发送FIN=1，ACK=1， seq = u，的断开连接请求报文，客户端进入到FIN-WAIT1状态；

**第二次挥手**：服务器接收后，立即发送ACK=1, ack=u+1, seq=v的确认收到的报文，并断开客户端发送信息，服务器接收信息的那一条连接，但保留服务器发送信息的连接，进入到CLOSE-WAIT状态；客户端接收到第二次挥手信息进入FIN-WAIT2状态，tcp连接进入半关闭状态，此时只能由服务器发送数据，客户端接收数据。

**第三次挥手**：服务器端发送完毕，向客户端发送FIN=1，ACK=1，ack=u+1，seq=w的断开连接报文，服务器进入到LAST-ACK状态

**第四次挥手**：客户端接收到第三次挥手信息后，法送ACK=1， ack=w+1, seq=u+1的确认报文，进入TIME-WAIT状态，时长2MSL，服务器接收到后，进入CLOSE状态

**TIME-WAIT状态：**

1. 为了防止服务器端没有收到第四次握手的确认报文，假如没有收到确认报文，那么会服务器的第三次挥手会超时重传，处于TIME-WAIT的客户端这时候可以发送确认报文，以关闭服务器，避免浪费资源

2. 在经过2msl时间后，本次连接所产生的报文会全部消失，可以保证下一次连接不会收到上一次连接所产生的的报文（TIME\_WAIT会占用端口内容，不释放端口（我理解的TCP连接，是两个进程的连接，这个时候进程是唯一的，端口是唯一的，但是IP地址是共享的），这样可以保证下次两个端口建立连接的时候，不会收到上一次相同连接所进行交流的任何报文。

1. tcp和udp的区别，概念以及适用范围
2. TCP是面向连接的一条双工可靠信道，UDP是一条无连接的不可靠信道。
3. TCP在传输数据时必须先建立连接，传送数据结束后，接收端应发送确认，保证信息到达，数据传输完全结束后，要释放连接；UDP在传送数据之前不需要建立连接，远地主机接收到UDP数据报后，不需要给出确认。
4. TCP不提供广播或多播服务，而UDP提供广播或多播服务
5. UDP只有一个字节的检验和作为数据检验；TCP引入了诸如确认，流量控制计时器，连接管理和拥塞管理等以确保提供可靠的，面向连接的运输服务
6. UDP发送的报文长度是应用程序给出的；TCP所发出的报文长度是根据对方给出的窗口值和当前的网络拥塞情况来决定一个报文中有多少字符

适用范围：

TCP：文件传送（FTP文件传送协议），电子邮件，万维网，远程终端接入

UDP：文件传送（TFTP简单文件传送协议），广播，路由器配置，IP地址配置，域名系统，网络管理，ip电话，流式多媒体通信和网络文件服务器等

1. socket编程，accept方法是干什么的，在第三次握手中属于第几次

accept的作用是从已经建立连接的客户端队列头取出队首，并为之分配一个地址和文件描述符

在第三次握手之后，因为函数作用说的很明白，在已经建立连接的客户端队列中取出队首，而三次握手的目的就是为了建立连接，那么accept的时候，三次握手一定已经结束了

1. TCP是怎么保证有序传输的
2. TCP为每一个字节标号，为保证数据报的可靠传递，发送方必须把已经发送的数据保留在缓冲区
3. 为每一个已经发送的数据报维护一个超时定时器
4. 如果定时器超时之前收到了对方发送的应答信息，则释放该数据报占用的缓冲区
5. 否则重传该数据报，直到应答或重传此数据超过规定的最大次数为止
6. 接收方收到数据报后，先进行CRC循环冗余校验，如果正确，则把数据交给上层协议，然后发送一个确认号x+1的累计应答包，表明截止到x号的所有数据都已经收到了
7. TCP拥塞机制、流量控制

拥塞机制：

拥塞控制方法：慢开始、拥塞避免、快重传和快恢复

cwnd(拥塞窗口)，SMSS（发送方的最大报文段的数值），ssthresh（慢开始门限）

**慢开始：**将cwnd设置为1，由小到大逐步增加拥塞窗口的值，拥塞窗口每次的增量为min（N，SMSS），N为原先未确认的，但现在刚刚收到的确认报文所确认的字节数，使用慢开始算法，每经过一个伦次，cwnd翻倍

**拥塞避免：**当cwnd的值大于ssthersh之后将开始使用拥塞避免算法，使拥塞窗口缓慢的增大，每经过一个往返的时间，拥塞窗口cwnd加1，把拥塞窗口控制为按照线性规律增长，使网络不易出现拥塞

**超时：**如果出现网络拥塞导致的超时，这时候将当前的ssthresh调整为cwnd/2，然后将cwnd设置为1，重新开始启动慢开始算法

**快速重传：**快速重传也是超时，不同的是，它是在出现报文丢失的情况下，假设M3丢失，接收端无论接下来接收到任何信息，都要发送对M2的确认，连续收到3个M2的确认，那么将判断为M3丢失，立即进行重传。

**快速恢复：**快速重传后，因为没有发生网络拥塞，所以不必设置cwnd为1，将cwnd与ssthresh同时设置为cwnd/2，后开始执行拥塞避免算法

**流量控制：**

**基础知识：**TCP在发送数据的时候，是有一个发送缓存以及接收缓存的，如果一次性发送的数据接收缓存存不下，会造成数据丢失（包括发送的数据包大小大于接收缓存，接收缓存中有未读完的信息，添加新的信息后，接收缓存装不下等情况）

**操作：**流量控制首先是TCP的报文段中存在一个窗口字段，服务器给出的所有的确认（包括建立连接的时候作为第一条握手报文确认报文的第二次握手报文）要同过发送自己的接收窗口，客户端在接收到这个接收窗口的大小后，调整自己的发送窗口，一定不大于对方的接收窗口。（接收窗口的大小是动态变化的）

这个时候会出现一个问题，就是对方接收窗口未0的时候，根据上述的规则，客户端就不发送数据了，这样也就没法得到对方的接收窗口大小了。所以，客户端在收到接收窗口未0的时候，要维护一个定时器，这个定时器超时的时候，要发送一个探测报文，里面有一个字节的数据，服务器在收到后，要对这个字节给出确认，确认报文中就存在了服务器的当前接收窗口的大小，如果还是0，重新定时，如果不为0，进行正常的数据传输

1. 我们知道UDP是不可靠传输，如果让你设计一个基于UDP的差不多可靠的算法，怎么设计？

\*\*\*准确性待定

从tcp协议吸取经验

1. 数据完整性上，加上一个16位或32位的验证码
2. 乱序的，加上数据报序列号
3. 丢包，引入确认和重传机制（在数据报包头位置加入确认号，udp序号）每次收到一定数据后进行阻塞，向客户端发送收到信息的确认）
4. 协议字段

或者

1. UDP数据报分为报头和报体，报头位置存储数据报信息
2. 类型、传输状态、报体序列号、传输的总报数
3. 类型可以标记0 文字类型， 1图片类型，2文件类型等
4. 传输状态可以写0客户端发送给服务器要传大文件，1 服务器发送给客户端准备好接收， 2 客户端发送文件中， 3客户端发送完成， 4接收完成等
5. 序列号为发送报在分割后的数据报中的位置
6. 大文件传输的分割报总数
7. TCP粘包半粘包和拆包问题怎么处理

粘包问题：

如果TCP数据报很短，远小于TCP发送缓冲区的大小，TCP将多次写入缓冲区的数据一次发送出去。将会发生粘包现象

接收数据端的应用层没有及时的读取缓冲区中的数据，也会发生粘包

拆包问题：

要发送的数据大于TCP发送缓冲区剩余空间的大小，将会发生拆包

要发送的数据大于MSS（最大报文长度），TCP在传输前将进行拆包

解决方法：

1. 发送端给每个数据包添加包首部，首部中应至少包含数据包的长度，这样接收端在接收到数据后，通过读取包首部的长度字段，便知道一个数据包的实际长度了
2. 发送端将每个数据包封装为固定长度（不够的的可以用0来补充），这样接收端每次从接收缓冲中读取固定长度的数据就自然而然的把每个数据包拆分开来
3. 在数据包之间添加特殊符号来设置边界，这样通过边界将不同的数据包拆分开
4. 什么是keepalive？如何使用？

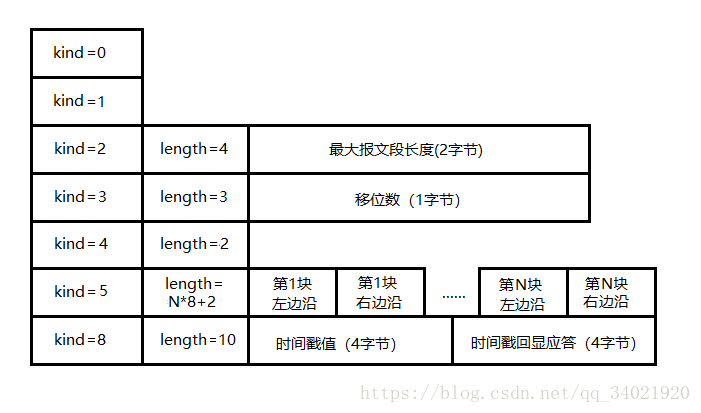
检测tcp socket并检测连接是否在运行或者是否被破坏

当建立一个TCP连接时，将一系列的定时器与该连接相关联。这些定时器中某些用于处理keepalive过程。当keepalive定时器变为0时，会给对方发送一个keepalive探针包，包内没有数据，并且ACK标识打开，另一方面，会收到一个来自远方主机的回应该回应同样没有数据，并且设置ACK标识

阻止因为网络连接不活跃（长时间没有数据包）而导致的连接中断

1. 列举所知道的TCP选项，并说明作用

常用选项有7种，如下图所示：



* Kind = 0， 选项表结束（EOP）选项
  + 一个报文段仅使用一次，放在末尾用于填充
  + 用途：说明首部已经没有更多的信息，应用数据在下一个32位开始处
* Kind = 1， 空操作（NOP）选项
  + 没有特殊含义，一般用于将TCP选项的总长度填充位4字节的整数倍
* Kind = 2， 最大报文段长度（MSS）选项
  + TCP连接初始化时，通信双方使用该选项来协商最法报文段长度。
* Kind = 3， 窗口扩大因子选项
  + TCP连接初始化时，通信双方使用该选项接收窗口的扩大因子
    - 在TCP头部中，接收窗口大小是16位表示的，故最大为65535字节，但实际上TCP模块允许的接收窗口大小远不止这个数，窗口扩大因子解决了这个问题
    - 假设TCP头部中的通告窗口大小为N，窗口扩大因子（位移数）是M，那么TCP报文段的实际接收通告窗口大小为（N\*2^M），或者说n左移M位，M的取值为0～14
* Kind = 4，选择性确认（SACK）选项
  + TCP通信时，如果某个TCP报文段丢失，则TCP会重传最后被确认的TCP报文段后续的所有报文段，这样原本已经正确传输的TCP报文以可能重新发送，从而降低了TCP性能。SACK技术正是为改善这种情况而产生的，它使TCP只重新发送丢失的TCP报文段，选择性确认选项用在连接初始化时，标识是否支持
* Kind = 5， SACK实际工作选项
  + 该选项的参数告诉发送方，本端已经接收到并缓存的不连续的数据块，从而让发送端根据此来检查并重发丢失的报文段
* Kind = 8, 设置时间戳

1. Socket什么情况下可读/可写

可读：

1. socket接收缓冲区中已经接收的字节数大于等于socket接收缓冲区低潮限度的当前值；对这样的socket的读操作不会阻塞，并返回一个大于0的值。我们可以用socket选项SO\_RCVLOWAT来设置低潮限度
2. 连续的读这一半关闭（也就是接收了FIN的TCP连接）。对这样的套接字读操作，将不会阻塞，而是返回0。
3. 该套接字时一个listen的监听套接字，并且目前已经完成的连接数不为0
4. 有一个错误套接字待处理

可写：

1. 该套接字发送缓存中的可用空间字节数大于等于套接字发送缓存取低水位标记
2. 该连接的写半部关闭（主动发送FIN包的TCP连接）。对这样的套接字写操作将会产生SINPIPE信号。
3. 使用非阻塞的connect套接字已经建立，或者connect已经以失败告终。
4. 有一个错误的套接字待处理
5. IO多路复用模型

IO多路复用主要的应用是select和epoll（poll可以进行了解，几乎不用）

**Select：**

1. 只有一个函数，对作为传入参数的三个不同的文件描述符队列进行监控（读、写、异常）。

2. 返回值是产生事件的文件描述符个数，sellect返回整个数组，**需要进行轮询判断。**

3. sellect底层与内核进行交互是通过**复制粘贴**的方式。

4. select有最大文件描述符限制（这个限制可以通过unlimit -n命令进行修改）

5. selrct只支持LT模式（水平触发）

**Epoll：**

1. epoll有三个函数，底层是一棵红黑树，建立监控需要向红黑树挂载epoll\_event节点，里面至少包含文件描述符和监控类型（可以了解一下epoll反应堆，这个是通过跟fd同处于一个共用体的void \*指针所实现的，它指向一个自己实现的结构体，这个结构体中至少包含fd和回调函数的函数指针）

2. 返回值是产生事件的文件描述符的个数，返回的数组，是产生事件的节点组成的数组，**不需要进行轮询。**

3. 底层与内核的交互是通过一块**共享内存**

4. epoll无最大文件描述符限制

5. epoll支持更加高效的ET模式（边缘触发）

**epoll和sellect的区别：**

1. Select只需要一个一个系统调用select，epoll需要三个epoll\_create, epoll\_ctl, epoll\_wait。
2. Select有最大连接数限制1024，epoll没有这个限制，select的限制可以通过unlimit -n 数字 修改
3. Select文件描述符在内核和用户态之间转换的时候通过复制的形式，时间开销大，epoll使用共享内存减少复制的时间
4. Select通过传递文件描述符进行监控，查询的时候是通过轮询的方式，通过FD\_ISSET查看，epoll底层是一棵红黑树，只有发生事件的fd才会调用callback函数返回，不需要轮询在活跃较少的时候不会有性能问题
5. Select可以跨平台，epoll不可以
6. 但是并不是说epoll一定比select要好，在很大访问量的情况下，epoll的性能可能并不比select好，因为epoll的通知机制会有很多回调函数

**ET模式和LT模式的区别：**

LT支持阻塞和非阻塞，**ET只支持非阻塞**，所以在使用ET模式的时候，红黑树上挂载的所有文件描述符都需要使用非阻塞模式

触发事件后，读取缓冲区中的内容：LT模式，如果没有读完，会再次触发事件；ET模式，如果没有读完，不会触发事件

1. SYN Flood攻击

三次握手是有缺陷的，服务器端在接收到第一次握手报文时，会首先建立一个**半连接状态**，将维持该状态的信息放在一个**未连接队列**中，在第二次握手中，当服务器发送了SYN+ACK报文后，若该连接的定时器超过超时重传时间服务器会对报文进行重传，当重传次数超过一定次数后，服务器认为TCP连接超时，而后才销毁半连接状态，释放TCP。如果攻击者用很快的速度，连续对目标服务器开放的端口发送SYN报文（这些报文的IP是相同的），服务器的资源将被消耗，导致服务器卡顿，以至于不能接收其他客户的正常连接请求。

攻击检测以及防御手段：

目前对于SYN Flood攻击一般通过实时检测tcp的新建连接速率来判断。

如果短时间内接收到同一IP发送的大量第一次握手连接请求，要这个时候要对该IP地址的流量进行**限流**。正常客户端有可能也会出现SYN flood攻击的表现，判断是否为正常用户的方法是，查看能否收到该IP的第三次握手报文，如果是攻击性的，不会回复第三次报文，如果是正常用户，会回复第三次报文。

防御的前提是区分出哪些请求是合法的，哪些是SYN泛洪，下面是实现SYN cookies的三种不同方式。

方法一

1. 当安全设备获取到发往服务器的SYN报文后，设备模拟服务器端会给此源IP回复ACK=Cookie的SYN/ACK报文
2. 正常客户端会恢复RST报文，攻击流不会回复
3. 安全设备收到RST报文并校验后可把正确报文的客户端IP加入到白名单
4. 后续安全设备放行属于该IP的正常流量

方法二

1. 当安全设备获取到发往服务器的SYN报文后，设备模拟服务器端会给此源IP回复ACK=Cookie的SYN/ACK报文
2. 正常客户端回复ACK报文，确认序号为Cookie+1
3. 安全设备收到ACK报文，并校验无误后，将该客户端的IP加入到白名单，并向客户端发送RST报文断开连接
4. 后续安全设备放行属于该IP的正常流量

方法三：

1. 当安全设备获取到发往服务器的SYN报文后，设备模拟服务器给源IP回复seq=Cookie的SYN/ACK报文
2. 正常客户端回复ACK报文，确认序号为Cookie+1
3. 安全设备收到ACK报文并校验无误后，设备与服务发送连接请求，通过三次握手后与服务器建立起连接
4. 客户端与防火墙之间建立了连接，防火墙与服务器之间也建立了连接，客户端与服务器间关于此次连接的后续数据报文都通过安全设备代理转发
5. TCP如何设定超时时间
6. 通过设定socket非阻塞，在select或者epoll等设置超时时间
7. 使用alarm信号量设置超时时间，不过使用这个方法，线程信号量掩码是线程私有的，当指定进程递交信号量时，操作系统会将信号量递交给进程中未屏蔽信号量的所有线程的随机之一
8. 基于socket网络编程和tcp/ip协议栈，讲讲从客户端send()开始，到服务端recv()结束的过程

客户端send()说明已经建立好了连接，并已经设置好了拥塞窗口，send()发送的数据，会先存储在tcp的发送缓存中，发送端为每个字符标号，每次取出发送窗口大小的字符按照顺序从发送窗口发送，建立tcp数据包的报头，后面添加报文数据，作为ip数据报的数据段，封装入ip数据报后，发送到路由器，路由器从路由表中查看目标服务器ip，找到记录的下一个路由器，并发送。如果ip与当前路由器相连，发送到服务器端。

服务器端接收到数据后，ip数据包检验，检验正确后将数据段也就是tcp数据包提交到运输层，tcp协议进行CRC循环冗余检验，检验无误后提交到应用层，recv()根据所指定的套接字从接收缓存中读取相应数据。

1. **http协议（端口号 80）**
   1. HTTP是客户端与服务器端之间传输数据的一种格式规范，格式简称为“超文本传输协议”
   2. http协议与TCP/IP协议的关系
      1. HTTP的长连接和短连接本质上是TCP的长连接和短连接
      2. HTTP属于应用层协议，在传输层使用的是TCP协议，在网络层使用的是IP协议
   3. HTTP协议的特征
      1. 支持客户/服务器模式
      2. 简单快速
      3. 灵活
      4. 无连接
      5. 无状态
   4. 如何理解http是无状态的
      1. HTTP协议是无状态的指的是对事物处理没有记忆能力，服务器不知道客户端是什么状态的，也就是说，打开一个服务器上的网页和上一次打开这个服务器上的网页之间没有任何联系。但是无状态不代表http不能保持tcp连接，更不代表http使用的是udp
   5. HTTP首部协议
      1. 通用首部字段（请求报文与响应报文都会使用的首部字段）
         1. Date：创建报文事件
         2. Connection：连接的管理
         3. Catche-Control：缓存的控制
         4. Transfer-encoding:报文主题的传输编码方式
      2. 请求首部字段（请求报文会使用的首部字段）
      3. 响应首部字段（响应报文会使用的首部字段）
      4. 实体首部字段（请求报文与响应报文的实体部分所使用的首部字段）
   6. HTTP状态码
      1. 1xx 信息提示
         1. 100：告诉客户端，应该继续发送请求
      2. 2xx成功
         1. 200：成功接收了请求，并将返回客户端所请求的结果
         2. 202：成功接收了请求，但是还没有进行处理，距离会不会处理，不确定
         3. 204：服务器成功处理了请求，但没有返回任何实体内容，可能返回新的头部信息
      3. 3xx 重定向
         1. 301：客户端请求的网页已经永久的移动到新的位置
      4. 4xx 客户端错误
         1. 404：请求失败
      5. 5xx 服务器错误
         1. 500：服务器遇到未知错误
         2. 503：服务器处于临时过载或维护
   7. http报文格式
      1. 请求报文
         1. 请求报文由请求行、请求头、空行和请求内容四个部分组成
         2. 请求行：请求行中包含，请求方法字段、URL字段、协议版本字段三部分组成，之间由空格分隔开
         3. 请求头：由不定数量的键值对组成，每行为一个键值对，key和value之间通过冒号分隔。作用：用于通知服务器有关客户端的请求信息
         4. 空行：用来告诉服务器，接下来就是请求的内容了
         5. 请求内容：涵盖请求
      2. 响应报文
         1. 由状态行、响应头、空行和响应内容4部分构成
         2. 状态行：HTTP协议版本、状态码、状态码描述三部分组成
         3. 响应头：
            1. Location: 服务器返回给客户端，用于重定向到新的位置
            2. Server: 包含服务器用来建立请求的软件信息以及版本信息
            3. Vary: 标识不可缓存的请求头列表
            4. Connection: 连接方式
            5. Close: 对于请求端，告诉服务器，端开连接，不用等待后续的请求；对于服务器，表示连接已经关闭
            6. Keep-alive: 对于请求端，告诉服务器，在完成本次响应请求后，保持连接，等待本次连接的后续请求；对于响应端，表示连接保持中
      3. 空行：告诉请求端，接下来的内容是响应内容
      4. 相应内容，服务端返回给请求端的文本信息
   8. HTTP协议中的请求方式

POST和GET的区别

GET从指定的资源获取数据

POST向指定的资源提交要被处理的数据

1. 后退/刷新：GET无害， POST数据会被重新提交
2. 书签：GET可以被收藏为书签，POST不可以
3. 缓存：GET可被缓存，POST不可缓存
4. 编码类型：GET只支持url编码，POST支持多种编码类型
5. 历史：GET的参数保留在浏览器历史中，POST不会保留
6. 对数据长度的限制：因为URL长度是有限制的，所以GET也受到限制（2048个字符），POST无限制
7. 对数据类型的限制：GET只允许ASCII字符，POST没有限制
8. 安全性：POST比GET安全性更好
9. 可见性：GET的数据在URL中，对所有人都可见，POST数据不会出现在URL中
10. 发送：GET发送一个包，POST发送两个包（先发送header，服务器响应100，再发送data)

POST和PUT的区别

1. PUST不是幂等操作，PUT是幂等操作
2. PUST用来更新或者创建资源，PUT只是更新资源
3. 如果提交多次结果是一致的，比如修改密码，用PUT，如果提交多次，结果不一致，比如创建用户，这个时候就用POST
   1. HTTP长连接（保持长连接通过心跳检测（keepalive）机制
      1. 定义：客户端与服务器建立连接后不断开，然后在进行报文发送或接收。这种常用于P2P通信
   2. HTTP短连接
      1. Client方与Server方每进行一次报文收发交易时才进行通讯连接。此方式常用于一对多点通信
   3. http协议实现的原理机制



1. https（端口号 443）
2. http是明文传输，在需要安全的场合下不够安全。
3. https利用http进行通信，但是需要用到SSL/TLS加密数据包

作用：

1. 建立信息安全通道，保证数据的安全（通过加密）
2. 确认网站的真实性（通过证书）

利用SSL/TLS协议进行加密传输之前，客户端和服务器之间有一个握手的过程。握手的目的是为了协商出一个共同使用的密钥。

加密分为对称加密和非对称加密

对称加密：加密解密使用同一个密钥

非对称加密：加密有公钥和私钥两个密钥，用公钥加密，私钥才能解密，反之亦然。非对称加密比对称加密更好，但花费时间长、速度慢，只时和对少量数据进行加密。

握手过程是非对称加密：

1. 客户端向服务器传送客户端SSL协议版本号，加密算法的种类，产生的随机数，以及各种其他信息
2. 服务器向客户端传送SSL协议的版本号，加密算法的种类，随机数以及其他相关信息，同时服务器还向客户端发送自己的证书
3. 利用服务器传送过来的信息验证服务器的合法性，包括：证书是否过期，发行证书的CA是否可靠，证书是否有效，证书上的域名和实际域名是否向匹配
4. 客户端随机产生一个新的随机数，然后用公钥进行加密，归还给服务器，同时利用随机数经过运算，得到密钥
5. 服务器利用私钥解密，得到随机数，得到与客户端相同的密钥
6. 握手结束后，服务器与客户端通过这把密钥进行对称加密传输
7. http与https的区别
8. https需要CA的证书，一般免费的证书较少
9. http是明文传输的，https在http的基础上加入了ssl加密传输协议
10. http和https使用不同的连接方式，用的端口也不一样，http是80，https是443
11. http连接很简单，是无状态的；https协议是由SSL+http构建的可进行加密传输，身份认证的网络协议，更加安全
12. **http2.0**

同时也要关注一下http1.x（这个我没关注，也没碰到面试官问这个问题，只能说我很幸运，不代表不会被问到）

1. http2.0将报文分成了headers帧和data帧，它们都是二进制格式的
2. 增添服务器推送
3. 首部压缩

客户端和服务器同时维护和更新一个包含之间前过的首部字段表，从而避免了重复传输，不仅如此，http2.0调整了编码方式，对首部进行了压缩

1. push怎么实现
2. 服务器生成应用服务器密钥，作用是表示服务器
3. 客户端请求用户授权消息推送，一旦授权，浏览器会发送一个pushScription去到服务器，用于后面是消息传递
4. 服务器发送push标准的api，由触发推送服务器的消息推送
5. 客户端接收消息推送

21. url地址各部分代表什么

协议://地址（域名地址/IP地址）:端口 /路径/…

22. 数据链路层协议

PPP协议

CSMA/CD协议

23. 网络层协议

IP协议

ICMP协议

NAT协议

24. 关注一下各个数据报的报头

例如

1. TCP的端口，端口号是在运输层加入的报文

2. TCP的伪首部内容（主要内容为双方的IP地址）、作用

3. IP的校验位只检验了报文段

4. 数据链路层进行循环冗余校验

还有一些，可以看一下各个报文的首部，有个印象，不要面试官问一些细节性问题，例如ip地址什么时候加入的报文、端口什么时候加入的报文等等，不要慌