1.主要的四种IO模型

首先是因为应用进程在进行网络的读写的时候，无论是发送方还是接收方，都会维护一个缓冲区，主要是为了减少频繁的IO的系统调用。

- 同步堵塞IO： 指的是需要内核IO操作彻底完成之后，才放回到用户空间，执行用户的操作，堵塞是指用户必须堵塞到知道内核io操作彻底完成之后
- 同步非堵塞IO：指的是不需要等待内核IO操作完成后，内核立即给用户返回一个状态值，用户空间无需等到内核的IO操作彻底完成之后，可以立即放回用户空间
- IO多路复用：通过一个线程来记录每个IO流的状态，当有数据到达时，socket被激活然后select函数返回，这个时候用户线程进行真正的读写
- 异步IO：将用户空间和内核空间的调用方式反过来，用户空间线程变成是被动接受的，内核系统才是主动调用者，就是进行回调

2.select，poll，epoll

- 使用select需要传入一个fd_set，fd_set是一个long类型的数组，每一个数组元素都能与一打开的文件句柄建立联系，然后当有就绪的描述符(一个非负整数，是一个索引值，当程序打开一个文件或者创建文件的时候返回一个描述符)的时候，需要进行遍历fd_set这个集合，也就是这需要花费一个o(n)的轮询时间复杂度。同时select还对fd_set的大小做了限制
- poll在select上做了改进，但是原理是一样的，poll只解决了fd_set的大小问题
- epoll是一个增进版本，改进了select下需要轮询这个文件描述符集合的问题，是基于信号驱动的模式的，它不需要遍历整个fd_set这个集合，而是遍历那些被IO事件异步唤醒加入到Readly队列的描述符集合就可以，所以说是大幅度提升了性能。

epoll有两种触发方式：

1. 水平触发：当检测出有某些事件描述符就绪就通知应用程序，应用程序可以不立即处理该事件，而是下次在通知这个事件
2. 边缘触发：当检测到某描述符事件就绪并通知应用程序时，应用程序必须立即处理该事件，如果不处理，下一次就不会通知该事件

3. TCP中的粘包和拆包

TCP发送报文使用的是滑动窗口，发送端和接收端都会维护一个缓冲buffer，读写数据都从buffer中取，因为TCP是面向字节流的，面向流，那么应用程序就不知道每一个包的终点在哪里

粘包：应用程序读取到了多个包，比如发送了ABC和DEF，而接收端收到的缺失ABCD

拆包：应用程序读到了不完整的一个包，比如发送了ABC，而接收端收到的是AB

解决办法：

- 发送方和接收方规定固定大小的缓冲区，也就是发送方和接收方使用相同的缓冲区
- 在TCP报文的数据头中存储数据正文的大小，这样接收方就知道边界在哪里
- 特殊字符结尾，通过特殊字符结尾就可以知道流的边界

4. HTTP与HTTPS的区别

HTTP的特点：

- 无状态：协议本身是没有状态存储功能的
- 无连接
- 基于请求和响应
- 简单、快速、灵活
- 通信使用的是明文，请求和响应不会和通信方确认、无法保证数据的完整性

HTTPS的特点：

- 内容加密：采用混合加密的方式，中间者无法直接查看传输的内容
- 验证身份：通过证书认证客户端访问的是自己的服务器
- 保护数据的完整性：防止传输的内容被中间人冒充

混合加密的方式：对称加密是指加密和解密都会使用同一把密钥，而非对称加密是指加密和解密使用不同的密钥，而对称加密只要拿到了密钥，加密就失去了意义，而非对称加密则需要进行加密和解密效率比较低，比较耗费资源，所以SSL采取了一种折中的方式，在创建连接的时候使用非对称加密传输对称加密的密钥，这样就可以在连接建立之后进行对称加密传输，提高效率

SSL的传输过程：

1. 客户端向服务器发送请求，请求服务器端的443端口，同时发送一个随机值1和一个支持的加密算法
2. 服务端收到信息之后，给与一个响应报文，该报文包括一个随机值2和一个具体使用的加密算法，该加密算法是上一次请求的加密算法的子集
3. 随后服务端向客户端发送第二个响应报文，将数字证书发送给客户端
4. 客户端解析证书，有客户端的TLS进行解析工作，主要是验证公钥是否有效等等，如果没有问题则再次生成一个随机值，叫做预主密钥
5. 接着客户端将随机值1、随机值2、预主密钥组成会话密钥，然后通过证书的公钥进行加密
6. 服务端收到后，使用证书的密钥进行解密会话密钥，同样通过随机值1和随机值2、预主密钥组成会话密钥
7. 接着客户端通过会话密钥随机发送一条消息给服务端，验证服务端是否可以正常接收客户端的消息
8. 同样服务端也通过会话密钥发送一条消息给客户端，如果可以正常接受的话，那么SSL连接就建立好了

5. HTTP的常见状态码

大类分析：

- 1xx：表示服务器收到请求，需要请求者继续执行操作
- 2xx：成功，表示操作被成功处理
- 3xx：重定向，需要进一步的操作以完成请求
- 4xx：客户端错误：请求包含错误或无法完成请求
- 5xx：服务器错误

细分：

- 100：表示客户端应该继续请求
- 200：OK
- 204：请求成功，但是无内容
- 301：永久重定向
- 302：临时重定向
- 304：缓存重定向
- 400：表示请求路径错误或者语法错误

- 401：未经许可
- 403：拒绝访问，访问权限出现问题
- 404：资源没有找到
- 500：服务器在执行的过程中出现错误
- 503：服务器处于超负载或者没有启动

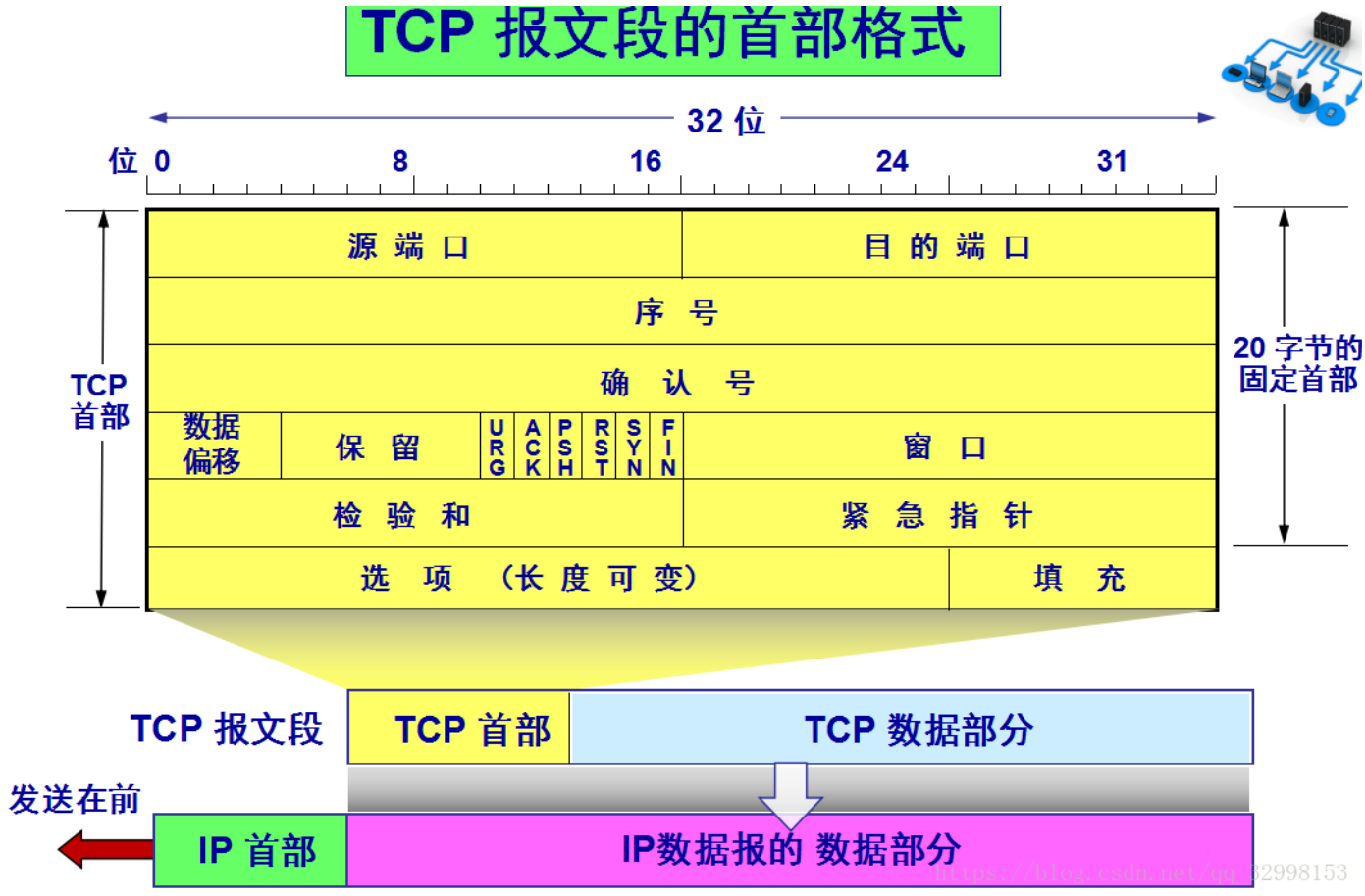
6. 转发和重定向

- 重定向：客户端行为，2次请求，会显示新的地址，请求域中的数据会丢失
- 转发：服务器行为，1次请求，地址栏不会变化，请求域中的数据不会丢失

7. TCP和UDP的首部格式、大小

TCP的首部格式：

- 原端口和目的端口：各占2个字节，也就是端口号大小在0 - 2^16之内 4
- 序号：在建立连接的时候由计算机生成的随机值作为其初始值，用来解决乱序问题，很多机制都需要它，窗口滑动、拥赛控制等等 4字节
- 确认应答号：指下一次期望收到的数据的序列号，用来解决不丢包的问题
- 数据偏移 保留位 6个控制位(ACK,SYN,FIN)： 2个字节 2
- 窗口：2个字节 2
- 检验和：2个字节 2
- 紧急指针：2个字节 2
- 选项和填充这是长度可变的但是一定是4字节的n倍



所以TCP的首部报文是20个字节 + 4n个不确定长度

UDP的首部格式：

- 源端口和目的端口：各占2字节
- 长度和校验和：各占2字节

所以UDP的首部报文时8个字节

9.TCP/IP五层协议

- 应用层：只需要关注为用户提供应用功能，而不用去关注具体是如何传输的，应用层是工作在用户态的，而其他4层是工作在内核态的
- 传输层：传输层是为应用层提供网络支持的，传输层有两个协议TCP和UDP，TCP提供可靠的传输，多了很多的特性，例如拥赛控制、流量控制、超时重传机制等，同时通过端口来标识每一个进程
- 网络层：处理实际的传输过程，网络层最常使用的是IP协议，IP协议会将传输层的保文作为数据部分，加上IP的头部信息，传输给下一层
- 数据链路层：用来标识网络中的设备，让数据在一个链路中传输，主要是为网络层提供链路级别传输的服务
- 物理层：将数据链路层的报文转为二进制进行传输

应用层、表示层、会话层、传输层、网络层、数据链路层、物理层

10.HTTP缓存问题

HTTP缓存问题时HTTP性能优化的常见手段，HTTP缓存一般有两种类型：

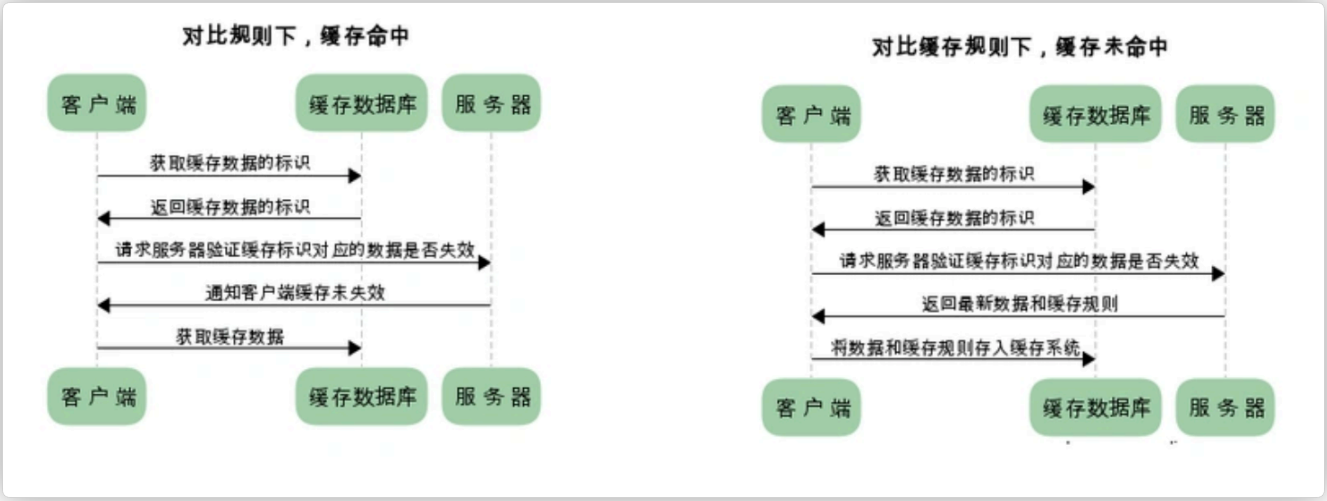
强制缓存

强制缓存是指在缓存数据未失效的情况下，可以直接使用缓存数据，而不需要在次请求服务器。强制缓存一般通过两个字段来标明失效规则：Expires和Cache-Control：

- Expires：Expires是HTTP1.0时候使用的，它的值为服务器返回的到期时间，如果客户端时间和服务器的时间不一致则会产生误差
- Cache-Control：在HTTP1.0，Cache-Control已经将Expires1取代，常见的取值有private、public表示缓存的权限，max-age：表示可以被缓存多长时间

对比缓存

就是需要进行判断是否可以使用缓存，就是说无论缓存是否命中，都需要向服务器发送请求，服务器判断成功之后返回304，通知客户端比较成功



对比缓存会使用两个字段来声明规则：

- Last-Modified/if-Modified-Since：当服务器第一次响应可缓存的请求的时候会返回一个Last-Modified字段，然后当客户端再次请求的时候会带上if-Modified-Since字段这个值就是上次的Last-Modified值，然后服务端会对比当前对资源的最后修改时间，如果资源没有被修改则可以使用缓存，返回304状态码
- Etag/If-None-Match（优先级要高）：当服务器响应请求的时候，通过Etag告诉客户端当前资源在服务器的唯一标识，当浏览器再次请求的时候就会带上一个If-None-Match字段，这个值就是上一次Etag的值服务器对比两个值是否相等，如果一致则可以使用缓存，返回304状态码

而使用这两种缓存策略是不同的刷新手段，例如在url输入栏中输入一般会使用强制缓存

11.HTTP常见的字段

- Host字段：客户端发送请求时用来指定服务区的域名，有了host字段就可以发往同一台服务器上的不同网站，一些虚拟主机技术
- Content-Length字段：表明本次回应的数据长度
- Connection字段：表明客户端要求服务器使用TCP持久连接，keep-alive这也是TCP的长连接要求的字段
- Content-type：告诉接收方，本次的数据格式，test/html
- User-Agent：用户代理，让服务器能够识别客户端的操作系统、CPU类型等信息

12.HTTP1.1优化手段

HTTP1.1引入了长连接，使用keep-alive字段，那么首先就可以将keepalive字段设置为长连接，可以从以下几个方向来优化HTTP1.1

- 尽量避免发送HTTP请求-使用缓存：对于一些重复性的HTTP请求，比如每次请求到的数据都是一样的，那么就可以将这个请求-响应缓存在本地，那么下次就可以直接读取本地的数据了，缓存有强制缓存和对比缓存
- 减少HTTP请求次数：减少重定向请求的次数，因为如果重定向的次数越多，那么客户端就需要发起多次HTTP请求；合并请求，可以吧多个小文件的请求合并成一个大文件的请求，传输的总资源是一样的，但是减少了请求的次数，例如对于图片的请求可以使用CSS Image Sprites合并成一个图片；延迟发送请求，对于不需要的资源按需获取
- 压缩响应资源，可以有无损压缩和有损压缩，无损压缩例如gzip格式等等，有损压缩一半使用于音视频

13.HTTP2.0优点

HTTP1.1虽然有上面那些优化手段，但是还是有以下几个问题：

- 延迟难以下降：虽然现在网络的带宽已经提升很多但是延迟下降已经到了瓶颈阶段
- 并发连接有限：最大连接并发数比较小
- 队头堵塞问题：同一个链接只能在处理完一个http事务之后，才能处理下一个事物，这样就容易造成堵塞
- HTTP头部重复问题：HTTP是无状态的，每一个请求可能都会携带cookie，cookie是很大的
- 不支持服务器主动推送消息：HTTP1.1都是服务器被动的接受来自客户端的消息

HTTP2.0则对上面这些问题做了优化：

- 头部压缩：HTTP2开发了HPACK算法，让客户端和服务端都会建立和维护一个字典，然后用哈夫曼编码进行压缩，为高频出现的头部字段建立一张静态表
- 并发传输：HTTP2支持并发传输，多个Stream复用一条TCP链接，也就是多路复用
- 二进制传输：将HTTP1.1的文本格式转为二进制格式传输数据，极大的提高了传输效率
- 服务器主动推送资源：例如一个html页面和css文件的请求，可以让服务器主动推送css文件，这样http请求就少了一个消息传输

当然，由于TCP的稳定传输问题，HTTP2还是存在队头堵塞问题，在HTTP3开始已经开始弃用TCP协议，使用UDP协议并且开发了新的QUIC运行在用户态的协议，其实就是保证了TCP的可靠传输

14.keep-alive

HTTP协议是请求-应答模式，当不是长连接的时候，每一个请求应答客户端和服务端都需要新建一个连接，当使用keep-alive模式的时候，是一个长连接的模式。在HTTP1.1时，已经是默认开启了长连接的模式，使用长连接能避免连接建立和释放的开销，但是长时间的TCP连接容易导致系统资源被无效占用

当保持长连接的时候可以通过Content-Length 和Transfer-Encoding来判断一次请求是否完成，Content-Length表示的是实体的长度，如果是静态资源则知道请求的长度，如果是动态资源，那么就需要使用Transfer-Encoding，Transfer-Encoding表示传输的编码，如果该值是chunked则表示是一段一段传输的，最后一个chunked为0则表示本次传输结束

一般的httpd守护进程都会设置一个keep-alive timeout参数等待时间，连接建立之后如果一直不发数据，过一段时间后TCP协议会自动发送一个数据为空的探测报文给对方，如果对方没有返回，则可以认为没有必要保持连接

15.Get和Post

Get是获取网络中的资源，Post是提交一般用于提交表单。Get将参数拼接到url上，而post将参数写在请求体里面，所以一般来说post更安全。但是本质上来说Get和Post都是使用的TCP/Ip协议，给Get加上请求体，给Post加上url 参数也是可以的。也就是说其实Get和Post都是TCP连接，只是在应用过程中有有一些不一样啊。还有个很重要的区别是Get方式发送一个TCP数据包，POST方式发送两个数据包，但这只是网上的说法，我在真正抓包实践的时候并没有抓到过。对于Post会先发header，等服务器返回100，然后接着发body，服务器响应200

16.为什么要有TCP层

因为IP层是不可靠的，不保证网络包的交付、不保证网络包的按需交付、不保证网络中包的数据完整性，因此需要保证网络连接的可靠性，就需要上层的TCP协议来负责

17.TCP、UDP

TCP特点：

- 面向连接：一定是一对一才能连接，而UDP可以一个主机同时向多个主机发送消息
- 可靠的：TCP的各种机制窗口、拥赛控制、确认机制可以保证一个报文一定可以到达接受端
- 字节流：TCP协议是以字节流的方式进行传输的

UDP特点：

- UDP不需要连接，直接交付数据
- UDP支持一对一、一对多、多对多的交互通信
- UDP尽最大努力交付，不保证数据的可靠性
- UDP的首部开销小只有8个字节，而TCP的首部开销大，一般是20字节+4n字节
- UDP一般用于DNS，视频、音频等多媒体通信

18.TCP三次握手

- 一开始，客户端和服务端都处于CLOSED状态，首先是服务端主动监听某个端口，处于LISTEN状态
- 客户端会随机初始化一个序列号，同时将SYN=1，将报文发给服务端，之后客户端处于SYN-SENT状态
- 服务端收到客户端的SYN报文之后，也会随机初始化自己的序列号，同时将SYN=1，ACK=1，确认应答号为收到的客户端的序列号+1，之后服务端处于SYN-RECV状态
- 客户端收到服务端报文之后，将确认应答号设置为收到的服务端的序列号+1，序号设置为上一个发送报文的序号+1，之后客户端处于ESTABLISHED状态，同时这个阶段是可以携带数据的
- 服务端收到后也处于ESTABLISHED

为什么是三次，而不是两次：

- 三次握手可以阻止重复历史连接的初始化，在复杂的网络环境下，有可能旧的SYN报文比新得SYN报文更先到达，如果只是两次连接的话，则无法判断这是旧的链接，三次连接时第三次报文，客户端可以通过服务端发送的确认报文中取出确认应答号通过上下文来判断这个序号是否是过时的链接
- 三次握手可以同步双方的初始序列号，对于TCP的可靠传输来说，序列号起这非常重要的作用，如果是两次的话，客户端没有办法同步服务端的序列号，通过三次握手，双方可以同步各自的序列号
- 防止资源浪费，如果只有两次握手，当服务端发送ACK确认报文的时候，这个报文堵塞了，那么在客户端就会重新发送SYN报文，如果只有两次握手，那么每收到一个客户端的SYN报文，就要建立一个链接，但是不清楚客户端是否收到了自己发送的ACK确认，就会导致建立冗余的链接，造成资源的浪费

为什么是随机序列号：

- 如果一个失效的连接被重用了，也就是说旧的请求报文到达了服务端，如果序号一样，则无法判断这个报文是不是历史报文，就会产出数据错乱，而建立随机序列则客户端可以在第三次握手的时候判断是不是旧的链接报文

序列化随机算法：

- MF算法，M是一个计时器，这个计时器咩隔4毫秒加1，F是一个hash算法，根据源IP、目的IP、源端口、目的端口生成一个随机值，可以使用md5算法

第一次SYN丢失怎么办

怎么模拟：tcpdump + wireshark 禁用服务器的网线

当在第一次握手的时候出现了SYN丢失，那么客户端会进行超时重重传，在linux下有一个tcp_syn_retries内核参数，在我的centos7机器上这个值是6，也就是重传6次，并且RTO也就是超时重传时间也是翻倍增长的

半连接队列与全联接队列

在TCP三次握手的时候，LInux内核会维护两个队列，半连接队列也叫SYN对列和全连接队列也叫accept队列

当服务端收到客户端发起的syn请求之后，内核会把该链接存储到半链接队列里，并向客户端响应syn + ack确认信息，当服务端收到第三次握手的ack之后，内核会把链接从半链接队列里移除掉，然后创建新的完全的链接，并将其添加到accept队列，等待进程调用accept函数时取出来

19.SYN泛洪攻击

TCP连接需要三次握手，如果攻击者段时间内伪造不同的IP地址的SYN报文，那么服务端每收到一个SYN-RECV状态，但是无法收到客户端的ACK应答，久而久之就会占满服务端的SYN接受队列，使得服务器不能为正常的用户提供服务

一个正常的SYN队列与Accpet队列应该是如下工作的：

- 当服务端接受到客户端的SYN报文的时候，会将其加入内核的SYN队列里
- 接着发送SYN+ACK给客户端，等待客户端回应ACK报文
- 当服务器接受到ACK报文的时候，从SYN队列中移除放入Accept队列
- 最后内核调用accept()socket接口，从Accpet队列中取出连接

解决办法：

- 修改Linux内核参数，控制队列大小和当队列满的时候应该做什么策略，比如说队列满的时候，对新的SYN报文可以直接回RST丢弃该连接
- SYN cookies的方式，当SYN队列满之后，后续服务器收到的SYN包不再进入SYN队列里面，而是计算出一个cookie值，再以SYN+ACK中的序列号返回客户端，当服务端收到客户端的应答报文的时候，服务器会通过cookie值检查这个ACK包的合法性，如果合法再放入到Accept队列里面，最后系统调用accept接口，从accpet队列中取出连接

20. TCP四次挥手？状态、2MSL

TCP是全双工通信，也就是说可以由任意一方断开连接，假设是客户端发起，那么客户端发起一个FIN请求报文表示要断开连接了，此时客户端进入到FIN-WAIT1状态，服务端收到FIN请求后，如果确认断开需要发送一个ACK确认报文，当客户端收到后此时客户端进入FIN-WAIT2状态，而服务器端进入CLOSE-WAIT状态

接着如果服务器端也要与客户端断开连接，那么也要发送一个FIN请求报文，当客户端收到后进入到TIME-WAIT状态，而服务器端进入到LAST-ACK状态，这个时候客户端需要等待2MSL的时候，同时发送确认关闭报文给服务端

下面解释为什么要Time-Wait状态：

- 为了保证客户端发送的最后一个ACK报文能够到达服务端，保证连接正确关闭
- 等待2MSL时间，可以让本连接事件内网络上的所有报文段都消失

同时需要注意的是Time-Wait状态只会出现在主动关闭连接的一方

下面解释为什么是2：

首先，MSL是指最长的报文寿命，这是一个工程值，根据情况来进行选取的。

在服务器端与客户端断开连接的时候，要发送一个确认报文，但是因为要断开连接，所以客户端不知道服务端到底有没有收到我客户端发送的ACK确认报文，那么此时可以有两种情况：①服务器没有收到，那么服务器会超时重传FIN报文 ②服务器收到，那么2MSL就是取这两种情况的最大值，因为发送确认报文需要一次MSL，而等待服务器发送一个重传报文也需要一个MSL报文，所以是2MSL

21.TIME-WAIT过多以及优化思路

如果服务器上出现time-wait，说明是由服务器自己断开的链接，在高并发短连接的服务器上，当服务器处理完请求后立即关闭连接，在这个场景下会大量的socket处于time-wait状态，因为是处于短链接，是指业务处理+数据传输的时间远远小于time-wait超时的时间

如果time-wait过多首先危害时服务端的内存资源占用，同时也是对客户端端口的占用，因为在time-wait这段时间内时不能处理请求的

优化time-wait的方式为：

- 让time-wait的socket为新的链接所用，可以开启内核参数 net.ipv4.tcp_tw_reuse和tcp_timestamps
- 可以将系统中的time-wait链接一旦超过这个值时，将后面的time-wait重置，这样就直接禁止了，一般不使用这种方式
- 通过设置socket选项，让TCP链接直接跳过time-wait状态，直接关闭

22.TCP的保活机制

在TCP的链接中，可以定义一个时间段，如果在这个时间段内，没有任何链接相关的活动，TCP保活机制会开始作用，每隔一段时间发送一个探测报文，如果连续几个探测报文没有得到响应，则会认为当前的TCP链接已经死亡

23.TCP的可靠保证

TCP是可靠的传输协议，TCP是通过序列号、确认应答、重发机制、连接管理以及窗口控制这些机制来实现可靠性的

重传机制

在TCP中，如果发送方的数据到达接收端的时候，接收端会返回一个确认应答消息，表示已经收到消息，当发送方没有收到这个ack应答消息的时候，即认为需要重传该数据包

在TCP中重传机制一般有超时重传、快速重传、SACK、D-SACK

超时重传的情况有两种，一是发送方发送的数据包丢失，二是接收方发送的确认消息丢失，当超过指定的时间没有收到接收方发送的确认消息的时候就需要重传该数据报文

快速重传，当发送方一连收到三个ack的确认报文的时候，就要进行快速重传，当然快速重传是需要重传这个报文之前的所有报文，还是只重传之前的一个

SACK方法，需要在TCP的头部选项字段添加一个SACK的东西，里面存储了接收方接收到的数据，当发生了三个连续报文的时候，通过SACK知道了丢失的报文段，所以可以直接重发这个报文段即可

D-SACK，D-SACK可以使用SACK方法来告诉发送方有哪些数据被重复接收了，可以精确得知道是发出去的包丢了还是接收方回应的ACK丢了

窗口

窗口，我理解就是网络发送中的一种缓冲，因为按照应答模式一问一答，这样的效率就比较低，就可以通过窗口使得发送方无需应答，在窗口的范围内可以发送数据，接收方可以进行累计确认，同时也可以进行流量控制，使的发送方发送的数据不会超过接收方可以接受的数据

流量控制

发送方不能无脑的发送数据给接收方，而是要考虑接收方的处理能力，TCP提供接受窗口和发送窗口来解决这种情况，这就是流量控制

24.操作系统缓冲区和滑动窗口的关系

当服务端的系统资源非常紧张的时候，操作系统可能回直接减少接受缓冲区的大小，当进程无法及时的读取数据的时候，就由可能发生数据丢失。出现这种情况一般是当服务端由于资源紧张减少了缓冲区的大小，也同时减少来窗口的大小，这个时候本来是要发送一个确认报告诉发送方这个窗口减少的情况，但是发送方已经将数据发送过来了超过窗口的大小，所以这个时候就会出现数据丢失的情况

所以TCP规定不允许同时减少缓存和窗口大小，而是先收缩窗口，过段时间再减少缓存，避免丢包

25.糊涂窗口综合症

如果接收方由于接收的很慢，一旦腾出来几个字节的空间，就告诉发送发的窗口这几个字节，如果发送方义无反顾的发送这几个字节的数据，因为TCP+IP的首部就有40个字节，所以，为了传输几个字节的数据而要花费那么大的头部开销，这就是糊涂窗口综合症

解决办法：

- 在发送方，不发送小数据
- 在接收方，不发送小窗口的确认报文

26.拥赛控制

流量控制的目的是商量发送方接收方之间的发送接收速度，使的接收方来的及接收，而拥赛控制则是防止发送方的数据填满整个网络，也就是在控制发送过程中。拥赛控制采取一个拥赛窗口，有了拥赛窗口，那么发送方的发送窗口的值应该为min（拥赛窗口，接收窗口）

拥赛控制一共有以下四种算法：

- 慢启动：TCP刚建立连接的时候，一点一点的提高发送方的发送窗口，规则是每收到一个ACK，拥赛窗口的值cwnd就加1，直到涨到慢开始门限值ssthresh，就启动拥赛避免算法
- 拥赛避免：规则是每收到一个ACK时，cwnd增加1/cwnd，也就是线性增长，如果出现丢包的情况，就会触发重传机制
- 当拥赛发生的时候，重传机制有两种，超时重传和快速重传，超时重传是真正的发生了网络拥堵，首先将ssthresh设置为cwnd /2 ，接着cwnd = 0，然后重新开始慢开始算法。快重传是指当发送方收到三个连续相同的ack确认报文的时候，发送端就会快速的重传，因为可以收到三个ack确认报文，则认为这个时候网络拥堵不会非常严重，所以重新进行拥赛避免算法，同时将cwnd = cwnd / 2， ssthresh = cwnd
- 快恢复：快恢复一般和快重传算法搭配使用，当一连收到三个连续确认的ack的时候

27.ICMP

ICMP是IP层的协议，也就是传输层的协议，互联网控制报文协议，用来反馈发生在通信环境中的遇到的各种问题，例如IP包是否成功送达，在网络通信过程中，如果遇到IP报文没有及时送达，则有ICMP负责通知错误信息。

ICMP一般有两种报文类型：负责诊断的报文`查询报文类型`以及负责通知出错的报文`差错报文类型`，通过发送报文的一些错误码来反馈出错的原因

28.IGMP

IGMP也是在IP层也就是传输层的协议，负责IP组播成员的管理协议，一般工作在主机和最后一跳路由之间也就是直连主机的路由器，规定了主机如何申请加入以及退出一个组

29 Ping的工作原理

ping是基于ICMP协议进行工作的，ICMP传输控制协议，用来反馈在通信过程中遇到的各种问题，一般分为两种报文类型：诊断报文以及通知出错报文里面定义了一些错误类型用于反馈出现错误的原因

当使用ping的时候，会发送一个ICMP的消息报文，同时携带一个序号用于区分不同的icmp包，每经过一个节点该序号值会加1，也会在报文的数据部分插入发送时间，用于计算RTT往返时间，当对面主机收到或者不可达的时候会返回一个icmp的报文，用来判断网络通信是否正常

30 tracert的工作原理

tracert为路由追踪命令，可以用来判断两个主机之间通信经过了什么路由器，路由追踪也是充分利用了icmp协议，利用IP包的生存时间，发送多个数据包，首先将第一个数据包的TTL设置为1，当这个数据包遇到第一个路由器的时候就牺牲了，所以发送一个icmp差错报文，同理依次将TTL的值递增，用来判断在通信过程中遇到了那些路由器