一、运输层TCP和UDP

1. TCP报头格式

2. UDP报头格式

3. TCP/UDP区别（不仅是宏观上的，最好能根据各自的机制讲解清楚）

4. TCP三次握手、四次挥手（这个问题真的要回答吐了，不过真的是面试官最喜欢问的，建议每天手撸一遍，而且不只是每次请求的过程，各种FIN\_WAIT、TIME\_WAIT状态也要掌握）。

5. TCP如何保证数据的可靠传输的（这个问题可以引申出很多子问题，拥塞控制慢开始、拥塞避免、快重传、滑动窗口协议、停止等待协议、超时重传机制，最好都能掌握）

二、应用层

1. HTTP协议（一些报头字段的作用，如cace-control、keep-alive）

2. HTTP状态码（最好结合使用场景，比如在缓存命中时使用哪个）

3. 打开网页到页面显示之间的过程（涵盖了各个方面，DNS解析过程，Nginx请求转发、连接建立和保持过程、浏览器内容渲染过程，考虑的越详细越好）。

4. http和https区别，https在请求时额外的过程，https是如何保证数据安全的

5. POST和GET区别

6. SESSION机制、cookie机制

7. DNS解析过程

三、网络层

1. IP报文格式

2. IP地址分类

3. IP地址子网划分

4. 地址解析协议ARP

5. 交换机和路由器的区别

四、基础

1. OSI7层模型（TCP4层）以及每层的协议

OSI协议、TCP/IP协议以及每层对应的协议。

2. 网络攻击（CSRF、XSS）、DDos攻击

3. (B)IO/NIO/AIO

三者原理，各个语言是怎么实现的

4. Netty

5. Linux内核select poll epoll

五、补充

1. Http1.1和Http1.0的区别

2. Http怎么处理长连接

3. Ping的整个过程。ICMP报文是什么。

4. C/S模式下使用socket通信，几个关键的函数。

一、运输层 – 数据单元：TCP报文段或UDP用户数据报

1. UDP报文格式

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| UDP首部格式 | | | | 数据部分 |
| 源端口(16) | 目的端口(16) | 长度(16) | 检验和(16) | 数据部分 |

(1) 用户数据报UDP有两个字段：<数据>字段和<首部>字段。

(2) 首部字段很简单，只有8个字节（每个字节有8位），由4个字段组成，每个字段的长度都是两个字节。

(3) 字段含义：

① 源端口：源端口号，在需要对方回信的时候选用，不需要的时候可用全0。

② 目的端口：目的端口号，这在终点交付报文时必须要使用到。

③ 长度：UDP用户数据报的长度(首部字段和数据字段)，其最小值是8，也即是只有首部。

④ 检验和：检测UDP用户数据报在传输的过程中是不是有错，有错就丢弃。

(4) 伪首部

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 源IP地址(4) | 目的IP地址(4) | 0(1) | 17(1) | UDP长度(2) |

在计算检验和时，要在UDP用户数据报之前增加12个字节的伪首部，伪首部既不向下传送也不向上递交，仅仅是为了计算检验和。

(5) UDP对应的协议

① DNS：用于域名解析服务，将域名地址转换为IP地址。DNS用的是53号端口。

② SNMP：简单网络管理协议，使用161号端口，是用来管理网络设备的。由于网络设备很多，无连接的服务就体现出其优势。

③ TFTP：，简单文件传输协议，该协议在熟知端口69上使用UDP服务。

2. TCP报文格式

|  |  |  |  |
| --- | --- | --- | --- |
| 源端口号(16) | | | 目的端口号(16) |
| 序列号(32) | | | |
| 确认应答号(32) | | | |
| 数据偏移(4) | 保留(6) | 代码位（6） | 窗口（16） |
| 校验和（16） | | | 紧急指针(16) |
| 选项（长度可变） | | | 填充 |
| 数据部份（长度可变） | | | |

(1) TCP报文段首部的前20个字节是固定的，后面有4N字节是根据需要而增加的选项，因此首部的最小长度是20字节。

(2) 固定部分各个字段的意义：

① 源端口和目的端口字段

各占2字节。端口是运输层与应用层的服务接口。

运输层的复用和分用功能都要通过端口才能实现。

② 序列号字段（重要）

占4字节。TCP是面向字节流的，一个TCP连接中传送的字节流中的每一个字节都按顺序编号。整个要传送的字节流的起始序号必须在连接建立时设置。首部中的序号值是指本报文段所发送的数据的第一个字节的序号。

③ 确认号 （重要）

4个字节，期待收到对方下一个报文段的第一个数据字节的序号。

若确认号=N，表明到序号N-1为止的所有数据都已经正确收到！

④ 数据偏移（即首部长度）

占4位，它指出 TCP 报文段的数据起始处距离 TCP 报文段的起始处有多远。“数据偏移”的单位是 32 位字（以 4 字节为计算单位）。 因首部中还有长度不确定的选项字段，故数据偏移字段是必要的而数据偏移的单位是4字节，则此字段是用来表TCP首部的长度的，最大长度是4\*15=60字节，即选项长度不超过40字节。

⑤ 保留字段

占 6 位，保留为今后使用，但目前应置为 0。

⑥（6个代码控制位，说明本报文段的性质）（重要）

a. 紧急URG：当 URG ==1 时，表明紧急指针字段有效。告诉系统此报文段中有紧急数据，应尽快传送(相当于高优先级的数据)。 而不是按原来的排队顺序传送。当URG=1，发送进程就告诉TCP有紧急数据需传送，TCP就会将紧急数据插入到本报文段的最前面，这需要和首部中的紧急指针字段配合使用.

b. 确认ACK(ACKnowlegment)：当ACK=1，确认号字段才有效，当ACK=0，确认号字段无效。

TCP规定，在连接建立后所有传送的报文段都必须把ACK置1.

c. 推送PSH(PuSH)：当两个进程通信时，有时一端的进程希望键入一个命令后，能立即收到对方的响应，这时TCP就可以将PSH=1，并立即创建一个报文段发送出去，接收方TCP收到PSH=1，就会尽快交付给接收端进程，而不会再等整个缓存填满后再交付。

d. 复位 RST (ReSeT)：当 RST=1 时，表明 TCP 连接中出现严重差错（如由于主机崩溃或其他原因），必须释放连接，然后再重新建立运输连接。RST置1可以用来拒绝一个非法的报文段或者拒绝打开一个连接。

e. 同步SYN：在建立连接时用来同步序号，当SYN=1&&ACK=0，表示这是一个请求连接的报文段，若对方同意建立连接，则在响应报文段中使得SYN=1&&ACK=1。故SYN=1：表示这是一个连接请求和连接接收报文。

f. 终止FIN：用来释放一个连接，当FIN=1，表示此报文段发送方的数据发送完毕，并要求释放连接。

⑦ 窗口：

2个字节，从0开始，窗口指的是发送本报文段的这一方的接收窗口(而不是自己的发送窗口)，窗口值告诉对方：从本报文段首部的确认号开始算起，接收方目前允许(窗口值是经常动态变化的)对发送方发送的数据量。

窗口字段明确指出了现在允许对方发送的数据量。

⑧ 检验和：

占 2 字节。检验和字段检验的范围包括首部和数据这两部分。在计算检验和时，要在 TCP 报文段的前面加上 12 字节的伪首部。

⑨ 紧急指针：2个字节，当URG=1，紧急指针才有意义，指出本报文段中的紧急数据的字节数。注意：当窗口值为0，也可以发送紧急数据。

⑩ 选项：最长为40字节，当没有选项时，TCP首部长度为20字节。

(3) TCP对应的协议

① FTP：定义了文件传输协议，使用21端口。

② Telnet：（远程终端协议）一种用于远程登陆的端口，使用23端口，用户可以以自己的身份远程连接到计算机上，可提供基于DOS模式下的通信服务。

③ SMTP：邮件传送协议，用于发送邮件。服务器开放的是25号端口。

④ POP3：它是和SMTP对应，POP3用于接收邮件。POP3协议所用的是110端口。

⑤ HTTP：是从Web服务器传输超文本到本地浏览器的传送协议。

3. TCP和UDP的区别

(1) TCP面向连接（如打电话要先拨号建立连接）;UDP是无连接的，即发送数据之前不需要建立连接。

(2) TCP提供可靠的服务。也就是说，通过TCP连接传送的数据，无差错，不丢失，不重复，且按序到达;UDP尽最大努力交付，即不可靠交付。

(3) TCP面向字节流，实际上是TCP把数据看成一连串无结构的字节流；UDP是面向报文的

UDP没有拥塞控制，因此网络出现拥塞不会使源主机的发送速率降低（对实时应用很有用，如IP电话，实时视频会议等）。

(4) 每一条TCP连接只能是点到点的；UDP支持一对一，一对多，多对一和多对多的交互通信。

(5) TCP首部开销20字节；UDP的首部开销小，只有8个字节。

(6) TCP的逻辑通信信道是全双工的可靠信道，UDP则是不可靠信道。

3.1. TCP和UDP编程区别

(1) 通常我们在说到网络编程时默认是指TCP编程，即用前面提到的socket函数创建一个socket用于TCP通讯，函数参数我们通常填为SOCK\_STREAM。即socket(PF\_INET, SOCK\_STREAM, 0)，这表示建立一个socket用于流式网络通讯。SOCK\_STREAM这种的特点是面向连接的，即每次收发数据之前必须通过connect建立连接，也是双向的，即任何一方都可以收发数据，协议本身提供了一些保障机制保证它是可靠的、有序的，即每个包按照发送的顺序到达接收方。

(2) 而SOCK\_DGRAM这种是User Datagram Protocol协议的网络通讯，它是无连接的，不可靠的，因为通讯双方发送数据后不知道对方是否已经收到数据，是否正常收到数据。任何一方建立一个socket以后就可以用sendto发送数据，也可以用recvfrom接收数据。根本不关心对方是否存在，是否发送了数据。它的特点是通讯速度比较快。大家都知道TCP是要经过三次握手的，而UDP没有。

(3) TCP

TCP编程的服务器端一般步骤是：

　　1、创建一个socket，用函数socket()；

　　2、设置socket属性，用函数setsockopt(); \* 可选

　　3、绑定IP地址、端口等信息到socket上，用函数bind();

　　4、开启监听，用函数listen()；

　　5、接收客户端上来的连接，用函数accept()； //被动接收连接

　　6、收发数据，用函数send()和recv()，或者read()和write();

　　7、关闭网络连接；

　　8、关闭监听；

TCP编程的客户端一般步骤是：

　　1、创建一个socket，用函数socket()；

　　2、设置socket属性，用函数setsockopt();\* 可选

　　3、绑定IP地址、端口等信息到socket上，用函数bind();\* 可选

　　4、设置要连接的对方的IP地址和端口等属性；

　　5、连接服务器，用函数connect()； //主动连接

　　6、收发数据，用函数send()和recv()，或者read()和write();

　　7、关闭网络连接；

(4) UDP

UDP编程的服务器端一般步骤是：

　　1、创建一个socket，用函数socket()；

　　2、设置socket属性，用函数setsockopt();\* 可选

　　3、绑定IP地址、端口等信息到socket上，用函数bind();

　　4、循环接收数据，用函数recvfrom();

　　5、关闭网络连接；

UDP编程的客户端一般步骤是：

　　1、创建一个socket，用函数socket()；

　　2、设置socket属性，用函数setsockopt();\* 可选

　　3、绑定IP地址、端口等信息到socket上，用函数bind();\* 可选

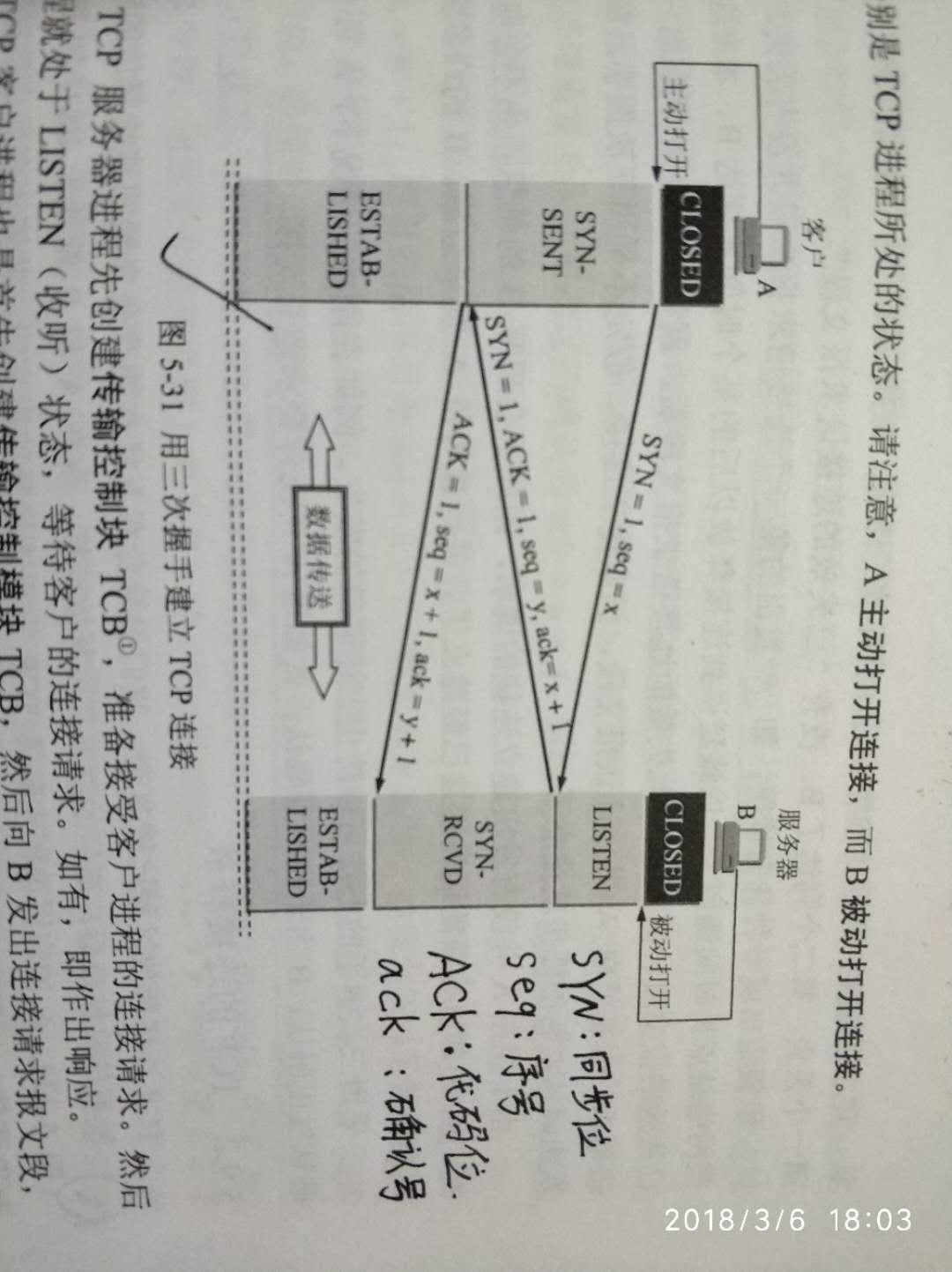
　　4、设置对方的IP地址和端口等属性;

　　5、发送数据，用函数sendto();

　　6、关闭网络连接；

4. TCP建立连接/释放连接

4.1 建立连接



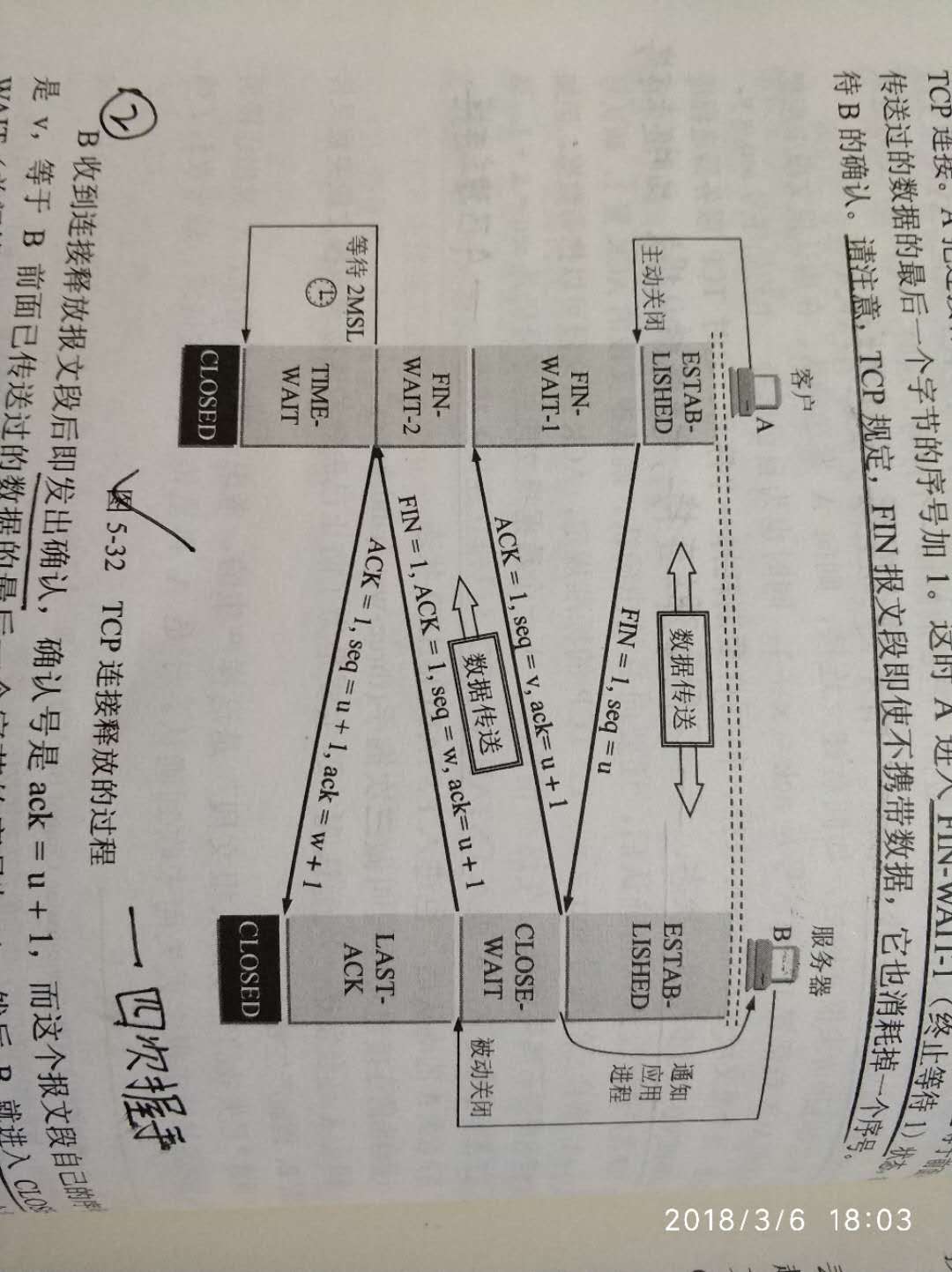
(1) 第一次握手：客户机A发送位码为syn＝1，同时选择一个初始序号seq=x。TCP中，SYN报文段不能携带数据但要消耗一个序号。主机B由SYN=1知道，A要求建立TCP连接。这时，TCP客户机进入SYN-SENT（同步已发送）状态。

(2) 第二次握手： 服务器B收到连接请求的报文段后，如同意建立连接，则向A发送确认。在确认报文中，要将代码位的SYN和ACK都置为1，确认号是ack=x+1，同时也为自己选择一个初始序号seq=y。这时，服务器B的TCP进程进入SYN-RCVD(同步收到)状态。

(3) 第三次握手：客户机A收到B的确认后，还要对B给出确认。确认报文段将ACK置为1，确认号为ack=y+1，自己的序号为seq=x+1。TCP中，ACK报文段可以携带数据，但是如果不携带数据则不消耗序号，在这种情况下，下一个数据报文段的序号仍是seq=x+1。这时候，TCP连接已经建立，A进入ESTAB-LISTED（已建立连接）状态，B收到A的确认后，也进入ESTAB-LISTED（已建立连接）状态。

完成三次握手，主机A与主机B开始传送数据。

4.2 释放连接（四次挥手）



(1) 第一次挥手：数据传输结束后，通信的双方都可以释放连接。A先向B发出连接释放报文段，并停止再发送数据，主动关闭TCP连接。报文的终止控制位FIN置为1，其序号为seq=u，该序号等于前面传送过的数据的最后一个字节加上1。TCP规定，FIN报文即使不携带数据，也要消耗一个序号。这时，A进入FIN-WAIT-1（终止等待1）状态。

(2) 第二次挥手：B收到连接释放报文段后发出确认报文，将确认为ACK置为1，确认号

ack=u+1，序号为seq=v，该序号等于B前面已传送过的数据的最后一个字节的序号加上1。然后B进入CLOSE-WAIT（关闭等待）状态。

① TCP服务器进程这时应通知高层应用进程，因而从A到B这个方向的连接就释放了，此时的TCP连接处于半关闭（half-close）状态，即A已经没有数据要发送了，但是B如果要发送数据，A仍要接收。从B到A这个方向的连接还没有关闭。

② A收到B的确认后，就进入FIN-WAIT-2（终止等待2）状态，等待B发出的释放连接报文段。

(3) 第三次挥手：如B已经没有要向A发送的数据了，B向A发送连接释放报文段。报文段将FIN置为1，ACK置为1，确认号ack=u+1（必须重复上一条的确认号），序号seq=w（假设B在第二次挥手后又发送了数据）。这时，B进入LAST-ACK（最后确认）状态，等待A的确认。

//等待TIME-WAIT是为了让A发送的最后一个ACK报文段能够到达B，也是为了不会让本链接的报文段影响到下一个连接

(4) 第四次挥手：A收到B的连接释放确认报文后，必须发出确认。报文段中将ACK置为1，确认号ack=w+1（上条确认报文序号加1），自己序号seq=u+1。然后进入TIME-WAIT（时间等待状态）。现在TCP连接还没有完全释放掉，必须经过事件等待计时器设置的时间

2MSL后，A才进入到CLOSED状态（B在收到A的确认报文后，就进入CLOSED状态）。当A撤销相应的传输控制块TCB后，就结束了这次的TCP连接。

5. TCP如何保证可靠的运输。（实现原理）

(1) 可靠传输的工作原理

① 停止等待协议

② 连续ARQ协议

(2) 可靠传输的实现

① 以字节为单位的滑动窗口

② 超时重传

③ 选择确认SACK

(3) 流量控制

(4) 拥塞控制（重点）

5.1 可靠传输的工作原理

(1) 停止等待协议

① 定义：“停止等待”就是每发送完一个分组就停止发送，等待对方的确认，在收到确认后再发送下一个分组。

(2) 连续ARQ协议

① 定义：

发送方维持一个发送窗口，发送方每收到一个确认，就把发送窗口向前滑动一个分组的位置。接收方一般采用累积确认的方式，就是不必对每个分组逐个发送确认，而是对按序到达的最后一个分组发送确认。

5.2 可靠传输的实现

(1) 滑动窗口协议

① TCP的滑动窗口协议是以字节为单位的。发送方A的发送窗口表示：在没有收到B的确认的情况下，A可以连续把窗口内的数据都发送出去。凡是已经发送过的数据，在未收到确认之前都必须暂时保留，以便在超时重传时使用。

② 发送窗口的位置由窗口前沿和后沿的位置共同确定。发送窗口后沿的变化情况有两种，一种是不动（没有收到新的确认），一种是前移（收到了新的确认）。

③ 发送方的应用进程把字节流写入TCP的发送缓存，接收方的应用进程从TCP的接收缓存中读取字节流。

(2) 超时重传

① TCP的发送方在规定的时间内没有收到确认就要重传已发送的报文段。

(3) 选择确认

① 应用背景：B收到的报文段没有差错，只是没有按照序号，中间还缺少一些序号的数据。

② 选择确认就是接收方先接收下这些数据，再将缺失的信息准确的告诉给发送方。在建立TCP连接时，要在TCP首部的选项中加上“允许SACK”的选项。

5.3 流量控制

(1) 流量控制就是让发送方的发送速率不要太快，要让接收方来得及接收。

5.4 拥塞控制

(1) 拥塞定义：

在某段时间，若对网络中某一资源的需求超过了该资源所能提供的可用部分，网络的性能就要变差。

(2) 拥塞控制的方法有：慢开始和拥塞避免，快重传，快恢复。

5.4.1慢开始和拥塞避免

(1) 发送方维持一个拥塞窗口的状态变量，大小取决于网络的拥塞程度，并且动态的变化。发送方让自己的发送窗口等于拥塞窗口，也可以根据接收方的接收能力将发送窗口小于拥塞窗口。

(2) 慢开始：（开始值设置比较小）

当主机开始发送数据时，由小到大逐渐增大发送窗口，也就是由小到大逐渐增大拥塞窗口的值。通常在刚刚开始的时候，先把拥塞窗口设置为一个最大报文段MSS的数值。在每收到一个对新的报文段的确认后，把拥塞窗口增加至多一个MSS的数值。

(3) 拥塞避免（线性增长）

让拥塞窗口缓慢地增大，即每经过一个往返时间RTT就把发送方的拥塞窗口加1，而不是加倍。这样让拥塞窗口按照线性规律缓慢增长，比慢开始的增长速率慢。

5.4.2 快重传和快恢复

(1) 快重传

快重传要求接收方每收到一个失序的报文段后马上发出重复确认，而不是等发送数据时候才捎带确认，这样可以使发送方尽快知道有报文段没有送达对方。

(2) 快恢复

① 当发送方连续收到三个重复确认后，就执行“乘法减小”算法，把慢开始的限制减半，这是为了预防网络发生拥塞。

② 然后将慢开始算法切换为拥塞避免算法，让 拥塞窗口缓慢地线性增长。

5.5 拥塞控制和流量控制的区别

(1) 拥塞控制：防止过多的数据注入到网络中，这样可以使网络中的路由器或链路不致过载。拥塞控制所要做的都有一个前提：网络能够承受现有的网络负荷。拥塞控制是一个全局性的过程，涉及到所有的主机、路由器，以及与降低网络传输性能有关的所有因素。

(2) 流量控制：指点对点通信量的控制，是端到端的问题。流量控制所要做的就是抑制发送端发送数据的速率，以便使接收端来得及接收。

二、应用层

1. HTTP协议报文（面向文本）

1.1 请求报文 – 从客户向服务器发送请求报文

【请求行】 方法(空格) URL(空格) 版本 CRLF

【首部行】 首部字段名：值CRLF //请求首部字段

首部字段名：值CRLF //通用首部字段

…… //实体首部字段

CRLF

【实体主体】 通常不用

(1) 开始行：用于区分是请求报文还是响应报文。在请求报文中叫做请求行，在响应报文中叫做状态行。在开始行的三个字段之间都以空格分隔开，最后“CR”和“LF”分别代表“回车”和“换行”。

(2) 首部行：用来说明浏览器、服务器或报文主体的一些信息。首部行的格式为字段名：值。每一行在最后都要有CRLF 。整个首部行结束后，还有一个CRLF将首部行和实体主体分隔开。

(3) 实体主体：在请求报文和响应报文一般都没有这个字段。

（请求报文方法）

|  |  |
| --- | --- |
| 方法（操作） | 意义 |
| OPTION | 请求一些选项的信息 |
| GET | 请求读取由URL所标志的信息 |
| HEAD | 请求读取由URL所标志的信息的首部 |
| POST | 给服务器添加信息（例如注释） |
| PUT | 在指明的URL下存储一个文档 |
| DELETE | 删除指明的URL所标志的资源 |
| TRACE | 用来进行环回测试的请求报文 |
| CONNECT | 用于代理服务器 |

1.2 响应报文 – 从服务器到客户的回答

【状态行】 版本(空格) 状态码(空格) 短语 CRLF

【首部行】 首部字段名：值CRLF //响应首部字段

首部字段名：值CRLF //通用首部字段

…… //实体首部字段

CRLF

【实体主体】 通常不用

1.3 4种首部字段类型

（通用首部字段）- 请求和响应报文都会使用的首部



（请求首部字段）

定义： 从客户端向服务器发送请求报文时使用的首部，补充了请求的附加内容、客户端信息、相应内容相关优先级信息。



（响应首部字段）

定义： 从服务器向客户端返回响应报文时使用的首部。补充了资源内容更新时间等与实体有关的信息。



（实体首部字段）

定义： 针对请求报文和响应报文的实体部分使用的首部，补充了资源内容更新时间与实体有关的信息。



1.4 响应报文的状态码

(1) 状态码都是三位数字，分为5大类共33种

|  |  |
| --- | --- |
| 1XX | 表示通知信息的，如请求收到了或正在处理。 |
| 2XX | 表示成功，如接受或知道了。 |
| 3XX | 表示重定向，如要完成请求还必须采取进一步的行动。 |
| 4XX | 表示客户的差错，如请求中有错误的语法或不能完成。 |
| 5XX | 表示服务器的差错，如服务器失效无法完成请求。 |

2. http的状态码，见1

3. 一个页面从输入URL 到页面加载显示完成的过程中都发生了什么！

3.1 简单过程

(1) 浏览器分析链接指向页面的URL，

(2) 浏览器根据请求的URL交给DNS域名解析，找到真实IP。（域名解析 ）

(3) 向服务器发起请求，和服务器建立TCP连接。

(4) 服务器交给后台处理完成后返回数据，浏览器接收文件（HTML、JS、CSS、图象等），

释放TCP连接。

(5) 浏览器对加载到的资源（HTML、JS、CSS等）进行语法解析，建立相应的内部数据结构（如HTML的DOM）；

(6) 载入解析到的资源文件，渲染页面，完成。

3.2 详细过程

(1) 当发送一个URL请求时，不管这个URL是Web页面的URL还是Web页面上每个资源的URL，浏览器会开启一个线程来处理这个请求，对URL 分析判断如果是 http 协议就按照 Web 方式来处理;

(2) 调用浏览器内核中的对应方法，比如 WebView 中的 loadUrl 方法;

(3) 在远程DNS服务器上启动一个DNS查询，这能使浏览器获得请求对应的IP地址。通过DNS解析获取网址的IP地址，设置 UA 等信息发出第二个GET请求;

(4) 进行HTTP协议会话，客户端发送报头(请求报头)：浏览器与远程Web服务器通过TCP三次握手协商来建立一个TCP/IP连接。该握手包括一个同步报文，一个同步-应答报文和一个应答报文，这三个报文在 浏览器和服务器之间传递。该握手首先由客户端尝试建立起通信，而后服务器应答并接受客户端的请求，最后由客户端发出该请求已经被接受的报文。

(5) 进入到web服务器上的 WebServer，如 Apache、Tomcat、Node.JS 等服务器;

(6) 进入部署好的后端应用，如PHP、Java、JavaScript、Python 等，找到对应的请求处理;

(7) 处理结束回馈报头，此处如果浏览器访问过，缓存上有对应资源，会与服务器最后修改时间对比，一致则返回304;

(8) 浏览器开始下载html文档(响应报头，状态码200)，同时使用缓存。下载完成后释放TCP连接。

(9) 文档树建立，根据标记请求所需指定MIME类型的文件（比如css、js）,同时设置了cookie;

(10) 浏览器会解析HTML生成DOM Tree，其次会根据CSS生成CSSRule Tree，而javascript又可以根据DOMAPI操作DOM，执行事件绑定等，页面显示完成。

4. HTTP和HTTPS区别

4.1 概念

(1) http是HTTP协议运行在TCP之上。所有传输的内容都是明文，客户端和服务器端都无法验证对方的身份。

(2) https是HTTP运行在SSL/TLS之上，SSL/TLS运行在TCP之上。所有传输的内容都经过加密，加密采用对称加密，但对称加密的密钥用服务器方的证书进行了非对称加密。此外客户端可以验证服务器端的身份，如果配置了客户端验证，服务器方也可以验证客户端的身份。

4.2 区别

(1) https协议需要到ca申请证书，一般免费证书较少，因而需要一定费用。

(2) http是超文本传输协议，信息是明文传输，https则是具有安全性的ssl加密传输协议。

(3) http和https使用的是完全不同的连接方式，用的端口也不一样，前者是80，后者是443。

(4) http的连接很简单，是无状态的；HTTPS协议是由SSL+HTTP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。

(5) HTTPS协议握手阶段比较费时，会使页面的加载时间延长近50%，增加10%到20%的耗电。

(6) HTTPS连接缓存不如HTTP高效，会增加数据开销和功耗，甚至已有的安全措施也会因此而受到影响。

5. POST和GET区别

(1) 使用GET方法时，查询字符串（键值对）被附加在URL地址后面一起发送到服务器：

使用POST方法时，查询字符串在POST信息中单独存在，和HTTP请求一起发送到服务器。所以使用POST会比较安全。

(2) GET请求会被浏览器主动cache（缓存），而POST不会，除非手动设置。

(3) GET请求参数会被完整保留在浏览器历史记录里，而POST中的参数不会被保留。

(4) GET请求在URL中传送的参数是有长度限制的，而POST没有。

(5) 对参数的数据类型，GET只接受ASCII字符，而POST没有限制。

(6) GET产生一个TCP数据包；POST产生两个TCP数据包。（对于GET方式的请求，浏览器会把http header和data一并发送出去，服务器响应200（返回数据）；而对于POST，浏览器先发送header，服务器响应100 continue，浏览器再发送data，服务器响应200 ok（返回数据））

6. Cookie和Session的区别

6.1 cookie机制

cookie机制采用的是在客户端保持状态的方案。

① 正统的cookie分发是通过扩展HTTP协议来实现的，服务器通过在HTTP的响应头中加上一行特殊的指示以提示浏览器按照指示生成相应的cookie。然而纯粹的客户端脚本如JavaScript或者VBScript也可以生成cookie。而cookie的使用是由浏览器按照一定的原则在后台自动发送给服务器的。浏览器检查所有存储的cookie，如果某个cookie所声明的作用范围大于等于将要请求的资源所在的位置，则把该cookie附在请求资源的HTTP请求头上发送给服务器。

② cookie的内容主要包括：名字，值，过期时间，路径和域。路径与域一起构成cookie的作用范围。若不设置过期时间，则表示这个cookie的生命期为浏览器会话期间，关闭浏览器窗口，cookie就消失。这种生命期为浏览器会话期的cookie被称为会话cookie。会话cookie一般不存储在硬盘上而是保存在内存里，当然这种行为并不是规范规定的。若设置了过期时间，浏览器就会把cooki保存到硬盘上，关闭后再次打开浏览器，这些cookie仍然有效直到超过设定的过期时间。存储在硬盘上的cookie可以在不同的浏览器进程间共享，比如两个IE窗口。而对于保存在内存里的cookie，不同的浏览器有不同的处理方式。

6.2 session机制

session机制是一种服务器端的机制，服务器使用一种类似于散列表的结构（也可能就是使用散列表）来保存信息。

① 当程序需要为某个客户端的请求创建一个session时，服务器首先检查这个客户端的请求里是否已包含了一个session标识（称为session id），如果已包含则说明以前已经为此客户端创建过session，服务器就按照session id把这个session检索出来使用（检索不到，会新建一个），如果客户端请求不包含session id，则为此客户端创建一个session并且生成一个与此session相关联的session id，session id的值应该是一个既不会重复，又不容易被找到规律以仿造的字符串，这个session id将被在本次响应中返回给客户端保存。

② 保存这个session id的方式可以采用cookie，这样在交互过程中浏览器可以自动的按照规则把这个标识发挥给服务器。一般这个cookie的名字都是类似于SEEESIONID。但cookie可以被人为的禁止，则必须有其他机制以便在cookie被禁止时仍然能够把session id传递回服务器。经常被使用的一种技术叫做URL重写，就是把session id直接附加在URL路径的后面。还有一种技术叫做表单隐藏字段。就是服务器会自动修改表单，添加一个隐藏字段，以便在表单提交时能够把session id传递回服务器。

6.3 区别

(1) 存取方式的不同

① Cookie中只能保管ASCII字符串，假如需求存取Unicode字符或者二进制数据，需求先进行编码。Cookie中也不能直接存取Java对象。若要存储略微复杂的信息，运用Cookie是比拟艰难的。

② Session中能够存取任何类型的数据，包括而不限于String、Integer、List、Map等。Session中也能够直接保管Java Bean乃至任何Java类，对象等，运用起来十分便当。能够把Session看做是一个Java容器类。

(2) 隐私策略的不同

Cookie存储在客户端阅读器中，对客户端是可见的，客户端的一些程序可能会窥探、复制以至修正Cookie中的内容。而Session存储在服务器上，对客户端是透明的，不存在敏感信息泄露的风险。

(3) 有效期上的不同

① 如果没有设置Cookie的过期时间，cookie就和session的生存周期一样，随着会话的结束而结束，此时cookie存在客户机内存中。如果设置过期时间，cookie则会被存储到客户机的硬盘上，有效期取决于过期时间。

② 由于Session依赖于名为JSESSIONID的Cookie，而Cookie JSESSIONID的过期时间默许为–1，只需关闭了阅读器该Session就会失效，因而Session不能完成信息永世有效的效果。运用URL地址重写也不能完成。而且假如设置Session的超时时间过长，服务器累计的Session就会越多，越容易招致内存溢出。Session是存在于服务器的内存中的。

(4) 服务器压力的不同

① Session是保管在服务器端的，每个用户都会产生一个Session。假如并发访问的用户十分多，会产生十分多的Session，耗费大量的内存。因而像Google、Baidu、Sina这样并发访问量极高的网站，是不太可能运用Session来追踪客户会话的。

② Cookie保管在客户端，不占用服务器资源。假如并发阅读的用户十分多，Cookie是很好的选择。关于Google、Baidu、Sina来说，Cookie或许是唯一的选择。

(5) 浏览器支持的不同

Cookie是需要客户端浏览器支持的。假如客户端禁用了Cookie，或者不支持Cookie，则会话跟踪会失效。假如客户端浏览器不支持Cookie，需要运用Session以及URL地址重写。需要注意的是一切的用到Session程序的URL都要进行URL地址重写，否则Session会话跟踪还会失效。

(6) 跨域支持上的不同

Cookie支持跨域名访问，例如将domain属性设置为“.biaodianfu.com”，则以“.biaodianfu.com”为后缀的一切域名均能够访问该Cookie。跨域名Cookie如今被普遍用在网络中，例如Google、Baidu、Sina等。而Session则不会支持跨域名访问。Session仅在他所在的域名内有效。

7. DNS解析过程

(1) 先查询本地的DNS缓存记录，如果有，则不需要向本地域名解析服务器查询，没有进行第2步。windows下我们也可以通过 ipconfig /displaydns 查看缓存记录

(2) 以UDP数据报的方式向本地域名解析服务器查询，本地DNS服务器会先查询自己的高速缓存，有则返回给客户机解析的结果，没有则进行第3步

(3) 本地DNS服务器向根域名服务器查询，根域名返回顶级DNS服务器的地址

(4) 顶级服务器接收到本地DNS服务器的请求后，返回权限服务器的地址

(5) 本地DNS服务器向权限DNS服务器发出请求后，得到解析结果，返回给客户机，并缓存。

本地缓存→本地域名服务器（缓存）→根域名服务器（顶级DNS服务器）→顶级DNS（权限DNS）→权限DNS服务器→解析结果

三、网络层

1. IP报文格式

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 固定部分 | 版本(4) | 首部长度(4) | | 区分服务(8) | 总长度(16) | |
| 标识(16) | | | | 标志(3) | 片偏移(13) |
| 生存时间(8) | | 协议(8) | | 首部检验和(16) | |
| 源地址(32) | | | | | |
| 目的地址(32) | | | | | |
| 可变部分 | 可选字段（长度可变） | | | | 填充 | |
| 数据部分 | | | | | | |

(1) 版本：指IP协议的版本

(2) 首部长度：表示首部的总长度，固定部分为20字节，可变部分最长为40字节。

(3) 区分服务：一般不使用

(4) 总长度：指首部和数据之和的长度，最大为2^16-1=65535字节

(5) 标识：计数器，每发一个数据报，计数器就加1，并将计数器的值赋值给标识。

(6) 标志：“MF”为1表示“还有分片”，“DF”为1表示“不能分片”。

(7) 片偏移：较长的分组在分片后，某片在原分组中的相对位置。

(8) 生存时间：数据报在网络中的寿命。

(9) 协议：指出该数据报使用的是什么协议。IP-4 , IPv6-41

(10) 首部检验和：只检验首部，防止数据报出错的措施。

2. IP地址分类

(1) A类地址 ：0xxxxxxx（8位） 主机号（24位） 占50%

0.0.0.0 开始到 127.x.x.x

(2) B类地址：10xxxxxx（16位）主机号（16位） 占25%

128.x.x.x 开始到 191.x.x.x

(3) C类地址：110xxxxx（24位）主机号（8位） 占12.5%

192.x.x.x 开始到 223.x.x.x

(4) D类地址：多播地址

(5) E类地址：保留地址

3. IP地址子网划分

(1) 子网掩码 “与” IP地址 得到 子网的网络地址

A类：255.0.0.0

B类：255.255.0.0

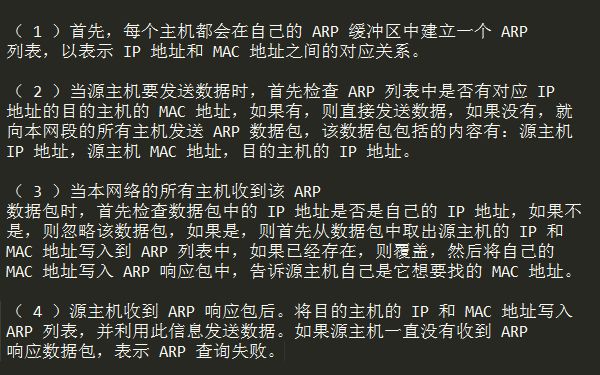
C类：255.255.255.0

注意：要除去子网全1或者全0两种情况。

4. 地址解析协议ARP

(1) 作用：从网络层的IP地址解析出在数据链路层使用的硬件地址。

(2) 具体过程：



5. 交换机和路由器的区别

(1) 工作所在的OSI层次不一样（根本区别，导致接下来的区别）

① 交换机工作在 OSI模型的数据链路层，所以工作原理比较简单；

② 路由器工作在OSI模型的网络层，具备更多的网络协议信息，所以可以做出更好的数据转发策略。

(2) 数据转发所依据的对象也不一样。

① 交换机工作在数据链路层，所以交换机转发数据依靠的是每个物理地址（MAC地址），MAC地址一般是设备生产商在设备出厂时固定在设备中的，不能进行更改。

② 路由器工作在网络层，所以其交换数据依靠网络地址（IP地址），而IP地址是由网络管理员自己分配或者系统自动获取的。

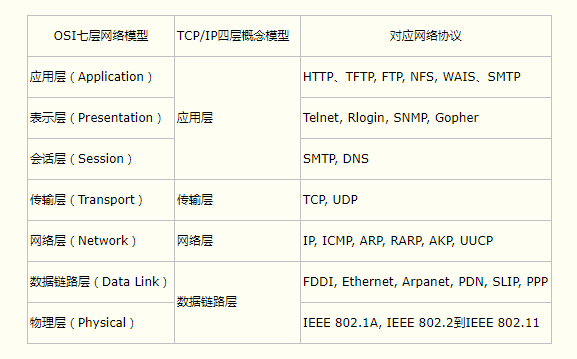
(3) 是否可以分割广播域

① 由交换机连接的所有端口仍然属于同一个广播域，所以极有可能会产生数据拥堵；

② 连接到路由器上的所有端口不在属于同一个广播域，所以不会产生类似的数据拥堵问题。

四、基础

1. OSI七层模型或者TCP/IP四层模型以及对应的协议



2. 网络攻击

(1) XSS全称跨站脚本攻击(Cross Site Scripting)，顾名思义，就是通过向某网站写入js脚本来实现攻击。

(2) CSRF全称跨站请求伪造（Cross-site request forgery）。一听好像跟XSS没什么差，确实XSS也是实现CSRF的一种手段。XSS点是在于跨站注入脚本，进而干坏事。CSRF点在于利用各种手段，实现伪造其他网站请求，不一定是通过XSS。

(3) 分布式拒绝服务(DDoS:Distributed Denial of Service)攻击指借助于客户/服务器技术，将多个计算机联合起来作为攻击平台，对一个或多个目标发动DDoS攻击，从而成倍地提高拒绝服务攻击的威力。

3. BIO 、NIO、 AIO

(1) 同步和异步的区别就在于读操作由谁完成：同步的Reactor是指程序发出读请求后，由分离器监听到可以进行读操作时（需要获得读操作条件）通知事件处理器进行读操作，异步的Proactor是指程序发出读请求后，操作系统立刻异步地进行读操作了，读完之后在通知分离器，分离器激活处理器直接取用已读到的数据。

(2) 同步阻塞IO（BIO）

我们熟知的Socket编程就是BIO，一个socket连接一个处理线程（这个线程负责这个Socket连接的一系列数据传输操作）。阻塞的原因在于：操作系统允许的线程数量是有限的，多个socket申请与服务端建立连接时，服务端不能提供相应数量的处理线程，没有分配到处理线程的连接就会阻塞等待或被拒绝。

(3) 同步非阻塞IO（NIO）

New IO是对BIO的改进，基于Reactor模型。我们知道，一个socket连接只有在特点时候才会发生数据传输IO操作，大部分时间这个“数据通道”是空闲的，但还是占用着线程。NIO作出的改进就是“一个请求一个线程”，在连接到服务端的众多socket中，只有需要进行IO操作的才能获取服务端的处理线程进行IO。这样就不会因为线程不够用而限制了socket的接入。客户端的socket连接到服务端时，就会在事件分离器注册一个 IO请求事件 和 IO 事件处理器。在该连接发生IO请求时，IO事件处理器就会启动一个线程来处理这个IO请求，不断尝试获取系统的IO的使用权限，一旦成功（即：可以进行IO），则通知这个socket进行IO数据传输。

(3) 异步阻塞IO（AIO）

NIO是同步的IO，是因为程序需要IO操作时，必须获得了IO权限后亲自进行IO操作才能进行下一步操作。AIO是对NIO的改进（所以AIO又叫NIO.2），它是基于Proactor模型的。每个socket连接在事件分离器注册 IO完成事件 和 IO完成事件处理器。程序需要进行IO时，向分离器发出IO请求并把所用的Buffer区域告知分离器，分离器通知操作系统进行IO操作，操作系统自己不断尝试获取IO权限并进行IO操作（数据保存在Buffer区），操作完成后通知分离器；分离器检测到 IO完成事件，则激活 IO完成事件处理器，处理器会通知程序说“IO已完成”，程序知道后就直接从Buffer区进行数据的读写。

4. 服务器常用端口以及服务



五、补充

1. Http1.1和Http1.0的区别

(1) HTTP 1.1支持长连接（PersistentConnection）和请求的流水线（Pipelining）处理

① HTTP 1.0规定浏览器与服务器只保持短暂的连接，浏览器的每次请求都需要与服务器建立一个TCP连接，服务器完成请求处理后立即断开TCP连接，服务器不跟踪每个客户也不记录过去的请求。

② HTTP 1.1则支持持久连接Persistent Connection, 并且默认使用persistent connection. 在同一个tcp的连接中可以传送多个HTTP请求和响应. 多个请求和响应可以重叠，多个请求和响应可以同时进行. 更加多的请求头和响应头(比如HTTP1.0没有host的字段).

③ 请求的流水线（Pipelining）处理，在一个TCP连接上可以传送多个HTTP请求和响应，减少了建立和关闭连接的消耗和延迟。例如：一个包含有许多图像的网页文件的多个请求和应答可以在一个连接中传输，但每个单独的网页文件的请求和应答仍然需要使用各自的连接。 HTTP 1.1还允许客户端不用等待上一次请求结果返回，就可以发出下一次请求，但服务器端必须按照接收到客户端请求的先后顺序依次回送响应结果，以保证客户端能够区分出每次请求的响应内容。

(2) HTTP 1.1增加host字段

① 在HTTP1.0中认为每台服务器都绑定一个唯一的IP地址，因此，请求消息中的URL并没有传递主机名（hostname）。但随着虚拟主机技术的,m2ds发展，在一台物理服务器上可以存在多个虚拟主机（Multi-homed Web Servers），并且它们共享一个IP地址。

② HTTP1.1的请求消息和响应消息都应支持Host头域，且请求`消息中如果没有Host头域会报告一个错误（400 Bad Request）。此外，服务器应该接受以绝对路径标记的资源请求。

(3) 100(Continue) Status(节约带宽)

HTTP/1.1加入了一个新的状态码100（Continue）。客户端事先发送一个只带头域的请求，如果服务器因为权限拒绝了请求，就回送响应码401（Unauthorized）；如果服务器接收此请求就回送响应码100，客户端就可以继续发送带实体的完整请求了。100 (Continue) 状态代码的使用，允许客户端在发request消息body之前先用request header试探一下server，看server要不要接收request body，再决定要不要发request body。

2. Http怎么处理长连接/、

(1) 短连接

在HTTP/1.0中，默认使用的是短连接。也就是说，浏览器和服务器每进行一次HTTP操作，就建立一次连接，但任务结束就中断连接。如果客户端浏览器访问的某个HTML或其他类型的 Web页中包含有其他的Web资源、，如JavaScript文件、图像文件、CSS文件等；当浏览器每遇到这样一个Web资源，就会建立一个HTTP会话。

(2) 长连接

但从 HTTP/1.1起，默认使用长连接，用以保持连接特性。使用长连接的HTTP协议，会在响应头有加入这行代码：Connection:keep-alive在使用长连接的情况下，当一个网页打开完成后，客户端和服务器之间用于传输HTTP数据的 TCP连接不会关闭，如果客户端再次访问这个服务器上的网页，会继续使用这一条已经建立的连接。Keep-Alive不会永久保持连接，它有一个保持时间，可以在不同的服务器软件（如Apache）中设定这个时间。实现长连接要客户端和服务端都支持长连接。HTTP协议的长连接和短连接，实质上是TCP协议的长连接和短连接。原理是TCP的保活功能。

3. ping某个域名的过程

(1) 首先本机发送域名请求数据到PC设置的DNS ip

(2) PC通过子网掩码判断DNS ip是本网段还是跨网段（这里只考虑跨网段）

(3) 由于是跨网段，PC发送DNS域名解析数据包到PC设置的网关ip上。（此时先要进行二层的mac转发，PC查看本机arp缓存表，如果表中有网关的mac地址，直接转发，如果没有，使用arp解析协议解析到网关的mac地址。之后封装成数据帧发送到三层网络层）此时PC发送三层数据到网关，源地址为PC内网地址，目的地址为DNS ip地址。而在二层源mac地址为PC mac地址，目的mac地址为网关mac地址。

(4) 路由内网网关收到数据包，根据数据包的目的地址，查看路由表。根据路由表发送数据到下一跳上。（发送前，数据到达路由外网端口，会根据nat地址转换配置。形成一条内网ip+port与外网ip+port的一一对应关系。）

(5) 发送到下一跳和内网通信都是一样的，查看路由arp缓存表，如果有下一跳mac地址，就直接发送，没有的话需要arp协议解析一下。

(6) 对端路由收到数据包，再接着根据路由表判断下一跳。这样一跳一跳地，最后到达DNS服务器。服务器将查询结果返回。

(7) 返回的数据包在ISP的网络里最后寻址到你的路由器上，你的路由器收到数据包后，会查询路由nat连接表，寻找ip+port关系对应的内网ip。拆分数据包，封装成帧，最后PC收到域名对应的ip地址。

【到这里，域名解析过程完成，接下来ping对方ip，过程与上面几乎一样】

(1) 再发起一次PC到目的域名ip地址的一次ping请求信息

(2) PC通过子网掩码判断对方ip是本网段还是跨网段（这里只考虑跨网段）

(3) 由于是跨网段，PC发送数据包到网关ip上。

(4) 路由内网网关收到数据包，根据数据包的目的地址，查看路由表。根据路由表发送数据到下一跳上。（发送前，数据到达路由外网端口，会根据nat地址转换配置。形成一条内网IP+port与外网ip+port的一一对应关系。）

(5) 发送到下一跳和内网通信都是一样的，查看路由arp缓存表，如果有下一跳mac地址，就直接发送，没有的话就是要arp协议解析一下。

(6) 服务器收到数据包后，会重新构建一个ICMP应答包，然后返回。

(7) 返回的数据包在ISP的网络里最后寻址到你的路由器上，你的路由器收到数据包后，会查询路由nat连接表，寻找ip+port关系对应的内网ip。拆分数据包，封装成帧，最后PC收到ICMP应答数据包。

(8) 整个过程到此结束。在整个这个过程中，源ip地址和目的ip地址是不变的（内网到路由器段不算在内）而mac地址是变的。

ICMP协议是保证IP协议中数据包能够确认交付一个协议。

4. C/S模式下使用socket通信，几个关键的函数。

(1) 具体流程

服务器调用socket()、bind()、listen()完成初始化后，调用accept()阻塞等待，处于监听端口的状态，客户端调用socket()初始化后，调用connect()发出SYN段并阻塞等待服务器应答，服务器应答一个SYN-ACK段，客户端收到后从connect()返回，同时应答一个ACK段，服务器收到后从accept()返回。

4.1 基本socket函数

(1) socket函数

包含头文件<sys/socket.h>

功能：创建一个套接字用于通信

原型：int socket(int domain, int type, int protocol);

参数

domain ：指定通信协议族（protocol family），AF\_INET、AF\_INET6、AF\_UNIX等

type：指定socket类型，流式套接字SOCK\_STREAM，数据报套接字SOCK\_DGRAM，原始套接字SOCK\_RAW

protocol ：协议类型，IPPROTO\_TCP等；一般由前两个参数就决定了协议类型，设置为0即可。

返回值：成功返回非负整数， 它与文件描述符类似，我们把它称为套接口描述字，简称套接字。失败返回-1

(2) bind函数

包含头文件<sys/socket.h>

功能：绑定一个本地地址到套接字

原型：int bind(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen);

参数

sockfd：socket函数返回的套接字

addr：要绑定的地址

addrlen：地址长度

返回值：成功返回0，失败返回-1

(3) listen函数

包含头文件<sys/socket.h>

功能：将套接字用于监听进入的连接

原型：int listen(int sockfd, int backlog);

参数

sockfd：socket函数返回的套接字

backlog：已完成三次握手的最大连接个数

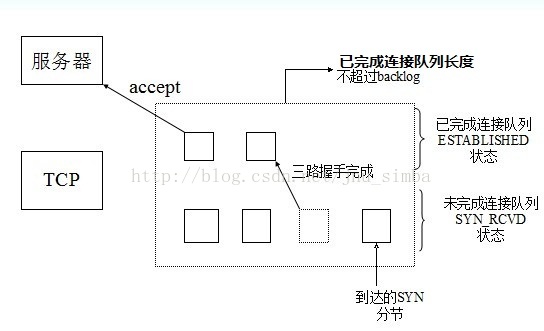
返回值：成功返回0，失败返回-1

一般来说，listen函数应该在调用socket和bind函数之后，调用函数accept之前调用。

对于给定的监听套接口，内核要维护两个队列：

1、已由客户发出并到达服务器，服务器正在等待完成相应的TCP三路握手过程

2、已完成连接的队列



(4) accept函数 -- 阻塞式

包含头文件<sys/socket.h>

功能：从已完成连接队列返回第一个连接，如果已完成连接队列为空，则阻塞。

原型：int accept(int sockfd, struct sockaddr \*addr, socklen\_t \*addrlen);

参数

sockfd：服务器套接字

addr：将返回对等方的套接字地址

addrlen：返回对等方的套接字地址长度

返回值：成功返回非负整数，失败返回-1

(5) connect函数 – 阻塞式

包含头文件<sys/socket.h>

功能：建立一个连接至addr所指定的套接字

原型：int connect(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen);

参数

sockfd：未连接套接字

addr：要连接的套接字地址

addrlen：第二个参数addr长度

返回值：成功返回0，失败返回-1