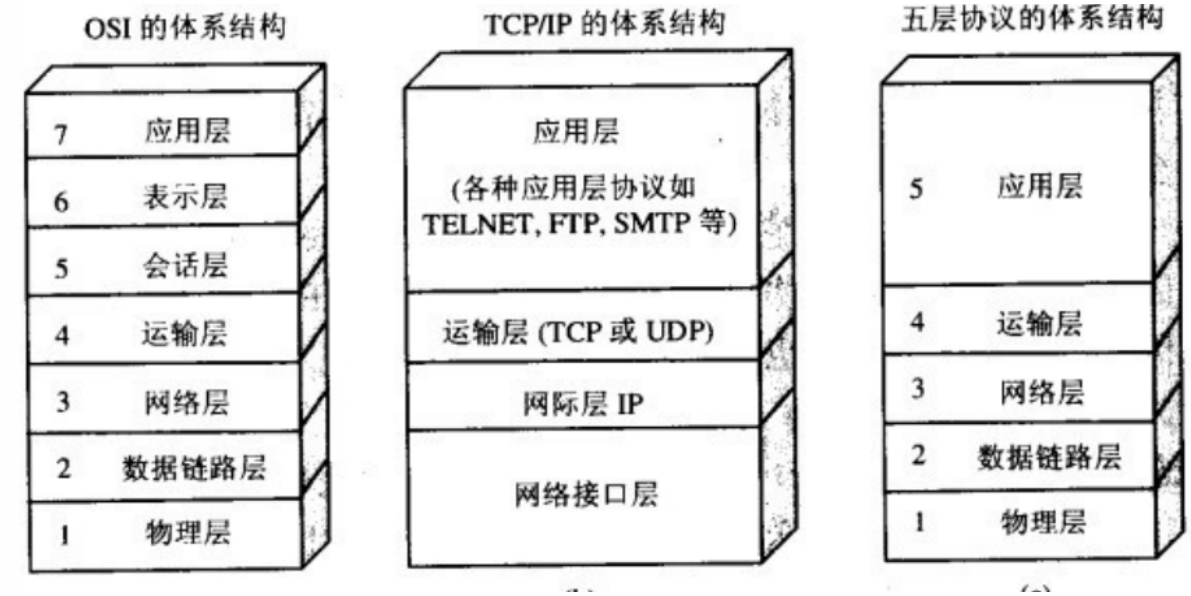
1.OSI 7层模型(5层协议体系)



一、应用层协议:

DNS, HTTP, HTTPS

二、传输层协议:

TCP-传输控制协议,面向连接的,可靠的数据传输服务

UDP-用户数据协议,无连接的,尽最大努力的传输(不可靠)

三、网络层:

IP:

ARP:

RARP:

四、数据链路层:

4.1 概述

封装成帧

- “帧”是数据链路层数据的基本单位
- 发送端在网络层的一段数据前后添加特定标记形成“帧”
- 接收端根据前后特定标记识别出“帧”

物理层才不管你“帧”不“帧”

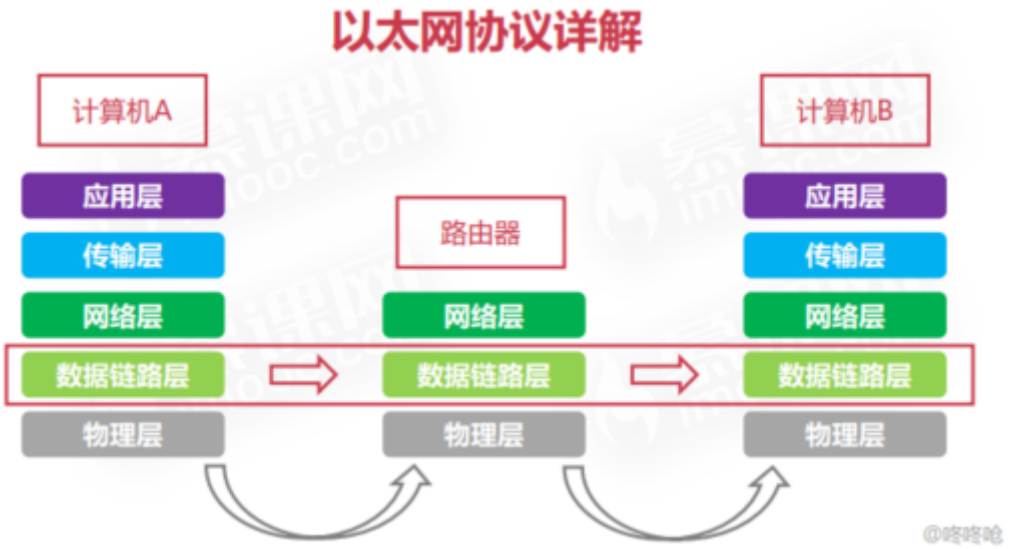
封装成帧

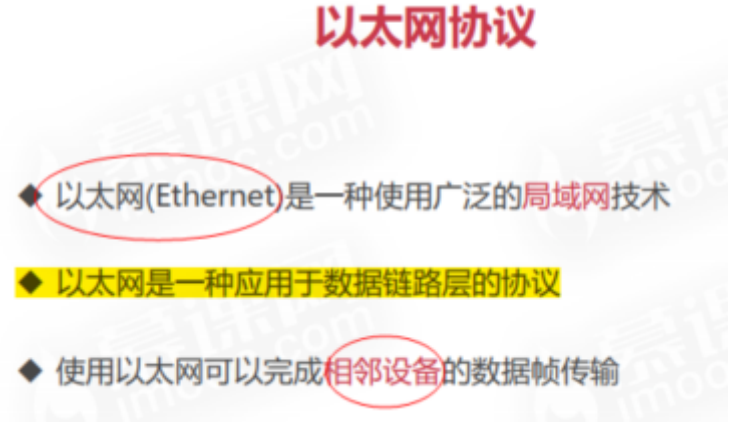
4.2 mac (ARP广播)

MAC地址

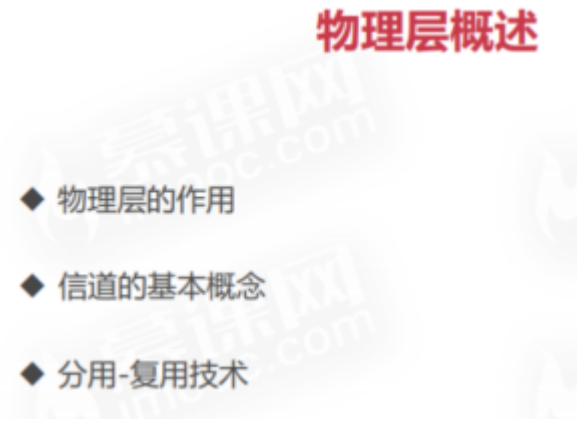
- MAC地址（物理地址、硬件地址）
- 每一个设备都拥有唯一的MAC地址
- MAC地址共48位，使用十六进制表示

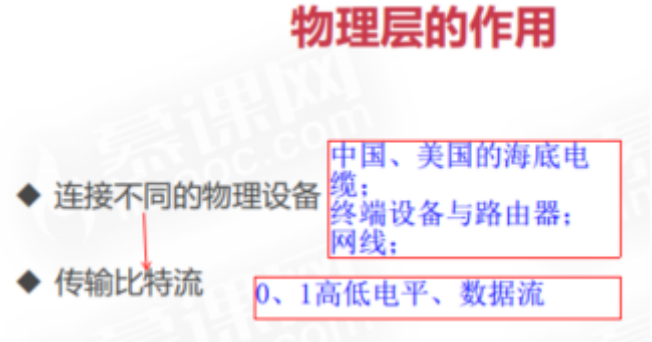
4.3 以太网协议：


以太网协议详解
计算机A
计算机B
应用层
传输层
网络层
数据链路层
物理层
路由器
应用层
传输层
网络层
数据链路层
物理层
@咯咯哈


以太网协议
以太网(Ethernet)是一种使用广泛的局域网技术
以太网是一种应用于数据链路层的协议
使用以太网可以完成相邻设备的数据帧传输

五、物理层


物理层概述
物理层的作用
信道的基本概念
分用-复用技术


物理层的作用
连接不同的物理设备
中国、美国的海底电缆；
终端设备与路由器；
网线；
传输比特流
0、1高低电平、数据流

信道的基本概念

- 信道是往一个方向传送信息的媒体
- 一条通信电路包含一个接收信道和一个发送信道

分用-复用技术

*六、总结：

知识点梳理

层	功能
应用层	向用户**提供一组常用的应用程序**，比如电子邮件、文件传输访问、远程登录。
传输层	提供**应用程序间的通信**。其功能包括：（1）格式化信息流；（2）提供可靠传输。
网络层	负责相邻**计算机之间的通信**，处理数据报、路径、流控、拥塞。
网络接口层	**定义物理介质的各种特性**，处理数据帧。

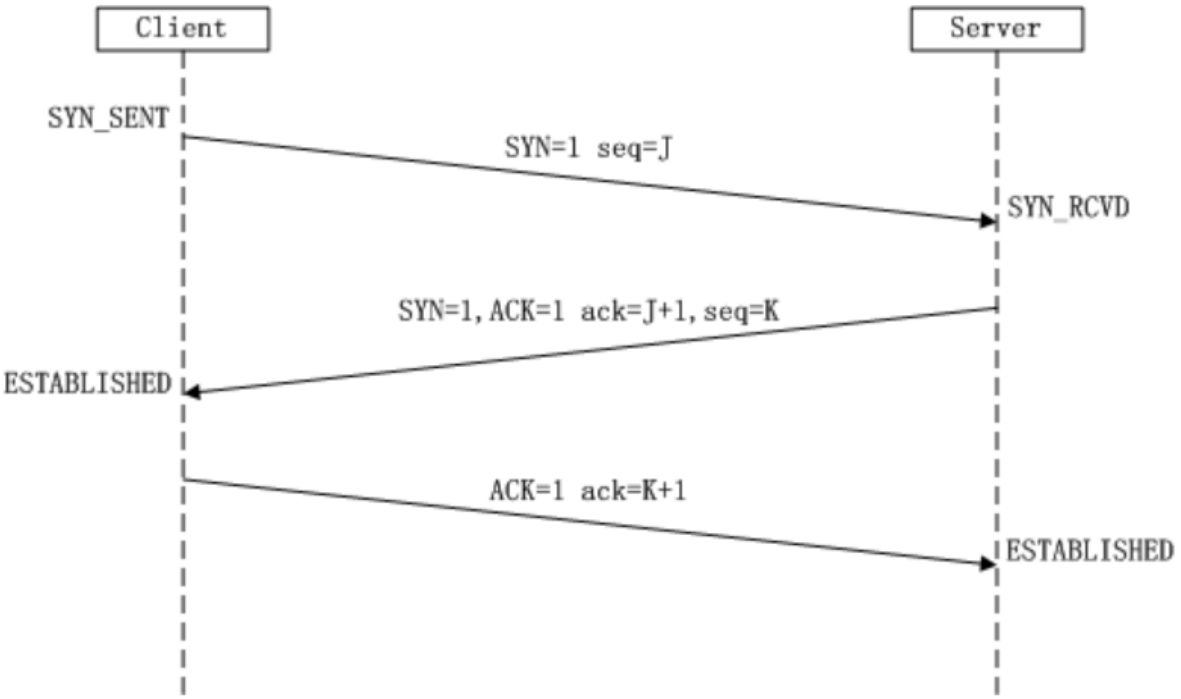
2.TCP的三次握手和四次挥手

2.1 三次握手

“我想给你发数据，可以吗？” （请提供序列号作为起始数据段）SYN：同步序列编号（Synchronize Sequence Numbers）

“可以，你什么时候发？” （已提供序列号）SYN+ACK应答

“我现在就发，你接着吧！” ACK消息响应



2.2 为什么需要三次握手

三次握手是需要发送方和接收方都要确认,自己和对方,接收和发送功能正常

第一次:client:什么都不能确认　　server:client发送正常,server接收正常

第二次:client:client接收,发送正常,server接收发送正常;

　　server:client发送正常,server接收正常

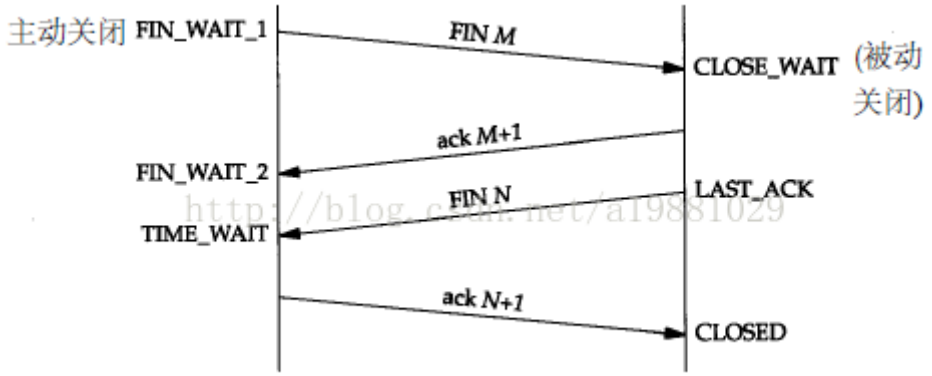
第三次:server:client接收,发送正常,server接收发送正常.

2.3 4次挥手

一般是server主动关闭

- 主动关闭连接的一方，调用close()；协议层发送FIN包
- 被动关闭的一方收到FIN包后，协议层回复ACK；然后被动关闭的一方，进入CLOSE_WAIT状态，主动关闭的一方等待对方关闭，则进入FIN_WAIT_2状态；此时，主动关闭的一方会等待被动关闭一方的应用程序调用close操作

- 被动关闭的一方在完成所有数据发送后，调用close()操作；此时，协议层发送FIN包给主动关闭的一方，等待对方的ACK，被动关闭的一方进入LAST_ACK状态；
- 主动关闭的一方收到FIN包，协议层回复ACK；此时，主动关闭连接的一方，进入TIME_WAIT状态；而被动关闭的一方，进入CLOSED状态
- 等待2MSL时间，主动关闭的一方，结束TIME_WAIT，进入CLOSED状态



2.4 TIME_WAIT问题

https://blog.csdn.net/fanren224/article/details/89849276

MSL 是Maximum Segment Lifetime英文的缩写，中文可以译为“报文最大生存时间”，他是任何报文在网络上存在的最长时间，

TIME_WAIT 存在是为了处理因为网络堵塞问题,主动方最后的ack可能未送达,处理被动方发来的FIN

占用内存,CPU和端口

解决方法:

通过优化相关的内核参数来解决这个问题

1.开启tw重用

2.开启tw回收 (更快速的回收)

1和2违背tcp原理,不推荐使用

1、客户端改用长连接

需要客户端的改动比较大，但能彻底解决问题，高并发的场景下，长连接的性能也明显好于短连接。

2、增加客户端的个数，避免在2MSL时间内使用到重复的端口

能够降低出问题概率，但需要增加成本，性价比不高。

3、降低net.ipv4.tcp_max_tw_buckets(有风险)

能够降低出问题概率，降低的程度视修改的参数值而定，设置为0可以完全解决。此方法在网络状况不好的情况下有风险，一般内网低延迟的网络风险不大。

4、客户端在断开连接时，不用quit的方式退出，直接发FIN或者RST

能够彻底解决问题，需要修改客户端底层库，有一定风险。

5、修改linux内核减小MSL时间

能够降低出问题的概率，需要修改linux内核，难度和风险都较大。

3.TCP如何保证可靠传输

1.大的应用数据被分割为多个数据块.并且编号保证顺序

2.首部校验和,如果校验和有问题,tcp会抛弃该包,并且不确认收到该包

3.丢弃重复数据包

4.流量控制

5.拥塞控制

6.超时重传

4.流量控制和拥塞控制

4.1 流量控制

TCP利用滑动窗口实现流量控制.

流量控制是为了控制发送方发送速率,保证接收方来得及接收.

接收方发送的确认报文中的窗口字可以用来控制发送窗口大小,进而影响发送方的发送速率.如果窗口字段为0,则发送方不能发送数据

4.2 拥塞控制

流量控制是点对点的控制,拥塞控制是一个全局性的控制

为了进行拥塞控制,TCP发送方要维持一个**拥塞窗口(cwnd)**.拥塞窗口的大小取决于网络的拥塞程度,并且动态变化.

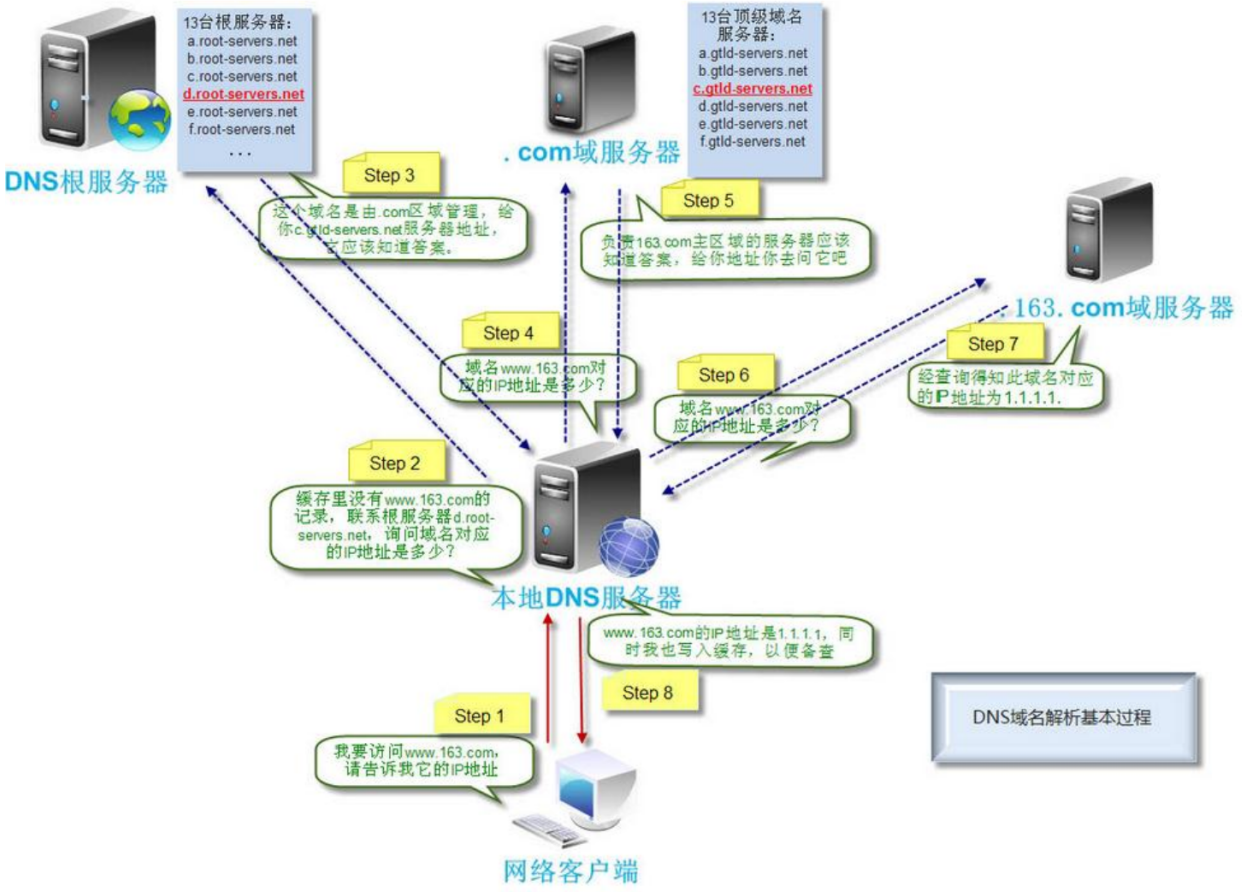
方式：慢启动

拥塞避免(当出现拥塞的时候,重新进入慢启动,拥塞窗口重置)

解决方法：快速重传,快速恢复

5. DNS

1.dns查询过程



- 网络客户端就是我们平常使用的电脑，打开浏览器，输入一个域名。比如输入www.163.com，这时，你使用的电脑会发出一个DNS请求到本地DNS服务器。本地DNS服务器一般都是你的网络接入服务器商提供，比如中国电信，中国移动。
- 查询www.163.com的DNS请求到达本地DNS服务器之后，本地DNS服务器会首先查询它的缓存记录，如果缓存中有此条记录，就可以直接返回结果。如果没有，本地DNS服务器还要向DNS根服务器进行查询。
- 根DNS服务器没有记录具体的域名和IP地址的对应关系，而是告诉本地DNS服务器，你可以到域服务器上去继续查询，并给出域服务器的地址。
- 本地DNS服务器继续向域服务器发出请求，在这个例子中，请求的对象是.com域服务器。.com域服务器收到请求之后，也不会直接返回域名和IP地址的对应关系，而是告诉本地DNS服务器，你的域名的解析服务器的地址。
- 最后，本地DNS服务器向域名的解析服务器发出请求，这时就能收到一个域名和IP地址对应关系，本地DNS服务器不仅要把IP地址返回给用户电脑，还要把这个对应关系保存在缓存中，以备下次别的用户查询时，可以直接返回结果，加快网络访问。

从查询的步骤来看,域名是有层级的。

> 举个例子来说, `www.tungee.com` 真正的域名是 `www.tungee.com.root`
>
> 因为所有的域名的根域名都是 `.root` 所以默认都是省掉的

根域名的下一级叫做'顶级域名'(top-level domain),比如 `.com` `.net`

再下一级则是"次级域名"(second-level domain),比如 `www.tungee.com` 里面的 `tungee`,这级域名用户是可以注册的。

再下一级是主机名(host),比如 `www.tungee.com` 里面的www,又称为"三级域名",这是用户在自己的域里面为服务器分配的名称,是用户可以任意分配的。

> 主机名.次级域名.顶级域名.根域名
>
> `www.tungee.com.root`

2.dns劫持

在dns解析过程中，有哪一环节出现问题的话，都可能会导致DNS解析错误，导致客户端（浏览器）得到一个假的ip地址，从而引导用户访问到这个冒名顶替，恶意的网站。

1.本机DNS劫持

攻击者通过某些手段使用户的计算机感染上木马病毒，或者恶意软件之后，恶意修改本地DNS配置，比如修改本地hosts文件，缓存等

2. 路由DNS劫持

很多用户默认路由器的默认密码，攻击者可以侵入到路由管理员账号中，修改路由器的默认配置

1.加强本地计算机病毒检查，开启防火墙等，防止恶意软件，木马病毒感染计算机

2.改变路由器默认密码，防止攻击者修改路由器的DNS配置指向恶意的DNS服务器

6.长连接和心跳保活机制

https://blog.csdn.net/H542723151/article/details/83716167

短连接:

短连接多用于操作频繁，点对点的通讯，而且连接数不能太多的情况。每个TCP连接的建立都需要三次握手，每个TCP连接的断开要四次挥手。适用于并发量大，但是每个用户又不需频繁操作的情况。

长连接:

长连接与持久连接本质上非常的相似，持久连接侧重于HTTP应用层，特指一次请求结束之后，服务器会在自己设置的keepalivetimeout时间到期后才关闭已经建立的连接.长连接则是client方与server方先建立连接，连接建立后不断开，然后再进行报文发送 和接收，直到有一方主动关闭连接为止。

长连接的适用场景也非常的广泛：

- 监控系统：后台硬件热插拔、LED、温度、电压发生变化等;
- IM应用：收发消息的操作;
- 即时报价系统：例如股市行情push等;
- 推送服务：各种App内置的push提醒服务;

长连接的好处:

对于客户端而言，使用TCP长连接来实现业务的好处在于：在当前连接可用的情况下，每一次请求都只是简单的数据发送和接受，免去了DNS解析，连接建立，TCP慢启动等时间，大大加快了请求的速度，同时也有利于接收服务器的实时消息。

使用TCP长连接的业务场景下，保持长连接的可用性非常重要。如果长连接无法很好地保持，在连接已经失效的情况下继续发送请求会导致迟迟收不到响应直到超时，又需要一次连接建立的过程，其效率甚至还不如直接使用短连接。而连接保持的前提必然是检测连接的可用性，并在连接不可用时主动放弃当前连接并建立新的连接。

心跳保活:

App实现长连接保活的方式通常是采用应用层心跳，通过心跳包的超时和其他条件(网络　　切换)来执行重连操作。心跳一般是指某端(绝大多数情况下是客户端)每隔一定时间向对端发送自定义指令，以判断双方是否存活，因其按照一定间隔发送，类似于心跳，故被称为心跳指令。

心跳过于频繁会带来耗电和耗流量的弊病，心跳频率过低则会影响连接检测的实时性。业内关于心跳时间的设置和优化，主要基于如下几个因素：

Keep-Alive可否实现保活?

1.HTTP中的Keep-Alive

标识确认此连接是长连接,不保存活

实现HTTP/1.0 keep-alive连接的客户端可以通过包含Connection:Keep-Alive首部请求将一条连接保持在打开状态，如果服务器愿意为下一条请求将连接保持在打开状态，就在响应中包含相同的首部。如果响应中没有Connection: Keep-Alive首部，客户端就认为服务器不支持keep-alive，会在发回响应报文之后关闭连接。HTTP/1.1以后Keep-Alive是默认打开的。

2.TCP中的Keep-Alive

TCP协议的实现中，提供了KeepAlive报文，用来探测连接的对端是否存活.虽然TCP提供了KeepAlive机制，但是并不能替代应用层心跳保活。原因主要如下：

> (1) Keep Alive机制开启后，TCP层将在定时时间到后发送相应的KeepAlive探针以确定连接可用性。默认时间为7200s(两小时)，失败后重试10次，每次超时时间75s。显然默认值无法满足移动网络下的需求；
> (2) 即便修改了(1)中的默认值，也不能很好的满足业务需求。TCP的KeepAlive用于检测连接的死活而不能检测通讯双方的存活状态。比如某台服务器因为某些原因导致负载超高，无法响应任何业务请求，但是使用TCP探针则仍旧能够确定连接状态，这就是典型的连接活着但业务提供方已死的状态，对客户端而言，这时的最好选择就是断线后重新连接其他服务器，而不是一直认为当前服务器是可用状态，一直向当前服务器发送些必然会失败的请求。
> (3) socks代理会让Keep Alive失效。socks协议只管转发TCP层具体的数据包，而不会转发TCP协议内的实现细节的包。所以，一个应用如果使用了socks代理，那么TCP的KeepAlive机制就失效了。
> (4) 部分复杂情况下Keep Alive会失效，如路由器挂掉，网线直接被拔除等；

因此，KeepAlive并不适用于检测双方存活的场景，这种场景还得依赖于应用层的心跳。应用层心跳也具备着更大的灵活性，可以控制检测时机，间隔和处理流程，甚至可以在心跳包上附带额外信息。

3.为什么TCPkeepLive不能替代应用层心跳包(简单总结版):

(1)默认参数不满足要求

(2)TCP的keepIive只能检测连接的状态,无法确定连接双方的状态(例如服务器服务挂掉了,但是连接保持,这是只能通过http心跳包才能检测)

(3)socks代理会让keepLive失效