第7章 网络编程面试题汇总

7.1 请你描述TCP三次握手与四次挥手的过程与意义

详细讲解

享学课堂移动互联网系统课程：设计思想解决开源框架《Java原生网络编程1》

这道题想考察什么？

这个问题属于网络体系中的基础理论知识，对于这种类型的问题如果没有一个清晰的认识，那会让你在掌握一些"高大上"技术的时没有支撑，也难以把整体框架理顺。比如Http、RTSP 、RTMP等被广泛运用的应用层协议都是基于TCP来实现的。所以被问到这个问题并不稀奇。

考察的知识点

网络的基础知识

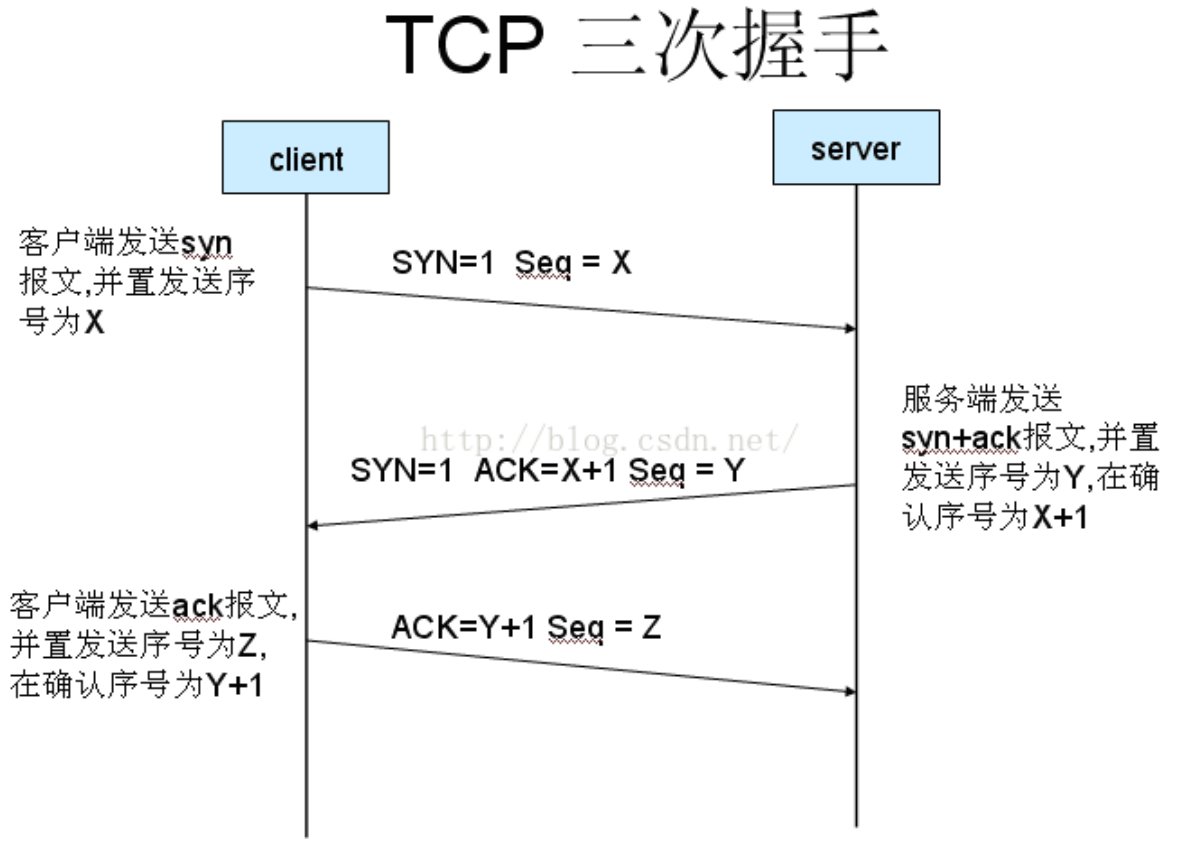
考生如何回答

TCP/IP协议定义了计算机在网络中如何发送数据、数据格式如何定义、发出消息后在网络中如何寻址找到目标计算机，最后目标计算机又如何检验收到消息的正确性、对数据拆解最后得到消息内容的一套处理标准。

有了这些标准后生产提供TCP/IP服务的软件商家就有了一套统一的规范，只要遵循这个规范去实现自己的软件功能。

三次握手

在进行业务通信前，必须建立好连接，而TCP/IP连接的建立需要经过三次握手的过程。其过程如下图：



1. 第一次握手：建立连接时，客户端发送syn包（syn=j）到服务器，并进入SYN_SENT状态，等待服务器确认；SYN：同步序列编号（Synchronize Sequence Numbers）。
2. 第二次握手：服务器收到syn包，必须确认客户的SYN（ack=j+1），同时自己也发送一个SYN包（syn=k），即SYN+ACK包，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包，向服务器发送确认包ACK(ack=k+1），此包发送完毕，客户端和服务器进入ESTABLISHED（TCP连接成功）状态，完成三次握手。

为什么要三次握手？

全双工通信

三次握手是确定通信双方通讯线路是全双工的最小次数，全双工通信是指：通信的双方可以同时发送和接收信息 。

> 正如双方电话通话：
>
> A：喂，能听到吗？
>
> B：可以
>
> 此时如果A没有反馈，B无法确定A是否能够接收数据。

保证可靠性

另外TCP是可靠传输协议，保证通信的可靠性的手段中包含序列号与确认应答机制。

- 序列号：TCP传输时将每个字节的数据都进行了编号，保证数据的有序性与可靠性（当接收到的数据总少了某个序号的数据时，能马上知道 ）；
- 确认应答：TCP传输的过程中，每次接收方收到数据后，都会对传输方进行确认应答。也就是发送ACK报文。这个ACK报文当中带有对应的确认序列号，告诉发送方，接收到了哪些数据，下一次的数据从哪里发。

而三次握手的同时也能确定通信双方的初始序列号。

1. C --> S SYN my sequence number is X
2. S <-- C ACK your sequence number is X my sequence number is Y
3. C --> S ACK your sequence number is Y

如果C 未确认收到 B 的。也就是说，只有 C 发送给 S 的包都是可靠的， 而 S 发送给 C 的则不是，所以这不是可靠的连接。

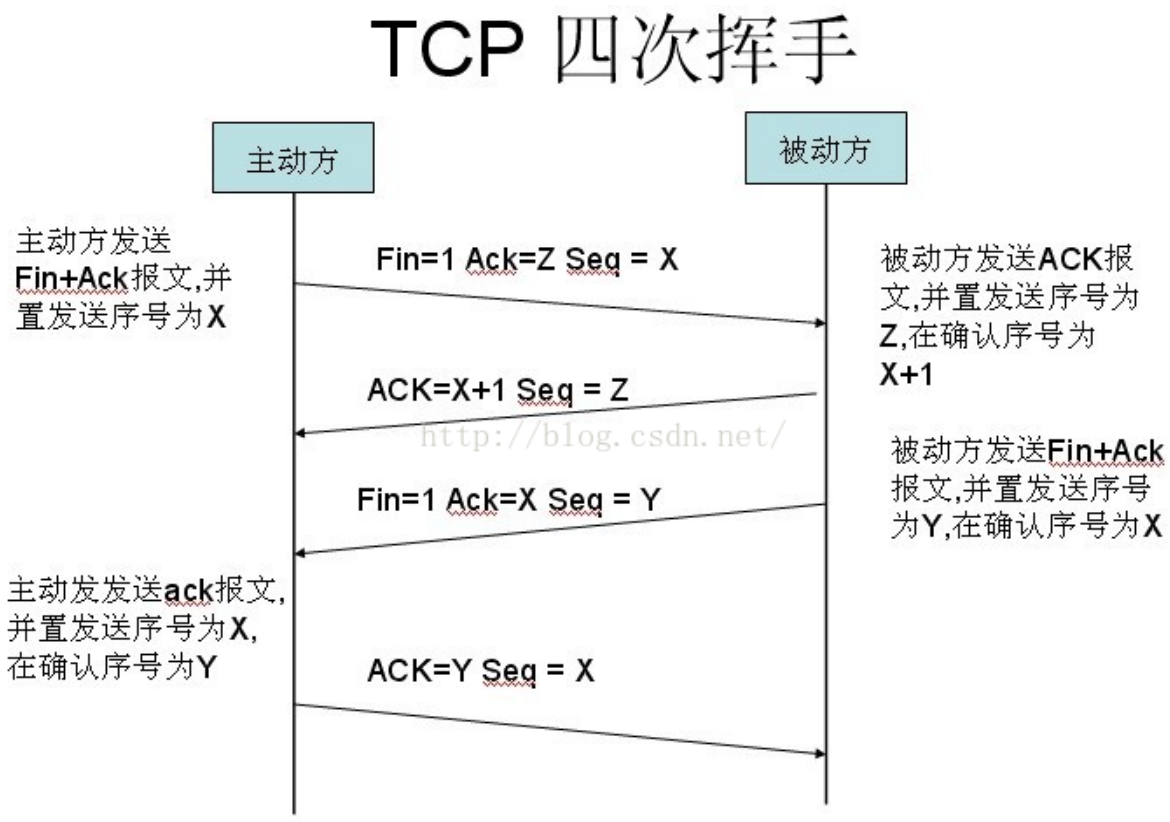
避免资源浪费

除此之外，第一次握手：客户端发送连接请求消息到服务端，服务端收到信息后需要进行第二次握手：应答告知客户端已经接收连接请求。而服务端发送出去的应答消息，需要等客户端第三次握手响应后，才能确定此次连接为有效连接。

若客户端发出去的第一个连接请求由于某些原因在网络节点中滞留了导致延迟，直到客户端放弃连接后的某个时间点才到达服务端，这是一个早已失效的报文，但是此时服务端仍然认为这是客户端的建立连接请求第一次握手，于是服务端第二次握手回应了客户端。如果没有第三次握手，那么到这里，连接就建立了，但是此时客户端并没有任何数据要发送，会让服务端空等，造成资源浪费。

四次挥手

在完成数据交互之后，如果选择关闭连接，以回收资源，则完成四次挥手来进行“和平分手”。过程如下图：



1. 第一次挥手：主动关闭方发送第一个包，其中FIN标志位为1，发送顺序号seq为X。
2. 第二次挥手：被动关闭方收到FIN包后发送第二个包，其中发送顺序号seq为Z，接收顺序号ack为X+1。
3. 第三次挥手：被动关闭方再发送第三个包，其中FIN标志位为1，发送顺序号seq为Y，接收顺序号ack为X。
4. 第四次挥手：主动关闭方发送第四个包，其中发送顺序号为X，接收顺序号为Y。至此，完成四次挥手。

为什么断开连接需要四次挥手？

三次握手是因为建立连接时，ACK和SYN可以放在一个报文里来发送。而关闭连接时，被动关闭方可能还需要发送一些数据后，再发送FIN报文表示同意现在可以关闭连接了，所以它这里的ACK报文和FIN报文多数情况下都是分开发送的。因此断开连接需要4次。

7.2 谈谈你对TCP与UDP的区别是什么的理解（腾讯）

详细讲解

享学课堂移动互联网系统课程：设计思想解决开源框架《Java原生网络编程4》

这道题想考察什么？

在平时的开发中大多数情况下都是使用Http/Https协议完成与服务端的网络交互，而Http底层是基于TCP的可靠连接。而TCP/IP 中有两个具有代表性的传输层协议，分别是 TCP 和 UDP。掌握二者的区别能够让我们在不同的场景中合理的选择最优的传输协议。

考察的知识点

网络的基础知识

考生如何回答

TCP/IP 是互联网相关的各类协议簇的总称，比如：TCP，UDP，IP，FTP，HTTP，ICMP，SMTP 等都属于 TCP/IP 协议簇 。之所以命名为TCP/IP协议，因为TCP、IP协议是两个很重要的协议，就用他两命名了。

UDP

UDP协议全称是用户数据报协议（**User Data Protocol**），在网络中它与TCP协议一样用于处理数据包，是一种无连接的协议。在OSI模型中，在第四层—传输层，处于IP协议的上一层。UDP有不提供数据包分组、组装和不能对数据包进行排序的缺点，也就是说，当报文发送之后，是无法得知其是否安全完整到达的。

它有以下几个特点：

- 面向无连接

  首先 UDP 是不需要和 TCP一样在发送数据前进行三次握手建立连接的，想发数据就可以开始发送了。并且也只是数据报文的搬运工，不会对数据报文进行任何拆分和拼接操作。

  具体来说就是：

  - 在发送端，应用层将数据传递给传输层的 UDP 协议，UDP 只会给数据增加一个 UDP 头标识下是 UDP 协议，然后就传递给网络层了
  - 在接收端，网络层将数据传递给传输层，UDP 只去除 IP 报文头就传递给应用层，不会任何拼接操作
- 有单播，多播，广播的功能

  UDP 不止支持一对一的传输方式，同样支持一对多，多对多，多对一的方式，也就是说 UDP 提供了单播，多播，广播的功能。

- UDP是面向报文的

  发送方的UDP对应用程序交下来的报文，在添加首部后就向下交付IP层。UDP对应用层交下来的报文，既不合并，也不拆分，而是保留这些报文的边界。因此，应用程序必须选择合适大小的报文

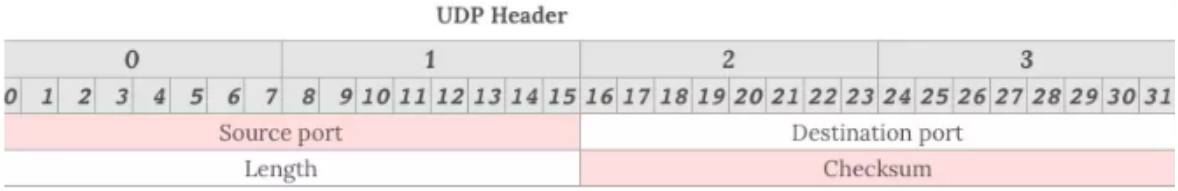
- 不可靠性

  首先不可靠性体现在无连接上，UDP只会把想发的数据报文一股脑的丢给对方，并不在意数据有无安全完整到达。 通信都不需要建立连接，想发就发，这样的情况肯定不可靠。

  并且收到什么数据就传递什么数据，并且也不会备份数据，发送数据也不会关心对方是否已经正确接收到数据了。

  再者网络环境时好时坏，但是 UDP 因为没有拥塞控制，一直会以恒定的速度发送数据。即使网络条件不好，也不会对发送速率进行调整。这样实现的弊端就是在网络条件不好的情况下可能会导致丢包，但是优点也很明显，在某些实时性要求高的场景（比如电话会议）就需要使用 UDP 而不是 TCP。

- 头部开销小，传输数据报文时是很高效的。



UDP 头部包含了以下几个数据：

1. 两个十六位的端口号，分别为源端口（可选字段）和目标端口
2. 整个数据报文的长度
3. 整个数据报文的检验和（IPv4 可选 字段），该字段用于发现头部信息和数据中的错误

因此 UDP 的头部开销小，只有八字节，相比 TCP 的至少二十字节要少得多，在传输数据报文时是很高效的。

TCP

TCP协议全称是传输控制协议（**Transmission Control Protocol** ），是一种面向连接的、可靠的、基于字节流的传输层通信协议，由 IETF 的RFC 793定义。TCP 是面向连接的、可靠的流（不间断的数据结构）协议。

> TCP连接过程见：描述TCP三次握手与四次挥手的过程与意义

它有以下几个特点：

- 面向连接

  面向连接，是指发送数据之前必须在两端建立连接。建立连接的方法是“三次握手”，这样能建立可靠的连接。建立连接，是为数据的可靠传输打下了基础。

- 仅支持单播传输

  每条TCP传输连接只能有两个端点，只能进行点对点的数据传输，不支持多播和广播传输方式。

- 面向字节流

  TCP不像UDP一样那样一个个报文独立地传输，而是在不保留报文边界的情况下以字节流方式进行传输。

- 可靠传输

  对于可靠传输，判断丢包，误码靠的是TCP的段编号以及确认号。TCP为了保证报文传输的可靠，就给每个包一个序号，同时序号也保证了传送到接收端实体的包的按序接收。然后接收端实体对已成功收到的字节发回一个相应的确认(ACK)；如果发送端实体在合理的往返时延(RTT)内未收到确认，那么对应的数据（假设丢失了）将会被重传。

- 提供拥塞控制

  当网络出现拥塞的时候，TCP能够减小向网络注入数据的速率和数量，缓解拥塞

- TCP提供全双工通信

  TCP允许通信双方的应用程序在任何时候都能发送数据，因为TCP连接的两端都设有缓存，用来临时存放双向通信的数据。当然，TCP可以立即发送一个数据段，也可以缓存一段时间以便一次发送更多的数据段（最大的数据段大小取决于MSS）

TCP和UDP的比较

	UDP	TCP
是否连接	无连接	面向连接
是否可靠	不可靠传输，不使用流量控制和拥塞控制	可靠传输，使用流量控制和拥塞控制
连接对象个数	支持一对一，一对多，多对一和多对多交互通信	只能是一对一通信
传输方式	面向报文	面向字节流
首部开销	首部开销小，仅8字节	首部最小20字节，最大60字节
适用场景	适用于实时应用（IP电话、视频会议、直播等）	适用于要求可靠传输的应用，例如文件传输

总结

- TCP向上层提供面向连接的可靠服务 ，UDP向上层提供无连接不可靠服务。
- 虽然 UDP 并没有 TCP 传输来的准确，但是也能在很多实时性要求高的地方有所作为
- 对数据准确性要求高，速度可以相对较慢的，可以选用TCP

7.3 谈谈你对TCP 流量控制与拥塞控制的理解（oppo）

详细讲解

享学课堂移动互联网系统课程：设计思想解决开源框架《Java原生网络编程4》

这道题想考察什么？

网络编程，TCP原理

考察的知识点

TCP流量控制与拥塞控制

考生应该如何回答

TCP的拥塞控制和流量控制虽然采取的动作很相似，但拥塞控制与网络的拥堵情况相关联，而流量控制与接收方的缓存状态相关联，是针对完全不同的问题而采取的措施 。两者从不同的方面保证TCP协议可靠性。

流量控制

双方在通信的时候，发送方的速率与接收方的速率是不一定相等，如果发送方的发送速率太快，会导致接收方处理不过来，这时候接收方只能把处理不过来的数据存在缓存区里。

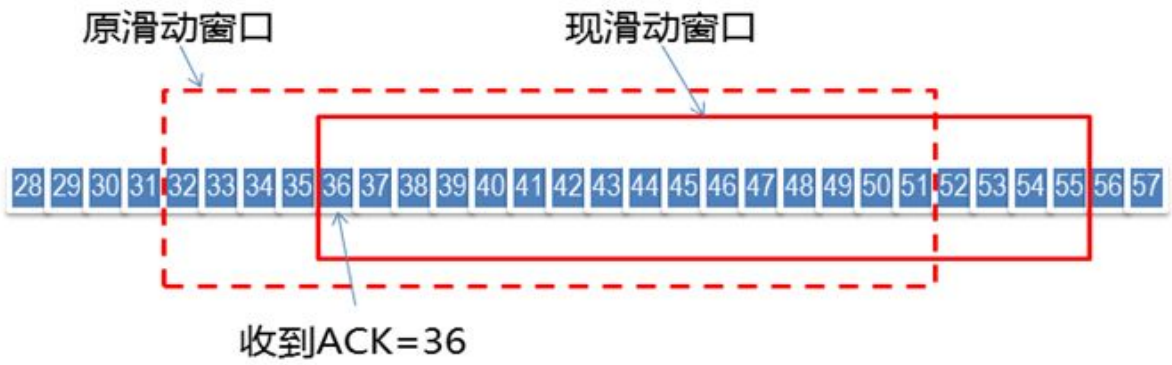
如果缓存区满了发送方还在疯狂着发送数据，接收方只能把收到的数据包丢掉，而流量控制就是控制发送者的发送速度从而使接收者来得及接收，防止丢失数据包的。

假设没有流量控制，发送端根据自己的实际情况发送数据，如果发送的速度太快，导致接收端的接收缓冲区很快填满了，此时发送端如果继续发送数据，接收端处理不过来，这时接收端就会把本来应该接收的数据丢弃，这会触发发送端的重发机制，从而导致网络流量的无端浪费。

滑动窗口

在TCP头中有一个Window字段，这个字段代表了接收端告诉发送端自己缓冲区还有多少剩余空间可以接收数据。TCP 利用**滑动窗口**实现流量控制的机制， 而滑动窗口大小就是通过TCP头部的Window字段来通知发送方。

接收端会在确认应答发送ACK报文时，将自己的即时窗口大小填入，并跟随ACK报文一起发送过去。而发送方根据ACK报文里的窗口大小的值进而改变自己的发送速度。



假设我们窗口大小为20（32-51），发送方发送32-51序列包后，接收到ACK=36，表示 接收方只接收到了32-35，下一次接收方期望接收到的是36序列号的包。如果接收方给到的ACK中窗口大小仍为20，发送方窗口滑动，36-55则是发送方可以发送的包。

这就是滑动窗口的工作机制，发送方在发送过程中始终保持着一个发送窗口，只有落在发送窗口内的帧才允许被发送；发送方收到ACK 计算获得接收方窗口大小之后，便会调整自己的发送速率，也就是调整自己发送窗口的大小实现流量控制。**但是当发送方收到接收窗口的大小为0时，发送方就会停止发送数据！**

零窗口

当发送方停止发送数据后，该怎样才能知道自己可以继续发送数据？

我们可以采用这样的策略：当接收方处理好数据，接受窗口 win > 0 时，接收方发个通知报文去通知发送方，告诉他可以继续发送数据了。当发送方收到窗口大于0的报文时，就继续发送数据。

不过这时候可能会遇到一个问题，如果发送端在重发超时的时间内都没有收到窗口更新的通知或者窗口更新的包丢失了，这时候就会引发一个问题：接收方发了通知报文后，继续等待发送方发送数据，而发送方则在等待接收方的通知报文，此时双方会陷入一种僵局。

为了解决这种问题，TCP为每一个连接设有一个**持续计时器** ：当发送方收到接受窗口 win = 0 时，这时发送方停止发送报文，并且同时开启一个定时器，每隔一段时间就发个测试报文去询问接收方，打听是否可以继续发送数据了，如果可以，接收方就告诉他此时接受窗口的大小；如果接受窗口大小还是为0，则发送方再次刷新启动定时器。

拥塞控制

流量控制是接收方怕发送方发的太快，使得自己来不及处理。而拥塞控制的对象是网络，怕发送方发的太快，造成网络拥塞，使得网络来不及处理，是一个全局性的过程。

拥塞控制就是防止过多的数据注入网络中，这样可以使网络中的路由器或链路不致过载。 拥塞是一个动态问题，我们没有办法用一个静态方案去解决，从这个意义上来说，拥塞是不可避免的。就好像上下班高峰期经常堵车，为了不让交通瘫痪，交警会去现场指挥，采用动态的方式对车辆进行限制，根据实际情况，慢慢放行。

比如主机A给主机B传输数据。

两台主机在传输数据包的时候，如果发送方迟迟没有收到接收方反馈的ACK，那么发送方就会认为它发送的数据包丢失了，进而会重新传输这个丢失的数据包。

然而实际情况有可能此时有太多主机正在使用信道资源，导致网络拥塞了，而A发送的数据包被堵在了半路，迟迟没有到达B。这个时候A误认为是发生了丢包情况，会重新传输这个数据包。

结果就是不仅浪费了信道资源，还会使网络更加拥塞。因此，我们需要进行拥塞控制。TCP的拥塞控制通过：慢启动、拥塞避免、快重传与快恢复完成。

慢启动

慢启动算法为TCP发送方新增的一个拥塞窗口 （ cwnd ）。对应到流量控制，发送方会根据拥塞窗口和滑动窗口的最小值作为发送上限。

当新建连接时，发送方不了解网络的情况，cwnd（拥塞窗口）初始化比较小的值，RFC建议2-4个MSS，具体视MSS的大小而定。

> **MSS：Maximum Segment Size，TCP一次传输发送的最大数据段长度。**

假设初cwnd为1个MSS。发送端开始按照拥塞窗口大小发送数据，如果被ACK，下次就发送2个。如果还是收到了ACK，则发送4个，接着8个……。这样cwnd的值就随着网络往返时间(Round Trip Time,RTT)呈指数级增长，事实上，慢启动的速度一点也不慢，只是它的起点比较低一点而已。

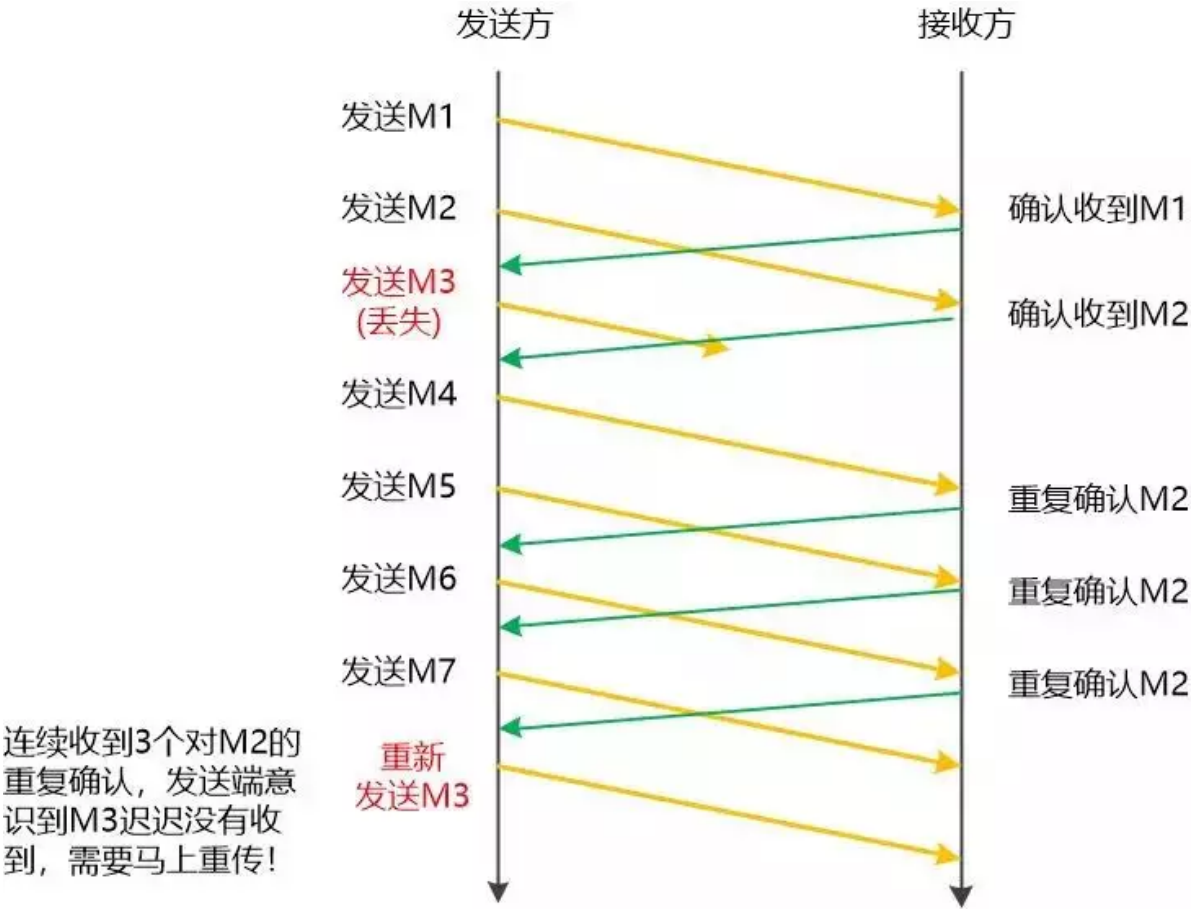
拥塞避免

从慢启动可以看到，cwnd可以很快的增长上来，从而最大程度利用网络带宽资源，但是cwnd不能一直这样无限增长下去，一定需要某个限制。TCP使用了一个叫慢启动门限(ssthresh)的变量，当cwnd超过该值后，慢启动过程结束，进入拥塞避免阶段。拥塞避免的主要思想是加法增大，也就是cwnd的值不再指数级往上升，开始加法增加。此时当窗口中所有的报文段都被确认时，cwnd的大小加1(此处加1指的是加1个MSS，下文同样如此)，cwnd的值就随着RTT开始线性增加，这样就可以避免增长过快导致网络拥塞，慢慢的增加调整到网络的最佳值。

快重传与快恢复

进入拥塞避免之后，最终还是会碰到拥塞点，发送方此时一直得不到接收端的确认。因此TCP在发送一个数据以后就开启一个计时器， 在一定时间内如果没有得到发送数据报的ACK报文，那么就重新发送数据，直到发送成功为止，这就是超时重传。

而快重传算法首先要求接收方每收到一个失序的报文段就立即发出重复确认。比如A给B发送M1, M2, M3, M4, M5，如果B收到了M1, M2, M4，M5；M3并没有接收到，那么在B接到M4时就会发送一次M2的ACK，接到M5，又会发送一次M2的ACK，这样重复确认M2意在告诉A，M3还没收到，可能是丢失了。



当A连续收到了三个确认M2的ACK，若M3超时事件还没发生，此时A也会假定M3丢失了，这个时候A就不必等待M3设置的计时器到期了，而是进行**快速重传**。

而**快速恢复**是在上述的**快速重传**后添加的，快速重传和快速恢复算法会同时使用，快恢复会：

1.当收到3个重复M2的ACK时，为了预防网络发生拥塞，把ssthresh设置为cwnd的一半，把cwnd设置为ssthresh的值加3，重传M3

> 快速恢复的思想是"数据包守恒"原则，即同一个时刻在网络中的数据包数量是恒定的，只有当"老"数据包离开了网络后，才能向网络中发送一个"新"的数据包，如果发送方收到一个重复的ACK，那么根据TCP的ACK机制就表明有一个数据包离开了网络，收到3个重复的ACK，表明有3个"老"的数据包离开了网络，因此此处加3。

2.再收到重复的ACK时，cwnd增加1。

3.当收到M4（新数据包）的ACK时，把cwnd设置为第一步中的ssthresh的值

通过新数据包的ACK(M4)确认了新的数据，说明从重复ACK时的数据(M3)已收到，该恢复过程结束，可以回到恢复之前的拥塞避免状态。

7.4 谈谈你对Http与Https的关系理解

详细讲解

享学课堂移动互联网系统课程：设计思想解决开源框架《Java原生网络编程4》

这道题想考察什么？

1. 是否了解互联网网络协议？

考察的知识点

1. Http协议的相关知识
2. Https协议的相关知识

考生应该如何回答

Http协议

HTTP（HyperText Transfer Protocol：超文本传输协议）是一种用于分布式、协作式和超媒体信息系统的应用层协议。 简单来说就是一种发布和接收 HTML 页面的方法，被用于在 Web 浏览器和网站服务器之间传递信息。

HTTP 默认工作在 TCP 协议 80 端口，用户访问网站 http:// 打头的都是标准 HTTP 服务。

HTTP 协议以明文方式发送内容，不提供任何方式的数据加密，如果攻击者截取了Web浏览器 和网站服务器之间的传输报文，就可以直接读懂其中的信息，因此，HTTP协议不适合传输一些敏感信息，比如：信用卡号、密码等支付信息。

Https协议

HTTPS（Hypertext Transfer Protocol Secure：超文本传输安全协议）是一种透过计算机网络进行安全通信的传输协议。HTTPS 经由 HTTP 进行通信，但利用 SSL/TLS 来加密数据包。HTTPS 开发的主要目的，是提供对网站服务器的身份认证，保护交换数据的隐私与完整性。

HTTPS 默认工作在 TCP 协议443端口，它的工作流程一般如以下方式：

1、TCP 三次同步握手

2、客户端验证服务器数字证书

3、DH 算法协商对称加密算法的密钥、hash 算法的密钥

4、SSL 安全加密隧道协商完成

5、网页以加密的方式传输，用协商的对称加密算法和密钥加密，保证数据机密性；用协商的 hash算法进行数据完整性保护，保证数据不被篡改。

加密

HTTPS 解决数据传输安全问题的方案就是使用加密算法，具体来说是混合加密算法，也就是对称加密和非对称加密的混合使用，这里有必要先了解一下这两种加密算法的区别和优缺点。

- 对称加密，顾名思义就是加密和解密都是使用同一个密钥，常见的对称加密算法有 DES、3DES 和 AES 等，其优缺点如下：

  优点：算法公开、计算量小、加密速度快、加密效率高，适合加密比较大的数据。

  缺点：
  1.交易双方需要使用相同的密钥，也就无法避免密钥的传输，而密钥在传输过程中无法保证不被截获，因此对称加密的安全性得不到保证。

2.每对用户每次使用对称加密算法时，都需要使用其他人不知道的唯一密钥，这会使得发收信双方所拥有的钥匙数量急剧增长，密钥管理成为双方的负担。对称加密算法在分布式网络系统上使用较为困难，主要是因为密钥管理困难，使用成本较高。

- 非对称加密
  非对称加密算法需要两个密钥：公开密钥（publickey:简称公钥）和私有密钥（privatekey:简称私钥）。公钥与私钥是一对，如果用公钥对数据进行加密，只有用对应的私钥才能解密。因为加密和解密使用的是两个不同的密钥，所以这种算法叫作非对称加密算法。 非对称加密算法实现机密信息交换的基本过程是：甲方生成一对密钥并将公钥公开，需要向甲方发送信息的其他角色(乙方)使用该密钥(甲方的公钥)对机密信息进行加密后再发送给甲方；甲方再用自己私钥对加密后的信息进行解密。甲方想要回复乙方时正好相反，使用乙方的公钥对数据进行加密，同理，乙方使用自己的私钥来进行解密。

流程

HTTPS 能够加密信息，以免敏感信息被第三方获取，所以很多银行网站或电子邮箱等等安全级别较高的服务都会采用 HTTPS 协议。

1、客户端发起 HTTPS 请求

用户在浏览器里输入一个 https 网址，然后连接到 server 的 端口。

2、服务端的配置
采用 HTTPS 协议的服务器必须要有一套数字证书，可以自己制作，也可以向组织申请，区别就是自己颁发的证书需要客户端验证通过，才可以继续访问，而使用受信任的公司申请的证书则不会弹出提示页面(startssl 就是个不错的选择，有 1 年的免费服务)。

3、传送证书

这个证书其实就是公钥，只是包含了很多信息，如证书的颁发机构，过期时间等等。

4、客户端解析证书

这部分工作是有客户端的TLS来完成的，首先会验证公钥是否有效，比如颁发机构，过期时间等等，如果发现异常，则会弹出一个警告框，提示证书存在问题。

如果证书没有问题，那么就生成一个随机值，然后用证书对该随机值进行加密，就好像上面说的，把随机值用锁头锁起来，这样除非有钥匙，不然看不到被锁住的内容。

5、传送加密信息

这部分传送的是用证书加密后的随机值，目的就是让服务端得到这个随机值，以后客户端和服务端的通信就可以通过这个随机值来进行加密解密了。

6、服务端解密信息

服务端用私钥解密后，得到了客户端传过来的随机值(私钥)，然后把内容通过该值进行对称加密，所谓对称加密就是，将信息和私钥通过某种算法混合在一起，这样除非知道私钥，不然无法获取内容，而正好客户端和服务端都知道这个私钥，所以只要加密算法够彪悍，私钥够复杂，数据就够安全。

7、传输加密后的信息

这部分信息是服务段用私钥加密后的信息，可以在客户端被还原。

8、客户端解密信息

客户端用之前生成的私钥解密服务段传过来的信息，于是获取了解密后的内容，整个过程第三方即使监听到了数据，也束手无策。

缺点

- 在相同网络环境中，HTTPS 相比 HTTP 无论是响应时间还是耗电量都有大幅度上升。
- HTTPS 的安全是有范围的，在黑客攻击、服务器劫持等情况下几乎起不到作用。
- 在现有的证书机制下，中间人攻击依然有可能发生。
- HTTPS 需要更多的服务器资源，也会导致成本的升高。

HTTP 与 HTTPS 区别

- HTTP 明文传输，数据都是未加密的，安全性较差，HTTPS（SSL+HTTP）数据传输过程是加密的，安全性较好。
- 使用 HTTPS 协议需要到 CA（Certificate Authority，数字证书认证机构）申请证书，一般免费证书较少，因而需要一定费用。证书颁发机构如：Symantec、Comodo、GoDaddy 和 GlobalSign 等。
- HTTP 页面响应速度比 HTTPS 快，主要是因为 HTTP 使用 TCP 三次握手建立连接，客户端和服务器需要交换 3 个包，而 HTTPS除了 TCP 的三个包，还要加上 ssl 握手需要的 9 个包，所以一共是 12 个包。
- 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。
  HTTPS 其实就是建构在 SSL/TLS 之上的 HTTP 协议，所以，要比较 HTTPS 比 HTTP 要更耗费服务器资源。

7.5 SSL握手的过程都经历过什么

详细讲解

享学课堂移动互联网系统课程：设计思想解决开源框架《网络协议和常用网络工具》

这道题想考察什么？

SSL协议与网络安全

考察的知识点

网络基础与思考问题的能力

考生如何回答

不使用SSL/TLS的HTTP通信，就是不加密的通信。所有信息明文传播，带来了三大风险。

1. **窃听风险**（eavesdropping）：第三方可以获知通信内容。
2. **篡改风险**（tampering）：第三方可以修改通信内容。
3. **冒充风险**（pretending）：第三方可以冒充他人身份参与通信。

SSL/TLS协议是为了解决这三大风险而设计的，希望达到：

1. 所有信息都是**加密传播**，第三方无法窃听。
2. 具有**校验机制**，一旦被篡改，通信双方会立刻发现。
3. 配备**身份证书**，防止身份被冒充。

互联网是开放环境，通信双方都是未知身份，这为协议的设计带来了很大的难度。而且，协议还必须能够经受所有匪夷所思的攻击，这使得SSL/TLS协议变得异常复杂。

互联网加密通信协议的历史，几乎与互联网一样长。

- 1994年，NetScape公司设计了SSL协议（Secure Sockets Layer）的1.0版，但是未发布。
- 1995年，NetScape公司发布SSL 2.0版，很快发现有严重漏洞。
- 1996年，SSL 3.0版问世，得到大规模应用。
- 1999年，互联网标准化组织ISOC接替NetScape公司，发布了SSL的升级版TLS 1.0版。
- 2006年和2008年，TLS进行了两次升级，分别为TLS 1.1版和TLS 1.2版。最新的变动是2011年TLS 1.2的修订版。

目前，应用最广泛的是TLS 1.0，接下来是SSL 3.0。但是，主流浏览器都已经实现了TLS 1.2的支持。TLS 1.0通常被标示为SSL 3.1，TLS 1.1为SSL 3.2，TLS 1.2为SSL 3.3。

SSL/TLS协议的基本思路是采用公钥加密法，也就是说，客户端先向服务器端索要公钥，然后用公钥加密信息，服务器收到密文后，用自己的私钥解密。

但是，这里有两个问题。

1. **如何保证公钥不被篡改？**

   解决方法：将公钥放在数字证书中。只要证书是可信的，公钥就是可信的。

2. **公钥加密计算量太大，如何减少耗用的时间？**

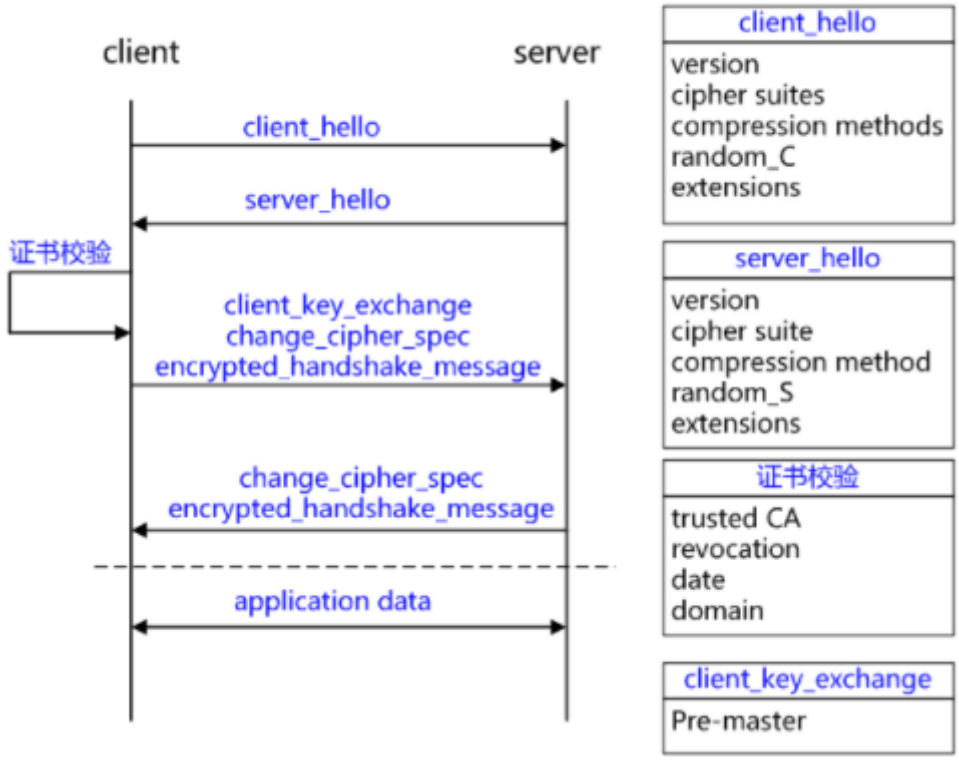
   解决方法：每一次对话（session），客户端和服务器端都生成一个"对话密钥"（session key），用它来加密信息。由于"对话密钥"是对称加密，所以运算速度非常快，而服务器公钥只用于加密"对话密钥"本身，这样就减少了加密运算的消耗时间。

因此，SSL/TLS协议的基本过程是这样的：

1. 客户端向服务器端索要并验证公钥。
2. 双方协商生成"对话密钥"。
3. 双方采用"对话密钥"进行加密通信

上面过程的前两步，又称为"握手阶段"（handshake）。

握手



"握手阶段"涉及四次通信，我们一个个来看。需要注意的是，"握手阶段"的所有通信都是明文的。

ClientHello

首先，客户端（通常是浏览器）先向服务器发出加密通信的请求，这被叫做ClientHello请求。

在这一步，客户端主要向服务器提供以下信息。

1. 支持的协议版本，比如TLS 1.0版。
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。

这里需要注意的是，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数字证书。这就是为什么通常一台服务器只能有一张数字证书的原因。

2006年，TLS协议加入了一个Server Name Indication（SNI）扩展，允许客户端向服务器提供它所请求的域名，主要解决的就是一台服务器只能使用一个证书(一个域名)的缺点，随着服务器对虚拟主机的支持，一个服务器上可以为多个域名提供服务，因此SNI必须得到支持才能满足需求。

SeverHello

服务器收到客户端请求后，向客户端发出回应，这叫做SeverHello。服务器的回应包含以下内容。

1. 确认使用的加密通信协议版本，比如TLS 1.0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. 一个服务器生成的随机数，稍后用于生成"对话密钥"；
3. 确认使用的加密方法，比如RSA公钥加密；
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的身份，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连入自己的网络，就会向正式客户提供USB密钥，里面就包含了一张客户端证书。

客户端回应

客户端收到服务器回应以后，首先验证服务器证书。如果证书不是可信机构颁布、或者证书中的域名与实际域名不一致、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。

如果证书没有问题，客户端就会从证书中取出服务器的公钥。然后，向服务器发送下面三项信息。

1. 一个随机数。该随机数用服务器公钥加密，防止被窃听。
2. 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。
3. 客户端握手结束通知，表示客户端的握手阶段已经结束。这一项同时也是前面发送的所有内容的hash值，用来供服务器校验。

上面第一项的随机数，是整个握手阶段出现的第三个随机数，又称"pre-master key"。有了它以后，客户端和服务器就同时有了三个随机数，接着双方就用事先商定的加密方法，各自生成本次会话所用的同一把"会话密钥"。

至于为什么一定要用三个随机数，来生成"会话密钥"？

"不管是客户端还是服务器，都需要随机数，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来的密钥的随机性。

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hello消息中的随机，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协议不信任每个主机都能产生完全随机的随机数，如果随机数不随机，那么pre master secret就有可能被猜出来，那么仅适用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，一个伪随机可能完全不随机，可是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。"

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。

服务端回应

服务器收到客户端的第三个随机数pre-master key之后，计算生成本次会话所用的"会话密钥"。然后，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，表示服务器的握手阶段已经结束。这一项同时也是前面发送的所有内容的hash值，用来供客户端校验。

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

7.6 谈谈你对Http的post与get请求区别的理解

这道题想考察什么？

对Http协议的掌握，并可能会进一步关联到TCP协议的问题。

考察的知识点

1. Http协议
2. TCP协议

考生应该如何回答

GET和POST是HTTP请求的两种常用的基本方法， 他们之前的区别有:

- GET在浏览器回退时是无害的，而POST会再次提交请求。
- GET产生的URL地址可以被Bookmark，而POST不可以。
- GET请求会被浏览器主动cache，而POST不会，除非手动设置。
- GET请求只能进行url编码，而POST支持多种编码方式。
- 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。
- GET请求在URL中传送的参数是有长度限制的，而POST么有。
- 对参数的数据类型，GET只接受ASCII字符，而POST没有限制。
- GET比POST更不安全，因为参数直接暴露在URL上，所以不能用来传递敏感信息。
- GET参数通过URL传递，POST放在Request body中。

但是站在TCP的角度来看，GET和POST本质上没有区别。HTTP的底层是TCP/IP。所以GET和POST的底层也是TCP/IP，也就是说，GET/POST都是TCP链接。GET和POST能做的事情是一样一样的。你要给GET加上request body，给POST带上url参数，技术上是完全行的通的。之所以出现这些区别是由于HTTP的规定和浏览器/服务器的限制，导致他们在应用过程中体现出一些不同。

7.7 输入一串URL到浏览器都经历过什么？(美团)

详细讲解

享学课堂移动互联网系统课程：设计思想解决开源框架《Java原生网络编程1》

这道题想考察什么？

这是一道经典的面试题，主要考察的是对网络知识的掌握情况。

考察的知识点

DNS解析、TCP/IP模型与通信过程，HTTP/HTTPS等

考生如何回答

总的来说，当你输入在浏览器里输入一个URL到页面加载，发生的顺序如下：

1. DNS解析
2. TCP连接，SSL握手
3. 发送HTTP/HTTPS请求
4. Server处理HTTP请求并返回HTTP报文
5. 浏览器解析并渲染页面
6. HTTP连接断开

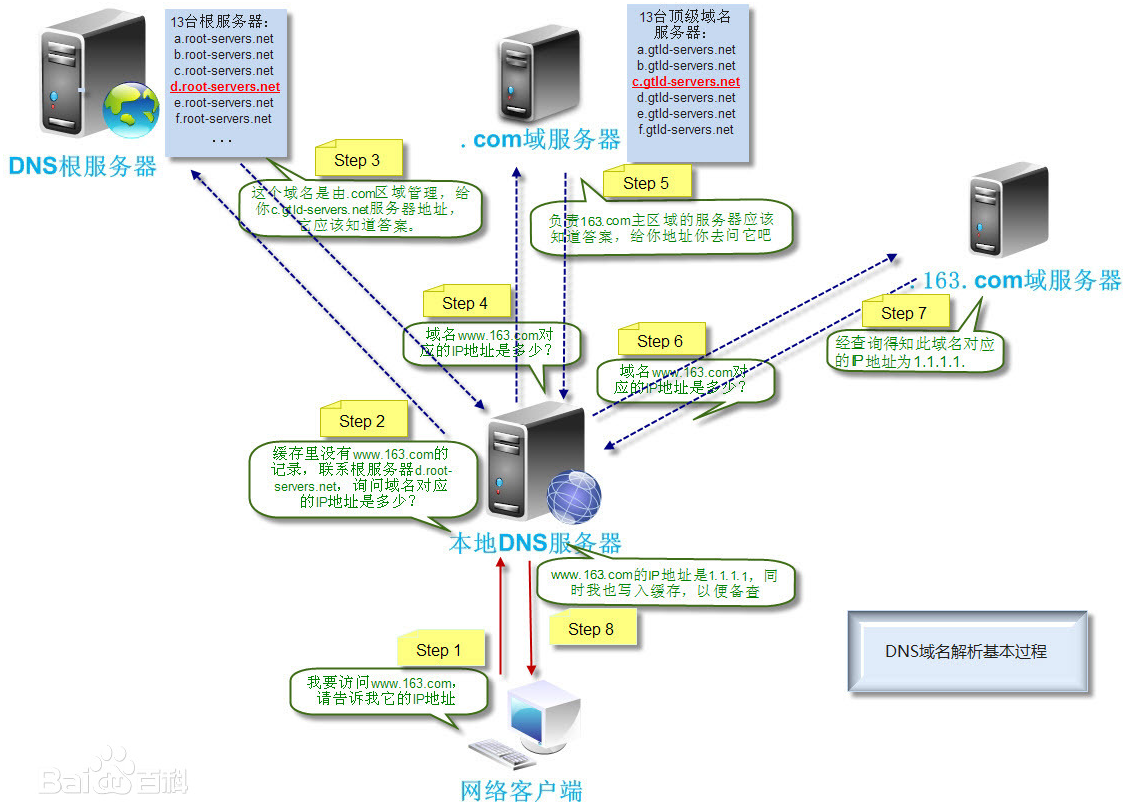
DNS解析

DNS解析流程

DNS解析一般指域名解析。互联网中的地址是数字的IP地址，域名的作用主要就是为了便于记忆。 域名解析就是域名到IP地址的转换过程，让人们通过注册的域名可以方便地访问到网站的一种服务。

假设输入的URL域名，那肯定会涉及到DNS解析；而如果URL本身就是IP，那就不需要进行DNS解析。

解析的步骤大致如下图：



在**本地域名服务器**可能存在对应的域名缓存记录，首先在**本地域名服务器**中根据域名查询IP地址，如果没有找到的情况下，本地域名服务器会向**根域名服务器**发送一个请求。如果根域名服务器也不存在该域名时，本地域名会向**com顶级域名服务器**（TLD）发送一个请求，依次类推下去。直到最后本地域名服务器得到对应的IP地址并把它缓存到本地，供下次查询使用。

TCP连接

DNS解析返回域名的IP之后，接下来因为HTTP是基于TCP上的，因此浏览器要和该IP进行TCP三次握手，建立连接。TCP三次握手过程见：**TCP三次握手四次挥手**

SSL握手

如果我们使用的是HTTPS。对于HTTPS，需要有一个SSL/TLS的鉴权/认证，才能进行后续数据的交互。握手完成后，基于此TCP连接的通信数据都会经过SSL协商的加解密方式来完成数据加解密。

HTTP/HTTPS请求和响应

TCP连接已经建立好了，意味着桥已经搭好了，下一步就该传输HTTP（HTTPS与HTTP的区别只是多了一步加解密）消息了。从代码的角度上来说，就是使用套接字Socket，完成与服务器的连接，然后使用Socket的OutputStream发出一段字符串数据，就完成了一次请求，服务器经过处理返回HTTP响应字符串。这些字符串就是HTTP报文，它们的格式如下：（请求头+请求体，响应头+响应体）

```
POST /v3/weather/weatherInfo?city=%E9%95%BF%E6%B2%99&key=ae6c53e2186f33bbf240a12d80672d1b HTTP/1.1
Content-Length: 7
Content-Type: application/x-www-form-urlencoded
Host: restapi.amap.com

a=1&b=2
```

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 57
Access-Control-Allow-Headers: DNT,X-CustomHeader,Keep-Alive,User-Agent,X-Requested-With,If-Modified-Since,Cache-Control,Content-Type,key,x-biz,x-info,platinfo,encr,enginever,gzipped,poiid
Access-Control-Allow-Methods: *
Access-Control-Allow-Origin: *
Content-Type: application/json;charset=UTF-8
Date: Fri, 16 Apr 2021 07:49:42 GMT
Gsid: 033001107127161855938233800013940901594
Gsid: 033001107127161855938233800013940901594
Keep-Alive: timeout=58
Sc: 0.080
Server: Tengine

{"status":"0","info":"INVALID_PARAMS","infocode":"20000"}
```

浏览器解析和渲染页面

浏览器接收到了服务器的返回内容，接下来浏览器该把内容呈现给用户了。服务器返回的内容可能是HTML页面，也可能是JSON数据、XML数据、文件等数据。浏览器根据响应头中的**Content-Type**，来识别服务器响应的不同类型数据，进行最终的显示或下载等处理，如上图中的：**“Content-Type：application/json;charset=utf-8”**，则表示json数据，目前chrome浏览器默认会显示响应体中的JSON数据。

结束

在完成显示之后，根据HTTP协议规则，从HTTP1.1开始，默认会在请求头中加入：“Connection：keep-alive”，浏览器可以将其设置为："Connection：close"，但是几乎没有浏览器会这样去做。这个请求头的意思是：是否保持长连接。同时响应头中也会存在此字段，如果双方都为："keep-alive"，则会保持此TCP连接，否则完成请求之后，直接进入TCP四次挥手阶段断开连接。

7.8 断点续传原理（腾讯）

这道题想考察什么？

是否掌握断点续传原理

考察的知识点

HTTP协议

IO处理

考生如何回答

在基于HTTP协议的通信中，断点续传功能最核心的原理就是利用HTTP请求中Range字段。

当请求服务器上的一个文件时，通过在请求头中设置"Range： bytes=x- "

```
HttpURLConnection httpConnection = (HttpURLConnection)url.openConnection();
httpConnection.setRequestProperty("RANGE","bytes=200");
```

Range表示需要服务器返回的文件字节数据从第x字节开始，指定需要传输的文件数据范围。

当需要对服务端传输的文件数据进行保存（下载）。在Java中也可以借助RandomAccessFile的seek方法访问记录并读取指定位置的数据。

```
RandomAccess rafile = new RandomAccessFile("down.zip","rw");
// 定位文件指针到 200 位置
rafile.seek(200);
// 前200字节不改动，从200字节开始写入数据到文件
rafile.write(...)
```

7.9 如何保证下载文件的完整性（网易）

这道题想考察什么？

文件下载程序鲁棒性

考察的知识点

文件校验

考生如何回答

以Android程序进行更新升级为例，一般情况下，APP启动都会请求服务端：是否有新版本APK？

如果有升级需要，那么服务端可能会返回如下数据：

```
{'msg':'更新日志','url':'下载地址', 'version':'版本号',......}
```

那么此时我们通过服务端响应的json数据中新版本APK的URL地址对APK文件进行下载，然而在互联网中进行文件传输过程中，可能由于各种原因导致文件的不完整。此时我们程序中若无法进行APK文件完整性的判断，直接请求安装此APK文件，很显然会显示：无法安装。

为了解决这个文件，服务端的响应数据中需要增加文件校验码(MD5、SHA1、CRC32)。以MD5为例，服务端在响应的数据中新增了：md5字段。

```
{'msg':'更新日志','url':'下载地址', 'version':'版本号','md5':'xxx',......}
```

每个文件都可以用 MD5 验证程序算出一个固定的 MD5 码。当客户端下载了文件后，可以计算所下载的文件的 MD5 码，和服务端响应的 MD5 码比较。如果计算出来的和服务端响应的不匹配，那么下载的APK文件不完整，或是被别人动过手脚。

因此可以通过文件校验的方式保证下载文件的完整性。