1.计算机网络-HTTP-get与post

1-1：get与post的区别

1. Get是请求从服务器获取资源，Post用于传输实体本体
2. get和post请求都能使用额外的参数，get参数是以查询字符串出现在URL中，post参数存储在实体主体中
3. Http方法不会改变服务器状态，get方法是安全的，而post由于是传送实体主体内容，这个内容可能是用户上传的表单数据，上传成功后，服务器可能把这个数据存储到 数据库中，因此状态也就发生了变化
4. get在调用多次时，客户端收到的结果是一样的， 所以是幂等；post调用多次，会增加多行记录，不是幂等
5. get可缓存，post不可缓存
6. 对于get请求，浏览器会把http 头和数据一并发送出去，服务器响应200；post请求，浏览器先发送header，服务器响应之后，浏览器在发送数据，服务器响应200

2.计算机网络-HTTP-报文结构与状态码

2xx （3种）

200 OK：表示从客户端发送给服务器的请求被正常处理并返回；

204 No Content：表示客户端发送给客户端的请求得到了成功处理，但在返回的响应报文中不含实体的主体部分（没有资源可以返回）；

206 Patial Content：表示客户端进行了范围请求，并且服务器成功执行了这部分的GET请求，响应报文中包含由Content-Range指定范围的实体内容。

3xx （5种）

301 Moved Permanently：永久性重定向，表示请求的资源被分配了新的URL，之后应使用更改的URL；

302 Found：临时性重定向，表示请求的资源被分配了新的URL，希望本次访问使用新的URL；

> 301与302的区别：前者是永久移动，后者是临时移动（之后可能还会更改URL）

303 See Other：表示请求的资源被分配了新的URL，应使用GET方法定向获取请求的资源；

> 302与303的区别：后者明确表示客户端应当采用GET方式获取资源

304 Not Modified：表示客户端发送附带条件（是指采用GET方法的请求报文中包含if-Match、If-Modified-Since、If-None-Match、If-Range、If-Unmodified-Since中任一首部）的请求时，服务器端允许访问资源，但是请求为满足条件的情况下返回改状态码；

307 Temporary Redirect：临时重定向，与303有着相同的含义，307会遵照浏览器标准不会从POST变成GET；（不同浏览器可能会出现不同的情况）；

4xx （4种）

400 Bad Request：表示请求报文中存在语法错误；

401 Unauthorized：未经许可，需要通过HTTP认证；

403 Forbidden：服务器拒绝该次访问（访问权限出现问题）

404 Not Found：表示服务器上无法找到请求的资源，除此之外，也可以在服务器拒绝请求但不想给拒绝原因时使用；

5xx （2种）

500 Inter Server Error：表示服务器在执行请求时发生了错误，也有可能是web应用存在的bug或某些临时的错误时；

503 Server Unavailable：表示服务器暂时处于超负载或正在进行停机维护，无法处理请求；

3.计算机网络-HTTP-HTTP各个型号

3-1：HTTP1.0优缺点

I.优点

1. HTTP基本的报文格式就是header + body，头部信息也是key-value简单文本的形式。
2. HTTP协议里的各类请求方法、URI/URL、状态码、头字段等每个组成要求都没有被固定死，都允许开发人员自定义和扩充。
3. HTTP由于是工作在应用层，则它下层可以随意变化。
4. 应用广泛和跨平台

II.缺点

1. 无状态 由于无状态，它在完成有关联性的操作时会非常麻烦。例如登录->添加购物车->下单->结算->支付，这系列操作都要知道用户的身份才行。但服务器不知道这些请求是有关联的，每次都要问一遍身份信息。
2. 明文传输

明文意味着在传输过程中的信息，是可方便阅读的，通过浏览器的控制台或抓包软件都可以直接肉眼查看，信息的内容都毫无隐私可言，很容易就能被窃取。

3-2：HTTP/1.1相对于HTTP1.0改善

1. 长连接，早期HTTP/1.0，那就是每发起一个请求需要三次握手四次挥手等等操作，增加了通信开销。为了解决这些问题，HTTP/1.1提出了长连接的通信方式只要任意一端没有明确提出断开连接，则保持 TCP 连接状态。
2. HTTP/1.1 采用了长连接的方式，可在同一个 TCP 连接里面，客户端可以发起多个请求，只要第一个请求发出去了，不必等其回来，就可以发第二个请求出去，可以减少整体的响应时间。
3. 错误状态响应码，HTTP1.1新增了很多错误装填响应码，让开发者更加了解错误根源
4. 在HTTP1.0中主要使用header里的If-Modified-Since,Expires来做为缓存判断的标准，在HTTP1.1中引入了更多的缓存控制策略
5. HTTP1.0中，存在一些浪费带宽的现象，例如客户端只是需要某个对象的一部分，而服务器却将整个对象送过来了，并且不支持断点续传功能，HTTP1.1则在请求头引入了range头域，它允许只请求资源的某个部分

3-3：HTTP1.1缺点

1. 请求 / 响应头部未经压缩就发送，首部信息越多延迟越大。
2. 服务器是按请求的顺序响应的，如果服务器响应慢，会招致客户端一直请求不到数据，也就是队头阻塞；
3. 请求只能从客户端开始，服务器只能被动响应。

3-4：HTTP长连接,短连接(也是TCP连接,短连接)

1. 在HTTP/1.0中默认使用短连接。也就是说，客户端和服务器每进行一次HTTP操作，就建立一次连接，任务结束就中断连接。

2. 从HTTP/1.1起，默认使用长连接。客户端和服务器之间用于传输HTTP数据的TCP连接不会关闭，客户端再次访问这个服务器时，会继续使用这一条已经建立的连接

3-5：HTTP/2 做了什么优化?

1. HTTP/2 会压缩头如果你同时发出多个请求，他们的头是一样的或是相似的，那么，协议会帮你消除重复的部分。

这就是所谓的 HPACK 算法：在客户端和服务器同时维护一张头信息表，所有字段都会存入这个表，生成一个索引号，以后就不发送同样字段了，只发送索引号，这样就提高速度了。

2. HTTP/2全面采用了二进制格式，头信息和数据体都是二进制，计算机收到报文后，直接解析二进制报文，这增加了数据传输的效率。

3. HTTP/2的数据包不是按顺序发送的

4. HTTP/2是可以在一个连接中并发多个请求或回应。

5. HTTP/2 还在一定程度上改善了传统的「请求 - 应答」工作模式，服务不再是被动地响应，也可以主动向客户端发送消息。

3-6：HTTP/2有哪些缺陷?

多个HTTP请求在复用一个TCP连接，下层的TCP协议是不知道有多少个HTTP请求的。所以一旦发生了丢包现象，就会触发TCP的重传机制，这样在一个TCP连接中的所有的 HTTP请求都必须等待这个丢了的包被重传回来。

3-7：HTTP/3做了哪些优化?

HTTP/3把HTTP下层的TCP协议改成了UDP

因为UDP发生是不管顺序，也不管丢包的，所以不会出现HTTP/1.1的队头阻塞和HTTP/2的一个丢包全部重传问题。

但是由于UDP是不可靠传输的，而基于UDP的QUIC协议可以实现类似TCP的可靠性传输。主要是依赖

1. 当某个流发生丢包时，只会阻塞这个流， 其他流不会受到影响。
2. 更改了头部压缩算法，升级成了 QPack 。
3. QUIC 直接把以往的TCP和TLS/1.3的6次交互合并成了3次，减少了交互次数。

3-8：Http请求组成

一个HTTP请求报文由四个部分组成：请求行、请求头部、空行、请求数据。

1. 请求行

请求行由请求方法字段、URL字段和HTTP协议版本字段3个字段组成，它们用空格分隔。比如 GET /data/info.html HTTP/1.1

2. 请求头部

HTTP客户程序(例如浏览器)，向服务器发送请求的时候必须指明请求类型(一般是GET或者 POST)。如有必要，客户程序还可以选择发送其他的请求头。大多数请求头并不是必需的，。

常见的请求头字段含义：

Accept： 浏览器可接受的MIME类型。

Accept-Charset：浏览器可接受的字符集。

Accept-Encoding：浏览器能够进行解码的数据编码方式，比如gzip。Servlet能够向支持gzip的浏览器返回经gzip编码的HTML页面。许多情形下这可以减少5到10倍的下载时间。

Accept-Language：浏览器所希望的语言种类，当服务器能够提供一种以上的语言版本时要用到。

Authorization：授权信息，通常出现在对服务器发送的WWW-Authenticate头的应答中。

Content-Length：表示请求消息正文的长度。

Host： 客户机通过这个头告诉服务器，想访问的主机名。Host头域指定请求资源的Intenet主机和端口号，必须表示请求url的原始服务器或网关的位置。HTTP/1.1请求必须包含主机头域，否则系统会以400状态码返回。

If-Modified-Since：客户机通过这个头告诉服务器，资源的缓存时间。只有当所请求的内容在指定的时间后又经过修改才返回它，否则返回304“Not Modified”应答。

Referer：客户机通过这个头告诉服务器，它是从哪个资源来访问服务器的(防盗链)。包含一个URL，用户从该URL代表的页面出发访问当前请求的页面。

User-Agent：User-Agent头域的内容包含发出请求的用户信息。浏览器类型，如果Servlet返回的内容与浏览器类型有关则该值非常有用。

Cookie：客户机通过这个头可以向服务器带数据，这是最重要的请求头信息之一。

Pragma：指定“no-cache”值表示服务器必须返回一个刷新后的文档，即使它是代理服务器而且已经有了页面的本地拷贝。

From：请求发送者的email地址，由一些特殊的Web客户程序使用，浏览器不会用到它。

Connection：处理完这次请求后是否断开连接还是继续保持连接。如果Servlet看到这里的值为“Keep- Alive”，或者看到请求使用的是HTTP 1.1(HTTP 1.1默认进行持久连接)，它就可以利用持久连接的优点，当页面包含多个元素时(例如Applet，图片)，显著地减少下载所需要的时间。要实现这一点，Servlet需要在应答中发送一个Content-Length头，最简单的实现方法是：先把内容写入 ByteArrayOutputStream，然后在正式写出内容之前计算它的大小。

Range：Range头域可以请求实体的一个或者多个子范围。例如，

表示头500个字节：bytes=0-499

表示第二个500字节：bytes=500-999

表示最后500个字节：bytes=-500

表示500字节以后的范围：bytes=500-

第一个和最后一个字节：bytes=0-0,-1

同时指定几个范围：bytes=500-600,601-999

但是服务器可以忽略此请求头，如果无条件GET包含Range请求头，响应会以状态码206(PartialContent)返回而不是以200 (OK)。

UA-Pixels，UA-Color，UA-OS，UA-CPU：由某些版本的IE浏览器所发送的非标准的请求头，表示屏幕大小、颜色深度、操作系统和CPU类型。

3. 空行

它的作用是通过一个空行，告诉服务器请求头部到此为止。

4. 请求数据

若方法字段是GET，则此项为空，没有数据

若方法字段是POST,则通常来说此处放置的就是要提交的数据

4.计算机网络-HTTP-HTTPs

4-1：HTTP与HTTPS区别

1. HTTP 是超文本传输协议，信息是明文传输，存在安全风险的问题。 HTTPS 则解决 HTTP 不安全的缺陷，在 TCP 和 HTTP 网络层之间加入了 SSL/TLS 安全协议，使得报文能够加密传输。
2. HTTP 连接建立相对简单，TCP 三次握手之后便可进行HTTP的报文传输。而 HTTPS 在 TCP 三次握手之后，还需进行SSL/TLS的握手过程，才可进入加密报文传输。
3. HTTP 的端口号是 80， HTTPS 的端口号是 443。
4. HTTPS 协议需要向 CA（证书权威机构）申请数字证书，来保证服务器的身份是可信的。

4-2：HTTPS 解决了 HTTP 的哪些问题?

HTTP 由于是明文传输，所以安全上存在以下三个风险：

1. 窃听风险，比如通信链路上可以获取通信内容，用户号容易没。
2. 篡改风险，比如强制植入垃圾广告，视觉污染，用户眼容易瞎。
3. 冒充风险，比如冒充淘宝网站，用户钱容易没。

HTTPS 在 HTTP 与 TCP 层之间加入了 SSL/TLS 协议，可以很好的解决了上述的风险：

1. 混合加密的方式实现信息的机密性，解决了窃听的风险。

HTTPS 采用的是对称加密和非对称加密结合的「混合加密」方式：

在通信建立 前 采用非对称加密的方式交换「会话秘钥」，后续就不再使用非对称加密；在 通信过程中 全部使用对称加密的「会话秘钥」的方式加密明文数据。

采用「混合加密」的方式的原因：

对称加密只使用一个密钥，运算速度快，密钥必须保密，无法做到安全的密钥交换。 非对称加密使用两个密钥：公钥和私钥，公钥可以任意分发而私钥保密，解决了密钥交换问题但速度慢。

2. 摘要算法的方式来实现完整性，它能够为数据生成独一无二的「指纹」，指纹用于校验数据的完整性，解决了篡改的风险。

客户端在发送明文之前会通过摘要算法算出明文的「指纹」，发送的时候把「指纹 + 明文」一同加密成密文后，发送给服务器，服务器解密后，用相同的摘要算法算出发送过来的明文，通过比较客户端携带的「指纹」和当前算出的「指纹」做比较，若「指纹」相同，说明数据是完整的。

3. 将服务器公钥放入到数字证书中，解决了冒充的风险。

4-3：HTTPS 是如何建立连接的？其间交互了什么？

1. 客户端向服务器索要并验证服务器的公钥。
2. 双方协商生产「会话秘钥」。
3. 双方采用「会话秘钥」进行加密通信。

4-4：SSL/TLS握手

1. ClientHello

首先，由客户端向服务器发起加密通信请求，也就是 ClientHello 请求。在这一步，客户端主要向服务器发送以下信息：

（1）客户端支持的 SSL/TLS 协议版本。

（2）客户端生产用于「会话秘钥」的随机数。

（3）客户端支持的密码套件列表。

2. SeverHello

服务器收到客户端请求后，向客户端发出响应，也就是 SeverHello 。服务器回应的内容有如下内容：

（1）确认 SSL/ TLS 协议版本，如果浏览器不支持，则关闭加密通信。

（2）服务器生产的随机数（ Server Random ），后面用于生产「会话秘钥」。

（3）确认的密码套件列表，如 RSA 加密算法。

（4）服务器的数字证书。

3. 客户端回应

客户端收到服务器的回应之后，首先通过浏览器或者操作系统中的 CA 公钥，确认服务器的数字证书的真实性。

如果证书没有问题，客户端会从数字证书中取出服务器的公钥，然后使用它加密报文，向服务器发送如下信息：

（1）一个随机数（ pre-master key ）

（2）加密通信算法改变通知

（3）客户端握手结束通知

4. 服务器的最后回应

服务器收到客户端的第三个随机数（ pre-master key ）之后，通过协商的加密算法，计算出本次通信的「会话秘钥」。然后，向客户端发生最后的信息：

（1）加密通信算法改变通知。

（2）服务器握手结束通知。

4-5：HTTPS的加密过程

1. 用户在浏览器发起HTTPS请求（如 https://www.mogu.com/），默认使用服务端的443端口进行连接；
2. HTTPS需要使用一套CA数字证书，证书内会附带一个公钥Pub，而与之对应的私钥Private保留在服务端不公开；
3. 服务端收到请求，返回配置好的包含公钥Pub的证书给客户端；
4. 客户端收到证书，校验合法性，主要包括是否在有效期内、证书的域名与请求的域名是否匹配，上一级证书是否有效（递归判断，直到判断到系统内置或浏览器配置好的 根证书），如果不通过，则显示HTTPS警告信息，如果通过则继续；
5. 客户端生成一个用于对称加密的随机Key，并用证书内的公钥Pub进行加密，发送给服务端；
6. 服务端收到随机Key的密文，使用与公钥Pub配对的私钥Private进行解密，得到客户端真正想发送的随机Key；
7. 服务端使用客户端发送过来的随机Key对要传输的HTTP数据进行对称加密，将密文返回客户端；
8. 客户端使用随机Key对称解密密文，得到HTTP数据明文；
9. 后续HTTPS请求使用之前交换好的随机Key进行对称加解密。

4-6:加密

1. 对称加密：密钥只有一个，加密解密为同一个密码，且加解密速度快，典型的对称加密，算法有DES、AES等；
2. 非对称加密：密钥成对出现（且根据公钥无法推知私钥，根据私钥也无法推知公钥），加密解密使用不同密钥（公钥加密需要私钥解密，私钥加密需要公钥解密），相对对称加密速度较慢，典型的非对称加密算法有RSA、 DSA等。

5.计算机网络-HTTP-Cookie与Session

5-1：Cookie 和 Session 的区别

1. 安全性： Session 比 Cookie 安全，Session 是存储在服务器端的，Cookie 是存储在客户端的。

2. 存取值的类型不同：Cookie 只支持存字符串数据，想要设置其他类型的数据，需要将其转换成字符串，Session 可以存任意数据类型。

3. 有效期不同： Cookie 可设置为长时间保持，比如我们经常使用的默认登录功能，Session 一般失效时间较短，客户端关闭（默认情况下）或者 Session 超时都会失效。

4. 存储大小不同： 单个 Cookie 保存的数据不能超过 4K，Session 可存储数据远高于 Cookie，但是当访问量过多，会占用过多的服务器资源。

5-2：Cookie作用

cookie是服务路发送到用户浏览器并保存在本地的小快数据，它会在浏览器之后向同一服务器再次发起请求时被携带上，用于告知服务端两个请求是否来自同一浏览器

5-3：Session用户登录状态过程

1. 用户进行登录时，用户提交包含用户名和密码的表单，放入HTTP请求报文中，
2. 服务器验证该用户名和密码，如果正确则把用户信息存储到 Redis中，它在Redis中Key称为Session ID:
3. 服务器返回的响应报文的Se-Coeo首部字段包含了这个Session ID.客户端收到响应报文之后将该Cookie值存入浏览器中:
4. 客户编之后对间一 个服务器进行请求时会包含该Cookie值，服务器收到之后提取出Session ID.从Redis中取出用户信息，维续之前的业务操作。

5-4：token的验证流程

1. 客户端使用用户名跟密码请求登录
2. 服务端收到请求，去验证用户名与密码
3. 验证成功后，服务端会签发一个 token 并把这个 token 发送给客户端
4. 客户端收到 token 以后，会把它存储起来，比如放在 cookie 里或者 localStorage 里
5. 客户端每次向服务端请求资源的时候需要带着服务端签发的 token
6. 服务端收到请求，然后去验证客户端请求里面带着的 token ，如果验证成功，就向客户端返回请求的数据

5-5：token和cookie实现的区别

1. Session 是一种记录服务器和客户端会话状态的机制，使服务端有状态化，可以记录会话信息。而Token是令牌，访问资源接口（API）时所需要的资源凭证。Token 使服务端无状态化，不会存储会话信息。
2. Token每一个请求都有签名还能防止监听以及重放攻击，而Session就必须依赖链路层来保障通讯安全了。
3. 如果你的用户数据可能需要和第三方共享，或者允许第三方调用 API 接口，用Token 。如果永远只是自己的网站，自己的 App，用什么就无所谓了。

5-6：HTTP是不保存状态的协议,如何保存用户状态?

通过Session机制解决，Session的主要作用就是通过服务端记录用户的状态。

如应用场景购物车，当你要添加商品到购物车的时候，系统不知道是哪个用户操作的，因为HTTP协议是无状态的。服务端给特定的用户创建特定Session之后就可以标 识这个用户并且跟踪这个用户了（一般情况下，服务器会在一定时间内保存这个Session，过了时间限制，就会销毁这个Session）

5-7：如何保存session

在服务端保存Session的方法很多，最常用的就是内存和数据库(比如是使用内存数据库redis保存)。

5-8：如何实现 Session 跟踪呢?

大部分情况下，我们都是通过在Cookie 中附加一个 Session ID 来方式来跟踪。

5-9：Cookie 被禁用怎么办?

最常用的就是利用 URL 重写把 Session ID 直接附加在URL路径的后面。

5-10：URI和URL的区别是什么?

URI的作用像身份证号一样， URL的作用更像家庭住址一样。

6.计算机网络-综合应用-输入网址

6-1：输入网址过程

1. 输入地址,对URL进行解析，从而生成发送给Web服务器的请求信息。
2. 浏览器查找域名的IP地址,因为委托操作系统发送消息时，必须提供通信对象的 IP 地址。
1. 浏览器会首先查看本地硬盘的hosts文件，看看其中有没有和这个域名对应的规则，如果有的话就直接使用hosts文件里面的ip地址。
2. 如果在本地的hosts文件没有能够找到对应的ip地址，浏览器会发出一个DNS请求到本地DNS服务器
3. 查询你输入的网址的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，此过程是递归的方式进行查询。如果没有，本地DNS服务器还要向DNS根服务器进行查询。
4. 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。这种过程是迭代的过程。
5. 本地DNS服务器继续向域服务器发出请求，比如说请求的对象是.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
6. 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。
3. 浏览器向web服务器发送一个HTTP请求

通过DNS获取到IP后，就可以把HTTP的传输工作交给操作系统中的协议栈。

协议栈的内部分为几个部分，分别承担不同的工作。上下关系是有一定的规则的，上面的部分会向下面的部分委 托工作，下面的部分收到委托的工作并执行。

应用程序也就是浏览器通过调用 Socket 库，来委托协议栈工作。

协议栈的上半部分有两块，分别是负责收发数据的 TCP 和 UDP 协议，它们两会接受应用层的委托执行收发数据的操作。

协议栈的下一半是用IP协议控制网络包收发操作，在互联网上传数据时，数据会被切分成一块块的网络包，而将网络包发送给对方的操作就是由 IP 负责的。

IP 下面的网卡驱动程序负责控制网卡硬件，而最下面的网卡则负责完成实际的收发操作，也就是对网线 中的信号执行发送和接收操作。

拿到域名对应的IP地址之后，浏览器会以一个随机端口向服务器的WEB程序80端口发起TCP的连接请求。这个连接请求到达服务器端后，进入到网卡，然后是进入到内核的TCP/IP协议栈，还有可能要经过防火墙的过滤，最终到达WEB程序，最终建立了TCP/IP的连接。

4. 服务器的永久重定向响应

服务器给浏览器响应一个301永久重定向响应，这样浏览器就会访问3w了。

5. 浏览器跟踪重定向地址，因为现在浏览器知道了 "http://www.google.com/"才是要访问的正确地址，所以它会发送另一个http请求

6. 服务器处理请求

http请求发送到了服务器，后端从在固定的端口接收到TCP报文开始，它会对TCP连接进行处理，对HTTP协议进行解析，并按照报文格式进一步封装成HTTP Request对象，供上层使用。

7. 服务器返回一个HTTP响应

服务器收到了我们的请求，也处理我们的请求，到这一步，它会把它的处理结果返回，也就是返回一个HTPP响应。

8. 浏览器显示 HTML,并请求获取嵌入在HTML的资源

6-2：为什么域名要分级设计

DNS 中的域名都是用句点来分隔的，代表了不同层次之间的界限。

域名的层级关系类似一个树状结构： 根 DNS 服务器 顶级域 DNS 服务器（com） 权威 DNS 服务器（server.com）

因此，客户端只要能够找到任意一台 DNS 服务器，就可以通过它找到根域 DNS 服务器，然后再一路顺 藤摸瓜 找到位于下层的某台目标 DNS 服务器。

6-3：重定向原因：

1. 网站调整（如改变网页目录结构）；

2. 网页被移到一个新地址；

3. 网页扩展名改变(如应用需要把.php改成.Html或.shtml)。

这种情况下，如果不做重定向，则用户收藏夹或搜索引擎数据库中旧地址只能让访问客户得到一个404页面错误信息，访问流量白白丧失；再者某些注册了多个域名的网站，也需要通过重定向让访问这些域名的用户自动跳转到主站点等。

7.各层协议

7-1：OSI与TCP/IP各层的结构与功能,都有哪些协议?

1. 应用层

为应用程序提供服务并且规定通信的规范和细节

常见的协议:

- HTTP(超文本传输协议)
- FTP(文件传输协议)
- TELNET(远程登录协议)
- SMTP(简单邮件传输协议)
- DNS(域名解析协议)

6. 表示层

主要负责数据格式的转换

5. 会话层

负责建立和断开通信连接

4.传输层

是唯一负责总体的数据传输和数据控制的一层。

- TCP: ~~面向连接,可靠性强,传输效率低~~
- UDP: ~~无连接,可靠性弱,传输效率快~~

3.网络层

将数据传输到目标地址；主要负责寻找地址和路由选择，网络层还可以实现拥塞控制、网际互连等功能

- IP
- IPX
- RIP
- OSPF等

2.数据链路层

物理地址寻址、数据的成帧、流量控制、数据的检错、重发等。

- ARP
- RARP
- SDLC
- HDLC
- PPP
- STP
- 帧中继等

1. 物理层

负责0、1比特流(0/1序列)与电压的高低、光的闪灭之间的转换

7-2：网络层与数据链路层有什么关系呢?

1. IP 的作用是主机之间通信用的，负责在「没有直连」的两个网络之间进行通信传输
2. MAC 的作用则是实现「直连」的两个设备之间通信。

理解一下：

就比如说，你想从xx村到海南市，你不得做公交车、汽车、火车、轮船到海南

那么你这整个的一个路线图，就是一个网络层，行程开端就是xx村---->>>源IP，目的IP---->行程结束就是海南

那么我从xx村到xx镇相当于是在这个区间内移动路线，也就是数据链路层，其中，xx村好比源 MAC 地址，xx镇好比目的 MAC 地址。 （只要是在线路（网络层包含的都是））

这个xx村、海南不会发生变化，但是中间的位置会一直在变，也就是说

IP源、目的不会变， Mac源、目的会变化

8.TCP的三次握手

8-1：TCP三次握手流程

1. 第一次握手： Client 什么都不能确认； Server 确认了对方发送正常，自己接收正常
2. 第二次握手： Client 确认了：自己发送、接收正常，对方发送、接收正常； Server 确认了：对方发送 正常，自己接收正常
3. 第三次握手： Client 确认了：自己发送、接收正常，对方发送、接收正常； Server 确认了：自己发 送、接收正常，对方发送、接收正常

8-2：TCP为什么要三次握手

1. 三次握手才可以阻止重复历史连接的初始化（主要原因）

- 比如说，客户端连续发送多次SYN建立连接的报文，在网络拥堵情况下：
    - 一个「旧 SYN 报文」比「最新的 SYN报文」早到达了服务端；
    - 那么此时服务端就会回一个SYN+ACK报文给客户端；
    - 客户端收到后可以根据自身的上下文，判断这是一个历史连接，序列号过期或超时，那么客户端就会发送 RST 报文给服 务端，表示中止这一次连接。

- 如果是两次握手连接，就不能判断当前连接是否是历史连接，三次握手则可以在客户端（发送方）准备发送第三次报文 时，客户端因有足够的上下文来判断当前连接是否是历史连接：
    - 如果是历史连接（序列号过期或超时），则第三次握手发送的报文是 RST 报文，以此中止历史连接；
    - 如果不是历史连接，则第三次发送的报文是 ACK 报文，通信双方就会成功建立连接

2. 三次握手才可以同步双方的初始序列号

   序列号是作为TCP可靠传输的一个关键因素，它的作用：

    - 接收方可以去除重复的数据；

- 接收方可以根据数据包的序列号按序接收；
- 可以标识发送出去的数据包中， 哪些是已经被对方收到的；

四次握手其实也能够可靠的同步双方的初始化序号，但由于客户端传输ACK和SYN可以优化成一步，所以就成了「三次握手」 。而两次握手只保证了一方的初始序列号能被对方成功接收，没办法保证双方的初始序列号都能被确认接收。

3. 三次握手才可以避免资源浪费

如果只有「两次握手」，当客户端的 SYN 请求连接在网络中阻塞，客户端没有接收到 ACK 报文，重复发送多次 SYN 报文，那么服务器在收到请求后就会建立多个冗余的无效链接，造成不必要的资源浪费。

8-3：TCP为什么SYN

 avatar

接收端传回发送端所发送的 SYN 是为了告诉发送端，我接收到的信息确实就是你所发送的信号了。

8-4：TCP除了SYN，为什么还要 ACK

双方通信无误必须是两者互相发送信息都无误。传了 SYN，证明发送方到接收方的通道没有问题，但是 接收方到发送方的通道还需要 ACK 信号来进行验证。

8-5：如何对三次握手进行性能优化

1. 三次握手建立连接的首要目的是「同步序列号」。只有同步了序列号才有可靠传输，所以当客户端发起 SYN 包时，可以通 过tcp_syn_retries 控制其重传的次数，比如内网中通讯时，就可以适当调低重试次数，尽快把错误暴露给应用程序

2. 当服务端SYN半连接队列溢出后，会导致后续连接被丢弃，可以通过 tcp_max_syn_backlog、somaxconn、backlog 参数 来调整 SYN 半连接队列的大小。

3. TCP Fast Open 功能可以绕过三次握手，使得 HTTP 请求减少了 1 个 RTT 的时间，所以也是一种性能优化方案

8-6：如何绕过三次握手发送数据

TCP Fast Open 功能可以绕过三次握手，使得 HTTP 请求减少了1个RTT的时间， Linux下可以通过tcp_fastopen开启该功能，同时必须保证服务端和客户端同时支持。

第一次发起 HTTP GET请求的时候，还是需要正常的三次握手流程。

之后发起 HTTP GET请求的时候，可以绕过三次握手，这就减少了握手带来的 1 个 RTT 的时间消耗。

8-7：TCP Fast Open的过程

I、客户端首次建立连接时的过程：

1. 客户端发送SYN报文，该报文包含Fast Open选项，且该选项的Cookie为空，这表明客户端请求Fast Open Cookie；
2. 支持 TCP Fast Open 的服务器生成 Cookie，并将其置于 SYN-ACK 数据包中的 Fast Open 选项以发回客户端；

3. 客户端收到 SYN-ACK 后，本地缓存 Fast Open 选项中的 Cookie。

II、如果客户端再次向服务器建立连接时的过程：

1. 客户端发送 SYN 报文，该报文包含「数据」以及此前记录的 Cookie；
2. 支持 TCP Fast Open 的服务器会对收到 Cookie 进行校验：如果 Cookie 有效，服务器将在 SYNACK 报文中对 SYN 和「数据」进行确认，服务器随后将「数据」递送至相应的应用程序；如果Cookie 无效，服务器将丢弃 SYN 报文中包含的「数据」，且其随后发出的 SYN-ACK 报文将只确认 SYN 的对应序列号；
3. 如果服务器接受了 SYN 报文中的「数据」，服务器可在握手完成之前发送「数据」， 这就减少了握手带来的 1 个 RTT 的时间消耗；
4. 客户端将发送 ACK 确认服务器发回的 SYN 以及「数据」，但如果客户端在初始的 SYN 报文中发送的「数据」没有被确认，则客户端将重新发送「数据」；
5. 此后的 TCP 连接的数据传输过程和非 TFO 的正常情况一致。

9 四次挥手

9-1：TCP四次挥手流程

1. 客户端-发送一个FIN，用来关闭客户端到服务器的数据传送
2. 服务器-收到这个FIN，它发回一个ACK，确认序号为收到的序号加1 。和SYN一样，一个FIN将占用一个序号
3. 服务器-关闭与客户端的连接，发送一个FIN给客户端
4. 客户端-发回 ACK 报文确认，并将确认序号设置为收到序号加1

9-2：TCP为什么要四次挥手

主要是从四次挥手双方发FIN包过程分析

1. 关闭连接时，客户端向服务端发送 FIN 时，仅仅表示客户端不再发送数据了但是还能接收数据。

2. 服务器收到客户端的 FIN 报文时，先回一个 ACK 应答报文，而服务端可能还有数据需要处理和发送，等服务端不再发送数 据时，才发送 FIN 报文给客户端来表示同意现在关闭连接。

服务端通常需要等待完成数据的发送和处理，所以服务端的ACK和FIN一般都会分开发送，从而比三次握手导致多了一次

9-3：如何对四次挥手进行优化

1. 主动发起FIN报文断开连接的一方，如果迟迟没收到对方的 ACK 回复，则会重传 FIN 报文，重传的次数由 tcp_orphan_retries 参数决定。

当主动方收到 ACK 报文后，连接就进入 FIN_WAIT2 状态，根据关闭的方式不同，优化的方式也不同：

- 如果这是 close 函数关闭的连接，那么它就是孤儿连接。如果 tcp_fin_timeout 秒内没有收到对方的 FIN 报文，连接就直 接关闭。同时，为了应对孤儿连接占用太多的资源， tcp_max_orphans定义了最大孤儿连接的数量，超过时连接就会直接释 放。
- 反之是 shutdown 函数关闭的连接，则不受此参数限制；

2. 当主动方接收到 FIN 报文，并返回 ACK 后，主动方的连接进入 TIME_WAIT 状态。这一状态会持续 1分钟，为了防止 TIME_WAIT 状态占用太多的资源， tcp_max_tw_buckets 定义了最大数量，超过时连接也

会直接释放。

3. 被动关闭的连接，它在回复 ACK 后就进入了 CLOSE_WAIT 状态，等待进程调用 close函数关闭连接。因此，出现大量 CLOSE_WAIT 状态的连接时，应当从应用程序中找问题。当被动方发送 FIN 报文后，连接就进入 LAST_ACK 状态，在未等到 ACK 时，会在tcp_orphan_retries 参数的控制下重发 FIN 报文。

9-4：为什么TIME_WAIT 等待的时间是 2MSL?

网络中可能存在来自发送方的数据包，当这些发送方的数据包被接收方处理后又会向对方发送响应，所以一来一回需要等待 2 倍的时间。

9-5：为什么需要TIME_WAIT 状态？（已经主动关闭连接了为啥还要保持资源一段时间呢？）

1. 防止具有相同「四元组」的「旧」数据包被收到；

   比如说服务端在关闭连接之前发送一个报文，但是被网络延迟了。这时有相同端口的TCP连接被复用后，被延迟的报文抵达了客户端，那么客户端是有可能正常接收这个过期的报文，这就会产生数据错乱等严重的问题。而使用TIME_WAIT =2MSL这个时间， 足以让两个方向上的数据包都被丢弃，使得原来连接的数据包在网络中都自然消失，再出现的数据包一定都是新建立连接所产生的。

2. 保证连接正确关闭

比如客户端四次挥手的最后一个 ACK 报文如果在网络中被丢失了，此时如果客户端TIME-WAIT 过短或没有，则就直接进入了 CLOSED 状态了，那么服务端则会一直处在 LASE_ACK状态。当客户端发起建立连接的 SYN 请求报文后，服务端会发送 RST 报文给客户端，连接建立的过程就会被终止。如果有这个时间，就会正常关闭，机试没有收到ACK报文，我也有时间重发并关闭

9-6：TIME_WAIT 过多有什么危害?

第一是内存资源占用；

第二是对端口资源的占用，一个 TCP 连接至少消耗一个本地端口；被占满就会导致无法创建新的连接。

9-7：如何优化 TIME_WAIT?

p146页

9-8：如果已经建立了连接，但是客户端突然出现故障了怎么办?

TCP 有一个机制是保活机制。

定义一个时间段，在这个时间段内，如果没有任何连接相关的活动， TCP 保活机制会开始作用，每隔一个时间间隔，发送一个 探测报文，该探测报文包含的数据非常少，如果连续几个探测报文都没有得到响应，则认为当前的 TCP 连接已经死亡，系统内 核将错误信息通知给上层应用程序。

10 TCP传输数据优化方案

10-1：TCP传输数据优化

TCP 会保证每一个报文都能够抵达对方，报文发出去后，必须接收到对方返回的确认报文 ACK，如果迟迟未收到，就会超时重发该报文，直到收到对方的 ACK 为止。所以， TCP 报文发出去后，并不会立马从内存中删除，因为重传时还需要用到它。这种方式的缺点是效率比较低的

可以采用并行批量发送报文，再批量确认报文即可，但是当接收方硬件不如发送方，或者系统繁忙、资源紧张时，是无法瞬间处理这么多报文的。于是，这些报文只能被丢掉，使得网络效率非常低。

为了解决这种现象发生， TCP 提供一种机制可以让「发送方」根据「接收方」的实际接收能力控制发送的数据量

因为网络的传输能力是有限的，当发送方依据发送窗口，发送超过网络处理能力的报文时，路由器会直接丢弃这些报文。影响了传输速度，发送缓冲区的大小最好是往带宽时延积靠近

11 TCP与UDP

11-1：TCP与UDP区别

avatar

1. UDP 在传送数据之前不需要先建立连接，TCP 提供面向连接的服务。在传送数据之前必须先建立连接，数据传送结束后要释放连接。 UDP 不提供可靠交付，但在某些情况下 UDP 确是一种最有效的工作方式（一般用于即时通信）， 比如： QQ 语音、 QQ 视频 、直播等等

2. UDP 确是一种最有效的工作方式（一般用于即时通信）,TCP一般用于文件传输、发送和接收邮件、远程登录等场景

11-2:TCP 协议如何保证可靠传输方式

1. 确认应答+序列号：TCP给发送的每一个包进行编号，接收方对数据包进行排序，把有序数据传送给应用层。

2. 校验和：TCP 将保持它首部和数据的检验和。目的是检测数据在传输过程中的任何变化。如果收到段的检验和有差错， TCP将丢弃这个报文段和不确认收到此报文段。

3. 流量控制：TCP 连接的每一方都有固定大小的缓冲空间， TCP的接收端只允许发送端发送接收端缓冲区能接纳的数据。当接收方来不及处理发送方的数据，能提示发送方降低发送的速率，防止包丢失。 TCP 使用的流量控制协议是可变大小的滑动窗口协议。 （TCP 利用滑动窗口实现流量控制）

4. 拥塞控制：当网络拥塞时，减少数据的发送。

5. ARQ协议： 也是为了实现可靠传输的，它的基本原理就是每发完一个分组就停止发送，等待对方确认。在收到确认后再发下 一个分组。

6. 重传机制：

7. 超时重传： 当 TCP 发出一个段后，它启动一个定时器，等待目的端确认收到这个报文段。如果 不能及时收到一个确认，将重发这个报文段。

11-3：TCP传输数据的性能优化

avatar

11-4：UDP如何做可靠传输

1、超时重传 2、有序接受 3、应答确认 4、滑动窗口流量控制

12.重传机制

12-1：常见的重传机制

```
1．超时重传
2．快速重传
3．SACK
4．D-SACK
```

12-2：超时重传

重传机制的其中一个方式，就是在发送数据时，设定一个定时器，当超过指定的时间后，没有收到对方的 ACK 确认应答报文，就会重发该数据，也就是我们常说的超时重传。

12.ARQ协议

12-1:什么是ARQ协议

ARQ协议是自动重传请求，他是OSI模型中数据链路层和传输层的错误纠正协议之一。它通过使用确认和超时这两个机制，在不可靠服务的基础上实现可靠的信息传输。如果发送方在发送后一段时间之内没有收到确认帧，它通常会重新发送。 ARQ包括停止等待ARQ协议和连续ARQ协议。

12-2：什么是停止等待ARQ协议

停止等待协议是为了实现可靠传输的，它的基本原理就是每发完一个分组就停止发送，等待对方确认（回复 ACK）。如果过了一段时间（超时时间后），还是没有收到 ACK 确认，说明没有发送成功，需要重新发送，直到收到确认后再发下一个分组；在停止等待协议中，若接收方收到重复分组，就丢弃该分组，但同时还要发送确认；

1. 优点： 简单
2. 缺点： 信道利用率低，等待时间长

12-3: 什么是连续ARQ协议

连续 ARQ 协议可提高信道利用率。发送方维持一个发送窗口，凡位于发送窗口内的分组可以连续发送出去，而不需要等待对方确认。接收方一般采用累计确认，对按序到达的最后一个分组发送确认，表明到这个分组为止的所有分组都已经正确收到了。

1. 优点： 信道利用率高，容易实现，即使确认丢失，也不必重传。
2. 缺点： 不能向发送方反映出接收方已经正确收到的所有分组的信息。
    - 比如：发送方发送了5条消息，中间第三条丢失（3号），这时接收方只能对前两个发送确认。发送方无法知道后三个分组的下落，而只好把后三个全部重传一次。这也叫 Go-Back-N（回退

N），表示需要退回来重传已经发送过的N 个消息。

13.滑动窗口和流量控制

13-1：什么是滑动窗口和流量控制

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段可以用来控制发送方窗口大小，从而影响发送方的发送速率。将窗口字段设置为0，则发送方不能发送数据。

14. 拥塞控制

14-1：什么是拥塞控制

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，网络的性能就要变坏。这种情况就叫拥塞。拥塞控制就是为了防止过多的数据注入到网络中，这样就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，就是网络能够承受现有的网络负荷。

14-2：拥塞控制算法

TCP的拥塞控制采用了四种算法：慢开始、拥塞避免、快重传快恢复。

1. 慢开始： 慢开始算法的思路是当主机开始发送数据时，如果立即把大量数据字节注入到网络，那么可能会引起网络阻塞，因为现在还不知道网络的符合情况。经验表明，较好的方法是先探测一下，即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数值。 cwnd初始值为1，每经过一个传播轮次， cwnd加倍。
2. 拥塞避免： 拥塞避免算法的思路是让拥塞窗口cwnd缓慢增大，即每经过一个往返时间RTT就把发送放的cwnd加1.
3. 快重传与快恢复： 它能快速恢复丢失的数据包。没有 FRR，如果数据包丢失了， TCP 将会使用定时器来要求传输暂停。在暂停的这段时间内，没有新的或复制的数据包被发送。有了 FRR，如果接收机接收到一个不按顺序的数据段，它会立即给发送机发送一个重复确认。如果发送机接收到三个重复确认，它会假定确认件指出的数据段丢失了，并立即重传这些丢失的数据段。有了FRR，就不会因为重传时要求的暂停被耽误。 当有单独的数据包丢失时，快速重传和恢复（FRR）能最有效地工作。当有多个数据信息包在某一段很短的时间内丢失时，它则不能很有效地工作。