

- 1.计算机网络-HTTP-get与post
 - 1-1: get与post的区别
- 2.计算机网络-HTTP-报文结构与状态码
- 3.计算机网络-HTTP-HTTP各个型号
 - 3-1: HTTP1.0优缺点
 - 3-2: HTTP/1.1相对于HTTP1.0改善
 - 3-3: HTTP1.1缺点
 - 3-4: HTTP长连接,短连接(也是TCP连接,短连接)
 - 3-5: HTTP/2 做了什么优化?
 - 3-6: HTTP/2有哪些缺陷?
 - 3-7: HTTP/3做了哪些优化?
- 4.计算机网络-HTTP-HTTPs
 - 4-1: HTTP与HTTPS区别
 - 4-2: HTTPS 解决了 HTTP 的哪些问题?
 - 4-3: HTTPS 是如何建立连接的? 其间交互了什么?
 - 4-4: SSL/TLS握手
 - 4-5: HTTPS的加密过程
 - 4-6:加密
- 5.计算机网络-HTTP-HTTPs-Cookie与Session
 - 5-1: Cookie 和 Session 的区别
 - 5-2: Cookie作用
 - 5-3: Session用户登录状态过程
 - 5-4: token的验证流程
 - 5-5: token和cookie实现的区别
 - 5-6: HTTP是不保存状态的协议,如何保存用户状态?
 - 5-7: 如何保存session
 - 5-8: 如何实现 Session 跟踪呢?
 - 5-9: Cookie 被禁用怎么办?
 - 5-10: URI和URL的区别是什么?
- 1.各层协议
 - 1-1: OSI与TCP/IP各层的结构与功能,都有哪些协议?
 - 1-2: 网络层与数据链路层有什么关系呢?
- 2.TCP的三次握手
 - 2-1: TCP三次握手流程
 - 2-2: TCP为什么要三次握手
 - 2-3: TCP为什么SYN
 - 2-4: TCP除了SYN, 为什么还要 ACK
 - 2-5: 如何对三次握手进行性能优化
 - 2-5-1:如何调整 SYN 半连接队列大小?
 - 2-5-2: 如果SYN半连接队列已满, 只能丢弃连接吗?
 - 2-5-3: syncookies的工作原理
 - 2-6: 如何绕过三次握手发送数据
 - 2-7: TCP Fast Open的过程
- 3 四次挥手
 - 3-1: TCP四次挥手流程
 - 3-2: TCP为什么要四次挥手

- 3-3: 如何对四次挥手进行优化
 - 3-4: 为什么TIME_WAIT 等待的时间是 2MSL?
 - 3-5: 大量closed_waited对服务端有什么影响
 - 已经主动关闭连接了为啥还要保持资源一段时间呢?
 - 什么命令可以查看有多少连接
- 4 TCP与UDP
 - 4-1: TCP与UDP区别
 - 4-2: TCP 协议如何保证可靠传输方式
 - 4-3: TCP传输数据的性能优化
 - 4-4: UDP如何做可靠传输
- 5. ARQ协议
 - 5-1: 什么是ARQ协议
 - 5-2: 什么是停止等待ARQ协议
 - 5-3: 什么是连续ARQ协议
- 6. 滑动窗口和流量控制
 - 6-1: 什么是滑动窗口和流量控制
- 7. 拥塞控制
 - 7-1: 什么是拥塞控制
 - 7-2: 拥塞控制算法
- 8. 在浏览器中输入url地址
 - 8.1: ->> 显示主页的过程(面试常客)
 - 8.2: ->> 浏览某个网页各个协议与HTTP关系
 - 8-3: DNS解析全过程
- 9. 状态码
- 11. IP
 - 11-1: 广播地址用于什么?
 - 11-2: IP分类的优缺点
 - 11-3: 如何解决IP分类存在的问题
 - 11-4: 为什么要分离网络号和主机号?
 - 解释一下ip地址 子网掩码与网关

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

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请求在复用一個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。

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「会话密钥」，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「会话密钥」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「会话密钥」。
3. 双方采用「会话密钥」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「会话密钥」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「会话密钥」。
- (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「会话密钥」。然后，向客户端发送最后的信息：

- (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-HTTPS-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是服务端发送到用户浏览器并保存在本地的小块数据，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 中的 Key称为Session ID: 服务器返回的响应报文的Set-Cookie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: 客户端之后对同一服务器进行请求时会包含该Cookie值，服务器收到之后提取出SessionID.从Redis中取出用户信息，继续之前的业务操作。

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。

1.各层协议

1-1: 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, 传输效率高~~

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

1-2: 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村到海南省，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

2.TCP的三次握手

2-1：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

2-2：TCP为什么要三次握手

1. 为了实现可靠数据传输，TCP 协议的通信双方，都必须维护一个序列号，以标识发送出去的数据包中，哪些是已经被对方收到的。
2. 如果只是两次握手，至多只有连接发起方的起始序列号能被确认，另一方选择的序列号则得不到确认

2-3：TCP为什么SYN



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。

2-4：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。

2-5：如何对三次握手进行性能优化



2-5-1:如何调整 SYN 半连接队列大小？

要想增大半连接队列，不能只单纯增大tcp_max_syn_backlog的值，还需一同增大somaxconn和backlog，也就是增大accept队列,否则只单纯增大tcp_max_syn_backlog是无效的。最后，改变了参数后，要重启Nginx服务，

因为SYN半连接队列和accept队列都是在listen()初始化的。

2-5-2：如果SYN半连接队列已满，只能丢弃连接吗？

并不是这样，开启syncookies功能就可以在不使用SYN半连接队列的情况下成功建立连接。

2-5-3：syncookies的工作原理

服务器根据当前状态计算出一个值，放在己方发出的SYN+ACK报文中发出，当客户端返回ACK报文时，取出该值验证，如果合法，就认为连接建立成功

syncookies 参数主要有以下三个值：0 值，表示关闭该功能；1 值，表示仅当 SYN 半连接队列放不下时，再启用它；2 值，表示无条件开启功能；那么在应对 SYN 攻击时，只需要设置为 1 即可

2-6：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。

2-7：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。

3 四次挥手

3-1：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

3-2：TCP为什么要四次挥手

任何一方都可以在数据传送结束后发出连接释放的通知，待对方确认后进入半关闭状态。当另一方也没有数据再发送的时候，则发出连接释放通知，对方确认后就完全关闭了TCP连接。

更多细节，请看[文章](#)

3-3：如何对四次挥手进行优化



3-4：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向对方发送响应，所以一来一回需要等待 2 倍的时间。

3-5：大量closed_waited对服务端有什么影响

导致cpu的使用率过高，可以通过netstat查看，但是我个人认为影响不是很大

已经主动关闭连接了为啥还要保持资源一段时间呢？

1. 防止上一次连接中的包，迷路后重新出现，影响新连接
2. 可靠的关闭TCP连接。在主动关闭方发送的最后一个ack(fin) 有可能丢失，这时被动方会重新发fin, 如果这时主动方处于 CLOSED 状态，就会响应 rst 而不是 ack。所以主动方要处于 TIME_WAIT 状态，而不能是 CLOSED。另外这么设计TIME_WAIT 会定时的回收资源，并不会占用很大资源的。

什么命令可以查看有多少连接

待定

4 TCP与UDP

4-1：TCP与UDP区别



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

4-2: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校验和。目的是检测数据在传输过程中的任何变化。如果收到段的校验和有差错, 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. 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。

4-3: TCP传输数据的性能优化



4-4: 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

5.ARQ协议

5-1:什么是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。

5-2: 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

5-3: 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。

6.滑动窗口和流量控制

6-1：什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。

7. 拥塞控制

7-1：什么是拥塞控制

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。

7-2：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。

8.在浏览器中输入url地址

8.1：->> 显示主页的过程(面试常客)


1. DNS解析
2. TCP连接
3. 发送HTTP请求
4. 服务器处理请求并返回HTTP报文

5. 浏览器解析渲染页面
6. 连接结束




文章


8.2: ->> 浏览某个网页各个协议与HTTP关系

想浏览 `http://demo.jp/xss` Web页面 客户端  DNS (告诉我demo.jp的IP地址吧)

1. HTTP协议的职责 生成针对目标Web服务器的HTTP请求报文 (请给我 `http://demo.jp/xss` 页面的资源)
2. TCP协议的职责 为了方便通信, 将HTTP请求报文分割成报文段 按序号分为多个报文段 把每个报文段可靠的传给对方

DNS  客户端 (demo.jp对应的IP地址是20X.189.105.112)

1. IP协议的职责 (路由器) 搜索对方的地址, 一边中转一边传送

客户端  demo.jp服务器 (IP地址: 20X.189.105.112)

1. TCP协议的职责 从对方那里收到的报文段 重组到达的报文段 按序号以原来的顺序重组请求报文
2. HTTP协议的职责 对Web服务器请求的内容的处理 (原来时想要这台计算机上的/xss/资源啊)

8-3:DNS解析全过程

1. 浏览器先检查自身缓存中有没有被解析过的这个域名对应的ip地址, 如果有, 解析结束。同时域名被缓存的时间也可通过TTL属性来设置。
2. 如果浏览器缓存中没有, 浏览器会检查操作系统缓存中有没有对应的已解析过的结果。
3. 如果至此还没有命中域名, 才会真正的请求本地域名服务器 (LDNS) 来解析这个域名,
4. 如果LDNS仍然没有命中, 就直接跳到Root Server 域名服务器请求解析
5. 根域名服务器返回给LDNS一个所查询域的主域名服务器
6. 此时LDNS再发送请求给上一步返回的gTLD
7. 接受请求的gTLD查找并返回这个域名对应的Name Server的地址, 这个Name Server就是网站注册的域名服务器
8. Name Server根据映射关系表找到目标ip, 返回给LDNS
9. LDNS缓存这个域名和对应的ip
10. LDNS把解析的结果返回给用户, 用户根据TTL值缓存到本地系统缓存中, 域名解析过程至此结束

9.状态码



11.IP



11-1：广播地址用于什么？

广播地址用于在同一个链路中相互连接的主机之间发送数据包。

学校班级中就有广播的例子，在准备上课的时候，通常班长会喊：“上课，全体起立！”，班里的同学听到这句话是不是全部都站起来了？这个句话就有广播的含义。

我认为就是我接受到了这个信息，上课之前要站起来

11-2:IP分类的优缺点

IP分类的优点

简单明了、选路（基于网络地址）简单。

IP分类的缺点

缺点一：同一网络下没有地址层次，缺少地址的灵活性。缺点二：不能很好的与现实网络匹配。如C类地址主机数量太少，B类主机数量太多等等

11-3：如何解决IP分类存在的问题

为了解决IP分类存在许多缺点，提出了无分类地址的方案，即CIDR。这种方式不再有分类地址的概念，32比特的IP地址被划分为两部分，前面是网络号，后面是主机号。

11-4：为什么要分离网络号和主机号？

1. 因为两台计算机要通讯，首先要判断是否处于同一个广播域内，即网络地址是否相同。如果网络地址相同，表明接受方在本网络上，那么可以把数据包直接发送到目标主机。

怎么进行子网划分？在上面我们知道可以通过子网掩码划分出网络号和主机号，那实际上子网掩码还有一个作用，那就是划分子网。子网划分实际上是将主机地址分为两个部分：子网网络地址和子网主机地址。形式如下：未做子网划分的ip地址：网络地址 + 主机地址 做子网划分后的ip地址：网络地址 + （子网网络地址 + 子网主机地址）假设对C类地址进行子网划分，网络地址192.168.1.0，使用子网掩码255.255.255.192对其进行子网划分。C类地址中前24位是网络号，最后8位是主机号，根据子网掩码可知从8位主机号中借用2位作为子网号。由于子网网络地址被划分成2位，那么子网地址就有4个，分别是00、01、10、11，具体划分如下图：划分后的4个子网如下表格：公有IP地址与私有IP地址在A、B、C分类地址，实际上有分公有IP地址和私有IP地址。

解释一下ip地址 子网掩码与网关