# 计算机网络

# TCP/IP 四层模型和 OSI 七层模型对应关系

|  |  |  |
| --- | --- | --- |
| **OSI七层模型** | **TCP/IP 四层模型** | **网络协议** |
| 应用层  （Application） | 应用层 | HTTP（超文本传输协议）  HTTPS（超文本传输安全协议）  FTP（文件传输协议）  SMTP（简单邮件传输协议）  DNS（域名服务）  等等 |
| 表示层  （Presentation） |
| 会话层  （Session） |
| 传输层  （Transport） | 传输层 | TCP（传输控制协议）  UDP（用户数据报协议） |
| 网络层  （Network） | 网际互连层 | IP（网际协议）  ICMP（网络控制消息协议）  IGMP（网络组管理协议） |
| 数据链路层  （Data Link） | 网络接口层 | 以太网  Wi-Fi  等等 |
| 物理层  （Physical） |

# 四层网络模型(每层的协议及应用)

1.网络接口层：

　网络接口层包括用于协作IP数据在已有网络介质上传输的协议。实际上TCP/IP标准并不定义与ISO数据链路层和物理层相对应的功能。相反，它定义像地址解析协议（Address Resolution Protocol,ARP）这样的协议，提供TCP/IP协议的[数据结构](http://lib.csdn.net/base/datastructure)和实际物理硬件之间的接口。

2.网际互联层：

网间层对应于OSI七层参考模型的网络层。本层包含IP协议、RIP协议（Routing Information Protocol，路由信息协议），负责数据的包装、寻址和路由。同时还包含网间控制报文协议（Internet Control Message Protocol,ICMP）用来提供网络诊断信息。

3.传输层：

传输层对应于OSI七层参考模型的传输层，它提供两种端到端的通信服务。其中TCP协议（Transmission Control Protocol）提供可靠的数据流运输服务，UDP协议（Use Datagram Protocol）提供不可靠的用户数据报服务。

4.应用层：

TCP/IP模型将OSI参考模型中的会话层和表示层的功能合并到应用层实现。　　  
应用层面向不同的网络应用引入了不同的应用层协议。其中，有基于TCP协议的，如文件传输协议（File Transfer Protocol，FTP）、虚拟终端协议（TELNET）、超文本链接协议（Hyper Text Transfer Protocol，HTTP），也有基于UDP协议的。

# 五层协议的体系结构

物理层、数据链路层、网络层、运输层、 应用层

**每一层的协议如下：**

物理层：RJ45、CLOCK、IEEE802.3    （中继器，集线器，网关）

数据链路：PPP、FR、HDLC、VLAN、MAC  （网桥，交换机）

网络层：IP、ICMP、ARP、RARP、OSPF、IPX、RIP、IGRP、 （路由器）

传输层：TCP、UDP、SPX

会话层：NFS、SQL、NETBIOS、RPC

表示层：JPEG、MPEG、ASII

应用层：FTP、DNS、Telnet、SMTP、HTTP、WWW、NFS

**每一层的作用如下：**

1.物理层：在这一层，**数据的单位称为比特（bit）。通过媒介传输比特,确定机械及电气规范。**

2.数据链路层：在这一层，**数据的单位称为帧（frame）。**将**比特组装成帧和点到点**的传递。

3.网络层：在这一层，**数据的单位称为数据包（packet）。**负责**数据包从源到宿的**传递和网际互连。

**4.传输层：在这一层，TCP的数据单元称为段 （segments），而UDP协议的数据单元称为“数据报（datagrams）。提供端到端的可靠报文传递和错误恢复。**

5.会话层：在这一层，**数据传送的单位不再另外命名，而是统称为报文**。**建立、管理和终止会话。**

6.表示层：对数据进行**翻译、加密和压缩。**

7.应用层：允许访问OSI环境的手段。

# 谈谈tcp,udp的区别，分别有哪些应用

**TCP的优缺点：**

优点： **可靠**，体现在TCP在传递数据之前，会有三次握手来建立连接，而且在数据传递时，有确认、窗口、重传、拥塞控制机制，在数据传完后，还会断开连接用来节约系统资源。 缺点： **慢，效率低，占用系统资源高，易被攻击**。 TCP在传递数据之前，要先建连接，这会消耗时间，而且在数据传递时，确认机制、重传机制、拥塞控制机制等都会消耗大量的时间，而且要在每台设备上维护所有的传输连接。事实上，每个连接都会占用系统的CPU、内存等硬件资源。 而且，因为TCP有确认机制、三次握手机制，这些也导致TCP容易被人利用，实现DOS、DDOS、CC等攻击。

**UDP的优缺点：**

优点： **快，比TCP稍安全** 。UDP没有TCP的握手、确认、窗口、重传、拥塞控制等机制，UDP是一个无状态的传输协议，所以它在传递数据时非常快。没有TCP的这些机制，UDP较TCP被攻击者利用的漏洞就要少一些。但UDP也是无法避免攻击的，比如：UDP Flood攻击……

缺点： **不可靠，不稳定 。**因为UDP没有TCP那些可靠的机制，在数据传递时，如果网络质量不好，就会很容易丢包。

**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，即不保证可靠交付;

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。

**应用:**

什么时候应该使用TCP？  
当对网络通讯质量有要求的时候，比如：整个数据要准确无误的传递给对方，这往往用于一些要求可靠的应用，比如HTTP、HTTPS、FTP等传输文件的协议，POP、SMTP等邮件传输的协议。  
在日常生活中，常见使用TCP协议的应用如下：

浏览器，用的HTTP

FlashFXP，用的FTP

Outlook，用的POP、SMTP

Putty，用的Telnet、SSH

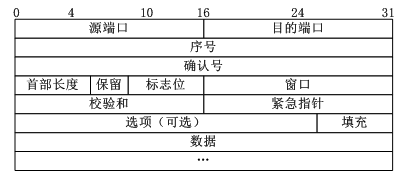
QQ文件传输

…………

什么时候应该使用UDP？  
当对网络通讯质量要求不高的时候，要求网络通讯速度能尽量的快，这时就可以使用UDP。  
比如，日常生活中，常见使用UDP协议的应用如下：QQ语音、QQ视频、TFTP等。

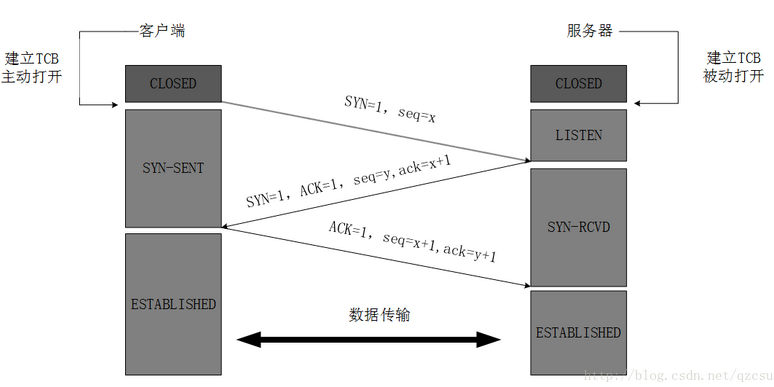
# TCP三次握手过程

## TCP报文格式



上图中有几个字段需要重点介绍下：  
        （1）序号：Seq序号，占32位，用来标识从TCP源端向目的端发送的字节流，发起方发送数据时对此进行标记。  
        （2）确认序号：Ack序号，占32位，只有ACK标志位为1时，确认序号字段才有效，Ack=Seq+1。  
        （3）标志位：共6个，即URG、ACK、PSH、RST、SYN、FIN等，具体含义如下：  
                （A）URG：紧急指针（urgent pointer）有效。  
                （B）ACK：确认序号有效。  
                （C）PSH：接收方应该尽快将这个报文交给应用层。  
                （D）RST：重置连接。  
                （E）SYN：发起一个新连接。  
                （F）FIN：释放一个连接。  
  
  需要注意的是：  
                （A）不要将确认序号Ack与标志位中的ACK搞混了。  
                （B）确认方Ack=发起方Seq+1，两端配对。

## 三次握手



1. TCP服务器进程先创建传输控制块TCB，时刻准备接受客户进程的连接请求，此时服务器就进入了LISTEN（监听）状态；

2. TCP客户进程也是先创建传输控制块TCB，然后向服务器发出连接请求报文，这时报文首部中的同部位SYN=1，同时选择一个初始序列号 seq=x ，此时，TCP客户端进程进入了 SYN-SENT（同步已发送状态）状态。TCP规定，SYN报文段（SYN=1的报文段）不能携带数据，但需要消耗掉一个序号。

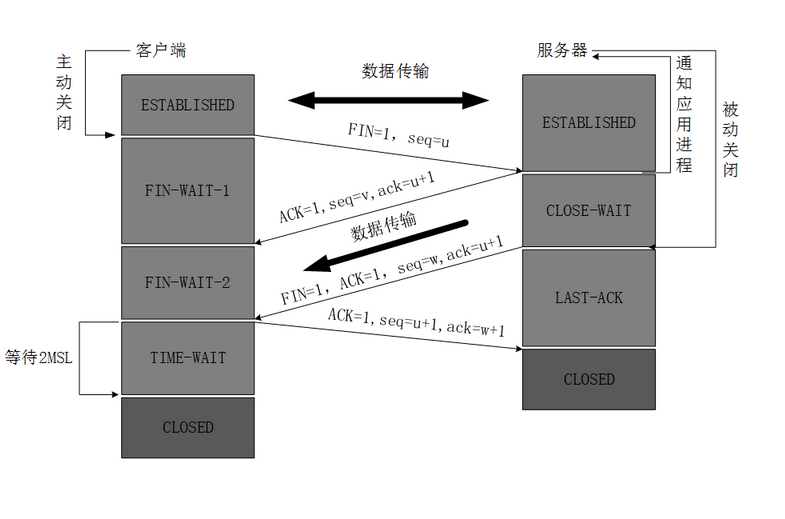
3. TCP服务器收到请求报文后，如果同意连接，则发出确认报文。确认报文中应该 ACK=1，SYN=1，确认号是ack=x+1，同时也要为自己初始化一个序列号 seq=y，此时，TCP服务器进程进入了SYN-RCVD（同步收到）状态。这个报文也不能携带数据，但是同样要消耗一个序号。

4. TCP客户进程收到确认后，还要向服务器给出确认。确认报文的ACK=1，ack=y+1，自己的序列号seq=x+1，此时，TCP连接建立，客户端进入ESTABLISHED（已建立连接）状态。TCP规定，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。

5. 当服务器收到客户端的确认后也进入ESTABLISHED状态，此后双方就可以开始通信了。

# 四次挥手

所谓四次挥手（Four-Way Wavehand）即终止TCP连接，就是指断开一个TCP连接时，需要客户端和服务端总共发送4个包以确认连接的断开。



 1.客户端进程发出连接释放报文，并且停止发送数据。释放数据报文首部，FIN=1，其序列号为seq=u（等于前面已经传送过来的数据的最后一个字节的序号加1），此时，客户端进入FIN-WAIT-1（终止等待1）状态。 TCP规定，FIN报文段即使不携带数据，也要消耗一个序号。

 2.服务器收到连接释放报文，发出确认报文，ACK=1，ack=u+1，并且带上自己的序列号seq=v，此时，服务端就进入了CLOSE-WAIT（关闭等待）状态。TCP服务器通知高层的应用进程，客户端向服务器的方向就释放了，这时候处于半关闭状态，即客户端已经没有数据要发送了，但是服务器若发送数据，客户端依然要接受。这个状态还要持续一段时间，也就是整个CLOSE-WAIT状态持续的时间。

 3.客户端收到服务器的确认请求后，此时，客户端就进入FIN-WAIT-2（终止等待2）状态，等待服务器发送连接释放报文（在这之前还需要接受服务器发送的最后的数据）。

 4.服务器将最后的数据发送完毕后，就向客户端发送连接释放报文，FIN=1，ack=u+1，由于在半关闭状态，服务器很可能又发送了一些数据，假定此时的序列号为seq=w，此时，服务器就进入了LAST-ACK（最后确认）状态，等待客户端的确认。

 5.客户端收到服务器的连接释放报文后，必须发出确认，ACK=1，ack=w+1，而自己的序列号是seq=u+1，此时，客户端就进入了TIME-WAIT（时间等待）状态。注意此时TCP连接还没有释放，必须经过2∗MSL（最长报文段寿命）的时间后，当客户端撤销相应的TCB后，才进入CLOSED状态。

 6.服务器只要收到了客户端发出的确认，立即进入CLOSED状态。同样，撤销TCB后，就结束了这次的TCP连接。可以看到，服务器结束TCP连接的时间要比客户端早一些。

# SYN攻击

在三次握手过程中，Server发送SYN-ACK之后，收到Client的ACK之前的TCP连接称为半连接（half-open connect），此时Server处于SYN\_RCVD状态，当收到ACK后，Server转入ESTABLISHED状态。SYN攻击就是Client在短时间内伪造大量不存在的IP地址，并向Server不断地发送SYN包，Server回复确认包，并等待Client的确认，由于源地址是不存在的，因此，Server需要不断重发直至超时，这些伪造的SYN包将产时间占用未连接队列，导致正常的SYN请求因为队列满而被丢弃，从而引起网络堵塞甚至系统瘫痪。SYN攻击时一种典型的DDOS攻击，检测SYN攻击的方式非常简单，即当Server上有大量半连接状态且源IP地址是随机的，则可以断定遭到SYN攻击了，使用如下命令可以让之现行：  
                #netstat -nap | grep SYN\_RECV

# TCP建立连接为什么是3次，2次可以吗？4次呢？(从失效的数据报谈2次为什么不行，从响应时间说为什么不用4次)

## 2次为什么不行？

一句话，主要防止已经失效的连接请求报文突然又传送到了服务器，从而产生错误。

如果使用的是两次握手建立连接，假设有这样一种场景，客户端发送了第一个请求连接并且没有丢失，只是因为在网络结点中滞留的时间太长了，由于TCP的客户端迟迟没有收到确认报文，以为服务器没有收到，此时重新向服务器发送这条报文，此后客户端和服务器经过两次握手完成连接，传输数据，然后关闭连接。此时此前滞留的那一次请求连接，网络通畅了到达了服务器，这个报文本该是失效的，但是，两次握手的机制将会让客户端和服务器再次建立连接，这将导致不必要的错误和资源的浪费。

如果采用的是三次握手，就算是那一次失效的报文传送过来了，服务端接受到了那条失效报文并且回复了确认报文，但是客户端不会再次发出确认。由于服务器收不到确认，就知道客户端并没有请求连接。

## 为什么不是4次？

因为三次握手已经为通信做好了准备，没有必要再增加一次握手来建立通信链接。造成资源浪费。

# 为什么连接的时候是三次握手，关闭的时候却是四次挥手？

建立连接的时候， 服务器在LISTEN状态下，收到建立连接请求的SYN报文后，把ACK和SYN放在一个报文里发送给客户端。   
而关闭连接时，服务器收到对方的FIN报文时，仅仅表示对方不再发送数据了但是还能接收数据，而自己也未必全部数据都发送给对方了，所以己方可以立即关闭，也可以发送一些数据给对方后，再发送FIN报文给对方来表示同意现在关闭连接，因此，己方ACK和FIN一般都会分开发送，从而导致多了一次。

# 为什么客户端最后还要等待2MSL？

1. 保证客户端发送的最后一个ACK报文能够到达服务器。因为这个ACK报文可能丢失，站在服务器的角度看来，我已经发送了FIN+ACK报文请求断开了，客户端还没有给我回应，应该是我发送的请求断开报文它没有收到，于是服务器又会重新发送一次，而客户端就能在这个2MSL时间段内收到这个重传的报文，接着给出回应报文，并且会重启2MSL计时器。

2. 防止类似与“三次握手”中提到了的“已经失效的连接请求报文段”出现在本连接中。客户端发送完最后一个确认报文后，在这个2MSL时间中，就可以使本连接持续的时间内所产生的所有报文段都从网络中消失。这样新的连接中不会出现旧连接的请求报文。

# Http

HTTP协议是超文本传输协议，是一个应用层的协议，是客户端和服务器端之间进行通信的一个协议， 一种典型的“请求-响应”式工作模式，即：客户端发起请求，服务器端回送响应。

HTTP协议一共有五大特点：

1、支持客户/服务器模式；

2、简单快速；

3、灵活；

4、无连接；

5、无状态；无状态是HTTP协议的主要特点之一。

# 浏览器发起请求到服务器端处理中间的整个流程（URL的请求流程）

https://www.cnblogs.com/wjlog/p/5790627.html

**整个流程如下：**

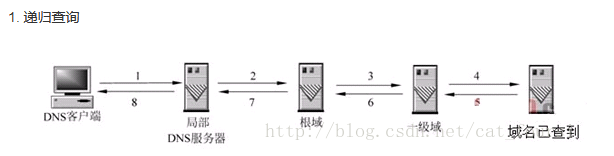
* 域名解析 -->
* 发起TCP的3次握手 -->
* 建立TCP连接后发起http请求 -->
* 服务器响应http请求，浏览器得到html代码 -->
* 浏览器解析html代码，并请求html代码中的资源（如js、css、图片等） -->
* 浏览器对页面进行渲染呈现给用户。

# DNS寻址过程

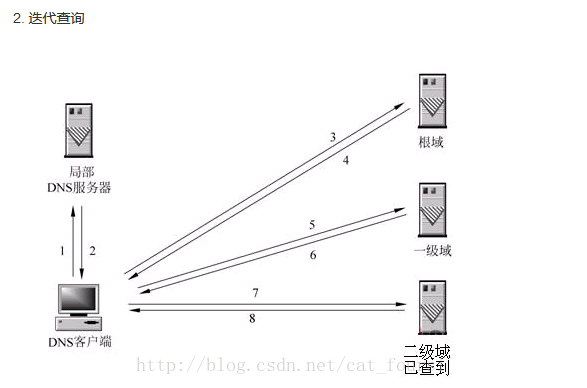
**DNS 查找 IP 地址**

DNS 是域名系统 (Domain Name System) 的缩写，是因特网的一项核心服务。它作为可以将域名和 IP 地址相互映射的一个分布式数据库，能够使人更方便的访问互联网，而不用去记住能够被机器直接读取的 IP 数字串。

两种查询方式：递归查询和迭代查询。



**递归查询：**主机向本地域名服务器的查询**一般都是采用递归查询**。所谓递归查询就是：如果主机所询问的本**地域名服务器不知道**被查询的域名的IP地址，那么本地域名服务器就以DNS客户的身份，**向其它根域名服务器**继续发出**查询请求报文**(即替主机继续查询)，而不是让主机自己进行下一步查询。因此，递归查询返回的查询结果或者是所要查询的IP地址，或者是报错，表示无法查询到所需的IP地址。



**迭代查询：**本地域名服务器向根域名服务器的查询的迭代查询。迭代查询的特点：当根域名服务器收到本地域名服务器发出的迭代查询请求报文时，要么给出所要查询的IP地址，要么告诉本地服务器：“**你下一步应当向哪一个域名服务器进行查询”。**然后让**本地服务器进行后续**的查询。根域名服务器通常是把自己知道的顶级域名服务器的IP地址告诉本地域名服务器，让本地域名服务器再向顶级域名服务器查询。顶级域名服务器在收到本地域名服务器的查询请求后，要么给出所要查询的IP地址，要么告诉本地服务器下一步应当向哪一个权限域名服务器进行查询。最后，知道了所要解析的IP地址或报错，然后把这个结果返回给发起查询的主机。

**DNS 进行域名解析的过程**：（以谷歌浏览器为例、递归查询）

1. 客户端发出 DNS 请求翻译 IP 地址或主机名；
2. 浏览器会首先**搜索浏览器自身的DNS缓存**（缓存时间比较短，大概只有1分钟，且只能容纳1000条缓存）。若查到请求的地址或名字，即向客户端发出应答信息；
3. 如果浏览器自身缓存找不到，则会**查看系统的DNS缓存,**如果找到且没有过期则停止搜索解析到此结束；
4. 如果本机仍然没有找到DNS缓存，则浏览器会发起一个DNS的系统调用，就会向本地配置的首选DNS服务器发起域名解析请求；（通过的是UDP协议向DNS的53端口发起请求，这个请求是递归的请求，也就是运营商的DNS服务器必须得提供给我们该域名的IP地址）;
5. 运营商的DNS服务器首先查找自身的缓存，找到对应的条目，且没有过期，则解析成功。如果没有找到对应的条目，则有由运营商的DNS代我们的浏览器发起迭代DNS解析请求。它首先是会找根域的DNS的IP地址，并依序从根域查找顶级域，由顶级查找二级域，二级域查找三级，直至找到要解析的地址或名字。然后，向客户端所在网络的 DNS 服务器发出应答信息，DNS 服务器收到应答后，先在缓存中存储，然后，将解析结果发给客户端。
6. 若没有找到，则返回错误信息。

# Http中有哪些请求方式

**GET:** 用于请求访问已经被URI(统一资源标识符)识别的资源,可以通过URL传参给服务器

**POST:** 用于传输信息给服务器,主要功能与GET方法类似,但一般推荐使用POST方式

**PUT:** 传输文件,报文主体中包含文件内容,保存到对应URI位置

**HEAD:** 获得报文首部,与GET方法类似,只是不返回报文主体,一般用于验证URI是否有效

**DELETE:** 删除文件,与PUT方法相反,删除对应URI位置的文件

**OPTIONS:** 查询响应URI支持的HTTP方法

# Http中get和post请求的区别

1、get是从服务器上获取数据，post是向服务器传送数据。

2、GET请求，请求的数据会附加在URL之后，以?分割URL和传输数据，多个参数用&连接。如如<http://127.0.0.1/Test/login.action?name=admin&password=admin，这个>过程用户是可见的。

POST请求：POST请求会把请求的数据放置在HTTP请求包的包体中。

因此，GET请求的数据会暴露在地址栏中，而POST请求则不会。

3、传输数据的大小

get传送的数据量较小，不能大于2KB。

post传送的数据量较大，一般被默认为不受限制。

4、

对于get方式，服务器端用Request.QueryString获取变量的值，对于post方式，服务器端用Request.Form获取提交的数据。

5、安全性

POST的安全性比GET的高。这里的安全是指真正的安全，不是不修改服务器的数据的安全。比如，在进行登录操作，通过GET请求，用户名和密码都会暴露再URL上，因为登录页面有可能被浏览器缓存以及其他人查看浏览器的历史记录的原因，此时的用户名和密码就很容易被他人拿到了。除此之外，GET请求提交的数据还可能会造成Cross-site request frogery攻击。

# post和put请求的区别

POST请求同PUT请求类似，都是向服务器端发送数据的，但是该请求会改变数据的种类等资源，就像数据库的insert操作一样，会创建新的内容。几乎目前所有的提交操作都是用POST请求的。

PUT请求是向服务器端发送数据的，从而改变信息，该请求就像数据库的update操作一样，用来修改数据的内容，但是不会增加数据的种类等，也就是说无论进行多少次PUT操作，其结果并没有不同。

# Http协议无状态协议，怎么解决?

无状态是指，当浏览器发送请求给服务器的时候，服务器响应，但是同一个浏览器再发送请求给服务器的时候，他会响应，但是他不知道你就是刚才那个浏览器，简单地说，就是服务器不会去记得你，所以是无状态协议。

解决方案：

两种用于保持HTTP连接状态的技术就应运而生了，一个是Cookie，而另一个则是Session。

Cookie是通过客户端保持状态的解决方案。

与Cookie相对的一个解决方案是Session，它是通过服务器来保持状态的。

# session机制、cookie机制

会话（Session）跟踪是Web程序中常用的技术，用来跟踪用户的整个会话。常用的会话跟踪技术是Cookie与Session。Cookie通过在客户端记录信息确定用户身份，Session通过在服务器端记录信息确定用户身份。

**Cookie机制**

Cookie技术是客户端的解决方案，Cookie就是由服务器发给客户端的特殊信息，而这些信息以文本文件的方式存放在客户端，然后客户端每次向服务器发送请求的时候都会带上这些特殊的信息。

Cookie实际上是一小段的文本信息。客户端请求服务器，如果服务器需要**记录该用户状态**，就产生一**个用户身份标识，**然后在响应消息中将该标识号以Cookie的形式传递给浏览器.**客户端浏览器会把Cookie保存起来。**浏览器在以后每次访问该web服务器时，浏览器把请求的**网址连同该Cookie**一同**提交给服务器**。服务器检查该Cookie，以此来辨认用户状态。

服务器还可以根据需要修改Cookie的内容。

Cookie不能被浏览器共享 ，Cookie具有不可跨域名性。例如浏览器访问Google只会携带Google的Cookie，而不会携带Baidu的Cookie。

Cookie的maxAge决定着Cookie的有效期，单位为秒（Second）。

本质上cookies就是http的一个扩展。有两个http头部是专门负责设置以及发送cookie的,它们分别是Set-Cookie以及Cookie。当服务器返回给客户端一个http响应信息时，其中如果包含Set-Cookie这个头部时，意思就是指示客户端建立一个cookie，并且在后续的http请求中自动发送这个cookie到服务器端，直到这个cookie过期。如果cookie的生存时间是整个会话期间的话，那么浏览器会将cookie保存在内存中，浏览器关闭时就会自动清除这个cookie。另外一种情况就是保存在客户端的硬盘中，浏览器关闭的话，该cookie也不会被清除，下次打开浏览器访问对应网站时，这个cookie就会自动再次发送到服务器端。

**默认情况下,cookie是一个会话级别的,用户退出浏览器后被删除。**

**session机制**

Session是另一种记录客户状态的机制，不同的是Cookie保存在客户端浏览器中，而Session保存在服务器上。客户端浏览器访问服务器的时候，服务器把客户端信息以某种形式记录在服务器上。这就是Session。客户端浏览器再次访问时只需要从该Session中查找该客户的状态就可以了。

如果说Cookie机制是通过检查客户身上的“通行证”来确定客户身份的话，那么Session机制就是通过检查服务器上的“客户明细表”来确认客户身份。Session相当于程序在服务器上建立的一份客户档案，客户来访的时候只需要查询客户档案表就可以了。

Session保存在服务器端。为了获得更高的存取速度，服务器一般把Session放在内存里。每个用户都会有一个独立的Session。如果Session内容过于复杂，当大量客户访问服务器时可能会导致内存溢出。因此，Session里的信息应该尽量精简。

Session生成后，只要用户继续访问，服务器就会更新Session的最后访问时间，并维护该Session。用户每访问服务器一次，无论是否读写Session，服务器都认为该用户的Session“活跃（active）”了一次。由于会有越来越多的用户访问服务器，因此Session也会越来越多。为防止内存溢出，服务器会把长时间内没有活跃的Session从内存删除。

**session工作原理:**

* web端会在客户端开始访问它时创建session,在访问特殊的程序并且该程序(servlet)决定与客户端开启会话时,服务器生成一个唯一值,称为Session ID（好像是通过取进程ID的方式取得的）。服务器开辟一块内存，对应于该Session ID。
* 服务器再将该Session ID写入浏览器的cookie。
* 服务器内有一进程，监视所有Session的活动状况，如果有Session超时或是主动关闭，服务器就释放改内存块。
* 当浏览器连入服务器时并请求Session时，服务器就读浏览器Cookie中的Session ID。
* 然后，服务检查该Session ID所对应的内存是否有效。
* 如果有效，就读出内存中的值。
* 如果无效，就建立新的Session。

**服务器端如何发送session的唯一标识到客户端?**

一般来说会有两种常用的方式：cookie和URL重写。

1、cookie

服务端只要设置Set-cookie头就可以将session的标识符传送到客户端，而客户端此后的每一次请求都会带上这个标识符；

2、URL重写

所谓URL重写，顾名思义就是重写URL。试想，在返回用户请求的页面之前，将页面内所有的URL后面全部以get参数的方式加上session标识符（或者加在path info部分等等），这样用户在收到响应之后，无论点击哪个链接或提交表单，都会在再带上session的标识符，从而就实现了会话的保持。读者可能会觉得这种做法比较麻烦，确实是这样，但是，如果客户端禁用了cookie的话，URL重写将会是首选。

# TOKEN

token的意思是“令牌”，是**服务端生成的一串字符串，**作为**客户端**进行请求的**一个标识**。

当用户第一次登录后，服务器生成一个token并将此token返回给客户端，以后客户端只需带上这个token前来请求数据即可，无需再次带上用户名和密码。

简单token的组成：**uid(用户唯一的身份标识)、time(当前时间的时间戳)、sign**（签名，token的前几位以哈希算法压缩成的一定长度的十六进制字符串。为防止token泄露）。

使用基于 Token 的身份验证方法，在服务端不需要存储用户的登录记录。大概的流程是这样的：   
 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验证成功后，服务端会签发一个 Token，再把这个 Token 发送给客户端 ；  
 3.客户端收到 Token 以后可以把它存储起来，比如放在 Cookie 里或者 Local Storage 里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；

4.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 。

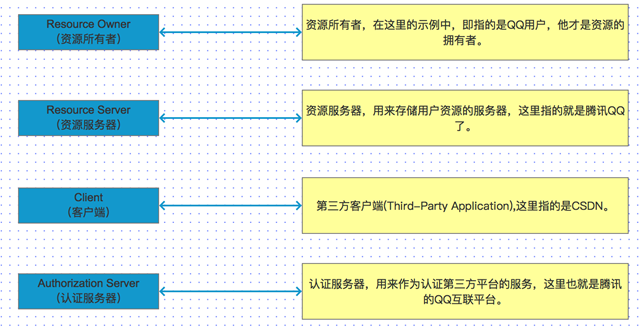
# OAuth2.0协议

1.什么是Oauth？

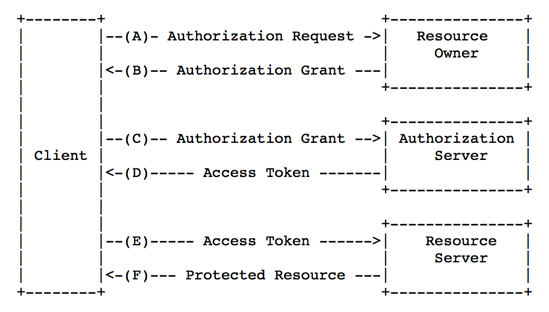
所谓的OAuth(Open Authorization)本质上就是一张开放的协议，OAuth协议为用户资源的授权提供了一个安全的、开放而又简易的标准。 与以往的授权方式不同该授权不会使第三方触及到用户的账户信息，即第三方无需使用用户名密码就可以申请该用户资源的授权，因此OAuth是安全的。

2. **OAuth 2.0中的4个成员**

在OAuth2.0中有4个成员，Resource Owner、Resource Server、Client、Authorization Server，如图所示：



3. **OAuth 2.0授权流程**



（A）用户打开客户端以后，客户端要求用户给予授权。

（B）用户同意给予客户端授权。

（C）客户端使用上一步获得的授权，向认证服务器申请令牌。

（D）认证服务器对客户端进行认证以后，确认无误，同意发放令牌。

（E）客户端使用令牌，向资源服务器申请获取资源。

（F）资源服务器确认令牌无误，同意向客户端开放资源。

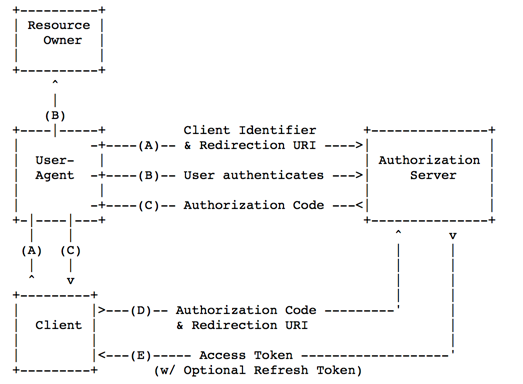
**4.OAuth 2.0授权模式**

此规范协议规定了4种授权类型：

* authorization code（授权码模式）
* implicit（简化模式）
* resource owner password credentials（密码模式）
* client credentials（客户端模式）

authorization code（授权码模式）

授权代码授权类型用于获取访问令牌和刷新令牌，并针对机密客户端进行优化。授权码模式是功能最完整、流程最严密的授权模式，它的特点就是通过客户端的后台服务器，与"服务提供商"的授权服务器进行互动。



流程解析：

1. 客户端通过用户代理，重定向请求授权服务器，所需参数：客户端标识及重定向URL；
2. 用户选择是否给予客户端授权；
3. 授权服务器给予客户端一个认证授权码，并跳转到指定的URI；
4. 客户端使用该授权码，重定向到授权服务器，获取令牌；
5. 授权服务器校验该授权码以及重定向的URI后，向客户端发送令牌或者更新令牌。

**A步骤中，客户端申请认证的URI，包含以下参数：**

* response\_type 必选项 表示的是要求指定的授权类型，此处必须设置为：code
* client\_id 必选项 客户端的唯一标识
* redirect\_uri 可选项 重定向的URI
* scope 可选项 授权的管道
* state 建议项 表示客户端状态，授权服务器会将该状态原值返回

**C步骤中，服务器回应客户端的URI，包含以下参数：**

* code：表示授权码，必选项。该码的有效期应该很短，通常设为10分钟，客户端只能使用该码一次，否则会被授权服务器拒绝。该码与客户端ID和重定向URI，是一一对应关系。
* state：如果客户端的请求中包含这个参数，认证服务器的回应也必须一模一样包含这个参数。

**D步骤中，客户端向认证服务器申请令牌的HTTP请求，包含以下参数：**

* grant\_type：表示使用的授权模式，必选项，此处的值固定为”authorization\_code”。
* code：表示上一步获得的授权码，必选项。
* redirect\_uri：表示重定向URI，必选项，且必须与A步骤中的该参数值保持一致。
* client\_id：表示客户端ID，必选项。

**E步骤中，认证服务器发送的HTTP回复，包含以下参数：**

* access\_token：表示访问令牌，必选项。
* token\_type：表示令牌类型，该值大小写不敏感，必选项，可以是bearer类型或m ac类型。
* expires\_in：表示过期时间，单位为秒。如果省略该参数，必须其他方式设置过期时间。
* refresh\_token：表示更新令牌，用来获取下一次的访问令牌，可选项。
* scope：表示权限范围，如果与客户端申请的范围一致，此项可省略。

# http和https区别

HTTPS：是以安全为目标的HTTP通道，简单讲是HTTP的安全版，即HTTP下加入SSL层，HTTPS的安全基础是SSL，因此加密的详细内容就需要SSL。

HTTPS和HTTP的区别主要如下：

　　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。

# 说说ssl四次握手的过程（https 的请求过程）

开始加密通信之前，客户端和服务器首先必须建立连接和交换参数，这个过程叫做握手（handshake）。

**1 客户端发出请求（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。

**2 服务器回应（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。（公钥）

**3 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。

**4 服务器的最后回应**

服务器通过私钥解密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"（会话秘钥是采用对称加密方式，而这里的公钥是采用非对称加密）。

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。

# https是如何保证数据安全的

Https中在Http的应用层和TCP/IP增加了一个SSL /TLS层，其实也是属于应用层，主要用来对数据进行加解密，保证数据的传输的正确性。

什么时候该使用 HTTPS?

银行网站、支付网关、购物网站、登录页、电子邮件以及一些企业部门的网站应该使用 HTTPS。

# 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。

# http1.0,1.1,2.0版本的区别

## ****HTTP1.0、 HTTP 1.1主要区别****

1.长连接

HTTP 1.0需要使用keep-alive参数来告知服务器端要建立一个长连接，而HTTP1.1默认支持长连接。

HTTP是基于TCP/IP协议的，创建一个TCP连接是需要经过三次握手的,有一定的开销，如果每次通讯都要重新建立连接的话，对性能有影响。因此最好能维持一个长连接，可以用个长连接来发多个请求。

2. 节约带宽

HTTP 1.1支持只发送header信息(不带任何body信息)，如果服务器认为客户端有权限请求服务器，则返回100，否则返回401。客户端如果接受到100，才开始把请求body发送到服务器。

这样当服务器返回401的时候，客户端就可以不用发送请求body了，节约了带宽。

3. HOST域

现在可以web server例如tomat，设置虚拟站点是非常常见的，也即是说，web server上的多个虚拟站点可以共享同一个ip和端口。

HTTP1.0是没有host域的，HTTP1.1才支持这个参数。

## ****HTTP1.1 、HTTP 2.0主要区别****

1.多路复用

HTTP2.0使用了多路复用的技术，做到同一个连接并发处理多个请求，而且并发请求的数量比HTTP1.1大了好几个数量级。

当然HTTP1.1也可以多建立几个TCP连接，来支持处理更多并发的请求，但是创建TCP连接本身也是有开销的。

2.数据压缩

HTTP1.1不支持header数据的压缩，HTTP2.0使用HPACK算法对header的数据进行压缩，这样数据体积小了，在网络上传输就会更快。

3.服务器推送

当我们对支持HTTP2.0的web server请求数据的时候，服务器会顺便把一些客户端需要的资源一起推送到客户端，免得客户端再次创建连接发送请求到服务器端获取。这种方式非常合适加载静态资源。

服务器端推送的这些资源其实存在客户端的某处地方，客户端直接从本地加载这些资源就可以了，不用走网络，速度自然是快很多的。

4.将通信的基本单位缩小为帧

即应用层(HTTP)和传输层(TCP or UDP)之间增加一个二进制分帧层，因此在多向请求和响应时，客户端和服务器可以把HTTP消息分解为互不依赖的帧，然后乱序发送，最后再在另一端把它们重新组合起来，解决了http 1.\*的对手阻塞问题。

5. 请求优先级

   HTTP 2.0 使用一个31比特的优先值,0表示最高优先级, 2(31)-1表示最低优先级，服务器端就可以根据优先级，控制资源分配，优先处理和返回最高优先级的请求帧给客户端。

# 长连接和短连接

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，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，每遇到这样一个Web资源，就会建立一个HTTP会话。

      从HTTP/1.1起，默认使用的是长连接，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：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。

**长连接短连接操作过程**

短连接的操作步骤是：建立连接——数据传输——关闭连接...建立连接——数据传输——关闭连接

长连接的操作步骤是：建立连接——数据传输...（保持连接）...数据传输——关闭连接

**长连接和短连接的优点和缺点**

长连接可以省去较多的TCP建立和关闭的操作，减少浪费，节约时间。但是如果Client与server之间的连接如果一直不关闭的话，随着客户端连接越来越多，server早晚有扛不住的时候。

这时候server端需要采取一些策略:

1.如关闭一些长时间没有读写事件发生的连接；

2.如果条件再允许就可以以客户端机器为颗粒度，限制每个客户端的最大长连接数；

      短连接对于服务器来说管理较为简单，存在的连接都是有用的连接，不需要额外的控制手段。但如果客户请求频繁，将在TCP的建立和关闭操作上浪费时间和带宽。

**什么时候用长连接，短连接？**  
 长连接多用于操作频繁，点对点的通讯，而且连接数不能太多情况。例如：数据库的连接用长连接， 如果用短连接频繁的通信会造成socket错误，而且频繁的socket 创建也是对资源的浪费。   
而像WEB网站的http服务一般都用短链接，因为长连接对于服务端来说会耗费一定的资源，而像WEB网站这么频繁的成千上万甚至上亿客户端的连接用短连接会更省一些资源，如果用长连接，而且同时有成千上万的用户，如果每个用户都占用一个连接的话，那可想而知吧。所以并发量大，但每个用户无需频繁操作情况下需用短连好。   
总之，长连接和短连接的选择要视情况而定。

# HTTP缓存

**什么是Http缓存？**

缓存缓存，就是把需要的东西存起来，不需要每次都去请求。主要目的减小服务器压力，放到客户端上来讲，还利于节省流量，还能流畅的把UI显示出来，提高了用户体验。对于Http缓存来讲，主要的就是校验缓存的有效性，也就是新鲜度。如果客户端不能及时响应服务端的数据变化，缓存一直不能被更新，那不就是得不偿失了？

**Http缓存策略**

**1.Expires策略**

Expires是Web服务器响应消息头字段，在响应http请求时告诉浏览器在过期时间前浏览器可以直接从浏览器缓存取数据，而无需再次请求。Expires规定了缓存失效时间，客户端拿当前时间和这个失效时间比较，如果超过了这个时间，即无效。

Expires策略有2个明显的缺点：   
1.缓存时间依赖于客户端，每个客户端的时间可能不相同，精度不高。   
2.缓存有效期的时间精度不高。

**2.Cache-control策略**

Cache -Control指定请求和响应遵循的缓存机制。

响应消息中的指令包括：

1.Public指示响应可被任何缓存区缓存。

2.Private指示对于单个用户的整个或部分响应消息，不能被共享缓存处理。这允许服务器仅仅描述当用户的部分响应消息，此响应消息对于其他用户的请求无效。

3.no-cache指示请求或响应消息不能缓存

4.no-store用于防止重要的信息被无意的发布。在请求消息中发送将使得请求和响应消息都不使用缓存。

5.max-age指示客户机可以接收生存期不大于指定时间（以秒为单位）的响应。

6.min-fresh指示客户机可以接收响应时间小于当前时间加上指定时间的响应。

7.max-stale指示客户机可以接收超出超时期间的响应消息。如果指定max-stale消息的值，那么客户机可以接收超出超时期指定值之内的响应消息。

**3.缓存校验字段**

1.Last-Modified/If-Modified-Since

**Last-Modified**：标示这个响应资源的最后修改时间。web服务器在响应请求时，告诉客户端资源的最后修改时间。

**If-Modified-Since**：当资源过期时（使用Cache-Control标识的max-age），发现资源具有Last-Modified声明，则再次向web服务器请求时带上头 If-Modified-Since，表示请求时间。web服务器收到请求后发现有头If-Modified-Since 则与被请求资源的最后修改时间进行比对。若最后修改时间较新，说明资源又被改动过，则响应整片资源内容（写在响应消息包体内），HTTP 200；若最后修改时间较旧，说明资源无新修改，则响应HTTP 304 (无需包体，节省浏览)，告知浏览器继续使用所保存的cache。

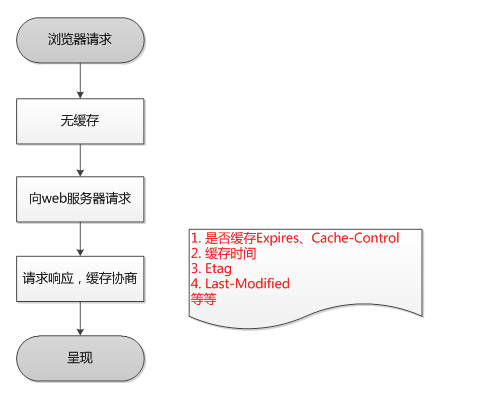
1. Etag/If-None-Match

**Etag**：web服务器响应请求时，告诉浏览器当前资源在服务器的唯一标识（生成规则由服务器觉得）。Apache中，ETag的值，默认是对文件的索引节（INode），大小（Size）和最后修改时间（MTime）进行Hash后得到的。

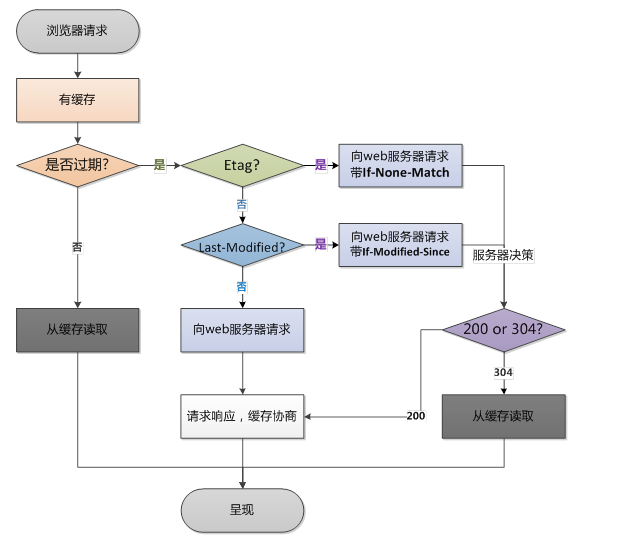
**If-None-Match**：当资源过期时（使用Cache-Control标识的max-age），发现资源具有Etage声明，则再次向web服务器请求时带上头If-None-Match （Etag的值）。web服务器收到请求后发现有头If-None-Match 则与被请求资源的相应校验串进行比对，决定返回200或304。

**请求流程：**

当第一次发起请求时：



当第二次发起请求时：



# HTTP状态码

一、1xx（临时响应）   
表示临时响应并需要请求者继续执行操作的状态代码。

1.http状态码 100 （继续） 请求者应当继续提出请求。 服务器返回此代码表示已收到请求的第一部分，正在等待其余部分。

2. http状态码 101 （切换协议） 请求者已要求服务器切换协议，服务器已确认并准备切换。

二、2xx （成功）   
表示成功处理了请求的状态代码。

1. http状态码 200 （成功） 服务器已成功处理了请求。 通常，这表示服务器提供了请求的网页。

2. http状态码 201 （已创建） 请求成功并且服务器创建了新的资源。

3.http状态码 202 （已接受） 服务器已接受请求，但尚未处理。

4. http状态码 203 （非授权信息） 服务器已成功处理了请求，但返回的信息可能来自另一来源。

5. http状态码 204 （无内容） 服务器成功处理了请求，但没有返回任何内容。

6. http状态码 205 （重置内容） 服务器成功处理了请求，但没有返回任何内容。

7. http状态码 206 （部分内容） 服务器成功处理了部分 GET 请求。

三、3xx （重定向）

表示要完成请求，需要进一步操作。 通常，这些状态代码用来重定向。

1. http状态码 300 （多种选择） 针对请求，服务器可执行多种操作。 服务器可根据请求者 (user agent) 选择一项操作，或提供操作列表供请求者选择。

2. http状态码 301 （永久移动） 请求的网页已永久移动到新位置。 服务器返回此响应（对 GET 或 HEAD 请求的响应）时，会自动将请求者转到新位置。

3.http状态码 302 （临时移动） 服务器目前从不同位置的网页响应请求，但请求者应继续使用原有位置来进行以后的请求。

4. http状态码 303 （查看其他位置） 请求者应当对不同的位置使用单独的 GET 请求来检索响应时，服务器返回此代码。

5. http状态码 304 （未修改） 自从上次请求后，请求的网页未修改过。 服务器返回此响应时，不会返回网页内容。

6. http状态码 305 （使用代理） 请求者只能使用代理访问请求的网页。 如果服务器返回此响应，还表示请求者应使用代理。

7. http状态码 307 （临时重定向） 服务器目前从不同位置的网页响应请求，但请求者应继续使用原有位置来进行以后的请求。

四、4xx（请求错误）   
这些状态代码表示请求可能出错，妨碍了服务器的处理。

1. http状态码 400 （错误请求） 服务器不理解请求的语法。   
2.http状态码 401 （未授权） 请求要求身份验证。 对于需要登录的网页，服务器可能返回此响应。

3. http状态码 403 （禁止） 服务器拒绝请求。   
4.http状态码 404 （未找到） 服务器找不到请求的网页。   
5.http状态码 405 （方法禁用） 禁用请求中指定的方法。

6.http状态码 406 （不接受） 无法使用请求的内容特性响应请求的网页。

7. http状态码 407 （需要代理授权） 此状态代码与 401（未授权）类似，但指定请求者应当授权使用代理。

8. http状态码 408 （请求超时） 服务器等候请求时发生超时。

9. http状态码 409 （冲突） 服务器在完成请求时发生冲突。 服务器必须在响应中包含有关冲突的信息。   
10.http状态码 410 （已删除） 如果请求的资源已永久删除，服务器就会返回此响应。

11. http状态码 411 （需要有效长度） 服务器不接受不含有效内容长度标头字段的请求。

12. http状态码 412 （未满足前提条件） 服务器未满足请求者在请求中设置的其中一个前提条件。

13. http状态码 413 （请求实体过大） 服务器无法处理请求，因为请求实体过大，超出服务器的处理能力。

14. http状态码 414 （请求的 URI 过长） 请求的 URI（通常为网址）过长，服务器无法处理。

15. http状态码 415 （不支持的媒体类型） 请求的格式不受请求页面的支持。

16. http状态码 416 （请求范围不符合要求） 如果页面无法提供请求的范围，则服务器会返回此状态代码。

17. http状态码 417 （未满足期望值） 服务器未满足”期望”请求标头字段的要求。

五、5xx（服务器错误）   
这些状态代码表示服务器在尝试处理请求时发生内部错误。 这些错误可能是服务器本身的错误，而不是请求出错。

1. http状态码 500 （服务器内部错误） 服务器遇到错误，无法完成请求。   
2.http状态码 501 （尚未实施） 服务器不具备完成请求的功能。 例如，服务器无法识别请求方法时可能会返回此代码。

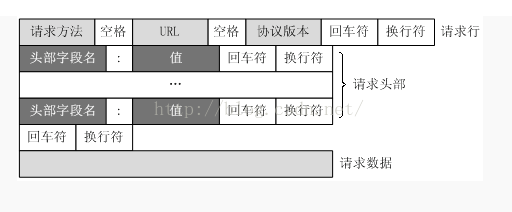
3. http状态码 502 （错误网关） 服务器作为网关或代理，从上游服务器收到无效响应。

4. **http状态码 503 （服务不可用） 服务器目前无法使用（由于超载或停机维护）。 通常，这只是暂时状态。**

**5.** http状态码 504 （网关超时） 服务器作为网关或代理，但是没有及时从上游服务器收到请求。   
6.http状态码 505 （HTTP 版本不受支持） 服务器不支持请求中所用的 HTTP 协议版本。

# HTTP的报文结构（包含请求头细节）

**HTTP请求报文格式就如下图所示:**



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。

方法字段就是HTTP使用的请求方法，比如常见的GET/POST。

2.请求头部

HTTP客户程序(例如[浏览器](http://www.2cto.com/os/liulanqi/))，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。

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。

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

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。

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。

3.空行

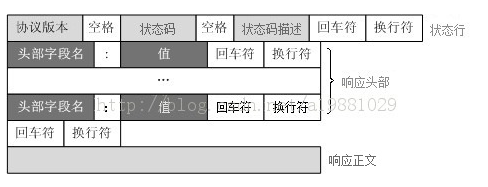
它的作用是通过一个空行，告诉服务器请求头部到此为止。

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。

**HTTP响应报文格式就如下图所示:**

****



一个具体的请求报文的例子：



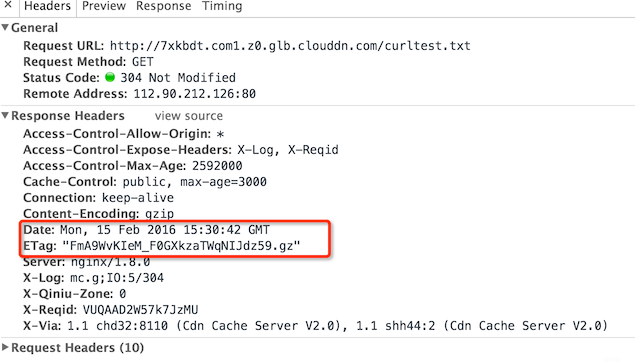
# 304状态码有什么含义？服务端是如何实现的？

304 的标准解释是：Not Modified。 客户端有缓冲的文档并发出了一个条件性的请求（一般是提供If-Modified-Since头表示客户只想比指定日期更新的文档）。服务器告诉客户，原来缓冲的文档还可以继续使用。

看一个关于304请求的响应头的信息，这里面有两个比较重要的请求头字段：**If-Modified-Since** 和 **If-None-Match**，这两个字段表示发送的是一个条件请求。



当客户端缓存了目标资源但不确定该缓存资源是否是最新版本的时候, 就会发送一个条件请求，这样就可以辨别出一个请求是否是条件请求，在进行条件请求时,客户端会提供给服务器一个If-Modified-Since请求头,其值为服务器上次返回的Last-Modified响应头中的Date日期值,还会提供一个If-None-Match请求头,值为服务器上次返回的ETag响应头的值。



服务器会读取到这两个请求头中的值,判断出客户端缓存的资源是否是最新的,如果是的话,**服务器就会返回HTTP/304 Not Modified响应头，但没有响应体**。客户端收到304响应后,就会从本地缓存中读取对应的资源。

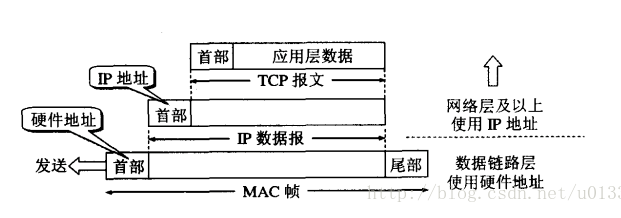
另一种情况是,如果服务器认为客户端缓存的资源已经过期了,那么服务器就会返回HTTP/200 OK响应,响应 体就是该资源当前最新的内容.客户端收到200响应后,就会用新的响应体覆盖掉旧的缓存资源。

只有在客户端缓存了对应资源且该资源的响应头中包含了Last-Modified或ETag的情况下,才可能发送条件请求.如果这两个头都不存在,则必须无条件(unconditionally)请求该资源,服务器也就必须返回完整的资源数据。

# ARP地址解析协议

## ****IP地址与MAC地址****

IP地址和MAC地址的区别在于MAC地址是数据链路层和物理层使用的地址，而IP地址是网络层和以上各层使用的地址。



在发送数据时，数据从高层到低层，然后才到通信链路上传输。使用IP地址的IP数据报一旦交给了数据链路层，就被封装成了MAC帧。MAC帧在传送时使用的源地址和目的地址都是硬件地址。连接在通信链路上的设备（主机或路由器）在接收MAC帧时，根据是MAC帧首部的硬件地址**。**在数据链路层看不到隐藏在MAC帧中的IP地址。只有在剥去MAC帧的首部和尾部后把MAC层的数据交给网络层后，网络层才能在IP数据报的首部中找到源IP地址和目的IP地址。

**ARP地址解析协议**

**ARP地址解析协议**（Address Resolution Protocol），其基本功能为透过目标设备的IP地址，查询目标设备的MAC地址，以保证通信的顺利进行。

每个主机都存有一个ARP高速缓存，存放本局域网上各主机和路由器的IP地址到MAC地址的映射表，称为ARP表。使用ARP协议动态维护此表。

当主机A要向本局域网上的某个主机B发送IP数据报时，就先在其ARP高速缓存中查找有无主机B的IP地址。如果有，就在ARP高速缓存中查出其对应的硬件地址，再把其硬件地址写入到MAC帧，然后通过局域网把该MAC帧发往此硬件地址。

如果主机高速缓存中没有则运行ARP按照以下步骤查找出主机B的硬件地址。

（1）ARP进程在本局域网上广播发送一个ARP请求分组，请求分组的内容包括主机A的ip地址和硬件地址，以及主机B的硬件地址；

（2）在本局域网上的所有主机上运行的ARP进程都会收到这个ARP分组请求。

（3）**主机B在ARP分组中见到自己的IP地址就向A发送ARP响应分组，并写入自己的硬件地址。注意，ARP请求分组是广播发送的，但ARP的相应分组是普通的单播。**

**（4）主机A收到主机B的相应分组后，就在他的ARP高速缓存中写入主机B的IP地址到硬件地址的映射。**

ARP是解决同一个局域网上主机与路由器的IP地址和硬件地址的映射问题。

如果所要找的主机和源主机不在同一个局域网上，那么通过ARP协议找到一个位于本局域网上的某个路由器硬件地址，然后把分组发送给这个路由器，让这个路由器把分组转发给下一个网络。

# RARP

逆地址解析协议 RARP 使只知道自己硬件地址的主机能够知道其 IP地址。

这种主机往往是无盘工作站。 因此 RARP协议目前已很少使用。

**RARP工作原理：**

1.网络上的每台设备都会有一个独一无二的硬件地址，通常是由设备厂商分配的MAC地址。PC1从网卡上读取MAC地址，然后在网络上发送一个RARP请求的广播数据包，请求RARP服务器回复该PC的IP地址。

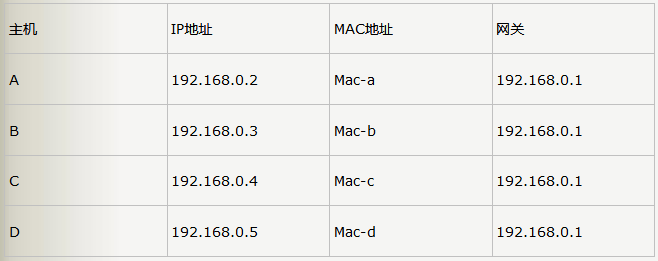
 2.RARP服务器收到了RARP请求数据包，为其分配IP地址，并将RARP回应发送给PC1。

 3.PC1收到RARP回应后，就使用得到的IP地址进行通讯。

ARP和RARP请求是广播方式，应答都是单播方式。

# ARP攻击

举个例子：假设局域网中有四台主机



主机A想和主机B通信。主机A会先查询自己的ARP缓存表里有没有B的联系方式，有的话，就将mac-b地址封装到数据包外面，发送出去。没有的话，A会向全网络发送一个ARP广播包，大声询问：我的IP地址是192.168.0.2，硬件地址是mac-a，我想知道IP地址是192.168.0.3的硬件地址是多少？   此时，局域网内所有主机都收到了，B收到后会单独私密回应：我是192.168.0.3，我的硬件地址是mac-b，其他主机不会理A的。

此时A知道了B的信息，同时也会动态的更新自身的缓存表。

**ARP协议的缺陷**

ARP协议是建立在信任局域网内所有节点的基础上的，他的效率很高。但是不安全。它是无状态的协议。他不会检查自己是否发过请求包，也不知道自己是否发过请求包。他也不管是否合法的应答，只要收到目标mac地址是自己的ARP reply或者ARP广播包（包括ARP reply和ARP request），都会接受并缓存。

**ARP攻击原理**

ARP欺骗攻击建立在局域网主机间相互信任的基础上的。

当A发广播询问：我想知道IP是192.168.0.3的硬件地址是多少？

此时B当然会回话：我是IP192.168.0.3我的硬件地址是mac-b，

可是此时IP地址是192.168.0.4的C也非法回了：我是IP192.168.0.3,我的硬件地址是mac-c。而且是大量的。

所以A就会误信192.168.0.3的硬件地址是mac-c，而且动态更新缓存表

这样主机C就劫持了主机A发送给主机B的数据，这就是ARP欺骗的过程。

假如C直接冒充网关，此时主机C会不停的发送ARP欺骗广播，大声说：我的ＩＰ是１９２.１６８.０.１，我的硬件地址是ｍａｃ－ｃ，

此时局域网内所有主机都被欺骗，更改自己的缓存表，此时Ｃ将会监听到整个局域网发送给互联网的数据报。

# 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）

# TCP流量控制-滑动窗口协议

滑动窗口

发送方维持一组连续的允许发送的帧的序号称为**发送窗口**。同时接收方维持一组连续的允许接收的帧序号，称为**接收窗口**。发送窗口是用来对发送方进行流量控制，接收窗口是用来控制可以接收哪些数据帧不可以接收哪些帧。

在发送端，收到一个确认帧，发送窗口就向前滑动一个帧位置，当发送窗口没有可以发送的帧时，发送方就停止发送。直到接收到接收方发送的确认帧使发送窗口向前移动。

在接收端，只有收到数据帧的序号落在接收窗口内才将该帧收下，否则一律丢弃。每收到一个帧后就发送回确认帧。

# TCP的拥塞控制

作用：拥塞控制就是为了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者链路不至于过载。拥塞控制要做的都有一个前提：就是网络能够承受现有的网络负荷。

为了更好的在运输层进行拥塞控制，因特网建议标准定义了以下4种算法：慢开始、拥塞避免、快重传和快恢复。

首先，对于每一个TCP连接，需要有以下两个状态变量：

**接收窗口 rwnd**: **接收方**根据当前缓存大小所许诺的最新窗口值.

**拥塞窗口 cwnd**: **发送方**根据自己估算的网络拥塞程度而设置的窗口值。

发送窗口的上限值=Min[rwnd,cwnf]。

**慢开始算法**：TCP刚连接好时，先令拥塞窗口cwnd =1 ,在每次收到一个对新报文段的确认时将cwnd加1（因此是成倍增长2的n次方). Cwnd的大小呈指数增长。

**拥塞避免算法**：是让cwnd缓慢的增加而不是加倍的增长，每经历过一次往返时间就使cwnd增加1，而不是加倍，这样使cwnd缓慢的增长，比慢启动要慢的多。

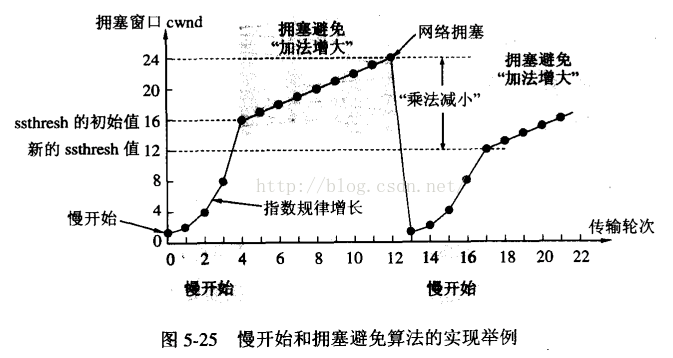
**拥塞避免：**为了防止cwnd增加过快而导致网络拥塞，所以需要设置一个**慢开始门限**ssthresh状态变量（我也不知道这个到底是什么，就认为他是一个拥塞控制的标识）,它的用法：

当cwnd < ssthresh,使用慢启动算法；

当cwnd > ssthresh,使用拥塞控制算法，停用慢启动算法；

当cwnd = ssthresh，这两个算法都可以。

无论是慢启动算法还是拥塞避免算法，只要判断网络出现拥塞，就要把慢启动开始门限(ssthresh)设置为设置为**发送窗口**的一半（>=2），cwnd(拥塞窗口)设置为1，然后再使用慢启动算法，这样做的目的能迅速的减少主机向网络中传输数据，使发生拥塞的路由器能够把队列中堆积的分组处理完毕。

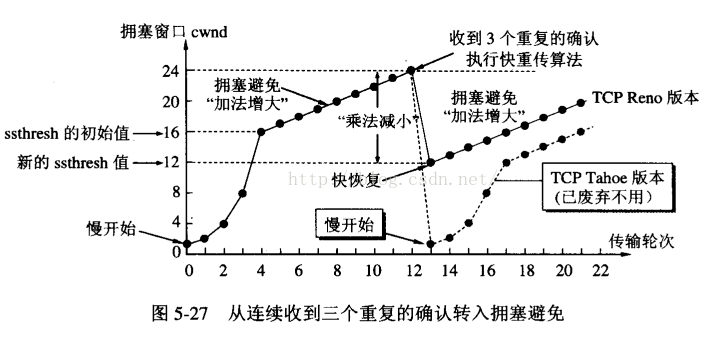


**快重传**：当发送方连续收到**三个重复的ACK报文时**，就**立即进行重传**。直接重传对方尚未收到的报文段，而不必等待那个报文段设置的重传计时器超时。

**快恢复**：

1. 当发送发连续接收到三个确认时，就执行乘法减小算法，把慢启动开始门限（ssthresh）减半，但是接下来并不执行慢开始算法。

2. 此时不执行慢启动算法，而是把cwnd设置为ssthresh的一半， 然后执行拥塞避免算法，使拥塞窗口缓慢增大。

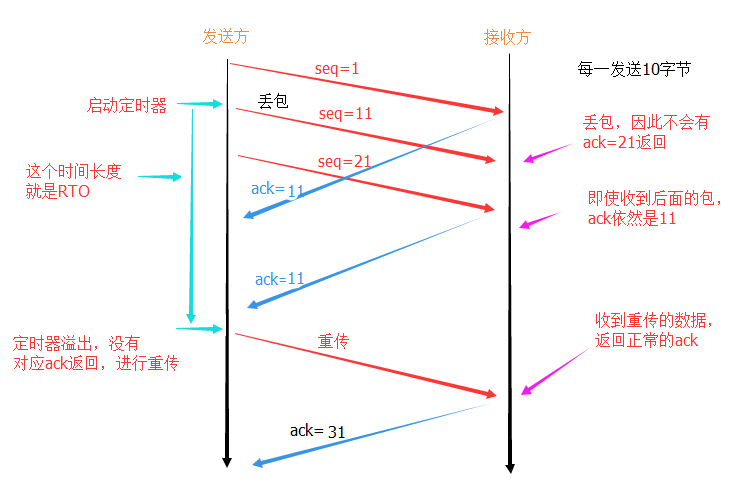


# TCP的重传机制

在发送一个数据之后，就**开启一个定时器**，若是在这个时间内**没有收到**发送数据的**ACK确认报文**，则**对该报文进行重传**，在达到一定次数还没有成功时放弃并发送一个复位信号

计时器设计的超时重传时间RTO应略大于网络中的RTT（**传输往返时间**），是自适应调整的。

通过图来了解重传机制：



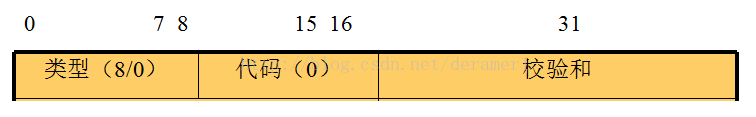
# ICMP

ICMP是网际报文控制协议，它是一个对IP协议的补充协议。允许主机或路由器报告差错情况和异常状况。

ICMP的报文分为两类，一类是ICMP询问报文，一类是ICMP差错报告报文。

**一、ICMP报文格式和各个字段的含义**

ICMP报文的格式如下：



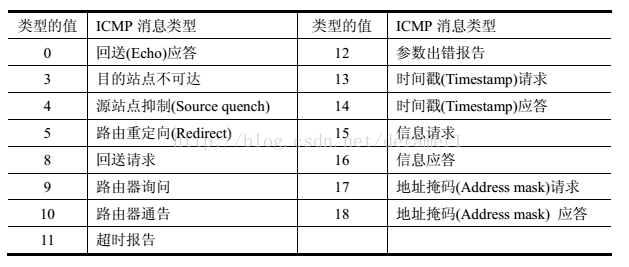
1.校验和：占两个字节，是对整个报文的报文信息的校验。

2.类型：占一个字节：用来表示ICMP的消息类型。

3.代码：占一个字节：用来对类型的进一步说明。

**二、ICMP常见的报文类型**

下表列出了几种常见的类型：



常见的ICMP询问报文有：8/0回送请求/应答报文，13/14时间戳回送请求/回答报文。

常见的ICMP差错报告报文有：3 目的站点不可达、 11时间超过、12参数问题、5改变路由、4源点抑制。

**差错报文：**

目的站点不可达：当路由器或主机不能交付数据的时候，就会向源点发送终点不可达的报文。

源点抑制：当路由器或主机因为拥塞而导致丢包的时候，就会向源点发送源点抑制报文，请求发送报文速度降低

时间超过:当路由器或主机发现生存时间TTL值为0时，会丢弃该报文，并向源点发送时间超过的信息。或者目的主机没在规定时间内收到所有的数据分片，会丢弃之前的数据分片，并发出报告。

参数问题：当路由器或主机发现数据包首部字段值不正确的时候，会丢弃该报文，并发送参数错误报文。

改变路由：路由器把改变路由报文发给主机。

**询问报文：**

回送请求和回答：请求 --向特定的目的主机发出询问。回答----收到询问的主机给一个回答。用来测试是否可达（PING）

时间戳请求和回答：请求---请某个主机或者路由器回答当前的日期和时间。

掩码地址请求和回答：请求子网掩码服务器得到某个端口地址掩码

路由询问和回答：本网络上的路由器是否正常工作。

# 交换机和路由器的区别

1. http和https？讲下？https是怎样的？如果我篡改了公钥呢？怎么防止？
2. 学过计算机网络是吧？socket熟悉吗？对它的读写缓冲区有理解吗？怎么的？那滑动窗口是怎样的？为什么这样设计？
3. 拥塞算法知道吗？哪些，分别怎样？
4. HTTP协议：问到后端给前端传递一条数据，HTTP是如何进行传递的？消息体里有哪些？后端传递的数据放在了HTTP哪里？（消息体这儿答得不太好，即数据内容怎么存放的？）
5. http: 了解http吗？get和post有什么区别？url中的host是什么意思？content-type是什么意思？如何获取http的状态码？状态码301和302有什么区别？http头有哪些字段？user-agent表示什么？