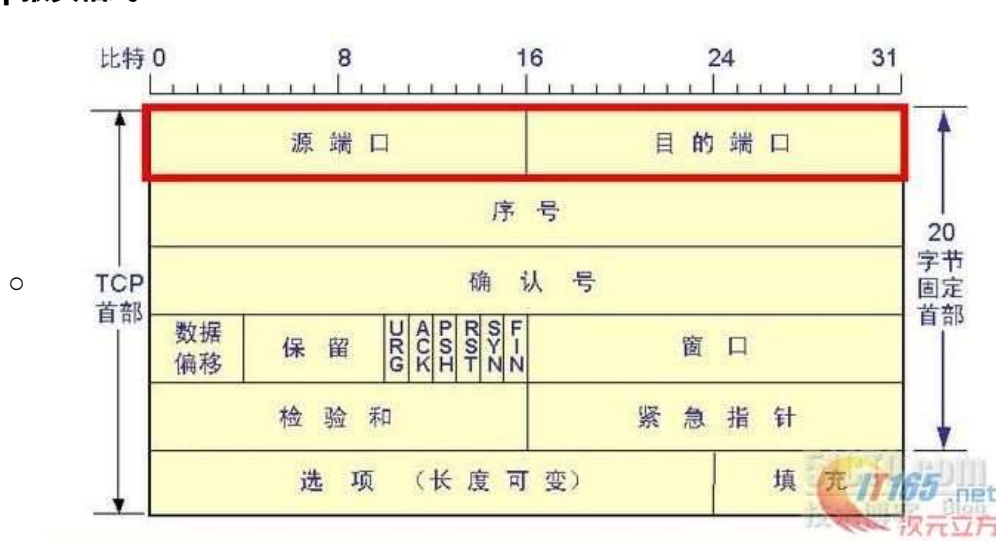


计算机网络

2017年8月20日 16:45

1. tcp报头格式



◆32位端口号:

源端口和目的端口各占16位， 2^{16} 等于65536，看端口的命令：netstat。

◆32位序号:

也称为顺序号 (Sequence Number)，简称为SEQ，

◆32位确认序号:

也称为应答号 (Acknowledgment Number)，简称为ACK。在握手阶段，确认序号将发送方的序号加1作为回答。

◆4位首部长度:

这个字段占4位，它的单位是32位 (4个字节)。本例值为7，TCP的头长度为28字节，等于正常的长度20字节加上可选项8个字节。，TCP的头长度最长可为60字节 (二进制1111换算为十进制为15， 15×4 字节=60字节)。

◆6位标志字段:

ACK 置1时表示确认号 (为合法，为0的时候表示数据段不包含确认信息，确认号被忽略)。

RST 置1时重建连接。如果接收到RST位时候，通常发生了某些错误。

SYN 置1时用来发起一个连接。

FIN 置1时表示发端完成发送任务。用来释放连接，表明发送方已经没有数据发送了。

URG 紧急指针，告诉接收TCP模块紧急指针域指着紧要数据。注：一般不使用。

PSH 置1时请求的数据段在接收方得到后就可直接送到应用程序，而不必等到缓冲区满时才传送。注：一般不使用。

◆16位检验和:

检验和覆盖了整个的TCP报文段：TCP首部和TCP数据。这是一个强制性的字段，一定由发端计算和存储，并由收端进行验证。

◆16位紧急指针:

注：一般不使用。

只有当URG标志置1时紧急指针才有效。紧急指针是一个正的偏移量，和序号字段中的值相加表示紧急数据最后一个字节的序号。

◆可选与变长选项：

通常为0，可根据首部长度推算。用于发送方与接收方协商最大报文段长度（MSS），或在高速网络环境下作窗口调节因子时使用。首部字段还定义了一个时间戳选项。

◆最常见的可选字段是最长报文大小，又称为MSS (Maximum Segment Size)。每个连接方通常都在握手的第一步中指明这个选项。它指明本端所能接收的最大长度的报文段。1460是以太网默认的大小。

2. UDP报文头部

UDP 首部:

16	31
源端口	目的端口
数据包长度	校验值
数据 DATA	

◆2字节源端口字段

源端口是一个大于1023的16位数字，由基于UDP应用程序的用户进程随机选择。

◆2字节的端口字段

◆2字节长度字段

指明了包括首部在内的UDP报文段长度。UDP长度字段的值是UDP报文头的长度(8字节)与UDP所携带数据长度的总和。

◆2字节校验和字段

是指整个UDP报文头和UDP所带的数据的校验和（也包括伪报文头）。伪报文头不包括在真正的UDP报文头中，但是它可以保证UDP数据被正确的主机收到了。因在校验和中加入了伪头标，故ICMP除能防止单纯数据差错之外，对IP分组也具有保护作用。

3. TCP/UDP区别，各自的应用场景

- <http://blog.csdn.net/leewccc/article/details/70225610>
- <http://blog.csdn.net/u013777351/article/details/49226101>

基本概念：

1：面向报文

面向报文的传输方式是应用层交给UDP多长的报文，UDP就照样发送，即一次发送一个报文。因此，应用程序必须选择合适大小的报文。若报文太长，则IP层需要分片，降低效率。若太短，会是IP太小。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。这也就是说，应用层交给UDP多长的报文，UDP就照样发送，即一次发送一个报文。

2：面向字节流

面向字节流的话，虽然应用程序和TCP的交互是一次一个数据块（大小不等），但TCP把应用程序看成是一连串的无结构的字节流。TCP有一个缓冲，当应用程序传送的数据块太长，TCP就可以把它划分短一些再传送。如果应用程序一次只发送一个字节，TCP也可以等待积累有足够多的字节后再构成报文段发送出去。

下图是TCP和UDP协议的一些应用。

应用层协议	应用	传输层协议
SMTP	电子邮件	TCP
TELNET	远程终端接入	
HTTP	万维网	
FTP	文件传输	
DNS	名字转换	UDP
TFTP	文件传输	
RIP	路由选择协议	
BOOTP, DHCP	IP 地址配置	
SNMP	网络管理	
NFS	远程文件服务器	
专用协议	IP 电话	
专用协议	流式多媒体通信	

下图是TCP和UDP协议的比较。

	TCP	UDP
可靠性	可靠	不可靠
连接性	面向连接	无连接
报文	面向字节流	面向报文（保留报文的边界）
效率	传输效率低	传输效率高
双工性	全双工	一对一、一对多、多对一、多对多
流量控制	有（滑动窗口）	无
拥塞控制	有（慢开始、拥塞避免、快重传、快恢复）	无
传输速度	慢	快
应用场合	对效率要求相对低，但对准确性要求相对高；或者是要求有连接的场景	对效率要求相对高，对准确性要求相对低的场景
应用示例	TCP一般用于文件传输（FTP http 对数据准确性要求高，速度可以相对慢），发送或接收邮件（pop imap SMTP 对数据准确性要求高，非紧急应用），远程登录（telnet SSH 对数据准确性有一定要求，有连接的概念）等等；	UDP一般用于即时通信（QQ聊天 对数据准确性和丢包要求比较低，但速度必须快），在线视频（rtsp 速度一定要快，保证视频连续，但是偶尔花了一个图像帧，人们还是能接受的），网络语音电话（VoIP 语音数据包一般比较小，需要高速发送，偶尔断音或串音也没有问题）等等。

两种协议都是传输层协议，为应用层提供信息载体。tcp协议是基于连接的可靠协议，有流量控制和差错控制，也正因为有可靠性的保证和控制手段，所以传输效率比UDP低；UDP协议是基于无连接的不可靠协议，没有控制手段，仅仅是将数据发送给对方，因此效率比tcp要高。

这里再详细说一下面向连接和面向无连接的区别：

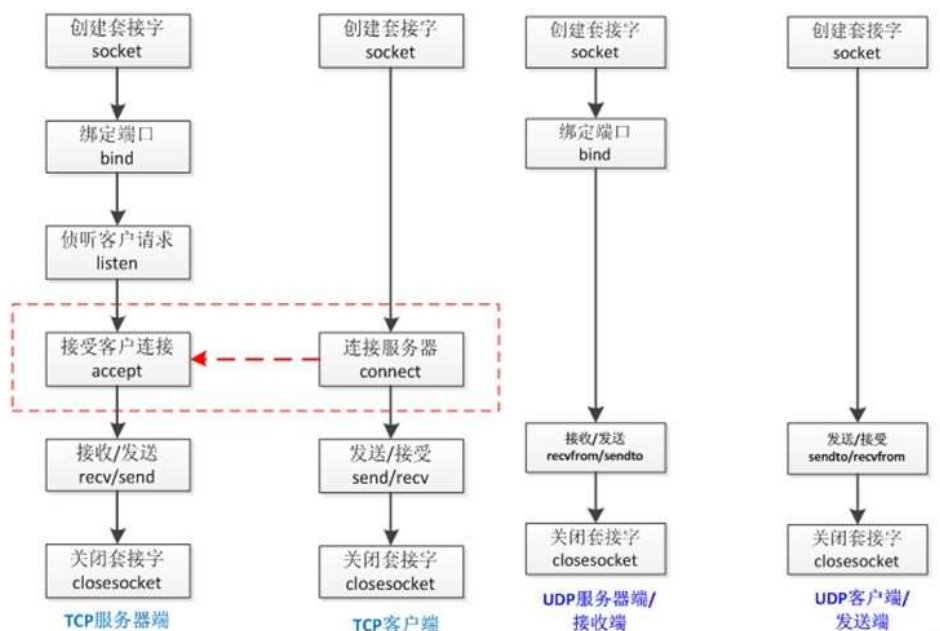
面向连接举例：两个人之间通过电话进行通信；

面向无连接举例：邮政服务，用户把信函放在邮件中期待邮政处理流程来传递邮政包裹。

显然，不可达代表不可靠。

TCP/UDP编程模型

从程序实现的角度来看，可以用下图来进行描述。



从上图也能清晰的看出，TCP通信需要服务器端侦听listen、接收客户端连接请求accept，等待客户端connect建立连接后才能进行数据包的收发（rcv/send）工作。而UDP则服务器和客户端的概念不明显，服务器端即接收端需要绑定端口，等待客户端的数据的到来。后续便可以进行数据的收发（rcvfrom/sendto）工作。

在前面讲解UDP时，提到了UDP保留了报文的边界，下面我们来谈谈TCP和UDP

中报文的边界问题。在默认的阻塞模式下，TCP无边界，UDP有边界。

对于TCP协议，客户端连续发送数据，只要服务端的这个函数的缓冲区足够大，会一次性接收过来，即客户端是分好几次发过来，是有边界的，而服务端却一次性接收过来，所以证明是无边界的；

而对于UDP协议，客户端连续发送数据，即使服务端的这个函数的缓冲区足够大，也只会一次一次的接收，发送多少次接收多少次，即客户端分几次发送过来，服务端就必须按几次接收，从而证明，这种UDP的通讯模式是有边界的。

TCP/UDP的优缺点：

TCP的优点：

可靠，稳定

TCP的可靠体现在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。

TCP的缺点：

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.....

UDP的缺点：

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。

TCP/UDP应用场景：

基于上面的优缺点，那么：

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

.....

那么什么时候应该使用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

.....

来自 <<http://blog.csdn.net/u013777351/article/details/49226101>>

4. HTTP状态码

分类	分类描述
1**	信息，服务器收到请求，需要请求者继续执行操作
2**	成功，操作被成功接收并处理
3**	重定向，需要进一步的操作以完成请求
4**	客户端错误，请求包含语法错误或无法完成请求
5**	服务器错误，服务器在处理请求的过程中发生了错误

HTTP状态码分类

HTTP状态码列表:

状态码	状态码英文名称	中文描述
100	Continue	继续。 客户端 应继续其请求
101	Switching Protocols	切换协议。服务器根据客户端的请求切换协议。只能切换到更高级的协议，例如，切换到HTTP的新版本协议
200	OK	请求成功。一般用于GET与POST请求
201	Created	已创建。成功请求并创建了新的资源
202	Accepted	已接受。已经接受请求，但未处理完成
203	Non-Authoritative	非授权信息。请求成功。但返回的meta信息不在原始的服务器，而
204	No Content	无内容。服务器成功处理，但未返回内容。在未更新网页的情况下，可确保浏览器继续显示当前文档
205	Reset Content	重置内容。服务器处理成功，用户终端（例如：浏览器）应重置文档视图。可通过此返回码清除浏览器的表单域
206	Partial Content	部分内容。服务器成功处理了部分GET请求

300	Multiple	多种选择。请求的资源可包括多个位置，相应可返回一个资源特征与地址的列表用于用户终端（例如：浏览器）选择
301	Moved Permanently	永久移动。请求的资源已被永久的移动到新URI，返回信息会包括新的URI，浏览器会自动定向到新URI。今后任何新的请求都应使用新
302	Found	临时移动。与301类似。但资源只是临时被移动。客户端应继续使用
303	See Other	查看其它地址。与301类似。使用GET和POST请求查看
304	Not Modified	未修改。所请求的资源未修改，服务器返回此状态码时，不会返回任何资源。客户端通常会缓存访问过的资源，通过提供一个头信息指出客户端希望只返回在指定日期之后修改的资源
305	Use Proxy	使用代理。所请求的资源必须通过代理访问
306	Unused	已经被废弃的HTTP状态码
307	Temporary Redirect	临时重定向。与302类似。使用GET请求重定向
400	Bad Request	客户端请求的语法错误，服务器无法理解
401	Unauthorized	请求要求用户的身份认证
402	Payment	保留，将来使用
403	Forbidden	服务器理解请求客户端的请求，但是拒绝执行此请求
404	Not Found	服务器无法根据客户端的请求找到资源（网页）。通过此代码，网站设计人员可设置"您所请求的资源无法找到"的个性页面
405	Method Not Allowed	客户端请求中的方法被禁止
406	Not Acceptable	服务器无法根据客户端请求的内容特性完成请求
407	Proxy Authentication Required	请求要求代理的身份认证，与401类似，但请求者应当使用代理进行
408	Request Time-	服务器等待客户端发送的请求时间过长，超时
409	Conflict	服务器完成客户端的PUT请求是可能返回此代码，服务器处理请求时
410	Gone	客户端请求的资源已经不存在。410不同于404，如果资源以前有现在被永久删除了可使用410代码，网站设计人员可通过301代码指定
411	Length	服务器无法处理客户端发送的不带Content-Length的请求信息
412	Precondition Failed	客户端请求信息的先决条件错误
413	Request Entity Too Large	由于请求的实体过大，服务器无法处理，因此拒绝请求。为防止客户端的连续请求，服务器可能会关闭连接。如果只是服务器暂时无法处理，则会包含一个Retry-After的响应信息
414	Request-URI Too Large	请求的URI过长（URI通常为网址），服务器无法处理
415	Unsupported Media Type	服务器无法处理请求附带的媒体格式

416	Requested range not	客户端请求的范围无效
417	Expectation	服务器无法满足Expect的请求头信息
500	Internal Server Error	服务器内部错误，无法完成请求
501	Not	服务器不支持请求的功能，无法完成请求
502	Bad Gateway	充当网关或代理的服务器，从远端服务器接收到了一个无效的请求
503	Service Unavailable	由于超载或系统维护，服务器暂时的无法处理客户端的请求。延时的长度可包含在服务器的Retry-After头信息中
504	Gateway Time-	充当网关或代理的服务器，未及时从远端服务器获取请求
505	HTTP Version not supported	服务器不支持请求的HTTP协议的版本，无法完成处理

HTTP状态码列表

来自 <<http://www.runoob.com/http/http-status-codes.html>>

5. SESSION机制

虽然session机制在web应用程序中被采用已经很长时间了，但是仍然有很多人不清楚session机制的本质，以至不能正确的应用这一技术。本文将详细讨论session的工作机制并且对在Java web application中应用session机制时常见的问题作出解答。

一、术语session

在我的经验里，session这个词被滥用的程度大概仅次于transaction，更加有趣的是transaction与session在某些语境下的含义是相同的。

session，中文经常翻译为会话，其本来的含义是指有始有终的一系列动作/消息，比如打电话时从拿起电话拨号到挂断电话这中间的一系列过程可以称之为一个session。有时候我们可以看到这样的话“在一个浏览器会话期间，...”，这里的会话一词用的就是其本义，是指从一个浏览器窗口打开到关闭这个期间^①。最混乱的是“用户（客户端）在一次会话期间”这样一句话，它可能指用户的一系列动作（一般情况下是同某个具体目的相关的一系列动作，比如从登录到选购商品到结账登出这样一个网上购物的过程，有时候也被称为一个transaction），然而有时候也可能仅仅是指一次连接，也有可能是指含义^①，其中的差别只能靠上下文来推断^②。

然而当session一词与网络协议相关联时，它又往往隐含了“面向连接”和/或“保持状态”这样两个含义，“面向连接”指的是在通信双方在通信之前要先建立一个通信的渠道，比如打电话，直到对方接了电话通信才能开始，与此相对的是写信，在你把信发出去的时候

你并不能确认对方的地址是否正确，通信渠道不一定能建立，但对发信人来说，通信已经开始了。“保持状态”则是指通信的一方能够把一系列的消息关联起来，使得消息之间可以互相依赖，比如一个服务员能够认出再次光临的老顾客并且记得上次这个顾客还欠店里一块钱。这一类的例子有“一个TCP session”或者“一个POP3 session”^③。

而到了web服务器蓬勃发展的时代，session在web开发语境下的语义又有了新的扩展，它的含义是指一类用来在客户端与服务器之间保持状态的解决方案^④。有时候session也用来指这种解决方案的存储结构，如“把xxx保存在session里”^⑤。由于各种用于web开发的语言在一定程度上都提供了对这种解决方案的支持，所以在某种特定语言的语境下，session也被用来指代该语言的解决方案，比如经常把Java里提供的javax.servlet.http.HttpSession简称为session^⑥。

鉴于这种混乱已不可改变，本文中session一词的运用也会根据上下文有不同的含义，请大家注意分辨。

在本文中，使用中文“浏览器会话期间”来表达含义^①，使用“session机制”来表达含义^④，使用“session”表达含义^⑤，使用具体的“HttpSession”来表达含义^⑥

二、HTTP协议与状态保持

HTTP协议本身是无状态的，这与HTTP协议本来的目的是相符的，客户端只需要简单的向服务器请求下载某些文件，无论是客户端还是服务器都没有必要纪录彼此过去的行为，每一次请求之间都是独立的，好比一个顾客和一个自动售货机或者一个普通的（非会员制）大卖场之间的关系一样。

然而聪明（或者贪心？）的人们很快发现如果能够提供一些按需生成的动态信息会使web变得更加有用，就像给有线电视加上点播功能一样。这种需求一方面迫使HTML逐步添加了表单、脚本、DOM等客户端行为，另一方面在服务器端则出现了CGI规范以响应客户端的动态请求，作为传输载体的HTTP协议也添加了文件上载、cookie这些特性。其中cookie的作用就是为了解决HTTP协议无状态的缺陷所作出的努力。至于后来出现的session机制则是另一种在客户端与服务器之间保持状态的解决方案。

让我们用几个例子来描述一下cookie和session机制之间的区别与联系。笔者曾经常去的一家咖啡店有喝5杯咖啡免费赠一杯咖啡的优惠，然而一次性消费5杯咖啡的机会微乎其微，这时就需要某种方式来纪录某位顾客的消费数量。想象一下其实也无外乎下面的几种方案：

- 1、该店的店员很厉害，能记住每位顾客的消费数量，只要顾客一走进咖啡店，店员就知道该怎么对待了。这种做法就是协议本身支持状态。

- 2、发给顾客一张卡片，上面记录着消费的数量，一般还有个有效期限。每次消费时，如果顾客出示这张卡片，则此次消费就会与以前或以后的消费相联系起来。这种做法就是在客户端保持状态。

3、发给顾客一张会员卡，除了卡号之外什么信息也不纪录，每次消费时，如果顾客出示该卡片，则店员在店里的纪录本上找到这个卡号对应的纪录添加一些消费信息。这种做法就是在服务器端保持状态。

由于HTTP协议是无状态的，而出于种种考虑也不希望使之成为有状态的，因此，后面两种方案就成为现实的选择。具体来说cookie机制采用的是在客户端保持状态的方案，而session机制采用的是在服务器端保持状态的方案。同时我们也看到，由于采用服务器端保持状态的方案在客户端也需要保存一个标识，所以session机制可能需要借助于cookie机制来达到保存标识的目的，但实际上它还有其他选择。

三、理解cookie机制

cookie机制的基本原理就如上面的例子一样简单，但是还有几个问题需要解决：“会员卡”如何分发；“会员卡”的内容；以及客户如何使用“会员卡”。

正统的cookie分发是通过扩展HTTP协议来实现的，服务器通过在HTTP的响应头中加上一行特殊的指示以提示浏览器按照指示生成相应的cookie。然而纯粹的客户端脚本如JavaScript或者VBScript也可以生成cookie。

而cookie的使用是由浏览器按照一定的原则在后台自动发送给服务器的。浏览器检查所有存储的cookie，如果某个cookie所声明的作用范围大于等于将要请求的资源所在的位置，则把该cookie附在请求资源的HTTP请求头上发送给服务器。意思是麦当劳的会员卡只能在麦当劳的店里出示，如果某家分店还发行了自己的会员卡，那么进这家店的时候除了要出示麦当劳的会员卡，还要出示这家店的会员卡。

cookie的内容主要包括：名字，值，过期时间，路径和域。

其中域可以指定某一个域比如.google.com，相当于总店招牌，比如宝洁公司，也可以指定一个域下的具体某台机器比如www.google.com或者froogle.google.com，可以用飘柔来做比。

路径就是跟在域名后面的URL路径，比如/或者/foo等等，可以用某飘柔专柜做比。路径与域合在一起就构成了cookie的作用范围。如果不设置过期时间，则表示这个cookie的生命期为浏览器会话期间，只要关闭浏览器窗口，cookie就消失了。这种生命期为浏览器会话期的cookie被称为会话cookie。会话cookie一般不存储在硬盘上而是保存在内存里，当然这种行为并不是规范规定的。如果设置了过期时间，浏览器就会把cookie保存到硬盘上，关闭后再次打开浏览器，这些cookie仍然有效直到超过设定的过期时间。

存储在硬盘上的cookie可以在不同的浏览器进程间共享，比如两个IE窗口。而对于保存在内存里的cookie，不同的浏览器有不同的处理方式。对于IE，在一个打开的窗口上按Ctrl-N（或者从文件菜单）打开的窗口可以与原窗口共享，而使用其他方式新开的IE进程则不能共享已经打开的窗口的内存cookie；对于Mozilla Firefox0.8，所有的进程和标签页都可以共

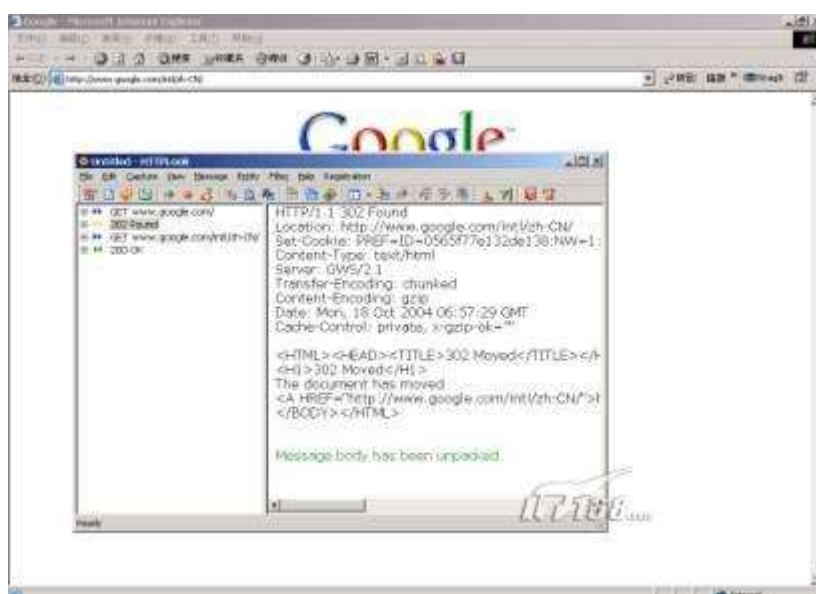
享同样的cookie。一般来说是用javascript的window.open打开的窗口会与原窗口共享内存cookie。浏览器对于会话cookie的这种只认cookie不认人的处理方式经常给采用session机制的web应用程序开发者造成很大的困扰。

下面就是一个google设置cookie的响应头的例子

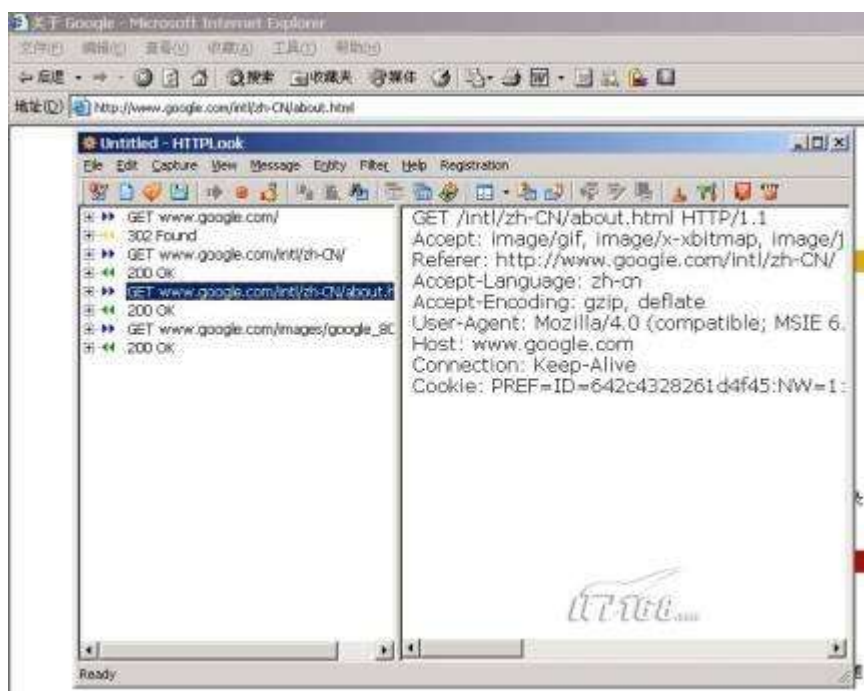
HTTP/1.1 302 Found

Location: <http://www.google.com/intl/zh-CN/>

Set-Cookie: PREF=ID=0565f77e132de138:NW=1:TM=1098082649:LM=1098082649:S=KaeaCFPo49RiA_d8;
expires=Sun, 17-Jan-2038 19:14:07 GMT; path=/; domain=.google.com
Content-Type: text/html



这是使用HTTPLook这个HTTP Sniffer软件来俘获的HTTP通讯纪录的一部分：



浏览器在再次访问google的资源时自动向外发送cookie：



使用Firefox可以很容易的观察现有的cookie的值，使用HTTPLook配合Firefox可以很容易的理解cookie的工作原理。



IE也可以设置在接受cookie前询问：
这是一个询问接受cookie的对话框。

四、理解session机制

session机制是一种服务器端的机制，服务器使用一种类似于散列表的结构（也可能就是使用散列表）来保存信息。

当程序需要为某个客户端的请求创建一个session的时候，服务器首先检查这个客户端的请求里是否已包含了一个session标识 - 称为session id，如果已包含一个session id则说明以前已经为此客户端创建过session，服务器就按照session id把这个session检索出来使用（如果检索不到，可能会新建一个），如果客户端请求不包含session id，则为此客户端创建一个session并且生成一个与此session相关联的session id，session id的值应该是一个既不会重复，又不容易被找到规律以仿造的字符串，这个session id将被在本次响应中返回给客户端保存。

保存这个session id的方式可以采用cookie，这样在交互过程中浏览器可以自动的按照规则把这个标识发给服务器。一般这个cookie的名字都是类似于JSESSIONID，而。比如weblogic对于web应用程序生成的cookie，JSESSIONID=ByOK3vjFD75aPnrF7C2HmdnV6QZcEbZWoWiBYEnLerjQ99zWpBng!-145788764，它的名字就是JSESSIONID。

由于cookie可以被人为的禁止，必须有其他机制以便在cookie被禁止时仍然能够把session id传递回服务器。经常被使用的一种技术叫做URL重写，就是把session id直接附加在URL路径的后面，附加方式也有两种，一种是作为URL路径的附加信息，表现形式为http://...../xxx;jsessionid=ByOK3vjFD75aPnrF7C2HmdnV6QZcEbZWoWiBYEnLerjQ99zWpBng!-145788764

另一种是作为查询字符串附加在URL后面，表现形式为http://...../xxx?jsessionid=ByOK3vjFD75aPnrF7C2HmdnV6QZcEbZWoWiBYEnLerjQ99zWpBng!-145788764

这两种方式对于用户来说是没有区别的，只是服务器在解析的时候处理的方式不同，采用第一种方式也有利于把session id的信息和正常程序参数区分开来。

为了在整个交互过程中始终保持状态，就必须在每个客户端可能请求的路径后面都包含这个session id。

另一种技术叫做表单隐藏字段。就是服务器会自动修改表单，添加一个隐藏字段，以便在表单提交时能够把session id传递回服务器。比如下面的表单：

```
<form name="testform" action="/xxx">
<input type="text">
</form>
```

在被传递给客户端之前将被改写成：

```
<form name="testform" action="/xxx">
<input type="hidden" name="jsessionid"
value="ByOK3vjFD75aPnrF7C2HmdnV6QZcEbZWoWiBYEnLerjQ99zWpBng!-145788764"
>
<input type="text">
</form>
```

这种技术现在已较少应用，笔者接触过的很古老的iPlanet6(SunONE应用服务器的前身)就使用了这种技术。

实际上这种技术可以简单的用对action应用URL重写来代替。

在谈论session机制的时候，常常听到这样一种误解“只要关闭浏览器，session就消失了”。其实可以想象一下会员卡的例子，除非顾客主动对店家提出销卡，否则店家绝对不会轻易删除顾客的资料。对session来说也是一样的，除非程序通知服务器删除一个session，否则服务器会一直保留，程序一般都是在用户做log off的时候发个指令去删除session。然而

浏览器从来不会主动在关闭之前通知服务器它将要关闭，因此服务器根本不会有机会知道浏览器已经关闭，之所以会有这种错觉，是大部分session机制都使用会话cookie来保存session id，而关闭浏览器后这个session id就消失了，再次连接服务器时也就无法找到原来的session。如果服务器设置的cookie被保存到硬盘上，或者使用某种手段改写浏览器发出的HTTP请求头，把原来的session id发送给服务器，则再次打开浏览器仍然能够找到原来的session。

恰恰是由于关闭浏览器不会导致session被删除，迫使服务器为session设置了一个失效时间，当距离客户端上一次使用session的时间超过这个失效时间时，服务器就可以认为客户端已经停止了活动，才会把session删除以节省存储空间。

五、理解javax.servlet.http.HttpSession

HttpSession是Java平台对session机制的实现规范，因为它仅仅是个接口，具体到每个web应用服务器的提供商，除了对规范支持之外，仍然会有一些规范里没有规定的细微差异。这里我们以BEA的Weblogic Server8.1作为例子来演示。

首先，Weblogic Server提供了一系列的参数来控制它的HttpSession的实现，包括使用cookie的开关选项，使用URL重写的开关选项，session持久化的设置，session失效时间的设置，以及针对cookie的各种设置，比如设置cookie的名字、路径、域，cookie的生存时间等。

一般情况下，session都是存储在内存里，当服务器进程被停止或者重启的时候，内存里的session也会被清空，如果设置了session的持久化特性，服务器就会把session保存到硬盘上，当服务器进程重新启动或这些信息将能够被再次使用，Weblogic Server支持的持久性方式包括文件、数据库、客户端cookie保存和复制。

复制严格说来不算持久化保存，因为session实际上还是保存在内存里，不过同样的信息被复制到各个cluster内的服务器进程中，这样即使某个服务器进程停止工作也仍然可以从其他进程中取得session。

cookie生存时间的设置则会影响浏览器生成的cookie是否是一个会话cookie。默认是使用会话cookie。有兴趣的可以用它来试验我们在第四节里提到的那个误解。

cookie的路径对于web应用程序来说是一个非常重要的选项，Weblogic Server对这个选项的默认处理方式使得它与其他服务器有明显的区别。后面我们会专题讨论。

关于session的设置参考[5] http://e-docs.bea.com/wls/docs70/webapp/weblogic_xml.html#1036869

六、HttpSession常见问题（在本小节中session的含义为☉和◎的混合）

1、session在何时被创建

一个常见的误解是以为session在有客户端访问时就被创建，然而事实是直到某server端程序调用`HttpServletRequest.getSession(true)`这样的语句时才被创建，注意如果JSP没有显示的使用`<%@page session="false"%>`关闭session，则JSP文件在编译成Servlet时将会自动加上这样一条语句`HttpSession session = HttpServletRequest.getSession(true);`这也是JSP中隐含的session对象的来历。

由于session会消耗内存资源，因此，如果不打算使用session，应该在所有的JSP中关闭它。

2、session何时被删除

综合前面的讨论，session在下列情况下被删除a.程序调用`HttpSession.invalidate()`;或b.距离上一次收到客户端发送的session id时间间隔超过了session的超时设置;或c.服务器进程被停止（非持久session）

3、如何做到在浏览器关闭时删除session

严格的讲，做不到这一点。可以做一点努力的办法是在所有的客户端页面里使用javascript代码`window.onclose`来监视浏览器的关闭动作，然后向服务器发送一个请求来删除session。但是对于浏览器崩溃或者强行杀死进程这些非常规手段仍然无能为力。

4、有个HttpSessionListener是怎么回事

你可以创建这样的listener去监控session的创建和销毁事件，使得在发生这样的事件时你可以做一些相应的工作。注意是session的创建和销毁动作触发listener，而不是相反。类似的与HttpSession有关的listener还有`HttpSessionBindingListener`，`HttpSessionActivationListener`和`HttpSessionAttributeListener`。

5、存放在session中的对象必须是可序列化的吗

不是必需的。要求对象可序列化只是为了session能够在集群中被复制或者能够持久保存或者在必要时server能够暂时把session交换出内存。在Weblogic Server的session中放置一个不可序列化的对象在控制台上会收到一个警告。我所用过的某个iPlanet版本如果session中有不可序列化的对象，在session销毁时会会有一个Exception，很奇怪。

6、如何才能正确的应付客户端禁止cookie的可能性

对所有的URL使用URL重写，包括超链接，form的action，和重定向的URL，具体做法参见[6]

<http://e-docs.bea.com/wls/docs70/webapp/sessions.html#100770>

7、开两个浏览器窗口访问应用程序会使用同一个session还是不同的session

参见第三小节对cookie的讨论，对session来说是只认id不认人，因此不同的浏览器，不同的窗口打开方式以及不同的cookie存储方式都会对这个问题的答案有影响。

8、如何防止用户打开两个浏览器窗口操作导致的session混乱

这个问题与防止表单多次提交是类似的，可以通过设置客户端的令牌来解决。就是在服务器每次生成一个不同的id返回给客户端，同时保存在session里，客户端提交表单时必须把这个id也返回服务器，程序首先比较返回的id与保存在session里的值是否一致，如果不一致则说明本次操作已经被提交过了。可以参看《J2EE核心模式》关于表示层模式的部分。需要注意的是对于使用javascript window.open打开的窗口，一般不设置这个id，或者使用单独的id，以防主窗口无法操作，建议不要再window.open打开的窗口里做修改操作，这样就可以不用设置。

9、为什么在Weblogic Server中改变session的值后要重新调用一次session.setValue

做这个动作主要是为了在集群环境中提示Weblogic Server session中的值发生了改变，需要向其他服务器进程复制新的session值。

10、为什么session不见了

排除session正常失效的因素之外，服务器本身的可能性应该是微乎其微的，虽然笔者在iPlanet6SP1加若干补丁的Solaris版本上倒也遇到过；浏览器插件的可能性次之，笔者也遇到过3721插件造成的问题；理论上防火墙或者代理服务器在cookie处理上也有可能会出现问

题。出现这一问题的大部分原因都是程序的错误，最常见的就是在一个应用程序中去访问另外一个应用程序。我们在下一节讨论这个问题。

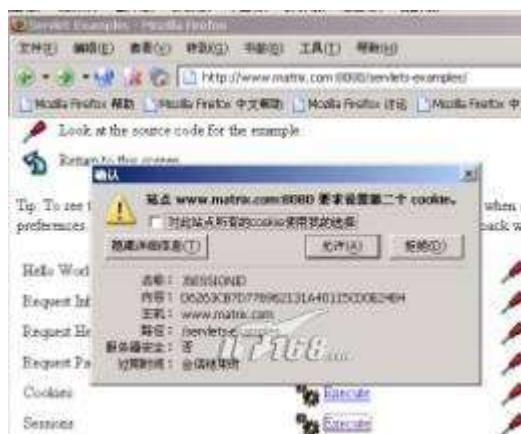
七、跨应用程序的session共享

常常有这样的情况，一个大项目被分割成若干小项目开发，为了能够互不干扰，要求每个小项目作为一个单独的web应用程序开发，可是到了最后突然发现某几个小项目之间需要共享一些信息，或者想使用session来实现SSO(single sign on)，在session中保存login的用户信息，最自然的要求是应用程序间能够访问彼此的session。

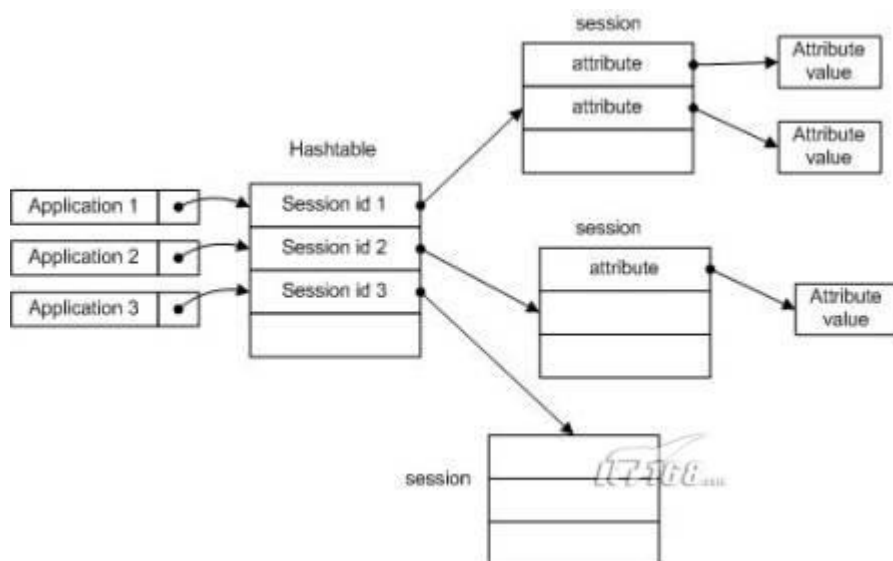
然而按照Servlet规范，session的作用范围应该仅仅限于当前应用程序下，不同的应用程序之间是不能够互相访问对方的session的。各个应用服务器从实际效果上都遵守了这一规范，但是实现的细节却可能各有不同，因此解决跨应用程序session共享的方法也各不相同。

首先来看一下Tomcat是如何实现web应用程序之间session的隔离的，从Tomcat设置的

cookie路径来看，它对不同的应用程序设置的cookie路径是不同的，这样不同的应用程序所用的session id是不同的，因此即使在同一个浏览器窗口里访问不同的应用程序，发送给服务器的session id也可以是不同的。



根据这个特性，我们可以推测Tomcat中session的内存结构大致如下。



笔者以前用过的iPlanet也采用的是同样的方式，估计SunONE与iPlanet之间不会有太大的差别。对于这种方式的服务器，解决思路很简单，实际实行起来也不难。要么让所有的应用程序共享一个session id，要么让应用程序能够获得其他应用程序的session id。

iPlanet中有一种很简单的方法来实现共享一个session id，那就是把各个应用程序的cookie路径都设为/（实际上应该是/NASApp，对于应用程序来讲它的作用相当于根）。

```
<session-info>
<path>/NASApp</path>
</session-info>
```

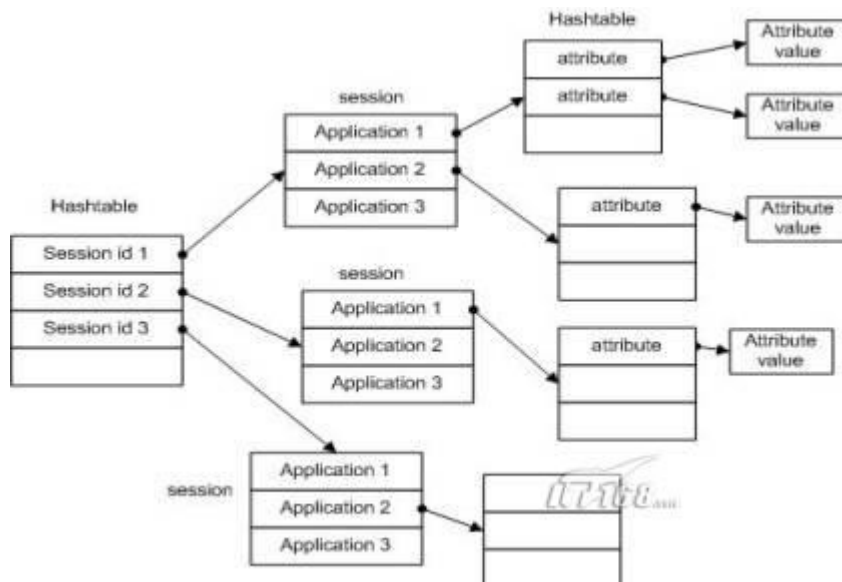
需要注意的是，操作共享的session应该遵循一些编程约定，比如在session attribute名字的前面加上应用程序的前缀，使得setAttribute("name", "neo")变成setAttribute("app1.name", "neo")，以防止命名空间冲突，导致互相覆盖。

在Tomcat中则没有这么方便的选择。在Tomcat版本3上，我们还可以有一些手段来共享session。对于版本4以上的Tomcat，目前笔者尚未发现简单的办法。只能借助于第三方的力量，比如使用文件、数据库、JMS或者客户端cookie，URL参数或者隐藏字段等手段。

我们再看一下Weblogic Server是如何处理session的。



从截屏画面上可以看到Weblogic Server对所有的应用程序设置的cookie的路径都是/，这不是意味着在Weblogic Server中默认的就可以共享session了呢？然而一个小实验即可证明即使不同的应用程序使用的是同一个session，各个应用程序仍然只能访问自己所设置的那些属性。这说明Weblogic Server中的session的内存结构可能如下：



对于这样一种结构，在session机制本身上来解决session共享的问题应该是不可能的了。除了借助于第三方的力量，比如使用文件、数据库、JMS或者客户端cookie，URL参数或者隐藏字段等手段，还有一种较为方便的做法，就是把一个应用程序的session放到ServletContext中，这样另外一个应用程序就可以从ServletContext中取得前一个应用程序的引用。示例代码如下，

应用程序A：

```
context.setAttribute("appA", session);
```

应用程序B：

```
contextA = context.getContext("/appA");
```

```
HttpSession sessionA = (HttpSession)contextA.getAttribute("appA");
```

值得注意的是这种用法不可移植，因为根据ServletContext的JavaDoc，应用服务器可以处于安全的原因对于context.getContext("/appA");返回空值，以上做法在Weblogic Server 8.1中通过。

那么Weblogic Server为什么要把所有的应用程序的cookie路径都设为/呢？原来是为了SSO，凡是共享这个session的应用程序都可以共享认证的信息。一个简单的实验就可以证明这一点，修改首先登录的那个应用程序的描述符weblogic.xml，把cookie路径修改为/appA访问另外一个应用程序会重新要求登录，即使是反过来，先访问cookie路径为/的应用程序，再访问修改过路径的这个，虽然不再提示登录，但是登录的用户信息也会丢失。注意做这个实验时认证方式应该使用FORM，因为浏览器和web服务器对basic认证方式有其他的处理方式，第二次请求的认证不是通过session来实现的。具体请参看[7] section 14.8 Authorization，你可以修改所附的示例程序来做这些试验。

八、总结

session机制本身并不复杂，然而其实现和配置上的灵活性却使得具体情况复杂多变。这也要求我们不能把仅仅某一次的经验或者某一个浏览器，服务器的经验当作普遍适用的经

验，而是始终需要具体情况具体分析。

感谢作者：郎云鹏。

看了tomcat的session处理机制：

Tomcat的Session管理(一) - Session的生成

<http://zddava.iteye.com/blog/311053>

Tomcat的Session管理(二) - Session后台处理

<http://zddava.iteye.com/blog/311136>

tomcat StandardSession

<http://nod0620.iteye.com/blog/1040185>

来自 <<http://justsee.iteye.com/blog/1570652>>

6. OSI七层：

物理层

在OSI参考模型中，物理层（Physical Layer）是参考模型的最低层，也是OSI模型的第一层。

物理层的主要功能是：利用传输介质为数据链路层提供物理连接，实现比特流的透明传输。物理层的作用是使相邻计算机节点之间比特流的透明传送，尽可能屏蔽掉具体传输介质和物理设备的差异。使其上面的数据链路层不必考虑网络的具体传输介质是什么。“透明传送比特流”表示经实际电路传送后的比特流没有发生变化，对传送的比特流来说，这个电路好像是看不见的。

数据链路层

数据链路层（Data Link Layer）是OSI模型的第二层，负责建立和管理节点间的链路。该层的主要功能是：通过各种控制协议，将有差错的物理信道变为无差错的、能可靠传输数据帧的数据链路。

在**计算机网络**中由于各种干扰的存在，物理链路是不可靠的。因此，这一层的主要功能是在物理层提供的比特流的基础上，通过差错控制、流量控制方法，使有差错的物理线路变为无差错的数据链路，即提供可靠的通过物理介质传输数据的方法。

该层通常又被分为**介质访问控制（MAC）**和**逻辑链路控制（LLC）**两个子层。

MAC子层的主要任务是解决共享型网络中多用户对信道竞争的问题，完成网络介质的访问控制；

LLC子层的主要任务是建立和维护网络连接，执行差错校验、流量控制和链路控制。

数据链路层的具体工作是接收来自物理层的位流形式的数据，并封装成帧，传送到上一层；同样，也将来自上层的数据帧，拆装为位流形式的数据转发到物理层；并且，还负责处理接收端发回的确认帧的信息，以便提供可靠的数据传输。

网络层

网络层（Network Layer）是OSI模型的第三层，它是OSI参考模型中**最复杂的一层**，也是通信子网的最高一层。它在下两层的基础上向资源子网提供服务。其主要任务是：通过路由选择**算法**，为报文或分组通过通信子网选择最适当的路径。该层控制数据链路层与传输层之间

的信息转发，建立、维持和终止网络的连接。具体地说，数据链路层的数据在这一层被转换为数据包，然后通过路径选择、分段组合、顺序、进/出路由等控制，将信息从一个网络设备传送到另一个网络设备。

一般地，数据链路层是解决同一网络内节点之间的通信，而网络层主要解决不同子网间的通信。例如在广域网之间通信时，必然会遇到路由（即两节点间可能有多条路径）选择问题。在实现网络层功能时，需要解决的主要问题如下：

寻址：数据链路层中使用的物理地址（如MAC地址）仅解决网络内部的寻址问题。在不同子网之间通信时，为了识别和找到网络中的设备，每一子网中的设备都会被分配一个唯一的地址。由于各子网使用的物理技术可能不同，因此这个地址应当是逻辑地址（如IP地址）。

交换：规定不同的信息交换方式。常见的交换技术有：线路交换技术和存储转发技术，后者又包括报文交换技术和分组交换技术。

路由算法：当源节点和目的节点之间存在多条路径时，本层可以根据路由算法，通过网络为数据分组选择最佳路径，并将信息从最合适的路径由发送端传送到接收端。

连接服务：与数据链路层流量控制不同的是，前者控制的是网络相邻节点间的流量，后者控制的是从源节点到目的节点间的流量。其目的在于防止阻塞，并进行差错检测。

传输层

OSI下3层的主要任务是数据通信，上3层的任务是数据处理。而传输层（Transport Layer）是OSI模型的第4层。因此该层是通信子网和资源子网的接口和桥梁，起到承上启下的作用。该层的主要任务是：向用户提供可靠的端到端的差错和流量控制，保证报文的正确传输。传输层的作用是向高层屏蔽下层数据通信的细节，即向用户透明地传送报文。该层常见的协议：TCP/IP中的TCP协议、Novell网络中的SPX协议和微软的NetBIOS/NetBEUI协议。

传输层提供会话层和网络层之间的传输服务，这种服务从会话层获得数据，并在必要时，对数据进行分割。然后，传输层将数据传递到网络层，并确保数据能正确无误地传送到网络层。因此，传输层负责提供两节点之间数据的可靠传送，当两节点的联系确定之后，传输层则负责监督工作。综上，传输层的主要功能如下：

传输连接管理：提供建立、维护和拆除传输连接的功能。传输层在网络层的基础上为高层提供“面向连接”和“面向无连接”的两种服务。

处理传输差错：提供可靠的“面向连接”和不太可靠的“面向无连接”的数据传输服务、差错控制和流量控制。在提供“面向连接”服务时，通过这一层传输的数据将由目标设备确认，如果在指定的时间内未收到确认信息，数据将被重发。

监控服务质量。

会话层

会话层（Session Layer）是OSI模型的第5层，是用户应用程序和网络之间的接口，主要任务是：向两个实体的表示层提供建立和使用连接的方法。将不同实体之间的表示层的连接称为会话。因此会话层的任务就是组织和协调两个会话进程之间的通信，并对数据交换进行管理。

用户可以按照半双工、单工和全双工的方式建立会话。当建立会话时，用户必须提供他们想要连接的远程地址。而这些地址与MAC（介质访问控制子层）地址或网络层的逻辑地址不同，它们是为用户专门设计的，更便于用户记忆。域名（DN）就是一种网络上使用的远程地址例如：www.3721.com就是一个域名。会话层的具体功能如下：

会话管理：允许用户在两个实体设备之间建立、维持和终止会话，并支持它们之间的数据交

换。例如提供单方向会话或双向同时会话，并管理会话中的发送顺序，以及会话所占时间的长短。

会话流量控制：提供会话流量控制和交叉会话功能。

寻址：使用远程地址建立会话连接。I

出错控制：从逻辑上讲会话层主要负责数据交换的建立、保持和终止，但实际的工作却是接收来自传输层的数据，并负责纠正错误。会话控制和远程过程调用均属于这一层的功能。但应注意，此层检查的错误不是通信介质的错误，而是磁盘空间、打印机缺纸等类型的高级错误。

表示层

表示层 (Presentation Layer) 是OSI模型的第六层，它对来自应用层的命令和数据进行解释，对各种语法赋予相应的含义，并按照一定的格式传送给会话层。其主要功能是“处理用户信息的表示问题，如编码、数据格式转换和加密解密”等。表示层的具体功能如下：

数据格式处理：协商和建立数据交换的格式，解决各应用程序之间在数据格式表示上的差异。

数据的编码：处理字符集和数字的转换。例如由于用户程序中的数据类型（整型或实型、有符号或无符号等）、用户标识等都可以有不同的表示方式，因此，在设备之间需要具有在不同字符集或格式之间转换的功能。

压缩和解压缩：为了减少数据的传输量，这一层还负责数据的压缩与恢复。

数据的加密和解密：可以提高网络的安全性。

应用层

应用层 (Application Layer) 是OSI参考模型的最高层，它是计算机用户，以及各种应用程序和网络之间的接口，其功能是直接向用户提供服务，完成用户希望在网络上完成的各种工作。它在其他6层工作的基础上，负责完成网络中应用程序与网络**操作系统**之间的联系，建立与结束使用者之间的联系，并完成网络用户提出的各种网络服务及应用所需的监督、管理和服务等各种协议。此外，该层还负责协调各个应用程序间的工作。

应用层为用户提供的服务和协议有：文件服务、目录服务、文件传输服务 (FTP)、远程登录服务 (Telnet)、电子邮件服务 (E-mail)、打印服务、安全服务、网络管理服务、**数据库**服务等。上述的各种网络服务由该层的不同应用协议和程序完成，不同的网络操作系统之间在功能、界面、实现技术、对硬件的支持、安全可靠性以及具有的各种应用程序接口等各个方面的差异是很大的。应用层的主要功能如下：

用户接口：应用层是用户与网络，以及应用程序与网络间的直接接口，使得用户能够与网络进行交互式联系。

实现各种服务：该层具有的各种应用程序可以完成和实现用户请求的各种服务。

OSI7层模型的小结

由于OSI是一个理想的模型，因此一般网络系统只涉及其中的几层，很少有系统能够具有所有的7层，并完全遵循它的规定。

在7层模型中，每一层都提供一个特殊的网络功能。从网络功能的角度观察：下面4层（物理层、数据链路层、网络层和传输层）主要提供数据传输和交换功能，即以节点到节点之间的通信为主；第4层作为上下两部分的桥梁，是整个网络体系结构中最关键的部分；而上3层（会话层、表示层和应用层）则以提供用户与应用程序之间的信息和数据处理功能为主。简言之，下4层主要完成通信子网的功能，上3层主要完成资源子网的功能。

来自 <http://blog.csdn.net/yaopeng_2005/article/details/7064869>

第一层: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协议的数据结构和实际物理硬件之间的接口。

第二层: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。

第三层:传输层

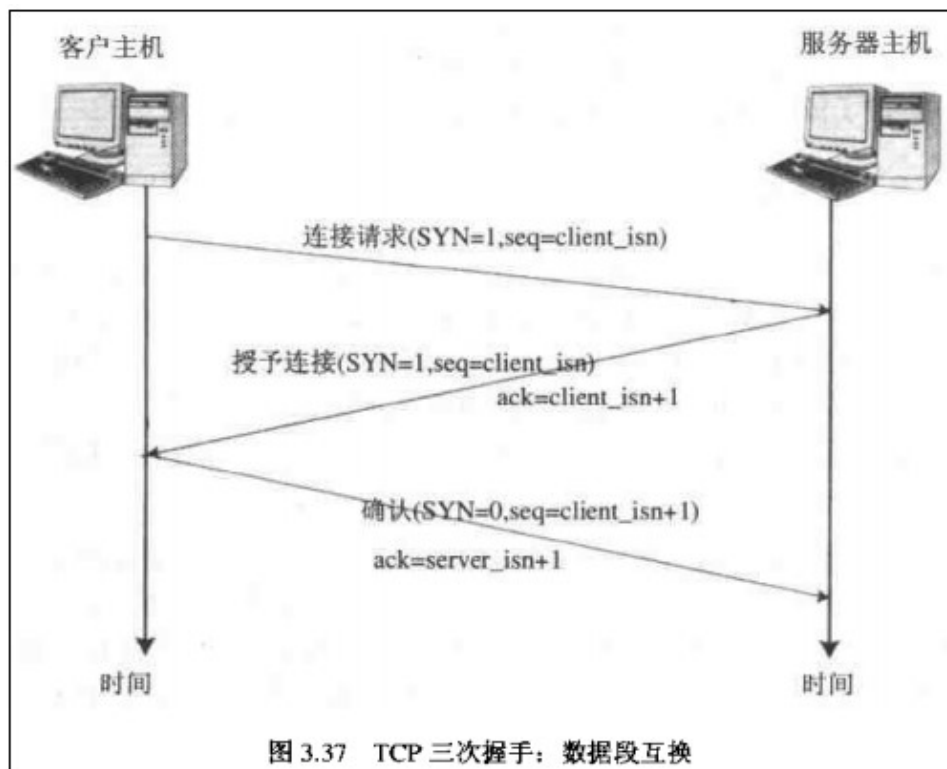
对应于OSI七层参考模型的传输层,它提供两种端到端的通信服务。其中TCP协议(Transmission Control Protocol)提供可靠的数据流运输服务,UDP协议(User Datagram Protocol)提供不可靠的用户数据报服务。

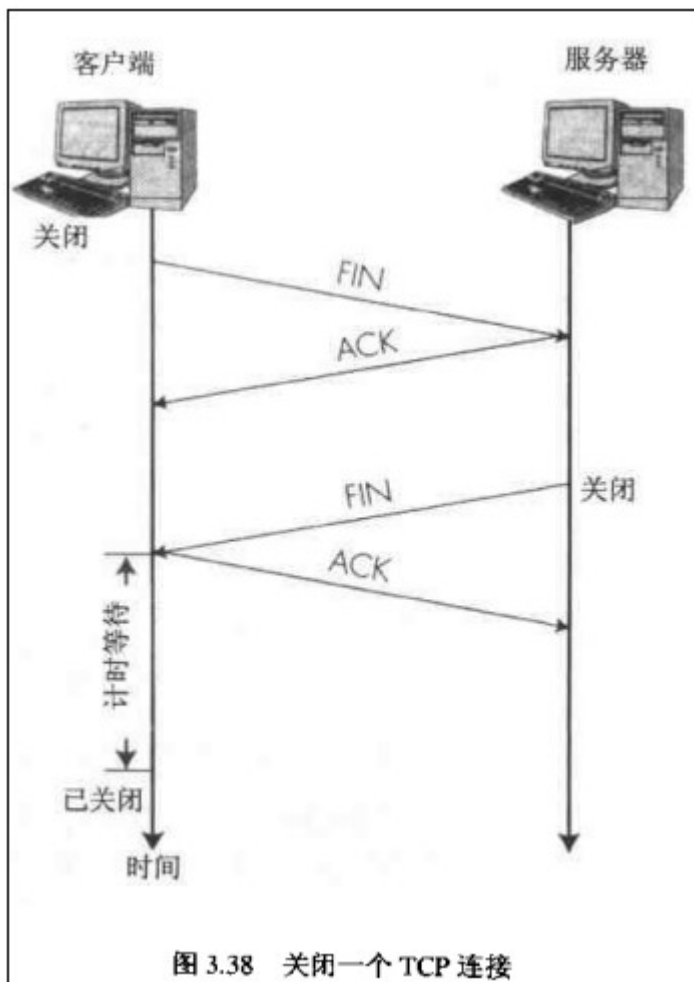
第四层:应用层

对应于OSI七层参考模型的应用层和表达层。因特网的应用层协议包括Finger、Whois、FTP(文件传输协议)、Gopher、HTTP(超文本传输协议)、Telnet(远程终端协议)、SMTP(简单邮件传送协议)、IRC(因特网中继会话)、NNTP(网络新闻传输协议)等

来自 <http://blog.csdn.net/yaopeng_2005/article/details/7064869>

7. 三次握手,四次挥手





因为当Server端收到Client端的SYN连接请求报文后，可以直接发送SYN+ACK报文。其中ACK报文是用来应答的，SYN报文是用来同步的。但是关闭连接时，当Server端收到FIN报文时，很可能并不会立即关闭SOCKET，所以只能先回复一个ACK报文，告诉Client端，“你发的FIN报文我收到了”。只有等到我Server端所有的报文都发送完了，我才能发送FIN报文，因此不能一起发送。故需要四步握手。

来自 <<http://blog.csdn.net/whuslei/article/details/6667471/>>

8. 为什么TIME_WAIT状态需要经过2MSL(最大报文段生存时间)才能返回到CLOSE状态？

答：虽然按道理，四个报文都发送完毕，我们可以直接进入CLOSE状态了，但是我们必须假设网络是不可靠的，有可能最后一个ACK丢失。所以TIME_WAIT状态就是用来重发可能丢失的ACK报文。

来自 <<http://blog.csdn.net/whuslei/article/details/6667471/>>

9. 数据包传输过程

一、ISO七层模型

1.应用层 提供应用程序接口

应用层向应用程序提供服务，这些服务按其向应用程序提供的特性分成组，并称为服务元素。有些可为多种

应用程序共同使用，有些则为较少的一类应用程序使用。应用层是开放系统的最高层，是

直接为应用进程提供
服务的

2.表示层 处理数据格式、数据加密

表示层的作用之一是为异种机通信提供一种公共语言，以便能进行互操作。这种类型的服务之所以需要，是

因为不同的计算机体系结构使用的数据表示法不同。例如，IBM主机使用EBCDIC编码，而大部分PC机使用的

是ASCII码。在这种情况下，便需要会话层来完成这种转换

3.会话层 建立、维护和管理会话

会话层提供的服务可使应用建立和维持会话，并能使会话获得同步。会话层使用校验点可使通信会话在通信

失效时从校验点继续恢复通信。这种能力对于传送大的文件极为重要。会话层、表示层、应用层构成开放系统的

高3层，面对应用进程提供分布处理，对话管理、信息表示、恢复最后的差错等

4.传输层 建立主机端到端连接和数据传输

传输层是两台计算机经过网络进行数据通信时，第一个端到端的层次，具有缓冲作用。当网络层服务质量不能

满足要求时，它将服务加以提高，以满足高层的要求；当网络层服务质量较好时，它只用很少的工作。传输

层还可进行复用，即在一个网络连接上创建多个逻辑连接。传输层也称为运输层。传输层只存在于端开放系统

中，是介于低3层通信子网系统和高3层之间的一层，但是很重要的一层。因为它是源端到目的端对数据传送进行控

制从低到高的最后一层。

5.网络层 路由选择和转发

网络层的产生也是网络发展的结果。在联机系统和线路交换的环境中，网络层的功能没有太大意义。当数据终端

增多时，它们之间有中继设备相连。此时会出现一台终端要求不只是与唯一的一台而是能和多台终端通信的情况，

这就是产生了把任意两台数据终端设备的数据链接起来的问题，也就是路由或者叫寻径。另外，当一条物理信道建

立之后，被一对用户使用，往往有许多空闲时间被浪费掉。人们自然会希望让多对用户共用一条链路，为解决这一

问题就出现了逻辑信道技术和虚拟电路技术

6.数据链路层 提供介质访问、链路管理等

数据链路可以粗略地理解为数据通道。物理层要为终端设备间的数据通信提供传输媒体及其连接,媒体是长期的,

连接是有生存期的,在连接生存期内,收发两端可以进行不等的一次或多次数据通信,每次通信都要经过建立通

信联络和拆除通信联络两过程,这种建立起来的数据收发关系就叫作数据链路,而在物理媒体上传输的数据难免

受到各种不可靠因素的影响而产生差错,为了弥补物理层上的不足,为上层提供无差错的数据传输,就要能对数据

进行检错和纠错,数据链路的建立、拆除,对数据的检错、纠错是数据链路层的基本任务。

7.物理层 比特流传输

物理层是OSI的第一层,它虽然处于最底层,却是整个开放系统的基础。物理层为设备之间的数据通信提供传输

媒体及互连设备,为数据传输提供可靠的环境。

物理层规定了激活、维持、关闭通信端之间的机械特性、电气特性、功能特性及过程特性。虽然物理层不提供

纠错服务,但它能够设定数据传输速率并监测数据出错率。

物理层的媒体包括架空明线、平衡电缆、光纤、无线信道等。通信用的互连设备指DTE和DCE间的互连设备。

DTE即数据终端设备,又称物理设备,如计算机、终端等都包括在内。而DCE则是数据通信设备或电路连接设

备,如调制解调器等。

二、分层的好处

- 1.使人们容易探讨和理解协议的许多细节。
- 2.在各层间标准化接口,允许不同的产品只提供各层功能的一部分,(如路由器在一到三层),或者只提供协议功能的一部分。
- 3.创建更好集成的环境。
- 4.减少复杂性,允许更容易编程改变或快速评估。
- 5.较低的层为较高的层提供服务。
- 6.把复杂的网络划分成为更容易管理的层。

三、TCP/IP协议体系概述

TCP/IP协议源于1969年,是针对Internet开发的一种体系结构和协议标准,目的在于解决异种计算机网络的通信问题。使得网络在互联时能为用户提供一种通用、一致的通信服务。是

Internet采用的协议标准。

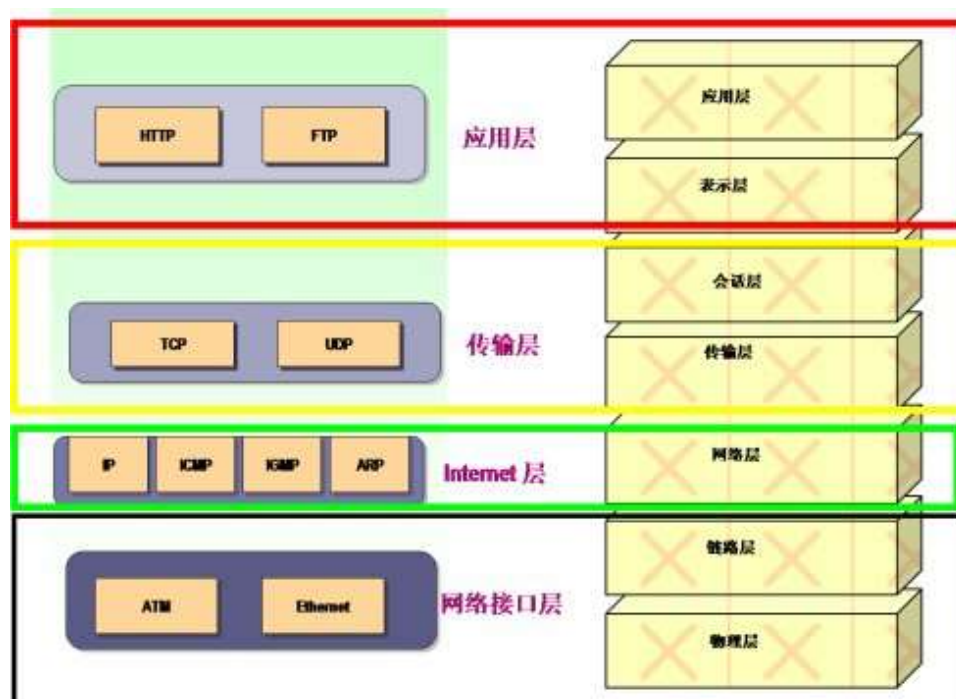
TCP/IP是一组通信协议的代名词，是由一系列协议组成的协议簇。

TCP/IP协议的基本传输单位是数据包（Datagram）。它本身指两个协议集：TCP和IP

TCP（传输控制协议）负责把数据分成若干个数据包，并给每个数据包加上包头，包头上有相应的编号，以保证在数据接收端能将数据还原为原来的格式。

IP（互联网络协议）在每个包头上再加上接收端主机地址，这样数据找到自己要去的地址，如果传输过程中出现数据丢失、失真等情况，TCP协议会自动要求数据重传。

四、TCP/IP各层协议与ISO



TCP/IP协议-应用层协议

- 1、Telnet：它允许一个用户在一个远程的客户机上，访问另一台机器上的资源。
- 2、FTP：文件传输协议实际上就是传输文件的协议，它可以应用在任意两个主机之间。
- 3、TFTP：简单文件传输协议是FTP的简化版本，只有在你确切地知道想得到的文件名及他的准确位置时，才可有选择的使用TFTP。
- 4、SNMP：简单网络管理协议采集并使用一些有价值的网络信息。
- 5、SMTP：简单邮件传输协议，是对应于我们普遍使用的被称为E-mail的应用，
- 6、DNS：域名服务可以解析主机名，特别是Internet名。
- 7、DHCP/BootP：动态主机配置协议可以为主机分配IP地址。
- 8、HTTP：超文本WWW。
- 9、HTTPS：加密WEB通信。他描述了邮件投递中的假脱机、排列及方法

传输层包括两个协议：

TCP协议：

即传输控制协议，是一个可靠的、面向连接的协议。

TCP将数据分成数据报，用能够到达目的地的路径信息连行包装,接收端则将这些数据进行重组。它提供可靠

的、面向连接的数据报传递服务。

TCP协议位于IP协议的上层，为数据提供错误校验，流量控制及序列信息用以补充IP协议的不足。

TCP是面向连接的协议。所谓连接，就是两个对等实体为进行数据通信而进行的一种结合。面向连接服务是

在数据交换之前，必须先建立连接。当数据交换结束后，则应终止这个连接。

面向连接服务具有：连接建立、数据传输和连接释放这三个阶段。在传送数据时是按序传送的。

TCP三次握手后建立连接。

UDP协议：

采用无连接的方式，不管发送的数据包是否到达目的主机，数据包是否出错。收到数据包的主机也不会告诉发送方是否正确收到了数据，它的可靠性是由上层协议来保障的

网络层协议

IP (Internet Protocol)：网际协议

主要负责在主机之间寻址和选择数据包的路由。IP协议不含错误恢复的编码,属于不可靠的协议

ICMP (Internet Control Message Protocol)：网络控制信息协议

ICMP(Internet Control Message Protocol)传递差错报文以及其他需要注意的信息。

ICMP报文通常被网络层或更高层协议（TCP或UDP）使用。一些ICMP报文把差错报文返回给用户进程。

ICMP用来传送一些关于网络和控制信息。如目标主机不可到达、路由重定向等。常用的ping命令就是

使用了ICMP协议。

ICMP不为数据提供错误控制服务，只是报告数据出错并不再传送错误的的数据，并在IP数据报的生存期过后将

其抛弃。

ARP (Address Resolution Protocol)：地址解析协议

ARP (Address Resolution Protocol)把基于TCP/IP软件使用的IP地址解析成局域网硬件使用的媒体访问控制

(MAC)地址。ARP是一个广播协议——网络上的每一台机器都能收到请求。每一台机器都检查请求的IP和自

己的地址，符合要求的主机回答请求。两台主机相互通信必须知道对方的MAC地址，通过ARP协议把IP地址

解析为MAC地址，两者才能够通信

RARP (Reverse Address Resolution Protocol) ：反向地址解析协议

RARP (Reverse Address Resolution Protocol) 一般仅适用于无盘工作站在启动时获取自身IP地址。通常主

机将自己的IP地址存放在硬盘中，无盘工作站因为没有盘无法记忆自己的IP地址。所有无盘工作站的IP地址由

RARP服务器集中保存，无盘工作站启动时通过发送RARP请求，从RARP服务器获得自己的IP地址。

数据链路层协议

点对点协议 (PPP) 为在点对点连接上传输多协议数据包提供了一个标准方法。PPP 最初设计是为两个对等

节点之间的 IP 流量传输提供一种封装协议。在 TCP-IP 协议集中它是一种用来同步调制连接的数据链路层协

议 (OSI 模式中的第二层) ，替代了原来非标准的第二层协议，即 SLIP。

封装：一种封装多协议数据报的方法。PPP 封装提供了不同网络层协议同时在同一链路传输的多路复用技

术。PPP 封装精心设计，能保持对大多数常用硬件的兼容性，克服了SLIP不足之处的一种多用途、点到点协

议，它提供的WAN数据链接封装服务类似于LAN所提供的封闭服务。所以，PPP不仅仅提供帧定界，而且提

供协议标识和位级完整性检查服务。

链路控制协议：一种扩展链路控制协议，用于建立、配置、测试和管理数据链路连接。

网络控制协议：协商该链路上所传输的数据包格式与类型，建立、配置不同的网络层协议；

HDLC——面向比特的同步协议：High Level Data Link Control (高级数据链路控制) 。

HDLC是面向比特的数据链路控制协议的典型代表，该协议不依赖于任何一种字符编码集；数据报文可透明传

输，用于实现透明传输的“0比特插入法”易于硬件实现；全双工通信，有较高的数据链路传输效率；所有帧

采用CRC检验，对信息帧进行顺序编号，可防止漏收或重份，传输可靠性高；传输控制功能与处理功能分离，

具有较大灵活性。

来自 <http://blog.sina.com.cn/s/blog_b43f755b0102vam5.html>

参考：<http://blog.csdn.net/waitforfree/article/details/50771777>

10. http和https区别

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；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；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；

首先谈谈什么是HTTPS

HTTPS(Secure Hypertext Transfer Protocol)是安全超文本传输协议

HTTPS(全称:Hyper Text Transfer Protocol over Secure Socket Layer)，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。它是一个URI scheme(抽象标识符体系)，句法类同http:体系。用于安全的HTTP数据传输。https:URL表明它使用了HTTP，但HTTPS存在不同于HTTP的默认端口及一个加密/身份验证层(在HTTP与TCP之间)。这个系统的最初研发由网景公司(Netscape)进行，并内置于其浏览器Netscape Navigator中，提供了身份验证与加密通讯方法。现在它被广泛用于万维网上安全敏感的通讯，例如交易支付方面。

简介

它是由Netscape开发并内置于其浏览器中，用于对数据进行压缩和解压操作，并返回网络上传送回的结果。HTTPS实际上应用了Netscape的安全套接层（SSL）作为HTTP应用层的子层。（HTTPS使用端口443，而不是像HTTP那样使用端口80来和TCP/IP进行通信。）SSL使用40 位关键字作为RC4流加密[算法](#)，这对于商业信息的加密是合适的。HTTPS和SSL支持使用X.509数字认证，如果需要的话用户可以确认发送者是谁。

也就是说它的主要作用可以分为两种：一种是建立一个信息安全通道，来保证数据传输的安全；另一种就是确认网站的真实性。

HTTPS和HTTP的区别

- 一、https协议需要到ca申请证书，一般免费证书很少，需要交费。
- 二、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。
- 三、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。
- 四、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。

HTTPS解决的问题

信任主机的问题

采用https的服务器必须从CA（Certificate Authority）申请一个用于证明服务器用途类型的证书。该证书只有用于对应的服务器的时候，客户端才信任此主机。所以目前所有的银行系统网站，关键部分应用都是https的。客户通过信任该证书，从而信任了该主机。其实这样做效率很低，但是银行更侧重安全。这一点对我们没有任何异议，我们的服务器，采用的证书不管是自己发布的还是从公众的地方发布的，其客户

端都是自己人，所以我们也就肯定信任该服务器。

通讯过程中的数据的泄密和被篡改

1、一般意义上的https，就是服务器有一个证书。

a) 主要目的是保证服务器就是他声称的服务器，这个跟第一点一样。

b) 服务端和客户端之间的所有通讯，都是加密的。

i. 具体讲，是客户端产生一个对称的密钥，通过服务器的证书来交换密钥，即一般意义上的握手过程。

ii. 接下来所有的信息往来就都是加密的。第三方即使截获，也没有任何意义，因为他没有密钥，当然篡改也就没有什么意义了。

2、少许对客户端有要求的情况下，会要求客户端也必须有一个证书。

a) 这里客户端证书，其实就类似表示个人信息的时候，除了用户名/密码，还有一个CA认证过的身份。因为个人证书一般来说是别人无法模拟的，所以这样能够更深的确认自己的身份。

b) 目前少数个人银行的专业版是这种做法，具体证书可能是拿U盘（即U盾）作为一个备份的载体。

限制

它的安全保护依赖浏览器的正确实现以及服务器软件、实际加密算法的支持。

一种常见的误解是“银行用户在线使用https就能充分彻底保障他们的银行卡号不被偷窃。”实际上，与服务器的加密连接中能保护银行卡号的部分，只有用户到服务器之间的连接及服务器自身。并不能绝对确保服务器自己是安全的，这点甚至已被攻击者利用，常见例子是模仿银行域名的钓鱼攻击。少数罕见攻击在网站传输客户数据时发生，攻击者会尝试窃听传输中的数据。

SSL简介

SSL是Netscape公司所提出的安全保密协议，在浏览器(如Internet Explorer、Netscape Navigator)和Web服务器(如Netscape的Netscape Enterprise Server、ColdFusion Server等等)之间构造安全通道来进行数据传输，SSL运行在TCP/IP层之上、应用层之下，为应用程序提供加密数据通道，它采用了RC4、MD5以及RSA等加密算法，使用40位的密钥，适用于商业信息的加密。

同时，Netscape公司相应开发了HTTPS协议并内置于其浏览器中，HTTPS实际上就是SSL over HTTP，它使用默认端口443，而不是像HTTP那样使用端口80来和TCP/IP进行通信。

HTTPS协议使用SSL在发送方把原始数据进行加密，然后在接受方进行解密，加密和解密需要发送方和接受方通过交换共知的密钥来实现，因此，所传送的数据不容易被网络黑客截获和解密。

然而，加密和解密过程需要耗费系统大量的开销，严重降低机器的性能，相关[测试](#)数据表明使用HTTPS协议传输数据的工作效率只有使用HTTP协议传输的十分之一。

假如为了安全保密，将一个网站所有的Web应用都启用SSL技术来加密，并使用HTTPS协议进行传输，那么该网站的性能和效率将会大大降低，而且没有这个必要，因为一般来说并不是所有数据都要求那么高的安全保密级别。

一般是在支付系统，银行系统对账户信息要求比较高的会采用https协议，总之http效率更好，https安全性很高。

来自 <http://blog.csdn.net/bryant_liu24/article/details/56488603>

11. https是如何保证数据安全的

参考：<http://blog.csdn.net/jasonjwl/article/details/50985271>

12. 如何实现共享session

session共享有很多解决方法，比较常用的如下：

一、以cookie加密的方式保存在客户端.优点是减轻服务器端的压力，缺点是受到cookie的大小限制，可能占用一定带宽，因为每次请求会在头部附带一定大小的cookie信息,另外这种方式在用户禁止使用cookie的情况下无效.

二、服务器间同步。定时同步各个服务器的session信息，此方法可能有一定延时，用户体验也不是很好。

三、以某种媒介共享session信息，比如memcached,NFS等

<http://www.zhihu.com/question/19651970>

1. 通过组播的方式进行集群间的共享，比如tomcat目前就具备这样的功能，优点是web容器自身支持，配置简单，适合小型网站。缺点是当一台机器的上的 session变更后会将变更的数据以组播的形式分发给集群间的所有节点，对网络和所有的web容器都是存在开销。集群越大浪费越严重。不能做到线性的扩展。

2. 利用NFS等一些共享存储来共享Session数据

大致就是有一台公共的NFS服务器做共享服务器，当然也可以采用数据库，所有的Web服务器都把session数据写到共享存储介质上，也都要来这台服务器获取session数据，通过这样的方式来实现Session数据的共享。相比前面组播的方式来说，网络开销较小。缺点是受制于存储设备的依赖，如果存储设备down掉，就无法工作了，要做好主备同步等一些容灾措施。另外，当访问量过大时，磁盘的IO也是一个非常大的问题。

3.利用Memcache来存储共享Session数据

这可能也是目前互联网中比较流行的一种用法。所有Web服务器都把Session写入到memcache，也都从memcache来获取。memcache本身就是一个分布式缓存，便于扩展。网络开销较小，几乎没有IO。性能也更好。缺点，受制于Memcache的容量（除非你有足够内存存储），如果用户量突然增多 cache由于容量的限制会将一些数据挤出缓存，另外memcache故障或重启session会完全丢失掉。

4.完全用cookie

将用户的session数据全部存放在cookie中，很多大型站点都在这么干。优点是服务器架构

也变得简单，每台web服务器都可以很独立。没有网络开销 和对磁盘IO，服务器重启也不会导致数据的丢失。缺点，cookie过于庞大会耗费单位页面的下载时间，所以要尽量保持cookie的精简。

来自 <<http://www.cnblogs.com/crane-practice/p/3662368.html>>

13. TCP超时重传机制

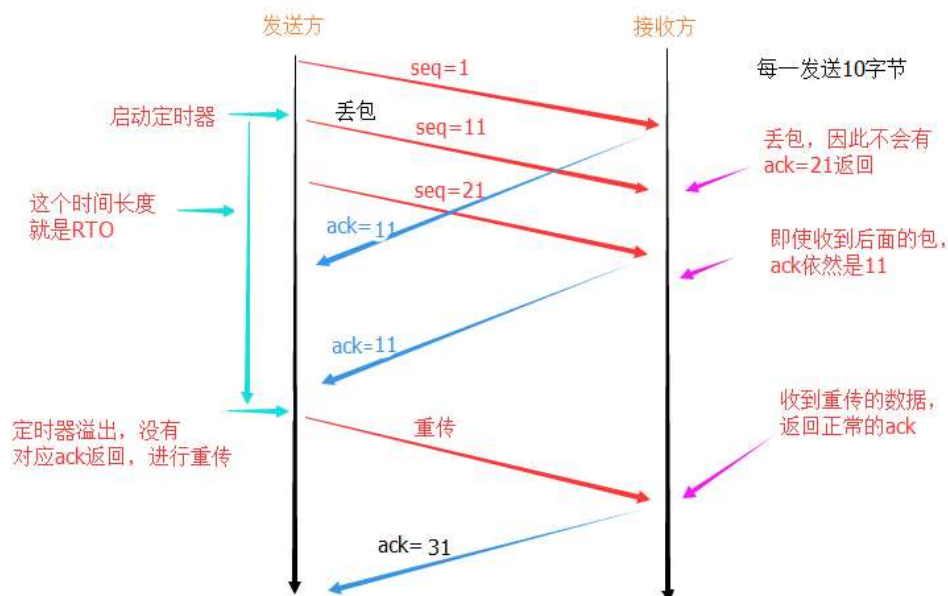
TCP超时与重传机制

TCP协议是一种面向连接的可靠的传输层协议，它保证了数据的可靠传输，对于一些出错，超时丢包等问题TCP设计的超时与重传机制。其基本原理：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。

这里比较重要的是重传超时时间，怎样设置这个定时器的时间（RTO），从而保证对网络资源最小的浪费。因为若RTO太小，可能有些报文只是遇到拥堵或网络不好延迟较大而已，这样就会造成不必要的重传。太大的话，使发送端需要等待过长的时间才能发现数据丢失，影响网络传输效率。

由于不同的网络情况不一样，不可能设置一样的RTO，实际中RTO是根据网络中的RTT（传输往返时间）来自适应调整的。具体关系参考相关[算法](#)。

通过图来了解重传机制：



从图可以知道，发送方连续发送3个数据包，其中第二个丢失，没有被接收到，因此不会返回对应的ACK，没发送一个数据包，就启动一个定时器，当第二个包的定时器溢出了还没有收到ack，这时就进行重传。

TCP慢启动

慢启动是TCP的一个拥塞控制机制，慢启动算法的基本思想是当TCP开始在一个网络中传输数据或发现数据丢失并开始重发时，首先慢慢的对网路实际容量进行试探，避免由于发送了过量的数据而导致阻塞。

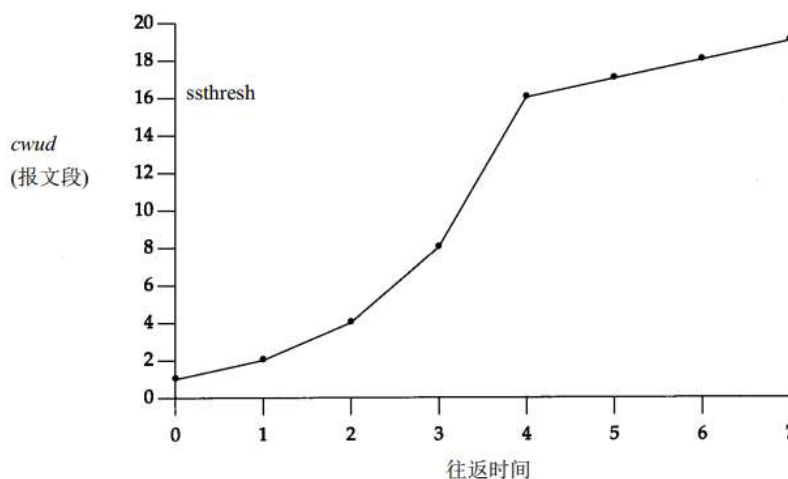
慢启动为发送方的TCP增加了另一个窗口：拥塞窗口(congestion window)，记为 cwnd。当与另一个网络的主机建立TCP连接时，拥塞窗口被初始化为 1 个报文段（即另一端通告的报文段大小）。每收到一个ACK，拥塞窗口就增加一个报文段（cwnd以字节为单位，但是慢启动以报文段大小为单位进行增加）。发送方取拥塞窗口与通告窗口中的最小值作为发送上限。拥塞窗口是发送方使用的流量控制，而通告窗口则是接收方使用的流量控制。发送方开始时发送一个报文段，然后等待 ACK。当收到该 ACK 时，拥塞窗口从1增加为2，即可以发送两个报文段。当收到这两个报文段的 ACK 时，拥塞窗口就增加为4。这是一种指数增加的关系。

拥塞避免算法

网络中拥塞的发生会导致数据分组丢失，需要尽量避免。在实际中，拥塞算法与慢启动通常在一起实现，其基本过程：

1. 对一个给定的连接，初始化cwnd为1个报文段，ssthresh为65535个字节。
2. TCP输出例程的输出不能超过cwnd和接收方通告窗口的大小。拥塞避免是发送方使用的流量控制，而通告窗口则是接收方进行的流量控制。前者是发送方感受到的网络拥塞的估计，而后者则与接收方在该连接上的可用缓存大小有关。
3. 当拥塞发生时（超时或收到重复确认），ssthresh被设置为当前窗口大小的一半（cwnd 和接收方通告窗口大小的最小值，但最少为2个报文段）。此外，如果是超时引起了拥塞，则 cwnd被设置为1个报文段（这就是慢启动）。
4. 当新的数据被对方确认时，就增加cwnd，但增加的方法依赖于是否正在进行慢启动或拥塞避免。如果cwnd小于或等于ssthresh，则正在进行慢启动，否则正在进行拥塞避免。慢启动一直持续到回到当拥塞发生时所处位置的半时候才停止（因为记录了在步骤2 中制造麻烦的窗口大小的一半），然后转为执行拥塞避免。

慢启动算法初始设置cwnd为1个报文段，此后每收到一个确认就加 1。那样，这会使窗口按指数方式增长：发送 1 个报文段，然后是2个，接着是4个……。



慢启动和拥塞避免的可视化描述

在该图中，假定当cwnd为32个报文段时就会发生拥塞。于是设置 ssthresh为16个报文段，而cwnd为1个报文段。在时刻 0 发送了一个报文段，并假定在时刻 1 接收到它的ACK，此时cwnd增加为2。接着发送了2个报文段，并假定在时刻 2 接收到它们的ACK，于是cwnd增加为4（对每个ACK增加1次）。这种指数增加算法一直进行到在时刻3和4之间收到8个ACK后cwnd等于ssthresh时才停止，从该时刻起，cwnd以线

性方式增加，在每个往返时间内最多增加 1 个报文段。

快速重传和快速恢复算法

这是数据丢包的情况下给出的一种修补机制。一般来说，重传发生在超时之后，但是如果发送端接收到 3 个以上的重复 ACK 的情况下（上面的图中第二个包丢失了，就收到了两个相同的 $ack=11$ ），就应该意识到，数据丢了，需要重新传递。这个机制是不需要等到重传定时器溢出的，所以叫做快速重传，它可以避免发送端因等待重传计时器的超时而空闲较长时间，以此增加网络吞吐量。而重新传递以后，因为走的不是慢启动而是拥塞避免算法，所以这又叫做快速恢复算法。算法流程如下：

1. 当收到第 3 个重复的 ACK 时，将 $ssthresh$ 设置为当前拥塞窗口 $cwnd$ 的一半。重传丢失的报文段。设置 $cwnd$ 为 $ssthresh$ 加上 3 倍的报文段大小。
2. 每次收到另一个重复的 ACK 时， $cwnd$ 增加 1 个报文段大小并发送 1 个分组（如果新的 $cwnd$ 允许发送）。
3. 当下一个确认新数据的 ACK 到达时，设置 $cwnd$ 为 $ssthresh$ （在第 1 步中设置的值）。这个 ACK 应该是在进行重传后的一个往返时间内对步骤 1 中重传的确认。另外，这个 ACK 也应该是对丢失的分组和收到的第 1 个重复的 ACK 之间的所有中间报文段的确认。

来自 <<http://blog.csdn.net/ahafg/article/details/51058467>>

14. TCP 怎么保证可靠性

面向连接：意味着两个使用 TCP 的应用（通常是一个客户和一个服务器）在彼此交换数据之前必须先建立一个 TCP 连接。在一个 TCP 连接中，仅有两方进行彼此通信。广播和多播不能用于 TCP。

TCP 通过下列方式来提供可靠性：

- 1、应用数据被分割成 TCP 认为最适合发送的数据块。这和 UDP 完全不同，应用程序产生的数据报长度将保持不变。**(将数据截断为合理的长度)**
- 2、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。**(超时重发)**
- 3、当 TCP 收到发自 TCP 连接另一端的数据，它将发送一个确认。这个确认不是立即发送，通常将推迟几分之一秒。**(对于收到的请求，给出确认响应)** (之所以推迟，可能是要对包做完整校验)
- 4、TCP 将保持它首部和数据的校验和。这是一个端到端的校验和，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。**(校验出包有错，丢弃报文段，不给出响应，TCP 发送数据端，超时时会重发数据)**
- 5、既然 TCP 报文段作为 IP 数据报来传输，而 IP 数据报的到达可能会失序，因此 TCP 报文段的到达也可能会失序。如果必要，TCP 将对收到的数据进行重新排序，将收到的数据以正确的顺序交给应用层。**(对失序数据进行重新排序，然后才交给应用层)**
- 6、既然 IP 数据报会发生重复，TCP 的接收端必须丢弃重复的数据。**(对于重复数据，能够丢弃重复数据)**

7、TCP还能提供流量控制。TCP连接的每一方都有固定大小的缓冲空间。TCP的接收端只允许另一端发送接收端缓冲区所能接纳的数据。这将防止较快主机致使较慢主机的缓冲区溢出。**(TCP可以进行流量控制，防止较快主机致使较慢主机的缓冲区溢出)**TCP使用的流量控制协议是可变大小的滑动窗口协议。

字节流服务::

两个应用程序通过TCP连接交换8bit字节构成的字节流。TCP不在字节流中插入记录标识符。我们将这称为字节流服务 (bytestreamservice) 。

TCP对字节流的内容不作任何解释:: TCP对字节流的内容不作任何解释。TCP不知道传输的数据字节流是二进制数据，还是ASCII字符、EBCDIC字符或者其他类型数据。对字节流的解释由TCP连接双方的应用层解释。

来自 <http://blog.csdn.net/jhh_move_on/article/details/45770087>

? 15. TCP滑动窗口协议

16. 浏览器如何解析一个url

1.首先当然是浏览器输入url地址，

但是当你输入baidu 为什么最终的URL地址是www.baidu.com呢？

当你输入baidu的时候，在url的后面是不是默认的给你添加上.com,如果是这样，哈哈，那这第一步就是没有错误的。

当你的URL输入baidu.com的时候，baidu.com是一个一级域名，那你访问一个域名的时候，在Apache或者nginx上面的配置中就会给你访问一个默认的二级域名（当你没有写完整时），这中间的过程十分的复杂，因为你直接访问baidu.com和访问www.baidu.com不仅仅是名字上的差别，而涉及到了一个重定向的问题，有想深入的孩子可以多查阅一下资料，或者也可以和我交流

2.输入你的url地址之后，浏览器就开始好好工作了，寻找浏览器缓存（可以从浏览器缓存中取数据），系统缓存，路由器（有时候也叫DNS缓存）缓存，看看有没有缓存过这个url中的信息啊，这对于快速反应有着很大的帮助，有缓存就可以直接调用缓存了，没有的话就去访问DNS服务器吧

浏览器缓存参考：[点击打开链接 http://blog.csdn.net/longxibendi/article/details/41630389](http://blog.csdn.net/longxibendi/article/details/41630389)

DNS缓存：首先客户机将域名查询请求发送到本地DNS服务器，本地DNS服务器先在之前的记录（缓存）中查找，如果有缓存，则直接利用缓存进行解析，如果没有缓存，则进入本地的缓存的寻找。

本地缓存：如果本地服务器不能在本地找到缓存，则将请求发送到根域名DNS服务器（全球13台呢哈哈）

本地服务器与网络服务器：本地服务器是内部局域网的设备才能访问，没有公网的IP，网络服务器有公网IP，属于城域网更大的巴拉巴拉，即属于internet

3.请求终于来到了DNS服务器，DNS服务器将域名解析成IP地址，

域名解析：由于网络识别不认识什么域名，这些都是字符组合的，计算机当然不认识，IP地址就认识了，域名解析就是将域名转化成ip地址

DNS：domain name system 域名系统

域名为什么存在呢，就是看一串ip地址会很难记住啊，就有了域名

域名结构：我们来看一个域名

.com 顶级域名 这是全球顶级域名

baidu.com 这是一级域名，一级域名就是在顶级域名前面加上一级

www.baidu.com 这是二级域名，百度的二级域名很多，还有tieba.baidu.com等等，可以

上网搜一搜

<http://>：大家基友疑问了，这不是域名吗？这是一种传输协议，还有很多其他的传输协议，就是网上传输东西的时候遵循的原则

IP地址：互联网上面的每一台主机有一个属于自己的IP地址，用来屏蔽物理性的差异，IP地址就像现实中的住址一样，有了IP就有了目标地了。

4.浏览器有了IP就可以找到服务器，两者之间就可以建立TCP连接，服务器需要和浏览器建立tcp三次握手（打好招呼，要来一发数据了）

TCP连接参考[点击打开链接](http://blog.163.com/hlz_2599/blog/static/142378474201151943414397/)：http://blog.163.com/hlz_2599/blog/static/142378474201151943414397/

三次握手：简化版：甲：你瞅啥 乙：瞅你咋地 甲：不服来一发啊 甲和乙就来一发数据了。

参考：<http://blog.csdn.net/whuslei/article/details/6667471/>（三次握手+四次握手）

5.握手成功后，就可以来一发数据了，不过首先浏览器得向服务器发送http请求（如果是http协议）和请求数据包

http请求就是用什么版本的协议请求，请求的方式是什么，你想要什么数据，这些数据是什么格式，

http请求参考：<http://canrry.iteye.com/blog/1331292>

<http://www.cnblogs.com/loveyakamoz/archive/2011/07/22/2113614.html>

6.请求通过网络，服务器收到了请求，进行处理后，将需要的数据（http响应头）返回浏览器

有请求就要响应，那服务器通过物理地址的取数据和逻辑处理，将数据以响应头的形式返回

数据在网络中传输的过程十分复杂，网络中传输一共有7层，每一层数据的形式有所差异

数据包网络的传输参考：<http://www.tuicool.com/articles/F3Qvie3>

<http://www.cnblogs.com/hnrainll/archive/2012/11/07/2758191.html>

http响应头参考：<http://canrry.iteye.com/blog/1331292>

7.浏览器收到http响应头，此时就要读取数据了，进行浏览器渲染，解析html

来自 <<http://blog.csdn.net/alwanyslongjing/article/details/52601920>>

17. POST和GET的区别

Http定义了与服务器交互的不同方法，最基本的方法有4种，分别是GET，POST，PUT，DELETE。URL全称是资源描述符，我们可以这样认为：一个URL地址，它用于描述一个网络上的资源，而HTTP中的GET，POST，PUT，DELETE就对应着对这个资源的查，改，增，删4个操作。到这里，大家应该有个大概的了解了，GET一般用于获取/查询资源信息，而POST一般用于更新资源信息。

1.根据HTTP规范，GET用于信息获取，而且应该是安全的和幂等的。

(1).所谓安全的意味着该操作用于获取信息而非修改信息。换句话说，GET 请求一般不应产生副作用。就是说，它仅仅是获取资源信息，就像数据库查询一样，不会修改，增加数据，不会影响资源的状态。

* 注意：这里安全的含义仅仅是指是非修改信息。

(2).幂等的意味着对同一URL的多个请求应该返回同样的结果。这里我再解释一下幂等这个概念：



幂等 (idempotent、idempotence) 是一个数学或计算机学概念，常见于抽象代数中。

幂等有一下几种定义：

对于单目运算，如果一个运算对于在范围内的所有的一个数多次进行该运算所得的结果和进行一次该运算所得的结果是一样的，那么我们就称该运算是幂等的。比如绝对值运算就是一个例子，在实数集中，有

$\text{abs}(a) = \text{abs}(\text{abs}(a))$ 。

对于双目运算，则要求当参与运算的两个值是等值的情况下，如果满足运算结果与参与运算的两个值相等，则称该运算幂等，如求两个数的最大值的函数，有在在实数集中幂等，即 $\max(x, x) = x$ 。



看完上述解释后，应该可以理解GET幂等的含义了。

但在实际应用中，以上2条规定并没有这么严格。引用别人文章的例子：比如，新闻站点的头版不断更新。虽然第二次请求会返回不同的一批新闻，该操作仍然被认为是安全的和幂等的，因为它总是返回当前的新闻。从根本上说，如果目标是当用户打开一个链接时，他可以确信从自身的角度来看没有改变资源即可。

2.根据HTTP规范，POST表示可能修改服务器上的资源的请求。继续引用上面的例子：还是新闻以网站为例，读者对新闻发表自己的评论应该通过POST实现，因为在评论提交后站点的资源已经不同了，或者说资源被修改了。

上面大概说了一下HTTP规范中GET和POST的一些原理性的问题。但在实际的做的时候，很多人却没有按照HTTP规范去做，导致这个问题的原因有很多，比如说：

- 1.很多人贪方便，更新资源时用了GET，因为用POST必须要到FORM（表单），这样会麻烦一点。
- 2.对资源的增，删，改，查操作，其实都可以通过GET/POST完成，不需要用到PUT和DELETE。
- 3.另外一个，早期的Web MVC框架设计者们并没有有意识地将URL当作抽象的资源来看待和设计，所以导致一个比较严重的问题是传统的Web MVC框架基本上都只支持GET和POST两种HTTP方法，而不支持PUT和DELETE方法。

* 简单解释一下MVC：MVC本来是存在于Desktop程序中的，M是指数据模型，V是指用户界面，C则是控制器。使用MVC的目的是将M和V的实现代码分离，从而使同一个程序可以使用不同的表现形式。

以上3点典型地描述了老一套的风格（没有严格遵守HTTP规范），随着架构的发展，现在出现REST(Representational State Transfer)，一套支持HTTP规范的新风格，这里不多说了，可以参考《RESTful Web Services》。

说完原理性的问题，我们再从[表面现象上面看看GET和POST的区别](#)：

1.GET请求的数据会附在URL之后（就是把数据放置在HTTP协议头中），以?分割URL和传输数据，参数之间以&相连，如：login.action?
name=hyddd&password=idontknow&verify=%E4%BD%A0%E5%A5%BD。如果数据是英文字母/数字，原样发送，如果是空格，转换为+，如果是中文/其他字符，则直接把字符串用BASE64加密，得出如：%E4%BD%A0%E5%A5%BD，其中%XX中的XX为该符号以16进制表示的ASCII。

POST把提交的数据则放置在是HTTP包的包体中。

2."GET方式提交的数据最多只能是1024字节，理论上POST没有限制，可传较大量的数据，IIS4中最大为80KB，IIS5中为100KB"??！

以上这句是我从其他文章转过来的，其实这样说是错误的，不准确的：

(1).首先是"GET方式提交的数据最多只能是1024字节"，因为GET是通过URL提交数据，那么GET可提交的数据量就跟URL的长度有直接关系了。而实际上，[URL不存在参数上限的问题](#)，[HTTP协议规范没有对URL长度进行限制](#)。这个限制是特定的浏览器及服务器对它的限制。IE对URL长度的限制是2083字节(2K+35)。对于其他浏览器，如Netscape、FireFox等，理论上没有长度限制，其限制取决于操作系统的支持。

注意这是限制是整个URL长度，而不仅仅是你的参数值数据长度。[见参考资料5]

(2).理论上讲，[POST是没有大小限制的](#)，[HTTP协议规范也没有进行大小限制](#)，说“POST数据量存在80K/100K的大小限制”是不准确的，POST数据是没有限制的，起限制作用的是服务器的处理程序的处理能力。

对于ASP程序，Request对象处理每个表单域时存在100K的数据长度限制。但如果使用Request.BinaryRead则没有这个限制。

由这个延伸出去，对于IIS 6.0，微软出于安全考虑，加大了限制。我们还需要注意：

1).IIS 6.0默认ASP POST数据量最大为200KB，每个表单域限制是100KB。

2).IIS 6.0默认上传文件的最大大小是4MB。

3).IIS 6.0默认最大请求头是16KB。

IIS 6.0之前没有这些限制。[见参考资料5]

所以上面的80K，100K可能只是默认值而已(注：关于IIS4和IIS5的参数，我还没有确认)，但肯定是可以自己设置的。由于每个版本的IIS对这些参数的默认值都不一样，具体请参考相关的IIS配置文档。

3.在ASP中，服务端获取GET请求参数用Request.QueryString，获取POST请求参数用Request.Form。在JSP中，用request.getParameter(\"XXXX\")来获取，虽然jsp中也有request.getQueryString()方法，但使用起来比较麻烦，比如：传一个test.jsp?name=hyddd&password=hyddd，用request.getQueryString()得到的是：name=hyddd&password=hyddd。在PHP中，可以用\$_GET和\$_POST分别获取GET和POST中的数据，而\$_REQUEST则可以获取GET和POST两种请求中的数据。值得注意的是，JSP中使用request和PHP中使用\$_REQUEST都会有隐患，这个下次再写个文章总结。

4.POST的安全性要比GET的安全性高。注意：这里所说的安全性和上面GET提到的“安全”不是同个概念。上面“安全”的含义仅仅是不作数据修改，而这里安全的含义是真正的Security的含义，比如：通过GET提交数据，用户名和密码将明文出现在URL上，因为(1)登录页面有可能被浏览器缓存，(2)其他人查看浏览器的历史纪录，那么别人就可以拿到你的账号和密码了，除此之外，使用GET提交数据还可能会造成Cross-site request forgery攻击。

来自 <<http://www.cnblogs.com/hydd/archive/2009/03/31/1426026.html>>

18. HTTP缓存机制

HTTP报文就是浏览器和服务器间通信时发送及响应的数据块。

浏览器向服务器请求数据，发送请求(request)报文；服务器向浏览器返回数据，返回响应(response)报文。

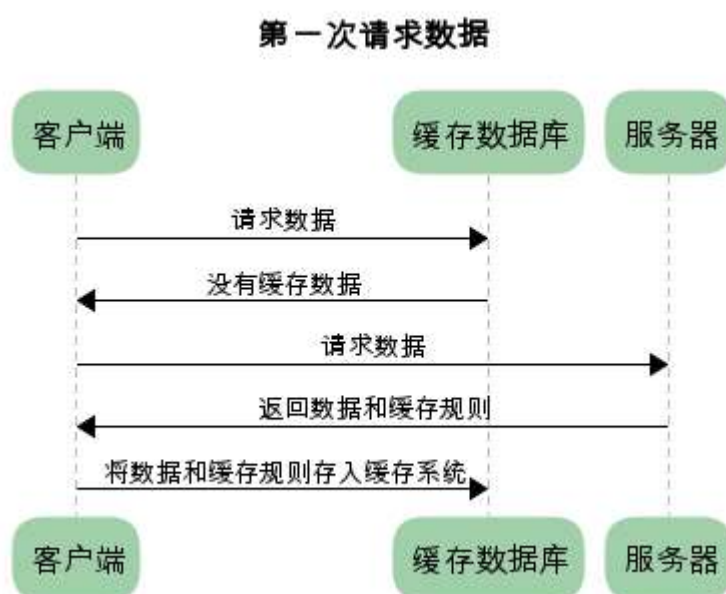
报文信息主要分为两部分

- 1.包含属性的首部(header)-----附加信息（ cookie，缓存信息等）与缓存相关的规则信息，均包含在header中
- 2.包含数据的主体部分(body)-----HTTP请求真正想要传输的部分

缓存规则解析

为方便大家理解，我们认为浏览器存在一个缓存数据库,用于存储缓存信息。

在客户端第一次请求数据时，此时缓存数据库中并没有对应的缓存数据，需要请求服务器，服务器返回后，将数据存储至缓存数据库中。

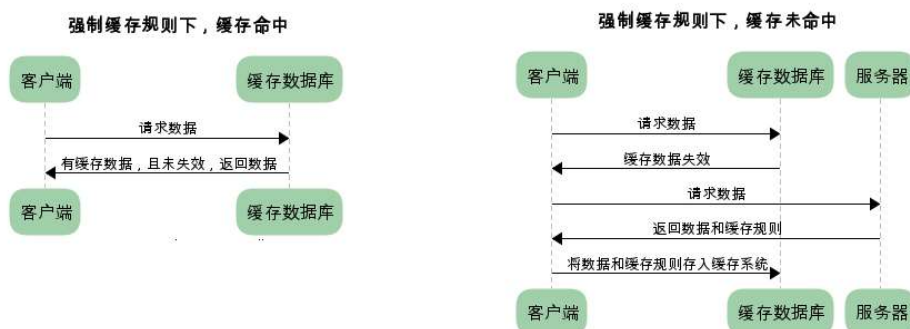


HTTP缓存有多种规则，根据是否需要重新向服务器发起请求来分类，我将其分

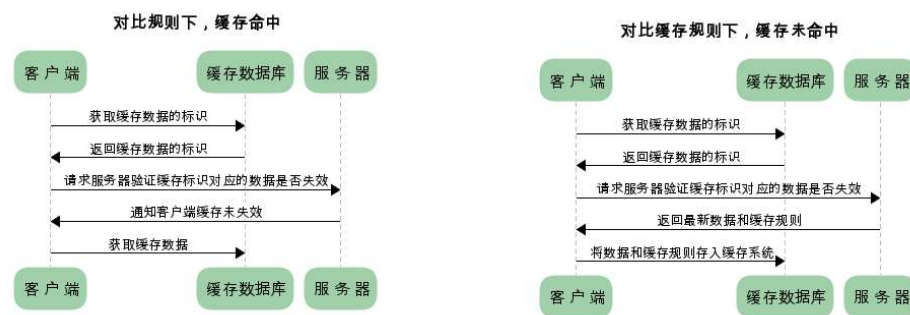
为两大类(**强制缓存**，**对比缓存**)

在详细介绍这两种规则之前，先通过时序图的方式，让大家对这两种规则有个简单了解。

已存在缓存数据时，仅基于**强制缓存**，请求数据的流程如下



已存在缓存数据时，仅基于**对比缓存**，请求数据的流程如下



对缓存机制不太了解的同学可能会问，基于**对比缓存**的流程下，不管是否使用缓存，都需要向服务器发送请求，那么还用缓存干什么？

这个问题，我们暂且放下，后文在详细介绍每种缓存规则的时候，会带大家答案。

我们可以看到两类缓存规则的不同，**强制缓存**如果生效，不需要再和服务器发生交互，而**对比缓存**不管是否生效，都需要与服务端发生交互。

两类缓存规则可以同时存在，**强制缓存**优先级高于**对比缓存**，也就是说，当执行**强制缓存**的规则时，如果缓存生效，直接使用缓存，不再执行**对比缓存**规则。

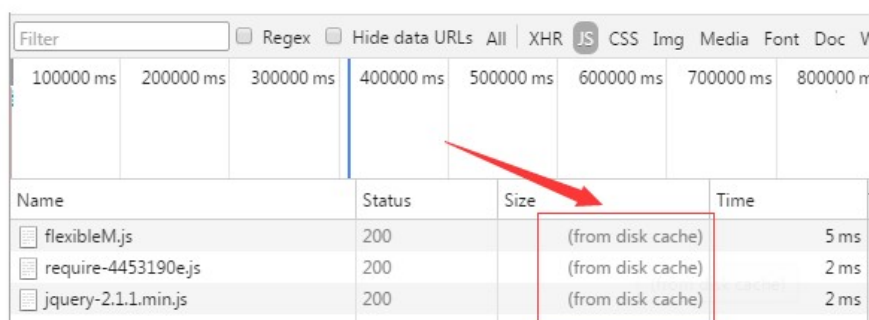
强制缓存

从上文我们得知，强制缓存，在缓存数据未失效的情况下，可以直接使用缓存数据，那么浏览器是**如何判断缓存数据是否失效**呢？

我们知道，在没有缓存数据的时候，浏览器向服务器请求数据时，服务器会将数据和缓存规则一并返回，**缓存规则信息包含在响应header中**。

对于强制缓存来说，响应header中会有两个字段来标明失效规则
(**Expires/Cache-Control**)

使用chrome的开发者工具，可以很明显的看到对于强制缓存生效时，网络请求的情况



The screenshot shows the Chrome DevTools Network tab with the 'JS' filter selected. A red arrow points to the 'Size' column of the network log, which contains the text '(from disk cache)' for three requests: 'flexibleM.js', 'require-4453190e.js', and 'jquery-2.1.1.min.js'. The 'Time' column shows 5 ms, 2 ms, and 2 ms respectively.

Name	Status	Size	Time
flexibleM.js	200	(from disk cache)	5 ms
require-4453190e.js	200	(from disk cache)	2 ms
jquery-2.1.1.min.js	200	(from disk cache)	2 ms

Expires

Expires的值为服务端返回的到期时间，即下一次请求时，请求时间小于服务端返回的到期时间，直接使用缓存数据。

不过Expires 是HTTP 1.0的东西，现在默认浏览器均默认使用HTTP 1.1，所以它的作用基本忽略。

另一个问题是，到期时间是由服务端生成的，但是客户端时间可能跟服务端时间有误差，这就会导致缓存命中的误差。

所以HTTP 1.1 的版本，使用**Cache-Control**替代。

Cache-Control

Cache-Control 是最重要的规则。常见的取值有private、public、no-cache、max-age、no-store，默认为private。

private: 客户端可以缓存

public: 客户端和代理服务器都可缓存（前端的同学，可以认为public和private是一样的）

max-age=xxx: 缓存的内容将在 xxx 秒后失效

no-cache: 需要使用**对比缓存**来验证缓存数据（后面介绍）

no-store: 所有内容都不会缓存，**强制缓存**，**对比缓存**都不会触发（对于前端开发来说，缓存越多越好，so...基本上和它说886）

举个板栗

▼ Response Headers [view source](#)

```
Cache-Control: max-age=31536000
Connection: keep-alive
Content-Encoding: gzip
Content-Type: application/javascript
Date: Tue, 24 Jan 2017 02:21:12 GMT
ETag: W/"58847adf-110d2d"
Last-Modified: Sun, 22 Jan 2017 09:26:55 GMT
```

图中Cache-Control仅指定了max-age，所以默认为private，缓存时间为31536000秒（365天）

也就是说，在365天内再次请求这条数据，都会直接获取缓存数据库中的数据，直接使用。




对比缓存

对比缓存，顾名思义，需要进行比较判断是否可以使用缓存。




浏览器第一次请求数据时，服务器会将缓存标识与数据一起返回给客户端，客户端将二者备份至缓存数据库中。

再次请求数据时，客户端将备份的缓存标识发送给服务器，服务器根据缓存标识进行判断，判断成功后，返回304状态码，通知客户端比较成功，可以使用缓存数据。

第一次访问：

Name	Status	Size	Time
 flexible-c207ebf8.js	200	1.3 KB	103 ms
 base-5c8a15c8.js	200	228 KB	429 ms
 index-6c1a969b.js	200	35.2 KB	151 ms

再次访问：

Name	Status	Size	Time
 flexible-c207ebf8.js	304	206 B	28 ms
 base-5c8a15c8.js	304	209 B	66 ms
 index-6c1a969b.js	304	208 B	37 ms

通过两图的对比，我们可以很清楚的发现，在**对比缓存**生效时，状态码为304，并且报文大小和请求时间大大减少。

原因是，服务端在进行标识比较后，只返回header部分，通过状态码通知客户

端使用缓存，不再需要将报文主体部分返回给客户端。

对于**对比缓存**来说，缓存标识的传递是我们着重需要理解的，它在请求header和响应header间进行传递，

一共分为两种标识传递，接下来，我们分开介绍。

Last-Modified / If-Modified-Since

Last-Modified :

服务器在响应请求时，告诉浏览器资源的最后修改时间。

▼ Response Headers

view source

Cache-Control: max-age=31536000
Connection: keep-alive
Content-Encoding: gzip
Content-Type: application/javascript
Date: Tue, 24 Jan 2017 07:26:54 GMT
ETag: W/"5886c231-8d9"
Last-Modified: Tue, 24 Jan 2017 02:55:45 GMT
Server: TGWEB
Transfer-Encoding: chunked
Vary: Accept-Encoding

第一次请求时，服务器返回的资源最后修改时间

If-Modified-Since :

再次请求服务器时，通过此字段通知服务器上次请求时，服务器返回的资源最后修改时间。

服务器收到请求后发现头If-Modified-Since 则与被请求资源的最后修改时间进行比对。

若资源的最后修改时间大于If-Modified-Since，说明资源又被改动过，则响应整片资源内容，返回状态码200；

若资源的最后修改时间小于或等于If-Modified-Since，说明资源无新修改，则响应HTTP 304，告知浏览器继续使用所保存的cache。

▼ Request Headers

view source

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate, sdch
Accept-Language: zh-CN,zh;q=0.8
Cache-Control: max-age=0
Connection: keep-alive
Host: m.51tiangou.com
If-Modified-Since: Tue, 24 Jan 2017 02:55:45 GMT
If-None-Match: W/"5886c231-8d9"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0

再次请求时，浏览器通知服务器，上次请求时返回的资源最后修改时间

Etag / If-None-Match (优先级高于Last-Modified / If-Modified-Since)

Etag :

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（生成规则由服务器决定）。

▼ Response Headers	view source
Cache-Control: max-age=31536000	
Connection: keep-alive	
Content-Encoding: gzip	
Content-Type: application/javascript	
Date: Tue, 24 Jan 2017 07:26:54 GMT	
Etag: W/"5886c231-8d9"	第一次请求时，服务器返回的资源唯一标识
Last-Modified: Tue, 24 Jan 2017 02:55:45 GMT	
Server: TGWEB	
Transfer-Encoding: chunked	
Vary: Accept-Encoding	

If-None-Match :

再次请求服务器时，通过此字段通知服务器客户端缓存数据的唯一标识。

服务器收到请求后发现头If-None-Match 则与被请求资源的唯一标识进行比对，

不同，说明资源又被改动过，则响应整片资源内容，返回状态码200；

相同，说明资源无新修改，则响应HTTP 304，告知浏览器继续使用所保存的cache。

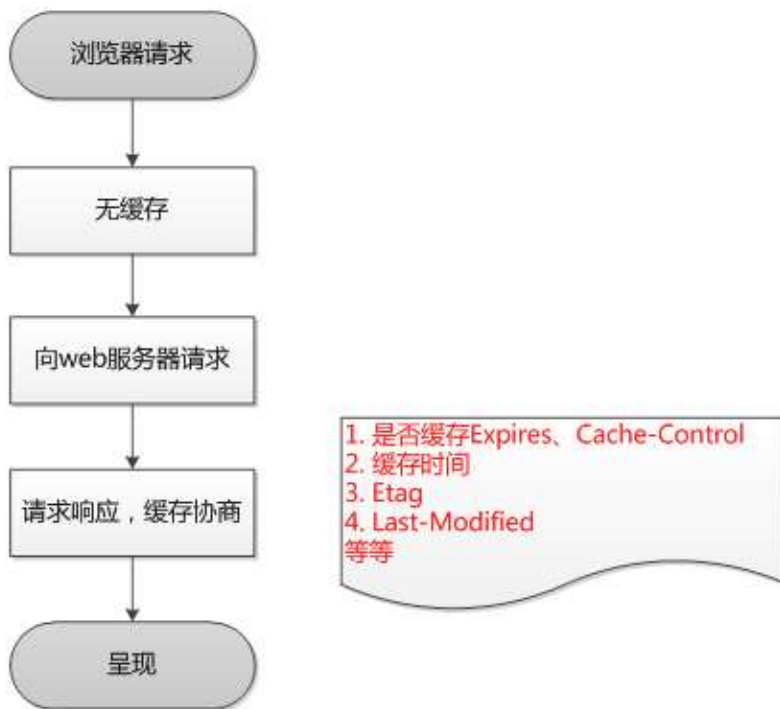
▼ Request Headers	view source
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8	
Accept-Encoding: gzip, deflate, sdch	
Accept-Language: zh-CN,zh;q=0.8	
Cache-Control: max-age=0	
Connection: keep-alive	
Host: m.51tiangou.com	
If-Modified-Since: Tue, 24 Jan 2017 02:55:45 GMT	
If-None-Match: W/"5886c231-8d9"	再次请求时，浏览器通知服务器上次返回的资源唯一标识
Upgrade-Insecure-Requests: 1	
User-Agent: Mozilla/5.0	

总结

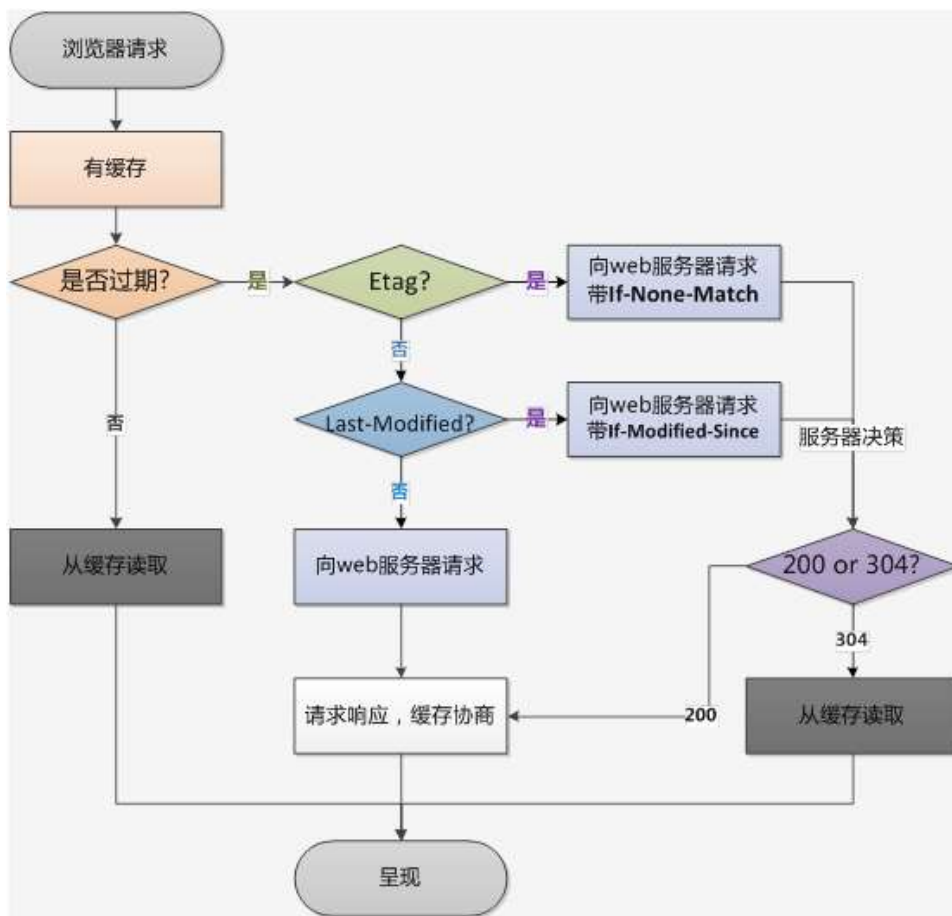
对于强制缓存，服务器通知浏览器一个缓存时间，在缓存时间内，下次请求，直接用缓存，不在时间内，执行比较缓存策略。

对于比较缓存，将缓存信息中的Etag和Last-Modified通过请求发送给服务器，由服务器校验，返回304状态码时，浏览器直接使用缓存。

浏览器第一次请求：



浏览器再次请求时：



来自 <<http://www.cnblogs.com/chengf/p/6386163.html>>

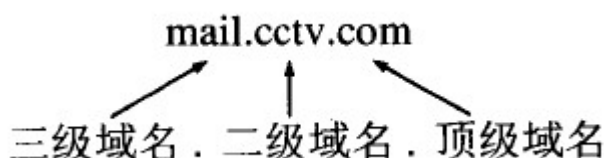
19. DNS解析过程

因特网的域名结构

由于因特网的用户数量较多，所以因特网在命名时采用的是层次树状结构的命名方法。任何一个连接在因特网上的主机或路由器，都有一个唯一的层次结构的名称，即域名(domain name)。这里，“域”(domain)是名字空间中一个可被管理的划分。

从语法上讲，每一个域名都是有标号(label)序列组成，而各标号之间用点(小数点)隔开。

如下例子所示：



这是中央电视台用于收发电子邮件的计算机的域名，它由三个标号组成，其中标号com是顶级域名，标号cctv是二级域名，标号mail是三级域名。

DNS规定，域名中的标号都有英文和数字组成，每一个标号不超过63个字符(为了记忆方便，一般不会超过12个字符)，也不区分大小写字母。标号中除连字符(-)外不能使用其他的标点符号。级别最低的域名写在最左边，而级别最高的字符写在最右边。由多个标号组成的完整域名总共不超过255个字符。DNS既不规定一个域名需要包含多少个下级域名，也不规定每一级域名代表什么意思。各级域名由其上一级的域名管理机构管理，而最高的顶级域名则由ICANN进行管理。用这种方法可使每一个域名在整个互联网范围内是唯一的，并且也容易设计出一种查找域名的机制。

域名只是逻辑概念，并不代表计算机所在的物理地点。据2006年12月统计，现在顶级域名TLD(Top Level Domain)已有265个，分为三大类：

(1)国家顶级域名nTLD：采用ISO3166的规定。如：cn代表中国，us代表美国，uk代表英国，等等。国家域名又常记为ccTLD(cc表示国家代码country-code)。

(2)通用顶级域名gTLD：最常见的通用顶级域名有7个，即：com(公司企业)，net(网络服务机构)，org(非营利组织)，int(国际组

织), gov(美国的政府部门), mil(美国的军事部门)。

(3)基础结构域名(infrastructure domain): 这种顶级域名只有一个, 即arpa, 用于反向域名解析, 因此称为反向域名。

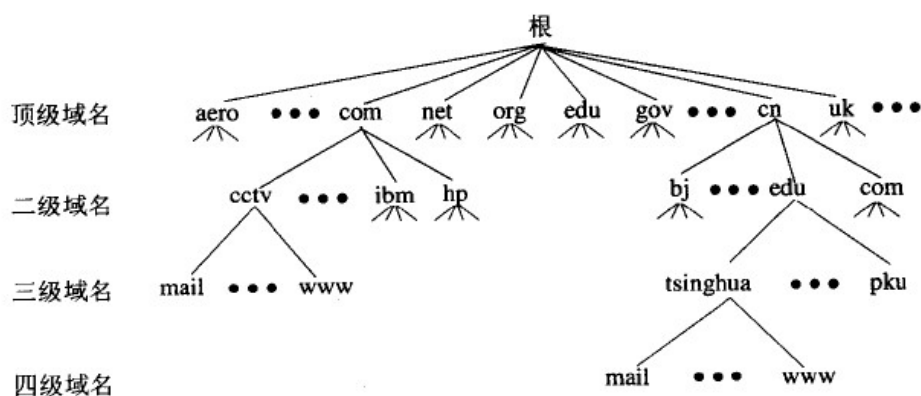


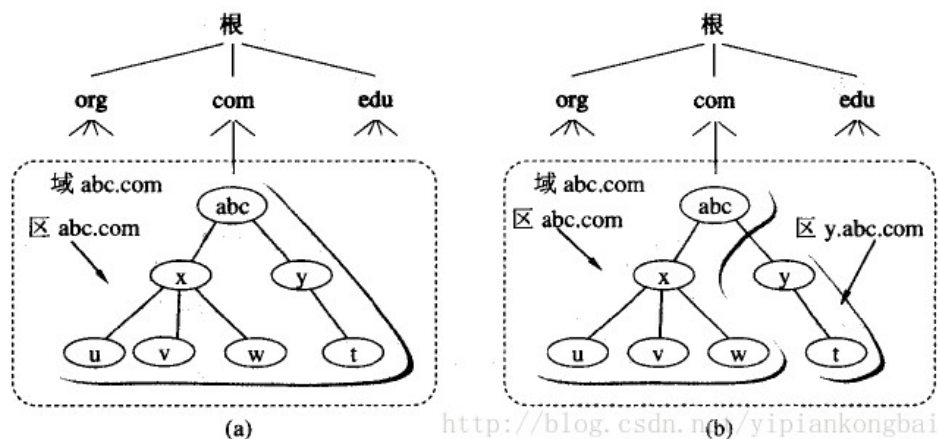
图 6-1 因特网的域名空间

3、域名服务器

如果采用上述的树状结构, 每一个节点都采用一个域名服务器, 这样会使得域名服务器的数量太多, 使域名服务器系统的运行效率降低。所以在DNS中, 采用划分区的方法来解决。

一个服务器所负责管辖(或有权限)的范围叫做区(zone)。各单位根据具体情况来划分自己管辖范围的区。但在一个区中的所有节点必须是能够连通的。每一个区设置相应的权限域名服务器, 用来保存该区中的所有主机到域名IP地址的映射。总之, DNS服务器的管辖范围不是以“域”为单位, 而是以“区”为单位。区是DNS服务器实际管辖的范围。区 ≤ 域。

下图是区的不同划分方法的举例。假定abc公司有下属部门x和y, 部门x下面有分三个分布们u,v,w, 而y下面还有下属部门t。图a表示abc公司只设一个区abc.com。这是, 区abc.com和域abc.com指的是同一件事。但图b表示abc公司划分为两个区: abc.com和y.abc.com。这两个区都隶属于域abc.com, 都各设置了相应的权限域名服务器。不难看出, 区是域的子集。



下图是以上图b中abc公司划分的两个区为例，给出了DNS域名服务器树状结构图。这种DNS域名服务器树状结构图可以更准确地反映出DNS的分布式结构。图中的每一个域名服务器都能够部分域名到IP地址的解析。当某个DNS服务器不能进行域名到IP地址的转换时，它就会设法找因特网上别的域名服务器进行解析。

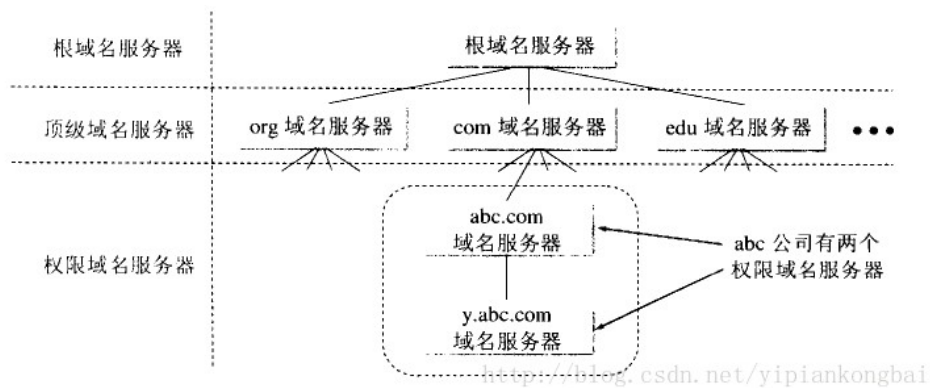
从下图可以看出，因特网上的DNS服务器也是按照层次安排的。每一个域名服务器只对域名体系中的一部分进行管辖。根据域名服务器所起的作用，可以把域名服务器划分为下面四种不同的类型。

根域名服务器：最高层次的域名服务器，也是最重要的域名服务器。所有的根域名服务器都知道所有的顶级域名服务器的域名和IP地址。不管是哪一个本地域名服务器，若要对因特网上任何一个域名进行解析，只要自己无法解析，就首先求助根域名服务器。所以根域名服务器是最重要的域名服务器。假定所有的根域名服务器都瘫痪了，那么整个DNS系统就无法工作。需要注意的是，在很多情况下，根域名服务器并不直接把待查询的域名直接解析出IP地址，而是告诉本地域名服务器下一步应当找哪一个顶级域名服务器进行查询。

顶级域名服务器：负责管理在该顶级域名服务器注册的二级域名。

权限域名服务器：负责一个“区”的域名服务器。

本地域名服务器：本地服务器不属于下图的域名服务器的层次结构，但是它对域名系统非常重要。当一个主机发出DNS查询请求时，这个查询请求报文就发送给本地域名服务器。



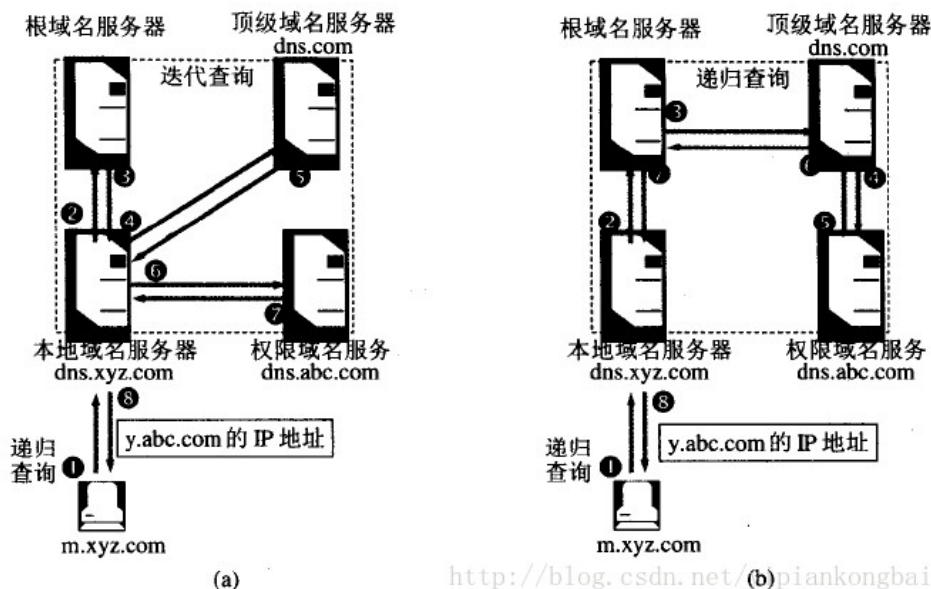
4、域名的解析过程

注意：

一、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。

二、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。

下图给出了这两种查询的差别



下面举一个例子演示整个查询过程：

假定域名为m.xyz.com的主机想知道另一个主机y.abc.com的IP地址。例如，主机m.xyz.com打算发送邮件给y.abc.com。这时就必须知道主机y.abc.com的IP地址。下面是上图a的几个查询步骤：

- 1、主机m.abc.com先向本地服务器dns.xyz.com进行递归查询。
- 2、本地服务器采用迭代查询。它先向一个根域名服务器查询。
- 3、根域名服务器告诉本地服务器，下一次应查询的顶级域名服务器dns.com的IP地址。
- 4、本地域名服务器向顶级域名服务器dns.com进行查询。
- 5、顶级域名服务器dns.com告诉本地域名服务器，下一步应查询的权限服务器dns.abc.com的IP地址。
- 6、本地域名服务器向权限域名服务器dns.abc.com进行查询。
- 7、权限域名服务器dns.abc.com告诉本地域名服务器，所查询的主机的IP地址。
- 8、本地域名服务器最后把查询结果告诉m.xyz.com。

整个查询过程共用到了8个UDP报文。

为了提高DNS查询效率，并减轻服务器的负荷和减少因特网上的DNS查询报文数量，在域名服务器中广泛使用了高速缓存，用来存放最近查询过的域名以及从何处获得域名映射信息的记录。

例如，在上面的查询过程中，如果在m.xyz.com的主机上不久前已经有用户查询过y.abc.com的IP地址，那么本地域名服务器就不必向根域名服务器重新查询y.abc.com的IP地址，而是直接把告诉缓存中存放的上次查询结果(即y.abc.com的IP地址)告诉用户。

由于名字到地址的绑定并不经常改变，为保持告诉缓存中的内容正确，域名服务器应为每项内容设置计时器并处理超过合理时间的项(例如每个项目两天)。当域名服务器已从缓存中删去某项信息后又被请求查询该项信息，就必须重新到授权管理该项的域名服务器绑定信息。当权限服务器回答一个查询请求时，在响应中都指明绑定有效存在的时间值。增加此时间值可减少网络开销，而减少此时间值可提高域名解析的正确性。

不仅在本地域名服务器中需要高速缓存，在主机中也需要。许多主机在启动时从本地服务器下载名字和地址的全部数据库，维护存放自己最近使用的域名的高速缓存，并且只在从缓存中找不到名字时才使用域名服务器。维护本地域名服务器数据库的主机应当定期地检查域名服务器以获取新的映射信息，而且主机必须从缓存中删除无效的项。由于域名改动并不频繁，大多数网点不需花精力就能维护数据库的一致性。

来自 <<http://blog.csdn.net/yipiankongbai/article/details/25031461>>

20.