每个程序员应该知道的计算机网络知识

每个程序员应该知道的计算机网络知识

*2016-09-09收听我*java一日一条 java一日一条

前言

作为一名程序员, 不可能不与网络打交道. 现在我们的手机, 电脑, 不夸张地说, 离开了网络就是一块'废铁', 它们的作用将大打折扣.. 本文的作用呢, 主要是针对不是非网络专业开发的人员准备的, 以'最短的时间, 了解计网最多的知识'为前提起笔.

目录

概述

先来了解下各种我们知道, 但是不太了解的专业名词的意思

因特网

因特网

因特网是当今世界上最大的网络, 是"网络的网络". 即因特网是所有网络互连起来的一个巨型网络.

因特网的组成 :

> 边缘部分 : 主机
>
> 核心部分 : 大量网络和连接这些网络的路由器(此路由器不是我们家用的路由器)

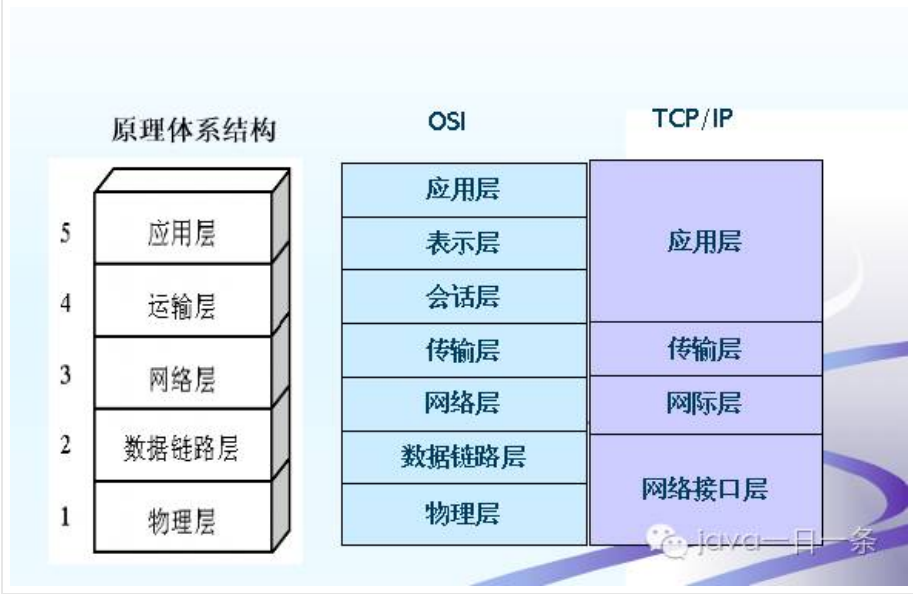
以太网

以太网是现在最常用的局域网通信协议, 以太网上传输的是MAC帧. 由于以太网同一时间只允许一台计算机发送数据, 所以必须有一套检测机制, 那就是CSMA/CD协议 :

> 多点接入 : 多台计算机以多点接入的方式连接在一根总线上
>
> 载波监听 : 不管是否正在发送, 每个站都必须不停地检测信道
>
> 碰撞检测 : 边发送边监听

OSI

开放系统互连基本参考模型, 只要遵守这个OSI标准, 任何两个系统都能进行通信. OSI是七层协议体系结构, 而TCP/IP是一个四层协议体系结构, 于是我们采取折中的方法, 学习计算机网络原理的时候往往用的是五层协议的体系结构 : 物理层, 数据链路层, 网络层, 传输层和应用层



协议体系结构

物理层

计算机的世界里只有0和1, 正如你现在所看这篇文章的文字, 存储在计算机中也是一大串0和1的组合. 但是这些数字不能在真实的物理介质中传输的, 而需要把它转换为光信号或者电信号, 所以这一层负责将这些比特流(0101)与光电信号进行转换.

如果没有物理层, 那么也就不存在互联网, 不存在数据的共享, 因为数据无法在网络中流动.

数据链路层

数据在这一层不再是以比特流的形式传输, 而是分割成一个一个的帧再进行传输.

MAC地址

又称计算机的硬件地址, 被固化在适配器(网卡)ROM上的占48位的地址. MAC地址可以用来唯一区别一台计算机, 因为它在全球是独一无二的

分组交换

由于数据在这次曾要被分割成一个一个的帧, 由于不同的链路规定了不同的最大帧长, 即MTU(最大传输单元), 凡是超出这个MTU的帧都必须被分块. 例如一台货车一次能运输5吨的货物, 而有条公路限载重2吨, 那么你只好分3次运输.

网桥

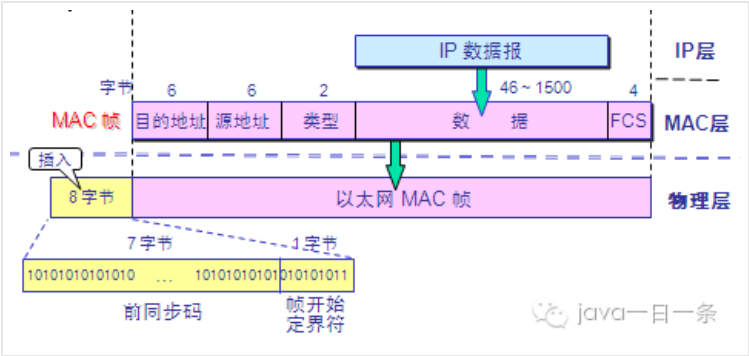
网桥工作在数据链路层, 根据MAC帧的目的地址对收到的帧进行转发和过滤.

以太网交换机

实际上就是一个多接口的网桥, 以太网交换机的每个接口都直接与一个单个主机或另一个集线器相连, 可以很容易实现VLAN(虚拟局域网)

以太网的MAC帧

MAC帧的格式为 :



MAC帧格式

目的地址 : 接收方48位的MAC地址

源地址 : 发送方48位的MAC地址

类型字段 : 标志上一层使用的是什么协议, 0×0800为IP数据报

网络层

如果只有数据链路层没有网络层, 数据就只能在同一条链路上传输, 不能跨链路传输. 有了网络层, 数据便能跨域不同的数据链路传输.

IP地址

IP地址又称为软件地址, 存储在计算机的存储器上, IPv4地址为32位, IPv6地址为128位

IP地址和MAC地址

网络层以上使用IP地址, 数据链路层以下使用MAC地址

IP地址是逻辑地址, MAC地址是物理地址

IP分组中首部的源地址和目的地址在传输中不会改变, MAC帧中首部的源地址和目的地址每到一个路由器会改变一次

IP地址分类

IP地址 = {<网络号>, <主机号>}

A类地址 : 0.0.0.0 ~ 127.0.0.0
B类地址 : 128.0.0.0 ~ 191.255.0.0
C类地址 : 192.0.0.0 ~ 223.255.255.0

划分子网之后的IP地址

IP地址 = {<网络号>, <子网号>, <主机号>}

例如某单位拥有一个B类IP地址, 145.13.0.0, 但凡目的地址为145.13.x.x的数据报都会被送到这个网络上的路由器R. 内部划分子网后变成 : `145.13.3.0` , `145.13.7.0` , `145.13.21.0` . 但是对外仍表现为一个网络, 即 `145.13.0.0` . 这样路由器R收到报文后, 再根据目的地址发到对应的子网上.

子网掩码

一般由一串1和一串0组成, 不管网络有没有划分子网, 将子网掩码和IP地址做按位与运算即可得出网络地址.

所有的网络都必须使用子网掩码, 同时在路由表中必须有子网掩码这一栏. 如果一个网络不划分子网, 那么该网络的子网掩码就是默认的子网掩码.

A类地址的默认子网掩码为255.0.0.0
B类地址的默认子网掩码为255.255.0.0
C类地址的默认子网掩码为255.255.255.0

尽管划分子网增加了灵活性, 但是却减少了能够连接在网络上的主机总数.

构成超网的IP地址

IP地址 = {<网络前缀>, <主机号>}

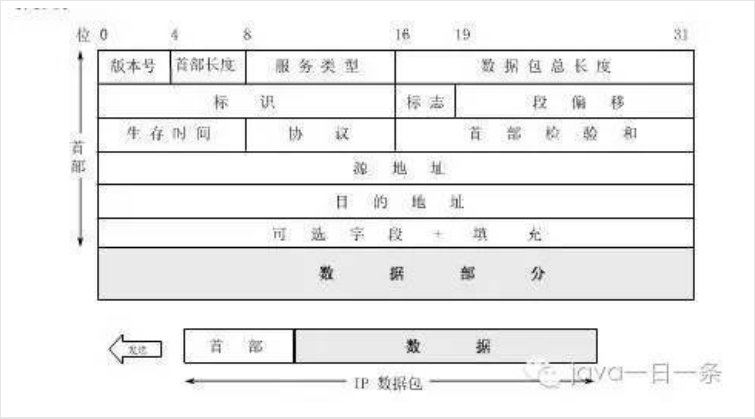
使用网络前缀, 无分类域间路由选择CIDR

例如, 128.14.35.7/20, 意思是前20位为网络前缀, 后12位为主机号. 另外, CIDR把网络前缀相同的连续的IP地址组成一个"CIDR地址块"

地址掩码 : CIDR使用32位的地址掩码, 类似于子网掩码.

IP数据报

在网络层, 数据是以IP数据报(IP分组)的形式传输的



IP数据报的格式

首部前20字节为固定长度, 是所有IP数据报必备的. 后4字节是可选字段, 其长度可变.

IP数据报首部固定的字段分析 :

> 版本号 : IP协议的版本, IPv4或IPv6
>
> 首部长度 : 记录了首部的长度, 最大为1111, 即15个32位字长, 即60字节. 当首部长度不是4字节的整数倍时, 需要使用最后的填充字段加以填充.
>
> 服务类型 : 一般无用
>
> 总长度 : 指首部和数据之和的长度. 最大为2^{16}-1 = 65535字节. 但是由于数据链路层规定每一帧的数据长度都有最大长度MTU, 以太网规定MTU为1500字节, 所以超出范围的数据报就必须进行分片处理
>
> 标识 : 每产生一个IP数据报, 计数器就+1, 并将此值赋值给标识字段. 再以后需要分片的数据报中, 标识相同说明是同一个数据报
>
> 标志 : 占3位, 最低位记为MF(More Fragment). MF = 1说明还有分片; MF = 0说明这已经是最后一个分片. 中间一位记为DF(Don't Fragment), 意思是不能分片. 只有当DF = 0时才允许分片.
>
> 段位移 : 又称片位移, 相对于用户数据字段的起点, 该片从何处开始. 片位移以8个字节为偏移单位. 所以, 每个分片的长度一定是8字节的整数倍.
>
> 生存时间 : TTL(Time To Live). 数据报能在因特网中经过路由器的最大次数为255次, 每经过一个路由器则TTL – 1, 为0时丢弃该报文.
>
> 协议 : 记录该报文所携带的数据是使用何种协议.
>
> 首部检验和 : 只检验数据报的首部, 不检验数据部分. 不为0则丢弃报文.

源地址和目的地址 : 不解释

IP层转发分组的流程

每个路由器内部都维护一个路由表, 路由表包含以下内容(`目的网络地址` , `下一跳地址`).

使用子网时分组转发时, 路由表必须包含以下三项内容: `目的网络地址` , `子网掩码` 和 `下一跳地址` .

特定主机路由 : 对特定的目的地址指明一个路由

默认路由 : 不知道分组该发给哪个路由器时就发给默认路由. 当一个网络只有很少的对外连接时使用默认路由非常合适.

路由器的分组转发算法

从数据报中拿到目的IP地址D, 得出目的网络地址N

若N就是与此路由器直接相连的某个网络地址, 则直接交付(不需要再交给其他路由器转发, 直接找到该目的主机交付), 否则 -> (3)

若路由表中有目的地址为D的特定主机路由, 则把数据报传给该路由器, 否则 -> (4)

若路由表中有到达网络N的路由, 则把数据报传给该路由器, 否则 -> (5)

若路由表中有默认路由, 则交给该路由器, 否则 -> (6)

报告转发分组出错

虚拟专用网VPN

因特网中的所有路由器对该目的地址是专用地址的数据报一律不转发, 下面有3种专用地址(虚拟IP地址)

10.0.0.0 ~ 10.255.255.255

172.16.0.0 ~ 172.31.255.255

192.168.0.0 ~ 192.168.255.255

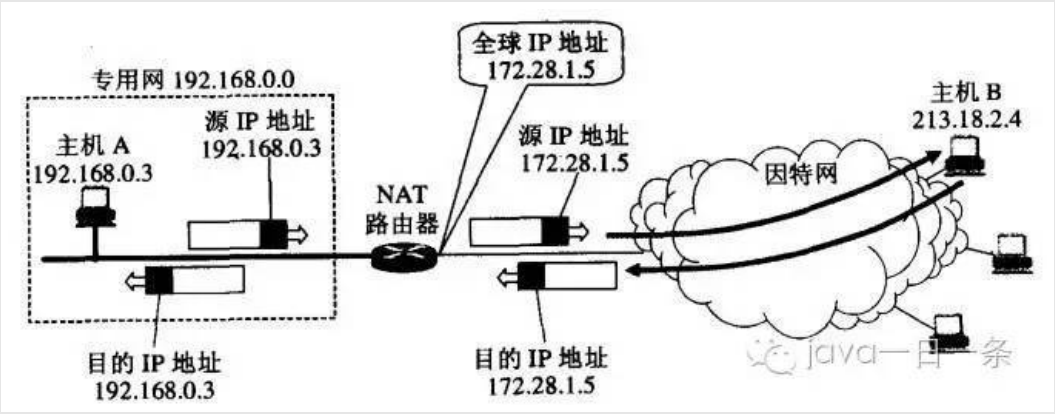
假设现在公司A有一个部门在广州和另一个在上海, 而他们在当地都有自己的专用网. 那么怎么将这两个专用网连接起来呢?

租用电信的通信线路为本机构专用, 但是太贵了

利用公用的因特网当做通信载体, 这就是虚拟专用网VPN

网络地址转换NAT

多个专用网内部的主机公用一个NAT路由器的IP地址, 在主机发送和接收IP数据报时必须先通过NAT路由器进行网络地址转换.



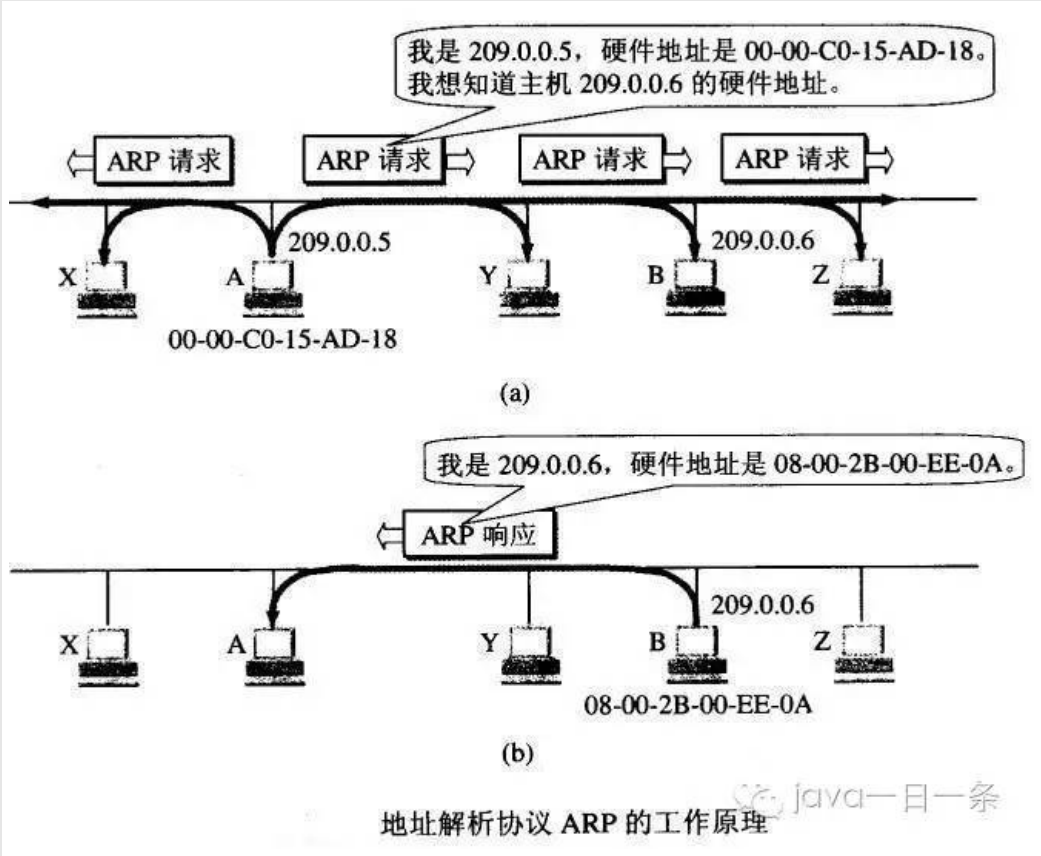
NAT路由器的工作原理

不仅如此, NAT还能使用端口号, 摇身一变成为网络地址和端口转换NAPT

ARP协议

ARP是解决同一个局域网上的主机或路由器的IP地址和MAC地址的映射问题, 即 IP地址 -> ARP -> MAC地址

每一个主机都有一个ARP高速缓存, 里面有本局域网上的各主机和路由器的IP地址到MAC地址的映射表. 以下是ARP的工作原理 :



ARP的工作原理.jpg

那如果是跨网络使用ARP呢?

在本网络上广播

未找到该主机, 则到路由器

路由器帮忙转发(在另一网络上广播)

找到了则完成ARP请求, 未找到则返回(2)

传输层

这一层是重中之重, 因为数据链路层, 网络层这两层的数据传输都是不可靠的, 尽最大能力交付的. 什么意思的? 就是它们不负责提交给你的就是正确的数据. 然而这一层的TCP协议将要提供可靠传输

这一层主要重点是两个协议 : UDP 和 TCP

用户数据报协议UDP

UDP主要特点 :

无连接

尽最大努力交付

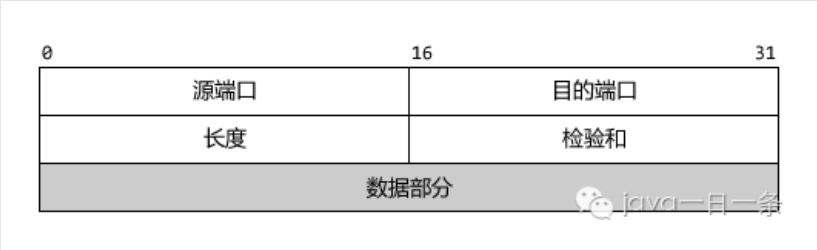
面向报文 : 应用层交下来的报文直接加上UDP头部就往IP层扔, 不合并也不拆分

没有拥塞控制

支持一对一, 一对多, 多对一和多对多的交互通信

首部开销小, 只有8个字节

UDP首部



UDP首部格式

源端口 : 源端口号. 在需要对方回信时选用, 不需要则全0

目的端口 : 目的端口号. 这在终点交付报文时必须要使用到

长度 : UDP数据报的长度, 最小值为8(仅有首部)

检验和 : 与IP数据报只检验首部不同的是, UDP需要把首部和数据部分一起检验

传输控制协议TCP

TCP主要特点 :

面向连接的运输层协议

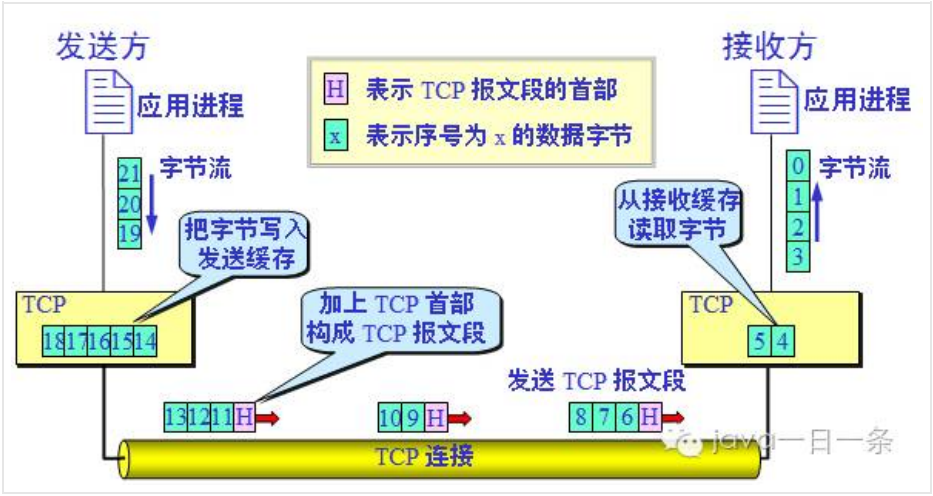
每一条TCP连接只能有2个端点, TCP是点对点的

提供可靠交付

全双工通信

面向字节流

TCP的工作流程



TCP字节流

TCP的连接

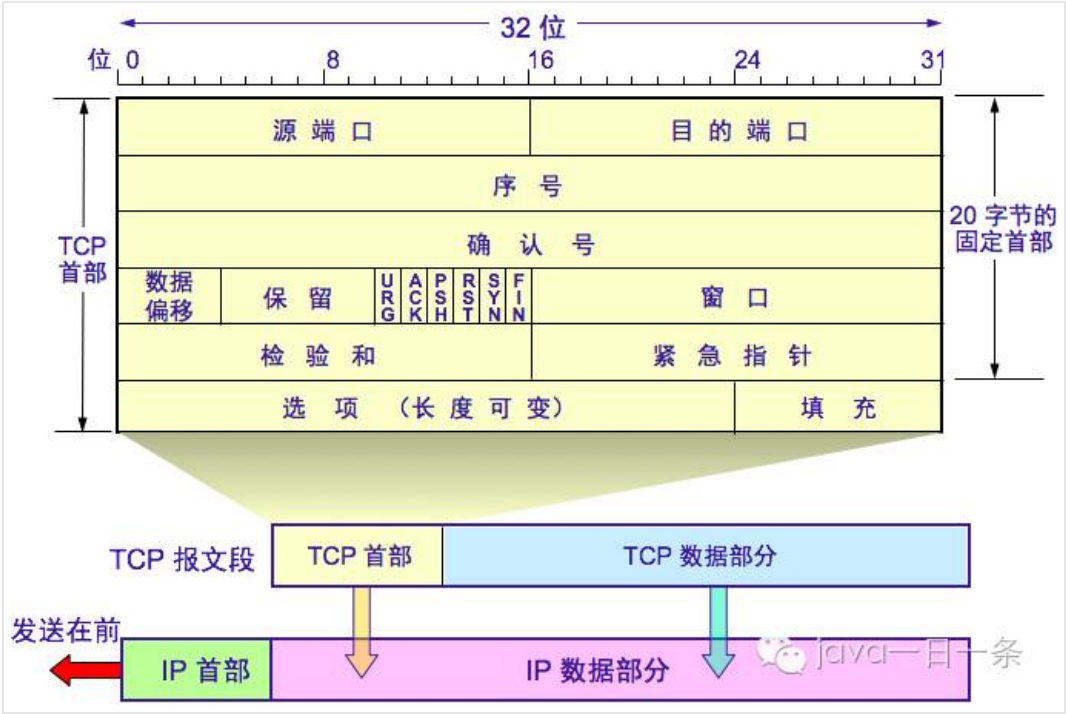
TCP连接的端点叫套接字(socket)

```
socket = (IP地址 : 端口号)
```

每一条TCP连接唯一地被通信两端的两个端点(socket)所确定. 即 :

```
TCP连接 ::= {socket1, socket2} = {(IP1 : port1), (IP2 : port2)}
```

TCP报文段的首部



TCP报文段的首部

源端口和目的端口 : 同UDP端口作用

序号 : 本报文段的数据的第一个字节的序号

确认号 : 期望收到对方下一个报文段的第一个数据字节的序号
`若确认号 = N, 则表明 : 到序号N-1为止的所有数据都已正常收到`

数据偏移 : TCP报文段的首部长度

保留 : 以后用, 目前为0

紧急URG : 若URG = 1时, 说明紧急指针字段有效, 告诉系统这是紧急数据, 应尽快传送. 例如突然要中断传送

确认ACK : ACK = 1时确认号才有效. ACK = 0时确认号无效. TCP规定, 连接建立后所有传送的报文段都必须把ACK置1

推送PSH : 若PSH = 1, 则接收方收到报文段之后不再等到整个缓存满而是直接向上交付

复位RST : 当RST = 1, 说明TCP连接有严重错误, 必须释放连接再重连

同步SYN : 在连接建立时用来同步序号. 当SYN = 1, ACK = 0时表明这是一个连接请求报文段, 对方若同意建立连接, 则在响应的报文段中置SYN = 1, ACK = 1

终止FIN : 当FIN = 1, 表明此报文段的发送方数据已发送完毕, 并要求释放连接

窗口 : 告诉对方 : 从本报文段首部中的确认号算起, 接收方目前允许对方发送的数据量. 这是作为接收方让发送方设置其发送窗口的依据

检验和 : 同UDP, 检验首部和数据部分

紧急指针 : 当URG = 1时有效, 指出紧急数据的末尾在报文段的位置

选项 : 最大可40字节, 没有则为0
`最大报文段长度MSS(Maximum Segment Size) : 每一个TCP报文段中数据字段的最大长度, 若不填写则为默认的536字节.`

窗口

TCP中很重要的一个概念, 那就是窗口(发送窗口和接收窗口)

窗口

由于停止等待协议非常低效, 于是衍生出窗口这一概念. 上图为发送方维持的发送窗口, 位于发送窗口的5个分组都可以连续发送出去而不需要等待对方的确认. 每收到一个确认, 就把发送窗口前移一个分组的位置. 这大大提高了信道利用率!

接收方不必发送每个分组的确认报文, 而是采用累积确认的方式. 也就是说, 对按序到达的最后一个分组发送确认报文.

超时重传

如果发送方等待一段时间后, 还是没收到 ACK 确认报文, 就会启动超时重传. 这个等待的时间为重传超时时间(RTO, Retransmission TimeOut).

然而, RTO 的值不是固定的, 这个时间总是略大于连接往返时间(RTT，Round Trip Time). 假设报文发送过去需要5秒, 对方收到后发送确认报文回来也需要5秒, 那么RTT就为10秒, 那这RTO就要比10秒要略大一些. 那么超过RTO之后还没有收到确认报文就认为报文丢失了, 就要重传.

流量控制

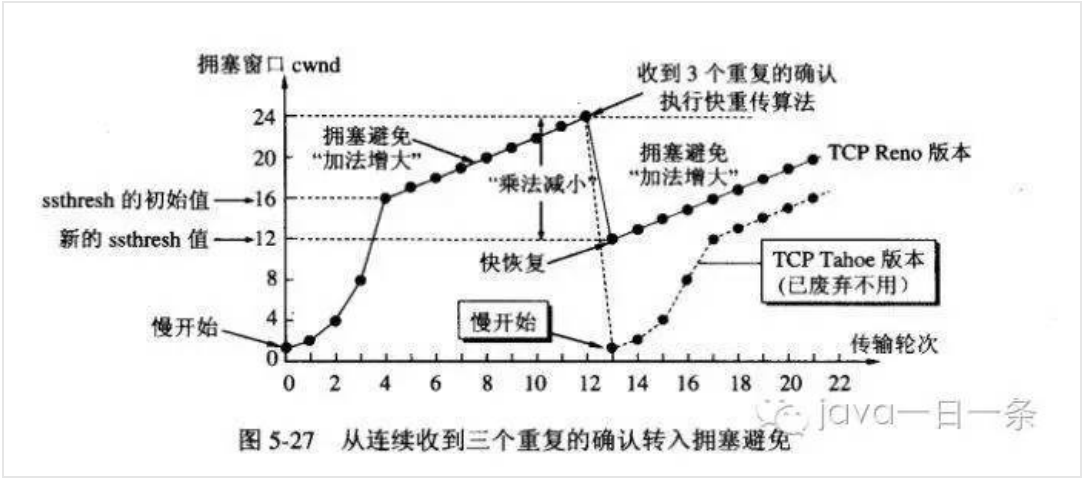
利用滑动窗口和报文段的发送时机来进行流量控制.

拥塞控制

发送方维持一个拥塞窗口cwnd, 发送窗口 = 拥塞窗口.

`慢开始` : cwnd = 1, 然后每经过一个传输轮次就翻倍
`拥塞避免` : 让cwnd缓慢增大, 每经过一个传输轮次就+1
`慢开始门限ssthresh` :

```
当cwnd < ssthresh, 使用慢开始算法
当cwnd > ssthresh, 使用拥塞避免算法
当cwnd = ssthresh, 随意
```



拥塞控制

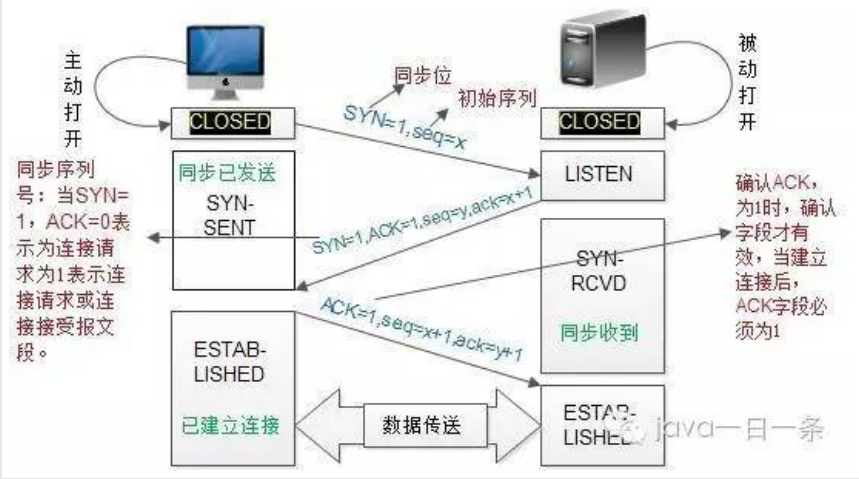
只要判断网络出现拥塞, 把ssthresh设为当前发送拥塞窗口的一半(不能小于2), 并把cwnd设为1, 重新执行慢开始算法.

除了慢开始和拥塞避免算法外, 还有一组快重传和快恢复算法 :

`快重传` : 接收方及时发送确认, 而发送方只要一连收到三个重复确认, 马上重传
`快恢复` : 当发送方一连收到三个重复确认时, ssthresh减半, cwnd设为ssthresh.

TCP三次握手

TCP三次握手建立连接和四次挥手断开连接是面试爱问的知识点.



TCP三次握手

> Q : 为什么要三次握手, 两次不可以吗?
>
> A : 试想一下, A第一次发送请求连接, 但是在网络某节点滞留了, A超时重传, 然后这一次一切正常, A跟B就愉快地进行数据传输了. 等到连接释放了以后, 那个迷失了的连接请求突然到了B那, 如果是两次握手的话, B发送确认, 它们就算是建立起了连接了. 事实上A并不会理会这个确认, 因为我压根没有要传数据啊. 但是B却傻傻地以为有数据要来, 苦苦等待. 结果就是造成资源的浪费.

更加接地气的解释就是 : A打电话给B

> 第一次握手 ： 你好， 我是A， 你能听到我说话吗
> 第二次握手 ： 听到了， 我是B， 你能听到我说话吗
> 第三次握手 ： 听到了， 我们可以开始聊天了
>
> 三次握手其实就是为了检测双方的发送和接收能力是否正常， 你说呢？

TCP四次挥手

TCP四次挥手

Q : 为什么要四次挥手, 而不是两次, 三次?

A :
首先, 由于TCP的全双工通信, 双方都能作为数据发送方. A想要关闭连接, 必须要等数据都发送完毕, 才发送FIN给B. (此时A处于半关闭状态)
然后, B发送确认ACK, 并且B此时如果要发送数据, 就发送(例如做一些释放前的处理)
再者, B发送完数据之后, 发送FIN给A. (此时B处于半关闭状态)
然后, A发送ACK, 进入TIME-WAIT状态
最后, 经过2MSL时间后没有收到B传来的报文, 则确定B收到了ACK了. (此时A, B才算是处于完全关闭状态)

PS : 仔细分析以上步骤就知道为什么不能少于四次挥手了.

Q : 为什么要等待2MSL(Maximum Segment Lifetime)时间, 才从TIME_WAIT到CLOSED?

A : 在Client发送出最后的ACK回复，但该ACK可能丢失。Server如果没有收到ACK，将不断重复发送FIN片段。所以Client不能立即关闭，它必须确认Server接收到了该ACK。Client会在发送出ACK之后进入到TIME_WAIT状态。Client会设置一个计时器，等待2MSL的时间。如果在该时间内再次收到FIN，那么Client会重发ACK并再次等待2MSL。MSL指一个片段在网络中最大的存活时间，2MSL就是一个发送和一个回复所需的最大时间。如果直到2MSL，Client都没有再次收到FIN，那么Client推断ACK已经被成功接收，则结束TCP连接。

更加接地气的解释 :

```
第一次挥手 ： A告诉B，我没数据发了，准备关闭连接了，你要发送数据吗
第二次挥手 ： B发送最后的数据
第三次挥手 ： B告诉A，我也要关闭连接了
第四次挥手 ： A告诉B你可以关闭了，我这边也关闭了
```

应用层

应用层协议最著名的就是HTTP, FTP了, 还有一个重要的DNS

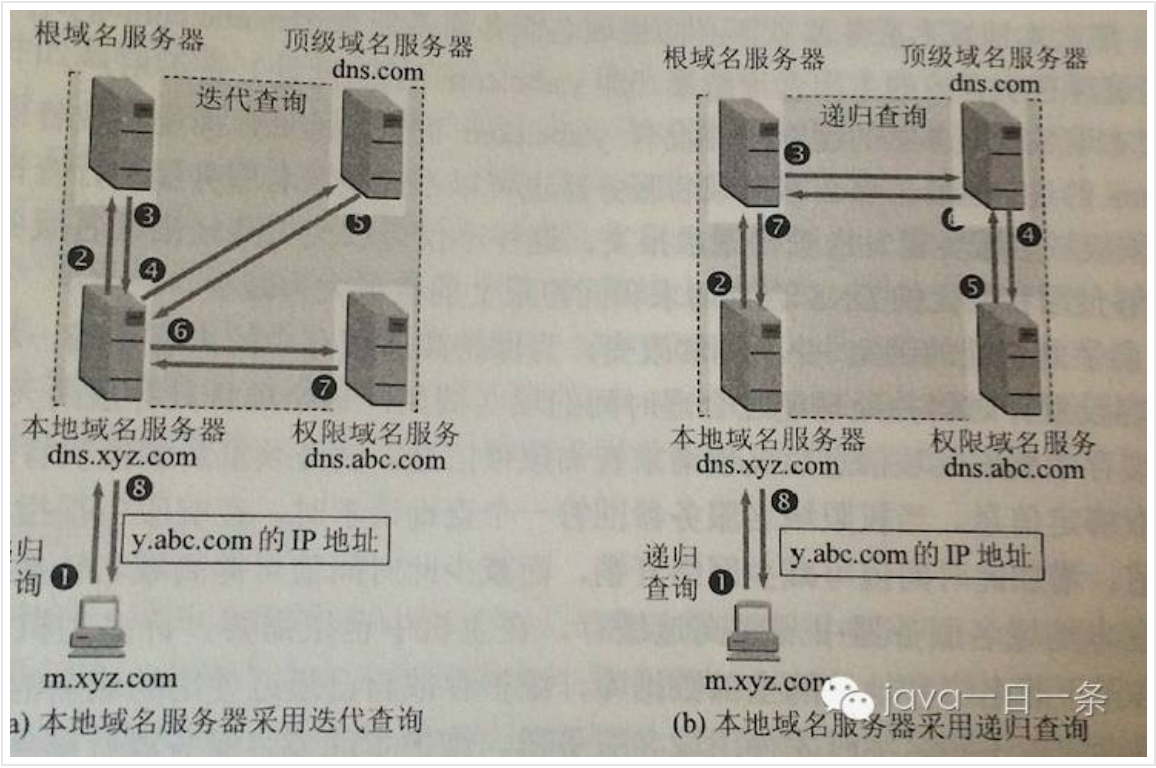
域名系统(DNS, Domain Name System)

DNS 能将域名(例如, www.jianshu.com)解析成IP地址.

域名服务器分类

> 根域名服务器 : 最高层次的域名服务器
>
> 顶级域名服务器 : 如其名
>
> 权限域名服务器 : 负责一个区的应服务器
>
> 本地域名服务器 : 主机发送DNS查询请求就是发给它

DNS查询



DNS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

本地域名服务器向根域名服务器的查询通常是采用迭代查询

```
递归查询 : B问A广州怎么去, A不知道, A就问C, C不知道就问D...直到知道了再一层一层转告直到A告诉B.
 迭代查询 : B问A广州怎么去, A不知道, A就告诉你可以去问C, 然后B就去问C, C不知道, C就告诉你可以去问D, 然后B就去问D...直到B知道为止
```

DNS查询例子：域名为x.tom.com的主机想知道y.jerry.com的IP地址

主机x.tom.com先向本地域名服务器dns.tom.com进行递归查询

本地域名服务器采用迭代查询. 它先问一个根域名服务器

根域名服务器告诉它, 你去问顶级域名服务器dns.com

本地域名服务器问顶级域名服务器dns.com

顶级域名服务器告诉它, 你去问权限域名服务器dns.jerry.com

本地域名服务器问权限域名服务器dns.jerry.com

权限域名服务器dns.jerry.com告诉它所查询的主机的IP地址

本地域名服务器把查询结果告诉主机x.tom.com

```
PS : 该查询使用UDP, 并且为了提高DNS查询效率, 每个域名服务器都使用高速缓存.
```

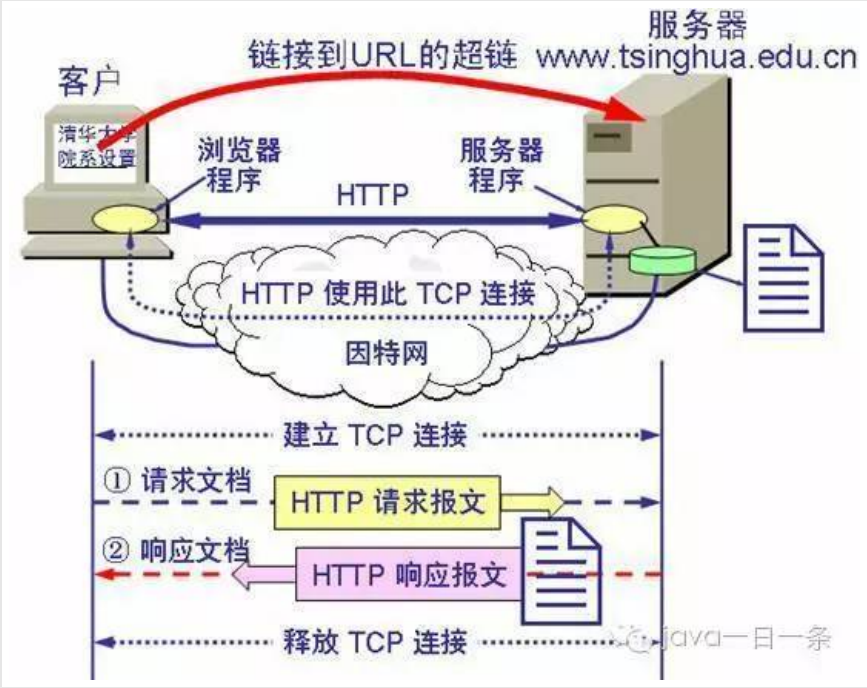
URL

URL的格式：`<协议>://<主机>:<端口>/<路径>`，端口和路径有时可省略.

使用HTTP协议的URL：`http://<主机>:<端口>/<路径>`，HTTP默认端口号是80

HTTP协议

HTTP是面向事务的, 即它传输的数据是一个整体, 要么全部收到, 要么全部收不到.



万维网的工作过程

每一次HTTP请求就需要建立一次TCP连接和释放TCP连接.

HTTP是无连接, 无状态的. 每一次请求都是作为一次新请求.

HTTP/1.0 缺点 : 无连接, 每一次请求都要重新建立TCP连接, 所以每一次HTTP请求都要花费2倍RTT时间(一次TCP请求, 一次HTTP请求)

HTTP/1.1 : 使用持续连接, 即保持TCP连接一段时间.

```
HTTP/1.1持续工作的两种工作方式 : 非流水线方式和流水线方式
    非流水线方式 : 收到一个请求的响应再发下一个请求, 效率低, 浪费资源
    流水线方式 : 能够同时发送多个请求, 效率高
```

HTTP的GET和POST

GET 请求通常用于查询、获取数据，而 POST 请求则用于发送数据
GET 请求的参数在URL中, 因此绝不能用GET请求传输敏感数据, 而POST 请求的参数在请求头中, 安全性略高于GET请求

```
ps : POST请求的数据也是以明文的形式存放在请求头中, 因此也不安全
```

Cookie

万维网使用Cookie来跟踪用户, 表示HTTP服务器和用户之间传递的状态信息.

Cookie工作原理 :

```
1. 用户浏览某网站, 该网站的服务器为用户产生一个唯一的识别码, 并以此为索引在服务器后端数据库中产生一个项目2. 返回给用户的HTTP响应报文中添加一条 "Set-cookie", 值为该识别码, 如1233. 用户的浏览器将该cookie保存起来, 在用于继续浏览该网站时发送的每一个HTTP请求都会有一行 Cookie: 123

于是, 这个网站就知道Cookie为123的这个用户做了什么, 为这个用户维护一个独立的列表(如购物车)
```

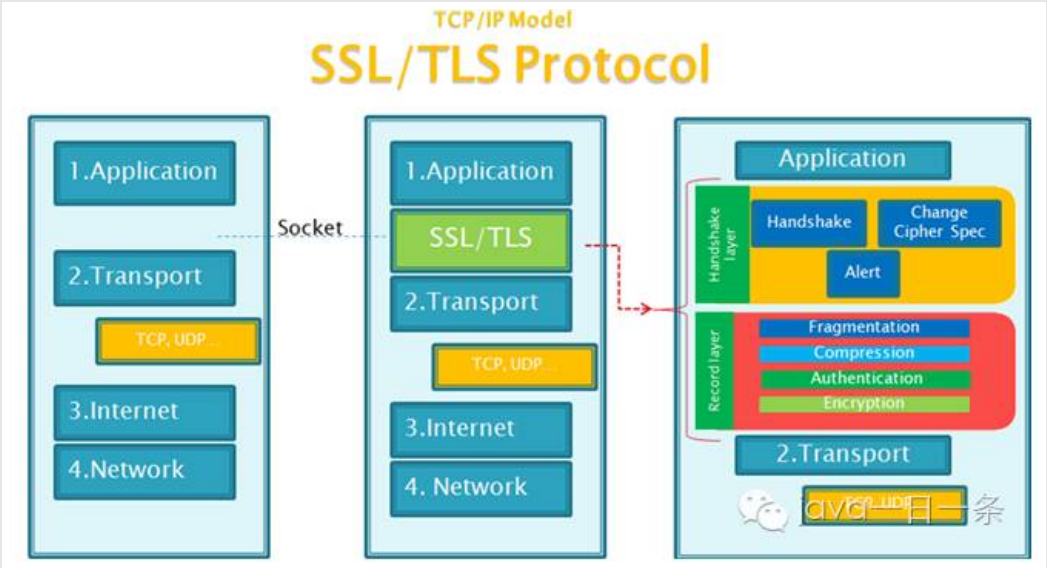
当然, Cookie是把双刃剑, 方便的同时也带有危险性, 例如隐私泄露等, 用户可以自行决定是否使用Cookie

Session

Cookie是保存在客户端上的, 而Session是保存在服务器中. 当服务器收到用户发出的Cookie时, 会根据Cookie中的SessionID来查找对应的Session, 如没有则会生成一个新的SessionID返回给用户

总而言之, Cookie和Session就是同一样东西存放地方不同而已.

HTTPS



HTTPS协议

HTTPS协议在HTTP协议的基础上, 在HTTP和TCP中间加入了一层SSL/TLS加密层, 解决了HTTP不安全的问题: 冒充, 篡改, 窃听三大风险.

对HTTPS是如何做到安全, 加密等有兴趣的可以参考以下文章

SSL/TLS协议运行机制的概述 http://www.codeceo.com/article/ssl-tls-run.html

阅读原文 投诉