# 【Java】留下没有基础眼泪的面试题

# 前言

只有光头才能变强

# 一、如何减少线程上下文切换

使用多线程时，**不是多线程能提升程序的执行速度**，使用多线程是为了**更好地利用 CPU 资源！**

程序在执行时，多线程是 CPU 通过给每个线程**分配 CPU 时间片来实现的**，时间片是 CPU 分配给每个线程执行的时间，因时间片非常短，所以 **CPU 通过不停地切换线程执行。**

线程**不是越多就越好的**，因为线程上下文切换是有**性能损耗**的，在使用多线程的同时需要考虑如何减少上下文切换

一般来说有以下几条经验：

- **无锁并发编程。**多线程竞争时，会引起上下文切换，所以多线程处理数据时，可以用一些办法来避免使用锁，如将数据的 ID 按照 Hash 取模分段，不同的线程处理不同段的数据
- **CAS 算法。**Java 的 Atomic 包使用 CAS 算法来更新数据，而不需要加锁。
- **控制线程数量。**避免创建不需要的线程，比如任务很少，但是创建了很多线程来处理，这样会造成大量线程都处于等待状态
- **协程。**在单线程里实现多任务的调度，并在单线程里维持多个任务间的切换

协程可以看成是用户态自管理的“线程”。不会参与 CPU 时间调度，没有均衡分配到时间。非抢占式的还可以考虑我们的应用是 **IO 密集型的还是 CPU 密集型的。**

- 如果是 IO 密集型的话，线程可以多一些。
- 如果是 CPU 密集型的话，线程不宜太多。

# 二、计算机网络

## 2.1 MAC 地址已经是唯一了，为什么需要 IP 地址？

简单总结一下为什么有了 MAC(IP)还需要 IP(MAC)：

- MAC 是链路层，IP 是网络层，每一层干每一层的事儿，之所以在网络上分链路层、

网络层...，就是将问题简单化。

- 历史的兼容问题。

已经有 IP 地址了，为什么需要 MAC 地址？？

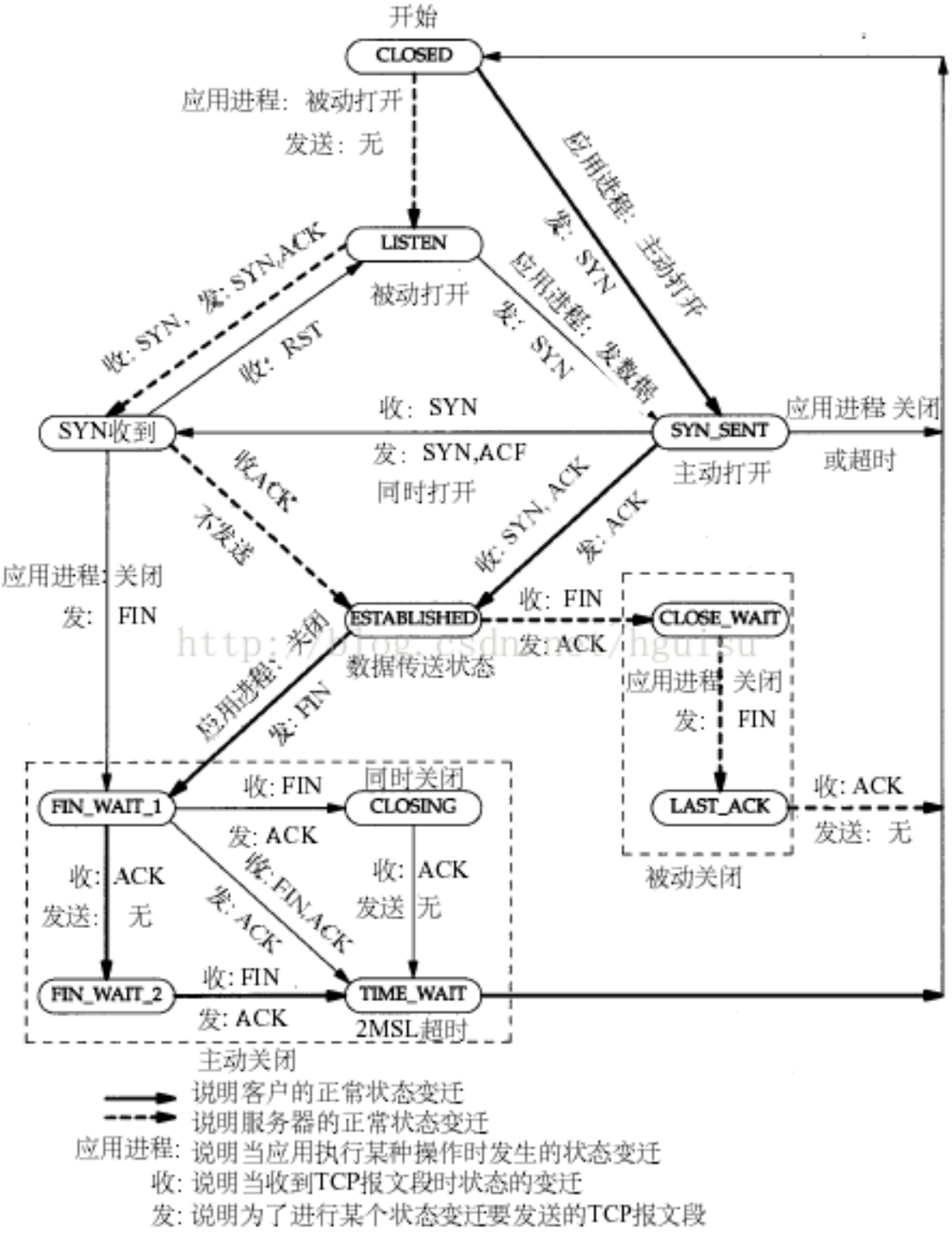
- 现阶段理由：DHCP 基于 MAC 地址分配 IP。

MAC 地址已经是唯一了，为什么需要 IP 地址？

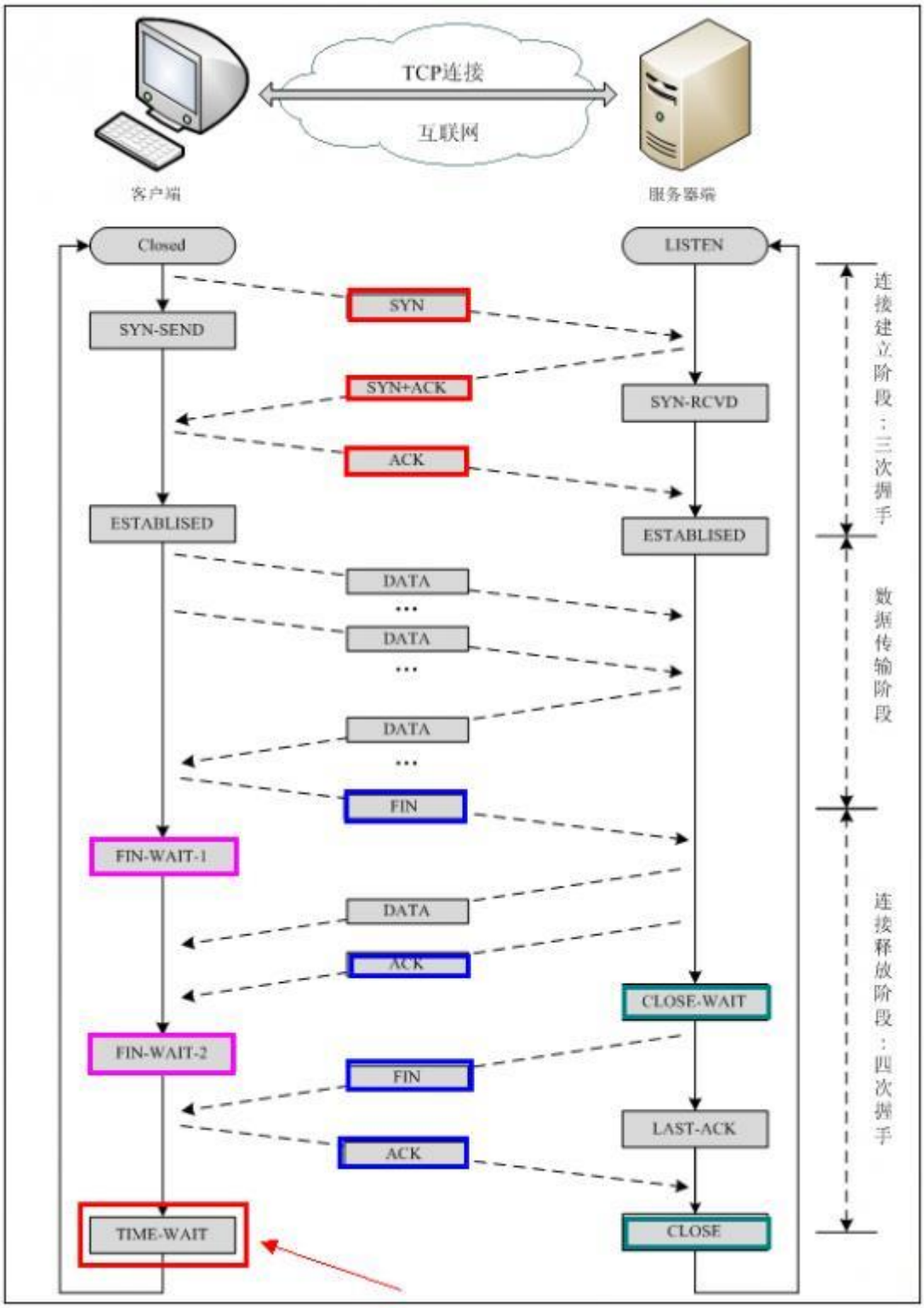
- MAC 无网段概念，非类聚，不好管理。

## 2.2 TCP 状态

TCP 总共有 11 个状态，状态之间的转换是这样的：



流程图：




下面我简单总结一下每个状态：

- **CLOSED**：初始状态，表示 TCP 连接是“关闭着的”或“未打开的”。
- **LISTEN**：表示服务器端的某个 SOCKET 处于监听状态，可以接受客户端的连接。
- **SYN-SENT**：表示客户端已发送 SYN 报文。当客户端 SOCKET 执行 connect()进行

连接时，它首先发送 SYN 报文，然后随即进入到 SYN_SENT 状态。

- **SYN_RCVD**：表示服务器接收到了来自客户端请求连接的 SYN 报文。当 TCP 连接处于此状态时，再收到客户端的 ACK 报文，它就会进入到 ESTABLISHED 状态。
- **ESTABLISHED**：表示 TCP 连接已经成功建立。
- **FIN-WAIT-1**：第一次主动请求关闭连接,等待对方的 ACK 响应。
- **CLOSE_WAIT**：对方发了一个 FIN 报文给自己，回应一个 ACK 报文给对方。此时进入 CLOSE_WAIT 状态。
  - 接下来呢，你需要检查自己是否还有数据要发送给对方，如果没有的话，那你也就可以 close()这个 SOCKET 并发送 FIN 报文给对方，即关闭自己到对方这个方向的连接
- **FIN-WAIT-2**：主动关闭端接到 ACK 后，就进入了 FIN-WAIT-2。在这个状态下，应用程序还有接受数据的能力，但是已经无法发送数据。
- **LAST_ACK**：当被动关闭的一方在发送 FIN 报文后，等待对方的 ACK 报文的时候，就处于 LAST_ACK 状态
- **CLOSED**：当收到对方的 ACK 报文后，也就可以进入到 CLOSED 状态了。
- **TIME_WAIT**：表示收到了对方的 FIN 报文，并发送出了 ACK 报文。TIME_WAIT 状态下的 TCP 连接会等待 2*MSL
- **CLOSING**：罕见的状态。表示双方都正在关闭 SOCKET 连接

TIME_WAIT 状态一般用来处理以下两个问题：

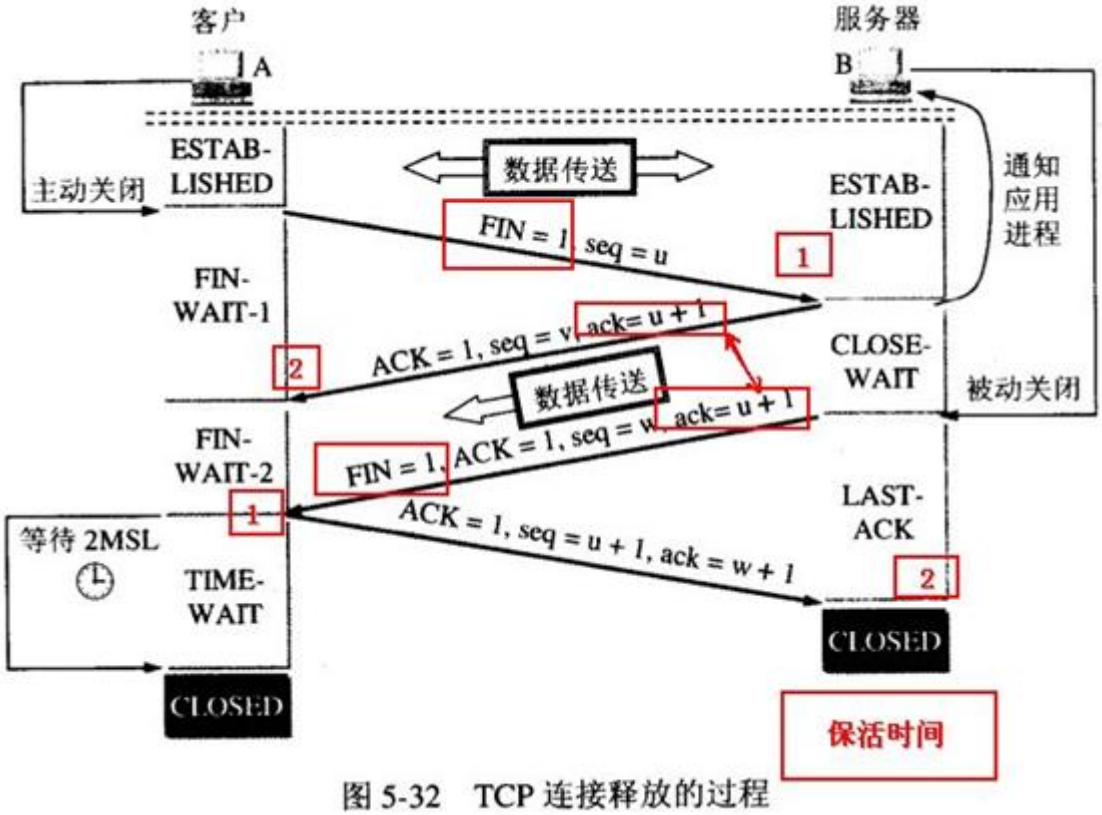



图 5-32 TCP 连接释放的过程

- 关闭 TCP 连接时，确保最后一个 ACK 正常运输(或者可以认为是 等待以便重传 ACK)
- 网络上可能会有残余的数据包，为了能够正常处理这些残余的数据包。使用 TIME-

WAIT 状态可以确保在创建新连接时，先前网络中残余的数据都丢失了。

TIME_WAIT 过多怎么解决？

如果在**高并发，多短链**接情景下，TIME_WAIT 就会过多。可以通过调整内核参数解决：vi /etc/sysctl.conf 加入以下内容设置：

- reuse 是表示是否允许重新应用处于 TIME-WAIT 状态的 socket 用于新的 TCP 连接；
- recyse 是加速 TIME-WAIT sockets 回收

我们可以知道 TIME_WAIT 状态是**主动关闭连接的一方出现的**，我们不要轻易去使用上边两个参数。先看看是不是可以**重用 TCP 连接**来尽量避免这个问题(比如我们 HTTP 的 KeepAlive)~

## 2.3 TCP 滑动窗口

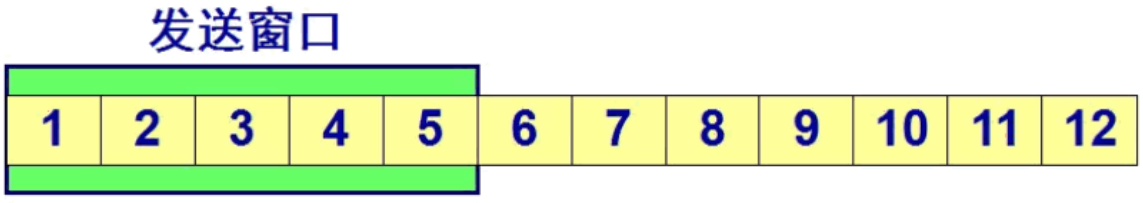
TCP 是一个可靠的传输协议，它要**保证所有的数据包都可以到达**，这需要重传机制来支撑。重传机制有以下几种：

- 超时重传
- 快速重传
- SACK 方法

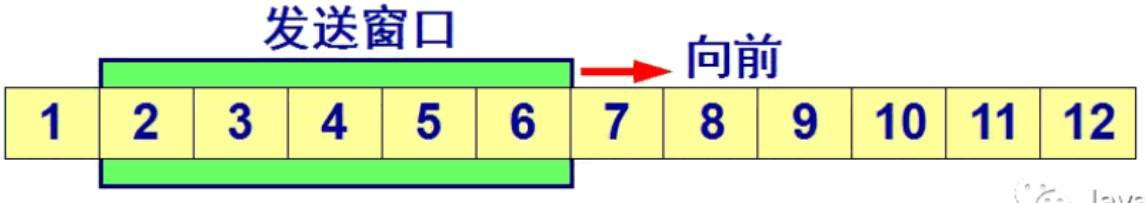
滑动窗口可以说是 TCP 非常重要的一个知识点。TCP 的滑动窗口主要有两个作用：

- 提供 TCP 的可靠性
- 提供 TCP 的流控特性

简略滑动窗口示意图：




(a) 发送方维持发送窗口（发送窗口是 5）

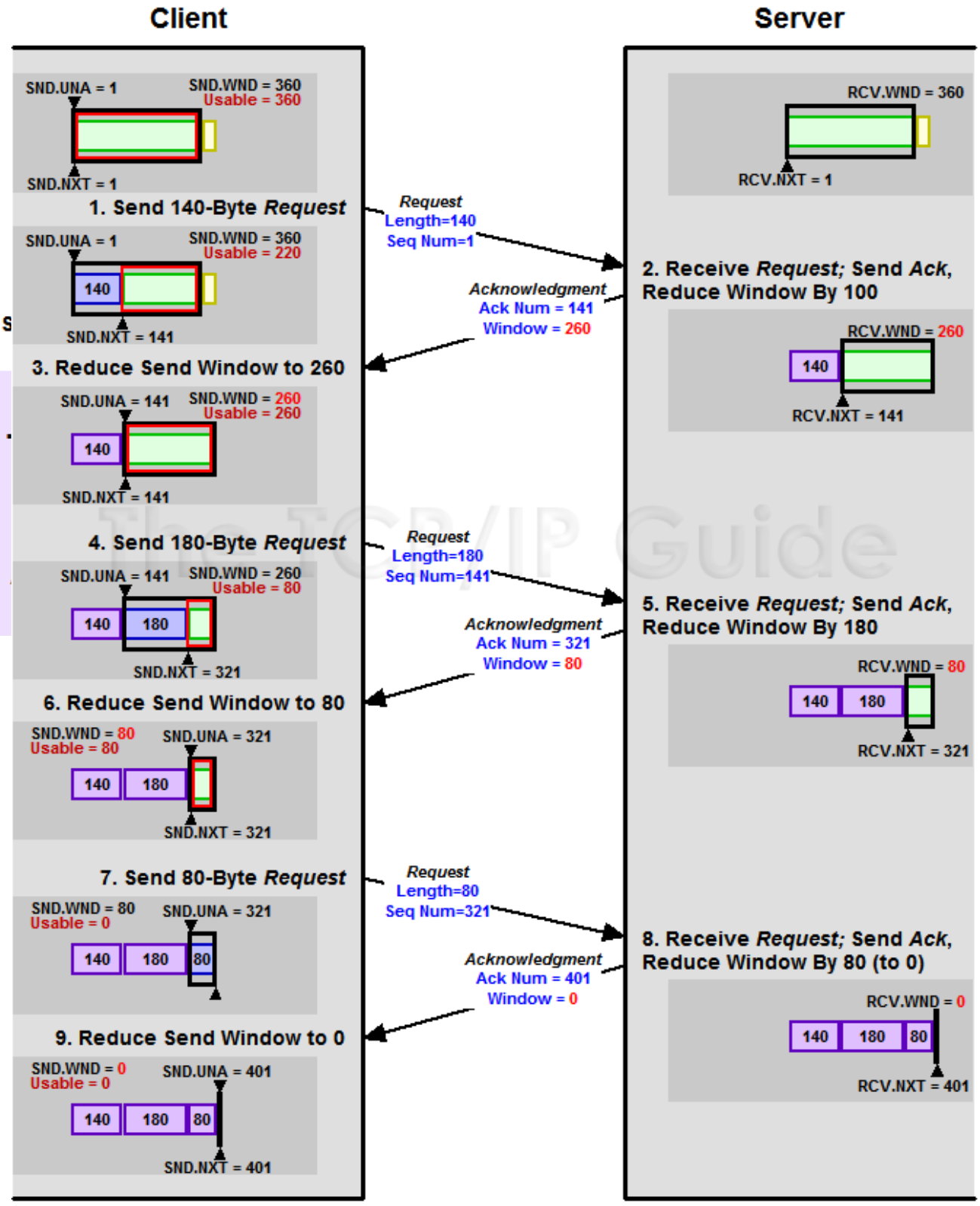



(b) 收到一个确认后发送窗口向前滑动

详细滑动窗口示意图：

- #1 已收到 ack 确认的数据。
- #2 发还没收到 ack 的。
- #3 在窗口中还没有发出的（接收方还有空间）。
- #4 窗口以外的数据（接收方没空间）

接受端控制发送端的图示：



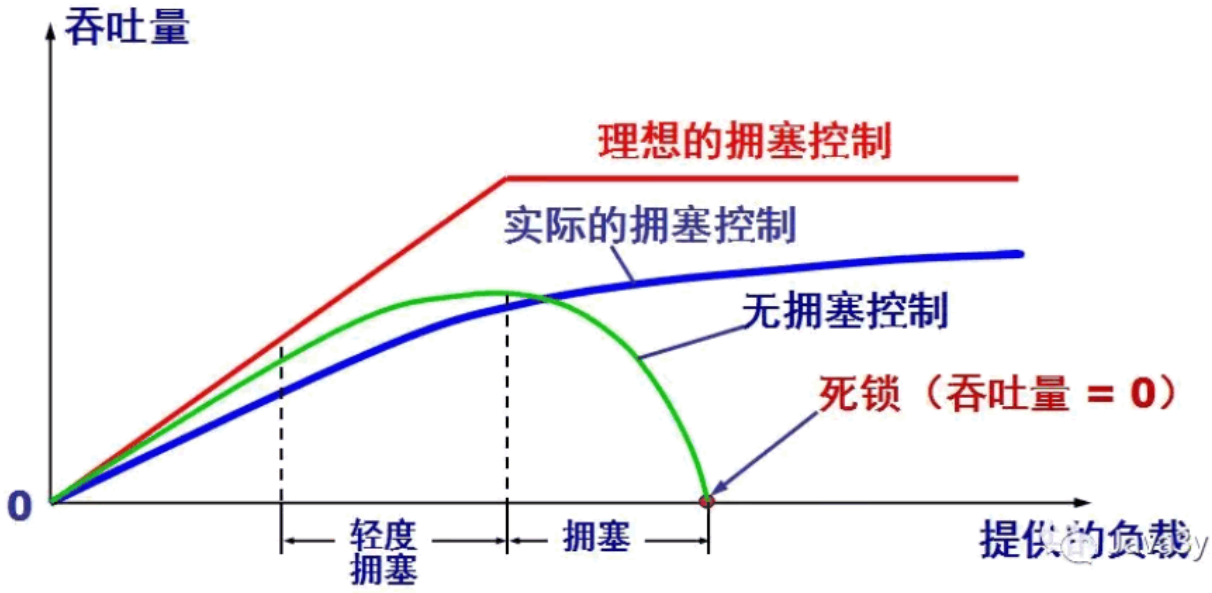
## 2.4 拥塞控制

TCP 不是一个自私的协议，当拥塞发生的时候，要做自我牺牲。就像交通阻塞一样，每个车都应该把路让出来，而不要再去抢路了。

拥塞控制主要是四个算法：

- 慢启动，
- 拥塞避免，
- 拥塞发生，
- 快速恢复

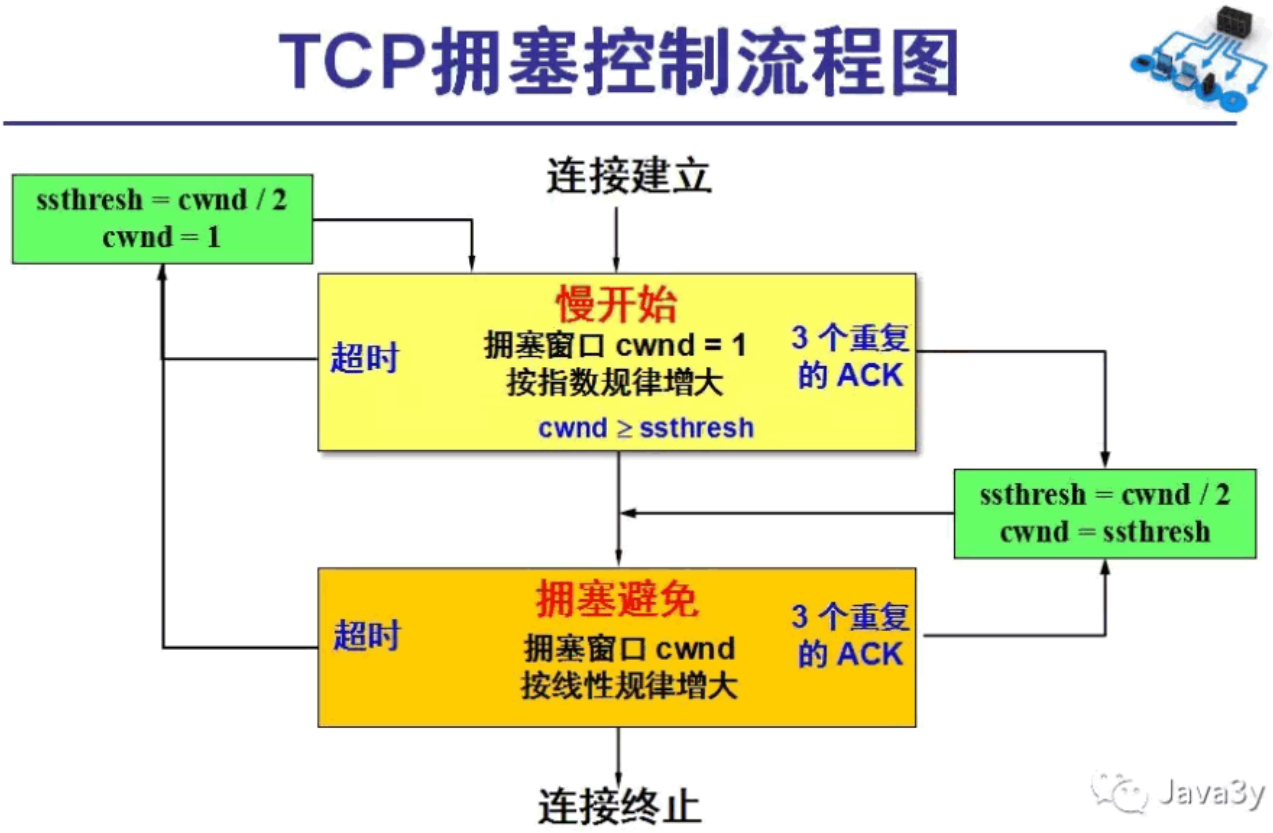
拥塞控制的作用：




拥塞的判断：

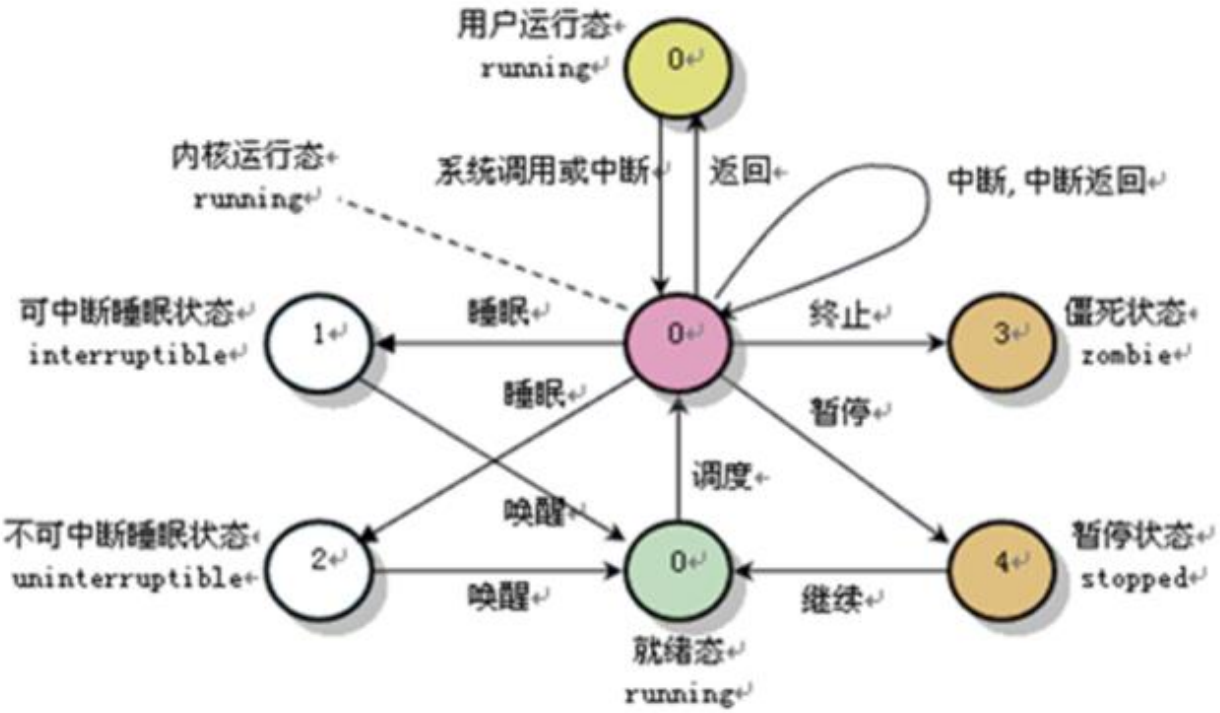
- 重传定时器超时

- 收到三个相同（重复）的 ACK



# 三、操作系统

## 3.1 僵尸进程和孤儿进程是什么(区别)




- **僵尸进程：**

  父进程创建出子进程，子进程退出了，父进程没有调用 wait 或 waitId 获取子进程的信息(状态)，**子进程的描述符仍在系统中。**

- **孤儿进程：**

  父进程退出，子进程仍在运行中。这些子进程就叫做孤儿进程，孤儿进程将被 **init 进程**(进程号为 1)所**收养**，并由 init 进程对它们完成状态收集工作。

- **僵尸进程危害：**
  - 系统进程表是一项有限资源，如果系统进程表被僵尸进程耗尽的话，系统就可能无法创建新的进程。
  - 一个父进程创建了很多子进程，就是不回收，会造成内存资源的浪费。
- **解决僵尸进程的手段：**
  - 杀掉父进程，余下的僵尸进程会成为孤儿进程，最后被 init 进程管理
  - 子进程退出时向父进程发送 SIGCHILD 信号，父进程处理 SIGCHILD 信号。在信号处理函数中调用 wait 进行处理僵尸进程
  - fork 两次：原理是将子进程成为孤儿进程，从而其的父进程变为 init 进程，通过 init 进程可以处理僵尸进程

## 3.2 操作系统进程间通信的方式有哪些？

首先要知道的是：**进程和线程的关注点是不一样的：**

- 进程间资源是独立的，关注的是通讯问题。
- 线程间资源是共享的，关注的是安全问题。

**操作系统进程间通信的方式有哪些？**

- 管道（pipe）：管道是一种半双工的通信方式，数据只能单向流动，而且只能在具有亲缘关系的进程间使用。进程的亲缘关系通常是指父子进程关系。
- 有名管道（named pipe）：有名管道也是半双工的通信方式，但是它允许无亲缘关系进程之间的通信。
- 消息队列（message queue）：消息队列是消息的链表，存放在内核中并由消息队列表示符标示。消息队列克服了信号传递信息少，管道只能承载无格式字节流以及缓冲区大小受限制等缺点。
- 共享内存（shared memory）：共享内存就是映射一段内被其它进程所访问的内存，共享内存由一个进程创建，但是多个进程都可以访问。共享内存是最快的 IPC，它是针对其它进程通信方式运行效率低的而专门设计的。它往往与其它通信机制。如信号量，配合使用，来实现进程间的同步和通信。
- 套接字（socket）：套接字也是进程间的通信机制，与其它通信机制不同的是，它可以用于不同机器间的进程通信。
- 信号（signal）：信号是一种比较复杂的通信方式，用于通知接受进程进程某个时间已经发生。
- 信号量（semaphore）：信号量是一个计数器，可以用来控制多个进程对共享资源的访问。
  - 它常作为一种锁的机制，防止某进程正在访问共享资源时，其它进程也访问该资源。因此它主要作为不同进程或者同一进程之间不同线程之间同步的手段。

## 3.3 操作系统线程间通信的方式有哪些？

**操作系统线程间通信的方式有哪些？**(可以直接理解成：**线程之间同步的方式有哪些**)

- 锁机制：包括互斥锁、条件变量、读写锁
- 信号量机制(Semaphore)：包括无名线程信号量和命名线程信号量
- 信号机制(Signal)：类似进程间的信号处理

线程间的通信目的**主要是用于线程同步。**

## 3.4 操作系统进程调度算法有哪些？

- 先来先服务算法(FCFS)

谁先来，就谁先执行

- 短进程/作业优先算法(SJF)

  谁用的时间少、就先执行谁

- 最高响应比优先算法(HRN)

  对 FCFS 方式和 SJF 方式的一种综合平衡

- 最高优先数算法

  系统把处理机分配给就绪队列中优先数最高的进程

- 基于时间片的轮转调度算法

  每个进程所享受的 CPU 处理时间都是一致的

- 最短剩余时间优先算法

  短作业优先算法的升级版，只不过它是抢占式的

- 多级反馈排队算法

  设置多个就绪队列，分别赋予不同的优先级，如逐级降低，队列 1 的优先级最高

# 四、拓展阅读

## 4.1 ConcurrentHashMap 中的扩容是否需要对整个表上锁？

总结(摘抄)要点：

- 通过给每个线程分配桶区间(默认一个线程分配的桶是 16 个)，避免线程间的争用。
- 通过为每个桶节点加锁，避免 putVal 方法导致数据不一致。
- 同时，在扩容的时候，也会将链表拆成两份，这点和 HashMap 的 resize 方法类似。

## 4.2 什么是一致性 Hash 算法（原理）？

总结(摘抄)要点：

一致性 Hash 算法将整个哈希值空间组织成一个虚拟的圆环，好处就是提高容错性和可扩展性。对于节点的增减都只需重定位环空间中的一小部分数据。

## 4.3 MySQL date、datetime 和 timestamp 类型的区别

总结(摘抄)要点：

- date 精确到天，datetime 和 timestamp 精确到秒
- datetime 和 timestamp 的区别：
  - timestamp 会跟随设置的时区变化而变化，而 datetime 保存的是绝对值不会变

化

- timestamp 储存占用 4 个字节，datetime 储存占用 8 个字节
- 可表示的时间范围不同 ,timestamp 只能到表示到 2038 年 ,datetime 可到 9999 年

## 4.4 判断一个链表是否有环/相交

判断一个链表是否有环(实际上就是看看有无遍历到重复的节点)，解决方式(3 种)：

1）for 遍历两次

2）使用 hashSet 做缓存，记录已遍历过的节点

3）使用两个指针，一前一后遍历，总会出现前指针==后指针的情况

判断**两个无环链表是否相交**，解决方式(2 种)：

- 将第一个链表尾部的 next 指针指向第二个链表，两个链表组成一个链表。
  - 判断这一个链表是否有环，有环则相交，无环则不相交
- 直接判断两个链表的尾节点是否相等，如果相等则相交，否则不相交

判断**两个有环链表**是否相交(注:当一个链表中有环，一个链表中没有环时 ,两个链表必不相交)：

找到第一个链表的环点，然后将环断开（当然不要忘记了保存它的下一个节点），然后再来遍历第二个链表，如果发现第二个链表从有环变成了无环，那么他们就是相交的嘛，否则就是不相交的了。

## 4.5 keepAlive 含义

- HTTP 协议的 Keep-Alive 意图在于连接复用，同一个连接上串行方式传递请求-响应数据
- TCP 的 KeepAlive 机制意图在于保活、心跳，检测连接错误