# 集合

## HashMap

**为什么Map桶中个数超过8才转为红黑树**

为什么要转换:因为Map中桶的元素初始化是链表保存的，其查找性能是O(n)，而树结构能将查找性能提升到O(log(n))。当链表长度很小的时候，即使遍历，速度也非常快，但是当链表长度不断变长，肯定会对查询性能有一定的影响，所以才需要转成树。

链表长度达到8就转成红黑树，当长度降到6就转成普通bin。为什么长度为8的时候转成红黑树？说白了就是trade-off，空间和时间的权衡：当hashCode离散性很好的时候，树型bin用到的概率非常小，因为数据均匀分布在每个bin中，几乎不会有bin中链表长度会达到阈值。但是在随机hashCode下，离散性可能会变差，然而JDK又不能阻止用户实现这种不好的hash算法，因此就可能导致不均匀的数据分布。不过理想情况下随机hashCode算法下所有bin中节点的分布频率会遵循泊松分布，我们可以看到，一个bin中链表长度达到8个元素的概率为0.00000006，几乎是不可能事件。所以，之所以选择8，不是拍拍屁股决定的，而是根据概率统计决定的。

# 线程池

单个线程: Executors.newSingleThreadExecutor();

缓存线程: Executors.newCachedThreadPool();

固定线程Executors.newFixedThreadPool(2);

定时线程: Executors.newScheduledThreadPool(3);

## ThreadPoolExecutor

ThreadPoolExecutor 继承 AbstractExecutorService；AbstractExecutorService 实现 ExecutorService， ExecutorService 继承 Executor

public ThreadPoolExecutor(int corePoolSize,

int maximumPoolSize,

long keepAliveTime,

TimeUnit unit,

BlockingQueue<Runnable> workQueue,

ThreadFactory threadFactory,

RejectedExecutionHandler handler) {...}

corePoolSize : 核心线程数的大小

maximumPoolSize : 线程池中允许的最大线程数

keepAliveTime : 空闲线程允许的最大的存活时间

unit : 存活时间的单位

workQueue : 阻塞任务队列

threadFactory : 线程工厂用来创建线程

handler : 拒绝策略，针对当队列满了时新来任务的处理方式

在刚刚创建线程池的时候，内部线程的数量是 0，当首个任务进行添加的时候，会根据参数的配置进行线程的创建，并随着任务数的增加，会逐渐创建新的线程直到不能创建新的线程为止。不能创建新的线程后，会将来的任务存放到阻塞队列中，让空闲的线程去处理。当没有空闲线程并且队列满了时候就会采用拒绝策略去丢弃或者其他策略来处理。 拒绝策略主要有四种，不同的拒绝策略有不同的使用场景，需要根据情况决定使用。

## ThreadPoolExecutor源码分析

根据源码可以发现整个线程池大致分为 3 个部分，1. 是创建 worker 线程，2. 添加任务到 workQueue; 3.worker 线程执行具体任务

# 加密算法

加密算法可以归结为三大类：哈希算法、对称加密算法、非对称加密算法。

哈希算法：如MD5算法，SHA系列算法

对称加密算法：DES算法，3DES算法，AES算法

非对称加密算法：RSA算法