# 普通容器

## 1 Map

最基本的map是HashMap，

HashMap是无序的，那有没有有序的map呢？

TreeMap和LinkedHashMap 有实现有什么不同？哪种好？

区别：TreeMap 使用k的比较器来实现大小的比较；

LinkedHashMap 的排序和key无关，按照节点的插入顺序或者访问顺序。

### HashMap

what：k-v, 随机

how: 底层数据结构是：数组+链表，对key做hash算法，使key尽量离散。

why：为什么这样实现？

1. key是确定的值，要快速找到其在内存中的位置。

Your：

### 2）TreeMap

what：k –v, k 是有序的

how：底层数据结构是 红黑树，同时有一个比较器。当插入一个元素时，

why： 相当于Hashmap中没有了数组（数组无法排序）。

your:

### 3) LinkedHashMap

是Hashmap的子类

what: 有两张排序方式：1 按照插入顺序 2 按照访问顺序

how: 使用双向链表记录数据（错误理解）。

其Entry继承自HashMap.Node，并多了前驱和后继成员。

巧妙的使用两个成员记录了插入或者访问顺序，使Entry构成了一个双向链表。

而且并不浪费查询性能。

当按照插入顺序排序时：每生成一个节点，并赋值前驱后继。

当按照访问顺序排序时：更改之前的前驱后继，并称为尾节点。

即节点在原来Hashmap的基础上，增加了两个成员变量，又成为了一个双向链表。

why:

your:

# 线程安全的容器

## 1 Queue

队列就是有进有出，多线程的时候同时进，同时出。

怎么实现的线程安全呢？怎么实现的阻塞呢？

分为阻塞队列和非阻塞队列

也可分为有界队列和无界队列

### LinkedBlockQueue

What：1 作为一个容器，多线程安全的去生产和去消费。

1. 根据构造函数的参数，可以是有界队列也可以是无界队列。
2. 当消费空的时候，消费线程要被阻塞；当生产满的时候，生产线程能被阻塞

how：

怎么实现的线程安全？

通过两个重入锁，putLock和takeLock，put的线程要获取putLock；take的线程要获取takeLock。

怎么实现的阻塞和及时通知？

使用Condition的await、signal，来实现生产线程和消费线程之间的信息传递。

### ArrayBlockingQueue

What：是一个阻塞队列，底层存储结构是一个数组。

当put和take的时候，都是线程安全的。（使用重入锁实现的线层安全，线程启去put或take的时候，都需要竞争一个全局锁，因此它的效率比起LinkedBlockingQueue要低）

当满的时候，put线程就等待，当有take线程消费了，就通知put线程； 当空的时候，take线程就等待，当有put线程生产了，就通知take线程去消费。（线程间的通信是使用Condition实现的）

### SynchronousQueue

## 2 Map

### ConcurrentHashMap

What：普通的HashMap不是线程安全的

How:

jdk7中使用分段锁实现。即其底层是好几个Hashmap。每次put的时候都会lock其对应的HashMap。

Jdk8 中 使用cas+ synchronized实现。当Put的时候，先命中数组 上的位置。如果没发生冲突，在这个位置上使用CAS操作进行替换。如果发生冲突，则synchronized 这个对象，相当于锁住了链表首部，然后再在链表中进行曹组。

Why： 这种实现思路非常巧妙，大大提高了效率。

# 内存

## 1 堆外内存

what： 堆外内存是属于操作系统管理的内存，不属于JVM管理。

how：

1 大小是多大?

a 默认是64M

b Runtime.getRuntime().maxMemory()：即同 –Xmx参数值一样大。

C 可用 -XX:MaxDirectMemorySize 重新设定

2 直接内存的申请

DirectByteBuffer类是Default的，外部不能访问。需要使用

ByteBuffer ByteBuffer.allocateDirect(int)来申请。

在申请的时候，先检查 剩余可使用的直接内存大小 是否能满足，如果不能满足, 会调用System.gc；如果仍不能满足，则抛出OOM异常。如果能满足，则分配内存，并初始胡Cleaner.

3 直接内存的回收

分为主动回收和被动回收。

被动回收：

a JVM 老生代满了，发生了Full Gc, 此时，会同时清理堆外内存。

b 自己调用System.gc(), 但不一定发生gc，或者-DisableExplicitGC

被禁用了，更不会发生。

C 通过在申请的时候，检查容量是否满足的时候，调用System.gc()



主动回收：

通过Clearer进行回收。这种回收方式同JVM中的回收是不一样的，其类似于C中的free函数。不做对象是否被是否的校验，直接把内存还给OS。

DirectByteBuffer implements了DirectBuffer，而DirectBuffer本身是public的，所以通过接口去调用内部的Clear对象来做clean方法

why：

为什么使用堆外内存？

1 减少IO的内存复制。

Linux分为内核态和用户态。如果使用堆内内存，我们发送Socket, 那么linux先要把字节数据数组内容从JVM拷贝到内核态的直接内存，然后再写入Socket发送出去；而使用堆外内存，就没必要再拷贝了。

2 减少了JVM的GC