线程安全的类

Vector底层实现是数组

String

StringBuffer

HashTable：k和v都不可以为null

线程不安全的类

ArrayList

HashMap：k和v都可以为null

LinkList线程不安全，如果想用链表来实现线程安全可以考虑使用ConcurrentQueue来实现。

并发容器

并发容器包 java.util.concurrent;

**Map数组**

HashMap非线程安全，k和v都可以为null。

TreeMap容器中的数据排好顺序

SortedMap也是排好序的，但是插入时效率较低

HashTable是线程安全的，所有的方法都加了锁，但是效率较低，k和v都不可以为null，是jdk比较早的实现类。

ConcurrentHashMap是线程安全的，并发专用容器效率比HashTable高.

HashTable是所有的插入方法都加了锁，ConcurrentHashMap是进行了分段插入，如果插入的值处在不同散列表中，就可以同时进行。

1.8以后貌似使用cas替代分段锁，放弃segment使用node+cas+synchronized

ConcurrentSkipListMap高并发情况下，有序的情况使用，跳表map。<http://blog.csdn.net/sunxianghuang/article/details/52221913>

非并发情况下使用TreeMap

Map和Set本质是一个东西，不过Map的集合包含k和v。而Set集合只有数据。

在并发不高的情况下可以使用Collections.sychronizedXXX()容器工具包，可以将不加锁的容器返回为加锁的容器

Collections.sychronizedXXX采用了装饰模式，将所有的容器传进去后直接调用原有方法，在原有方法上加锁。

CopyOnWrite

写时复制，当你要添加元素的时候，它会从新赋值一份元素，在复制的元素上进行修改

**Queue队列**

ConcurrentQueue并发队列：

Queue<> quque = new ConcurrentLinkedQueue<>();

queue.offer();向容器中添加数据，返回值boolean，当容器已满，添加失败时返回值为false；

queue.add();向容器中添加数据，返回值也为boolean，当容器已满时会抛出异常，返回值也代表是否已满

queue.poll();从容器头部取出节点数据，并删除队列中的节点，当容器为空时返回null

queue.peek();查看容器头的节点数据，不删除原有数据，当容器为空时返回null

BlockingQueue阻塞队列：

BlockingQueue<> queue = new ArrayBlockingQueue<>(【数组长度】);

保留队列原有功能

quque.add(); quque.offer(); queue.poll(); queue.peek();

queue.offer(【存储数据】，【等待时间】，【时间单位】);阻塞队列提供可以按时间段来阻塞的功能，添加超时后会放弃添加，在未超时时间内队列空闲就可添加成功。

quque.put();添加队列时如果队列已满就会进入阻塞，直到队列出现空闲。

queue.get();取出队列头数据，如果队列为空，就进入阻塞，直到有新数据添加进队列。

BlockingQueue<> queue = new LinkedBlockingQueue<>();

LinkedBlockingQueue称之为无界队列，可以无限相对列添加，但是当队列为空时取数据操作会阻塞。

其他用法和ArrayBlockingQueue相同。

BlockingQueue<> tasks = new DelayQueue<>();

放入队列中的元素要求实现Delayed接口，该接口又继承Comparable接口，因此要实现compareTo和getDelayed方法

任务队列

根据compareTo()排好顺序

通过getDelayed()获执行时间

最后通过task.take();启动定时任务

启动任务后，队列中的方法不会马上执行，只有当达到执行时间后才会执行并返回，从而达到演示执行目的

LinkTransferQueue<> strs = new LinkTransferQueue<>();

strs.transfer();

消费者先启动，生产者启动后，在往队列里扔数据前会先判断是否存在等候的消费者，如果存在则将生产的数据直接传给消费者而不是添加到队列中，适用于并发更高的情况

当生产者找不到消费者时，线程会阻塞，也就是我发送的消息必须有人处理，否则就会阻塞。用于处理实时消息。

当然该方法依旧可以使用put、add、offer方法正常的使消息进入到队列，不会发生阻塞。

BlockingQueue strs = new SynchronousQueue<>();

与transferQueue类似，但是transferQueue仍可以调用put、add、offer方法存入到队列中

SynchronousQueue是一个容量为0的队列，调用add或offer方法等同于向满队列添加元素，会抛异常和添加失败。

而当调用put方法时，因为队列已满，会进入阻塞状态。

put方法的底层实际调用transfer方法，当有消费者出现时，阻塞的生产者就会直接将消息发送给消费者，省去了入队和出队的过程。