Semaphore信号量

* 资源有限共享
* 停车场为例

import java.util.Random;  
import java.util.concurrent.Semaphore;  
import java.util.concurrent.TimeUnit;

public class CarDemo {  
  
 public static void main(String[] args) {  
 // 创建Semaphore (信号量对象)  
 Semaphore semaphore = new Semaphore(5);  
  
 Thread[] cars = new Thread[10];  
 for(int i=0; i<10; i++){  
 cars[i] = new Thread(()->{  
  
 // 请求许可  
 try {  
 TimeUnit.*SECONDS*.sleep(2);  
 semaphore.acquire();  
 System.*out*.println(Thread.*currentThread*().getName() + "可以进入停车场.");  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
  
 // 使用资源 (车子在停车场中做自己的事)  
 try {  
 int time = new Random().nextInt(10);  
 TimeUnit.*SECONDS*.sleep(time);  
 System.*out*.println(Thread.*currentThread*().getName() + "停留了" + time + "秒");  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
  
 // 释放资源 （离开）  
 semaphore.release();  
 System.*out*.println(Thread.*currentThread*().getName() + "离开停车场");  
 },"car["+i+']');  
  
 cars[i].start();  
  
 }  
 }  
  
}

并发集合

List set map

并发环境下

遍历的过程（读）中不容许更新操作（增删改）

ConcurrentCollection

**非阻塞式集合（Non-Blocking Collection） ConcurrentLinkedDeque**

这类集合也包括添加和移除数据的方法。如果方法不能立即被执行，则返回null或抛出异常，但是调用这个方法的线程不会被阻塞。

实例

添加大量的数据到一个列表中；

从同一个列表中移除大量的数据。

**阻塞式集合（Blocking Collection） LinkedBlockingDeque**

阻塞式集合（Blocking Collection）：这类集合包括添加和移除数据的方法。当集合已满或为空时，被调用的添加或者移除方法就不能立即被执行，那么调用这个方法的线程将被阻塞，一直到该方法可以被成功执行。

分析ConcurrentHashMap

HsahMap

DataStructure

**ConcurrentLinkedDeque**

[**add**](http://tool.oschina.net/uploads/apidocs/jdk-zh/java/util/concurrent/ConcurrentLinkedQueue.html#add(E))([E](http://tool.oschina.net/uploads/apidocs/jdk-zh/java/util/concurrent/ConcurrentLinkedQueue.html) e)           将指定元素插入此队列的尾部。

[**offer**](http://tool.oschina.net/uploads/apidocs/jdk-zh/java/util/concurrent/ConcurrentLinkedQueue.html#offer(E))([E](http://tool.oschina.net/uploads/apidocs/jdk-zh/java/util/concurrent/ConcurrentLinkedQueue.html) e)           将指定元素插入此队列的尾部。

[**poll**](http://tool.oschina.net/uploads/apidocs/jdk-zh/java/util/concurrent/ConcurrentLinkedQueue.html#poll())()        获取并移除此队列的头，如果此队列为空，则返回 null。

Node

**LinkedBlockingDeque**

|  |
| --- |
| [**LinkedBlockingDeque**](http://tool.oschina.net/uploads/apidocs/jdk-zh/java/util/concurrent/LinkedBlockingDeque.html#LinkedBlockingDeque())()            创建一个容量为 [Integer.MAX\_VALUE](http://tool.oschina.net/uploads/apidocs/jdk-zh/java/lang/Integer.html#MAX_VALUE) 的 LinkedBlockingDeque。 |
| [**LinkedBlockingDeque**](http://tool.oschina.net/uploads/apidocs/jdk-zh/java/util/concurrent/LinkedBlockingDeque.html#LinkedBlockingDeque(java.util.Collection))([Collection](http://tool.oschina.net/uploads/apidocs/jdk-zh/java/util/Collection.html)<? extends [E](http://tool.oschina.net/uploads/apidocs/jdk-zh/java/util/concurrent/LinkedBlockingDeque.html)> c)            创建一个容量为 [Integer.MAX\_VALUE](http://tool.oschina.net/uploads/apidocs/jdk-zh/java/lang/Integer.html#MAX_VALUE) 的 LinkedBlockingDeque，最初包含给定 collection 的元素，以该 collection 迭代器的遍历顺序添加。 |
| [**LinkedBlockingDeque**](http://tool.oschina.net/uploads/apidocs/jdk-zh/java/util/concurrent/LinkedBlockingDeque.html#LinkedBlockingDeque(int))(int capacity)            创建一个具有给定（固定）容量的 LinkedBlockingDeque。 |

[**put**](http://tool.oschina.net/uploads/apidocs/jdk-zh/java/util/concurrent/LinkedBlockingDeque.html#put(E))([E](http://tool.oschina.net/uploads/apidocs/jdk-zh/java/util/concurrent/LinkedBlockingDeque.html) e)   
          将指定的元素插入此双端队列表示的队列中（即此双端队列的尾部），必要时将一直等待可用空间。

[**take**](http://tool.oschina.net/uploads/apidocs/jdk-zh/java/util/concurrent/LinkedBlockingDeque.html#take())()   
          获取并移除此双端队列表示的队列的头部（即此双端队列的第一个元素），必要时将一直等待可用元素。

**HashMap entry<k,v>**

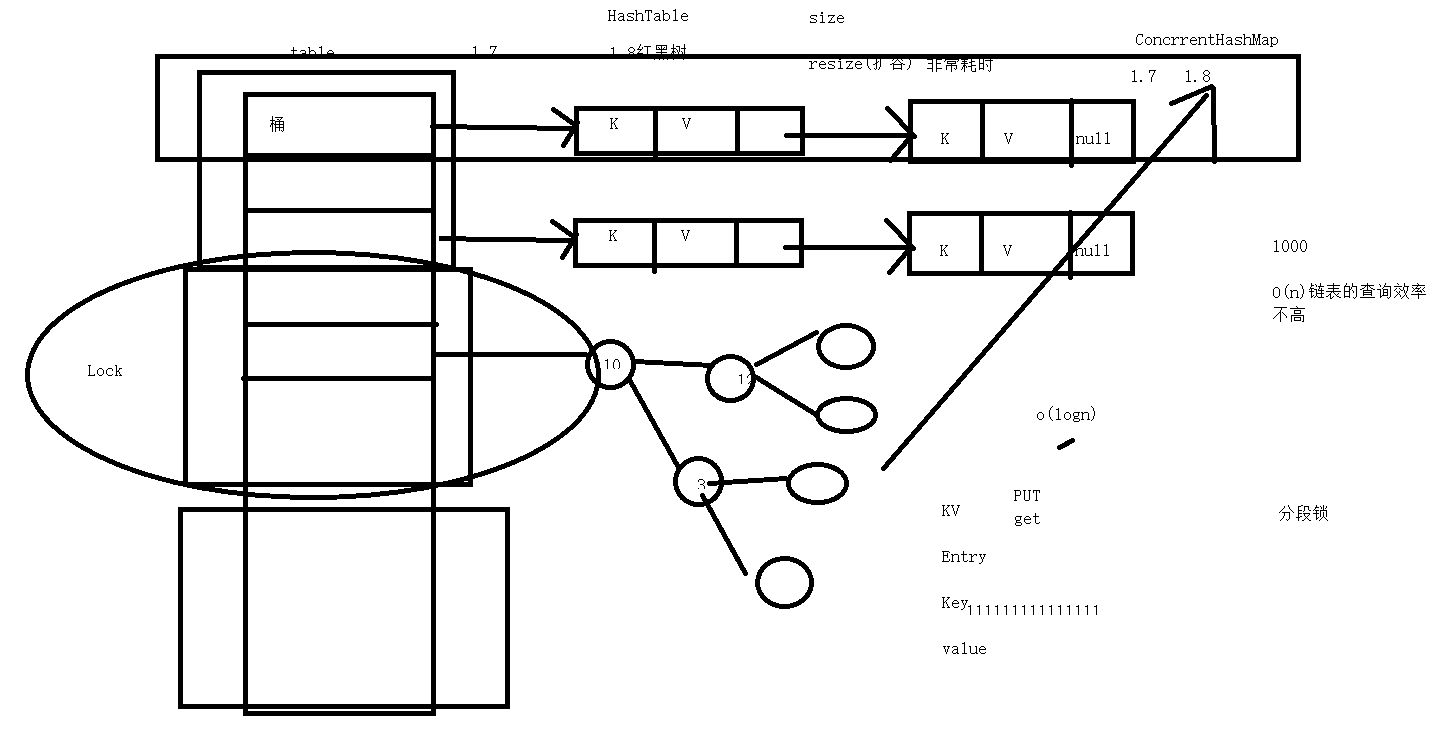
**LinkedTransferQueue 生产-消费**

**PriorityBlockingQueue 优先级**

[ArrayBlockingQueue](http://tool.oschina.net/uploads/apidocs/jdk-zh/java/util/concurrent/ArrayBlockingQueue.html)   
[ConcurrentHashMap](http://tool.oschina.net/uploads/apidocs/jdk-zh/java/util/concurrent/ConcurrentHashMap.html)   
[ConcurrentLinkedQueue](http://tool.oschina.net/uploads/apidocs/jdk-zh/java/util/concurrent/ConcurrentLinkedQueue.html)   
[ConcurrentSkipListMap](http://tool.oschina.net/uploads/apidocs/jdk-zh/java/util/concurrent/ConcurrentSkipListMap.html)   
[ConcurrentSkipListSet](http://tool.oschina.net/uploads/apidocs/jdk-zh/java/util/concurrent/ConcurrentSkipListSet.html)   
[CopyOnWriteArrayList](http://tool.oschina.net/uploads/apidocs/jdk-zh/java/util/concurrent/CopyOnWriteArrayList.html)   
[CopyOnWriteArraySet](http://tool.oschina.net/uploads/apidocs/jdk-zh/java/util/concurrent/CopyOnWriteArraySet.html)

**HashMap**

**数组+链表**



* **HashTable**

1. 每一个数组元素叫做桶，桶里放的是列表，每个列表的元素就是键值对

2. 数组的长度size一开始是固定的，但是可以扩容的（很耗时）

3. 扩容因子0.75，当桶的数量达到size的75%时，就会扩容

**- JDK1.7版本的列表查询效率不高，o(n)**

**- JDK1.8版本增加了红黑树,查询效率提高了，o(logn)**

* **红黑树：**

1. 根节点和叶子节点是黑色的，其余的节点是红色的

2. 左点小于父节点，右节点大于父节点（类似平衡二叉树），查询效率更高

ConcurrentHashMap：

* 分段式加锁，分段锁：segment

原子操作

CAS

AtomicInteger

[tomicBoolean](http://tool.oschina.net/uploads/apidocs/jdk-zh/java/util/concurrent/atomic/AtomicBoolean.html)   
[AtomicInteger](http://tool.oschina.net/uploads/apidocs/jdk-zh/java/util/concurrent/atomic/AtomicInteger.html)

[AtomicLong](http://tool.oschina.net/uploads/apidocs/jdk-zh/java/util/concurrent/atomic/AtomicLong.html)

Long

64

32 32

CAS

CompareAndSet

 此引用所引用的对象类型。

ABA问题

1. -》120--》100
2. 》120

[AtomicStampedReference](http://tool.oschina.net/uploads/apidocs/jdk-zh/java/util/concurrent/atomic/AtomicStampedReference.html)

100 1---->120 2----->100----3

100 1------120