### 7. wait()/notify()与sleep()

7.1 wait()/notify()与sleep()的异同

1.相同点

①.都可以使线程堵塞

②.都可以响应中断

2.不同点

①.wait()、notify()方法必须写在同步方法中，是为了防止死锁和永久等待，使线程更安全，而sleep()方法不需要有这个限制。

②.wait()方法调用后会释放锁sleep()方法调用后不会释放锁。

③.sleep()方法必须要指定时间参数；wait()方法可以指定时间参数。

④.两个方法所属类不同，sleep()方法属于Thread类；wait()属于Object类中，放在Object类中是因为Java中每个类都可以是一把锁。

7.2 为什么线程通信的方法wait()/notify()定义在Object类，而sleep()定义在Thread类

wait、notify、notifyAll被设计在Object类中的原因是，JAVA提供的锁是对象级的而不是线程级的，每个对象都有个锁，而线程是可以获得这个对象的。因此线程需要等待某些锁，那么只要调用对象中的wait()方法便可以了。而wait()方法如果定义在Thread类中的话，那么线程正在等待的是哪个锁就不明确了。这也就是说wait，notify和notifyAll都是锁级别的操作，所以把他们定义在Object类中是因为锁是属于对象的原因。

sleep的作用是：让线程在预期的时间内执行，其他时候不要来占用CPU资源。从上面的话术中，便可以理解为sleep是属于线程级别的，它是为了让线程在限定的时间后去执行。而且sleep方法是不会去释放锁的

### 9. volatile关键字和synchornized关键字

volatile是Java提供的一种轻量级的同步机制，在并发编程中，它也扮演着比较重要的角色。同synchronized相比（synchronized通常称为重量级锁），volatile更轻量级，相比使用synchronized所带来的庞大开销，倘若能恰当的合理的使用volatile，自然是美事一桩。

**可见性：**一个线程修改共享变量时，其他线程能够立即知道这个修改；

使用Volatile关键字将解决线程之间可见性, 强制线程每次读取该值的时候都去“主内存”中取值

9.1 volatile是什么，适用场合及其作用（建议了解一下原理）

* 对变量的写操作不依赖于当前值。
* 该变量没有包含在具有其他变量的不变式中。

9.2 synchorinzed的作用及使用方式（建议了解一下原理）

9.3 volatile和synchornized的关系

1.volatile本质是在告诉jvm当前变量在寄存器（工作内存）中的值是不确定的，需要从主存中读取；synchronized则是锁定当前变量，只有当前线程可以访问该变量，其他线程被阻塞住；

2.volatile仅能使用在变量级别；synchronized则可以使用在变量、方法、和类级别的；

3.volatile仅能实现变量的修改可见性，不能保证原子性；而synchronized则可以保证变量的修改可见性和原子性；

4.volatile不会造成线程的阻塞；synchronized可能会造成线程的阻塞；

5.volatile标记的变量不会被编译器优化；synchronized标记的变量可以被编译器优化。

**10. synchornized和Reentrantlock的区别**

① 底层实现上来说，synchronized 是JVM层面的锁，是Java关键字，通过monitor对象来完成（monitorenter与monitorexit），对象只有在同步块或同步方法中才能调用wait/notify方法，ReentrantLock 是从jdk1.5以来（java.util.concurrent.locks.Lock）提供的API层面的锁。

② **是否可手动释放：**

synchronized 不需要用户去手动释放锁，synchronized 代码执行完后系统会自动让线程释放对锁的占用； ReentrantLock则需要用户去手动释放锁，如果没有手动释放锁，就可能导致死锁现象。一般通过lock()和unlock()方法配合try/finally语句块来完成，使用释放更加灵活。

③ **是否可中断**

synchronized是不可中断类型的锁，除非加锁的代码中出现异常或正常执行完成； ReentrantLock则可以中断，可通过trylock(long timeout,TimeUnit unit)设置超时方法或者将lockInterruptibly()放到代码块中，调用interrupt方法进行中断。

④  **是否公平锁**

synchronized为非公平锁 ReentrantLock则即可以选公平锁也可以选非公平锁，通过构造方法new ReentrantLock时传入boolean值进行选择，为空默认false非公平锁，true为公平锁。

⑤ **锁是否可绑定条件Condition**

synchronized不能绑定； ReentrantLock通过绑定Condition结合await()/singal()方法实现线程的精确唤醒，而不是像synchronized通过Object类的wait()/notify()/notifyAll()方法要么随机唤醒一个线程要么唤醒全部线程。

⑥ **锁的对象**

synchronzied锁的是对象，锁是保存在对象头里面的，根据对象头数据来标识是否有线程获得锁/争抢锁；ReentrantLock锁的是线程，根据进入的线程和int类型的state标识锁的获得/争抢。

### 11. JUC包下面的一些类

JUC: <https://www.jianshu.com/p/1f19835e05c0>

四大常用类：

<https://blog.csdn.net/weixin_43232955/article/details/99634321?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-2.control&dist_request_id=&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-2.control>

### 1. CountDownLatch-闭锁

### 2. CyclicBarrier-循环栅栏

### 3. Semaphore-信号量

### 4. Exchanger-线程数据交换器

### 12. Java并发相关代码

12.1 实现两个线程轮流打奇偶数

https://blog.csdn.net/wang603603/article/details/89349881

12.2 实现生产者消费者模型

https://blog.csdn.net/weixin\_35985385/article/details/80967737

12.3 单例模式的书写及相关问题

懒汉模式、饿汉模式

<https://blog.csdn.net/fd2025/article/details/79711198>

### 13.集合容器概述

**1、什么是集合**

**Java集合是什么：**

Java 中的集合类库可以帮助我们在程序设计中实现传统的数据结构。

**Java的集合类是一个用来存放对象的容器，有以下特点:**

1、Java集合只能存放对象。加入添加了一个基本数据类型，会被自动装箱后存入集合。

2、集合存放的是多个对象的引用，对象本身是在堆内存中的。

3、集合可以存放不同类型，不限数量的数据类型。

**集合分三种**:1、Set 2 、List 3、Map

集合类是用来存放某类对象的。集合类有一个共同特点，就是它们只容纳对象（实际上是对象名，即指向地址的指针）。这一点和数组不同，数组可以容纳对象和简单数据。如果在集合类中既想使用简单数据类型，又想利用集合类的灵活性，就可以把简单数据类型数据变成该数据类型类的对象，然后放入集合中处理，但这样执行效率会降低。  
集合类容纳的对象都是Object类的实例，一旦把一个对象置入集合类中，它的类信息将丢失，也就是说，集合类中容纳的都是指向Object类对象的指针。这样的设计是为了使集合类具有通用性，因为Object类是所有类的祖先，所以可以在这些集合中存放任何类而不受限制。当然这也带来了不便，这令使用集合成员之前必须对它重新造型。

**4、使用集合框架的好处  
5、常用的集合类有哪些？  
6、List，Set，Map三者的区别？List、Set、Map 是否继承自 Collection 接口？List、Map、Set 三个接口存取元素时，各有什么特点？  
7、集合框架底层数据结构  
8、哪些集合类是线程安全的？**

其中Vector、HashTable、Properties是线程安全的。其中ArrayList、LinkedList、HashSet、TreeSet、HashMap、TreeMap等都是线程不安全的。（线程不安全是指：当多个线程访问同一个集合或Map时，如果有超过一个线程修改了ArrayList集合，则程序必须手动保证该集合的同步性。）

当多个并发同时对非线程安全的集合进行增删改的时候会破坏这些集合的数据完整性

**9、Java集合的快速失败机制 “fail-fast”？**

https://blog.csdn.net/ch717828/article/details/46892051?utm\_medium=distribute.pc\_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromMachineLearnPai2%7Edefault-1.control&dist\_request\_id=1328680.54190.16164221148386353&depth\_1-utm\_source=distribute.pc\_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromMachineLearnPai2%7Edefault-1.control  
**10、怎么确保一个集合不能被修改？**

https://blog.csdn.net/fanbaodan/article/details/103237298

**14.List接口**  
**1、迭代器 Iterator 是什么？**

**它是一个对象，它的工作是遍历并选择序列中的对象，而客户端的程序员不必知道或关心该序列底层的结构**

**2、Iterator 怎么使用？有什么特点？**

（1） Iterator遍历集合元素的过程中不允许线程对集合元素进行修改，否则会抛出ConcurrentModificationEception的异常。

（2）Iterator遍历集合元素的过程中可以通过remove方法来移除集合中的元素，删除的是上一次Iterator.next()方法返回的对象。

（3）Iterator必须依附于一个集合类对象而存在，Iterator本身不具有装载数据对象的功能。

（4）next（）方法，该方法通过游标指向的形式返回Iterator下一个元素。

**3、如何边遍历边移除 Collection 中的元素？**

Iterator<Integer> it = list.iterator();

while(it.hasNext()){

\*// do something\*

it.remove();

}

**4、Iterator 和 ListIterator 有什么区别？**

（1）ListIterator有add()方法，可以向List中添加对象，而Iterator不能

（2）ListIterator和Iterator都有hasNext()和next()方法，可以实现顺序向后遍历，但是ListIterator有hasPrevious()和previous()方法，可以实现逆向（顺序向前）遍历。Iterator就不可以。

（3）ListIterator可以定位当前的索引位置，nextIndex()和previousIndex()可以实现。Iterator没有此功能。

（4）都可实现删除对象，但是ListIterator可以实现对象的修改，set()方法可以实现。Iierator仅能遍历，不能修改。

**5、****遍历一个 List 有哪些不同的方式？每种方法的实现原理是什么？Java 中 List 遍历的最佳实践是什么？**

https://blog.csdn.net/HJF\_HUANGJINFU/article/details/51220253

**6、说一下 ArrayList 的优缺点**

ArrayList的优点如下：

ArrayList 底层以数组实现，是一种随机访问模式。ArrayList 实现了 RandomAccess 接口，因此查找的时候非常快。

ArrayList 在顺序添加一个元素的时候非常方便。

ArrayList 的缺点如下：

删除元素的时候，需要做一次元素复制操作。如果要复制的元素很多，那么就会比较耗费性能。

插入元素的时候，也需要做一次元素复制操作，缺点同上。

ArrayList 比较适合顺序添加、随机访问的场景。

**7、如何实现数组和 List 之间的转换？**

https://blog.csdn.net/qq\_45239139/article/details/103523828  
**8、ArrayList 和 LinkedList 的区别是什么？**

（1）ArrayList的实现是基于数组，LinkedList的实现是基于双向链表。

（2） 对于随机访问，ArrayList优于LinkedList，ArrayList可以根据下标以O(1)时间复杂度对元素进行随机访问。而LinkedList的每一个元素都依靠地址指针和它后一个元素连接在一起，在这种情况下，查找某个元素的时间复杂度是O(n) 。

（3）对于插入和删除操作，LinkedList优于ArrayList，因为当元素被添加到LinkedList任意位置的时候，不需要像ArrayList那样重新计算大小或者是更新索引。

（4） LinkedList比ArrayList更占内存，因为LinkedList的节点除了存储数据，还存储了两个引用，一个指向前一个元素，一个指向后一个元素。

**9、ArrayList 和 Vector 的区别是什么？**

表格

描述已自动生成

**10、插入数据时，ArrayList、LinkedList、Vector谁速度较快？阐述 ArrayList、Vector、LinkedList 的存储性能和特性？**

https://blog.csdn.net/strggle\_bin/article/details/109363125  
**11、多线程场景下如何使用 ArrayList？**

1. 最常用的方法是**通过 Collections 的 synchronizedList 方法**将 ArrayList 转换成线程安全的容器后再使用。

2）为list.add()方法加锁

3）使用线程安全的 CopyOnWriteArrayList 代替线程不安全的 ArrayList。

4）使用ThreadLocal变量确保线程封闭性(封闭线程往往是比较安全的， 但由于使用ThreadLocal封装变量，相当于把变量丢进执行线程中去，每new一个新的线程，变量也会new一次，一定程度上会造成性能[内存]损耗，但其执行完毕就销毁的机制使得ThreadLocal变成比较优化的并发解决方案)。

**12、为什么 ArrayList 的 elementData 加上 transient 修饰？**

http://www.javashuo.com/article/p-moiudvqi-gy.html  
**13、List 和 Set 的区别**

同：

1：都是Collection的子接口；

2：List和Set存储元素的类型都是引用数据类型。

异：

1：List的实现类：ArrayList、LinkedList、vector；Set的实现类：HashSet、LinkedSet、TreeSetList存储的元素是有序的、可重复的。Set存储的元素是无序的、不可重复的。

2：Set中表示添加，无顺序。List中表示在最后一位追加，有顺序。

3： List可根据索引获取特定的值，Set不能获取特定的值。

4：List有index，Set有HashCode。Set要重写HashCode（）和equals（）方法。

# synchronized保证线程安全的原理（jvm层面）

<https://blog.csdn.net/cckkpp/article/details/88595419>

*二、Set接口*  
1、说一下 HashSet 的实现原理？

https://blog.csdn.net/qq\_41026809/article/details/90449073  
2、HashSet如何检查重复？HashSet是如何保证数据不可重复的？

map.containsKey(o);   
3、HashSet与HashMap的区别

仅保存对象和保存键值对

|  |  |
| --- | --- |
| \*HashMap\* | \*HashSet\* |
| HashMap实现了Map接口 | HashSet实现了Set接口 |
| HashMap储存键值对 | HashSet仅仅存储对象 |
| 使用put()方法将元素放入map中 | 使用add()方法将元素放入set中 |
| HashMap中使用键对象来计算hashcode值 | HashSet使用成员对象来计算hashcode值，对于两个对象来说hashcode可能相同，所以equals()方法用来判断对象的相等性，如果两个对象不同的话，那么返回false |
| HashMap比较快，因为是使用唯一的键来获取对象 | HashSet较HashMap来说比较慢 |

三、Queue  
1、BlockingQueue是什么？  
2、在 Queue 中 poll()和 remove()有什么区别？

<https://blog.csdn.net/qq_41026809/article/details/90647569>

poll()和remove()都将**移除**并且返回对头，但是在poll()在队列为空时返回null，而remove()会抛出NoSuchElementException异常。

**四、Map接口**  
1、说一下 HashMap 的实现原理？

https://blog.csdn.net/samniwu/article/details/90550196  
2、HashMap在JDK1.7和JDK1.8中有哪些不同？HashMap的底层实现  
3、JDK1.8之前  
4、JDK1.8之后  
5、JDK1.7 VS JDK1.8 比较  
6、HashMap的put方法的具体流程？

https://blog.csdn.net/the\_one\_and\_only/article/details/81665098?utm\_medium=distribute.pc\_relevant\_t0.none-task-blog-2%7Edefault%7EBlogCommendFromMachineLearnPai2%7Edefault-1.control&dist\_request\_id=&depth\_1-utm\_source=distribute.pc\_relevant\_t0.none-task-blog-2%7Edefault%7EBlogCommendFromMachineLearnPai2%7Edefault-1.control  
7、HashMap的扩容操作是怎么实现的？  
8、HashMap是怎么解决哈希冲突的？  
9、什么是哈希？

**HashMap对象的hashcode 是取key的hashcode和value的hashcode 按位异或运算,然后把他们依次相加**

10、什么是哈希冲突？  
11、HashMap的数据结构

数组+链表

Jdk1.8 数组+链表+红黑树（链表数>8转为红黑树）  
12、hash()函数  
13、JDK1.8新增[红黑树](file:////jump/super-jump/word%3fword=%25E7%25BA%25A2%25E9%25BB%2591%25E6%25A0%2591" \t "_blank)

总结  
1、能否使用任何类作为 Map 的 key？

可以 只要他的equals函数遵从hashcode规律

1. 同一个对象未发生改变时多次调用hashCode()返回值必须相同，
2. 两个对象equals不相等，那么两对象的hashCode()返回必定不同（此处可用来提高哈希表性能）
3. 两个对象的hashCode()返回值相同，两对象不一定相同，还需要通过equals()再次判断
4. 当equals方法被重写时，通常有必要重写 hashCode 方法

2、为什么HashMap中String、Integer这样的包装类适合作为K？

String、Integer等包装类的特性能够保证Hash值的不可更改性和计算准确性，能够有效的减少Hash碰撞的几率~

因为

* 它们都是final修饰的类，不可变性，保证key的不可更改性，不会存在获取hash值不同的情况~
* 它们内部已重写了equals()、hashCode()等方法，遵守了HashMap内部的规范

3、如果使用Object作为HashMap的Key，应该怎么办呢？

需要让key object覆盖如下hashCode()和equals(Object obj)的实现。

4、HashMap为什么不直接使用hashCode()处理后的哈希值直接作为table的下标？

hashCode()方法返回的是int整数类型，其范围为-(2 ^ 31)~(2 ^ 31 - 1)，约有40亿个映射空间，而HashMap的容量范围是在16（初始化默认值）~2 ^ 30，HashMap通常情况下是取不到最大值的，并且设备上也难以提供这么多的存储空间，从而导致通过hashCode()计算出的哈希值可能不在数组大小范围内，进而无法匹配存储位置；

那怎么解决呢？

HashMap自己实现了自己的hash()方法，通过两次扰动使得它自己的哈希值高低位自行进行异或运算，降低哈希碰撞概率也使得数据分布更平均；

在保证数组长度为2的幂次方的时候，使用hash()运算之后的值与运算（&）（数组长度 - 1）来获取数组下标的方式进行存储，这样一来是比取余操作更加有效率，二来也是因为只有当数组长度为2的幂次方时，h&(length-1)才等价于h%length，三来解决了“哈希值与数组大小范围不匹配”的问题

5、HashMap 的长度为什么是2的幂次方

为了能让 HashMap 存取高效，尽量较少碰撞，也就是要尽量把数据分配均匀，每个链表/红黑树长度大致相同。这个实现就是把数据存到哪个链表/红黑树中的算法。

这个算法应该如何设计呢？

我们首先可能会想到采用%取余的操作来实现。但是，重点来了：“取余(%)操作中如果除数是2的幂次则等价于与其除数减一的与(&)操作（也就是说 hash%length==hash&(length-1)的前提是 length 是2的 n 次方；）。” 并且 采用二进制位操作 &，相对于%能够提高运算效率，这就解释了 HashMap 的长度为什么是2的幂次方

<https://blog.csdn.net/weixin_43907800/article/details/104995655>

6、HashMap 与 HashTable 有什么区别？

<https://blog.csdn.net/u010983881/article/details/49762595>

hashtable线程安全

7、如何决定使用 HashMap 还是 TreeMap？

如果你需要得到一个有序的结果时就应该使用TreeMap（因为HashMap中元素的排列顺序是不固定的）。除此之外，由于HashMap有更好的性能，所以大多不需要排序的时候我们会使用HashMap。

8、HashMap 和 ConcurrentHashMap 的区别

**总结ConcurrentHashMap就是一个分段的hashtable ，根据自定的hashcode算法生成的对象来获取对应hashcode的分段块进行加锁，不用整体加锁，提高了效率**

9、ConcurrentHashMap 和 Hashtable 的区别？

**总结ConcurrentHashMap就是一个分段的hashtable ，根据自定的hashcode算法生成的对象来获取对应hashcode的分段块进行加锁，不用整体加锁，提高了效率**

10、ConcurrentHashMap 底层具体实现知道吗？实现原理是什么？

![图示

描述已自动生成]()

在JDK1.7版本中，ConcurrentHashMap的数据结构是由一个Segment数组和多个HashEntry组成，Segment数组的意义就是将一个大的table分割成多个小的table来进行加锁，也就是上面的提到的锁分离技术，而每一个Segment元素存储的是HashEntry数组+链表，这个和HashMap的数据存储结构一样

JDK1.8的实现已经摒弃了Segment的概念，而是直接用Node数组+链表+红黑树的数据结构来实现，并发控制使用Synchronized和CAS来操作，整个看起来就像是优化过且线程安全的HashMap，虽然在JDK1.8中还能看到Segment的数据结构，但是已经简化了属性，只是为了兼容旧版本

说明：ConcurrentHashMap的数据结构（数组+链表+红黑树），桶中的结构可能是链表，也可能是红黑树，红黑树是为了提高查找效率。

**辅助工具类**  
1、Array 和 ArrayList 有何区别？

可以将  ArrayList想象成一种“会自动扩增容量的Array”。   
Array（[]）：最高效；但是其容量固定且无法动态改变；   
     ArrayList：  容量可动态增长；但牺牲效率；

2、如何实现 Array 和 List 之间的转换？

Array转为list：Arrays.asList(strs)

List转为array：list.toArray()

3、comparable 和 comparator的区别？

两种方法各有优劣， 用Comparable 简单， 只要实现Comparable接口的对象直接就成为一个可以比较的对象，但是需要修改源代码。 用Comparator 的好处是不需要修改源代码， 而是另外实现一个比较器， 当某个自定义的对象需要作比较的时候，把比较器和对象一起传递过去就可以比大小了， 并且在Comparator 里面用户可以自己实现复杂的可以通用的逻辑，使其可以匹配一些比较简单的对象，那样就可以节省很多重复劳动了。

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成  
4、Collection 和 Collections 有什么区别？

**1.java.util.Collection 是一个集合接口。**

它提供了对集合对象进行基本操作的通用接口方法。Collection接口在Java 类库中有很多具体的实现。Collection接口的意义是为各种具体的集合提供了最大化的统一操作方式。

以下接口实现了Collection接口：

set，list

**2.java.util.Collections 是一个包装类。**  
它包含有各种有关集合操作的静态多态方法。此类不能实例化，就像一个工具类，服务于Java的Collection框架。

Collections.sort(list);

5、TreeMap 和 TreeSet 在[排序](file:////jump/super-jump/word%3fword=%25E6%258E%2592%25E5%25BA%258F" \t "_blank)时如何比较元素？Collections 工具类中的 sort()方法如何比较元素？

TreeSet要求存放的对象所属的类必须实现Comparable接口，该接口提供了比较元素的compareTo()方法，当插入元素时会回调该方法比较元素的大小。TreeMap要求存放的键值对映射的键必须实现Comparable接口从而根据键对元素进行排序。Collections工具类的sort方法有两种重载的形式，第一种要求传入的待排序容器中存放的对象必须实现Comparable接口以实现元素的比较；第二种不强制性的要求容器中的元素必须可比较，但是要求传入第二个参数，参数是Comparator接口的子类型（需要重写compare方法实现元素的比较），相当于一个临时定义的排序规则，其实就是通过接口注入比较元素大小的算法，也是对回调模式的应用（Java中对函数式编程的支持）

**Java里所有类都继承哪个类？**

Object类,它是所有类的父类。你可以继承它的很多方法,比如常用的toString()等。