## Java多线程笔记

### 1.多线程的优点：

1）资源利用率更好  
2）程序设计在某些情况下更简单  
3）程序响应更快

### 2.线程安全

在同一程序中运行多个线程本身不会导致问题，问题在于多个线程访问了相同的资源。如同一内存区（变量、数组或对象）、系统（数据库、web services等）或文件。实际上，这些问题只有在一或多个线程向这些资源做了写操作时才有可能发生，只要资源没有发生变化，多个线程读取相同的资源就是安全的。

当两个线程竞争同一资源时，如果对资源的访问顺序敏感，就称存在竞态条件。导致竞态条件发生的代码区称作临界区。

如果一个资源的创建，使用，销毁都在同一个线程内完成，且永远不会脱离该线程的控制，则该资源的使用就是线程安全的。

### 3.java同步块

Java中的同步块用synchronized标记。同步块在Java中是同步在某个对象上。所有同步在一个对象上的同步块在同时只能被一个线程进入并执行操作。所有其他等待进入该同步块的线程将被阻塞，直到执行该同步块中的线程退出。

有四种不同的同步块：

实例方法

静态方法

实例方法中的同步块

静态方法中的同步块

实例方法同步：

|  |
| --- |
| public synchronized void add(int value) {  this.count += value;  } |

Java实例方法同步是同步在拥有该方法的对象上。这样，每个实例其方法同步都同步在不同的对象上，即该方法所属的实例。只有一个线程能够在实例方法同步块中运行。如果有多个实例存在，那么一个线程一次可以在一个实例同步块中执行操作。一个实例一个线程。

静态方法同步：

|  |
| --- |
| public static synchronized void add(int value) {  count += value;  } |

静态方法的同步是指同步在该方法所在的类对象上。因为在Java虚拟机中一个类只能对应一个类对象，所以同时只允许一个线程执行同一个类中的静态同步方法。

实例方法中的同步块：

|  |
| --- |
| public void add(int value){  synchronized(this){  this.count += value;  }  } |

注意Java同步块构造器用括号将对象括起来。在上例中，使用了“this”，即为调用add方法的实例本身。在同步构造器中用括号括起来的对象叫做监视器对象。上述代码使用监视器对象同步，同步实例方法使用调用方法本身的实例作为监视器对象。一次只有一个线程能够在同步于同一个监视器对象的Java方法内执行。

下面两个例子都同步他们所调用的实例对象上，因此他们在同步的执行效果上是等效的。

|  |
| --- |
| public class MyClass {  public synchronized void test(String msg1, String msg2){  log.writeln(msg1);  log.writeln(msg2);  }  public void test2(String msg1, String msg2){  synchronized(this){  log.writeln(msg1);  log.writeln(msg2);  }  }  } |

静态方法中的同步块：

|  |
| --- |
| public class MyClass {  public static synchronized void log1(String msg1, String msg2){  log.writeln(msg1);  log.writeln(msg2);  }  public static void log2(String msg1, String msg2){  synchronized(MyClass.class){  log.writeln(msg1);  log.writeln(msg2);  }  }  } |

这两个方法不允许同时被线程访问。如果第二个同步块不是同步在MyClass.class这个对象上。那么这两个方法可以同时被线程访问。

### 关于HashMap和Hashtable线程安全性的理解

HashMap和HashTable都实现了Map接口，但决定用哪一个之前先要弄清楚它们之间的分别。主要的区别有：线程安全性，同步(synchronization)，以及速度。

(1).HashMap几乎可以等价于Hashtable，除了HashMap是非synchronized的，并可以接受null(HashMap可以接受为null的键值(key)和值(value)，而Hashtable则不行)。

(2).HashMap是非synchronized，而Hashtable是synchronized，这意味着Hashtable是线程安全的，多个线程可以共享一个Hashtable；而如果没有正确的同步的话，多个线程是不能共享HashMap的。Java 5提供了ConcurrentHashMap，它是HashTable的替代，比HashTable的扩展性更好。

(3).由于Hashtable是线程安全的也是synchronized，所以在单线程环境下它比HashMap要慢。如果你不需要同步，只需要单一线程，那么使用HashMap性能要好过Hashtable。

(4).HashMap不能保证随着时间的推移Map中的元素次序是不变的。