线程安全与共享资源

原文链接 作者：Jakob Jenkov 译者：毕冉 校对：丁一

允许被多个线程同时执行的代码称作线程安全的代码。线程安全的代码不包含竞态条件。当多个线程同时更新共享资源时会引发竞态条件。因此，了解Java线程执行时共享了什么资源很重要。

局部变量

局部变量存储在线程自己的栈中。也就是说，局部变量永远也不会被多个线程共享。所以，基础类型的局部变量是线程安全的。下面是基础类型的局部变量的一个例子：

```
1  public void someMethod(){
2
3    long threadSafeInt = 0;
4
5    threadSafeInt++;
6  }
```

局部的对象引用

对象的局部引用和基础类型的局部变量不太一样。尽管引用本身没有被共享，但引用所指的对象并没有存储在线程的栈内。所有的对象都存在共享堆中。如果在某个方法中创建的对象不会逃逸出（*译者注：即该对象不会被其它方法获得，也不会被非局部变量引用到*）该方法，那么它就是线程安全的。实际上，哪怕将这个对象作为参数传给其它方法，只要别的线程获取不到这个对象，那它仍是线程安全的。下面是一个线程安全的局部引用样例：

```
01  public void someMethod(){
02
03    LocalObject localObject = new LocalObject();
04
05    localObject.callMethod();
06    method2(localObject);
07  }
08
09  public void method2(LocalObject localObject){
10    localObject.setValue("value");
11  }
```

样例中LocalObject对象没有被方法返回，也没有被传递给someMethod()方法外的对象。每个执行someMethod()的线程都会创建自己的LocalObject对象，并赋值给localObject引用。因此，这里的LocalObject是线程安全的。事实上，整个someMethod()都是线程安全的。即使将LocalObject作为参数传给同一个类的其它方法或其它类的方法时，它仍然是线程安全的。当然，如果LocalObject通过某些方法被传给了别的线程，那它就不再是线程安全的了。

对象成员

对象成员存储在堆上。如果两个线程同时更新同一个对象的同一个成员，那这个代码就不是线程安全的。下面是一个样例：

```
1  public class NotThreadSafe{
2      StringBuilder builder = new StringBuilder();
3
4      public add(String text){
```

```
5         this.builder.append(text);
6     }
7 }
```

如果两个线程同时调用同一个NotThreadSafe实例上的add()方法，就会有竞态条件问题。例如：

```
01 NotThreadSafe sharedInstance = new NotThreadSafe();
02
03 new Thread(new MyRunnable(sharedInstance)).start();
04 new Thread(new MyRunnable(sharedInstance)).start();
05
06 public class MyRunnable implements Runnable{
07   NotThreadSafe instance = null;
08
09   public MyRunnable(NotThreadSafe instance){
10     this.instance = instance;
11   }
12
13   public void run(){
14     this.instance.add("some text");
15   }
16 }
```

注意两个MyRunnable共享了同一个NotThreadSafe对象。因此，当它们调用add()方法时会造成竞态条件。

当然，如果这两个线程在不同的NotThreadSafe实例上调用call()方法，就不会导致竞态条件。下面是稍微修改后的例子：

```
1 new Thread(new MyRunnable(new NotThreadSafe())).start();
2 new Thread(new MyRunnable(new NotThreadSafe())).start();
```

现在两个线程都有自己单独的NotThreadSafe对象，调用add()方法时就会互不干扰，再也不会有竞态条件问题了。所以非线程安全的对象仍可以通过某种方式来消除竞态条件。

线程控制逃逸规则

线程控制逃逸规则可以帮助你判断代码中对某些资源的访问是否是线程安全的。

```
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。
```

资源可以是对象，数组，文件，数据库连接，套接字等等。Java中你无需主动销毁对象，所以“销毁”指不再有引用指向对象。

即使对象本身线程安全，但如果该对象中包含其他资源（文件，数据库连接），整个应用也许就不再是线程安全的了。比如2个线程都创建了各自的数据库连接，每个连接自身是线程安全的，但它们所连接到的同一个数据库也许不是线程安全的。比如，2个线程执行如下代码：

```
检查记录X是否存在，如果不存在，插入X
```

如果两个线程同时执行，而且碰巧检查的是同一个记录，那么两个线程最终可能都插入了记录：

```
线程1检查记录X是否存在。检查结果：不存在
线程2检查记录X是否存在。检查结果：不存在
线程1插入记录X
线程2插入记录X
```

同样的问题也会发生在文件或其他共享资源上。因此，区分某个线程控制的对象是资源本身，还是仅仅到某个资源的引用很重要。

原文链接：http://ifeve.com/thread-safety/#more-3730