第二章 线程安全性

#### 一．共享与可变（状态）

共享

1. 概念

共享指变量可以被多个线程同时访问

1. 安全性

（1）非共享变量是线程安全的。

（2）同步的共享变量是线程安全的

可变

1. 概念

可变指变量的值在其生命周期内可以发生变化。

2. 安全性

（1）不可变变量是线程安全的。

（2）非共享变量是线程安全的。

（3）同步的共享变量是线程安全的。

Java的加锁方式

1. synchronized

2. volatile

3. 显示锁

4. 原子变量

良好的规范

1. 当设计线程安全的类时，良好的面向对象技术、不可修改性，以及明晰的不变性规范都能起到一定的帮助作用。

#### 二. 线程安全性

正确性

1. 概念

正确性是指，某个类的行为与其规范完全一致。

线程安全性

1. 概念

当多个线程访问某个类时，这个类始终都能表现出正确的行为，那么就称这个类是线程安全的。

有状态对象

1. 概念

有数据存储功能，有实例变量的对象。

无状态对象

1. 概念

一次操作，不保存数据，没有实例的对象。

2. 安全性

无状态对象一定是线程安全的。

#### 三. 原子性

竞态条件

1. 概念

某个计算的正确性取决于多个线程的交替执行时序时，那么就会发生竞态条件。

换句话说，就是正确的结果取决于运气。

2. 常见类型

先检查后执行：通过一次可能失效的观测结果来决定下一步的动作。

复合操作

1. 概念

包含多个原子操作的操作。

原子操作

1. 概念

对于访问同一个状态的所有操作（包括这个操作本身）来说，这个操作是一个以原子方式执行的操作。

2. 应该尽可能使用现有的线程安全类来管理类的状态。例如Atomic原子类。

#### 四. 加锁机制

1. 要保持状态的一致性，就需要在单个原子操作中更新所有相关的状态变量。即对多个原子操作加锁。

内置锁

1. Synchronized

Java的内置锁包括两部分，一个是锁的对象引用，一个是锁保护的代码块。

Synchronzied(lock){

}

其中lock就是内置锁，也叫监视器。线程进入同步代码块之前会首先获得锁，并且在退出的时候释放锁。

Synchronized在修饰方法时，内置锁就是调用该方法的对象，如果是静态方法，则内置锁锁住的是整个类，这跟以类.Class是同一个道理。

Java的内置锁是一种互斥锁，即同一时刻只有一个线程能够持有锁。

重入

1. 概念

如果一个线程试图获得一个已经由它自己持有的锁，那么这个请求就会成功。重入意味着获取锁的粒度是线程而不是调用。

2. 实现

为每个锁关联一个获取计数值和一个持有者线程。当计数值为0时代表该锁没有被任何线程所持有。当线程请求获取一个未被持有的锁时，JVM将记下线程持有者并且把计数值置为1。如果这个线程再次获取这个锁，那么计数值将+1递增。当线程退出代码块，计数值会递减。当计数值为0时，这个线程将被释放。

3. 优点

重入进一步提升了加锁行为的封装性，因此简化了面向对象并发代码的开发。

用锁来保护状态

1. 概念

对于一个可以被多个线程同时访问的可变状态变量，在访问它时都需要持有一个锁，这时我们成状态变量是由这个锁保护的。

当获取一个与对象关联的锁时并不能阻止其它线程访问该对象，只能组织其它线程获得锁。之所以每个对象都有一个内置锁，只是为了避免显示地创建锁对象，你需要自行构建加锁协议或者同步策略。

2. 每个共享的和可变的变量都应该只由一个锁来保护，从而使维护人员知道是哪一个锁。

3. 对于每个包含多个变量的不变性条件，其中涉及的所有变量都需要由同一个锁来保护。

#### 五. 活跃性与性能

1. 缩小同步代码块

应该尽量将不影响共享状态且执行时间较长的操作从同步代码块中分离出去，从而在这些操作的执行过程中，其它线程可以访问共享状态。

2. 在简单性与性能之间存在着相互制约因素。当实现某个同步策略时，一定不要盲目地为了性能而牺牲简单性（这可能会破坏安全性）。

3. 当执行时间较长的计算或者可能无法快速完成的操作时（例如，网络I/O或控制台I/O），一定不要持有锁。