第四章 对象的组合

#### 一．设计线程安全的类

线程安全类

1. 设计线程安全类的过程中三个基本要素

（1） 找出构成状态的所有对象

（2） 找出约束状态变量的不变性条件

（3） 建立对象状态的并发访问管理策略

2. 同步策略

收集同步需求

1. 不可变条件

2. 后验条件

依赖状态的操作

1. 概念

包含基于状态的先验条件的操作

2. 常用

通过现有库中的类（简单的方式）

（1） BlockingQueue

（2） Semaphore

状态的所有权

1. 声明一个HashMap相当于创建了多个对象：HashMap对象、HashMap包含的多个对象、以及在Map.Entry中可能包含的内部对象。

2. 垃圾收集器在一定程度上帮我们简化了对象所有权的问题。

#### 二. 实例封闭

1. 将数据封闭在对象内部，可以将数据的访问限制在对象的方法上，从而更容易确保线程在访问数据时总能持有正确的锁。

2. 例如

Collections.synchronizedList方法能将非线程安全的ArrayList等转化为线程安全的。

实现：

装饰器模式。Collections.synchronizedList能将容器类封装在一个同步的包装器对象中，包装器能将接口中的每个方法都实现为同步方法。

3. 优点

封闭机制更容易构造线程安全的类，因为当封闭类的状态时，在分析类的线程安全性时就无须检查整个程序。

Java监视器模式

1. 概念

遵循Java监视器模式的对象会把所有可变状态都封装起来，并由对象自己的内置锁来保护。

2. 私有锁的优点

私有锁可以封装对象，使客户代码无法得到锁，但客户代码可以通过公有的方式来获得锁，以便参与到它的同步策略中。

3.

#### 三. 线程安全性的委托

1. 可以将线程安全性委托给单个线程安全的变量，也可以将线程安全性委托给多个线程安全的变量。

2. 发布底层的状态变量

如果一个状态变量是线程安全的，并且没有任何不变性条件来约束它的值，在变量的操作上也不存在任何不允许的状态转换，那么就可以安全的发布这个变量。

#### 四. 在现有的线程安全类中添加功能

1. 优先选择java类库中现有的类。

2. 如果现有的类不能满足你，则使用现有的类的功能在不破坏安全性的条件下来实现新的功能。

3. 实现方式

（1）修改现有的类（这通常无法做到）

（2）扩展类（继承）

比（1）更脆弱，如果底层的类改变了它的同步策略，那么子类会被破坏。因此父类最好定义它的同步策略，如Vector。

会破坏封装性。

（3）客户端加锁（更脆弱）

会破坏封装性。

（4）组合（更好）

客户端加锁机制

1. List加锁的那些坑。

以下代码非线程安全。因为synchronized和list用的锁不是一个锁。其它方法调用list的put方法可能会对putIfAbsent方法造成影响。

class BadListHelper <E> {

public List<E> list = Collections.synchronizedList(new ArrayList<E>());

public synchronized boolean putIfAbsent(E x) {

boolean absent = !list.contains(x);

if (absent)

list.add(x);

return absent;

}

}

组合

1. 通过实现现有的List接口，来将list对象的操作委托给底层的list实例来实现list的操作。

#### 五. 将同步策略文档化

1. 在文档中说明客户端代码需要了解的线程安全性保证，以及代码维护人员需要了解的同步策略。

2. SimpleDateFormate不是线程安全的。

3. JDBC Connection不是线程安全的。