# Thinking in java

--三人务于精熟，而亮独观其大略

# 本书优点

Thinking in Java为啥为这么成功？

0.代码非常丰富

基本上每个主题都有代码

1.能够用最短的例子，把某个Java功能说的清楚；

2.每个主题之间互不关联。不存在先要学习示例代码1，才能学习示例代码2的问题。

# Annotation

## 背景

我们在开发影像新的后台服务的时候，开始大量接触到了annotation。在实践过程中，annotation确实对于提升代码的可读性，有非常大的帮助。在SpringBoot体系中，注解也是非常重要的一个组成部分。

因此，我们在这里专题学习注解。主要目标是：1.了解注解的实现原理；2.能够自己实现注解；3.了解SpringBoot体系中核心注解的实现原理。

## 总结

# Concurrency

## Basic Threading

1.

CallableDemo.java

说明如何从线程获取返回值

2.

SleepingTask.java

说明线程休眠一段时间

3.

SimplePriorities.java

说明如何设置线程优先级(貌似设置一下优先级也没啥用)

4.

ResponsiveUI.java

说明线程在交互式UI中的应用

## **Sharing resources**

1.

通过一个偶数生成器，来判断判断线程是否安全

EvenChecker.java

IntGenerator.java

线程不安全的偶数生成器

EvenGenerator.java

线程安全的偶数生成器

MutexEvenGenerator.java

SynchronizedEvenGenerator.java

2.

序列号生成器(通过优化，保证序列号生成器线程安全)

SerialNumberChecker.java

SerialNumberGenerator

AtomicIntegerTest.java

3.

CriticalSection的概念

CriticalSection.java

这个代码要好好总结一下，尤其是PairChecker，通过异步线程检测数据是否一致，后续可以参考

SyncObject.java

4.

说明Thread local storage的应用，说白了就是把共享资源在各个线程中各存放一份，以免竞争

ThreadLocalVariableHolder.java

5.

Sharing resource总结：

这个小结主要介绍了集中方案，用于避免线程不安全导致数据异常：

1.synchronize关键字

2.try lock

3.Java自带的各个Atomic类

4.critical section(缩小synchronize范围)

5.Thread local

## **Terminating tasks**

1.

说明各个情况下的线程能否被中断

Interrupting.java

2.

如何关闭子线程中的资源

CloseResource.java

3.

NIO的方式，只要关闭子线程，就能够自动将子线程中的NIO 连接资源关闭

NIOInterruption.java

4.

MultiLock.java

5.

这个代码为了说明*Lock.lockInterruptibly()*的用法

Interrupting2.java

6.

这个代码主要是为了说明，我们中断一个子线程的时候，需要特别注意关闭子线程中的资源

InterruptingIdiom.java

Terminating tasks 《总结》

1、2、3是一个系列，层层递进，说明了关闭子线程的时候，如何释放将子线程中各种类型的资源。

## **Cooperating between tasks**

1.这个代码主要是借助汽车上蜡、擦拭这两个独立的线程，说明线程间如何通信

关键字是*notifyAll()*

WaxOMatic.java

2.

NotifyVsNotifyAll.java

3.

主要是通过生产者、消费者模型，说明进程间是如何通信的，关键字是*notifyAll()*

Restaurant.java

WaxOMatic2.java

TestBlockingQueues.java

PipedIO.java

## **Deadlock**

这个package下的代码，说明了死锁的的场景(DeadlockingDiningPhilosopher.java)，

以及如何解决死锁(FixedDiningPhilosopher.java)。

代码示例就是非常典型的哲学家模型，思路非常清晰

// 哲学家类

Philosopher.java

// 会造成死锁的场景

DeadlockingDiningPhilosopher.java

// 修复了死锁的场景

FixedDiningPhilosopher.java

## **New Lib**

// 这个文件主要是为了说明*CountDownLatch*这个用于并发的类库

CountDownLatchDemo.java

// 赛马的场景，主要是为了说明 *CyclicBarrier* 的用法

HorseRace.java

DelayQueueDemo.java

PriorityBlockingQueueDemo.java

GreenhouseScheduler.java

// 下面三个文件，主要是通过实现一个object pool，验证Semaphore的功能

Pool.java

Fat.java

SemaphoreDemo.java

ExchangerDemo.java

## **Simulation**

BankTellerSimulation.java

RestaurantWithQueues.java

CarBuilder.java

## **Performance Tuning**

SimpleMicroBenchmark.java

SynchronizationComparisons.java

Tester.java

MapComparisons.java

FastSimulation.java

ReaderWriterList.java

ActiveObjectDemo.java

## **总结**

Java的并发还是有很多内容可以讲的。各种并发模型