jdk源码&多线程&高并发-【阶段2、深入多线程设计模式】

讲师简介

smart哥，互联网悍将，历经从传统软件公司到大型互联网公司的洗礼，入行即在中兴通讯等大型通信公司担任项目leader，后随着互联网的崛起，先后在美团支付等大型互联网公司担任架构师，公派旅美期间曾与并发包大神Doug Lea探讨java多线程等最底层的核心技术。对互联网架构底层技术有相当的研究和独特的见解，在多个领域有着丰富的实战经验。

一、多线程设计模式概论

不是gof的23种设计模式（针对业务的非线程代码）

shi山

线程代码的设计模式（12种），是**doug lea** 和 josh bloch jsr166的规范，jsr133

第三阶段 学习juc的一个理论基础。老外是先有一套方法论，然后在这一套方法论的基础之上进行落地

E.F.CODD 数据库的范式

doug lea的主页：

http://gee.cs.oswego.edu/

根据doug lea的分类，多线程程序的评价标准如下：

1、安全性

不损坏对象

通常是指对象的属性出现了意想不到的情况，例如 i--，在没有保护的情况之下就有可能出现问题

安全性只是状态变化了

安全性和兼容性

arraylist如何安全？？

2、生存性

必要的处理能够被执行，出现假死，最典型的就是死锁

状态没有变，但是程序停了，或者即使程序没有死但是也是不动，卡死了（死锁）

生存性和安全性是相互制约的，一味的强调安全，那么就有可能出现生存性问题

3、可重复性

juc里面有大量的可重复性利用的类

4、性能

doug lea总结了影响性能的关键因素，如下：

1）吞吐量

2）响应性

3）容量

第1,2两点是必要的，第3,4两点是锦上添花

二、Single Threaded Execution模式

所谓“Single Threaded Execution”，即“以一个线程执行”，该模式用于设置限制，以确保同一时间内只让一个线程执行处理。

1、案例：多个人过安检门

结论：只能一个一个的通过，如果大家一起通过的话，那么就有可能出现数据不一致的问题

一个线程执行pass方法会经历1次写操作，2次读操作

2、1个线程执行临界区，能不能最多同时N个线程来执行临界区？

Semaphore 信号量

需求：一个饭店同时最多允许8个人用餐，如何实现？？

自定义一个semaphore

总结：**基于aqs。只要是基于aqs的实现理论上都是可以使用synchronized替代的**

三、Immutable模式

Immutable模式中存在着确保实例状态不发生改变的类（immutable类）。在访问这些实例时并不需要执行耗时的互斥处理，因此若能巧妙利用该模式，定能提高程序性能。

1、案例：和single threaded execution比较

没有写操作（类的属性不会发生变化），只有读操作

属性用final和private修饰 ，更加说明类中不会提供setter方法

总结：不可变模式就是属性没有写，只有读，所有不需要synchronized关键字修饰临界区

2、分析：jdk中哪些类是使用这种模式呢？

有一部分是immutable，还有一部分是mutable

String类和StringBuffer类

String类是immutable。StringBuffer类是mutable

标准类库中的immutable模式

ArrayList(可变的)

CopyOnWriteArrayList（可变部分给拷贝出来）

提问：immutable模式是否能够提升性能？？

可以提升性能，但是测试未必能测的出来，因为synchronized做了大量优化，要具备一定的条件才能测出来

综合案例：外部传入的可变对象如何保证安全？

见视频

四、Guarded Suspension模式

又叫“保护性暂挂”模式，如果执行现在的处理会造成问题，就让执行处理的线程等待，通过让线程等待来保证实例的安全性。

1、提问：保护的是谁？暂时挂起的又是谁？

在tomcat中，client发送request请求，server端接受到请求，然后做解析，缓存起来，然后server端线程会获取到request请求，然后对请求做进一步处理（交给springmvc去做了）

服务器的前端（依然在服务器这一端），专指服务接受请求

服务器后端，请求转发给框架

4个实体：

1）客户端线程，发送request到队列

2）request

3）queue

4）服务端线程，负责到队列里面获取request做进一步处理

保护的是queue，暂时挂起的是消费者线程，说白了就是队列里面如果没有东西，那么消费者线程必须等待

核心代码：

while（队列为空）{

```
    synchronized（queue）{
    wait（）；//挂起消费者线程
    }
}
```

这里其实有多个设计模式：

1）有synchronized，那么说明用了单线程执行的设计模式

2）Immutable模式（Request）

3）Guarded Suspension

五、Balking模式

如果现在不合适执行这个操作，或者没必要执行这个操作，就停止处理，直接返回。

有多个（起码2个）线程，某一个线程正在准备做某件事的时候，然后突然呢，另外一个线程抢先做了，那么这个线程就不执行这个动作了。

1、图解

http://huatu.xuanheng05.cn/lct/#R7766daf15a1cda0603488a0842a528cd

2、比较Balking设计模式和GuardedSuspension设计模式

角色分析：

相同点

1）受保护对象（公共资源）

Balking模式：电子屏

GS模式：queue

总结：只有公共资源里才会有synchronized关键字，而在线程类里面是不会出现synchronized关键字的

2）受保护的方法（公共资源里的被多线程访问的方法）

3）对象的状态（公共资源的状态）

多线程的执行一定是遵循共有资源的状态的变化而采取不同的执行策略

不同点

4）深层次的比较

细节：Balking模式是不需要等待和唤醒的，而是判定状态不对，就立马放回

GS模式是需要等待和唤醒操作的。

3、Balking模式的使用场景

1）不需要再执行了（不做了）。提高了程序的性能

2）不需要等待守护条件成立时，返回。那这种模式提高了程序的响应性

3）守护条件仅仅在第一次成立时（服务器初始化）

演示见视频

六、Producer-Consumer模式

Producer：生成数据的线程

Consumer：使用数据的线程

Producer-Consumer模式在生产者和消费者之间加入了一个“桥梁角色”，该桥梁用于消除线程间处理速度的差异。

队列就是“桥梁角色”，起到缓冲的作用

1、案例

角色分析：

生产者

消费者

中间桥梁（MyQueue）

测试类

该模式中其实也包含保护性暂挂模式

七、Read-Write Lock模式

当线程读取实例的状态时，实例的状态不会变化。实例的状态仅在线程执行写入操作时才会发生变化。

1、案例

1）角色分析：

读线程（多个）

写线程（多个）

共享数据（公共资源）

数组

自定义读写锁

测试类

2）问题及优化

见视频

2、分析模式

包含的设计模式：

1）单线程执行模式

2）immutable模式

3）gs模式

守护条件在readLock和writeLock里面

3、使用场景

适合读取操作比较繁忙的时候，读的频率高于写的频率（读线程比写线程多），会提供性能

物理锁：synchronized

逻辑锁：就是我们自己借助于synchronized实现的readLock和writeLock

所谓的比较就是比较读写锁和synchronized直接的性能差别

性能提升演示见视频

4、答疑

1）不加锁会怎样？

信息会错乱

2）跟juc中的读写锁有什么区别？

juc读写锁是基于aqs实现的吧，

3）数据库里面的共享锁和排它锁跟读写锁有什么关系？

mysql底层的共享锁和排它锁跟读写锁原理底层是一样的

hashmap模拟数据库

八、Thread-Per-Message模式

为每个命令或请求新分配一个线程，由这个线程来执行处理。

tomcat早期的BIO模型（bio，apr，**nio**，aio）

1、案例

总结：

1）提高了系统响应，缩短了时间

2）对操作顺序没有特别要求

3）没有返回值

4）适用于服务器

2、实现一个tomcat的BIO通道的雏形

需求：刷新浏览器，然后浏览器上不断打印"hello,smart哥！"打印10次，每次间隔1秒

请求： http://localhost:8888/hello

响应：hello，smart哥！

九、Worker Thread模式

在Worker Thread模式中，工人线程（work thread）会逐个取回工作并进行处理。当所有工作全部完成后，工人线程会等待新的工作到来。

这个模式又叫线程池模式

2个容器：

队列：装任务

hashset：装线程

1、案例

需求：

一个通用的地产商-建筑承建商系统

地产商（万科，soho，恒大等）

工程承包给建筑商来做

某一个建筑公司下面有10个建筑队

建筑1队

建筑2队

建筑3队

。。。。

2、代码实现

角色分析：

1）地产商线程

2）合同

3）合同容器

4）施工队线程

5）施工队线程容器

十、Future模式

Future的意思是未来的。假设有一个方法需要花费很长时间才能获取运行结果，那么，与其一直等待结果，不如先拿一张“提货单”。这里的“提货单”就称为Future角色，是“未来”可以转化为实物的凭证。

跟Thread-Per-Message的差异：

Thread-Per-Message没有返回值

而Future模式有返回值

1、案例

分析角色：

client线程

server

创建产品的线程-new thread（这个线程非常重要，就是这个线程把请求一分为二，从而达到异步的目的）

futureProduct（提货单）

realProduct（真实数据）

测试类

2、代码实现

见视频

用了哪些模式？

Thread-per-message模式

balking模式

gs模式

3、用juc中FutureTask替换

见视频

4、juc中的FutureTask源码解析

见视频

5、实际案例：爬虫

www.baidu.com

需求：**采集数据，然后保存数据**（本地文件），分析数据

改造：在采集和保存这两个动作直接用一个新的thread来做？？

十一、Two-Phase Termination模式

开始 -> 操作中 -> 终止处理中 -> 终止

先从“操作中”状态变为“终止处理中”的状态，然后再真正地终止线程。

该模式将停止线程这个动作拆解为**准备阶段和执行阶段**，从而优雅，安全的停止线程

1、案例

需求：一个学生写暑假作业，然后时间到了，妈妈叫他睡觉，那么他先准备（收拾课本，洗漱），后执行（上床睡觉）

问题：sleep时间很长怎么办？？终止命令已经发出，但是对sleep，wait等挂起操作不起作用

2、游戏服务器停止案例

1）图解案例

2）实现两个子系统（公告和活动）

服务器统一发送终止指令，各子系统获取指令后开始分阶段终止线程

十二、Thread-Specific Storage模式

即使只有**一个入口**，也会在内部为**每个线程分配特有的**存储空间的模式。

说白了就是一个线程一份，各玩各的，有2种：

1）局部变量（定义在方法里的，或者是方法参数）

天然的一个线程一份

2）实例变量（定义在类里面的）共有资源

new instance

threadlocal

1、图解

见视频

2、案例

前后端分离情况下的用户数据共享

十三、Active Object模式

在Active Object模式中出场的主动对象可不仅仅“有自己特有的线程”。它同时还具有可以从外部接收和处理异步消息并根据需要返回处理结果的特征。

Active Object模式中的主动对象（集合）会通过自己特有的线程在合适的时机处理从外部接收到的异步消息。

Active Object模式综合来Producer-Consumer模式、Thread-Per Message模式、Future模式等各种模式，有时也被称为Actor模式。reactor

rmi，rpc

目的：实现方法调用和任务执行进行分离

1、示例及分析

需求：打印机发起请求（打印，复印）

分析角色：

1）委托者（客户端线程）

发起请求

打印线程

复印线程

2）主动对象（active object）

给委托者提供接口

3）proxy（代理）

负责把请求转化为对象，并传递给执行者线程（不是真正去处理）

4）执行者（调度者）线程

又是消费者线程，具体负责入队和出队（出队后调用服务，真正执行）

5）MethodRequest

请求转化为对应的MethodRequest对象，该对象会被塞进队列，**且定义对应的服务处理类及返回值，入参**

6）服务类（PrinterService）

具体处理请求的服务对象

7）队列

存放MethodRequest

8）返回值

futureResult，realResult

2、分析

1）实现了调用和执行的分离（实现了异步）

2）遵循设计模式里的开闭原则

proxy（active object）是否安全？？

service是否安全？？

3、用juc中的类替代

线程池替代（printerService，queue，executor线程）

methodrequest去掉，直接实现Runnable，Callable

futureResutl和realresult用future替代

注意：本阶段所有模式图解如下

http://huatu.xuanheng05.cn/lct/#R7766daf15a1cda0603488a0842a528cd