# 1、数据架构853769620

## 1.1 数据库ER模型

## 1.2 数据库逻辑模型

## 1.3 数据库物理模型

# 2、接口

## 2.1 采购入库

InventoryFacadeService

Boolean informPurchaseInputFinished(PurchaseInputOrderDTO

purchaseInputOrderDTO)

## 2.2 提交订单

InventoryFacadeService

Boolean informSubmitOrderEvent(OrderDTO orderDTO)

## 2.3 支付订单

InventoryFacadeService

Boolean informPayOrderEvent(OrderDTO orderDTO)

## 2.4 取消订单

InventoryFacadeService

Boolean informCancelOrderEvent(OrderDTO orderDTO)

## 2.5 退货入库

InventoryFacadeService

Boolean informReturnGoodsInputFinished(ReturnGoodsInputOrderDTO

returnGoodsInputOrderDTO)

## 2.6 查询商品库存

（1） 商品详情页中需要显示商品库存，选择好sku的销售属性之后，ajax异步来加载controller中的接口

（2） 购物车中，在显示购物车数据的时候，购物车的后台，会去调用商品中心获取sku数据，也会调用库存中心来查找这个sku的库存数据

InventoryFacadeService

Long getSaleStockQuantity(Long goodsSkuId)

# 3、开发架构

## 3.1 实现类图

### 3.1.1 命令模式+模板方法模式+工厂方法模式

命令模式，工厂模式，模板方法模式

我们可以设计一个Command，有一个抽象基类，这个AbstractCommand里面，定义好了更新库存的一套步骤：

PurchaseInputOrderDTO purchaseInputOrderDTO

updateStockQuantity() {

updateSaleStockQuantity()

updateLockedStockQuantity()

updateSaledStockQuantity()

updateStockStatus()

updateGmtModified();

}

updateSaleStockQuantity()

updateLockedStockQuantity()

updateSaledStockQuantity()

updateStockStatus()

private void updateGmtModified() {

}

子command去实现的，相当于是将模板方法模式和命令模式给结合起来了

工厂方法模式，有一个工厂接口，但是针对不同的command有不同的子工厂，这个子工厂会做一些通用的一些事情，然后创建一个command出来

我们先写设计模式的代码，结合咱们的库存更新的业务来做，然后命令模式+模板方法模式+工厂方法模式都给做好了之后，然后我给大家去讲解一下，为什么在这个场景里，要用这些设计模式

1、模板方法设计模式

设计一系列的command

如果不用模板方法设计模式，大家想象一下这个代码会写成什么样子？其实更新库存，都是按照一定的规则，固定的流程和步骤去走的，先更新什么库存，然后更新什么库存，接着检查库存状态，设置修改时间，最后执行更新。

但是如果每个接口里面你就放任自己去随便写的话，可能逻辑是对的，但是每个接口里面的具体的执行步骤顺序可能就不一样，使用的方法命名可能都不一样

本来各种库存更新的过程，应该看起来是类似的，是一样的，结果的话呢，因为人为写代码的问题，导致写出来的代码看起来很不一样，对于后续代码的维护，扩展，修改，都没有什么好处

更不用说，不同的库存更新的过程，都是有一些是一样的通用的步骤

所以说，我们采取了模板方法设计模式，将一些通用的步骤抽取的到抽象基类，另外一个基于模板的模式限定了每个库存更新的过程都是一样，按照一样的步骤和顺序去走，很清晰，后面呢要修改库存更新的逻辑，或者是新增一种库存更新的逻辑，都是按照一样的步骤和顺序去走

2、工厂方法设计模式

工厂方法设计模式，就是将工厂模式与模板方法模式，结合起来了

就是说，可能你需要的不是一个工厂，需要不同的工厂，不同的工厂创建不同的产品，但是呢，这些工厂之间有一些通用的逻辑，可以抽取到父工厂里面去，子工厂就专注干自己的事情就可以了

3、命令模式

我们将不同类型的库存更新的操作，封装了不同的库存更新命令

有同学之前跟我说，命令模式不是跟策略模式是一样的，代码实现上可能类似，策略模式是用来替换到大段大段的if else if else if else

我将一些操作封装成这个命令，然后后续可以对这些命令做一些高级的操作，比如说将命令发送到异步队列里面去，这个是命令模式最最常用的一个应用场景

库存更新的场景，我们后面可以去看一下，是否可以做成基于队列异步化的这么一种模式，命令模式还有一个很经典的场景，就是做这个命令撤销。

如果我们在执行这个命令的过程中，发现命令中的某个步骤失败了，我们可以在command里面实现一套cancel的逻辑，撤销这个命令之前做的所有的操作，对已经完成的操作执行反步骤

我们之所以在这里使用命令模式，虽然暂时看起来只是简单的去执行了这个命令中的逻辑，但是我们是为了以后打下一个扩展的基础，如果以后要对库存更新的操作，比如说实现异步化，或则是库存更新操作的撤销，等等一些高级的操作，就可以基于围绕command命令的概念去做

### 3.1.2 中介者模式+观察者模式+备忘录模式

（1）订单中心对库存中心的库存操作调用，以及对调度中心的调度发货的调用，都通过内存队列做成异步的方式，来让订单中心跟库存中心以及调度中心来解耦，异步，可以让订单中心的操作非常的快速。中介者模式的思想

中介者模式的本质，就是解耦，将一个组件对另外一个组件的调用解耦，不要让一个组件直接调用另外一个组件，而是让一个组件调用中介者，然后中介者去调用另外一个组件

但是实际使用场景中，几乎我是从来没有严格按照中介者模式的代码去写，异步解耦，我们一个组件不要直接调用另外一个组件，而是将消息写入一个队列，然后另外一个组件去队列里面消费消息然后自行处理

队列，又有两种大的场景：jvm内存队列；分布式队列（RabbitMQ，RocketMQ，Kafka）

如果是一个系统内部的异步解耦的行为，在系统内部用jvm内存队列即可；如果是一个系统跟另外一个系统之间的异步解耦的通信，一般会用外部的MQ。这两种场景都有，这两种方式都有

电商系统V1.0版本，本来就是一个单块应用，一个jvm进程，那肯定要做异步，就是用内存队列就可以了。。等到后面我们将这个电商系统升级到分布式系统架构之后，我们可以采取外部的MQ来重构整个系统

1. 观察者模式，订单中心执行了异步的操作之后，他要去对其他的中心注册一个观察者，等待其他中心完成操作之后，将结果通知给订单中心

观察者模式非常经典的在分布式系统里面的一个应用，就是做分布式协调，异步调用之后，系统A发送一个消息到了MQ里去，系统B从MQ里面消费了数据来处理，系统A想要了解系统B是否成功处理完了这个数据，该怎么办呢？

消息B处理完之后回过来去通知系统A，如果系统A专门为这个事儿开个口子，开个接口，很麻烦。系统A发送完了异步消息之后，直接监听一个东西，系统B处理完之后，就触发一下系统A的监听器，通知系统A处理的结果。

如果是跟纯jvm内存里面的队列来结合，那就是用jdk监听器api来实现观察者模式即可；如果是跟外部的分布式队列来结合，那肯定是用zookeeper了。

用的是jvm内存队列，所以说我们这个观察者模式的体现，就是用的jdk观察者api来做就可以了。

（3）备忘录模式，一旦订单中心发现说内存队列满了，写不进去了，那么此时就需要通过备忘录将数据离线存储到磁盘上去。等后面内存队列恢复了，就可以通过备忘录将数据给恢复回来。

很简单，大家不要搞的太过于玄乎，memento，吓尿了。如果你有一份数据，搞着搞着，弄到一半儿，系统用这份数据用到一半，发现说需要把这份数据给它暂存一下，暂存到redis里面？jvm ehcache缓存？磁盘？es？hbase？mysql？cassandra？

等到了后面，你这份数据系统跑着跑着跑着，突然又要使用了，这个时候再从之前暂存的那个地方给他读出来，不就可以了吗？

备忘录模式，备忘录，记录个东西的，将某个数据存储在备忘录里面，过一段时间再从备忘录里面给读取出来，备忘录就是一个存储，redis，es，hbase，都可能。

在用MQ的时候，备忘录模式，是一个非常经典的使用场景，为什么呢？你如果是写内存队列，那肯定是要用阻塞队列的，因为避免说你的数据量过大，把非阻塞的队列可能会写爆，jvm oom；写分布式队列，rabbitmq，rabbitmq如果挂了怎么办？一年就挂一次，但是一旦挂掉，就会导致你们公司的业务瘫痪。

写内存队列的时候，如果内存队列满了，阻塞了，为了避免整个系统被内存队列卡住，此时你需要将数据暂存到比如磁盘上去，事后内存队列不再阻塞之后，再将数据从磁盘上读取出来，再往队列里面去写；如果是写分布式队列，在队列挂掉之后，需要将数据离线存储起来，事后当分布式队列恢复之后，再次从离线存储中读取数据写入队列

非常经典的一个备忘录模式的使用场景

每次放队列里面写数据的时候先判断一下，如果队列的大小已经满了，就不要往里面去写了

此时将消息给写入一个mysql的表中，mysql的表是需要有一个字段保持顺序的，其实按照自增长的主键就最ok

然后需要更新内存中的状态标识位，表明说现在已经触发了离线存储

然后之后每次消息过来，都会在将消息写入内存队列之前，先判断一下内存中的状态标志位，看看是否触发了离线存储，如果触发了离线存储，需要看一下，内存队列当前是否还阻塞，如果已经不阻塞了

此时一旦触发了离线存储，之后所有的消息都会直接写入mysql，此时就相当于是通过一个备忘录组件，往备忘录里去存储数据，备忘录底层是基于mysql来实现的

但是一旦某次消息发现说内存队列已经不阻塞了，没有积压了，此时就会启动一个后台线程，由那个后台线程去批量的从mysql中查询数据

那么可以通过备忘录来获取数据，底层将mysql中的数据，按照顺序一批一批的查询出来，查询出来一批，就往内存队列里面去写入，写入成功之后，就将mysql中的数据给批量的删除掉

然后循环往复，循环执行上面的操作，直到这个后台线程将mysql中的数据全部都写入了内存队列，都ok了，mysql里一条数据都没有了

此时就可以由后台线程更新内存里的标志位，说不需要再出发离线存储了

再之后过来的请求，就不会写入mysql了，而是正常写入内存队列中

这是一个很经典的方案，因为很多系统里都会使用内存队列的这个技术，做异步化，缓冲化，但是只要用了内存队列，就要考虑内存队列会满：消费者程序bug了，不再消费了；或者是消费者程序bug了，消费速度突然变得很慢

就一定要考虑内存队列满了，数据要写入外部的存储，这是一个我至少2个以上的重要的线上系统，用到内存队列的时候都结合具体的业务设计了数据离线存储和恢复的方案，而且我们在生产环境中真的遇到过内存队列满了，整个系统卡死，case

但是不同的系统具体的业务场景不一样，技术栈也不一样，我采取了一样的而思想，但是具体的实现肯定跟我的线上系统不一样。我这里呢，就采取适合我们课程里目前这个系统的情况，我来做一套方案出来

完美的可以去使用备忘录模式

## 3.2 包设计

# 4、运行流程

## 4.1 库存管理

### 4.1.1 采购入库

库存中心，是不care仓库和货位层面的东西，一旦有采购入库之后

将某个商品的销售库存累加上对应的值

如果这个商品之前的所属仓库是无，同时销售库存是0，库存状态是无库存，那么此时加入新库存之后，除了更新销售库存，同时会更新一下商品的所属仓库，还会去更新库存状态为有库存

### 4.1.2 提交订单

用户提交订单之后，订单中心会来通知库存中心，将商品的可销售库存扣减1，同时将商品的锁定库存加1。接着库存中心，需要去通知调度中心，去更新这个库存。

### 4.1.3 支付订单

用户支付了订单之后，订单中心会来通知库存中心，将商品的锁定库存减1，将商品的已销售库存加1。接着库存中心需要通知这个调度中心，将调度中心的库存也进行更新操作。

### 4.1.4 取消订单

将商品的锁定库存减1，将商品的可销售库存加1。接着通知调度中心更新库存。

### 4.1.5 退货入库

商品的可销售库存加1，已销售库存减1

你之前如果把这个商品销售出去以后，是减去销售库存，增加已销售库存；现在退货入库之后，是不是就要增加销售库存，但是要减去已销售库存

# 5、测试用例设计

# 6、日志设计