Redis事务

解决多线程**超发**的情况。

**Redis的事务指的是提供一种将多个命令打包，一次性按顺序地执行**。

Redis的事务可以保证只有在执行完事务中的所有命令后，才会继续处理客户端的其他命令。也就是说只有一个用户可以操作事务当中的数据

**Redis中的事务从开始到结束要经历3个阶段**

开启事务-> 命令入列->执行事务/放弃事务

**开启事务命令**

Redis事务四大指令：MULTI、EXEC、DISCARD、WATCH

\*MULTI 开启一个事务；

\*EXEC 执行一个事务；

\*DISCARD 取消一个事务；

\*WATCH 用于客户端并发情况下，为事务提供一个锁（CAS，Check And Set）可以用watch命令来监控一个或多个变量如果在执行事务之前，某个监控被修改了，那么整个事务就会终止执行（watch必须写在事务的前面，不能写在事务当中）

1.Redis数据初始化

既然Redis是缓存，就要考虑到什么时机把初始数据存入到数据库。

可以在项目启动时初始化，在标记为@PostConstruct的方法把输入存入Redis

2.执行事务

RedisTemplate.execute( )是执行器方法，可以执行一系列的操作。使用Redis事务时写法固定：

redisTemplate.execute(**new** SessionCallback<List<Object>>() {

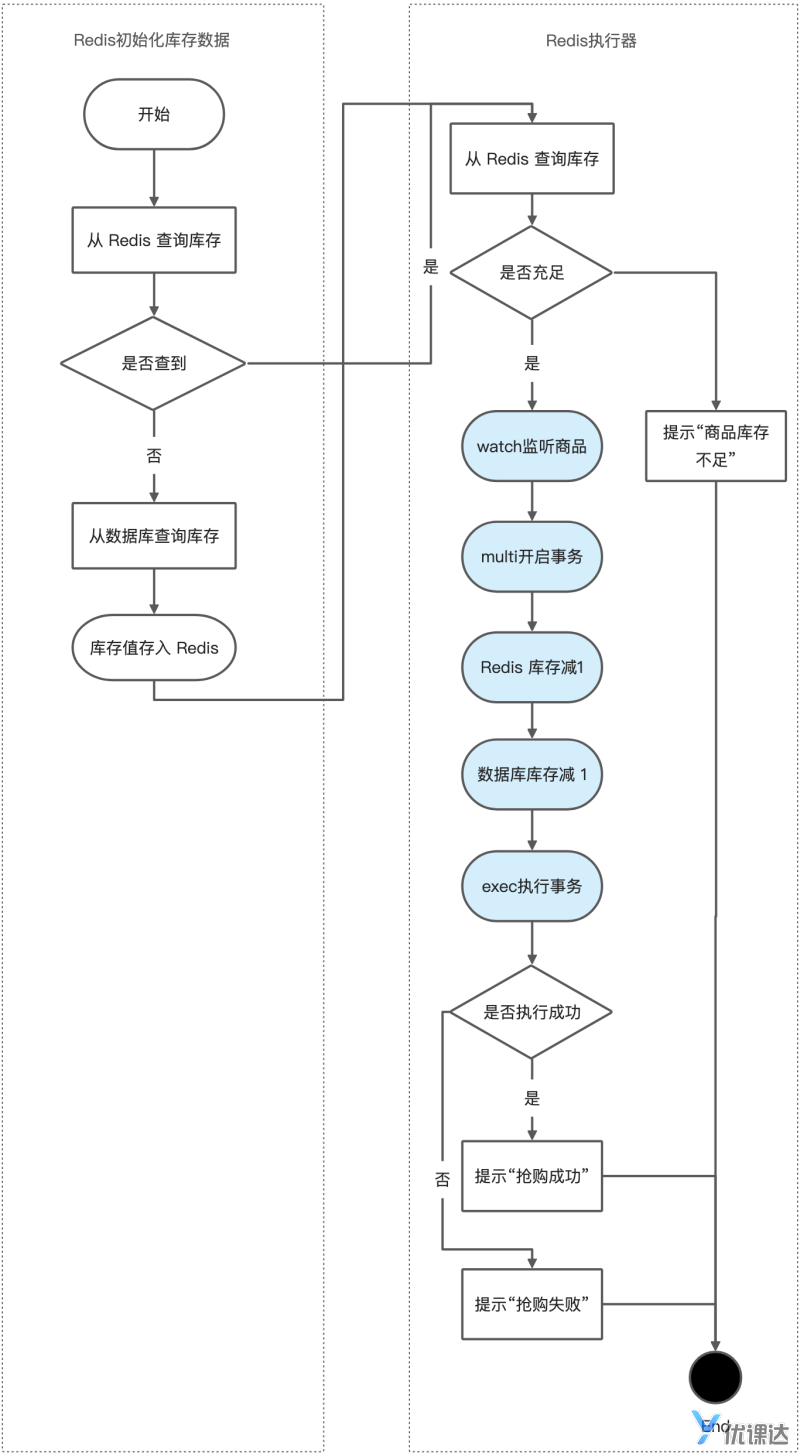
**@Override**

**public** List<Object> **execute**(RedisOperations operations) **throws** DataAccessException {

}

});

用WATCH来处理抢购的并发的情况



**public** Result<Boolean> **snappedUp**(**@RequestParam("id")** Long id) {

*//初始化返回数据*

Result result = **new** Result();

result.data(**true**);

result.setSuccess(**true**);

*//实现购买代码*

*//先去redis查询一下*

Object value = redisTemplate.opsForValue().get(id);

**int** stock = 0;

*//如果redis没有则去数据库查询*

**if** (value == **null**) {

*//去数据库查询该商品的信息*

ProductDO product = productDAO.selectById(id);

*//将信息缓存到redis里边*

stock = product.getStock();

redisTemplate.opsForValue().set(product.getId(), stock);

} **else** {

stock = (**int**)value;

}

redisTemplate.execute(**new** SessionCallback<List<Object>>() {

**@Override**

**public** List<Object> **execute**(RedisOperations operations) **throws** DataAccessException {

Integer stock = (Integer)operations.opsForValue().get(id);

*//判断该商品的库存是否大于1*

**if** (Integer.valueOf(stock) >= 1) {

*//监听商品的名字，redis里边的key*

operations.watch(id);

*//开启事务*

operations.multi();

*//将该商品的库存自减1*

operations.opsForValue().set(id, stock - 1);

*//修改mysql数据库库存数量*

ProductDO productDO = **new** ProductDO();

productDO.setId(id);

productDO.setStock(stock - 1);

productDAO.updateStock(productDO);

*// 执行事务*

List exec = operations.exec();

**if** (exec.size() > 0) {

*// TODO:可以有其它业务逻辑，例如插入订单等，视具体需求而定*

result.setMessage("抢购成功");

result.setData(**true**);

} **else** {

result.setMessage("抢购失败");

result.setData(**false**);

}

**return** exec;

} **else** {

result.setMessage("商品库存不足");

result.setData(**false**);

**return** **null**;

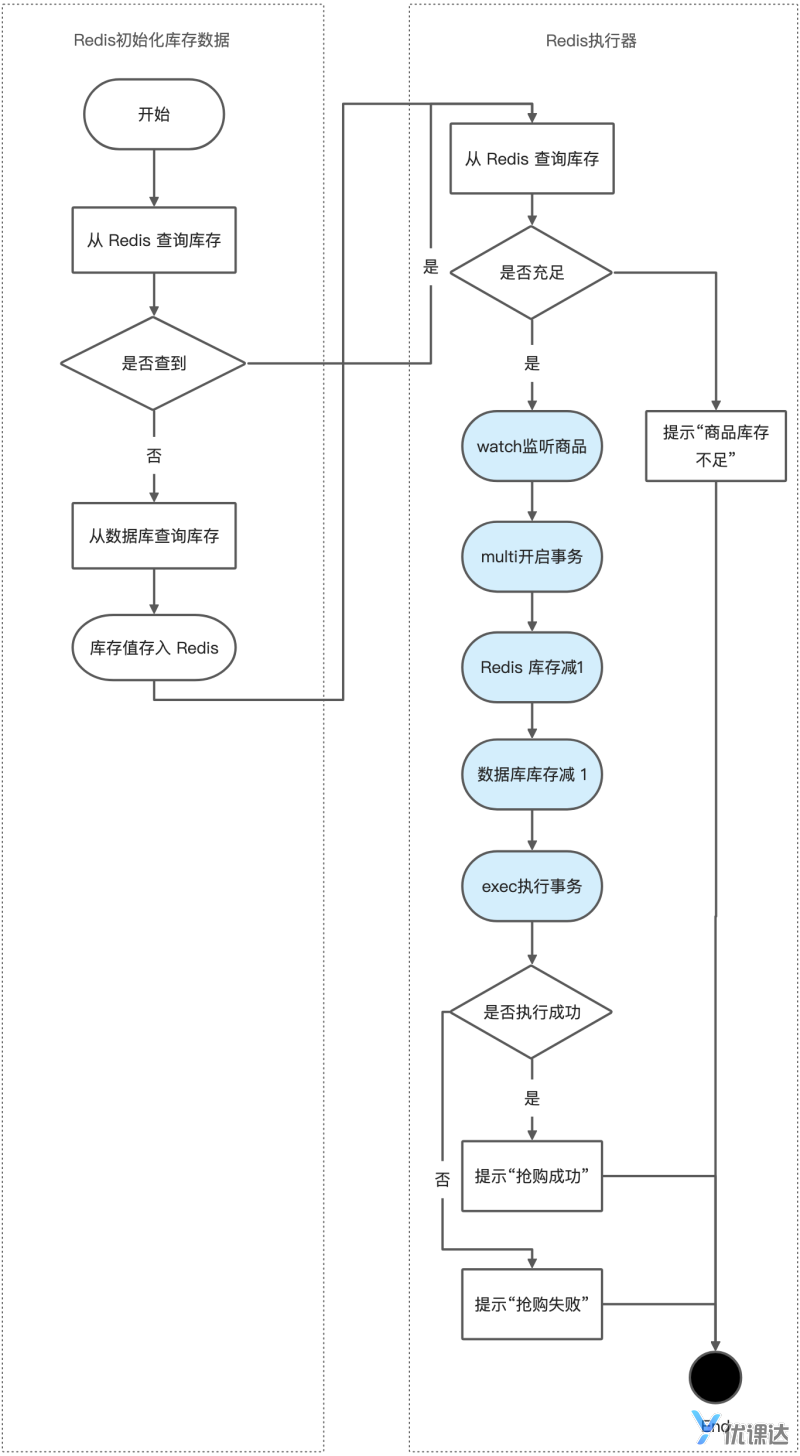
}

}

});

**return** result;

}



**public** Result<Boolean> **snappedUp**(**@RequestParam("id")** Long id) {

*//初始化返回数据*

Result result = **new** Result();

result.data(**true**);

result.setSuccess(**true**);

*//实现购买代码*

*//先去redis查询一下*

Object value = redisTemplate.opsForValue().get(id);

**int** stock = 0;

*//如果redis没有则去数据库查询*

**if** (value == **null**) {

*//去数据库查询该商品的信息*

ProductDO product = productDAO.selectById(id);

*//将信息缓存到redis里边*

stock = product.getStock();

redisTemplate.opsForValue().set(product.getId(), stock);

} **else** {

stock = (**int**)value;

}

redisTemplate.execute(**new** SessionCallback<List<Object>>() {

**@Override**

**public** List<Object> **execute**(RedisOperations operations) **throws** DataAccessException {

Integer stock = (Integer)operations.opsForValue().get(id);

*//判断该商品的库存是否大于1*

**if** (Integer.valueOf(stock) >= 1) {

*//监听商品的名字，redis里边的key*

operations.watch(id);

*//开启事务*

operations.multi();

*//将该商品的库存自减1*

operations.opsForValue().set(id, stock - 1);

*//修改mysql数据库库存数量*

ProductDO productDO = **new** ProductDO();

productDO.setId(id);

productDO.setStock(stock - 1);

productDAO.updateStock(productDO);

*// 执行事务*

List exec = operations.exec();

**if** (exec.size() > 0) {

*// TODO:可以有其它业务逻辑，例如插入订单等，视具体需求而定*

result.setMessage("抢购成功");

result.setData(**true**);

} **else** {

result.setMessage("抢购失败");

result.setData(**false**);

}

**return** exec;

} **else** {

result.setMessage("商品库存不足");

result.setData(**false**);

**return** **null**;

}

}

});

**return** result;

}

**执行事务**

redisTemplate.execute() 是执行器方法，可以执行一系列的操作。使用Redis事务时，

redisTemplate.execute(**new** SessionCallback<List<Object>>() {

**@Override**

**public** List<Object> **execute**(RedisOperations operations) **throws** DataAccessException {

}

});

属于固定写法。

事务的主要逻辑：

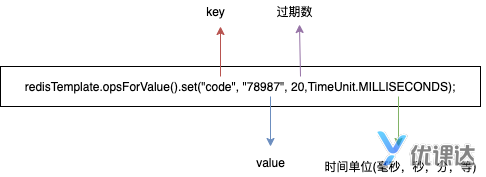
监听->开启事务->读写数据->执行事务

Redis过期处理

**设置过期时间**

设置1000ms过期：

edisTemplate.opsForValue().set("code", "78987", 1000,TimeUnit.MILLISECONDS);



**删除策略**

1.惰性删除

2.定期删除

3.定时删除