Redisson 分布式锁实现秒杀系统

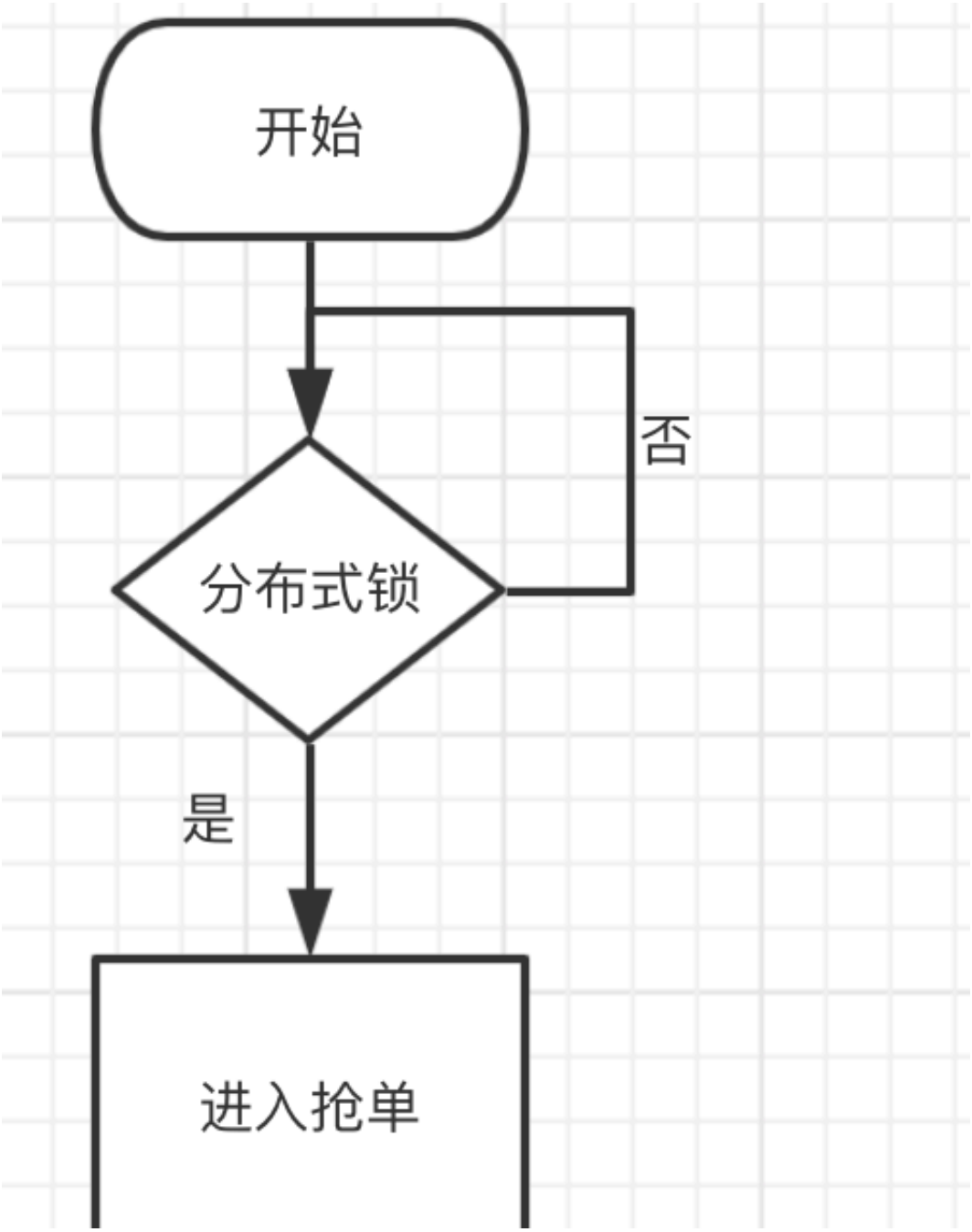
1、设计思路

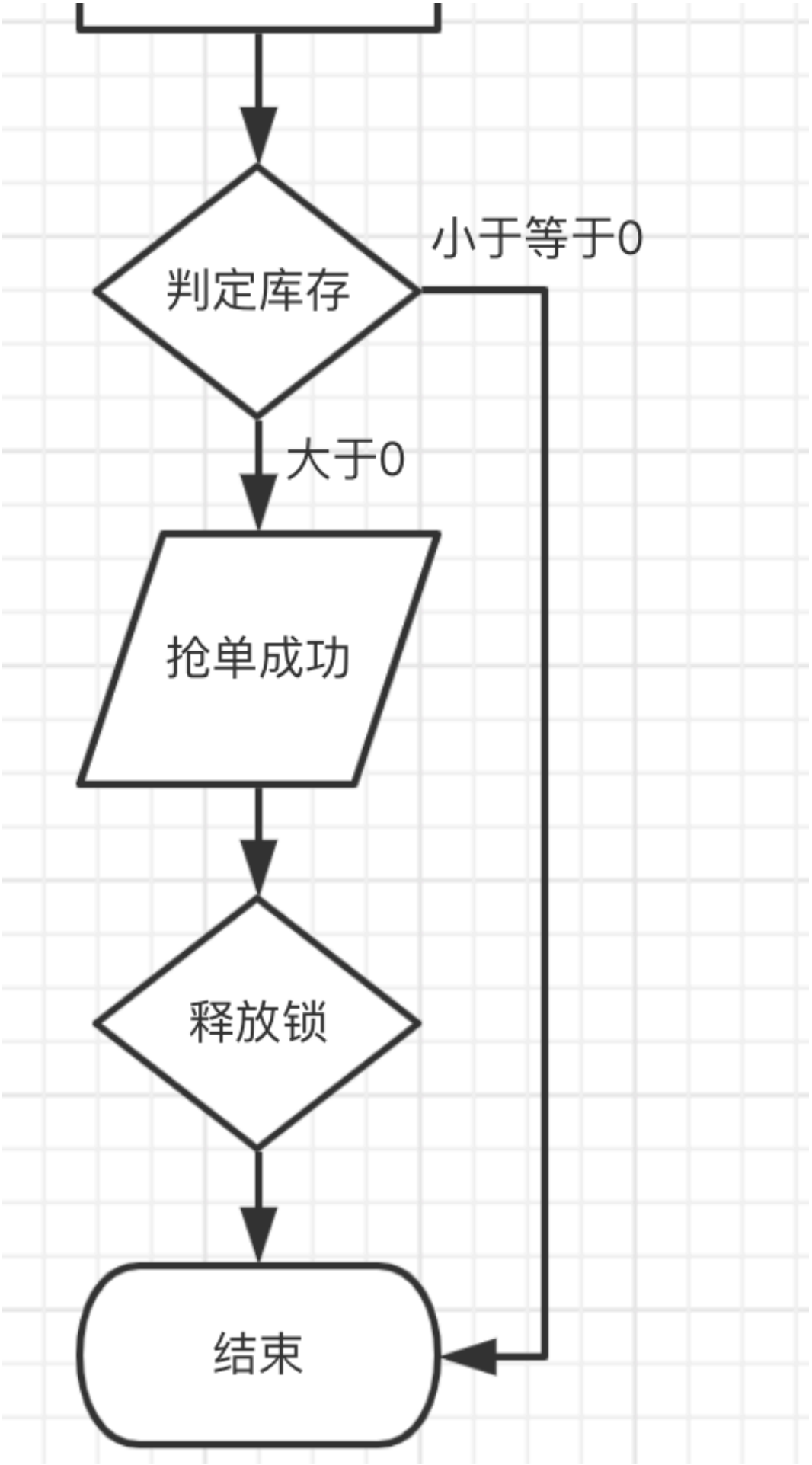
（1）通过 redis 的redisson分布式锁控制高并发，防止超卖

（2）构造用户

（3）通过parallelStream流，来模拟并发

2、设计流程图





3、设计相关类

RedissonConfig.java

```java
package com.xkcoding.redisson.config;

import lombok.Data;
import lombok.extern.slf4j.Slf4j;
import org.redisson.Redisson;
import org.redisson.api.RedissonClient;
import org.redisson.codec.JsonJacksonCodec;
import org.redisson.config.Config;
import
org.springframework.boot.context.properties.ConfigurationPro
perties;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.util.StringUtils;

import java.io.IOException;

/**
 * 功能描述:
 *
 * @Author dawei.lv  -  daweilv@pateo.com.cn
 * @Date 2020/3/26 16:26
 */
@Slf4j
@Data
@ConfigurationProperties(prefix = "redisson")
@Configuration
public class RedissonConfig {

    private String address;
    private String password;
    private Integer database;

    @Bean(name = "redissonClient")
```

```
33      public RedissonClient redissonClientSingle() throws
IOException {
34          Config config = new Config();
35          config.useSingleServer().setDatabase(database);
36          config.useSingleServer().setAddress(address);
37
38          // 此处设置编码，如果设成其它编码，在 redis 中看到的是二
进制样式
39          config.setCodec(new JsonJacksonCodec());
40
41          if (!StringUtils.isEmpty(password)) {
42              config.useSingleServer().setPassword(password);
43          }
44
45          RedissonClient redissonClient =
Redisson.create(config);
46
47          log.info("redissonClient: {}",
redissonClient.getConfig().toJSON());
48
49          return redissonClient;
50      }
51
52
53  }
```

IDistributedLocker.java

```
1  package com.xkcoding.redisson.lock;
2
3  import org.redisson.api.RLock;
4
5  import java.util.concurrent.TimeUnit;
6
7  /**
8   * 功能描述:
9   *
```

```
 * @Author dawei.lv  -  daweilv@pateo.com.cn
 * @Date 2020/3/26 16:36
 */
public interface IDistributedLocker {

    /**
     * 拿不到锁就不罢休，不然线程一直 block
     * @param lockKey
     * @return
     */
    RLock lock(String lockKey);

    /**
     * leaseTime 为加锁时间，单位为秒
     * @param lockey
     * @param leaseTime
     * @return
     */
    RLock lock(String lockey, long leaseTime);

    /**
     * timeout为加锁时间，时间单位由unit确定
     * @param lockKey
     * @param unit
     * @param timeout
     * @return
     */
    RLock lock(String lockKey, TimeUnit unit, long timeout);

    /**
     *tryLock()，马上返回，拿到lock就返回true，不然返回false。
     * 带时间限制的tryLock()，拿不到lock，就等一段时间，超时返回false.
     * @param lockKey
     * @param unit
     * @param waitTime
     * @param leaseTime
```

```
     * @return
     */
    boolean tryLock(String lockKey, TimeUnit unit, long waitTime, long leaseTime);

    /**
     *  释放 key 的锁
     * @param lockKey
     */
    void unlock(String lockKey);

    /**
     * 释放锁
     * @param lock
     */
    void unlock(RLock lock);
}
```

RedissonDistributedLocker.java

```
package com.xkcoding.redisson.lock;

import lombok.extern.slf4j.Slf4j;
import org.redisson.api.RLock;
import org.redisson.api.RedissonClient;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import javax.annotation.Resource;
import java.util.concurrent.TimeUnit;

/**
 * 功能描述:
 *
 * @Author dawei.lv  -  daweilv@pateo.com.cn
 * @Date 2020/3/26 16:39
```

```
 */
@Slf4j
@Component
public class RedissonDistributedLocker implements
IDistributedLocker {

    @Resource
    RedissonClient redissonClient;


    @Override
    public RLock lock(String lockKey) {
        RLock lock = redissonClient.getLock(lockKey);
        lock.lock();
        return lock;
    }

    @Override
    public RLock lock(String lockKey, long leaseTime) {
        RLock lock = redissonClient.getLock(lockKey);
        lock.lock(leaseTime, TimeUnit.SECONDS);
        return lock;
    }

    @Override
    public RLock lock(String lockKey, TimeUnit unit, long
timeout) {
        RLock lock = redissonClient.getLock(lockKey);
        lock.lock(timeout, unit);
        return lock;
    }

    @Override
    public boolean tryLock(String lockKey, TimeUnit unit,
long waitTime, long leaseTime) {
        RLock lock = redissonClient.getLock(lockKey);
        try {
```

```
            return lock.tryLock(waitTime, leaseTime, unit);
        } catch (Exception e) {
            log.info("tryLock 失败, {}", e);
            return false;
        }
    }

    @Override
    public void unlock(String lockKey) {
        RLock lock = redissonClient.getLock(lockKey);
        lock.unlock();
    }

    @Override
    public void unlock(RLock lock) {
        lock.unlock();
    }
}
```

QiangHongBaoController.java

```
package com.xkcoding.controller;

import com.google.common.collect.Lists;
import com.xkcoding.redisson.lock.RedissonDistributedLocker;
import lombok.extern.slf4j.Slf4j;
import org.redisson.api.RBucket;
import org.redisson.api.RedissonClient;
import org.springframework.util.StringUtils;
import
org.springframework.web.bind.annotation.RequestMapping;
import
org.springframework.web.bind.annotation.RestController;

import javax.annotation.Resource;
import java.util.ArrayList;
import java.util.List;
```

```java
import java.util.concurrent.TimeUnit;
import java.util.stream.IntStream;

/**
 * 功能描述:  通过 redis 的分布式锁实现秒杀系统
 *
 * @Author dawei.lv  -  daweilv@pateo.com.cn
 * @Date 2020/3/27 14:01
 */
@Slf4j
@RestController
@RequestMapping
public class QiangHongBaoController {

    private Integer kucun;

    private String goodsKey = "goodsKey";

    private int timeout = 30 * 1000;

    @Resource
    RedissonDistributedLocker redissonDistributedLocker;
    @Resource
    RedissonClient redissonClient;

    @RequestMapping("/qingdan")
    public List<String> qingHongBao() {

        ArrayList<String> shopUsers = Lists.newArrayList();
        ArrayList<String> users = Lists.newArrayList();

        IntStream.range(0, 10).forEach(b -> {
            users.add("用户" + b);
        });

        RBucket<Object> bucket =
redissonClient.getBucket("aaaa");
```

```
        kucun = (Integer) bucket.get();

        users.parallelStream().forEach(b -> {
            String shopUser = qiangdan(b);
            if (!StringUtils.isEmpty(shopUser)) {
                shopUsers.add(shopUser);
            }
        });

        System.out.println("抢单结束");
        return shopUsers;
    }

    private String qiangdan(String user) {


        if (kucun <= 0) {
            return "";
        }

        boolean lock =
redissonDistributedLocker.tryLock(goodsKey,
TimeUnit.SECONDS, timeout, 1000);
        //
        if (lock) {
            log.info("用户{}拿到锁。。。", user);

            try {


                if (kucun <= 0) {
                    return "";
                }

                try {
                    TimeUnit.NANOSECONDS.sleep(100);
                } catch (InterruptedException e) {
```

```
                    e.printStackTrace();
                }


                kucun -= 1;
                RBucket<Object> bucket = redissonClient.getBucket("aaaa");
                bucket.set(kucun);

                log.info("用户{}抢单成功，所剩库存：{}", user, kucun);

                return user + "抢单成功，所剩库存：" + kucun;
            } finally {
                log.info("用户{}释放锁...", user);
                redissonDistributedLocker.unlock(goodsKey);
            }

        } else {

        }
        return "";
    }
}
```

application.yml

```
redisson:
  address: redis://localhost:6379
  password:
  database: 5

mybatis-plus:
  mapper-locations: classpath:mapper/*.xml
  type-aliases-package: com.xkcoding.mybatis.entity

spring:
```

```yaml
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://localhost:3306/spring-boot-demo?useUnicode=true&characterEncoding=utf-8&useSSL=false
    username: root
    password: 123456
  hikari:
    auto-commit: true
    connection-timeout: 30000
    idle-timeout: 600000
    max-lifetime: 1800000
    minimum-idle: 5
    maximum-pool-size: 10
    read-only: false

server:
  port: 9090
```