### 预约挂号功能中，存在着多位患者同时预约一位医生，但仅剩余一个预约名额的情况。为避免这种情况造成的冲突，我们有必要引入高并发处理机制。目前比较常用的处理高并发的方法有：消息队列、锁机制等，而锁机制又包括synchronized同步锁，Redis分布式锁、Zookeeper分布式锁等等。经过对比，该项目最终选择了Redis分布式锁，我们将详细对比解释选择Redis锁的原因。

### 1. synchronized同步锁

synchronized是Java自带的关键字，因此其最大的优点是实现简单，并且不需要手动释放，由JVM自行控制。但我们的项目类似于秒杀活动，JVM级别的锁不能解决分布式场景的问题。

**synchronized**(**this**)

{  
 */\*预约挂号相关功能\*/* }

### 2. ZooKeeper分布式锁

ZooKeeper分布式锁作用原理为，某一个客户端如果抢到资源，会加锁并建立一个节点，直到执行结束该节点才会删除。其他客户端在这期间的访问都会被拒绝，但是会创建一个监听器，待锁被释放，便会抢占资源。因此，区别于Redis的不断访问以试图获得资源的使用权，它拥有更高的效率。但是，ZooKeeper实现起来比较复杂，需要维持一个相当大的堆栈。Redis实现的分布式锁对于我们的社区医院流量已经足够，因此，该项目最终还是选择了Redis锁。

### 4.2.4 Redis分布式锁

Redis锁实现比较简单，仅需要导入需要的jar包，并引入需要的工具类即可，并且具有很高的性能，可以支撑高并发的加锁、释放锁操作，因此该项目选择了Redis锁解决我们的高并发问题，下面是为了增强锁的健壮性采取的一些操作：

1. 基于SETNX实现Redis分布式锁

为了确保每一次释放的锁与当前被锁住的锁是同一个资源，我们使用lockKey为锁添加一个标记，保证加锁和释放的操作都是针对同一个对象。其次，为了防止预约挂号的相关操作出现JVM异常，导致解锁操作无法正常执行，将解锁操作放到finally中，确保不会出现死锁。

1. 基于SETEX实现Redis分布式锁

为防止宕机问题的出现导致其他线程始终得不到资源，还需要引入expireTime，为锁设置过期时间，如果没有在截止时间前完成，也会自动释放资源，进一步防止死锁的出现。

1. 其他线程导致的解锁问题

当我们引入了过期时间后，假设线程一到达了自己的过期时间释放资源，线程二自动获取资源并加锁，但线程一仍继续运行了解锁过程，导致此时拥有锁的线程二被意外解锁。为了防止这样的问题出现，我们加入了requestId，并赋值为当前时间，因为当前时间具有唯一性，所以每一个进程的requestId都不相同，当执行解锁时，不仅会判断lockKey是否一致，还会判断其对应的requestId是否一致。这样我们就避免了其他线程错误的解锁当前线程的问题了。

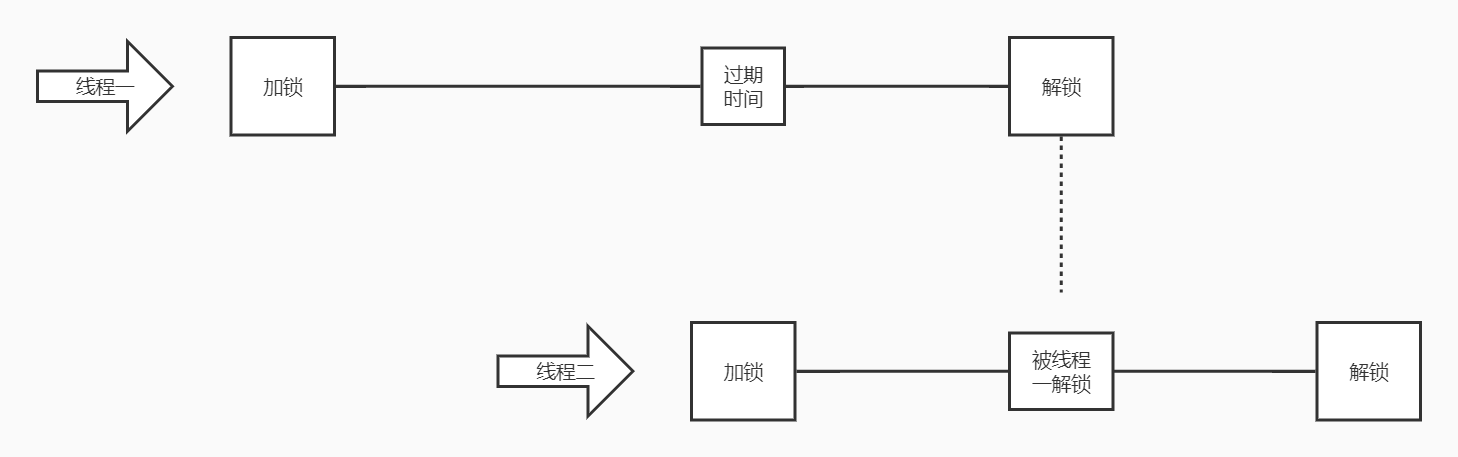


图4.6 其他线程误解锁问题示例

经过以上三种方式的改进，我们将Redis实现的分布式锁运用到预约挂号的处理中，从而解决对资源获取的高并发问题。核心代码如下：

（在IDEA，AppAppointmentService.java类的appointment方法中）



关于以上三个锁的理论知识，感兴趣的话可以看一下【你从没有见过的，高并发架构实战 教学-哔哩哔哩】 IMG_256https://b23.tv/DcmqhqD

安装Redis

【Windows系统的Redis安装和简单使用-哔哩哔哩】 IMG_256https://b23.tv/YlKyaZy

看到4分30秒就可以了，后面的配置不需要弄

如果安装不顺利，可以把相关代码注释掉，找到下面的代码

boolean lock = RedisTool.*tryGetDistributedLock*("Unlock",timeListBean.getRemainingNum()+"",10000);  
if(lock){  
 if(timeListBean.getRemainingNum().compareTo(0)<=0){  
 return RetUtils.*fail*("该时间段已经预约人满");  
 }  
 Appointment appointment = new Appointment();  
 appointment.setPatientId(userLogin.getRelation());  
 appointment.setDate(date);  
 appointment.setDepartment(department);  
 appointment.setDoctorId(doctorId);  
 appointment.setDoctorName(doctor.getName());  
 appointment.setTimeEnum(timeEnum.getValue());  
 appointment.setStatus(0);  
 b = appointment.save();  
 RedisTool.*releaseDistributedLock*("Unlock",timeListBean.getRemainingNum()+"");  
}

改为（预约挂号也可以正常使用）：

if(timeListBean.getRemainingNum().compareTo(0)<=0){  
 return RetUtils.*fail*("该时间段已经预约人满");  
 }  
 Appointment appointment = new Appointment();  
 appointment.setPatientId(userLogin.getRelation());  
 appointment.setDate(date);  
 appointment.setDepartment(department);  
 appointment.setDoctorId(doctorId);  
 appointment.setDoctorName(doctor.getName());  
 appointment.setTimeEnum(timeEnum.getValue());  
 appointment.setStatus(0);  
 b = appointment.save();

如果安装顺利，可以试一下，把上面的代码改为，这是最完整的Redis

LocalDate now = LocalDate.*now*();  
boolean lock = RedisTool.*tryGetDistributedLock*("Unlock",now+"",10000);  
try{  
 if(lock){  
 if(timeListBean.getRemainingNum().compareTo(0)<=0){  
 return RetUtils.*fail*("该时间段已经预约人满");  
 }  
 Appointment appointment = new Appointment();  
 appointment.setPatientId(userLogin.getRelation());  
 appointment.setDate(date);  
 appointment.setDepartment(department);  
 appointment.setDoctorId(doctorId);  
 appointment.setDoctorName(doctor.getName());  
 appointment.setTimeEnum(timeEnum.getValue());  
 appointment.setStatus(0);  
 b = appointment.save();}  
}finally {  
 RedisTool.*releaseDistributedLock*("Unlock",now+"");  
}