36 从容不迫：重写锁的设计结构和细节

更新时间：2019-11-15 11:04:05

受苦的人，没有悲观的权利。

——尼采

引导语

有的面试官喜欢让同学在说完锁的原理之后，让你重写一个新的锁，要求现场在白板上写出大概的思路和代码逻辑，这种面试题目，蛮难的，我个人觉得其侧重点主要是两个部分：

1. 考察一下你对锁原理的理解是如何来的，如果你对源码没有解读过的话，只是看看网上的文章，或者背面试题，也是能够说出大概的原理，但你很难现场写出一个锁的实现代码，除非你真的看过源码，或者有和锁相关的项目经验；
2. 我们不需要创造，我们只需要模仿 Java 锁中现有的 API 进行重写即可。

如果你看过源码，这道题真的很简单，你可以挑选一个你熟悉的锁进行模仿。

在锁章节中我们之前说的都是排它锁，这小节我们以共享锁作为案列，自定义一个共享锁。

1 需求

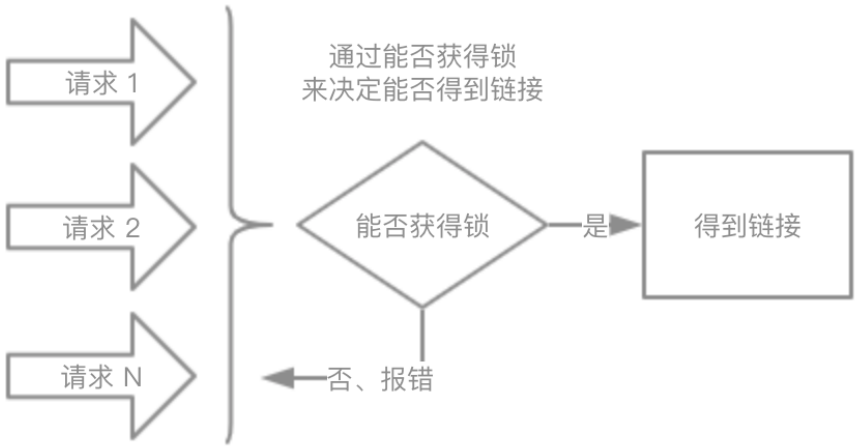
一般自定义锁的时候，我们都是根据需求来进行定义的，不可能凭空定义出锁来，说到共享锁，大家可能会想到很多场景，比如说对于共享资源的读锁可以是共享的，比如对于数据库链接的共享访问，比如对于 Socket 服务端的链接数是可以共享的，场景有很多，我们选择共享访问数据库链接这个场景来定义一个锁。

2 详细设计

假定(以下设想都为假定)我们的数据库是单机 mysql，只能承受 10 个链接，创建数据库链接时，我们是通过最原始 JDBC 的方式，我们用一个接口把用 JDBC 创建链接的过程进行了封

共享访问数据库链接的整体要求如下：所有请求加在一起的 mysql 链接数，最大不能超过 10（包含 10），一旦超过 10，直接报错。

在这个背景下，我们进行了下图的设计：



这个设计最最关键的地方，就是我们通过能否获得锁，来决定是否可以得到 mysql 链接，如果能获得锁，那么就能得到链接，否则直接报错。

接着我们一起来看下落地的代码：

2.1 定义锁

首先我们需要定义一个锁出来，定义时需要有两个元素：

1. 锁的定义：同步器 Sync；
2. 锁对外提供的加锁和解锁的方法。

共享锁的代码实现如下：

```
// 共享不公平锁
public class ShareLock implements Serializable{
      // 同步器
  private final Sync sync;
  // 用于确保不能超过最大值
  private final int maxCount;

  /**
   * 初始化时给同步器 sync 赋值
   * count 代表可以获得共享锁的最大值
   */
  public ShareLock(int count) {
    this.sync = new Sync(count);
    maxCount = count;
  }

  /**
   * 获得锁
   * @return true 表示成功获得锁，false 表示失败
   */
  public boolean lock(){
    return sync.acquireByShared(1);
  }
```

```
   */
  public boolean unLock(){
    return sync.releaseShared(1);
  }
}
```

从上述代码中可以看出，加锁和释放锁的实现，都依靠同步器 Sync 的底层实现。

唯一需要注意的是，锁需要规定好 API 的规范，主要是两方面：

1. API 需要什么，就是锁在初始化的时候，你需要传哪些参数给我，在 ShareLock 初始化时，需要传最大可共享锁的数目；
2. 需要定义自身的能力，即定义每个方法的入参和出参。在 ShareLock 的实现中，加锁和释放锁的入参都没有，是方法里面写死的 1，表示每次方法执行，只能加锁一次或释放锁一次，出参是布尔值，true 表示加锁或释放锁成功，false 表示失败，底层使用的都是 Sync 非公平锁。

以上这种思考方式是有方法论的，就是我们在思考一个问题时，可以从两个方面出发：API 是什么？API 有什么能力？

2.2 定义同步器 Sync

Sync 直接继承 AQS ，代码如下：


```
class Sync extends AbstractQueuedSynchronizer{
  // 表示最多有 count 个共享锁可以获得
  public Sync(int count) {
    setState(count);
  }

  // 获得 i 个锁
  public boolean acquireByShared(int i) {
    // 自旋保证 CAS 一定可以成功
    for(;;){
      if(i<=0){
        return false;
      }
      int state = getState();
      // 如果没有锁可以获得，直接返回 false
      if(state <=0 ){
        return false;
      }
      int expectState = state - i;
      // 如果要得到的锁不够了，直接返回 false
      if(expectState < 0 ){
        return false;
      }
      // CAS 尝试得到锁,CAS 成功获得锁，失败继续 for 循环
      if(compareAndSetState(state,expectState)){
        return true;
      }
    }
  }

  // 释放 i 个锁
  @Override
```

```
      return false;
    }
    int state = getState();
    int expectState = state + arg;
    // 超过了 int 的最大值，或者 expectState 超过了我们的最大预期
    if(expectState < 0 || expectState > maxCount){
      log.error("state 超过预期，当前 state is {},计算出的 state is {}",state
      ,expectState);
      return false;
    }
    if(compareAndSetState(state, expectState)){
      return true;
    }
   }
  }
}
```

整个代码比较清晰，我们需要注意的是：

1. 边界的判断，比如入参是否非法，释放锁时，会不会出现预期的 state 非法等边界问题，对于此类问题我们都需要加以判断，体现出思维的严谨性；
2. 加锁和释放锁，需要用 for 自旋 + CAS 的形式，来保证当并发加锁或释放锁时，可以重试成功。写 for 自旋时，我们需要注意在适当的时机要 return，不要造成死循环，CAS 的方法 AQS 已经提供了，不要自己写，我们自己写的 CAS 方法是无法保证原子性的。

2.3 通过能否获得锁来决定能否得到链接

锁定义好了，我们需要把锁和获取 Mysql 链接结合起来，我们写了一个 Mysql 链接的工具类，叫做 MysqlConnection，其主要负责两大功能：

1. 通过 JDBC 建立和 Mysql 的链接；
2. 结合锁，来防止请求过大时，Mysql 的总链接数不能超过 10 个。

首先我们看下 MysqlConnection 初始化的代码：

```
public class MysqlConnection {
  private final ShareLock lock;

  // maxConnectionSize 表示最大链接数
  public MysqlConnection(int maxConnectionSize) {
    lock = new ShareLock(maxConnectionSize);
  }
}
```

我们可以看到，在初始化时，需要制定最大的链接数是多少，然后把这个数值传递给锁，因为最大的链接数就是 ShareLock 锁的 state 值。

接着为了完成 1，我们写了一个 private 的方法：

```
// 得到一个 mysql 链接，底层实现省略
private Connection getConnection(){}
```

然后我们实现 2，代码如下：

```
// 即使出现未知异常，也无需释放锁
public Connection getLimitConnection() {
  if (lock.lock()) {
    return getConnection();
  }
  return null;
}

// 对外释放 mysql 链接的接口
public boolean releaseLimitConnection() {
  return lock.unLock();
}
```

逻辑也比较简单，加锁时，如果获得了锁，就能返回 Mysql 的链接，释放锁时，在链接关闭成功之后，调用 releaseLimitConnection 方法即可，此方法会把锁的 state 状态加一，表示链接被释放了。

以上步骤，针对 Mysql 链接限制的场景锁就完成了。

3 测试

锁写好了，接着我们来测试一下，我们写了一个测试的 demo，代码如下：

```
public static void main(String[] args) {
  log.info("模仿开始获得 mysql 链接");
  MysqlConnection mysqlConnection = new MysqlConnection(10);
  log.info("初始化 Mysql 链接最大只能获取 10 个");
  for(int i =0 ;i<12;i++){
    if(null != mysqlConnection.getLimitConnection()){
      log.info("获得第{}个数据库链接成功",i+1);
    }else {
      log.info("获得第{}个数据库链接失败：数据库连接池已满",i+1);
    }
  }
  log.info("模仿开始释放 mysql 链接");
  for(int i =0 ;i<12;i++){
    if(mysqlConnection.releaseLimitConnection()){
      log.info("释放第{}个数据库链接成功",i+1);
    }else {
      log.info("释放第{}个数据库链接失败",i+1);
    }
  }
  log.info("模仿结束");
}
```


以上代码逻辑如下：

1. 获得 Mysql 链接逻辑：for 循环获取链接，1~10 都可以获得链接，11~12 获取不到链接，因为链接被用完了；
2. 释放锁逻辑：for 循环释放链接，1~10 都可以释放成功，11~12 释放失败。

我们看下运行结果，如下图：

目录

```
17:21:37.239 [main] INFO demo.sixth.MysqlConnection - 获得第4个数据库链接成功
17:21:37.240 [main] INFO demo.sixth.MysqlConnection - 获得第5个数据库链接成功
17:21:37.240 [main] INFO demo.sixth.MysqlConnection - 获得第6个数据库链接成功
17:21:37.240 [main] INFO demo.sixth.MysqlConnection - 获得第7个数据库链接成功
17:21:37.240 [main] INFO demo.sixth.MysqlConnection - 获得第8个数据库链接成功
17:21:37.240 [main] INFO demo.sixth.MysqlConnection - 获得第9个数据库链接成功
17:21:37.240 [main] INFO demo.sixth.MysqlConnection - 获得第10个数据库链接成功
17:21:37.240 [main] INFO demo.sixth.MysqlConnection - 获得第11个数据库链接失败：数据库连接池已满
17:21:37.240 [main] INFO demo.sixth.MysqlConnection - 获得第12个数据库链接失败：数据库连接池已满
17:21:37.240 [main] INFO demo.sixth.MysqlConnection - 模仿开始释放 mysql 链接
17:21:37.240 [main] INFO demo.sixth.MysqlConnection - 释放第1个数据库链接成功
17:21:37.240 [main] INFO demo.sixth.MysqlConnection - 释放第2个数据库链接成功
17:21:37.241 [main] INFO demo.sixth.MysqlConnection - 释放第3个数据库链接成功
17:21:37.241 [main] INFO demo.sixth.MysqlConnection - 释放第4个数据库链接成功
17:21:37.242 [main] INFO demo.sixth.MysqlConnection - 释放第5个数据库链接成功
17:21:37.245 [main] INFO demo.sixth.MysqlConnection - 释放第6个数据库链接成功
17:21:37.246 [main] INFO demo.sixth.MysqlConnection - 释放第7个数据库链接成功
17:21:37.249 [main] INFO demo.sixth.MysqlConnection - 释放第8个数据库链接成功
17:21:37.251 [main] INFO demo.sixth.MysqlConnection - 释放第9个数据库链接成功
17:21:37.251 [main] INFO demo.sixth.MysqlConnection - 释放第10个数据库链接成功
17:21:37.251 [main] ERROR demo.sixth.ShareLock - state 超过预期，当前 state is 10,计算出的 state is 11
17:21:37.251 [main] INFO demo.sixth.MysqlConnection - 释放第11个数据库链接失败
17:21:37.252 [main] ERROR demo.sixth.ShareLock - state 超过预期，当前 state is 10,计算出的 state is 11
17:21:37.252 [main] INFO demo.sixth.MysqlConnection - 释放第12个数据库链接失败
17:21:37.252 [main] INFO demo.sixth.MysqlConnection - 模仿结束

Process finished with exit code 0
```

从运行的结果，可以看出，我们实现的 ShareLock 锁已经完成了 Mysql 链接共享的场景了。

4 总结

同学们阅读到这里不知道有没有两点感受：

1. 重写锁真的很简单，最关键的是要和场景完美贴合，能满足业务场景的锁才是好锁；
2. 锁真实只是来满足业务场景的，本质都是 AQS，所以只要 AQS 学会了，在了解清楚场景的情况下，重写锁都不难的。

锁章节最核心的就是 AQS 源码解析的两章，只要我们把 AQS 弄懂了，其余锁的实现，只要稍微看下源码实现，几乎马上就能知道其底层实现的原理，大多数都是通过操作 state 来完成不同的场景需求，所以还是建议大家多看 AQS 源码，多 debug AQS 源码，只要 AQS 弄清楚了，锁都很简单。

← 35 经验总结：各种锁在工作中使用场景和细节

37 ThreadPoolExecutor 源码解析 →

精选留言 1

欢迎在这里发表留言，作者筛选后可公开显示

慕码人6169125

锁这一系列环环相扣，老师写的太好了。目前读过的解析AQS相当清楚的文章。结合源码看收获很大

0 回复 2019-11-18

文贺 回复 **慕码人6169125**

谢谢肯定，一起加油进步。

回复 2019-11-23 16:44:22

目录

千学不如一看，千看不如一练