慕课专栏

: ■ 面试官系统精讲Java源码及大厂真题 / 35 经验总结 : 各种锁在工作中使用场景和细节

目录

第1章 基础

01 开篇词: 为什么学习本专栏

02 String、Long 源码解析和面试题

03 Java 常用关键字理解

04 Arrays、Collections、Objects 常 用方法源码解析

第2章 集合

05 ArrayList 源码解析和设计思路

06 LinkedList 源码解析

07 List 源码会问哪些面试题

08 HashMap 源码解析

09 TreeMap 和 LinkedHashMap 核心 源码解析

10 Map源码会问哪些面试题

11 HashSet、TreeSet 源码解析

12 彰显细节: 看集合源码对我们实际 工作的帮助和应用

13 差异对比: 集合在 Java 7 和 8 有何 不同和改进

14 简化工作: Guava Lists Maps 实际 工作运用和源码

第3章 并发集合类

15 CopyOnWriteArrayList 源码解析 和设计思路

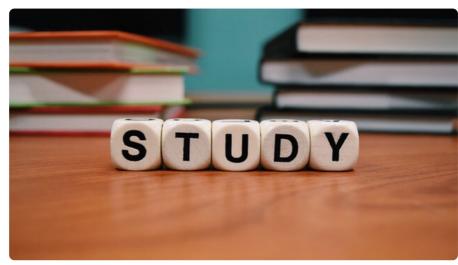
16 ConcurrentHashMap 源码解析和 设计思路

17 并发 List、Map源码面试题

18 场景集合:并发 List、Map的应用

35 经验总结:各种锁在工作中使用场景和细节

更新时间: 2019-11-14 11:04:28



富贵必从勤苦得。

引导语

本章主要说一说锁在工作中的使用场景,主要以 synchronized 和 CountDownLatch 为例,会 分别描述一下这两种锁的使用场景和姿势。

1 synchronized

synchronized 是可重入的排它锁,和 ReentrantLock 锁功能相似,任何使用 synchronized 的地方,几乎都可以使用 ReentrantLock 来代替,两者最大的相似点就是:可重入 + 排它锁, 两者的区别主要有这些:

- 1. ReentrantLock 的功能更加丰富,比如提供了 Condition,可以打断的加锁 API、能满足 锁 + 队列的复杂场景等等:
- 2. ReentrantLock 有公平锁和非公平锁之分, 而 synchronized 都是非公平锁;
- 3. 两者的使用姿势也不同,ReentrantLock 需要申明,有加锁和释放锁的 API,而 synchronized 会自动对代码块进行加锁释放锁的操作, synchronized 使用起来更加方 便。

synchronized 和 ReentrantLock 功能相近,所以我们就以 synchronized 举例。

1.1 共享资源初始化

在分布式的系统中,我们喜欢把一些死的配置资源在项目启动的时候加锁到 JVM 内存里面去, 这样请求在拿这些共享配置资源时,就可直接从内存里面拿,不必每次都从数据库中拿,减少了 时间开销。

慕课专栏

目录

第1章 基础

01 开篇词: 为什么学习本专栏

02 String、Long 源码解析和面试题

03 Java 常用关键字理解

04 Arrays、Collections、Objects 常 用方法源码解析

第2章 集合

05 ArrayList 源码解析和设计思路

06 LinkedList 源码解析

07 List 源码会问哪些面试题

08 HashMap 源码解析

09 TreeMap 和 LinkedHashMap 核心 源码解析

10 Map源码会问哪些面试题

11 HashSet、TreeSet 源码解析

12 彰显细节: 看集合源码对我们实际 工作的帮助和应用

13 差异对比: 集合在 Java 7 和 8 有何 不同和改进

14 简化工作: Guava Lists Maps 实际 工作运用和源码

第3章 并发集合类

15 CopyOnWriteArrayList 源码解析 和设计思路

16 ConcurrentHashMap 源码解析和 设计思路

17 并发 List、Map源码面试题

18 场景集合:并发 List、Map的应用

面试官系统精讲Java源码及大厂真题 / 35 经验总结 : 各种锁在工作中使用场景和细节 :=

共享负源创始化的步骤一般习: 项目后动 -> 雕友创始化动作 -> 串线程从数据库中捞取数据 -> 组装成我们需要的数据结构 -> 放到 JVM 内存中。

在项目启动时,为了防止共享资源被多次加载,我们往往会加上排它锁,让一个线程加载共享资 源完成之后,另外一个线程才能继续加载,此时的排它锁我们可以选择 synchronized 或者 ReentrantLock, 我们以 synchronized 为例, 写了 mock 的代码, 如下:

```
// 共享资源
private static final Map<String, String> SHARED_MAP = Maps.newConcurrentMap();
// 有无初始化完成的标志位
private static boolean loaded = false;
* 初始化共享资源
*/
@PostConstruct
public void init(){
 if(loaded){
  return:
 synchronized (this){
  // 再次 check
  if(loaded){
   return:
  log.info("SynchronizedDemo init begin");
  // 从数据库中捞取数据,组装成 SHARED_MAP 的数据格式
  loaded = true:
  log.info("SynchronizedDemo init end");
```

不知道大家有没有从上述代码中发现 @PostConstruct 注解, @PostConstruct 注解的作用是 在 Spring 容器初始化时,再执行该注解打上的方法,也就是说上图说的 init 方法触发的时机, 是在 Spring 容器启动的时候。

大家可以下载演示代码,找到 DemoApplication 启动文件,在 DemoApplication 文件上右击 run, 就可以启动整个 Spring Boot 项目, 在 init 方法上打上断点就可以调试了。

我们在代码中使用了 synchronized 来保证同一时刻,只有一个线程可以执行初始化共享资源的 操作,并且我们加了一个共享资源加载完成的标识位(loaded),来判断是否加载完成了,如 果加载完成,那么其它加载线程直接返回。

如果把 synchronized 换成 ReentrantLock 也是一样的实现,只不过需要显示的使用 ReentrantLock 的 API 进行加锁和释放锁,使用 ReentrantLock 有一点需要注意的是,我们 需要在 try 方法块中加锁,在 finally 方法块中释放锁,这样保证即使 try 中加锁后发生异常, 在 finally 中也可以正确的释放锁。

有的同学可能会问,不是可以直接使用了 ConcurrentHashMap 么,为什么还需要加锁呢?的 确 ConcurrentHashMap 是线程安全的,但它只能够保证 Map 内部数据操作时的线程安全, 是无法保证多线程情况下,查询数据库并组装数据的整个动作只执行一次的,我们加 synchronized 锁住的是整个操作,保证整个操作只执行一次。

完整 demo 见 SynchronizedDemo。

← 慕课专栏

: ■ 面试官系统精讲Java源码及大厂真题 / 35 经验总结: 各种锁在工作中使用场景和细节

目录

第1章 基础

01 开篇词: 为什么学习本专栏

02 String、Long 源码解析和面试题

03 Java 常用关键字理解

04 Arrays、Collections、Objects 常用方法源码解析

第2章 集合

05 ArrayList 源码解析和设计思路

06 LinkedList 源码解析

07 List 源码会问哪些面试题

08 HashMap 源码解析

09 TreeMap 和 LinkedHashMap 核心 源码解析

10 Map源码会问哪些面试题

11 HashSet、TreeSet 源码解析

12 彰显细节:看集合源码对我们实际 工作的帮助和应用

13 差异对比:集合在 Java 7 和 8 有何不同和改进

14 简化工作:Guava Lists Maps 实际 工作运用和源码

第3章 并发集合类

15 CopyOnWriteArrayList 源码解析和设计思路

16 ConcurrentHashMap 源码解析和设计思路

17 并发 List、Map源码面试题

18 场景集合:并发 List、Map的应用

2.1 场景

1:小明在淘宝上买了一个商品,觉得不好,把这个商品退掉(商品还没有发货,只退钱),我们叫做单商品退款,单商品退款在后台系统中运行时,整体耗时 30 毫秒。

2: 双 11, 小明在淘宝上买了 40 个商品, 生成了同一个订单(实际可能会生成多个订单, 为了方便描述, 我们说成一个), 第二天小明发现其中 30 个商品是自己冲动消费的, 需要把 30 个商品一起退掉。

2.2 实现

此时后台只有单商品退款的功能,没有批量商品退款的功能(30 个商品一次退我们称为批量),为了快速实现这个功能,同学 A 按照这样的方案做的: for 循环调用 30 次单商品退款的接口,在 qa 环境测试的时候发现,如果要退款 30 个商品的话,需要耗时: 30 * 30 = 900 毫秒,再加上其它的逻辑,退款 30 个商品差不多需要 1 秒了,这个耗时其实算很久了,当时同学 A 提出了这个问题,希望大家帮忙看看如何优化整个场景的耗时。

同学 B 当时就提出,你可以使用线程池进行执行呀,把任务都提交到线程池里面去,假如机器的 CPU 是 4 核的,最多同时能有 4 个单商品退款可以同时执行,同学 A 觉得很有道理,于是准备修改方案,为了便于理解,我们把两个方案都画出来,对比一下:

同学 A 于是就按照演变的方案去写代码了,过了一天,抛出了一个问题: 向线程池提交了 30 个任务后,主线程如何等待 30 个任务都执行完成呢? 因为主线程需要收集 30 个子任务的执行情况,并汇总返回给前端。

大家可以先不往下看,自己先思考一下,我们前几章说的那种锁可以帮助解决这个问题?

CountDownLatch 可以的,CountDownLatch 具有这种功能,让主线程去等待子任务全部执行完成之后才继续执行。

此时还有一个关键,我们需要知道子线程执行的结果,所以我们用 Runnable 作为线程任务就不行了,因为 Runnable 是没有返回值的,我们需要选择 Callable 作为任务。

我们写了一个 demo, 首先我们来看一下单个商品退款的代码:

// 单商品退款,耗时 30 毫秒,退款成功返回 true,失败返回 false @Slf4j

public class RefundDemo {

/**

- * 根据商品 ID 进行退款
- * @param itemId
- * @return

← 慕课专栏

■ 面试官系统精讲Java源码及大厂真题 / 35 经验总结: 各种锁在工作中使用场景和细节

目录

第1章 基础

01 开篇词: 为什么学习本专栏

02 String、Long 源码解析和面试题

03 Java 常用关键字理解

04 Arrays、Collections、Objects 常用方法源码解析

第2章 集合

05 ArrayList 源码解析和设计思路

06 LinkedList 源码解析

07 List 源码会问哪些面试题

08 HashMap 源码解析

09 TreeMap 和 LinkedHashMap 核心 源码解析

10 Map源码会问哪些面试题

11 HashSet、TreeSet 源码解析

12 彰显细节:看集合源码对我们实际 工作的帮助和应用

13 差异对比:集合在 Java 7 和 8 有何不同和改进

14 简化工作:Guava Lists Maps 实际 工作运用和源码

第3章 并发集合类

15 CopyOnWriteArrayList 源码解析和设计思路

16 ConcurrentHashMap 源码解析和设计思路

17 并发 List、Map源码面试题

18 场景集合:并发 List、Map的应用

178

```
// 线程沉睡 30 毫秒,模拟单个商品退款过程
Thread.sleep(30);
log.info("refund success,itemId is {}", itemId);
return true;
} catch (Exception e) {
log.error("refundByItemError,itemId is {}", itemId);
return false;
}
}
```

接着我们看下 30 个商品的批量退款, 代码如下:

```
@Slf4i
public class BatchRefundDemo {
    // 定义线程池
 public static final ExecutorService EXECUTOR_SERVICE =
   new ThreadPoolExecutor(10, 10, 0L,
                  TimeUnit.MILLISECONDS,
                   new LinkedBlockingQueue<>(20));
 @Test
 public void batchRefund() throws InterruptedException {
  // state 初始化为 30
  CountDownLatch countDownLatch = new CountDownLatch(30);
  RefundDemo refundDemo = new RefundDemo();
  // 准备 30 个商品
  List<Long> items = Lists.newArrayListWithCapacity(30);
  for (int i = 0; i < 30; i++) {
   items.add(Long.valueOf(i+""));
  }
  // 准备开始批量退款
  List<Future> futures = Lists.newArrayListWithCapacity(30);
  for (Long item: items) {
   // 使用 Callable, 因为我们需要等到返回值
   Future < Boolean > future = EXECUTOR_SERVICE.submit(new Callable < Boolean > () {
    @Override
    public Boolean call() throws Exception {
     boolean result = refundDemo.refundByltem(item);
     // 每个子线程都会执行 countDown, 使 state -1 , 但只有最后一个才能真的唤醒主线程
     countDownLatch.countDown();
     return result:
    }
   });
   // 收集批量退款的结果
   futures.add(future);
  log.info("30 个商品已经在退款中");
  // 使主线程阻塞, 一直等待 30 个商品都退款完成, 才能继续执行
  countDownLatch.await();
  log.info("30 个商品已经退款完成");
  // 拿到所有结果进行分析
  List<Boolean> result = futures.stream().map(fu-> {
    // get 的超时时间设置的是 1 毫秒,是为了说明此时所有的子线程都已经执行完成了
    return (Boolean) fu.get(1,TimeUnit.MILLISECONDS);
   } catch (InterruptedException e) {
    e.printStackTrace();
   } catch (ExecutionException e) {
```

← 慕课专栏

: ■ 面试官系统精讲Java源码及大厂真题 / 35 经验总结: 各种锁在工作中使用场景和细节

目录

第1章 基础

01 开篇词: 为什么学习本专栏

02 String、Long 源码解析和面试题

03 Java 常用关键字理解

04 Arrays、Collections、Objects 常用方法源码解析

第2章 集合

05 ArrayList 源码解析和设计思路

06 LinkedList 源码解析

07 List 源码会问哪些面试题

08 HashMap 源码解析

09 TreeMap 和 LinkedHashMap 核心 源码解析

10 Map源码会问哪些面试题

11 HashSet、TreeSet 源码解析

12 彰显细节:看集合源码对我们实际 工作的帮助和应用

13 差异对比:集合在 Java 7 和 8 有何不同和改进

14 简化工作:Guava Lists Maps 实际工作运用和源码

第3章 并发集合类

15 CopyOnWriteArrayList 源码解析和设计思路

16 ConcurrentHashMap 源码解析和设计思路

17 并发 List、Map源码面试题

18 场景集合:并发 List、Map的应用

10 勿泉来日:月及 1.31、 1/140月300万

```
}
return false;
}).collect(Collectors.toList());
// 打印结果统计
long success = result.stream().filter(r->r.equals(true)).count();
log.info("执行结果成功{},失败{}",success,result.size()-success);
}
}
```

上述代码只是大概的底层思路,真实的项目会在此思路之上加上请求分组,超时打断等等优化措施。

我们来看一下执行的结果:

```
14:43:27.411 [pool-1-thread-6] INFO
14:43:27.411 [pool-1-thread-7]
                                          是交efur商品
                                                           refund success, itemId is
14:43:27.407 [pool-1-thread-3]
                                                           refund success.itemId is 2
14:43:27.407 [pool-1-thread-2]
                                                           refund success, itemId is 1
14:43:27.408 [pool-1-thread-4]
                                                           refund success, itemId is 3
14:43:27.408 [pool-1-thread-5] INFO de
14:43:27.407 [pool-
14:43:27.401 [main] INFO demo.sixth.BatchRefundDemo
                                                    30 个商品已经在退款中
14:43:27.421 [pool
            [pool主线程在这里就开始等待
                                                          retund success,itemId is 7
14:43:27.427
                                                      mo - refund success,itemId is 8
14:43:27.427 [pool-1-thread-10] INFO demo.sixth.RefundDemo - refund success,itemId is 9
14:43:27.443 [pool-1-thread-6] INFO demo.sixth.RefundDemo - refund success,itemId is 10
14:43:27.445 [pool-1-thread-2] INFO demo.sixth.RefundDemo - refund success,itemId is 12
14:43:27.445 [pool-1-thread-7] INFO demo.sixth.RefundDemo - refund success,itemId is 13
14:43:27.445 [pool-1-thread-4]
                             INFO demo.sixth.RefundDemo - refund success,itemId
                                                                                is 14
14:43:27.445 [pool-1-thread-5] INFO demo.sixth.RefundDemo - refund success,itemId is 15
14:43:27.445 [pool-1-thread-1] INFO demo.sixth.RefundDemo - refund success,itemId is 16
14:43:27.445 [pool-1-thread-3] INFO demo.sixth.RefundDemo - refund success,itemId is 11
14:43:27.454 [pool-1-thread-8] INFO demo.sixth.RefundDemo - refund success,itemId is 17
14:43:27.459 [pool-1-thread-10] INFO demo.sixth.RefundDemo - refund success,itemId is 19
14:43:27.459 [pool-1-thread-9] INFO demo.sixth.RefundDemo - refund success,itemId is 18
14:43:27.474 [pool-1-thread-6] INFO demo.sixth.RefundDemo - refund success,itemId is 20
14:43:27.477 [pool-1-thread-7] INFO demo.sixth.RefundDemo - refund success,itemId is 21
14:43:27.477 [pool-1-thread-1] INFO demo.sixth.RefundDemo - refund success,itemId is 25
14:43:27.478 [pool-1-thread-2] INFO demo.sixth.RefundDemo - refund success,itemId is 22
14:43:27.478 [pool-1-thread-5] INFO demo.sixth.RefundDemo - refund success,itemId is 24
14:43:27.477 [pool-1-thread-4] INFO demo.sixth.RefundDemo - refund success,itemId is 23
14:43:27.479 [pool-1-thread-3] INFO demo.sixth.RefundDemo - refund success,itemId is 26
14:43:27.487 [pool-1-thread-8] INFO demo.sixth.RefundDemo - refund success,itemId is 27
14:43子线程都执行完成 NFO 主线程继续执行
                                                           refund success, itemId is 28
14:43:27.490 [mair] INFO demo.sixth.BatchRefundDemo
14:43:27.677 [mair] INFO demo.sixth.BatchRefundDemo -
                                                    执行结果成功30.失败0
```

从执行的截图中,我们可以明显的看到 CountDownLatch 已经发挥出了作用,主线程会一直等 到 30 个商品的退款结果之后才会继续执行。

接着我们做了一个不严谨的实验(把以上代码执行很多次,求耗时平均值),通过以上代码, 30 个商品退款完成之后,整体耗时大概在 200 毫秒左右。

而通过 for 循环单商品进行退款,大概耗时在 1 秒左右,前后性能相差 5 倍左右,for 循环退款的代码如下:

```
long begin1 = System.currentTimeMillis();
for (Long item : items) {
    refundDemo.refundByItem(item);
}
log.info("for 循环单个退款耗时{}",System.currentTimeMillis()-begin1);
```

性能的巨大提升是线程池 + 锁两者结合的功劳。

慕课专栏

: ■ 面试官系统精讲Java源码及大厂真题 / 35 经验总结 : 各种锁在工作中使用场景和细节

目录

第1章 基础

01 开篇词: 为什么学习本专栏

02 String、Long 源码解析和面试题

本章举了实际工作中的两个小案列,看到了 CountDownLatch 和 synchronized (ReentrantLock) 是如何结合实际需求进行落地的,特别是 CountDownLatch 的案列,使用线程池 + 锁结合的方式大大提高了生产效率,所以在工作中如 果你也遇到相似的场景,可以毫不犹豫地用起来。

34 只求问倒:连环相扣系列锁面

36 从容不迫: 重写锁的设计结构

03 Java 常用关键字理解

04 Arrays、Collections、Objects 常 用方法源码解析

第2章 集合

05 ArrayList 源码解析和设计思路

06 LinkedList 源码解析

07 List 源码会问哪些面试题

08 HashMap 源码解析

09 TreeMap 和 LinkedHashMap 核心 源码解析

10 Map源码会问哪些面试题

11 HashSet、TreeSet 源码解析

12 彰显细节: 看集合源码对我们实际 工作的帮助和应用

13 差异对比: 集合在 Java 7 和 8 有何 不同和改进

14 简化工作: Guava Lists Maps 实际 工作运用和源码

第3章 并发集合类

15 CopyOnWriteArrayList 源码解析 和设计思路

16 ConcurrentHashMap 源码解析和 设计思路

17 并发 List、Map源码面试题

18 场景集合:并发 List、Map的应用

精选留言 1

欢迎在这里发表留言,作者筛选后可公开显示

敲木鱼的小和尚

老师,可以多搞点工作总结吗,AQS看的真心爽

6 0 回复 2019-12-05

干学不如一看,干看不如一练