Java 并发与多线程实验报告

2025 年 4 月 23 日

一、设计思路：

1. 创建一个商店管理线程，管理其他线程，设置为守护线程，这样在主线程退出后商店管理线程就会自动退出，以此来实现特定时间后终止程序

2. 进货线程：每种售卖唱片都有一条自己的进货线程，该线程以该种售卖唱片对象为锁，调用该对象的 wait 方法以此来达到每秒执行一次的效果，而购买线程在遇到储存量不足的情况就会使用该售卖唱片的 notified 方法来唤醒进货线程补货

3. 售卖线程：调用 soldcd 的 sold 方法，判断是否有足够存量 cd 可以售卖，若没有，则用随机数决定是否等待，若不等待则睡眠一段时间后再次购买，若等待，则唤醒进货线程并且 wait，需要注意的是，若将整个 run 方法基本完全使用 synchronized 修饰，会导致如果有多个线程同时要购买缺货的 cd 的时候会被阻塞在 wait 方法外面，但是补货线程的存在很好地解决了这个问题，一旦有线程购买缺货的 cd 并且选择等待，补货线程就会启动，很快就会使 cd 的状态变为满货状态，所以出现如上所述的阻塞在 wait 方法外的可能性极低

4. 租借线程：与售卖线程的实现方法类似，但比售卖线程的情况复杂。在 rentcd 类中有一个 bool 类型变量判定是否借出，需要注意的是，该变量需要声明为 volatile，因为线程会频繁访问到该变量，而该变量又极其易变，且改变该变量状态的函数需要使用 synchronized 修饰，以防被多个线程改变

若将程序主体声明为同步（我源码中注释的 synchronized 位置），则会导致一个线程未归还 cd 前其他想要租借该 cd 的线程直接被阻隔在程序外面，根本不会执行到若 cd 已租借出去的逻辑，所以只将带有 wait 和 notified 的地方声明为同步，但这会导致以下问题

租借线程相比售卖线程复杂的地方是，若租借线程想要租借已经出租的 cd，该 cd 不会立刻被归还，这就导致线程被阻隔在 wait 函数外面的可能性大幅度增加，同时，判断 rentcd 是否已被出租的功能也不能直接在租借线程的 run 方法里用一个 if 语句判断，因为很可能在没有修改状态的时候就会有另一个线程闯入，这样很可能导致有两个线程租到同一个 cd，故需要单独写一个判断是否租借出去的状态函数，声明为同步函数

经过各方面的查询，最终发现 wait 函数会自动释放锁资源，故线程被阻隔在 wait 函数外的问题不会发生

5. 日志记录

更改 log.dir 值为 thread/src/log 使日志记录在 log 目录下，对每种过滤等级均设置自己的 appender，配置单独的过滤器，保证 info 文件中记录所有 info 记录，error 文件中记录所有 error 记录，在主函数中所有的 catch 语句中会执行进行 error 日志处理的代码，正常运行到最后会进行 info 日志的处理，源代码中会给出配置的完整 xml 文件

二. 心得与收获：

1. 使用守护线程可以很容易地控制程序在什么时候停止，守护线程是为其他线程提供服务的一类线程，在其他线程结束后一段时间会自动终止

2. 使用日志系统可以很方便地记录程序的运行情况，遇到的错误等信息，可以在测试和运维阶段提供良好的参考

3. 对于那些容易被多个线程访问及改变的变量需要用 volatile 修饰以使每次访问到的值都是最新的

4. 对于那些容易被多个线程调用的方法或代码块，要使用 synchronized 修饰以保证同一时间内只有一个线程执行对应方法，以此来保证程序进行的可靠性

5.wait-notified 机制可以使线程调用更加灵活和高效，在消费者仍有数据可用的情况下生产者可以不生产数据，将这部分资源释放出来做其他事，当消费者无数据可用的时候也可以迅速唤醒生产者线程来生产数据，待生产结束后消费者也可以立刻进行处理数据，该机制在生产者和消费者之间建立了一条高效可靠的通信通道

6. 使用同步代码块的位置一定要特别注意，因为一个线程拿到锁之后其他线程就不会进入该同步代码块了，故如果有些逻辑不想处理为未拿到锁的线程无法进入的话这部分逻辑不应该被纳入同步代码块的范围

三. 源代码

```java
package org.example.cdshop;

public class CD {

  static int id=0;
  public String ISBN;
  public String cdName;
  public CD() {
    ISBN="ISBN"+id;
    cdName="CD"+id;
    id++;

  }
  public CD(Integer id)
  {
    ISBN="ISBN"+id;
    cdName="CD"+id;
  }
}
```

```java
package org.example.cdshop;

import java.util.Vector;

public class CDShop {

  SoldCD[] soldcds=new SoldCD[10];
  Vector<RentCD> rentcds;
  CDShop() {
    rentcds=new Vector<>();
    for(int i=0;i<10;i++) {
      soldcds[i]=new SoldCD();
      rentcds.add(new RentCD());
    }

  }
  public static void main(String[] arg) throws
      InterruptedException {
    System.out.println("你好");
```

```
    new CDshopControlThread(new CDShop()).start();
    Thread.sleep(1000*120);
  }
}
```

```
package org.example.cdshop;

import lombok.extern.slf4j.Slf4j;

import java.util.Random;

@Slf4j
public class CDshopControlThread extends Thread{

  CDShop shop;

  public CDshopControlThread(CDShop shop) {
    super();
    this.shop = shop;
    this.setDaemon(true);
  }
  @Override
  public void run() {
    for(SoldCD cd:shop.soldcds) {
      new GetInThread(cd).start();
    }
    Random rand = new Random();
    int soldthreadcount=rand.nextInt(2,10);
    log.info("售卖和租借线程各有：{}个",soldthreadcount);
    for(int i=0;i<soldthreadcount;i++) {
      new SoldThread(shop.soldcds).start();
    }
    for(int i=0;i<soldthreadcount;i++) {
      new RentThread(shop.rentcds).start();
    }
    }

}
```

```java
package org.example.cdshop;

public class GetInThread extends Thread {

  SoldCD cd;
  GetInThread(SoldCD cd){
    this.cd=cd;
    this.setName(this.getName()+cd.cdName+"进货线程");
  }
  @Override
  public void run() {
    try {
      while(true) {
    synchronized(cd) {
      cd.wait(1000);
      cd.getin();
    }
      }
    }
    catch(Exception e) {

    }
  }
}
```

```java
package org.example.cdshop;

import java.util.ArrayList;

public class RentCD extends CD {
    static int cdNum=1;
    public volatile boolean ifRented = false;
    public RentCD() {
        super(cdNum++);
    }
    synchronized public void changeRented() {
        ifRented=!ifRented;
    }
```

```
    synchronized public boolean tryRent() {
        if(!ifRented) {
            ifRented=true;
            return true;
        }
        return false;
    }
}
```

```
package org.example.cdshop;


import lombok.extern.slf4j.Slf4j;

import java.util.Date;
import java.util.Random;
import java.util.Vector;

@Slf4j
public class RentThread extends Thread {
    private Vector<RentCD> rentCD;
    public RentThread(Vector<RentCD> rentCD) {
        super();
        this.rentCD = rentCD;
        this.setName(this.getName()+"租借线程");
    }
    @Override
    public void run() {
        //Random rand;
        while (true) {
            Random rand = new Random();
            int rentId = rand.nextInt(1, 10);
            RentCD wantedCd = rentCD.get(rentId);
            //synchronized (wantedCd){
            while (true) {
                if (!wantedCd.tryRent()) {
                    boolean ifwait = rand.nextBoolean();
                    if (ifwait) {
                        int waitTime = rand.nextInt(200);
```

```
                    log.info(new Date() + wantedCd.cdName +
                         "已借出" + Thread.currentThread().
                        getName() + "等待中");
                    synchronized (wantedCd){
                        try {
                            wantedCd.wait(waitTime);
                        } catch (InterruptedException e) {
                            throw new RuntimeException(e);
                        }
                    }
                }
                else {
                    log.info(new Date() + wantedCd.cdName +
                         "已借出" + Thread.currentThread().
                        getName() + "放弃");
                    break;
                }
            }

            else {
                int sleepTime = rand.nextInt(200, 300);
                log.info(new Date() + Thread.currentThread
                    ().getName() + "租到了cd: " + wantedCd.
                    cdName + "持有" + sleepTime + "ms");
                try {
                    sleep(sleepTime);
                } catch (InterruptedException e) {
                    throw new RuntimeException(e);
                }
                log.info(new Date() + Thread.currentThread
                    ().getName() + "归还cd: " + wantedCd.
                    cdName);
                synchronized (wantedCd){
                    wantedCd.notify();
                    wantedCd.changeRented();
                }
                break;
            }
        }
```

```
            int sleepTime = rand.nextInt(200);
            log.info(
                    new Date() + Thread.currentThread().getName
                        () + "睡眠时间" + sleepTime);
            try {
                this.sleep(sleepTime);
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        }

        //}
    }
}
```

```
package org.example.cdshop;

import lombok.extern.slf4j.Slf4j;

import java.util.Date;
@Slf4j
public class SoldCD extends CD {

  public int count;
  public SoldCD() {
    count=10;
  }
  synchronized public boolean sold(int num) {
    if(count>=num) {
      count-=num;
      log.info(
          new Date()+
          Thread.currentThread().getName()
          +this.cdName
          +"销售了"+num
          +"剩余"+count);
      return true;
    }
    else
```

```
    {
      log.info(new Date()+
      Thread.currentThread().getName()
      +this.cdName
      +"数量不足");
      return false;
    }

  }
  synchronized public void getin() {
    count=10;

    log.info(
        new Date()+
        Thread.currentThread().getName()
        +this.cdName
        +"进货");

  }
}
```

```
package org.example.cdshop;

import lombok.extern.slf4j.Slf4j;

import java.util.Date;
import java.util.Random;
@Slf4j
public class SoldThread extends Thread {

  SoldCD[] cds;

  public SoldThread(SoldCD[] cds) {
    super();
    this.cds = cds;
    this.setName(this.getName()+"销售线程");
  }

  @Override
```

```
19  public void run() {
20    while (true) {
21      Random r = new Random();
22      int index = r.nextInt(cds.length);
23      SoldCD cd = cds[index];
24      int num = r.nextInt(6);
25      try {
26        while (true) {
27          boolean solded = cd.sold(num);
28          if (solded) {
29            break;
30          } else {
31            if (r.nextBoolean()) {
32              log.info(
33                  new Date() + Thread.currentThread().getName()
                        + cd.cdName + "数量不足, 不销售");
34              break;
35            } else {
36              log.info(
37                  new Date() + Thread.currentThread().getName()
                        + cd.cdName + "数量不足, 继续等候,唤醒进
                        货线程");
38              synchronized (cd) {
39                cd.notifyAll();
40                cd.wait(r.nextInt(200)
41                );
42              }
43            }
44          }
45
46        }
47        int sleep=r.nextInt(200);
48        log.info(
49            new Date() + Thread.currentThread().getName() +"睡
                眠时间"+sleep);
50        this.sleep(sleep);
51      } catch (Exception e) {
52
53      }
```

```
    }
  }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <!--格式化输出
        %d表示日期，后跟日期格式    %thread表示线程名
        %-5level: 级别从左显示5个字符宽度
        %msg: 日志消息    %n是换行符
        %logger{36}: logger名称，最多显示36个字符
        %highlight、%green、%boldGreen: 高亮显示、绿色显示、加粗绿色显示
    -->
    <property name="log.dir" value="./thread/src/log"/>
    <!-- CONSOLE 控制台日志 -->
    <appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>%date{yyyy-MM-dd HH:mm:ss.SSS} %highlight(%-5level) [%green(%thread)] %boldGreen(%logger{36}) - %highlight(%msg%n)
            </pattern>
        </encoder>
    </appender>

    <!-- File是输出的方向通向文件的 -->
    <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <filter class="ch.qos.logback.classic.filter.ThresholdFilter">
            <level>INFO</level>
        </filter>
        <encoder>
            <pattern>%date{yyyy-MM-dd HH:mm:ss.SSS} %-5level [%thread] %logger{36} - %msg%n</pattern>
            <charset>utf-8</charset>
        </encoder>
        <!--日志输出路径-->
```

```
29          <!--指定日志文件拆分和压缩规则-->
30          <rollingPolicy
31                  class="ch.qos.logback.core.rolling.
                        SizeAndTimeBasedRollingPolicy">
32              <!--通过指定压缩文件名称，来确定分割文件方式-->
33              <fileNamePattern>${log.dir}/log.%d{yyyy-MM-dd}.%i.
                    log</fileNamePattern>
34              <!--文件拆分大小-->
35              <maxFileSize>4MB</maxFileSize>
36          </rollingPolicy>
37      </appender>
38
39      <!--
40      level:用来设置打印级别，大小写无关：TRACE，DEBUG，INFO，
            WARN，ERROR，ALL 和 OFF，默认debug
41      <root>可以包含零个或多个<appender-ref>元素，标识这个输出位
            置将会被本日志级别控制。
42      -->
43      <root>
44          <appender-ref ref="CONSOLE"/>
45          <appender-ref ref="FILE"/>
46      </root>
47  </configuration>
```