班级： 软工11601 姓名： 赵惹儿 序号： 31

上机2 多线程与定时器

**一、上机目的**

1、通过疯狂泳池管理员的例子掌握定时器的使用；

2、通过疯狂泳池管理员的例子深入理解多线程的概念；

3、最后，初步了解线程池管理工具；

**二、上机准备**

软件：Ecplise；

1. **上机内容**

**程序描述：**疯狂的泳池管理员

(1)泳池的储水量是一万立方米

(2)泳池每晚八点换一次水

(3)进水速度为5立方米每秒，出水速度为3立方米每秒，问多久能把泳池续满

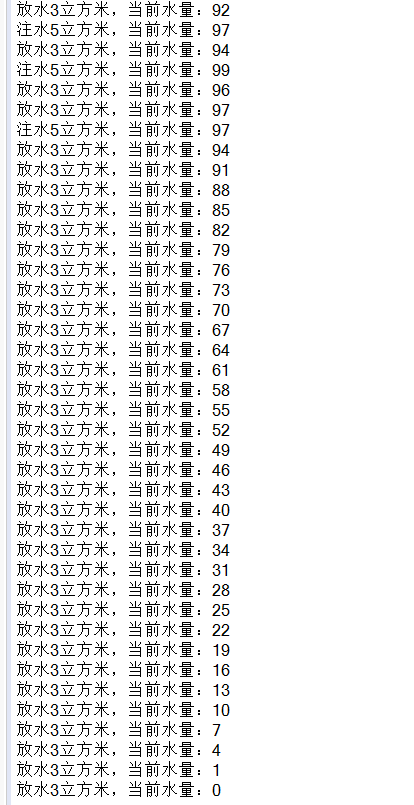
**分析：**

1. 泳池储水量可以用阻塞队列的长度来模拟，使用阻塞队列的意义在于，当水池已满，不能进水；水池为空，不能放水。正是典型的生产者与消费者的模型。
2. 定时换水可用定时器实现。
3. 一边进水，一边放水可用多线程来模拟，两个线程并发。

具体程序见附录

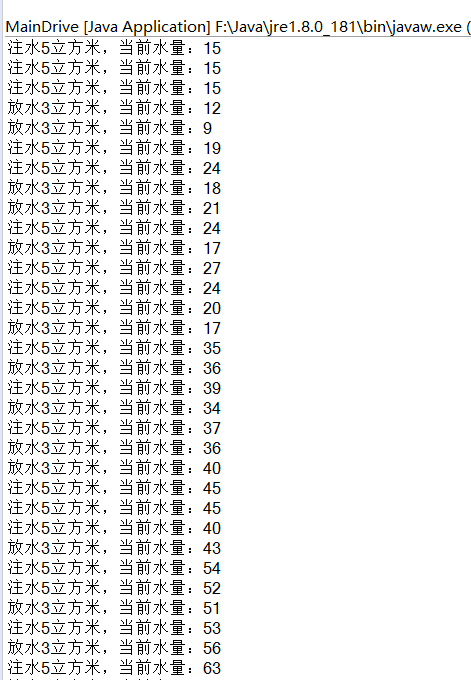
1. **上机结果**

结果1：有时会出现注满水，又把水全放光了的结果。



分析：注水线程停了，所以放水线程一直放水，直到该水池为空。

结果2：结果中的前三行，刚开始水量都是15，显然不合理。



分析：对同一数据进行操作，没有实现线程同步。在run()方法中加入线程同步即可解决。

对于把注水线程与放水线程的休眠时间改掉，把休眠1ms和休眠10s的结果对比来看，除了后者等待的时间长点，其他没有什么变化。这点有待深究。

**五、上机小结**

**1.多线程实现**

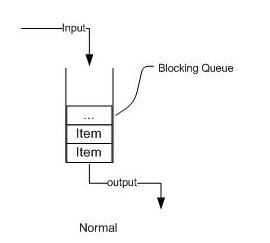
Java中的多线程有三种实现方式，本程序是实现Callable接口，重写call方法。Callable接口与Runnable接口的功能类似，但提供了比Runnable更强大的功能。主要有以下三点:

（1）Callable可以在任务结束后提供一个返回值，Runnable没有提供这个功能。

（2）Callable中的call方法可以抛出异常，而Runnable的run方法不能抛出异常。

（3）运行Callable可以拿到一个Future对象，表示异步计算的结果，提供了检查计算是否完成的方法。

**2.对阻塞队列的认识**



从上图我们可以很清楚看到，通过一个共享的队列，可以使得数据由队列的一端输入，从另外一端输出。

在阻塞队列中，线程阻塞有这样的两种情况：

1.当阻塞队列为空时，获取队列元素的线程将等待，直到该则阻塞队列非空；

2.当阻塞队列变满时，使用该阻塞队列的线程会等待，直到该阻塞队列变成非满。

**BlockingQueue的核心方法：**

1.放入数据

（1）offer(anObject):表示如果可能的话,将anObject加到BlockingQueue里,即如果BlockingQueue可以容纳,则返回true,否则返回false.（本方法不阻塞当前执行方法的线程）。

（2）offer(E o, long timeout, TimeUnit unit)：可以设定等待的时间，如果在指定的时间内，还不能往队列中加入BlockingQueue，则返回失败。

（3）put(anObject):把anObject加到BlockingQueue里,如果BlockQueue没有空间,则调用此方法的线程被阻断直到BlockingQueue里面有空间再继续。

2. 获取数据

（1）poll(time):取走BlockingQueue里排在首位的对象,若不能立即取出,则可以等time参数规定的时间,取不到时返回null;

（2）poll(long timeout, TimeUnit unit)：从BlockingQueue取出一个队首的对象，如果在指定时间内，队列一旦有数据可取，则立即返回队列中的数据。否则直到时间超时还没有数据可取，返回失败。

（3）take():取走BlockingQueue里排在首位的对象,若BlockingQueue为空,阻断进入等待状态直到BlockingQueue有新的数据被加入;

**3.定时器**

在java中一个完整定时任务需要由Timer、TimerTask两个类来配合完成。可以这样理解Timer是一种定时器工具，用来在一个后台线程计划执行指定任务，而TimerTask一个抽象类，它的子类代表一个可以被Timer计划执行的任务。

**Timer类**

在工具类Timer中，提供了四个构造方法，每个构造方法都启动了计时器线程，同时Timer类可以保证多个线程可以共享单个Timer对象而无需进行外部同步，所以Timer类是线程安全的。

当程序初始化完成Timer后，定时任务就会按照我们设定的时间去执行，Timer提供了schedule方法，该方法有多种重载方式来适应不同的情况，如下：

schedule(TimerTask task, Date time)：安排在指定的时间执行指定的任务。

schedule(TimerTask task, Date firstTime, long period) ：安排指定的任务在指定的时间开始进行重复的固定延迟执行。

schedule(TimerTask task, long delay) ：安排在指定延迟后执行指定的任务。

schedule(TimerTask task, long delay, long period) ：安排指定的任务从指定的延迟后开始进行重复的固定延迟执行。

同时也重载了scheduleAtFixedRate方法，scheduleAtFixedRate方法与schedule相同，只不过他们的侧重点不同。

scheduleAtFixedRate(TimerTask task, Date firstTime, long period)：安排指定的任务在指定的时间开始进行重复的固定速率执行。

scheduleAtFixedRate(TimerTask task, long delay, long period)：安排指定的任务在指定的延迟后开始进行重复的固定速率执行。

scheduleAtFixedRate与schedule方法的侧重点不同：schedule方法侧重保存间隔时间的稳定，而scheduleAtFixedRate方法更加侧重于保持执行频率的稳定。在schedule方法中会因为前一个任务的延迟而导致其后面的定时任务延时，而scheduleAtFixedRate方法则不会，如果第n个task执行时间过长导致systemCurrentTime>= scheduledExecutionTime(n+1)，则不会做任何等待，他会立即执行第n+1个task，所以scheduleAtFixedRate方法执行时间的计算方法不同于schedule，而是scheduledExecutionTime(n)=firstExecuteTime +n\*periodTime，该计算方法永远保持不变。所以scheduleAtFixedRate更加侧重于保持执行频率的稳定。

**TimerTask类**

TimerTask类是一个抽象类，由Timer 安排为一次执行或重复执行的任务。它有一个抽象方法run()方法，该方法用于执行相应计时器任务要执行的操作。因此每一个具体的任务类都必须继承TimerTask，然后重写run()方法。

另外它还有两个非抽象的方法：

boolean cancel()：取消此计时器任务。

long scheduledExecutionTime()：返回此任务最近实际执行的安排执行时间。

**4.对线程池的理解**

程序中缺少ThreadUtil类，它是一个线程管理工具类。借鉴网上的代码，深入学习了线程池的具体使用和线程管理工具的实现思路。

线程池有四种类型，分别是以下四种：

（1）newCachedThreadPool创建一个可缓存线程池，如果线程池长度超过处理需要，可灵活回收空闲线程，若无可回收，则新建线程。

（2）newFixedThreadPool 创建一个定长线程池，可控制线程最大并发数，超出的线程会在队列中等待。

（3）newScheduledThreadPool 创建一个定长线程池，支持定时及周期性任务执行。

（4）newSingleThreadExecutor 创建一个单线程化的线程池，它只会用唯一的工作线程来执行任务，保证所有任务按照指定顺序(FIFO, LIFO, 优先级)执行。

一般情形下，使用最多的类型是第一种，使用方法具体如下：

//1.初始化线程

ExcutorService exec = Excutors.newCachedThreadPool();

//2.添加线程并返回该线程的Future

Future Future = exec.submit(Runnable);

//3.批量添加线程并返回一个Future集合

List<Future> futureList = exec.invokeAll(List);

**注：**

ExecutorService是JDK并发工具包提供的一个核心接口，相当于一个线程池，提供执行任务和管理生命周期的方法。

#### Future模式：对于多线程而言，如果线程A要等待线程B的结果，那么线程A没必要一直等待B，直到B有结果，可以先拿到一个未来的Future，等B有结果时再取真实的结果值。

submit()方法是在ExecutorService中声明的方法，在AbstractExecutorService就已经有了具体的实现，在ThreadPoolExecutor中并没有对其进行重写，这个方法也是用来向线程池提交任务的，但是它和execute()方法不同，它能够返回任务执行的结果，去看submit()方法的实现，会发现它实际上还是调用的execute()方法，只不过它利用了Future来获取任务执行结果。

ExecutorService中定义了两个批量执行任务的方法，invokeAll()和invokeAny()，在批量执行或多选一的业务场景中非常方便。invokeAll()在所有任务都完成（包括成功/被中断/超时）后才会返回，invokeAny()在任意一个任务成功（或ExecutorService被中断/超时）后就会返回。

**使用线程池的好处:**  
    a.减少在创建和销毁线程上所花的时间以及系统资源的开销   
    b.如不使用线程池，有可能造成系统创建大量线程而导致消耗完系统内存以及”过度切换”。

**总结:**通过疯狂泳池管理员的例子，对多线程有了更深的理解，把课上讲的知识真正运用起来，同时学会使用了一些具体的函数。因为没有学过操作系统原理，对多线程的实现原理不是很清楚，等看完操作系统的进程管理，再来对报告做个补充。