## 疯狂的游泳池管理员-----生产者与消费者模式

此次案例利用生产者消费者模式，主要用到了Executor框架，线程池管理线程的问题，以及定时器的使用，利用多线程占用一个队列。

1. **任务的产生：Runnable接口和Callable接口**

OutputThread，InputThread即注水与排水任务

这两个对象都是任务对象，工具类Execuor可以将Runnale产生的对象封装成Callable对象。当我们有了以上两个任务对象后，我们就交给ExecutorService（它的一个实例对象）**private** **static** ExecutorService *exec*然后进入第二阶段任务处理。

1. **任务的处理：Executor接口—>ExecutorService接口**

任务的处理就是将任务丢进线程池中，由线程池提供线程将任务消费掉。

ExecutorService提供2种线程池，TheardPoolExecutor和ServiceTheardPoolExecutor,

两种线程池都由工厂类Executors来创建

*exec* = Executors.*newCachedThreadPool*();//利用工厂类创建ExecutorService的**实例**

**⑴：ThreadPoolExecutor类**

工厂类可以创建3种类型的ThreadPoolExecutor类：

①：FixedThreadPool：拥有固定数量线程的线程池，限制了线程的数目，适用于负载比较重的服务器。

②：SingleThreadPool：单个线程的线程池，适用于需要保证顺序的执行各个任务；任意时间点，不会有多个线程活动。

③：CachedThreadPool：大小无界的线程池，适用于执行很多的短期异步任务的小程序，或者是负载较轻的服务器。

**⑵：ScheduleThreadPoolExecutor类**

工厂类可以创建2种类型的SchedulePoolExecutor类：

①：ScheduleThreadPoolExecutor：包含若干线程。

②：SingleThreadScheduleExecutor:单个线程。

这里比较符合要求的是CachedThreadPool

**3，任务结果的获取：Future接口**

**private** **static** List<Object> getAllCallableReturn(List<Future<Object>> futureList) **throws** ExecutionException, InterruptedException

轮询获取所有Callable线程的返回值，直到所有返回值都被获取到（阻塞）

futureList所有线程的管理类集合

public static void runCheckRunnable(List<Runnable> RunnableList,boolean isBlock) throws InterruptedException

线程池管理工具 传入一组不带有返回值的线程，然后由工具类统一管理，对当前的线程进行监听

**Parameters:**

**RunnableList** 线程集合

**isBlock** 是否阻塞标示位，如果为false，则线程启动后立即返回，否则等待所有线程均有结果后返回（推荐使用true）

**Returns:**

线程运行结果集合，仅返回已运行结束的线程集合，如都未结束，则返回尺寸为0的list

看个图：Executor主要由三部分组成：任务产生部分，任务处理部分，结果获取部分。（设计模式：生产者与消费者模式）

