线程池介绍及简单实现

参考https://www.cnblogs.com/kuoAT/p/6714762.html

一、问题引出

我们使用线程的时候就去创建一个线程，这样实现起来非常简便，但是就会有一个问题：

如果并发的线程数量很多，并且每个线程都是执行一个时间很短的任务就结束了，这样频繁创建线程就会大大降低系统的效率，因为频繁创建线程和销毁线程需要时间。

那么有没有一种办法使得线程可以复用，就是执行完一个任务，并不被销毁，而是可以继续执行其他的任务?

二、线程池优点

合理利用线程池能够带来三个好处。

- 降低资源消耗。通过重复利用已创建的线程降低线程创建和销毁造成的消耗。
- 提高响应速度。当任务到达时，任务可以不需要等到线程创建就能立即执行。
- 提高线程的可管理性。线程是稀缺资源，如果无限制的创建，不仅会消耗系统资源，还会降低系统的稳定性，使用线程池可以进行统一的分配，调优和监控。

三、常见四种线程池：

1、CachedThreadPool()：可缓存线程池

2、FixedThreadPool： 定长线程池

3、SingleThreadPool

4、ScheduledThreadPool

四、线程池简单实现

1、实现接口

接口： java.util.concurrent.**ExecutorService**

Java里面线程池的顶级接口是 java.util.concurrent.Executor ，但是严格意义上讲 Executor 并不是一个线程池，而只是一个执行线程的工具。

类：java.util.concurrent.**Executors**

要配置一个线程池是比较复杂的，尤其是对于线程池的原理不是很清楚的情况下，很有可能配置的线程池不是较优的，因 线程工厂类里面提供了一些静态工厂，生成一些常用的线程池。

2、官方建议使用Executors工程类来创建线程池对象。

Executors类中有个创建线程池的方法如下：

public static ExecutorService newFixedThreadPool(int nThreads) ：返回线程池对象。(创建的是有界线程池,也就是池中的线程个数可以指定最大数量)

获取到了一个线程池ExecutorService 对象，那么怎么使用呢，在这里定义了一个使用线程池对象的方法如下：

public Future<?> submit(Runnable task) ： 获取线程池中的某一个线程对象，并执行

Future接口： 用来记录线程任务执行完毕后产生的结果。线程池创建与使用。

3、使用线程池中线程对象的步骤：

1. 创建线程池对象。

2. 创建Runnable接口子类对象。(task)

3. 提交Runnable接口子类对象。(take task)

4. 关闭线程池(一般不做)

4、代码实现

线程池测试类

```
public class ThreadPoolDemo {
    public static void main(String[] args) {
        // 创建线程池对象
        ExecutorService service = Executors.newFixedThreadPool(2);//包含2个线程对象
        // 创建Runnable实例对象
        MyRunnable r = new MyRunnable();
        //自己创建线程对象的方式
```

```
        // Thread t = new Thread(r);
        // t.start(); ---> 调用MyRunnable中的run()
        // 从线程池中获取线程对象,然后调用MyRunnable中的run()
        service.submit(r);
        // 再获取个线程对象，调用MyRunnable中的run()
        service.submit(r);
        service.submit(r);
        // 注意：submit方法调用结束后，程序并不终止，是因为线程池控制了线程的关闭。
        // 将使用完的线程又归还到了线程池中
        // 关闭线程池
        //service.shutdown();
    }
}
```

Runnable实现类代码：

```
public class MyRunnable implements Runnable {
    @Override
    public void run() {
        System.out.println("我要一个教练");
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("教练来了： " + Thread.currentThread().getName());
        System.out.println("教我游泳,交完后，教练回到了游泳池");
    }
```