

JDK1.5之引入了了一个关于并发访问的包。java.util.concurrent，对于线程池操作的类和相关接口就定义在此包中，这里面有两个核心的接口：

- 关于普通线程池：public interface ExecutorService extends Executor
- 关于调度线程池：public interface ScheduledExecutorService extends ExecutorService

线程池的创建

进行线程池的创建，一般可以使用Executors类完成，有如下几种方法：


- 创建无大小限制的线程池：public static ExecutorService newCachedThreadPool()
- 创建固定大小的线程池：public static ExecutorService newFixedThreadPool(int nThreads)
- 单线程池：public static ExecutorService newSingleThreadExecutor()
- 创建定时调度池：public static ScheduledExecutorService newScheduledThreadPool(int corePoolSize)

1. 创建无大小限制的线程池

```
public class Test {
    public static void main(String[] args) {
        // 创建一个无大小限制的线程池
        ExecutorService newCachedThreadPool = Executors.newCachedThreadPool();

        for(int i = 0; i < 10; i++) {
            int index = i;
            // 将任务加入到线程池中
            newCachedThreadPool.submit(new Runnable() {

                @Override
                public void run() {
                    System.out.println(Thread.currentThread().getName()+" ,i = " +index);
                }
            });
        }
        // 关闭线程池，与创建线程池为对称操作
        newCachedThreadPool.shutdown();
    }
}
```

运行结果：

由运行结果我们可以得到如下结论：

- 线程池中的线程启动顺序也是随机的。
- 无限大小的线程池为每个加入的任务创建一个线程并去执行。

我再来看一下 submit() 方法。

```
// 参数为一个 Runnable 接口对象
submit(Runnable task);
```


所以我们可以将继承了Runnable接口的线程任务或例子中的匿名Runnable类传入线程池，即该线程任务加入到线程池中。

2. 创建单线程的线程池

```
public class Test {
    public static void main(String[] args) {
        // 创建一个单线程的线程池
        ExecutorService newSingleThreadExecutor =
            Executors.newSingleThreadExecutor();

        for(int i = 0; i < 10; i++) {
            int index = i;
            // 将任务添加到线程池中
            newSingleThreadExecutor.submit(new Runnable() {

                @Override
                public void run() {
                    System.out.println(Thread.currentThread().getName()+" ,i = " +index);
                }
            });
        }
        // 关闭线程池
        newSingleThreadExecutor.shutdown();
    }
}
```

运行结果: 

单线程的线程池，多个线程任务由这一个线程完成。

3. 创建固定大小的线程池

```
public class Test {
    public static void main(String[] args) {
        // 创建拥有三个线程的线程池
        ExecutorService newFixedThreadPool = Executors.newFixedThreadPool(3);

        for(int i = 0; i < 10; i++) {
            int index = i;
            // 将任务加入到线程池中
            newFixedThreadPool.submit(new Runnable() {

                @Override
                public void run() {
                    System.out.println(Thread.currentThread().getName()+" ,i = " +index);
                }
            });
        }
        // 关闭线程池
    }
}
```

```

        newFixedThreadPool.shutdown();
    }
}

```

运行结果: 

多个线程任务由线程池中随机线程去执行。

4. 创建定时调度池

```

public class Test {
    public static void main(String[] args) {
        // 创建拥有三个线程的定时调度池
        ScheduledExecutorService newScheduledThreadPool = Executors.newScheduledThreadPool(3);

        for(int i = 0; i < 10; i++) {
            int index = i;
            // 将线程任务添加到定时调度池中, 0秒后启动线程, 每隔两秒在启动一次, 时间单位为秒
            newScheduledThreadPool.scheduleAtFixedRate(new Runnable() {

                @Override
                public void run() {
                    // TODO Auto-generated method stub
                    System.out.println(Thread.currentThread().getName()+" ,i = " +index);
                }
            }, 0, 2, TimeUnit.SECONDS);
            Thread.sleep(2000);
        }
        // 定时调度池没有关闭的需求
    }
}

```

我们看一下 scheduleAtFixedRate() 方法。

```

// command: 线程对象
// initialDelay: 加入的线程开始执行时间
// period: 每多长时间再执行该线程
// unit: 时间单位
public ScheduledFuture<?> scheduleAtFixedRate(Runnable command,
                                                long initialDelay,
                                                long period,
                                                TimeUnit unit);

```

点开 TimeUnit 的定义, 可以看到: 

TimeUnit为一个枚举类, 里面定义了各种时间单位。

线程池带来的好处是, 多个线程按照组的模式进行程序的处理。这样在某些业务逻辑复杂的环境下, 性能就会得到很好的提升。