链滴

Callable 和常用辅助类（Juc-04）

作者：yscxy

原文链接：https://ld246.com/article/1605944814005

来源网站：链滴

简介

```
@FunctionalInterface
public interface Callable<V>
```

返回结果的一个任务，并可能抛出异常。用户定义一个不带参数调用 call

的Callable接口类似于Runnable，，是专为其实例的类可能被另一个线程执行。然而，Runnable，不返回结果并不能抛出异常。

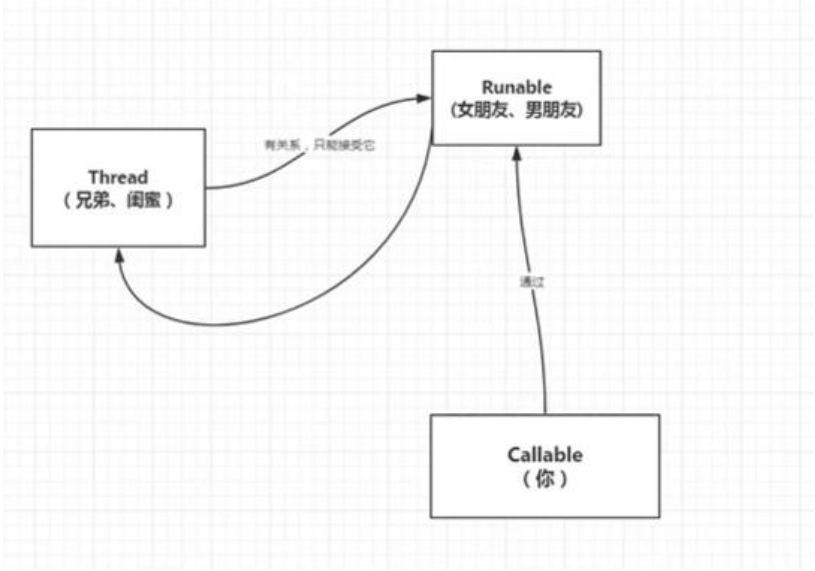
Executors类包含的实用方法与其他常见的形式转换为Callable类。

1.可以有返回值

2.可以抛出异常

3.方法不同run（）/call()

代码测试




```
package net.yscxy.callable;

import java.util.concurrent.Callable;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.FutureTask;

/**
 * @Author WangFuKun
 * @create 2020/11/21 10:27
 */
public class CallableTest {
    public static void main(String[] args) throws ExecutionException, InterruptedException {
/*      new Thread(new Runnable()).start();
        new Thread(new FutureTask<V>());
        new Thread(new FutureTask<V>(callbale)).start();
*/
        MyThread myThread = new MyThread();
        FutureTask futureTask = new FutureTask(myThread);
        //适配类
        new Thread(futureTask, "A").start();
        //返回结果打印
        System.out.println(futureTask.get());
    }
}

class MyThread implements Callable<String> {

    @Override
    public String call() throws Exception {
        System.out.println("call( )");
        return "123456";
    }
}
```

细节问题：

1. 有缓存

2. 结果可能需要等待，会阻塞

```
package net.yscxy.callable;

import java.util.concurrent.Callable;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.FutureTask;
import java.util.concurrent.TimeUnit;

/**
 * @Author WangFuKun
 * @create 2020/11/21 10:27
 */
public class CallableTest {
    public static void main(String[] args) throws ExecutionException, InterruptedException {
/*      new Thread(new Runnable()).start();
        new Thread(new FutureTask<V>());
        new Thread(new FutureTask<V>(callbale)).start();
*/
        MyThread myThread = new MyThread();
        FutureTask futureTask = new FutureTask(myThread);
        //适配类
        new Thread(futureTask, "A").start();
        new Thread(futureTask, "B").start();//结果会被缓存，效率高
        //返回结果打印
        System.out.println(futureTask.get());
        System.out.println(futureTask.get());
    }
}

class MyThread implements Callable<String> {

    @Override
    public String call() throws Exception {
        TimeUnit.SECONDS.sleep(2);
        System.out.println("call( )");
        return "123456";
    }
}
```

常用的辅助类

1.CountDownLatch

```
package net.yscxy.add;

import java.util.concurrent.CountDownLatch;
import java.util.concurrent.TimeUnit;

/**
 * @Author WangFuKun
 * @create 2020/11/21 15:01
 */
```

```
public class CountDownLatchDemo {
    public static void main(String[] args) throws InterruptedException {
        //总数是6
        CountDownLatch downLatch = new CountDownLatch(6);
        for (int i = 1; i <= 6; i++) {
            new Thread(() -> {
                System.out.println(Thread.currentThread().getName()+"-->Go Out");
                downLatch.countDown();
            }, String.valueOf(i)).start();
        }
        System.out.println("结束了！");
        downLatch.await();//等待计数器归零，然后再向下执行
        System.out.println("这才是真正结束");
    }
}
```

原理

1. 数量减一

2. 等待计数器归零

每次有线程调用countDown数量减一，假设计数器变为0 countDownLatch.await()就会被唤醒

继续执行。

2.CyclicBarrier

```
package net.yscxy.add;

import java.util.concurrent.BrokenBarrierException;
import java.util.concurrent.CyclicBarrier;

/**
 * @Author WangFuKun
 * @create 2020/11/21 15:21
 */
/*
 * 集齐7颗龙珠召唤神龙
 * */
public class CyclicBarrierDemo {
    public static void main(String[] args) {
        CyclicBarrier cyclicBarrier = new CyclicBarrier(7, () -> {
            System.out.println("召唤神龙成功！");
        });
        for (int i = 1; i <= 7; i++) {
            final int tmp = i;
            new Thread(() -> {
                System.out.println(Thread.currentThread().getName() + "搜集" + tmp + "个龙珠");
                try {
                    cyclicBarrier.await();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                } catch (BrokenBarrierException e) {
                    e.printStackTrace();
```

```
            }
        },String.valueOf(i)).start();
    }
  }
}
```

3.Semaphore

semaphone 信号量

```
package net.yscxy.add;

import java.util.concurrent.Semaphore;
import java.util.concurrent.TimeUnit;

/**
 * @Author WangFuKun
 * @create 2020/11/21 15:34
 */
public class SemaphoreDemo {
    public static void main(String[] args) {
        //线程数量：也就是我们的停车位,一般限流的时候用
        Semaphore semaphore = new Semaphore(3);
        for (int i = 1; i <= 6; i++) {
            new Thread(() -> {
                try {
                    //得到
                    semaphore.acquire();
                    System.out.println(Thread.currentThread().getName() + "抢到了车位");
                    TimeUnit.SECONDS.sleep(2);
                    System.out.println(Thread.currentThread().getName() + "离开车位");
                } catch (InterruptedException e) {
                    e.printStackTrace();
                } finally {
                    //释放
                    semaphore.release();
                }

            }, String.valueOf(i)).start();
        }
    }
}
```

semaphore.acquire() 获得，假设已经满了，等待被释放为止！

semaphore.release()释放，会将当前信号量释放+1，然后唤醒等待的线程

作用：多个共享资源互斥的使用！并发限流，控制最大的线程数量