<https://cloud.tencent.com/developer/article/1362835>

java创建线程有哪几种方法

1. 继承 Thread 类，然后调用 start 方法。

class SampleThread extends Thread {

//重写run方法，线程运行后，跑的就是run方法

public void run(){

//System.out.println("");

}

public static void main(String[] args){

Thread t1 = new SampleThread();

Thread t2 = new SampleThread();

t1.start(); //线程运行,调用的 run()方法.

t2.start(); //线程运行,调用的 run()方法..

}

}

2. 实现 Runnable 接口的 run 方法, 然后再用 Thread 类包裹后，调用 start 方法。

class A implements Runnable{

@Override

public void run() {

// implement run method here

}

public static void main() {

final A obj = new A();

Thread t1 = new Thread(new A());

t1.start();

}

}

当传入一个Runnable target参数给Thread后，Thread的run()方法就会调用target.run()，参考JDK源代码：

public void run() {

　　if (target != null) {

　　 target.run();

　　}

}

3. 实现 Callable 接口的 call 方法，用 FutureTask 类包裹 Callable 对象。然后再用 Thread 类包裹 FutureTask 类，并调用 start 方法。call() 方法可以有返回值。

class MyCallable implements Callable {

@Override

public Integer call() throws Exception {

int sum = 0;

for (int i = 1; i <= 100; i++) {

sum += i;

}

return sum;

}

public static void main(String[] args) throws Exception {

MyCallable mc = new MyCallable(); //实例化 callable

FutureTask oneTask = new FutureTask(mc); //用FutureTask包裹

Thread oneThread = new Thread(oneTask); //用Thread包裹

oneThread.start();

System.out.print(oneTask.get()); //获取返回值

}

}

Callable 方法在 Java 8 后，支持拉姆达表达式的写法，可以创建一个 FutureTask 类，语句上不是太罗嗦。 Callable 方式有以下几个优点：

* 可以捕获线程上的异常。
* 可以通过 get 方法得到返回值。
* get 方法阻塞当前线程，直到调用的线程运行结束。
* 可以取消线程的运行。

Java 6 之后，还可以通过创建线程池来创建线程，使用 ExecutorService 的 execute 方法:

ExecutorService es = Executors.newCachedThreadPool();

Runnable r = <your runnable here>;

es.execute(r);

方式间的对比：

1.直接继承Thread和实现Runnable无法获取执行结果。

2.从Java 1.5开始，就提供了Callable和Future，通过它们可以在任务执行完毕之后得到任务执行结果。

3.Future就是对于具体的Runnable或者Callable任务的执行结果进行取消、查询是否完成、获取结果。必要时可以通过get方法获取执行结果，该方法会阻塞直到任务返回结果。

4.FutureTask类实现了RunnableFuture接口，RunnableFuture继承了Runnable接口和Future接口，而FutureTask实现了RunnableFuture接口。所以它既可以作为Runnable被线程执行，又可以作为Future得到Callable的返回值。FutureTask是Future接口的一个唯一实现类。  
5.　如果为了可取消性而使用 Future 但又不提供可用的结果，则可以声明 Future<?> 形式类型、并返回 null 作为底层任务的结果。