在日常开发中，线程已经是必备的知识了，这次让我们一块研究下线程有哪些创建方式吧。

1、使用Thread实现线程

2、实现Runnable接口实现线程

3、使用Callable实现线程

一、继承Thread创建

实现Thread子类并启动线程分为以下几步：

1、继承Thread类，重写run方法，实现线程逻辑

2、实例化线程类对象

3、调用start方法，启动线程（可以在启动前，修改线程的名称和守护线程等属性）

具体实现一下

@Slf4j  
public class MgDemo202063002 extends Thread {  
 @Override  
 public void run() {  
 *log*.info("运行线程1");  
 }  
 public static void main(String[] args) {  
 MgDemo202063002 demo = new MgDemo202063002();  
 demo.start();  
 }  
}

Thread实现的优点是，实现简单，实例化对象即线程对象，直接启动即可。缺点是使用的是继承，如果有其他继承需求很难处理。

二、实现Runnable接口

实现Runnable接口并启动线程包括一下步骤

1、实现Runnable接口，并在run方法中实现逻辑

2、实例化Runable对象

3、将实例化对象传递到Thead对象中

4、执行Thread对象的start方法

实现Runnable对象的方法有三种

1、创建一个class对象，实现接口Runable

2、直接实现Runnable的匿名类

3、使用lamda表达式实现匿名类

具体实现一下

@Slf4j  
public class MgDemo202063001 implements Runnable {  
 @Override  
 public void run() {  
 *log*.info("运行线程1");  
 }  
  
 public static void main(String[] args) {  
 Thread thread1 = new Thread(new MgDemo202063001());  
 thread1.start();  
  
 Thread thread2 = new Thread(new Runnable() {  
 @Override  
 public void run() {  
 *log*.info("运行线程2");  
 }  
 });  
 thread2.start();  
  
 Thread thread3 = new Thread(()-> {  
 *log*.info("运行线程3");  
 });  
 thread3.start();  
 }  
}

代码中thread2中的runnable是通过匿名类来实现，thread3是通过lamda表达式实现。

Runnable实现的好处

1、实现的是接口，还可以继承其他类

2、Runnable的实例化对象可以传递到多个线程中，处理共享资源很方便

缺点是实现比Thread要复杂一点

三、实现Callable接口

通过Callable需要以下步骤

1、实现Callable接口对象，并实例化

2、将实例化对象，传递到FutureTask实例化对象中

3、将FutureTask对象，传递到Thread中，并启动线程

使用Callable实现的线程可以在任务结束有返回值，可以很好地监听任务状态。

让我们具体实现一下

@Slf4j  
public class MgDemo202063003 implements Callable<String> {  
 @Override  
 public String call() throws Exception {  
 Thread.*sleep*(1000\*5);  
 return Thread.*currentThread*().getName() +" task finish";  
 }  
  
 public static void main(String[] args) throws ExecutionException, InterruptedException {  
 MgDemo202063003 demo = new MgDemo202063003();  
 FutureTask<String> future = new FutureTask<String>(demo);  
 Thread thread = new Thread(future);  
 thread.setName("mgTask1");  
 thread.start();  
 *log*.info(future.isDone()+"");  
 *log*.info(future.get());  
 *log*.info(future.isDone()+"");  
 }  
}

通过future.get()可以获取返回值，不过执行这个方法会进入阻塞状态直到任务结束。future.isDone()方法可以获取到任务状态。

这种实现的优点刚才已经说过，又返回结果，并且很方便的监听任务状态。

线程的实现方法就讨论到这里，有感兴趣或者想深入了解的知识点，可以留言或者关注公众号