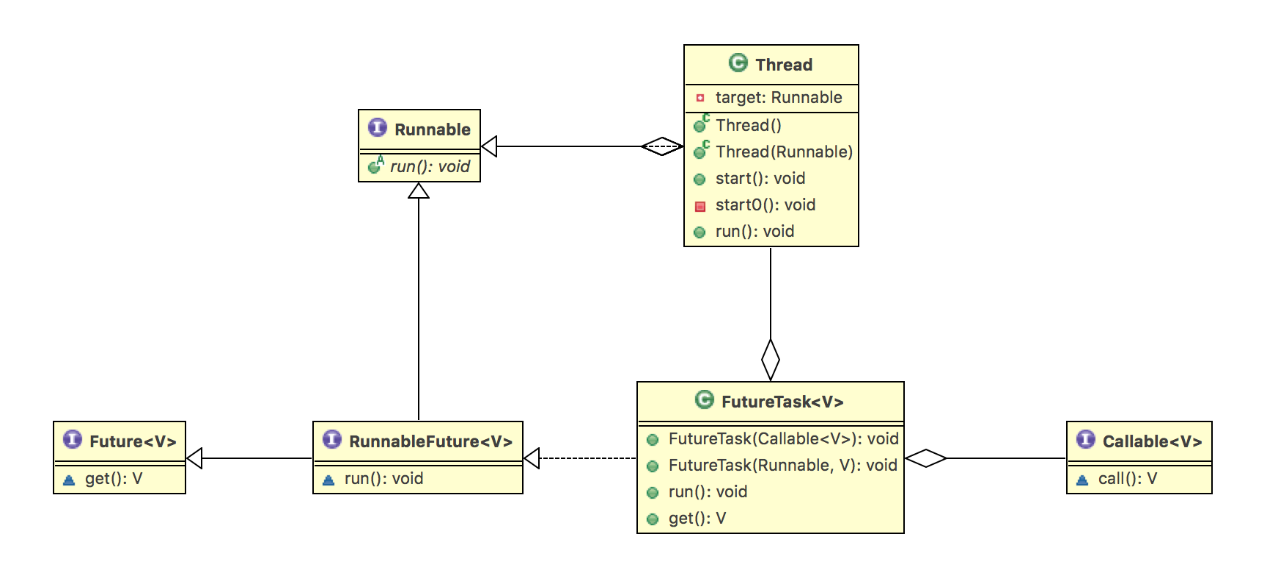
多线程实现方式

多线程有几种实现方式?如果被问到这个问题一定很头疼，因为答案不一。但本质上来讲，个人认为只有一种方式：实现Runnable接口。



1. 实现Runnable接口

public class DemoThreadTask implements Runnable{

@Override

public void run() {

// TODO Auto-generated method stub

}

public static void main(String[] args) {

DemoThreadTask task = new DemoThreadTask();

Thread t = new Thread(task);

t.start();

...

}

}

实现Runnable接口，利用Runnable实例构造Thread，是较常用且最本质实现。此构造方法相当于对Runnable实例进行一层包装，在线程启动时，调用Thread的run方法从而间接调用target.run():

public class Thread implements Runnable {

/\* What will be run. \*/

private Runnable target;

public void run() {

if (target != null) {

target.run();

}

}

...

}

1. 继承Thread类

public class DemoThread extends Thread{

@Override

//重写run方法

public void run() {

// TODO Auto-generated method stub

}

public static void main(String[] args) {

DemoThread t = new DemoThread();

t.start();

...

}

}

这种实现方式是显式地继承了Thread，但从类图中我们可以看到，Threa类本身就继承自Runnable，所以继承Thread地本质依然是实现Runnable接口定义的run方法。

需要注意的是继承Thread方式，target对象为null，重写了run方法，导致方式1中的Thread原生的run方法失效，因此并不会调用到target.run()的逻辑，而是直接调用子类重写的run方法。

因为Java是单继承，此方式一般不常用。

1. 实现Callable接口并通过FutureTask包装

先看demo:

public class DemoCallable implements Callable<String>{

@Override

public String call() throws Exception {

// TODO Auto-generated method stub

return null;

}

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

DemoCallable c = new DemoCallable();

FutureTask<String> future = new FutureTask<>(c);

Thread t = new Thread(future);

t.start();

...

String result = future.get(); //同步获取返回结果

System.out.println(result);

}

}

实现Callable接口通过FutureTask包装，可以获取到线程的处理结果，future.get()方法获取返回值，如果线程还没执行完，则会阻塞。

这个方法里，明明没有看到run方法，没有看到Runnable，为什么说本质也是实现Runnable接口呢？

回看开篇的类图，FutureTask实现了RunnableFuture，RunnableFuture则实现了Runnable和Future两个接口。因此构造Thread时，FutureTask还是被转型为Runnable使用。因此其本质还是实现Runnable接口。

至于FutureTask的工作原理，后续篇章继续分析。

1. 匿名内部类

匿名内部类也有多种变体，上述三种方式都可以使用匿名内部类来隐式实例化。

public class Demo{

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

//方式一：Thread匿名内部类

new Thread(){

@Override

public void run() {

// TODO Auto-generated method stub

}

}.start();

//方式二：Runnable匿名内部类

new Thread(new Runnable() {

@Override

public void run() {

// TODO Auto-generated method stub

}

}).start();

...

}

}

匿名内部类的有点在于使用方便，不用额外定义类，缺点就是代码可读性差。

1. Lambda表达式

Lambda表达式时jdk8引入的，已不是什么新东西，demo如下：

public class Demo{

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

new Thread(() -> System.out.println("running") ).start() ;

...

}

}

如此简洁的Lambda表达式，有没有吸引到你呢？当然本质不多说，还是基于Runnable接口。

1. 线程池

public class DemoThreadTask implements Runnable{

@Override

public void run() {

// TODO Auto-generated method stub

System.out.println("running");

}

public static void main(String[] args) {

DemoThreadTask task = new DemoThreadTask();

ExecutorService ex = Executors.newCachedThreadPool();

ex.execute(task);

...

}

}

线程池与前面所述其他方式的区别在于执行线程的时候由ExecutorService去执行，最终还是利用Thread创建线程。线程池的优势在于线程的复用，从而提高效率。

关于线程池，后续篇章会继续详解。

1. 定时器

public class DemoTimmerTask {

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

Timer timer = new Timer();

timer.scheduleAtFixedRate((new TimerTask() {

@Override

public void run() {

System.out.println("定时任务1执行了....");

}

}), 2000, 1000);

}

}

TimeTask实现了Runnable接口，Timer内部有个TimerThread继承自Thread，因此绕回来还是Thread + Runnable。

总结，多线程的实现方式，在代码中写法千变万化，但其本质万变不离其宗。