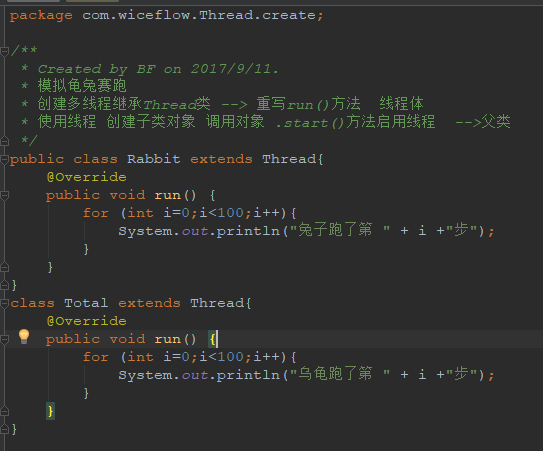
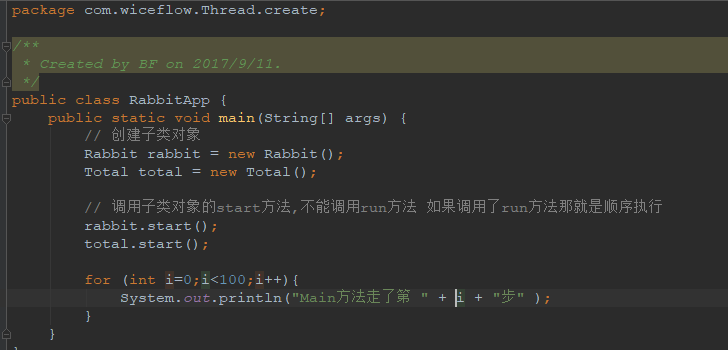
# 线程创建的三种方式

## 继承Thread + run()



### Mian方法测试



### 注意！

// 调用子类对象的start方法,不能调用run方法 如果调用了run方法那就是顺序执行

## 实现Runnable +run() 推荐这个方法

这里使用了静态代理模式，当新建的线程体继承了Runnable接口后，Runnable接口并没有实例化，它没有启动线程等各种方法。

这时候还是得创建Thread对象，用代理模式启用线程

1)、避免单继承的局限性

2)、便于共享资源

使用 Runnable 创建线程

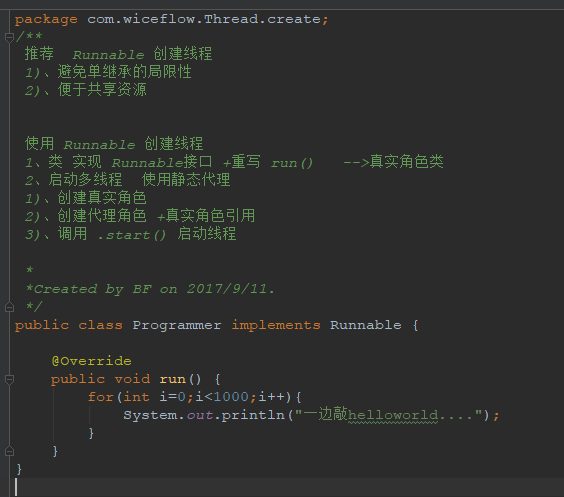
1、类 实现 Runnable接口 +重写 run() -->真实角色类

2、启动多线程 使用静态代理

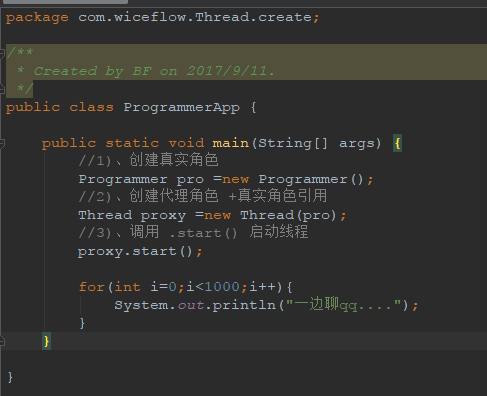
1)、创建真实角色

2)、创建代理角色 +真实角色引用

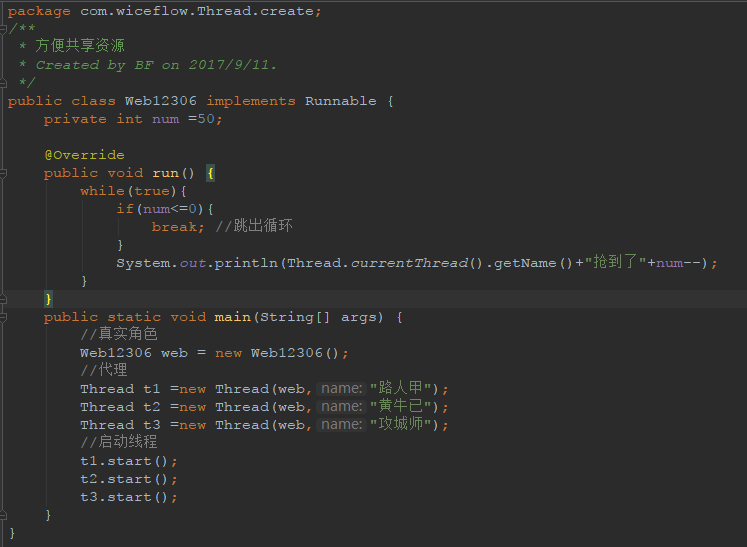
3)、调用 .start() 启动线程



### 测试方法



### 便于共享的例子



## 三 通过Callable接口实现多线程

此方法可以设置返回值和自定义抛出异常

优点：可以获取返回值 Callable 和 Future接口 Callable是类似于Runnable的接口，实现Callable接口的类和实现Runnable的类都是可被其它线程执行的任务。 Callable和Runnable有几点不同： （1）Callable规定的方法是call()，而Runnable规定的方法是run(). （2）call()方法可抛出异常，而run()方法是不能抛出异常的。 （3） Callable的任务执行后可返回值，运行Callable任务可拿到一个Future对象，而Runnable的任务是不能返回值的。 Future 表示异步计算的结果。它提供了检查计算是否完成的方法，以等待计算的完成，并检索计算的结果。 通过Future对象可了解任务执行情况，可取消任务的执行，还可获取任务执行的结果。 缺点 :繁琐思路: 1)、创建 Callable实现类+重写call 2)、借助 执行调度服务 ExecutorService,获取Future对象ExecutorService ser=Executors.newFixedThreadPool(2);Future result =ser.submit(实现类对象) 3）、获取值 result.get()4 )、 停止服务 ser.shutdownNow();