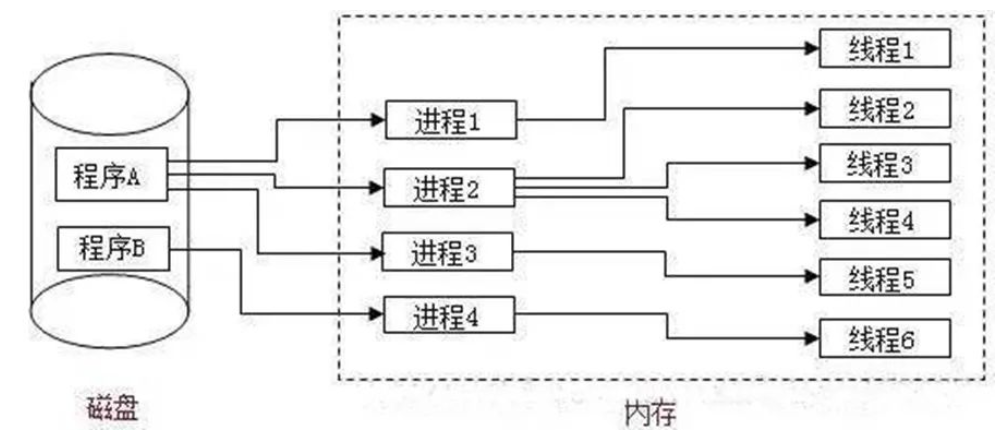
# Java多线程和并发

## 理论基础

进程：操作系统在内存中进行资源分配的基本单位

线程（thread）：进程的执行运算的最小单位，一个进程中的多个线程可以并发执行。main方法为单独的路径

多线程：软硬件多条执行的流程。



cpu一般都会有多个核心，每个核心都调度一个线程执行。cpu几个核心就最多可以调度多少个线程。

线程不安全：多个线程对同一个共享数据进行访问而不采取同步操作。

## 多线程

### 多线程的创建

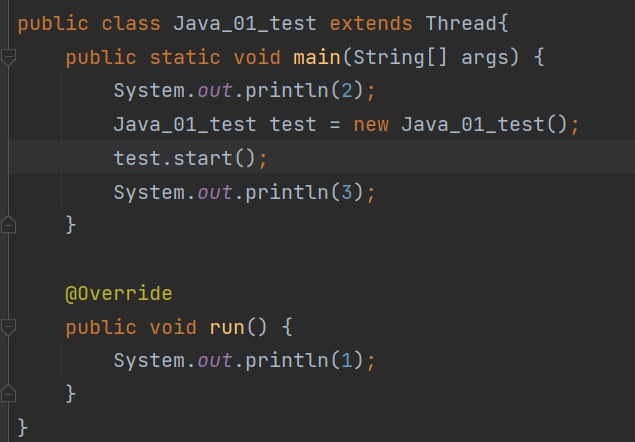
多线程的创建有三种方法

#### 继承Thread方法

定义一个子类MyThread继承线程类Thread，重写run()方法

创建MyThread类的对象

调用线程对象的start()方法启动线程，启动后还是执行run方法



主线程(main)和子线程（test）是同时执行的，多次打印结果可能不一样。要看start()方法的顺序位置，start()方法执行了才启动了子线程。

缺点：过于繁琐

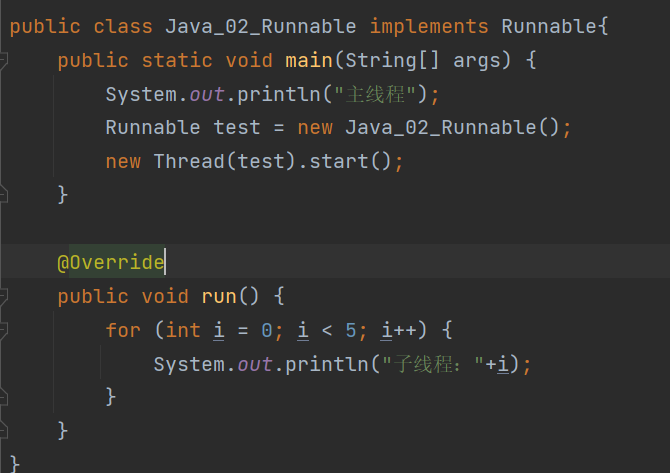
#### 实现Ruunable接口

定义一个线程任务类MyRunnable实现Runable接口，重写run()方法

创建MyRunnable任务对象

把MyRunnable任务对象交给Thread处理

调用线程对象的start()方法启动线程



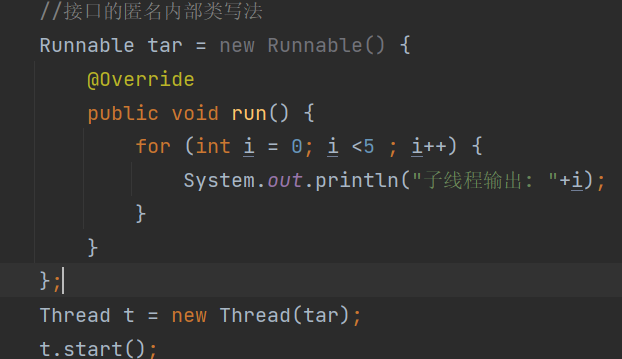
缺点：和方式一，子线程没有返回值。只能执行子线程的任务。

#### ·以匿名内部类的方式构建子线程

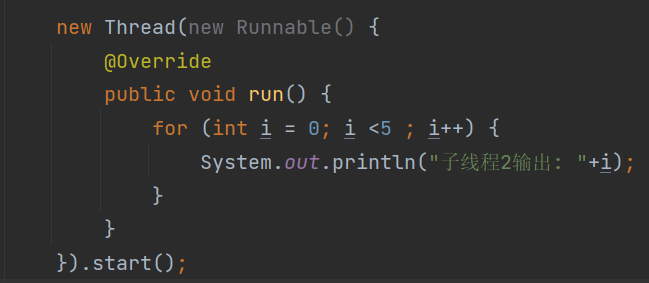
创建Runnable的匿名内部类

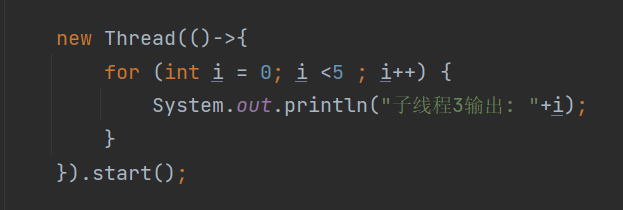
交给Thread处理

调用start()方法启动线程



简化写法：



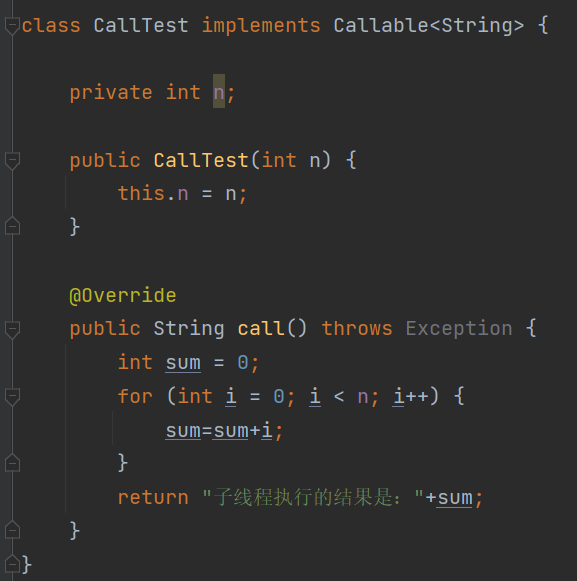


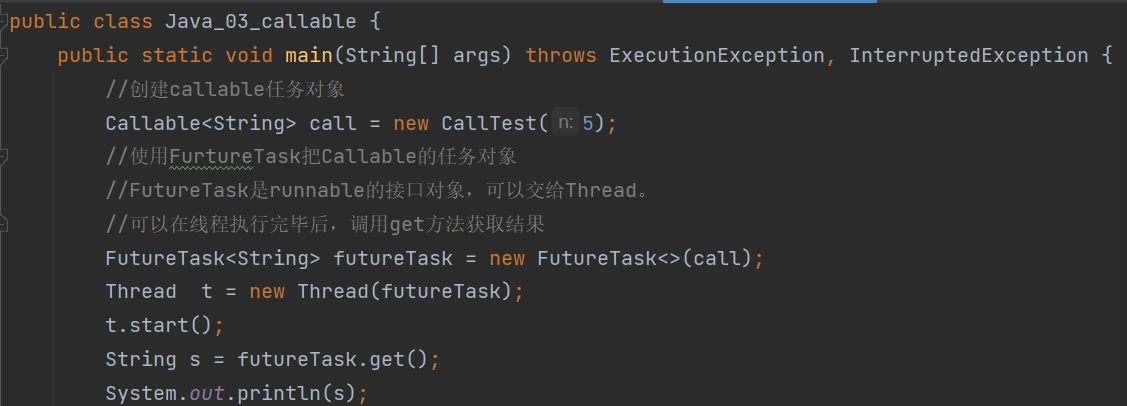
#### 实现Callable接口

可以得到子线程的返回值。

1. 得到任务对象，定义类来实现callable接口，重写call方法，call方法可以得到返回结果。
2. 用FurtureTask把Callable对象封装成线程任务的对象。
3. 把任务对象交给Thread处理
4. 调用Thread的start()方法执行任务
5. 线程调用完毕后，通过Furture的get()方法去获取任务执行的结果

get()方法会等到子线程执行完毕在获取结果。





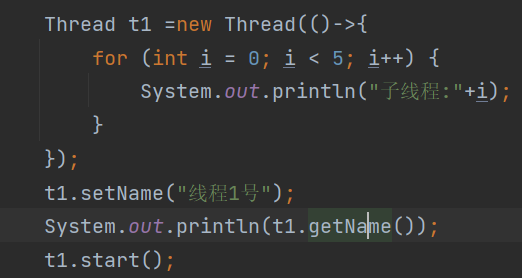
### Thread的常用方法



·常用方法：获取线程名称getNamr(),设置线程名称setName，获取当前线程对象currentThread().

·Thread类提高的方法:yield,join,unterrupt

#### setName和getName

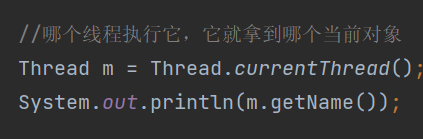


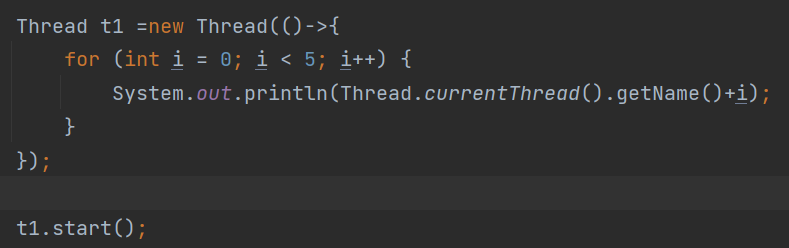
使用构造器可以取代set方法



#### currentThread()

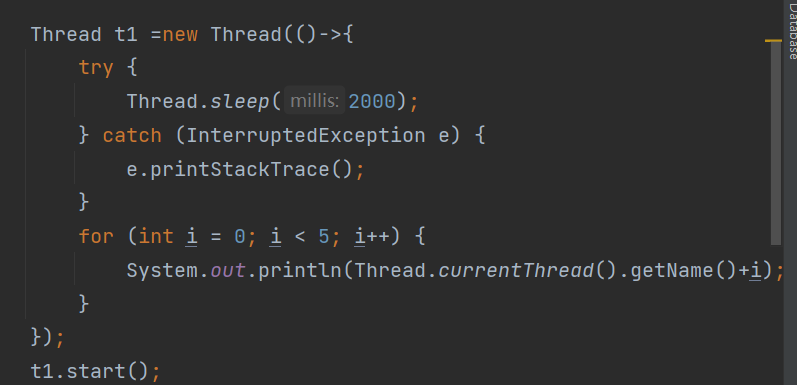
打印主线程的名字





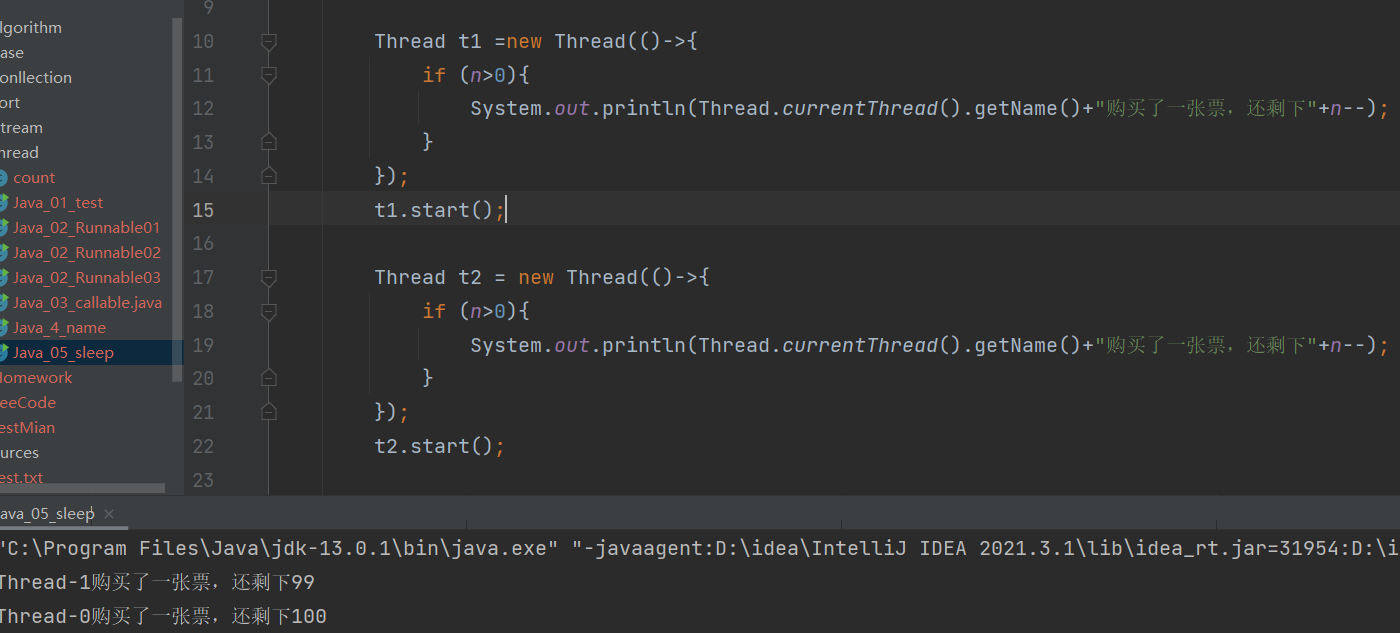
#### sleep()

休眠方法，让当前的线程休眠指定的时间后在继续执行，单位为毫秒



### 线程安全

多个线程同时操作同一个共享资源的时候可能会出现业务安全问题



例如多人操作无法进行顺序减。

线程安全的解决方法：线程同步

在多个线程的时候实现依次先后访问共享资源。也就是上锁，给共享资源上锁，每次只能一个线程访问占用，使用完后在解锁，其他的线程才能进去。

#### 线程同步

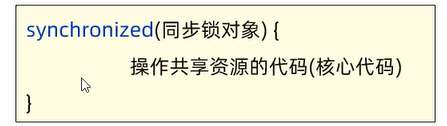
使用线程同步来使各个线程直接互不影响

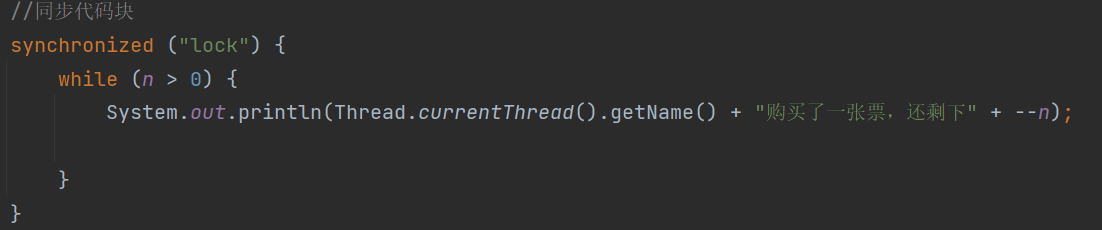
#### 同步代码块

把出现线程安全的代码上锁

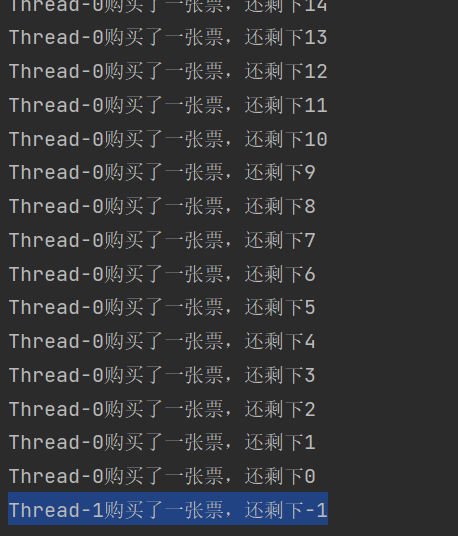
使用synchronized关键字

同步锁对象给一个常量就好





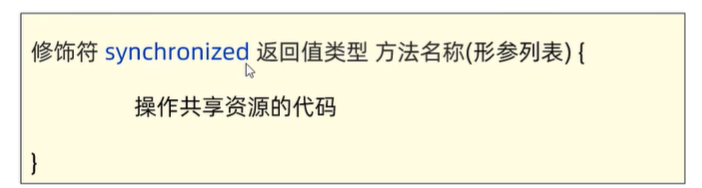
缺点：任意的无关变量影响了其他的线程的执行



只有Thread-0结束了后才能继续的执行。

#### 同步方法

对核心方法进行上锁



#### LOCK锁



### 线程通信

线程间相互发送数据，线程通信通常通过共享一个数据的方式实现

线程间会根据共享数据的情况决定自己怎么做，以及通知其他线程怎么做。

常见模型：

生产者和消费者模型：生产者负责生成数据，消费者负责消费数据。

要求：生产者产生完数据后，唤醒消费者，然后等待自己；消费者消费完数据，唤醒生产者，等待自己。

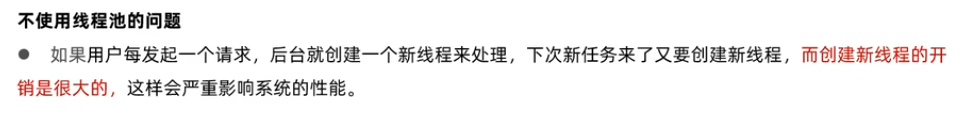
前提：多个线程操作同一个共享资源且保障线程安全。



这些方法需要使用锁对象调用。

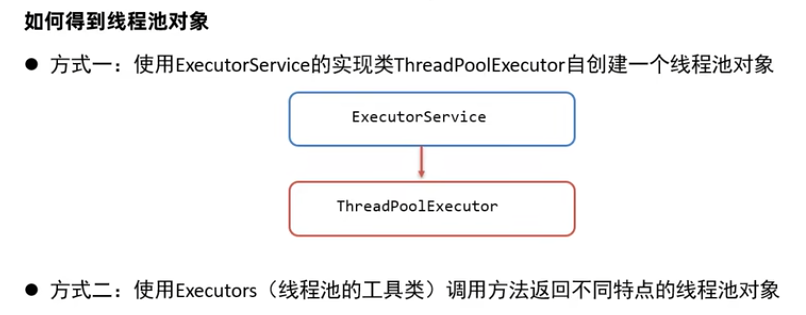
### 线程池

线程池：可以复用线程的技术

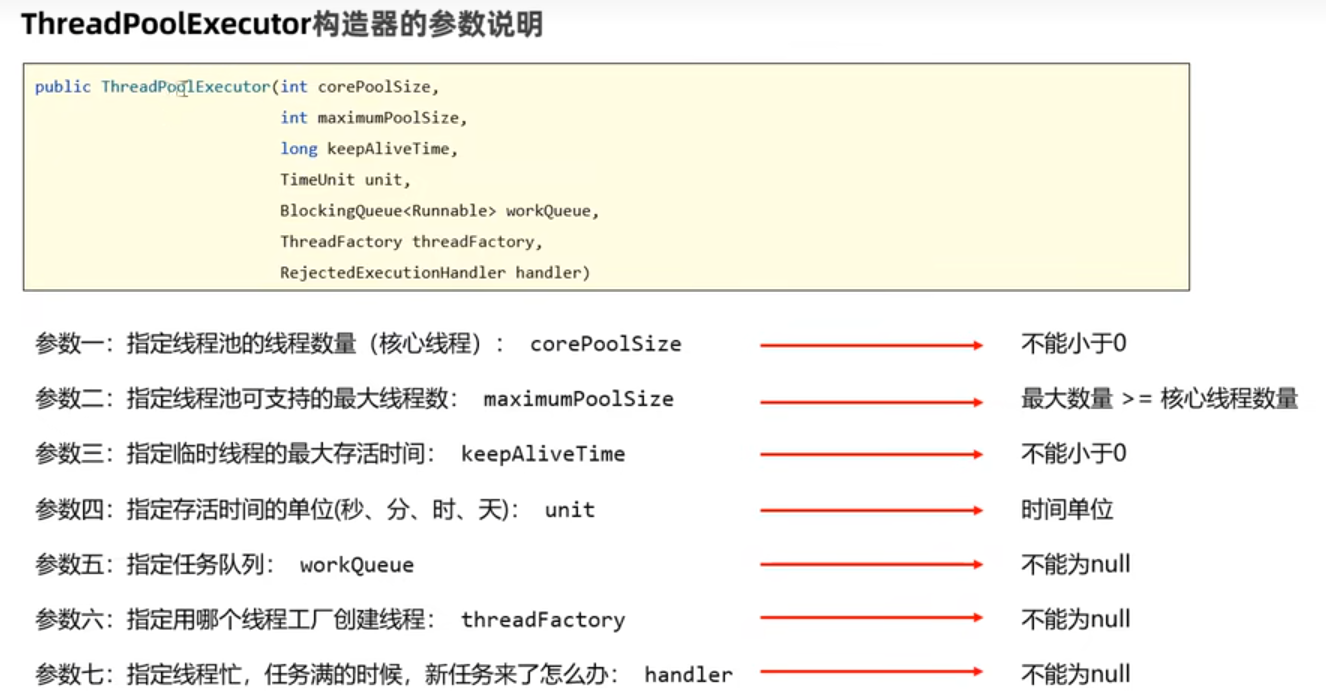


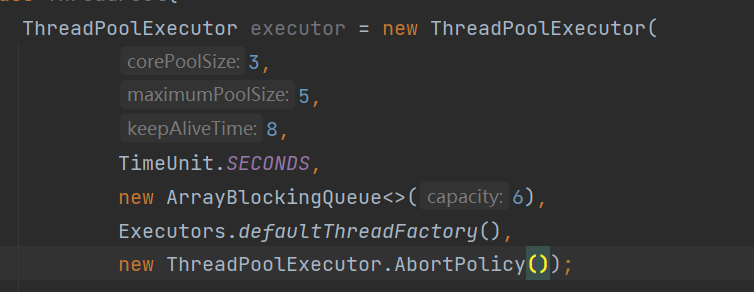
使用任务队列，固定任务线程的数量依次调用任务队列的任务。

代表线程池的接口：ExecuotorService



线程池的API和7个参数：





临时线程什么时候创建？

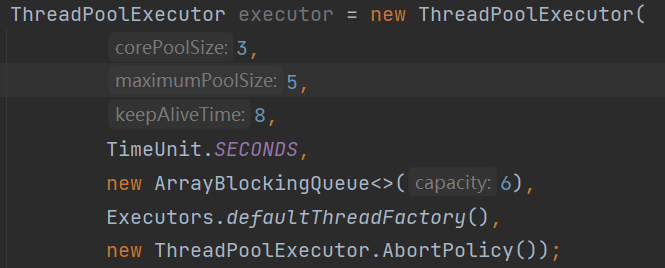
答：新任务提交的时候发现核心线程都在忙，任务队列也满的时候，并且还可以创建临时线程的时候，才创建核心线程

什么时候开始拒绝任务？

核心和临时线程都在忙，任务队列也满了。

可以使用shutDownNow()可以关闭线程池

#### 创建线程池

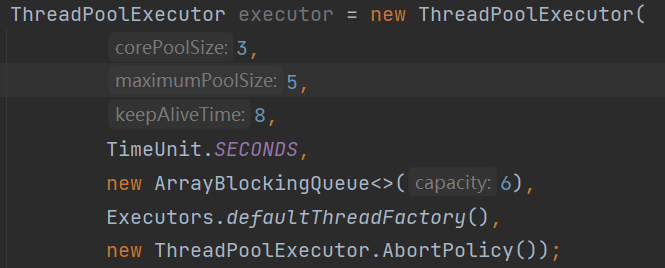




#### 线程池处理Runnable任务：

处理Runnable任务，意思为创建Runnable接口线程，使用

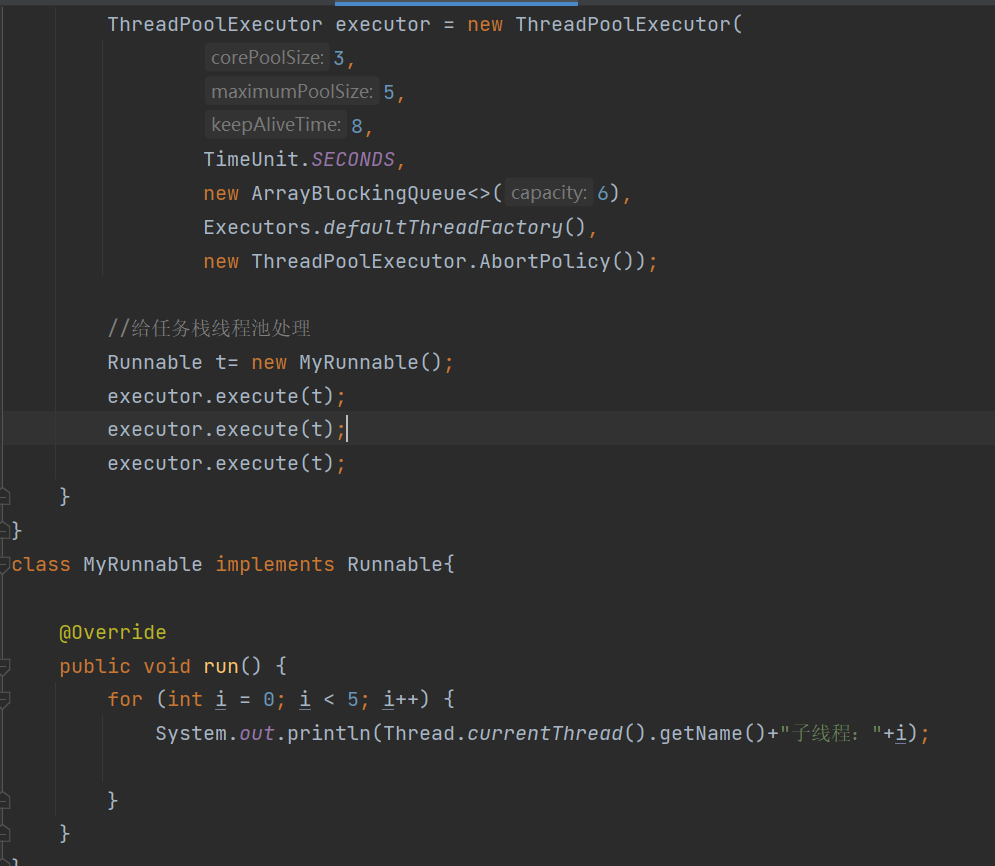
ExecutorsService类来创建Runnable接口。



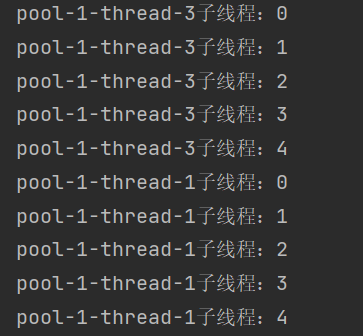




基本使用:

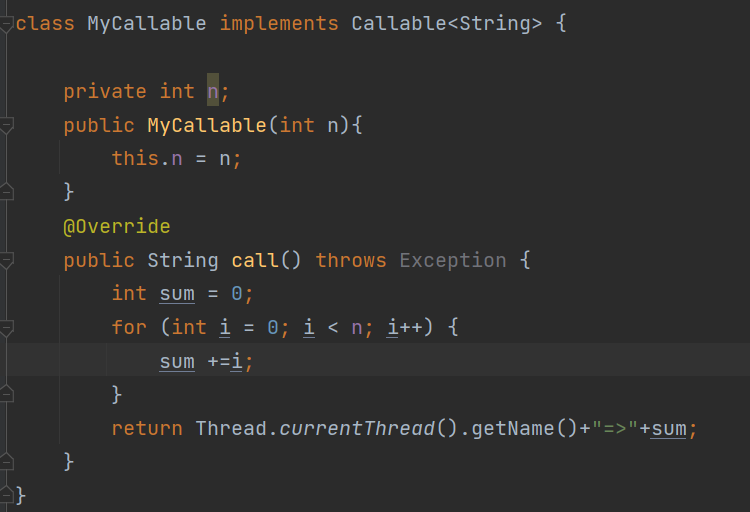


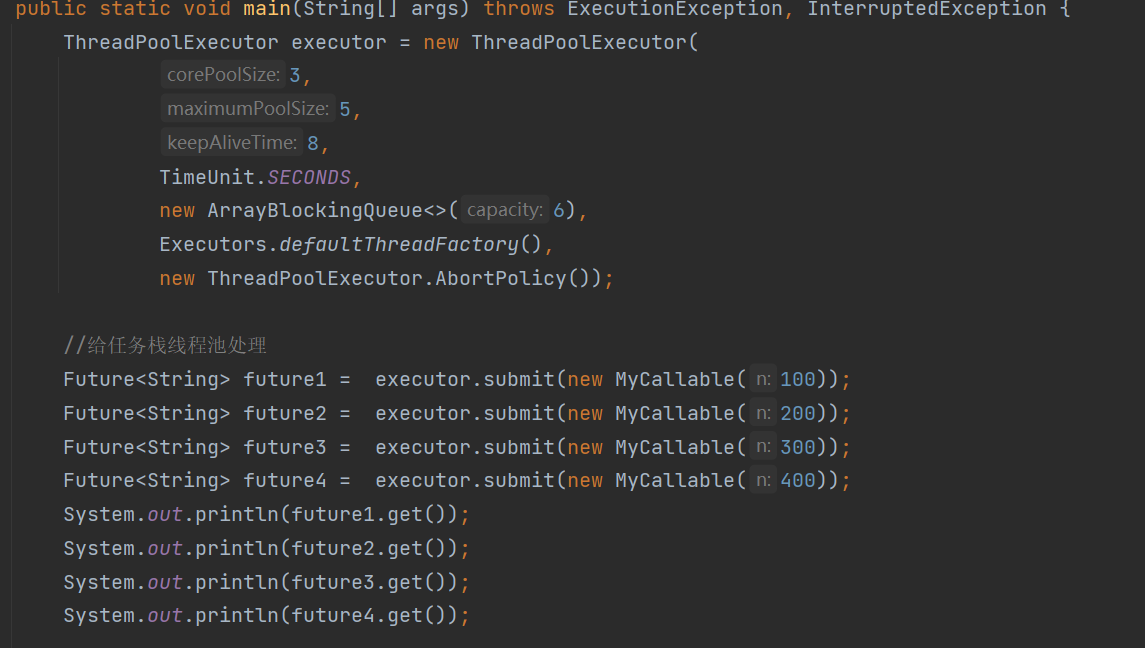
结果是三个线程同时完成任务：



如果是四个线程，会线程复用，忙完后会从队列中拿出任务，用忙玩的线程去完成这个任务。

#### 线程池处理Callable任务：





线程池使用submit方法执行Callable的实现类，使用future来获取返回的值，使用future的get方法来读取返回的值。

### 定时器和线程状态

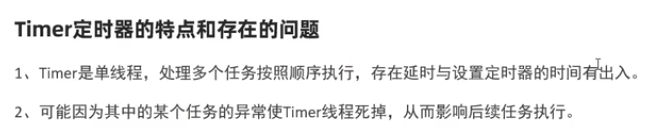
控制任务延时调用，或者周期调用

实现方式：

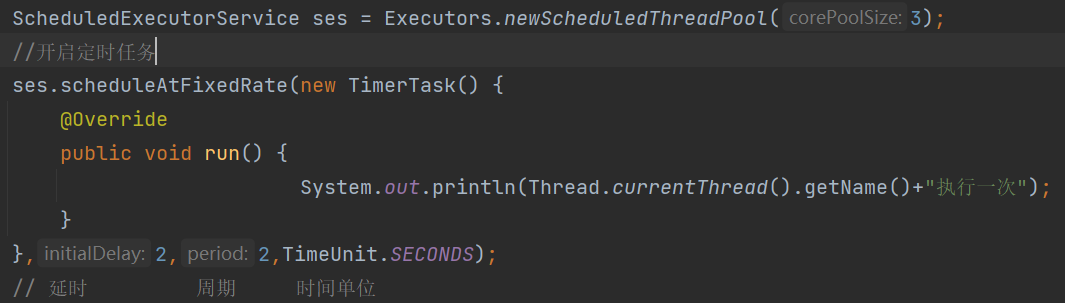


#### Timer定时器。



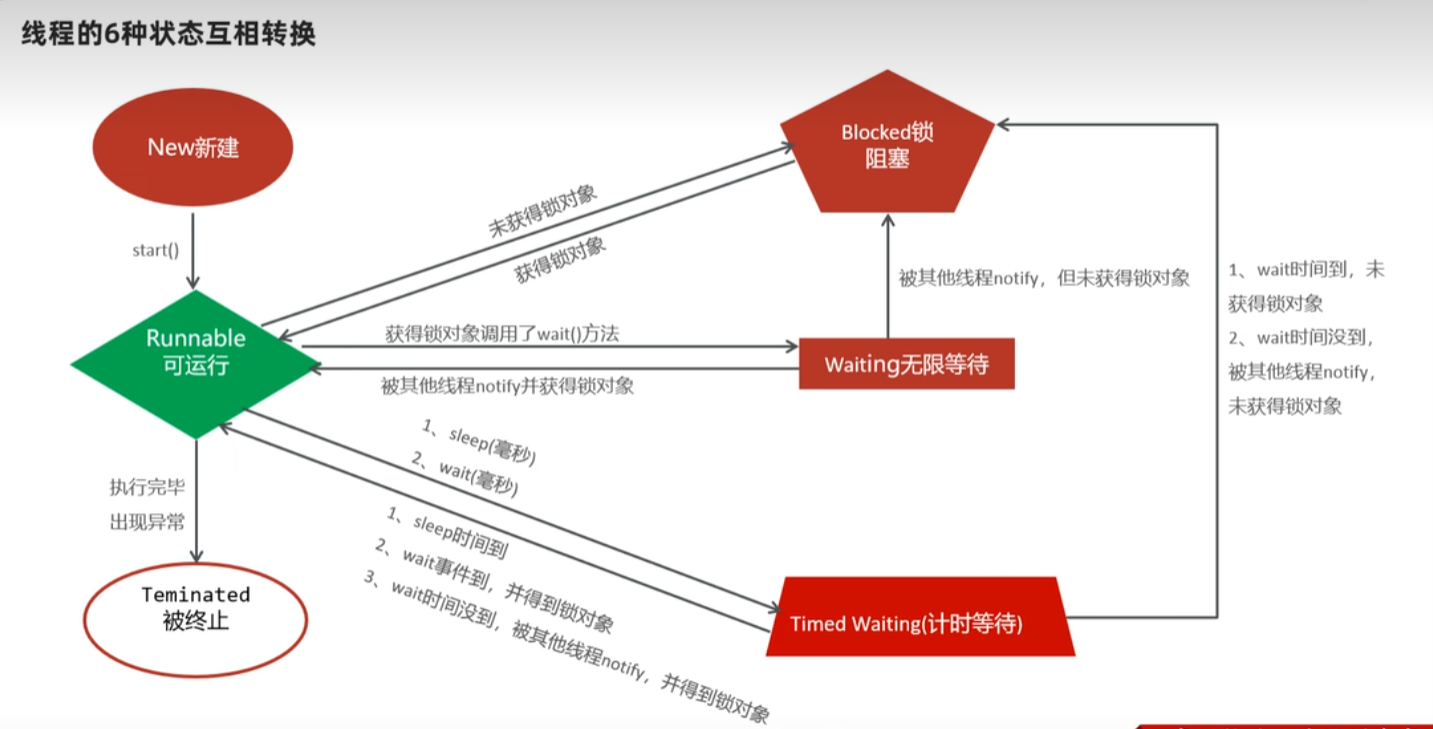


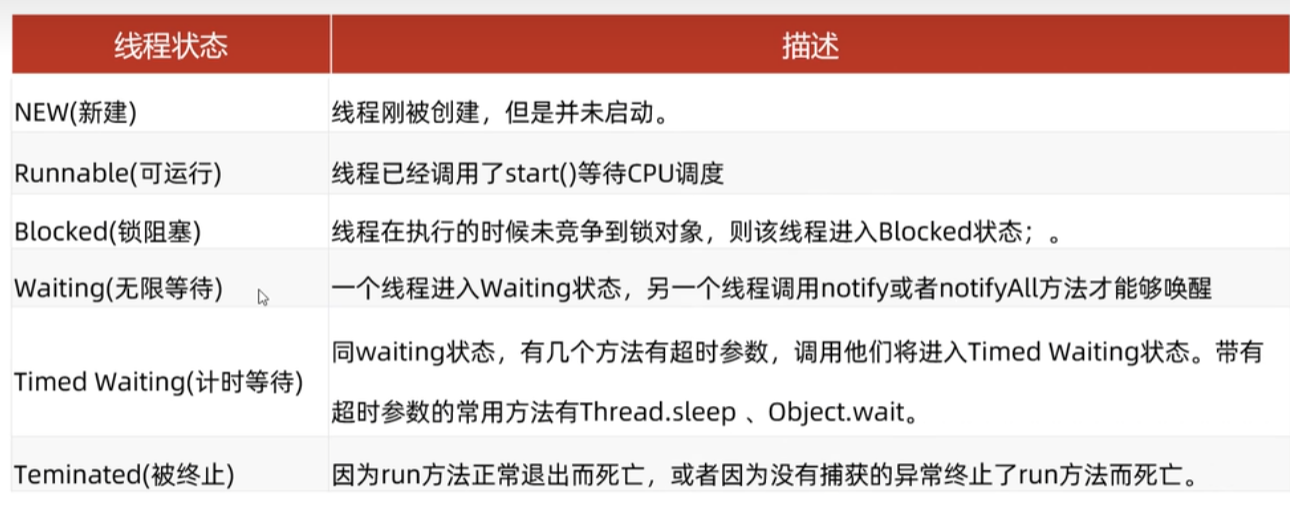
#### ScheduleExecuteService定时器





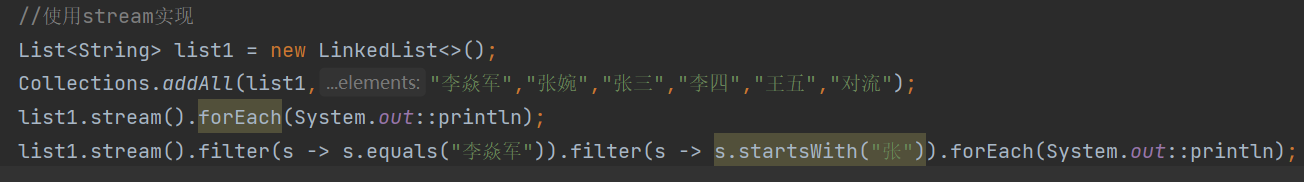
#### 线程状态





# Stream流

结合Lambda简化集合和数组的操作。



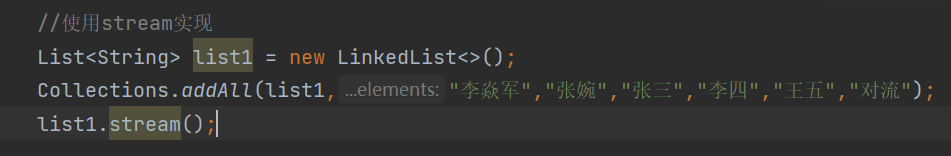
流的思想：将集合或者数组的数据放入Stream流，利用其api来对数据操作。

（System.out::println中的::为引用）

## Stream流的获取

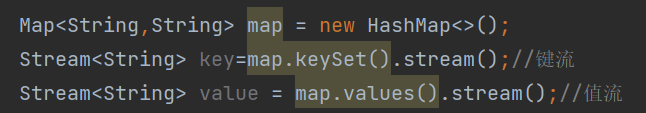
1. 集合使用Collection接口的默认方法stream()生成流。

list的默认Stream流

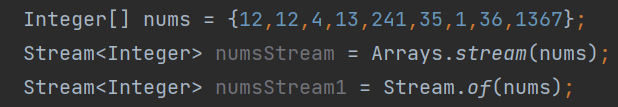


map的默认数据流



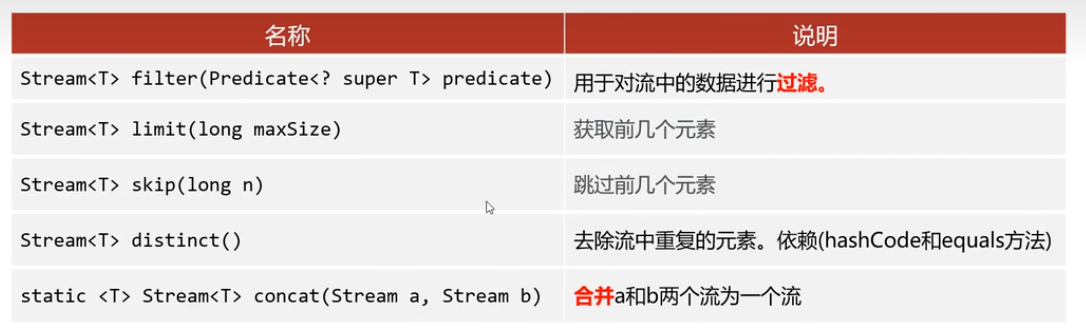


2.数组获取流使用Arrays接口的stream()方法，或者Stream的of方法。

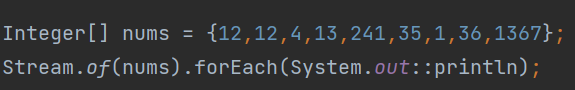


## Stream常用API

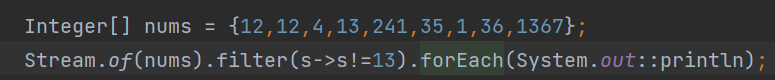
stream流的常用API的中间操作方法，中间方法调用后可以继续的调用中间方法，形成链式编程



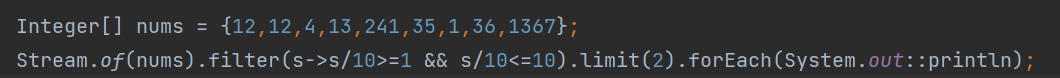
foreach：逐一遍历



filter ：过滤



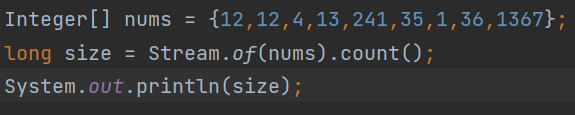
limit ：获取前几个元素



skip ：跳过前几个元素，和limit一样，为跳过前几个元素

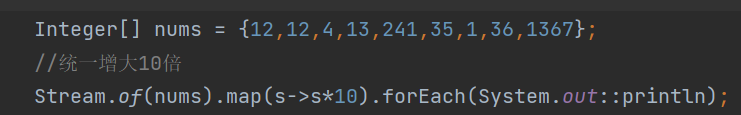
count ：统计个数

--long count(); count是long类型的



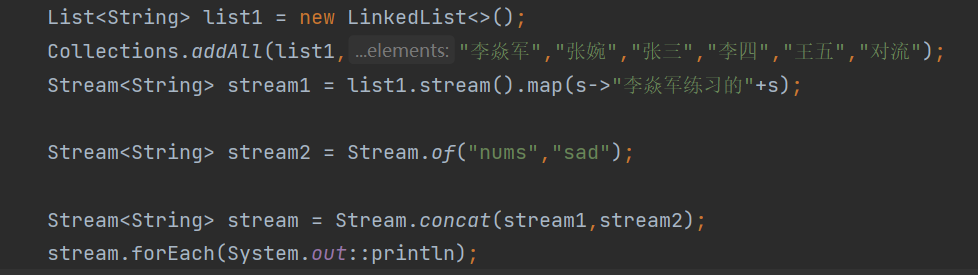
map ：加工方法





concot ：合并流

将两个流合并



max ：获取最大值

使用max后要制定比较规则



## File和IO

将数据放在硬盘的的操作和读取硬盘文件内容

File类定位文件，删除文件，获取文件文本信息。但是不能读写文件内容，读写文件需要使用IO流。

### File类

File类在java.io.File，代表操作系统的文件对象（文件，文件夹）。

File文件可以，定位文件，删除，创建文件。

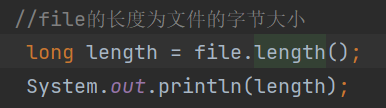
File类创建对象：

file的路径，为了避免转义字符，使用\\，/，或者使用file.separator获取文件分隔符。

file创建对象，支持绝对路径和相对路径（在项目的文件，例如配置文件）。



输入文件的大小。



Fiel类的api

判断文件类型，获取文件信息



创建文件，删除空文件夹



遍历文件夹



### 字符集

计算机底层只能存储二进制，0，1

例如：

·ASCII编码，美国信息交换集，，一个字节一个字符，一个字节8位，共128个字符。

·GBK编码，windows默认编码表，兼容ASCII。2个字节一个字符。

·Unicode编码，统一码，例如UTF-8，容纳大多数国家的常见文字，utf-8后一个汉字是3个字节。

### IO流

IO流被称位输入，输出流，用来读写数据。

读，I，数据从硬盘文件到内存的过程。

写，O，是从内存程序的数据写到硬盘文件的过程。

IO流按照数据的最小单位可以分位字节流和字符流。

字节流适合读取一切内容。（更适合影视频文件）

InputStream 字节输入流，OutputStream 字节输入流

字符流适合读文本内容。

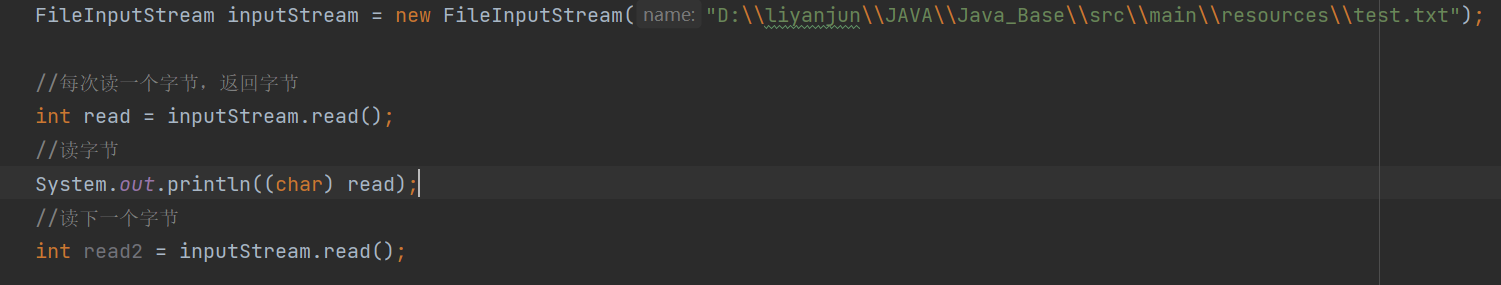
Reader 字符输入流， Wirter 字节输出流

#### 字节流

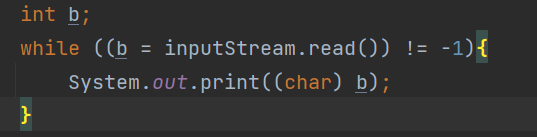
##### ·FileInputStream

read()

读一个字节，返回字节，字节变成内容需要强转char，无法读中文



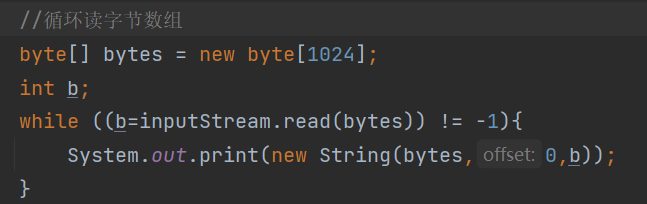
循环读：read返回-1就是读完了



读一个字节数组，一次性读设定的字节



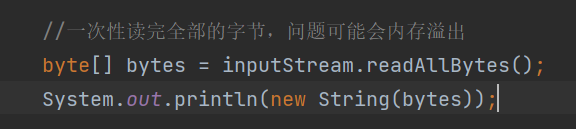
循环读字节数组，设置的数组大小的一次读1kb。



也可以在数组中直接定义文件的大小，

·readAllBytes

一次性读完全部的字节



##### ·FileOutputStream

写数据进硬盘文件

api：



每次写数据会将原文件的数据，在写入新的数据进去。

在new OutputStream的时候加一个true就表示最加数据了。

用write写后需要使用flush方法刷新，不然写的内容还是缓存中，没有到硬盘中。

释放资源close();

在所有的流操作都推荐使用try，catch，finally，用finally来释放资源。

#### 字符流

##### ·FileRead

操作api和字节流差不多，差别在与是一次读一个字符

##### ·FileWriter

操作api和字节流差不多，差别在与是一次写一个字符

#### 缓冲流

提高读写性能

自带了缓冲区，可以提高原始字节流，字符流读写数据的性能。

缓存流的实现类为：BufferedInputStream，BufferedOutputStream

BufferedRead，BufferedWriter。操作和字节流和字符流的操作是一样的。只是效率的提高。

#### 打印流

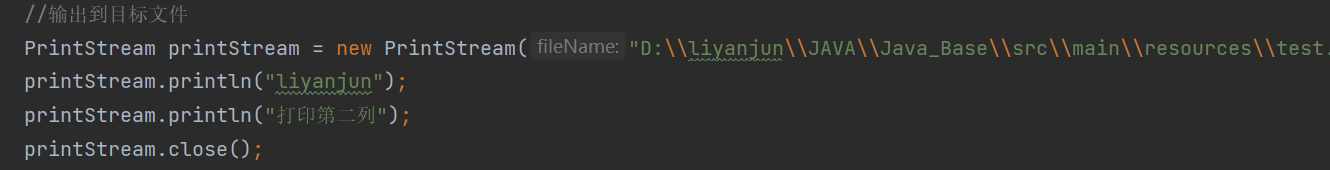
方便高效的写数据到文件中。使用了缓冲输入流。

PrintStream 字节输出，PrintWirter 字符输出。两种没有任何区别。

可以打印任何东西。



写数据很灵活和强大。

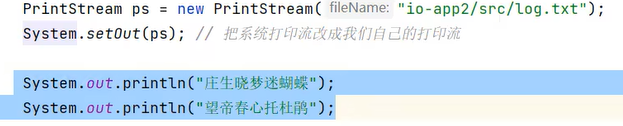


需要追加到文件的话，使用低级管道FileOutputStream，在后面加上true。



### 输出语句的重定向

将输出语句打印到文件中。

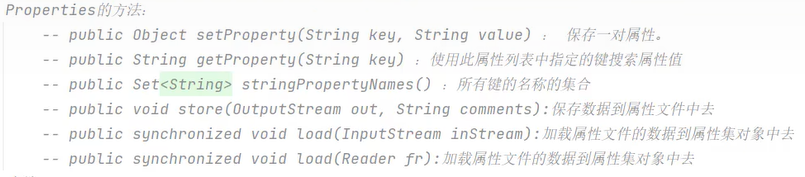


### Properties

一个Map集合，代表的是属性文件，可以把自己的对象中的键值对信息存储到一个属性文件中。

保存到propertie配置文件的方法是store(),第二个参数为注释信息。





# JAVA中变量的存放位置

在方法中声明的变量，即该变量是局部变量，每当程序调用方法时，系统都会为该方法建立一个方法栈，其所在方法中声明的变量就放在方法栈中，当方法结束系统会释放方法栈，其对应在该方法中声明的变量随着栈的销毁而结束，这就局部变量只能在方法中有效的原因

在方法中声明的变量可以是基本类型的变量，也可以是引用类型的变量。

​ （1）当声明是基本类型的变量的时，其变量名及值（变量名及值是两个概念）是放在JAVA虚拟机栈中

​ （2）当声明的是引用类型变量时，所声明的变量名（该变量实际上是在方法中存储的是内存地址值）是放在

​ JAVA虚拟机的栈中，该变量所指向的对象是放在堆类存中的。

在类中声明的变量是成员变量，也叫全局变量，放在堆中的（因为全局变量不会随着某个方法执行结束而销毁）。同样在类中声明的变量即可是基本类型的变量 也可是引用类型的变量

​ （1）当声明的是基本类型的变量其变量名及其值放在堆内存中的

​ （2）引用类型时，其声明的变量仍然会存储一个内存地址值，该内存地址值指向所引用的对象。引用变量名和对应的对象仍然存储在相应的堆中

注意：

○在高级语言中，程序函数调用、函数中定义的变量都用到栈（stack）。

○用malloc, calloc, realloc等函数分配得到变空间是在堆（heap）上。

○在所有函数体外定义的是全局量。

○加了static修饰符后不管放在在哪里都属于静态变量，存放在全局区（静态区）

○在所有函数体外定义的static变量表示在该文件中有效，不能extern到别的文件用。

○在函数体内定义的static表示只在该函数体内有效。

○函数中的"abcdef"这样的字符串存放在常量区。