今日完成任务6

线程池概念

问题：1.线程是宝贵的内存资源、单线程约占1MB空间，过多的分配造成内存溢出。

2.频繁的创建及销毁线程会增加虚拟机回收频率、资源开销，造成程序性能下降。

线程池：

1.线程容器，可设定线程分配的数量上限。

2.将预先创建的线程对象存入池中，并重用线程池中的线程对象。

3.避免频繁的创建和销毁

原理：将任务提交给线程池，由线程池分配线程、运行任务，并在当前任务结束后复用线程

创建线程池

常见的线程池接口和类（所在的包java.util.concurrent）

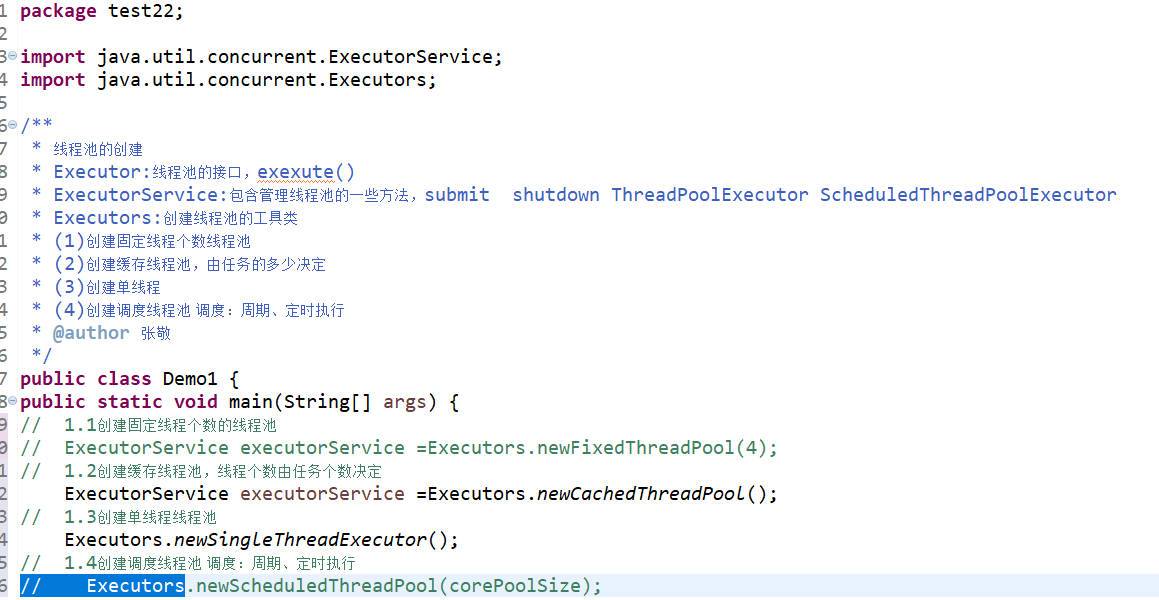
Executer:线程池的顶级接口

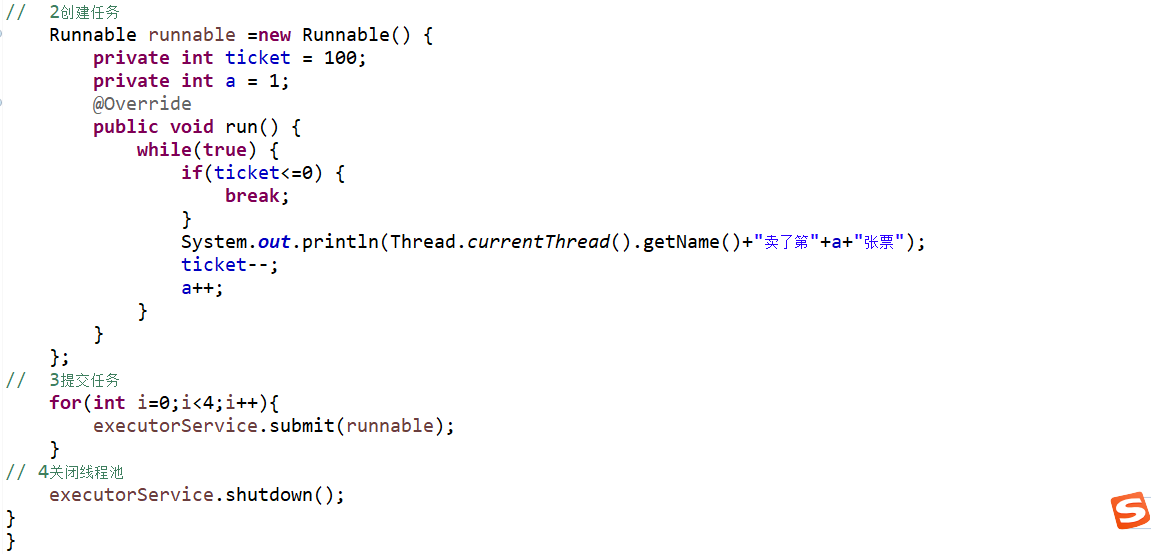
ExecutorService:线程池接口，可通过submit(Runnable task)提交任务代码

Executors工厂类：通过此类可以获得一个线程池

通过newFixedThreadPool(int nThreads)获得固定数量的线程池。参数：指定线程池中线程的数量

通过newCachedThreadPool()获得动态数量的线程池，如不够则创建新的，没有上限。





Callable接口

public interface Callable<V>{

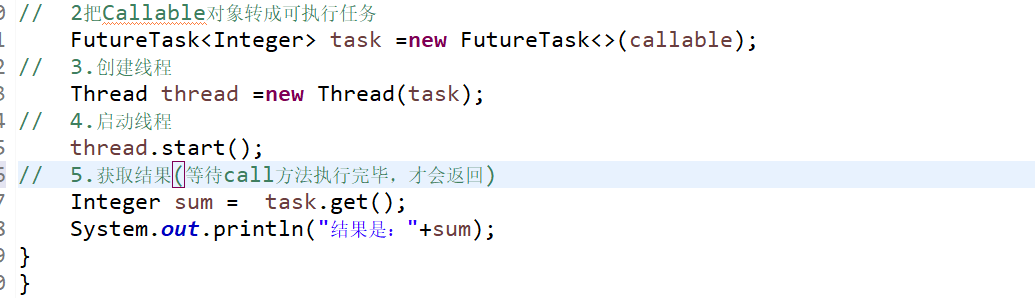
public V Call() throws Exception;

}

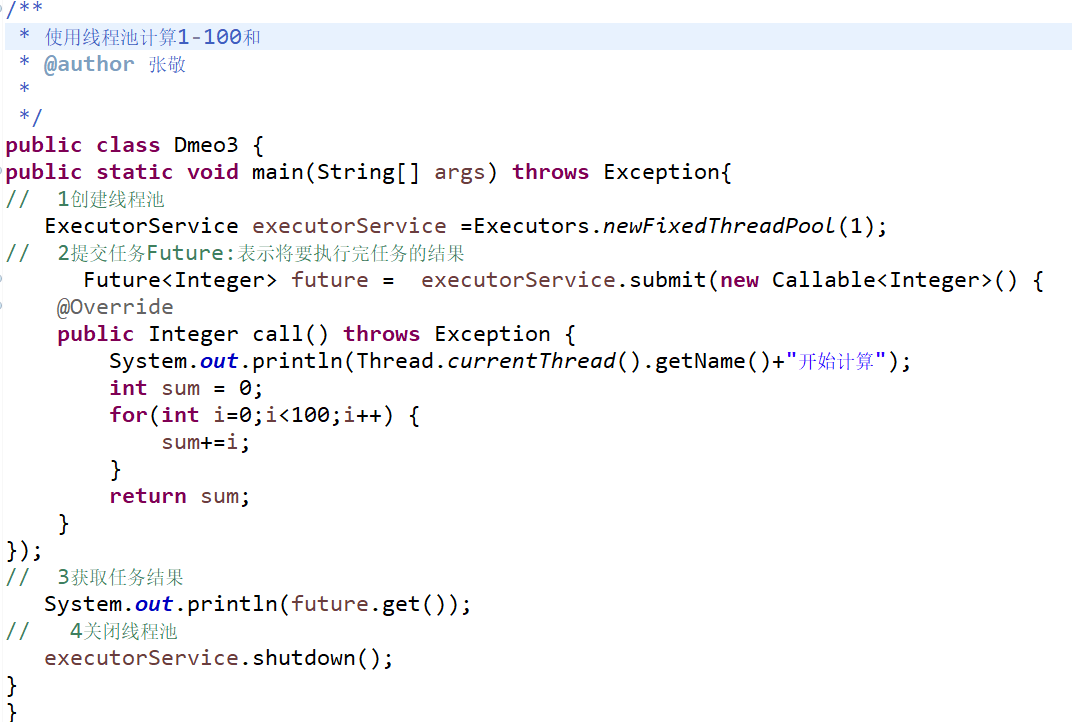
与Runnable接口类似，实现之后代表一个线程任务

Callable具有泛式返回值、可以声明异常





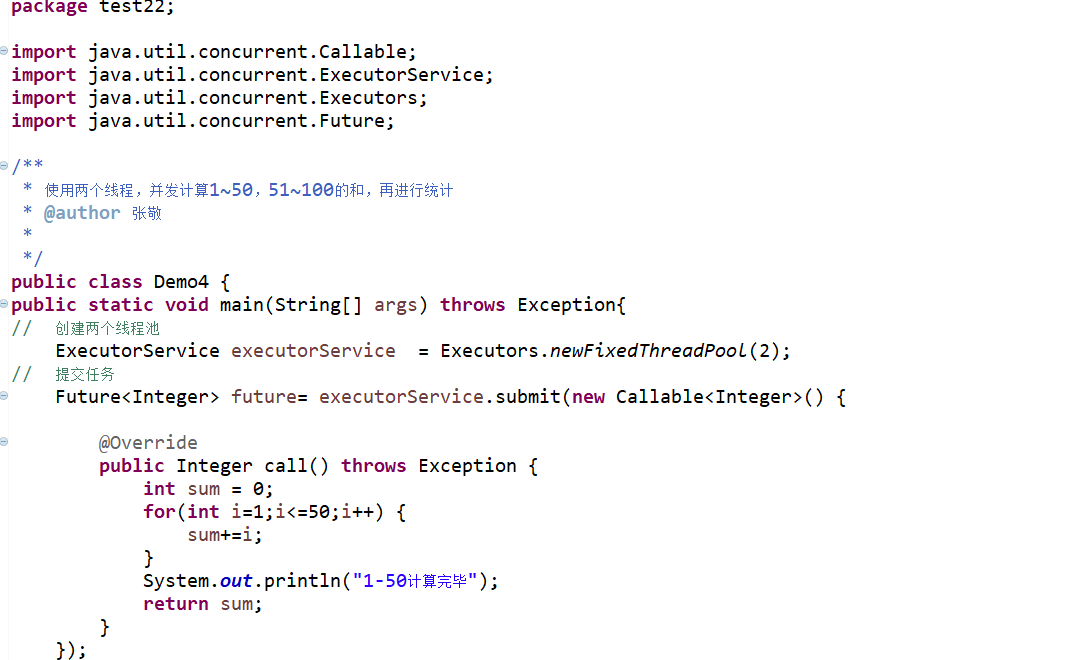
Callable结合线程池使用

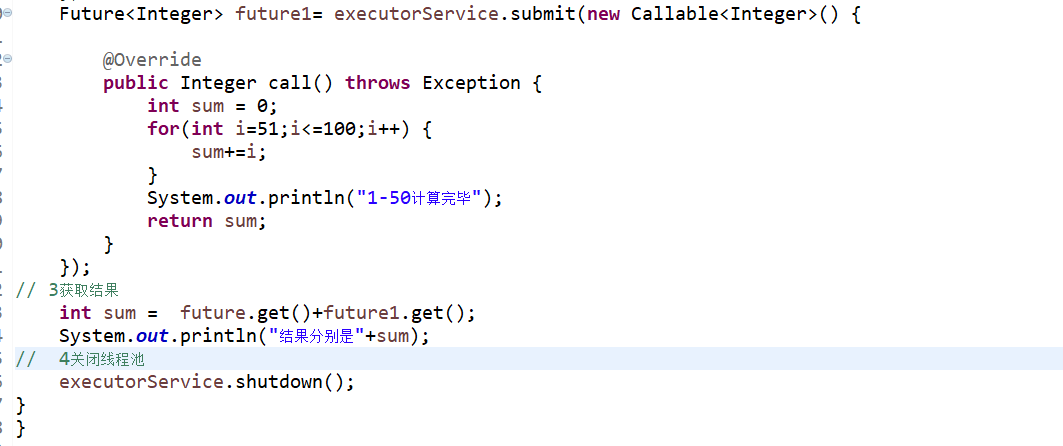


Future接口

Future:表示将要完成的任务结果

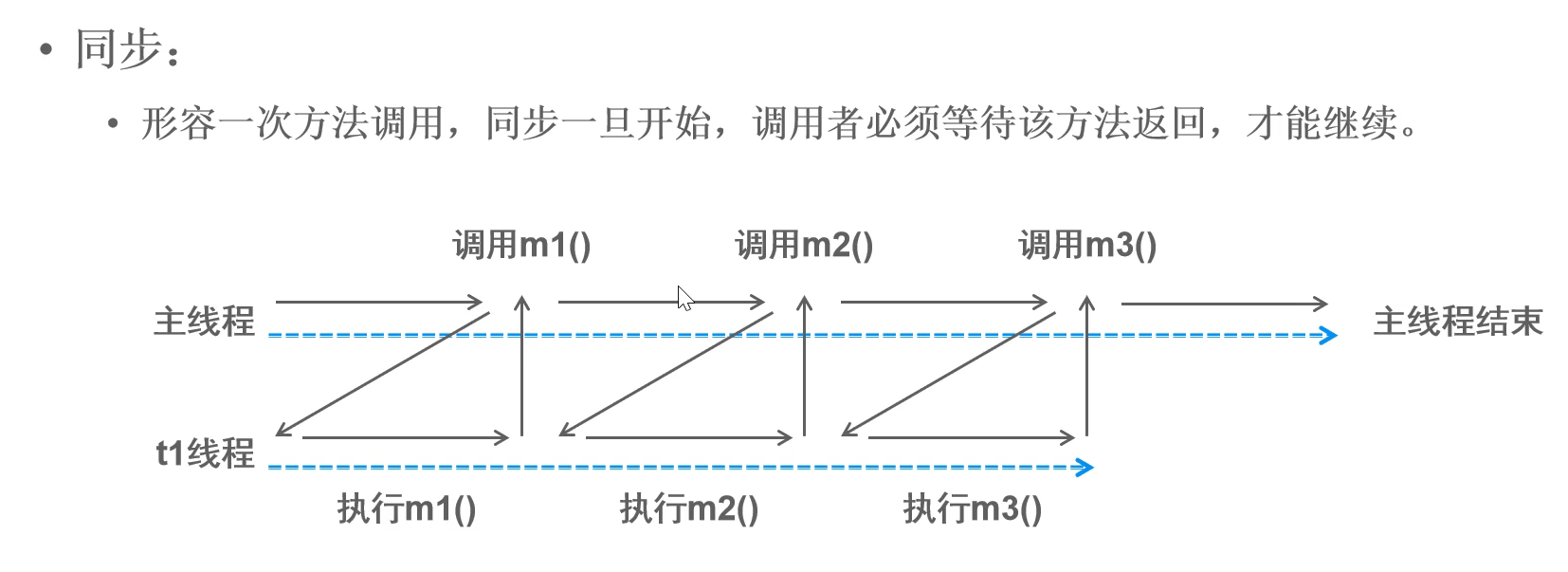
需求：使用两个线程，并发计算1~50，51~100的和，再进行统计

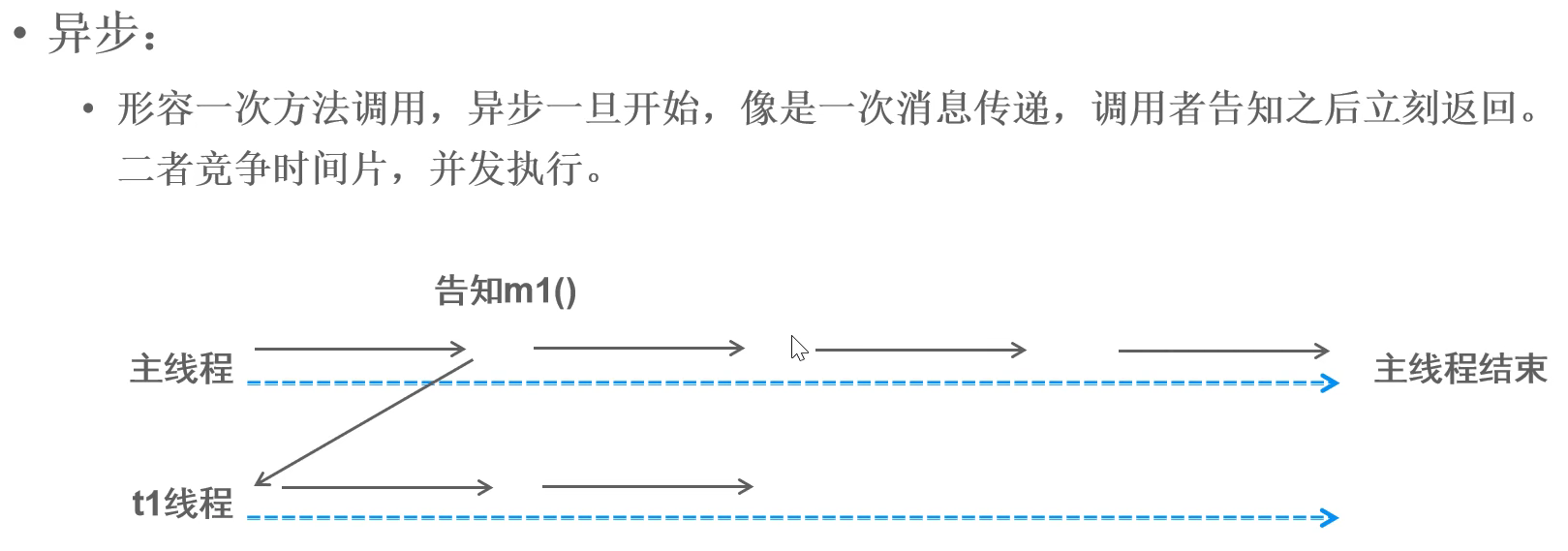




表示ExecutorService.submit()所返回的状态结果，就是call()结果

方法：V get()以阻塞形式等待Future中的异步处理结果（call()的返回值）





lock接口

与synchronized比较，显示定义，结构更灵活

提供更多实用性的方法，功能强大、性能优越

常用方法：

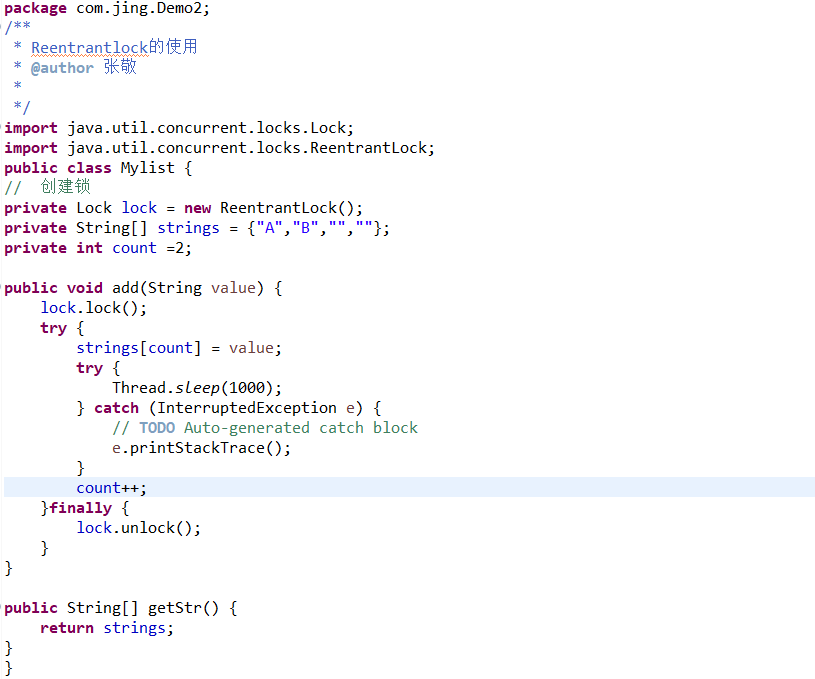
void lock();//获取锁，如锁本占用，则等待。

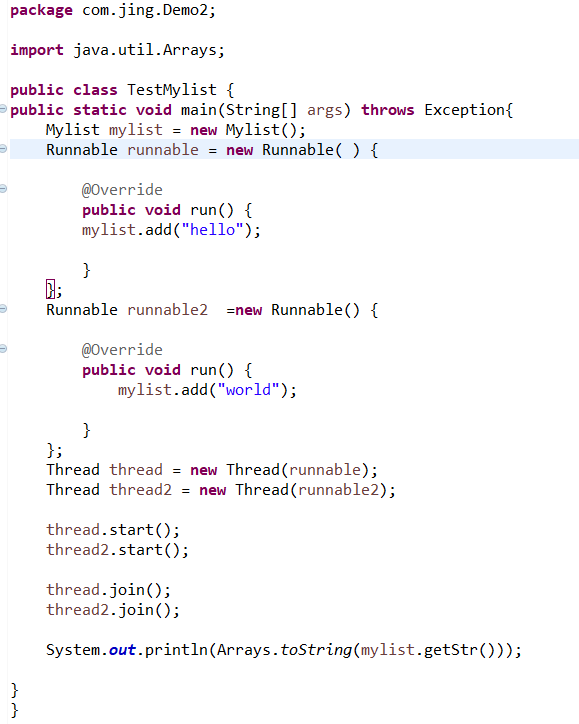
boolean trylock();尝试获取锁(成功返回true.失败返回false,不阻塞)

void unlock();释放锁

重入锁

Reentrantlock:lock接口的实现类，与synchronizedzed一样的互斥锁功能





读写锁

ReentranReadWriteLock:

一种支持一写多读的同步锁，读写分离，可分别分配读锁、写锁

支持多次分配读锁，使多个读操作可以并发执行

互斥规则：

写-写：互斥，阻塞

读-写：互斥，读阻塞写、写阻塞读

读-读：不互斥、不阻塞

在读操作远远高于写操作的环境中，可在保障线程安全的情况下，提高运行效率。

