# 线程的实现

在java中实现多线程，比在C或者C++中实现多线程要方便的多。

java提供了相关的API的支持多线程编程。

## 一、实现线程的方式

### 1、继承java.lang.Thread类

```
package com.psfd.thread;

//自己写一个类，继承Thread类，重写其run方法。
//逻辑处理，都要写在run方法中
public class ThreadA extends Thread {

    private String name;

    public ThreadA(String name) {
        super();
        this.name = name;
    }

    @Override
    public void run() {
        for (int i = 0; i < 5; i++) {
            System.out.println(name + "第 " + (i+1) +" 次执行");
        }
    }

}
```

调用：

```
package thread;

import com.psfd.thread.ThreadA;

public class Demo {
    public static void main(String[] args) {
        System.out.println("主线程main启动");

        //创建2个线程，
        Thread thread1 = new ThreadA("A线程");
        Thread thread2 = new ThreadA("B线程");
```

```
        //调用线程start方法，启动线程。
        //调用start方法，并不是说线程就启动了。而是线程处于就绪状态。
        //真正执行，是调用线程run方法
        thread1.start();
        thread2.start();

        System.out.println("主线程结束");
    }
}
```

运行结果：
第一次：
主线程main启动
主线程结束
A线程第 1 次执行
B线程第 1 次执行
A线程第 2 次执行
B线程第 2 次执行
A线程第 3 次执行
B线程第 3 次执行
A线程第 4 次执行
B线程第 4 次执行
A线程第 5 次执行
B线程第 5 次执行

第二次：
主线程main启动
主线程结束
A线程第 1 次执行
A线程第 2 次执行
A线程第 3 次执行
A线程第 4 次执行
A线程第 5 次执行
B线程第 1 次执行
B线程第 2 次执行
B线程第 3 次执行
B线程第 4 次执行
B线程第 5 次执行

多线程的情况下，调用的顺序完全是随机的。
因为操作系统对线程的调度，采用是时间片的抢占式调度。

## 2、实现java.lang.Runnable接口

```
package com.psfd.thread;
```

```
public class ThreadB implements Runnable {

    private String name;

    public ThreadB(String name) {
        this.name = name;
    }

    @Override
    public void run() {
        for (int i = 0; i < 5; i++) {
            System.out.println(name + "第 " + (i + 1) + " 次执行");
        }
    }

}
```

调用

```
package thread;

import com.psfd.thread.ThreadA;
import com.psfd.thread.ThreadB;

public class Demo {
    public static void main(String[] args) {
        System.out.println("主线程main启动");
        //继承thread类
        Thread threadA = new ThreadA("A线程");
        //实现Runnable
        Thread threadB = new Thread(new ThreadB("B线程"));

        threadA.start();
        threadB.start();

        System.out.println("主线程结束");
    }
}
```

# 作业：

1、模拟售票

三个售票窗口同时出售20张票

程序分析：(1)票数要使用同一个静态值

设计思路：(1)创建一个售票窗口类Station，继承Thread，重写run方法，在run方法里面执行售票操作！

2、设计2个线程，1个线程每次对i加1，另一个线程每次对i减去1

i是一个静态值。

# 二、复习线程的概念

### 1、进程：

对应电脑上的程序。一个进程，对应一个程序的运行。并不是说一个程序只有一个进程，也可能会有多个。比

如：chrome浏览器。

进程是操作系统进行资源分配（内存、寄存器、显存）的基本单元；

### 2、线程：

程序执行的最小单元。

一个进程可能对应有多个线程。（在任务管理器中能看到线程数）

但是一个线程只属于一个进程。

单个进程中的多个线程，共享资源（内存、寄存器、显存）

### 3、单线程：

单个线程的程序执行。

### 4、多线程：

同时启动多个线程，每一个线程对应一个子任务（Runnable本身的意思就是“可运行的任务”）。

# 三、线程的同步

synchronized 同步的关键字。

线程的同步，简单的理解就是：加锁。如何理解加锁？上厕所！

上厕所的正确流程：进去，关门，锁门。完事后开锁，出来，走人。

与线程同步类似的地方：锁门。

锁门：厕所是你家的啦。人家不能用了。资源被你抢占了。

其他人想用，必须要等你用完。

### 1、synchronized概念

synchronized是Java语言的关键字，当它用来修饰一个方法或者一个代码块的时候，能够保证在同一时刻最多只有一个线程执行该段代码

同步的锁是一种互斥锁。（排他锁），能到达到互斥访问目的的锁。

互斥：互相排斥。A拿到锁资源，B就不能访问。

在Java中，可以使用synchronized关键字来标记一个方法或者代码块，当某个线程调用该对象的synchronized方法或者访问synchronized代码块时，这个线程便获得了该对象的锁，其他线程暂时无法访问这个方法，只有等待这个方法执行完毕或者代码块执行完毕，这个线程才会释放该对象的锁，其他线程才能执行这个方法或者代码块。

## 2、synchronized弊端

弊端就是不好的地方。

当我们使用了synchronized，没错，确实解决了线程安全的问题，控制了并发。

但同时也带了非常严重的性能问题。当访问量较大的时候，后面访问会等待很长时间，用户体验极差。

所以：如果要使用synchronized，加锁的范围能少则少。

假设我要使用教室里的某一台电脑，并且不想被人打扰。

两种方案：

1、直接锁教室门。

会导致全班45号人，全部等着。

2、锁我要用的那一台电脑（麻子）。

锁住吴麻子的电脑。其他的44号人，照常使用。只有吴麻子受到影响。

所以：

在代码中使用synchronized的时候，如果能使用同步块，就不要使用同步方法。这叫减小锁的粒度，使代码更大程度的并发。

## 3、线程同步分为几种：

**synchronized代码块**

**synchronized方法**

被修饰的方法成为同步方法，其作用范围是整个方法，作用对象是调用这个方法的对象

同一对象的同一方法。

**synchronized静态方法**

修饰一个static静态方法，其作用范围是整个静态方法，作用对象是这个类的所有对象。

同一个类的所有实例的这个方法

**synchronized类**

不用管！

**synchronized()同步对象**

```
package thread;

public class Thread2 {

    public void m4t1() {
        synchronized (this) {
            int i = 5;
            while (i-- > 0) {
                System.out.println(Thread.currentThread().getName() + " : " + i);
                try {
                    Thread.sleep(500);
                } catch (InterruptedException ie) {
                }
            }
        }
    }

    public void m4t2() {
        int i = 5;
        while (i-- > 0) {
            System.out.println(Thread.currentThread().getName() + " : " + i);
            try {
                Thread.sleep(500);
            } catch (InterruptedException ie) {
            }
        }
    }

    public static void main(String[] args) {
        final Thread2 myt2 = new Thread2();

        Thread t1 = new Thread(new Runnable() {
            public void run() {
                myt2.m4t1();
            }
        }, "t1");
        Thread t2 = new Thread(new Runnable() {
            public void run() {
                myt2.m4t2();
            }
        }, "t2");
        t1.start();
        t2.start();
    }
}
```

当一个线程进入到了一个对象的同步资源的时候，其他的线程只能访问该对象的非同步资源。

该对象的其他同步资源，也不能访问。

## 4、什么叫线程安全？

即为：并发问题。

多个线程同一时刻访问同一资源。

## 5、并发和并行

并行：

齐头并进，你走你的阳光道，我过我的独木桥。

多个线程同一时刻访问各自不同的资源，一点问题都没有。平行线，不会有交点，不会造成资源的冲突。

并发：

多个线程访问同一个资源。

条条大路通罗马。多个线程同一时刻访问同一个资源，会造成资源的冲突。

作业：

1、将昨天的作业，完善。使其功能合理，不能出现数据混乱。

2、两个人AB通过一个账户，A在柜台取钱和B在ATM机取钱！

程序分析：

钱的数量要设置成一个静态的变量，两个人要取的同一个对象值

3、

1）、将若干个Student对象，若干个Teacher对象,写出到d:/data/a.txt中,

2）、启动两个线程，其中一个线程将该文件中所有的Student对象反序列化回来,装入List

另一个线程将所有的Teacher对象反序列化回来装入另一个List

然后将student信息和teacher信息打印到控制台

# 四、线程状态