第11天 多线程

今日内容介绍

* 多线程概述
* 线程实现
* 多线程安全问题产生 & 解决方案

# 多线程概述

学习多线程之前，我们先要了解几个关于多线程有关的概念。

A:进程：进程指正在运行的程序。确切的来说，当一个程序进入内存运行，即变成一个进程，进程是处于运行过程中的程序，并且具有一定独立功能。

B:线程：线程是进程中的一个执行单元，负责当前进程中程序的执行，一个进程中至少有一个线程。一个进程中是可以有多个线程的，这个应用程序也可以称之为多线程程序。

C:简而言之：一个程序运行后至少有一个进程，一个进程中可以包含多个线程

什么是多线程呢？即就是一个程序中有多个线程在同时执行。

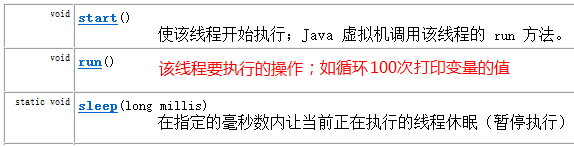
# 线程实现

## 实现线程一:继承Thread类

该如何创建线程呢？通过API中搜索，查到Thread类。通过阅读Thread类中的描述。Thread是程序中的执行线程。Java 虚拟机允许应用程序并发地运行多个执行线程。







A:创建线程的步骤：

1.定义一个类继承Thread。

2.重写run方法。

3.创建子类对象，就是创建线程对象。

4.调用start方法，开启线程并让线程执行，同时还会告诉jvm去调用run方法

### 案例代码一:

**package** com.itheima\_01;

**public** **class** MyThread **extends** Thread {

@Override

**public** **void** run() {

**for** (**int** i = 0; i < 100; i++) {

System.*out*.println(getName() + ":" + i);

}

}

}

**package** com.itheima\_01;

/\*

\* 多线程的实现方式：

\* 方式1：一种方法是将类声明为 Thread 的子类。该子类应重写 Thread 类的 run 方法。接下来可以分配并启动该子类的实例

\*

\* Thread

\* String getName() 返回该线程的名称。

\* void setName(String name) 改变线程名称，使之与参数 name 相同。

\*

\*

\* CPU执行程序的随机性

\*/

**public** **class** ThreadDemo2 {

**public** **static** **void** main(String[] args) {

//创建线程实例

MyThread mt = **new** MyThread();

//修改线程名字

mt.setName("张三");

//启动线程

mt.start();

//创建线程实例

MyThread mt2 = **new** MyThread();

mt2.setName("老王");

//启动线程

mt2.start();

}

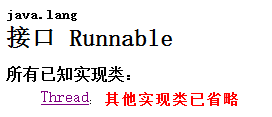
}

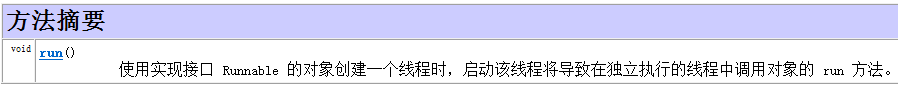
## 实现线程二:实现Runnable接口

创建线程的另一种方法是声明实现 Runnable 接口的类。该类然后实现 run 方法。然后创建Runnable的子类对象，传入到某个线程的构造方法中，开启线程。

为何要实现Runnable接口，Runable是啥玩意呢？继续API搜索。

查看Runnable接口说明文档：Runnable接口用来指定每个线程要执行的任务。包含了一个 run 的无参数抽象方法，需要由接口实现类重写该方法。







创建线程的步骤。

1、定义类实现Runnable接口。

2、覆盖接口中的run方法。。

3、创建Thread类的对象

4、将Runnable接口的子类对象作为参数传递给Thread类的构造函数。

5、调用Thread类的start方法开启线程。

### 案例代码二:

**package** com.itheima\_02;

**public** **class** MyThread2 **implements** Runnable {

**int** num;

**public** MyThread2(**int** num) {

**this**.num = num;

}

@Override

**public** **void** run() {

**for** (**int** i = 0; i < 100; i++) {

//Thread t = Thread.currentThread();

//System.out.println(t.getName() + ":" + i);

//链式编程

System.*out*.println(Thread.*currentThread*().getName() + ":" + i + num);

}

}

}

# 多线程安全问题产生&解决方案

## 多线程卖票案例

需求:用三个线程模拟三个售票窗口,共同卖100张火车票,每个线程打印出卖第几张票

### 案例代码三:

**package** com.itheima\_03;

**public** **class** TicketThread **implements** Runnable {

**int** tickets = 100;//火车票数量

@Override

**public** **void** run() {

//出售火车票

**while**(**true**) {

//当火车票小于0张，则停止售票

**if**(tickets > 0) {

/\*

\* t1,t2,t3

\* 假设只剩一张票

\* t1过来了，他一看有票，他就进来了，但是他突然肚子不舒服，然后他就去上卫生间了

\* t2也过来了，他一看也有票，他也进来了，但是他的肚子也不舒服，他也去上卫生间了

\*

\* t1上完了卫生间回来了，开始售票

\* tickets = 0;

\* t2也上完卫生间回来了，他也进行售票

\* tickets = -1;

\*

\*

\*/

**try** {

Thread.*sleep*(100);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.*out*.println(Thread.*currentThread*().getName() + ":" +tickets--);

}

}

}

}

## 多线程安全问题解决

### 使用同步代码块解决

格式:

synchronized(锁对象){

//需要同步的代码

}

#### 案例代码四:

**package** com.itheima\_04;

/\*

\* 问题出现的原因：

\* 要有多个线程

\* 要有被多个线程所共享的数据

\* 多个线程并发的访问共享的数据

\*

\* 在火车上上厕所

\* 张三来了，一看门是绿的，他就进去了，把门锁上了，门就变红了

\* 李四来了，一看门市红色的，他就只能憋着

\* 张三用完了厕所，把锁打开了，门就变成了绿色

\* 李四一看门变绿了，他就进去了，把门锁上，门就变红了

\* 王五来了，一看们是红色的，他也只能憋着

\* 李四用完测试了，把锁打开了，肚子又不舒服了，扭头回去了，又把门锁上了，

\*

\* synchronized:同步（锁），可以修饰代码块和方法，被修饰的代码块和方法一旦被某个线程访问，则直接锁住，其他的线程将无法访问

\*

\* 同步代码块：

\* synchronized(锁对象){

\*

\* }

\*

\* 注意：锁对象需要被所有的线程所共享

\*

\*

\* 同步：安全性高，效率低

\* 非同步：效率高，但是安全性低

\*

\*/

**public** **class** TicketThread **implements** Runnable {

**int** tickets = 100;//火车票数量

Object obj = **new** Object();

@Override

**public** **void** run() {

//出售火车票

**while**(**true**) {

**synchronized** (obj) {

**if**(tickets > 0) {

**try** {

Thread.*sleep*(100);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.*out*.println(Thread.*currentThread*().getName() + ":" +tickets--);

}

}

}

}

}

**package** com.itheima\_04;

**public** **class** TicktetTest {

**public** **static** **void** main(String[] args) {

//创建线程对象

TicketThread tt = **new** TicketThread();

Thread t = **new** Thread(tt);

t.setName("窗口1");

Thread t2 = **new** Thread(tt);

t2.setName("窗口2");

Thread t3 = **new** Thread(tt);

t3.setName("窗口3");

//启动线程对象

t.start();

t2.start();

t3.start();

}

}

### 使用同步方法解决

格式:

修饰符 synchronized 返回值 方法名(){

}

#### 案例代码五:

**package** com.itheima\_05;

/\*

\* 同步方法:使用关键字synchronized修饰的方法，一旦被一个线程访问，则整个方法全部锁住，其他线程则无法访问

\*

\* synchronized

\* 注意：

\* 非静态同步方法的锁对象是this

\* 静态的同步方法的锁对象是当前类的字节码对象

\*/

**public** **class** TicketThread **implements** Runnable {

**static** **int** *tickets* = 100;// 火车票数量

Object obj = **new** Object();

@Override

**public** **void** run() {

// 出售火车票

**while** (**true**) {

/\*synchronized (obj) {

method();

}\*/

//method();

*method2*();

}

}

**private** **synchronized** **void** method() {

**if** (*tickets* > 0) {

**try** {

Thread.*sleep*(100);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.*out*.println(Thread.*currentThread*().getName() + ":" + *tickets*--);

}

}

**private** **static** **synchronized** **void** method2() {

**if** (*tickets* > 0) {

**try** {

Thread.*sleep*(100);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.*out*.println(Thread.*currentThread*().getName() + ":" + *tickets*--);

}

}

}

**package** com.itheima\_05;

**public** **class** TicktetTest {

**public** **static** **void** main(String[] args) {

//创建线程对象

TicketThread tt = **new** TicketThread();

Thread t = **new** Thread(tt);

t.setName("窗口1");

Thread t2 = **new** Thread(tt);

t2.setName("窗口2");

Thread t3 = **new** Thread(tt);

t3.setName("窗口3");

//启动线程对象

t.start();

t2.start();

t3.start();

}

}

## 线程生命周期图

