

黑马程序员™  
[www.itheima.com](http://www.itheima.com)

传智播客旗下  
高端IT教育品牌

# 多线程



# 目录 Contents

- ◆ 实现多线程
- ◆ 线程同步
- ◆ 生产者消费者

# 1. 实现多线程

## 1.1 进程

进程：是正在运行的程序

- 是系统进行资源分配和调用的独立单位
- 每一个进程都有它自己的内存空间和系统资源



# 1. 实现多线程

## 1.2 线程

线程：是进程中的单个顺序控制流，是一条执行路径

- 单线程：一个进程如果只有一条执行路径，则称为单线程程序
- 多线程：一个进程如果有多条执行路径，则称为多线程程序

举例

- 记事本程序
- 扫雷程序



# 1. 实现多线程

## 1.3 多线程的实现方案

方案1: 继承Thread类

- 定义一个类MyThread继承Thread类
- 在MyThread类中重写run()方法
- 创建MyThread类的对象
- 启动线程

两个小问题:

- 为什么要重写run()方法?  
因为run()是用来封装被线程执行的代码
- run()方法和start()方法的区别?  
run(): 封装线程执行的代码, 直接调用, 相当于普通方法的调用  
start(): 启动线程; 然后由JVM调用此线程的run()方法



# 1. 实现多线程

## 1.4 设置和获取线程名称

Thread类中设置和获取线程名称的方法

- void `setName(String name)`: 将此线程的名称更改为等于参数 name
- String `getName()`: 返回此线程的名称
- 通过构造方法也可以设置线程名称

如何获取main()方法所在的线程名称?

- public static Thread `currentThread()`: 返回对当前正在执行的线程对象的引用

# 1. 实现多线程

## 1.5 线程调度

线程有两种调度模型

- 分时调度模型：所有线程轮流使用 CPU 的使用权，平均分配每个线程占用 CPU 的时间片
- 抢占式调度模型：优先让优先级高的线程使用 CPU，如果线程的优先级相同，那么会随机选择一个，优先级高的线程获取的 CPU 时间片相对多一些

Java使用的是抢占式调度模型

假如计算机只有一个 CPU，那么 CPU 在某一个时刻只能执行一条指令，线程只有得到CPU时间片，也就是使用权，才可以执行指令。所以说多线程程序的执行是有**随机性**，因为谁抢到CPU的使用权是不一定的

Thread类中设置和获取线程优先级的方法

- public final int **getPriority()**: 返回此线程的优先级
- public final void **setPriority(int newPriority)**: 更改此线程的优先级

线程默认优先级是**5**；线程优先级的范围是：**1-10**

线程优先级高仅仅表示线程获取的CPU时间片的**几率高**，但是要在次数比较多，或者多次运行的时候才能看到你想要的效果

# 1. 实现多线程

## 1.6 线程控制

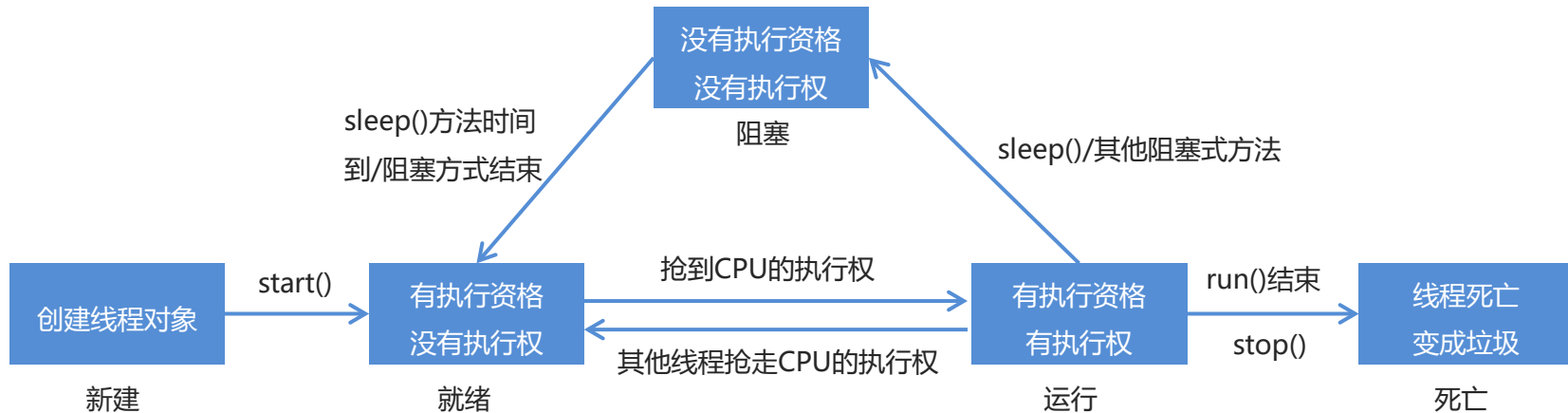
方法名	说明
static void sleep(long millis)	使当前正在执行的线程停留（暂停执行）指定的毫秒数
void join()	等待这个线程死亡
void setDaemon(boolean on)	将此线程标记为守护线程，当运行的线程都是守护线程时，Java虚拟机将退出





# 1. 实现多线程

## 1.7 线程生命周期





# 1. 实现多线程

## 1.8 多线程的实现方式

方式2：实现Runnable接口

- 定义一个类MyRunnable实现Runnable接口
- 在MyRunnable类中重写run()方法
- 创建MyRunnable类的对象
- 创建Thread类的对象，把MyRunnable对象作为构造方法的参数
- 启动线程

多线程的实现方案有两种

- 继承Thread类
- 实现Runnable接口

相比继承Thread类，实现Runnable接口的好处

- 避免了Java单继承的局限性
- 适合多个相同程序的代码去处理同一个资源的情况，把线程和程序的代码、数据有效分离，较好的体现了面向对象的设计思想

# 目录 Contents

- ◆ 实现多线程
- ◆ 线程同步
- ◆ 生产者消费者



## 2. 线程同步



### 案例：卖票

需求：某电影院目前正在上映国产大片，共有100张票，而它有3个窗口卖票，请设计一个程序模拟该电影院卖票

思路：

- ① 定义一个类SellTicket实现Runnable接口，里面定义一个成员变量：`private int tickets = 100;`
- ② 在SellTicket类中重写run()方法实现卖票，代码步骤如下
  - A：判断票数大于0，就卖票，并告知是哪个窗口卖的
  - B：卖了票之后，总票数要减1
  - C：票没有了，也可能有人来问，所以这里用死循环让卖票的动作一直执行
- ③ 定义一个测试类SellTicketDemo，里面有main方法，代码步骤如下
  - A：创建SellTicket类的对象
  - B：创建三个Thread类的对象，把SellTicket对象作为构造方法的参数，并给出对应的窗口名称
  - C：启动线程

## 2. 线程同步

### 2.1 卖票案例的思考

刚才讲解了电影院卖票程序，好像没有什么问题。但是在实际生活中，售票时出票也是需要时间的，所以，在售出一张票的时候，需要一点时间的延迟，接下来我们去修改卖票程序中卖票的动作：每次出票时间100毫秒，用sleep()方法实现

卖票出现了问题

- 相同的票出现了多次
- 出现了负数的票

问题原因：

- 线程执行的随机性导致的

### 2.2 卖票案例数据安全问题的解决

为什么出现问题?(这也是我们判断多线程程序是否会有数据安全问题的标准)

- 是否是多线程环境
- 是否有共享数据
- 是否有多条语句操作共享数据

如何解决多线程安全问题呢?

- 基本思想: 让程序没有安全问题的环境      破坏掉三条里的一条即可

怎么实现呢?

- 把多条语句操作共享数据的代码给锁起来, 让任意时刻只能有一个线程执行即可
- Java提供了同步代码块的方式来解决



## 2. 线程同步

### 2.3 同步代码块

锁多条语句操作共享数据，可以使用同步代码块实现

- 格式：

```
synchronized(任意对象) {  
    多条语句操作共享数据的代码  
}
```

- synchronized(任意对象)：就相当于给代码加锁了，任意对象就可以看成是一把锁

同步的好处和弊端

- 好处：解决了多线程的数据安全问题
- 弊端：当线程很多时，因为每个线程都会去判断同步上的锁，这是很耗费资源的，无形中会降低程序的运行效率



## 2. 线程同步

### 2.4 同步方法

**同步方法**：就是把synchronized关键字加到**方法**上

- 格式：

修饰符 **synchronized** 返回值类型 方法名(方法参数) { }

同步方法的锁对象是什么呢？

- **this**

**同步静态方法**：就是把synchronized关键字加到**静态方法**上

- 格式：

修饰符 static **synchronized** 返回值类型 方法名(方法参数) { }

同步静态方法的锁对象是什么呢？

- **类名.class**



### 2.5 线程安全的类

#### StringBuffer

- 线程安全，可变的字符序列
- 从版本JDK 5开始，被StringBuilder 替代。通常应该使用StringBuilder类，因为它支持所有相同的操作，但它更快，因为它不执行同步

#### Vector

- 从Java 2平台v1.2开始，该类改进了List接口，使其成为Java Collections Framework的成员。与新的集合实现不同，Vector被同步。如果不需要线程安全的实现，建议使用ArrayList代替Vector

#### Hashtable

- 该类实现了一个哈希表，它将键映射到值。任何非null对象都可以用作键或者值
- 从Java 2平台v1.2开始，该类进行了改进，实现了Map接口，使其成为Java Collections Framework的成员。与新的集合实现不同，Hashtable被同步。如果不需要线程安全的实现，建议使用HashMap代替Hashtable

### 2.6 Lock锁

虽然我们可以理解同步代码块和同步方法的锁对象问题，但是我们并没有直接看到在哪里加上了锁，在哪里释放了锁，为了更清晰的表达如何加锁和释放锁，JDK5以后提供了一个新的锁对象Lock

Lock实现提供比使用synchronized方法和语句可以获得更广泛的锁定操作

Lock中提供了获得锁和释放锁的方法

- void lock(): 获得锁
- void unlock(): 释放锁

Lock是接口不能直接实例化，这里采用它的实现类ReentrantLock来实例化

ReentrantLock的构造方法

- ReentrantLock(): 创建一个ReentrantLock的实例

# 目录 Contents

- ◆ 实现多线程
- ◆ 线程同步
- ◆ 生产者消费者



## 3. 生产者消费者

### 3.1 生产者消费者模式概述



## 3. 生产者消费者

### 3.1 生产者消费者模式概述

生产者消费者模式是一个十分经典的多线程协作的模式，弄懂生产者消费者问题能够让我们对多线程编程的理解更加深刻。所谓生产者消费者问题，实际上主要是包含了两类线程：

- 一类是生产者线程用于生产数据
- 一类是消费者线程用于消费数据

为了解耦生产者和消费者的关系，通常会采用共享的数据区域，就像是一个仓库

- 生产者生产数据之后直接放置在共享数据区中，并不需要关心消费者的行为
- 消费者只需要从共享数据区中去获取数据，并不需要关心生产者的行为



## 3. 生产者消费者

### 3.1 生产者消费者模式概述

为了体现生产和消费过程中的等待和唤醒，Java就提供了几个方法供我们使用，这几个方法在Object类中  
Object类的等待和唤醒方法：**不能只等待不唤醒，反之同理**

方法名	说明
void wait()	导致当前线程等待，直到另一个线程调用该对象的 notify()方法或 notifyAll()方法
void notify()	唤醒正在等待对象监视器的单个线程
void notifyAll()	唤醒正在等待对象监视器的所有线程

### 3.2 生产者消费者案例

生产者消费者案例中包含的类：

- 奶箱类(Box)：定义一个成员变量，表示第x瓶奶，提供存储牛奶和获取牛奶的操作
- 生产者类(Producer)：实现Runnable接口，重写run()方法，调用存储牛奶的操作
- 消费者类(Customer)：实现Runnable接口，重写run()方法，调用获取牛奶的操作
- 测试类(BoxDemo)：里面有main方法，main方法中的代码步骤如下
  - ① 创建奶箱对象，这是共享数据区域
  - ② 创建生产者对象，把奶箱对象作为构造方法参数传递，因为在这个类中要调用存储牛奶的操作
  - ③ 创建消费者对象，把奶箱对象作为构造方法参数传递，因为在这个类中要调用获取牛奶的操作
  - ④ 创建2个线程对象，分别把生产者对象和消费者对象作为构造方法参数传递
  - ⑤ 启动线程



传智播客旗下高端IT教育品牌