多线程概述

2018年8月25日 11:38

1. 进程和线程

a. 进程

所谓的进程其实就是一个程序或服务运行的过程在操作系统中的体现，操作系统中一个独立运行的程序或服务就是一个进程。

b. 多进程

现代的操作系统都可以支持同时运行多个程序和服务，体现在进程上，就是多个进程可以并行的执行，这称之为操作系统支持多进程。

c. 线程

一个进程的内部还可以划分出多个并行执行的过程，称之为在进程内部存在线程。

d. 多线程

一个进程内部可以存在多个线程，且这些线程可以并行的执行，这样的机制称之为进程支持多线程。

**计算中只有一个cpu，实际上同一个时刻，只能处理一个运算，但是由于cpu的运算速度非常的快，通过不停的切换处理的任务，从而使多个进程 多个线程 依次都能得到处理，处理的很快，切换的也很快，在人看来似乎这些进程和线程都在同时执行

**jvm本身也是进程，可以在其中开辟多个线程执行并发的任务，这样的开发多个线程的技术就称之为java的多线程技术。

Java中的多线程

2018年8月25日 14:08

1. Thread类

java是面向对象的语言，万物皆对象，在java中也是用对象来代表底层的物理线程，来方便我们操作。这样的线程对象归属于Thread类。

java.lang

类 Thread

2. 重要API

构造方法：

构造方法摘要
Thread() 分配新的 Thread 对象。
Thread(Runnable target) 分配新的 Thread 对象。

重要方法：

void	**start**() 使该线程开始执行；Java 虚拟机调用该线程的 run 方法。

3. 启动线程的两种方式

   a. 启动线程的方式1：

      i. 创建一个类 实现Runnable接口 在其中的run()方法中编写启动的线程要执行的代码

ii. 创建改类的对象，传入Thread的构造方法 ，创建Thread对象
iii. 调用Thread对象的start()方法，启动线程

b. 启动线程的方式2：
   i. 写一个类继承Thread,覆盖其中的run()方法，在其中编写启动的线程要执行的代码
   ii. 创建该Thread类的子类的对象
   iii. 调用Thread类的子类的对象的start()方法，启动线程

**两种线程启动方式的比较：

java是单继承的，继承的方式创建线程，将会占用了extends关键字，这在类本身需要继承其他类的情况下无法使用

java是多实现的，实现接口的数量没有线程，所以实现接口创建线程的方式并不会受到单继承的线程

4. 线程并发的细节

a. 主线程和其他线程比起来，唯一特殊的地方是它是程序的入口，除此之外没有任何高低 先后 差别。
b. 多个线程并发的过程中，线程在不停的无序的争夺cpu，某一时刻哪个线程抢夺到，哪个线程就执行，由于cpu运行速度非常快，看起来似乎这些线程都在并发的执行。
c. 线程是在进程内部执行的，进程内部只有任意一个非守护线程存活，进程就不会结束。

5. 关闭线程

stop方法可以显示的命令线程立即停止，但此方法具有固有的不安全性，所以目前已经被过时 废弃掉了。不要去使用。

void	**stop**() **已过时。** *该方法具有固有的不安全性。*

在废弃掉stop方法之后，jdk中也没有提供任何其他类似的方法，官方的建议是，应该由程序本身提供相应的开关，来控制线程的运行和停止。通常由一个静态的布尔类型来作为这种开关。

6. Thread中的其他常用方法

线程优先级相关的字段，本质上是int类型的常量值，取值范围为1 - 10 ，值越大优先级越高

字段摘要	
static int	**MAX_PRIORITY** 线程可以具有的最高优先级。
static int	**MIN_PRIORITY** 线程可以具有的最低优先级。
static int	**NORM_PRIORITY** 分配给线程的默认优先级。

static Thread	**currentThread**() 返回对当前正在执行的线程对象的引用。
long	**getId**() 返回该线程的标识符。
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。
int	**getPriority**() 返回线程的优先级。
void	**setPriority**(int newPriority)

	更改线程的优先级。
Thread.State	**getState**() 返回该线程的状态。
static void	**sleep**(long millis) 在指定的毫秒数内让当前正在执行的线程休眠（暂停执行），此操作受到系统计时器和调度程序精度和准确性的影响。

Java的多线程并发安全问题

2018年8月25日 15:30

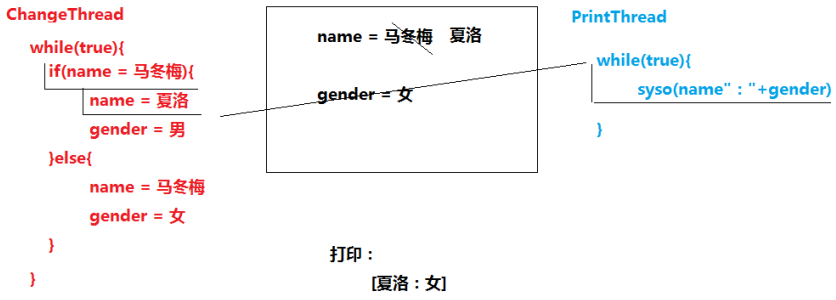
1. 多线程并发安全问题

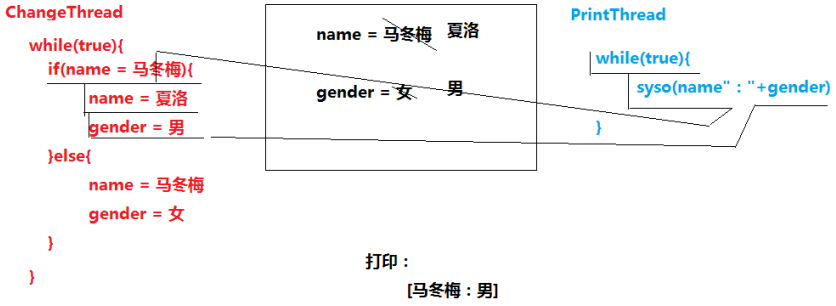
多线程环境下，多个线程是并发执行的，并且给予无序的cpu的争夺机制，线程的执行顺序是不确定的，此时如果多个线程同时去操作共享资源，就有可能因为线程的无序执行，产生一些意外的情况，这种问题就统称为多线程并发安全问题。

案例 - 修改并打印：

```
package cn.tedu.thread;

/**
 * 多线程并发安全问题
 */
public class ThreadDemo05 {
    public static String name = "马冬梅";
    public static String gender = "女";

    public static void main(String[] args) {
        new Thread(new PrintThread()).start();
        new Thread(new ChangeThread()).start();
    }
}

class ChangeThread implements Runnable{
    @Override
    public void run() {
        while(true){
            if("马冬梅".equals(ThreadDemo05.name)){
                ThreadDemo05.name = "夏洛";
                ThreadDemo05.gender = "男";
            }else{
                ThreadDemo05.name = "马冬梅";
                ThreadDemo05.gender = "女";
            }
```

```
            }
        }
    }

    class PrintThread implements Runnable{
        @Override
        public void run() {
            while(true){
                System.out.println("姓名："+ThreadDemo05.name+",性
                别："+ThreadDemo05.gender);
            }
        }
    }
```

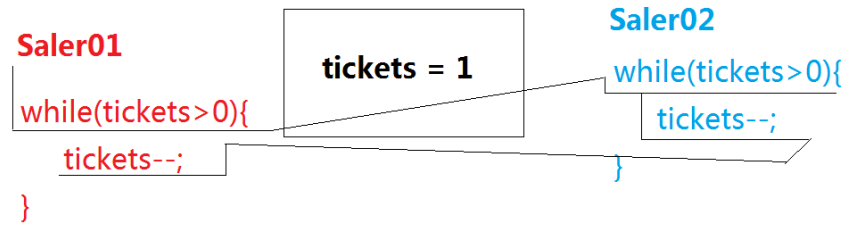




案例 - 卖火车票：

```
    package cn.tedu.thread;

    public class ThreadDemo06 {
        public static int tickets = 10;
        public static void main(String[] args) {
            new Thread(new Saler()).start();
            new Thread(new Saler()).start();
        }
    }

    class Saler implements Runnable{
```

```
        @Override
        public void run() {
            while(ThreadDemo06.tickets>0){
                ThreadDemo06.tickets--;
                System.out.println(Thread.currentThread().getId()+"窗口，卖出去了一张票。。剩余票数为"+ThreadDemo06.tickets);
                try {
                    Thread.sleep(10);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    }
```



2. 多线程并发安全问题产生的条件

a. 有共享资源

b. 有多线程并发操作了共享资源

c. 有多线程并发操作了共享资源 且涉及到了修改操作

3. 解决多线程并发安全问题

解决多线程并发安全问题的关键，就是破坏产生多线程并发安全问题的条件

禁止共享资源 -- ThreadLocal

禁止多线程并发操作 -- Syncronized

禁止修改 -- ReadWriteLock

4. Syncronized同步代码块使用

原理：

通过控制并发线程无法同时操作共享资源，从而实现线程安全问题的解决。

细节：

在存在多线程并发安全问题的场景下，可以使用Syncronized代码块，将产生多线程并发安全问题的代码包裹起来，并选择一个锁对象。

锁对象可以任意的选择，但是要保证，多个并发的线程操作的都是同一个锁对象。

锁对象并没有实际的作用，唯一的作用是在锁对象身上会标记一个锁的状态 - [关锁状态] [开锁状态]

在线程执行到同步代码块时，会在当前锁对上上检查，如果是锁是[开锁状态]，则进入同步代码块，同时将锁状态改为[关锁状态]

此时再有其他线程试图获取该对象上的锁时，锁已经是[关锁状态]，无法获得到锁，线程只能在锁下等待，进入阻塞状态。

直到之前得到锁的线程将Syncronized代码块执行完，释放锁，重新将锁变为 [开锁状态]，等待的线程，才可以从阻塞状态中恢复，重新参与锁的争夺。

因此，同一时刻，只能有一个并发线程执行可能造成线程安全问题的代码，破坏了多线程并发的状态，从而解决多线程并发安全问题

同步代码块的结构：

```
Syncronized(锁对象){
    要同步的代码
}
```

锁对象的选择的原则：

任意对象都可以作为锁对象使用

但要保证，所有的并发线程都应该能够访问到该所对象，且访问的必须是同一个锁对象

常用的锁对象：

自己创建一个专用的对象

使用共享资源作为锁对象

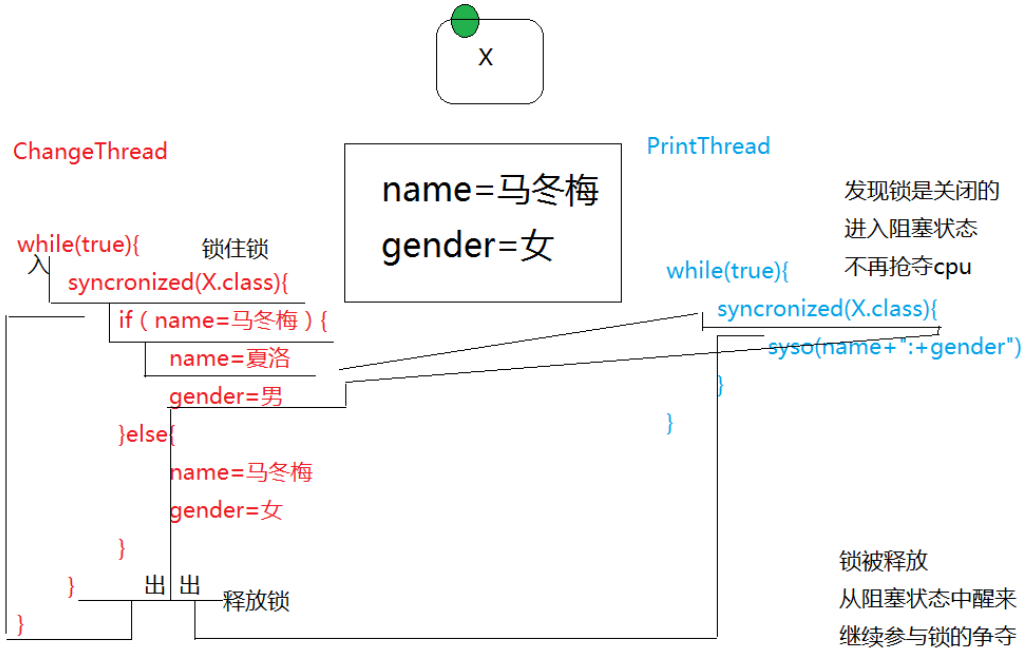
使用类的字节码对象作为锁对象

案例：利用同步代码块 解决上面的 修改打印案例中的 线程并发安全问题

代码：

```
package cn.tedu.thread;

/**
 * 多线程并发安全问题
 */
public class ThreadDemo05 {
    public static String name = "马冬梅";
    public static String gender = "女";

    public static void main(String[] args) {
        new Thread(new PrintThread()).start();
        new Thread(new ChangeThread()).start();
    }
}

class ChangeThread implements Runnable{
    @Override
    public void run() {
        while(true){
            synchronized (ThreadDemo05.class) {
                if("马冬梅".equals(ThreadDemo05.name)){
                    ThreadDemo05.name = "夏洛";
                    ThreadDemo05.gender = "男";
                }else{
                    ThreadDemo05.name = "马冬梅";
                    ThreadDemo05.gender = "女";
                }
            }
        }
    }
}

class PrintThread implements Runnable{
    @Override
    public void run() {
        while(true){
```

```
                synchronized (ThreadDemo05.class) {
                    System.out.println("姓名：" + ThreadDemo05.name + ",
                    性别：" + ThreadDemo05.gender);
                }
            }
        }
    }
```



案例：利用同步代码块 解决上面的 卖火车票案例中的 线程并发安全问题

代码：

```
package cn.tedu.thread;

public class ThreadDemo06 {
    public static int tickets = 10;
    public static void main(String[] args) {
        new Thread(new Saler()).start();
        new Thread(new Saler()).start();
    }
}

class Saler implements Runnable{
    @Override
    public void run() {
        while(ThreadDemo06.tickets>0){
            synchronized (ThreadDemo06.class) {
                ThreadDemo06.tickets--;
                System.out.println(Thread.currentThread().getId() + "窗
                口，卖出去了一张票。。剩余票数为" +
```

```
                    ThreadDemo06.tickets);
                }
                try {
                    Thread.sleep(10);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    }
```

作业

2018年8月25日 17:04

IO流

- 打印流 - 了解
- Properties - 掌握
- 序列化 - 掌握

多线程 - 掌握

- 多线程的概念
- 创建线程的方式
- 线程的执行细节
- 多线程关闭方式
- 多线程并发安全问题