Java 程序设计

第14章 多线程

田英鑫 tyx@hit.edu.cn
哈尔滨工业大学软件学院

第14章 多线程

& 本章导读

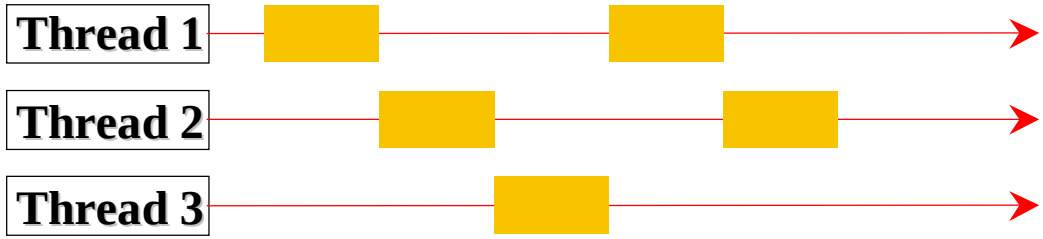
- 14.1 多线程简介
- 14.2 Thread 类
- 14.3 Runnable 接口
- 14.4 控制线程
- 14.5 线程的优先级
- 14.6 线程的生命周期
- 14.7 多线程同步
- 14.8 管道流

14.1 多线程简介

- 什么是多线程？

多个线程在多个CPU上运行

多个线程可以分享单个CPU



14.1 多线程简介

- 什么是多线程？
  - 一个线程（thread）是程序中完成一个任务的有始有终的执行流
  - 多线程可以使程序反应更快、交互性更强，并能够提高执行效率
  - 可以通过扩展Thread类或实现Runnable接口来创建线程

14.2 Thread 类

n 扩展 Thread 类创建线程

```
public class CustomThread extends Thread
{
    public CustomThread(…)
    {
    }
     //覆盖run方法
    public void run()
    {
    }
}
```

```
public class Client
{
    public someMethod()
    {
        //创建线程对象
        CustomThread thread = new CustomThread();
        //启动线程
        thread.start();  //启动线程对象的run方法
    }
}
```

14.2 Thread 类

- 示例：使用 Thread 类创建并运行线程
  - 在这个例子中，创建并运行如下三个线程:
    - 第一个的线程打印100次字母a
    - 第二个的线程打印100次字母b
    - 第三个线程打印整数1到100

参见: TestThreads.java

14.3 Runnable 接口

n 实现 Runnable 接口创建线程

```
public class CustomThread
    implements Runnable
{
    public CustomThread(…)
    {
    }
     //覆盖run方法
    public void run()
    {
    }
}
```

```
public class Client
{
    public someMethod()
    {
        //创建实现Runnable接口的类的实例
        CustomThread customThread = new CustomThread();
        //创建线程对象
        Thread thread = new Thread(customThread);
        //启动线程
        thread.start(); //启动线程对象的run方法
    }
}
```

14.3 Runnable 接口

- 示例：使用 Runnable 接口创建并运行线程
  - 在这个例子中，创建并运行如下三个线程:
    - 第一个的线程打印100次字母a
    - 第二个的线程打印100次字母b
    - 第三个线程打印整数1到100

参见: TestRunnable.java

14.4 控制线程

- 控制线程
  - void run()
    - Java运行系统调用该方法来执行线程。必须覆盖该方法并且提供线程执行的代码
  - void start()
    - 启动线程，引起对run()方法的调用。客户类中的可运行对象调用本方法
  - void stop()
    - 停止线程

14.4 控制线程

- 控制线程
  - void suspend()
    - 挂起线程
    - 使用 resume() 方法唤醒进程
  - void resume()
    - 唤醒用suspend() 方法挂起的进程
  - static void sleep(long millis) throws InterruptedException
    - 将可运行的对象置为休眠状态，休眠时间为指定的毫秒
    - 该方法是一个静态方法

14.4 控制线程

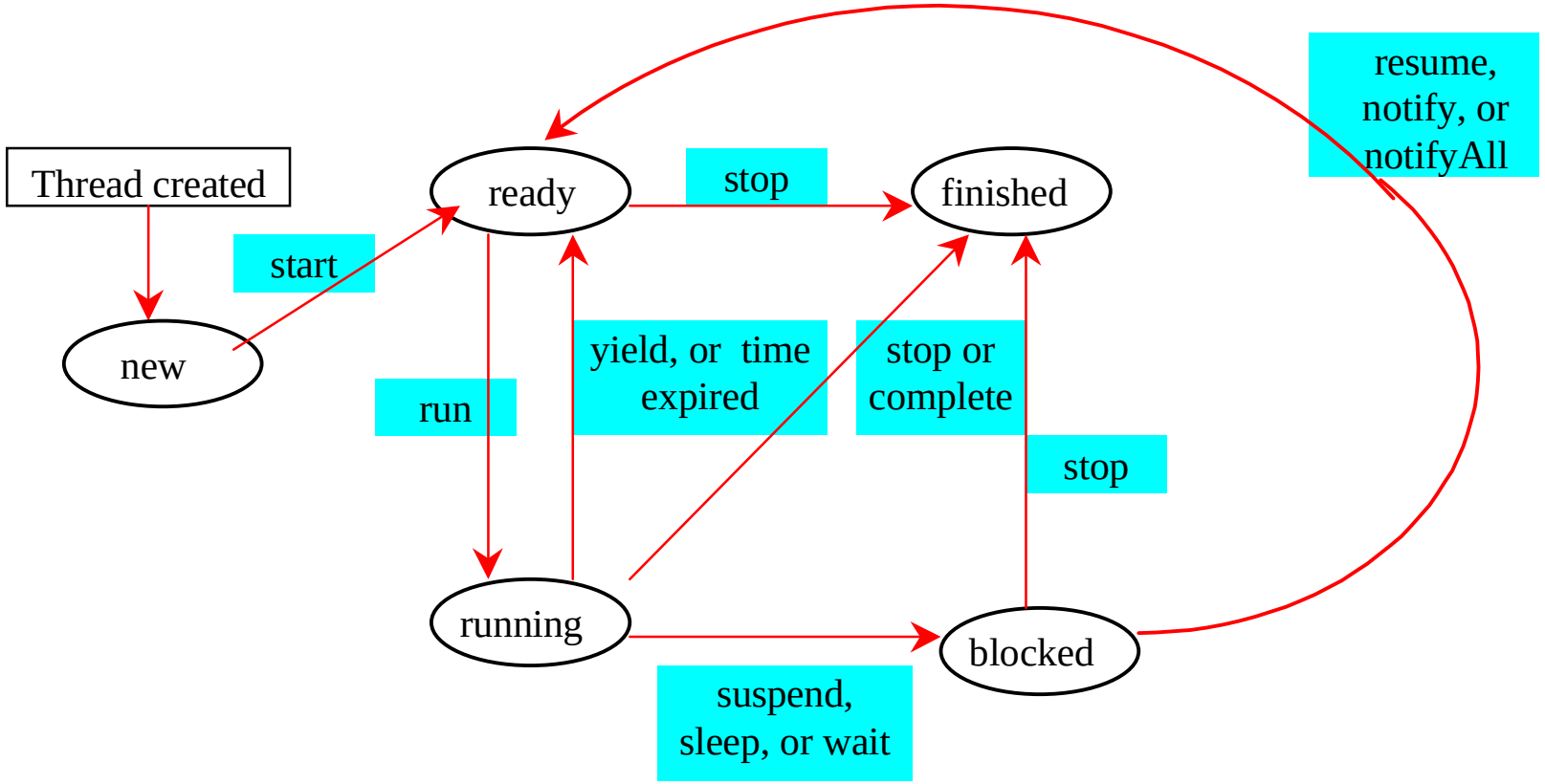
- 线程组
  - 构造线程组
    - ThreadGroup g = new ThreadGroup("timer thread group");
  - 使用Thread 构造方法，将一个线程放到线程组中
    - Thread t = new Thread(g, new ThreadClass(), "This thread");
  - 使用activeCount() 确定组里有多少个线程处于运行阶段
    - System.out.println("The number of runnable threads in the group " + g.activeCount());

14.5 线程的优先级

- 线程的优先级
  - Java给每个线程指定一个优先级，默认情况下线程继承生成它的线程的优先级
  - 改变和得到线程优先级
    - setPriority
    - getPriority
  - 优先级包含的一些常量
    - Thread.MIN_PRIORITY
    - Thread.MAX_PRIORITY
    - Thread.NORM_PRIORITY

14.6 线程的生命周期

线程状态



14.6 线程的生命周期

- 线程状态
  - 新线程态（New Thread）
    - 产生一个Thread对象就生成一个新线程。当线程处于“新线程”状态时，仅仅是一个空线程对象，它还没有分配到系统资源。因此只能启动或终止它。任何其他操作都会引发异常
  - 可运行态（Runnable）（就绪状态+运行状态）
    - start方法产生运行线程所必须的资源,调度线程执行,并且调用线程的run方法。在这时线程处于可运行态
    - 该状态不称为运行态是因为这时的线程并不总是一直占用处理机。特别是对于只有一个处理机的PC而言,任何时刻只能有一个处于可运行态的线程占用处理 机
    - Java通过调度来实现多线程对处理机的共享

14.6 线程的生命周期

- 线程状态
  - 非运行态（Not Runnable）（阻塞状态）
    - 当以下事件发生时,线程进入非运行态
      - suspend方法被调用
      - sleep方法被调用
      - 线程使用wait来等待条件变量
      - 线程处于I/O等待
  - 死亡态（Dead）（结束状态）
    - 当run方法返回或别的线程调用stop方法，线程进入死亡态
    - 通常Applet使用它的stop方法来终止它产生的所有线程

14.7 多线程同步

- 资源冲突
  - 如果一个共享资源被多个线程同时访问，可能会遭到破坏
  - 例如：两个unsynchronized 线程同时进入相同的银行账户，资源将会引起冲突

Step	balance	thread[i]	thread[j]
1	0	newBalance =bank.getBalance() + 1;	
2	0		newBalance =bank.getBalance() + 1;
3	1	bank.setBalance(newBalance);	
4	1		bank.setBalance(newBalance);

14.7 多线程同步

- 示例：演示资源冲突
  - 在这个例子中，创建并启动了100个线程，每个线程都往储钱罐中投入一便士，假定开始时储钱罐是空的

参见: PiggyBankWithoutSync.java

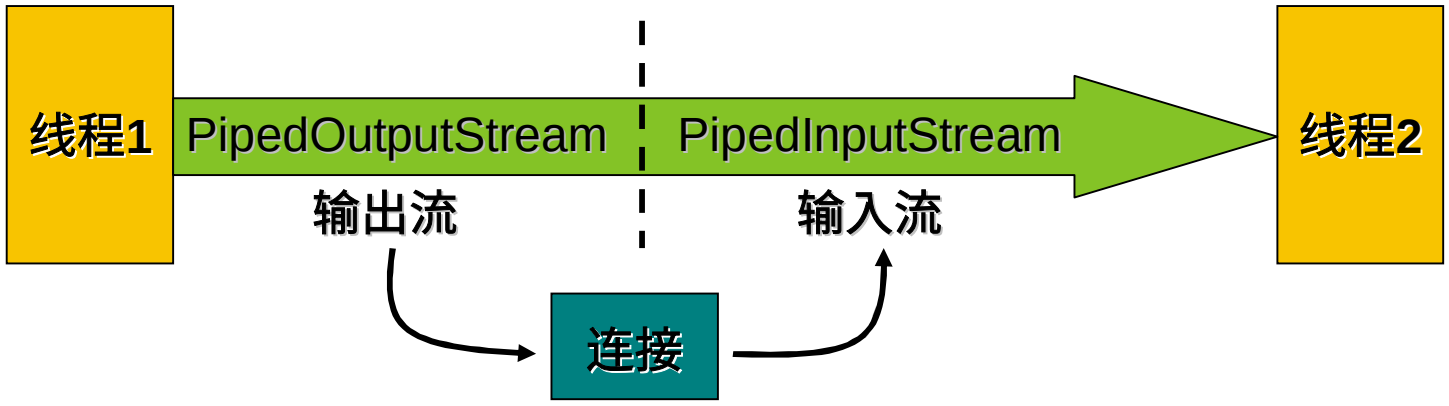
14.7 多线程同步

- 避免资源冲突
  - 为避免资源冲突，Java使用synchronized关键字使方法通信同步
  - 重写上面的程序如下

参见: PiggyBankWithSync.java

14.8 管道流

- 什么是管道流？
  - 管道流用来把一个线程的输出连接到另一个线程的输入
  - 管道输入流作为一个通信管道的接收端，管道输出流作为发送端
  - 管道流必须是输入和输出并用，使用前两者必须进行连接



14.8 管道流

- 管道流的连接
  - 在构造方法中连接
    - PipedInputStream(PipedOutputStream pos)
    - PipedOutputStream(PipedInputStream pis)
  - 通过connect方法连接
    - 在PipedInputStream中，调用方法 connect(PipedOutputStream pos)
    - 在PipedOutputStream中，调用方法 connect(PipedInputStream pis)

参见: PipedStreamTest.java

Any Question?