[**线程的创建**](http://www.blogjava.net/junglesong/archive/2008/02/22/181356.html)

一般来说，在没有线程的帮助下，程序在一个时间段只能执行一段代码，其它代码段只有在等待它完成后才能执行。该程序的处理流程从头到尾只有一条线，这样的程序我们称之为单线程程序（Single Thread Program）

典型的单线程程序：  
public class SingleThreadProgram{  
  public static void main(String[] args){  
    for(int i=0;i<1000;i++){  
      System.out.print("SingleThreadProgram");  
    }  
  }  
}

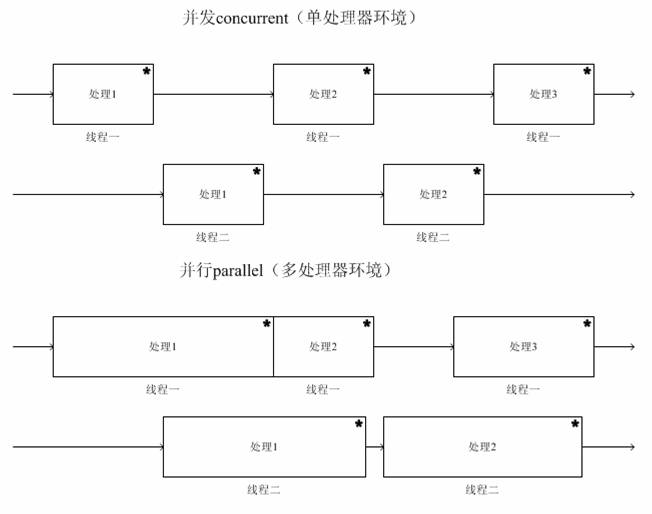
**多线程程序**

当程序由一个以上的线程所构成时，称此程序为多线程程序（Multithread Program），java从设计伊始就把程序的多线程能力列入了考虑范围。  
典型的多线程程序有：

1）GUI应用程序，我们目前做的Swing桌面程序就属于此类。  
2）较花费时间的I/O处理,一般来说,文件和网络的输入/输出处理比较花费时间,如果在这段无法进行其它处理,则程序性能会大打折扣,遇到这种情况首先要想到用多线程解决问题.  
3）多连接网络处理。

**并发(Concurrent)与并行(Parallel)**

当有多个线程在操作时,如果系统只有一个CPU,则它根本不可能真正同时进行一个以上的线程,它只能把CPU运行时间划分成若干个时间段,再将时间段分配给各个线程执行,在一个时间段的线程代码运行时,其它线程处于挂起状态.这种方式我们称之为并发(Concurrent).  
当系统有一个以上CPU时,则线程的操作有可能非并发.当一个CPU执行一个线程时,另一个CPU可以执行另一个线程,两个线程互不抢占CPU资源,可以同时进行,这种方式我们称之为并行(Parallel)

  
  
**多线程在并发和并行环境中的不同作用**

在并发环境时,多线程不可能真正充分利用CPU,节约运行时间,它只是以”挂起->执行->挂起”的方式以很小的时间片分别运行各个线程,给用户以每个线程都在运行的错觉.在这种环境中,多线程程序真正改善的是系统的响应性能和程序的友好性.  
在并行环境中, 一个时刻允许多个线程运行,这时多线程程序才真正充分利用了多CPU的处理能力, 节省了整体的运行时间.在这种环境中,多线程程序能体现出它的四大优势:充分利用CPU,节省时间,改善响应和增加程序的友好性.

PS:在多核时代来临后,开发多线程程序的能力更是每个程序员都该具备的.

**创建多线程程序**

创建多线程程序我们通常有两种方法：  
1）让类继承java.lang.Thread,这种方法优势在于调用稍微方便,一般用于后台批处理程序的场合,但劣势是类无法再继承别的类。  
2）让类实现接口java.lang.Runnable，这种方法调用时需要借助Thread的帮助，稍显麻烦，但优势在于对类继承体系没有影响，这是使用线程时最常用的方法。  
两种方法的线程执行部分都在run（）函数中，它们的效率没有差别。

**多线程程序创建和启动示例**

创建线程

// 继承Thread类  
public class Thread1 extends Thread{  
  public void run(){  
    while(true){  
      System.out.println("<Thread1 extends Thread>");  
    }  
  }  
}

// 实现Runnable接口  
public class Thread2 implements Runnable{  
  public void run(){  
    while(true){  
      System.out.println("<Thread2 implements Runnable>");  
    }  
  }  
}

启动线程

public class Main{  
  public static void main(String[] args){  
    // 启动线程1,Thread1直接继承自java.lang.Thread类  
    Thread1 th1=new Thread1();  
    th1.start();  
      
    // 启动线程2，thread2实现自java.lang.Runnable接口  
    Thread2 thread2=new Thread2();  
    Thread th2=new Thread(thread2);  
    th2.start();  
      
    while(true){  
      System.out.println("<Main Thread>");  
    }  
  }  
}

**概念解析Start和Run**

public void run()  
这个函数容纳线程启动后执行的代码块,线程启动起来,run函数中的代码会得到执行.

Thead.start()  
这是启动一个线程的方法,调用了这个方法后,线程才会得到执行.

**取得线程执行的结果**

通过观察run函数的签名public void run()我们可以发现,它既没有输入参数,也没有返回值,那如何取得线程的返回值呢?一般来说我们有三种办法:  
1)让线程修改公有变量,如某类的静态公有字段.这种方式古老而危险,最好不要采用.  
2)轮询线程执行结果,线程执行的结果放在线程类的一个字段中,外界不断通过轮询去查看执行结果.这种方式会浪费很多时间,结果也不可靠,不建议采用.  
3)回调方式,把调用方的指针通过线程类的构造函数传入线程类的一个字段中,当线程执行完取得结果后再通过这个字段反向调用调用方的函数.这是取得线程执行结果的最佳解决方案.

下面请看**回调方式的实现**.

Boss类  
这个类用于启动Secretary线程去查找文件, findFile()是启动线程并查找的函数, giveBossResult(String file,String reult)是供Secretary类回调的函数.

public class Boss{  
  private String name;  
    
  public Boss(String name){  
    this.name=name;  
  }  
    
  public void giveBossResult(String file,String reult){  
    if(reult!=null){  
      System.out.println("文件"+file+"序列号等于："+reult);  
    }  
    else{  
      System.out.println("无法找到文件"+file);  
    }  
  }  
    
  public void findFile(){     
    Map<String,String> files=new Hashtable<String,String>();      
    files.put("001", "员工花名册");  
    files.put("002", "企业收支");  
    files.put("003", "客户花名录");  
    files.put("004", "对手状况分析");  
    files.put("005", "当月收支");  
    files.put("006", "市场份额分析");  
    files.put("007", "大连酒店一览");  
    files.put("008", "娱乐场所名录");  
    files.put("009", "关系单位联系名录");  
      
    Secretary andy=new Secretary("Andy",this,"员工花名册",files);  
    Thread th1=new Thread(andy);  
    th1.start();  
      
    Secretary cindy=new Secretary("cindy",this,"上市情况分析",files);  
    Thread th2=new Thread(cindy);  
    th2.start();  
  }  
    
  public static void main(String[] args){  
    Boss boss=new Boss("Bill");  
    boss.findFile();  
  }  
}

Secretary类

这个类是进行多线程查找文件的类,查找的结果通过回调方法告知Boss实例.  
Boss实例,查找的文件名,查找的集合都通过Secretary类的构造函数传进来.

public class Secretary implements Runnable{  
  private String name;  
  private Boss boss;  
  private String file;  
  private Map<String,String> files;  
    
  public Secretary(String name,Boss boss,String file,Map<String,String> files){  
    this.name=name;  
    this.boss=boss;  
    this.file=file;  
    this.files=files;  
  }  
    
  public void run(){  
    for(Map.Entry<String,String> entry:files.entrySet()){  
         if(entry.getValue().equals(file)){  
           boss.giveBossResult(file,entry.getKey());  
           return;  
         }  
    }  
      
    boss.giveBossResult(file,null);  
  }  
}