**Java 多线程编程**

Java给多线程编程提供了内置的支持。一个多线程程序包含两个或多个能并发运行的部分。程序的每一部分都称作一个线程，并且每个线程定义了一个独立的执行路径。

多线程是多任务的一种特别的形式，但多线程使用了更小的资源开销。

这里定义和线程相关的另一个术语 - 进程：一个进程包括由操作系统分配的内存空间，包含一个或多个线程。一个线程不能独立的存在，它必须是进程的一部分。一个进程一直运行，直到所有的非守候线程都结束运行后才能结束。

多线程能满足程序员编写高效率的程序来达到充分利用CPU的目的。

**一个线程的生命周**

线程经过其生命周期的各个阶段。下图显示了一个线程完整的生命周期。



* **新建状态:**

使用 **new** 关键字和 **Thread** 类或其子类建立一个线程对象后，该线程对象就处于新建状态。它保持这个状态直到程序 **start()**这个线程。

* **就绪状态:**

当线程对象调用了start()方法之后，该线程就进入就绪状态。就绪状态的线程处于就绪队列中，要等待JVM里线程调度器的调度。

* **运行状态:**

如果就绪状态的线程获取 CPU 资源，就可以执行 **run()**，此时线程便处于运行状态。处于运行状态的线程最为复杂，它可以变为阻塞状态、就绪状态和死亡状态。

* **阻塞状态:**

如果一个线程执行了sleep（睡眠）、suspend（挂起）等方法，失去所占用资源之后，该线程就从运行状态进入阻塞状态。在睡眠时间已到或获得设备资源后可以重新进入就绪状态。

* **死亡状态:**

一个运行状态的线程完成任务或者其他终止条件发生时，该线程就切换到终止状态。

**线程的优先级**

每一个Java线程都有一个优先级，这样有助于操作系统确定线程的调度顺序。

Java线程的优先级是一个整数，其取值范围是1 （Thread.MIN\_PRIORITY ） - 10 （Thread.MAX\_PRIORITY ）。

默认情况下，每一个线程都会分配一个优先级NORM\_PRIORITY（5）。

具有较高优先级的线程对程序更重要，并且应该在低优先级的线程之前分配处理器资源。但是，线程优先级不能保证线程执行的顺序，而且非常依赖于平台。

**创建一个线程**

Java提供了两种创建线程方法：

* 通过实现Runable接口；
* 通过继承Thread类本身。

**通过实现Runnable接口来创建线程**

创建一个线程，最简单的方法是创建一个实现Runnable接口的类。

为了实现Runnable，一个类只需要执行一个方法调用run()，声明如下：

publicvoid run()

你可以重写该方法，重要的是理解的run()可以调用其他方法，使用其他类，并声明变量，就像主线程一样。

在创建一个实现Runnable接口的类之后，你可以在类中实例化一个线程对象。

Thread定义了几个构造方法，下面的这个是我们经常使用的：

Thread(Runnable threadOb,String threadName);

这里，threadOb 是一个实现Runnable 接口的类的实例，并且 threadName指定新线程的名字。

新线程创建之后，你调用它的start()方法它才会运行。

void start();

**实例**

下面是一个创建线程并开始让它执行的实例：

// 创建一个新的线程

class NewThread implements Runnable {

Thread t;

NewThread() {

// 创建第二个新线程

t = new Thread(this, "Demo Thread");

System.out.println("Child thread: " + t);

t.start(); // 开始线程

}

// 第二个线程入口

public void run() {

try {

for(int i = 5; i > 0; i--) {

System.out.println("Child Thread: " + i);

// 暂停线程

Thread.sleep(50);

}

} catch (InterruptedException e) {

System.out.println("Child interrupted.");

}

System.out.println("Exiting child thread.");

}

}

public class ThreadDemo {

public static void main(String args[]) {

new NewThread(); // 创建一个新线程

try {

for(int i = 5; i > 0; i--) {

System.out.println("Main Thread: " + i);

Thread.sleep(100);

}

} catch (InterruptedException e) {

System.out.println("Main thread interrupted.");

}

System.out.println("Main thread exiting.");

}

}

编译以上程序运行结果如下：

Child thread: Thread[Demo Thread,5,main]

Main Thread: 5

Child Thread: 5

Child Thread: 4

Main Thread: 4

Child Thread: 3

Child Thread: 2

Main Thread: 3

Child Thread: 1

Exiting child thread.

Main Thread: 2

Main Thread: 1

Main thread exiting.

**通过继承Thread来创建线程**

创建一个线程的第二种方法是创建一个新的类，该类继承Thread类，然后创建一个该类的实例。

继承类必须重写run()方法，该方法是新线程的入口点。它也必须调用start()方法才能执行。

**实例**

// 通过继承 Thread 创建线程

class NewThread extends Thread {

NewThread() {

// 创建第二个新线程

super("Demo Thread");

System.out.println("Child thread: " + this);

start(); // 开始线程

}

// 第二个线程入口

public void run() {

try {

for(int i = 5; i > 0; i--) {

System.out.println("Child Thread: " + i);

// 让线程休眠一会

Thread.sleep(50);

}

} catch (InterruptedException e) {

System.out.println("Child interrupted.");

}

System.out.println("Exiting child thread.");

}

}

public class ExtendThread {

public static void main(String args[]) {

new NewThread(); // 创建一个新线程

try {

for(int i = 5; i > 0; i--) {

System.out.println("Main Thread: " + i);

Thread.sleep(100);

}

} catch (InterruptedException e) {

System.out.println("Main thread interrupted.");

}

System.out.println("Main thread exiting.");

}

}

编译以上程序运行结果如下：

Child thread: Thread[Demo Thread,5,main]

Main Thread: 5

Child Thread: 5

Child Thread: 4

Main Thread: 4

Child Thread: 3

Child Thread: 2

Main Thread: 3

Child Thread: 1

Exiting child thread.

Main Thread: 2

Main Thread: 1

Main thread exiting.

**Thread 方法**

下表列出了Thread类的一些重要方法：

|  |  |
| --- | --- |
| **序号** | **方法描述** |
| 1 | **public void start()** 使该线程开始执行；**Java** 虚拟机调用该线程的 run 方法。 |
| 2 | **public void run()** 如果该线程是使用独立的 Runnable 运行对象构造的，则调用该 Runnable 对象的 run 方法；否则，该方法不执行任何操作并返回。 |
| 3 | **public final void setName(String name)** 改变线程名称，使之与参数 name 相同。 |
| 4 | **public final void setPriority(int priority)**  更改线程的优先级。 |
| 5 | **public final void setDaemon(boolean on)** 将该线程标记为守护线程或用户线程。 |
| 6 | **public final void join(long millisec)** 等待该线程终止的时间最长为 millis 毫秒。 |
| 7 | **public void interrupt()** 中断线程。 |
| 8 | **public final boolean isAlive()** 测试线程是否处于活动状态。 |

测试线程是否处于活动状态。 上述方法是被Thread对象调用的。下面的方法是Thread类的静态方法。

|  |  |
| --- | --- |
| **序号** | **方法描述** |
| 1 | **public static void yield()** 暂停当前正在执行的线程对象，并执行其他线程。 |
| 2 | **public static void sleep(long millisec)** 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。 |
| 3 | **public static boolean holdsLock(Object x)** 当且仅当当前线程在指定的对象上保持监视器锁时，才返回 true。 |
| 4 | **public static Thread currentThread()** 返回对当前正在执行的线程对象的引用。 |
| 5 | **public static void dumpStack()** 将当前线程的堆栈跟踪打印至标准错误流。 |

**实例**

如下的ThreadClassDemo 程序演示了Thread类的一些方法：

// 文件名 : DisplayMessage.java

// 通过实现 Runnable 接口创建线程

public class DisplayMessage implements Runnable

{

private String message;

public DisplayMessage(String message)

{

this.message = message;

}

public void run()

{

while(true)

{

System.out.println(message);

}

}

}

// 文件名 : GuessANumber.java

// 通过继承 Thread 类创建线程

public class GuessANumber extends Thread

{

private int number;

public GuessANumber(int number)

{

this.number = number;

}

public void run()

{

int counter = 0;

int guess = 0;

do

{

guess = (int) (Math.random() \* 100 + 1);

System.out.println(this.getName()

+ " guesses " + guess);

counter++;

}while(guess != number);

System.out.println("\*\* Correct! " + this.getName()

+ " in " + counter + " guesses.\*\*");

}

}

// 文件名 : ThreadClassDemo.java

public class ThreadClassDemo

{

public static void main(String [] args)

{

Runnable hello = new DisplayMessage("Hello");

Thread thread1 = new Thread(hello);

thread1.setDaemon(true);

thread1.setName("hello");

System.out.println("Starting hello thread...");

thread1.start();

Runnable bye = new DisplayMessage("Goodbye");

Thread thread2 = new Thread(bye);

thread2.setPriority(Thread.MIN\_PRIORITY);

thread2.setDaemon(true);

System.out.println("Starting goodbye thread...");

thread2.start();

System.out.println("Starting thread3...");

Thread thread3 = new GuessANumber(27);

thread3.start();

try

{

thread3.join();

}catch(InterruptedException e)

{

System.out.println("Thread interrupted.");

}

System.out.println("Starting thread4...");

Thread thread4 = new GuessANumber(75);

thread4.start();

System.out.println("main() is ending...");

}

}

运行结果如下，每一次运行的结果都不一样。

Starting hello thread...

Starting goodbye thread...

Hello

Hello

Hello

Hello

Hello

Hello

Hello

Hello

Hello

Thread-2 guesses 27

Hello

\*\* Correct! Thread-2 in 102 guesses.\*\*

Hello

Starting thread4...

Hello

Hello

..........remaining result produced.

**线程的几个主要概念:**

在多线程编程时，你需要了解以下几个概念：

* 线程同步
* 线程间通信
* 线程死锁
* 线程控制：挂起、停止和恢复

**多线程的使用**

有效利用多线程的关键是理解程序是并发执行而不是串行执行的。例如：程序中有两个子系统需要并发执行，这时候就需要利用多线程编程。

通过对多线程的使用，可以编写出非常高效的程序。不过请注意，如果你创建太多的线程，程序执行的效率实际上是降低了，而不是提升了。

请记住，上下文的切换开销也很重要，如果你创建了太多的线程，CPU花费在上下文的切换的时间将多于执行程序的时间！