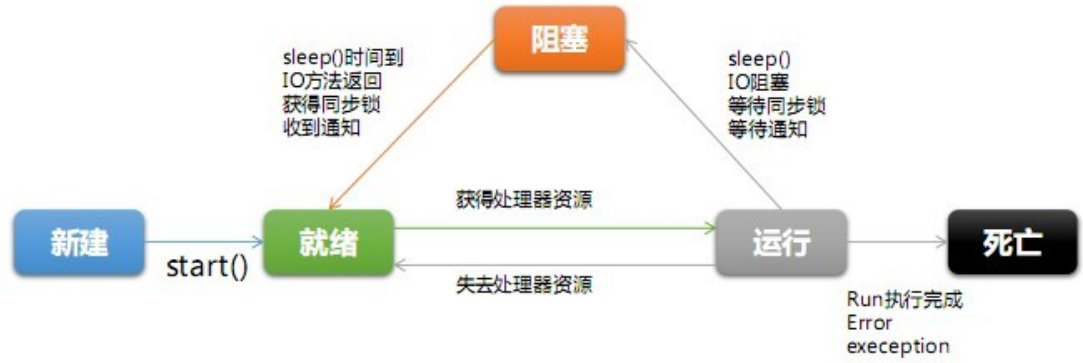
java多线程学习

笔记本:	java-多线程		
创建时间:	2019/5/3 16:49	**更新时间:**	2019/5/3 17:49
作者:	ANNER		
URL:	https://blog.csdn.net/Evankaka/article/details/44153709		

进程和线程的关系

- 进程：每个进程都有独立的代码和数据空间（进程上下文），进程间的切换会有较大的开销，一个进程包含1--n个线程。（**进程是资源分配的最小单位**）
- 线程：同一类线程共享代码和数据空间，每个线程有独立的运行栈和程序计数器(PC)，线程切换开销小。（**线程是cpu调度的最小单位**）

线程和进程一样分为五个阶段：创建、就绪、运行、阻塞、终止



- 多进程是指操作系统能同时运行多个任务（程序）。
- 多线程是指在同一程序中有多个顺序流在执行。

实现简单的多线程程序

1. 继承Thread类的方法
   、

```java
package java_base.thread.base;

public class Thread1 extends Thread {
    private String name;
    public Thread1(String name){
        this.name=name;
    }
    @Override
    public void run(){
        for(int i=0;i<5;i++){
            System.out.println(name+"运行:"+i);
            try{
                sleep( millis: (int)Math.random()*10);
            }catch (InterruptedException e){
                e.printStackTrace();
            }
        }
    }

    public static void main(String[] args) {
        Thread1 t1=new Thread1( name: "A");
        Thread1 t2=new Thread1( name: "B");
        t1.start();
        t2.start();
    }
}
```

注意：start()方法的调用后并不是立即执行多线程代码，而是使得该线程变为可运行态（Runnable），什么时候运行是由操作系统决定的。

2. 采用Runnable也是非常常见的一种，我们只需要重写run方法即可。下面也来看个实例。

```java
package java_base.thread.base;

public class Thread2 implements Runnable {
    private String name;
    public Thread2(String name){
        this.name=name;
    }
    public void run() {
        for(int i=0;i<5;i++){
            System.out.println(name+"运行 "+i);
            try{
                Thread.sleep( millis: (int)Math.random()*10);
            }catch (InterruptedException e){
                e.printStackTrace();
            }
        }
    }

    public static void main(String[] args) {
       new Thread(new Thread2( name: "A")).start();
       new Thread(new Thread2( name: "B")).start();
    }
}
```

Thread2类通过实现Runnable接口，使得该类有了多线程类的特征。run（）方法是多线程程序的一个约定。所有的多线程代码都在run方法里面。Thread类实际上也是实现了Runnable接口的类。

在启动的多线程的时候，需要先通过Thread类的构造方法Thread(Runnable target) 构造出对象，然后调用Thread对象的start()方法来运行多线程代码。

实际上所有的多线程代码都是通过运行Thread的start()方法来运行的。因此，不管是扩展Thread类还是实现Runnable接口来实现多线程，最终还是通过Thread的对象的API来控制线程的，熟悉Thread类的API是进行多线程编程的基础。

实现Runnable接口比继承Thread类所具有的优势：

1）：适合多个相同的程序代码的线程去处理同一个资源

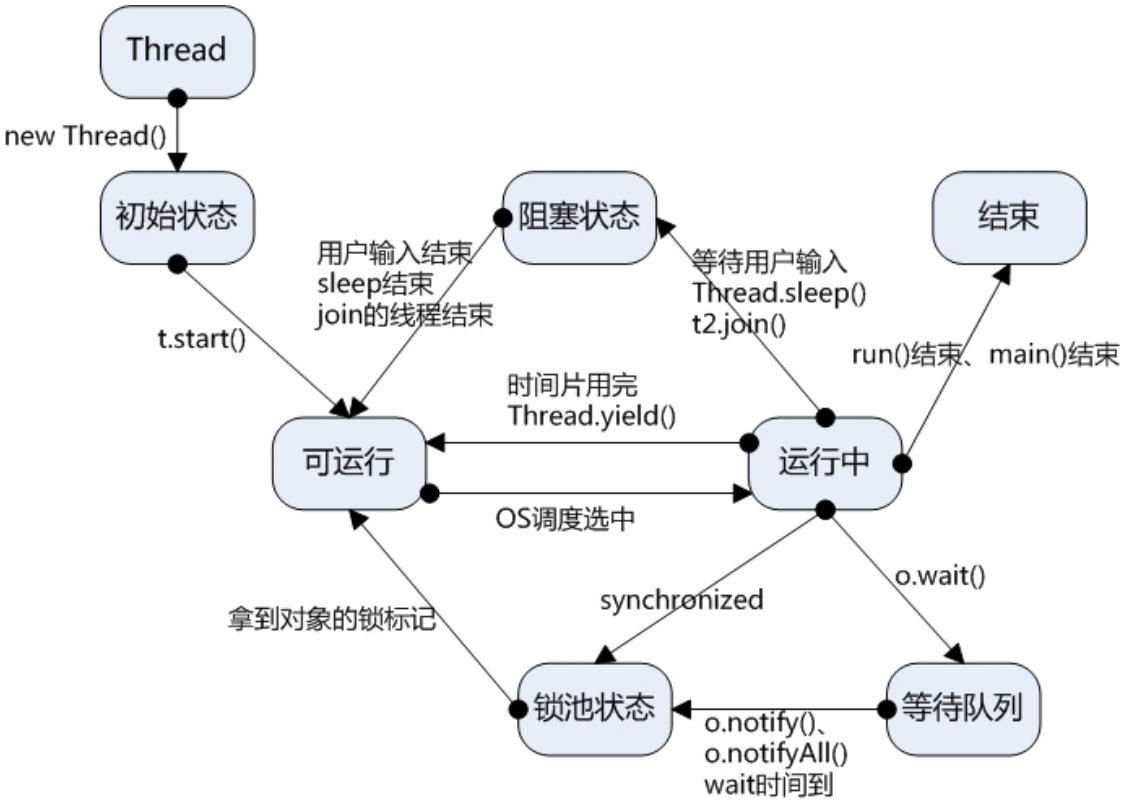
2）：可以避免java中的单继承的限制

3）：增加程序的健壮性，代码可以被多个线程共享，代码和数据独立

4）：线程池只能放入实现Runable或callable类线程，不能直接放入继承Thread的类

在java中，每次程序运行至少启动2个线程。一个是main线程，一个是垃圾收集线程。因为每当使用java命令执行一个类的时候，实际上都会启动一个ＪＶＭ，每一个ｊ ＶM实习在就是在操作系统中启动了一个进程。

线程状态切换



阻塞状态（Blocked）：阻塞状态是线程因为某种原因放弃CPU使用权，暂时停止运行。直到线程进入就绪状态，才有机会转到运行状态。阻塞的情况分三种：
（一）、等待阻塞：运行的线程执行wait()方法，JVM会把该线程放入等待池中。(wait会释放持有的锁)
（二）、同步阻塞：运行的线程在获取对象的同步锁时，若该同步锁被别的线程占用，则JVM会把该线程放入锁池中。
（三）、其他阻塞：运行的线程执行sleep()或join()方法，或者发出了I/O请求时，JVM会把该线程置为阻塞状态。当sleep()状态超时、

线程调度

Java线程有优先级，优先级高的线程会获得较多的运行机会

java线程的优先级用整数表示，取值范围是1~10，Thread类有以下三个静态常量：

- static int MAX_PRIORITY 线程可以具有的最高优先级，取值为10。
- static int MIN_PRIORITY 线程可以具有的最低优先级，取值为1。
- static int NORM_PRIORITY 分配给线程的默认优先级，取值为5

Thread类的setPriority()和getPriority()方法分别用来设置和获取线程的优先级。

每个线程都有默认的优先级。**主线程的默认优先级为Thread.NORM_PRIORITY。**

线程的优先级有继承关系，比如A线程中创建了B线程，那么B将和A具有相同的优先级。

JVM提供了10个线程优先级，但与常见的操作系统都不能很好的映射。如果希望程序能移植到各个操作系统中，应该仅仅使用Thread类有以下三个静态常量作为优先级，这样能保证同样的优先级采用了同样的调度方式。

常用函数说明

1. 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）
2. join是Thread类的一个方法，启动线程后直接调用，即join()的作用是：“等待该线程终止”，这里需要理解的就是该线程是指的主线程等待子线程的终止。也就是在**子线程调用了join()方法后面的代码，只有等到子线程结束了才能执行**
   为什么使用join?
   在很多情况下，主线程生成并起动了子线程，如果子线程里要进行大量的耗时的运算，主线程往往将于子线程之前结束，但是如果主线程处理完其他的事务后，需要用到子线程的处理结果，也就是主线程需要等待子线程执行完成之后再结束，这个时候就要用到join()方法了

```
    public static void main(String[] args) {
        Thread t1=new Thread(new TestJoin( name: "A"));
        Thread t2=new Thread(new TestJoin( name: "B"));
        t1.start();
        t2.start();
        try{
            t1.join();
        }catch (InterruptedException e){
            e.printStackTrace();
        }
        try{
            t2.join();
        }catch (InterruptedException e){
            e.printStackTrace();
        }
        System.out.println("主线程运行结束");
    }
}
```

3. yield()应该做的是让当前运行线程回到可运行状态，以允许具有相同优先级的其他线程获得运行机会。因此，使用yield()的目的是让相同优先级的线程之间能适当的轮转执行。但是，**实际中无法保证yield()达到让步目的，因为让步的线程还有可能被线程调度程序再次选中。**

```java
public class TestYield implements Runnable {
    private String name;
    public TestYield(String name){
        this.name=name;
    }
    public void run() {
       for(int i=0;i<50;i++){
           System.out.println(name+"运行"+i+"-------");
           if (i==30)Thread.yield();
       }
    }

    public static void main(String[] args) {
        Thread t1=new Thread(new TestYield( name: "A"));
        Thread t2=new Thread(new TestYield( name: "B"));
        t1.start();
        t2.start();
    }
}
```

sleep()和yield()的区别:

sleep()和yield()的区别):sleep()使当前线程进入停滞状态，所以执行sleep()的线程在指定的时间内肯定不会被执行；yield()只是使当前线程重新回到可执行状态，所以执行yield()的线程有可能在进入到可执行状态后马上又被执行。

4. **interrupt():不要以为它是中断某个线程！它只是线线程发送一个中断信号，让线程在无限等待时（如死锁时）能抛出抛出，从而结束线程，但是如果你吃掉了这个异常，那么这个线程还是不会中断的！**

线程同步