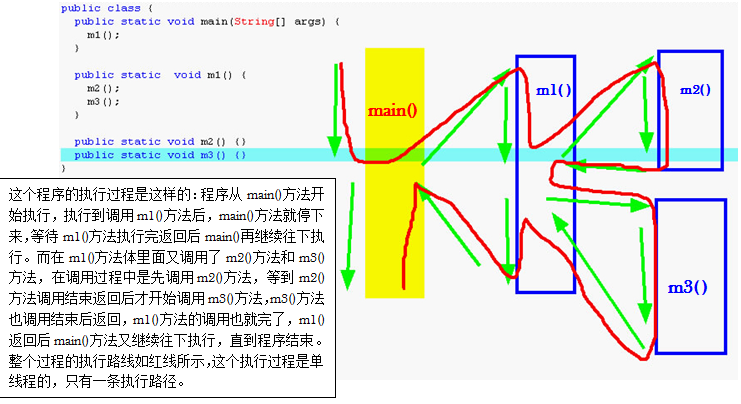
## 一、线程的基本概念D:\xxxxxxxxxxxxxxxx\download.png

线程理解：线程是一个程序里面不同的执行路径****

　　每一个分支都叫做一个线程，main()叫做主分支，也叫主线程。

　　程只是一个静态的概念，机器上的一个.class文件，机器上的一个.exe文件，这个叫做一个进程。程序的执行过程都是这样的：首先把程序的代码放到内存的代码区里面，代码放到代码区后并没有马上开始执行，但这时候说明了一个进程准备开始，进程已经产生了，但还没有开始执行，这就是进程，所以进程其实是一个静态的概念，它本身就不能动。平常所说的进程的执行指的是进程里面主线程开始执行了，也就是main()方法开始执行了。进程是一个静态的概念，在我们机器里面实际上运行的都是线程。

　　Windows操作系统是支持多线程的，它可以同时执行很多个线程，也支持多进程，因此Windows操作系统是支持多线程多进程的操作系统。Linux和Uinux也是支持多线程和多进程的操作系统。DOS就不是支持多线程和多进程了，它只支持单进程，**在同一个时间点只能有一个进程在执行，这就叫单线程**。

　　CPU难道真的很神通广大，能够同时执行那么多程序吗？不是的，CPU的执行是这样的：CPU的速度很快，一秒钟可以算好几亿次，因此CPU把自己的时间分成一个个小时间片，我这个时间片执行你一会，下一个时间片执行他一会，再下一个时间片又执行其他人一会，虽然有几十个线程，但一样可以在很短的时间内把他们通通都执行一遍，但对我们人来说，CPU的执行速度太快了，因此看起来就像是在同时执行一样，但实际上在一个时间点上，CPU只有一个线程在运行。

学习线程首先要理清楚三个概念：

1. 进程：进程是一个静态的概念
2. 线程：一个进程里面有一个主线程叫main()方法，是一个程序里面的，一个进程里面不同的执行路径。
3. 在同一个时间点上，一个CPU只能支持一个线程在执行。因为CPU运行的速度很快，因此我们看起来的感觉就像是多线程一样。

　　什么才是真正的多线程？如果你的机器是双CPU，或者是双核，这确确实实是多线程。

## 二、线程的创建和启动D:\xxxxxxxxxxxxxxxx\download.png

　　在JAVA里面，JAVA的线程是通过java.lang.Thread类来实现的，每一个Thread对象代表一个新的线程。创建一个新线程出来有两种方法：第一个是从Thread类继承，另一个是实现接口runnable。VM启动时会有一个由主方法(public static void main())所定义的线程，这个线程叫主线程。可以通过创建Thread的实例来创建新的线程。你只要new一个Thread对象，一个新的线程也就出现了。每个线程都是通过某个特定的Thread对象所对应的方法run()来完成其操作的，方法run()称为线程体。

**范例1：使用实现Runnable接口创建和启动新线程**

**开辟一个新的线程来调用run方法**

1 package cn.galc.test;

2

3 public class TestThread1{

4 public static void main(String args[]){

5 Runner1 r1 = new Runner1();//这里new了一个线程类的对象出来

6 //r1.run();//这个称为方法调用，方法调用的执行是等run()方法执行完之后才会继续执行main()方法

7 Thread t = new Thread(r1);//要启动一个新的线程就必须new一个Thread对象出来

8 //这里使用的是Thread(Runnable target) 这构造方法

9 t.start();//启动新开辟的线程，新线程执行的是run()方法，新线程与主线程会一起并行执行

10 for(int i=0;i<10;i++){

11 System.out.println("maintheod："+i);

12 }

13 }

14 }

15 /\*定义一个类用来实现Runnable接口，实现Runnable接口就表示这个类是一个线程类\*/

16 class Runner1 implements Runnable{

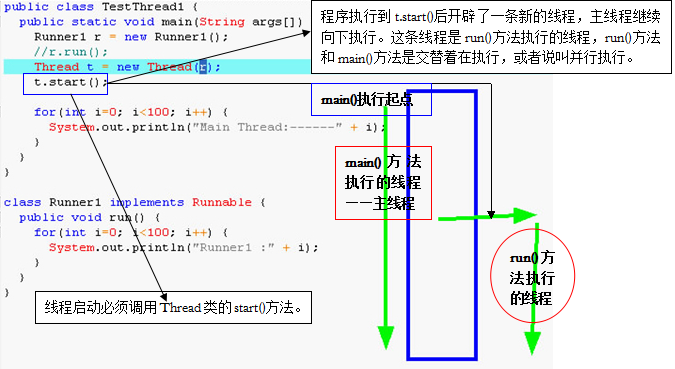
17 public void run(){

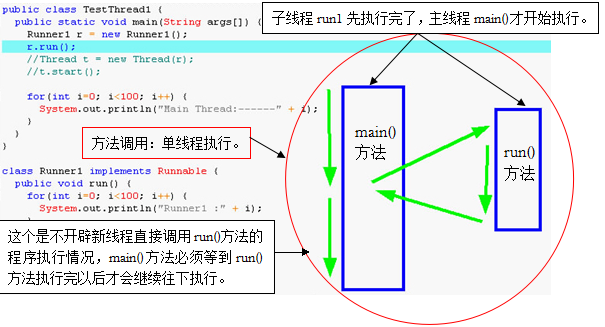
18 for(int i=0;i<10;i++){

19 System.out.println("Runner1："+i);

20 }

21 }

22 }****

**不开辟新线程直接调用run方法**

**范例2：继承Thread类，并重写其run()方法创建和启动新的线程**

1 package cn.galc.test;

2

3 /\*线程创建与启动的第二种方法：定义Thread的子类并实现run()方法\*/

4 public class TestThread2{

5 public static void main(String args[]){

6 Runner2 r2 = new Runner2();

7 r2.start();//调用start()方法启动新开辟的线程

8 for(int i=0;i<=10;i++){

9 System.out.println("mainMethod："+i);

10 }

11 }

12 }

13 /\*Runner2类从Thread类继承

14 通过实例化Runner2类的一个对象就可以开辟一个新的线程

15 调用从Thread类继承来的start()方法就可以启动新开辟的线程\*/

16 class Runner2 extends Thread{

17 public void run(){//重写run()方法的实现

18 for(int i=0;i<=10;i++){

19 System.out.println("Runner2："+i);

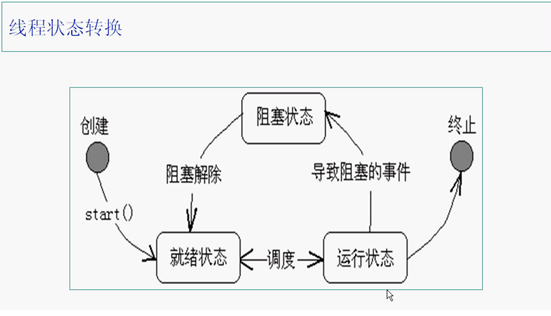
20 }

21 }

22 }

　　使用实现Runnable接口和继承Thread类这两种开辟新线程的方法的选择应该优先选择实现Runnable接口这种方式去开辟一个新的线程。因为接口的实现可以实现多个，而类的继承只能是单继承。因此在开辟新线程时能够使用Runnable接口就尽量不要使用从Thread类继承的方式来开辟新的线程。

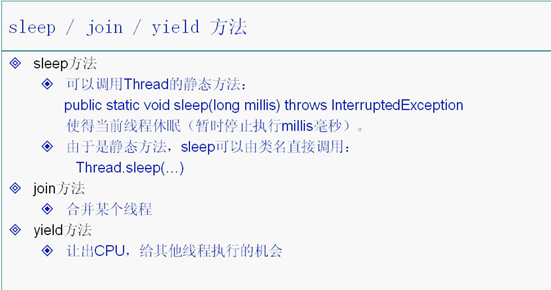
**三、线程状态转换**



**3.1.线程控制的基本方法**



### 3.2. sleep/join/yield方法介绍



**sleep方法的应用范例：**

1 package cn.galc.test;

2

3 import java.util.\*;

4

5 public class TestThread3 {

6 public static void main(String args[]){

7 MyThread thread = new MyThread();

8 thread.start();//调用start()方法启动新开辟的线程

9 try {

10 /\*Thread.sleep(10000);

11 sleep()方法是在Thread类里面声明的一个静态方法，因此可以使用Thread.sleep()的格式进行调用

12 \*/

13 /\*MyThread.sleep(10000);

14 MyThread类继承了Thread类，自然也继承了sleep()方法，所以也可以使用MyThread.sleep()的格式进行调用

15 \*/

16 /\*静态方法的调用可以直接使用“类名.静态方法名”

17 或者“对象的引用.静态方法名”的方式来调用\*/

18 MyThread.sleep(10000);

19 System.out.println("主线程睡眠了10秒种后再次启动了");

20 //在main()方法里面调用另外一个类的静态方法时，需要使用“静态方法所在的类.静态方法名”这种方式来调用

21 /\*

22 所以这里是让主线程睡眠10秒种

23 在哪个线程里面调用了sleep()方法就让哪个线程睡眠，所以现在是主线程睡眠了。

24 \*/

25 } catch (InterruptedException e) {

26 e.printStackTrace();

27 }

28 //thread.interrupt();//使用interrupt()方法去结束掉一个线程的执行并不是一个很好的做法

29 thread.flag=false;//改变循环条件，结束死循环

30 /\*\*

31 \* 当发生InterruptedException时，直接把循环的条件设置为false即可退出死循环，

32 \* 继而结束掉子线程的执行，这是一种比较好的结束子线程的做法

33 \*/

34 /\*\*

35 \* 调用interrupt()方法把正在运行的线程打断

36 相当于是主线程一盆凉水泼上去把正在执行分线程打断了

37 分线程被打断之后就会抛InterruptedException异常，这样就会执行return语句返回，结束掉线程的执行

38 所以这里的分线程在执行完10秒钟之后就结束掉了线程的执行

39 \*/

40 }

41 }

42

43 class MyThread extends Thread {

44 boolean flag = true;// 定义一个标记，用来控制循环的条件

45

46 public void run() {

47 /\*

48 \* 注意：这里不能在run()方法的后面直接写throw Exception来抛异常，

49 \* 因为现在是要重写从Thread类继承而来的run()方法,重写方法不能抛出比被重写的方法的不同的异常。

50 \* 所以这里只能写try……catch()来捕获异常

51 \*/

52 while (flag) {

53 System.out.println("==========" + new Date().toLocaleString() + "===========");

54 try {

55 /\*

56 \* 静态方法的调用格式一般为“类名.方法名”的格式去调用 在本类中声明的静态方法时调用时直接写静态方法名即可。 当然使用“类名.方法名”的格式去调用也是没有错的

57 \*/

58 // MyThread.sleep(1000);//使用“类名.方法名”的格式去调用属于本类的静态方法

59 sleep(1000);//睡眠的时如果被打断就会抛出InterruptedException异常

60 // 这里是让这个新开辟的线程每隔一秒睡眠一次，然后睡眠一秒钟后再次启动该线程

61 // 这里在一个死循环里面每隔一秒启动一次线程，每个一秒打印出当前的系统时间

62 } catch (InterruptedException e) {

63 /\*

64 \* 睡眠的时一盘冷水泼过来就有可能会打断睡眠

65 \* 因此让正在运行线程被一些意外的原因中断的时候有可能会抛被打扰中断(InterruptedException)的异常

66 \*/

67 return;

68 // 线程被中断后就返回，相当于是结束线程

69 }

70 }

71 }

72 }

**join方法的使用范例：**

1 package cn.galc.test;

2

3 public class TestThread4 {

4 public static void main(String args[]) {

5 MyThread2 thread2 = new MyThread2("mythread");

6 // 在创建一个新的线程对象的同时给这个线程对象命名为mythread

7 thread2.start();// 启动线程

8 try {

9 thread2.join();// 调用join()方法合并线程，将子线程mythread合并到主线程里面

10 // 合并线程后，程序的执行的过程就相当于是方法的调用的执行过程

11 } catch (InterruptedException e) {

12 e.printStackTrace();

13 }

14 for (int i = 0; i <= 5; i++) {

15 System.out.println("I am main Thread");

16 }

17 }

18 }

19

20 class MyThread2 extends Thread {

21 MyThread2(String s) {

22 super(s);

23 /\*

24 \* 使用super关键字调用父类的构造方法

25 \* 父类Thread的其中一个构造方法：“public Thread(String name)”

26 \* 通过这样的构造方法可以给新开辟的线程命名，便于管理线程

27 \*/

28 }

29

30 public void run() {

31 for (int i = 1; i <= 5; i++) {

32 System.out.println("I am a\t" + getName());

33 // 使用父类Thread里面定义的

34 //public final String getName()，Returns this thread's name.

35 try {

36 sleep(1000);// 让子线程每执行一次就睡眠1秒钟

37 } catch (InterruptedException e) {

38 return;

39 }

40 }

41 }

42 }

**yield方法的使用范例：**

1 package cn.galc.test;

2

3 public class TestThread5 {

4 public static void main(String args[]) {

5 MyThread3 t1 = new MyThread3("t1");

6 /\* 同时开辟了两条子线程t1和t2，t1和t2执行的都是run()方法 \*/

7 /\* 这个程序的执行过程中总共有3个线程在并行执行，分别为子线程t1和t2以及主线程 \*/

8 MyThread3 t2 = new MyThread3("t2");

9 t1.start();// 启动子线程t1

10 t2.start();// 启动子线程t2

11 for (int i = 0; i <= 5; i++) {

12 System.out.println("I am main Thread");

13 }

14 }

15 }

16

17 class MyThread3 extends Thread {

18 MyThread3(String s) {

19 super(s);

20 }

21

22 public void run() {

23 for (int i = 1; i <= 5; i++) {

24 System.out.println(getName() + "：" + i);

25 if (i % 2 == 0) {

26 yield();// 当执行到i能被2整除时当前执行的线程就让出来让另一个在执行run()方法的线程来优先执行

27 /\*

28 \* 在程序的运行的过程中可以看到，

29 \* 线程t1执行到(i%2==0)次时就会让出线程让t2线程来优先执行

30 \* 而线程t2执行到(i%2==0)次时也会让出线程给t1线程优先执行

31 \*/

32 }

33 }

34 }

35 }

## 四、线程的优先级别



**线程优先级别的使用范例：**

1 package cn.galc.test;

2

3 public class TestThread6 {

4 public static void main(String args[]) {

5 MyThread4 t4 = new MyThread4();

6 MyThread5 t5 = new MyThread5();

7 Thread t1 = new Thread(t4);

8 Thread t2 = new Thread(t5);

9 t1.setPriority(Thread.NORM\_PRIORITY + 3);// 使用setPriority()方法设置线程的优先级别，这里把t1线程的优先级别进行设置

10 /\*

11 \* 把线程t1的优先级(priority)在正常优先级(NORM\_PRIORITY)的基础上再提高3级

12 \* 这样t1的执行一次的时间就会比t2的多很多

13 \* 默认情况下NORM\_PRIORITY的值为5

14 \*/

15 t1.start();

16 t2.start();

17 System.out.println("t1线程的优先级是：" + t1.getPriority());

18 // 使用getPriority()方法取得线程的优先级别，打印出t1的优先级别为8

19 }

20 }

21

22 class MyThread4 implements Runnable {

23 public void run() {

24 for (int i = 0; i <= 1000; i++) {

25 System.out.println("T1：" + i);

26 }

27 }

28 }

29

30 class MyThread5 implements Runnable {

31 public void run() {

32 for (int i = 0; i <= 1000; i++) {

33 System.out.println("===============T2：" + i);

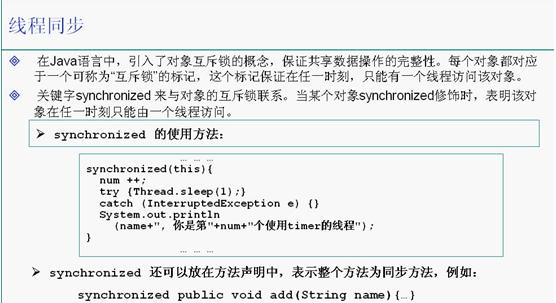
34 }

35 }

36 }

　run()方法一结束，线程也就结束了。

## 五、线程同步



**synchronized关键字的使用范例：**

1 package cn.galc.test;

2

3 public class TestSync implements Runnable {

4 Timer timer = new Timer();

5

6 public static void main(String args[]) {

7 TestSync test = new TestSync();

8 Thread t1 = new Thread(test);

9 Thread t2 = new Thread(test);

10 t1.setName("t1");// 设置t1线程的名字

11 t2.setName("t2");// 设置t2线程的名字

12 t1.start();

13 t2.start();

14 }

15

16 public void run() {

17 timer.add(Thread.currentThread().getName());

18 }

19 }

20

21 class Timer {

22 private static int num = 0;

23

24 public/\* synchronized \*/void add(String name) {// 在声明方法时加入synchronized时表示在执行这个方法的过程之中当前对象被锁定

25 synchronized (this) {

26 /\*

27 \* 使用synchronized(this)来锁定当前对象，这样就不会再出现两个不同的线程同时访问同一个对象资源的问题了 只有当一个线程访问结束后才会轮到下一个线程来访问

28 \*/

29 num++;

30 try {

31 Thread.sleep(1);

32 } catch (InterruptedException e) {

33 e.printStackTrace();

34 }

35 System.out.println(name + "：你是第" + num + "个使用timer的线程");

36 }

37 }

38 }

**线程死锁的问题：**

1 package cn.galc.test;

2

3 /\*这个小程序模拟的是线程死锁的问题\*/

4 public class TestDeadLock implements Runnable {

5 public int flag = 1;

6 static Object o1 = new Object(), o2 = new Object();

7

8 public void run() {

9 System.out.println(Thread.currentThread().getName() + "的flag=" + flag);

10 /\*

11 \* 运行程序后发现程序执行到这里打印出flag以后就再也不往下执行后面的if语句了

12 \* 程序也就死在了这里，既不往下执行也不退出

13 \*/

14

15 /\* 这是flag=1这个线程 \*/

16 if (flag == 1) {

17 synchronized (o1) {

18 /\* 使用synchronized关键字把对象01锁定了 \*/

19 try {

20 Thread.sleep(500);

21 } catch (InterruptedException e) {

22 e.printStackTrace();

23 }

24 synchronized (o2) {

25 /\*

26 \* 前面已经锁住了对象o1，只要再能锁住o2，那么就能执行打印出1的操作了

27 \* 可是这里无法锁定对象o2，因为在另外一个flag=0这个线程里面已经把对象o1给锁住了

28 \* 尽管锁住o2这个对象的线程会每隔500毫秒睡眠一次，可是在睡眠的时候仍然是锁住o2不放的

29 \*/

30 System.out.println("1");

31 }

32 }

33 }

34 /\*

35 \* 这里的两个if语句都将无法执行，因为已经造成了线程死锁的问题

36 \* flag=1这个线程在等待flag=0这个线程把对象o2的锁解开，

37 \* 而flag=0这个线程也在等待flag=1这个线程把对象o1的锁解开

38 \* 然而这两个线程都不愿意解开锁住的对象，所以就造成了线程死锁的问题

39 \*/

40

41 /\* 这是flag=0这个线程 \*/

42 if (flag == 0) {

43 synchronized (o2) {

44 /\* 这里先使用synchronized锁住对象o2 \*/

45 try {

46 Thread.sleep(500);

47 } catch (InterruptedException e) {

48 e.printStackTrace();

49 }

50 synchronized (o1) {

51 /\*

52 \* 前面已经锁住了对象o2，只要再能锁住o1，那么就能执行打印出0的操作了 可是这里无法锁定对象o1，因为在另外一个flag=1这个线程里面已经把对象o1给锁住了 尽管锁住o1这个对象的线程会每隔500毫秒睡眠一次，可是在睡眠的时候仍然是锁住o1不放的

53 \*/

54 System.out.println("0");

55 }

56 }

57 }

58 }

59

60 public static void main(String args[]) {

61 TestDeadLock td1 = new TestDeadLock();

62 TestDeadLock td2 = new TestDeadLock();

63 td1.flag = 1;

64 td2.flag = 0;

65 Thread t1 = new Thread(td1);

66 Thread t2 = new Thread(td2);

67 t1.setName("线程td1");

68 t2.setName("线程td2");

69 t1.start();

70 t2.start();

71 }

72 }

解决线程死锁的问题最好只锁定一个对象，不要同时锁定两个对象

**生产者消费者问题：**

1 package cn.galc.test;

2

3 /\* 范例名称：生产者--消费者问题

4 \* 源文件名称：ProducerConsumer.java

5 \* 要 点：

6 \* 1. 共享数据的不一致性/临界资源的保护

7 \* 2. Java对象锁的概念

8 \* 3. synchronized关键字/wait()及notify()方法

9 \*/

10

11 public class ProducerConsumer {

12 public static void main(String args[]){

13 SyncStack stack = new SyncStack();

14 Runnable p=new Producer(stack);

15 Runnable c = new Consumer(stack);

16 Thread p1 = new Thread(p);

17 Thread c1 = new Thread(c);

18

19 p1.start();

20 c1.start();

21 }

22 }

23

24

25 class SyncStack{ //支持多线程同步操作的堆栈的实现

26 private int index = 0;

27 private char []data = new char[6];

28 public synchronized void push(char c){

29 if(index == data.length){

30 try{

31 this.wait();

32 }catch(InterruptedException e){}

33 }

34 this.notify();

35 data[index] = c;

36 index++;

37 }

38 public synchronized char pop(){

39 if(index ==0){

40 try{

41 this.wait();

42 }catch(InterruptedException e){}

43 }

44 this.notify();

45 index--;

46 return data[index];

47 }

48 }

49

50

51 class Producer implements Runnable{

52 SyncStack stack;

53 public Producer(SyncStack s){

54 stack = s;

55 }

56 public void run(){

57 for(int i=0; i<20; i++){

58 char c =(char)(Math.random()\*26+'A');

59 stack.push(c);

60 System.out.println("produced："+c);

61 try{

62 Thread.sleep((int)(Math.random()\*1000));

63 }catch(InterruptedException e){

64 }

65 }

66 }

67 }

68

69

70 class Consumer implements Runnable{

71 SyncStack stack;

72 public Consumer(SyncStack s){

73 stack = s;

74 }

75 public void run(){

76 for(int i=0;i<20;i++){

77 char c = stack.pop();

78 System.out.println("消费："+c);

79 try{

80 Thread.sleep((int)(Math.random()\*1000));

81 }catch(InterruptedException e){

82 }

83 }

84 }

85 }