[线程 1](#_Toc528941446)

[线程的概念 1](#_Toc528941447)

[线程的创建方式 2](#_Toc528941448)

[线程的状态和生命周期 6](#_Toc528941449)

[线程的通信和同步 8](#_Toc528941450)

[线程同步 8](#_Toc528941451)

[Synchronized关键字 8](#_Toc528941452)

[线程访问变量的过程 8](#_Toc528941453)

[线程安全问题 9](#_Toc528941454)

[线程的API方法 9](#_Toc528941455)

[构造方法 9](#_Toc528941456)

[其它的常用API方法 9](#_Toc528941457)

[课堂练习 9](#_Toc528941458)

[预习 12](#_Toc528941459)

[家庭作业 12](#_Toc528941460)

[参考 12](#_Toc528941461)

# 线程

## 线程的概念

想要说线程首先得所下进程.什么是进程呢?一个活动的应用程序就是一个进程.进程本身是一个动态的概念.

把进程微型化得到的就是线程,一个进程中可以有N个线程.这N个线程共享进程的所有资源.

一个标准的线程由线程ID，当前指令[指针](https://baike.baidu.com/item/%E6%8C%87%E9%92%88/2878304)(PC），[寄存器](https://baike.baidu.com/item/%E5%AF%84%E5%AD%98%E5%99%A8/187682)集合和[堆栈](https://baike.baidu.com/item/%E5%A0%86%E6%A0%88/1682032)组成。

线程是程序中一个单一的顺序控制流程。进程内有一个相对独立的、可调度的执行单元，是系统独立调度和分派CPU的基本单位指令[运行](https://baike.baidu.com/item/%E8%BF%90%E8%A1%8C)时的程序的调度单位。在单个程序中同时运行多个线程完成不同的工作，称为[多线程](https://baike.baidu.com/item/%E5%A4%9A%E7%BA%BF%E7%A8%8B/1190404)。

## 线程的创建方式

Java的代码都是自行在JVM中的.如果把JVM看做是一个进程.是Window的一个进程.都在JVM中占据一个独立的线程.所有Java是多线程的.

在Java的类库中提供了:Process来代表进程.因为是抽象类我们不可以直接new对象出来.

在Java中我们很少去使用进程.一般我们使用的多数都是Thread.Thread就是Java的线程.

在Java中创建线程有四种方式:

1. 继承Thread(本质上说这是唯一的一种创建线程方式)
2. 实现Runnable接口
3. 实现Callable接口
4. 通过线程池来创建和管理线程(实际管理的是2和3的实例)这种方式是使用率最高的.
5. /\*\*
6. \*
7. \*/
8. package com.hwua.demo01;
10. /\*\*
11. \* @author Administrator
12. \*
13. \*/
14. **public** **class** Demo01 {
15. /\*\*
16. \* ' 通过继承Thread来定义一个线程类
17. \*
18. \* @author Administrator
19. \*
20. \*/
21. **class** MyThread1 extends Thread {
23. **public** MyThread1() {
24. super();
25. // TODO Auto-generated constructor stub
26. }
28. **public** MyThread1(Runnable target) {
29. super(target);
30. // TODO Auto-generated constructor stub
31. }
33. @Override
34. **public** **void** run() {
35. System.out.println("这是Thread的Run方法.");
36. }
37. }
39. /\*\*
40. \* ' 通过实现Runnable接口来定义一个线程.该线程的执行从run开始
41. \*/
42. **class** MyThread2 implements Runnable {
44. @Override
45. **public** **void** run() {
46. System.out.println("这是Runnable的Run方法.");
47. }
49. }
51. **public** **static** **void** main(String[] args) {
52. Demo01 d = **new** Demo01();
54. // -- 根据Thread的源代码可以发现.当构造Thread时,如果没有提供了Runnable对象
55. // -- 那执行的是Thread子类必须重写的run方法内容.
56. Thread t1 = d.**new** MyThread1();
57. // -- 根据Thread的源代码可以发现.当构造Thread时,如果提供了Runnable对象
58. // -- 那执行的就是Runnable对象的run方法
59. Thread t2 = d.**new** MyThread1(d.**new** MyThread2());
61. Thread t3 = **new** Thread(d.**new** MyThread2());
62. // -- 启动一个线程是通过start方法来启动
63. t1.start();// Thread
64. t2.start();// Thread
65. t3.start();// Runnable
66. }
67. }

这两种方式是必须掌握,要熟练使用的.

1. /\*\*
2. \*
3. \*/
4. package com.hwua.demo01;
6. import java.util.concurrent.Callable;
7. import java.util.concurrent.ExecutionException;
8. import java.util.concurrent.FutureTask;
10. /\*\*
11. \* @author Administrator
12. \*
13. \*/
14. **public** **class** Demo02 {
16. **class** MyThread implements Callable<String> {
17. /\*\*
18. \* ' 返回值的类型有泛型决定
19. \*/
20. @Override
21. **public** String call() throws Exception {
22. // TODO Auto-generated method stub
23. **return** "ABC";
24. }
25. }
27. **public** **static** **void** main(String[] args) {
28. Demo02 d = **new** Demo02();
29. // Callable 接口的实现类对象
30. MyThread mt = d.**new** MyThread();
32. // -- FutureTask 看做是Runnable的对象.
33. FutureTask<String> ft = **new** FutureTask<>(mt);
35. /\*
36. \* Thread的参数可以接收Runnable.没有接收Callable方法.
37. \*
38. \*/
39. Thread t = **new** Thread(ft);
40. t.start();
42. String result = "";
43. **try** {
44. result = ft.get();
45. System.out.println(result);
46. } **catch** (InterruptedException | ExecutionException e) {
47. // TODO Auto-generated catch block
48. e.printStackTrace();
49. }
51. }
53. }

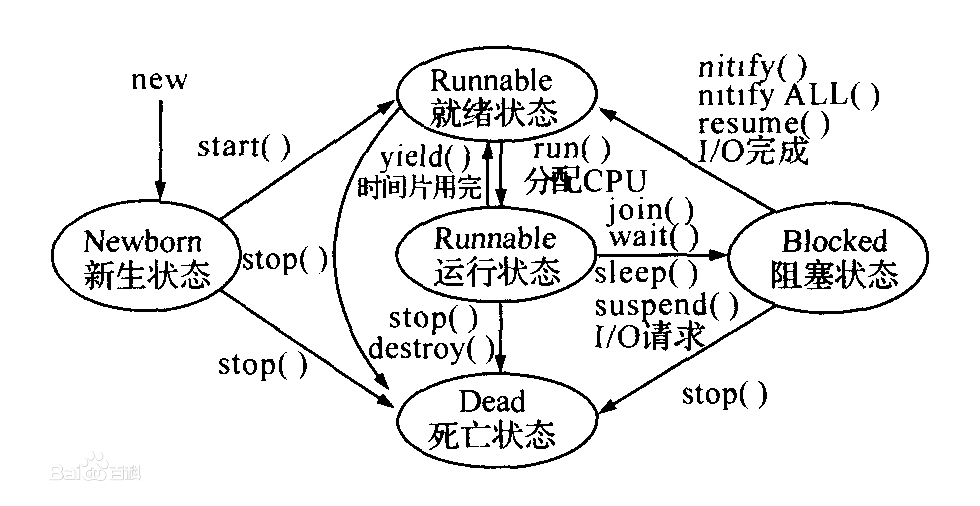
这个了解就行.

## 线程的状态和生命周期

只有Thread和Thread的子类才可以叫做线程.无论是Runnable还是Callable都只能作为线程的执行单元.

我们从执行的方式上就可以发现.Thread的对象可以直接调用start方法.而Callable的对象和Runnable的对象.必须以作为Thread类的构造方法参数,传递到Thread中.

线程从创建到运行到销毁一共要经历几个过程.我们把这几个过程叫做线程的生命周期.每个周会有一个与之对应的状态.在Thread中设计了枚举来保存这些状态.



当一个线程对象new出来.该线程处于新建状态.

当一个新建的线程对象调用start方法.该线程进入就绪状态.就绪状态的线程已经被分配资源(堆,栈)唯独没有的就是CPU的使用权.

当一个处于就绪状态的线程.被OS调度即被CPU分配了时间片段获得了CPU使用权时,就进入运行状态

当一个处于运行状态的线程.它的run方法执行结束.该线程进入死亡状态.一个死亡的线程是不可以被重新启动的.线程池中的线程例外.因为它没有真正的死亡.除非手工释放.

当一个处于运行状态的线程.因调用sleep,wait,yield等方法.或者等待数据库资源,或者等待IO响应时,会进行阻塞状态.直到所等待的资源被释放.

阻塞状态的线程是不可以直接回到运行状态的.要重新回到就绪状态.等待CPU调度.(sleep特殊)

## 线程的通信和同步

### 线程同步

#### Synchronized关键字

我们发现在执行代码时,本应该作为一个整体存在,因为CPU时间片段被拆分成多次执行.在这拆分的过程中,其它线程也有可能会执行.为了解决这个问题.在线程中引入原子性概念

原子性指:一个操作时不可中断的.即使在多线程环境中.一旦某一线程开始执行某个操作.不会受到其它线程的影响.

Synchronized在Java中有两种使用方式

1. Synchronized块
2. 使用它作为关键字来修饰方法.

## 线程访问变量的过程

## 线程安全问题

# 线程的API方法

## 构造方法

线程常用的构造方法四个:

1. 无参
2. String作为参数 .作为线程的名称存在.
3. Runnable作为参数
4. String和Runnable都为参数

## 其它的常用API方法

# 课堂练习

1. 设计账户类.有账户名称,余额两个字段.设计一个线程用于从该账户中取款.创建两个线程对象,代表两个地方同时在取款.假设每次取款50.账户余额500.求取款过程.
2. 有100个桃子.有三只猴子每只猴子取走剩余桃子的一半.求取的过程.
3. 编一个程序，实现两个线程 A 和 B ，A 和 B 同时启动，A线程每隔5秒显示字符串”AAA”，B线程每隔 1秒显示字符‘B’。提示需要用到两个方法wait和notify
4. /\*\*
5. \*
6. \*/
7. package com.hwua.demo04;
9. /\*\*
10. \* @author Administrator
11. \*
12. \*/
13. **public** **class** Demo01 {
15. **public** **static** **void** main(String[] args) {
16. final Object obj = **new** Object();
18. Thread w1 = **new** WorK1(obj);
19. w1.start();
21. Thread w2 = **new** WorK2(obj);
22. w2.start();
23. }
25. }
27. **class** WorK1 extends Thread {
28. **private** Object obj;
30. /\*\*
31. \* @param obj
32. \*/
33. **public** WorK1(Object obj) {
34. **this**.obj = obj;
35. }
37. @Override
38. **public** **void** run() {
39. **while** (**true**) {
40. synchronized (obj) {
41. // -- 唤醒别人
42. obj.notify();
44. System.out.println("A");
46. **try** {
47. sleep(333);
48. obj.wait();
49. } **catch** (InterruptedException e) {
50. // TODO Auto-generated catch block
51. e.printStackTrace();
52. }
54. }
55. }
56. }
57. }
59. **class** WorK2 extends Thread {
60. **private** Object obj;
62. /\*\*
63. \* @param obj
64. \*/
65. **public** WorK2(Object obj) {
66. **this**.obj = obj;
67. }
69. @Override
70. **public** **void** run() {
71. **while** (**true**) {
72. synchronized (obj) {
73. obj.notify();
74. System.out.println("B");
75. **try** {
76. sleep(333);
77. obj.wait();
78. } **catch** (InterruptedException e) {
79. // TODO Auto-generated catch block
80. e.printStackTrace();
81. }
82. }
83. }
84. }
85. }

# 预习

1. Lock
2. BlockQueue
3. 线程池
4. 生产者和消费者(线程通信的基本模型.必回的)

# 家庭作业

1. 桌上用一个盘子,可以放一个水果,父亲放苹果到盘子上,母亲放香蕉到盘子上;儿子专吃香蕉;而女儿专吃苹果
2. 锁在使用的过程中会产生死锁.尝试模拟一把
3. 客户和联系人的增删改查

# 参考

1. [JVM的内存](http://www.cnblogs.com/prayers/p/5515245.html)
2. [Java锁介绍](https://blog.csdn.net/zqz_zqz/article/details/70233767)