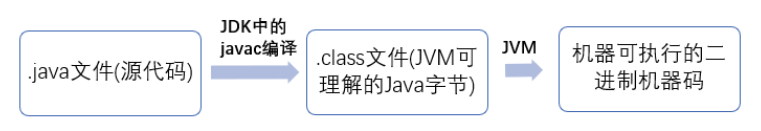
1. Java基础功
   1. Java入门
      1. Java语言有哪些特点？
2. 简单易学
3. 面向对象（封装、继承、多态、抽象）
4. 平台无关性（Java虚拟机实现平台无关性）
5. 可靠性
6. 安全性
7. 支持多线程（C++语言没有内置的多线程机制）
8. 支持网络编程而且很方便
9. 编译与解释并存
   * 1. 关于JVM、JDK和JRE最详细通俗的解答

JVM是运行Java字节码的虚拟机。JVM有针对不同系统的特点实现，目的是使用相同的字节码，它们都会给出相同的结果。（**相同字节码->JVM不同操作系统->相同结果**）

什么是字节码？采用字节码的好处是什么？

JVM可以理解的代码就叫做字节码，只面向虚拟机。由于字节码不针对一种特定的机器。因此，Java程序无需重新编译便可以在多种不同的操作系统的计算机上运行。



JVM是运行Java字节码的虚拟机。JVM有针对不同系统的特定实现，**目的是使用相同的字节码，它们都会给出相同的结果。**字节码和不同系统的JVM实现是Java语言的“一次编译，随处可以运行” 的关键所在。

* + 1. Java和C++的区别

1. 都是面向对象的语言，都支持封装、继承和多态。
2. Java不提供指针来直接访问内存，程序内存更加安全。
3. Java的类是单继承的，C++支持多继承；虽然Java的类不可以继承，但是接口可以多继承。
4. Java有自动内存管理垃圾回收机制（GC），不需要程序员手动释放无用内存。
5. **在 C 语言中，字符串或字符数组最后都会有一个额外的字符'\0'来表示结束。但是，Java 语言中没有结束符这一概念。**
   * 1. **为什么说Java语言“编译与解释并存”？**

**编译型语言是指编译器针对特定的操作系统将源代码一次性翻译成可被该平台执行的机器码。**

**解释型语言是指解释器对源程序逐行翻译成特定平台的机器码并立即执行。**

**Java语言既具有编译型语言的特征，也具有解释型语言的特性，因为Java程序需要经过先编译后解释两个步骤。**

**由Java编写的程序需要先经过编译步骤，生成.class字节码，这种字节码必须经过Java解释器JVM来解释执行。因此，我们可以认为Java语言编译与解释并存。**

* 1. **Java语法**
     1. **字符型常量与字符串常量的区别？（char和string）**

1. **形式上：字符常量是单引号引起的一个字符；字符串常量是双引号引起的0个或若干个字符。**
2. **含义上：字符常量相当于一个整型值（ASCII），可以参加表达式运算；字符串常量代表一个地址值。**
3. **占内存大小，字符常量只占2个字节（16bits）；字符串常量站若干个字节。**

**1字节=8bits**

**Int 4字节 float 4字节（32位）**

**Long 8字节 double 8字节**

**Java每种基本类型所占存储空间的大小，不随机器硬件架构的变化而变化。这种存储空间大小的不变性是Java语言比其他大多数语言编写的程序更具移植性的原因之一。**

* + 1. **Java泛型**

**Java的泛型是伪泛型，这是因为Java在编译期间，所有的泛型信息都会被擦掉，这也就是通常所说类型擦除 。**

**泛型类、泛型接口、泛型方法**

* + 1. **==和equals的区别**

**==：它的作用是判断两个对象的地址是否相等。即判断两个对象是否是同一个对象。基本数据类型==是比较的数值，引用数据类型==比较的是内存地址。**

**Equals(): 他的作用也是判断两个对象是否相等，它不能用于比较基本数据类型。存在于Objects类中。**

**情况1：没有被覆盖，等价与==，比较的是两个对象的内存地址，是否为同一个对象。**

**情况2：被覆盖。比较两个对象的内容是否相等，即比较的是两个对象的值。**String 中的 equals 方法是被重写过的，因为 Object 的 equals 方法是比较的对象的内存地址，**而 String 的 equals 方法比较的是对象的值**。

* + 1. hashcode和equals

1. hashcode介绍

hashcode的作用是获取哈希码，也称为散列码，返回int整数，这个哈希码的作用是确定该对象在哈希表中的索引位置。用C++实现，将对象的内存地址转化为整数地址。

1. 为什么要用hashcode?

当你把对象加入hashset时，会先计算对象的hashcode来判断对象加入的位置，同时也会与其他已经加入的对象的hashcode值进行比较，如果没有相符的hashcode,hashcode会假设对象没有重复出现。

但是如果相同的hashcode,这时会调用equals方法来检查hashcode相等的话，两个对象是否相等。如果两者相等，hashset就不会让其加入操作成功。如果不同的话，则加入。

这样我们就大大减少了equals的次数，大大提高了执行速度。

1. 为什么重写equals时必须重写hashcodes方法？

hashCode()的默认行为是对堆上的对象产生独特值。如果没有重写 hashCode()，则该 class 的两个对象无论如何都不会相等（即使这两个对象指向相同的数据）

1. **为什么两个对象有相同的hashcoded值，它们也不一定相等？**

**因为hashcode所使用的杂凑算法也许刚好会让多个对象传回相同的杂凑值。如果hashset在对比时，同样的hashcode 有多个对象，它会使用equals来判断是否真的相同。也就是hashcode只是用来缩小查找成本。**

* 1. **基本数据类型**
     1. **Java中的几种基本数据类型**

**Java中有8种基本数据类型：**

1. **6种数字类型：byte 1\*8、short 2\*8、int 4\*8、long 8\*8、float 4\*8、double 8\*8**
2. **1种字符类型：char 2\*8**
3. **1种布尔型：boolean 1bit**

**对应的包装类（引用类型）分别为：**

**Byte、Short、Integer、Long、Float、Double**

**Character、**

**Boolean**

* + 1. **自动拆箱与装箱**

**装箱：将基本数据类型用它们对应的引用类型包装起来**

**拆箱：将包装类型转化为基本数据类型**

* + 1. **包装类和常量池**

**Java 基本类型的包装类的大部分都实现了常量池技术，即 Byte,Short,Integer,Long,Character,Boolean；前面 4 种包装类默认创建了数值[-128，127] 的相应类型的缓存数据，Character创建了数值在[0,127]范围的缓存数据，Boolean 直接返回True Or False。如果超出对应范围仍然会去创建新的对象。**

* 1. **方法**
     1. **为什么Java中只有值传递**

1. **按值调用表示方法接受的是调用者提供的值，而按引用调用表示方法接受的时调用者提供的变量的地址。**
2. **一个方法可以修改传递引用所对应的变量值，而不能修改传递值调用所对应的变量值。**
3. **Java是采用值传递，即方法得到的是所有参数值的一个拷贝，也就是不能修改传递值调用所对应的变量值。**

**一个方法不能修改基本数据类型的参数，而对象引用作为参数时，或者数组也是对象，方法得到的是对象引用的拷贝，对象引用以及其他的拷贝同时引用的是同一个对象，即可修改。**

public class Test {

public static void main(String[] args) {

// TODO Auto-generated method stub

Student s1 = new Student("小张");

Student s2 = new Student("小李");

Test.swap(s1, s2);

System.out.println("s1:" + s1.getName());

System.out.println("s2:" + s2.getName());

}

public static void swap(Student x, Student y) {

Student temp = x;

x = y;

y = temp;

System.out.println("x:" + x.getName());

System.out.println("y:" + y.getName());

}

}

x:小李

y:小张

s1:小张

s2:小李

**方法并没有改变存储在变量 s1 和 s2 中的对象引用。swap 方法的参数 x 和 y 被初始化为两个对象引用的拷贝，这个方法交换的是这两个拷贝**

**总结：**

**Java对象采用的不是引用调用，实际上，对象引用是按照值传递，得到的是所有参数值的拷贝，也就是不能修改传递值调用所对应的变量值。**

1. **一个方法不能修改一个基本数据类型的参数。**
2. **一个方法可以改变一个对象参数的状态。**
3. **一个方法不能让对象参数引用一个新的对象。**
   * 1. 重载和重写的区别

重载：就是同一个方法能够根据输入数据的不同，做出不同的处理。

重写：就是当子类继承自父类的相同方法，输入数据一样，但是要做出有别父类的响应时，就得覆盖数据方法。

* + - 1. 重载

如果多个方法具有相同的名字，不同的参数，便产生了重载。参数类型不同、个数不同、顺序不同。

返回值不是方法签名的一部分。也就是说，不能有两个名字相同，参数类型也相同却不同类型值的方法。

重载就是同一个类中多个同名方法根据不同的传参来执行不同的逻辑处理。

* + - 1. 重写

父类->子类

重写发生在运行期，是子类对父类运行访问的方法的实现过程进行重新编写。

遵循“两同两小一大“：两同即方法名相同、形参列表相同

两小：子类方法**返回值类型**应该父类返回值类型更小或相等，子类方法声明**抛出异常类**应比父类方法声明抛出的异常类更小或相等。

一大指的是子类方法的访问权限应比父类方法的访问权限更大。

* + 1. 深拷贝vs浅拷贝

1. 浅拷贝：对基本数据类型进行值传递，对引用数据类型进行引**用传递般的拷贝**，此为浅拷贝。
2. 深拷贝：对基本数据类型进行值传递，对引用数据类型，创建一个新的对象，并复制其内容，此为深拷贝。
   * 1. 方法的四种类型

无参数无返回值方法

有参数无返回值方法

有返回值无参数方法

有返回值有参数方法

1. Java面向对象
   1. 类和对象
      1. 面向对象和面向过程的区别

面向过程：**面向过程性能比面向对象高。** 因为类调用时需要实例化，开销比较大，比较消耗资源，所以当性能是最重要的考量因素的时候，比如单片机、嵌入式开发、Linux/Unix 等一般采用**面向过程开发**。但是，**面向过程没有面向对象易维护、易复用、易扩展。**

**面向对象易维护、易复用、易扩展。** 因为面向对象**有封装、继承、多态性**的特性，所以可以设计出低耦合的系统，使系统更加灵活、更加易于维护。但是，**面向对象性能比面向过程低**。

* + 1. **构造器是否能被重写？**

构造器不能被重写(继承重写)，但是可以被重载。

* + 1. 在Java中定义个不做事且没有参数的构造方法的作用

父类如果只定义了有参数的构造方法，则默认不会构造无参数的构造方法。如果此时，子类没有采用super()来调用，则编译时发生错误，因为无法找到没有参数的构造方法可供执行。解决办法是在父类里加上一个不做事且没有参数的构造方法。

* + 1. 成员变量与局部变量的区别

1. 从语法形式上来看：成员变量是属于类的，而局部变量是在代码块或方法中定义的变量；成员变量可以被public、private、static等修饰符修饰。而局部变量不能被访问修饰符及static修饰;但是，成员变量和局部变量都能被final修饰。
2. 从变量在内存中的存储方式来看：如果成员变量是使用static修饰的，n那么这个成员变量是属于类的，如果没有使用static修饰，这个成员变量是属于实例的。**对象存于堆内存，局部变量存于栈内存。**
3. 从变量在内存中的生存时间上看：成员变量是对象的一部分，它随着对象的创建而存在。而局部变量随着方法的调用而自动消失。
4. 成员变量如果没有被赋初值，则会自动以类型的默认值而赋值。而局部变量则不会自动赋值。

对象实例存放在堆内存，而对象引用存放在栈内存。

一个对象可以指向0/1个对象。一个对象可以有N个对象指向它。

* + 1. 一个类的构造方法的作用是什么，如果没有声明构造方法，该程序能正确执行吗？

**主要作用是完成对类对象的初始化工作**。可以执行。因为一个类即使没有声明构造方法也会有默认的不带参数的构造方法。

如果我们自己添加了类的构造方法无论是否有参，Java就不会添加默认的构造方法了，同时new 对象的时候，要传递参数，所有我们一直在不知不觉地构造方法，得这需要我们在创建对象的时候后面要加一个括号，因为要调用无参的构造方法。

如果我们重载了有参的构造方法，记得要把无参的构造方法也写出来（无论是否用到），因为这可以帮助我们在创建对象的时候少踩坑。

* + 1. 构造方法有哪些特性？

1. 名字与类名相同
2. 没有返回值，但不能用void声明对象
3. **生成类的对象时自动执行，无需调用**
   * 1. 在调用子类的构造方法之前会调用父类没有参数的构造方法，目的是？

帮助子类做初始化工作

* + 1. 对象的相等与指向它们的引用相等有什么不同

对象相等，比的是内存中存放的内容是否相等（值相等）。引用相等比较的是它们指向的内存地址是否相等。

* 1. 面向对象的三大特征
     1. 封装

封装是指把一个对象的状态信息（属性）隐藏在对象内部，不允许外部对象直接访问对象的内部信息。但是可以提供一些被外界访问的方法来操作属性。

* + 1. 继承

继承是使用已存在的类的定义作为基础建立新类的技术，新类的定义可以增加新的数据或新的功能，也可以用父类的功能，但是不能选择性的继承父类。

通过使用继承，可以快速的创建新的类，可以提高代码的重用，程序的可维护性，节省大量创建新类的时间，提高我们的开发效率。

1. 子类拥有父类对象的所有方法和属性，包括私有属性和方法。但是父类中的私有属性和方法不能访问，只是拥有。
2. 子类可以拥有自己的属性和方法，即子类可以对父类进行扩展。
3. 子类可以用自己的方式实现父类的方法。
   * 1. 多态

多态，顾名思义，表示一个对象具有多种的状态。具体表现为父类的引用指向子类的实例。

1. 对象类型和引用类型之间具有继承（类）/实现（接口）的关系；
2. 对象类型不可变，引用类型可变；
3. 方法具有多态性，属性不具有多态性；
4. 多态不能调用“只在子类存在但在父类不存在”的方法；
5. 如果子类重写了父类的方法，真正执行的是子类覆盖的方法，如果子类没有覆盖父类的方法，执行的是父类的方法。
   1. 修饰符
      1. 在静态方法内调用一个非静态成员为什么是非法的？

由于静态方法可以不通过对象进行调用，因此在静态方法里，不能调用其他非静态变量，也不可以访问非静态变量成员。

* + 1. 静态方法和实例方法的不同

1. 在外部调用静态方法时，可以采用类名.方法名的方式，也可以采用对象名.方法名的方式。而实例方法只有后面这种形式。也就是说，调用静态方法时候无需创建对象。
2. 静态方法在访问成员时，只允许访问静态成员（即静态成员变量和静态方法），而不允许访问实例成员变量和实例方法。实例方法则无此限制。
   * 1. final关键字
3. final修饰的类不能被继承，final类中的所有成员方法都会被隐式指定为final方法
4. fianl修饰的方法不能被重写
5. final修饰的变量为常量。如果式基本数据类型的变量，则数据一旦初始化后便不能修改。如果是引用类型的变量，则在对其初始化后便不能让其指向另一个对象。

使用final原因。第一个是把方法锁定，以防止任何继承类修改它的涵义。第二个就是效率。

* + 1. static关键字

1. 修饰成员变量和方法

被 static 修饰的成员属于类，不属于单个这个类的某个对象，被类中所有对象共享，可以并且建议通过类名调用。

1. 静态代码块

 静态代码块定义在类中方法外, 静态代码块在非静态代码块之前执行(静态代码块—>非静态代码块—>构造方法)。 该类不管创建多少对象，静态代码块只执行一次.

1. 静态内部类

静态内部类与非静态内部类之间存在一个最大的区别: 非静态内部类在编译完成之后会隐含地保存着一个引用，该引用是指向创建它的外围类，但是静态内部类却没有。没有这个引用就意味着：1. 它的创建是不需要依赖外围类的创建。2. 它不能使用任何外围类的非static成员变量和方法。

1. 静态导包

 这两个关键字连用可以指定导入某个类中的指定静态资源，并且不需要使用类名调用类中静态成员，可以直接使用类中静态成员变量和成员方法。

* + 1. this关键字

this关键字用于引用类的当前实例。

* + 1. super关键字

用于从子类访问父类的变量和方法。

1. 调用父类中的其他构造方法时，该语句必须处于构造器的首行，否则编译器会报错。另外，this 调用本类中的其他构造方法时，也要放在首行。
2. This、super不能用在static方法中

被static修饰的成员属于类，不属于单个这个类的某个对象，被类中所有对象共享。而this代表对本类对象的引用，指向本类对象；而super指向父类对象。This和super是属于对象范畴的东西，而静态方法是属于类范畴的东西。

* 1. 接口和抽象类
     1. 接口和抽象类的区别是什么

1. 接口的方法默认为public，所有方法在接口中不能实现。

抽象类可以有非抽象类的方法。

1. 接口除了static、final变量，不能有其他变量。

抽象类则不一定。

1. 一个类可以扩展多个接口，但是只能实现一个抽象类。

接口本身可以通过extends关键字扩展多个接口。

1. 接口方法是public，抽象方法可以有public、protected和default这些修饰符。

抽象方法就是为了被重写所以不能使用private关键字修饰。

1. 从设计层面来说，抽象是对类的抽象，是一种模板设计。而接口是对行为的抽象，是一种行为规范。
   1. 其他重要知识点
      1. String 、 StringBuffer 和 StringBuilder的区别是什么？
2. String类中使用**final关键字**修饰字符数组来保存字符串，**所以String对象是不可变的。可理解为常量，线程安全的。**
3. StringBuffer 和 StringBuilde都是继承类，使用字符数组来保存字符串，但是没有使用fianl关键字，所以是**可以变**的。
4. StringBuffer 对方法加了**同步锁**或者对调用的方法加了同步锁，所以是**线程安全**的。StringBuilder 并没有对方法进行加同步锁，所以是**非线程安全**的。

每次对 String 类型进行改变的时候，**都会生成一个新的 String 对象**，然后将指针指向新的**String 对象**。StringBuffer 每次都会对 StringBuffer 对象本身进行操作，而不是生成新的对象并改变对象引用。相同情况下使用 StringBuilder 相比使用 StringBuffer 仅能获得 10%~15% 左右的性能提升，但却要冒多线程不安全的风险。

**String：不可变的。线程安全。操作少量数据**

**StringBuilder: 可变的。没有加同步锁，非线程安全的。单线程大量数据**

**StringBuffer：可变的。加了同步锁，线程安全的。多线程大量数据**

* + 1. Object类的常见方法总结

public final native Class<?> getClass()//native方法，用于返回当前运行时对象的Class对象，使用了final关键字修饰，故不允许子类重写。public native int hashCode() //native方法，用于返回对象的哈希码，主要使用在哈希表中，

public boolean equals(Object obj)//用于比较2个对象的内存地址是否相等，String类对该方法进行了重写用户比较字符串的值是否相等。

protected native Object clone() throws CloneNotSupportedException//naitive方法，用于创建并返回当前对象的一份拷贝。一般情况下，对于任何对象 x，表达式 x.clone() != x 为true，x.clone().getClass() == x.getClass() 为true。Object本身没有实现Cloneable接口，所以不重写clone方法并且进行调用的话会发生CloneNotSupportedException异常。

public String toString()//返回类的名字@实例的哈希码的16进制的字符串。建议Object所有的子类都重写这个方法。

public final native void notify()//native方法，并且不能重写。唤醒一个在此对象监视器上等待的线程(监视器相当于就是锁的概念)。如果有多个线程在等待只会任意唤醒一个。

public final native void notifyAll()//native方法，并且不能重写。跟notify一样，唯一的区别就是会唤醒在此对象监视器上等待的所有线程，而不是一个线程。

public final native void wait(long timeout) throws InterruptedException//native方法，并且不能重写。暂停线程的执行**。注意：sleep方法没有释放锁，而wait方法释放了锁 。timeout是等待时间。**

protected void finalize() throws Throwable { }//实例被垃圾回收器回收的时候触发的操作

* + 1. == 与 equals

重复

public class test1 {

public static void main(String[] args) {

String a = new String("ab"); // a 为一个引用

String b = new String("ab"); // b为另一个引用,对象的内容一样

String aa = "ab"; // 放在常量池中

String bb = "ab"; // 从常量池中查找

if (aa == bb) // true

System.out.println("aa==bb");

if (a == b) // false，非同一对象

System.out.println("a==b");

if (a.equals(b)) // true String对象 equals已经被重写，比较两者的值是否相等

System.out.println("aEQb");

if (42 == 42.0) { // true

System.out.println("true");

}

}

}

String 中的 equals 方法是被重写过的，因为 object 的 equals 方法是比较的对象的内存地址，而 String 的 equals 方法比较的是对象的值。

* + 1. Java中有字段不想进行序列号

对于不想进行序列化的变量，使用transient关键字修饰。

* + 1. 获取键盘输入常用的两种方法

1. Scanner

Scanner input = new Scanner(System.in);

String s = input.nextLine();

input.close();

1. BufferedReader

BufferedReader input = new BufferedReader(new InputStreamReader(System.in));

String s = input.readLine();

1. Java 核心技术
   1. 异常

在 Java 中，所有的异常都有一个共同的祖先 java.lang 包中的 **Throwable 类**。Throwable： 有两个重要的子类：**Exception（异常）** 和 **Error（错误）** ，二者都是 Java 异常处理的重要子类，各自都包含大量子类。

**Error（错误）:是程序无法处理的错误**，表示运行应用程序中较严重问题。大多数错误与代码编写者执行的操作无关，而表示代码运行时 JVM（Java 虚拟机）出现的问题。

**Exception（异常）:是程序本身可以处理的异常**。Exception 类有一个重要的子类 **RuntimeException**。RuntimeException 异常由 Java 虚拟机抛出。**NullPointerException**（要访问的变量没有引用任何对象时，抛出该异常）、**ArithmeticException**（算术运算异常，一个整数除以 0 时，抛出该异常）和 **ArrayIndexOutOfBoundsException** （下标越界异常）

**注意：异常和错误的区别：异常能被程序本身处理，错误是无法处理。**

* 1. **多线程**

**线程：一个进程在其执行的过程中可以产生多个线程**。与进程不同的是同类的多个线程共享同**一块内存空间和一组系统资源**，所以系统在产生一个线程，或是在各个线程之间作切换工作时，负担要比进程小得多，也正因为如此，线程也被称为轻量级进程。

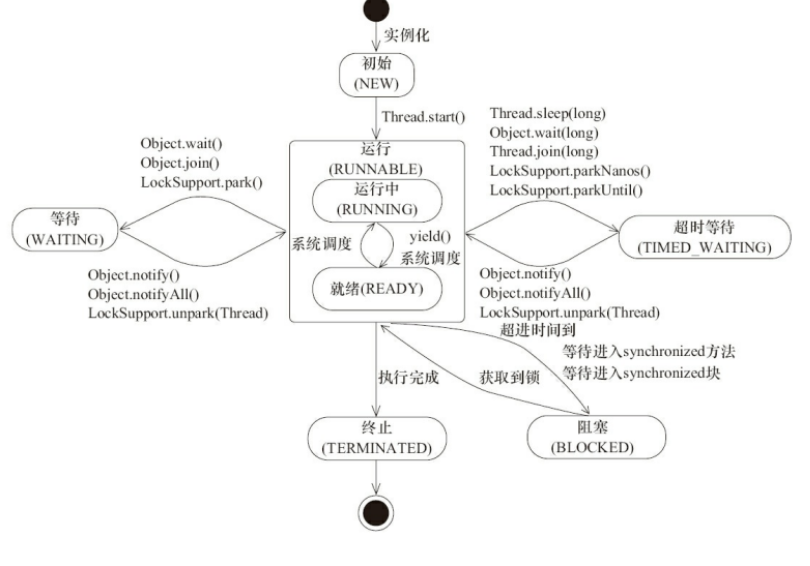
**程序：**是含有指令和数据的文件，被存储在磁盘或其他的数据存储设备中，也就是说程序是静态的代码。

**进程**是程序的一次执行过程，是系统运行程序的基本单位，因此进程是动态的。**系统运行一个程序即是一个进程从创建，运行到消亡的过程**。

 线程是进程划分成的更小的运行单位。

线程和进程最大的不同在于基本上各进程是独立的，而各线程则不一定，**因为同一进程中的线程极有可能会相互影响**。从另一角度来说，进程属于操作系统的范畴，主要是同一段时间内，可以同时执行一个以上的程序，而线程则是在同一程序内几乎同时执行一个以上的程序段。

线程具有哪些执行状态？



1. 线程创建之后它将处于 **NEW（新建）** 状态，调用 start() 方法后开始运行，线程这时候处于 **READY（可运行）** 状态。可运行状态的线程获得了 cpu 时间片（timeslice）后就处于 **RUNNING（运行）** 状态。
2. 当线程执行 wait()方法之后，线程进入 \*\*WAITING（等待）\*\*状态。进入等待状态的线程需要依靠其他线程的通知才能够返回到运行状态。
3. 而 **TIME\_WAITING(超时等待)** 状态相当于在等待状态的基础上增加了超时限制，比如通过 sleep（long millis）方法或 wait（long millis）方法可以将 Java 线程置于 TIMED WAITING 状态。当超时时间到达后 Java 线程将会返回到 RUNNABLE 状态。
4. 当线程调用同步方法时，在没有获取到锁的情况下，线程将会进入到 **BLOCKED（阻塞）** 状态。
5. 线程在执行 Runnable 的run()方法之后将会进入到 **TERMINATED（终止）** 状态。