什么是移动开发

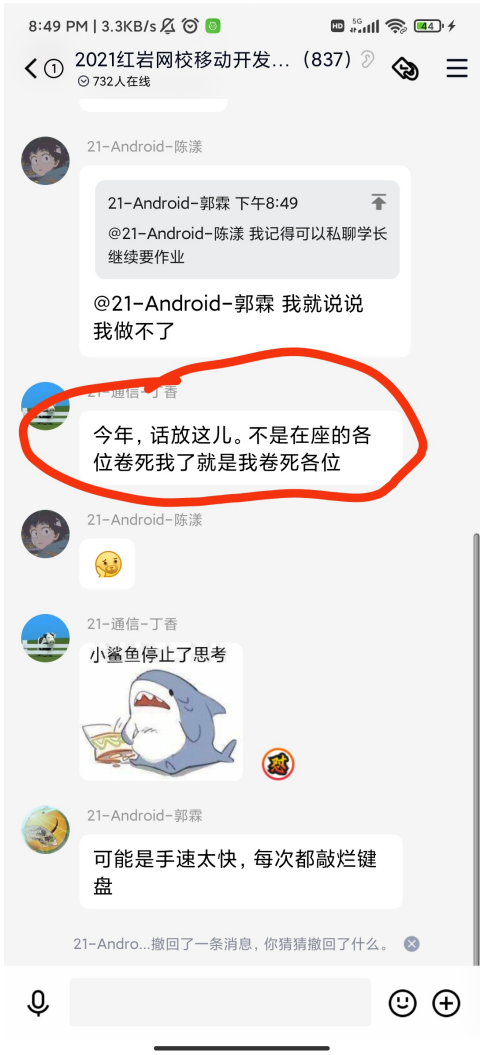
移动开发也称为**手机开发**，或叫做移动**互联网开发**。是指以手机等**便携终端**为基础，进行相应的开发工作，由于这些随身设备基本都采用无线上网的方式，因此，业内也称作为无线开发。

我个人理解的就是**便捷设备**上面app的开发。比如手机，平板，手表。

在我们移动开发部，你可以获得开发**任意一款**你想要开发的app。

学习计划介绍

大家可以看一看我写的2021级秋季课程大纲安排，就在移动开发群的群文件里面。



开卷小课堂

大家在之前可能或多或少听到过有的学长说红岩很难，课程进度很快等等一系列的问题。其实我想告诉大家是 不是红岩的课很难或者怎么样，而是自己太懒。 大家都是刚刚经历过高考，你们觉得高三苦吗？累吗？ 大家刚来到大一，一个非常自由的大学生活，

很多人其实并不想努力，而是想玩，而且很多高中老师可能也说过，等你们上了大学就可以轻松了。但是大家来了大学生活一段时间就会发现，其实-- 高中老师的嘴，骗人的鬼！（当然开个玩笑啦!）

相比于那些大学喜欢玩的人，我们在这里的每一个人都已经赢了他们第一步，那么日后我们就会赢他们很多次，等到你们之间的差距越来越大 就是当你正在考虑 我到底是去腾讯好喃？还是字节跳动好一点南？亦或是华为更好？..... 而他还在绞尽脑汁的找工作.....

比起相信运气可以改变我的人生，那么我更加的信赖自己的能力。

既然大家选择来到了红岩，那就一直努力成为一名真正的Redrocker!

我相信你在红岩的一年会学到很多东西。

零基础

关于零基础，零基础并不可怕，可怕的是你不去努力。我上大一时也是彻彻底底的零基础，红岩里面也有很多学长大一都是零基础的，所以不要害怕他们有基础的（有基础的同学就更要努力了，一不小心可能就会被他们超越），只要你肯努力，**我花开后百花杀!**

因为起点低就越贴近地面，就能更好的**蓄势待发**，我认为世间**真正的精彩**不是一开始就在前面的人**一直在前面。**而是那些起点低落后的人却能够**奋起直追**，再到**与之并行**，最后再**将之超越**才是世界上真正的精彩，不是吗?

当从零基础变成一个大佬的时候，你可能就需要装逼.....那么

如何优雅的装逼?

装逼可能是人类的第一大需求，从古至今都是如此。

我理解的装逼就是：**希望通过某种方式获得别人的认可！**装逼装的好也是人生的一种境界！

大家在红岩网校学习，可能会比其他人学到更多的东西，你也许会觉得你很厉害，有时候就希望获得他们的**认可。**装逼是可以，但是大家不能天天去装逼，就是我们不需要表现出**一直很厉害**，只需要**某个时候很厉害。**如果你**每天**都去吹嘘自己很厉害，或者说一些其他人不懂的东西来显的自己很厉害，别人就会觉得你很讨厌，看起来也很沙雕。你可以在他需要你帮助的时候，展示出自己的一些能力，不要过分的展示，这样你就可以**顺理成章的装逼**，别人也不会抵触你。

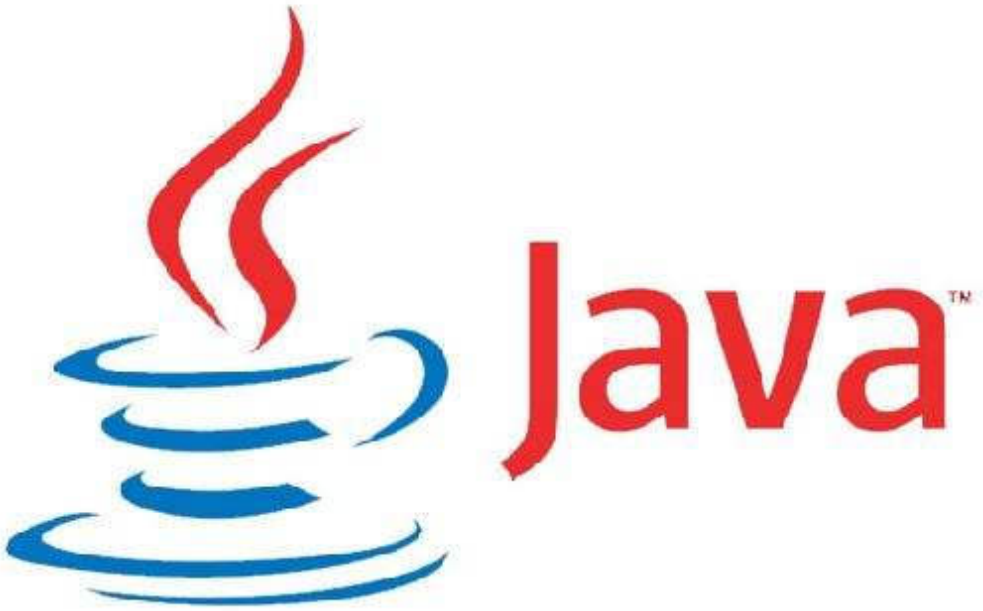
- **不要有总是希望显得自己比别人厉害这种心理。**
- **还有如果有时候你实在要装逼，也千万不要通过对比别人，或者贬低别人的方式进行装逼。**

其实我真正想说的不是让大家怎么去装逼，而是如何变得内敛，不要太**浮躁**或者**骄傲。**要像**水**一样，**既可以水波不兴亦可惊涛骇浪**

看似好像在红岩学了很多的东西，（对于同龄人来说）确实有很多的东西，其实你仔细一看好像又没有很多的东西，（对于计算机来说）确实又没有好多东西。

- **人外有人，天外有天。**
- **低调的做人，高调的做事。**
- **不鸣则已，一鸣惊人。**

Java的前世今生

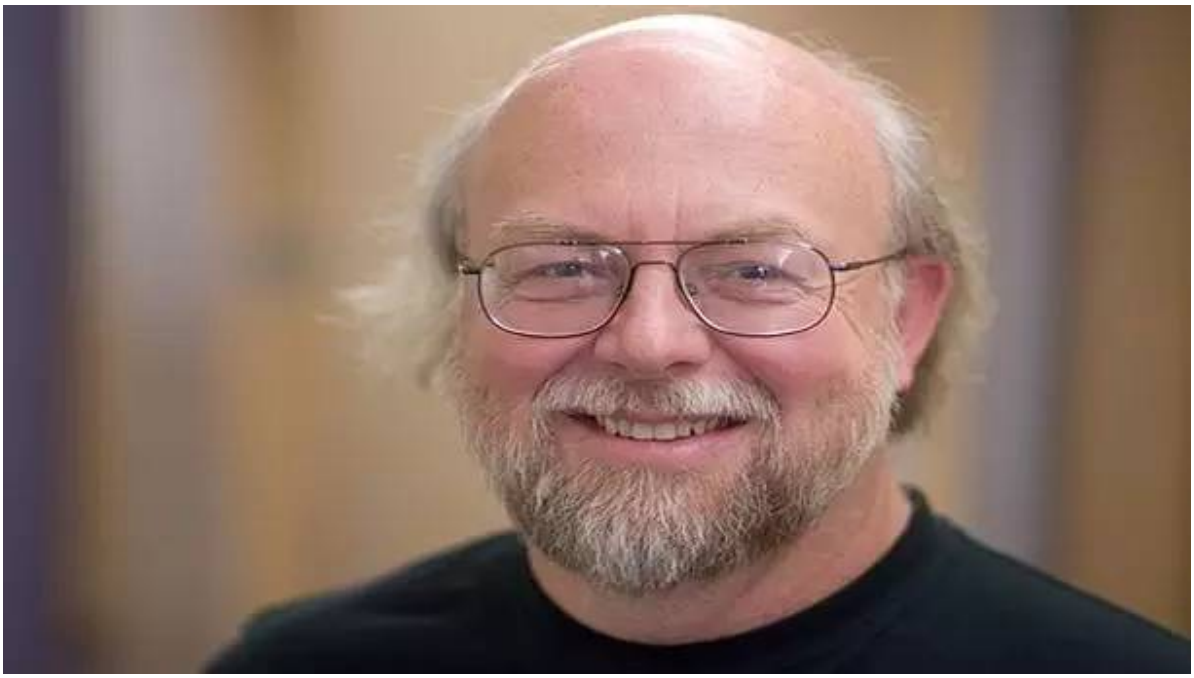


揽一杯java，卷三秋!

让你欲罢不能，陪你入夜，帮你暖胃!

那么今天我们就来刨了java的祖坟!

它的**老爹**就是 **詹姆斯·高斯林（James Gosling）**



当年Java诞生时，是专为**TV机顶盒**所设计的。最初这个语言只有一个代号，叫**Green**（呼伦贝尔大草原，绿的发慌。**）项目。有一天，Gosling的开发小组决定给这个语言起一个名字。原本大家一致想叫它**Oak，**Oak是一种**橡树**的名字，这种树在硅谷非常多，很有代表性。原本这名字已经定下来了，但就在产品即将发布的关键时期，律师却发现，已经有另一家公司**注册**了这个名字，这个名字**不可能再用**了。

大家都觉得这个语言非常的棒！然后他们就找了一些起名专家起名字。

- 排在第一位的是**Silk**（丝绸） 但是被它老爹一票否决了。

- 排在第二和第三的都没有通过商标律师这一关。 当时没有合适的名字甚至**延误**了它面世的时间。
- 詹姆斯·高斯林最喜欢的是排在第三位的Lyric（抒情诗），但也已经被注册。
- 传说当时头脑风暴的时候**马克**(话说《灵笼》真好看)面前的一杯**咖啡上面写着java**，他就想到了这个名字。

Java是印度尼西亚**爪哇**岛的英文名称，因盛产咖啡而闻名。国外的许多咖啡店用Java来命名或宣传，以彰显其咖啡的**品质**。

Java语言中的许多库类名称，多与**咖啡**有关，如JavaBeans(咖啡豆)、NetBeans(网络豆)以及ObjectBeans (对象豆)等等。

Android的前世今生

既然我们已经刨完了爪哇的祖坟，那Android的祖坟也不能放过

历程

2003年10月，Andy Rubin（安迪鲁宾）等人创建Android公司，并组建Android团队。

因为这个Andy Rubin 是一个机器人爱好者，所以就取名为Android（机器人）

2005年8月17日，Google低调收购了成立仅22个月的高科技企业Android及其团队。安迪鲁宾成为Google公司工程部副总裁，继续负责Android项目。

然后他们就做了一份十几年的甜品

- 吃货的成长历程

甜点命名法开始于Android 1.5发布的时候。作为每个版本代表的甜点的尺寸越变越大，然后按照26个字母顺序，基本可以说是一部甜品进化史：

- 甜品进化史
    - Android 1.5 Cupcake（**纸杯蛋糕**）
      Android 1.6 Donut（**甜甜圈**）
      Android 2.0/2.1 Eclair （**松饼**）
      Android 2.2 Froyo（**冻酸奶**）
      Android 2.3 Gingerbread（**姜饼**）
      Android 3.0 Honeycomb （**蜂巢**）
      Android 4.0 Ice Cream Sandwich（**冰激凌三明治**）
      Android 4.1/4.2 Jelly Bean（**果冻豆**）
      Android 4.4 KitKat（**奇巧巧克力**）
      Android 5.0 Lollipop（**棒棒糖**）
      Android 6.0 Marshmallow（**棉花糖**）
      Android 7.0 Nougat（**牛轧（ga）糖** （牛gá糖））
      Android 8.0 Oreo（**奥利奥**）

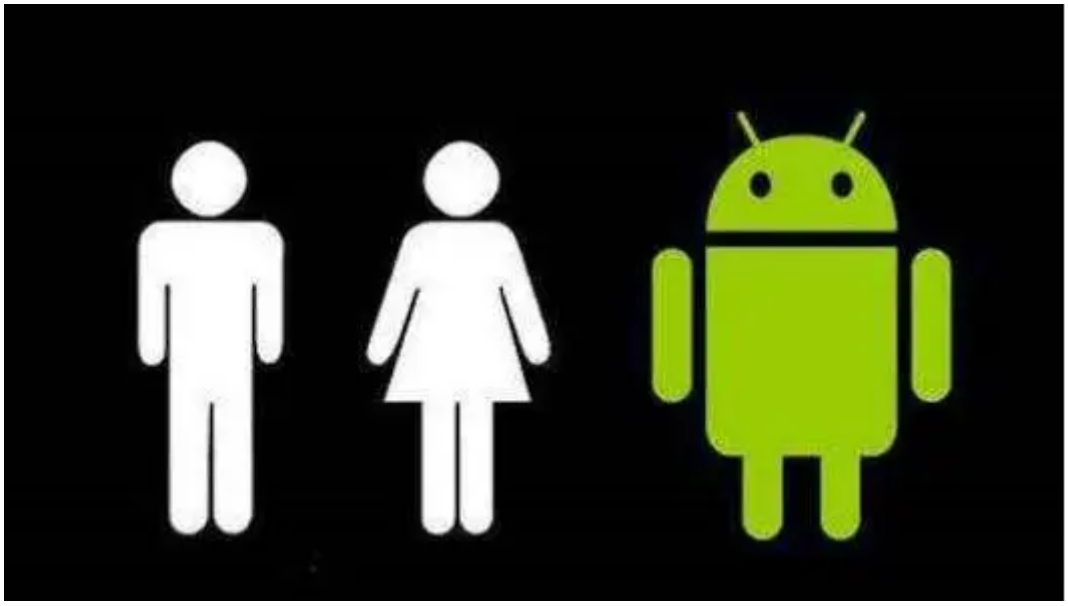
      Android 9.0 Pie：**馅饼**

Android 10 Q版本，Google取消了这一传统，但仍旧保留了内部代号

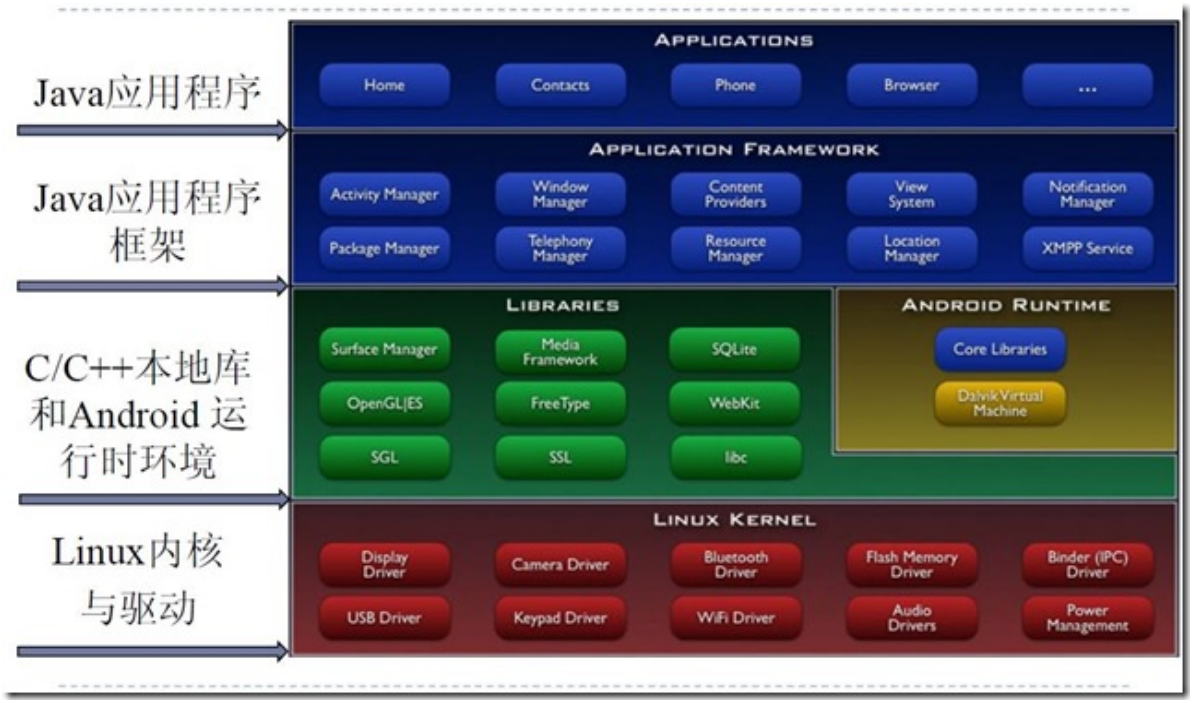
据说Android 13 叫**Tiramisu，提拉米苏**

Logo

- **设计灵感源于男女厕所门上的图形符号**
- 没想到甜品这玩意居然跟厕所有过勾当 以后再也不吃了



介绍Android系统层次



想要详细了解Android 系统架构的同学请点我]

计算机为什么要使用二进制?

我们来回想一下在原始社会我们人类是怎样计数的，我们是通过打结🪢来计数的。

而到了后来的不过是罗马数字还是阿拉伯数字计数。

我们来看看计数的本质，我觉得是一种状态，比如说一根绳子上面没有结这是一种状态，有一个结又是另一种状态，有两个结又是一种新的状态.....　　原始社会根据绳子的不同状态来判断数量的多少。 又比如0-9这十个数字 我们可以把它看作是**十种状态** 用这十种十种状态就可以来表示0-9的数量。**所以我们要计数其实就是要找到不同的状态。**

电子计算机出现以后，使用**电子管来表示十种状态过于复杂**，所以所有的电子计算机中只有**两种基本的状态**，**开**和**关**。

也就是说，**电子管的两种状态**决定了**以电子管为基础**的电子计算机采用**二进制**来表示**数字**和**数据**。

电脑的基层部件是由**集成电路**组成的，这些集成电路可以看成是一个个**门电路**组成，（当然事实上没有这么简单的）。当计算机工作的时候，电路通电工作，于是每个输出端就有了电压。**电压的高低通过模数转换即转换成了二进制**：**高电平是由1表示，低电平由0表示。**也就是说将模拟路转换成为数字电路。这里的高电平与低电平可以人为确定，一般地，2.5伏以下即为低电平，3.2伏以上为高电平。 电子计算机能以极高速度进行信息处理和加工，包括数据处理和加工，而且有极大的信息存储能力。

数据在计算机中以器件的**物理状态(比如说 南北磁极)**表示，采用**二进制数字系统**，计算机处理所有的**字符**或**符号**也要用**二进制编码**来表示。

当然，做成能够表示0～9这10种状态的开关，进而让计算机采用10进制计数法，这在理论上也是可能的。但是电路就**会变得非常的复杂**，**性能也会较差**，**抗干扰能力也会大大降低**，**生产的费用也会大大的提高。**

原因

1.从可行性来说，采用二进制，只有0和1两个状态，能够表示0和1两种状态的电子器件有很多，比如开关的**接通和断开**、**晶体管的导通和截止**、**磁原件的正负剩磁**、**电位电平的高低**等都可以**表示**0和1两个数。使用二进制，电子器件具有实现的**可行性。**

2.从运算的简易性来说，二进制的**运算法则**少，运算简单，使计算机运算器的**硬件结构大大简化**（十进制乘法九九口诀有**55条公式**，而二进制乘法只有**四条规则**）。

3.从逻辑上讲，由于二进制0和1正好和逻辑代码**假和真**相对应，有逻辑代数的理论基础，用二进制表示二值**逻辑很自然。**

二进制简单介绍

1. 一种逢2进1的进制，即只使用**0、1**来**表示数值。**在计算机技术中广泛利用。机器可以使用**高低电平**来表示二进制，**二进制可以用来表示数字**，**用数字就可以表示字母。。。**依次类推，从而造就我们如今的信息时代。下面是(8bit原码表示)举例：
2. **.bit:位 （小写b) 也称比特**

   是英文 binary digit的缩写 二进制数系统中，每个0或1就是一个位(bit)

   位是数据存储（计算机中信息）的**最小单位**
   计算机中的CPU位数指的是CPU一次能处理的最大位数。例如32位计算机的CPU一次最多能处理32位（比特）数据

- 0=0000 0000
- 1=0000 0001
- 2=0000 0010
- 3=0000 0011
- 4=0000 0100
- 5=0000 0101
- 6=0000 0110
- 7=0000 0111
- 8=0000 1000
- 9=0000 1001
- 10=0000 1010

单位

不用担心，仅作为了解，现阶段不需要特别深入。

- 位作为信息中的最小单位。一位就表示一个二进制数字。
- 8bit = 1Byte（字节）　1024 Byte = 1Kb
- 1024Kb = 1Mb　1024Mb = 1Gb

补码，原码，反码

原码

用二进制定点表示法，**最高位**是**符号位**，**0**代表**正数**，**1**代表**负数。**其余的**余位**表示**数值**的**大小。**

例如：

- 10的原码就是 **0**0001010
- -10的原码就是**1**0001010

反码

正数的反码和其**原码相同**，**负数**的反码是对其**原码逐位取反**，但**符号位除外。**（即把除去符号位的剩余位1改成0，0改成1）

例如：

- 10的反码是　**0**0001010
- -10的反码就是**1**1110101

补码

正数的补码等于其**原码**，**负数**的补码等于负数的**反码+1**.

例如：

- 10的补码是00001010
- -10的补码就是11110110

为什么要讲这个来折磨我?

首先, 因为人脑可以知道**第一位是符号位**, 在计算的时候我们会根据符号位, 选择对真值(一个数在计算机中的二进制表示形式,叫做这个数的**机器数。**机器数是带符号的。**除开符号位**的真正数值称为机器数的**真值**)区域的加减。

但是对于计算机, 加减乘数已经是**最基础**

的运算, 要设计的尽量简单. 计算机**辨别"符号位"**显然会让计算机的**基础电路设计**变得**十分复杂!**

于是人们想出了将符号位也参与运算的方法. 我们知道, 根据**运算法则**减去一个正数等于加上一个负数, 即:

1-1 = 1 + (-1) = 0

其实就是他们想让符号位也参与计算机的运算 但是计算机不能够识别符号位

所以**机器**可以**只有加法而没有减法**（计算机只会加法而不会减法）, 这样计算机运算的设计就更简单了.

人们开始探索 将符号位参与运算, 并且只保留加法的方法.计算十进制的表达式: 1-1=1+（-1）=0

原码运算

首先来看原码:如果用原码表示, 让**符号位也参与计算**, 显然对于减法来说, 结果是不正确的.这也就是**为何计算机内部不使用原码表示一个数.**

1 - 1 = 1 + (-1) = [00000001]原 + [10000001]原 = [10000010]原 = -2

10-1=10+(-1)=[00001010]原 + [10000001]原 = 10001011 ， 即 -11（原码参与加法运算）

反码运算

规则：

- 符号位也参与运算
- 反码的符号位相加之后，如果有进位出现，则要把他送到最低位去相加。
- 反码运算的结果亦为反码。在转换真值时，若符号位为0，数位不变，若符号位为1，应该结果求反才是真值。

为了解决**原码做减法**的问题, 出现了反码:

10-1=10+(-1)=[00001010]原 + [10000001]原 = [00001010]反+[111111110]反=[00001001]原，即9

1-10=1+(-10)=[00000001]反+[11110101]反=[11110110]反=[10001001]原=-9

发现用**反码计算减法,** 结果的**真值**部分是正确的. 但是

1 - 1 = 1 + (-1) = [0000 0001]原 + [1000 0001]原= [0000 0001]反 + [1111 1110]反 = [1111 1111]反 = [1000 0000]原 = -0

而唯一的问题其实就出现在"0"这个特殊的数值上. 虽然人们理解上+0和-0是一样的, 但是0带符号是没有任何意义的. 而且会有**[0000 0000]**原和**[1000 0000]**原**两个编码表示0**

补码运算

规则：

- 结果为两数之和的补码形式。
- 补码运算的结果亦为补码。在转换真值时，若符号位为0，数位不变，若符号位为1，应该结果**逆**（求补码的过程倒回去）才是真值。

1-1=1+（-1）=[00000001]补+[11111111]补 = 00000000=0

10-1=10+（-1）=[00001010]补+[11111111]补=00001001=9

1-10=1+（-10）=[00000001]补+[11110110]补=[11110111]=[10001001]=-9

这样**0**用**[0000 0000]**表示, 而以前出现问题的**-0**则不存在了.而且可以用**[1000 0000]**表示-128:

(-1) + (-127) = [1000 0001]原 + [1111 1111]原 = [1111 1111]补 + [1000 0001]补 = [1000 0000]补 =-128

-1-127的结果应该是-128, 在用**补码运算**的结果中, **[1000 0000]**补 就是-128.

但是注意因为实际上是使用以前的**-0的补码来表示-128**,

所以**-128**并没有**原码**和**反码**表示.(**对-128的补码表示[1000 0000]补算出来的原码是[0000 0000]原, 这是不正确的**)

使用补码, 不仅仅修复了0的符号以及存在两个编码的问题, 而且还能够**多表示一个最低数**.

这就是为什么**8位二进制**, 使用**原码**或**反码**表示的范围为**[-127, +127]**, 而使用**补码**表示的范围为**[-128, 127]**

java的特性简单介绍

- **java具有简单性** 语言的语法与 C 语言和 C++ 语言很相近，使得很多程序员学起来很容易。对 Java 来说，它舍弃了很多 C++ 中难以理解的特性，和多继承等，而且 Java 语言不使用指针，加入了**垃圾回收机制**，解决了程序员需要管理内存的问题，使编程变得更加简单。
- **Java介于编译型语言和解释型语言之间** 编译型语言和解释型语言的区别
- **java是面对对象的** 你有对象吗？没有的话 就来new一个吧。面向过程与对象的区别和联系
- **java 是多线程的** 能够使应用程序在同一时间并行执行多项任务。
- **一次编写，到处运行.（Write once，run anywhere）**
- 还有很多特性.....
- **大家现在不用非常的明白** 我只是向大家简单的介绍一下 以后大家在学习的过程中 **逐步感受java这杯咖啡的浓烈**
- **听说java和凌晨更配哦**（有一说一 大家少熬夜）

java的简单语法

对象：**对象是一个类的实例**，有**状态**（**属性**）和**行为**（**函数方法**）例如，一条狗是一个对象，

它的**状态**有：颜色、名字、品种；**行为**有：摇尾巴、叫、吃等。

类：**类是一个模板**，它描述一类对象的行为和状态

方法：方法就是**行为**，一个**类**可以有很多方法，逻辑运算、数据修改以及所有动作都是在方法中完成的。

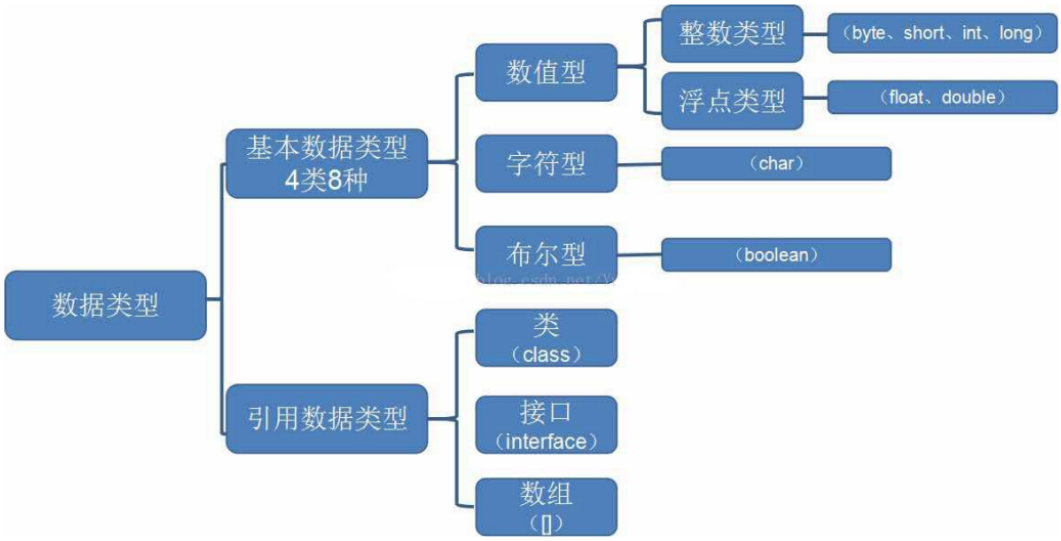
实例变量：每个对象都有独特的实例，对象的状态由这些实例变量的值确定。

java基本数据类型

学习编程的第一步

学习基础的语法，
数据类型、变量

学习如何使用
Google.



基本数据类型

byte：

- byte 数据类型是**8位**一个字节、**有符号**的，以二进制**补码**表示的整数；
- 最小值是 **-128（-2^7）**
- 最大值是 **127（2^7-1）**
- 默认值是 **0**；
- byte 类型用在**大型数组**中**节约空间**，主要**代替整数**，因为 byte 变量占用的空间只有 int 类型的**四分之一**
- 例子：byte a = 100，byte b = -50。

short：

- short 数据类型是 **16 位**两个字节、**有符号**的以二进制**补码**表示的整数
- 最小值是 **-32768（-2^15）**；
- 最大值是 **32767（2^15 - 1）**；
- Short 数据类型也可以像 byte 那样节省空间。一个short变量是int型变量所占空间的**二分之一**；
- 默认值是 **0**；
- 例子：short s = 1000，short r = -20000。

int：

- int 数据类型是**32位**四个字节、有符号的以二进制**补码**表示的整数；
- 最小值是 **-2,147,483,648（-2^31）**；
- 最大值是 **2,147,483,647（2^31 - 1）**；
- 一般地整型变量**默认**为 int 类型；
- 默认值是 **0**；
- 例子：int a = 100000, int b = -200000。

long：

- long 数据类型是 **64 位**八个字节、有符号的以二进制**补码**表示的整数；
- 最小值是 **-9,223,372,036,854,775,808（-2^63）**；
- 最大值是 **9,223,372,036,854,775,807（2^63 -1）**；
- 这种类型主要使用在需要**比较大整数**的系统上；
- 默认值是 **0L**；

- 例子： long a = 100000L，Long b = -200000L。
  "L"理论上不分大小写，但是若写成"l"容易与数字"1"混淆，不容易分辩。**所以最好大写。**

float:

- float 数据类型是单精度、**32位**、符合IEEE 754标准的浮点数；
- float 在**储存大型浮点数组**的时候可节省内存空间；
- 默认值是 **0.0f**；
- 范围
- 浮点数不能用来表示精确的值，如货币；
- 例子：float f1 = 234.5f。

double:

- double 数据类型是双精度、**64 位**、符合 IEEE 754 标准的浮点数；
- 浮点数的默认类型为 double 类型；
- double类型同样不能表示精确的值，如货币；
- 例子：

```
double   d1  = 7D ;
double   d2  = 7.;
double   d3  =  8.0;
double   d4  =  8.D;
double   d5  =  12.9867;
```

7 是一个 int 字面量，而 7D，7. 和 8.0 是 double 字面量。

boolean:

- boolean数据类型表示一位的信息；
- 只有两个取值：true 和 false；
- 这种类型只作为一种标志来记录 true/false 情况；
- 默认值是 **false**；
- 例子：boolean one = true。

char:

- char 类型是一个单一的 **16 位** Unicode 字符；
- char 数据类型可以**储存任何字符**
- 例子：char letter = 'A';

代码

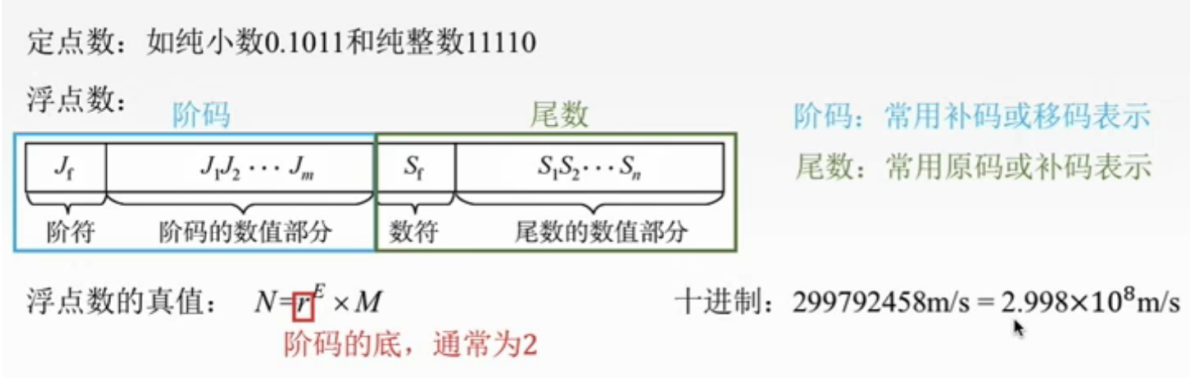
```
// byte
        System.out.println("基本类型：byte 二进制位数：" + Byte.SIZE);
        System.out.println("包装类：java.lang.Byte");
        System.out.println("最小值：Byte.MIN_VALUE=" + Byte.MIN_VALUE);
        System.out.println("最大值：Byte.MAX_VALUE=" + Byte.MAX_VALUE);
        System.out.println();

        // short
        System.out.println("基本类型：short 二进制位数：" + Short.SIZE);
        System.out.println("包装类：java.lang.Short");
        System.out.println("最小值：Short.MIN_VALUE=" + Short.MIN_VALUE);
        System.out.println("最大值：Short.MAX_VALUE=" + Short.MAX_VALUE);
```

```
        System.out.println();

        // int
        System.out.println("基本类型：int 二进制位数：" + Integer.SIZE);
        System.out.println("包装类：java.lang.Integer");
        System.out.println("最小值：Integer.MIN_VALUE=" + Integer.MIN_VALUE);

        System.out.println("最大值：Integer.MAX_VALUE=" + Integer.MAX_VALUE);

        System.out.println();

        // long
        System.out.println("基本类型：long 二进制位数：" + Long.SIZE);
        System.out.println("包装类：java.lang.Long");
        System.out.println("最小值：Long.MIN_VALUE=" + Long.MIN_VALUE);
        System.out.println("最大值：Long.MAX_VALUE=" + Long.MAX_VALUE);
        System.out.println();

        // float
        System.out.println("基本类型：float 二进制位数：" + Float.SIZE);
        System.out.println("包装类：java.lang.Float");
        System.out.println("最小值：Float.MIN_VALUE=" + Float.MIN_VALUE);
        System.out.println("最大值：Float.MAX_VALUE=" + Float.MAX_VALUE);
        System.out.println();

        // double
        System.out.println("基本类型：double 二进制位数：" + Double.SIZE);
        System.out.println("包装类：java.lang.Double");
        System.out.println("最小值：Double.MIN_VALUE=" + Double.MIN_VALUE);
        System.out.println("最大值：Double.MAX_VALUE=" + Double.MAX_VALUE);
        System.out.println();

        // char
        System.out.println("基本类型：char 二进制位数：" + Character.SIZE);
        System.out.println("包装类：java.lang.Character");
        // 以数值形式而不是字符形式将Character.MIN_VALUE输出到控制台
        System.out.println("最小值：Character.MIN_VALUE="
                + (int) Character.MIN_VALUE);
        // 以数值形式而不是字符形式将Character.MAX_VALUE输出到控制台
        System.out.println("最大值：Character.MAX_VALUE="
                + (int) Character.MAX_VALUE);
```

问题

明明float只占32位 但是它的范围却比long类型大得多 这是为什么南?

浮点数在计算机当中的**存储方式**与整数类型并**不相同**，所以他们**不能**够**相互转化**（类型强转）**采用**的其实是一种**类似于科学计数法的方式**



实数N可以表示为：

```
N=S×r^j
```

s为尾数，j为阶码，r为基数，对于二进制来说一般r=2.

例如：**10.0111=0.100111×2^10**

引用数据类型

- 在Java中，引用类型的变量非常**类似**于C/C++的**指针。**引用类型指向一个对象，指向对象的变量是引用变量。这些变量在声明时被指定为一个特定的类型，比如 Employee、Puppy 等。变量一旦声明后，类型就不能被改变了。
- **对象**、**数组**都是引用数据类型。
- 所有引用类型的默认值都是null。
- 一个引用变量可以用来引用任何**与之兼容的类型。**
- 例子：Site site = new Site("Runoob")。

对于**基本类型**来说，**栈**区域包含的是基本类型的内容，也就是**值**；

例如:

```
int a =2;
```

对于**引用类型**来说，**栈**区域包含的是指向真正内容的**指针**（真正内容在**堆中的地址**），真正的 内容被分配在了**堆**中。

例如:

```
Person dingshen=new Person();
```

大家现在不清楚也没有关系，但是以后敲代码或者学习的时候多思考 你们就可以理解了

Java 常量

常量在程序运行时是**不能被修改**的。

在 Java 中使用 **final 关键字**来修饰常量，声明方式和变量类似：

```
final double PI = 3.1415927;
```

类型转换

这个知识点 大家自己下去玩　我今天就不说了　也很简单的　相信大家都会

java变量类型

- **类变量**：独立于方法之外的变量，用 **static** 修饰。(也称为静态变量)
- **实例变量**：独立于方法之外的变量，不过没有 static 修饰。
- **局部变量**：类的方法中的变量。（不能使用private等修饰）

```
public class Variable{
    static int allClicks=0;    // 类变量

    String str="hello world";  // 实例变量

    public void method(){

        int i =0;  // 局部变量

    }


}
```

局部变量

- 局部变量声明在**方法**、**构造方法**或者**语句块**中；
- 局部变量在方法、构造方法、或者语句块**被执行的时候创建**，当它们执行完成后，**变量将会被销毁**；（函数的调用就对应着压栈和出栈）
- **访问修饰符不能**用于局部变量；
- 局部变量**只**在声明它的**方法**、**构造方法**或者**语句块**中**可见**；
- 局部变量是在**栈**上分配的（大家只需要了解即可 并不要求现在就掌握）
- 局部变量**没有默认值**，所以局部变量被声明后，必须经过初始化，才可以使用。

```
   public void pupAge(){
       int age = 0;
       age = age + 7;
       System.out.println("小狗的年龄是: " + age);
    }

    public static void main(String[] args){
       Test test = new Test();
       test.pupAge();
    }
```

实例变量

- 实例变量**声明在一个类中**，但在方法、构造方法和语句块**之外**；
- 当一个对象被实例化之后，每个**实例变量的值就跟着确定**；
- 实例变量在**对象创建的时候创建**，**在对象被销毁的时候销毁**；

```
import java.io.*;
public class Employee{
   // 这个实例变量对子类可见
   public String name;
   // 私有变量，仅在该类可见
   private double salary;
   //在构造器中对name赋值
   public Employee (String empName){
       name = empName;
   }
   //设定salary的值
   public void setSalary(double empSal){
       salary = empSal;
   }
   // 打印信息
   public void printEmp(){
       System.out.println("名字 : " + name );
       System.out.println("薪水 : " + salary);
   }

   public static void main(String[] args){
       Employee empOne = new Employee("RUNOOB");
       empOne.setSalary(1000.0);
       empOne.printEmp();
   }
}
```

类变量

- 类变量也称为静态变量，在类中以 static 关键字声明，但必须在方法之外。
- 静态变量储存在**静态存储区**。经常被声明为常量，**很少单独使用static声明变量**。
- 静态变量在第一次**被访问**时**创建**，在程序**结束时销毁**

```
//salary是静态的私有变量
   private static double salary;
   // DEPARTMENT是一个常量
   public static final String DEPARTMENT = "开发人员";
   public static void main(String[] args){
   salary = 10000;
       System.out.println(DEPARTMENT+"平均工资:"+salary);
   }
```

也许讲到这里大多数人其实可能有点迷糊了 但是没事 现在大家没有掌握的很好 也没有关系 在后面的学习当中多敲代码 慢慢的体会

java修饰符

访问修饰符

- Java中，可以使用Java中，可以**使用访问修饰符**来**保护**对**类**、**变量**、**方法**和**构造方法**的**访问**。
- **default** (即默认，什么也不写）：在同一包内可见，不使用任何修饰符。使用对象：**类**、**接口**、**变量**、**方法**。
- **private** : 在同一类内可见。使用对象：变量、方法。 **注意：不能修饰类（外部类）**
- **public** : 对所有类可见。使用对象：**类**、**接口**、**变量**、**方法**
- **protected** : 对**同一包**内的类和**所有子类**可见。使用对象：**变量**、**方法**。 **注意：不能修饰类**（外部类）。
-

访问包位置	类修饰符		
	private	protected	public
本类	可见	可见	可见
同包其他类或子类	不可见	可见	可见
其他包的类或子类	不可见	不可见	可见

非访问修饰符

static

静态变量：

static 关键字用来声明独立于对象的静态变量，无论一个类实例化多少对象，它的**静态变量只有一份拷贝**。 静态变量也被称为类变量。**局部变量不能被声明为 static 变量。**

静态方法：

- static 关键字用来声明独立于对象的静态方法。**静态方法**不能使用类的**非静态变量**。
- （大家不用现在就弄清楚原因，可以在以后的学习中 慢慢体会）

对类变量和方法的访问可以直接使用 **classname.variablename** 和**classname.methodname** 的方式访问

```
public class InstanceCounter {
   private static int numInstances = 0;
```

```
   protected static int getCount() {
      return numInstances;
   }

   private static void addInstance() {
      numInstances++;
   }

   InstanceCounter() {
      InstanceCounter.addInstance();
   }

   public static void main(String[] arguments) {
      System.out.println("Starting with " +
      InstanceCounter.getCount() + " instances");
      for (int i = 0; i < 500; ++i){
         new InstanceCounter();
          }
      System.out.println("Created " +
      InstanceCounter.getCount() + " instances");
   }
}
```

final

final变量:

final表示"最后的，**最终**的"含义，变量**一旦赋值后**，**不能被重新赋值。**被 final 修饰的实例变量必须**显式指定初始值**，final 修饰符通常和 static 修饰符一起使用来创建类常量

final方法

父类中的final方法**可**以被子类**继承**，但**不**能被**重写。**所以，只需要记住，声明final方法的主要目的是防止该方法的内容被修改。

final类🧐

final类不能被**继承**，没有类能够继承final类的任何特性。

abstract

这个是抽象修饰符 我今天在这里就不在赘述了

Java运算符

算术运算符

(+,-,*,/,%,++,--)

```
 int a = 10;
     int b = 20;
     int c = 25;
     int d = 25;
     System.out.println("a + b = " + (a + b) );
     System.out.println("a - b = " + (a - b) );
     System.out.println("a * b = " + (a * b) );
     System.out.println("b / a = " + (b / a) );
```

```
9       System.out.println("b % a = " + (b % a) );
10      System.out.println("c % a = " + (c % a) );
11      System.out.println("a++   = " +  (a++) );
12      System.out.println("a--   = " +  (a--) );
13      // 查看  d++ 与 ++d 的不同
14      System.out.println("d++   = " +  (d++) );
15      System.out.println("++d   = " +  (++d) );
```

关系运算符

（只会返回ture或者false），包括（==，！=，>, <, >=,<=）

```
1       int a = 10;
2       int b = 20;
3       System.out.println("a == b = " + (a == b) );
4       System.out.println("a != b = " + (a != b) );
5       System.out.println("a > b = " + (a > b) );
6       System.out.println("a < b = " + (a < b) );
7       System.out.println("b >= a = " + (b >= a) );
8       System.out.println("b <= a = " + (b <= a) );
```

位运算符

因为现阶段用的极少，所以就先不讲了

（可能**数电**（这个课蛮有意思的），或者 面Android涉及到Color的时候会用到）

逻辑运算符

操作符	描述	例子
&&	称为逻辑与运算符。当且仅当两个操作数都为真，条件才为真。	（A && B）为假。
||	称为逻辑或操作符。如果任何两个操作数任何一个为真，条件为真。	（A || B）为真。
!	称为逻辑非运算符。用来反转操作数的逻辑状态。如果条件为true，则逻辑非运算符将得到false。	！（A && B）为真。

赋值运算符

(=，+=，-=，*=，/=，（%）=....)

三元运算符

（? : ），是if...else的缩写。

```
variable x = (expression) ? value if true : value if false
```

```
    int a , b;
     a = 10;
     // 如果 a 等于 1 成立，则设置 b 为 20，否则为 30
     b = (a == 1) ? 20 : 30;
     System.out.println( "Value of b is : " +  b );

     // 如果 a 等于 10 成立，则设置 b 为 20，否则为 30
     b = (a == 10) ? 20 : 30;
     System.out.println( "Value of b is : " + b );
```

Java运算符优先级

这个一般的开发中不怎么会为难你，除非是考试，或者炫技的程序员（不过这种一般会被喷）。

没人会这样写代码吧：

```
        int y = 0;
        int x = 5;
        y = x++ + ++x + x * 10 - ++y + y++ + ++y;
        // 即y = 5 + 7 + 70-1+1+3 = 85
        System.out.println(y);
```

Java输入输出函数

nextInt()

nextInt(): it only reads the int value, nextInt() places the cursor（光标） in the same line after reading the input.（nextInt()只读取数值，剩下"\n"还没有读取，并将cursor放在本行中）。

next()

read the input only till the space. It can't read two words separated by space. Also, next() places the cursor in the same line after reading the input.（next()只读空格之前的数据，并且cursor指向本行）

next() 方法遇见第一个有效字符（非空格，非换行符）时，开始扫描，当遇见第一个分隔符或结束符(空格或换行符)时，结束扫描，获取扫描到的内容，**即获得第一个扫描到的不含空格、换行符的单个字符串。**

nextLine()

reads input including space between the words (that is, it reads till the end of line \n). Once the input is read, nextLine() positions the cursor in the next line.
nextLine()时，则可以扫描到一行内容并作为一个字符串而被获取到。

由于sc.nextLine()和sc.next()处理结束键的方式不一样，所以要避免混合使用。
如果在sc.nextLine()之前调用过sc.next()此类函数，应该加入一个无效的sc.nextLine()作缓冲，抵消之前函数遗留的enter键。

例子

```
 String s1,s2;
先用nextline，再用next的函数没问题
        Scanner sc=new Scanner(System.in);
        System.out.print("请输入第一个字符串：");
        s1=sc.nextLine();
        System.out.print("请输入第二个字符串：");
        s2=sc.next();
        System.out.println("输入的字符串是："+s1+" "+s2);

先用next，再用nextline就有问题   因为它会吃掉上一次的回车
 String s1,s2;
        Scanner sc=new Scanner(System.in);
        System.out.print("请输入第一个字符串：");
        s2=sc.next();
        //s1=sc.nextLine();
        System.out.print("请输入第二个字符串：");
        s1=sc.nextLine();
        System.out.println("输入的字符串是："+s1+" "+s2);

```

流程控制

条件语句

if

- **if**

```
if(布尔表达式){
   //如果布尔表达式的值为true
}
```

- **if....else**

```
if(布尔表达式){
   //如果布尔表达式的值为true
}else{
   //如果布尔表达式的值为false
}
```

- **if...else if ...else**

```
if(布尔表达式 1){
   //如果布尔表达式 1的值为true执行代码
}else if(布尔表达式 2){
   //如果布尔表达式 2的值为true执行代码
}else if(布尔表达式 3){
   //如果布尔表达式 3的值为true执行代码
}else {
   //如果以上布尔表达式都不为true执行代码
}
```

- **嵌套**

```
int x = 30;
      int y = 10;

      if( x == 30 ){
         if( y == 10 ){
             System.out.print("X = 30 and Y = 10");
          }
       }
```

Switch

- switch 语句中的变量类型可以是： byte、short、int 或者 char。从 Java SE 7 开始，switch 支持字符串 String 类型了，同时 case 标签必须为字符串常量或字面量。

```
switch(expression){
    case value :
       //语句
       break; //可选
    case value :
       //语句
       break; //可选
    //你可以有任意数量的case语句
    default : //可选
       //语句
}
```

```
 char grade = 'C';
      switch(grade)
      {
         case 'A' :
            System.out.println("优秀");
            break;
         case 'B' :
         case 'C' :
            System.out.println("良好");
            break;
         case 'D' :
            System.out.println("及格");
            break;
         case 'F' :
            System.out.println("你需要再努力努力");
            break;
         default :
```

```
            System.out.println("未知等级");
        }
        System.out.println
            ("你的等级是 " + grade);
```

循环语句

while

语法

```
while( 布尔表达式 ) {
  //循环内容
}
```

例子:

```
 int x = 10;
      while( x < 20 ) {
         System.out.print("value of x : " + x );
         x++;
         System.out.print("\n");
      }
```

do..while

对于 while 语句而言，如果不满足条件，则不能进入循环。但有时候我们需要即使不满足条件，也至少执行一次。

do...while 循环和 while 循环相似，不同的是，do...while 循环至少会执行一次。

语法

```
do {
       //代码语句
}while(布尔表达式);
```

例子:

```
int x = 10;
      do{
         System.out.print("value of x : " + x );
         x++;
         System.out.print("\n");
      }while( x < 20 );
```

for

虽然所有循环结构都可以用 while 或者 do...while表示，但 Java 提供了另一种语句 —— for 循环，使一些循环结构变得更加简单。

for循环执行的次数是在执行前就确定的。语法格式如下：

```
for(初始化; 布尔表达式; 更新) {
    //代码语句
}
```

关于 for 循环有以下几点说明：

- 最先执行初始化步骤。可以声明一种类型，但可初始化**一个或多个**循环控制变量，也可以是**空语句**。
- 然后，检测布尔表达式的值。如果为 true，循环体被执行。如果为false，循环终止，开始执行循环体后面的语句。
- 执行一次循环后，更新循环控制变量。
- 再次检测布尔表达式。循环执行上面的过程。

例子：

```
for(int x = 10; x < 20; x = x+1) {
         System.out.print("value of x : " + x );
         System.out.print("\n");
      }
```

break

- break 主要用在循环语句或者 switch 语句中，用来**跳出整个语句块**。
- break 跳出**最里层**的循环，并且继续执行**该循环下面**的语句。

```
 int [] numbers = {10, 20, 30, 40, 50};
      for(int x : numbers ) {
         // x 等于 30 时跳出循环
         if( x == 30 ) {
            break;
         }
         System.out.print( x );
         System.out.print("\n");
      }


```

跳出内层循环

```
for (int i = 0; i < 5; i++) {
      System.out.print("Pass " + i + ": ");
      for (int j = 0; j < 100; j++) {
        if (j == 10)
          break; // terminate loop if j is 10
        System.out.print(j + " ");
      }
      System.out.println();
    }
    System.out.println("Loops complete.");
```

跳出外层循环

```
outer: for (int i = 0; i < 10; i++) {
    for (int j = 0; j < 10; j++) {
        if (j + 1 < i) {
            System.out.println("退出外层循环的时候i="+i+"  j="+j);
            break outer;
        }

        System.out.println("i*j=" + (i * j));
    }
}
System.out.println();
```

continue

- continue 适用于**任何循环控制结构中**。作用是让程序立刻跳转到**下一次循环的迭代。**
- 在 for 循环中，continue 语句使程序立即**跳转**到**更新语句。**
- 在 while 或者 do...while 循环中，程序立即**跳转到**布尔表达式的判断语句。

```
int [] numbers = {10, 20, 30, 40, 50};

     for(int x : numbers ) {
        if( x == 30 ) {
       continue;
        }
        System.out.print( x );
        System.out.print("\n");
     }
```

增强for循环

```
for(声明语句 : 表达式)
{
   //代码句子
}
```

例子：

```
 int [] numbers = {10, 20, 30, 40, 50};

      for(int x : numbers ){
         System.out.print( x );

         System.out.print(",");
      }
      System.out.print("\n");
      String [] names ={"James", "Larry", "Tom", "Lacy"};
      for( String name : names ) {
         System.out.print( name );
         System.out.print(",");
      }
```

数组

存储固定大小的同类型元素

其实数组就相当于一个装同一类型的容器 我们把相同的元素放在一个容器里面 然后对它进行一些统一的操作

特点：

1. **类型相同**
2. **长度固定**
3. **地址空间连续**

声明：

```
double[] myList;          // 首选的方法
或
double myList[];

//下面这些写发全都可以
int[] i;
i=new int[5];
int[] j=new int[5];
int[] k=new int[]{1,2,3,4,5};
int[] l={1,2,3,4,5};
```

创建数组：

Java语言使用**new**操作符来创建数组，语法如下：

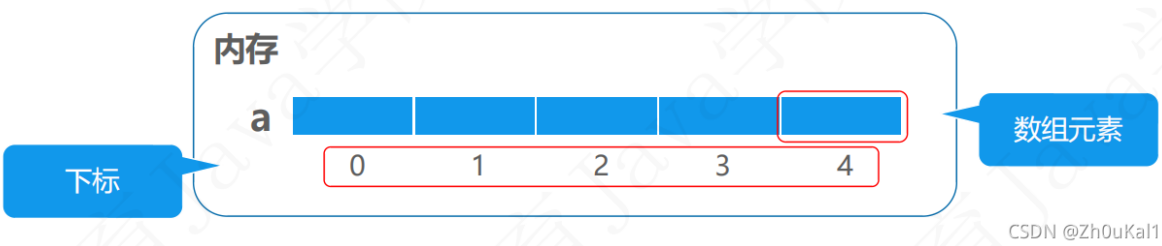
```
arrayRefVar = new dataType[arraySize];
```

上面的语法语句做了两件事：

- 一、使用 dataType[arraySize] 创建了一个数组。
- 二、把新创建的数组的引用赋值给变量 arrayRefVar。

```
dataType[] arrayRefVar = {value0, value1, ..., valuek};
```

组成

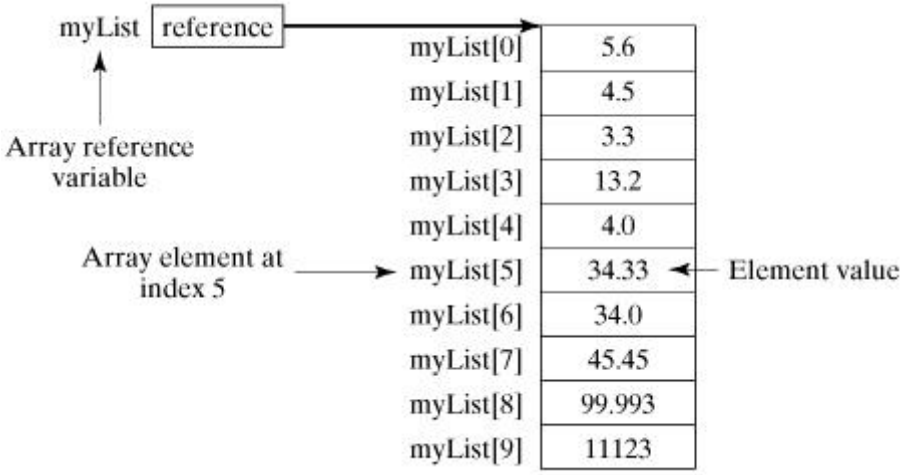


例子：

```
  // 数组大小
      int size = 10;
      // 定义数组
      double[] myList = new double[size];
      myList[0] = 5.6;
      myList[1] = 4.5;
```

```
        myList[2] = 3.3;
        myList[3] = 13.2;
        myList[4] = 4.0;
        myList[5] = 34.33;
        myList[6] = 34.0;
        myList[7] = 45.45;
        myList[8] = 99.993;
        myList[9] = 11123;
        // 计算所有元素的总和
        double total = 0;
        for (int i = 0; i < size; i++) {
           total += myList[i];
        }
        System.out.println("总和为：  " + total);
```

现在我们来对这个数组分析一下



操作数组

普通的for循环

```
double[] myList = {1.9, 2.9, 3.4, 3.5};
        // 打印所有数组元素
        for (int i = 0; i < myList.length; i++) {
           System.out.println(myList[i] + " ");
        }

        // 计算所有元素的总和
        double total = 0;
        for (int i = 0; i < myList.length; i++) {
           total += myList[i];
        }

        System.out.println("Total is " + total);
        // 查找最大元素
        double max = myList[0];
        for (int i = 1; i < myList.length; i++) {
           if (myList[i] > max) max = myList[i];
        }

        System.out.println("Max is " + max);
```

for-each 循环

```
double[] myList = {1.9, 2.9, 3.4, 3.5};
      // 打印所有数组元素
      for (double element: myList) {
         System.out.println(element);
      }
```

多维数组

多维数组可以看成是**数组的数组**，比如**二维数组**就是一个特殊的一维数组，其**每一个元素**都是**一个一维数组**，例如：

```
{{3,3},{2,2},{1, 2},{2, 5}}
```

声明并创建二维数组

```
type[][] typeName = new type[typeLength1][typeLength2];
```

多维数组的遍历（二维数组为例）

```
int[][] arr2 = {
    {1,2},{3,4,5},{6,7,8,9,10}
                };
        int sum2 = 0;
        for (int i=0; i<arr2.length; i++) {
            for (int j=0; j<arr2[i].length; j++) {
                 //System.out.println(arr2[i][j])
                sum2 += arr2[i][j];
            }
        }
        System.out.println("sum2= "+ sum2);
```

哪怕是三位的数组 其实也只是 持续套娃的过程

```
 int t[][][]={{{3, 3}},{{2, 2, 6, 9}},{{1}},{{2}}};

 System.out.println(t[0][0][0]);
```

排序算法

排序算法是《数据结构与算法》中最基本的算法之一。

排序算法可以分为**内部排序**和**外部排序**，内部排序是数据记录在内存中进行排序，而外部排序是因排序的数据很大，一次不能容纳全部的排序记录，在排序过程中需要访问外存。常见的内部排序算法有：**插入排序、希尔排序、选择排序、冒泡排序、归并排序、快速排序、堆排序、基数排序等。**用一张图概括

排序算法	平均时间复杂度	最好情况	最坏情况	空间复杂度	排序方式	稳定性
冒泡排序	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	In-place	稳定
选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	In-place	不稳定
插入排序	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	In-place	稳定
希尔排序	$O(n \log n)$	$O(n \log^2 n)$	$O(n \log^2 n)$	$O(1)$	In-place	不稳定
归并排序	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$	Out-place	稳定
快速排序	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$O(\log n)$	In-place	不稳定
堆排序	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(1)$	In-place	不稳定
计数排序	$O(n + k)$	$O(n + k)$	$O(n + k)$	$O(k)$	Out-place	稳定
桶排序	$O(n + k)$	$O(n + k)$	$O(n^2)$	$O(n + k)$	Out-place	稳定
基数排序	$O(n \times k)$	$O(n \times k)$	$O(n \times k)$	$O(n + k)$	Out-place	稳定

冒泡排序

算法简介:

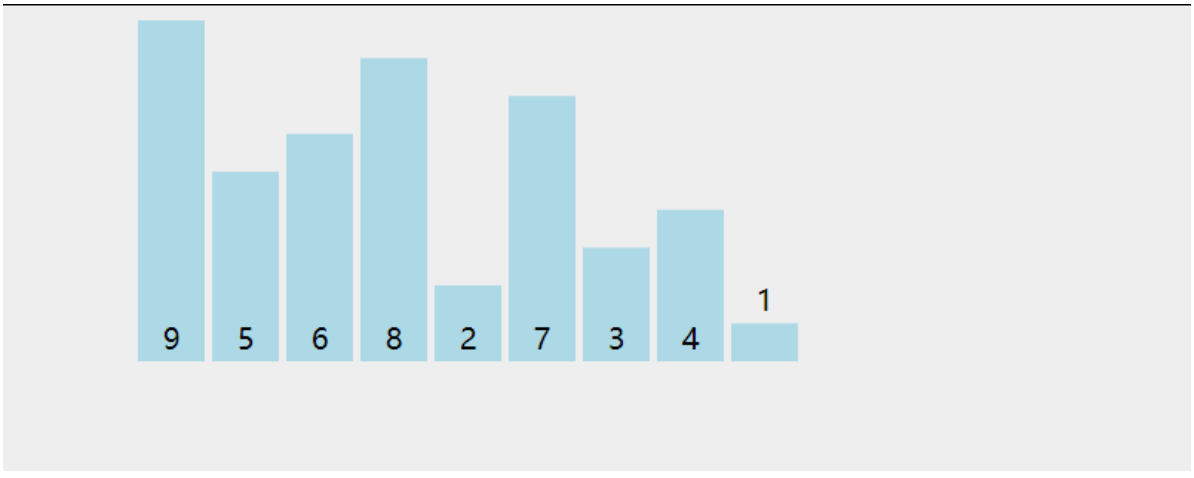
1.比较**相邻**的元素，**前一个比后一个大**（或者前一个比后一个小）调换位置

2.每一对**相邻**的元素进行**重复**的工作，从开始对一直到结尾对，这步完成后，**结尾**为**最大**或**最小**的数.

3.针对**除了最后一个元素**重复进行上面的步骤。

4.**重复**1-3步骤直到完成排序

动画演示:



核心代码

```
int temp=arr[j];
arr[j]=arr[j+1];
arr[j+1]=temp;
```

代码:

```
   int[] arr = {9, 5, 6, 8, 2, 7, 3, 4, 1};//创建数组
        System.out.println("排序前");
        showArr(arr);//打印显示排序前
        bubbleSort(arr);
        System.out.println("排序后");
        showArr(arr);
    }

    /**
     * 循环实现冒泡排序
     *
     * @param arr
     */
    private static void bubbleSort(int[] arr) {

        for (int i = 0; i < arr.length - 1; i++) {
            for (int j = 0; j < arr.length - i - 1; j++) {
                if (arr[j] < arr[j + 1]) {     //关于升顺和降序 则只用改这里的符号 （只要后一个比前一个大就交换）
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
                System.out.println("第" + i + "轮的" + "第" + j + "步的排序结果为");
                showArr(arr);
            }
            System.out.println("\033[31;4m" + "第" + i + "轮排序结果:" + "\033[0m");
            showArr(arr);
        }
    }

    /**
     * 打印整个数组
     *
     * @param arr
     */
    private static void showArr(int[] arr) {
        //增强for循环打印
        for (int a : arr) {
            System.out.print(a + "\t");
        }
        System.out.println();

    }
```

- 排序前:总共有**9**个数据
- 9 5 6 8 2 7 3 4 1
- 第0轮排序结果： 第1轮排序共执行**8**步
- 5 6 8 2 7 3 4 1 **9**
- 第1轮排序结果：第2轮排序共执行**7**步
- 5 6 2 7 3 4 1 **8 9**
  第2轮排序结果：第3轮排序共执行**6**步
- 5 2 6 3 4 1 **7 8 9**
- 第3轮排序结果：第4轮排序共执行**5**步

- 2 5 3 4 1 **6 7 8 9**
- 第4轮排序结果：第5轮排序共执行**4**步
- 2 3 4 1 **5 6 7 8 9**
- 第5轮排序结果：第6轮排序共执行**3**步
- 2 3 1 **4 5 6 7 8 9**
- 第6轮排序结果：第7轮排序共执行**2**步
- 2 1 **3 4 5 6 7 8 9**
- 第7轮排序结果：第8轮排序共执行**1**步
- 1 **2 3 4 5 6 7 8 9**
- 排序后
- 1 **2 3 4 5 6 7 8 9**

特点：每进行一轮排序，就会少**比较一次**，因为每进行一轮排序都会找出一个最大值（最小值）。

如上例：**第一趟比较之后**，排在最后的一个数**一定是最大**的一个数，

第二趟排序的时候，只需要比较**除了最后一个数**以外的**其他的数**，同样也能找出一个**最大的数排在参与第二趟比较的数后面**，

第三趟比较的时候，只需要比**较除了最后两个数以外的其他的数**，以此类推.....

这里面的temp可以这样理解：

假设你有**两个**杯子 一个**红色**杯子里面装的是**java**一个**绿色**杯子装的是**纸杯蛋糕**

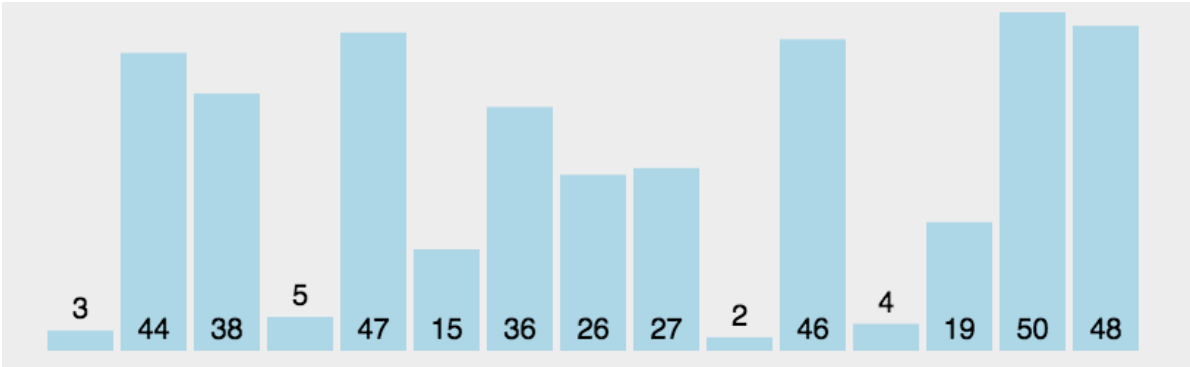
你想要实现 **绿色**杯子里面装**java 红色**杯子里面装**纸杯蛋糕** 你不能直接就互相倒吧，所以我们需要第三个杯子临时存放一下就是这个**temp**

简单选择排序

基本思想：每一趟从待排序的数据元素中选择**最小**（或最大）的一个元素**作为首元素**，**直到所有元素排完为止。**

算法实现：每一趟**通过不断地比较交换来使得首元素为当前最小**，交换是一个比较耗时间的操作，我们可以通过设置一个值来记录较小元素的下标，循环结束后存储的就是当前最小元素的下标，这时再进行交换就可以了。

对于数组一个无序数组{4，6，8，5，9}来说，我们以min来记录较小元素的小标，i和j结合来遍历数组，初始的时候min和i都指向数组的首元素，j指向下一个元素，j开始从右向左进行遍历数组元素，若有元素比min元素更小则进行交换，然后min为更小元素的小标，i再向右走，这样循环到i走到最后一个元素就完成了排序，过程如下图所示：




```
    int[] arr = new int[]{5, 3, 6, 2, 10, 2, 1};
         System.out.println("初始状态");
         showArr(arr);
         selectSort(arr);
```

```
    }

    /**
     * 这个函数的实现主要有两坨
     *
     * @param arr
     */
    public static void selectSort(int[] arr) {
        for (int i = 0; i < arr.length - 1; i++) {


            /**
             * 这一坨代码主要用来找最小值
             */
            int minIndex = i; // 用来记录最小值的索引位置，默认值为i
            for (int j = i + 1; j < arr.length; j++) {
                if (arr[j] < arr[minIndex]) {
                    minIndex = j; // 遍历 i+1~length 的值，找到其中最小值的位置
                }
            }
            /**
             * 默认最小值和真正最小值 不一致时交换
             */
            // 交换当前索引 i 和最小值索引 minIndex 两处的值
            if (i != minIndex) {
                int temp = arr[i];
                arr[i] = arr[minIndex];
                arr[minIndex] = temp;
            }
            System.out.println("\033[31;4m" + "第" + i + "轮排序结果:" +
"\033[0m");
            showArr(arr);
            // 执行完一次循环，当前索引 i 处的值为最小值，直到循环结束即可完成排序
        }
    }

    //打印方法
    private static void showArr(int[] arr) {
        //增强for循环打印
        for (int a : arr) {
            System.out.print(a + "\t");
        }
        System.out.println();


    }
```

关于其它的排序算法 有兴趣的同学可以自己去研究 [十大排序算法总结]

最后

很高兴看到这么多的同学们加入红岩网校，红岩网校的学习进度相对于学校来说确实很快，但是你只要更上我们的步伐，你写的每一次作业，你敲的每一次代码都不会白做，它们都会为你以后的生活和学习带来惊喜。万事开头难，对于一开始来说你们也许会遇到很多的困难，但是大家不要那么容易就放弃了，希望大家都能在红岩网校收获到自己想要的！