# (一) Java基础语法

## (1)环境搭建（myeclipse16及jdk的安装）

|  |
| --- |
| 步骤1. |

## (2) Java概述

|  |
| --- |
| /\* Java是什么?  Java是一个跨平台的编程语言，Java是一个纯软件平台。  \*/  /\* Java有什么?  JavaSE Java标准版，定位在桌面应用程序，同时是其余两个平台的基础。  JavaME Java缩微版，定位在移动设备电子产品，例如Android系统的应用和游戏。  JavaEE Java企业版，定位在大中型企业级的应用。  \*/  /\* JDK是Java开发工具包，用于开发Java程序。  JRE是Java运行时环境，用于运行Java程序，JRE是平台有关的。  JDK与Java的关系及区别?  http://www.cnblogs.com/myitm/archive/2011/05/03/2035942.html  \*/    // Java应用程序的入口(即程序开始执行的地方)  // 未注释的地方即为源代码（以下为例）  **package** com;  **public** **class** Test {  **public** **static** **void** main(String[] args) {  System.***out***.println("Hello world");  }  } |

## (3) Java的八种基本数据类型

|  |
| --- |
| **public static void** main(String[] args) {  /\* 整数类型的有4种，按照存储范围由小到大依次为：byte、short、int、long,  java 整数默认情况下使用int类型(定义的变量没有超出int的范围，编译器自动向下转化，超出,编译器直接报错，亦可以进行强制转化)  \*/  (1)byte字节类型，占8位，取值范围-128~127（2的7次方，第一位是符号位）  **byte** b = 1;  System.*out*.println("byte字节类型的值是：" + b);  // (2)short短整型，占16位，取值范围-32768~32767（2的15次方）  **short** s = 10;  System.*out*.println("short短整型的值是：" + s);  // (3)int整型，占32位（2的31次方）  **int** i = 100;  System.*out*.println("int整型的值是：" + i);  // (4)long长整型，占64位（2的63次方）  **long** g = 1000;  System.*out*.println("long长整型的值是：" + g);  /\* 浮点数类型有2种，按照存储范围由小到大依次为：float、double,  Java浮点数默认使用double类型  \*/  // (5)float单精度浮点数，占32位，实际值比int大  // 注意：float的数值需要添加F或者f指定是float类型  **float** f = 1.5F;  System.*out*.println("float单精度浮点数的值是：" + f);  // (6)double双精度浮点数，占64位，实际值比long大  // 注意：D或者d表示double类型  **double** d = 1.23D;  System.*out*.println("double双精度浮点数的值是：" + d);  /\* (7)char字符类型，单引号，只能存储单个数字、单个字母、单个汉子、占16位  \*/  **char** c = '你';  System.*out*.println("char字符类型的值是：" + c);  // (8)boolean布尔类型，只有true或者false两个值  **boolean** boo = **true**;  boo = **false**;  System.*out*.println("boolean布尔类型的值是：" + boo);  } |

## (4)变量与常量

|  |
| --- |
| **public static void** main(String[] args) {  // Java的八种基本数据类型的变量可以理解为是一个容器，可以改变的量。  **int** a = 2;  **int** b = 3;  //让两个变量中的值互换  **int** t = a;  a = b;  b = t;  System.*out*.println(a);  System.*out*.println(b);  // 常量使用关键字final定义，表示恒定的量，不能被修改。  /\* 在方法中定义的变量叫做局部变量，局部变量必须要声明而且初始化之后才能使用。若final int a = 2;a=9;则编译器报错  \*/  } |

## (5)基本数据类型之间的转换

|  |
| --- |
| **public static void** main(String[] args) {  /\*(1)同类型的转换，低精度赋值给高精度没问题，高精度赋值给低精度的需要强制类型转换\*/  **int** i = 10;  **short** s = (**short**) i;  System.*out*.println(s);  // (2)不同类型的转换，浮点数转换为整数的时候需要注意强制类型转换  **double** d = 1.34;  **long** g = (**long**) d;  System.*out*.println(g);  **double** dd = 10L;  // (3)int和char之间的转换  **int** v = 20320;  **char** c = (**char**) v;  System.*out*.println(c);  } |

## (6) 运算符

### (6.1)赋值运算符

|  |
| --- |
| // (1)赋值运算符  int a = 2;  // a = a + 3;  a += 3;  System.*out*.println(a);  **short** s = 10;  /\* 当数据类型不一致进行计算时，Java会先将低精度的数值往高精度转换，然后再计算。\*/  // s = (short)(s + a);  s += a;  System.*out*.println(s); |

### (6.2)算数运算符

|  |
| --- |
| //算数运算符  System.*out*.println(3 + 2);  System.*out*.println(3 - 2);  System.*out*.println(3 \* 2);  System.*out*.println(3.0 / 2);  System.*out*.println(3 % 2); //取余数 |

### (6.3)比较运算符

|  |
| --- |
| // 比较运算符，结果一定是boolean类型的，true或者false  System.*out*.println(2 > 3);  System.*out*.println(2 >= 3);  System.*out*.println(2 < 3);  System.*out*.println(2 <= 3);  System.*out*.println(2 == 3);  System.*out*.println(2 != 3); |

### (6.4)逻辑运算符

|  |
| --- |
| // 逻辑运算符，true和false的运算  // 逻辑非，取反  // System.out.println(!true);  // 逻辑与，两边都为true结果为true，其余情况都为false  // System.out.println(true && false);  // System.out.println(false && true);  // System.out.println(true && true);  // System.out.println(false && false);  // 逻辑或，两边有一边为true结果为true，其余情况都为false  System.*out*.println(**true** || **false**);  System.*out*.println(**false** || **true**);  System.*out*.println(**true** || **true**);  System.*out*.println(**false** || **false**); |

### (6.5)自增自减运算符

|  |
| --- |
| // 自增自减运算符  **int** i = 3;  // ++i表示先自增1，然后再参与计算，--i同理  // i++表示先参与计算，然后再自增1，i--同理  System.*out*.println((3 == i++)); |

### (6.6)三目运算符

|  |
| --- |
| // 三目运算符?:如果a>b，则将a赋值给max否则将b赋值给max  **int** a = 2;  **int** b = 3;  **int** max = (a > b) ? a : b;  System.*out*.println(max);  // 例子：使用三目运算符找出3个变量中的最大值  **int** v1 = 2;  **int** v2 = 3;  **int** v3 = 5;  **int** m = (v1 > v2) ? v1 : v2;  m = (m> v3) ? m : v3;  System.*out*.println(m); |

## (7)流程控制语句

### (7.1)if分支语句

|  |
| --- |
| // (1)if单分支语句，特点是语句块有可能不执行  **if** (**true**) {  System.*out*.println("if单分支语句");  }  // (2)if两个分支语句，至少执行一个分支  **if** (**false**) {  System.*out*.println("if两个分支语句true");  } **else** {  System.*out*.println("if两个分支语句false");  }  // 例子：求两个变量中的最大值  **int** a = 2;  **int** b = 5;  **if** (a > b) {  System.*out*.println(a);  } **else** {  System.*out*.println(b);  }  // (3)if多个分支语句，最多只能执行一个分支，而且只能执行一个分支  **if** (**true**) {  System.*out*.println("1");  } **elseif** (**true**) {  System.*out*.println("2");  } **else** {  System.*out*.println("3");  }  // 例子：三个变量中找最小值  **int** x = 2;  **int** y = 5;  **int** z = 9;  **if** (x < y && x < z) {  System.*out*.println(x);  } **elseif** (y < x && y < z) {  System.*out*.println(y);  } **else** {  System.*out*.println(z);  } |

### (7.2)switch分支语句

|  |
| --- |
| /\* jdk1.7之前switch分支语句中的类型必须是：byte、short、int、char及  其对应的封装类或者枚举Enum类型,枚举在讲面相对象的时候讲到\*/  // break关键字用于跳出switch语句  **int** i = 85;  **switch** (i / 10) {  **case** 9: {  System.*out*.println("优");  **break**;  }  **case** 8: {  System.*out*.println("良");  **break**;  }  **case** 7: {  System.*out*.println("中");  **break**;  }  **case** 6: {  System.*out*.println("差");  **break**;  }  **default**: {  System.*out*.println("不及格!");  }  }  // jdk1.7之后也支持String类型（String不是基本数据类型，是类类型）  String i = "优秀";  **switch** (i) {  **case** "优秀": {  System.***out***.println("优");  **break**;  }  **case** "良好": {  System.***out***.println("良");  **break**;  }  **default**: {  System.***out***.println("不及格!");  }  } |

### (7.3)for循环语句（简单循环在讲解对象的时候再提）

|  |
| --- |
| /\*for(<初始化>; <[条件表达式](https://www.baidu.com/s?wd=%E6%9D%A1%E4%BB%B6%E8%A1%A8%E8%BE%BE%E5%BC%8F&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d9PjwBujKhuHbsm16zPhcz0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHfLPH6dnjRY)>; <增量>){};执行顺序：  初始化->条件表达式成立->语句块->增量->条件表达式成立->语句块…  若其中条件表达式不成立，则跳出当前循环  \*/  // 例子：求1 + 3 + 5的和  **int** s = 0;  **for** (**int** i = 1; i <= 5; i++) {  **if** (i == 2 || i == 4) {  **continue**;  }  s = s + i;  }  System.***out***.println(s);  /\*循环中的两个关键字：break关键字和continue关键字，  break关键字：跳出当前循环、跳出switch语句。  continue关键字：终止当前这次循环，直接执行下一次循环。  /\*for(;;){}代表无限循环，for(int i=0;;i+=2)也是无限循环，  初始化只进行一次。i+=2表示每次的步数是增加2。  \*/  //无限循环的例子  **int** i = 0;  **for**(;;){  **if**(i>4){  **break**;  }**else**{  i++;  }  }  System.***out***.println(i); |

### (7.4)while与do-while循环语句

|  |
| --- |
| //对于第一次循环,while是先判断再执行,do-while是先执行,再判断。  //如下的while不会打印出任何信息  **int** i = 10;  **while** (i < 10) {  System.*out*.println(i);  i++;  }  //如下的do-while会打印结果  **int** i=10;  **do**{  System.***out***.println(i);  i++;  }**while**(i<10);  //打印结果的是10,i=11 |

## (8)数组

|  |
| --- |
| // Java中的类型有两种：一是基本数据类型。二是类类型。  // 一维数组  // (1)创建一维数组几种定义格式  **int**[] arr1=**new** **int**[10];//指定数组大小，大小0-9  **int**[] arr2={1,2,3,6}; //指定数组内容  **int**[] arr3=**new** **int**[]{1,2,3};//此种效果与上一个相同，会创建两个对象，不能指定int[]大小  // (2)通过下标存取  **int**[] arr = **new** **int**[3];  arr[0] = 1;  arr[1] = 2;  **int** x = arr[0];  System.***out***.println(x);  // (3) 遍历数组,length是数组的属性（此处暂且记住，讲到类的时候会提到）  **int**[] arr = { 1, 2, 3 };  **for** (**int** i = 0; i < arr.length; i++) {  System.***out***.println(arr[i]);  }  // 二维数组(用一维数组的方式来理解二维数组)  // (1) 创建二维数组几种定义格式  **int**[][] arr = { { 1, 2 }, { 3, 4 }, { 5 } };//定义的同时赋初始值  **int**[][] arr1 = **new** **int**[2][];//不规则数组  **int**[][] arr2 = **new** **int**[2][3];//定义的同时分配空间大小  // (2)下标存取  **int**[][] arr1 = **new** **int**[2][];  arr1[0]=**new** **int**[5];// 为第1行分配5列，也可以认为给第一个一维数组装进一个数组  arr1[0][4]=9;//一行五列值是9，也可认为是一维数组的第一个元素内的数组的第5个位置  System.***out***.println(arr1[0][4]);  (3)遍历二维数组  **int**[][] arr = { { 1 }, { 4, 5 }, { 1, 2, 3 } };  **for** (**int** i = 0; i < arr.length; i++) {  **for** (**int** j = 0; j < arr[i].length; j++) {  System.***out***.print(arr[i][j]);  }  System.***out***.println();  } |

## (9)函数

|  |
| --- |
| // 等到下一章节讲到类的时候，就会涉及到函数，在面向过程开发被称为函数，在面向对象中被称为方法  // 函数需要掌握：  // 调用函数：根据定义调用函数。注意：方法名和參數列表必須完全一致。  // 定义函数：根据需求写出函数定义。  // 现阶段的函数定义格式：  // (1)没有返回值的函数方法签名  // public static void 方法名(参数列表) { 方法体 }  // (2)有返回值的函数方法签名，return关键字返回到方法的调用处。  // public static 返回类型方法名(参数列表) { 方法体 return 返回值;}  // 如下例子:  /\*第一次被调用的时候m的值是2，然后返回到调用此函数的地方，即m1(a,b),  \* 第二次被调用m的值是2，然后返回到调用此函数的地方，即m1(m1(a, b), c)的值是2  \*/  **public** **static** **int** m1(**int** a, **int** b) {    **int** m = (a > b) ? b : a;  **return** m;  }  /\*此时a,b,c分别为2,3,5，程序会先调用内部的m1(a,b)，然后得到返回值再调用m1(m1(a,b)的返回值,c),经过两次m1函数的返回值，int m的值是2，然后将m的值返回到调用m2(int a, int b, int c)的地方，即int max的值是2，然后打印出max的值  \*/  **public** **static** **int** m2(**int** a, **int** b, **int** c) {    **int** m = *m1*(*m1*(a, b), c);  **return** m;  }  //此为入口程序  **public** **static** **void** main(String[] arr) {  /\*会调用方法名为m2，参数列表即m2括号中的2,3,5，类型分别为int,int,int,程序会自动寻找符合这样条件的函数，执行完m2函数后的返回值赋给max\*/  **int** max = *m2*(2, 3, 5);  System.***out***.println(max);  } |

# (二)面向对象的分析和设计OOAD

## (1)类与对象

|  |
| --- |
| /\*类：图纸，模板。  类中有类的成员变量和类的成员方法。  成员变量：在类中定义的变量叫做成员变量，用来表示类的属性和状态。  成员方法：在类总定义的方法叫做成员方法，用来表示类的动作和行为。  构造方法：方法名和类同名，没有返回值，也不加void关键字。  构造方法的作用：一是创建对象，二是初始化。  构造方法注意：如果不写构造方法，系统将提供一个默认的无参的构造方法。\*/  **package** com;  **public** **class** Test {  **private** String name = "张三";  **private** **int** age = 30;  **public** Test() {//构造方法，方法体是空的，则只是创建对象  }  **public** Test(**int** a) {//有参数的构造方法，也只是创建对象  }  // 对象：实例，实体。  //static后面会专门讲到  **public** **static** **void** run() {  System.***out***.println("跑步");  }  **public** **static** **void** eat() {  System.***out***.println("吃饭");  }  **public** **static** **void** main(String[] args) {  // 创建对象的语法格式：  // 类的名字 变量名字 = new 关键字类的构造方法();  Test test = **new** Test(4);  System.***out***.println(test.name);  System.***out***.println(test.age);  test.*eat*();  test.*run*();  }  } |

## (2)引用与实例

|  |
| --- |
| /\*Java中的类型有2种：八种基本数据类型，类类型(引用类型)。  含义：创建一个类的对象，返回对象的内存地址存储到变量中，因此类类型的变量存储的是对象的地址，即引用。  Test b = null;声明了一个对象（B的引用），但jvm并没有开辟内存放入一个对象,  所以b内没有引用，暂时理解null就是什么也没有  b = new Test();// 此对象会被java的垃圾回收机制收回GC  b = new Test();  引用与实例的关系：  一个引用可以引用零个或一个对象。  一个对象可以同时被多个变量引用。\*/  **public** **class** Test {    **public** **static** **void** main(String args){  //最后b1,b2,b3都会存有同一个指向new Test()这个对象的引用  Test b1 = **new** Test();  Test b2 = b1;  Test b3 = b1;  }  } |

## (3)值传递和引用传递

|  |
| --- |
| /\*值传递：  方法调用时，实际参数把它的值传递给对应的形式参数，方法接收的是原始值的一个copy，此时内存中存在两个相等的基本类型，即实际参数和形式参数，后面方法中的操作都是对形参这个值的修改，不影响实际参数的值。  引用传递：  也称为传地址。方法调用时，实际参数的引用(地址，而不是参数的值)被传递给方法中相对应的形式参数，方法接收的是原始值的内存地址；  在方法执行中，形参和实参内容相同，指向同一块内存地址，方法执行中对引用的操作将会影响到实际对象。\*  总结：  值传递是8种基本类型值的传递，除了基本类型其他都是类类型，是引用传递，传递的是地址  注意：这里要特殊考虑String，以及Integer、Double等几个基本类型包装类，它们都是immutable类型（即不可变类型），可以认为是和基本数据类型相似，传值操作。  \*/  //值传递，并没有改变原始值  **public** **class** Test {  **int** m;  **public** **static** **void** main(String[] args){  **int** k = 9;  *m1*(9);  System.***out***.println(k);  }  **public** **static** **void** m1(**int** m ) {  m = 4;  }  }  //引用传递，改变引用的实际对象的值  **public** **class** Test {  **public** **static** **void** main(String[] args){  **int**[] arr = { 1, 2, 3 };  *m1*(arr);  System.***out***.println(arr[0]);  }  **public** **static** **void** m1(**int**[] m ) {  m[0] = 4;  }  } |

## (4)成员变量和局部变量

|  |
| --- |
| /\*成员变量：在类中定义的变量，作用域在类中都可见。系统会默认赋值。  局部变量：在方法或语句块中定义的变量，作用域在方法中都可见。必须初始化才能使用。\*/  **public** **class** Test {    **int** age; //成员变量，类初始化时，若不赋值，会默认赋值  **public** **static** **void** main(String[] args) {    **int** k = 0; //局部变量，若此处不赋值，则调用k的地方会报错，编译不会通过  System.***out***.println(k);    }  }  /\*8种基本类型默认赋值： byte、short、int、long默认都是0，float、double是0.0  Boolean：false,char:空字符  类类型默认赋值：null  \*/ |

## (5)关键字

|  |
| --- |
| /\*关键字：被Java语言赋予了特殊含义，用做专门用途的字符串（单词）,java的关键字都是小写的，java共有51个关键字，2个保留字（现有版本没用，以后的版本可能会用）。定义变量时，避免与这53个单词发生冲突。  51个关键字分别为：  ①访问修饰符的关键字(共3个)：  public、private、protected  ②定义类、接口、抽象类和实现接口、继承类的关键字、实例化对象（共6个）：  class、interface、abstract、implements、extends、new  ③包的关键字（共2个）：  import、package  ④数据类型的关键字（共12个）：  byte、char、boolean、short、int 、float、long、double、void、null、true、false  ⑤条件循环（流程控制）（共12个）：  if、else、while、for、switch、case、default、do、break、continue、return、instanceof  ⑥修饰方法、类、属性和变量（共9个）：  static、final、super、this、native、strictfp、synchronized、transient、volatile  ⑦错误的处理（共5个）：  catch、try、finally、throw、throws  ⑧枚举、断言（共2个）：  enum、assert  2个保留字为：  const、goto  ------------------------------------------------------------------  这些关键字有些已经学过，以后一点一点都会接触到。  ------------------------------------------------------------------ |

### (5.1)关键字static

|  |
| --- |
| static 关键字：  ①被static修饰的成员属于类，称为类成员，静态成员，因此通过“类.成员”的方式来调用。  ②不被static修饰的成员属于对象，称为对象成员，非静态成员，因此通过“对象.成员”的方式来调用。  ③静态成员被所有对象共享。\*/  **public** **class** Test {  **static** **int** *a* = 20;    **int** b = 30;    **static** **void** m1() {  System.***out***.println("访问静态方法m1");  }    **void** m2() {  System.***out***.println("访问非静态方法m2");  }    **public** **static** **void** main(String[] args) {    //static int m = 6;//这样定义是错误的，static只能修饰类的成员变量或方法    System.***out***.println(Test.*a*);//类直接访问自身的静态属性  Test.*m1*();//类直接调用自身的静态方法    Test test = **new** Test();  System.***out***.println(test.b);//对象访问非静态成员变量  System.***out***.println(test.*a*);//对象访问静态成员变量也可以，但是有黄线警告，//建议使用类直接访问  test.m2();//对象访问非静态方法    System.***out***.println(test.*a*);  test.*m1*();//对象访问静态方法，有黄线警告，但并不影响程序运行，建议最好使用//Test调用静态方法  //Test.m2();//此处编译报错，因为m2没有被static修饰  //m2();//此处编译报错，静态方法不能访问非静态方法  //b=50;//此处编译报错，静态方法不能访问非静态成员变量  }  }  /\*①类的初始化：创建对象时调用（目前可认为是new的时候），可以用于完成初始化属性值（在构造方法中初始化）、加载其他的类的功能。  ②类的初始化顺序：加载静态-》加载非静态-》调用构造方法（目前可以这么理解，以后会提到父类的加载）  ③静态不能包含非静态的内容：静态早于非静态初始化，所以无法访问。\*/  **public** **class** Test {    // 静态变量  **public** **static** String *staticField* = "静态变量";  //暂时不用理解这个字符串数组，只为证明static是初始化一次  **public** **static** String[] *strArr* = {"1","2"};  // 变量  **public** String field = "变量";  // 静态初始化块：一般在里面放一些需要提前加载又不需要多次加载的东西（目前不需要  // 太明白，一般用于servlet或者框架的启动加载）  **static** {  System.***out***.println(*staticField*);  System.***out***.println("静态初始化块");  m1();  //看控制台结果，静态只初始化一次，非静态创建几个对象就初始化几次  main(*strArr*);  }  // 初始化块：创建类的时候会调用.  //作用:①类似于构造方法的初始化工作。  // ②用于匿名类的初始化，匿名类无构造方法。  {  System.***out***.println(field);  System.***out***.println("初始化块");  m2();  }  // 构造器  **public** Test() {  System.***out***.println("构造器");  }  **public** **static** **void** m1(){  System.***out***.println("静态方法m1");  }  **public** **void** m2(){  System.***out***.println("非静态方法m2");  }  **public** **static** **void** main(String[] args) {  **new** Test();  }  } |

### (5.2)关键字final

|  |
| --- |
| /\*final常量关键字  (1)被final修饰的变量是常量，不能被修改。  (2)被final修饰的方法不能被覆写。  (3)被final修饰的类不能被继承。  (4)final不能用于修饰构造方法  (5) static final一起使用修饰成员变量或方法时，可以理解为“全局常量”，均可通过类名直接访问  注意：被final修饰的变量必须在声明的同时初始化，因为final变量在编译器加载(否则IDE报错)。\*/  **public** **class** Test {  **final** **static** **int** ***b*** = 3;  **public** **static** **void** main(String[] args) {  //final int a = 2;//局与变量也可以定义final但是并没有什么意义  System.***out***.println(***b***);  }  } |

### (5.3)关键字this

|  |
| --- |
| /\*this关键字表示当前类的对象  this的作用：①用于区分成员变量和局部变量。  ②用于构造方法之间的调用，入口一致\*/  **public** **class** Test {  **private** String name;  **private** **int** age;  //使用new Test()会调用这个无参数的构造方法  **public** Test() {  **this**("");//会调用当前对象参数是(String)的构造方法  }  //使用new Test("参数值")会调用这个只有一个String参数的构造方法  **public** Test(String name) {  **this**(name, 0);//会调用当前对象参数是(String，String)的构造方法  System.***out***.println(name);//打印看看是如何调用  }  //使用new Test("参数值","参数值")会调用这个有两个String参数的构造方法  **public** Test(String name, **int** age) {  **this**.name = name;//this代表当前类的对象  **this**.age = age;  System.***out***.println(name+"+++"+age);  }  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }  **public** **int** getAge() {  **return** age;  }  **public** **void** setAge(**int** age) {  **this**.age = age;  }  **public** String toString() {  **return** **this**.name + "\t" + **this**.age;  }  **public** **static** **void** main(String[] args) {  Test p1 = **new** Test();  p1.setName("张三");  p1.setAge(20);  System.***out***.println(p1);    Test p2 = **new** Test("李四");  p2.setAge(30);  System.***out***.println(p2);    Test p3 = **new** Test("王五", 40);  System.***out***.println(p3);  }  } |

### (5.4)关键字null

|  |
| --- |
| /\*①null是关键字，像public、static、final。它是大小写敏感的，你不能将null写成Null或NULL，编译器将不能识别它们然后报错。  ②null既不是对象也不是一种类型，它仅是一种特殊的值，你可以将其赋予任何引用类型，你也可以将null转化成任何类型  ③null可以赋值给引用变量，你不能将null赋给基本类型变量  ④你可以使用==或者!=操作来比较null值，但是不能使用其他算法或者逻辑操作，例如小于或者大于。在Java中null==null将返回true\*/  String str = **null**;  Integer i = **null**;//8种基本类型的封装类可以  //int i = null;//不能将null赋值给8种基本类型  Double d = **null**;  String myStr = (String) **null**;  Integer myI = (Integer) **null**;  Double myD = (Double) **null**;  System.***out***.println(myStr); |

### (5.5)关键字super(后面讲到继承的时候会讲到)

|  |
| --- |
| /\*super概念：super关键字表示父类的引用  super作用：调用父类的构造方法，调用父类的被覆写的方法。  注意：  如果一个类不写构造方法，系统将提供一个默认公共的无参构造方法。  子类的构造方法第一句会默认调用super()  好的习惯：最好每个类中都写一个无参的构造方法。\*/ |

## (6)封装

|  |
| --- |
| /\*Java是一个面向对象的语言，它三个特征  ①封装②继承③多态  封装的理解:①Java将相关的变量和方法封装到类中。  ②Java通过可见性来实现封装。  可见性有4个范围3个关键字:  private关键字：当前类可见。  默认：当前类可见，本包可见。  protected关键字：当前类可见，本包可见，外包子类可见。  public关键字：都可见。  修饰class的可见性要么是public，要么是本包可见\*/ |

## (7)String

|  |
| --- |
| /\*String是Java中最常用的引用类型，Java中的类型有两种：8种基本数据类型、类类型。  String字符串类，它的使用频率比8种基本数据类型还要高。非常重要。\*/  **public** **class** Test {    **public** **static** **void** main(String[] args){  // 字符串的创建  // 第一种方式采用的是"字面值"，每次使用Java会去字符串常量池检查是否有此字符//串，如果有直接引用，没有则创建并放入字符串常量池中。  String s1 = "你好";  /\*每次都会创建新的对象，建议使用上面的方式创建，此种方式创建了两个对象,new与“你好”,若果常量池中有“你好”，则创建一个对象\*/  String s2 = **new** String("你好");  //编译器会自动编译成你好hello,还是两个对象  String s3 = **new** String("你好"+"hello");  // 字符串的连接  String s = "a";  s += "b";  s += "c";  System.***out***.println(s);  // 连接频繁的时候使用StringBuffer  StringBuffer strbuff = **new** StringBuffer();  strbuff.append("a");  strbuff.append("b");  strbuff.append("c");  System.***out***.println(strbuff);  // 字符串的本质是字符数组  String strArr = "123456789";  **for** (**int** i = 0; i < strArr.length(); i++) {  System.***out***.println(strArr.charAt(i));  }  // 要求至少记住String中的11个方法  String str = "ab,c ";  str.length();//字符串长度  //str.charAt(i);//获取字符  str.toLowerCase();//小写  str.toUpperCase();//大写  str.trim();//去除两边空格  str.substring(1, 2);//截取字符串,不包含截止点，即在此不包含2  str.startsWith("a");//以开头  str.endsWith("c");//以结尾  str.equals("abc");//判断值相等  str.indexOf("bc");//获取下标  str.split(",");//以,号开始分割字符串(也可以以别的做分割标识)  }  } |

### (7.1)equals与==的区别

|  |
| --- |
| /\*都是判断相等。  Equals：用于比较两个独立对象的内容是否相同。  ==：①用于基本数据类型的比较。②判断引用是否指向堆内存的同一块地址。  \*/  **public** **static** **void** main(String[] args){  // String str1 = new String("str");  // String str2 = new String("str");  // //false,str1与str2引用的对象的地址不一样，故false  // System.out.println(str1==str2);  // //true,值都是str，所以相等  // System.out.println(str1.equals(str2));    String s1 = **new** String("str");  String s2 = s1;  //true,s1与s2的地址相同  System.***out***.println(s1==s2);  //true,值相同  System.***out***.println(s1.equals(s2));  }  //例子（理解常量池）  **public** **static** **void** main(String[] args){  String s1="Hello";    String s2="Hello";    String s3=**new** String("Hello");    System.***out***.println("s1和s2 引用地址是否相同："+(s1 == s2));  System.***out***.println("s1和s2 值是否相同："+s1.equals(s2));    System.***out***.println("s1和s3 引用地址是否相同："+(s1 == s3));  System.***out***.println("s1和s3 值是否相同："+s1.equals(s3));  } |

### (7.2)String与StringBuilder,StringBuffer的区别

|  |
| --- |
| /\*①String:由于String是被final修饰的，所以它是字符串常量。以下代码打印出的结果是abc1,为什么改变值了？实际上s的值改变的时候s的引用是又创建了一个对象s+1,原来的abc仍然在，只不过现在又增加了一个新的对象abc1\*/  String s = "abc";  s = s+1;  System.***out***.print(s);  /\*②连接频繁的时候使用StringBuffer或者StringBuilder\*/  StringBuffer strbuff = **new** StringBuffer();  strbuff.append("a");  strbuff.append("b");  strbuff.append("c");  System.***out***.println(strbuff);  StringBuilder strBuilder = **new** StringBuilder();  strBuilder.append("a");  strBuilder.append("b");  strBuilder.append("c");  System.***out***.println(strBuilder);  /\*③总结：String：字符串常量，StringBuffer：字符创变量;线程非安全的，StringBuilder：\* 字符创变量;线程安全的，  \* 如果要操作少量的数据用 = String,  \* 单线程操作字符串缓冲区下操作大量数据 = StringBuilder,  \* 多线程操作字符串缓冲区下操作大量数据 = StringBuffer  \* \*/ |

## (8)继承

|  |
| --- |
| /\*(1) java8以前：当定义一个类时，该类会默认继承Object，Java中的单根继承（java8是2014年发布的，至今使用不广泛，2017年oracle发布java9,目前学习java7即可）\*/ |