**3.1、Java简介（了解）**

Java是一门编程语言，Java发展到今天，已经成为了一个真正意义上的语言标准，如果学习过（C、C++、Java）可以发现语法结构是很类似的，但是Java的标准指的是一种作为应用层封装的标准，使用Java可以调用一些底层的操作，例如，今天的Android开发，就是利用了Java调用了Linux内核操作形成的。在2003年的时候，Java成功的应用在了一枚美国的火箭上。

2001年的时候有一份报道：在美国，从事于C++的开发人员的年薪是$6.5，而从事Java开发的人员的年薪是$7W，7 \* 9 = ￥63W（梦想。。。），第一次在国内的报道上看到Java是1998年的时候。

如果要想追溯Java的发展，那么首先需要从1991年的GREEN项目开始说起，这个项目当时是在email特别盛行的时候提出来的，指的是使用email去控制各个家电产品的运行（物联网），最早SUN的工程师打算使用C++进行项目的开发，但是后来考虑到C++的复杂性，所以使用C++开发出了一个新的平台（Java使用的是C++开发的，但是比C++更加的简单） —— OAK（橡树）平台。不过遗憾的是，在与NetScape的SGL竞标的时候不幸落败（等待淘汰）。不过后来SUN的工程师们开始向网景公司学习浏览器技术，推出了HotJava浏览器（HotJava程序设计，**王克宏**），从而向浏览器技术开始发展，于是在1995年的时候正式的将OAK更名为Java（咖啡），但是Java的历史发展，可以归纳为如下的几个阶段：

· 第一阶段（完善期）：**JDK 1.0** ~ **JDK 1.2**；

· 第二阶段（平稳期）：JDK 1.3 ~ JDK 1.4；

· 第三阶段（发展期）：**JDK 1.5** ~ JDK 1.7；

在1995年的时候推出了JDK 1.0、在1998年的时候推出了JDK 1.2（更名为Java 2）、2005年的时候推出了JDK 1.5。

对于Java的制造公司 —— SUN（中文翻译为：太阳公司，斯坦伏大学网络），是一家主要从事于硬件生产的公司，其中SUN最为著名的就是它的小型机（成功的应用案例：应用在Amazon书店上），而最悲催的是 SUN的确是Java的缔造者，但是真正用它赚到钱的是IBM（Websphere Studio）。而且SUN公司在2000年之后的互联网风暴之中就再也没有缓过劲来，于是在2009年的时候被IBM提议收购，不过没有谈成，马上Oracle跟进，最终被Oracle以69亿美金收购。

**面试题：**请你谈一谈，Oracle收购SUN公司有什么用？

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **对比** | **Oracle** | **Microsoft** |
| 1 | 操作系统 | UNIX | Windows |
| 2 | 数据库 | Oracle大型数据库 | SQL Server中小型数据库 |
| 3 | 中间件 | OAS、收购了BEA得到WebLogic | IIS |
| 4 | 编程语言 | PLSQL、收购SUN得到Java | .NET |

而一旦Oracle收购了SUN公司之后，市场上的编程语言的格局，变为三家公司：Microsoft、Oracle、Adobe（FLEX，Flash编程），不过这几年Adobe也比较悲惨，被Microsoft、Oracle、Apple，一起抵制Flash。

那么从Java的开发地位而言，也分为以下的几个方面：

· J2SE（2005年之后更名为JAVA SE）：指的是进行桌面单机程序的开发；

· J2EE（2005年之后更名为JAVA EE）：指的是企业平台开发；

· J2ME（2005年之后更名为JAVA ME）：进行嵌入式开发。

JAVA EE：只要是进行大型的企业项目开发，像银行、电信等服务都会使用此架构；

JAVA ME：Nokia盛行的时候，Java的手机游戏推广的比较好，但是Java ME并没有得到很好的发展，而是后来被Android所取代了，而到Android时代，可以说是真正的将Java最早的嵌入式开发的设想给做出来的，并且有了大量的游戏和软件出现。而Android的出现，正式的标志着移动互联网时代的开启。

但是对于Android开发而言，国内现在可以做手机的人有，但是可以做整体的人很少。

Java语言的本身的特点如下：

1、 Java语言足够简单，相对于其他的任何语言而言，是很容易学的（入门简单）；

2、 Java避免了C/C++之中复杂的指针关系，而使用了更为简单的引用方式来进行内存传递；

3、 Java是为数不多的支持多线程开发的编程语言；

4、 Java提供了自动的垃圾收集机制，可以定期释放出无用的垃圾空间；

5、 Java语言的安全性较高；

6、 Java最大的特点是具备可移植性，即：同一个程序在不同的操作系统上都可以运行。

如果从编程语言的角度而言，应该分为两种：

· 编译型：如果学习过C的同学应该知道，编译之后会形成出一个\*.exe的文件；

· 解释型：像ASP语言那样，直接将代码放到服务器上进行解释。

但是Java本身却属于两种类型的集合，观察图形：

通过如上的图形可以发现，任何一个\*.java程序首先必须经过编译，编译之后会形成一个\*.class的文件（字节码文件），而后在电脑上执行的不是\*.java，而是编译之后的\*.class文件（这个文件可以理解为“加密”的文件），但是解释程序的电脑并不是一台真正意义上的电脑，而是一台由软件和硬件模拟出来的电脑 —— Java虚拟机。

Java虚拟机的最大作用是起到平台的支持上，通过如上的图形可以发现，所有要解释的程序在JVM上执行，但是由不同版本的JVM去匹配不同的操作系统，这样只要JVM的支持不变，程序可以任意的在不同的操作系统上运行。但是这种运行方式很明显没有直接运行在操作系统上性能高，不过随着硬件技术的发展，这些问题几乎可以忽略了。

## 3.2、JDK的安装与配置（重点）

如果要进行Java的程序开发，必须有JDK的支持，JDK指的是Java的开发工具，本次使用的版本是JDK 1.7（不过在实际的开发之中，可能JDK 1.5还是主流使用版本），可以直接登录www.oracle.com（www.sun.com）上进行下载。

**在进行JDK的安装之前，建议关闭本机的病毒防火墙。**

为了日后的方便维护，所以将JDK安装到：d:\java目录下。

安装JDK的过程之中还提示是否安装JRE（Java运行时解释），主要的功能是解释\*.class程序的，此处的安装目的是要更新本机的JRE版本，不过JDK本身是可以解释程序的。

安装完成之后会出现以下的提示信息。

对于Java程序开发而言，主要会使用JDK的两个命令：javac.exe、java.exe。路径：D:\Java\jdk 1.7.0 \_09\bin。但是这些命令由于不属于windows自己的命令，所以要想使用，就需要进行路径配置。

配置步骤：【我的电脑】 🡺 【属性】 🡺 【高级】 🡺 【环境变量】 🡺 【添加新的PATH】，不同的PATH之间使用“;”分隔，修改如下：

**3.3、第一个Java程序：永远的“Hello World !”（重点）**

第一个要测试的程序永远从“Hello World !”开始，Java程序的文件后缀必须是\*.java。

**范例：**定义一个新的文件：**Hello.java**

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  System.out.println("Hello World !") ;  }  } |

当一个\*.java程序编写完成之后，可以按照如下的步骤执行：

· 编译程序，通过命令行进入到程序所在的路径，执行：javac **Hello.java**，形成“Hello.class”（字节码）；

· 解释程序，对生成的Hello.class在JVM上执行，输入：java Hello。

在正常的情况下，本程序已经可以执行了，但是现在发现出现了如下的错误提示：

|  |
| --- |
| Exception in thread "main" java.lang.**UnsupportedClassVersionError**: Hello (Unsupported major.minor version 51.0) |

直接提示“UnsupportedClassVersionError”（不支持的类版本错误），现在编译的时候使用的是JDK 1.7，那么解释的时候也应该JDK 1.7，那么来验证一下当前的JRE版本，输入：java -version，信息如下：

|  |
| --- |
| java version " 1.4.2 \_03"  Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.2 \_03-b02)  Java HotSpot(TM) Client VM (build 1.4.2 \_03-b02, mixed mode) |

发现现在解释程序的版本是JRE 1.4，因为Oracle 10g安装之后，本身就默认提供了JDK，而这个JDK在path中的配置如下：

|  |
| --- |
| D:\oracle\product\ 10.1.0 \db\_1\jre\1.4.2\bin\client;  D:\oracle\product\ 10.1.0 \db\_1\jre\1.4.2\bin; |

那么现在有两种解决方法：

· 方法一：删除掉Oracle的所有JRE配置，太残忍了；

· 方法二：由于path的内容采用的是顺序读取方式，可以将新的配置写在最前面，修改PATH：

**3.4、第一个程序解释（重点）**

第一个程序编写完成之后，下面来对此程序的组成进行说明：

**1、 关于类的定义：**

所有的java程序一定要被类所管理，那么定义类的格式如下：

|  |
| --- |
| [public] class 类名称 {} |

对于类的定义现在就有了两种形式：

· public class定义：类名称必须和文件名称保持一致，在一个\*.java之中只能有一个public class；

· class定义：类名称可以和文件名称不一致，但是生成的是class定义的名称，在一个\*.java程序之中可以同时存在多个class的定义，编译之后会分为不同的\*.class文件；

**额外声明：**

· 在讲课过程之中为了方便理解，所以在一个\*.java程序之中会同时存在public class和class定义的类，而在日后的自己编写的代码过程之中，一个\*.java文件之中基本上都只包含一个public class，不会有其他class单独定义；

· **所有类名称必须有自己的命名规范，每一个单词的开头首字母大写，例如：TestDemo；**

**2、 主方法**

主方法表示的是一个程序起点，要放在一个类之中，主方法定义格式如下：

|  |
| --- |
| public static void main(String args[]) {} |

这些字母的组成是完全固定的，今天先记住，以后进行完整的讲解。

**额外声明：**日后主方法所在的类都将其称为主类，一般主类都使用public class声明。

今天的所有程序都在主类之中编写，而至于类是什么，以后再介绍。

**3、 系统输出**

可以直接在屏幕上显示输出信息，操作语法：

|  |  |
| --- | --- |
| **输出后加换行：** | System.out.println(输出内容) ; |
| **输出后不加换行：** | System.out.print(输出内容) ; |

这个基本的格式是Java程序运行的基本结构，一定要记下来。

**3.5、classpath（重点）**

如果说现在要想执行某一个java程序，那么一定要进入到程序所在的路径下才可以执行，例如：现在程序的路径是在d:\testjava文件夹之中，如果要想执行这个文件夹之中的所有的\*.class文件，则需要进入到此目录下执行，那么如果现在希望在不同的目录下也可以执行呢？那么会直接提示用户，找不到这个类。那么现在非要执行的话，则必须配置CLASSPATH，配置语法如下：

|  |
| --- |
| SET CLASSPATH=\*.class文件所在的路径 |

**范例：**将CLASSPATH配置到d:\testjava目录之中

|  |
| --- |
| SET CLASSPATH=d:\testjava |

此时，再次执行“java Hello”命令，发现程序可以正常的执行完毕。而通过这个演示也可以得出一个结论：**当使用java命令执行一个类的时候，会首先通过CLASSPATH找到指定的路径，而后在此路径下加载所需要的\*.class文件。**

但是，如果像本程序这样，到处乱指CLASSPATH也是不可能的，最好的做法还是从当前所在的路径下加载所需要的\*.class文件比较合适，那么这个时候往往将CLASSPATH设置为“.”。

|  |
| --- |
| SET CLASSPATH=. |

这个“.”也属于默认的配置，之所以要强调“.”的问题主要原因是在于，日后可能有一些其他的程序自动的修改本机的CLASSPATH，那么这个时候只能依靠手工配置，不过以上的配置方式都只是针对于一个命令行完成的，如果要针对于所有的命令行方式完成，则就需要增加一个新的环境属性。

**配置步骤：**【我的电脑】 🡺 【属性】 🡺 【高级】 🡺 【环境变量】 🡺 【新建】 🡺 【输入属性的名称和内容】

**面试题：**请解释PATH和CLASSPATH的区别？

· PATH：是操作系统的环境属性，指的是可以执行命令的程序路径；

· CLASSPATH：是所有\*.class文件的执行路径，java命令执行的时候将利用此路径加载所需要的\*.class文件。

## 3.6、标识符和关键字（重点）

在程序之中用于定义名称的都表示标识符，例如：类的名称、方法名称或变量名称等等，在java之中的标识符的定义格式：由字母、数字、\_、$所组成，其中不能以数字开头，不能是Java中的保留字。

但是在这里面需要提示的是，所有的标识符之中用户不要去使用“$”定义，而且标识符一定要有自身的意义，不要随意起名称，一般都建议使用英文字母组成，例如：studetName，**但是在定义变量（标识符）或方法的时候也有一个明确的要求：第一个单词的首字母小写，之后每个单词的首字母大写，例如：studentName。而在定义类名称的时候（标识符），每一个单词的首字母大写，例如：TestDemo**。

关键字也被称为保留字，指的是一些有特殊含义的内容，在定义标识符的时候不能够去使用，而Java之中的保留字定义如下：

一共有49个关键字，但是严格来讲，这些关键有如下几个说明：

· 未使用到的关键字：goto（无条件跳转）、const（定义常量）；

· 有特殊含义的标记（严格来讲不算关键字）：true、false、null；

· JDK 1.4之后引入的新关键字：assert；

· JDK 1.5之后引入的新关键字：enum。

## 3.7、数据类型（重点）

任何一门语言都是由若干种不同的数据类型所组成，在java之中数据类型一共分为两类：

· 基本数据类型（数值操作）： 默认值

|- 数值型：

|- 整型：byte、short、int、long； 🡺 0

|- 浮点型：float、double； 🡺 0.0

|- 字符型：char； 🡺 '\u0000'

|- 布尔型：boolean； 🡺 false

· 引用数据类型（内存操作）：

|- 数组、类、接口； 🡺 null

今天主要讲解基本类型的数据，而且每种基本数据类型也都有其自己的保存数据范围，这些范围如下。

· byte的数据长度是8位，-128 ~ 127；

· int数据的长度为32位，-2147483648 ~ 2147483647；

· double可以保存的数据范围是最大的。

但是对于以上给出的基本数据类型的定义，如果从实际的开发角度上，以下的几种类型最为常用：

· int型：只要是看见了整型的定义，其类型都是int；

· double型：只要是看见了小数的定义，其类型都是double；

· byte型：日后进行数据传输的时候使用此类型，讲解到IO，和编码转换的操作上使用；

· boolean：用于程序的逻辑操作使用；

· long：表示日期时间、表示文件长度的时候。

### 3.7.1、整型

整型就表示一个基本的整数，可以直接使用int定义，而且在java之中默认的一个整数，其类型就是int。

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  int x = 10 ; // 10是一个整数，就属于int型  int result = x \* 2 ; // int型 \* int型 = int型  System.out.println(result) ;  }  } |

但是，对于以上的操作代码，下面有两点说明：

· 说明一：请保持良好的编程习惯，在每一个操作之中都加上一个“ ”。

· 说明二：所有的变量在使用之前一定要为其赋予默认值。

**范例：**错误的操作

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  int x ; // 按照道理来讲，x是int型，没有赋值，结果应该是0  System.out.println(x) ;  }  } |

因为x变量只定义了而未被初始化，修改程序：

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  int x ; // 按照道理来讲，x是int型，没有赋值，结果应该是0  **x = 10 ; // 在使用之前为x变量赋值**  System.out.println(x) ;  }  } |

但是这种代码的成功编译只能针对于JDK 1.5之上的版本完成，而在JDK 1.4之前，以上的代码是错误的，是不能使用的，所以为了防止这种版本的差异，明确给出要求：**所有的变量一定要在其定义的时候直接赋值**。

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  **int x = 10 ; // 定义变量给出默认值**  System.out.println(x) ;  }  } |

int型数据本身也是有自己的保存范围的，那么如果说现在操作的数据已经超过了其int的范围呢？

为了方便验证，下面给出两个操作，这两个操作日后讲解，可以通过它得出int的最大值和最小值：

· 取得int的最大值：Integer.MAX\_VALUE；

· 取得int的最小值：Integer.MIN\_VALUE。

**范例：**极限操作

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  int max = Integer.MAX\_VALUE ; // 最大值  int min = Integer.MIN\_VALUE ; // 最小值  System.out.println(max) ; // 复制当前行代码：CTRL + J  System.out.println(min) ;  // 最大值 + 1 = 最小值、最小值 - 1 = 最大值  System.out.println(max + 1) ; // int型 + int型 = int型，-2147483648  System.out.println(max + 2) ; // int型 + int型 = int型，-2147483647  System.out.println(min - 1) ; // int型 - int型 = int型，2147483647  }  } |

这个地方就好象出现了一个循环的操作，这种情况在程序上称为数据的溢出，而要想解决这种数据溢出的主要方法就可以通过扩大数据范围来完成，比int大的范围是long，所以现在有如下的两种方式完成：数字L、(long)数字。

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  int max = Integer.MAX\_VALUE ; // 最大值  int min = Integer.MIN\_VALUE ; // 最小值  System.out.println(max) ; // 复制当前行代码：CTRL + J  System.out.println(min) ;  // 最大值 + 1 = 最小值、最小值 - 1 = 最大值  System.out.println(max + 1L ) ; // int型 + long型 = long型，2147483648  System.out.println(min - (long) 1) ; // int型 - long型 = long型，-2147483649  }  } |

所有的程序的操作之中的数据类型转换：

· 如果小范围和大范围操作，小范围自动变为大范围，例如：int + long = long；

· 如果现在把表示范围大的数据变成表示范围小的数据，则必须强制完成，例如：int型 = (int) long型；

当然，下面再通过一个简单的代码进一步说明转性的操作。

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  int x = 10 ; // int型数据  long y = x ; // int变为long型  long result = x \* y ; // int \* long = long ;  int temp = (int) result ; // long --> int，强制转换  System.out.println(temp) ;  }  } |

另外还需要强调的是byte型数据，这个表示的是字节的定义。此类型的范围是：-128 ~ 127之间。

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  int x = 200 ; // 超过了byte范围  byte b = (byte) x ; // 强制转换，溢出数据  System.out.println(b) ;  }  } |

可是，说到byte型数据，额外提醒一下，有一个问题，算是一个特殊支持：

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  byte b = 20 ; // 20是int型  System.out.println(b) ;  }  } |

在为byte类型赋值的时候，如果其给出的数据范围在byte范围之中，则自动的将int变为byte型数据，当然这种操作只是在直接赋值的情况下完成。

### 3.7.2、浮点型

浮点型指的是小数，在java之中，默认的一个小数实际上对应的类型就是double型数据。

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  double x = 10.02 ; // 定义浮点型数据  System.out.println(x \* x) ;  }  } |

这个时候的计算结果为：“100.40039999999999”。按照数学的角度，肯定结果不正确，但是这个结果是由于JVM本身的bug所决定。

而在浮点型数据之中，实际上也分为两种：double、float，float的范围要小于double的范围，那么既然默认的小数属于double型数据，如果要为float赋值呢？必须强制转换，而转换的方式：数字F、(float)数字；

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  float x = 10.02F ; // double --> float  float y = (float) 10.02 ; // double --> float  System.out.println(x \* y) ;  }  } |

但是需要说明的是，整型数据也可以自动的向浮点型转换。

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  int x = 10 ;  double y = x ; // int --> double  System.out.println(y \* x) ; // double \* int = double  }  } |

一般在进行数学计算的时候，需要将一些数据变为浮点型，例如：以下的计算：

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  System.out.println(9 / 2) ; // 4  }  } |

这个时候的结果是4，但是实际上的结果应该是4.5，因为现在的计算都是两个整型数据完成的，所以其计算结果也一定还是int，int不包含小数位，所以小数位如果存在则直接取消，如果要想得出正确的结果，则需要将其中的一个数据变为double或float即可。

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  System.out.println(9 / (double)2) ; // int / double = double  }  } |

以后定义变量的时候，是整型就使用int定义，是小数就使用double进行定义。而且所有数据的转型：

· 自动转型（由小到大）：byte 🡺 short 🡺 int 🡺 long 🡺 float 🡺 double；

· 强制转型（由大到小）：double 🡺 float 🡺 long 🡺 int 🡺 short 🡺 byte。

至于说到底使用到何种的转换，我个人的认为是几乎不要去转换，只需要表示出该有的数据即可。例如，定义年龄肯定是int型数据，定义成绩是double（能使用double就使用double，不要用float）型数据。

### 3.7.3、字符型

在Java之中使用单引好“'”定义的内容就表示一个字符，例如：'A'、'B'，那么定义字符的时候类型使用char完成。

在各个语言之中，char和int之间是可以互相转换的，***在C语言之中转换的编码是ASC II码***，当时的编码范围：

· 大写字母范围：65 ~ 90；

· 小写字母范围：97 ~ 122。

大写字母和小写字母之间查了32。

**范例：**定义字符

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  char c = 'A' ; // 定义一个字符  int x = c ; // char --> int  x += 32 ; // 变为小写编码  char temp = (char) x ; // int --> char  System.out.println(temp) ;  }  } |

但是需要提醒的是，Java在定义字符的时候所使用的并不是ASC II码，而是UNICODE编码，这是一种使用十六进制定义的编码，可以表示出任意的文字，这就包含了中文定义。

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  char c = '张' ; // 定义一个字符，23452  int x = c ; // char --> int  System.out.println(x) ;  }  } |

UNICODE设计的时候包含了所有的ASC码，所以有一部分的编码是完全和ASC II码重复的，但是也有许多是不重复的编码。

### 3.7.4、布尔型

布尔型主要表示的是一种逻辑的判断，其中布尔是一个数学家的名字，而布尔型数据只有两种取值：true、false。

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  boolean flag = true ; // 定义布尔型数据  // if语句判断的就是布尔型数据  if (flag) { // flag == true  System.out.println("Hello World .") ;  }  }  } |

但是需要提醒的是，在许多的语言之中，布尔型也使用0（false）或非0（true）来表示，不过此概念在Java之中无效，只有true和false两个内容。

### 3.7.5、初见String

String本身不属于Java的基本数据类型，因为它属于一个类（引用型数据），但是这个类使用起来可以像基本数据类型那样方便的操作，而且也使用很多，只要是开发都会存在字符串，字符串是使用“"”定义的一串数据。

在字符串之中可以使用“+”进行字符串的连接操作。

**范例：**定义字符串

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  String str = "Hello " ; // 定义字符串  str += " World " ; // 字符串连接  str = str + "!!!" ; // 字符串连接  System.out.println(str) ;  }  } |

但是在使用字符串的过程之中，也需要注意一点，在char类型之中实际上除了定义单个字符之外，也存在了一组转义字符：\\（单个\）、\t（表示按下了tab键）、\n（换行）、\"、\'。

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  String str = "\"Hello World !!!\"\n\tHello MLDN\\" ; // 定义字符串  System.out.println(str) ;  }  } |

在讲课的过程之中，这些符号还是会出现的。

**范例：**观察以下程序的运行结果

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  double x = 10.2 ;  int y = 20 ;  String result = "加法计算结果：" + ( x + y ) ;  System.out.println(result) ;  }  } |

所有的数据类型只要是碰到了String的连接操作（+），那么所有的类型都会先自动向String转型，之后再进行字符串的连接操作。

## 3.8、运算符（重点）

+、-、\*、/都属于运算符，在Java之中运算符太多了。并且这些运算符之间是存在着优先级的，可是正常人，真的不怎么背它。但是在现实之中比较无奈的事，以下的代码会经常出现。

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  int x = 10 ;  int y = 20 ;  int result = ++ x \* y -- / y \* 10 + ++x \* -- y ;  System.out.println(result) ;  }  } |

以上的程序本人绝对不会分析，一般国家的计算机等级考试都这些烂玩意。如果非要有人写，你可以心里把这个人的祖宗十八代一代代的使劲骂，包括这个人可能产生的后代也要狠狠的骂，譬如：孩子没屁眼。

### 3.8.1、三目运算符

三目运算是一种赋值运算符，其语法格式如下：

|  |
| --- |
| 数据类型 变量 = 布尔表达式?满足条件设置的内容:不满足条件设置的内容 ; |

**范例：**将两个int型数据大的数值赋值给变量

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  int x = 10 ;  int y = 20 ;  int result = x > y ? x : y ;  System.out.println(result) ;  }  } |

当然，对于这样的计算，也可以通过if…else完成。

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  int x = 10 ;  int y = 20 ;  int result = 0 ; // 定义变量，保存结果  if (x > y) {  result = x ;  } else {  result = y ;  }  System.out.println(result) ;  }  } |

这样的三目运算，在日后从事的开发之中，一定会出现，最近的出现时间断：4周之后。

### 3.8.2、逻辑运算符

逻辑运算一共包含三种：与（多个条件一起满足）、或（多个条件有一个满足）、非（true变false，false变true）。

**1、 与操作：表示将若干个条件一起进行连接判断，同时满足返回true，有一个不满足返回false，对于与操作有两种运算符：&、&&。**

**范例：**普通与操作，&

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  if (1 == 2 & 10 / 0 == 0) {  System.out.println("条件满足。") ;  }  }  } |

此时程序出现了错误，而这个错误是由“10 / 0 == 0”造成的，那么证明所有的条件都进行了验证，但是与操作的特点是属于 —— 有一个条件不满足，结果就是false，那么如果前面已经存在了不满足条件的运算，后面不管有多少个满足的条件，其结果都是false，那么就真的没有必要进行判断了，这个时候可以换一个符号 —— 短路与；

**范例：**短路与操作，&&

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  if (1 == 2 && 10 / 0 == 0) {  System.out.println("条件满足。") ;  }  }  } |

因为前面的条件（1 == 2）的结果是false，那么后面的就没有必要再继续进行判断了，最终的结果就是false。

2、 或操作：若干个条件一起判断，其中只要有一个返回true，结果就是true，只有都返回false的时候结果才是false，或操作有两种运算：|、||

**范例：**普通或操作，|

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  if (1 == 1 | 10 / 0 == 0) {  System.out.println("条件满足。") ;  }  }  } |

使用普通或操作的过程之中，发现即使前面的条件满足了，后面的也进行正常的判断，但是后面的判断似乎没有任何的意义，因为不管返回是何种结果都不会影响最终的结果就是true。

**范例：**短路或操作，||

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  if (1 == 1 || 10 / 0 == 0) {  System.out.println("条件满足。") ;  }  }  } |

可以发现，前面的条件（1 == 1）满足了就会返回true，那么不管后面是何条件最终的结果都是true。

### 3.8.3、位运算符（了解）

位运算在Java之中存在：&、|、^、~、>>、<<、>>>，但是如果要想进行位运算之前，那么首先必须先知道如何将十进制数据变为二进制数据，原则：数据除2取余，最后倒着排列，例如，下面演示一个操作。

19 🡺 十进制 ， 00000000 00000000 00000000 0010011

÷ 2

9 …… 1 ↑

÷ 2

4 …… 1 ↑

÷ 2

2 …… 0 ↑

÷ 2

1 …… 0 ↑

÷ 2

0 …… 1 ↑

**范例：**观察位与运算

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  int x = 19 ;  int y = 20 ;  System.out.println(x & y) ;  }  } |

19的二进制数据： 00000000 00000000 00000000 0010011

20的二进制数据： 00000000 00000000 00000000 0010100

&的结果： 00000000 00000000 00000000 0010000 🡺 16

**范例：**观察或操作

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  int x = 19 ;  int y = 20 ;  System.out.println(x | y) ;  }  } |

19的二进制数据： 00000000 00000000 00000000 0010011

20的二进制数据： 00000000 00000000 00000000 0010100

|结果： 00000000 00000000 00000000 0010111 🡺 23

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  int x = 19 ;  System.out.println(x >>> 2) ;  }  } |

19的二进制数据： 00000000 00000000 00000000 0010011

向右边移位： 00000000 00000000 00000000 0000100 🡺 4

**面试题：**请问如何可以更快的计算出2的3次方

向左边移位2位。

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  int x = 2 ;  System.out.println(x << 2) ;  }  } |

2到二进制数据： 00000000 00000000 00000000 00000010；

向左边移位： 00000000 00000000 00000000 00001000； 🡺 8

**面试题：**请解释&和&&、|和||的区别？

· 逻辑运算上：

|- &：表示普通与，所有的判断条件都要依次执行；

|- &&：表示短路与，若干个条件，如果前面的条件返回了false，那么后面的不再判断，结果就是false；

|- |：表示普通或，所有的判断条件都要依次执行；

|- ||：表示短路或，若干个条件，如果前面的条件返回了true，后面的不再判断，结果就是true。

· 位运算：&表示位与计算、|表示位或的计算。

除了这两道面试题之外，任何的二进制数据的操作和先阶段没有任何的关系，大概清楚含义即可，不用做特别深入的研究，如果非要研究，可以参考《汇编语言程序设计》一书。

## 3.9、程序结构（重点）

程序结构在语言之中一共分为三种：顺序结构、选择结构、循环结构。

### 3.9.1、顺序结构

所有的代码按照先后的顺序依次进行执行，例如，如下代码：

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  int x = 2 ;  x = x + 2 ;  System.out.println(x) ;  }  } |

### 3.9.2、分支结构（选择结构）

就相当于提供了一些条件判断，根据判断的结果来选择执行何种操作，对于分支结构主要操作语法：if、if..else、if..else if…else，这三种结构的完整语法如下：

|  |  |  |
| --- | --- | --- |
| **if语法：** | **if…else语法：** | **if…else if…else语法：** |
| if (布尔表达式) {  条件满足时执行的程序 ;  } | if (布尔表达式) {  条件满足时执行的程序 ;  } else {  条件不满足时执行的程序 ;  } | if (布尔表达式1) {  条件满足时执行的程序 ;  } else if (布尔表达式2) {  条件满足时执行的程序 ;  } ... else {  所有条件都不满足时执行的程序 ;  } |

**范例：**if语句

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  int age = 16 ;  if (age < 18) {  System.out.println("少儿不宜观看。") ;  }  }  } |

**范例：**if..else语句

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  int age = 26 ;  if (age < 18) {  System.out.println("少儿不宜观看。") ;  } else {  System.out.println("青年观看的教育片。") ;  }  }  } |

**范例：**if…else if…else操作

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  int age = 260 ;  if (age <= 18) {  System.out.println("您属于青少年。") ;  } else if (age > 18 && age <= 50) {  System.out.println("您属于青状年。") ;  } else if (age > 50 && age <250) {  System.out.println("您属于老年。") ;  } else {  System.out.println("妖怪，您不是人。") ;  }  }  } |

但是对于多条件判断使用if..else if…else是可以判断布尔条件的，如果是多数值判断，可以通过switch完成，语法：

|  |
| --- |
| switch (判断值) {  case 数值1:  满足此数值时执行的语句 ;  [break ;]  case 数值2:  满足此数值时执行的语句 ;  [break ;]  case 数值3:  满足此数值时执行的语句 ;  [break ;]  ...  default:  所有条件都不满足时执行的语句 ;  [break ;]  } |

对于switch操作，在最早主要使用的是整型或者是字符来完成。

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  int ch = 0 ;  switch(ch) {  case 0 :  System.out.println("数值是0。") ;  break ;  case 1 :  System.out.println("数值是1。") ;  break ;  case 2 :  System.out.println("数值是2。") ;  break ;  default :  System.out.println("没有条件满足。") ;  break ;  }  }  } |

但是对于switch随着版本的不同，也有所更改，JDK 1.5之后也可以利用枚举作为判断条件，而JDK 1.7之后，switch里面也可以判断字符串了（String），这个可以理解为JDK 1.7才支持的新功能。

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  String str = "two" ;  switch(str) {  case "one" :  System.out.println("壹") ;  break ;  case "two" :  System.out.println("贰") ;  break ;  case "three" :  System.out.println("叁") ;  break ;  default :  System.out.println("没有") ;  break ;  }  }  } |

### 3.9.3、循环结构

循环的概念主要指的是某一块代码可以被重复执行多次。而循环的操作，分为两种语法：while循环、for循环。

**1、 while循环：**

|  |  |
| --- | --- |
| **do…while（98%不会看见使用）：** | **while(布尔表达式)：** |
| do {  循环体 ;  循环条件修改 ;  } while (循环判断) ; | while (循环判断) {  循环体 ;  循环条件修改 ;  } |

通过这样的语法就可以发现，实际上do..while表示先执行后判断，而while循环表示先判断后执行，如果循环条件都不满足的情况下，do..while至少执行一次，而while一次都不会执行。

通过以上给出的两个格式，应该可以发现出循环结构的特点：

· 循环的结束判断；

· 每次循环体执行的时候，循环条件要求修改。

**范例：**使用while两种循环实现1 ~ 100的累加

|  |  |
| --- | --- |
| **do…while：** | **while(布尔表达式)：** |
| public class Hello {  public static void main(String args[]) {  int sum = 0 ; // 保存累加的结果  int x = 1 ; // 进行结束的判断  do {  sum += x ; // 累加  x ++ ; // x自增长  } while (x <= 100) ; // 满足则继续执行  System.out.println(sum) ;  }  } | public class Hello {  public static void main(String args[]) {  int sum = 0 ; // 保存累加的结果  int x = 1 ; // 进行结束的判断  while (x <= 100) {  sum += x ; // 累加  x ++ ; // x自增长  }  System.out.println(sum) ;  }  } |

**2、 for循环：**

for循环的最大特点是已经明确的知道了循环次数，for循环的语法如下：

|  |
| --- |
| for (循环的初始值 ; 循环的判断条件 ; 循环条件的修改) {  循环体 ;  } |

**范例：**实现1 ~ 100的累加

|  |  |
| --- | --- |
| **推荐作法：** | **不推荐作法：** |
| public class Hello {  public static void main(String args[]) {  int sum = 0 ; // 保存累加的结果  for (int x = 1 ; x <= 100 ; x ++) {  sum += x ;  }  System.out.println(sum) ;  }  } | public class Hello {  public static void main(String args[]) {  int sum = 0 ; // 保存累加的结果  **int x = 1 ; // 初始值**  for ( ; x <= 100 ; ) {  sum += x ;  **x ++ ; // 循环条件修改**  }  System.out.println(sum) ;  }  } |

**个人总结，关于循环的出现情况：**

· while循环：在不确定循环次数，但是确定循环结束条件的情况下使用；

· for循环：确定循环次数的情况下使用。

对于循环而言，也可以进行循环的嵌套操作。

**范例：**输出一个乘法口诀表，需要两层循环

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  for (int x = 1 ; x <= 9 ; x ++) { // 控制循环行  for (int y = 1; y <= x ; y ++ ) {  System.out.print(x + "\*" + y + "=" + x \* y + "\t") ;  }  System.out.println() ; // 换行  }  }  } |

**范例：**打印三角型

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  int line = 9 ; // 打印9行  for (int x = 0 ; x < 9 ; x ++) { // 循环次数，控制行  for (int y = 0 ; y < line - x ; y ++) {  System.out.print(" ") ;  }  for (int y = 0 ; y <= x ; y ++) {  System.out.print("\* ") ;  }  System.out.println() ;  }  }  } |

循环操作的基本功能就是重复执行一段代码，但是以上的循环操作，只是作为学习过程之中的程序，循环基本的逻辑结构，实际之中意义不大。

## 3.10、方法（重点）

如果说之前的所有语法和各个语言类似，那么现在的方法就稍微特殊一些了。

### 3.10.1、方法的基本定义

方法在很多地方又被称为函数（Java之中的英文单词是Method，而其他语言之中的英文单词是Function），方法是一段可以被重复调用代码块。**但是需要说明的是，今天所讲解的方法，本身是在主类之中定义的，并且由主方法调用的**。所以方法的定义格式为：

|  |
| --- |
| **public static** 返回值类型 方法名称 (参数列表) {  [return [返回值] ;]  } |

对于返回值类型有两种：

· void：表示此方法没有返回值；

· 数据类型：基本类型和引用类型。

**范例：**定义一个无参的无返回值的方法

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  printInfo() ; // 主方法之中直接调用  printInfo() ; // 主方法之中直接调用  printInfo() ; // 主方法之中直接调用  }  public static void printInfo() { // 方法名称  System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*") ;  System.out.println("\* Hello World \*") ;  System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*") ;  }  } |

方法名称的命名要求|：第一个单词的首字母小写，之后每个单词的首字母大写，例如：printInfo()。

**范例：**定义一个有参的，无返回值方法，例如：将之前打印三角形程序定义唯一个方法，每次只需要传入打印的行即可。

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  printInfo(3) ; // 主方法之中直接调用  printInfo(5) ; // 主方法之中直接调用  printInfo(9) ; // 主方法之中直接调用  }  public static void printInfo(int line) { // 方法名称  for (int x = 0 ; x < line ; x ++) { // 循环次数，控制行  for (int y = 0 ; y < line - x ; y ++) {  System.out.print(" ") ;  }  for (int y = 0 ; y <= x ; y ++) {  System.out.print("\* ") ;  }  System.out.println() ;  }  }  } |

方法负责完成某一特定功能，而后用户就可以根据方法定义的格式进行方法的调用。

**范例：**定义一个有参有返回值的方法

· 定义一个方法，用于判断一个数字是奇数还是偶数。很明显，这个方法的返回值类型应该定义为boolean比较合适，而且如果一个方法上返回的是boolean型数据，则这个方法的名称应该以**isXxx()**的形式命名。

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  if (isType(3)) {  System.out.println("偶数") ;  } else {  System.out.println("奇数") ;  }  }  // true表示是偶数，false表示为奇数  public static boolean isType(int num) {  return num % 2 == 0;  }  } |

讲解到方法的时候需要额外强调一点，如果一个方法使用了void声明，理论上此方法不能够返回数据，但是这个方法却可以通过return结束调用（即：return之后的程序不再执行）。

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  fun(10) ;  fun(30) ;  }  public static void fun(int num) {  if (num == 10) {  return ; // 结束方法调用  }  System.out.println("数值：" + num) ;  }  } |

而这一结束的操作和循环控制的break与continue是一样的。

· break：表示的是退出整个循环；

· continue：表示的是退出一次循环；

|  |  |
| --- | --- |
| **break控制：** | **continue控制：** |
| public class Hello {  public static void main(String args[]) {  for (int x = 0 ; x < 10 ; x ++) {  if (x == 3) {  break ;  }  System.out.println(x) ;  }  }  } | public class Hello {  public static void main(String args[]) {  for (int x = 0 ; x < 10 ; x ++) {  if (x == 3) {  continue ;  }  System.out.println(x) ;  }  }  } |

这三种操作都离不开if语句判断，只要是结束的操作都肯定需要使用if判断。

### 3.10.2、方法的重载（重点）

在讲解重载操作之前，首先完成这样的一种定义，要求定义方法，此方法可以完成两个整数的相加，或者是两个小数的相加，或者是三个整数的相加，那么如果按照习惯性的思路，现在一定要定义三个方法，可能方法就编写如下了：

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  System.out.println("两个整型相加：" + add1(10,20)) ;  System.out.println("三个整型相加：" + add2(10,20,30)) ;  System.out.println("两个浮点型相加：" + add3(10.2,20.3)) ;  }  public static int add1(int x,int y) {  return x + y ;  }  public static int add2(int x,int y,int z) {  return x + y + z ;  }  public static double add3(double x,double y) {  return x + y ;  }  } |

现在已经成功的完成了代码的调用，但是如果按照这种思路，现在有1000种这样的增加方法，这个时候调用起来就很麻烦，还要首先区分出方法的编号是多少，这种不合适的操作一定不是我们所使用的。

在这种情况下可以使用方法的重载（Overloading）来解决问题，**方法重载指的是方法名称相同，参数的类型或个数不同，调用的时候将会按照传递的参数类型和个数完成不同的方法体的执行。**

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  System.out.println("两个整型相加：" + add(10,20)) ;  System.out.println("三个整型相加：" + add(10,20,30)) ;  System.out.println("两个浮点型相加：" + add(10.2,20.3)) ;  }  public static int add(int x,int y) {  return x + y ;  }  public static **int** add(int x,int y,int z) {  return x + y + z ;  }  public static **double** add(double x,double y) {  return x + y ;  }  } |

它是自动的根据参数的类型和个数的不同调用不同的方法体进行执行。

但是讨论一下，以下的代码属于方法重载吗？

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  System.out.println("两个整型相加：" + add(10,20)) ;  System.out.println("两个浮点型相加：" + add(10,20)) ;  }  public static **int** add(int x,int y) {  return x + y ;  }  public static **double** add(int x,int y) { // 返回值不同  return x + y ;  }  } |

可以发现，这个时候除了方法的返回值类型不一样之外，方法的参数类型及个数完全相同，所以这种操作不符合于方法重载的定义。

**额外提醒：**

方法重载的时候并没有规定出返回值类型必须统一，即：重载的方法返回值类型可以不一样，但是从开发的角度而言，建议所有方法重载之后返回值类型统一，但是这种规则不是死的，至少在我们自己编写的代码之中，80%的情况下可以满足。

**范例：**继续观察如下代码

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  System.out.println("Hello World") ; // 输出String  System.out.println(100) ; // 输出int  System.out.println(3000.9) ; // 输出double  System.out.println('A') ; // 输出char  System.out.println(true) ; // 输出boolean  }  } |

所以，可以得出一个结论：System.out.println()实际上就是进行了方法的重载。

### 3.10.3、递归调用（了解）

递归操作指的是方法自己调用自己的形式，但是在进行递归操作的时候必须满足如下的几个条件：

· 必须有结束条件；

· 每次调用的时候都需要改变传递的参数。

**范例：**完成一个1 ~ 100的累加

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  int sum = 0 ;  int x = 1 ;  while( x <= 100 ) {  sum += x ;  x ++ ;  }  System.out.println(sum) ;  }  } |

几乎所有的while循环都可以改变为递归操作。

**范例：**递归操作

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  System.out.println(add(100)) ;  }  public static int add(int num) {  if (num == 1) { // 结束条件  return 1 ; // 不再向后继续加了  }  return num + add(num - 1) ; // 修改参数内容  }  // 第1次调用：return 100 + add(99)；  // 第2次调用：return 100 + 99 + add(98)；  // 倒数第2次调用：return 100 + 99 + ... + 3 + add(2) ;  // 最后一次调用：return 100 + 99 + ... + 3 + 2 + 1 ;  } |

对于递归操作，现在只要求可以理解这个含义即可，而在实际的工作之中，递归尽量少去使用，因为使用不当，就可能造成内存溢出。

# 4、总结

1、 JDK的安装和配置；

2、 classpath和path的区别；

3、 数据类型的划分及默认值；

4、 程序的三种结构；

5、 switch在JDK 1.7之后可以对字符串数据进行判断了；

6、 方法以及方法重载的操作。

# 5、预习任务

面向对象的概念、类与对象、构造方法、private封装、匿名对象、数组、对象数组。

# 6、作业

1、 要求计算出：1! + 2! + … + 60!的结果。

2、 编写一个方法，此方法可以将一个整数变为二进制输出（提示：二进制的计算方法：数字除2取余，倒着取余数。）。

3、 有5个人坐在一起，问第5个人多少岁？答：比第4个人大2岁，问第四个人多少岁的时候，比第三个人大2岁，问第三个人多少岁的时候，比第二个人大2岁，问第一个人的时候，第一个人说自己是8岁。