Java核心技术 卷一

2018年12月15日　　9:23

第一章：

- 即时编译：将执行最频繁的字节码序列翻译成机器码
- 可移植性：
  - Java中，数据类型具有固定的大小，比如int永远为32位；
  - 二进制数据以固定的格式进行存储和运输，消除了字节顺序的困扰；
  - 字符串使用标准的Unicode格式存储的
  - 可移植的接口，接口统一，在不同平台有不同实现，对用户透明
- 动态性：java动态加载class文件，且加载方式不限

第二章

- 版本名称解释：Java SE 8u31-java SE 8第31次更新。u-代表小bug修复更新

第三章：

- 基本数据类型8种，4整型，2浮点类型，1种Unicode编码的字符单元的字符类型char，1种真值类型boolean
  - 整型：（十六进制前缀0x或0X，八进制前缀0，二进制前缀0b或0B，可以位数字字面量加下划线易读：1_000(编译器会去掉)）
    - Int（4字节）
    - Long（8字节）：数值后缀L或l
    - Shot（2字节）
    - byte（1字节）
  - 浮点类型：
    - Float（4字节）：后缀F或f，
    - double（8字节）：没有默认为double
    - 溢出和出错情况：
      - 正无穷大Double.POSITIVE_INFINITY或Float.POSITIVE_INFINITY，正整数除以0
      - 负无穷大Double.NEGATIVE_INFINITY或Float.NEGATIVE_INFINITY
      - NaN(不是一个数字），任意两个NaN不相等。可以用Double.isNaN(x)检测是否为NaN
  - char类型：使用UTF-16编码表示Unicode码点的代码单元，字符不等于char
    - 可以表示为十六进制，前缀\u，范围\u0000~\uffff
    - 转义序列有\u和特殊字符（比如\b退格，\t制表符）。两种都可以出现在引号内表示char类型，但只有\u可以出现在引号外表示char类型。比如'\u2122',"Hello\n", main(String\u005B\u005D args)(\u005B表示[,\u005D表示])
    - Unicode转义序列会在解析代码前处理
    - Unicode和char类型：
      - 码点：指一个与编码表中某个字符对于的代码值
      - Unicode中码点使用十六进制书写，前缀为U+，分为17个代码级别。
        - 基本的多语言级别：U+0000~U+FFFF，包括经典的Unicode代码
        - 其余16个级别：U+10000~U+10FFFF，包括一些辅助字符
      - UTF-16中使用不同长度的编码表示Unicode码点，基本的多语言级别每个字符用16位标识，称为代码单元，其中有U+D800~U+DBFF和U+DC00~U+DFFF为空闲，称为替代区域，各有1024个；辅助字符采用一对连续的代码单元编码，分别使用第一个替代区和第二个替代区代表第一个和第二个代码单元，这样就有1024*1024个表示了。
      - 所以java中字符有的使用一个代码单元表示，则只占2个字节,只为一个char，有的使用两个代码单元表示，占4个字节，表示为两个char。
  - boolean类型：
    - 整型和布尔值之间不能相互转化，if(1) if(x=12) 将编译错误
- 变量：
  - 变量名：字母开头+字母或数字，这里的字母包括'a~z,A~Z,_,$'或在某种语言中表示字母的任何Unicode字符，可以使用Character类的isJavaIdentifierStart和isJavaIdentifierPart方法来检查
- 运算符：
  - 整数被0除会产生编译异常，浮点数被0除会得到无穷大或者NaN
  - strictfp修饰的方法或类必须使用严格的浮点计算来生成可再生的结果，即必须采用IEEE745标准。这是因为比如一个double类型的计算，有些处理器始终使用64位存储数值，而有的处理器使用80位的浮点寄存器，这样增加了中间值的精度，但是结果就不一致了。因此，默认情况时可以使用80位寄存器扩展，但当使用strictfp修饰时，要将80位寄存器截断为64位，保持严格统一
  - java中一般除法/和取余%，若-1为被除数，2为除数，则结果为0，余数为-1。若使用Math.floorDir和Math.floorMod则可以改变为结果为-1，余数为1
  - boolean运算符&&和||是短路运算，即若前表达式满足跳出条件，则跳出。位运算符&和|不是短路运算，会将两个表达式都计算。（有时候蛮有用的）
  - 移位运算符包括>>,<<,>>>，其中>>是算术移位符，使用高位填充，>>>是逻辑移位符，使用0填充。此外要注意右操作数会先进行模32的运算，而当左操作数部位long时进行模64运算
  - 运算符优先级：

运算符	结合性
[] . () (方法调用)	从左向右
! ~ ++ -- +(一元运算) -(一元运算)	从右向左
* / %	从左向右
+ -	从左向右
<< >> >>>	从左向右
< <= > >= instanceof	从左向右
== !=	从左向右
&	从左向右
^	从左向右
|	从左向右
&&	从左向右
||	从左向右
?:	从右向左
=	从右向左

- 字符串：
  - 字串连接：String.join("分隔符","目标字符串"…)
  - 不可变字符串：字符串用于不可修改，“改变”字符串实际上是引用另外一个字符串。实际上Java在通过字面量（还有“+”连接字面量）定义String时，是直接在一个公共池里新建一个字符串，然后引用（这是当原先没有时，若原先有了则直接引用）。这样达到字符串共享的效果。但当通过new String()方式定义时，则是在堆里创建一个String对象（jdk8里String对象主要是一个char[]和hashvalue组成），然后将变量指向这个新的堆里的对象。此外，当使用String.substring、toString之类的方法生成的字符串，以及“+”连接字符串变量（比如String s1 = "abc",String s2 = s1+"cd")这些时候都是在堆里产生新的String对象。所以当使用“==”要注意，只有指向同一个地方时才能正确。可以参考这些例子：https://blog.csdn.net/lubiaopan/article/details/4776000
  - 由char部分可知，java里有些辅助字符是双代码单元的，是使用一个长度为2的char数组表示的，而string.length()返回的是代码单元的数量，string.charAt(index)也是返回为index的代码单元，如果遇到一个双代码单元的字符s，s.charAt(1)将不会返回空，而是s的第二个代码单元。想要获取码点数量可以使用string.codePointCount(0,String.length()).获取第几个string.codePointAt(index)(此时若是普通字符，则返回对应码点的int值，若是辅助字符的高位，则返回整个辅助字符的完全码点的int值，若是辅助字符的低位，则返回辅助字符低位的int值）。简单遍历字符串码点，可以先将string.codePoints.toArray()转化为int数组，反之再将码点数组转

化为字符串new String(codePoints,0,codePoints.length)。所以推荐不要使用char，也不要使用charAt之类的方法。可以参考博客：
https://blog.csdn.net/gjb724332682/article/details/51324036

- String.intern(x)方法，判断在String Pool中是否存在equal x的字符串，若存在则返回该字符串的引用，否则先在池里创建一个Sting对象，再返回引用。这个过程是在1.7之前，他不管调用者是一个new出来的在堆上的对象还是在String Pool里面的对象。但是在jdk1.7之后，若调用者是一个new出来的，即存在于堆上的String对象，则若不存在则不会在String Pool中新建一个String对象，而是新建一个指向堆上的String对象的引用，再返回。所以有这种特殊情况：

```
String str1 = new String("1")+new String("1");//此时池里有1，但是没有11，堆上有11
str1.intern();//在1.7前会直接在池里新建一个11；在1.7之后会新建一个指向堆上11的引用
String str2 = "11";
str1 ?== str2 //这里是否相等就要看jkd版本了。可以参考博客：https://blog.csdn.net/seu_calvin/article/details/52291082
```

- 输入与输出：
  - 输入：
    - Scanner in = new Scanner(System.in);//输入可见
    - Console cons = System.console() ;char[] cons.readPassword("Password:")// 输入不可见
    - Scanner in= new Scanner(File对象)
  - 输出：
    - System.out
    - PrintWriter out = new PrintWriter(stringPath,stringCharset)//可以像System.out一样使用print,println,printf命令
    - 格式化输出：

表 3-5　用于 printf 的转换符

转换符	类　型	举　例	转换符	类　型	举　例
d	十进制整数	159	s	字符串	Hello
x	十六进制整数	9f	c	字符	H
o	八进制整数	237	b	布尔	True
f	定点浮点数	15.9	h	散列码	42628b2
e	指数浮点数	1.59e+01	tx 或 Tx	日期时间（T强制大写）	已经过时，应当改为使用java.time类，参见卷Ⅱ第6章
g	通用浮点数	—	%	百分号	%
a	十六进制浮点数	0x1.fccdp3	n	与平台有关的行分隔符	—

    - %s转化符格式化任意的对象，若对象实现了Formattable接口，将调用formatTo方法，否则调用toString方法。

可以使用多个标志，例如，“%,(.2f”使用分组的分隔符并将负数括在括号内。

表 3-6　用于 printf 的标志

标　志	目　的	举　例
+	打印正数和负数的符号	+3333.33
空格	在正数之前添加空格	| 3333.33|
0	数字前面补 0	003333.33
-	左对齐	|3333.33 |
(	将负数括在括号内	(3333.33)
,	添加分组分隔符	3,333.33
#（对于 f 格式）	包含小数点	3,333.
#（对于 x 或 0 格式）	添加前缀 0x 或 0	0xcafe
$	给定被格式化的参数索引。例如，%1$d，%1$x 将以十进制和十六进制格式打印第 1 个参数	159 9F
<	格式化前面说明的数值。例如，%d%<x 以十进制和十六进制打印同一个数值	159 9F

    - $标志的参数索引值是从1开始的

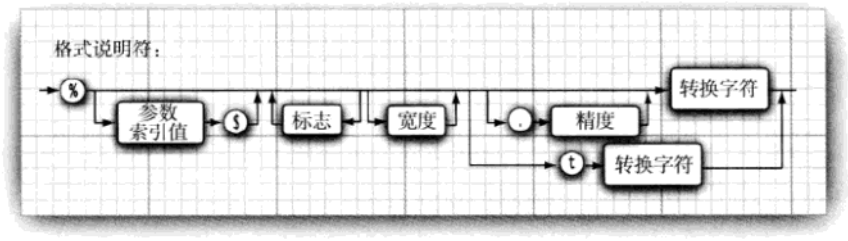

图 3-6　格式说明符语法

- 控制流程：
  - 标签+:，可以通过break+标签，跳转到标签点。实现goto