Java学习精要

# 前言

本文章主要面向具有C/C++基础而想要快速掌握Java的人。主要研究Java的语言特性。其实通过后边的学习，你会发现Java与C++相比很简单。可以认为是一个C++的子集。本文先学习Java的语言特性，至于Java语言的庞大的类库的学习，会在后边的一系列文档中列出。主要是这些类库的实现代码的剖析。最后学习Java的优秀框架，例如Spring，以及RPC框架的源码，在此，RPC框架就学习腾讯开源的Tars吧。在此过程中，尽量去学习和了解JVM和Java编译器的知识。这对于深入学习Java是非常重要的。由于Java在图形用户界面上的使用非常一般，因此，这一系列的文章不会包含AWT和Swing相关的图形界面编程，而是集中在网络编程、IO、多线程、集合框架、数据库操作、JVM这些基础知识上。下边就先从Java基本的语言特性开始学习。

# 1、Java基本概念

Java与C/C++最为不同的地方在于Java是解释执行的，即：java的编译器将Java的源文件编译成字节码之后由Java虚拟机解释执行，正因为有了Java虚拟机，Java才能够实现跨平台，即：Java虚拟机的平台相关性实现了Java的平台无关性。

Java中的任何元素（方法、变量等）必须属于一个类，main方法也一样。这点与C/C++是不同的。

Java与C/C++中的数据不一样，Java中没有无符号数，而且每种类型也是平台无关的，不像C/C++，每种整型占用多少个字节与CPU体系结构以及操作系统平台有关。Java中每种整型的大小都是确定的（不用问，java虚拟机在实现的时候针对不同的体系结构用了很多条件编译才能做到这一点）。而且java中的整型有四种：byte、short、int和long，注意，char型不是整型，在java中char占用两个字节（范围为：\u0000~\uffff），表示一个代码单元。为了明白char类型，必须理解Unicode编码和java中的代码点和代码单元。

Unicode打破了传统字符编码方法的限制。在Unicode出现之前，已经有许多种不同的标准：美国的ASCII、西欧语言中的ISO 8859-1、俄国的KOI-8、中国的GB118030和BIG-5等等，这样就产生了下面两个问题：一个是对于任意给定的代码值（一个整数），在不同的编码方案下有可能对应不同的字母；二是采用大字符集的语言其编码长度有可能不同。例如，有些常用的字符采用单字节编码，而另一些字符则需要两个或更多个字节（最为典型的为UTF8）。

设计Unicode编码的目的就是要解决这些问题。在20世纪80年代开始启动设计工作时，人们认为两个字节的代码宽度足以能够对世界上各种语言的所有字符进行编码，并有足够的空间留给未来的扩展。在1991年发布了Unicode 1.0，当时仅占用65 536个代码值中不到一半的部分。在设计Java时决定采用16位的Unicode字符集，这样会比使用8位字符集的程序设计语言有很大的改进。十分遗憾，经过一段时间，不可避免的事情发生了。Unicode字符超过了65 536个，其主要原因是增加了大量的汉语、日语和韩国语言中的表意文字。现在，16位的char类型已经不能满足描述所有Unicode字符的需要了。

下面利用一些专用术语解释一下Java语言解决这个问题的基本方法。从JDK 5.0开始，代码点（code point）是指与一个编码表中的某个字符对应的代码值（一个整数）。在Unicode标准中，代码点采用十六进制书写，并加上前缀U+，例如U+0041就是字母A的代码点。Unicode代码点可以分成17个代码级别（code plane）。第一个代码级别称为基本的多语言组别（basic multilingual plane），代码点从U+0000到U+FFFF，其中包栝了经典的Unicode代码。其余的16个附加级别，代码点从U+10000到U+10FFFF，其中包括了一些辅助字符（supplementarycharacter）。

UTF-16编码采用不同长度的编码表示所有Unicode代码点。在基本的多语言级别中，每个字符用16位表示，被称为代码单元（code unit）；而辅助字符采用对连续的代码单元进行编码。在Java中，char类型用UTF-16编码描述一个代码单元。强烈建议不要在程序中使用char类型，除非确实需要对UTF-16代码单元进行操作。最好将需要处理的字符串用抽象数据类型表示（有关这方面的内容将在稍后讨论）。 Java字符串（String）由char序列组成。从前面已经看到，字符数据类型是一个采用UTF-16编码表示Unicode代码点的代码单元。大多数的常用Unicode字符使用一个代码单元就可以表示，而辅助字符需要一对代码单元表示。length方法将返回采用UTF-16编码表示的给定字符串所需要的代码单元数量。例如：

        Stringgreeting = "Hello";

**int** n = greeting.length();// is 5

要想得到实际的长度，即代码点数量，可以调用：

**int** cpCount =greeting.codePointCount(0, greeting.length());

调用s.charAt(n)将返回位置n的代码单元，n介于0~s.length()-1之间。例如：

**char** first =greeting.charAt(0); // first is 'H'

**char** last =greeting.charAt(4); // last is 'o'

要想得到第i个代码点，应该使用下列语句

**int** index =greeting.offsetByCodePoints(0, i);

**int** cp =greeting.codePointAt(index);

为什么会对代码单元如此大惊小怪？请考虑下列语句：

    Ƶ is the set of integers

    使用UTF-16编码表示Ƶ需要两个代码单元。调用**char** ch =sentence.charAt(1);返回的不是空格，而是Ƶ的第二个代码单元。为了避免这种情况的发生，请不要使用char类型。这太低级了。

如果想要遍历一个字符串，并且依次査看每一个代码点，可以使用下列语句：

**int** cp =sentence.codePointAt(i);

**if** (Character.*isSupplementaryCodePoint*(cp))

i += 2;

**else**

 i++;

可以使用下列语句实现回退操作：

        i--;

**if** (Character.*isSupplementaryCodePoint*(cp))

i—

**int** cp =sentence.codePointAt(i);

C++的字符串（string）的基本元素为C++中的char（java中的byte）。

Java还提供了大数值相关的类，BigInteger和BigDecimal实现了任意精度的运算。

Java中的控制语句和C/C++中基本一致，唯一不同之处是：Java中的break可以指定标签，这个功能与C/C++中的goto一样。

在Java中，除了基本类型直接使用外其他类型我们都是通过指向该类型对象的指针来使用其对象实例。基本类型有：byte、short、int、long、float、char。数组也不是基本类型。因此：

int[] a;

只是得到一个指向数组的指针。

int[] a = new int[100]才真正得到一个数组。其他对象也是这样的。

明白了这一点对于理解java的很多现象非常有帮助。

例如：Class1 a; 也可以理解为得到一个指针，要得到一个对象也要new Class1()。

Java中的方法调用都是值传递，这时注意，对于对象，可以理解为它的值就是对象的指针。

# 2、Java的类与对象

在Java中，一定要认识到：一个对象变量并没有实际包含一个对象，而仅仅是对象的一个地址。注意，你可以理解成C++中对象的指针而非引用（引用不可为空，而且不可以赋值）。因此：

Class1 obj; //java

等价于

Class \*obj; //C++

Java中，所有的对象存储在堆中，使用new操作符获得。Java中，当两个对象变量赋值时，得到的为两个对象变量（理解为指针）指向同一个堆对象。要真正拷贝一个对象，必须使用clone方法来完成，C++中通过赋值操作符实现这种深度拷贝。

对象即是类的实例，类就是生成对象时所用的模板。

Java中类的定义与C++相似，只不过可以修饰Java中的类的可见性，而C++中类的可见性都是public的，C++中只是可以修饰类成员的可见性。java中，类的可见性分为两种，public和包内可见。

Java和C++的访问都是基于类的访问权限，意思就是：类的方法可以访问该类的所有对象的私有成员。

Java中对象的构造：

Java中对象的构造会对数据域进行初始化，其流程如下：

1）所有数据域被初始化为默认值；

2）按照在类申明中出现的次序，依次执行所有域初始化语句和初始化块；

3）如果构造函数第一行调用了第二个构造函数，则执行第二个构造函数；

4）执行这个构造函数。

有必要说一下域初始化语句为：

class A

{

private int i = 0;

........

}

C++中不允许这种初始化方式。

对象初始化块与以上类似，只不过将多个初始化语句放在{}中：

class A

{

private int i ;

private int j;

//object initialization block

{

i = 0;

j = 1;

}

........

}

C++中也不允许这种初始化方式。

java中也可以调用同一个类的另一个构造器：

class A

{

public A(int i)

{

this(“xufubo”, i); //调用另一个构造函数A，其参数为：(String, int)

.......

}

}

C++中不可以调用同类的另一个构造函数，可以调用父类的构造函数。

Java中也可以调用父类的构造函数。方法如下：super(.....);

Java中调用父类的其他函数：super.funcName(......);

C++中的调用父类构造函数形式为：Child::Child(......) : Father(......) {}

C++中调用父类其他函数为：Father::funcName(......);

与对象初始化块想对应的为静态初始化块，静态初始化块与对象的初始化没有关系，静态初始化块中的语句只能对类的静态域进行初始化。静态初始化块在Java虚拟机的类装载器装载类的时候执行，这时与对象构造没有任何关系。例如：

public class A

{

static

{

System.out.println(“hello world”);

}

}

将会打印hello world。然后提示“main is not defined”错误，之后退出。

# 3、继承、接口、内部类

# 4、Java的修饰符总结

Java中的修饰符包括访问控制修饰符和非访问控制修饰符，用于修饰类、数据域、方法。详解如下：

**一、类的修饰符分为：访问控制符和非访问控制符两种。**

访问控制符：public。

非访问控制符：abstract、final。

1）公共类修饰符public：Java语言中类的可访问控制符只有一个：public即公共的。每个Java程序的**主类**都**必须**是public类，作为公共工具供其它类和程序使用的应定义为public类。

2）抽象类修饰符abstract：凡是用abstract修饰符修饰的类，被称为抽象类。所谓抽象类是指这种类没有具体**对象**的一种概念类。这样的类就是Java语言的abstract类。

3）最终类修饰符 final：当一个类不可能有子类时可用修饰符final把它说明为最终类。被定义为final的类通常是一些**有固定作用**、**用来完成某种标准功能**的类。

4）类缺省访问控制符：如果一个类没有访问控制符，说明它具有缺省的访问控制符特性。此时，这个类只能被同一个包中的类访问或引用。这一访问特性又称为包访问性。

**二、域的控制修饰符也分为：可访问控制符和非访问控制符两类。**

访问控制符有4种：公共访问控制符：public；私有访问控制符：private；保护访问控制符：protected；访问控制符（不加修饰）：default。

非访问控制符有4种：静态域修饰符：static；最终域修饰符：final；易失 ( 共享 ) 域修饰符：volatile；暂时性域修饰符：transient。

1、公共访问控制符public：用public修饰的域称为公共域。如果公共域属于一个公共类，则可以被所有其它类所直接引用。由于public修饰符会降低运行的安全性和数据的封装性，所以一般应减少public域的使用。

2、私有访问控制符private：用private修饰的成员变量 ( 域 ) 只能被该类自身所访问，而不能被任何其它类（包括子类）所引用。

3、保护访问控制符protected：用protected修饰的成员变量可以被三种类所引用：①该类自身；②与它在同一个包中的其它类（包含子类）；③在其它包中的该类的子类。使用修饰符protected的主要作用是允许其它**包**中它的子类来访问父类的特定属性。

4、默认访问控制符：默认情况下（不写访问控制符）成员变量可以被该类本身或与该类同包的类访问和引用，**注意，不可以被位于不同包中的子类访问**。

5、静态域修饰符static：用static修饰的成员变量仅属于类的变量，而不属于任何一个具体的对象，静态成员变量的值是保存在类的内存区域的公共 存储单元，而不是保存在某一个对象的内存区间。任何一个类的对象访问它时取到的都是相同的数据；任何一个类的对象修改它时，也都是对同一个内存单元进行操作。

6、最终域修饰符final：最终域修饰符final是用来定义符号常量的。一个类的域 ( 成员变量 )如果被修饰符final说明，则它的取值在程序的整个执行过程中都是不变的。  
7、易失（共享）域修饰符volatile：易失 ( 共享 ) 域修饰符 volatile是用来说明这个成员变量可能被几个线程所控制和修改。也就是说在程序运行过程中，这个成员变量有可能被其它的程序影响或改变它的取值。因此，在使用中要注意这种成员变量取值的变化。通常volatile用来修饰接受外部输入的域。

8、暂时性域修饰符transient：暂时性域修饰符 transient用来定义一个暂时性变量。其特点是：用修饰符transient限定的暂时性变量，将指定**Java虚拟机**认定该暂时性变量不属于永久状态，以实现不同对象的存档功能。否则，类中所有变量都是对象的永久状态的一部分，存储对象时必须同时保存这些变量。

**三、方法的控制修饰符也分为：可访问控制符和非访问控制符两类。**

可访问控制符有 4 种：公共访问控制符：public；私有访问控制符：private；保护访问控制符：protected；访问控制符（不加修饰）：default。

非访问控制符有 5 种：抽象方法控制符：abstract；静态方法控制符：static；最终方法控制符：final；本地方法控制符：native；同步方法控制符：synchronized。

1）抽象方法控制符abstract：用修饰符abstract修饰的方法称为抽象方法。抽象方法是一种仅有方法头，没有方法体和操作实现的一种方法。

2）静态方法控制符static：用修饰符static修饰的方法称为静态方法。静态方法是属于**整个类**的类方法；而不使用static修饰、限定的方法是属于**某个具体类对象**的方法。由于static方法是属于整个类的，所以它不能操纵和处理属于某个对象的成员变量，而只能处理属于整个类的成员变量，即static方法只能处理static的域。

3）最终方法控制符final：用修饰符final修饰的方法称为最终方法。最终方法是功能和内部语句不能更改的方法，即最终方法不能重载。这样，就固定了这个方法所具有的功能和操作，防止当前类的子类对父类关键方法的错误定义，保证了程序的安全性和正确性。所有被private修饰符限定为私有的方法，以及所有包含在final类 ( 最终类 ) 中的方法，都被认为是最终方法。

4）本地方法控制符native：用修饰符native修饰的方法称为本地方法。为了提高程序的运行速度，需要用其它的高级语言书写程序的方法体，那么该方法可定义为本地方法用修饰符native来修饰。

5）同步方法控制符synchronized：该修饰符主要用于多线程共存的程序中的协调和同步。

6）对于各种访问控制修饰符，与域的情况一致。

总结一下访问控制修饰符：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **访问控制模式** | **在同一个类中** | **在同一个包中** | **在不同包的子类中** | **在所有类** |
| public（公共模式） | Y | Y | Y | Y |
| protected（保护模式） | Y | Y | Y |  |
| default（默认模式） | Y | Y |  |  |
| private（私有模式） | Y |  |  |  |

**Y表示可以访问。注意:对于default模式，非同一个包的子类中也不可访问哈。**

其他：

**static修饰内部类：**