xinklabi

C++与Java比较(转)

博客分类：
- C/C++

C++C#CJava嵌入式

“作为一名C++程序员，我们早已掌握了面向对象程序设计的基本概念，而且Java的语法无疑是非常熟悉的。事实上，Java本来就是从C++衍生出来的。”

然而，C++和Java之间仍存在一些显著的差异。可以这样说，这些差异代表着技术的极大进步。一旦我们弄清楚了这些差异，就会理解为什么说Java是一种优秀的程序设计语言。本附录将引导大家认识用于区分Java和C++的一些重要特征。

(1) 最大的障碍在于速度：解释过的Java要比C的执行速度慢上约20倍。无论什么都不能阻止Java语言进行编译。写作本书的时候，刚刚出现了一些准实时编译器，它们能显著加快速度。当然，我们完全有理由认为会出现适用于更多流行平台的纯固有编译器，但假若没有那些编译器，由于速度的限制，必须有些问题是Java不能解决的。

(2) 和C++一样，Java也提供了两种类型的注释。

(3) 所有东西都必须置入一个类。不存在全局函数或者全局数据。如果想获得与全局函数等价的功能，可考虑将static方法和static数据置入一个类里。注意没有象结构、枚举或者联合这一类的东西，一切只有“类”（Class）！

(4) 所有方法都是在类的主体定义的。所以用C++的眼光看，似乎所有函数都已嵌入，但实情并非如何（嵌入的问题在后面讲述）。

(5) 在Java中，类定义采取几乎和C++一样的形式。但没有标志结束的分号。没有class foo这种形式的类声明，只有类定义。

```
class aType()
void aMethod() {/* 方法主体*/}
}
```

(6) Java中没有作用域范围运算符“::”。Java利用点号做所有的事情，但可以不用考虑它，因为只能在一个类里定义元素。即使那些方法定义，也必须在一个类的内部，所以根本没有必要指定作用域的范围。我们注意到的一项差异是对static方法的调用：使用ClassName.methodName()。除此以外，package（包）的名字是用点号建立的，并能用import关键字实现C++的“#include”的一部分功能。例如下面这个语句：

import java.awt.*;

（#include并不直接映射成import，但在使用时有类似的感觉。）

(7) 与C++类似，Java含有一系列“主类型”（Primitive type），以实现更有效率的访问。在Java中，这些类型包括boolean，char，byte，short，int，long，float以及double。所有主类型的大小都是固有的，且与具体的机器无关（考虑到移植的问题）。这肯定会对性能造成一定的影响，具体取决于不同的机器。对类型的检查和要求在Java里变得更苛刻。例如：

■条件表达式只能是boolean（布尔）类型，不可使用整数。

■必须使用象X+Y这样的一个表达式的结果；不能仅仅用“X+Y”来实现“副作用”。

(8) char（字符）类型使用国际通用的16位Unicode字符集，所以能自动表达大多数国家的字符。

(9) 静态引用的字串会自动转换成String对象。和C及C++不同，没有独立的静态字符数组字串可供使用。

(10) Java增添了三个右移位运算符“>>>”，具有与“逻辑”右移位运算符类似的功用，可在最末尾插入零值。“>>”则会在移位的同时插入符号位（即“算术”移位）。

(11) 尽管表面上类似，但与C++相比，Java数组采用的是一个颇为不同的结构，并具有独特的行为。有一个只读的length成员，通过它可知道数组有多大。而且一旦超过数组边界，运行期检查会自动丢弃一个异常。所有数组都是在内存“堆”里创建的，我们可将一个数组分配给另一个（只是简单地复制数组句柄）。数组标识符属于第一级对象，它的所有方法通常都适用于其他所有对象。

(12) 对于所有不属于主类型的对象，都只能通过new命令创建。和C++不同，Java没有相应的命令可以“在堆栈上”创建不属于主类型的对象。所有主类型都只能在堆栈上创建，同时不使用new命令。所有主要的类都有自己的“封装（器）”类，所以能够通过new创建等价的、以内存“堆”为基础的对象（主类型数组是一个例外：它们可象C++那样通过集合初始化进行分配，或者使用new）。

(13) Java中不必进行提前声明。若想在定义前使用一个类或方法，只需直接使用它即可——编译器会保证使用恰当的定义。所以和在C++中不同，我们不会碰到任何涉及提前引用的问题。

(14) Java没有预处理机。若想使用另一个库里的类，只需使用import命令，并指定库名即可。不存在类似于预处理机的宏。

(15) Java用包代替了命名空间。由于将所有东西都置入一个类，而且由于采用了一种名为“封装”的机制，它能针对类名进行类似于命名空间分解的操作，所以命名的问题不再进入我们的考虑之列。数据包也会在单独一个库名下收集库的组件。我们只需简单地“import”（导入）一个包，剩下的工作会由编译器自动完成。

(16) 被定义成类成员的对象句柄会自动初始化成null。对基本类数据成员的初始化在Java里得到了可靠的保障。若不明确地进行初始化，它们就会得到一个默认值（零或等价的值）。可对它们进行明确的初始化（显式初始化）：要么在类内定义它们，要么在构建器中定义。采用的语法比C++的语法更容易理解，而且对于static和非static成员来说都是固定不变的。我们不必从外部定义static成员的存储方式，这和C++是不同的。

(17) 在Java里，没有象C和C++那样的指针。用new创建一个对象的时候，会获得一个引用（本书一直将其称作“句柄”）。例如：

String s = new String("howdy");

然而，C++引用在创建时必须进行初始化，而且不可重定义到一个不同的位置。但Java引用并不一定局限于创建时的位置。它们可根据情况任意定义，这便消除了对指针的部分需求。在C和C++里大量采用指针的另一个原因是为了能指向任意一个内存位置（这同时会使它们变得不安全，也是Java不提供这一支持的原因）。指针通常被看作在基本变量数组中四处移动的一种有效手段。Java允许我们以更安全的形式达到相同的目标。解决指针问题的终极方法是“固有方法”（已在附录A讨论）。将指针传递给方法时，通常不会带来太大的问题，因为此时没有全局函数，只有类。而且我们可传递对对象的引用。Java语言最开始声称自己“完全不采用指针！”但随着许多程序员都质问没有指针如何工作？于是后来又声明“采用受到限制的指针”。大家可自行判

断它是否“真”的是一个指针。但不管在何种情况下，都不存在指针“算术”。

(18) Java提供了与C++类似的“构建器”（Constructor）。如果不自己定义一个，就会获得一个默认构建器。而如果定义了一个非默认的构建器，就不会为我们自动定义默认构建器。这和C++是一样的。注意没有复制构建器，因为所有自变量都是按引用传递的。

(19) Java中没有“破坏器”（Destructor）。变量不存在“作用域”的问题。一个对象的“存在时间”是由对象的存在时间决定的，并非由垃圾收集器决定。有个finalize()方法是每一个类的成员，它在某种程度上类似于C++的“破坏器”。但finalize()是由垃圾收集器调用的，而且只负责释放“资源”（如打开的文件、套接字、端口、URL等等）。如需在一个特定的地点做某样事情，必须创建一个特殊的方法，并调用它，不能依赖finalize()。而在另一方面，C++中的所有对象都会（或者说“应该”）破坏，但并非Java中的所有对象都会被当作“垃圾”收集掉。由于Java不支持破坏器的概念，所以在必要的时候，必须谨慎地创建一个清除方法。而且针对类内的基础类以及成员对象，需要明确调用所有清除方法。

(20) Java具有方法“过载”机制，它的工作原理与C++函数的过载几乎是完全相同的。

(21) Java不支持默认自变量。

(22) Java中没有goto。它采取的无条件跳转机制是“break 标签”或者“continue 标准”，用于跳出当前的多重嵌套循环。

(23) Java采用了一种单根式的分级结构，因此所有对象都是从根类Object统一继承的。而在C++中，我们可在任何地方启动一个新的继承树，所以最后往往看到包含了大量树的“一片森林”。在Java中，我们无论如何都只有一个分级结构。尽管这表面上看似乎造成了限制，但由于我们知道每个对象肯定至少有一个Object接口，所以往往能获得更强大的能力。C++目前似乎是唯一没有强制单根结构的唯一一种OO语言。

(24) Java没有模板或者参数化类型的其他形式。它提供了一系列集合：Vector（向量），Stack（堆栈）以及Hashtable（散列表），用于容纳Object引用。利用这些集合，我们的一系列要求可得到满足。但这些集合并非是为实现象C++“标准模板库”（STL）那样的快速调用而设计的。Java 1.2中的新集合显得更加完整，但仍不具备正宗模板那样的高效率使用手段。

(25) “垃圾收集”意味着在Java中出现内存漏洞的情况会少得多，但也并非完全不可能（若调用一个用于分配存储空间的固有方法，垃圾收集器就不能对其进行跟踪监视）。然而，内存漏洞和资源漏洞多是由于编写不当的finalize()造成的，或是由于在已分配的一个块尾释放一种资源造成的（“破坏器”在此时显得特别方便）。垃圾收集器是在C++基础上的一种极大进步，使许多编程问题消弥于无形之中。但对少数几个垃圾收集器力有不逮的问题，它却是不大适合的。但垃圾收集器的大量优点也使这一处缺点显得微不足道。

(26) Java内建了对多线程的支持。利用一个特殊的Thread类，我们可通过继承创建一个新线程（放弃了run()方法）。若将synchronized（同步）关键字作为方法的一个类型限制符使用，相互排斥现象会在对象这一级发生。在任何给定的时间，只有一个线程能使用一个对象的synchronized方法。在另一方面，一个synchronized方法进入以后，它首先会“锁定”对象，防止其他任何synchronized方法再使用那个对象。只有退出了这个方法，才会将对象“解锁”。在线程之间，我们仍然要负责实现更复杂的同步机制，方法是创建自己的“监视器”类。递归的synchronized方法可以正常运作。若线程的优先等级相同，则时间的“分片”不能得到保证。

(27) 我们不是象C++那样控制声明代码块，而是将访问限定符（public，private和protected）置入每个类成员的定义里。若未规定一个“显式”（明确的）限定符，就会默认为“友好的”（friendly）。这意味着同一个包里的其他元素也可以访问它（相当于它们都成为C++的“friends”——朋友），但不可由包外的任何元素访问。类——以及类内的每个方法——都有一个访问限定符，决定它是否能在文件的外部“可见”。private关键字通常很少在Java中使用，因为与排斥同一个包内其他类的访问相比，“友好的”访问通常更加有用。然而，在多线程的环境中，对private的恰当运用是非常重要的。Java的protected关键字意味着“可由继承者访问，亦可由包内其他元素访问”。注意Java没有与C++的protected关键字等价的元素，后者意味着“只能由继承者访问”（以前可用“private protected”实现这个目的，但这一对关键字的组合已被取消了）。

(28) 嵌套的类。在C++中，对类进行嵌套有助于隐藏名称，并便于代码的组织（但C++的“命

名空间”已使名称的隐藏显得多余）。Java的“封装”或“打包”概念等价于C++的命名空间，所以不再是一个问题。Java 1.1引入了“内部类”的概念，它秘密保持指向外部类的一个句柄——创建内部类对象的时候需要用到。这意味着内部类对象也许能访问外部类对象的成员，毋需任何条件——就好象那些成员直接隶属于内部类对象一样。这样便为回调问题提供了一个更优秀的方案——C++是用指向成员的指针解决的。

(29) 由于存在前面介绍的那种内部类，所以Java里没有指向成员的指针。

(30) Java不存在“嵌入”（inline）方法。Java编译器也许会自行决定嵌入一个方法，但我们对此没有更多的控制权力。在Java中，可为一个方法使用final关键字，从而“建议”进行嵌入操作。然而，嵌入函数对于C++的编译器来说也只是一种建议。

(31) Java中的继承具有与C++相同的效果，但采用的语法不同。Java用extends关键字标志从一个基础类的继承，并用super关键字指出准备在基础类中调用的方法，它与我们当前所在的方法具有相同的名字（然而，Java中的super关键字只允许我们访问父类的方法——亦即分级结构的上一级）。通过在C++中设定基础类的作用域，我们可访问位于分级结构较深处的方法。亦可用super关键字调用基础类构建器。正如早先指出的那样，所有类最终都会从Object里自动继承。和C++不同，不存在明确的构建器初始化列表。但编译器会强迫我们在构建器主体的开头进行全部的基础类初始化，而且不允许我们在主体的后面部分进行这一工作。通过组合运用自动初始化以及来自未初始化对象句柄的异常，成员的初始化可得到有效的保证。

```
public class Foo extends Bar {
 public Foo(String msg) {
 super(msg); // Calls base constructor
 }
 public baz(int i) { // Override
 super.baz(i); // Calls base method
 }
}
```

(32) Java中的继承不会改变基础类成员的保护级别。我们不能在Java中指定public，private或者protected继承，这一点与C++是相同的。此外，在衍生类中的优先方法不能减少对基础类方法的访问。例如，假设一个成员在基础类中属于public，而我们用另一个方法代替了它，那么用于替换的方法也必须属于public（编译器会自动检查）。

(33) Java提供了一个interface关键字，它的作用是创建抽象基础类的一个等价物。在其中填充抽象方法，且没有数据成员。这样一来，对于仅仅设计成一个接口的东西，以及对于用extends关键字在现有功能基础上的扩展，两者之间便产生了一个明显的差异。不值得用abstract关键字产生一种类似的效果，因为我们不能创建属于那个类的一个对象。一个abstract（抽象）类可包含抽象方法（尽管并不要求在它里面包含什么东西），但它也能包含用于具体实现的代码。因此，它被限制成一个单一的继承。通过与接口联合使用，这一方案避免了对类似于C++虚拟基础类那样的一些机制的需要。

为创建可进行“例示”（即创建一个实例）的一个interface（接口）的版本，需使用implements关键字。它的语法类似于继承的语法，如下所示：

```
public interface Face {
 public void smile();
}
public class Baz extends Bar implements Face {
 public void smile( ) {
 System.out.println("a warm smile");
 }
}
```

(34) Java中没有virtual关键字，因为所有非static方法都肯定会用到动态绑定。在Java中，程序员不必自行决定是否使用

文章出处：飞诺网(http://dev.firnow.com/course/3_program/c++/cppjs/20090403/164032.html)

JAVA和C++都是面向对象语言。也就是说，它们都能够实现面向对象思想（封装，继承，多态）。而由于C++为了照顾大量的C语言使用者， 而兼容了C，使得自身仅仅成为了带类的C语言，多多少少影响了其面向对象的彻底性！JAVA则是完全的面向对象语言，它句法更清晰，规模更小，更易学。它是在对多种程序设计语言进行了深入细致研究的基础上，据弃了其他语言的不足之处，从根本上解决了c++的固有缺陷。 Java和c++的相似之处多于不同之处，但两种语言几处主要的不同使得Java更容易学习，并且编程环境更为简单。

我在这里不能完全列出不同之处，仅列出比较显著的区别：

1、指针

JAVA语言让编程者无法找到指针来直接访问内存无指针，并且增添了自动的内存管理功能，从而有效地防止了c / c++语言中指针操作失误，如野指针所造成的系统崩溃。但也不是说JAVA没有指针，虚拟机内部还是使用了指针，只是外人不得使用而已。这有利于Java程序的安全。

2、多重继承

c++支持多重继承，这是c++的一个特征，它允许多父类派生一个类。尽管多重继承功能很强，但使用复杂，而且会引起许多麻烦，编译程序实现它也很不容易。Java不支持多重继承，但允许一个类继承多个接口(extends+implement)，实现了c++多重继承的功能，又避免了c++中的多重继承实现方式带来的诸多不便。

3、数据类型及类

Java是完全面向对象的语言，所有函数和变量部必须是类的一部分。除了基本数据类型之外，其余的都作为类对象，包括数组。对象将数据和方法结合起来，把它们封装在类中，这样每个对象都可实现自己的特点和行为。而c++允许将函数和变量定义为全局的。此外，Java中取消了c / c++中的结构和联合，消除了不必要的麻烦。

4、自动内存管理

Java程序中所有的对象都是用new操作符建立在内存堆栈上，这个操作符类似于c++的new操作符。下面的语句由一个建立了一个类Read的对象，然后调用该对象的work方法： Read r＝new Read()； r.work()；语句Read r＝new Read()；在堆栈结构上建立了一个Read的实例。Java自动进行无用内存回收操作，不需要程序员进行删除。而c++中必须由程序员释放内存资源，增加了程序设计者的负担。Java中当一个对象不被再用到时，无用内存回收器将给它加上标签以示删除。Java里无用内存回收程序是以线程方式在后台运行的，利用空闲时间工作。

5、操作符重载

Java不支持操作符重载。操作符重载被认为是c++的突出特征，在Java中虽然类大体上可以实现这样的功能，但操作符重载的方便性仍然丢失了不少。Java语言不支持操作符重载是为了保持Java语言尽可能简单。

6、预处理功能

Java不支持预处理功能。c / c++在编译过程中都有一个预编译阶段，即众所周知的预处理器。预处理器为开发人员提供了方便，但增加丁编译的复杂性。JAVA虚拟机没有预处理器，但它提供的引入语句(import)与c++预处理器的功能类似。

7、 Java不支持缺省函数参数，而c++支持

在c中，代码组织在函数中，函数可以访问程序的全局变量。c++增加了类，提供了类算法，该算法是与类相连的函数，c++类方法与Java类方法十分相似，然而，由于c++仍然支持c，所以不能阻止c++开发人员使用函数，结果函数和方法混合使用使得程序比较混乱。 Java没有函数，作为一个比c++更纯的面向对象的语言，Java强迫开发人员把所有例行程序包括在类中，事实上，用方法实现例行程序可激励开发人员更好地组织编码。

8、字符串

c和c++不支持字符串变量，在c和c++程序中使用null终止符代表字符串的结束，在Java中字符串是用类对象(String和StringBuffer)来实现的，这些类对象是Java语言的核心，用类对象实现字符串有以下几个优点：

(1)在整个系统中建立字符串和访问字符串元素的方法是一致的；

(2)字符串类是作为Java语言的一部分定义的，而不是作为外加的延伸部分；

(3)Java字符串执行运行时检空，可帮助排除一些运行时发生的错误；

(4)可对字符串用“十”进行连接操作。

9、数组
java引入了真正的数组。不同于c++中利用指针实现的“伪数组”，Java引入了真正的数组，同时将容易造成麻烦的指针从语言中去掉，这将有利于防止在c++程序中常见的因为数组操作越界等指针操作而对系统数据进行非法读写带来的不安全问题。

10、“goto“语句

“可怕”的goto语句是c和c++的“遗物”，它是该语言技术上的合法部分，引用goto语句引起了程序结构的混乱，不易理解，goto语句子要用于无条件转移子程序和多结构分支技术。鉴于以广理由，Java不提供goto语句，它虽然指定goto作为关键字，但不支持它的使用，使程序简洁易读。

11、类型转换

在c和c++中有时出现数据类型的隐含转换，这就涉及了自动强制类转换问题。例如，在c++中可将一浮点值赋予整型变量，并去掉其尾数。Java不支持c++中的自动强制类型转换，如果需要，必须由程序显式进行强制类型转换。

12、异常

JAVA中的异常机制用于捕获例外事件，增强系统容错能力 try{//可能产生例外的代码 }catch(exceptionType name){ //处理 } 其中exceptionType表示异常类型。而C++则没有如此方便的机...