文章编号:1001-9081(2005)01-0073-03

Java 中的组件复用相关技术

邱万彬,张国杰,裘鸿林
(国家数字交换系统工程技术研究中心,河南 郑州 450002)
(qiuwanbin@yahoo.com.cn)

摘　要:组件技术其实是模块概念发展而来的。除了在开发中分解复杂度外,更重要的在于提高代码复用和互操作性。这里主要讨论了 Java 中反射技术与 JNI 技术及其对组件复用的贡献,并例举了相应实例。

关键词:Java;组件复用;反射;Java 本地接口
中图分类号: TP311.11　　**文献标识码**:A

Related techniques of component reuse in Java

QIU Wan-bin, ZHANG Guo-jie, QIU Hong-lin
(National Digital Switching System Engineering & Technological R&D Center, Zhengzhou Henan 450002, China)

Abstract: Component technology is evolved from the conception of module. It can not only reduce the complexity of software development, but also improve the reusability and cooperativity of the code. An introduction of java reflection and JNI, including their contributions to component reuse, as well as some examples, were presented in this paper.

Key words: Java; component reuse; reflection; JNI

0 引言

Java 语言以其平台独立、安全、面向对象等特性,获得业界越来越广泛应用。结合多种基于组件的技术,J2EE 能为企业应用提供灵活而强大的方案。包括 JavaBean、JSP-Servlet、EJB、JMS、JAXP、SAAJ 等。组件技术可以用来业务代码封装。技术框架需要与具体业务进行动态绑定。为了代价最小化,或者由于技术方面的原因,有时还要使用以前开发的 COM 组件。这里从 Java 组件与 COM 组件两类来介绍 Java 中的组件复用,每个方面又分为组件提供服务与组件执行服务两方面来讲述。

1 简易组件

先讨论简单情形。如果我们要使某些业务方法或通用方法能被复用,只要在 Java 组件中开发相应的方法,并对外公布即可。方法访问权限设为 public 比较方便。比如,我们在 EJB 中实现业务方法,在程序各处均可调用。有时我们先开发接口,再让类实现接口,这样比较规范和一致。通常,我们需要知道方法名和参数格式,才知道如何调用。简易组件通常比较关注其功能性和模块化意义,在复用与互操作方面,灵活性相对较低。

2 动态绑定

Java 组件作为动作执行者时,需要把业务代码绑定到 Java 组件。由于业务方法的多样性和可变性,必须使用反射(Reflection)技术来获取和调用业务方法(及属性)。

反射是一种比一般 Java 编程更为底层的技术。反射访问的特殊类型的元数据是 JVM 中类和对象的描述数据。它使我们创建灵活的代码,这些代码可以在运行时动态装配。这样,我们的软件在结构和行为上能适应需求以至技术的变化。

2.1 反射机理

反射机制涉及两部分内容:元层(meta level)和基层(base level)。元层由元对象组成,用于描述类(组件)本身而不是它们的用法,我们称之为内省(introspection)。元层对象由元对象协议 MOP(Meta Object Protocol)来规约。Java 核心类提供了 Reflection API,这就为 Java 类提供了动态内省的工具和手段。基层用于定义应用逻辑,使用元对象来实现[1,2,3]。很多流行语言都提供反射机制,如 C#和 Java。而新的软件工程方法模型驱动构架 MDA(Model Driven Architecture)更是以 OMG 标准 MOF(Meta Object Facility)作为统一元数据的基础。形成层次结构的反射接口(reflective interfaces)定义了通用的操作,用于发现和操纵基于 MOF 元模型的模型。反射机制在编程语言中一般提供如下功能:

(1) 运行时识别对象及其类型;

(2) 运行时创建对象:根据类名,动态创建类实例;

(3) 运行时提取对象元信息:包括类型、是否抽象、构造器、方法、属性、超类、接口、内部类等各方面信息。

具体到 Java 平台,提供了 Java.lang.reflect 包。普通类的元信息抽取机制由 Class 类来承担。我们最关心它的三个方法:

(1) forName:使用当前装载器,装载给定名字的类。

(2) getName:以 String 对象返回类的名字,这在区分对象引用时很有用。

(3) newInstance:调用类的构造函数(如果存在),返回一个类的实例。

收稿日期:2004-06-09;**修订日期**:2004-11-27

作者简介:邱万彬(1975-),男,江西赣州人,软件工程师,硕士研究生,主要研究方向:软件工程、通信系统; 张国杰(1967-),男,河南内乡人,教授,主要研究方向:通信系统; 裘鸿林(1965-),男,江苏盱眙人,副教授,博士研究生,主要研究方向:软件工程.

当然还有很多关于方法、属性等的方法，如 getMethod、getField。使用反射，甚至私有属性和方法都能被访问。

2.2 实例

通过反射机制，可以获取具体类及其方法的元信息，从而在我们的程序中调用组件中的方法。这里以一个实际项目中的组件来作说明。

我们在 3G 项目复合移动通信系统（CMT）的操作维护管理（OAM）子系统的统计后处理功能模块中，实现了一个命令预约子功能，实质上就是计划任务功能。因为具体业务是可能变化的（比如添加功能，或以后升级），则要使计划任务功能与具体业务方法分离。下面这个称之为“命令预约”的组件以定时器驱动，根据用户的预约设置，定时器中，在适当时刻调用了 doCmd 方法，以运行具体业务方法。doCmd 方法中的部分代码如下：

```
//cmdStr 是组件外传递进来的命令字符串，假设为
//"methodName( parameterName) "
//objInst 即用户传进来的 this 对象
//取得方法名和参数名:
String sMethodName = cmdStr. substring( 0,
  cmdStr. indexOf( "(")). trim( );
String sFieldName = cmdStr. substring( cmdStr. indexOf( "(") + 1,
  cmdStr. indexOf( ")")). trim( );
//取得属性:
Field field = objInst. getClass( ). getDeclaredField( sFieldName);
Class par[ ] = new Class[ 1];
par[ 0] = field. getType( );
//获取方法:
Method method =
    objInst. getClass( ). getDeclaredMethod( sMethodName, par);
//方法执行的实参
Object pp[ ] = new Object[ 1];
// 当属性是 private 时，保证可访问
field. setAccessible( true);
pp[ 0] = field. get( objInst);                    //获取参数值
//调用方法，执行业务命令:
method. invoke( objInst, pp);
```

这样，组件就可以动态地调用外部类的方法，可以单独修改业务方法。还能把此组件用于其他如 OA 系统中，达到复用的目的。

3 Java 中调用 COM 组件

3.1 COM 技术

COM 组件技术是微软开发的程序设计标准，也是由模块化思想发展而来的。目标是简化程序设计，提高系统或对象的互操作性。它规定了对象之间如何相互通讯，符合 COM 规范的对象也叫做 COM 对象。COM 对象位于服务器中，服务器分为进程内服务器、本地服务器、远程服务器三种。进程内服务器是与客户应用程序在同一进程内运行，一般以 DLL 或 OCX 文件的形式存在；本地服务器则作为一个单独的进程，在自己的地址空间中运行，一般以 EXE 文件的形式存在；远程服务器是另一台计算机上的一个 DLL 文件或 EXE 文件，它和自动化控制器不在同一台机器上。远程服务器如果是一个 DLL 文件的话，由一个被称为“Surrogate”的代理程序调用它。[4]

Java 不能直接调用 COM 组件的方法，需要借助 JNI（Java Native Method Interface）来完成。通常，Java 调用 COM 发生在以下场合：

（1）标准 Java 不能支持的平台相关特性。如获取系统信息，操作 word、excel 之类。

（2）为了复用以前的工作成果，而这些成果不是 Java 写成的。

（3）实时性要求高的应用中，Java 不能满足要求，需要用低级语言来实现。

当然 COM 组件毕竟通常不是 Java 语言的（如果是就用不着这么麻烦了），这样就会有些限制或缺点：

（1）由于安全机制，Apple 中使用 JNI 有些限制。

（2）Java 指针安全性可能丢失。COM 分配的内存不在 JVM 的管理范围。

（3）Java 代码的移植性有所降低。主要是受限于 COM 的移植性。

3.2 通过 JNI 调用 COM 的原理

COM 服务器（支持 COM 的应用程序）提供 COM 对象使 COM 客户可以访问自己的方法和属性。COM 服务器必须至少包含一个或多个供其他应用程序联接和访问的 IDispatch 接口和 invoke（ ）方法。Java 客户通过 JNI 来与 COM 通信，见图 1。

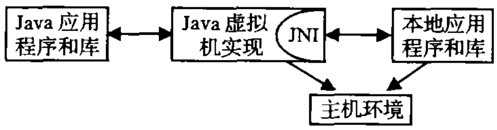

图 1　JNI 的角色

JNI 调用 COM 的过程中大都要进行 Java 变量类型到 COM 类型的转换。JNI 定义了 Java 类型与 COM 类型之间的映射。而 COM 定义了一个类型包装结构 Variant 用于包装所有可能类型。与之对应，实现 JNI 调用的 Java 组件（代理）定义了一个相应的 Jvariant 来包装 Java 的各种数据类型。举个具体例子，Java 中 String 先转换成代理中的 jstring，而你的 C 程序不能直接使用 jstring，还要经过 JNI 转换成 C 里的 string（即 char ＊）。如果我们使用像 3.4 实例中那样的代理组件，我们只要把变量用 Jvariant 来包装就行了。[1,5] 图 2 显示了 Java 与 COM 的交互。

图 2　Java 与 COM 的交互

3.3 编程步骤

下面讲述 Java-JNI-COM 的实现。步骤如下[1]：

（1）编写 Java 代码。创建一个 Java 类，其中声明了本地方法，装载包含 COM 组件的共享库，并调用本地方法。如，类 HelloWorld. Java 包含以下部分代码：

```
public native void displayHelloWorld( );
System. loadLibrary( "hello");
new HelloWorld( ). displayHelloWorld( );
```

（2）编译 Java 代码。

（3）创建 C/C++头文件。使用 Javah 工具，从 Java 类创建 JNI 风格的. h 头文件。头文件为本地方法的实现提供了函数签名。

```
Javah -jni HelloWorld
```

（4）本地方法实现。编写 C/C++ 源文件 HelloWorldImp. c，实现方法。源文件要包含上一步生成的头

文件。

```
#include <jni.h>
#include "HelloWorld.h"
#include <stdio.h>
JNIEXPORT void JNICALL
Java_HelloWorld_displayHelloWorld(JNIEnv *env, jobject obj)
{ printf("Hello world! \n");
  return;
}
```

(5) 创建共享库。用 C 编译器编译(4)生成的源文件为共享库,如成生 DLL 文件。若用 VC 的编译器,命令如:

```
cl -Ic:\java\include -Ic:\java\include\win32 -LD HelloWorldImp.c
-Fehello.dll
```

(6) 运行程序。

3.4 实例

有一个开源项目 JACOB,是一个 Java-COM 桥,它使用了 JNI,能使我们方便地由 Java 调用常见的 COM 自动化组件以获得相应服务,如 MS Word,Excel,PPT 等。这里以一个简单的例子来说明其使用方法。先下载 jacob.jar 包和 jacob.dll。假定已有一个 excel 数据文档,我们调用 excel 自动化服务,使其另存为一个 XML 文档。

```
import java.io.File;
import com.jacob.com.*;
import com.jacob.activeX.*;
public class TestExcel {
  public static void main(String[] args) {
    //输入 excel 文件
    String excelFile = "D:\\test\\excelFile.xls";
    //输出 xml 文件
    String xmlFile = "D:\\test\\xmlFile.xml";
    //启动 excel
    ActiveXComponent app =
        new ActiveXComponent("Excel.Application");

    try {
      Variant f = new Variant(false);
      Variant t = new Variant(true);
      //设置 excel 不可见
      app.setProperty("Visible", f);
      Object workbooks = app.getProperty("Workbooks").
      toDispatch();
      //打开 excel 文件
      Object workbook = Dispatch.invoke(workbooks, "
      Open", Dispatch.Method, new Object[]{excelFile, f,
      t}, new int[1]).toDispatch();
      //以 xml 格式保存到目标文件
      Dispatch.invoke(workbook, " SaveAs ", Dispatch.
      Method, new Object[]{xmlFile, new Variant(46)}, new
      int[1]);
      //关闭
      Dispatch.call(workbook, "Close", f);
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        app.invoke("Quit", new Variant[] {});
    } //finally
  }//main
}//class
```

4 COM 组件回调 Java

如果要在 COM 中回调 Java 代码方法,那就得求助于 JNI 中实现的 JNI 函数了。Java 传递给本地方法有两个必不可少的参数 JNIEnv * jEnv 和 jobject this。通过 jEnv 可以访问 JNI 函数。这些 JNI 函数主要用来取得对 Java 类、属性、方法等的引用,以及取得/设置属性值、调用 Java 方法、异常处理。事实上,通过 JNI 函数,我们可以完成类似于 Reflection 提供的一些功能:创建、检查及更新 Java 对象(包括数组和字串)、装载类、获取类信息、运行期类型检查。[1,5]

下例,Java 类 MyClass 中有属性和方法。在本地 C 实现中可以修改 Java 属性和调用 Java 方法。

```
public class MyClass {
  public void addOne() { aValue +/= 1; }
  public int aValue;
  private native void callBack();
  static {
   System.loadLibrary("CallBackImpl");
  }
  public static void main(String[] args) {
    MyClass app = new MyClass();
    aValue = 2;
    app.callBack();
    System.out.println("Java: " + aValue);
  }
}
```

```
JNIEXPORT void JNICALL
Java_MyClass_callBack(JNIEnv * env, jobject jThis) {
  jclass cls;
  jfieldID fieldId;
  jmethodID methodId;
  int value;
  cls = env->GetObjectClass(jThis);
  fieldId = env->GetFieldID(cls, "aValue", "I");
  methodId = env->GetMethodID(cls, " addOne", "()V");
  value = env->GetIntField(jThis, fieldId);
  printf("Native: %d\n", value);
  env->SetIntField(jThis, fieldId, 5);
  env->CallVoidMethod(jThis, methodId);
                                    //回调 Java 的方法
  value = env->GetIntField(jThis, fieldId);
  printf("Native: %d\n", value);
}
```

5 结语

与 Java 的组件复用(及互操作)相关的技术还有如下等一些技术:网景的"Java 运行期接口"(Java Runtime Interface),微软的"J/Direct"、"本原接口"(Raw Native Interface, RNI)、Java/COM 集成方案,以及更通用的 CORBA 集成技术。随着软件技术尤其近来 MDA 技术的发展及应用,复用及互操作必将在更高的元层次上得到强化。

参考文献:

[1] Sun Microsystems[Z]. The Java Tutorial. 2003.

[2] ECKEL B. Java 编程思想[M]. 侯捷,译. 北京:机械工业出版社,2002.

[3] HORSTMANN CS, *et al.* Core Java 2: Volume I – Fundamentals [M]. Prentice Hall PTR, 2000.

[4] 潘爱民. COM 原理与应用[M]. 北京:清华大学出版社,1999.

[5] HORSTMANN CS, *et al.* Core Java 2: Volume II – Advanced Features[M]. Prentice Hall PTR, 2001.