基于 IOC 的 GUI 框架设计与实现

廖福保，张文梅

（广东农工商职业技术学院 计算机科学系，广东 广州 510507）

摘 要：传统的图形用户界面 GUI（Graphics User Interface）设计中，存在过度耦合、组件与事件之间的映射关系混乱等问题。对此，提出了基于控制反转（IOC）的 GUI 框架，该框架采用 Java 反射机制，解析 xml 配置文件完成组件实例化、组件添加事件监听。实验表明，利用该框架建立的 GUI 实现了业务对象的松散耦合，组件和事件处理方法分离，缩短开发周期，具有较高的可扩展性。

关键词：控制反转；图形用户界面；Java 反射机制

中图分类号：TP311 **文献标识号**：A **文章编号**：1674-7720(2011)10-0012-03

Design and implementation of GUI based on IOC

Liao Fubao, Zhang Wenmei

(Computer Science College, Guangdong AIB Polytechnic College, Guangzhou 510507, China)

Abstract: The traditional GUI is over coupled and the mapping relationship between component and event is confusion. To solve this issue, this paper gives a GUI framework based on IOC. This framework parses xml configuration file, implements component to be instantiated and adds listener for component by using Java reflection mechanism. The practice shows that the GUI by using this framework to build can complement loosely coupled of business object, separate component and events handling method, reduce the cycle of development, increase the scalability.

Key words: inversion of control; graphics user interface; Java reflection mechanism

Java 是目前最优秀的软件开发语言之一，由于其结构简单、面向对象、跨平台等优越特性使它具有极强的生存力，并得到了广泛的应用。基于 Java 的图形用户界面（GUI）中，AWT 是 Java 提供的用来建立和设置 Java 图形用户界面的第一代开发工具。AWT 由 java.awt 包提供，其中包含了许多可以用来建立与平台无关的 GUI 类。由于 AWT 组件占有系统资源较多，常把 java.awt 组件称为重量级组件。Java Swing 是 Java Foundation Classes (JFC)的一部分，解决了 AWT 的很多缺点，相对于 AWT，Swing 是轻量级组件。Swing 提供了许多比 AWT 更好的屏幕显示元素，使用纯 Java 写成，与 Java 一样可以跨平台运行[1]。

图形用户界面（GUI）借助于多种组件，包括菜单、按钮、文本框、选择框、列表框等，通过相应的事件处理机制，实现与用户的动态交互。

1 图形用户界面的建立

1.1 创建 GUI 窗口

javax.swing.JFrame 类是用来建立用户界面的底层窗口容器，能够容纳其他组件的对象，如标签、按钮、文本组件等。JFrame 类提供的 add()方法把不同的组件添加到容器中，通过容器类的 setLayout()方法可以设定容器的布局，安排各种组件在容器中。

使用 JFrame 类创建 GUI 窗口的基本步骤如下：用 JFrame 类或其子类创建一个对象即窗体；设置窗口的部分属性，如标题、宽度、高度、可见性、图标等；添加内容面板、组件；编写事件处理方法；组件添加事件监听。

1.2 Java 事件处理

在 Java 中，程序与用户的交互通过响应各种事件来实现。每当一个事件发生，Java 虚拟机就会将事件的消息传递给程序，由程序中的事件处理方法对事件进行处理。Java 通过委托型事件处理机制来解决对事件的响应。

事件处理机制可表述如下[2]：事件源对象封装了事件源、组件状态等必要信息；当事件源对象发生改变时，向它所注册的所有监听器发出通知，各监听器判断事件类型是否为自己管辖范围，若是，则通知给该监听器的执行器，执行器从事件中获取事件信息，并执行相应函

数,改变组件的状态。

1.3 传统创建窗口和事件处理的局限性

在传统的 GUI 创建过程中,存在一些局限性。

(1)组件创建、添加都采用硬编码方式,造成程序的过度耦合。

(2)如果窗体中有很多组件,组件要添加注册监听,则在代码中看到很多重复注册监听的代码,而这些注册监听的代码都与界面本身设计无关,组件与事件之间的映射关系将会很混乱。

(3)事件处理方法定义在别的类中,无法得到窗体及其组件的引用,只能得到事件源,而无法改变其他组件的状态;或者把事件处理与窗体设计放在一起,这样程序的可维护性又不好。

(4)不利于代码重用,基于 MVC 的思想,应该把事件处理方法分离出来;在需要修改事件处理代码时,就无需修改界面本身的源代码。

2 图形用户界面设计的改进

2.1 控制反转(IOC)

IOC 就是控制反转[3](Inversion of Control)的缩写,也称为依赖注入,控制反转 IOC 是一种用于控制业务对象之间依赖关系的机制,将其设计的类与类之间的关系都交由外部容器进行管理,仅需调用类在容器中注册的名字就可以得到类的实例,有效降低了业务对象之间的依赖程度,实现了业务对象之间的松散耦合。

IOC 的实际意义就是把组件之间的依赖关系(调用关系)反转出来,对象之前的依赖关系用 xml 配置文件描述;这样,各个组件之间就不存在硬编码的关联,任何组件都可以最大程度地得到重用。

考虑如下接口和类的定义:

```
public interface ICar{void operate();}
public class Toyota implements ICar{…}
public class Honda implements ICar{…}
public class Driver{
    private ICar car;
    public void setCar(ICar car){this.car = car;}
    public ICar getCar(){return car;}
    public void drive(){car.operator();}
}
```

类 Driver 依赖于 ICar,而类 Toyota 和 Honda 实现了接口 ICar,即类 Driver 可以依赖于 Toyota 或 Honda。

运用了 IOC 模式后就不再需要自己管理组件之间的依赖关系,只需要声明由 xml 配置文件描述去实现这种依赖关系,就好像把对组件之间的依赖关系的控制进行了倒置,不再由组件自己来建立这种依赖关系而是交给 xml 配置文件去管理。

2.2 设计的改进

在改进的 GUI 编程中,把窗体中组件的创建、组件的外观设置和组件触发事件时执行什么方法,不是以硬编码的方式组合在一起,而是通过配置文件来配置。这样开发人员无须关心组件的创建、组件的样式设置、事件的监听与实现,只需要设置相应的 get、set 方法来存取组件、属性等,事件处理方法能在任意类中实现,方法名可以自定义,并且在其他类中能够得到窗体对象及其组件的引用。当组件的样式发生改变时,只需改动配置文件即可。

该改进设计通过配置文件,并利用控制反转和 Java 反射机制得以实现,这就需要有框架和良好的设计。

3 框架运行机理

框架中各组成部分在运行过程中的调用关系如图 1 所示。

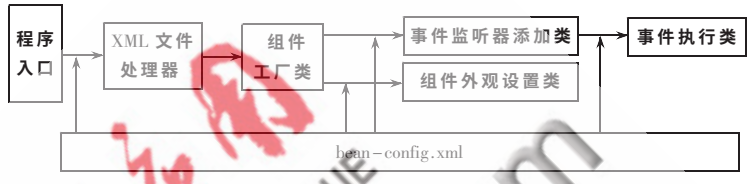

图 1 框架中各组成部分的调用关系

当程序入口启动时,框架解析 bean-config.xml 文件;组件工厂类根据 xml 配置文件创建各种组件对象;组件外观设置类查找 xml 文件为每个组件设置相应的外观;事件监听器类查找 xml 文件为每个组件添加对应的事件监听器;事件执行类查找 xml 文件为每个组件设置事件触发时执行的方法;最后还需要一个保存窗体对象的类。

GUI 程序开发人员只需要设置相应的 get、set 方法来存取组件,事件发生时要执行的方法和配置 xml 文件。组件的建立、外观的设置、事件监听添加、事件处理方法都由框架来完成。一个编码的例子如下:

```
public class JFrameDemo extends JFrame{
    private JTextField input ;
    private JButton ok ;
    //省略的 get, set 方法
    //省略构造方法,该方法用于添加组件到窗体
}
//事件处理类和方法
public class EventOperator{
    public void operate(){
        //从保存窗体对象的类中获得窗体
        //通过窗体的 get 方法获得组件
        //执行所需的操作并修改组件状态
    }
}
```

4 框架的具体实现

4.1 xml 配置文件格式

xml 是一种标记语言,用于各种配置文件和不同语言间交换信息,它只负责信息的存储,而不负责信息的表达。本框架 bean-config.xml 文件的设计格式如下:

```
<?xml version="1.0" encoding="GB2312"?>
<beans>
```

```
    <bean id="input" class="java.awt.JTextField">
        <setColumns>10;Integer</setColumns>
    </bean>
    <bean id="ok" class="java.awt.JButton">
        <setText>计算;String</setText>
        <event type="ActionListener" class="test.Event-
            Operator" method="operate"></event>
    </bean>
    <bean id="frame" class="test.JFrameDemo">
        <ref>input</ref>
        <ref>ok</ref>
    </bean>
</beans>
```

配置文件说明如下：

(1)根节点为 beans。

(2)bean 节点中的 id 属性用来唯一地标识一个组件，该值要与代码里的组件名一致，class 属性用来表示所对应的类名。

(3)event 节点的 type 属性表示监听器的类型，class 属性表示事件触发时将要执行的方法所对应的类名，method 属性表示事件触发时将要执行的方法。如上面 xml 文件中，表示当 ok 组件发生单击事件时，将执行 test. EventOperator 类的 operate 方法。

(4)ref 子节点值表示该组件需要依赖的其他 bean 的标识。

(5)bean 其他子节点为设置组件外观的方法，子节点值为调用该方法所需的参数值和对应的参数类型。

4.2 Java 的反射机制

因为所对应的类、方法都保存在 xml 文件中，而对 xml 解析得到的类名和方法名都是字符串类型，要把字符串实例化成相应的对象并调用就要用到 Java 的反射技术[4]。

Java 的反射机制允许程序在运行时透过 Reflection APIs 取得任何一个已知名称的类的内部信息，包括其访问权限、父类、实现接口，也包括成员变量和方法的所有信息，并可在运行时改变成员变量的内容或执行方法。

本框架主要利用反射机制来实例化对象和调用方法。其关键代码如下(className,methodName 均为字符串)：

```
Class instance = Class.forName(className).newInstance();
                    //获得目标类实例，传入目标类名及包名
Class c = Class.forName(className);
Method m = c.getMethod(methodName,new Class[]{...});
                        //传入方法名和参数类型数组
m.invoke(instance, new Object[]{});
    //方法执行，传入目标类的实例和方法参数值数组
```

4.3 xml 文件处理器

xml 文件处理器主要用于对 bean-config.xml 文件进行解析，本框架采用 jdk1.5 自带的 org.w3c.dom 包来解析 xml 文档，为文档对象模型(DOM) 提供接口。

xml 文件处理器根据传入的 xml 文件生成 Document 节点，Document 可看做是 xml 在内存中的一个镜像，对 Document 操作能够直接同步到该 xml 文件。关键代码如下：

```
DocumentBuilderFactory dbf=DocumentBuilderFactory.new
    Instance();
DocumentBuilder db=dbf.newDocumentBuilder();
                    //通过工厂得到一个 DocumentBuilder
Document doc=db.parse("bean-config.xml");
//DocumentBuilder 通过解析 xml 文件得到一个 Document
```

4.4 组件工厂类的实现

根据 xml 文件的 bean 节点建立组件对象，首先利用 Document 的 getElementsByTagName 方法获得所有 bean 节点的 NodeList 对象，遍历 NodeList 对象获得每个 bean 节点的 Node 对象，再利用 Node 的 getAttributes 方法获得该节点的所有属性，然后根据获得的 id、class 属性就可以实例化组件。关键代码如下：

```
NodeList nodes = doc.getElementsByTagName("bean");
                        //获得所有的 bean 节点
... ...
Node node = nodes.item(i);//获得其中一个 bean 节点
NamedNodeMap attributes = node.getAttributes();
                        //取出该节点的所有属性值
... ...
Class cl = Class.forName(class 属性值);
Object instance = cl.newInstance();    //创建该类的实例
```

4.5 组件外观设置类实现

从组件工厂类中获得组件对象并从 xml 文件中获得的方法名、参数值和参数类型，利用 Java 反射技术就可以为组件执行方法设置组件外观。

4.6 事件执行类

事件执行类继承多个事件接口，同时实现接口对应的方法。在每个实现的方法中，获得 xml 文件中 event 节点的 class 属性值以及 method 属性值，利用 Java 反射技术就可以执行方法。这时当组件触发事件时，执行事件执行类的对应方法，而事件执行类的方法是调用 method 属性值的方法。这样就实现了当组件触发事件时，执行 method 属性值的方法。

通过事件执行类，可以自定义触发事件时执行的方法名，实现了事件监听与事件处理的分离。事件执行类采用单例模式实现即仅有一个实例运行，节省了内存消耗。

4.7 事件监听器添加类

传统 GUI 编程中，事件监听器的添加是利用组件调用相应的方法，并传入对应的事件监听器对象。在本框架事件监听器添加类中，首先获得 event 节点的 type 属性值，通过 Java 反射技术把事件执行类实例添加到组件中，这样当组件触发事件时就可以执行事件执行类的相关方法。

在GUI设计中将组件设计和事件处理交予本文框架管理，降低了对象之间的依赖程度。在代码中仅需要编写get、set方法，也不需注册监听器、实现接口等代码，减少了代码编写量，实现了业务对象的松散耦合。事件触发和事件执行实现了分离，提高了程序的可维护性。对组件状态或事件信息的改变不需修改源代码，只需要修改配置文件，易于实现重构。

实践表明，该框架简单易用，建立的图形用户界面(GUI)具有较高的灵活性、可维护性和可扩展性，对构建中小型的GUI应用具有良好的支撑作用和借鉴意义。

参考文献

[1] 钱银中.Java程序设计案例教程[M]. 北京：机械工业出版社，2010.

[2] 宋淼，袁兆山，陈刚.Java事件处理机制中设计模式的分析[J].安徽工业大学学报，2004，27(11)：1383-1386.

[3] 魏学松，张育平.IOC框架的研究与设计[J]. 计算机技术与发展，2006，16(3)：213-216.

[4] 吴其庆. Java编程思想与实践[M]. 北京：冶金工业出版社，2006.

(收稿日期：2010-12-20)

作者简介：

廖福保，男，1976年生，讲师，硕士研究生，主要研究方向：计算机应用与软件的研究开发。