一种新的设计模式——接口呼叫模型①

A new design pattern —— Interface – calling Model

汪诚波　骆静峰　（浙江大学宁波理工学院计算机科学与技术系 宁波 315100）
何钦铭　（浙江大学计算机学院 杭州 310000）

摘　要： 本文提出了基于接口呼叫模式的新型设计模式，详细描述了该模式的运行机理及实现方法，分析了该设计模式的特点，重点分析了其解耦特性。这种设计模式特别适合大型的软件框架设计，本文给出了该设计模式在 Web 软件构架中的应用实例。

关键词： 设计模式　接口呼叫模型　Java 反射机制　耦合度　软件框架

1　前言

软件项目成功的关键之一是构建一个松耦合、可维护性好、可扩展性好、重用度高的软件框架。软件框架的设计基于开发平台的设计模式。Java 是一个较为优秀的开发语言，具有某些其他语言所不具备的功能，如反射机制、面向接口编程等，因此利用 J2EE 作为设计框架的技术基础越来越流行，Java 的主要设计模式有：创建模式、集合模式、结构模式、行为模式、并发模式等。这些模式各有各的特点，并在软件架构设计中广泛应用。不同模式应用场合不同，解决问题的方法也不同，在实际构建一个软件架构中，一般综合了多个设计模式—架构设计模式，如：Web 开发架构之一——MVC 架构综合了行为模式，策略模式等设计模式，在此基础了，出现许多优秀的 web 软件开发框架（如：Struts，Spring 等），它们各具特色。本文提出了接口呼叫模型——一种架构级的设计模式，它也综合了工厂模式，接口模式，控制反转模式，调停模式等多个设计模式，具有松耦合，可维护性好，可扩展性好的特点。

2　接口呼叫模型

2.1　背景

在设计模式中，OCP 原则（开—闭原则）是一个基本设计原则，任何设计模式无论通过什么工作模式实现，都必须遵循 OCP 原则。许多设计模式（如：工厂方法模式、接口模式、控制反转模式等）在 Java 平台具体实现设计过程中，广泛采用面向接口的编程方法。实际上，上述设计模式的 Java 实现，其核心归结为接口的实例化。

这些模式在具体实施过程中，各有特点，接口实例化方法也不同，下面是对现有三种方式特点分析：

（1）接口模式。Ia a = (Ia) (new A())；Ia 为接口名，A 为 Ia 的实现类名。

在调用类中直接利用实现类对接口进行实例化，代码直观，但是在代码中调用者和被调用者紧密耦合。

（2）工厂模式。Ia a = Factory. getA()；Factory 为工厂类，getA() 方法中的实现是 return (Ia) (new A())；

利用工厂类对接口和实现类的创建进行类封装，调用者和被调用者的依赖关系被放到了工厂类，以后修改工厂类即可，实现了一定程度的解耦，但是不适合大型系统，大型系统中类似于 Factory 的工厂类的数量会很庞大，维护困难。

（3）控制反转模式（Ioc 模式）。用来解决组件（实际上也可以是简单的 Java 类）之间依赖关系、配置及生命周期的设计模式，其中对组件依赖关系的处理是 IoC 的精华部分。IoC 的实际意义就是把组件之间的依赖关系提取（反转）出来，由容器来具体配置。

例如 Spring（一种流行的 Web 软件框架）的 Ioc 模

① 项目资助：宁波市重点专业建设项目

式,Spring 加载 application context 文件的时候会创建和缓存每个已配置 bean 的实例,编制程序时,只要写被调用者的接口代码,具体子类实例可通过 Spring 配置实现。通过 setter 方法或调用者的构造函数将接口对象注入给调用者。简单的说就是通过 Ioc 将与调用类有关的接口对象注入到调用类中,代码松耦合,但是依赖关系配置在配置文件中,配置文件中存在紧耦合问题,维护复杂,大量使用反射机制,运行效率较低。

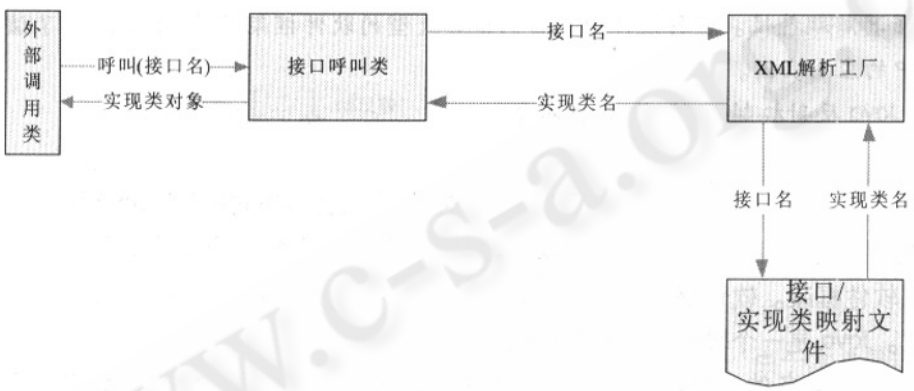

图 1 接口呼叫模型框图

一种好的接口实例化方法能够提高软件的可维护性,可移植性,可扩展性以及运行效率,并且软件中的各个部分松散耦合,使软件在日后能够满足用户不断增长的需求。

接口呼叫模型结合了现有接口实例化的三种方式的优点,综合多种设计模式,它实际上不是单纯的设计模式,属于架构级的设计模式。它集成了多种设计模式的优点。以下就其模型的构成、运行机理等展开分析。

2.2 模型机理

接口呼叫模型属于架构级的设计模式,它综合了工厂模式,接口模式,控制反转模式,调停模式等多种模式,其工作模式是:呼叫注入(CI)。

2.2.1 模型组成

呼叫模型主要由三部分组成,接口/实现类映射(XML),XML 解析工厂,接口呼叫类(Java 反射机制)。呼叫模型的工作模式为呼叫注入(CI),在需要用到接口的时候通过接口名呼叫外部模型(接口呼叫模型),将特定的实现类对象注入到调用类中,具体的对象的实例化由接口呼叫模型完成。

2.2.2 模型运行过程

(1) 当外部调用类需要某个接口的方法时,通过所需接口的接口名(即接口名作为传递参数),利用接口呼叫类提供的静态方法 getDao(String name),得到该接口某个实现类的对象。简单的说就是利用接口名得到实现类对象,而不是通过实现类的类名得到对象,这样在代码中实现了解耦。

(2) 接口呼叫类的 getDao 方法中封装了 XML 解析工厂的 getClassObj(String name)方法,getClassObj 方法的作用就是:通过传入的接口名返回映射文件中与其对应的实现类名。getDao 方法会将返回的类名通过反射机制装载成 Class 对象,并通过 Class 对象实例化一个对象,以 Object 形式返回给外部调用类。

(3) 接口和实现类的对应关系配置在接口/实现类映射文件(XML)中。映射文件结构为:<接口名>实现类的完整类名</接口名>,其中接口名也可为接口别名,用来适应一个接口有多个实现类的情况,因为在 XML 文件中只是配置简单的接口和实现类之间的对应关系,并没有配置类与类之间的依赖关系,所以在配置文件中几乎不存在紧耦合问题。

2.3 特点分析

接口呼叫模型结合了现有多个模式的优点,具有代码直观,松耦合,不需要书写工厂类来管理依赖关系,在注入前无需生成事先对象,效率高,配置文件简单,配置项之间独立且无依赖关系,小巧灵活等优点。

2.4 具体实现

2.4.1 接口/实现类映射(XML)

映射文件用来管理接口与其实现类的对应关系,结构简单,不存在着配置文件紧耦合的问题。结构为:<接口名>实现类的完整类名</接口名>,其中接口名也可为接口别名如 IstuDao1,用来适应一个接口有多个实现类的情况。

<lstuDao > com. nit. luojf. hibernate. DAO. StuDao </lstuDao >

<luserDao > com. nit. luojf. hibernate. DAO. UserDao </luserDao >

2.4.2 XML 解析工厂

利用 DOM 技术对接口/实现类映射文件进行解析,具体方法:在工厂类中编写 static 块,在这个 static 中用 DOM 解析接口/实现类映射文件,优点是:在各方法调用前就进行一系列的初始化,而且只进行一次,XML 文件解析后放在内存中,以后要调用类中的方法操作文件时,不再需要再解析文件,直接调用工厂类中的方法就可以对 XML 文件进行操作,可以提高呼叫效率,减少呼叫响应时间。这里运用最频繁是 getClassObj(String name)方法,作用是通过接口名得到该接口的实现类的类名,是接口呼叫模型的底层实现。

解析工厂还提供了动态加载功能,在系统运行期间,如果开发者对映射文件进行了更新,解析工厂会自动对更新后的接口/实现类映射文件进行重新解析,而不需要重启整个应用系统,提高了系统维护的效率和灵活性。

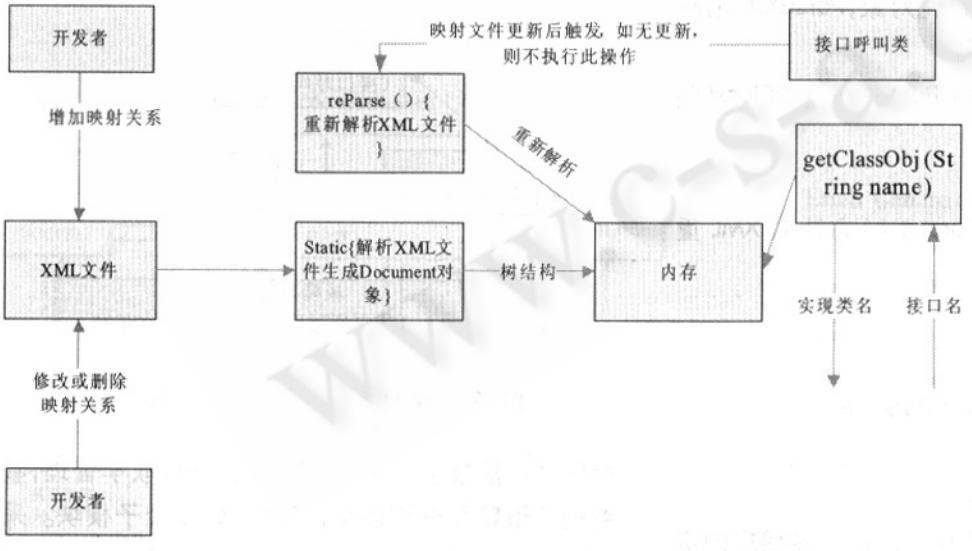

图 2　解析工厂流程图

解析工厂具体实现(主要部分):

```
public class ParseXml{
    ........
    static{
    ........
    path = "接口/实现类映射文件的路径";
    ........
    doc = db.parse(path);//doc 为解析后的 Document 对象,在整个系统运行期间都存在。
    ........
    }
    public static String getClassObj ( String name ) throws Exception {
      //参数为接口名,通过接口名得到实现类的类名
      ........
      Node t = e.getFirstChild( );//文字内容,所要的到的类的全名
      String s =t.getNodeValue( ).toString( );
      return s;
    }
    public static void reParase( ){
          //对 XML 文件重新解析,功能和代码与 static{ } 中基本一样
      }
      ........
}
```

2.4.3 接口呼叫类

利用接口呼叫类的呼叫注入模式,可以在外部类中直接通过接口名来得到接口的实例,不需要显式的调用接口实现类的构造函数,类中也不需要出现任何实现类的代码,只要在这个类中知道接口名称就可以,通过接口呼叫模型来完成接口的对象的注入。

用到呼叫模型的类需要导入接口呼叫类,DaoFac-

tory. java 为接口呼叫类，在类只有一个静态方法 getDao(String name)，该方法中利用反射机制将 XML 解析工厂 getClassObj 方法返回的实现类类名装载成 Class 对象，通过类对象实例化，以 Object 的形式返回给外部类（调用 getDao 方法的业务类），完成呼叫注入。

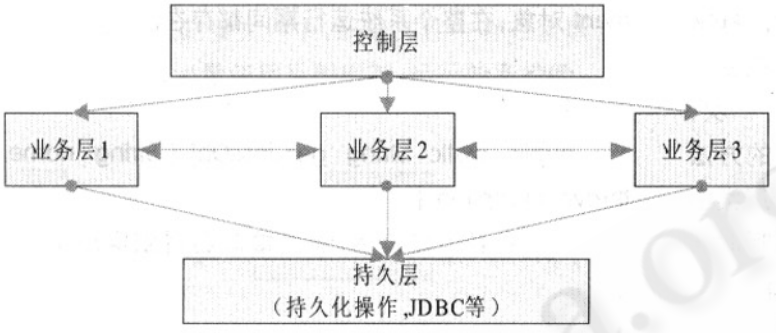

图 3

接口呼叫类实现：

```
    import com. nit. luojf. xmlparsemodel. ParseXml;//
导入 XML 解析工厂
public class DaoFactory {
  public static Object getDao( String name) {
      try{
//加载类对象，用类对象来实例化对象，ParseXml. get-
ClassObj( name) 返回实现类名
       return Class. forName ( ParseXml. getClassObj
( name) ). newInstance( ) ;
    }catch( Exception e) {
      //触发解析工厂的 reParase ( ) 方法对 XML 重
新解析
      ParseXml. reParase( ) ;
      try{//重新装载
        return Class. forName( ParseXml. getClassObj
( name) ). newInstance( )
      }catch( Exception ee) {
        return null;//仍然没有与该接口名相应的实
现类，返回空
      }
    }
  }
}
```

3　应用实例

接口呼叫模型（Call Model）在 Java 软件构架中可独立出来作为接口层，独立与具体应用，用于管理业务处理，松散应用系统的耦合，提高开发效率、可维护性和扩展性。以 Web 构架为例，未使用呼叫模型前的软件结构框图如图 3 所示。

由图可知，层与层之间存在着紧密的耦合度，可重用度低，几乎不具备可扩展性。

接口呼叫模型可以使软件结构松耦合，提高软件重用度和可维护性，层次结构分明。如图 4 所示。

这里以 Web 构架为例，采用 Struts + Hibernate + CallModel (SHC) 框架开发网络教学及监控平台，网络教学平台及监控平台包括教学评价，教学互动平台，教学管理，教学监控及预警系统等模块，模块下分多个子模块。采用 SHC 框架进行开发的过程中，通过 Hibernate 的 O/R Mapping 将数据库中的表映射为 POJO，并实现对象之间的关系映射。通过过滤器管理数据库连接和事务处理，将开发者从烦琐的重复代码中解放出来。模块的功能实现采用针对接口编程，在调用类中如果要用到

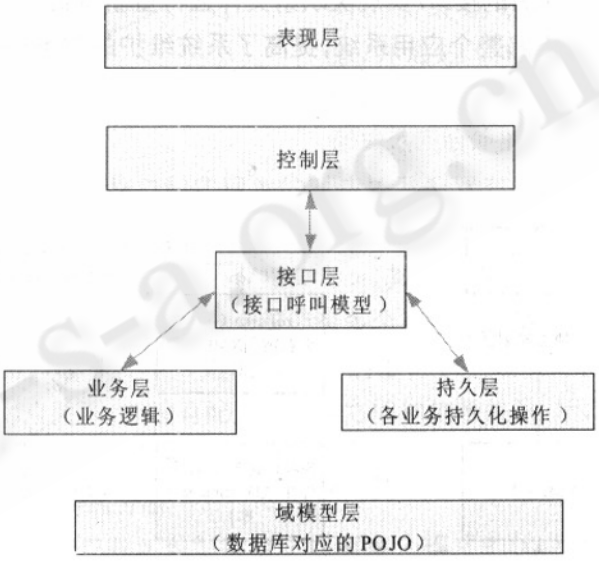

图 4　接口呼叫模型的 Web 软件构架

其他接口，则通过接口呼叫模型对接口进行实例化，实例化方式为

接口名 接口对象 = (接口名) DaoFactory. getDao ("接口名");

以后在该调用类可以直接利用接口对象使用接口提供的方法处理业务逻辑，在代码中实现松耦合。映射文件也很简单，易于维护。

4 总结

接口模型结合了多种设计模式的优点，通过接口呼叫模型的呼叫注入模式，可以完成接口对象的实例化，代码耦合度松散，映射文件配置简单，易于维护，利用呼叫模型开发的软件构架层次结构清晰，软件重用度高，模块之间相对独立，具体功能实现通过接口层封装，安全性高。模型不事先生成实例对象，在运行中生成，访问一次内存即可得到实例对象，效率高。

但接口呼叫模型有其自身的缺点，它不具备事务处理，任务调度服务，远程调用服务，Hibernate 持久化操作等的能力，在某些需要用到这些服务的应用系统中需要借助其他工具来完成这些服务。

参考文献

1 E Gamma Object - oriented software development based on ET + + :design patterns, class library, tools(in German). PhD thesis, University of Zurich, 1991[M].

2 G Booch. Object - oriented analysis and design with applications, 2nd edition. Benjamiin - cummings Pub. Co. Redwood City, California, 1993[M].

3 Partha Kuchana. Software Architecture Design Patterns in Java(中文版:Java 软件体系结构设计模式), 北京:电子工业出版社, 2006.2[M].

4 张友生等, 软件体系结构, 2004 年第 1 版, 北京:清华大学出版社, 2004.1[M].

5 赵会群、王国仁、高远, 软件体系结构抽象模型, 计算机学报, 2002(7):730 ~ 736[J].