设计模式及组件技术在业务逻辑层中的应用①

刘　锋[1]，孙　咏[2]

[1](中国科学院 研究生院，北京 100049)
[2](中国科学院 沈阳计算技术研究所，沈阳 110171)

摘　要：在应用程序的后期维护中，大部分的维护是由于业务逻辑的变化而导致的，但是在基于 MVC 的开发模式中，只是实现了模型-视图-控制的解耦合，而在业务层中业务之间还存在紧密的耦合关系，如果一个模块发生改动就会产生连锁反映，导致一系列相关模块的改动。以举报业务为例，采用代理模式、工厂模式、接口模式，实现了业务组件调用的动态化及业务组件之间访问的间接化，业务功能与业务逻辑的分离，构建了一个扩展性强、易于维护和配置灵活的业务逻辑层模型。

关键词：设计模式；组件技术；软件维护性；解耦合

Application of Design Patterns and Component Technology to Business Logic Layer

LIU Feng[1], SUN Yong[2]

[1](Graduate University, Chinese Academy of Sciences, Beijing 100049, China)
[2](Shenyang Institute of Computing Technology, Chinese Academy of Sciences, Shenyang 110171, China)

Abstract: In the post-maintenance of software application, the change of business logic layer occupies the majority, but some traditional development frames, for example structs, spring and so on, which based on MVC pattern has only realized the model-view-control solution coupling, however the business also has the close coupling relations in the business logic layer. If a change happens in the module, that will cause a chain-like reflection and a series of related modules change. This article uses the business case report as an example, which achieved the business component transfers and the business component visits indirectly, separated the business functions from business logic, so constructed an extension strong, easy to maintain and configure nimble module in business logic layer.

Key words: design patterns; component Technology; software maintenance; decoupling

为了保护辽河流域的环境安全，应建立和完善“预防、预警、应急”三位一体的管理体系，实现对污染源的有效监督管理；为了全面反映环境质量状况和趋势，及时掌握河流水质变化情况，对水环境突发事件进行有效预警，建立了辽河流域水环境预警系统。但采用典型的 MVC 开发模式时，系统通常被分为 3 层：表示层、业务层、控制层。如一些流行的 web 开发框架都是基于 MVC 模式的，如 structs、spring 等[1]，而这些框架没有解决业务逻辑之间的耦合关系。对于一个系统来讲，设计一项“业务”是它最复杂的部分。通常情况下完成一项业务需要很多步骤，这些步骤随着时间的推移变得不稳定，因此系统就会随着业务的变化开始产生不稳定。通过简单统计，业务流程的调整及业务发生变化占到后期维护工作的 58%[2]。

辽河流域水环境预警系统是一个复杂的系统，这极大地增加了软件系统开发难度和后期的维护难度，这不但增加了开发成本，也使开发周期延长。因此采用适合的架构和设计模式，改善软件系统的适应性、灵活性，进而提高系统的维护性、复用性的研究就显得非常有价值。

1　业务层模型

大多数基于MVC的开发框架只是面向程序架构，并没有描述业务逻辑，实际上任何一个模块都面临着

① 基金项目:国家水体污染控制与治理科技重大专项(2009ZX07528-006-05)
收稿时间:2011-01-28;收到修改稿时间:2011-03-03

多种输入/输出选择和不止一个的关联，每一个业务对于发过来的请求至少都会有两种反馈：正常结果和异常结果。大多数的业务模块还会和数据库及其他业务模块打交道，这样业务模块就会变得刚性十足，其复用性和可维护性变得很低，这也就是我们的系统为什么经常会变得难以维护和扩展的原因之一。

1.1 业务层模型的提出

下图为传统的 MVC 框架开发开发举报业务的示意图，从图中可以看出各个模块之间存在很强的耦合性。

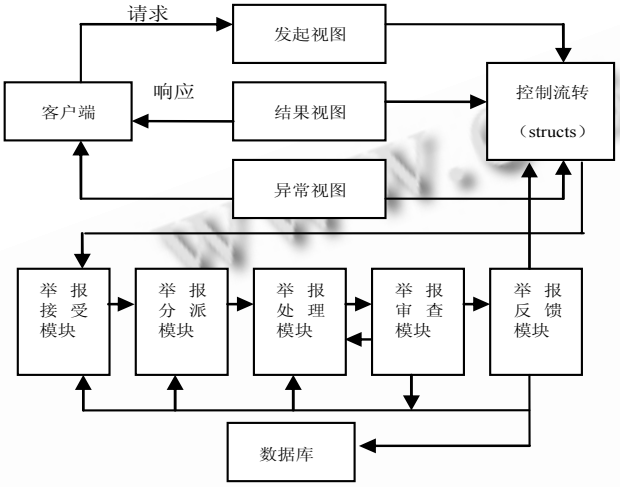

图 1 MVC 框架对于举报业务的示意图

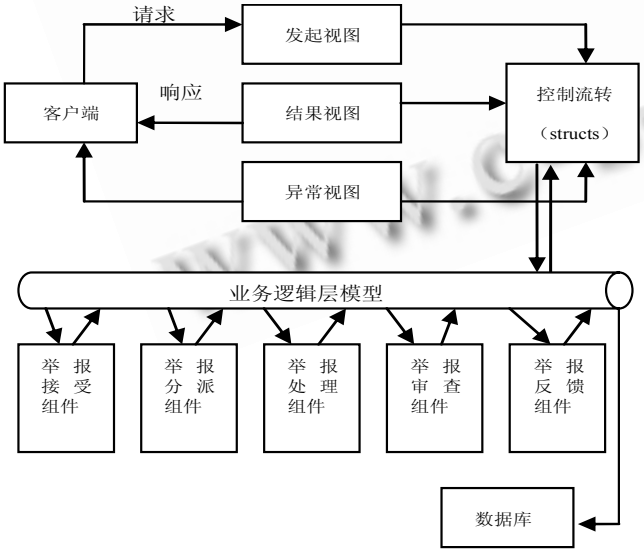

图 2 采用业务层模型后的示意图

本文设计的业务层模型采用组件技术，代理模式、工厂模式及接口模式实现了功能和逻辑的解耦合，基本上实现了 OCP 原则。业务层模型根据业务配置信息，调用相应的组件来完成一项业务，使每个业务组件之间不再发生直接联系，因此一个业务组件的更改不会影响到其他的组件。

1.2 业务组件

采用业务组件后，变动业务只需要改动业务配置信息和根据需要增减实体业务组件。采用这种开发方式基本上实现了 OCP 原则，OCP（Open-Closed principle）原则讲的是：一个软件实体应当对扩展开放，对修改关闭。这一原则最早由 Bentrand Meyer 提出。所有的软件系统都有一个共同的性质，即对它们的需求都会随时间的推移而发生变化。在软件系统面临新的需求时，系统的设计必须是稳定的。满足 OCP 原则的系统具有两个无可比拟的优越性：①通过扩展已有的软件系统，可以提供新的行为，以满足对软件的新需求，使变化中的软件系统有一定的适应性和灵活性。②已有的软件模块，特别是最重要的抽象层模块不能再修改，这就使变化中的软件有一定的稳定性和延续性[3]。

1.2.1 业务组件的优势

在传统的开发模式中没有实现业务层的解耦合，在业务层中如果程序中的一处改动就会产生连锁反映，导致一系列相关模块的改动，因此程序在修改或维护时变得极为不便。比如客户要求举报业务包括举报接收、举报分派、举报处理、举报反馈四个部分。

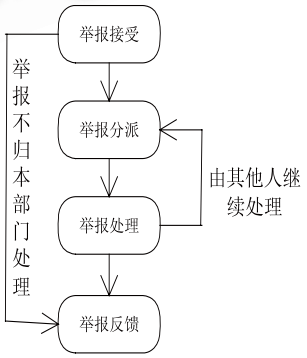

图 3 举报系统业务逻辑关系图

当有举报发生时，①举报接受模块先判断是否归本部门处理，如果可以处理就为举报事件指定核查员，如果不能处理就告诉举报人不能处理。②举报分派模块根据举报类型指定处理人。③举报处理模块处理举

报，如果是需要多个业务员或部门共同处理的举报，则把未完成的举报交给其它业务员或部门继续处理。④举报反馈模块把举报处理结果反馈给举报人。从图 3 中可以看出，四个业务模块存在相互调用关系，紧密的耦合在一起。例如客户试用一段时间后，要求举报在反馈给举报人之前要有相关领导的审查，审查合格后才能反馈给举报人，否则发回相应的步骤重新处理。这样的改动是灾难性的，不但要增加一个举报审查模块，而且几乎要改动其它所有的模块。

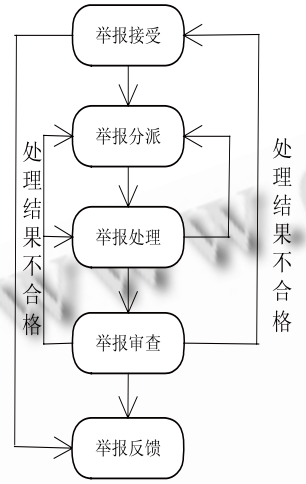

图 4　举报业务需求变动后的逻辑关系图

造成应用程序修改困难的原因是业务逻辑与功能耦合在一起，一般情况下，业务逻辑是不稳定的，易改变的，而功能单元是稳定的。如果把功能单元设计成组件，功能组件只实现单一的业务功能，不包含业务逻辑。

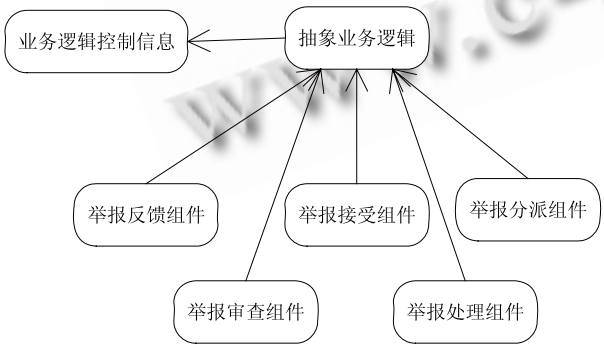

图 5　引进组件后的举报业务关系图

一项业务的所有功能组件继承同一个抽象业务逻辑或者实现同一个抽象业务逻辑中的接口，应用程序根据业务逻辑配置信息调用相应的抽象业务逻辑类，来完成一项业务，这样就实现了业务逻辑与功能的解耦合。业务逻辑的改动不会影响到具体的功能组件。

1.2.2 业务组件的设计、

通过分析已有组件的优缺点[4-7]，设计并实现了适合本模型的组件。图 6 是本文设计的业务组件，它包括四部分：①组件标识部分是区别不同业务组件的标识符；②组件属性记录一些基本的属性：作者、版本、创建时间、使用环境等；③功能描述介绍了此业务组件的运行环境和所实现的功能，方便检索及复用此组件；④对外接口提供了应用程序调用组件的接口。

本文设计的业务组件应具有四点要求：

① 唯一的输入和输出接口（满足单入单出要求）；

② 组件之间不允许直接通讯（满足间接通讯要求）；

③ 组件的增减不影响其他组件(满足扩展开放要求)；

④ 业务逻辑类具有足够抽象(满足对修改封闭要求)。

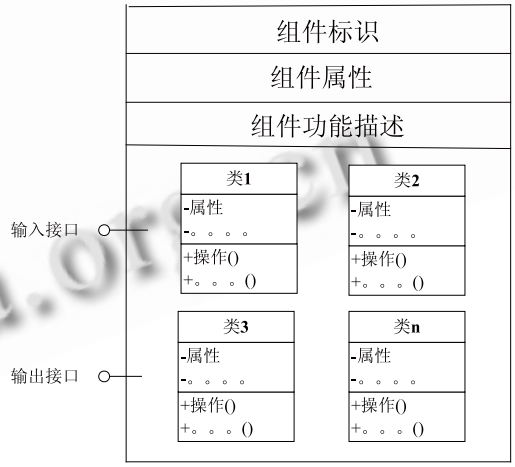

图 6　业务组件模型图

2　业务层模型的实现

代理模式一个重要好处：重要关系的分离。任何对业务发起的请求都变成对业务代理的调用，这样就利用代理模式将访问间接化。根据举报请求，业务代理从业务配置管理系统中读取举报业务组件的配置信息，然后业务加载工厂利用 Java 反射机制，实例化业务功能组件。业务代理根据举报业务的流程配置信息，依次调用相应的业务组件，完成举报业务。如图 7 所

示。

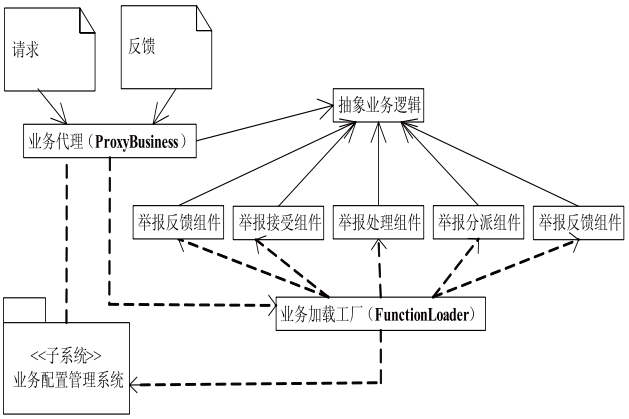

图 7 业务模型的局部示意图

当有举报发生时，业务代理组件通过 getMessage 方法获得前台传来的举报消息，从业务配置管理系统中读出举报业务的组件信息（图 11 所示），然后调用业务加载工厂 loader()方法把举报业务所需要的五个组件实例化；业务代理组件再通过 dowork 方法获得举报业务的流程配置信息（图 12 所示），dowork 是抽象业务逻辑中的接口，每个具体业务组件都实现 dowork 方法，业务代理根据流程配置信息依次调用 dowork 方法完成举报业务，而不用知道具体的业务组件。

2.1 抽象业务逻辑

抽象业务逻辑统领所有的业务组件，包含业务中最重要的逻辑和对整个业务来说重要的战略决定，而具体的业务组件则含有一些次要的与实现有关的算法和逻辑。具体的业务组件依赖抽象业务逻辑类，它决定业务组件的实现和具体算法的改变。这种设计思想是软件设计中“依赖倒转原则（Dependence Inversion Principle）”的体现。本文中几乎所有的实体组件对象都基于抽象业务逻辑类。如图 8 所示。

抽象业务逻辑类（**AbstractBusiness**）
-data：Object=null
+getMessage(data:Object):void() +doWork(data:Object):void() +returnResult():Object()

图 8 抽象业务类的类图

抽象业务类提供了单入单出的基本方式，并且奠定了业务组件的基本输入输出方式：调用者通过 get Message 方法获得处理请求，dowork 为处理业务请求的接口，通过 returnResult 方法传回处理结果。

2.2 业务代理

任何对业务发起的请求都变成对业务代理的调用，这样就利用代理模式将访问间接化，业务代理根据请求的类型调用相应的业务组件。

业务代理（**ProxyBusiness**）
-被代理对象（ConcreteBusiness）:AbstractBusiness=NULL
+权限检查（CheckPermissions()）:boolean() +预处理（Pretreat）:boolean() +功能处理（doWork）:void()

图 9 基本业务代理的类图

这里要说明几点：

① 被代理对象 ConcreteBusiness，或者说叫真实的业务对象，是通过工厂化方法构造出来的，使用者无需知道返回的到底是哪个真实的业务组件对象，只要按照抽象业务声明的方法去使用即可。

② 预处理 pretreat，是一个重要的方法，在代理接收完请求后马上就调用该方法，该方法是将“请求”真正传递给真实业务组件前动作。该方法实际至少执行了两个动作：权限检查和高级功能处理。

③ 权限检查 checkPermissions，是一个保护方法，它被预处理调用，它根据发过来的请求，向权限管理子系统发出验证请求，并根据反馈对原始请求做出判定。

④ 功能处理 doWork，调用业务组件完成一项业务等等。

2.3 业务加载工厂

业务加载工厂是有效的工具，他们使得在创建业务组件实例的时候无需依赖这些组件的具体实现。

业务加载工厂（**ConcreteBusinessLoader**）
-data:Object=null
+readConfigureFile():String() +Loader():Object()

图 10 业务加载工厂类图

业务加载工厂就是根据业务代理的请求，实例化业务组件，反馈给业务代理。通过这种方式的工厂模式，对于增减业务功能组件的操作无需修改原有代码，只需要调整配置文件即可。这里要说明几点：

① 实例化的业务组件对象 Object，业务组件只有被业务加载工厂实例化后，才可以被抽象业务逻辑调用，完成一项业务。

② 读取业务组件配置文件方法 readConfigureFile()，从业务配置管理系统中读取业务组件的基本信息，包括：组件的路径等。

③ 业务组件加载方法 Loader()，根据配置信息通过 Java 反射机制生成相应的组件对象。

2.4 业务配置信息

在业务配置管理系统中的业务配置信息是在前人的基础上分析总结而来[8-10]。配置信息分为组件信息和业务逻辑信息两类，其基本格式如图 11、图 12 所示。

```
<?xml version="1.0" encoding="UTF-8"?>
<business-set>
<business-set-name>complaints</business-set-name>
<business>
  <business-name>receive-complaints</business-name>
  <business-class>
    cn.com.business.complains.receivecomplaints
  </business-class>
</business>
<business>
   <business-name>assign-mission</business-name>
   <business-class>
     cn.com.business.complains.assignmission
   </business-class>
</business>
<business>
   <business-name>deel-complaints</business-name>
   <business-class>
     cn.com.business.complains.deelcomplaints
   </business-class>
</business>
<business>
  <business-name>examine-feedback</business-name>
  <business-class>
    cn.com.business.complains.examinefeedback
  </business-class>
</business>
<business>
  <business-name>feedback-result</business-name>
  <business-class>
    cn.com.business.complains.feedbackresult
  </business-class>
</business>
```

图 11　举报业务的组件信息

以下是举报业务组件信息文档的说明：

<business-set>表示即将声明一项业务，文档中可有多个<business-set>声明，但不能重名；<business-set-name>描述业务的名称叫 complaints；<business-name>说明了一个叫 receive-complaints 的业务组件，其对应的类为 cn.com. business. complaints. receive-complaints；一个<business-set>块内可以有多个<business>，但是它们的名子<business-name>不能重复，并且每个业务声明块内只能有唯一的名字，但在不同的业务内允许重复使用。

业务代理根据业务流程信息调用业务组件来完成一项业务，由于篇幅所限业务组件所需要的数据在此没有给出。下图为举报业务流程配置信息。

以下是对业务流程配置信息文档的说明：

<business-flow>表示业务组件的调用关系，<jump-condition>表示业务流程中此组件调用下一个组件的条件，<business-next>表示业务流程中要调用的下一个组件。应用程序根据组件产生的结果，通过<jump-condition>的值决定下一步调用那个组件。<business-next>的值指明了下一步的组件。

```
<business-flow>
   <business-name>receive-complaints</business-name>
   <jump-condition>YES</jump-condition>
   <business-next>assign-mission</business-next>
   <jump-condition>NO</jump-condition>
   <business-next>feedback-result</business-next>
</business-flow>
<business-flow>
   <business-name>assign-mission</business-name>
   <jump-condition>ok</jump-condition>
   <business-next>examine-feedback</business-next>
</business-flow>
<business-flow>
   <business-name>deel-complaints</business-name>
   <jump-condition>ok</jump-condition>
   <business-next>examine-feedback</business-next>
   <jump-condition>continue</jump-condition>
   <business-next>assign-mission</business-next>
</business-flow>
<business-flow>
   <business-name>examine-feedback</business-name>
   <jump-condition>ok</jump-condition>
   <business-next>feedback-result</business-next>
   <jump-condition>deel-wrong</jump-condition>
   <business-next>deel-complaints</business-next>
   <jump-condition>assign-wrong</jump-condition>
   <business-next>assign-mission</business-next>
   <jump-condition>receive-wrong</jump-condition>
   <business-next>assign-mission</business-next>
</business-flow>
</business-set>
```

图 12　举报业务流程配置信息

当有举报消息到达时，业务代理根据举报业务的流程配置信息首先调用举报接受组件，举报接受组件处理后，把处理结果返给业务代理，如果结果中的跳转控制信息是 YES 则调用举报分派组件，如果是 NO 则调用举报反馈组件。业务代理根据举报业务流程组件配置信息依次调用相应的业务组件，完成举报业务。

2.5 模型的应用效果

本模型已经应用于辽河流域水环境预警系统中，通过上文的论述可以看到，辽河流域水环境预警系统中的子系统和业务都是可以配置的。某项业务变更产生的影响只限定在与此项业务相关的组件上，这样系统设计人员和程序员将更多的注意力放到粒度更小的功能设计和实现上，对于新建和修改业务均无需修改原有代码，只要对配置文件进行修改即可。对于增减业务组件，也无需修改其他已有组件类。

对于已存在的组件，通过组件库统一管理。这样既可以节约重复设计组件的时间，又避免重新研发带来的新 BUG。对辽河流域水环境预警系统的简单统计，本模型在系统的复用方面较原来的模型提高近 40%，修改、扩展一个业务的工程时间缩短 35%，在

日常维护中所需要的维护时间明显缩短。

3 结语

结构良好的开发模型及组件技术可以提供更好的维护性，使软件开发及维护更容易，更具有应用价值。设计模式的使用能使模型结构变得更简洁、更易于理解，其中 OCP(“开--闭”原则)是面向对象可复用的基石，采用遵循 OCP 原则的模型可以有效地提高系统的复用性，同时提高系统的可维护性。目前，该模型已经应用到辽河流域水环境预警系统中，但是在配置文件管理，系统响应效率等方面还有进一步的改进空间。

参考文献

1 孙卫琴.精通 Struts:基于 MVC 的 Java Web 设计与开发.第 2 版.北京:电子工业出版社,2004.98–101.
2 孙咏.基于 OCP 软件应用架构的设计与实现[博士学位论文].北京:中国科学院研究生院,2009.
3 阎宏.JAVA 与模式.北京:电子工业出版社,2002.41–44.
4 郑志远.组件化在 J2EE 中的应用[博士学位论文].厦门:厦门大学,2008.
5 李俊峰.MVC 设计模式在 web 中的复用研究与实现[博士学位论文].武汉:华中科技大学,2007.
6 刘晓文,蒋泽涛，等.一种基于复用组件的 WEB 测控软件架构设计.微计算机信息,2007,22(2):300–305.
7 Wang XB. Research and Implementation of Design Pattern-Oriented Model Transformation.Computer Society: 2008,18(4):73–76.
8 王超,梁义芝.基于 WEB 数据库的软件配置管理研究.计算机与数字工程,2010,(1):85–86.
9 裴树军,陈德云,陈晓雪.软件配置管理在软件开发平台中的应用.哈尔滨理工大学学报,2010,(2):65–69.
10 Liang P, Li JY. A Change-Oriented Conceptual Framework of Software Configuration Management. Computer Society: 2009,14(8):85–88.

（上接第 43 页）

3 结语

本文设计的基于 DSP 的煤岩识别系统能够有效地分离煤岩混合声波信号，在这个系统的基础上，如果在加上更有效的分离算法，还可以进一步提高系统的准确性和实时性。此系统的实现对煤矿自动化的发展有着重要的实际应用价值。

参考文献

1 任芳,杨兆建,熊诗波.国内外煤岩界面识别技术研究动态综述,2001,10(4):54−55.
2 杨福生,洪波.独立分量分析的原理与应用.北京:清华大学出版社,2006.
3 申丽岩,方滨,沈毅.基于负熵极大的独立分量分析方法.中北大学学报,2005,26(6):396−400.
4 张倩蓉,王新新.混合语音信号的盲分离.山西电子技术,2008:16−17.
5 赵加祥.DSP 系统设计和 BIOS 编程及应用实例.北京:机械工业出版社,2008.219−230.