第16卷 第9期
2006年9月

计算机技术与发展
COMPUTER TECHNOLOGY AND DEVELOPMENT

Vol.16 No.9
Sep. 2006

基于DI模式对Struts框架的扩展研究

戚荣志，王志坚
（河海大学 计算机及信息工程学院，江苏 南京 210098）

摘 要：为了解决使用标准Struts框架开发程序时所带来的组件之间的耦合程度比较高的缺点，在标准Struts框架中引入了Dependency Injection(DI)模式。在介绍了DI模式的基本原理和3种实现类型之后，基于该模式对标准的Struts1.2框架进行了扩展，主要加入了设值方法注入功能。扩展框架实现了对ActionForm Bean、数据库连接和一般对象的注入。在实际项目中基于此扩展框架进行了软件开发，发现该框架降低了系统的组件之间的耦合，简化了对单个组件的单元测试，效果非常明显。

关键词：控制反转；依赖注入；组件；Struts框架
中图分类号：TP311　　**文献标识码**：A　　**文章编号**：1673－629X(2006)09－0038－03

Research on Extension of Struts Framework Based on DI Pattern

QI Rong-zhi, WANG Zhi-jian
(School of Computer & Information Engineering, Hehai University, Nanjing 210098, China)

Abstract: In order to solve the shortcoming of the higer coupling between the components while developing with the standard Struts framework, the article intorduces the Dependency Injection (DI) pattern in it. This paper makes a detailed introduction on the basic principle of the DI pattern and its three types of implementation, and extends the standard Struts 1.2 framework by adding the setter injection. The extended framework implements the function of injecting ActionFrom Bean, database connection and general objects. This extended framework has been used to develop a software project. During the development, have found that this framework reduced the coupling between the components of the system greatly, simplified the unit test of the individual component. The result is very obvious.

Key words: inversion of control; dependency injection; component; Struts framework

0 引 言

J2EE开发者在程序开发过程中经常遇到这样一个问题：如何将不同的组件组装在一起，比如说如果Web控制器体系结构和数据库接口是由不同的团队开发的，相互之间对彼此的程序结构并不了解，开发者应该如何让它们配合工作。很多框架，如Spring[1]，PicoContainer[2]等，都对这个问题进行了研究，提出了组装的方案，帮助开发者组装程序组件。仔细分析这些解决方案，可以发现在这些框架的背后有着同一个模式：控制反转(Inversion of Control, IoC)。控制反转即由容器控制程序之间的关系，而不是在早期的实现中，由程序直接控制。Martin Fowler在文献[3]中认为，控制反转这个名字太泛了，不足以说明该模式的特点，于是他给该模式起了一个更能说明其特点的名字，叫做依赖注入(Dependency Injection, DI)。依赖注入就是组件之间的依赖关系由容器在运行期决定，形象地说，即由容器动态地将某种依赖关系注入到组件之中。DI是一种将调用者与被调用者分离的思想。

1 依赖注入

1.1 传统方法的缺点

在面向对象程序设计中，开发者将各种组件组装起来以协同工作，形成应用程序。组件之间存在着一定的依赖关系，客户端需要知道和哪些组件通信，如何定位那些组件，如何和那些组件通信。通常在客户端的逻辑中嵌入了定位和实例化其它组件的逻辑，这些被实例化的组件的实例是在编译期连入程序的。因此，当被实例化的组件的访问方式发生变化时，客户端的程序代码就要做相应的变化。在涉及的类不多以及类之间的关系不复杂时，需要进行的改动还不算大。但是，在实际的系统中，往往会有数十个组件，并且组件之间的关系也比较复杂，这时改动量就会变得很庞大。比如，如果在程序的某个类中需要访问数据源(DataSource)对象时，传统的编码模式通过编码初始化DataSource实例，这时DataSource实例是在编译期连入程序的，当DataSource发生变化时就需要进行相应的程序代码的修改。这种方法缺少灵活性，组件之间的相互依赖关系比较大，耦合程度比较高。

收稿日期：2005－12－18
作者简介：戚荣志(1980－)，男，江苏兴化人，助教，硕士研究生，研究方向为分布式对象技术、软件复用技术；王志坚，教授，博士生导师，博士，研究方向为分布式对象技术、软件复用技术、网络软件系统集成技术。

1.2 依赖注入的工作原理

针对上述方法的缺点,依赖注入可以很有效地降低对象之间的耦合。程序中客户端需要依赖于其它组件的实现时,通常在客户端组件的逻辑中并不嵌入实例化其它组件的逻辑,而是申明它们对其它组件的依赖关系,同时外部有专门的程序负责对其它组件的定位和实例化操作。这些专门的程序就是DI容器的一部分。这些被实例化的组件的实例将由容器在运行期动态注入。这样,当被实例化的组件的访问方式发生变化时,客户端的程序代码就不需要做相应的变化。这种实现方式就是依赖注入。依赖注入可以在运行期为组件配置所需资源,而无需在编写组件代码时就加以指定。对照上面的例子,DataSource的具体配置和初始化是由容器在运行期完成的,使用客户端类中的DataSource将由容器在运行期动态注入。通过依赖注入,不需要编写代码,只要进行一定的配置就可以指定客户端中的DataSource实例,在运行期客户端就能够利用容器注入的DataSource实例,完成自身的业务逻辑。当DataSource发生变化时,只需修改相应的配置代码或者配置文件(一般为XML文件),而无需修改客户端代码。对比传统的实现方式,可以看出,依赖注入使得系统实现非常灵活简洁,减轻了组件之间的依赖关系,大大降低了组件之间的耦合,提高了组件的可移植性和重用性。

引入依赖注入后组件之间的依赖关系如图1所示。

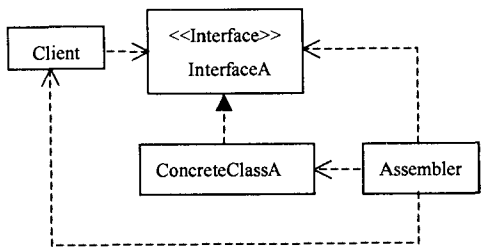

图1　引入依赖注入后组件之间的依赖关系图

图中Client类只依赖于InterfaceA接口,装配器(Assembler类)是DI容器的一部分,负责定位InterfaceA接口的具体实现,并将其实例赋给Client类的一个字段。Assembler在运行期将InterfaceA接口的具体实现注入到Client类中。

1.3 依赖注入的几种实现类型

依赖注入的形式主要有3种:构造子注入(Constructor Injection)、设值方法注入(Setter Injection)和接口注入(Interface Injection)[3]。

1)构造子注入。构造子注入是通过构造函数完成依赖关系的设定。在类的构造函数中包含所有需要注入的元素,容器在运行期通过调用类的构造方法将其所需的依赖关系注入其中。Spring和PicoContainer都支持构造子注入。

2)设值方法注入。设值方法注入是通过类的setter方法完成依赖关系的设定。在类中包含所有需要注入的元素的setter方法,运行期容器通过调用类中相应的setter方法向该类注入所需的依赖关系。Spring和PicoContainer都支持设值方法注入。

3)接口注入。接口注入是在接口中定义需要注入的信息,并通过接口完成注入。

在这3种依赖注入模式中,设值方法注入在实际开发中的应用最为广泛。下面将基于DI模式对Struts框架进行扩展,使其支持设值方法注入。

2 扩展Struts框架

在使用标准的Struts1.2框架[4]进行Web应用程序开发时,当某个类的功能的实现需要依赖于其它类的实例的时候,使用的方法是直接实例化这个类。这也就是上文提到的传统方法所使用的程序结构。为了解决传统方法的缺点,基于DI模式对Struts框架进行了扩展,并基于扩展后的框架进行实际开发。

2.1 扩展思路

1) Struts插件:AppContainerPlugIn。Struts1.2框架提供了动态插入和加载组件的功能,这种组件称为Struts插件(PlugIn)[5]。Struts的插件机制比较方便、灵活,基于该机制可以很方便地在Struts框架中加入依赖注入的功能。实现依赖注入功能的模块称为AppContainer。AppContainer中包含的Struts插件为AppContainerPlugIn。该插件实现了org.apache.struts.action.PlugIn接口。AppContainer通过Struts的插件机制被初始化,保存到Web服务器的Application范围内,从而使Web应用的上下文中只存在一个AppContainer。

2)扩展Action类。实现DefaultAction类,该类继承了org.apache.struts.action.Action类。框架将所有的用户请求都转发给该类。DefaultAction类最主要的功能是实现对AppContainer的调用。DefaultAction由org.apache.struts.action.ActionMapping类的getParameter()方法取得的参数字符串决定调用哪个业务对象的哪个方法,AppContainer根据此字符串调用相应的业务对象的方法。DefaultAction类的实现使得框架具有很好的通用性。

3)加入依赖注入机制。为了使业务对象能够定位到所依赖的对象,提高组件的重用性,这里采用了依赖注入机制以取代传统的显式实例化类的方法。在AppContainer中,AppContainer类用于实现图1中Assembler的功能。该类支持对业务对象进行设值方法注入。只要注入元素遵循一定的规则,并且业务对象中存在注入元素的setter方法,AppContainer类就能够将该元素注入到业务对象中。这里的规则可以在配置代码中进行配置,或者在配置文件中进行配置。文中选择了更为灵活、简便的XML配置文件。同时,AppContainer类提供call()方法来实现对业务对象的方法的调用。AppContainer类还支持对业务对象进行数据库连接(Connection)的注入。

2.2 扩展框架的架构

依据上面的扩展思路,可以得到如图2所示的扩展框

架的基本架构。

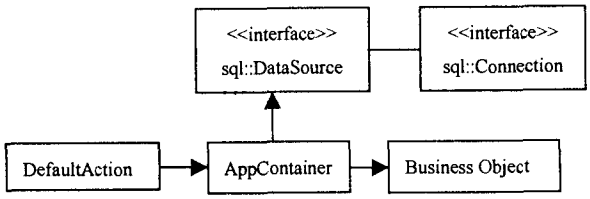

图 2 扩展框架的基本架构

2.3 设值方法注入功能

在扩展框架中,为了完成业务对象所需的逻辑,业务对象所依赖的对象都是在运行期通过 AppContainer 注入的。AppContainer 类实现了设值方法注入功能。在 XML 配置文件(AppContainer - config. xml)中配置好需要注入的对象,使该对象在整个 Web 应用中有惟一的一个标识符,接着在 AppContainer 类中解析该 XML 配置文件。AppContainer 类中的 injectProperties()方法负责将配置文件中配置的对象赋给业务对象的一个字段,该字段和被注入的对象有相同的标识符,并且业务对象中存在该字段的 setter 方法,以确保注入成功。setter 方法在对象实例被容器构造之后立即执行,且在任何业务方法调用之前。

AppContainer 的设值方法注入下面 3 种形式。

2.3.1 ActionForm Bean 的注入

框架将所有的用户请求都转发给 DefaultAction 类,DefaultAction 类决定了所要调用的业务对象的业务方法。对于某个用户请求,如果为对应于该请求的 Action 配置了 ActionForm Bean,AppContainer 直接将该 Bean 注入到业务对象中(该业务对象中存在该 Bean 的 setter 方法),而不需要在 AppContainer 的配置文件中配置关于 ActionForm Bean 的信息。

2.3.2 数据库连接的注入

为了方便对具有数据库操作功能的业务对象进行单元测试,AppContainer 实现了对数据库连接的动态注入。在 AppContainer 的配置文件中配置数据源的信息以及基于此数据源的数据库连接信息。程序运行时,AppContainer 根据该配置文件中的配置信息将该 connection 注入到业务对象中(该业务对象中存在该 connection 的 setter 方法)。在对该业务对象进行单元测试时,可以在测试类中构造假的 connection,通过 setter 方法注入到业务对象中,而不需要进行真实的数据库连接。这样,AppContainer 将业务对象和 Web 服务器割离开来,即使在 Web 服务器没有启动的情况下,也能够进行业务对象的测试,确保业务对象的测试容易性。

2.3.3 一般对象的注入

对于位于 request 或者 session 范围内的一般对象,AppContainer 根据配置文件中的配置信息将该对象注入到业务对象中(该业务对象中存在该对象的 setter 方法)。一般对象注入的方法和数据库连接的注入相同。

3 总 结

Dependency Injection 模式体现了一种将调用者与被调用者分离的思想。分析了依赖注入的工作原理和几种实现类型,并且基于 DI 模式对标准的 Struts1.2 框架进行了扩展,在标准框架中加入了设值方法注入功能。我们开发组的一个实际的百货店管理系统(Web)就是基于此扩展框架进行开发的。在开发过程中该框架大大降低了系统的组件之间的耦合,大大简化了对单个组件的单元测试,效果非常明显。但是此扩展框架现在只支持设值方法注入,并不支持构造子注入和接口注入。这些正是下一步的工作方向。

参考文献:

[1] Johnson R. The Spring Framework[EB/OL]. http://www.springframework.org/.2005 - 06 - 15.

[2] Pico Container Organization. The Pico Container Home Page [EB/OL]. http://www.picocontainer.org/.2005 - 06 - 25.

[3] Fowler M. Inversion of Control Containers and the Dependency Injection pattern [EB/OL]. http://martinfowler.com/articles/injection.html,2004 - 01 - 23.

[4] The Apache Software Foundation. The Struts project [EB/OL]. http://struts.apache.org/.2005 - 06 - 10.

[5] 孙卫琴. 精通 Struts:基于 MVC 的 Java Web 设计与开发[M]. 北京:电子工业出版社,2004.191 - 194.

(上接第 37 页)

$$a(0)b(0) \vee a(0)b(1) \vee a(1)b(0) \vee a(1)b(2) \vee a(2)c(0) \vee a(3)b(2) \vee a(3)b(0) \rightarrow 0$$

$$a(1)b(1) \vee a(3)b(1) \vee a(2)c(1) \rightarrow 1$$

5 结束语

最近几年,粗糙集理论由于在数据挖掘方面的应用而倍受关注。粗糙集理论的应用领域包括知识约简与规则提取,以及决策分析等方面。基于粗糙集理论的数据约简是属性选择的重要手段。基于粗糙集的知识发现是数据挖掘的一个新的方法,挖掘出的知识具有便于理解、表达、存储和使用等特点。

参考文献:

[1] Pawlak Z. Rough set[J]. International Journal of Information and Computer Science,1982,11(5):341 - 356.

[2] Pawlak Z,Grzymals,Slowinski R. Rough sets[J]. Communications of the ACM,1995,38(11):89 - 95.

[3] 张文修,吴伟志,梁吉业,等. 粗糙集理论与方法[M]. 北京:科学出版社,2001.

[4] 刘 清. Rough 集及 Rough 推理[M]. 北京:科学出版社,2001.

[5] 史忠植. 知识发现[M]. 北京:清华大学出版社,2002.147 - 148.