高内聚低耦合软件架构的构建①

Construction of High Coheres and Low Coupling Software Architecture

程春蕊 刘万军 (辽宁工程技术大学 电子与信息工程学院 辽宁 葫芦岛 125105)

摘 要: 论文从现代软件系统设计的趋势出发，引出架构设计及其目标：设计出“高内聚，低耦合” 的应用系统。就应用成熟开源框架 Struts，Spring 和 Hibernate 进行架构设计如何做到高内聚、低耦合展开分析，得出结论。

关键词: 架构设计 高内聚 低耦合

Donald E.Knuth 在其历史性经典巨著 The Art of Computer Programming 中提出关于程序结构的描述：“程序＝数据结构＋算法”[1]。现代软件系统的设计不仅仅停留在满足对算法和数据结构的选择方面，而是更多的对软件系统的设计思想和原则、设计方法、系统架构及体系结构、设计模式、运行平台、开发工具等方面进行综合考虑和合理取舍。

要想设计开发一个高水平、高质量的软件，最重要的工作就是系统的架构设计，因为一个好的架构能够使软件易于维护、易于扩展、易于适应变更，且能够最大化的重用。

1 架构设计的目标

架构(Architecture)：指可以预制和重构的软件框架结构[2]。普遍指通过某种特定平台，而达到完成整体软件的功能。

架构设计是指对软件、硬件、网络、运营、政策等软件设计中的需求和要素进行决策，主要包括体系结构设计和各个层的模块设计。架构设计目标有 3 个[2]：

(1) 能够最大化的重用

首先，要在架构的设计中灵活地使用各种共享的，特别是开源的框架技术；因为共享的架构可以方便开发组分解问题，从而对项目中的功能模块分为需要内部解决和使用已有外部服务两类，避免了重复开发实现。其次，尽量使用成熟的框架。由于服务器端软件系统的开发，涉及的知识、内容、要解决的技术问题很多，在某些方面使用第三方成熟的框架，相当于让别人帮助开发者完成了一些基础性的工作，此时开发者只需要集中精力完成系统业务逻辑的设计和实现。

(2) 使软件系统实现可扩展性

在技术上灵活地使用各种架构模式和代码设计模式，并且在使用代码设计模式的同时，使用其所提倡的面向接口编程，会对软件系统的可扩展性和可移植性的提高有所帮助。

(3) 希望能够设计出“高内聚、低耦合”的应用系统

这是架构设计最主要的目标，也是本论文将重点研究的问题。实现系统的高内聚、低耦合遵从以下原则：

①利用分层架构实现系统在纵向上的低耦合

②利用开源框架进一步确保纵向分层的具体实现

③按照功能划分子系统来实现横向上的低耦合

④利用包结构确保横向上低耦合的具体实现

2 架构模式及框架的选择

总结架构设计的目标，要设计一个好的架构，作者首先要做的工作是对架构模式和实现此模式的框架的选择。因为选择一个好的架构模式，并在此基础上选择现有成熟的开源框架是设计出“高内聚、低耦合”应用系统的基础。

对于开发有大量用户界面，并且业务逻辑复杂的大型应用软件，通过合理地应用 MVC 架构模式和层

① 收稿时间:2008-11-09

体系架构模式(Layers Architecture Pattern)[3]来进行系统设计，将会使软件在健壮性、代码重用和逻辑结构方面上一个新的台阶。在 J2EE 平台上，有很多实现了 MVC 架构模式的框架，利用这样的框架，很容易搭建模型、视图、控制器各司其职的系统，但是这些框架大多都只是提供了 MVC 的部分解决方法，无法实现模型、视图、控制器等所有的组件。设计基于 J2EE 多层体系结构的 Web 整体开发架构时，实现各层功能的框架的选择是开发中的重点。

目前流行的表示层开源框架有 Struts、JSF/Tapestry。前者只是单纯的 MVC 模式框架[4]，后者则是一种事件驱动型的组件模型。构建一个成功的表示层，比较的就是框架的易用性，在类似的功能条件下，最佳选择就是简单易学的框架。Struts 在 Java Web 层的标准化、规范化、学习的简单性，以及技术的成熟性使它成为我们开发应用的首选。

业务逻辑层轻量级解决方案包括 Spring、Hivemind 等框架，目前使用最广泛的是 Spring。Spring 是一个轻量级的控制反转和面向切面的容器框架，是为了解决企业应用开发的复杂性而创建的。它的出现使得以前只可能由 EJB 完成的事情现在使用基本的 JavaBeans 来完成变得可能，而且它提供了对 ORM 技术的支持。

持久层的实现框架主要有 Hibernate、JDO 以及 iBatis。Hibernate 是轻量级的开源框架，提供了将 Java 中的对象与对象关系映射至关系数据库中的表格与表格关系的自动对应转换方案，任何使用 JDBC 的场合都可以采用 Hibernate 来完成数据持久。能够使不懂 SQL 的人也进行数据库的操作是它最优秀的地方所在。

3　高内聚低耦合的实现

选择 MVC 架构模式和层体系架构模式，使用 struts-spring-hibernate 构建应用系统，是实现“高内聚、低耦合”架构设计主要目标的基础，如何才能够真正地实现该设计目标呢？下面，本文以 Struts-Spring-Hibernate 的整合架构为例，对实现系统的高内聚、低耦合主要遵从的纵向原则以及横向原则进行分析。

3.1 纵向原则分析

纵向分层的目的是为了实现表示逻辑、控制逻辑、业务逻辑和数据访问逻辑的分离和层与层之间的松散耦合。相邻的层次之间，采用面向接口编程的思想，使系统易于维护和升级。在分层设计的时候，将应用系统划分为表示层、控制层、业务逻辑层、服务层和数据访问层 5 个层次。

表示层+控制层采用 Struts 框架来实现。原因如下：

・应用广泛，容易找到现成的开源系统范例以供使用。

・提供了丰富的标签技术，能大大提高软件开发效率。

・提供了异常处理机制和数据库连接池管理，使得程序设计工作减少。

・采用前端控制器的架构模式，使得表示层与控制层之间实现松散耦合。

业务逻辑层采用 Spring 框架来实现。原因如下：

借助 Spring 框架中的面向切面编程(AOP)技术，为项目提供各种形式的“拦截器”通知组件，以分离系统中的技术关注点[5]。控制反转(IoC)主要采用了模板模式[6]的设计思想，使得在程序设计时不必再为系统的控制人为地调用系统类库，而只需实现系统定义的方法或是补充适量的自定义方法即可。即把控制权交给了框架，减小了开发的工作量，同时实现了非侵入。

服务层＋数据访问层采用 DAO 设计模式和 Hibernate 框架来实现。

Hibernate 是一个开源的对象-关系映射框架的解决方案，简称为 ORM[7]。其对 JDBC 进行了非常轻量级的对象封装，使得开发者可以随心所欲的使用面向对象编程思想来操纵数据库。同时它又实现了数据缓存和事务管理等，简化了程序开发过程。利用 DAO 模式，主要是对 Hibernate 的进一步封装。用 DAO 实现服务层的功能可以避免业务逻辑处理层中直接调用有关数据访问的操作，实现业务逻辑层与数据访问层之间的松散耦合。

SSH 整合框架的工作流程如图 1 所示，整合方式论文不予论述，在系统架构设计中如何合理地进行分层，层与层之间如何进行连接都是很重要的设计工作。如下系统处理流程图从总体上描述了各个层之间的调用关系，除了表示工作流程规范外，还表示出了各个层相应的命名规范，各层的每个实现类都遵照该命名规范来命名。

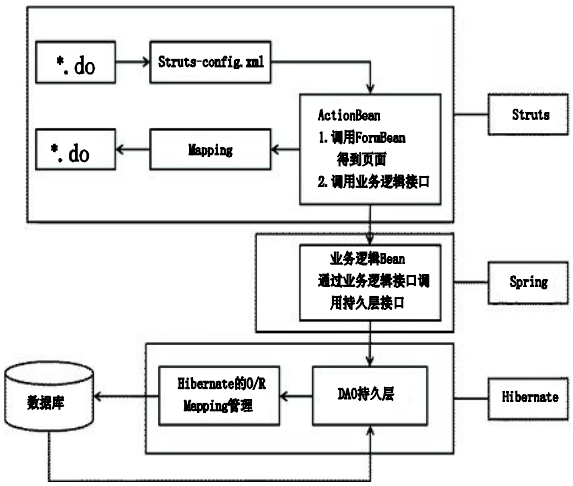

图 1　SSH 整合工作流程

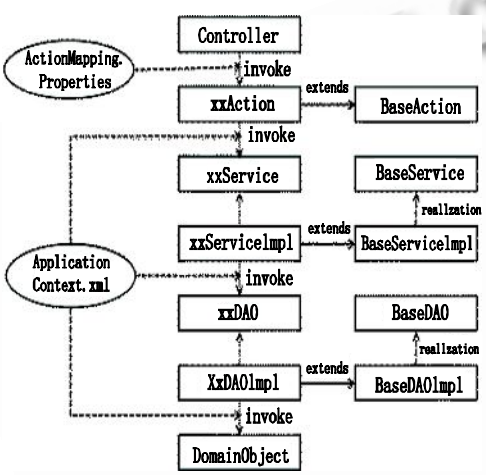

图 2　系统实现工作流程

系统架构设计决定软件系统各部分之间的关系，论文利用集成架构中各个层之间的关系对系统进行分层设计。域模型层对各个模块抽象出的实体进行对象封装，并将其与数据库表的记录相对应，它的类的设计直接体现在业务层和持久层。下面对图 2 中所示的调用关系进行说明。

(1) 写 DAO 时不直接使用 hibernate 或 spring 对 hibernate 的支持。

直接继承 spring 对 hibernate 的封装类 HibernateDaoSupport，使得开发的代码不得不依赖于它们的某个版本。版本升级时 DAO 中所有对 spring-hibernate 的引用都必须修改。如图所示，接口 DAO 中定义了一个父类接口 BaseDAO 以及具体 DAO 接口。这样，接口实现类 DAOImpl 中就有 BaseDAO 接口的实现类 BaseDAOImpl 以及具体的接口实现类 XxDAOImpl，具体的接口实现类 XxDAOImpl 都继承 BaseDAOImpl。在 BaseDAOImpl 中定义了一些通用的方法，如信息报错的方法，这样它的子类 XxDAOImpl 就拥有了这些方法，不必再多写代码实现。XxDAOImpl 类中都定义了相关的对数据库的操作。同时 XxDAOImpl 将与 Domain Object 进行交互，达到操作数据库的目的。

(2) 当业务层需要获取别的模块的数据时，不直接使用该模块的 DAO 和 DAO 类的设计相似，Service 接口中也定义了一个父类接口 BaseService。这样，接口实现类 ServiceImpl 中就有 BaseService 接口的实现类 BaseServiceImpl 以及各个具体的接口实现类 XxServiceImpl，具体的接口实现类 XxServiceImpl 都继承 BaseServiceImpl。在 BaseServiceImpl 中，也定义了一些通用的方法，使得它的子类 XxServiceImpl 也就拥有了这些方法。同样的，不必再多写代码实现。类 XxServiceImpl 将与持久层的多个 XxDAO 接口进行交互。

(3) 写 Action 时不直接使用 spring 和 spring 的继承类

由于 Action 通常不纳入 spring 的管理，因此 Action 在通过 spring 调用某个业务逻辑时，往往是引用一个叫 Spring Context 的类，然后使用它的 getBean()方法。如此的使用，我们的 Action 将依赖与 spring。如图所示，所有的 Action 都继承 Base Action 类。Base Action 封装了控制层公共的信息，如 Action 中跳转到目标页面的方法 execute()。这样，其它的 Action 类就不用考虑公共信息，只需要通过 BaceAction 就可以得到。具体的 Action 通过读取配置文件调用相关 Service 接口的方法，达到和业务层交互的目的。一个 Action 可以调用多个 XxService。XxAction 使用 XxFrom，而 XxForm 继承 Action-Form 类，他们实现了对客户端的表单数据的良好封装和支持。

3.2 横向原则分析

按照业务逻辑进行包的划分。在每个包的内部，同样采用分层的思想，针对接口编程以实现松散耦合。同时再为每个包提供一个或者多个接口，以实现横向上的低耦合；在包内充分应用封装策略，实现复杂性内聚[2]。如图 3 所示。

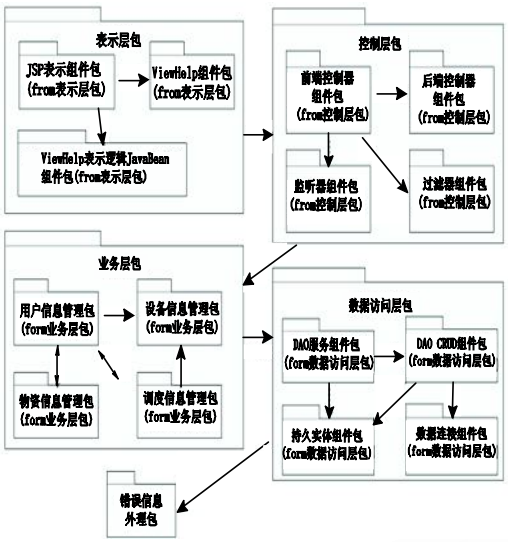

图 3　项目系统架构包图

包图是保持系统整体结构简明、清晰的重要工具。图中业务层中的包是本架构的目标系统 Xx 煤矿企业管理信息系统的部分包图。其中的设备管理子系统源码所在的包以及其包括内容如下。系统所有子系统都如此设计。

com.dhms.equ.form----Struts 的 Action-Form 类

com.dhms.equ.action----Struts 的 Aciton 类

com.dhms.equ.dao----Hibernate 的持久化类接口

com.dhms.equ.daoimpl----Hibernate 的持久化类接口的实现

com.dhms.equ.service----Spring 的业务类接口

com.dhms.equ.serviceimpl----Spring 的业务类接口的实现

com.dhms.equ.po---- 域对象，包含 mappings 文件，即 PO 的映射文件*.hbm.xml。

com.dhms.equ.vo----值对象，负责业务层的数据传输。

目标系统按系统功能的聚集关系进行划分，将不同的子系统放到不同的目录下实现系统横向上的划分。设计好子系统之间调用的接口，各子系统面向接口编程，实现松散耦合。

在此架构的指导下，目标系统的开发在短时间内顺利完成。系统采用分层的设计方式，面向接口编程；各个模块功能相互独立封装，层与层之间关联少，保持松耦合连接，系统的稳定性高，便于扩展和维护。

另外，项目中的每一层所采用的技术都可以替换，在每个层中都不同程度地应用了 J2EE 平台中常用的设计模式，从而使得系统易于测试、便于移植。

4　结论

软件开发专家 Alistair Cockburn 在《敏捷软件开发》中说过，软件在整个生命周期中变更是无时无刻不发生的[8]。作者认为，软件的变更一方面是技术的更新，系统应当不太依赖某个具体的技术或框架。另一方面，用户的需求是动态变化的。一句经典的描述："当我看到时我的需求就变更了。"

建立低耦合、高内聚的软件结构是敏捷开发提出的应对用户变更的办法之一。软件系统架构设计的依据是基于对系统需求的满足，用户需求的复杂多样以及其动态的变化，要求软件系统的架构设计随着系统的需求变化而不断地完善系统架构设计，这样才有可能实现高内聚低耦合的设计目标。

参考文献

1 Knuth DE. The Art of Computer Programming. 5th ed Addison-Wesley, 2005.

2 杨少波,卢苇.J2EE 项目实训－UML 及设计模式.北京:清华大学出版社, 2008:74－114.

3 谢新华.软件架构设计的思想与模式.北京:中科院计算所培训中心:77－79.

4 孙卫琴.精通 STRUTS:基于 MVC 的 JAVA WEB 设计与开发.第二版.北京:电子工业出版社, 2006.

5 郭峰.Java 开发利器:Spring 从入门到精通.北京:清华大学出版社, 2006.

6 莫勇腾.深入浅出设计模式.北京:清华大学出版社, 2006:80－81.

7 王国辉,马文强.Hibernate 应用开发完全手册.北京:人民邮电出版社, 2008.

8 Martin RC.敏捷软件开发(影印版).第二版.北京:中国电力出版社, 2003.