Software Engineering and Applications 软件工程与应用, 2012, 1, 24-30
http://dx.doi.org/10.12677/sea.2012.12005 Published Online December 2012 (http://www.hanspub.org/journal/sea.html)

Template Designation and Implementation of the J2EE Architecture

Yuanpeng Sun[1], Wenyu Chen[2], Yongqiang Wu[2], Weishun Li[2], Lingli Guo[2]

[1]Shenzhen Zhenhua Microelectronics Co., Ltd., Shenzhen
[2]School of Computer Science and Engineering, University of Electronic Science & Technology of China, Chengdu
Email: sunyp@zhm.com.cn, cwy@uestc.edu.cn

Received: Oct. 31st, 2012; revised: Nov. 15th, 2012; accepted: Nov. 19th, 2012

Abstract: On the basis of the lightweight layered architecture of the traditional J2EE, a template of the operating module is designed by using generic and reflection in the business and data access layer, providing a common operation logic to the module of specific domain object. Through this template designation, it not only can avoid the repeated writing the similar logic of each domain module, but also is conducive to the robustness, maintainability of the system, and flexibility to the changes of whole business needs. Template structure can improve and perfect the traditional J2EE architecture based on the open source framework. Utilizing the architecture in the application whose business logic is relatively simple and single has some guiding significance to the J2EE development.

Keywords: J2EE; Lightweight Framework; Generic

J2EE 架构的模板化设计与实现

孙元鹏[1]，陈文宇[2]，吴永强[2]，李维顺[2]，郭凌立[2]

[1]深圳市振华微电子有限公司，深圳
[2]电子科技大学计算机学院，成都
Email: sunyp@zhm.com.cn, cwy@uestc.edu.cn

收稿日期：2012 年 10 月 31 日；修回日期：2012 年 11 月 15 日；录用日期：2012 年 11 月 19 日

摘　要：在传统轻量级 J2EE 分层架构的基础上，利用泛型与反射技术在业务层和数据访问层设计模版化的操作模块，可以为其具体领域对象模块提供公共操作逻辑。本文提出模版化构架方式，不仅避免了每个领域模块相似逻辑的重复编写，提高开发效率，而且统一的操作策略增强了应用的健壮性、可维护性、能灵活应对需求的总体变动。模版化构架方式，对改进与完善基于开源框架的 J2EE 架构进行了一定的探索，应用此种架构对业务逻辑相对单一简单的应用开发具有一定的指导意义。

关键词：J2EE；轻量级框架；泛型

1. 引言

在全球信息化、数字化建设过程中，J2EE 的多层架构方式已成为企业级应用开发的标准平台，但由于传统 J2EE 的复杂性和侵入式设计，开源框架的兴起近几年成了其代替方案。在经典的轻量级框架 Struts2 + Spring + Hibernate 的 J2EE 架构中，由于其分层架构和 MVC 模式[1]思想，使应用具有松耦合、易维护、低成本等特点而广泛用于中小型企业开发中。但是，此架构方式往往少不了在数据访问层中各个域模型增删改查方法的反复声明与实现和在业务层中各域模型的相似逻辑功能加工处理的重复操作，其结果是代码显得冗余混杂，且分散的操作策略容易引起程序异常，影响开发进度且不便于后期的管理维护。

针对这些问题，本文提出了在数据访问层应用泛

型 Dao 模式和在业务层应用泛型 Service 模式的改进方案，通过模版化的架构方式，不仅能简化了开发过程，提高了开发过效率，而且模版化的操作策略为应用的扩展与集成提供了便利。

2. 相关技术

层架构方式中，整个应用分为三层：Web 表现层、业务逻辑层、数据访问层[2]，应用分层构架方式可使各层间分工明确，功能不相互干扰。层之间通过暴露的接口进行通信，使方法功能的改变只用修改层内具体实现而无需改动调用层的代码。这种面向接口的开发方式，有利于应用并行开发，又可以降低各层组件间的耦合性，增强系统的可移植性、复用性、维护性和扩展性[3]。

经典的 SSH 架构方式中，Struts2 作为 Web 表现层框架，其简化了基于 MVC 模式的 Web 应用开发，主要职责为拦截管理用户请求、执行 UI 数据校验、业务控制器通过 Façade 模式调用业务组件生成视图层待用的值对象等。Spring 作为业务逻辑层框架，应用其 Ioc 容器来管理和装配所有对象，能有效组织系统各组件，提高组件的模块化，降低组件间的耦合度；其 AOP 的开发思想，有利于业务逻辑与公共管理逻辑的分离，尤其是在事务管理上具有得天独厚的优势，其主要职责为：为控制层业务控制器提供统一接口，处理业务校验和业务逻辑、管理对象间的依赖关系、拦截业务方法执行进行事务管理、提供控制层上下文而获得对业务托管服务等。以 Hibernate 为数据访问层框架，现实 ORM 映射机制，通过对 JDBC 的轻量级封装，使开发人员可以以一贯的面向对象的编程思想来操作关系型数据库，其主要职责为：为业务层提供统一的数据操作接口支持，通 HQL、QBC、QBE 的方式或者 Session 对象的 API 进行查询、存储、更新、删除数据库中信息，对数据库操作进行性能优化等。

如图 1 所示传统的 SSH 架构层次图，可见三种框架的集成整合，可以充分利用各框架的优势，搭建快速高效的开发环境[4]。但是在常规的开发中，在数据访问层和业务逻辑层都需开发每个域模型所对应的组件模块，这种情况下势必会产生大量相似或相同的代码，尤其在大型开发中，冗余臃肿的代码往往易隐藏异常、难以维护。

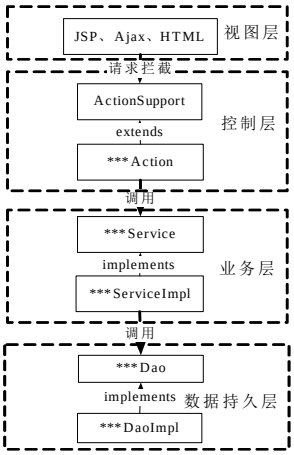

Figure 1. The traditional SSH architecture hierarchy diagram
图 1. 传统 SSH 架构层次图

3. 基于泛型设计方式

模板化架构方式的思想是借助泛型技术[5]，实现依赖类型的参数化，通过把组件依赖类型设为一个类型参数，在使用时才用实际类型予以代替，从而使算法可以不依赖于类型而独立存在，保证一些有着共同需求的处理能跨越类型的限制而有效运行。

3.1. 泛型 Dao

在数据访问层中，应用 Dao 模式[6]来抽象与封装对数据源的操作，通过抽取出一个通用泛型 Dao 组件，使每个 Dao 组件通过继承方式实现其对应域模型实体的 CRUD 方法。整合 Hibernate 框架实现了 Dao 组件对 SQL 的依赖上升为对域类型的依赖，使数据库操作可直接表现为对域模型实体的操作。作为 Dao 组件与数据源间适配器的 Hibernate 框架，其与数据源交互的 Session 对象都是基于 Object 类型的操作，借助泛型技术可统一对操作类型进行约束，并对查询结果实施统一的转换，泛型的编译时类型检查技术可保证操作域类型的合法性和结果类型转换的安全性。

在 Session 查询方法中，需要提供域类型信息作参数。在泛型 Dao 组件中，获取其依赖的域类型的途径常用的解决方案是依赖注入，即在泛型 Dao 组件中声明域类型并在其子类中用 setter 方式注入域类型具体值。但其缺点是在声明子类 Dao 组件时已经对其依赖的域类型参数进行指定，在容器中注册子类 Dao 组件时又注入其依赖域类型的字节码对象，显得是一种重复操作。

本文提出了应用反射技术的域类型获取方式，其核心思想是每次执行数据访问操作总是由具体的子类 Dao 组件实例进行，可通过此子类 Dao 组件的字节码信息即可获取在其在继承泛型 Dao 组件时用来实例化类型参数的域类型，具体代码如下：

```
public <T> Class getRealType(){
    Class<T> clazz = this.getClass();
Type genType = clazz.getGenericSuperclass();
if (!(genType instanceof ParameterizedType)){
    return Object.class;
}
Type[] params = ((ParameterizedType) gen-
Type).getActualTypeArguments();
    if (!(params[0] instanceof Class)){
    return Object.class;
}
return (Class) params[0];
}
```

其具体过程是：在泛型 Dao 组件中，得到运行时对象(子类 Dao 对象)的字节码后，查看其父类是否为泛型类，如果是就得到泛型类型参数的实际类型数组，如果数组的第一个元素是引用类型返回此元素值。

可以看出，基于反射的方式比传统依赖注入的方式更具有优势，它不仅省去了每个子类 Dao 组件的注入配置，降低了泛型 Dao 组件对子类 Dao 组件的侵入性，而且使程序编写更简洁优雅，使泛型 Dao 组件更具有模版化的特性。

由此可构建泛型 Dao 模式如图 2 所示。

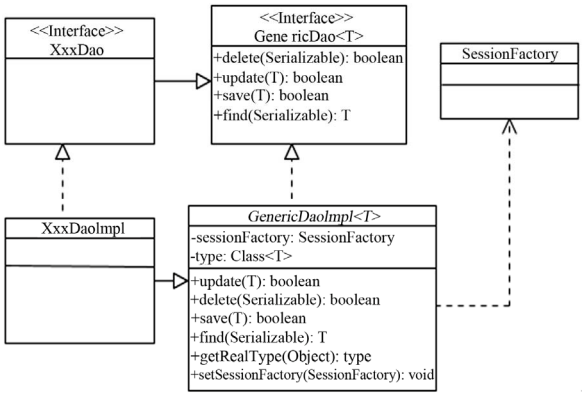

Figure 2. The generic Dao mode UML diagrams
图 2. 泛型 Dao 模式 UML 图

定义了泛型接口 GenericDao，里面规定了最常用的 CRUD 方法，其实现类 GenericDaoImpl 通过 Spring 的依赖注入获得其依赖对象 SessionFactory，通过 Session 对象现实接口中规定的方法。其中最关键的是在实现查找方法时，获取泛型类型参数 T 的实例字节码信息即其 type 属性。在图中 type 是调用 getRealType (this)方法得到，getRealType 方法的思想是每次执行数据访问操作总是由具体的子类 Dao 来进行，通过子类 Dao 字节码信息，反射可得到泛型类型参数的实例。其代码如下：

```
public <T> Class getRealType(Object object){
    //得到对象的字节码
    Class<T> clazz = object.getClass();
//得到泛型父类
Type genType = clazz.getGenericSuperclass();
/* 如果没有实现 ParameterizedType 接口，即不支持泛型，直接返回 Object.class */
if (!(genType instanceof ParameterizedType)){
    return Object.class;
}
//得到类型参数实例的 Type 对象的数组
Type[] params = ((ParameterizedType) gen-
Type).getActualTypeArguments();
    /* 如果类型参数实例不是 Class 对象，直接返回 Object.class */
if (!(params[0] instanceof Class)){
    return Object.class;
}
return (Class) params[0];
}
```

得到泛型类型参数实例后，在 GenericDaoImpl 类中，就可以根据指定的域类型实现接口规定的方法，代码如下：

```
public class GenericDaoImpl<T> implements
GenericDao<T> {
private SessionFactory sessionFactory;
//得到泛型类型参数实例
protected Class<T> type = getRealType(this);
    protected Session session = getSession();
public Session getSession() {
    return sessionFactory.getCurrentSession();
```

```
}
public T find(Serializable entityId) {
        /* 通过 Session 对象的 API 和得到的域类型得到实体域对象 */
    return (T) session.get(this.type, entityId);
}
……
public Boolean delete(Serializable entityId){
    T entity = find(entityId);
    session.delete(entity);
    session.clear();
    return true;
}
}
```

在子类 XxxDaoImpl 只要继承 GenericDaoImpl 并指定泛型类型参数实例即子类 Dao 对应的域类型，子类 Dao 就拥有了对应域模型的数据操作方法，而不用再具体编写每一个操作方法。

public class XxxDaoImpl extends GenericDao-Impl<Xxx> implements XxxDao{}

3.2. 泛型 Service

对一些有着公共业务需求的逻辑操作，且操作只针对单一域模型进行的情况下，可以借助泛型 Dao 的思想抽取出泛型 Service 模块来提供公共业务服务。在泛型 Dao 中基于对域模型类的依赖，而在泛型 Service 中，不仅依赖于域模型类，还依赖于子类 Dao 实例，对这两个依赖类全部参数化可得整个泛型 Service 的依赖图如图 3 所示。

如图 3 所示，在 BaseService 中规定公共的逻辑操作接口，其实现类 BaseServiceImpl 是关键的一步是得到其依赖的类型参数 M 的实例对象(即 XxxDao 对象)，这里也通过反射技术得到，核心思想是每次业务逻辑的执行是通过具体子类 Service 对象进行，而子类域模型 Service 上又必有一个其对应的子类 Dao 成员变量，通过子类 Service 字节码可以得到子类 Dao 实例。

与泛型 Dao 组件不同的是，泛型 Service 组件是对子类 Dao 组件实例的依赖，而泛型 Dao 组件是对域类型字节码的依赖。从字节码到其实例对象间的转换的通常做法是调用 newInstance()方法直接生成得到，但是此方式使编程又回到“单粒”开发中来，不利于实例对象统一管理；通过“依赖拖拽”直接从容器中获取对象的方式虽然遵循了对象工程化对象管理的思想，但是又使 Spring 框架 API 侵入到 POJO[7]内部，违背了轻量级框架的初衷。本文再次应用反射技术，其核心思想是，每次业务逻辑的执行是通过具体子类 Service 组件实例进行，而子类 Service 组件上有其依

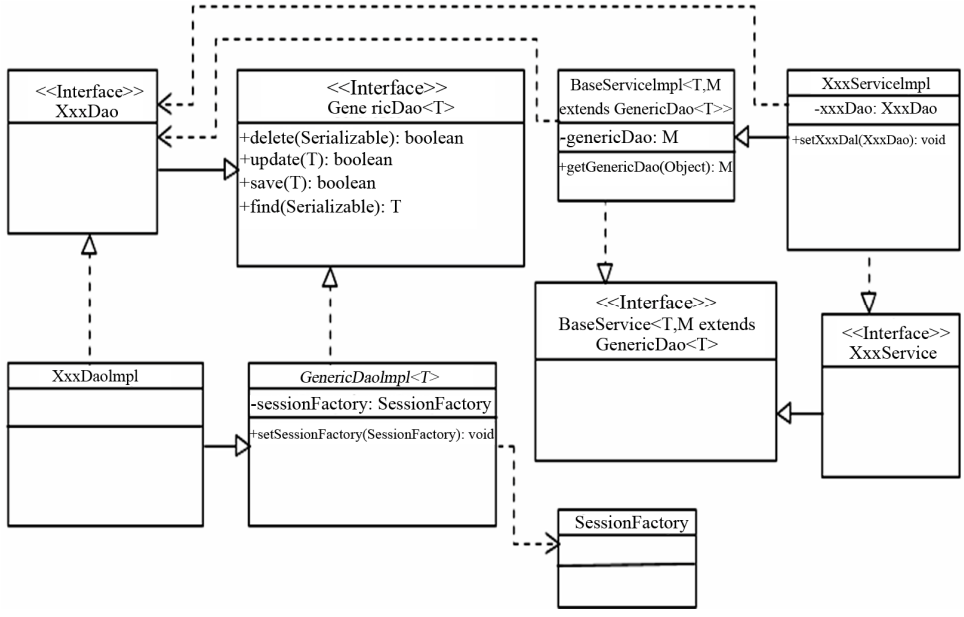

Figure 3. The generic Service mode UML diagrams
图 3. 泛型 Service 模式 UML 图

赖的子类 Dao 成员变量，且变量值由 Spring 框架启动时就注入了进来。

在 BaseServiceImpl 中得 XxxDao 对象的步骤为：

首先，获得泛型 Service 的第二个类型参数实例，其与泛型 Dao 的获得类型参数实例原理一样，只是返回第二个数组值。

其次，在子类 Service 中，查看是否有同一类型的成量变量，如果有就返回成量变量名。代码如下：

```
String fieldName = null;
//得到子类 Service 上声明的全部域对象
Field[] fields = this.getClass().getDeclaredFields();
//遍历声明的域对象数组
for (Field field : fields){
    //域对象与泛型类型参数实例如果为同一类型
if (field.getType().isAssignableFrom(type)){
     fieldName = field.getName();
         break;
     }}
```

最后，根据成员变量的名字，获得子类 Service 里成员变量的对象。

```
Object result = null;
//获取指定名字的域对象
Field field = this.getClass().getDeclaredField(fieldName);
// 保存它的访问标识符
boolean accessible = field.isAccessible();
// 如何为私有，将其设置为可以访问
field.setAccessible(true);
// 获取变量实例值
result = field.get(this);
// 设置访问标识符
field.setAccessible(accessible);
}
```

得到子类 Dao 对象后，就可以根据业务的需要在 BaseServiceImpl 里通过子类 Dao 对象对数据库的操作，实行具体的公共逻辑操作。

在子类 Service 中，只要继承 BaseServiceImpl 并指定泛型类型参数的域模型类型和子类 Dao 类型就能享有公共逻辑模块里的业务操作：

```
public class XxxServiceImpl extends BaseServiceImpl<Xxx,XxxDao> implements XxxService{}
```

3.3. 设计小结

通过参数化类型的方式，使对象之间的依赖关系用类型信息来传递，而非基于传统依赖注入的方式是这种泛型设计的核心。巧用反射根据运行时当前对象(this 对象)的不同，从而得到泛型类型参数不同的实例，使泛型模版产生不同执行效果，是这种设计的思路。

通过利用泛型和抽取出公共的模块的设计方式，从各域模型角度来看，通过继承实现的公共业务逻辑和数据库操作功能，统一的执行过程，省去了人工在每个域模型业务组件和数据访问 Dao 组件中反复声明与实现逻辑相似的方法。而从公共组件的角度来看，只用对泛型类型参数赋以不同的域类型或数据访问类型，模版组件就能对不同域模型进行加式处理，实现了模版的跨域模型服务。

4. J2EE 模版化改进

原来分层框架的基础上，对架构提出改进，分别在业务层和数据持久层提出公共模版组件来对最常用操作进行封装。与传统架构方式一样，整个系统架构遵循面向接口的设计思想[8]，在业务层、数据持久层的组件中都有接口和其具体实现，在业务层中，使用“泛型 Service 模式”，为域模型业务组件提供公共的业务逻辑操作，在数据访问层中，应用“泛型 Dao 模式”，为域模型数据访问组件提供最基本的数据访问操作策略。其结构如图 4 所示。

如图 4 所示，整个系统架构遵循“面向接口”的设计思想，在业务层、数据持久层都有接口和它们对应的实现，在业务层使用“泛型 Service 模式”，为域模型业务实体组件提供公共的业务逻辑操作，在数据访问层中，应用“泛型 Dao 模式”，为域模型数据访问组件提供最基本的数据访问操作策略。

使用此种构架方式可以带来为应用开发带来以下便利：

1) 使开发人员从原本机械重复的编码中解脱出来，直接对泛型 Dao 和泛型 Service 类的继承就使模块拥有了最基本的功能，而使开发人员可以投入更多

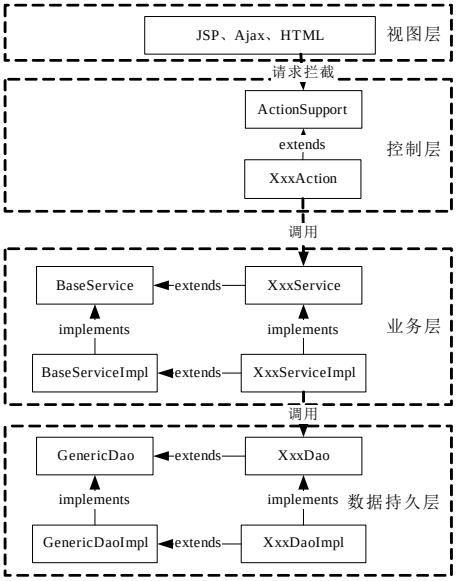

Figure 4. Improved SSH architecture hierarchy diagram
图 4. 改进后 SSH 框架层次图

的精力在其它的复杂逻辑上，减小了开发人员工作量，提高了开发效率。

2) 通过继承方式给补充数据库操作和业务处理逻辑留下了空间，在具体域模型定义与实现过程中，开发人员可以追加自己方法实现，来满足具体不同域模型里的功能需求。

3) 有利于后期需求的总体变动，由于模块化了公共业务和数据操作，使以后因业务需求更变或因数据库的迁移而需改动的代码量大大减小，通过对公共业务模块和数据操作模块的改动可以对系统的整体功能做出调整。

4) 保持了 J2EE 的分层原则，各层功能清晰，结构分明，有利于各模块的分工合作，提高了模块的复用度，便于软件开发维护。

5) 支持传统策略模式多态效果，由于泛型 Service 依赖的是子类 Dao 实例，泛型 Service 可以给不同的子类 Dao 提供服务，因此相同公共业务接口执行，可以得到不同的数据操作执行效果或返回不同的业务方法执行结果。

6) 统一的工程管理思想。所有对象间的依赖关系实际是由 Spring Ioc 容器内注入的，同时所有对象由容器统一管理，避免了类里面任意生成实例化对象。

7) 保证了系统的健壮性和扩展性，基于泛型 Service 统一处理公共业务逻辑，系统架构完毕后，可通过修改表现层参数来改变页面呈现方式，通过数据访问层改动来扩展新增功能应用。而业务层中逻辑实现是封闭起来的，防止人为的篡改业务层核心代码，保证了系统的稳定可靠。

5. 应用对比

利用此架构方式应用在节能监管平台的开发中，该项目中的一个子功能模块是各种能耗情况的查询，其包括总、人均、单位面积的水电气三项能耗情况的查询，最终以是以柱状图和列表的形式呈现在查询网页上。借助此架构方式，用泛型 Dao 模块统一数据查询，用泛型 Service 模块把查询域模型结果统一转换为前台需要的 XML 格式和 Json 格式数据，使整个数据处理都在模版模块中集中处理。此处用 eclipse IDE 进行代码行数统计，下面给出其系统架构改进前后代码行数随功能模块增的加对比情况，如图 5 所示。

架构改进前，应用的源代码量随功能模块的增加主要是由前台页面呈现模块和后台数据处理模块的新增造成的；而改进后，应用就只主要面临着前台代码量的增加。通过图可以看出，随着功能模块的增多，此种构架方式的优越性就越能够突显。

6. 结束语

利用模版化的设计方式对传统的 J2EE 架构进行改进，可以有效的解决相似代码重复编写的问题，提高了软件的模块化程度，使系统具有良好的复用性、扩展性和维护性。同时模版化的一致性操作策略，有利于应对需求总体变动，减少了代码的改动量。本文提出的模版化构架方式对改进与完善开源框架的 J2EE 架构方式有一定的探索，对于这种架构方式在业务逻辑相对单一简单的应用中具有一定的指导意义。

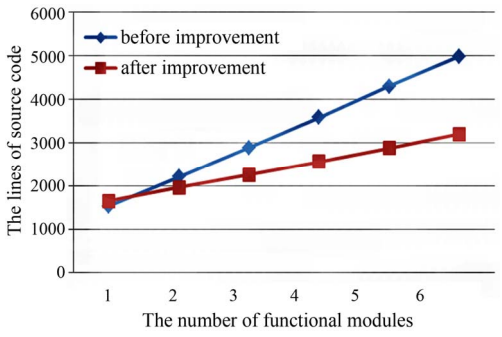

Figure 5. Comparison chart of the number of lines of code before and after the architecture improvements
图 5. 架构改进前后应用代码行数对比图

参考文献 (References)

[1] E. R. Collins, Jr., Y. Huang. A programmable dynamometer for testing rotating machinery using a three-phase induction machine. IEEE Transactions on Energy Conversion, 1994, 9(3): 521-527.

[2] M. Cai, Y. Y. Cheng. Research and application of J2EE platform. Computer Application and Software, 2004, 21(1): 42-43, 128.

[3] X. J. Zhou, X. Zou. Research on J2EE-based enterprise application architecture. Hong Kong: Proceedings of 2010 Ninth International Symposium of Distributed Computing and Applications to Business Engineering and Science (DCABES), August 2010: 431-434.

[4] R. Johnson. J2EE development frameworks. Computer, 2005, 38(1): 107-110.

[5] R. W. Clay, A. Donald and S. Scot. Professional Java JDK (6th edition). Weston: Wiley Publishing, 2007.

[6] Fang Cheng. A new Dao pattern with dynamic extensibility. Manchester: Proceedings of the 2nd International Conference of Information and Computing Science (ICIC'09), May 2009, 1: 23-26.

[7] A. Mukherjee, Z. Tari and P. Bertok. A spring based framework for verification of service composition. Washington DC: Proceedings of 2011 IEEE International Conference of Services Computing (SCC), July 2011: 258-265.

[8] H. L. Pan, W. R. Jiang and S. W. Lin. Research on collaboration software based on JBPM and lightweight J2EE framework. Guangzhou: Proceedings of 2010 International Conference of E-Business and E-Government (ICEE), May 2010: 191-194.