Database accessing middleware based on factory pattern and strategy pattern

Zhaohui Li[1], Hongxia Yang[2*]

[1]*Transportation Management College, Dalian Maritime University, Dalian 116026, Liaoning, China*

[2]*Schoos of Management, Liaoning Normal University, Dalian 116029, Liaoning, China*

Received 1 June 2014, www.cmnt.lv

Abstract

On the detail discussion of database accessing technology, this paper puts forward a database accessing middleware with combination of factory pattern and strategy pattern, and applying this database accessing middleware in the construction of a deli network trades platform. The actual application shows that this proposed middleware contributes to simplification of code fragment. Moreover, it enhances system extensibility and maintainability through the Factory and Strategy design pattern, and makes data processing more flexible, easier to modify and reuse.

Keywords: database accessing middleware, trading platform, factory pattern, strategy pattern

1 Introduction

Traditional database access is more mature in technology, but there are also the following two questions: one is inconvenient to transplant in heterogeneous databases; the other is using complex, so its requirements is high for programmers [1]. Therefore, interface technology is widely applied in the database access technology, such as Hibernate. Hibernate is an open source object relation mapping framework, which is a very lightweight object encapsulation. It mainly use Hibernate API to access database, on the other hand, Struts use the JDBC to access the database [2]. The combination of Hibernate with Struts2 needs to add a middleware.

Currently, the main technologies about database middleware are to modify the XML configuration file, and get the components according to the component ID in the XML file. If the system contains a large number of ID components, it will be complex. Through detailed analysis of database accessing processes, we found that there are mainly two kinds of solutions to solve the different data sources. One is to use middleware, and other is to use XML DTD or RDF as a model [3]. According to the specific environment, this paper designs and implements a database access middleware based on factory pattern and strategy pattern to make data accessing interface easily to modify and adapt.

2 Background

Dalian Deli Trading Centre is the key livelihood project established by Dalian Municipal Government in 1998, and also the carrier to start up food safety project. It implements "Market links together with supply place", "Market links together with supply factory", established strict market access system, certificate and invoice requiring system, inspection and test system and product quality traceability system, which effectively ensures the food quality safety. The Cooked Foods Trading system is based on the combination of Struts2 and Hibernate, as shown in Figure 1. There are huge amounts of trading data need to be stored in database.

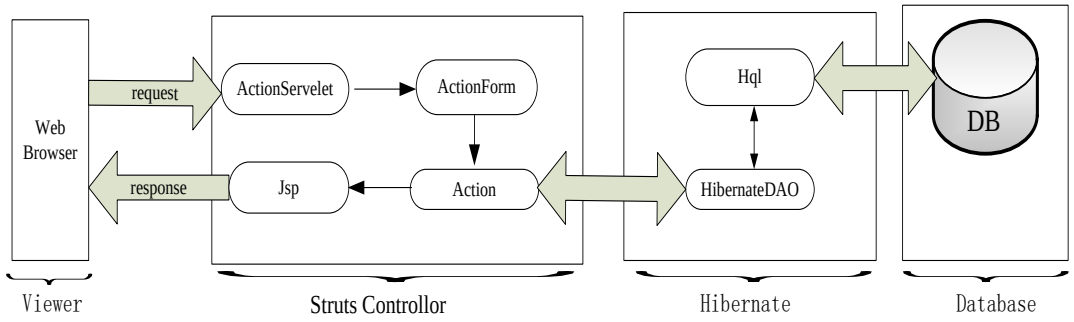

FIGURE 1 Application structure of Dalian cooked foods trading centre

* ***Corresponding author's*** e-mail: yhxseasky@163.com

In the system, many business entity objects are often used as memory objects. Some of operations save store the business entity object to the database, others transmit the data from the database to control layer. As shown in Figure 2, each Action identified in struts-config.xml, and Action class implements a portion of a Web application and returns an object, when the user queries various coding tables, it has to be built a DAO (Data Access Object) to call the method, at the same time, some operations also need create code lists DAO.

The query process is easy to make mistakes because of overmuch coding tables and the corresponding coding table data access objects. But all the operations are database query operations which return a domain object after querying. Moreover, there are very little attributes in the coding tables, and most of the query process is similar. According to the above reasons, this paper puts forward the coding tables DAO get together through middleware technology, optimizing the inquiring process, improving system expansibility.

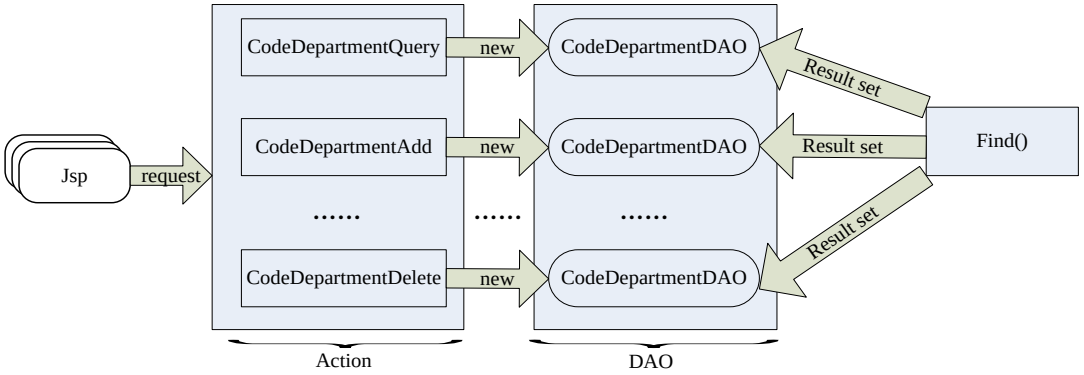

FIGURE 2 Traditional query process

3 Design of database accessing middleware

Middleware is an application program interface, it is also an expandable code stored in servers in the network system. It has the function to provide connectivity between application client and database server, which make programmer avoid various communication protocols and interfaces. Because the middleware is standard programming interface and protocol, data sharing and data manipulation can be achieved in different hardware and operating system platforms. According to the different function [4, 5], middleware can be divided into four categories: Database Access middleware, Message Oriented Middleware, Transaction Processing Middleware and Object-oriented middleware

Among the Object-oriented technology, encapsulation and inheritance of object can provide a very good foundation for software reusability, object-oriented transparency also meet to the requirement of middleware technology, so object-oriented middleware technology have developed rapidly. Its basic idea is to provide a unified interface among objects that is the middleware, which makes the objects calling and data sharing do not to concern with the position of the object, language and the resident operating system, but unified treatment by the interface (middleware) [6, 7].

This paper puts forward a database access middleware between Action and DAO. The main role is to package the function and provide the interface for users to query. The variable should follow the interface that is defined by abstract classes, not statements it as specific class instance. As long as the users know the interface, users don't know the specific type of using object, manipulating objects is according to the interface of the abstract class definition. The users only know which abstract class defines interfaces, without concerning the system how to achieve, and it can greatly reduce interdependence among the subsystems. In order to make the middleware play the real effect, and achieve the real decoupling between Action and DAO layer. Package selecting and creating objects in the middleware, which is not directly in the Action. So Action just needs to call the interface layer. As for middleware, Action need not know how to choose, how to create. And for the data access layer Hibernate, it does not change.

This design pattern can make two frames to transfer data as much as possible, but not to know each other's existence, and carry out their duties. The display layer JSF simply do know by Hibernate persistence layer system is responsible for the display layer for Hibernate, JSF framework also unimportant. In an integrated system, the transparency between the various frameworks determines the system modifications convenience.

The core function of middleware is focused on the core class. Core class is to realize the selection and creating of object, and then it calls the specific class to query. Since there are multiple coding tables exist in the specific class, it will abstract a class through inheritance and rewriting methods. For the behaviour of creating, it is based on the parameters that JSP page sends to the Action, and then creates objects. If it need increase new objects, which only need expand interface layer, add new objects in it. For the called method, if need to increase, it can also add them in abstract classes. And then it completes to rewrite in the specific class. The application structure is as showed in Figure 3.

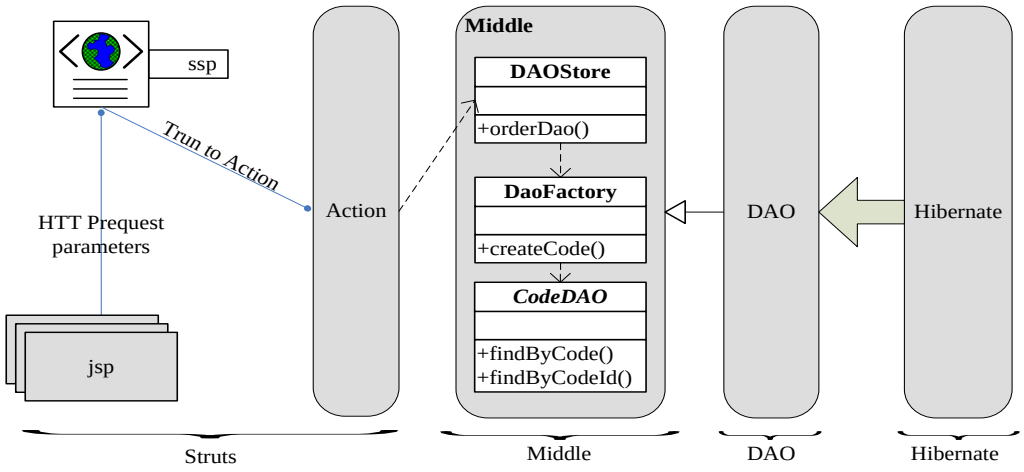

FIGURE 3. Structure of Database Accessing Middleware

3.1 APPLICATION OF FACTORY PATTERN

The factory pattern refers to two distinct design patterns, both first described by the "Gang of Four" [8]. The "abstract factory" pattern provides an interface with which a client can obtain instances of classes conforming to a particular interface or protocol without having to know precisely what class they are obtaining. This has advantages for encapsulation and code reuse, since implementations can be modified without necessitating any changes to client code. Factories can also be used to closely manage the allocation and initialization process, since a factory need not necessarily allocate a new object each time it is asked for one. To obtain an instance, a programmer would first obtain a reference to one of the hidden factory subclasses, usually through a factory method in the abstract factory superclass, and then use that reference to create an object of the product type. The "factory method" pattern is related but simpler: like the abstract factory pattern, the factory method pattern allows a client to obtain objects of an unknown class that implement a particular interface. Rather than relying on a separate factory class to create instances of the product classes, the product class itself has a factory method that returns an object that conforms to the interface defined by that class. Typically, a class implementing a factory method pattern would be an abstract class with several concrete subclasses [9].

The class diagram of Database Accessing Middleware is as shown in Figure 4. The factory pattern provides an interface to create different instances of objects; these objects do not need to develop specific class. It obtains a special class, which is responsible for instantiating a specific object. Usually the class called factory class, which encapsulates the related object's constructor logic, separates subclass object constructor definition, and makes the maintainability of software product greatly enhanced. The user does not need to know how would they do, only need to know the interface.

The factory pattern is equivalent to create instances of objects of the new, it often creates the instance object according to Class, such as A a=new A(). Factory pattern is used to create an instance object, that is a common pattern for creation of polymorphic objects of different concrete types so when creating many objects, the paper considers to use the factory pattern, although doing so many more work, but gives a system to bring greater extensibility and to minimize the amount of modification.

As shown in Figure 4, **DAOStore** class function as data access interface, it provides the definition for object and query processing. But creating object is achieved by **DaoFactory** class, and querying is achieved by **CodeDAO** class (an abstract class defined many methods to rewrite) and its subclasses. The createCode method of **DaoFactory** class used parameters which are transmitted by JSP to determine and create corresponding data access object, and then return query result through CodeDAO method. By means of this query processing, external object can achieve data access objects via creating instance of DAOStore class, and code errors can be reduces through object encapsulation. This separation of specific classes not only help the users to access the database through the unified interface provided by the abstract factory, but also help the users to access among different tables. If the users want to increase coding tables, what would do is just to add an object class in **DaoFactory** class.

3.2 APPLICATION OF STRATEGY PATTERN

The Strategy pattern enables the use and interchange of different algorithms or implementations for a certain policy. It prescribes the definition of an abstract type (abstract class or interface), representing the policy contract (Strategy), and a series of different concrete subtypes (Concrete Strategies) that correspond to alternative implementations of the policy [8]. On the other hand, the State design pattern allows an object to alter its behaviour as a result of changes to its state. The object

(instance of the Context class) appears to its clients as changing its class at runtime. The State pattern's realization is based on the introduction of an abstract type (usually abstract class) that represents an Abstract State and defines methods corresponding to state-dependent operations of the Context class. Each discrete Context state is mapped to a concrete subtype of Abstract State (Concrete State) that provides the state-specific implementation of Abstract State's methods and controls transitions to appropriate target states [10].

The query method of coding tables is very similar, they are all realized query processes through one or more attributes. If these query processes are implemented by a series instances of Action object. This would not be able to reach the complete encapsulation of middleware. So as to encapsulate specific method, it would be needed to abstract the specific method to be a class. On the other hand, if these specific methods are defined in CodeDAO class, the CodeDAO class would be more complex and burden too much. Aiming to these application requirements, the strategy pattern should be adopted to realize the real separation of functions, and then to realize the maintainability, expansibility and reusability of system. Through the strategy factory, the query processes of coding tables are encapsulated, and algorithms of query processes can be replaced each other. Thus, the replacement of algorithms would not influence the users.

As showed in Figure 4, this strategy pattern has three parts. The first is Context role. It is named DAOStore class, which holds specific applications for the users. The second is abstract strategy role. It is named CodeDAO, which usually is implemented by an interface or abstract class and give the requirement interfaces of all concrete strategy classes. The third is concrete strategy. It encapsulated algorithms and methods, which includes CodeStorageTypeDAO, CodeTestResultDAO, Code-DepartmentDAO, CodePruductMajorDAO, CodeProduct-MinorDAO and so on. A series of alternative algorithms and methods are defined in CodeDAO class. Then all the DAO which implement specific coding tables inherited from CodeDAO class, and inheritance contribute to abstract public functions of these algorithms. Through different numbers of parameters, methods can be overwritten, and the subclasses decide which method is chosen. At the same time, all algorithms are classified. The classes derived from CodeDAO define and implement various kinds of algorithms and methods. Through inheritance and derivation, the public method is abstracted and defined, at the same time, various kinds of algorithms and methods are encapsulated in the strategy factory. Consequently, the choosing of algorithms and methods are independence from the changing requirements, which contributes to the difficult problems of program modification.

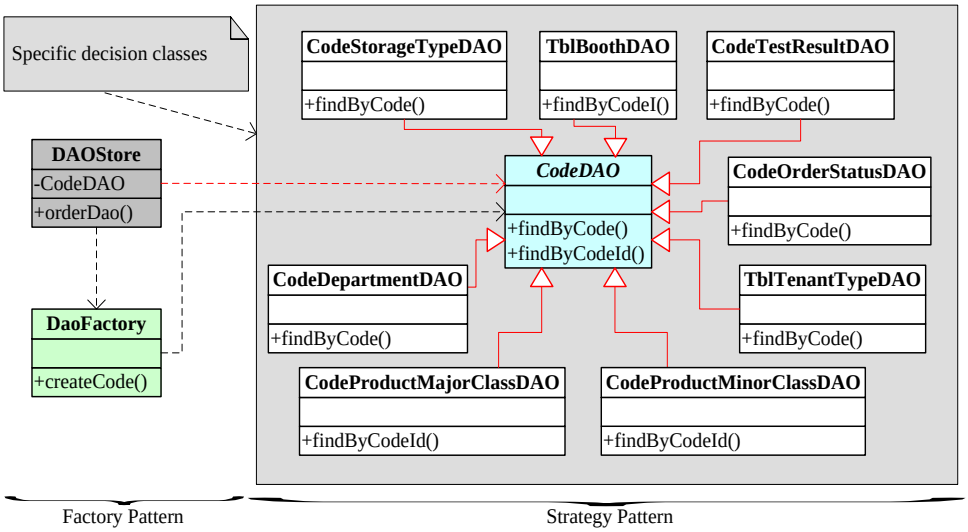

FIGURE 4 Class diagram of database accessing middleware

4 Design of database accessing middleware

In the Deli Trading system, the main operation is to access the database. After using the middleware, when querying a coding table, the user can create the DAOStore object in QueryAction class firstly, and then use the created object to call methods. For example, when querying the department's code table, the code fragment in CodeDepartmentQuery is as following in Figure 5.

```
DAOStore dao=new DAOStore(null);
List<CodeDepartment>codeDeptList=new
   ArrayList<CodeDepartment>();
CodeDeptList=(List<CodeDepartment>)dao.order
   Dao(department,departmentId);
```

FIGURE 5 Illustration of CodeDepartmentQuery in a code fragment

DAOStore class is the interface of Action. A DAOStore object can be created in QueryAction, and this interface can be implemented the public method

orederDao(). This orderDao() method usually has two parameters. One is the value transmitted by JSP page, the other is the attribute ID in database table. Subsequently, a specific object will be created by method createCode() in DaoFactory class, and then invoking query method in CodeDAO class to achieve record set which typed List. The code fragment in DAOStore class is as following in Figure 6.

```
public DAOStore( DaoFactory factory){
    this.factory=factory; }
public List orderDao(String type,String id){
      CodeDAO codedao = null;
      factory=new DaoFactory();
      codedao=factory.createCode(type);
      List list=codedao.findByCode(id);
      return list;
}…….
DAOStore dao=new DAOStore(null);
List<CodeDepartment>codeDeptList=new
      ArrayList<CodeDepartment>();
CodeDeptList=(List<CodeDepartment>)dao.order
Dao(department,departmentId);
```

FIGURE 6 Illustration of DAOStore in a code fragment

This application pattern separates object instantiated of specific coding table out and put it inside the factory class. The specific object is created by the createCode() method in DAOFactory class. This process aims to increase code flexibility. A series of algorithms and methods are defined in abstract CodeDAO class. These algorithms and methods are the channel of database querying. If the programmer needs to add a new query method, it should be added in the CodeDAO class directly, and then overwrite this method in the inherited class. This improves the extensibility of application system.

The process of applying this Database Accessing Middleware in the Deli Trading system is as shown in Figure 7. For Hibernate and Struts2 based system, this paper proposes a new solution of applying middleware to integrate the Struts2 and Hibernate, instead of re-adding a Spring Framework. Accordingly, it is adopted that using factory pattern to implement business logic processing. Through the Database Accessing Middleware, objects are encapsulated in the middleware, which provides better decoupling. At the same time, the strategy pattern is adopted to improve extensibility.

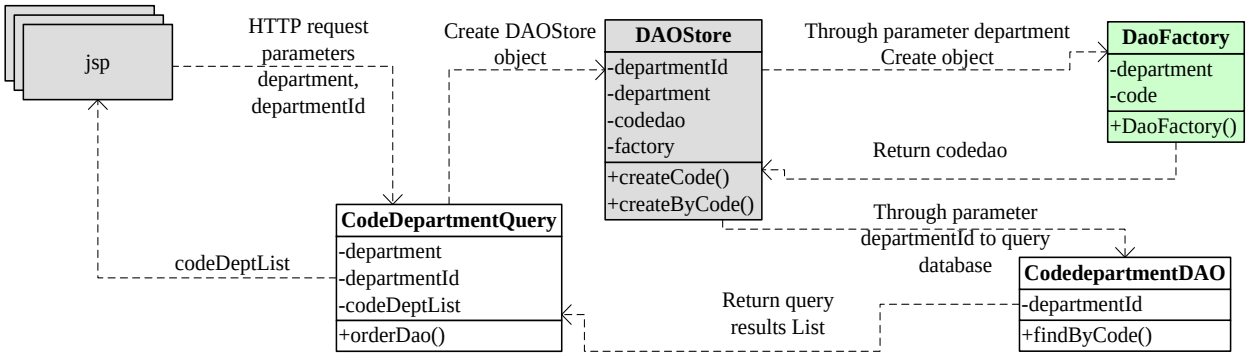

FIGURE 7 Process of applying database accessing middleware

5 Conclusions

With the gradual mature of the framework technology and object oriented technology, it is important to consider how to use the optimal design idea to complete the internal structure of the system, which will makes it easy to adapt code modification, method extension and polymorphism. Middleware has been proved successful in assisting distributed application development, making development process faster and easier and significantly enhancing software reuse. Considering the above factors, this paper designs and implements a Database Accessing Middleware based on factory and strategy design pattern. This Database Accessing Middleware can adopt different framework according to different application systems. Through applying design pattern into the code fragment, it contributes to development maintainable, expandable, reusable and flexible code fragment.

Acknowledgments

This work was supported by the Fundamental Research Funds for the Central Universities (Grant No. 3132014307 and Grant No. 3132014081)

References

[1] Llopis M, Ferrández A 2013 How to make a natural language interface to query databases accessible to everyone: An example *Computer Standards and Interfaces* **35**(5) 470-81

[2] Li Dan, Liu Lihua 2014 Design of LWMS based on Struts and Hibernate *Lecture Notes in Electrical Engineering* (4) 113-9

[3] Abd El-Aziz A A, Kannan A 2012 Storing XML rules in relational storage of XML DTD *Proceedings of the 2nd International Conference on Computational Science, Engineering and Information Coimbatore India* 408-12

[4] Fisteus J A, Norberto F, Fernández L S, Fuentes-Lorenzo D 2014 Ztreamy: A middleware for publishing semantic streams on the Web *Journal of Web Semantics* **25** 16-23
[5] Ajana M E, Harroud H, Boulmalf M, Elkoutbi M 2011 FlexRFID middleware in the supply chain: Strategic values and challenges *International Journal of Mobile Computing and Multimedia Communications* **3**(2) 19-32
[6] Huang J, Dong H, Cai Z, Wu Q 2012 Research on a Novel Multi-database Middleware for Multiple Applications *Journal of Convergence Information Technology* **7**(19) 250-7
[7] Li Q, Zhou M 2010 The Future-Oriented Middleware Technology *Journal Of Computers* **5**(2) 655-9
[8] Gamma E, Helm R, Johnson R, Vlissides J 1995 Design Patterns: Elements of Reusable Object-Oriented Software *Addison-Wesley Longman Publishing Co Inc. Boston*
[9] Ellis B, Stylos J, Myers B 2007 The Factory Pattern in API Design: A Usability Evaluation *29th International Conference on Software Engineering Minneapolis MN United states* 302-11
[10] Christopoulou A, Giakoumakis E A, Zafeiris V E, Soukara V 2012 Automated refactoring to the Strategy design pattern *Information and Software Technology* **54**(6) 1202-14

Authors

Zhaohui Li, 04.04.1974, China.

Current position, grades: associate professor at Transportation Management College, Dalian Maritime University, China.
University studies: PhD degree in management science and engineering from Dalian University of Technology, China in 2005.
Scientific interest: information system and software component technology.

Hongxia Yang, 12.13.1972, China.

Current position, grades: lecturer at School of Management, Liaoning Normal University, China.
University studies: Master degree in Control Engineering from Dalian University of Technology, China in 2004.
Scientific interest: E-Business, information system and logistics control engineering.