In-depth Study and Discussion of AOP

Baozhong Qi [a], Mindan Bai [b]
Communication University of China, Beijing, China
[a] qbzhong2012@163.com, [b] mdyx@cuc.edu.cn

Abstract. Object oriented programming (OOP) has always been the mainstream method, it's thought has solved many problems in the development, but as the business is strengthened, logic becomes more and more complex and functions more and more. At this time, it is necessary to consider the problem of reducing the coupling degree. The aspect-oriented programming (AOP) can solve this problem well. The crosscutting code is woven into the logical component. This paper will give a detailed analysis of the AOP principle and its application in Spring.

Keywords: Face oriented programming, OOP, cross section, Spring Technology.

1. Introduction

Object-Oriented Programming (OOP) has become the mainstream development idea nowadays. It has reusability, flexibility and extensibility to model the object-oriented processing. However, when dealing with such functions as log, it is necessary to embed log-related program code into business code, which results in serious tight coupling between modules and is not conducive to code. Later maintenance, in this case, AOP can be used for aspect-oriented programming separation.

AOP Aspect-Oriented Programming can enhance its functions without changing its own program code, such as adding database transaction processing and logging functions to your original code. This enhancement is realized through the principle of Proxy agent.

Proxy popular explanation such as buying books in Dangdang. Dangdang only needs to put the books in the mail package and write down your destination address. The rest of the tasks are delivered to you by the express company. In this process, the express company is equivalent to the proxy company of Dangdang Online Book Delivery Service. Dangdang Online is not responsible for delivering books. The task of delivering books is represented by the express company, which is the agent model. so, if you want to enhance a class without changing the original code, you just need to enhance the proxy class. What to enhance and what to enhance are decided by the proxy class.

2. Agent Design Model of AOP Principle

The underlying principle of AOP is the agent design pattern. Spring's AOP technology is based on the agent design pattern. The agent design pattern is to provide an agent for other objects to control access to this object. In some cases, an object is not suitable or can not directly refer to another object, while a proxy object can act as a mediator between the client and the target object.

The agent design pattern can enhance the function without changing the original code, so that the code in the original object can be fully decoupled from the enhanced code. Agent mode is divided into static agent and dynamic agent. In order to understand AOP technology better, we need to study them deeply.

A. In Static Proxy, the Proxy Object and the Proxy Object Must Realize the Same Interface, Keep the Interface Style of the Proxy Object Completely, and Keep the Principle of the Interface Unchanged All the Time. Examples are as Follows: Create Test Interfaces:

```
public interface ISendBook{
 public void sendBook();
}
public class DangDangBook implements ISendBook {
 public void sendBook() {
    System.out.println("dangdang run");
```

```
        }
    }
```

New Shunfeng Express Agent Category:

Shunfeng is the proxy class, and the proxy class is DangDangBook or JDBook.

```
public class SFBookProxy implements ISendBook {
 private ISendBook sendBook;
 public SFBookProxy(ISendBook sendBook) {
    super();
    this.sendBook = sendBook;
 }
 public void sendBook() {
    System.out.println("SF  run-affir  start-connection  start-Enhance
start");
    sendBook.sendBook();
    System.out.println("SF arrive-affir commit-connection close-enhance
close");
    }
}
```

The proxy class SFBookProxy. Java also implements the ISendBook. java interface to achieve behavior consistency.

New Test.java runtime class:

```
public class Test {
 public static void main(String[] args){
    JDBook jdbook = new JDBook();
    DangDangBook dangdangBook = new DangDangBook();
    ISendBook sf1 = new SFBookProxy(jsBook);
    sf1.sendBook();
    system.out.println();
    ISendBook sf2 = new SFBookProxy(dangdangBook);
    sf2.sendBook();
 }
}
```

The results are as follows:

SF store-affir start-connection start-enhance start

Jd will send your book

SF arrived-affir commit-connection close-enhance close

SF store-affir start-connection start-enhance start

Without changing the original DangDang. java code on the basis of functional enhancements, output in the console, "When the network know your address, phone, notes, give you a book!" Print out the information of "Shunfeng Harvest-Transaction Open-Connection Open-Enhancement Start" and "Shunfeng Service-Transaction Commit-Connection Close-Enhancement End" before and after the information respectively. This is a typical log or transaction function enhancement model, which implements transaction processing without changing the original code and log processing without changing the original code.

But the main disadvantage of the static proxy class is its poor scalability, because the SFBookProxy. Java class binds the ISendBook. Java interface. If Shunfeng Express wants to proxy more delivery tasks, it needs to create more proxy classes.

B. Implementation of Dynamic Agent

The disadvantage of static proxy is that the proxy class binds a fixed interface, which is not conducive to expansion, but dynamic proxy is not. By dynamic proxy, the function of any class that implements an interface can be enhanced, and the situation of proxy class binding interface will not occur.

The object of dynamic proxy class in Java is created by the new ProxyInstance () method of Proxy. Java class. This shows that the proxy class in dynamic proxy is not created by programmers themselves like static proxy. The proxy class object in dynamic proxy class is created by JVM, and the enhanced algorithm needs to be implemented by InvocationHandler interface.

Examples are as follows:

The new enhancement algorithm class LogInvocationHandler. java code is as follows:

```
public class LogInvocationHandler implements InvocationHandler {
    private Object anyObject;

    public LogInvocationHandler(Object anyObject) {
        super();
        this.anyObject = anyObject;
    }

    @Override
    public Object invoke(Object proxy, Method method, Object[] args) throws
Throwable {
        System.out.println("log begin time=" + System.currentTimeMillis());
        method.invoke(anyObject);
        System.out.println("log end time" + System.currentTimeMillis());
        return null;
    }
}
```

According to the above three points can be summarized:

(1) Proxy classes are created by JVM. Programmers do not need to create proxy classes themselves. The number of proxy classes *. java files has dropped dramatically.

(2) The proxy class no longer binds the fixed interface, and achieves decoupling between the proxy class and the interface.

(3) In Invocation Handler, the enhancement algorithm can be handled more flexibly by reflection technology.

Both static proxy and dynamic proxy are aimed at the enhancement of public void sendBook () method. Although reflection technology is used in dynamic proxy, Spring only supports the enhancement of Method method, and does not support field-level enhancement. Spring believes that doing so violates the idea of OOP object-oriented programming, so it is most appropriate to support the enhancement of Method method, and it is also suitable for S. Other modules of pring are more standard when they are integrated and developed.

3. Subtechnology of AOP

The principle of AOP technology in Spring framework is based on dynamic proxy, and mastering dynamic proxy is the premise of mastering AOP technology. AOP Aspect-Oriented Programming has become an independent technology with its own unique terminology.

3.1 Cross-cutting-Concerns

In developing software projects, we need to pay attention to some "general" functions, such as calculating the execution time of the method, whether there is permission to access resources, and doing some routine logs. The contents of the logs include which user is logged in, what operation is done at what time, whether the result of the operation is normal or abnormal, and other information that needs to be recorded. Opening and closing of database Connection connections scattered throughout the system, and so on. Most of the code for these "generic" functions is interwoven in Service business objects.

In developing software projects, we need to pay attention to some "general" functions, such as calculating the execution time of the method, whether there is permission to access resources, and doing some routine logs. The contents of the logs include which user is logged in, what operation is done at what time, whether the result of the operation is normal or abnormal, and other information that needs to be recorded. Opening and closing of database Connection connections scattered throughout the system, and so on. Most of the code for these "generic" functions is interwoven in Service business objects.

The "decoupling" in DI dependency injection is mainly the decoupling between objects, while the "decoupling" in AOP mainly separates the code related to "crosscutting concerns" from the "business code" and decouples them. Although they are all decoupled, they are different in nature.

3.2 Aspect and AOP Aspect-Oriented Programming

The process is divided into several departments in AOP, In AService. java, BService. Java and CService. java, all of the functions in three aspects need to be extracted from the service. Java service class. When the code in the service business class is executed, the functional code in the aspect is dynamically invoked, which realizes the reuse of the universal functional code of "crosscutting concerns" and achieves the goal of decoupling.

Although the process can be divided into several parts by "cut", the execution process of the main process will not be interrupted. For example, AService. java's process code will not be interrupted because of the existence of facets, and will run until the end. In this process, the existence of facets has some characteristics similar to filter, intercept, process, and release. Filter filters only intercept request requests, while facets intercept method calls. The characteristics of operation are basically the same, but for different purposes, one is request requests, the other is method calls.

3.3 Join Point

Connection point is a point that can insert a section in the process of software execution. Connection points can be called join points before calling methods, after calling methods, when methods throw exceptions, after methods return values, etc. These join points can insert generic functions defined in the cut surface and add new software behavior.

It is unrealistic to apply sections to all joints. In most cases, we only want to apply sections to "partial joints". These "partial joints" are called "tangents". They are applied to tangents.

3.4 Advice

The tangent point defines the precise location of the application aspect, but when the application aspect is applied is determined by the notification Advice, such as when the application aspect can be applied in the case of exceptions before and after the execution of the method, which can be called notification Advice.

In Spring's AOP, there are five types of notifications:

(1) Before: Before a method is called

(2) After: After the method is called

(3) Around notification: before and after a method is called

(4) After-returning notification: Method returns a value

(5) After-throwing: The method has an exception

Functional codes in facets can be applied to all five notification Advice types.

Cut bread contains notice and cut point, which is the combination of the two. Notification and cut point are the most basic elements of cut surface.

(1) Notification: Defines when to engage in all aspects

(2) Tangent Points: Defines which join points you can place a tangent

3.5 Weaving

Weaving is the application of cut surface to specified objects. The principle of weaving in Spring AOP is to create proxy objects by JVM, call methods in original objects in proxy objects, and then

implement them with enhancement algorithm. So, the principle of AOP technology in Spring is agent design pattern. Because AOP technology in Spring is based on dynamic proxy, AOP in Spring only supports method connection points, and does not support field-level connection points.

Aspect is the modularization of cross-cutting concerns, i.e. extracting the functional code of cross-cutting concerns and putting them into a single class for unified processing. This class is "cross-cutting class", also known as "cross-cutting Aspect". In AOP programming, the main purpose is to design code for aspect, so the whole process of AOP is face-oriented programming.

By putting the functional code of cross-cutting concerns into the method of cross-cutting concerns, the modularization of cross-cutting concerns is achieved. The methods in cross-cutting concerns can be accessed by many Java classes in a shared way.

4. Conclusion

AOP (Aspect-Oriented Programming) is actually a supplement and improvement of the idea of OOP (Object-Oriented Programing). It uses a technique called "crosscutting" that breaks the inside of encapsulated objects and encapsulates common behaviors that affect multiple classes and are unrelated to specific services into a single module (called a slice). What's more, it can restore these cracked cuts in a clever way, without leaving traces into the core business logic. In this way, the editing and reuse of the cross-cut function in the future can bring great convenience. The specific implementation of AOP technology is actually a static "weaving" method through dynamic proxy technology or during program compilation. Its advantages are bound to save development costs, optimize the development process, and applications in Spring will be more popular.

References

[1]. Rod JoHeson, Juergen Hoeller, et al. Translated by Jiang Pei. Spring Framework Advanced Programming. Beijing: Machinery Industry Press, 2006.

[2]. Lin Xinliang. Spring 2.0 Technical Manual [M]. Beijing: Electronic Industry Press, 2007 (4).

[3]. Xingjun. Research and application of AOP development process [D]. Beijing: Beijing University of Technology, 2009.

[4]. Shen Lijun. Research and application of MOVC model based on AOP [D]. Dalian: Dalian Maritime University, 2011.

[5]. Xu Yingjie, Wang Jian, Lu Liming. Application of AOP in Web Development [J]. Computer Science, 2010.

[6]. Liu Ronghui, Xue Bing, Design of Spring AOP System Based on Annotation [J]. Computer Application and Software, 2009.9.