Application of Adapter pattern in Eclipse plug-in development

Xianchao Yang

Hangzhou Vango Technologies, Inc., Hangzhou, China

yangxc@vangotech.com

Abstract. The Adapter design pattern is widely used in software development. It can transform the interface of one class into another desired by the client, so that two classes that cannot work together because of interface mismatch can work together[1]. It is the core of the extensibility of the Eclipse platform. It is especially important in Eclipse, and it is necessary for anyone who is working on or preparing for Eclipse plug-in development to master it. It is a necessary condition for becoming a good developer. Compared with the traditional Adapter mode, there are some differences in the use of Adapter mode in Eclipse. From the perspective of Eclipse plug-in development, this paper firstly introduces the traditional Adapter mode, and then introduces the Adapter mode in Eclipse plug-in. And through the example let the reader have a deeper understanding.

Keywords: Eclipse,Adapter, plug-in development, design patterns,Java.

1. Introduction

With the development of software industry, more and more easy to use and cool integrated development environment emerged, such as IDEA, NetBeans, etc., the old integrated development environment Eclipse seems to be abandoned by people, and Eclipse plug-in development technology is even less popular. However, But with the rapid development of semiconductor industry in recent years, Eclipse plug-in development seems to become active. Automotive semiconductor, chip tool chain, semiconductor equipment, industrial equipment, etc., all need to customize their own integrated development environment or operation interface based on Eclipse, such as NXP MCUXpressoIDE in the semiconductor industry. STM32 STM32CubeIDE, Advantest 93K and so on are based on Eclipse development. Eclipse is an open source project of IBM. It is not only an integrated development environment, but also an extensible platform. It provides many basic modules on which developers can quickly build their own software platforms, which is why many manufacturers choose it to build software systems. Want to develop a set of good software, the demand for talent is essential, but because of the industry is unpopular, learning materials and other reasons, few people choose this industry or give up this industry, Adapter pattern is the core idea of Eclipse plug-in development, it runs through the entire Eclipse system, if you often read Eclipse source code, You'll find Adapter popping up all over the Eclipse source code, which can be difficult for beginners and inexperienced developers to understand, but must be mastered. This article will help you understand the application of the Adapter pattern in Eclipse through a short language.

2. The traditional Adapter pattern

The adapter pattern is a structural pattern, which can associate the two classes that are not intended to be closed by adding a new adapter class, so as to achieve the function of collaborative work. According to the different relationship between the adapter class and the adaptee class, the adapter pattern can be divided into two types: class adapter and object adapter. The adapter pattern usually contains four elements,respectively:

Client: The caller of the target interface.

Target (target Abstract class) : The type of interface expected by the client class, which can be an interface, abstract class, or concrete class.

Adapter: The core of the adapter pattern that transforms from the adapter class to the target class.

Adaptee: The adaptee is the adapted role, which is an existing interface that contains the business method that the client wants to use, but is not compatible with the target class.

2.1 Class Adapters

In the class adapter pattern, the adapter class implements the target interface class and inherits the adaptee class, and then calls the methods of the adaptee class in the method implementation of the target interface to achieve the adaptation effect. However, this method is not commonly used, because it has certain drawbacks. If the target class and the adaptee class are both concrete classes, it cannot be implemented in a single inheritance language such as Java, and the adapter class will become bloated through inheritance. The class adapter pattern structure is shown in Figure 1 below:

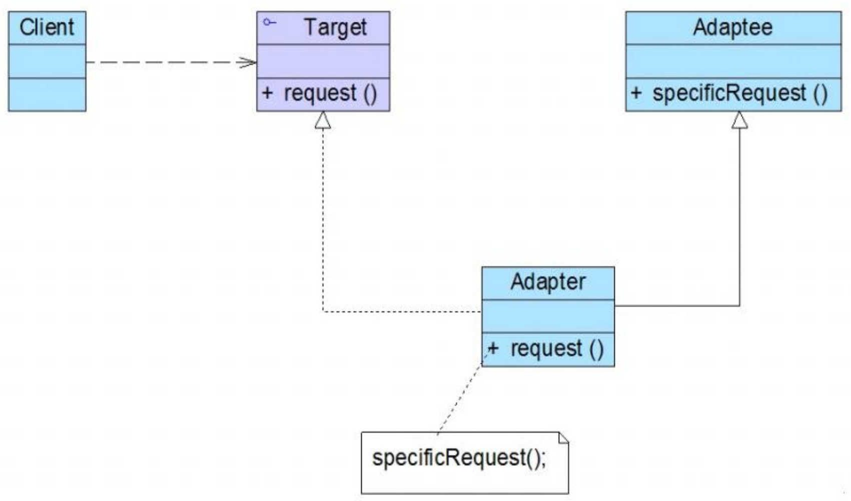


Fig. 1 Class adapter structure diagram.

2.2 Object Adapters

In the object adapter pattern, the adapter and the adaptee are containment relationships. The adapter implements the target interface type and contains an instance of the adaptere class. The adaptee's method is called in the implementation of the target interface. The client calls the interface of the adapter, which indirectly calls the method of the adaptee to achieve the role of adaptation. This method can reduce coupling and is a common method. The object adapter pattern structure is shown in Figure 2 below:

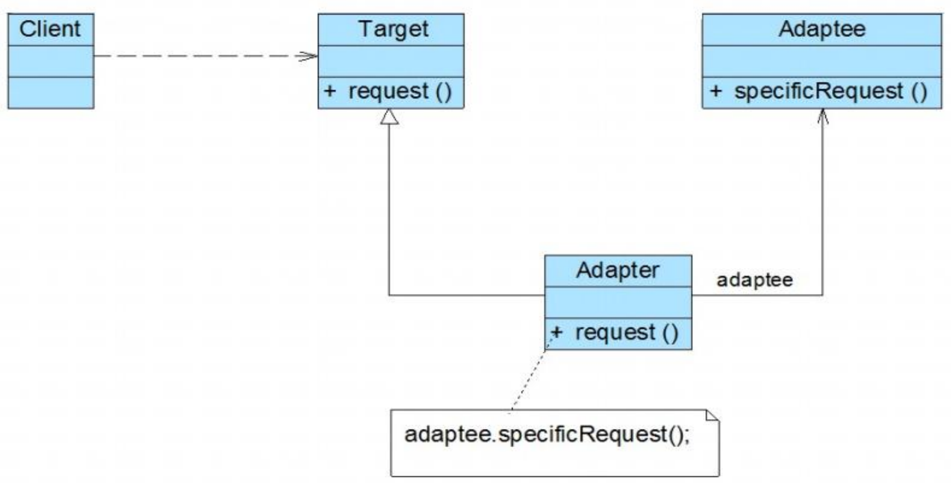


Fig. 2 Object adapter structure diagram.

2.3 An easy to understand example

For those who are not familiar with the adapter pattern, the above introduction may not make sense. Here is a simple example to illustrate: Xiao Ming dug out a pair of round-hole earphones from the drawer, he wanted to test the sound quality of this earphone. He has a Huawei phone and an Apple phone that can play music, and the jack of Huawei phone is Type-C, while the jack of Apple phone is Lightning. Obviously, neither of them can be plugged into the round-hole earphone. In the adapter pattern, the earphone will act as the client's role, it can expect a play music can be inserted into the round hole headset devices, apple phone and huawei phone acted as the role of adaptee, make them work together the simplest way is to introduce an apple connector or adapter, huawei and that this adapter is served for the roles of the adapter, It has the circular port expected for headphones, but also connects the phone, achieving the function of adaption. This is shown in Figure 3:

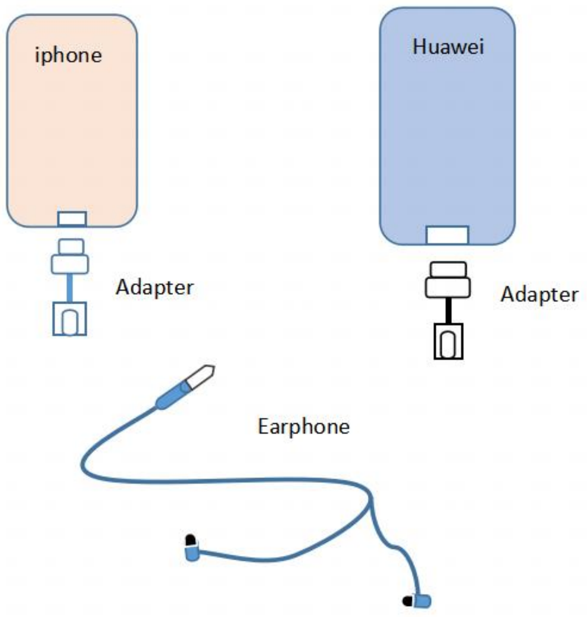


Fig. 3 Example illustration.

3. Adapter pattern in Eclipse plug-ins

Eclipse is not only an integrated development environment, but also an extensible platform. Eclipse provides many standard interfaces for developers to extend, but for some reason the extension class cannot implement the interface or want to extend an existing class, you can use Adapter pattern. But Eclipse is for all developers, and it doesn't know whether a class will fit into the desired type. So Eclipse provides a mechanism for clients to proactively obtain adapters for a class. This is the biggest difference in usage from the traditional Adapter pattern. The main interfaces and classes involved in adaptation are as follows, and the class relationship is shown in Figure 4 below:

IAdaptable, the class that implements the interface indicates that it's adaptable.
PlatfromObject, a subclass of IAdaptable, provides the basic implementation.
IAdapterManager, register adapter factory, and participate in adaptation.
IAdapterFactory, the IAdapterFactory, is responsible for adapting one type to another.
Adapters, This is a tool class, responsible for coordinating the adaptation process.

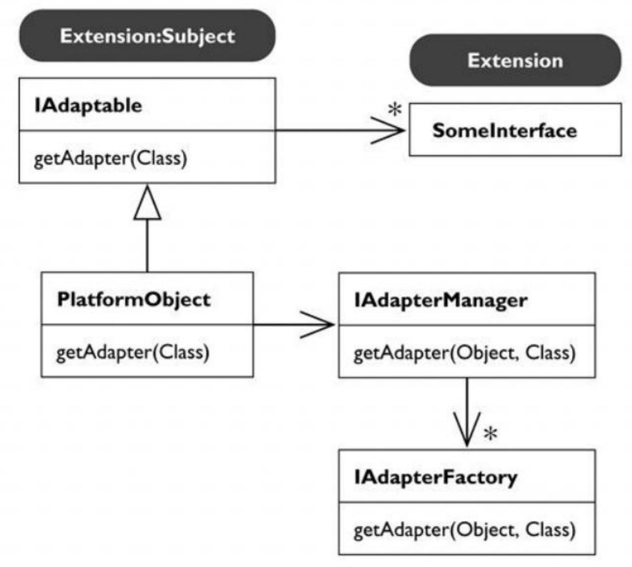
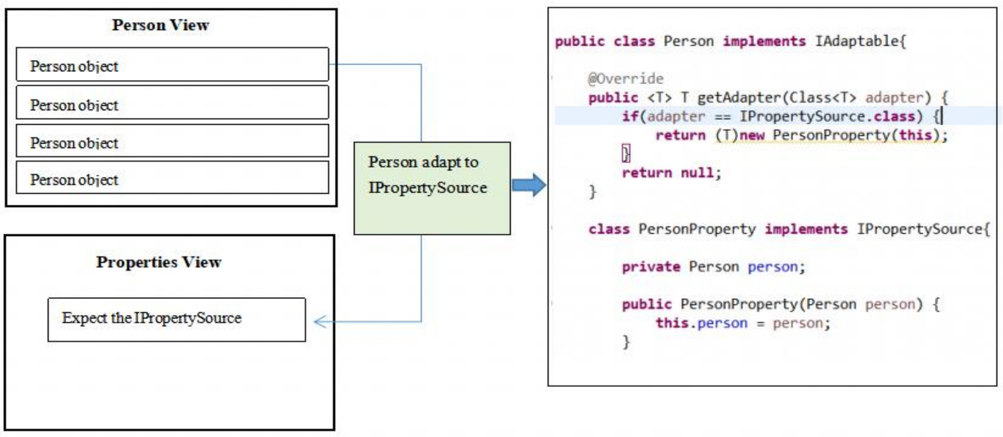


Fig. 4 Adapters in eclipse.

3.1 IAdaptable

Eclipse provides an IAdaptable interface, it has a getAdapter(Class) method that allows callers to retrieve a particular type of adapter, it has an implementation class PlatformObject, in the getAdapter method in the class, The implementation of the adapter is delegated to the AdapterManager. Implementing this interface is a good choice for a new type. If you want to adapt to more than one type, you just need to check in the getAdapter(Class) method. The classic use of Aadpater in Eclipse is in the Properties view, for example:

The user develops a Person view, creates a TableViewer in the view to display personal information, and registers the view with SelectionProvider. Due to the limited content of the table, complete personal information cannot be displayed in detail. In this case, the Properties view provided by Eclipse is used to display personal details. When a Person entry in the Person view is selected, the SelectionProvider passes the Person object to the Properties view. Since the interface type expected by the Properties view is IPropertySource, you need to match the Person type to IPropertySource in order to display it in the Properties view. When the Properties view gets the Person object, The getAdapter method of the Person object is called to get an adapter of type IPropertySource, with the diagram and key code shown in Figure 5 below:

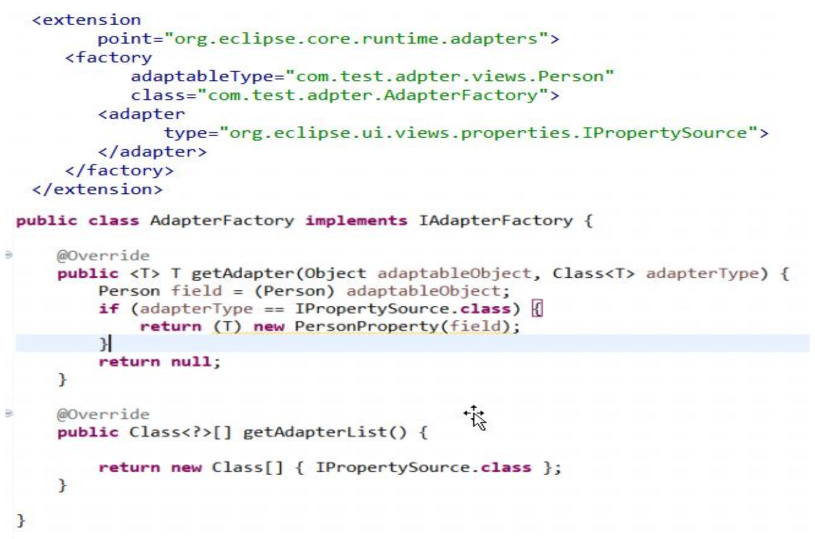


```
public class Person implements IAdaptable{

    @Override
    public <T> T getAdapter(Class<T> adapter) {
        if(adapter == IPropertySource.class) {
            return (T)new PersonProperty(this);
        }
        return null;
    }

    class PersonProperty implements IPropertySource{

        private Person person;

        public PersonProperty(Person person) {
            this.person = person;
        }
```

Fig. 5 IAdaptable.

3.2 IAdaptableFactory

The above is the adaptation of new types. How to adapt existing types? Eclipse provides the IAdaptableFactory interface, which allows users to adapt existing types. That is, you can adapt an object type to any object type through IAdaptable Factory. At this time, we can implement the effect of 2.1 in another way. If Person is an existing type and does not implement the IAdaptable interface, we can use the org.eclipse.core.runtime provided by Eclipse Adapters extension point, add the type Person that needs to be adapted, the adaptable type IPropertySource (the adaptable type can be multiple), create an adapter factory class, and adapt the Person class in its getAdapter method to display detailed information in the Properties view. The key code is shown in Figure 6 below:



```
<extension
      point="org.eclipse.core.runtime.adapters">
    <factory
          adaptableType="com.test.adpter.views.Person"
          class="com.test.adpter.AdapterFactory">
        <adapter
              type="org.eclipse.ui.views.properties.IPropertySource">
        </adapter>
    </factory>
</extension>
public class AdapterFactory implements IAdapterFactory {

    @Override
    public <T> T getAdapter(Object adaptableObject, Class<T> adapterType) {
        Person field = (Person) adaptableObject;
        if (adapterType == IPropertySource.class) {
            return (T) new PersonProperty(field);
        }
        return null;
    }

    @Override
    public Class<?>[] getAdapterList() {

        return new Class[] { IPropertySource.class };
    }

}
```

Fig. 6 IAdaptableFactory.

3.3 Adaptation Process

It can be seen from the analysis of source code that when a Person in the table is selected, the SelectionProvider will pass the Person object to the Properties view, and the Properties view will get the Person object, which will be delegated to an Adapters tool class to request an adapter of IPropertySource type. In the Adapters, the following operations will be performed: first, judge whether the object is IAdaptable, and if so, try to obtain the adapter, If the acquisition is not null, the adapter will be returned directly. If the acquisition is null or not an adaptable type, the AdapterManager will be called for adaptation. The implementation of the AdapterManager is to find out whether an IAdaptable Factory can adapt to this type through the extension point. The internal implementation of Adapters is shown in Figure 7 below:

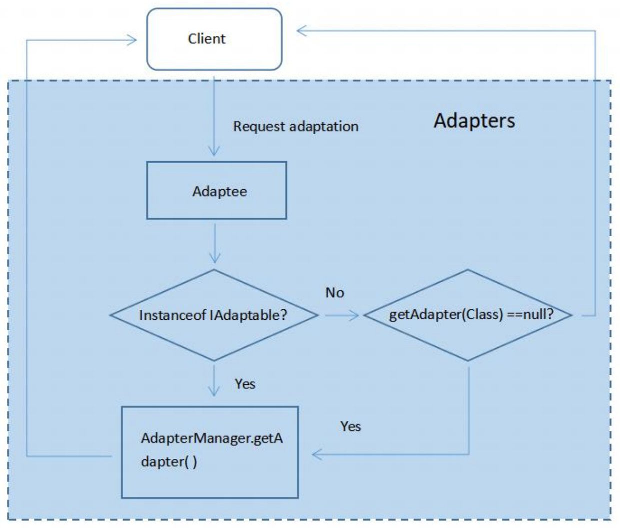


Fig. 7 Adapters Adapt Process.

3.4 Principle of adaptation

In theory, the adapter pattern can adapt any class to the type expected by the client. However, no one will do this in actual use, because there is a certain premise to use the adapter mode, that is, the adapter does have some functions that the client needs, and only because the interface does not match can it be used directly, so it needs to be used after adaptation. You can't associate flying with dogs, because dogs don't have the ability to fly at all, and they can't fly even if they are forced to install wings.

4. Conclusion

Design patterns are often ignored in software development. Many developers only focus on function implementation and rarely think about design patterns when developing. When we read excellent code, we will find that a large number of design patterns are used in it. For those who do not understand design patterns, it will be difficult to read. This article focuses on the application of Adapter pattern in Eclipse. It can be seen from the above examples, In Eclipse, not only the Properties view, but also many other places are realized through the adapter method. Extensibility is an important feature in Eclipse. As an Eclipse plug-in developer, mastering this feature and its importance, mastering the Adapter mode, will basically grasp the core idea. This article introduces the traditional adapter mode, then explains the implementation of the adapter in Eclipse through examples, and finally analyzes the source code. It can not only help Eclipse plug-in developers, but also help other developers.

References

[1] Liu Wei Java design pattern.
[2] RCP Programming.
[3] Zhang Peng, Jiang Hao, Xu Li. Eclipse Plug-in Development Learning Notes,2008.7 ISBN 978-7-121-05498-3.
[4] Na Jing, Core application of Eclipse SWT/JFace.
[5] Clayberg,E. (USA), Rubel,D. (USA),Eclipse Plug-in Development, 3rd Edition Chinese version.
[6] premise, that is, the adaptor does have some functions that the client needs.