简述

避免请求发送者与接收者耦合在一起，让多个对象都有可能接收请求，将这些对象连接成一条链，并且沿着这条链传递请求，直到有对象处理请求，客户只需要将请求发送到职责链上即可，无须关心请求的处理细节和请求的传递，所以职责链将请求的发送者和请求的处理者解耦

使用场景与示例

1、Spring Mvc 的拦截器
2、jsp servlet 的 Filter

代码

问题类

```
public class Problem {

    private String type;

    public Problem(String type) {
        this.type = type;
    }

    public String getType() {
         return type;
     }

```

```
    public void setType(String type) {
        this.type = type;
    }
}
```

处理问题的抽象类

```

public abstract class Handler {

    protected String type;

    protected Handler next;

    public Handler(String type){
        this.type = type;
    }

    public Handler setNext(Handler next){
        this.next = next;
        return next;
    }

    /**
     * 处理问题
     * @param problem
     */
    public final void doHandler(Problem problem){
        if (canDo(problem)){
            done(problem);
        }else if (next != null){
            System.out.println(type + "老师 无法解决" + problem.getType() + " 问题");
            next.doHandler(problem);
        }else {
            fail(problem);
        }
    }

    private void fail(Problem problem){
        System.out.println("谁都无法无法解决" + problem.getType() + "问题");
    }

    // 为了简化代码，这里只把canDo 由实现类去实现，done放到父类处理了
    private void done(Problem problem){
        System.out.println(type + "老师 解决了 " + problem.getType() + " 问题");
    }

    protected abstract boolean canDo(Problem problem);
}
```

处理问题的具体类：化学问题

```
public class ChemistryHandler extends Handler {

    public ChemistryHandler(String type) {
```

```
        super(type);
    }

    @Override
    protected boolean canDo(Problem problem) {
        return this.type.equals(problem.getType());
     }
}
```

处理问题的具体类：英语问题

```
public class EnglishHandler extends Handler{
    public EnglishHandler(String type) {
        super(type);
    }

    @Override
    protected boolean canDo(Problem problem) {
        return this.type.equals(problem.getType());
    }
}
```

处理问题的具体类：语文问题

```
public class LanguageHandler extends Handler {
    public LanguageHandler(String type) {
        super(type);
    }

    @Override
    protected boolean canDo(Problem problem) {
        return this.type.equals(problem.getType());
    }
}
```

处理问题的具体类：数学问题

```
public class MathematicsHandler extends Handler {

    public MathematicsHandler(String type) {
        super(type);
    }

    @Override
    protected boolean canDo(Problem problem) {
        return this.type.equals(problem.getType());
     }
}
```

处理问题的具体类：物理问题

```
public class PhysicalHandler extends Handler{

    public PhysicalHandler(String type) {
        super(type);
```

```
    }

    @Override
    protected boolean canDo(Problem problem) {
        return this.type.equals(problem.getType());
     }

}
```

客户端

```
public class Main {


    public static void main(String[] args) {
        String[] problemArray = {"language", "math", "english", "physical", "chemistry", "computer"}

        // 化学老师解决化学问题
        ChemistryHandler chemistryHandler = new ChemistryHandler("chemistry");

         // 英语老师解决英语问题
         EnglishHandler englishHandler = new EnglishHandler("english");

         // 语文老师解决语文问题
         LanguageHandler languageHandler = new LanguageHandler("language");

         // 数学老师解决数学问题
         MathematicsHandler mathematicsHandler = new MathematicsHandler("math");

         // 物理老师解决物理问题
         PhysicalHandler physicalHandler = new PhysicalHandler("physical");

         chemistryHandler.setNext(englishHandler).setNext(languageHandler).setNext(mathematicsHandle


         for (int i = 0; i < 10; i++) {
             int index = new Random().nextInt(problemArray.length);
             Problem problem = new Problem(problemArray[index]);
             System.out.println("问题:  " + problem.getType());
             chemistryHandler.doHandler(problem);
             System.out.println();
         }
     }
}
```

总结

优点

1、降低耦合度。它将请求的发送者和接收者解耦。

2、简化了对象。使得对象不需要知道链的结构。
3、增强给对象指派职责的灵活性。通过改变链内的成员或者调动它们的次序，允许动态地新增或者删除责任。
4、增加新的请求处理类很方便

缺点

1、不能保证请求一定被接收。
2、系统性能将受到一定影响（相对于直接定位到处理者），而且在进行代码调试时不太方便，可能会造成循环调用。
3、可能不容易观察运行时的特征，有碍于除错