责任链设计模式

定义：责任链模式(Chain of Responsibility)使多个对象都有机会处理请求，从而避免请求的发送者和接受者之间的耦合关系。将这些对象连成一条链，并沿着这条链传递该请求，直到有对象能够处理它。

类型：行为类模式

考虑以下这样的场景：

你要去给某公司借款 1 万元，当你来到柜台的时候向柜员发起 "借款 1 万元" 的请求时，柜员认为金额太多，处理不了这样的请求，他转交这个请求给他的组长，组长也处理不了这样的请求，那么他接着向经理转交这样的请求。

用 Java 代码表示为：

**public** **void** **test(**Request request**)** **{**

**int** money **=** request**.**getRequestMoney**();**

**if(**money **<=** 1000**)** **{**

Clerk**.**response**(**request**);**

**}** **else** **if(**money **<=** 5000**)** **{**

Leader**.**response**(**request**);**

**}** **else** **if(**money **<=** 10000**)** **{**

Manager**.**response**(**request**);**

**}**

**}**

代码的业务逻辑就是这样：根据的借款金额来判定谁来处理这个借款请求 (request)

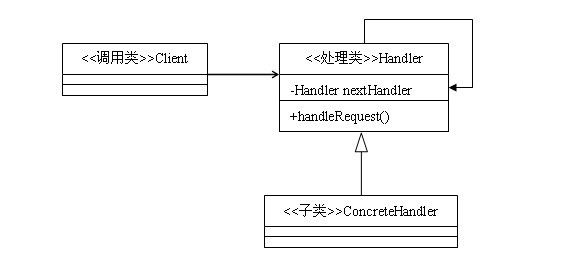
* 如果请求借款金额小于 1000 元，那么柜台职员就可以直接处理这个请求(比如签字)
* 如果请求借款金额小于 5000 元但大于 1000 元，那么职员处理不了，该请求转交给组长，组长能够处理这样的请求(比如签字)
* 如果请求借款金额大于 5000 元但小于 10000 元，那么职员和组长都处理不了(没有权限)，那么这个请求就会转交给经理，经理能够处理这样的请求(比如签字)

在编程中，这样处理业务逻辑的方法非常常见，这样的方法非常直观，简单明了，并且也比较容易维护，但是同时它也存在着着问题:

1. 代码臃肿: 实际应用中的判定条件通常不是这么简单地判断金额，也许需要复杂的操作，也许需要查询数据库等等，这就会产生许多额外的代码，如果判断条件再比较多的话，那么代码就会大量地堆积在同一个文件中。
2. 耦合度高：如果我们想继续添加处理请求的类，那么就需要添加 else if 的判定条件；另外，这个条件判定的顺序也是写死的。如果想改变顺序，那么也只能修改这个条件语句。

下面就介绍责任链模式：

责任链模式的类图非常简单，如下图：



1. **抽象处理类:**主要包含一个指向下一处理类的成员变量 nextHandler 和一个处理请求的方法 handRequest，handRequest 方法的主要思想是，如果满足处理的条件，则有本处理类来进行处理，否则由 nextHandler 来处理。

2. **具体处理类：**具体处理类的主要是对具体的处理逻辑和处理的适用条件进行实现。

了解完责任链的大体思想之后，在看看 Java 代码是如何实现的：

等级类：

**public** **class** **Level** **{**

**private** **int** level **=** 0**;**

**public** **Level(int** level**)** **{**

**this.**level **=** level**;**

**}**

**public** **int** **getLevel()** **{**

**return** level**;**

**}**

**public** **boolean** **above(**Level level**)** **{**

**if(this.**level **>=** level**.**getLevel**())** **{**

**return** **true;**

**}else** **{**

**return** **false;**

**}**

**}**

**}**

请求与响应：

*//请求*

**class** **Request** **{**

Level level**;**

**public** **Request(**Level level**)** **{**

System**.**out**.**println**(**"开始请求..."**);**

**this.**level **=** level**;**

**}**

**public** Level **getLevel()** **{**

**return** level**;**

**}**

**}**

*//响应*

**class** **Response** **{**

**private** String message**;**

**public** **Response(**String message**)** **{**

System**.**out**.**println**(**"处理完请求"**);**

**this.**message **=** message**;**

**}**

**public** String **getMessage()** **{**

**return** message**;**

**}**

**}**

代码中 Request、Response 分别代表请求和响应，Level 类代表等级，above 方法用于比较等级，等级低的无法处理等级高的请求，所以返回 false，等级高的可以处理等级相同或者低的请求，返回 true。

抽象处理类和具体处理类：

*//抽象处理器*

**abstract** **class** **Handler** **{**

**private** Handler nextHandler **=** **null;**

**public** **void** **setNextHandler(**Handler handler**)** **{**

nextHandler **=** handler**;**

**}**

**public** **final** Response **handlerRequest(**Request request**)** **{**

Response response **=** **null;**

**if(this.**getHandlerLevel**().**above**(**request**.**getLevel**()))** **{**

response **=** **this.**response**(**request**);**

**}else** **{**

**if(**nextHandler **!=** **null)** **{**

response **=** **this.**nextHandler**.**handlerRequest**(**request**);**

**}else** **{**

System**.**out**.**println**(**"没有合适的处理器处理该请求..."**);**

**}**

**}**

**return** response**;**

**}**

**protected** **abstract** Level **getHandlerLevel();**

**public** **abstract** Response **response(**Request request**);**

**}**

*//具体的处理器 1*

**class** **ConcreteHandler1** **extends** Handler **{**

**protected** Level **getHandlerLevel()** **{**

**return** **new** Level**(**1**);**

**}**

**public** Response **response(**Request request**)** **{**

System**.**out**.**println**(**"该请求由 ConcreteHandler1 处理"**);**

**return** **new** Response**(**"响应结果 1"**);**

**}**

**}**

*//具体的处理器 2*

**class** **ConcreteHandler2** **extends** Handler **{**

**protected** Level **getHandlerLevel()** **{**

**return** **new** Level**(**2**);**

**}**

**public** Response **response(**Request request**)** **{**

System**.**out**.**println**(**"该请求由 ConcreteHandler2 处理"**);**

**return** **new** Response**(**"响应结果 2"**);**

**}**

**}**

*//具体的处理器 3*

**class** **ConcreteHandler3** **extends** Handler **{**

**protected** Level **getHandlerLevel()** **{**

**return** **new** Level**(**3**);**

**}**

**public** Response **response(**Request request**)** **{**

System**.**out**.**println**(**"该请求由 ConcreteHandler3 处理"**);**

**return** **new** Response**(**"响应结果 3"**);**

**}**

**}**

抽象类 Handler 中主要进行条件的判断，只有处理类的处理等级高于 Request 的等级才能处理，否则交给下一个处理者处理。

客户端调用：

**public** **class** **Client** **{**

**public** **static** **void** **main(**String**[]** args**)** **{**

Handler ch1 **=** **new** ConcreteHandler1**();**

Handler ch2 **=** **new** ConcreteHandler2**();**

Handler ch3 **=** **new** ConcreteHandler3**();**

ch1**.**setNextHandler**(**ch2**);**

ch2**.**setNextHandler**(**ch3**);**

Response res1 **=** ch1**.**handlerRequest**(new** Request**(new** Level**(**2**)));**

**if** **(**res1 **!=** **null)** **{**

System**.**out**.**println**(**res1**.**getMessage**());**

**}**

Response res2 **=** ch1**.**handlerRequest**(new** Request**(new** Level**(**4**)));**

**if** **(**res2 **!=** **null)** **{**

System**.**out**.**println**(**res2**.**getMessage**());**

**}**

**}**

**}**

在 Client 类中处理好链的前后执行顺序，执行时将请求交给第一个处理类，也就是等级最低的处理类，这就是责任链模式，它完成的功能与前文 if...else... 方法相同。

运行结果：



**总结：**

1. 责任链模式与 if...else 相比，他的耦合性要低一些，因为它将条件判定分散到各个处理类中，并且这些处理类的优先处理顺序可以随意的设定，并且如果想要添加新的 handler 类也是十分简单的，这符合开放闭合原则。
2. 责任链模式带来了灵活性，但是在设置处理类前后关系时，一定要避免在链中出现循环引用的问题。

最后我们使用责任链模式来完成一个实际的场景：

有一个银行的借款系统可以帮助用户借款，负责该系统的有职员，组长还有经理，他们的等级由低到高，并且他们能够允许借款的额度也是从低到高的，分别是：

1. 职员最高可以批准 5000 元的借款额度
2. 组长最高可以批准 20000 元的借款额度
3. 经理最高可以批准 100000 元的借款额度

当有转账的请求过来时，先由等级低的员工处理，若无法处理，则将请求移交给上级处理。

根据上面的场景，运用责任链模式，用 Java 代码表示为：

首先先定义借款请求：

**class** **BorrowRequest** **{**

**private** **int** requestMoney**;**

**public** **BorrowRequest(int** money**)** **{**

System**.**out**.**println**(**"有新请求，需要借款 " **+** money **+** " 元"**);**

requestMoney **=** money**;**

**}**

**public** **int** **getMoney()** **{**

**return** requestMoney**;**

**}**

**}**

然后再定义一个抽象职员类，用于实现责任链：

**abstract** **class** **AbstractClerk** **{**

**private** AbstractClerk superior **=** **null;**

**protected** String type**;**

**public** **void** **setSuperior(**AbstractClerk superior**)** **{**

**this.**superior **=** superior**;**

**}**

**public** **void** **approveRequest(**BorrowRequest request**)** **{**

**if(**getLimit**()** **>=** request**.**getMoney**())** **{**

System**.**out**.**println**(**getType**()** **+** "同意借款请求"**);**

**}else** **{**

**if(this.**superior **!=** **null)** **{**

**this.**superior**.**approveRequest**(**request**);**

**}else** **{**

System**.**out**.**println**(**"没有人能够同意借款请求"**);**

**}**

**}**

**}**

**public** **abstract** **int** **getLimit();**

**public** String **getType()** **{**

**return** type**;**

**}**

**}**

实现完抽象职员类之后，只需要创建具体的员工类，并设置其额度就可以了。

**class** **Clerk** **extends** AbstractClerk**{**

**public** **Clerk()** **{**

**super.**type **=** "职员"**;**

**}**

**public** **int** **getLimit()** **{**

**return** 5000**;**

**}**

**}**

**class** **Leader** **extends** AbstractClerk**{**

**public** **Leader()** **{**

**super.**type **=** "组长"**;**

**}**

**public** **int** **getLimit()** **{**

**return** 20000**;**

**}**

**}**

**class** **Manager** **extends** AbstractClerk**{**

**public** **Manager()** **{**

**super.**type **=** "经理"**;**

**}**

**public** **int** **getLimit()** **{**

**return** 100000**;**

**}**

**}**

最后在客户端调用:

**public** **class** **Client** **{**

**public** **static** **void** **main(**String**[]** args**)** **{**

AbstractClerk clerk **=** **new** Clerk**();**

AbstractClerk leader **=** **new** Leader**();**

AbstractClerk manager **=** **new** Manager**();**

clerk**.**setSuperior**(**leader**);**

leader**.**setSuperior**(**manager**);**

*//有人借款 10000 元*

clerk**.**approveRequest**(new** BorrowRequest**(**10000**));**

*//有人借款 111000 元*

clerk**.**approveRequest**(new** BorrowRequest**(**111000**));**

**}**

**}**

运行结果：

这时候如果添加了新的职位老版，他的额度为 1000000 元，那么我们只需创建一个新的具体职员类继承抽象职工类即可。

**class** **Boss** **extends** AbstractClerk**{**

**public** **Boss()** **{**

**super.**type **=** "老版"**;**

**}**

**public** **int** **getLimit()** **{**

**return** 1000000**;**

**}**

**}**

**public** **class** **Client** **{**

**public** **static** **void** **main(**String**[]** args**)** **{**

AbstractClerk clerk **=** **new** Clerk**();**

AbstractClerk leader **=** **new** Leader**();**

AbstractClerk manager **=** **new** Manager**();**

AbstractClerk boss **=** **new** Boss**();**

clerk**.**setSuperior**(**leader**);**

leader**.**setSuperior**(**manager**);**

manager**.**setSuperior**(**boss**);**

*//有人借款 10000 元*

clerk**.**approveRequest**(new** BorrowRequest**(**10000**));**

*//有人借款 111000 元*

clerk**.**approveRequest**(new** BorrowRequest**(**111000**));**

**}**

**}**

运行结果：