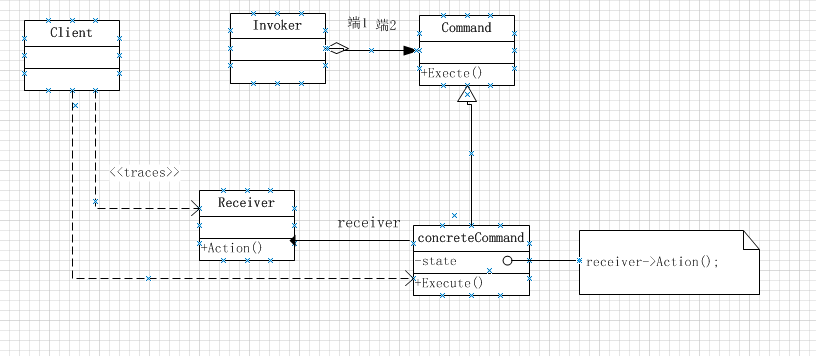
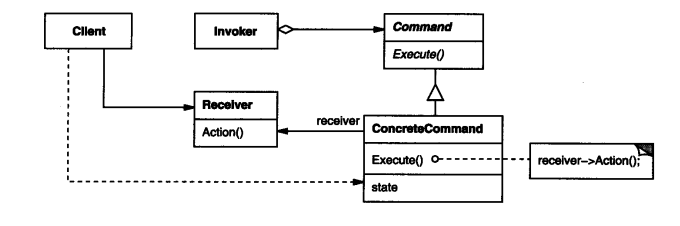
# Undo/Redo实现设计

## 框架设计





框架基本工作原理：

客户根据具体需要创建一个具体的命令类（ConcreteCommand）,并为其指定接收类，命令管理类（Invoker），要求该命令执行这个命令请求（Execute函数），最终接收者(Receiver)会执行具体的行为（Action）.

* Command

声明执行操作的接口

* ConcreteCommand

将一个接收者绑定于一个动作

调用接收者相应的操作，以实现Execute.

* Clinet

创建一个具体的命令对象并设定它的接收者。

* Invoker

要求该命令执行这个请求

* Receiver

知道如何实施与执行一个请求相关的操作。任何类都可以作为一个接收者。

## 框架详解

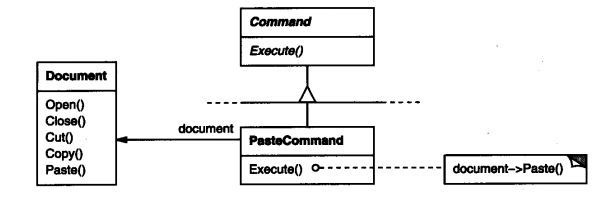
针对undo/redo，一共分为三个部分，即：请求封装成命令类；命令的管理;命令的接受者。

### 请求封装成命令类(Command)

将请求封装成类，从而使你可用不同的请求对客户进行数据化，如：可以储存命令，删除命令等操作。

同时可扩展，当想新增加一个Command子类时，只用继承Comand类即可。

如, pasteCommand支持从剪贴板向一个文档 ( D o c u m e n t )粘贴正文。 PasteCommand的接收者是一个文档对象，该对象是实例化时提供的。Execute操作将调用该Documen的Paste操作。



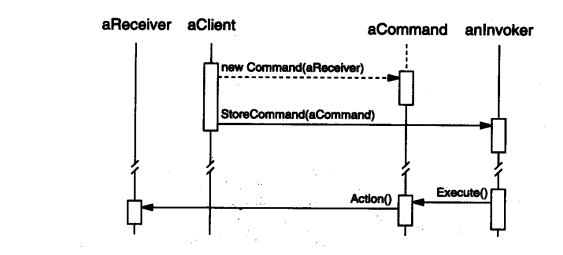
### 命令的管理(Invoker)

为实现Undo/Redo，在命令管理类中维护着2个栈（Undo Stack和Redo Stack），并增加相应的操作栈的私有方法。同时也会实现命令的调用操作。

### 命令的接受者类（Receiver）

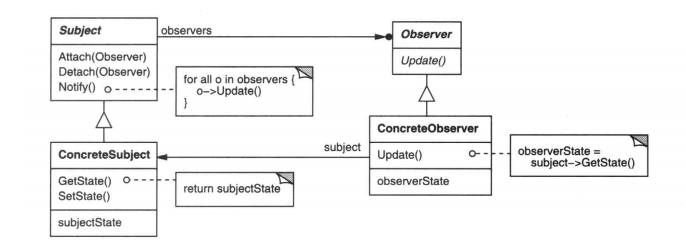
一个命令的真正执行在Receiver类中。如上图的Document类，命令的真正要实现的行为为document->Paste();同时任何类都可以成为一个接收者。

## 对象之间的交互



* Client创建一个ConcreteCommand对象并指定它的Receiver对象。
* 某Invoker对象存储该ConcreteCommand对象。
* 该Invoker通过调用Command对象的Execute操作来提交一个请求。若该命令是可撤消的，ConcreteCommand就在执行Execute操作之前存储当前状态以用于取消该命令。
* ConcreteCommand对象对调用它的Receiver的一些操作以执行该请求。

# Observe结构



支持Undo/Redo的应用程序，一般都有“Undo”和“Redo”两个按钮，那么当命令的Undo/Redo栈内容变化的时候，两个按钮会根据是否“可撤销”和“可恢复”相应地变化为Enabled或Disabled状态。那么框架支持这个功能是通过观察者（Observer）设计模式来完成的。

简单的说，当CommandManager的Undo和Redo栈由空变为不空时，或由不空变为空时，都会调用Subject的Notify函数通知所有观察者，来更新UI（比如：Undo/Redo按钮的Enabled/Disabled状态）。

• S u b j e c t（目标）

— 目标知道它的观察者。可以有任意多个观察者观察同一个目标。

— 提供注册和删除观察者对象的接口。

• O b s e r v e r（观察者）

— 为那些在目标发生改变时需获得通知的对象定义一个更新接口。

• C o n c r e t e S u b j e c t（具体目标）

— 将有关状态存入各C o n c r e t e O b s e r v e r对象。

— 当它的状态发生改变时 , 向它的各个观察者发出通知。

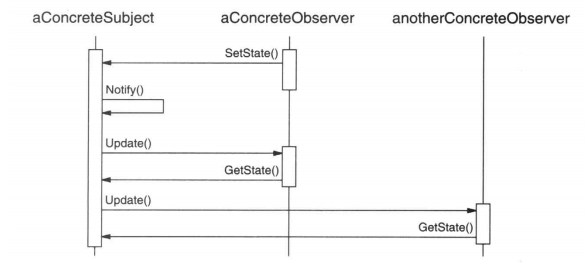
• C o n c r e t e O b s e r v e r（具体观察者）

— 维护一个指向C o n c r e t e S u b j e c t对象的引用。

— 存储有关状态，这些状态应与目标的状态保持一致。

— 实现O b s e r v e r的更新接口以使自身状态与目标的状态保持一致。

## 交互图



# CommandManager类做成单件

CommandManager实现为单件，即整个整个程序中只有一份CommandManager类的实例，降低与系统框架的耦合性，保持独立的设计。