行为模式

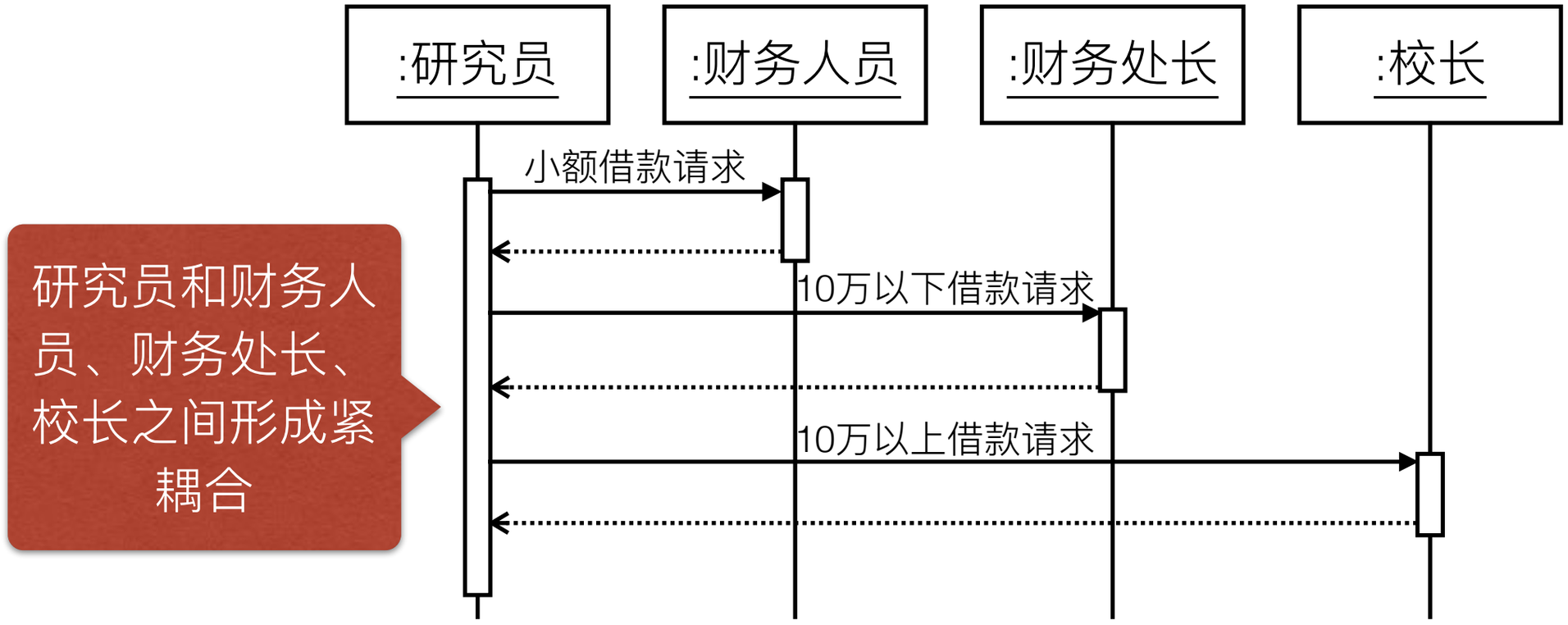
Behavioral Patterns

概述

- 行为模式关注对象之间的通信
  - 重在分解控制流中的耦合性

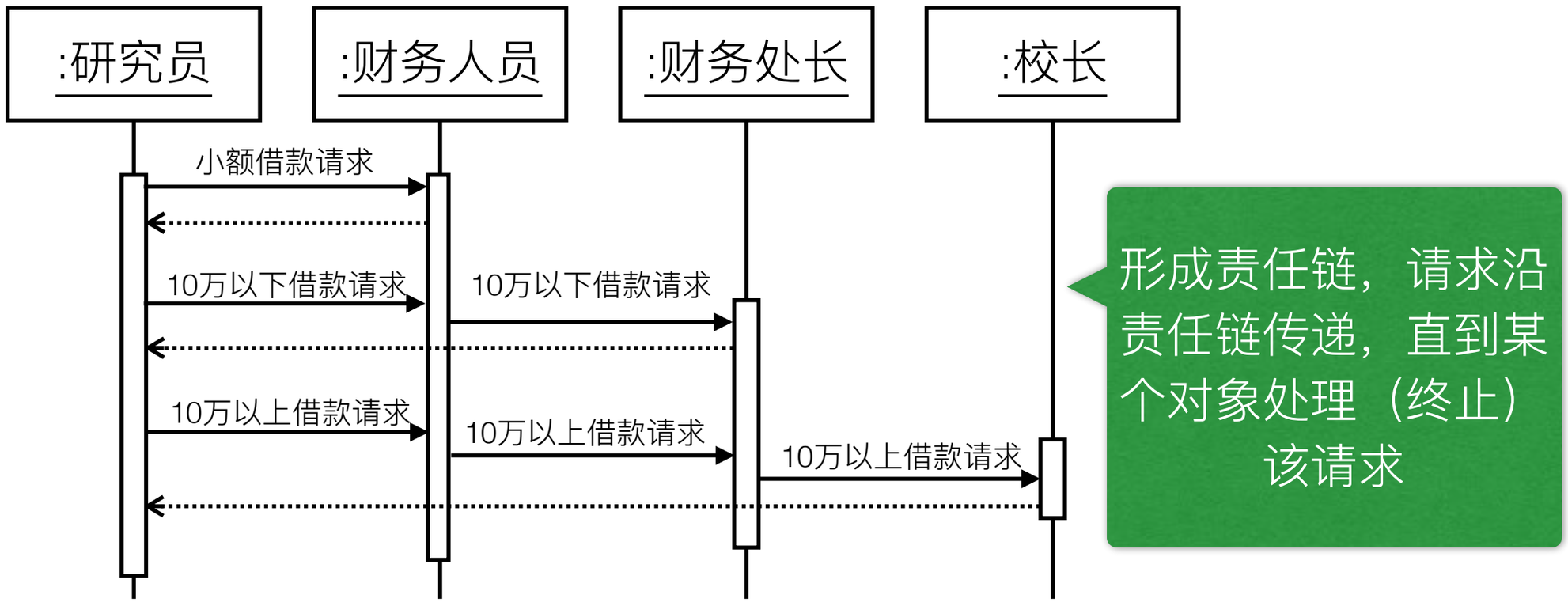
责任链
Chain of Responsibility

- 意图:
  - 通过为多个对象提供相应请求的机会，避免请求发送者和请求接受者之间的耦合



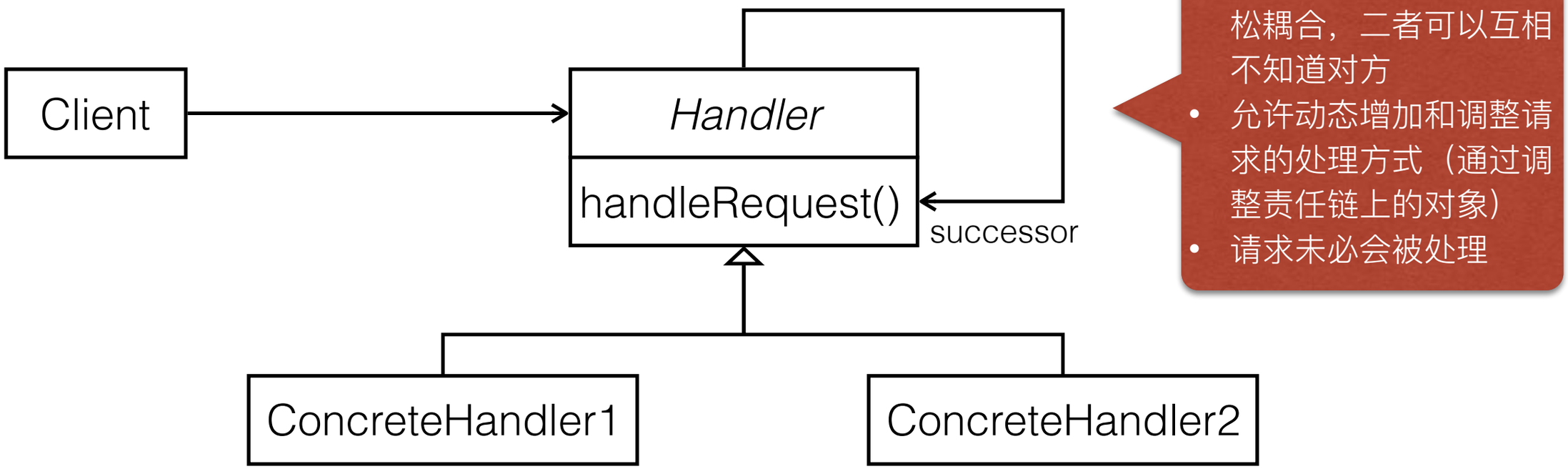
责任链
Chain of Responsibility

- 科研工作者眼中的理想状态



责任链
Chain of Responsibility

- 结构



- 使用场景
  - 多个对象都可以处理一个请求，且他们之间互相不知道优先次序
  - 向一组对象发送请求，而不明确指定对象的接受者

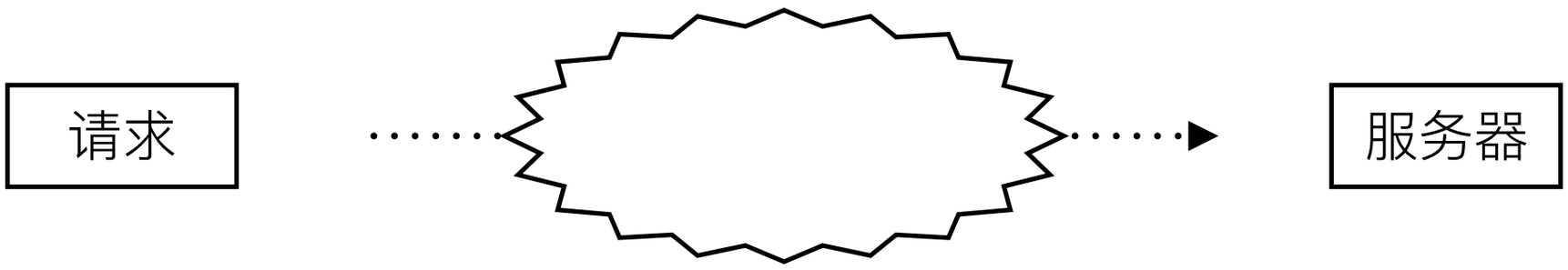
案例研究

- 网页需要响应鼠标事件。从事件响应的角度，如何设计事件响应模型，使得每个DOM元素都有机会响应鼠标事件？从浏览器的角度，如何发送这个事件（发送给谁），才能让该事件被正确处理?

责任链可以是一条线、一棵树，或者更复杂的结构

案例研究

- 客户的访问请求在被实际处理之前需要经过一系列验证和预处理。验证过程负责检查请求的合法性，预处理过程负责规范化请求数据。假设不同的客户具有不同的访问权限（导致检查过程有所不同）。同时随着系统运行，在发现新的非法请求后，需要在不停机的状态下增加 / 修改验证和规范化操作。如何设计以保证整个模块的灵活性?



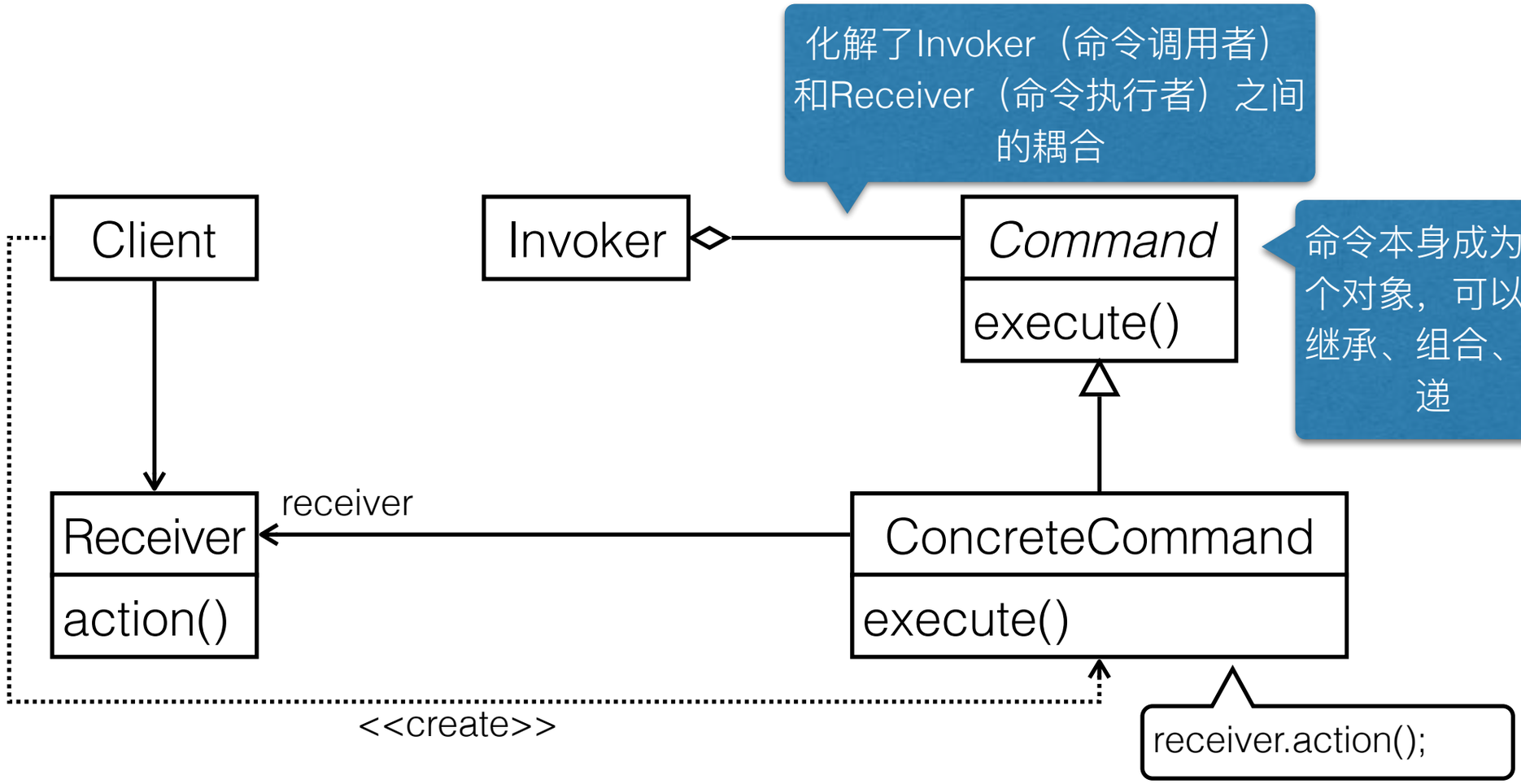
命令模式
Command

- 意图:
  - 将请求封装成一个对象，以便易于排序、记录、回滚请求处理过程，并利用该请求配置其它对象

列举现实当中的命令模式

命令模式
Command

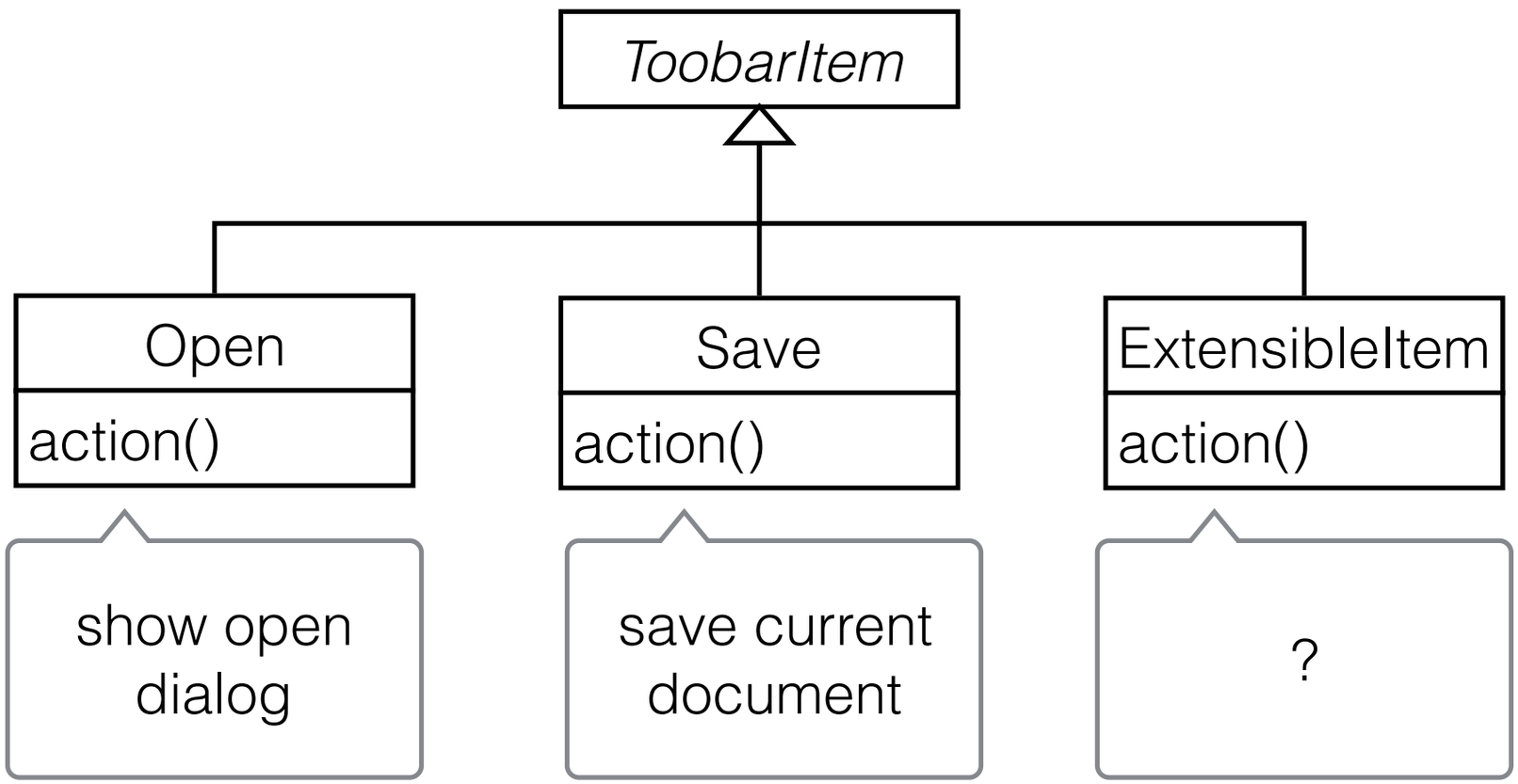
- 结构



- 使用场景
  - 通过一个命令去配置一个对象的行为 / 功能
  - 在不同的时刻构造、排序、执行命令
  - 支持回滚、支持记录
  - 允许通过组合基本命令构造复杂系统行为

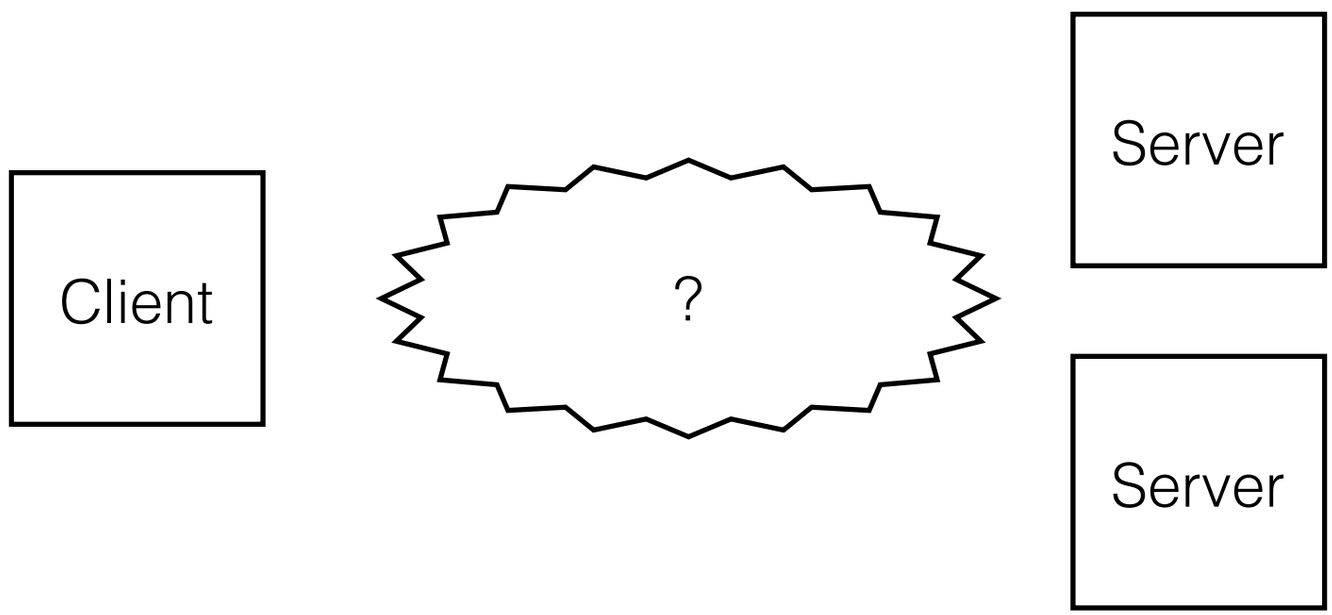
案例分析

- 某程序允许动态扩充工具栏按钮，以便增加某些快捷操作。假设所增加的快捷操作是不固定的（用户可配置），那么如何设计这个工具栏模块?



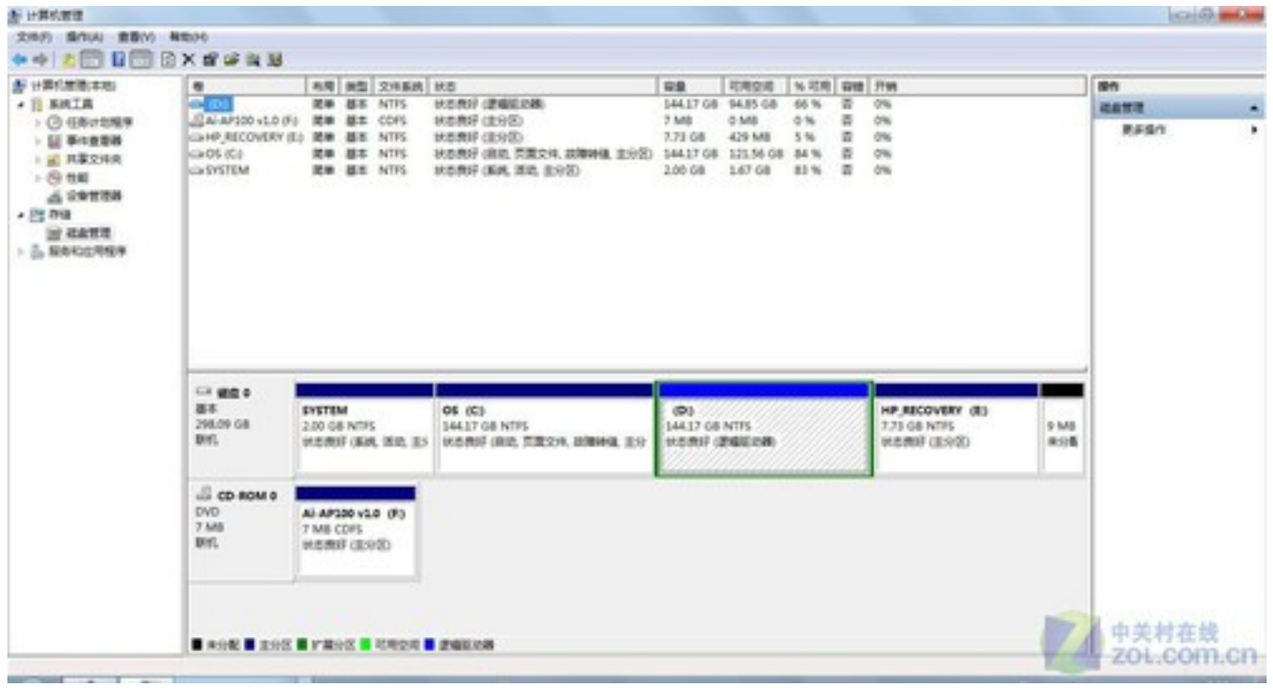
案例分析

- 一个CS系统采用多个并行server的设计。client需要通过请求的方式调用server的功能。每个server虽然都能支持相同的业务功能，但具有不同的IP地址、请求格式，它们所提供的基本操作也各不相同（但总体上都能实现相同的业务功能）。请问，如何设计才能支持这样的CS结构?



案例分析

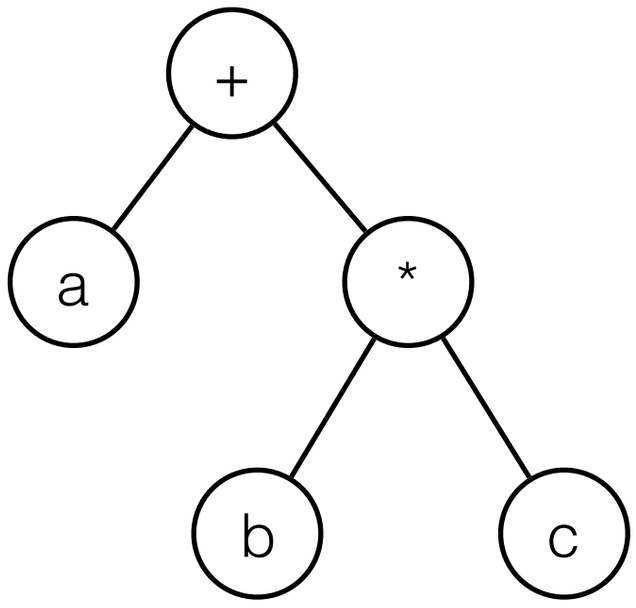
- 磁盘管理工具为用户提供调整磁盘空间、格式化、修改盘符等功能。怎样设计该工具才能保证最佳性能并能够验证错误操作?



解释器模式
Interpreter

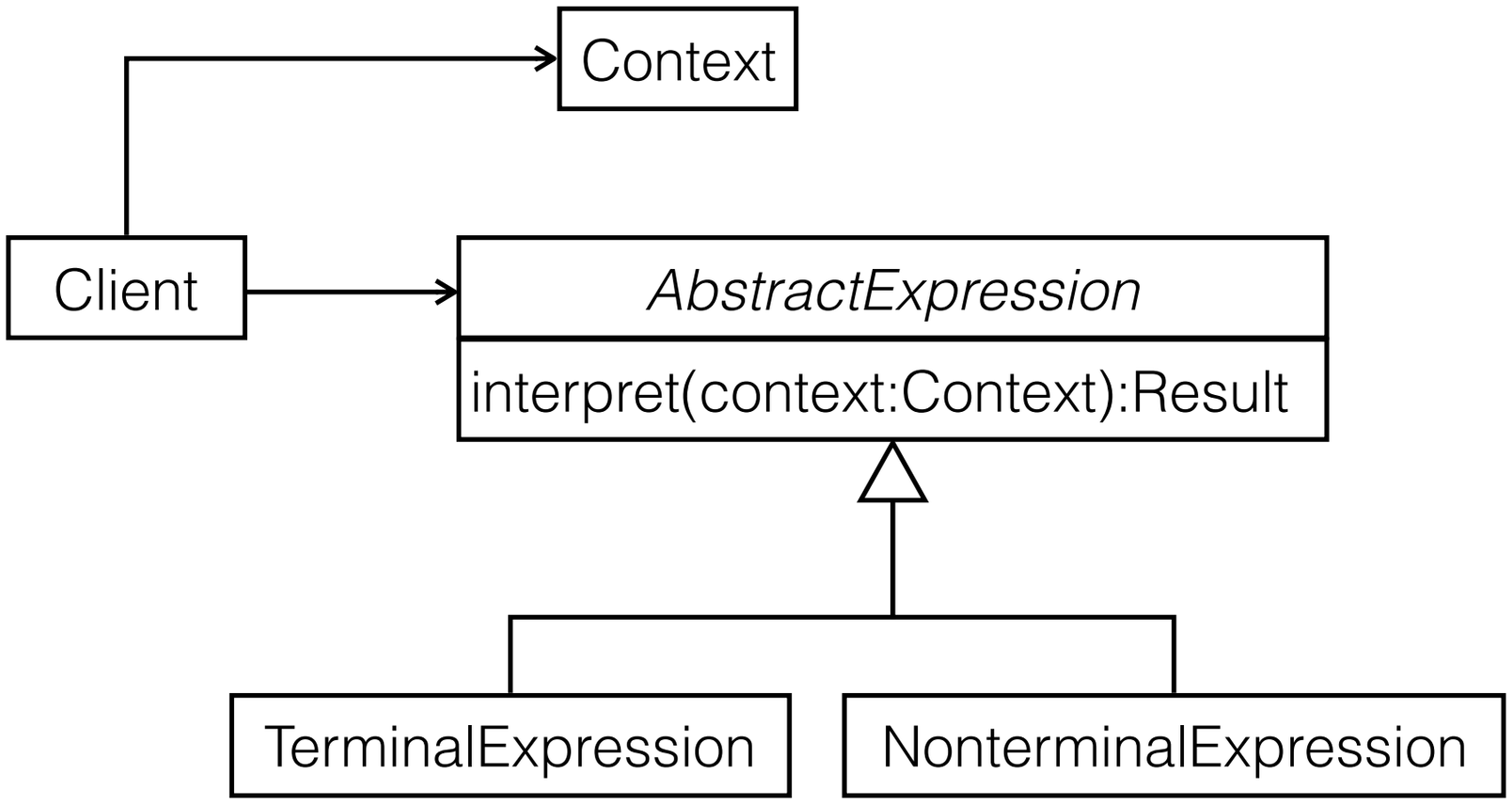
- 意图
  - 对于一种语言，定义一种该语言的语法表示，基于该语法表示可以利用一个解释器解释该语言的语句。

a+b*c

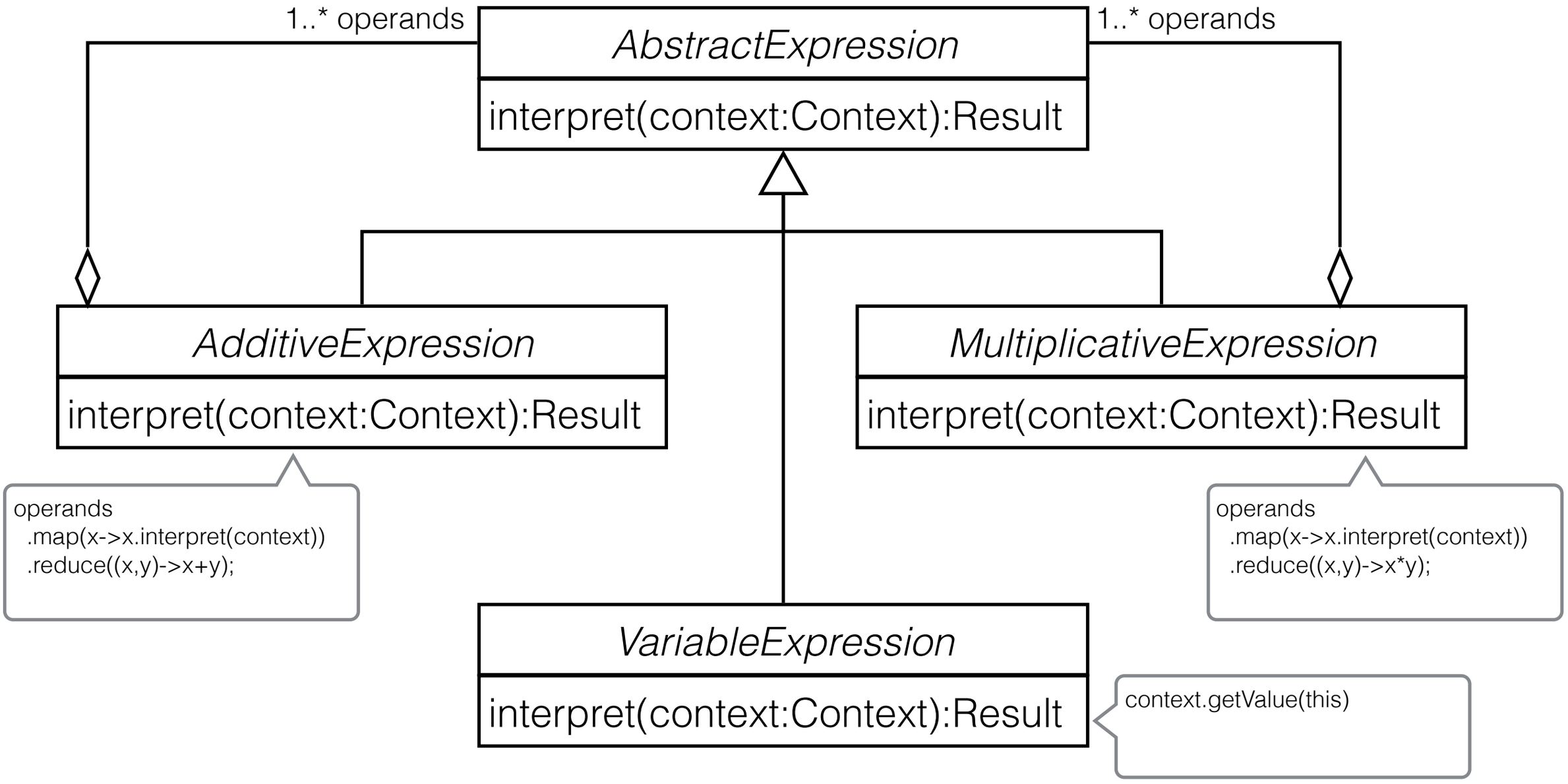


解释器模式
Interpreter

- 结构



解释器模式
Interpreter



案例研究

- 设计一个构件，能够将HTML网页转换成一个DOC文档。

- 设计一个构件，能够将一个右值是复杂表达式的赋值语句拆分成多个右值是简单表达式的赋值语句。

f=a+b*c

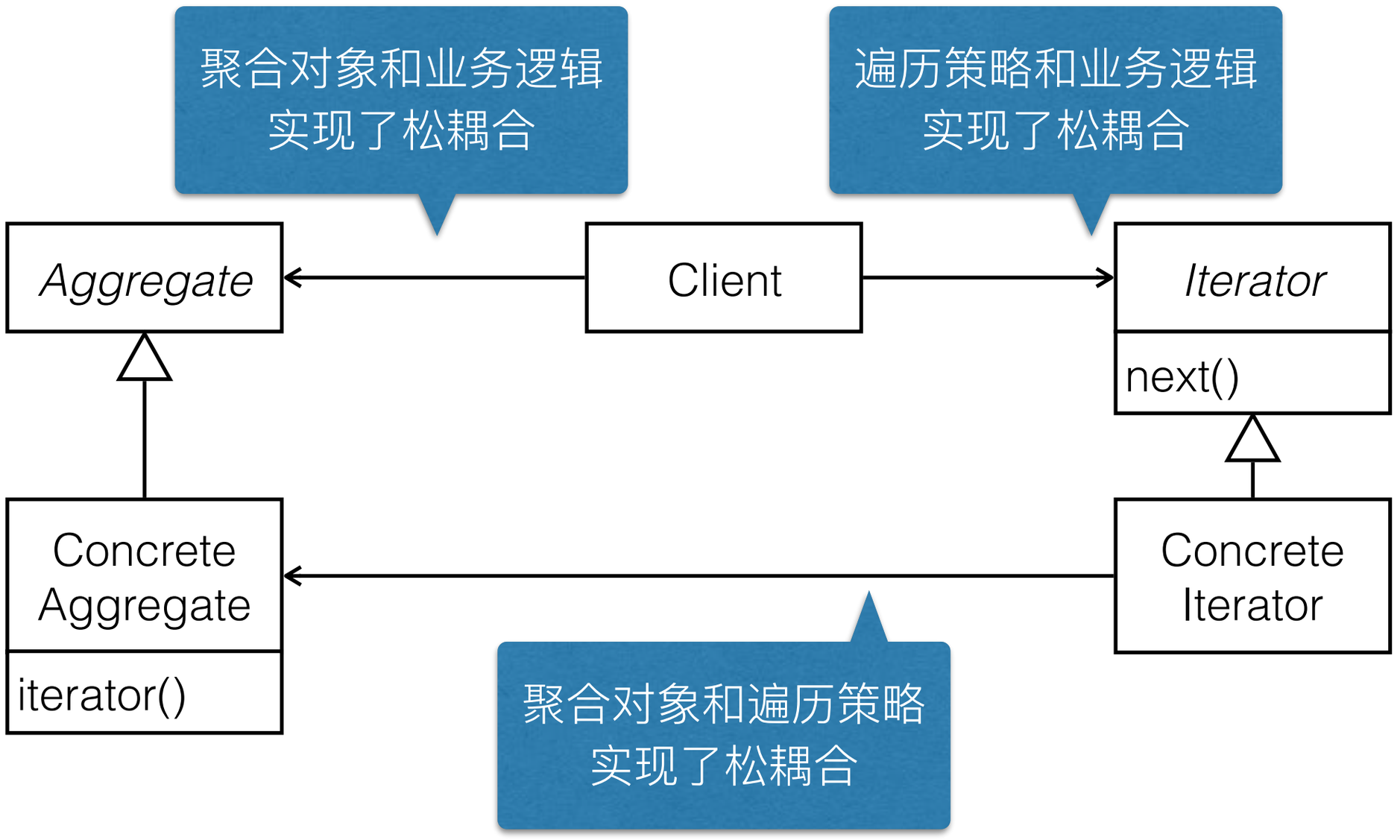
t0=b*c;
f=a+f0;

迭代器模式
Iterator

- 意图
  - 提供一种顺序访问聚合对象内部元素的方法，且不暴露聚合对象的内部结构
- 思考：如果没有……迭代器模式
  - 从前往后，打印数组中每个元素的字面值
  - 从后往前，打印链表中每个元素的字面值
  - 按照中序打印二叉树上每个节点的字面值

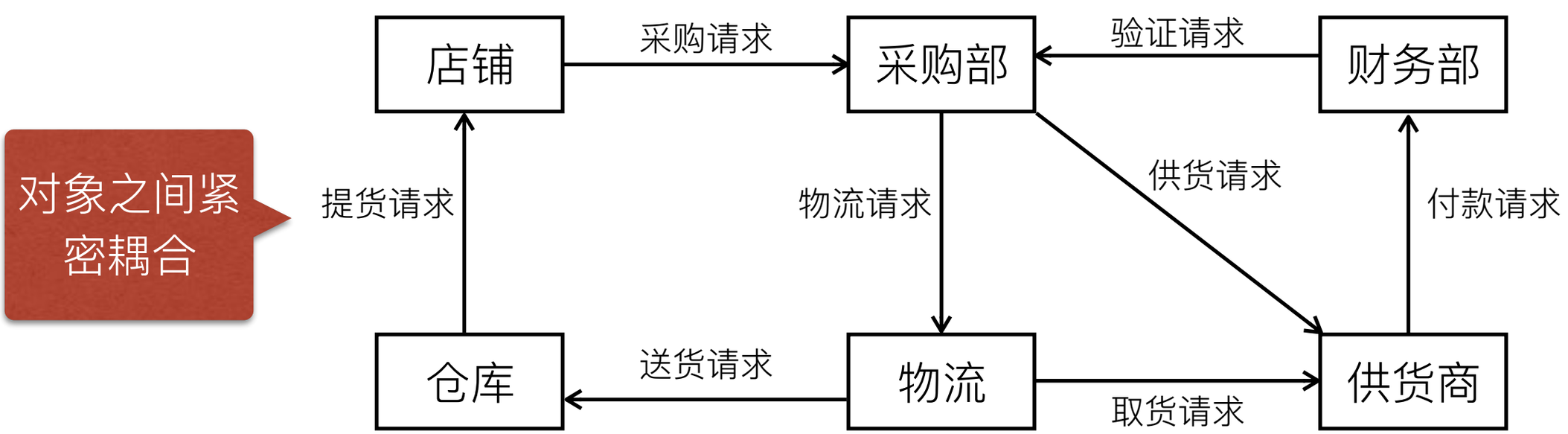
迭代器模式
Iterator

- 结构



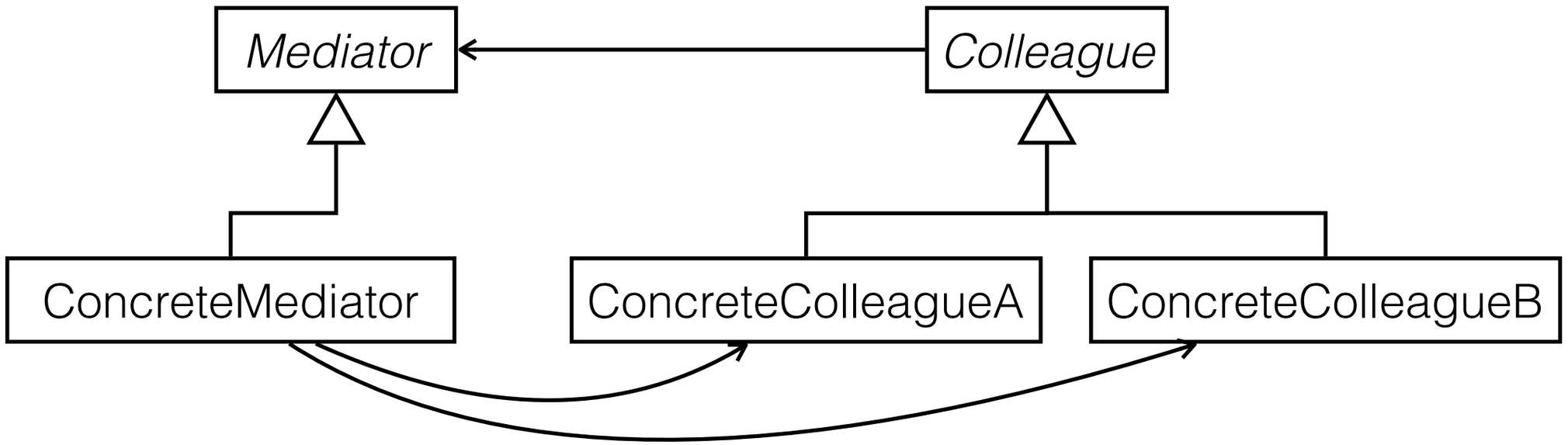
中介者模式
Mediator

- 意图
  - 将一组对象的交互封装成一个对象，使得这组对象之间的耦合度降低，并允许进一步改变这组对象的交互方式



中介者模式
Mediator

- 结构



- 使用场景与效果
  - 多个对象之间存在复杂的交互
  - 该模式能够简化每个对象的交互协议，降低耦合
  - 该模式能够集中控制多个对象

案例研究

- 某数据分析软件的一个界面包含五个界面元素：文件输入框，文件选择列表，分析命令输入框，分析命令选择框，执行按钮。其交互关系如下：
    - 文件输入框中输入文件名时，文件选择列表会自动筛选可能的文件
    - 在文件选择列表中选择一个文件时，该文件名会被自动填充到文件输入框中
    - 分析命令输入框中输入分析命令名时，分析命令选择列表会自动筛选可能的命令
    - 在分析命令选择列表中选择一个命令时，该命令名会被自动填充到分析命令输入框中
    - 当文件输入框中包含一个合法文件名时，会根据该文件扩展名筛选可应用的分析命令
    - 当输入的文件名有效，且分析命令有效时，执行按钮可以被点击

案例研究

- 事件驱动系统
  - Android / iOS应用都是事件驱动的。通常，一个控件会注册一组事件监听器，用于捕获和处理某种事件。这种模式使得事件的处理分散到各个控件中。当这些控件需要相互协作时，这样的处理机制可能会导致意想不到的复杂性。
  - 但反过来讲，事件驱动系统能够更好地支持异步和并发请求

备忘录模式
Memento

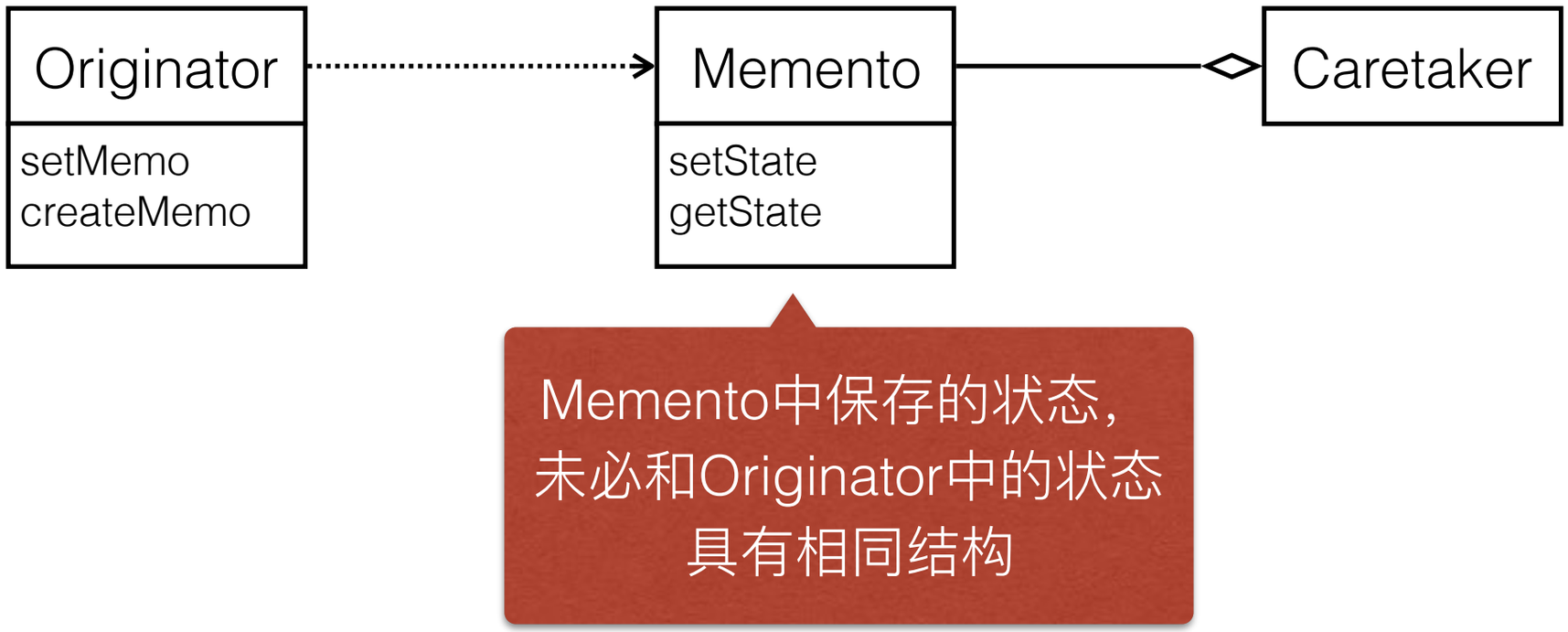
- 意图
  - 在不破坏对象封装性的前提下，捕获和外化对象的内部状态，以便将来进行状态回滚和恢复

当一个需要回滚的操作是不可逆的

ArrayList是一个使用定长数组作为基础数据结构的不定长集合类。当添加一个元素后，集合的长度大于现有数组长度时，需要重新分配一块更大的内存空间，并将原有元素拷贝到新的空间。从对象内部状态来看，增加元素和删除元素不是各自的逆函数

备忘录模式
Memento

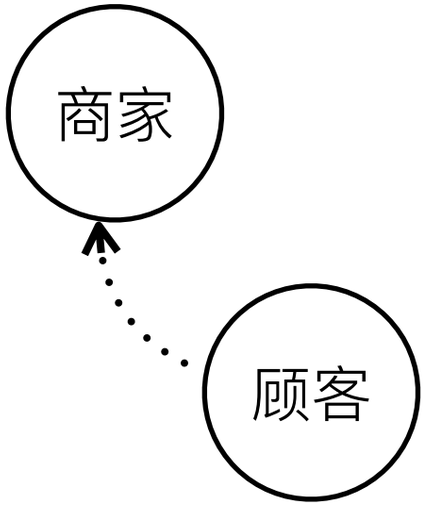
- 结构



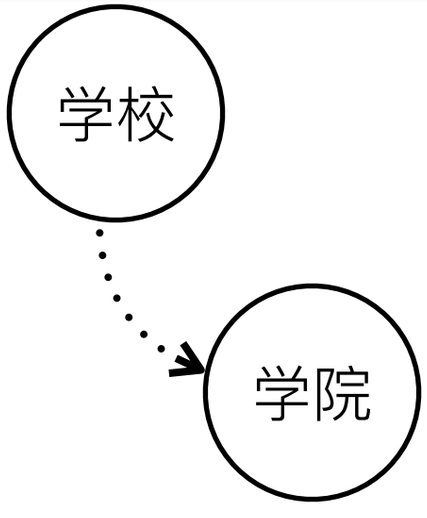
观察者模式
Observer

- 意图
  - 定义一种一到多的依赖关系，当一个对象更新后，所有依赖它的对象都会被自动通知

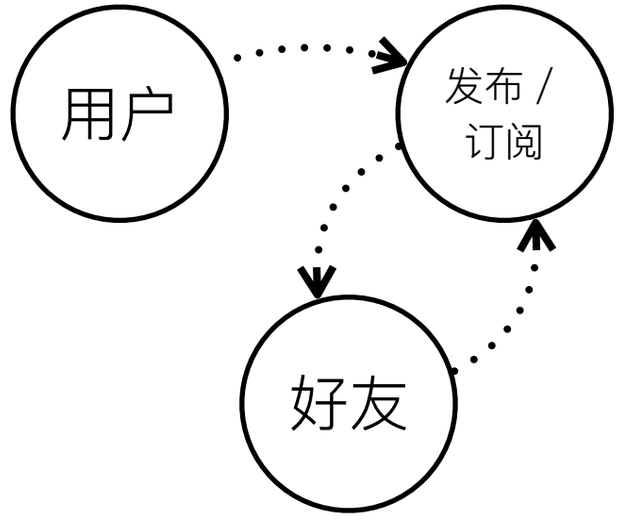
顾客随时查看商家发布的促销信息



学校新颁布的管理规定总是会传达给各个学院



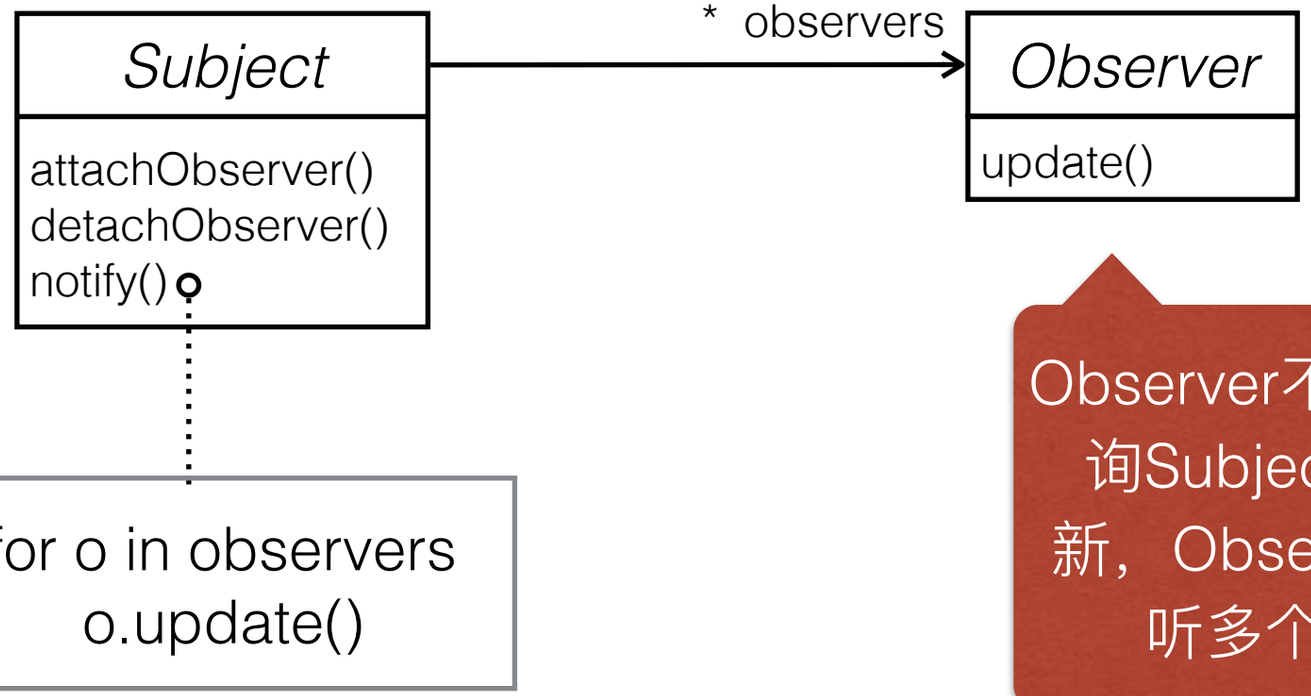
在朋友圈发布消息后，你的好友会自动得到通知



观察者模式
Observer

- 结构

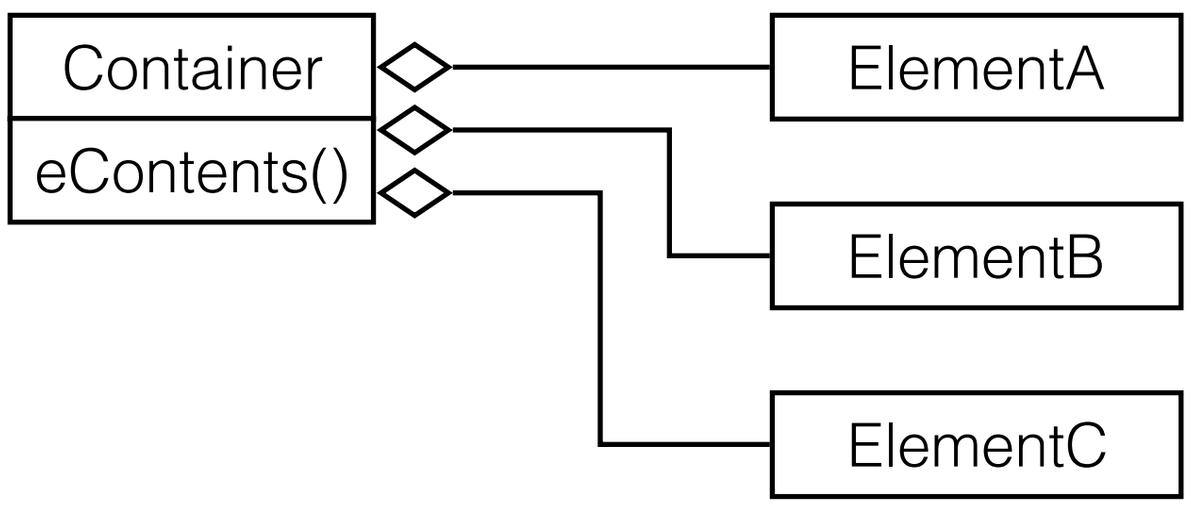
Subject不需要知道Observer的具体语义，也不需要主动寻找Observer的位置



Observer不需要主动查询Subject的状态更新，Observer 可以监听多个Subject

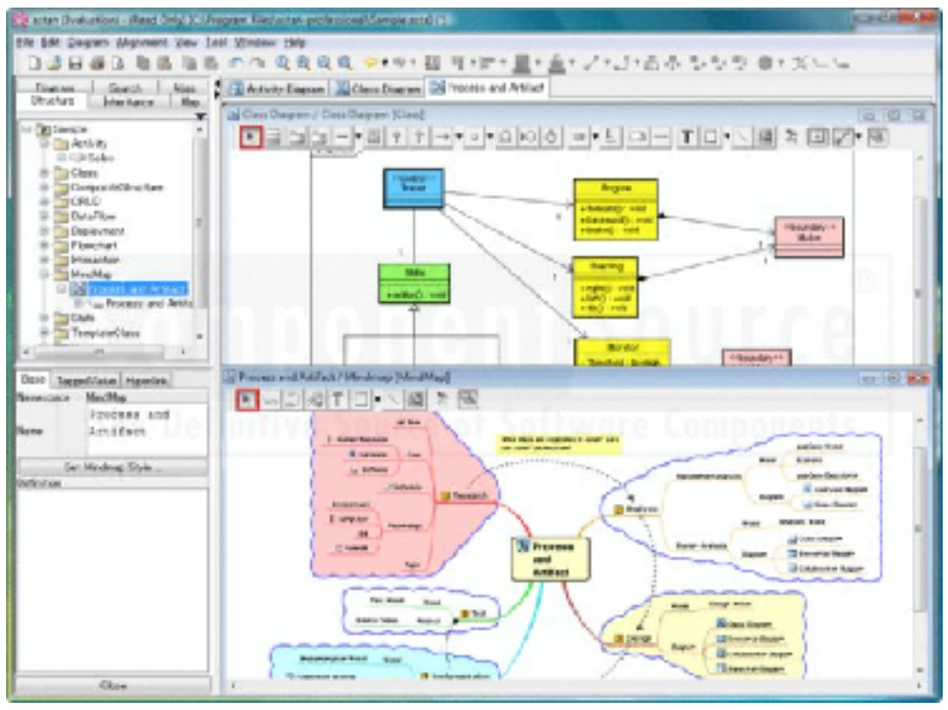
案例分析

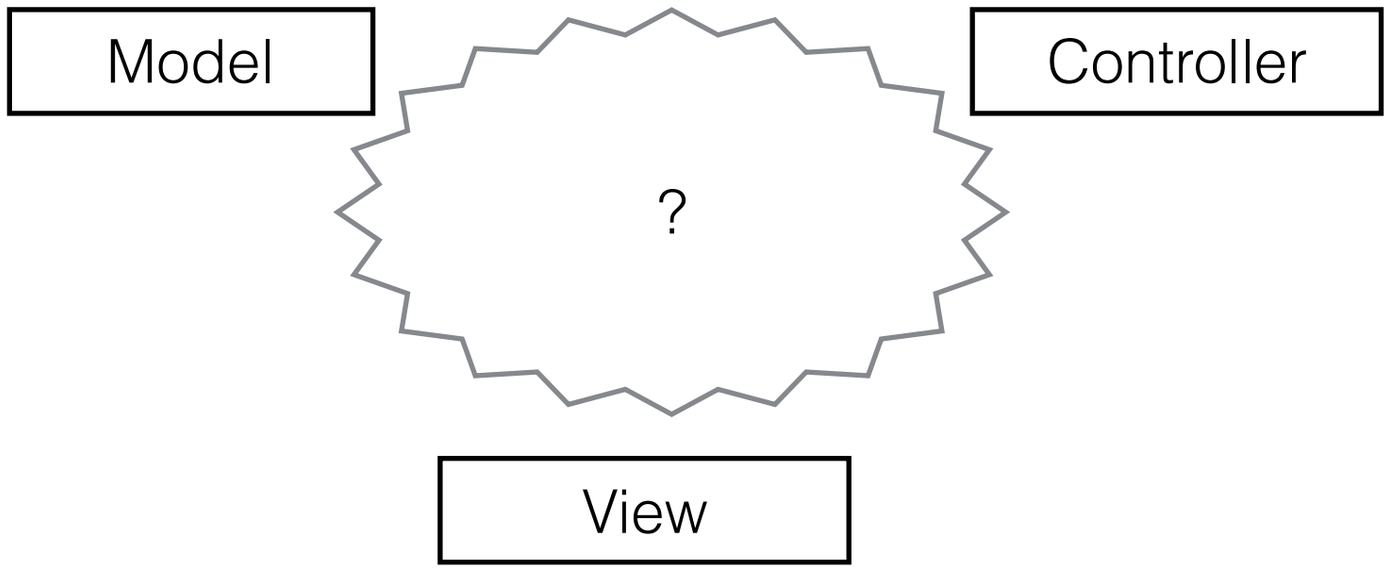
- 一个对象可以通过多个聚合关系拥有很多的子对象（部分对象），eContents()能够返回所有的子对象，不论它们是通过哪个聚合关系组合进来的。请问，如何高效地实现eContents方法。



案例分析

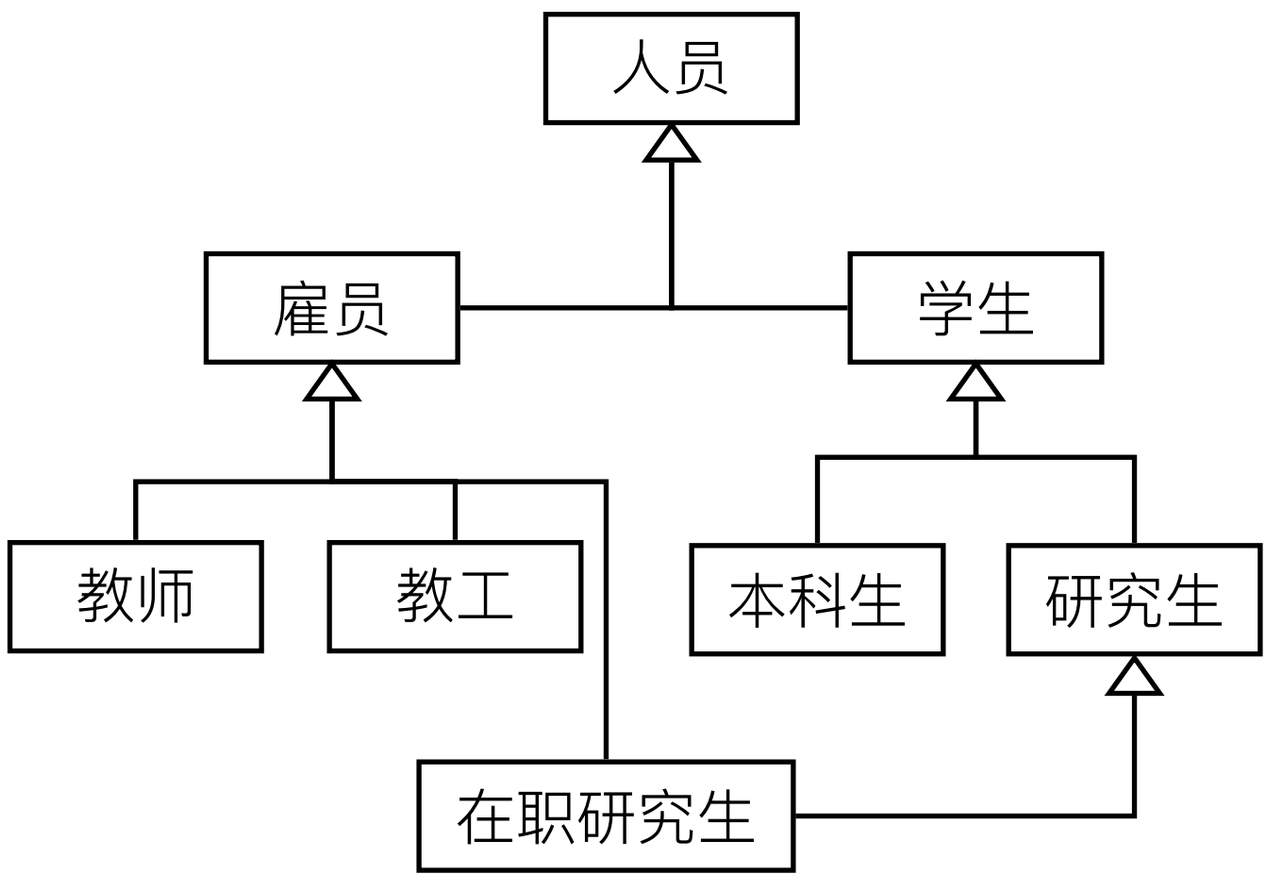
- 可视化编辑器中，用户可以通过编辑器主界面、大纲视图、属性视图等修改模型元素。模型元素修改后，视图元素会被更新。如何设计这样的MVC框架?





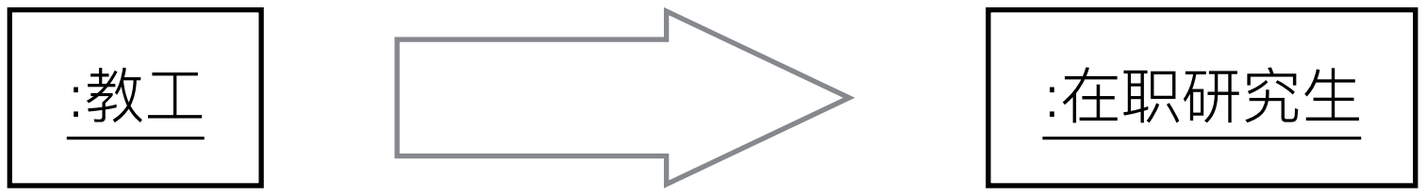
状态模式
State

- 意图：
  - 允许对象随着其内部状态的改变而（动态）改变其行为



假设一个教工对象在系统运行时从教工变成在职研究生对象，系统如何处理这一个变化？

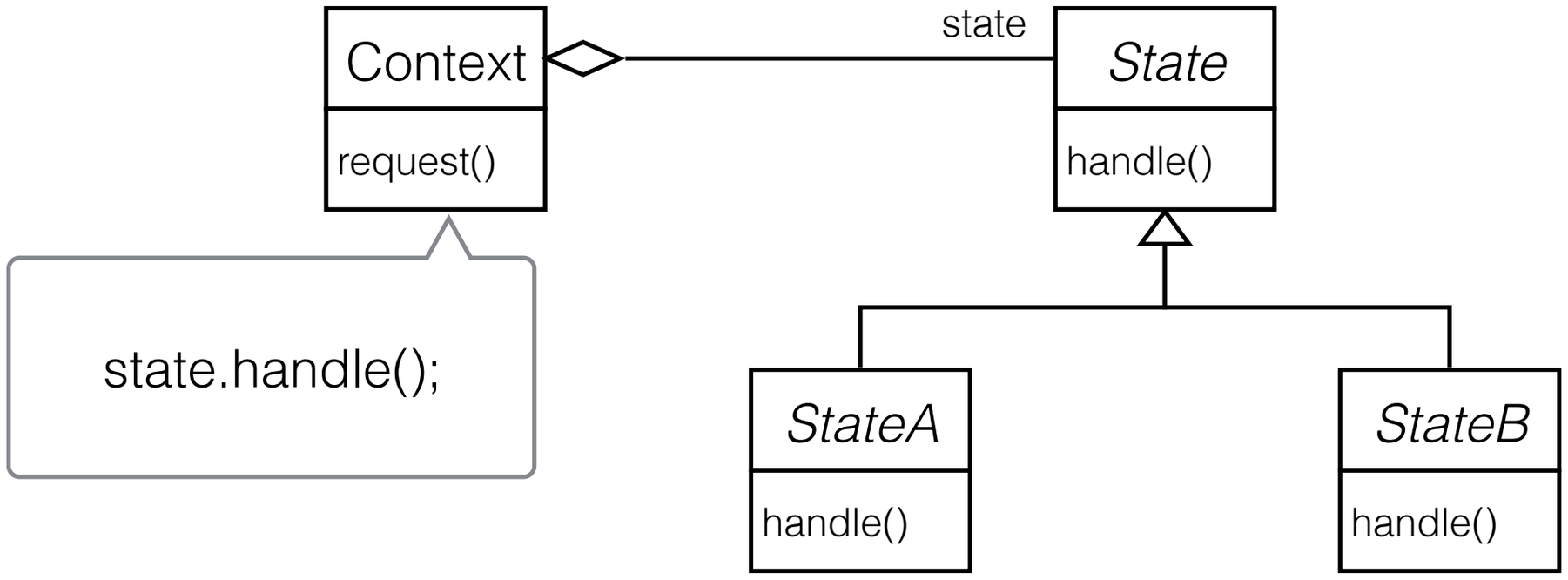
状态模式
State



如何在编程语言中将一个对象从一个类型改变成另一个类型?

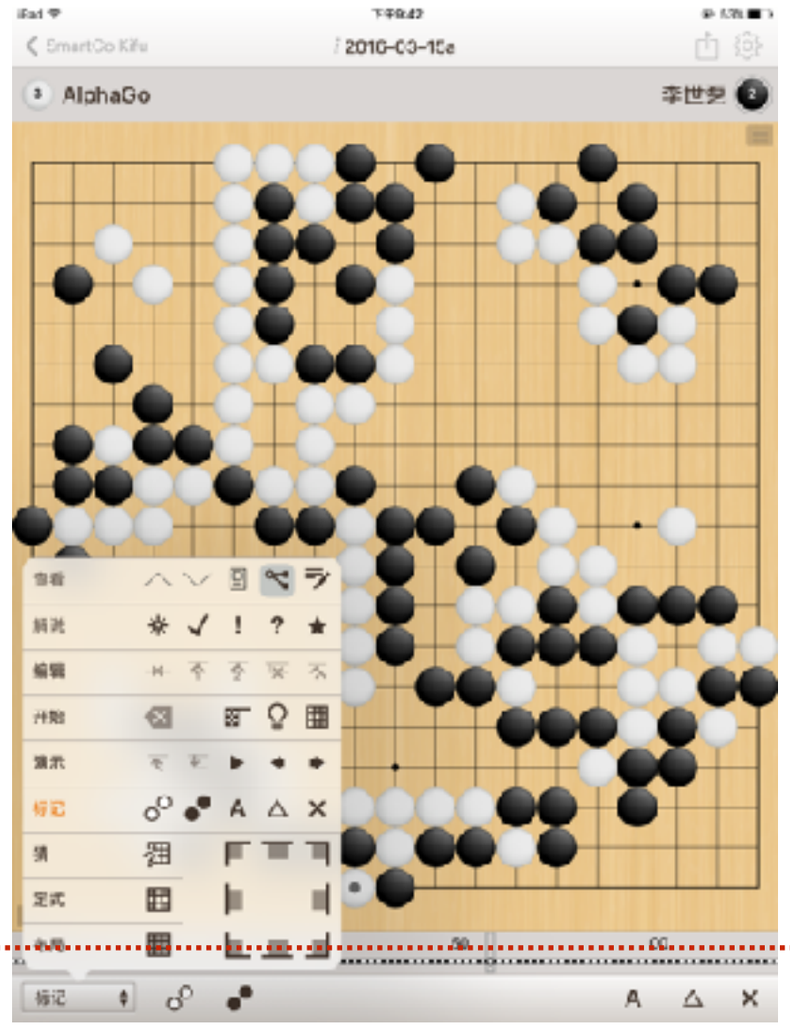
状态模式
State

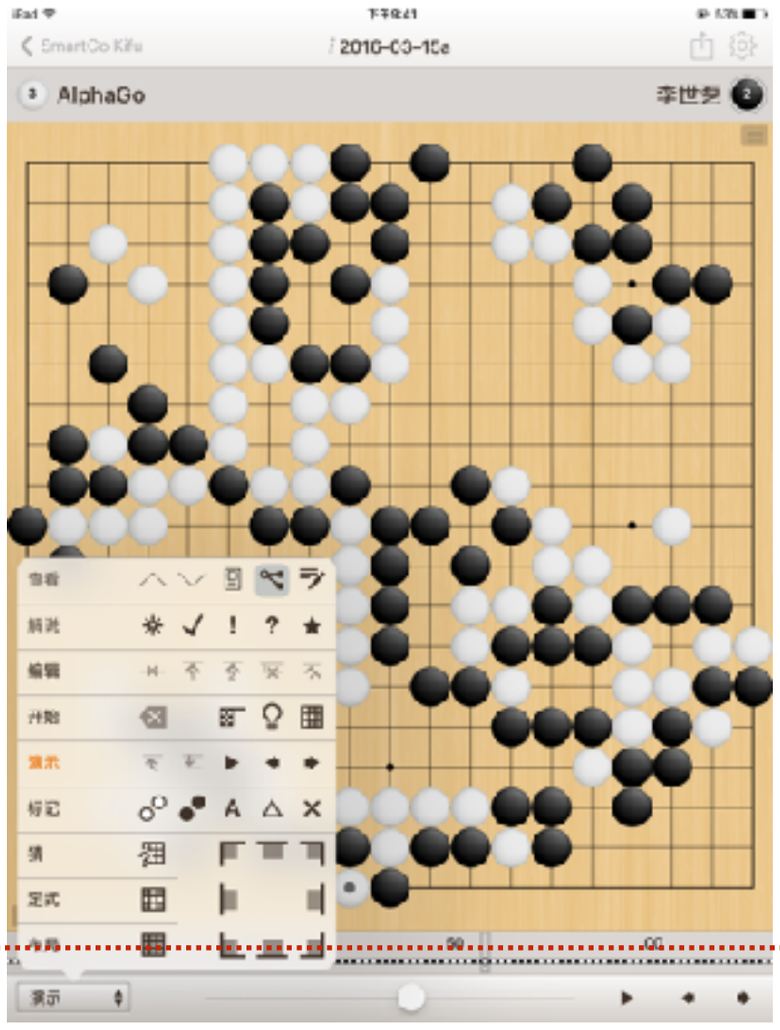
- 结构



案例分析

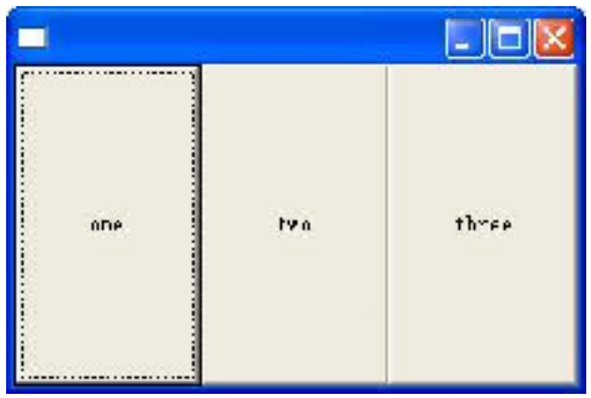
- SmartGo的控制台具有不同的模式，每种模式下拥有不同的操作

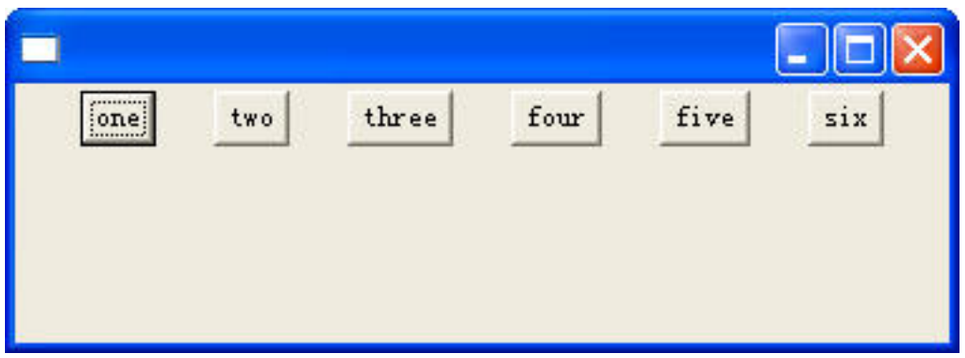


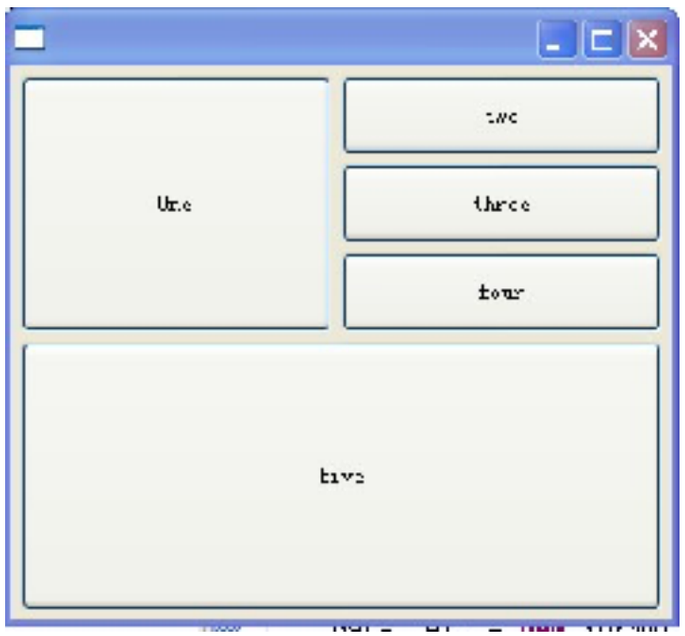


策略模式
Strategy

- 意图
  - 将某类算法定义成一个对象，使得它们可以相互替换

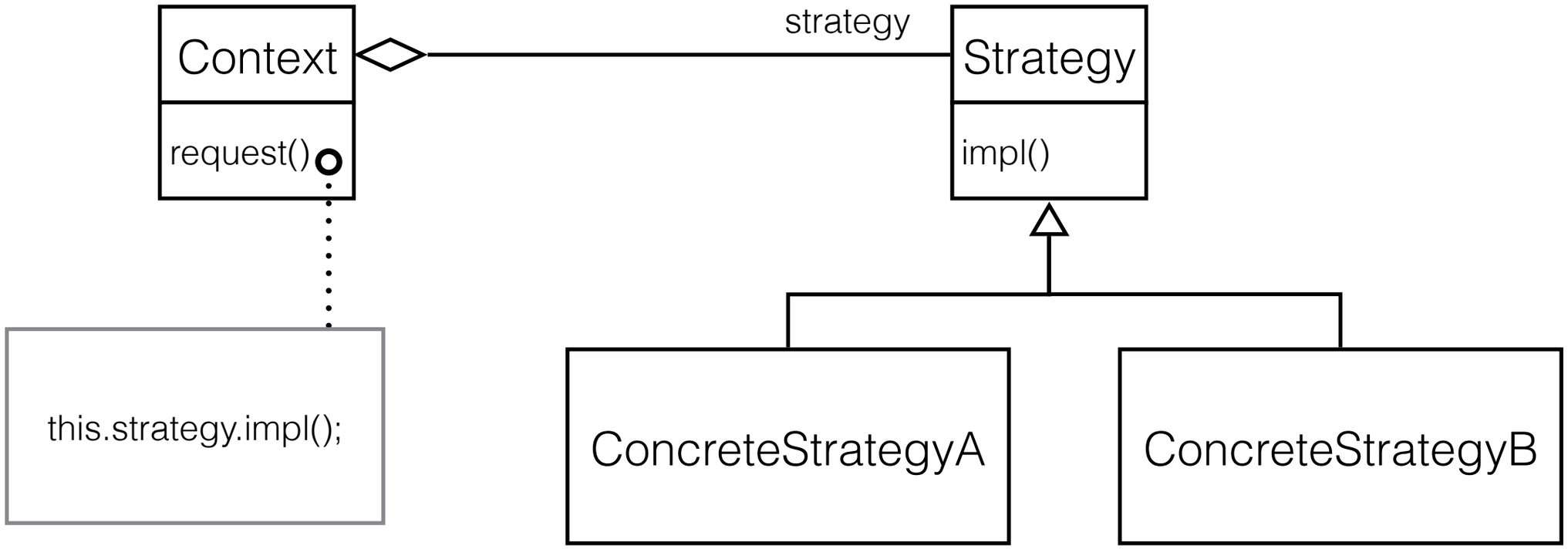






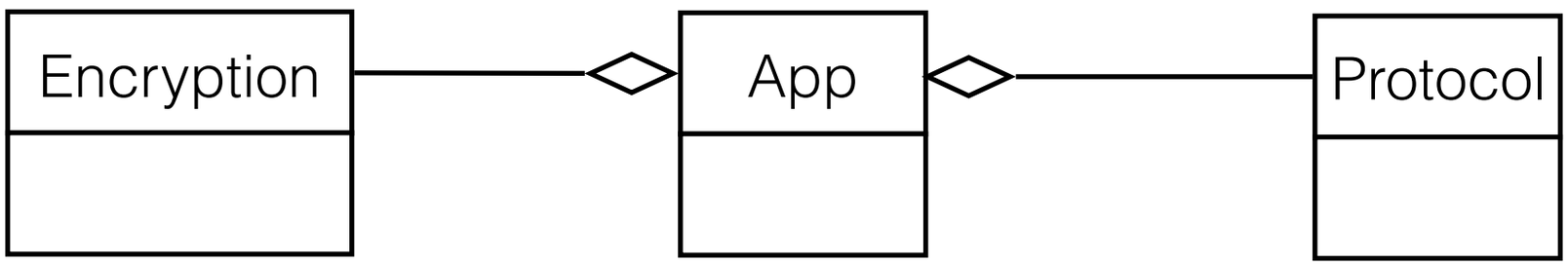
策略模式
Strategy

- 结构



案例分析

- 一个支持网络通信的软件系统需要能够支持不同的通信和加密协议



案例分析

- “王安顺同志因工作调动，向市人大常委会提出了辞去北京市市长职务的请求。同时，中共北京市委向市人大常委会提出推荐蔡奇同志任北京市 副市长、代理市长”

模版方法
Template Method

- 意图
  - 定义一个算法的骨架，将其中的某些步骤封装成方法，并交给子类去实现

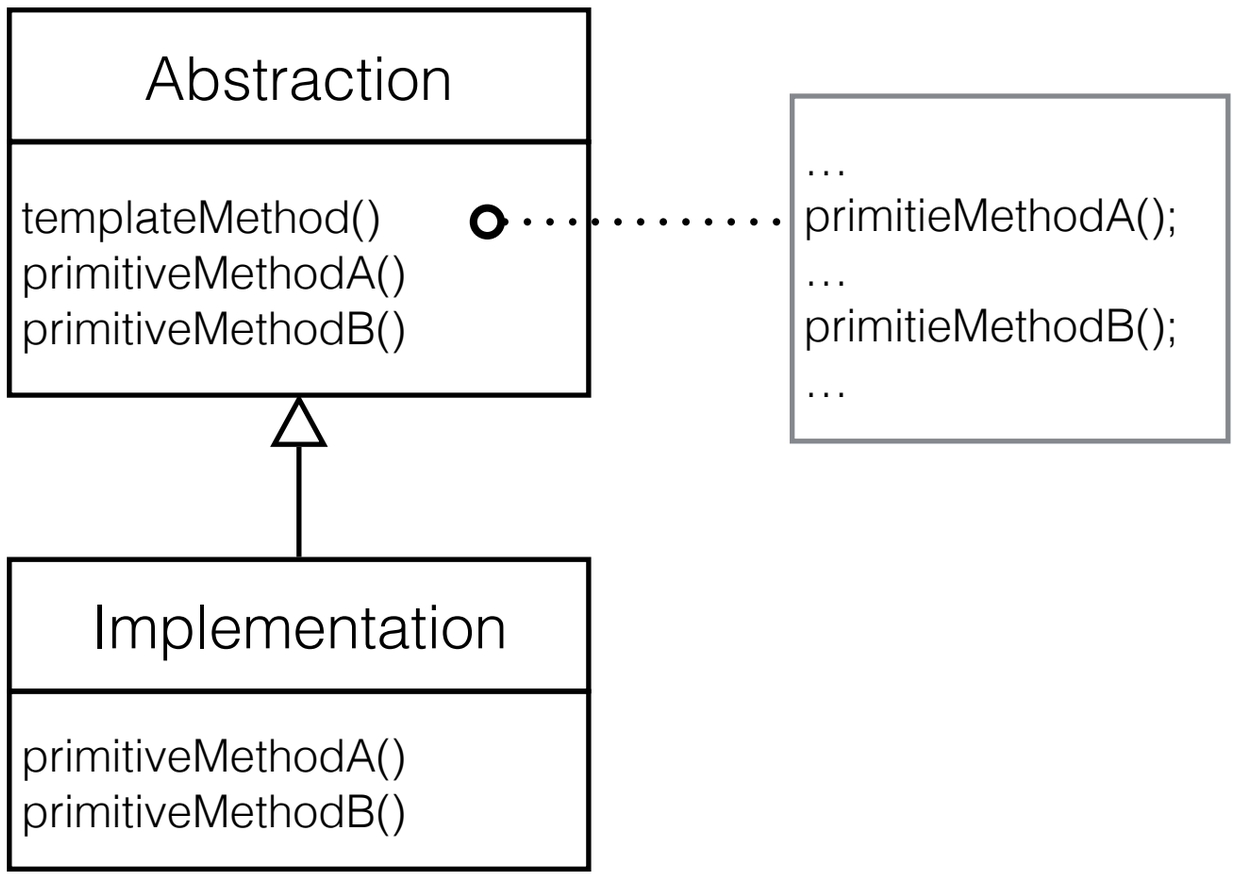
排序算法的基本过程可以看作是如下实现：

```
List<T> sort(List<T> list) {
  List<List<T>> subLists = split(list);
  List<List<T>> subResults = sortAll(subLists);
  List<T> result = merge(subResults);
  return result
}
```

如何基于上述算法骨架实现不同的排序算法?

模版方法
Template Method

- 结构



•何时使用?
- 当一个算法可以分解为可变部分和不可变部分时
- 当需要限制子类对于算法的扩展
- 具有潜在可并行化的算法
- 自顶向下的算法设计

•设计难点
- 识别算法中的基本操作
- 识别算法中的可变与不可变操作

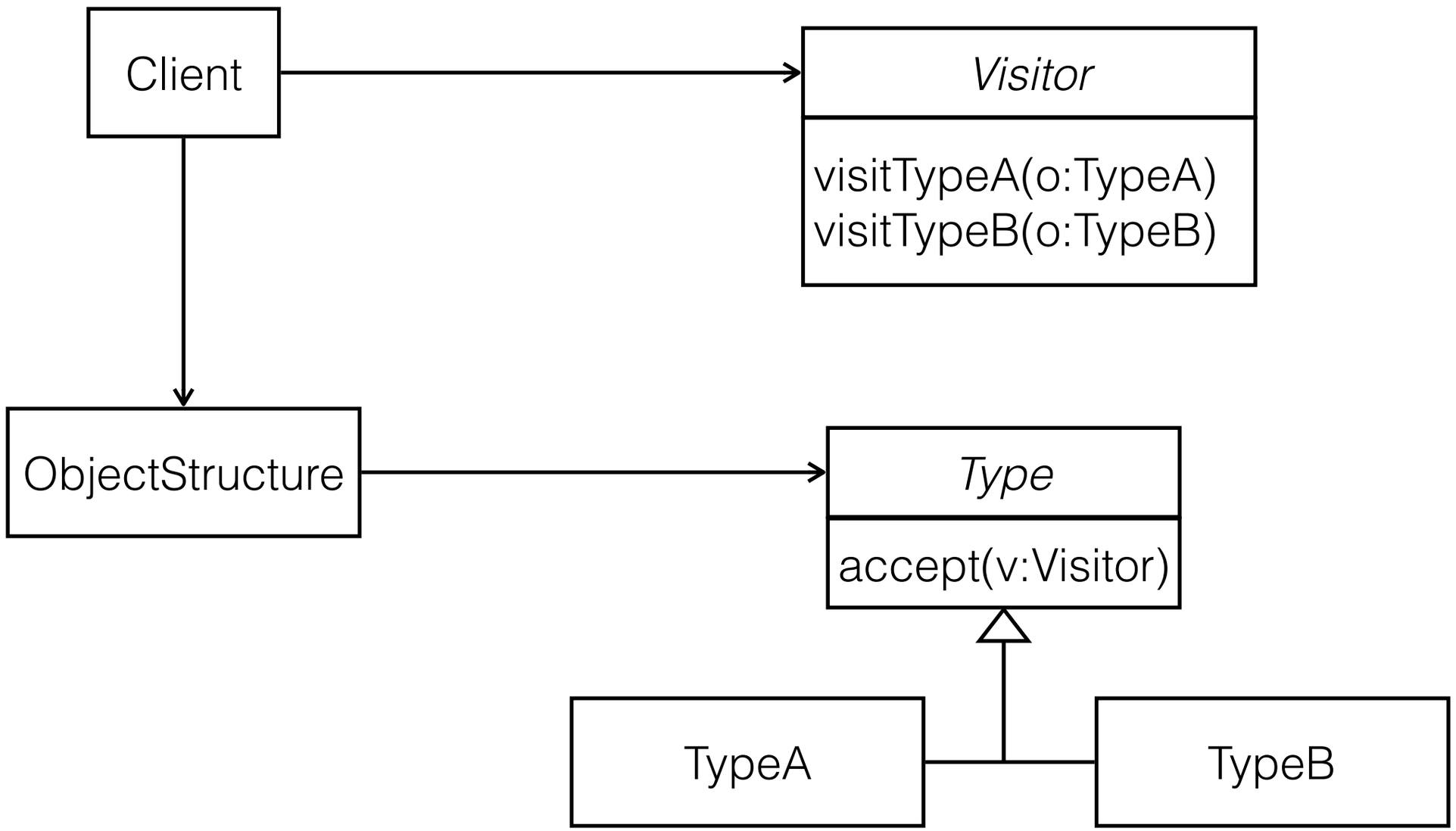
访问者模式
Visitor

- 意图
  - 将对象可以执行的操作定义为一个外部对象，从而能够在不修改原有对象的情况下扩展其行为

设计一个构件能够打印、检查、简化HTML文件

访问者模式
Visitor

- 结构



访问者模式
Visitor

- 行为