行为型设计模式
Behavioral Patterns

吴映波
wyb@cqu.edu.cn
虎溪Office: 虎溪学院楼410
A区Office：九教205
Tel：13594686661

Behavioral Patterns

- Chain of Responsibility
- Command
- Interpreter
- Iterator
- Mediator
- Memento
- Observer
- State
- Strategy
- Template Method
- Visitor

模式 12：Command(一)

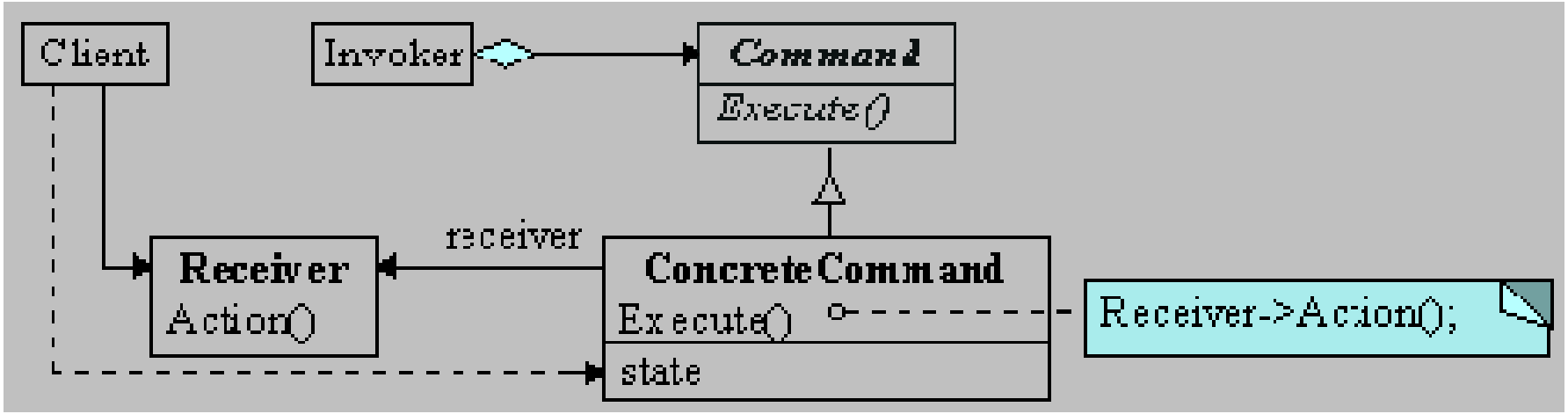
- Aliases
  - Action, Transaction
  - functor (function object)
- Intent
  - Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.
- Motivation
  - 把请求信息和请求执行过程封装起来
  - framework往往需要把命令请求与处理请求的对象分开，command模式可以把调用操作的对象与操作的目标对象分开
  - 允许通过多种途径调用同一个请求。——请求的重用

Command模式(二)

- Applicability：Use the Command pattern when：
  - **parameterize objects by an action to perform，代替回调**
  - **specify, queue, and execute requests at different times**
  - **support undo**
  - **support logging changes so that they can be reapplied in case of a system crash**
  - **structure a system around high-level operations built on primitives operations —— transactions**

Command模式(三)

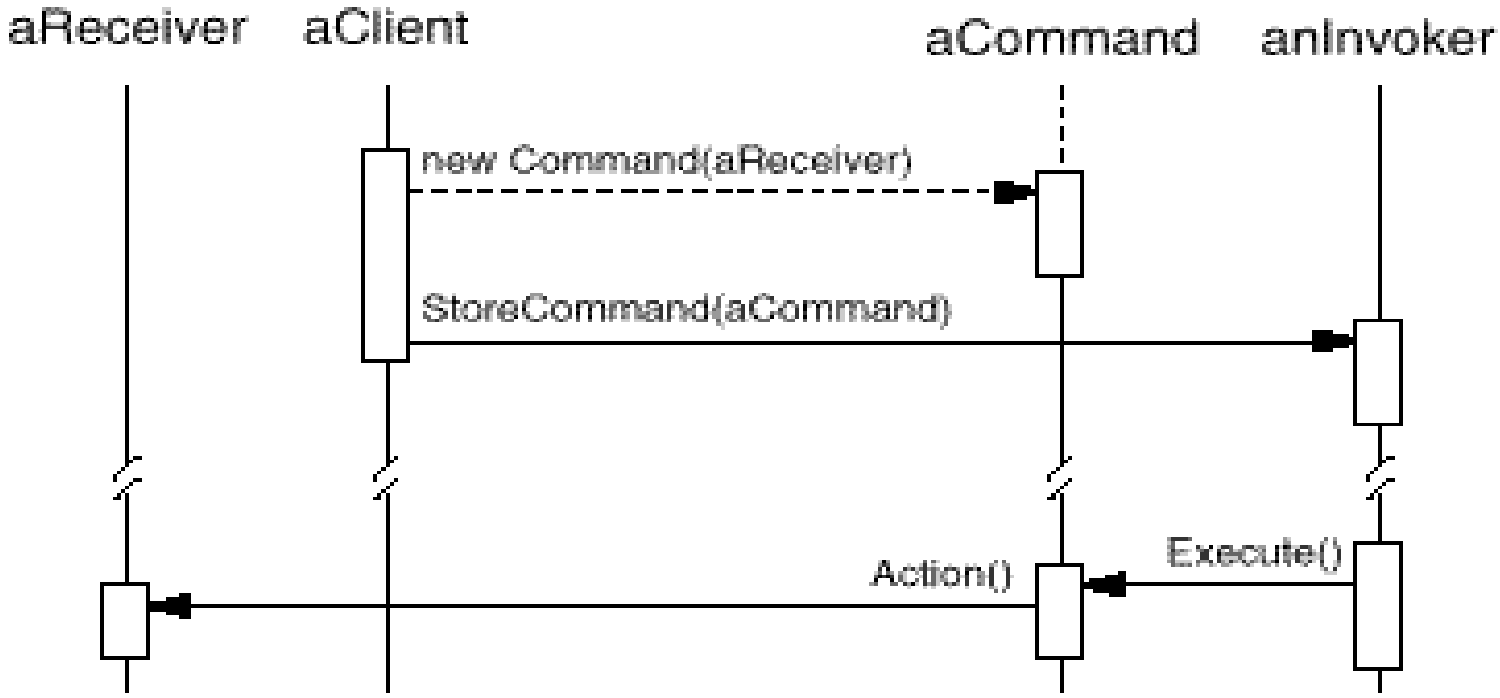
- Struct



- Participants
  - Client, Command、ConcreteCommand、Invoker、Receiver

Command模式(四)

- Collaborations

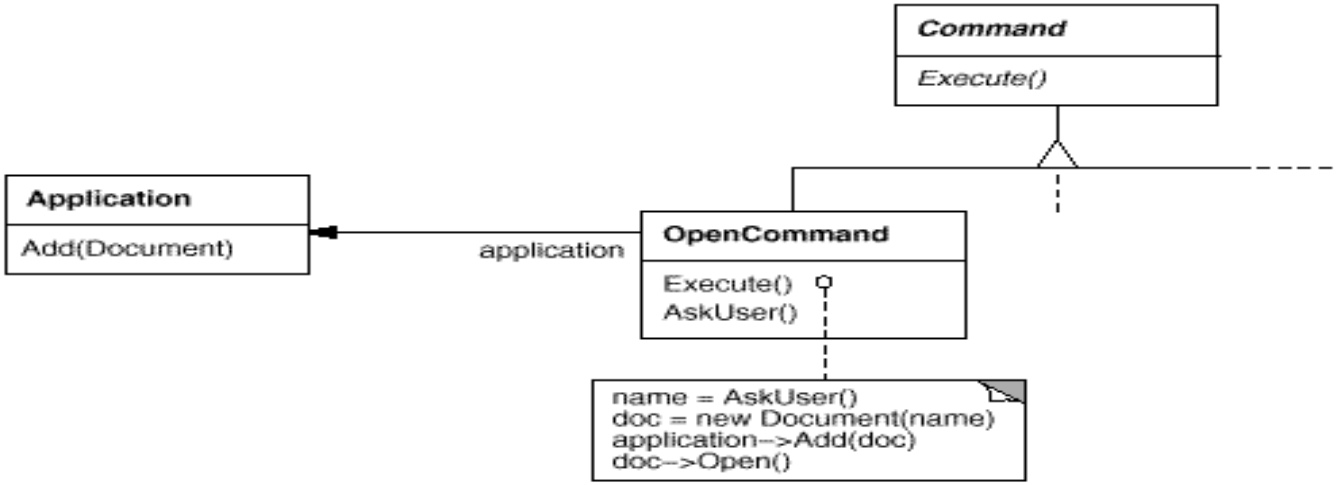


Command模式(五)

- Evaluation
  - Command decouples the object that invokes the operation from the one that knows how to perform it.
  - Commands are first-class objects. They can be manipulated and extended like any other object.
  - You can assemble commands into a composite command. An example is MacroCommand.
  - It's easy to add new Commands, because you don't have to change existing classes.
- Implementation
  - How intelligent should a command be?
  - Supporting undo and redo
  - Avoiding error accumulation in the undo process
  - Using C++ templates

Command模式(六)

- Related Patterns
  - Composite模式可用来实现command组合
  - 为实现undo/redo，可以用其他行为模式来管理状态，如memento模式。Command被放到history list之前，可以用prototype模式复制自身
- Exampl



模式 13：Iterator(一)

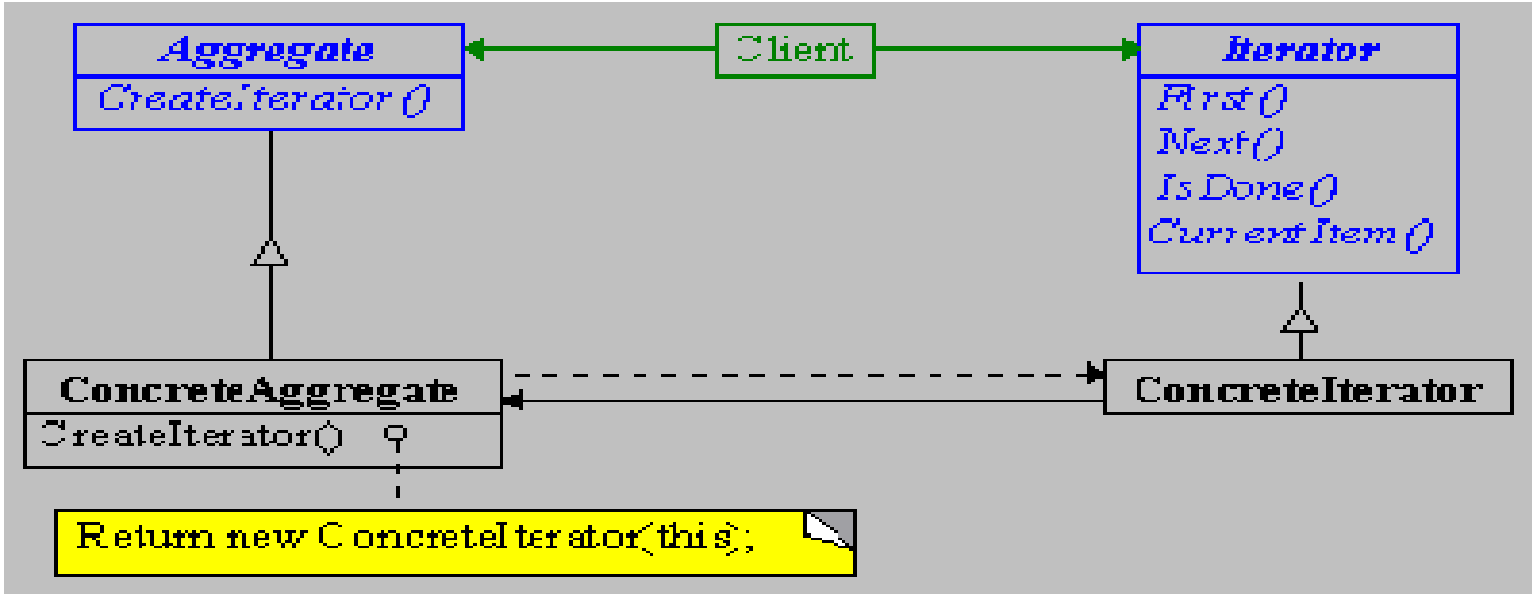
- Aliases： Cursor
- Intent
  - Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.
- Motivation
  - An aggregate object such as a list should give you a way to access its elements without exposing its internal structure.
  - Separating the traversal mechanism from the List object lets us define iterators for different traversal policies without enumerating them in the List interface.

Iterator模式(二)

- Applicability：Use the Iterator pattern when :
  - to access an aggregate object's contents without exposing its internal representation.
  - to support multiple traversals of aggregate objects.
  - to provide a uniform interface for traversing different aggregate structures (that is, to support polymorphic iteration).

Iterator模式(三)

- Struct



- Participants
  - Iterator、ConcreteIterator、Aggregate、ConcreteAggregate

Iterator模式(四)

- Evaluation
  - It supports variations in the traversal of an aggregate
  - Iterators simplify the Aggregate interface
  - More than one traversal can be pending on an aggregate
- Implementation
  - 实现可以非常灵活
  - Who controls the iteration?
    - external iterator *versus* internal iterator
  - Who defines the traversal algorithm?
    - Aggregate本身定义算法 —— Cursor mode
    - iterator定义算法 —— iterator如何访问数据
  - How robust is the iterator?

Iterator模式(五)

- Implementation(续)
  - **Additional Iterator operations.**
    - **基本操作：First, Next, IsDone, and CurrentItem**
  - **Using polymorphic iterators —— iterator资源释放**
  - **Iterators may have privileged access**
  - **Iterators for composites —— 适合于internal iterator或者cursor方式的iterator**
  - **Null iterators**

Iterator模式(六)

- Related Patterns
  - Composite：iterator常被用于composite模式的复合结构
  - Polymorphic iterators rely on factory methods to instantiate the appropriate Iterator subclass.

- Examples
  - COM enumerator：connectable object、…
  - ADO/OLE DB
  - C++ STL
    - 在STL中，iterator是连接algorithm和container的桥梁

模式 14：Observer(一)

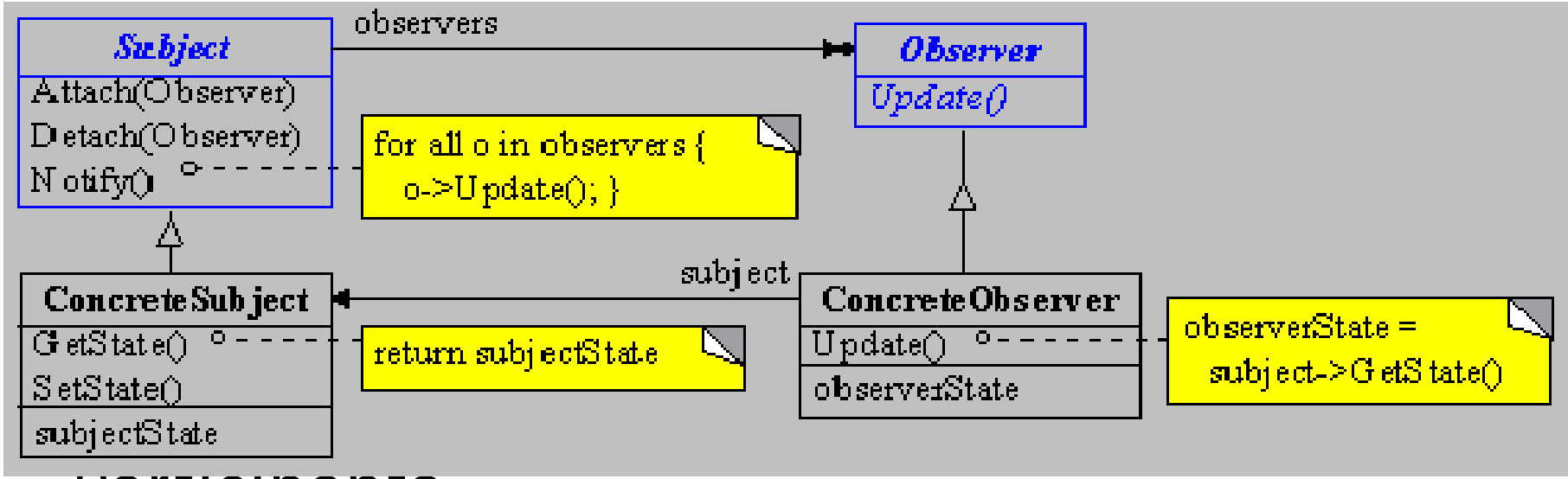
- Aliases ： Dependents, Publish-Subscribe
- Intent
  - Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.
- Motivation
  - 把系统分成一些相互关联的类或者对象，如何维护这些类的实例一致性？
  - The key objects in this pattern are subject and observer
    - One-to-many relationship
    - A subject may have any number of dependent observers.
    - All observers are notified whenever the subject undergoes a change in state.

Observer模式(二)

- Applicability：Use the Observer pattern when：
  - **When an abstraction has two aspects, one dependent on the other. Encapsulating these aspects in separate objects lets you vary and reuse them independently.**
  - **When a change to one object requires changing others, and you don't know how many objects need to be changed.**
  - **When an object should be able to notify other objects without making assumptions about who these objects are. In other words, you don't want these objects tightly coupled.**

Observer模式(三)

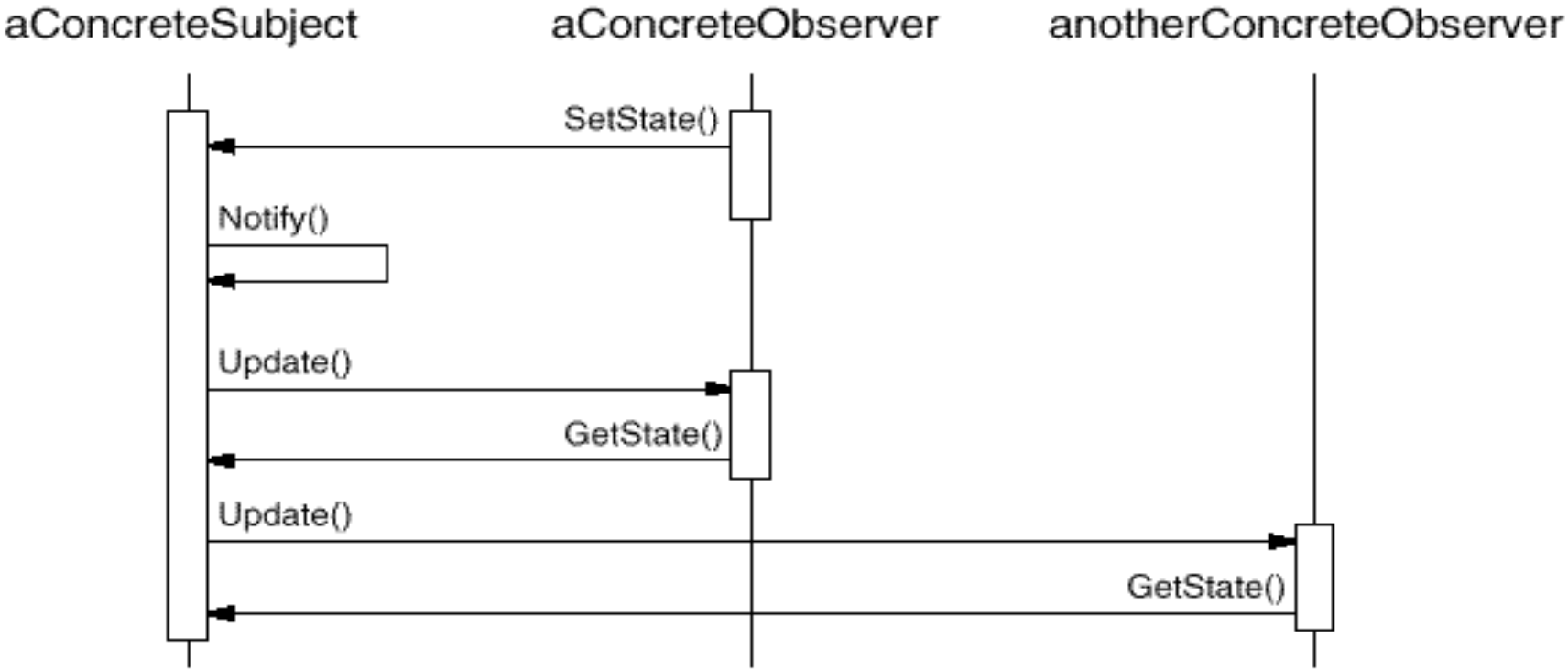
- Struct



- Participants
  - Subject、ConcreteSubject、Observer、ConcreteObserver

Observer模式(四)

- Collaborations

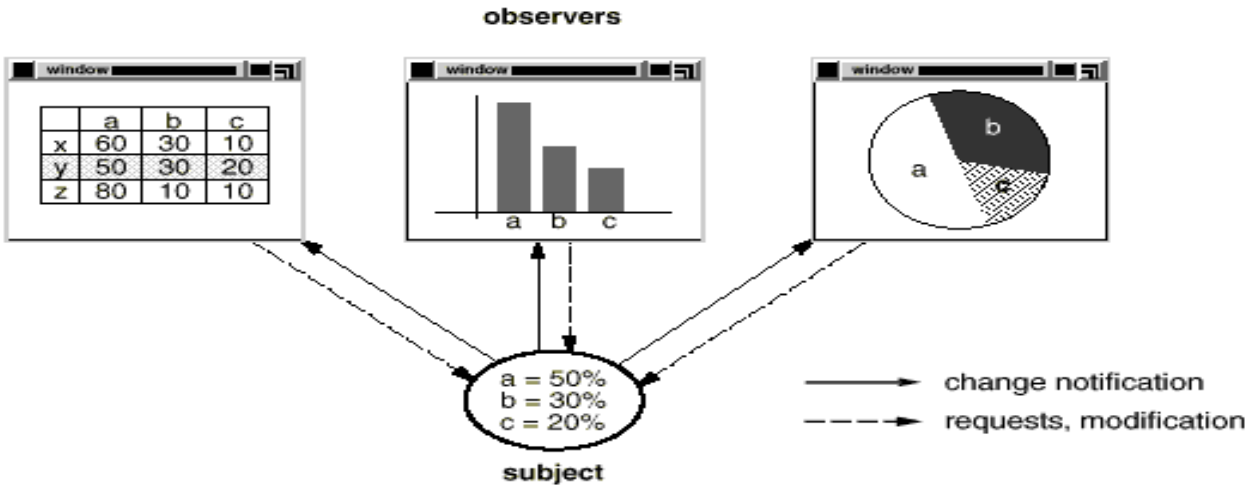


Observer模式(五)

- Evaluation
  - Abstract coupling between Subject and Observer
  - Support for broadcast communication
  - Unexpected updates
- Implementation
  - Mapping subjects to their observers.
  - Observing more than one subject
  - Who triggers the update? Client or subject?
  - Making sure Subject state is self-consistent before notification
  - subject向observer传递变化信息
  - 中间插入ChangeManager

Observer模式(六)

- Related Patterns
  - Mediator：用Mediator模式封装复杂的更新语义
- Examples
  - COM property page
  - COM+ Event Model,
  - MVC



模式 15：Strategy(一)

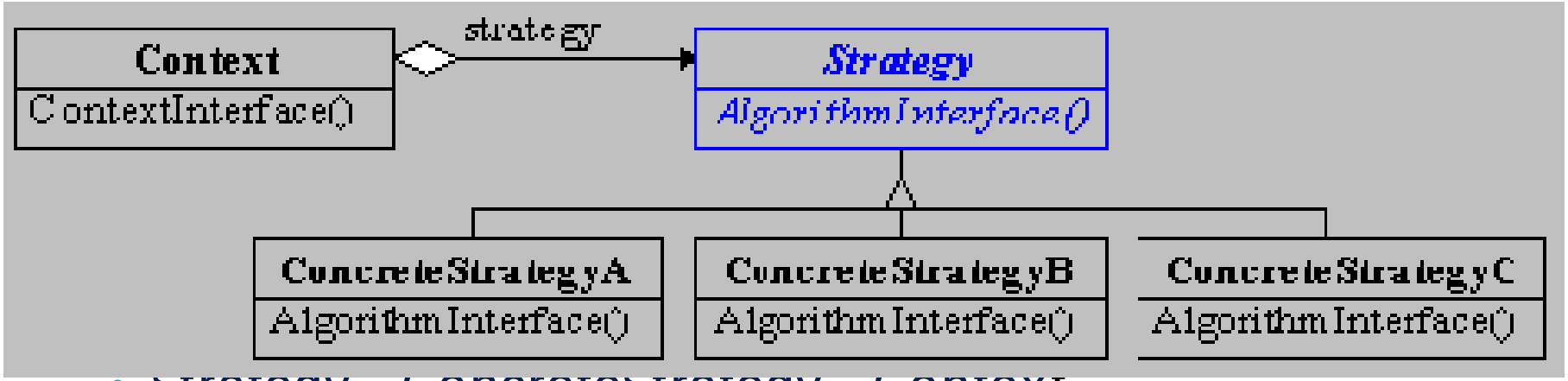
- Aliases ： Policy
- Intent
  - Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.
- Motivation
  - 有些算法对于某些类是必不可少的，但是不适合于硬编进类中。客户可能需要算法的多种不同实现，允许增加新的算法实现或者改变现有的算法实现
  - 我们可以把这样的算法封装到单独的类中，称为strategy

Strategy模式(二)

- Applicability：Use the Strategy pattern when：
  - many related classes differ only in their behavior.
  - you need different variants of an algorithm.
  - an algorithm uses data that clients shouldn't know about.
  - a class defines many behaviors, and these appear as multiple conditional statements in its operations.

Strategy模式(三)

- Struct



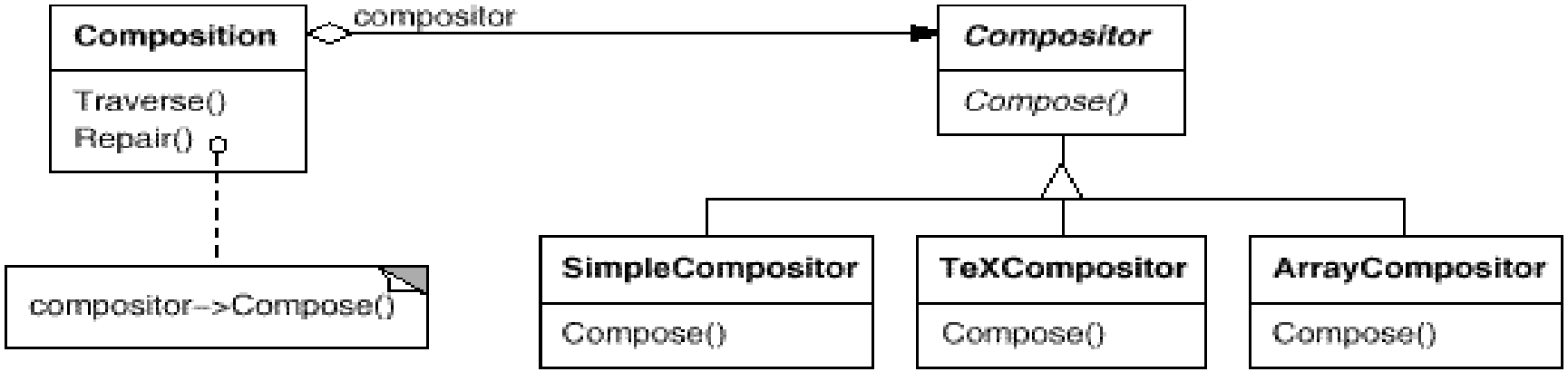
  - Strategy、ConcreteStrategy、Context
- Collaborations
  - Strategy and Context interact to implement the chosen algorithm
  - A context forwards requests from its clients to its strategy

Strategy模式(四)

- Evaluation
  - Families of related algorithms
  - An alternative to subclassing
  - Strategies eliminate conditional statements
  - Clients must be aware of different Strategies
  - Communication overhead between Strategy and Context
  - Increased number of objects
- Implementation
  - Defining the Strategy and Context interfaces
  - Strategies as template parameters
  - Making Strategy objects optional

Strategy模式(五)

- Related Patterns
  - flyweight：考虑用flyweight模式来实现strategy对象
- Examples

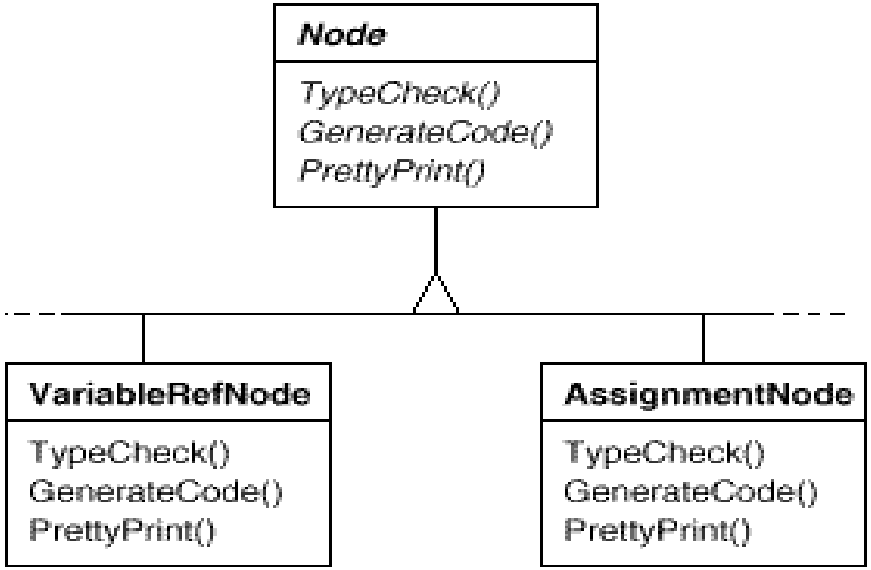


模式 16：Visitor(一)

- Intent
  - Represent an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates.
- Motivation
  - 为了把一个操作作用于一个对象结构中，一种做法是把这个操作分散到每一个节点上。导致系统难以理解、维护和修改
  - 把这样的操作包装到一个独立的对象(visitor)中。然后在遍历过程中把此对象传递给被访问的元素。

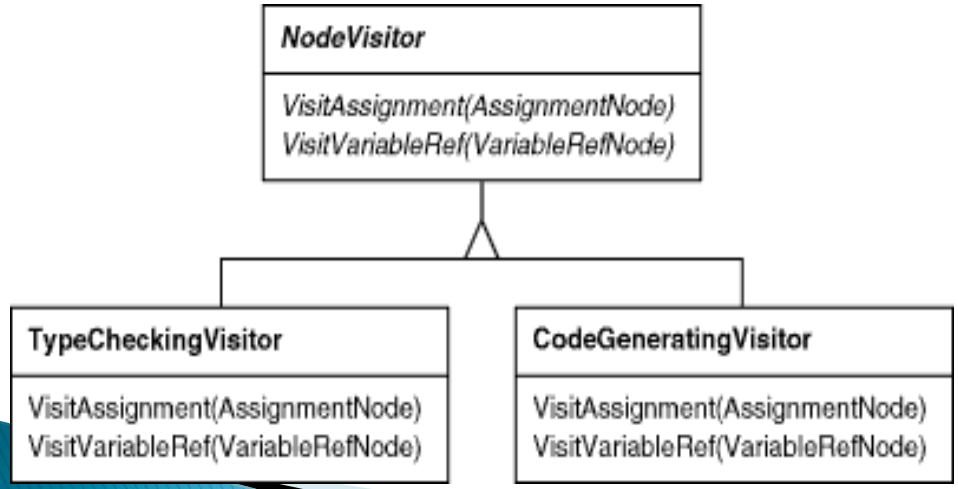
不用visitor的compiler例子

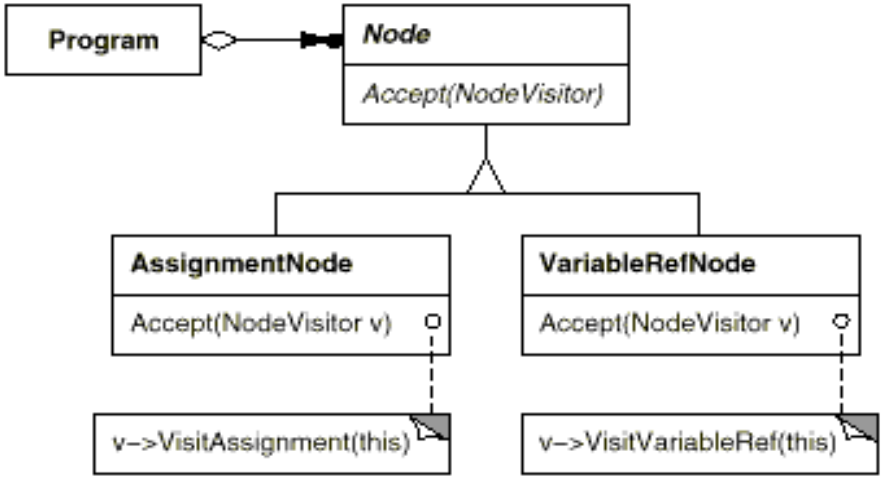
Operation / Class	TypeCheck	GenerateCode	PrettyPrint
VariableRefNode			
AssignmentNode			
...			



使用visitor的compiler例子

class / Class / operation	VariableRefNode	AssignmentNode
TypeCheckVisitor	VisitVariableRef	VisitAssignment
GenerateCodeVisitor	VisitVariableRef	VisitAssignment
PrettyPrintVisitor	VisitVariableRef	VisitAssignment



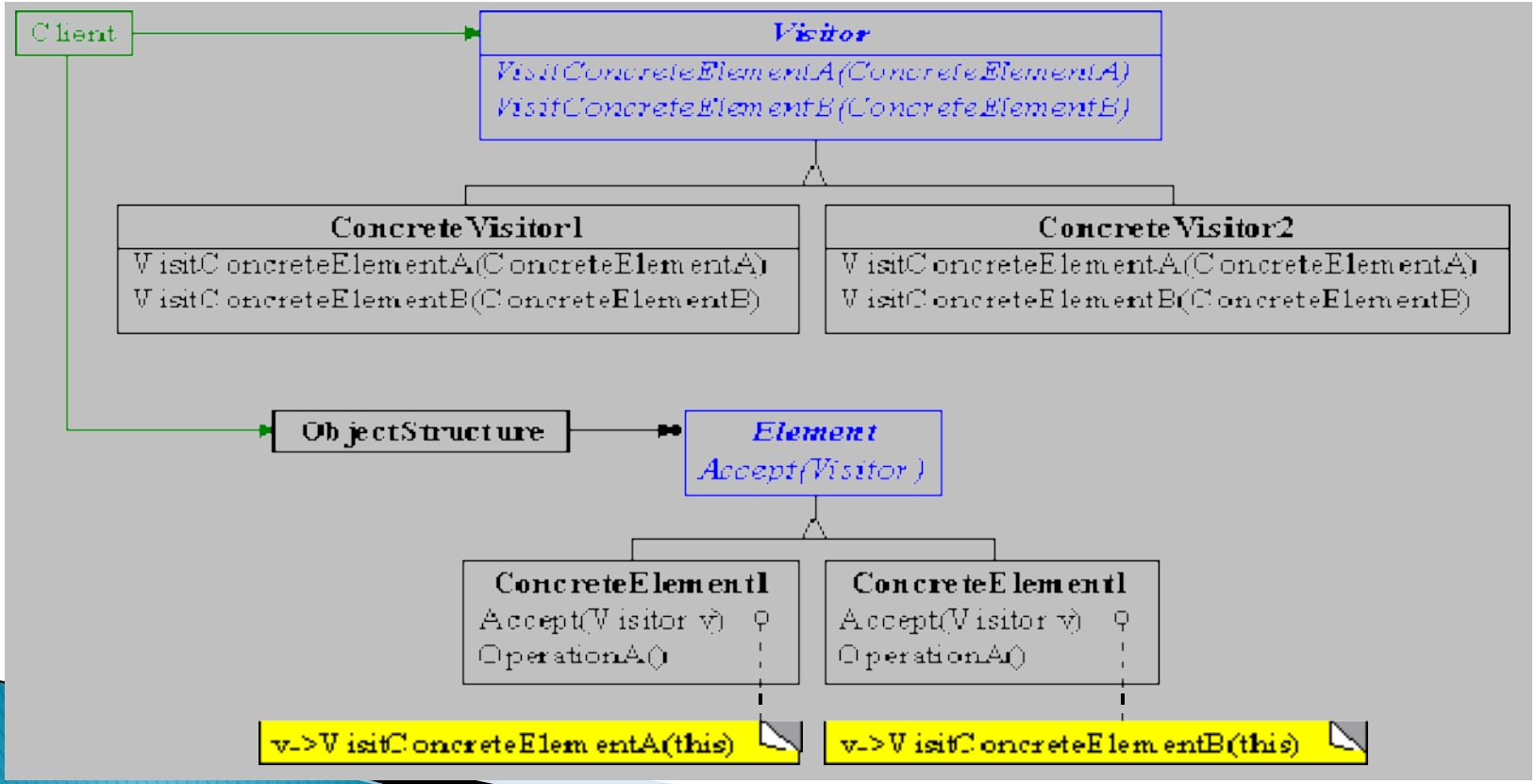


Visitor模式(二)

- Applicability：Use the Visitor pattern when
  - 一个对象结构包含许多对象类，我们想执行一些依赖于具体类的操作
  - 要对一个对象结构中的对象进行很多不同的并且不相关的操作，又不想改变这些对象类
  - 定义对象结构的类很少改变，但是经常要在此结构上定义新的操作。改变对象结构类，需要重定义所有visitor的接口

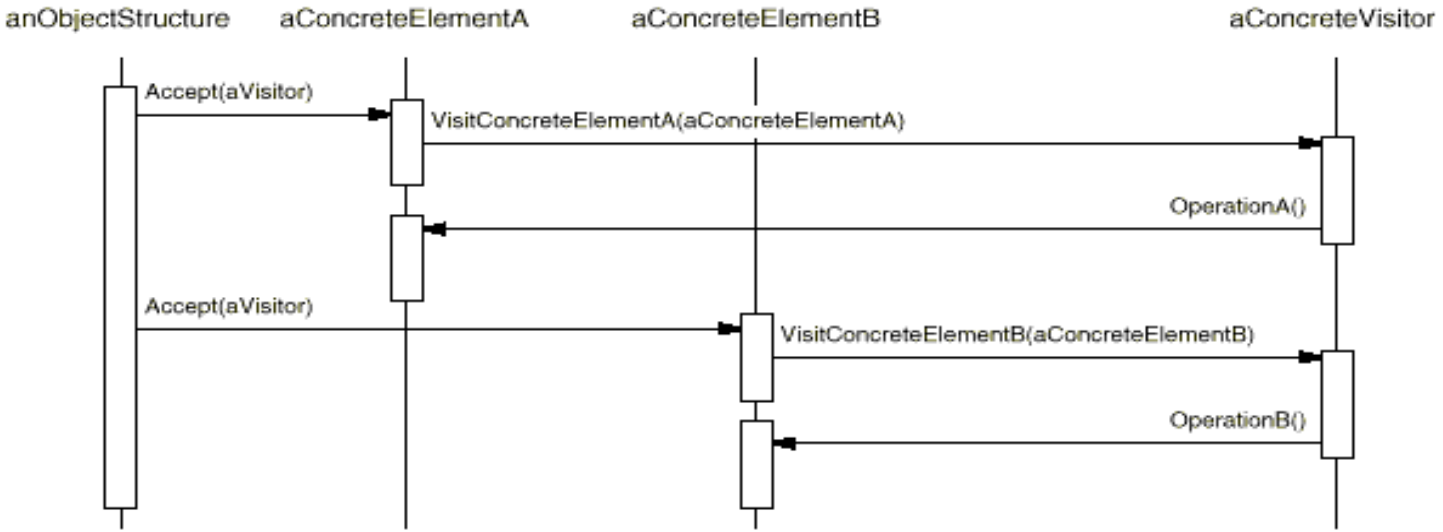
Visitor模式(三)

- Struct



Visitor模式(四)

- Participants
  - Client、Visitor、ConcreteVisitor、ObjectStructure、Element、ConcreteElement
- Collaborations
  - client先创建一个ConcreteVisitor，然后遍历ObjectStructure



Visitor模式(五)

- Evaluation
  - Visitor makes adding new operations easy
  - A visitor gathers related operations and separates unrelated ones
  - Adding new ConcreteElement classes is hard.
  - 即使不是类层次，visitor也可以实施
  - 状态累积
  - visitor要访问每个元素的状态，所以要打破封装
- Implementation
  - double-dispatch，Accept实现了double dispatch
    This is the key to the Visitor pattern: The operation that gets executed depends on both the type of Visitor and the type of Element it visits.
  - Who is responsible for traversing the object structure?

Visitor模式(六)

- Related Patterns
  - Composite：visitor常常被用于composite模式组成的结构中
- Examples
  - 编译器实现

其他Behavioral Patterns

- Chain of Responsibility
  - 请求的处理过程，沿着链传递，decouple发送和接收方
- Interpreter
  - 在类层次结构中，在特定环境的“interpret”过程
- Mediator
  - 用一个mediator来decouple各同等单元
- Memento
  - 在对象之外保存对象的内部状态
- State
  - 把一个对象的状态独立出来，动态可变换状态对象的类型
- Template Method
  - 在基类中定义算法的骨架，把某些细节延迟到子类中

Behavioral Patterns小结

- Strategy、Iterator、Mediator、State、Command
  - 用一个对象来封装某些特性，比如变化、交互、状态、行为、命令
- Mediator、Observer
  - Observer建立起subject和observer之间的松耦合连接
  - mediator把约束限制集中起来 -〉中心控制
- command、Chain of Responsibility、interpreter
  - command模式侧重于命令的总体管理
  - Chain of Responsibility侧重于命令被正确处理
  - interpreter用于复合结构中操作的执行过程