Design Patterns

Overview

什么是设计模式

- 设计模式源自建筑学

在某一背景下某个问题的一种解决方案

- Christopher Alexander

从建筑模式到软件设计模式

- 20世纪90年代初

  - 软件中是否存在不断重复出现，可以以某种相同方式解决问题
  - 是否可能用模式方法来设计，即先找出模式，然后根据这些模式创建特定的解决方案

GOF 设计模式

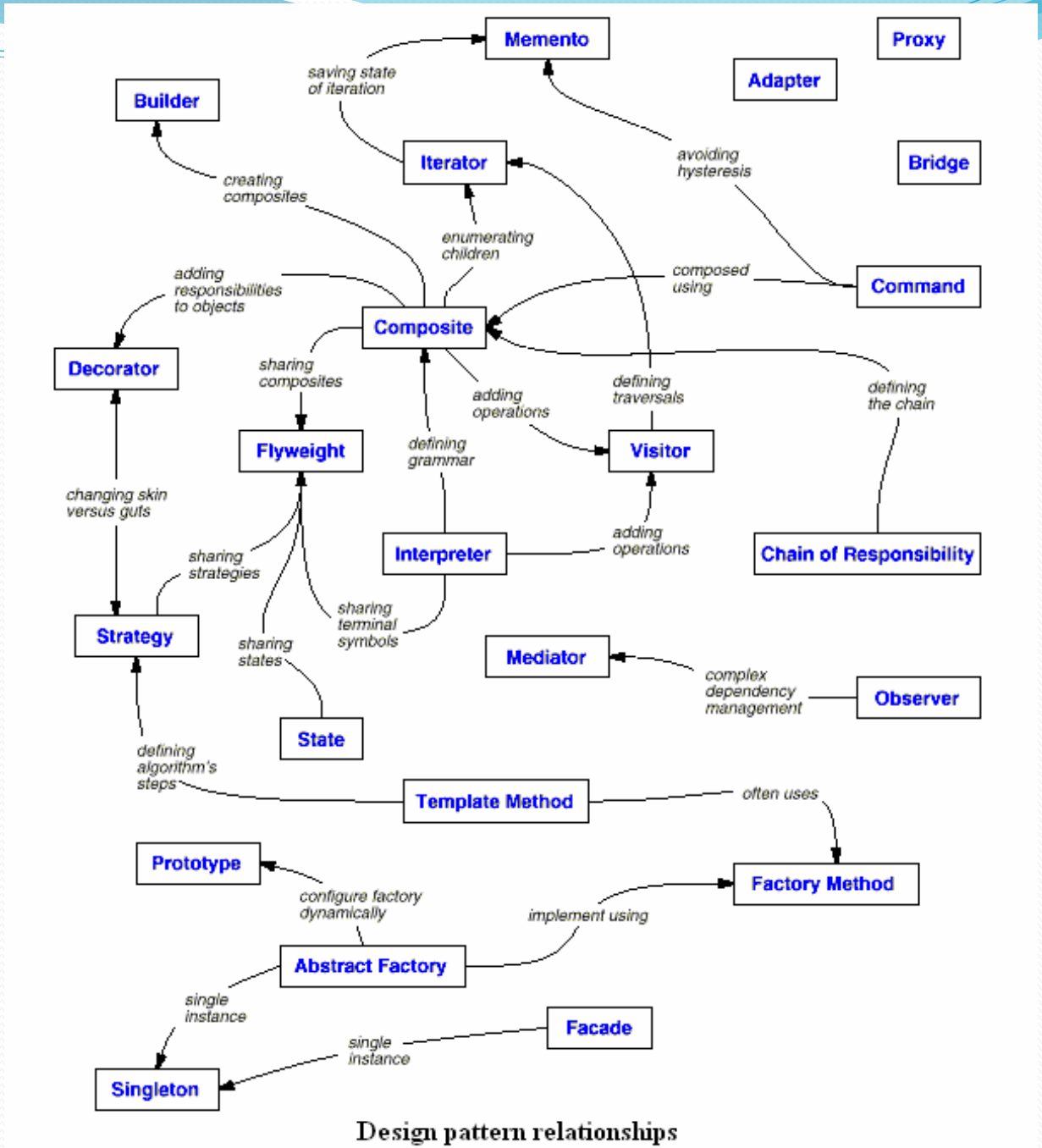
- Gamma、Helm、Johnson、Vlissides

  - 将设计模式的思想应用于软件设计
  - 编录了23个设计模式

设计模式分类

		Purpose		
		Creational	**Structural**	**Behavioral**
Scope	**Class**	Factory Method	Adapter (class)	Interpreter Template Method
	Object	Abstract Factory Builder Prototype Singleton	Adapter (object) Bridge Composite Decorator Flyweight Facade Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

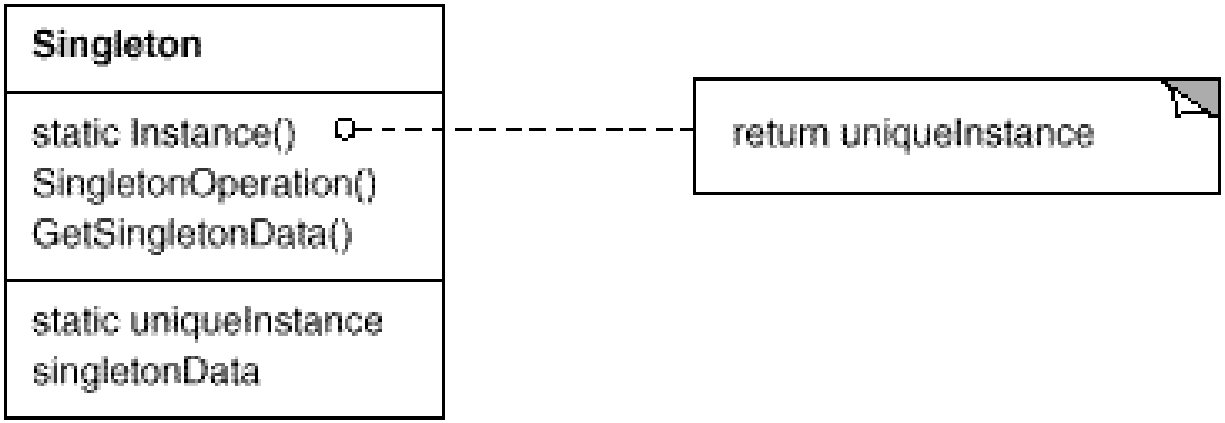
模式关系



Design pattern relationships

Singleton

- 意图:
  - 保证一个类仅有一个实例，并提供一个访问它的全局访问点。
- 动机：
  - 对于某些类只有一个实例很重要，如全局配置。
- 适用性：
  - 当类只能有一个实例而客户可以从一个众所周知的访问点访问它时
  - 当这个唯一实例应该通过子类化可扩展的，并且客户应该无需更改代码就能使用一个扩展实例


Singleton
static Instance()
SingletonOperation()
GetSingletonData()
static uniqueInstance
singletonData
return uniqueInstance

Singleton 实现

- C++ 实现

- C 实现(libvirt)

Composite

- 意图:

  将对象组合成树形结构以表示“部分整体”的层次结构

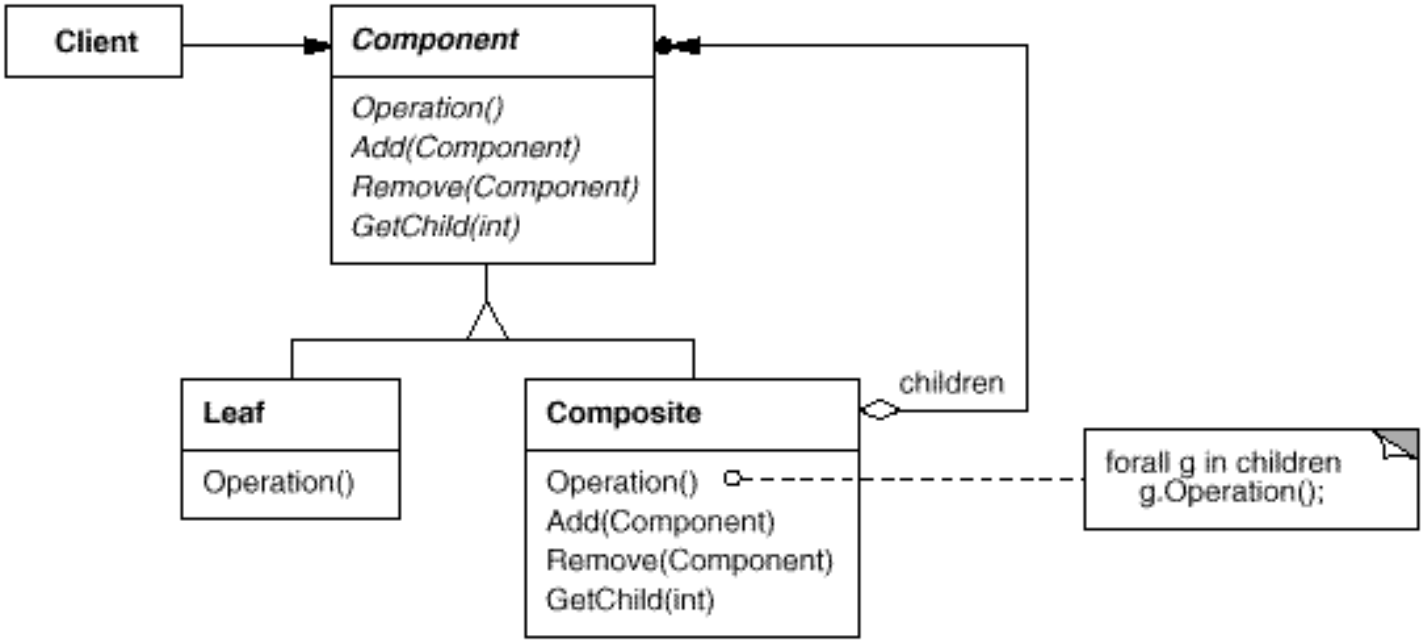
- 动机：

  使在大部分情况对客户来说对象是一样的。Composite描述了如何使用递归组合

- 适用性：

  你想表达部分-整体层次结构

  你希望用户忽略组合对象与单个对象的不同，用户将统一地使用组合结构中的所有对象


Client
Component
Operation()
Add(Component)
Remove(Component)
GetChild(int)
Leaf
Operation()
Composite
Operation()
Add(Component)
Remove(Component)
GetChild(int)
children
forall g in children
g.Operation();

Composite 实现

- C++实现 - XML
- C 实现 - Linux Kernel Device Object Model

Observer

- 意图:

  定义对象间一种一对多的依赖关系，当一个对象的状态发生变化时，所有依赖于它的对象都得到通知并被自动通知。

- 动机：

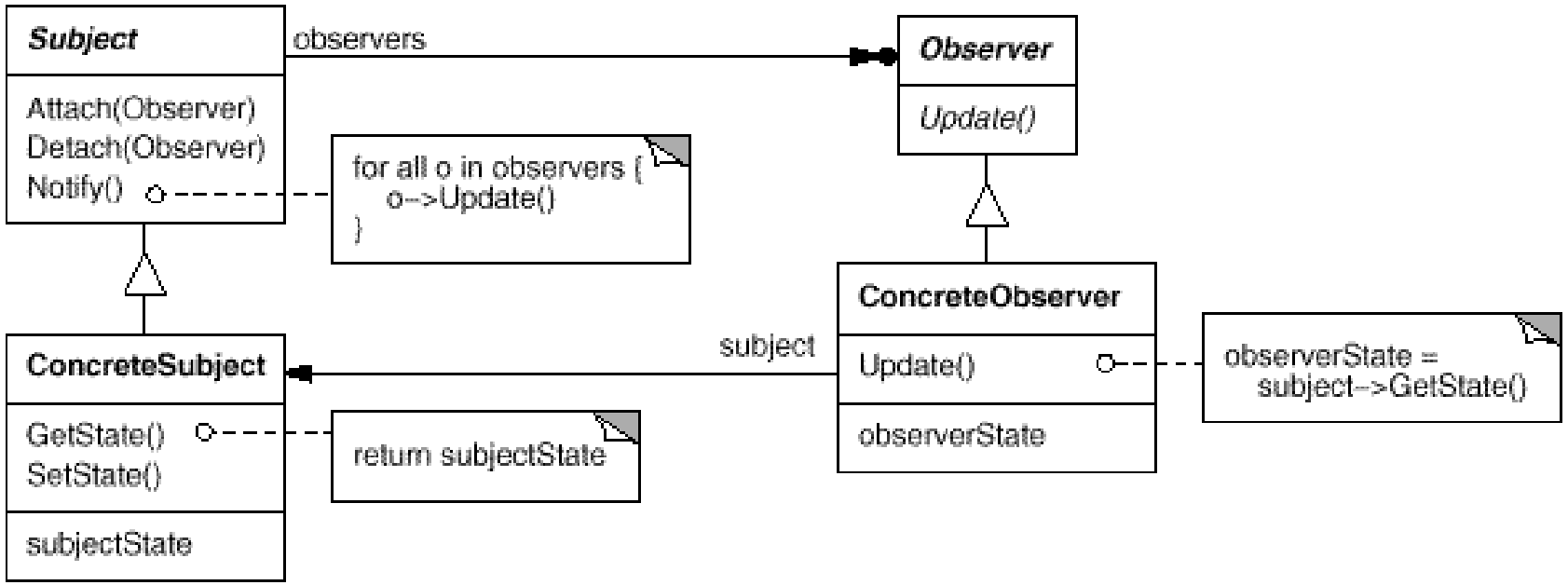
  讲观察者与目标之间解耦

- 适用性：

  当一个抽象模型有两个方面，其中一个方面依赖于另一个方面。将这二者封装在独立的对象中以使他们可以各自独立地改变和复用。

  当一个对象的改变需要同时改变其他对象，而不知道具体有多少对象有待改变。

  当一个对象必须通知其他对象。而它又不能假定其他对象是谁。换言之，你不希望这些对象是紧密耦合的。


Subject
observers
Observer
Attach(Observer)
Detach(Observer)
Notify()
Update()
for all o in observers {
o->Update()
}
ConcreteObserver
subject
ConcreteSubject
Update()
observerState =
subject->GetState()
GetState()
SetState()
return subjectState
observerState
subjectState

Observer 实现

- C 实现 (libvirt)

Thank you!