Lecture10 设计模式

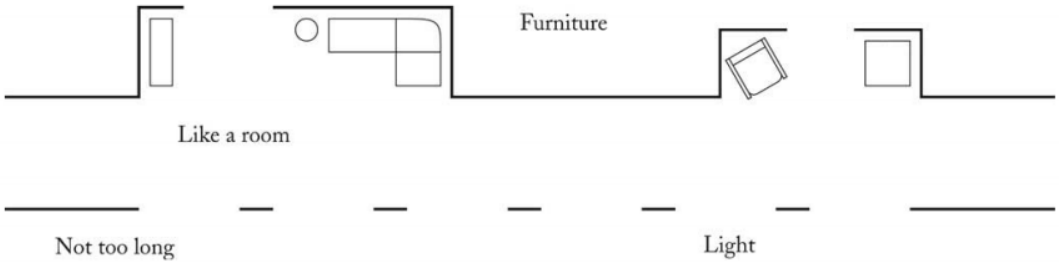
1. 模式 Pattern

模式的介绍

每一个模式都有一个

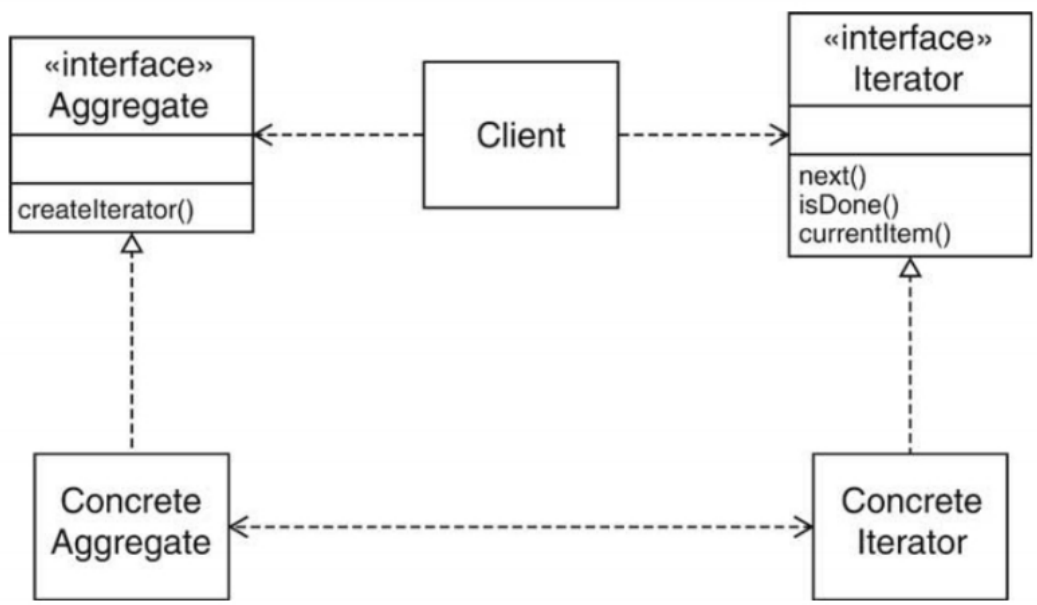
- name：简短的名称
- context：对内容的简要描述
- problem：对问题的冗长的描述
- solution：解决方案的处理方案

建筑模式示例 Short Passage Pattern 短走廊模式

- context：漫长、枯燥的走廊注定了现代建筑的一切弊端
- problem
  - 一幅令人沮丧的图片
  - 光纤和家具的问题
  - 关于医院中病人焦虑
  - 研究表明超过 50 英尺的走廊被认为是不舒服的
- solution
  - 保持走廊尽可能短，把它弄得尽可能像房间一样，铺上地毯或木地板，书架。美丽的窗户，要使它们的形状宽敞，并且总是给它们充足的光线：最好的走廊和走廊是那些沿着整面墙都有窗户的
  - 

2. 迭代器模式 Iterator Pattern

介绍



- context
  - 一个包含基本**元素对象**的**聚合对象**
  - 客户端需要访问**元素对象**
  - **聚合对象**不应该公开其内部结构
  - 多个客户端可能需要独立访问
- solution
  - 定义一个迭代器，每次获取一个元素对象
  - 每个迭代器对象都跟踪下一个元素的位置
  - 如果有多个聚合/迭代器变体，最好是聚合类和迭代器类实现公共接口类型

具体实现

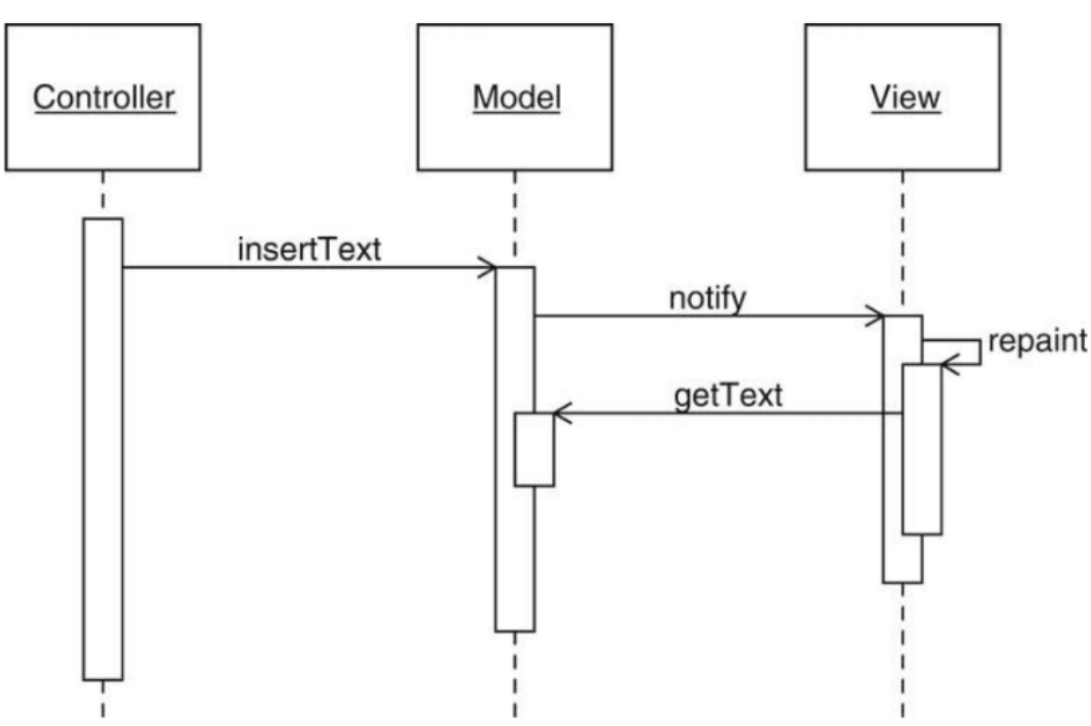
- 模式中的名称只是一种示例
- 在不同的实现中，命名是不同的

以 Java 中的实现为例

设计模式用的名称	实际的名称（Java 的实现）
ConcreteAggregate	LinkedList
Aggregate	List
Iterator	ListIterator
ConcreteIterator	实现 ListIterator 的匿名类
createIterator()	listIterator()
isDone()	!hasNext()
currentItem()	next() 的返回值

3. MVC 模式

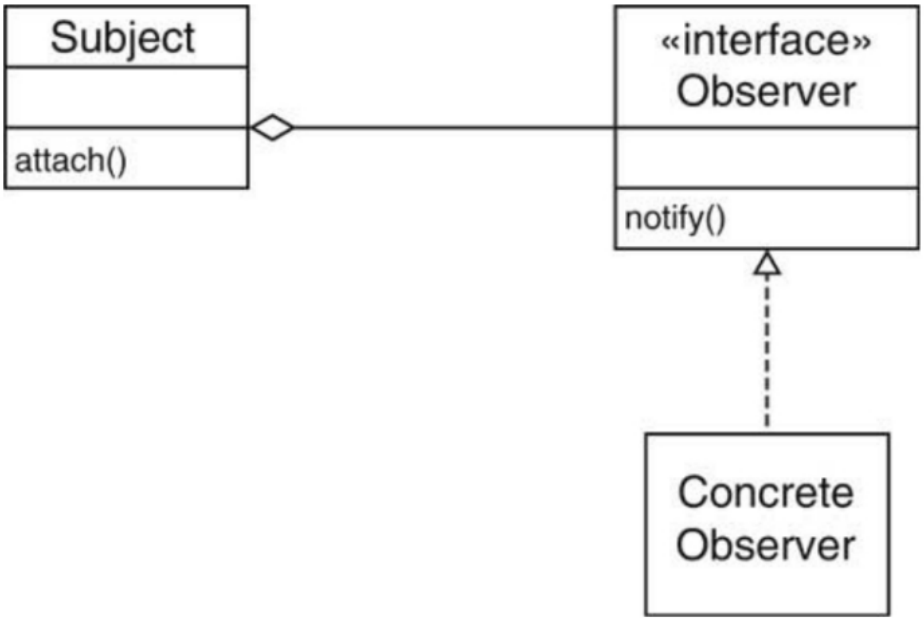
介绍



- context
  - 名词解释
    - Model：数据结构，不是可视化的表达
    - View：可视化的表达
    - Controller：用户的交互
  - View、Controller 更新 Model
  - Model 告诉 View 数据进行了更新
  - View 重新绘图

4. 观察者模式 Observer Pattern

介绍



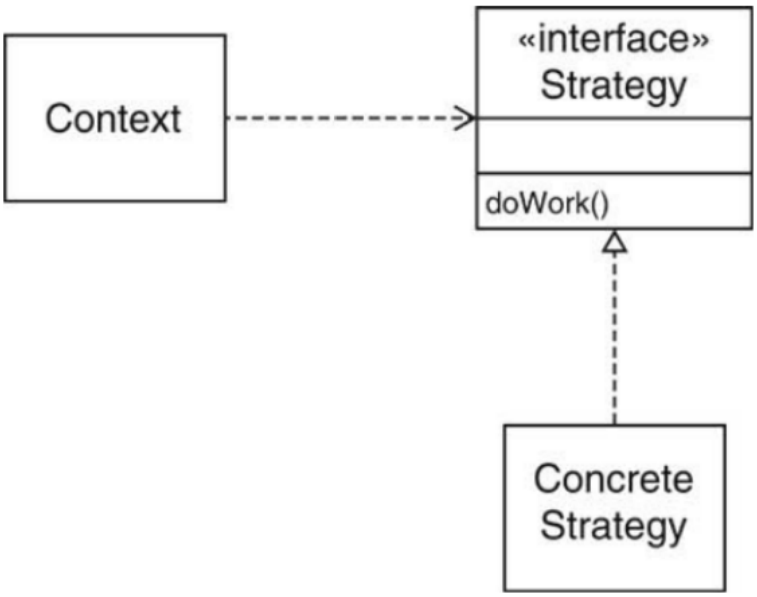
- context
  - 概括：Observer 将自己附加到 Subject 上，以便得到通知
    - 当一些有趣的事情发生时，model 通知 view
    - 当一些有趣的事情发生时，按钮通知动作监听器
    - view 把自己附加到 model 上，以便被通知
    - 动作侦听器将自己附加到按钮上，以便得到通知
  - 一个对象，叫做 Subject，是事件发生的源头
  - 当某个事件发生时，一个或者多个 Observer 想要得到通知
- solution
  - 定义 Observer 接口类型，所有具体的 Observer 类都实现它
  - Subject 维护一个 Observers 的集合
  - Subject 提供挂在 Observer 和取消挂在 Observer 的丰富
  - 当某些事件发生时， Subject 通知所有被挂在的 Observer

具体实现

设计模式用的名称	实际的名称（Java Swing Button）
Subject	Button
Observer	ActionListener
ConcreteObserver	实现 ActionListener 接口的类
attach()	addActionListener()
notify	actionPerformed()

5. 策略模式 Strategy Pattern

介绍



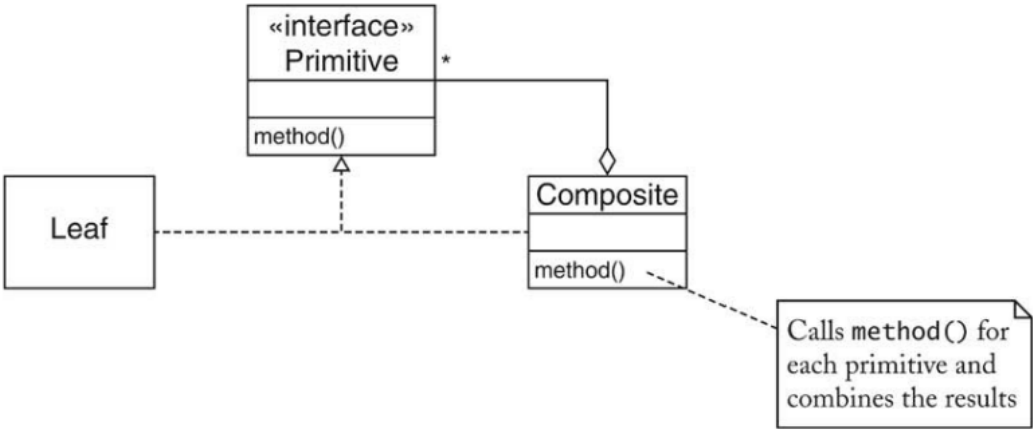
- context
  - 类可以从算法的不同变体中受益
  - 客户有时想用自定义版本替换标准算法
- solution
  - 定义一个接口类型，它是算法的抽象
  - 实际的策略类实现了这个接口类型
  - 客户可以提供策略对象
  - 每当需要执行算法时，调用策略对象的适当方法

具体实现

设计模式用的名称	实际的名称（Java Sorting）
Context	Collections
Strategy	Comparator
ConcreteStrategy	实现 Comparator 接口的类
doWork()	compare

6. 组合模式 Composite Pattern (装饰者模式)

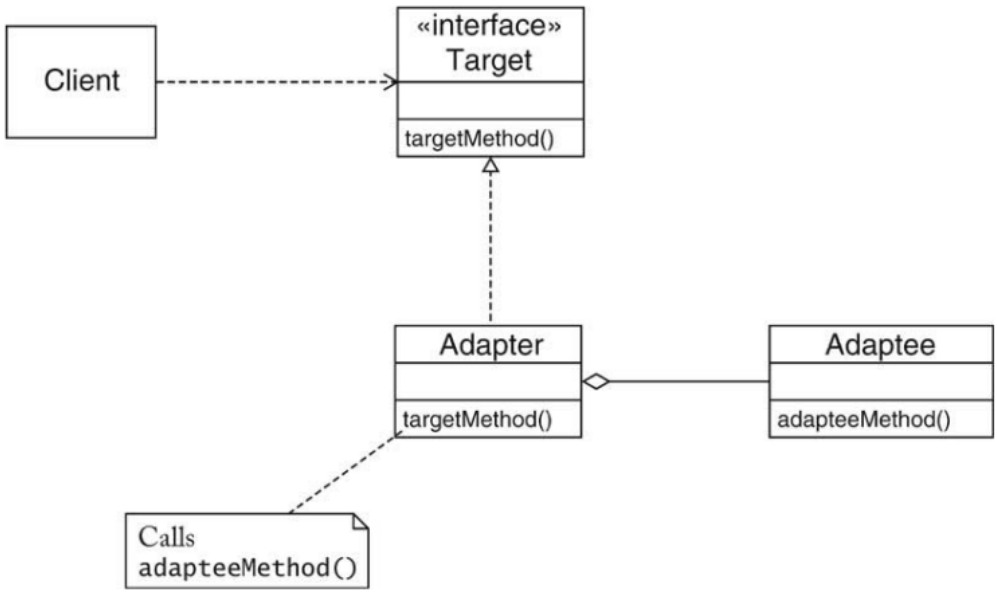
介绍



- context
  - 基本对象可以组合为复合对象
  - 客户端将复合对象视为基本对象
- solution
  - 定义一个接口类型，它是基本对象的抽象
  - 复合对象为基本对象的集合
  - 复合类和基元类实现相同的接口类型
  - 当实现来自接口类型的方法时，复合类将该方法应用于其基本对象并组合结果

6. 适配器模式 Adapter Pattern

介绍



- context
  - 希望使用一个现有的类（作为被适配者 adaptee）而不修改它
  - target 接口和 adaptee 接口在概念上是相关的
- solution

- 定义一个 adapter 类实现了 target 接口
- adapter 类保存了对 adaptee 的引用，它将 target 方法转化为 adaptee 方法
- 客户端将 adaptee 封装在 adapter 类对象中

具体实现

设计模式用的名称	实际的名称（Java Stream -> Reader）
Adaptee	InputStream
Target	Reader
Adapter	InputStreamReader
Client	希望从 InputStream 中读取文本的类
targetMethod()	read(读一个字符)
adapteeMethod	read(读一个字节)

示例

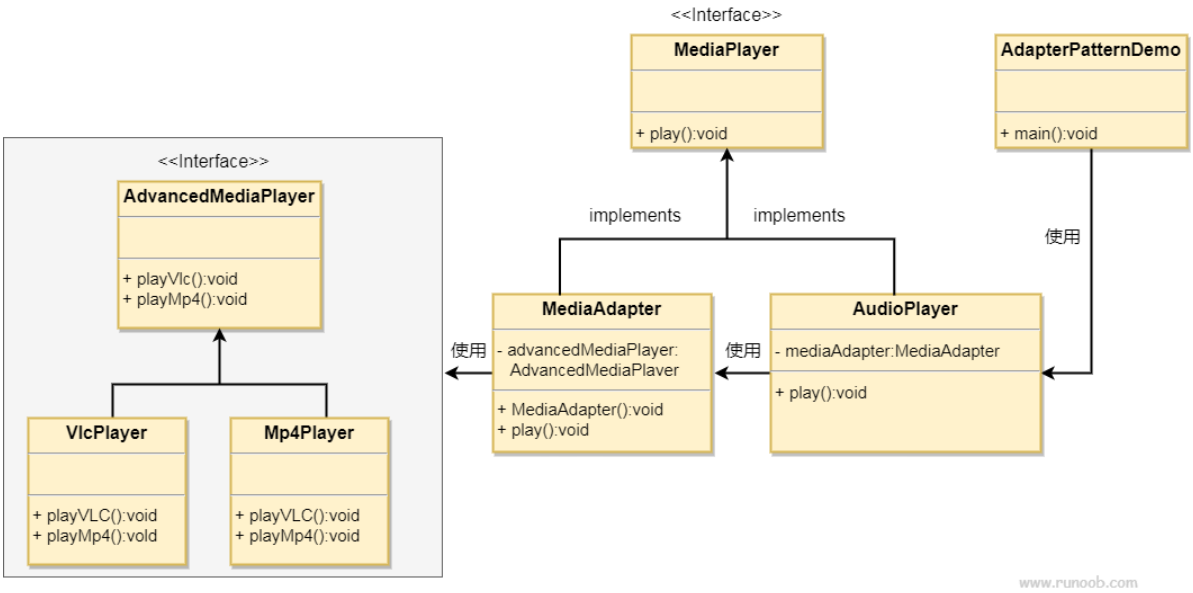
> https://www.runoob.com/design-pattern/adapter-pattern.html

我们有一个 *MediaPlayer* 接口和一个实现了 *MediaPlayer* 接口的实体类 *AudioPlayer*。默认情况下，*AudioPlayer* 可以播放 mp3 格式的音频文件。

我们还有另一个接口 *AdvancedMediaPlayer* 和实现了 *AdvancedMediaPlayer* 接口的实体类。该类可以播放 vlc 和 mp4 格式的文件。

我们想要让 *AudioPlayer* 播放其他格式的音频文件。为了实现这个功能，我们需要创建一个实现了 *MediaPlayer* 接口的适配器类 *MediaAdapter*，并使用 *AdvancedMediaPlayer* 对象来播放所需的格式。

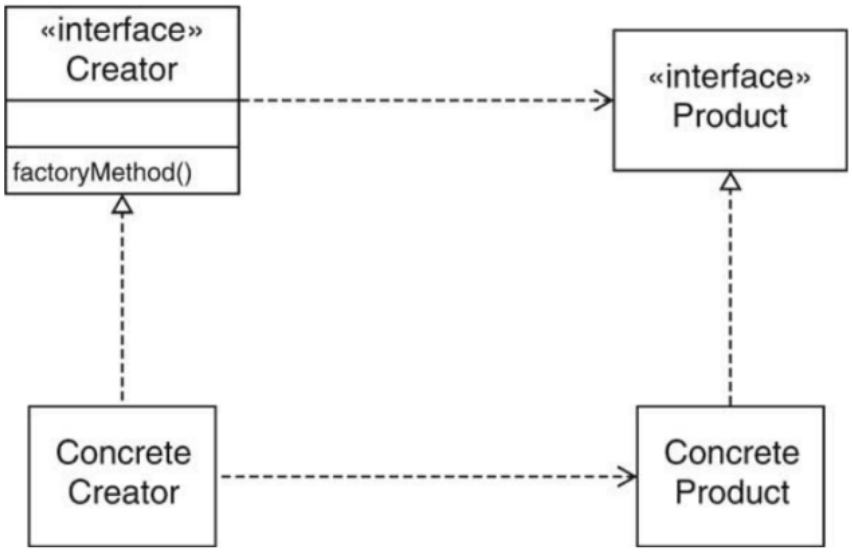
AudioPlayer 使用适配器类 *MediaAdapter* 传递所需的音频类型，不需要知道能播放所需格式音频的实际类。*AdapterPatternDemo* 类使用 *AudioPlayer* 类来播放各种格式。



设计模式用的名称	实际的名称（Java Stream -> Reader）
Adaptee	VlcPlayer, Mp4Player（实现了AdvancedMediaPlayer）
Target	MediaPlayer
Adapter	MediaAdapter
Client	AdapterPatternDemo
targetMethod()	play(播放mp3、vlc、mp4格式)
adapteeMethod	play(播放vlc格式)、play(播放mp4格式)

7. 工厂模式 Factory Pattern

介绍



- context
  - 一个类型（Creator）创造其它的类型（Product）
  - Creator 的各种各样的实现了（工厂）需要创造各种各样不同的实现了 Product 的对象（商品）
  - 客户端不需要知道 Product 对象的具体类型
- solution
  - 定义 Creator 接口表达所有工厂实现类的共性
  - 定义 Product 接口表达所有商品实现了的共性
  - 在 Creator 内定义一个方法，叫**工厂方法 Factory Method**，该方法产生一个 Product 对象
  - 每一个具体的工厂实现类实现了工厂方法，使得它产生出具体的实现了 Product 类的对象

具体实现

设计模式用的名称	实际的名称（Java Collection 迭代器）
Creator	Collection
ConcreteCreator	Collection 的具体实现类
factoryMethod()	iterator()
Product	Iterator
ConcreteProduct	Iterator 的具体实现类（通常的匿名的）

8. 访问者模式 Visitor Pattern

介绍

- problem
  - 不同的 element 的类型所执行的操作可能是不一样的，不能仅依赖于多态
  - 多态假设在超类中定义了一组**固定数量的方法 fixed set of method**
- trick
  - 如果**子类的数量是固定的 set of classes is fixed**，可以使用**可变的方法集合 variable set of method**

提供分离的 visitor 方法

```
public interface Visitor{
    void visitElementType1(ElementType1 element);
    void visitElementType2(ElementType2 element);
    void visitElementType3(ElementType3 element);
}
```

Visitor

- Visitor 是一个接口类型
- 为每个新操作提供单独的类
- 一般有一个 accept 方法

```
1   public void accept(Visitor v){v.visit(this);}
```

- 必须实现 visit 方法

示例