

# 第十五章 MVC架构

任课教师:武永亮

wuyongliang@edu2act.org

# **■上节回顾**

## ₩课程内容

- ■MVC概述
- ■MVC实现细节
- ■MVC之设计模式
- ■表现层演化MVP

## ₩课程内容

- ■MVC概述
- ■MVC实现细节
- ■MVC之设计模式
- ■表现层演化MVP

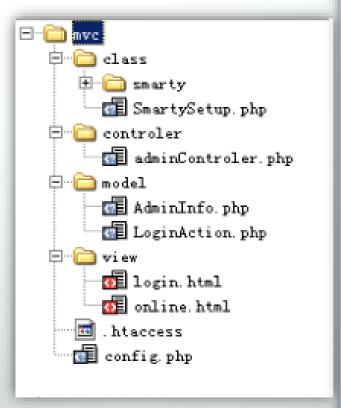
#### ■MVC的起源

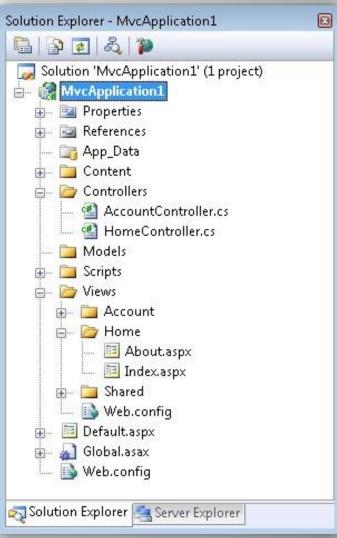
■MVC架构最早是smalltalk语言研究团提出的 ,应用于用户交互应用程序中。

#### ■MVC的设计思想

■把一个应用的输入、处理、输出流程按照 Model、View、Controller的方式进行分离。

## ■我们知道的MVC.....

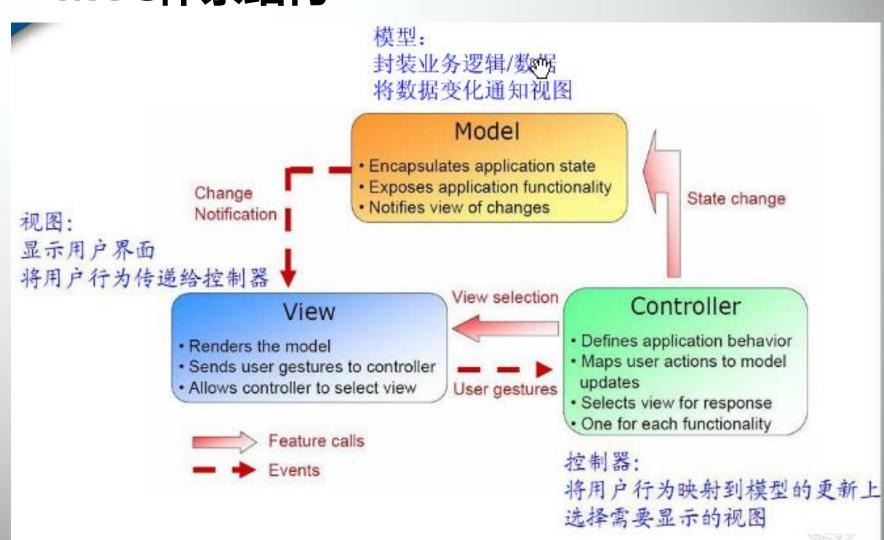




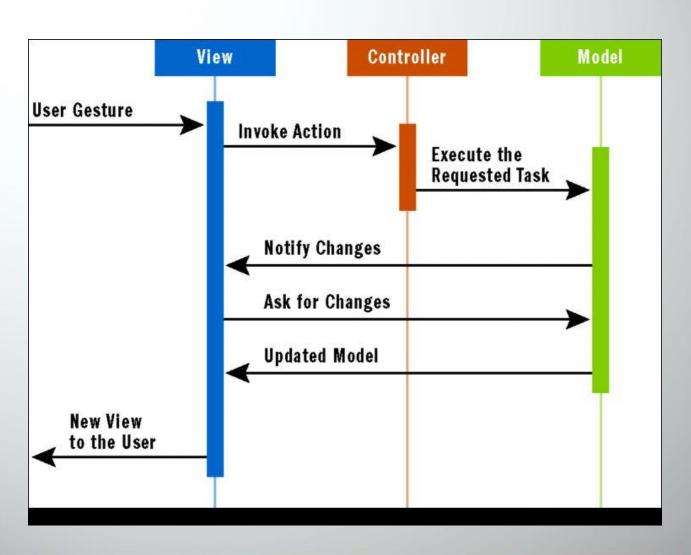
## ₩课程内容

- ■MVC概述
- ■MVC实现细节
- ■MVC之设计模式
- ■表现层演化MVP

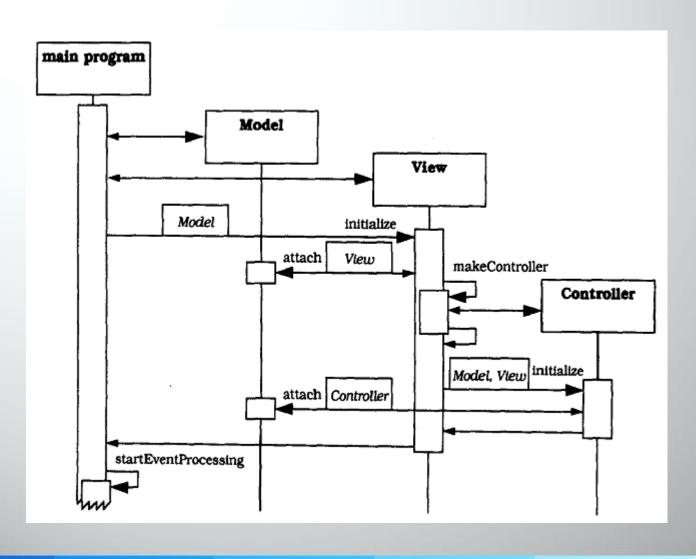
#### ■MVC体系结构



## ■MVC的动态行为



## ■MVC结构如何初始化



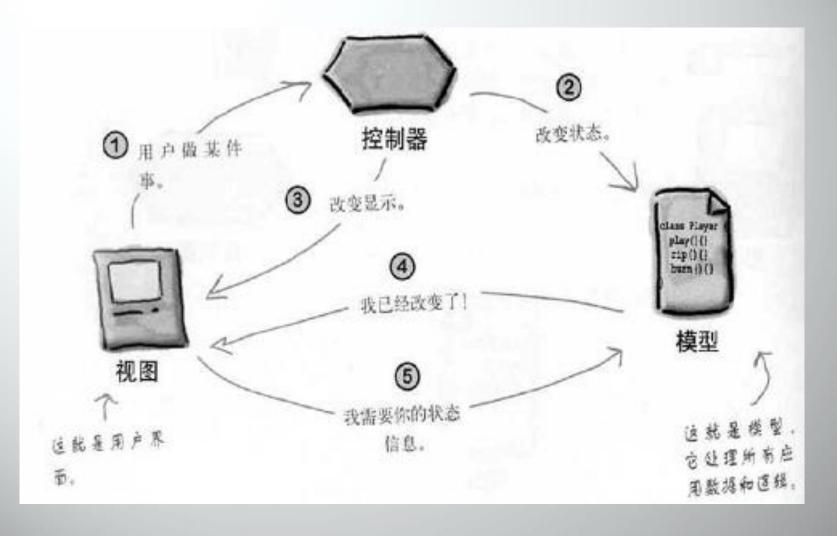
## ₩课程内容

- ■MVC概述
- ■MVC实现细节
- ■MVC之设计模式
- ■表现层演化MVP

## ■ MVC——复合模式之王

- ■观察者模式
- ■策略模式
- ■组合模式
- •••••

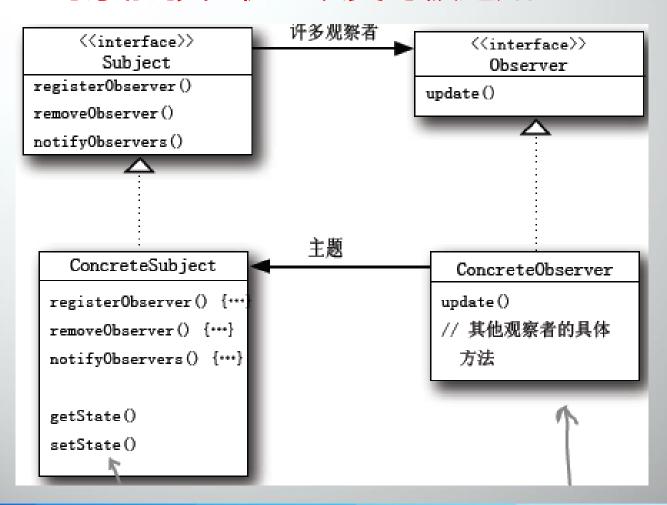
## ■MVC观察者运作细节



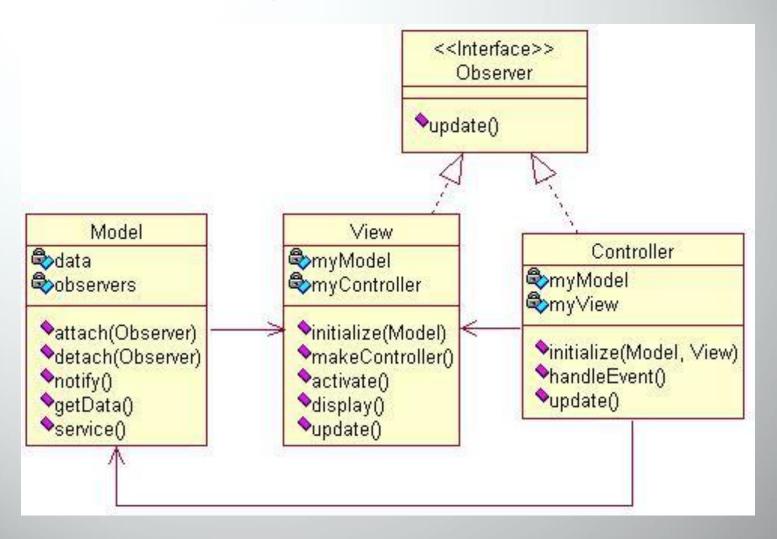
**MVC观察者运作细节** 被观察者 变状态。 控制器 3 改变显示。 play()() rip()() burn()() 我已经改变了! 模型 视图 (5) 我需要你的状态 信息。 它处理所有应 用数据和逻辑。

## ■观察者模式回顾

■让对象能够在状态改变时被通知



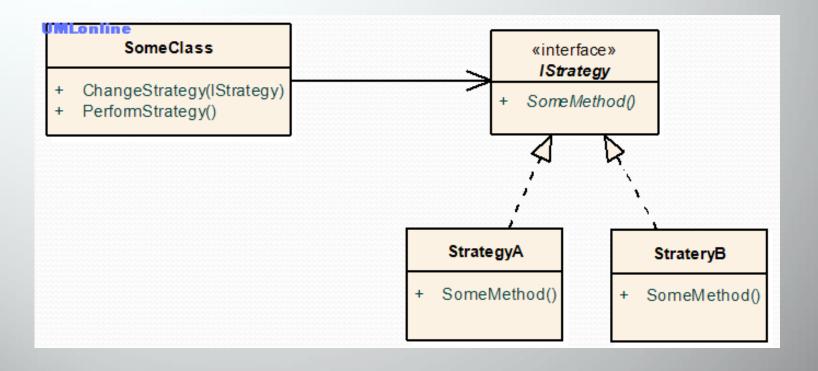
#### ■MVC的设计类图



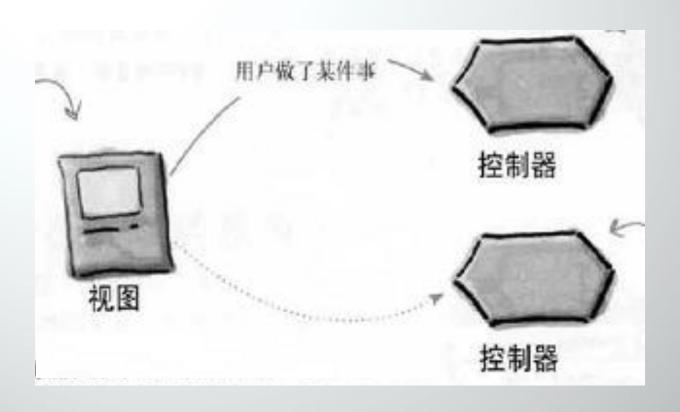
■MVC策略运作细节 变状态。 控制器 3 改变显示。 play()() rip()() burn()() 我已经改变了 模型 视图 (5) 我需要你的状态 信息。 用数据和逻辑。

#### ■策略模式

■策略模式定义了一系列的算法,并将每一个算法封装起来,而且使它们还可以相互替换。策略模式让算法独立于使用它的客户而独立变化



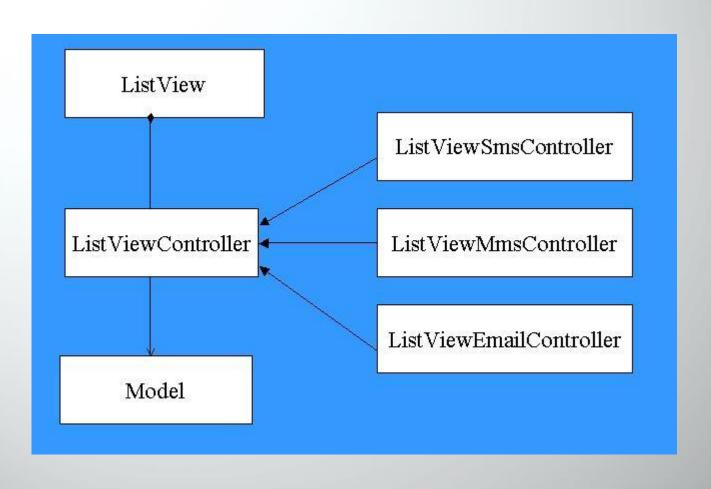
## ■策略模式在MVC中的应用



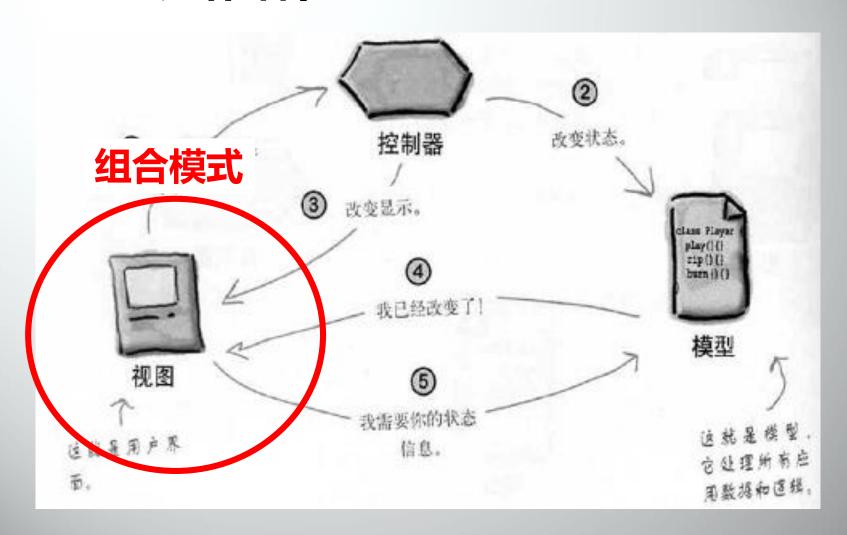
## ■策略模式在MVC中的应用



## ■策略模式在MVC中的应用

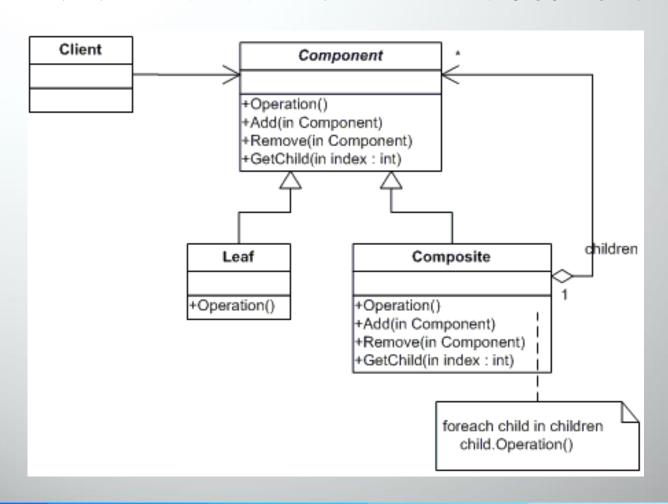


## ■MVC运作细节

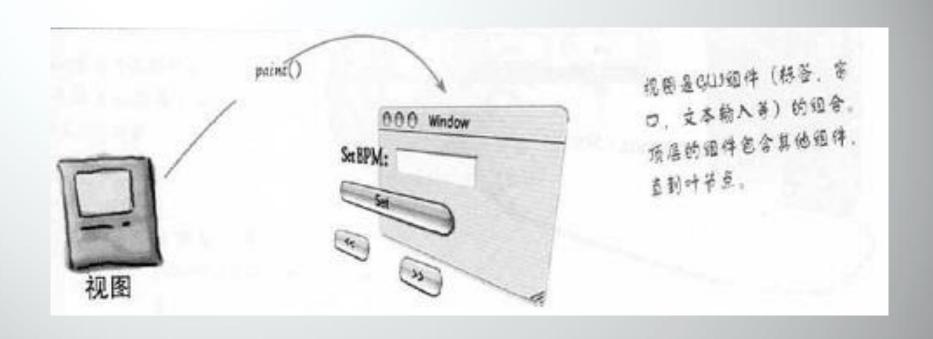


## ■组合模式回顾

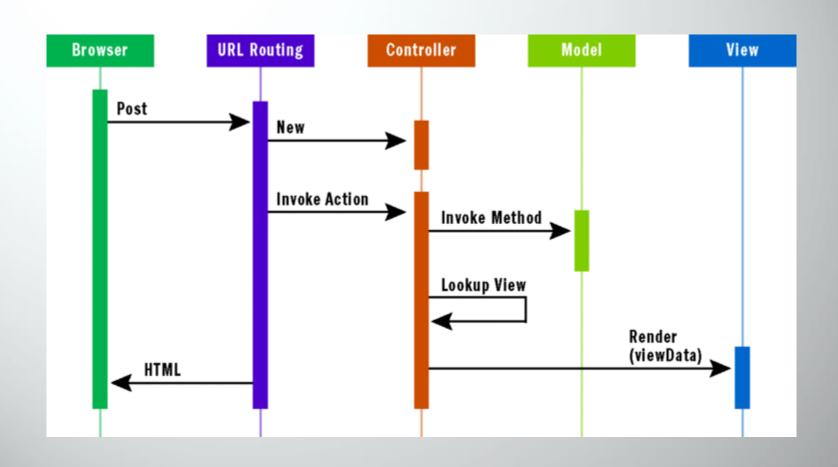
■客户用一致的方式处理对象集合和单个对象。



## ■组合模式在MVC中的应用



## **■ MVC模式变体——Model2**



#### **■ MVC不足**

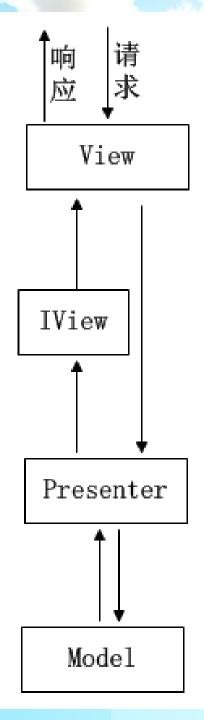
- ■View是可以直接访问Model的
- View是依赖于Model的
- ■有一些业务逻辑在View里实现

## ₩课程内容

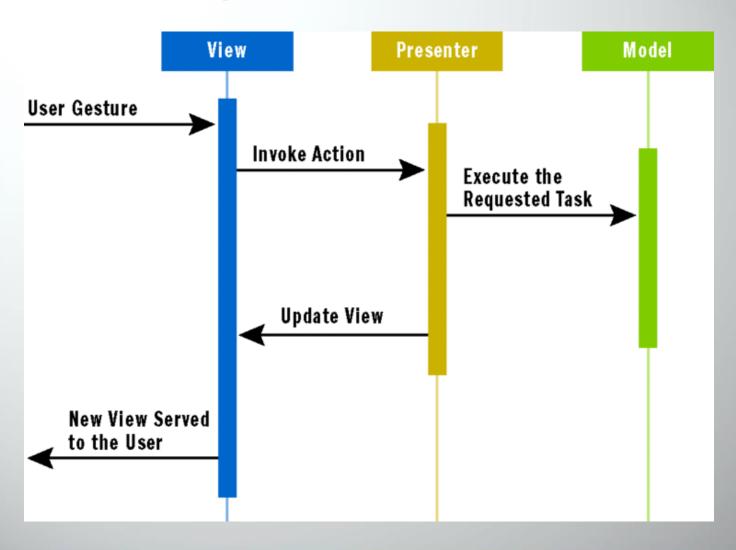
- ■MVC概述
- ■MVC实现细节
- ■MVC之设计模式
- ■表现层演化MVP

#### ■表现层的演化——MVP

- ■模型-视图-展示器
- Model View Presenter
  - ■模型与视图完全分离
  - ■可以更高效地使用模型,因为所有的交互都发生在一个地方——Presenter内部
  - ■我们可以将一个Presener用于多个 视图,而不需要改变Presenter的逻辑。这个特性非常的有用,因为视 图的变化总是比模型的变化频繁
  - ■如果我们把逻辑放在Presenter中,那么我们就可以<mark>脱离用户接口来测试</mark>这些逻辑(单元测试)

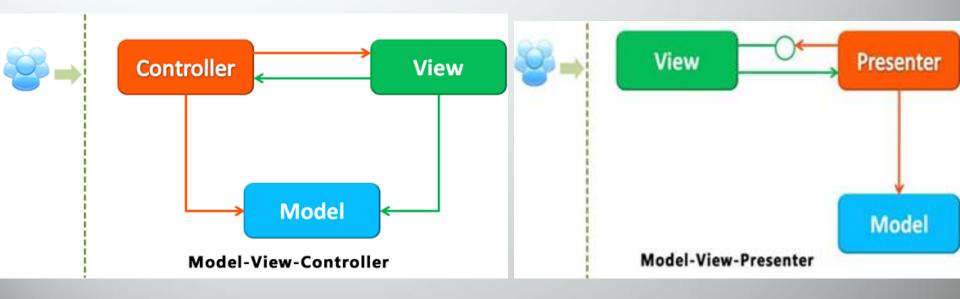


## ■MVP交互过程



## ■MVP与MVC对比

- ■视图并不了解模型
- ■展示器将忽略视图中使用的具体UI技术
- ■视图可以被模拟,以便测试



#### class Use Case Model 之例实例 CustomerDto CustomerService customers: Collection<CustomerDto> «property» + CompanyName(): string + CustomerService() + ContactName(): string LoadAllCustomer(): Collection<CustomerDto> Country(): string + LookupCustomer(string): CustomerDto CustomerID(): string Form Form1 presenter: CustomerDetailsPresenter «interface» **ICustomerDetailsView** AddCustomer(CustomerDto): void + AddCustomer(CustomerDto): void Form1() «property» Form1\_Load(object, EventArgs): void + CompanyName(): string IstCustomers\_SelectedIndexChanged(object, EventArgs): void ContactName(): string «property» Country(): string CompanyName(): string CustomerID(): string -view + ContactName(): string SelectedItemID(): string + Country(): string + CustomerID(): string SelectedItemID(): string +presenter System Web. UI. Page Default CustomerDetailsPresenter presenter: CustomerDetailsPresenter view: ICustomerDetailsView {readOnly} + AddCustomer(CustomerDto): void ClearView(): void OnInit(EventArgs): void -presenter CustomerDetailsPresenter(ICustomerDetailsView) # Page\_Load(object, EventArgs): void Initialize(): void «property» LoadAllCustomer(): Collection<CustomerDto> CompanyName(): string + ShowCustomerDetails(): void ContactName(): string UpdateView(CustomerDto): void Country(): string «property» CustomerID(): string

SelectedItemID(): string

SelectedCustomerID(): string

## ■小结

# Thank You, 谢谢!