假如现在要开发一个电子商务的网站

有订单管理、角色管理、用户管理3大基本功能

我们建立三个控制器

Ctrl\_order

Ctrl\_role

Ctrl\_user

3个控制器下有若干的action实现具体功能

需求不停的添加，于是三个控制器的代码也越来越多，于是拆分

Ctrl\_order1

Ctrl\_order2

…..

控制器会越来越多，于是在控制层再添加一个叫做订单模块的文件夹，将订单的控制器放于此文件夹中，便 于管理

这样就变成了

Order

Order1

Order2

这个其实就是组件化的一个趋形

在面向客户的开发过程中就会出现 订制化的概念

如：客户A想在添加订单过程中，再加一个<说明>字段，客户B想在订单结算的时候加一个<提醒>字段

传统的做法就是在数据库中添加两个字段，然后用各种IF进行判断

这样就要修改控制器的各种代码，使方法变的不可维护 臃肿，同时数据库也变的很大

解决方法：

面向组件模式进行开发

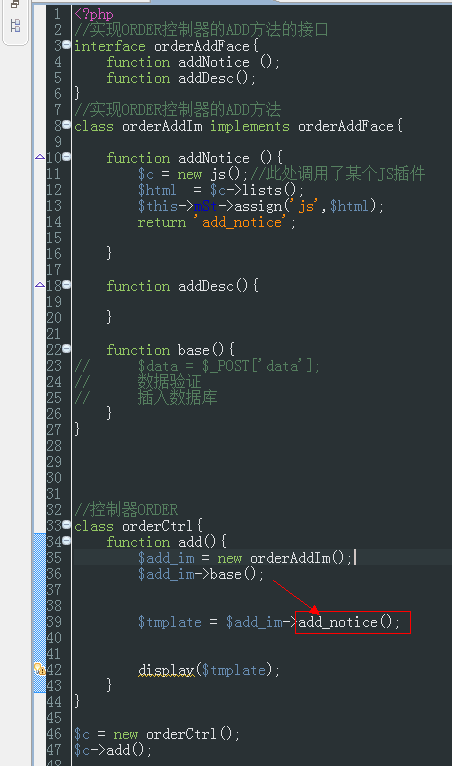
现在，order1把之前添加订单功能，拆分成两个的ACTION，就变成了

Function add\_desc() //增加提醒字段的方法

Fun ction add\_Desc() //增加说明字段的方法

现在的问题就是：如何根据不同用户，而调用不同的实现方法，如果用IF做判断，这就又跟上面没有太大的区别了~

于是，引入接口概念



(先留下一个问题：如何能确定，order1 是该调用add\_desc还是add\_notice)

分析一下上面这段代码：

有几个关键的：orderCtrl类，orderAddIm类，orderAddFace接口。

分析orderCtrl:

$add\_im = new orderAddIm();//实例化具体的实现类

$add\_im->base();//此方法的基本方法，

如：

$data = $\_POST['data'];

数据验证

插入数据库

$tmplate = $add\_im->add\_notice();//调用实现方法

具体调用哪个模版文件display($tmplate);

orderAddIm：//这就是实现添加订单的扩展方法，实际中我们只需要编写这部分就行了

$c = new js();//此处调用了某个JS插件

$html = $c->lists();

$this->mSt->assign('js',$html);//这部分也是个重点，我在后面介绍

return 'add\_notice';//返回调用的模板文件名

orderAddFace：这个文件，主要是用来生成权限表的

JS类，插件中的base方法中有定义

Assign(‘color’,xxx),这是定义了颜色为xxx

而在实现的过程中，list\_blue,覆盖了base方法中的color,改成了blue

