

## MVC 设计模式、自定义框架

导入：

### 一、过程式PHP项目的缺点：

- 1、 代码中，HTML 和 PHP 代码混合，不利于项目的分工和维护，页面风格的切换很难。
- 2、 代码中存在大量冗余代码，代码复用较少。例如，分页功能、页面跳转、数据库表的增删改查操作。
- 3、 编码周期长。工作量大。
- 4、 权限验证工作量大（每个页面都要做）。
- 5、 数据库操作安全性欠佳（SQL 注入），效率低。

### 二、面向对象编程的优点：

从现实的角度解决问题。例如：要实现用户的登录，首先定义用户类，用户类有登录的操作，然后通过用户类型的对象的登录方法解决登录问题。

通过封装能够隐藏实现细节，通过继承能够实现代码的重用。

**新的问题：**如何对代码进行管理？定义的类有没有类别上的划分？如何实现逻辑与显示的分离？

## 一、MVC 设计思想

## 一个简单的体现 MVC 思想的实例：

1、index.php 使用面向过程的方式显示的联系人表的所有记录。

```
1 <?php
2     /*
3         面向过程的方式，先访问数据库，获取数据。
4         再显示到网页上。
5         网页代码整体上大致呈现逻辑和显示的分离。|
6     */
7     $conn = mysql_connect("localhost", 'root', 'root');
8     mysql_select_db('jytxl', $conn);
9     $resultset = mysql_query("select * from member");
10
11     //将结果集资源转换成二维数组，方便下面遍历
12     while ($row = mysql_fetch_assoc($resultset)) {
13         $arr[] = $row;
14     }
15 ?>
16 <!DOCTYPE html>
17 <html lang="en">
18 <head>
19     <meta charset="UTF-8">
20     <title>简易通讯录</title>
21 </head>
22 <body>
23     <table border="1" width="400">
24         <?php foreach ($arr as $one): ?>
25
26             <tr>
27                 <td><?=$one['name']?></td>
28                 <td><?=$one['telephone']?></td>
29                 <td><a href="lookinfo.php?id=<?=$one['id']?>">查看详情</a></td>
30             </tr>
31         <?php endforeach ?>
32     </table>
33 </body>
34 </html>
```

2、将 index.php 的代码进行拆分，拆分成 3 个文件：

memList\_m.php: 与数据库交互获取数据结果的逻辑放置在这里；

memList\_v.html: 显示结果的代码放置在这里；

memList\_c.php: 控制整个流程的功能调度，将 m 和 v 整合在一

起的代码放在这里。

```
index.php x memList_c.php x memList_m.php x memList_v.html x
1 <?php
2
3 /**
4  * [memberList 获取所有联系人]
5  * @return [array] [二维数组保存所有联系人数据]
6  */
7 function memberList(){
8     $conn = mysql_connect("localhost",'root','root');
9     mysql_select_db('jytxl',$conn);
10    $resultset = mysql_query("select * from member");
11
12    //将结果集资源转换成二维数组，方便下面遍历
13    while ($row = mysql_fetch_assoc($resultset)) {
14        $arr[] = $row;
15    }
16    mysql_free_result($resultset);
17    mysql_close();
18    return $arr;
19 }
20
21 ?>
```

```
index.php x memList_c.php x memList_m.php x memList_v.html x
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>简易通讯录</title>
6 </head>
7 <body>
8     <table border="1" width="400">
9         <?php foreach ($arr as $one): ?>
10
11             <tr>
12                 <td><?=$one['name']?></td>
13                 <td><?=$one['telephone']?></td>
14                 <td><a href="lookinfo.php?id=<?=$one['id']?>">查看详情</a></td>
15             </tr>
16         <?php endforeach ?>
17     </table>
18 </body>
19 </html>
```

```
index.php x memList_c.php x memList_m.php x memList_v.html x
1 <?php
2 //先调用M模型文件，处理数据库查询，获得结果
3 require "memList_m.php";
4 $arr = memberList(); //调用方法获取数据
5
6 //再调用V视图文件，显示结果
7 require "memList_v.html";
8
9 ?>
```

在浏览器访问时，访问的是 memList\_c.php。

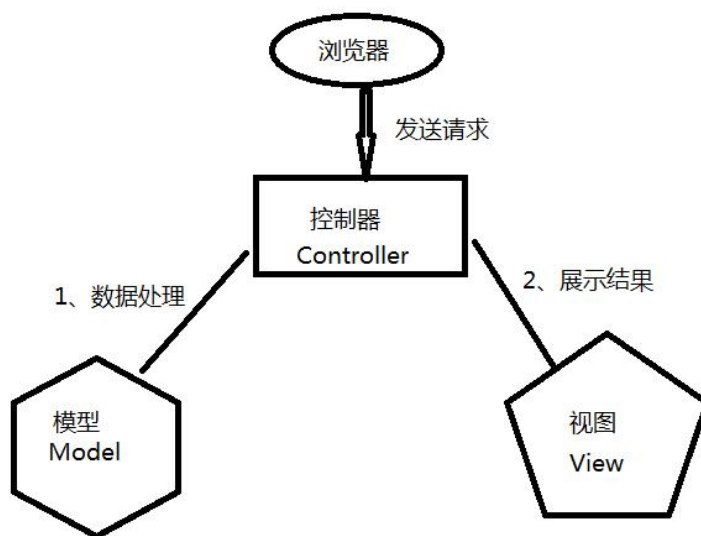
运行结果：



The screenshot shows a web browser window with the address bar displaying 'localhost:81/mvcdemo2/step1/memList\_c.php'. The page content is a table with three columns: Name, ID, and a link to '查看详情' (View Details).

张三	12345678	<a href="#">查看详情</a>
李四	888888	<a href="#">查看详情</a>
安贝贝	333	<a href="#">查看详情</a>
伯伯	777	<a href="#">查看详情</a>
丁丁	555	<a href="#">查看详情</a>
高兴	3456	<a href="#">查看详情</a>
焦傲	888888	<a href="#">查看详情</a>
康健	9754	<a href="#">查看详情</a>
马晓林	6578	<a href="#">查看详情</a>
侯俐	5555	<a href="#">查看详情</a>
花千骨	5555	<a href="#">查看详情</a>

上面编码的分割方式体现了 MVC 设计模式的思想。它的优势：逻辑与显示彻底分离。



总结：

**MVC 设计模式** 是一种使用 Model +View +Controller ( 模型-视图-控制器 ) 设计创建 Web 应用程序的模式：

- Model ( 模型 ) ——应用程序核心 ( 负责数据的业务逻辑处理 )。
- View ( 视图 ) ——显示数据 ( 负责结果的展示 )。
- Controller ( 控制器 ) ——控制调度 ( 负责接收用户的请求，并进行整体功能的控制和调度 )。

MVC 分层有助于管理复杂的应用程序，因为您可以在一个时间内专门关注一个方面。例如，您可以在不依赖业务逻辑的情况下专注于视图设计。同时也让应用程序的测试更加容易。MVC 分层同时也简化了分组开发。不同的开发人员可同时开发视图、控制器逻辑和业务逻辑。

注意事项：

1、浏览器请求的[永远是控制器](#)，所以后面编写 MVC 程序时定义的超链接 URL 地址或者页面跳转的目标地址指向的只能是 Controller



控制器（不能是模型或视图）。

2、Model 模型和 View 视图都是由控制器调用的。M 需要的用户提供的数据由 C 提供，得到的结果返回给 C；V 需要显示的数据是由 C 提供的。

3、Model 模型和 View 视图之间不需要交互。

## 二、自定义 MVC 框架

**MVC 框架：**ThinkPHP、Yii、Lavarel 等等。

**框架：**框架就是通过提供一个开发 Web 程序的基本架构，PHP 开发框架把 PHP Web 程序开发摆到了流水线上。换句话说，PHP 开发框架有助于促进快速软件开发，这节约了你的时间，有助于创建更为稳定的程序，并减少开发者的重复编写代码的劳动。这些框架还通过确保正确的数据库操作以及只在表现层编程的方式帮助初学者创建稳定的程序。PHP 开发框架使得你可以花更多的时间去创造真正的 Web 程序，而不是编写重复性的代码。

**简单的说：**框架只提供基础结构和基础代码，不包含具体的业务逻辑的实现。我们可以应用框架，增加需要的业务逻辑后就可以搭建任何类型的 Web 应用。

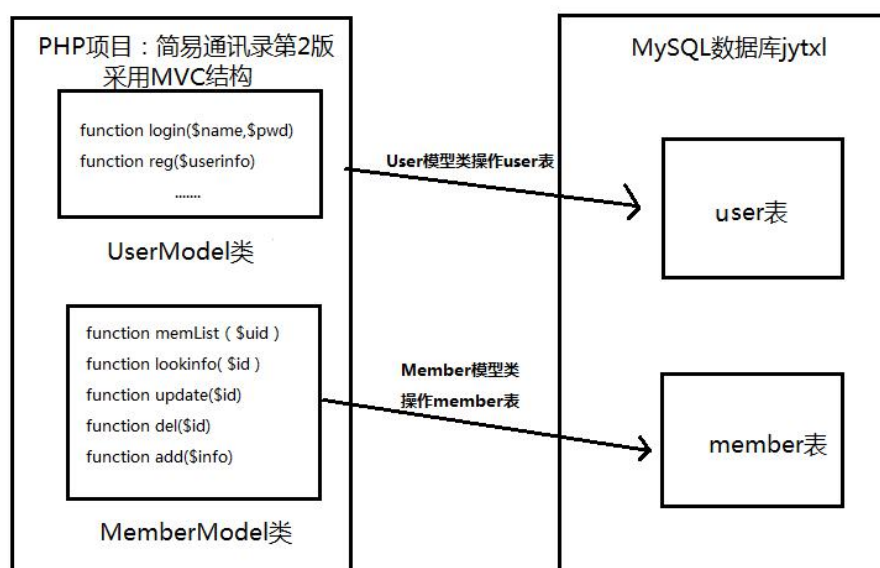
**本阶段终极任务：**应用 MVC 思想制作一个简易的框架，为了体现框架的基本结构，掌握框架的思想，为后续其他框架的学习打基础。应用这个框架实现简易通讯录的第 2 版。

## 第一步：模型层的典型实现

MVC 基本都采用面向对象的编程方式。所以实现一个项目的模型层就是为该项目编写多个模型类的过程。一个模型类就是封装了对某个表的所有的交互操作的类，所以每个需要操作的表都对应 MVC 项目中的一个模型类。

### 1、定义模型类

例如：要改写简易通讯录项目，需要创建两个模型类：UserModel 类和 MemberModel 类。



**实训：实现简易通讯录的登录功能**

需要编写如下 6 个文件：



### (1) UserModel.class.php 模型类



### (2) 登录控制器 login\_c.php



```
UserModel.class.php x login_c.php x log
1 <?php
2 //显示登录视图文件
3 require "login_v.html";
4 ?>
```

### (3) 登录视图 login\_v.html

```
UserModel.class.php x login_c.php x logaction_c.php x index_c.php x index_v.html x login_v.html x
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4 <meta charset="UTF-8">
5 <title>请您登录</title>
6 </head>
7 <body>
8 <form action="logaction_c.php" method="post" name="form1">
9 用户名: <input type="text" name="name" id="name"><br>
10 密码: <input type="password" name="pwd" id="pwd"><br>
11 <input type="submit" value="登录" onclick="return check()">
12 <p>没有账号, 去<a href="reg.html">注册</a></p>
13 </form>
14 </body>
15 </html>
16 <script>
17 function check(){
18 var error='';
19 if (form1.name.value=='') {
20 error += '用户名不能为空\n';
21 }
22 if (form1.pwd.value=='') {
23 error += '密码不能为空\n';
24 }
25 if (error!='') {
26 alert(error);
27
28 return false;
29 }
30 return true;
31 }
32 </script>
```

提交到logaction\_c.php控制器文件

### (4) 登录验证控制器 logaction\_c.php



### (5)主页控制器 index\_c.php



### (6)主页视图 index\_v.html



**新问题：**在所有的模型类中都需要 MySQLDB 类型的对象作为类属性，

并且在构造方法里要进行初始化，这样造成了代码重复的问题。如何解决呢？

## 2、基础模型类

使用继承来解决模型层的代码重复问题——创建基础模型类 Model，项目中的功能模型类都继承基础模型类。



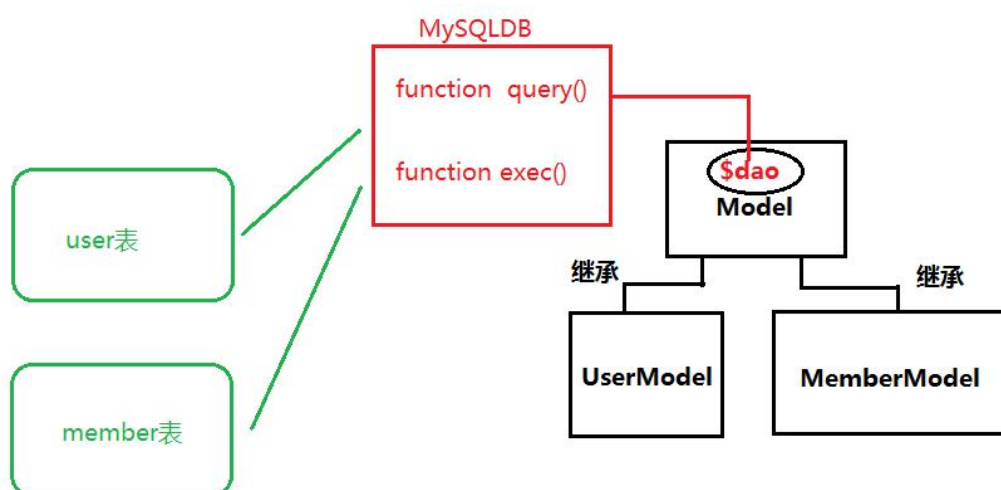
```
1 <?php
2 /**
3  * 基础模型类，被Member类继承
4  */
5 class Model
6 {
7     protected $dao; //引用MySQLDB类的单例
8     private $config=array(
9         'host'=>'localhost',
10        'port'=>'3306',
11        'user'=>'root',
12        'password'=>'root',
13        'dbName'=>'jytx1',
14        'charset'=>'utf8'
15    );
16     public function __construct()
17     {
18         $this->initDao();
19     }
20     //初始化属性$dao,代表MySQLDB的唯一实例
21     private function initDao(){
22         $this->dao = MySQLDB::getInstance($this->config);
23     }
24 }
25 ?>
```

```
1 <?php
2 include "mysqlDb.class.php";
3 include "Model.class.php";
4 /**
5  * 操作User表的模型类
6  */
7 class UserModel extends Model{
8     //从Model类继承了属性$dbao, 创建UserModel对象时实现了对$dbao属性的初始化
9     public function login($name,$pwd){
10         $data = $this->dao->query("select ID,name from user where name = '$name'
11             and pwd = '".md5($pwd)."'");
12         //如果$data是空数组说明没有找到记录, 就是用户名或密码不正确。
13         if (empty($data)) {
14             return false; //登录失败
15         }else{
16             return $data[0]; //登录成功返回包含用户ID和name的一维数组
17         }
18     }
19     public function reg($name,$pwd){
20
21     }
22 }
```

减少了重复代码, \$dao属性从父类Model类继承过来

设计基础模型类 Model 的目的就是为了让所有功能模型类自动获取一个可与数据库交互的工具 (MySQLDB 的对象), Model 类本身不需要实例化。

### 3、基础模型类 Model、功能模型类 UserModel|MemberModel 和 MySQLDB 类的关系





## 4、模型对象单例化

**新问题：**在处理用户注册时，先要验证用户是否存在，再决定是否添加到 user 表中。这两个操作是 UserModel 模型类的两个方法，这时需要创建几个模型对象来调用这两个方法比较好？

```
1 <?php
2 if (isset($_POST['name'])&&isset($_POST['pwd'])){
3     //提交表单时处理注册操作
4     //1、验证用户名是否存在，存在不可以注册
5     $user1 = new UserModel();
6     if($user1->check($_POST['name'])){
7         //存在
8         echo "<script>alert('用户名已存在，不能注册');location.href='reg_c.php';</script>";
9     }
10
11     //2、不存在则新增用户
12     /*方法1：直接使用$user1->reg()*/
13     if ($user1->reg($_POST['name'],$_POST['pwd'])) {
14         echo "<script>alert('注册成功，请登录');location.href='login_c.php';</script>";
15     }else{
16         echo "<script>alert('注册失败，重新注册');location.href='reg_c.php';</script>";
17     }
18
19     /*方法2：新建变量$user2，在使用$user2->reg()*/
20     $user2 = new UserModel();
21     if ($user2->reg($_POST['name'],$_POST['pwd'])) {
22         echo "<script>alert('注册成功，请登录');location.href='login_c.php';</script>";
23     }else{
24         echo "<script>alert('注册失败，重新注册');location.href='reg_c.php';</script>";
25     }
26 }else{
```

在上例中，更好的处理方式是创建两个引用变量，引用模型对象。因为如果后期允许用户名重名注册，就需要删除 5-9 行的逻辑代码，这时如果 reg()是由变量\$user1 调用的,则会出错。

**结论：**如果在一个控制器内有多个业务逻辑属于同一个模型类，那么根据逻辑功能“独立完整”的原则，最好是为每一个逻辑使用独立的模型对象引用变量进行调用。避免因为某个逻辑功能因为需求的变动被删除掉了，其他逻辑功能被影响。但是如果创建的引用变量引用的内存中的多个模型对象又会造成资源的浪费，所以这时可以通过模型



对象的单例化解决这个问题。——引用变量是多个，但模型对象只有一个。（如上例\$user1和\$user2引用的是同一个UserModel对象）

解决方法：采用工厂模式实现，并且处理自动加载类。

实训：完成用户的注册功能。

### （1）创建工厂类



```
1 <?php
2 //工厂类
3 class Factory
4 {
5     //获取某个模型类的单一实例的方法M 静态方法M用于创建单例模型类对象
6     public static function M($class_name)
7     {
8         //静态局部变量，函数调用后，不会消失，下次调用还会存在
9         static $instance_list = array();
10        if (!isset($instance_list[$class_name])) {
11            $instance_list[$class_name] = new $class_name();
12        }
13        return $instance_list[$class_name];
14    }
15 }
16 //自动加载类
17 function myAutoLoad($class_name){ 自定义自动加载类函数，并进行注册
18     $autoList = array(
19         'Model' => 'Model.class.php',
20         'UserModel' => 'UserModel.class.php',
21         'MySQLDB' => 'mysqlpdb.class.php'
22     );
23     if (isset($autoList[$class_name])) {
24         require $autoList[$class_name];
25     }
26 }
27 spl_autoload_register('myAutoLoad');
28 ?>
```

### （2）改写模型实例化代码：

```
reg_c.php x Factory.class.php x login_c.php x logaction_c.php x UserModel.class.php x login_v.html x
1 <?php
2 //提交了表单的话，创建模型类对象，调用登录方法
3 if (isset($_POST[name])&&isset($_POST[pwd])) {
4     include "Factory.class.php";
5     $user = Factory::M('UserModel');
6     if($result = $user->login($_POST[name],$_POST[pwd])){
7         //登录成功
8         session_start();
9         $_SESSION['jytxl_user'] = $result;
10        echo "<script>alert('登录成功');location.href='index_c.php';</script>";
11    }else{
```

(3) 在 UserModel 类中增加方法 check()用于验证用户是否存在。

```
reg_c.php x Factory.class.php x login_c.php x logaction_c.php x UserModel.class.php x login_v.html x reg_v.html
22         return false; //注册失败
23     }else{
24         return true; //注册成功
25     }
26 }
27 /**
28  * [check 检查用户名是否存在]
29  * @param [string] $name [用户名]
30  * @return [boolean] [true表示存在，false表示不存在]
31  */
32 public function check($name){
33     $result = $this->dao->query("select * from user where name='{ $name}'");
34     //如果$result为false则代表插入失败，如果等于1说明插入成功1条记录
35
36     if (!empty($result)) {
37         return true; //存在此用户名
38     }else{
39         return false; //不存在此用户名
40     }
41 }
42 }
```

(4) 新建注册视图 reg\_v.html.

```
reg_c.php x Factory.class.php x reg_v.html x login_c.php x logaction_c.php x UserModel.class.php x
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>注册</title>
6 </head>
7 <body>
8     <form action="reg_c.php" method="post" name="form1">
9         用户名: <input type="text" name="name" id="name"><br>
10        密码: <input type="password" name="pwd" id="pwd"><br>
11        重复密码: <input type="password" name="pwd1" id="pwd1"><br>
12        <input type="submit" value="注册" onclick="return check()">
13    </form>
14 </body>
15 </html>
16 <script>
17 function check(){
18     var error='';
19     if (form1.name.value=='') {
20         error += '用户名不能为空\n';
21     }
22     if (form1.pwd.value=='') {
23         error += '密码不能为空\n';
24     }
25     if (form1.pwd.value!=form1.pwd1.value) {
26         error += '两次密码不一致\n';
27     }
28     if (error!='') {
29         alert(error);
30         return false;
31     }
32     return true;
33 }
34 }
35 </script>
```

(5) 新建注册控制器 reg\_c.php。



```
<?php
1
2  if (isset($_POST['name'])&&isset($_POST['pwd'])){
3      //提交表单时处理注册操作
4      //1. 验证用户名是否存在, 存在不可以注册
5      require "Factory.class.php";
6      $user1 = Factory::M("UserModel");
7
8      if($user1->check($_POST['name'])){
9          //存在
10         echo "<script>alert('用户名已存在, 不能注册');location.href='reg_c.php';</script>";
11     }
12
13     //2. 不存在则新增用户
14     $user2 = Factory::M('UserModel');
15     if ($user2->reg($_POST['name'],$_POST['pwd'])) {
16         echo "<script>alert('注册成功, 请登录');location.href='login_c.php';</script>";
17     }else{
18         echo "<script>alert('注册失败, 重新注册');location.href='reg_c.php';</script>";
19     }
20 }else{
21     //为提交表单时显示注册视图
22     require "reg_v.html";
23 }
24 ?>
```

## 第二步：控制器层的实现

MVC 基本都采用面向对象的编程方式。所以实现一个项目的控制器层就是为该项目编写多个控制器类的过程。一个控制器类就是封装了一系列相关的操作的类，每个操作就是控制器类中的一个方法，称之为动作（action）。

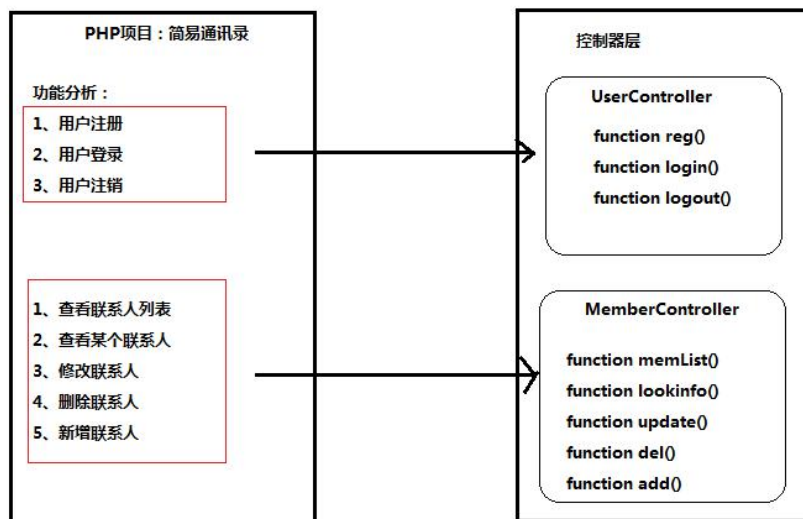
### 1、定义控制器类

采用面向对象的实现效果就是：

控制器类——存储一系列功能。

控制器类中定义的某个方法，对应某个具体的功能。





**实训：**编写控制器类 `UserController`,实现与用户相关的所有功能（注册操作、登录操作、注销操作）。

```
UserController.class.php x
1  <?php
2  /**
3   * 和用户相关的所有操作封装在UserController控制器类中
4   */
5  class UserController
6  {
7
8      /**
9       * [login 登录]
10      * @return [type] [description]
11      */
12     function login()
13     {
14         //显示登录视图
15         require "login_v.html";
16     }
17     /**
18      * [logaction 验证登录]
19      * @return [void] [无返回值，直接跳转]
20      */
21     function logaction(){
22         //提交了表单的话，创建模型类对象，调用登录方法
23         if (isset($_POST[name])&&isset($_POST[pwd])) {
24             include "Factory.class.php";
```



```

25         $user = Factory::M('UserModel');
26         if($result = $user->login($_POST[name],$_POST[pwd])){
27             //登录成功
28             session_start();
29             $_SESSION['jytxl_user'] = $result;
30             echo "<script>alert('登录成功');location.href='index_c.php';</script>";
31         }else{
32             //登录失败
33             echo "<script>alert('登录失败');location.href='login_c.php';</script>";
34         }
35     }else{
36         //没有提交表单直接访问的话跳回到登录控制器
37         header("location:login_c.php");
38     }
39 }
40 /**
41  * [reg 提供注册界面及注册处理]
42  * @return [void] [无返回值]
43  */
44 function reg(){
45     if (isset($_POST['name'])&&isset($_POST['pwd'])){
46         //提交表单时处理注册操作
47
48         //1、验证用户名是否存在，存在不可以注册
49         include "Factory.class.php";
50         $user1 = Factory::M('UserModel');
51         if($user1->check($_POST['name'])){
52             //存在
53             echo "<script>
54                 alert('用户名已存在，不能注册');location.href='reg_c.php';</script>";
55         }
56
57         //2、不存在则新增用户
58
59         /*方法2：新建变量$user2,再使用$user2->reg()*/
60         $user2 = Factory::M('UserModel');
61         if ($user2->reg($_POST['name'],$_POST['pwd'])) {
62             echo "<script>
63                 alert('注册成功，请登录');location.href='login_c.php';</script>"
64             ;
65         }else{
66             echo "<script>
67                 alert('注册失败，重新注册');location.href='reg_c.php';</script>"
68             ;
69         }
70     }else{
71         //没提交表单时显示注册视图
72         require "reg_v.html";
73     }
74 }
75 }
76 }
77 }
78 }
79 }
80 }
81 }
82 }
83 }
84 }
85 }
86 }
87 }
88 }
89 }
90 }
91 }
92 }
93 }
94 }
95 }
96 }
97 }
98 }
99 }
100 }
101 }
102 }
103 }
104 }
105 }
106 }
107 }
108 }
109 }
110 }
111 }
112 }
113 }
114 }
115 }
116 }
117 }
118 }
119 }
120 }
121 }
122 }
123 }
124 }
125 }
126 }
127 }
128 }
129 }
130 }
131 }
132 }
133 }
134 }
135 }
136 }
137 }
138 }
139 }
140 }
141 }
142 }
143 }
144 }
145 }
146 }
147 }
148 }
149 }
150 }
151 }
152 }
153 }
154 }
155 }
156 }
157 }
158 }
159 }
160 }
161 }
162 }
163 }
164 }
165 }
166 }
167 }
168 }
169 }
170 }
171 }
172 }
173 }
174 }
175 }
176 }
177 }
178 }
179 }
180 }
181 }
182 }
183 }
184 }
185 }
186 }
187 }
188 }
189 }
190 }
191 }
192 }
193 }
194 }
195 }
196 }
197 }
198 }
199 }
200 }
201 }
202 }
203 }
204 }
205 }
206 }
207 }
208 }
209 }
210 }
211 }
212 }
213 }
214 }
215 }
216 }
217 }
218 }
219 }
220 }
221 }
222 }
223 }
224 }
225 }
226 }
227 }
228 }
229 }
230 }
231 }
232 }
233 }
234 }
235 }
236 }
237 }
238 }
239 }
240 }
241 }
242 }
243 }
244 }
245 }
246 }
247 }
248 }
249 }
250 }
251 }
252 }
253 }
254 }
255 }
256 }
257 }
258 }
259 }
260 }
261 }
262 }
263 }
264 }
265 }
266 }
267 }
268 }
269 }
270 }
271 }
272 }
273 }
274 }
275 }
276 }
277 }
278 }
279 }
280 }
281 }
282 }
283 }
284 }
285 }
286 }
287 }
288 }
289 }
290 }
291 }
292 }
293 }
294 }
295 }
296 }
297 }
298 }
299 }
300 }
301 }
302 }
303 }
304 }
305 }
306 }
307 }
308 }
309 }
310 }
311 }
312 }
313 }
314 }
315 }
316 }
317 }
318 }
319 }
320 }
321 }
322 }
323 }
324 }
325 }
326 }
327 }
328 }
329 }
330 }
331 }
332 }
333 }
334 }
335 }
336 }
337 }
338 }
339 }
340 }
341 }
342 }
343 }
344 }
345 }
346 }
347 }
348 }
349 }
350 }
351 }
352 }
353 }
354 }
355 }
356 }
357 }
358 }
359 }
360 }
361 }
362 }
363 }
364 }
365 }
366 }
367 }
368 }
369 }
370 }
371 }
372 }
373 }
374 }
375 }
376 }
377 }
378 }
379 }
380 }
381 }
382 }
383 }
384 }
385 }
386 }
387 }
388 }
389 }
390 }
391 }
392 }
393 }
394 }
395 }
396 }
397 }
398 }
399 }
400 }
401 }
402 }
403 }
404 }
405 }
406 }
407 }
408 }
409 }
410 }
411 }
412 }
413 }
414 }
415 }
416 }
417 }
418 }
419 }
420 }
421 }
422 }
423 }
424 }
425 }
426 }
427 }
428 }
429 }
430 }
431 }
432 }
433 }
434 }
435 }
436 }
437 }
438 }
439 }
440 }
441 }
442 }
443 }
444 }
445 }
446 }
447 }
448 }
449 }
450 }
451 }
452 }
453 }
454 }
455 }
456 }
457 }
458 }
459 }
460 }
461 }
462 }
463 }
464 }
465 }
466 }
467 }
468 }
469 }
470 }
471 }
472 }
473 }
474 }
475 }
476 }
477 }
478 }
479 }
480 }
481 }
482 }
483 }
484 }
485 }
486 }
487 }
488 }
489 }
490 }
491 }
492 }
493 }
494 }
495 }
496 }
497 }
498 }
499 }
500 }
501 }
502 }
503 }
504 }
505 }
506 }
507 }
508 }
509 }
510 }
511 }
512 }
513 }
514 }
515 }
516 }
517 }
518 }
519 }
520 }
521 }
522 }
523 }
524 }
525 }
526 }
527 }
528 }
529 }
530 }
531 }
532 }
533 }
534 }
535 }
536 }
537 }
538 }
539 }
540 }
541 }
542 }
543 }
544 }
545 }
546 }
547 }
548 }
549 }
550 }
551 }
552 }
553 }
554 }
555 }
556 }
557 }
558 }
559 }
560 }
561 }
562 }
563 }
564 }
565 }
566 }
567 }
568 }
569 }
570 }
571 }
572 }
573 }
574 }
575 }
576 }
577 }
578 }
579 }
580 }
581 }
582 }
583 }
584 }
585 }
586 }
587 }
588 }
589 }
590 }
591 }
592 }
593 }
594 }
595 }
596 }
597 }
598 }
599 }
600 }
601 }
602 }
603 }
604 }
605 }
606 }
607 }
608 }
609 }
610 }
611 }
612 }
613 }
614 }
615 }
616 }
617 }
618 }
619 }
620 }
621 }
622 }
623 }
624 }
625 }
626 }
627 }
628 }
629 }
630 }
631 }
632 }
633 }
634 }
635 }
636 }
637 }
638 }
639 }
640 }
641 }
642 }
643 }
644 }
645 }
646 }
647 }
648 }
649 }
650 }
651 }
652 }
653 }
654 }
655 }
656 }
657 }
658 }
659 }
660 }
661 }
662 }
663 }
664 }
665 }
666 }
667 }
668 }
669 }
670 }
671 }
672 }
673 }
674 }
675 }
676 }
677 }
678 }
679 }
680 }
681 }
682 }
683 }
684 }
685 }
686 }
687 }
688 }
689 }
690 }
691 }
692 }
693 }
694 }
695 }
696 }
697 }
698 }
699 }
700 }
701 }
702 }
703 }
704 }
705 }
706 }
707 }
708 }
709 }
710 }
711 }
712 }
713 }
714 }
715 }
716 }
717 }
718 }
719 }
720 }
721 }
722 }
723 }
724 }
725 }
726 }
727 }
728 }
729 }
730 }
731 }
732 }
733 }
734 }
735 }
736 }
737 }
738 }
739 }
740 }
741 }
742 }
743 }
744 }
745 }
746 }
747 }
748 }
749 }
750 }
751 }
752 }
753 }
754 }
755 }
756 }
757 }
758 }
759 }
760 }
761 }
762 }
763 }
764 }
765 }
766 }
767 }
768 }
769 }
770 }
771 }
772 }
773 }
774 }
775 }
776 }
777 }
778 }
779 }
780 }
781 }
782 }
783 }
784 }
785 }
786 }
787 }
788 }
789 }
790 }
791 }
792 }
793 }
794 }
795 }
796 }
797 }
798 }
799 }
800 }
801 }
802 }
803 }
804 }
805 }
806 }
807 }
808 }
809 }
810 }
811 }
812 }
813 }
814 }
815 }
816 }
817 }
818 }
819 }
820 }
821 }
822 }
823 }
824 }
825 }
826 }
827 }
828 }
829 }
830 }
831 }
832 }
833 }
834 }
835 }
836 }
837 }
838 }
839 }
840 }
841 }
842 }
843 }
844 }
845 }
846 }
847 }
848 }
849 }
850 }
851 }
852 }
853 }
854 }
855 }
856 }
857 }
858 }
859 }
860 }
861 }
862 }
863 }
864 }
865 }
866 }
867 }
868 }
869 }
870 }
871 }
872 }
873 }
874 }
875 }
876 }
877 }
878 }
879 }
880 }
881 }
882 }
883 }
884 }
885 }
886 }
887 }
888 }
889 }
890 }
891 }
892 }
893 }
894 }
895 }
896 }
897 }
898 }
899 }
900 }
901 }
902 }
903 }
904 }
905 }
906 }
907 }
908 }
909 }
910 }
911 }
912 }
913 }
914 }
915 }
916 }
917 }
918 }
919 }
920 }
921 }
922 }
923 }
924 }
925 }
926 }
927 }
928 }
929 }
930 }
931 }
932 }
933 }
934 }
935 }
936 }
937 }
938 }
939 }
940 }
941 }
942 }
943 }
944 }
945 }
946 }
947 }
948 }
949 }
950 }
951 }
952 }
953 }
954 }
955 }
956 }
957 }
958 }
959 }
960 }
961 }
962 }
963 }
964 }
965 }
966 }
967 }
968 }
969 }
970 }
971 }
972 }
973 }
974 }
975 }
976 }
977 }
978 }
979 }
980 }
981 }
982 }
983 }
984 }
985 }
986 }
987 }
988 }
989 }
990 }
991 }
992 }
993 }
994 }
995 }
996 }
997 }
998 }
999 }
1000 }

```

**新问题：**如何执行控制器类 UserController 的动作？

## 2、前端控制器(入口文件)

解决方法: 新建 index.php, 在此文件中实例化控制器类 UserController 的对象, 并且调用控制器类内的方法(动作)实现功能。这个 index.php 称为前端控制器。



```
1 <?php
2
3 include "Factory.class.php";
4 //处理登录 前端控制器中创建UserController对象, 调用login方法处理登录
5
6 $user_c = new UserController();
7 $user_c->login();
8
9 ?>
```

新问题:

用户登录验证表单页提交到哪里? 难道再新建一个前端控制器? 答案是否定的。那么如何使前端控制器 index.php 做到可以执行任意控制器类的任意动作?



```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4 <meta charset="UTF-8">
5 <title>请您登录</title>
6 </head>
7 <body>
8 <form action="logaction_c.php" method="post" name="form1">
9 用户名: <input type="text" name="name" id="name"><br>
10 密码: <input type="password" name="pwd" id="pwd"><br>
11 <input type="submit" value="登录" onclick="return check();">
12 <p>没有账号, 去<a href="reg_c.php">注册</a></p>
13 </form>
14 </body>
15 </html>
```

## 3、前端控制器的分发处理。

解决方法: 通过在请求 index.php 时, 向其传递 get 参数, 包含要访

问的控制器类和具体的操作。引入 URL 参数 c 和 URL 参数 a。

c(controller)、a(action)

例如：

用户登录                      index.php?c=User&a=login

用户登录验证                index.php?c=User&a=logaction

查看联系人信息列表        index.php?c=Member&a=memList

更新联系人信息            index.php?c=Member&a=update&mid=2

注意：参数 c 的值是控制器类的简写形式,如 User 而不是

UserController。

如何设置 c 参数和 a 参数？

通过超链接或者在表单提交 action 或页面间跳转时传递参数。



```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>请您登录</title>
6 </head>
7 <body>
8   <form action="index.php?c=User&a=logaction" method="post" name="form1">
9     用户名: <input type="text" name="name" id="name"><br>
10    密码: <input type="password" name="pwd" id="pwd"><br>
11    <input type="submit" value="登录" onclick="return check()">
12    <p>没有账号, 去<a href="index.php?c=User&a=reg">注册</a></p>
13  </form>
14 </body>
15 </html>
```

接下来只需要在 index.php 中读取\$\_GET['c']来创建控制器类对象，读取\$\_GET['a']来获取调用哪个方法。

**实训：**完成用户登录后显示主页。

(1) 修改前端控制器文件 index.php。

在入口文件中统一开启会话。



```
1  <?php
2  session_start();//开启会话
3  /**
4   * 第一步：判断要执行的控制器和操作
5   */
6  //指定默认的控制类和控制类名
7  $defaultControlClass = "IndexController";
8  $defaultControl = 'Index';
9
10 $currentControlClass=''; //当前的控制器类名
11 $currentControl=''; //当前的控制器名
12
13 //通过URL参数了解查看哪个控制器
14 if (isset($_GET['c'])&&$_GET['c']!='') {
15     $currentControlClass = $_GET['c']."Controller";
16     $currentControl = $_GET['c'];
17 }else{
18     $currentControlClass = $defaultControlClass;
19     $currentControl = $defaultControl;
20 }
21
22 //判断控制器类文件是否被定义
23 if (!is_file($currentControlClass.'.class.php')) {
24     die('您访问的控制器类不存在');
25 }
26
27 $currentAction='index'; //默认执行index操作
28 //通过URL参数了解查看控制器的哪个操作
29 if (isset($_GET['a'])&&$_GET['a']!='') {
30     $currentAction = $_GET['a'];
31 }
32
33 /**
34 * 第二步：创建控制器对象、执行操作
35 */
36 include "Factory.class.php";
37
38 //创建控制器类对象
39 $c_obj = new $currentControlClass(); //可变类
40 //执行操作
41 $c_obj->$currentAction(); //可变方法
42 ?>
```

(2) 修改 UserController 类中的页面跳转地址。



```

UserController.class.php x index.php x index_c.php x IndexController.class.php x MemberController.class.php x login_v.html x
1 <?php
2 /**
3  * 和用户相关的所有操作封装在UserController控制器类中
4  */
5 class UserController
6 {
7     //魔术方法,当调用不存在的操作时自动调用处理
8     public function __call($funName,$args){
9         die("UserController不存在{$funName}操作");
10    }
11
12    /**
13     * [logaction 验证登录]
14     * @return [void] [无返回值, 直接跳转]
15     */
16    function logaction(){
17        //提交了表单的话, 创建模型类对象, 调用登录方法
18        if (isset($_POST['name'])&&isset($_POST['pwd'])) {
19            $user = Factory::M('UserModel');
20            if($result = $user->login($_POST['name'],$_POST['pwd'])){
21                //登录成功
22                session_start();
23                $_SESSION['jytxl_user'] = $result;
24                echo "<script>alert('登录成功');location.href='index.php';</script>";
25            }else{
26                //登录失败
27                echo "<script>alert('登录失败');location.href='index.php?c=User&a=login';</script>";
28            }
29        }else{
30            //没有提交表单直接访问的话跳回到登录控制器
31            header("location:index.php?c=User&a=login");
32        }
33    }
34
35    /**
36     * [reg 提供注册界面及注册处理]
37     * @return [void] [无返回值]
38     */
39    function reg(){
40        if (isset($_POST['name'])&&isset($_POST['pwd'])){
41            //提交表单时处理注册操作
42            //1、验证用户名是否存在, 存在不可以注册
43            $user1 = Factory::M('UserModel');
44            if($user1->check($_POST['name'])){
45                //存在
46                echo "<script>alert('用户名已存在, 不能注册');location.href='index.php?c=User&a=reg';</script>";
47            }
48
49            //2、不存在则新增用户
50
51            /*方法2: 新建变量$user2,再使用$user2->reg()*/
52            $user2 = Factory::M('UserModel');
53            if ($user2->reg($_POST['name'],$_POST['pwd'])) {
54                echo "<script>alert('注册成功, 请登录');location.href='index.php?c=User&a=login';</script>";
55            }else{
56                echo "<script>alert('注册失败, 重新注册');location.href='index.php?c=User&a=reg';</script>";
57            }
58        }else{
59            //没提交表单时显示注册视图
60            require "reg_v.html";
61        }
62    }
63
64 }
65
66 }
67
68 }
69
70 }
71
72
73
74 ?>

```

新增魔术方法定义, 处理访问不存在的操作的情况



所有页面的跳转地址都要改成带分发参数形式的 URL 地址。例如注册页：



```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>注册</title>
6 </head>
7 <body>
8     <form action="index.php?c=User&a=reg" method="post" name="form1">
9         用户名: <input type="text" name="name" id="name"><br>
10        密码: <input type="password" name="pwd" id="pwd"><br>
11        重复密码: <input type="password" name="pwd1" id="pwd1"><br>
12        <input type="submit" value="注册" onclick="return check()>
13    </form>
14 </body>
15 </html>
```

(3) 修改 index\_v.html 文件名为 memList\_v.html，为后续添加联系人列表作准备。



```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>简易通讯录</title>
6 </head>
7 <body>
8     <h2><?=$_SESSION['jytxl_user']['name']?>的通讯录</h2>
9     <p>如下显示当前用户的联系人列表，待完成</p>
10 </body>
11 </html>
```

由原来的index\_v.html更名为memList\_v.html  
表示查看联系人列表

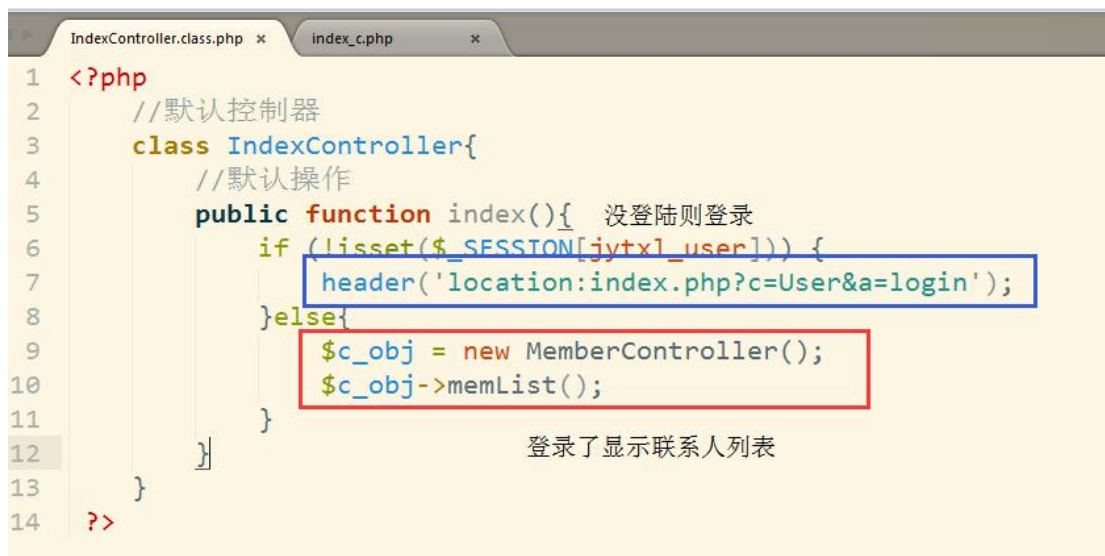
(4) 新建 MemberController，定义 memList()方法、lookinfo()方法等。



```
1 <?php
2 /**
3  * 和联系人相关的所有操作封装在MemberController控制器类中
4  */
5  class MemberController
6  {
7      //魔术方法,当调用不存在的操作时自动调用处理
8      public function __call($funName,$args){
9          die("User控制器不存在{$funName}操作");
10     }
11
12     function memList()
13     {
14         require "memList_v.html";
15     }
16     function lookInfo($id){
17
18     }
19 }
20 ?>
```

查看联系人列表的动作

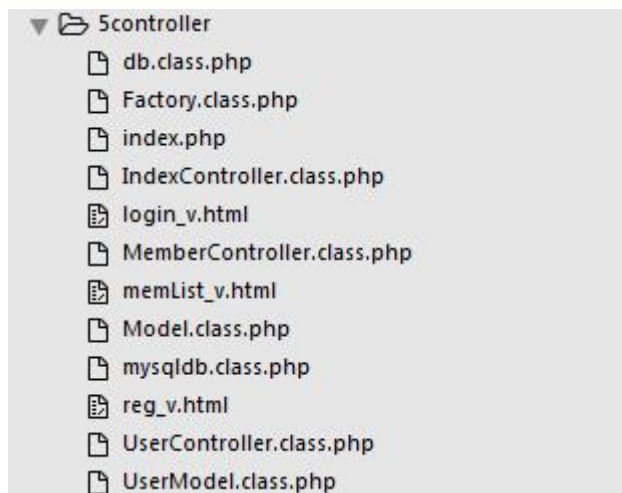
(5) 新建默认的控制类 IndexController, 增加默认操作 index(), 默认执行 MemberController::memList().



```
1 <?php
2 //默认控制器
3 class IndexController{
4     //默认操作
5     public function index(){ 没登陆则登录
6         if (!isset($_SESSION['jytxl_user'])) {
7             header('location:index.php?c=User&a=login');
8         }else{
9             $c_obj = new MemberController();
10            $c_obj->memList();
11        }
12    }
13 }
14 ?>
```

登录了显示联系人列表

(6) 删除无用的文件。全新的网站结构如下:



包含 3 个控制器类（MemberController、UserController、IndexController），1 个前端控制器文件 index.php，3 个视图文件（login\_v.html、reg\_v.html、memList\_v.html）。

总结：c 和 a 参数称之为请求分发参数，index.php 也被称之为请求分发器。Index.php 又叫入口文件。就是说只要是项目下的功能，都需要经由该 index.php 完成，也就是项目的入口。

## 第三步：项目目录结构划分

### 1、相对路径的问题

（1）当文件被其他文件包含时，采用相对路径可能会造成不可预知的错误。

（2）当前工作路径（CWD）

（3）从入口文件执行所有功能解决了这个问题。

所有的功能执行的当前工作目录：

## 2、分目录存储

将文件根据功能的不同放在不同的目录中。

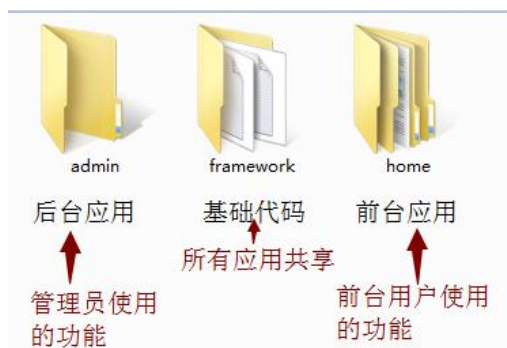
### (1) 划分原则：基础代码与功能逻辑代码分离。

① 基础代码是所有项目都可以使用的代码，属于框架部分，如 MySQLDB、Model、Factory;

② 功能逻辑代码是本项目需要的处理业务逻辑的代码，如 UserModel,UserController 等。

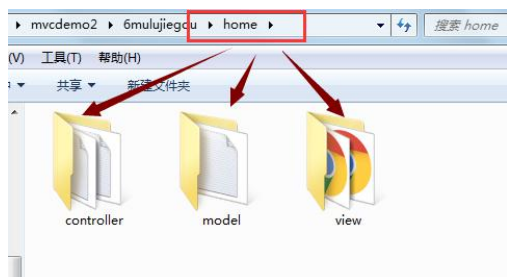
### (2) 划分步骤：

第一步：设计总体项目架构。



第二步：提取属于公共使用的框架部分文件至 framework 文件夹中。

第三步：在每个应用中创建 Controller、Model、View 三个子目录。



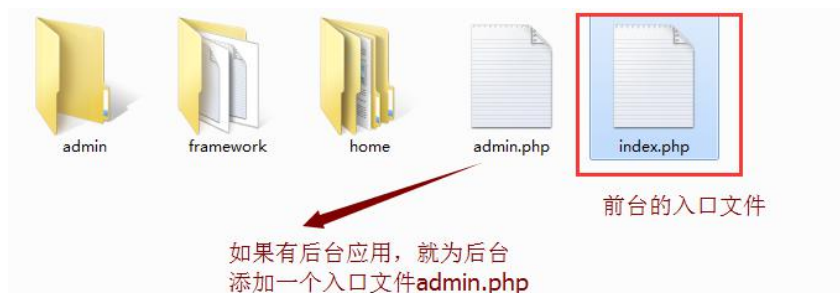
第四步：图片、CSS、JS 资源放置位置：属于视图层，放置在

view/resource/images,view/resource/css,view/resource/js 中。

第五步：提取各个应用的代码，按照类别分别放至 home 中的 Controller、Model、View 中。

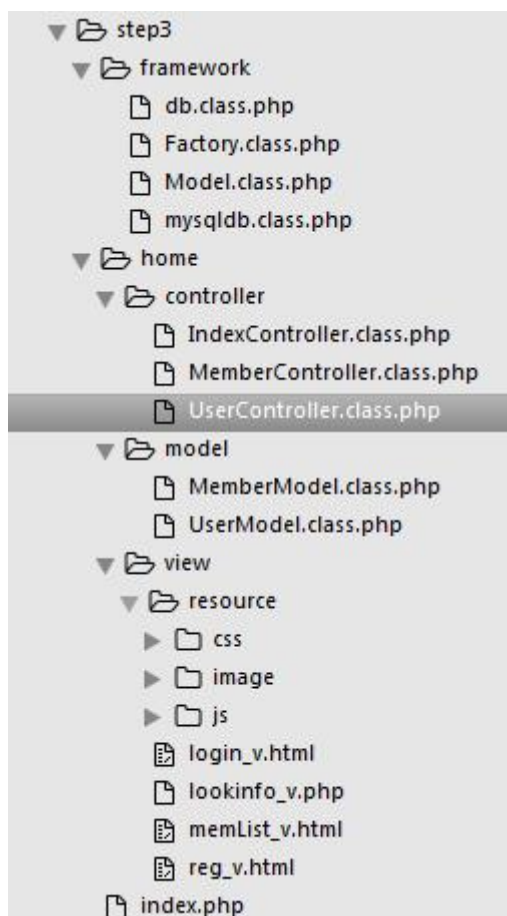


第六步：保证每个应用的入口文件在项目的根目录下。从而保证浏览器能够快速找到相应的应用。





### (3) 总结目录结构树



### 3、修改路径使功能正常运行

主要工作：处理所有的路径的变更；修改类的自动加载。

(1)修改入口文件 index.php。定义几个常量保存新增的路径。

```
reg_v.html x index.php x Factory.class.php x IndexController.class.php x Memt
1 <?php
2 session_start();//开启会话
3
4 //为增加的路径结构定义常量，方便使用
5 define('FW', './framework/');
6 define('APP', './home/');
7 define('MODEL', './home/model/');
8 define('CONTROLLER', './home/controller/');
9 define('VIEW', './home/view/');
10
11 /**
12  * 第一步：判断要执行的控制器和操作
13  */
14 //指定默认的控制器类和控制器名
15 $defaultControlClass = "IndexController";
16 $defaultControl = 'Index';
17
18 $currentControlClass=''; //当前的控制器类名
19 $currentControl=''; //当前的控制器名
20
21 //判断控制器类文件是否被定义
22 if (!is_file(CONTROLLER.$currentControlClass.'.class.php')) {
23     die('您访问的控制器类不存在');
24 }
25
26 $currentAction='index'; //默认执行index操作
27 //通过URL参数了解查看控制器的哪个操作
28 if (isset($_GET['a'])&&$_GET['a']!='') {
29     $currentAction = $_GET['a'];
30 }
31
32 /**
33  * 第二步：创建控制器对象、执行操作
34  */
35 include FW."Factory.class.php";
36
37 //创建控制器类对象
38 $c_obj = new $currentControlClass(); //可变类
39 //执行操作
40 $c_obj->$currentAction(); //可变方法
41 ?>
```

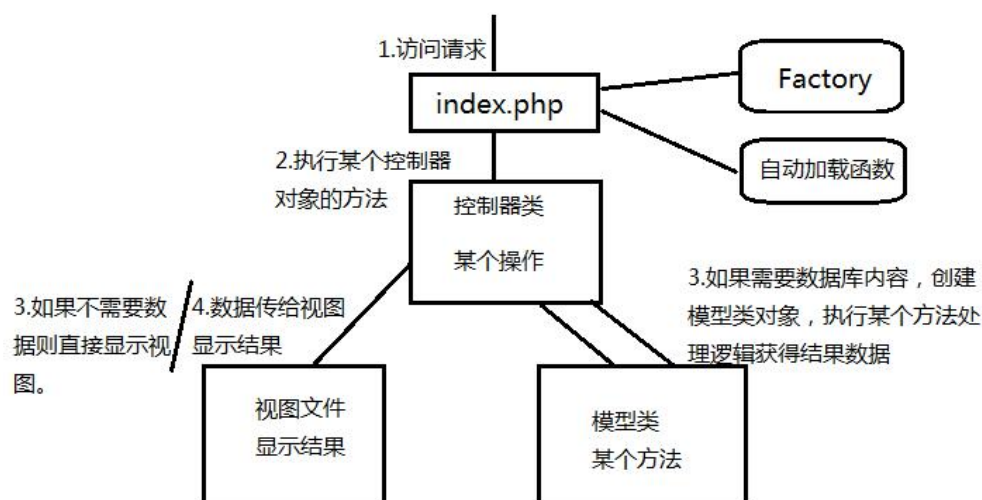
(2)工厂类中自动加载添加新的处理逻辑，对框架里的类和控制器模型类分别处理。

```
15 }
16 //自动加载类
17 function myAutoLoad($class_name){
18     //映射列表下定义框架公共类
19     $autoList = array(
20         'Model'=>FW.'Model.class.php',
21         'MySQLDB'=>FW.'mysqlpdb.class.php'
22     );
23     if (isset($autoList[$class_name])) {
24         require $autoList[$class_name];
25         return; //加载成功就返回
26     }
27     //在M、C层下的类通过命名规则定义
28     if (substr($class_name,-10)=='Controller') {
29         require CONTROLLER.$class_name.".class.php";
30         return; //加载成功就返回
31     }else if(substr($class_name,-5)=='Model'){
32         require MODEL.$class_name.".class.php";
33         return; //加载成功就返回
34     }
35 }
36 spl_autoload_register('myAutoLoad');
37 ?>
```

(3)修改各个页面内的文件包含路径。例如视图文件的包含。

```
UserController.class.php *
1 <?php
2 /**
3  * 和用户相关的所有操作封装在UserController控制器类中
4  */
5 class UserController
6 {
7     //魔术方法,当调用不存在的操作时自动调用处理
8     public function __call($funName,$args){
9         die("User控制器不存在{$funName}操作");
10    }
11    /**
12     * [login 登录]
13     * @return [type] [description]
14     */
15    function login()
16    {
17        //显示登录视图
18        require VIEW."login_v.html";
19    }
}
```

#### (4)页面加载顺序:



### 第四步：页面跳转

在项目中，控制器的操作里经常处理页面跳转。例如登录成功跳转到主页，登录失败跳回登录页；注册成功跳转到登录页等等。我们可以将跳转的操作封装在控制器的成员方法里，需要时调用即可。

所有的控制器都可能需要跳转的操作，所以定义一个所有控制器类的父类 **Controller（基础控制器类）** 来定义这些公共操作是可行的。

#### 1、直接跳转

例如：`header('Location:index.php?c=User&a=login');`

#### 2、提示一个信息后再跳转

例如：`echo "<script>alert('成功');location.href='index.php';</script>";`

在控制器内经常出现打印出的中文乱码问题，原因是控制器本身没有指定页面类型及使用的编码格式。所以也可以在 **Controller** 类中增加定义文档类型的方法。



```
Controller.class.php
1 <?php
2 /**
3  * 基础控制器类。属于框架代码。所有控制器类的父类
4  */
5 class Controller 基础控制器类
6 {
7     function __construct()
8     {
9         $this->setContentType();
10    }
11    //声明文档类型
12    protected function setContentType($charset = 'utf8'){
13        header("Content-type:text/html; charset={$charset}");
14    }
15    //直接跳转到$url
16    protected function redirectNow($url='') {
17        header("Location:$url");
18        die(); //header跳转后代码会继续执行,所以需要die()来终止执行。
19    }
20    //提示信息后再跳转页面到$url
21    protected function redirectMess($url='', $mess='') {
22        echo "<script>alert('{ $mess }'); location.href='{ $url }';</script>";
23        die(); //js跳转后代码会继续执行,所以需要die()来终止执行。
24    }
25 }
26 ?>
```

## 第五步：集中验证的处理

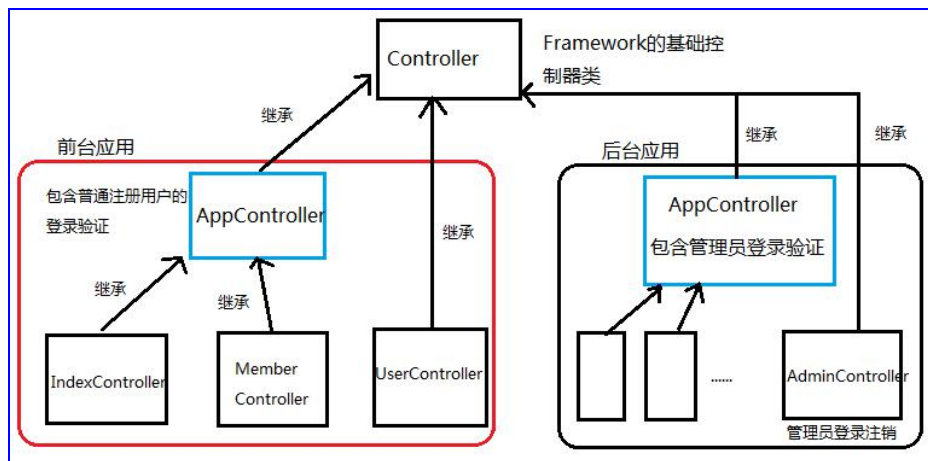
在一个网站的后台应用中，所有的模块的所有操作都是需要先登录后才能执行的，但在所有操作内都验证 Session 信息太繁琐不现实。那么如何实现集中验证呢？

将登陆验证的代码放置在一个公共位置可以解决所有控制器的访问，那么是放在基础控制器类 Controller 中吗？所有应用的控制器类都会继承 Controller 类，如果在 Controller 类中添加集中验证代码将导致某些无需验证身份的应用也得自动验证，这是不合理的。

如果在各个应用中添加自己独立的基础控制器类 **AppController**，在这个基础控制器类添加集中验证权限的代码，并让需要验证的应用



内的相应控制器继承 `AppController` 即可。注意：登录、注册等功能应该越过权限验证，否则将陷入死循环。



```
framework
├── Controller.class.php
├── db.class.php
├── Factory.class.php
├── Model.class.php
├── mysql.class.php
├── home
│   └── controller
│       ├── AppController.class.php
│       ├── IndexController.class.php
│       ├── MemberController.class.php
│       └── UserController.class.php
├── model
│   └── UserModel.class.php
└── view
    ├── login_v.html
    ├── memList_v.html
    └── reg_v.html
```

```
<?php
/**
 * 应用基础控制器,为了本应用内控制器集中验证登录权限
 */
class AppController extends Controller
{
    function __construct()
    {
        //由于重写了父类Controller的构造方法,所以需要显式调用Controller的构造方法
        parent::__construct();
        //放在构造方法里让控制器自动验证登录权限
        $this->checkAuthority();
    }
    /**
     * [checkAuthority 验证登录权限]
     * @return [void] [无返回值]
     */
    private function checkAuthority(){
        if (!isset($_SESSION['jytxl_user'])) {
            $this->redirectNow('index.php?c=User&a=login');
            exit;
        }
    }
}
```

```
<?php
//默认控制器
class IndexController extends AppController{
    //默认操作
    public function index(){
        $c_obj = new MemberController();
        $c_obj->memList();
    }
    //index()内无需再验证
}
```

```
<?php
/**
 * 和联系人相关的所有操作封装在MemberController控制器类中
 */
class MemberController extends AppController
{
    function memList()
    {
        require VIEW."memList_v.html";
    }
    function lookInfo($id){}
}
```

```
<?php
/**
 * 和用户相关的所有操作封装在UserController控制器类中
 */
class UserController extends Controller
{
    /**
     * [login 登录]
     * @return [type] [description]
     */
    function login()
    {
        //注意: UserController越过AppController
        //继承Controller,否则将无法实现登录进入死循环
    }
}
```

新的问题：如果除了 `User` 控制器，其他控制器的某些操作不需要登

录验证，即存在某些特例情况，如何将它们排除在验证外？

例如：简易通讯录第2版中，增加一个项目介绍功能。该功能不用登录就能查看，如何实现？



The first screenshot shows the `IndexController` class in `IndexController.class.php`, which extends `AppController`. It includes an `index()` method that initializes a `MemberController` object and calls `memList()`, and a `sysInfo()` method that requires the `sysInfo.html` view. The `sysInfo()` method is highlighted with a red box.

The second screenshot shows the `sysInfo.html` view, which contains HTML markup for a project introduction page, including a title "项目简介" and a description of the system.

The third screenshot shows the `AppController` class, specifically the `checkAuthority()` method. It highlights the logic for setting the `$currentAction` to `'sysInfo'` and defining constants `CUR_C` and `CUR_A` for the current controller and action. Red boxes and arrows highlight these specific lines of code.

解决方法：在 `AppController` 的 `checkAuthority()` 中定义一个不用验证

的列表\$noCheckList，将可以不用验证的控制器类的操作名写在列表中，然后判断当前执行的控制器类的操作是不是在列表中，如果是直接返回不用验证。



## 第六步：制作视图层

采用 Smarty 模板引擎技术将视图层的静态代码与 PHP 代码彻底分离。