# **运行框架**

## **一、实验简介**

### **1.1 实验目的**

本次实验将做一些后期的完善和细节的调整，测试我们编写的框架。

### **1.2 实验**

* sublime，一个方便快速的文本编辑器。点击桌面左下角： 应用程序菜单/开发/sublime

### **1.3 预期效果**

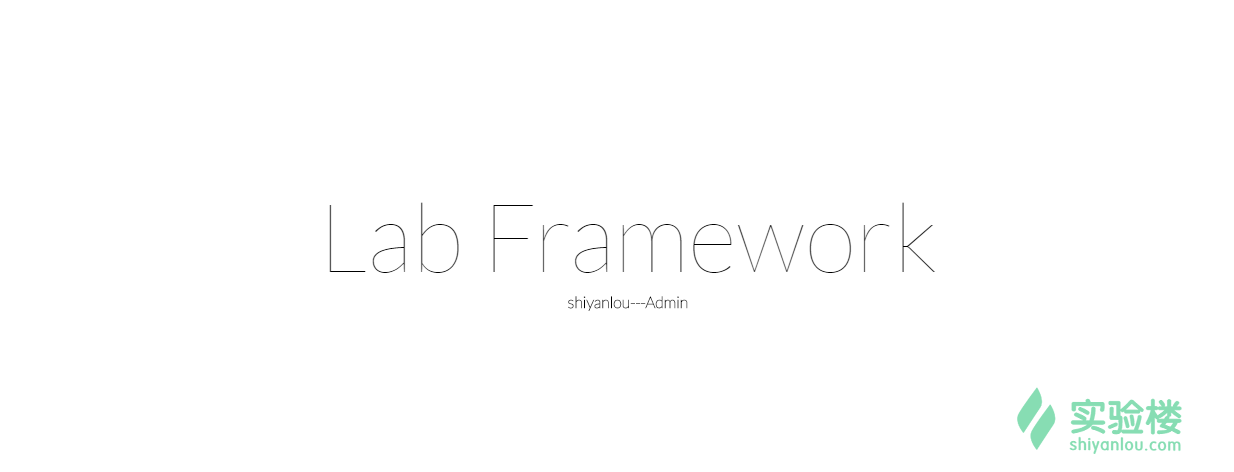
开启服务器：

$ php -S localhost:8080

开启数据库服务：

$ sudo service mysql start

首页效果：



## **二、运行测试**

经历了漫长的准备和编写过程，终于到了检验成果的时候。不用感叹，直接测试！

### **2.1 测试控制器和视图**

在 app/home/controller/ 下新建一个 IndexController.php 作为首页控制器，一定要注意命名规范。

编辑并写下如下内容：

<?php namespace home\controller;

use core\Controller;/\*\*

\* index控制器

\*/class IndexController extends Controller{

public function index()

{

$this->assign('name','shiyanlou---Admin'); //模板变量赋值

$this->display(); //模板展示

}

}

相信大家都看得懂上面的代码，方法中用到了模板，所以我们需要在 view/ 下建立一个 index.php 的视图文件，对应于上面的 index 方法，得益于我们之前在启动文件中做的工作，当我们调用 display 方法时，不需要显式去指定该调用哪个模板，框架会默认到视图目录去寻找与该方法同名的模板文件。

在 index.php 中写下如下内容：

<!DOCTYPE html><html>

<head>

<title>Lab Framwork</title>

<link href="https://fonts.googleapis.com/css?family=Lato:100" rel="stylesheet" type="text/css">

<style>

html, body {

height: 100%;

}

body {

margin: 0;

padding: 0;

width: 100%;

display: table;

font-weight: 100;

font-family: 'Lato';

}

.container {

text-align: center;

display: table-cell;

vertical-align: middle;

}

.content {

text-align: center;

display: inline-block;

}

.title {

font-size: 96px;

}

.line {

color: black;

}

</style>

</head>

<body>

<div class="container">

<div class="content">

<div class="title">Lab Framework</div>

<span class="line"><b>{$name}</b></span>

</div>

</div>

</body></html>

看似很多，其实没什么重要的东西，你也可以随点写点 html 代码。上面的代码中，我写了一句：{$name} ，如果你还记得，这是这是我们约定的模板语法，也是模板解析类主要解析的内容，待会儿这部分将会展示控制器中 index 方法中赋值的模板变量的值。

现在打开浏览器，输入 localhost:8080 ，应该就会出现上面展示的图片效果。而且在 /rumtime/compile/ 下也存在一个名字很长的文件。查看其中的内容，你会发现和模板的内容不同：

<div class="container">

<div class="content">

<div class="title">Lab Framework</div>

<span class="line"><b><?php echo $this->vars['name']; ?></b></span>

</div>

</div>

这就是经过模板解析类编译生成的编译文件，已将将模板变量解析为正确的输出方式。

现在到配置文件中把自动缓存：auto\_cache 配置为 true。再次刷新页面：没有任何变化。不过现在你可以发现在 /runtime/cache/ 下多了一个名为 cache\_index.html 的 html 文件，没错，这就是缓存的页面文件，打开并查看其中的内容，你会发现内容又变了：

<div class="container">

<div class="content">

<div class="title">Lab Framework</div>

<span class="line"><b>shiyanlou---Admin</b></span>

</div>

</div>

这就是最普通并且可以直接展示的 html 代码！当我们下次再访问这个页面地址时，如果开启了自动缓存，且此缓存文件存在，且模板文件和编译文件都存在并且都没有修改过，那么框架就会自动调用这个缓存文件，而不需要在经过编译、渲染的步骤，大大提高了网页的加载速度。

### **2.2 测试模型**

在控制器文件中再建立一个用户控制器：UserController.php，在其中添加 index 方法，并写下：

<?php namespace home\controller;

use core\Controller;use home\model\UserModel;/\*\*

\* yong

\*/class UserController extends Controller{

public function index()

{

$model = new UserModel();

if ($model->save(['name'=>'hello','password'=>'shiyanlou'])) {

$model->free(); //释放连接

echo "Success";

} else {

$model->free(); //释放连接

echo 'Failed';

}

}

}

由于这是用户控制器，所以应该建立一个用户模型，上面的代码中也明确了要使用 home/model/ 目录下的 UserModel，所以在 model/ 目录下新建一个用户模型：UserModel.php ，并且编辑如下：

<?php namespace home\model;

use core\Model;/\*\*

\* UserModel

\*/class UserModel extends Model{

function \_\_construct()

{

parent::\_\_construct('user');

}

}

这里只是为了起到示范作用，所以只写了构造方法，大家也可以向其中添加更多的自定义方法。

在UserModel 的构造方法中，我们需要显式向模型基类传递数据表名，明确此模型需要和哪一张表交互，Model 类会根据传入的表名自动添加表前缀，形成完整的表名。这里的 user 将会对应数据库里的 frw\_user 表。

在浏览器输入地址：localhost:8080/index.php/home/user/index.html，应该会显示 ‘Success’ 。其他模型操作方法，就留给你们自己去测试。

### **2.4 添加跳转方法**

上面的控制器方法中，当模型操作失败的时候，应该给出提示并跳转页面，执行成功也应该执行跳转操作。所以为了实现这个功能，我们可以添加一个页面跳转的方法。框架的跳转功能应该作为核心且通用的功能来实现，所以我们应该将他放到核心类文件中。不过这里的实现方式要和普通类不同，我们可以使用 [Trait](http://www.php.net/manual/zh/language.oop5.traits.php" \t "https://www.shiyanlou.com/courses/607/labs/2035/_blank) 来实现。

Trait：是为类似 PHP 的单继承语言而准备的一种代码复用机制。Trait 为了减少单继承语言的限制，使开发人员能够自由地在不同层次结构内独立的类中复用 method。Trait 和 Class 组合的语义定义了一种减少复杂性的方式，避免传统多继承和 Mixin 类相关典型问题。

Trait 和 Class 相似，但仅仅旨在用细粒度和一致的方式来组合功能。 无法通过 trait 自身来实例化。它为传统继承增加了水平特性的组合；也就是说，应用的几个 Class 之间不需要继承。

回到核心类文件目录，在 core/ 下新建一个 traits 的文件夹，主要用于存放类似跳转这种 trait 文件，在其下新建一个 Jump.php 。编辑如下：

<?php namespace core\traits;

trait Jump

{

public static function success($msg = '',$url = '',$data = '')

{

$code = 1;

if (is\_numeric($msg)) {

$code = $msg;

$msg = '';

}

if (is\_null($url) && isset($\_SERVER["HTTP\_REFERER"])) {

$url = $\_SERVER["HTTP\_REFERER"];

}

$result = [

'code' => $code, //状态码

'msg' => $msg, //显示信息

'data' => $data, //输出数据

'url' => $url, //跳转url

];

$output = 'code:'.$result['code'].'\n'.'msg:'.$result['msg'].'\n'.'data:'.$result['data'];

echo "<script> alert('$output');location.href='".$result['url']."'</script>";

exit();

}

public static function error($msg = '',$url = '',$data = '')

{

$code = 0;

if (is\_numeric($msg)) {

$code = $msg;

$msg = '';

}

if (is\_null($url) && isset($\_SERVER['HTTP\_REFERER'])) {

$url = $\_SERVER['HTTP\_REFERER'];

}

$result = [

'code' => $code,

'msg' => $msg,

'data' => $data,

'url' => $url,

];

$output = 'code:'.$result['code'].'\n'.'msg:'.$result['msg'].'\n'.'data:'.$result['data'];

echo "<script> alert('$output');location.href='".$result['url']."'</script>";

exit();

}

}

上面的代码实现了一个非常简单的跳转功能（虽然样式不美观）。成功的操作跳转需要指定跳转地址，如不指定，将会回退到前一页。失败的跳转将跳转至前一页。都可以指定提示信息和必要的数据。

由于主要的逻辑操作在控制器中进行，所以跳转功能也主要在控制器中执行。我们直接在控制器基类中添加跳转功能，子类就可以直接使用。

修改 Controller.php :

use core\View;use core\traits\Jump; //新增语句

\*\*\*

use Jump; //新增语句

protected $vars = [];

protected $tpl;

接下来就可以直接在子类控制器中调用 success 和 error 方法。刚才的用户控制器的提示方式可以改为：

if ($model->save(['name'=>'hello','password'=>'shiyanlou'])) {

$this->success('Success','/'); //执行成功，弹出信息，跳转至首页

} else {

$this->error('error'); //如果你在这个页面尝试，将会一直弹出错误信息.因为找不到上一页

}

效果：





上面的信息提示确实不太美观，不过我们的目的不在于多美观。只要能实现功能就行！

## **三、总结**

到目前为止，我们使用 PHP 搭建的简单 MVC 框架就制作完毕了。实现了运行的基本流程，完成模板页面的解析与展示，路由处理，逻辑处理，以及与数据库的交互。当然，我们制作的框架还有很多不完善和不足的地方，有兴趣研究的同学可以继续深入了解框架的知识。本次课程所制作的框架仅用于学习的作用，目的是希望大家能从实际的代码编写中提升 PHP 技术，对 PHP 框架的实现原理有个一大致的了解。为以后学习其他框架打好基础。因为本框架仅用于练习的目的，所以代码质量可能不是最优的，实现方式可能不是最好的。如果文档中有误的地方，希望能够指出，也希望大家有什么意见或建议到课程评论中给我留言。

## **四、源码下载**

wget http://labfile.oss.aliyuncs.com/courses/607/Labframe.tar