# **框架搭建**

## **一、实验简介**

### **1.1 实验目的**

​ 本次实验将带领大家快速搭建框架结构，分析文件和目录作用。做一些框架运行的前期准备工作。

## **1.2 开发工具**

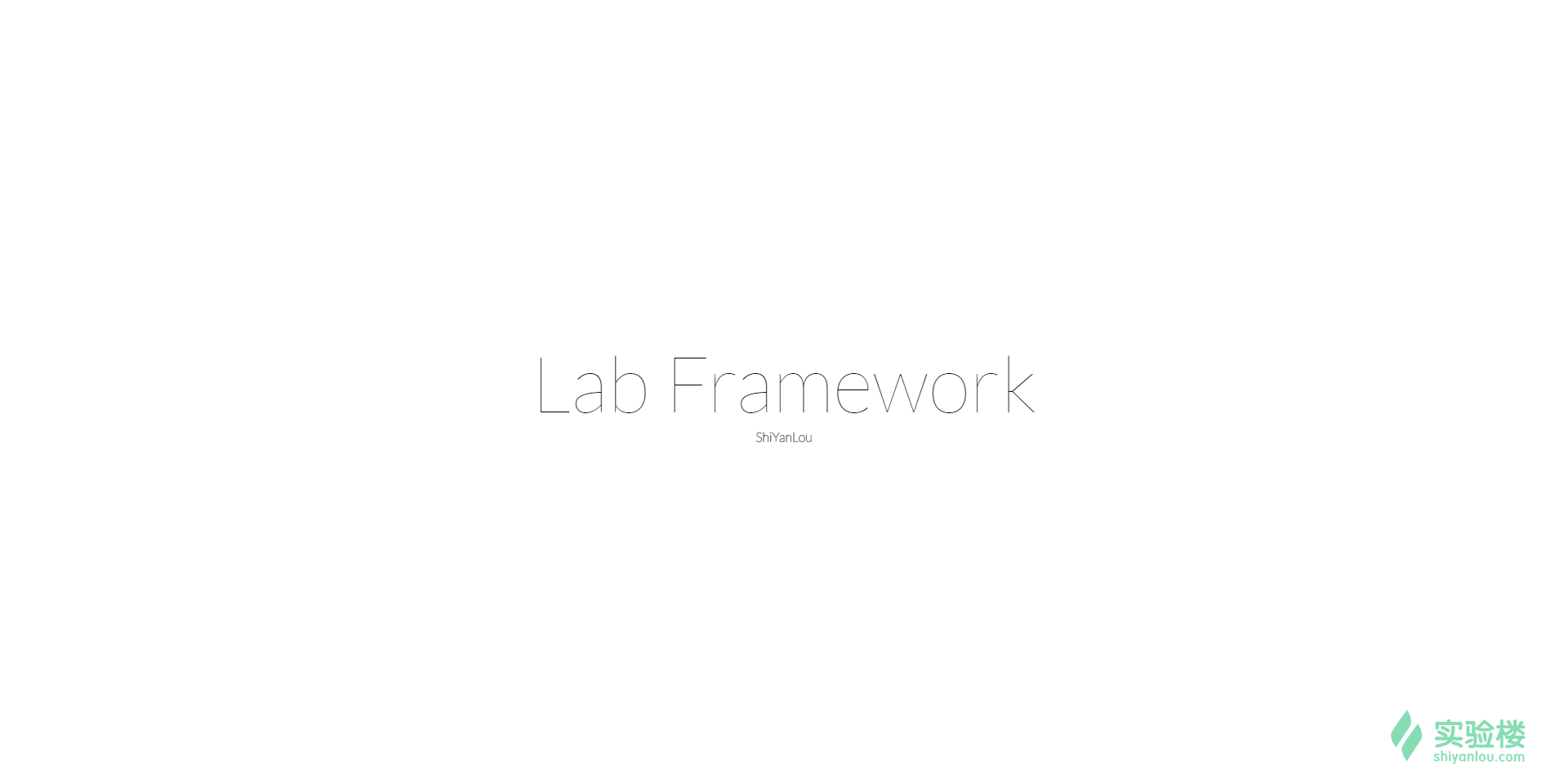
* sublime，一个方便快速的文本编辑器。点击桌面左下角： 应用程序菜单/开发/sublime。

### **1.3 最终效果**

开启内置服务器：

$ php -S localhost:8080

框架启动效果：



### **1.4 参考框架和项目**

* Laravel5.2
* ThinkPHP5
* [tiny-php-framework](https://github.com/yuansir/tiny-php-framework" \t "https://www.shiyanlou.com/courses/607/labs/2021/_blank)

## **二、框架搭建**

### **2.1 编码规范**

本项目遵循 [PSR-2](https://github.com/php-fig/fig-standards/blob/master/accepted/PSR-2-coding-style-guide.md" \t "https://www.shiyanlou.com/courses/607/labs/2021/_blank) 命名规范和 [PSR-4](https://github.com/php-fig/fig-standards/blob/master/accepted/PSR-4-autoloader.md" \t "https://www.shiyanlou.com/courses/607/labs/2021/_blank) 自动加载规范，并且注意如下规范：

#### **目录和文件**

* 目录不强制规范，驼峰及小写+下划线模式均支持；
* 类库、函数文件统一以.php为后缀；
* 类的文件名均以命名空间定义，并且命名空间的路径和类库文件所在路径一致；
* 类名和类文件名保持一致，统一采用驼峰法命名（首字母大写）；

#### **函数和类、属性命名**

* 函数的命名使用小写字母和下划线（小写字母开头）的方式，例如 get\_client\_ip；
* 方法的命名使用驼峰法，并且首字母小写，例如 getUserName；
* 属性的命名使用驼峰法，并且首字母小写，例如 tableName、instance；

#### **常量和配置**

* 常量以大写字母和下划线命名，例如 APP\_PATH和 CORE\_PATH；
* 配置参数以小写字母和下划线命名，例如 url\_route\_on 和url\_convert

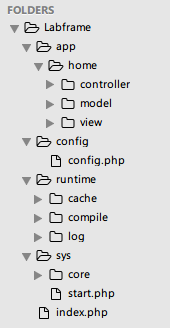
#### **数据表和字段**

* 数据表和字段采用小写加下划线方式命名，并注意字段名不要以下划线开头，例如 lab\_user 表和 lab\_name字段，不建议使用驼峰和中文作为数据表字段命名。

### **2.2 目录结构**

在 /home/shiyanlou/Code 下建立项目的目录 Labframe，我们开发的这个框架就叫 Labframe ，你也可以给你的框架取一个好听的名字。

接下来开始搭建框架的主要目录，目录结构如下图：



目录讲解：

app/：应用程序目录。用户在其中进行功能开发

home/：模块目录。一般分为前台（home）和后台模块（admin），这里只建立的前台模块

* + - controller/：前台控制器目录，存放控制器文件。主要处理前台模块的操作
    - model/：前台模型目录，存放模型文件。处理前台模型的相关操作
    - view/：前台视图目录，存放视图文件。前台展示的模板文件。

config/：配置文件目录

* + config.php：框架的配置文件

runtime/：运行时目录，保存框架运行时产生的数据。

* + cache/：缓存目录。用于存放缓存的模板文件
  + complie/：编译目录。用于存放经过编译的模板文件
  + log/：日志文件。用于记录框架运行期间的行为

sys/：框架目录。用于存放框架文件

* + core/：框架核心目录。存放框架运行所需的核心文件
  + start.php：框架启动文件。

index.php：框架入口文件。所有请求都经过此文件处理

目录中有一点需要再讲一下：index.php。这是整个框架的入口文件，叫做单一入口文件。这里涉及到一个知识点：单一入口模式和多入口模式。

* 单一入口模式：单一入口通常是指一个项目或者应用具有一个统一（但并不一定是唯一）的入口文件，也就是说项目的所有功能操作都是通过这个入口文件进行的，并且往往入口文件是第一步被执行的。
* 多入口模式：多入口即是通过不同的入口文件访问后台。比如常用的多入口：index.php（前台入口），admin.php（后台入口）

我们的框架采用单入口机制。

### **2.3 全局配置**

由于我们的框架规模比较小，所以我们可以只需要一个配置文件，不区分前后台，作为全局配置。

编辑 config.php ，基础配置如下（可根据情况自行修改）：

<?php

return [

//数据库相关配置

'db\_host' => '127.0.0.1',

'db\_user' => 'root',

'db\_pwd' => '',

'db\_name' => 'labframe',

'db\_table\_prefix' => 'lab\_', //数据表前缀

'db\_charset' => 'utf8',

'default\_module' => 'home', //默认模块

'default\_controller' => 'Index', //默认控制器

'default\_action' => 'index', //默认操作方法

'url\_type' => 2, // RUL模式：【1：普通模式，采用传统的 url 参数模式】【2：PATHINFO 模式，也是默认模式】

'cache\_path' => RUNTIME\_PATH . 'cache' .DS, //缓存存放路径

'cache\_prefix' => 'cache\_', //缓存文件前缀

'cache\_type' => 'file', //缓存类型（只实现 file 类型）

'compile\_path' => RUNTIME\_PATH . 'compile' .DS, //编译文件存放路径

'view\_path' => APP\_PATH .'home' . DS . 'view' . DS, // 模板路径

'view\_suffix' => '.php', // 模板后缀

'auto\_cache' => true, //开启自动缓存

'url\_html\_suffix' => 'html', // URL伪静态后缀

];

上面用到了一些暂时尙未定义常量，我们将会在后面的文件中定义：

RUNTIME\_PATH：运行时目录路径

DS：目录分隔符。在win下为 '\',在 linux 下为 '/'

APP\_PATH：应用程序目录路径

### **2.4 数据库准备**

上面的配置文件中配置了一些连接数据库的相关参数，所以这里我们可以先建立好相关的数据库和数据表。

开启数据库服务：

$ sudo service mysql start

进入MySQL:

$ mysql -u root

键入以下 sql 语句新建数据库：

CREATE DATABASE IF NOT EXISTS labframe;

新建数据表（以 user 表作为示例）：

> USE labframe;

> CREATE TABLE lab\_user(

`id` INT(10) NOT NULL PRIMARY KEY AUTO\_INCREMENT,

`name` varchar(30) NOT NULL,

`password` varchar(32) NOT NULL

);

向其中插入一条测试数据：

> INSERT INTO lab\_user (name,password) VALUES ('admin','shiyanlou');

数据库准备完毕。

### **2.5 入口文件**

上面提到过的 index.php 就是框架的入口文件。接下来可以向里面写点东西了。

<?php //框架入口文件

define('DS', DIRECTORY\_SEPARATOR); //定义目录分隔符（上面用到过）

define('ROOT\_PATH', \_\_DIR\_\_ . DS); //定义框架根目录require 'sys/start.php'; //引入框架启动文件

core\App::run(); //框架启动

就这么简单几句就行了。

### **2.6 框架启动文件**

编辑 start.php 文件：

<?php //框架启动文件

define('APP\_PATH', ROOT\_PATH . 'app' . DS); //定义应用程序目录路径

define('RUNTIME\_PATH', ROOT\_PATH . 'runtime' . DS); //定义框架运行时目录路径

define('CONF\_PATH', ROOT\_PATH . 'config' . DS); //定义全局配置目录路径

define('CORE\_PATH', ROOT\_PATH . 'sys' .DS . 'core' . DS); //定义框架核心目录路径

//引入自动加载文件require CORE\_PATH.'Loader.php';

//实例化自动加载类

$loader = new core\Loader();

$loader->addNamespace('core',ROOT\_PATH . 'sys' .DS . 'core'); //添加命名空间对应base目录

$loader->addNamespace('home',APP\_PATH . 'home');

$loader->register(); //注册命名空间

//加载全局配置

\core\Config::set(include CONF\_PATH . 'config.php');

在上面的框架启动文件中，我们定义了一些必要的路径常量，引入了自动加载类文件，并且添加并注册了两个主要的命名空间。最后将我们的全局配置文件载入框架。

### **2.7 自动加载类**

相信很多人都遇到过这个问题，当一个文件需要使用其他文件的其他类的时候，往往需要使用很多的 require 或 include 来引入这些类文件，当需要的类特别多的时候，这样做就显得很笨拙，不仅使文件变的混乱不堪，各种引入关系也相当复杂。所以在大型项目中，往往采用按需加载，自动加载的方式来实现类的加载。本框架自动加载的实现由 core\Loader 类库完成，自动加载规范符合PHP的 PSR-4。

Labframe 采用了命名空间的特性，因此只需要给类库正确定义所在的命名空间，而命名空间的路径与类库文件的目录一致，那么就可以实现类的自动加载。

core/ 是框架运行的核心目录，我们的自动加载类应该放在这里。在 core/ 下新建一个加载类文件：Loader.php。编辑如下：

<?php namespace core;

class Loader{

/\*\*

\* An associative array where the key is a namespace prefix and the value

\* is an array of base directories for classes in that namespace.

\*

\* @var array

\*/

protected static $prefixes = [];

/\*\*

\* 在 SPL 自动加载器栈中注册加载器

\*

\* @return void

\*/

public static function register()

{

spl\_autoload\_register('core\\Loader::loadClass');

}

/\*\*

\* 添加命名空间前缀与文件base目录对

\*

\* @param string $prefix 命名空间前缀

\* @param string $base\_dir 命名空间中类文件的基目录

\* @param bool $prepend 为 True 时，将基目录插到最前，这将让其作为第一个被搜索到，否则插到将最后。

\* @return void

\*/

public static function addNamespace($prefix, $base\_dir, $prepend = false)

{

// 规范化命名空间前缀

$prefix = trim($prefix, '\\') . '\\';

// 规范化文件基目录

$base\_dir = rtrim($base\_dir, '/') . DIRECTORY\_SEPARATOR;

$base\_dir = rtrim($base\_dir, DIRECTORY\_SEPARATOR) . '/';

// 初始化命名空间前缀数组

if (isset(self::$prefixes[$prefix]) === false) {

self::$prefixes[$prefix] = [];

}

// 将命名空间前缀与文件基目录对插入保存数组

if ($prepend) {

array\_unshift(self::$prefixes[$prefix], $base\_dir);

} else {

array\_push(self::$prefixes[$prefix], $base\_dir);

}

}

/\*\*

\* 由类名载入相应类文件

\*

\* @param string $class 完整的类名

\* @return mixed 成功载入则返回载入的文件名，否则返回布尔 false

\*/

public static function loadClass($class)

{

// 当前命名空间前缀

$prefix = $class;

// 从后面开始遍历完全合格类名中的命名空间名称, 来查找映射的文件名

while (false !== $pos = strrpos($prefix, '\\')) {

// 保留命名空间前缀中尾部的分隔符

$prefix = substr($class, 0, $pos + 1);

// 剩余的就是相对类名称

$relative\_class = substr($class, $pos + 1);

// 利用命名空间前缀和相对类名来加载映射文件

$mapped\_file = self::loadMappedFile($prefix, $relative\_class);

if ($mapped\_file) {

return $mapped\_file;

}

// 删除命名空间前缀尾部的分隔符，以便用于下一次strrpos()迭代

$prefix = rtrim($prefix, '\\');

}

// 找不到相应文件

return false;

}

/\*\*

\* 根据命名空间前缀和相对类来加载映射文件

\*

\* @param string $prefix The namespace prefix.

\* @param string $relative\_class The relative class name.

\* @return mixed Boolean false if no mapped file can be loaded, or the

\* name of the mapped file that was loaded.

\*/

protected static function loadMappedFile($prefix, $relative\_class)

{

//命名空间前缀中有base目录吗？?

if (isset(self::$prefixes[$prefix]) === false) {

return false;

}

// 遍历命名空间前缀的base目录

foreach (self::$prefixes[$prefix] as $base\_dir) {

// 用base目录替代命名空间前缀,

// 在相对类名中用目录分隔符'/'来替换命名空间分隔符'\',

// 并在后面追加.php组成$file的绝对路径

$file = $base\_dir

. str\_replace('\\', DIRECTORY\_SEPARATOR, $relative\_class)

. '.php';

$file = $base\_dir

. str\_replace('\\', '/', $relative\_class)

. '.php';

// 当文件存在时，载入之

if (self::requireFile($file)) {

// 完成载入

return $file;

}

}

// 找不到相应文件

return false;

}

/\*\*

\* 当文件存在，则从文件系统载入之

\*

\* @param string $file 需要载入的文件

\* @return bool 当文件存在则为 True，否则为 false

\*/

protected static function requireFile($file)

{

if (file\_exists($file)) {

require $file;

return true;

}

return false;

}

}

以上 Loader.php 便可实现类的按需自动加载。可能理解起来比较困难，有兴趣的同学可以多花点时间理解一下 Loader 类的实现方法。文档中也不太好详细解释，更多详情请参阅 [PSR-4-autoloade](https://github.com/php-fig/fig-standards/blob/master/accepted/PSR-4-autoloader-examples.md" \t "https://www.shiyanlou.com/courses/607/labs/2021/_blank)。上面的代码主要掌握两个重点：[命名空间](http://php.net/manual/zh/language.namespaces.rationale.php" \t "https://www.shiyanlou.com/courses/607/labs/2021/_blank)和[自动加载](http://php.net/manual/zh/function.spl-autoload-register.php" \t "https://www.shiyanlou.com/courses/607/labs/2021/_blank)。如果暂时理解起来比较困难或不感兴趣的同学可以先不用管这些，就仿照上面写，知道如何使用就行。

还记得 start.php 中的添加命名空间的方法么，我们添加了两个根命名空间以及其对应的base目录：core->/sys/core；home->/app/home，以后当我们在两个base目录下编写类的时候，只要明确类文件与base目录的相对路径，赋予其正确的命名空间， Loader 便可以正确的加载他们。

## **三、总结**

本次实验我们大致了解了 Labframe 框架的目录结构，明确了框架开发的代码规范。还遇到了一个重点：****类的自动加载****，这在框架开发中非常重要，不过理解起来也比较困难。这些都是框架开发的准备工作。在接下来的实验中，我们将会具体开发框架的核心文件。