# **框架核心类（一）**

## **一、实验简介**

### **1.1 实验目的**

从本次实验开始，我们将会逐步开始框架核心类的编写，由于核心类数量较多，所以将分为两次实验为大家讲解。希望大家能从实践中得到提高。

## **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 任务清单**

* App.php：框架启动类
* Config.php：配置类
* Router.php：路由类

## **二、核心类编写**

### **2.1 框架启动类**

在 sys/core/ 下新建一个 App.php 文件。先编辑如下：

<?php namespace core; //定义命名空间

use core\Config; //使用配置类use core\Router; //使用路由类/\*\*

\* 框架启动类

\*/class App{

//启动

public static function run()

{

}

//路由分发

public static function dispatch($url\_array = [])

{

}

}

启动类主要包含了两个核心的方法：run（启动），dispatch（路由分发）。

启动

执行框架的运行流程。首先需要分析路由，然后分发路由。

public static $router; //定义一个静态路由实例

public static function run()

{

self::$router = new Router(); //实例化路由类

self::$router->setUrlType(Config::get('url\_type')); //读取配置并设置路由类型

$url\_array = self::$router->getUrlArray(); //获取经过路由类处理生成的路由数组

self::dispatch($url\_array); //根据路由数组分发路由

}

举一个简单的例子，由于我们默认的路由配置模式为 2：pathinfo 模式，所以当你在地址栏输入：localhost:8080/home/index/index.html 时，经过路由类处理之后，就会得到一个路由数组，形如：['module'=>'home','controller'=>'index','action'=>'index']，然后执行路由分发操作，将会执行home模块下的index控制器下的index方法，这样就完成了路由的访问。

路由分发

将会根据路由数组分发到具体的模块、控制器和方法。

public static function dispatch($url\_array = [])

{

$module = '';

$controller = '';

$action= '';

if (isset($url\_array['module'])) { //若路由中存在 module，则设置当前模块

$module = $url\_array['module'];

} else {

$module = Config::get('default\_module'); //不存在，则设置默认的模块（home）

}

if (isset($url\_array['controller'])) { //若路由中存在 controller，则设置当前控制器，首字母大写

$controller = ucfirst($url\_array['controller']);

} else {

$controller = ucfirst(Config::get('default\_controller')); //不存在，则设置默认的控制器（index）,首字母大写

}

//拼接控制器文件路径

$controller\_file = APP\_PATH . $module . DS . 'controller' .DS . $controller . 'Controller.php';

if (isset($url\_array['action'])) { //同上，设置操作方法

$action = $url\_array['action'];

} else {

$action = Config::get('default\_action');

}

//判断控制器文件是否存在

if (file\_exists($controller\_file)) {

require $controller\_file; //引入该控制器

$className = 'module\controller\IndexController'; //命名空间字符串示例

$className = str\_replace('module',$module,$className); //使用字符串替换功能，替换对应的模块名和控制器名

$className = str\_replace('IndexController',$controller.'Controller',$className);

$controller = new $className; //实例化具体的控制器

//判断访问的方法是否存在

if (method\_exists($controller,$action)) {

$controller->setTpl($action); //设置方法对应的视图模板

$controller->$action(); //执行该方法

} else {

die('The method does not exist');

}

} else {

die('The controller does not exist');

}

}

因为我们的类的命名规范采用驼峰法，且首字母大写，所以在分发控制器的时候需要确保 url 中的控制器名首字母大写。同在在 Windows 下面是不需要区分大小写的，但是在 Linux 环境下却要格外注意。

上面代码的主要流程：定位模块 --> 定位控制器 ---> 定位方法（同时设置对应的模板）。这样就完成了路由分发的功能。

### **2.2 配置类**

在 Loader 中我们使用到了 Config 来读取各种配置值，所以我们马上来实现 Config 来完成对应的功能。 在 core/ 下新建一个 Config.php 文件，这就是配置类。主要结构如下：

<?php namespace core;

/\*\*

\* 配置类

\*/class Config{

private static $config = []; //存放配置

//读取配置

public static function get($name = null)

{

}

//动态设置配置

public static function set($name,$value = null)

{

}

//判断是否存在配置

public static function has($name)

{

}

//加载其他配置文件

public static function load($file)

{

}

}

核心方法就只有上面四个。

读取配置值

public static function get($name = null)

{

if (empty($name)) {

return self::$config;

}

//若存在配置项，则返回配置值。否则返回 null

return isset(self::$config[strtolower($name)]) ? self::$config[strtolower($name)] : null;

}

动态设置配置项

public static function set($name,$value = null)

{

if (is\_string($name)) { //字符串，直接设置

self::$config[strtolower($name)] = $value;

} elseif (is\_array($name)) { //数组，循环设置

if (!empty($value)) {

self::$config[$value] = isset(self::$config[$value]) ? array\_merge(self::$config[$value],$name) : self::$config[$value] = $name;

} else {

return self::$config = array\_merge(self::$config,array\_change\_key\_case($name));

}

} else { //配置方式错误，返回当前全部配置

return self::$config;

}

}

上面的代码可以根据传入的参数类型来对应不同的配置方法。

* + 只有 name（string）：name => null；
  + name（string） 和 value：name => value；
  + name（array）：array\_merge
  + name（array）和 value：array\_merge 或者 value => $name (二级配置)

是否存在配置

public static function has($name)

{

return isset(self::$config[strtolower($name)]);

}

加载其他配置文件

public static function load($file)

{

if (is\_file($file)) {

$type = pathinfo($file,PATHINFO\_EXTENSION);

if ($type != 'php') {

return self::$config;

} else {

return self::set(include $file);

}

} else {

return self::$config;

}

}

我们默认只是用 config/config.php 作为唯一的全局配置文件。但是如果你想为某个模块做单独的配置，或者需要覆盖默认的配置项，那么你就可以使用这个方法来实现 。

这样，我们的配置类就完成了，可以任意的读取、设置、加载配置项。

### **2.3 路由类**

​ 主要负责处理路由信息，将路由地址处理为路由数组，供启动类分发。当然，你也可以把启动类里的路由分发功能放到路由类中实现。在core/ 下新建一个 Router.php 文件，作为路由类。主要结构如下：

<?php namespace core;

/\*\*

\* 路由类

\*/class Router{

public $url\_query; //URL 串

public $url\_type; //UTL 模式

public $route\_url =[]; //URL数组

function \_\_construct()

{

}

//设置 URL 模式

public function setUrlType($url\_type = 2)

{

}

//获取URL数组

public function getUrlArray()

{

}

//处理 URL

public function makeUrl()

{

}

//将参数形式转为数组

public function queryToArray()

{

}

//将 pathinfo 转为数组

public function pathinfoToArray()

{

}

构造方法

function \_\_construct()

{

$this->url\_query = parse\_url($\_SERVER['REQUEST\_URI']);

}

使用 $\_SERVER['REQUEST\_URI']是取得当前URL的 路径地址。再使用 [parse\_url](http://php.net/manual/zh/function.parse-url.php" \t "https://www.shiyanlou.com/courses/607/labs/2032/_blank) 解析 url：主要分为路径信息 [path] 和 参数信息 [query] 两部分。

设置 URL 模式

public function setUrlType($url\_type = 2)

{

if ($url\_type > 0 && $url\_type < 3) {

$this->url\_type = $url\_type;

}else{

exit('Specifies the URL does not exist!');

}

}

默认为 2 => pathinfo 模式。

获取经过处理的 URL数组

public function getUrlArray(){

$this->makeUrl();

return $this->route\_url;

}

处理 URL

public function makeUrl()

{

switch ($this->url\_type) {

case 1:

$this->queryToArray();

break;

case 2:

$this->pathinfoToArray();

break;

}

}

根据 url 模式的不同选择不同的方式构造 url 数组。

将参数形模式转为数组

// ?xx=xx&xx=xxpublic function queryToArray()

{

$arr = !empty($this->url\_query['query']) ? explode('&',$this->url\_query['query']) : [];

$array = $tmp = [];

if (count($arr) > 0) {

foreach ($arr as $item) {

$tmp = explode('=',$item);

$array[$tmp[0]] = $tmp[1];

}

if (isset($array['module'])) {

$this->route\_url['module'] = $array['module'];

unset($array['module']);

}

if (isset($array['controller'])) {

$this->route\_url['controller'] = $array['controller'];

unset($array['controller']);

}

if (isset($array['action'])) {

$this->route\_url['action'] = $array['action'];

unset($array['action']);

}

if (isset($this->route\_url['action']) && strpos($this->route\_url['action'],'.')) {

//判断url方法名后缀 形如 'index.html',前提必须要在地址中以 localhost:8080/index.php 开始

if (explode('.',$this->route\_url['action'])[1] != Config::get('url\_html\_suffix')) {

exit('suffix errror');

} else {

$this->route\_url['action'] = explode('.',$this->route\_url['action'])[0];

}

}

} else {

$this->route\_url = [];

}

}

将 pathinfo 转为数组

// xxx/xxx/xx public function pathinfoToArray()

{

$arr = !empty($this->url\_query['path']) ? explode('/',$this->url\_query['path']) : [];

if (count($arr) > 0) {

if ($arr[1] == 'index.php') { //以 'localhost:8080/index.php'开始

if (isset($arr[2]) && !empty($arr[2])) {

$this->route\_url['module'] = $arr[2];

}

if (isset($arr[3]) && !empty($arr[3])) {

$this->route\_url['controller'] = $arr[3];

}

if (isset($arr[4]) && !empty($arr[4])) {

$this->route\_url['action'] = $arr[4];

}

//判断url后缀名

if (isset($this->route\_url['action']) && strpos($this->route\_url['action'],'.')) {

if (explode('.',$this->route\_url['action'])[1] != Config::get('url\_html\_suffix')) {

exit('Incorrect URL suffix');

} else {

$this->route\_url['action'] = explode('.',$this->route\_url['action'])[0];

}

}

} else { //直接以 'localhost:8080'开始

if (isset($arr[1]) && !empty($arr[1])) {

$this->route\_url['module'] = $arr[1];

}

if (isset($arr[2]) && !empty($arr[2])) {

$this->route\_url['controller'] = $arr[2];

}

if (isset($arr[3]) && !empty($arr[3])) {

$this->route\_url['action'] = $arr[3];

}

}

} else {

$this->route\_url = [];

}

}

若服务器开启了rewrite 模块，可以隐藏 index.php。在本课程中，若要添加url后缀名，则必须以 'localhost:8080/index.php' 开头。若以 'localhost:8080' 开头，则末尾不能添加 '.html' 或 '.php' 等后缀名。

## **三、总结**

本次实验大家亲手编写了三个框架核心类。主要完成框架的启动，配置文件的加载与路由的处理。对框架的运行流程有了一定的了解。限于篇幅，我们将会在下一个实验中继续编写剩下的核心类，主要包括：视图，模型，控制器，模板解析等，继续完善我们的框架。