实现一个简易的PHP框架

发表于2017-01-08 | 分类于技术原理

月份的时候，参照慕客网的视频，照猫画虎折腾了一个非常简陋的PHP框架半成品，简陋到只勉强实现了自动加载类和路由功能，由于功力暂时将这个小项目放在了旁边，到后面补习了一些PHP的基础知识，又花了大概一周的时间去了解Laravel，对于PHP框架的运行流程稍微长一点。现在趁着周末重新整理了之前的笔记：一个简易的PHP MVC框架的实现。

more-->

MVC框架

概念

的PHP框架都是采用MVC形式，就我现在对于MVC的理解是这种设计将整个项目分为了三层：

M(模型层),主要负责与数据路交互并提供获取相应数据的接口
V(视图层),主要负责数据的提交和显示,与用户进行交互

- C(控制层),主要处理业务逻辑，连接模型层与视图层。

简单地说，控制层就是获取用户的请求，并将对应的请求转发到模型层，获取模型层返回的数据，然后再提供给视图层展示。而实际上，一个PHP程序也可以抽象地划分为3个组成部分，然后在MVC结构中：

- 数据的采集(从视图层提交数据)
- 数据的处理(控制层捕获到数据并转交给模型层进行逻辑处理，然后从模型层返回经过处理的数据到控制层)
- 数据的输出(控制层将模型返回的数据传递给视图)

目录

单入口文件

C相关还有另外一个比较重要的概念：**单入口文件**。比如，ThinkPHP根目录下的 `index.php` 和Laravel根目录下的 `ic/index.php` ，就是整个框架的入口文件。 单入口文件的原理是：通过单入口文件加载整个项目的核心文件，核心文件中通过解析路 需要调用的控制器，然后加载对应的控制器并调用相关方法，控制器方法处理对应的逻辑并加载需要的视图文件，最后输出视图。 那么， 说，访问所有的控制器方法，实际上都是访问这个 `index.php` 文件(假设我们的入口文件名就是 `index.php` ，下同)，在默认情况，每 开头都会带有 `index.php` 。

情况下，都会通过 `.htaccess` 文件进行服务器路径重写，隐藏URL路径中的 `index.php` ，我们看到的URL一般就成了 `xxx.com/Index/index` 这样的形式。

简易框架的单入口文件中，我们需要做两件事情：

定义路径常量，整个框架主要包含**核心文件夹**（用于存放框架的核心文件，诸如路由，数据库，日志类以及相关的配置文件）和**项目文件夹**（用于存放项目代码，包括控制器和模型）。
注册自动调用方法

```
// 定义路径常量
define('ROOT',dirname(__FILE__));
define('CORE',ROOT.'/Core');
define('APP',ROOT.'/App');
define('MOUDLE','App');
```

```
// 调试模式
ini_set('display_errors','1');

// 加载函数库
ude CORE.'/Common/function.php';
加载核心文件
ude CORE.'/Core.php';

自动加载类
autoload_register('Core\Core::load');

启动程序
e\Core::run();
```

目录

路由

访问 index.php 是无法完成我们的业务逻辑的，我们必须通过解析URL的参数，调用对应的方法。这里使用的方法是PHP的超全局变量 $_SERVER 的属性 ['REQUEST_URI'] 来实现URL的解析的。

$_SERVER

$_SERVER 是一个包含了诸如头信息(header)、路径(path)、以及脚本位置(script locations)等等信息的数组。这个关联数组中的属性由 Web 服务器创建（但是不能保证每个服务器都提供全部属性）。 我们使用的是其中的'REQUEST_URI'属性，也就是 'URI' 用来指定要访问的页面。

2.2. URI

那么问题来了，'URI'和'URL'长的这么像，他们的关系是什么呢？首先来看一看什么是URI。参考网络爬虫;

> URI，是uniform resource identifier，统一资源标识符，用来唯一地标识一个资源，Web上每种可用的资源，如 HTML页面、图像等都
> 这个标识符进行定位。

一个URI由三部分组成，比如 `http://www.shy.com/Index/index` ：

访问资源的命名机制 `http`

存放资源的主机名 `www.shy.com`

可以通过路径访问 `Index/index`

需要解析的就是这个 `Index/index` ，这正是 `$_SERVER['REQUEST_URI']` 提供的值（所以前面隐藏 `index.php` 也是为了更轻松的获
URI）。

URL

就是常说的URL了。

> RL，是uniform resource locator，统一资源定位符，主要用在各种WWW客户程序和服务器程序上，采用URL可以用一种统一的格式来
> 述各种信息资源，包括文件、服务器的地址和目录等。

URL的格式如下：

```
protocol :// hostname[:port] / path / name
```

- 协议名
- 主机名，有时还包括端口号
- 资源保存的路径和名称

看URL跟URI貌似并没有什么区别嘛!实际上可以把URI看做URL更低层次的抽象，只一种字符串文本标准。两者的区别在于：

URI表示请求服务器的路径，定义这么一个资源（并没有指定它的用途）；

而URL同时说明要如何访问这个资源（指明网址 ftp服务器 文件路径）。

引申还之前看见的一个题目:"输入 www.shy.com 和 www.shy.com/ 的区别。咳咳，偏离文章主题了，就此打住。

小结

获取 $_SERVER['REQUEST_URI'] ，然后将他分解成所需要的控制器和方法名，再考虑实现GET参数的传递，一个简单的路由类就实现

```
space Core\Lib;
s Route {
public $ctrl = 'Index';
public $action = 'index';

public function __construct(){
    $urlArr = explode('/',trim($_SERVER['REQUEST_URI'],'/'));
    if (isset($urlArr[0]) && $urlArr[0] != ''){
        $this->ctrl = $urlArr[0];
    }
    if (isset($urlArr[1])){
        $this->action = $urlArr[1];
    }
```

```
        // 如果传参则必须显式声明控制器和方法名
        if (isset($urlArr[2])){
            $getArr = explode('&',$urlArr[2]);
            foreach($getArr as $v ){
                $pair = explode('=',$v);
                $_GET[$pair[0]] = $pair[1];
            }
        }
    }
```

输入的URL为： `http://www.shy.com/Index/index/id=100&age=200` ，则路由解析获得的结果是：

Index控制器

index方法

$_GET[id]=100,$_GET[age]=200

路由方式参照的是ThinkPHP的做法，然而我并没有去研究TP的源码，里面应该有更全面的处理方式。实际上我更倾向于Laravel的自定义路由，虽然只会一点点皮毛，哈哈。

自动加载类和命名空间

从路由中获取到了请求的控制器和方法名，然后只要加载相应的控制器文件，顺利调用目标方法，整个程序就能跑起来了。加载控制器类，需要用到两个语言基础：自动加载类和命名空间。这里从《Modern PHP》这本书中收获颇多。

3.1. 自动加载类

自动加载类指的是，PHP在运行时按需要查找类并加载到相关文件，从而不需要显式地提前声明 `require` 或 `include` 。在入口文件看见了 `spl_autoload_register` 这个函数

> ool spl_autoload_register ([callable $autoload_function [, bool $throw = true [, bool $prepend = false]]])，将函数注册到 PL__autoload 函数队列中。如果该队列中的函数尚未激活，则激活它们。

自定义一个注册函数，这个函数接收一个类名作为参数，当程序运行到需要实例一个对象却找不到对应的类的时候，就会自动调用 utoload ，通过在注册函数中定义的规则加载相应的类文件。

可以随心所欲的定义注册函数中的加载规则。实际上，PSR-4标准指定的自动加载策略依赖PHP命名空间和文件系统目录结构查找并加载类，即：将命名空间的前缀和文件系统中的目录对应起来（我们知道命名空间和实际的文件目录并没有直接关系）。通过将完全命名空间映射为文件目录，可以定义很直观的类加载规则。

PHP中的命名空间是怎么一回事呢？

命名空间

在C++中就知道了命名空间的概念，却没有明白它真正的意义。之后再Laravel的使用中体会到了命名空间的强大。有了命名空间，我们可以直观地组织和封装相关的PHP类（假装这里体会到了模块化思想），避免与第三方的类库发生命名冲突。

使用 `namespace` 关键字来定义命名空间，与目录和文件的关系相似（但是并没有直接的关联），PHP命名空间也允许指定层次化的命名空间的名称。因此，命名空间的名字可以使用分层次的方式定义（这正是实现自动加载类所需要的）。需要注意的是必须在文件的开头部分声明整个文

目录

件的命名空间（尽管一个文件可以同时声明数个命名空间，但是一般不推荐这么做）。 定义一个命名空间之后，其中声明的类和函数就有了名称限定，调用命名空间中的类或函数可以通过三种方式引用。

2.2.1. 非限定性名称

定性名称,或者不包含前缀的类名，会被默认解析在当前文件的命名空间下，如果当前文件没有命名空间，则会解析为全局函数名称（即不包含命名空间）或者常量名称。 以上面的代码为例，如果在file2中以非限定性名称调用:

```
space name2;
ude '1.php';
);
```

foo()解析为name2\foo()，如果在name2的命名空间下不存在这个函数就会报错；如果不声明name2，就会在将foo()解析为一个不在任何命名空间下的全局函数，当然，如果无法找到依旧会报错。

非限定性的名称就会出现一种情况：如果需要使用的类名或变量是全局的，而当前文件存在命名空间，如果不加以处理，会优先将其解析为命名空间下，出现错误。那么，如何解决这种问题呢？这时就可以使用完全限定名称。

2.2.2. 完全限定的名称

名空间前加上全局前缀操作符 \ ，就会从全局开始，依照命名空间的层级寻找对应的变量，实际上，只需要在文件使用namespce声明命名空间名称的前面在加上 \ 就可以了（因为声明命名空间必须显式地指定命名空间的层级）。

而前小节提出的问题现在也很容易解决：只需要在全局函数或类名全加上全局前缀操作符：

```
namespace name2;
include '1.php'; // 注意此时1.php中是不存在命名空间的
\foo();
```

. 局部限定名称

限定名称和非限定性名称的区别就在于：在函数或类名前使用了一个不是完全限定的命名空间（没有全局前缀操作符）。 此时，如果会将当
中的命名空间存在，则会将该命名空间名称添加到所使用的局部限定名称前 ；如果不存在，则会直接去寻找对应名称的命名空间下的类或函
限非限定性名称一样）。

文件路径的小问题

到命名空间的一个小细节： \ 反斜杠限定符。这个斜杠跟文件路径 / 十分相似！虽然已经写了很多次路径，也没有发现什么错误，但是决
做深究一下这个问题。

工，又称左斜杠，符号是 / ；反斜杠，也称右斜杠，符号是 \ 。

在Unix/Linux中，路径的分隔采用正斜杠 / ，比如 /home/index ；
在Windows中，路径分隔采用反斜杠 \ ，比如 C:\Windows\System 。

这么做的原因是：在Windows设计初期，正斜杆 / 作为DOS命令提示符的参数标志，而文件路径为了和Unix一些特征区别开，因此...(任
而常见的浏览器地址使用的是正斜杠，此外，网络文件路径也必须使用正斜杠。而在windows下的文件路径，使用反斜杠来表示。 然而我发现即使是在windows(win10)下，也可以使用正斜杠来访问到指定路径的正确文件。一种解释是Windows的资源管理器正确的处理了用户有可能产生的输入错误，因此就算是我们正反斜杠混合使用都是可以实现对于文件的定位。

由于反斜杠也是转义符标识，所以在书写路径的时候有可能出现解析错误的情况，因此有时候也会看到 `C:\\Windows\\System` 这样对反斜杠先进行转义的路径写法。

更新：在PHP中，预定于了一个关于目录分隔符的常量 `DIRECTORY_SEPARATOR` ，该变量会返回一个跟操作系统相关的路径分割符，这样就不用考虑使用正斜杠还是反斜杠的问题了。

小结

上面的基础，实现一个自动加载类就很简单了：需要加载的、带有命名空间的类名，传递给注册函数，在注册函数中将命名空间映射为文件，并加载定义该类的文件，大功告成。此外，如果考虑路径使用反斜杠，则最好对路径中的反斜杠进行转义，这里我的做法是将在使用正斜表示路径（在win10下是可以正常运行的哦）。

目录

```
ic public function load($class){
$class = str_replace('\\','/',$class);
$path = ROOT.'/'.$class.'.php';
if (is_file($path)){
    require_once $path;
}
```

只要我们将文件路径和命名空间保存一致，就可以实现自动加载了。为了方便，将这个注册函数定义为了Core核心类的静态方法。

控制器和方法

4.1. 调用

前面我们通过路由类，获取到了目标控制器和方法名；通过注册函数，实现自动加载类的功能。接下来，只需要实例控制器对象，调用指定方法就行了，这项工作放在 `Core.php` 核心文件中实现。

```
ic public function run(){

    $route = new Lib\Route();
    $ctrl = $route->ctrl;
    $action = $route->action;

    // 控制器完全限定名称
    $ctrlName = '\\'.MOUDLE.'\Controller\\'.$ctrl.'Controller';

    // 加载控制器文件
    try{
        $ctrl = new $ctrlName();
        $ctrl->$action();
    }catch (\Exception $e){
        echo '找不到控制器:'.$ctrl;
        echo $e->getMessage();
    }
```

目录

我们在 `APP/Controller/` 目录下建立 `IndexController.php` ，并定义好相关的命名空间和类，然后访问 `shy.com/Index/index` 就可以访问相应的控制器方法了。

```
space App\Controller;
use Core\Lib\Controller;

class IndexController{
    public function index(){
        dd('PHP是世界上...');
```

```
    }
}
```

这里将 `Controller` 目录写死并不是最明智的做法，也还没有对项目进行分组，但是，一切都从最简单的开始吧，接下来，需要处理的是

控制器基类

在介绍MVC的时候提到，控制器的任务分为如下两方面：

处理传入数据（在控制器中需要对视图层的提交数据进行过滤和检测，并根据检测结果判断用户是否合法），然后将数据参数传递给模型；

从模型获取返回数据，根据视图需要进行处理，然后将结果集赋值到页面上(assign())，并展示相应的视图页面(view())

无看输出视图的方面，实际上，输出视图可以简化为加载相应的视图文件，然后通过 `<?php echo $var ?>` 进行赋值就可以了。我们将 `Controller` 基类放在 `\Core\Lib` 目录下，当然，命名空间也是。而关于向视图赋值方面，使用的是 `extract()` 将数组打散获取单个变

目录

```
space Core\Lib;
s Controller {
public $assignArr = [];

public function assign($key,$val){
    $this->assignArr[$key] = $val;
}

public function view($file){
    try{
        extract($this->assignArr);

        $pathArr = explode('\\',get_class($this));
        $pathArr[count($pathArr) - 2] ='View';
```

```
            $ctrName = $pathArr[count($pathArr) - 1];
            $pathArr[count($pathArr) - 1] = str_replace('Controller','',$ctrName);

            $path = ROOT.'/'.implode($pathArr,'/').'/'.$file.'.html';

            include_once $path;
        }catch(\Exception $e) {
            echo $e->getMessage();
        }
    }
```

上 IndexController 继承 Controller ，大功告成。

```
s IndexController extends Controller{
public function index(){
        $this->assign([
         'text'=>'Hello  PHP'
    ])
    $this->view('index');
}
```

小结

样，简单地实现了视图文件的加载与变量赋值。上面的代码写的比较丑陋，因为我正在取舍TP关于视图文件的管理和Laravel的视图文件管理，我更倾向于后者（尽管视图文件和控制器文件在文件目录下隔了很远），这是进一步学习需要深思的问题，总之，大概的视图文件加载原理就是这样。此外，由于原生的PHP标签缺乏判断和循环等模板标签，加之我实在不想再HTML文件中书写PHP代码，，因此之前的处理方式是使用 Vue.js ，这么用 Vue 会不会被打死...

5. 最后

现在，一个很简陋的PHP框架就搭好了，好吧，这根本算不上是个框架，不过，还是能够学到一点关于框架工作的流程。关于模型是另外一个很重要的知识点，因此打算单独整理一篇关于PDO的文章。学习一点后端知识，只是为了更好的写前端，好好努力啦！

ireJS使用心得

实现一个简易的JS模板引擎

目录

说点什么吧...

目录

提交

来发评论吧~

Powered By Valine
v1.4.18

世人的悲欢并不相通，我只是觉得他们吵闹。

目录