框架核心类（二）

一、实验简介

1.1 实验目的

本次实验将会在上一个实验的基础上继续编写 Labframe 框架的核心类。通过实践类的编写，更深入的了解框架的结构和运行流程。本次实验任务较多，将要完成框架所需要的全部核心类。

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

- Controller.php：控制器类
- Model.php：模型类
- View.php：视图类
- Parser.php：模板解析类

二、核心类编写

2.1 控制器类

也就是 MVC 中的 C 模块。主要负责处理一些具体的业务逻辑，并调用模型进行操作。不过我们现在即将编写的类属于控制器的基类，主要用于封装一些高层次的操作，让具体的控制器继承于它。所以我们不必要写过多的内容。

在 `core/` 下新建一个 `Controller.php` 作为控制器基类。主要结构如下：

```
<?php namespace core;

use core\View;    //使用视图类/**

* 控制器基类

*/class Controller{

    protected $vars = [];    //模板变量

    protected $tpl;        //视图模板



      //变量赋值

    final protected function assign($name,$value = '')

    {



    }

      //设置模板

    final public function setTpl($tpl='')

    {



    }
```

```
    //模板展示
    final protected function display()
    {


    }
}
```

- 变量赋值

- 主要用于在模板中展示变量的数据。平时，如果我们需要在 html 模板中打印一个变量，一般都会采用`<?php echo $var; ?>` 这种方式输出，但是当需要输出的变量很多的时候，如果每次都这样去写的话，那将是非常繁琐且无意义的工作。正是有这样的需求，模板引擎应运而生，只需要写下类似 `{$var}` 的形式便可以完成刚才的功能。关于模板引擎的内容，待会儿再讲解。这里你需要知道的就是，你想要展示数据，先给它赋值。

-

```
//将其设置为 final，子类不能改写     final protected function assign($name,$value = '')
    {
        if (is_array($name)) {
```

```
            $this->vars = array_merge($this->vars,$name);
            return $this;
        } else {
            $this->vars[$name] = $value;
        }
    }
```

-

assign 方法也同样延续了和配置参数一样的方式，支持单个变量赋值，也支持数组批量赋值。

-
-

模板设置

-

如果你还有印象的话，我应该在启动文件中的路由分发部分提到过关于模板设置的问题，那里也用到了这个方法。可以自动匹配当前方法对应的视图模板文件。

-

```
//同样为 final 类型，子类不能改写    final public function setTpl($tpl='')
    {
        $this->tpl = $tpl;
    }
```

-
-

视图展示

-

调用视图类展示此方法的视图文件。

-

```
//同上    final protected function display()
    {
        $view = new View($this->vars);    //调用视图类
        $view->display($this->tpl);    //视图类展示方法
    }
```

-

控制器基类到此编写完毕。

2.2 视图类

见名知意，这是用于展示和处理视图模板的类。供控制器调用。

在 `core/` 下建立 `View.php` 作为视图类，结构如下：

```
<?php namespace core;
use core\Config;    //使用配置类 use core\Parser;    //使用模板解析类/**
* 视图类
```

```
*/class View{

    //模板变量

    public $vars = [];



    function __construct($vars =[])

    {



    }

    //展示模板

    public function display($file)

    {



    }

}
```

- 构造方法

- ```
  function __construct($vars =[])
  ```
  ```

```
    {
        if (!is_dir(Config::get('cache_path')) || !is_dir(Config::get('complie_p
ath')) || !is_dir(Config::get('view_path'))) {
            exit('The directory does not exist');
        }
        $this->vars = $vars;
    }
```

上面的构造方法中，第一步：做了几个目录存在性判断，缓存目录是否存在，编译目录是否存在，模板文件目录是否存在，其中任何一个目录不存在都会退出程序。第二步：接受从控制器传来的模板变量。

模板展示方法

```
    public function display($file)
    {
        //模板文件
        $tpl_file = Config::get('view_path').$file.Config::get('view_suffix');
        if (!file_exists($tpl_file)) {
```

```
            exit('Template file does not exist');
        }
          //编译文件(文件名用 MD5 加密加上原始文件名)
        $parser_file = Config::get('complie_path').md5("$file").$file.'.php';
          //缓存文件(缓存前缀加原始文件名)
        $cache_file = Config::get("cache_path").Config::get("cache_prefix").$f
ile.'.html';
        //是否开启了自动缓存
      if (Config::get('auto_cache')) {
            if (file_exists($cache_file) && file_exists($parser_file)) {
                if (filemtime($cache_file) >= filemtime($parser_file) && file
mtime($parser_file) >= filemtime($tpl_file)) {
                    return include $cache_file;
                }
            }
        }
          //是否需要重新编译模板
```

```
        if (!file_exists($parser_file) || filemtime($parser_file) < filemtime($tpl_file)) {
            $parser = new Parser($tpl_file);
            $parser->compile($parser_file);
        }
        include $parser_file;     //引入编译文件
          //若开启了自动缓存则缓存模板
        if (Config::get('auto_cache')) {
            file_put_contents($cache_file,ob_get_contents());
            ob_end_clean();
        }
    }
```

-

上面的逻辑也挺简单。调用 *display* 方法需要传入一个模板文件名，然后根据传入的文件名到视图目录去寻找是否存在该模板，若不存在，退出程序。若存在，定义对应的编译文件和缓存文件。接下来判断在配置选项中是否开启了自动缓存：

-
    -

若开启了缓存，若对应的缓存文件存在且编译文件也存在，若缓存的文件的最后修改时间大于对应的编译文件且编译文件的最后修改时间大于模板文件

的修改时间，则表明缓存的文件是最新的内容，直接可以引入缓存文件，函数返回。

-

-

若不满足使用缓存文件的条件，则向下执行。若编译文件不存在或编译文件存在但是最后修改时间小于模板文件的修改时间，表明编译文件无效，需要重新编译模板文件。实例化一个编译类的对象，调用其编译方法（传入编译文件名）。

-

做完上面的操作，就可以引入编译文件了。

-

若开启了自动缓存，则生成缓存文件。这里用到了一个函数

`ob_get_contents` ，将本来输出在屏幕上的内容输入到缓冲区。再将缓冲区的内容写到缓存文件。这样就生成了缓存文件，下次就可以不用再经过编译的过程而直接展示。

-

2.3 模板解析类

这一部分内容也是模板引擎的工作核心，对模板文件进行解析，编译。我们的这个类也可以称作一个简单的模板引擎。模板引擎的出现，使得业务逻辑和视图展示得以分离，代码结构更加清晰，更多的关于模板引擎的内容，大家可以自行了解。

在 `core/` 下建立 `Parser.php` 作为模板解析类。主要内容如下：

```
<?php namespace core;

/**

*  解析

*/class Parser{
```

```
    private $content;
    function __construct($file)
    {
        $this->content = file_get_contents($file);
        if (!$this->content) {
            exit('Template file read failed');
        }
    }
    //解析普通变量
    private function parVar()
    {
        $patter = '/\{\$([\w]+)\}/';
        $repVar = preg_match($patter,$this->content);
        if ($repVar) {
            $this->content = preg_replace($patter,"<?php echo \$this->vars['$1
']; ?>",$this->content);
        }
    }
```

```
    private function parIf()
    //编译
    public function compile($parser_file){
        $this->parVar();
        file_put_contents($parser_file,$this->content);
    }
}
```

上面的内容给大家做了一个示例，只定义了解析普通变量的方法。这里使用了正则表达式来解析，首先获取模板文件的内容，然后使用正则表达式去寻找符合条件的内容，找到之后执行内容替换操作，就换成了我们所熟悉的编写方式。此方法的处理效果：将混在 html 代码中的形如 {$var} 的内容，替换为<?php echo $this->vars['var']; ?> ，这样就可以将模板变量在模板文件中展示出来，而不用每次都写很多重复的代码。

由于这个类只是负责解析模板中的特定语法，而不是真正渲染模板内容，所以不需要使用模板变量。真正的渲染过程将会在 View 中执行。我们这里默认约定的模板语法：普通模板变量，使用 {$var} 标识。当然，这不是固定的写法，你可以自行设计模板语法或直接在配置文件中设定，然后在解析的时候做一些匹配修改就行。一个完整的模板引擎所做的功能远远不止这一点，还包括了解析条件判断语法，循环语法，系统变量语法，函数使用方法等等，大家完全可以仿照上面解析普通变量的方法继续完善其他模板语法的解析：parIf()，parWhile()，parSys()，parFunc() 等。

2.4 模型类

这是 MVC 中的 M，也是最重要的一个模块。主要负责与数据库交互，我们需要在其中封装一些预设的方法，方便控制器调用以及方便的执行 CURD（增删查改）操作，并做一些日志的记录，方便我们查找失败的原因。

在 core/ 下新建一个 Model.php 作为模型基类，其他模型类都继承于它，主要结构：

```
<?php namespace core;

use core\Config;use PDO;

class Model{

    protected $db;

    protected $table;

    function __construct($table = '')

    {

        $this->db = new PDO('mysql:host='.Config::get('db_host').';dbname='.Con
fig::get('db_name').';charset='.Config::get('db_charset'),Config::get('db_user'),Confi
g::get('db_pwd'));

        $this->table = Config::get('db_table_prefix').$table;     //补充完整数据表名

    }



    /*获取数据表字段*/

    public function getFields()
```

```
    {



    }
    /*获取数据库所有表*/
    public function getTables()
    {



    }
    /*释放连接*/
    protected function free()
    {
        $this->db = null;
    }


    /*获得客户端真实的 IP 地址*/
    protected function getip()
    {
```

```
    }

    /*新增数据*/

    public function save($data = [])

    {



    }

    /*更新数据*/

    public function update($data = [],$wheres = [],$options = 'and')

    {



    }

    /*查找数据*/

    public function select($fields,$wheres = [],$options = 'and')

    {



    }

    /*删除数据*/

    public function delete($wheres = [],$options = 'and')
```

```
    {



    }
    /*错误日志记录*/
    protected function log_error($message = '',$sql = '')
    {



    }
}
```

以上就是在模型基类中主要的方法，包括增删查改，展示数据库和数据表，错误日志记录。基本上可以满足我们的使用，你也可以加一些其他的方法。

首先在构造方法中，我们需要接受一个表名，用来设置与哪个数据表交互。这会在具体的模型子类中执行。然后开始使用 PDO 方式连接数据库，你也可以使用 mysqli_ 系列函数来实现数据库的连接。

- 获取数据表字段

```
    public function getFields()
    {
```

```
        $sql = 'SHOW COLUMNS FROM `' . $this->table . '`';    //拼接
SQL 语句
        $pdo = $this->db->query($sql);    //执行
        $result = $pdo->fetchAll(PDO::FETCH_ASSOC);    //转换为索引数
组
        $info = [];
        if ($result) {
            foreach ($result as $key => $val) {
                $val = array_change_key_case($val);
                $info[$val['field']] = [
                    'name' => $val['field'],
                    'type' => $val['type'],
                    'notnull' => (bool)('' === $val['null']),
                    'default' => $val['default'],
                    'primary' => (strtolower($val['key']) == 'pri'),
                    'auto' => (strtolower($val['extra']) == 'auto_increment
'),
                ];
            }
```

```
        return $info;
        }
    }
```

-
-

获取数据库所有表

-

```
    public function getTables()
    {
        $sql = 'SHOW TABLES';
        $pdo = $this->db->query($sql);
        $result = $pdo->fetchAll(PDO::FETCH_ASSOC);
        $info = [];
        foreach ($result as $key => $val) {
            $info['key'] = current($val);
        }
        return $info;
    }
```

获得客户端真实的 IP 地址

```
    function getip() {
        if (getenv("HTTP_CLIENT_IP") && strcasecmp(getenv("HTTP_CLIENT_IP"), "unknown")) {
            $ip = getenv("HTTP_CLIENT_IP");
        } else
            if (getenv("HTTP_X_FORWARDED_FOR") && strcasecmp(getenv("HTTP_X_FORWARDED_FOR"), "unknown")) {
                $ip = getenv("HTTP_X_FORWARDED_FOR");
            } else
                if (getenv("REMOTE_ADDR") && strcasecmp(getenv("REMOTE_ADDR"), "unknown")) {
                    $ip = getenv("REMOTE_ADDR");
                } else
                    if (isset ($_SERVER['REMOTE_ADDR']) && $_SERVER['REMOTE_ADDR'] && strcasecmp($_SERVER['REMOTE_ADDR'], "unknown")) {
```

```
                    $ip = $_SERVER['REMOTE_ADDR'];
                } else {
                    $ip = "unknown";
                }
        return ($ip);
    }
```

-

这个方法主要用来获取客户端的 ip 地址，将会用于错误日志记录。主要用到了一些 PHP 预定义的函数来实现，有兴趣的可以查一下这些函数的用法。

-
-

新增数据

-

```
    public function save($data = [])
    {
        $keys = '';
        $values = '';
        foreach ($data as $key => $value) {
            $keys .= "$key,";
```

```
            $values .= "'".$value."',";
        }
        $keys = substr($keys,0,strlen($keys)-1);
        $values = substr($values,0,strlen($values)-1);
        $sql = 'INSERT INTO `'.$this->table.'` ('.$keys.') VALUES ('.$values.
')';
        $pdo = $this->db->query($sql);
        if ($pdo) {
            return true;
        }else{
            $this->log_error('save error',$sql);
            return false;
        }
    }
```

-

实现向数据表插入一条数据，需要传入一个包含字段和值的数组，形如 ['field'=>'value']，接下来将数组拆开，将其拼接成 SQL 语句并执行，执行成功返回 true，执行失败则进行错误日志的记录，返回 false。此方法也支持多字段插入。

-

-

更新数据

-

```
public function update($data = [],$wheres = [],$options = 'and')
{
    $keys = '';
    $where = '';
    foreach ($data as $key => $value) {
        $keys .= $key." = '".$value."',";
    }
    if (count($wheres) > 1) {
        foreach ($wheres as $key => $value) {
            $where .= $key . " = '" . $value . "' " . $options . " ";
        }
        $where = substr($where,0,strlen($where)-strlen($options)-2);
    } else {
        foreach ($wheres as $key => $value) {
            $where .= $key . " = '" . $value ."'";
```

```
            }
        }
        $keys = substr($keys,0,strlen($keys)-1);
        $sql = 'UPDATE '.$this->table .' SET '.$keys .' WHERE '.$where;
        $pdo = $this->db->query($sql);
        if ($pdo) {
            return true;
        } else {
            $this->log_error('update error',$sql);
            return false;
        }
    }
```

-

这个方法与新增数据类似，只是多了几个参数。更新数据需要得到更新的字段和值，更新的条件，条件之间的关系。所以此方法至少需要这三个参数。`$data` 是由更新字段和更新值组成，`$where` 是更新的条件数组，形如 `['field'=>'value']`，默认为等值关系，可支持多个条件。`$options` 是条件之间的逻辑关系。当更行条件不止一个的时候，它们之间就存在逻辑关系，例如 `id > 1 and id < 5` 这样的关系，默认

使用 `and` 。如果更新条件只有一个，就不需要考虑这个参数。如果理解比较困难，可以多花点时间看一下。上面都是一些简单的字符串操作，这里就不详解了。

-
-

查找数据

-

```
public function select($fields,$wheres = [],$options = 'and')
{
    $field = '';
    if (is_string($fields)) {
        $field = $fields;
    } elseif (is_array($fields)) {
        foreach ($fields as $key => $value) {
            $field .= $value.',';
        }
        $field = substr($field,0,strlen($field)-1);
    }
    $where = '';
    foreach ($wheres as $key => $value) {
```

```
            $where .= $key.' '.$options." '$value',";
        }
        $where = substr($where,0,strlen($where)-1);
        $sql = 'SELECT '.$field.' FROM '.$this->table.' WHERE '.$where;
        $pdo = $this->db->query($sql);
        if ($pdo) {
            $result = $pdo->fetchAll(PDO::FETCH_ASSOC);
            return $result;
        } else {
            $this->log_error('select error',$sql);
            return false;
        }
    }
```

-
-

删除数据

-

```
    public function delete($wheres = [],$options = 'and')
```

```
    {
        $where = '';
        foreach ($wheres as $key => $value) {
            $where .= $key.' '.$options." '$value',";
        }
        $where = substr($where,0,strlen($where)-1);
        $sql = 'DELETE FROM '.$this->table.' WHERE '.$where;
        $pdo = $this->db->query($sql);
        if ($pdo) {
            return true;
        } else {
            $this->log_error('delete error',$sql);
            return false;
        }
    }
```

-
  -

错误日志记录

-

```
    protected function log_error($message = '',$sql = '')
    {
        $ip = $this->getip();
        $time = date("Y-m-d H:i:s");
        $message = $message . "\r\n$sql" . "\r\n客户IP:$ip" . "\r\n时间：
$time" . "\r\n\r\n";
        $server_date = date("Y-m-d");
        $filename = $server_date . "_SQL.txt";
        $file_path = RUNTIME_PATH. 'log' . DS .$filename;
        $error_content = $message;
        $file = RUNTIME_PATH. 'log'; //设置文件保存目录
        //建立文件夹
        if (!file_exists($file)) {
            if (!mkdir($file, 0777)) {
                //默认的 mode 是 0777，意味着最大可能的访问权
                die("upload files directory does not exist and creation f
ailed");
            }
```

```
        }
        //建立 txt 日期文件
        if (!file_exists($file_path)) {
            //echo "建立日期文件";
            fopen($file_path, "w+");
            //首先要确定文件存在并且可写
            if (is_writable($file_path)) {
                //使用添加模式打开$filename，文件指针将会在文件的开头
                if (!$handle = fopen($file_path, 'a')) {
                    echo "Cannot open $filename";
                    exit;
                }
                //将$somecontent 写入到我们打开的文件中。
                if (!fwrite($handle, $error_content)) {
                    echo "Cannot write $filename";
                    exit;
                }
                //echo "文件 $filename 写入成功";
```

```
                echo "Error logging is saved!";
                //关闭文件
                fclose($handle);
            } else {
                echo "File $filename cannot write";
            }
        } else {
            //首先要确定文件存在并且可写
            if (is_writable($file_path)) {
                //使用添加模式打开$filename，文件指针将会在文件的开头
                if (!$handle = fopen($file_path, 'a')) {
                    echo "Cannot open $filename";
                    exit;
                }
                //将$somecontent 写入到我们打开的文件中。
                if (!fwrite($handle, $error_content)) {
                    echo "Cannot write $filename";
                    exit;
```

```
                }
                //echo "文件 $filename 写入成功";
                echo "——Error logging is saved!!";
                //关闭文件
                fclose($handle);
            } else {
                echo "File $filename cannot write";
            }
        }
    }
```

-

这个方法主要用来做日志处理，如果 CURD 操作失败，都会调用这个方法，记录下详细的错误信息，方便我们查看。

-

模型基类到此就编写完毕了。

-

三、总结

本次实验带领大家完成了框架所必须的全部核心类，包含视图、模板、控制器、模板解析。通过本次学习，你应该掌握了一个简单框架核心的运行流程和层次调用关系。这在很多大型框架中也是类似的实现方式。我们将会在下一节实验中做一些后期的完善工作和测试框架的运行效果。