## 基础

## 一、获取无敌软外PHP框架

下载链接:

## 二、环境要求

## 支持的服务器和数据库环境

* 支持Windows/Unix服务器环境
* 可运行于包括Apache、IIS和nginx在内的多种WEB服务器和模式
* 支持Mysql、MsSQL、PgSQL、Sqlite、Oracle、Ibase、Mongo以及PDO等多种数据库和连接

对于刚刚接触PHP的新手，我们推荐使用集成开发环境phpStudy3.0来使用软外无敌PHP框架进行本地开发和测试。

## 目录结构

web\_com 应用部署目录

├─app 应用目录（可设置）

│ ├─config.php 应用配置文件

│ ├─controller 控制器目录

│ ├─core 核心类文件

│ ├─dbType 数据库配置文件

│ ├─Exception 异常处理文件

│ ├─interface 数据库接口

│ ├─Lib Smarty模板的引用

│ ├─model 模型目录

│ ├─showInfo 错误信息提示界面

│ ├─view 视图目录

├─html WEB 部署目录（对外访问目录）

│ ├─templates

│ │ ├─css

│ │ ├─images

│ │ ├─js

│ ├─templates\_c

├─public 自动生成定义文件（参考）

│ ├─css

│ ├─image

│ ├─js

## 控制器

我们可以在目录某盘:\phpStudy\PHPTutorial\WWW\web\_com\app\controller下面找到一个IndexController.class.php 文件，这就是默认的Index控制器文件。

控制器类的命名方式是：**控制器名（驼峰法，首字母大写）+Controller**

控制器文件的命名方式是：**类名+class.php（类文件后缀）**

默认的欢迎页面其实就是访问的Index控制器类的Index操作方法 我们修改默认的index操作方法如下：

1. class IndexController extends Controller{
2. public function Index(){
3. $this->Tpl->display("index.tpl");
4. }
5. }

再次在浏览器输入http://localhost/web\_com/，浏览器会显示：欢迎使用小清新框架的页面。

## 开发规范

## 命名规范

使用PHP开发的过程中应该尽量遵循下列命名规范：

* 类文件都是以.class.php为后缀（这里是指的PHP内部使用的类库文件，不代表外部加载的类库文件），使用驼峰法命名，并且首字母大写，例如 DbMysql.class.php；
* 类的命名空间地址和所在的路径地址一致，例如 app\ Controller\UserController类所在的路径应该是 app /Controller/UserController.class.php；
* 确保文件的命名和调用大小写一致，是由于在类Unix系统上面，对大小写是敏感的；
* 类名和文件名一致（包括上面说的大小写一致），例如 UserController类的文件命名是UserController.class.php， InfoModel类的文件名是InfoModel.class.php， 并且不同的类库的类命名有一定的规范；
* 函数、配置文件等其他类库文件之外的一般是以.php为后缀（第三方引入的不做要求）；
* 方法的命名使用驼峰法，并且首字母小写或者使用下划线“\_”，例如 getUserName，\_parseType，通常下划线开头的方法属于私有方法；
* 属性的命名使用驼峰法，并且首字母小写或者使用下划线“\_”，例如 tableName、\_instance，通常下划线开头的属性属于私有属性；
* 以双下划线“\_\_”打头的函数或方法作为魔法方法，例如 \_\_call 和 \_\_autoload；
* 常量以大写字母和下划线命名，例如 HAS\_ONE和 MANY\_TO\_MANY；
* 配置参数以大写字母和下划线命名，例如HTML\_CACHE\_ON；
* 语言变量以大写字母和下划线命名，例如MY\_LANG，以下划线打头的语言变量通常用于系统语言变量，例如 \_CLASS\_NOT\_EXIST\_；
* 由于PHP默认全部使用UTF-8编码，所以请确保你的程序文件采用UTF-8编码格式保存
* PHP的模板文件默认是以.html 为后缀（可以通过配置修改）；

## 开发建议

在使用PHP进行开发的过程中，我们给出如下建议，会让你的开发变得更轻松：

* 遵循框架的命名规范和目录规范；
* 开发过程中尽量开启调试模式，及早发现问题；
* 多看看日志文件，查找隐患问题；

## 架构

1. **入口文件:**

文件夹最外面的index.php

//index.php引导文件

//定义root绝对路径

1 define("ROOT",dirname(\_\_FILE\_\_));

//设置当前为调试模式

2 define('DEBUG',1);

//设置错误提示类型语言包 1：中文 2：英文 默认为中文

3 define('ERRORINFOLANGUAGE',1);

//引入核心模块

4 require ROOT."/app/core/Service.php";

//引入异常类

//核心类加载

5 Service::run();

1. **URL访问**：

某盘:\phpStudy\PHPTutorial\WWW\web\_com\app\core\UrlFilter.php

//获取url控制器路径

1.$this->path = $\_SERVER['REQUEST\_URI'];

2.$pathArr = explode('/',$this->path);

## 配置

1. **配置目录**

│ ├─config.php 应用配置文件

│ │ ├─’webInfo’ 站点信息配置管理选项

│ │ ├─’core’ 核心加载类配置管理

│ │ ├─’db’ 数据库配置信息管理

**2、配置格式**

return array("webInfo"=>array(

'title'=>'学生管理系统-bs架构',

'host' =>'http://localhost/web\_com',

'port' =>'80' ),

//核心加载类配置管理

'core'=>array(

/\*引入模块\*/ 'interface/InterDb',

'interface/InterSmarty',

/\*引入核心类\*/

'UrlFilter', //url过滤器

'Controller', //控制器

'Db',

'Page',

'Tpl' //模板引擎类 ),

'page'=>array(

'pageNum'=>7,//每页显示的个数

'between'=>3,//分页区间 ),

//数据库配置信息管理

'db'=>array(

'host'=>'p:localhost',

'dbName'=>'web\_com',

'username'=>'root',

'password'=>'root',

'port'=>3306,

'dbType'=>'Mysql\_Pre',

'charset'=>'utf8'

)

)

**3、配置加载**

详情于service.php文件中。

**\*应用配置**

应用配置文件也就是调用所有模块之前都会首先加载的公共配置文件（默认位于app/config/config.setting.php）。

如果更改了公共模块的名称的话，公共配置文件的位置也相应改变

1 class Service{

//定义配置文件路径

2 public static $configPath = '/app/config/config.setting.php';

//定义配置文件

3 public static $configArr;

//对象容器

4 public static $singleArr = array();

5 /\*\*

\* 加载核心类

6 \*

7 \* @return [type] [description]

8 \*/

9 public function \_\_construct(){

//引入异常类文件

10 require ROOT."/app/Exception/MsgException.php";

11 }

12

13 public static function loadCore()

//引入配置文件

//判断当前配置文件是否存在

14 if(file\_exists(ROOT.self::$configPath)){

15 self::$configArr = require ROOT.self::$configPath;

//文件数组是否存在

16 if(isset(self::$configArr) && !is\_null(self::$configArr)){

//加载核心配置类文件,检测core数组是否存在配置文件

17 if(array\_key\_exists('core', self::$configArr)){

//遍历核心类

18 foreach(self::$configArr['core'] as $key=>$value){

//判断是否核心类存在

//引入核心类文件

19 $index = strpos($value,'/');

//核心类模块接口加载

20 if($index>0){

21 if(file\_exists(ROOT."/app/core/".$value.".interface.php")){

22 require ROOT."/app/core/".$value.".interface.php";

23 }else{

24 throw new MsgException($value,1004);

}

//核心类加载

25 }else{

26 if(file\_exists(ROOT."/app/core/".$value.".php")){

27 require ROOT."/app/core/".$value.".php";

//实例化核心类对象

//单列模式实现模块对象初始化

// 单例模式实现对象容器存储

28 if(!isset(self::$singleArr[$value]) || is\_null(self::$singleArr[$value])){

29 self::$singleArr[$value] = new $value(self::$configArr);

30 }

31 }else{

32 throw new MsgException($value,3001);

}

}

}

33 }else{

34 //echo "配置文件加载核心类选项不存在!";

35 throw new MsgException('core',1009);

36 }

37 }else{

//echo "配置文件数组不存在或者没有设置!";

38 throw new MsgException(self::$configArr,1002);

//throw new MsgException("配置文件数组不存在或者没有设置!");

39 }

40 }else{

41 throw new MsgException(self::$configPath,1001);

//throw new MsgException("配置文件不存在!");

42 }

/\*$url = new Url();

$db = new Db();

$tree = new Tree();\*/

}

43 public static function run(){

44 new self;

//加载异常类

//加载核心类文件

45 self::loadCore();

//url路径解析

46 self::$singleArr['UrlFilter']->getUrlFilter(self::$singleArr);

}

}

## 控制器

**1.控制器定义**

class Index {

public function Index()

{

return 'index';

}}

控制器类文件的实际位置是

app\index\controller\Index.php

**2、控制器初始化**

class IndexController extends Controller{

public function Index()

{

$this->Tpl->display("index.tpl");

}

}

会输出

index.Tpl

**3、跳转和重定向**

class PowerController extends Controller{

//修改信息

public function UpdatePower()

{

$uid = $this->UrlFilter->appPar['uid'];

$where = array('id'=>$uid);

$result = $this->Db->Table('power')

->Field(' id,powerName,url ')

->Where($where)

->Order()

->Select();

$this->Tpl->assign("arr",$result);

$this->Tpl->display("powerReg.tpl");

}

}

**4. 操作绑定到类**

**定义:**

我们提供了把每个操作方法定位到一个类的功能，可以让你的开发工作更细化。

**例如：**

原来的控制器文件定义位置为：

1. app/ Controller/IndexController.class.php

控制器类的定义如下：

1. class IndexController extends Controller{
2. public function index(){
3. echo '执行Index控制器的index操作';
4. }
5. }

可以看到，实际上我们调用的是 app\Controller\IndexController 类的**index**方法。

控制器文件位置改为：

1. app/Controller/Index/index.class.php

控制器类的定义如下：

1. class index extends Controller{
2. public function run(){
3. echo '执行Index控制器的index操作';
4. }
5. }

现在，我们调用的其实是 app\Controller\Index\index 类的**run**方法。

## 数据库

1. **配置定义**

config.setting.php

//数据库配置信息管理

'db'=>array(

'host'=>'p:localhost',

'dbName'=>'web\_com',

'username'=>'root',

'password'=>'root',

'port'=>3306,

'dbType'=>'Mysql\_Pre',

'charset'=>'utf8'

)

**2、方法配置**

Controller.php

1 class Controller{

2 public $app; //核心累对象属性

/\*构造函数，初始化控制器类\*/

3 public $db;

4 public function \_\_construct($configArr='',$singArr=''){

5 $this->configArr = $configArr;

//$this->singArr = $singArr;

// $this->db = $singArr['Db']->dbObject;

6 if($singArr!=''){

7 foreach($singArr as $key=>$value){

//判断object这个key值是否存在在数组中

8 if(!array\_key\_exists('Object',$value)){

$this->$key = $value;

continue;

}

9 $this->$key = $value->Object;

}

}

}

**3、模型类定义**

步骤：

\* 1、加载数据库类

\* 2、实例化数据库

\* 3、实例化数据库对象赋值给db类的成员属性

class Db{

1 public $Object; //数据库对象

2 public $configArr; //配置文件

//通过配置文件驱动数据库操作类

3 public function \_\_construct($configArr=''){

$this->configArr = $configArr;

$this->loadDbObject();

}

//加载数据库类

4 public function loadDbObject(){

数据库文件安全目录扫描：

1、遍历dbType文件目录是否有非法文件存在 如果有后缀、扩展名为.exe .bat .sh 则非法,相反如果文件都为.php 则目录安全

2、opendir();打开一个文件句柄

3、读取文件数据流 readdir()

4、判断当前是否为目录或文件

5、如果是目录递归，否则获取文件的扩展名，判断是否为.php 如果不是删除

6、导入数据库操作类文件

//调用scanDbDir检测

5 $this->scanDbDir();

6 require ROOT.'/app/dbType/'.$this->configArr['db']['dbType'].'DB.php';

7 $db = $this->configArr['db']['dbType'].'DB';

8 $this->Object = new $db($this->configArr);

9 if (!$this->Object instanceof InterDb) {

10 throw new MsgException('',3001);

11 }

12 }、

13public function scanDbDir($path=''){

14 if($path!=''){

15 $path = ROOT."\app\dbType\\".$path;

16 }else{

17 $path = ROOT."\app\dbType";

18 }

16 $od = opendir($path); //打开一个文件句柄，打开文件的数据流

17 readdir($od);

18 readdir($od);

19 while($dirStr=readdir($od)){

20 $newPath = $path."\\".$dirStr;

//判断是否为目录

21 if(is\_dir($newPath)){

//递归

22 $this->scanDbdir($dirStr);

23 }else{

24 $extention = pathinfo($dirStr);

25 if($extention['extension']!='php'){

26 throw new MsgException($dirStr,4001);

}

}

}

}

27 public function getDbObject(){

28 echo "我是数据库对象";

}

}

**4、查询数据库**

**以**PowrController.class.php**为例:**

1 class PowrController extends Controller{

2 public $app;

3 public function powrManage(){

/\*可变变量 当前页 查询数据的limit的偏移量 数据库的总条数

总的分页逻辑结构

通过获取分页对象，然后调用getPage方法获取整个分页\*/

//获取当前页面

4 $page = $this->getPage();

//获取当前排序方式 asc升序 desc降序

5 $order = $this->getOrder();

//获取数据库总行数

6 $count = $this->getTotalRow("powr");

//返回分页页码

7 $pageList = $this->Page->getPage($page,$count,'Powr');

8 $result = $this->Db->Table("powr")

9 ->Field()

10 ->Order($order)

11 ->Limit($this->Page->offset,$this->Page->pageNum)

12 ->Select();

13 $this->Tpl->assign('pageList',$pageList);

14 $this->Tpl->assign('arrList',$result);

15 $this->Tpl->display("powrManage.tpl");

}

**4、删除数据库**

**以**PowrController.class.php**为例:**

思路：

获取用户id

根据用户id删除用户

删除成功用户

通过get获取url地址参数

1 public function DelePowr(){

2 $uid = $this->UrlFilter->appPar['uid'];

3 $result = $this->Db->Table("powr")

4 ->where("id={$uid}")

5 ->Delete();

6 if($result){

//删除成功

7 $this->Notice('用户删除成功','?/Powr/PowrManage');

8 }else{

//删除失败

9 $this->Notice('用户删除失败','?/Powr/PowrManage');

}

## 模型

1. **定义**

模型类并非必须定义，只有当存在独立的业务逻辑或者属性的时候才需要定义。

php数据提交流程

前端表单提交过来的数据

-》写入到数据的持久层

-》在控制器中调用数据持久层数据

—》把数据写入到数据库

**2.初始化**

1.class UserModel implements Model {

2.}

**例如：**

1 class UserModel implements Model{

2 public $userName; //用户名

3 public $passWord; //密码

4 public $email; //邮箱

5 public function setName($userName){ //设置用户名

6 $this->userName = $userName;

7 }

8 public function getUserName(){ //获取用户名

9 return $this->userName;

}

}

## 视图

1. 视图实例化
2. 模板引擎
3. 模板赋值

## 安全

**防止sql注入:**

对于WEB应用来说，SQL注入攻击无疑是首要防范的安全问题，系统底层对于数据安全方面本身进行了很多的处理和相应的防范机制，例如：

1.$User = M("User"); // 实例化User对象

2.$User->find($\_GET["id"]);

即便用户输入了一些恶意的id参数，系统也会强制转换成整型，避免恶意注入。这是因为，系统会对数据进行强制的数据类型检测，并且对数据来源进行数据格式转换。而且，对于字符串类型的数据，PHP都会进行escape\_string处理(real\_escape\_string,mysql\_escape\_string)，如果你采用PDO方式的话，还支持参数绑定。

通常的安全隐患在于你的查询条件使用了字符串参数，然后其中一些变量又依赖由客户端的用户输入。

**要有效的防止SQL注入问题，我们建议：**

* 查询条件尽量使用数组方式，这是更为安全的方式；
* 如果不得已必须使用字符串查询条件，使用预处理机制；
* 使用自动验证和自动完成机制进行针对应用的自定义过滤；
* 如果环境允许，尽量使用PDO方式，并使用[参数绑定](http://document.thinkphp.cn/manual_3_2.html" \l "param_bind)。

**查询条件预处理：**

where方法使用字符串条件的时候，支持预处理（安全过滤），并支持两种方式传入预处理参数，例如：

1.$Model->where("id=%d and username='%s' and xx='%f'",array($id,$username,$xx))->select();

2.// 或者

3.$Model->where("id=%d and username='%s' and xx='%f'",$id,$username,$xx)->select();

模型的query和execute方法 同样支持预处理机制，例如：

1.$model->query('select \* from user where id=%d and status=%d',$id,$status);

2.//或者

3.$model->query('select \* from user where id=%d and status=%d',array($id,$status));

execute方法用法同query方法。

## 附录

**鸣谢：**

在无敌软外PHP手册的编写过程中，要感谢我们的带队领导以及包括王馨、何施彦、戴澳甜、肖宁峰、吴浩在内的38名小组成员、社区核心团队成员和官方QQ群活跃成员、论坛活跃用户的参与和反馈，由于人数众多，不再一一列出他们的名字，谨对他们的工作和付出表示感谢！

把这个手册免费开放给大家，愿大家前程似锦，未来可期。