环境要求：PHP>=5.4,安装PDO数据库驱动。

1.命名规范

1.1目录和文件

* 目录小写+下划线；
* 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，以.class.php为后缀；；
* 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）；

1.2函数和类、属性命名

* 类的命名采用驼峰法，并且首字母大写，例如 User、UserType，默认不需要添加后缀，例如UserController直接命名为User；
* 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；
* 以双下划线“\_\_”打头的函数或方法作为魔法方法，例如 \_\_call 和 \_\_autoload；

1.3常量和配置

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

2.目录结构

project 应用部署目录

├─webroot 外部访问根目录

│ ├─index.php 应用入口文件

├─app 应用目录（可设置）

│ ├─common 公共模块目录

│ ├─extend 扩展类库目录

│ ├─modules 模块目录

│ │ ├─view 公共视图目录

│ │ ├─home home模块目录

│ │ │ ├─controller home模块控制器目录

│ │ │ ├─view home模块视图目录

│ │ ├─admin admin模块目录

│ │ │ ├─controller admin模块控制器目录

│ │ │ ├─view admin模块视图目录

│ │ ├─…… 更多模块

│ ├─model 模型目录

│ ├─config.php 应用配置文件

│ └─init.php 应用初始化文件

├─mvc 框架目录

│ ├─common 公共函数与页面目录

│ │ ├─function 公共函数目录

│ │ ├─page 公共页面目录

│ ├─config 框架配置目录

│ ├─lib 框架类库目录

│ ├─core 框架Web应用核心目录

│ ├─util 框架工具包目录

│ ├─require\_list.php 载入文件配置

│ ├─init.php 框架引导文件

3.架构

3.1入口文件

入口文件主要完成：

* 定义框架路径、项目路径（可选）
* 定义框架相关常量（可选）
* 定义框架需要绑定的模块（可选）
* 载入框架入口文件（必须）

define('APP\_PATH',\_\_dir\_\_.'/../app/'); //di定义应用目录

define('MODULE','home,admin'); //定义需绑定的模块，以“,”号分割

define('DEBUG',true); //定义调试模式

require(\_\_dir\_\_.'/../mvc/init.php'); //载入框架

3.2 URL模式

3.2.1大小写

URL是不区分大小写的，采用驼峰或下划线均可，也就是说 URL里面的**模块/控制器/操作名**会自动转换为小写，控制器和方法在最后调用的时候会转换为驼峰法处理。

例如：

http://localhost/index.php?c=Index&a=read

// 和下面的访问是等效的

http://localhost/index.php?c=index&a=read

在这种URL不区分大小写情况下，如果要访问驼峰法的控制器类，则需要使用：

http://localhost/index.php?c=index\_test&a=read

3.2.2普通模式

**普通模式**也就是传统的GET传参方式来指定当前访问的模块和操作，例如： http://localhost/?m=home&c=user&a=login&var=value

m参数表示模块，c参数表示控制器，a参数表示操作（当然这些参数都是可以配置的），后面的表示其他GET参数。

3.2.3 PATHINFO模式

**PATHINFO**对应上面的URL模式，PATHINFO模式下面的URL访问地址是： http://localhost/index.php/home/user/login?var=value

3.3命名空间

所有定义的类都需要有命名空间，命名空间需要和所在的目录一直，不包括应用程序根目录 。

例如：controller的命名空间为，home\controller，model的命名空间为model。

3.4配置

3.4.1配置定义

配置文件以数组返回的形式定义，并且所有的配置为只读，例如，app目录下的config.php内容为：

//项目配置文件

return [

'DB\_CONFIG'=>[

'DB\_HOST' => '', //数据库地址

'DB\_USER' => '', //数据库用户名

'DB\_PWD' => '', //数据库密码

'DB\_NAME' => '', //数据库名

'DB\_PREFIX' => '', //表前缀

'DB\_MAP' => '', //实体类和数据库字段名映射方式，支持hump（大驼峰），smallHumnp（小驼峰），humpToSmall（小写下划线）

'DB\_TYPE' => '' //数据库驱动类型，支持mysqli和pdo

]

];

3.4.2配置读取

读取配置文件需要用到预定义函数 config(键值) 来读取，例如：需要访问上述的数据库配置，使用 config(‘DB\_CONFIG’) 来获取数据库配置。如果配置项不存在，则返回空字符串。传入两个参数可读取二级配置项，最多支持读取三级配置项。

4.控制器

4.1控制器定义

一个典型的控制器类定义如下：

namespace home\controller;

class Index {

public function index(){

return 'hello world!';

}

}

返回类型支持字符串，数组和模板类。数组将被转换为json字符串输出。

4.2渲染输出

如果控制器定义为：

namespace home\controller;

class Index {

public function index(){

return view();

}

}

上述控制器会将view/index/index.php的内容呈现出来，如果需要呈现其他页面文件则将控制器定义为：。

namespace home\controller;

class Index {

public function index(){

return view(‘test’);

}

}

这样控制器就会将view/index/test.php的内容呈现出来，如果需要访问其他控制器目录下的视图文件则将控制器定义为：

namespace home\controller;

class Index {

public function index(){

view(‘test/test’);

}

}

这样控制器就会将view/test/test.php的内容呈现出来。

4.3依赖注入

控制器可根据声明的参数的名字，传入对应的值，内置支持的请求参数包括，$request（Request对象），$post（post参数），$get（get参数）。还可以直接过去$post，$get中的参数值。

namespace home\controller;

class Index {

//获取Request对象

public function index1($request){

……;

}

//获取$get,$post参数

public function index2($get,$post){

……;

}

//获取$get或$post中name与age字段的值

public function index3($name,$age){

……;

}

}

也可以通过Context来进行自定义对象的注入，注入的对象都为单一实例。

//注入Student对象

Context()->factory(‘student’,function(){

return new Student()

});

//注入hello值

Context()->factory(‘hello’,function($student){

return $student->say(‘hello world’);

});

namespace home\controller;

class Index {

//获取Student对象

public function index1($student){

……;

}

public function index1($hello){

……;

}

}

4.4方法注解

可通过注解的方式来对请求方法进行控制。

namespace home\controller;

class Index {

/\*\*

\* 声明为get方法，其他方法请求无效

\* @method get

\*/

public function index1(){

……;

}

/\*\*

\* 声明为post方法，其他方法请求无效,且将被重定向到index1方法

\* @method post

\* @redirect index.php?a=index1

\*/

public function index2(){

……;

}

}

5.请求

5.1请求信息

5.1.1实例化

如果要获取当前的请求信息，可以使用core\Request类，  
除了下文中的

$request=Request::instance();

也可以使用助手函数

$request=request();

5.1.2获取URL信息

假设被请求的URL是 http://example.com/admin/index.php/product?id=100

$request=request();

// 获取当前域名

echo $request->url() . '<br/>';

// 返回 /admin/index.php/product?id=100

// 获取完整URL地址

echo $request->absoluteUrl() . '<br/>';

// 返回  http://example.com/admin/admin/index.php/product?id=100

// 获取path\_info信息

echo $request->pathinfo() . '<br/>';

// 返回 /product

5.1.3获取 模块/控制器/操作名称

$request = request();

echo $request->module(); // 获取模块名称

echo $request->controller(); // 获取控制器名称

echo $request->action(); // 获取操作名称

5.1.4获取请求参数

$request = request();

echo $request->method() . '<br/>'; // 获取请求方法

echo $request->ip() . '<br/>'; // 获取请求IP地址

echo $request->port() . '<br/>'; // 获取请求端口号地址

echo $request->heard(‘key’). '<br/>'; // 获取请求头信息

echo $request->time(). '<br/>'; // 获取请求的时间戳

5.2输入变量

可以通过Request对象完成全局输入变量的检测、获取和安全过滤，支持包括$\_GET、$\_POST和$\_SERVER变量

// 获取所有get请求参数

request()->get();

// 获取name参数信息

request()->get(‘name’);

// 获取name参数信息,如果为空则返回null

// 获取name参数信息,并设置默认值

request()->get(‘name’,’’);

// 获取name参数信息,如果为空则返回空字符串

// 检测变量是否存在

request()->hasGet(‘name’);

// 获取int类型数据

request()->getInt(‘age’);

// 获取float类型数据

request()->getFloat(‘age’);

// 获取double类型数据

request()->getDouble(‘age’);

// 获取所有post请求参数

request()->post();

// 获取name参数信息

request()->post(‘name’);

// 获取name参数信息,如果为空则返回null

// 获取name参数信息,并设置默认值

request()->post(‘name’,’’);

// 获取name参数信息,如果为空则返回空字符串

// 获取int类型数据

request()->postInt(‘age’);

// 获取float类型数据

request()->postFloat(‘age’);

// 获取double类型数据

request()->postDouble(‘age’);

// 检测变量是否存在

request()->hasPost(‘name’);

// 获取$\_SERVER变量信息

request()->server(‘key’)

// 判断是否为get方法

request()->isGet();

//判断是否为post方法

request()->isPost();

5.3请求头信息

可以使用Request对象的header方法获取当前请求的HTTP 请求头信息，例如：

$info = Request::instance()->header();

echo $info['accept'];

echo $info['accept\_encoding'];

echo $info['user\_agent'];

也可以直接获取某个请求头信息，例如：

$agent = Request::instance()->header('user\_agent');

5.4请求转发

Request可以将客户端的请求转发给其它的模块，控制器及操作方法，例如reques()->forward($module,$controller,$action)。

6.数据库

6.1数据库配置

在app目录下的config中的返回数组中加入以下数组进行数据库配置。

return [

……

'DB\_CONFIG'=>[

'DB\_HOST' => '', //数据库地址

'DB\_USER' => '', //数据库用户名

'DB\_PWD' => '', //数据库密码

'DB\_NAME' => '', //数据库名

'DB\_PREFIX' => '', //表前缀

'DB\_MAP' => '', //实体类和数据库字段名映射方式，支持hump（大驼峰），smallHumnp（小驼峰），humpToSmall（小写下划线）

'DB\_TYPE' => '' //数据库驱动类型，支持mysqli和pdo

]

];

6.2基本使用

配置了数据库连接信息后，就可以直接使用数据库运行原生SQL操作了。在进行sql传参时最好采用?代替原始变量，以防sql注入。

Db::query('select \* from test\_user where id=?',8);

Db::execute('insert into test\_user (id, name) values (?,?)',8,'test');

6.3数据查询

查询一个数据使用：

Db::table('test\_user')->where('id',1)->find();

find 方法查询结果不存在，返回 null

查询数据集使用：

Db::table('test\_user')->where('status',1)->select();

select 方法查询结果不存在，返回空数组

如果设置了数据表前缀参数的话，可以使用（假设表前缀为test\_）：

Db::name('user')->where('id',1)->find();

Db::name('user')->where('status',1)->select();

可以使用以下方式查询全表名。

Db::name('/test\_user')->where('id',1)->find();

Db::name('/testuser')->where('status',1)->select();

在find和select方法之前可以使用所有的链式操作方法。

默认情况下，find和select方法返回的都是数组。

6.4助手函数

系统提供了一个db助手函数，可以更方便的查询：

db('test\_user')->where('id',1)->find();

db('test\_user')->where('status',1)->select();

按列查询可使用：

db('test\_user')->where('id',1)->find(‘name’);

db('user')->where('status',1)->select(‘id,name’);

6.4表别名

db('test\_user[user]')->where('id',1)->find(‘name’);

6.5数据添加

6.5.1添加一条数据

使用 Db 类的 add 方法想数据库提交数据

$data = ['foo' => 'bar', 'bar' => 'foo'];

Db::table('test\_user')->add($data);

6.5.2添加多条数据

添加多条数据直接向 Db 类的 addAll 方法传入需要添加的数据即可

$data = [

['foo' => 'foo', 'bar' => 'bar'],

['foo' => 'foo1', 'bar' => 'bar1'],

['foo' => 'foo2', 'bar' => 'bar2']

];

Db::table('test\_user')->addAll($data);

6.6数据更新

6.6.1更新一条数据

$data = ['foo' => 'bar', 'bar' => 'foo'];

Db::table('test\_user') ->where(‘id’,1)->update($data);

6.6.2更新多条数据

$data = [

['foo' => 'foo', 'bar' => 'bar'],

['foo' => 'foo1', 'bar' => 'bar1'],

['foo' => 'foo2', 'bar' => 'bar2']

];

Db::table('test\_user')->where(‘id’,1)->updateAll(‘foo’,$data);

6.7数据删除

Db::table('test\_user')->where(‘id’,1)->delete();

6.8链式操作

支持的连贯操作方法有：

| **链式操作** | **作用** | **支持的参数类型** |
| --- | --- | --- |
| where | 用于AND查询 | 字符串 |
| whereOr | 用于OR查询 | 字符串 |
| condition | 用于原生查询 | 字符串 |
| table | 用于定义要操作的数据表名称 | 字符串 |
| data | 用于新增或者更新数据之前的数据对象赋值 | 数组 |
| order | 用于对结果排序 | 字符串 |
| limit | 用于限制查询结果数量 | 数字 |
| page | 用于查询分页（内部会转换成limit） | 数字 |
| group | 用于对查询的group支持 | 字符串 |
| having | 用于对查询的having支持 | 字符串 |
| join | 用于对查询的join支持 | 字符串 |
| union | 用于对查询的union支持 | 字符串、Db对象 |
| distinct | 用于查询的distinct支持 | 无参数 |
| sql | 只返回sql语句不执行 | 无参数 |

6.8.1 where

db('test\_user')->where(‘id’,1)->select();

// 生成的sql语句为SELECT \* FROM `test\_user` WHERE `id`=1

db('test\_user')->where(‘name’,‘test’)->select();

// 生成的sql语句为SELECT \* FROM `test\_user` WHERE `name`=’test’

db('test\_user')

->where('name','test')

->where('age','>',20)

->select();

// 生成的sql语句为SELECT \* FROM `test\_user` WHERE `name`=‘test’ and `age`>20

6.8.2 table

Db::table('test\_user')->select();

或者

db('test\_user')->select();

6.8.3 data

db('test\_user')->data([‘name’=>‘test’,‘age’=>‘20’])->insert();

db('test\_user')->data([‘name’=>‘test’,‘age’=>‘21’])->where(‘id=?’,1)->update();

// 与下面是等价的

db('test\_user')->insert([‘name’=>‘test’,‘age’=>‘20’]);

db('test\_user')->where(‘id=?’,1)->update([‘name’=>‘test’,‘age’=>‘21’]);

6.8.4 order

db('test\_user')->order(‘id’)->select(); // 按id升序进行排序

db('test\_user')->order(‘id desc’)->select(); // 按id降序排序

db('test\_user')->order(‘id desc,name’)->select(); // 按id降序排序,name升序排序

6.8.5 limit

db('test\_user')->limit(10)->select(); // 查询前10行

db('test\_user')->limit(5,10)->select(); // 查询前6至15行

6.8.6 page

db('test\_user')->page(2,10)->select(); // 每页10行，获取第2页数据

db('test\_user')->page(3,20)->select(); //每页20行，获取第3页数据

db('test\_user')->limit(10)->page(2)->select(); //每页10行，获取第2页数据

6.8.7 group

db('test\_user')->group(‘name’)->select(‘name,max(age)’);

// 生成的sql语句为select name,max(age) from test\_user group by name

6.8.8 having

db('test\_user')->group(‘name’)->having(‘sum(score)>?’,250)->select(‘name,max(age)’);

// 生成的sql语句为select name,max(age) from test\_user group by name having sum(score)>250

6.8.9 join

**参数**

join

要关联的表（不包括表前缀）。

condition

关联条件。

type

关联类型。可以为:inner、left、right、full，不区分大小写，默认为inner

db('test\_user u')->join('test\_admin a','u.age=a.age')->select()

// 生成的sql语句为select from test\_user u inner join test\_admin on u.age=a.age

6.8.10 union

$db1=Db::table('test\_user')->where(‘age>?’,20);

$db2=Db::table('test\_admin')->where(‘age>?’,20);

Db::union(‘name,age’, $db1, $db2)->select();

// 生成的sql语句为：

select from test\_user where age>20 unin select from test\_admin where age>20

// union 允许重复

Db::union(‘name,age’, $db1, $db2,true)->select();

// 生成的sql语句为：

select from test\_user where age>20 unin all select from test\_admin where age>20

// union all不允许重复

6.8.11 distinct

db('test\_user')->distinct()->select(‘age’);

// 生成的sql语句为：select distinct age from test\_user

6.8.12 sql

sql方法用于不执行查询，直接返回上sql语句：

db('test\_user')->where(‘name=?’,test)->select();

// 返回sql语句为：select \* from test\_user where name=‘test’

7模型

7.1实体类

定义一个Person实体类：

namespace model;

use core\Model;

class Person{

}

模型会自动对应数据表，模型类的命名规则是除去表前缀的数据表名称，采用驼峰法命名，并且首字母大写，例如：

| **模型名** | **约定对应数据表（假设数据库的前缀定义是 test\_）** |
| --- | --- |
| Person | test\_person |
| UserType | test\_user\_type |

若想更改表名，可在类上面添加注解，来设置表名：

namespace model;

use core\Model;

/\*\*

\* 假设表前缀为test\_,则全表名为test\_new\_table

\* @table new\_table

\*/

class Person extends Model{

}

7.2实体类映射

namespace model;

use core\Model;

class Person {

//@id表示为id字段，@column表示字段将被映射为数据库字段，后面为字段名称，无字段名称时，将属性名称格式化为大驼峰，小驼峰，下划线三种模式（配置文件中定义）

/\*\*

\* @id

\* @column PersonID

\*/

public $id;

//@column表示字段将被映射为数据库字段，@rule验证规则，@msg错误提示

/\*\*

\* @column Name

\* @rule required|name

\* @msg 姓名必须|姓名格式不正确

\*/

public $name;

//@column表示字段将被映射为数据库字段，@rule验证规则，@msg错误提示

/\*\*

\* @column NickName

\* @rule required|regex:/^[\x{4e00}-\x{9FA5}\x{f900}-\x{fa2d}\w\-]+$/u|length:4,30

\* @msg 用户昵称不能为空|用户昵称由4-30个中文、英文、数字、-、\_ 组成;

\*/

public $nickName;

7.3模型类

// 实例化

$model=Model::map(‘Person’)

// 构建实体类，返回Person对象，姓名为myname，年龄为20

$person=$model->factory([‘name’=>’myname’,’age’=>20])

7.4添加

$person=new Person();

$person->name=’myname’;

$person->age=20;

$model=Model::map(‘Person’)

$model->add($person)

7.5查找

如果查找失败则返回null，如果查找成功则返回全部属性。

$model=Model::map(‘Person’);

$model->findById(‘name’,’age’,1);

// 等同于

$model->where(‘id’,1)->find(‘name’,’age’)

7.6更新

$person=new Person();

$person->age=18;

$person->id=1;

$model=Model::map(‘Person’);

$model->update($person);

7.7删除

$model=Model::map(‘Person’);

$model->deleteById(1);

8视图

8.1视图输出

输出视图使用view函数或者View类

namespace home\controller;

class Index {

public function index(){

return view(‘test’); //输出app/home/view/index/test.php

}

}

8.2视图定位

模板文件目录默认位于模块的view目录下面，视图类的fetch方法中的模板文件的定位规则为：控制器@视图文件

namespace home\controller;

class Index {

public function index(){

view(‘test/test.php’); //输出app/home/view/test/test.php

}

}

8.3变量输出

namespace home\controller;

class Index {

public function index(){

view(‘test.php’)->assign([‘data’=>’hello world’]); //输出app/home/view/index/test.php

}

}

//test.php

<?php

echo $data

8.4 for循环输出

使用内置函数volist可对数组进行for循环输出。volist(循环输出函数,为空时输出函数,输出起始位置，输出长度，输出步长)。

| **参数** | **参数类型** | **必须** | **默认值** |
| --- | --- | --- | --- |
| $array | array | 是 | 无 |
| $output | callable | 是 | 无 |
| $empty | callable | 否 | null |
| $offest | int | 否 | 0 |
| $length | int | 否 | null |
| $step | int | 否 | 1 |

8.4.1默认输出

输出数组全部如下所示：

namespace home\controller;

class Index {

public function index(){

view('test.php',['array'=>['java','php','test','c++','python']]);

}

}

//test.php

<?php

volist($array,function($item){

echo $item.'<br/>';

};

输出后的内容为：

php

java

test

c++

python

8.4.2输出索引

volist($array,function($item,$id){

echo $id.'-'.$item.'<br/>';

};

输出后的内容为：

0-php

1-java

2-test

3-c++

4-python

8.4.3空数组输出

volist($array,function($item,$id){

echo $id.'-'.$item.'<br/>';

},function(){

echo '无数据<br/>';

});

数组不为空时输出：

0-php

1-java

2-test

3-c++

4-python

数组为空时输出为：

无数据

8.4.4部分输出

volist($array,function($item,$id){

echo $id.'-'.$item.'<br/>';

},null,2,3);

输出后的内容为：

2-test

3-c++

4-python

8.5.5奇数输出

volist($array,function($item,$id){

echo $id.'-'.$item.'<br/>';

},null,0,null,2);

输出后的内容为：

0-php

2-test

4-python

8.5.6偶数输出

输出后的内容为：

1-java

3-c++

8.6 foreach循环输出

使用内置函数voeach可对数组进行foreach循环输出。voeach (循环输出函数,为空时输出函数)。

| **参数** | **参数类型** | **必须** | **默认值** |
| --- | --- | --- | --- |
| $array | array | 是 | 无 |
| $output | callable | 是 | 无 |
| $empty | callable | 否 | null |

8.6.1默认输出

输出数组全部如下所示：

namespace home\controller;

class Index {

public function index(){

view('test.php',['array'=>['java','php','test','c++','python']]);

}

}

//test.php

<?php

voeach($array,function($item){

echo $item.'<br/>';

};

输出后的内容为：

php

java

test

c++

python

8.6.2输出索引

voeach ($array,function($item,$id){

echo $id.'-'.$item.'<br/>';

};

输出后的内容为：

0-php

1-java

2-test

3-c++

4-python

8.6.3空数组输出

voeach ($array,function($item,$id){

echo $id.'-'.$item.'<br/>';

},function(){

echo '无数据<br/>';

});

数组不为空时输出：

0-php

1-java

2-test

3-c++

4-python

数组为空时输出为：

无数据

9上下文context

9.1实例化

Context类负责应用管理。

$context=Context::instance();

也可以使用助手函数

$context=context();

9.1获取本机IP

context()->ip();

9.2获取本机域名

context()->name();

9.3上下文参数

上下文参数用来在程序运行时进行传参，任何地方都能获取到该参数值

// 设置参数

context()->set(‘data’,‘hello world’);

// 获取参数,不存在时为null

context()->get(‘data’);

// 获取参数,设置默认值

context()->get(‘data’,‘hello world’);

不存在时为hello world

// 判断参数是否存在

context()->has(‘data’);

// 移除参数

context()->remove(‘data’);

9.4过滤器

过滤器用来对url进行过滤，匹配到符合规则的url会进行过滤操作，url匹配的是去除http://域名或IP后的url。

注册过滤器使用context的filter\_register方法来进行注册，注册的操作必须在控制器操作前执行。程序运行后会先进行过滤器操作。

// 移除参数

context()->register(‘.\*’,function(Request request){

// 执行过滤操作

});

上例的注册器会匹配任何的url，在程序运行前会执行函数内的操作。

10响应

10.1实例化

Respose类负责进行响应输出。

$response=Response::instance();

也可以使用助手函数

$response=response ();

10.2设置响应头

response()->header(‘Cache-Control: no-cache’);

10.3设置状态码

response()->code(404);

10.4重定向

response()->redirect($url,$code=302); // url可以为相对路径，状态码默认为302

11 Session

11.1实例化

Session类实例化。

$session=Session::instance();

也可以使用助手函数。

$session=session();

11.2 Session管理

session()->start(); // 启动会话

session()->close(); // 关闭会话

session()->destory(); // 销毁Session

session()->isAlive(); // 判断会话是否开启

session()->get(); // 获取session全部值

session()->get($key); //获取seesion某个值，不存在返回null

session()->get($key,$default); // 获取seesion某个值，不存在返回$default

session()->set($key,$value); // 设置session

session()->set($key,$value,$timeout); // 设置session，并设置超时时间

12 Cookie

12.1实例化

Cookie类实例化。

$cookie=Cookie::instance();

也可以使用助手函数。

$cookie=cookie();

12.2 Cookie管理

cookie()->get(); // 获取cookie全部值

cookie()->get($key); //获取cookie某个值，不存在返回null

cookie()->get($key,$default); // 获取cookie某个值，不存在返回$default

cookie()->set($key,$value,$time=null); // 设置cookie值与过期时间

13 文件上传

13.1文件上传配置

在app目录下的config中的返回数组中加入以下数组进行文件上传配置。

return [

'UPLOAD\_FILE'=>[

'TYPE'=>[

'.jpg','.jpeg','.gif','.bmp','.png', // 图片类型

'.swf','.flv','.mp4','.webm','.ogg','.mp3','.rmvb', //视频音频类型

'.zip','.doc','.docx','.xls','.xlsx','.rar','.pdf','.tar','.gz','.ppt','.pptx' //附件类型

], //不在列表中的文件类型均不解释

'MAX\_SIZE'=>10000000, //大小限制（10M）

'NAME\_RULE'=>'date', //文件命名规则，支持日期与md5（date,md5）

'PATH'=>'file' //上传文件保存根路径（相对于入口文件路径）

]

]

// 以上为框架默认配置，可自行在app目录下的配置文件中更改其配置

13.2获取上传文件

假设表单如下：

<form action="index.php?a=upload" method="post" enctype="multipart/form-data">

<input type="file" name="file"/>

<input type="button" value="上传"/>

</form>

获取上传文件如下所示：

$file=request()->file('file'); // 获取上传文件，会根据配置文件中的配置进行过滤，如果成功则返回UploadFile类的实例，如果失败则返回null。

echo $file->name(); // 获取上传文件名

echo $file->type(); // 获取上传文键类型

echo $file->tempName(); // 获取上传文件临时名称

echo $file->size(); // 获取上传文件大小

echo $file->suffix(); // 获取上传文件后缀名

13.3文件过滤器

13.3.1实例化

FileFilter是文件过滤类，用于对上传文件进行过滤。过滤规则默认采取配置文件中的过滤规则。

$filter=new FileFilter();

$file=request()->file('file',$filter); // 如果成功则返回UploadFile类的实例，如果失败则返回null。

// 上面和下面是等价的

$file=request()->file('file')

13.3.1更改文件类型

使用FileFilter类的changeType可以更改文件类型。

$filter=new FileFilter();

$filter->changeType([‘.jpg’,‘.png’,‘.jpeg’]); // 更改文件类型

$file=request()->file('file',$filter);

13.3.2增加文件类型

使用FileFilter类的addType可以添加文件类型。

$filter=new FileFilter();

$filter->addType(‘.gif’); // 添加一个文件类型

$filter->addType([‘.docx’,‘.doc’]); // 添加多个文件类型

$file=request()->file('file',$filter);

13.3.3移除文件类型

使用FileFilter类的removeType可以移除文件类型。

$filter=new FileFilter();

$filter->removeType(‘.gif’); // 移除一个文件类型

$filter->removeType([‘.docx’,‘.doc’]); // 移除多个文件类型

$file=request()->file('file',$filter);

13.3.4更改大小限制

使用FileFilter类的maxSize可以更改大小限制。

$filter=new FileFilter();

$filter->max(1000000); // 更改大小限制为1M

$file=request()->file('file',$filter);

13.3.5自定义文件过滤器

自定义文件过滤器只需要继承FileFilter，然后覆写setType与setSize方法。

class ImageFilter extends FileFilter{

public function setType(){

return [‘.png’,‘.jpg’, ‘.jpeg’, ‘.bpm’, ‘.gif’]; // 类型限制

}

public function setSize(){

return 1000000; // 大小限制为1M

}

}

13.4保存上传文件

13.4.1文件保存

在得到UploadFile的实例后，使用save方法可以保存上传文件，命名规则默认采用配置文件中配置的命名规则，如果为date，则根据日期和微秒数来成命名，如果为md5，则根据文件的md5值来命名。保存后会返回SplFileObject类，可以对文件进行后续操作,如果不需要后续操作，可不接受返回值。

$file=request()->file('file');

if($file!=null){

$file->save(); // 保存文件，默认保存在配置文件中的目录下,采用默认命名规则

}

// 或者

if($file!=null){

$fileObj=$file->save(); // 保存文件

…… // 后续操作

}

13.4.2更改命名规则

如果需要更改命名规则，可以使用rule方法进行更改。

$file=request()->file('file');

if($file!=null){

$file->rule(‘md5’)->save(); // 更改命名规则

}

13.4.3自定义命名

除了内置的命名规则，还可以手动对文件进行命名。

$file=request()->file('file');

if($file!=null){

$file->save(‘filename’); // filename为文件名称，不包含后缀，后缀名会根据上传文件的后缀名进行判断，保存后的文件名为filename.后缀

}

13.4.4更改保存目录

save方法除了手动命名外还可以指定保存目录。

$file=request()->file('file');

if($file!=null){

$file->save(‘dirname/’); // 假设配置文件中的根目录为file，那么最终保存的路径及文件名为：file/dirname/14751195154697.后缀

}

// 更改保存目录并指定命名

if($file!=null){

$file->save(‘dirname/filename’); // 更改保存目录并指定文件名称，最终保存的路径及文件名为：file/dirname/filename. 后缀

}

13.4.5回调操作

上传后若需要回调操作，例如在数据库中保存文件路径，可以在save方法中传入一个FileSuccess类的实例进行回调操作。

首先定义一个回调类，需要调用FileSucess接口：

class ImgageSuccess implements FileSucess{

//回调方法中会传入文件的路径

public function success($filename){

//回调操作

}

}

然后在save方法中传入该类的实例：

$file=request()->file('file');

if($file!=null){

$file->save(null,new ImageSuccess()); // 第一个参数为null则表示采用默认的命名规则进行命名

}

14 验证器

14.1基本用法

$validate = new Validate([

'name' => 'required|name|length:6,16', // 必须，名字类型，长度为6-16个字符

'email' => 'email' // email类型

]);

$data = [

'name' => 'tester',

'email' => 'tester@qq.com'

];

或者

$validate = new Validate([

'required|name|length:6,16', // 必须，名字类型，长度为6-16个字符

'email' // email类型

]);

$data = [

'tester',

'tester@qq.com'

];

验证规则的索引或键值和数据的索引或键值一致即可。

var\_dump($validate->check($data)); // 验证通过则返回true，未通过则返回false

14.2验证规则

$rules = [

'name' => 'required|name|length:6,16', // 必须，名字类型，长度为6-16个字符

'email' => 'email' // email类型

....... // 更多验证规则，验证规则为空则表示

];

$validate = new Validate($rules); // 实例化时传入规则

14.3错误信息

$rule = [

'name' => 'required|name|length:6,16',

'age' => 'num|range:1-100',

'email' => 'email'

];

$data = [

'name' => 'tester',

'age' => 101,

'email' => 'testerqq.com',

'address' => '地球',

];

$msg = [

'name' => '名字不能为空|名字不符合格式|长度必须为6-16个字符',

'age' => '年龄必须为数字|年龄必须在1-100之间',

'email' => '邮箱格式不正确',

];

$validate = new Validate($rule);

$result = $validate->check($data);

if(!$result){

echo $validate->getError();

}

会输出：年龄必须在1-100之间。

上述例子在其中一个字段验证未通过后，就会结束验证，并返回相应的错误信息。如果需要获取全部错误信息，则：

$validate = new Validate($rule);

$result = $validate->check($data,true); // 第二个参数为true，表示验证全部错误信息

if(!$result){

var\_dump($validate->getError());

}

会输出：array(‘age’=>年龄必须在1-100之间,’mail’=>’邮箱格式不正确’)。

14.4验证规则类

验证器除了数组传入，还可以传入验证规则类：

use core\ValidateRule;

class TestValiateRule implements ValidateRule{

public function getRule() {

return [

'name' => 'required|name|length:6,16',

'age' => 'num|range:1-100',

'email' => 'email'

];

}

public function getMsg() {

return [

'name' => '名字不能为空|名字不符合格式|长度必须为6-16个字符',

'age' => '年龄必须为数字|年龄必须在1-100之间',

'email' => '邮箱格式不正确',

];

}

}

$data = [

'name' => 'tester',

'age' => 101,

'email' => 'testerqq.com',

'address' => '地球',

];

$validate = new Validate(new TestValiateRule());

$result = $validate->check($data);

if(!$result){

echo $validate->getError();

}

14.5静态调用

Validate::checkData需要传入3个参数，分别为数据，验证规则和错误信息，其中错误信息非必须。

Validate::checkData('tester', 'required|name|length:6,16', '名字不能为空|名字不符合格式|长度必须为6-16个字符');

Validate::checkData(101,'num|range:1-100','年龄必须为数字|年龄必须在1-100之间');

Validate::checkData('tester@qq.com ','email','邮箱格式不正确');

// 验证通过则返回true，未通过返回错误信息

14.5内置规则

14.5.1非空验证

required

某个字段必须：

'name'=>'required'

14.5.2字段格式验证

email

邮箱格式：

'mail'=>'email'

name

姓名格式：

'name'=>'name'

qq

QQ号格式：

'qq'=>'qq'

phone

手机号格式：

'phone'=>'phone'

domain

域名格式：

'domain'=>'domain'

idCard

身份证号格式：

'idCard'=>'idCard'

enAndNum

必须为英文或者数字：

'loginname'=>'enAndNum'

enAndNumAndChs

必须为英文或者数字或者汉字：

'loginname'=>'enAndNumAndChs'

en

只能为英文字母：

'loginname'=>'en'

chs

只能为汉字：

'name'=>'chs'

num

只能为数字：

'num'=>'num'

port

端口号格式：

'port'=>'port'

ip

IP地址格式：

'ip'=>'ip'

url

url格式：

'url'=>'url'

date

日期格式：

'date'=>'date'

regex

正则表达式：

'name'=>'regex:#^[\x{4e00}-\x{9fa5}a-zA-Z]+$#u'

14.5.3字符验证

contain

是否包含某段字符：

'name'=>'contain:test' // 是否包含test字符，区分大小写

'name'=>'contain:test，false' // 是否包含test字符，不区分大小写

equals

是否等于某段字符：

'name'=>'equals:test' // 是否等于test，区分大小写

'name'=>'equals:test，false' // 是否等于test字符，不区分大小写

in

是否在某些字符或数字内：

'num'=>'in:1,2,3'

'name'=>'in:php,java,c'

notIn

是否不在某些字符或数字内：

'num'=>'notIn:1,2,3'

'name'=>'notIn:php,java,c'

14.5.4数字验证

gt

是否大于某个数字：

'num'=>'gt:20'

egt

是否大于等于某个数字

'num'=>'egt:20'

lt

是否小于某个数字：

'num'=>'lt:20'

lgt

是否小于等于某个数字

'num'=>'lgt:20'

range

数字是否在某个区间内：

'num'=>'range:20,0'

14.5.5字符长度验证

length

字符长度：

'name'=>'length:6,16' // 长度在6-16个字符之间

'name'=>'length:6' // 长度在6个字符

max

字符最大值：

'name'=>'max:16'

min

字符最小值：

'name'=>'min:16'

14.5.6日期范围验证

before

在某个日期之前：

'birthday'=>'brfore:2000-01-01'

after

在某个日期之后：

'birthday'=>'after:2000-01-01'

dateRange

在某两个日期之间：

'birthday'=>'range:2000-01-01，2001-01-01'

14.5.6 IP范围验证

ipRange

IP在某个范围内

'ip'=>'ipRange:211.69.32.0，211.69.32.100'

ipIn

IP在某个网段内

'ip'=>'ipIn:211.69.32.0/24'

14.6规则扩展

扩展规则需要继承validate类，并且覆写geClassName方法：

use core\Validate

class TestValiate extends Validate{

public static function getClassName(){

return \_\_CLASS\_\_;

}

public static function isMinus ($data){ // 是否为负数

if($data>0){

return false;

}else{

return true;

}

}

public static function testArg($data,$arg){

// $arg为参数

// 进行验证

}

public static function testArgs($data,$arg){

$args=self::getArgs($arg); // 获取多个参数

// $args为参数数组

// 进行验证

}

$validate = new TestValidate([

'num' =>'isMinus'

]);

$data = [

'num' => -1

];

var\_dump($validate->check($data));

// 或者

var\_dump(TestValidate::checkData(-1,'isMinus'))

已更改内容：

1、兼容到php5.3

2、数据库where与condition操作。

3、config函数

待更新内容

1、数据库操作升级，支持多个高级查询

2、验证器升级，支持&与|操作，正则兼容，验证场景

数据库操作类的修改

validate类的修改

FileFilter.类的修改

Context类的修改