# 1，Smarty视图模板引擎

|  |  |  |
| --- | --- | --- |
| **Smarty简单配置**：  /\*  \* 实例化smarty对象  \*/  $smarty=**new** Smarty();  /\*  \* 配置smarty  \*/  $smarty->left\_delimiter="{";//左定界符  $smarty->right\_delimiter="}";//右定界符。模板中定界符内的内容会被交给smarty处理  $smarty->setTemplateDir("tpl");//html模板文件目录  $smarty->setCompileDir("templates\_c");//模板生成的文件目录  $smarty->setCacheDir("cache");//缓存目录  // //开启缓存，一般不使用smarty的缓存，仅测试  // $smarty->caching=true;  // $smarty->cache\_lifetime=120;//缓存时间  **Smarty最重要方法**：  $smarty->assign("title","文章标题");//给模板中元素赋值  $smarty->display("test.tpl");//显示模板生成内容 | **变量调节器**： | **if语句**：    **selection循环语句**： |
| **Foreach循环语句：** | **Smarty文件引用：**    **Smarty类的使用：**    模板中使用对象名->方法名的方式进行调用 | **Smarty函数使用**：    **Smarty插件**： |
| 修饰插件：注册时选择modifier，与函数的区别是在模板中调用的格式不一样，其他一致。  区块函数插件：调用格式与函数相同，多了一个区块结束符，并且区块中的内容也会作为参数进入函数处理：  定义： | 模板调用区块函数： |  |

# 2，ThinkPHP框架

## 2.1 杂项

具体的ThinkPHP参考手册请点击以下链接，以下仅为简单零散知识点 [ThinkPHP5.0参考手册](http://www.kancloud.cn/manual/thinkphp5)

|  |  |  |
| --- | --- | --- |
|  | **ThinkPHP框架**：    **安装**  在application目录下创建build.php，定义模块的基本信息，执行**php console build**就自动生成目录结构了。 | **访问URL**：  默认方式：需开启pathinfo  http://**serverName/index.php（或者其它应用入口文件）/模块/控制器/操作/[参数名/参数值...]**  兼容模式：  http**://serverName/index.php（或者其它应用入口文件）?s=/模块/控制器/操作/[参数名/参数值...]**  不支持传统的url文件路径方式访问。URL模式不区分模块/控制器等的大小写，打开大小写限制：application/config.php中：  // 关闭控制器名的自动转换  'url\_controller\_convert' => false,  // 关闭操作名的自动转换  'url\_action\_convert' => false, |
| **控制器数据输出**：  控制器输出统一使用response的方式，如下：  $data = ['name'=>'thinkphp','url'=>'thinkphp.cn'];  return ['data'=>$data,'code'=>1,'message'=>'操作完成'];  在config.php中可以设置以下值设置返回格式：  'default\_return\_type'=>'json'，可取值有json，html，text，jsonp | **TP配置：**  所有可以使用的配置项在**thinkphp/convention.php**文件中，并且已经有默认配置，可以选择需要的配置项放入config.php进行配置。  配置格式  return [  '配置参数1'=>'配置值',  '配置参数1'=>'配置值',  // ... 更多配置  ];  如果使用其他格式可以进行转换：  Config::parse(APP\_PATH.'my\_config.ini','ini');  Config::parse(APP\_PATH.'my\_config.xml','xml'); | 读取配置：  echo Config::get('配置参数1');  echo C('配置参数1');  动态配置：  Config::set('配置参数','配置值');  C('配置参数','配置值'); |
| **输入变量检测：**  系统提供了\think\Input类来完成全局输入变量的检测、获取和安全过滤，支持包括$\_GET、$\_POST、$\_REQUEST、$\_SERVER、$\_SESSION、$\_COOKIE、$\_ENV等系统变量，以及文件上传信息。主要用于检测和安全过滤：  （1）全局静态配置  'default\_filter'=> ['strip\_tags','htmlspecialchars'],  （2）全局动态配置：  Input::$filters = ['strip\_tags','htmlspecialchars'];  （3）获取变量时过滤：  Input::get('name','','htmlspecialchars'); // 获取get变量 并用htmlspecialchars函数过滤 | **Session类：**  （1）session初始化或配置文件设置  Session::init([  'prefix' => 'module',//不同module session不同  'type' => '',  'auto\_start' => true, //是否自动启动  ]);  （2）存取数据  Session::set('name','thinkphp',['think']); // 赋值(think作用域)  // 判断think作用域下面是否赋值  Session::has('name',['think']); // 判断(think作用域)下面是否赋值  Session::get('name',['think']); // 取值(think作用域)  Session::prefix('think'); // 指定当前作用域  Session::delete('name',['think']); // 删除(think作用域)下面的值  Session::clear('think'); // 清除think作用域session | **Cookie类**：  （1）初始化  // cookie初始化  Cookie::init([配置参数]); 支持的参数如右边  （2）存取数据  Set,get,delete,clear方法是用和session一致(不支持作用域) |

## 2.2 路由配置

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **URL路由模式**：  简化URL访问地址，并根据定义的路由类型做出正确的解析。首先需要在config.php中进行配置：   |  |  | | --- | --- | | 路由和默认模式并存：  'url\_route\_on' => true,  'url\_route\_must'=> false, | 只允许路由模式：  'url\_route\_on' => true,  'url\_route\_must'=> true, |   **注册路由规则**：  （1）动态注册：方法类型有get,post,put,delete,\*  Route::register('new/:id','New/update','POST'); //注册到New控制器的update操作，并且以“:”开头的参数都表示动态变量，并且会自动对应一个GET参数  （2）批量注册：  \think\Route::register([  '路由规则1'=>'路由地址和参数',  '路由规则2'=>['路由地址和参数','匹配参数（数组）','变量规则（数组）']  ...  ],'','请求类型','**匹配参数**（数组）','**变量规则'**);  （3）路由配置文件：route.php  return [  'new/:id' => 'New/read',  'blog/:id' => ['Blog/update',['method' => 'post|put'], ['id' => '\d+']],  ] ;  （4）直接指定操作方法：不去调用控制器类方法  Route::get('hello/:name',function($name){  return 'Hello,'.$name;  }); | **变量规则**：限定变量的类型  （1）全局：  Route::pattern([  'name' => '\w+', //字符串  'id' => '\d+', //数字  ]);  （2）局部：在配置路由规则时附带  'blog/:id' => ['Blog/update',['method' => 'post|put'], [**'id' => '\d+']],**  **路由匹配参数**： | **路由地址格式**：    **域名路由**：可以代替虚拟主机，不同域名指向不同的地址  Config.php: 'url\_domain\_deploy' => true  Route.php:  return [  '\_\_domain\_\_'=>[  'blog' => 'blog',  // 泛域名规则建议在最后定义  '\*.user' => 'user',  '\*' => 'book',  ],  // 下面是路由规则定义  ] |

## 2.3 控制器

|  |  |  |
| --- | --- | --- |
| **控制器内跳转和重定向**：  use \think\Controller; //使用success或error跳转都需要继承controller类  class Index **extends \think\Controller //必须继承controller类**  {  public function hello($name){  //return $this->success($name.",你找到我了","shidandan");//跳转或error  //return $this->redirect('index/index/index');//重定向到其他控制器或模块  return $this->redirect('https://cn.bing.com');//重定向到外部网页  }  } | **空操作**：url为不存在的操作时调用  public function \_empty($name){//用于操作不存在的时候调用  return $this->hello($name);  } | **空控制器**：定义Error控制器类在控制器不存在时调用，Error名可以config中修改  class Error  {  public function index()  {  return '没有此控制器';  }  } |

## 2.4 模型

|  |  |
| --- | --- |
| **数据库连接**：  ThinkPHP内置了抽象数据库访问层，把不同的数据库操作封装起来，我们只需要在database.php配置数据库连接参数，使用**公共的Db类**进行操作。  配置了数据库连接信息后，我们就可以直接使用数据库运行原生SQL操作了，支持**query（查询操作）和execute（写入操作）**方法，并且支持参数绑定。  Db::query('select \* from think\_user where id=?',[8]);  Db::execute('insert into think\_user (id, name) values (?, ?)',[8,'thinkphp']);  **Model属性:** 通常一个model对应一个数据表 | **Model实例化**：  $User = \think\Loader::model('admin/User'); 或  $User = D('admin/User');  // 相当于 $User = new \app\admin\model\User(); |

## 2.5 视图和模板

|  |  |  |
| --- | --- | --- |
| 视图功能由\think\View类配合视图驱动（模板引擎）类一起完成，目前的内置模板引擎包含PHP原生模板和Think模板引擎  **视图实例化**：  （1）模板输出  // 实例化视图类  $view = new \think\View();  // 渲染模板输出 并赋值模板变量  return $view->fetch('hello',['name'=>'thinkphp']); //hello模板的路径应该是**view/控制器名/操作名.html**  （2）内容输出  $view = new \think\View();  return $view->show($content,$vars); //直接解析内容而不通过模板文件  **模板引擎**：  模板主要参数如下，可以在实例化view时赋值传入： | **模板赋值**：   1. 调用fetch方法时传入 2. Assign方法赋值   // 实例化视图类  $view = new \think\View();  $view->assign([  'name'=>'ThinkPHP',  'email'=>'thinkphp@qq.com'  ]);  // 模板输出  return $view->fetch('index');  （3） view对象直接赋值  // 实例化视图类  $view = new \think\View();  $view->name = 'ThinkPHP';  $view->email= 'thinkphp@qq.com';  // 模板输出  return $view->fetch('index');  **模板：**  ThinkPHP内置了一个基于XML的性能卓越的模板引擎，这是一个专门为ThinkPHP服务的内置模板引擎，使用了XML标签库技术的编译型模板引擎，支持两种类型的模板标签，使用了动态编译和缓存技术，而且支持自定义标签库   1. 普通标签   普通标签用于变量输出和模板注释，普通模板标签默认以**{ 和 }** 作为开始和结束标识   1. 标签库标签   标签库标签可以用于模板变量输出、文件包含、条件控制、循环输出等功能，而且完全可以自己扩展功能。普通模板标签默认以**{ 和 }** 作为开始和结束标识 | **模板示例**：  View端：  <table align="center" width="70%" border="1"  <tr>  <td><b>管理员编号</b></td>  <td><b>管理员名称</b></td>  <td><b>管理员邮箱</b></td>  </tr>  {volist name="result" id="row"} //volist用于循环输出，类似foreach  <tr>  <td align="center">{$row.id}</td>  <td>{$row.username}</td>  <td>{$row.email}</td>  </tr>  {/volist}  </table>  控制器端：  $user=D("barret/User");  $result=$user->admin();  $view=new \think\View();  $view->assign("result",$result); //assign赋值给array  return $view->fetch("users");//模板渲染输出  **系统变量输出**：不需要在控制器中赋值 |
| **原样输出**：防止被模板引擎解析  可以使用literal标签来防止模板标签被解析，例如：  {literal}  Hello,{$name}！ //{$name}会原样输出，不会被解析成php变量  {/literal}  **模板注释**：  {/\* 注释内容 \*/ } 或 {// 注释内容 }  **模板布局配置**：选择合适的模板文件  （1）全局模板：整个项目用同一个样式的模板，config.php  'template' => [  'layout\_on'=>true, //开启全局模板  'layout\_name'=>'layout', //全局模板文件名  ]  全局模板示例：  {include file="public/header" /}  {\_\_CONTENT\_\_} //这里会填充fetch时指定的模板内容  {include file="public/footer" /}  （2）模板布局标签：不配置config.php  {layout name="layout" /} //直接在模板中声明要使用的模板布局   1. 控制器中指定模板布局   $view->engine->layout('Layout/newlayout');  // 临时关闭当前模板的布局功能  $view->engine->layout(false); | **标签库：**  内置标签库无需导入即可使用，并且不需要加XML中的标签库前缀，ThinkPHP内置的标签库是Cx标签库。 | 比较标签  调用格式：  {比较标签 name="变量" value="值"}  内容  {/比较标签} |

## 2.6 验证器和缓存

|  |  |  |
| --- | --- | --- |
| **Validate验证器类使用**：  $validate = new Validate();  $rule = [  **'name' => ['require','max'=>25],**  **'email' => ['email']**  ];  $data = [  'name'=>'thinkphp',  'email'=>'thinkphp@qq.com'  ];  if(!$validate->check($data,$rule)){  dump($validate->getError());  }  可以通过validate的rule方法添加新的规则(类似require/email等)，设置正则表达式 | **验证规则和错误信息**：  验证规则的另一种格式，并且可以附带验证失败时的提示信息：  $rule = [  ['name','require|max:25','名称必须|名称最多不能超过25个字符'],  ['age','number|between:1,120','年龄必须是数字|年龄必须在1~120之间'],  ['email','email','邮箱格式错误']  ];  **模型验证器**：  Model中有validate方法，可以直接调用该方法进行验证，不需要定义validate类了。 | **缓存简单实用：**  $options = [  'type' => 'File', // 缓存类型为File  'expire'=> 0, // 缓存有效期为永久有效  'length'=> 3, // 缓存队列长度  'path' => APP\_PATH.'Runtime/cache/', // 指定缓存目录  ];  S($options); // 缓存初始化  S('name',$value,3600); // 设置缓存数据  var\_dump(S('name')); // 获取缓存数据  S('name',NULL); // 删除缓存数据 |

# 3，YII框架

## 3.1 简介入门

|  |  |  |
| --- | --- | --- |
| Yii 是一个通用的 Web 编程框架，即可以用于开发各种用 PHP 构建的 Web 应用。因为基于组件的框架结构和设计精巧的缓存支持，它特别适合开发大型应用，如门户网站、社区、内容管理系统（CMS）、电子商务项目和 RESTful Web 服务等  Yii主要目录结构：    **URL访问路径**：  https://www.barretren.com/ /YII/web/index.php?**r=控制器名/操作名[&参数名=参数值]** | **控制器命名规则**：  继承Controller，类名为**控制器名Controller**，操作名为**action操作名**，如下：  namespace app\controllers;  class HelloController extends Controller{  //接收用户输入，输出为内容，默认输出hello world  public function actionHello($msg="hello world"){//action 前缀后面的名称被映射为操作的 ID  //渲染输出到view，附带传递变量,会自动应用views/layouts下的mian布局  return $this->render("hello",['message'=>$msg]);  }  }  **视图命名规则**：  视图为php文件，名称要与render中指定的一致，存放路径为**views/控制器名/视图名.php**  **模型命名规则**：  继承model（普通模型）或activeRecord（数据包关联），成员名可与form表单名称一致，数据会自动填充： | //继承model，普通数据模型  class EntryForm extends Model{  //数据成员，与form表单同名  public $name;  public $email;  //设置数据验证规则，验证要调用validate方法  public function rules(){  return [  [['name','email'],'required'],  ['email','email']  ];}}  Model中数据被自动填充，控制器中就不需要再获取post参数了：  public function actionEntry(){  $model=new EntryForm();  if ($model->load(\Yii::$app->request->post())  && $model->validate()){ //直接验证，不再填充数据  return $this->render("entry-confirm",['model'=>$model]);  }else{  return $this->render("entry",['model'=>$model]);  }} |
| **ActiveRecord模型与数据库关联**：  （1）Config/db.php中设置数据库信息：  return [  'class' => 'yii\db\Connection',  'dsn' => 'mysql:host=localhost;dbname=test',  'username' => 'barret',  'password' => '1230',  'charset' => 'utf8',  ];  （2）模型继承ActiveRecord，类名与数据包名一致  //框架会根据类型自动匹配数据表名  class Country extends ActiveRecord{ }  （3）控制器中直接调用模型的静态方法操作数据库  // 获取 country 表的所有行并以 name 排序  $countries = Country::find()->orderBy('name')->all();  // 获取主键为 “US” 的行  $country = Country::findOne('US');  // 输出 “United States”  echo $country->name;  // 修改 name 为 “U.S.A.” 并在数据库中保存更改  $country->name = 'U.S.A.';  $country->save(); | **Yii自带分页插件：yii\data\Pagination**  （1）分页类初始化  $pagination = new Pagination([  'defaultPageSize' => 5, //每页记录数  'totalCount' => $query->count(), //总记录数  ]);  （2）根据分页类提供的offset和limit查询数据库获取数据  $countries Country::find()->orderBy('name')  ->offset($pagination->offset)  ->limit($pagination->limit)  ->all();  （3）渲染模板时传递数据和分页类实例（用于分页选择链接）  return $this->render('country', [  'countries' => $countries, //数据  'pagination' => $pagination, //类实例  ]);  （4）视图中显示数据，并通过插件显示分页选择链接  use yii\widgets\LinkPager;  echo LinkPager::widget(['pagination' => $pagination]);  效果： | **Session处理**:    **Cookie处理**： |

## 3.2 视图和模型

|  |  |  |
| --- | --- | --- |
| **控制器渲染视图的两种方式**：    **在一个视图中显示另一个视图**：    ==========================================================  **activeRecoed操作数据表的方法**：  （1）where条件查询    （2）数据删除 | （3）添加数据：直接创建模型类赋值保存即可    （4）修改数据：    （5）多表关联查询：**hasMany/hasOne**设置关联关系并获取数据    **关联查询缓存**：使用完unset下，下次才会从新从数据库获取数据  **关联查询多次查询**：使用with降低查询次数 |  |

## 3.3 延迟加载和缓存

|  |  |  |
| --- | --- | --- |
| **类延迟加载**：使用时才导入文件，之前不报错。YII自身就是用的这种技术。延迟加载的示例：    **类映射表**：指定某个类所在的类文件路径，更快的加载类，提高性能  示例： | **数据缓存**：  （1）缓存的基本使用    （2）缓存有效期：单位秒 | 片段依赖： |