# **1.ThinkPHP 的安装和设置**

前提：此教程适用于ThinkPHP 3.2+

## **一、实验介绍**

### **1.1 实验内容**

本节实验课程主要学习ThinkPHP的安装与配置。

### **1.2 实验知识点**

* ThinkPHP 基础
* ThinkPHP 安装

## **二、实验步骤**

### **2.1安装**

实验楼内下载地址：

http://labfile.oss.aliyuncs.com/courses/352/ThinkPHP.zip

下载完成之后，用zip命令行解压：

cd /home/shiyanlou

wget http://labfile.oss.aliyuncs.com/courses/352/ThinkPHP.zip

unzip ThinkPHP.zip

解压后会得到一个叫ThinkPHP的文件夹，里面包含了ThinkPHP相关的程序文件以及创建数据表所需的sql文件：

cd /home/shiyanlou/ThinkPHP

ls -l

启动mysql服务，准备好导入数据表：

sudo service mysql start

默认情况下登录mysql不需要输入密码：

mysql -u root

创建数据库think，在mysql提示符下执行：

create database think;

切换到think数据库：

use think;

导入数据：

source /home/shiyanlou/ThinkPHP/think.sql;

退出mysql提示符状态只需要输入quit回车，或按组合键Ctrl + D即可

以上几个安装步骤也可以合并起来，通过一句话脚本来完成：

curl -s http://labfile.oss.aliyuncs.com/courses/352/install\_thinkphp.sh | sh

至此ThinkPHP的安装就已经完成了，且前期数据准备也完毕了，正式进入课程。

****以下的安装方式来自于官方文档，作为用户本地搭建环境的参考（实验楼已配置环境，无需操作）****

安装ThinkPHP的方法很多，你可以直接到[ThinkPHP的官网](http://www.thinkphp.cn/down/framework.html" \t "https://www.shiyanlou.com/courses/96/labs/1184/_blank)去下载，下载完之后直接解压就可以了；或者你也可以到ThinkPHP官方维护的Git地址下载

1. Github： [https://github.com/liu21st/thinkphp](https://github.com/liu21st/thinkphp" \t "https://www.shiyanlou.com/courses/96/labs/1184/_blank)
2. Oschina： [http://git.oschina.net/liu21st/thinkphp.git](http://git.oschina.net/liu21st/thinkphp.git" \t "https://www.shiyanlou.com/courses/96/labs/1184/_blank)
3. Code： [https://code.csdn.net/topthink2011/ThinkPHP](https://code.csdn.net/topthink2011/ThinkPHP" \t "https://www.shiyanlou.com/courses/96/labs/1184/_blank)

当然，作为一个流行的PHP框架，你也可以直接用composer来安装：

composer create-project topthink/thinkphp your-project-name

### **2.2 设置**

我们现在开始学习ThinkPHP第一步：设置ThinkPHP。一个框架的原始模样可能无法满足你的开发需求，但你可以通过设置来满足。在学习ThinkPHP的配置的时候，首先要明白：ThinkPHP框架中所有配置文件的定义格式都是采用返回PHP数组的方式来定义的。

我们最常操作的是应用配置，默认的就是在Application/Common/Conf/config.php文件中：

<?php

return array(

'URL\_ROUTER\_ON' => true,

'URL\_ROUTE\_RULES'=>array(

'blogs/:id' => array('Index/read'),

'article/:id' => array('Article/show')

),

'URL\_MAP\_RULES'=>array(

'new/top' => 'Index/top?type=top'

),

'DB\_TYPE' => 'mysql',

'DB\_HOST' => 'localhost',

'DB\_NAME' => 'think',

'DB\_USER' => 'root',

'DB\_PWD' => '',

'DB\_PORT' => '3306',

'DB\_PREFIX' => 'think\_',

);

说明：ThinkPHP的配置参数（一级参数）是不区分大小写的，因为不管大写小写，最后都会转为小写。但是为了在编程的过程中更符合规范，建议统一使用大写来设置配置参数。上面的第一个配置URL\_ROUTER\_ON，我们开启了路由重写功能，为后面的 URL\_ROUTE\_RULES 奠定基础（详细的后面我们会在路由章节说到）。最后几个带DB\_的设置项是表示设置连接数据库的参数，几乎每一个web应用都会用到数据库，这些设置为我们后续进一步学习的基础。

<?php

return array(

'USER\_CONFIG' => array(

'USER\_AUTH' => true,

'USER\_TYPE' => 2,

),

);

如上面的USER\_CONFIG下的USER\_AUTH和USER\_TYPE是区分大小写的。

这个只是示例，如果在项目中真的需要二级配置，请区分大小写，这些配置也是在Application/Common/Conf/config.php中配置的。

了解了ThinkPHP的配置格式后，我们再看看ThinkPHP的配置加载顺序，理解配置项的加载顺序在开发的时候很重要，因为在同名的配置下，后加载的配置会覆盖前面加载的顺序，而生效的只有后加载的顺序。

惯例配置->应用配置->模式配置->调试配置->状态配置->模块配置->扩展配置->动态配置

上面的顺序就是ThinkPHP的配置加载顺序，而在一般情况下，这些配置都是自动加载的。我们最常操作的是应用配置，默认的就是在Application/Common/Conf/config.php文件中。在开发的时候我们可以在这里设置自己的配置，如果你不熟悉你可以配置什么值，你可以打开ThinkPHP/Conf/convention.php文件来查看相对应的配置项

### **2.3 读取配置**

在开发的过程中，我们有时候需要读取应用的配置值，在ThinkPHP中统一使用C('配置参数名')来读取配置。比如：

$model = C('URL\_MODEL');

或者

$model = C('url\_model');

这两个是等效的，都是可以读取到系统的URL访问模式的设置值，因为在ThinkPHP的配置项是不分大小写的。但是建议统一使用大写方式。

可以利用config首字母来记忆C()方法。

## **三、课后习题**

查看ThinkPHP/Conf/convention.php的内容，对其中的配置项进行初步了解

## **四、参考文档：**

本实验课程基于ThinkPHP的官方文档：

[http://document.thinkphp.cn/manual\_3\_2.html#start](http://document.thinkphp.cn/manual_3_2.html" \l "start" \t "https://www.shiyanlou.com/courses/96/labs/1184/_blank)

# **2.ThinkPHP 路由**

## **一、实验介绍**

### **1.1 实验内容**

本节实验课程主要学习ThinkPHP的路由的定义和规则路由，正则路由和静态路由的用法。

### **1.2 实验知识点**

* ThinkPHP 基础
* ThinkPHP 路由

### **1.3 实验环境**

本实验与前一节具有连续性，如果是直接开始本实验，则需要对实验环境进行初始化，部署安装ThinkPHP。直接执行初始化脚本：

curl -s http://labfile.oss.aliyuncs.com/courses/352/install\_thinkphp.sh | sh -

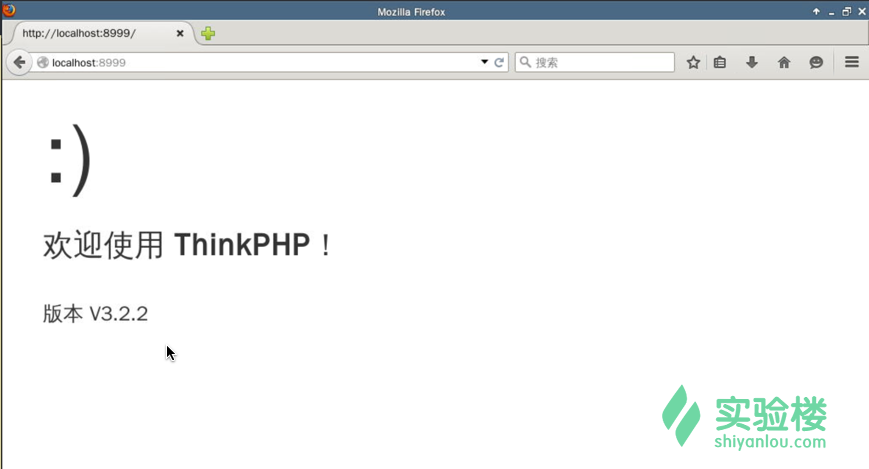
## **二、 路由 (Routes)**

有了基本配置，我们就可以来访问我们的应用默认首页了。进入到项目目录，可以直接使用PHP内置服务器来开始访问，比如：

cd /home/shiyanlou/ThinkPHP

php -S localhost:8999

浏览器输入localhost:8999就可以看到ThinkPHP的默认首页了：一个笑脸。



在这里，我们访问到的是ThinkPHP自带的默认入口文件index.php也就是访问到的是IndexController的index()方法，这是因为ThinkPHP默认设置：

'DEFAULT\_CONTROLLER' => 'Index'

如果你查看过ThinkPHP/Conf/convention.php文件，应该就会明白这个其实就是设置默认的控制器。

关于控制器(Controller)我们后面会仔细说，控制器的文件位于Application/Home/Controller 文件夹之下

了解这些基本知识之后，那么如果我们需要访问其它的页面，访问其他的控制器和方法呢？答案就在本节的路由教程中。

### **2.1 路由定义规则**

在使用路由之前，确保你的URL支持PATH\_INFO（或者兼容URL模式也可以，采用普通URL模式的情况下不支持路由功能）并且确认已开启一下的路由设置：

'URL\_ROUTER\_ON' => true

这里涉及到两个设置项，PATH\_INFO和URL\_ROUTER\_ON，这些在ThinkPHP/Conf/convention.php文件都可以找到。





在满足以上两个条件之后，就可以配置路由规则了。在配置文件中使用URL\_ROUTE\_RULES参数进行配置，配置格式是一个数组，其格式为： '路由表达式'=>'路由地址和传入参数'每个元素都代表一个路由规则，比如：

'URL\_ROUTE\_RULES'=>array(

'blogs/:year/:month/:day' => array('Index/archive', 'status=1'),

'blogs/:id' => 'Index/read',

),

如图：



ThinkPHP按定义的顺序依次匹配路由规则，一旦匹配到的话，就会定位到路由定义中的控制器和操作方法去执行（你可以传入其他的参数），而后面的规则不会继续匹配

以上的路由配置说明：在每个路由表达式中，:后面跟参数名称，比如上面的:year,:month,:id都是参数名称，以:id为例，它指向Index控制器的read方法，这个方法接受一个$id的参数：

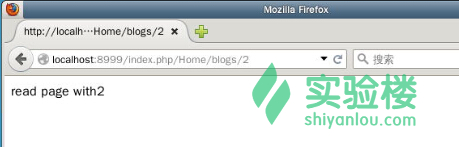
public function read($id){

echo "read page with" .$id;

}

你需要在IndexController中添加上面的代码，IndexController位于Application/Home/Controller之下

在浏览器输入http://localhost:8999/index.php/Home/blogs/2就可以看到



Home就代表Home模块，你可以简单地将它映射到相应的Home目录，这是由于在默认的配置中

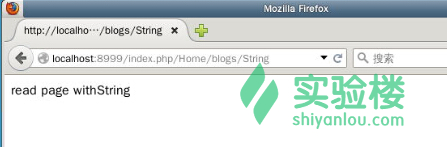
'DEFAULT\_MODULE' => 'Home'

你可以根据自己的需求修改，但本课依旧采用默认的Home模块.

如果你还需要传人额外的参数，像第一条的规则array('Index/archive', 'status=1')中的status一样传入，你看设置多个这样的参数。

如果你尝试在浏览器输入：

http://localhost:8999/index.php/Home/blogs/string



ThinkPHP也给我们返回了string，但在日常的开发中，我们通常需要限制:id变量是整数，那该怎么做呢？只需要稍稍改动就可以了，写成

'blogs/:id\d' => 'Index/read',

以上\d表示限制变量id只能是数字。

对于可选参数，可以用[]包含表示，比如：

'blogs/:year/:month/[:day]' => array('Index/archive', 'status=1'),

上面的day现在就是可选参数了，你可以传入，也可以不传。

在ThinkPHP中，还支持在限制路由的后缀和使用正则路由。

限制路由后缀，通常使用在平时常见的html,htm等后缀，还是以上面的规则为例：

'blogs/:id' => array('Index/read',array('ext'=>'html'))

你就可以限制这条规则只能在.html的路由后缀生效。

### **2.2 正则路由**

正则本身就是一门很大的学问，在学习ThinkPHP的正则路由之前，最好是具备一定的正则表达式的基础。

路由表达式支持的正则定义必须以/开头，否则就视为规则表达式，比如：

'#^blog\/(\d+)$#' => 'Index/read'

这会解析为规则路由而不是正则路由，因为路由表达式并没有以/开始，所以，我们需要这样写：

'/^new\/(\d{4})\/(\d{2})$/' => 'Index/achive?year=:1&month=:2',

以上就是一条正确的正则路由。对于正则表达式中的每个正则规则子模式）部分(如\d{4}和\d{2})，如果需要在后面的路由地址中引用，可以采用:1、:2这样的方式，序号就是子模式的序号

### **2.3 静态路由**

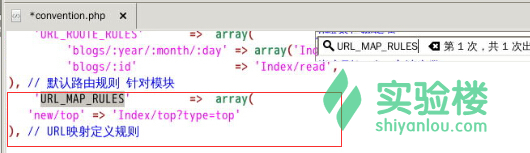
ThinkPHP框架其实还有一个路由机制叫静态路由，这实际上就是规则路由的静态简化版，路由定义中不包含动态参数(如上面的路由规则中id参数)，静态路由不需要遍历路由规则而是直接定位，因此执行效率会较高。静态路由采用URL\_MAP\_RULES来定义规则：

'URL\_ROUTER\_ON' => true,'URL\_MAP\_RULES'=>array(

'new/top' => 'Index/top?type=top'

)

如图：



由于Index/top?type=top中Index表示控制器，第一个top表示方法，所以我们需要在Index控制器中创建top()方法：

public function top(){

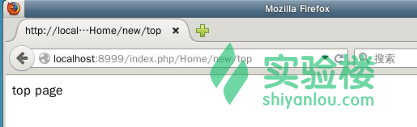
echo "top page </br>";

}

你需要在IndexController中添加上面的代码，IndexController位于Application/Home/Controller之下

根据上面这条规则，如果我们访问到

http://localhost:8999/index.php/Home/new/top



其实我们访问的是：

http://localhost:8999/index.php/Home/index/top/type/top

转译成就是new/top对应的是Index控制器的top()方法，传人的参数为type，参数值为top，所以就有了index/top/type/top

但是，当我们访问http://localhost:8999/index.php/Home/new/top/var/test尽管URL地址前面也有new/top，然而由于静态路由是完整匹配的性质，所以不会匹配到index/top/type/top

## **三、课后习题**

'URL\_ROUTER\_ON' => true, 'URL\_ROUTE\_RULES' => array(

'new/:id\d' => 'Index/read',

'new/:name' => 'Index/read',

),

在路由配置如上的情况下，修改IndexController的read()方法，使得read()方法不再绑定参数，使用$\_GET方式实现访问下面的两个地址都可以正确获取相应的返回结果：

http://localhost:8999/index.php/Home/new/8

http://localhost:8999/index.php/Home/new/hello

## **四、参考文档：**

本试验课程基于ThinkPHP官方文档的路由篇，更多知识请查看：

[http://document.thinkphp.cn/manual\_3\_2.html#route\_define](http://document.thinkphp.cn/manual_3_2.html" \l "route_define" \t "https://www.shiyanlou.com/courses/96/labs/1185/_blank)

# **3.ThinkPHP 控制器**

## **一、实验介绍**

### **1.1 实验内容**

本节实验课程主要学习ThinkPHP的控制器的知识，包括定义，伪静态实现，跳转和重定向，前置与后置操作等

### **1.2 实验知识点**

* ThinkPHP 基础
* ThinkPHP 控制器

### **1.3实验环境**

本实验与前一节具有连续性，如果是直接开始本实验，则需要对实验环境进行初始化，部署安装ThinkPHP。直接执行初始化脚本：

curl -s http://labfile.oss.aliyuncs.com/courses/352/install\_thinkphp.sh | sh -

## **二、 控制器 (Controller)**

在上一课程中，你可能会对ThinkPHP的路由会有一丝丝疑惑，不过没关系，学完本课程，很多事都会豁然开朗。

控制器文件命名遵守IndexController.class.php的方式

### **2.1 控制器的定义**

在开始之前，我们还是需要明确一下控制器的定义：

<?phpnamespace Home\Controller;use Think\Controller;class IndexController extends Controller {

public function read($id){

echo "read page with </br>" .$id;

}

public function top(){

echo "top page </br>";

}

}

如所见，前面在路由篇提到的控制器就是这么定义的：

1. 使用相应的命名空间，默认是namespace Home\Controller
2. 加载Think\Controller
3. 新建控制器继承于Controller（或子类）
4. 采用驼峰命名法，注意首字母大写

控制器内的公共方法可以看作一个操作，比如上面的read()和top()方法就可以看作操作，我们在路由篇的时候都验证过了。

http://localhost:8999/index.php/Home/Index/top

就是访问到top()方法，会在页面上打印出top page ，再次明确Home代表的是Home模块

有时候可能会遇到有和系统的关键字冲突的方法，这时候就可以使用设置操作方法后缀来解决了，具体请看官方文档：

[http://document.thinkphp.cn/manual\_3\_2.html#define\_controller](http://document.thinkphp.cn/manual_3_2.html" \l "define_controller" \t "https://www.shiyanlou.com/courses/96/labs/1186/_blank)

### **2.2 前置和后置操作**

前置和后置操作指的是在执行某个操作方法之前和之后会自动调用的方法，不过仅对访问控制器有效,如在IndexController中为top()方法添加前置后置方法：

public function \_before\_top(){

echo "before top page </br>";

}

public function top(){

echo "top page </br>";

}

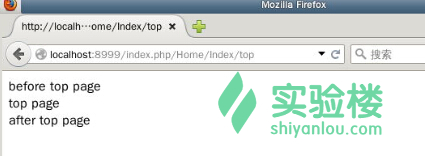
public function \_after\_top(){

echo "after top page </br>";

}

访问：[http://localhost:8999/index.php/Home/Index/top](http://localhost:8999/index.php/Home/Index/top" \t "https://www.shiyanlou.com/courses/96/labs/1186/_blank)

就会看到：



****使用前置和后置操作要注意如下两点：****

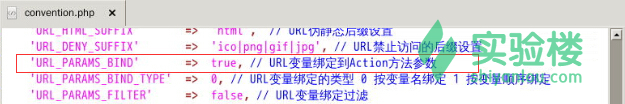
如果当前的操作并没有定义操作方法，而是直接渲染模板文件，那么如果定义了前置和后置方法的话，依然会生效。真正有模板输出的可能仅仅是当前的操作，前置和后置操作一般情况是没有任何输出的。

需要注意的是，在有些方法里面使用了exit或者错误输出之类的话 有可能不会再执行后置方法了。例如，如果在当前操作里面调用了系统Action的error方法，那么将不会再执行后置操作，但是不影响success方法的后置方法执行

### **2.3 参数绑定**

参数绑定是通过直接绑定URL地址中的变量作为操作方法的参数，可以简化方法的定义甚至路由的解析。

'URL\_PARAMS\_BIND' => true



参数绑定功能默认是开启的，其原理是把URL中的参数（不包括模块、控制器和操作名）和操作方法中的参数进行绑定。

参数绑定有两种方式：按照变量名绑定和按照变量顺序绑定，默认使用的是按照变量名绑定，比如看下面的例子：

public function read($id){

echo "read page with </br>".$id;

}

public function archive($year, $month){

echo "$year </br>".$month;

}

对，这个就是上一篇路由所涉及的内容，在之前路由的路由设置处

'blogs/:id' => array('Index/read')

我们将:id直接映射给read()方法的参数$id，所以现在回头再看，其实路由规则就是给了你一个自定义URL的功能。如果去掉上面的路由设置，我们正确的访问方式是：

http://localhost:8999/Home/index/read/id/3

上面的URl中id就是变量名,如果你写成：

public function read($title){

echo "read page with </br>".$title;

}

那么访问地址就是：

http://localhost:8999/index.php/Home/index/read/title/3

对于多个参数绑定的情况，只要将相应的变量名和值传进来就可以了，不在乎顺序，比如下面两个会返回相同的结果：

http://localhost:8999/index.php/Home/index/archive/year/2012/month/12

http://localhost:8999/index.php/Home/index/archive/month/12/year/2012

需要注意的是，不管那种情况之下，当你访问

http://localhost:8999/index.php/Home/index/read/

是会报错的：

参数错误或者未定义:id

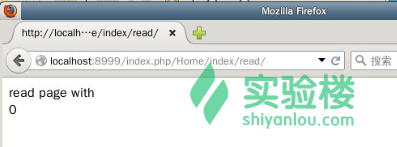
解决的一个好方法就是，给绑定的参数设置默认值，比如：

public function read($id＝0){

echo "read page with </br>".$id;

}

这样再次访问上面的URL，页面结果：



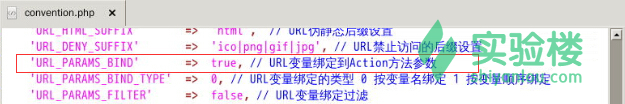
Tips:给绑定参数设置默认值是一个避免报错的好办法

在实际的开发中，我们其实会见到没有显示变量名这样的URL，如：

http://localhost:8999/index.php/Home/index/read/3

怎么解决呢？这个时候，我们其实就可以用到第二种参数绑定：按照变量顺序绑定。要使用这种参数绑定，需要先在设置项中设置：

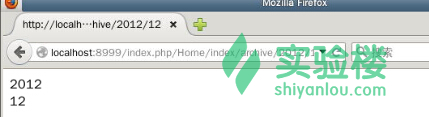
'URL\_PARAMS\_BIND\_TYPE' => 1



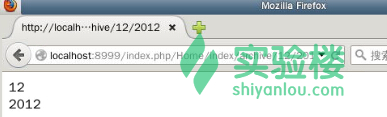
一旦设置变量顺序绑定，这种情况下URL地址中的参数顺序非常重要，不能随意调整。但是这种情况下操作方法的定义不需要改变，只是访问的URL变了而已，现在用上面的方式访问就可以正确访问了。

如果在变量顺序绑定的情况下，我们访问：

http://localhost:8999/index.php/Home/index/archive/2012/12



http://localhost:8999/index.php/Home/index/archive/12/2012



这两个结果显然是不一样，后者并不是我们想要的。所以这种情况需要严格遵守顺序来传值。

### **2.4 伪静态**

URL伪静态通常是为了满足更好的SEO效果，就是在平时上网的时候你经常会看到很多链接都是html结尾的，这对于搜索引擎的收录会更友好，ThinkPHP支持伪静态URL设置，可以通过设置URL\_HTML\_SUFFIX参数随意在URL的最后增加你想要的静态后缀，而不会影响当前操作的正常执行，默认情况下，伪静态的设置为html。但我们可以自己设置，例如

'URL\_HTML\_SUFFIX'=>'shtml'

如果希望支持多个伪静态后缀，可以直接设置如下:

'URL\_HTML\_SUFFIX' => 'html|shtml|xml'

如果此项设置留空则表示可以支持所有的静态后缀。

也可以设置禁止访问的URL后缀通过URL\_DENY\_SUFFIX来设置，例如：

'URL\_DENY\_SUFFIX' => 'pdf|ico|png|gif|jpg',

注： URL\_DENY\_SUFFIX的优先级比URL\_HTML\_SUFFIX要高。

### **2.5 URL生成**

为了配合所使用的URL模式，我们需要能够动态的根据当前的URL设置生成对应的URL地址，为此，ThinkPHP内置提供了U()方法，用于URL的动态生成，可以确保项目在移植过程中不受环境的影响。

****定义规则****

U()方法的定义规则如下（方括号内参数根据实际应用决定）：

U('地址表达式',['参数'],['伪静态后缀'],['显示域名'])

地址表达式

地址表达式的格式定义如下：

[模块/控制器/操作#锚点@域名]?参数1=值1&参数2=值2...

如果不定义模块的话 就表示当前模块名称，下面是一些简单的例子：

U('User/add') // 生成User控制器的add操作的URL地址

U('Article/read?id=1') // 生成Article控制器的read操作 并且id为1的URL地址

U('Admin/User/select') // 生成Admin模块的User控制器的select操作的URL地址

****参数****

U()方法的第二个参数支持数组和字符串两种定义方式，如果只是字符串方式的参数可以在第一个参数中定义，例如：

U('Article/cate',array('cate\_id'=>1,'status'=>1))

U('Article/cate','cate\_id=1&status=1')

U('Article/cate?cate\_id=1&status=1')

三种方式是等效的，都是生成Article控制器的cate()操作 并且cate\_id为1 status为1的URL地址

但是不允许使用下面的定义方式来传参数：

U('Article/cate/cate\_id/1/status/1');

****生成路由地址****

U方法还可以支持路由，如果我们定义了一个路由规则为：

'blogs/:id\d'=>'Index/read'

那么可以使用

U('/blogs/1');

最终生成的URL地址是：

http://localhost:8999/index.php/Home/blogs/1

### **2.6 跳转和重定向**

这应该是在开发中最常用的功能之一。在应用开发中，经常会遇到一些带有提示信息的跳转页面，例如操作成功或者操作错误页面，并且自动跳转到另外一个目标页面。系统的\Think\Controller类内置了两个跳转方法success()和error()，用于页面跳转提示。

#### **跳转**

这个在实际开发Web程序的过程中非常有用，比如一个用户注册之后该怎么跳转等。使用方法很简单，比如我们在Index控制器下新建一个方法user()，写上下面的内容：

public function user()

{

$User = M('User');

$data['username'] = 'Think';

$data['email'] = 'Think@gmail.com';

$result = $User->add($data);

if($result){

$this->success('success', '/Home/User/addUser');

} else {

$this->error('failed');

}

}

M('User')表示实例化一个User对象，add()方法是向数据库添加一条纪录。然后我们需要新建一个UserController，在里面写上addUser()方法

<?phpnamespace Home\Controller;

use Think\Controller;

class UserController extends Controller {

public function addUser()

{

echo 'add user done!';

}

}

然后在浏览器中访问http://localhost:8999/Home/Index/user，就可以看到add user done!了，下面详细来说说这两个重定向方法。

success()和error()方法的第一个参数表示提示信息，第二个参数表示跳转地址，第三个参数是跳转时间（单位为秒），例如：

// redirect to /Article/index after 3 seconds when success$this->success('done','/Home/Article/index',3);// redirect to /Article/error after 5 seconds when failed$this->error('failed','/Home/Article/error',5);

如果不设置跳转时间，默认的等待时间success()方法是1秒，error()方法是3秒。看到上面的两个跳转地址前面都带上了/Home，如果你想简写为/Article/index，你需要在ThinkPHP的入口文件（项目目录下的index.php）中加上下面一行：

define('BIND\_MODULE','Home');

而且这两个方法都有对应的模板，默认的设置是两个方法对应的模板都是：

'TMPL\_ACTION\_ERROR' => THINK\_PATH . 'Tpl/dispatch\_jump.tpl',

'TMPL\_ACTION\_SUCCESS' => THINK\_PATH . 'Tpl/dispatch\_jump.tpl',

你可以根据自己的需要来修改模版。

#### **重定向**

Controller类的redirect()方法可以实现页面的重定向功能。

redirect()方法的参数用法和U()函数的用法一致（参考上一部分URL生成部分），例如：

$this->redirect('/Home/Article/show', array('id' => 2), 3, 'Redirecting...');

上面的用法是停留3秒后跳转到Article控制器的show()操作，并且显示页面跳转中字样Redirecting...，重定向后会改变当前的URL地址。

为了成功进行测试，我们在IndexController下添加redirectToArticle()方法并写上上面那行代码：

public function redirectToArticle()

{

$this->redirect('/Home/Article/show', array('id' => 2), 3, 'Redirecting...');

}

然后我们创建一个ArticleController，并且为他添加show()方法：

namespace Home\Controller;

use Think\Controller;

class ArticleController extends Controller {

public function show($id)

{

echo 'This is an Article Page';

}

}

然后在浏览器访问：http://localhost:8999/Home/Index/redirectToArticle，等待三秒，你就可以看到跳转之后的页面了。

但是如果你仅仅是想重定向要一个指定的URL地址，而不是到某个模块的操作方法，可以直接使用redirect()函数重定向，例如

$this->redirect('/Home/Article/show/id/3', 'Redirecting...',3);

如图所示：



由于实验楼暂时不支持中文，实际的运行结果页面会出现乱码。

注：控制器的redirect()方法和redirect()函数的区别在于前者是用URL规则定义跳转地址，后者是一个纯粹的URL地址

注：好像官方文档是像下面这样写的但我测试并没有成功，成功的是上面的写法。

$this->redirect('/New/category/cate\_id/2', 5, '页面跳转中...');

官方文档可能是出于疏忽的原因有误，参见：

[http://document.thinkphp.cn/manual\_3\_2.html#page\_jump\_redirect](http://document.thinkphp.cn/manual_3_2.html" \l "page_jump_redirect" \t "https://www.shiyanlou.com/courses/96/labs/1186/_blank)

## **三、课后习题**

U('Article/cate',array('cate\_id'=>1,'status'=>1))

U('Article/cate','cate\_id=1&status=1')

U('Article/cate?cate\_id=1&status=1')

分别打印输出以上U()方法生成的url，考虑我们可以在控制器的方法中可以通过哪些方式来获取cate\_id和status变量的值。

对于控制器的更多内容，请仔细查看文档，了解更多

[http://document.thinkphp.cn/manual\_3\_2.html#define\_controller](http://document.thinkphp.cn/manual_3_2.html" \l "define_controller" \t "https://www.shiyanlou.com/courses/96/labs/1186/_blank)

## **四、参考文档：**

本实验课程基于ThinkPHP官方文档控制器篇：

[http://document.thinkphp.cn/manual\_3\_2.html#define\_controller](http://document.thinkphp.cn/manual_3_2.html" \l "define_controller" \t "https://www.shiyanlou.com/courses/96/labs/1186/_blank)

# **4.ThinkPHP 模型**

## **一、实验介绍**

### **1.1 实验内容**

本节实验课程主要学习ThinkPHP的模型知识，从定义开始，涵盖模型的配置，CURD，ActiveRecord，字段验证和关联模型等。本节课中的数据库表已经创建好了，里面也会有相应的数据记录。

### **1.2 实验知识点**

* ThinkPHP 基础
* ThinkPHP 模型

### **1.3 实验环境**

本实验与前一节具有连续性，如果是直接开始本实验，则需要对实验环境进行初始化，部署安装ThinkPHP。直接执行初始化脚本：

curl -s http://labfile.oss.aliyuncs.com/courses/352/install\_thinkphp.sh | sh -

## **二、模型定义**

默认情况下，ThinkPHP的模型类是位于/Home/Model/目录之下，模型类通常需要继承系统的\Think\Model类或其子类，下面是一个Home\Model\UserModel类的定义：

文件命名遵守UserModel.class.php的方式，跟控制器的命名一样

<?phpnamespace Home\Model;

use Think\Model;

class UserModel extends Model {

}

模型类的作用大多数情况是操作数据表的，如果按照系统的规范来命名模型类的话，大多数情况下是可以自动对应数据表，但你可以根据自己的需求来定制自己的数据表设置和操作。

首先我们需要在配置文件设置我们的数据库连接信息：

'DB\_TYPE' => 'mysql',

'DB\_HOST' => 'localhost',

'DB\_NAME' => 'database',

'DB\_USER' => 'username',

'DB\_PWD' => 'password',

'DB\_PORT' => '3306',

这些配置信息是在/Home/Conf/config.php文件里设置，我们第一课的时候就看到过这些配置。

### **2.1 指定数据表前缀**

指定标前缀，我们在第一课的配置项已经指定，以下有关数据表前缀的课程只是为了说明你可以灵活配置你的数据表。

protected $tablePrefix = 'top\_';

1. 列表项. 列表项. 如果数据库的表没有表前缀，使用空字符串代替

protected $tablePrefix = '';

1. 列表项. 列表项. 指定数据表，此处的指定的数据表的不需要添加表前缀：

protected $tableName = 'user';

举个例子说，比如说你的数据库中有一个没有表前缀的，名为users的数据表，可以用以下的两种方法在模型中进行下面的定义：

第一，直接根据系统的规范来命名模型类来命名模型，比如说就命名为UsersModel那么只需要在这个类里面加上下面的设置就可以了：

protected $tablePrefix = '';

ThinkPHP系统就会自动定位到users表了。

第二种情况时，如果你的模型类没有按照系统规范来命名，比如说不小心命名为UserModel，这种情况下可以同时指定表前缀和表名，比如：

protected $tablePrefix = '';

protected $tableName = 'users';

或者你直接指定trueTableName:

protected $trueTableName = 'users';

既然模型通常是用来操作数据表，那么我们来看看模型的基本CURD:

注：为了方便演示，我们在UserController中定义一个testDemo()方法用于演示

public function testDemo()

{

}

以下的代码将会一段一段在这个方法里演示，你可以通过访问http://localhost:8999/index.php/Home/User/testDemo来看到实际效果。

### **2.2添加记录**

$user = M('User');

$data['username'] = 'ThinkPHP';

$data['email'] = 'ThinkPHP@gmail.com';

$user->create($data);

$record = $user->add();

dump($record);

add()返回的是插入数据的id,对于不存在的表字段，add()方法会自动过滤。

### **2.3读取记录**

在ThinkPHP中读取数据的方式很多，通常分为读取数据、读取数据集和读取字段值

$user = M('User');

$record = $user->where('username="ThinkPHP"')->find();

dump($record);

### **2.4 读取字段值**

$user = M('User');

$record = $user->where('id=3')->getField('username');

dump($record);

默认情况下，当只有一个字段的时候，返回满足条件的数据表中的该字段的第一行的值.如果getField()传入多个字段，返回值将是一个关联数组：

$user = M('User');

$record = $user->getField('username,email');

dump($record);

这个数组总是以传入的第一个第一个字段为键值的。如果修改为：

$user = M('User');

$record = $user->getField('email,username');

dump($record);

将上面的两次代码分别放到testDemo()，你就会看到不一样的结果集。

### **2.5 用save()方法更新数据**

$user = M('User');

$data['username'] = 'ThinkPHPSave';

$data['email'] = 'ThinkPHPSave@outlook.com';

$record = $user->where('id=3')->save($data);

dump($record);

这里的$record返回的事1，表示成功更改。

当然，你也可以这样：

$user = M('User');

$user->username = 'ThinkPHP';

$user->email = 'ThinkPHP@outlook.com';

$record = $user->where('id=3')->save();

dump($record);

日常开发的时候经常会遇到一些只更新某些字段的情况，可以通过下面的方式来实现：

$user = M("User"); $record = $user->where('id=4')->setField('username','ThinkPHPChangeName');

dump($record);

同时更新多个字段，可以将数据以数组的形式传给setField()方法：

$user = M('User');

$data = array('username'=>'ThinkPHPChangeArray','email'=>'ThinkPHP@array.com');

$record = $user-> where('id=6')->setField($data);

dump($record);

### **2.6 ThinkPHP删除数据使用delete()方法**

$user = M('User');

$record = $user->where('id=3')->delete();dump($record);

或者你可以直接使用：

$record = $user->delete('1,2,5');dump($record);

这样就达到了删除主键1,2,5这三条记录了。

### **2.7 ActiveRecords**

ThinkPHP实现了ActiveRecords模式的ORM模型，采用了非标准的ORM模型：表映射到类，记录映射到对象。以下实例将使用ActiveRecords重现对数据表的CURD，看看ActiveRecords给我们带来了什么好处。

$user = M("User");

$user->username = 'ThinkPHPWithActive';

$user->email = 'ThinkPHPActive@gmail.com';

$record = $user->add();

dump($record);

### **2.8 读取记录**

AR最大的特点可能就是它的查询模式了，模式简单易用，因为更多情况下面查询条件都是以主键或者某个关键的字段。这种类型的查询，ThinkPHP有着很好的支持。

比如说获取主键为2的用户信息：

$user = M("User");

$record = $user->find(2);

dump($record);

直接不用where()查询了，简单友好吧。再比如：

$user = M("User");

$record = $user->getByUsername("jelly");

dump($record);

如果是查询多条记录，使用以下方式：

$user = M("User");

$record = $user->select('1,3,8');

dump($record);

### **2.9 更新记录**

$user = M("User");

$user->find(21);

$user->username = 'TOPThinkChangeWithAR';

$record = $user->save();

dump($record);

### **2.10 删除记录**

1. 删除单条记录

$user = M("User");

$record = $user->delete(8);

dump($record);

1. 删除多条记录

$user = M("User");

$record = $user->delete('15,16');

dump($record);

### **2.11 自动完成**

自动完成是ThinkPHP提供用来完成数据自动处理和过滤的方法，当使用create()方法创建数据对象的时候会触发自动完成数机制。

因此，在ThinkPHP鼓励使用create()方法来创建数据对象，因为这是一种更加安全的方式，直接通过add()或者save()方法实现数据写入无法出发自动完成机制。

自动完成通常用来完成默认字段写入(比如添加时间戳)，安全字段过滤(比如加密密码)以及业务逻辑的自动处理等。可以通过模型类里面通过$\_auto属性定义处理规则。下面演示如何自动完成添加时间戳：

在UserModel中，声明自动完成的定义数组$\_auto ：

protected $\_auto = array (

array('created\_at','date("Y-m-d H:i:s", time())',3,'function'),

array('updated\_at','date("Y-m-d H:i:s", time())',3,'function'),

);

还有一种是理由auto()方法动态设置自动完成的机制，可以到官方文档去看看

设置完成之后，我们在testDemo()方法中创建一条用户数据：

$user = D('User');

$data['username'] = "ThinkPHP";

$data['email'] = "ThinkPHP@gmail.com";

$user->create($data);

$record = $user->add();dump($record);

测试，如果返回记录的id值，说明用户记录创建成功。要验证数据是否自动完成，你可以直接使用：

$user = D('User');

$record = $user->find(id);dump($record);

### **2.12 自动验证**

自动验证是ThinkPHP模型层提供的一种数据验证方法，可以在使用create()创建数据对象的时候自动进行数据验证。

数据验证可以进行数据类型、业务规则、安全判断等方面的验证操作。

通常用于表单验证

****数据验证有两种方式：****

静态方式：在模型类里面通过$\_validate属性定义验证规则。

动态方式：使用模型类的validate()方法动态创建自动验证规则。

无论是什么方式，验证规则的定义是统一的规则，定义格式为：

array(

array(验证字段1,验证规则,错误提示,[验证条件,附加规则,验证时间]),

array(验证字段2,验证规则,错误提示,[验证条件,附加规则,验证时间]),

......

);

下面以$\_validate静态方式举例如何使用自动验证：

1. 在UserController中创建register()方法，对，几乎每一个Web应用都需要实现用户注册这一步。

public function register()

{

$this->display();

}

对，就是这么简单，这个方法只是将相应的视图文件渲染出来。所以接下来我们创建对应的视图文件，也就是：./Application/Home/View/User/register.html

<extend name="Index/base" /><block name="main" ><form method="post" action="\_\_URL\_\_/registerValidate">

<div class="form-group">

<label for="exampleInputName">Name</label>

<input type="text" name="username" class="form-control" id="exampleInputName" placeholder="Name">

</div>

<div class="form-group">

<label for="exampleInputEmail">Email</label>

<input type="email" name="email" class="form-control" id="exampleInputEmail" placeholder="Email">

</div>

<button type="submit" class="btn btn-default">Submit</button></form></block>

上面就是一些HTML代码和一点模板的知识，对于模板，我们后续会讲到，但不管怎样，现在我们访问

http://localhost:8999/Home/User/register，就可以看到我们的注册表单页面了。

注意到form表单中，action="\_\_URL\_\_/registerValidate"，这表示提交到当前的控制器的registerValidate()方法处理，所以我们在UserController中增加registerValidate()方法：

public function registerValidate()

{

$data['username'] = $\_POST['username'];

$data['email'] = $\_POST['email'];

$user = D("User");

if ( !$user->create($data) ) {

exit($user->getError());

}

//todo: validation passes, add data to database and redirect somewhere

echo 'validation passes';

}

这里的if ( !$user->create($data) )会触发自动验证并判断验证是否通过验证。你可以尝试在表单里填写不同的数据来进行测试，也可以修改一下验证规则，更多规则可以到官网查看：

[http://document.thinkphp.cn/manual\_3\_2.html#auto\_validate](http://document.thinkphp.cn/manual_3_2.html" \l "auto_validate" \t "https://www.shiyanlou.com/courses/96/labs/1187/_blank)

### **2.13 关联模型**

通常我们所说的关联关系包括下面三种：

一对一关联 ：ONE\_TO\_ONE，包括HAS\_ONE 和 BELONGS\_TO

一对多关联 ：ONE\_TO\_MANY，包括HAS\_MANY 和 BELONGS\_TO

多对多关联 ：MANY\_TO\_MANY

#### **关联定义**

ThinkPHP可以很轻松的完成数据表的关联CURD操作，目前支持的关联关系包括下面四种：

HAS\_ONE、BELONGS\_TO、HAS\_MANY和MANY\_TO\_MANY。

一个模型根据业务模型的复杂程度可以同时定义多个关联，不受限制，所有的关联定义都统一在模型类的 $\_link 成员变量里面定义，并且可以支持动态定义。要支持关联操作，模型类必须继承Think\Model\RelationModel类，关联定义的格式类似于：

namespace Home\Model;use Think\Model\RelationModel;class UserModel extends RelationModel{

protected $\_link = array(

'关联' => array(

'关联属性1' => '定义',

'关联属性N' => '定义',

),

);

}

关于关联属性的定义和值，你可以到官方文档仔细查看，我们下面也会给出一些最常用的。

在我们的讲解例子中，会采用HAS\_MANY和BELONGS\_TO来演示，对于其他的几个关系模型，可以参考官方文档举一反三。

首先我们知道数据库里面有两张表，用户表和文章表，并且我们也为其创建了不同的模型(UserModel ArticelModel)。

现在我们仔细来想想他们之间的对应关系：一个用户可以拥有多篇文章，而每一篇文章都属于某个特定的用户。所以我们可以分别为这两种关系添加关联模型：

在UserModel中：

protected $\_link = array(

'Article' => self::HAS\_MANY

);

在ArticleModel中：

protected $\_link = array(

'User' => self::BELONGS\_TO

);

以上者两种都是最简洁的模型关联声明。因为在最开始设计数据库的时候，我们遵守了ThinkPHP的官方的规范：

外键的默认规则是当前数据对象名称\_id，例如：UserModel对应的可能是表think\_user，那么think\_user表的外键默认为user\_id，如果你的外键不是user\_id，而是其他自定义的字段如：user\_identify，那么就必须在定义关联的时候定义foreign\_key 。如下：

在UserModel中：

protected $\_link = array(

'mapping\_type' => self::HAS\_MANY,

'class\_name' => 'Article',

'foreign\_key' => 'user\_identify',

);

更多自定义的关联模型参数可以到官网查看。

有了以上的定义之后，我们就可以在检索用户数据的同时将属于他的文章也一起检索出来，使用relation()。

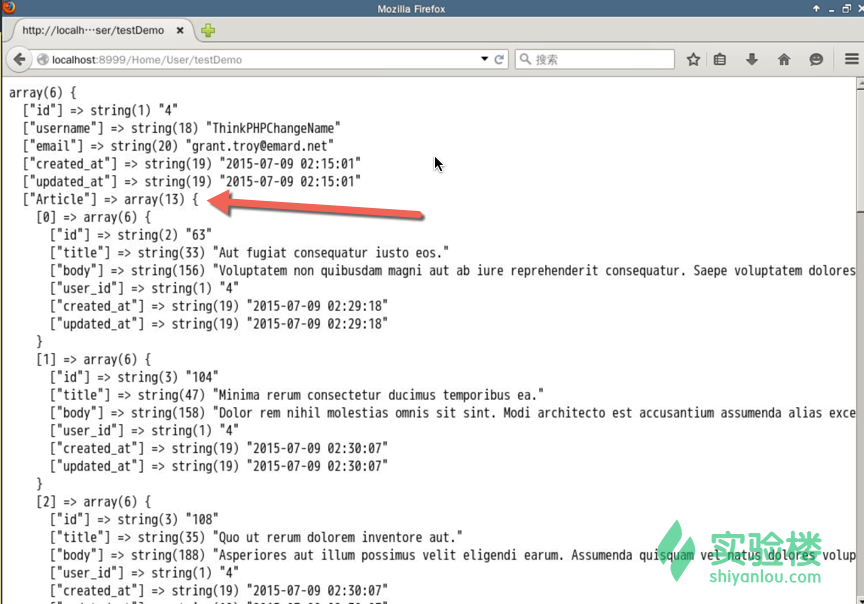
同样是在testDemo()这个方法中：

$user = D('User');

$record = $user->relation(true)->find(4);

dump($record);

访问熟悉的http://localhost:8999/Home/User/testDemo，你将会看到神奇的结果。



## **三、课后习题**

1. 设计一个think\_profile的数据库表结构，使得可以于think\_user表形成一对一的关系，在模型中给出对应关系的定义。
2. 考虑如果是多对多的关系，比如有think\_article和think\_tag这两个表，如果实现多对多的对应关系

## **四、参考文档**

本试验课程基于ThinkPHP的官方文档模型篇

[http://document.thinkphp.cn/manual\_3\_2.html#define\_model](http://document.thinkphp.cn/manual_3_2.html" \l "define_model" \t "https://www.shiyanlou.com/courses/96/labs/1187/_blank)

# **5.ThinkPHP 视图**

## **一、实验介绍**

### **1.1 实验内容**

本节实验课程主要学习ThinkPHP的视图配置，设置主题，变量赋值等知识点。

### **1.2 实验知识点**

* ThinkPHP 基础
* ThinkPHP 视图

### **1.3实验环境**

本实验与前一节具有连续性，如果是直接开始本实验，则需要对实验环境进行初始化，部署安装ThinkPHP。直接执行初始化脚本：

curl -s http://labfile.oss.aliyuncs.com/courses/352/install\_thinkphp.sh | sh -

## **二、视图 (View)**

本节主要是讲解ThinkPHP的视图管理系统，从视图的基本概念，配置到赋值，渲染等。

在ThinkPHP中，每个模块的模板文件是独立的，为了对模板文件更加有效的管理，ThinkPHP对模板文件进行目录划分，默认的模板文件定义规则是：

视图目录/[模板主题/]控制器名/操作名+模板后缀

默认的视图目录是模块的View/目录（一个模块可以有多个视图文件目录，这取决于你的应用需要），框架的默认视图文件后缀是.html。 新版模板主题默认是空（表示不启用模板主题功能），即在没有主题的情况之下：

视图目录/控制器名/操作名+模板后缀

在每个模板主题下面，是以模块(常用的/Home/Controller就是一个模块)下面的控制器名为目录，然后是每个控制器的具体操作模板文件，例如:

Index控制器的index操作 对应的视图文件就应该是：/Application/Home/View/Index/index.html

注意与控制器相对应的文件夹大小写，就像上例第一个的Index一样需要大写

比如我们要为ArticleController的show()方法渲染一个视图文件，这个视图文件应该在：

./Application/Home/View/Article/show.html

视图文件的默认后缀的情况是.html，也可以通过 TMPL\_TEMPLATE\_SUFFIX 来配置成其他的。例如，我们可以配置：

'TMPL\_TEMPLATE\_SUFFIX'=>'.tpl'

定义后，Index控制器的index()操作 对应的视图文件就变成是： /Application/Home/View/Index/index.tpl

### **2.1 模板主题**

一个模块如果需要支持多套视图文件的话，就可以使用模板主题功能。 默认情况下，没有开启模板主题功能，如果需要开启，设置 DEFAULT\_THEME 参数即可：

'DEFAULT\_THEME' => 'default'

采用视图主题后，需要在视图目录(View/)下面创建对应的主题目录，和不启用模板主题的情况相比，模板文件只是多了一层目录：

View/Index/index.html // before using a theme

View/default/Index/index.html // after using a theme

### **2.2 模板赋值**

如果要在视图文件中输出变量，必须在在控制器中把变量传递给视图，系统提供了assign()方法对视图变量赋值，无论何种变量类型都统一使用assign()赋值，比如在IndexController中index()操作之中。

public function index(){

$user = D('User');

$userInfo = $user->find(2);

$this->assign('user',$userInfo);

$this->display();

}

assign()方法必须在display()和show()方法之前调用，并且系统只会输出设定的变量，其它变量不会输出（系统变量例外），一定程度上保证了变量的安全性。如果要保证上面的代码正常运行，我们需要创建一个对应的视图文件，根据上面的规律就是：

/Application/Home/View/Index/index.html

在模板里写上简单的HTML：

<html><head><title>IndexController index</title></head><body><h1>This is IndexController index view !!!</h1></body></html>

访问首页，你就会看到一串This is IndexController index view !!!。

既然有了赋值，那就可以在视图文件中输出变量了，如果使用的是内置模板的话，就可以这样输出： {$user['email']}，后面章节会具体讲到。

### **2.3 模板渲染**

视图正确定义后就可以渲染视图输出，系统也支持直接渲染内容输出，就像上面说到的一样，视图赋值assign()必须在模板渲染display()和show()之前操作。

渲染视图文件输出最常用的是使用display()方法，调用格式：

display('[视图文件]'[,'字符编码'][,'输出类型'])

视图的经典用法就是不带任何参数，就像在视图赋值部分看到的一样：

$this->display();

上面这行代码表示，系统会自动寻找在这个控制器之下的具体操作的视图(跟方法名一样的视图文件)，比如上面的Index控制器index()操作视图。

如果没有按照模板定义规则来定义视图文件（或者需要调用其他控制器下面的某个视图文件），可以使用：

$this->display('edit');

如果我们使用了模板主题功能，那么也可以支持跨主题调用，使用：

$this->theme('blue')->display('Index:edit');

表示调用blue主题下面的Index控制器的edit视图文件。

## **三、 作业**

为Article控制器的各个方法创建默认视图。

## **四、参考文档**

本试验课程基于ThinkPHP官方文档视图篇

[http://document.thinkphp.cn/manual\_3\_2.html#template\_define](http://document.thinkphp.cn/manual_3_2.html" \l "template_define" \t "https://www.shiyanlou.com/courses/96/labs/1188/_blank)

# **6.ThinkPHP 模板**

## **一、实验介绍**

### **1.1 实验内容**

本节实验课程主要学习ThinkPHP的模版知识，包括变量输出，变量的各种操作，内置标签和模板的继承与布局等。

### **1.2 实验知识点**

* ThinkPHP 基础
* ThinkPHP 模板

### **1.3 实验环境**

本实验与前一节具有连续性，如果是直接开始本实验，则需要对实验环境进行初始化，部署安装ThinkPHP。直接执行初始化脚本：

curl -s http://labfile.oss.aliyuncs.com/courses/352/install\_thinkphp.sh | sh -

## **二、 模板 (Template)**

在上面的视图中，我们曾提到过给视图赋值，那时没有详细讲解时因为就单单ThinkPHP的模板知识也牵涉到比较多的知识点，故本节将详细讲解ThinkPHP的模板。

### **2.1 变量输出**

在赋值给模板之后，在模板中输出变量很简单，如在前面，我们将查询到的用户信息赋给了index模板：

public function index(){

$user = D('User');

$userInfo = $user->find(2);

$this->assign('user',$userInfo);

$this->display();

}

那么在index模板中就可以这样输出变量，

<html><head><title>IndexController index</title></head><body><h1>Hello {$user['username']} !</h1></body></html>

将变量值包含在一对大括号内，系统就会自动编译输出变量值，编译后其实是这样的：

Hello,<?php echo($user['username']);?>！

注意{和$user之间是没有空格的，如果两者存在空格，写出{ $user['username']}，系统就不采用编译机制，而是直接输出字符串$user['username']。

普通标签默认开始标记是{，结束标记是 }。也可以通过设置TMPL\_L\_DELIM和TMPL\_R\_DELIM进行更改。例如，我们在项目配置文件中定义：

'TMPL\_L\_DELIM'=>'{{','TMPL\_R\_DELIM'=>'}}',

那么，上面的变量输出标签就应该改成：

Hello {{$user['username']}} !

当然如果你觉得{$user['username']}不太符合口味，你还可以使用以下类似取属性方式输出变量值：

Hello {$user.username} !

两者都会得到一样的效果。

### **2.2 变量操作**

对于赋值给模板的变量，我们可以对其进行一些常规的操作和运算，包括对变量的+，– ，\*，/和%等

比如：

{$a + $b}

{$a % $b}

{$a ++}

不过需要注意的是，在进行变量计算操作的时候，变量不再支持上述的类似取属性的方法(就是点语法)，所以下面的语法是错的：

{$user.score+10}

正确的写法是：

{$user['score']+10}

### **2.3 使用三元运算符**

在模板中也支持使用三元运算符，不过在这里也不支持点语法

{$status?'published':'archive'}

{$info['status']?$info['msg']:$info['error']}

### **2.4 默认值输出**

这很好地解决了当我们输出的值为空值的时候，我们可以在模板处为它设置默认值，比如：

{$user.username|default="Harry"}

### **2.5 模板标签库**

内置的模板引擎除了支持普通变量的输出之外，更强大的地方在于标签库功能。

标签库类似于Java的Struts中的JSP标签库，每一个标签库是一个独立的标签库文件，标签库中的每一个标签完成某个功能，采用XML标签方式（包括开放标签和闭合标签）。

标签库分为内置和扩展标签库，内置标签库是Cx标签库

### **2.6 内置标签**

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，所以，Cx标签库中的所有标签，我们可以在模板文件中直接使用，我们可以这样使用：

<eq name="status" value="1 >normal</eq>

### **2.7 volist标签**

volist标签通常用于查询数据集（select()方法）的结果输出，通常模型的select()方法返回的结果是一个二维数组，可以直接使用volist标签进行输出。 在控制器中首先对模版赋值

$user = M('User');

$list = $user->limit(10)->select();$this->assign('list',$list);

在模版定义如下，循环输出用户的用户名和邮箱：

<volist name="list" id="vo">

{$vo.username}:{$vo.email}<br/></volist>

Volist标签的name属性表示模板赋值的变量名称$this->assign('list',$list)中的list，因此不可随意在模板文件中改变。id表示当前的循环变量，可以随意指定，但确保不要和name属性冲突，例如

<volist name="list" id="data">

{$data.username}:{$data.email}<br/></volist>

### **2.8 IF标签**

<if condition="($name eq 1) OR ($name gt 100) "> value1

<elseif condition="$name eq 2"/>value2

<else /> value3

</if>

在condition属性中可以支持eq等判断表达式，同上面的比较标签，但是不支持带有>、<等符号的用法，因为会混淆模板解析，所以下面的用法是错误的：

<if condition="$id < 5 ">value1

<else /> value2

</if>

必须改成：

<if condition="$id lt 5 ">value1

<else /> value2

</if>

由于if标签的condition属性里面基本上使用的是php语法，尽可能使用判断标签和Switch标签会更加简洁，原则上来说，能够用switch和比较标签解决的尽量不用IF标签完成。因为switch和比较标签可以使用变量调节器和系统变量。如果某些特殊的要求下面，IF标签仍然无法满足要求的话，可以使用原生php代码或者PHP标签来直接书写代码

### **2.9 switch标签**

switch标签的格式如下：

<switch name="变量" >

<case value="值1" break="0或1">输出内容1</case>

<case value="值2">输出内容2</case>

<default />默认情况

</switch>

上面的中文是问了便于理解。

使用方法：

<switch name="User.level">

<case value="1">value1</case>

<case value="2">value2</case>

<default />default

</switch>

对于case的value属性可以支持多个条件的判断，使用|进行分割，而且case标签还有一个break属性，表示是否需要break，默认是会自动添加break，如果不要break，可以使用：

<switch name="Think.get.type">

<case value="gif|png|jpg" break="0">Images</case>

<case value=mp4|mov|flv">Videos</case>

<default />Other format</switch>

### **2.10 Foreach标签**

foreach标签类似与volist标签，只是更加简单，没有太多额外的属性，例如： {$vo.id}:{$vo.name}

name表示数据源 item表示循环变量

<foreach name="list" item="vo" >

{$key}|{$vo}

</foreach>

也可以定义索引的变量名

<foreach name="list" item="vo" key="k" >

{$k}|{$vo}

</foreach>

### **2.11 模板继承**

模板继承是一项更加灵活的模板布局方式，模板继承不同于模板布局，甚至来说，应该在模板布局的上层。模板继承其实并不难理解，就好比类的继承一样，模板也可以定义一个基础模板（或者是布局），并且其中定义相关的区块（block），然后继承（extend）该基础模板的子模板中就可以对基础模板中定义的区块进行重载。

因此，模板继承的优势其实是设计基础模板中的区块（block）和子模板中替换这些区块。

每个区块由标签组成。 下面就是基础模板中的一个典型的区块设计（用于设计网站标题）：

<block name="title"><title>Website title</title></block>

block标签必须指定name属性来标识当前区块的名称，这个标识在当前模板中应该是唯一的，block标签中可以包含任何模板内容，包括其他标签和变量，例如

<block name="title"><title>{$web\_title}</title></block>

一个模板中可以定义任意多个名称标识不重复的区块，例如下面我们在View/Index中定义了一个base.html基础模板：

<!DOCTYPE html><html lang="zh-CN"><head><meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1"><block name="title"><title>ThinkPHP Base Template</title></block>

<import type='css' file='css.bootstrap'/></head><body><div class="container"><div class="col-md-2"></div><block name="menu"></block><div class="row">

<div class="col-md-2"><block name="left"></block></div>

<div class="col-md-8"><block name="main"></block></div>

<div class="col-md-2"><block name="right"></block></div></div>

<div class="col-md-2"></div><block name="footer"></block></div></body></html>

如果你仔细看了前面在自动验证那一节内容的HTML表单代码，你就会理解之前<extend name="Index/base" />是什么意思了：其实就是当时的HTML表单的模板继承了上面的base.html

这里需要注意的地方是，在继承模板的时候，注意名字要一一对应，也就是确保和base.html的block区块的name属性一致。

## **三、课后习题**

1. 查阅更多ThinkPHP的内置模板标签：[http://document.thinkphp.cn/manual\_3\_2.html#inner\_tag](http://document.thinkphp.cn/manual_3_2.html" \l "inner_tag" \t "https://www.shiyanlou.com/courses/96/labs/1189/_blank)
2. 仔细阅读文档模版继承的部分，理解模板的继承和布局：[http://document.thinkphp.cn/manual\_3\_2.html#template\_extend](http://document.thinkphp.cn/manual_3_2.html" \l "template_extend" \t "https://www.shiyanlou.com/courses/96/labs/1189/_blank)

## **四、参考文档**

本试验课程基于ThinkPHP的官方稳定模板篇：

[http://document.thinkphp.cn/manual\_3\_2.html#var\_output](http://document.thinkphp.cn/manual_3_2.html" \l "var_output" \t "https://www.shiyanlou.com/courses/96/labs/1189/_blank)

# **7.ThinkPHP 文件与验证码**

## **一、实验介绍**

### **1.1 实验内容**

本节实验课程主要学习ThinkPHP的内置一些重要的专题，包括文件上传，图片操作和验证码知识等。

### **1.2 实验知识点**

* ThinkPHP 基础
* ThinkPHP 文件
* ThinkPHP 扩展

### **1.3实验环境**

本实验与前一节具有连续性，如果是直接开始本实验，则需要对实验环境进行初始化，部署安装ThinkPHP。直接执行初始化脚本：

curl -s http://labfile.oss.aliyuncs.com/courses/352/install\_thinkphp.sh | sh -

## **二、文件上传**

在ThinkPHP中使用上传功能无需进行特别处理。例如，下面是一个带有附件上传的表单提交：

<form action="\_\_URL\_\_/upload" enctype="multipart/form-data" method="post" ><input type="text" name="name" /><input type="file" name="photo" /><input type="submit" value="提交" ></form>

如果是多文件上传，推荐使用下面的写法：

<input type='file' name='photo[]'>

<input type='file' name='photo[]'>

<input type='file' name='photo[]'>

实际例子：我们在UserController中添加一个uploadDemo()方法，用来渲染视图

public function uploadDemo()

{

$this->display();

}

默认渲染的是View\User\uploadDemo.html,所以创建该文件：

<extend name="Index/base" /><block name="main" >

<form method="post" action="\_\_URL\_\_/upload" enctype="multipart/form-data">

<div class="form-group">

<label for="exampleInputFile">Choose File</label>

<input type="file" name="file" id="exampleInputFile">

<p class="help-block">Choose your file to upload</p>

</div>

<button type="submit" class="btn btn-default">Submit</button>

</form></block>

代码跟之前自动验证的那节差不多，只是注意上传文件的时候指定enctype="multipart/form-data"，上面视图也是继承于Index\base.html，表单提交到当前控制器的uploadFile()方法，所以，我们在UserController中实现之：

public function uploadFile(){

$upload = new \Think\Upload();

$upload->maxSize = 3145728 ;

$upload->exts = array('jpg', 'gif', 'png', 'jpeg');

$upload->rootPath = './Uploads/';

$upload->savePath = 'images/';

$info = $upload->upload();

if(!$info) {

$this->error($upload->getError());

}else{

$this->success('upload done！');

// redirect some where

}

}

实例化上传类之后，我们可以指定上传文件的大小，文件类型，保存路径等。需要注意的是保存上传文件的根路径./Uploads/需要手动创建，不然上传文件会失败。上传的文件ThinkPHP会默认在savePath之下根据日期创建文件夹，文件名也会跟着系统的默认方法来命名，在上传成功后$this->success('upload done！')只是简单的提示上传成功，在实际项目中，如果用户上传的是头像，你可以跳转到用户的主页等;

### **2.1 图片处理**

上面说到了图片的上传，实际的项目中，我们需要对上传的文件进行多种多样的处理，像是在用户上传头像的时候进行裁剪，像是在微博中上传图片的时候加上水印等。这些技术的实现在ThinkPHP里面很简单，我们仔细来看看

ThinkPHP使用Think\Image类进行图像处理功能，支持Gd库和Imagick库，包括对GIf图像处理的支持

$image = new \Think\Image();

默认使用GD库进行图像操作，如果需要使用Imagick库操作的话，需要改成：

如果需要使用Imagick库，请先通过apt-get install -y php5-imagick来安装。

$image = new \Think\Image(\Think\Image::IMAGE\_IMAGICK);

//or

$image = new \Think\Image('Imagick');

### **2.2 裁剪图片**

使用crop()和save()方法完成裁剪图片功能。再次将目光回到UserController的testDemo()方法，我们还是在这里测试：

$image = new \Think\Image();

$image->open('./shiyanlou.png');

$crop = $image->crop(400, 400)->save('./Uploads/images/crop.png');

首先使用open()方法打开一张图片，假设当前入口文件目录下面有一个shiyanlou.png文件。然后使用crop()裁剪，最后执行save()保存图片。在浏览器访问http://localhost:8999/index.php/Home/User/testDemo，然后查看./Uploads/images/文件夹，你就会看到裁剪出来的照片了。

crop()还支持坐标式裁剪，坐标参数x， y作为crop()第三，第四两个参数传入：

### **2.4 生成缩略图**

使用thumb()方法生成缩略图

$image->thumb(256,256)->save('./Uploads/images/thumb.png');

回到./Uploads/images/查看thumb.png，我们看到实际生成的缩略图并不是256\*256，因为默认采用原图等比例缩放的方式生成缩略图，最大宽度是256。当然，ThinkPHP还可以支持其他类型的缩略图生成，具体参考下面的参数，可以将其作为第三个参数传入thumb()方法：

IMAGE\_THUMB\_SCALE = 1 ; //等比例缩放类型IMAGE\_THUMB\_FILLED = 2 ; //缩放后填充类型IMAGE\_THUMB\_CENTER = 3 ; //居中裁剪类型IMAGE\_THUMB\_NORTHWEST = 4 ; //左上角裁剪类型IMAGE\_THUMB\_SOUTHEAST = 5 ; //右下角裁剪类型IMAGE\_THUMB\_FIXED = 6 ; //固定尺寸缩放类型

### **2.5 添加图片水印**

$image->open('./shiyanlou.png')->water('./logo.png')->save("./Uploads/images/water\_mark.png");

到保存文件的文件夹中查看water\_mark.png，你会看到右下角的地方打上实验楼的logo。

## **三、验证码**

Think\Verify类可以支持验证码的生成和验证功能。

在web应用中常常会看到验证码的应用，通常是在登录或注册的时候使用，我们看看ThinkPHP时怎么实现这个功能的。

### **3.1 生成验证码：**

在UserController中增加下面的verify()方法，后面我们会用到：

public function verify()

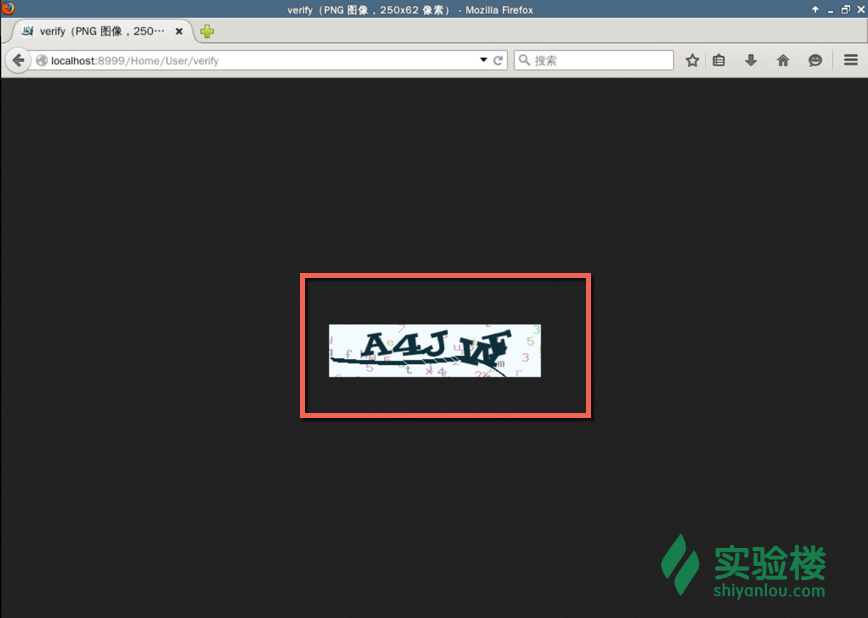
{

$Verify = new \Think\Verify();

$Verify->entry();

}

首先你可以通过直接访问来体验一下ThinkPHP的验证码：http://localhost:8999/index.php/Home/User/verify。你应该会开到一张验证码图片。



### **3.2 配置个性化的验证码：**

$config = array(

'fontSize' => 30, // 验证码字体大小

'length' => 3, // 验证码位数

'useNoise' => false, // 关闭验证码杂点

);

$Verify = new \Think\Verify($config);

$Verify->entry();

更多详情配置可以到官网查看：

[http://document.thinkphp.cn/manual\_3\_2.html#verify](http://document.thinkphp.cn/manual_3_2.html" \l "verify" \t "https://www.shiyanlou.com/courses/96/labs/1190/_blank)

作业：看了官方文档之后，设置一个中文验证码试试。

### **3.3 验证码检测**

可以用Think\Verify类的check()方法检测验证码的输入是否正确。结合前面在注册页面的自动验证的例子，我们为其加上验证码验证。

首先在./Application/Home/View/User/register.html下加入验证码的表单：

<div class="form-group">

<label for="exampleInputEmail">Email</label>

<input type="email" name="email" class="form-control" id="exampleInputEmail" placeholder="Email">

</div>

<div class="form-group">

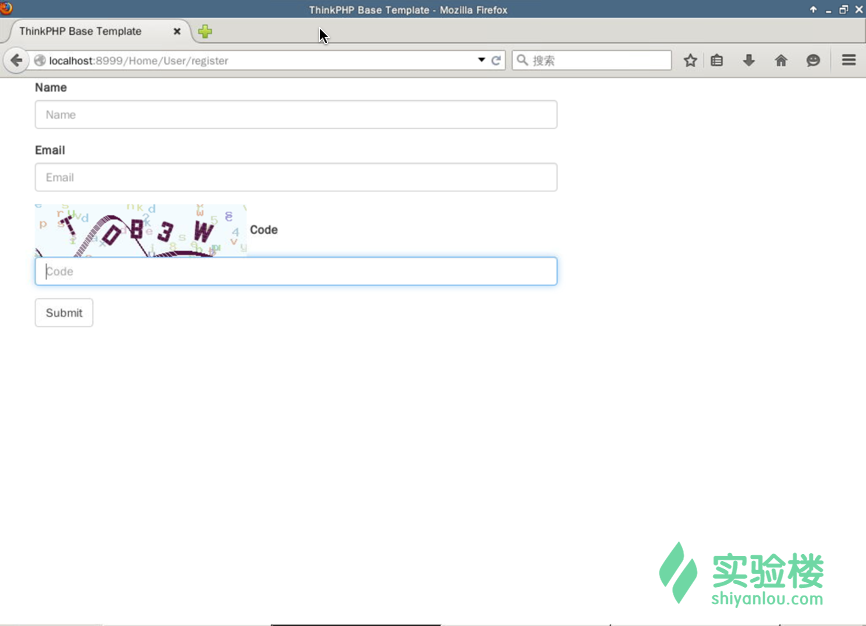
<img id="verifyImg" src="\_\_URL\_\_/verify" />

<label for="exampleInputCode">Code</label>

<input type="text" name="captcha" class="form-control" id="exampleInputCode" placeholder="Code">

</div>

表单中最后一个输入框就是验证码的输入框，其中<img id="verifyImg" src="\_\_URL\_\_/verify" />中的src="\_\_URL\_\_/verify"表示验证码图片的地址，也就是当前控制器(UserController)的verify()方法。



然后在UserController再添加一个checkVerify()，用于检验验证码是否通过。这个方法的核心其实就是调用\Think\Verify类的check()方法

function checkVerify($code, $id = '')

{

$verify = new \Think\Verify();

return $verify->check($code, $id);

}

这些都准备好之后，我们可以来到UserController下的registerValidate()中，稍微修改一下代码：

public function registerValidate()

{

$data['username'] = $\_POST['username'];

$data['email'] = $\_POST['email'];

if ( $this->checkVerify($\_POST['captcha']) ) {

$user = D("User");

if ( !$user->create($data) ) {

exit($user->getError());

}

// validation passes, add data to database and redirect somewhere

echo 'validation passed';

}else{

echo 'validation failed';

}

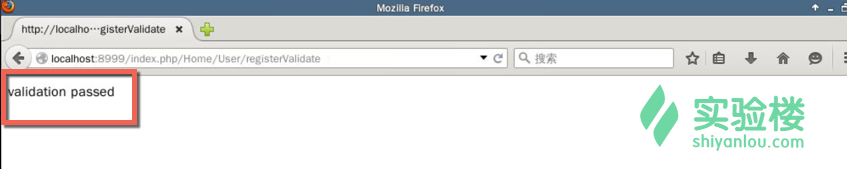
}

其他的几乎都没有改变，就直接在创建用户数据的外面加上if条件判断，只有在验证通过的情况下我们才创建用户数据。

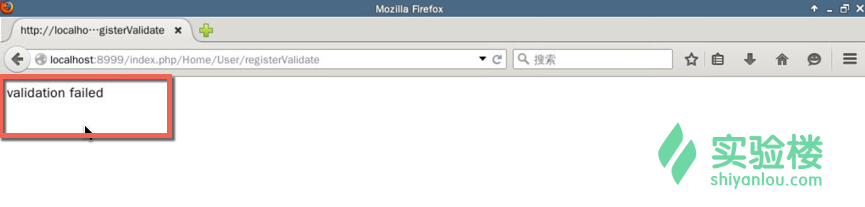
访问：http://localhost:8999/index.php/Home/User/register

分别尝试填写正确的验证码和错误的验证码，你会看到不一样的返回结果。

正确的是这样的：



错误的是这样的：



## **四、课后习题**

1. 实现多文件上传
2. 实现图片的居中裁剪和坐标式裁剪
3. 实现中文验证码