目录

[一、基础 2](#_Toc452367516)

[快速入门（一）：基础 3](#_Toc452367517)

[简介 3](#_Toc452367518)

[官网下载 3](#_Toc452367519)

[目录结构 4](#_Toc452367520)

[运行环境 4](#_Toc452367521)

[入口文件 5](#_Toc452367522)

[调试模式 6](#_Toc452367523)

[控制器 6](#_Toc452367524)

[视图 9](#_Toc452367525)

[[ 新手须知 ] 10](#_Toc452367526)

[读取数据 11](#_Toc452367527)

[总结 13](#_Toc452367528)

[二、URL和路由 15](#_Toc452367529)

[快速入门（二）：URL和路由 15](#_Toc452367530)

[URL请求 15](#_Toc452367531)

[生成URL地址 16](#_Toc452367532)

[四、查询语言 17](#_Toc452367533)

[快速入门（四）：查询语言 17](#_Toc452367534)

[查询表达式 17](#_Toc452367535)

[批量查询 22](#_Toc452367536)

[快捷查询 23](#_Toc452367537)

[获取数值 24](#_Toc452367538)

[获取列数据 24](#_Toc452367539)

[聚合查询 25](#_Toc452367540)

[字符串查询 27](#_Toc452367541)

[快速入门（五）：模型 28](#_Toc452367542)

[（1）模型定义 28](#_Toc452367543)

[模型定义 28](#_Toc452367544)

[设置数据表 31](#_Toc452367545)

[数据库前缀+当前的模型类名（不含命名空间） 31](#_Toc452367546)

[（2）基础操作 31](#_Toc452367547)

[新增数据 31](#_Toc452367548)

[批量新增 33](#_Toc452367549)

[查询数据 34](#_Toc452367550)

[数据列表 36](#_Toc452367551)

[更新数据 38](#_Toc452367552)

[删除数据 39](#_Toc452367553)

[（3）读取器和修改器 40](#_Toc452367554)

[读取器 40](#_Toc452367555)

[get + 属性名的驼峰命名+ Attr 41](#_Toc452367556)

[修改器 42](#_Toc452367557)

[set + 属性名的驼峰命名+ Attr 43](#_Toc452367558)

[（4）类型转换和自动完成 44](#_Toc452367559)

[类型转换 44](#_Toc452367560)

[自动时间戳 45](#_Toc452367561)

[自动完成 46](#_Toc452367562)

[（5）查询范围 49](#_Toc452367563)

[查询范围 49](#_Toc452367564)

[scope + 查询范围名称 50](#_Toc452367565)

[（6）输入和验证 51](#_Toc452367566)

[表单提交 52](#_Toc452367567)

[（8）模型输出 54](#_Toc452367568)

[输出数组 54](#_Toc452367569)

[输出JSON 55](#_Toc452367570)

[快速入门（六）：视图和模板 57](#_Toc452367571)

[模板输出 57](#_Toc452367572)

[分页输出 61](#_Toc452367573)

[模板定位 63](#_Toc452367574)

[输出替换 65](#_Toc452367575)

[七、请求和响应 68](#_Toc452367576)

[输入参数 68](#_Toc452367577)

**一、快速入门（一）：基础**

本篇介绍了ThinkPHP5.0的安装及基本使用，主要包含：

* + [简介](http://www.kancloud.cn/thinkphp/thinkphp5_quickstart/145249#u7B80u4ECB)
  + [官网下载](http://www.kancloud.cn/thinkphp/thinkphp5_quickstart/145249#u5B98u7F51u4E0Bu8F7D)
  + [Composer安装和更新](http://www.kancloud.cn/thinkphp/thinkphp5_quickstart/145249#composer-)
  + [目录结构](http://www.kancloud.cn/thinkphp/thinkphp5_quickstart/145249#u76EEu5F55u7ED3u6784)
  + [运行环境](http://www.kancloud.cn/thinkphp/thinkphp5_quickstart/145249#u8FD0u884Cu73AFu5883)
  + [入口文件](http://www.kancloud.cn/thinkphp/thinkphp5_quickstart/145249#u5165u53E3u6587u4EF6)
  + [调试模式](http://www.kancloud.cn/thinkphp/thinkphp5_quickstart/145249#u8C03u8BD5u6A21u5F0F)
  + [控制器](http://www.kancloud.cn/thinkphp/thinkphp5_quickstart/145249#u63A7u5236u5668)
  + [视图](http://www.kancloud.cn/thinkphp/thinkphp5_quickstart/145249#u89C6u56FE)
  + [读取数据](http://www.kancloud.cn/thinkphp/thinkphp5_quickstart/145249#u8BFBu53D6u6570u636E)
  + [总结](http://www.kancloud.cn/thinkphp/thinkphp5_quickstart/145249#u603Bu7ED3)

简介

ThinkPHP是一个快速、简单的基于MVC和面向对象的轻量级PHP开发框架，遵循Apache2开源协议发布，从诞生以来一直秉承简洁实用的设计原则，在保持出色的性能和至简的代码的同时，尤其注重开发体验和易用性，并且拥有众多的原创功能和特性，为WEB应用开发提供了强有力的支持。

**ThinkPHP5.0版本是一个颠覆和重构版本，也是ThinkPHP十周年献礼版本，基于PHP5.4设计（支持PHP7），采用全新的架构思想，引入了很多的PHP新特性，优化了核心，减少了依赖，支持Composer，实现了真正的惰性加载，并且为API开发做了深入的支持，在性能上尤为突出。**

官网下载

ThinkPHP最新的稳定版本可以在（[官方网站下载页](http://thinkphp.cn/down/framework.html)http://thinkphp.cn/down/framework.html）下载，不过官网下载版本并不是实时更新的，我们会在每个版本更新发布的时候重新打包。

目录结构

安装后（或者下载后的压缩文件解压后）可以看到下面的目录结构：

tp5

├─README.md README文件

├─composer.json Composer定义文件

├─build.php 自动生成定义文件（参考）

├─composer.json composer定义文件

├─console 命令行工具

├─LICENSE.txt 授权说明文件

├─README.md README文件

│

├─vendor第三方类库目录（Composer）

├─extend扩展类库目录（可定义）

├─application应用目录

├─public对外访问目录

└─thinkphp框架目录

tp5所在的目录被定义为系统的ROOT\_PATH常量，其中，thinkphp为框架核心目录，被定义为系统的THINK\_PATH常量，其目录结构如下：

├─thinkphp框架系统目录

│ ├─lang语言包目录

│ ├─library框架核心类库目录

│ │ ├─thinkthink类库包目录

│ │ └─traits系统traits目录

│ ├─mode应用模式目录

│ ├─tpl系统模板目录

│ │

│ ├─.htaccess用于apache的重写

│ ├─.travis.ymlCI定义文件

│ ├─base.php基础定义文件

│ ├─composer.jsoncomposer定义文件

│ ├─convention.php惯例配置文件

│ ├─helper.php助手函数文件（可选）

│ ├─LICENSE.txt授权说明文件

│ ├─phpunit.xml单元测试配置文件

│ ├─README.mdREADME文件

│ └─start.php框架引导文件

运行环境

在开始之前，你需要一个Web服务器和PHP5.4+运行环境，团队在进行tp5开发时，统一使用php5.6开发环境，请下载集成php5.6的XAMPP。

入口文件

ThinkPHP5.0版本的默认自带的入口文件位于public/index.php（实际部署的时候public目录为你的应用唯一对外访问目录），入口文件内容如下：

// 定义应用目录

define('APP\_PATH', \_\_DIR\_\_ . '/../application/');

// 开启调试模式

define('APP\_DEBUG', true);

// 加载框架引导文件

require\_\_DIR\_\_ . '/../thinkphp/start.php';

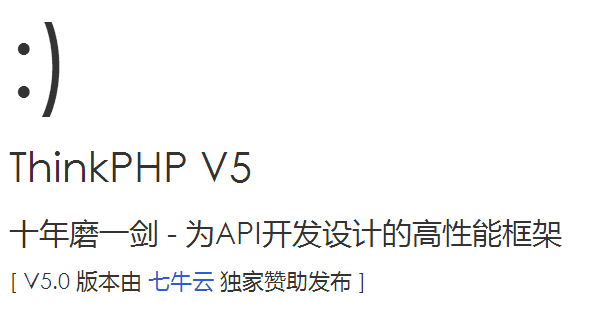
这段代码的作用就是定义应用目录APP\_PATH和加载ThinkPHP框架的入口文件，并且开启了调试模式，这是所有基于ThinkPHP开发应用的第一步。

比如我们的XAMPP安装在D盘，tp5下载后的解压位置为d:\xampp\htdocs\tp5我们可以在浏览器中访问入口文件

http://localhost/tp5/public/

访问上述页面，意味着本机中存在d:\xampp\htdocs\tp5\public\index.php文件，如果正能正常访问，1检查apache是否成功开启，2检查上述路径是否存在index.php文件。

运行后我们会看到欢迎页面：



官方提供的默认应用的实际目录结构和说明如下：

├─application应用目录（可设置）

│ ├─common公共模块目录（可更改）

│ ├─index模块目录(可更改)

│ │ ├─config.php模块配置文件

│ │ ├─common.php模块函数文件

│ │ ├─controller控制器目录

│ │ ├─model模型目录

│ │ └─view视图目录

│ │

│ ├─command.php命令行工具配置文件

│ ├─common.php应用公共（函数）文件

│ ├─config.php应用（公共）配置文件

│ ├─database.php数据库配置文件

│ └─route.php路由配置文件

调试模式

ThinkPHP支持调试模式，默认情况下是开启。调试模式以除错方便优先，关闭任何缓存，而且尽可能多的抛出错误信息，所以对性能有一定的影响。

我们强烈建议开发者在使用ThinkPHP开发的过程中使用调试模式，5.0默认情况下可以捕获到任何细微的错误并抛出异常，这样可以更好的获取错误提示和避免一些问题和隐患。

开发完成后，我们实际进行项目部署的时候，删除这行常量定义代码即可，或者改成：

// 定义应用目录

define('APP\_PATH', \_\_DIR\_\_ . '/../application/');

// 关闭调试模式

define('APP\_DEBUG', false);

// 加载框架引导文件

require\_\_DIR\_\_ . '/../thinkphp/start.php';

以上代码文件位置public/index.php

为了安全考虑，避免泄露你的服务器WEB目录信息等资料，一定记得正式部署的时候关闭调试模式。

控制器

我们找到index模块的Index控制器（文件位于application/index/controller/Index.php），我们把Index控制器类的index方法修改为Hello,World！。

<?php

namespaceapp\index\controller;

classIndex

{

publicfunctionindex()

{

return'Hello,World！';

}

}

访问URL地址

<http://127.0.0.1/tp5/public/index.php>或http://127.0.0.1/tp5/public

就会看到Hello,World！的输出结果。

可以为操作方法定义参数，例如：

<?php

namespaceapp\index\controller;

classIndex

{

publicfunctionindex($name = 'World')

{

return'Hello,' . $name . '！';

}

}

当我们带name参数访问入口文件地址（例如 http://127.0.0.1/tp5/public/?name=ThinkPHP）的时候，在浏览器中可以看到如下输出：

hello,ThinkPHP！

控制器类可以包括多个操作方法，但如果你的操作方法是protected或者private类型的话，是无法直接通过URL访问到该操作的。

我们把Index控制器类的方法修改如下：

<?php

Namespace app\index\controller;

class Index

{

public function hello()

{

return'hello,thinkphp!';

}

public function test()

{

return'这是一个测试方法!';

}

protected function hello2()

{

return'只是protected方法!';

}

private function hello3()

{

return'这是private方法!';

}

}

当我们访问如下URL地址的时候，前面两个是正常访问，后面两个则会显示异常。

http://127.0.0.1/tp5/public/index.php/index/index/hello

http://127.0.0.1/tp5/public/index.php/index/index/test

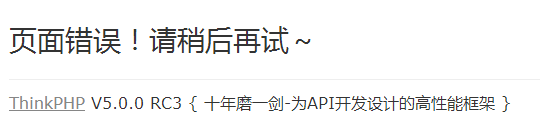
http://127.0.0.1/tp5/public/index.php/index/index/hello2

http://127.0.0.1/tp5/public/index.php/index/index/hello3

当我们访问hello2和hello3操作方法后的结果都会显示类似的异常信息：



异常页面包含了详细的错误信息，是因为开启了调试模式，如果关闭调试模式的话，看到的默认信息如下：



视图

现在我们在给控制器添加视图文件功能，我们在application/index目录下面创建一个view目录，然后添加模板文件view/index/hello.html，我们添加模板内容如下：

<html>

<head>

<title>hello {$name}</title>

</head>

<body>

hello, {$name}!

</body>

</html>

要输出视图，必须在控制器方法中进行模板渲染输出操作，例如：

<?php

namespace app\index\controller;

use think\Controller;

class Index extends Controller

{

public function hello($name = 'thinkphp')

{

$this->assign('name', $name);

return $this->fetch();

}

}

[ 新手须知 ]

这里使用了use来导入一个命名空间的类库，然后可以在当前文件中直接使用该别名而不需要使用完整的命名空间路径访问类库。也就说，如果没有使用

use think\Controller;

就必须使用 class Index extends \think\Controller 这种完整命名空间方式。

在后面的内容中，如果我们直接调用系统的某个类的话，都会假设已经在类的开头使用use进行了别名导入。

注意，Index控制器类继承了 think\Controller类之后，我们可以直接使用封装好的**assign和fetch**方法进行模板变量赋值和渲染输出。

fetch方法中我们没有指定任何模板，所以按照系统默认的规则（**视图目录/控制器/操作方法**）输出了view/index/hello.html模板文件。

接下来，我们在浏览器访问输出：hello,thinkphp!

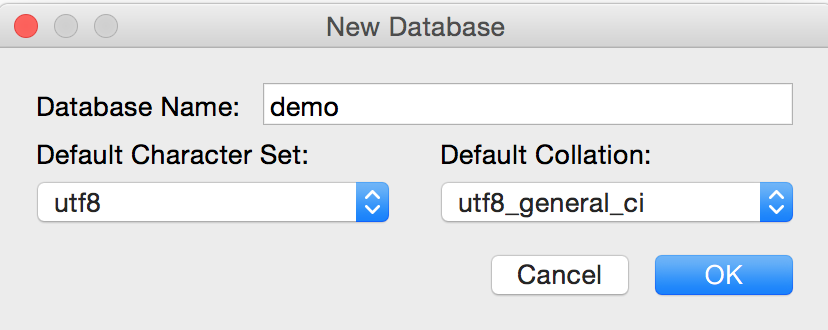
**读取数据**

在开始之前，我们首先在数据库thinkphp中创建一个think\_data数据表（这里以mysql数据库为例）：

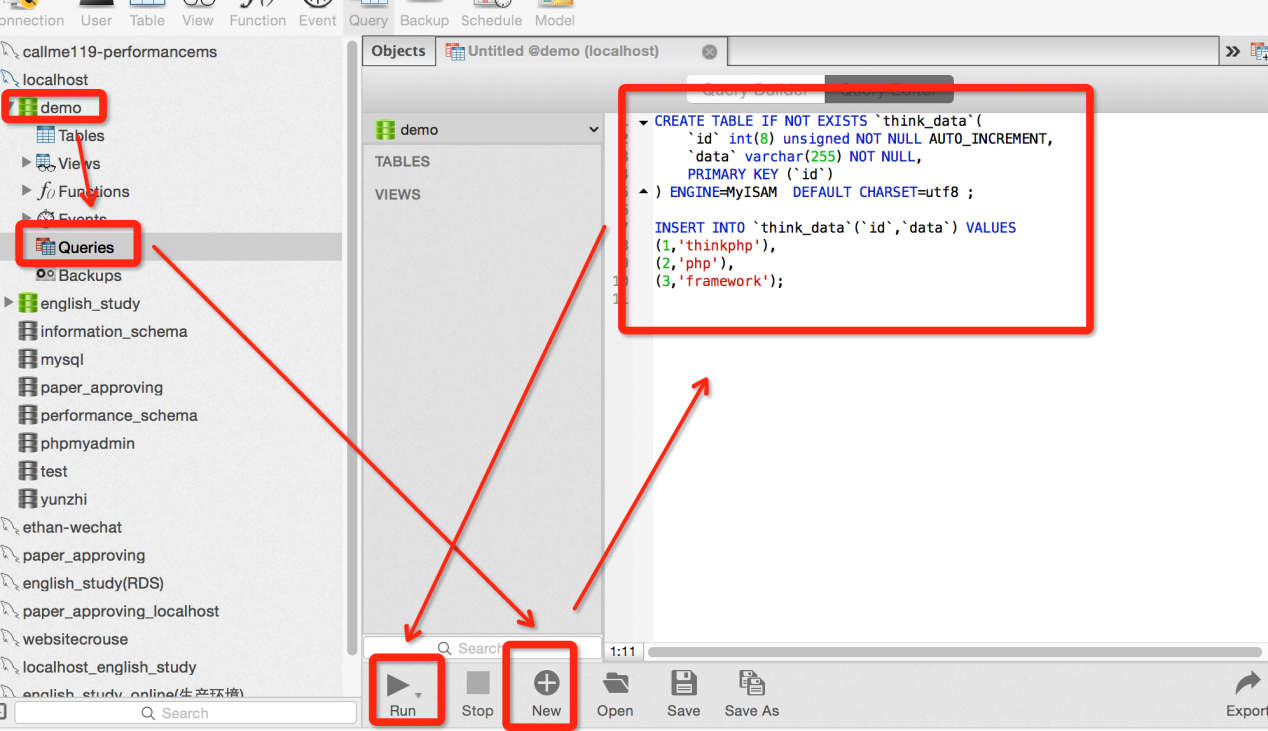
(请下载navicat,并启动xmapp中的mysql服务)

如何新建一个连接，使得navicat能够对mysql进行操作，找自己的小组长吧。

如果不存在数据表，请新建数据表demo.



然后按下图新建查询，并复制相关数据代码：



数据代码如下：

CREATE TABLE IF NOT EXISTS `think\_data`(

`id` int(8) unsigned NOT NULL AUTO\_INCREMENT,

`data` varchar(255) NOT NULL,

PRIMARY KEY (`id`)

) ENGINE=MyISAM DEFAULT CHARSET=utf8 ;

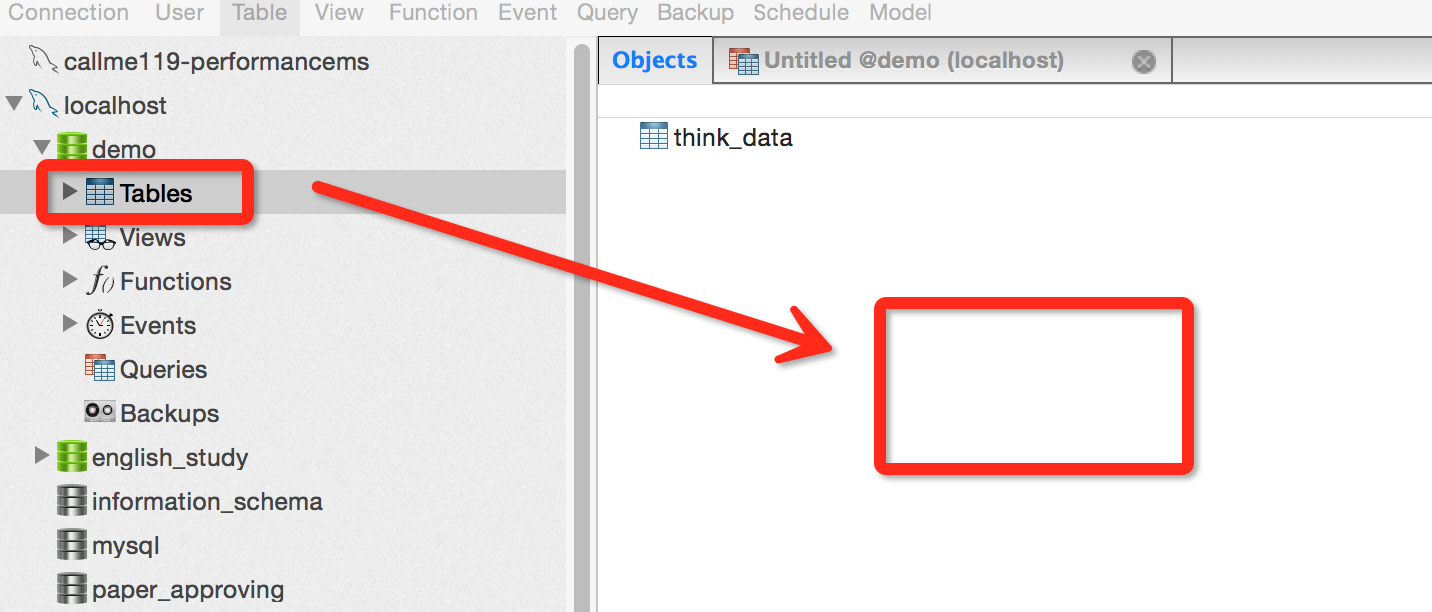
INSERT INTO `think\_data`(`id`,`data`) VALUES

(1,'thinkphp'),

(2,'php'),

(3,'framework');

运行后，点击tables并在右侧进行刷新，显示新建数据表成功:



此时我们需要在应用的数据库配置文件application/database.php中添加数据库的连接信息如下：

return [

// 数据库类型

'type' =>'mysql',

// 服务器地址

'hostname' =>'127.0.0.1',

// 数据库名

'database' =>'demo',

// 数据库用户名

'username' =>'root',

// 数据库密码

'password' =>'',

// 数据库连接端口

'hostport' =>'',

// 数据库连接参数

'params' => [],

// 数据库编码默认采用utf8

'charset' =>'utf8',

// 数据库表前缀

'prefix' =>'think\_',

// 数据库调试模式

'debug' =>true,

];

接下来，我们修改下控制器方法，添加读取数据的代码：

<?php

namespaceapp\index\controller;

usethink\Controller;

use think\Db;

classIndexextendsController

{

publicfunctionindex()

{

$data = Db::name('data')->find();

$this->assign('result', $data);

return$this->fetch();

}

}

定义好控制器后，我们修改模板文件，添加数据输出标签如下：

<html>

<head>

<title></title>

</head>

<body>

{$result.id}--{$result.data}

</body>

</html>

模板标签的用法和Smarty类似，就是用于输出数据的字段，这里就表示输出think\_data表的id和data字段的值。

我们访问会输出：

1--thinkphp

总结

本篇我们学习了如何安装ThinkPHP和框架的目录结构，如何创建项目的入口文件和开启调试模式，以及控制器、模板和数据库的基础认识。

**二、快速入门（二）：URL和路由**

本篇讲解URL访问和路由的使用，主要包含：

* + [URL请求](http://www.kancloud.cn/thinkphp/thinkphp5_quickstart/145250#url-)
  + [生成URL地址](http://www.kancloud.cn/thinkphp/thinkphp5_quickstart/145250#-url--2)

URL请求

ThinkPHP采用单一入口模式访问应用，对应用的所有请求都定向到应用的入口文件，系统会从URL参数中解析当前请求的模块、控制器和操作，下面是一个标准的URL访问格式：

http://serverName/index.php/模块/控制器/操作

应用的index模块的Index控制器定义如下：

<?php

namespace app\index\controller;

class Index

{

public function index()

{

return'index';

}

public function hello($name = 'World')

{

return'Hello,' . $name . '！';

}

}

如果我们直接访问入口文件的话，由于URL中没有模块、控制器和操作，因此系统会访问默认模块（index）下面的默认控制器（Index）的默认操作（index），因此下面的访问是等效的：

http://127.0.0.1/tp5/public/index.php

http://127.0.0.1/tp5/public/index.php/index/index/index

如果你的服务器环境不支持path info方式的URL访问，可以使用兼容方式，例如：

http://127.0.0.1/tp5/public/index.php?s=/index/Index/index

其中变量s的名称的可以配置的。

生成URL地址

定义路由规则之后，我们可以通过Url类来方便的生成实际的URL地址（路由地址），针对上面的路由规则，我们可以用下面的方式生成URL地址。

我们可以使用系统提供的助手函数url来简化

url('blog/read?name=thinkphp');

// 等效于

Url::build('blog/read?name=thinkphp');

如果我们的路由规则发生调整，生成的URL地址会自动变化。

如果你在config.php中配置了url\_html\_suffix参数的话，生成的URL地址会带上后缀，例如：

'url\_html\_suffix' =>'html',

那么生成的URL地址类似

blog/thinkphp.html

blog/2015/05.html

**四、快速入门（四）：查询语言**

本篇带你领略ThinkPHP的查询语法，以及如何轻松构建查询，主要包括：

* + [查询表达式](http://www.kancloud.cn/thinkphp/thinkphp5_quickstart/147280#u67E5u8BE2u8868u8FBEu5F0F)
  + [批量查询](http://www.kancloud.cn/thinkphp/thinkphp5_quickstart/147280#u6279u91CFu67E5u8BE2)
  + [快捷查询](http://www.kancloud.cn/thinkphp/thinkphp5_quickstart/147280#u5FEBu6377u67E5u8BE2)
  + [视图查询](http://www.kancloud.cn/thinkphp/thinkphp5_quickstart/147280#u89C6u56FEu67E5u8BE2)
  + [闭包查询](http://www.kancloud.cn/thinkphp/thinkphp5_quickstart/147280#u95EDu5305u67E5u8BE2)
  + [使用Query对象](http://www.kancloud.cn/thinkphp/thinkphp5_quickstart/147280#-query-)
  + [获取数值](http://www.kancloud.cn/thinkphp/thinkphp5_quickstart/147280#u83B7u53D6u6570u503C)
  + [获取列数据](http://www.kancloud.cn/thinkphp/thinkphp5_quickstart/147280#u83B7u53D6u5217u6570u636E)
  + [聚合查询](http://www.kancloud.cn/thinkphp/thinkphp5_quickstart/147280#u805Au5408u67E5u8BE2)
  + [字符串查询](http://www.kancloud.cn/thinkphp/thinkphp5_quickstart/147280#u5B57u7B26u4E32u67E5u8BE2)
  + [分块查询](http://www.kancloud.cn/thinkphp/thinkphp5_quickstart/147280#u5206u5757u67E5u8BE2)

## 查询表达式

最普通的查询就是判断某个字段是否等于某个值，例如，我们查询think\_data数据表中id等于1的数据，用法如下：

$data = Db::name('data')

->where('id', 1)

->find();

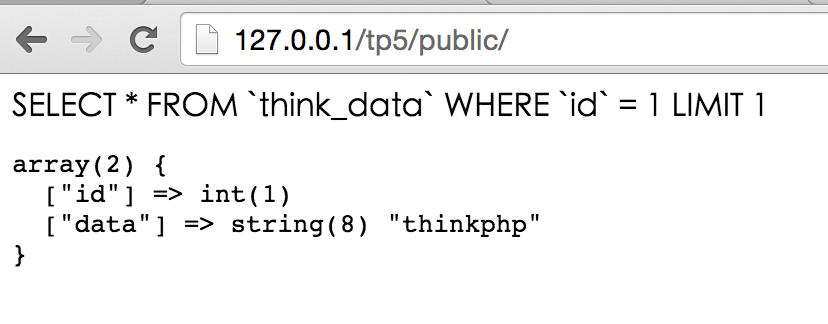
find方法用于查找满足条件第一个记录（即使你的查询条件有多个符合的数据），如果查询成功，返回的是一个一维数组，没有满足条件的话则默认返回false（也支持设置是否抛出异常）。

生成的SQL语句是：

SELECT \* FROM`think\_data`WHERE`id` = 1

实验结果：SELECT \* FROM `think\_data` WHERE `id` = 1 LIMIT 1

示例代码及输出结果如下：



上述的查询其实等同于：

$data = Db::name('data')

->where('id', '=', 1)

->find();

使用表达式查询的时候，where方法的参数依次为：

where( 字段名，条件表达式，查询值 )

可以支持的查询表达式包括如下：

| **表达式** | **含义** |
| --- | --- |
| EQ、= | 等于（=） |
| NEQ、<> | 不等于（<>） |
| GT、> | 大于（>） |
| EGT、>= | 大于等于（>=） |
| LT、< | 小于（<） |
| ELT、<= | 小于等于（<=） |
| LIKE | 模糊查询 |
| [NOT] BETWEEN | （不在）区间查询 |
| [NOT] IN | （不在）IN 查询 |
| [NOT] NULL | 查询字段是否（不）是NULL |
| [NOT] EXISTS | EXISTS查询 |
| EXP | 表达式查询，支持SQL语法 |

其中条件表达式不区分大小写

下面就来查询id大于等于1的数据，使用如下代码：

$data = Db::name('data')

->where('id', '>=', 1)

->limit(10)

->select();

因为这里需要返回多条记录，因此这里我们使用了select方法，并且使用limit方法限制了返回的最多记录数。

select方法用于查询数据集，如果查询成功，返回的是一个二维数组，如果没有满足条件的话则返回空数组（也支持设置是否需要抛出异常）。

生成的SQL语句是：

SELECT \* FROM`think\_data`WHERE`id`>= 1LIMIT10

SELECT \* FROM `think\_data` WHERE `id` >= 1 LIMIT 10



如果使用EXP条件表达式的话，表示后面是原生的SQL语句表达式，例如上面的查询可以改成：

$data = Db::name('data')

->where('id', 'exp', '>= 1')

->limit(10)

->select();

生成的SQL语句和前面是一样的。

SELECT \* FROM `think\_data` WHERE ( `id` >=1 ) LIMIT 10

如果要查询id的范围，可以使用

$data = Db::name('data')

// id 是1、2、3其中的数字

->where('id', 'in', [1, 2, 3])

->select();

生成的SQL语句是：

SELECT \* FROM`think\_data`WHERE`id`IN (1,2,3)

或者

$data = Db::name('data')

// id 在5到8之间的

->where('id', 'between', [5, 8])

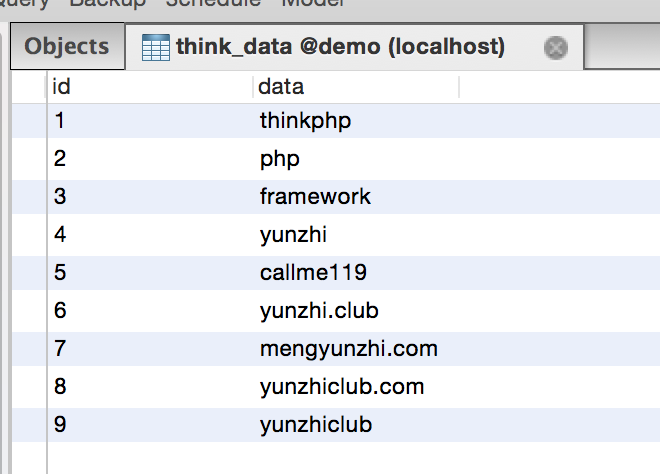
->select();

生成的SQL语句是：

SELECT \* FROM` think\_data`WHERE `id` BETWEEN 5 AND 8

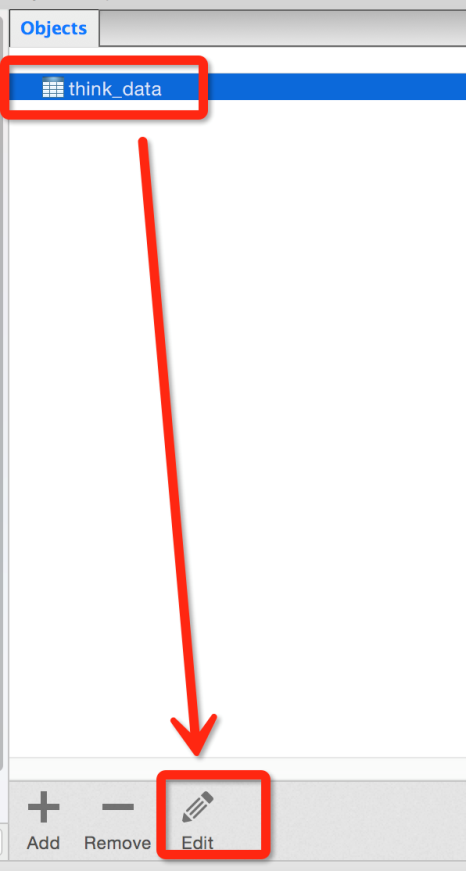
什么？dump后数据为空.如果进行到这步，还没有想到需要去数据表中添加测试数据的话，那么说明你需要更多的努力，因为我们的基础差决定了我们现在的天赋还少了一些。

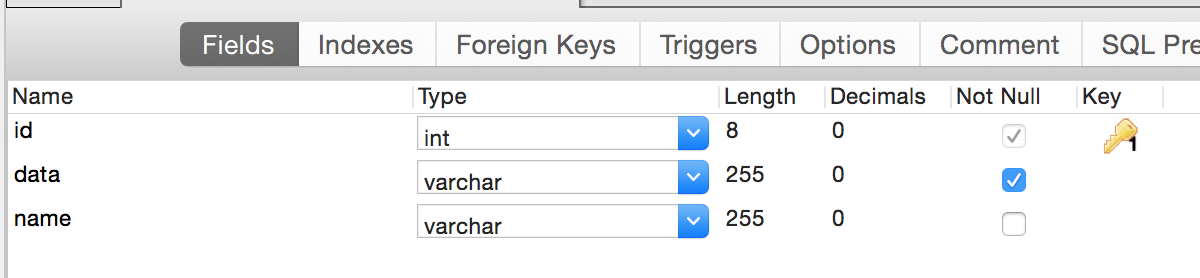
添加测试数据后的think\_data表至少是这样的.



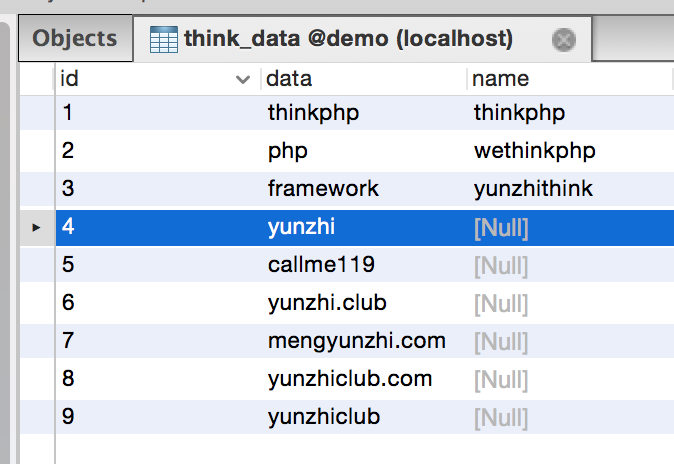
接下来，使用多个字段查询：

我们需要重新规划think\_data表，添加name字段，并且添加相应的测试数据，必要时，需要对测试数据进行修改.





（注意：name字段：not null不勾上，为后续判是否为null做基础，见22页-批量查询前的“查询某个字段是否为NULL”示例；否则不填内容将默认为””,是有值的）



$data = Db::name('data')

// id 在 1到3之间的

->where('id', 'between', [1, 3])

// name 中包含think

->where('name', 'like', '%think%')

->select();

这样生成的查询语句为：

SELECT \* FROM`think\_data`WHERE`id`BETWEEN5AND8AND`name`LIKE'%think%'

如果要查询某个字段是否为NULL，可以使用：

$data = Db::name('data')

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

->select();

这样生成的查询语句为：

SELECT \* FROM `think\_data` WHERE `name` IS NULL

## 批量查询

我们可以使用一个方法完成多个查询条件，例如上面的查询可以改成：

$data = Db::name('data')

->where([

'id' => ['between', '1,3'],

'name' => ['like', '%think%'],

])->select();

这样生成的查询语句还是和之前一样：

SELECT \* FROM`think\_data`WHERE`id`BETWEEN'1'AND'3'AND`name`LIKE'%think%'

我们再来看一些复杂的用法，使用OR和AND混合条件查询，例如：

$data = Db::name('data')

// name 中包含think

->where('name', 'like', '%think%')

->where('id', ['in', [1, 2, 3]], ['between', '5,8'], 'or')

->limit(10)

->select();

或者使用批量方式：

$data = Db::name('data')

->where([

'id' => [['in', [1, 2, 3]], ['between', '5,8'], 'or'],

'name' => ['like', '%think%'],

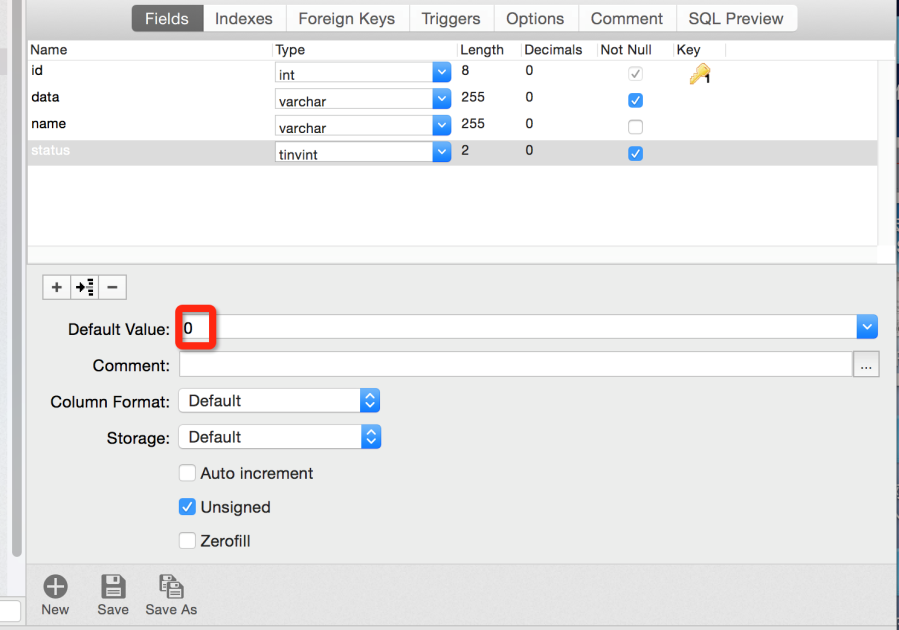
])->limit(10)->select();

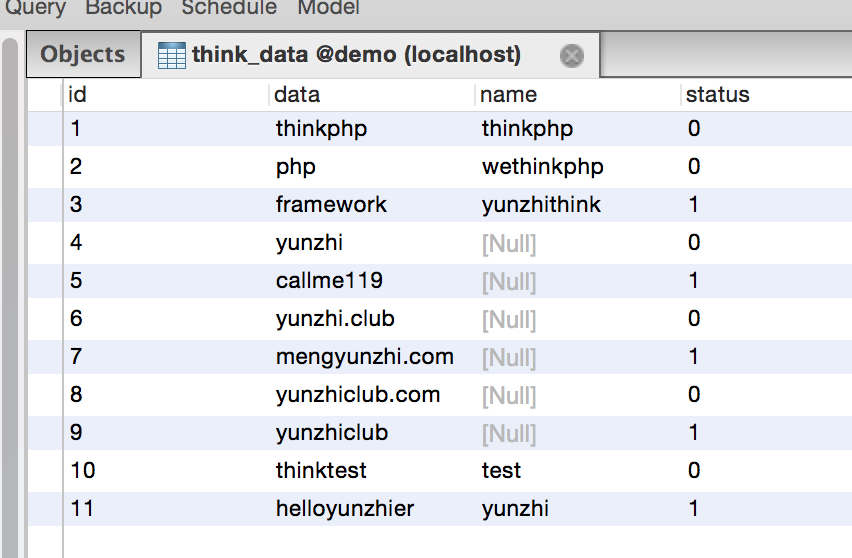
生成的SQL语句为：

SELECT \* FROM`think\_data`WHERE ( `id`IN (1,2,3) or`id`BETWEEN'5'AND'8' ) AND`name`LIKE'%think%'LIMIT10

## 快捷查询

我们在数据表中，再增加一个状态status字段，并将默认值设置为0,且适当的进行测试数据的添加,状态值随机进行0，1的设置，并将记录条数增加至11条.





如果你有多个字段需要使用相同的查询条件，可以使用快捷查询。例如，我们要查询id和status都大于0的数据，可以使用：

$data = Db::name('data')

->where('id&status', '>', 0)

->limit(5)

->select();

生成的SQL语句为：

SELECT \* FROM`think\_data`WHERE ( `id`>0AND`status`>0 ) LIMIT5

也可以使用or方式查询，例如：

$data = Db::name('data')

->where('id|status', '>', 0)

->limit(5)

->select();

生成的SQL语句为：

SELECT \* FROM`think\_data`WHERE ( `id`>0OR`status`>0 ) LIMIT5

## 获取数值

如果仅仅是需要获取某行表的某个值，可以使用value方法：

// 获取id为8的data数据的name字段值

$name = Db::name('data')

->where('id', 8)

->value('name');

注意：相应的修改查询语句后返回语句

//查看返回数据

dump($name);

name的结果为：null

## 获取列数据

也支持获取某个列的数据，使用column方法，例如：

// 获取data表的name列

$list = Db::name('data')

->where('status', 1)

->column('name');

注意：相应的修改查询语句后返回语句

//查看返回数据

dump($list);

返回的结果类似下面：

array (size=5)

0 => string 'thinkphp'

1 => string 'onethink'

2 => string 'topthink'

3 => string 'kancloud'

4 => string 'thinkphp'

如果希望返回以id为索引的name列数据，可以改成：

// 获取data表的name列并且以id为索引

$list = Db::name('data')

->where('status', 1)

->column('name', 'id');

返回的结果类似下面：

array(5) {

[3] => string(11) "yunzhithink"

[5] => NULL

[7] => NULL

[9] => NULL

[11] => string(6) "yunzhi"

}

## 聚合查询

thinkphp为聚合查询提供了更便捷的方法，如下：

// 统计data表的数据

$count = Db::name('data')

->where('status', 1)

->count();

为此，我们新建think\_user表。需要在mysql的查询中，运行以下语句:

SET NAMES utf8;

SET FOREIGN\_KEY\_CHECKS = 0;

-- ----------------------------

-- Table structure for `think\_user`

-- ----------------------------

DROP TABLE IF EXISTS `think\_user`;

CREATE TABLE `think\_user` (

`id` int(8) unsigned NOT NULL AUTO\_INCREMENT,

`name` varchar(30) DEFAULT NULL,

`status` tinyint(2) unsigned DEFAULT '0',

`score` int(2) DEFAULT '0',

PRIMARY KEY (`id`)

) ENGINE=MyISAM AUTO\_INCREMENT=14 DEFAULT CHARSET=utf8;

-- ----------------------------

-- Records of `think\_user`

-- ----------------------------

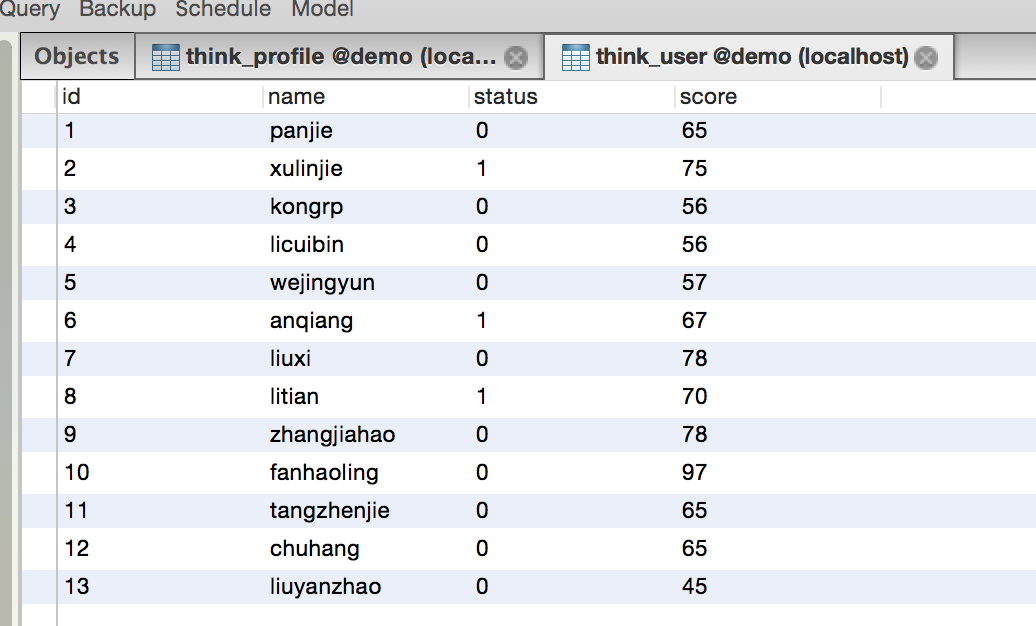
BEGIN;

INSERT INTO `think\_user` VALUES ('1', 'panjie', '0', '65'), ('2', 'xulinjie', '1', '75'), ('3', 'kongrp', '0', '56'), ('4', 'licuibin', '0', '56'), ('5', 'wejingyun', '0', '57'), ('6', 'anqiang', '1', '67'), ('7', 'liuxi', '0', '78'), ('8', 'litian', '1', '70'), ('9', 'zhangjiahao', '0', '78'), ('10', 'fanhaoling', '0', '97'), ('11', 'tangzhenjie', '0', '65'), ('12', 'chuhang', '0', '65'), ('13', 'liuyanzhao', '0', '45');

COMMIT;

SET FOREIGN\_KEY\_CHECKS = 1;

则新建think\_user表，且同步插入数据如下：



// 统计user表的最高分

$count = Db::name('user')

->where('status', 1)

->max('score');

支持的聚合查询方法包括：

| **方法** | **说明** | **参数** |
| --- | --- | --- |
| count | 统计数量 | 统计的字段名（可选） |
| max | 获取最大值 | 统计的字段名（必须） |
| min | 获取最小值 | 统计的字段名（必须） |
| avg | 获取平均值 | 统计的字段名（必须） |
| sum | 获取总分 | 统计的字段名（必须） |

## 字符串查询

在必要的时候，仍然可以使用原生的字符串查询，但建议是**配合参数绑定**一起使用，可以避免注入问题，例如：

$data = Db::name('data')

->where('id > :id AND name IS NOT NULL', ['id' =>5])

->select();

可以直接在where方法中使用字符串查询条件，并支持第二个参数传入参数绑定，上面这个查询生成的SQL语句是：

SELECT \* FROM`think\_data`WHERE ( id>'5'ANDnameISNOTNULL )

**五、快速入门（五）：模型**

本章主要帮助大家学习和掌握模型的使用方法。

ThinkPHP5.0的模型是一种对象-关系映射（Object/Relation Mapping，简称ORM）的封装，并且提供了简洁的ActiveRecord实现。一般来说，每个数据表会和一个“模型”对应。

ORM的基本特性就是表映射到记录，记录映射到对象，字段映射到对象属性。模型是一种对象化的操作封装，而不是简单的CURD操作，简单的CURD操作直接使用前面提过的Db类即可。

CURD: CURD是一个数据库技术中的缩写词，一般的项目开发的各种参数的基本功能都是CURD。它代表创建（Create）、更新（Update）、读取（Read）和删除（Delete）操作。CURD 定义了用于处理数据的基本[原子操作](http://baike.baidu.com/view/809659.htm)。

模型类和Db类的区别主要在于对象的封装，Db类的查询默认返回的是数组（或者集合），而模型类返回的是当前的模型对象实例（或者集合）。

# （1）模型定义

为了更好的理解，我们首先在数据库对think\_user表进行重新规划：

DROP TABLE IF EXISTS `think\_user`;

CREATE TABLE `think\_user`(

`id`int(8) unsigned NOT NULL AUTO\_INCREMENT,

`nickname` varchar(50) NOT NULL COMMENT'昵称',

`email` varchar(255) NULL DEFAULT NULL COMMENT '邮箱',

`birthday` int(11) UNSIGNED NOT NULL DEFAULT '0' COMMENT'生日',

`status` tinyint(2) NOT NULL DEFAULT '0' COMMENT'状态',

`create\_time` int(11) UNSIGNED NOT NULL DEFAULT '0' COMMENT'注册时间',

`update\_time` int(11) UNSIGNED NOT NULL DEFAULT '0' COMMENT'更新时间',

PRIMARY KEY (`id`)

) ENGINE=MyISAM DEFAULT CHARSET=utf8 ;

数据库配置文件定义如下：(application/database.php)

return [

// 数据库类型

'type' =>'mysql',

// 服务器地址

'hostname' =>'127.0.0.1',

// 数据库名

'database' =>'demo',

// 数据库用户名

'username' =>'root',

// 数据库密码

'password' =>'',

// 数据库连接端口

'hostport' =>'',

// 数据库连接参数

'params' => [],

// 数据库编码默认采用utf8

'charset' =>'utf8',

// 数据库表前缀

'prefix' =>'think\_',

// 数据库调试模式

'debug' =>true,

];

并添加路由定义（application/route.php）如下：

return [

// 全局变量规则定义

'\_\_pattern\_\_' => [

'id' =>'\d+',

],

'user/index' =>'index/user/index',

'user/create' =>'index/user/create',

'user/add' =>'index/user/add',

'user/add\_list' =>'index/user/addList',

'user/update/:id' =>'index/user/update',

'user/delete/:id' =>'index/user/delete',

'user/:id' =>'index/user/read',

];

我们为think\_user表定义一个模型User如下：

如果到这里，我们还没有弄清楚以下代码的文件位置，那么需要针对命名空间这个东西，单独的去找其它成员或小组长进行深入的学习。

namespace app\index\model;

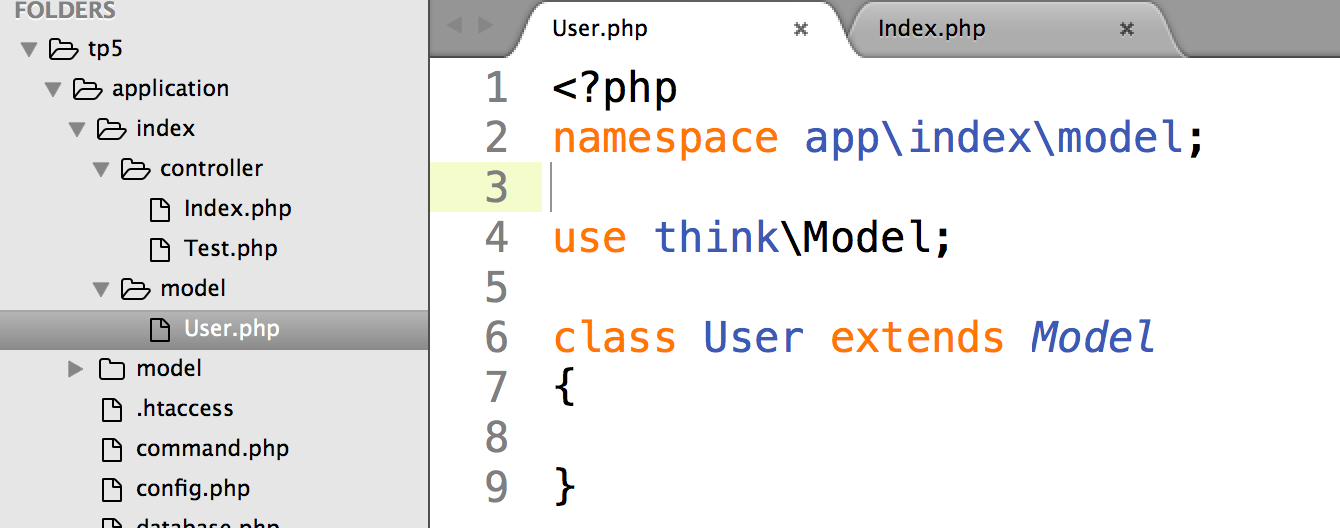
use think\Model;

class User extends Model

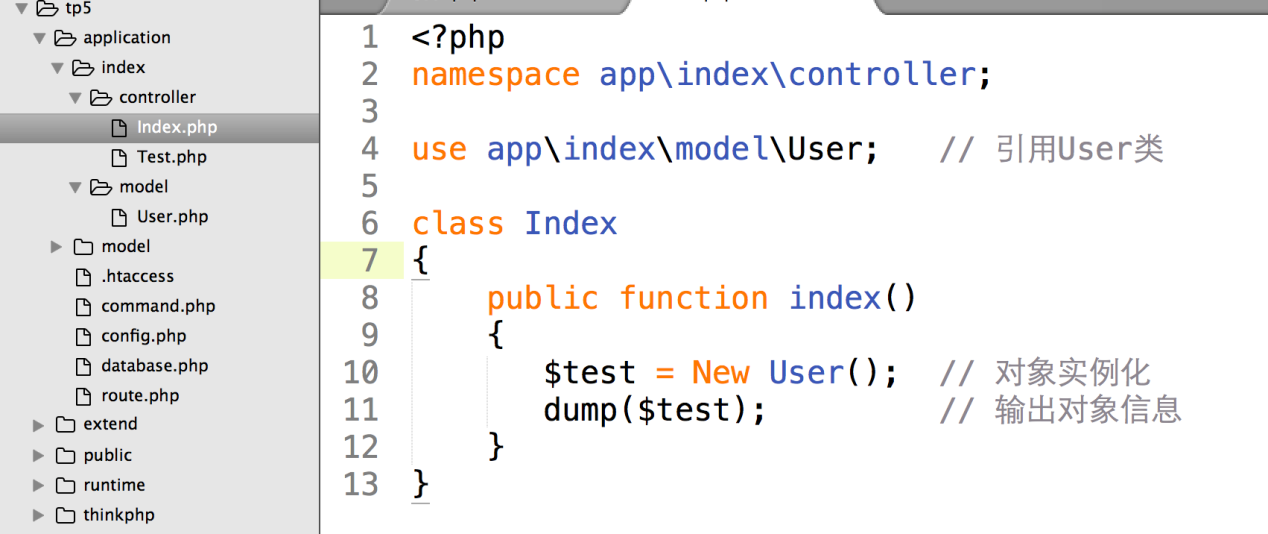
{

}

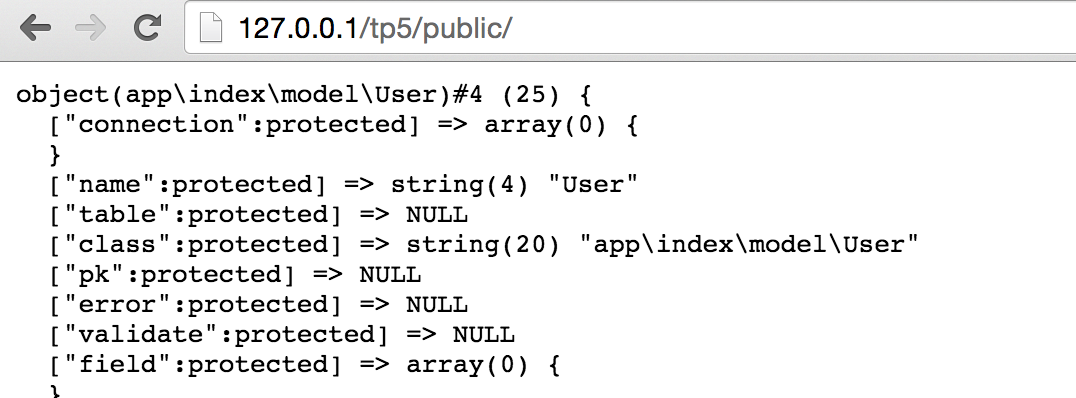
代码截图如下：



大多情况下，我们无需为模型定义任何的属性和方法即可完成基础的操作。

测试代码：

正确的测试返回数据



## 设置数据表

模型会自动对应一个数据表，规范是：数据库前缀+当前的模型类名（不含命名空间），比如本例中数据库前缀为think\_当前模型类名为User,则对应数据表为think\_user(包括有大写字母的，自动进行转换，所以是think\_user而不是think\_User)

# （2）基础操作

* + [新增数据](http://www.kancloud.cn/thinkphp/thinkphp5_quickstart/147285#u65B0u589Eu6570u636E)
  + [批量新增](http://www.kancloud.cn/thinkphp/thinkphp5_quickstart/147285#u6279u91CFu65B0u589E)
  + [查询数据](http://www.kancloud.cn/thinkphp/thinkphp5_quickstart/147285#u67E5u8BE2u6570u636E)
  + [数据列表](http://www.kancloud.cn/thinkphp/thinkphp5_quickstart/147285#u6570u636Eu5217u8868)
  + [更新数据](http://www.kancloud.cn/thinkphp/thinkphp5_quickstart/147285#u66F4u65B0u6570u636E)
  + [删除数据](http://www.kancloud.cn/thinkphp/thinkphp5_quickstart/147285#u5220u9664u6570u636E)

新增数据

我们先来看下如何写入模型数据，创建一个User控制器并增加add操作方法如下：

<?php

Namespace app\index\controller;

use app\index\model\User as UserModel; // 设置别名

class User

{

// 新增用户数据

public function add()

{

$user = new UserModel;

$user->nickname = '流年';

$user->email = 'thinkphp@qq.com';

$user->birthday = strtotime('1977-03-05');

if ($user->save()) {

return '用户[ ' . $user->nickname . ':' . $user->id . ' ]新增成功';

}

else {

return $user->getError();

}

}

}

这里有一个地方要注意，在当前文件中给app\index\model\User模型定义了一个别名UserModel是为了避免和当前的app\index\controller\User产生冲突，如果你当前的控制器类不是User的话可以不需要定义UserModel别名。

接下来，我们访问

http://127.0.0.1/tp5/public/user/add

如果看到输出

用户[ 流年:1 ]新增成功 //表示用户模型写入成功了。

如果你觉得上面的方式给User对象一个个赋值太麻烦，可以改为下面的方式：

// 新增用户数据

public function add()

{

$user['nickname'] = '看云';

$user['email'] = 'kancloud@qq.com';

$user['birthday'] = strtotime('2015-04-02');

if ($result = UserModel::create($user)) {

return'用户[ ' . $result->nickname . ':' . $result->id . ' ]新增成功';

} else {

return'新增出错';

}

}

create方法可以传入数组或者标准对象，你可以在外部统一赋值后传入，当然也可以直接传入表单数据（我们后面会有专门的描述）。

我们刷新刚才的访问地址后，页面输出结果为：

用户[ 看云:2 ]新增成功

批量新增

也可以直接进行数据的批量新增，给控制器添加如下addList操作方法：

<?php

namespace app\index\controller;

use app\index\model\User as UserModel; // 设置别名

// 批量新增用户数据

public function addList()

{

$user = new UserModel;

$list = [

['nickname' =>'张三', 'email' =>'zhanghsan@qq.com', 'birthday' => strtotime('1988-01-15')],

['nickname' =>'李四', 'email' =>'lisi@qq.com', 'birthday' => strtotime('1990-09-19')],

];

if ($user->saveAll($list)) {

return '用户批量新增成功';

}

else

{

return $user->getError();

}

}

访问URL地址

注意是add\_list而不是addList

http://127.0.0.1/tp5/public/user/add\_list

最后的输出结果为：

用户批量新增成功

为什么不是addList呢？我们说addList才是function名字呀。

这是由于我们在前面添加了路由：

return [

// 全局变量规则定义

'\_\_pattern\_\_' => [

'id' =>'\d+',

],

'user/index' =>'index/user/index',

'user/create' =>'index/user/create',

'user/add' =>'index/user/add',

'user/add\_list' =>'index/user/addList',

'user/update/:id' =>'index/user/update',

'user/delete/:id' =>'index/user/delete',

'user/:id' =>'index/user/read',

];

红色字体的意思是：当用户输入user/add\_list时，我们为其指向为index模块下的user控制器下的addList方法

查询数据

接下来添加User模型的查询功能，给User控制器增加如下read操作方法：

// 读取用户数据

public function read($id='')

{

$user = UserModel::get($id);

echo $user->nickname . '<br/>';

echo $user->email . '<br/>';

echo date('Y/m/d', $user->birthday) . '<br/>';

}

模型的get方法用于获取数据表的数据并返回当前的模型对象实例，通常只需要传入主键作为参数，如果没有传入任何值的话，则表示获取第一条数据。

访问如下URL地址

http://127.0.0.1/tp5/public/user/1

输出结果是：

流年

thinkphp@qq.com

1977/03/05

访问如下URL地址

http://127.0.0.1/tp5/public/user/2

输出结果是：

看云

kancloud@qq.com

2016/04/02

模型的get方法和Db类的find方法返回结果的区别在于，Db类默认返回的只是数组（注意这里说的默认，其实仍然可以设置为对象），而模型的get方法查询返回的一定是当前的模型对象实例。

如果上面这段话理解不了，说明面向对象的思想，还没有形成，没关系，这并不影响我们前期开发，而且只要我们想理解，有一天你会发现自己就真的理解了。

但是系统为模型实现了ArrayAccess接口，因此仍然可以通过数组的方式访问对象实例，把控制器的read操作方法改成如下：

// 读取用户数据

public function read($id = '')

{

$user = UserModel::get($id);

echo $user['nickname'] . '<br/>';

echo $user['email'] . '<br/>';

echo date('Y/m/d', $user['birthday']) . '<br/>';

}

再次访问URL地址

http://127.0.0.1/tp5/public/user/1

不但没有报错，而且最终的输出结果和之前是一样的：

流年

thinkphp@qq.com

1977/03/05

如果我想通过用户的email来查询模型数据的话，应该如何操作呢？  
下面是一个查询的例子：

// 根据email读取用户数据

public function read()

{

$user = UserModel::getByEmail('thinkphp@qq.com');

echo $user->nickname . '<br/>';

echo $user->email . '<br/>';

echo date('Y/m/d', $user->birthday) . '<br/>';

}

访问如下URL地址：

<http://127.0.0.1/tp55/public/user/1>

或者：http://127.0.0.1/tp55/public/index/user/read

原因，打开route.php



输出的结果是：

流年

thinkphp@qq.com

1977/03/05

如果不是根据主键查询的话，可以传入数组作为查询条件，例如：

// 根据nickname读取用户数据

public function read()

{

$user = UserModel::get(['nickname'=>'流年']);

echo $user->nickname . '<br/>';

echo $user->email . '<br/>';

echo date('Y/m/d', $user->birthday) . '<br/>';

}

更复杂的查询则可以使用查询构建器来完成，例如：

// 根据nickname读取用户数据

public function read()

{

$user = UserModel::where('nickname', '流年')->find();

echo $user->nickname . '<br/>';

echo $user->email . '<br/>';

echo date('Y/m/d', $user->birthday) . '<br/>';

}

数据列表

如果要查询多个数据，可以使用模型的all方法，我们在控制器中添加index操作方法用于获取用户列表：

// 获取用户数据列表

public function index()

{

$list = UserModel::all();

foreach ($list as $user) {

echo $user->nickname . '<br/>';

echo $user->email . '<br/>';

echo date('Y/m/d', $user->birthday) . '<br/>';

echo'----------------------------------<br/>';

}

}

然后访问

http://127.0.0.1/tp5/public/user/index

就可以看到输出结果为：

流年

thinkphp@qq.com

1977/03/05

-------------------------------------

看云

kancloud@qq.com

2015/04/02

-------------------------------------

张三

zhanghsan@qq.com

1988/01/15

-------------------------------------

李四

lisi@qq.com

1990/09/19

-------------------------------------

如果不是使用主键查询，可以直接传入数组条件查询，例如：

// 获取用户数据列表

public function index()

{

$list = UserModel::all(['status'=>1]);

foreach ($list as $user) {

echo $user->nickname . '<br/>';

echo $user->email . '<br/>';

echo date('Y/m/d', $user->birthday) . '<br/>';

echo'----------------------------------<br/>';

}

}

我们也可以使用数据库的查询构建器完成更多的条件查询，例如：

// 获取用户数据列表

public function index()

{

$list = UserModel::where('id','<',3)->select();

foreach ($listas$user) {

echo $user->nickname . '<br/>';

echo $user->email . '<br/>';

echo date('Y/m/d', $user->birthday) . '<br/>';

echo'----------------------------------<br/>';

}

}

刷新页面访问输出的结果是：

流年

thinkphp@qq.com

1977/03/05

-------------------------------------

看云

kancloud@qq.com

2015/04/02

-------------------------------------

更新数据

我们可以对查询出来的数据进行更新操作，下面添加一个update操作方法：

// 更新用户数据

public function update($id)

{

$user = UserModel::get($id);

$user->nickname = '刘晨';

$user->email = 'liu21st@gmail.com';

if (false !== $user->save()) {

return'更新用户成功';

} else {

return $user->getError();

}

}

访问下面的URL地址

http://127.0.0.1/tp5/public/user/update/1

会输出

更新用户成功

然后我们再次访问

http://127.0.0.1/tp5/public/user/1

会看到输出结果变成：

刘晨

liu21st@gmail.com

1977/03/05

说明我们的更新操作已经生效了。

默认情况下，查询模型数据后返回的模型示例执行save操作都是执行的数据库update操作，如果你需要实例化执行save执行数据库的insert操作，请确保在save方法之前调用isUpdate方法：

$user->isUpdate(false)->save();

原来add和insert是对应的。edit和update是对应的。

ActiveRecord模式的更新数据方式需要首先读取对应的数据，如果需要更高效的方法可以把update方法改成：

// 更新用户数据

public function update($id)

{

$user['id'] = (int) $id;

$user['nickname'] = '刘晨';

$user['email'] = 'liu21st@gmail.com';

$result = UserModel::update($user);

if (false !== $result) {

return'更新用户成功';

} else {

return $user->getError();

}

}

删除数据

我们给User控制器添加delete方法用于删除用户。

// 删除用户数据

public function delete($id)

{

$user = UserModel::get($id);

if ($user) {

$user->delete();

return'删除用户成功';

} else {

return'删除的用户不存在';

}

}

然后访问

http://127.0.0.1/tp5/public/user/delete/1

输出结果为：

删除用户成功

如果刷新页面后输出结果为：

删除的用户不存在

同样我们也可以直接使用destroy方法删除模型数据，例如把上面的delete方法改成如下：

// 删除用户数据

public function delete($id)

{

$result = UserModel::destroy($id);

if ($result) {

return'删除用户成功';

} else {

return'删除的用户不存在';

}

}

目前为止，你已经掌握了最基本的模型操作，后面会引申一些高级的用法。

# （3）读取器和修改器

## 读取器

前面读取用户生日的时候，使用了date方法进行日期的格式处理输出，但是每次读取数据后都需要这样处理就显得非常麻烦。

使用读取器功能就可以简化类似的数据处理操作，例如，我们给User模型添加读取器的定义方法。

<?php

namespace app\index\model;

use think\Model;

class User extends Model

{

// birthday读取器

protected function getBirthdayAttr($birthday)

{

return date('Y-m-d', $birthday);

}

}

这里，我们添加了一个getBirthdayAttr读取器方法用于读取User模型的birthday属性的值，该方法会在读取birthday属性值的时候自动执行。

读取器方法的命名规范是：

get + 属性名的驼峰命名+ Attr

所以，getBirthdayAttr读取器读取的是birthday属性，而getUserBirthdayAttr 读取器读取的则是user\_birthday属性。

定义完修改器后，修改控制器的read操作方法如下：

// 读取用户数据

public function read($id='')

{

$user = UserModel::get($id);

echo $user->nickname . '<br/>';

echo $user->email . '<br/>';

echo $user->birthday . '<br/>';

}

访问URL地址

http://127.0.0.1/tp5/public/user/1

最后的输出结果为：

流年

thinkphp@qq.com

1977-03-05

读取器还可以定义读取数据表中不存在的属性，例如把原始生日和转换的格式分开两个属性birthday和user\_birthday，我们只需定义user\_birthday属性的读取器方法：

<?php

namespace app\index\model;

use think\Model;

class User extends Model

{

// user\_birthday读取器

protected function getUserBirthdayAttr($value,$data)

{

return date('Y-m-d', $data['birthday']);

}

}

这里的读取器方法使用了第二个参数，表示传入所有的属性数据。因为原始的user\_birthday属性数据是不存在的，所以我们需要通过data参数获取。

read操作方法修改为：

// 读取用户数据

public function read($id='')

{

$user = UserModel::get($id);

echo $user->nickname . '<br/>';

echo $user->email . '<br/>';

echo $user->birthday . '<br/>';

echo $user->user\_birthday . '<br/>';

}

当刷新页面的时候，最终输出的结果为：

流年

thinkphp@qq.com

226339200

1977-03-05

## 修改器

由于birthday属性是时间戳格式的，因此我们必须在写入数据前进行时间戳转换，前面使用的方法是每次赋值的时候进行转换处理：

$user['birthday'] = strtotime('2015-04-02');

为了避免每次都进行日期格式的转换操作，可以定义修改器方法来自动处理，修改User模型如下：

<?php

namespace app\index\model;

use think\Model;

class User extends Model

{

// 读取器

protected function getUserBirthdayAttr($birthday, $data)

{

return date('Y-m-d', $data['birthday']);

}

// birthday修改器

protected function setBirthdayAttr($value)

{

return strtotime($value);

}

}

修改器方法的命名规范是：

set + 属性名的驼峰命名+ Attr

所以，setBirthdayAttr方法修改的是birthday属性，而setUserBirthdayAttr 方法修改的则是user\_birthday属性。

控制器的add操作方法修改如下：

// 新增用户数据

public function add()

{

$user = new UserModel;

$user->nickname = '流年';

$user->email = 'thinkphp@qq.com';

$user->birthday = '1977-03-05';

if ($user->save()) {

return'用户[ ' . $user->nickname . ':' . $user->id . ' ]新增成功';

} else {

return $user->getError();

}

}

访问URL地址：

http://127.0.0.1/tp5/public/user/add

最后的输出结果为：

用户[ 流年:10 ]新增成功

接着我们访问

http://127.0.0.1/tp5/public/user/10

页面输出结果为：

流年

thinkphp@qq.com

1977-03-05

通过定义修改器和读取器，完成了时间戳方式存储的birthday属性的写入和读取的自动处理。

TODO：thinkphp3.2.3的老成员们，以入我们再也不能简单的用save($\_POST)来完成数据的保存和更新操作了。

# （4）类型转换和自动完成

类型转换

对于前面的时间戳birthday的例子，还可以进行进一步的简化，这里需要用到类型强制转换的功能，在User模型类中添加定义：

<?php

namespace app\index\model;

use think\Model;

class User extends Model

{

protected $dateFormat = 'Y/m/d';

protected $type = [

'birthday' => 'datetime',

];

}

不需要定义任何修改器和读取器，我们完成了相同的功能。

对于datetime类型，如果不设置dateFormat属性，默认的日期显示格式为：Y-m-d H:i:s。

ThinkPHP5.0支持的自动转换类型包括：

| **类型** | **描述** |
| --- | --- |
| integer | 整形 |
| float | 浮点型 |
| boolean | 布尔型 |
| array | 数组 |
| json | JSON类型 |
| object | 对象 |
| datetime | 日期（整形） |
| timestamp | 时间戳 |

如果没有设置任何类型转换的话，属性默认是字符串类型或者使用写入和读取的原始类型，更多关于类型转换的描述请参考完全开发手册。

自动时间戳

在数据库配置文件中添加设置：

// 开启自动写入时间戳字段

'auto\_timestamp' =>true,

再次访问

http://127.0.0.1/tp5/public/user/add

会发现系统已经自动写入了think\_user数据表中的的create\_time、update\_time字段，如果自动写入的时间戳字段不是这两个的话，需要修改模型类的属性定义，例如：

<?php

namespace app\index\model;

use think\Model;

class User extends Model

{

// 定义类型转换

protected $type = [

'birthday' => 'datetime:Y/m/d',

];

// 定义时间戳字段名

protected $createTime = 'create\_at';

protected $updateTime = 'update\_at';

}

如果个别数据表不需要自动写入时间戳字段的话，也可以在模型里面直接关闭：

<?php

namespace app\index\model;

use think\Model;

class User extends Model

{

// 定义类型转换

protected $type = [

'birthday' => 'datetime:Y/m/d',

];

// 关闭自动写入时间戳

protected $autoWriteTimestamp = false;

}

关闭自动写入时间戳后，我们再次访问URL地址：

http://127.0.0.1/tp5/public/user/add

重新生成的数据已经没有自动写入时间戳了，而是数据库默认值写入。

自动完成

系统已经自动写入了think\_user数据表中的的create\_time、update\_time字段，如果我们希望自动写入其它的字段，则可以使用自动完成功能，例如下面实现新增的时候自动写入status字段。

<?php

namespace app\index\model;

use think\Model;

class User extends Model

{

// 定义类型转换

protected $type = [

'birthday' => 'datetime:Y/m/d',

'create\_time' => 'datetime',

'update\_time' => 'datetime',

];

// 定义自动完成的属性

protected $insert = ['status' =>1];

}

除了insert属性之外，自动完成共有三个属性定义，分别是：

| **属性** | **描述** |
| --- | --- |
| auto | 新增及更新的时候自动完成的属性数组 |
| insert | 仅新增的时候自动完成的属性数组 |
| update | 仅更新的时候自动完成的属性数组 |

自动完成属性里面一般来说仅仅需要定义属性的名称，然后配合修改器或者类型转换来一起完成，如果写入的是一个固定的值，就无需使用修改器。status属性的自动写入可以直接使用：

'status' =>1

完成后，我们访问URL地址：

http://127.0.0.1/tp5/public/user/add

最后的输出结果为：

用户[ 流年:12 ]新增成功

为了便于看到效果，我们修改控制器的read操作方法输出更多的属性：

// 读取用户数据

publicfunctionread($id='')

{

$user = UserModel::get($id);

echo $user->nickname . '<br/>';

echo $user->email . '<br/>';

echo $user->birthday . '<br/>';

echo $user->status . '<br/>';

echo $user->create\_time . '<br/>';

echo $user->update\_time . '<br/>';

}

然后，访问URL地址：

http://127.0.0.1/tp5/public/user/12

最后的输出结果为：

流年

thinkphp@qq.com

1977/03/05

1

2016-05-0216:21:33

2016-05-0216:21:33

可以看到status、create\_time和update\_time都实现了自动写入。

如果你的status属性的值不是固定的，而是需要条件判断，那么我们可以定义修改器来配合自动完成。

<?php

namespace app\index\model;

use think\Model;

class User extends Model

{

// 定义类型转换

protected$type = [

'birthday' =>'datetime:Y/m/d',

'create\_time' =>'datetime',

'update\_time' =>'datetime',

];

// 定义自动完成的属性

protected $insert = ['status'];

// status属性修改器

protected function setStatusAttr($value, $data)

{

return'流年' == $data['nickname'] ? 1 : 2;

}

// status属性读取器

protected function getStatusAttr($value)

{

$status = [-1 =>'删除', 0 =>'禁用', 1 =>'正常', 2 =>'待审核'];

return$status[$value];

}

}

我们访问下面的URL地址进行批量新增

http://127.0.0.1/tp5/public/user/add\_list

之后，访问

http://127.0.0.1/tp5/public/user/18

最后的输出结果为：

张三

zhanghsan@qq.com

1988/01/15

待审核

2016-05-0216:40:57

2016-05-0216:40:57

# （5）查询范围

查询范围

对于一些常用的查询条件，我们可以封装成查询范围来进行方便的调用。

例如，邮箱地址为thinkphp@qq.com和status为1这两个常用查询条件，可以定义为模型类的两个查询范围方法：

<?php

namespace app\index\model;

use think\Model;

class User extends Model

{

// 定义类型转换

protected $type = [

'birthday' =>'datetime:Y/m/d',

'create\_time' =>'datetime',

'update\_time' =>'datetime',

];

// 定义自动完成的属性

protected $auto = ['update\_time'];

protected $insert = ['create\_time', 'status'];

// status读取器

protected function setStatusAttr($value, $data)

{

return'流年' == $data['nickname'] ? 1 : 2;

}

// status修改器

protected function getStatusAttr($value)

{

$status = [-1 =>'删除', 0 =>'禁用', 1 =>'正常', 2 =>'待审核'];

return $status[$value];

}

// email查询

public function scopeEmail($query)

{

$query->where('email', 'thinkphp@qq.com');

}

// status查询

public function scopeStatus($query)

{

$query->where('status', 1);

}

}

查询范围方法的定义规范为：

scope + 查询范围名称

我们修改控制器的index方法如下：

// 根据查询范围获取用户数据列表

public function index()

{

$list = UserModel::scope('email,status')->all();

foreach ($list as $user) {

echo $user->nickname . '<br/>';

echo $user->email . '<br/>';

echo $user->birthday . '<br/>';

echo $user->status . '<br/>';

echo'-------------------------------------<br/>';

}

}

最后查询的SQL语句是：

SELECT \* FROM`think\_user`WHERE`email` = 'thinkphp@qq.com'AND`status` = 1

支持多次调用scope方法，并且可以追加新的查询及链式操作，例如：

// 根据查询范围获取用户数据列表

public function index()

{

//

$list = UserModel::scope('email')

->scope('status')

->scope(function($query){

$query->order('id', 'desc');

})

->all();

foreach ($listas$user) {

echo $user->nickname . '<br/>';

echo $user->email . '<br/>';

echo $user->birthday . '<br/>';

echo $user->status . '<br/>';

echo'-------------------------------------<br/>';

}

}

上面的scope方法使用了闭包，闭包里面支持所有的链式操作方法。

最后生成的SQL语句是：

SELECT \* FROM` think\_user` WHERE `email` = 'thinkphp@qq.com 'AND` status` = 1 OR DERBY `id` desc

（6）输入和验证

现在我们来进一步使用表单提交数据完成模型的对象操作，主要内容包含：

* + [表单提交](http://www.kancloud.cn/thinkphp/thinkphp5_quickstart/147289#u8868u5355u63D0u4EA4)
  + [表单验证](http://www.kancloud.cn/thinkphp/thinkphp5_quickstart/147289#u8868u5355u9A8Cu8BC1)
  + [错误提示](http://www.kancloud.cn/thinkphp/thinkphp5_quickstart/147289#u9519u8BEFu63D0u793A)
  + [自定义验证规则](http://www.kancloud.cn/thinkphp/thinkphp5_quickstart/147289#u81EAu5B9Au4E49u9A8Cu8BC1u89C4u5219)

表单提交

首先创建一个视图模板文件 application/index/user/create.html，内容如下：

<!doctype html>

<html>

<head>

<meta charset="UTF-8">

<title>创建用户</title>

<style>

body {

font-family:"Microsoft Yahei","Helvetica Neue",Helvetica,Arial,sans-serif;

font-size:16px;

padding:5px;

}

.form{

padding:15px;

font-size:16px;

}

.form.text {

padding:3px;

margin:2px10px;

width:240px;

height:24px;

line-height:28px;

border:1px solid #D4D4D4;

}

.form.btn{

margin:6px;

padding:6px;

width:120px;

font-size:16px;

border:1px solid #D4D4D4;

cursor: pointer;

background:#eee;

}

a{

color:#868686;

cursor: pointer;

}

a:hover{

text-decoration: underline;

}

h2{

color:#4288ce;

font-weight:400;

padding:6px0;

margin:6px00;

font-size:28px;

border-bottom:1px solid #eee;

}

div{

margin:8px;

}

.info{

padding:12px0;

border-bottom:1px solid #eee;

}

.copyright{

margin-top:24px;

padding:12px0;

border-top:1px solid #eee;

}

</style>

</head>

<body>

<h2>创建用户</h2>

<FORM method="post" class="form" action="{:url('index/user/add')}">

昵称：<INPUT type="text" class="text" name="nickname"><p/>

邮箱：<INPUT type="text" class="text" name="email"><p/>

生日：<INPUT type="text" class="text" name="birthday"><p/>

<INPUT type="submit" class="btn" value=" 提交 ">

</FORM>

<div class="copyright">

<a title="官方网站" href="http://www.thinkphp.cn">ThinkPHP</a>

<span>V5</span>

<span>{ 十年磨一剑-为API开发设计的高性能框架 }</span>

</div>

</body>

</html>

User控制器增加新的操作方法create如下：

// 创建用户数据页面

public function create()

{

return view();

}

view方法是系统封装的助手函数用于快速渲染模板文件，这里没有传入模板文件，则按照系统默认的解析规则会自动渲染当前操作方法对应的模板文件，也就是默认视图目录（application/index/view）下面的user/create.html文件，所以如果改成下面的方式是相同的：

// 创建用户数据页面

public function create()

{

return view('user/create');

}

并且修改之前的add方法如下：

// 新增用户数据

public function add()

{

$user = new UserModel;

if ($user->save(input('post.'))) {

return'用户[ ' . $user->nickname . ':' . $user->id . ' ]新增成功';

} else {

return$user->getError();

}

}

我们访问URL地址：

http://127.0.0.1/tp5/public/user/create

# （8）模型输出

如果需要输出当前的模型对象有几种方式。

## 输出数组

可以使用toArray方法把当前的模型对象输出为数组。  
修改User控制器的read操作方法如下：

// 读取用户数据并输出数组

public function read($id = '')

{

$user = UserModel::get($id);

dump($user->toArray());

}

访问 http://127.0.0.1/tp5/public/user/20 页面输出结果为：

array (size=7)

'id' => int 20

'nickname' => string '张三' (length=6)

'email' => string 'zhanghsan@qq.com' (length=16)

'birthday' => string '1988/01/15' (length=10)

'status' => string '待审核' (length=9)

'create\_time' => string '2016-05-02 16:40:57' (length=19)

'update\_time' => string '2016-05-02 16:40:57' (length=19)

可以看到，User模型的数据已经经过了读取器方法处理。

如果输出的时候需要隐藏某些属性，可以使用：

// 读取用户数据并输出数组

public function read($id = '')

{

$user = UserModel::get($id);

dump($user->hidden(['create\_time','update\_time'])->toArray());

}

再次访问 http://127.0.0.1/tp5/public/user/20 页面输出结果变成：

array (size=5)

'id' => int 20

'nickname' => string '张三' (length=6)

'email' => string 'zhanghsan@qq.com' (length=16)

'birthday' => string '1988/01/15' (length=10)

'status' => string '待审核' (length=9)

## 输出JSON

对于API开发而言，经常需要返回JSON格式的数据，修改read操作方法改成JSON输出：

// 读取用户数据输出JSON

public function read($id = '')

{

$user = UserModel::get($id);

echo $user->toJson();

}

访问 http://127.0.0.1/tp5/public/user/20 页面输出结果为：

{"id":22,"nickname":"张三","email":"zhanghsan@qq.com","birthday":"1988\/01\/15","status":"待审核","create\_time":"2016-05-02 16:40:57","update\_time":"2016-05-02 16:40:57"}

或者采用更简单的方法输出JSON，下面的方式是等效的：

// 读取用户数据直接输出JSON

public function read($id = '')

{

echo UserModel::get($id);

}

模型的toJson方法默认使用了JSON\_UNESCAPED\_UNICODE编码参数输出，支持传入JSON编码参数，例如：

// 读取用户数据输出JSON

public function read($id = '')

{

$user = UserModel::get($id);

echo $user->toJson(JSON\_PRETTY\_PRINT | JSON\_UNESCAPED\_UNICODE);

}

输出结果为：

{ "id": 22, "nickname": "张三", "email": "zhanghsan@qq.com", "birthday": "1988\/01\/15", "status": "待审核", "create\_time": "2016-05-02 16:40:57", "update\_time": "2016-05-02 16:40:57"}

**六、快速入门（六）：视图和模板**

本篇主要来学习视图和模板的用法。

前面只是在控制器方法里面直接输出而没有使用视图模板功能，从现在开始来了解下如何把变量赋值到模板，并渲染输出，主要内容包括：

* + [模板输出](http://www.kancloud.cn/thinkphp/thinkphp5_quickstart/147290#u6A21u677Fu8F93u51FA)
  + [分页输出](http://www.kancloud.cn/thinkphp/thinkphp5_quickstart/147290#u5206u9875u8F93u51FA)
  + [公共模板](http://www.kancloud.cn/thinkphp/thinkphp5_quickstart/147290#u516Cu5171u6A21u677F)
  + [模板定位](http://www.kancloud.cn/thinkphp/thinkphp5_quickstart/147290#u6A21u677Fu5B9Au4F4D)
  + [模板布局](http://www.kancloud.cn/thinkphp/thinkphp5_quickstart/147290#u6A21u677Fu5E03u5C40)
  + [标签定制](http://www.kancloud.cn/thinkphp/thinkphp5_quickstart/147290#u6807u7B7Eu5B9Au5236)
  + [输出替换](http://www.kancloud.cn/thinkphp/thinkphp5_quickstart/147290#u8F93u51FAu66FFu6362)
  + [渲染内容](http://www.kancloud.cn/thinkphp/thinkphp5_quickstart/147290#u6E32u67D3u5185u5BB9)

模板输出

首先来看如何输出一个数据集，我们修改User控制器的index方法如下：

<?php

namespace app\index\controller;

use app\index\model\UserasUserModel;

use think\Controller;

class User extends Controller

{

// 获取用户数据列表并输出

public function index()

{

$list = UserModel::all();

$this->assign('list', $list);

$this->assign('count', count($list));

return $this->fetch();

}

}

这里的User控制器和之前模型章节有所不同，继承了系统的\think\Controller类，该类对视图类的方法进行了封装，所以可以在无需实例化视图类的情况下，直接调用视图类的相关方法，包括：

| **方法** | **描述** |
| --- | --- |
| assign | 模板变量赋值 |
| fetch | 渲染模板文件 |
| display | 渲染内容 |
| engine | 初始化模板引擎 |

这里用到的其中两个方法assign和fetch，也是最常用的两个方法。

assign方法可以把任何类型的变量赋值给模板，关键在于模板中如何输出，不同的变量类型需要采用不同的标签输出。

前面我们已经学习过，fetch方法默认渲染输出的模板文件应该是当前控制器和操作对应的模板，也就是：

application/index/view/user/index.html

接下来，定义视图文件的内容，采用volist标签输出数据集：

<!doctype html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>查看用户列表</title>

<style>

body{

color:#333;

font:16px Verdana, "Helvetica Neue", helvetica, Arial, 'Microsoft YaHei', sans-serif;

margin:0px;

padding:20px;

}

a{

color:#868686;

cursor: pointer;

}

a:hover{

text-decoration: underline;

}

h2{

color:#4288ce;

font-weight:400;

padding:6px0;

margin:6px00;

font-size:28px;

border-bottom:1px solid #eee;

}

div{

margin:8px;

}

.info{

padding:12px0;

border-bottom:1px solid #eee;

}

.copyright{

margin-top:24px;

padding:12px0;

border-top:1px solid #eee;

}

</style>

</head>

<body>

<h2>用户列表（{$count}）</h2>

{volist name="list" id="user" }

<div class="info">

ID：{$user.id}<br/>

昵称：{$user.nickname}<br/>

邮箱：{$user.email}<br/>

生日：{$user.birthday}<br/>

</div>

{/volist}

<div class="copyright">

<a title="官方网站" href="http://www.thinkphp.cn">ThinkPHP</a>

<span>V5</span>

<span>{ 十年磨一剑-为API开发设计的高性能框架 }</span>

</div>

</body>

</html>

ThinkPHP5.0默认使用的是一个内置的编译型模板引擎，包含了一系列的模板标签，我们会陆续介绍一些常用的标签用法。

index方法给模板赋值了两个变量count和list，分别是标量和二维数组，标量的输出很简单，使用： {$count}便可，一看就明白。

二维数组通常使用volist标签输出。

{volist name="list" id="user"}

ID：{$user.id}<br/>

昵称：{$user.nickname}<br/>

邮箱：{$user.email}<br/>

生日：{$user.birthday}<br/>

------------------------<br/>

{/volist}

volist标签的name属性就是模板变量的名称，id属性则是定义每次循环输出的变量，在volist标签中间使用{$user.id}表示输出当前用户的id属性，以此类推下面的内容则依次输出用户的相关属性。

ID：{$user.id}<br/>

昵称：{$user.nickname}<br/>

邮箱：{$user.email}<br/>

生日：{$user.birthday}<br/>

来看下实际的输出效果，访问URL地址：

http://127.0.0.1/tp5/public/user/index

就可以看到页面输出结果如图：



当然，实际看到的数据可能有所出入。

分页输出

可以很简单的输出用户的分页数据，控制器index方法修改为：

// 获取用户数据列表

public function index()

{

// 分页输出列表每页显示3条数据

$list = UserModel::paginate(3);

$this->assign('list',$list);

return $this->fetch();

}

模板文件修改为：

<link rel="stylesheet" href="/static/bootstrap/css/bootstrap.min.css" />

<h2>用户列表（{$list->total()}）</h2>

{volist name="list" id="user"}

ID：{$user.id}<br/>

昵称：{$user.nickname}<br/>

邮箱：{$user.email}<br/>

生日：{$user.birthday}<br/>

------------------------<br/>

{/volist}

{$list->render()}

注意，由于演示需要，在模板中引入了bootstrap样式文件。请按照所示位置存放。

如果我们还不太清楚bootstrap的样式文件具体应该放在哪个位置，则应应该打开firefox或firebug的控制台中网络部分，查看报错的红色字体部分，它会告诉你，具体的路径应该在哪。即使这样，前期我们也不见得能很少的掌握，如果看了网络中的报错还没有解决的话，请咨询其他队员或自己的组长。另外，自行google绝对路径与相对路径的区别.

多插入一些数据后，访问下面的地址：

http://127.0.0.1/tp5/public/user/index

后可以看到输出如图所示：  


模板定位

fetch方法的第一个参数表示渲染的模板文件或者模板表达式。

通常我们都是使用的模板表达式，而不需要使用完整的文件名。

模板文件名可以随意命名，如果把index.html文件改成：

application/index/view/user/list.html

index操作方法中的fetch方法需要改成：

return $this->fetch('list');

而如果fetch方法改成：

return $this->fetch('index/list');

那么实际渲染的模板文件则是

application/index/view/index/list.html

当然，你可以设置更多的目录级别，例如：

return $this->fetch('one/two/three/list');

那么实际渲染的模板文件则是

application/index/view/one/two/three/list.html

有一些和模板定位相关的设置参数能够帮助你调整模板文件的位置和名称。  
通常来说，模板相关的参数可以直接在配置文件中配置template参数，默认的配置如下：

'template' => [

// 模板引擎类型支持 php think 支持扩展

'type' =>'Think',

// 模板路径

'view\_path' =>'',

// 模板后缀

'view\_suffix' =>'.html',

// 模板文件名分隔符

'view\_depr' => DS,

// 模板引擎普通标签开始标记

'tpl\_begin' =>'{',

// 模板引擎普通标签结束标记

'tpl\_end' =>'}',

// 标签库标签开始标记

'taglib\_begin' =>'{',

// 标签库标签结束标记

'taglib\_end' =>'}',

],

view\_path参数决定了你的模板文件的根目录，如果没有设置的话系统会默认使用当前模块的视图目录view。

如果希望自定义模板文件的位置、命名和后缀，可以对模板参数稍加修改如下：

'template' => [

// 模板引擎类型支持 php think 支持扩展

'type' =>'Think',

// 模板路径

'view\_path' =>'../template/index/',

// 模板后缀

'view\_suffix' =>'.tpl',

// 模板文件名分隔符

'view\_depr' =>'\_',

],

通过配置我们把当前渲染的模板文件移动到了

ROOT\_PATH/template/index/user\_index.tpl

输出替换

为了更加清晰，需要把资源文件独立出来，并在模板文件中引入，例如增加public/static/common.css文件：

body{

color:#333;

font:16px Verdana, "Helvetica Neue", helvetica, Arial, 'Microsoft YaHei', sans-serif;

margin:0px;

padding:20px;

}

a{

color:#868686;

cursor: pointer;

}

a:hover{

text-decoration: underline;

}

h2{

color:#4288ce;

font-weight:400;

padding:6px0;

margin:6px00;

font-size:28px;

border-bottom:1px solid #eee;

}

div{

margin:8px;

}

.info{

padding:12px0;

border-bottom:1px solid #eee;

}

.copyright{

margin-top:24px;

padding:12px0;

border-top:1px solid #eee;

}

我们在header.html文件中引入资源文件

<html>

<head>

<meta charset="UTF-8">

<title>[title]</title>

<link charset="utf-8" rel="stylesheet" href="/static/common.css">

</head>

<body>

但这样有一个问题，如果部署的目录变化的话，资源文件的路径就会跟着变化，这里我们采用输出替换功能，使得资源文件的引入动态化  
可以在输出之前对解析后的内容进行替换，使用：

// 读取用户数据

public function read($id='')

{

$user = UserModel::get($id);

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

$this->view->replace([

'\_\_PUBLIC\_\_' =>'/static',

]);

return$this->fetch();

}

模板文件改为：

<html>

<head>

<meta charset="UTF-8">

<title>[title]</title>

<link charset="utf-8" rel="stylesheet" href="\_\_PUBLIC\_\_/common.css">

</head>

<body>

最终输出的时候，会自动进行\_\_PUBLIC\_\_替换。

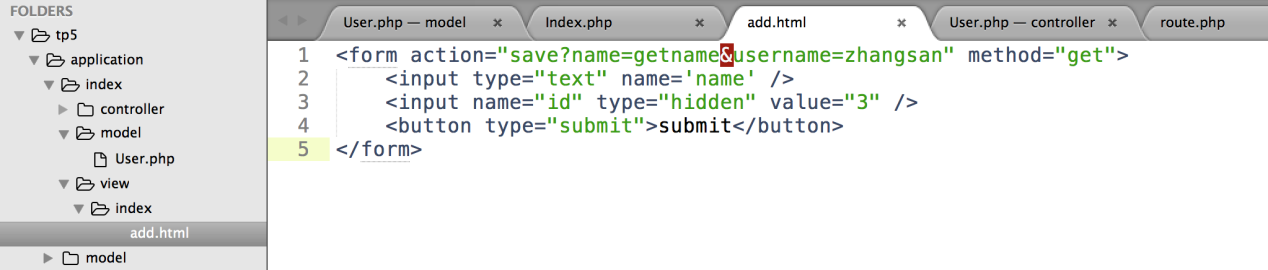
当然了，这些内容，我们往往会放在项目的根控制器中来完成，在这里了解一样，当看到的\_\_PUBLICE\_\_出在模板中时，能想到我们在这里学习过，然后能找到相关的内容，即是我们此次前期学习的目标。

**七、快速入门（七）：请求与响应**

输入参数

如果仅仅是需要获取输入参数，可以使用系统提供的think\Input类和input助手函数，例如：





示例效果:



使用助手函数的代码为：



如果要进行页面重定向跳转，可以使用：

public function hello()

{

return redirect('index/error',301);

}

redirect方法默认使用302跳转。