**第八章 MVC模式下的ThinkPHP 5.x开发框架的使用**

古人云：“路遥知马力，日久见人心”。很多时候在讨论某一个框架好与坏的时候，只有真正的使用过，并深入了解过后才有发言权，如同讨论某某手机比某某手机好是一样的，只有两者都用过之后才能最终下结论。而框架的使用也是如此，一个框架如果能经过5-10年的发展，还依然被许多人所接受，那么它就一定有它的优势，哪怕是文档写的好，这也是优势，同时在这么多年的发展过程中框架本身也会不断吸收其他优秀框架的有点，让自己变的更加优秀。因此找一个合适自己的框架，并能够用该框架完成一些项目才是最关键，切不可一心追新、追变，就像PHP一样，诞生于上个世纪，但现在依然活跃在编程领域，甚至需求量还在逐年攀升。本章将着重讨论ThinkPHP的使用方法，以及相关工程应用，在这个过程中除了了解如何使用ThinkPHP去开发一个项目，还需要了解如果通过工程的思想来进行改造和升级。

**本章内容**

1、MVC模式在Web开发中的应用

2、ThinkPHP框架的概述与项目部署

3、HTTP请求与ThinkPHP控制器的应用

4、ThinkPHP中路由的设置与URL的生成

5、ORM模型的创建与数据库的CURD操作

6、ThinkPHP内置模板引擎与视图的使用

8.1 MVC模式在Web开发中的应用

在Web开发中很多时候对于初学者或者不注重软件工程的开发人员来说，开发只是一个功能的实现，因此经常出现软件的前台和后台的代码混淆，造成后后期难以维护的结果，而那稍有些的开发经验的人则会想办法把各个模块进行独立，但是这样的操作通常不能适用于所有项目，因为要使现有程序从结构上进行分离，同时有需要满足可复用的要求是十分困难的，不过好在已经有程序设计的大神们帮助我们去实现了这一想法，并提供了一条可行的道路，即MVC模式。该模式的有点在于，它不和任何的语言进行绑定，而是一个经过无数软件验证的思想，是多年软件开发经验的高度总结。虽然采用这种方式会增加不少代码，但是不妨碍它成为使用最为广泛的软件架构模式。

8.1.1 MVC的工作模式和原理

所谓MVC，并不是一个单词，而是选取了三个英文单词的首字母，即Model（模型）、View（视图）、Controller（控制器），其中Model（模型）的作用表示建立的业务模型，以及响应各种查询业务，View（视图）则主要负责业务处理和查询结果的展示，以及传递用户请求，而Controller（控制器）则主要负责接收从View视图传递过来的用户请求，经过处理后传递给Model模型能够处理的数据，同时选择相应的视图View来进行展示，如图8-1所示。

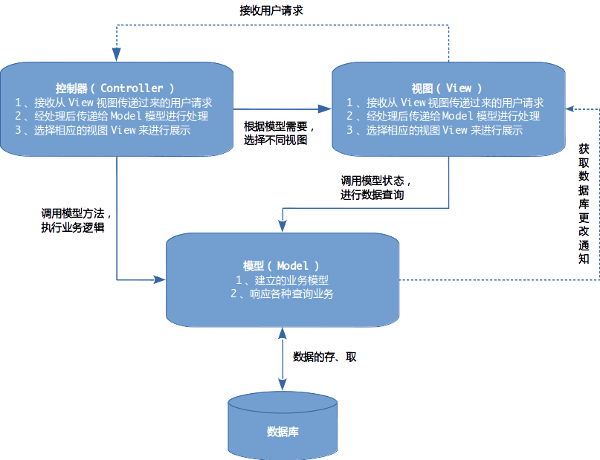


图8-1 MVC模式中各模块的关系

在这个模型中最重要的就是Model模型的设计，因为它是整个业务的核心，整个业务的流程都在模型中完成，并且对外只提供简单的数据接口和业务请求类型，因此从模型的外部向里看是看到不到任何内容，也就是屏蔽了控制器和视图的的不同，如果把模型单独拿出来，也可以自成一个体系，只要业务内容和流程不发生改变，那么无论控制器和视图如何改变都不会影响到模型。其次就是Controller控制器，其主要功能就是承上启下，所谓“承上”就是指接收用户的各类请求，并且根据请求类型的不同，分发到各个模型中，而“启下”则是指在最终送到到模型前，把数据转化为模型能够接受的数据，使得从请求到执行能否顺畅运行，因此对于控制器来说，它屏蔽了视图成各种各样的请求，把这些请求化整为零，使得无论模型如何改变，又或者视图如何改变都可以在它们之间建立一个通畅的桥梁。最后就是View视图，其主要功能是接收用户各种各样的请求，例如点击提交、查看按钮等，此外还可以以不同形式展现模型中的数据，即可以为同一模型定制不同的视图或不同模型定义相同的视图。

因此采用MVC架构的程序能够满足用户需求快速变化的环境，例如制作上一个商城系统，其功能都是订单、购买、收藏等，而展示则可能采用仿照淘宝、京东或自己设计，此时整个系统的开发的焦点就只有View视图，而不包括其他部分，从而大大降低了重复代码，提高了可维护性。此外，当模型层有新的技术时也可以在保持整个业务逻辑不发生改变的情况进行修改和更新，所以MVC模式总结下来具有如下优势：

1、多个视图对应一个模型的能力。

2、由于一个应用被分为三层，因此有时只需要改变其中的某一层就能满足需求，实现快速响应。

3、由于各层各司其职，有利于项目的工程化管理。

虽然MVC有诸多优点，但是从头开始实现MVC却不是一个容易的事，并且MVC只是一个顶层设计实现，而到达每一层后，还需要更为详细的设计，同时在MVC架构中View可以对模型进行操作，也就意味着在View中会有一部分对模型的依赖，使得View和Model之间没有完全的解耦，并且由于各层之间完全隔离，也使得需要更多的代码来实现该架构，增加了很多代码量。

8.1.2 常用的几种架构模式与PHP框架

除了上面讲到的MVC架构，实际上目前使用比较多的还有另外两种，分别是MVP和MVVM，这两种都是在MVC的基础上进行改进得来。例如在MVP架构中，直接抢断了视图View和模型Model之间关系，解决了MVC架构中View和Model没有完全解耦的问题，这是MVP和MVC架构最大的不同点。同时在MVP中把原先MVC的Controller控制器改为了Presenter，即呈现层，负责整个程序的逻辑，以及在Model和View之间做到承上启下的作用。而MVVM架构则是在MVP的基础上进行再次演进，把原先MVP中的Presenter改为了ViewModel，实现View和ViewModel之间的双向绑定，使得当View改变时会自动反应在ViewModel上，而当ViewModel改变时也会自动反应到View上，例如Google的Angular和Micsoft的WPF就是这种架构。

所谓开发框架，实际上就是使用某一种语言对某一种顶层架构模式的实现，以完成某一个问题领域类的任务，同时框架对诸多细节进行抽象，并使用设计模式来实现这些这些抽象后模块之间的解耦。因此框架并不会把所有的事情做完，更多的就像一个毛坯房，只不过这个房子已经装上了地暖、中央空调、新风等高级系统，但是里面如何装修等能让房子主人满意，这个就是装修设计师要完成的内容。另外使用框架有两个好处，首先开发人员在使用的过程中可以学习和了解这些框架的优点和缺点，在今后自己的开发中也可以借鉴这些优秀的思想，从而提高程序的质量，同时使用框架可以帮助开发人员屏蔽很多繁琐、重复的内容，使得开发人员可以把精力完全集中在业务逻辑的开发中，而且使用框架可以方便进行扩展和维护，并且框架还提供了许多细节处理，例如线程安全、IO安全等特性，从而提高软件的整体质量。

目前在PHP中，通用的框架有许多，但目前国内使用比较多的主要集中在下面四个，分别是Yii、Laravel、ZendFramework、ThinkPHP，其中Yii、Laravel、ZendFramework是由国外的开发人员进行维护和编写，而ThinkPHP则是由国内的开发人员进行维护和编写，下面对这几个框架进行简单的介绍：

1、Yii：由于该框架大量使用了延迟加载技术，即不会包含一个类文件，直到该类第一次使用，并且在第一次访问某个对象之前它不会创建该对象。所以Yii是一个高性能，基于组件的PHP框架，用于快速开发现代的Web应用程序。同时Yii还是实现了MVC架构的全栈框架，并基于该架构组织代码，并且对关系型数据库和NoSQL数据库都提供了查询生成器和ActiveRecord，也支持基于RESTful API开发等等。

2、Laravel：Laravel是一套简洁、优雅的PHP Web开发框架。它可以让开发人员从面条一样杂乱的代码中解脱出来，从而构建一个完善的网络APP，而且每行代码都可以简洁、富于表达力。在Laravel中已经具有了一套高级的PHP ActiveRecord实现。它能方便的将”约束”应用到关系的双方，这样开发人员就具有了对数据的完全控制，而且使用到ActiveRecord的所有便利。

3、ZendFramework：Zend Framework是PHP的母公司Zend公司开发的一套基于MVC架构的PHP开发框架技术，它提供了一个优秀的、简单的综合开发环境，提供了很多可用的解决方案，可以用来建立一个稳定的、可升级的的Web应用。同时Zend Framework还包含了PHP中所有的新技术，但是随着Zend Framework的不断发展，目前的Zend Framework已经非常庞大，而且复杂，所以不适合刚接触框架的人来学习。

4、ThinkPHP：ThinkPHP是国人开发的一个快速、简单，并且面向对象的轻量级的PHP框架，它也是免费、开源，可以用于任何商业软件开发的框架。同时，ThinkPHP还遵循PSR-2、PSR-4规范，并支持Composer和单元测试，框架内异常严谨的错误检测和安全机制，也使得开发的程序更加稳定，而RESTful的支持使得可以提供更好的API开发。这些特性的都保证ThinkPHP能够开发现代Web的所有功能，所以该框架的使用也是本书要讲的主要内容。

8.2 ThinkPHP框架的概述与项目部署

ThinkPHP自2006年诞生以来经过了10多年的技术沉淀，目前已经国内开发PHP相关项目最主要的框架之一。ThinkPHP的设计理念源自Struts，并在其基础上做了大量的改进，同时还引入了许多国外优秀的框架思想和模式。目前ThinkPHP主要使用的版本有两个，分别是3.x和5.x，其中3.x是ThinkPHP获得成功的主要版本，并且得到了大量项目的使用和验证其可靠性，但是目前3.x已经完成了历史使命。在2017年的2月ThinkPHP发布了中5.x版本，该版本是在3.x的基础上进行全面的改写，具有更加规范、更加严谨、扩展更加灵活、全面支持Restful接口等特性，因此5.x版本和3.x版本并不兼容，建议在新项目上使用5.x进行开发，本节内容主要以最新的5.x为基础进行相关内容的讲解。

8.2.1 ThinkPHP的安装与目录结构

ThinkPHP是在PHP基础上的一个框架，因此它依赖于PHP作为其运行，目前ThinkPHP 5.x的运行环境需要PHP版本在5.4以上，并且至少安装了PDO、MBString、CURL这三个PHP插件，其中PDO用于实现数据库的存取、MBString用于实现字符编码的转换，而CURL则实现各种HTTP通信。在确认完成运行环境之后就可以进行ThinkPHP的安装，目前ThinkPHP提供三种方式进行安装，分别如下：

1、第一种方法是进入ThinkPHP的官网（http://www.thinkphp.cn/），点击并下载右边的“ThinkPHP5.x完整版”，然后解压缩到Web服务器的目录下。

2、第二种方法是使用Composer工具进行下载，该工具是一个基于项目的PHP依赖管理工具，通过该工具可以为PHP项目添加相关依赖程序，例如PHP单元测试、symfony模板库等，类似与Android的Gradle。要使用Composer非常简单，对于Linux或者MAC用户来说只需要执行下面的指令即可完成安装，而Windows用户需下载Composer-Setup（https://getcomposer.org/Composer-Setup.exe）即可。

# 下载Composer

curl -sS https://getcomposer.org/installer | php

# 把下载的composer.phar修改为系统指令

mv composer.phar /usr/local/bin/composer

# 由于国内网络环境的原因，需要把composer的镜像源切换到国内镜像

composer config -g repo.packagist composer https://packagist.phpcomposer.com

在安装完Composer并切换到国内镜像后就可以通过Composer指令进行ThinkPHP的安装，在该指令中“--prefer-dist”选项的作用是告诉Composer直接下载不带Git信息的压缩包，具体指令如下：

composer create-project topthink/think 创建项目的名称 --prefer-dist

3、第三种则是使用Git进行安装，目前ThinkPHP被分为两个相对独立的模块，一个是ThinkPHP的应用模块，另一个是ThinkPHP的核心模块，因此要完整的使用ThinkPHP需要分别下载这两个模块，然后把它们组合起来使用。首先创建一个项目文件夹，然后通过Git指令克隆（clone）ThinkPHP的应用模块到该文件夹，然后在项目文件夹下创建一个文件夹叫thinkphp，最后进入thinkphp文件夹把ThinkPHP的核心模块的核心模块克隆（clone）到该文件夹即可，具体指令如下：

# 创建一个项目文件夹，并进入该目录

mkdir 项目文件夹;cd 项目文件夹

# 克隆（clone）ThinkPHP的应用模块到项目文件夹

git clone https://github.com/top-think/think

# 创建ThinkPHP模型模块文件夹thinkphp

mkdir thinkphp

# 克隆（clone）ThinkPHP的核心模块到thinkphp文件夹

git clone https://github.com/top-think/framework thinkphp

采用这三种方式所创建的项目都会有所区别，其中采用Git方式创建的项目是紧跟ThinkPHP官方的脚步，包括了所有最新的改动，但是这些改动可能不是非常稳定，而采用Composer方式创建的项目则是官方最新发布的正式版，即可以放心使用，最后就是去官网下载的方式，这种方式的有点是包含了很多第三方的库，但是不一定是最新版，但一定是一个稳定版。因此综合上面的说法，笔者在这里推荐采用Composer的方式进行安装，有以下三点原因：

1、目前不论国内还是国外，一个成熟的PHP项目都会采用Composer的方式来进行依赖管理，因此试着理解和学习Composer的使用对于今后和现在的工作都有很大好处。

2、使用官网下载的内容最大的优势在与包含了很多第三方的库，例如workerman、symfony等，但是实际上这些库实际上都可以通过Composer获取，并且获取的是最新稳定版。

3、通过Composer创建项目后，可以根据自己的需要添加和删除第三方库，从而动态整个项目的大小。

在本书中，所有的ThinkPHP的安装，以及依赖管理都采用Composer方式，因此在Web服务器的目录下执行下面的指令来创建一个名叫“tp5”的ThinkPHP项目，具体指令如下：

composer create-project topthink/think tp5 --prefer-dist

当执行完该指令后，就会在服务器目录下创建一个tp5的文件，该文件夹中包含了ThinkPHP的所有内容，最后在浏览器中输入“http://localhost/tp5/public/”就可以看到如图8-2所示的ThinkPHP欢迎界面，此时就表示ThinkPHP安装并初始化完成，接下来就可以开始编程。

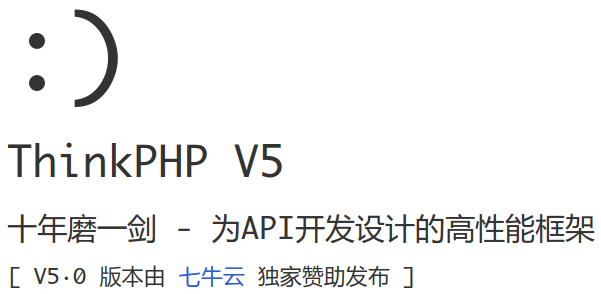


图8-2 ThinkPHP初始化

要正确使用ThinkPHP进行项目开发，首先就需要了解ThinkPHP的目录结构，只有了解了其结构才能知道哪些地方是存放PHP代码，哪些地方是存放HTML代码等。首先进入到TP（下对ThinkPHP的简称）的目录，会看到有12个文件和文件夹，分别为application、extend、public、runtime、thinkphp、vendor、build.php、composer.json、composer.lock、LICENSE.txt、README.md、think，这12个文件和文件夹的功能分别如表8-1所示。

表8-1 TP根目录下的文件与文件夹说明

|  |  |
| --- | --- |
| 文件或文件夹名称 | 说明 |
| application | 应用程序的目录，随后编写的程序都在该文件夹下 |
| extend | 扩展类库目录 |
| public | 供外部访问的目录，用于存放图片、JS库、入口文件等公共内容 |
| runtime | 应用的运行时产生的临时文件 |
| thinkphp | ThinkPHP的核心模块 |
| vendor | 存放使用Composer下载的第三方类库 |
| build.php | 自动生成定义文件 |
| composer.json | Composer依赖库的定义文件 |
| composer.lock | 保存使用Composer安装依赖库时的的版本号 |
| LICENSE.txt | 开源软件的证书文件 |
| README.md | 项目说明文件 |
| think | 命令行的入口文件 |

在这些文件和文件夹中，有一个文件夹需要特别说明，即application，该文件夹是项目构建的核心文件夹，是应用程序主要编写和存放的地方，进入到该文件夹下有6个文件和1个文件夹，分别为index、command.php、common.php、config.php、database.php、route.php、tags.php，这7个文件和文件夹的分别如表8-2所示。

表8-2 application目录下的文件与文件夹说明

|  |  |
| --- | --- |
| 文件或文件夹名称 | 说明 |
| index | 应用程序的模块目录 |
| command.php | 命令行工具配置文件 |
| common.php | 应用程序公共函数文件 |
| config.php | 应用程序公共配置文件 |
| database.php | 数据库配置文件 |
| route.php | 路由配置文件 |
| tags.php | 应用行为扩展定义文件 |

这些文件和文件夹的意义从上表中可以清楚的知道，但是index却不是特别清楚。在编写一个完整的应用程序时，通常会包含前台、后台、移动端等各种各样独立的系统，这些系统在TP体系中就称为模块。在TP中模块的概念是一个独立的，并且具有完整功能的系统，例如前台模块中可以有自己的UI、可以有自己的数据库操作、可以有自己的HTTP通信等，而后台也可以包含这些功能，两者之间相互独立，当然后期也可以通过一些设置来实现模块和模块之间某些部分的公用，总之每一个模块下都会按照MVC架构进行文件夹和文件的排列，例如现在进入index目录就可以看到一个controller文件夹，该文件夹里面的内容就是控制器，而其他View和Model在后面的讲解中会不断创建。因此可以根据项目的需求在application目录下创建多个与index目录平行的模块文件夹来进行区分。

8.2.2 ThinkPHP的体系结构与生命周期

为了更加清楚的了解TP的体系结构，有一些概念需要提前说明，因为这些概念都会贯穿于本章的所有内容，同时在今后的项目开发中也起到非常重要的作用，并且在了解这些概念后，也为之后学习其他框架建立一个良好的基础。在TP中有十个概念非常重要，具体如下：

**1、入口文件：** 该文件是所有请求的入口，其作用有两个，首先该文件指定了TP核心框架的路径，使得在所有请求中都能载入TP的框架代码，其次该文件还制定的应用的路径，最后在TP中因为只有载入入口文件的请求才有可能成功，所以防止具有敏感信息的页面被直接访问。

**2、应用：**在TP中应用是一个完整的待开发的系统，并且也是TP生命周期的执行对象。通常来说，一个应用就是一个TP的项目，但一个项目却可以有多个模块，上节在讲目录结构时提到的application其实就是这里的应用。

**3、模块：**这个概念在上节讲过，即一个应用的通常会有单独的模块，如前台、后台等，并且这些模块具有各自独立的配置文件、公共文件等。

**4、控制器：**每个模块拥有独立的MVC模型，并且每个模块下面都会有多个控制器负责响应请求，每个控制器就是一个独立的控制器类。控制器主要负责接收各类请求，并调用相关的业务模型进行处理，并选择相应的视图进行输出，但是这里有一点要说明的是，从MVC模型的角度来说控制器应该只负责业务流程的分发，而不负责业务的逻辑处理。

**5、操作（方法）：**所谓方法就是控制器内的函数，一个控制器内可以有多个操作，并且这些操作是真正接收请求和执行请求的对象。

**6、模型：**模型是整个TP的核心模块，该模块主要用于数据操作和业务逻辑，并且返回和视图无关的数据。很多在刚入门TP时会把模型只作为对数据库操作的对象，但是实际上这只是TP的一部分功能，更多的应该是对模型进行更加细致的分层设计，从而实现整个业务的逻辑层、数据层等。

**7、视图：**控制器根据请求的不同调用不同的视图模板，这些模板由HTML、CSS、JavaScript组成，最后根据视图数据的类型把从模型中获取的无格式数据进行转化，填入模板中，从而呈现在Web端。

**8、驱动：**所谓驱动可以理解某一个组件的扩展，例如数据库驱动、缓存驱动等，但是这些驱动都提供了统一的接口，方便TP进行读写操作和驱动扩展。

**9、行为：**把一类被频繁使用的操作进行抽象和隔离，然后让多个不同的流程共用这一个流程，通过行为的使用可以让TP根据更加通用的灵活性。

**10、命名空间：**简单说就是把各种类进行分类，当这些分类中有冲突时，加上命名空间，以区分各自的不同。

这些概念中，有些非常好理解，比如入口文件、控制器等，而有些就不是特别好理解，如驱动、行为等，不需要担心，这里只是做一个简单介绍，在后续的内容中会具体讲解。在了解了TP中的概念之后，接下来就来了解TP程序运行的生命周期，对于生命周期的理解能够帮助读者更加清楚的了解整个程序运行流程，并且在未来修改程序时也能够知道在哪里修改最为合适。一个完整的TP生命周期包含了十个步骤，具体如下：

**第一步：入口文件，**用户发起一个请求，首先经过的就是入口文件。TP的入口文件位于public目录下的index.php文件。该文件中首先定义了一个指定应用程序入口位置的常量“APP\_PATH“，在默认情况下应用程序的入口位置在application目录下，其中“\\_\\_DIR\\_\\_”是PHP的一个常量表示当前文件的路径，因此"\\_\\_DIR\\_\\_ . '/../application/'"就表示当前文件的上一级目录下的application文件夹，即应用程序的根目录。其次该文件执行并导入了框架的引导文件start.php，该文件位于与application同级的thinkphp目录下，具体代码如下：

// 定义应用目录

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

// 加载框架引导文件

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

**第二步：执行引导文件，**在引导文件里，首先加载了项目初始化文件base.php，在该文件中首先完成了TP系统常量的定义，其次加载环境变量定义文件，然后注册了系统自动加载机制和注册系统错误和异常的处理机制，最后加载系统配置文件。当这些都完成后就执行应用程序，具体代码如下：

// 加载基础文件

require \_\_DIR\_\_ . '/base.php';

// 执行应用

App::run()->send();

**第三步：注册自动加载程序，**当引导程序完成环境变量的定义之后，就会注册自动加载的程序，这些程序首先会注册自动加载程序autoload，然后就加载包括使用Composer安装的相关依赖，命名空间，以及extend目录下的程序等，具体代码如下。有兴趣的读者可以查看该自动加载的源代码，位于"thinkphp\library\think\Loader.php"。

// 注册自动加载

\think\Loader::register();

**第四步：注册系统错误和异常处理程序，**注册完自动加载后就是注册系统错误和异常处理，这里一共注册了三种处理函数，分别是应用关闭函数appShutdown、错误处理函数appError以及异常处理函数appException。当整个程序发生异常时程序的执行会停止，那么此时TP就会捕捉到这些异常，并显示这些错误。

// 注册错误和异常处理机制

\think\Error::register();

**第五步：应用初始化，**在该阶段主要是加载应用程序的各类配置文件，以及公共函数，并设置时区等。当执行完这些之后，整个TP内部已经初始化完毕，接下来就是对用户请求进行分析。

**第六步：URL正确性检测，**到这一步会检查请求的地址时候符合规范，在TP中典型的请求地址包括服务器地址、入口文件、模块、控制器、操作，以及函数等，并且不区分大小写，因此如果传递的请求中包含大写字母时，也会被系统自动转成小写字母，一个标准的URL请求如下：

http://serverName/index.php/模块/控制器/操作/[参数名/参数值...]

在8.2.1节中通过Composer创建了名字为tp5的TP后，在浏览器中输入”http://localhost/tp5/public/“就可以看到TP的欢迎界面，这是因为默认创建的控制器和方法都是以Index命名，所以浏览器就直接读取，而完整的URL请求应该如下：

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

在这个URL中public/index.php表示读取入口文件，而后面第一个index表示的是application目录下的index模块，第二个index则表示在index模块下的controller文件夹下的index控制器，该控制器的文件名为Index.php，因为在TP中URL不区分大小写，因此这里不论是写index或者Index都会被转化为index，而最后一个index则表示在index控制器index方法，打开Index.php就可以看到该方法。接下来在Index.php中添加一个函数叫helloWorld()，具体代码如下：

class Index

{

    public function index()

    {

        return 'xxx';

    }

    public function helloWorld()

    {

        return "Hello World";

    }

}

然后在浏览器中重新输入地址”http://localhost/tp5/public/index.php/index/index/helloWorld“就可以看到浏览器输出”Hello World“，表示请求成功。

**第七步：路由检测，**从第六步的例子可以初步看出，一个完整的URL是非常长的，那么路由的目的就在于简化URL，使得URL的表达更加的简洁和明确。

**第八步：分发请求，**不论时候检测到路由，此时TP都会把URL转化为真实的地址，这是整个生命周期中非常重要的一个环节，在该阶段TP会根据真实的URL寻找对应的模块、控制器和方法，并执行这些方法。

**第九步：响应输出，**在第六步中看到，不论index还是helloWord都是采用return的方式进行数据返回，而不是采用echo函数，这是因为当控制器return后，TP内部还会调用系统的Response::send方法将需要返回的数据输出到页面或者客户端，此时的关键在于采用return返回时，如果返回的不是HTML，那么就会把数据转化为在配置文件中配置的数据类型，如JSON、XML等，而采用echo输出则不会自动转换，因此对于采用return方式返回的数据不需要考虑类型转换，比echo返回简化了许多操作。

**第十步：应用结束，**实际上到了这一步，基本上已经和开发人员没有太大的关系，剩下的就是TP内部的日志操作等。

到这里TP的整个生命周期和体系结构就已经讲清楚了，从上面的内容可以看到，对于开发人员来说，除了模型的构建外，最重要的就是控制器和方法的实现，因为这个是数据请求的唯一途径，而且对于某些应用来说，如微信小程序、APP等，是不需要实现视图，至于后面的路由等功能都是在完成模型和控制器基础上的进一步优化。

8.2.3 ThinkPHP的系统配置和配置作用域

在使用TP时经常需要进行一些数据库配置、路由配置等，而TP中的配置信息采用数组的方式进行部署，并且当TP读取这些配置项时，TP会把这些配置项名称不论大小写都自动转化为小写，因此对于TP来说配置项名称不区分小写，并且TP还支持在数组里面套用多维数组的方式进行配置，从而提供更加丰富的配置内容，示例代码如下：

return [

    // 配置向信息

    '配置项名称' =>  '配置项的值',

    // 二维数组的配置项

    '配置项名称' => [

        '配置项名称' => '配置项的值',

        // ...

    ],

    // ...

];

为了获得最大的灵活性，在TP中配置项采用六层配置模式，即全局配置、应用配置、扩展配置、场景配置、模块配置、动态配置，这些配置由前到后依次被覆盖，也就是说如果应用配置中的配置项和全局配置的配置项冲突时，TP就会采用应用配置项的内容，而如果模块配置项和应用配置项冲突了，那么就会采用模块配置项的内容，使用这种方式可以非常灵活的实现公用配置和个别配置之间的分离。

在TP项目的thinkphp目录下有个convention.php文件，该文件是TP项目的默认全局配置文件，里面包含了应用设置、模块设置、URL设置等，但是该文件通常不需要修改，因为这是全局的配置文件，牵涉到整个TP的正常运行。如果需要修改某些配置项，那么应该在后续的配置文件中进行设置。此外，这里有个注意点，即观察convention.php文件可以看出所有的配置项都采用小写的方式进行编写，这也是在5.x之后官方所推荐的方式。

全局配置的上一层就是应用配置，该配置项位于TP的application目录下，并且由四个文件组成，分别是config.php、database.php、route.php、tags.php，其中database.php表示数据库相关配置，route.php表示路由相关配置，tags.php表示应用行为的扩展定义，而除了数据库、路由、应用行为扩展等信息之外的所有配置项都是在config.php文件完成配置，如图8-3所示。

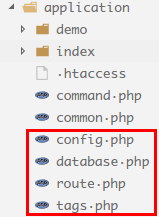


图8-3 ThinkPHP的应用配置项

在应用配置之后就是扩展配置，到笔者写书为止，TP的版本是5.0.10，而在5.0.1版本之前TP可以使用extra\\_config\\_list的配置参数来定义扩展配置，但是在5.0.1之后TP放弃这个做法，而是采用更为简洁的方式，即只要在application目录下创建extra文件夹，并且在该文件夹中创建PHP文件，同时在该文件中返回一个配置数组，如图8-4所示。

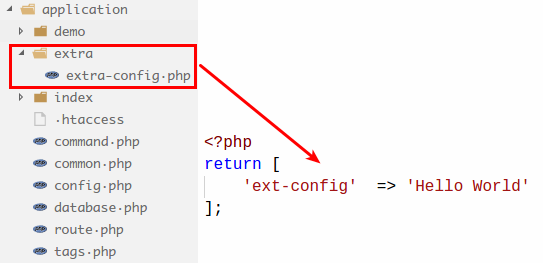


图8-4 ThinkPHP的扩展配置项

在创建完扩展配置文件之后就可以通过TP提供的配置读取函数进行参数的读取，在TP中参数的读取有两个函数，分别是Config::get()和config()，通过向这两个函数传递配置项名称就可以获取配置项的值，实际上config()函数是Config::get()函数的在封装，具体代码如下：

/\*\*

\* 获取和设置配置参数

\* @param string|array  $name 参数名

\* @param mixed         $value 参数值

\* @param string        $range 作用域

\* @return mixed

\*/

function config($name = '', $value = null, $range = '')

{

    if (is\_null($value) && is\_string($name)) {

        return 0 === strpos($name, '?') ? Config::has(substr($name, 1), $range) : Config::get($name, $range);

    } else {

        return Config::set($name, $value, $range);

    }

}

上面的代码就是如何通过封装使config()函数实现参数的读取、设置，以及时候存在的判断，其中$range表示参数数组中二维数组的数组名。在该函数中，首先判断参数值$value时候为空，并且参数名称$name时候是字符串，如果满足$value为空且$name为字符串，那么就表示读取参数，然后通过strpos()函数判断字符“?”时候存在于参数名称的第一个位置，如果“?”在第一个位置，那么就通过Config::has()函数判断参数时候存在，如果“?”不在第一个位置，那么就通过Config::get()函数获取参数。如果不满足$value为空且$name为字符串，那么就通过Config::set()函数进行参数的设置。

了解了扩展配置的使用之后就是场景配置，打开application目录下的config.php文件，在该文件里搜索表示不同场景的配置关键字app\\_status，默认情况下该关键字为空，表示不使用场景配置文件，而当该关键字的值不为空时，那么TP就会到application目录下读取以app\\_status值命名的场景配置文件。例如，当app\\_status值为“school”时，那么TP就会读取application目录下的school.php文件做为场景配置文件，如图8-5所示。

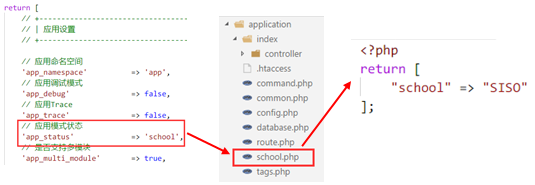


图8-5 ThinkPHP的场景配置项

接下来就是模块配置，该配置文件是每一个模块下的config.php文件，当TP从最初的全局配置读取到达场景配置时，这些配置信息都会对每一个模块造成影响，而只有模块配置才只对所在模块产生影响，不会对其他模块造成影响。在模块配置之后就是TP的最后一个配置层，即动态配置。所谓动态配置是指通过代码的方式，使得程序能在运行时动态的添加配置项到运行的内存中，但是当本条指令执行完成就会清空这些配置项。要用动态配置需要使用TP中的Config::set()或config()助手函数，并且该函数还能够结构数组对象来批量的动态添加配置项，具体代码如下：

Config::set([

    '配置参数1'=>'配置值',

    '配置参数2'=>'配置值'

]);

// 或者使用助手函数

config([

    '配置参数1'=>'配置值',

    '配置参数2'=>'配置值'

]);

8.3 HTTP请求与ThinkPHP控制器的应用

在ThhinkPHP的系统配置完成之后，接下来就要创建各类系统对外的接口来实现系统的交互和数据传递。在本章的第二节讲到ThinkPHP基于MVC架构进行编写，其中和外部进行交互的接口就是Controller（控制器），所以可以认为控制器的定义就是定义系统交互和数据通信的入口。因此，本节将讲解如何创建控制器和目前应用最为广泛，也是最为推荐的RESTful形式接口，以及如何通过接口进行数据的传递。

8.3.1 控制器的创建与重定向

正如前面所说，TP是基于MVC框架进行构建，而该框架的实现又是基于面向对象的相关技术，因此控制器从语法上来说其实质就是一个控制器的类，而不是一个函数。TP的控制器通常位于“application\模块名\controller“目录下，并且控制器的名字要和控制器的文件名相同。TP中控制器的类型分为两种，一种是继承于父类Controller的控制器，用于提供页面渲染和访问接口，而另一种则是不继承于父类Controller的控制器，此类控制器只用于提供访问接口。实质上这两种控制器之间可以项目转换，并且如果全部用功能更为全面，且继承于父类Controller的控制器也不会影响程序的运行，但如果能够进行控制器类型的区分，则可使得程序更加轻量，并且程序运行更加高效。此外，还需要特别说明的是每个控制器都需要有一个命名空间（namespace），用于区分控制器模块和其他模块，示例代码如下：

第一种：只提供访问接口的控制器

namespace application\index\controller; // 命名空间

class Index

{

    public function index()

    {

        return 'index';

    }

}

第二种：具有页面渲染和访问接口的控制器

namespace application\index\controller; // 命名空间

use think\Controller; // 导入控制器命名空间

class Index extends Controller // 继承于父类Controller

{

    public function index()

    {

        return $this->fetch('index'); // 解析并渲染index.html文件

    }

}

而控制器类的文件则在下面的位置：

TP5框架根目录/application/index/controller/Index.php

有了控制器，接下来就是定义其外部可访问的接口，访问接口的定义非常简单，就是在控制器类中添加相应的接口函数，如上面例子中第一种的index()函数。有了接口函数后就可以通过符合格式要求的URL进行访问，并在浏览器中看到打印了“Hello ThinkPHP5”，具体URL地址如下。

http://localhost/TP5/public/index.php/index/index/index

如果此时读者运行第二种控制器就会出现如图8-6所示的结果，但是从结果中可以得到的信息非常少，所以不利于程序调试，此时可以打开application目录下的config.php文件，并且把app\\_debug和app\\_trace的值改为true，再次运行上面的URL就可以看到详细的错误信息，如图8-7所示。从图中可以看到是因为找不到index.html导致的错误，这是因为控制器继承了父类Controller，使得该控制器可以通过fetch()函数进行HTML页面的渲染，但是因为此时的HTML文件不存在，使得渲染错误。

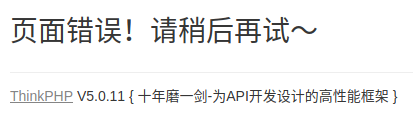


图8-6 ThinkPHP的场景配置项



图8-7 渲染HTML的错误

TP中的HTML渲染文件保存在模块的view目录下，因此首先需要在Index模块目录下创建一个view目录，又因为调用了Index控制器，所以要在view目录下创建一个与Index控制器对应的index目录，最后因为调用的是Index控制器中的index函数，所以要在index目录下创建一个index.html文件来作为index函数的默认HTML模板。默认情况下，模板的文件名和控制器函数的名称相同，不过也可以通过向fetch('模板名称')函数传递模板名称的方式来修改和指定模板。

当模板渲染时TP会在runtime目录下创建相应的缓存文件以提高TP的效率和显示速度，但是如果读者使用的是Linux系统下的Web服务器，那么当进行模板渲染时就会出现如图8-8所示的错误。从图中可以知道，程序运行的错误是因为Web服务器没有权限去修改runtime目录下的文件和文件夹，因此要解决这个问题只需要让Web服务器能够具有runtime文件夹的读写权限，而其中www-data就是Apache服务器的用户名，如果使用的是Nginx服务器，那么就需要把用户改为www，具体指令如下：

sudo chown -R www-data(www) runtime/



图8-8 runtime文件读写权限错误

当一个个HTML页面能够通过不同的控制器进行显示，此时就存在一个页面跳转的问题。在TP中，页面的跳转有两种，一种是操作成功或失败的页面跳转，另一种则是无关操作成功或失败的页面跳转，也叫页面的重定向。要实现这一功能首先就是要让控制器继承于父类Controller，然后根据不同的情况调用$this->success()、$this->error()或$this->redirect()，下面通过一个例子进行讲解，具体如例8-1所示：

namespace app\index\controller;

use think\Controller;

class Index extends Controller

{

    public function index()

    {

        $result = true;

        if ($result) {

            // 跳转到操作成功的页面

            $this->success('操作成功', 'Index/successHtml');

        } else {

            // 跳转到操作失败的页面

            $this->error('操作失败', 'Index/errorHtml');

        }

    }

    public function redirectUrl()

    {

        // 跳转到指定页面

        $this->redirect('Index/redirectHtml');

    }

    public function successHtml()

    {

        return $this->fetch();

    }

    public function errorHtml()

    {

        return $this->fetch();

    }

    public function redirectHtml()

    {

        return $this->fetch();

    }

}

例8-1 页面跳转

在上面的例子中可以看到，通过在浏览器中输入“http://localhost/TP5/public/index.php/index/index/index”就可以执行index()函数，该函数通过变量$result来判断是执行success()还是error()函数，该函数完整的参数列表一共有5个，分别是跳转提示的信息、跳转的地址、返回的数据、跳转等待时间，以及Header信息，具体如下：

/\*\*

\* 操作成功/错误跳转的快捷方法

\* @access protected

\* @param mixed     $msg 提示信息

\* @param string    $url 跳转的URL地址

\* @param mixed     $data 返回的数据

\* @param integer   $wait 跳转等待时间

\* @param array     $header 发送的Header信息

\* @return void

\*/

protected function success/error($msg = '', $url = null, $data = '', $wait = 3, array $header = [])

其中最常用的是开始的三个参数，第一个是页面跳转时显示在过度页面上的信息，第二个是页面跳转的目标地址，第三个是传递给目标页面的URL数据。其中目标地址不仅可以是HTTP地址，而且在大部分情况下传递的是"控制器/控制器方法"这样的地址，如上例所示的“Index/successHtml”就表示跳转到Index控制器中的successHtml方法，如果此时在“view/index”目录下创建了successhtml.html，那么就会在页面中显示该页面的内容。此外还有一个redirect()函数也可以实现页面的跳转或者叫做页面的重定向，该函数完整的参数列表一共有4个，分别是跳转的地址、传递的参数、HTTP状态码，以及隐式传递的参数，与success()和error()函数类似，redirect()函数最常用的参数也是最开始的两个参数，第一个参数传递的是要进行页面跳转的地址，而第二个参数是传递给目标地址的URL数据数组，具体如下：

/\*\*

\* URL重定向

\* @access protected

\* @param string         $url 跳转的URL表达式

\* @param array|integer  $params 其它URL参数

\* @param integer        $code HTTP状态码

\* @param array          $with 隐式传参

\* @return void

\*/

protected function redirect($url, $params = [], $code = 302, $with = [])

8.3.2 HTTP的请求获取与参数绑定

在上节中多次提到了向目标地址传递参数，这个动作在TP的开发中非常常见，例如传递用户名、密码或者购买商品的参数等，而要获取这些数据就需要通过代码Request::instance()来创建Request对象。获取Request对象之后就可以得到各类传入的参数。但是，在获取各类参数之前，首先要判断的就是请求的类型，而在TP5之前TP采用系统常量的方式来完成类型的判断，如IS\\_GET、IS\\_POST等，而从TP5开始则把这些类型的判断全部统一在Request对象中，通过isXXX()函数来判断这些请求的类型，具体代码如下：

// 是否为 GET 请求

if (Request::instance()->isGet()) echo "当前为 GET 请求";

// 是否为 POST 请求

if (Request::instance()->isPost()) echo "当前为 POST 请求";

// 是否为 PUT 请求

if (Request::instance()->isPut()) echo "当前为 PUT 请求";

// 是否为 DELETE 请求

if (Request::instance()->isDelete()) echo "当前为 DELETE 请求";

// 是否为 Ajax 请求

if (Request::instance()->isAjax()) echo "当前为 Ajax 请求";

// 是否为 Pjax 请求

if (Request::instance()->isPjax()) echo "当前为 Pjax 请求";

// 是否为手机访问

if (Request::instance()->isMobile()) echo "当前为手机访问";

// 是否为 HEAD 请求

if (Request::instance()->isHead()) echo "当前为 HEAD 请求";

// 是否为 Patch 请求

if (Request::instance()->isPatch()) echo "当前为 PATCH 请求";

// 是否为 OPTIONS 请求

if (Request::instance()->isOptions()) echo "当前为 OPTIONS 请求";

// 是否为 cli

if (Request::instance()->isCli()) echo "当前为 cli";

// 是否为 cgi

if (Request::instance()->isCgi()) echo "当前为 cgi";

还是以上节中的例子为例，修改页面重定向的代码如例8-2所示：

namespace app\index\controller;

use think\Controller;

class Index extends Controller

{

    public function redirectUrl()

    {

        // 跳转到指定页面

        $this->redirect('Index/redirectHtml', ['cate\_id' => 2]);

    }

    public function redirectHtml()

    {

        if (Request::instance()->isGet()) {

            return "输入的参数类型为GET, cate\_id = " . Request::instance()->route('cate\_id');

        }

    }

}

例8-2 带参数页面跳转

在上面的代码中，redirectUrl()函数通过重定向函数redirect()对目标函数redirectHtml()进行跳转，同时向重定向函数传入一个需要向目标函数传递的参数数组。此时，在浏览器的地址栏中输入“http://localhost/TP5/public/index.php/index/index/redirecturl”，TP就会能实现页面的跳转。细心的读者就会发现地址栏中一些特别的东西，即地址变为了“http://localhost/TP5/public/index.php/index/index/redirecthtml/cate\\_id/2.html”，其中cate\\_id之前的地址就是目标函数的地址，这个很好理解，而是之后的“cate\\_id/2.html”则有些不同，但是仔细观察重定向函数redirect()就会发现“cate\\_id/2.html”实际上就是该函数的第二个参数，只是TP把这个参数以PATHINFO形式进行传递，所谓PATHINFO就是把参数名和参数值以“/”作为分隔符组成的一串数据，并将其跟在地址后面。当采用PATHINFO传递数据时，数据以GET方式进行传递，所以在目标函数redirectHtml()中可以通过isGet()函数来判断数据传递的类型，又因为采用的是PATHINFO，所以需要通过请求对象Request的route()函数来获取数据。

从上面的表述可以看到，要读取以PATHINFO形式存在的数据，其流程是比较复杂的，而在TP中为了简化这一过程提供了一种叫做“控制器函数的参数绑定”的方式。所谓“控制器函数的参数绑定”就是在编写控制器函数时为该函数提供一个或多个形参，当通过PATHINFO传递数据，那么TP就会自动把PATHINFO中的数据和传入函数的形参进行匹配，如果传入参数的参数名和形参名匹配，那么就可以把传入数据的值传递给形参，从而实现数据的传递。接下来修改例8-2中的redirectHtml()函数，使其实现参数绑定的功能，具体如下：

public function redirectHtml($cate\_id)

{

    return "输入的参数类型为GET, cate\_id = " . $cate\_id;

}

在上面的函数中，为其添加一个$cate\\_id的形参，当调用重定向函数redirect()时，TP会自动把$cate\\_id作为参数传递到redirectHtml()函数，并且TP会判断传入的参数名是否为“cate\\_id”，如果是直接把数据传递给$cate\\_id。这样的做法不仅可以提高数据解析的效率，还能增加代码的可读性。此外，通过参数绑定还可以检查传递参数时的错误问题，因为当参数传递到redirectHtml()函数时，TP如果发现传入的参数无法与绑定的形参进行匹配，那么就会出现如图8-9的错误。为了避免图8-9的问题，TP建议在开发控制器函数时为需要进行参数绑定的函数添加一个默认值，当向控制器函数传入的数据异常或者无法和控制器函数的形参匹配时，那么TP就会使用默认参数来进行赋值，从而避免错误的出现，具体代码如下：

public function redirectHtml($cate\_id = 0)

{

    return "输入的参数类型为GET, cate\_id = " . $cate\_id;

}



图8-9 参数绑定的错误

从上面的内容可以看到，在Web开发中数据传入的方式是多种多样的，有GET、POST、PATHINFO等，因此TP提供了多种数据读取的函数，从这些函数功能上来说可以分为两类，即判断参数是否存在和参数获取方式，具体他如下：

1、判断参数时候存在

在TP中判断参数是否存在的函数是Request::instance()->has()，该函数共有三个参数，分别是要判断的目标变量名、传入参数的类型，以及是否验证空数据，其中第一个参数目标变量名是必填项，其他两个参数都不是必填参数，当向第二参数传入参数的类型传递字符串“GET”时，表示该判断函数是在GET数据上进行判断，而传递“POST”时表示在POST数据上进行判断，又或是在其他传入方式上进行判断，当向最后一个参数是否验证空数据传递true时，表示当传入的数据为空时，则函数返回false，否则就返回true，当传递false时，则表示不进行控制判断，具体函数原型和使用如下：

/\*\*

\* 是否存在某个请求参数

\* @access public

\* @param string    $name 判断的目标变量名

\* @param string    $type 传入参数的类型

\* @param bool      $checkEmpty 是否验证空数据

\* @return mixed

\*/

public function has($name, $type = 'param', $checkEmpty = false)

// has()函数的使用

Request::instance()->has('id','get');

Request::instance()->has('name','post');

2、参数获取方式

**（1）Request::instance()->param()：**该函数是TP提供的一个用于自动识别GET、POST或者PUT请求的一种变量获取方式，同时该函数也是TP所推荐的获取请求参数的方法，该函数有三种使用方法，分别是param(“变量名”)、param()、param(true)，其中第一种表示通过变量名获取变量值，第二种表示获取所有的变量和变量值，并以数组的形式返回，第三种则是返回所有数据，包含上传的文件。

**（2）Request::instance()->get()：**该函数是TP提供的一个用于识别GET请求的一种变量获取方式，该函数有两种使用方法，分别是get(“变量名”)、get()，其中第一种表示通过变量名获取变量值，第二种表示获取所有GET中的变量和变量值，并以数组的形式返回。

**（3）Request::instance()->post()：**该函数是TP提供的一个用于识别POST请求的一种变量获取方式，该函数有两种使用方法，分别是post(“变量名”)、post()，其中第一种表示通过变量名获取变量值，第二种表示获取所有POST中的变量和变量值，并以数组的形式返回。

**（4）Request::instance()->put()：**该函数是TP提供的一个用于识别PUT请求的一种变量获取方式，该函数有两种使用方法，分别是put(“变量名”)、put()，其中第一种表示通过变量名获取变量值，第二种表示获取所有PUT中的变量和变量值，并以数组的形式返回。

**（5）Request::instance()->request()：**该函数是TP提供的一个用于识别REQUEST请求的一种变量获取方式，该函数有两种使用方法，分别是request(“变量名”)、request()，其中第一种表示通过变量名获取变量值，第二种表示获取所有REQUEST中的变量和变量值，并以数组的形式返回。

**（5）Request::instance()->session()：**该函数是TP提供的一个用于识别SESSION请求的一种变量获取方式，该函数有两种使用方法，分别是session(“变量名”)、session()，其中第一种表示通过变量名获取变量值，第二种表示获取所有SESSION中的变量和变量值，并以数组的形式返回。

**（6）Request::instance()->cookie()：**该函数是TP提供的一个用于识别Cookie请求的一种变量获取方式，该函数有两种使用方法，分别是cookie(“变量名”)、cookie()，其中第一种表示通过变量名获取变量值，第二种表示获取所有Cookie中的变量和变量值，并以数组的形式返回。

8.3.3 RESTful的概念与Rest控制器的使用

在上一节讲了如何通过创建控制器和控制器函数来为前端提供各类的接口，然后由于应用的种类各种各样，并且开发者的水平参差不齐，使得在很长时间里面Web开发的接口（API）都相当的混乱，系统和系统之间对接困难，但是到了2000年这样的现象因为一篇博士论文出现开始发生的改变。2000年，HTTP协议的主要设计者、Apache服务器软件的作者之一、Apache基金会第一任主席Roy Thomas Fielding在加利福尼亚大学欧文分校攻读博士学位时发表了一篇名为《Architectural Styles and the Design of Network-based Software Architectures》（架构风格与基于网络应用软件的架构设计）的博士论文，在该文中提出了一个对互联网发展具有重要意义的概念——Representational State Transfer，即REST概念。

REST的中文翻译叫做“资源的表现层状态转化”，其中所谓的“资源”是指用于定位资源的URL地址，“表现层”是指以JSON、XML或者其他数据类型为形式的数据，而“状态转化”则是指以HTTP的POST、GET、PUT、DELETE操作来表达对资源的增、删、改、查操作。在Roy Thomas Fielding的认知中，Web API其本质是对Web中各类资源的增、删、改、查操作，如添加学生信息、查询学员信息、修改学生信息和删除学生信息等，但是由于过去各系统的开发者的认识各不相同，使得开发的Web API很难在各系统之间直接调用，而Roy Thomas Fielding发现了这个问题，并且把这些问题进行抽象和总结，提出在设计Web API中的两点建议，并且只要符合这两点建议的Web API就是RESTful，具体建议如下：

1、以名词结尾。因为Web资源是一个很具体的实体，而定位资源的URL地址应表达的是资源内容，而不是对资源的操作，所以Roy Thomas Fielding建议在设计URL时应以资源的名字作为结尾，表示该URL将对某个资源进行操作，而是把动作直接加在URL中，即URL中不应出现动词。

2、以HTTP方法来表达操作。既然URL不能表达对资源的操作，那么Roy Thomas Fielding就为HTTP的POST、GET、PUT、DELETE这四种操作赋予了资源操作的语义，如GET表示读取一个或多个数据资源，POST表示新建一个或多个资源，PUT表示更新一个或多个资源，DELETE表示删除一个或多个资源。

从上面的两个点建议可以看到，Web API在设置时其URL表示所要操作的资源，并且在API中不说明要操作的动作，而是在API中通过鉴别不同的提交动作来判断是要读取资源还是删除资源等，例如要设计一个符合RESTful的学生信息操作接口，其Web API的URL应该如下：

http://服务器地址/资源链接前缀/studentinfo

1、POST：新建一个学生信息

2、GET：获取一个或多个学生信息

3、PUT：更新一个或多个学生信息

4、DELETE：删除一个或多个学生信息

在TP5的众多特点中，能够创建和使用符合RESTful接口标准的控制器也是其一大特点，然后TP通过URL路由的方式实现了不同HTTP操作与不同函数之间的对应关系。在TP5中通过创建要资源控制器的方式来实现RESTful接口，并且TP5提供了一条便捷的指令来帮助开发者进行自动创建，只需要开发人员把控制台或者终端切换到项目的根目录，然后执行如下指令：

php think make:controller index/Blog

在上面的指令中“php think make:controller”表示创建一个控制器，而后面的”index/Blog“表示在index模块下创建一个Blog控制器。指令执行完成后就会在index模块的controller文件夹下发现多出一个Blog.php，该文件就创建好的资源控制器，然后打开application目录下的路由配置文件route.php，使用路由类Route中的资源来绑定资源控制器和路由之间的关系，添加如下的代码：

use think\Route;

Route::resource('blog','index/blog');

因为原本路由配置文件route.php中已经包含了一个简单的路由范例，但是该范例是通过return的方式进行设置，而上面所说的资源路由设置则是通过路由类Route来进行设置，因此这里需要注意的是要把设置代码添加在return上面，而不能在return下方。当创建完资源控制器，并添加资源路由之后就可以通过URL进行访问，打开Blog.php就会看到如下代码：

namespace app\index\controller;

use think\Controller;

use think\Request;

class Blog extends Controller

{

    /\*\*

     \* 显示资源列表

     \*

     \* @return \think\Response

     \*/

    public function index() { }

    /\*\*

     \* 显示创建资源表单页.

     \*

     \* @return \think\Response

     \*/

    public function create() { }

    /\*\*

     \* 保存新建的资源

     \*

     \* @param  \think\Request  $request

     \* @return \think\Response

     \*/

    public function save(Request $request) { }

    /\*\*

     \* 显示指定的资源

     \*

     \* @param  int  $id

     \* @return \think\Response

     \*/

    public function read($id) { }

    /\*\*

     \* 显示编辑资源表单页.

     \*

     \* @param  int  $id

     \* @return \think\Response

     \*/

    public function edit($id) { }

    /\*\*

     \* 保存更新的资源

     \*

     \* @param  \think\Request  $request

     \* @param  int  $id

     \* @return \think\Response

     \*/

    public function update(Request $request, $id) { }

    /\*\*

     \* 删除指定资源

     \*

     \* @param  int  $id

     \* @return \think\Response

     \*/

    public function delete($id) { }

}

从上面的代码不难看出其对资源的操作，但是需要注意的是因为这里已经注册的资源路由，因此如果直接访问该控制就会出现如图8-10的错误。则是因为在注册资源路由时已经把URL中的路径”index/blog“注册为”blog“，因此在访问”index/blog“时，应该是用“blog”来替代，不应出现如下的链接：

http://localhost/TP5/public/index.php/index/blog/index

而是应该以下面这样的形式进行访问

http://localhost/TP5/public/index.php/blog/index

又因为在资源路由注册时，系统已经把所有的函数进行了路由绑定，因此正确的访问资源方法应该如表8-3所示。

表8-3 TP根目录下的文件与文件夹说明

|  |  |  |  |
| --- | --- | --- | --- |
| 请求类型 | 访问链接 | 对应操作 | 说明 |
| GET | .../index.php/blog | index | 显示资源列表 |
| GET | .../index.php/blog/create | create | 显示创建资源表单页 |
| POST | .../index.php/blog | save | 保存新建的资源 |
| GET | .../index.php/blog/id | read | 显示指定的资源 |
| GET | .../index.php/blog/id/edit | edit | 显示编辑资源表单页 |
| PUT | .../index.php/blog/id | update | 保存更新的资源 |
| DELETE | .../index.php/blog/id | delete | 删除指定资源 |



图8-10 资源控制器的访问错误

仔细观察Blog.php文件就会发现在该文件中多次出现了Request类，并且该类出现的位置都是要提交数据的函数，这是因为Request类存放的是当前请求的信息，包括在使用POST和PUT请求时所提交的数据，以及各类请求的信息，这个信息可以分为四类，分别是获取URL信息、获取模块/控制器/操作名称、获取请求参数、获取路由和调度信息，具体代码如下：

1、获取URL信息

$request = Request::instance();

// 获取当前域名

$request->domain();

// 获取当前入口文件

$request->baseFile();

// 获取当前URL地址 不含域名

$request->url();

// 获取包含域名的完整URL地址

$request->url(true);

// 获取当前URL地址 不含QUERY\_STRING

$request->baseUrl();

// 获取URL访问的ROOT地址

$request->root();

// 获取URL访问的ROOT地址

$request->root(true);

// 获取URL地址中的PATH\_INFO信息

$request->pathinfo();

// 获取URL地址中的PATH\_INFO信息 不含后缀

$request->path();

// 获取URL地址中的后缀信息

$request->ext();

2、获取模块/控制器/操作名称

$request = Request::instance();

// 获得当前模块名称

$request->module();

// 获得当前控制器名称

$request->controller();

// 获得当前操作名称

$request->action();

3、获取请求参数

$request = Request::instance();

// 获得请求方法

$request->method();

// 获得资源类型

$request->type();

// 获得访问ip地址

$request->ip();

// 获得是否AJax请求

var\_export($request->isAjax(), true);

// 获得请求参数

dump($request->param());

// 获得请求参数：仅包含name

dump($request->only(['name']));

// 获得请求参数：排除name

dump($request->except(['name']));

4、获取路由和调度信息

$request = Request::instance();

// 获得路由信息

dump($request->route());

// 获得调度信息

dump($request->dispatch());

8.4 ThinkPHP中路由的设置与URL的生成

在复杂的Web系统开发中，控制器的使用通常会采用分层、分级的方式进行实现，每个控制器又会包含多个控制器函数，并且为了保证代码的可读性，因此控制器名和函数名都会采用动宾结合的方式进行命名，但是这样的做法就会造成URL地址非常长，不利于Web API的使用和维护。为了解决这个问题，TP提供了路由功能来实现URL地址的隐藏，以及把原本复杂的URL地址变为更加容易理解，并且结构简单的地址，例如下面的URL。本节讲讲述如何在ThinkPHP实现路由，以及如果通过代码实现地址的自动生成。

路由之前的地址：

http://server/public/index.php/product/5/123

路由之后的地址：

http://server/product/5/123

8.4.1 路由的定义与批量注册

TP中路由的添加通常放在路由配置文件“application/route.php”文件中，并且采用Route类中的rule()方法进行注册和定义，其rule()函数的原型如下：

// rule()函数的原型

Route::rule('路由表达式', '路由地址', '请求类型', '路由参数（数组）', '变量规则（数组）');

使用方法如下：

use think\Route;

// 注册路由到index模块的News控制器的read操作

Route::rule('new/:id','index/News/read');

在上面的例子中，Route::rule()函数中的第一个参数表示路由的表达式，其中“:id”表示new后面跟的是一个名字为id的参数，而第二个参数表示需要进行路由的原地址。因此进过路由之后，当访问目标地址“http://server/public/index.php/new/ID参数”就会自动的路由到地址“http://server/public/index.php/index/News/read/ID参数”中，并且原来的访问地址就会自动失效。

在上一节讲到在创建RESTFul接口时，主要是通过POST、GET、PUT、DELETE等HTTP请求类型来实现对数据的增、删、改、查，因此在设计路由时也可以向路由函数的第三个参数传递HTTP请求类型，从而实现在不同请求类型下执行不同的函数，具体方法如下：

use think\Route;

// 注册路由到index模块的News控制器的read操作

Route::rule('new/:id','index/News/read', 'POST');

上面的例子中设置的请求类型为POST，那么当系统通过POST方式向“new/:id”提交数据时，TP系统会自动路由到“index/News/read”中。此外，如果该路由需要支持多种请求类型，那么就可以通过HTTP请求类型之间取或的方式实现，具体方法如下：

use think\Route;

// 注册路由到index模块的News控制器的read操作

Route::rule('new/:id','index/News/read', 'POST|GET');

在创建路由时，路由表达式可以有多种形式，如不带参数，只访问某个地址的静态路由，又如带参数，并访问某个地址的动态路由等，因此路由表达式的创建规则如表8-4所示。

表8-4 路由表达式的规则

|  |  |  |  |
| --- | --- | --- | --- |
| 类型 | 表达式 | 例子 | 说明 |
| 静态地址路由 | '路由地址' => '原父地址/原子地址' | 'my' => 'Member/myinfo' | 只要访问'my'就会被路由到'Member/myinfo' |
| 动态地址 | '路由地址/:参数名/:参数名/...' => '原父地址/原子地址' | 'blog/:id' => 'Blog/read' | 当向blog地址传递参数ID时，会自动路由到'Blog/read' |
| 可选参数的动态地址 | '路由地址/:参数名/.../[:参数名]' => '原父地址/原子地址' | 'blog/:year/[:month]' => 'Blog/archive' | 要访问'Blog/archive'只有通过访问blog，并传递参数year才能实现，而最后的一个参数month是可选参数，表示在访问"blog/:year"时可以传递month，也可以不传递month |

目前已经知道如何通过TP中的Route::rule()函数创建一条路由，其实TP还提供了两种更多方便的方式来同时创建多条路由，一种是通过Route::rule()函数来完成，还有一种是在route.php文件中通过编写路由数组的方式来实现路由的注册，其实质是相同，只是采用了两种不同的形式，具体如下：

1、采用Route::rule()函数进行批量注册

该函数的第一个参数，除了可以直接传递一个字符串形式的路由表达式之外，还可以传递一个以路由表达式为数据项的数组来进行批量的路由注册，具体方法和实例如下：

Route::rule([

'路由规则1'=>'路由地址1和参数1',

'路由规则2'=>'路由地址2和参数2',

...

],'空字符串','请求类型','匹配参数（数组）','变量规则');

实例如下：

Route::rule([

    'new/:id'  =>  'News/read',

    'blog/:id' =>  'Blogs/read',

    ...

],'','GET',['ext'=>'html'],['id'=>'\d+']);

2、采用路由数组的方式进行批量注册

除了上面采用函数的方法进行路由定义和添加之外，还可以在路由配置“application/route.php”中通过返回数组的方式直接进行路由的定义，具体方法如下：

return [

    '路由规则1'   => ['路由地址1',['method' => '请求类型']],

    '路由规则2'   => ['路由地址2',['method' => '请求类型']],

    ...

];

实例如下：

return [

    'new/:id' => 'News/read',

    'blog/:id' => ['Blog/update',['method' => 'post|put']],

];

8.4.2 路由的分组与URL生成

在上一节讲了TP中基本的路由使用方法，但在实际开发中很多路由都会指向同一个控制器，然后再分解到控制器中的不同函数或者通过HTTP方法调用不同的控制器，当TP进行路由时会遍历完整的路由规则，而对于相同控制器下的路由，TP提供了更为高效的路由系统，即分组路由，通过分组路由的设置可以把具有相同前缀（即控制器）的路由进行合并，从而避免了TP每次都去遍历完整的路由规则，具体代码如下：

原本在route.php中的路由规则：

return [

    'blog/:id' => ['Blog/read', ['method' => 'get']],

    'blog/:name' => ['Blog/read', ['method' => 'post']]

]

分组后的路由规则：

return [

    '[blog]' => [

        ':id'   => ['Blog/read', ['method' => 'get']],

        ':name' => ['Blog/read', ['method' => 'post']]

    ]

]

除了在route.php中通过return方式进行路由数组的返回外，TP还能通过Route::group()函数进行路由的分组，具体代码如下：

Route::group('blog',[

    ':id'   => ['Blog/read', ['method' => 'get']],

    ':name' => ['Blog/read', ['method' => 'post']],

]);

在TP中由于存在各种的路由、控制器，因此造成URL地址的种类非常多，而TP提供了一种统一生成URL的方法，并且可以支持所有的路由方式，使得开发人员无需再为因为路由定义和变化造成URL地址的改变而烦恼。在TP中URL的生成是由URL类中的build()函数来完成，具体代码如下：

Url::build('地址表达式', ['参数'], ['URL后缀'], ['域名'])

其中“地址表达式”是表示要生成URL的“模块=>控制器=>操作”，而“参数”则表示向该地址传递的参数，而“URL后缀”则表示URL的后缀名，如.html、.shtml等，最后一个“域名”则表示生成URL地址中时候要加入域名，具体代码如下：

添加路由：

Route::rule('blog/:id','index/blog/read');

生成URL地址的函数：

// 生成index模块 blog控制器的read操作 URL访问地址

Url::build('index/blog/read', ['id'=>5,'name'=>'thinkphp'], 'shtml');

生成的URL地址：

/index.php/blog/5/name/thinkphp.shtml

在上面的代码中，当通过Url::build()函数生成URL地址，会检索相关的路由配置文件，并且在其基础上生成经过路由定义地址，并且在地址的最后添加后面html，当如果此时设置的URL后缀为“shtml”，那么生成的URL后缀就会变成“shtml”。

8.5 ORM模型的创建与数据库的CURD操作

模型是MVC结构中最为重要的部分，是实现整个系统逻辑的核心，TP中的模型主要分为两类，一类是用于实现数据库操作的模型，也是本节讲解的重点，而另一类则是用于实现系统业务模型的模型，这个模型通常是在数据库模型的上层，完成各种业务模块之间的数据交互。对于具有数据库操作的模型来说，TP已经提供了非常方面的类和函数进行数据库的增、删、改、查。在TP内部已经对数据库的访问进行了抽象，把不同的数据库操作进行封装，对于开发者来说只需要使用统一的数据库操作类或函数就可以实现对数据的操作，而无需针对不同的数据库编写不同的代码和底层实现，数据库类会自动调用内置的数据库驱动来处理。TP中对于数据库的操作采用PHP数据库对象PDO（PHP Data Object）来实现，目标TP支持大部分的数据库，如MySQL、SQLServer、PostgreSQL、SQLite等数据库的支持。

8.5.1 数据库的的参数配置

在TP中初级的模型通常始于数据库，因此要让TP能够顺利的连接数据库，首先就要在TP中对需要连接的数据库进行配置。在TP中，数据库的配置参数文件放在“application”目录下的“database.php”中，打开该文件就可以就会发现其本质是一个很长的数组，其中定义了很多与数据库连接相关的信息，如果数据库类型、数据库账户、数据库编码方式等，下面就给出表来进行详细说明常用的数据库配置参数，如下8-5所示。

表8-5 常用的数据库连接参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 值 | 默认值 | 说明 |
| type | mysql、pgsql、sqlite、sqlsrv | 无 | 支持的数据库类型，目前支持MySQL、SQLServer、PostgreSQL、SQLite，如果还需要支持的其他数据库，就需要自己编写驱动文件 |
| hostname | 数据库服务器地址 | 127.0.0.1 | 数据库服务器的IP地址或域名地址 |
| database | 数据库名称 | 无 | 要连接的数据库名称 |
| username | 用户名 | 无 | 连接数据库的用户名 |
| password | 密码 | 无 | 连接数据库的密码 |
| hostport | 数据库服务器端口号 | 无 | 不同的数据库服务器，其端口号都不相同，例如MySQL服务器的默认端口是3306 |
| dsn | 连接数据库的DSN | 无 | 数据库连接DSN信息，其信息格式为“数据库类型://数据库连接用户名:数据库连接密码@数据库服务器地址:数据库服务器端口号/连接的数据库名称#数据库采用的编码方式”，以本地的MySQL数据库为例，其DSN信息为“mysql://root:1234@127.0.0.1:3306/thinkphp#utf8” |
| params | 数据库连接参数 | 空 | 数据库连接时的PDO参数，可以参考PHP在线手册中的以PDO::ATTR\_开头的常量，不同对于一般应用来说，改参数需要设置的情况较少 |
| charset | 数据库编码方式 | utf8 | 数据库中的数据默认采用的编码方式 |
| prefix | 数据库表前缀 | 无 | 当在一个数据库中有个多张数据表时，用于快速定这些数据表的前缀，例如think\_user、think\_role等，那么此时就可以在该参数中设置一个前缀为“think\_”。 |
| debug | true、false | false | 数据库调试模式 |
| fields\_strict | true、false | true | 是否严格检查字段是否存在 |
| resultset\_type | array | array | 数据集返回类型，默认情况下数组 |
| auto\_timestamp | true、false | false | 自动写入时间戳字段 |
| datetime\_format | 符合PHP规范的时间格式 | Y-m-d H:i:s | 时间字段取出后的默认时间格式，具体可是可以参PHP时间格式(http://php.net/manual/zh/function.date.php) |

当配置完成之后就可以连接数据库，并进行数据的读取，在数据库服务器中创建数据库“think\\_test”，然后在其中创建一张person\\_info的数据表，并加入3条测试数据，具体SQL代码如下：

--创建数据库think\_test

CREATE DATABASE  IF NOT EXISTS `think\_test`;

--选择数据库think\_test

USE `think\_test`;

--创建数据表person\_info

DROP TABLE IF EXISTS `person\_info`;

CREATE TABLE `person\_info` (

  `id` int(10) unsigned NOT NULL AUTO\_INCREMENT,

  `name` varchar(45) NOT NULL,

  `sex` tinyint(1) NOT NULL DEFAULT '1',

  PRIMARY KEY (`id`),

  UNIQUE KEY `id\_UNIQUE` (`id`)

);

--向数据表person\_info中插入数据

INSERT INTO `person\_info` VALUES (1,'张三',1, 90),(2,'李四',0, 80),(3,'王五',1, 77);

当数据完成后，就在可以在Index模块的index()函数中可以使用Db类中query()和execute()函数来读取数据库的操作。在下面的例子中将通过查询语句来实现数据表“person\_info”的查询，并且为了得到更好的显示效果，可以在config.php文件中把“default\\_return\\_type”的值由“html”改成“json”，具体代码和结果如下：

public function index()

{

    $data = Db::query('select \* from person\_info');

    return $data;

}

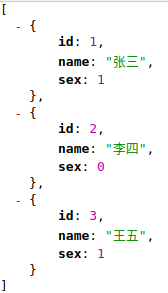


图8-11 数据库查询结果

在上面例子中讲到Db类中有两个常用的函数，分别是query()和execute()函数，这两个函数的区别在于query()通常用于执行“select”查询语句，因为该函数会返回一个查询结果的数据，而execute()函数则通常用于执行“insert”、“delete”、“update”语句，并且返回这些语句影响的行数，函数原型如下：

/\*\*

\* 执行查询 返回数据集

\* @access public

\* @param string      $sql    sql指令

\* @param array       $bind   参数绑定

\* @param boolean     $master 是否在主服务器读操作

\* @param bool|string $class  指定返回的数据集对象

\* @return mixed

\* @throws BindParamException

\* @throws PDOException

\*/

public function query($sql, $bind = [], $master = false, $class = false)

/\*\*

\* 执行语句

\* @access public

\* @param string $sql  sql指令

\* @param array  $bind 参数绑定

\* @return int

\* @throws BindParamException

\* @throws PDOException

\*/

public function execute($sql, $bind = [])

这两个函数的参数列表中最重要的就是前面两个参数，第一个参数顾名思义就是完整的SQL语句，而第二个参数表示的是SQL语句中通过“？”或者“:参数名”的方式进行占位后对应的参数数组，通过占位符的使用可以简化SQL语句的编写，免去了通过字符串拼接的方式来实现完整的SQL语句，具体例子如下：

// “？”占位符

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

Db::execute('insert into think\_user (id, name) values (?, ?)',[8,'thinkphp']);

// “:参数名”占位符

Db::query('select \* from think\_user where id=:id',['id'=>8]);

Db::execute('insert into think\_user (id, name) values (:id, :name)',['id'=>8,'name'=>'thinkphp']);

占位符的使用有两类，在第一类“？”占位符中“？”的顺序就对应后面参数数组的顺序，也就是说第一个“？”对应后面参数数组中的第一个值，而第二个“？”则对应后面参数数组中的第二个值，而对于第二类“:参数名”占位符来说其顺序和参数名是一一对应，与排列的顺序没有关系。

8.5.2 ORM数据库模型的定义与基本的CURD操作

使用TP构建一个Web系统，其对数据库操作是非常常见，也是必不可少的一个部分，目前我们已经知道在TP中如果要对数据库进行操作，可以通过Db类中的query()和execute()等函数进行数据的增、删、改、查操作，这种方法固然简单和直接，但是这样的操作是直接对数据进行操作，破坏了面向对象中基本特性，即封装性。因此，TP为了解决这个问题引入了ORM（Object-Relational Mapping，对象关系映射）的概念，通过这个概念的引入能够把创建的对象映射到基于SQL的关系型数据库中，使得对数据库的操作变为对对象的操作，而不再需要直接和数据或者SQL语句打交道，ORM技术为对象和数据库之间建立了一个桥梁，使得对象数据和数据库数据之间可以项目转化，同时也提高了系统的稳定性。在TP中体现ORM思想的就是模型，每个模型都对应一个数据库中的数据表，通过对模型对象的使用来实现对数据库的操作。在TP中模型的定义非常简单，当数据库配置文件配置完成后只需要在“模块”目录下创建一个“model”文件夹，并且在该文件中创建数据表对应的模型文件即可。但是要注意的是，模型文件和模型类的命名有其自身的规定，必须采用驼峰命名法，即除去表前缀后的数据表，把以下划线为间隔的单词以首字母大写的方式进行组合，如表8-5所示。

表8-5 模型文件的命名

|  |  |
| --- | --- |
| 模型名 | 约定对应数据表（假设数据库的前缀定义是 think\_） |
| Person.php | think\_person |
| PersonInfo.php | think\_person\_info |

当把模型文件创建完成之后，并在该文件中添加一个和模型文件名相同的类，为了时模型类能够时间ORM思想，那么就需要让该类继承于父类Model，同时在该类的成员变量中添加主键变量，例如此时在数据库中有一个数据表叫“think\_person”，当在数据库配置文件中已经配置了数据表前缀为“think\\_”后，那么就只需要创建一个Person.php，并在该文件中添加一个类名为Person的类，当模型文件创建完毕后就可以在需要的地方创建模型对象，具体代码如下：

namespace app\index\model; // 命名空间不能少

use think\Model; // 引入模型的命名空间

class Person extends Model // 创建think\_person数据表对应的模型类

{

    protected $pk = 'id'; // 添加think\_person数据表对应的主键

}

$person = new Person() // 在需要的地方创建模型对象

当模型对象创建完成之后就是通过参数传递等方法，实现数据表的操作，下面将通过四点内容来详细讲解如果通过模型来实现增、删、改、查。

1、数据的增加：通过模型向数据表中添加数据，主要有两种方式，分别是属性赋值法和数组赋值法，具体代码如下。

// 第一种方式：属性赋值法

$person = new Person();

$person->name = '李磊';

$person->sex = 1;

$person->save(); // 保存到数据表中

// 第二种方式：数组赋值法

$person = new Person();

$person->data([

    'name' => '李磊',

    'sex' => 1

]);

$person->save(); // 保存到数据表中

在Web开发中，经常会使用通过HTTP POST或者GET方式来获取数据，然后在添加到数据表的情况，但是这样传来的数据除了包含数据表所需要的字段之外，还经常会包含了数据表不需要的字段，而开发人员需要通过提取这些有效的数据来更新数据表，而在TP提供了一种非常方便的方式来解决这个问题，即在调用模型save()函数之前调用allowField()函数，并传入参数true，当添加数据完成之后就可以调用模型属性的方式来获取ID，具体代码如下：

// 创建模型对象

$person = new Person();

// 获取要添加的数据

$person->data($\_POST);

// 调用allowField()函数来过滤POST数组中的非数据表字段数据

$person->allowField(true)->save();

// 获取自增ID

echo $person->id;

上面的案例都是添加一条数据，实际上还有一种情况也是开发人员经常碰到的，即添加多条数据，很多开发人员在遇到这种情况的时候会使用如下的代码：

// 添加第一条数据

$person = new Person();

$person->name = '李磊';

$person->sex = 1;

$person->save();

// 添加第二条数据

$person->name = '张三';

$person->sex = 0;

$person->save();

但是实际运行时就会发现这种写法没办法实现第二条数据的添加，而是把第一条添加的数据信息给重新覆盖，那是因为当填完第一条数据之后Person对象已经和新添加的第一条数据之间建立了绑定，因此无法再添加第二条数据，那么如果要实现数据的批量添加就需要使用saveAll()函数，具体代码如下：

$person = new Person();

// 构建批量数据的数组

$datas = [

    ['name' => '李磊', 'sex' => 0],

    ['name' => '韩梅梅', 'sex' => 1]

];

// 添加批量数据

return $person->saveAll($datas);

当通过saveAll()函数批量添加数据之后，该函数会返回一个包含新增数据，并且这个数据中包含数据ID的数组，如图8-11所示。



图8-11 批量添加数据

2、数据的更新：通过模型更新数据表中的数据，主要有两种方式，一种是通过save()函数进行数据更新，另外一种则是通过模型对象提供的函数进行更新，具体代码如下：

$person = new Person();

// 方式一：通过save()函数更新数据表

$person->save([

    'name'  => '海盗路飞',

    'sex' => 1

],['id' => 1]);

在上面的代码中可以到save()函数一共要传递两个参数，第一个参数是要更新的数据，第二个参数是更新数据的条件，因此上面的语句可以翻译为如下的SQL语句：

UPDATE `think\_test`.`think\_person` SET `name`='海盗路飞', `sex`='1' WHERE `id`='1';

当使用save()函数进行数据更新时，该函数会返回操作影响的函数，如果上面的数据更新语句更新了数据表中的两项，那么save()函数就会返回2，如果上面的数据更新语句更新了数据表中的一项，那么save()函数就只会返回1。如果需要更新多条数据，就需要通过saveAll()函数，但是这里需要注意的是通过saveAll()函数批量更新数据时，TP只会根据主键值进行更新，因此在传入saveAll()函数的数据集时需要包含主键信息，那么saveAll()函数才能正常完成数据的批量更新，具体代码如下：

$person = new Person();

// 构建批量更新的数据，一定要包含主键ID的信息

$datas = [

    ['id'=>1, 'name'=>'海盗路飞', 'sex'=>'1'],

    ['id'=>2, 'name'=>'山治', 'sex'=>'1']

];

// 更新数据，返回完整的数据列表

$person->saveAll($datas);

除了采用上面的方式来更新数据表信息之外，还可以使用语义更加明确的模型对象函数where()和update()来实现数据的更新，其中where()用于传递一个查询条件字符串，而update()函数则是传一个要更新的数据，并且返回数据更新后影响的行数，具体代码如下：

$person = new Person();

// 使用where()函数和update()函数进行数据更新

$person->where('id=1')->update(['name' => '李磊']);

3、数据的删除： 通过模型删除数据表中的数据，主要使用模型中提供的where()函数来添加删除的条件，然后再利用delete()函数来实现删除，并且delete()函数会返回删除数据项时影响的行数，具体代码如下：

$person = new Person();

// 使用where()函数和delete()函数进行数据的删除

return $person->where('id=1')->delete();

执行上面的程序，会看到delete()函数返回数字1，这是因为删除的条件是根据主键来进行删除，而主键是唯一的，因此只会影响1条数据。

4、数据的查询：通过模型来查询数据表中的数据都需要先使用where()函数进行检索数据的条件添加，当过滤完成后就可以通过两种方式进行查询，一种是通过模型对象的find()函数进行数据查询，另外一种则是通过模型对象的select()函数进行数据查询，其区别在find()函数只返回一个一维数组，并且一个数组中只存放一个数据，如果查询的结果实际上为多个数据，那么也只返回这些数据中的第一个数据，因此find()函数更加适合通过主键进行查询的语句，而select()函数则返回一个二维数组，并且这个二维数组中可以存放多条数据，如果查询的结果为多个数据，那么这个二维数组则包含所有数据，如果查询的结果只有一条数据，那么这个二维数组也就只包含一个数据，具体代码如下：

$person = new Person();

// 方式一：使用where()函数和find()函数进行数据的查询

return $person->where('id > 0')->find();

// 方式二：使用where()函数和select()函数进行数据的查询

return $person->where('id > 0')->select();

从上面两种查询数据的方式可以看到，其条件语句都相同，唯一的不同就在于使用find()函数或者select()函数，其结果如图8-12所示。。但在数据查询时有一点需要注意，如果需要在模型内部进行查询，那么就不能使用“$this->字段名”的方式来获取数据，而是应该使用“$this->getAttr('字段名')”进行替代。



图8-12 两种方式进行数据查询的结果

8.5.3 数据模型中的高级应用

在TP模型中除了提供了常用的数据表增、删、改、查之外，还提供了很多非常方便的功能，能够帮助开发人员省去不少的开发时间和代码量，本节将会介绍其中常用了六个功能，分别是模型中的聚合方法、获取器、修改器、自动时间戳、模型的软删除、模型的分层设计，下面将详细介绍这八个功能。

1、模型中的聚合方法： 所谓聚合，就是和SQL语句中的聚合具有相同的概念，只是在TP中提供了更为便捷的五个函数用于完成数据库的聚合方法，具体代码如下：

表8-6 模型中的聚合方法

|  |  |
| --- | --- |
| 方法名 | 说明 |
| count() | 统计查询后的数据项的个数 |
| max("字段名") | 获取最大值，参数为要进行判断最大值的字段名 |
| min("字段名") | 获取最小值，参数为要进行判断最小值的字段名 |
| avg("字段名") | 获取平均值，参数为要进行求和取平均的字段名 |
| sum("字段名") | 获取求和值，参数为要进行求和的字段名 |

通过SQL语句修改原先的“think\_person”数据表，在数据表的“sex”字段之后添加“score”字段，用于保存学生的分数，然后就可以通过聚合函数来计算分数的平均值、最高分、最低分等，具体SQL代码和PHP代码如下：

ALTER TABLE `think\_test`.`think\_person`

ADD COLUMN `socre` VARCHAR(45) NOT NULL AFTER `sex`;

$person = new Person();

// 获取查询后数据项的个数

return $person->where('id > 0')->cout();

// 获取查询后分数的最高值

return $person->where('id > 0')->max(“score”);

// 获取查询后分数的最低分

return $person->where('id > 0')->min(“score”);

// 获取查询后分数的平均值

return $person->where('id > 0')->avg(“score”);

// 获取查询后分数的求和

return $person->where('id > 0')->sum(“score”);

2、获取器： 获取器的目的在于通过模型获取数据时，TP可以对这些数据进行自动的处理，例如在数据库中表示性别通常是使用0或1来表达，但是这样的数据不利于数据的显示，因此可以利用TP中的获取器把这些简单的状态数据转化为能够识别的数据。获取器的使用非常简单，只需要在模型内部创建一个“get属性名Attr($value)”的函数，并且该属性名采用驼峰命名法，就可以实现字符的转化，具体代码如下：

class Person extends Model

{

    protected $pk = 'id';

    // 添加一个获取器，把sex字段自动转化为男和女

    public function getSexAttr($value)

    {

        $sex = [0 => '女', 1 => '男'];

        return $sex[$value];

    }

}

当在Person模型中添加上述代码之后，再重新执行下面的代码，就可以得到如图8-13所示的结果，从结果可以看出其”sex“字段已经改为了”男“和”女“，而不再是”0“和”1“。

$person = new Person();

return $person->where('id > 0')->select();

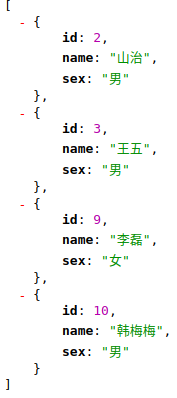


图8-13 获取器的使用

3、修改器： 修改器的目的在于通过模型设置数据时，TP可以对这些数据进行自动的处理，例如在向数据表中的字段”name“赋值时，可以把所有的字母都转化为小写，以便把数据转化为统一格式。修改器的使用非常简单，只需要在模型内部创建一个“set属性名Attr($value)”的函数，并且该属性名采用驼峰命名法，就可以实现字符的转化，具体代码如下：

class Person extends Model

{

    protected $pk = 'id';

    // 添加一个修改器，把name字段自动转化为小写

    public function setNameAttr($value)

    {

        return strtolower($value);

    }

}

当在Person模型中添加上述代码之后，再重新执行下面的代码，就可以在数据库中得到如图8-14所示的结果，从结果可以看出其”name“字段值已经改为了小写。

$person = new Person();

$person->name = 'SISO';

$person->sex = 1;

return $person->save();



图8-14 修改器的使用

4、自动时间戳：所谓自动时间戳，就是在向数据表中添加数据时，如果需要填入创建的时间或者修改的时间，那么TP就会自动向这些字段中填入相应的时间戳。要在模型中使用自动时间戳共分为两步走，分别如下：

第一步：要在模型中添加属性”protected $autoWriteTimestamp = true“，时间戳支持的数据类型有三种，分别是timestamp、datetime和int，默认情况下时间戳字段的数据类型为int，但如果在创建该字段时不是使用的int类型，而是datetime类型，那么就需要把$autoWriteTimestamp属性改为”protected $autoWriteTimestamp = 'datetime'”。

第二步：就是在数据表中设置自动时间戳的字段名称，分别是create\\_time和update\\_time，用于表示数据项的创建时间和修改时间。

上面的两个步骤都完成之后，当向数据表中添加数据时，TP就会自动向create\\_time字段添加当前时间，而当修改字段时，那么TP就会添加或者更新update\\_time中的时间，具体代码如下：

class Person extends Model

{

    protected $pk = 'id';

    // 添加自动时间戳，并且数据类型为datetime

    protected $autoWriteTimestamp = 'datetime';

}

添加往上面的代码之后，接下来修改Person数据表，并且在该数据表中添加两个datetime类型的字段，分别是create\\_time和update\\_time，具体的SQL代码如下：

ALTER TABLE `think\_test`.`think\_person`

ADD COLUMN `create\_time` DATETIME NULL AFTER `socre`,

ADD COLUMN `update\_time` DATETIME NULL AFTER `create\_time`;

当在Person数据表中添加了create\\_time和update\\_time字段之后，接下来重新执行下面的代码，就可以在数据库中得到如图8-15所示的结果，从结果可以看出其”create\\_time“和”update\\_time“字段值都已经被自动添加。

$person = new Person();

$person->name = '鸣人';

$person->sex = 1;

return $person->save();

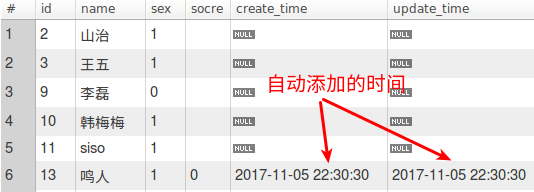


图8-15 自动时间戳的使用

5、模型的软删除：在实际项目中，如果经常对数据频繁使用删除操作会导致比较严重的性能问题，并且直接删除数据也不利于后续数据的恢复，因此通常的做法是在数据表字段中加入一个字段来表示该数据时候已经被删除，这种方法称为软删除。因此，所谓软删除其实质就是把数据项上加一个删除标记，而不是真正的删除。要在TP中使用软删除，也是要经过两个步骤，具体如下：

第一步：在模型中引入SoftDelete特性，即在模型中添加SoftDelete的头，并且在模型中使用SoftDelete。

第二步：在数据表中添加一个叫“delete\\_tiem”的字段，该字段用于存放删除的时间，并且模型情况下值为NUll。

经过上面的两个步骤就完成了模型软删除的添加，具体代码如下：

use think\Model;

// 添加软删除的引用

use traits\model\SoftDelete;

class Person extends Model

{

    use SoftDelete;

    protected $pk = 'id';

    protected $autoWriteTimestamp = 'datetime';

    // 绑定软删除在数据表中的字段名

    protected $deleteTime = 'delete\_time';

}

添加往上面的代码之后，接下来修改Person数据表，并且在该数据表中添加字段datetime类型的字段“delete\_time”，具体的SQL代码如下：

ALTER TABLE `think\_test`.`think\_person`

ADD COLUMN `delete\_time` DATETIME NULL AFTER `update\_time`;

当在Person数据表中添加了delete\\_time字段之后，接下来重新执行下面的代码，就可以在数据库中看到delete\\_time字段值都已经被自动添加。

$person = new Person();

return $person->where('id=3')->delete();

6、模型的分层设计：所谓模型分层，是为了更好的实现代码之间的解耦，实现模块功能的单一化，例如要把index模块中的模型分为三个部分，分别是数据层、逻辑层、服务层，那么就可以在index模块目录下创建model、logic和service三个文件夹，在这三个文件夹中再创建的继承于Model的模型类就分别对应上面的三个层次，这三个层次的功能如下：

（1）数据层：application\index\model\User.php 用于定义数据相关的自动验证和自动完成和数据存取接口

（2）逻辑层：application\index\logic\User.php 用于定义用户相关的业务逻辑

（3）服务层：application\index\service\User.php 用于定义用户相关的服务接口等

因为TP默认情况下只认model目录下的模型，因此需要通过Loader类下的model()函数来指定载入模型的位置，具体代码如下：

// 载入index\model目录下的User.php模型类

Loader::model('User');

// 载入index\logic目录下的User.php模型类

Loader::model('User','logic');

// 载入index\service目录下的User.php模型类

Loader::model('User','service');

从上面的Loader::model()函数可以看出该函数的第一个参数为模型类的类名，而第二个参数则表示的是模型类型所在的文件夹的名称。

8.6 ThinkPHP内置模板引擎与视图的使用

在进行Web开发时，对于PHP程序来说最希望的就是能够把精力专注于后台服务的开发和实现上，而至于前端的展示页面则有美工和前端工程师来完成，但是很多事情都往往事与愿违，因为PHP代码是一种内嵌于HTML，并执行在Web服务器端的脚本语言，所以Web服务器会把PHP代码解析为HTML代码后进行合并输出，但这样的做法就会造成PHP代码混合在HTML之间，虽然已经有很多的基于MVC架构的库出现解决PHP开发中各种强耦合的问题，但是对于Web程序的输入、输出和具有逻辑特性的页面呈现都会让PHP代码和HTML代码之间产生很强的耦合性。为了解决这个问题，Andrei Zmievski和Monte Orte两位前辈从1999年就开始研究和编写解决该问题的方法和思路，于是乎就有了目前使用最为普遍，并且被PHP官方推荐的PHP模板引擎——Smarty，PHP模板引擎的出现使得PHP彻底和HTML进行了分离，使得PHP开发人员只需要通过简单的PHP函数就可以设置HTML中的动态数据，实现真正的MVC架构。TP本身支持PHP的原生模板引擎Smarty，也支持TP自己开发模板引擎Think，因为目前TP自己开发的模板引擎Think能够满足大部分的需求，因此本节也将以该模板为基础讲解模板引擎和视图的使用。

8.6.1 视图初始化与模板中值的传递

在TP中，和模板引擎绑定最为紧密的就是控制器，也就是说通常要用于显示HTML的控制器就必须继承于父类Controller，再通过以下常用的两个函数来实现对HTML模板的操作，这些函数如表8-6所示。

表8-6 TP中模板函数

|  |  |
| --- | --- |
| 函数名 | 说明 |
| assign() | 模板变量的赋值 |
| fetch() | 模板的渲染并输出 |

其中assign()函数主要用于向HTML模板中的变量进行赋值，并且该函数有两种参数的重载，一种形式是assign($name, $value)，该形式的第一个参数表示HTML模板中的变量名，第二个参数则表示HTML模板中的变量值，而第二种形式则是assign($array)，该形式的参数表示的是一个以键值对形式保存的数组，当控制器调用第二种形式时，会解析参数数组，然后根据数组中的键值对的Key找到HTM模板中的变量，传入该Key值对应的值，具体代码如下：

namespace index\app\controller;

use think\Controller;

class Index extends Controller

{

    public function index()

    {

        // 模板变量赋值

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

        $this->assign('email','thinkphp@qq.com');

        // 或者批量赋值

        $this->assign([

            'name'  => 'ThinkPHP',

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

        ]);

    }

}

而fetch()函数则主要用于模板的渲染，即把动态的、包含模板变量等属性的HTML模板渲染成静态的HTML代码进行显示，fetch()函数除了具有模板渲染功能之外，还能够实现assign()的部分功能，即向模板中的变量传值，fetch()函数的原型如下：

fetch('[模板文件]'[, '模板变量（数组）']);

从原型可以看到fetch()函数实际上可以不带任何参数，如果只带一个参数，那么第一个参数表示指定模板的文件名，而如果带两个参数，那么第二个参数就是要传递到HTML模板的键值对数组。实际上，该函数使用最多的情况就是不带任何参数，此时fetch()函数会默认渲染“当前模块/view/当前控制器（小写）/当前操作（小写）.html”，例如在Index控制器下面的index()函数中调用fetch()，那么就会默认的渲染“index/view/index/index.html”。如果fetch()指定了模板名称，那么TP就会到“index/view/index”文件夹下找和传入的模板名称相同的HTML文件，如果此时fetch()函数还带了键值对数组，那么就会把键值对数组传入模板，替换模板中的变量，具体代码如下，结果如图8-16。

namespace index\app\controller;

use think\Controller;

class Index extends Controller

{

    public function index()

    {

        // 模板变量赋值

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

        // 不带任何参数 自动定位当前操作的模板文件

        return $this->fetch();

    }

}



图8-16 TP中变量的赋值和渲染

从上面的例子可以看到，模板中的变量都是通过大括号的方式，把模板变量包在其中，如{$变量名}。此外，在TP中除了可以向模板中传递单个变量值之外，还可以传递数组变量，当传递数组变量时，HTML模板的替换变量的形式为“{$数组名.数组项名}”，很多时候为了防止在页面渲染时，模板内的变量没有赋值引发的错误或者异常，TP模板提供了变量的默认值来解决这个问题，其形式为“{$变量名|default="默认值"}”，具体代码如下：

PHP代码：

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

$data['email'] = 'thinkphp@qq.com';

$view->assign('data',$data);

HTML模板代码：

<!-- 模板中数组变量的使用 -->

Name：{$data.name}

Email：{$data.email}

<!-- 模板中变量默认值的使用 -->

{$user.nickname|default="张三"}

除了通过TP向模板传递变量之外，在TP的模板中还可以调用PHP中的相关函数，这种函数的调用分为两种：

1、只有一个参数的函数调用。当在模板中要调用只有一个参数的函数时，采用的形式为{$变量名|函数名}，具体代码如下：

<!-- 模板中函数的使用 -->

{$data.name|md5}

2、有多个参数的函数调用。当在模板中要调用带有多个参数的函数时，采用的形式为{$变量名|函数名="变量名1", ###, ...}，即每个参数使用逗号作为分割，当函数的参数需要用到“|”左边的变量名时，则使用特殊符号“###”来表示，具体代码如下：

<!-- 模板中函数的多参数的使用 -->

{$create\_time|date="y-m-d",###}

8.6.2 模板中运算符和条件语句的使用

在Web开发中，经常会碰到不同账号显示的页面结果不同，以及页面显示的数据需要通过不同数据计算后才能得到，以及在列表显示时需要从数组数据中通过循环方式把每一项数据都取出来，对于这些场景的应用TP模板提供三类基本的功能，分别是模板中运算符的使用、模板中循环的使用、模板中条件判断的使用。

1、模板中运算符的使用：在TP模板中可以支持所有的简单运算符，以及三目运算符，具体如表8-7所示。

表8-7 模板中运算符

|  |  |
| --- | --- |
| 运算符 | 使用方法 |
| + | {$a+$b} |
| - | {$a-$b} |
| \* | {$a\*$b} |
| / | {$a/$b} |
| % | {$a%$b} |
| ++ | {$a++}或{++$a} |
| -- | {$a--}或{--$a} |
| 布尔类型表达式 ？ 表达式1 : 表达式2 | $status? '正常' : '错误' |

2、模板中循环的使用：在Web页面中，经常会出现表格、列表等一系列可以循环输出的内容，为了解决这个问题，在TP模板中提供两种循环的方式来解决该问题，第一种是使用关键字volist来实现数据集的查询，第二种则是通过for关键字来实现行为的循环执行。

1、关键字volist来实现数据集的查询：当在TP中向模板传递数组时，在HTML模板代码可以通过关键字volist来解析，并且volist关键字有两个参数，分别是name和id，其中name表示传递到模板的变量名，而id则表示变量名所对应的数组中的数组项，具体代码如下，效果如图8-17所示。

PHP代码：

$list = array (

    array('name' => '张三', 'tel' => '12345'),

    array('name' => '李四', 'tel' => '23456'),

    array('name' => '王五', 'tel' => '34567')

);

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

HTML模板代码：

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

    {$data.name}:{$data.tel}<br/>

{/volist}

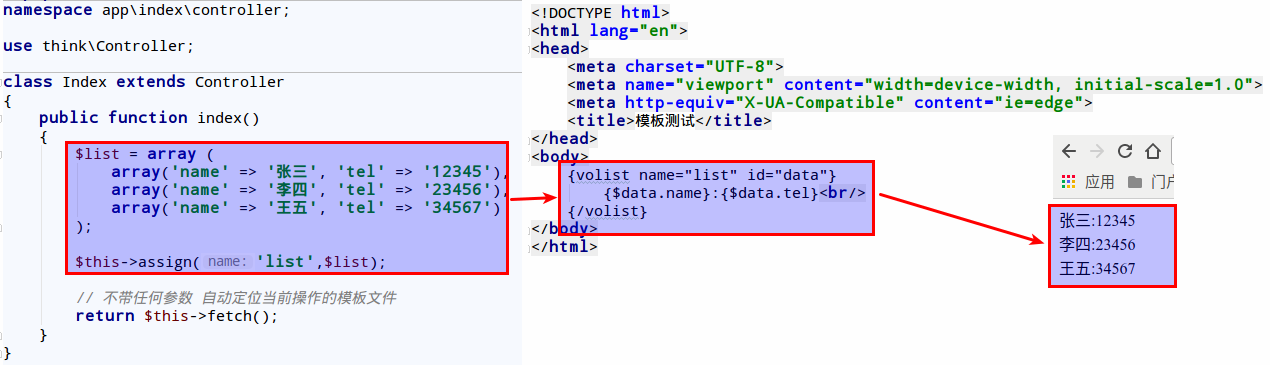


图8-17 volist的使用

2、关键字for来实现行为的循环执行：首先读者要知道的是for语句的使用场景和volist是有区别的，volist主要是用于获取数据集中的数据，而for循环则用于调节某个行为执行的次数，例如表格要显示几行，列表要显示几个等。for关键字有五个参数，其中开始值、结束值、步进值和循环变量都以变量的形式存在，并且开始值和结束值是必须，其他是可选。comparison表示循环判断的条件，默认值是lt，表示当循环变量名的值小于结束值时，循环继续执行，如果要判断大于的话，则comparison的值为gt，name的默认值是i，步进值的默认值是1，表示执行玩一次循环，循环变量名的值就加1，具体代码如下：

{for start="开始值" end="结束值" comparison="" step="步进值" name="循环变量名" }

{/for}

3、模板中条件判断的使用： 模板的条件判断分为六种，分别是SWITCH、IF、IN和NOTIN、BETWEEN和NOTBETWEEN。其中特别要说明的是IN和NOTIN表示的是时候属于一组数据列表中的一个，其value是多个值，而BETWEEN和NOTBETWEEN表示的是在某个范围之类，其value只有两个字，分别是这个范围的最小和最大值，具体代码如下：

{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}

{if condition="($name == 1) OR ($name > 100) "} value1

    {elseif condition="$name eq 2"/}value2

    {else /} value3

{/if}

{in name="id" value="1,2,3"}

    id在范围内

{/in}

{notin name="id" value="1,2,3"}

    id不在范围内

{/notin}

{between name="id" value="1,10"}

    输出内容1

{/between}

{notbetween name="id" value="1,10"}

    输出内容2

{/notbetween}

{between name="id" value="1,10"}

    输出内容1

    {else/}

    输出内容2

{/between}

8.6.3 模板的继承与使用

在Web开发中，很多时候一个页面会被重复使用，而只有其中的一个部分发生改动，在传统的开发中就需要为每个页面中重复编写相同的代码，这样的做法虽然可以实现最终页面的展示，但是对于页面修改和扩展却带来很多麻烦，因为开发人员需要把每个页面中重复的内容都进行修改，这样的做到既带来大量重复的工作，也使得这项工作非常容易出错，因此TP为了解决这个问题，提出了模板继承的概念。所谓模板继承，就是指开发人员定义一个基础模板，并且在其中通过关键字block定义需要在子模板中被重写的区域，然后当子模板通过关键字extend继承该基础模板时，开发人员只需要在子模板中对基础模板中的某些block区域进行重写即可。这样的做法的好处是，开发人员只需要把相同的部分编写一次，而在其他模板文件中只需要编写个性的东西，免去了相同部分重复编写的问题，并且当基础模板发生改变时也只需要修改一个文件就可以实现所有模板发生改动。

基础模板的编写和普通的HTML文件在大部分内容上没有任何区别，而区别在于需要在模板文件中定义多个由{block}{/block}标签组成的区块，具体代码如下：

<html>

<head>

    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">

    <title>模板的基层</title>

</head>

<body>

    <p>基础模板的内容</p>

    {block name="main"}基础模板的主内容{/block}

    <br />

    {block name="footer"}基础模板的底部内容{/block}

</body>

</html>

在上面的代码中block标签定义了需要在子模板中被重写的区域，每个block标签都必须指定一个name属性用来标识当前区块的名称，并且这个name属性值必须在当前模板中是唯一的。block标签的使用非常灵活，其中可以包含任何模板内容，包括变量、条件判断标签等。在定义完基础模板之后，就可以定义子模板（即控制器的视图），这里以Index模块下的Index控制器中的index函数的视图文件index.html为例，具体代码如下：

{extend name="base" /}

{block name="main"}

{$main}

{/block}

{block name="footer"}

{\_\_block\_\_}+{$footer}

{/block}

在上面的代码中首先看到使用extend关键字标签定义了子模板继承于基础模板“base.html”，这里需要注意的是如果没有特别指定，那么这里的基础模板“base.html”默认就位于Index模块下view目录下。在子模板中，可以通过重新定义和基础模板中具有相同name属性值的block标签来告诉TP模板渲染器对该部分要进行重新渲染，当然也可以在基础模板中不重新定义具有相同name属性值的block标签，那么此时TP就会使用子模板进行重新渲染，而是沿用基础模板中block标签所定义的内容，而如果定义了一个和基础模板相同的block标签，但是确在该标签中没有添加任何东西，那么就表示删除基础模板中的该区块内容。此外，在上面子模板中还有一个{\\_\\_block\\_\\_}标签，该标签的作用是引用原先基础模板中内容到{\\_\\_block\\_\\_}标签处，然后和子模板中该区域的其他内容进行合并，得到新的内容，结果如图8-18所示。

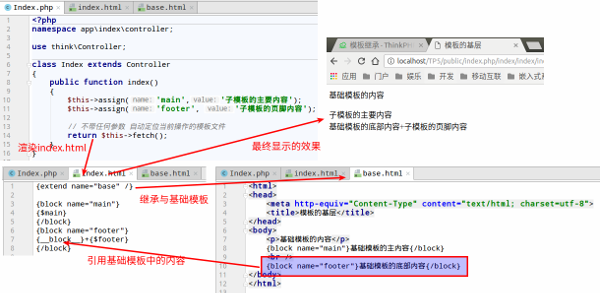


图8-18 模板基层的效果

这里还有一点需要注意的是，在当前子模板继承了基础模板，那么子模板中只能在block标签的定义区类编写相应代码，而不能除block标签之外的地方添加代码，如果添加了TP渲染时也会把这些代码直接忽略，即不显示在最终的页面中，示例代码如下：

{extend name="base" /}

{block name="main"}

{$main}

{/block}

{block name="footer"}

{\_\_block\_\_}+{$footer}

{/block}

<!-- 因为不在block标签内，因此这部分代码被忽略 -->

<a href="" >首页</a>

<a href="" >资讯</a>

<a href="" >论坛</a>

8.7 小结

通过本章的学习，了解了基于MVC架构的ThinkPHP的基本使用方法，并且深入分析了ThinkPHP中的控制器、模型、视图、路由等非常重要的概念。本章首先深入分析了MVC的一些基本特性，以及常用的体系结构，并且了解了MVC中各模块之间的相互作用，帮助开发人员对于ThinkPHP的体系结构有一个基本的认识，为后续的学习提供了理论基础。其次详细讲解了ThinkPHP的整个部署过程，以及ThinkPHP中各个文件夹的作用，通过这部分的学习可以让开人员能够了解在编写代码时的相应编写位置，同时体会ThinkPHP在设计时是如何与MVC的思想进行结合。随后使用大量篇幅解了ThinkPHP中非常重要的控制器，因为控制器是所有操作的路口，同时还讲了如何通过控制器接收各种HTTP请求，通过这部分的学习就可以让开发人员知道如何向ThinkPHP发送的请求，以及在ThinkPHP中如何解析这些请求。然后，本章还讲了ThinkPHP中最为核心，也是业务流程中最为核心的模型，以及数据库的ORM，通过这部分的讲解，开发人员可以掌握如果通过模型实现业务，并且通过分层的方式把业务分为不同的层次。最后讲了ThinkPHP中的内置模板引擎，通过模板引擎就可以实现HTML文件和控制器之间的绑定，并且通过模板的之间继承使得开发页面的效率大为提高。但是本章的内容实际上还有很多可以深入的方法，例如使用软件工程的概念设置业务模型、通过ThinkPHP提供的验证器来实现Web中各类信息的验证，以及如何在ThinkPHP中借助PHP的GD库实现验证码图片等，这些对于开发人员来说可以在后续的使用中逐步研究。