第七章 路由详解

7.1 路由优势和路由定义

7.1.1 路由优势

路由是应用开发中比较关键的一个环节，其主要作用包括但不限于：

- 让URL更规范以及优雅；
- 隐式传入额外请求参数；
- 统一拦截并进行权限检查等操作；
- 绑定请求数据；
- 使用请求缓存；
- 路由中间件支持；

路由解析的过程一般包含：

- 路由定义：完成路由规则的定义和参数设置；
- 路由检测：检查当前的URL请求是否有匹配的路由；
- 路由解析：解析当前路由实际对应的操作（方法或闭包）；
- 路由调度：执行路由解析的结果调度；

路由定义文件

路由规则的注册必须在应用的路由定义文件中完成。路由定义和检测是针对应用的，因此如果你采用的是多应用模式，每个应用的路由都是完全独立的，并且路由地址不能跨应用（除非采用重定向路由）。

注意：`route`目录下的任何路由定义文件都是有效的，分开多个路由定义文件并没有实际的意义，纯粹出于管理方便而已。默认的路由定义文件是`route.php`，但你完全可以更改文件名，或者添加多个路由定义文件。

```
├─route                  路由定义目录
|   ├─route.php          路由定义
|   ├─api.php            路由定义
|   └─...                更多路由定义
```

如果你使用了多应用模式，那么路由定义文件需要增加应用子目录，类似于下面：

```
├─route                  路由定义目录
|   ├─index              index应用路由定义
|   | ├─route.php        路由定义文件
|   | └─...              更多路由定义
|   ├─admin              admin应用路由定义
|   | ├─route.php        路由定义文件
|   | └─...              更多路由定义
```

多应用模式下面，如果你开启了自动多应用，路由的规则是指在URL地址的应用名之后的部分，也就是说URL中的应用名是不能省略和改变的，例如你在`index`应用中定义了路由。

```
Route::rule('hello/:name', 'index/hello');
```

在没有开启自动多应用的情况下，URL地址是

```
http://serverName/index.php/hello/think
```

一旦你开启了自动多应用，那么实际的URL地址应该是

```
http://serverName/index.php/index/hello/think
```

关闭路由

如果你的某个应用不需要使用路由功能，那么可以在应用的`app.php`配置文件中设置：

```
// 关闭应用的路由功能
'with_route'    =>    false,
```

7.1.2 路由定义

要使用`Route`类注册路由必须首先在路由定义文件开头添加引用（后面不再重复说明）

```
use think\facade\Route;
```

注册路由

最基础的路由定义方法是：

Route::rule('路由表达式', '路由地址', '请求类型');

```
// 例如注册如下路由规则（假设为单应用模式）：
// 注册路由到News控制器的read操作
Route::rule('new/:id','News/read');

// 我们访问：
http://serverName/new/5
// 会自动路由到：
http://serverName/news/read/id/5
```

并且原来的访问地址会自动失效。
可以在rule方法中指定请求类型（不指定的话默认为任何请求类型有效），例如：

```
Route::rule('new/:id', 'News/update', 'POST');
```

7.2 路由表达式和路由地址

7.2.1 路由表达式

表达式通常包含静态规则和动态规则，以及两种规则的结合，例如下面都属于有效的规则表达式：

```
Route::rule('/', 'index'); // 首页访问路由
Route::rule('my', 'Member/myinfo'); // 静态地址路由
Route::rule('blog/:id', 'Blog/read'); // 静态地址和动态地址结合
Route::rule('new/:year/:month/:day', 'News/read'); // 静态地址和动态地址结合
Route::rule(':user/:blog_id', 'Blog/read'); // 全动态地址
```

注意：规则表达式的定义以`/`为参数分割符（无论你的`PATH_INFO`分隔符设置是什么，请确保在定义路由规则表达式的时候统一使用`/`进行URL参数分割，除非是使用组合变量的情况）。

每个参数中可以包括动态变量，例如`:变量`或者`<变量>`都表示动态变量（新版推荐使用第二种方式，更利于混合变量定义），并且会自动绑定到操作方法的对应参数。

7.2.2 路由地址

路由地址表示定义的路由表达式最终需要路由到的实际地址（或者响应对象）以及一些需要的额外参数，支持下面几种方式定义：

路由到控制器/操作

```
// 解析规则是从操作开始解析，然后解析控制器，例如：
// 路由到blog控制器
Route::get('blog/:id','Blog/read');
```

Blog类定义如下：

```
<?php
namespace app\index\controller;

class Blog
{
    public function read($id)
    {
        return 'read:' . $id;
    }
}
```

路由到类的方法

这种方式的路由可以支持执行任何类的方法，而不局限于执行控制器的操作方法。

路由地址的格式为（动态方法）

\完整类名@方法名 或者 \完整类名::方法名

```
Route::get('blog/:id','\app\index\service\Blog@read');
```

执行的是`\app\index\service\Blog`类的`read`方法。
也支持执行某个静态方法，例如：

```
Route::get('blog/:id','\app\index\service\Blog::read');
```

重定向路由

可以直接使用`redirect`方法注册一个重定向路由

```
Route::redirect('blog/:id', 'http://blog.thinkphp.cn/read/:id', 302);
```

路由到模板

支持路由直接渲染模板输出。

```
// 路由到模板文件
Route::view('hello/:name', 'index/hello');
```

表示该路由会渲染当前应用下面的view/index/hello.html模板文件输出。
模板文件中可以直接输出当前请求的param变量，如果需要增加额外的模板变量，可以使用：

```
Route::view('hello/:name', 'index/hello', ['city'=>'shanghai']);
```

在模板中可以输出`name`和`city`两个变量。

```
Hello,{$name}--{$city}!
```

路由到闭包

我们可以使用闭包的方式定义一些特殊需求的路由，而不需要执行控制器的操作方法了，例如：

```
Route::get('hello', function () {
    return 'hello,world!';
});
```

7.3 路由参数

7.3.1 路由参数

路由参数主要完成路由匹配检测以及后续行为。

路由参数可以在定义路由规则的时候直接传入（批量），推荐使用方法配置更加清晰。

参数	说明	方法名
ext URL	后缀检测，支持匹配多个后缀	ext
deny_ext	URL禁止后缀检测，支持匹配多个后缀	denyExt
https	检测是否https请求	https
domain	域名检测	domain
complete_match	是否完整匹配路由	completeMatch
model	绑定模型	model
cache	请求缓存	cache
ajax	Ajax检测	ajax
pjax	Pjax检测	pjax
json	JSON检测	json
validate	绑定验证器类进行数据验证	validate
append	追加额外的参数	append
middleware	注册路由中间件	middleware
filter	请求变量过滤	filter

```
// 用法举例
Route::get('new/:id', 'News/read')
    ->ext('html')
    ->https();  // 这些路由参数可以混合使用，只要有任何一条参数检查不通过，当前路由就不会生效，继续检测后面的路由规则。
```

如果你需要批量设置路由参数，也可以使用`option`方法。

```
Route::get('new/:id', 'News/read')
    ->option([
        'ext'   => 'html',
        'https' => true
    ]);
```

7.4 注解路由和路由分组

7.4.1 注解路由

ThinkPHP支持使用注解方式定义路由（也称为注解路由），如果需要使用注解路由需要安装额外的扩展：

```
composer require topthink/think-annotation
```

然后只需要直接在控制器类的方法注释中定义，例如：

```
<?php
namespace app\controller;

use think\annotation\Route;

class Index
{
    /**
     * @param  string $name 数据名称
     * @return mixed
     * @Route("hello/:name")
     */
    public function hello($name)
    {
        return 'hello,'.$name;
    }
}
```

`@Route("hello/:name")` 就是注解路由的内容，请务必注意注释的规范，不能在注解路由里面使用单引号，否则可能导致注解路由解析失败，可以利用IDE生成规范的注释。如果你使用 `PHPStorm` 的话，建议安装 `PHP Annotations` 插件：https://plugins.jetbrains.com/plugin/7320-php-annotations ，可以支持注解的自动完成。

然后就使用下面的URL地址访问：

```
http://tp5.com/hello/thinkphp
```

页面输出

```
hello,thinkphp
```

默认注册的路由规则是支持所有的请求，如果需要指定请求类型，可以在第二个参数中指定请求类型：

```
<?php
namespace app\controller;

use think\annotation\Route;
```

```
class Index
{
    /**
     * @param  string $name 数据名称
     * @return mixed
     * @Route("hello/:name", method="GET")
     */
    public function hello($name)
    {
        return 'hello,'.$name;
    }
}
```

如果有路由参数需要定义，可以直接在后面添加方法，例如：

```
<?php
namespace app\controller;

use think\annotation\Route;

class Index
{
    /**
     * @param string $name 数据名称
     * @Route('hello/:name', method="GET", https=1, ext="html")
     * @return mixed
     */
    public function hello($name)
    {
        return 'hello,'.$name;
    }
}
```

7.4.2 路由分组

路由分组功能允许把相同前缀的路由定义合并分组，这样可以简化路由定义，并且提高路由匹配的效率，不必每次都去遍历完整的路由规则（尤其是开启了路由延迟解析后性能更佳）。

使用`Route`类的`group`方法进行注册，给分组路由定义一些公用的路由设置参数，例如：

```
Route::group('blog', function () {
    Route::rule(':id', 'blog/read');
    Route::rule(':name', 'blog/read');
})->ext('html')->pattern(['id' => '\d+', 'name' => '\w+']);
```

7.5 快捷路由和路由标识

7.5.1 快捷路由

框架为不同的请求类型定义了快捷方法，包括。

类型	描述	快捷方法
GET	GET请求	get
POST	POST请求	post
PUT	PUT请求	put
DELETE	DELETE请求	delete
PATCH	PATCH请求	patch
*	任何请求类型	any

快捷注册方法的用法为：

Route::快捷方法名('路由表达式', '路由地址');

```
Route::get('new/<id>','News/read'); // 定义GET请求路由规则
Route::post('new/<id>','News/update'); // 定义POST请求路由规则
Route::put('new/:id','News/update'); // 定义PUT请求路由规则
Route::delete('new/:id','News/delete'); // 定义DELETE请求路由规则
Route::any('new/:id','News/read'); // 所有请求都支持的路由规则
```

注册多个路由规则后，系统会依次遍历注册过的满足请求类型的路由规则，一旦匹配到正确的路由规则后则开始执行最终的调度方法，后续规则就不再检测。

7.5.2 路由标识

如果你需要快速的根据路由生成URL地址，可以在定义路由的时候指定生成标识（但要确保唯一）。

例如：

```
// 注册路由到News控制器的read操作
Route::rule('new/:id','News/read')
    ->name('new_read');
```

生成路由地址的时候就可以使用

```
url('new_read', ['id' => 10]);
```

如果不定义路由标识的话，系统会默认使用路由地址作为路由标识，例如可以使用下面的方式生成

```
url('News/read', ['id' => 10]);
```

7.6 路由绑定和域名路由

7.6.1 路由绑定

可以使用路由绑定简化URL或者路由规则的定义，绑定支持如下方式：

绑定到控制器/操作

把当前的URL绑定到控制器/操作，最多支持绑定到操作级别，例如在路由定义文件中添加：

```
// 绑定当前的URL到 Blog控制器
Route::bind('blog');
// 绑定当前的URL到 Blog控制器的read操作
Route::bind('blog/read');
```

该方式针对路由到控制器/操作有效，假如我们绑定到了blog控制器，那么原来的访问URL从

```
http://serverName/blog/read/id/5
```

可以简化成

```
http://serverName/read/id/5
```

如果定义了路由

```
Route::get('blog/:id','blog/read');
```

那么访问URL就变成了

```
http://serverName/5
```

绑定到类

把当前的URL直接绑定到某个指定的类，例如：

```
// 绑定到类
Route::bind('\app\index\controller\Blog');
```

那么，我们接下来只需要通过

```
http://serverName/read/id/5
```

就可以直接访问 `\app\index\controller\Blog` 类的read方法。

7.6.2 域名路由

ThinkPHP支持完整域名、子域名和IP部署的路由和绑定功能，同时还可以起到简化URL的作用。
可以单独给域名设置路由规则，例如给blog子域名注册单独的路由规则：

```
Route::domain('blog', function () {
    // 动态注册域名的路由规则
    Route::rule('new/:id', 'news/read');
    Route::rule(':user', 'user/info');
});
```

一旦定义了域名路由，该域名的访问就只会读取域名路由定义的路由规则。闭包中可以使用路由的其它方法，包括路由分组，但不能再包含域名路由。

支持同时对多个域名设置相同的路由规则：

```
Route::domain(['blog', 'admin'], function () {
    // 动态注册域名的路由规则
    Route::rule('new/:id', 'news/read');
    Route::rule(':user', 'user/info');
});
```

如果你需要设置一个路由跨所有域名都可以生效，可以对分组路由或者某个路由使用 `crossDomainRule` 方法设置：

```
Route::group( function () {
    // 动态注册域名的路由规则
    Route::rule('new/:id', 'news/read');
    Route::rule(':user', 'user/info');
})->crossDomainRule();
```

7.7 URL生成

ThinkPHP支持路由URL地址的统一生成，并且支持所有的路由方式，以及完美解决了路由地址的反转解析，无需再为路由定义和变化而改变URL生成。

使用路由标识

对使用不同的路由地址方式，地址表达式的定义有所区别。参数单独通过第二个参数传入，假设我们定义了一个路由规则如下：

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

在没有指定路由标识的情况下，可以直接使用路由地址来生成URL地址：

```
Route::buildUrl('blog/read', ['id' => 5, 'name' => 'thinkphp']);
```

如果我们在注册路由的时候指定了路由标识

```
Route::rule('blog/:id','blog/read')->name('blog_read');
```

那么必须使用路由标识来生成URL地址

```
Route::buildUrl('blog_read', ['id' => 5, 'name' => 'thinkphp']);
```

url后缀

默认情况下，系统会自动读取`url_html_suffix`配置参数作为URL后缀（默认为html），如果我们设置了：

```
'url_html_suffix'   => 'shtml'
```

那么自动生成的URL地址变为：

```
/index.php/blog/5.shtml
```

如果我们设置了多个URL后缀支持

```
'url_html_suffix'   => 'html|shtml'
```

则会取第一个后缀来生成URL地址，所以自动生成的URL地址还是：

```
/index.php/blog/5.html
```

如果你希望指定URL后缀生成，则可以使用：

```
Route::buildUrl('blog/read', ['id'=>5])->suffix('shtml');
```

加上入口文件

有时候我们生成的URL地址可能需要加上`index.php`或者去掉`index.php`，大多数时候系统会自动判断，如果发现自动生成的地址有问题，可以使用下面的方法：

```
Route::buildUrl('index/blog/read', ['id'=>5])->root('/index.php');
```

助手函数

系统提供了一个`url`助手函数用于完成相同的功能，例如：

```
url('index/blog/read', ['id'=>5])
    ->suffix('html')
    ->domain(true)
    ->root('/index.php');
```

各位同学，以上就是路由详解的全部内容。