第四章 模型详解

4.1 模型定义

4.1.1 定义模型

定义一个模型类很简单，例如下面是一个`User`模型：

```
<?php
namespace app\model;

use think\Model;

class User extends Model
{
}
```

模型会自动对应数据表，模型类的命名规则是除去表前缀的数据表名称，采用驼峰法命名，并且首字母大写，例如：

```
模型名                              约定对应数据表（假设数据库的前缀定义是 think_）
User                                think_user
UserType                            think_user_type

注意：
    如果你的规则和上面的系统约定不符合，那么需要设置Model类的数据表名称属性，以确保能够找到对应的数据表。
    模型自动对应的数据表名称都是遵循小写+下划线规范，如果你的表名有大写的情况，必须通过设置模型的table属性。
```


4.1.2 模型设置

默认主键为`id`，如果你没有使用`id`作为主键名，需要在模型中设置属性：

```
<?php
namespace app\model;

use think\Model;

class User extends Model
{
    protected $pk = 'uid';
}
```

如果你想指定数据表甚至数据库连接的话，可以使用：

```php
<?php
namespace app\model;

use think\Model;

class User extends Model
{
    // 设置当前模型对应的完整数据表名称
    protected $table = 'think_user';

    // 设置当前模型的数据库连接
    protected $connection = 'db_config';
}
```

常用的模型设置属性包括（以下属性都不是必须设置）：

属性	描述
name	模型名（相当于不带数据表前后缀的表名，默认为当前模型类名）
table	数据表名（默认自动获取）
suffix	数据表后缀（默认为空）
pk	主键名（默认为id）
connection	数据库连接（默认读取数据库配置）
query	模型使用的查询类名称
field	模型允许写入的字段列表（数组）
schema	模型对应数据表字段及类型
type	模型需要自动转换的字段及类型
strict	是否严格区分字段大小写（默认为true）
disuse	数据表废弃字段（数组）

4.1.3 模型操作

在模型中除了可以调用数据库类的方法之外（换句话说，**数据库的所有查询构造器方法模型中都可以支持**），可以定义自己的方法，所以也可以把模型看成是数据库的增强版。

模型的操作方法无需和数据库查询一样调用必须首先调用`table`或者`name`方法，因为模型会按照规则自动匹配对应的数据表，例如：

```php
Db::name('user')->where('id','>',10)->select();
```

改成模型操作的话就变成

```php
User::where('id','>',10)->select();
```

模型操作和数据库操作的另外一个显著区别是模型支持包括获取器、修改器、自动时间写入在内的一系列自动化操作和事件，简化了数据的存取操作，但随之而来的是性能有所下降（其实并没下降，而是自动帮你处理了一些原本需要手动处理的操作），后面会逐步领略到模型的这些特色功能。

4.2 模型基本操作

4.2.1 模型数据查询

获取单个数据

```php
// 取出主键为1的数据
$user = User::find(1);
echo $user->name;

// 使用查询构造器查询满足条件的数据
$user = User::where('name', 'thinkphp')->find();
echo $user->name;
```

注意：模型使用find方法查询，如果数据不存在返回Null，否则返回当前模型的对象实例

获取多个数据

```php
// 根据主键获取多个数据
$list = User::select([1,2,3]);
// 对数据集进行遍历操作
foreach($list as $key=>$user){
    echo $user->name;
}
```

要更多的查询支持，一样可以使用查询构造器：

```php
// 使用查询构造器查询
$list = User::where('status', 1)->limit(3)->order('id', 'asc')->select();
foreach($list as $key=>$user){
    echo $user->name;
}
```


获取某个字段或者某个列的值

```php
// 获取某个用户的积分
User::where('id',10)->value('score');
// 获取某个列的所有值
User::where('status',1)->column('name');
// 以id为索引
User::where('status',1)->column('name','id');
```

注意：value和column方法返回的不再是一个模型对象实例，而是纯粹的值或者某个列的数组。

聚合查询

```php
User::count();
User::where('status','>',0)->count();
User::where('status',1)->avg('score');
User::max('score');
```

4.2.2 模型数据新增

可以直接静态调用`create`方法创建并写入：

```php
$user = User::create([
    'name'  =>  'thinkphp',
    'email' =>  'thinkphp@qq.com'
]);
echo $user->name;
echo $user->email;
```

```php
echo $user->id; // 获取自增ID

// 批量增加数据
$user = new User;
$list = [
    ['name'=>'thinkphp','email'=>'thinkphp@qq.com'],
    ['name'=>'onethink','email'=>'onethink@qq.com']
];
$user->saveAll($list);
// saveAll方法新增数据返回的是包含新增模型（带自增ID）
```

4.2.3 模型数据更新

使用模型的静态`update`方法更新：

```php
User::update(['name' => 'thinkphp'], ['id' => 1]);

注意：模型的update方法返回模型的对象实例

// 如果你的第一个参数中包含主键数据，可以无需传入第二个参数（更新条件）
User::update(['name' => 'thinkphp', 'id' => 1]);

// 如果你需要只允许更新指定字段，可以使用
User::update(['name' => 'thinkphp', 'email' => 'thinkphp@qq.com'], ['id' => 1],
['name']);
```


4.2.4 模型数据删除

删除模型数据，可以在查询后调用`delete`方法。

```php
$user = User::find(1);
$user->delete();  // delete方法返回布尔值

// 条件删除
User::where('id','>',10)->delete();
```

4.3 JSON 字段

模型提供了很便捷的方式操作JSON字段

4.3.1 写入 JSON 数据

```php
// 在模型添加 $json属性
<?php
namespace app\model;

use think\Model;
class User extends Model
{
    // 设置json类型字段
    protected $json = ['info'];
}

// 以数组形式写入json数据
$user = new User;
$user->name = 'thinkphp';
```

```
$user->info = [
    'email'    => 'thinkphp@qq.com',
    'nickname '=> '流年',
];
$user->save();
```

4.3.2 查询 JSON 数据

```
$user = User::find(1);
echo $user->name; // thinkphp
echo $user->info->email; // thinkphp@qq.com
echo $user->info->nickname; // 流年

// 查询条件为JSON数据
$user = User::where('info->nickname','流年')->find();
echo $user->name; // thinkphp
echo $user->info->email; // thinkphp@qq.com
echo $user->info->nickname; // 流年
```

4.3.3 更新 JSON 数据

```
$user = User::find(1);
$user->name = 'kancloud';
$user->info->email = 'kancloud@qq.com';
$user->info->nickname = 'kancloud';
$user->save();
```


4.4 获取器与修改器

4.4.1 获取器

获取器的作用是对模型实例的（原始）数据做出自动处理。一个获取器对应模型的一个特殊方法（该方法必须为 `public` 类型），方法命名规范为： get `FieldName` Attr

```
FieldName为数据表字段的驼峰转换，定义了获取器之后会在下列情况自动触发：

模型的数据对象取值操作（$model->field_name）；
模型的序列化输出操作（$model->toArray()及toJson()）；
显式调用getAttr方法（$this->getAttr('field_name')）；

获取器的场景包括：

时间日期字段的格式化输出；
集合或枚举类型的输出；
数字状态字段的输出；
组合字段的输出；

// 我们需要对状态值进行转换，可以使用：
<?php
namespace app\model;

use think\Model;

class User extends Model
{
    public function getStatusAttr($value)
```

```
    {
        $status = [-1=>'删除',0=>'禁用',1=>'正常',2=>'待审核'];
        return $status[$value];
    }
}

// 数据表的字段会自动转换为驼峰法，一般status字段的值采用数值类型，我们可以通过获取器定义，自动转换为字符串描述。
$user = User::find(1);
echo $user->status; // 例如输出“正常”


// 动态获取
User::withAttr('name', function($value, $data) {
    return strtolower($value);
})->select();
```

4.4.2 修改器

和获取器相反，修改器的主要作用是对模型设置的数据对象值进行处理。

```
// 修改器方法的命名规范为： setFieldNameAttr
修改器的使用场景和读取器类似：

时间日期字段的转换写入；
集合或枚举类型的写入；
数字状态字段的写入；
某个字段涉及其它字段的条件或者组合写入；
定义了修改器之后会在下列情况下触发：

模型对象赋值；
调用模型的data方法，并且第二个参数传入true；
调用模型的save方法，并且传入数据；
显式调用模型的setAttr方法；

<?php
namespace app\model;

use think\Model;

class User extends Model
{
    public function setNameAttr($value)
    {
        return strtolower($value);
    }
}

// 如下代码实际保存到数据库中的时候会转为小写
$user = new User();
$user->name = 'THINKPHP';
$user->save();
echo $user->name; // thinkphp
```


4.5 自动时间戳

系统支持自动写入创建和更新的时间戳字段（默认关闭），有两种方式配置支持。

第一种方式是全局开启，在数据库配置文件中进行设置：

```
// 开启自动写入时间戳字段
'auto_timestamp' => true,
```

第二种是在需要的模型类里面单独开启：

```
<?php
namespace app\model;

use think\Model;

class User extends Model
{
    protected $autoWriteTimestamp = true;
}
```

一旦配置开启的话，会自动写入create_time和update_time两个字段的值，默认为整型（int），如果你的时间字段不是int类型的话，可以直接使用：

```
<?php
namespace app\model;

use think\Model;

class User extends Model
{
    protected $autoWriteTimestamp = 'datetime';
}
```

默认的创建时间字段为create_time，更新时间字段为update_time，支持的字段类型包括timestamp/datetime/int。

```
$user = new User();
$user->name = 'thinkphp';
$user->save();
echo $user->create_time; // 输出类似 2016-10-12 14:20:10
echo $user->update_time; // 输出类似 2016-10-12 14:20:10
```


4.6 软删除和类型转换

4.6.1 软删除

在实际项目中，对数据频繁使用删除操作会导致性能问题，软删除的作用就是把数据加上删除标记，而不是真正的删除，同时也便于需要的时候进行数据的恢复。

要使用软删除功能，需要引入`SoftDelete` trait，例如`User`模型按照下面的定义就可以使用软删除功能：

```php
<?php
namespace app\model;

use think\Model;
use think\model\concern\SoftDelete;

class User extends Model
{
    use SoftDelete;
    protected $deleteTime = 'delete_time';
}
```

备注：deleteTime属性用于定义你的软删除标记字段，ThinkPHP的软删除功能使用时间戳类型（数据表默认值为Null），用于记录数据的删除时间。

定义好模型后，我们就可以使用：

```php
$user = User::find(1);
// 软删除
$user->delete();
// 真实删除
$user->force()->delete();
```

默认情况下查询的数据不包含软删除数据，如果需要包含软删除的数据，可以使用下面的方式查询：

```php
User::withTrashed()->find();
User::withTrashed()->select();
```

如果仅仅需要查询软删除的数据，可以使用：

```php
User::onlyTrashed()->find();
User::onlyTrashed()->select();
```

恢复被软删除的数据

```php
$user = User::onlyTrashed()->find(1);
$user->restore();
```

4.6.2 类型转换

支持给字段设置类型自动转换，会在写入和读取的时候自动进行类型转换处理，例如：

```php
<?php
namespace app\model;

use think\Model;

class User extends Model
{
    protected $type = [
        'status'    =>  'integer',
        'score'     =>  'float',
        'birthday'  =>  'datetime',
```

```
        'info'      =>  'array',
    ];
}
```

下面是一个类型自动转换的示例：

```
$user = new User;
$user->status = '1';
$user->score = '90.50';
$user->birthday = '2015/5/1';
$user->info = ['a'=>1,'b'=>2];
$user->save();
var_dump($user->status); // int 1
var_dump($user->score); // float 90.5;
var_dump($user->birthday); // string '2015-05-01 00:00:00'
var_dump($user->info);// array (size=2) 'a' => int 1  'b' => int 2
```

数据库查询默认取出来的数据都是字符串类型，如果需要转换为其他的类型，需要设置，支持的类型包括如下类型：

类型	描述
integer	设置为integer（整型）后，该字段写入和输出的时候都会自动转换为整型。
float	该字段的值写入和输出的时候自动转换为浮点型。
boolean	该字段的值写入和输出的时候自动转换为布尔型。
array	如果设置为强制转换为array类型，系统会自动把数组编码为json格式字符串写入数据库，取出来的时候会自动解码。
object	该字段的值在写入的时候会自动编码为json字符串，输出的时候会自动转换为stdclass对象。
serialize	指定为序列化类型的话，数据会自动序列化写入，并且在读取的时候自动反序列化。
json	指定为json类型的话，数据会自动json_encode写入，并且在读取的时候自动json_decode处理。

4.7 关联模型

通过模型关联操作把数据表的关联关系对象化，解决了大部分常用的关联场景，封装的关联操作比起常规的数据库联表操作更加智能和高效，并且直观。

4.7.1 一对一关联

定义一对一关联，例如，一个用户都有一个个人资料，我们定义`User`模型如下：

```
<?php
namespace app\model;

use think\Model;

class User extends Model
{
    public function profile()
    {
        return $this->hasOne(Profile::class);
    }
}
```

`hasOne`方法的参数包括：hasOne('关联模型类名', '外键', '主键');

除了关联模型外，其它参数都是可选。

- **关联模型**（必须）：关联模型类名
- **外键**：默认的外键规则是当前模型名（不含命名空间，下同）+`_id`，例如`user_id`
- **主键**：当前模型主键，默认会自动获取也可以指定传入

关联查询

```
// 定义好关联之后，就可以使用下面的方法获取关联数据：
$user = User::find(1);
// 输出Profile关联模型的email属性
echo $user->profile->email;

// 根据关联条件来查询当前模型对象数据
// 查询用户昵称是think的用户
// 注意第一个参数是关联方法名（不是关联模型名）
$users = User::hasWhere('profile', ['nickname'=>'think'])->select();

// 可以使用闭包查询
$users = User::hasWhere('profile', function($query) {
    $query->where('nickname', 'like', 'think%');
})->select();
```

关联自动写入

```
// 我们可以使用together方法更方便的进行关联自动写入操作。
$blog = new Blog;
$blog->name = 'thinkphp';
$blog->title = 'ThinkPHP5关联实例';
$content = new Content;
$content->data = '实例内容';
$blog->content = $content;
$blog->together(['content'])->save();
```

关联更新

```
// 查询
$blog = Blog::find(1);
$blog->title = '更改标题';
$blog->content->data = '更新内容';
// 更新当前模型及关联模型
$blog->together(['content'])->save();
```

关联删除

```
// 查询
$blog = Blog::find(1,'content');
// 删除当前及关联模型
$blog->together(['content'])->delete();
```

4.7.2 一对多关联

关联定义

一对多关联的情况也比较常见，使用`hasMany`方法定义，参数包括：

hasMany('关联模型','外键','主键');

除了关联模型外，其它参数都是可选。

- **关联模型**（必须）：关联模型类名
- **外键**：关联模型外键，默认的外键名规则是当前模型名+`_id`
- **主键**：当前模型主键，一般会自动获取也可以指定传入

```
// 一篇文章可以有多个评论
<?php
namespace app\model;

use think\Model;

class Article extends Model
{
    public function comments()
    {
        return $this->hasMany(Comment::class);
    }
}
```

关联查询

```
$article = Article::find(1);
// 获取文章的所有评论
dump($article->comments);
// 也可以进行条件搜索
dump($article->comments()->where('status',1)->select());

// 可以根据关联条件来查询当前模型对象数据，例如：
// 查询评论超过3个的文章
$list = Article::has('comments','>',3)->select();
// 查询评论状态正常的文章
$list = Article::hasWhere('comments',['status'=>1])->select();
```

关联新增

```
$article = Article::find(1);
// 增加一个关联数据
$article->comments()->save(['content'=>'test']);
// 批量增加关联数据
$article->comments()->saveAll([
    ['content'=>'thinkphp'],
    ['content'=>'onethink'],
]);
```

关联删除

```
// 在删除文章的同时删除下面的评论
$article = Article::with('comments')->find(1);
$article->together(['comments'])->delete();
```

六星教育

sixstaredu.com