



ThinkPHP Framework 1.0

Relation Operation

ThinkPHP 1.0

关联操作

编写：ThinkPHP 文档组

最后更新：2008-07-31

目录

1	概述.....	3
2	关联定义.....	3
3	关联查询.....	9
4	关联操作.....	12
5	手动关联.....	14

1 概述

本文描述了 ThinkPHP 的关联操作的定义和使用。

2 关联定义

ThinkPHP 支持数据表的关联操作，目前支持的关联关系包括下面三种：

- 1、ONE_TO_ONE (包括 HAS_ONE 和 BELONGS_TO)
- 2、ONE_TO_MANY(包括 HAS_MANY 和 BELONGS_TO)
- 3、MANY_TO_MANY

一个模型根据业务模型可以定义多个关联，所有的关联定义都统一在 `$_link` 成员变量里面定义。下面我们来分析下各个关联方式的定义：

HAS_ONE

HAS_ONE 关联表示当前模型拥有一个子对象，例如，每个员工都有一个人事档案。我们可以建立一个用户模型 `UserModel`，并且添加如下关联定义：

```
class UserModel extends Model
{
    protected $_link = array(
        array('mapping_type'=>HAS_ONE,
            'class_name'    =>'Profile',
            'mapping_name'=>'profile',
            ),
    );
}
```

关联 HAS_ONE 定义支持的关联属性有：

mapping_type 关联类型，这个在 HAS_ONE 关联里面必须使用 HAS_ONE 常量定义。

class_name 要关联的模型类名

例如，class_name 定义为 Profile 的话则表示存在一个另外的模型类，是首先需要定义的，例如：

```
class ProfileModel extends Model{}
```

mapping_name 关联的映射名称，用于获取数据用

该名称不要和当前模型的字段有重复，否则会导致关联数据获取的冲突。

foreign_key 关联的外键名称

外键的默认规则是当前数据对象名称_id，例如：

UserModel 对应的可能是表 think_user （注意：think 只是一个表前缀，可以随意配置）

那么 think_user 表的外键默认为 user_id 如果不是 就必须在定义关联的时候显式定义 foreign_key 。

condition 关联条件

关联查询的时候会自动带上外键的值，如果有额外的查询条件，可以通过定义关联的 condition 属性。

mapping_fields 关联要查询的字段

默认情况下，关联查询的关联数据是关联表的全部字段，如果只是需要查询个别字段，可以定义关联的 mapping_fields 属性。

as_fields 直接把关联的字段值映射成数据对象中的某个字段

这个特性是 ONE_TO_ONE 关联特有的，可以直接吧关联数据映射到数据对象中，而不是作为一个关联数据。

在 1.0.5beta 版本之前，属性 mapping_type 和 class_name 是必须定义的，如果 mapping_name 没有定义的话，会取 class_name 的定义作为 mapping_name。

在 1.0.5beta 版本以后，还支持下面的几种定义：

```
class UserModel extends Model
```

```
{  
    protected $_link = array(  
        'profile'=>array(  
            'mapping_type' =>HAS_ONE,  
            'class_name'=>'Profile',  
        ),  
    );  
}
```

又或者更加简化成：

```
class UserModel extends Model  
{  
    protected $_link = array(  
        'Profile'=> HAS_ONE,  
    );  
}
```

这样，就等于 class_name 和 mapping_name 都是 Profile。

我们看到上面的定义并没有定义关联的外键名称，其实是采用了默认规则。

例如，下面的例子我们定义了外键是 userId，而不再是默认的 user_id。

```
class UserModel extends Model  
{  
    protected $_link = array(  
        'profile'=>array(  
            'mapping_type' =>HAS_ONE,  
            'class_name'=>'Profile',  
            'foreign_key'=>'userId',  
        ),  
    );  
}
```

BELONGS_TO

Belongs_to 关联表示当前模型从属于另外一个父对象，例如每个用户都属于一个部门。我们可以做如下关联定义。

```
array( 'mapping_type'=>BELONGS_TO,  
       'class_name'=>'Dept',  
       'foreign_key'=>'userId',  
       'mapping_name'=>'dept'),
```

关联 HAS_ONE 定义支持的关联属性有：

mapping_name 关联的映射名称，用于获取数据用

该名称不要和当前模型的字段有重复，否则会导致关联数据获取的冲突。

foreign_key 关联的外键名称

mapping_fields 关联要查询的字段

condition 关联条件

parent_key 自引用关联的关联字段

默认为 parent_id

自引用关联是一种比较特殊的关联，也就是关联表就是当前表。

as_fields 直接把关联的字段值映射成数据对象中的某个字段

HAS_MANY

HAS_MANY 关联表示当前模型拥有多个子对象，例如每个用户有多篇文章，我们可以这样来定义：

```
array( 'mapping_type'=>HAS_MANY,  
       'class_name'=>'Article',  
       'foreign_key'=>'userId',
```

```
'mapping_name'=>'articles',  
'mapping_order'=>'cTime desc'),
```

关联 HAS_MANY 定义支持的关联属性有：

mapping_name 关联的映射名称，用于获取数据用

该名称不要和当前模型的字段有重复，否则会导致关联数据获取的冲突。

foreign_key 关联的外键名称

外键的默认规则是当前数据对象名称_id，例如：

UserModel 对应的可能是表 think_user （注意：think 只是一个表前缀，可以随意配置）

那么 think_user 表的外键默认为 user_id，如果不是，就必须在定义关联的时候定义 foreign_key 。

parent_key 自引用关联的关联字段

默认为 parent_id

condition 关联条件

关联查询的时候会自动带上外键的值，如果有额外的查询条件，可以通过定义关联的 condition 属性。

mapping_fields 关联要查询的字段

默认情况下，关联查询的关联数据是关联表的全部字段，如果只是需要查询个别字段，可以定义关联的 mapping_fields 属性。

mapping_limit 关联要返回的记录数目

mapping_order 关联查询的排序

MANY_TO_MANY

HAS_MANY 关联表示当前模型可以属于多个对象，而父对象则可能包含有多个子对象，通常两者之间

需要一个中间表类约束和关联。例如每个用户可以属于多个组，每个组可以有多个用户：

```
array( 'mapping_type'=>MANY_TO_MANY,
```

```
'class_name'=>'Group',  
'mapping_name'=>'groups',  
'foreign_key'=>'userId',  
'relation_foreign_key'=>'goupId',  
'relation_table'=>'think_gourpUser')
```

mapping_name 关联的映射名称，用于获取数据用

该名称不要和当前模型的字段有重复，否则会导致关联数据获取的冲突。

foreign_key 关联的外键名称

外键的默认规则是当前数据对象名称_id，例如：

UserModel 对应的可能是表 think_user （注意：think 只是一个表前缀，可以随意配置）

那么 think_user 表的外键默认为 user_id，如果不是，就必须在定义关联的时候定义 foreign_key。

relation_foreign_key 关联表的外键名称

默认的关联表的外键名称是表名_id

mapping_limit 关联要返回的记录数目

mapping_order 关联查询的排序

relation_table 关联的表名称

多对多的中间表默认表规则如下：

如果 think_user 和 think_group 存在一个对应的中间表，默认的表名应该是

如果是由 group 来操作关联表，中间表应该是 think_group_user，如果是从 user 表来操作，那么应该是 think_user_group，也就是说，多对多关联的设置，必须有一个 Model 类里面需要显式定义中间表，否则双向操作会出错。

中间表无需另外的 id 主键（但是这并不影响中间表的操作），通常只是由 user_id 和 group_id 构成。

默认会通过当前模型的 getRelationTableName 方法来自动获取，如果当前模型是 User，关联模型是

Group，那么关联表的名称也就是使用 user_group 这样的格式，如果不是默认规则，需要指定 relation_table 属性。

3 关联查询

定义之后，我们如何使用关联查询呢？

关联查询有自动关联查询和手动关联查询两种用法，先看自动关联查询，对于模型的关联关系不复杂的情况下可以直接使用。

```
$User = D("User");  
$user = $User->xfind(1);
```

和普通的 find 方法不同，区别只是多了一个 x，但是查询的结果却不同，因为查询结果已经自动加上了关联数据。

输出\$user 结果可能是类似于下面的数据：

```
array(  
  'id'      => 1,  
  'account' => 'ThinkPHP',  
  'password' => 'name',  
  'Profile' => array(  
    'email'      => 'liu21st@gmail.com',  
    'nickname'   => '流年',  
  ),  
)
```

我们可以看到，用户的关联数据已经被映射到数据对象的属性里面了。其中 Profile 就是关联定义的 mapping_name 属性。

如果我们按照下面的凡事定义了 as_fields 属性的话，

```
protected $_link = array(  
    'profile'=>array(  
        'mapping_type' =>HAS_ONE,  
        'class_name' =>'Profile',  
        'foreign_key'=>'userId',  
        'as_fields'=>'email,nickname',  
    ),  
);
```

查询的结果就变成了下面的结果

```
array(  
    'id' => 1,  
    'account' => 'ThinkPHP',  
    'password' => 'name',  
    'email' => 'liu21st@gmail.com',  
    'nickname' => '流年',  
)
```

email 和 nickname 两个字段已经作为 user 数据对象的字段来显示了。

如果关联数据的字段名和当前数据对象的字段有冲突的话，怎么解决呢？

我们可以用下面的方式来变化下定义：

```
'as_fields'=>'email,nickname:username',
```

表示关联表的 nickname 字段映射成当前数据对象的 username 字段。

除了使用 xfind 方法外，我们还可以使用连贯方法来控制是否需要获取关联数据。

```
$User = D("User");  
$user = $User->relation(true)->find(1);
```

上面的写法和使用 xfind 方法是等效的。

还有一种方法是直接在模型里面定义是否需要自动获取关联数据。

```
class UserModel extends Model
{
    protected $autoReadRelations = TRUE;
}
```

这样定义了 autoReadRelations 属性之后，即使你用 find 查询，系统也会自动查询关联数据。

自动关联查询会把所有定义的关联数据都查询出来，有时候我们并不希望这样，就可以采用手动关联查询的方式。例如：

```
$User = D("User");
$user = $User->find(1);
$user = $User->getRelation($user, 'Profile');
```

这样就只会获取关联名称为 Profile 的关联数据。如果使用：

```
$user = $User->getRelation($user);
```

这样就会获取 User 模型查询结果的所有关联数据。

注意：

1.0.5beta 版本之前，getRelation 方法的第二个参数必须是数组，包含

\$relation['type'] 和 \$relation['name']

1.0.5beta 版本之后，getRelation 方法的第二个参数是个字符串，也就代表了之前的 relation['name']，

而\$relation['type']是自动定位的。

关联查询一样可以支持 findAll 方法，如果要查询多个数据，并同时获取相应的关联数据，可以改成：

```
$User = D("User");
$list = $User->xFindAll();
```

或者

```
$User = D("User");
```

```
$list = $User->findAll();  
$list = $User->getRelations($user, 'Profile');
```

注意这里使用的是 `getRelations` 而不是 `getRelation` 方法了。

关联查询在定义和操作方面都有点复杂，官方建议对于一对一关联采用视图模型进行查询操作，而且定义也比较直观。

4 关联操作

除了关联查询外，系统也支持关联数据的自动写入、更新和删除

关联写入

```
$User = D("User");  
$data = array();  
$data["account"] = "ThinkPHP";  
$data["password"] = "123456";  
$data["Profile"] = array(  
    'email' => 'liu21st@gmail.com',  
    'nickname' => '流年',  
);  
$result = $User->add($data, true);
```

Add 方法的第二个参数就表示是否需要自动插入关联数据，否则的话 Profile 对象是不会自动插入数据库里面。如果设置了模型的 `autoAddRelations` 属性为 `True`，也可以自动关联插入。

```
class UserModel extends Model  
{  
    protected $autoAddRelations = TRUE;  
}
```

定义之后，使用

```
$result = $User->add($user);
```

仍然会自动写入关联的 Profile 数据。

关联更新

数据的关联更新和关联写入类似

```
$User = D("User");  
$data = array();  
$data["account"] = "ThinkPHP";  
$data["password"] = "123456";  
$data["Profile"] = array(  
    'email'      => 'liu21st@gmail.com',  
    'nickname'   => '流年',  
);  
$result = $User->save($data, "id=3", true);
```

save 方法的第三个参数就表示是否需要自动更新关联数据，否则的话 Profile 对象是不会自动更新数据的。如果设置了模型的 autoSaveRelations 属性为 True，也可以自动关联更新。

```
class UserModel extends Model  
{  
    protected $autoSaveRelations = TRUE;  
}
```

定义之后，使用

```
$result = $User->save($data, "id=3");
```

仍然会自动更新关联的 Profile 数据。

关联删除

支持关联删除的方法有

```
deleteById($id,$autoLink=false)
```

```
deleteByIds($ids,$limit="",$order="",$autoLink=false)
```

```
deleteBy($field,$value,$limit="",$order="",$autoLink=false)
```

```
delete($data=null,$limit="",$order="",$autoLink=false)
```

```
deleteAll($condition="",$autoLink=false)
```

例如：

删除用户 ID 为 3 的记录的同时删除关联数据

```
$result = $User->deleteById("3",true);
```

如果设置了模型的 `autoDelRelations` 属性为 `True`，无需使用 `autoLink` 参数也可以自动关联删除。

```
class UserModel extends Model
{
    protected $autoDelRelations = TRUE;
}
```

定义之后，使用

```
$result = $User->deleteById("3");
```

一样可以自动删除关联数据。其他删除方法的用法和 `deleteById` 基本类似。

需要注意的是，如果 `User` 模型定义了多个关联数据，会全部删除，使用自动完成的关联删除无法删除

其中一个关联数据。

5 手动关联

除了系统自动完成的关联操作外，我们还可以手动进行关联操作。

关联查询

```
getRelation($data,$name)
```

relationGet(\$name)

例如：

```
$User = D("User");  
$user = $User->find(1);  
  
// 获取查询结果的关联 Profile 数据  
$profile = $User->getRelation($user, "Profile");  
  
// 1.0.5 以上还可以使用下面的（表示对当前查询的数据对象进行关联数据获取）  
$profile = $User->relationGet("Profile");
```

关联写入

relationAdd(\$data,\$name)

例如：

```
$User = D("User");  
$data = array();  
$data["account"] = "ThinkPHP";  
$data["password"] = "123456";  
$data["Profile"] = array(  
    'email' => 'liu21st@gmail.com',  
    'nickname' => '流年',  
);  
$result = $User->add($data);  
  
// 关联写入 Profile 数据  
$User->relationAdd($data, "Profile");
```

注意：relationAdd 方法本身并不存在，是动态方法，包括下面会涉及到的 relationSave 和 relationDel

都是同样的情况。系统利用了__call 魔术方法实现了关联操作的手动方法。

关联更新

relationSave(\$data,\$name)

例如：

```
$User = D("User");  
$User->find(1);  
$User->password    =    "123456";  
$User->save();  
  
// 关联更新 Profile 数据  
  
$User->Profile      =    array(  
    'email'          =>'liu21st@gmail.com',  
    'nickname'       =>'流年',  
);  
  
$User->relationSave("", "Profile");
```

关联删除

relationDel(\$data,\$name)

例如：

```
$User = D("User");  
$User->find(1);  
$User->delete();  
  
// 关联删除 Profile 数据  
  
$User->relationDel("", "Profile");
```

6 注意事项

✧ 关联操作必须建立对应的模型

- ✧ 关联操作效率并不高 有时候可以使用视图模型提高效率（参考视图模型文档）
- ✧ 当给模型定义了多个关联的时候，要尽量避免使用自动关联操作，以免影响性能