ThinkPHP Framework 1.0
View-Model

ThinkPHP 1.0
视图模型

编写：ThinkPHP 文档组

最后更新：2008-07-31

目录

1 概述

本文描述了 ThinkPHP 视图模型的定义和使用。

2 视图和视图模型

视图通常是指数据库的视图，视图是一个虚拟表，其内容由查询定义。同真实的表一样，视图包含一系列带有名称的列和行数据。但是，视图并不在数据库中以存储的数据值集形式存在。行和列数据来自由定义视图的查询所引用的表，并且在引用视图时动态生成。对其中所引用的基础表来说，视图的作用类似于筛选。定义视图的筛选可以来自当前或其它数据库的一个或多个表，或者其它视图。分布式查询也可用于定义使用多个异类源数据的视图。如果有几台不同的服务器分别存储组织中不同地区的数据，而您需要将这些服务器上相似结构的数据组合起来，这种方式就很有用。

视图在有些数据库下面并不被支持，但是 ThinkPHP 模拟实现了数据库的视图模型，该功能可以用于多表联合查询。非常适合解决 HAS_ONE 和 BELONGS_TO 类型的关联查询。

3 定义视图

要定义视图对象，需要设置模型的 viewModel 属性为 true，然后设置 viewFields 属性即可。例如下面的例子，我们定义了一个 BlogView 模型对象，其中包括了 Blog 模型的 id、name、title 和 User 模型的 name，以及 Category 模型的 title 字段，我们通过创建 BlogView 模型来快速读取一个包含了 User 名称和类别名称的 Blog 记录（集）。

```
class BlogViewModel extends Model
{
        protected $viewModel = true;
        protected $viewFields = array(
```

```
                'Category'=>array('title'=>'categoryName'),
                'User'=>array('name'=>'userName'),
                'Blog'=>array('id','name','title'),
                );
        protected $viewCondition = array(
        'Blog.categoryId'=>array('eqf','Category.id'),
        'Blog.userId'=>array('eqf','User.id'),
        );
}
```

我们来解释一下定义的格式代表了什么。

`$viewFields` 属性表示视图模型包含的字段，每个元素定义了某个数据表或者模型的字段。

例如：

```
'Blog'=>array('id','name','title')
```

表示 BlogView 视图模型要包含 Blog 模型中的 id、name 和 title 字段属性，这个其实很容易理解，就和数据库的视图要包含某个数据表的字段一样。而 Blog 相当于是给 Blog 模型对应的数据表定义了一个别名。

BlogView 视图模式除了包含 Blog 模型之外，还包含了 Category 和 User 模型，下面的定义：

```
'Category'=>array('title'=>'categoryName')
```

和上面类似，表示 BlogView 视图模型还要包含 Category 模型的 title 字段，因为视图模型里面已经存在了一个 title 字段，所以我们通过 `'title'=>'categoryName'` 把 Category 模型的 title 字段映射为 categoryName 字段，如果有多个字段，可以使用同样的方式添加。最后的 User 模型的定义方式同样也就很容易理解了。

`$viewCondition` 属性表示了视图的基础关联条件

例如：

```
'Blog.categoryId'=>array('eqf','Category.id'),
'Blog.userId'=>array('eqf','User.id'),
```

注意 eqf 指的是后面的条件不是字符串，而是 SQL 字段操作，这样定义后，在进行其它查询的时候，会自动带上这个基础条件。最终的解析结果其实是：

```
Blog.categoryId = Category.id AND Blog.userId = User.id
```

最后，我们把视图模型的定义翻译成 SQL 语句就更加容易理解视图模型的原理了。假设我们不带任何其他条件查询全部的字段，那么查询的 SQL 语句就是

Select
Blog.id as id,
Blog.name as name,
Blog.title as title,
Category.title as categoryName,
User.name as UserName
from think_blog Blog JOIN think_category Category JOIN think_user User
where Blog.categoryId=Category.id AND Blog.userId=User.id

视图模型的定义一般需要先单独定义其中的模型类，但是这并不是必须的，如果没有定义其中的模型类，系统会默认按照系统的规则进行数据表的定位。如果 Blog 模型并没有定义，那么系统会自动根据当前模型的表前缀和后缀来自动获取对应的数据表。也就是说，如果我们并没有定义 Blog 模型类，那么上面的定义后，系统在进行视图模型的操作的时候会根据 Blog 这个名称和当前的表前缀设置（假设为 Think_ ）获取到对应的数据表可能是 think_blog。不过系统建议为了操作方便，最好还是单独定义 Blog 模型，这样单表和视图都可以操作。

4 视图查询

接下来，我们就可以和使用普通模型一样对视图模型进行操作了 。

```
$Model = D("BlogView");
$Model->findAll();
$Model->field('id,name,title,categoryName,userName')->order('id desc')->findAll();
```

看起来和普通的模型操作并没有什么大的区别，可以使用普通模型的大多数查询方法。如果发现查询的结果存在重复数据，还可以使用 group 方法来处理。

```
$Model->field('id,name,title,categoryName,userName')->order('id desc')->group('id')->findAll();
```

我们可以看到，即使不定义视图模型，其实我们也可以通过方法来操作，但是显然非常繁琐。

```
$Model = D("Blog");
$Model->table(
'think_blog Blog,
think_category Category,
think_user User'
)->field(
'Blog.id,Blog.name,
Blog.title,
Category.title as categoryName,
User.username as userName'
)->order('Blog.id desc'
)->where('Blog.categoryId=Category.id AND Blog.userId=User.id')
->findAll();
```

而定义了视图模型之后，所有的字段会进行自动处理，添加表别名和字段别名，以及加上基础条件进行查询，从而简化了原来视图的复杂查询。

5 查询条件

在使用视图模型的时候，需要注意查询条件的问题，因为我们已经有定义了基础的查询条件，如果要在查询的时候增加额外的条件，尽可能使用 Map 对象或者数组方式，否则就需要手动添加基础条件。

例如 ：

```
$Model = D("BlogView");
```

```
$where['id']    =   array('gt', '10');
$Model->field('id,name,title,categoryName,userName')->order('id
desc')->where($where)->findAll();
```

如果我们使用字符串方式的条件，那么必须手动添加视图的基础条件

```
$Model->findAll('Blog.categoryId=Category.id AND Blog.userId=User.id AN
D Blog.id in (1,3,6)');
```

6 注意事项

- ✧ 目前的视图模型仅仅用于查询操作，其他操作暂时使用关联操作进行处理；
- ✧ 视图模型对应的数据表并非一定要创建单独的模型，但是建议创建；
- ✧ 视图模型目前对某些数据库的支持和测试尚未完善