SaeModel 使用说明

一，简介

SaeModel 是一个能运行在 SAE 平台的简单的数据库模型。为我们数据库操作带来方便，他实现了基本的 CURD 操作，具有 ORM 和 ActiveRecords 模式，实现了连贯操作，自动验证等功能。

我们不用写复杂的 sql 语句，也不用写复杂的字段验证代码。

比如我们要将表单中的数据存入数据库，如果表单的字段很多时，写原生的 sql 语句，将会写很长，而且容易出错。如果使用 SaeModel， 只需要简单的两行代码：

```
<?php
M（‘TableName’）->create();//这一句会自动提取表单数据对应相应的数据库字段。
M（‘TableName’）->add();//这一句会将数据存入数据库中。
?>
```

字段验证也是我们经常需要用到的功能，比如验证字段是否为邮箱格式，是否为网址格式等。使用 SaeModel，你不用写复杂的验证代码，SaeModel 自带常用的验证规则，你只需要定义规则就可以了。

SaeModel 具有 ThinkPHP 操作习惯，你如果熟悉 ThinkPHP，下面的用法你也应该很熟悉，当然也有些和 ThinkPHP 不同的用法。也有些功能强于 ThinkPHP 的地方。如果你还不会 ThinkPHP，准备学习 ThinkPHP。那么你也可以先借助这个小工具作为 ThinkPHP 的入门。

二， 使用说明

1，初始化

在使用之前，你需要做一些初始化工作。主要是导入 SaeModel 类库和配置 SaeModel。如下代码：

```
<?php
include_once 'SaeModel.class.php'; //导入类库
SaeModel::$tablePrefix='sae_'; //配置表前缀
SaeModel::$debug=true; //是否开启调试
SaeModel::$_validate=new MyValidate(); //定义验证类
SaeModel::$cache_expire=0; //定义默认查询缓存过期时间。
?>
```

SaeModel 使用静态属性的方式到达配置的效果。

我们用 M 函数来实例化 SaeModel，函数的参数为你要进行操作的数据库表名（不带前缀），比如 M（‘Table’）， 如果我们定义的配置项 SaeModel::$tablePrefix='sae_'， 实际表示我们要操作 sae_table 表。M 的参数首字母大小写都可以，我们习惯使用首字母大写的方式。

SaeModel::$debug 属性配置是否开启调试。如果关闭调试时会有数据库的字段缓存，你如果更改了表结构，无法对新增的字段进行操作。在代码开发阶段，我们需要开启调试。等代码正式运行时再设置此配置项为 false。

另外还有两个属性配置后面遇到时再作介绍。

2，CURD 操作

添加数据：

使用 SaeModel 的 add 方法可以进行添加数据， 如：

```
$data=array(
//数组的下标对应数据库表字段，数组的值对应存入数据库字段的值。
  'name' =>' SAE',
  'email' =>' saemail@sina.com'
);
$user=M( 'User');//用 M 函数实例化 SaeModel。
$user->add($data);
```

我们也可以使用 data 方法来存储数据，进行连贯操作

```
$user ->data($data)->add();
```

如果你的数据是来自于表单提交的 POST 数据， 可以自己用 create 方法自动获得数据

```
$user ->create();
$user ->add();
```

表单中文本域名称要对应数据库中的字段名称。

当我们使用了 create 或者 data 方法获得了数据后，我们可以用访问属性的方式来访问和修改数据。比如：

```
$user->create();
var_dump($user->email);  //输出 emai 数据值。
$user->email=" xxx@sina.com";//修改 email 值
$user->add(); //不管表单填写的时候是什么 ,存入到数据库的 email 为 xxx@sina.com
```

add 方法，如果添加成功返回添加数据在数据库中的主键。如果添加失败返回 false。

修改数据

使用 SaeModel 的 save 或者 setField 方法可以修改数据。

比如：

```
$data=array(
  'name' =>' SAE',
  'email' =>' saemail@sina.com'
);
M('User')->where("id=5")->save($data);
```

修改主键 id 为 5 的这条数据。

我们可以把主键值放在数据中。

```
$data=array(
  'id' =>5,
  'name' =>' SAE',
  'email' =>' saemail@sina.com'
);
M('User')->save($data);
```

这样也能修改主键为 5 的这条数据。

通过可以使用 data 方法获得数据，进行连贯操作：

```
M('User')->data($data)->save();
```

save 方法也可以结合 create 来自动获得表单数据，这时候需要在 from 表单中有主键的隐藏域。

如果你只修改一个字段的数据， 用 setField 更为方便：

```
M('User')->where('id=5')->setField('email',' saemail@sina.com');
```

删除数据

我们可以使用 delete 方法来删除数据，如：

M（‘User’）->where(“id=5”)->delete();

或者

M(‘User’)->delete(5);

delete 方法如果跟参数 ,传递的值表示是主键的值。 可以传递多个主键 ,用逗号分割：

M(‘User’)->delete(“1,2,3,5”);

查询数据

可以使用 select，find，getField。等方法来进行数据查询。

查询多条数据用 select， 如：

M(‘User’)->select();// 将以二维数组形式返回查询到的数据

有时候我们会加上查询条件

M(‘User’)->where(“name=’sae’”)->select();

我们还可以加上更多的连贯操作,比如：

M(‘User’)->where(“name=’sae’”)->page(5)->limit(10)->order(“id desc”)->select();

上面代码表示查询用户表 name 字段为 sae 按照 id 倒序排列，并进行分页，每页显示 10 条记录，读取第 5 页的数据。更多的连贯操作，将在后面进行介绍。

查询单条数据可以用 find。

M(‘User’)->where(‘id=5’)->find();//会返回一个一维数组。

如果查询条件是主键，可以简单写成。

M(‘User’)->find(5);

getBy+字段名称 表示查询字段为某个值的数据，如：

M（‘User’）->getByName（‘sae’）;//查询 name 字段为 sae 的数据

只读取某个字段的值：

M（‘User’）->where（‘id=5’）->getField（‘name’）;

返回 id 为 5 的那条数据中 name 字段的值。

下面的代码能到达同样的效果：

M（’ User’ ）->getFieldById(5, ’ name’）;

getField 方法还能指定返回多个字段值， 如：

M（‘User’）->getField（‘id,name’）;

此时，返回为一个键名为 id 的值， 键值为 name 的值的数组。

另外，SaeModel 还有统计查询的功能：

M（‘User’）->where（“name=’ sae’ ”）->count();// 统计数据总条数。

M（‘User’）->sum（‘score’）;//计算总和

M（‘User’）->max（‘score’）;//最大值

M（‘User’）->min（‘score’）;//最小值

M（‘User’）->avg（‘score’）;//平均值

3，连贯操作

SaeModel 提供连贯操作方法，能提高我们的开发效率，让代码更加清晰。比如我们要查询 User 表状态为 1 的前 10 条数据，按主键倒序排序。

M（‘User’）->where（‘status=1’）->limit(10)->order（“id desc”）->select();

这里除了 select 方法，其他方法都是可以任意调换顺序的。我们也可以这样写：

M（‘User’）-> order（“id desc”）-> limit(10) ->where（‘status=1’）->select();

这里的 order，limit，where 方法被成为连贯操作方法。

我们可以用以下方式查看程序生成的 sql 语句：

echo M（‘User’）->_sql();

大家可以再执行完连贯操作后输出原生 sql 语句看看，有助于我们理解。

下面对连贯操作方法依次做下介绍。

cache 方法

查询缓存，例如：

M（‘User’）->cache(true,60)->select();

此时会将查询的结果缓存 60 秒，60 秒内不会连接数据库进行查询，而是读取缓存中的数据，SaeModel 会自动用 Memcache 缓存数据。

cache 方法可以跟两个参数，格式为 cache($key,$expire=’’）,$key 可以是布尔值或字符串，当为字符串时，表示定义缓存在 Memcache 中的名称，当为 true 时也能生成缓存，此时缓存名称为查询操作生成的 sql 语句的 md5 值。当为 false 时，表示不进行查询缓存。$expire 为过期时间，单位为秒，可以省略第二个参数，当省略第二个参数时，缓存的过期时间为配置项： SaeModel:: $cache_expire， 这个配置项默认值为 0，表示永不过期。

where 方法

它可以跟一个字符串参数。如：

M（‘User’）->

where（“name=’sae’ and email=’saemail@sina.com’”）->select();

ThinkPHP 的 where 方法还支持字符串等更复杂的参数形式， SaeModel 现只支持

字符串形式。

table 方法

可用 table 定义你需要查询的表，如：

M()->table(“sae_table_a,sae_table_b”)->select();

表示查询 table_a,table_b 两个表。要带上数据库表前缀， 我们也可以采用下面的形式不带表前缀

M()->table(“__TABLE_A__,__TABLE_B__”)->select();

你需要将表名字母都大写，并在前后分别加上两个斜杠，程序就自动给你加上表前缀了。

我们也可以定义表别名，如：

M()->table(“__TABLE_A__ a,__TABLE_B b”)->where(“a.name=’sae’ and b.id=a.id”)->select();

echo M()->_sql();//输出生成的 sql 语句，看看是什么。

data 方法

在增加或者修改数据之前，我们可以使用 data 来生成需要插入或修改的数据。data 方法需要传递一个数组，数组的键名为数据库表的字段名称，数组的键值为字段的值。

如：

```
$data=array(
    ‘name’ =>’ sae’,
     ‘email’ =>’ saemail@sina.com’
);
M(‘User’)->data($data)->add();
```

data 传递的数组中，如果有键名不是数据库表的字段，在进行数据库操作之前，程

序会自动去除多余的数据。

field 方法

定义要查询的字段。如：

M（‘User’）->field（‘name,email’）->select(); //只返回 name 和 email 字段。

我们同样可以定义字段别名:

M（‘User’）->field（‘name as n,email as e’）->select();

order 方法

进行排序时需要使用到 order 方法，如：

M（‘User’）->order（‘id desc’）->select();

支持多个字段排序

M（‘User’）->order（‘id desc,status desc’）->select();

limit 方法

用于对查询结果的限制。格式为 limit（“offset,length”）， 我们也可以只跟一个数字，如：

M（‘User’）->limit(10)->select();

等效于：

M（‘User’）->limit（“0,10”）->select();

page 方法。

有助于我们快速进行页面分页，我们给 page 方法只传递页数，他能自动计算出 limit 的起点和长度。

如： M（‘User’）->page（“2,5”）->select();

每一页显示五条信息，显示第二页的数据。 上面的代码等效于：

M（‘User’）->page(2)->limit(5)->select();

join 方法

进行 join 查询。 如：

M（‘User’）->join（‘sae_info on sae_user.id=sae_info.uid’）->select();

你同样可以使用不带表前缀的方式，也可以对表定义别名

M()->table（‘__USER__ u’）->join（‘__INFO__ i on u.id=i.uid’）->select();

4，字段验证

SaeModel 字段验证方式和 ThinkPHP 的差别是比较大的。

当我们使用 create 方法获得数据时，SaeModel 会自动对数据进行验证和对数据进行强制类型转换。 比如我们字段的类型为 char(20)。 当 create 时发现数据大于 20 字节，此时 create 返回 false， 我们可以用 getError 方法获得错误原因。所以，我们在进行 create 操作时一般需要判断是否成功。如：

```
$user=M（‘User’）;
if(!$user->create()){
    echo $user->getError();//输出错误信息
}else{
   $user->add();//如果 create 成功才进行添加数据
}
```

create 时还会强制转换数据类型，比如字段的类型为 int，再在 create 时这个字段的数据会被 intval 函数强制转换为整数类型。这样使得我们的数据更加安全。

当然，有时候我们除了对字段长度需要验证外，还需要对字段格式进行验证，如是否为邮箱格式，是否为网址格式等。 这时候我们需要自己建立一个验证类。比如我

们需要验证 User 表的 email 字段是否为邮箱类型,验证 name 字段是否唯一：

```
class  myValidate extend Validate{
  public $user=>array(
//格式：‘字段’=>array(‘字段名称’,[‘规则’],[‘错误提示’],[验证时间],[字段状态])
    ‘email’=>array(‘邮箱’,’email’,’邮箱格式不正确’),
    ‘name’=>array(‘名称’,’unique’,’名称已存在，请输入其他名称’)
  );
  }
```

然后我们需要在 SaeModel 初始化的时候配置属性 SaeModel::$_validate，它的值为验证类的实例化对象。如：

```
SaeModel::$_validate=new myValidate();
```

这样我们在进行 create 值，如果 email 的数据不为邮箱格式，或 name 的数据不唯一也会返回 false

下面详细介绍一下如果建立验证类。

首先，我们建立的类需要继承于 Validate 基类。

然后，定义类的属性。

属性，属性的名称为表名（不带前缀），属性的值为一个数组，定义了字段的验证规则。一般格式为：

```
‘字段’=>array(‘字段名称’,[‘验证规则’],[‘错误提示’],[验证时间],[字段状态])
```

字段：为数据库表中的字段名称，一般都是英文的。

字段名称：这个字段的中文名称，比如 email 的中文名称为邮箱，这个中文名称会在数据超出长度时显示。如果定义了 email 的中文名称为邮箱，则当 email 字段数据超出长

度时错误信息会提示 “邮箱超出长度”，如果我们没有定义 email 规则，则提示只会显示“email 超出长度”。

验证规则：可选，如果没有定义验证规则，则此字段只会验证长度。这里的验证规则为验证类的方法名称，Validate 基类已经定义了一些常用的验证方法，有：notempty(不能为空),email(邮箱格式),url(网站格式),unique(判断是否唯一),phone(座机格式),tel(手机格式),chinese(中文格式),english（英文格式），float(小数个数),int(整数格式),ip(ip 格式),idcard(身份证格式)。 如果这些内置的验证方法不能满足你的需求，你也可以在自己的验证类里建立验证方法。验证方法返回 false 表示验证失败，返回 true 表示验证成功。

错误提示： 可选，当验证失败后的错误提示。如果你是自定义验证方法。你还可以在自己方法中动态设置错误提示,只需要定义类的属性，如：$this->_error=”错误提示”，

验证时间： 可选，1 表示插入数据时验证，2 表示更新数据时验证，3 表示插入和更新时都验证。默认为 3。如果你觉得数字不好记忆，你也可以使用常量来表示。 self:: MODEL_INSERT 插入时验证，self:: MODEL_UPDATE 更新时验证 ,self:: MODEL_BOTH 两者都验证。

字段状态：可选，表示当字段是什么状态时才验证。1 为必须验证，2 为字段数据存在时才验证，3 为字段数据不为空时验证。 同样也有常量表示， self:: MUST_VALIDATE 必须验证，self:: EXISTS_VAILIDATE，存在时验证，self:: VALUE_VAILIDATE 不为空时验证。

多重条件的情况：

有时候一个字段需要多个验证规则。比如我们验证 name 字段既要唯一也要是英文。

‘name’ =>array(‘名称’，

array(‘unique’,’ 名称已存在’),

array(‘english’,’名称必须为英文字符’).

);

上面给一个字段定义了两个规则，有时候我们还需要定义规则之间的关系。默认为 and 关系，所有条件都要满足；我们还可以定义 or 关系，只有一个条件满足就通过。

如电话，它可以是手机格式也可以是座机格式

‘phone’=>array(‘电话’，

array(‘tel’),

array(‘phone’),

‘_error’=>’电话格式不正确’，

‘_logic’=>’or’

);

注:create 方法会强制你进行字段验证，如果在做某些特殊操作不需要字段验证时，你可以使用 data 来获取数据， 如：

M(‘User’)->data($_POST)->add();

5，防止 sql 注入。

经过 create 或 data 方法处理过的数据是很安全的，他经过了强制类型转换，又经过了 mysqli_real_escape_string 函数的过滤。

当我们使用 where 方法时应该注意对字符串进行过滤，SaeModel 提供了 es 的快捷方式，方便我们使用，如：

M(‘User’)->where(“name=’”.es($_GET[‘name’])."’")->find();

如果能使用 getByXX 和 getFieldByXX 等方式进行条件查询就尽量使用他们，因为他们会自动对字符串进行过滤。不用我们手动用 es 函数过滤。 如：

M（‘User’）->getByName($_GET[‘name’]);

M（‘User’）->getByFieldName($_GET[‘name’],’email’);

上面的代码会自动对数据进行安全过滤。

另外，使用 find 方法传递主键值的方式也会自动安全过滤。 如：

M（‘User’）->find($_GET[‘id’]);

6， 开发技巧。

（1），使用 M（‘Table’）->_sql(); 可以打印出原生 sql 语句进行查看

（2），可输出页面执行的所有 sql 语句， 在程序执行结束前，打印出

SaeModel : : $ $sqlLog 属性能够查看到页面执行的所有 sql 语句。 它是一个数组，

我们可以循环输出，如：

```
Foreach(SaeModel : : $ $sqlLog as $sql){
    echo $sql.’ <br />’;
}
```

(3)，当执行失败是，你可以使用 M（‘Table’）->getError(); 看看是否有验证错误，如果没有验证错误，你可以使用 M（‘Table’）->getDbError();查看是否有数据库保存信息。

以上方法都是在开发中利于我们调试的。

希望 SaeModel 能给你带来方便，在使用中如遇到问题，可以在新浪微博上@luofei614 进行询问。