## initialize和construct的不同

initialize是thinkphp中的方法，php中没有；

initialize: 子类自动调用父类的\_initialize方法；

\_\_construct:子类想要调用父类的构造方法，必须先调用parent::\_\_construct();

例：

class Father {

public function \_initialize(){

echo “I’m the father”;

}

}

class Son extends Father{

public function index(){}

}

$son = new Son;

$son->index();//自动调用Father的\_initialize()方法；

## import(‘类库命名空间字符串’ , ’起始路径’ , ‘类库后缀’)

导入系统基类库 即library/Think下的

1. 例：import('Think.Util.Array');

等同于：

require THINK\_PATH.'Lib/Util/Array.class.php';

导入扩展类库 Org等

第三方类库：Vender，用法跟import一样，只是它引入的类库都放在vender目录下面

详解见：http://www.thinkphp.cn/info/126.html

## 空操作

空操作是指系统在找不到请求的操作方法的时候，会定位到空操作（\_empty）方法来执行，利用这个机制，我们可以实现错误页面和一些URL的优化。

http://网址/index.php/Home/Main/hello 空操作

1. 在Main控制器里定义一个\_empty(){}方法

function \_empty(){

$this->display(‘error’);

}

在View/Main目录下建一个error.html页面；

当访问的操作不存在时，会自动走\_empty()方法；

## 空控制器

Home/Controller/EmptyController.class.php

<?php

namespace Home\Controller;

use Think\Controller;

class EmptyController extends Controller

{

public function \_empty()

{

echo "访问的页面不存在！";

}

}

当访问不存在的控制器，系统会尝试定位空控制器EmptyController；

## TP下的事务支持

事务是针对数据库本身的，所以可以跨模型操作的 。

|  |
| --- |
| 例：  // 在User模型中启动事务  $User->startTrans()  // 进行相关的业务逻辑操作  $Info = M("Info"); // 实例化Info对象  $Info->save($User); // 保存用户信息  if (操作成功){  // 提交事务  $User->commit()  }else{  // 事务回滚  $User->rollback()  } |

## 状态配置

在办公室：

入口文件定义：

define('APP\_STATUS','office');

就会自动加载该状态对应的配置文件（位于Application/Common/Conf/office.php）

在家：

define('APP\_STATUS','home');

就会自动加载该状态对应的配置文件（位于Application/Common/Conf/home.php）

## 根命名空间

例如 \Org\Util\File,Org就是一个根命名空间，对应的初始命名空间目录就是系统的类库目录（ThinkPHP/Library）。Library目录下面的子目录会自动识别为根命名空间，这些命名空间无需注册即可使用。

在实例化\Org\Util\Auth类的时候，系统会自动加载 ThinkPHP/Library/Org/Util/Auth.class.php 文件。

框架的Library目录下面的命名空间都可以自动识别和定位。

Library目录下面的子目录都是一个根命名空间，也就是说以Think、Org为根命名空间的类都可以自动加载。

若想自动加载ThinkPHP目录下的其他目录，可以注册其他的根命名空间，例如：

'AUTOLOAD\_NAMESPACE' => array(

'My' => THINK\_PATH.'My',

'One' => THINK\_PATH.'One',

)

配置了上面的AUTOLOAD\_NAMESPACE后，如果我们实例化下面的类库

new \My\Net\IpLocation();

new \One\Util\Log();

会自动加载对应的类库文件

ThinkPHP/My/Net/IpLocation.class.php

ThinkPHP/One/Util/Log.class.php

**自动加载的优先级**

在实际的应用类库加载过程中，往往会涉及到自动加载的优先级问题，以Test\MyClass类为例，自动加载的优先顺序如下：

1判断是否有注册了Test\MyClass类库映射，如果有则自动加载类库映射定义的文件；

2判断是否存在Library/Test目录，有则以该目录为初始目录加载；

3判断是否有注册Test根命名空间，有则以注册的目录为初始目录加载；

4如果以上都不成立，则以Test为模块目录进行初始目录加载；

然后以上面获取到的初始目录加载命名空间对应路径的文件；

**手动加载第三方类库**

import（‘Vendor.Zend.Server’）;

## 前置后置操作

|  |
| --- |
| 系统会检测当前操作是否具有前置和后置操作，如果存在就会按照顺序执行，前置和后置操作的定义方式如下：  <?php  namespace Home\Controller;  use Think\Controller;  class IndexController extends Controller{  //前置操作方法  public function \_before\_index(){  echo 'before<br/>';  }  public function index(){  echo 'index<br/>';  }  //后置操作方法  public function \_after\_index(){  echo 'after<br/>';  }  } |

前置和后置操作的注意事项如下：

|  |
| --- |
| 1. 如果当前的操作并没有定义操作方法，而是直接渲染模板文件，那么如果定义了前置和后置方法的话，依然会生效。真正有模板输出的可能仅仅是当前的操作，前置和后置操作一般情况是没有任何输出的。 2. 需要注意的是，在有些方法里面使用了exit或者错误输出之类的话 有可能不会再执行后置方法了。例如，如果在当前操作里面调用了控制器类的error方法，那么将不会再执行后置操作，但是不影响success方法的后置方法执行。 |

## AJAX返回（json）

系统的\Think\Controller类提供了ajaxReturn方法用于AJAX调用后返回数据给客户端。并且支持JSON、JSONP、XML和EVAL四种方式给客户端接受数据，并且支持配置其他方式的数据格式返回。

$data['status'] = 1;

$data['content'] = 'content';

$this->ajaxReturn($data);

默认配置采用JSON格式返回数据

## 连贯操作

1. 使用\_\_USER\_\_可以表示带前缀的完整表名
2. 多表操作：

$Model->field('user.name,role.title')

->table(array('think\_user'=>'user','think\_role'=>'role'))

->limit(10)->select();

3） 命名范围（封装一些特定的查询语句）

在Model类里定义：

|  |
| --- |
| protected $\_scope = array(  // 命名范围normal  'normal'=>array(  'where'=>array('status'=>1),  ),  // 命名范围latest  'latest'=>array(  'order'=>'create\_time DESC',  'limit'=>10,  ),  );  } |

调用：

|  |
| --- |
| $Model = D('News'); // 这里必须使用D方法 因为命名范围在模型里面定义  $Model->scope('normal')->select();  $Model->scope('latest')->select(); |

## Create()

$User = M('User');

$User->create(); //创建User数据对象，将$data赋值给私有属性了

$User->status = 1; // 设置默认的用户状态

$User->create\_time = time(); // 设置用户的创建时间

$User->add(); // 把用户对象写入数据库，add（）不传参数时，默认把私有属性$data赋给他。

注意：add和save,当没有任何参数时，将会获取当前数据对象的值，即$data进行处理。

create()返回值是$data,并将$data赋值给对象的私有属性protected $data，以供data()调用；

模型类里：用insertFields和updateFields属性定义允许插入和更新的字段。

|  |
| --- |
| namespace Home\Model;  use Think\Model;  class UserModel extends Model{  protected $insertFields = 'name,email'; // 新增数据的时候允许写入name和email字段  protected $updateFields = 'email'; // 编辑数据的时候只允许写入email字段  } |

## add（）

$User = M("User"); // 实例化User对象

$User->data($data)->add();

若在add（）之前已经创建了数据对象的话（使用create或者data），add方法就不用再传入数据了。否则需要$User->add($data);

数据过滤：add()方法里有一项\_façade（）操作 ，会过滤掉数据库里没有的字段。

## find()

find()查询一条记录，查询之后会有这步操作：

$this->data = $data;

将查询到的结果赋给数据对象。可以用$User->data()来获取：

|  |
| --- |
| $User = M("User"); // 实例化User对象  $User->where('status=1 AND name="thinkphp"')->find();  dump($User->data()); |

但是一般find()返回的也是这个数据对象，所以直接：

$data = $User->where(…)->find();

## getField()

常用来返回某个字段的一个值，配合where使用

## save()

返回值：成功返回影响的记录数，失败返回false。

更新个别字段用setField（），搭配where使用

|  |
| --- |
| $User-> where('id=5')->setField('name','ThinkPHP');//参数是字段名，值  $data = array('name'=>'ThinkPHP','email'=>'ThinkPHP@gmail.com');  $User-> where('id=5')->setField($data);//参数是数组 |

对于数组类型更新：

|  |
| --- |
| $User->where('id=5')->setInc('score'); // 用户的积分加1  $User->where('id=5')->setDec('score',5); // 用户的积分减5 |

## delete

delete($id)

返回值：删除的记录数

## 动态查询

getBy:根据字段的值查询数据

$user = $User->getByAddress('中国深圳');//根据地址查询到此条记录。

getFieldBy:根据某个字段的值查询另外一个字段的值

$userId = $User->getFieldByName('liu21st','id');//通过name=liu21st,查询到那条记录，然后返回此条记录的id值。

## 子查询

当select(false)时，不进行查询，只返回查询语句；

或者用buildSql(),也是生成sql语句；

以上两种用于子查询：$model->table($subQuery.' a')->where()->order()->select()

## 自动验证

array(

array(验证字段1,验证规则,错误提示,[验证条件,附加规则,验证时间]),

array(验证字段2,验证规则,错误提示,[验证条件,附加规则,验证时间]),

......

);

附加规则：默认是regex，正则验证；

equire 字段必须、email 邮箱、url URL地址、currency 货币、number 数字这些都属于正则验证。

验证时间：

self::MODEL\_INSERT或者1新增数据时候验证

self::MODEL\_UPDATE或者2编辑数据时候验证

self::MODEL\_BOTH或者3全部情况下验证（默认）

注意：还可以根据实际情况进行设定，比如，create($data,4)为登录提交的数据，此时在验证规则里对应验证条件也为4，即是登录时需要验证的规则；

静态验证规则，必须定义模型类，只能用D（）函数实例化。

动态验证：$User->validate($rules)->create()

## error()

在controller中使用$this->error($msg);是返回json数据；

模型类的$this->getError();是获得模型的错误信息；

两者结合，在控制器中使用：$this->error($model->getError());