**TP5**

[一、 用户操作 2](#_Toc19384)

[添加用户 3](#_Toc28684)

[删除用户 3](#_Toc10132)

[二、 角色操作 3](#_Toc29794)

[添加角色 4](#_Toc20726)

[删除角色 4](#_Toc23213)

[编辑角色 4](#_Toc26081)

[三、 菜单操作(权限操作) 5](#_Toc30503)

[添加菜单 5](#_Toc26338)

[删除菜单 5](#_Toc14462)

[四、 数据操作返回值 6](#_Toc32502)

[五、 RBAC数据库设计 6](#_Toc24725)

[角色表(think\_auth\_group) 6](#_Toc10189)

[用户角色表(think\_auth\_group) 7](#_Toc11350)

[权限(规则)表(think\_auth\_rules) 7](#_Toc29911)

[菜单表(think\_auth\_menus) 8](#_Toc10994)

[六、 验证器 9](#_Toc25596)

[控制器中独立验证 10](#_Toc18496)

[控制器中使用验证器 10](#_Toc18682)

[控制器验证(控制器继承controller) 10](#_Toc27434)

[独立验证 11](#_Toc20660)

[验证器 11](#_Toc20535)

[验证器规则设置 12](#_Toc17402)

[验证器规则定义 12](#_Toc18873)

[验证器属性的定义和中文名字的设置 13](#_Toc11668)

[验证器规则自定义 13](#_Toc6702)

[七、 菜单的操作 13](#_Toc25999)

[八、 分类操作 13](#_Toc715)

[分类删除 13](#_Toc21544)

[细节注释 14](#_Toc12116)

[批量更新数据 14](#_Toc4286)

[数组删除问题 14](#_Toc12706)

# 用户操作

超级管理员只有一个，所以添加用户的时候，不能选择超级管理员这个角色的

## 添加用户

|  |  |
| --- | --- |
| **过程** | 1、选择用户对应的角色，一个用户可能有多个角色(think\_auth\_group\_access) 2、添加用户信息用到用户表(think\_auth\_user)， |
| **涉及表** | think\_auth\_user,think\_auth\_group\_access |
| **事务处理** | 是 |
| **思考** | 1. 新用户默认的角色是？如果是注册用户，那么他的默认的角色是普通用户 |

## 删除用户

|  |  |
| --- | --- |
| **过程** | 1. 删除用户信息表中对应的信息(think\_auth\_user)， 2. 删除用户角色表中该用户的信息think\_auth\_group\_access(首先要看改用户是否有权限，有再去做删除操作) |
| **涉及表** | think\_auth\_user,think\_auth\_group\_access |
| **事务处理** | 是 |
| **思考** |  |

# 角色操作

角色操作过程中，我们可以发现

1. 超级管理员是不可以编辑的
2. 超级管理员拥有所有的权限，所以不需要写入think\_auth\_group，同时
3. 超级管理员拥有所有角色功能所以think\_auth\_group\_access(角色分组就不需要，相当于拥有了所有的角色)

## 添加角色

|  |  |
| --- | --- |
| **过程** | 添加角色只需将对应的信息保存到用户角色表中(think\_auth\_group) |
| **涉及表** | think\_auth\_group |
| **事务处理** | 否 |
| **思考** | 1. 是不是在添加角色的时候就立刻进行授权呢？ |

## 删除角色

|  |  |
| --- | --- |
| **过程** | 1、删除角色表中的信息(think\_auth\_group)  2、删除用户角色表中对应的角色信息(think\_auth\_group\_access),确保所有用户不能拥有该角色！！！ |
| **涉及表** | think\_auth\_group、think\_auth\_group\_access |
| **事务处理** | 是 |
| **思考** | 1、我删除了一个角色，但是因为有很多用户，所以遍历用户角色表(think\_auth\_group\_access)找到对应关联的角色删除该条信息，性能肯定会出问题，怎么破 |

## 编辑角色

|  |  |
| --- | --- |
| **过程** |  |
| **涉及表** |  |
| **事务处理** |  |
| **思考** |  |

# 菜单操作(权限操作)

添加菜单：

首先插入到menus表中，如果插入成功以后，

这个时候在auth\_rule中要去查找对应的项，如果没有找到，那么再插入数据

添加菜单的时候根本不需要去操作auth\_group 表中的值

编辑的时候：

首先先获取当前id对应的数据，并保存！

然后更新menus表中数据

然后通过id去auth\_rule查找数据，如果找到，那么就更新，如果没有找到那么就插入数据

删除的时候：

删除menu中的数据

删除auth\_rule中的数据

删除 auth\_group 中的数据

## 添加菜单

|  |  |
| --- | --- |
| **过程** | 添加菜单的时候，相当于添加了权限一样，每添加一个菜单就是添加一权限，菜单可以设置是否显示在侧边栏，菜单表(think\_auth\_menus)和权限表(think\_auth\_rules)是关联的(但是二者却是没有foreign进行关联)   1. 添加菜单到菜单表(think\_admin\_menus) 2. 添加对应信息到权限表(think\_admin\_rules) 3. 超级管理员要添加该id到rules中(think\_auth\_group)，因为超级管理员拥有全部的权限 |
| **涉及表** | think\_admin\_menus,think\_admin\_rules,think\_auth\_group |
| **事务处理** | 否 |
| **思考** | 我们添加菜单的时候，添加一个菜单就在admin\_menu添加一条数据，然后在王auth中添加一条数据，就是这个原理，同时要知道admin\_menu中的这条数据是否是只做菜单 |

## 删除菜单

|  |  |
| --- | --- |
| **过程** | 1、删除菜单表信息(think\_admin\_menus)   1. 删除权限表对应信息(think\_admin\_rules) 2. 删除角色中所有该权限id(think\_auth\_group) |
| **涉及表** | think\_admin\_menus,think\_admin\_rules,think\_auth\_group |
| **事务处理** | 是 |
| **思考** |  |

# 数据操作返回值

|  |  |  |
| --- | --- | --- |
| **插入** | Db | insert 方法添加数据成功返回添加成功的条数insert 正常情况返回 1(返回新增数据的自增主键getLastInsID,insertGetId )  insertAll 返回的时候受影响记录条数，即插入数量  插入失败返回值不详，但是不管是0 还是false 都是false |
| 模型 | 模型的save操作会默认返回主键，如果插入失败就是返回false |
| **更新** | Db | 如果数据发生了变化则会修改成功，返回1，  如何数据还是原来的执行程序则会返回0.  如果失败返回false |
| 模型 | 如果数据发生了变化则会修改成功，返回1，  如何数据还是原来的执行程序则会返回0.  如果失败返回false |
| **删除** | Db | delete 方法返回影响数据的条数，没有删除返回 0 |
| 模型 | 没有删除返回0 |
| **思考** |  |  |

# RBAC数据库设计

## 角色表(think\_auth\_group)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **字段** | **类型** | **是否空** | **默认值** | **注释** |
| id | mediumint(8) | NOT NULL | DEFAULT '' | 角色id |
| title | char(100) | NOT NULL | DEFAULT '' | 角色标题 |
| description | varchar(100) | NOT NULL, |  | 角色描述 |
| status | tinyint(1) | NOT NULL | DEFAULT '1' | 角色是否禁用,1表示启用 |
| rules | char(80) | NOT NULL | DEFAULT '' | 角色拥有的规则 |
| CREATE TABLE `think\_auth\_group` (  `id` mediumint(8) unsigned NOT NULL AUTO\_INCREMENT,  `title` char(100) NOT NULL DEFAULT '',  `description` varchar(100) NOT NULL,  `status` tinyint(1) NOT NULL DEFAULT '1',  `rules` char(80) NOT NULL DEFAULT '',  PRIMARY KEY (`id`) ) ENGINE=MyISAM AUTO\_INCREMENT=40 DEFAULT CHARSET=utf8 | | | | |

## 用户角色表(think\_auth\_group)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **字段** | **类型** | **是否空** | **默认值** | **注释** |
| uid | mediumint(8) | NOT NULL | 无 | 用户id |
| group\_id | mediumint(8) | NOT NULL | 无 | 角色id |
| 1. 两个并没有外键的关系 2. 两个字段都是普通索引 3. 两个字段都是唯一索引 | | | | |
| CREATE TABLE `think\_auth\_group\_access` (  `uid` mediumint(8) unsigned NOT NULL COMMENT '用户id',  `group\_id` mediumint(8) unsigned NOT NULL COMMENT '用户组id',  UNIQUE KEY `uid\_group\_id` (`uid`,`group\_id`),  KEY `uid` (`uid`),  KEY `group\_id` (`group\_id`) ) ENGINE=MyISAM DEFAULT CHARSET=utf8 COMMENT='用户组明细表' | | | | |

key 是数据库的**物理结构**，它包含两层意义和作用，

**一是约束（偏重于约束和规范数据库的结构完整性），**

**二是索引（辅助查询用的）。**

**UNIQUE KEY的用途：主要是用来防止数据插入的时候重复的。**

## 权限(规则)表(think\_auth\_rules)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **字段** | **类型** | **是否空** | **默认值** | **注释** |
| id | mediumint(8) | NOT NULL |  | 主键 |
| name | mediumint(8) | NOT NULL | DEFAULT '' | 规则唯一标识 |
| title | char(20) | NOT NUL | DEFAULT '', | 规则中文名称 |
| type | tinyint(1) | NOT NUL | DEFAULT '1' | 如果type为1， condition字段就可以定义规则表达式 |
| status | tinyint(1) | NOT NUL | DEFAULT '1' | 状态：为1正常，为0禁用 |
| condition | char(100) | NOT NUL | DEFAULT '' | # 规则附件条件,满足附加条件的规则,才认为是有效的规则  规则表达式，为空表示存在就验证，不为空表示按照条件验证 |
| menu\_id | int(11) |  |  |  |
| 1、menu\_id是菜单的id,添加菜单的时候会关联到权限表,但是该字段没有任何的外键关联 | | | | |
| |  | | --- | | CREATE TABLE `think\_auth\_rules` (  `id` mediumint(8) unsigned NOT NULL AUTO\_INCREMENT,  `name` char(80) NOT NULL DEFAULT '',  `title` char(20) NOT NULL DEFAULT '',  `type` tinyint(1) unsigned NOT NULL DEFAULT '1',  `status` tinyint(1) NOT NULL DEFAULT '1',  `condition` char(100) NOT NULL DEFAULT '',  `menu\_id` int(11) unsigned NOT NULL DEFAULT '0' COMMENT '关联菜单id',  PRIMARY KEY (`id`),  UNIQUE KEY `name` (`name`) ) ENGINE=MyISAM AUTO\_INCREMENT=60 DEFAULT CHARSET=utf8 COMMENT='权限规则表' | | | | | |

## 菜单表(think\_auth\_menus)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段 | 类型 | 是否空 | 默认值 | 注释 |
| menu\_id | int(11) |  |  | 主键 |
| parent\_id | int(10) | NOT NULL | DEFAULT '0' | 父级id |
| is\_show | tinyint(1) | NOT NULL | DEFAULT '1' | 是否显示 |
| title | varchar(50) | NOT NULL | DEFAULT '1' | 菜单名称 |
| url | varchar(100) | NOT NULL | 无 | 模块/控制器/方法 |
| param | varchar(100) | NOT NULL | DEFAULT '' | 参数 |
| icon | varchar(50) | NOT NULL | DEFAULT '' | 菜单图标 |
| log\_type | tinyint(1) | NOT NULL | DEFAULT '0' | |  | | --- | | 0不记录日志，1get，2post，3put，4delete | |
| sort\_id | smallint(5) | NOT NULL | DEFAULT '100' | 排序id |
| create\_time | int(10) | NOT NULL | DEFAULT '0' |  |
| update\_time | int(10) | NOT NULL | DEFAULT '1' |  |
| status | tinyint(1) | NOT NULL |  |  |
| CREATE TABLE `think\_admin\_menus` (  `menu\_id` int(11) unsigned NOT NULL AUTO\_INCREMENT COMMENT '菜单id',  `parent\_id` int(10) unsigned NOT NULL DEFAULT '0' COMMENT '父级id',  `is\_show` tinyint(1) unsigned NOT NULL DEFAULT '1' COMMENT '是否显示',  `title` varchar(50) NOT NULL COMMENT '菜单名称',  `url` varchar(100) NOT NULL COMMENT '模块/控制器/方法',  `param` varchar(100) NOT NULL DEFAULT '',  `icon` varchar(50) NOT NULL DEFAULT 'fa-circle-o' COMMENT '菜单图标',  `log\_type` tinyint(1) unsigned NOT NULL DEFAULT '0' COMMENT '0'不记录日志，1get，2post，3put，4delete，先这些啦',  `sort\_id` smallint(5) unsigned NOT NULL DEFAULT '100' COMMENT '排序id',  `create\_time` int(10) unsigned NOT NULL DEFAULT '0',  `update\_time` int(10) unsigned NOT NULL DEFAULT '0',  `status` tinyint(1) unsigned NOT NULL DEFAULT '1' COMMENT '状态：1默认正常，2禁用',  PRIMARY KEY (`menu\_id`) ) ENGINE=InnoDB AUTO\_INCREMENT=29 DEFAULT CHARSET=utf8 COMMENT='后台菜单表' | | | | |

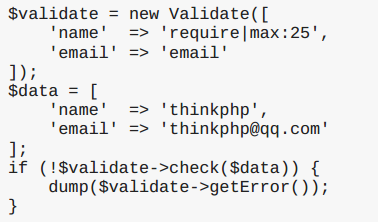
# 验证器

## 控制器中独立验证

引入类 use think\Validate

实例化

进行check



## 控制器中使用验证器

### 控制器验证(控制器继承controller)

可以通过￥this->validate()的方式来进行独立验证。其中还可以分为

第一种：

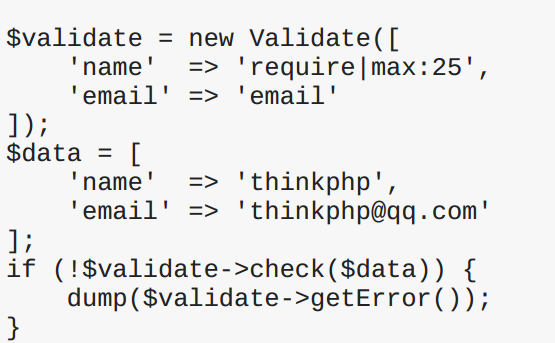


第二种：使用验证器规则



## 独立验证

直接在控制器里面定义：



注意调用的方式

## 验证器

这是 5.0 推荐的验证方式，为具体的验证场景或者数据表定义好验证器类，直接调用验证类的 check 方法即可完成验证

我们在admin 模块下定义了validate文件夹，然后定义一个验证器

Class User extends Validate

定义一个$rule,然后再定义一个$message,可以定义一个场景$scene

然后再控制中$validate = Loader::validate('User'); 引入验证器，然后调用：

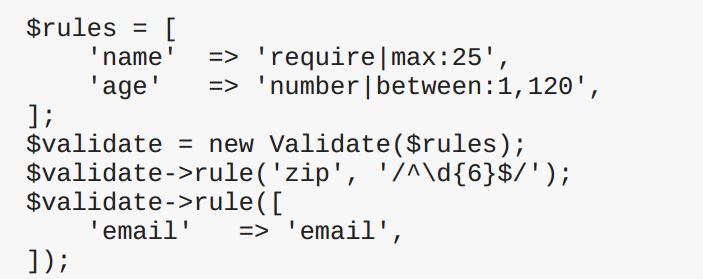
$validate->check();

$validate->getError();

注意调用验证器的规则：

$validate = Loader::validate('User');

## 验证器规则设置



## 验证器规则定义

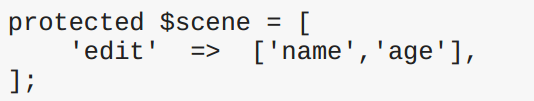


## 验证器属性的定义和中文名字的设置

## 

## 验证器规则自定义

**验证场景的定义**



# 菜单的操作

其实菜单的操作可以分为两大类，可以有以下两个思考：

第一种：我在查询数据的时候就进行的排序，后面再根据pid获取子元素，最终的结果也是一样

第二种：直接查数据,然后通过sort进行排序！！

菜单可以：编辑，禁用，删除，添加下级

# 分类操作

## 分类删除

1. 该分类有子类，那么要连子类一起删除
2. 该分类相关的文章，不能删除，但是需要把对应的文章分类修改为未分类

# 细节注释

## 批量更新数据

$resData = Db::table('think\_test')->update([

['username' => 'thinkphp', 'id' => 1],

['username' => 'thinkphp', 'id' => 2],

]);

这种方式是错误的。根本不能执行，所以方案只能是一个个更新通过foreach

## 数组删除问题

**$groupRule = explode(",", $value["rules"]);**

**foreach ($groupRule as $k => $v) {**

**if (in\_array($v, $arr)) {**

**//如果在arr里面，那么就要删除**

**//array\_splice($groupRule, $k, 1);**

**unset($groupRule[$k]);**

**}**

**}**

**为什么不用array\_splice删除的时候不保持索引(所以删除不干净，剩下一个)，**unset 并未改变数组的原有索引！