三创工作联系

网上管理平台

制作

**目 录**

[1 引用 3](#_Toc445668483)

[1.1 编写目的 3](#_Toc445668484)

[1.2 项目背景 3](#_Toc445668485)

[1.3 名词解释 3](#_Toc445668486)

[2 系统分析 4](#_Toc445668487)

[2.1 系统功能分析 4](#_Toc445668488)

[3 详细设计 5](#_Toc445668489)

[3.1 登录 5](#_Toc445668490)

[3.1.1 功能描述 5](#_Toc445668491)

[3.1.2 功能实现 5](#_Toc445668492)

[3.1.3 功能截图 5](#_Toc445668493)

[3.2 超级管理员功能 5](#_Toc445668494)

[3.2.1 功能描述 5](#_Toc445668495)

[3.2.2 功能实现 6](#_Toc445668496)

[3.2.3 功能截图 6](#_Toc445668497)

[3.3 发布者管理功能 7](#_Toc445668498)

[3.3.1 功能描述： 7](#_Toc445668499)

[3.3.2 功能实现： 7](#_Toc445668500)

[3.3.3 界面截图 7](#_Toc445668501)

[3.4 工作单位管理功能 9](#_Toc445668502)

[3.4.1 功能描述： 9](#_Toc445668503)

[3.4.2 功能实现： 9](#_Toc445668504)

[3.4.3 界面截图 10](#_Toc445668505)

[3.5 管理员管理界面 12](#_Toc445668506)

[3.5.1 功能描述 12](#_Toc445668507)

[3.5.2 功能实现 13](#_Toc445668508)

[3.5.3 界面截图 13](#_Toc445668509)

[4 代码 19](#_Toc445668510)

[4.1 框架 19](#_Toc445668511)

[4.2 登录 19](#_Toc445668512)

[4.3 问题基层代码 20](#_Toc445668513)

[4.4 用户的基层代码 42](#_Toc445668514)

[4.5 单位分类 47](#_Toc445668515)

[4.6 代码说明 50](#_Toc445668516)

[5 项目总结 51](#_Toc445668517)

# 引用

## 编写目的

本文档旨在阐明该项目开发所涉及的相关细节，以及项目的整体结构和各页面之间的关系。

预期读者：后期维护人员，评审人员，项目开发人员，后期开发人员。

## 项目背景

1. 系统名称：三创工作联系网上管理平台
2. 本项目由崇明堡镇政府提出，由陈松平，闻香，尚杰，三人共同完成。
3. 该项目是一个管理问题处理进度的系统。

## 名词解释

|  |  |  |
| --- | --- | --- |
| 关键名词 | 含义 | 备注 |
| HTML | 超文本标记语言 | 用于浏览器解释 |
| CSS | 层叠样式表 | 用于网页的布局 |
| JavaScript | 脚本语言 | 用于和用户的交互 |
| PHP | [超文本](http://baike.baidu.com/view/156868.htm)[预处理器](http://baike.baidu.com/view/499651.htm) | 系统的基础运行语言 |
| mysql | 数据库 | 存储问题数据的介质 |
|  | | |

# 系统分析

## 系统功能分析

|  |  |
| --- | --- |
| **功能** | **说明** |
| **用户登录** | 发布者，工作单位，管理员均从统一入口登录 |
| **发布问题** | 发布者发布新的问题 |
| **分配问题** | 管理员将未分配问题非配给处理单位 |
| **解决问题** | 处理单位收到分配的问题之后去处理，上传处理结果。 |
| **申请延时** | 处理单位收到问题之后，申请延长处理问题的时间 |
| **延时处理** | 管理员决定处理单位的延时申请（同意或拒绝），同意之后，处理单位响应的增加申请延时的天数 |
| **审核问题** | 管理员对处理单位的处理结果进行审核（打回或通过） |
| **删除问题** | 管理员删除无意义的问题，或者发布者删除自己发布的无效问题 |
| **问题列表** | 系统人员查看不同状态的问题列表。管理员可以查看全部问题，发布者只能查看自己发布的全部问题，处理单位只能查看自己处理的问题。 |
| **反馈汇总** | 管理员查看指定时间内分配到各个处理单位的问题情况汇总，处理单位查看指定时间内分配给自己的问题情况汇总 |
| **整改汇总** | 管理员查看指定时间内处理单位的问题处理数量汇总，处理单位查看指定时间内分配给自己的问题处理数量汇总 |
| **用户添加** | 添加发布者或者处理单位 |
| **用户管理** | 查看所有用户，修改用户信息，禁用用户，重置用户密码 |
| **单位分类**  **添加** | 添加单位分类以供添加处理单位时使用 |
| **单位分类**  **管理** | 查看所有单位分类，修改单位分类信息，禁用单位分类 |
| **管理员**  **管理** | 超级管理员对管理员进行添加修改禁用操作 |
|  | |

# 详细设计

## 登录

### 功能描述

用户登录

### 功能实现

用户可以根据用户名和密码登入系统，信息不能为空，如果登录信息有异常，按钮下方会进行提示。登入之前用户可以根据需要选择是否下次自动登入。成功登录系统会根据不同身份跳转到不同的页面，发布者跳转到发布问题页面，管理员跳转到未分配问题列表，工作单位跳转到未处理问题列表，超级管理员跳转到管理员列表。

### 功能截图

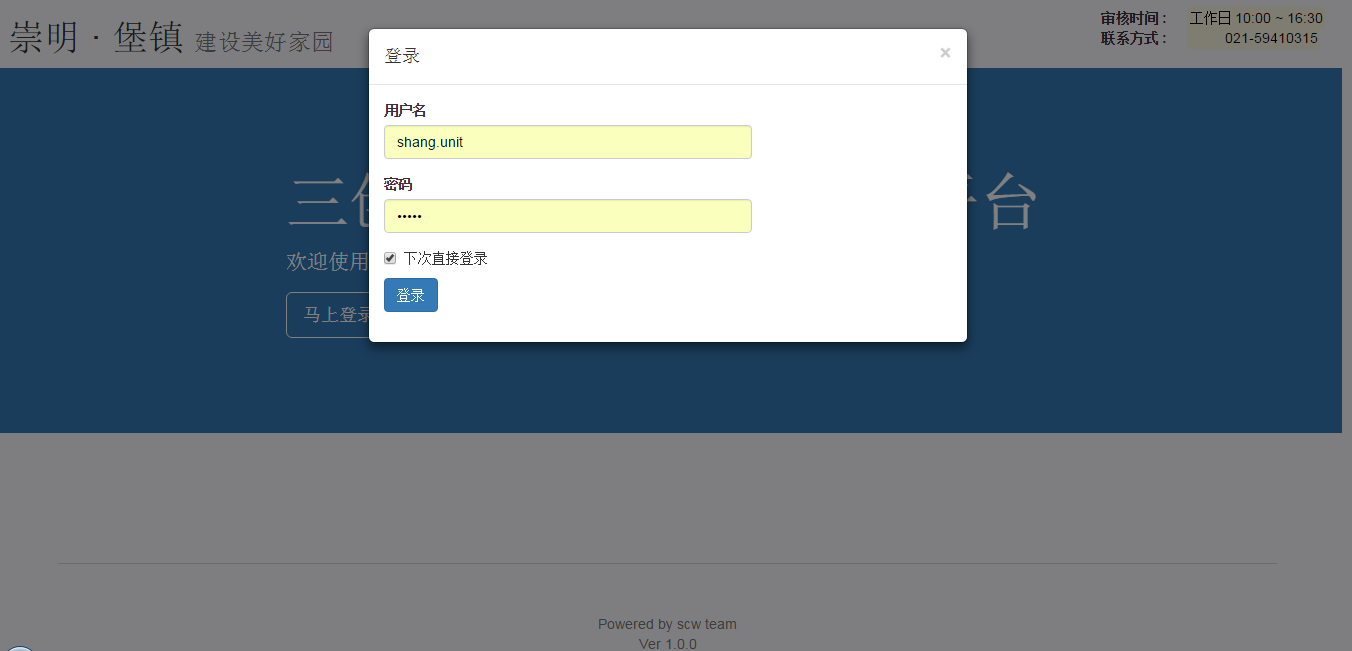


图3-1 登录界面

## 超级管理员功能

### 功能描述

查看管理员列表，列表页面中可以对管理员进行修改和禁用操作。

添加管理员

### 功能实现

1. 查看管理员列表，根据条件搜索管理员
2. 禁用或者启用管理员
3. 修改管理员信息
4. 重置管理员密码
5. 添加管理员

### 功能截图



图3-2 管理员列表和禁用（启用）页面



图3-3 修改管理员信息和重置管理员密码页面



图3-4 添加管理员页面

## 发布者管理功能

### 功能描述：

发布者登录之后发布新的问题

查看不同状态下自己发布的问题

删除未分配的问题

查看自己发布的问题的详情

### 功能实现：

* 1. 发布问题：要处理问题地址，问题描述，上传问题图片，等待管理员分配问题
  2. 删除问题
  3. 查看不同状态下的自己发布的问题列表，并在选中状态下搜索问题
  4. 从问题列表中进入到问题详情页面

### 界面截图



图3-5 发布问题



图3-6 问题列表

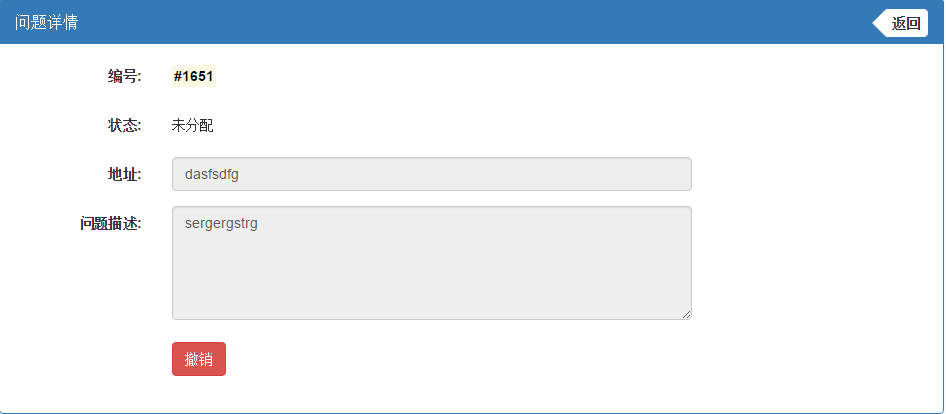


图3-7 删除问题



图3-8 问题详情

## 工作单位管理功能

### 功能描述：

工作单位登录之后发布问题处理结果

查看不同状态下分配给自己的问题

申请处理问题延时

查看分配给自己的问题详情

查看分配给自己的问题反馈汇总

查看分配给自己的问题整改汇总

### 功能实现：

1. 发布问题处理结果：将处理结果拍摄的图片提交到系统等待管理员审核
2. 根据问题的情况结合自己的单位工作情况提出适当的延时
3. 查看不同状态下的分配给自己问题列表，并在选中状态下搜索问题。
4. 从问题列表中进入到问题详情，并查看自己的申请延时的信息

### 界面截图

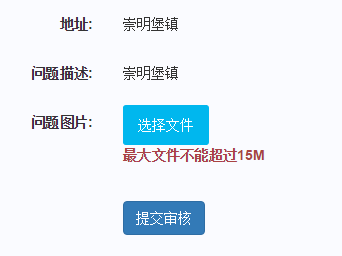


图3-9 上传处理结果界面



图3-10 问题列表

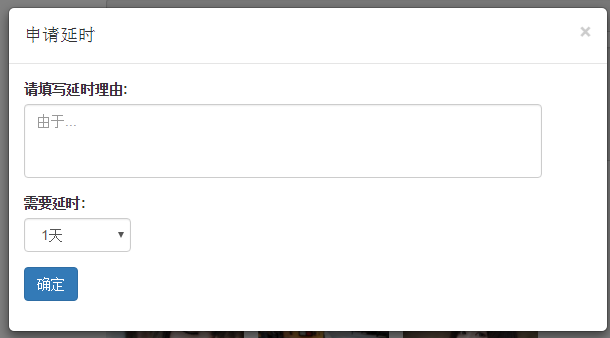


图3-12 申请延时



图3-13 问题详情



图3-14 反馈汇总

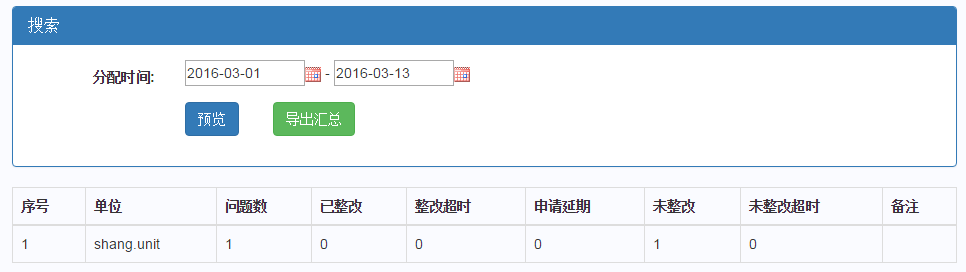


图3-15 整改汇总

## 管理员管理界面

### 功能描述

管理登陆之后，分配未分配问题。

删除问题

查看不同状态的问题

处理问题延时处理申请

审核工作单位的处理结果

查看问题的详情

查看反馈汇总

查看整改汇总

添加用户

修改用户信息，重置用户密码

查看用户列表

添加单位分类

修改单位分类信息

查看单位分类列表

查看系统操作日志

### 功能实现

1. 管理员将未分配的问题非配给相关处理单位
2. 删除发布者发布的没有意义的问题
3. 查看问题详情
4. 查看不同状态下的问题列表，并在选中状态下搜索问题。
5. 处理工作单位的延时申请
6. 审核工作单位上传的处理结果
7. 设置指定时间后查看反馈汇总
8. 设置指定时间后查看处理汇总
9. 添加新的系统用户
10. 修改系统用户的信息，禁用或者开启用户
11. 查看用户列表，设置不同的条件查询用户
12. 添加新的工作单位
13. 修改工作单位信息，禁用或者开启工作单位
14. 查看单位分类列表，这只不同的条件查询单位分类

### 界面截图



图3-16 分配或者删除问题



图3-17问题列表



图3-18 工作单位的延时申请处理



图3-19 反馈汇总



图3-20整改汇总



图3-21 审核处理单位的处理结果



图3-22 添加用户

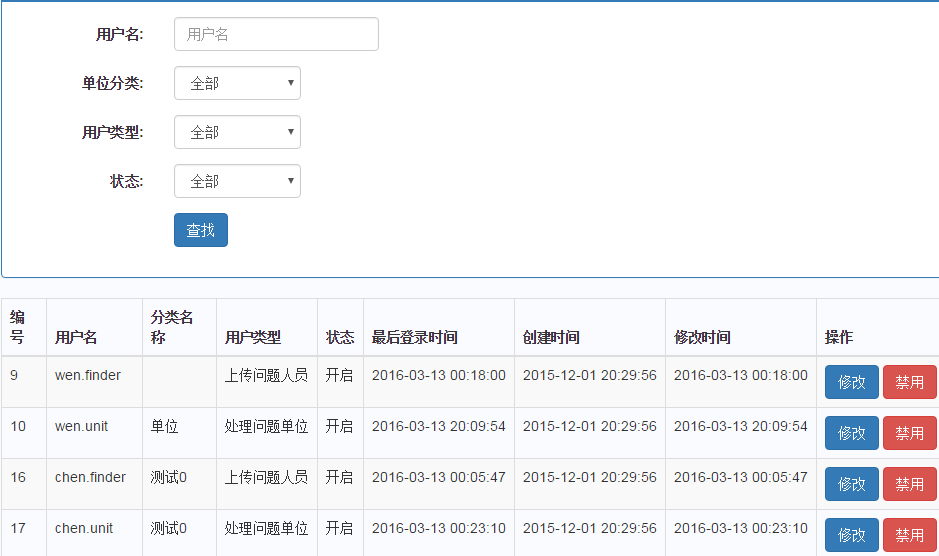


图3-23 用户列表（禁用或者启用用户）



图3-24 修改用户信息（重置用户密码）



图3-25 添加单位分类

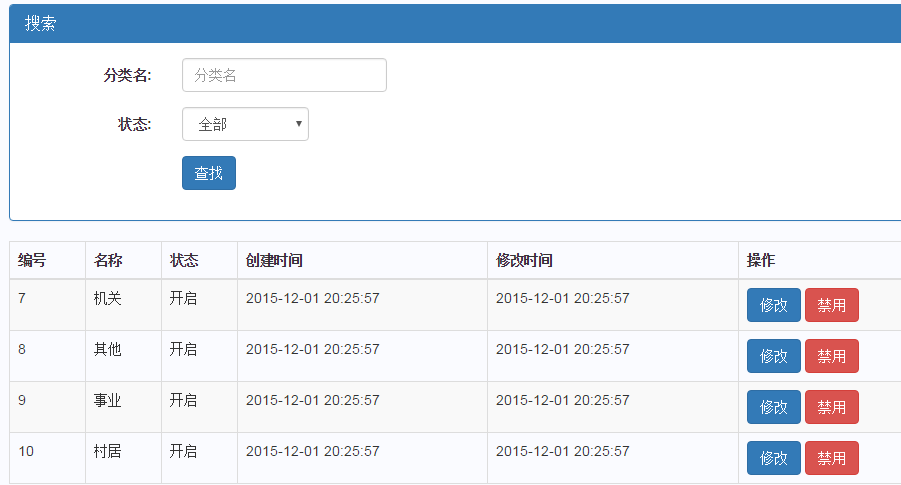


图3-26 单位分类列表（禁用或启用单位分类）



图3-27 修改单位分类信息

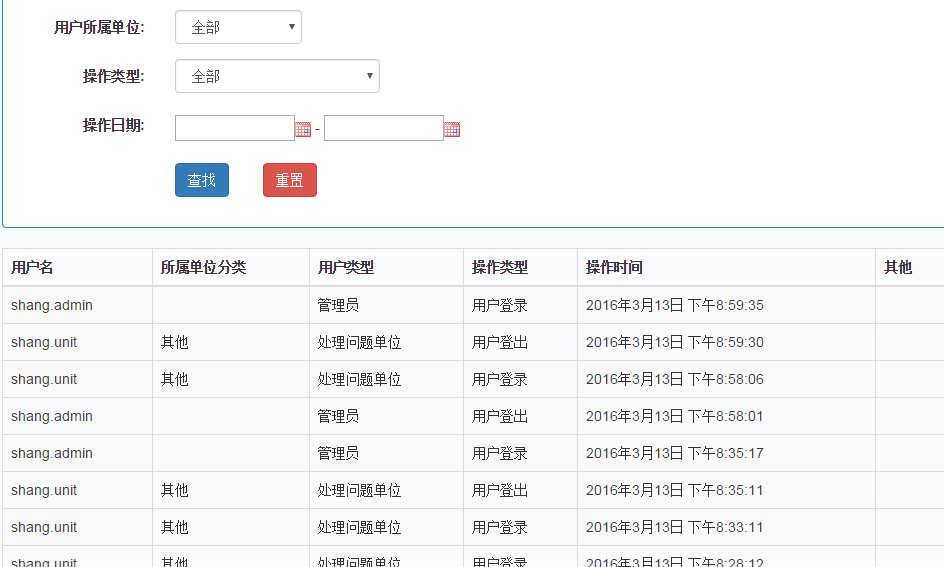


图3-28 系统的操作记录

# 代码

## 框架

此系统基于流行的Yii框架开发，Yii [Framework](http://baike.baidu.com/view/96479.htm)是一个基于组件、用于开发大型 Web 应用的高性能 PHP 框架，Yii是最有效率的PHP框架之一。

快速：Yii只加载您需要的功能。它具有强大的[缓存](http://baike.baidu.com/view/907.htm)支持。它明确的设计能与 AJAX 一起高效率的工作。

安全：Yii 的标准是安全的。它包括了输入验证，输出过滤，SQL 注入和跨站点脚本的预防。

专业：Yii 可帮助您开发清洁和可重用的代码。它遵循了 [MVC](http://baike.baidu.com/view/31.htm)模式，确保了清晰分离逻辑层和[表示层](http://baike.baidu.com/view/239615.htm)。

## 登录

public function actionDoLogin() {

$request = Yii::app()->getRequest();

if (!$request->getIsAjaxRequest()) {

echo CJSON::encode(array("ok"=>false, "msg"=>"Invalid request."));

Yii::app()->end();

}

$error = "";

$username = $request->getParam("username", "");

$password = $request->getParam("password", "");

$remember\_me = $request->getParam("remember\_me", "on");

if (trim($username) == "") {

$error = "用户名不能为空";

} else if ($password == "") {

$error = "密码不能为空";

} else {

$identity = new UserIdentity($username, $password);

if($identity->authenticate()) {

$duration = $remember\_me == "on" ? 3600\*24\*7 : 0;

Yii::app()->user->login($identity, $duration);

list($updated, $error) = Yii::app()->user->updateLastLoginTime();

echo $updated ? CJSON::encode(array("ok"=>true, "msg"=>"登陆成功")) : CJSON::encode(array("ok"=>true, "msg"=>$error));

Yii::app()->end();

} else {

$error = $identity->errorMessage;

}

}

echo CJSON::encode(array("ok"=>false, "msg"=>$error));

}

以上为用户登录的代码，用户登录成功之后会执行

protected function goToMainPage() {

/\*\* @var OperationLogService $op\_log\_service \*/

$op\_log\_service = $this->op\_log\_service;

if (Yii::app()->user->checkAccess("unit")) {

$op\_log\_service->writeOperationLog(Yii::app()->user->getId(), ConstVariables::OP\_USER\_LOGIN);

$this->redirect("/problem/index?nav\_status=5");

} else if (Yii::app()->user->checkAccess("finder")) {

$op\_log\_service->writeOperationLog(Yii::app()->user->getId(), ConstVariables::OP\_USER\_LOGIN);

$this->redirect("/problem/release");

} else if (Yii::app()->user->checkAccess("admin")) {

$op\_log\_service->writeOperationLog(Yii::app()->user->getId(), ConstVariables::OP\_USER\_LOGIN);

$this->redirect("/problem");

} else if (Yii::app()->user->checkAccess("superAdmin")) {

$op\_log\_service->writeOperationLog(Yii::app()->user->getId(), ConstVariables::OP\_USER\_LOGIN);

$this->redirect("/user/manager");

} else {

Yii::app()->user->logout();

throw new CHttpException(500, Yii::t("Site", "privilige error."));

}

}

执行以上代码根据用户的不同角色跳转到不同的默认页面

## 问题基层代码

<?php

class ProblemService extends Service

{

//问题状态

const BE\_CREATED = 0;//未分配

const BE\_ASSIGNED = 1;//已经分配(现已废除, 管理员分配问题后直接置为处理中)

const APPLY\_DELAYING = 2;//申请延时

const APPLY\_ASSISTING = 3;//申请联动

const BE\_BACKING = 4;//退单

const BE\_DEALING = 5;//处理中

const WAIT\_CHECKING = 6;//待审核

const BE\_UNQUALIFIED = 7;//打回

const BE\_QUALIFIED = 8;//审核通过

const BE\_CLOSED = 9;//关闭

const BE\_CANCELED = 10;//撤销

// const MEM\_KEY\_ADMIN\_NEED\_DISPATCH = "admin\_need\_dispatch";

// const MEM\_KEY\_ADMIN\_NEED\_APPROVAL\_DELAY = "admin\_need\_approval\_delay";

public static $status = array(

self::BE\_CREATED => '未分配',

self::BE\_ASSIGNED => '已经分配',

self::APPLY\_DELAYING => '申请延时',

self::APPLY\_ASSISTING => '申请联动',

self::BE\_BACKING => '退单',

self::BE\_DEALING => '处理中',

self::WAIT\_CHECKING => '待审核',

self::BE\_UNQUALIFIED => '打回',

self::BE\_QUALIFIED => '审核通过',

self::BE\_CLOSED => '关闭',

self::BE\_CANCELED => '取销'

);

public static $status\_label\_class = array(

self::BE\_CREATED => 'label-ext-new',

self::BE\_ASSIGNED => 'label-default',

self::APPLY\_DELAYING => 'label-ext-delay',

self::APPLY\_ASSISTING => 'label-default',

self::BE\_BACKING => 'label-default',

self::BE\_DEALING => 'label-ext-deal',

self::WAIT\_CHECKING => 'label-ext-approval',

self::BE\_UNQUALIFIED => 'label-ext-unqualified',

self::BE\_QUALIFIED => 'label-ext-qualified',

self::BE\_CLOSED => 'label-ext-closed',

self::BE\_CANCELED => 'label-ext-closed'

);

/\*\*

\* 创建新的问题

\* @param array $data 问题信息

\* @param array $p\_imgs 问题图片

\* @throws Exception 异常信息

\* @return boolean 创建结果

\*/

public function addNewProblem($data = array(), $p\_imgs = array())

{

if(empty($data)){

self::$errorMsg = '问题资料有缺失';

return false;

}

$transaction = Yii::app()->db->beginTransaction();

try{

$cur\_time = $\_SERVER['REQUEST\_TIME'];

$model = new Problem();

$model->attributes = $data;

$model->create\_time = $cur\_time;

$model->update\_time = $cur\_time;

$res1 = $model->save();

if(!$res1){

throw new Exception(print\_r($model->getErrors(), true));

}

$res2 = true;

if(!empty($p\_imgs)){

$pimg\_service = new ProblemImageService();

$res2 = $pimg\_service->addNewProblemImage($p\_imgs, $model->id);

}

if(!$res2){

throw new Exception($pimg\_service->getLastErrMsg());

}

$plog\_service = new ProblemLogService();

$log\_data = array(

'pid' => $model->id,

'pre\_status' => 0,

'cur\_status' => 0,

'oper\_uid' => Yii::app()->user->id,

'oper\_user' => Yii::app()->user->name,

'log\_desc' => Yii::app()->user->name.'发布问题',

'update\_time' => $cur\_time,

'create\_time' => $cur\_time

);

$res3 = $plog\_service->addNewProblemLog($log\_data);

if(!$res3){

throw new Exception(self::getLastErrMsg());

}

$res = true;

}

catch(Exception $e){

self::$errorMsg = $e->getMessage();

$res = false;

}

if($res){

$transaction->commit();

}

else{

$transaction->rollback();

}

return $res;

}

// public function addNeedDispatchProblemNum($count = 1) {

// $cache = Yii::app()->getCache();

// $cache->add(self::MEM\_KEY\_ADMIN\_NEED\_DISPATCH, 0);

// $cache->set(self::MEM\_KEY\_ADMIN\_NEED\_DISPATCH, $cache->get(self::MEM\_KEY\_ADMIN\_NEED\_DISPATCH) + $count);

// }

// public function getNeedDispatchProblemNum() {

// $cache->add(self::MEM\_KEY\_ADMIN\_NEED\_DISPATCH, 0);

// return $cache->get(self::MEM\_KEY\_ADMIN\_NEED\_DISPATCH);

// }

/\*\*

\* 获得指定问题信息

\* @param int $pid 问题ID

\* @return Problem 问题信息

\*/

public function getProlemById($pid = 0)

{

$problem = Problem::model()->findByPk($pid);

return $problem;

}

/\*\*

\* 获得问题列表

\* @param int $page 当前页数

\* @param int $limit 每页问题数量

\* @param array $condition 搜索条件

\* @param boolean $is\_count 是否只进行计数

\* @return int or array 数量或者问题集合

\*/

public function getProblemByPage($page = 1, $limit = 10 ,$condition = array(), $is\_count = false)

{

$criteria = $this->getFindCond($condition);

if($is\_count){

$count = Problem::model()->count($criteria);

return $count;

}

$criteria->limit = $limit;

$criteria->offset = ($page - 1) \* $limit;

$criteria->order = 'status ASC, update\_time DESC';

$problems = Problem::model()->findAll($criteria);

return $problems;

}

/\*\*

\* 分配处理问题单位

\* @param int $pid 问题ID

\* @param array $data 分配信息

\* @throws Exception 错误信息

\* @return boolean 分配结果

\*/

public function assginDealUser($pid = 0, $data = array())

{

$deal\_uid = isset($data['deal\_uid'])?intval($data['deal\_uid']):0;

if(empty($pid) || empty($deal\_uid)){

self::$errorMsg = '分配单位有缺失';

return false;

}

$transaction = Yii::app()->db->beginTransaction();

try{

$cur\_time = $\_SERVER['REQUEST\_TIME'];

$deal\_month = isset($data['deal\_month'])?intval($data['deal\_month']):0;

$deal\_day = isset($data['deal\_day'])?intval($data['deal\_day']):1;

$deal\_time = $deal\_month \* 30 \*24 + $deal\_day \* 24;

$need\_assistant = isset($data['need\_assistant'])?intval($data['need\_assistant']):0;

$uint\_uids = isset($data['user\_ids'])?$data['user\_ids']:array();

$user\_service = new UserService();

$deal\_user = $user\_service->getGovUserById($deal\_uid);

$problem = $this->getProlemById($pid);

$pre\_status = $problem->status;

$cur\_status = self::BE\_DEALING;

$problem->deal\_cate\_id = $deal\_user->gov\_cate\_id;

$problem->deal\_uid = $deal\_uid;

$problem->deal\_username = $deal\_user->username;

$problem->deal\_time = $deal\_time;

$problem->is\_delay = 0;

$problem->delay\_count = 0;

$problem->delay\_time = 0;

$problem->status = $cur\_status;

$problem->assign\_time = $cur\_time;

$problem->update\_time = $cur\_time;

$problem->is\_assistant = $need\_assistant;

$problem->assist\_unit = json\_encode($uint\_uids);

$res1 = $problem->save();

if(!$res1){

throw new Exception(print\_r($problem->getErrors(), true));

}

$plog\_service = new ProblemLogService();

list($count, $affacted) = $plog\_service->setDelayApplyInvalid($pid);

if ($count > $affacted) {

throw new Exception("延时申请无效化失败");

}

$log\_data = array(

'pid' => $pid,

'pre\_status' => $pre\_status,

'cur\_status' => $cur\_status,

'oper\_uid' => Yii::app()->user->id,

'oper\_user' => Yii::app()->user->name,

'log\_desc' => Yii::app()->user->name.'将问题分配给'.$deal\_user->username,"，时长：".$deal\_time,

'update\_time' => $cur\_time,

'create\_time' => $cur\_time

);

$res2 = $plog\_service->addNewProblemLog($log\_data);

if(!$res2){

throw new Exception(self::getLastErrMsg());

}

$res = true;

}

catch(Exception $e){

$res = false;

self::$errorMsg = $e->getMessage();

}

if($res){

$transaction->commit();

}

else{

$transaction->rollback();

}

return $res;

}

/\*\*

\* 单位接收问题

\* @param int $pid 问题ID

\* @throws Exception 错误信息

\* @return boolean 接受结果

\*/

public function acceptProblem($pid = 0)

{

if(empty($pid)){

self::$errorMsg = '问题信息缺失';

return false;

}

$transaction = Yii::app()->db->beginTransaction();

try{

$cur\_time = $\_SERVER['REQUEST\_TIME'];

$problem = $this->getProlemById($pid);

$pre\_status = $problem->status;

$cur\_status = self::BE\_DEALING;

$problem->status = $cur\_status;

$problem->update\_time = $cur\_time;

$problem->accept\_time = $cur\_time;

$res1 = $problem->save();

if(!$res1){

throw new Exception(print\_r($problem->getErrors(), true));

}

$plog\_service = new ProblemLogService();

$log\_data = array(

'pid' => $pid,

'pre\_status' => $pre\_status,

'cur\_status' => $cur\_status,

'oper\_uid' => Yii::app()->user->id,

'oper\_user' => Yii::app()->user->name,

'log\_desc' => Yii::app()->user->name.'接受处理问题',

'update\_time' => $cur\_time,

'create\_time' => $cur\_time

);

$res2 = $plog\_service->addNewProblemLog($log\_data);

if(!$res2){

throw new Exception(self::getLastErrMsg());

}

$res = true;

}

catch(Exception $e){

self::$errorMsg = $e->getMessage();

$res = false;

}

if($res){

$transaction->commit();

}

else{

$transaction->rollback();

}

return $res;

}

/\*\*

\* 单位提交处理问题凭证

\* @param int $pid 问题ID

\* @param array $solve\_images

\* @throws Exception 错误信息

\* @return boolean 提交结果

\*/

public function solveProblem($pid = 0, $solve\_images = array())

{

if(empty($pid) && empty($solve\_images)){

self::$errorMsg = '凭证信息缺失';

return false;

}

$transaction = Yii::app()->db->beginTransaction();

try{

$cur\_time = $\_SERVER['REQUEST\_TIME'];

$problem = $this->getProlemById($pid);

$pre\_pstatus = $problem->status;

$cur\_status = self::WAIT\_CHECKING;

$problem->status = $cur\_status;

$problem->update\_time = $cur\_time;

$res1 = $problem->save();

if(!$res1){

throw new Exception(print\_r($problem->getErrors(), true));

}

$pimg\_service = new ProblemImageService();

$res2 = $pimg\_service->addNewProblemImage($solve\_images, $pid, 2);

if(!$res2){

throw new Exception($pimg\_service->getLastErrMsg());

}

$plog\_service = new ProblemLogService();

$log\_data = array(

'pid' => $pid,

'pre\_status' => $pre\_pstatus,

'cur\_status' => $cur\_status,

'oper\_uid' => Yii::app()->user->id,

'oper\_user' => Yii::app()->user->name,

'log\_desc' => Yii::app()->user->name.'提交处理问题凭证',

'update\_time' => $cur\_time,

'create\_time' => $cur\_time

);

$res3 = $plog\_service->addNewProblemLog($log\_data);

if(!$res3){

throw new Exception(self::getLastErrMsg());

}

list($count, $affacted) = $plog\_service->setDelayApplyCancel($pid);

if ($count > $affacted) {

throw new Exception("延时申请撤销失败");

}

$res = true;

}

catch(Exception $e){

self::$errorMsg = $e->getMessage();

$res = false;

}

if($res){

$transaction->commit();

}

else{

$transaction->rollback();

}

return $res;

}

/\*\*

\* 设置单位处理问题审核结果

\* @param int $pid 问题ID

\* @param int $solve\_result 处理结果

\* @param string $problem\_log\_remark 处理说明

\* @throws Exception 错误信息

\* @return boolean 设置结果

\*/

public function setSolveResult($pid = 0, $solve\_result = 0, $problem\_log\_remark = '')

{

if(empty($pid) && strlen($solve\_result) == 0){

self::$errorMsg = '凭证信息缺失';

return false;

}

$transaction = Yii::app()->db->beginTransaction();

try{

$cur\_time = $\_SERVER['REQUEST\_TIME'];

$problem = $this->getProlemById($pid);

$pre\_pstatus = $problem->status;

$cur\_status = $solve\_result==1?self::BE\_QUALIFIED:self::BE\_UNQUALIFIED;

$problem->status = $cur\_status;

$problem->check\_time = $cur\_time;

$problem->update\_time = $cur\_time;

// Check whether times up.

if ($cur\_status == self::BE\_QUALIFIED && $problem->assign\_time + ($problem->deal\_time + $problem->delay\_time) \* 3600 < $cur\_time) {

$problem->times\_up = 1;

}

$res1 = $problem->save();

if(!$res1){

throw new Exception(print\_r($problem->getErrors(), true));

}

$plog\_service = new ProblemLogService();

$log\_data = array(

'pid' => $pid,

'pre\_status' => $pre\_pstatus,

'cur\_status' => $cur\_status,

'oper\_uid' => Yii::app()->user->id,

'oper\_user' => Yii::app()->user->name,

'log\_desc' => Yii::app()->user->name.'审核'.$problem->deal\_username.'的处理结果',

'remark' => $problem\_log\_remark,

'update\_time' => $cur\_time,

'create\_time' => $cur\_time

);

$res2 = $plog\_service->addNewProblemLog($log\_data);

if(!$res2){

throw new Exception(self::getLastErrMsg());

}

$res = true;

}

catch(Exception $e){

self::$errorMsg = $e->getMessage();

$res = false;

}

if($res){

$transaction->commit();

}

else{

$transaction->rollback();

}

return $res;

}

/\*\*

\* 单位申请退单

\* @param int $pid 问题ID

\* @param string $problem\_log\_remark 退单理由

\* @throws Exception 错误信息

\* @return boolean 退单申请结果

\*/

public function backProblem($pid = 0, $problem\_log\_remark = '')

{

if(empty($pid) && strlen($problem\_log\_remark) == 0){

self::$errorMsg = '退单请求信息缺失';

return false;

}

$transaction = Yii::app()->db->beginTransaction();

try{

$cur\_time = $\_SERVER['REQUEST\_TIME'];

$problem = $this->getProlemById($pid);

$pre\_pstatus = $problem->status;

$cur\_status = self::BE\_BACKING;

$problem->status = $cur\_status;

$problem->update\_time = $cur\_time;

$res1 = $problem->save();

if(!$res1){

throw new Exception(print\_r($problem->getErrors(), true));

}

$plog\_service = new ProblemLogService();

$log\_data = array(

'pid' => $pid,

'pre\_status' => $pre\_pstatus,

'cur\_status' => $cur\_status,

'oper\_uid' => Yii::app()->user->id,

'oper\_user' => Yii::app()->user->name,

'log\_desc' => Yii::app()->user->name.'申请问题退单',

'remark' => $problem\_log\_remark,

'update\_time' => $cur\_time,

'create\_time' => $cur\_time

);

$res2 = $plog\_service->addNewProblemLog($log\_data);

if(!$res2){

throw new Exception(self::getLastErrMsg());

}

$res = true;

}

catch(Exception $e){

self::$errorMsg = $e->getMessage();

$res = false;

}

if($res){

$transaction->commit();

}

else{

$transaction->rollback();

}

return $res;

}

/\*\*

\* 设置问题延时信息

\* @param int $pid 问题ID

\* @param string $problem\_log\_remark 延时理由

\* @param int $delay\_time 延时时间(小时)

\* @param int $delay\_status 延时状态

\* @throws Exception 错误信息

\* @return boolean 申请结果

\*/

public function delayProblem($pid = 0, $problem\_log\_remark = '', $delay\_time = 0, $delay\_status = 1)

{

if(empty($pid) && strlen($problem\_log\_remark) == 0){

self::$errorMsg = '延时请求信息缺失';

return false;

}

$transaction = Yii::app()->db->beginTransaction();

try{

$cur\_time = $\_SERVER['REQUEST\_TIME'];

$problem = $this->getProlemById($pid);

$pre\_pstatus = $problem->status;

$cur\_status = self::APPLY\_DELAYING;

$problem->status = $cur\_status;

$problem->update\_time = $cur\_time;

$res1 = $problem->save();

if(!$res1){

throw new Exception(print\_r($problem->getErrors(), true));

}

$plog\_service = new ProblemLogService();

$log\_data = array(

'pid' => $pid,

'pre\_status' => $pre\_pstatus,

'cur\_status' => $cur\_status,

'oper\_uid' => Yii::app()->user->id,

'oper\_user' => Yii::app()->user->name,

'log\_desc' => Yii::app()->user->name.'申请延时'.$delay\_time.'个小时',

'remark' => $problem\_log\_remark,

'data' => CJSON::encode(array("hour"=>$delay\_time)),

'update\_time' => $cur\_time,

'create\_time' => $cur\_time

);

$res2 = $plog\_service->addNewProblemLog($log\_data);

if(!$res2){

throw new Exception(self::getLastErrMsg());

}

$res = true;

}

catch(Exception $e){

self::$errorMsg = $e->getMessage();

$res = false;

}

if($res){

$transaction->commit();

}

else{

$transaction->rollback();

}

return $res;

}

/\*\*

\* 问题联动设置

\* @param int $pid 问题ID

\* @param string $problem\_log\_remark 联动理由

\* @param array $unit\_users 联动用户ID集合

\* @param int $is\_assistant 联动状态

\* @throws Exception 错误信息

\* @return boolean 设置联动结果

\*/

public function assitedProblem($pid = 0, $problem\_log\_remark = '', $unit\_users = array(), $is\_assistant = 1)

{

if(empty($pid) && strlen($problem\_log\_remark) == 0){

self::$errorMsg = '联动请求信息缺失';

return false;

}

$transaction = Yii::app()->db->beginTransaction();

try{

$cur\_time = $\_SERVER['REQUEST\_TIME'];

$problem = $this->getProlemById($pid);

$pre\_pstatus = $problem->status;

$cur\_status = self::APPLY\_ASSISTING;

$problem->status = $cur\_status;

$problem->is\_assistant = $is\_assistant;

if(!empty($unit\_users)){

$problem->assist\_unit = json\_encode($unit\_users);

}

$problem->update\_time = $cur\_time;

$res1 = $problem->save();

if(!$res1){

throw new Exception(print\_r($problem->getErrors(), true));

}

$plog\_service = new ProblemLogService();

$log\_data = array(

'pid' => $pid,

'pre\_status' => $pre\_pstatus,

'cur\_status' => $cur\_status,

'oper\_uid' => Yii::app()->user->id,

'oper\_user' => Yii::app()->user->name,

'log\_desc' => Yii::app()->user->name.'申请联动APPLYASSITED['.json\_encode($unit\_users).']APPLYASSITED个小时',

'remark' => $problem\_log\_remark,

'update\_time' => $cur\_time,

'create\_time' => $cur\_time

);

$res2 = $plog\_service->addNewProblemLog($log\_data);

if(!$res2){

throw new Exception(self::getLastErrMsg());

}

$res = true;

}

catch(Exception $e){

self::$errorMsg = $e->getMessage();

$res = false;

}

if($res){

$transaction->commit();

}

else{

$transaction->rollback();

}

return $res;

}

/\*\*

\* 发布者撤销问题

\* @param int $pid 问题ID

\* @throws Exception 错误信息

\* @return boolean 撤销结果

\*/

public function cancelProblem($pid = 0)

{

if(empty($pid)){

self::$errorMsg = '撤销请求信息缺失';

return false;

}

$transaction = Yii::app()->db->beginTransaction();

try{

$cur\_time = $\_SERVER['REQUEST\_TIME'];

$problem = $this->getProlemById($pid);

$pre\_pstatus = $problem->status;

$cur\_status = self::BE\_CANCELED;

$problem->status = $cur\_status;

$problem->update\_time = $cur\_time;

$res1 = $problem->save();

if(!$res1){

throw new Exception(print\_r($problem->getErrors(), true));

}

$plog\_service = new ProblemLogService();

$log\_data = array(

'pid' => $pid,

'pre\_status' => $pre\_pstatus,

'cur\_status' => $cur\_status,

'oper\_uid' => Yii::app()->user->id,

'oper\_user' => Yii::app()->user->name,

'log\_desc' => Yii::app()->user->name.'撤销问题',

'update\_time' => $cur\_time,

'create\_time' => $cur\_time

);

$res2 = $plog\_service->addNewProblemLog($log\_data);

if(!$res2){

throw new Exception(self::getLastErrMsg());

}

$res = true;

}

catch(Exception $e){

self::$errorMsg = $e->getMessage();

$res = false;

}

if($res){

$transaction->commit();

}

else{

$transaction->rollback();

}

return $res;

}

/\*\*

\* 获取延时申请 获取还没有审核通过同时又提出延时申请的记录

\* @param int $deal\_uid 处理问题人员ID

\* @return array [{`id`, `description`, `remark`, `data`, `log\_id`}, ...]

\*/

public function getWaitingDelayApplies() {

$sql = "SELECT `p`.`id`, `p`.`description`, `pl`.`remark`, `pl`.`data`, `pl`.`id` AS `log\_id`

FROM `problem` AS `p`

INNER JOIN `problem\_log` AS `pl` ON `p`.`id`=`pl`.`pid`

WHERE `p`.`status`=:status\_apply\_delaying AND

`pl`.`status`=:status\_delay\_waiting AND `pl`.`cur\_status`=:status\_apply\_delaying

ORDER BY `pl`.`create\_time` DESC";

$params = array(

":status\_apply\_delaying"=>self::APPLY\_DELAYING,

":status\_delay\_waiting"=>ProblemLogService::STATUS\_DELAY\_WAIT

);

return Yii::app()->getDb()->createCommand($sql)->queryAll(true, $params);

}

/\*\*

\* 同意延时申请

\* @param int $id 问题ID

\* @param int $log\_id 延时申请ID

\* @return boolean 操作是否成功

\*/

public function agreeDelay($id, $log\_id) {

$transaction = Yii::app()->db->beginTransaction();

$res = false;

try {

// Get delay hour.

$sql = "SELECT `data` FROM `problem\_log` WHERE `id`=:log\_id";

$params = array(":log\_id"=>$log\_id);

$data = Yii::app()->getDb()->createCommand($sql)->queryScalar($params);

$data = $data ? CJSON::decode($data) : array();

$delay\_time = isset($data["hour"]) ? $data["hour"] : 0;

// Update problem. If apply has approvaled by other admin, the update will not success and rollback transaction.

$sql = "UPDATE `problem`

INNER JOIN `problem\_log` ON `problem\_log`.`pid`=`problem`.`id`

SET `problem`.`status`=:status\_be\_dealing, `problem`.`is\_delay`=:is\_delay,

`problem`.`delay\_count`=`problem`.`delay\_count`+1, `problem`.`delay\_time`=`problem`.`delay\_time`+:delay\_time,

`problem`.`update\_time`=:update\_time

WHERE `problem`.`id`=:id AND `problem\_log`.`id`=:log\_id AND

`problem`.`status`=:status\_apply\_delaying AND `problem\_log`.`status`=:status\_delay\_wait";

$params = array(

":is\_delay" => 1,

":delay\_time" => $delay\_time,

":update\_time" => Util::time(),

":id" => $id,

":log\_id" => $log\_id,

":status\_be\_dealing" => self::BE\_DEALING,

":status\_apply\_delaying" => self::APPLY\_DELAYING,

":status\_delay\_wait" => ProblemLogService::STATUS\_DELAY\_WAIT,

);

$affacted = Yii::app()->getDb()->createCommand($sql)->execute($params);

if ($affacted == 0) {

throw new CHttpException(500, "更新问题失败");

}

// Update problem log. If apply has approvaled by other admin, the update will not success and rollback transaction.

$sql = "UPDATE `problem\_log`

INNER JOIN `problem` ON `problem\_log`.`pid`=`problem`.`id`

SET `problem\_log`.`status`=:status\_delay\_agree, `problem\_log`.`update\_time`=:update\_time

WHERE `problem\_log`.`id`=:log\_id AND `problem`.`id`=:id AND

`problem\_log`.`status`=:status\_delay\_wait";

$params = array(

":id" => $id,

":log\_id" => $log\_id,

":status\_delay\_wait" => ProblemLogService::STATUS\_DELAY\_WAIT,

":status\_delay\_agree" => ProblemLogService::STATUS\_DELAY\_AGREE,

":update\_time" => Util::time(),

);

$affacted = Yii::app()->getDb()->createCommand($sql)->execute($params);

if ($affacted == 0) {

throw new CHttpException(500, "更新延时申请失败");

}

$res = true;

} catch(Exception $e){

self::$errorMsg = $e->getMessage();

$res = false;

}

if ($res) {

$transaction->commit();

} else {

$transaction->rollback();

}

return $res;

}

/\*\*

\* 拒绝延时申请

\* @param int $id 问题ID

\* @param int $log\_id 延时申请ID

\* @return boolean 操作是否成功

\*/

public function refuseDelay($id, $log\_id, $remark) {

$transaction = Yii::app()->db->beginTransaction();

$res = false;

try {

// Put refuse remark into delay apply data.

$sql = "SELECT `data` FROM `problem\_log` WHERE `id`=:log\_id";

$params = array(":log\_id"=>$log\_id);

$data = Yii::app()->getDb()->createCommand($sql)->queryScalar($params);

$data = $data ? CJSON::decode($data) : array();

$data["delay\_refuse\_remark"] = $remark;

$data = CJSON::encode($data);

// Update problem. If apply has approvaled by other admin, the update will not success and rollback transaction.

$sql = "UPDATE `problem`

INNER JOIN `problem\_log` ON `problem\_log`.`pid`=`problem`.`id`

SET `problem`.`status`=:status\_be\_dealing, `problem`.`update\_time`=:update\_time

WHERE `problem`.`id`=:id AND `problem\_log`.`id`=:log\_id AND

`problem`.`status`=:status\_apply\_delaying AND `problem\_log`.`status`=:status\_delay\_wait";

$params = array(

":update\_time" => Util::time(),

":id" => $id,

":log\_id" => $log\_id,

":status\_be\_dealing" => self::BE\_DEALING,

":status\_apply\_delaying" => self::APPLY\_DELAYING,

":status\_delay\_wait" => ProblemLogService::STATUS\_DELAY\_WAIT,

);

$affacted = Yii::app()->getDb()->createCommand($sql)->execute($params);

if ($affacted == 0) {

throw new CHttpException(500, "更新问题失败");

}

// Update problem log. If apply has approvaled by other admin, the update will not success and rollback transaction.

$sql = "UPDATE `problem\_log`

INNER JOIN `problem` ON `problem\_log`.`pid`=`problem`.`id`

SET `problem\_log`.`status`=:status\_delay\_refuse, `problem\_log`.`data`=:data,

`problem\_log`.`update\_time`=:update\_time

WHERE `problem\_log`.`id`=:log\_id AND `problem`.`id`=:id AND

`problem\_log`.`status`=:status\_delay\_wait";

$params = array(

":id" => $id,

":log\_id" => $log\_id,

":status\_delay\_wait" => ProblemLogService::STATUS\_DELAY\_WAIT,

":status\_delay\_refuse" => ProblemLogService::STATUS\_DELAY\_REFUSE,

":update\_time" => Util::time(),

":data" => $data,

);

$affacted = Yii::app()->getDb()->createCommand($sql)->execute($params);

if ($affacted == 0) {

throw new CHttpException(500, "更新延时申请失败");

}

$res = true;

} catch(Exception $e){

self::$errorMsg = $e->getMessage();

$res = false;

}

if ($res) {

$transaction->commit();

} else {

$transaction->rollback();

}

return $res;

}

/\*\*

\* 管理员关闭问题

\* @param int $pid 问题ID

\* @throws Exception 错误信息

\* @return boolean 撤销结果

\*/

public function closeProblem($pid = 0)

{

if(empty($pid)){

self::$errorMsg = '关闭请求信息缺失';

return false;

}

$transaction = Yii::app()->db->beginTransaction();

try{

$cur\_time = $\_SERVER['REQUEST\_TIME'];

$problem = $this->getProlemById($pid);

$pre\_pstatus = $problem->status;

$cur\_status = self::BE\_CLOSED;

$problem->status = $cur\_status;

$problem->update\_time = $cur\_time;

$res1 = $problem->save();

if(!$res1){

throw new Exception(print\_r($problem->getErrors(), true));

}

$plog\_service = new ProblemLogService();

$log\_data = array(

'pid' => $pid,

'pre\_status' => $pre\_pstatus,

'cur\_status' => $cur\_status,

'oper\_uid' => Yii::app()->user->id,

'oper\_user' => Yii::app()->user->name,

'log\_desc' => Yii::app()->user->name.'关闭问题',

'update\_time' => $cur\_time,

'create\_time' => $cur\_time

);

$res2 = $plog\_service->addNewProblemLog($log\_data);

if(!$res2){

throw new Exception(self::getLastErrMsg());

}

$res = true;

}

catch(Exception $e){

self::$errorMsg = $e->getMessage();

$res = false;

}

if($res){

$transaction->commit();

}

else{

$transaction->rollback();

}

return $res;

}

}

以上为问题的基本操作

## 用户的基层代码

<?php

//用户服务

class UserService extends Service

{

const DEFAULT\_PWD = '123456';

/\*\*

\* 获得用户列表

\* @param int $page 页数

\* @param int $limit 每页个数

\* @param array $condition 条件

\* @return array 用户集合

\*/

public function getAllUsersByPage($page = 1, $limit = 10, $condition = array())

{

$criteria = $this->getFindCond($condition);

$criteria->addInCondition('u\_type', array(1,2));

$count = GovUser::model()->count($criteria);

$criteria->limit = $limit;

$criteria->offset = ($page - 1) \* $limit;

$lists = GovUser::model()->findAll($criteria);

return array($lists, $count);

}

/\*\*

\* 获得管理员列表

\* @param int $page 页数

\* @param int $limit 每页个数

\* @param array $condition 条件

\* @return array 管理员集合

\*/

public function getAllManagersByPage($page = 1, $limit = 10, $condition = array())

{

$criteria = $this->getFindCond($condition);

$criteria->compare('u\_type', 3);

$count = GovUser::model()->count($criteria);

$criteria->limit = $limit;

$criteria->offset = ($page - 1) \* $limit;

$lists = GovUser::model()->findAll($criteria);

return array($lists, $count);

}

/\*\*

\* 创建用户

\* @param array $data 用户信息

\* @return GovUser 用户信息

\*/

public function createGovUser($data = array())

{

if(empty($data)){

self::$errorMsg = '用户信息不能为空';

return null;

}

$username = isset($data['username'])?trim($data['username']):"";

if(empty($username)){

self::$errorMsg = '用户名不能为空';

return null;

}

if($this->getGovUserByName($username)){

self::$errorMsg = '该用户名已经存在';

return null;

}

$model = new GovUser();

$cur\_time = $\_SERVER['REQUEST\_TIME'];

$cate\_service = new CategoryService();

$cate\_id = isset($data['gov\_cate\_id'])?$data['gov\_cate\_id']:0;

$cate\_info = $cate\_service->getGovCateById($cate\_id);

$model->attributes = $data;

$model->password = CPasswordHelper::hashPassword(self::DEFAULT\_PWD);

$model->gov\_cate\_name = $cate\_info===null?"":$cate\_info->cate\_name;

$model->create\_time = $cur\_time;

$model->update\_time = $cur\_time;

if($model->save()){

if (isset(Yii::app()->params["gov\_type\_role"][$model->u\_type])) {

Yii::app()->authManager->assign(Yii::app()->params["gov\_type\_role"][$model->u\_type], $model->id);

}

return $model;

}

self::$errorMsg = print\_r($model->getErrors(), true);

return null;

}

/\*\*

\* 修改用户

\* @param int $uid 用户ID

\* @return GovUser 用户信息

\*/

public function updateGovUser($uid = 0, $data = array())

{

$model = $this->getGovUserById($uid);

if($model === null){

self::$errorMsg = '该用户不存在';

return null;

}

$username = isset($data['username'])?trim($data['username']):"";

if(empty($username)){

self::$errorMsg = '用户名不能为空';

return null;

}

if($username != $model->username && $this->getGovUserByName($username)){

self::$errorMsg = '该用户名已经存在';

return null;

}

$old\_u\_type = $model->u\_type;

$cur\_time = $\_SERVER['REQUEST\_TIME'];

$cate\_service = new CategoryService();

$cate\_id = isset($data['gov\_cate\_id'])?$data['gov\_cate\_id']:0;

$cate\_info = $cate\_service->getGovCateById($cate\_id);

$model->attributes = $data;

$model->password = CPasswordHelper::hashPassword(self::DEFAULT\_PWD);

$model->gov\_cate\_name = $cate\_info===null?"":$cate\_info->cate\_name;

$model->update\_time = $cur\_time;

if($model->save()){

if ($old\_u\_type != $model->u\_type && isset(Yii::app()->params["gov\_type\_role"][$old\_u\_type]) && isset(Yii::app()->params["gov\_type\_role"][$model->u\_type])) {

Yii::app()->authManager->revoke(Yii::app()->params["gov\_type\_role"][$old\_u\_type], $model->id);

Yii::app()->authManager->assign(Yii::app()->params["gov\_type\_role"][$model->u\_type], $model->id);

}

return $model;

}

self::$errorMsg = print\_r($model->getErrors(), true);

return null;

}

/\*\*

\* 根据ID查用户

\* @param int $uid 用户ID

\* @return GovUser 用户信息

\*/

public function getGovUserById($uid = 0)

{

$model = GovUser::model()->findByPk($uid);

return $model;

}

/\*\*

\* 根据ID查用户

\* @param array() $uid 用户ID集合

\* @return GovUser 用户信息集合

\*/

public function getGovUserByIds($uids = array())

{

$criteria = new CDbCriteria();

$criteria->addInCondition('id', $uids);

$users = GovUser::model()->findAll($criteria);

return $users;

}

/\*\*

\* 根据用户名查用户

\* @param string $username 用户名

\* @return GovUser 用户信息

\*/

public function getGovUserByName($username = '')

{

$model = GovUser::model()->findByAttributes(array('username' => $username));

return $model;

}

/\*\*

\* 修改用户状态

\* @param int $uid 用户ID

\* @param int $status 状态

\* @return boolean 修改结果

\*/

public function changeStatus($uid = 0, $status = 0)

{

$res = GovUser::model()->updateByPk($uid, array('status' => $status));

return $res;

}

/\*\*

\* 重置用户密码

\* @param int $uid 用户ID

\* @return boolean 重置结果

\*/

public function resetUserPwd($uid = 0)

{

$res = GovUser::model()->updateByPk($uid, array('password' => CPasswordHelper::hashPassword(self::DEFAULT\_PWD)));

return $res;

}

/\*\*

\* 根据单位分类查询用户，用于分配任务

\* @param int $cate\_id 单位分类ID

\* @return array 用户集合

\*/

public function getUserByCate($cate\_id = 0)

{

$criteria = new CDbCriteria();

$criteria->compare('gov\_cate\_id', $cate\_id);

$criteria->compare('status', 1);

$criteria->compare('u\_type', 2);

$users = GovUser::model()->findAll($criteria);

return $users;

}

/\*\*

\* 根据指定类型查询用户

\* @param int $u\_type 用户类型

\* @return array 用户集合

\*/

public function getUserByType($u\_type = 0)

{

$criteria = new CDbCriteria();

$criteria->compare('status', 1);

$criteria->compare('u\_type', $u\_type);

$users = GovUser::model()->findAll($criteria);

return $users;

}

/\*\*

\* 获得所有有效的用户

\* @return GovUser 分类集合

\*/

public function getAvailableUsers()

{

$users = GovUser::model()->findAllByAttributes(array('status' => 1, 'u\_type' => 2));

return $users;

}

}

?>

以上为用户的基本操作代码

## 单位分类

<?php

//单位分类服务

class CategoryService extends Service

{

/\*\*

\* 获得单位分类列表

\* @param int $page 页数

\* @param int $limit 每页个数

\* @param array $condition 条件

\* @return array 单位分类集合

\*/

public function getAllCatesByPage($page = 1, $limit = 10, $condition = array())

{

$criteria = $this->getFindCond($condition);

$count = GovCategory::model()->count($criteria);

$criteria->limit = $limit;

$criteria->offset = ($page - 1) \* $limit;

$lists = GovCategory::model()->findAll($criteria);

return array($lists, $count);

}

/\*\*

\* 获得所有有效的分类

\* @return GovCategory 分类集合

\*/

public function getAvailableCate()

{

$model = GovCategory::model()->findAllByAttributes(array('status' => 1));

return $model;

}

/\*\*

\* 创建单位分类

\* @param string $cate\_name 单位名称

\* @param int $status 状态

\* @return GovCategory 单位分类信息

\*/

public function createGovCate($cate\_name = '', $status = 0)

{

if(empty($cate\_name)){

self::$errorMsg = '分类名不能为空';

return null;

}

if($this->getGovCateByName($cate\_name)){

self::$errorMsg = '该分类名已经存在';

return null;

}

$cur\_time = $\_SERVER['REQUEST\_TIME'];

$model = new GovCategory();

$model->cate\_name = $cate\_name;

$model->status = $status;

$model->create\_time = $cur\_time;

$model->update\_time = $cur\_time;

if($model->save()){

return $model;

}

self::$errorMsg = print\_r($model->getErrors(), true);

return null;

}

/\*\*

\* 修改单位分类

\* @param int $cid 分类ID

\* @param string $cate\_name 单位名称

\* @param int $status 状态

\* @return GovCategory 单位分类信息

\*/

public function updateGovCate($cid = 0, $cate\_name = '', $status = 0)

{

$model = $this->getGovCateById($cid);

if($model === null){

self::$errorMsg = '该分类不存在';

return null;

}

if(empty($cate\_name)){

self::$errorMsg = '分类名不能为空';

return null;

}

if($cate\_name != $model->cate\_name && $this->getGovCateByName($cate\_name)){

self::$errorMsg = '该分类名已经存在';

return null;

}

$cur\_time = $\_SERVER['REQUEST\_TIME'];

$model->cate\_name = $cate\_name;

$model->status = $status;

$model->update\_time = $cur\_time;

if($model->save()){

return $model;

}

self::$errorMsg = print\_r($model->getErrors(), true);

return null;

}

/\*\*

\* 根据ID查分类

\* @param int $cid 分类ID

\* @return GovCategory 单位分类信息

\*/

public function getGovCateById($cid = 0)

{

$model = GovCategory::model()->findByPk($cid);

return $model;

}

/\*\*

\* 根据分类名查分类

\* @param string $cate\_name 分类名

\* @return GovCategory 单位分类信息

\*/

public function getGovCateByName($cate\_name = '')

{

$model = GovCategory::model()->findByAttributes(array('cate\_name' => $cate\_name));

return $model;

}

/\*\*

\* 修改分类状态

\* @param int $cid 分类ID

\* @param int $status 状态

\* @return boolean 修改结果

\*/

public function changeStatus($cid = 0, $status = 0)

{

$res = GovCategory::model()->updateByPk($cid, array('status' => $status));

return $res;

}

}

?>

## 代码说明

以上贴出来的都是service代码，系统所有的功能都有他们来完成。

和使用者相接触的是controller和view文件中的代码

Controller和view中调用service中的代码实现系统的功能

此次系统开发，代码注释都很清楚，方便后来人的理解和阅读，代码逻辑很清楚，代码复用性和扩展性很强，易于后期开发者方便维护系统，并且快速开发新功能。

此系统的代码的稳健性很强。

# 项目总结

本次项目我们做出了很大的努力，既完成了预定好的功能，并且引入了一些先进的技术，使的改系统更加的完美。

从零做起，将此项目完完整整的交出来。

谢谢。