package com.bizlogic.oa.action;

import java.io.PrintWriter;

import java.text.DateFormat;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.ArrayList;

import java.util.Calendar;

import java.util.Date;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import com.bizlogic.base.DayCode;

import com.bizlogic.base.User;

import com.bizlogic.common.action.BaseAction;

import com.bizlogic.oa.bean.AskForLeave;

import com.bizlogic.oa.bean.AskForLeaveType;

import com.bizlogic.oa.bean.CheckWorkType;

import com.bizlogic.oa.bean.LeavePlanLog;

import com.bizlogic.oa.bean.UserCheckWork;

import com.opensymphony.xwork2.ModelDriven;

public class AttendanceAction extends BaseAction implements ModelDriven {

/\*\*

\* 获取日考勤list页面数据

\*/

public void getDayAtt() throws Exception {

// 获得页面传递值 考勤结束日期

String eCode = super.getParameterValue("code");

// 年份和月份

String ym = eCode.substring(0, 6);

// 设置开始日期

String sCode = ym + "01";

// 如果页面传递的值只有年份加月份，则根据当月天数设置结束日期

if(eCode.length()==6){

// 获取考勤月天数

long dayCount = getDayCount(ym);

eCode = ym + dayCount;

}

// 开始时间

Date sDate = getDate(sCode);

// 结束时间

Date eDate = getDate(eCode);

// 获取当前人所在部门编号

String departmentId = super.getVisit().getDepartment().getId();

// 分页查询与当前登录人同一部门的人

List<User> uList = this.getUserList();

// 查询本月日期信息

List<DayCode> dList = getDayCodeList(sCode, eCode);

// 返回值 格式{"total":总条数,"rows":[{},{}]}

StringBuffer sb = new StringBuffer();

sb.append("{\"total\":"+uList.size()+",\"rows\":[");

for (int i=0; i<uList.size(); i++) {

// 获取被考勤用户

User user = uList.get(i);

// 创建数组，存放考勤信息，用于在for循环外拼接json，arr[0]代表1号的考勤信息

String[] arr = new String[dList.size()];

// 是否全勤标示，true为全勤

boolean flag = true;

// 遍历考勤日期

for (int j=0; j<dList.size(); j++) {

// 日期休假类型

String type = dList.get(j).getDayType();

// 初始默认所有人为全勤

if (type.equals("班")) {

arr[j] = "√";

} else {

arr[j] = "";

}

}

// 获取请假记录

List<AskForLeave> aList = getAskForLeaveList(sDate, eDate, user);

// 判断是否全勤

if (aList.size() > 0) {

// 遍历请假记录

for (int j = 0; j < aList.size(); j++) {

AskForLeave ask = aList.get(j);

// 请假开始时间

Date dStart = ask.getStartTime();

// 请假开始日

int start = dStart.getDate();

// 请假结束时间

Date dEnd = ask.getEndTime();

// 请假结束日

int end = dEnd.getDate();

// 如果请假开始时间小于月初，则开始日应该设为本月的第一天

if (dStart.getTime() < sDate.getTime()) {

start = 1;

}

// 如果请假结束时间大于昨天，则结束日应该设为昨天

if (dEnd.getTime() > eDate.getTime()) {

end = dList.size();

}

// 遍历日期

for (int k = 0; k < dList.size(); k++) {

// 日期休假类型

String type = dList.get(k).getDayType();

// 判断该天是否请假

if (start<=(k+1) && (k+1)<=end) {

// 请假类型

String leaveType = ask.getLeaveType().getLeaveTypeName();

// 请假类型简称

String leaveTypeName = ask.getLeaveType().getNameCode();

//判断请假状态，如果是主管未审批3001，考勤时显示"X"----------------20140922

if(ask.getLeaveStatus().getStatusId().equals("3001")){

leaveTypeName = "X";

}

if (leaveType.equals("探亲假") || leaveType.equals("产假")) {

arr[k] = leaveTypeName;

} else if (type.equals("班")){

arr[k] = leaveTypeName;

}

}

}

}

// 请假信息集合大于0，证明非全勤

flag = false;

}

// 查询考勤记录

List<UserCheckWork> wList = getUserCheckWorkList(sDate, eDate, user);

// 判断是否全勤

if (wList.size() > 0) {

// 遍历日期

for (int z = 0; z < dList.size(); z++) {

// 遍历请假记录

for (int y = 0; y < wList.size(); y++) {

UserCheckWork uck = wList.get(y);

// 考勤日期

Date dateTime = uck.getCheckDate();

int num = dateTime.getDate();

// 将考勤信息放入数组中对应的位置

arr[num-1] = uck.getCheckWorkType().getNameCode();

}

}

// 考勤信息集合大于0，证明非全勤

flag = false;

}

// 拼接姓名

sb.append("{\"name\":\""+user.getName()+"\",\"posName\":\""+user.getPositionName()+"\",");

// 拼接全勤

if (flag) {

sb.append("\"fullWork\":\"√\",");

} else {

sb.append("\"fullWork\":\"X\",");

}

// 遍历数组拼接考勤信息

for (int k=0; k<arr.length; k++) {

sb.append("\""+(k+1)+"\":\""+arr[k]+"\",");

}

// 去掉最后一个逗号

sb.delete(sb.length()-1, sb.length());

sb.append("},");

}

// 去掉最后一个逗号

sb.delete(sb.length()-1, sb.length());

sb.append("]}");

super.getResponse().setCharacterEncoding("utf-8");

super.getResponse().setContentType("text/plain");

PrintWriter out = this.getResponse().getWriter();

out.println(sb.toString());

out.flush();

out.close();

}

/\*\*

\* 去个人日考勤信息管理页面

\*/

public String toAttendanceManage() {

// 获取页面传递条件

String flag = super.getParameterValue("flag");

String[] arr = flag.split(",");

String name = arr[0];

String time = arr[1];

// 将条件发送到页面

this.getRequest().setAttribute("manageFlag", flag);

this.getRequest().setAttribute("manageTime", time);

this.getRequest().setAttribute("manageName", name);

return "toAttendanceManage";

}

/\*\*

\* 获取个人日考勤信息管理页面信息

\*/

public void getAttendanceManage() throws Exception {

// 获取页面传递值

String name = super.getParameterValue("name");

String time = super.getParameterValue("time");

// 获取分页条件

int page = super.getParameterIntValue("page");

int rows = super.getParameterIntValue("rows");

// 页面传递年份

String yid = time.substring(0,4);

// 页面传递月份

String mid = time.substring(4,6);

// 考勤开始时间编号

String sCode = time+"01";

// 考勤结束时间编号

String eCode = "";

// 当前时间

Date date = new Date();

// 当前时间不是1号的时候日期要减一天

if (date.getDate() != 1) {

// 日期减一天

date = minus(date);

}

// 转化字符串

String code = getCode(date);

// 当前时间年份

String nowYid = code.substring(0,4);

// 当前时间月份

String nowMid = code.substring(4,6);

if (yid.equals(nowYid) && mid.equals(nowMid)) {

eCode = code;

} else {

String dayCode = yid+mid;

long number = super.getDefaultManager().getCount("from DayCode d where d.monthCode.id=?", dayCode);

eCode = yid+mid+number;

}

// 考勤开始时间

Date sDate = getDate(sCode);

// 考勤结束时间

Date eDate = getDate(eCode);

// 分页查询考勤信息

List<UserCheckWork> uList = super.getDefaultManager().find(

"from UserCheckWork u where u.status=1 and u.user.name=? " +

"and u.checkDate between ? and ? order by u.checkDate", (page-1)\*rows,

page\*rows, name, sDate, eDate);

// 查询考勤总条数

long count = super.getDefaultManager().getCount(

"from UserCheckWork u where u.status=1 and u.user.name=? " +

"and u.checkDate between ? and ?", name, sDate, eDate);

// 返回值 格式{"total":总条数,"rows":[{},{}]}

StringBuffer sb = new StringBuffer();

sb.append("{\"total\":"+count+",\"rows\":[");

// 遍历集合拼接json

for (int i=0; i<uList.size(); i++) {

UserCheckWork uck = uList.get(i);

String checkDate = getCode1(uck.getCheckDate());

String createTime = getCode1(uck.getCreateTime());

sb.append("{\"number\":\""+uck.getId()+

"\",\"leaveType\":\""+uck.getCheckWorkType().getName()+

"\",\"checkDate\":\""+checkDate+

"\",\"createTime\":\""+createTime+

"\",\"createUser\":\""+uck.getCreateUser().getName()+"\"},");

}

if(uList.size() > 0) {

// 删除最后一个逗号

sb.delete(sb.length()-1, sb.length());

}

sb.append("]}");

super.getResponse().setCharacterEncoding("utf-8");

super.getResponse().setContentType("text/plain");

PrintWriter out = this.getResponse().getWriter();

out.println(sb.toString());

out.flush();

out.close();

}

/\*\*

\* 去新增个人日考勤信息页面

\*/

public String toAttendanceManageAdd() {

// 获取页面传递条件

String flag = super.getParameterValue("flag");

String[] arr = flag.split(",");

String name = arr[0];

List<CheckWorkType> cList = super.getDefaultManager().findAll("from CheckWorkType c where c.status=1");

this.getRequest().setAttribute("addName", name);

this.getRequest().setAttribute("addFlag", flag);

this.getRequest().setAttribute("addType", cList);

return "toAttendanceManageAdd";

}

/\*\*

\* 去修改个人日考勤信息页面

\*/

public String toAttendanceManageEdit() {

// 页面传递条件

String flag = super.getParameterValue("flag");

String[] arr = flag.split(",");

String id = arr[0];

String name = arr[1];

String time = arr[2];

String editFlag = name + "," + time;

// 查询考勤实体

UserCheckWork uck = (UserCheckWork) super.getDefaultManager().getObject("from UserCheckWork u where u.id=?", id);

// 查询考勤类型

List<CheckWorkType> cList = super.getDefaultManager().findAll("from CheckWorkType c where c.status=1");

this.getRequest().setAttribute("uck", uck);

this.getRequest().setAttribute("editType", cList);

this.getRequest().setAttribute("editFlag", editFlag);

return "toAttendanceManageEdit";

}

/\*\*

\* 新增日考勤

\*/

public void saveAttendance() {

// 干部名字

String name = super.getParameterValue("name");

User user = (User) super.getDefaultManager().getObject("from User u where u.name = ?", name);

// 创建人

String creatName = super.getVisit().getName();

User createUser = (User) super.getDefaultManager().getObject("from User u where u.name = ?", creatName);

// 考勤类型

String typeId = super.getParameterValue("typeId");

CheckWorkType type = (CheckWorkType) super.getDefaultManager().getObject("from CheckWorkType c where c.id=?", typeId);

// 考勤日期

String code = super.getParameterValue("checkDate");

Date checkDate = getDate1(code);

UserCheckWork uck = new UserCheckWork();

uck.setUser(user);

uck.setCreateUser(createUser);

uck.setCheckDate(checkDate);

uck.setCheckWorkType(type);

uck.setStatus(1);

uck.setCreateTime(new Date());

super.getDefaultManager().insert(uck);

}

/\*\*

\* 删除日考勤

\*/

public void delAttendance() {

// 要删除的考勤id

String id = super.getParameterValue("id");

// 查询考勤实体

UserCheckWork uck = (UserCheckWork) super.getDefaultManager().getObject("from UserCheckWork u where u.id=?", id);

uck.setStatus(0);

super.getDefaultManager().update(uck);

}

/\*\*

\* 修改日考勤

\*/

public void updateAttendance() {

// 要修改的考勤id

String id = super.getParameterValue("id");

// 查询考勤实体

UserCheckWork uck = (UserCheckWork) super.getDefaultManager().getObject("from UserCheckWork u where u.id=?", id);

// 考勤类型

String typeId = super.getParameterValue("typeId");

CheckWorkType type = (CheckWorkType) super.getDefaultManager().getObject("from CheckWorkType c where c.id=?", typeId);

// 考勤日期

String code = super.getParameterValue("checkDate");

Date checkDate = getDate1(code);

uck.setCheckWorkType(type);

uck.setCheckDate(checkDate);

super.getDefaultManager().update(uck);

}

/\*\*

\* 去打印科室月考勤统计页面

\*/

public String toPrintMonth() {

String code = super.getParameterValue("code");

String dayCode = code.split(";")[0];

String civil = code.split(";")[1];

String yid = dayCode.substring(0,4);

String mid = dayCode.substring(4,6);

this.getRequest().setAttribute("yid", yid);

this.getRequest().setAttribute("mid", mid);

this.getRequest().setAttribute("civil", civil);

return "toPrintMonth";

}

/\*\*

\* 去打印科室年考勤统计页面

\*/

public String toPrintYear() {

String code = super.getParameterValue("code");

String yearCode = code.split(";")[0];

String civil = code.split(";")[1];

this.getRequest().setAttribute("yid", yearCode);

this.getRequest().setAttribute("civil", civil);

return "toPrintYear";

}

/\*\*

\* 去计划休假统计查看页，即打印页

\*/

public String toPrintPlan() {

// 获取查看年份

String planYear = super.getParameterValue("planYear");

// 获取与当前登录人同一部门的人 有效 、 等级大于0

List<User> uList = getUserList();

// 返回集合

List<HashMap> list = new ArrayList<HashMap>();

// 遍历用户集合查询计划集合

for (int i = 0; i < uList.size(); i++) {

HashMap map = new HashMap();

User user = uList.get(i);

// 休假计划

List<LeavePlanLog> pList = (List<LeavePlanLog>) super.getDefaultManager().findAll(

"from LeavePlanLog l where l.leavePlan.planYear = ? " +

"and l.leavePlan.leaveUser.active=1 and l.leavePlan.status=2 " +

"and l.leavePlan.leaveUser.id=? order by l.logId",

planYear, user.getId());

if (pList.size() > 0) {

map.put("pList", pList);

} else {

map.put("user", user);

}

map.put("count", pList.size());

list.add(map);

}

this.getRequest().setAttribute("list", list);

this.getRequest().setAttribute("planYear", planYear);

return "toPrintPlan";

}

/\*\*

\* 获取科室月考勤数据

\*/

public void toMonth() throws Exception {

// 获得页面传递值 考勤结束日期

String eCode = super.getParameterValue("code");

// 获得页面传递值 是否公务员判断

String civil = super.getParameterValue("civil");

// 年份和月份

String ym = eCode.substring(0, 6);

// 设置开始日期

String sCode = ym + "01";

// 如果页面传递的值只有年份加月份，则根据当月天数设置结束日期

if(eCode.length()==6){

// 获取考勤月天数

long dayCount = getDayCount(ym);

eCode = ym + dayCount;

}

// 开始时间

Date sDate = getDate(sCode);

// 结束时间

Date eDate = getDate(eCode);

// 获取考勤日期集合

List<DayCode> dList = getDayCodeList(sCode, eCode);

// 获取当前登录人所在科室所有用户集合

List<User> uList = null;

// 判断是否需要区分是否是公务员，用于打印页面

if (null != civil && !"".equals(civil)) {

uList = getUserList(civil);

} else {

uList = getUserList();

}

// 返回json 格式[{"":"","":""},{"":"","":""}]

StringBuffer sb = new StringBuffer();

sb.append("[");

// 遍历用户集合，拼接返回json值

for (int i = 0; i < uList.size(); i++) {

User user = uList.get(i);

// 创建数组，存放考勤信息，arr[0]代表1号的考勤信息

String[] arr = new String[dList.size()];

// 遍历日期信息集合，初始化考勤信息

for (int j=0; j<dList.size(); j++) {

// 日期休假类型

String type = dList.get(j).getDayType();

// 初始默认所有人为全勤

if (type.equals("班")) {

arr[j] = "√";

} else {

arr[j] = "";

}

}

// 全勤标识 true为全勤

boolean flag = true;

// 获取请假记录

List<AskForLeave> aList = getAskForLeaveList(sDate, eDate, user);

// 判断是否全勤

if (aList.size() > 0) {

// 遍历请假记录

for (int j = 0; j < aList.size(); j++) {

AskForLeave ask = aList.get(j);

// 请假开始时间

Date dStart = ask.getStartTime();

// 请假开始日

int start = dStart.getDate();

// 请假结束时间

Date dEnd = ask.getEndTime();

// 请假结束日

int end = dEnd.getDate();

// 如果请假开始时间小于月初，则开始日应该设为本月的第一天

if (dStart.getTime() < sDate.getTime()) {

start = 1;

}

// 如果请假结束时间大于昨天，则结束日应该设为昨天

if (dEnd.getTime() > eDate.getTime()) {

end = dList.size();

}

// 遍历日期

for (int k = 0; k < dList.size(); k++) {

// 日期休假类型

String type = dList.get(k).getDayType();

// 判断该天是否请假

if (start<=(k+1) && (k+1)<=end) {

// 请假类型

String leaveType = ask.getLeaveType().getLeaveTypeName();

// 请假类型简称

String leaveTypeName = ask.getLeaveType().getNameCode();

//判断请假状态，如果是主管未审批3001，考勤时显示"X"----------------20140922

if(ask.getLeaveStatus().getStatusId().equals("3001")){

leaveTypeName = "X";

}

if (leaveType.equals("探亲假") || leaveType.equals("产假")) {

arr[k] = leaveTypeName;

} else if (type.equals("班")){

arr[k] = leaveTypeName;

}

}

}

}

// 请假信息集合大于0，证明非全勤

flag = false;

}

// 获取考勤记录

List<UserCheckWork> wList = getUserCheckWorkList(sDate, eDate, user);

// 判断是否全勤

if (wList.size() > 0) {

// 遍历考勤记录

for (int j = 0; j < wList.size(); j++) {

UserCheckWork uck = wList.get(j);

// 考勤日期

Date dateTime = uck.getCheckDate();

int num = dateTime.getDate();

// 将考勤信息放入数组中对应的位置

arr[num-1] = uck.getCheckWorkType().getNameCode();

}

// 考勤信息集合大于0，证明非全勤

flag = false;

}

// 姓名拼接

sb.append("{\"name\":\""+user.getName()+"\",");

// 职务拼接

sb.append("\"posName\":\""+user.getPositionName()+"\",");

// 全勤拼接

if (flag) {

sb.append("\"fullWork\":\"√\",");

} else {

sb.append("\"fullWork\":\"X\",");

}

// 遍历数组拼接考勤信息

for (int j=0; j<arr.length; j++) {

sb.append("\""+(j+1)+"\":\""+arr[j]+"\",");

}

// 去掉最后一个逗号

sb.delete(sb.length()-1, sb.length());

sb.append("},");

}

// 删除最后一个逗号

sb.delete(sb.length()-1, sb.length());

sb.append("]");

super.getResponse().setCharacterEncoding("utf-8");

super.getResponse().setContentType("text/plain");

PrintWriter out = this.getResponse().getWriter();

out.println(sb);

out.flush();

out.close();

}

/\*\*

\* 获取科室月考勤统计数据

\*/

public void getMonth() throws Exception {

// 返回json 格式[{"":"","":""},{"":"","":""}]

StringBuffer result = new StringBuffer();

result.append("[");

// 获得页面传递值 考勤结束日期

String eCode = super.getParameterValue("code");

// 获得页面传递值 是否公务员判断

String civil = super.getParameterValue("civil");

// 年份和月份

String ym = eCode.substring(0, 6);

// 设置考勤开始日期

String sCode = ym + "01";

// 如果页面传递值只有年份和月份，则根据当月天数设置结束日期

if(eCode.length()==6){

// 本月天数

long l = getDayCount(ym);

eCode = ym + l;

}

// 开始时间

Date sDate = getDate(sCode);

// 结束时间

Date eDate = getDate(eCode);

// 获取科室下所有用户的集合

List<User> uList = null;

// 判断是否需要区分是否是公务员，用于打印页面

if (null != civil && !"".equals(civil)) {

uList = getUserList(civil);

} else {

uList = getUserList();

}

// 查询本月日期信息

List<DayCode> dList = getDayCodeList(sCode, eCode);

// 获取所有请假类型

List<AskForLeaveType> askTypeList = getAskTypeList();

// 获取所有考勤类型

List<CheckWorkType> cList = getCheckWorkTypeList();

// 遍历用户集合

for (int i=0; i<uList.size(); i++) {

User user = uList.get(i);

result.append("{\"name\":\""+user.getName()+"\",");

result.append("\"posName\":\""+user.getPositionName()+"\",");

// 遍历请假类型集合

for (int j=0; j<askTypeList.size(); j++) {

// 用于统计天数的变量

int count = 0;

AskForLeaveType askType = askTypeList.get(j);

// 获取某人在考勤时间范围内的某种请假类型信息

List<AskForLeave> aList = getAskForLeaveList(sDate, eDate, user, askType);

// 遍历请假信息，统计请假天数

for (int k=0; k<aList.size(); k++) {

AskForLeave ask = aList.get(k);

count += getSum(ask, askType.getLeaveTypeName(), sDate, eDate);

}

// 拼接请假信息

if (count == 0) {

result.append("\""+askType.getLeaveTypeName()+"\":\"\",");

} else {

result.append("\""+askType.getLeaveTypeName()+"\":\""+count+"\",");

}

}

// 遍历考勤类型集合

for (int j=0; j<cList.size(); j++) {

// 用于统计天数的变量

int count = 0;

CheckWorkType checkType = cList.get(j);

// 获取某人在考勤时间范围内的某种考勤类型信息

List<UserCheckWork> wList = getUserCheckWorkList(sDate, eDate, user, checkType);

// 遍历考勤信息，统计天数

count = wList.size();

// 拼接考勤信息

if (count == 0) {

result.append("\""+checkType.getName()+"\":\"\",");

} else {

result.append("\""+checkType.getName()+"\":\""+count+"\",");

}

}

// 删除最后一个逗号

result.delete(result.length()-1, result.length());

result.append("},");

}

// 删除最后一个逗号

result.delete(result.length()-1, result.length());

result.append("]");

super.getResponse().setCharacterEncoding("utf-8");

super.getResponse().setContentType("text/plain");

PrintWriter out = this.getResponse().getWriter();

out.println(result);

out.flush();

out.close();

}

/\*\*

\* 获取科室年考勤统计数据

\*/

public void getYear() throws Exception {

// 返回json 格式[{"":"","":""},{"":"","":""}]

StringBuffer result = new StringBuffer();

result.append("[");

// 统计结束

String eCode = super.getParameterValue("code");

// 是否为公务员

String civil = super.getParameterValue("civil");

// 统计开始

String sCode = eCode.substring(0, 4);

sCode = sCode + "0101";

if (eCode.length() == 4) {

eCode += "1231";

}else if(eCode.length() == 6) {

// 查询当月天数

long l = getDayCount(eCode);

eCode += l;

}

// 开始时间

Date sDate = getDate(sCode);

// 结束时间

Date eDate = getDate(eCode);

// 获取请假类型

List<AskForLeaveType> aList = getAskTypeList();

// 获取考勤类型

List<CheckWorkType> cList = getCheckWorkTypeList();

// 获取与当前登录人同一部门的人

List<User> uList = null;

// 判断是否需要区分是否是公务员，用于打印页面

if (null != civil && !"".equals(civil)) {

uList = getUserList(civil);

} else {

uList = getUserList();

}

// 遍历用户集合，查询请假信息

for (int i=0; i<uList.size(); i++) {

User user = uList.get(i);

result.append("{\"name\":\""+user.getName()+"\",");

result.append("\"posName\":\""+user.getPositionName()+"\",");

// 遍历请假类型集合，查询天数

for (int j=0; j<aList.size(); j++) {

AskForLeaveType askType = aList.get(j);

// 获取某人的当月,某种请假类型天数

List<AskForLeave> askList = getAskForLeaveList(sDate, eDate, user, askType);

int count = 0;

if (askList.size() > 0) {

for (int k=0; k<askList.size(); k++) {

AskForLeave ask = askList.get(k);

count += getSum(ask, askType.getLeaveTypeName(), sDate, eDate);

}

}

if (count == 0) {

result.append("\""+askType.getLeaveTypeName()+"\":\"\",");

} else {

result.append("\""+askType.getLeaveTypeName()+"\":\""+count+"\",");

}

}

// 遍历考勤类型集合

for (int j=0; j<cList.size(); j++) {

// 用于统计天数的变量

int count = 0;

CheckWorkType checkType = cList.get(j);

// 获取某人在考勤时间范围内的某种考勤类型信息

List<UserCheckWork> wList = getUserCheckWorkList(sDate, eDate, user, checkType);

// 遍历考勤信息，统计天数

count = wList.size();

// 拼接考勤信息

if (count == 0) {

result.append("\""+checkType.getName()+"\":\"\",");

} else {

result.append("\""+checkType.getName()+"\":\""+count+"\",");

}

}

// 删除最后一个逗号

result.delete(result.length() - 1, result.length());

result.append("},");

}

// 删除最后一个逗号

result.delete(result.length() - 1, result.length());

result.append("]");

super.getResponse().setCharacterEncoding("utf-8");

super.getResponse().setContentType("text/plain");

PrintWriter out = this.getResponse().getWriter();

out.println(result);

out.flush();

out.close();

}

/\*\*

\* 获取科室年统计数据详细信息

\*/

public String getAttendanceInfo() throws Exception {

// 姓名

String name = super.getParameterValue("name");

// 统计类型

String type = super.getParameterValue("type");

// 年份

String yearId = super.getParameterValue("yearId");

// 标识

int flag = super.getParameterIntValue("flag");

// 开始时间 年初

String sCode = yearId+"0101";

Date sDate = getDate(sCode);

// 结束时间 年末或当前时间的前一天

String eCode = yearId+"1231";

Date eDate = getDate(eCode);

Date date = new Date();

if (eDate.getTime() > date.getTime()) {

date = minus(date);

eCode = getCode(date);

eDate = getDate(eCode);

}

// 返回list

List list = new ArrayList();

int count = 0;

// 考勤

if (flag == 0) {

List<UserCheckWork> uList = super.getDefaultManager().findAll(

"from UserCheckWork a where a.checkDate between ? and ? " +

"and a.status=1 and a.user.name=? and a.checkWorkType.name=? " +

"order by a.checkDate", sDate, eDate, name, type);

for (int i=0; i<uList.size(); i++) {

UserCheckWork ucw = uList.get(i);

Map map = new HashMap();

map.put("start", getCode1(ucw.getCheckDate()));

map.put("end", getCode1(ucw.getCheckDate()));

map.put("dayNum", 1);

map.put("eCode", getCode1(eDate));

map.put("dayCount", 1);

list.add(map);

}

}

// 请假

if (flag == 1) {

// 查询考勤

List<AskForLeave> askList = super.getDefaultManager().findAll(

"from AskForLeave a where (a.startTime between ? and ? " +

"or a.endTime between ? and ? or a.startTime < ? and a.endTime > ?) " +

"and a.appLeaveUser.name=? and a.leaveType.leaveTypeName=? " +

"and a.leaveStatus.statusId = '9999' order by a.startTime",

sDate, eDate, sDate, eDate, sDate, eDate, name, type);

for (int i=0; i<askList.size(); i++) {

AskForLeave ask = askList.get(i);

Date dStart = ask.getStartTime();

Date dEnd = ask.getEndTime();

Map map = new HashMap();

map.put("start", getCode1(dStart));

map.put("end", getCode1(dEnd));

map.put("dayNum", ask.getDayNum());

if (eDate.getTime() > dEnd.getTime()) {

map.put("eCode", getCode1(dEnd));

} else {

map.put("eCode", getCode1(eDate));

}

count = getSum(ask, ask.getLeaveType().getLeaveTypeName(), sDate, eDate);

map.put("dayCount", count);

list.add(map);

}

}

this.getRequest().setAttribute("list", list);

return "getAttendanceInfo";

}

/\*\*

\* 获取街道月考勤统计数据

\*/

public void getTreeMonth() throws Exception {

// 返回json 格式[{"":"","":""},{"":"","":""}]

StringBuffer result = new StringBuffer();

result.append("[");

// 统计结束

String eCode = super.getParameterValue("code");

// 统计开始

String sCode = eCode.substring(0, 6);

sCode = sCode + "01";

// 根据页面传值判断是加载还是查询

if (eCode.length() == 6) {

// 当属查询时，去数据库查询当月天数设置月末时间

long code = getDayCount(eCode);

eCode += code;

}

// 开始时间

Date sDate = getDate(sCode);

// 结束时间

Date eDate = getDate(eCode);

// 获得街道全部街道干部，包括（书记、主任)

List<User> uList = getTreeUserList();

// 请假类型

List<AskForLeaveType> tList = getAskTypeList();

// 获取所有考勤类型

List<CheckWorkType> cList = getCheckWorkTypeList();

// 遍历用户集合，查询请假信息

for (int i=0; i<uList.size(); i++) {

User user = uList.get(i);

result.append("{\"depName\":\""+user.getDepartment().getName()+"\"," +

"\"name\":\""+user.getName()+"\"," +

"\"posName\":\""+user.getPositionName()+"\",");

// 遍历请假类型集合，查询天数

for (int j=0; j<tList.size(); j++) {

AskForLeaveType type = tList.get(j);

// 获取某人的当月,某种请假类型天数

List<AskForLeave> aList = getAskForLeaveList(sDate, eDate, user, type);

int count = 0;

if (aList.size() > 0) {

for (int y=0; y<aList.size(); y++) {

AskForLeave ask = aList.get(y);

count += getSum(ask, type.getLeaveTypeName(), sDate, eDate);

}

}

result.append("\""+type.getLeaveTypeName()+"\":\""+count+"\",");

}

// 遍历考勤类型集合

for (int j=0; j<cList.size(); j++) {

// 用于统计天数的变量

int count = 0;

CheckWorkType checkType = cList.get(j);

// 获取某人在考勤时间范围内的某种考勤类型信息

List<UserCheckWork> wList = getUserCheckWorkList(sDate, eDate, user, checkType);

// 遍历考勤信息，统计天数

count = wList.size();

// 拼接考勤信息

result.append("\""+checkType.getName()+"\":\""+count+"\",");

}

// 删除最后一个逗号

result.delete(result.length() - 1, result.length());

result.append("},");

}

// 删除最后一个逗号

result.delete(result.length() - 1, result.length());

result.append("]");

super.getResponse().setCharacterEncoding("utf-8");

super.getResponse().setContentType("text/plain");

PrintWriter out = this.getResponse().getWriter();

out.println(result);

out.flush();

out.close();

}

/\*\*

\* 获取街道年考勤统计数据

\*/

public void getTreeYear() throws Exception {

// 返回json 格式[{"":"","":""},{"":"","":""}]

StringBuffer result = new StringBuffer();

result.append("[");

// 统计结束

String eCode = super.getParameterValue("code");

// 统计开始

String sCode = eCode.substring(0, 4);

sCode = sCode + "0101";

if (eCode.length() == 4) {

eCode = eCode + "1231";

}

// 开始时间

Date sDate = getDate(sCode);

// 结束时间

Date eDate = getDate(eCode);

// 请假类型

List<AskForLeaveType> tList = getAskTypeList();

// 获取街道所有人

List<User> uList = getTreeUserList();

// 获取所有请假类型

List<AskForLeaveType> askTypeList = getAskTypeList();

// 获取所有考勤类型

List<CheckWorkType> cList = getCheckWorkTypeList();

// 遍历用户集合，查询请假信息

try{

for (int i=0; i<uList.size(); i++) {

User user = uList.get(i);

result.append("{\"depName\":\""+user.getDepartment().getName()+"\"," +

"\"name\":\""+user.getName()+"\"," +

"\"posName\":\""+user.getPositionName()+"\",");

// 遍历请假类型集合，查询天数

for (int j=0; j<tList.size(); j++) {

AskForLeaveType type = tList.get(j);

// 获取某人的当月,某种请假类型天数

List<AskForLeave> aList = getAskForLeaveList(sDate, eDate, user, type);

int count = 0;

if (aList.size() > 0) {

for (int y=0; y<aList.size(); y++) {

AskForLeave ask = aList.get(y);

if(ask.getDayNum()==null||ask.getDayNum().equals("")){

System.out.println("-------------------------------------------------------------------------------");

System.out.println(ask.getAppLeaveUser().getName()+","+ask.getStartTime()+":"+ask.getEndTime());

}

count += getSum(ask, type.getLeaveTypeName(), sDate, eDate);

}

}

result.append("\""+type.getLeaveTypeName()+"\":\""+count+"\",");

}

// 遍历考勤类型集合

for (int j=0; j<cList.size(); j++) {

// 用于统计天数的变量

int count = 0;

CheckWorkType checkType = cList.get(j);

// 获取某人在考勤时间范围内的某种考勤类型信息

List<UserCheckWork> wList = getUserCheckWorkList(sDate, eDate, user, checkType);

// 遍历考勤信息，统计天数

count = wList.size();

// 拼接考勤信息

result.append("\""+checkType.getName()+"\":\""+count+"\",");

}

// 删除最后一个逗号

result.delete(result.length() - 1, result.length());

result.append("},");

}

}catch(Exception e){

e.printStackTrace();

}

// 删除最后一个逗号

result.delete(result.length() - 1, result.length());

result.append("]");

super.getResponse().setCharacterEncoding("utf-8");

super.getResponse().setContentType("text/plain");

PrintWriter out = this.getResponse().getWriter();

out.println(result);

out.flush();

out.close();

}

/\*\*

\* 获取科室名字和登录人名字

\*/

public void getName() throws Exception {

// 当前登录人

String name = super.getVisit().getName();

// 科室

String depName = super.getVisit().getDepartment().getName();

String result = name+","+depName;

super.getResponse().setCharacterEncoding("utf-8");

super.getResponse().setContentType("text/plain");

PrintWriter out = this.getResponse().getWriter();

out.println(result);

out.flush();

out.close();

}

/\*\*

\* 查询月工作日数量

\*/

public void getCount() throws Exception {

// 获得页面传递值

String code = super.getParameterValue("code");

// 年月

String ym = code.substring(0, 6);

// 查询工作日数量

long count = super.getDefaultManager().getCount("from DayCode d where d.dayType='班' and d.monthCode='"+ym+"'");

// 设置返回值

int result = Integer.valueOf(count+"");

super.getResponse().setCharacterEncoding("utf-8");

super.getResponse().setContentType("text/plain");

PrintWriter out = this.getResponse().getWriter();

out.println(result);

out.flush();

out.close();

}

/\*\*

\* 判断日期是否是工作日

\*/

public void judgment() throws Exception {

// 获取页面传递值

String flag = super.getParameterValue("code");

String time = flag.split(",")[0];

String name = flag.split(",")[1];

// 年

String yid = time.substring(0,4);

// 月

String mid = time.substring(5,7);

// 日

String did = time.substring(8,10);

// 拼接无-的日期编号

String code = yid+mid+did;

Date date = getDate(code);

// 返回值 默认为true

String result = "true";

// 查询工作状态

String type = (String) super.getDefaultManager().getObject("select d.dayType from DayCode d where d.id=?", code);

// 如果是工作日则返回提示信息

if (type.equals("假")) {

result = yid+"年"+mid+"月"+did+"日为休息日，不需要考勤！";

}

// 查询当天是否请假

AskForLeave ask = (AskForLeave) super.getDefaultManager().getObject(

"from AskForLeave a where a.leaveStatus.statusId = '9999' " +

"and a.startTime<? and a.endTime>? and a.appLeaveUser.name=?",

date, date, name);

// 如果ask不为空，即当天请假，返回提示信息

if (null!=ask && !"".equals(ask)) {

result = yid+"年"+mid+"月"+did+"日"+name+"已经请了"+ask.getLeaveType().getLeaveTypeName()+"，不需要考勤！";

}

// 查询当天是否已经考勤

UserCheckWork uck = (UserCheckWork) super.getDefaultManager().getObject(

"from UserCheckWork u where u.status=1 and u.checkDate=?", date);

// 如果uck不为空，即当天已经有考勤信息，返回提示信息

if (null!=uck && !"".equals(uck)) {

result = yid+"年"+mid+"月"+did+"日"+name+"已经被考勤为"+uck.getCheckWorkType().getName()+",如有变化请到修改页面进行修改操作！";

}

super.getResponse().setCharacterEncoding("utf-8");

super.getResponse().setContentType("text/plain");

PrintWriter out = this.getResponse().getWriter();

out.print(result);

out.flush();

out.close();

}

public Object getModel() {

// TODO Auto-generated method stub

return null;

}

/\*

\* 根据字符串获取时间 字符串格式 yyyyMMdd

\*/

private Date getDate(String code) {

DateFormat fmt = new SimpleDateFormat("yyyyMMdd");

Date date = new Date();

try {

date = fmt.parse(code);

} catch (ParseException e) {

e.printStackTrace();

}

return date;

}

/\*

\* 根据字符串获取时间 字符串格式 yyyy-MM-dd

\*/

private Date getDate1(String code) {

DateFormat fmt = new SimpleDateFormat("yyyy-MM-dd");

Date date = new Date();

try {

date = fmt.parse(code);

} catch (ParseException e) {

e.printStackTrace();

}

return date;

}

/\*

\* 根据时间获取字符串 字符串格式 yyyyMMdd

\*/

private String getCode(Date date) {

SimpleDateFormat sdf = new SimpleDateFormat("yyyyMMdd");

return sdf.format(date);

}

/\*

\* 根据时间获取字符串 字符串格式 yyyy-MM-dd

\*/

private String getCode1(Date date) {

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");

return sdf.format(date);

}

/\*

\* 日期加一天

\*/

private Date plus(Date date) {

Calendar now = Calendar.getInstance();

now.setTime(date);

now.set(Calendar.DATE, now.get(Calendar.DATE) + 1);

return now.getTime();

}

/\*

\* 日期减一天

\*/

private Date minus(Date date) {

Calendar now = Calendar.getInstance();

now.setTime(date);

now.set(Calendar.DATE, now.get(Calendar.DATE) - 1);

return now.getTime();

}

/\*

\* 获取当前登录人所在科室的所有人 包括处级领导在内

\*/

private List<User> getUserList() {

List<User> uList = new ArrayList();

// 获取当前人所在部门编号

String departmentId = super.getVisit().getDepartment().getId();

// 获取与当前登录人同一部门的处级领导

List<User> uList1 = super.getDefaultManager().findAll(

"select d.user from DepartmentLeadership d "

+" where d.user.active=1 "

+" and d.user.userLevel>0 "

+" and d.department.id=? "

+" order by d.user.userLevel"

, departmentId);

// 遍历查询结果将用户加入用户集合，用于页面排序显示

for (int i=0; i<uList1.size(); i++) {

uList.add(uList1.get(i));

}

// 获取与当前登录人同一部门的科室成员

List<User> uList2 = super.getDefaultManager().findAll(

"from User u where u.active=1 "

+" and u.userLevel>0 "

+" and u.department.id=? "

+" order by u.userLevel", departmentId);

// 遍历查询结果将用户加入用户集合，用于页面排序显示

for (int i=0; i<uList2.size(); i++) {

uList.add(uList2.get(i));

}

return uList;

}

/\*

\* 根据是否是公务员的条件获取当前登录人所在科室的人

\*/

private List<User> getUserList(String civil) {

List<User> uList = new ArrayList();

// 获取当前人所在部门编号

String departmentId = super.getVisit().getDepartment().getId();

// 获取与当前登录人同一部门的处级领导

List<User> uList1 = super.getDefaultManager().findAll(

"select d.user from DepartmentLeadership d "

+" where d.user.active=1 "

+" and d.user.userLevel>0 "

+" and d.department.id=? "

+" and d.user.civil=? "

+" order by d.user.userLevel"

, departmentId, Integer.valueOf(civil));

// 遍历查询结果将用户加入用户集合，用于页面排序显示

for (int i=0; i<uList1.size(); i++) {

uList.add(uList1.get(i));

}

// 获取与当前登录人同一部门的科室成员

List<User> uList2 = super.getDefaultManager().findAll(

"from User u where u.active=1 "

+" and u.userLevel>0 "

+" and u.department.id=? "

+" and u.civil=? "

+" order by u.userLevel", departmentId, Integer.valueOf(civil));

// 遍历查询结果将用户加入用户集合，用于页面排序显示

for (int i=0; i<uList2.size(); i++) {

uList.add(uList2.get(i));

}

return uList;

}

/\*

\* 查询街道所有人

\*/

private List<User> getTreeUserList() {

//社保所属于事业单位，街道考核部计算在内

return super.getDefaultManager().findAll(

"from User u where u.active=1 and u.userLevel>0 "

//+" and u.department.name!='社保所' "

+" order by u.department.lineNumber,u.userLevel");

}

/\*

\* 根据考勤开始时间、结束时间查询考勤日期集合

\*/

private List<DayCode> getDayCodeList(String sCode, String eCode) {

return super.getDefaultManager().findAll("from DayCode d where d.id between ? and ? order by d.id", sCode, eCode);

}

/\*

\* 根据月份id查询考勤月天数

\*/

private long getDayCount(String ym) {

return super.getDefaultManager().getCount("from DayCode d where d.monthCode='"+ym+"'");

}

/\*

\* 查询所有请假类型

\*/

private List<AskForLeaveType> getAskTypeList() {

return super.getDefaultManager().findAll(

"from AskForLeaveType a order by a.leaveTypeId");

}

/\*

\* 根据考勤开始时间、结束时间、用户id查询请假信息

\*/

private List<AskForLeave> getAskForLeaveList(Date sDate, Date eDate, User user) {

return super.getDefaultManager().findAll(

"from AskForLeave a where (a.startTime between ? and ? " +

"or a.endTime between ? and ? or a.startTime < ? and a.endTime > ?) " +

"and a.leaveStatus.statusId in ('9999','3001') " +

"and a.appLeaveUser.id=?", sDate, eDate, sDate, eDate, sDate, eDate, user.getId());

}

/\*

\* 根据考勤开始时间、结束时间、用户id、请假类型查询某一类型请假信息

\*/

private List<AskForLeave> getAskForLeaveList(Date sDate, Date eDate, User user, AskForLeaveType askType) {

return super.getDefaultManager().findAll(

"from AskForLeave a where (a.startTime between ? and ? " +

"or a.endTime between ? and ? or a.startTime < ? and a.endTime > ?) " +

"and a.appLeaveUser.id=? and a.leaveType.leaveTypeId=? " +

"and a.leaveStatus.statusId in ('9999','3001') ",

sDate, eDate, sDate, eDate, sDate, eDate, user.getId(), askType.getLeaveTypeId());

}

/\*

\* 查询所有考勤类型

\*/

private List<CheckWorkType> getCheckWorkTypeList() {

return super.getDefaultManager().findAll(

"from CheckWorkType c where c.status=1 order by c.id");

}

/\*

\* 根据考勤开始时间、结束时间、用户id查询考勤信息

\*/

private List<UserCheckWork> getUserCheckWorkList(Date sDate, Date eDate, User user) {

return super.getDefaultManager().findAll(

"from UserCheckWork a where a.checkDate between ? and ? " +

"and a.status=1 and a.user.id=?", sDate, eDate, user.getId());

}

/\*

\* 根据考勤开始时间、结束时间、用户id、类型id查询考勤信息

\*/

private List<UserCheckWork> getUserCheckWorkList(Date sDate, Date eDate, User user, CheckWorkType type) {

return super.getDefaultManager().findAll(

"from UserCheckWork a where a.checkDate between ? and ? " +

"and a.status=1 and a.user.id=? and a.checkWorkType.id=?", sDate, eDate, user.getId(), type.getId());

}

/\*

\* 统计请假天数

\*/

private int getSum(AskForLeave ask, String askType, Date sDate, Date eDate) {

int sum = 0;

int count = 0;

// 请假开始时间

Date dStart = ask.getStartTime();

// 请假结束时间

Date dEnd = ask.getEndTime();

// 如果请假开始时间小于统计开始时间

while (dStart.getTime() < sDate.getTime()) {

String code = getCode(dStart);

String dayType = (String) super.getDefaultManager().getObject("select d.dayType from DayCode d where d.id='"+code+"'");

if (askType.equals("探亲假") || askType.equals("产假")) {

++sum;

} else if (dayType.equals("班")){

++sum;

}

dStart = plus(dStart);

}

// 如果请假结束时间大于统计结束时间

while (dEnd.getTime() > eDate.getTime()) {

String code = getCode(dEnd);

String dayType = (String) super.getDefaultManager().getObject("select d.dayType from DayCode d where d.id='"+code+"'");

if (askType.equals("探亲假") || askType.equals("产假")) {

++sum;

} else if (dayType.equals("班")){

++sum;

}

dEnd = minus(dEnd);

}

return count += Integer.valueOf(ask.getDayNum()) - sum;

}

}

package com.bizlogic.oa.action;

import java.io.IOException;

import java.io.PrintWriter;

import java.text.DateFormat;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.ArrayList;

import java.util.Calendar;

import java.util.Date;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import org.hibernate.Hibernate;

import org.hibernate.type.Type;

import com.alibaba.fastjson.JSON;

import com.alibaba.fastjson.serializer.SerializerFeature;

import com.bizlogic.base.Department;

import com.bizlogic.base.User;

import com.bizlogic.base.front.action.BaseUserListAction;

import com.bizlogic.common.action.helper.ListInfo;

import com.bizlogic.oa.bean.AskForCar;

import com.bizlogic.oa.bean.AskForCarStatus;

import com.bizlogic.oa.bean.AskForLeave;

import com.bizlogic.oa.bean.AskForLeaveHomeType;

import com.bizlogic.oa.bean.AskForLeaveStatus;

import com.bizlogic.oa.bean.AskForLeaveType;

import com.opensymphony.xwork2.ModelDriven;

public class AskForCarAction extends BaseUserListAction implements ModelDriven{

AskForCar po=new AskForCar();

private String ids[];

public String[] getIds() {

return ids;

}

public void setIds(String[] ids) {

this.ids = ids;

}

List paramList = new ArrayList();

@Override

protected long doGetCount(ListInfo listInfo) {

return super.getDefaultManager().getCount("from deptinfo t ");

}

protected List doGetCount() {

return super.getDefaultManager().getCountSQL(

"select a.use\_carapp\_id as useCarAppId," +

"a.user\_num as userNum,"+

"a.send\_status as sendStatus,"+

"a.description as description,"+

"a.need\_driver as needDriver,"+

"a.start\_time as startTime,"+

"a.return\_time as returnTime,"+

"a.send\_car\_time as sendCarTime,"+

"a.return\_remarks as returnRemarks,"+

"a.section\_chief\_status as sectionChiefStatus,"+

"a.section\_chief\_time as sectionChiefTime,"+

"a.section\_remarks as sectionRemarks,"+

"a.create\_date as createDate,"+

"a.driver\_id as driverId,"+

"h.driver\_name as driverName,"+

"a.car\_id as carId,"+

"i.car\_name as carName,"+

"i.car\_code as carCode,"+

"b.user\_id as useCarAppUserId,"+

"b.name as useCarAppUserName,"+

"c.user\_id as sectionChiefId,"+

"c.name as sectionChiefName,"+

"d.user\_id as sendCarUserId,"+

"d.name as sendCarUserName,"+

"e.user\_id as createUserId,"+

"e.name as createUserName,"+

"f.department\_id as departmentId,"+

"f.name as departmentName,"+

"g.status\_id as statusId,"+

"g.name as statusName "+

"from t\_oa\_usecarapp a left join t\_oa\_car\_driver h on a.driver\_id=h.user\_id left join t\_oa\_car i on a.car\_id=i.car\_id left join t\_bpm\_user d on a.send\_car\_user\_id=d.user\_id,t\_bpm\_user b ,t\_bpm\_user c ,t\_bpm\_user e,t\_bpm\_department f,t\_oa\_usecar\_status g "+

"where a.user\_id=b.user\_id and a.section\_chief\_id=c.user\_id and a.create\_user\_id=e.user\_id and a.department\_id=f.department\_id and a.status\_id=g.status\_id"

+sqlCondition

,paramList.toArray());

}

@Override

protected List doSearch(ListInfo listInfo) {

List list = super.getDefaultManager().find("from OAResource t where t.active=1 order by t.id",

listInfo.getFirstNo(), listInfo.getMaxNo());

return list;

}

//返回到list页面

public String getCarAppList() throws IOException

{

int aa=this.doGetCount().size();

super.getResponse().setCharacterEncoding("utf-8");

sqlCondition = this.getConditions(po);

String result=toJSONStringByFastjson(doGetCount().size(),this.getList());

System.out.println(result);

PrintWriter out = this.getResponse().getWriter();

out.println(result);

out.flush();

out.close();

return "toList";

}

public List getList(){

List<AskForCar> deptListpo=new ArrayList<AskForCar>();

String propery[]={"useCarAppId","userNum", "sendStatus", "description", "needDriver", "startTime", "returnTime", "sendCarTime", "returnRemarks", "sectionChiefStatus","sectionChiefTime", "sectionRemarks", "createDate", "driverId", "driverName", "carId", "carName", "carCode", "useCarAppUserId", "useCarAppUserName","sectionChiefId","sectionChiefName","sendCarUserId","sendCarUserName","createUserId","createUserName","departmentId","departmentName","statusId","statusName"};

Type type[]={Hibernate.STRING, Hibernate.INTEGER ,Hibernate.INTEGER ,Hibernate.STRING,Hibernate.INTEGER ,Hibernate.TIMESTAMP,Hibernate.TIMESTAMP,Hibernate.TIMESTAMP, Hibernate.STRING, Hibernate.STRING, Hibernate.TIMESTAMP,Hibernate.STRING, Hibernate.TIMESTAMP,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING, Hibernate.STRING, Hibernate.STRING, Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING};

deptListpo = super.getDefaultManager().findSQLpo(

"select a.use\_carapp\_id as useCarAppId," +

"a.user\_num as userNum,"+

"a.send\_status as sendStatus,"+

"a.description as description,"+

"a.need\_driver as needDriver,"+

"a.start\_time as startTime,"+

"a.return\_time as returnTime,"+

"a.send\_car\_time as sendCarTime,"+

"a.return\_remarks as returnRemarks,"+

"a.section\_chief\_status as sectionChiefStatus,"+

"a.section\_chief\_time as sectionChiefTime,"+

"a.section\_remarks as sectionRemarks,"+

"a.create\_date as createDate,"+

"a.driver\_id as driverId,"+

"h.driver\_name as driverName,"+

"a.car\_id as carId,"+

"i.car\_name as carName,"+

"i.car\_code as carCode,"+

"b.user\_id as useCarAppUserId,"+

"b.name as useCarAppUserName,"+

"c.user\_id as sectionChiefId,"+

"c.name as sectionChiefName,"+

"d.user\_id as sendCarUserId,"+

"d.name as sendCarUserName,"+

"e.user\_id as createUserId,"+

"e.name as createUserName,"+

"f.department\_id as departmentId,"+

"f.name as departmentName,"+

"g.status\_id as statusId,"+

"g.name as statusName "+

"from t\_oa\_usecarapp a left join t\_oa\_car\_driver h on a.driver\_id=h.user\_id left join t\_oa\_car i on a.car\_id=i.car\_id left join t\_bpm\_user d on a.send\_car\_user\_id=d.user\_id,t\_bpm\_user b ,t\_bpm\_user c ,t\_bpm\_user e,t\_bpm\_department f,t\_oa\_usecar\_status g "+

"where a.user\_id=b.user\_id and a.section\_chief\_id=c.user\_id and a.create\_user\_id=e.user\_id and a.department\_id=f.department\_id and a.status\_id=g.status\_id"

+sqlCondition,super.begin,super.end,propery,type,AskForCar.class,paramList.toArray());

return deptListpo;

}

//这是各级别人员看到自己列表的查询条件。

protected String getConditions(AskForCar po)

{

User u=(User)super.getSession().getAttribute("visituser");

String conditionsStr = "";

if(u.getUserLevel()==4||u.getUserLevel()==5){//科员/副科长级别 只能查到能查到自己请的假。

if(u.getId()!=null && !u.getId().equals("")){

conditionsStr += " and a.user\_id = ? and a.status\_id not in(0000)";

paramList.add(u.getId());

}

}else if(u.getUserLevel()==3){//科长级别 能查到让自己审批的和自己请的假，并且看不到真正编辑的请假单。

if(u.getId()!=null && !u.getId().equals("")){

if(this.getParameterValue("formy")!=null){

if(this.getParameterValue("formy").equals("formy")){

conditionsStr += " and a.SECTION\_CHIEF\_ID = ? and a.user\_id !=? and a.status\_id not in(0000,1001)";

paramList.add(u.getId());

paramList.add(u.getId());

}else if(this.getParameterValue("formy").equals("my")){

conditionsStr += " and a.user\_id = ? and a.status\_id not in(0000)";

paramList.add(u.getId());

}else if(this.getParameterValue("formy").equals("all")){

conditionsStr += " and a.status\_id in(9999)";

}

}

}

}

@Override

protected long doGetCount(ListInfo listInfo) {

return super.getDefaultManager().getCount("from deptinfo t ");

}

protected List doGetCount() {

return super.getDefaultManager().getCountSQL(

"select a.car\_id as carId," +

"a.car\_name as carName,"+

"a.car\_code as carCode,"+

"a.most\_num as mostNum,"+

"a.status as status,"+

"a.create\_date as createDate,"+

"a.remark as remark,"+

"a.create\_user\_id as createUserId,"+

"a.status\_id as statusId, "+

"a.user\_id as driverId, "+

"d.driver\_name as driverName "+

"from t\_oa\_car a ,t\_bpm\_user b,t\_oa\_car\_use\_status c ,t\_oa\_car\_driver d "+

"where a.create\_user\_id=b.user\_id and a.status\_id=c.status\_id and a.user\_id=d.user\_id "

+sqlCondition

,paramList.toArray());

}

//返回到list页面

public String getCarList() throws IOException

{

int aa=this.doGetCount().size();

super.getResponse().setCharacterEncoding("utf-8");

sqlCondition = this.getConditions(po);

String result=toJSONStringByFastjson(doGetCount().size(),this.getList());

System.out.println(result);

PrintWriter out = this.getResponse().getWriter();

out.println(result);

out.flush();

out.close();

return "toList";

}

public List getList(){

List<Car> carListpo=new ArrayList<Car>();

String propery[]={"carId", "carName", "carCode", "mostNum", "status", "createDate", "remark","createUserId" ,"statusId","driverId","driverName"};

Type type[]={Hibernate.STRING, Hibernate.STRING ,Hibernate.STRING,Hibernate.INTEGER ,Hibernate.INTEGER ,Hibernate.TIMESTAMP,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING};

carListpo = super.getDefaultManager().findSQLpo(

"select a.car\_id as carId," +

"a.car\_name as carName,"+

"a.car\_code as carCode,"+

"a.most\_num as mostNum,"+

"a.status as status,"+

"a.create\_date as createDate,"+

"a.remark as remark,"+

"a.create\_user\_id as createUserId,"+

"a.status\_id as statusId, "+

"a.user\_id as driverId, "+

"d.driver\_name as driverName "+

"from t\_oa\_car a ,t\_bpm\_user b,t\_oa\_car\_use\_status c ,t\_oa\_car\_driver d "+

"where a.create\_user\_id=b.user\_id and a.status\_id=c.status\_id and a.user\_id=d.user\_id "

+sqlCondition,super.begin,super.end,propery,type,Car.class,paramList.toArray());

return carListpo;

}

@Override

protected List doSearch(ListInfo listInfo) {

List list = super.getDefaultManager().find("from OAResource t where t.active=1 order by t.id",

listInfo.getFirstNo(), listInfo.getMaxNo());

return list;

}

//这个是请假申请单id的查询条件

if(po.getUseCarAppId()!=null && !po.getUseCarAppId().equals("")){

conditionsStr += " and a.use\_carapp\_id = ?";

paramList.add(po.getUseCarAppId());

}

if(po.getUseCarAppUserName()!=null && !po.getUseCarAppUserName().equals("")){

conditionsStr += " and b.name like ?";

paramList.add("%"+po.getUseCarAppUserName()+"%");

}

if(po.getUseCarAppUserName1()!=null && !po.getUseCarAppUserName1().equals("")){

conditionsStr += " and b.name like ?";

paramList.add("%"+po.getUseCarAppUserName1()+"%");

}

if(po.getSectionChiefStatus()!=null && !po.getSectionChiefStatus().equals("")){

conditionsStr += " and a.section\_chief\_status = ?";

paramList.add(po.getSectionChiefStatus());

}

return conditionsStr;

}

public static String toJSONStringByFastjson(int total, List rows)

{

Map jsonMap = new HashMap();

jsonMap.put("total", Integer.valueOf(total));

jsonMap.put("rows", rows);

String result=JSON.toJSONStringWithDateFormat(jsonMap, "yyyy-MM-dd HH:mm");

return result;

}

//去添加页面。

public String toAddCarApp(){

User u=(User)super.getSession().getAttribute("visituser");

User user2=new User();

if(u.getUserLevel()==3){

//科长申请车时，审批人是他本人。

user2=(User)super.getDefaultManager().getObject("from User u where u.id='"+u.getId()+"'");

}else if(u.getUserLevel()==4||u.getUserLevel()==5){

//科员申请车时，审批人是他的上级（也就是他的科长）

user2=(User)super.getDefaultManager().getObject("from User u where u.id='"+u.getParentUser().getId()+"'");//科长

}

this.getRequest().setAttribute("user2", user2);

return "toAdd";

}

//去list页面。

public String carAppList(){

return "toList";

}

//保存申请

public void saveCarApp() throws IOException{

String sbtype="";

Date now = new Date();

User u=(User)super.getSession().getAttribute("visituser");

if(u.getUserLevel()==2||u.getUserLevel()==3){

po.setSectionChiefStatus("1");//科长/主管主任申请假时，自动审批成功。

po.setSectionChiefTime(now);

}else{

po.setSectionChiefStatus("0");

}

User AppCarUser=super.getDefaultManager().get(User.class,po.getUseCarAppUserId());

User CreateUser=super.getDefaultManager().get(User.class,po.getCreateUserId());

User SectionChief=super.getDefaultManager().get(User.class,po.getSectionChiefId());

Department dept=super.getDefaultManager().get(Department.class,po.getDepartmentId());

AskForCarStatus appCarStatus=super.getDefaultManager().get(AskForCarStatus.class,"2001");//科长审批状态

AskForCarStatus appCarStatus1=super.getDefaultManager().get(AskForCarStatus.class,"1001");//编辑状态

AskForCarStatus appCarStatus2=super.getDefaultManager().get(AskForCarStatus.class,"9999");//编辑状态

SimpleDateFormat formatDate = new SimpleDateFormat("yy-MM-dd HH:mm:ss");

//String str = formatDate.format(po.getsTime());

Date time=null;

try {

time = formatDate.parse(po.getsTime()+":00");

}catch(ParseException e) {

e.printStackTrace();

}

po.setStartTime(time);

po.setAppCarUser(AppCarUser);//保存申请人

po.setCreateUser(CreateUser);//保存创建人

po.setSectionChief(SectionChief);//保存科长

po.setDept(dept);//保存部门

if(po.getType().equals("1")){//各个级别的人员保存时。请假单都为编辑状态1001。

sbtype="b";

po.setAppCarStatus(appCarStatus1);

}else if(po.getType().equals("2")){//各个级别人员保存并提交时。

sbtype="a";

if(u.getUserLevel()==2||u.getUserLevel()==3){

po.setAppCarStatus(appCarStatus2);

}else{

po.setAppCarStatus(appCarStatus);

}

}

po.setCreateDate(now);//保存创建日期

super.getDefaultManager().save(po);//保存请假申请单

PrintWriter out = this.getResponse().getWriter();

out.println(sbtype);

out.flush();

out.close();

}

//去查看页面

public String toDetailCarApp(){

String aa=null;

if(this.getParameterValue("t").equals("1")){//查看草稿箱

}else{//查看

aa=this.getConditions(po);

}

String propery[]={"useCarAppId","userNum", "sendStatus", "description", "needDriver", "startTime", "returnTime", "sendCarTime", "returnRemarks", "sectionChiefStatus","sectionChiefTime", "sectionRemarks", "createDate", "driverId", "driverName", "carId", "carName", "carCode", "useCarAppUserId", "useCarAppUserName","sectionChiefId","sectionChiefName","sendCarUserId","sendCarUserName","createUserId","createUserName","departmentId","departmentName","statusId","statusName"};

Type type[]={Hibernate.STRING, Hibernate.INTEGER ,Hibernate.INTEGER,Hibernate.STRING,Hibernate.INTEGER ,Hibernate.TIMESTAMP,Hibernate.TIMESTAMP,Hibernate.TIMESTAMP, Hibernate.STRING, Hibernate.STRING, Hibernate.TIMESTAMP,Hibernate.STRING, Hibernate.TIMESTAMP,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING, Hibernate.STRING, Hibernate.STRING, Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING};

List<AskForCar> AskForCarpo=new ArrayList<AskForCar>();

AskForCarpo=super.getDefaultManager().findSQLpo(

"select a.use\_carapp\_id as useCarAppId," +

"a.user\_num as userNum,"+

"a.send\_status as sendStatus,"+

"a.description as description,"+

"a.need\_driver as needDriver,"+

"a.start\_time as startTime,"+

"a.return\_time as returnTime,"+

"a.send\_car\_time as sendCarTime,"+

"a.return\_remarks as returnRemarks,"+

"a.section\_chief\_status as sectionChiefStatus,"+

"a.section\_chief\_time as sectionChiefTime,"+

"a.section\_remarks as sectionRemarks,"+

"a.create\_date as createDate,"+

"a.driver\_id as driverId,"+

"h.driver\_name as driverName,"+

"a.car\_id as carId,"+

"i.car\_name as carName,"+

"i.car\_code as carCode,"+

"b.user\_id as useCarAppUserId,"+

"b.name as useCarAppUserName,"+

"c.user\_id as sectionChiefId,"+

"c.name as sectionChiefName,"+

"d.user\_id as sendCarUserId,"+

"d.name as sendCarUserName,"+

"e.user\_id as createUserId,"+

"e.name as createUserName,"+

"f.department\_id as departmentId,"+

"f.name as departmentName,"+

"g.status\_id as statusId,"+

"g.name as statusName "+

"from t\_oa\_usecarapp a left join t\_oa\_car\_driver h on a.driver\_id=h.user\_id left join t\_oa\_car i on a.car\_id=i.car\_id left join t\_bpm\_user d on a.send\_car\_user\_id=d.user\_id,t\_bpm\_user b ,t\_bpm\_user c ,t\_bpm\_user e,t\_bpm\_department f,t\_oa\_usecar\_status g "+

"where a.user\_id=b.user\_id and a.section\_chief\_id=c.user\_id and a.create\_user\_id=e.user\_id and a.department\_id=f.department\_id and a.status\_id=g.status\_id"

+aa,super.begin,super.end,propery,type,AskForCar.class,paramList.toArray());

this.getRequest().setAttribute("carappdetail", AskForCarpo.get(0));

return "toDetail";

}

//去更新页面

public String toUpdateCarApp(){

String aa=null;

aa=this.getConditions(po);//用的是查询list的条件。

User u=(User)super.getSession().getAttribute("visituser");

User user2=new User();

if(u.getUserLevel()==3){

user2=(User)super.getDefaultManager().getObject("from User u where u.id='"+u.getId()+"'");

}else if(u.getUserLevel()==4||u.getUserLevel()==5){

user2=(User)super.getDefaultManager().getObject("from User u where u.id='"+u.getParentUser().getId()+"'");

}

this.getRequest().setAttribute("user2", user2);

String propery[]={"useCarAppId","userNum", "sendStatus", "description", "needDriver", "startTime", "returnTime", "sendCarTime", "returnRemarks", "sectionChiefStatus","sectionChiefTime", "sectionRemarks", "createDate", "driverId", "driverName", "carId", "carName", "carCode", "useCarAppUserId", "useCarAppUserName","sectionChiefId","sectionChiefName","sendCarUserId","sendCarUserName","createUserId","createUserName","departmentId","departmentName","statusId","statusName"};

Type type[]={Hibernate.STRING, Hibernate.INTEGER ,Hibernate.INTEGER,Hibernate.STRING,Hibernate.INTEGER ,Hibernate.TIMESTAMP,Hibernate.TIMESTAMP,Hibernate.TIMESTAMP, Hibernate.STRING, Hibernate.STRING, Hibernate.TIMESTAMP,Hibernate.STRING, Hibernate.TIMESTAMP,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING, Hibernate.STRING, Hibernate.STRING, Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING};

List<AskForCar> AskForCarpo=new ArrayList<AskForCar>();

AskForCarpo=super.getDefaultManager().findSQLpo(

"select a.use\_carapp\_id as useCarAppId," +

"a.user\_num as userNum,"+

"a.send\_status as sendStatus,"+

"a.description as description,"+

"a.need\_driver as needDriver,"+

"a.start\_time as startTime,"+

"a.return\_time as returnTime,"+

"a.send\_car\_time as sendCarTime,"+

"a.return\_remarks as returnRemarks,"+

"a.section\_chief\_status as sectionChiefStatus,"+

"a.section\_chief\_time as sectionChiefTime,"+

"a.section\_remarks as sectionRemarks,"+

"a.create\_date as createDate,"+

"a.driver\_id as driverId,"+

"h.driver\_name as driverName,"+

"a.car\_id as carId,"+

"i.car\_name as carName,"+

"i.car\_code as carCode,"+

"b.user\_id as useCarAppUserId,"+

"b.name as useCarAppUserName,"+

"c.user\_id as sectionChiefId,"+

"c.name as sectionChiefName,"+

"d.user\_id as sendCarUserId,"+

"d.name as sendCarUserName,"+

"e.user\_id as createUserId,"+

"e.name as createUserName,"+

"f.department\_id as departmentId,"+

"f.name as departmentName,"+

"g.status\_id as statusId,"+

"g.name as statusName "+

"from t\_oa\_usecarapp a left join t\_oa\_car\_driver h on a.driver\_id=h.user\_id left join t\_oa\_car i on a.car\_id=i.car\_id left join t\_bpm\_user d on a.send\_car\_user\_id=d.user\_id,t\_bpm\_user b ,t\_bpm\_user c ,t\_bpm\_user e,t\_bpm\_department f,t\_oa\_usecar\_status g "+

"where a.user\_id=b.user\_id and a.section\_chief\_id=c.user\_id and a.create\_user\_id=e.user\_id and a.department\_id=f.department\_id and a.status\_id=g.status\_id"

+aa,super.begin,super.end,propery,type,AskForCar.class,paramList.toArray());

this.getRequest().setAttribute("carappdetail", AskForCarpo.get(0));

return "toUpdate";

}

//草稿保存、保存并提交时

public void saveSub(){

Date now = new Date();

User u=(User)super.getSession().getAttribute("visituser");

if(u.getUserLevel()==2||u.getUserLevel()==3){

po.setSectionChiefStatus("1");//科长/主管主任申请假时，自动审批成功。

po.setSectionChiefTime(now);

}else{

po.setSectionChiefStatus("0");

}

User AppCarUser=super.getDefaultManager().get(User.class,po.getUseCarAppUserId());

User CreateUser=super.getDefaultManager().get(User.class,po.getCreateUserId());

User SectionChief=super.getDefaultManager().get(User.class,po.getSectionChiefId());

Department dept=super.getDefaultManager().get(Department.class,po.getDepartmentId());

AskForCarStatus appCarStatus=super.getDefaultManager().get(AskForCarStatus.class,"2001");//科长审批状态

AskForCarStatus appCarStatus1=super.getDefaultManager().get(AskForCarStatus.class,"1001");//编辑状态

AskForCarStatus appCarStatus2=super.getDefaultManager().get(AskForCarStatus.class,"9999");//编辑状态

po.setAppCarUser(AppCarUser);//保存申请人

po.setCreateUser(CreateUser);//保存创建人

po.setSectionChief(SectionChief);//保存科长

po.setDept(dept);//保存部门

if(po.getType().equals("1")){//各个级别的人员保存时。请假单都为编辑状态1001。

po.setAppCarStatus(appCarStatus1);

}else if(po.getType().equals("2")){//各个级别人员保存并提交时。

if(u.getUserLevel()==2||u.getUserLevel()==3){

po.setAppCarStatus(appCarStatus2);

}else{

po.setAppCarStatus(appCarStatus);

}

}

po.setCreateDate(now);//保存创建日期

super.getDefaultManager().update(po);

}

//删除

public String deleteCarApp(){

AskForCar[] pos = new AskForCar[ids.length];

for (int i = 0; i < pos.length; i++) {

pos[i] = new AskForCar();

pos[i].setUseCarAppId(ids[i]);

}

for (AskForCar po : pos) {

super.getDefaultManager().executeSQL("update t\_oa\_usecarapp set status\_id=? where use\_carapp\_id=?","0000",po.getUseCarAppId());

}

return "toList";

}

//去打印页面

@SuppressWarnings("unchecked")

public String toPrintCarApp(){

String aa=this.getConditions(po);

String propery[]={"useCarAppId","userNum", "sendStatus", "description", "needDriver", "startTime", "returnTime", "sendCarTime", "returnRemarks", "sectionChiefStatus","sectionChiefTime", "sectionRemarks", "createDate", "driverId", "driverName", "carId", "carName", "carCode", "useCarAppUserId", "useCarAppUserName","sectionChiefId","sectionChiefName","sendCarUserId","sendCarUserName","createUserId","createUserName","departmentId","departmentName","statusId","statusName"};

Type type[]={Hibernate.STRING, Hibernate.INTEGER ,Hibernate.INTEGER,Hibernate.STRING,Hibernate.INTEGER ,Hibernate.TIMESTAMP,Hibernate.TIMESTAMP,Hibernate.TIMESTAMP, Hibernate.STRING, Hibernate.STRING, Hibernate.TIMESTAMP,Hibernate.STRING, Hibernate.TIMESTAMP,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING, Hibernate.STRING, Hibernate.STRING, Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING};

List<AskForCar> AskForCarpo=new ArrayList<AskForCar>();

AskForCarpo=super.getDefaultManager().findSQLpo(

"select a.use\_carapp\_id as useCarAppId," +

"a.user\_num as userNum,"+

"a.send\_status as sendStatus,"+

"a.description as description,"+

"a.need\_driver as needDriver,"+

"a.start\_time as startTime,"+

"a.return\_time as returnTime,"+

"a.send\_car\_time as sendCarTime,"+

"a.return\_remarks as returnRemarks,"+

"a.section\_chief\_status as sectionChiefStatus,"+

"a.section\_chief\_time as sectionChiefTime,"+

"a.section\_remarks as sectionRemarks,"+

"a.create\_date as createDate,"+

"a.driver\_id as driverId,"+

"h.driver\_name as driverName,"+

"a.car\_id as carId,"+

"i.car\_name as carName,"+

"i.car\_code as carCode,"+

"b.user\_id as useCarAppUserId,"+

"b.name as useCarAppUserName,"+

"c.user\_id as sectionChiefId,"+

"c.name as sectionChiefName,"+

"d.user\_id as sendCarUserId,"+

"d.name as sendCarUserName,"+

"e.user\_id as createUserId,"+

"e.name as createUserName,"+

"f.department\_id as departmentId,"+

"f.name as departmentName,"+

"g.status\_id as statusId,"+

"g.name as statusName "+

"from t\_oa\_usecarapp a left join t\_oa\_car\_driver h on a.driver\_id=h.user\_id left join t\_oa\_car i on a.car\_id=i.car\_id left join t\_bpm\_user d on a.send\_car\_user\_id=d.user\_id,t\_bpm\_user b ,t\_bpm\_user c ,t\_bpm\_user e,t\_bpm\_department f,t\_oa\_usecar\_status g "+

"where a.user\_id=b.user\_id and a.section\_chief\_id=c.user\_id and a.create\_user\_id=e.user\_id and a.department\_id=f.department\_id and a.status\_id=g.status\_id"

+aa,super.begin,super.end,propery,type,AskForCar.class,paramList.toArray());

this.getRequest().setAttribute("carappdetail", AskForCarpo.get(0));

return "toPrint";

}

//去审批页面

public String toApprovalCarApp(){

String aa=null;

aa=this.getConditions(po);

String propery[]={"useCarAppId","userNum", "sendStatus", "description", "needDriver", "startTime", "returnTime", "sendCarTime", "returnRemarks", "sectionChiefStatus","sectionChiefTime", "sectionRemarks", "createDate", "driverId", "driverName", "carId", "carName", "carCode", "useCarAppUserId", "useCarAppUserName","sectionChiefId","sectionChiefName","sendCarUserId","sendCarUserName","createUserId","createUserName","departmentId","departmentName","statusId","statusName"};

Type type[]={Hibernate.STRING, Hibernate.INTEGER ,Hibernate.INTEGER ,Hibernate.STRING,Hibernate.INTEGER ,Hibernate.TIMESTAMP,Hibernate.TIMESTAMP,Hibernate.TIMESTAMP, Hibernate.STRING, Hibernate.STRING, Hibernate.TIMESTAMP,Hibernate.STRING, Hibernate.TIMESTAMP,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING, Hibernate.STRING, Hibernate.STRING, Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING};

List<AskForCar> AskForCarpo=new ArrayList<AskForCar>();

AskForCarpo=super.getDefaultManager().findSQLpo(

"select a.use\_carapp\_id as useCarAppId," +

"a.user\_num as userNum,"+

"a.send\_status as sendStatus,"+

"a.description as description,"+

"a.need\_driver as needDriver,"+

"a.start\_time as startTime,"+

"a.return\_time as returnTime,"+

"a.send\_car\_time as sendCarTime,"+

"a.return\_remarks as returnRemarks,"+

"a.section\_chief\_status as sectionChiefStatus,"+

"a.section\_chief\_time as sectionChiefTime,"+

"a.section\_remarks as sectionRemarks,"+

"a.create\_date as createDate,"+

"a.driver\_id as driverId,"+

"h.driver\_name as driverName,"+

"a.car\_id as carId,"+

"i.car\_name as carName,"+

"i.car\_code as carCode,"+

"b.user\_id as useCarAppUserId,"+

"b.name as useCarAppUserName,"+

"c.user\_id as sectionChiefId,"+

"c.name as sectionChiefName,"+

"d.user\_id as sendCarUserId,"+

"d.name as sendCarUserName,"+

"e.user\_id as createUserId,"+

"e.name as createUserName,"+

"f.department\_id as departmentId,"+

"f.name as departmentName,"+

"g.status\_id as statusId,"+

"g.name as statusName "+

"from t\_oa\_usecarapp a left join t\_oa\_car\_driver h on a.driver\_id=h.user\_id left join t\_oa\_car i on a.car\_id=i.car\_id left join t\_bpm\_user d on a.send\_car\_user\_id=d.user\_id,t\_bpm\_user b ,t\_bpm\_user c ,t\_bpm\_user e,t\_bpm\_department f,t\_oa\_usecar\_status g "+

"where a.user\_id=b.user\_id and a.section\_chief\_id=c.user\_id and a.create\_user\_id=e.user\_id and a.department\_id=f.department\_id and a.status\_id=g.status\_id"

+aa,super.begin,super.end,propery,type,AskForCar.class,paramList.toArray());

this.getRequest().setAttribute("carappdetail", AskForCarpo.get(0));

return "toApproval";

}

//去派车页面

public String toSendCarApp(){

String aa=null;

aa=this.getConditions(po);

String propery[]={"useCarAppId","userNum", "sendStatus", "description", "needDriver", "startTime", "returnTime", "sendCarTime", "returnRemarks", "sectionChiefStatus","sectionChiefTime", "sectionRemarks", "createDate", "driverId", "driverName", "carId", "carName", "carCode", "useCarAppUserId", "useCarAppUserName","sectionChiefId","sectionChiefName","sendCarUserId","sendCarUserName","createUserId","createUserName","departmentId","departmentName","statusId","statusName"};

Type type[]={Hibernate.STRING, Hibernate.INTEGER ,Hibernate.INTEGER,Hibernate.STRING,Hibernate.INTEGER ,Hibernate.TIMESTAMP,Hibernate.TIMESTAMP,Hibernate.TIMESTAMP, Hibernate.STRING, Hibernate.STRING, Hibernate.TIMESTAMP,Hibernate.STRING, Hibernate.TIMESTAMP,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING, Hibernate.STRING, Hibernate.STRING, Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,Hibernate.STRING};

List<AskForCar> AskForCarpo=new ArrayList<AskForCar>();

AskForCarpo=super.getDefaultManager().findSQLpo(

"select a.use\_carapp\_id as useCarAppId," +

"a.user\_num as userNum,"+

"a.send\_status as sendStatus,"+

"a.description as description,"+

"a.need\_driver as needDriver,"+

"a.start\_time as startTime,"+

"a.return\_time as returnTime,"+

"a.send\_car\_time as sendCarTime,"+

"a.return\_remarks as returnRemarks,"+

"a.section\_chief\_status as sectionChiefStatus,"+

"a.section\_chief\_time as sectionChiefTime,"+

"a.section\_remarks as sectionRemarks,"+

"a.create\_date as createDate,"+

"a.driver\_id as driverId,"+

"h.driver\_name as driverName,"+

"a.car\_id as carId,"+

"i.car\_name as carName,"+

"i.car\_code as carCode,"+

"b.user\_id as useCarAppUserId,"+

"b.name as useCarAppUserName,"+

"c.user\_id as sectionChiefId,"+

"c.name as sectionChiefName,"+

"d.user\_id as sendCarUserId,"+

"d.name as sendCarUserName,"+

"e.user\_id as createUserId,"+

"e.name as createUserName,"+

"f.department\_id as departmentId,"+

"f.name as departmentName,"+

"g.status\_id as statusId,"+

"g.name as statusName "+

"from t\_oa\_usecarapp a left join t\_oa\_car\_driver h on a.driver\_id=h.user\_id left join t\_oa\_car i on a.car\_id=i.car\_id left join t\_bpm\_user d on a.send\_car\_user\_id=d.user\_id,t\_bpm\_user b ,t\_bpm\_user c ,t\_bpm\_user e,t\_bpm\_department f,t\_oa\_usecar\_status g "+

"where a.user\_id=b.user\_id and a.section\_chief\_id=c.user\_id and a.create\_user\_id=e.user\_id and a.department\_id=f.department\_id and a.status\_id=g.status\_id"

+aa,super.begin,super.end,propery,type,AskForCar.class,paramList.toArray());

this.getRequest().setAttribute("carappdetail", AskForCarpo.get(0));

return "toSendCarApp";

}

//派车操作

public void sendCar(){

//更新

super.getDefaultManager().executeSQL("update t\_oa\_usecarapp set send\_status=?,driver\_Id=?,car\_Id=?,send\_car\_user\_id=? where use\_carapp\_id=?","1",po.getDriverId(),po.getCarId(),po.getSendCarUserId(),po.getUseCarAppId());

//设置车辆状态

super.getDefaultManager().executeSQL("update t\_oa\_car set STATUS\_ID=? where car\_id=?","0001",po.getCarId());

//设置司机状态

super.getDefaultManager().executeSQL("update t\_oa\_car\_driver set STATUS\_ID=? where user\_id=?","0001",po.getDriverId());

}

//审批用车申请

public void approval(){

Date now = new Date();

User u=(User)super.getSession().getAttribute("visituser");

if(po.getSectionChiefStatus().equals("1")){//审批通过

super.getDefaultManager().executeSQL("update t\_oa\_usecarapp set status\_id=?, section\_chief\_time=?,section\_chief\_status=?,section\_remarks=? where use\_carapp\_id=?","9999",now,po.getSectionChiefStatus(),po.getSectionRemarks(),po.getUseCarAppId());

}else if(po.getSectionChiefStatus().equals("2")){//审批不通过

super.getDefaultManager().executeSQL("update t\_oa\_usecarapp set status\_id=?, section\_chief\_time=?,section\_chief\_status=?,section\_remarks=? where use\_carapp\_id=?","1002",now,po.getSectionChiefStatus(),po.getSectionRemarks(),po.getUseCarAppId());

}

}

//交车操作

public void returnCarApp(){

Date now = new Date();

super.getDefaultManager().executeSQL("update t\_oa\_usecarapp set send\_status=?,return\_time=? where use\_carapp\_id=?","2",now,po.getUseCarAppId());

//设置车辆状态

super.getDefaultManager().executeSQL("update t\_oa\_car set STATUS\_ID=? where car\_id=?","0000",po.getCarId());

//设置司机状态

super.getDefaultManager().executeSQL("update t\_oa\_car\_driver set STATUS\_ID=? where user\_id=?","0000",po.getDriverId());

}

//获得待审批的请假单的数量。

public void getAppNum() throws IOException{

String cond=this.getAppNumCondition();

List list=super.getDefaultManager().getCountSQL(

"select a.use\_carapp\_id as useCarAppId," +

"a.user\_num as userNum,"+

"a.send\_status as sendStatus,"+

"a.description as description,"+

"a.need\_driver as needDriver,"+

"a.start\_time as startTime,"+

"a.return\_time as returnTime,"+

"a.send\_car\_time as sendCarTime,"+

"a.return\_remarks as returnRemarks,"+

"a.section\_chief\_status as sectionChiefStatus,"+

"a.section\_chief\_time as sectionChiefTime,"+

"a.section\_remarks as sectionRemarks,"+

"a.create\_date as createDate,"+

"a.driver\_id as driverId,"+

"h.driver\_name as driverName,"+

"a.car\_id as carId,"+

"i.car\_name as carName,"+

"i.car\_code as carCode,"+

"b.user\_id as useCarAppUserId,"+

"b.name as useCarAppUserName,"+

"c.user\_id as sectionChiefId,"+

"c.name as sectionChiefName,"+

"d.user\_id as sendCarUserId,"+

"d.name as sendCarUserName,"+

"e.user\_id as createUserId,"+

"e.name as createUserName,"+

"f.department\_id as departmentId,"+

"f.name as departmentName,"+

"g.status\_id as statusId,"+

"g.name as statusName "+

"from t\_oa\_usecarapp a left join t\_oa\_car\_driver h on a.driver\_id=h.user\_id left join t\_oa\_car i on a.car\_id=i.car\_id left join t\_bpm\_user d on a.send\_car\_user\_id=d.user\_id,t\_bpm\_user b ,t\_bpm\_user c ,t\_bpm\_user e,t\_bpm\_department f,t\_oa\_usecar\_status g "+

"where a.user\_id=b.user\_id and a.section\_chief\_id=c.user\_id and a.create\_user\_id=e.user\_id and a.department\_id=f.department\_id and a.status\_id=g.status\_id"

+cond

,paramList.toArray());

PrintWriter out = this.getResponse().getWriter();

out.println(list.size());

out.flush();

}

public String getAppNumCondition(){

String conditionsStr="";

User u=(User)super.getSession().getAttribute("visituser");

if(u.getUserLevel()==4||u.getUserLevel()==5){

}else if(u.getUserLevel()==3){

conditionsStr += " and a.SECTION\_CHIEF\_ID = ? and b.USER\_ID !=? and g.status\_id not in(0000,1001) and a.section\_chief\_status =0";

paramList.add(u.getId());

paramList.add(u.getId());

}

return conditionsStr;

}

//获得需要派车的数量。

public void getSendCarNum() throws IOException{

String cond=this.getSendCarCondition();

List list=super.getDefaultManager().getCountSQL(

"select a.use\_carapp\_id as useCarAppId," +

"a.user\_num as userNum,"+

"a.send\_status as sendStatus,"+

"a.description as description,"+

"a.need\_driver as needDriver,"+

"a.start\_time as startTime,"+

"a.return\_time as returnTime,"+

"a.send\_car\_time as sendCarTime,"+

"a.return\_remarks as returnRemarks,"+

"a.section\_chief\_status as sectionChiefStatus,"+

"a.section\_chief\_time as sectionChiefTime,"+

"a.section\_remarks as sectionRemarks,"+

"a.create\_date as createDate,"+

"a.driver\_id as driverId,"+

"h.driver\_name as driverName,"+

"a.car\_id as carId,"+

"i.car\_name as carName,"+

"i.car\_code as carCode,"+

"b.user\_id as useCarAppUserId,"+

"b.name as useCarAppUserName,"+

"c.user\_id as sectionChiefId,"+

"c.name as sectionChiefName,"+

"d.user\_id as sendCarUserId,"+

"d.name as sendCarUserName,"+

"e.user\_id as createUserId,"+

"e.name as createUserName,"+

"f.department\_id as departmentId,"+

"f.name as departmentName,"+

"g.status\_id as statusId,"+

"g.name as statusName "+

"from t\_oa\_usecarapp a left join t\_oa\_car\_driver h on a.driver\_id=h.user\_id left join t\_oa\_car i on a.car\_id=i.car\_id left join t\_bpm\_user d on a.send\_car\_user\_id=d.user\_id,t\_bpm\_user b ,t\_bpm\_user c ,t\_bpm\_user e,t\_bpm\_department f,t\_oa\_usecar\_status g "+

"where a.user\_id=b.user\_id and a.section\_chief\_id=c.user\_id and a.create\_user\_id=e.user\_id and a.department\_id=f.department\_id and a.status\_id=g.status\_id"

+cond

,paramList.toArray());

PrintWriter out = this.getResponse().getWriter();

out.println(list.size());

out.flush();

}

public String getSendCarCondition(){

String conditionsStr="";

User u=(User)super.getSession().getAttribute("visituser");

conditionsStr += " and a.send\_status = ? and g.status\_id =? ";

paramList.add("0");

paramList.add("9999");

return conditionsStr;

}

public String toForMyCarApp(){

return "toForMyCarApp";

}

@Override

public Object getModel() {

return po;

}

}

package com.bizlogic.oa.action;

import java.io.IOException;

import java.io.PrintWriter;

import java.math.BigDecimal;

import java.util.Date;

import java.util.List;

import org.apache.commons.lang.StringUtils;

import org.apache.commons.lang.time.DateFormatUtils;

import net.sf.json.JSONObject;

import com.bizlogic.base.User;

import com.bizlogic.common.action.BaseAction;

/\*\*

\* 请假分析图Ajax请求

\* @author Administrator

\*/

public class AskForLeaveAjaxAction extends BaseAction{

private String act = "";

private String leaveType;

private String countType;

private String searchYear;

private String department;

private String month;

public String getAct() {

return act;

}

public void setAct(String act) {

this.act = act;

}

public String getLeaveType() {

return leaveType;

}

public void setLeaveType(String leaveType) {

this.leaveType = leaveType;

}

public String getCountType() {

return countType;

}

public void setCountType(String countType) {

this.countType = countType;

}

public String getSearchYear() {

return searchYear;

}

public void setSearchYear(String searchYear) {

this.searchYear = searchYear;

}

public String getDepartment() {

return department;

}

public void setDepartment(String department) {

this.department = department;

}

public String getMonth() {

return month;

}

public void setMonth(String month) {

this.month = month;

}

public void ajaxMethod() throws IOException {

JSONObject res = new JSONObject();

super.getResponse().setCharacterEncoding("UTF-8");

if(StringUtils.isEmpty(searchYear)){

searchYear = DateFormatUtils.format(new Date(), "yyyy");

}

if(StringUtils.isEmpty(countType) || countType.equals("P")){

countType = "count(\*) pcnt";

}else{

countType = "sum(t.daynum) dcnt";

}

if(StringUtils.equals(act, "getAllAskLeave")) {

res.put("content", getAllAskLeave(searchYear,countType,leaveType));

}else if(StringUtils.equals(act, "getDeptAskLeave")) {

res.put("content", getDeptAskLeave(searchYear,countType,leaveType,department));

}else if(StringUtils.equals(act, "getStaffAskLeave")) {

res.put("content", getStaffAskLeave(searchYear,countType,leaveType,department));

}else if(StringUtils.equals(act, "getLeaveList")) {

res.put("content", getLeaveList(month,department,leaveType));

}

//结果输出

PrintWriter out = this.getResponse().getWriter();

out.println(res.toString());

out.flush();

out.close();

}

private List getLeaveList(String month,String department,String leaveType) {

String sql = "select b.name,c.leave\_type\_name,a.daynum,to\_char(a.start\_time,'yyyy-mm-dd'),to\_char(a.end\_time,'yyyy-mm-dd')"

+ " from t\_oa\_leaveapp a,t\_bpm\_user b,t\_oa\_leavetype c where a.leave\_user\_id = b.user\_id and a.leave\_type\_id = c.leave\_type\_id"

+ " and a.status\_id = '9999' and to\_char(a.start\_time, 'yyyymm') = '" + month + "' and a.dept\_id = '" + department + "' and a.leave\_type\_id = '" + leaveType + "'"

+ " order by a.start\_time asc";

return super.getDefaultManager().findSQL(sql);

}

private List getAllAskLeave(String searchYear,String countType,String leaveType) {

User user = super.getDefaultManager().get(User.class, getVisit().getId());

int userLevel = user.getUserLevel();

String sql = "";

if(userLevel == 1||this.getRightLeaveC(getVisit().getId(),"LeaveRightGroup")){

sql = "select t.dept\_id,t1.name dept\_name," + countType

+ " from (select a.dept\_id,a.daynum from t\_oa\_leaveapp a where a.status\_id = '9999' ";

if(StringUtils.isNotEmpty(leaveType)){

sql += " and a.leave\_type\_id = '" + leaveType + "'";

}

sql += " and to\_char(a.start\_time, 'yyyy') = '" + searchYear + "') t,t\_bpm\_department t1 where t.dept\_id = t1.department\_id ";

sql += " group by t.dept\_id,t1.name";

}else if(userLevel == 2){

sql = "select t.dept\_id,t1.name dept\_name," + countType

+ " from (select a.dept\_id,a.daynum from t\_oa\_leaveapp a where a.status\_id = '9999' and to\_char(a.start\_time, 'yyyy') = '" + searchYear + "'";

if(StringUtils.isNotEmpty(leaveType)){

sql += " and a.leave\_type\_id = '" + leaveType + "'";

}

sql += ") t,t\_bpm\_department t1,t\_oa\_department\_leadership t2 "

+ " where t.dept\_id = t1.department\_id and t1.department\_id = t2.department\_id and t2.user\_id = '" + getVisit().getId() + "' group by t.dept\_id,t1.name";

}else{

sql = "select t.dept\_id,t1.name dept\_name," + countType

+ " from (select a.dept\_id,a.daynum from t\_oa\_leaveapp a where a.status\_id = '9999' and to\_char(a.start\_time, 'yyyy') = '" + searchYear + "'";

if(StringUtils.isNotEmpty(leaveType)){

sql += " and a.leave\_type\_id = '" + leaveType + "'";

}

sql += ") t,t\_bpm\_department t1"

+ " where t.dept\_id = t1.department\_id and t.dept\_id = '" + getVisit().getDepartment().getId()+ "' group by t.dept\_id,t1.name";

}

return super.getDefaultManager().findSQL(sql);

}

private List getDeptAskLeave(String searchYear,String countType,String leaveType, String department) {

String sql = "select t.leave\_type\_id,t1.leave\_type\_name," + countType

+ " from (select a.leave\_type\_id,a.daynum from t\_oa\_leaveapp a where a.status\_id = '9999' and to\_char(a.start\_time, 'yyyy') = '" + searchYear + "' and a.dept\_id = '" + department + "'";

if(StringUtils.isNotEmpty(leaveType)){

sql += " and a.leave\_type\_id = '" + leaveType + "'";

}

sql += ") t,t\_oa\_leavetype t1 where t.leave\_type\_id = t1.leave\_type\_id group by t.leave\_type\_id,t1.leave\_type\_name";

return super.getDefaultManager().findSQL(sql);

}

private List getStaffAskLeave(String searchYear,String countType,String leaveType, String department) {

String sql = "select t.mondate," + countType

+ " from (select a.daynum,to\_char(a.start\_time, 'yyyymm') mondate from t\_oa\_leaveapp a where a.status\_id = '9999'"

+ " and to\_char(a.start\_time, 'yyyy') = '" + searchYear + "' and dept\_id = '" + department + "' and leave\_type\_id = '" + leaveType + "'"

+ ") t group by t.mondate order by t.mondate asc";

return super.getDefaultManager().findSQL(sql);

}

/\*\*

\* 用户是否有权限

\*/

protected boolean getRightLeaveC(String userId, String group\_code) {

BigDecimal conutRight = (BigDecimal) super.getDefaultManager()

.getObjectSQL(

"select count(\*) from t\_bpm\_dict "

+ " where code=? and group\_code=?", userId, group\_code);

int conutRightSize = conutRight.intValue();

if (conutRightSize > 0)

return true;

else return false;

}

}

package com.bizlogic.oa.action;

import java.io.ByteArrayInputStream;

import java.io.ByteArrayOutputStream;

import java.io.IOException;

import java.io.InputStream;

import java.io.UnsupportedEncodingException;

import java.text.SimpleDateFormat;

import java.util.Calendar;

import org.apache.commons.lang.StringUtils;

import org.apache.poi.hssf.usermodel.HSSFCell;

import org.apache.poi.hssf.usermodel.HSSFCellStyle;

import org.apache.poi.hssf.usermodel.HSSFRow;

import org.apache.poi.hssf.usermodel.HSSFSheet;

import org.apache.poi.hssf.usermodel.HSSFWorkbook;

import org.apache.poi.ss.usermodel.Font;

import org.apache.poi.ss.util.CellRangeAddress;

import com.bizlogic.base.User;

import com.bizlogic.common.action.BaseAction;

import com.bizlogic.oa.bean.AskForLeave;

public class AskForLeaveExportAction extends BaseAction{

//审批表编号

private String askForLeaveId;

public String execute() throws Exception

{

return super.execute();

}

public String getFileName() {

try {

return new String(("干部休假审批表.xls").getBytes("gbk"), "ISO8859\_1");

} catch (UnsupportedEncodingException e) {

return "干部休假审批表.xls";

}

}

public InputStream getExcelFile() {

//创建excel工作簿

HSSFWorkbook workbook = new HSSFWorkbook();

String str = "";

str = "干部休假审批表";

//创建一页

HSSFSheet sheet = workbook.createSheet(str);

//设置列宽

sheet.setColumnWidth(0, 4000);

sheet.setColumnWidth(1, 4000);

sheet.setColumnWidth(2, 4000);

sheet.setColumnWidth(3, 4000);

sheet.setColumnWidth(4, 4000);

sheet.setColumnWidth(5, 4000);

sheet.setColumnWidth(6, 4000);

//序号、提议人姓名、汇报人姓名样式

HSSFCellStyle style0 = workbook.createCellStyle();

//议题样式

HSSFCellStyle style1 = workbook.createCellStyle();

//时间和地点样式

HSSFCellStyle style2 = workbook.createCellStyle();

//字段标题

HSSFCellStyle style3 = workbook.createCellStyle();

//强制换行

style1.setWrapText(true);

//标题样式

HSSFCellStyle style4 = workbook.createCellStyle();

//style0设置边框

style0.setBorderTop(HSSFCellStyle.BORDER\_THIN);

style0.setBorderLeft(HSSFCellStyle.BORDER\_THIN);

style0.setBorderRight(HSSFCellStyle.BORDER\_THIN);

style0.setBorderBottom(HSSFCellStyle.BORDER\_THIN);

//style1设置边框

style1.setBorderTop(HSSFCellStyle.BORDER\_THIN);

style1.setBorderLeft(HSSFCellStyle.BORDER\_THIN);

style1.setBorderRight(HSSFCellStyle.BORDER\_THIN);

style1.setBorderBottom(HSSFCellStyle.BORDER\_THIN);

//style3设置边框

style3.setBorderTop(HSSFCellStyle.BORDER\_THIN);

style3.setBorderLeft(HSSFCellStyle.BORDER\_THIN);

style3.setBorderRight(HSSFCellStyle.BORDER\_THIN);

style3.setBorderBottom(HSSFCellStyle.BORDER\_THIN);

//边框颜色

style0.setRightBorderColor((short) 00000);

style1.setRightBorderColor((short) 00000);

style3.setRightBorderColor((short) 00000);

//内容在单元格中间显示

style0.setVerticalAlignment(HSSFCellStyle.VERTICAL\_CENTER);

style1.setVerticalAlignment(HSSFCellStyle.ALIGN\_LEFT);

style3.setVerticalAlignment(HSSFCellStyle.VERTICAL\_CENTER);

style4.setVerticalAlignment(HSSFCellStyle.VERTICAL\_CENTER);

//内容居中显示

style0.setAlignment(HSSFCellStyle.ALIGN\_CENTER);

style1.setAlignment(HSSFCellStyle.ALIGN\_LEFT);

style3.setAlignment(HSSFCellStyle.ALIGN\_CENTER);

style4.setAlignment(HSSFCellStyle.ALIGN\_CENTER);

//序号、议题、姓名字体

Font font = workbook.createFont();

font.setBoldweight((short) 10);//设置粗体

font.setColor((short) 00000);//字体颜色

font.setFontHeightInPoints((short) 12); //字体大小

font.setFontName("宋体");//设置字体

//会议名称

Font font2 = workbook.createFont();

font2.setBoldweight((short) 10);//设置粗体

font2.setColor((short) 00000);//字体颜色

font2.setFontHeightInPoints((short) 16);//字体大小

font2.setFontName("黑体");//设置字体

//字段标题

Font font3 = workbook.createFont();

font3.setBoldweight((short) 10);//设置粗体

font3.setColor((short) 00000);//字体颜色

font3.setFontHeightInPoints((short) 12);//字体大小

font3.setFontName("黑体");

style0.setFont(font);

style1.setFont(font);

style2.setFont(font);

style3.setFont(font3);

style4.setFont(font2);

//设置合并单元格（0行到0行，0列到5列进行合并）

sheet.addMergedRegion(new CellRangeAddress(0,0,0,5));

//设置合并单元格（1行到1行，4列到5列进行合并)

sheet.addMergedRegion(new CellRangeAddress(1,1,4,5));

//获取当前年月

String date = getCDate();

date = date.replaceFirst("-", "年").replace("-", "月");

date = date + "日";

//第0行

HSSFRow row0 = sheet.createRow(0);

//0行0列值

HSSFCell cell0 = row0.createCell(0);

cell0.setCellValue("干部休假审批表");

cell0.setCellStyle(style4);

//第一行

HSSFRow row1 = sheet.createRow(1);

HSSFCell cell1 = row1.createCell(0);

cell1.setCellValue("");

cell1.setCellStyle(style2);

cell1 = row1.createCell(1);

cell1.setCellValue("");

cell1.setCellStyle(style2);

cell1 = row1.createCell(2);

cell1.setCellValue("");

cell1.setCellStyle(style2);

cell1 = row1.createCell(3);

cell1.setCellValue("");

cell1.setCellStyle(style2);

cell1 = row1.createCell(4);

cell1.setCellValue(date);

cell1.setCellStyle(style2);

if(askForLeaveId != null && !"".equals(askForLeaveId)){

//根据id获取数据

AskForLeave askForLeave = super.getDefaultManager().get(AskForLeave.class, askForLeaveId);

//第一行

HSSFRow row = sheet.createRow(2);

row.setHeight((short) 600);

row.setHeight((short) 1300);

//姓名

HSSFCell cell = row.createCell(0);

cell.setCellStyle(style0);

cell.setCellValue("姓名");

//姓名

cell = row.createCell(1);

cell.setCellStyle(style1);

cell.setCellValue(askForLeave.getAppLeaveUser().getName());

//工作部门

cell = row.createCell(2);

cell.setCellStyle(style0);

cell.setCellValue("工作部门");

//工作部门

cell = row.createCell(3);

cell.setCellStyle(style0);

cell.setCellValue(askForLeave.getDept().getName());

//参加工作时间

cell = row.createCell(4);

cell.setCellStyle(style0);

cell.setCellValue("参加工作时间");

//参加工作时间

cell = row.createCell(5);

cell.setCellStyle(style0);

cell.setCellValue(DateToString(askForLeave.getAppLeaveUser().getWorkTime()));

//第二行

HSSFRow row2 = sheet.createRow(3);

row2.setHeight((short) 1300);

//应修天数

HSSFCell cell2 = row2.createCell(0);

cell2.setCellStyle(style0);

cell2.setCellValue("应修天数");

//应修天数

cell2 = row2.createCell(1);

cell2.setCellStyle(style1);

cell2.setCellValue(askForLeave.getTotleDayNum());

//已休假天数

cell2 = row2.createCell(2);

cell2.setCellStyle(style0);

cell2.setCellValue("已休假天数");

//已休假天数

cell2 = row2.createCell(3);

cell2.setCellStyle(style0);

cell2.setCellValue(askForLeave.getHaveDayNum());

//休假起止时间

cell2 = row2.createCell(4);

cell2.setCellStyle(style0);

cell2.setCellValue("休假起止时间");

//参加工作时间

cell2 = row2.createCell(5);

cell2.setCellStyle(style1);

cell2.setCellValue(DateToString1(askForLeave.getStartTime())+"至"+DateToString1(askForLeave.getEndTime()));

//第三行

HSSFRow row3 = sheet.createRow(4);

row3.setHeight((short) 1300);

//科长签字

HSSFCell cell3 = row3.createCell(0);

cell3.setCellStyle(style0);

cell3.setCellValue("科长签字");

//科长签字

cell3 = row3.createCell(1);

cell3.setCellStyle(style1);

cell3.setCellValue(askForLeave.getSectionChief().getName());

//主管领导审批

cell3 = row3.createCell(2);

cell3.setCellStyle(style0);

cell3.setCellValue("主管领导审批");

//主管领导审批

cell3 = row3.createCell(3);

cell3.setCellStyle(style0);

cell3.setCellValue(askForLeave.getExecutiveDirector().getName());

//备注

cell3 = row3.createCell(4);

cell3.setCellStyle(style0);

cell3.setCellValue("备注");

//备注

cell3 = row3.createCell(5);

cell3.setCellStyle(style0);

if(askForLeave.getRemarks() != null){

cell3.setCellValue(askForLeave.getRemarks());

}else{

cell3.setCellValue("");

}

User learUser=askForLeave.getAppLeaveUser();

String islj= askForLeave.getIslj();

String daynum=askForLeave.getDayNum();

boolean isFlag=false;

if(learUser.getUserLevel()<4&&Integer.parseInt(daynum)>2){

isFlag = true;

}

if(learUser.getUserLevel()<4&&StringUtils.trimToEmpty(islj).equals("是")){

isFlag = true;

}

//第四行,第五行，主任审批,书记审批

if(isFlag){

HSSFRow row4 = sheet.createRow(5);

row4.setHeight((short) 600);

//主任审批

HSSFCell cell4 = row4.createCell(0);

cell4.setCellStyle(style0);

cell4.setCellValue("主任审批");

HSSFRow row5 = sheet.createRow(6);

row5.setHeight((short) 600);

//书记审批

HSSFCell cell5 = row5.createCell(0);

cell5.setCellStyle(style0);

cell5.setCellValue("书记审批");

//合并单元格

sheet.addMergedRegion(new CellRangeAddress(5,5,1,5));

sheet.addMergedRegion(new CellRangeAddress(6,6,1,5));

HSSFCell cell41 = row4.createCell(1);

cell41 = row4.createCell(1);

cell41.setCellStyle(style1);

cell41 = row4.createCell(2);

cell41.setCellStyle(style0);

cell41 = row4.createCell(3);

cell41.setCellStyle(style0);

cell41 = row4.createCell(4);

cell41.setCellStyle(style0);

cell41 = row4.createCell(5);

cell41.setCellStyle(style0);

HSSFCell cell51 = row5.createCell(1);

cell51 = row5.createCell(1);

cell51.setCellStyle(style1);

cell51 = row5.createCell(2);

cell51.setCellStyle(style0);

cell51 = row5.createCell(3);

cell51.setCellStyle(style0);

cell51 = row5.createCell(4);

cell51.setCellStyle(style0);

cell51 = row5.createCell(5);

cell51.setCellStyle(style0);

}

}

ByteArrayOutputStream baos = new ByteArrayOutputStream();

try {

workbook.write(baos);

} catch (IOException e) {

e.printStackTrace();

}

byte[] ba = baos.toByteArray();

ByteArrayInputStream bais = new ByteArrayInputStream(ba);

return bais;

}

/\*\*

\* 将Date 类型 格式的日期转换成 String 类型 yyyy-MM-dd

\*

\* @param param

\* @return

\*/

private static String DateToString(java.sql.Date param) {

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");

return sdf.format(param);

}

private static String DateToString1(java.util.Date param) {

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");

return sdf.format(param);

}

//根据指定格式获取当前日期

public static String getCurrentDate() {

return getCurrentTime("yyyy-MM");

}

public static String getCDate() {

return getCurrentTime("yyyy-MM-dd");

}

/\*\*

\* 取当前时间按指定类型格式化

\* @param pattern时间格式

\* @return String类型的时间

\*/

public static String getCurrentTime(String pattern) {

SimpleDateFormat sdf = new SimpleDateFormat(pattern);

return sdf.format(Calendar.getInstance().getTime());

}

public String getAskForLeaveId() {

return askForLeaveId;

}

public void setAskForLeaveId(String askForLeaveId) {

this.askForLeaveId = askForLeaveId;

}

}

package com.bizlogic.oa.action;

import java.io.IOException;

import java.io.PrintWriter;

import java.util.List;

import net.sf.json.JSONObject;

import org.apache.commons.lang.StringUtils;

import com.bizlogic.base.Company;

import com.bizlogic.base.front.action.BaseUserEditAction;

import com.bizlogic.common.Element;

import com.bizlogic.oa.bean.CadreComArchitecture;

import com.bizlogic.oa.bean.CadreUserArchitecture;

public class CadreCommunityEditAction extends BaseUserEditAction{

//干部下社区的人员编号

private String cadreCommunityUserId;

//社区集合

private List<Company> companyList;

protected void prepareShow(){

CadreComArchitecture cadreComArchitecture = (CadreComArchitecture)element;

companyList= super.getDefaultManager().findAll("from Company a where a.parent!=null order by a.id");

}

protected boolean prepareSave(Element element){

CadreComArchitecture cadreComArchitecture = (CadreComArchitecture)element;

return super.prepareSave(element);

}

/\*\*

\* ajax请求

\* @throws IOException

\*/

public void ajaxMethod() throws IOException {

JSONObject res = new JSONObject();

//Response编码设置

super.getResponse().setCharacterEncoding("utf-8");

//保存下社区的信息

if(StringUtils.equals(act, "save")) {

//获得干部下社区的人员编号

String userArchitectureId=getParameterValue("userId");

//获得所选社区的id

String companyId=getParameterValue("companyId");

//根据干部下社区的人员编号和社区id进行查询

List cadreCommunityList = super.getDefaultManager().findAll("from CadreComArchitecture c where c.cadreUserArchitecture.id=? and c.company.id=?", userArchitectureId,companyId);

//查询的结果不为空

if(cadreCommunityList.size()>0){

res.put("content", "error");

//查询的结果为空,对提交的信息进行保存

}else{

//查找到所有组员

List<CadreUserArchitecture> userArchitectureList = super.getDefaultManager().findAll("from CadreUserArchitecture t " +

" where " +

" t.group.id=?", userArchitectureId);

for(int i=0;i<userArchitectureList.size();i++){

//架构人员对象

CadreUserArchitecture userArchi = userArchitectureList.get(i);

CadreComArchitecture cadreComArchitecture = new CadreComArchitecture();

cadreComArchitecture.setCompany(super.getDefaultManager().get(Company.class, companyId));

cadreComArchitecture.setCadreUserArchitecture(userArchi);

super.getDefaultManager().save(cadreComArchitecture);

res.put("content", "success");

}

}

}

//结果输出

PrintWriter out = this.getResponse().getWriter();

out.println(res.toString());

out.flush();

out.close();

}

protected Class getElementClass()

{

return CadreComArchitecture.class;

}

public List<Company> getCompanyList() {

return companyList;

}

public void setCompanyList(List<Company> companyList) {

this.companyList = companyList;

}

public String getCadreCommunityUserId() {

return cadreCommunityUserId;

}

public void setCadreCommunityUserId(String cadreCommunityUserId) {

this.cadreCommunityUserId = cadreCommunityUserId;

}

}

package com.bizlogic.oa.action;

import java.util.Calendar;

import java.util.Date;

import java.util.List;

import com.bizlogic.base.front.action.BaseUserEditAction;

import com.bizlogic.common.Element;

import com.bizlogic.oa.bean.CadreComArchitecture;

import com.bizlogic.oa.bean.CadreCommunity;

import com.bizlogic.oa.bean.CadreCommunityTask;

import com.bizlogic.oa.bean.CadreCommunityUser;

import com.bizlogic.oa.bean.CadreCommunityUserTask;

import com.bizlogic.oa.bean.CadreUserArchitecture;

public class CadreCommunityTaskEditAction extends BaseUserEditAction{

protected void prepareShow(){

CadreCommunityTask cadreCommunityTask = (CadreCommunityTask)element;

}

protected boolean prepareSave(Element element){

CadreCommunityTask cadreCommunityTask = (CadreCommunityTask)element;

cadreCommunityTask.setSendDate(new Date());

cadreCommunityTask.setStatus(0);

Calendar rightNow = Calendar.getInstance();

int year = rightNow.get(Calendar.YEAR);

cadreCommunityTask.setTaskYear(String.valueOf(year));

return super.prepareSave(element);

}

public String save()

{

CadreCommunityTask cadreCommunityTask = (CadreCommunityTask)element;

super.save();

//将之前下社区人员更新为历史人员

super.getDefaultManager().execute("update CadreCommunityUser t set t.effect=0");

/\*\*

\* 产生新的主管主任数据

\*/

//获取主管主任人员架构列表

List<CadreUserArchitecture> cadreUserArchitectureList1 = super.getDefaultManager().findAll("from CadreUserArchitecture t " +

" where " +

" t.userLevel=1 " +

" and t.id=t.group.id");

//将之前干部对应社区数据更新为无效

super.getDefaultManager().execute("update CadreCommunity t set t.effect=0 where t.effect=1");

for(int i=0;i<cadreUserArchitectureList1.size();i++){

CadreUserArchitecture cadreUserArchitecture = cadreUserArchitectureList1.get(i);

CadreCommunityUser cadreCommunityUser = new CadreCommunityUser();

//人员编号

cadreCommunityUser.setUser(cadreUserArchitecture.getUser());

//科室数

cadreCommunityUser.setDepartmentCount(cadreUserArchitecture.getDepartmentCount());

//科员数

cadreCommunityUser.setClerkCount(cadreUserArchitecture.getClerkCount());

//状态

cadreCommunityUser.setStatus(cadreUserArchitecture.getStatus());

//用户等级

cadreCommunityUser.setUserLevel(cadreUserArchitecture.getUserLevel());

//是否有效数据

cadreCommunityUser.setEffect(1);

//设置科室(所在科室有可能产生变化，所以该处需要设置人员架构当时所在的科室)

cadreCommunityUser.setDepartment(cadreUserArchitecture.getUser().getDepartment());

//设置不下社区原因

cadreCommunityUser.setReason(cadreUserArchitecture.getReason());

super.getDefaultManager().save(cadreCommunityUser);

//设置组编号

cadreCommunityUser.setGroup(cadreCommunityUser);

super.getDefaultManager().save(cadreCommunityUser);

/\*\*

\* 产生新的对应社区数据

\*/

List<CadreComArchitecture> cadreComArchitectureList = super.getDefaultManager().findAll("from CadreComArchitecture t " +

" where t.cadreUserArchitecture=?",cadreUserArchitecture);

for(int x=0;x<cadreComArchitectureList.size();x++){

//干部对应社区架构

CadreComArchitecture cadreComArchitecture = cadreComArchitectureList.get(x);

//创建新的干部对应社区数据

CadreCommunity cy = new CadreCommunity();

cy.setCadreCommunityUser(cadreCommunityUser);

cy.setCompany(cadreComArchitecture.getCompany());

cy.setEffect(1);

super.getDefaultManager().save(cy);

}

}

/\*\*

\* 产生新的主管主任组员数据

\*/

//获取主管主任组员

List<CadreUserArchitecture> groupList = super.getDefaultManager().findAll("from CadreUserArchitecture t " +

" where " +

" t.userLevel=1 " +

" and t.id!=t.group.id");

for(int i=0;i<groupList.size();i++){

CadreUserArchitecture groupUser = groupList.get(i);

//创新的对象

CadreCommunityUser cadreCommunityUser = new CadreCommunityUser();

//人员编号

cadreCommunityUser.setUser(groupUser.getUser());

//科室数

cadreCommunityUser.setDepartmentCount(groupUser.getDepartmentCount());

//科员数

cadreCommunityUser.setClerkCount(groupUser.getClerkCount());

//状态

cadreCommunityUser.setStatus(groupUser.getStatus());

//用户等级

cadreCommunityUser.setUserLevel(groupUser.getUserLevel());

//是否有效数据

cadreCommunityUser.setEffect(1);

//设置科室(所在科室有可能产生变化，所以该处需要设置人员架构当时所在的科室)

cadreCommunityUser.setDepartment(groupUser.getUser().getDepartment());

//设置不下社区原因

cadreCommunityUser.setReason(groupUser.getReason());

//设置组

CadreCommunityUser ca = (CadreCommunityUser)super.getDefaultManager().getObject("from CadreCommunityUser t " +

" where " +

" t.user=? " +

" and t.effect=1", groupUser.getGroup().getUser());

cadreCommunityUser.setGroup(ca);

super.getDefaultManager().save(cadreCommunityUser);

}

/\*\*

\* 产生新的科长数据

\*/

//获取科长架构列表

List<CadreUserArchitecture> cadreUserArchitectureList2 = super.getDefaultManager().findAll("from CadreUserArchitecture t " +

" where " +

" t.higherLevel.higherLevel is null");

for(int i=0;i<cadreUserArchitectureList2.size();i++){

CadreUserArchitecture cadreUserArchitecture = cadreUserArchitectureList2.get(i);

CadreCommunityUser cadreCommunityUser = new CadreCommunityUser();

//人员编号

cadreCommunityUser.setUser(cadreUserArchitecture.getUser());

//科室数

cadreCommunityUser.setDepartmentCount(cadreUserArchitecture.getDepartmentCount());

//科员数

cadreCommunityUser.setClerkCount(cadreUserArchitecture.getClerkCount());

//状态

cadreCommunityUser.setStatus(cadreUserArchitecture.getStatus());

//用户等级

cadreCommunityUser.setUserLevel(cadreUserArchitecture.getUserLevel());

//是否有效数据

cadreCommunityUser.setEffect(1);

//获取上级领导

CadreCommunityUser higherLevel = (CadreCommunityUser)super.getDefaultManager().getObject("from CadreCommunityUser t " +

" where " +

" t.user=? " +

" and t.effect=1", cadreUserArchitecture.getHigherLevel().getUser());

cadreCommunityUser.setHigherLevel(higherLevel);

//设置科室(所在科室有可能产生变化，所以该处需要设置人员架构当时所在的科室)

cadreCommunityUser.setDepartment(cadreUserArchitecture.getUser().getDepartment());

//设置不下社区原因

cadreCommunityUser.setReason(cadreUserArchitecture.getReason());

super.getDefaultManager().save(cadreCommunityUser);

}

List<CadreUserArchitecture> cadreUserArchitectureList3 = super.getDefaultManager().findAll("from CadreUserArchitecture t " +

" where " +

" t.higherLevel.higherLevel.higherLevel is null");

for(int i=0;i<cadreUserArchitectureList3.size();i++){

CadreUserArchitecture cadreUserArchitecture = cadreUserArchitectureList3.get(i);

CadreCommunityUser cadreCommunityUser = new CadreCommunityUser();

cadreCommunityUser.setUser(cadreUserArchitecture.getUser());

cadreCommunityUser.setDepartmentCount(cadreUserArchitecture.getDepartmentCount());

cadreCommunityUser.setClerkCount(cadreUserArchitecture.getClerkCount());

//状态

cadreCommunityUser.setStatus(cadreUserArchitecture.getStatus());

cadreCommunityUser.setUserLevel(cadreUserArchitecture.getUserLevel());

//是否有效数据

cadreCommunityUser.setEffect(1);

//获取上级领导

CadreCommunityUser higherLevel = (CadreCommunityUser)super.getDefaultManager().getObject("from CadreCommunityUser t " +

" where " +

" t.user=? " +

" and t.effect=1", cadreUserArchitecture.getHigherLevel().getUser());

cadreCommunityUser.setHigherLevel(higherLevel);

cadreCommunityUser.setDepartment(cadreUserArchitecture.getUser().getDepartment());

//设置不下社区原因

cadreCommunityUser.setReason(cadreUserArchitecture.getReason());

super.getDefaultManager().save(cadreCommunityUser);

}

List<CadreCommunityUser> cadreCommunityUserList = super.getDefaultManager().findAll(

"from CadreCommunityUser t where t.effect = 1 and (t.group is null or t.group.id=t.id) order by t.userLevel");

for(int i=0;i<cadreCommunityUserList.size();i++){

CadreCommunityUser cadreCommunityUser = cadreCommunityUserList.get(i);

CadreCommunityUserTask cadreCommunityUserTask = new CadreCommunityUserTask();

cadreCommunityUserTask.setCadreCommunityTask(cadreCommunityTask);

cadreCommunityUserTask.setCadreCommunityUser(cadreCommunityUser);

cadreCommunityUserTask.setSelfMustCount(2);

cadreCommunityUserTask.setMustCount((cadreCommunityUser.getClerkCount()+1)\*2);

if(cadreCommunityUser.getStatus() == 1){

cadreCommunityUserTask.setSelfCompleteCount(0);

cadreCommunityUserTask.setCompleteCount(0);

cadreCommunityUserTask.setStatus(1);

}else{

cadreCommunityUserTask.setSelfCompleteCount(2);

------------END-----------