**海豚文件管理系统软件源程序**

package Dao;

import Model.File;

import org.apache.ibatis.annotations.Param;

import org.apache.ibatis.annotations.Select;

import org.apache.ibatis.annotations.Update;

public interface FileMapper {

// 根据MD5码查询物理文件记录

@Select("select \* from [File] where MD5=#{md5};")

File selectByMd5(@Param("md5") String MD5);

// 根据ID将文件指针加1

@Update("update [File] set point = point+1 WHERE id=#{ID}")

int updatePointByID(@Param("ID") Integer ID);

// 根据ID将文件指针减1

@Update("update [File] set point = point-1 WHERE id=#{ID}")

int updatesubPointByID(@Param("ID") Integer ID);

int deleteByPrimaryKey(Integer id);

int insert(File record);

int insertSelective(File record);

File selectByPrimaryKey(Integer id);

int updateByPrimaryKeySelective(File record);

int updateByPrimaryKey(File record);

}

package Dao;

import Model.FileNode;

import org.apache.ibatis.annotations.Param;

import org.apache.ibatis.annotations.Select;

import java.util.List;

public interface FileNodeMapper {

/\*\*

\* 根据目录节点ID和文件名称获取文件信息

\* @param NodeID

\* @param FileName

\* @return FileNode

\*/

@Select("select \* from FileNode where NodeID=#{NodeID} and [Name]=#{FileName}")

FileNode selectByNodeIDandFileName(@Param("NodeID") Integer NodeID,@Param("FileName") String FileName);

//获取某一文件节点下所有文件

@Select("select \* from FileNode where NodeID=#{NodeID}")

List<FileNode> selectAllFilebyNodeID(@Param("NodeID") Integer NodeID);

int deleteByPrimaryKey(Integer pathnode);

int insert(FileNode record);

int insertSelective(FileNode record);

FileNode selectByPrimaryKey(Integer pathnode);

int updateByPrimaryKeySelective(FileNode record);

int updateByPrimaryKey(FileNode record);

}

package Dao;

import Model.Node;

import org.apache.ibatis.annotations.Param;

import org.apache.ibatis.annotations.Select;

import java.util.List;

public interface NodeMapper {

// 给出节点ID与当前用户ID查看是否存在这个节点

@Select("select \* from Node where ID=#{nodeID} and UserID=#{userID}")

Node selectNodeByNodeIDAndUserID(@Param("nodeID") int nodeID,@Param("userID") int userID);

// 返回当前节点下所有子节点

List<Node> selectAllChildNode(int nodeID);

// 获取某一节点下子节点中特定节点名称信息

Node selectChildByNodeIDAndNodeName(@Param("nodeID") int nodeID,@Param("nodeName") String nodeName);

int deleteByPrimaryKey(Integer id);

int insert(Node record);

int insertSelective(Node record);

Node selectByPrimaryKey(Integer id);

int updateByPrimaryKeySelective(Node record);

int updateByPrimaryKey(Node record);

}

package Dao;

import Model.ChildNode;

import org.apache.ibatis.annotations.Delete;

import org.apache.ibatis.annotations.Param;

import org.apache.ibatis.annotations.Select;

public interface ChildNodeMapper {

/\*\*

\* 删除父级ID下所有子节点引用，慎用

\* @param ParentID

\* @return

\*/

@Delete("delete from ChildNode where ParentID=#{ParentID}")

int deleteAllChildByParentID(@Param("ParentID") Integer ParentID);

/\*\*

\* 根据父级节点和子节点获取ID

\* @param ParentID

\* @param ChildID

\* @return

\*/

@Select("select \* from ChildNode where ParentID=#{ParentID} and ChildID=#{ChildID}")

ChildNode selectByParentChild(Integer ParentID, Integer ChildID);

/\*\*

\* 删除父级目录下某一个子目录

\* @param ParentID

\* @param ChildID

\* @return

\*/

@Delete("delete from ChildNode where ParentID=#{parentID} and ChildID=#{ChildID}")

int deleteByParentIDAndChildID(@Param("parentID") Integer ParentID, @Param("ChildID") Integer ChildID);

int deleteByPrimaryKey(Integer id);

int insert(ChildNode record);

int insertSelective(ChildNode record);

ChildNode selectByPrimaryKey(Integer id);

int updateByPrimaryKeySelective(ChildNode record);

int updateByPrimaryKey(ChildNode record);

}

package Dao;

import Model.Log;

import org.apache.ibatis.annotations.Select;

import java.util.List;

public interface LogMapper {

@Select("select \* from Log")

List<Log> selectAll();

int deleteByPrimaryKey(Integer id);

int insert(Log record);

int insertSelective(Log record);

Log selectByPrimaryKey(Integer id);

int updateByPrimaryKeySelective(Log record);

int updateByPrimaryKey(Log record);

}

package Dao;

import Model.User;

import org.apache.ibatis.annotations.Delete;

import org.apache.ibatis.annotations.Param;

import org.apache.ibatis.annotations.Select;

import org.springframework.stereotype.Service;

import java.util.ArrayList;

import java.util.List;

public interface UserMapper {

// 根据用户名称查询用户信息

List<User> selectByUserName(String name);

// 根据邮箱查询用户信息

@Select("select \* from [User] where E\_Mail=#{value}")

List<User> selectByEMail(String value);

// 根据主账号ID获取所有子账号信息

List<User> selectChildByParentID(int value);

// 根据用户名称删除账号

@Delete("delete [User] where UserName=#{value}")

int deleteByUserName(@Param("value") String value);

int deleteByPrimaryKey(Integer id);

int insert(User record);

int insertSelective(User record);

User selectByPrimaryKey(Integer id);

int updateByPrimaryKeySelective(User record);

int updateByPrimaryKey(User record);

}

package Dao;

import Model.ExtLink;

import org.apache.ibatis.annotations.Param;

import org.apache.ibatis.annotations.Select;

import java.util.List;

public interface ExtLinkMapper {

/\*\*

\* 根据UUID名称获取当前的外链

\* @param UUID

\* @return

\*/

@Select("select \* from ExtLink where Name=#{name}")

ExtLink selectByName(@Param("name") String UUID);

/\*\*

\* 获取当前用户下所有外链集合

\* @param UserID

\* @return

\*/

@Select("select \* from ExtLink where UserID=#{userid}")

List<ExtLink> selectByUserID(@Param("userid") Integer UserID);

int deleteByPrimaryKey(Integer id);

int insert(ExtLink record);

int insertSelective(ExtLink record);

ExtLink selectByPrimaryKey(Integer id);

int updateByPrimaryKeySelective(ExtLink record);

int updateByPrimaryKey(ExtLink record);

}

package Dao;

import Model.ExtObj;

import org.apache.ibatis.annotations.Param;

import java.util.List;

public interface ExtObjMapper {

List<ExtObj> selectAllExtobjByExLinkName(@Param("name") String UUID);

int deleteByPrimaryKey(Integer id);

int insert(ExtObj record);

int insertSelective(ExtObj record);

ExtObj selectByPrimaryKey(Integer id);

int updateByPrimaryKeySelective(ExtObj record);

int updateByPrimaryKey(ExtObj record);

}

package Function;

import org.springframework.util.DigestUtils;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.io.IOException;

import java.io.InputStream;

import java.security.MessageDigest;

import java.util.UUID;

public abstract class MD5 {

/\*\*

\* 生成不可逆的md5打码

\* @param Str

\* @return string md5密码

\*/

public static String getstring(String Str) {

MessageDigest md5 = null;

try {

md5 = MessageDigest.getInstance("MD5");

} catch (Exception e) {

System.out.println(e.toString());

e.printStackTrace();

return "";

}

char[] charArray = Str.toCharArray();

byte[] byteArray = new byte[charArray.length];

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

byteArray[i] = (byte) charArray[i];

}

byte[] md5Bytes = md5.digest(byteArray);

StringBuffer hexValue = new StringBuffer();

for (int i = 0; i < md5Bytes.length; i++) {

int val = ((int) md5Bytes[i]) & 0xff;

if (val < 16){

hexValue.append("0");

}

hexValue.append(Integer.toHexString(val));

}

return hexValue.toString();

}

/\*\*

\* md5加密算法(可逆算法)

\* @param inStr

\* @return

\*/

public static String getsec(String inStr) {

return getstring(inStr);

// char[] a = inStr.toCharArray();

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

// a[i] = (char) (a[i] ^ 't');

// }

// String s = new String(a);

// return s;

}

public static String getres(String string) {

return getsec(string);

}

/\*\*

\* 获取某个路径下文件的md5码

\* @param path

\* @return 16进制的md5码

\* @throws IOException

\*/

public static String getFile(String path) throws IOException {

return DigestUtils.md5DigestAsHex(new FileInputStream(path));

}

/\*\*

\* 获取某个FileStream流中的md5码

\* @param stream

\* @return

\* @throws IOException

\*/

public static String getFile(InputStream stream) throws IOException {

return DigestUtils.md5DigestAsHex(stream);

}

/\*\*

\* 获取UUID

\* @return 返回32位数据库中标准格式

\*/

public static String getUUID() {

return UUID.randomUUID().toString().replace("-", "");

}

}

package Function;

/\*\*

\* 这是验证类，由于遵循前端不可信原则，所有的代码在后端都需要验证，因此将一些常用方法封装在这里

\* @author zyz

\*/

public abstract class Vaild {

public static boolean E\_mail(String e\_mail) {

String EMAIL\_REGEX = "^[\\w-\\.+]\*[\\w-\\.]\\@([\\w-]+\\.)+[\\w]+[\\w]{1}";

return e\_mail.matches(EMAIL\_REGEX);

}

}

package Function;

import Model.User;

import com.alibaba.fastjson.JSON;

import com.alibaba.fastjson.JSONObject;

import com.alibaba.fastjson.serializer.SimplePropertyPreFilter;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

/\*\*

\* @author zyz

\* 将对象转换为json格式封装，设置项目统一的json返回接口，此类方法为静态方法不可实例化

\* 其中每个方法中status表示状态码，此处状态码具体在接口文档中规范，但是Msg.OK表示为通过，Msg.ERR表示为参数不完整错误

\* 其他错误请按照接口文档给定相应的状态值

\*

\*/

public abstract class Msg {

/\*\*

\* 状态集合

\*/

/\*\*

\* OK 请求成功

\*/

public final static Integer OK=200;

/\*\*

\* ERR 请求方法错误，请查看接口文档

\*/

public final static Integer ERR=555;

/\*\*

\* LoginAuth 当前没有登录需要登录权限

\*/

public final static Integer LoginAuth = 556;

/\*\*

\* NoAuth 当前用户没有权限

\*/

public final static Integer NoAuth=557;

/\*\*

\* UnKnow 未知错误，应该是服务器出现问题

\*/

public final static Integer UnKnow=599;

/\*\*

\* 将一个字符串封装为json格式并序列化

\*

\* @param Status 状态为类中的状态

\* @param Title 消息标题

\* @param message 消息信息

\* @return String json序列化的字符串,格式{"status":"状态","file"}

\*/

public static String ParseStr(int Status, String Title, String message) {

Map<String,Object> map = new HashMap<String,Object>();

map.put("status", Status);

map.put("title", Title);

map.put("type", "string");

map.put("data", message);

return JSON.toJSONString(map);

}

/\*\*

\* 将一个字典数据转换为json字符串

\* @param Status 状态

\* @param Title 标题

\* @param map 数据字典

\* @return json格式字符串

\*/

public static String ParseMap(int Status, String Title, Object map) {

Map<String, Object> ret = new HashMap<String, Object>();

ret.put("status", Status);

ret.put("title", Title);

ret.put("type", "map");

ret.put("data", map);

return JSON.toJSONString(ret);

}

/\*\*

\* 将一个列表数据转换为json字符串

\* @param Status

\* @param Title

\* @param list

\* @return

\*/

public static String ParseList(int Status, String Title, List list) {

Map<String, Object> ret = new HashMap<String, Object>();

ret.put("status", Status);

ret.put("title", Title);

ret.put("type", "list");

ret.put("data", list);

return JSON.toJSONString(ret);

}

/\*\*

\* 获取对象的某个部分，并转换为标准的返回格式，直接调用

\* @param Status 返回的状态码int

\* @param Title 标题

\* @param object 返回的对象，可以是一个字典，或者一个Bean

\* @param properties 需要包含在对象内的参数

\* @return

\*/

public static String ParseInclude(int Status, String Title, Object object, String... properties) {

Map res = Include(object, properties);

return ParseMap(Status, Title, res);

}

/\*\*

\* 获取对象的某个部分，并转换为dic的json格式(包含）

\* @param object 需要序列化的对象

\* @param properties 需要选择的原型

\* @return 返回一个选择封装后的Map对象

\*/

public static Map Include(Object object, String...properties) {

SimplePropertyPreFilter filter = new SimplePropertyPreFilter();

for (String add : properties) {

filter.getIncludes().add(add);

}

return JSONObject.parseObject(JSON.toJSONString(object, filter), Map.class);

}

/\*\*

\* 获取排除了对象的某个部分，并转换为标准的返回格式，直接调用

\* @param Status 返回的状态码int

\* @param Title 标题

\* @param object 返回的对象，可以是一个字典，或者一个Bean

\* @param properties 需要排除在对象内的参数

\* @return

\*/

public static String ParseExclude(int Status, String Title, Object object, String... properties) {

Map res = Exclude(object, properties);

return ParseMap(Status, Title, res);

}

/\*\*

\* 获取对象的某个部分，并转换为dic的json格式(排除)

\* @param object 需要序列化的对象

\* @param properties 需要排除的原型

\* @return 返回一个选择封装后的Map对象

\*/

public static Map Exclude(Object object, String...properties) {

SimplePropertyPreFilter filter = new SimplePropertyPreFilter();

for (String add : properties) {

filter.getExcludes().add(add);

}

return JSONObject.parseObject(JSON.toJSONString(object, filter), Map.class);

}

}

package Function;

import javafx.scene.shape.Path;

import java.util.List;

/\*\*

\* 文件名称或者路径解析类，主要完成文件名称与后缀的分离，文件路径的解析功能

\*

\* @author zyz

\*/

public abstract class FileParse {

/\*\*

\* 根据文件完整名称返回文件名和后缀两个部分

\* @param FileName

\* @return String数组，第一个元素为文件名称，第二个元素为文件后缀，中间没有.好分开

\*/

public static String[] FileName(String FileName) {

String a[] = FileName.split("\\.");

if (a.length > 2) {

String b[]=new String[2];

String c=a[0];

for (int i = 1; i < a.length - 1; i++) {

c+="."+a[i];

}

b[0]=c;

b[1] = a[a.length - 1];

return b;

}

else

{

String c[] = new String[2];

c[0] = a.length > 0 ? a[0] : "";

c[1] = a.length > 1 ? a[1] : "";

return c;

}

}

public static String[] PathName(String PathName) {

PathName = PathName.replace("\\", "/");

String a[] = PathName.split("/");

return a;

}

}

package Controllor.File;

import Function.Msg;

import Model.User;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.ResponseBody;

import javax.servlet.http.HttpSession;

@Controller

@RequestMapping("/file")

public class ModifyFileName extends FileControllerFather {

/\*\*

\* 修改文件名称

\* @param FileNode

\* @param FilePreName

\* @param FileNewName

\* @param session

\* @return

\*/

@RequestMapping("modifyfilename")

public @ResponseBody

String modifyfilename(Integer FileNode,

String FilePreName,

String FileNewName,

HttpSession session) {

User loginUser = (User) session.getAttribute("user");

Model.FileNode filechange = fileNodeMapper.selectByPrimaryKey(FileNode);

// 验证

try {

// 验空

if (FileNode == null || FileNewName == null || FilePreName == null) {

throw new Exception(Msg.ERR.toString());

}

// 文件是否存在

if (filechange == null) {

throw new Exception("502");

}

// 原始文件一致性

if (!filechange.getName().equals(FilePreName)) {

throw new Exception("501");

}

// 验证文件是否有权限

if (!AuthCheck(filechange.getNodeid(), loginUser)) {

throw new Exception(Msg.NoAuth.toString());

}

// 查找重名

if (fileNodeMapper.selectByNodeIDandFileName(filechange.getNodeid(), FileNewName)!=null) {

throw new Exception("503");

}

} catch (Exception e) {

return Msg.ParseStr(Integer.parseInt(e.getMessage()), "/file/modifyfilename", "");

}

// 处理

filechange.setName(FileNewName);

fileNodeMapper.updateByPrimaryKeySelective(filechange);

// 返回

return Msg.ParseStr(Msg.OK, "/file/modifyfilename", "");

}

}

package Controllor.File;

import Function.Msg;

import Model.ChildNode;

import Model.FileNode;

import Model.Node;

import Model.User;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.ResponseBody;

import javax.servlet.http.HttpSession;

import java.util.\*;

@Controller

@RequestMapping("/file")

public class ModifyFilePosition extends FileControllerFather {

/\*\*

\* 移动目录或者文件到一个新的目录下，PS没有实现跨节点移动阻止功能

\* @param UserName

\* @param Nodes

\* @param NewPosisitonNodeID

\* @param PrePositionNodeID

\* @param session

\* @return

\*/

@RequestMapping("/modifyfileposition")

public @ResponseBody

String modifyfileposition(String UserName,

String Nodes,

Integer NewPosisitonNodeID,

Integer PrePositionNodeID,

HttpSession session) {

User loginUser = (User) session.getAttribute("user");

List<innerNodes> NodeLists = new ArrayList<innerNodes>();

// 欺骗

UserName = loginUser.getUsername();

if(NewPosisitonNodeID==0) NewPosisitonNodeID = loginUser.getLogicnode();

if(PrePositionNodeID==0) PrePositionNodeID = loginUser.getLogicnode();

// 验证

try {

// 验证空

if (UserName == null || Nodes == null || NewPosisitonNodeID == null || PrePositionNodeID == null) {

throw new Exception(Msg.ERR.toString());

}

// 用户登录一致性

if (UserName != loginUser.getUsername()) {

throw new Exception(Msg.LoginAuth.toString());

}

// 预处理需要移动的节点

for(String t : Nodes.split("\\|"))

{

if(t.length()<2) throw new Exception(Msg.ERR.toString());

NodeLists.add(new innerNodes(Integer.parseInt(t.substring(0, 1)), Integer.parseInt(t.substring(1)),

nodeMapper,fileNodeMapper));

}

// 节点存在性和权限

boolean flag = true;

for (innerNodes n : NodeLists) {

if(n.checkExisit()==false||n.CheckAuth(loginUser)==false)

{

flag=false;break;

}

}

if(flag==false) throw new Exception("502");

} catch (Exception e) {

return Msg.ParseMap(Integer.parseInt(e.getMessage()), "/file/modifyfileposition", null);

}

// 处理

for (innerNodes inner : NodeLists) {

if (inner.Type == 0) {

// 移动目录

Node cur = nodeMapper.selectByPrimaryKey(inner.NodeID);

childNodeMapper.deleteByParentIDAndChildID(cur.getParentnode(), cur.getId());

cur.setParentnode(NewPosisitonNodeID);

nodeMapper.updateByPrimaryKeySelective(cur);

ChildNode childNode = new ChildNode();

childNode.setChildid(cur.getId());

childNode.setParentid(NewPosisitonNodeID);

childNodeMapper.insert(childNode);

}else

{

FileNode cur = fileNodeMapper.selectByPrimaryKey(inner.NodeID);

cur.setNodeid(NewPosisitonNodeID);

cur.setModifydate(new Date());

fileNodeMapper.updateByPrimaryKeySelective(cur);

}

}

// 返回

Map res = new HashMap();

res.put("NodeID", NewPosisitonNodeID);

return Msg.ParseMap(Msg.OK, "/file/modifyfileposition", res);

}

}

package Controllor.File;

import Function.Msg;

import Model.ChildNode;

import Model.FileNode;

import Model.Node;

import Model.User;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.ResponseBody;

import javax.servlet.http.HttpSession;

import java.util.\*;

@Controller

@RequestMapping("/file")

public class CopyFile extends FileControllerFather {

/\*\*

\* 文件复制，实现对目录或者文件的复制

\* @param UserName

\* @param PreNodeID

\* @param NewNodeID

\* @param CopyNodes

\* @param session

\* @return

\*/

@RequestMapping("copyfile")

public @ResponseBody

String copyfile(String UserName,

Integer PreNodeID,

Integer NewNodeID,

String CopyNodes,

HttpSession session) {

User loginUser = (User) session.getAttribute("user");

List<innerNodes> NodeLists = new ArrayList<innerNodes>();

// 欺骗

UserName = loginUser.getUsername();

if(PreNodeID==0) PreNodeID = loginUser.getLogicnode();

if(NewNodeID==0) NewNodeID = loginUser.getLogicnode();

// 验证

try {

// 验证空

if (UserName == null || PreNodeID == null || NewNodeID == null || CopyNodes == null) {

throw new Exception(Msg.ERR.toString());

}

// 用户登录一致性

if (UserName != loginUser.getUsername()) {

throw new Exception(Msg.LoginAuth.toString());

}

// 预处理需要复制的节点

for(String t : CopyNodes.split("\\|"))

{

if(t.length()<2) throw new Exception(Msg.ERR.toString());

NodeLists.add(new innerNodes(Integer.parseInt(t.substring(0, 1)), Integer.parseInt(t.substring(1)),

nodeMapper,fileNodeMapper));

}

// 节点存在性和权限

boolean flag = true;

for (innerNodes n : NodeLists) {

if(n.checkExisit()==false||n.CheckAuth(loginUser)==false||n.CheckRepeat(NewNodeID)==false)

{

flag=false;break;

}

}

if(flag==false) throw new Exception("502");

} catch (Exception e) {

return Msg.ParseMap(Integer.parseInt(e.getMessage()), "/file/modifyfileposition", null);

}

// 处理

for (innerNodes inner : NodeLists) {

if (inner.Type == 0) {

// 目录复制（深度优先）

Stack<Node> nodeStack = new Stack<Node>();

nodeStack.push(nodeMapper.selectByPrimaryKey(inner.NodeID));

while (!nodeStack.empty()) {//判断栈不为空

Node cur = nodeStack.pop();

Node newnode = new Node();

newnode.setParentnode(cur.getId());

newnode.setUserid(cur.getUserid());

newnode.setName(cur.getName());

nodeMapper.insert(newnode);

// 添加子节点信息

ChildNode temp = new ChildNode();

temp.setParentid(cur.getId());

temp.setChildid(newnode.getId());

childNodeMapper.insert(temp);

// 获取当前目录下的所有文件，同样执行一次复制

List<FileNode> curFileNode = fileNodeMapper.selectAllFilebyNodeID(cur.getId());

for (FileNode fileNode : curFileNode) {

fileNode.setPathnode(null);

fileNode.setNodeid(newnode.getId());

fileNode.setModifydate(null);

fileNode.setUploaddate(null);

fileNodeMapper.insertSelective(fileNode);

}

// 获取当前目录下的所有子目录，并压入堆栈

nodeStack.addAll(nodeMapper.selectAllChildNode(cur.getId()));

}

}

else {

// 文件复制

FileNode cur = fileNodeMapper.selectByPrimaryKey(inner.NodeID);

cur.setPathnode(null);

cur.setNodeid(NewNodeID);

cur.setUploaddate(null);

cur.setModifydate(null);

fileNodeMapper.insertSelective(cur);

fileMapper.updatePointByID(cur.getFileid());

}

}

// 返回

Map res = new HashMap();

res.put("NodeID", NewNodeID);

return Msg.ParseMap(Msg.OK, "/file/copyfile", res);

}

}

package Controllor.File;

import Dao.\*;

import Model.FileNode;

import Model.Node;

import Model.User;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.web.context.ServletConfigAware;

import javax.servlet.ServletConfig;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

@Controller

public class FileControllerFather implements ServletConfigAware {

@Autowired

protected UserMapper userMapper;

@Autowired

protected FileMapper fileMapper;

@Autowired

protected FileNodeMapper fileNodeMapper;

@Autowired

protected NodeMapper nodeMapper;

@Autowired

protected ChildNodeMapper childNodeMapper;

protected ServletConfig servletConfig;

public void setServletConfig(ServletConfig servletConfig) {

this.servletConfig = servletConfig;

}

// region tool 工具

/\*\*

\* 判断当前账号是否拥有节点权限

\* @param nodeID

\* @param loginUser

\* @return ture,有权限 false，没有权限

\*

\*/

public boolean AuthCheck(int nodeID, User loginUser)

{

//账号为子账号

if(loginUser.getParentid()!=0)

{ //验证主账号是否有权限

if(nodeMapper.selectNodeByNodeIDAndUserID(nodeID,loginUser.getParentid())==null)

return true;

// 递归验证子账号是否有权限

Node Curnode = nodeMapper.selectByPrimaryKey(nodeID);

Node t=Curnode;

do{

t= nodeMapper.selectByPrimaryKey(t.getParentnode());

}

while (t.getId()!=loginUser.getLogicnode()&&t.getParentnode()!=loginUser.getLogicnode());

// 考虑如果子账号获取主账号所有权限时

if(t.getParentnode()==loginUser.getLogicnode() && t.getId()!=loginUser.getLogicnode() )

return false;

else

return true;

}

//账号为主账号

else

{

return !(nodeMapper.selectNodeByNodeIDAndUserID(nodeID,loginUser.getId())==null);

}

}

/\*\*

\* 返回前端的map格式

\* @param NodeID

\* @param NodeName

\* @param Type

\* @param Childs

\* @return

\*/

public static Map getres(Integer NodeID, String NodeName, int Type, List Childs) {

Map res = new HashMap();

res.put("NodeID", NodeID);

res.put("NodeName", NodeName);

res.put("Type", Type);

res.put("Childs", Childs);

return res;

}

// endregion

}

/\*\*

\* 这是专门属于文件类操作的私有类

\* @author zyz

\*

\*/

class innerNodes{

private NodeMapper nodeMapper;

private FileNodeMapper fileNodeMapper;

public innerNodes(Integer Type, Integer NodeID,NodeMapper nodeMapper,FileNodeMapper fileNodeMapper) {

this.Type = Type;

this.NodeID = NodeID;

this.fileNodeMapper = fileNodeMapper;

this.nodeMapper = nodeMapper;

}

public Integer Type;

public Integer NodeID;

/\*\*

\* 检查当前节点是否存在

\* @return

\*/

public boolean checkExisit() {

if (this.Type == 0) {

//查找目录存在

return nodeMapper.selectByPrimaryKey(NodeID)!=null;

}

else

{

//查找文件存在

return fileNodeMapper.selectByPrimaryKey(NodeID)!=null;

}

}

/\*\*

\* 获取当前节点是否有访问权限

\* @param user

\* @return

\*/

public boolean CheckAuth(User user) {

int NodeID;

if (this.Type == 0) {

NodeID = this.NodeID;

}

else

{

FileNode file = fileNodeMapper.selectByPrimaryKey(this.NodeID);

NodeID = file.getNodeid();

}

Node t = nodeMapper.selectNodeByNodeIDAndUserID(NodeID,

user.getParentid() == 0 ? user.getId() : user.getParentid());

return t != null;

}

public boolean CheckRepeat(int nodeID) {

if(this.Type==1)

{

if(fileNodeMapper.selectByNodeIDandFileName(nodeID,fileNodeMapper.selectByPrimaryKey(this.NodeID).getName())!=null) return false;

}

else

{

if (nodeMapper.selectChildByNodeIDAndNodeName(nodeID, nodeMapper.selectByPrimaryKey(this.NodeID).getName())!=null) {

return false;

}

}

return true;

}

}

package Controllor.File;

import Function.Msg;

import Model.FileNode;

import Model.Node;

import Model.User;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.ResponseBody;

import javax.servlet.http.HttpSession;

import java.util.ArrayList;

import java.util.List;

import java.util.Stack;

@Controller

@RequestMapping("/file")

public class DelFile extends FileControllerFather {

@RequestMapping("/delfile")

public @ResponseBody

String delfile(String UserName,

String Nodes,

HttpSession session) {

User loginUser = (User) session.getAttribute("user");

List<innerNodes> NodeLists = new ArrayList<innerNodes>();

// 欺骗

UserName = loginUser.getUsername();

// 验证

try {

// 验证空

if (UserName == null || Nodes == null ) {

throw new Exception(Msg.ERR.toString());

}

// 用户登录一致性

if (!UserName.equals(loginUser.getUsername())) {

throw new Exception(Msg.LoginAuth.toString());

}

// 预处理需要复制的节点

for(String t : Nodes.split("\\|"))

{

if(t.length()<2) throw new Exception(Msg.ERR.toString());

NodeLists.add(new innerNodes(Integer.parseInt(t.substring(0, 1)), Integer.parseInt(t.substring(1)),

nodeMapper,fileNodeMapper));

}

// 节点存在性和权限

boolean flag = true;

for (innerNodes n : NodeLists) {

if(n.checkExisit()==false||n.CheckAuth(loginUser)==false)

{

flag=false;break;

}

}

if(flag==false) throw new Exception("502");

} catch (Exception e) {

return Msg.ParseMap(Integer.parseInt(e.getMessage()), "/file/modifyfileposition", null);

}

// 处理

for (innerNodes inner : NodeLists) {

if (inner.Type == 0) {

// 目录删除（深度优先）

Node current = nodeMapper.selectByPrimaryKey(inner.NodeID);

childNodeMapper.deleteByParentIDAndChildID(current.getParentnode(), current.getId());

Stack<Node> nodeStack = new Stack<Node>();

nodeStack.push(nodeMapper.selectByPrimaryKey(inner.NodeID));

while (!nodeStack.empty()) {//判断栈不为空

Node cur = nodeStack.pop();

// 获取当前目录下的所有文件，执行删除

List<FileNode> curFileNode = fileNodeMapper.selectAllFilebyNodeID(cur.getId());

for (FileNode fileNode : curFileNode) {

fileMapper.updatesubPointByID(fileNode.getFileid());

fileNodeMapper.deleteByPrimaryKey(fileNode.getPathnode());

}

// 获取当前目录下的所有子目录，并压入堆栈

nodeStack.addAll(nodeMapper.selectAllChildNode(cur.getId()));

// 删除当前节点所有子节点引用

childNodeMapper.deleteAllChildByParentID(cur.getId());

nodeMapper.deleteByPrimaryKey(cur.getId());

}

}

else {

// 文件删除

FileNode cur = fileNodeMapper.selectByPrimaryKey(inner.NodeID);

fileMapper.updatesubPointByID(cur.getFileid());

fileNodeMapper.deleteByPrimaryKey(cur.getPathnode());

}

}

return Msg.ParseStr(Msg.OK, "/file/delfile", "");

}

}

package Controllor.File;

import Function.FileParse;

import Function.MD5;

import Function.Msg;

import Model.File;

import Model.FileNode;

import Model.User;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.bind.annotation.ResponseBody;

import javax.servlet.http.HttpSession;

import java.io.\*;

import java.util.HashMap;

import java.util.UUID;

@Controller

@RequestMapping("/file")

public class NewFile extends FileControllerFather {

/\*\*

\* 新建一个文件

\* @param FileName

\* @param NodeID

\* @param UserName

\* @param session

\* @return

\*/

@RequestMapping(value = "/newfile",method = RequestMethod.POST)

public @ResponseBody

String NewFile (String FileName,

Integer NodeID,

String UserName,

HttpSession session) {

String path=servletConfig.getServletContext().getInitParameter("path");

User loninUser = (User) session.getAttribute("user");

// 欺骗

UserName = loninUser.getUsername();

// 验证

try {

// 验证空

if (FileName == null || NodeID == null || UserName == null)

throw new Exception(Msg.ERR.toString());

if(NodeID==0) NodeID = loninUser.getLogicnode();

// 验证目录是否存在

if(nodeMapper.selectByPrimaryKey(NodeID)==null)

throw new Exception("501");

// 用户是否有权限

if (!AuthCheck(NodeID,loninUser))

throw new Exception(Msg.NoAuth.toString());

// 当前目录下是否有重名

if (fileNodeMapper.selectByNodeIDandFileName(NodeID,FileName)!=null)

throw new Exception("503");

// 登录状态是否一致

if(!loninUser.getUsername().equals(UserName))

throw new Exception(Msg.LoginAuth.toString());

} catch (Exception e) {

return Msg.ParseMap(Integer.parseInt(e.getMessage()), "/file/newfile", null);

}

// 处理

FileNode fileNode = new FileNode();

fileNode.setName(FileParse.FileName(FileName)[0]);

fileNode.setSuffix(FileParse.FileName(FileName)[1]);

fileNode.setNodeid(NodeID);

// 创建空文件

InputStream stream = new ByteArrayInputStream(new byte[0]);

try {

File file = fileMapper.selectByMd5(MD5.getFile(stream));

// 文件存在，执行指针加1操作,直接提交sql语句防止出现并发问题

if (file != null)

{

fileMapper.updatePointByID(file.getId());

fileNode.setFileid(file.getId());

}

else

{

String uuid = UUID.randomUUID().toString().replace("-","");

file = new File();

file.setPoint(1);

file.setUuid(uuid);

file.setSize(0);

file.setMd5(MD5.getFile(stream));

// 创建物理存储文件

java.io.File f = new java.io.File(path+uuid);

f.createNewFile();

// File写入数据库

fileMapper.insert(file);

fileNode.setFileid(file.getId());

}

} catch (IOException e) {

return Msg.ParseMap(Msg.UnKnow, "/file/newfile", null);

}

// FileNode写入

fileNodeMapper.insertSelective(fileNode);

fileNode = fileNodeMapper.selectByPrimaryKey(fileNode.getPathnode());

// 返回

HashMap data = new HashMap();

data.put("FileID", fileNode.getPathnode());

data.put("NodeID", NodeID);

data.put("FileName", fileNode.getName());

data.put("Suffix", fileNode.getSuffix());

data.put("UploadData", fileNode.getUploaddate());

return Msg.ParseMap(Msg.OK, "/dir/newfile", data);

}

}

package Controllor.File;

import Function.Msg;

import Model.FileNode;

import Model.Node;

import Model.User;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.ResponseBody;

import javax.servlet.http.HttpSession;

import java.util.\*;

@Controller

@RequestMapping("/file")

public class GetFileTree extends FileControllerFather {

@RequestMapping("/getfiletree")

public @ResponseBody

String getfiletree(Integer NodeID,

Integer Deep,

HttpSession session) {

User loginUser = (User) session.getAttribute("user");

String UserName = loginUser.getUsername();

// 默认从根节点开始往下返回

if (NodeID == null||NodeID==0) {

NodeID = loginUser.getLogicnode();

}

// 验证

try {

// 验空

if (UserName == null || NodeID == null || Deep == null) {

throw new Exception(Msg.ERR.toString());

}

// 登录状态一致性

if (UserName != loginUser.getUsername()) {

throw new Exception(Msg.LoginAuth.toString());

}

// 验证当前目录是否存在

if (nodeMapper.selectByPrimaryKey(NodeID) == null) {

throw new Exception("500");

}

// 文件权限

if (!AuthCheck(NodeID, loginUser)) {

throw new Exception(Msg.NoAuth.toString());

}

} catch (Exception e) {

return Msg.ParseMap(Integer.parseInt(e.getMessage()), "/file/getfiletree", null);

}

// 处理

// 子账号换取主账号操作权限

int userID = loginUser.getParentid() == 0 ? loginUser.getId() : loginUser.getParentid();

// 获取目录

List<Map> allNodes = new ArrayList<Map>();

Stack<Integer> queueNodes = new Stack<Integer>();

// 推入初始节点

queueNodes.push(NodeID);

for (int i = 0; i <= Deep; i++) {

List<Integer> curLevelID = new ArrayList<Integer>();

while (!queueNodes.empty()) {

List<Integer> curChilds = new ArrayList<Integer>();

Node cur = nodeMapper.selectByPrimaryKey(queueNodes.pop());

List<Node> curLevel = nodeMapper.selectAllChildNode(cur.getId());

for (Node t : curLevel) {//添加当前节点下的子节点信息

curChilds.add(t.getId());

}

// 添加到当前全部下级ID中，在下次while后压入堆栈

curLevelID.addAll(curChilds);

// 添加到节点列表中

if(i!=0)

allNodes.add(getres(cur.getId(), cur.getName(), 0, curChilds));

}

queueNodes.addAll(curLevelID);

}

// 处理文件

List<Map> allFiles = new ArrayList<Map>();

// 添加根节点下的文件

List<FileNode> re = fileNodeMapper.selectAllFilebyNodeID(NodeID);

for (FileNode res : re) {

allFiles.add(getres(res.getPathnode(), res.getName(), 1, null));

}

//// 添加其余节点下的文件

// for (Map t : allNodes) {

// re = fileNodeMapper.selectAllFilebyNodeID((Integer) t.get("NodeID"));

// for (FileNode res : re) {

// allFiles.add(getres(res.getPathnode(), res.getName(), 1, null));

// }

// }

// 合并

allNodes.addAll(allFiles);

// 返回

Map res = new HashMap();

res.put("Node", allNodes);

res.put("Deep", Deep);

res.put("UserName", loginUser.getUsername());

return Msg.ParseMap(Msg.OK, "/file/getfiletree", res);

}

}

package Controllor.Transfer;

import Function.MD5;

import Function.Msg;

import Model.File;

import Model.FileNode;

import Model.Node;

import Model.User;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.bind.annotation.ResponseBody;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpSession;

import java.util.\*;

@Controller

@RequestMapping("/transfer")

public class DownloadLink extends TransferFather {

/\*\*

\* 根据选中的内容保存文件下载信息

\* @param NodeList

\* @param session

\* @param request

\* @return

\*/

@RequestMapping(value = "/downloadlink", method = RequestMethod.POST)

public @ResponseBody

String downloadlink(String NodeList,

HttpSession session,

HttpServletRequest request) {

User loginUser = (User) session.getAttribute("user");

List<innerNodes> NodeLists = new ArrayList<innerNodes>();

// 验证

try {

// 用户登录验证

if (loginUser == null) throw new Exception(Msg.LoginAuth.toString());

// 预处理需要复制的节点

for(String t : NodeList.split("\\|"))

{

if(t.length()<2) throw new Exception(Msg.ERR.toString());

NodeLists.add(new innerNodes(Integer.parseInt(t.substring(0, 1)), Integer.parseInt(t.substring(1)),

nodeMapper,fileNodeMapper,fileMapper));

}

// 节点存在性和权限

boolean flag = true;

for (innerNodes n : NodeLists) {

if(n.checkExisit()==false||n.CheckAuth(loginUser)==false)

{

flag=false;break;

}

}

if(flag==false) throw new Exception("502");

} catch (Exception e) {

return Msg.ParseMap(Integer.parseInt(e.getMessage()), "/transfer/downloadlink", null);

}

// 处理

List<downFile> downfiles = new ArrayList<downFile>();

for (innerNodes innernode : NodeLists) {

if (innernode.Type == 1) {

// 文件

FileNode fn = fileNodeMapper.selectByPrimaryKey(innernode.NodeID);

File f = fileMapper.selectByPrimaryKey(fn.getFileid());

if(fn.getSuffix().equals(""))

downfiles.add(new downFile(fn.getName(), "", f.getUuid()));

else

downfiles.add(new downFile(fn.getName() +"."+ fn.getSuffix(), "", f.getUuid()));

}

else

{

// 目录

Node t = nodeMapper.selectByPrimaryKey(innernode.NodeID);

List<FileNode> curf = fileNodeMapper.selectAllFilebyNodeID(t.getId());

for (FileNode curt : curf) {

File f = fileMapper.selectByPrimaryKey(curt.getFileid());

// 添加当前目录下的文件

downfiles.add(new downFile(curt.getName() + curt.getSuffix(),

""+ t.getName()+"/",

f.getUuid()));

}

innerNodes.DFS(downfiles, innernode.NodeID, "" + t.getName(),

nodeMapper, fileNodeMapper, fileMapper);

}

}

// 返回

// 保存session，生成外部链接

String uuid = MD5.getUUID();

session.setAttribute(uuid, downfiles);

Map res = new HashMap();

res.put("url", uuid);

return Msg.ParseMap(Msg.OK, "/transfer/downloadlink", res);

}

}

package Controllor.Transfer;

import Dao.FileMapper;

import Dao.FileNodeMapper;

import Dao.NodeMapper;

import Dao.UserMapper;

import Model.File;

import Model.FileNode;

import Model.Node;

import Model.User;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.context.ServletConfigAware;

import javax.servlet.ServletConfig;

import java.io.OutputStream;

import java.util.List;

public class TransferFather implements ServletConfigAware {

@Autowired

protected NodeMapper nodeMapper;

@Autowired

protected UserMapper userMapper;

@Autowired

protected FileMapper fileMapper;

@Autowired

protected FileNodeMapper fileNodeMapper;

protected ServletConfig servletConfig;

public void setServletConfig(ServletConfig servletConfig) {

this.servletConfig = servletConfig;

}

/\*\*

\* 判断当前账号是否拥有节点权限

\* @param nodeID

\* @param loginUser

\* @return ture,有权限 false，没有权限

\*

\*/

public boolean AuthCheck(int nodeID, User loginUser)

{

//账号为子账号

if(loginUser.getParentid()!=0)

{ //验证主账号是否有权限

if(nodeMapper.selectNodeByNodeIDAndUserID(nodeID,loginUser.getParentid())==null)

return true;

// 递归验证子账号是否有权限

Node Curnode = nodeMapper.selectByPrimaryKey(nodeID);

Node t=Curnode;

do{

t= nodeMapper.selectByPrimaryKey(t.getParentnode());

}

while (t.getId()!=loginUser.getLogicnode()&&t.getParentnode()!=loginUser.getLogicnode());

// 考虑如果子账号获取主账号所有权限时

if(t.getParentnode()==loginUser.getLogicnode() && t.getId()!=loginUser.getLogicnode() )

return false;

else

return true;

}

//账号为主账号

else

{

return !(nodeMapper.selectNodeByNodeIDAndUserID(nodeID,loginUser.getId())==null);

}

}

}

/\*\*

\* 这是专门属于文件类操作的私有类

\* @author zyz

\*

\*/

class innerNodes{

@Autowired

private NodeMapper nodeMapper;

@Autowired

private FileNodeMapper fileNodeMapper;

@Autowired

private FileMapper fileMapper;

public innerNodes(Integer Type, Integer NodeID,NodeMapper nodeMapper,FileNodeMapper fileNodeMapper,FileMapper fileMapper) {

this.Type = Type;

this.NodeID = NodeID;

this.nodeMapper = nodeMapper;

this.fileMapper = fileMapper;

this.fileNodeMapper = fileNodeMapper;

}

public Integer Type;

public Integer NodeID;

/\*\*

\* 检查当前节点是否存在

\* @return

\*/

public boolean checkExisit() {

if (this.Type == 0) {

//查找目录存在

return nodeMapper.selectByPrimaryKey(NodeID)!=null;

}

else

{

//查找文件存在

return fileNodeMapper.selectByPrimaryKey(NodeID)!=null;

}

}

/\*\*

\* 获取当前节点是否有访问权限

\* @param user

\* @return

\*/

public boolean CheckAuth(User user) {

int NodeID;

if (this.Type == 0) {

NodeID = this.NodeID;

}

else

{

FileNode file = fileNodeMapper.selectByPrimaryKey(this.NodeID);

NodeID = file.getNodeid();

}

Node t = nodeMapper.selectNodeByNodeIDAndUserID(NodeID,

user.getParentid() == 0 ? user.getId() : user.getParentid());

return t != null;

}

public static boolean DFS(List<downFile> res, Integer nodeID, String path,

NodeMapper nodeMapper, FileNodeMapper fileNodeMapper, FileMapper fileMapper) {

if(nodeID==null) return true;

List<Node> cur = nodeMapper.selectAllChildNode(nodeID);

if(cur==null) return true;

else

{

for (Node t : cur) {

List<FileNode> curf = fileNodeMapper.selectAllFilebyNodeID(nodeID);

for (FileNode curt : curf) {

File f = fileMapper.selectByPrimaryKey(curt.getFileid());

// 添加当前目录下的文件

res.add(new downFile(curt.getName() + curt.getSuffix(),

path + "/"+ t.getName(),

f.getUuid()));

}

if(!t.getId().equals(nodeID))

innerNodes.DFS(res, t.getId(), path+"/"+t.getName(),

nodeMapper,fileNodeMapper,fileMapper);

}

}

return true;

}

}

class downFile{

private String filename;

private String path;

private String uuid;

private OutputStream stream;

public downFile(String filename, String path,String uuid) {

this.filename = filename;

this.path = path;

this.uuid=uuid;

}

public String getPath() {

return path;

}

public void setPath(String path) {

this.path = path;

}

public String getFilename() {

return filename;

}

public void setFilename(String filename) {

this.filename = filename;

}

public OutputStream getStream() {

return stream;

}

public void setStream(OutputStream stream) {

this.stream = stream;

}

public String getUuid() {

return uuid;

}

public void setUuid(String uuid) {

this.uuid = uuid;

}

}

package Controllor.Transfer;

import Function.Msg;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.ResponseBody;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

import javax.swing.event.DocumentEvent;

import java.io.\*;

import java.util.ArrayList;

import java.util.List;

import java.util.zip.ZipEntry;

import java.util.zip.ZipOutputStream;

@Controller

@RequestMapping("/transfer")

public class Download extends TransferFather {

@RequestMapping("/download")

public @ResponseBody

String Download(String Url, HttpSession session,

HttpServletRequest request, HttpServletResponse response) throws IOException {

// 获取是否存在对象

List<downFile> down = (ArrayList<downFile>) session.getAttribute(Url);

if (down == null) {

return Msg.ParseStr(Msg.ERR, "", "请求错误");

}

else{

// 处理返回问题

String path=servletConfig.getServletContext().getInitParameter("path");

List<filestreaminfo> res = new ArrayList<filestreaminfo>();

response.setContentType("text/html");

response.setHeader("Content-Disposition", "attachment;fileName=" + "hello"+".zip");

ZipOutputStream zos = new ZipOutputStream(response.getOutputStream());

for (downFile t : down) {

zos.putNextEntry(new ZipEntry(t.getPath() + t.getFilename()));

FileInputStream fis = new FileInputStream(new File(path+t.getUuid()));

int len = 0;

byte[] b = new byte[1024];

while((len = fis.read(b)) > 0) {

zos.write(b, 0, len);

}

fis.close();

response.flushBuffer();

}

}

return "完成";

}

}

class filestreaminfo{

private OutputStream stream;

private String name;

private String path;

public OutputStream getStream() {

return stream;

}

public void setStream(OutputStream stream) {

this.stream = stream;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getPath() {

return path;

}

public void setPath(String path) {

this.path = path;

}

}

package Controllor.Transfer;

import Function.FileParse;

import Function.MD5;

import Function.Msg;

import Model.FileNode;

import Model.User;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.ResponseBody;

import org.springframework.web.multipart.MultipartFile;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpSession;

import java.io.File;

import java.io.IOException;

import java.util.HashMap;

import java.util.Map;

@Controller

@RequestMapping("/transfer")

public class Upload extends TransferFather{

/\*\*

\* 文件上传方法，注意该方法一次只可以上传一个文件，同时只支持文件的上传不支持目录的上传！

\* @param file

\* @param NodeID

\* @param req

\* @param session

\* @return

\* @throws IOException

\*/

@RequestMapping(value = "/upload" ,method = RequestMethod.POST)

public @ResponseBody

String upload(@RequestParam("file") MultipartFile file,

Integer NodeID,

HttpServletRequest req,

HttpSession session) throws IOException {

User loginUser = (User) session.getAttribute("user");

// 欺骗

if(NodeID==0) NodeID = loginUser.getLogicnode();

// 验证

try {

if (file.isEmpty() || NodeID == null) {

throw new Exception(Msg.ERR.toString());

}

// 验证目录存在

if(nodeMapper.selectByPrimaryKey(NodeID)==null) throw new Exception("500");

// 验证目录权限

if(!AuthCheck(NodeID,loginUser)) throw new Exception(Msg.NoAuth.toString());

} catch (Exception e) {

return Msg.ParseList(Integer.parseInt(e.getMessage()), "", null);

}

// 处理

// 获取原文件名

String fileName[] = FileParse.FileName(file.getOriginalFilename());

// 根据MD5码判断当前是否已经存在

String md5 = MD5.getFile(file.getInputStream());

Model.File file1 = fileMapper.selectByMd5(md5);

if (file1==null) {

file1 = new Model.File();

file1.setUuid(MD5.getUUID());

file1.setPoint(1);

file1.setMd5(md5);

fileMapper.insertSelective(file1);

}

// 封装文件的逻辑目录

FileNode fileNode = new FileNode();

fileNode.setNodeid(NodeID);

fileNode.setName(fileName[0]);

fileNode.setFileid(file1.getId());

fileNode.setSuffix(fileName[1]);

// 提交数据库

fileNodeMapper.insertSelective(fileNode);

// 获取文件存储路径（绝对路径）

String path = servletConfig.getServletContext().getInitParameter("path");

// 创建文件实例

File filePath = new File(path, file1.getUuid());

// 如果文件目录不存在，创建目录

if (!filePath.getParentFile().exists()) {

filePath.getParentFile().mkdirs();

System.out.println("创建目录" + filePath);

}

// 写入文件

file.transferTo(filePath);

// 返回

Map res = new HashMap();

res.put("NodeID", fileNode.getPathnode());

res.put("NodeName", fileNode.getName() + fileNode.getSuffix());

res.put("Type", 1);

res.put("Childs", null);

return Msg.ParseMap(Msg.OK,"/transfer/upload",res);

}

}

package Controllor.Chain;

import Function.Msg;

import Model.ExtLink;

import Model.User;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.ResponseBody;

import javax.servlet.http.HttpSession;

import java.util.List;

@RequestMapping("/chain")

@Controller

public class GetAll extends Father {

/\*\*

\* 获取当前用户下的所有外链信息

\* @param session

\* @return

\*/

@RequestMapping("/getall")

public @ResponseBody

String getall(HttpSession session) {

User loginUser = (User) session.getAttribute("user");

// 验证

if (loginUser == null) {

return Msg.ParseList(Msg.ERR, "", null);

}

List<ExtLink> res = extLinkMapper.selectByUserID(loginUser.getId());

return Msg.ParseList(Msg.OK, "", res);

}

}

package Controllor.Chain;

import Function.MD5;

import Function.Msg;

import Model.ExtLink;

import Model.ExtObj;

import Model.User;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.ResponseBody;

import javax.servlet.http.HttpSession;

import java.text.SimpleDateFormat;

import java.util.\*;

@Controller

@RequestMapping("/chain")

public class NewChain extends Father {

@RequestMapping("/newchain")

public @ResponseBody

String newchain(Integer Day,

String Nodes,

boolean isCheck,

HttpSession session) {

User loginUser = (User) session.getAttribute("user");

List<ExtObj> NodeLists = new ArrayList<ExtObj>();

// 验证

try {

// 判空

if(Day==null||Nodes==null) throw new Exception(Msg.ERR.toString());

// 预处理需要复制的节点

for(String t : Nodes.split("\\|"))

{

if(t.length()<2) throw new Exception(Msg.ERR.toString());

ExtObj temp = new ExtObj();

temp.setType(t.substring(0, 1));

temp.setObjid( Integer.parseInt(t.substring(1)));

NodeLists.add(temp);

}

// 节点存在性和权限

boolean flag = true;

for (ExtObj n : NodeLists) {

if(!checkExisit(Integer.parseInt(n.getType()),n.getObjid())||

!CheckAuth(loginUser,Integer.parseInt(n.getType()),n.getObjid()))

{

flag=false;break;

}

}

if(!flag) throw new Exception("502");

} catch (Exception e) {

return Msg.ParseMap(Integer.parseInt(e.getMessage()), "", null);

}

// 处理

// 写入外链

ExtLink extLink = new ExtLink();

extLink.setUserid(loginUser.getId());

extLink.setName(MD5.getUUID());

if (Day > 0) {

Calendar calendar = Calendar.getInstance();

calendar.add(Calendar.DATE, Day);

extLink.setEndtime(calendar.getTime());

extLink.setType(0);

} else if (Day == 0) {

extLink.setType(-1);

} else {

extLink.setType(1);

}

if (isCheck == true) {

extLink.setPassword(MD5.getUUID().substring(0, 4));

}

extLinkMapper.insertSelective(extLink);

for (ExtObj n : NodeLists) {

n.setExtid(extLink.getId());

extObjMapper.insert(n);

}

// 返回

Map map = new HashMap();

map.put("UrlID", extLink.getName());

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

if(extLink.getEndtime()!=null)

map.put("EndTime", sdf.format(extLink.getEndtime()));

map.put("Password", extLink.getPassword());

return Msg.ParseMap(Msg.OK, "", map);

}

}

package Controllor.Chain;

import Dao.ExtLinkMapper;

import Dao.ExtObjMapper;

import Dao.FileNodeMapper;

import Dao.NodeMapper;

import Model.FileNode;

import Model.Node;

import Model.User;

import org.springframework.beans.factory.annotation.Autowired;

import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

public class Father {

@Autowired

protected NodeMapper nodeMapper;

@Autowired

protected ExtLinkMapper extLinkMapper;

@Autowired

protected ExtObjMapper extObjMapper;

@Autowired

protected FileNodeMapper fileNodeMapper;

/\*\*

\* 检查当前节点是否存在

\* @return

\*/

public boolean checkExisit(int Type,Integer NodeID) {

if (Type == 0) {

//查找目录存在

return nodeMapper.selectByPrimaryKey(NodeID)!=null;

}

else

{

//查找文件存在

return fileNodeMapper.selectByPrimaryKey(NodeID)!=null;

}

}

/\*\*

\* 获取当前节点是否有访问权限

\* 鉴权存在bug没有判定子账号是否有权限访问目录

\* @param user

\* @return

\*/

public boolean CheckAuth(User user,int Type,Integer NodeID) {

int NodeIDs;

if (Type == 0) {

NodeIDs = NodeID;

}

else

{

FileNode file = fileNodeMapper.selectByPrimaryKey(NodeID);

NodeIDs = file.getNodeid();

}

Node t = nodeMapper.selectNodeByNodeIDAndUserID(NodeIDs,

user.getParentid() == 0 ? user.getId() : user.getParentid());

return t != null;

}

public Map CreateRes(Integer NodeParentID, Integer NodeID, String NodeName,

int Type, Date UploadDate, Date ModifyDate, List ChildsID) {

Map res = new HashMap();

res.put("NodeParentID", NodeParentID);

res.put("NodeID", NodeID);

res.put("NodeName", NodeName);

res.put("Type", Type);

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

res.put("UploadDate", UploadDate==null?null:sdf.format(UploadDate));

res.put("ModifyDate", ModifyDate==null?null:sdf.format(ModifyDate));

res.put("ChildsID", ChildsID);

return res;

}

}

package Controllor.Chain;

import Function.Msg;

import Model.ExtLink;

import Model.ExtObj;

import Model.FileNode;

import Model.Node;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.ResponseBody;

import java.util.ArrayList;

import java.util.List;

import java.util.Map;

import java.util.Stack;

@Controller

@RequestMapping("/chain")

public class Get extends Father {

@RequestMapping("/get")

public @ResponseBody

String get(String UUID,

String Password) {

ExtLink cur = extLinkMapper.selectByName(UUID);

// 验证

try {

if (cur == null) {

throw new Exception(Msg.ERR.toString());

}

if (!cur.getPassword() .equals(Password) ) {

throw new Exception(Msg.NoAuth.toString());

}

} catch (Exception e) {

return Msg.ParseList(Integer.parseInt(e.getMessage()), "", null);

}

// 处理

List<ExtObj> objs = extObjMapper.selectAllExtobjByExLinkName(UUID);

// 读取文件或者目录信息

List<Map> res = new ArrayList<Map>();

for (ExtObj obj : objs) {

if (obj.getType() .equals("1")) {

// 处理文件

FileNode temp = fileNodeMapper.selectByPrimaryKey(obj.getObjid());

res.add(CreateRes(-1, temp.getPathnode()

, temp.getName(), 1, temp.getUploaddate(), temp.getModifydate(), null));

}

else

{

// 处理文件夹

Stack<Node> curNode = new Stack<Node>();

curNode.push(nodeMapper.selectByPrimaryKey(obj.getObjid()));

while (!curNode.empty()) {

Node c = curNode.pop();

List<Node> childNodes = nodeMapper.selectAllChildNode(c.getId());

List<Integer> childIDs = new ArrayList<Integer>();

for (Node child : childNodes) {

childIDs.add(child.getId());

}

// 添加当前节点

res.add(CreateRes(-1, c.getId(), c.getName(), 0,

null, null, childIDs));

// 将当前子节点压入堆栈

curNode.addAll(childNodes);

// 添加文件节点

List<FileNode> curFIles = fileNodeMapper.selectAllFilebyNodeID(c.getId());

for (FileNode f : curFIles) {

res.add(CreateRes(c.getId(), f.getPathnode(), f.getName() + f.getSuffix(),1,

f.getUploaddate(), f.getModifydate(), null));

}

}

}

}

return Msg.ParseList(Msg.OK, "", res);

}

}

package Controllor.User;

import Dao.NodeMapper;

import Dao.UserMapper;

import Function.MD5;

import Function.Msg;

import Function.Vaild;

import Model.ExtObj;

import Model.Node;

import Model.User;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpMethod;

import org.springframework.http.HttpRequest;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.\*;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

import java.io.IOException;

import java.util.List;

@Controller

@RequestMapping(value = "/user")

public class UserControllor {

@Autowired

private UserMapper userMapper;

@Autowired

private NodeMapper nodeMapper;

/\*\*

\* 用户注册

\* @param session

\* @param name

\* @param password

\* @param e\_mail

\* @param type

\* @param logicnode

\* @return

\*/

@RequestMapping(value = "/register",method = RequestMethod.POST)

public @ResponseBody

String register(HttpSession session,

@RequestParam(defaultValue = "") String name,

@RequestParam(defaultValue = "") String password,

@RequestParam(defaultValue = "") String e\_mail,

@RequestParam(defaultValue = "-1") int type,

@RequestParam(defaultValue = "-1") int logicnode) {

// 验证空

if (name == "" || password == "" || e\_mail == "" || type != 0&& type !=1) {

return Msg.ParseStr(Msg.ERR, "/user/register", "");

}

// 验证

try {

if(password.length()<8) throw new Exception("密码长度错误");

if(!Vaild.E\_mail(e\_mail)) throw new Exception("邮箱格式错误");

if (userMapper.selectByUserName(name).size() > 0 || userMapper.selectByEMail(e\_mail).size() > 0) {

throw new Exception("用户名或邮箱重复");

}

} catch (Exception e) {

return Msg.ParseStr(Msg.ERR, "/user/register", e.getMessage());

}

// 处理数据

User user = new User();

user.setPassword(MD5.getsec(password));

user.setValid(0);

user.setUsername(name);

user.seteMail(e\_mail);

Node node = new Node();

if (type == 0) {

// 主账号

node.setParentnode(0);

node.setName("/");

nodeMapper.insertSelective(node);//插入根目录信息

user.setLogicnode(node.getId());

user.setParentid(0);

}

else

{

// 子账号

User logUser = (User) session.getAttribute("user");

user.setParentid(logUser.getId());

user.setLogicnode(logicnode);

}

userMapper.insert(user);

if(user.getParentid()==0)

{

node.setUserid(user.getId());

nodeMapper.updateByPrimaryKey(node);

}

// 成功跳转登录

return Msg.ParseStr(Msg.OK, "/user/register", "注册成功");

}

/\*\*

\* 用于用户登录

\* @param session

\* @param name

\* @param password

\* @return

\*/

@RequestMapping(value = "/login", method = RequestMethod.POST)

public @ResponseBody

String login(HttpSession session,

@RequestParam(defaultValue = "") String name,

@RequestParam(defaultValue = "") String password) {

// 验证

if(name.equals("")||password.equals("")){

return Msg.ParseStr(Msg.ERR, "/user/login", "");

}

// 处理数据

List<User> getUsers = userMapper.selectByUserName(name);

if(getUsers.size()==0) return Msg.ParseStr(501, "/user/login", "");

if (getUsers.get(0).getPassword().compareTo(MD5.getsec(password)) ==0) {

// 登录成功

session.setAttribute("user", getUsers.get(0));

// 跳转到首页

}

else{

return Msg.ParseStr(501, "/user/login", "");

}

return Msg.ParseStr(Msg.OK, "/user/login", "");

}

/\*\*

\* 验证重复邮箱或者重复信息

\* @param value

\* @param type

\* @param request

\* @return

\*/

@RequestMapping(value = "/checkrepeat", method = RequestMethod.POST)

public @ResponseBody

String checkrepeat( @RequestParam(defaultValue = "") String value,

@RequestParam(defaultValue="-1") int type, HttpServletRequest request) {

// 验证

if(value==""||(type!=0&&type!=1)){

return Msg.ParseStr(Msg.ERR, "/user/checkrepeat", "");

}

// 处理

boolean flag=true;

if (type == 0) {

// 用户名判断

List<User> get = userMapper.selectByUserName(value);

if (get.size()>0) flag = false;

} else if (type == 1) {

// 邮箱判断

List<User> get = userMapper.selectByEMail(value);

if(get.size()>0) flag = false;

}

else

{

return Msg.ParseStr(Msg.ERR, "/user/checkrepeat", "");

}

// 返回结果

if (flag) {

return Msg.ParseStr(Msg.OK, "/user/checkrepeat", "");

}

else

{

return Msg.ParseStr(500, "/user/checkrepeat", "");

}

}

@RequestMapping(value = "/logout")

public @ResponseBody String logout(HttpSession session,HttpServletResponse response) throws IOException {

session.setAttribute("user",null);

response.sendRedirect("../index.jsp");

return "index";

}

}

package Controllor.Dir;

import Dao.ChildNodeMapper;

import Dao.NodeMapper;

import Function.Msg;

import Model.ChildNode;

import Model.Node;

import Model.User;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.ResponseBody;

import javax.servlet.http.HttpSession;

import javax.xml.transform.Templates;

import java.lang.reflect.Array;

import java.util.\*;

import java.util.prefs.PreferenceChangeEvent;

@Controller

@RequestMapping(value = "/dir")

public class DirControllor {

// region dependiency 依赖

@Autowired

private NodeMapper nodeMapper;

@Autowired

private ChildNodeMapper childNodeMapper;

// endregion

/\*\*

\* 获取当前节点下指定层次的目录，注意Node如果为0标识获取当前用户根目录

\* @param Deep

\* @param Node

\* @param session

\* @return

\*/

@RequestMapping(value = "/dir", method = RequestMethod.POST)

public @ResponseBody

String dir(@RequestParam(defaultValue = "1") int Deep,

int Node,

HttpSession session) {

// 验证

try {

// 验证空

if((Integer)Node==null) throw new Exception(((Integer) Msg.ERR).toString());

User loginUser = (User) session.getAttribute("user");

if(Node==0)

Node=loginUser.getLogicnode();

// 验证当前节点是否存在

if(nodeMapper.selectByPrimaryKey(Node)==null) throw new Exception(((Integer) Msg.ERR).toString());

// 验证当前节点是否有权限

if (!AuthCheck(Node, loginUser)) {

throw new Exception(((Integer) Msg.NoAuth).toString());

}

} catch (Exception e) {

return Msg.ParseMap(Integer.parseInt(e.getMessage()), "/dir/dir", null);

}

// 处理

List<Map> res = new ArrayList<Map>();

Stack<Integer> Nodequeue = new Stack<Integer>();

Nodequeue.push(Node);//推入初始节点

for (int i = 0; i <= Deep; i++) {

ArrayList<Integer> childID = new ArrayList<Integer>();

ArrayList<Integer> childID\_all = new ArrayList<Integer>();

while(!Nodequeue.empty())

{

int curNodeID = Nodequeue.pop();

Node cur = nodeMapper.selectByPrimaryKey(curNodeID);

// 获取当前节点下所有子节点

List<Node> t = nodeMapper.selectAllChildNode(curNodeID);

for (Node index : t) {

childID\_all.add(index.getId());

childID.add(index.getId());

}

// 将当前节点和子节点信息压入返回结果中

if(i!=0)

res.add(tool.resMap(cur.getId(), cur.getName(), childID\_all, cur.getParentnode()));

}

// 保存下一层级子元素

Nodequeue.addAll(childID);

}

// 映射结果集，返回

return Msg.ParseList(Msg.OK, "/dir/dir", res);

}

/\*\*

\* 新建一个目录

\* @param CurNodeID

\* @param NewNodeName

\* @param session

\* @return

\*/

@RequestMapping(value = "/newdir", method = RequestMethod.POST)

public @ResponseBody

String newdir(Integer CurNodeID,

String NewNodeName,

HttpSession session) {

// 验证

User loginUser = (User) session.getAttribute("user");

try {

// 验证空

if (CurNodeID == null || NewNodeName == null) throw new Exception(Msg.ERR.toString());

if(CurNodeID==0) CurNodeID = loginUser.getLogicnode();

// 验证用户

if (!AuthCheck(CurNodeID,loginUser)) throw new Exception(Msg.NoAuth.toString());

// 验证重名

if(nodeMapper.selectChildByNodeIDAndNodeName(CurNodeID,NewNodeName)!=null)

throw new Exception("501");

} catch (Exception e) {

return Msg.ParseMap((Integer.parseInt(e.getMessage())), "/dir/newdir", null);

}

// 处理

Node node = new Node();

node.setUserid(loginUser.getParentid() == 0 ? loginUser.getId() : loginUser.getParentid());

node.setName(NewNodeName);

node.setParentnode(CurNodeID);

// 开始事务

nodeMapper.insert(node);

ChildNode childNode = new ChildNode();

childNode.setParentid(CurNodeID);

childNode.setChildid(node.getId());

childNodeMapper.insert(childNode);

// 返回

Map res = new HashMap();

res.put("NodeID", node.getId());

res.put("NodeName", node.getName());

return Msg.ParseMap(Msg.OK, "/dir/newdir", res);

}

/\*\*

\* 修改节点名称

\* @param NodeID

\* @param NodePreName

\* @param NodeNewName

\* @param session

\* @return

\*/

@RequestMapping(value = "/modifydirname", method = RequestMethod.POST)

public @ResponseBody

String modifydirname(Integer NodeID,

String NodePreName,

String NodeNewName,

HttpSession session) {

User loginUser = (User) session.getAttribute("user");

// 验证

try {

// 验空

if(NodeID==null||NodePreName==null||NodeNewName==null) throw new Exception(Msg.ERR.toString());

if(NodeID==0) NodeID = loginUser.getLogicnode();

// 节点存在性

Node curNode = nodeMapper.selectByPrimaryKey(NodeID);

if(curNode==null) throw new Exception("501");

// 当前用户没有权限

if(!AuthCheck(NodeID,loginUser)) throw new Exception(Msg.NoAuth.toString());

if (!curNode.getName().equals(NodePreName) ) {

throw new Exception("501");

}

// 存在重名

if(nodeMapper.selectChildByNodeIDAndNodeName(curNode.getParentnode(),NodeNewName)!=null)

throw new Exception("502");

} catch (Exception e) {

return Msg.ParseMap(Integer.parseInt(e.getMessage()), "/dir/modifydirname", null);

}

// 处理

Node node = new Node();

node.setId(NodeID);

node.setName(NodeNewName);

nodeMapper.updateByPrimaryKeySelective(node);

// 返回

Map res = new HashMap();

res.put("NodeID", node.getId());

res.put("NodeNewName", node.getName());

return Msg.ParseMap(Msg.OK, "/dir/modifydirname", res);

}

/\*\*

\* 移动某个节点的位置

\* @param CurNodeID

\* @param PreFatherNodeID

\* @param NowFatherNodeID

\* @param session

\* @return

\*/

@RequestMapping(value = "/modifydirposition", method = RequestMethod.POST)

public @ResponseBody

String modifydirposition(Integer CurNodeID,

Integer PreFatherNodeID,

Integer NowFatherNodeID,

HttpSession session) {

User loginUser = (User) session.getAttribute("user");

Node node = nodeMapper.selectByPrimaryKey(CurNodeID);

// 验证

try {

// 验证空

if (CurNodeID == null || PreFatherNodeID == null || NowFatherNodeID == null) {

throw new Exception(Msg.ERR.toString());

}

// 验证一致性

if (node.getParentnode() != PreFatherNodeID) {

throw new Exception(Msg.ERR.toString());

}

// 验证节点是否存在

if(nodeMapper.selectByPrimaryKey(CurNodeID)==null||

nodeMapper.selectByPrimaryKey(PreFatherNodeID)==null||

nodeMapper.selectByPrimaryKey(NowFatherNodeID)==null)

throw new Exception("501");

// 权限验证

if(!AuthCheck(CurNodeID,loginUser)||!AuthCheck(PreFatherNodeID,loginUser)||!AuthCheck(NowFatherNodeID,loginUser))

throw new Exception(Msg.NoAuth.toString());

} catch (Exception e) {

return Msg.ParseMap(Integer.parseInt(e.getMessage()), "/dir/modifydirposition", null);

}

// 处理

// 更新父节点记录

node.setParentnode(NowFatherNodeID);

nodeMapper.updateByPrimaryKey(node);

// 更新子节点记录

ChildNode childNode = childNodeMapper.selectByParentChild(PreFatherNodeID, CurNodeID);

childNode.setParentid(NowFatherNodeID);

childNodeMapper.updateByPrimaryKey(childNode);

// 返回

Map res = new HashMap();

res.put("NowFatherNodeID", childNode.getParentid());

return Msg.ParseMap(Msg.OK, "/dir/modifydirposition", res);

}

// region tool 工具

/\*\*

\* 判断当前账号是否拥有节点权限

\* @param nodeID

\* @param loginUser

\* @return ture,有权限 false，没有权限

\*

\*/

public boolean AuthCheck(int nodeID,User loginUser)

{

//账号为子账号

if(loginUser.getParentid()!=0)

{ //验证主账号是否有权限

if(nodeMapper.selectNodeByNodeIDAndUserID(nodeID,loginUser.getParentid())==null)

return true;

// 递归验证子账号是否有权限

Node Curnode = nodeMapper.selectByPrimaryKey(nodeID);

Node t=Curnode;

do{

t= nodeMapper.selectByPrimaryKey(t.getParentnode());

}

while (t.getId()!=loginUser.getLogicnode()&&t.getParentnode()!=loginUser.getLogicnode());

// 考虑如果子账号获取主账号所有权限时

if(t.getParentnode()==loginUser.getLogicnode() && t.getId()!=loginUser.getLogicnode() )

return false;

else

return true;

}

//账号为主账号

else

{

return !(nodeMapper.selectNodeByNodeIDAndUserID(nodeID,loginUser.getId())==null);

}

}

// endregion

}

package Controllor.Dir;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

public class tool {

/\*\*

\* 用于目录结果集的构建，返回一个键值对

\* @param NodeID

\* @param NodeName

\* @param Childs

\* @param Father

\* @return

\*/

public static Map resMap(int NodeID, String NodeName, List<Integer> Childs, int Father) {

HashMap res = new HashMap();

res.put("NodeID", NodeID);

res.put("NodeName", NodeName);

res.put("Childs", Childs);

res.put("Father", Father);

return res;

}

}

package Controllor.hello;

import Dao.UserMapper;

import Function.Msg;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.stereotype.Service;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.ResponseBody;

import org.springframework.web.servlet.ModelAndView;

import javax.servlet.http.HttpServletResponse;

@Controller

@Service

@RequestMapping("/hello")

public class hello {

@RequestMapping("/RetJsp")

public String hello() {

return "hello";

}

@RequestMapping("/ret")

public ModelAndView ret() {

ModelAndView modelAndView = new ModelAndView("hello");

modelAndView.addObject("message", "hello");

return modelAndView;

}

@RequestMapping(value = "/reason")

public @ResponseBody

String user(HttpServletResponse response) {

response.setCharacterEncoding("utf-8");

return "这是一个测试";

}

@Autowired

UserMapper userMapper;

@RequestMapping(value = "/reasonv")

public @ResponseBody String user1() {

return Msg.ParseMap(Msg.OK, "test", userMapper.selectByPrimaryKey(6));

}

}

package Controllor.fitter;

import Function.Msg;

import org.springframework.stereotype.Component;

import org.springframework.web.servlet.HandlerInterceptor;

import org.springframework.web.servlet.ModelAndView;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

import java.io.IOException;

import java.io.PrintWriter;

public class AuthFitter implements HandlerInterceptor {

//拦截器可以使用spring的依赖注入

public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {

//可以在这里处理用户认证

HttpSession session = request.getSession();

if (session.getAttribute("user") == null) {

// 用户没有登录

response.setCharacterEncoding("utf-8");

PrintWriter out = response.getWriter();

out.print(Msg.ParseStr(Msg.LoginAuth, "fitter", "用户没有登录"));

// 执行跳转

response.sendRedirect("../index.jsp");

return false;

}

return true;

}

public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler, ModelAndView modelAndView) throws Exception {

}

//所有请求完成之后调用

public void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex) throws Exception {

System.out.println("所有请求完成之后调用");

}

}

package Controllor.fitter;

import Dao.LogMapper;

import Model.User;

import com.alibaba.fastjson.JSON;

import org.springframework.web.portlet.handler.HandlerInterceptorAdapter;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

import java.util.Enumeration;

import java.util.HashMap;

public class Log extends HandlerInterceptorAdapter {

// 请求信息

public boolean preHandle(HttpServletRequest request,

HttpServletResponse response, Object handler, LogMapper logMapper)throws Exception {

HashMap req = new HashMap();

HashMap params = new HashMap();

Enumeration t = request.getParameterNames();

while(t.hasMoreElements())

{

String temp = (String) t.nextElement();

params.put(temp, request.getParameter(temp));

}

req.put("title", request.getRequestURL());

req.put("params", params);

// 存入session中

HttpSession session = request.getSession();

// 存入数据库

Model.Log log = new Model.Log();

log.setUserid(((User) session.getAttribute("user")).getId());

log.setRequest(JSON.toJSONString(req));

logMapper.insertSelective(log);

// 获取列表

session.setAttribute("req", req);

return true;

}

// 返回信息

public void afterCompletion(HttpServletRequest request,

HttpServletResponse response, Object handler,

Exception ex) throws Exception {

}

}

package Controllor.Child;

import Dao.NodeMapper;

import Dao.UserMapper;

import Function.MD5;

import Function.Msg;

import Function.Vaild;

import Model.User;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.objenesis.instantiator.annotations.Instantiator;

import org.springframework.stereotype.Controller;

import org.springframework.transaction.annotation.Transactional;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.ResponseBody;

import javax.servlet.http.HttpSession;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

@Controller

@RequestMapping(value = "/child")

public class ChildControllor {

@Autowired

private UserMapper userMapper;

@Autowired

private NodeMapper nodeMapper;

/\*\*

\* 获取某个用户下所有子账号信息

\* @param session

\* @param UserName 用户名称用于检测登录的一致性

\* @return

\*/

@Transactional(rollbackFor = Throwable.class)//配置spring事务，出现错误则自动滚回

@RequestMapping(value = "/getchilds", method = RequestMethod.POST)

public @ResponseBody String getchilds(HttpSession session,

@RequestParam(defaultValue = "") String UserName)

{

// 验证

User loginUser = (User) session.getAttribute("user");

if(UserName.equals("")) UserName = loginUser.getUsername();

if (loginUser==null||!loginUser.getUsername().equals(UserName))

return Msg.ParseList(Msg.LoginAuth, "/child/getchilds", new ArrayList());

if (loginUser.getParentid()!=0)

return Msg.ParseList(557,"/child/getchilds",new ArrayList());

// 处理数据

List<User> getlist = userMapper.selectChildByParentID(loginUser.getId());

ArrayList<Map> res = new ArrayList<Map>();

for (User user : getlist) {

Map temp = new HashMap();

temp.put("name", user.getUsername());

temp.put("logicnode", user.getLogicnode());

temp.put("logicanodeName", nodeMapper.selectByPrimaryKey(user.getLogicnode()));

temp.put("e\_mail", user.geteMail());

res.add(temp);

}

// 返回结果

return Msg.ParseList(Msg.OK, "/child/getchilds", res);

}

/\*\*

\* 修改子账号信息

\* @param ChildName

\* @param LogicNode

\* @param Password

\* @param E\_mail

\* @param CheckPassword

\* @param session

\* @return

\*/

@RequestMapping(value = "/modifychildinfo",method = RequestMethod.POST)

public @ResponseBody

String modifychildinfo( String ChildName,

Integer LogicNode,

String Password,

String E\_mail,

String CheckPassword,

HttpSession session

) {

// 验证

if (ChildName == null || (LogicNode== null && Password == null && E\_mail == null )|| CheckPassword == null) {

return Msg.ParseList(Msg.ERR, "/child/modifychildinfo", new ArrayList());

}

User loginUser = (User) session.getAttribute("user");

try {

if (Password!=null&&Password.length()<8) throw new Exception("503");

if(E\_mail!=null&&!Vaild.E\_mail(E\_mail)) throw new Exception("502");

if (!MD5.getsec(CheckPassword) .equals(loginUser.getPassword()) ) throw new Exception("500");

if(loginUser.getParentid()!=0) throw new Exception(((Integer) Msg.NoAuth).toString());

if(LogicNode!=null&&nodeMapper.selectByPrimaryKey(LogicNode)==null) throw new Exception("501");

} catch (Exception e) {

return Msg.ParseList(Integer.parseInt(e.getMessage()), "/child/modifychildinfo", null);

}

// 处理

List<User> getlist = userMapper.selectByUserName(ChildName);

if(getlist.size()==0)

{

// 子账号不存在

return Msg.ParseList(504,"/child/modifychildinfo",null);

}

ArrayList<Map> res = new ArrayList<Map>();

User user = getlist.get(0);

if(E\_mail!=null) {

user.seteMail(E\_mail);

Map temp = new HashMap();

temp.put("key", "E\_mail");

temp.put("value", E\_mail);

res.add(temp);

}

if(Password!=null)

{

user.setPassword(MD5.getsec(Password));

user.seteMail(Password);

Map temp = new HashMap();

temp.put("key", "Password");

temp.put("value", Password);

res.add(temp);

}

if(LogicNode!=null) {

user.setLogicnode(LogicNode);

Map temp = new HashMap();

temp.put("key", "LogicNode");

temp.put("value", LogicNode);

res.add(temp);

}

// 更新用户信息

if (userMapper.updateByPrimaryKey(user) == 1) {

return Msg.ParseList(Msg.OK, "/child/modifychildinfo", res);

}

else

{

return Msg.ParseList(Msg.ERR, "", null);

}

}

/\*\*

\* 删除子账号

\* @param ChildName

\* @param Password

\* @param session

\* @return

\*/

@RequestMapping(value = "/delchild", method = RequestMethod.POST)

public @ResponseBody String delchild(String ChildName,

String Password,

HttpSession session) {

User loginUser = (User)session.getAttribute("user");

String UserName = loginUser.getUsername();

// 验证

try {

if (UserName == null || Password == null || ChildName == null)

throw new Exception(((Integer) Msg.ERR).toString());

if (!loginUser.getUsername().equals(UserName))

throw new Exception(((Integer) Msg.LoginAuth).toString());

if(loginUser.getParentid()!=0)

throw new Exception(((Integer) Msg.NoAuth).toString());

if(!MD5.getsec(Password) .equals(loginUser.getPassword()))

throw new Exception("503");

} catch (Exception e) {

return Msg.ParseStr(Integer.parseInt(e.getMessage()), "/child/delchild", "");

}

// 处理

if (userMapper.deleteByUserName(ChildName) == 1) {

return Msg.ParseStr(Msg.OK, "/child/delchild", "");

}

else

return Msg.ParseStr(Msg.ERR, "/child/delchild", "");

}

/\*\*

\* 查看当前子账号集合中是否包含

\* @param list

\* @param ChildID

\* @return boolean true标识包含在内，false标识不包含在内

\*/

public boolean isChildInclude(List<User> list, int ChildID) {

for (User user : list) {

if(user.getId()==ChildID) return true;

}

return false;

}

}

package Model;

public class Node {

private Integer id;

private String name;

private Integer parentnode;

private Integer userid;

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name == null ? null : name.trim();

}

public Integer getParentnode() {

return parentnode;

}

public void setParentnode(Integer parentnode) {

this.parentnode = parentnode;

}

public Integer getUserid() {

return userid;

}

public void setUserid(Integer userid) {

this.userid = userid;

}

}

package Model;

import java.util.Date;

public class FileNode {

private Integer pathnode;

private Integer fileid;

private Integer nodeid;

private String name;

private String suffix;

private Date uploaddate;

private Date modifydate;

public Integer getPathnode() {

return pathnode;

}

public void setPathnode(Integer pathnode) {

this.pathnode = pathnode;

}

public Integer getFileid() {

return fileid;

}

public void setFileid(Integer fileid) {

this.fileid = fileid;

}

public Integer getNodeid() {

return nodeid;

}

public void setNodeid(Integer nodeid) {

this.nodeid = nodeid;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name == null ? null : name.trim();

}

public String getSuffix() {

return suffix;

}

public void setSuffix(String suffix) {

this.suffix = suffix == null ? null : suffix.trim();

}

public Date getUploaddate() {

return uploaddate;

}

public void setUploaddate(Date uploaddate) {

this.uploaddate = uploaddate;

}

public Date getModifydate() {

return modifydate;

}

public void setModifydate(Date modifydate) {

this.modifydate = modifydate;

}

}

package Model;

import com.alibaba.fastjson.JSON;

public class User {

private Integer id;

private String username;

private String password;

private Integer parentid;

private Integer logicnode;

private String eMail;

private Integer valid;

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username == null ? null : username.trim();

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password == null ? null : password.trim();

}

public Integer getParentid() {

return parentid;

}

public void setParentid(Integer parentid) {

this.parentid = parentid;

}

public Integer getLogicnode() {

return logicnode;

}

public void setLogicnode(Integer logicnode) {

this.logicnode = logicnode;

}

public String geteMail() {

return eMail;

}

public void seteMail(String eMail) {

this.eMail = eMail == null ? null : eMail.trim();

}

public Integer getValid() {

return valid;

}

public void setValid(Integer valid) {

this.valid = valid;

}

public String toString() {

return JSON.toJSONString(this);

}

}

package Model;

import java.util.Date;

public class File {

private Integer id;

private String uuid;

private String md5;

private Integer point;

private Integer size;

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getUuid() {

return uuid;

}

public void setUuid(String uuid) {

this.uuid = uuid == null ? null : uuid.trim();

}

public String getMd5() {

return md5;

}

public void setMd5(String md5) {

this.md5 = md5 == null ? null : md5.trim();

}

public Integer getPoint() {

return point;

}

public void setPoint(Integer point) {

this.point = point;

}

public Integer getSize() {

return size;

}

public void setSize(Integer size) {

this.size = size;

}

}

package Model;

public class ChildNode {

private Integer id;

private Integer parentid;

private Integer childid;

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public Integer getParentid() {

return parentid;

}

public void setParentid(Integer parentid) {

this.parentid = parentid;

}

public Integer getChildid() {

return childid;

}

public void setChildid(Integer childid) {

this.childid = childid;

}

}

package Model;

import java.util.Date;

public class Log {

private Integer id;

private Integer userid;

private String request;

private Date time;

private String response;

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public Integer getUserid() {

return userid;

}

public void setUserid(Integer userid) {

this.userid = userid;

}

public String getRequest() {

return request;

}

public void setRequest(String request) {

this.request = request == null ? null : request.trim();

}

public Date getTime() {

return time;

}

public void setTime(Date time) {

this.time = time;

}

public String getResponse() {

return response;

}

public void setResponse(String response) {

this.response = response == null ? null : response.trim();

}

}