package org.deta.boot.controller;

import java.util.Map;

import org.lyg.stable.StableData;

import org.lyg.vpc.process.portImpl.RestDBConfigImpl;

import mapProcessor.VtoV;

public class ConfigController {

public static String exec(String string, Map<String, String> data)

throws Exception {

if(string.equalsIgnoreCase(StableData.REST\_PATH\_SET\_DB\_PATH)){

return VtoV.ObjectToJsonString(RestDBConfigImpl.setDBPath(data

.get("baseName")

, data.get("baseName"), data.get("baseName")));

}

if(string.equalsIgnoreCase(StableData.REST\_PATH\_SET\_DB\_TABLE)){

return VtoV.ObjectToJsonString(RestDBConfigImpl.setDBTable(data

.get("tableName")

, data.get("token"), data.get("auth")));

}

return "";

}

}

package org.deta.boot.controller;

import java.util.Map;

import mapProcessor.VtoV;

import org.lyg.stable.StableData;

import org.lyg.vpc.process.portImpl.RestControllerPortImpl;

public class DBCategoryController {

public static String exec(String string, Map<String, String> data)

throws Exception {

if(string.equalsIgnoreCase(StableData.REST\_GET\_DB\_CATEGORY)){

return VtoV.ObjectToJsonString(RestControllerPortImpl

.getDBCategory(data.get(StableData.DB\_BASE\_NAME)

, data.get(StableData.LOGIN\_TOKEN),data

.get(StableData.LOGIN\_AUTH)));

}

if(string.equalsIgnoreCase(StableData.REST\_GET\_ALL\_DB\_CATEGORY)){

return VtoV.ObjectToJsonString(RestControllerPortImpl

.getAllDBCategory(data.get(StableData.LOGIN\_TOKEN)

, data.get(StableData.LOGIN\_AUTH)));

}

return StableData.STRING\_EMPTY;

}

}

package org.deta.boot.controller;

import java.util.Map;

import mapProcessor.VtoV;

import org.lyg.stable.StableData;

import org.lyg.vpc.process.portImpl.RestDBDeleteImpl;

public class DeleteController {

public static String exec(String string, Map<String, String> data)

throws Exception {

if(string.equalsIgnoreCase(StableData.REST\_PATH\_DELETE\_ROWS\_BY\_TABLE\_PATH\_AND\_INDEX)){

return VtoV.ObjectToJsonString(RestDBDeleteImpl

.deleteRowByTablePathAndIndex(data.get("tablePath")

, data.get("pageIndex"), data.get("token")

, data.get("email"), data.get("password"), data.get("auth")));

}

return StableData.STRING\_EMPTY;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.deta.boot.controller;

import java.util.Map;

import mapProcessor.VtoV;

import org.lyg.stable.StableData;

import org.lyg.vpc.process.portImpl.RestDBInsertImpl;

public class InsertController {

public static String exec(String string, Map<String, String> data)

throws Exception {

if(string.equalsIgnoreCase(StableData.REST\_PATH\_INSERT\_ROW\_BY\_BASE\_NAME)){

return VtoV.ObjectToJsonString(RestDBInsertImpl

.insertRowByBaseName(data.get("baseName")

, data.get("tableName"), data.get("culumnOfNewRow")

, data.get("token"), data.get("email")

, data.get("password"), data.get("auth")));

}

if(string.equalsIgnoreCase(StableData.REST\_PATH\_INSERT\_ROW\_BY\_TABLE\_PATH)){

return VtoV.ObjectToJsonString(RestDBInsertImpl

.insertRowByTablePath(data.get("tablePath")

, data.get("pageIndex"), data.get("culumnOfNewRow")

, data.get("token"), data.get("email")

, data.get("password"), data.get("auth")));

}

return StableData.STRING\_EMPTY;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.deta.boot.controller;

import java.util.Map;

import mapProcessor.VtoV;

import org.lyg.stable.StableData;

import org.lyg.vpc.process.portImpl.RestDBSelectImpl;

public class SelectController {

public static String exec(String string, Map<String, String> data)

throws Exception {

if(string.equalsIgnoreCase(StableData.REST\_PATH\_SELECT\_ROWS\_BY\_ATTRIBUTE)){

return VtoV.ObjectToJsonString(RestDBSelectImpl

.selectRowsByAttribute(data.get("baseName")

, data.get("tableName"), data.get("culumnName")

, data.get("value"), data.get("token")

, data.get("email"), data.get("password"), data.get("auth")));

}

if(string.equalsIgnoreCase(StableData.REST\_PATH\_SELECT\_ROWS\_BY\_TABLE\_PATH)){

return VtoV.ObjectToJsonString(RestDBSelectImpl

.selectRowsByTablePath(data.get("tablePath")

, data.get("pageBegin"), data.get("pageEnd")

, data.get("direction"), data.get("token")

, data.get("email"), data.get("password"), data.get("auth")));

}

return StableData.STRING\_EMPTY;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.deta.boot.controller;

import java.util.Map;

import mapProcessor.VtoV;

import org.lyg.stable.StableData;

import org.lyg.vpc.process.portImpl.RestDBUpdateImpl;

public class UpdateController {

public static String exec(String string, Map<String, String> data)

throws Exception {

if(string.equalsIgnoreCase(StableData.REST\_PATH\_UPDATE\_ROW\_BY\_TABLE\_PATH\_AND\_INDEX)){

return VtoV.ObjectToJsonString(RestDBUpdateImpl

.updateRowByTablePathAndIndex(data.get("tablePath")

, data.get("pageIndex"), data.get("culumnOfUpdateRow")

, data.get("token"), data.get("email")

, data.get("password"), data.get("auth")));

}

return StableData.STRING\_EMPTY;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.deta.boot.rest;

import java.io.File;

import java.io.IOException;

import java.util.Map;

import org.deta.boot.controller.ConfigController;

import org.deta.boot.controller.DBCategoryController;

import org.deta.boot.controller.DeleteController;

import org.deta.boot.controller.InsertController;

import org.deta.boot.controller.SelectController;

import org.deta.boot.controller.UpdateController;

import org.lyg.stable.StableData;

import org.lyg.vpc.process.portImpl.RestDBPLSQLImpl;

import org.lyg.vpc.process.portImpl.RestLoginPortImpl;

import mapProcessor.VtoV;

public class VPC {

public static String forward(String string, Map<String, String> data)

throws Exception {

//controller

if(string.contains(StableData.REST\_PATH\_SELECT)){

return SelectController.exec(string, data);

}

if(string.contains(StableData.REST\_PATH\_SETDB)){

return ConfigController.exec(string, data);

}

if(string.contains(StableData.REST\_PATH\_INSERT)){

return InsertController.exec(string, data);

}

if(string.contains(StableData.REST\_PATH\_DELETE)){

return DeleteController.exec(string, data);

}

if(string.contains(StableData.REST\_PATH\_UPDATE)){

return UpdateController.exec(string, data);

}

if(string.contains(StableData.REST\_PATH\_DB\_CATEGORY)){

return DBCategoryController.exec(string, data);

}

//plsql

if(string.equalsIgnoreCase(StableData.REST\_PATH\_EXEC\_DETA\_PLSQL)){

return VtoV.ObjectToJsonString(RestDBPLSQLImpl.restDBPLSQLImpl(data

.get("token") ,data.get("email"), data.get("password"), data.get("auth")

, data.get("LYGQuery"), data.get("mod")));

}

//restMap

if(string.equalsIgnoreCase(StableData.REST\_PATH\_LOGIN)){

return VtoV.ObjectToJsonString(RestLoginPortImpl.login(data.get("uEmail")

, data.get("uPassword")));

}

if(string.equalsIgnoreCase(StableData.REST\_PATH\_FIND)){

return VtoV.ObjectToJsonString(RestLoginPortImpl.find(data.get("uEmail")));

}

if(string.equalsIgnoreCase(StableData.REST\_PATH\_LOGOUT)){

return VtoV.ObjectToJsonString(RestLoginPortImpl.logout(data.get("uEmail")

, data.get("uToken")));

}

if(string.equalsIgnoreCase(StableData.REST\_PATH\_REGISTER)){

return VtoV.ObjectToJsonString(RestLoginPortImpl.register(data.get("uEmail")

, data.get("uEmailEnsure")

, data.get("uName"), data.get("uPassword"), data.get("uPassWDEnsure")

, data.get("uAddress")

, data.get("uPhone"), data.get("uWeChat"), data.get("uQq"), data.get("uAge")

, data.get("uSex")));

}

if(string.equalsIgnoreCase(StableData.REST\_PATH\_CHANGE)){

return VtoV.ObjectToJsonString(RestLoginPortImpl.change(data.get("uEmail")

, data.get("uChange")

, data.get("uChangeEnsure"),data.get("uToken"), data.get("uPassword")));

}

if(string.equalsIgnoreCase(StableData.REST\_PATH\_CHECK\_STATUS)){

return VtoV.ObjectToJsonString(RestLoginPortImpl.checkStatus(data.get("token")));

}

return StableData.STRING\_EMPTY;

}

public static String getCode(String filePath) throws IOException{

if(filePath.contains(StableData.FILE\_HTML)||filePath.contains(StableData.FILE\_JS)){

return "UTF-8";

}

return "GBK";

}

public static String getFilePath(String string) {

String root = new File("src/main/resources/static").getAbsolutePath().replace("\\", "/");

if(string.equalsIgnoreCase("")||string.equalsIgnoreCase("/")){

string = "/index.html";

}

return root + string;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.deta.boot.server;

import java.io.IOException;

import org.deta.boot.vpc.controller.ServerInitController;

public class BootVPCS {

public static void main(String[] args) throws IOException {

ServerInitController.initServer();

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.deta.boot.vpc.controller;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileInputStream;

import java.io.IOException;

import java.io.InputStreamReader;

import java.net.InetAddress;

import java.net.Socket;

import java.util.Map;

import java.util.concurrent.ConcurrentHashMap;

import org.deta.boot.vpc.vision.VPCSRequest;

import org.deta.boot.vpc.vision.VPCSResponse;

import org.lyg.common.utils.DetaDBUtil;

@SuppressWarnings("unused")

public class RequestFilterController {

static Map<String, Boolean> ipBlock;

public static void main(String[] args){

}

public static void requestIpFilter(Socket socket) {

// TODO Auto-generated method stub

}

public static void requestLinkFilter(Socket socket) {

// TODO Auto-generated method stub

}

public static void requestIpFilter(VPCSRequest vPCSRequest, VPCSResponse vPCSResponse) throws IOException {

//过滤block

if(ipBlock.containsKey(vPCSRequest.getRequestIp())){

vPCSResponse.returnErrorCode(403);

}

//同源csrf

if(vPCSRequest.getRequestIp().equalsIgnoreCase(InetAddress.getLocalHost().getHostAddress())){

vPCSResponse.returnErrorCode(405);

}

}

public static void requestLinkFilter(VPCSRequest vpcsRequest, VPCSResponse vPCSResponse) throws IOException {

// //限制ddos

// String ipCount = DetaDBUtil.cacheRequest("get?key=" + vpcsRequest.getRequestIp() + "&email="

// + "313699483@qq.com" + "&password=" + "Fengyue1985!");

//

// if(null == ipCount){

// DetaDBUtil.cacheRequest("put?key=" + vpcsRequest.getRequestIp() + "&value=" + "1" + "&time="

// + (2000+System.currentTimeMillis()) + "&email=" + "313699483@qq.com" + "&password=" + "Fengyue1985!");

// }else if(ipCount.contains("unsuccess")){

// DetaDBUtil.cacheRequest("put?key=" + vpcsRequest.getRequestIp() + "&value=" + "1" + "&time="

// + (2000+System.currentTimeMillis()) + "&email=" + "313699483@qq.com" + "&password=" + "Fengyue1985!");

// }else if(Integer.valueOf(ipCount) <= 30){

// int digit = Integer.valueOf(ipCount) + 1;

// DetaDBUtil.cacheRequest("put?key=" + vpcsRequest.getRequestIp() + "&value=" + digit + "&time="

// + 0 + "&email=" + "313699483@qq.com" + "&password=" + "Fengyue1985!");

// }else{

// vPCSResponse.returnErrorCode(400);

// }

}

public static void initBlockList() throws IOException {

ipBlock = new ConcurrentHashMap<>();

FileInputStream fileInputStream = new FileInputStream(

new File("src/main/resources/ipBlock.ips"));

InputStreamReader inputStreamReader = new InputStreamReader(fileInputStream, "UTF-8");

BufferedReader bufferedReader = new BufferedReader(inputStreamReader);

String line = null;

while ((line = bufferedReader.readLine()) != null) {

ipBlock.put(line, true);

}

bufferedReader.close();

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.deta.boot.vpc.controller;

import java.net.Socket;

import org.deta.boot.vpc.vision.VPCSRequest;

import org.deta.boot.vpc.vision.VPCSResponse;

public class RequestFixController {

public static void main(String[] args){

}

public static void requestIpFix(Socket socket) {

}

public static void requestLinkFix(Socket socket) {

}

public static void requestIpFix(VPCSRequest vPCSRequest

, VPCSResponse vPCSResponse) {

}

public static void requestLinkFix(VPCSRequest vPCSRequest

, VPCSResponse vPCSResponse) {

}

}  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.deta.boot.vpc.controller;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.net.URLDecoder;

import java.util.Map;

import java.util.concurrent.ConcurrentHashMap;

import org.deta.boot.vpc.vision.VPCSRequest;

import org.deta.boot.vpc.vision.VPCSResponse;

import org.lyg.stable.StableData;

public class RequestRecordController {

public static void requestIpRecoder(VPCSRequest vPCSRequest, VPCSResponse vPCSResponse) {

vPCSRequest.setRequestIp(vPCSResponse.getSocket().getInetAddress().getHostAddress());

vPCSRequest.setRequestName(vPCSResponse.getSocket().getInetAddress().getHostName());

}

public static void requestLinkRecoder(VPCSRequest vPCSRequest, VPCSResponse vPCSResponse)

throws Exception {

BufferedReader br = new BufferedReader(new InputStreamReader(vPCSResponse.getSocket()

.getInputStream(), StableData.CHARSET\_GBK));

String mess = br.readLine();

if(null == mess){

vPCSResponse.returnErrorCode(StableData.HTTP\_400);

return;

}

if(mess.equalsIgnoreCase(StableData.STRING\_EMPTY)){

vPCSResponse.returnErrorCode(StableData.HTTP\_400);

return;

}

String[] type = mess.split(StableData.STRING\_SPACE);

if(type.length < StableData.INT\_TWO){

vPCSResponse.returnErrorCode(StableData.HTTP\_500);

return;

}

String[] content = type[StableData.INT\_ONE].split(StableData.STRING\_SLASH\_QUESTION);

if(content.length == StableData.INT\_TWO){

vPCSRequest.setRequestIsRest(true);

if(content[StableData.INT\_ONE] == null){

vPCSResponse.returnErrorCode(StableData.HTTP\_500);

return;

}

}

if(content[StableData.INT\_ZERO].contains(StableData.STRING\_QUATE)){

// vPCSRequest.setRequestIsRest(false);

}

if(vPCSRequest.getRequestIsRest()){

String[] column = content[StableData.INT\_ONE].split(StableData.STRING\_JUNCTION);

Map<String, String> data = new ConcurrentHashMap<>();

for(String cell:column){

String[] cells = cell.split(StableData.MATH\_EQUAL);

data.put(cells[StableData.INT\_ZERO], URLDecoder.decode(cells[StableData.INT\_ONE]

, StableData.CHARSET\_UTF\_8));

}

vPCSRequest.setRequestValue(data);

}

vPCSRequest.setRequestLink(content[StableData.INT\_ZERO]);

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.deta.boot.vpc.controller;

import java.io.IOException;

import java.io.PrintWriter;

import java.net.Socket;

import org.deta.boot.vpc.sleeper.SleeperHall;

public class ResponseController {

public static void main(String[] args){

}

@SuppressWarnings("unused")

private static void error404(Socket socket, SleeperHall sleeperHall

, Integer sId) throws IOException {

PrintWriter pw=new PrintWriter(socket.getOutputStream(),true);

pw.println("HTTP/1.1 404 OK\n\n");

pw.flush();

pw.close();

socket.close();

sleeperHall.removeThreadById(sId);

return;

}

@SuppressWarnings("unused")

private static void error500(Socket socket, SleeperHall sleeperHall

, Integer sId) throws IOException {

PrintWriter pw=new PrintWriter(socket.getOutputStream(),true);

pw.println("HTTP/1.1 500 OK\n\n");

pw.flush();

pw.close();

socket.close();

sleeperHall.removeThreadById(sId);

return;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.deta.boot.vpc.controller;

import java.io.File;

import java.io.FileInputStream;

import java.io.IOException;

import java.net.ServerSocket;

//import java.net.ServerSocket;

import java.util.Properties;

import org.deta.boot.vpc.process.TimeProcess;

import org.deta.boot.vpc.sleeper.Sleeper;

import org.deta.boot.vpc.sleeper.SleeperHall;

import org.deta.vpcs.hall.DatabaseLogHall;

import org.lyg.cache.DetaDBBufferCacheManager;

import org.lyg.common.utils.DetaUtil;

import org.lyg.stable.StableData;

public class ServerInitController {

private static ServerSocket server;

private static Properties properties;

private static int port;

static {

properties = new Properties();

try {

properties.load(new FileInputStream

(new File("src/main/resources/property.proterties")));

System.out.println("----德塔VPCS数据库服务器资源载入:成功！");

}catch (IOException e) {

e.printStackTrace();

}

}

public static void initService() throws IOException {

try {

port = Integer.parseInt(properties.getProperty(StableData.TCP\_PORT));

server = new ServerSocket(port);

System.out.println("----德塔VPCS数据库服务器端口启动:" + port);

DetaUtil.initDB();

System.out.println("----德塔VPCS数据库服务器DMA确认:成功！");

RequestFilterController.initBlockList();

System.out.println("----德塔VPCS数据库服务器IP过滤服务启动:成功！");

DetaDBBufferCacheManager.reflection();

System.out.println("----德塔VPCS数据库服务器启动整库过程映射服务:成功！");

DatabaseLogHall.createBinLogHall();

System.out.println("----德塔VPCS数据库服务器启动整库过程映射服务:成功！");

// BootBackup.bootBackupByUsingGzip(CacheManager

//.getCacheInfo("LogPath").getValue().toString()+"/zipCover");

// UnZip.unZipWithPath("C:/DetaLog/zipCover/zip\_1549583065203.zip"

//, "C:/DetaLog/zipCover/cover");

} catch (Exception e) {

e.printStackTrace();

}

}

private static void haoHiYooFaker(SleeperHall sleeperHall) {

sleeperHall.callSkivvy();

}

public static void initServer() throws IOException {

System.out.println("----DETA VPCS--1.8");

System.out.println("----Author: 罗瑶光");

System.out.println("----浏阳德塔软件开发有限公司开源项目");

TimeProcess timeProcess=new TimeProcess();

timeProcess.begin();

SleeperHall sleeperHall = new SleeperHall();

initService();

timeProcess.end();

System.out.println("----德塔VPCS数据库服务器启动一切正常-总耗时:"

+ timeProcess.duration()+ "毫秒");

while(true){

if(sleeperHall.getThreadsCount() < StableData.SLEEPERS\_RANGE){

Sleeper sleeper = new Sleeper();

try {

sleeper.hugPillow(sleeperHall, server.accept()

, sleeper.hashCode());

sleeper.start();

} catch (IOException e) {

e.printStackTrace();

}

}else {

haoHiYooFaker(sleeperHall);

}

}

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.deta.boot.vpc.sleeper;

import java.io.IOException;

import java.net.Socket;

import org.deta.boot.vpc.vision.VPCSRequest;

import org.deta.boot.vpc.vision.VPCSResponse;

public class Sleeper extends Thread implements Runnable{

private VPCSRequest vPCSRequest;

private VPCSResponse vPCSResponse;

public Sleeper(){

vPCSRequest = new VPCSRequest();

vPCSResponse = new VPCSResponse();

vPCSResponse.setHashCode(this.hashCode());

}

public void run(){

try{

org.deta.boot.vpc.controller.RequestRecordController

.requestIpRecoder(vPCSRequest, vPCSResponse);

if(vPCSResponse.getSocket().isClosed()) {

return;

}

org.deta.boot.vpc.controller.RequestRecordController

.requestLinkRecoder(vPCSRequest, vPCSResponse);

if(vPCSResponse.getSocket().isClosed()) {

return;

}

org.deta.boot.vpc.controller.RequestFilterController

.requestIpFilter(vPCSRequest, vPCSResponse);

if(vPCSResponse.getSocket().isClosed()) {

return;

}

org.deta.boot.vpc.controller.RequestFilterController

.requestLinkFilter(vPCSRequest, vPCSResponse);

if(vPCSResponse.getSocket().isClosed()) {

return;

}

org.deta.boot.vpc.controller.RequestFixController

.requestIpFix(vPCSRequest, vPCSResponse);

if(vPCSResponse.getSocket().isClosed()) {

return;

}

org.deta.boot.vpc.controller.RequestFixController

.requestLinkFix(vPCSRequest, vPCSResponse);

if(vPCSResponse.getSocket().isClosed()) {

return;

}

org.deta.boot.vpc.vision.ForwardVision.getForwardType(vPCSRequest, vPCSResponse);

if(vPCSResponse.getSocket().isClosed()) {

return;

}

org.deta.boot.vpc.vision.ForwardVision.forwardToRestMap(vPCSRequest, vPCSResponse);

if(vPCSResponse.getSocket().isClosed()) {

return;

}

org.deta.boot.vpc.vision.RestMapVision.getResponse(vPCSRequest, vPCSResponse);

if(vPCSResponse.getSocket().isClosed()) {

return;

}

org.deta.boot.vpc.vision.RestMapVision.returnResponse(vPCSRequest, vPCSResponse);

if(vPCSResponse.getSocket().isClosed()) {

return;

}

}catch(Exception e){

try {

vPCSResponse.returnErrorCode(500);

e.printStackTrace();

} catch (IOException e1) {

System.gc();

e1.printStackTrace();

}

}

}

public void hugPillow(SleeperHall sleeperHall, Socket accept, int hashCode) {

sleeperHall.addExecSleeper(hashCode, this);

vPCSResponse.setSocket(accept);

vPCSResponse.setSleeperHall(sleeperHall);

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.deta.boot.vpc.sleeper;

import java.util.Map;

import java.util.concurrent.ConcurrentHashMap;

public class SleeperHall{

private Map<Integer, Sleeper> sleepersMap;

public SleeperHall(){

sleepersMap = new ConcurrentHashMap<>();

}

public int getThreadsCount() {

return sleepersMap.size();

}

public void addExecSleeper(Integer sid, Sleeper sleeper) {

sleepersMap.put(sid, sleeper);

}

public void removeThreadById(Integer sid) {

if(sleepersMap.containsKey(sid)){

sleepersMap.remove(sid);

}

}

public void callSkivvy() {

sleepersMap.clear();

System.gc();

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.deta.boot.vpc.vision;

import java.io.IOException;

import java.net.Socket;

import org.deta.boot.rest.VPC;

import org.lyg.stable.StableData;

public class ForwardVision {

public static void main(String[] args){

}

public static void getForwardType(Socket socket) {

}

public static void forwardToRestMap(Socket socket) {

}

public static void getForwardType(VPCSRequest vPCSRequest

, VPCSResponse vPCSResponse) throws IOException {

if(vPCSRequest.getRequestIsRest()){

String filePath = VPC.getFilePath(vPCSRequest.getRequestLink());

if(filePath.contains(StableData.FILE\_TTF)

||filePath.contains(StableData.FILE\_EOT)

||filePath.contains(StableData.FILE\_SVG)

||filePath.contains(StableData.FILE\_WOFF)

||filePath.contains(StableData.FILE\_WOFF2)

||filePath.contains(StableData.FILE\_OTF)){

String code = VPC.getCode(filePath);

vPCSRequest.setRequestFilePath(filePath);

vPCSRequest.setRequestFileCode(code);

vPCSRequest.setRequestForwardType(StableData.STREAM\_BUFFER);

vPCSResponse.setResponseContentType(StableData.HEADER\_CONTENT\_TYPE\_FONT\_WOFF);

return;

}

vPCSRequest.setRequestForwardType(StableData.STREAM\_REST);

}else{

String filePath = VPC.getFilePath(vPCSRequest.getRequestLink());

String code = VPC.getCode(filePath);

vPCSRequest.setRequestFilePath(filePath);

vPCSRequest.setRequestFileCode(code);

if(filePath.contains(StableData.FILE\_PNG)){

vPCSRequest.setRequestForwardType(StableData.STREAM\_BYTES);

vPCSResponse.setResponseContentType(StableData.HEADER\_CONTENT\_TYPE\_PNG);

}

if(filePath.contains(StableData.FILE\_JPEG)){

vPCSRequest.setRequestForwardType(StableData.STREAM\_BYTES);

vPCSResponse.setResponseContentType(StableData.HEADER\_CONTENT\_TYPE\_JPEG);

}

if(filePath.contains(StableData.FILE\_JPG)){

vPCSRequest.setRequestForwardType(StableData.STREAM\_BYTES);

vPCSResponse.setResponseContentType(StableData.HEADER\_CONTENT\_TYPE\_JPG);

}

if(filePath.contains(StableData.FILE\_GIF)){

vPCSRequest.setRequestForwardType(StableData.STREAM\_BYTES);

vPCSResponse.setResponseContentType(StableData.HEADER\_CONTENT\_TYPE\_GIF);

}

if(filePath.contains(StableData.FILE\_JS) && code.equalsIgnoreCase(StableData.CHARSET\_UTF\_8)){

vPCSRequest.setRequestForwardType(StableData.STREAM\_BYTES\_BUFFER);

vPCSResponse.setResponseContentType(StableData.HEADER\_CONTENT\_TYPE\_JS);

}

if(filePath.contains(StableData.FILE\_CSS)){

vPCSRequest.setRequestForwardType(StableData.STREAM\_BUFFER);

vPCSResponse.setResponseContentType(StableData.HEADER\_CONTENT\_TYPE\_CSS);

}

if(filePath.contains(StableData.FILE\_HTML)){

vPCSRequest.setRequestForwardType(StableData.STREAM\_BUFFER);

vPCSResponse.setResponseContentType(StableData.HEADER\_CONTENT\_TYPE\_HTML);

}

if(filePath.contains(StableData.FILE\_WAV)){

vPCSRequest.setRequestForwardType(StableData.STREAM\_BUFFER);

vPCSResponse.setResponseContentType(StableData.HEADER\_CONTENT\_TYPE\_WAV);

}

}

}

public static void forwardToRestMap(VPCSRequest vPCSRequest

, VPCSResponse vPCSResponse) throws Exception {

if(null == vPCSRequest || null == vPCSRequest.getRequestForwardType()){

vPCSResponse.return404();

return;

}

if(vPCSRequest.getRequestForwardType().equalsIgnoreCase(StableData.STREAM\_REST)){

RestMapVision.processRest(vPCSRequest, vPCSResponse);

}

if(vPCSRequest.getRequestForwardType().equalsIgnoreCase(StableData.STREAM\_BYTES)){

RestMapVision.processBytes(vPCSRequest, vPCSResponse);

}

if(vPCSRequest.getRequestForwardType().equalsIgnoreCase(StableData.STREAM\_BUFFER)){

RestMapVision.processBuffer(vPCSRequest, vPCSResponse);

}

if(vPCSRequest.getRequestForwardType().equalsIgnoreCase(StableData.STREAM\_BYTES\_BUFFER)){

RestMapVision.processBufferBytes(vPCSRequest, vPCSResponse);

}

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.deta.boot.vpc.vision;

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.ByteArrayOutputStream;

import java.io.DataOutputStream;

import java.io.File;

import java.io.FileInputStream;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.OutputStreamWriter;

import java.io.PrintWriter;

import java.io.UnsupportedEncodingException;

import java.net.Socket;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

import org.deta.boot.rest.VPC;

import org.lyg.cache.DetaCacheManager;

import org.lyg.stable.StableData;

import zipProcessor.GzipUtil;

public class RestMapVision {

public static void main(String[] args){

}

public static void getResponse(Socket socket) {

}

public static void returnResponse(Socket socket) {

}

public static void getResponse(VPCSRequest vPCSRequest, VPCSResponse vPCSResponse) {

}

public static void returnResponse(VPCSRequest vPCSRequest, VPCSResponse vPCSResponse) {

vPCSResponse.getSleeperHall().removeThreadById(vPCSResponse.getHashCode());

}

public static void processRest(VPCSRequest vPCSRequest

, VPCSResponse vPCSResponse) throws Exception {

String output = VPC.forward(vPCSRequest.getRequestLink()

, vPCSRequest.getRequestValue());

PrintWriter pw = new PrintWriter(new BufferedWriter(new OutputStreamWriter(vPCSResponse.getSocket()

.getOutputStream(),StableData.CHARSET\_UTF\_8)),true);

pw.println("HTTP/1.1 200 OK\n\n");

output=output.charAt(StableData.INT\_ZERO)=='"'?output.substring(StableData.INT\_ONE,output.length())

:output;

output=output.charAt(output.length()-StableData.INT\_ONE)=='"'?output.substring(StableData.INT\_ZERO

, output.length()-StableData.INT\_ONE):output;

pw.println(output.replace("\\\"","\""));

System.out.println("db:"+4);

pw.flush();

pw.close();

vPCSResponse.getSleeperHall().removeThreadById(vPCSResponse.getSocket().hashCode());

}

public static void processView(VPCSRequest vPCSRequest, VPCSResponse vPCSResponse) {

}

public static void processBytes(VPCSRequest vPCSRequest

, VPCSResponse vPCSResponse) throws IOException {

List<byte[]> list;

DataOutputStream dataOutputStream = new DataOutputStream(vPCSResponse.getSocket().getOutputStream());

if(null != DetaCacheManager.getCacheOfBytesList(vPCSRequest.getRequestFilePath())){

list = DetaCacheManager.getCacheOfBytesList(vPCSRequest.getRequestFilePath());

}else{

FileInputStream fileInputStream = new FileInputStream(new File(vPCSRequest.getRequestFilePath()));

ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();

byte[] byteArray = new byte[StableData.BUFFER\_RANGE\_MAX];

int lengthOfFile = StableData.INT\_ZERO;

list = new ArrayList<>();

//这段while函数思想来自 这篇文章

：https://blog.csdn.net/top\_code/article/details/41042413

//非常轻松处理len的长度溢出。谢谢。

while((lengthOfFile = fileInputStream.read(byteArray, StableData.INT\_ZERO

, StableData.BUFFER\_RANGE\_MAX)) != StableData.INT\_MINES\_ONE){

byteArrayOutputStream.write(byteArray, StableData.INT\_ZERO, lengthOfFile);

}

fileInputStream.close();

byte[] sniper = GzipUtil.compress(byteArrayOutputStream.toByteArray());

list.add(0, vPCSResponse.getResponseContentType().getBytes(StableData.CHARSET\_UTF8));

list.add(0, (StableData.HEADER\_CONTENT\_LENGTH + sniper.length + StableData.STRING\_SPACE\_ENTER)

.getBytes(StableData.CHARSET\_UTF8));

list.add(0, StableData.HEADER\_CACHE\_CONTROL.getBytes(StableData.CHARSET\_UTF8));

list.add(0, StableData.HEADER\_CONTENT\_ENCODING\_GZIP.getBytes(StableData.CHARSET\_UTF8));

list.add(0, StableData.HEADER\_ACCEPT\_RANGES\_BYTES.getBytes(StableData.CHARSET\_UTF8));

list.add(0, StableData.HEADER\_HOST.getBytes(StableData.CHARSET\_UTF8));

list.add(0, StableData.HEADER\_HTTP\_200\_OK.getBytes(StableData.CHARSET\_UTF8));

if(null != sniper && sniper.length>StableData.INT\_ZERO) {

list.add(sniper);

}

DetaCacheManager.putCacheOfBytesList(vPCSRequest.getRequestFilePath(), list);

}

Iterator<byte[]> iterator = list.iterator();

while(iterator.hasNext()){

dataOutputStream.write(iterator.next());

}

dataOutputStream.flush();

dataOutputStream.close();

}

public static void processBuffer(VPCSRequest vPCSRequest

, VPCSResponse vPCSResponse) throws IOException {

String builderToString;

if(null != DetaCacheManager.getCacheOfString(vPCSRequest.getRequestFilePath())){

builderToString = DetaCacheManager.getCacheOfString(vPCSRequest.getRequestFilePath());

}else{

StringBuilder stringBuilder = new StringBuilder();

stringBuilder.append(StableData.HEADER\_HTTP\_200\_OK);

stringBuilder.append(StableData.HEADER\_HOST);

stringBuilder.append(StableData.HEADER\_CACHE\_CONTROL);

stringBuilder.append(vPCSResponse.getResponseContentType());

FileInputStream fileInputStream = new FileInputStream(new File(vPCSRequest.getRequestFilePath()));

InputStreamReader inputStreamReader = new InputStreamReader(fileInputStream

, vPCSRequest.getRequestFileCode());

BufferedReader bufferedReader = new BufferedReader(inputStreamReader);

String line = null;

while ((line = bufferedReader.readLine()) != null) {

stringBuilder.append(line);

}

bufferedReader.close();

builderToString = stringBuilder.toString();

DetaCacheManager.putCacheOfString(vPCSRequest.getRequestFilePath(), builderToString);

}

BufferedWriter bufferedWriter = new BufferedWriter(new OutputStreamWriter(vPCSResponse.getSocket()

.getOutputStream(), vPCSRequest.getRequestFileCode()));

bufferedWriter.write(builderToString);

bufferedWriter.flush();

bufferedWriter.close();

}

public static void processBufferBytes(VPCSRequest vPCSRequest

, VPCSResponse vPCSResponse) throws UnsupportedEncodingException, IOException {

StringBuilder stringBuilder = new StringBuilder();

stringBuilder.append(StableData.HEADER\_HTTP\_200\_OK);

stringBuilder.append(StableData.HEADER\_HOST);

stringBuilder.append(StableData.HEADER\_CACHE\_CONTROL);

stringBuilder.append(StableData.HEADER\_CONTENT\_ENCODING\_GZIP);

stringBuilder.append(vPCSResponse.getResponseContentType());

String builderToString = stringBuilder.toString();

String contentBuilderToString;

if(null != DetaCacheManager.getCacheOfString(vPCSRequest.getRequestFilePath())){

contentBuilderToString = DetaCacheManager

.getCacheOfString(vPCSRequest.getRequestFilePath());

}else{

StringBuilder contentBuilder = new StringBuilder();

FileInputStream fileInputStream = new FileInputStream(new File(vPCSRequest

.getRequestFilePath()));

int lengthOfFile = StableData.INT\_ZERO;

byte[] byteArray = new byte[StableData.BUFFER\_RANGE\_MAX];

while ((lengthOfFile = fileInputStream.read(byteArray)) != StableData.INT\_MINES\_ONE){

contentBuilder.append(new String(byteArray, StableData.INT\_ZERO, lengthOfFile

, StableData.CHARSET\_UTF\_8));

}

fileInputStream.close();

contentBuilderToString = contentBuilder.toString();

DetaCacheManager.putCacheOfString(vPCSRequest.getRequestFilePath(), contentBuilderToString);

}

DataOutputStream dataOutputStream = new DataOutputStream(vPCSResponse.getSocket().getOutputStream());

dataOutputStream.write(builderToString.getBytes(StableData.CHARSET\_UTF8));

dataOutputStream.write(GzipUtil.compress(contentBuilderToString.getBytes(StableData.CHARSET\_UTF8)));

dataOutputStream.flush();

dataOutputStream.close();

}

}}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.deta.boot.vpc.vision;

import java.io.IOException;

import java.io.PrintWriter;

import java.net.Socket;

import org.deta.boot.vpc.sleeper.SleeperHall;

public class VPCSResponse{

public Socket getSocket() {

return socket;

}

public void setSocket(Socket socket) {

this.socket = socket;

}

public SleeperHall getSleeperHall() {

return sleeperHall;

}

public void setSleeperHall(SleeperHall sleeperHall) {

this.sleeperHall = sleeperHall;

}

public Integer getHashCode() {

return hashCode;

}

public void setHashCode(Integer hashCode) {

this.hashCode = hashCode;

}

public int getErrorCode() {

return errorCode;

}

public void setErrorCode(int errorCode) {

this.errorCode = errorCode;

}

public String getResponseContentType() {

return ResponseContentType;

}

public void setResponseContentType(String responseContentType) {

ResponseContentType = responseContentType;

}

private Socket socket;

private SleeperHall sleeperHall;

private Integer hashCode;

private int errorCode;

private String ResponseContentType;

public void return404() throws IOException {

if(socket.isClosed()) {

this.sleeperHall.removeThreadById(this.hashCode);

return;

}

PrintWriter pw = new PrintWriter(this.socket.getOutputStream(), true);

pw.println("HTTP/1.1 404 OK\n\n");

pw.flush();

pw.close();

socket.close();

this.sleeperHall.removeThreadById(this.hashCode);

}

public void returnErrorCode(Integer errorCode) throws IOException {

if(socket.isClosed()) {

this.sleeperHall.removeThreadById(this.hashCode);

return;

}

PrintWriter pw = new PrintWriter(this.socket.getOutputStream(), true);

pw.println("HTTP/1.1 " + errorCode + " OK\n\n");

pw.flush();

pw.close();

socket.close();

this.sleeperHall.removeThreadById(this.hashCode);

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.deta.vpcs.hall;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import cacheProcessor.CacheManager;

import zipProcessor.GzipUtil;

import zipProcessor.UnZip;

import org.lyg.db.plsql.imp.ExecPLSQLImp;

import org.lyg.stable.StableData;

@SuppressWarnings("unused")

public class DatabaseLogHall {

static String logCategoryPath;

static String logCurrentFilePath;

static String logCurrentFile;

public static void createBinLogHall() throws Exception {

//db write operation

initLogCategoryPath();

initCurrentFilePath();

//write error rollback

//binlog

}

private static void initCurrentFilePath() {

long yearMonthDay = System.currentTimeMillis();

long day = yearMonthDay/(1000 \* 60 \* 60 \* 24);

logCurrentFilePath = logCategoryPath + "/log/logger" + day + ".det";

}

private static void initLogCategoryPath() throws Exception {

if(null != CacheManager.getCacheInfo("DBPath")) {

logCategoryPath = CacheManager.getCacheInfo("DBPath").getValue().toString();

}else {

throw new Exception();

}

}

public static void writeLogFile(long when, String who, String plsql) {

checkCurrentFileRange();

//zip text string and write; im thinking about a new method to make string write small.

String logString ="#@:" + when + "@:" + who + "@:" + plsql;

FileWriter fw = null;

try {

fw = new FileWriter(logCurrentFilePath, true);

fw.write("\n\r\n");

fw.write(new String(GzipUtil.compress(logString.getBytes(StableData.CHARSET\_UTF8))

, StableData.CHARSET\_UTF8));

// fw.write(logString);

fw.close();

} catch (IOException e) {

e.printStackTrace();

}

}

private static void checkCurrentFileRange() {

long yearMonthDay = System.currentTimeMillis();

long day = yearMonthDay/(1000 \* 60 \* 60 \* 24);

String willMakeFile = logCategoryPath + "/log/logger" + day + ".det";

String willMakeFilePath = logCategoryPath + "/log";

File file = new File(willMakeFile);

File fileWillMakeFilePath = new File(willMakeFilePath);

if(!fileWillMakeFilePath.exists()) {

fileWillMakeFilePath.mkdirs();

//dont make new for this class, will get memory leakage

}

if(!file.exists()) {

//dont make new for this class, will get memory leakage

logCurrentFilePath = willMakeFile;

}

//if currentfiletime - currenttime > 1 day

//if currentfilesize > 100mb will make discussion later,now just make one file per day.

//makenew file;

}

private static void coverageByTime(long time) throws Exception {

// 1删除已损坏的数据库 已完成

File needClear = new File("C:/DetaDB");

needClear.delete();

// 2解压备份数据库 已完成

UnZip.unZipWithPath("C:/DetaLog/zipCover/zip\_1549583065203.zip", "C:/DetaLog/zipCover/cover");

// 3循环执行备份plsql命令，直到等于大于时间戳完成返回。

BufferedReader reader = new BufferedReader(new FileReader("C:/DetaLog/log/logger.det"));

String tempString;

while ((tempString = reader.readLine()) != null) {

//解gzip压缩并执行数据库恢复

tempString = new String(GzipUtil.uncompress(tempString.getBytes(StableData.CHARSET\_UTF8))

, StableData.CHARSET\_UTF8);

long currentTime =Long.valueOf(tempString.split("@:")[1]);

if(currentTime < time) {//逐条恢复到点。

ExecPLSQLImp.ExecPLSQL(tempString.split("@:")[3], true);

package org.lyg.cache;

import java.util.Iterator;

import java.util.List;

import java.util.concurrent.ConcurrentHashMap;

public class DetaCacheManager {

private static ConcurrentHashMap<String, DetaCache> cacheMap = new ConcurrentHashMap<>();

private static ConcurrentHashMap<String, List<byte[]>> bytesMap = new ConcurrentHashMap<>();

private static ConcurrentHashMap<String, String> stringMap = new ConcurrentHashMap<>();

private DetaCacheManager() {

super();

}

public static String putCache(String key, String value, long timeOut){

DetaCache c = new DetaCache();

c.setValue(value);

c.setTimeOut(timeOut);

cacheMap.put(key, c);

return "success";

}

public static String getCache(String key){

DetaCache c = cacheMap.get(key);

if(null==c){

return "unsuccess nofind cache";

}

long now = System.currentTimeMillis();

if(c.getTimeOut() < now){

cacheMap.remove(key);

return "unsuccess timeout";

}

return c.getValue();

}

@SuppressWarnings("rawtypes")

public static Iterator getCacheIterator(){

return cacheMap.entrySet().iterator();

}

public static void putCacheOfBytesList(String filePath, List<byte[]> list) {

bytesMap.put(filePath, list);

}

public static List<byte[]> getCacheOfBytesList(String filePath) {

if(bytesMap.containsKey(filePath)){

return bytesMap.get(filePath);

}

return null;

}

public static String getCacheOfString(String filePath) {

if(stringMap.containsKey(filePath)){

return stringMap.get(filePath);

}

return null;

}

public static void putCacheOfString(String filePath, String stringBuilder) {

stringMap.put(filePath, stringBuilder);

}

} }

}

reader.close();

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.lyg.cache;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.IOException;

import java.util.concurrent.ConcurrentHashMap;

import org.lyg.db.reflection.Base;

import org.lyg.db.reflection.Cell;

import org.lyg.db.reflection.DB;

import org.lyg.db.reflection.Row;

import org.lyg.db.reflection.Spec;

import org.lyg.db.reflection.Table;

import cacheProcessor.CacheManager;

@SuppressWarnings("unused")

public class DetaDBBufferCacheManager {

public static DB db = new DB();

public static boolean dbCache = false;

private DetaDBBufferCacheManager() {

super();

}

public static void reflection() throws IOException {

ConcurrentHashMap<String, Base> bases = new ConcurrentHashMap<>();

db.setBases(bases);

//1获取db路径；

String dBPath = CacheManager.getCacheInfo("DBPath").getValue().toString();

File fileDBPath = new File(dBPath);

if (fileDBPath.isDirectory()) {

String[] baseNames = fileDBPath.list();

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

loopBases(db, dBPath, baseNames[i]);

}

}

dbCache = true;

}

private static void loopBases(DB db, String dBPath, String baseName) throws IOException {

Base base = new Base();

ConcurrentHashMap<String, Table> tables = new ConcurrentHashMap<>();

base.setTables(tables);

String dBasePath = dBPath + "/" + baseName;

//get base

File fileDBasePath = new File(dBasePath);

if (fileDBasePath.isDirectory()) {

ConcurrentHashMap<String, Object> tableMap = new ConcurrentHashMap<>();

//get tables

String[] tableNames = fileDBasePath.list();

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

loopTables(base, dBasePath, tableNames[i]);

}

}

db.putBase(baseName, base);

}

private static void loopTables(Base base, String dBasePath, String tableName) throws IOException {

Table table = new Table();

String tablePath = dBasePath + "/" + tableName;

File fileTablePath = new File(tablePath);

if (fileTablePath.isDirectory()) {

String specPath = tablePath + "/spec";

String rowsPath = tablePath + "/rows";

loopSpec(table,specPath);

loopRows(table,rowsPath);

}

base.putTable(tableName, table);

}

private static void loopSpec(Table table, String specPath) throws IOException {

Spec spec = new Spec();

ConcurrentHashMap<String, String> culumnTypes = new ConcurrentHashMap<>();

spec.setCulumnTypes(culumnTypes);

File fileSpecPath = new File(specPath);

if (fileSpecPath.isDirectory()) {

String[] specs = fileSpecPath.list();

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

String specCulumnPath = specPath + "/" + specs[i];

String specCulumnValuePath = specCulumnPath + "/value.lyg";

//if get

BufferedReader reader = new BufferedReader(new FileReader(specCulumnPath + "/" + "value.lyg"));

String temp = "";

String tempString = "";

while ((tempString = reader.readLine()) != null) {

temp += tempString;

}

reader.close();

spec.setCulumnType(specs[i], temp);

}

}

table.setSpec(spec);

}

private static void loopRows(Table table, String rowsPath) throws IOException {

ConcurrentHashMap<String, Row> rows = new ConcurrentHashMap<>();

table.setRows(rows);

File fileRowsPath = new File(rowsPath);

if (fileRowsPath.isDirectory()) {

String[] rowIndex = fileRowsPath.list();

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

loopRow(table, fileRowsPath, rowIndex[i]);

}

}

}

private static void loopRow(Table table, File fileRowsPath, String rowIndex) throws IOException {

Row row = new Row();

ConcurrentHashMap<String, Cell> cells = new ConcurrentHashMap<>();

row.setCells(cells);

String rowIndexPath = fileRowsPath + "/" + rowIndex;

File fileRowPath = new File(rowIndexPath);

if (fileRowPath.isDirectory()) {

String[] culumns = fileRowPath.list();

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

String rowCulumnPath = rowIndexPath + "/" + culumns[i];

String rowCulumnValuePath = rowCulumnPath + "/value.lyg";

//if get

if(!culumns[i].contains("is\_delete")) {

BufferedReader reader = null;

try {

reader = new BufferedReader(new FileReader(rowCulumnValuePath));

}catch(Exception e){

e.printStackTrace();

}

String temp = "";

String tempString = "";

while ((tempString = reader.readLine()) != null) {

temp += tempString;

}

reader.close();

Cell cell = new Cell();

cell.setCellValue(temp);

row.putCell(culumns[i], cell);

}else {

Cell cell = new Cell();

cell.setCellValue("");

row.putCell(culumns[i], cell);

}

}

}

table.putRow(rowIndex, row);;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.lyg.common.utils;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.net.HttpURLConnection;

import java.net.URL;

import java.util.Map;

import java.util.concurrent.ConcurrentHashMap;

public class DetaDBUtil {

public static Map<String, Boolean> culumnType;

public static String backEndRequest(String request) throws IOException {

URL url = new URL("http://localhost:8080/" + request);

HttpURLConnection conn = (HttpURLConnection) url.openConnection();

conn.setRequestMethod("POST");

conn.setRequestProperty("Accept", "application/json");

if (conn.getResponseCode() != 200) {

throw new RuntimeException("Failed : HTTP error code : " + conn.getResponseCode());

}

BufferedReader br = new BufferedReader(new

InputStreamReader((conn.getInputStream()),"UTF-8"));

String out = "";

String out1;

while ((out1 = br.readLine()) != null) {

out += out1;

}

conn.disconnect();

return out;

}

public static String cacheRequest(String request) throws IOException {

URL url = new URL("http://localhost:6379/" + request);

HttpURLConnection conn = (HttpURLConnection) url.openConnection();

conn.setRequestMethod("POST");

conn.setRequestProperty("Accept", "application/json;charset=utf-8");

conn.setRequestProperty("Content-Type", "application/json;charset=utf-8");

if (conn.getResponseCode() != 200) {

throw new RuntimeException("Failed : HTTP error code : " + conn.getResponseCode());

}

BufferedReader br = new BufferedReader(new

InputStreamReader((conn.getInputStream()),"UTF-8"));

String out = "";

String out1;

while ((out1 = br.readLine()) != null) {

out += out1;

}

conn.disconnect();

return out;

}

public static void initCulumnNameType() {

culumnType = new ConcurrentHashMap<>();

culumnType.put("int", true);

culumnType.put("long", true);

culumnType.put("double", true);

culumnType.put("string", true);

culumnType.put("objectJPG", true);

culumnType.put("objectPDF", true);

culumnType.put("objectPNG", true);

culumnType.put("objectMP4", true);

culumnType.put("objectAVI", true);

culumnType.put("objectGIF", true);

culumnType.put("objectGIF", true);

}

public static boolean withoutCulumnNameType(String culumnTypeString) {

if(!culumnType.containsKey(culumnTypeString)) {

return true;

}

return false;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.lyg.common.utils;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import cacheProcessor.Cache;

import cacheProcessor.CacheManager;

public class DetaUtil {

public static void initDB(){

File config = new File("C:/DBconfig.lyg");

if (config.exists()) {

try {

BufferedReader reader = new BufferedReader(new FileReader(config));

String tempString;

while ((tempString = reader.readLine()) != null) {

Cache c = new Cache();

c.setValue(tempString.split("->")[1]);

CacheManager.putCache("DBPath", c);

File fileDBPath = new File(tempString.split("->")[1]);

if (fileDBPath.isDirectory()) {

}else {

fileDBPath.mkdir();

}

}

reader.close();

} catch (FileNotFoundException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

}else {

FileWriter fw = null;

try {

fw = new FileWriter("C:/DBconfig.lyg", true);

fw.write("path->" + "C:/DetaDB");

fw.close();

File fileDBPath = new File("C:/DetaDB");

if (fileDBPath.isDirectory()) {

}else {

fileDBPath.mkdir();

}

Cache c = new Cache();

c.setValue("C:/DetaDB");

CacheManager.putCache("DBPath", c);

} catch (IOException e) {

e.printStackTrace();

}

}

Cache c = new Cache();

c.setValue("C:/DetaLog");

CacheManager.putCache("LogPath", c);

DetaDBUtil.initCulumnNameType();

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.lyg.db.create.imp;

import java.io.File;

import java.io.FileWriter;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import org.lyg.common.utils.DetaDBUtil;

import cacheProcessor.CacheManager;

@SuppressWarnings("unchecked")

public class CreateTablesImp {

public static void createTable(Map<String, Object> object, boolean mod) throws Exception {

if(!object.containsKey("baseName")||!object.containsKey("tableName")){

return;

}

//get base

String DBPath = CacheManager.getCacheInfo("DBPath").getValue().toString() + "/" +

object.get("baseName").toString();

File DBPathFile = new File(DBPath);

if(!DBPathFile.isDirectory()) {

return;

}

//make table dir

String tablePath = DBPath + "/" + object.get("tableName").toString();

File tablePathFile = new File(tablePath);

if(tablePathFile.exists()) {

return;

}

//make spec dir

tablePathFile.mkdirs();

String tableSpecPath = DBPath + "/spec";

String tableRowsPath = DBPath + "/rows";

File tableSpecPathFile = new File(tableSpecPath);

File tableRowsPathFile = new File(tableRowsPath);

tableSpecPathFile.mkdir();

tableRowsPathFile.mkdir();

//make data

List<String[]> culumnNames = (List<String[]>) object.get("culumnName");

Iterator<String[]> iterator = culumnNames.iterator();

while(iterator.hasNext()) {

String[] culumnDefinition = iterator.next();

//create name

String culumnNamePath=tableSpecPath + "/" + culumnDefinition[2];

File culumnNameFile = new File(culumnNamePath);

if(DetaDBUtil.withoutCulumnNameType(culumnDefinition[3])) {

throw new Exception();

}

culumnNameFile.mkdir();

//create file

File file = new File(culumnNamePath + "/value.lyg");

if(file.exists()) {

throw new Exception();

}

if(mod) {

FileWriter fw = null;

fw = new FileWriter(file, true);

fw.write(culumnDefinition[3]);

fw.close();

}

}

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.lyg.db.delete.imp;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.IOException;

import java.util.HashMap;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import org.lyg.cache.DetaDBBufferCacheManager;

import org.lyg.db.reflection.Table;

import org.lyg.db.select.imp.SelectRowsImp;

import cacheProcessor.CacheManager;

public class DeleteRowsImp {

public static Map<String, Object> deleteRowByTablePathAndIndex(String tablePath, String pageIndex, boolean mod)

throws FileNotFoundException, IOException {

int rowInsertIndex = Integer.valueOf(pageIndex);

File fileDBTable = new File(tablePath);

if (fileDBTable.isDirectory()) {

String DBTableRowsPath = tablePath + "/rows";

File fileDBTableRowsPath = new File(DBTableRowsPath);

if (fileDBTableRowsPath.isDirectory()) {

String DBTableRowIndexPath = DBTableRowsPath + "/row" + rowInsertIndex ;

File readDBTableRowIndexFile = new File(DBTableRowIndexPath);

if (readDBTableRowIndexFile.exists()) {

readDBTableRowIndexFile.mkdir();

String needCreatCulumnPath0 = DBTableRowIndexPath + "/is\_delete\_0";

File needCreatCulumn0 = new File(needCreatCulumnPath0);

if(mod) {

needCreatCulumn0.delete();

}

String needCreatCulumnPath = DBTableRowIndexPath + "/is\_delete\_1";

File needCreatCulumn = new File(needCreatCulumnPath);

if(mod) {

needCreatCulumn.mkdir();

}

}

}

}

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

output.put("totalPages", rowInsertIndex);

String[] sets = tablePath.split("/");

String baseName = sets[sets.length-2];

String tableName = sets[sets.length-1];

String indexName = "row"+pageIndex;

Table table = DetaDBBufferCacheManager.db.getBase(baseName).getTable(tableName);

if(mod) {

table.removeRow(indexName);

}

return output;

}

@SuppressWarnings({"unchecked"})

public static void deleteRowByAttributesOfCondition(Map<String, Object> object, boolean mod) throws IOException {

if(!object.containsKey("baseName")||!object.containsKey("tableName")){

return;

}

//get base

String DBPath = CacheManager.getCacheInfo("DBPath").getValue().toString() + "/" + object.get("baseName").toString();

File DBPathFile = new File(DBPath);

if(!DBPathFile.isDirectory()) {

return;

}

//make table dir

String tablePath = DBPath + "/" + object.get("tableName").toString();

List<Map<String, Object>> obj = (List<Map<String, Object>>) SelectRowsImp.selectRowsByAttributesOfCondition(object);

Iterator<Map<String, Object>> iterator = obj.iterator();

while(iterator.hasNext()) {

Map<String, Object> row = iterator.next();

Map<String, Object> rowValue = (Map<String, Object>) row.get("rowValue");

Map<String, Object> indexCell = (Map<String, Object>) rowValue.get("Index");

String indexValue = indexCell.get("culumnValue").toString();

deleteRowByTablePathAndIndex(tablePath, indexValue, mod);

//delete buffer also

// DetaDBBufferCacheManager.db.getBase(object.get("baseName").toString()).getTable(object.get("tableName")

// .toString()).removeRow(indexValue);

}

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.lyg.db.insert.imp;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileWriter;

import java.io.IOException;

import java.util.HashMap;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import java.util.concurrent.ConcurrentHashMap;

import cacheProcessor.CacheManager;

import org.json.JSONObject;

import org.lyg.cache.DetaDBBufferCacheManager;

import org.lyg.db.reflection.Cell;

import org.lyg.db.reflection.Row;

import org.lyg.db.reflection.Table;

@SuppressWarnings("unchecked")

public class InsertRowsImp {

public static Map<String, Object> insertRowByTablePathAndIndex(String tablePath, String pageIndex, JSONObject culumnOfNewRow) throws FileNotFoundException, IOException {

String[] sets = tablePath.split("/");

int rowInsertIndex = Integer.valueOf(pageIndex);

File fileDBTable = new File(tablePath);

if (fileDBTable.isDirectory()) {

String DBTableRowsPath = tablePath + "/rows";

File fileDBTableRowsPath = new File(DBTableRowsPath);

if (fileDBTableRowsPath.isDirectory()) {

Row row = new Row();

ConcurrentHashMap<String, Cell> cells = new ConcurrentHashMap<>();

row.setCells(cells);

String DBTableRowIndexPath = DBTableRowsPath + "/row" + rowInsertIndex ;

File readDBTableRowIndexFile = new File(DBTableRowIndexPath);

if (!readDBTableRowIndexFile.exists()) {

readDBTableRowIndexFile.mkdir();

Iterator<String> it = culumnOfNewRow.keys();

while(it.hasNext()) {

String culumnName = it.next();

String culumnValue = culumnOfNewRow.getString(culumnName);

String needCreatCulumnPath = DBTableRowIndexPath + "/" + culumnName;

File needCreatCulumn = new File(needCreatCulumnPath);

needCreatCulumn.mkdir();

FileWriter fw = null;

try {

fw = new FileWriter(needCreatCulumnPath + "/value.lyg", true);

fw.write(null==culumnValue?"":culumnValue);

fw.close();

//buffer reflection update

Cell cell = new Cell();

cell.setCellValue(null == culumnValue ? "" : culumnValue);

row.putCell(culumnName, cell);

} catch (IOException e) {

e.printStackTrace();

}

}

String needCreatCulumnPath = DBTableRowIndexPath + "/is\_delete\_0";

File needCreatCulumn = new File(needCreatCulumnPath);

needCreatCulumn.mkdir();

}

DetaDBBufferCacheManager.db.getBase(sets[sets.length-2]).getTable(sets[sets.length-1]).putRow("row" + rowInsertIndex, row);

}

}

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

output.put("totalPages", rowInsertIndex);

return output;

}

public static Map<String, Object> insertRowByBaseName(String baseName, String tableName, JSONObject jsobj, boolean mod) throws Exception {

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

String tablePath = CacheManager.getCacheInfo("DBPath").getValue().toString();

tablePath += "/" + baseName + "/" + tableName;

File fileDBTable = new File(tablePath);

if (fileDBTable.isDirectory()) {

String DBTableRowsPath = tablePath + "/rows";

Row row = new Row();

ConcurrentHashMap<String, Cell> cells = new ConcurrentHashMap<>();

row.setCells(cells);

File fileDBTableRowsPath = new File(DBTableRowsPath);

if (fileDBTableRowsPath.isDirectory()) {

int rowInsertIndex = fileDBTableRowsPath.list().length;

output.put("totalPages", rowInsertIndex);

String DBTableRowIndexPath = DBTableRowsPath + "/row" + rowInsertIndex ;

File readDBTableRowIndexFile = new File(DBTableRowIndexPath);

if (!readDBTableRowIndexFile.exists()) {

readDBTableRowIndexFile.mkdir();

Iterator<String> it = jsobj.keys();

while(it.hasNext()) {

String culumnName = it.next();

String culumnValue = jsobj.get(culumnName).toString();

if(culumnValue.equalsIgnoreCase("random")){

culumnValue = "" + rowInsertIndex;

}

String needCreatCulumnPath = DBTableRowIndexPath + "/" + culumnName;

File needCreatCulumn = new File(needCreatCulumnPath);

if(!needCreatCulumn.exists()) {

if(mod) {

needCreatCulumn.mkdir();

}

}

File needCreatCulumnPathFile= new File(needCreatCulumnPath + "/value.lyg");

if(needCreatCulumnPathFile.exists() && !needCreatCulumnPathFile.canWrite()) {

throw new Exception();

}

if(mod) {

FileWriter fw = null;

try {

fw = new FileWriter(needCreatCulumnPath + "/value.lyg", true);

fw.write(null == culumnValue? "" : culumnValue);

fw.close();

} catch (IOException e) {

e.printStackTrace();

}

}

//add buffer

Cell cell = new Cell();

cell.setCellValue(null == culumnValue? "" : culumnValue);

if(mod) {

row.putCell(culumnName, cell);

}

}

String needCreatCulumnPath = DBTableRowIndexPath + "/is\_delete\_0";

File needCreatCulumn = new File(needCreatCulumnPath);

if(!needCreatCulumn.exists()) {

if(mod){

needCreatCulumn.mkdir();

}

}

}

Table table = DetaDBBufferCacheManager.db.getBase(baseName).getTable(tableName);

if(mod) {

table.putRow("row" + rowInsertIndex, row);

}

}

}

return output;

}

public static void insertRowByAttributes(Map<String, Object> object, boolean mod) throws Exception {

JSONObject jsobj = new JSONObject();

//for late will make an exception record queue system, to control all of the db write;

List<String[]> culumnValues = (List<String[]>)object.get("culumnValue");

Iterator<String[]> iterator = culumnValues.iterator();

//list to json

while(iterator.hasNext()) {

String[] strings = iterator.next();

jsobj.put(strings[1], strings[2]);

}

insertRowByBaseName(object.get("baseName").toString(), object.get("tableName").toString(), jsobj, mod);

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.lyg.db.plsql.imp;

import java.util.Map;

import java.util.concurrent.ConcurrentHashMap;

public class ExecPLSQLImp {

public static Map<String, Object> ExecPLSQL(String plsql, boolean mod) throws Exception{

//working for here

Map<String, Object> output = new ConcurrentHashMap<>();

//1make container

output.put("start", "0");

output.put("countJoins", "0");

//2make line

String[] commands = plsql.replace(" ", "").replace("\n", "").split(";");

for(String command:commands) {

String[] acknowledge = command.split(":");

if(acknowledge[0].equals("setRoot")) {

PLSQLCommandImp.proceseSetRoot(acknowledge, output);

}

if(acknowledge[0].equals("baseName")) {

PLSQLCommandImp.processBaseName(acknowledge, output);

}

if(acknowledge[0].equals("tableName")) {

PLSQLCommandImp.processTableName(acknowledge, output);

}

if(acknowledge[0].equals("culumnName")) {

PLSQLCommandImp.processListNeedStart(acknowledge, output);

}

if(acknowledge[0].equals("changeCulumnName")) {

PLSQLCommandImp.processListNeedStart(acknowledge, output);

}

if(acknowledge[0].equals("culumnValue")) {

PLSQLCommandImp.processListNeedStart(acknowledge, output);

}

if(acknowledge[0].equals("join")) {

PLSQLCommandImp.processJoin(acknowledge, output);

}

if(acknowledge[0].equals("condition")) {

PLSQLCommandImp.processListNeedStart(acknowledge, output);

}

if(acknowledge[0].equals("relation")) {

PLSQLCommandImp.processListNeedStart(acknowledge, output);

}

if(acknowledge[0].equals("aggregation")) {

PLSQLCommandImp.processListNeedStart(acknowledge, output);

}

if(acknowledge[0].equals("getCulumns")) {

PLSQLCommandImp.processListNeedStart(acknowledge, output);

}

output.put("newCommand", acknowledge[0]);

PLSQLCommandImp.processExec(acknowledge, output, mod);

output.put("lastCommand", output.get("newCommand"));

}

PLSQLCommandImp.processCheck(output.get("newCommand").toString(), output, mod);

return output;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.lyg.db.plsql.imp;

import java.io.File;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import java.util.concurrent.CopyOnWriteArrayList;

import cacheProcessor.Cache;

import cacheProcessor.CacheManager;

import org.lyg.cache.DetaDBBufferCacheManager;

import org.lyg.db.create.imp.CreateTablesImp;

import org.lyg.db.delete.imp.DeleteRowsImp;

import org.lyg.db.insert.imp.InsertRowsImp;

import org.lyg.db.select.imp.SelectJoinRowsImp;

//import org.lyg.db.select.imp.SelectNestRowsImp;

import org.lyg.db.select.imp.SelectRowsImp;

import org.lyg.db.update.imp.UpdateJoinRowsImp;

import org.lyg.db.update.imp.UpdateRowsImp;

@SuppressWarnings("unchecked")

public class PLSQLCommandImp {

public static void proceseSetRoot(String[] acknowledge, Map<String, Object> output) throws Exception {

String dbPath = acknowledge[1];

for(int i=2; i<acknowledge.length; i++) {

dbPath += ":" + acknowledge[i];

}

if(null != CacheManager.getCacheInfo("DBPath")) {

File file = new File(dbPath);

if(!file.exists()) {

file.mkdirs();

Cache c = new Cache();

c.setValue(dbPath);

CacheManager.putCache("DBPath", c);

}else if(file.isFile()) {

throw new Exception();

}else if(file.isDirectory()) {

Cache c = new Cache();

c.setValue(dbPath);

CacheManager.putCache("DBPath", c);

}

}

}

public static void processBaseName(String[] acknowledge, Map<String, Object> object) {

object.put(acknowledge[0], acknowledge[1]);

}

public static void processTableName(String[] acknowledge, Map<String, Object> object) {

object.put(acknowledge[0], acknowledge[1]);

object.put("type", acknowledge[2]);

}

public static void processListNeedStart(String[] acknowledge, Map<String, Object> object) {

object.put("start", "1");

if(object.containsKey(acknowledge[0])) {

List<String[]> relationValues = (List<String[]>) object.get(acknowledge[0]);

relationValues.add(acknowledge);

object.put(acknowledge[0], relationValues);

return;

}

List<String[]> relationValues = new CopyOnWriteArrayList<>();

relationValues.add(acknowledge);

object.put(acknowledge[0], relationValues);

}

public static void processJoin(String[] acknowledge, Map<String, Object> object) {

if(object.get("countJoins").toString().equals("1")) {

object.put("countJoins", "n");

}

if(object.get("countJoins").toString().equals("0")) {

object.put("countJoins", "1");

}

object.put("joinBaseName", acknowledge[1]);

object.put("joinTableName", acknowledge[2]);

}

public static void processExec(String[] acknowledge, Map<String, Object> object, boolean mod) throws Exception {

if(object.get("start").toString().equals("1")) {

if(!acknowledge[0].equalsIgnoreCase(object.get("lastCommand").toString())

&&(object.get("lastCommand").toString().contains("changeCulumnName")

||object.get("lastCommand").toString().contains("culumnValue")

||object.get("lastCommand").toString().contains("condition")

||object.get("lastCommand").toString().contains("relation")

||object.get("lastCommand").toString().contains("aggregation")

||object.get("lastCommand").toString().contains("getCulumns")

||object.get("lastCommand").toString().contains("culumnName")

||object.get("lastCommand").toString().contains("relation"))) {

processExecKernel(object, mod);

}

}

}

private static void processExecKernel(Map<String, Object> object, boolean mod) throws Exception{

if(object.get("type").toString().equalsIgnoreCase("select") &&

(object.get("countJoins").toString().equalsIgnoreCase("0") ||

(object.get("countJoins").toString().equalsIgnoreCase("1") && object.get("newCommand").toString().equalsIgnoreCase("join")))){

if(object.containsKey("condition")) {

object.put("obj", SelectRowsImp.selectRowsByAttributesOfCondition(object));

}

if(object.containsKey("aggregation")) {

object.put("obj", SelectRowsImp.selectRowsByAttributesOfAggregation(object));

}

if(object.containsKey("getCulumns")) {

object.put("obj", SelectRowsImp.selectRowsByAttributesOfGetCulumns(object));

}

object.remove("recordRows");

}

if(object.get("type").toString().equalsIgnoreCase("select") &&

(object.get("countJoins").toString().equalsIgnoreCase("n") ||

(object.get("countJoins").toString().equalsIgnoreCase("1") && !object.get("newCommand").toString().equalsIgnoreCase("join")))){

if(object.containsKey("condition")) {

object.put("joinObj", SelectJoinRowsImp.selectRowsByAttributesOfJoinCondition(object));

}

if(object.containsKey("relation")) {

object.put("obj", SelectJoinRowsImp.selectRowsByAttributesOfJoinRelation(object));

}

if(object.containsKey("aggregation")) {

object.put("obj", SelectJoinRowsImp.selectRowsByAttributesOfJoinAggregation(object));

}

if(object.containsKey("getCulumns")) {

object.put("joinObj", SelectJoinRowsImp.selectRowsByAttributesOfJoinGetCulumns(object));

}

object.remove("recordRows");

}

if(object.get("type").toString().equalsIgnoreCase("create")){

if(object.containsKey("culumnName")) {

CreateTablesImp.createTable(object, mod);

}

object.remove("recordRows");

}

//离散数学的conjuction变换 a^&&b^&&c \* kernel[] = (a^&&b^)^^&&c \* kernel[] = (a||b)^&&c \* kernel[]

if(object.get("type").toString().equalsIgnoreCase("update") &&

(object.get("countJoins").toString().equalsIgnoreCase("0") ||

(object.get("countJoins").toString().equalsIgnoreCase("1") && object.get("newCommand").toString().equalsIgnoreCase("join")))){

if(object.containsKey("condition")) {

object.put("updateObj", UpdateRowsImp.updateRowsByAttributesOfCondition(object, mod));

}

if(object.containsKey("aggregation")) {

object.put("updateObj", UpdateRowsImp.updateRowsByAttributesOfAggregation(object, mod));

}

if(object.containsKey("culumnValue")) {

UpdateRowsImp.updateRowsByRecordConditions(object, mod);

}

object.remove("recordRows");

}

if(object.get("type").toString().equalsIgnoreCase("update") &&

(object.get("countJoins").toString().equalsIgnoreCase("n") ||

(object.get("countJoins").toString().equalsIgnoreCase("1") && !object.get("newCommand").toString().equalsIgnoreCase("join")))){

if(object.containsKey("condition")) {

object.put("updateJoinObj", UpdateJoinRowsImp.updateRowsByAttributesOfJoinCondition

(object, mod));

}

if(object.containsKey("relation")) {

object.put("updateObj", UpdateJoinRowsImp.updateRowsByAttributesOfJoinRelation

(object, mod));

}

if(object.containsKey("aggregation")) {

object.put("updateObj", UpdateJoinRowsImp.updateRowsByAttributesOfJoinAggregation

(object, mod));

}

if(object.containsKey("culumnValue")) {

UpdateRowsImp.updateRowsByRecordConditions(object, mod);

}

object.remove("recordRows");

}

if(object.get("type").toString().equalsIgnoreCase("insert")) {

if(object.containsKey("culumnValue")) {

InsertRowsImp.insertRowByAttributes(object, mod);

}

}

if(object.get("type").toString().equalsIgnoreCase("delete")) {

if(object.containsKey("condition")) {

DeleteRowsImp.deleteRowByAttributesOfCondition(object, mod);

}

}

object.remove("condition");

object.remove("culumnName");

object.remove("changeCulumnName");

object.remove("getCulumns");

object.remove("relation");

object.remove("aggregation");

object.remove("getCulumns");

object.put("start", "0");

}

public static void processCheck(String acknowledge, Map<String, Object> object, boolean mod) throws Exception {

if(object.get("start").toString().equals("1")) {

processExecKernel(object, mod);

}

List<Map<String, Object>> obj = ((List<Map<String, Object>>)(object.get("obj")));

int totalPages = 0;

if(obj != null) {

totalPages = obj.size();

}

int rowBeginIndex = object.containsKey("pageBegin")? Integer.valueOf(object.get("pageBegin").toString()):0;

int rowEndIndex = object.containsKey("pageEnd")?Integer.valueOf(object.get("pageEnd").toString()):totalPages>15?15:totalPages;

object.put("pageBegin", rowBeginIndex);

object.put("pageEnd", rowEndIndex);

String DBPath = CacheManager.getCacheInfo("DBPath").getValue().toString() + "/" + object.get("baseName").toString();

String DBTablePath = DBPath + "/" + object.get("tableName").toString();

object.put("tablePath", DBTablePath);

object.put("returnResult", "success");

object.put("totalPages",totalPages);

object.put("loginInfo", "success");

List<Object> spec = new ArrayList<>();

Iterator<String> iterator;

if(obj == null || obj.size() < 1) {

iterator = DetaDBBufferCacheManager.db.getBase(object.get("baseName").toString()).getTable(object.get("tableName").toString())

.getSpec().getCulumnTypes().keySet().iterator();

}else {

iterator = ((Map<String, Object>)obj.get(0).get("rowValue")).keySet().iterator();

}

while(iterator.hasNext()) {

spec.add(iterator.next());

}

object.put("spec", spec);

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.lyg.db.plsql.imp;

import java.math.BigDecimal;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

@SuppressWarnings({"unused"})

public class ProcessAggregationPLSQL {

public static void processAggregationLimitMap(String[] sets

, List<Map<String, Object>> output) {

List<Map<String, Object>> outputTemp = new ArrayList<>();

Iterator<Map<String, Object>> iterator = output.iterator();

int count = 0;

while(iterator.hasNext()) {

int rowid = count++;

Map<String, Object> row = iterator.next();

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

if(sets[1].equalsIgnoreCase("~")) {

if(rowid >= new BigDecimal(sets[0]).doubleValue() && rowid

<= new BigDecimal(sets[2]).doubleValue()) {

outputTemp.add(row);

}

}

}

output.clear();

output.addAll(outputTemp);

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.lyg.db.plsql.imp;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileReader;

import java.io.IOException;

import java.math.BigDecimal;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import org.lyg.cache.DetaDBBufferCacheManager;

import org.lyg.db.reflection.Cell;

import org.lyg.db.reflection.Row;

import org.lyg.db.reflection.Table;

@SuppressWarnings({ "unused", "unchecked"})

public class ProcessConditionPLSQL {

public static void processCache(String[] sets, List<Map<String, Object>> output

, String tableName, String baseName, Map<String, Object> object) {

Table table = DetaDBBufferCacheManager.db.getBase(baseName).getTable(tableName);

Iterator<String> iterator = table.getRows().keySet().iterator();

int rowindex=0;

while(iterator.hasNext()) {

int count = rowindex++;

String rowIndex = iterator.next();

Row row = table.getRow(rowIndex);

Cell cell=new Cell();

cell.setCellValue(rowIndex.replace("row", ""));

row.putCell("Index", cell);

if(sets[1].equalsIgnoreCase("<")||sets[1].equalsIgnoreCase("-lt")) {

double rowCellFromBigDecimal = new BigDecimal(row.getCell(sets[0])

.getCellValue().toString()).doubleValue();

if(rowCellFromBigDecimal < new BigDecimal(sets[2]).doubleValue()

&& row.containsCell("is\_delete\_0")) {

if(!((Map<Integer, Boolean>)(object.get("recordRows"))).containsKey(count)) {

output.add(rowToRowMap(row));

Map<Integer, Boolean> recordRows = (Map<Integer, Boolean>) object.get("recordRows");

recordRows.put(count, true);

}

}

}

if(sets[1].equalsIgnoreCase("<=")||sets[1].equalsIgnoreCase("=<")

||sets[1].equalsIgnoreCase("-lte")) {

double rowCellFromBigDecimal = new BigDecimal(row.getCell(sets[0])

.getCellValue().toString()).doubleValue();

if(rowCellFromBigDecimal<= new BigDecimal(sets[2]).doubleValue()

&& row.containsCell("is\_delete\_0")) {

if(!((Map<Integer, Boolean>)(object.get("recordRows"))).containsKey(count)) {

output.add(rowToRowMap(row));

Map<Integer, Boolean> recordRows = (Map<Integer, Boolean>) object.get("recordRows");

recordRows.put(count, true);

}

}

}

if(sets[1].equalsIgnoreCase("==")||sets[1].equalsIgnoreCase("=")

||sets[1].equalsIgnoreCase("===")) {

double rowCellFromBigDecimal = new BigDecimal(row.getCell(sets[0])

.getCellValue().toString()).doubleValue();

if(rowCellFromBigDecimal == new BigDecimal(sets[2]).doubleValue()

&& row.containsCell("is\_delete\_0")) {

if(!((Map<Integer, Boolean>)(object.get("recordRows"))).containsKey(count)) {

output.add(rowToRowMap(row));

Map<Integer, Boolean> recordRows = (Map<Integer, Boolean>) object.get("recordRows");

recordRows.put(count, true);

}

}

}

if(sets[1].equalsIgnoreCase(">=")||sets[1].equalsIgnoreCase("=>")

||sets[1].equalsIgnoreCase("-gte")) {

double rowCellFromBigDecimal = new BigDecimal(row.getCell(sets[0])

.getCellValue().toString()).doubleValue();

if(rowCellFromBigDecimal >= new BigDecimal(sets[2]).doubleValue()

&& row.containsCell("is\_delete\_0")) {

if(!((Map<Integer, Boolean>)(object.get("recordRows"))).containsKey(count)) {

output.add(rowToRowMap(row));

Map<Integer, Boolean> recordRows = (Map<Integer, Boolean>) object.get("recordRows");

recordRows.put(count, true);

}

}

}

if(sets[1].equalsIgnoreCase(">")||sets[1].equalsIgnoreCase("-gt")) {

double rowCellFromBigDecimal = new BigDecimal(row.getCell(sets[0])

.getCellValue().toString()).doubleValue();

if(rowCellFromBigDecimal > new BigDecimal(sets[2]).doubleValue()

&& row.containsCell("is\_delete\_0")) {

if(!((Map<Integer, Boolean>)(object.get("recordRows"))).containsKey(count)) {

output.add(rowToRowMap(row));

Map<Integer, Boolean> recordRows = (Map<Integer, Boolean>) object.get("recordRows");

recordRows.put(count, true);

}

}

}

if(sets[1].equalsIgnoreCase("!=")||sets[1].equalsIgnoreCase("=!")) {

double rowCellFromBigDecimal = new BigDecimal(row.getCell(sets[0])

.getCellValue().toString()).doubleValue();

if(rowCellFromBigDecimal != new BigDecimal(sets[2]).doubleValue()

&& row.containsCell("is\_delete\_0")) {

if(!((Map<Integer, Boolean>)(object.get("recordRows"))).containsKey(count)) {

output.add(rowToRowMap(row));

Map<Integer, Boolean> recordRows = (Map<Integer, Boolean>) object.get("recordRows");

recordRows.put(count, true);

}

}

}

if(sets[1].equalsIgnoreCase("equal") && row.containsCell("is\_delete\_0")) {

String rowCellFromString = row.getCell(sets[0]).getCellValue().toString();

if(rowCellFromString.equalsIgnoreCase(sets[2])) {

if(!((Map<Integer, Boolean>)(object.get("recordRows"))).containsKey(count)) {

output.add(rowToRowMap(row));

Map<Integer, Boolean> recordRows = (Map<Integer, Boolean>) object.get("recordRows");

recordRows.put(count, true);

}

}

}

if(sets[1].equalsIgnoreCase("!equal")) {

String rowCellFromString = row.getCell(sets[0]).getCellValue().toString();

if(!rowCellFromString.equalsIgnoreCase(sets[2]) && row.containsCell("is\_delete\_0")) {

if(!((Map<Integer, Boolean>)(object.get("recordRows"))).containsKey(count)) {

output.add(rowToRowMap(row));

Map<Integer, Boolean> recordRows = (Map<Integer, Boolean>) object.get("recordRows");

recordRows.put(count, true);

}

}

}

if(sets[1].equalsIgnoreCase("in")) {

String rowCellFromString = row.getCell(sets[0]).getCellValue().toString();

String set = "," + sets[2] + ",";

if(set.contains("," + rowCellFromString + ",") && row.containsCell("is\_delete\_0")){

if(!((Map<Integer, Boolean>)(object.get("recordRows"))).containsKey(count)) {

output.add(rowToRowMap(row));

Map<Integer, Boolean> recordRows = (Map<Integer, Boolean>) object.get("recordRows");

recordRows.put(count, true);

}

}

}

if(sets[1].equalsIgnoreCase("!in")) {

String rowCellFromString = row.getCell(sets[0]).getCellValue().toString();

String set = "," + sets[2] + ",";

if(!set.contains("," + rowCellFromString + ",") && row.containsCell("is\_delete\_0")){

if(!((Map<Integer, Boolean>)(object.get("recordRows"))).containsKey(count)) {

output.add(rowToRowMap(row));

Map<Integer, Boolean> recordRows = (Map<Integer, Boolean>) object.get("recordRows");

recordRows.put(count, true);

}

}

}

}

}

//以后优化成统一对象输出，不需要再转换。2019-1-15 tin

private static Map<String, Object> rowToRowMap(Row row) {

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

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

Iterator<String> iterator = row.getCells().keySet().iterator();

while(iterator.hasNext()) {

String cellName = iterator.next();

if(!cellName.contains("is\_delete")) {

Cell cell = row.getCell(cellName);

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

culumnMap.put("culumnName", cellName);

culumnMap.put("culumnValue", cell.getCellValue().toString());

culumnMaps.put(cellName, culumnMap);

}

}

rowMap.put("rowValue", culumnMaps);

return rowMap;

}

public static void processMap(String[] sets, List<Map<String, Object>> output, String dBTablePath) {

List<Map<String, Object>> outputTemp = new ArrayList<>();

Iterator<Map<String, Object>> iterator = output.iterator();

int rowid = 0;

while(iterator.hasNext()) {

Map<String, Object> row = iterator.next();

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

if(sets[1].equalsIgnoreCase("<")||sets[1].equalsIgnoreCase("-lt")) {

String rowCellFromString = ((Map<String, Object>)(((Map<String, Object>)(row.get("rowValue")))

.get(sets[0]))).get("culumnValue").toString();

if(new BigDecimal(rowCellFromString).doubleValue() < new BigDecimal(sets[2]).doubleValue()) {

outputTemp.add(row);

}

}

if(sets[1].equalsIgnoreCase("<=")||sets[1].equalsIgnoreCase("=<")

||sets[1].equalsIgnoreCase("-lte")) {

String rowCellFromString = ((Map<String, Object>)(((Map<String, Object>)(row.get("rowValue")))

.get(sets[0]))).get("culumnValue").toString();

if(new BigDecimal(rowCellFromString).doubleValue() <= new BigDecimal(sets[2]).doubleValue()) {

outputTemp.add(row);

}

}

if(sets[1].equalsIgnoreCase("==")||sets[1].equalsIgnoreCase("=")||sets[1].equalsIgnoreCase("===")) {

String rowCellFromString = ((Map<String, Object>)(((Map<String, Object>)(row.get("rowValue")))

.get(sets[0]))).get("culumnValue").toString();

if(new BigDecimal(rowCellFromString).doubleValue() == new BigDecimal(sets[2]).doubleValue()) {

outputTemp.add(row);

}

}

if(sets[1].equalsIgnoreCase(">=")||sets[1].equalsIgnoreCase("=>")

||sets[1].equalsIgnoreCase("-gte")) {

String rowCellFromString = ((Map<String, Object>)(((Map<String, Object>)(row.get("rowValue")))

.get(sets[0]))).get("culumnValue").toString();

if(new BigDecimal(rowCellFromString).doubleValue() >= new BigDecimal(sets[2]).doubleValue()) {

outputTemp.add(row);

}

}

if(sets[1].equalsIgnoreCase(">")||sets[1].equalsIgnoreCase("-gt")) {

String rowCellFromString = ((Map<String, Object>)(((Map<String, Object>)(row.get("rowValue")))

.get(sets[0]))).get("culumnValue").toString();

if(new BigDecimal(rowCellFromString).doubleValue() > new BigDecimal(sets[2]).doubleValue()) {

outputTemp.add(row);

}

}

if(sets[1].equalsIgnoreCase("!=")||sets[1].equalsIgnoreCase("=!")) {

String rowCellFromString = ((Map<String, Object>)(((Map<String, Object>)(row.get("rowValue")))

.get(sets[0]))).get("culumnValue").toString();

if(new BigDecimal(rowCellFromString).doubleValue() != new BigDecimal(sets[2]).doubleValue()) {

outputTemp.add(row);

}

}

if(sets[1].equalsIgnoreCase("equal")) {

String rowCellFromString = ((Map<String, Object>)(((Map<String, Object>)(row.get("rowValue")))

.get(sets[0]))).get("culumnValue").toString();

if(rowCellFromString.equalsIgnoreCase(sets[2])) {

outputTemp.add(row);

}

}

if(sets[1].equalsIgnoreCase("!equal")) {

String rowCellFromString = ((Map<String, Object>)(((Map<String, Object>)(row.get("rowValue")))

.get(sets[0]))).get("culumnValue").toString();

if(!rowCellFromString.equalsIgnoreCase(sets[2])) {

outputTemp.add(row);

}

}

if(sets[1].equalsIgnoreCase("in")) {

String rowCellFromString = ((Map<String, Object>)(((Map<String, Object>)(row.get("rowValue")))

.get(sets[0]))).get("culumnValue").toString();

String set = "," + sets[2] + ",";

if(set.contains("," + rowCellFromString + ",")){

outputTemp.add(row);

}

}

if(sets[1].equalsIgnoreCase("!in")) {

String rowCellFromString = ((Map<String, Object>)(((Map<String, Object>)(row.get("rowValue")))

.get(sets[0]))).get("culumnValue").toString();

String set = "," + sets[2] + ",";

if(!set.contains("," + rowCellFromString + ",")){

outputTemp.add(row);

}

}

}

output.clear();

output.addAll(outputTemp);

}

public static void processTable(String[] sets, List<Map<String, Object>> output

, String DBTablePath, Map<String, Object> object) throws IOException {

String DBTableRowsPath = DBTablePath + "/rows";

File fileDBTableRowsPath = new File(DBTableRowsPath);

if (fileDBTableRowsPath.isDirectory()) {

String[] rowList = fileDBTableRowsPath.list();

int count=0;

NextRow:

for(String row: rowList) {

count++;

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

String DBTableRowIndexPath = DBTablePath + "/rows/" + row;

File readDBTableRowIndexFile = new File(DBTableRowIndexPath);

if (readDBTableRowIndexFile.isDirectory()) {

String isDelete = DBTableRowIndexPath + "/is\_delete\_1" ;

File isDeleteFile = new File(isDelete);

if(isDeleteFile.exists()) {

continue NextRow;

}

String DBTableRowIndexCulumnPath = DBTableRowIndexPath + "/" + sets[0];

File readDBTableRowIndexCulumnFile = new File(DBTableRowIndexCulumnPath);

if(readDBTableRowIndexCulumnFile.isDirectory()) {

BufferedReader reader = new BufferedReader

(new FileReader(readDBTableRowIndexCulumnFile + "/" + "value.lyg"));

String temp = "";

String tempString = "";

while ((tempString = reader.readLine()) != null) {

temp += tempString;

}

reader.close();

if(sets[1].equalsIgnoreCase("<")||sets[1].equalsIgnoreCase("-lt")) {

if(new BigDecimal(temp.toString()).doubleValue()

< new BigDecimal(sets[2].toString()).doubleValue()) {

if(!((Map<Integer, Boolean>)(object.get("recordRows"))).containsKey(count)) {

processkernel(row, readDBTableRowIndexCulumnFile

, readDBTableRowIndexCulumnFile, reader

, row, output, row, rowMap);

Map<Integer, Boolean> recordRows = (Map<Integer, Boolean>) object.get("recordRows");

recordRows.put(count, true);

}

}

}

if(sets[1].equalsIgnoreCase("<=")||sets[1].equalsIgnoreCase("=<")

||sets[1].equalsIgnoreCase("-lte")) {

if(new BigDecimal(temp.toString()).doubleValue()

<= new BigDecimal(sets[2].toString()).doubleValue()) {

if(!((Map<Integer, Boolean>)(object.get("recordRows"))).containsKey(count)) {

processkernel(row, readDBTableRowIndexCulumnFile

, readDBTableRowIndexCulumnFile, reader

, row, output, row, rowMap);

Map<Integer, Boolean> recordRows = (Map<Integer, Boolean>) object.get("recordRows");

recordRows.put(count, true);

}

}

}

if(sets[1].equalsIgnoreCase("==")||sets[1].equalsIgnoreCase("=")

||sets[1].equalsIgnoreCase("===")) {

if(new BigDecimal(temp.toString()).doubleValue()

== new BigDecimal(sets[2].toString()).doubleValue()) {

if(!((Map<Integer, Boolean>)(object.get("recordRows"))).containsKey(count)) {

processkernel(row, readDBTableRowIndexCulumnFile

, readDBTableRowIndexCulumnFile, reader

, row, output, row, rowMap);

Map<Integer, Boolean> recordRows = (Map<Integer, Boolean>) object.get("recordRows");

recordRows.put(count, true);

}

}

}

if(sets[1].equalsIgnoreCase(">=")||sets[1].equalsIgnoreCase("=>")

||sets[1].equalsIgnoreCase("-gte")) {

if(new BigDecimal(temp.toString()).doubleValue()

>= new BigDecimal(sets[2].toString()).doubleValue()) {

if(!((Map<Integer, Boolean>)(object.get("recordRows"))).containsKey(count)) {

processkernel(row, readDBTableRowIndexCulumnFile

, readDBTableRowIndexCulumnFile, reader

, row, output, row, rowMap);

Map<Integer, Boolean> recordRows = (Map<Integer, Boolean>) object.get("recordRows");

recordRows.put(count, true);

}

}

}

if(sets[1].equalsIgnoreCase(">")||sets[1].equalsIgnoreCase("-gt")) {

if(new BigDecimal(temp.toString()).doubleValue()

> new BigDecimal(sets[2].toString()).doubleValue()) {

if(!((Map<Integer, Boolean>)(object.get("recordRows"))).containsKey(count)) {

processkernel(row, readDBTableRowIndexCulumnFile

, readDBTableRowIndexCulumnFile, reader

, row, output, row, rowMap);

Map<Integer, Boolean> recordRows = (Map<Integer, Boolean>) object.get("recordRows");

recordRows.put(count, true);

}

}

}

if(sets[1].equalsIgnoreCase("!=")||sets[1].equalsIgnoreCase("=!")) {

if(new BigDecimal(temp.toString()).doubleValue()

!= new BigDecimal(sets[2].toString()).doubleValue()) {

if(!((Map<Integer, Boolean>)(object.get("recordRows"))).containsKey(count)) {

processkernel(row, readDBTableRowIndexCulumnFile

, readDBTableRowIndexCulumnFile, reader

, row, output, row, rowMap);

Map<Integer, Boolean> recordRows = (Map<Integer, Boolean>) object.get("recordRows");

recordRows.put(count, true);

}

}

}

if(sets[1].equalsIgnoreCase("equal")) {

String rowCellFromString = temp.toString();

if(rowCellFromString.equalsIgnoreCase(sets[2].toString())) {

if(!((Map<Integer, Boolean>)(object.get("recordRows"))).containsKey(count)) {

processkernel(row, readDBTableRowIndexCulumnFile

, readDBTableRowIndexCulumnFile, reader

, row, output, row, rowMap);

Map<Integer, Boolean> recordRows

= (Map<Integer, Boolean>) object.get("recordRows");

recordRows.put(count, true);

}

}

}

if(sets[1].equalsIgnoreCase("!equal")) {

String rowCellFromString = temp.toString();

if(!rowCellFromString.equalsIgnoreCase(sets[2].toString())) {

if(!((Map<Integer, Boolean>)(object.get("recordRows"))).containsKey(count)) {

processkernel(row, readDBTableRowIndexCulumnFile, readDBTableRowIndexCulumnFile, reader

, row, output, row, rowMap);

Map<Integer, Boolean> recordRows = (Map<Integer, Boolean>) object.get("recordRows");

recordRows.put(count, true);

}

}

}

if(sets[1].equalsIgnoreCase("in")) {

String rowCellFromString = temp.toString();

String set = "," + sets[2] + ",";

if(set.contains("," + rowCellFromString + ",")) {

if(!((Map<Integer, Boolean>)(object.get("recordRows")))

.containsKey(count)) {

processkernel(row, readDBTableRowIndexCulumnFile

, readDBTableRowIndexCulumnFile, reader

, row, output, row, rowMap);

Map<Integer, Boolean> recordRows = (Map<Integer, Boolean>) object.get("recordRows");

recordRows.put(count, true);

}

}

}

if(sets[1].equalsIgnoreCase("!in")) {

String rowCellFromString = temp.toString();

String set = "," + sets[2] + ",";

if(!set.contains("," + rowCellFromString + ",")) {

if(!((Map<Integer, Boolean>)(object.get("recordRows"))).containsKey(count)) {

processkernel(row, readDBTableRowIndexCulumnFile

, readDBTableRowIndexCulumnFile, reader

, row, output, row, rowMap);

Map<Integer, Boolean> recordRows

= (Map<Integer, Boolean>) object.get("recordRows");

recordRows.put(count, true);

}

}

}

}

}

}

}

}

private static void processkernel(String temp, File readDBTableRowIndexCulumnFile, File readDBTableRowIndexFile

, BufferedReader reader, String DBTableRowIndexPath, List<Map<String, Object>> output, String tempString

, Map<String, Object> rowMap) throws IOException {

String[] culumnList = readDBTableRowIndexFile.list();

NextFile:

for(String culumn: culumnList) {

if(culumn.contains("is\_delete")) {

continue NextFile;

}

String DBTableCulumnIndexPath = DBTableRowIndexPath + "/" + culumn;

File readDBTableCulumnIndexPathFile = new File(DBTableCulumnIndexPath);

if (readDBTableRowIndexCulumnFile.isDirectory()) {

reader = new BufferedReader(new FileReader(readDBTableCulumnIndexPathFile + "/" + "value.lyg"));

temp = "";

while ((tempString = reader.readLine()) != null) {

temp += tempString;

}

reader.close();

rowMap.put(culumn, temp);

}else {

rowMap.put(culumn, null);

}

}

output.add(rowMap);

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.lyg.db.plsql.imp;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

@SuppressWarnings({"unused", "unchecked"})

public class ProcessGetCulumnsPLSQL {

public static Object getCulumnsMapWithAs(String[] sets, Map<String, Object> row) {

return row.get(sets[2]);

}

public static Object getCulumnsMap(String[] sets, Map<String, Object> row) {

return row.get(sets[0]);

}

public static Object processGetCulumnsMap(List<Map<String, Object>> obj, String[] getCulumnsValueArray) {

List<Map<String, Object>> newobj = new ArrayList<Map<String, Object>>();

Iterator<Map<String, Object>> iterator = obj.iterator();

int count = 0;

NextRow:

while(iterator.hasNext()) {

int rowId = count ++;

Map<String, Object> row = iterator.next();

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

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

NextCell:

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

String[] sets = getCulumnsValueArray[i].split("\\|");

if(null != sets && ((Map<String, Object>)row.get("rowValue")).containsKey(sets[0])) {

Map<String, Object> cell

= (Map<String, Object>)((Map<String, Object>)row.get("rowValue")).get(sets[0]);

if(1 == sets.length) {

rowValue.put(sets[0], cell);

continue NextCell;

}

if(3 == sets.length && sets[1].equalsIgnoreCase("as")) {

cell.put("culumnName", sets[2]);

rowValue.put(sets[2], cell);

continue NextCell;

}

}

}

newRow.put("rowValue", rowValue);

newobj.add(newRow);

}

obj.clear();

return obj.addAll(newobj);

}

}

// i' m tin god

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.lyg.db.plsql.imp;

import java.math.BigDecimal;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

@SuppressWarnings({"unused","unchecked"})

public class ProcessRelationPLSQL {

public static void processAndMap(String[] sets, List<Map<String, Object>> obj

, List<Map<String, Object>> joinObj

, Map<String, Object> object, List<Map<String, Object>> newObj) {

List<Map<String, Object>> newObjTemp = new ArrayList<>();

Iterator<Map<String, Object>> iterator = newObj.iterator();

int count = 0;

while(iterator.hasNext()) {

int objRowId = count++;

Map<String, Object> objRow = iterator.next();

if(objRow.containsKey(sets[0])&&objRow.containsKey(sets[2])) {

if(sets[1].equalsIgnoreCase("==") || sets[1].equalsIgnoreCase("===")) {

if(new BigDecimal(objRow.get(sets[0]).toString()).doubleValue()

== new BigDecimal(objRow.get(sets[2]).toString()).doubleValue()) {

newObjTemp.add(objRow);

}

}

if(sets[1].equalsIgnoreCase("!=") || sets[1].equalsIgnoreCase("=!")

|| sets[1].equalsIgnoreCase("<>") || sets[1].equalsIgnoreCase("><")) {

if(new BigDecimal(objRow.get(sets[0]).toString()).doubleValue()

!= new BigDecimal(objRow.get(sets[2]).toString()).doubleValue()) {

newObjTemp.add(objRow);

}

}

if(sets[1].equalsIgnoreCase(">=") || sets[1].equalsIgnoreCase("=>")) {

if(new BigDecimal(objRow.get(sets[0]).toString()).doubleValue()

>= new BigDecimal(objRow.get(sets[2]).toString()).doubleValue()) {

newObjTemp.add(objRow);

}

}

if(sets[1].equalsIgnoreCase(">")) {

if(new BigDecimal(objRow.get(sets[0]).toString()).doubleValue()

> new BigDecimal(objRow.get(sets[2]).toString()).doubleValue()) {

newObjTemp.add(objRow);

}

}

if(sets[1].equalsIgnoreCase("<")) {

if(new BigDecimal(objRow.get(sets[0]).toString()).doubleValue()

< new BigDecimal(objRow.get(sets[2]).toString()).doubleValue()) {

newObjTemp.add(objRow);

}

}

if(sets[1].equalsIgnoreCase("<=") || sets[1].equalsIgnoreCase("<=")) {

if(new BigDecimal(objRow.get(sets[0]).toString()).doubleValue()

<= new BigDecimal(objRow.get(sets[2]).toString()).doubleValue()) {

newObjTemp.add(objRow);

}

}

if(sets[1].equalsIgnoreCase("equal")) {

if(objRow.get(sets[0]).toString().equals(objRow.get(sets[2]).toString())){

newObjTemp.add(objRow);

}

}

if(sets[1].equalsIgnoreCase("!equal") || sets[1].equalsIgnoreCase("equal!")) {

if(!objRow.get(sets[0]).toString().equals(objRow.get(sets[2]).toString())){

newObjTemp.add(objRow);

}

}

if(sets[1].equalsIgnoreCase("in")) {

String set = "," + objRow.get(sets[2]).toString() + ",";

if(set.contains(objRow.get(sets[0]).toString())){

newObjTemp.add(objRow);

}

}

if(sets[1].equalsIgnoreCase("!in")) {

String set = "," + objRow.get(sets[2]).toString() + ",";

if(!set.contains(objRow.get(sets[0]).toString())){

newObjTemp.add(objRow);

}

}

}

}

}

public static void processOrMap(String[] sets, List<Map<String, Object>> obj

, List<Map<String, Object>> joinObj

, Map<String, Object> object, List<Map<String, Object>> newObj

, Map<String, Boolean> findinNewObj) {

Iterator<Map<String, Object>> iterator = obj.iterator();

int count = 0;

while(iterator.hasNext()) {

int objRowId = count++;

Map<String, Object> objRow = iterator.next();

Map<String, Object> row = (Map<String, Object>) objRow.get("rowValue");

Iterator<Map<String, Object>> iteratorJoin = joinObj.iterator();

int countJoin = 0;

while(iteratorJoin.hasNext()) {

int objJoinRowId = countJoin++;

Map<String, Object> objJoinRow = iteratorJoin.next();

Map<String, Object> joinRow = (Map<String, Object>) objJoinRow.get("rowValue");

Map<String, Object> cell = (Map<String, Object>) row.get(sets[0]);

Map<String, Object> cellJoin = (Map<String, Object>) joinRow.get(sets[2]);

if(sets[1].equalsIgnoreCase("==") || sets[1].equalsIgnoreCase("===")) {

if(new BigDecimal(cell.get("culumnValue").toString()).doubleValue()

== new BigDecimal(cellJoin.get("culumnValue").toString()).doubleValue()) {

if(!findinNewObj.containsKey(objRowId + ":" + objJoinRowId)) {

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

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

newRow.putAll((Map<? extends String, ? extends Object>) objJoinRow.get("rowValue"));

newRow.putAll((Map<? extends String, ? extends Object>) objRow.get("rowValue"));

newObjRow.put("rowValue", newRow);

newObj.add(newObjRow);

findinNewObj.put(objRowId + ":" + objJoinRowId, true);

}

}

**package** org.lyg.db.reflection;

**public** **class** Cell{

**public** Object getCellValue() {

**return** cellValue;

}

**public** **void** setCellValue(Object cellValue) {

**this**.cellValue = cellValue;

}

**private** Object cellValue;

} }

if(sets[1].equalsIgnoreCase("equal")) {

if(cell.get("culumnValue").toString().equals(cellJoin.get("culumnValue").toString())) {

if(!findinNewObj.containsKey(objRowId + ":" + objJoinRowId)) {

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

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

newRow.putAll((Map<? extends String, ? extends Object>) objJoinRow.get("rowValue"));

newRow.putAll((Map<? extends String, ? extends Object>) objRow.get("rowValue"));

newObjRow.put("rowValue", newRow);

newObj.add(newObjRow);

findinNewObj.put(objRowId + ":" + objJoinRowId, true);

}

}

}

}

}

}

}}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.lyg.db.select.imp;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileReader;

import java.io.IOException;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import java.util.concurrent.ConcurrentHashMap;

import cacheProcessor.CacheManager;

import org.lyg.cache.DetaDBBufferCacheManager;

import org.lyg.db.plsql.imp.ProcessAggregationPLSQL;

import org.lyg.db.plsql.imp.ProcessConditionPLSQL;

import org.lyg.db.plsql.imp.ProcessGetCulumnsPLSQL;

import org.lyg.db.plsql.imp.ProcessRelationPLSQL;

import org.lyg.db.reflection.Spec;

@SuppressWarnings({"unused", "unchecked"})

public class SelectJoinRowsImp {

public static Object selectRowsByAttributesOfJoinCondition(Map<String, Object> object)

throws IOException {

if(!object.containsKey("recordRows")) {

Map<String, Boolean> recordRows = new ConcurrentHashMap<>();

object.put("recordRows", recordRows);

}

Spec spec = new Spec();

spec.setCulumnTypes(new ConcurrentHashMap<String, String>());

String objectType = "";

List<Map<String, Object>> output = new ArrayList<>();

//锁定数据库

String DBPath = CacheManager.getCacheInfo("DBPath").getValue().toString() + "/"

+ object.get("joinBaseName").toString();

//锁定表

File fileDBPath = new File(DBPath);

if (fileDBPath.isDirectory()) {

String DBTablePath = DBPath + "/" + object.get("joinTableName").toString();

File fileDBTable = new File(DBTablePath);

if (fileDBTable.isDirectory()) {

String DBTableCulumnPath = DBTablePath + "/spec";

File fileDBTableCulumn = new File(DBTableCulumnPath);

if (fileDBTableCulumn.isDirectory()) {

//读取列数据格式

String[] fileList = fileDBTableCulumn.list();

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

File readDBTableSpecCulumnFile = new File(DBTableCulumnPath + "/"

+ fileList[0] + "/value.lyg");

BufferedReader reader = new BufferedReader(new FileReader(readDBTableSpecCulumnFile));

String tempString = null;

while ((tempString = reader.readLine()) != null) {

objectType = tempString;

}

reader.close();

spec.setCulumnType(fileList[i], objectType);

}

List<String[]> conditionValues = (List<String[]>) object.get("condition");

Iterator<String[]> iterator = conditionValues.iterator();

while(iterator.hasNext()) {

boolean overMap = output.size() == 0? false: true;

String[] conditionValueArray = iterator.next();

String type = conditionValueArray[1];

boolean andMap = type.equalsIgnoreCase("and")?true:false;

for(int i = 2; i < conditionValueArray.length; i++) {

String[] sets = conditionValueArray[i].split("\\|");

if(overMap && andMap) {

ProcessConditionPLSQL.processMap(sets, output, DBTablePath);//1

}else if(DetaDBBufferCacheManager.dbCache){

ProcessConditionPLSQL.processCache(sets, output

, object.get("joinTableName").toString()

, object.get("joinBaseName").toString(), object);//1

}else {

ProcessConditionPLSQL.processTable(sets, output, DBTablePath, object);//1

}

}

}

}

}

}

return output;

}

public static Object selectRowsByAttributesOfJoinAggregation(Map<String, Object> object) {

if(!object.containsKey("joinObj")) {

return new ArrayList<>();

}

List<Map<String, Object>> obj = ((List<Map<String, Object>>)(object.get("obj")));

List<String[]> aggregationValues = (List<String[]>) object.get("aggregation");

Iterator<String[]> iterator = aggregationValues.iterator();

while(iterator.hasNext()) {

boolean overMap = obj.size() == 0? false: true;

String[] aggregationValueArray = iterator.next();

String type = aggregationValueArray[1];

boolean limitMap = type.equalsIgnoreCase("limit")?true:false;

for(int i = 2; i < aggregationValueArray.length; i++) {

String[] sets = aggregationValueArray[i].split("\\|");

if(limitMap) {

ProcessAggregationPLSQL.processAggregationLimitMap(sets, obj);

}

//基于sort key 前序treeMap 之后排序功能设计

//基于sort key 后序treeMap

}

}

return obj;

}

public static Object selectRowsByAttributesOfJoinGetCulumns(Map<String, Object> object) {

if(!object.containsKey("joinObj")) {

return new ArrayList<>();

}

List<Map<String, Object>> obj = ((List<Map<String, Object>>)(object.get("joinObj")));

List<String[]> getCulumnsValues = (List<String[]>) object.get("getCulumns");

Iterator<String[]> iterator = getCulumnsValues.iterator();

while(iterator.hasNext()) {

boolean overMap = obj.size() == 0? false: true;

String[] getCulumnsValueArray = iterator.next();

if(overMap) {

ProcessGetCulumnsPLSQL.processGetCulumnsMap(obj, getCulumnsValueArray);

}

}

return obj;

}

public static Object selectRowsByAttributesOfJoinRelation(Map<String, Object> object) {

if(!object.containsKey("obj")||!object.containsKey("joinObj")) {

return new ArrayList<>();

}

Map<String,Boolean> findinNewObj = new HashMap<>();

List<Map<String, Object>> newObj = new ArrayList<Map<String, Object>>();

List<Map<String, Object>> obj = ((List<Map<String, Object>>)(object.get("obj")));

List<Map<String, Object>> joinObj= ((List<Map<String, Object>>)(object.get("joinObj")));

List<String[]> relationValues = (List<String[]>) object.get("relation");

Iterator<String[]> iterator = relationValues.iterator();

while(iterator.hasNext()) {

boolean overObjMap= obj.size() == 0? false: true;

boolean overJoinObjMap= joinObj.size() == 0? false: true;

String[] getRelationValueArray = iterator.next();

String type = getRelationValueArray[1];

boolean andMap = type.equalsIgnoreCase("and")?true:false;

for(int i= 2; i< getRelationValueArray.length; i++) {

String[] sets = getRelationValueArray[i].split("\\|");

if(overObjMap&& overJoinObjMap&&andMap && i>2) {

ProcessRelationPLSQL.processAndMap(sets, obj, joinObj,object, newObj);

}else {

ProcessRelationPLSQL.processOrMap(sets, obj, joinObj, object

, newObj, findinNewObj);

}

}

}

return newObj;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.lyg.db.select.imp;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileReader;

import java.io.IOException;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import java.util.concurrent.ConcurrentHashMap;

import cacheProcessor.CacheManager;

import org.lyg.cache.DetaDBBufferCacheManager;

import org.lyg.db.plsql.imp.ProcessAggregationPLSQL;

import org.lyg.db.plsql.imp.ProcessConditionPLSQL;

import org.lyg.db.plsql.imp.ProcessGetCulumnsPLSQL;

import org.lyg.db.plsql.imp.ProcessRelationPLSQL;

import org.lyg.db.reflection.Spec;

@SuppressWarnings({"unused", "unchecked"})

public class SelectNestRowsImp {

public static Object selectRowsByAttributesOfNestCondition(Map<String

, Object> object) throws IOException {

if(!object.containsKey("recordRows")) {

Map<String, Boolean> recordRows = new ConcurrentHashMap<>();

object.put("recordRows", recordRows);

}

Spec spec = new Spec();

spec.setCulumnTypes(new ConcurrentHashMap<String, String>());

String objectType = "";

List<Map<String, Object>> output = new ArrayList<>();

//锁定数据库

String DBPath = CacheManager.getCacheInfo("DBPath").getValue().toString() + "/"

+ object.get("nestBaseName").toString();

//锁定表

File fileDBPath = new File(DBPath);

if (fileDBPath.isDirectory()) {

String DBTablePath = DBPath + "/" + object.get("nestTableName").toString();

File fileDBTable = new File(DBTablePath);

if (fileDBTable.isDirectory()) {

String DBTableCulumnPath = DBTablePath + "/spec";

File fileDBTableCulumn = new File(DBTableCulumnPath);

if (fileDBTableCulumn.isDirectory()) {

//读取列数据格式

String[] fileList = fileDBTableCulumn.list();

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

File readDBTableSpecCulumnFile = new File(DBTableCulumnPath + "/"

+ fileList[0] + "/value.lyg");

BufferedReader reader = new BufferedReader(new FileReader(readDBTableSpecCulumnFile));

String tempString = null;

while ((tempString = reader.readLine()) != null) {

objectType = tempString;

}

reader.close();

spec.setCulumnType(fileList[i], objectType);

}

List<String[]> conditionValues = (List<String[]>) object.get("condition");

Iterator<String[]> iterator = conditionValues.iterator();

while(iterator.hasNext()) {

boolean overMap = output.size() == 0? false: true;

String[] conditionValueArray = iterator.next();

String type = conditionValueArray[1];

boolean andMap = type.equalsIgnoreCase("and")?true:false;

for(int i = 2; i < conditionValueArray.length; i++) {

String[] sets = conditionValueArray[i].split("\\|");

if(overMap && andMap) {

ProcessConditionPLSQL.processMap(sets, output, DBTablePath);

}else if(DetaDBBufferCacheManager.dbCache){

ProcessConditionPLSQL.processCache(sets, output

, object.get("nestTableName").toString()

, object.get("nestBaseName").toString(), object);

}else {

ProcessConditionPLSQL.processTable(sets, output, DBTablePath, object);

}

}

}

}

}

}

return output;

}

public static Object selectRowsByAttributesOfNestAggregation(Map<String, Object> object) {

if(!object.containsKey("joinObj")) {

return new ArrayList<>();

}

List<Map<String, Object>> obj = ((List<Map<String, Object>>)(object.get("obj")));

List<String[]> aggregationValues = (List<String[]>) object.get("aggregation");

Iterator<String[]> iterator = aggregationValues.iterator();

while(iterator.hasNext()) {

boolean overMap = obj.size() == 0? false: true;

String[] aggregationValueArray = iterator.next();

String type = aggregationValueArray[1];

boolean limitMap = type.equalsIgnoreCase("limit")?true:false;

for(int i = 2; i < aggregationValueArray.length; i++) {

String[] sets = aggregationValueArray[i].split("\\|");

if(limitMap) {

ProcessAggregationPLSQL.processAggregationLimitMap(sets, obj);

}

//基于sort key 前序treeMap 之后排序功能设计

//基于sort key 后序treeMap

}

}

return obj;

}

public static Object selectRowsByAttributesOfNestGetCulumns(Map<String, Object> object) {

if(!object.containsKey("joinObj")) {

return new ArrayList<>();

}

List<Map<String, Object>> obj = ((List<Map<String, Object>>)(object.get("joinObj")));

List<String[]> getCulumnsValues = (List<String[]>) object.get("getCulumns");

Iterator<String[]> iterator = getCulumnsValues.iterator();

while(iterator.hasNext()) {

boolean overMap = obj.size() == 0? false: true;

String[] getCulumnsValueArray = iterator.next();

if(overMap) {

ProcessGetCulumnsPLSQL.processGetCulumnsMap(obj, getCulumnsValueArray);

}

}

return obj;

}

public static Object selectRowsByAttributesOfNestRelation(Map<String, Object> object) {

if(!object.containsKey("obj")||!object.containsKey("joinObj")) {

return new ArrayList<>();

}

Map<String,Boolean> findinNewObj = new HashMap<>();

List<Map<String, Object>> newObj = new ArrayList<Map<String, Object>>();

List<Map<String, Object>> obj = ((List<Map<String, Object>>)(object.get("obj")));

List<Map<String, Object>> joinObj= ((List<Map<String, Object>>)(object.get("joinObj")));

List<String[]> relationValues = (List<String[]>) object.get("relation");

Iterator<String[]> iterator = relationValues.iterator();

while(iterator.hasNext()) {

boolean overObjMap= obj.size() == 0? false: true;

boolean overJoinObjMap= joinObj.size() == 0? false: true;

String[] getRelationValueArray = iterator.next();

String type = getRelationValueArray[1];

boolean andMap = type.equalsIgnoreCase("and")?true:false;

for(int i= 2; i< getRelationValueArray.length; i++) {

String[] sets = getRelationValueArray[i].split("\\|");

if(overObjMap&& overJoinObjMap&&andMap && i>2) {

ProcessRelationPLSQL.processAndMap(sets, obj, joinObj, object, newObj);

}else {

ProcessRelationPLSQL.processOrMap(sets, obj, joinObj, object

, newObj, findinNewObj);

}

}

}

return newObj;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.lyg.db.select.imp;

import java.io.BufferedInputStream;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.IOException;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import java.util.concurrent.ConcurrentHashMap;

import cacheProcessor.CacheManager;

import org.lyg.cache.DetaDBBufferCacheManager;

import org.lyg.db.plsql.imp.ProcessAggregationPLSQL;

import org.lyg.db.plsql.imp.ProcessConditionPLSQL;

import org.lyg.db.plsql.imp.ProcessGetCulumnsPLSQL;

import org.lyg.db.reflection.Spec;

@SuppressWarnings({ "unused", "unchecked" })

public class SelectRowsImp {

public static List<Map<String, Object>> selectRowsByAttribute(String currentDB

, String tableName, String culmnName, Object value) throws IOException{

if(value==null) {

value="";

}

String objectType = "";

List<Map<String, Object>> output = new ArrayList<>();

//锁定数据库

String DBPath = CacheManager.getCacheInfo("DBPath").getValue().toString() + "/" + currentDB;

//锁定表

File fileDBPath = new File(DBPath);

if (fileDBPath.isDirectory()) {

String DBTablePath = DBPath + "/" + tableName;

File fileDBTable = new File(DBTablePath);

if (fileDBTable.isDirectory()) {

String DBTableCulumnPath = DBTablePath + "/spec/" + culmnName;

File fileDBTableCulumn = new File(DBTableCulumnPath);

if (fileDBTableCulumn.isDirectory()) {

//读取列数据格式

String[] fileList = fileDBTableCulumn.list();

File readDBTableSpecCulumnFile = new File(DBTableCulumnPath + "/" + fileList[0]);

BufferedReader reader = new BufferedReader(new FileReader(readDBTableSpecCulumnFile));

String tempString = null;

while ((tempString = reader.readLine()) != null) {

objectType = tempString;

}

reader.close();

if(objectType.contains("string")) {

String DBTableRowsPath = DBTablePath + "/rows";

File fileDBTableRowsPath = new File(DBTableRowsPath);

if (fileDBTableRowsPath.isDirectory()) {

String[] rowList = fileDBTableRowsPath.list();

NextRow:

for(String row: rowList) {

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

String DBTableRowIndexPath = DBTablePath + "/rows/" + row;

File readDBTableRowIndexFile = new File(DBTableRowIndexPath);

if (readDBTableRowIndexFile.isDirectory()) {

String isDelete = DBTableRowIndexPath + "/is\_delete\_1" ;

File isDeleteFile = new File(isDelete);

if(isDeleteFile.exists()) {

continue NextRow;

}

String DBTableRowIndexCulumnPath = DBTableRowIndexPath + "/" + culmnName;

File readDBTableRowIndexCulumnFile = new File(DBTableRowIndexCulumnPath);

if (readDBTableRowIndexCulumnFile.isDirectory()) {

reader = new BufferedReader(new FileReader(readDBTableRowIndexCulumnFile + "/" + "value.lyg"));

String temp="";

while ((tempString = reader.readLine()) != null) {

temp += tempString;

}

reader.close();

if(temp.equalsIgnoreCase(value.toString())) {

String[] culumnList = readDBTableRowIndexFile.list();

NextFile:

for(String culumn: culumnList) {

if(culumn.contains("is\_delete")) {

continue NextFile;

}

String DBTableCulumnIndexPath = DBTableRowIndexPath + "/" + culumn;

File readDBTableCulumnIndexPathFile

= new File(DBTableCulumnIndexPath);

if (readDBTableRowIndexCulumnFile.isDirectory()) {

reader = new BufferedReader(

new FileReader(readDBTableCulumnIndexPathFile + "/" + "value.lyg"));

temp="";

while ((tempString = reader.readLine()) != null) {

temp += tempString;

}

reader.close();

rowMap.put(culumn, temp);

}else {

rowMap.put(culumn, null);

}

}

output.add(rowMap);

}

}

}

}

}

}

}

}

}

return output;

}

@SuppressWarnings("static-access")

public static void main(String[] args) {

try {

new SelectRowsImp().selectRowsByAttribute("backend", "login"

, "usr\_name", "yaoguangluo");

// deletefile("D:/file");

} catch (FileNotFoundException ex) {

} catch (IOException ex) {

}

System.out.println("ok");

}

public static Map<String, Object> selectRowsByTablePath(String tablePath, String pageBegin

, String pageEnd, String direction) throws IOException {

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

int totalPages = 0;

output.put("tablePath", tablePath);

int rowBeginIndex = Integer.valueOf(pageBegin);

int rowEndIndex = Integer.valueOf(pageEnd);

String objectType = "";

List<Object> rowMapList = new ArrayList<>();

File fileDBTable = new File(tablePath);

if (fileDBTable.isDirectory()) {

String DBTableRowsPath = tablePath + "/rows";

File fileDBTableRowsPath = new File(DBTableRowsPath);

if (fileDBTableRowsPath.isDirectory()) {

File[] files = fileDBTableRowsPath.listFiles();

totalPages = files.length;

int i = 0;

int index = 0;

Here:

while(i<15) {

String DBTableRowIndexPath = DBTableRowsPath + "/row" + (direction.contains("next")? rowEndIndex++: --rowBeginIndex);

File readDBTableRowIndexFile = new File(DBTableRowIndexPath);

if (!readDBTableRowIndexFile.exists()) {

break;

}

File deleteTest = new File(DBTableRowIndexPath + "/is\_delete\_1");

if (deleteTest.exists()) {

continue Here;

}

i++;

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

String[] readDBTableRowCulumnsIndexFile = readDBTableRowIndexFile.list();

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

NextFile:

for(String readDBTableRowCulumnIndexFile: readDBTableRowCulumnsIndexFile) {

if(readDBTableRowCulumnIndexFile.contains("is\_delete")) {

continue NextFile;

}

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

String DBTableRowIndexCulumnPath = DBTableRowIndexPath + "/" + readDBTableRowCulumnIndexFile;

File readDBTableRowIndexCulumnFile = new File(DBTableRowIndexCulumnPath);

if (readDBTableRowIndexCulumnFile.exists()) {

String temp = "";

FileInputStream fis = new FileInputStream(new File(DBTableRowIndexCulumnPath + "/value.lyg"));

BufferedInputStream bis = new BufferedInputStream(fis);

byte[] buffer = new byte[1024];

int cnt;

while((cnt = bis.read(buffer)) != -1) {

temp += new String(buffer, 0, cnt);

}

fis.close();

bis.close();

culumnMap.put("culumnName", readDBTableRowCulumnIndexFile);

culumnMap.put("culumnValue", temp);

culumnMaps.put(readDBTableRowCulumnIndexFile, culumnMap);

}

}

rowMap.put("rowValue", culumnMaps);

if(direction.contains("next")) {

rowMap.put("rowIndex", rowEndIndex-1);

rowMapList.add(rowMap);

}else {

rowMap.put("rowIndex", rowBeginIndex);

rowMapList.add(0, rowMap);

}

}

}

}

if(direction.contains("next")) {

output.put("pageBegin", Integer.valueOf(pageEnd));

output.put("pageEnd", rowEndIndex);

output.put("totalPages", totalPages);

}else {

output.put("pageBegin", rowBeginIndex);

output.put("pageEnd", Integer.valueOf(pageBegin));

output.put("totalPages", totalPages);

}

output.put("obj", rowMapList);

List<Object> spec= new ArrayList<>();

Map<String, Object> row = (Map<String, Object>) rowMapList.get(0);

Map<String, Object> culumns = (Map<String, Object>) row.get("rowValue");

Iterator<String> it=culumns.keySet().iterator();

while(it.hasNext()) {

spec.add(((Map<String, Object>)culumns.get(it.next())).get("culumnName").toString());

}

output.put("spec", spec);

return output;

}

public static Object selectRowsByAttributesOfCondition(Map<String, Object> object) throws IOException {

if(!object.containsKey("recordRows")) {

Map<String, Boolean> recordRows = new ConcurrentHashMap<>();

object.put("recordRows", recordRows);

}

Spec spec = new Spec();

spec.setCulumnTypes(new ConcurrentHashMap<String, String>());

String objectType = "";

List<Map<String, Object>> output = new ArrayList<>();

//锁定数据库

String DBPath = CacheManager.getCacheInfo("DBPath").getValue().toString() + "/"

+ object.get("baseName").toString();

//锁定表

File fileDBPath = new File(DBPath);

if (fileDBPath.isDirectory()) {

String DBTablePath = DBPath + "/" + object.get("tableName").toString();

File fileDBTable = new File(DBTablePath);

if (fileDBTable.isDirectory()) {

String DBTableCulumnPath = DBTablePath + "/spec";

File fileDBTableCulumn = new File(DBTableCulumnPath);

if (fileDBTableCulumn.isDirectory()) {

//读取列数据格式

String[] fileList = fileDBTableCulumn.list();

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

File readDBTableSpecCulumnFile = new File(DBTableCulumnPath + "/"

+ fileList[0]+"/value.lyg");

BufferedReader reader = new BufferedReader(new FileReader(readDBTableSpecCulumnFile));

String tempString = null;

while ((tempString = reader.readLine()) != null) {

objectType = tempString;

}

reader.close();

spec.setCulumnType(fileList[i], objectType);

}

List<String[]> conditionValues = (List<String[]>) object.get("condition");

Iterator<String[]> iterator = conditionValues.iterator();

while(iterator.hasNext()) {

boolean overMap = output.size() == 0? false: true;

String[] conditionValueArray = iterator.next();

String type = conditionValueArray[1];

boolean andMap = type.equalsIgnoreCase("and")?true:false;

for(int i = 2; i < conditionValueArray.length; i++) {

String[] sets = conditionValueArray[i].split("\\|");

if(overMap && andMap) {

ProcessConditionPLSQL.processMap(sets, output, DBTablePath);//1

}else if(DetaDBBufferCacheManager.dbCache){

ProcessConditionPLSQL.processCache(sets, output, object.get("tableName").toString()

, object.get("baseName").toString(), object);//1

}else {

ProcessConditionPLSQL.processTable(sets, output, DBTablePath, object);//1

}

}

}

}

}

}

return output;

}

public static List<Map<String, Object>> selectRowsByAttributesOfAggregation(Map<String, Object> object) {

if(!object.containsKey("obj")) {

return new ArrayList<>();

}

List<Map<String, Object>> obj = ((List<Map<String, Object>>)(object.get("obj")));

List<String[]> aggregationValues = (List<String[]>) object.get("aggregation");

Iterator<String[]> iterator = aggregationValues.iterator();

while(iterator.hasNext()) {

boolean overMap = obj.size() == 0? false: true;

String[] aggregationValueArray = iterator.next();

String type = aggregationValueArray[1];

boolean limitMap = type.equalsIgnoreCase("limit")?true:false;

for(int i = 2; i < aggregationValueArray.length; i++) {

String[] sets = aggregationValueArray[i].split("\\|");

String DBPath = CacheManager.getCacheInfo("DBPath").getValue().toString()

+ "/" + object.get("baseName").toString();

String dBTablePath = DBPath + "/" + object.get("tableName").toString();

if(limitMap) {

ProcessAggregationPLSQL.processAggregationLimitMap(sets, obj);

}

//基于sort key 前序treeMap 之后排序功能设计

//基于sort key 后序treeMap

}

}

return obj;

}

public static Object selectRowsByAttributesOfGetCulumns(Map<String, Object> object) {

if(!object.containsKey("obj")) {

return new ArrayList<>();

}

List<Map<String, Object>> obj = ((List<Map<String, Object>>)(object.get("obj")));

List<String[]> getCulumnsValues = (List<String[]>) object.get("getCulumns");

Iterator<String[]> iterator = getCulumnsValues.iterator();

while(iterator.hasNext()) {

boolean overMap = obj.size() == 0? false: true;

String[] getCulumnsValueArray = iterator.next();

if(overMap) {

ProcessGetCulumnsPLSQL.processGetCulumnsMap(obj, getCulumnsValueArray);

}

}

return obj;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.lyg.db.update.imp;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileReader;

import java.io.IOException;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import java.util.concurrent.ConcurrentHashMap;

import cacheProcessor.CacheManager;

import org.lyg.cache.DetaDBBufferCacheManager;

import org.lyg.db.plsql.imp.ProcessAggregationPLSQL;

import org.lyg.db.plsql.imp.ProcessConditionPLSQL;

import org.lyg.db.plsql.imp.ProcessGetCulumnsPLSQL;

import org.lyg.db.plsql.imp.ProcessRelationPLSQL;

import org.lyg.db.reflection.Spec;

@SuppressWarnings({"unused", "unchecked"})

public class UpdateJoinRowsImp {

public static Object updateRowsByAttributesOfJoinCondition(Map<String, Object> object

, boolean mod) throws IOException {

if(!object.containsKey("recordRows")) {

Map<String, Boolean> recordRows = new ConcurrentHashMap<>();

object.put("recordRows", recordRows);

}

Spec spec = new Spec();

spec.setCulumnTypes(new ConcurrentHashMap<String, String>());

String objectType = "";

List<Map<String, Object>> output = new ArrayList<>();

//锁定数据库

String DBPath = CacheManager.getCacheInfo("DBPath").getValue().toString() + "/"

+ object.get("joinBaseName").toString();

//锁定表

File fileDBPath = new File(DBPath);

if (fileDBPath.isDirectory()) {

String DBTablePath = DBPath + "/" + object.get("joinTableName").toString();

File fileDBTable = new File(DBTablePath);

if (fileDBTable.isDirectory()) {

String DBTableCulumnPath = DBTablePath + "/spec";

File fileDBTableCulumn = new File(DBTableCulumnPath);

if (fileDBTableCulumn.isDirectory()) {

//读取列数据格式

String[] fileList = fileDBTableCulumn.list();

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

File readDBTableSpecCulumnFile = new File(DBTableCulumnPath + "/"

+ fileList[0] + "/value.lyg");

BufferedReader reader = new BufferedReader(new FileReader(readDBTableSpecCulumnFile));

String tempString = null;

while ((tempString = reader.readLine()) != null) {

objectType = tempString;

}

reader.close();

spec.setCulumnType(fileList[i], objectType);

}

List<String[]> conditionValues = (List<String[]>) object.get("condition");

Iterator<String[]> iterator = conditionValues.iterator();

while(iterator.hasNext()) {

boolean overMap = output.size() == 0? false: true;

String[] conditionValueArray = iterator.next();

String type = conditionValueArray[1];

boolean andMap = type.equalsIgnoreCase("and")?true:false;

for(int i = 2; i < conditionValueArray.length; i++) {

String[] sets = conditionValueArray[i].split("\\|");

if(overMap && andMap) {

ProcessConditionPLSQL.processMap(sets, output, DBTablePath);

}else if(DetaDBBufferCacheManager.dbCache){

ProcessConditionPLSQL.processCache(sets, output, object.get("joinTableName").toString()

, object.get("joinBaseName").toString(), object);

}else {

ProcessConditionPLSQL.processTable(sets, output, DBTablePath, object);

}

}

}

}

}

}

return output;

}

public static Object updateRowsByAttributesOfJoinAggregation(Map<String, Object> object, boolean mod) {

if(!object.containsKey("joinObj")) {

return new ArrayList<>();

}

List<Map<String, Object>> obj = ((List<Map<String, Object>>)(object.get("updateObj")));

List<String[]> aggregationValues = (List<String[]>) object.get("aggregation");

Iterator<String[]> iterator = aggregationValues.iterator();

while(iterator.hasNext()) {

boolean overMap = obj.size() == 0? false: true;

String[] aggregationValueArray = iterator.next();

String type = aggregationValueArray[1];

boolean limitMap = type.equalsIgnoreCase("limit")?true:false;

for(int i = 2; i < aggregationValueArray.length; i++) {

String[] sets = aggregationValueArray[i].split("\\|");

if(limitMap) {

ProcessAggregationPLSQL.processAggregationLimitMap(sets, obj);

}

//基于sort key 前序treeMap 之后排序功能设计

//基于sort key 后序treeMap

}

}

return obj;

}

public static Object updateRowsByAttributesOfJoinGetCulumns(Map<String, Object> object) {

if(!object.containsKey("joinObj")) {

return new ArrayList<>();

}

List<Map<String, Object>> obj = ((List<Map<String, Object>>)(object.get("updateJoinObj")));

List<String[]> getCulumnsValues = (List<String[]>) object.get("getCulumns");

Iterator<String[]> iterator = getCulumnsValues.iterator();

while(iterator.hasNext()) {

boolean overMap = obj.size() == 0? false: true;

String[] getCulumnsValueArray = iterator.next();

if(overMap) {

ProcessGetCulumnsPLSQL.processGetCulumnsMap(obj, getCulumnsValueArray);

}

}

return obj;

}

public static Object updateRowsByAttributesOfJoinRelation(Map<String, Object> object, boolean mod) {

if(!object.containsKey("updateObj")||!object.containsKey("updateJoinObj")) {

return new ArrayList<>();

}

Map<String,Boolean> findinNewObj = new HashMap<>();

List<Map<String, Object>> newObj = new ArrayList<Map<String, Object>>();

List<Map<String, Object>> obj = ((List<Map<String, Object>>)(object.get("updateObj")));

List<Map<String, Object>> joinObj= ((List<Map<String, Object>>)(object.get("updateJoinObj")));

List<String[]> relationValues = (List<String[]>) object.get("relation");

Iterator<String[]> iterator = relationValues.iterator();

while(iterator.hasNext()) {

boolean overObjMap= obj.size() == 0? false: true;

boolean overJoinObjMap= joinObj.size() == 0? false: true;

String[] getRelationValueArray = iterator.next();

String type = getRelationValueArray[1];

boolean andMap = type.equalsIgnoreCase("and")?true:false;

for(int i= 2; i< getRelationValueArray.length; i++) {

String[] sets = getRelationValueArray[i].split("\\|");

if(overObjMap&& overJoinObjMap&&andMap && i>2) {

ProcessRelationPLSQL.processAndMap(sets, obj, joinObj,object, newObj);//1

}else {

ProcessRelationPLSQL.processOrMap(sets, obj, joinObj, object, newObj, findinNewObj);

}

}

}

return newObj;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.lyg.db.update.imp;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import java.util.concurrent.ConcurrentHashMap;

import org.json.JSONObject;

import cacheProcessor.CacheManager;

import org.lyg.cache.DetaDBBufferCacheManager;

import org.lyg.db.plsql.imp.ProcessAggregationPLSQL;

import org.lyg.db.plsql.imp.ProcessConditionPLSQL;

import org.lyg.db.plsql.imp.ProcessGetCulumnsPLSQL;

import org.lyg.db.reflection.Cell;

import org.lyg.db.reflection.Row;

import org.lyg.db.reflection.Spec;

@SuppressWarnings("unchecked")

public class UpdateRowsImp {

public static Map<String, Object> updateRowByTablePathAndIndex(String tablePath, String pageIndex,

JSONObject jaculumnOfUpdateRow) throws FileNotFoundException, IOException {

String[] sets = tablePath.split("/");

int rowInsertIndex = Integer.valueOf(pageIndex);

File fileDBTable = new File(tablePath);

if (fileDBTable.isDirectory()) {

String DBTableRowsPath = tablePath + "/rows";

File fileDBTableRowsPath = new File(DBTableRowsPath);

if (fileDBTableRowsPath.isDirectory()) {

String DBTableRowIndexPath = DBTableRowsPath + "/row" + rowInsertIndex ;

File readDBTableRowIndexFile = new File(DBTableRowIndexPath);

if (readDBTableRowIndexFile.exists()) {

readDBTableRowIndexFile.mkdir();

Iterator<String> it = jaculumnOfUpdateRow.keys();

while(it.hasNext()) {

String culumnName = it.next();

String culumnValue = jaculumnOfUpdateRow.getString(culumnName);

String needCreatCulumnPath = DBTableRowIndexPath + "/" + culumnName;

File needCreatCulumn = new File(needCreatCulumnPath);

needCreatCulumn.delete();

needCreatCulumn.mkdir();

File needCreatCulumnFile = new File(needCreatCulumnPath + "/value.lyg");

needCreatCulumnFile.delete();

FileWriter fw = null;

try {

fw = new FileWriter(needCreatCulumnPath + "/value.lyg", true);

fw.write(null == culumnValue?"":culumnValue);

fw.close();

//fix buffer refresh

Cell cell = new Cell();

cell.setCellValue(null == culumnValue ? "" : culumnValue);

DetaDBBufferCacheManager.db.getBase(sets[sets.length - 2])

.getTable(sets[sets.length - 1]).getRow("row" + pageIndex).putCell(culumnName, cell);

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

}

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

output.put("totalPages", rowInsertIndex);

return output;

}

public static Map<String, Object> updateRowByTablePathAndAttribute(String tablePath

, String culumnName, String culumnValue,

JSONObject jobj) throws IOException {

String[] sets = tablePath.split("/");

File fileDBTable = new File(tablePath);

if (fileDBTable.isDirectory()) {

String DBTableRowsPath = tablePath + "/rows";

File fileDBTableRowsPath = new File(DBTableRowsPath);

if (fileDBTableRowsPath.isDirectory()) {

Here:

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

String DBTableRowIndexPath = DBTableRowsPath + "/row" + i;

File readDBTableRowIndexFile = new File(DBTableRowIndexPath);

if (readDBTableRowIndexFile.exists()) {

readDBTableRowIndexFile.mkdir();

File check = new File(DBTableRowIndexPath + "/" + culumnName + "/value.lyg");

if(!check.exists()) {

continue Here;

}

BufferedReader reader = new BufferedReader(new FileReader(check));

String temp = "";

String tempString;

while ((tempString = reader.readLine()) != null) {

temp += tempString;

}

reader.close();

if(!temp.equalsIgnoreCase(culumnValue)) {

continue Here;

}

Iterator<String> it = jobj.keys();

while(it.hasNext()) {

String culumnNameOfjs = it.next();

String culumnValueOfjs = jobj.get(culumnNameOfjs).toString();

String needCreatCulumnPath = DBTableRowIndexPath + "/" + culumnNameOfjs;

File needCreatCulumn = new File(needCreatCulumnPath);

needCreatCulumn.delete();

needCreatCulumn.mkdir();

File needCreatCulumnFile = new File(needCreatCulumnPath + "/value.lyg");

needCreatCulumnFile.delete();

FileWriter fw = null;

try {

fw = new FileWriter(needCreatCulumnPath + "/value.lyg", true);

fw.write(null == culumnValueOfjs? "" : culumnValueOfjs);

fw.close();

//fix buffer refresh

Cell cell = new Cell();

cell.setCellValue(null == culumnValueOfjs ? "" : culumnValueOfjs);

DetaDBBufferCacheManager.db.getBase(sets[sets.length - 2])

.getTable(sets[sets.length - 1]).getRow("row" + i).putCell(culumnNameOfjs, cell);

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

}

}

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

return output;

}

public static Object updateRowsByRecordConditions(Map<String, Object> object, boolean mod) throws IOException {

String DBPath = CacheManager.getCacheInfo("DBPath").getValue().toString() + "/"

+ object.get("baseName").toString();

String DBtablePath = DBPath + "/" + object.get("tableName").toString();

String DBTableRowsPath = DBtablePath + "/rows";

List<Map<String, Object>> updateObj = (List<Map<String, Object>>)object.get("updateObj");

Iterator<Map<String, Object>> updateObjIterator = updateObj.iterator();

List<String[]> culumnValues = (List<String[]>) object.get("culumnValue");

Iterator<String[]> culumnValuesIterator = culumnValues.iterator();

while(updateObjIterator.hasNext()) {

Map<String, Object> objRow = updateObjIterator.next();

Map<String, Object> objCells = (Map<String, Object>) objRow.get("rowValue");

Map<String, Object> onjCell = (Map<String, Object>) objCells.get("Index");

String rowIndex = "row";

rowIndex += onjCell.get("culumnValue").toString();

while(culumnValuesIterator.hasNext()) {

String[] culumns = culumnValuesIterator.next();

String filePath = DBTableRowsPath + "/" + rowIndex + "/" +culumns[1] + "/value.lyg";

File currentCellChange = new File(filePath);

if(currentCellChange.exists()) {

if(mod) {

currentCellChange.delete();

FileWriter fw = null;

try {

fw = new FileWriter(filePath, true);

fw.write(culumns[2]);

fw.close();

} catch (IOException e) {

fw.close();

e.printStackTrace();

}

}

Cell cell = new Cell();

cell.setCellValue(culumns[2]);

Row row = DetaDBBufferCacheManager.db.getBase(object.get("baseName").toString())

.getTable(object.get("tableName").toString()).getRow(rowIndex);

if(mod) {

row.putCell(culumns[1], cell);

}

}

}

}

return object;

}

public static Object updateRowsByAttributesOfCondition(Map<String, Object> object

, boolean mod) throws IOException {

if(!object.containsKey("recordRows")) {

Map<String, Boolean> recordRows = new ConcurrentHashMap<>();

object.put("recordRows", recordRows);

}

Spec spec = new Spec();

spec.setCulumnTypes(new ConcurrentHashMap<String, String>());

String objectType = "";

List<Map<String, Object>> output = new ArrayList<>();

//锁定数据库

String DBPath = CacheManager.getCacheInfo("DBPath").getValue().toString() + "/"

+ object.get("baseName").toString();

//锁定表

File fileDBPath = new File(DBPath);

if (fileDBPath.isDirectory()) {

String DBTablePath = DBPath + "/" + object.get("tableName").toString();

File fileDBTable = new File(DBTablePath);

if (fileDBTable.isDirectory()) {

String DBTableCulumnPath = DBTablePath + "/spec";

File fileDBTableCulumn = new File(DBTableCulumnPath);

if (fileDBTableCulumn.isDirectory()) {

//读取列数据格式

String[] fileList = fileDBTableCulumn.list();

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

File readDBTableSpecCulumnFile = new File(DBTableCulumnPath + "/"

+ fileList[0]+"/value.lyg");

BufferedReader reader = new BufferedReader(new FileReader(readDBTableSpecCulumnFile));

String tempString = null;

while ((tempString = reader.readLine()) != null) {

objectType = tempString;

}

reader.close();

spec.setCulumnType(fileList[i], objectType);

}

List<String[]> conditionValues = (List<String[]>) object.get("condition");

Iterator<String[]> iterator = conditionValues.iterator();

while(iterator.hasNext()) {

boolean overMap = output.size() == 0? false: true;

String[] conditionValueArray = iterator.next();

String type = conditionValueArray[1];

boolean andMap = type.equalsIgnoreCase("and")?true:false;

for(int i = 2; i < conditionValueArray.length; i++) {

String[] sets = conditionValueArray[i].split("\\|");

if(overMap && andMap) {

ProcessConditionPLSQL.processMap(sets, output, DBTablePath);

}else if(DetaDBBufferCacheManager.dbCache){

ProcessConditionPLSQL.processCache(sets, output

, object.get("tableName").toString()

, object.get("baseName").toString(), object);

}else {

ProcessConditionPLSQL.processTable(sets, output, DBTablePath, object);

}

}

}

}

}

}

return output;

}

public static List<Map<String, Object>> updateRowsByAttributesOfAggregation(Map<String, Object> object, boolean mod) {

if(!object.containsKey("obj")) {

return new ArrayList<>();

}

List<Map<String, Object>> obj = ((List<Map<String, Object>>)(object.get("updateObj")));

List<String[]> aggregationValues = (List<String[]>) object.get("aggregation");

Iterator<String[]> iterator = aggregationValues.iterator();

while(iterator.hasNext()) {

// boolean overMap = obj.size() == 0? false: true;

String[] aggregationValueArray = iterator.next();

String type = aggregationValueArray[1];

boolean limitMap = type.equalsIgnoreCase("limit")?true:false;

for(int i = 2; i < aggregationValueArray.length; i++) {

String[] sets = aggregationValueArray[i].split("\\|");

//String DBPath = CacheManager.getCacheInfo("DBPath").getValue().toString() + "/" + object.get("baseName").toString();

//String dBTablePath = DBPath + "/" + object.get("tableName").toString();

if(limitMap) {

ProcessAggregationPLSQL.processAggregationLimitMap(sets, obj);

}

//基于sort key 前序treeMap 之后排序功能设计

//基于sort key 后序treeMap

}

}

return obj;

}

public static Object updateRowsByAttributesOfGetCulumns(Map<String, Object> object) {

if(!object.containsKey("obj")) {

return new ArrayList<>();

}

List<Map<String, Object>> obj = ((List<Map<String, Object>>)(object.get("updateObj")));

List<String[]> getCulumnsValues = (List<String[]>) object.get("getCulumns");

Iterator<String[]> iterator = getCulumnsValues.iterator();

while(iterator.hasNext()) {

boolean overMap = obj.size() == 0? false: true;

String[] getCulumnsValueArray = iterator.next();

if(overMap) {

ProcessGetCulumnsPLSQL.processGetCulumnsMap(obj, getCulumnsValueArray);

}

}

return obj;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**package** org.lyg.stable;

**public** **interface** StableData {

//DB

**public** **static** **final** String ***DB\_BASE\_NAME*** = "baseName";

//LOGIN

**public** **static** **final** String ***LOGIN\_TOKEN*** = "token";

**public** **static** **final** String ***LOGIN\_EMAIL*** = "email";

**public** **static** **final** String ***LOGIN\_AUTH*** = "auth";

//STRING

**public** **static** **final** String ***STRING\_EMPTY*** = "";

**public** **static** **final** String ***STRING\_SPACE\_ENTER*** = " \n";

**public** **static** **final** String ***STRING\_ENTER*** = "\n";

**public** **static** **final** String ***STRING\_QUATE*** = ".";

**public** **static** **final** String ***STRING\_JUNCTION*** = "&";

//NUMBER

**public** **static** **final** **int** ***INT\_ZERO*** = 0;

**public** **static** **final** **int** ***INT\_MINES\_ONE*** = -1;

**public** **static** **final** **int** ***INT\_ONE*** = 1;

**public** **static** **final** **int** ***INT\_TWO*** = 2;

**public** **static** **final** **int** ***INT\_THREE*** = 3;

**public** **static** **final** **int** ***INT\_FOUR*** = 4;

**public** **static** **final** **int** ***INT\_FIVE*** = 5;

**public** **static** **final** **int** ***INT\_SIX*** = 6;

//BUFFER RANGE

**public** **static** **final** **int** ***BUFFER\_RANGE\_MAX*** = 1024;

//REST

**public** **static** **final** String ***REST\_GET\_DB\_CATEGORY*** = "/getDBCategory";

**public** **static** **final** String ***REST\_GET\_ALL\_DB\_CATEGORY*** = "/getAllDBCategory";

//SLEEPERS

**public** **static** **final** **int** ***SLEEPERS\_RANGE*** = 1500;

//TCP

**public** **static** **final** String ***TCP\_PORT*** = "port";

**public** **static** **final** String ***STRING\_SPACE*** = " ";

**public** **static** **final** String ***STRING\_SLASH\_QUESTION*** = "\\?";

//MATH

**public** **static** **final** String ***MATH\_EQUAL*** = "=";

//HTTP

**public** **static** **final** **int** ***HTTP\_500*** = 500;

**public** **static** **final** **int** ***HTTP\_400*** = 400;

**public** **static** **final** **int** ***HTTP\_200*** = 200;

**public** **static** **final** **int** ***HTTP\_404*** = 404;

**public** **static** **final** **int** ***HTTP\_300*** = 300;

//CHARSET

**public** **static** **final** String ***CHARSET\_UTF\_8*** = "UTF-8";

**public** **static** **final** String ***CHARSET\_UTF8*** = "UTF8";

**public** **static** **final** String ***CHARSET\_GBK*** = "GBK";

//FILE FORMAT

**public** **static** **final** String ***FILE\_EOT*** = ".eot";

**public** **static** **final** String ***FILE\_SVG*** = ".svg";

**public** **static** **final** String ***FILE\_OTF*** = ".otf";

**public** **static** **final** String ***FILE\_WOFF*** = ".woff";

**public** **static** **final** String ***FILE\_WOFF2*** = ".woff2";

**public** **static** **final** String ***FILE\_TTF*** = ".ttf";

**public** **static** **final** String ***FILE\_PNG*** = ".png";

**public** **static** **final** String ***FILE\_JPG*** = ".jpg";

**public** **static** **final** String ***FILE\_JPEG*** = ".jpeg";

**public** **static** **final** String ***FILE\_WAV*** = ".wav";

**public** **static** **final** String ***FILE\_GIF*** = ".gif";

**public** **static** **final** String ***FILE\_JS*** = ".js";

**public** **static** **final** String ***FILE\_CSS*** = ".css";

**public** **static** **final** String ***FILE\_HTML*** = ".html";

//FILE Stream

**public** **static** **final** String ***STREAM\_BUFFER*** = "buffer";

**public** **static** **final** String ***STREAM\_BYTES*** = "bytes";

**public** **static** **final** String ***STREAM\_BYTES\_BUFFER*** = "bytesBuffer";

**public** **static** **final** String ***STREAM\_REST*** = "rest";

//HTTP HEADER

**public** **static** **final** String ***HEADER\_CONTENT\_TYPE\_PNG*** = "Content-Type: image/png \n\n";

**public** **static** **final** String ***HEADER\_CONTENT\_TYPE\_JPEG*** = "Content-Type: image/jpeg \n\n";

**public** **static** **final** String ***HEADER\_CONTENT\_TYPE\_JPG*** = "Content-Type: image/jpg \n\n";

**public** **static** **final** String ***HEADER\_CONTENT\_TYPE\_GIF*** = "Content-Type: image/gif \n\n";

**public** **static** **final** String ***HEADER\_CONTENT\_TYPE\_CSS*** = "Content-Type: text/css \n\n";

**public** **static** **final** String ***HEADER\_CONTENT\_TYPE\_HTML*** = "Content-Type: text/html \n\n";

**public** **static** **final** String ***HEADER\_CONTENT\_TYPE\_WAV*** = "Content-Type: audio/wav \n\n";

**public** **static** **final** String ***HEADER\_CONTENT\_TYPE\_FONT\_WOFF*** = "Content-Type: image/font-woff \n\n";

**public** **static** **final** String ***HEADER\_CONTENT\_TYPE\_JS*** = "content-type: text/javascript; charset:UTF-8 \n\n";

**public** **static** **final** String ***HEADER\_CACHE\_CONTROL*** = "Cache-control: max-age=315360000 \n";

**public** **static** **final** String ***HEADER\_HTTP\_200\_OK*** = "http/1.1 200 ok \n";

**public** **static** **final** String ***HEADER\_HOST*** = "Host:deta software \n";

**public** **static** **final** String ***HEADER\_CONTENT\_ENCODING\_GZIP*** = "Content-Encoding:gzip \n";

**public** **static** **final** String ***HEADER\_ACCEPT\_RANGES\_BYTES*** = "Accept-Ranges: bytes \n";

**public** **static** **final** String ***HEADER\_CONTENT\_LENGTH*** = "Content-Length: ";

//REST PATH

**public** **static** **final** String ***REST\_PATH\_SELECT*** = "/select";

**public** **static** **final** String ***REST\_PATH\_SETDB*** = "/setDB";

**public** **static** **final** String ***REST\_PATH\_INSERT*** = "/insert";

**public** **static** **final** String ***REST\_PATH\_DELETE*** = "/delete";

**public** **static** **final** String ***REST\_PATH\_UPDATE*** = "/update";

**public** **static** **final** String ***REST\_PATH\_DB\_CATEGORY*** = "DBCategory";

**public** **static** **final** String ***REST\_PATH\_EXEC\_DETA\_PLSQL*** = "/execDetaPLSQL";

**public** **static** **final** String ***REST\_PATH\_LOGIN*** = "/login";

**public** **static** **final** String ***REST\_PATH\_FIND*** = "/find";

**public** **static** **final** String ***REST\_PATH\_LOGOUT*** = "/logout";

**public** **static** **final** String ***REST\_PATH\_REGISTER*** = "/register";

**public** **static** **final** String ***REST\_PATH\_CHANGE*** = "/change";

**public** **static** **final** String ***REST\_PATH\_CHECK\_STATUS*** = "/checkStatus";

**public** **static** **final** String ***REST\_PATH\_SET\_DB\_PATH*** = "/setDBPath";

**public** **static** **final** String ***REST\_PATH\_SET\_DB\_TABLE*** = "/setDBTable";

**public** **static** **final** String ***REST\_PATH\_DELETE\_ROWS\_BY\_TABLE\_PATH\_AND\_INDEX*** = "/deleteRowByTablePathAndIndex";

**public** **static** **final** String ***REST\_PATH\_INSERT\_ROW\_BY\_BASE\_NAME*** = "/insertRowByBaseName";

**public** **static** **final** String ***REST\_PATH\_INSERT\_ROW\_BY\_TABLE\_PATH*** = "/insertRowByTablePath";

**public** **static** **final** String ***REST\_PATH\_SELECT\_ROWS\_BY\_ATTRIBUTE*** = "/selectRowsByAttribute";

**public** **static** **final** String ***REST\_PATH\_SELECT\_ROWS\_BY\_TABLE\_PATH*** = "/selectRowsByTablePath";

**public** **static** **final** String ***REST\_PATH\_UPDATE\_ROW\_BY\_TABLE\_PATH\_AND\_INDEX*** = "/updateRowByTablePathAndIndex";

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.lyg.vpc.process.companyImpl;

import java.io.IOException;

import java.util.Date;

import java.util.Map;

import org.json.JSONObject;

import org.lyg.common.utils.StringUtil;

import org.lyg.common.utils.TokenUtil;

import org.lyg.vpc.process.factoryImpl.LoginDAOImpl;

import MD5Processor.Usr;

import MD5Processor.UsrToken;

public class LoginServiceImpl {

// @Autowired

// private LoginDAO loginDAO;

public static Usr findUsrByUEmail(String uEmail) throws IOException {

Usr usr = LoginDAOImpl.selectUsrByUEmail(uEmail);

return usr;

}

public static UsrToken findUsrTokenByUId(Integer uId) throws IOException {

UsrToken usrToken = LoginDAOImpl.selectUsrTokenByUId(uId);

return usrToken;

}

public static void updateUsrTokenByUId(Integer uId, String key

, String uPassword, long uTime) throws IOException {

LoginDAOImpl.updateUsrTokenByUId(uId, key, uPassword, uTime);

}

public static void insertRowByTablePath(String baseName, String tableName

, JSONObject jsobj) throws Exception {

LoginDAOImpl.insertRowByTablePath(baseName, tableName, jsobj);

}

public static String checkTokenStatus(String token, String level) throws Exception {

if (null == token) {

return "invalid 秘钥丢失请重新登陆。";

}

String json = StringUtil.decode(token);

JSONObject js;

try {

js = new JSONObject(json);

}catch(Exception e) {

return "invalid 秘钥错误请重新登陆。";

}

long uTime = js.getLong("uTime");

String uPassword = js.getString("mPassword");

String uEmail = js.getString("uEmail");

Usr usr = findUsrByUEmail(uEmail);

UsrToken usrToken = findUsrTokenByUId(usr.getuId());

String password = TokenUtil.getFirstMD5Password(js.getString("uKey"), usrToken.getuPassword());

if (!uPassword.equals(password)) {

return "invalid 密码错误。";

}

long nowTime = new Date().getTime();

if(uTime + 600000 < nowTime) {

return "invalid 10分钟超时，请重新登陆。";

}

if(level.contains("level")) {

String uLevel = usrToken.getuLevel();

if(!uLevel.contains("high")) {

return "invalid 权限不够";

}

}

return "valid";

}

public static String checkRightsStatus(String inEmail, String inPassword

, String level) throws Exception {

if (null == inEmail) {

return "invalid 秘钥丢失请重新登陆。";

}

//String uPassword = inPassword;

String uEmail = inEmail;

Usr usr = findUsrByUEmail(uEmail);

//UsrToken usrToken = this.findUsrTokenByUId(usr.getuId());

String password = TokenUtil.getSecondMD5Password(inPassword);

if (!usr.getuPassword().equals(password)) {

return "invalid 密码错误。";

}

return "valid";

}

public static String checkTokenStatusAndGetLevel(String token, String level

, Map<String, Object> output) throws Exception {

if (null == token || token.equalsIgnoreCase("undefined")) {

return "invalid 秘钥丢失请重新登陆。";

}

String json = StringUtil.decode(token);

JSONObject js;

try {

js = new JSONObject(json);

}catch(Exception e) {

return "invalid 秘钥错误请重新登陆。";

}

long uTime = js.getLong("uTime");

String uPassword = js.getString("mPassword");

String uEmail = js.getString("uEmail");

Usr usr = findUsrByUEmail(uEmail);

UsrToken usrToken = findUsrTokenByUId(usr.getuId());

String password = TokenUtil.getFirstMD5Password(js.getString("uKey")

, usrToken.getuPassword());

if (!uPassword.equals(password)) {

return "invalid 密码错误。";

}

long nowTime = new Date().getTime();

if(uTime + 600000 < nowTime) {

return "invalid 10分钟超时，请重新登陆。";

}

if(level.contains("level")) {

String uLevel = usrToken.getuLevel();

if(!uLevel.contains("high")) {

return "invalid 权限不够";

}

}

output.put("usrName", "咨询专家" + usr.getuId());

return "valid" + usrToken.getuLevel();

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.lyg.vpc.process.factoryImpl;

import java.io.IOException;

import java.util.List;

import java.util.Map;

import org.json.JSONObject;

import cacheProcessor.CacheManager;

import org.lyg.db.insert.imp.InsertRowsImp;

import org.lyg.db.select.imp.SelectRowsImp;

import org.lyg.db.update.imp.UpdateRowsImp;

import org.lyg.vpc.view.UsrFull;

import MD5Processor.Usr;

import MD5Processor.UsrToken;

public class LoginDAOImpl {

// @Autowired

// private SelectRows selectRows;

//

// @Autowired

// private UpdateRows updateRows;

//

// @Autowired

// private InsertRows insertRows;

public static Usr selectUsrByUId(Integer uId) throws IOException {

List<Map<String, Object>> list = SelectRowsImp.selectRowsByAttribute("backend", "usr", "u\_id", "" + uId);

Usr usr = new Usr();

if(list.size() > 0) {

usr.setuAddress(list.get(0).get("u\_address")

!=null?list.get(0).get("u\_address").toString():"");

usr.setuAge(Integer.valueOf(list.get(0).get("u\_age")

!=null?list.get(0).get("u\_age").toString():"0"));

usr.setuClass(list.get(0).get("u\_class")!=null?list.get(0).get("u\_class").toString():"");

usr.setuEmail(list.get(0).get("u\_email").toString());

usr.setuId(Integer.valueOf(list.get(0).get("u\_id").toString()));

usr.setuName(list.get(0).get("u\_name").toString());

usr.setuPhone(list.get(0).get("u\_phone")!=null?list.get(0).get("u\_phone").toString():"");

usr.setuQq(list.get(0).get("u\_qq")!=null?list.get(0).get("u\_qq").toString():"");

usr.setuSex(list.get(0).get("u\_sex")!=null?list.get(0).get("u\_sex").toString():"");

usr.setuWeChat(list.get(0).get("u\_weChat")!=null?list.get(0).get("u\_weChat").toString():"");

}

return usr;

}

public static UsrToken selectUsrTokenByUId(Integer uId) throws IOException {

List<Map<String, Object>> list

= SelectRowsImp.selectRowsByAttribute("backend", "usrToken", "u\_id", ""+uId);

UsrToken usrToken = new UsrToken();

if(list.size() > 0) {

usrToken.setuId(Integer.valueOf(list.get(0).get("u\_id").toString()));

usrToken.setuKey(list.get(0).get("u\_key")

!= null?list.get(0).get("u\_key").toString():"");

usrToken.setuPassword(list.get(0).get("u\_password").toString());

usrToken.setuTime(Integer.valueOf(list.get(0).get("u\_time")

!= null?list.get(0).get("u\_time").toString():"0"));

usrToken.setuLevel(list.get(0).get("u\_level")

!= null?list.get(0).get("u\_level").toString():"");

}

return usrToken;

}

public static UsrFull selectUsrFullByUId(Integer uId) {

// TODO Auto-generated method stub

return null;

}

public static Usr selectUsrByUEmail(String uEmail) throws IOException {

List<Map<String, Object>> list = SelectRowsImp.selectRowsByAttribute("backend", "usr", "u\_email", uEmail);

Usr usr = new Usr();

if(list.size() > 0) {

usr.setuAddress(list.get(0).get("u\_address")

!= null?list.get(0).get("u\_address").toString():"");

usr.setuAge(Integer.valueOf(list.get(0).get("u\_age")

!= null?list.get(0).get("u\_age").toString():"0"));

usr.setuClass(list.get(0).get("u\_class")

!= null?list.get(0).get("u\_class").toString():"");

usr.setuEmail(list.get(0).get("u\_email").toString());

usr.setuId(Integer.valueOf(list.get(0).get("u\_id").toString()));

usr.setuName(list.get(0).get("u\_name").toString());

usr.setuPhone(list.get(0).get("u\_phone")

!= null?list.get(0).get("u\_phone").toString():"");

usr.setuQq(list.get(0).get("u\_qq")

!= null?list.get(0).get("u\_qq").toString():"");

usr.setuSex(list.get(0).get("u\_sex")

!= null?list.get(0).get("u\_sex").toString():"");

usr.setuWeChat(list.get(0).get("u\_weChat")

!= null?list.get(0).get("u\_weChat").toString():"");

usr.setuPassword(list.get(0).get("u\_password")

!= null?list.get(0).get("u\_password").toString():"");

}

return usr;

}

public static void updateUsrByUId(Integer uId, String uName, String uAge

, String uSex, String uPhone, String uAddress,

String uWeChat, String uClass, String uEmail, String uQq) throws IOException {

JSONObject jobj = new JSONObject();

jobj.put("u\_id", uId);

jobj.put("u\_name", uName);

jobj.put("u\_age", uAge);

jobj.put("u\_sex", uSex);

jobj.put("u\_phone", uPhone);

jobj.put("u\_address", uAddress);

jobj.put("u\_weChat", uWeChat);

jobj.put("u\_class", uClass);

jobj.put("u\_email", uEmail);

jobj.put("u\_qq", uQq);

UpdateRowsImp.updateRowByTablePathAndAttribute(CacheManager.getCacheInfo("DBPath")

.getValue()+"/backend/usr"

, "u\_id", "" + uId, jobj);

// TODO Auto-generated method stub

}

public static void updateUsrTokenByUId(Integer uId, String uKey, String uPassword

, long uTime) throws IOException {

JSONObject jobj = new JSONObject();

jobj.put("u\_id", uId);

jobj.put("u\_key", uKey);

jobj.put("u\_password", uPassword);

jobj.put("u\_time", uTime);

UpdateRowsImp.updateRowByTablePathAndAttribute(CacheManager.getCacheInfo("DBPath")

.getValue()+"/backend/usrToken"

, "u\_id", "" + uId, jobj);

// TODO Auto-generated method stub

}

public static void insertUsr(String uName, String uAge, String uSex, String uPhone

, String uAddress, String uWeChat,

String uClass, String uEmail, String uQq) {

// TODO Auto-generated method stub

}

public static void insertUsroken(Integer uId, String uKey, String uPassword, long uTime) {

// TODO Auto-generated method stub

}

public static void insertRowByTablePath(String baseName, String tableName

, JSONObject jsobj) throws Exception {

InsertRowsImp.insertRowByBaseName(baseName, tableName, jsobj, true);

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.lyg.vpc.process.portImpl;

import org.json.JSONException;

import cacheProcessor.CacheManager;

import org.lyg.vpc.process.companyImpl.LoginServiceImpl;

import java.io.BufferedReader;

import java.io.File;

import java.io.IOException;

import java.io.InputStreamReader;

import java.net.HttpURLConnection;

import java.net.MalformedURLException;

import java.net.URL;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

public class RestControllerPortImpl {

public static Map<String, Object> startResults(int aa, String token, String auth)

throws NumberFormatException, JSONException, Exception {

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

String checkStatus = LoginServiceImpl.checkTokenStatus(token, "common");

if(checkStatus.contains("invalid")&&(auth.contains("1"))) {

result.put("loginInfo", "unsuccess");

result.put("returnResult", checkStatus);

return result;

}

result.put("end", aa);

// System.out.println("4444" + result.get("end"));

// String AA = helloBean.saySomething(aa);

// result.put("end", AA);

return result;

//return Response.status(Status.OK).entity(result).build();

/\*

Session session = HibernateUtil.getSessionFactory().openSession();

session.beginTransaction();

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

String hql = "FROM Lolroler as l where l.name= :userName";

Query query = session.createQuery(hql);

result.put("end", results);

query.setString("userName", "Vi");

List<Lolroler> results = query.list();

session.getTransaction().commit();

return Response.status(Status.OK).entity(result).build();

\*/

/\*

if(null == redisTemplate.opsForValue().get("click")){

redisTemplate.opsForValue().set("click", "0" , 24000 , TimeUnit.HOURS);

}else{

String click = redisTemplate.opsForValue().get("click");

long click\_long = Long.parseLong(click);

click\_long += 1;

redisTemplate.opsForValue().set("click", String.valueOf(click\_long) , 24000 , TimeUnit.HOURS);

}

System.out.println(1111);

EventDispatcher dispatcher = new EventDispatcher();

dispatcher.registerHandler(UserCreatedEvent.class, new UserCreatedEventHandler());

System.out.println(2222);

User user = new User("iluwatar");

dispatcher.dispatch(new UserCreatedEvent(user));

System.out.println(3333);

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

result.put("end", user.getUsername());

System.out.println("4444"+result.get("end"));

return Response.status(Status.OK).entity(result).build();

\*/

}

public static Map<String, Object> startResultsBb(int bb, String token, String auth)

throws NumberFormatException, JSONException, Exception {

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

String checkStatus = LoginServiceImpl.checkTokenStatus(token, "common");

if(checkStatus.contains("invalid")&&(auth.contains("1"))) {

output.put("loginInfo", "unsuccess");

output.put("returnResult", checkStatus);

return output;

}

try {

URL url = new URL("http://localhost:3306/aa?aa=1");

System.out.println("http://localhost:3306/aa?aa=1");

HttpURLConnection conn = (HttpURLConnection) url.openConnection();

conn.setRequestMethod("GET");

conn.setRequestProperty("Accept", "application/json");

if (conn.getResponseCode() != 200) {

throw new RuntimeException("Failed : HTTP error code : "

+ conn.getResponseCode());

}

BufferedReader br = new BufferedReader(new InputStreamReader(

(conn.getInputStream())));

String out = "";

String out1;

System.out.println("Output from Server .... \n");

while ((out1 = br.readLine()) != null) {

out += out1;

}

output.put("result", out);

conn.disconnect();

} catch (MalformedURLException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

return output;

}

public static Map<String, Object> getDBCategory(String baseName, String token, String auth) throws Exception {

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

if(token != null && !token.equalsIgnoreCase("")){

String checkStatus = LoginServiceImpl.checkTokenStatus(token, "common");

if(checkStatus.contains("invalid")&&((auth==null?"1":auth).contains("1"))) {

output.put("loginInfo", "unsuccess");

output.put("returnResult", checkStatus);

return output;

}

}else{

output.put("loginInfo", "unsuccess");

output.put("returnResult", "invalid request");

return output;

}

String DBPath = CacheManager.getCacheInfo("DBPath").getValue().toString() + "/" + baseName;

//锁定表

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

File fileDBPath = new File(DBPath);

if (fileDBPath.isDirectory()) {

String[] files = fileDBPath.list();

for(String file:files) {

table.put(file, DBPath + "/" + file);

}

}

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

bases.put(baseName, table);

output.put("obj", bases);

return output;

}

public static Map<String, Object> getAllDBCategory(String token, String auth) throws Exception {

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

if(token != null && !token.equalsIgnoreCase("")){

String checkStatus = LoginServiceImpl.checkTokenStatus(token, "common");

if(checkStatus.contains("invalid")&&((auth==null?"1":auth).contains("1"))) {

output.put("loginInfo", "unsuccess");

output.put("returnResult", checkStatus);

return output;

}

}else{

output.put("loginInfo", "unsuccess");

output.put("returnResult", "invalid request");

return output;

}

String DBPath = CacheManager.getCacheInfo("DBPath").getValue().toString();

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

List<Object> baseList = new ArrayList<>();

File fileDBPath = new File(DBPath);

if (fileDBPath.isDirectory()) {

String[] files = fileDBPath.list();

for(String file:files) {

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

String DBBasePath = DBPath + "/" + file;

File fileDBBasePath = new File(DBBasePath);

if (fileDBBasePath.isDirectory()) {

List<Object> tableList = new ArrayList<>();

String[] filesInfileDBBasePath = fileDBBasePath.list();

for(String fileInfileDBBasePath: filesInfileDBBasePath) {

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

String DBTablePath = DBBasePath + "/" + fileInfileDBBasePath;

table.put("tableName", fileInfileDBBasePath);

table.put("tablePath", DBTablePath);

tableList.add(table);

}

base.put("tableList", tableList);

}

base.put("baseName", file);

base.put("basePath", DBBasePath);

baseList.add(base);

}

db.put("baseList", baseList);

}

db.put("dbName", "deta");

db.put("dbPath", DBPath);

//锁定表

return db;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.lyg.vpc.process.portImpl;

import cacheProcessor.Cache;

import cacheProcessor.CacheManager;

import org.lyg.vpc.process.companyImpl.LoginServiceImpl;

import java.io.File;

import java.io.FileWriter;

import java.util.HashMap;

import java.util.Map;

public class RestDBConfigImpl {

public static Map<String, String> setDBPath(String basePath

, String token, String auth) throws Exception {

Map<String, String> output= new HashMap<String, String>();

String checkStatus = LoginServiceImpl.checkTokenStatus(token, "level");

if(checkStatus.contains("invalid")&&(auth.contains("1"))) {

output.put("loginInfo", "unsuccess");

output.put("returnResult", checkStatus);

return output;

}

//检查配置文件

File config = new File("C:/DBconfig.lyg");

if (config.exists()) {

config.delete();

}

//重写配置文件

FileWriter fw = null;

fw = new FileWriter("C:/DBconfig.lyg", true);

fw.write("path->" + basePath);

fw.close();

//写缓存

Cache c = new Cache();

c.setValue(basePath);

CacheManager.putCache("DBPath", c);

//锁定表

File fileDBPath = new File(basePath);

if (fileDBPath.isDirectory()) {

output.put("info", "isDirectory" + CacheManager.getCacheInfo("DBPath")

.getValue().toString());

}else {

fileDBPath.mkdir();

output.put("info", "isCreated" + CacheManager.getCacheInfo("DBPath")

.getValue().toString());

}

return output;

}

public static Map<String, String> setDBTable(String tableName, String token

, String auth) throws Exception {

Map<String, String> output = new HashMap<String, String>();

String checkStatus = LoginServiceImpl.checkTokenStatus(token, "level");

if(checkStatus.contains("invalid")&&(auth.contains("1"))) {

output.put("loginInfo", "unsuccess");

output.put("returnResult", checkStatus);

return output;

}

//是否有

String DBPath = CacheManager.getCacheInfo("DBPath").getValue().toString() +"/"+ tableName;

//锁定表

File fileDBPath = new File(DBPath);

if (fileDBPath.isDirectory()) {

output.put("info", "isDirectory:"+DBPath);

}else {

fileDBPath.mkdir();

output.put("info", "isCreated:"+DBPath);

}

//有就输出

//没有就创建

return output;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.lyg.vpc.process.portImpl;

import java.util.HashMap;

import java.util.Map;

import org.lyg.db.delete.imp.DeleteRowsImp;

import org.lyg.vpc.process.companyImpl.LoginServiceImpl;

public class RestDBDeleteImpl {

public static Map<String, Object> deleteRowByTablePathAndIndex(String tablePath

, String pageIndex, String token

, String email, String password, String auth) throws Exception {

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

if(token != null && !token.equalsIgnoreCase("")){

String checkStatus = LoginServiceImpl.checkTokenStatus(token, "level");

if(checkStatus.contains("invalid")&&(auth.contains("1"))) {

output.put("loginInfo", "unsuccess");

output.put("returnResult", checkStatus);

return output;

}

}else if(email != null && !email.equalsIgnoreCase("")){

String checkStatus = LoginServiceImpl.checkRightsStatus(email, password, "DB");

if(checkStatus.contains("invalid")) {

output.put("loginInfo", "unsuccess");

output.put("returnResult", checkStatus);

return output;

}

}else{

output.put("loginInfo", "unsuccess");

output.put("returnResult", "invalid request");

return output;

}

return DeleteRowsImp.deleteRowByTablePathAndIndex(tablePath, pageIndex, true);

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.lyg.vpc.process.portImpl;

import org.json.JSONObject;

import org.lyg.db.insert.imp.InsertRowsImp;

import org.lyg.vpc.process.companyImpl.LoginServiceImpl;

import java.util.HashMap;

import java.util.Map;

public class RestDBInsertImpl {

public static Map<String, Object> insertRowByTablePath(String tablePath, String pageIndex

, String culumnOfNewRow, String token, String email, String password

, String auth) throws Exception {

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

if(token != null && !token.equalsIgnoreCase("")){

String checkStatus = LoginServiceImpl.checkTokenStatus(token, "common");

if(checkStatus.contains("invalid")&&(auth.contains("1"))) {

output.put("loginInfo", "unsuccess");

output.put("returnResult", checkStatus);

return output;

}

}else if(email != null && !email.equalsIgnoreCase("")){

String checkStatus = LoginServiceImpl.checkRightsStatus(email, password, "DB");

if(checkStatus.contains("invalid")) {

output.put("loginInfo", "unsuccess");

output.put("returnResult", checkStatus);

return output;

}

}else{

output.put("loginInfo", "unsuccess");

output.put("returnResult", "invalid request");

return output;

}

JSONObject jaculumnOfNewRow=new JSONObject(culumnOfNewRow);

output = InsertRowsImp.insertRowByTablePathAndIndex(tablePath, pageIndex, jaculumnOfNewRow);

return output;

}

public static Map<String, Object> insertRowByBaseName(String baseName, String tableName

, String culumnOfNewRow, String token, String email, String password

, String auth) throws Exception {

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

String checkStatus = LoginServiceImpl.checkRightsStatus(email, password, "DB");

if(checkStatus.contains("invalid")) {

output.put("loginInfo", "unsuccess");

output.put("returnResult", checkStatus);

return output;

}

checkStatus = LoginServiceImpl.checkTokenStatus(token, "common");

if(checkStatus.contains("invalid")&&(auth.contains("1"))) {

output.put("loginInfo", "unsuccess");

output.put("returnResult", checkStatus);

return output;

}

JSONObject jaculumnOfNewRow=new JSONObject(culumnOfNewRow);

output = InsertRowsImp.insertRowByBaseName(baseName, tableName, jaculumnOfNewRow, true);

return output;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.lyg.vpc.process.portImpl;

import java.util.HashMap;

import java.util.Map;

import org.deta.vpcs.hall.DatabaseLogHall;

import org.json.JSONObject;

import org.lyg.common.utils.StringUtil;

import org.lyg.db.plsql.imp.ExecPLSQLImp;

import org.lyg.vpc.process.companyImpl.LoginServiceImpl;

//

//baseName:backend;

//tableName:usr:update;

//condition:or:u\_id|<|200;

//culumnValue:u\_email:77777;

public class RestDBPLSQLImpl {

public static Map<String, Object> restDBPLSQLImpl(String token,

String email, String password, String auth, String plsql

, String mod) throws Exception{

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

String who = "";

//security monitor

if(token != null && !token.equalsIgnoreCase("")){

String checkStatus = LoginServiceImpl.checkTokenStatus(token, "common");

if(checkStatus.contains("invalid")&&(auth.contains("1"))) {

output.put("loginInfo", "unsuccess");

output.put("returnResult", checkStatus);

return output;

}

String json = StringUtil.decode(token);

JSONObject js = new JSONObject(json);

who = js.getString("uEmail");

}else if(email != null && !email.equalsIgnoreCase("")){

String checkStatus = LoginServiceImpl.checkRightsStatus(email, password, "DB");

if(checkStatus.contains("invalid")) {

output.put("loginInfo", "unsuccess");

output.put("returnResult", checkStatus);

return output;

}

who = email;

}else{

output.put("loginInfo", "unsuccess");

output.put("returnResult", "invalid request");

return output;

}

//write monior

if(plsql.contains("update")||plsql.contains("insert")||plsql.contains("delete")

||plsql.contains("drop")||plsql.contains("change")||plsql.contains("create")) {

DatabaseLogHall.writeLogFile(System.currentTimeMillis(), who, plsql);

try {

ExecPLSQLImp.ExecPLSQL(plsql, false);

}catch(Exception e) {

output.put("loginInfo", "unsuccess");

output.put("returnResult", "invalid plsql");

return output;

}

if(null != mod && mod.equalsIgnoreCase("true")) {

output = ExecPLSQLImp.ExecPLSQL(plsql, true);

}

}else {

output = ExecPLSQLImp.ExecPLSQL(plsql, true);

}

return output;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.lyg.vpc.process.portImpl;

import org.lyg.db.select.imp.SelectRowsImp;

import org.lyg.vpc.process.companyImpl.LoginServiceImpl;

import java.util.HashMap;

import java.util.Map;

public class RestDBSelectImpl {

public static Map<String, Object> selectRowsByAttribute(String baseName,

String tableName, String culumnName, String value, String token,

String email, String password, String auth) throws Exception{

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

if(token != null && !token.equalsIgnoreCase("")){

String checkStatus = LoginServiceImpl.checkTokenStatus(token, "common");

if(checkStatus.contains("invalid")&&(auth.contains("1"))) {

output.put("loginInfo", "unsuccess");

output.put("returnResult", checkStatus);

return output;

}

}else if(email != null && !email.equalsIgnoreCase("")){

String checkStatus = LoginServiceImpl.checkRightsStatus(email, password, "DB");

if(checkStatus.contains("invalid")) {

output.put("loginInfo", "unsuccess");

output.put("returnResult", checkStatus);

return output;

}

}else{

output.put("loginInfo", "unsuccess");

output.put("returnResult", "invalid request");

return output;

}

output.put("obj", SelectRowsImp.selectRowsByAttribute(baseName, tableName, culumnName, value));

return output;

}

public static Map<String, Object> selectRowsByTablePath(String tablePath, String pageBegin

, String pageEnd, String direction

, String token, String email, String password, String auth) throws Exception {

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

if(token != null && !token.equalsIgnoreCase("")){

String checkStatus = LoginServiceImpl.checkTokenStatus(token, "common");

if(checkStatus.contains("invalid")&&(auth.contains("1"))) {

output.put("loginInfo", "unsuccess");

output.put("returnResult", checkStatus);

return output;

}

}else if(email != null && !email.equalsIgnoreCase("")){

String checkStatus = LoginServiceImpl.checkRightsStatus(email, password, "DB");

if(checkStatus.contains("invalid")) {

output.put("loginInfo", "unsuccess");

output.put("returnResult", checkStatus);

return output;

}

}else{

output.put("loginInfo", "unsuccess");

output.put("returnResult", "invalid request");

return output;

}

// if(CacheManager.getCacheInfo(tablePath + ":" + pageBegin + ":" + pageEnd

//+ ":" + direction) != null) {

// output = (Map<String, Object>)(CacheManager.getCacheInfo(tablePath

//+ ":" + pageBegin + ":" + pageEnd + ":" + direction).getValue());

// return output;

// }

output = SelectRowsImp.selectRowsByTablePath(tablePath, pageBegin, pageEnd, direction);

// if(tablePath.equalsIgnoreCase("c:/DetaDB/frontend/login")) {

// Cache c = new Cache();

// c.setValue(output);

// CacheManager.putCache(tablePath + ":" + pageBegin + ":" + pageEnd + ":" + direction, c);

// }

return output;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.lyg.vpc.process.portImpl;

import org.json.JSONObject;

import org.lyg.db.update.imp.UpdateRowsImp;

import org.lyg.vpc.process.companyImpl.LoginServiceImpl;

import java.util.HashMap;

import java.util.Map;

public class RestDBUpdateImpl {

public static Map<String, Object> updateRowByTablePathAndIndex(String tablePath

, String pageIndex,String culumnOfUpdateRow, String token, String email

, String password, String auth) throws Exception {

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

if(token != null && !token.equalsIgnoreCase("")){

String checkStatus = LoginServiceImpl.checkTokenStatus(token, "level");

if(checkStatus.contains("invalid")&&(auth.contains("1"))) {

output.put("loginInfo", "unsuccess");

output.put("returnResult", checkStatus);

return output;

}

}else if(email != null && !email.equalsIgnoreCase("")){

String checkStatus = LoginServiceImpl.checkRightsStatus(email, password, "DB");

if(checkStatus.contains("invalid")) {

output.put("loginInfo", "unsuccess");

output.put("returnResult", checkStatus);

return output;

}

}else{

output.put("loginInfo", "unsuccess");

output.put("returnResult", "invalid request");

return output;

}

JSONObject jaculumnOfUpdateRow = new JSONObject(culumnOfUpdateRow);

return UpdateRowsImp.updateRowByTablePathAndIndex(tablePath, pageIndex, jaculumnOfUpdateRow);

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.lyg.vpc.process.portImpl;

import org.json.JSONException;

import org.lyg.vpc.process.companyImpl.LoginServiceImpl;

import org.lyg.vpc.transaction.TransactionDelegate;

import java.io.IOException;

import java.util.HashMap;

import java.util.Map;

public class RestLoginPortImpl {

public static Map<String, Object> login(String uEmail, String uPassword) throws Exception {

Map<String, Object> map = TransactionDelegate.transactionLogin(uEmail, uPassword);

return map;

}

public static Map<String, Object> logout(String uEmail, String uToken) throws IOException {

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

output.put("userEmail", "friend");

output.put("userToken", "123456");

return output;

}

public static Map<String, Object> register(String uEmail, String uEmailEnsure

, String uName, String uPassword,

String uPassWDEnsure, String uAddress, String uPhone, String uWeChat,

String uQq, String uAge, String uSex) throws Exception {

Map<String, Object> output = TransactionDelegate.transactionRegister(uEmail

, uEmailEnsure, uName, uPassword,

uPassWDEnsure, uAddress, uPhone, uWeChat,

uQq, uAge, uSex);

return output;

}

public static Map<String, Object> change(String uEmail, String uChange

, String uChangeEnsure, String uToken,

String uPassword) throws IOException {

return null;

}

public static Map<String, Object> find(String uEmail) throws IOException {

return null;

}

public static Map<String, Object> checkStatus(String token) throws NumberFormatException

, JSONException, Exception {

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

String checkStatus = LoginServiceImpl.checkTokenStatusAndGetLevel(token, "level", output);

if(checkStatus.contains("invalid")) {

output.put("loginInfo", "unsuccess");

output.put("returnResult", checkStatus);

return output;

}

output.put("loginInfo", "success");

output.put("returnResult", checkStatus);

return output;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package org.lyg.vpc.transaction;

import com.google.gson.Gson;

import MD5Processor.Token;

import MD5Processor.Usr;

import MD5Processor.UsrToken;

import org.json.JSONObject;

import org.lyg.common.utils.StringUtil;

import org.lyg.common.utils.TokenUtil;

//import org.lyg.vpc.controller.company.LoginService;

import org.lyg.vpc.process.companyImpl.LoginServiceImpl;

import java.util.HashMap;

import java.util.Map;

public class TransactionDelegate {

public static Map<String, Object> transactionLogin(String uEmail, String uPassword)throws Exception {

Usr usr = LoginServiceImpl.findUsrByUEmail(uEmail);

UsrToken usrToken = LoginServiceImpl.findUsrTokenByUId(usr.getuId());

String password = TokenUtil.getSecondMD5Password(uPassword);

if (!password.equals(usr.getuPassword())) {

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

out.put("loginInfo", "unsuccess");

out.put("returnResult", "密码不正确");

return out;

}

Token token = TokenUtil.getNewTokenFromUsrAndUsrToken(usr, usrToken);

String json = new Gson().toJson(token);

String jsonToken = StringUtil.encode(json);

LoginServiceImpl.updateUsrTokenByUId(usr.getuId(), token.getuKey()

, password, token.getuTime()/1000);

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

out.put("userToken", jsonToken);

out.put("userEmail", uEmail);

out.put("loginInfo", "success");

return out;

}

public static Map<String, Object> transactionRegister(String uEmail, String uEmailEnsure

, String uName, String uPassword, String uPassWDEnsure, String uAddress

, String uPhone, String uWeChat, String uQq, String uAge,String uSex) throws Exception {

Usr usr = LoginServiceImpl.findUsrByUEmail(uEmail);

if(usr.getuEmail()!=null) {

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

out.put("loginInfo", "unsuccess");

out.put("returnResult", "邮箱已注册");

return out;

}

JSONObject jsobj=new JSONObject();

jsobj.put("u\_email", uEmail);

jsobj.put("u\_name", uName);

jsobj.put("u\_password", TokenUtil.getSecondMD5Password(uPassword));

jsobj.put("u\_address", uAddress);

jsobj.put("u\_phone", uPhone);

jsobj.put("u\_weChat", uWeChat);

jsobj.put("u\_qq", uQq);

jsobj.put("u\_age", uAge);

jsobj.put("u\_sex", uSex);

jsobj.put("u\_id", "random");

LoginServiceImpl.insertRowByTablePath("backend", "usr", jsobj);

usr = LoginServiceImpl.findUsrByUEmail(uEmail);

JSONObject jsobjToken = new JSONObject();

jsobjToken.put("u\_id", usr.getuId());

jsobjToken.put("u\_level", "low");

jsobjToken.put("u\_password", TokenUtil.getSecondMD5Password(uPassword));

LoginServiceImpl.insertRowByTablePath("backend", "usrToken", jsobjToken);

return transactionLogin(uEmail, uPassword);

}

}