第四章 德塔Socket 流PLSQL 数据库.

研发说明

德塔Socket流可编程数据库语言引擎系统API 1.0.0 说明书 final

作者: 罗瑶光
ID:430181198505250014
浏阳德塔软件开发有限公司
2019 年 6 月 24 日

1. 起源动机

作者第一次在南京炮兵学院学习王珊的数据库概论理论基础比较扎实。
作者2008 年在印度基督大学把图书馆里面所有的数据库书习题都做了一遍。
作者2009 年去长沙星沙国土局报考时当时冲动了一下说自己能独立制作做数据库软件。
作者2012 年上卡拉森教授的课用oracle 感觉太大了，想要是小点多好。
作者2015 年在ChinaCache(蓝讯) 用go 连接mysql，觉得很不错，如果数据库能将缓存插件化多好。。动机强烈了。
作者2016 年在亚米(yamibuy)上班时，公司发生了一起数据表误删的事件，但是作者有想法制作自动热备异 常恢复的binlog 系统，但是mysql 内核闭源，作者想自己研发的动机越来越明确。
2017 年作者在英特尔上班，接触了 mssql，发现索引序列和对象化编程有些冗杂，于是确定自己写了。
2018 年 11 月开始正式确立这个研发计划。为了满足作者类人生命系统的记忆管理功能。

2. 简介

德塔 Socket 流可编程数据库语言引擎系统作为 Deta 人工智能的核心组成部份主要任务就是记忆存储和管理，该数据的亮点为：一次执行多项复杂任务， 极为人性化的简易命令行 PLSQL 编程语言模型，打破现在所有传统关系数据库数据的单任务执行模式。基于双向队列的日志分析功能和命令语句执行方式和量子碎片文件存储思想，永远告别死锁。自动修复数据库，自动 rollback 异常数据操作，同时提供 rest 数据库操作接口和 plsql 命令行执行接口 2 种需求，满足各种场合的数据库操作应用。该数据库 admin 系统建立在 deta VPCS HTTP 服务器中， 整个系统启动时间 50 毫秒。有效和运维交互。

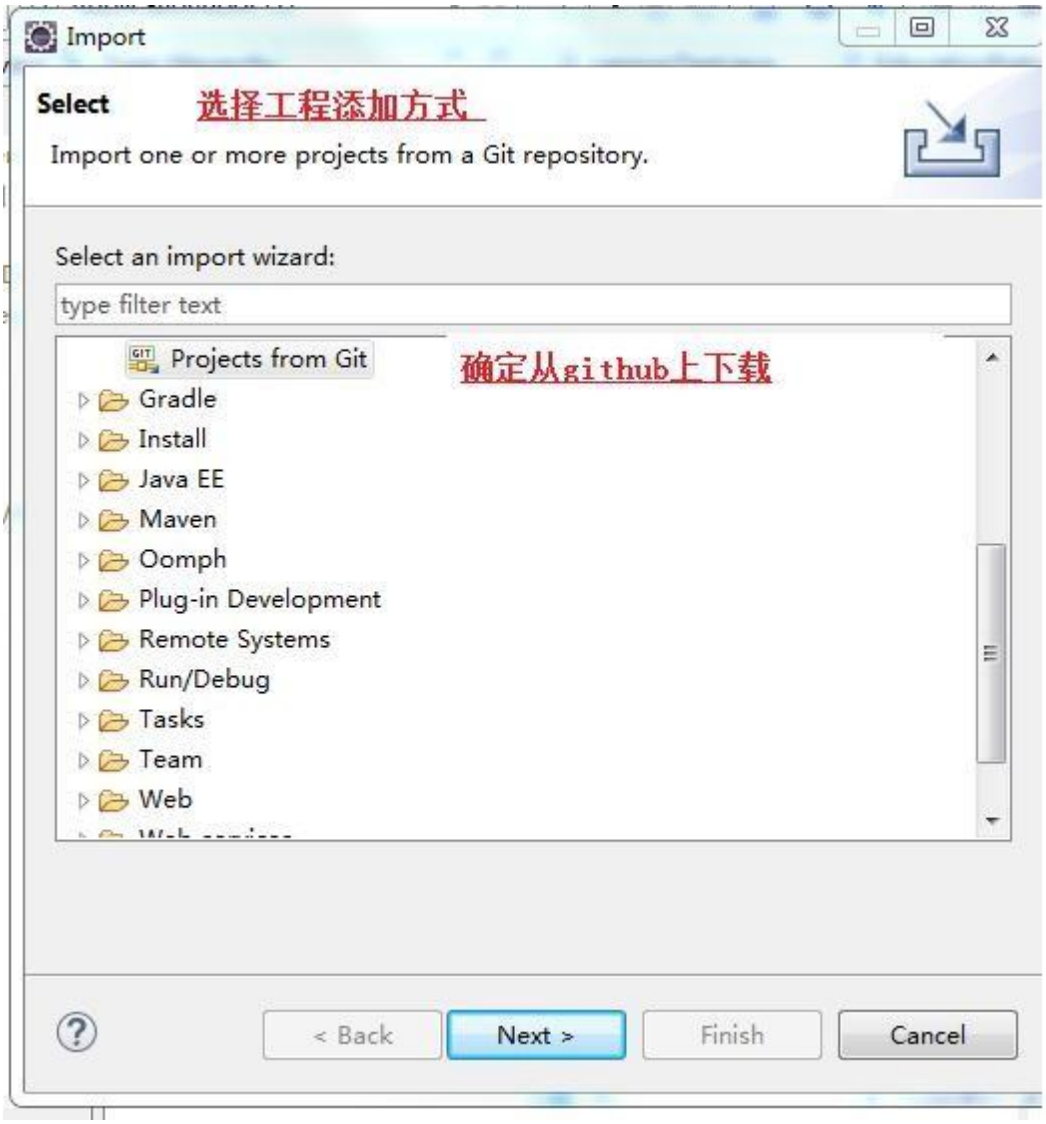
---

3 使用方法

3.1 下载 java 开发软件:
Eclipse: https://www.eclipse.org/

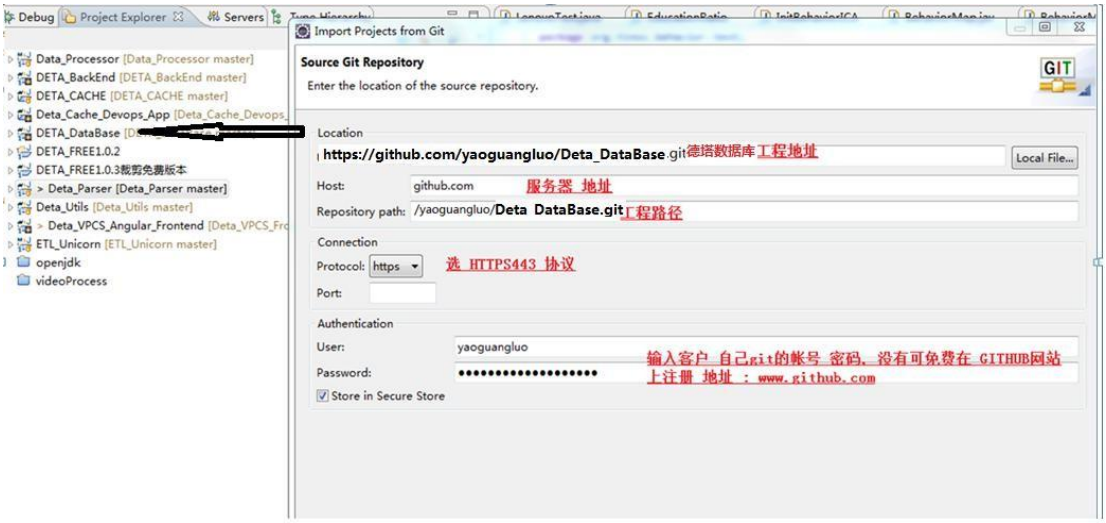
Intellij: https://www.jetbrains.com/idea/

3.2 导入 deta 图灵 api (API 是类库,接口 的意思, select 是选择 的意思)

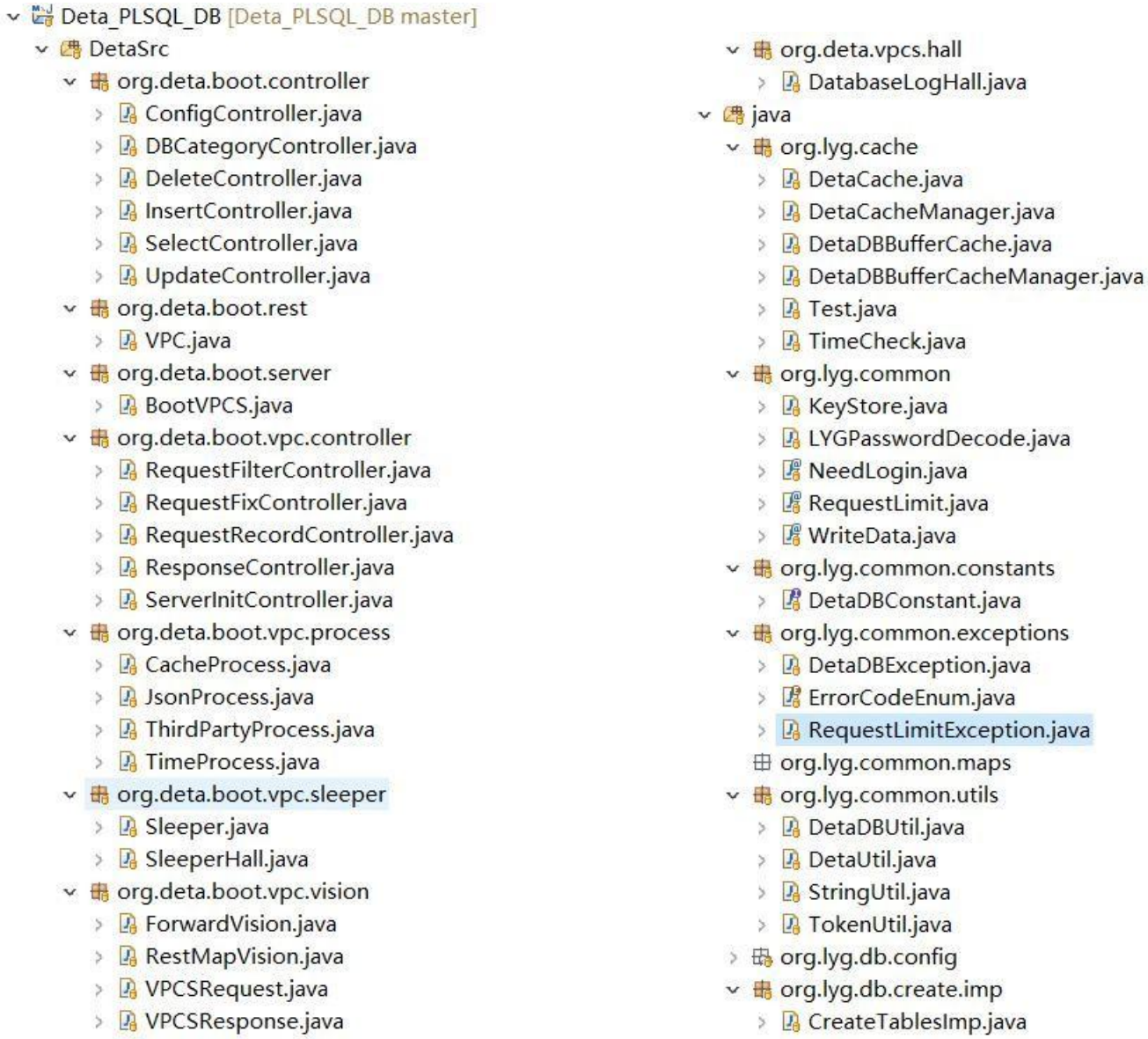


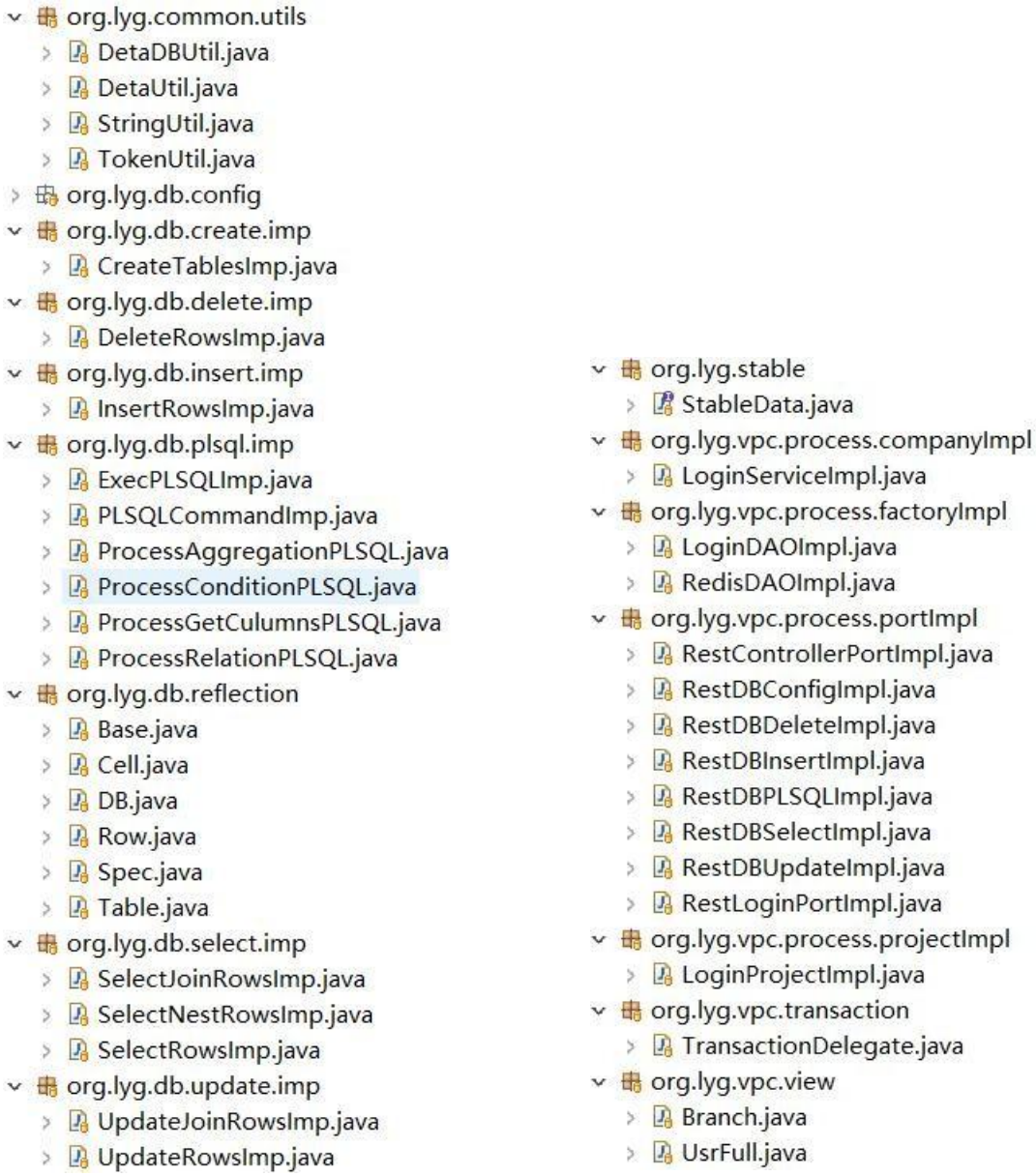
3.3 点 URI (uri 是互联网传输的一种协议规范关键字)

3.4 输入 Git 导入目标地址 (git 是版本持续化控制软件, repository 是 git 工程的下载标识, host 是远程 主机, repository path 是 git 工程 在主机上下载链接, protocol 是是通信协议, port 是端口, authentication 是密钥, user 是帐户名, password 是密码, store in secure store 是记录保存)



3.5 生成 eclipse 工程 因为是无插件底层源码, 所以可以自由集成为pom, gradle, web,或者general 工程模式. (POM 是 xml 形式的库标识 标识, gradle 是 模板形式, web 是 web 2.0 动态java 工程, general 是普通java 工程)


Deta_PLSQL_DB [Deta_PLSQL_DB master]
DetaSrc
org.deta.boot.controller
ConfigController.java
DBCategoryController.java
DeleteController.java
InsertController.java
SelectController.java
UpdateController.java
org.deta.boot.rest
VPC.java
org.deta.boot.server
BootVPCS.java
org.deta.boot.vpc.controller
RequestFilterController.java
RequestFixController.java
RequestRecordController.java
ResponseController.java
ServerInitController.java
org.deta.boot.vpc.process
CacheProcess.java
JsonProcess.java
ThirdPartyProcess.java
TimeProcess.java
org.deta.boot.vpc.sleeper
Sleeper.java
SleeperHall.java
org.deta.boot.vpc.vision
ForwardVision.java
RestMapVision.java
VPCSRequest.java
VPCSResponse.java
org.deta.vpcs.hall
DatabaseLogHall.java
java
org.lyg.cache
DetaCache.java
DetaCacheManager.java
DetaDBBufferCache.java
DetaDBBufferCacheManager.java
Test.java
TimeCheck.java
org.lyg.common
KeyStore.java
LYGPasswordDecode.java
NeedLogin.java
RequestLimit.java
WriteData.java
org.lyg.common.constants
DetaDBConstant.java
org.lyg.common.exceptions
DetaDBException.java
ErrorCodeEnum.java
RequestLimitException.java
org.lyg.common.maps
org.lyg.common.utils
DetaDBUtil.java
DetaUtil.java
StringUtil.java
TokenUtil.java
org.lyg.db.config
org.lyg.db.create.imp
CreateTablesImp.java



3.6 运行例子就可以了 所有 demo 和 test 都是 可运行实例 (demo 是例子的意思, test 是测试的意思 鼠标右键, 点运行就可以了.)
启动文件是：`BootVPCS.java`

3.7 可看到软件应用 例子:
http://tinos.qicp.vip/ 的登录系统

4 具体重要功能展示

4.1 Deta plsql 语言语法：见副本。《Deta_Database_PLSQL _V1.0 论文》

5. 适用范围

Deta 机器人记忆管理系统
Deta 复杂逻辑命令行数据库

Deta 分布式数据库大脑的记忆存储服务器节点。.

6. 注意

注意 1: 该作品免费版本使用权由国际软件研发协议apache-2.0 证书保护. 任何单位任意修改集成使用时请标注 Deta 公司 关键字: “浏阳德塔软件开发有限公司” 或者 “罗瑶光”
注意 5: 当前版本 2019-06-24 统一修改为 1.0.0, 一直在优化中,有任何bug 请直接联系作者.
QQ: 2080315360 (qq: 腾讯)
WECHAT: 15116110525 (WECHAT 微信)
TEL: 15116110525 (tel: 电话号码)
EMAIL: 2080315360@qq.com (email: 邮件地址)

7. 感谢

1 感谢中科大非对称筛子加密论文。
2 感谢谷歌提供了gson 的 json 数据格式包。
3 感谢南京炮兵学院，印度基督大学，和加洲路德大学德数据库课题组。为作者提供了近 7 年的数据理论知识。
4 感谢蓝讯集团，亚米，走四方，因特尔为作者提供了近 4 年的国外数据库应用管理知识。
5 感谢印度基督大学罗锡尼教授为作者传授了扎实的数据构造理论知识。

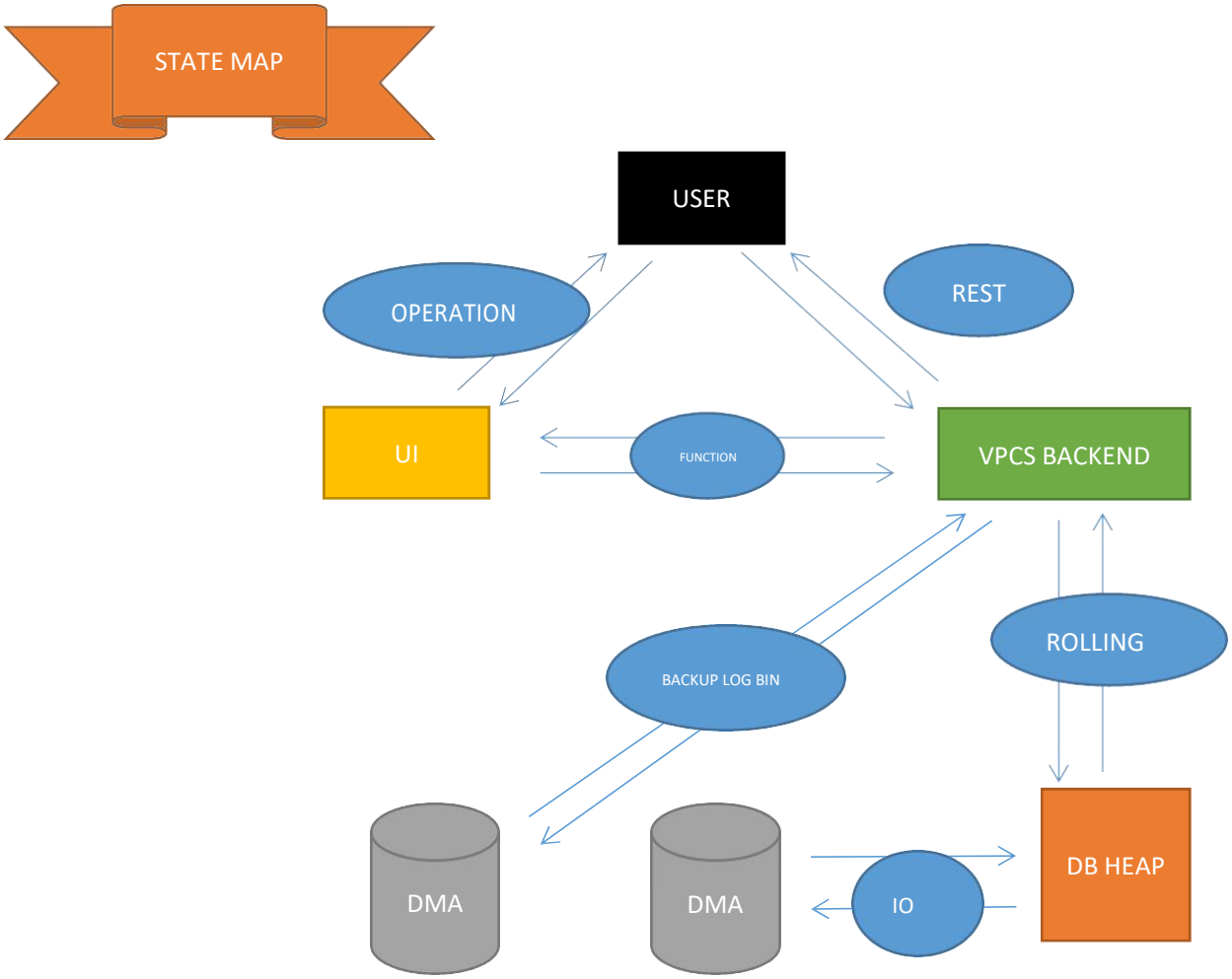
Deta 项目设计 采用 Mind Master 软件.
Deta 项目研发 采用 Eclipse IDE 软件.
Deta 项目测试 采用 JUNIT API 软件.
Deta 项目作品 主要采用 JAVA JDK8+.
作者长期使用 微星开发机器， 联想笔记本 windows 10 操作系统开发此项目, 电脑装 Avaster 杀毒软件和 360 软件保证其高效研发环境. 感谢 github 和 gitee 备份, 节省了作者 大量的存储硬盘, 同时方便 查阅, 逻辑 的鼠标键盘 给作者 提供了迅捷 的输入输出 便利 .当然 电信的网络, 老爸,老妈, 都要感谢的.

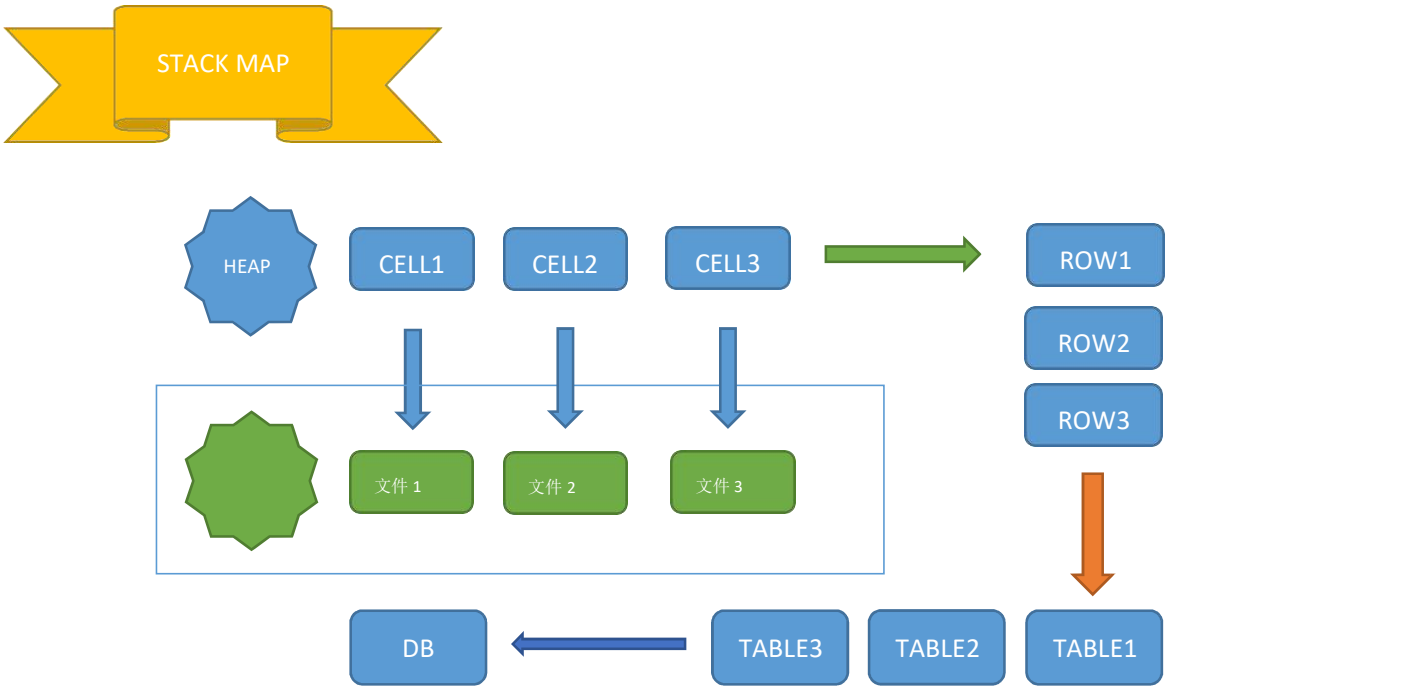
8 研发需要清单

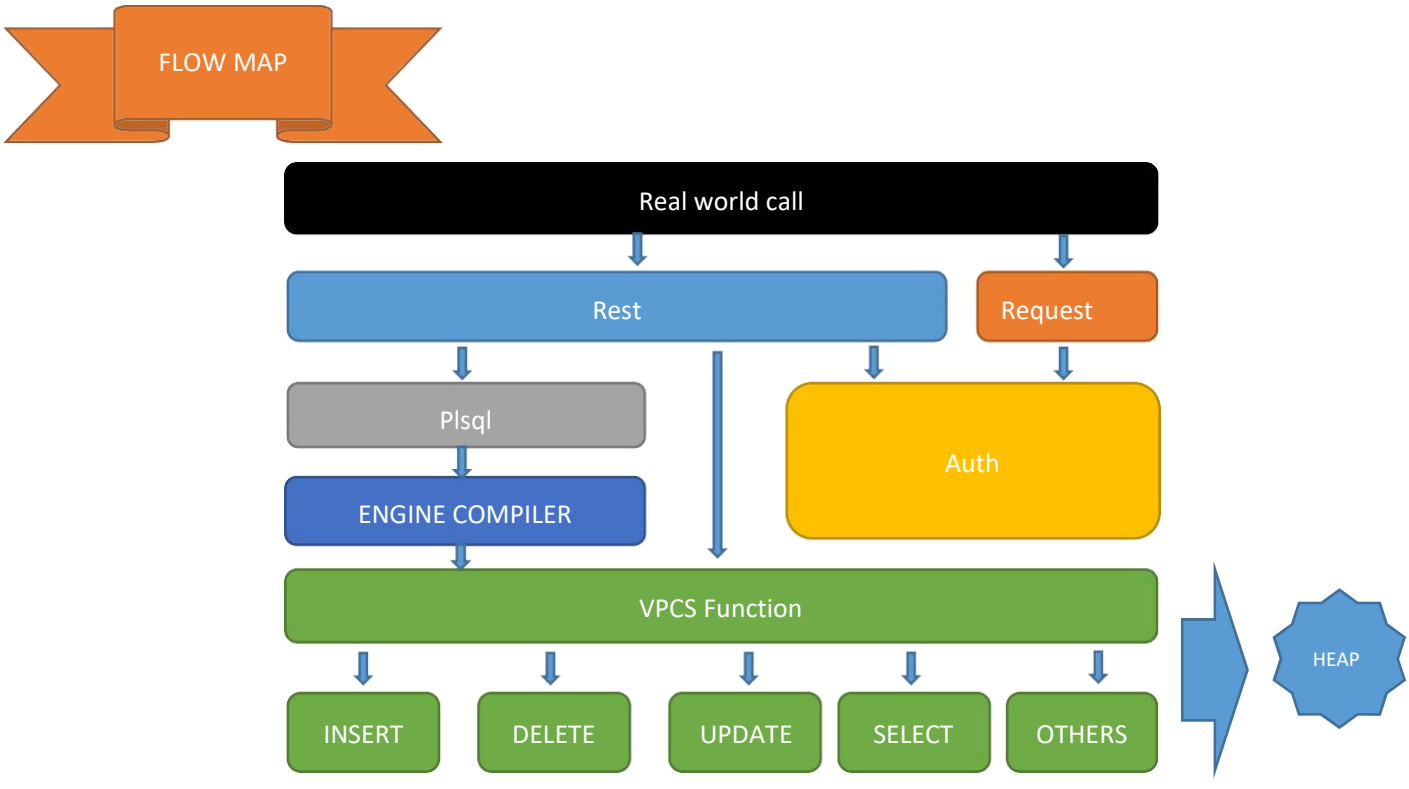
8.1 Java 编辑器.
8.2 Jdk8+. Java 虚拟机运行环境.
8.3 Junit 测试包.
8.4 一台连网的电脑.

DETA Socket PLSQL Database Framework

Yaoguang. Luo




STACK MAP
HEAP
CELL1
CELL2
CELL3
ROW1
ROW2
ROW3
文件 1
文件 2
文件 3
DB
TABLE3
TABLE2
TABLE1


FLOW MAP
Real world call
Rest
Request
Plsql
Auth
ENGINE COMPILER
VPCS Function
INSERT
DELETE
UPDATE
SELECT
OTHERS
HEAP

PLSQL 示例

DETA Database PLSQL

Mr. Yaoguang. Luo

Liu Yang Deta Software Development Limited Company, Hunan, China,

313699483@qq.com

Outline: *this document paper makes a pretty explanation of how does DETA database works by using PLSQL Method. At the same time, I will spend more care about the DETA PLSQL runs in the command line or rest call service, with a lot of real world samples.*

DETA PLSQL Commands

- **setRoot**:[path];
- **baseName**:[baseName];
- **tableName**:[tableName]:[operation];
- **getCulumns**:[difinition1]:[difinition2]:[difinition3]:[difinition4]:[difinition5].....;
- **culumnName**:[culumnName]:[dataType];
- **changeCulumnName**:[newCulumnName]:[oldCulumnName];
- **culumnValue**:[culumnName]:[culumnValue];
- **condition**:[operation]:[difinition1]:[difinition2]:[difinition3]....;
- **join**:[baseName]:[tableName];
- **relation**[operation]:[difinition1]:[difinition2]:[difinition3]....;
- **aggregate**[operation]:[difinition1]:[difinition2]:[difinition3]....;

Commands Definition

- **setRoot:[path];**

The setRoot:[path]; is mostly used for set the database path.

- **baseName:[baseName];**

The baseName:[baseName]; is mostly used for set the current database name in the PLSQL language compiler system.

- **tableName:[tableName]:[operation];**

The tableName:[tableName]'[operation]; is mostly used for set the current table name in current database with the operations. For example tableName:tableName:select; this command will tell PLSQL system, now begin to do the select function section.

- **getCulumns:[difinition1]:[difinition2]:[difinition3]:[difinition4]:[difinition5] ;**

The getCulumns:[difinition1]:[difinition2]:[difinition3]:[difinition4]:[difinition5]:.....; mostly be used for select columns.

- **columnName:[columnName]:[dataType];**

The columnName:[columnName]:[dataType]; mostly be used for create the table columns.

- **changeCulumnName:[oldCulumnName]:[newCulumnName];**

The changeCulumnName:[newCulumnName]:[oldCulumnName]; mostly be used for change the table columns.

- **culumnValue:[columnName]:[columnValue];**

The culumnValue:[columnName]:[columnValue]; mostly be used for update the columns value.

- **condition:[operation]:[difinition1]:[difinition2]:[difinition3]:...;**

The condition:[operation]:[difinition1]:[difinition2]:[difinition3]:...; mostly be used for

- **join:[baseName]:[tableName];**

The join:[baseName]:[tableName]; mostly be used for select and update of delete with conditions.

- **relation[operation]:[difinition1]:[difinition2]:[difinition3]:...;**

The relation[operation]:[difinition1]:[difinition2]:[difinition3]:...; mostly be used for join section condition

- **aggregate[operation]:[difinition1]:[difinition2]:[difinition3]:...;**

The aggregate[operation]:[difinition1]:[difinition2]:[difinition3]:...; mostly be used for limit, sort or addition operations.

Command Samples

select samples

tableName:test:select;

condition:or:testCulumn1|<|20:testCulumn2|==|fire;

condition:and:testCulumn1|>|100:testCulumn2|==|fire;

select where in samples

setRoot:C:/DetaDB;

baseName:backend;

tableName:usr:select;

condition:or:u_id|in|3,4,5;

select join samples

tableName:utest:select;

condition:or:testCulumn1|<|20:testCulumn2|==|fire;

condition:and:testCulumn1|>|100:testCulumn2|==|fire;

join:stest;

relation:or:uid|==|sid:ussd|==|sssd;

relation:and:utoken|=!|stoken:umap|==|smap;

select join samples

tableName:utest:select;

condition:or:utestCulumn1|<|20:utestCulumn2|==|fire;

condition:and:utestCulumn1|>|100:utestCulumn2|==|fire;

getCulumns:utestCulumn1|as|uid::utestCulumn2|as|ussd:utoken:umap;

join:backend:stest;

condition:and:stestCulumn1|>|100:stestCulumn2|==|fire;

getCulumns:stestCulumn1|as|sid|:stestCulumn2|as|sssd:stoken:smap;

relation:or:uid|==|sid:ussd|==|sssd;

relation:and:utoken|=!|stoken:umap|==|smap; **aggregation**:limit:2|~|10;

insert samples

tableName:test:insert;

culumnValue:date0:19850525;

culumnValue:date1:19850526;

culumnValue:date2:19850527;

culumnValue:date3:19850528;

culumnValue:date4:19850529;

update samples

tableName:test:update;

condition:or:testCulumn1|<|20:testCulumn2|==|fire;

condition:and:testCulumn1|>|100:testCulumn2|==|fire;

culumnValue:date0:19850525;

culumnValue:date1:19850526;

update samples

tableName:test:update;

condition:or:testCulumn1|<|20:testCulumn2|==|fire;

condition:and:testCulumn1|>|100:testCulumn2|==|fire;

join:backend:utest;

condition:and:uCulumn3|<|20;

relation:and:testCulumn1|==|uCulumn1:testCulumn2|!=|uCulumn2;

culumnValue:date0:19850525;

culumnValue:date1:19850526;

delete samples

tableName:test:delete;

condition:or:testCulumn1|<|20:testCulumn2|==|fire;

condition:and:testCulumn1|>|100:testCulumn2|==|fire;

create samples

tableName:test:create;

culumnName:pk:culumn1:string;

culumnName:uk:culumn1:long;

culumnName:uk:culumn1:obj;

culumnName:nk:culumn1:double;

drop samples

tableName:test:drop;

change samples

tableName:test:change;

changeCulumnName:oldCulumnName:newCulumnName;

Real World Samples By Using DETA PLSQL Database

setRoot:C:/DetaDB;

baseName:backend;

tableName:usr:select;

condition:or:u_id|<=|3:u_id|>|7;

condition:and:u_email|!equal|321:u_name|!equal|123;

getCulumns:u_id|as|detaId:u_email|as|detaEmail;

join:backend:usrToken;

condition:and:u_level|equal|low;

getCulumns:u_id|as|sId:u_level:u_password|as|SSID;

relation:and:detaId|==|sId;

aggregation:limit:0|~|1;

compare Tranditioanl SQL:

SELECT u.u_id as detaId，u.u_email as detaEmail, t.u_id as sId，t.u_level，t.u_password as SSID
FROM usr as U
INNER JOIN (**SELECT** t.u_id as sId，t.u_level，t.u_password as SSID
FROM usrToken as t
WHERE t.u_level equal "low") AS B on U.detaId == B.sId;

```
WHERE (u.u_id <=3 || u.u_id>7 ) && (u.u_email !equal '321' && u.u_name !equal 123);
LIMIT 0,1;
```

Acknowledgement

The DETA PLSQL database system source code link:
https://github.com/yaoguangluo/DETA_DataBase

研发源码

```
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)){ ret
            urn VtoV.ObjectToJsonString(RestDBConfigImpl.setDBPath(data
                    .get("baseName")
                    , data.get("baseName"), data.get("baseName")));
        }
        if(string.equalsIgnoreCase(StableData.REST_PATH_SET_DB_TABLE)){ re
            turn 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)){ ret
            urn 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_INDE
X)){
            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)){ r
            eturn 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)){ r
            eturn 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)){ ret
            urn 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_INDE
X)){
            return VtoV.ObjectToJsonString(RestDBUpdateImpl
                    .updateRowByTablePathAndIndex(data.get("tablePath")
                            , data.get("pageIndex"), data.get("columnOfUpdateRow")
                            , 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)){ ret
            urn ConfigController.exec(string, data);
        }
        if(string.contains(StableData.REST_PATH_INSERT)){ ret
            urn InsertController.exec(string, data);
        }
        if(string.contains(StableData.REST_PATH_DELETE)){ re
            turn DeleteController.exec(string, data);
        }
        if(string.contains(StableData.REST_PATH_UPDATE)){ re
            turn 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())){ vPCSRes
            ponse.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.HTT
            P_400); return;
        }
        if(mess.equalsIgnoreCase(StableData.STRING_EMPTY)){ v
            PCSResponse.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)){ Stri
                ng 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(StableDat
                a.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(StableDat
                a.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)){ RestMapVi
            sion.processRest(vPCSRequest, vPCSResponse);
        }
        if(vPCSRequest.getRequestForwardType().equalsIgnoreCase(StableData.STREAM_BYTES)){ RestMap
            Vision.processBytes(vPCSRequest, vPCSResponse);
        }
        if(vPCSRequest.getRequestForwardType().equalsIgnoreCase(StableData.STREAM_BUFFER)){ RestMa
            pVision.processBuffer(vPCSRequest, vPCSResponse);
        }

    if(vPCSRequest.getRequestForwardType().equalsIgnoreCase(StableData.STREAM_BYTES_BUFFE
R)){
            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.l
ength())
                :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())){ con
            tentBuilderToString = 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.printStackTrac
                        e();
                    }
                    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(column.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(column, 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){ ProcessC
                                onditionPLSQL.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){ ProcessC
                                onditionPLSQL.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.pr
                                ocessCache(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.pr
                                ocessCache(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){ ProcessC
                            onditionPLSQL.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().s
                et("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);
        }
}
```

第四节 VPCS Standard函数提取

```
package OSI.AOP.MS.VPC.server;
import java.io.IOException;
import java.net.ServerSocket;
import MS.VPC.PP.Time_P;
import MS.VPC.SH.Sleeper_H;
//import OM.config.Config;
import SVQ.stable.StableWeb;
//VPCS标准函数,准备之后所有服务器 走继承这个文件。
//罗瑶光 20200811
public class ServerInit_Standard {
 private ServerSocket server;
 private int port;
 public void IV_Service(String 前端接口Txt, String 服务器名) throws IOException {
            try {
                        //port= Integer.parseInt(properties.getProperty(StableData.TCP_PORT));
                        port= Integer.valueOf(前端接口Txt);
                        //port= Config.detaVPCSDBPort;
                        server= new ServerSocket(port);
                        System.out.println("----德塔VPCS"+ 服务器名+ "端口启动:" + port);
                        System.out.println("----德塔VPCS"+ 服务器名+ "DMA确认:成功！");
                        RequestFilter_C.IV_BlockList();
                        System.out.println("----德塔VPCS"+ 服务器名+ "IP过滤服务启动:成功！");
            } catch (Exception e) {
                        e.printStackTrace();
            }
```

```
 }

 private void haoHiYooFaker(Sleeper_H sleeper_H) {
        sleeper_H.callSkivvy();
 }

 public void IV_Server(String 前端接口Txt, String 服务器名) throws IOException {
        System.out.println("----DETA VPCS--3.0");
        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");
        Time_P time_P= new Time_P();
        time_P.begin();
        Sleeper_H sleeper_H= new Sleeper_H();
        IV_Service(前端接口Txt, 服务器名);
        time_P.end();
        System.out.println("----德塔VPCS"+ 服务器名+ "启动一切正常-总耗时:"
                        + time_P.duration()+ "毫秒");
        while(true){
                if(sleeper_H.getThreadsCount()< StableWeb.SLEEPERS_RANGE){
                        ServerSleeper_Standard sleeper= new ServerSleeper_Standard(前端接口Txt);
                        try {
                                sleeper.hugPillow(sleeper_H, server.accept()
                                                , sleeper.hashCode());
                                sleeper.start();
                        } catch (IOException e) {
                                e.printStackTrace();
                        }
                }else {
                        haoHiYooFaker(sleeper_H);
                }
        }
 }
}
```

```
package OSI.AOP.MS.VPC.server;

import java.io.IOException;
import java.net.Socket;
import MS.VPC.SH.Sleeper_H;
//这是标准的sleepr函数，我稍后会设计 frontend sleeper， backend sleeper
//，cache sleeper， database sleeper 继承它， 避免循环嵌套。
public class ServerSleeper_Standard extends Thread implements Runnable{
 public VPCSRequest vPCSRequest;
 public VPCSResponse vPCSResponse;

 public ServerSleeper_Standard(String port){
```

```
		vPCSRequest= new VPCSRequest();
		vPCSRequest.I_RequestPort(port);
		vPCSResponse= new VPCSResponse();
		vPCSResponse.I_HashCode(this.hashCode());
}

//合并
public ServerSleeper_Standard(){
		vPCSRequest= new VPCSRequest();
		vPCSResponse= new VPCSResponse();
		vPCSResponse.I_HashCode(this.hashCode());
}
public void run(){
		try{
			RequestRecord_C.requestIpRecoder(vPCSRequest, vPCSResponse);
			if(vPCSResponse.getSocket().isClosed()) {
				return;
			}
			RequestRecord_C.requestLinkRecoder(vPCSRequest, vPCSResponse);
			if(vPCSResponse.getSocket().isClosed()) {
				return;
			}
			RequestFilter_C.requestIpFilter(vPCSRequest, vPCSResponse);
			if(vPCSResponse.getSocket().isClosed()) {
				return;
			}
			RequestFilter_C.requestLinkFilter(vPCSRequest, vPCSResponse);
			if(vPCSResponse.getSocket().isClosed()) {
				return;
			}
			RequestFix_C.requestIpFix(vPCSRequest, vPCSResponse);
			if(vPCSResponse.getSocket().isClosed()) {
				return;
			}
			RequestFix_C.requestLinkFix(vPCSRequest, vPCSResponse);
			if(vPCSResponse.getSocket().isClosed()) {
				return;
			}
			ServerForward_Standard.IQ_ForwardType(vPCSRequest, vPCSResponse);
			if(vPCSResponse.getSocket().isClosed()) {
				return;
			}
			ServerForward_Standard.forwardToRestMap(vPCSRequest, vPCSResponse);
			if(vPCSResponse.getSocket().isClosed()) {
				return;
			}
			ServerRestMap_Standard.IQ_Response(vPCSRequest, vPCSResponse);
			if(vPCSResponse.getSocket().isClosed()) {
```

```
                        return;
                }
                ServerRestMap_Standard.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(Sleeper_H sleeper_H, Socket accept, int hashCode) {
        sleeper_H.I_E_Sleeper(hashCode, this);
        vPCSResponse.I_Socket(accept);
        vPCSResponse.I_Sleeper_H(sleeper_H);
 }
}
```

```
package OSI.AOP.MS.VPC.server;

import java.util.Map;

public class VPCSRequest {
 private String requestIp;
 private String requestName;
 private String requestLink;
 private boolean requestIsRest;
 private String requestFilePath;
 private String requestFileCode;
 private String requestForwardType;
 //避免循环嵌套，加个port标识。稍后看要不要做继承分类。20210730
 private String requestPort= "";
 private String requestIpFix;
 private String requestNameFix;
 private String requestLinkFix;
 private Map<String, String> requestValue;

 public String getRequestPort() {
        return requestPort;
 }
```

```
public void I_RequestPort(String requestPort) {
	this.requestPort = requestPort;
}

public String getRequestIp() {
	return requestIp;
}

public void I_RequestIp(String requestIp) {
	this.requestIp = requestIp;
}

public String getRequestName() {
	return requestName;
}

public void I_RequestName(String requestName) {
	this.requestName = requestName;
}

public String getRequestLink() {
	return requestLink;
}

public void I_RequestLink(String requestLink) {
	this.requestLink = requestLink;
}

public String getRequestIpFix() {
	return requestIpFix;
}

public void I_RequestIpFix(String requestIpFix) {
	this.requestIpFix = requestIpFix;
}

public String getRequestNameFix() {
	return requestNameFix;
}

public void I_RequestNameFix(String requestNameFix) {
	this.requestNameFix = requestNameFix;
}

public String getRequestLinkFix() {
	return requestLinkFix;
}
```

```
public void I_RequestLinkFix(String requestLinkFix) {
		this.requestLinkFix = requestLinkFix;
}

public Map<String, String> getRequestValue() {
		return requestValue;
}

public void I_RequestValue(Map<String, String> requestValue) {
		this.requestValue = requestValue;
}

public boolean getRequestIsRest() {
		return requestIsRest;
}

public void I_RequestIsRest(boolean requestIsRest) {
		this.requestIsRest = requestIsRest;
}

public String getRequestForwardType() {
		return requestForwardType;
}

public void I_RequestForwardType(String requestForwardType) {
		this.requestForwardType = requestForwardType;
}

public String getRequestFilePath() {
		return requestFilePath;
}

public void I_RequestFilePath(String requestFilePath) {
		this.requestFilePath = requestFilePath;
}

public String getRequestFileCode() {
		return requestFileCode;
}

public void I_RequestFileCode(String requestFileCode) {
		this.requestFileCode = requestFileCode;
}
}
```

```
package OSI.AOP.MS.VPC.server;
import java.io.IOException;
import java.io.PrintWriter;
import java.net.Socket;

import MS.VPC.SH.Sleeper_H;
public class VPCSResponse{
 public Socket getSocket() {
		return socket;
 }

 public void I_Socket(Socket socket) {
		this.socket = socket;
 }

 public Sleeper_H getSleeper_H() {
		return sleeper_H;
 }

 public void I_Sleeper_H(Sleeper_H sleeper_H) {
		this.sleeper_H = sleeper_H;
 }

 public Integer getHashCode() {
		return hashCode;
 }

 public void I_HashCode(Integer hashCode) {
		this.hashCode = hashCode;
 }

 public int getErrorCode() {
		return errorCode;
 }

 public void I_ErrorCode(int errorCode) {
		this.errorCode = errorCode;
 }

 public String getResponseContentType() {
		return ResponseContentType;
 }

 public void I_ResponseContentType(String responseContentType) {
		ResponseContentType = responseContentType;
 }

 private Socket socket;
```

```
 private Sleeper_H sleeper_H;
 private Integer hashCode;
 private int errorCode;
 private String ResponseContentType;
 public void return404() throws IOException {
        if(socket.isClosed()) {
                this.sleeper_H.D_ThreadById(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.sleeper_H.D_ThreadById(this.hashCode);
 }

 public void returnErrorCode(Integer errorCode) throws IOException {
        if(socket.isClosed()) {
                this.sleeper_H.D_ThreadById(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.sleeper_H.D_ThreadById(this.hashCode);
 }
}
```

```
package OSI.AOP.MS.VPC.server;
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 MS.OP.SM.AOP.MEC.SIQ.cache.DetaCache_M;
import SVQ.stable.StablePOS;
import SVQ.stable.StableWeb;
import OSI.AOP.VPC.rest.VPC;
import PEU.P.zip.*;
//这里出现了VPC的标识,让走四方看清楚,我有很多方法来继承,我就不用走四方的那种. VPCS 的STABLE就可以解决
//是不是我写的.（详细描述下，走四方当年php 多线程卡到爆，戴海燕总被我念的头皮发麻 最后用了java， 我overload了接口）
//去app 函数标识，准备继承 用standard来区分。（当时是php 之前员工已经写好了，我只是java翻译而已，所以没用我的思维写，
//下面代码我的习惯是如下，我喜欢类c的小片段继承。从不用overload）
// 罗瑶光
public class ServerRestMap_Standard {
 public static void main(String[] args){

 }

 public static void IQ_Response(Socket socket) {

 }

 public static void returnResponse(Socket socket) {

 }

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

 }

 public static void returnResponse(VPCSRequest vPCSRequest, VPCSResponse vPCSResponse) {
        vPCSResponse.getSleeper_H().D_ThreadById(vPCSResponse.getHashCode());
 }

 public static void P_Rest(VPCSRequest vPCSRequest, VPCSResponse vPCSResponse) throws Exception {
        //VPC属于子继承, 如果不用 overrider 来分配, 也有很多方法, 如osgi, 当然,我现在用最快map标识,更爽.
        //indexVPCMapPillows.get(vPCSRequest.gettag())...;
        //现在仅仅deta的网站处理服务器有web页, 养疗经app还没有涉及, 因此 pillow tag 区分VPC 函数的rest map 设计优先级稍后
        String output= VPC.forward(vPCSRequest.getRequestLink()
                        , vPCSRequest.getRequestValue());
        output= output.length()> 0? output:" ";
        PrintWriter printWriter= new PrintWriter(new BufferedWriter(new OutputStreamWriter(vPCSResponse.getSocket()
                        .getOutputStream(),StableWeb.CHARSET_UTF_8)),true);
        printWriter.println(StableWeb.HEADER_HTTP_200_OK_DOUBLE_ENTER);
        output=output.charAt(StablePOS.INT_ZERO)==""?output.substring(StablePOS.INT_ONE, output.length())
                        :output;
        output=output.charAt(output.length()-StablePOS.INT_ONE)==""?output.substring(StablePOS.INT_ZERO
                        , output.length()-StablePOS.INT_ONE):output;
        printWriter.println(output.replace("\\\"","\""));
        System.out.println("db:"+4);
        printWriter.flush();
```

```
        printWriter.close();
        vPCSResponse.getSleeper_H().D_ThreadById(vPCSResponse.getSocket().hashCode());
}

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

}

public static void P_Bytes(VPCSRequest vPCSRequest
                , VPCSResponse vPCSResponse) throws IOException {
        List<byte[]> list;
        DataOutputStream dataOutputStream = new DataOutputStream(vPCSResponse.getSocket().getOutputStream());
        if(null != DetaCache_M.getCacheOfBytesList(vPCSRequest.getRequestFilePath())){
                list = DetaCache_M.getCacheOfBytesList(vPCSRequest.getRequestFilePath());
        }else{
                FileInputStream fileInputStream = new FileInputStream(new File(vPCSRequest.getRequestFilePath()));
                ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
                byte[] byteArray = new byte[StableWeb.BUFFER_RANGE_MAX];
                int lengthOfFile = StablePOS.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, StablePOS.INT_ZERO
                                , StableWeb.BUFFER_RANGE_MAX)) != StablePOS.INT_ERROR){
                        byteArrayOutputStream.write(byteArray, StablePOS.INT_ZERO, lengthOfFile);
                }
                fileInputStream.close();
                byte[] sniper = GzipUtil.compress(byteArrayOutputStream.toByteArray());
                list.add(0, vPCSResponse.getResponseContentType().getBytes(StableWeb.CHARSET_UTF8));
                list.add(0, (StableWeb.HEADER_CONTENT_LENGTH + sniper.length + StableWeb.STRING_SPACE_ENTER)
                                .getBytes(StableWeb.CHARSET_UTF8));
                list.add(0, StableWeb.HEADER_CACHE_CONTROL.getBytes(StableWeb.CHARSET_UTF8));
                list.add(0, StableWeb.HEADER_CONTENT_ENCODING_GZIP.getBytes(StableWeb.CHARSET_UTF8));
                list.add(0, StableWeb.HEADER_ACCEPT_RANGES_BYTES.getBytes(StableWeb.CHARSET_UTF8));
                list.add(0, StableWeb.HEADER_HOST.getBytes(StableWeb.CHARSET_UTF8));
                list.add(0, StableWeb.HEADER_HTTP_200_OK.getBytes(StableWeb.CHARSET_UTF8));
                if(null != sniper && sniper.length>StablePOS.INT_ZERO) {
                        list.add(sniper);
                }
                DetaCache_M.putCacheOfBytesList(vPCSRequest.getRequestFilePath(), list);
        }
        Iterator<byte[]> iterator = list.iterator();
        while(iterator.hasNext()){
                dataOutputStream.write(iterator.next());
        }
        dataOutputStream.flush();
        dataOutputStream.close();
}
```

```
public static void P_Buffer(VPCSRequest vPCSRequest, VPCSResponse vPCSResponse) throws IOException {
	String builderToString;
	if(null != DetaCache_M.getCacheOfString(vPCSRequest.getRequestFilePath())){
		builderToString = DetaCache_M.getCacheOfString(vPCSRequest.getRequestFilePath());
	}else{
		StringBuilder stringBuilder = new StringBuilder();
		stringBuilder.append(StableWeb.HEADER_HTTP_200_OK);
		stringBuilder.append(StableWeb.HEADER_HOST);
		stringBuilder.append(StableWeb.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();
		DetaCache_M.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 P_BufferBytes(VPCSRequest vPCSRequest
		, VPCSResponse vPCSResponse) throws UnsupportedEncodingException, IOException {
	StringBuilder stringBuilder = new StringBuilder();
	stringBuilder.append(StableWeb.HEADER_HTTP_200_OK);
	stringBuilder.append(StableWeb.HEADER_HOST);
	stringBuilder.append(StableWeb.HEADER_CACHE_CONTROL);
	stringBuilder.append(StableWeb.HEADER_CONTENT_ENCODING_GZIP);
	stringBuilder.append(vPCSResponse.getResponseContentType());
	String builderToString= stringBuilder.toString();
	String contentBuilderToString;
	if(null != DetaCache_M.getCacheOfString(vPCSRequest.getRequestFilePath())){
		contentBuilderToString = DetaCache_M
				.getCacheOfString(vPCSRequest.getRequestFilePath());
	}else{
		StringBuilder contentBuilder = new StringBuilder();
		FileInputStream fileInputStream = new FileInputStream(new File(vPCSRequest
				.getRequestFilePath()));
		int lengthOfFile = StablePOS.INT_ZERO;
```

```
                byte[] byteArray = new byte[StableWeb.BUFFER_RANGE_MAX];
                while ((lengthOfFile = fileInputStream.read(byteArray)) != StablePOS.INT_ERROR){
                        contentBuilder.append(new String(byteArray, StablePOS.INT_ZERO, lengthOfFile
                                        , StableWeb.CHARSET_UTF_8));
                }
                fileInputStream.close();
                contentBuilderToString = contentBuilder.toString();
                DetaCache_M.putCacheOfString(vPCSRequest.getRequestFilePath(), contentBuilderToString);
        }
        DataOutputStream dataOutputStream = new DataOutputStream(vPCSResponse.getSocket().getOutputStream());
        dataOutputStream.write(builderToString.getBytes(StableWeb.CHARSET_UTF8));
        dataOutputStream.write(GzipUtil.compress(contentBuilderToString.getBytes(StableWeb.CHARSET_UTF8)));
        dataOutputStream.flush();
        dataOutputStream.close();
}

public static void P_BytesWithoutZip(VPCSRequest vPCSRequest, VPCSResponse vPCSResponse) throws IOException {
        List<byte[]> list;
        DataOutputStream dataOutputStream = new DataOutputStream(vPCSResponse.getSocket().getOutputStream());
        if(DetaCache_M.getCacheOfBytesList(vPCSRequest.getRequestFilePath()) != null){
                list = DetaCache_M.getCacheOfBytesList(vPCSRequest.getRequestFilePath());
        }else{
                FileInputStream fileInputStream = new FileInputStream(new File(vPCSRequest.getRequestFilePath()));
                ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
                byte[] byteArray = new byte[1024];
                int len = 0;
                list= new ArrayList<>();
                //下面三行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((len = fileInputStream.read(byteArray, 0, 1024))!=-1){
                        byteArrayOutputStream.write(byteArray, 0, len);
                }
                fileInputStream.close();
                byte[] sniper =byteArrayOutputStream.toByteArray();
                list.add(0, vPCSResponse.getResponseContentType().getBytes(StableWeb.CHARSET_UTF8));
                list.add(0, (StableWeb.HEADER_CONTENT_LENGTH + sniper.length + " \n").getBytes(StableWeb.CHARSET_UTF8));
                list.add(0, (StableWeb.HEADER_CACHE_CONTROL).getBytes(StableWeb.CHARSET_UTF8));
                list.add(0, StableWeb.HEADER_ACCEPT_RANGES_BYTES.getBytes(StableWeb.CHARSET_UTF8));
                list.add(0, StableWeb.HEADER_HOST.getBytes(StableWeb.CHARSET_UTF8));
                list.add(0, StableWeb.HEADER_HTTP_200_OK.getBytes(StableWeb.CHARSET_UTF8));
                list.add(sniper);
                DetaCache_M.putCacheOfBytesList(vPCSRequest.getRequestFilePath(), list);
        }
        Iterator<byte[]> iterator = list.iterator();
        while(iterator.hasNext()){
                byte[] bytes = null;
                int i= 0;
                try {
```

```
                        bytes=iterator.next();
                        if(bytes.length>10000) {
                                int last= bytes.length%10000;
                                for(i= 0; i< bytes.length- 10000; i+= 10000) {
                                        byte[] serparBytes = new byte[10000];
                                        for(int j= 0; j< 10000; j++) {
                                                serparBytes[j]= bytes[i+ j];
                                        }
                                        Thread.sleep(250);
                                        dataOutputStream.write(serparBytes);
                                        dataOutputStream.flush();
                                }
                                byte[] serparBytes = new byte[last];
//                                      i-=10000;
                                for(int j=0; j<last;j++) {
                                        serparBytes[j]= bytes[i+ j];
                                }
                                dataOutputStream.write(serparBytes);
                        }else {
                                dataOutputStream.write(bytes);
                        }
                }catch(Exception e) {
                        System.out.print(i);
                        System.out.print(bytes.length);
                        System.out.print(vPCSRequest.getRequestLink());
                        System.out.print(vPCSRequest.getRequestFilePath());
                        e.printStackTrace();
                }
        }
        dataOutputStream.flush();
        dataOutputStream.close();
 }
}
```

```
package OSI.AOP.MS.VPC.server;
import java.io.File;
import java.io.IOException;
import java.util.Map;
import OSI.AOP.MS.VPC.C.DBCategory_C;
import OSI.AOP.VPC.C.Config_C;
import OSI.AOP.VPC.C.D_C;
import OSI.AOP.VPC.C.IU_C;
import OSI.AOP.VPC.C.Q_C;
import OSI.AOP.VPC.C.U_C;
import OSI.AOP.VPC.PP.port_E.RestAskPort_E;
import OSI.AOP.VPC.PP.port_E.RestLoginPort_E;
```

```
import PEU.P.map.*;
//作者 罗瑶光
public class ServerVPC_Standard {

 public static String forward(String string, Map<String, String> data) throws Exception {
		//			if(!DNAAuth.DNAAuthStatusCheckEmailAndPassword(app, string, data)) {
		//				return "";
		//			}
		//controller
		if(string.contains("/select")){
			return Q_C.exec(string, data);
		}
		if(string.contains("/setDB")){
			return Config_C.exec(string, data);
		}
		if(string.contains("/insert")){
			return IU_C.exec(string, data);
		}
		if(string.contains("/delete")){
			return D_C.exec(string, data);
		}
		if(string.contains("/update")){
			return U_C.exec(string, data);
		}
		if(string.contains("DBCategory")){
			return DBCategory_C.exec(string, data);
		}
		//restMap // 准备改成1次进入由原来的 前端 后端数据库， 变成前端 数据库。 模式为前端加cache，后端加数据库。 罗瑶光
		if(string.equalsIgnoreCase("/login")){
			return VtoV.ObjectToJsonString(RestLoginPort_E.login(data.get("uEmail"),data.get("uPassword")));
		}
		if(string.equalsIgnoreCase("/find")){
			return VtoV.ObjectToJsonString(RestLoginPort_E.find(data.get("uEmail")));
		}
		if(string.equalsIgnoreCase("/logout")){
			return VtoV.ObjectToJsonString(RestLoginPort_E.logout(data.get("uEmail"), data.get("uToken")));
		}
		if(string.equalsIgnoreCase("/register")){
			return VtoV.ObjectToJsonString(RestLoginPort_E.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("/change")){
			return VtoV.ObjectToJsonString(RestLoginPort_E.change(data.get("uEmail"), data.get("uChange")
					, data.get("uChangeEnsure"),data.get("uToken"), data.get("uPassword")));
		}
		//			if(string.equalsIgnoreCase("/checkStatus")){
		//				return VtoV.ObjectToJsonString(RestLoginPortImpl.checkStatus(data.get("token")));
```

```
//                        }
if(string.equalsIgnoreCase("/ask")){
        return VtoV.ObjectToJsonString(RestAskPort_E.ask(data.get("ip"), data.get("token")
                , data.get("message"), data.get("pointIp")));
}
if(string.equalsIgnoreCase("/feedBack")){
        return VtoV.ObjectToJsonString(RestAskPort_E.feedBack(data.get("ip"), data.get("token")
                , data.get("pointIp")));
}
if(string.equalsIgnoreCase("/getAskers")){
        return VtoV.ObjectToJsonString(RestAskPort_E.getAskers(data.get("token")));
}
if(string.equalsIgnoreCase("/dataWS")){
        return RestAskPort_E.dataWS(data.get("message"));
}
if(string.equalsIgnoreCase("/dataCX")){
        return RestAskPort_E.dataCX(data.get("message"));
}
if(string.equalsIgnoreCase("/dataCY")){
        return RestAskPort_E.dataCY(data.get("message"));
}
if(string.equalsIgnoreCase("/dataXL")){
        return RestAskPort_E.dataXL(data.get("message"));
}
if(string.equalsIgnoreCase("/dataRN")){
        return RestAskPort_E.dataRN(data.get("message"));
}
if(string.equalsIgnoreCase("/dataCG")){
        return RestAskPort_E.dataCG(data.get("message"));
}
if(string.equalsIgnoreCase("/dataCJ")){
        return RestAskPort_E.dataCJ(data.get("message"));
}
if(string.equalsIgnoreCase("/dataCL")){
        return RestAskPort_E.dataCL(data.get("message"));
}
if(string.equalsIgnoreCase("/dataXX")){
        return RestAskPort_E.dataXX(data.get("message"));
}
if(string.equalsIgnoreCase("/dataHF")){
        return RestAskPort_E.dataHF(data.get("message"));
}
if(string.equalsIgnoreCase("/dataCP")){
        return RestAskPort_E.dataCP(data.get("message"));
}
if(string.equalsIgnoreCase("/dataZF")){
        return RestAskPort_E.dataZF(data.get("message"));
}
```

```
		if(string.equalsIgnoreCase("/dataZY")){
			return RestAskPort_E.dataZY(data.get("message"));
		}
		if(string.equalsIgnoreCase("/dataXY")){
			return RestAskPort_E.dataXY(data.get("message"));
		}
		if(string.equalsIgnoreCase("/dataYT")){
			return RestAskPort_E.dataYT(data.get("message"));
		}
		if(string.equalsIgnoreCase("/dataZT")){
			return RestAskPort_E.dataZT(data.get("message"));
		}
		if(string.equalsIgnoreCase("/dataXT")){
			return RestAskPort_E.dataXT(data.get("message"));
		}
		return "";
	}

	public static String getCode(String filePath) throws IOException{
		if(filePath.contains(".html")||filePath.contains(".js")){
			return "UTF-8";
		}
		return "GBK";
	}

	public static String getFilePath(String string) {
		String root = new File("src/main/resources/static").getAbsolutePath().replace("\\", "/");
		if(null==string||string.equalsIgnoreCase("")||string.equalsIgnoreCase("/")){
			string = "/index.html";
		}
		return root + string;
	}
}
```

```
package OSI.AOP.MS.VPC.server;
import java.io.IOException;
import java.net.Socket;
import OSI.AOP.VPC.rest.VPCYangliaojing;
import SVQ.stable.StableWeb;
//import OSI.AOP.VPC.PP.port_E.RestLoginPort_E;
//import PEU.P.map.VtoV;
//合并 数据库和 前端的 vison文件
//罗瑶光 20210730
public class ServerForward_Standard {
	public static void main(String[] args){
	}
```

```
public static void IQ_ForwardType(Socket socket) {
}

public static void forwardToRestMap(Socket socket) {
}

public static void IQ_ForwardType(VPCSRequest vPCSRequest
                , VPCSResponse vPCSResponse) throws IOException {
        if(vPCSRequest.getRequestIsRest()){
                String filePath = VPCYangliaojing.getFilePath(vPCSRequest.getRequestLink());
                if(filePath.contains(StableWeb.FILE_TTF)
                                ||filePath.contains(StableWeb.FILE_EOT)
                                ||filePath.contains(StableWeb.FILE_SVG)
                                ||filePath.contains(StableWeb.FILE_WOFF)
                                ||filePath.contains(StableWeb.FILE_WOFF2)
                                ||filePath.contains(StableWeb.FILE_OTF)){
                        String code = VPCYangliaojing.getCode(filePath);
                        vPCSRequest.I_RequestFilePath(filePath);
                        vPCSRequest.I_RequestFileCode(code);
                        vPCSRequest.I_RequestForwardType(StableWeb.STREAM_BUFFER);
                        vPCSResponse.I_ResponseContentType(StableWeb.HEADER_CONTENT_TYPE_FONT_WOFF);
                        return;
                }
                vPCSRequest.I_RequestForwardType(StableWeb.STREAM_REST);
        }else{
                String filePath = VPCYangliaojing.getFilePath(vPCSRequest.getRequestLink());
                String code = VPCYangliaojing.getCode(filePath);
                vPCSRequest.I_RequestFilePath(filePath);
                vPCSRequest.I_RequestFileCode(code);
                if(filePath.contains(StableWeb.FILE_PNG)){
                        vPCSRequest.I_RequestForwardType(StableWeb.STREAM_BYTES);
                        vPCSResponse.I_ResponseContentType(StableWeb.HEADER_CONTENT_TYPE_PNG);
                }
                if(filePath.contains(StableWeb.FILE_JPEG)){
                        vPCSRequest.I_RequestForwardType(StableWeb.STREAM_BYTES);
                        vPCSResponse.I_ResponseContentType(StableWeb.HEADER_CONTENT_TYPE_JPEG);
                }
                if(filePath.contains(StableWeb.FILE_JPG)){
                        vPCSRequest.I_RequestForwardType(StableWeb.STREAM_BYTES);
                        vPCSResponse.I_ResponseContentType(StableWeb.HEADER_CONTENT_TYPE_JPG);
                }
                if(filePath.contains(StableWeb.FILE_GIF)){
                        vPCSRequest.I_RequestForwardType(StableWeb.STREAM_BYTES);
                        vPCSResponse.I_ResponseContentType(StableWeb.HEADER_CONTENT_TYPE_GIF);
                }
                if(filePath.contains(StableWeb.FILE_JS) && code.equalsIgnoreCase(StableWeb.CHARSET_UTF_8)){
                        vPCSRequest.I_RequestForwardType(StableWeb.STREAM_BYTES_BUFFER);
                        vPCSResponse.I_ResponseContentType(StableWeb.HEADER_CONTENT_TYPE_JS);
```

```
                }
                if(filePath.contains(StableWeb.FILE_CSS)){
                        vPCSRequest.I_RequestForwardType(StableWeb.STREAM_BUFFER);
                        vPCSResponse.I_ResponseContentType(StableWeb.HEADER_CONTENT_TYPE_CSS);
                }
                if(filePath.contains(StableWeb.FILE_HTML)){
                        vPCSRequest.I_RequestForwardType(StableWeb.STREAM_BUFFER);
                        vPCSResponse.I_ResponseContentType(StableWeb.HEADER_CONTENT_TYPE_HTML);
                }
                if(filePath.contains(StableWeb.FILE_PDF)){
                        vPCSRequest.I_RequestForwardType(StableWeb.STREAM_BYTES_WITHOUT_ZIP);
                        vPCSResponse.I_ResponseContentType(StableWeb.HEADER_CONTENT_TYPE_PDF);
                }
                if(filePath.contains(StableWeb.FILE_ZIP)){
                        vPCSRequest.I_RequestForwardType(StableWeb.STREAM_BYTES_WITHOUT_ZIP);
                        vPCSResponse.I_ResponseContentType(StableWeb.HEADER_CONTENT_TYPE_ZIP);
                }
                if(filePath.contains(StableWeb.FILE_WAV)){
                        vPCSRequest.I_RequestForwardType(StableWeb.STREAM_BUFFER);
                        vPCSResponse.I_ResponseContentType(StableWeb.HEADER_CONTENT_TYPE_WAV);
                }
                        if(filePath.contains(StableWeb.FILE_RAR)){
                        vPCSRequest.I_RequestForwardType(StableWeb.STREAM_BYTES_WITHOUT_ZIP);
                        vPCSResponse.I_ResponseContentType(StableWeb.HEADER_CONTENT_TYPE_RAR);
                }
                if(filePath.contains(StableWeb.FILE_XML)){
                        vPCSRequest.I_RequestForwardType(StableWeb.STREAM_BYTES_WITHOUT_ZIP);
                        vPCSResponse.I_ResponseContentType(StableWeb.HEADER_CONTENT_TYPE_XML);
                }
                if(filePath.contains(StableWeb.FILE_TXT)){
                        vPCSRequest.I_RequestForwardType(StableWeb.STREAM_BYTES_WITHOUT_ZIP);
                        vPCSResponse.I_ResponseContentType(StableWeb.HEADER_CONTENT_TYPE_TXT);
                }
        }
 }

//      public static void forwardToRestMap(App app, VPCSRequest vPCSRequest
//                      , VPCSResponse vPCSResponse) throws Exception {
//              if(null== vPCSRequest || null== vPCSRequest.getRequestForwardType()){
//                      vPCSResponse.return404();
//                      return;
//              }
//              if(vPCSRequest.getRequestForwardType().equalsIgnoreCase(StableWeb.STREAM_REST)){
//                      //与其设计个sleeper继承，不如在这里设计个condition函数， 然后根据port识别 来继承这个condition 就是
了。， 罗瑶光 20210730
//                      if(vPCSRequest.getRequestPort().equals(StableHTTP.PORT_DATABASE)) {
//                              RestMap_V.P_RestDB(app, vPCSRequest, vPCSResponse);
//                      }else {
```

```
//                                      RestMap_V.P_Rest(app, vPCSRequest, vPCSResponse);
//                              }
//                      }
//                      if(vPCSRequest.getRequestForwardType().equalsIgnoreCase(StableWeb.STREAM_BYTES)){
//                              RestMap_V.P_Bytes(vPCSRequest, vPCSResponse);
//                      }
//                      if(vPCSRequest.getRequestForwardType().equalsIgnoreCase(StableWeb.STREAM_BUFFER)){
//                              RestMap_V.P_Buffer(vPCSRequest, vPCSResponse);
//                      }
//                      if(vPCSRequest.getRequestForwardType().equalsIgnoreCase(StableWeb.STREAM_BYTES_BUFFER)){
//                              RestMap_V.P_BufferBytes(vPCSRequest, vPCSResponse);
//                      }
//                      if(vPCSRequest.getRequestForwardType().equalsIgnoreCase(StableWeb.STREAM_BYTES_WITHOUT_ZIP)){
//                              RestMap_V.P_BytesWithoutZip(vPCSRequest, vPCSResponse);
//                      }
//              }

 public static void forwardToRestMap(VPCSRequest vPCSRequest, VPCSResponse vPCSResponse) throws Exception {
              if(vPCSRequest.getRequestForwardType() == null){
                      vPCSResponse.return404();
                      return;
              }
              if(vPCSRequest.getRequestForwardType().equalsIgnoreCase(StableWeb.STREAM_REST)){
                      ServerRestMap_Standard.P_Rest(vPCSRequest, vPCSResponse);
              }
              if(vPCSRequest.getRequestForwardType().equalsIgnoreCase(StableWeb.STREAM_BYTES)){
                      ServerRestMap_Standard.P_Bytes(vPCSRequest, vPCSResponse);
              }
              if(vPCSRequest.getRequestForwardType().equalsIgnoreCase(StableWeb.STREAM_BUFFER)){
                      ServerRestMap_Standard.P_Buffer(vPCSRequest, vPCSResponse);
              }
              if(vPCSRequest.getRequestForwardType().equalsIgnoreCase(StableWeb.STREAM_BYTES_BUFFER)){
                      ServerRestMap_Standard.P_BufferBytes(vPCSRequest, vPCSResponse);
              }
              if(vPCSRequest.getRequestForwardType().equalsIgnoreCase(StableWeb.STREAM_BYTES_WITHOUT_ZIP)){
                      ServerRestMap_Standard.P_BytesWithoutZip(vPCSRequest, vPCSResponse);
              }
 }
}
```