package SEM.bloom;

import java.util.HashMap;

import java.util.Iterator;

import java.util.Map;

//用来索引24组花

//罗瑶光

//1 稍后细化 元基花接口

//2 稍后将接口统一用 function， class，元基组 3层map

//3 function 统一走 interface接口。

public class StaticRootMap{

public static Map<String, StaticClassMap> staticRootMap;

public void initMap() throws Exception {

staticRootMap= new HashMap<>();

StaticClassMap map\_A\_VECS= new StaticClassMap("A\_VECS");

StaticClassMap map\_A\_IDUQ= new StaticClassMap("A\_IDUQ");

StaticClassMap map\_O\_VECS= new StaticClassMap("O\_VECS");

StaticClassMap map\_O\_IDUQ= new StaticClassMap("O\_IDUQ");

StaticClassMap map\_P\_VECS= new StaticClassMap("P\_VECS");

StaticClassMap map\_P\_IDUQ= new StaticClassMap("P\_IDUQ");

StaticClassMap map\_M\_VECS= new StaticClassMap("M\_VECS");

StaticClassMap map\_M\_IDUQ= new StaticClassMap("M\_IDUQ");

StaticClassMap map\_V\_AOPM= new StaticClassMap("V\_AOPM");

StaticClassMap map\_V\_IDUQ= new StaticClassMap("V\_IDUQ");

StaticClassMap map\_E\_AOPM= new StaticClassMap("E\_AOPM");

StaticClassMap map\_E\_IDUQ= new StaticClassMap("E\_IDUQ");

StaticClassMap map\_C\_AOPM= new StaticClassMap("C\_AOPM");

StaticClassMap map\_C\_IDUQ= new StaticClassMap("C\_IDUQ");

StaticClassMap map\_S\_AOPM= new StaticClassMap("S\_AOPM");

StaticClassMap map\_S\_IDUQ= new StaticClassMap("S\_IDUQ");

StaticClassMap map\_I\_AOPM= new StaticClassMap("I\_AOPM");

StaticClassMap map\_I\_VECS= new StaticClassMap("I\_VECS");

StaticClassMap map\_D\_AOPM= new StaticClassMap("D\_AOPM");

StaticClassMap map\_D\_VECS= new StaticClassMap("D\_VECS");

StaticClassMap map\_U\_AOPM= new StaticClassMap("U\_AOPM");

StaticClassMap map\_U\_VECS= new StaticClassMap("U\_VECS");

StaticClassMap map\_Q\_AOPM= new StaticClassMap("Q\_AOPM");

StaticClassMap map\_Q\_VECS= new StaticClassMap("Q\_VECS");

//

//A

staticRootMap.put("A\_VECS", map\_A\_VECS);

staticRootMap.put("A\_IDUQ", map\_A\_IDUQ);

//O

staticRootMap.put("O\_VECS", map\_O\_VECS);

staticRootMap.put("O\_IDUQ", map\_O\_IDUQ);

//P

staticRootMap.put("P\_VECS", map\_P\_VECS);

staticRootMap.put("P\_IDUQ", map\_P\_IDUQ);

//M

staticRootMap.put("M\_VECS", map\_M\_VECS);

staticRootMap.put("M\_IDUQ", map\_M\_IDUQ);

//V

staticRootMap.put("V\_AOPM", map\_V\_AOPM);

staticRootMap.put("V\_IDUQ", map\_V\_IDUQ);

//E

staticRootMap.put("E\_AOPM", map\_E\_AOPM);

staticRootMap.put("E\_IDUQ", map\_E\_IDUQ);

//C

staticRootMap.put("C\_AOPM", map\_C\_AOPM);

staticRootMap.put("C\_IDUQ", map\_C\_IDUQ);

//S

staticRootMap.put("S\_AOPM", map\_S\_AOPM);

staticRootMap.put("S\_IDUQ", map\_S\_IDUQ);

//I

staticRootMap.put("I\_VECS", map\_I\_VECS);

staticRootMap.put("I\_AOPM", map\_I\_AOPM);

//D

staticRootMap.put("D\_VECS", map\_D\_VECS);

staticRootMap.put("D\_AOPM", map\_D\_AOPM);

//U

staticRootMap.put("U\_VECS", map\_U\_VECS);

staticRootMap.put("U\_AOPM", map\_U\_AOPM);

//Q

staticRootMap.put("Q\_VECS", map\_Q\_VECS);

staticRootMap.put("Q\_AOPM", map\_Q\_AOPM);

}

@SuppressWarnings("static-access")

public static void tinShellV003(String[] shellCommands, Map<String, Object> shellOutput) throws Exception{

//稍后准备把 下面main的测试代码 进行封装 调通 一句执行命令， 多句执行命令，多句并发执行命令。

//然后并入tinshell。像shell replace命令那样。

//罗瑶光

String[] strings= shellCommands;

Map<String, Object> output= shellOutput;

//开始设计传参。

StaticRootMap staticRootMap= new StaticRootMap();

staticRootMap.initMap();

for(String string:strings) {

Iterator<String> iterator= staticRootMap.staticRootMap.keySet().iterator();

while(iterator.hasNext()) {

String callMapKey= iterator.next();

//case 染色体接口

if(string.contains(callMapKey)) {

if(callMapKey.equalsIgnoreCase("U\_VECS")) {

doU\_VECS\_Case(staticRootMap.staticRootMap, string, output);

}

if(callMapKey.equalsIgnoreCase("U\_AOPM")) {

doU\_AOPM\_Case(staticRootMap.staticRootMap, string, output);//稍后分出去

}

if(callMapKey.equalsIgnoreCase("A\_VECS")) {

doA\_VECS\_Case(staticRootMap.staticRootMap, string, output);

}

if(callMapKey.equalsIgnoreCase("A\_IDUQ")) {

doA\_IDUQ\_Case(staticRootMap.staticRootMap, string, output);

}

if(callMapKey.equalsIgnoreCase("O\_VECS")) {

doO\_VECS\_Case(staticRootMap.staticRootMap, string, output);

}

if(callMapKey.equalsIgnoreCase("O\_IDUQ")) {

doO\_IDUQ\_Case(staticRootMap.staticRootMap, string, output);

}

if(callMapKey.equalsIgnoreCase("P\_VECS")) {

doP\_VECS\_Case(staticRootMap.staticRootMap, string, output);

}

if(callMapKey.equalsIgnoreCase("P\_IDUQ")) {

doP\_IDUQ\_Case(staticRootMap.staticRootMap, string, output);

}

if(callMapKey.equalsIgnoreCase("M\_VECS")) {

doM\_VECS\_Case(staticRootMap.staticRootMap, string, output);

}

if(callMapKey.equalsIgnoreCase("M\_IDUQ")) {

doM\_IDUQ\_Case(staticRootMap.staticRootMap, string, output);

}

if(callMapKey.equalsIgnoreCase("V\_AOPM")) {

doV\_AOPM\_Case(staticRootMap.staticRootMap, string, output);

}

if(callMapKey.equalsIgnoreCase("V\_IDUQ")) {

doV\_IDUQ\_Case(staticRootMap.staticRootMap, string, output);

}

if(callMapKey.equalsIgnoreCase("E\_AOPM")) {

doE\_AOPM\_Case(staticRootMap.staticRootMap, string, output);

}

if(callMapKey.equalsIgnoreCase("E\_IDUQ")) {

doE\_IDUQ\_Case(staticRootMap.staticRootMap, string, output);

}

if(callMapKey.equalsIgnoreCase("C\_AOPM")) {

doC\_AOPM\_Case(staticRootMap.staticRootMap, string, output);

}

if(callMapKey.equalsIgnoreCase("C\_IDUQ")) {

doC\_IDUQ\_Case(staticRootMap.staticRootMap, string, output);

}

if(callMapKey.equalsIgnoreCase("S\_AOPM")) {

doS\_AOPM\_Case(staticRootMap.staticRootMap, string, output);

}

if(callMapKey.equalsIgnoreCase("S\_IDUQ")) {

doS\_IDUQ\_Case(staticRootMap.staticRootMap, string, output);

}

if(callMapKey.equalsIgnoreCase("I\_AOPM")) {

doI\_AOPM\_Case(staticRootMap.staticRootMap, string, output);

}

if(callMapKey.equalsIgnoreCase("I\_VECS")) {

doI\_VECS\_Case(staticRootMap.staticRootMap, string, output);

}

if(callMapKey.equalsIgnoreCase("D\_AOPM")) {

doD\_AOPM\_Case(staticRootMap.staticRootMap, string, output);

}

if(callMapKey.equalsIgnoreCase("D\_VECS")) {

doD\_VECS\_Case(staticRootMap.staticRootMap, string, output);

}

if(callMapKey.equalsIgnoreCase("Q\_AOPM")) {

doQ\_AOPM\_Case(staticRootMap.staticRootMap, string, output);

}

if(callMapKey.equalsIgnoreCase("Q\_VECS")) {

doQ\_VECS\_Case(staticRootMap.staticRootMap, string, output);

}

}

}

//写法 3

}

}

public static void main(String[] argv) throws Exception {

//写法 1

//StaticRootMap staticRootMap= new StaticRootMap();

//staticRootMap.initMap();

//StaticClassMap staticClassMap= staticRootMap.staticRootMap.get("U\_VECS");

//StaticFunctionMapU\_VECS\_E staticFunctionMapU\_VECS\_C

//= (StaticFunctionMapU\_VECS\_E) staticClassMap.staticClassMap.get("U\_VECS");

//staticFunctionMapU\_VECS\_C.main(null);

//写法 2

String[] strings= new String[3];

strings[0]= "执行 U\_VECS 下 main 接口, 参数是null";

//strings[1]= "执行 I\_VECS 下 main 接口, 参数是null";

strings[1]= "执行 U\_VECS 下 main 接口, 参数是null";

//

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

String[] 传参因子= new String[2];

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

double[] doubles= new double[5];

doubles[0]= 2.2222262;

doubles[1]= 3.2226222;

doubles[2]= 6.2622222;

doubles[3]= 4.6226222;

doubles[4]= 1.2222226;

double dou= 2.22;

传参因子[0]= "input";//像神一样的tin god

传参因子[1]= "rank";

inputValue.put(传参因子[0], doubles);

inputValue.put(传参因子[1], dou);

output.put("传参因子", 传参因子);

output.put("inputValues", inputValue);

strings[2]= "执行 U\_AOPM 下 min\_v 接口, 参数是 传参因子";

//...

StaticRootMap.tinShellV003(strings, output);

//写法 3

}

@SuppressWarnings("static-access")

private static void doA\_VECS\_Case(Map<String, StaticClassMap> staticRootMap, String string, Map<String, Object> output)

throws Exception {

StaticClassMap staticClassMap= staticRootMap.get("A\_VECS");

StaticFunctionMapA\_VECS\_E staticFunctionMapA\_VECS\_C

= (StaticFunctionMapA\_VECS\_E) staticClassMap.staticClassMap.get("A\_VECS");

//case 函数名接口

Iterator<String> callFunction= staticFunctionMapA\_VECS\_C.annotationMap.keySet().iterator();

while(callFunction.hasNext()) {

String callFunctionKey= callFunction.next();

if(string.contains(callFunctionKey)) {

StaticFunctionMap.doA\_VECS\_CaseFunction(callFunctionKey, string, staticFunctionMapA\_VECS\_C, output);

}

}

}

@SuppressWarnings("static-access")

private static void doP\_VECS\_Case(Map<String, StaticClassMap> staticRootMap, String string, Map<String, Object> output)

throws Exception {

StaticClassMap staticClassMap= staticRootMap.get("P\_VECS");

StaticFunctionMapP\_VECS\_E staticFunctionMapP\_VECS\_C

= (StaticFunctionMapP\_VECS\_E) staticClassMap.staticClassMap.get("P\_VECS");

//case 函数名接口

Iterator<String> callFunction= staticFunctionMapP\_VECS\_C.annotationMap.keySet().iterator();

while(callFunction.hasNext()) {

String callFunctionKey= callFunction.next();

if(string.contains(callFunctionKey)) {

StaticFunctionMap.doP\_VECS\_CaseFunction(callFunctionKey, string, staticFunctionMapP\_VECS\_C, output);

}

}

}

@SuppressWarnings("static-access")

private static void doO\_IDUQ\_Case(Map<String, StaticClassMap> staticRootMap, String string, Map<String, Object> output)

throws Exception {

StaticClassMap staticClassMap= staticRootMap.get("O\_IDUQ");

StaticFunctionMapO\_IDUQ\_E staticFunctionMapO\_IDUQ\_C

= (StaticFunctionMapO\_IDUQ\_E) staticClassMap.staticClassMap.get("O\_IDUQ");

//case 函数名接口

Iterator<String> callFunction= staticFunctionMapO\_IDUQ\_C.annotationMap.keySet().iterator();

while(callFunction.hasNext()) {

String callFunctionKey= callFunction.next();

if(string.contains(callFunctionKey)) {

StaticFunctionMap.doO\_IDUQ\_CaseFunction(callFunctionKey, string, staticFunctionMapO\_IDUQ\_C, output);

}

}

}

@SuppressWarnings("static-access")

private static void doO\_VECS\_Case(Map<String, StaticClassMap> staticRootMap, String string, Map<String, Object> output)

throws Exception {

StaticClassMap staticClassMap= staticRootMap.get("O\_VECS");

StaticFunctionMapO\_VECS\_E staticFunctionMapO\_VECS\_C

= (StaticFunctionMapO\_VECS\_E) staticClassMap.staticClassMap.get("O\_VECS");

//case 函数名接口

Iterator<String> callFunction= staticFunctionMapO\_VECS\_C.annotationMap.keySet().iterator();

while(callFunction.hasNext()) {

String callFunctionKey= callFunction.next();

if(string.contains(callFunctionKey)) {

StaticFunctionMap.doO\_VECS\_CaseFunction(callFunctionKey, string, staticFunctionMapO\_VECS\_C, output);

}

}

}

@SuppressWarnings("static-access")

private static void doA\_IDUQ\_Case(Map<String, StaticClassMap> staticRootMap, String string, Map<String, Object> output)

throws Exception {

StaticClassMap staticClassMap= staticRootMap.get("A\_IDUQ");

StaticFunctionMapA\_IDUQ\_E staticFunctionMapA\_IDUQ\_C

= (StaticFunctionMapA\_IDUQ\_E) staticClassMap.staticClassMap.get("A\_IDUQ");

//case 函数名接口

Iterator<String> callFunction= staticFunctionMapA\_IDUQ\_C.annotationMap.keySet().iterator();

while(callFunction.hasNext()) {

String callFunctionKey= callFunction.next();

if(string.contains(callFunctionKey)) {

StaticFunctionMap.doA\_IDUQ\_CaseFunction(callFunctionKey, string, staticFunctionMapA\_IDUQ\_C, output);

}

}

}

@SuppressWarnings("static-access")

private static void doU\_VECS\_Case(Map<String, StaticClassMap> staticRootMap, String string, Map<String, Object> output)

throws Exception {

StaticClassMap staticClassMap= staticRootMap.get("U\_VECS");

StaticFunctionMapU\_VECS\_E staticFunctionMapU\_VECS\_C

= (StaticFunctionMapU\_VECS\_E) staticClassMap.staticClassMap.get("U\_VECS");

//case 函数名接口

Iterator<String> callFunction= staticFunctionMapU\_VECS\_C.annotationMap.keySet().iterator();

while(callFunction.hasNext()) {

String callFunctionKey= callFunction.next();

if(string.contains(callFunctionKey)) {

StaticFunctionMap.doU\_VECS\_CaseFunction(callFunctionKey, string, staticFunctionMapU\_VECS\_C, output);

}

}

}

@SuppressWarnings("static-access")

private static void doP\_IDUQ\_Case(Map<String, StaticClassMap> staticRootMap, String string, Map<String, Object> output)

throws Exception {

StaticClassMap staticClassMap= staticRootMap.get("P\_IDUQ");

StaticFunctionMapP\_IDUQ\_E staticFunctionMapP\_IDUQ\_C

= (StaticFunctionMapP\_IDUQ\_E) staticClassMap.staticClassMap.get("P\_IDUQ");

//case 函数名接口

Iterator<String> callFunction= staticFunctionMapP\_IDUQ\_C.annotationMap.keySet().iterator();

while(callFunction.hasNext()) {

String callFunctionKey= callFunction.next();

if(string.contains(callFunctionKey)) {

StaticFunctionMap.doP\_IDUQ\_CaseFunction(callFunctionKey, string, staticFunctionMapP\_IDUQ\_C, output);

}

}

}

@SuppressWarnings("static-access")

private static void doM\_VECS\_Case(Map<String, StaticClassMap> staticRootMap, String string, Map<String, Object> output)

throws Exception {

StaticClassMap staticClassMap= staticRootMap.get("M\_VECS");

StaticFunctionMapM\_VECS\_E staticFunctionMapM\_VECS\_C

= (StaticFunctionMapM\_VECS\_E) staticClassMap.staticClassMap.get("M\_VECS");

//case 函数名接口

Iterator<String> callFunction= staticFunctionMapM\_VECS\_C.annotationMap.keySet().iterator();

while(callFunction.hasNext()) {

String callFunctionKey= callFunction.next();

if(string.contains(callFunctionKey)) {

StaticFunctionMap.doM\_VECS\_CaseFunction(callFunctionKey, string, staticFunctionMapM\_VECS\_C, output);

}

}

}

@SuppressWarnings("static-access")

private static void doM\_IDUQ\_Case(Map<String, StaticClassMap> staticRootMap, String string, Map<String, Object> output)

throws Exception {

StaticClassMap staticClassMap= staticRootMap.get("M\_IDUQ");

StaticFunctionMapM\_IDUQ\_E staticFunctionMapM\_IDUQ\_C

= (StaticFunctionMapM\_IDUQ\_E) staticClassMap.staticClassMap.get("M\_IDUQ");

//case 函数名接口

Iterator<String> callFunction= staticFunctionMapM\_IDUQ\_C.annotationMap.keySet().iterator();

while(callFunction.hasNext()) {

String callFunctionKey= callFunction.next();

if(string.contains(callFunctionKey)) {

StaticFunctionMap.doM\_IDUQ\_CaseFunction(callFunctionKey, string, staticFunctionMapM\_IDUQ\_C, output);

}

}

}

@SuppressWarnings("static-access")

private static void doV\_AOPM\_Case(Map<String, StaticClassMap> staticRootMap, String string, Map<String, Object> output)

throws Exception {

StaticClassMap staticClassMap= staticRootMap.get("V\_AOPM");

StaticFunctionMapV\_AOPM\_E staticFunctionMapV\_AOPM\_C

= (StaticFunctionMapV\_AOPM\_E) staticClassMap.staticClassMap.get("V\_AOPM");

//case 函数名接口

Iterator<String> callFunction= staticFunctionMapV\_AOPM\_C.annotationMap.keySet().iterator();

while(callFunction.hasNext()) {

String callFunctionKey= callFunction.next();

if(string.contains(callFunctionKey)) {

StaticFunctionMap.doV\_AOPM\_CaseFunction(callFunctionKey, string, staticFunctionMapV\_AOPM\_C, output);

}

}

}

@SuppressWarnings("static-access")

private static void doV\_IDUQ\_Case(Map<String, StaticClassMap> staticRootMap, String string, Map<String, Object> output)

throws Exception {

StaticClassMap staticClassMap= staticRootMap.get("V\_IDUQ");

StaticFunctionMapV\_IDUQ\_E staticFunctionMapV\_IDUQ\_C

= (StaticFunctionMapV\_IDUQ\_E) staticClassMap.staticClassMap.get("V\_IDUQ");

//case 函数名接口

Iterator<String> callFunction= staticFunctionMapV\_IDUQ\_C.annotationMap.keySet().iterator();

while(callFunction.hasNext()) {

String callFunctionKey= callFunction.next();

if(string.contains(callFunctionKey)) {

StaticFunctionMap.doV\_IDUQ\_CaseFunction(callFunctionKey, string, staticFunctionMapV\_IDUQ\_C, output);

}

}

}

@SuppressWarnings("static-access")

private static void doE\_AOPM\_Case(Map<String, StaticClassMap> staticRootMap, String string, Map<String, Object> output)

throws Exception {

StaticClassMap staticClassMap= staticRootMap.get("E\_AOPM");

StaticFunctionMapE\_AOPM\_E staticFunctionMapE\_AOPM\_C

= (StaticFunctionMapE\_AOPM\_E) staticClassMap.staticClassMap.get("E\_AOPM");

//case 函数名接口

Iterator<String> callFunction= staticFunctionMapE\_AOPM\_C.annotationMap.keySet().iterator();

while(callFunction.hasNext()) {

String callFunctionKey= callFunction.next();

if(string.contains(callFunctionKey)) {

StaticFunctionMap.doE\_AOPM\_CaseFunction(callFunctionKey, string, staticFunctionMapE\_AOPM\_C, output);

}

}

}

@SuppressWarnings("static-access")

private static void doE\_IDUQ\_Case(Map<String, StaticClassMap> staticRootMap, String string, Map<String, Object> output)

throws Exception {

StaticClassMap staticClassMap= staticRootMap.get("E\_IDUQ");

StaticFunctionMapE\_IDUQ\_E staticFunctionMapE\_IDUQ\_C

= (StaticFunctionMapE\_IDUQ\_E) staticClassMap.staticClassMap.get("E\_IDUQ");

//case 函数名接口

Iterator<String> callFunction= staticFunctionMapE\_IDUQ\_C.annotationMap.keySet().iterator();

while(callFunction.hasNext()) {

String callFunctionKey= callFunction.next();

if(string.contains(callFunctionKey)) {

StaticFunctionMap.doE\_IDUQ\_CaseFunction(callFunctionKey, string, staticFunctionMapE\_IDUQ\_C, output);

}

}

}

@SuppressWarnings("static-access")

private static void doC\_AOPM\_Case(Map<String, StaticClassMap> staticRootMap, String string, Map<String, Object> output)

throws Exception {

StaticClassMap staticClassMap= staticRootMap.get("C\_AOPM");

StaticFunctionMapC\_AOPM\_E staticFunctionMapC\_AOPM\_C

= (StaticFunctionMapC\_AOPM\_E) staticClassMap.staticClassMap.get("C\_AOPM");

//case 函数名接口

Iterator<String> callFunction= staticFunctionMapC\_AOPM\_C.annotationMap.keySet().iterator();

while(callFunction.hasNext()) {

String callFunctionKey= callFunction.next();

if(string.contains(callFunctionKey)) {

StaticFunctionMap.doC\_AOPM\_CaseFunction(callFunctionKey, string, staticFunctionMapC\_AOPM\_C, output);

}

}

}

@SuppressWarnings("static-access")

private static void doC\_IDUQ\_Case(Map<String, StaticClassMap> staticRootMap, String string, Map<String, Object> output)

throws Exception {

StaticClassMap staticClassMap= staticRootMap.get("C\_IDUQ");

StaticFunctionMapC\_IDUQ\_E staticFunctionMapC\_IDUQ\_C

= (StaticFunctionMapC\_IDUQ\_E) staticClassMap.staticClassMap.get("C\_IDUQ");

//case 函数名接口

Iterator<String> callFunction= staticFunctionMapC\_IDUQ\_C.annotationMap.keySet().iterator();

while(callFunction.hasNext()) {

String callFunctionKey= callFunction.next();

if(string.contains(callFunctionKey)) {

StaticFunctionMap.doC\_IDUQ\_CaseFunction(callFunctionKey, string, staticFunctionMapC\_IDUQ\_C, output);

}

}

}

@SuppressWarnings("static-access")

private static void doS\_AOPM\_Case(Map<String, StaticClassMap> staticRootMap, String string, Map<String, Object> output)

throws Exception {

StaticClassMap staticClassMap= staticRootMap.get("S\_AOPM");

StaticFunctionMapS\_AOPM\_E staticFunctionMapS\_AOPM\_C

= (StaticFunctionMapS\_AOPM\_E) staticClassMap.staticClassMap.get("S\_AOPM");

//case 函数名接口

Iterator<String> callFunction= staticFunctionMapS\_AOPM\_C.annotationMap.keySet().iterator();

while(callFunction.hasNext()) {

String callFunctionKey= callFunction.next();

if(string.contains(callFunctionKey)) {

StaticFunctionMap.doS\_AOPM\_CaseFunction(callFunctionKey, string, staticFunctionMapS\_AOPM\_C, output);

}

}

}

@SuppressWarnings("static-access")

private static void doS\_IDUQ\_Case(Map<String, StaticClassMap> staticRootMap, String string, Map<String, Object> output)

throws Exception {

StaticClassMap staticClassMap= staticRootMap.get("S\_IDUQ");

StaticFunctionMapS\_IDUQ\_E staticFunctionMapS\_IDUQ\_C

= (StaticFunctionMapS\_IDUQ\_E) staticClassMap.staticClassMap.get("S\_IDUQ");

//case 函数名接口

Iterator<String> callFunction= staticFunctionMapS\_IDUQ\_C.annotationMap.keySet().iterator();

while(callFunction.hasNext()) {

String callFunctionKey= callFunction.next();

if(string.contains(callFunctionKey)) {

StaticFunctionMap.doS\_IDUQ\_CaseFunction(callFunctionKey, string, staticFunctionMapS\_IDUQ\_C, output);

}

}

}

@SuppressWarnings("static-access")

private static void doI\_AOPM\_Case(Map<String, StaticClassMap> staticRootMap, String string, Map<String, Object> output)

throws Exception {

StaticClassMap staticClassMap= staticRootMap.get("I\_AOPM");

StaticFunctionMapI\_AOPM\_E staticFunctionMapI\_AOPM\_C

= (StaticFunctionMapI\_AOPM\_E) staticClassMap.staticClassMap.get("I\_AOPM");

//case 函数名接口

Iterator<String> callFunction= staticFunctionMapI\_AOPM\_C.annotationMap.keySet().iterator();

while(callFunction.hasNext()) {

String callFunctionKey= callFunction.next();

if(string.contains(callFunctionKey)) {

StaticFunctionMap.doI\_AOPM\_CaseFunction(callFunctionKey, string, staticFunctionMapI\_AOPM\_C, output);

}

}

}

@SuppressWarnings("static-access")

private static void doI\_VECS\_Case(Map<String, StaticClassMap> staticRootMap, String string, Map<String, Object> output)

throws Exception {

StaticClassMap staticClassMap= staticRootMap.get("I\_VECS");

StaticFunctionMapI\_VECS\_E staticFunctionMapI\_VECS\_C

= (StaticFunctionMapI\_VECS\_E) staticClassMap.staticClassMap.get("I\_VECS");

//case 函数名接口

Iterator<String> callFunction= staticFunctionMapI\_VECS\_C.annotationMap.keySet().iterator();

while(callFunction.hasNext()) {

String callFunctionKey= callFunction.next();

if(string.contains(callFunctionKey)) {

StaticFunctionMap.doI\_VECS\_CaseFunction(callFunctionKey, string, staticFunctionMapI\_VECS\_C, output);

}

}

}

@SuppressWarnings("static-access")

private static void doD\_AOPM\_Case(Map<String, StaticClassMap> staticRootMap, String string, Map<String, Object> output)

throws Exception {

StaticClassMap staticClassMap= staticRootMap.get("D\_AOPM");

StaticFunctionMapD\_AOPM\_E staticFunctionMapD\_AOPM\_C

= (StaticFunctionMapD\_AOPM\_E) staticClassMap.staticClassMap.get("D\_AOPM");

//case 函数名接口

Iterator<String> callFunction= staticFunctionMapD\_AOPM\_C.annotationMap.keySet().iterator();

while(callFunction.hasNext()) {

String callFunctionKey= callFunction.next();

if(string.contains(callFunctionKey)) {

StaticFunctionMap.doD\_AOPM\_CaseFunction(callFunctionKey, string, staticFunctionMapD\_AOPM\_C, output);

}

}

}

@SuppressWarnings("static-access")

private static void doD\_VECS\_Case(Map<String, StaticClassMap> staticRootMap, String string, Map<String, Object> output)

throws Exception {

StaticClassMap staticClassMap= staticRootMap.get("D\_VECS");

StaticFunctionMapD\_VECS\_E staticFunctionMapD\_VECS\_C

= (StaticFunctionMapD\_VECS\_E) staticClassMap.staticClassMap.get("D\_VECS");

//case 函数名接口

Iterator<String> callFunction= staticFunctionMapD\_VECS\_C.annotationMap.keySet().iterator();

while(callFunction.hasNext()) {

String callFunctionKey= callFunction.next();

if(string.contains(callFunctionKey)) {

StaticFunctionMap.doD\_VECS\_CaseFunction(callFunctionKey, string, staticFunctionMapD\_VECS\_C, output);

}

}

}

@SuppressWarnings("static-access")

private static void doQ\_AOPM\_Case(Map<String, StaticClassMap> staticRootMap, String string, Map<String, Object> output)

throws Exception {

StaticClassMap staticClassMap= staticRootMap.get("Q\_AOPM");

StaticFunctionMapQ\_AOPM\_E staticFunctionMapQ\_AOPM\_C

= (StaticFunctionMapQ\_AOPM\_E) staticClassMap.staticClassMap.get("Q\_AOPM");

//case 函数名接口

Iterator<String> callFunction= staticFunctionMapQ\_AOPM\_C.annotationMap.keySet().iterator();

while(callFunction.hasNext()) {

String callFunctionKey= callFunction.next();

if(string.contains(callFunctionKey)) {

StaticFunctionMap.doQ\_AOPM\_CaseFunction(callFunctionKey, string, staticFunctionMapQ\_AOPM\_C, output);

}

}

}

@SuppressWarnings("static-access")

private static void doQ\_VECS\_Case(Map<String, StaticClassMap> staticRootMap, String string, Map<String, Object> output)

throws Exception {

StaticClassMap staticClassMap= staticRootMap.get("Q\_VECS");

StaticFunctionMapQ\_VECS\_E staticFunctionMapQ\_VECS\_C

= (StaticFunctionMapQ\_VECS\_E) staticClassMap.staticClassMap.get("Q\_VECS");

//case 函数名接口

Iterator<String> callFunction= staticFunctionMapQ\_VECS\_C.annotationMap.keySet().iterator();

while(callFunction.hasNext()) {

String callFunctionKey= callFunction.next();

if(string.contains(callFunctionKey)) {

StaticFunctionMap.doQ\_VECS\_CaseFunction(callFunctionKey, string, staticFunctionMapQ\_VECS\_C, output);

}

}

}

@SuppressWarnings("static-access")

private static void doU\_AOPM\_Case(Map<String, StaticClassMap> staticRootMap, String string, Map<String, Object> output)

throws Exception {

StaticClassMap staticClassMap= staticRootMap.get("U\_AOPM");

StaticFunctionMapU\_AOPM\_E staticFunctionMapU\_AOPM\_C

= (StaticFunctionMapU\_AOPM\_E) staticClassMap.staticClassMap.get("U\_AOPM");

//case 函数名接口

Iterator<String> callFunction= staticFunctionMapU\_AOPM\_C.annotationMap.keySet().iterator();

while(callFunction.hasNext()) {

String callFunctionKey= callFunction.next();

if(string.contains(callFunctionKey)) {

StaticFunctionMap.doU\_AOPM\_CaseFunction(callFunctionKey, string, staticFunctionMapU\_AOPM\_C, output);

}

}

}

}

package SEM.bloom;

import java.io.IOException;

import java.util.HashMap;

//import java.util.ArrayList;

//import java.util.List;

import java.util.Map;

//用来索引文件。

//罗瑶光

//流程，1先工程归纳，然后2分类，最后3统计执行接口的相关java文件，4进行map索引。

public class StaticClassMap{

public static Map<String, Object> staticClassMap= new HashMap<>();

public StaticClassMap(String string) throws IOException {

if("A\_VECS".equals(string)) {

//..遍历工程文件。。。导入注册函数。

//分词 读心术 情感分析，教育分析，文本分析，视觉动画

//分词

StaticFunctionMapA\_VECS\_E staticFunctionMapA\_VECS\_E =new StaticFunctionMapA\_VECS\_E();

staticClassMap.put("A\_VECS", staticFunctionMapA\_VECS\_E);//CE接口热化

StaticFunctionMapA\_VECS\_C.load(staticFunctionMapA\_VECS\_E);//static 检查

}

if("A\_IDUQ".equals(string)) {

//..肽展公式， dna加密，vpcs服务器，

StaticFunctionMapA\_IDUQ\_E staticFunctionMapA\_IDUQ\_E =new StaticFunctionMapA\_IDUQ\_E();

staticClassMap.put("A\_IDUQ", staticFunctionMapA\_IDUQ\_E);//CE接口热化

StaticFunctionMapA\_IDUQ\_C.load(staticFunctionMapA\_IDUQ\_E);//static 检查

}

if("O\_VECS".equals(string)) {

//..tinshell， 中文发音，数据库语言，离散余弦变换，

StaticFunctionMapO\_VECS\_E staticFunctionMapO\_VECS\_E =new StaticFunctionMapO\_VECS\_E();

staticClassMap.put("O\_VECS", staticFunctionMapO\_VECS\_E);//CE接口热化

StaticFunctionMapO\_VECS\_C.load(staticFunctionMapO\_VECS\_E);//static 检查

}

if("O\_IDUQ".equals(string)) {

//..etl 文档流执行， 保存。

StaticFunctionMapO\_IDUQ\_E staticFunctionMapO\_IDUQ\_E =new StaticFunctionMapO\_IDUQ\_E();

staticClassMap.put("O\_IDUQ", staticFunctionMapO\_IDUQ\_E);//CE接口热化

StaticFunctionMapO\_IDUQ\_C.load(staticFunctionMapO\_IDUQ\_E);//static 检查

}

if("P\_VECS".equals(string)) {

//..数据预测完整包。

StaticFunctionMapP\_VECS\_E staticFunctionMapP\_VECS\_E =new StaticFunctionMapP\_VECS\_E();

staticClassMap.put("P\_VECS", staticFunctionMapP\_VECS\_E);//CE接口热化

StaticFunctionMapP\_VECS\_C.load(staticFunctionMapP\_VECS\_E);//static 检查

}

if("P\_IDUQ".equals(string)) {

//..dna遗传杂交组件

StaticFunctionMapP\_IDUQ\_E staticFunctionMapP\_IDUQ\_E =new StaticFunctionMapP\_IDUQ\_E();

staticClassMap.put("P\_IDUQ", staticFunctionMapP\_IDUQ\_E);//CE接口热化

StaticFunctionMapP\_IDUQ\_C.load(staticFunctionMapP\_IDUQ\_E);//static 检查

}

if("M\_VECS".equals(string)) {

//..数据库 增删改查。界面控件，打印插件，三维词花组件

StaticFunctionMapM\_VECS\_E staticFunctionMapM\_VECS\_E =new StaticFunctionMapM\_VECS\_E();

staticClassMap.put("M\_VECS", staticFunctionMapM\_VECS\_E);//CE接口热化

StaticFunctionMapM\_VECS\_C.load(staticFunctionMapM\_VECS\_E);//static 检查

}

if("M\_IDUQ".equals(string)) {

//..欧拉 元基进制环路， 元基进制变换。

StaticFunctionMapM\_IDUQ\_E staticFunctionMapM\_IDUQ\_E =new StaticFunctionMapM\_IDUQ\_E();

staticClassMap.put("M\_IDUQ", staticFunctionMapM\_IDUQ\_E);//CE接口热化

StaticFunctionMapM\_IDUQ\_C.load(staticFunctionMapM\_IDUQ\_E);//static 检查

}

if("V\_AOPM".equals(string)) {

//..dna 概率钥匙非对称变换加密， 缓存，三维视觉，

StaticFunctionMapV\_AOPM\_E staticFunctionMapV\_AOPM\_E =new StaticFunctionMapV\_AOPM\_E();

staticClassMap.put("V\_AOPM", staticFunctionMapV\_AOPM\_E);//CE接口热化

StaticFunctionMapV\_AOPM\_C.load(staticFunctionMapV\_AOPM\_E);//static 检查

}

if("V\_IDUQ".equals(string)) {

//..etl核心组件，界面，etl读取文档，

StaticFunctionMapV\_IDUQ\_E staticFunctionMapV\_IDUQ\_E =new StaticFunctionMapV\_IDUQ\_E();

staticClassMap.put("V\_IDUQ", staticFunctionMapV\_IDUQ\_E);//CE接口热化

StaticFunctionMapV\_IDUQ\_C.load(staticFunctionMapV\_IDUQ\_E);//static 检查

}

if("E\_AOPM".equals(string)) {

//..德塔分词核心组件。

StaticFunctionMapE\_AOPM\_E staticFunctionMapE\_AOPM\_E =new StaticFunctionMapE\_AOPM\_E();

staticClassMap.put("E\_AOPM", staticFunctionMapE\_AOPM\_E);//CE接口热化

StaticFunctionMapE\_AOPM\_C.load(staticFunctionMapE\_AOPM\_E);//static 检查

}

if("E\_IDUQ".equals(string)) {

//..etl 界面操作组件，类 osgi 插件注册组件

StaticFunctionMapE\_IDUQ\_E staticFunctionMapE\_IDUQ\_E= new StaticFunctionMapE\_IDUQ\_E();

staticClassMap.put("E\_IDUQ", staticFunctionMapE\_IDUQ\_E);//CE接口热化

StaticFunctionMapE\_IDUQ\_C.load(staticFunctionMapE\_IDUQ\_E);//static 检查

}

if("C\_AOPM".equals(string)) {

//..vpcs服务器中心， 自然语言处理组件

StaticFunctionMapC\_AOPM\_E staticFunctionMapC\_AOPM\_E= new StaticFunctionMapC\_AOPM\_E();

staticClassMap.put("C\_AOPM", staticFunctionMapC\_AOPM\_E);//CE接口热化

StaticFunctionMapC\_AOPM\_C.load(staticFunctionMapC\_AOPM\_E);//static 检查

}

if("C\_IDUQ".equals(string)) {

//..自然语言 控制中心

StaticFunctionMapC\_IDUQ\_E staticFunctionMapC\_IDUQ\_E= new StaticFunctionMapC\_IDUQ\_E();

staticClassMap.put("C\_IDUQ", staticFunctionMapC\_IDUQ\_E);//CE接口热化

StaticFunctionMapC\_IDUQ\_C.load(staticFunctionMapC\_IDUQ\_E);//static 检查

}

if("S\_AOPM".equals(string)) {

//..数据记录中心

StaticFunctionMapS\_AOPM\_E staticFunctionMapS\_AOPM\_E=new StaticFunctionMapS\_AOPM\_E();

staticClassMap.put("S\_AOPM", staticFunctionMapS\_AOPM\_E);//CE接口热化

StaticFunctionMapS\_AOPM\_C.load(staticFunctionMapS\_AOPM\_E);//static 检查

}

if("S\_IDUQ".equals(string)) {

//..线性，非线性数据操作中心

StaticFunctionMapS\_IDUQ\_E staticFunctionMapS\_IDUQ\_E=new StaticFunctionMapS\_IDUQ\_E();

staticClassMap.put("S\_IDUQ", staticFunctionMapS\_IDUQ\_E);//CE接口热化

StaticFunctionMapS\_IDUQ\_C.load(staticFunctionMapS\_IDUQ\_E);//static 检查

}

if("I\_AOPM".equals(string)) {

//..语音记录，三维数据分析登记

StaticFunctionMapI\_AOPM\_E staticFunctionMapI\_AOPM\_E= new StaticFunctionMapI\_AOPM\_E();

staticClassMap.put("I\_AOPM", staticFunctionMapI\_AOPM\_E);//CE接口热化

StaticFunctionMapI\_AOPM\_C.load(staticFunctionMapI\_AOPM\_E);//static 检查

}

if("I\_VECS".equals(string)) {

//..肽腐蚀， 非卷积视觉，图片读脏

StaticFunctionMapI\_VECS\_E staticFunctionMapI\_VECS\_E=new StaticFunctionMapI\_VECS\_E();

staticClassMap.put("I\_VECS", staticFunctionMapI\_VECS\_E);//CE接口热化

StaticFunctionMapI\_VECS\_C.load(staticFunctionMapI\_VECS\_E);//static 检查

}

if("D\_AOPM".equals(string)) {

//..数据异常处理，

StaticFunctionMapD\_AOPM\_E staticFunctionMapD\_AOPM\_E= new StaticFunctionMapD\_AOPM\_E();

staticClassMap.put("D\_AOPM",staticFunctionMapD\_AOPM\_E);//CE接口热化

StaticFunctionMapD\_AOPM\_C.load(staticFunctionMapD\_AOPM\_E);//static 检查

}

if("D\_VECS".equals(string)) {

//..数据异常处理，，稍后把功能是 删除的 移到这里来。

StaticFunctionMapD\_VECS\_E staticFunctionMapD\_VECS\_E= new StaticFunctionMapD\_VECS\_E();

staticClassMap.put("D\_VECS", staticFunctionMapD\_VECS\_E);//CE接口热化

StaticFunctionMapD\_VECS\_C.load(staticFunctionMapD\_VECS\_E);//static 检查

}

if("U\_AOPM".equals(string)) {

//..卷积变换处理

StaticFunctionMapU\_AOPM\_E staticFunctionMapU\_AOPM\_E= new StaticFunctionMapU\_AOPM\_E();

staticClassMap.put("U\_AOPM",staticFunctionMapU\_AOPM\_E);//CE接口热化

StaticFunctionMapU\_AOPM\_C.load(staticFunctionMapU\_AOPM\_E);//static 检查

}

if("U\_VECS".equals(string)) {

//..数据变换处理

StaticFunctionMapU\_VECS\_E staticFunctionMapU\_VECS\_E= new StaticFunctionMapU\_VECS\_E();

staticClassMap.put("U\_VECS", staticFunctionMapU\_VECS\_E);//CE接口热化

StaticFunctionMapU\_VECS\_C.load(staticFunctionMapU\_VECS\_E);//static 检查

//。。..继续注册。。

//。。

}

if("Q\_AOPM".equals(string)) {

//..六元dna杂交计算框架

StaticFunctionMapQ\_AOPM\_E staticFunctionMapQ\_AOPM\_E= new StaticFunctionMapQ\_AOPM\_E();

staticClassMap.put("Q\_AOPM",staticFunctionMapQ\_AOPM\_E);//CE接口热化

StaticFunctionMapQ\_AOPM\_C.load(staticFunctionMapQ\_AOPM\_E);//static 检查

}

if("Q\_VECS".equals(string)) {

//..dna搜索， 数据库orm， 函数语言

StaticFunctionMapQ\_VECS\_E staticFunctionMapQ\_VECS\_E= new StaticFunctionMapQ\_VECS\_E();

staticClassMap.put("Q\_VECS",staticFunctionMapQ\_VECS\_E);//CE接口热化

StaticFunctionMapQ\_VECS\_C.load(staticFunctionMapQ\_VECS\_E);//static 检查

}

// TODO Auto-generated constructor stub

}

}

package SEM.bloom;

import java.awt.HeadlessException;

import java.io.IOException;

import java.util.HashMap;

import java.util.Map;

import javax.sound.sampled.UnsupportedAudioFileException;

//用来索引函数 注册类

//罗瑶光

public class StaticFunctionMap{

public static void doA\_VECS\_CaseFunction(String callFunctionKey, String string

, StaticFunctionMapA\_VECS\_E staticFunctionMapA\_VECS\_C, Map<String, Object> output) throws IOException {

if(callFunctionKey.equalsIgnoreCase("main")) {

//....

}

StaticFunctionMapA\_VECS\_C.callFunction(callFunctionKey, staticFunctionMapA\_VECS\_C, output);

}

public static void doA\_IDUQ\_CaseFunction(String callFunctionKey, String string

, StaticFunctionMapA\_IDUQ\_E staticFunctionMapA\_IDUQ\_C, Map<String, Object> output) throws IOException {

StaticFunctionMapA\_IDUQ\_C.callFunction(callFunctionKey, staticFunctionMapA\_IDUQ\_C, output);

}

public static void doO\_VECS\_CaseFunction(String callFunctionKey, String string

, StaticFunctionMapO\_VECS\_E staticFunctionMapO\_VECS\_C, Map<String, Object> output) throws Exception {

StaticFunctionMapO\_VECS\_C.callFunction(callFunctionKey, staticFunctionMapO\_VECS\_C, output);

}

public static void doO\_IDUQ\_CaseFunction(String callFunctionKey, String string

, StaticFunctionMapO\_IDUQ\_E staticFunctionMapO\_IDUQ\_C, Map<String, Object> output) throws IOException

, UnsupportedAudioFileException, InterruptedException, CloneNotSupportedException {

StaticFunctionMapO\_IDUQ\_C.callFunction(callFunctionKey, staticFunctionMapO\_IDUQ\_C, output);

}

public static void doP\_VECS\_CaseFunction(String callFunctionKey, String string

, StaticFunctionMapP\_VECS\_E staticFunctionMapP\_VECS\_C, Map<String, Object> output) throws IOException

, InstantiationException, IllegalAccessException {

StaticFunctionMapP\_VECS\_C.callFunction(callFunctionKey, staticFunctionMapP\_VECS\_C, output);

}

public static void doP\_IDUQ\_CaseFunction(String callFunctionKey, String string

, StaticFunctionMapP\_IDUQ\_E staticFunctionMapP\_IDUQ\_C, Map<String, Object> output) throws IOException {

StaticFunctionMapP\_IDUQ\_C.callFunction(callFunctionKey, staticFunctionMapP\_IDUQ\_C, output);

}

public static void doM\_VECS\_CaseFunction(String callFunctionKey, String string

, StaticFunctionMapM\_VECS\_E staticFunctionMapM\_VECS\_C, Map<String, Object> output) throws Exception {

StaticFunctionMapM\_VECS\_C.callFunction(callFunctionKey, staticFunctionMapM\_VECS\_C, output);

}

public static void doM\_IDUQ\_CaseFunction(String callFunctionKey, String string

, StaticFunctionMapM\_IDUQ\_E staticFunctionMapM\_IDUQ\_C, Map<String, Object> output) throws IOException {

StaticFunctionMapM\_IDUQ\_C.callFunction(callFunctionKey, staticFunctionMapM\_IDUQ\_C, output);

}

public static void doV\_AOPM\_CaseFunction(String callFunctionKey, String string

, StaticFunctionMapV\_AOPM\_E staticFunctionMapV\_AOPM\_C, Map<String, Object> output) throws IOException {

StaticFunctionMapV\_AOPM\_C.callFunction(callFunctionKey, staticFunctionMapV\_AOPM\_C, output);

}

public static void doV\_IDUQ\_CaseFunction(String callFunctionKey, String string

, StaticFunctionMapV\_IDUQ\_E staticFunctionMapV\_IDUQ\_C, Map<String, Object> output) throws IOException {

StaticFunctionMapV\_IDUQ\_C.callFunction(callFunctionKey, staticFunctionMapV\_IDUQ\_C, output);

}

public static void doE\_AOPM\_CaseFunction(String callFunctionKey, String string

, StaticFunctionMapE\_AOPM\_E staticFunctionMapE\_AOPM\_C, Map<String, Object> output) throws IOException {

StaticFunctionMapE\_AOPM\_C.callFunction(callFunctionKey, staticFunctionMapE\_AOPM\_C, output);

}

public static void doE\_IDUQ\_CaseFunction(String callFunctionKey, String string

, StaticFunctionMapE\_IDUQ\_E staticFunctionMapE\_IDUQ\_C, Map<String, Object> output) throws IOException {

StaticFunctionMapE\_IDUQ\_C.callFunction(callFunctionKey, staticFunctionMapE\_IDUQ\_C, output);

}

public static void doC\_AOPM\_CaseFunction(String callFunctionKey, String string

, StaticFunctionMapC\_AOPM\_E staticFunctionMapC\_AOPM\_C, Map<String, Object> output) throws Exception {

StaticFunctionMapC\_AOPM\_C.callFunction(callFunctionKey, staticFunctionMapC\_AOPM\_C, output);

}

public static void doC\_IDUQ\_CaseFunction(String callFunctionKey, String string

, StaticFunctionMapC\_IDUQ\_E staticFunctionMapC\_IDUQ\_C, Map<String, Object> output) throws IOException {

StaticFunctionMapC\_IDUQ\_C.callFunction(callFunctionKey, staticFunctionMapC\_IDUQ\_C, output);

}

public static void doS\_AOPM\_CaseFunction(String callFunctionKey, String string

, StaticFunctionMapS\_AOPM\_E staticFunctionMapS\_AOPM\_C, Map<String, Object> output) throws Exception {

StaticFunctionMapS\_AOPM\_C.callFunction(callFunctionKey, staticFunctionMapS\_AOPM\_C, output);

}

public static void doS\_IDUQ\_CaseFunction(String callFunctionKey, String string

, StaticFunctionMapS\_IDUQ\_E staticFunctionMapS\_IDUQ\_C, Map<String, Object> output) throws IOException

, CloneNotSupportedException {

StaticFunctionMapS\_IDUQ\_C.callFunction(callFunctionKey, staticFunctionMapS\_IDUQ\_C, output);

}

public static void doI\_AOPM\_CaseFunction(String callFunctionKey, String string

, StaticFunctionMapI\_AOPM\_E staticFunctionMapI\_AOPM\_C, Map<String, Object> output) throws IOException

, HeadlessException, InterruptedException {

StaticFunctionMapI\_AOPM\_C.callFunction(callFunctionKey, staticFunctionMapI\_AOPM\_C, output);

}

public static void doI\_VECS\_CaseFunction(String callFunctionKey, String string

, StaticFunctionMapI\_VECS\_E staticFunctionMapI\_VECS\_C, Map<String, Object> output) throws IOException {

StaticFunctionMapI\_VECS\_C.callFunction(callFunctionKey, staticFunctionMapI\_VECS\_C, output);

}

public static void doD\_AOPM\_CaseFunction(String callFunctionKey, String string

, StaticFunctionMapD\_AOPM\_E staticFunctionMapD\_AOPM\_C, Map<String, Object> output) throws IOException {

StaticFunctionMapD\_AOPM\_C.callFunction(callFunctionKey, staticFunctionMapD\_AOPM\_C, output);

}

public static void doD\_VECS\_CaseFunction(String callFunctionKey, String string

, StaticFunctionMapD\_VECS\_E staticFunctionMapD\_VECS\_C, Map<String, Object> output) throws IOException {

StaticFunctionMapD\_VECS\_C.callFunction(callFunctionKey, staticFunctionMapD\_VECS\_C, output);

}

public static void doU\_AOPM\_CaseFunction(String callFunctionKey, String string

, StaticFunctionMapU\_AOPM\_E staticFunctionMapU\_AOPM\_C, Map<String, Object> output) throws Exception {

StaticFunctionMapU\_AOPM\_C.callFunction(callFunctionKey, staticFunctionMapU\_AOPM\_C, output);

}

@SuppressWarnings("static-access")

public static void doU\_VECS\_CaseFunction(String callFunctionKey, String string

, StaticFunctionMapU\_VECS\_E staticFunctionMapU\_VECS\_C, Map<String, Object> output) throws IOException {

// if(callFunctionKey.equalsIgnoreCase("main")) {//稍后分出去

// //写法1

// //case 参数

// if(string.contains("null")) {

// //其他参数可用object， json 都可以

// staticFunctionMapU\_VECS\_C.main(null);//稍后分出去

// output.put("U\_VECS\_main", "void");//非void接口就直接put进去即可。

// }

// //写法2

// //可以插件遍历，可以 接口遍历，可以web的outowire 遍历，

// //无数种方法遍历

// }

//写法2

//我准备设计一种callFunctionKey对应的接口call模式

StaticFunctionMapU\_VECS\_C.callFunction(callFunctionKey, staticFunctionMapU\_VECS\_C, output);

}

public static void doQ\_AOPM\_CaseFunction(String callFunctionKey, String string

, StaticFunctionMapQ\_AOPM\_E staticFunctionMapQ\_AOPM\_C, Map<String, Object> output) throws IOException {

StaticFunctionMapQ\_AOPM\_C.callFunction(callFunctionKey, staticFunctionMapQ\_AOPM\_C, output);

}

public static void doQ\_VECS\_CaseFunction(String callFunctionKey, String string

, StaticFunctionMapQ\_VECS\_E staticFunctionMapQ\_VECS\_C, Map<String, Object> output) throws IOException {

StaticFunctionMapQ\_VECS\_C.callFunction(callFunctionKey, staticFunctionMapQ\_VECS\_C, output);

}

@SuppressWarnings("unchecked")

public static Map<String, Object> preValues(Map<String, Object> output, String[] 传参因子) {

boolean find= false;

Map<String, Object> inputValues = null;

//取值方法, 先检查上一个接口

if(null!= output.get("lastInterfaceBackfeed")) {

if(output.get("lastInterfaceBackfeed").toString().equalsIgnoreCase("success")) {

String lastInterfaceReturn= (String) output.get("lastInterfaceName");//取 上一次运行接口名

if(null!= lastInterfaceReturn) {

Map<String, Object> lastReturns= (Map<String, Object>) output.get(lastInterfaceReturn);

//取上一次运行接口的返回结果。

inputValues= (Map<String, Object>) lastReturns.get("interfaceReturn");//

find= true;

}

}

}

//检查上一个接口是否匹配;

if(find) {

//if(inputValues.containsKey("score")&& inputValues.containsKey("nameScore")) {

// find= true;

//}else {

// find= false;

//}

for(int i= 0; i< 传参因子.length; i++) {//轮训传参 string i++

if(!inputValues.containsKey(传参因子[i])){

find= false;

}

}

}

//////////////////////////////////////////////////////////////////////////////////////

//操作方法,就检查全局传参

if(!find) {//当上一个接口没有返回这个接口需要的数据时，就检查全局传参

inputValues= (Map<String, Object>) output.get("inputValues");//取存储值

}

//检查特定输入参数是否匹配

if(null!= inputValues) {

//if(inputValues.containsKey("score")&& inputValues.containsKey("nameScore")) {

// find= true;

//}

find= true;

for(int i= 0; i< 传参因子.length; i++) {//轮训传参 string i++

if(!inputValues.containsKey(传参因子[i])){

find= false;

}

}

}//本来想设计成插件模式，但是速度降低100倍不止，先不考虑，

/////////////////////////////////////////////////////////////////////////////////////////////////

inputValues.put("find", find);

return inputValues;

}

public static void postValues(Map<String, Object> output, boolean find, Object map, String callFunctionKey) {

if(find) {

//存储方法

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

returnValue.put("interfaceReturn", map);

//输出

output.put(callFunctionKey, returnValue);

output.put("lastInterfaceName", callFunctionKey);//稍后设计成可 理论完美并行的模式。

output.put("lastInterfaceBackfeed", "success");

}else {

output.put("lastInterfaceName", callFunctionKey);

output.put("lastInterfaceBackfeed", "faild");

}

}

}

package SEM.bloom;

import java.io.IOException;

import java.util.Map;

import OSI.OPE.SI.SD.SU.SQ.ASU.OSU.PSU.MSU.AVQ.ASQ.ASU.MPE.procedure.pde.FullDNATokenPDI\_XCDX;

import SVQ.stable.StableCommon;

//将dna加密的 main test 进行封装成函数。准备优化下。

//著作权人+ 作者= 罗瑶光

public class StaticFunctionMapA\_IDUQ\_C {

public static void callFunction(String callFunctionKey

, StaticFunctionMapA\_IDUQ\_E staticFunctionMapA\_IDUQ\_E, Map<String, Object> output) throws IOException {

String[] 传参因子= (String[]) output.get("传参因子");

int 因子= 0;

Object map = null;

if(callFunctionKey.equalsIgnoreCase("getPDW")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= StaticFunctionMapA\_IDUQ\_C.getPDW((String)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("getLock")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= StaticFunctionMapA\_IDUQ\_C.getLock();

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("getCode")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= StaticFunctionMapA\_IDUQ\_C.getCode((String)inputValues.get(传参因子[因子++])

, (String)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("doPDE")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= StaticFunctionMapA\_IDUQ\_C.doPDE((FullDNATokenPDI\_XCDX)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("doPrefixPDE")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= StaticFunctionMapA\_IDUQ\_C.doPrefixPDE((FullDNATokenPDI\_XCDX)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("doPostfixPDE")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= StaticFunctionMapA\_IDUQ\_C.doPostfixPDE((FullDNATokenPDI\_XCDX)inputValues.get(传参因子[因子++])

, (FullDNATokenPDI\_XCDX)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("doSurffixPDE")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= StaticFunctionMapA\_IDUQ\_C.doSurffixPDE((FullDNATokenPDI\_XCDX)inputValues.get(传参因子[因子++])

, (FullDNATokenPDI\_XCDX)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

}

public static void load(StaticFunctionMapA\_IDUQ\_E staticFunctionMapA\_IDUQ\_E) {

//稍后封装

staticFunctionMapA\_IDUQ\_E.annotationMap.put("getPDW", "getPDW");

staticFunctionMapA\_IDUQ\_E.annotationMap.put("getLock", "getLock");

staticFunctionMapA\_IDUQ\_E.annotationMap.put("getCode", "getCode");

staticFunctionMapA\_IDUQ\_E.annotationMap.put("doPDE", "doPDE");

staticFunctionMapA\_IDUQ\_E.annotationMap.put("doPrefixPDE", "doPrefixPDE");

staticFunctionMapA\_IDUQ\_E.annotationMap.put("doPostfixPDE", "doPostfixPDE");

staticFunctionMapA\_IDUQ\_E.annotationMap.put("doSurffixPDE", "doSurffixPDE");

}

//肽语

public static String getPDW(String string) {

FullDNATokenPDI\_XCDX pDE\_RNA\_FullFormular= new FullDNATokenPDI\_XCDX();

pDE\_RNA\_FullFormular.text= string.toString();

pDE\_RNA\_FullFormular.pdw= pDE\_RNA\_FullFormular.initonSect(pDE\_RNA\_FullFormular.text);

return pDE\_RNA\_FullFormular.pdw;

}

//肽锁

public static String getLock() {

FullDNATokenPDI\_XCDX pDE\_RNA\_FullFormular= new FullDNATokenPDI\_XCDX();

String[] lock= new String[12];

lock[0] = "A"; lock[3] = "O"; lock[6] = "P"; lock[9] = "M";

lock[1] = "V"; lock[4] = "E"; lock[7] = "C"; lock[10] = "S";

lock[2] = "I"; lock[5] = "D"; lock[8] = "U"; lock[11] = "Q";

int i= (int)(Math.random()\* 12)% 12;

pDE\_RNA\_FullFormular.lock+= lock[i];

i= (int)(Math.random()\* 12)% 12;

pDE\_RNA\_FullFormular.lock+= lock[i];

i= (int)(Math.random()\* 12)% 12;

pDE\_RNA\_FullFormular.lock+= lock[i];

i= (int)(Math.random()\* 12)% 12;

pDE\_RNA\_FullFormular.lock+= lock[i];

return pDE\_RNA\_FullFormular.lock;

}

//散列肽语 // 第二次修正会增加vpcs 接口多样化。 罗瑶光

public static String getCode(String lock, String pdw) {

FullDNATokenPDI\_XCDX pDE\_RNA\_FullFormular= new FullDNATokenPDI\_XCDX();

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

pDE\_RNA\_FullFormular.code+= lock + pdw.charAt(i);

}

return pDE\_RNA\_FullFormular.code;

}

//pde计算 确定pDE\_RNA\_FullFormular 变量中要有 肽语 肽锁 散列 的输入值。

public static FullDNATokenPDI\_XCDX doPDE(FullDNATokenPDI\_XCDX pDE\_RNA\_FullFormular) {

System.out.println("肽语: "+ pDE\_RNA\_FullFormular.pdw);

System.out.println("肽锁: "+ pDE\_RNA\_FullFormular.lock);

System.out.println("散列肽语:"+ pDE\_RNA\_FullFormular.code);

//pDE\_RNA\_FullFormular.bys= "0.6/0.3/0.5/0.632";

System.out.println("静态密钥: "+ pDE\_RNA\_FullFormular.bys);

pDE\_RNA\_FullFormular.doKeyPress(pDE\_RNA\_FullFormular.code, pDE\_RNA\_FullFormular, false);

System.out.println("静态肽展降元概率钥匙E: "+ pDE\_RNA\_FullFormular.pdedeKey);

System.out.println("静态肽展降元概率钥匙S: "+ pDE\_RNA\_FullFormular.pdedsKey);

System.out.println("静态肽展降元: "+ pDE\_RNA\_FullFormular.pds);

System.out.println("静态肽展增元概率钥匙E: "+ pDE\_RNA\_FullFormular.pdeieKey);

System.out.println("静态肽展增元概率钥匙S: "+ pDE\_RNA\_FullFormular.pdeisKey);

System.out.println("静态肽展增元: "+ pDE\_RNA\_FullFormular.pde);

return pDE\_RNA\_FullFormular;

}

//前序计算 确定pDE\_RNA\_FullFormular 变量中要有 肽语 肽锁 散列 概率钥匙 等相关输入值。

public static FullDNATokenPDI\_XCDX doPrefixPDE(FullDNATokenPDI\_XCDX pDE\_RNA\_FullFormular) {

pDE\_RNA\_FullFormular.time= "" + System.currentTimeMillis();

pDE\_RNA\_FullFormular.cacheId= "ID" + Math.random() + StableCommon.STRING\_SYMBOL\_PER + Math.random();

System.out.println("时间: " + pDE\_RNA\_FullFormular.time);

System.out.println("账号随机缓存字符串: " + pDE\_RNA\_FullFormular.cacheId);

pDE\_RNA\_FullFormular.session\_key= pDE\_RNA\_FullFormular.pde;

System.out.println("Session: " + pDE\_RNA\_FullFormular.session\_key);

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

System.out.println("开始前序验证：");

System.out.println("开始Session解析： " + pDE\_RNA\_FullFormular.session\_key);

System.out.println("开始概率钥匙解析：" + pDE\_RNA\_FullFormular.pdedeKey+ pDE\_RNA\_FullFormular.pdedsKey

+ pDE\_RNA\_FullFormular.pdeieKey+ pDE\_RNA\_FullFormular.pdeisKey);

FullDNATokenPDI\_XCDX pDE\_RNA\_FullFormular1= new FullDNATokenPDI\_XCDX();

pDE\_RNA\_FullFormular1.pdedeKey= pDE\_RNA\_FullFormular.pdedeKey.toString();

pDE\_RNA\_FullFormular1.pdedsKey= pDE\_RNA\_FullFormular.pdedsKey.toString();

pDE\_RNA\_FullFormular1.pdeieKey= pDE\_RNA\_FullFormular.pdeieKey.toString();

pDE\_RNA\_FullFormular1.pdeisKey= pDE\_RNA\_FullFormular.pdeisKey.toString();

pDE\_RNA\_FullFormular.doKeyUnPress(pDE\_RNA\_FullFormular.code, pDE\_RNA\_FullFormular1, true);

System.out.println();

System.out.println("得到原降元元基DNA序列："+ pDE\_RNA\_FullFormular.pds);

System.out.println("得到新降元元基DNA序列："+ pDE\_RNA\_FullFormular1.pds);

System.out.println("得到原元基DNA序列："+ pDE\_RNA\_FullFormular.pde);

System.out.println("得到新元基DNA序列："+ pDE\_RNA\_FullFormular1.pde);

System.out.println("验证正确？");

System.out.println(pDE\_RNA\_FullFormular.pde.equals(pDE\_RNA\_FullFormular1.pde)? "正确": "失败");

return pDE\_RNA\_FullFormular1;

}

//后序计算 确定pDE\_RNA\_FullFormular 和 pDE\_RNA\_FullFormular1 变量中要有 肽语 肽锁 散列 概率钥匙 等相关输入值。

public static FullDNATokenPDI\_XCDX doPostfixPDE(FullDNATokenPDI\_XCDX pDE\_RNA\_FullFormular

, FullDNATokenPDI\_XCDX pDE\_RNA\_FullFormular1) {

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

System.out.println("开始后序验证：");

FullDNATokenPDI\_XCDX pDE\_RNA\_FullFormular2= new FullDNATokenPDI\_XCDX();

pDE\_RNA\_FullFormular2.pdeieKey= pDE\_RNA\_FullFormular.pdedeKey.toString();

pDE\_RNA\_FullFormular2.pdeisKey= pDE\_RNA\_FullFormular.pdedsKey.toString();

pDE\_RNA\_FullFormular2.pdedeKey= pDE\_RNA\_FullFormular.pdeieKey.toString();

pDE\_RNA\_FullFormular2.pdedsKey= pDE\_RNA\_FullFormular.pdeisKey.toString();

System.out.println("准备计算元基DNA序列："+ pDE\_RNA\_FullFormular1.pde);

pDE\_RNA\_FullFormular2.doSessionKeyUnPress(pDE\_RNA\_FullFormular1.pde, pDE\_RNA\_FullFormular2, true);

System.out.println();

System.out.println("得到原续降元元基DNA序列："+ pDE\_RNA\_FullFormular1.pds);

System.out.println("得到后续降元元基DNA序列："+ pDE\_RNA\_FullFormular2.pds);

System.out.println("验证正确？");

System.out.println(pDE\_RNA\_FullFormular1.pds.equals(pDE\_RNA\_FullFormular2.pds)? "正确": "失败");

return pDE\_RNA\_FullFormular2;

}

//整序计算 确定pDE\_RNA\_FullFormular， pDE\_RNA\_FullFormular1 变量中要有 肽语 肽锁 散列 概率钥匙 等相关输入值。

public static FullDNATokenPDI\_XCDX doSurffixPDE(FullDNATokenPDI\_XCDX pDE\_RNA\_FullFormular

, FullDNATokenPDI\_XCDX pDE\_RNA\_FullFormular1) {

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

System.out.println("开始整序验证：");

FullDNATokenPDI\_XCDX pDE\_RNA\_FullFormular3= new FullDNATokenPDI\_XCDX();

pDE\_RNA\_FullFormular3.pdeieKey= pDE\_RNA\_FullFormular.pdeieKey.toString();

pDE\_RNA\_FullFormular3.pdeisKey= pDE\_RNA\_FullFormular.pdeisKey.toString();

pDE\_RNA\_FullFormular3.pdedeKey= pDE\_RNA\_FullFormular.pdeieKey.toString();

pDE\_RNA\_FullFormular3.pdedsKey= pDE\_RNA\_FullFormular.pdeisKey.toString();

System.out.println("准备计算元基DNA序列："+ pDE\_RNA\_FullFormular1.pde);//pde加成

pDE\_RNA\_FullFormular3.doFullSessionKeyUnPress(pDE\_RNA\_FullFormular1.pde, pDE\_RNA\_FullFormular3, true);

System.out.println();

System.out.println("得到原续降元元基DNA序列："+ pDE\_RNA\_FullFormular1.pds);

System.out.println("得到后续降元元基DNA序列："+ pDE\_RNA\_FullFormular3.pds);

System.out.println("验证正确？");

System.out.println(pDE\_RNA\_FullFormular1.pds.equals(pDE\_RNA\_FullFormular3.pds)? "正确": "失败");

System.out.println("准备整序计算元基DNA序列："+ pDE\_RNA\_FullFormular1.pde);

System.out.println("准备整序计算元基DNA序列："+ pDE\_RNA\_FullFormular3.pde);

System.out.println(pDE\_RNA\_FullFormular1.pde.equals(pDE\_RNA\_FullFormular3.pde)? "正确": "失败");

return pDE\_RNA\_FullFormular3;

}

}

package SEM.bloom;

import java.util.HashMap;

import java.util.Map;

import OSI.OPE.SI.SD.SU.SQ.ASU.OSU.PSU.MSU.AVQ.ASQ.ASU.MPE.procedure.pde.FullDNATokenPDI\_XCDX;

public class StaticFunctionMapA\_IDUQ\_E {

public Map<String, String> annotationMap= new HashMap<>();

@SuppressWarnings("unused")

public static void main(String[] argv) {

String string= "罗瑶光";

String pdw= StaticFunctionMapA\_IDUQ\_C.getPDW(string);

String lock= StaticFunctionMapA\_IDUQ\_C.getLock();

String code= StaticFunctionMapA\_IDUQ\_C.getCode(pdw, lock);

FullDNATokenPDI\_XCDX pDE\_RNA\_FullFormular= new FullDNATokenPDI\_XCDX();

pDE\_RNA\_FullFormular.pdw= pdw.toString();

pDE\_RNA\_FullFormular.lock= lock.toString();

pDE\_RNA\_FullFormular.code= code.toString();

pDE\_RNA\_FullFormular= StaticFunctionMapA\_IDUQ\_C.doPDE(pDE\_RNA\_FullFormular);

FullDNATokenPDI\_XCDX pDE\_RNA\_FullFormular1= StaticFunctionMapA\_IDUQ\_C.doPrefixPDE(pDE\_RNA\_FullFormular);

FullDNATokenPDI\_XCDX pDE\_RNA\_FullFormular2= StaticFunctionMapA\_IDUQ\_C.doPostfixPDE(pDE\_RNA\_FullFormular

, pDE\_RNA\_FullFormular1);

FullDNATokenPDI\_XCDX pDE\_RNA\_FullFormular3= StaticFunctionMapA\_IDUQ\_C.doSurffixPDE(pDE\_RNA\_FullFormular

, pDE\_RNA\_FullFormular1);

}

}

package SEM.bloom;

import java.io.IOException;

import java.util.ArrayList;

import java.util.List;

import java.util.Map;

import AEU.OCI.AVC.SUQ.estimation.C.EmotionSample;

import OEI.ME.analysis.E.CogsBinaryForest\_AE;

//用来索引函数 注册类

//罗瑶光

public interface StaticFunctionMapA\_VECS\_C {

public static void callFunction(String callFunctionKey

, StaticFunctionMapA\_VECS\_E staticFunctionMapA\_VECS\_C, Map<String, Object> output) throws IOException {

String[] 传参因子= (String[]) output.get("传参因子");

int 因子= 0;

Object map = null;

if(callFunctionKey.equalsIgnoreCase("parserMixedString")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= StaticFunctionMapA\_VECS\_C.parserMixedString((String)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("parserMixedStringToList")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= staticFunctionMapA\_VECS\_C.parserMixedStringToList((String)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("posReader")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= staticFunctionMapA\_VECS\_C.posReader((String)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("dnnReader")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= staticFunctionMapA\_VECS\_C.dnnReader((String)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("mindReader")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= staticFunctionMapA\_VECS\_C.mindReader((String)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("emotionReader")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= staticFunctionMapA\_VECS\_C.emotionReader((String)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("educationReader")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= staticFunctionMapA\_VECS\_C.educationReader((String)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("environmentReader")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= staticFunctionMapA\_VECS\_C.environmentReader((String)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("lenovoReader")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= staticFunctionMapA\_VECS\_C.lenovoReader((String)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("frequencyReader")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= staticFunctionMapA\_VECS\_C.frequencyReader((String)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

}

public static void load(StaticFunctionMapA\_VECS\_E staticFunctionMapA\_VECS\_E) {

//扫描插件

//扫描接口

//扫描继承

//稍后封装

staticFunctionMapA\_VECS\_E.annotationMap.put("parserMixedString", "parserMixedString");

staticFunctionMapA\_VECS\_E.annotationMap.put("parserMixedStringToList", "parserMixedStringToList");

staticFunctionMapA\_VECS\_E.annotationMap.put("posReader", "posReader");

staticFunctionMapA\_VECS\_E.annotationMap.put("dnnReader", "dnnReader");

staticFunctionMapA\_VECS\_E.annotationMap.put("mindReader", "mindReader");

staticFunctionMapA\_VECS\_E.annotationMap.put("emotionReader", "emotionReader");

staticFunctionMapA\_VECS\_E.annotationMap.put("educationReader", "educationReader");

staticFunctionMapA\_VECS\_E.annotationMap.put("environmentReader", "environmentReader");

staticFunctionMapA\_VECS\_E.annotationMap.put("lenovoReader", "lenovoReader");

staticFunctionMapA\_VECS\_E.annotationMap.put("frequencyReader", "frequencyReader");

}

//分词

public static List<String> parserMixedString(String inputString) throws IOException {

CogsBinaryForest\_AE \_A = new CogsBinaryForest\_AE();

\_A.IV\_Mixed();

List<String> sets = new ArrayList<>();

sets= \_A.parserMixedString(inputString);

//罗瑶光

return sets;

}

public List<String> parserMixedStringToList(String inputString) throws IOException;

//文本分析

public List<String> posReader(String inputString) throws IOException;

//文本分析

public List<String> dnnReader(String inputString) throws IOException;

//读心术

public List<String> mindReader(String inputString) throws IOException;

//情感分析，

public List<String> emotionReader(String inputString) throws IOException;

//教育分析

public List<String> educationReader(String inputString) throws IOException;

//环境分析

public Map<String, EmotionSample> environmentReader(String inputString) throws IOException;

//联想分析

public Map<String, Object> lenovoReader(String inputString) throws IOException;

//词频分析

public List<String> frequencyReader(String inputString) throws IOException;

//视觉动画

//

//

//

}

package SEM.bloom;

import java.io.IOException;

import java.util.ArrayList;

import java.util.HashMap;

//import java.util.HashMap;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import AEU.AVC.SUQ.engine.EmotionInit;

import AEU.AVC.SUQ.engine.EnvironmentInit;

import AEU.AVC.SUQ.engine.LenovoInit;

import AEU.OCI.AVC.SUQ.estimation.C.EmotionSample;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;

import OEI.ME.analysis.E.CogsBinaryForest\_AE;

//import OSI.OPE.SI.SD.SU.SQ.ASU.OSU.PSU.MSU.AVQ.ASQ.ASU.MPE.procedure.pde.FullDNATokenPDI\_XCDX;

import SVQ.stable.StableCommon;

//用来索引函数 注册类

//罗瑶光

public class StaticFunctionMapA\_VECS\_E implements StaticFunctionMapA\_VECS\_C{

public Map<String, String> annotationMap= new HashMap<>();

@Override

public List<String> parserMixedStringToList(String inputString) throws IOException {

CogsBinaryForest\_AE \_A = new CogsBinaryForest\_AE();

\_A.IV\_Mixed();

List<String> sets = new ArrayList<>();

sets= \_A.parserMixedString(inputString);

//罗瑶光

return sets;

}

@Override

public List<String> posReader(String inputString) throws IOException {

CogsBinaryForest\_AE \_A = new CogsBinaryForest\_AE();

\_A.IV\_Mixed();

Map<String, String> pos = \_A.getPosCnToCn();

List<String> sets = new ArrayList<>();

sets= \_A.parserMixedString(inputString);

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

Iterator<String> iterator= sets.iterator();

while(iterator.hasNext()) {

String string= iterator.next();

if (!string.replaceAll("\\s+", "").equals("")) {

list.add(string+ "/"+ pos.get(string));

}else {

list.add(string+ "/"+ "未知");

}

}

return list;

}

@Override

public List<String> dnnReader(String inputString) throws IOException {

// TODO Auto-generated method stub

//dnn不属于这个元基组， 稍后并出去。

return null;

}

@Override

public List<String> mindReader(String inputString) throws IOException {

// TODO Auto-generated method stub

return null;

}

//先把main test 加进来， 稍后整改 输出。

@Override

public List<String> emotionReader(String inputString) throws IOException {

EmotionInit emotionInit = new EmotionInit();

emotionInit.IV\_(inputString);

return null;

}

@Override

public List<String> educationReader(String inputString) throws IOException {

// TODO Auto-generated method stub

return null;

}

@Override

public Map<String, EmotionSample> environmentReader(String inputString) throws IOException {

EnvironmentInit environmentInit = new EnvironmentInit();

environmentInit.IV\_(inputString);

Map<String, EmotionSample> environmentSampleMap = environmentInit.getEmotionSampleMap();

return environmentSampleMap;

}

// 词库计算在C aopm 中，稍后并出去。

@Override

public Map<String, Object> lenovoReader(String inputString) throws IOException {

LenovoInit lenovoInit= new LenovoInit();

lenovoInit.IV\_(inputString);

//Map<String, EmotionSample> environmentSampleMap= lenovoInit.getEnvironmentInit().getEmotionSampleMap();

Map<String, Object> lenovo= lenovoInit.getSensingMap().getLenovoMap();

return lenovo;

}

@Override

public List<String> frequencyReader(String inputString) throws IOException {

CogsBinaryForest\_AE \_A = new CogsBinaryForest\_AE();

\_A.IV\_Mixed();

List<String> sets = new ArrayList<>();

sets= \_A.parserMixedString(inputString);

Map<Integer, WordFrequency> fwa = \_A.getWordFrequencyByReturnSortMap(sets);

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

Iterator<Integer> iterator= fwa.keySet().iterator();

while(iterator.hasNext()) {

int intValue= iterator.next();//32bit 上限 65535

list.add(fwa.get(intValue).getWord() + StableCommon.STRING\_SYMBOL\_PER + fwa.get(intValue).getFrequency());

}

return list;//是前序遍历，应用记得从大到小。

}

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

List<String> list= new StaticFunctionMapA\_VECS\_E().frequencyReader("君不见黄河之水天上来，奔流到海不复还");

Iterator<String> iteraor= list.iterator();

while(iteraor.hasNext()) {

System.out.println(iteraor.next());

}

list= new StaticFunctionMapA\_VECS\_E().posReader("君不见黄河之水天上来，奔流到海不复还");

iteraor= list.iterator();

while(iteraor.hasNext()) {

System.out.println(iteraor.next());

}

list= new StaticFunctionMapA\_VECS\_E().frequencyReader("君不见黄河之水天上来，奔流到海不复还");

iteraor= list.iterator();

while(iteraor.hasNext()) {

System.out.println(iteraor.next());

}

// Map<String, Object> map= new StaticFunctionMapA\_VECS\_E().lenovoReader("C AOPM组，以后并出去 ");

// iteraor= map.keySet().iterator();

// while(iteraor.hasNext()) {

// //HashMap<String, Object> hash= (HashMap<String, Object>)iteraor.next();

// System.out.println(iteraor.next());

// }

// Map<String, EmotionSample> map= new StaticFunctionMapA\_VECS\_E()

// .environmentReader("SensingTest 函数 在P VECS组，稍后并出去");

// iteraor= map.keySet().iterator();

// while(iteraor.hasNext()) {

// EmotionSample hash= map.get(iteraor.next());

// //System.out.println(hash);

// }

}

}

package SEM.bloom;

import java.io.IOException;

import java.io.UnsupportedEncodingException;

import java.net.Socket;

import java.util.Map;

import javax.swing.JTextPane;

import ME.APM.VSQ.App;

import MS.VPC.SH.Sleeper\_H;

import OSI.AOP.MS.VPC.server.VPCSRequest;

import OSI.AOP.MS.VPC.server.VPCSResponse;

//著作权人+ 作者= 罗瑶光

public interface StaticFunctionMapC\_AOPM\_C {

@SuppressWarnings("unchecked")

public static void callFunction(String callFunctionKey

, StaticFunctionMapC\_AOPM\_E staticFunctionMapC\_AOPM\_C, Map<String, Object> output) throws Exception {

String[] 传参因子= (String[]) output.get("传参因子");

int 因子= 0;

Object map = null;

if(callFunctionKey.equalsIgnoreCase("BootVPCSBackEnd")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

staticFunctionMapC\_AOPM\_C.BootVPCSBackEnd((App)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("requestIpFilter")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

staticFunctionMapC\_AOPM\_C.requestIpFilter((Socket)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("requestLinkFilter")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

staticFunctionMapC\_AOPM\_C.requestLinkFilter((Socket)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("requestIpFix")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

staticFunctionMapC\_AOPM\_C.requestIpFix((Socket)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("requestLinkFix")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

staticFunctionMapC\_AOPM\_C.requestLinkFix((Socket)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("IV\_BlockList")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

staticFunctionMapC\_AOPM\_C.IV\_BlockList();

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("requestLinkRecoder")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

staticFunctionMapC\_AOPM\_C.requestLinkRecoder((VPCSRequest)inputValues.get(传参因子[因子++]), (VPCSResponse)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("IQ\_ForwardType")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

staticFunctionMapC\_AOPM\_C.IQ\_ForwardType((Socket)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("forwardToRestMap")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

staticFunctionMapC\_AOPM\_C.forwardToRestMap((Socket)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("IV\_ServerInit\_C\_VPCSFrontEnd")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

staticFunctionMapC\_AOPM\_C.IV\_ServerInit\_C\_VPCSFrontEnd();

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("IV\_Server")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

staticFunctionMapC\_AOPM\_C.IV\_Server((App)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("IV\_Service")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

staticFunctionMapC\_AOPM\_C.IV\_Service((JTextPane)inputValues.get(传参因子[因子++])

, (String)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("IV\_ServerServerInit\_C")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

staticFunctionMapC\_AOPM\_C.IV\_ServerServerInit\_C((App)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("IQ\_Response")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

staticFunctionMapC\_AOPM\_C.IQ\_Response((Socket)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("returnResponse")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

staticFunctionMapC\_AOPM\_C.returnResponse((Socket)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("P\_Rest")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

staticFunctionMapC\_AOPM\_C.P\_Rest((VPCSRequest)inputValues.get(传参因子[因子++])

, (VPCSResponse)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("P\_View")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

staticFunctionMapC\_AOPM\_C.P\_View((VPCSRequest)inputValues.get(传参因子[因子++])

, (VPCSResponse)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("P\_Bytes")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

staticFunctionMapC\_AOPM\_C.P\_Bytes((VPCSRequest)inputValues.get(传参因子[因子++])

, (VPCSResponse)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("P\_Buffer")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

staticFunctionMapC\_AOPM\_C.P\_Buffer((VPCSRequest)inputValues.get(传参因子[因子++])

, (VPCSResponse)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("P\_BufferBytes")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

staticFunctionMapC\_AOPM\_C.P\_BufferBytes((VPCSRequest)inputValues.get(传参因子[因子++])

, (VPCSResponse)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("P\_BytesWithoutZip")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

staticFunctionMapC\_AOPM\_C.P\_BytesWithoutZip((VPCSRequest)inputValues.get(传参因子[因子++])

, (VPCSResponse)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("hugPillow")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

staticFunctionMapC\_AOPM\_C.hugPillow((Sleeper\_H)inputValues.get(传参因子[因子++])

, (Socket)inputValues.get(传参因子[因子++]), (int)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("forward")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= staticFunctionMapC\_AOPM\_C.forward((String)inputValues.get(传参因子[因子++])

, (Map<String, String>)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("getFilePath")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= staticFunctionMapC\_AOPM\_C.getFilePath((String)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("getCode")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= staticFunctionMapC\_AOPM\_C.getCode((String)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

}

public static void load(StaticFunctionMapC\_AOPM\_E staticFunctionMapC\_AOPM\_E) {

// TODO Auto-generated method stub

//稍后封装

staticFunctionMapC\_AOPM\_E.annotationMap.put("BootVPCSBackEnd", "BootVPCSBackEnd");

staticFunctionMapC\_AOPM\_E.annotationMap.put("requestIpFilter", "requestIpFilter");

staticFunctionMapC\_AOPM\_E.annotationMap.put("requestLinkFilter", "requestLinkFilter");

staticFunctionMapC\_AOPM\_E.annotationMap.put("IV\_BlockList", "IV\_BlockList");

staticFunctionMapC\_AOPM\_E.annotationMap.put("requestIpFix", "requestIpFix");

staticFunctionMapC\_AOPM\_E.annotationMap.put("requestLinkFix", "requestLinkFix");

staticFunctionMapC\_AOPM\_E.annotationMap.put("requestIpRecoder", "requestIpRecoder");

staticFunctionMapC\_AOPM\_E.annotationMap.put("requestLinkRecoder", "requestLinkRecoder");

staticFunctionMapC\_AOPM\_E.annotationMap.put("IQ\_ForwardType", "IQ\_ForwardType");

staticFunctionMapC\_AOPM\_E.annotationMap.put("forwardToRestMap", "forwardToRestMap");

staticFunctionMapC\_AOPM\_E.annotationMap.put("IV\_ServerInit\_C\_VPCSFrontEnd", "IV\_ServerInit\_C\_VPCSFrontEnd");

staticFunctionMapC\_AOPM\_E.annotationMap.put("IV\_Server", "IV\_Server");

staticFunctionMapC\_AOPM\_E.annotationMap.put("IV\_Service", "IV\_Service");

staticFunctionMapC\_AOPM\_E.annotationMap.put("IV\_ServerServerInit\_C", "IV\_ServerServerInit\_C");

staticFunctionMapC\_AOPM\_E.annotationMap.put("IQ\_Response", "IQ\_Response");

staticFunctionMapC\_AOPM\_E.annotationMap.put("returnResponse", "returnResponse");

staticFunctionMapC\_AOPM\_E.annotationMap.put("P\_Rest", "P\_Rest");

staticFunctionMapC\_AOPM\_E.annotationMap.put("P\_View", "P\_View");

staticFunctionMapC\_AOPM\_E.annotationMap.put("P\_Bytes", "P\_Bytes");

staticFunctionMapC\_AOPM\_E.annotationMap.put("P\_Buffer", "P\_Buffer");

staticFunctionMapC\_AOPM\_E.annotationMap.put("P\_BufferBytes", "P\_BufferBytes");

staticFunctionMapC\_AOPM\_E.annotationMap.put("P\_BytesWithoutZip", "P\_BytesWithoutZip");

staticFunctionMapC\_AOPM\_E.annotationMap.put("hugPillow", "hugPillow");

staticFunctionMapC\_AOPM\_E.annotationMap.put("forward", "forward");

staticFunctionMapC\_AOPM\_E.annotationMap.put("getFilePath", "getFilePath");

staticFunctionMapC\_AOPM\_E.annotationMap.put("getCode", "getCode");

}

//BootVPCSBackEnd extends Thread

public void BootVPCSBackEnd(App app) throws IOException ;

public void BootVPCSBackEnd() throws IOException ;

// 因为首页已经init了,我之后会改成\_A带入就是了. 罗瑶光20210420

// public void IV\_() ;

// public void bootBackEnd() throws IOException;

//

// //RequestFilter\_C {

// public void main(String[] args);

public void requestIpFilter(Socket socket) ;

public void requestLinkFilter(Socket socket) ;

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

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

public void IV\_BlockList() throws IOException;

//RequestFix\_C {

public void requestIpFix(Socket socket);

public void requestLinkFix(Socket socket);

public void requestIpFix(VPCSRequest vPCSRequest, VPCSResponse vPCSResponse);

public void requestLinkFix(VPCSRequest vPCSRequest, VPCSResponse vPCSResponse);

//RequestRecord\_C {

public void requestIpRecoder(VPCSRequest vPCSRequest, VPCSResponse vPCSResponse);

public void requestLinkRecoder(VPCSRequest vPCSRequest, VPCSResponse vPCSResponse)throws Exception;

//ServerForward\_Standard {

public void IQ\_ForwardType(Socket socket) ;

public void forwardToRestMap(Socket socket) ;

public void IQ\_ForwardType(VPCSRequest vPCSRequest

, VPCSResponse vPCSResponse) throws IOException ;

public void forwardToRestMap(VPCSRequest vPCSRequest, VPCSResponse vPCSResponse) throws Exception;

//ServerInit\_C\_VPCSFrontEnd {// 稍后命名区分下

public void IV\_ServerInit\_C\_VPCSFrontEnd() throws IOException ;

public void IV\_Server(App app) throws IOException ;

//ServerInit\_C {

public void IV\_Service(JTextPane jTextPane, String 前端接口Txt) throws IOException ;

public void IV\_ServerServerInit\_C(App app) throws IOException ;

//ServerInit\_Standard {

public void IV\_Service(String 前端接口Txt, String 服务器名) throws IOException;

public void IV\_Server(String 前端接口Txt, String 服务器名) throws IOException;

//ServerRestMap\_Standard {

public void IQ\_Response(Socket socket) ;

public void returnResponse(Socket socket) ;

public void IQ\_Response(VPCSRequest vPCSRequest, VPCSResponse vPCSResponse) ;

public void returnResponse(VPCSRequest vPCSRequest, VPCSResponse vPCSResponse) ;

public void P\_Rest(VPCSRequest vPCSRequest, VPCSResponse vPCSResponse) throws Exception ;

public void P\_View(VPCSRequest vPCSRequest, VPCSResponse vPCSResponse) ;

public void P\_Bytes(VPCSRequest vPCSRequest

, VPCSResponse vPCSResponse) throws IOException ;

public void P\_Buffer(VPCSRequest vPCSRequest, VPCSResponse vPCSResponse) throws IOException;

public void P\_BufferBytes(VPCSRequest vPCSRequest

, VPCSResponse vPCSResponse) throws UnsupportedEncodingException, IOException ;

public void P\_BytesWithoutZip(VPCSRequest vPCSRequest, VPCSResponse vPCSResponse) throws IOException;

// ServerSleeper\_Standard extends Thread implements Runnable{

public void hugPillow(Sleeper\_H sleeper\_H, Socket accept, int hashCode) ;

//ServerVPC\_Standard {

public String forward(String string, Map<String, String> data) throws Exception ;

public String getCode(String filePath) throws IOException;

public String getFilePath(String string);

//小接口略

}

package SEM.bloom;

import java.io.IOException;

import java.io.UnsupportedEncodingException;

import java.net.Socket;

import java.util.HashMap;

import java.util.Map;

import javax.swing.JTextPane;

import ME.APM.VSQ.App;

import MS.VPC.SH.Sleeper\_H;

import OSI.AOP.MS.VPC.server.RequestFilter\_C;

import OSI.AOP.MS.VPC.server.RequestFix\_C;

import OSI.AOP.MS.VPC.server.RequestRecord\_C;

import OSI.AOP.MS.VPC.server.ServerForward\_Standard;

import OSI.AOP.MS.VPC.server.ServerInit\_C;

import OSI.AOP.MS.VPC.server.ServerInit\_C\_VPCSFrontEnd;

import OSI.AOP.MS.VPC.server.ServerInit\_Standard;

import OSI.AOP.MS.VPC.server.ServerRestMap\_Standard;

import OSI.AOP.MS.VPC.server.ServerSleeper\_Standard;

import OSI.AOP.MS.VPC.server.ServerVPC\_Standard;

import OSI.AOP.MS.VPC.server.VPCSRequest;

import OSI.AOP.MS.VPC.server.VPCSResponse;

//著作权人+ 作者= 罗瑶光

//vpcs 服务器的 STANDARD 标准示例

public class StaticFunctionMapC\_AOPM\_E implements StaticFunctionMapC\_AOPM\_C{

public Map<String, String> annotationMap= new HashMap<>();

//BootVPCSBackEnd extends Thread

public void BootVPCSBackEnd(App app) throws IOException{

new OSI.AOP.MS.VPC.server.BootVPCSBackEnd(app);

}

public void BootVPCSBackEnd() throws IOException {

new OSI.AOP.MS.VPC.server.BootVPCSBackEnd();

}

// // // 因为首页已经init了,我之后会改成\_A带入就是了. 罗瑶光20210420 可自适应稍后vpcs细化接口

// // public void IV\_(){

// // }

// public void bootBackEnd() throws IOException{

//

// }

//RequestFilter\_C {

// public void main(String[] args){

//

// }

public void requestIpFilter(Socket socket) {

RequestFilter\_C.requestIpFilter(socket);

}

public void requestLinkFilter(Socket socket) {

RequestFilter\_C.requestLinkFilter( socket);

}

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

RequestFilter\_C. requestIpFilter( vPCSRequest, vPCSResponse) ;

}

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

RequestFilter\_C. requestLinkFilter( vpcsRequest, vPCSResponse);

}

public void IV\_BlockList() throws IOException{

RequestFilter\_C. IV\_BlockList();

}

//RequestFix\_C {

public void requestIpFix(Socket socket){

RequestFix\_C.requestIpFix( socket);

}

public void requestLinkFix(Socket socket){

RequestFix\_C.requestLinkFix( socket);

}

public void requestIpFix(VPCSRequest vPCSRequest, VPCSResponse vPCSResponse){

RequestFix\_C. requestIpFix( vPCSRequest, vPCSResponse);

}

public void requestLinkFix(VPCSRequest vPCSRequest, VPCSResponse vPCSResponse){

RequestFix\_C.requestLinkFix( vPCSRequest, vPCSResponse);

}

//RequestRecord\_C {

public void requestIpRecoder(VPCSRequest vPCSRequest, VPCSResponse vPCSResponse){

RequestRecord\_C. requestIpRecoder( vPCSRequest, vPCSResponse);

}

public void requestLinkRecoder(VPCSRequest vPCSRequest, VPCSResponse vPCSResponse)throws Exception{

RequestRecord\_C.requestLinkRecoder( vPCSRequest, vPCSResponse);

}

//ServerForward\_Standard {

public void IQ\_ForwardType(Socket socket) {

ServerForward\_Standard. IQ\_ForwardType( socket) ;

}

public void forwardToRestMap(Socket socket) {

ServerForward\_Standard.forwardToRestMap( socket);

}

public void IQ\_ForwardType(VPCSRequest vPCSRequest

, VPCSResponse vPCSResponse) throws IOException {

ServerForward\_Standard.IQ\_ForwardType( vPCSRequest, vPCSResponse);

}

public void forwardToRestMap(VPCSRequest vPCSRequest, VPCSResponse vPCSResponse) throws Exception{

ServerForward\_Standard.forwardToRestMap( vPCSRequest, vPCSResponse);

}

//ServerInit\_C\_VPCSFrontEnd {// 稍后命名区分下 改成 return

public void IV\_ServerInit\_C\_VPCSFrontEnd() throws IOException {

new ServerInit\_C\_VPCSFrontEnd();

}

public void IV\_Server(App app) throws IOException {

new ServerInit\_C\_VPCSFrontEnd(). IV\_Server( app);

}

//ServerInit\_C {

public void IV\_Service(JTextPane jTextPane, String 前端接口Txt) throws IOException {

new ServerInit\_C().IV\_Service( jTextPane, 前端接口Txt);

}

public void IV\_ServerServerInit\_C(App app) throws IOException {

new ServerInit\_C().IV\_Server( app) ;

}

//ServerInit\_Standard {

public void IV\_Service(String 前端接口Txt, String 服务器名) throws IOException{

new ServerInit\_Standard(). IV\_Service( 前端接口Txt, 服务器名) ;

}

public void IV\_Server(String 前端接口Txt, String 服务器名) throws IOException{

new ServerInit\_Standard().IV\_Server( 前端接口Txt, 服务器名);

}

//ServerRestMap\_Standard {

public void IQ\_Response(Socket socket) {

ServerRestMap\_Standard.IQ\_Response( socket);

}

public void returnResponse(Socket socket) {

ServerRestMap\_Standard.returnResponse( socket);

}

public void IQ\_Response(VPCSRequest vPCSRequest, VPCSResponse vPCSResponse) {

ServerRestMap\_Standard.IQ\_Response( vPCSRequest, vPCSResponse);

}

public void returnResponse(VPCSRequest vPCSRequest, VPCSResponse vPCSResponse) {

ServerRestMap\_Standard.returnResponse( vPCSRequest, vPCSResponse);

}

public void P\_Rest(VPCSRequest vPCSRequest, VPCSResponse vPCSResponse) throws Exception {

ServerRestMap\_Standard.P\_Rest( vPCSRequest, vPCSResponse);

}

public void P\_View(VPCSRequest vPCSRequest, VPCSResponse vPCSResponse) {

ServerRestMap\_Standard. P\_View( vPCSRequest, vPCSResponse);

}

public void P\_Bytes(VPCSRequest vPCSRequest, VPCSResponse vPCSResponse) throws IOException {

ServerRestMap\_Standard. P\_Bytes( vPCSRequest, vPCSResponse) ;

}

public void P\_Buffer(VPCSRequest vPCSRequest, VPCSResponse vPCSResponse) throws IOException{

ServerRestMap\_Standard. P\_Buffer( vPCSRequest, vPCSResponse) ;

}

public void P\_BufferBytes(VPCSRequest vPCSRequest

, VPCSResponse vPCSResponse) throws UnsupportedEncodingException, IOException {

ServerRestMap\_Standard. P\_BufferBytes( vPCSRequest, vPCSResponse);

}

public void P\_BytesWithoutZip(VPCSRequest vPCSRequest, VPCSResponse vPCSResponse) throws IOException{

ServerRestMap\_Standard.P\_BytesWithoutZip( vPCSRequest, vPCSResponse);

}

// ServerSleeper\_Standard extends Thread implements Runnable{

public void hugPillow(Sleeper\_H sleeper\_H, Socket accept, int hashCode) {

new ServerSleeper\_Standard().hugPillow( sleeper\_H, accept, hashCode);

}

//ServerVPC\_Standard {

public String forward(String string, Map<String, String> data) throws Exception {

return ServerVPC\_Standard.forward( string, data);

}

public String getCode(String filePath) throws IOException{

return ServerVPC\_Standard. getCode( filePath) ;

}

public String getFilePath(String string){

return ServerVPC\_Standard.getFilePath( string);

}

//小接口略

}

package SEM.bloom;

import java.awt.Container;

import java.io.File;

import java.io.IOException;

import java.util.HashMap;

import java.util.Map;

import OSI.OPE.AOPM.VECS.IDUQ.OVU.PQE.flash.GUISample;

import OSI.OPE.OEI.PVI.SOI.SMQ.load.LoadFile;

import OSI.OPE.OVU.MVQ.OVU.PQE.nodeView.NodeShow;

import OSI.OPE.OVU.MVU.OVU.PQE.nodeEdit.LinkList;

import OSI.OPE.OVU.MVU.OVU.PQE.nodeEdit.LinkNode;

import PEI.thread.MakeContainerSJFX;

import javax.swing.JFrame;

import javax.swing.JTextPane;

import ME.APM.VSQ.App;

import MVQ.tabbedpane.DetabbedPane;

//import OCI.ME.analysis.C.A;

import OEI.ME.analysis.E.CogsBinaryForest\_AE;

//著作权人+ 作者= 罗瑶光

public class StaticFunctionMapV\_IDUQ\_E implements StaticFunctionMapV\_IDUQ\_C {

public Map<String, String> annotationMap= new HashMap<>();

@Override

public MakeContainerSJFX 初始ETL(App app, Container jpanelSecond) {

MakeContainerSJFX makeContainerSJFX= new MakeContainerSJFX(app.tableData\_old, app.text, app.\_A

, jpanelSecond, app, app.jTabbedpane,app.tabNames, app.pos, app.pose, app.etc, app.cte);

makeContainerSJFX.start();

return makeContainerSJFX;

}

@Override

public GUISample 仅仅初始ETL(Object[][] tableData\_old, JTextPane text, App u

, CogsBinaryForest\_AE \_A, Map<String, String> pos) {

GUISample gUISample= new GUISample();

gUISample.IV\_(tableData\_old,text, u, \_A, pos);

gUISample.start();

gUISample.validate();

return gUISample;

}

//调通了下，不要使用老接口，建议写新的节点。参照已有的 image read节点即可。

//之后进行复杂场景设计。

@Override

public void 展示ETL() throws IOException {

GUISample gUISample= new GUISample();

App app= new App();

app.gUISample= gUISample;

CogsBinaryForest\_AE \_A= new CogsBinaryForest\_AE();

\_A.IV\_Mixed();

Map<String, String> pos= \_A.getPosCnToCn();

JTextPane text= new JTextPane();

app.jTabbedpane= new DetabbedPane(0, 0, null);

gUISample.IV\_(new Object[10][10], text, app, \_A, pos);

gUISample.start();

gUISample.validate();

JFrame jFrame= new JFrame();

jFrame.add(gUISample);

jFrame.setSize(1490, 980);

jFrame.setVisible(true);

}

@Override

public void ETL文档读取() {

}

@Override

public void ETL文档执行() {

}

@Override

public void ETL文档保存() {

}

// LoadFile{

public String getOrigianlTextByLock(String inputString, String lockString) {

return LoadFile.getOrigianlTextByLock(inputString, lockString);

}

public LinkNode Load(LinkNode first, NodeShow nodeView, File file, LinkList thislist) {

return LoadFile.Load(first, nodeView, file, thislist);

}

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

new StaticFunctionMapV\_IDUQ\_E().展示ETL();

}

}

package SEM.bloom;

import java.awt.Container;

import java.io.File;

import java.io.IOException;

import java.util.Map;

import javax.swing.JTextPane;

import ME.APM.VSQ.App;

import OEI.ME.analysis.E.CogsBinaryForest\_AE;

import OSI.OPE.AOPM.VECS.IDUQ.OVU.PQE.flash.GUISample;

import OSI.OPE.OVU.MVQ.OVU.PQE.nodeView.NodeShow;

import OSI.OPE.OVU.MVU.OVU.PQE.nodeEdit.LinkList;

import OSI.OPE.OVU.MVU.OVU.PQE.nodeEdit.LinkNode;

import PEI.thread.MakeContainerSJFX;

//著作权人+ 作者= 罗瑶光

public interface StaticFunctionMapV\_IDUQ\_C {

@SuppressWarnings("unchecked")

public static void callFunction(String callFunctionKey, StaticFunctionMapV\_IDUQ\_E staticFunctionMapV\_IDUQ\_C

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

String[] 传参因子= (String[]) output.get("传参因子");

int 因子= 0;

Object map = null;

if(callFunctionKey.equalsIgnoreCase("初始ETL")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= staticFunctionMapV\_IDUQ\_C.初始ETL((App) inputValues.get(传参因子[因子++])

, (Container) inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("仅仅初始ETL")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= staticFunctionMapV\_IDUQ\_C.仅仅初始ETL((Object[][]) inputValues.get(传参因子[因子++])

, (JTextPane) inputValues.get(传参因子[因子++]), (App) inputValues.get(传参因子[因子++])

, (CogsBinaryForest\_AE) inputValues.get(传参因子[因子++])

, (Map<String, String>) inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("展示ETL")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

staticFunctionMapV\_IDUQ\_C.展示ETL();

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("ETL文档读取")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

staticFunctionMapV\_IDUQ\_C.ETL文档读取();

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("ETL文档执行")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

staticFunctionMapV\_IDUQ\_C.ETL文档执行();

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("ETL文档保")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

staticFunctionMapV\_IDUQ\_C.ETL文档保存();

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("getOrigianlTextByLock")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= staticFunctionMapV\_IDUQ\_C.getOrigianlTextByLock((String)inputValues.get(传参因子[因子++])

,(String)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("Load")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= staticFunctionMapV\_IDUQ\_C.Load((LinkNode) inputValues.get(传参因子[因子++])

, (NodeShow) inputValues.get(传参因子[因子++])

, (File) inputValues.get(传参因子[因子++]), (LinkList) inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

}

public static void load(StaticFunctionMapV\_IDUQ\_E staticFunctionMapV\_IDUQ\_E) {

//稍后封装

staticFunctionMapV\_IDUQ\_E.annotationMap.put("初始ETL", "初始ETL");

staticFunctionMapV\_IDUQ\_E.annotationMap.put("仅仅初始ETL", "仅仅初始ETL");

staticFunctionMapV\_IDUQ\_E.annotationMap.put("展示ETL", "展示ETL");

staticFunctionMapV\_IDUQ\_E.annotationMap.put("ETL文档读取", "ETL文档读取");

staticFunctionMapV\_IDUQ\_E.annotationMap.put("ETL文档执行", "ETL文档执行");

staticFunctionMapV\_IDUQ\_E.annotationMap.put("ETL文档保存", "ETL文档保存");

staticFunctionMapV\_IDUQ\_E.annotationMap.put("getOrigianlTextByLock", "getOrigianlTextByLock");

staticFunctionMapV\_IDUQ\_E.annotationMap.put("Load", "Load");

}

public MakeContainerSJFX 初始ETL(App app, Container jpanelSecond);

public GUISample 仅仅初始ETL(Object[][] tableData\_old, JTextPane text, App u, CogsBinaryForest\_AE \_A, Map<String, String> pos);

public void 展示ETL() throws IOException;

public void ETL文档读取();

public void ETL文档执行();

public void ETL文档保存();

// LoadFile{

public String getOrigianlTextByLock(String inputString, String lockString);

public LinkNode Load(LinkNode first, NodeShow nodeView, File file, LinkList thislist);

}

package SEM.bloom;

import java.io.IOException;

import java.util.HashMap;

import java.util.Map;

import OSI.OPE.SI.SD.SU.SQ.ASU.OSU.PSU.MSU.AVQ.ASQ.ASU.MPE.procedure.pde.FullDNATokenPDI;

import VPC.VQS.DSU.utils.DetaDBUtil;

import VPC.VQS.DSU.utils.DetaFrontEndUtil;

import VPC.VQS.DSU.utils.DetaUtil;

import VPC.VQS.DSU.utils.GzipUtil;

import VPC.transaction.PdeSwap;

import VPC.transaction.PdeSwapFix;

//著作权人+ 作者= 罗瑶光

//Refer的源码来自 《DNA 元基催化与肽计算 第三修订版本 V039010912》

//证书编号：国作登字-2021-L-00268255 (中华人民共和国 国家版权登记中心)

public class StaticFunctionMapV\_AOPM\_E implements StaticFunctionMapV\_AOPM\_C {

public Map<String, String> annotationMap= new HashMap<>();

public static void load() {

// TODO Auto-generated method stub

}

//PdeSwap{

public String PdeSwapPdcToPde(String pdc, String lock, String de, String ds, String ie, String is) {

return PdeSwap.PdcToPde(pdc, lock, de, ds, ie, is);

}

public String PdeSwapPdcToPds(String pdc, String lock, String de, String ds, String ie, String is) {

return PdeSwap.PdcToPds(pdc, lock, de, ds, ie, is);

}

//刚检查pds to pde 似乎被猫腻了，我测试下

//罗瑶光 MPOASCEV

public String PdeSwapPdeToPds(String pds, String lock, String de, String ds, String ie, String is) {

return PdeSwap.PdeToPds(pds, lock, de, ds, ie, is);

}

//把FullDNATokenPDI 类里 do\_PDE\_RNA\_FullFormular\_Back 函数中的 SCEV MPOA 注释的部分 分出来到在这里。

//罗瑶光 VECSAOPM

public String PdeSwapPdsToPde(String pds, String lock, String de, String ds, String ie, String is) {

return PdeSwap.PdsToPde(pds, lock, de, ds, ie, is);

}

//PdeSwapFix{

public String PdeSwapFixpdcToPde(String pdc, String lock, String de, String ds, String ie, String is) {

return PdeSwapFix.pdcToPde(pdc, lock, de, ds, ie, is);

}

public String PdeSwapFixpdcToPds(String pdc, String lock, String de, String ds, String ie, String is) {

return PdeSwapFix.pdcToPds(pdc, lock, de, ds, ie, is);

}

//刚检查pds to pde 似乎被猫腻了，我测试下

//罗瑶光 MPOASCEV

public String PdeSwapFixpdeToPds(String pds, String lock, String de, String ds, String ie, String is) {

return PdeSwapFix.pdeToPds(pds, lock, de, ds, ie, is);

}

//把FullDNATokenPDI 类里 do\_PDE\_RNA\_FullFormular\_Back 函数中的 SCEV MPOA 注释的部分 分出来到在这里。

//罗瑶光 VECSAOPM

public String PdeSwapFixpdsToPde(String pds, String lock, String de, String ds, String ie, String is){

return PdeSwapFix.pdsToPde(pds, lock, de, ds, ie, is);

}

public String PdeSwapFixtextToPdw(FullDNATokenPDI pDE\_RNA\_FullFormular, String password) {

return PdeSwapFix.textToPdw(pDE\_RNA\_FullFormular, password);

}

public String PdeSwapFixpdwToPdc(FullDNATokenPDI pDE\_RNA\_FullFormular) {

return PdeSwapFix.pdwToPdc(pDE\_RNA\_FullFormular);

}

//下面这个test demo 展示了 密码-> 肽文-> 肽锁+肽密码-> 密钥,pds和pde密码 -> pds转pde验证-> pde转pds验证

//全部封装成函数

//罗瑶光 20210830

public static void main(String[] argv) {

FullDNATokenPDI pDE\_RNA\_FullFormular= new FullDNATokenPDI();

pDE\_RNA\_FullFormular.text= "控制吸收";

pDE\_RNA\_FullFormular.pdw= PdeSwapFix.textToPdw(pDE\_RNA\_FullFormular, pDE\_RNA\_FullFormular.text);

pDE\_RNA\_FullFormular.code= PdeSwapFix.pdwToPdc(pDE\_RNA\_FullFormular);

System.out.println("肽语: "+ pDE\_RNA\_FullFormular.pdw);

System.out.println("肽锁: "+ pDE\_RNA\_FullFormular.lock);

System.out.println("散列肽语:"+ pDE\_RNA\_FullFormular.code);

////////////////////////////////////////////////////

pDE\_RNA\_FullFormular.doKeyPress(pDE\_RNA\_FullFormular.code, pDE\_RNA\_FullFormular, false);

System.out.println("静态肽展降元概率钥匙E: "+ pDE\_RNA\_FullFormular.pdedeKey);

System.out.println("静态肽展降元概率钥匙S: "+ pDE\_RNA\_FullFormular.pdedsKey);

System.out.println("静态肽展降元: "+ pDE\_RNA\_FullFormular.pds);

System.out.println("静态肽展增元概率钥匙E: "+ pDE\_RNA\_FullFormular.pdeieKey);

System.out.println("静态肽展增元概率钥匙S: "+ pDE\_RNA\_FullFormular.pdeisKey);

System.out.println("静态肽展增元: "+ pDE\_RNA\_FullFormular.pde);

//////////////////////////////////////////////////////

pDE\_RNA\_FullFormular.time= "" + System.currentTimeMillis();

pDE\_RNA\_FullFormular.cacheId= "ID" + Math.random() + ":" + Math.random();

System.out.println("时间: " + pDE\_RNA\_FullFormular.time);

System.out.println("账号随机缓存字符串: " + pDE\_RNA\_FullFormular.cacheId);

/////////////////////////////////////////////////////

pDE\_RNA\_FullFormular.session\_key= pDE\_RNA\_FullFormular.pde;

System.out.println("Session: " + pDE\_RNA\_FullFormular.session\_key);

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

System.out.println("开始前序验证：");

System.out.println("开始Session解析： " + pDE\_RNA\_FullFormular.session\_key);

System.out.println("开始概率钥匙解析：" + pDE\_RNA\_FullFormular.pdedeKey+ pDE\_RNA\_FullFormular.pdedsKey

+ pDE\_RNA\_FullFormular.pdeieKey+ pDE\_RNA\_FullFormular.pdeisKey);

/////////////////////////////////////////////////

FullDNATokenPDI pDE\_RNA\_FullFormular1= new FullDNATokenPDI();

pDE\_RNA\_FullFormular1.pdedeKey= pDE\_RNA\_FullFormular.pdedeKey.toString();

pDE\_RNA\_FullFormular1.pdedsKey= pDE\_RNA\_FullFormular.pdedsKey.toString();

pDE\_RNA\_FullFormular1.pdeieKey= pDE\_RNA\_FullFormular.pdeieKey.toString();

pDE\_RNA\_FullFormular1.pdeisKey= pDE\_RNA\_FullFormular.pdeisKey.toString();

pDE\_RNA\_FullFormular.doKeyUnPress(pDE\_RNA\_FullFormular.code, pDE\_RNA\_FullFormular1, true);

System.out.println();

System.out.println("得到原降元元基DNA序列："+ pDE\_RNA\_FullFormular.pds);

System.out.println("得到新降元元基DNA序列："+ pDE\_RNA\_FullFormular1.pds);

System.out.println("得到原元基DNA序列："+ pDE\_RNA\_FullFormular.pde);

System.out.println("得到新元基DNA序列："+ pDE\_RNA\_FullFormular1.pde);

System.out.println("验证正确？");

System.out.println(pDE\_RNA\_FullFormular.pde.equals(pDE\_RNA\_FullFormular1.pde)? "正确": "失败");

/////////////////////////////////////////

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

System.out.println("开始pde降元验证：");

FullDNATokenPDI pDE\_RNA\_FullFormular2= new FullDNATokenPDI();

pDE\_RNA\_FullFormular2.pdeieKey= pDE\_RNA\_FullFormular.pdeieKey.toString();

pDE\_RNA\_FullFormular2.pdeisKey= pDE\_RNA\_FullFormular.pdeisKey.toString();

pDE\_RNA\_FullFormular2.pdedeKey= pDE\_RNA\_FullFormular.pdeieKey.toString();

pDE\_RNA\_FullFormular2.pdedsKey= pDE\_RNA\_FullFormular.pdeisKey.toString();

System.out.println("准备计算元基DNA序列："+ pDE\_RNA\_FullFormular1.pde);

String pds= PdeSwapFix.pdeToPds(pDE\_RNA\_FullFormular1.pde, "", pDE\_RNA\_FullFormular2.pdedeKey

, pDE\_RNA\_FullFormular2.pdedsKey

, pDE\_RNA\_FullFormular2.pdeieKey

, pDE\_RNA\_FullFormular2.pdeisKey);

System.out.println("pds");

System.out.println("pds");

System.out.println(pDE\_RNA\_FullFormular1.pds);

System.out.println(pds);

//////////////////////////////////////////////////////////////////

System.out.println("开始pds增元验证：");

FullDNATokenPDI pDE\_RNA\_FullFormular3= new FullDNATokenPDI();

pDE\_RNA\_FullFormular3.pdeieKey= pDE\_RNA\_FullFormular.pdeieKey.toString();

pDE\_RNA\_FullFormular3.pdeisKey= pDE\_RNA\_FullFormular.pdeisKey.toString();

pDE\_RNA\_FullFormular3.pdedeKey= pDE\_RNA\_FullFormular.pdeieKey.toString();

pDE\_RNA\_FullFormular3.pdedsKey= pDE\_RNA\_FullFormular.pdeisKey.toString();

String pde= PdeSwapFix.pdsToPde(pDE\_RNA\_FullFormular1.pds, "", pDE\_RNA\_FullFormular3.pdedeKey

, pDE\_RNA\_FullFormular3.pdedsKey

, pDE\_RNA\_FullFormular3.pdeieKey

, pDE\_RNA\_FullFormular3.pdeisKey);

System.out.println("pde");

System.out.println("pde");

System.out.println(pDE\_RNA\_FullFormular1.pde);

System.out.println(pde);

}

//DetaDBUtil {

public String DetaDBUtilDBRequest(String request) throws IOException{

return DetaDBUtil.DBRequest(request);

}

public String DetaDBUtilbackEndRequest(String request) throws IOException{

return DetaDBUtil.backEndRequest(request);

}

public String DetaDBUtilcacheRequest(String request) throws IOException{

return DetaDBUtil.cacheRequest(request);

}

public void IV\_CulumnNameType() {

DetaDBUtil.IV\_CulumnNameType();

}

public boolean withoutCulumnNameType(String culumnTypeString) {

return DetaDBUtil.withoutCulumnNameType(culumnTypeString);

}

// DetaFrontEndUtil {

public String DetaFrontEndUtilbackEndRequest(String request) throws IOException{

return DetaFrontEndUtil.backEndRequest(request);

}

//先不动 稍后归纳 华瑞集rest走 前端还是后端还是数据库。

public String DetaFrontEndUtilhuaRuiJiRequest(String request) throws IOException {

return DetaFrontEndUtil.huaRuiJiRequest(request);

}

public String DetaFrontEndUtilcacheRequest(String request) throws IOException {

return DetaFrontEndUtilcacheRequest(request);

}

//DetaUtil {

public void IV\_DB(String dbConfigPath) {

DetaUtil.IV\_DB(dbConfigPath);

}

//GzipUtil {

// 压缩

public byte[] compress(byte[] data) throws IOException{

return GzipUtil.compress(data);

}

public byte[] compress(String str, String stringTypes) throws IOException{

return GzipUtil.compress(str, stringTypes);

}

public byte[] uncompress(byte[] data) throws IOException{

return GzipUtil.compress(data);

}

//jogl 画图略

}

package SEM.bloom;

import java.io.IOException;

import java.util.Map;

import OSI.OPE.SI.SD.SU.SQ.ASU.OSU.PSU.MSU.AVQ.ASQ.ASU.MPE.procedure.pde.FullDNATokenPDI;

import VPC.transaction.PdeSwapFix;

//著作权人+ 作者= 罗瑶光

//Refer的源码来自 《DNA 元基催化与肽计算 第三修订版本 V039010912》

//证书编号：国作登字-2021-L-00268255 (中华人民共和国 国家版权登记中心)

public interface StaticFunctionMapV\_AOPM\_C {

public static void callFunction(String callFunctionKey, StaticFunctionMapV\_AOPM\_E staticFunctionMapV\_AOPM\_C

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

String[] 传参因子= (String[]) output.get("传参因子");

int 因子= 0;

Object map = null;

if(callFunctionKey.equalsIgnoreCase("PdeSwapPdcToPde")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= staticFunctionMapV\_AOPM\_C.PdeSwapPdcToPde((String)inputValues.get(传参因子[因子++])

,(String)inputValues.get(传参因子[因子++]),(String)inputValues.get(传参因子[因子++])

,(String)inputValues.get(传参因子[因子++]),(String)inputValues.get(传参因子[因子++])

,(String)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("PdeSwapPdcToPds")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= staticFunctionMapV\_AOPM\_C.PdeSwapPdcToPds((String)inputValues.get(传参因子[因子++])

,(String)inputValues.get(传参因子[因子++]),(String)inputValues.get(传参因子[因子++])

,(String)inputValues.get(传参因子[因子++]),(String)inputValues.get(传参因子[因子++])

,(String)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("PdeSwapPdeToPds")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= staticFunctionMapV\_AOPM\_C.PdeSwapPdeToPds((String)inputValues.get(传参因子[因子++])

,(String)inputValues.get(传参因子[因子++]),(String)inputValues.get(传参因子[因子++])

,(String)inputValues.get(传参因子[因子++]),(String)inputValues.get(传参因子[因子++])

,(String)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("PdeSwapPdsToPde")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= staticFunctionMapV\_AOPM\_C.PdeSwapPdsToPde((String)inputValues.get(传参因子[因子++])

,(String)inputValues.get(传参因子[因子++]),(String)inputValues.get(传参因子[因子++])

,(String)inputValues.get(传参因子[因子++]),(String)inputValues.get(传参因子[因子++])

,(String)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("PdeSwapFixpdcToPde")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= staticFunctionMapV\_AOPM\_C.PdeSwapFixpdcToPde((String)inputValues.get(传参因子[因子++])

,(String)inputValues.get(传参因子[因子++]),(String)inputValues.get(传参因子[因子++])

,(String)inputValues.get(传参因子[因子++]),(String)inputValues.get(传参因子[因子++])

,(String)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("PdeSwapFixpdcToPds")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= staticFunctionMapV\_AOPM\_C.PdeSwapFixpdcToPds((String)inputValues.get(传参因子[因子++])

,(String)inputValues.get(传参因子[因子++]),(String)inputValues.get(传参因子[因子++])

,(String)inputValues.get(传参因子[因子++]),(String)inputValues.get(传参因子[因子++])

,(String)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("PdeSwapFixpdsToPde")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= staticFunctionMapV\_AOPM\_C.PdeSwapFixpdsToPde((String)inputValues.get(传参因子[因子++])

,(String)inputValues.get(传参因子[因子++]),(String)inputValues.get(传参因子[因子++])

,(String)inputValues.get(传参因子[因子++]),(String)inputValues.get(传参因子[因子++])

,(String)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("PdeSwapFixpdeToPds")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= staticFunctionMapV\_AOPM\_C.PdeSwapFixpdeToPds((String)inputValues.get(传参因子[因子++])

,(String)inputValues.get(传参因子[因子++]),(String)inputValues.get(传参因子[因子++])

,(String)inputValues.get(传参因子[因子++]),(String)inputValues.get(传参因子[因子++])

,(String)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("PdeSwapFixtextToPdw")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= staticFunctionMapV\_AOPM\_C.PdeSwapFixtextToPdw((FullDNATokenPDI)inputValues.get(传参因子[因子++])

,(String)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("PdeSwapFixtextToPdw")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= staticFunctionMapV\_AOPM\_C.PdeSwapFixtextToPdw((FullDNATokenPDI)inputValues.get(传参因子[因子++])

,(String)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("PdeSwapFixpdwToPdc")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= staticFunctionMapV\_AOPM\_C.PdeSwapFixpdwToPdc((FullDNATokenPDI)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("DetaDBUtilDBRequest")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= staticFunctionMapV\_AOPM\_C.DetaDBUtilDBRequest((String)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("DetaDBUtilbackEndRequest")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= staticFunctionMapV\_AOPM\_C.DetaDBUtilbackEndRequest((String)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("DetaDBUtilcacheRequest")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= staticFunctionMapV\_AOPM\_C.DetaDBUtilcacheRequest((String)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("IV\_CulumnNameType")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

staticFunctionMapV\_AOPM\_C.IV\_CulumnNameType();

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("withoutCulumnNameType")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= staticFunctionMapV\_AOPM\_C.withoutCulumnNameType((String)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("DetaFrontEndUtilbackEndRequest")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= staticFunctionMapV\_AOPM\_C.DetaFrontEndUtilbackEndRequest((String)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("DetaFrontEndUtilbackEndRequest")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= staticFunctionMapV\_AOPM\_C.DetaFrontEndUtilbackEndRequest((String)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("DetaFrontEndUtilhuaRuiJiRequest")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= staticFunctionMapV\_AOPM\_C.DetaFrontEndUtilhuaRuiJiRequest((String)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("DetaFrontEndUtilcacheRequest")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= staticFunctionMapV\_AOPM\_C.DetaFrontEndUtilcacheRequest((String)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("IV\_DB")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

staticFunctionMapV\_AOPM\_C.IV\_DB((String)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("compress")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= staticFunctionMapV\_AOPM\_C.compress((String)inputValues.get(传参因子[因子++])

,(String)inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

if(callFunctionKey.equalsIgnoreCase("uncompress")){

Map<String, Object> inputValues= StaticFunctionMap.preValues(output, 传参因子);

if((boolean) inputValues.get("find")) {

map= staticFunctionMapV\_AOPM\_C.uncompress((byte[])inputValues.get(传参因子[因子++]));

}

StaticFunctionMap.postValues(output, (boolean) inputValues.get("find"), map, callFunctionKey);

};

}

public static void load(StaticFunctionMapV\_AOPM\_E staticFunctionMapV\_AOPM\_E) {

// TODO Auto-generated method stub

//稍后封装

staticFunctionMapV\_AOPM\_E.annotationMap.put("PdeSwapPdcToPde", "PdeSwapPdcToPde");

staticFunctionMapV\_AOPM\_E.annotationMap.put("PdeSwapPdcToPds", "PdeSwapPdcToPds");

staticFunctionMapV\_AOPM\_E.annotationMap.put("PdeSwapPdeToPds", "PdeSwapPdeToPds");

staticFunctionMapV\_AOPM\_E.annotationMap.put("PdeSwapPdsToPde", "PdeSwapPdsToPde");

staticFunctionMapV\_AOPM\_E.annotationMap.put("PdeSwapFixpdcToPde", "PdeSwapFixpdcToPde");

staticFunctionMapV\_AOPM\_E.annotationMap.put("PdeSwapFixpdcToPds", "PdeSwapFixpdcToPds");

staticFunctionMapV\_AOPM\_E.annotationMap.put("PdeSwapFixpdeToPds", "PdeSwapFixpdeToPds");

staticFunctionMapV\_AOPM\_E.annotationMap.put("PdeSwapFixpdsToPde", "PdeSwapFixpdsToPde");

staticFunctionMapV\_AOPM\_E.annotationMap.put("PdeSwapFixtextToPdw", "PdeSwapFixtextToPdw");

staticFunctionMapV\_AOPM\_E.annotationMap.put("PdeSwapFixpdwToPdc", "PdeSwapFixpdwToPdc");

staticFunctionMapV\_AOPM\_E.annotationMap.put("DetaDBUtilDBRequest", "DetaDBUtilDBRequest");

staticFunctionMapV\_AOPM\_E.annotationMap.put("DetaDBUtilbackEndRequest", "DetaDBUtilbackEndRequest");

staticFunctionMapV\_AOPM\_E.annotationMap.put("DetaDBUtilcacheRequest", "DetaDBUtilcacheRequest");

staticFunctionMapV\_AOPM\_E.annotationMap.put("IV\_CulumnNameType", "IV\_CulumnNameType");

staticFunctionMapV\_AOPM\_E.annotationMap.put("withoutCulumnNameType", "withoutCulumnNameType");

staticFunctionMapV\_AOPM\_E.annotationMap.put("DetaFrontEndUtilbackEndRequest", "DetaFrontEndUtilbackEndRequest");

staticFunctionMapV\_AOPM\_E.annotationMap.put("DetaFrontEndUtilbackEndRequest", "DetaFrontEndUtilbackEndRequest");

staticFunctionMapV\_AOPM\_E.annotationMap.put("DetaFrontEndUtilhuaRuiJiRequest", "DetaFrontEndUtilhuaRuiJiRequest");

staticFunctionMapV\_AOPM\_E.annotationMap.put("DetaFrontEndUtilcacheRequest", "DetaFrontEndUtilcacheRequest");

staticFunctionMapV\_AOPM\_E.annotationMap.put("IV\_DB", "IV\_DB");

staticFunctionMapV\_AOPM\_E.annotationMap.put("compress", "compress");

staticFunctionMapV\_AOPM\_E.annotationMap.put("uncompress", "uncompress");

}

//PdeSwap{

public String PdeSwapPdcToPde(String pdc, String lock, String de, String ds, String ie, String is);

public String PdeSwapPdcToPds(String pdc, String lock, String de, String ds, String ie, String is);

//刚检查pds to pde 似乎被猫腻了，我测试下

//罗瑶光 MPOASCEV

public String PdeSwapPdeToPds(String pds, String lock, String de, String ds, String ie, String is);

//把FullDNATokenPDI 类里 do\_PDE\_RNA\_FullFormular\_Back 函数中的 SCEV MPOA 注释的部分 分出来到在这里。

//罗瑶光 VECSAOPM

public String PdeSwapPdsToPde(String pds, String lock, String de, String ds, String ie, String is);

//PdeSwapFix{

public String PdeSwapFixpdcToPde(String pdc, String lock, String de, String ds, String ie, String is);

public String PdeSwapFixpdcToPds(String pdc, String lock, String de, String ds, String ie, String is);

//刚检查pds to pde 似乎被猫腻了，我测试下

//罗瑶光 MPOASCEV

public String PdeSwapFixpdeToPds(String pds, String lock, String de, String ds, String ie, String is);

//把FullDNATokenPDI 类里 do\_PDE\_RNA\_FullFormular\_Back 函数中的 SCEV MPOA 注释的部分 分出来到在这里。

//罗瑶光 VECSAOPM

public String PdeSwapFixpdsToPde(String pds, String lock, String de, String ds, String ie, String is);

public String PdeSwapFixtextToPdw(FullDNATokenPDI pDE\_RNA\_FullFormular, String password) ;

public String PdeSwapFixpdwToPdc(FullDNATokenPDI pDE\_RNA\_FullFormular) ;

//下面这个test demo 展示了 密码-> 肽文-> 肽锁+肽密码-> 密钥,pds和pde密码 -> pds转pde验证-> pde转pds验证

//全部封装成函数

//罗瑶光 20210830

public static void main(String[] argv) {

FullDNATokenPDI pDE\_RNA\_FullFormular= new FullDNATokenPDI();

pDE\_RNA\_FullFormular.text= "控制吸收";

pDE\_RNA\_FullFormular.pdw= PdeSwapFix.textToPdw(pDE\_RNA\_FullFormular, pDE\_RNA\_FullFormular.text);

pDE\_RNA\_FullFormular.code= PdeSwapFix.pdwToPdc(pDE\_RNA\_FullFormular);

System.out.println("肽语: "+ pDE\_RNA\_FullFormular.pdw);

System.out.println("肽锁: "+ pDE\_RNA\_FullFormular.lock);

System.out.println("散列肽语:"+ pDE\_RNA\_FullFormular.code);

////////////////////////////////////////////////////

pDE\_RNA\_FullFormular.doKeyPress(pDE\_RNA\_FullFormular.code, pDE\_RNA\_FullFormular, false);

System.out.println("静态肽展降元概率钥匙E: "+ pDE\_RNA\_FullFormular.pdedeKey);

System.out.println("静态肽展降元概率钥匙S: "+ pDE\_RNA\_FullFormular.pdedsKey);

System.out.println("静态肽展降元: "+ pDE\_RNA\_FullFormular.pds);

System.out.println("静态肽展增元概率钥匙E: "+ pDE\_RNA\_FullFormular.pdeieKey);

System.out.println("静态肽展增元概率钥匙S: "+ pDE\_RNA\_FullFormular.pdeisKey);

System.out.println("静态肽展增元: "+ pDE\_RNA\_FullFormular.pde);

//////////////////////////////////////////////////////

pDE\_RNA\_FullFormular.time= "" + System.currentTimeMillis();

pDE\_RNA\_FullFormular.cacheId= "ID" + Math.random() + ":" + Math.random();

System.out.println("时间: " + pDE\_RNA\_FullFormular.time);

System.out.println("账号随机缓存字符串: " + pDE\_RNA\_FullFormular.cacheId);

/////////////////////////////////////////////////////

pDE\_RNA\_FullFormular.session\_key= pDE\_RNA\_FullFormular.pde;

System.out.println("Session: " + pDE\_RNA\_FullFormular.session\_key);

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

System.out.println("开始前序验证：");

System.out.println("开始Session解析： " + pDE\_RNA\_FullFormular.session\_key);

System.out.println("开始概率钥匙解析：" + pDE\_RNA\_FullFormular.pdedeKey+ pDE\_RNA\_FullFormular.pdedsKey

+ pDE\_RNA\_FullFormular.pdeieKey+ pDE\_RNA\_FullFormular.pdeisKey);

/////////////////////////////////////////////////

FullDNATokenPDI pDE\_RNA\_FullFormular1= new FullDNATokenPDI();

pDE\_RNA\_FullFormular1.pdedeKey= pDE\_RNA\_FullFormular.pdedeKey.toString();

pDE\_RNA\_FullFormular1.pdedsKey= pDE\_RNA\_FullFormular.pdedsKey.toString();

pDE\_RNA\_FullFormular1.pdeieKey= pDE\_RNA\_FullFormular.pdeieKey.toString();

pDE\_RNA\_FullFormular1.pdeisKey= pDE\_RNA\_FullFormular.pdeisKey.toString();

pDE\_RNA\_FullFormular.doKeyUnPress(pDE\_RNA\_FullFormular.code, pDE\_RNA\_FullFormular1, true);

System.out.println();

System.out.println("得到原降元元基DNA序列："+ pDE\_RNA\_FullFormular.pds);

System.out.println("得到新降元元基DNA序列："+ pDE\_RNA\_FullFormular1.pds);

System.out.println("得到原元基DNA序列："+ pDE\_RNA\_FullFormular.pde);

System.out.println("得到新元基DNA序列："+ pDE\_RNA\_FullFormular1.pde);

System.out.println("验证正确？");

System.out.println(pDE\_RNA\_FullFormular.pde.equals(pDE\_RNA\_FullFormular1.pde)? "正确": "失败");

/////////////////////////////////////////

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

System.out.println("开始pde降元验证：");

FullDNATokenPDI pDE\_RNA\_FullFormular2= new FullDNATokenPDI();

pDE\_RNA\_FullFormular2.pdeieKey= pDE\_RNA\_FullFormular.pdeieKey.toString();

pDE\_RNA\_FullFormular2.pdeisKey= pDE\_RNA\_FullFormular.pdeisKey.toString();

pDE\_RNA\_FullFormular2.pdedeKey= pDE\_RNA\_FullFormular.pdeieKey.toString();

pDE\_RNA\_FullFormular2.pdedsKey= pDE\_RNA\_FullFormular.pdeisKey.toString();

System.out.println("准备计算元基DNA序列："+ pDE\_RNA\_FullFormular1.pde);

String pds= PdeSwapFix.pdeToPds(pDE\_RNA\_FullFormular1.pde, "", pDE\_RNA\_FullFormular2.pdedeKey

, pDE\_RNA\_FullFormular2.pdedsKey

, pDE\_RNA\_FullFormular2.pdeieKey

, pDE\_RNA\_FullFormular2.pdeisKey);

System.out.println("pds");

System.out.println("pds");

System.out.println(pDE\_RNA\_FullFormular1.pds);

System.out.println(pds);

//////////////////////////////////////////////////////////////////

System.out.println("开始pds增元验证：");

FullDNATokenPDI pDE\_RNA\_FullFormular3= new FullDNATokenPDI();

pDE\_RNA\_FullFormular3.pdeieKey= pDE\_RNA\_FullFormular.pdeieKey.toString();

pDE\_RNA\_FullFormular3.pdeisKey= pDE\_RNA\_FullFormular.pdeisKey.toString();

pDE\_RNA\_FullFormular3.pdedeKey= pDE\_RNA\_FullFormular.pdeieKey.toString();

pDE\_RNA\_FullFormular3.pdedsKey= pDE\_RNA\_FullFormular.pdeisKey.toString();

String pde= PdeSwapFix.pdsToPde(pDE\_RNA\_FullFormular1.pds, "", pDE\_RNA\_FullFormular3.pdedeKey

, pDE\_RNA\_FullFormular3.pdedsKey

, pDE\_RNA\_FullFormular3.pdeieKey

, pDE\_RNA\_FullFormular3.pdeisKey);

System.out.println("pde");

System.out.println("pde");

System.out.println(pDE\_RNA\_FullFormular1.pde);

System.out.println(pde);

}

//DetaDBUtil {

public String DetaDBUtilDBRequest(String request) throws IOException ;

public String DetaDBUtilbackEndRequest(String request) throws IOException ;

public String DetaDBUtilcacheRequest(String request) throws IOException ;

public void IV\_CulumnNameType() ;

public boolean withoutCulumnNameType(String culumnTypeString) ;

// DetaFrontEndUtil {

public String DetaFrontEndUtilbackEndRequest(String request) throws IOException;

//先不动 稍后归纳 华瑞集rest走 前端还是后端还是数据库。

public String DetaFrontEndUtilhuaRuiJiRequest(String request) throws IOException ;

public String DetaFrontEndUtilcacheRequest(String request) throws IOException ;

//DetaUtil {

public void IV\_DB(String dbConfigPath);

//GzipUtil {

// 压缩

public byte[] compress(byte[] data) throws IOException;

public byte[] compress(String str, String stringTypes) throws IOException;

public byte[] uncompress(byte[] data) throws IOException;

//jogl 画图略

}

共有24组染色体组， 模式同理。超出60页，略

如 16元基进制函数增加了F的催化计算比值

if(IDUQ.charAt(k)== 'F') {//F = E+ C + S, 酸 = H, 碱 = V

if(Math.random()\* 100< VECS) {

OIQ[0][k]= 'H';

}else {

OIQ[0][k]= 'V';

}

}

第五代极快速微分排序增加一个等于号。都有相应的索引计算接口用语言生成调用命令。

private int partition(double[] array, int leftPoint, int rightPoint) {

double x= array[leftPoint]<**=** array[rightPoint]? array[leftPoint]: array[rightPoint];//等于号不能省，见从大到小的老版本，> 的非为 <=，已经在养疗经中测试通过。罗瑶光

int leftPointReflection= leftPoint;

while(leftPointReflection< rightPoint){

//我设立个top2D , --细节竟然没有一个人关注这些细节...20210716

while(!(array[leftPointReflection]> x|| leftPointReflection++ >= rightPoint)) {}

while(array[rightPoint--]> x) {}

if(leftPointReflection< ++rightPoint){

double temp= array[rightPoint];

array[rightPoint]= array[leftPointReflection];

array[leftPointReflection]= temp;

}

}

array[leftPoint]= array[rightPoint];

array[rightPoint]= x;

return rightPoint;

}