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)) {
    //..肱腐蚀， 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 {
        //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 {
```

```
        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 {
        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());
		}
	} }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) {
        //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() {
    }
    //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);
    }
    //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);
    }
    //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) {
        //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);
    //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);
    //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}//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。package
OEU.LYG4DQS4D;import ASQ.PSU.test.TimeCheck;//基于算法导论快排 4 衍生极速小高峰缺陷过滤理论快速排序第 8 代 线性
数字数组排序法函数Java 完整版本实现。//思想:算法导论快排4理论,罗瑶光小高峰过滤理论。//实现:罗瑶光//时间:20140101~
20200711//复制一份 稍后准备 元基新陈代谢优化 public class LYG9DWithDoubleTopSort5D{
        int range;
        int deeps;
        public double[] sort(double[] array, int range, int deeps) {
            this.range= range;
            this.deeps= deeps;
            processDouble(array, 0, array.length- 1, 0);
            return array;
        }
        private void processDouble(double[] array, int leftPoint, int rightPoint, int deep) {
            int c= rightPoint- leftPoint+ 1;
            if(!(c< this.range|| deep> this.deeps)) {//增加了 deep
                int pos= partition(array, leftPoint, rightPoint);
                if(leftPoint< pos- 1) {
                    processDouble(array, leftPoint, pos- 1, deep+ 1);
                }
                if(pos+ 1< rightPoint) {
                    processDouble(array, pos+ 1, rightPoint, deep+ 1);
                }
                return;
            }
            int i= leftPoint;
            for(int j= i+ 1; j< leftPoint+ c; j= i++){
```

```
                while(j> leftPoint){
                    if(array[j]< array[--j]){
                        double temp= array[j+ 1];
                        array[j+ 1]= array[j];
                        array[j]= temp;
                    }
                }
            }
        }
        //养疗经表格出现 关于 xnor 的 =号剔除问题， 这个版本测试成功。已经集成入养疗经
        //从早期把从大到小的>= 的非改为< 当出现大量等值或 0 的例子， 依旧有个别的重名。
        //增加等于后 消除了重名这个问题，我在思考，immutable 的对象比对需要本身，所以这里不是 非的问题，是 Xnor 的问题。
        //罗瑶光
        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;
        }
        public static void main(String[] argv) {
            double[] doubles=new double[9999999];
            for(int i= 0; i< doubles.length; i++) {
                doubles[i]= Math.random();
            }
            LYG9DWithDoubleTopSort5D lYG9DWithDoubleTopSort2D= new LYG9DWithDoubleTopSort5D();
            TimeCheck timecheck=new TimeCheck();
            timecheck.begin();
            lYG9DWithDoubleTopSort2D.sort(doubles, 7, 70);
            timecheck.end();
            timecheck.duration();
            for(int i= 0; i< doubles.length- 1; i++) {
                if(doubles[i]> doubles[i+ 1]) {
                    System.out.println(i+"->"+ doubles[i]);
                }
            }
            System.out.println("end");
```

```
}}package OSI.OPE.SI.SD.SU.SQ.ASU.OSU.PSU.MSU.AVQ.ASQ.ASU.MPE.procedure.pde;

import java.util.HashMap;import java.util.Map;

import SVQ.stable.StableMapsInitons;

@SuppressWarnings("unused")public class RangePDI{
	public static void main(String[] argv) {
		//System.out.println(new RangePDI().IOE(16, 20));
	}
	//240/4 600
	public int[][] IOE(int[][] ps, int VECS) {
		for(int i= 0; i< ps.length; i++) {
			for(int j= 0; j< ps[0].length; j++) {
				String IDUQ= new RangePDI().PDS_P_USQ_ECP_I(ps[i][j], 4);
				int[][] OIQ= new int[1][IDUQ.length()];
				for(int k= 0; k< IDUQ.length(); k++) {
					if(IDUQ.charAt(k)== '2') {
						if(Math.random()* 100> VECS) {
							OIQ[0][k]= 3;
						}else {
							OIQ[0][k]= 1;
						}
					}else {
						OIQ[0][k]= Integer.valueOf(""+ IDUQ.charAt(k));
					}
				}
				ps[i][j]= new InitonsPDS().DO_ACP_IDV(OIQ, 4);
			}
		}
		return ps;
	}
	//#### 元基数字 = 元基符号= 肽展公式元基数字变换 = (肽概率展开数字逻辑集合) #### 1 位 ##### E = I = I =(I)
	//##### F = U = I++ OR Q-- =(I, Q)
	//##### G = Q = Q =(Q)
	//#### 1~2 位
	//##### 0 = D = DD =(D, DD)
	//#### 2 位
	//##### 1 = C = DI =(DI)
	//##### 3 = E = IU, DU =(IU, DU)
	//##### D = V = UQ =(UQ)
	//##### 9 = S = QI =(QI)
	//#### 2~4 位
	//##### 4 = H = (IU, DU) OR DI =(IU, DU, DI) OR (IUDI, DUDI)
	//#### 4 位
	//##### 2 = P = (IU, DU) + DI =(IUDI, DUDI)
	//##### A = O = (IU, DU) + QI =(IUQI, DUQI)
	//##### 7 = A = UQQI =(UQQI)
	//#### 4~6 位
```

```
    //##### 5 = HC- = ((IU, DU) OR DI) + DI =(IUDI, DUDI, DIDI) OR (IUDIDI, DUDIDI)
    //##### B = HE+ = ((IU, DU) OR DI) + (IU, DU) =(IUIU, IUDU, DUIU, DUDU, DIIU, DIDU) OR (IUDIIU, IUDIDU,
DUDIIU, DUDIDU)
    //#### 6~8 位
    //##### 8 = M = ((IU, DU) OR DI) + DI + QI =(IUDIQI, DUDIQI, DIDIQI) OR (IUDIDIQI, DUDIDIQI)
    //##### 6 = X = UQ + ((IU, DU) OR DI) + DI =(UQIUDI, UQDUDI, UQDIDI) OR (UQIUDIDI, UQDUDIDI)
    //##### C = T = UQ + ((IU, DU) OR DI) + (IU, DU) =(UQIUIU, UQIUDU, UQDUIU, UQDUDU, UQDIIU, UQDIDU) OR
(UQIUDIIU, UQIUDIDU, UQDUDIIU, UQDUDIDU)
    public int[][] IPE_AOPM_VECS_IDUQ_TXH(int[][] ps, int VECS) {
        for(int i= 0; i< ps.length; i++) {
            for(int j= 0; j< ps[0].length; j++) {
                String IDUQ= new RangePDI().PDS_P_USQ_ECP_I_17(ps[i][j], 17);
                char[][] OIQ= new char[1][IDUQ.length()];
                for(int k= 0; k< IDUQ.length(); k++) {
                    if(IDUQ.charAt(k)== 'U') {//g          //DIUQ
                        if(Math.random()* 100> VECS) { // 按生化计算来。大于酸 小数，小于碱 大数
                            OIQ[0][k]= 'Q';
                        }else {
                            OIQ[0][k]= IDUQ.charAt(k);
                        }
                    }else if(IDUQ.charAt(k)== 'Q') {//s
                        if(Math.random()* 100< VECS) {
                            OIQ[0][k]= 'D';
                        }else {
                            OIQ[0][k]= IDUQ.charAt(k);
                        }
                    }else if(IDUQ.charAt(k)== 'I') {//s
                        if(Math.random()* 100< VECS) {
                            OIQ[0][k]= 'U';
                        }else {
                            OIQ[0][k]= IDUQ.charAt(k);
                        }
                    }else if(IDUQ.charAt(k)== 'D') {//g
                        if(Math.random()* 100> VECS) {
                            OIQ[0][k]= 'I';
                        }else {
                            OIQ[0][k]=IDUQ.charAt(k);
                        }
                    }else if(IDUQ.charAt(k)== 'V') {//U     //DIUQ
                        if(Math.random()* 100> VECS) {      //SEVC 相对应 //符号写翻了纠正 >20210820
                            OIQ[0][k]= 'C';
                        }else {
                            OIQ[0][k]= 'V';
                        }
                    }else if(IDUQ.charAt(k)== 'E') {//I
                        if(Math.random()* 100< VECS) {
                            OIQ[0][k]= 'V';
                        }else {
                            OIQ[0][k]= 'E';
```

```
                        }
                    }else if(IDUQ.charAt(k)== 'C') {//Q
                        if(Math.random()* 100< VECS) {
                            OIQ[0][k]= 'S';
                        }else {
                            OIQ[0][k]= 'C';
                        }
                    }else if(IDUQ.charAt(k)== 'S') {//D
                        if(Math.random()* 100> VECS) { //符号写翻了纠正 >20210820
                            OIQ[0][k]= 'E';
                        }else {
                            OIQ[0][k]= 'S';
                        }
                    }else {
                        OIQ[0][k]= IDUQ.charAt(k);
                    }
                }
                ps[i][j]= new InitonsPDS().DO_ACP_IDV_17(OIQ, 17);
            }
        }
        return ps;
    }
    //上面似乎又被猫腻了两个符号， 统一 VECS 小数是碱 大数是酸。 按标准肽展公式模拟下计算机视觉
    public int[][] IPE_AOPM_VECS_IDUQ_TXH_AC(int[][] ps, int VECS) {
        for(int i= 0; i< ps.length; i++) {
            for(int j= 0; j< ps[0].length; j++) {
                //String IDUQ= new RangePDI().PDS_P_USQ_ECP_I_17(ps[i][j], 17);
                String IDUQ= new RangePDI().PDS_P_USQ_ECP_I_16(ps[i][j], 16);
                char[][] OIQ= new char[1][IDUQ.length()];
                for(int k= 0; k< IDUQ.length(); k++) {
                    if(IDUQ.charAt(k)== 'U') {//g
                        if(Math.random()* 100< VECS) {
                            OIQ[0][k]= 'Q';
                        }else {
                            OIQ[0][k]= 'U';
                        }
                    }else if(IDUQ.charAt(k)== 'Q') {//s
                        if(Math.random()* 100< VECS) {
                            OIQ[0][k]= 'Q';
                        }else {
                            OIQ[0][k]= 'U';
                        }
                    }else if(IDUQ.charAt(k)== 'I') {//s
                        if(Math.random()* 100< VECS) {
                            OIQ[0][k]= 'I';
                        }else {
                            OIQ[0][k]= 'D';
                        }
                    }else if(IDUQ.charAt(k)== 'D') {//g
```

```
            if(Math.random()* 100< VECS) {
                OIQ[0][k]= 'I';
            }else {
                OIQ[0][k]= 'D';
            }
        }else if(IDUQ.charAt(k)== 'V') {//U      //DIUQ
            if(Math.random()* 100< VECS) {       //SEVC 相对应
                OIQ[0][k]= 'C';
            }else {
                OIQ[0][k]= 'V';
            }
        }else if(IDUQ.charAt(k)== 'E') {//I
            if(Math.random()* 100< VECS) {
                OIQ[0][k]= 'E';
            }else {
                OIQ[0][k]= 'S';
            }
        }else if(IDUQ.charAt(k)== 'C') {//Q
            if(Math.random()* 100< VECS) {
                OIQ[0][k]= 'C';
            }else {
                OIQ[0][k]= 'V';
            }
        }else if(IDUQ.charAt(k)== 'S') {//D
            if(Math.random()* 100< VECS) {
                OIQ[0][k]= 'E';
            }else {
                OIQ[0][k]= 'S';
            }
        }else if(IDUQ.charAt(k)== 'A') {//A = V + S, 酸 = C + E = P,  碱 = V + S = A
            if(Math.random()* 100< VECS) {
                OIQ[0][k]= 'P';
            }else {
                OIQ[0][k]= 'A';
            }
        }else if(IDUQ.charAt(k)== 'O') {//O = E + S, 酸= E + E = E ,  碱 = V + S = A
            if(Math.random()* 100< VECS) {
                OIQ[0][k]= 'E';
            }else {
                OIQ[0][k]= 'A';
            }
        }else if(IDUQ.charAt(k)== 'P') {//P = E + C, 酸 = E + C = P,  碱 = S + V = A
            if(Math.random()* 100< VECS) {
                OIQ[0][k]= 'P';
            }else {
                OIQ[0][k]= 'A';
            }
        }else if(IDUQ.charAt(k)== 'M') {//M = C + S, 酸 = C + E = P,  碱 = V + S = A
            if(Math.random()* 100< VECS) {
```

```
                                OIQ[0][k]= 'P';
                            }else {
                                OIQ[0][k]= 'A';
                            }
                        }else 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';
                            }
                        }else {
                            OIQ[0][k]= IDUQ.charAt(k);
                        }
                    }
                    //ps[i][j]= new InitonsPDS().DO_ACP_IDV_17(OIQ, 17);
                    ps[i][j]= new InitonsPDS().DO_ACP_IDV_16(OIQ, 16);
                }
            }
            return ps;
        }
        public int[][] IPE(int[][] ps, int VECS) {
            for(int i= 0; i< ps.length; i++) {
                for(int j= 0; j< ps[0].length; j++) {
                    String IDUQ= new RangePDI().PDS_P_USQ_ECP_I(ps[i][j], 4);
                    int[][] OIQ= new int[1][IDUQ.length()];
                    for(int k= 0; k< IDUQ.length(); k++) {
                        if(IDUQ.charAt(k)== '2') {//g
                            if(Math.random()* 100> VECS) {
                                OIQ[0][k]= 3;
                            }else {
                                OIQ[0][k]= Integer.valueOf(""+ IDUQ.charAt(k));
                            }
                        }else if(IDUQ.charAt(k)== '3') {//s
                            if(Math.random()* 100< VECS) {
                                OIQ[0][k]= 0;
                            }else {
                                OIQ[0][k]= Integer.valueOf(""+ IDUQ.charAt(k));
                            }
                        }else if(IDUQ.charAt(k)== '1') {//s
                            if(Math.random()* 100< VECS) {
                                OIQ[0][k]= 2;
                            }else {
                                OIQ[0][k]= Integer.valueOf(""+ IDUQ.charAt(k));
                            }
                        }else if(IDUQ.charAt(k)== '0') {//g
                            if(Math.random()* 100> VECS) {
                                OIQ[0][k]= 1;
                            }else {
                                OIQ[0][k]= Integer.valueOf(""+ IDUQ.charAt(k));
```

```
                                }
                            }
                        }
                        ps[i][j]= new InitonsPDS().DO_ACP_IDV(OIQ, 4);
                    }
                }
                return ps;
            }
            public int[][] QPE(int[][] ps, int VECS) {
                for(int i= 0; i< ps.length; i++) {
                    for(int j= 0; j< ps[0].length; j++) {
                        String IDUQ= new RangePDI().PDS_P_USQ_ECP_I(ps[i][j], 4);
                        int[][] OIQ= new int[1][IDUQ.length()];
                        for(int k= 0; k< IDUQ.length(); k++) {
                            if(IDUQ.charAt(k)== '0') {//g D I U Q
                                if(Math.random()* 100> VECS) {
                                    OIQ[0][k]= 1;
                                }else {
                                    OIQ[0][k]= 0;
                                }
                            }else if(IDUQ.charAt(k)== '1') {//s
                                if(Math.random()* 100> VECS) {
                                    OIQ[0][k]= 1;
                                }else {
                                    OIQ[0][k]= 2;
                                }
                            }else if(IDUQ.charAt(k)== '2') {//s
                                if(Math.random()* 100> VECS) {
                                    OIQ[0][k]= 3;
                                }else {
                                    OIQ[0][k]= 0;
                                }
                            }else if(IDUQ.charAt(k)== '3') {//g
                                if(Math.random()* 100> VECS) {
                                    OIQ[0][k]= 3;
                                }else {
                                    OIQ[0][k]= 0;
                                }
                            }
                        }
                        ps[i][j]= new InitonsPDS().DO_ACP_IDV(OIQ, 4);
                    }
                }
                return ps;
            }

            public String PDSEncode(String VSQ) {
                while(VSQ.length()> 0){
                //      VSQ.concat(VSQ.replace("", ""))> 0;
```

```
        }
        return VSQ;
    }
    public String PDS_OEC_IID(String VSQ_IIE, int OCI_PPE) {
        String $_CID= "AOPMVECS";
        //VSQ_IIE= $_CID.charAt(VSQ_IIE.length());
        return VSQ_IIE;
    }
    public String PDS_P_USQ_ECP(int P_VSQ, int MSP) {
        String ISQ= "";
        while(P_VSQ> 0) {
            ISQ+= P_VSQ/ MSP;
            P_VSQ%= MSP;
        }
        ISQ+= P_VSQ;
        return ISQ;
    }
    //64/4    8/4 2
    public String PDS_P_USQ_ECP_I(int P_VSQ, int MSP) {
        String ISQ= "";
        while(P_VSQ>= MSP) {
            ISQ+= P_VSQ/ MSP;
            P_VSQ%= MSP;
        }
        ISQ+= P_VSQ;
        return ISQ;
    }
    //现在 PDS 函数太大了，不想 new，之后会用 DecadeToPDS 函数
    public String PDS_P_USQ_ECP_I_17(int P_VSQ, int MSP) {
        Map<String, String> initonsSet= new HashMap<>();
        initonsSet.put("0", "0");
        initonsSet.put("1", "1");
        initonsSet.put("2", "2");
        initonsSet.put("3", "3");
        initonsSet.put("4", "4");
        initonsSet.put("5", "5");
        initonsSet.put("6", "6");
        initonsSet.put("7", "7");
        initonsSet.put("8", "8");
        initonsSet.put("9", "9");
        initonsSet.put("10", "A");
        initonsSet.put("11", "B");
        initonsSet.put("12", "C");
        initonsSet.put("13", "D");
        initonsSet.put("14", "E");
        initonsSet.put("15", "F");
        initonsSet.put("16", "G");
        Map<String, String> initonsCode= new HashMap<>();
        initonsCode.put("0", "D");
```

```
        initonsCode.put("1", "C");
        initonsCode.put("2", "P");
        initonsCode.put("3", "E");
        initonsCode.put("4", "H");
        initonsCode.put("5", "-");
        initonsCode.put("6", "X");
        initonsCode.put("7", "A");
        initonsCode.put("8", "M");
        initonsCode.put("9", "S");
        initonsCode.put("A", "O");
        initonsCode.put("B", "+");
        initonsCode.put("C", "T");
        initonsCode.put("D", "V");
        initonsCode.put("E", "I");
        initonsCode.put("F", "U");
        initonsCode.put("G", "Q");
        String ISQ= "";
        while(P_VSQ>= MSP) {
            ISQ+= initonsCode.get(initonsSet.get(""+ P_VSQ/ MSP));// P_VSQ;
            P_VSQ%= MSP;
        }
        ISQ+= initonsCode.get(initonsSet.get(""+ P_VSQ));// P_VSQ;
        return ISQ;
    }
    public String PDS_P_USQ_ECP_I_16(int P_VSQ, int MSP) {
        Map<String, String> initonsSet= new HashMap<>();
        initonsSet.put("0", "0");
        initonsSet.put("1", "1");
        initonsSet.put("2", "2");
        initonsSet.put("3", "3");
        initonsSet.put("4", "4");
        initonsSet.put("5", "5");
        initonsSet.put("6", "6");
        initonsSet.put("7", "7");
        initonsSet.put("8", "8");
        initonsSet.put("9", "9");
        initonsSet.put("10", "A");
        initonsSet.put("11", "B");
        initonsSet.put("12", "C");
        initonsSet.put("13", "D");
        initonsSet.put("14", "E");
        initonsSet.put("15", "F");
        Map<String, String> initonsCode= new HashMap<>();
        initonsCode.put("0", "D");
        initonsCode.put("1", "C");
        initonsCode.put("2", "P");
        initonsCode.put("3", "E");
        initonsCode.put("4", "T");
        initonsCode.put("5", "H");
```

```
        initonsCode.put("6", "O");
        initonsCode.put("7", "S");
        initonsCode.put("8", "M");
        initonsCode.put("9", "A");
        initonsCode.put("A", "X");
        initonsCode.put("B", "F");
        initonsCode.put("C", "V");
        initonsCode.put("D", "I");
        initonsCode.put("E", "U");
        initonsCode.put("F", "Q");
        String ISQ= "";
        while(P_VSQ>= MSP) {
            ISQ+= initonsCode.get(initonsSet.get(""+ P_VSQ/ MSP));// P_VSQ;
            P_VSQ%= MSP;
        }
        ISQ+= initonsCode.get(initonsSet.get(""+ P_VSQ));// P_VSQ;
        return ISQ;
    }
    public String PDS_P_USQ_ECP_I_17_Stable(int P_VSQ, int MSP) {
        String ISQ= "";
        while(P_VSQ>= MSP) {
            ISQ+= StableMapsInitons.initonsCode.get(StableMapsInitons.initonsSet.get(""+ P_VSQ/ MSP));// P_VSQ;
            P_VSQ%= MSP;
        }
        ISQ+= StableMapsInitons.initonsCode.get(StableMapsInitons.initonsSet.get(""+ P_VSQ));// P_VSQ;
        return ISQ;
    }
    public String ESU_ECS_SVQ_PDS_OEU(String SQA) {
        String[] PDS= new String[]{"AOPM", "VECS"};
        String ISQ_PSD= "";
        while(injectPDI(ISQ_PSD).length()> 0) {
            ISQ_PSD+= PDSEncode(SQA);
        }
        return ISQ_PSD;
    }
    public String ESU_P_SEQ_PDS_OEU(String SQA) {
        String[] PDS= new String[]{"AOPM", "VECS", "DIUQ", "HTX"};
        String ISQ_PSD= "";
        int i= 0;
        while(injectPDI(ISQ_PSD).length()> 0) {
            ISQ_PSD+= PDSEncode(SQA);
            SQA+= PDSEncode(PDS[i]);
        }
        return ISQ_PSD;
    }
}}
```