第十九章DNA 元基染色体

第一节 元基造字

//今天开始造字。 语法 = 语义.生化
//我准备将语义部分用部首偏旁组合造字.

零/D
一/C
二/P
三/E
四/H
五/HC
六/X
七/A
八/M
九/S
十/O
十一/HE
十二/T
十三/V
十四/I
十五/U
十六/Q
//
金/H.AQT
木/H.OEU
水/H.MXS
火/H.PVD
土/H.CDI
//
休/XMS.OU
生/EIX.TS
殇/HOE.IP
杜/VUH.AQ
景/VPD.DH
死/DDC.DE
惊/CAT.CX
开/TQS.MV
//
酸/VUI.AQ
甘/VUI.PI
苦/VUI.DH
辣/VUI.CX
咸/VUI.OU
涩/VUI.MV
平/VUD.ST
腻/VUI.ED

天干:甲、乙、丙、丁。戊、己、庚、辛、壬、癸、
五行:木、木、火、火、土、土、金、金、水、水、
数字:一、二、三、四、五、六、七、八、九、十、

甲/H.OEU.C
乙/H.OEU.P
丙/H.PVD.E
丁/H.PVD.H
戊/H.CDI.HC
己/H.CDI.X
庚/H.AQT.A
辛/H.AQT.M
壬/H.MXS.S
癸/H.MXS.O

子丑寅卯辰巳午未申酉戌亥
水土木木土火火土金金土水
11-12-1-2-3-4-5-6-7-8-9-10
HE-T-C-P-E-H-HC-X-A-M-S-O
鼠牛虎兔龙蛇马羊猴鸡狗猪 ·

//明朝刘基著 太乙六壬遁甲:地支属性如下/修正了下同元连元基不化简
子/葵水/H.MXS.O.H.MXS.HE=H.MXS.O.HE
丑/巳土/H.CDI.X.H.CDI.T=H.CDI.X.T
寅/甲木/H.OEU.C.H.OEU.C=H.OEU.C.C
卯/乙木/H.OEU.P.H.OEU.P=H.OEU.P.P
辰/戊土/H.CDI.HC.H.CDI.E=H.CDI.HC.E
巳/丙火/H.PVD.E.H.PVD.H=H.PVD.E.H
午/丁火/H.PVD.H.H.PVD.HC=H.PVD.H.HC
未/巳土/H.CDI.X.H.CDI.X=H.CDI.X.X
申/庚金/H.AQT.A.H.AQT.A=H.AQT.A.A
酉/辛金/H.AQT.M.H.AQT.M=H.AQT.M.M
戌/戊土/H.CDI.HC.H.CDI.S=H.CDI.HC.S
亥/壬水/H.MXS.S.H.MXS.O=H.MXS.S.O

上面来自第二卷281页，286页，311页．和第三卷的十七进制推导．提供了参考，
布好局先．

罗瑶光

为了让我的git记录查询方便，我准备每10几个字 上传备份一次，按康熙字典来。

//因为偏旁部首 很多意思和字的解释 已经无效。所以我的造字定义思路是五行+八卦+双元

忄心　竖心旁 按心字计/H.PVD.VPD.DH.VUI.DH= PVD.DH.VUI
氵水　三点旁 按水字计/H.MXS.XMS.OU.VUI.OU= MXS.OU.VUI
犭犬　犬字旁 按犬字计/H.OEU.HOE.IP.VUI.AQ= OEU.IP.VUI.AQ
礻示　半礼旁 按示字计/H.AQT.CAT.CX.VUI.CX= AQT.CAT.CX.VUI
王玉　斜玉旁 按玉字计/H.OEU.VUH.AQ.VUI.PI= OEU.VUH.AQ.VUI.PI
艹草　草字头 按草字计/H.OEU.VUH.AQ.VUI.AQ= OEU.VUH.AQ.VUI.AQ
衤衣　衣字旁 按衣字计/H.CDI.EIX.TS= CDI.EIX.TS
月肉　肉字旁 按肉字计/H.PVD.H.CDI.EIX.TS= PVD.CDI.EIX.TS
辶走　走马旁 按走字计/H.OEU.H.AQT.TQS.MV= OEU.H.AQT.TQS.MV
阝邑　右耳旁 按邑字计/H.CDI.H.AQT.EIX.TS= CDI.H.AQT.EIX.TS
扌手　提手旁 按手字计/H.AQT.TQS.MV= AQT.TQS.MV
阝卓　左耳旁 按卓字计/H.PVD.VPD.DH.VUI.PI= PVD.DH.VUI.PI

整理后如下:

```
//计数
零/D
一/C
二/P
三/E
四/H
五/HC
六/X
七/A
八/M
九/S
十/O
十一/HE
十二/T
十三/V
十四/I
十五/U
十六/Q

//五行
金/AQT
木/OEU
水/MXS
火/PVD
土/CDI

//生化
酸/I.AQ
甘/I.PI
苦/I.DH
辣/I.CX
咸/I.OU
涩/I.MV
平/D.ST
腻/I.ED
```

```
//八卦
休/XMS.OU
生/EIX.TS
殇/HOE.IP
杜/VUH.AQ
景/VPD.DH
死/DDC.DE
惊/CAT.CX
开/TQS.MV

//方位
坎/XMS.I.OU=XMS.IOU
艮/EIX.D.ST=EIX.DST
震/HOE.I.IP=HOE.IIP
巽/VUH.I.AQ=VUH.IAQ
离/VPD.I.DH=VPD.IDH
坤/DDC.I.DE=DDC.IDE
兑/CAT.I.CX=CAT.ICX
乾/TQS.I.MV=TQS.IMV
//

//方向=方位+计数
东/HOE.I.IP.C=HOE.IIP.C
南/VPD.I.DH.P=VPD.IDH.P
西/CAT.I.CX.E=CAT.ICX.E
北/XMS.I.OU.H=XMS.IOU.H

//四季=八卦+计数
春/HOE.IP.C=HOE.IPC
夏/VPD.DH.P=VPD.DHP
秋/CAT.CX.E=CAT.CXE
冬/XMS.OU.H=XMS.OUH


//五行计数 天干
甲/OEU.C
乙/OEU.P
丙/PVD.E
丁/PVD.H
戊/CDI.HC
己/CDI.X
庚/AQT.A
辛/AQT.M
壬/MXS.S
癸/MXS.O
```

//五行计数 地支
子/MXS.O.HE=MXS.OHE
丑/CDI.X.T=CDI.XT
寅/OEU.C.C=OEU.CC
卯/OEU.P.P=OEU.PP
辰/CDI.HC.E=CDI.HCE
巳/PVD.E.H=PVD.EH
午/PVD.H.HC=PVD.HHC
未/CDI.X.X=CDI.XX
申/AQT.A.A=AQT.AA
酉/AQT.M.M=AQT.MM
戌/CDI.HC.S=CDI.HCS
亥/MXS.S.O=MXS.SO
//
亻字计/PVD.DH.VUI
氵字计/MXS.OU.VUI
犭字计/OEU.IP.VUI.AQ
礻字计/AQT.CAT.CX.VUI
王字计/OEU.VUH.AQ.VUI.PI
艹字计/OEU.VUH.AQ.VUI.AQ
衤字计/CDI.EIX.TS
月字计/PVD.CDI.EIX.TS
辶字计/OEU.H.AQT.TQS.MV
阝字计/CDI.H.AQT.EIX.TS
扌字计/AQT.TQS.MV
阝字计/PVD.DH.VUI.PI
//
心字计/PVD.DH.VUI
水字计/MXS.OU.VUI
犬字计/OEU.IP.VUI.AQ
示字计/AQT.CAT.CX.VUI
玉字计/OEU.VUH.AQ.VUI.PI
草字计/OEU.VUH.AQ.VUI.AQ
衣字计/CDI.EIX.TS
肉字计/PVD.CDI.EIX.TS
走字计/OEU.H.AQT.TQS.MV
邑字计/CDI.H.AQT.EIX.TS
手字计/AQT.TQS.MV
卓字计/PVD.DH.VUI.PI

等/VSQ
加/VSI
减/VSD
非/VSU

山/土金/CDI.AQT
日/土火/CDI.PVD
泥/土水/CDI.MXS
棺/土木/CDI.OEU

花/木火/OEU.PVD
药/木水/OEU.MXS
根/木土/OEU.CDI
机/木金/OEU.AQT

雪/水金/MXS.AQT
泽/水土/MXS.CDI
鸟/水木/MXS.OEU
温/水火/MXS.PVD

风/金木/AQT.OEU
墨/金水/AQT.MXS
岩/金土/AQT.CDI
光/金火/AQT.PVD

神/火金/PVD.AQT
妖/火木/PVD.OEU
魔/火水/PVD.MXS
怪/火土/PVD.CDI

//稍后
丙(火)/H.PVD.
代(火)/H.PVD.
旦(火)/H.PVD.
叨(火)/H.PVD.
氐(火)/H.PVD.
叮(火)/H.PVD.
冬(火)/H.PVD.(康熙字典上竟然归纳冬为火，先不管)
叻(火)/H.PVD.
立(火)/H.PVD.
旭(火)/H.PVD.

//十七进制由来

```
package OSI.SSI.ASU.OSU.PSU.MSU.ouler;

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

public class Q_OulerRing{
	//这段函数用于观测元基映射的欧拉回路模型
	//思想 罗瑶光
	//算法 欧拉
	//程序员 罗瑶光
	//QUIVT+OSMAX-HEPCD 9 结果输出 有很多，我先任意选一种，就这个了 参考第48行。

//下面是所有的输出结果，没时间耗在这上面。
//	HEPCD3
//	2
//	1
//	UIVT+OSMAX-11
//	10
//	9
//	8
//	Q8
//	7
//	6
//	5
//	4
//	3
//	2
//	1
//	0
//
//	AMSO+HEPCD8
//	7
//	6
//	UIVT9
//	8
//	7
//	Q7
//	6
//	5
//	-X6
//	5
//	4
//	3
//	2
//	1
//	0
//
//、、、、、、、、、、、、、、、、、、、、、、、、--》》》》》》 OSMAX-HEPCD9
```

```
public static void main(String[] args) {
        //init AOPM VECS IDUQ TXH DD
        //初始环路
        Map<String, Boolean> initonsLink= new HashMap<>();
        //环路探索
        Map<String, Boolean> didInitonsLink= new HashMap<>();
        initonsLink.put("DC", true);
        initonsLink.put("CD", true);
        initonsLink.put("IV", true);
        initonsLink.put("VI", true);
        initonsLink.put("IU", true);
        initonsLink.put("UI", true);
        initonsLink.put("UE", true);
        initonsLink.put("EU", true);
        initonsLink.put("UQ", true);
        initonsLink.put("QU", true);
        initonsLink.put("QS", true);
        initonsLink.put("SQ", true);

        initonsLink.put("VT", true);
        initonsLink.put("TV", true);
        initonsLink.put("ET", true);
        initonsLink.put("TE", true);
        initonsLink.put("EH", true);
        initonsLink.put("HE", true);
        initonsLink.put("EP", true);
        initonsLink.put("PE", true);
        initonsLink.put("H+", true);//HE + HC -
        initonsLink.put("+H", true);
        initonsLink.put("H-", true);
        initonsLink.put("-H", true);
        initonsLink.put("HC", true);
        initonsLink.put("CH", true);
        initonsLink.put("CP", true);
        initonsLink.put("PC", true);
        initonsLink.put("SM", true);
        initonsLink.put("MS", true);
        initonsLink.put("SO", true);
        initonsLink.put("OS", true);

        initonsLink.put("XA", true);
        initonsLink.put("AX", true);
        initonsLink.put("MA", true);
        initonsLink.put("AM", true);
        initonsLink.put("X-", true);
        initonsLink.put("-X", true);
        initonsLink.put("M-", true);
        initonsLink.put("-M", true);
        initonsLink.put("T+", true);
        initonsLink.put("+T", true);
        initonsLink.put("O+", true);
        initonsLink.put("+O", true);
```

```
		String[] initons= new String[]{"H", "A", "O", "P", "M", "V", "E", "C", "S", "I", "D", "U", "Q", "T",
"X", "+", "-"};
		int[] initonsCount= new int[17];
		//for loop
		//开始计算 路径总数
		//String didInitons= "";
 int count= 0;
		for(int i= 0; i< initons.length; i++) {
			//System.out.println(temp);
			System.out.print(initons[i]);
			initonsCount[i]++;
			recur(initons[i], initonsLink, didInitonsLink, initons, initonsCount, count, i);
			//System.out.println(count);
			//下一个
			count= 0;
			System.out.println();
			didInitonsLink.clear();
			initonsCount= new int[17];
		}
		// print loop initons
		//打印可能模式
		System.out.println(count);
	}
	//递归 继续修改。等会加 隔开观测。
	public static void recur(String firstChar, Map<String, Boolean> initonsLink
			, Map<String, Boolean> didInitonsLink, String[] initons, int[] initonsCount, int count, int i) {
		for(int j= 0; j< initons.length; j++) {
			if(!firstChar.equals(initons[j])) {
				String temp= ""+ firstChar+ initons[j];
				//有路径  //没有遍历 //遍历了两次
				if(initonsLink.containsKey(temp)&& !didInitonsLink.containsKey(temp)&&
initonsCount[j]< 1) {
					initonsCount[j]++;
					didInitonsLink.put(""+ firstChar+ initons[j], true);
					System.out.print(initons[j]);
					recur(initons[j], initonsLink, didInitonsLink, initons, initonsCount, count+ 1,
j);
					System.out.println(count);
				}
			}
		}
	}
}
```

第二节 最新笔记:

This project bases on the Extension project of DETA Socket PLSQL DB.

20210320 Initon Math Yaoguang Luo
20210320 元基数学 罗瑶光

自从有了AOPM VECS IDUQ TXH DD ，16个元基成分，我今天定义为16进制的数字，对应为

既然是严谨定义， 自然要用生化和语义双元基罗盘来进行推导开始。
我先设未知的为X
A XXXX
O XXXX
P XXXX
M XXXX
V XXXX
E XXXX
C XXXX
S XXXX
I XXX1
D XXX0
U XXX2
Q XXX3
T XXXX
X XXXX
HE XXXX
HC XXXX
DD 补码

根据第一卷 和 第二卷283和284页，我能列出来的 新增 关系式 E -> HE，C -> HC.

根据 数字逻辑 和 离散数学 位列比 和 寄存法则 推导 VECS 为 ：
A XXXX
O XXXX
P XXXX
M XXXX

V XXX1
E XXX2
C XXX0
S XXX3

I 0001
D 0000
U 0002
Q 0003

T XXXX
X XXXX
HE XXX2
HC XXX0

DD 补码

准备写个欧拉路径算法开始计算 。第一卷的 数据预测 包 此时派上了用场。

方便大家理解。

刚计算了欧拉元基环路 QUIVT+OSMAX-HEPCD
我定义17进制的数据为
QUIVT+OSMAX-HEPCD DD
GFEDCBA9876543210 CARRY
我在思考怎么缩进成16进制。
先保证逻辑的严谨性，
我先用17进制走一段路程。

17进制数据已经问世，我今天深入下，进行元基础加法 探索，wechat已经发布了，在这里整理如下；
元基础数字 = 元基符号 = 生化名称
0 = D = 胞嘧啶
1 = C = 鸟嘌呤
2 = P = 尿胞变鸟苷
3 = E = 尿变嘌呤
4 = H = 黄嘌呤
5 = - =
6 = X = 变感腺鸟苷
7 = A = 变感腺腺苷
8 = M = 鸟腺苷
9 = S = 腺嘌呤
A = O = 尿胞变腺苷
B = + =
C = T = 变感腺尿变苷
D = V = 变感腺嘌呤
E = I = 尿嘧啶
F = U = 变嘧啶
G = Q = 胸腺嘧啶

数字逻辑的推导(C=U+D+D)
语义肽展公式推导
元基数字 =元基符号 = 肽展公式数字变换
0 = D = 0 + 0
1 = C = 0 + F
2 = P = 3 + 1
3 = E = F + 0
4 = H = 3 OR 1
5 = - =
6 = X = D + -
7 = A = D + 9
8 = M = - + 9
9 = S = G + E
A = O = 3 + 9
B = + =
C = T = D + 3
D = V = F + G
E = I =
F = U =
G = Q =

元基数学加法表 根据 4 的归纳完整推导如下
元基数字 = 元基符号= 肽展公式数字变换
0 = D = 0 + 0
1 = C = 0 + F
2 = P = 3 + 1
3 = E = F + 0
4 = H = 3 OR 1
5 = - = 4 + 1
6 = X = D + 5
7 = A = D + 9
8 = M = 5 + 9
9 = S = G + E
A = O = 3 + 9
B = + = 4 + 3
C = T = D + B
D = V = F + G
E = I = E
F = U = E++ OR G--
G = Q = G

20210322 今早把十七进制的元基组合数学变换 定义了，归纳整理如下：
我的思路是 元基稳定化DEFG变换
元基数字 = 元基符号= 肽展公式元基变换
0 = D = 00
1 = C = 02
2 = P = 2002
3 = E = 20
4 = H = 20, 02
5 = - = 2002, 0202
6 = X = 23(2002, 0202)
7 = A = 2331
8 = M = (2002, 0202)31
9 = S = 31
A = O = 2031
B = + = 2020, 0220
C = T = 23(2020, 0220)
D = V = 23
E = I = 1
F = U = 2
G = Q = 3
稳定化后于是元基替换为 0123-> DIUQ 如下

0 = D = D + D
1 = C = DU
2 = P = UDDU
3 = E = UD
4 = H = UD, DU
5 = - = (UD, DU)DU
6 = X = UQ(UD, DU)DU
7 = A = UQQI
8 = M = (UD, DU)DUQI
9 = S = QI
A = O = UDQI
B = + = (UD, DU)UD
C = T = UQ(UD, DU)UD
D = V = UQ
E = I = I
F = U = U
G = Q = Q

我在思考 这个括号内的元基如果进行之后计算的唯一化。
到现在 十进制常数进行元基码 变换的思路已经问世了，下一步，养疗经真实应用。
这里的 568B 我推测又是一组概率钥匙酸碱控制。我也会真实应用测试论证。

今天多做一点推导：我把0到G的欧拉顺序 改成 线性数学顺序观测如下：
0 = D = D + D
E = I = I
F = U = U
G = Q = Q
1 = C = DU
D = V = UQ
3 = E = UD
9 = S = QI
7 = A = UQQI
2 = P = UDDU
A = O = UDQI
4 = H = UD, DU
5 = - = (UD, DU)DU
B = + = (UD, DU)UD
6 = X = UQ(UD, DU)DU
C = T = UQ(UD, DU)UD
8 = M = (UD, DU)DUQI

```
#### 我想这个顺序别有用途，先搁置。
#### 下一步 H 化简HE+, HC-, 然后重新线性排列如下
##### 0 =                D  =    D
##### E =                I  =    I
##### F =                U  =    U
##### G =                Q  =    Q
##### 1 =                C  =    DU
##### 3 =                E  =    UD
##### 4 =                H  =    UD, DU
##### D =                V  =    UQ
##### 9 =                S  =    QI
##### 5 =                -  =    DUDU
##### 2 =                P  =    UDDU
##### B =                +  =    UDUD
##### A =                O  =    UDQI
##### 7 =                A  =    UQQI
##### 8 =                M  =    DUDUQI
##### 6 =                X  =    UQDUDU
##### C =                T  =    UQUDUD

修正后如下
##### 0 =                D  =    D
##### E =                I  =    I
##### F =                U  =    U
##### G =                Q  =    Q
##### 1 =                C  =    DI
##### 3 =                E  =    UD
##### 4 =                H  =    UD, DU
##### D =                V  =    UQ
##### 9 =                S  =    QI
##### 5 =                -  =    DUDU
##### 2 =                P  =    UDDU
##### B =                +  =    UDUD
##### A =                O  =    UDQI
##### 7 =                A  =    UQQI
##### 8 =                M  =    DUDUQI
##### 6 =                X  =    UQDUDU
##### C =                T  =    UQUDUD

修正下 C=DU改成 DI, 因为肽展公式(补码计算) C= DDU, DD是补码
### 肽展公式的推导(肽展计算)(C=I+D)
### 开始语义肽展公式验证### 元基数学加法表 根据 4 的归纳完整推导如下
### 元基数字  =   元基符号=   肽展公式元基变换
#### 0  =                D  =    D + D
#### 1  =                C  =    I + D
#### 2  =                P  =    E + C
#### 3  =                E  =    I + U, D + U
#### 4  =                H  =    E OR C
#### 5  =                -  =    H + C
#### 6  =                X  =    V + HC
#### 7  =                A  =    V + S
#### 8  =                M  =    HC + S

#### 9  =                S  =    Q + I
#### A  =                O  =    E + S
#### B  =                +  =    H + E

#### C  =                T  =    V + HE
#### D  =                V  =    U + Q
#### E  =                I  =    I
#### F  =                U  =    I++ OR Q--
#### G  =                Q  =    Q
```

###于是元基数字归纳
元基数字 = 元基符号= 肽展公式元基数字变换
0 = D = 0 + 0
1 = C = 1 + 0
2 = P = (12, 02) + 10
3 = E = 1 + 2, 0 + 2
4 = H = (12, 02) OR 10
5 = HC = (12, 02) OR 10 + 10
6 = X = 23 + (12, 02) OR 10 + 10
7 = A = 23 + 31
8 = M = (12, 02) OR 10 + 10 + 31

9 = S = 3 + 1
A = O = (12, 02) + 31
B = HE = (12, 02) OR 10 + 12, 02

C = T = 23 + (12, 02) OR 10 + (12, 02)
D = V = 2 + 3
E = I = 1
F = U = 1++ OR 3--
G = Q = 3

###于是元基肽展归纳如下
元基数字 = 元基符号= 肽展公式元基数字变换
0 = D = D + D
1 = C = I + D
2 = P = (IU, DU) + ID
3 = E = I + U, D + U
4 = H = (IU, DU) OR ID
5 = HC = (IU, DU) OR ID + ID
6 = X = UQ + (IU, DU) OR ID + ID
7 = A = UQ + QI
8 = M = (IU, DU) OR ID + ID + QI

9 = S = Q + I
A = O = (IU, DU) + QI
B = HE = (IU, DU) OR ID + (IU, DU)

C = T = UQ + (IU, DU) OR ID + (IU, DU)
D = V = U + Q
E = I = I
F = U = I++ OR Q--
G = Q = Q

###开始整理
元基数字 = 元基符号= 肽展公式元基数字变换
0 = D = DD
1 = C = ID
2 = P = IUID, DUID
3 = E = IU, DU
4 = H = (IU, DU) OR ID
5 = HC = (IU, DU) OR ID + ID
6 = X = UQ + (IU, DU) OR ID + ID
7 = A = UQQI
8 = M = (IU, DU) OR ID + ID + QI

9 = S = QI
A = O = (IU, DU) + QI
B = HE = (IU, DU) OR ID + (IU, DU)

C = T = UQ + (IU, DU) OR ID + (IU, DU)
D = V = UQ
E = I = I
F = U = I++ OR Q--
G = Q = Q

我得到一个结论，肽展公式的推导(C=I+D)比数字逻辑的推导(C=U+D+D)更准确。

###开始线性整理
元基数字 = 元基符号= 肽展公式元基数字变换
0 = D = DD
E = I = I
F = U = I++ OR Q--
G = Q = Q

1 = C = ID
3 = E = IU, DU
4 = H = (IU, DU) OR ID
D = V = UQ
9 = S = QI

2 = P = (IU, DU) + ID
A = O = (IU, DU) + QI
5 = HC = ((IU, DU) OR ID) + ID
B = HE = ((IU, DU) OR ID) + (IU, DU)
8 = M = ((IU, DU) OR ID) + ID + QI

7 = A = UQQI
6 = X = UQ + ((IU, DU) OR ID) + ID
C = T = UQ + ((IU, DU) OR ID) + (IU, DU)

###线性整理优化
元基数字 = 元基符号= 肽展公式元基数字变换
0 = D = DD
E = I = I
F = U = I++ OR Q--
G = Q = Q

1 = C = ID
3 = E = IU, DU
4 = H = (IU, DU) OR ID
5 = HC = ((IU, DU) OR ID) + ID
B = HE = ((IU, DU) OR ID) + (IU, DU)
D = V = UQ
9 = S = QI

2 = P = (IU, DU) + ID
A = O = (IU, DU) + QI
7 = A = UQQI

8 = M = ((IU, DU) OR ID) + ID + QI
6 = X = UQ + ((IU, DU) OR ID) + ID
C = T = UQ + ((IU, DU) OR ID) + (IU, DU)

我推导出语义元基的次序为
A O P M - T X H DD - V E C S - I D U Q
现在的元基数字逻辑次序为
M X T - P O A - C E H HC HE V S - D I U Q

###酸碱肽展开归纳如下
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)
0 = D = DD =(D, DD)
E = I = I =(I)
F = U = I++ OR Q-- =(I, Q)
G = Q = Q =(Q)

1 = C = ID =(ID)
3 = E = IU, DU =(IU, DU)
4 = H = (IU, DU) OR ID =(IU, DU, ID)
D = V = UQ =(UQ)
9 = S = QI =(QI)

2 = P = (IU, DU) + ID =(IUID, DUID)
5 = HC = ((IU, DU) OR ID) + ID =(IUID, DUID, IDID)
B = HE = ((IU, DU) OR ID) + (IU, DU) =(IUIU, IUDU, DUIU, DUDU, IDIU, IDDU)
A = O = (IU, DU) + QI =(IUQI, DUQI)
7 = A = UQQI =(UQQI)

8 = M = ((IU, DU) OR ID) + ID + QI =(IUIDQI, DUIDQI, IDIDQI)
6 = X = UQ + ((IU, DU) OR ID) + ID =(UQIUID, UQDUID, UQIDID)
C = T = UQ + ((IU, DU) OR ID) + (IU, DU) =(UQIUIU, UQIUDU, UQDUIU, UQDUDU, UQIDIU, UQIDDU)

归纳后的元基数字逻辑次序为
M X T - P HC HE O A - C E H V S - D I U Q

归纳后的元基数字活性次序为
T X M - HE HC O P A - H E C V S - U D I Q

准备应用于养疗经DNA视觉进行简单验证下，优化后用于DNA数据库的数字层计算。

在这次序表中D在I的前面，于是我准备修正C=ID为DI，于是如下：

修正C后的最新肽展计算公式观测
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)
0 = D = DD =(D, DD)
E = I = I =(I)
F = U = I++ OR Q-- =(I, Q)
G = Q = Q =(Q)

1 = C = DI =(DI)
3 = E = IU, DU =(IU, DU)
4 = H = (IU, DU) OR DI =(IU, DU, DI)
D = V = UQ =(UQ)
9 = S = QI =(QI)

2 = P = (IU, DU) + DI =(IUDI, DUDI)
5 = HC = ((IU, DU) OR DI) + DI =(IUDI, DUDI, DIDI)
B = HE = ((IU, DU) OR DI) + (IU, DU) =(IUIU, IUDU, DUIU, DUDU, DIIU, DIDU)
A = O = (IU, DU) + QI =(IUQI, DUQI)
7 = A = UQQI =(UQQI)

8 = M = ((IU, DU) OR DI) + DI + QI =(IUDIQI, DUDIQI, DIDIQI)
6 = X = UQ + ((IU, DU) OR DI) + DI =(UQIUDI, UQDUDI, UQDIDI)
C = T = UQ + ((IU, DU) OR DI) + (IU, DU) =(UQIUIU, UQIUDU, UQDUIU, UQDUDU, UQDIIU, UQDIDU)

继续跟进了下在离散数学中 H = (IU, DU) OR DI= (IU, DU) + DI= IUDI, DUDI, 上面的肽展公式在 离散数学中可以继续展开如下

```
#### 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)
#### 0  =               D  =    DD =(D, DD)
#### E  =               I  =    I  =(I)
#### F  =               U  =    I++ OR Q--  =(I, Q)
#### G  =               Q  =    Q  =(Q)

#### 1  =               C  =    DI  =(DI)
#### 3  =               E  =    IU, DU  =(IU, DU)
#### 4  =               H  =    (IU, DU) OR DI  =(IU, DU, DI) OR (IUDI, DUDI)
#### D  =               V  =    UQ  =(UQ)
#### 9  =               S  =    QI  =(QI)


#### 2  =               P  =    (IU, DU) + DI  =(IUDI, DUDI)
#### 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)
#### A  =               O  =    (IU, DU) + QI  =(IUQI, DUQI)
#### 7  =               A  =    UQQI  =(UQQI)


#### 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)
```

似乎开始完美。于是活性顺序又打乱了，再整理下如下：

```
#### 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)
```

C还是 = DI 次序， 所以上一步的公式不用变动。
整理后：
元基活性次序为 <IUQ D CEVS H POA -+ MXT>

我得到一个结论:IDQ 是稳定元基， UH是活性元基.(2021024 结论更新， IQ是稳定元基， DUH是活性元基)

归纳出核心
黄嘌呤:用于肽展换元 计算
变嘧啶:用于 血氧峰 计算
胞嘧啶:用于 补码 计算

肽展公式参考:
AOPM VECS IDUQ 肽展公式推导与元基编码进化计算以及它的应用发现 1.2.2 国家软著申请 流水号 <2020Z11L0356797> 国作登字 2021-A-00942587 (中华人民共和国 国家版权局)
AOPM-VECS-IDUQ Catalytic INITONS PDE LAW and Its Application
https://gitee.com/DetaChina/collection-of-papers-by-deta/blob/master/%E8%91%97%E4%BD%9C%E6%9D%83%E7%89%88%E6%9C%ACPDE_Formular_1_2_2.pdf
#####
https://github.com/yaoguangluo/Deta_Resource/blob/master/%E8%91%97%E4%BD%9C%E6%9D%83%E7%89%88%E6%9C%ACPDE_Formular_1_2_2.pdf

元基命名参考:
<见类人DNA与 神经元基于催化算子映射编码方式 V_1.2.2版本国家软著申请 流水号 <2020Z11L0333706> 国作登字-2021-A-00097017
https://gitee.com/DetaChina/collection-of-papers-by-deta/blob/master/DNA%20%E7%BC%96%E7%A0%81%E6%96%B9%E5%BC%8F1.2.2%20%E4%B8%AD%E8%AF%91%E8%8B%B1%E6%B7%B7%E5%90%88%E7%89%88.pdf
#####
https://github.com/yaoguangluo/Deta_Resource/blob/master/DNA%20%E7%BC%96%E7%A0%81%E6%96%B9%E5%BC%8F1.2.2%20%E4%B8%AD%E8%AF%91%E8%8B%B1%E6%B7%B7%E5%90%88%E7%89%88.pdf

DNA 催化 与 肽展计算 和 AOPM-TXH-VECS-IDUQ 元基解码 V013_026中文版本 国家著作申请 流水号 <2020Z11L0386462> 国作登字 2021-A-00942586 (中华人民共和国 国家版权局)
https://github.com/yaoguangluo/Deta_Resource/blob/master/DNA%20Initon%20解码%20013026软著申请中文最终版本.pdf
https://gitee.com/DetaChina/collection-of-papers-by-deta/blob/master/DNA%20Initon%20解码%20013026软著申请中文最终版本.pdf
https://gitee.com/DetaChina/collection-of-papers-by-deta/blob/master/DNA%20Initon%20解码%20013026软著申请中文最终版本%20修正'食'字.pdf
https://github.com/yaoguangluo/Deta_Resource/blob/master/DNA%20Initon%20解码%20013026软著申请中文最终版本%20修正'食'字.pdf

元基欧拉环计算参考:
https://gitee.com/DetaChina/dna_-db/blob/master/Initon_Math/org/math/initon/ouler/FindOulerRing.java

双元罗盘参考:
多人著作(罗瑶光, 罗荣武) DNA元基催化与肽计算第二卷 养疗经应用研究 20210305 国家著作申请 流水号 <2021Z11L1057159>
https://gitee.com/DetaChina/collection-of-papers-by-deta/blob/master/罗瑶光-DNA催化与肽计算第二卷20210305.pdf
https://github.com/yaoguangluo/Deta_Catalytic_DNA/blob/master/罗瑶光-DNA催化与肽计算第二卷20210305.pdf

走到这, 元基数学 公式表已经出来了, 下一步就开始简单应用。

第三节 研究心得

最近研发肽化版本产生几个心得总结下:
1 养疗经和华瑞集的肽化观测最后是一个文件夹体系。这个体系全部由文件夹组成, 文件夹的名字是元基组成。
2 说明 养疗经和华瑞集有对应的肽结构, 这个结构我首次命名为类人的智慧永生肽或者生命肽。
---20210426

上面这个心得可以进行人造染色体和归纳元基染色体, 并提供了真实环境的理论科学依据。。
我之前的元基分类 研究排上用场了。

我得到一个结论, 目前人类的染色体和生命特征有各种缺陷,

关于元基:
2年前,我认为元基是 一个催化计算的 初始单元和基础. 我当时命名为initon - init aton for caculation
1年多前开始DNA的元基编码, 我当时庆幸 元基能解释我的索引单元, 我认为是语义的编码索引最小单元.
后来,我开始DNA的元基计算和肽展公式推导应用, 我觉得元基是 对应(基因元 嘌呤嘧啶)基元的 语义变换单元.
现在随着养疗经的深入,我开始系统的命名, 元基-是智慧生命进行逻辑表达的最小单元和基础成分.

关于催化计算,我在微信中2021-05-19-12.07分描述了一句如下:
将某一类未解的问题现象和集合,通过已知的已经健全的学术体系去映射模拟计算, 最终结果量化并探索能得到未知的答案,属于一种无理级计算模式.

关于肽花的计算,我在微信中2021-0518描述如下:
数据肽索引化后的花状植株,可不是药用这么简单,我刚思考了下,这种蒲公英丝状元基组织是软件的孢子.一种全新的数据类软件生命体表达形式,
也是永生的必经之路. 我按这种方式进行肽索引,软件函数分类越来越均匀规则.这是一种前沿性智慧分类扩展技术.
这个数据生命是以人类的DNA和思维方式为参照设计的. 如果用其他方式构造元基,就一定能模拟其他物种,需要严谨的论证.
先别想那么远,把描述文件弄丰富点.

2021年4月26日我在微信说过 华瑞集和养疗经 肽化后的花 就是一种永生药, 现在发现,永生根本就不需要到那个巨的级别, 小小的某一类永生花孢子 就能实现.

我要做的不是永生探讨,这是医学任务,我要做的是实现孢子软件的自我繁殖, 我定义为 生命ETL插件节点. 我的目标是软件的SDLC的自我维护.

今天不写代码,写点思想,描述我是怎么走到这一步的.为了确定我的研发计划是否出现跳跃和分歧.完善下 UML.

2021年5月25日下午16.36我在微信笔记说过,均匀化元基索引最后的结果就是均匀化染色体分类,呈X状,并且 呈蒲公英球状丝化肽元基索引文件夹结构.
Java文件可以逐渐的元基文件夹化.形成文件夹花植株.
具体可以参考 如下文件
M_UnicornNeroThemeETL\OSI\OPE\SI\MCI\OEI\OVU\PQE\extOSGI\OSI_OSU_ASQ_OCQ_OSI_PCI_PCU_MCI_MCU_MSI_OSI_PCI.tvm

20210530 我得到一个信息:随着工程的日益发展演化,P元基的类会越来越多, 如果不进行提取和索引自我AOM元基转化,便会产生智慧级(AOPM)元基分类不均匀,这种倾斜会产生各种问题. --罗瑶光
这是我一个动力,因为O元基和A元基的函数比例越来越小,在均匀染色体索引的领域,我迫切需要做点什么.

20210610 在github gitee 留言中描述下: 我做了一个简单的应用操作,将 包中的相同 元基组 进行过滤并 变成文件夹名中. 我定义为 元基的根枝扩展. (Initons Root Extension IRE)
IRE 是 未来数据分类的 一个产业趋势. 作用 分类精简, 数据代谢, 数据生长,
我猜测生物的生长过程也是这样的.共用一种模式.

今天开始我把 代码中的analyzer 进行整体替换 为A, 我定义下规则 为 1 函数名 A, 2 定义域名 _A, 3 固定名的后缀 _A. 关于2和3的区别通过观察测前缀是否为开头即可.
例子如下:
A 函数
_A 变量
A_A 函数

202106132341今天重定义下 元基词汇编码规则. 比如已经有的词汇 suggest/...OPE.SIU..., 我定义为 .OPESIU. 和 _OPESIU_, 变量为 _开头接OPESIU
这样以后词汇就好区分了, 同时, 词汇可以进行更迅捷的肽展变换.

随着包的越来越小, 现在 函数集合大小仅仅几十兆, 说明元基索引价值越来越明确. 减少内存占比, 提高计算效率, 缩短寻址时间, 算能提高越来越可观.

索引后进行元基催化计算是一个数字生命诞生的标志. 我迫切需要做些什么, 先把 均匀索引 实现.

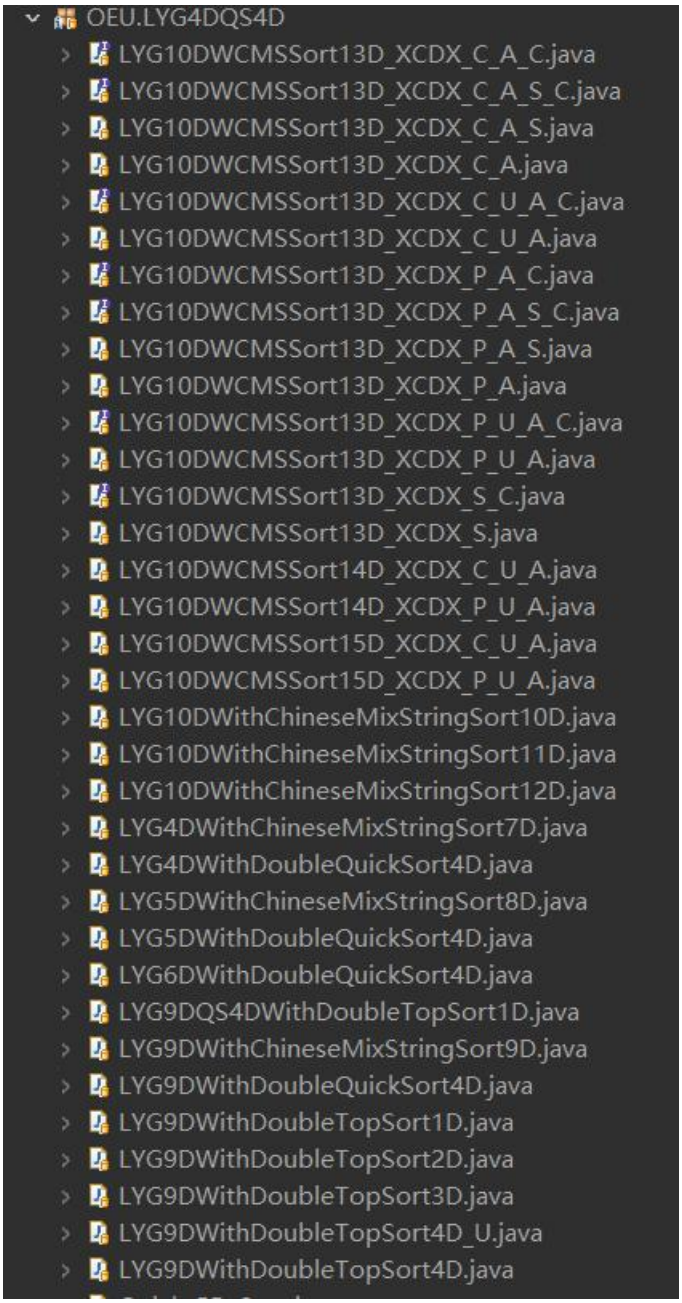
20210628 这几天在不断的索引均匀和 染色体模拟, 结合我在英特尔当后端测试的一些工作经验, 我发现了一些规律.
1 sonar lint的 if,while,等关键字的 嵌套问题化简, 与我现在的新陈代谢的大文件裁剪 都属于 数据进化的几种模式.
2 元基染色体组均匀索引后, 相同的相似的执行逻辑在同一区间,非常方便 以后的 元基芯片 设计 和 软件工程调度. 提高计算密度. 节约算能
3 我找到了一些元基索引规律如 三元基 对称索引, 非对称索引, 移位索引, 反向索引, 主元基索引. 5种模式.
4 关于TXH 的 计算元基 索引, 我会分出一个RNA 包工程来分开处理. 目前这24组双链 为DNA 包.

20210629 今天在裁剪monitor的文件时候,我把2000多行的源码拆成几个300行的文件, 运行也不错.我在思考. 函数像sonar 那样把嵌套if while 拆分,是一种危险的
执行方法. 正确的思路应该是
1 将大文件裁剪成小文件.
2 小文件将多个函数拆成多个文件.
3 一个函数一个文件还复杂,应该进行变量全局化重复提取到1,2 环节.
4 如果出现嵌套 if和 while, 不是复杂的计算逻辑, sonar 强行语法分析报错 价值不大. 当年我写分词就没按sonar来. 思考了下, 当时按照sonar模式搞,之后我的分词肽化会出问题.
5 想起当年 为了搞个 sonar 测试覆盖率非要搞个100%, 这是一个浪费大量时间的弊端.
6 最后 sonar的规范中有价值的地方也有, 如 每行的宽度 限制 我不多广告了, sonar是普通开源软件, 任何人都能下得到.

20210630 今天在裁剪monitor 模拟新陈代谢,我把思想规则 浓缩了下.变成方法如下.
1 首先将裁剪的文件名 加 函数名用_XCDX_ 隔开, 如 APP ... _XCDX_ ... 函数名.java
2 隔开后生成的文件中如果有变量变成未知,则进行全局化, 引用过来.
3 全局化的变量再进行 duplication, 之后变成单例类.
4 单例类之后变成接口 进行 implement.
5 然后整体进行纠正, 将XCDX函数类变成一个 进化插件植株. 这就是二次新陈代谢模拟.

20210701 今天在处理数据库的裁剪按20210630的方案,我有些笔记如下.
1 一个文件按一类函数的裁剪价值巨大. 方便以后操作系统级别的 调度肽化.
2 逐步的脱离养疗经华瑞集医学属性, 24组元基组 满足API的数据工程适用化,像java一样, 接口引用,参与计算.
3 我准备设计一种local static 技术, 满足全局静态. 因为DNA 不同于 RNA, 我不希望它new来new去. 不但损耗算能, 还损耗内存.
4 这个local static 技术, 我之后会在S_AOPM 里面用...SME... 元基组下设计研发.

一些新的探索:
1 排序 新陈代谢



```
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 LYG9DWithDoubleTopSort4D_U{
	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;
				}
			}
		}
	}
```

```
	private int partition(double[] array, int leftPoint, int rightPoint) {
		double x= array[leftPoint]< array[rightPoint]? array[leftPoint]: array[rightPoint];
		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[299999];
		for(int i= 0; i< doubles.length; i++) {
			if(i%3 == 1) {
				doubles[i]= 9999;
			}else {
				doubles[i]= Math.random();
			}
		}
		LYG9DWithDoubleTopSort4D_U lYG9DWithDoubleTopSort2D= new
LYG9DWithDoubleTopSort4D_U();
		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");
	}
}
```

2 笔画排序

```
package OEU.LYG4DQS4D;
//20200314 集成了最新的小高峰过滤催化排序5代思想。
//20200818 集成了最新的小高峰过滤催化排序9代思想。
//增加同拼音同笔画的字按char的int大小区分20210529
//罗瑶光

//今天将新陈代谢技术应用到 中文拼音笔画分词上.
//罗瑶光
```

```
public class LYG10DWCMSSort15D_XCDX_C_U_A extends LYG10DWCMSSort13D_XCDX_C_A implements
LYG10DWCMSSort13D_XCDX_C_U_A_C {

	public void processKernel(String[] kernel, int leftPosition
			, int rightPosition, int scale, int point) {
		int rightPositionReflection= rightPosition;
		if(point> scale) {
			return;
		}
		processQS4DLYG9D(kernel, leftPosition, rightPosition, scale, point, 0);
		int i;
		for(i= leftPosition; i<= rightPosition; i++) {
			if(!(kernel[i].length()<= point|| kernel[leftPosition].length()<= point)) {
				if(kernel[i].charAt(point)!= kernel[leftPosition].charAt(point)){
					rightPositionReflection= i- 1;
					processKernel(kernel, leftPosition, rightPositionReflection, scale, point+ 1);
					leftPosition= i;
				}
			}
		}
		if(leftPosition!= rightPosition) {
			processKernel(kernel, leftPosition, i- 1, scale, point+ 1);
		}
	}

	public void processSort(String[] kernel, int leftPosition
			, int rightPosition, int scale, int point) {
		if(point> scale) {
			return;
		}
		for(int i= leftPosition; i<= rightPosition; i++) {
			Here:
				for(int j= i; j<= rightPosition; j++) {
					if(i== j) {
						continue Here;
					}
					if(kernel[i].length()<= point|| kernel[j].length()<= point) {
						if(kernel[i].length()< kernel[j].length()) {
							for(int p= 0; p< scale; p++) {
								if(!(kernel[i].length()<= p|| kernel[j].length()<= p)) {
									if(kernel[i].charAt(p)!= kernel[j].charAt(p)) {
										continue Here;
									}
								}
							}
							String temp= kernel[i].toString();;
							kernel[i]= kernel[j].toString();;
							kernel[j]= temp;
						}
						continue Here;
					}else {
						boolean hasXi= pinyin.containsKey(""+ kernel[i].charAt(point));
						boolean hasXj= pinyin.containsKey(""+ kernel[j].charAt(point));
						boolean hasBi= bihua.containsKey(""+ kernel[i].charAt(point));
						boolean hasBj= bihua.containsKey(""+ kernel[j].charAt(point));
						if(!(!hasXi|| !hasXj)){//都有拼音
							String[] js= new String[2];
							js[0]= this.pinyin.get(""+ kernel[i].charAt(point));
							js[1]= this.pinyin.get(""+ kernel[j].charAt(point));
							if(js[0].equalsIgnoreCase(js[1])) {
```

```
                                        if(!(!hasBi|| !hasBj)){//都有笔画
                                                if(this.bihua.get(""+ kernel[i].charAt(point))
                                                                > this.bihua.get(""+
kernel[j].charAt(point))) {

                                                        String temp= kernel[i].toString();
                                                        kernel[i]= kernel[j].toString();
                                                        kernel[j]= temp;
                                                        continue Here;
                                                }else if(this.bihua.get(""+ kernel[i].charAt(point))
                                                                == this.bihua.get(""+
kernel[j].charAt(point))) {

                                                        int asci=kernel[i].charAt(point);
                                                        int ascj=kernel[j].charAt(point);
                                                        if(asci< ascj) {//根据前面select的sort定
义来规范,盲目改成大于会出错.

                                                                String temp=
kernel[i].toString();

                                                                kernel[i]= kernel[j].toString();
                                                                kernel[j]= temp;
                                                                continue Here;
                                                        }
                                                }
                                        }
                                }


                                boolean change= processSortpinyin(js, 3);
                                if(!(!change|| i>= j)) {
                                        String temp= kernel[i].toString();
                                        kernel[i]= kernel[j].toString();
                                        kernel[j]= temp;
                                }
                                continue Here;
                        }else if(!(hasXi|| !hasXj)){//其中一个有拼音
                                if(i< j) {
                                        if(!(i== rightPosition+1 || j== rightPosition+1)) {
                                                String temp= kernel[i].toString();
                                                kernel[i]= kernel[j].toString();
                                                kernel[j]= temp;
                                        }
                                }
                                continue Here;
                        }else if(!(!hasXi|| hasXj)){
                                if(i> j) {
                                        if(!(i== rightPosition+1 || j== rightPosition+1)) {
                                                String temp= kernel[i].toString();
                                                kernel[i]= kernel[j].toString();
                                                kernel[j]= temp;
                                        }
                                }
                                continue Here;
                        }else if(!(hasXi|| hasXj)){//都没有拼音
                                if(kernel[i].toLowerCase().charAt(point)
                                                > kernel[j].toLowerCase().charAt(point)) {
                                        if(i< j) {
                                                String temp= kernel[i].toString();
                                                kernel[i]= kernel[j].toString();
                                                kernel[j]= temp;
                                        }
                                        continue Here;
                                }
```

```
                                                    if(kernel[i].toLowerCase().charAt(point)
                                                            == kernel[j].toLowerCase().charAt(point)) {
                                                        if(kernel[i].charAt(point)> kernel[j].charAt(point)) {
                                                            if(i< j) {
                                                                String temp= kernel[i].toString();
                                                                kernel[i]= kernel[j].toString();
                                                                kernel[j]= temp;
                                                            }
                                                        }
                                                        continue Here;
                                                    }
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }

        public void processQS4DLYG9D(String[] kernel, int leftPosition
                , int rightPosition, int scale, int point, int deep) {
            if(leftPosition< rightPosition){
                int c= rightPosition- leftPosition+ 1;
                if(!(c< this.range|| deep> this.deeps)) {//增加了deep
                    int pos= partition(kernel, leftPosition, rightPosition, scale, point);
                    if(leftPosition< pos- 1) {
                        processQS4DLYG9D(kernel, leftPosition, pos- 1, scale, point, deep+ 1);
                    }
                    if(pos+ 1< rightPosition) {
                        processQS4DLYG9D(kernel, pos+ 1, rightPosition, scale, point, deep+ 1);
                    }
                    return;
                }
                processSort(kernel, leftPosition, rightPosition, scale, point);
                return;
            }
        }

        public int partition(String[] array, int leftPosition, int rightPosition, int scale, int point) {
            String x= findSmall(array, scale, point, leftPosition, rightPosition, rightPosition)
                    ? array[rightPosition]: array[leftPosition];
            int leftPositionReflection= leftPosition;
            while(leftPositionReflection < rightPosition) {
                while(!(findSmallWithTwoChar(array[leftPositionReflection]
                        , x, scale, point)|| leftPositionReflection++ >= rightPosition)) {}
                while(findSmallWithTwoChar(array[rightPosition--], x, scale, point)){}
                if(leftPositionReflection< ++rightPosition){
                    String temp= array[rightPosition].toString();;
                    array[rightPosition]= array[leftPositionReflection].toString();;
                    array[leftPositionReflection]= temp;
                }
            }
            array[leftPosition]= array[rightPosition].toString();
            array[rightPosition]= x;
            return rightPosition;
        }
}
```

```
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 LYG9DWithDoubleTopSort4D{
	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;
				}
			}
		}
	}

	private int partition(double[] array, int leftPoint, int rightPoint) {
		double x= array[leftPoint]< array[rightPoint]? array[leftPoint]: array[rightPoint];
		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();
		}
		LYG9DWithDoubleTopSort4D lYG9DWithDoubleTopSort2D= new LYG9DWithDoubleTopSort4D();
		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 OEU.LYG4DQS4D;
//20200314 集成了最新的小高峰过滤催化排序5代思想。
//20200818 集成了最新的小高峰过滤催化排序9代思想。
//增加同拼音同笔画的字按char的int大小区分20210529
//罗瑶光

//今天将新陈代谢技术应用到 中文拼音笔画分词上.
//罗瑶光
public class LYG10DWCMSSort15D_XCDX_P_U_A extends LYG10DWCMSSort13D_XCDX_P_A implements
LYG10DWCMSSort13D_XCDX_P_U_A_C{

	public void processKernel2D(String[][] kernel, int leftPosition
			, int rightPosition, int scale, int point, int culumn) {
		int rightPositionReflection= rightPosition;
		if(point> scale) {
			return;
		}
		processQS4DLYG9D2D(kernel, leftPosition, rightPosition, scale, point, 0, culumn);
		int i;
		for(i= leftPosition; i<= rightPosition; i++) {
			if(!(kernel[i][column].length()<= point
					|| kernel[leftPosition][culumn].length()<= point)) {
				if(kernel[i][culumn].charAt(point)
						!= kernel[leftPosition][culumn].charAt(point)){
					rightPositionReflection= i- 1;
					processKernel2D(kernel, leftPosition, rightPositionReflection, scale, point+ 1,
culumn);
					leftPosition= i;
				}
			}
		}
		if(leftPosition!= rightPosition) {
			processKernel2D(kernel, leftPosition, i- 1, scale, point+ 1, culumn);
		}
	}
```

```
	public void processQS4DLYG9D2D(String[][] kernel, int leftPosition
			, int rightPosition, int scale, int point, int deep, int culumn) {
		if(leftPosition< rightPosition){
			int c= rightPosition- leftPosition+ 1;
			if(!(c< this.range|| deep> this.deeps)) {//增加了deep
				int pos= partition2D(kernel, leftPosition, rightPosition, scale, point, culumn);
				if(leftPosition< pos- 1) {
					processQS4DLYG9D2D(kernel, leftPosition, pos- 1, scale, point, deep+ 1, culumn);
				}
				if(pos+ 1< rightPosition) {
					processQS4DLYG9D2D(kernel, pos+ 1, rightPosition, scale, point, deep+ 1, culumn);
				}
				return;
			}
			processSort2D(kernel, leftPosition, rightPosition, scale, point, culumn);
			return;
		}
	}

	public int partition2D(String[][] array, int leftPosition, int rightPosition
			, int scale, int point, int culumn) {
		String[] x= findSmall2D(array, scale, point, leftPosition, rightPosition, rightPosition, culumn)
				? array[rightPosition]: array[leftPosition];
		int leftPositionReflection= leftPosition;
		while(leftPositionReflection< rightPosition) {
			while(!(findSmallWithTwoChar(array[leftPositionReflection][culumn]
					, x[culumn], scale, point)|| leftPositionReflection++ >= rightPosition)) {}
			while(findSmallWithTwoChar(array[rightPosition--][culumn]
					, x[culumn], scale, point)){}
			if(leftPositionReflection< ++rightPosition){
				String[] temp= array[rightPosition].clone();;
				array[rightPosition]= array[leftPositionReflection].clone();;
				array[leftPositionReflection]= temp.clone();
			}
		}
		array[leftPosition]= array[rightPosition].clone();
		array[rightPosition]= x.clone();
		return rightPosition;
	}

	public void processSort2D(String[][] kernel, int leftPosition
			, int rightPosition, int scale, int point, int culumn) {
		if(point> scale) {
			return;
		}
		for(int i= leftPosition; i<= rightPosition; i++) {
			Here:
				for(int j= i; j<= rightPosition; j++) {
					if(i== j) {
						continue Here;
					}
					if(kernel[i][culumn].length()<= point
							|| kernel[j][culumn].length()<= point) {
						if(kernel[i][culumn].length()< kernel[j][culumn].length()) {
							for(int p= 0; p< scale; p++) {
								if(!(kernel[i][culumn].length()<= p
										|| kernel[j][culumn].length()<= p)) {
									if(kernel[i][culumn].charAt(p)
											!= kernel[j][culumn].charAt(p))
{
										continue Here;
									}
```

```
                                                }
                                        }
                                        String[] temp= kernel[i].clone();;
                                        kernel[i]= kernel[j].clone();;
                                        kernel[j]= temp.clone();
                                }
                                continue Here;
                        }else {
                                boolean hasXi= pinyin.containsKey(""+ kernel[i][culumn].charAt(point));
                                boolean hasXj= pinyin.containsKey(""+ kernel[j][culumn].charAt(point));
                                if(!(!hasXi|| !hasXj)){
                                        String[] js= new String[2];
                                        js[0]= this.pinyin.get(""+ kernel[i][culumn].charAt(point));
                                        js[1]= this.pinyin.get(""+ kernel[j][culumn].charAt(point));
                                        boolean change= processSortpinyin(js, 3);
                                        if(!(!change|| i>= j)) {
                                                String[] temp= kernel[i].clone();
                                                kernel[i]= kernel[j].clone();
                                                kernel[j]= temp.clone();
                                        }
                                        continue Here;
                                }else if(!(hasXi|| !hasXj)){
                                        if(i< j) {
                                                if(!(i== rightPosition+1 || j== rightPosition+1)) {
                                                        String[] temp= kernel[i].clone();
                                                        kernel[i]= kernel[j].clone();
                                                        kernel[j]= temp.clone();
                                                }
                                        }
                                        continue Here;
                                }else if(!(hasXi|| hasXj)){
                                        if(kernel[i][culumn].toLowerCase().charAt(point)
                                                        > kernel[j][culumn].toLowerCase().charAt(point))
{
                                                if(i< j) {
                                                        String[] temp= kernel[i].clone();
                                                        kernel[i]= kernel[j].clone();
                                                        kernel[j]= temp.clone();
                                                }
                                                continue Here;
                                        }
                                        if(kernel[i][culumn].toLowerCase().charAt(point)
                                                        ==
kernel[j][culumn].toLowerCase().charAt(point)) {
                                                if(kernel[i][culumn].charAt(point)>
kernel[j][culumn].charAt(point)) {
                                                        if(i< j) {
                                                                String[] temp= kernel[i].clone();
                                                                kernel[i]= kernel[j].clone();
                                                                kernel[j]= temp.clone();
                                                        }
                                                }
                                        }
                                }
                        }
                }
        }
}
```

```
package OEU.LYG4DQS4D;
////import java.io.BufferedReader;
////import java.io.InputStream;
////import java.io.InputStreamReader;
////import java.util.HashMap;
////20200314 集成了最新的小高峰过滤催化排序5代思想。
//
////20200818 集成了最新的小高峰过滤催化排序9代思想。
////增加同拼音同笔画的字按char的int大小区分20210529
////罗瑶光
//import java.util.Map;
//import SVQ.stable.StableFile;
//import PEU.S.verbal.VerbalSource;
//今天将新陈代谢技术应用到 中文拼音笔画分词上.
//罗瑶光
public class LYG10DWCMSSort13D_XCDX_P_A extends LYG10DWCMSSort13D_XCDX_S implements
LYG10DWCMSSort13D_XCDX_P_A_C{

	public boolean findSmall2D(String[][] kernel, int scale, int point
			, int i, int j, int rightPosition, int culumn) {
		if(kernel[i][culumn].length()<= point
				|| kernel[j][culumn].length()<= point) {
			if(kernel[i][culumn].length()< kernel[j][culumn].length()) {
				for(int p= 0; p< scale; p++) {
					if(!(kernel[i][culumn].length()<= p|| kernel[j][culumn].length()<= p)) {
						if(kernel[i][culumn].charAt(p)!= kernel[j][culumn].charAt(p)) {
							return false;
						}
					}
				}
				return true;
			}
			return false;
		}else {
			boolean hasXi= pinyin.containsKey(""+ kernel[i][culumn].charAt(point));
			boolean hasXj= pinyin.containsKey(""+ kernel[j][culumn].charAt(point));
			if(!(!hasXi|| !hasXj)){
				String[] js= new String[2];
				js[0]= this.pinyin.get(""+ kernel[i][culumn].charAt(point));
				js[1]= this.pinyin.get(""+ kernel[j][culumn].charAt(point));
				boolean change= processSortpinyin(js, 3);
				if(!(!change|| i>= j)) {
					return true;
				}
				return false;
			}else if(!(hasXi|| hasXj)){
				if(kernel[i][culumn].toLowerCase().charAt(point)
						> kernel[j][culumn].toLowerCase().charAt(point)) {
					if(i< j) {
						return true;
					}
					return false;
				}else if(kernel[i][culumn].toLowerCase().charAt(point)
						== kernel[j][culumn].toLowerCase().charAt(point)) {
					if(kernel[i][culumn].charAt(point)
							> kernel[j][culumn].charAt(point)) {
						if(i< j) {
							return true;
						}
						return false;
					}
```

```
                                        return false;
                                }
                                return false;
                        }else if(!(hasXi|| !hasXj)){
                                if(i< j) {
                                        if(!(i== rightPosition || j== rightPosition)) {
                                                return true;
                                        }
                                        return false;
                                }
                                return false;
                        }
                }
                return false;
        }
}
```

```
//import java.io.InputStreamReader;
import java.util.HashMap;
//20200314 集成了最新的小高峰过滤催化排序5代思想。

//20200818 集成了最新的小高峰过滤催化排序9代思想。
//增加同拼音同笔画的字按char的int大小区分20210529
//罗瑶光
import java.util.Map;

//import SVQ.stable.StableFile;
//import PEU.S.verbal.VerbalSource;
//今天将新陈代谢技术应用到 中文拼音笔画分词上.
//罗瑶光
public class LYG10DWCMSSort13D_XCDX_S implements LYG10DWCMSSort13D_XCDX_S_C{
        Map<String, Boolean> find= new HashMap<>();
        Map<String, String> pinyin;
        Map<String, Integer> filter= new HashMap<>();
        int range;
        int deeps;
        String numbers= "0123456789.Ee";
        protected Map<String, Integer> bihua;

        public boolean findSmallWithTwoChar(String x1, String x2
                        , int scale, int point) {
                if(x1.length()<= point|| x2.length()<= point) {
                        if(x1.length()< x2.length()) {
                                for(int p= 0; p< scale; p++) {
                                        if(!(x1.length()<= p|| x2.length()<= p)) {
                                                if(x1.charAt(p)!= x2.charAt(p)) {
                                                        return false;
                                                }
                                        }
                                }
                                return true;
                        }
                        return false;
                }else {
```

```
                boolean hasX1= pinyin.containsKey(""+ x1.charAt(point));
                boolean hasX2= pinyin.containsKey(""+ x2.charAt(point));
                if(!(!hasX1|| !hasX2)){
                        String[] js= new String[2];
                        js[0]= this.pinyin.get(""+ x1.charAt(point));
                        js[1]= this.pinyin.get(""+ x2.charAt(point));
                        boolean changepinyin= processSortpinyin(js, 3);
                        if(changepinyin) {
                                return true;
                        }
                        return false;
                }else if(!(hasX1|| hasX2)){
                        if(x1.toLowerCase().charAt(point)> x2.toLowerCase().charAt(point)) {
                                return true;
                        }else if(x1.toLowerCase().charAt(point)== x2.toLowerCase().charAt(point)) {
                                if(x1.charAt(point)> x2.charAt(point)) {
                                        return true;
                                }
                                return false;
                        }
                        return false;
                }else if(!(hasX1|| !hasX2)){
                        return true;
                }
        }
        return false;
    }

    public boolean processSortpinyin(String[] kernel, int scale) {
        for(int k= 0; k< scale; k++) {
                if(kernel[0].length()<= k|| kernel[1].length()<= k) {
                        if(kernel[0].length()< kernel[1].length()) {
                                return true;
                        }
                        return false;
                }
                if(kernel[0].toLowerCase().charAt(k)
                                > kernel[1].toLowerCase().charAt(k)) {
                        return true;
                }
                if(kernel[0].toLowerCase().charAt(k)
                                < kernel[1].toLowerCase().charAt(k)) {
                        return false;
                }
        }
        if(kernel[0].length()< kernel[1].length()) {
                return true;
        }
        return false;
    }
}
```

```
package OEU.LYG4DQS4D;
//import java.io.BufferedReader;
//import java.io.InputStream;
package OEU.LYG4DQS4D;
//import java.io.BufferedReader;
//import java.io.InputStream;
//import java.io.InputStreamReader;
//import java.util.HashMap;
//20200314 集成了最新的小高峰过滤催化排序5代思想。

//20200818 集成了最新的小高峰过滤催化排序9代思想。
//增加同拼音同笔画的字按char的int大小区分20210529
//罗瑶光
//import java.util.Map;

//import SVQ.stable.StableFile;
//import PEU.S.verbal.VerbalSource;
//今天将新陈代谢技术应用到 中文拼音笔画分词上.
//罗瑶光
public interface LYG10DWCMSSort13D_XCDX_S_C{
	boolean findSmallWithTwoChar(String x1, String x2
			, int scale, int point);

	boolean processSortpinyin(String[] kernel, int scale);
}
```

```
package OEU.LYG4DQS4D;
//20200314 集成了最新的小高峰过滤催化排序5代思想。
//20200818 集成了最新的小高峰过滤催化排序9代思想。
//增加同拼音同笔画的字按char的int大小区分20210529
//罗瑶光

//今天将新陈代谢技术应用到 中文拼音笔画分词上.
//罗瑶光
public class LYG10DWCMSSort15D_XCDX_C_U_A extends LYG10DWCMSSort13D_XCDX_C_A implements
LYG10DWCMSSort13D_XCDX_C_U_A_C {

	public void processKernel(String[] kernel, int leftPosition
			, int rightPosition, int scale, int point) {
		int rightPositionReflection= rightPosition;
		if(point> scale) {
			return;
		}
		processQS4DLYG9D(kernel, leftPosition, rightPosition, scale, point, 0);
		int i;
		for(i= leftPosition; i<= rightPosition; i++) {
			if(!(kernel[i].length()<= point|| kernel[leftPosition].length()<= point)) {
				if(kernel[i].charAt(point)!= kernel[leftPosition].charAt(point)){
					rightPositionReflection= i- 1;
					processKernel(kernel, leftPosition, rightPositionReflection, scale, point+ 1);
					leftPosition= i;
				}
			}
		}
		if(leftPosition!= rightPosition) {
			processKernel(kernel, leftPosition, i- 1, scale, point+ 1);
		}
	}
```

```
	public void processSort(String[] kernel, int leftPosition
			, int rightPosition, int scale, int point) {
		if(point> scale) {
			return;
		}
		for(int i= leftPosition; i<= rightPosition; i++) {
			Here:
				for(int j= i; j<= rightPosition; j++) {
					if(i== j) {
						continue Here;
					}
					if(kernel[i].length()<= point|| kernel[j].length()<= point) {
						if(kernel[i].length()< kernel[j].length()) {
							for(int p= 0; p< scale; p++) {
								if(!(kernel[i].length()<= p|| kernel[j].length()<= p)) {
									if(kernel[i].charAt(p)!= kernel[j].charAt(p)) {
										continue Here;
									}
								}
							}
							String temp= kernel[i].toString();;
							kernel[i]= kernel[j].toString();;
							kernel[j]= temp;
						}
						continue Here;
					}else {
						boolean hasXi= pinyin.containsKey(""+ kernel[i].charAt(point));
						boolean hasXj= pinyin.containsKey(""+ kernel[j].charAt(point));
						boolean hasBi= bihua.containsKey(""+ kernel[i].charAt(point));
						boolean hasBj= bihua.containsKey(""+ kernel[j].charAt(point));
						if(!(!hasXi|| !hasXj)){//都有拼音
							String[] js= new String[2];
							js[0]= this.pinyin.get(""+ kernel[i].charAt(point));
							js[1]= this.pinyin.get(""+ kernel[j].charAt(point));
							if(js[0].equalsIgnoreCase(js[1])) {
								if(!(!hasBi|| !hasBj)){//都有笔画
									if(this.bihua.get(""+ kernel[i].charAt(point))
											> this.bihua.get(""+
kernel[j].charAt(point))) {

										String temp= kernel[i].toString();
										kernel[i]= kernel[j].toString();
										kernel[j]= temp;
										continue Here;
									}else if(this.bihua.get(""+ kernel[i].charAt(point))
											== this.bihua.get(""+
kernel[j].charAt(point))) {

										int asci=kernel[i].charAt(point);
										int ascj=kernel[j].charAt(point);
										if(asci< ascj) {//根据前面select的sort定
义来规范,盲目改成大于会出错.

											String temp=
kernel[i].toString();

											kernel[i]= kernel[j].toString();
											kernel[j]= temp;
											continue Here;
										}
									}
								}
							}
```

```
                                        boolean change= processSortpinyin(js, 3);
                                        if(!(!change|| i>= j)) {
                                                String temp= kernel[i].toString();
                                                kernel[i]= kernel[j].toString();
                                                kernel[j]= temp;
                                        }
                                        continue Here;
                                }else if(!(hasXi|| !hasXj)){//其中一个有拼音
                                        if(i< j) {
                                                if(!(i== rightPosition+1 || j== rightPosition+1)) {
                                                        String temp= kernel[i].toString();
                                                        kernel[i]= kernel[j].toString();
                                                        kernel[j]= temp;
                                                }
                                        }
                                        continue Here;
                                }else if(!(!hasXi|| hasXj)){
                                        if(i> j) {
                                                if(!(i== rightPosition+1 || j== rightPosition+1)) {
                                                        String temp= kernel[i].toString();
                                                        kernel[i]= kernel[j].toString();
                                                        kernel[j]= temp;
                                                }
                                        }
                                        continue Here;
                                }else if(!(hasXi|| hasXj)){//都没有拼音
                                        if(kernel[i].toLowerCase().charAt(point)
                                                        > kernel[j].toLowerCase().charAt(point)) {
                                                if(i< j) {
                                                        String temp= kernel[i].toString();
                                                        kernel[i]= kernel[j].toString();
                                                        kernel[j]= temp;
                                                }
                                                continue Here;
                                        }
                                        if(kernel[i].toLowerCase().charAt(point)
                                                        == kernel[j].toLowerCase().charAt(point)) {
                                                if(kernel[i].charAt(point)> kernel[j].charAt(point)) {
                                                        if(i< j) {
                                                                String temp= kernel[i].toString();
                                                                kernel[i]= kernel[j].toString();
                                                                kernel[j]= temp;
                                                        }
                                                }
                                                continue Here;
                                        }
                                }
                                }
                        }
                }
        }

public void processQS4DLYG9D(String[] kernel, int leftPosition
                , int rightPosition, int scale, int point, int deep) {
        if(leftPosition< rightPosition){
                int c= rightPosition- leftPosition+ 1;
                if(!(c< this.range|| deep> this.deeps)) {//增加了deep
                        int pos= partition(kernel, leftPosition, rightPosition, scale, point);
                        if(leftPosition< pos- 1) {
                                processQS4DLYG9D(kernel, leftPosition, pos- 1, scale, point, deep+ 1);
                        }
                        if(pos+ 1< rightPosition) {
```

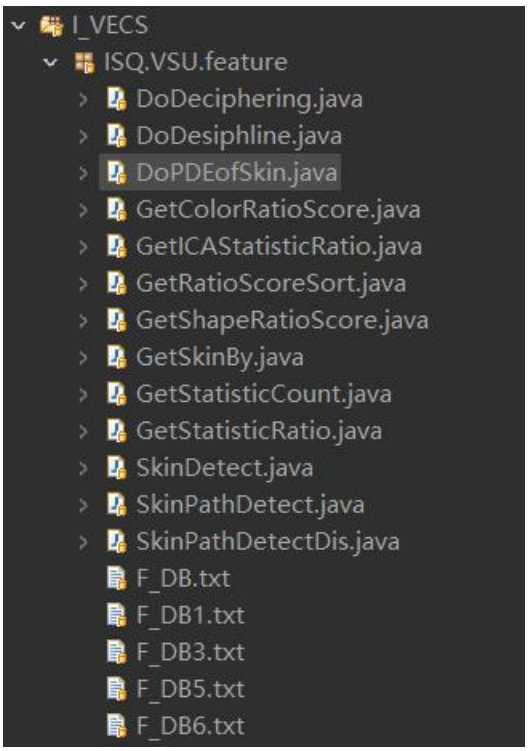
```
                                        processQS4DLYG9D(kernel, pos+ 1, rightPosition, scale, point, deep+ 1);
                                }
                                return;
                        }
                        processSort(kernel, leftPosition, rightPosition, scale, point);
                        return;
                }
        }

        public int partition(String[] array, int leftPosition, int rightPosition, int scale, int point) {
                String x= findSmall(array, scale, point, leftPosition, rightPosition, rightPosition)
                                ? array[rightPosition]: array[leftPosition];
                int leftPositionReflection= leftPosition;
                while(leftPositionReflection< rightPosition) {
                        while(!(findSmallWithTwoChar(array[leftPositionReflection]
                                        , x, scale, point)|| leftPositionReflection++ >= rightPosition)) {}
                        while(findSmallWithTwoChar(array[rightPosition--], x, scale, point)){}
                        if(leftPositionReflection< ++rightPosition){
                                String temp= array[rightPosition].toString();;
                                array[rightPosition]= array[leftPositionReflection].toString();;
                                array[leftPositionReflection]= temp;
                        }
                }
                array[leftPosition]= array[rightPosition].toString();
                array[rightPosition]= x;
                return rightPosition;
        }
}
```

```
package OEU.LYG4DQS4D;
//import java.io.BufferedReader;
//import java.io.InputStream;
//import java.io.InputStreamReader;
//import java.util.HashMap;
//20200314 集成了最新的小高峰过滤催化排序5代思想。

//20200818 集成了最新的小高峰过滤催化排序9代思想。
//增加同拼音同笔画的字按char的int大小区分20210529
//罗瑶光
//import java.util.Map;

//import SVQ.stable.StableFile;
//import PEU.S.verbal.VerbalSource;
//今天将新陈代谢技术应用到 中文拼音笔画分词上.
//罗瑶光
public class LYG10DWCMSSort13D_XCDX_C_A extends LYG10DWCMSSort13D_XCDX_S implements
LYG10DWCMSSort13D_XCDX_C_A_C{
        public boolean findSmall(String[] kernel, int scale, int point
                        , int i, int j, int rightPosition) {
                if(kernel[i].length()<= point|| kernel[j].length()<= point) {
                        if(kernel[i].length()< kernel[j].length()) {
                                for(int p= 0; p< scale; p++) {
                                        if(!(kernel[i].length()<= p|| kernel[j].length()<= p)) {
                                                if(kernel[i].charAt(p)!= kernel[j].charAt(p)) {
                                                        return false;
                                                }
                                        }
                                }
```

```
                        return true;
                    }
                    return false;
                }else {
                    boolean hasXi= pinyin.containsKey(""+ kernel[i].charAt(point));
                    boolean hasXj= pinyin.containsKey(""+ kernel[j].charAt(point));
                    if(!(!hasXi|| !hasXj)){
                        String[] js= new String[2];
                        js[0]= this.pinyin.get(""+ kernel[i].charAt(point));
                        js[1]= this.pinyin.get(""+ kernel[j].charAt(point));
                        boolean change= processSortpinyin(js, 3);
                        if(!(!change|| i>= j)) {
                            return true;
                        }
                        return false;
                    }else if(!(hasXi|| hasXj)){
                        if(kernel[i].toLowerCase().charAt(point)
                                > kernel[j].toLowerCase().charAt(point)) {
                            if(i< j) {
                                return true;
                            }
                            return false;
                        }else if(kernel[i].toLowerCase().charAt(point)
                                == kernel[j].toLowerCase().charAt(point)) {
                            if(kernel[i].charAt(point)> kernel[j].charAt(point)) {
                                if(i< j) {
                                    return true;
                                }
                                return false;
                            }
                            return false;
                        }
                        return false;
                    }else if(!(hasXi|| !hasXj)){
                        if(i< j) {
                            if(!(i== rightPosition || j== rightPosition)) {
                                return true;
                            }
                            return false;
                        }
                        return false;
                    }
                }
                return false;
        }
}
```

接口函数略

3 图片识别




```
package ISQ.VSU.feature;

import java.io.BufferedReader;

import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Map;

import ESU.array.Double_ESU;
import OEU.LYG4DQS4D.LYG9DWithDoubleTopSort4D;

//准备用肽展公式来设计个 皮肤病检测程序.
//罗瑶光 20210710
//软件思想 肽展腐蚀变换. 计算机视觉.
//硬件工具, imageIO, javaCV. 像素头
//因为肽展公式和思想GPL2.0开源, 本程序源码同样GPL2.0开源.
//
```

```
public class SkinPathDetectDis{
	public static List<Double[]> fileCells;
	public static List<String> fileNames;
	public static void main(String[] argv) throws IOException {
		//皮肤病图片识别
		//1 读取一张图片
		//String testImagePath= "C:\\Users\\Lenovo\\Desktop\\deciphering\\F_DB.jpg";
		String testImagePath= "D:\\bu20210606\\搜索图片\\搜索图片\\pgSearch\\皮肤性病学\\阿洪病-寻常狼疮.jpg";
		//2 计算图片训练值
		int 腐蚀浓度= 5;
		int[][][] getSkinBy= new GetSkinBy().getSkinBy(testImagePath);
		int[][][] doPDEofSkin= new DoPDEofSkin().doPDEofSkin(getSkinBy, 腐蚀浓度);
		//new ReadWritePng().writePNG("C:\\Users\\Lenovo\\Desktop\\deciphering\\F_DB1.jpg", doPDEofSkin);
		int 像素阀值= 10;
		int 像素比精度= 5;
		int 像素差精度= 8;
		double[] getStatisticCount= new GetStatisticCount().getStatisticCount(doPDEofSkin, 像素阀值, 像素比精度,
像素差精度);
		//这个getStatisticCount地方可以优化为 像奥运会比赛那样,明显太多德杂色进行自动或者认为剔除, 如字
的黑色, 一些图片的红色.
		//剔除后也可以最小值剔除, 如 一些散落的灰色(高斯噪), 无特征的三原同位色阶 颜色, 图片颜色等.
		double[] getStatisticRatio= new GetStatisticRatio().getStatisticRatio(getStatisticCount);
		//
		//3 遍历疾病数据表.
		initDeciphring("C:\\Users\\Lenovo\\Desktop\\deciphering\\F_DB6.txt");
		//4 打分
		Double[] score= new Double[fileNames.size()];
		String[] nameScore= new String[fileNames.size()];
		int pcaScale= 25;
		int upcaScale= 15;
		int icaScale= 15;
		int ecaScale= 20;
		GetColorRatioScore.scoreDeciphring(score, nameScore, getStatisticRatio, fileCells, fileNames
				, pcaScale, upcaScale, icaScale, ecaScale);
		//
		腐蚀浓度= 95;
		getSkinBy= new GetSkinBy().getSkinBy(testImagePath);
		doPDEofSkin= new DoPDEofSkin().doPDEofSkin(getSkinBy, 腐蚀浓度);
		//new ReadWritePng().writePNG("C:\\Users\\Lenovo\\Desktop\\deciphering\\F_DB1.jpg", doPDEofSkin);
		像素阀值= 10;
		像素比精度= 8;
		像素差精度= 8;
		getStatisticCount= new GetStatisticCount().getStatisticCount(doPDEofSkin, 像素阀值, 像素比精度, 像素差精
度);
		//这个getStatisticCount地方可以优化为 像奥运会比赛那样,明显太多德杂色进行自动或者认为剔除, 如字
的黑色, 一些图片的红色.
		//剔除后也可以最小值剔除, 如 一些散落的灰色(高斯噪), 无特征的三原同位色阶 颜色, 图片颜色等.
		getStatisticRatio= new GetStatisticRatio().getStatisticRatio(getStatisticCount);
		//
		//3 遍历疾病数据表.
		initDeciphring("C:\\Users\\Lenovo\\Desktop\\deciphering\\F_DB5.txt");
		//4 打分
		Double[] score酸= new Double[fileNames.size()];
		GetColorRatioScore.scoreDeciphring(score酸, nameScore, getStatisticRatio, fileCells, fileNames
				, pcaScale, upcaScale, icaScale, ecaScale);
		//融合
		for(int i= 0; i< score酸.length; i++) {
			score[i]+= score酸[i];
```

```
		}

		//5 筛选
		double[] scoreDouble= new double[score.length];
		for(int i= 0; i< scoreDouble.length; i++) {
			scoreDouble[i]= score[i];//
		}
		//改成map
		Map<Double, Map<String, Boolean>> map= Double_ESU.getMapFromDoubleStringArray(score, nameScore);

		int 递归深度= 70; //避免同值冗余内存高峰
		int 堆栈广度= 7; //避免堆栈浪费计算高峰
		new LYG9DWithDoubleTopSort4D().sort(scoreDouble, 堆栈广度, 递归深度);
		//new Quick9DLYGWithString_ESU().sort(scoreInt, nameScore);

		//6 推荐
		Here:
		for(int i= 0; i< scoreDouble.length; i++) {
			if(!map.containsKey(scoreDouble[i])) {
				i++;
				continue Here;
			}
			Iterator<String> iterator= map.get(scoreDouble[i]).keySet().iterator();
			while(iterator.hasNext()) {
				String string= iterator.next();
				if(string.contains("狼")) {
					System.out.println("相似图片:"+ i+ "位"+ string+ "-----分数:"+ scoreDouble[i]);
				}else{
					if(i< 20) {
						System.out.println("相似图片:"+ i+ "位"+ string+ "-----分数:"+
scoreDouble[i]);
					}
				}
			}
			map.remove(scoreDouble[i]);
		//	System.out.println("相似图片:"+ nameScore[i]+ "-----分数:"+ scoreInt[i]);
		}
	}

	private static void initDeciphring(String string) throws NumberFormatException, IOException {
		fileNames= new ArrayList<>();
		fileCells= new ArrayList<>();
		String ctempString;
		InputStream inputStream= new FileInputStream(string);
		BufferedReader cReader= new BufferedReader(new InputStreamReader(inputStream, "GBK"));
		while ((ctempString= cReader.readLine())!= null) {
			String[] lineCell= ctempString.split(">d>");
			if(lineCell.length> 1) {
				fileNames.add(lineCell[0]);
				Double[] lineDouble= new Double[lineCell.length- 1];
				for(int i= 0; i< lineDouble.length; i++) {
					lineDouble[i]= Double.valueOf(lineCell[i+ 1]);
				}
				fileCells.add(lineDouble);
			}
		}
		cReader.close();
	}
}
```

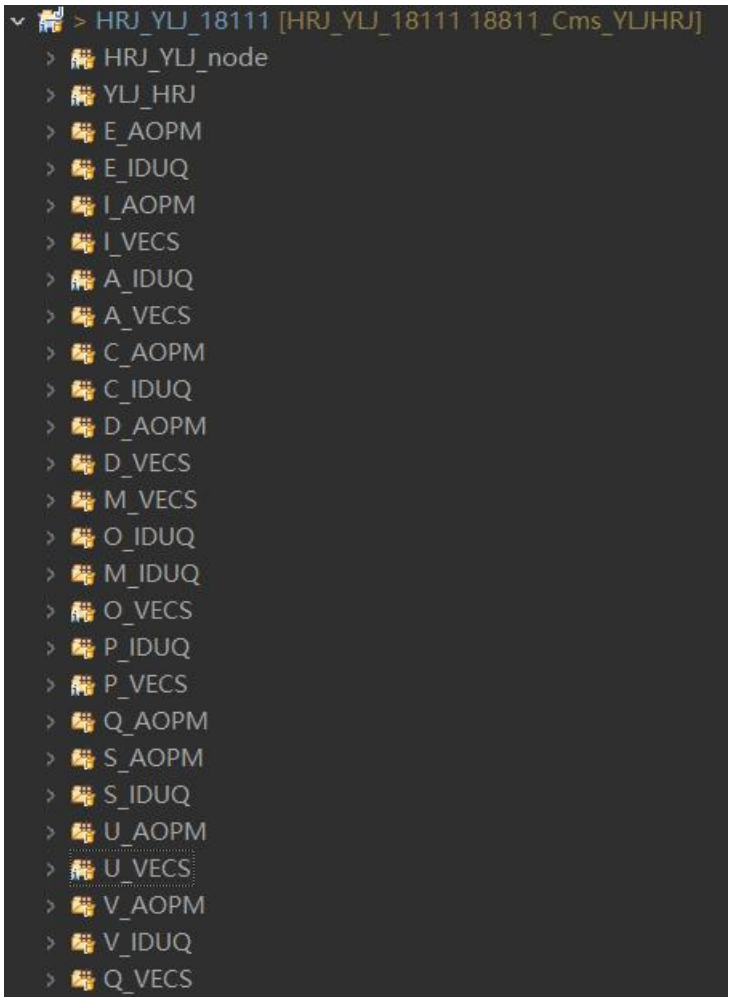
```
public int[][][] doPDEofSkin(int[][][] rgb, int ratio) {
		int[][][] output= new int[3][][];
		//int ratio= 50;
		int[][] rp= new RangePDI().IPE(rgb[0], ratio);
		int[][] gp= new RangePDI().IPE(rgb[1], ratio);
		int[][] bp= new RangePDI().IPE(rgb[2], ratio);
		output[0]= rp;
		output[1]= gp;
		output[2]= bp;
		return output;
	}


package ISQ.VSU.feature;
//准备用肽展公式来设计个 皮肤病检测程序.
//罗瑶光 20210710
//软件思想 肽展腐蚀变换. 计算机视觉.
//硬件工具, imageIO, javaCV.
//因为肽展公式和思想GPL2.0开源,本程序源码同样GPL2.0开源.
public class GetICAStatisticRatio{
	int corr= 15;
	int scale= 7;
	public GetICAStatisticRatio(int scale, int corr) {
		this.scale= scale;
		this.corr= corr;
	}
	public int getPinkStatisticCount(int[][][] rgb) {
		int ri= 255;
		int gi= 0;
		int bi= 255;
		return getCountOfKernel(rgb, ri, gi, bi);
	}

	public int getWhiteStatisticCount(int[][][] rgb) {
		int ri= 255;
		int gi= 255;
		int bi= 255;
		return getCountOfKernel(rgb, ri, gi, bi);
	}

	public int getPupilStatisticCount(int[][][] rgb) {
		int ri= 255;
		int gi= 0;
		int bi= 255;
		return getCountOfKernel(rgb, ri, gi, bi);
	}

	public int getYellowStatisticCount(int[][][] rgb) {
		int ri= 255;
		int gi= 255;
		int bi= 0;
		return getCountOfKernel(rgb, ri, gi, bi);
	}

	public int getRedStatisticCount(int[][][] rgb) {
		int ri= 255;
		int gi= 0;
		int bi= 0;
		return getCountOfKernel(rgb, ri, gi, bi);
	}

	public int getGreenStatisticCount(int[][][] rgb) {
		int ri= 0;
		int gi= 255;
		int bi= 0;
		return getCountOfKernel(rgb, ri, gi, bi);
```

```
	}

	public int getBlueStatisticCount(int[][][] rgb) {
		int ri= 0;
		int gi= 0;
		int bi= 255;
		return getCountOfKernel(rgb, ri, gi, bi);
	}
	public int getMidPinkStatisticCount(int[][][] rgb) {
		int ri= 128;
		int gi= 0;
		int bi= 128;
		return getCountOfKernel(rgb, ri, gi, bi);
	}
	public int getMidWhiteStatisticCount(int[][][] rgb) {
		int ri= 128;
		int gi= 128;
		int bi= 128;
		return getCountOfKernel(rgb, ri, gi, bi);
	}
	public int getMidPupilStatisticCount(int[][][] rgb) {
		int ri= 128;
		int gi= 0;
		int bi= 128;
		return getCountOfKernel(rgb, ri, gi, bi);
	}
	public int getMidYellowStatisticCount(int[][][] rgb) {
		int ri= 128;
		int gi= 128;
		int bi= 0;
		return getCountOfKernel(rgb, ri, gi, bi);
	}
	public int getMidRedStatisticCount(int[][][] rgb) {
		int ri= 128;
		int gi= 0;
		int bi= 0;
		return getCountOfKernel(rgb, ri, gi, bi);
	}
	public int getMidGreenStatisticCount(int[][][] rgb) {
		int ri= 0;
		int gi= 128;
		int bi= 0;
		return getCountOfKernel(rgb, ri, gi, bi);
	}
	public int getMidBlueStatisticCount(int[][][] rgb) {
		int ri= 0;
		int gi= 0;
		int bi= 128;
		return getCountOfKernel(rgb, ri, gi, bi);
	}
	public int getCountOfKernel(int[][][] rgb, int ri, int gi, int bi) {
		int count= 0;
		for(int i= 0; i< rgb[0].length; i++) {
			for(int j= 0; j< rgb[0][0].length; j++) {
				if(rgb[0][i][j]* scale> ri- corr&& rgb[0][i][j]* scale< ri+ corr) {
					if(rgb[1][i][j]* scale> gi- corr&& rgb[1][i][j]* scale< gi+ corr) {
						if(rgb[2][i][j]* scale> bi- corr&& rgb[2][i][j]* scale< bi+
corr) {
							count++;
						}
					}
				}
			}
		}
		return count;
	}
}
```

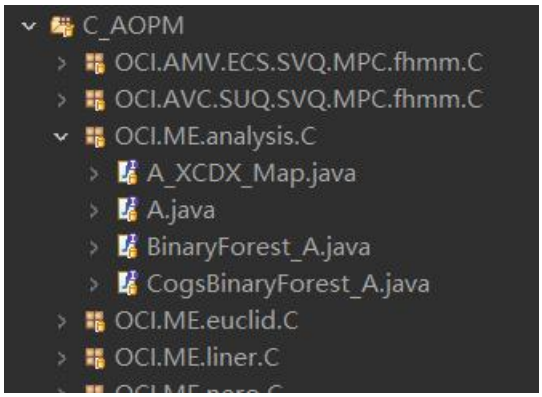
```
package ISQ.VSU.feature;

import java.util.Iterator;
import java.util.List;

//准备用肽展公式来设计个 皮肤病检测程序.
//罗瑶光 20210710
//软件思想 肽展腐蚀变换. 计算机视觉.
//硬件工具, imageIO, javaCV.
//因为肽展公式和思想GPL2.0开源,本程序源码同样GPL2.0开源.
public class GetColorRatioScore{
	//RGBY
	public void getRatioScore() {

	}
	public void getRGRatioScore() {

	}
	public void getRBRatioScore() {

	}
	public void getGBRatioScore() {

	}
	public void getGYRatioScore() {

	}
	public void getBYRatioScore() {

	}
	public void getRYRatioScore() {

	}
	static void scoreDeiphring(Double[] score, String[] nameScore, double[] getStatisticRatio
			, List<Double[]> fileCells, List<String> fileNames) {
		int lineCount= 0;
		Iterator<Double[]> iteratorCells= fileCells.iterator();
		Iterator<String> iteratorNames= fileNames.iterator();
		while(iteratorNames.hasNext()) {
			String name= iteratorNames.next();
			nameScore[lineCount++]= name.toString();
		}
		lineCount= 0;
		while(iteratorCells.hasNext()) {
			Double[] Doubles= iteratorCells.next();
			score[lineCount]= new Double(0);
			for(int i= 0; i< getStatisticRatio.length; i++) {
				try {
					if(0== Doubles[i]|| 0== getStatisticRatio[i]) {
						score[lineCount]+= 10000;
					}else {
						Double ratio= (getStatisticRatio[i]* 10000)/ Doubles[i];
						if(ratio>2||ratio<0.5) {
							score[lineCount]+= 10000;
						}else {
							score[lineCount]+= ratio;
						}
					}
				}catch(Exception e) {
					System.out.println(Doubles[i]);
					score[lineCount]+= 0;
				}
```

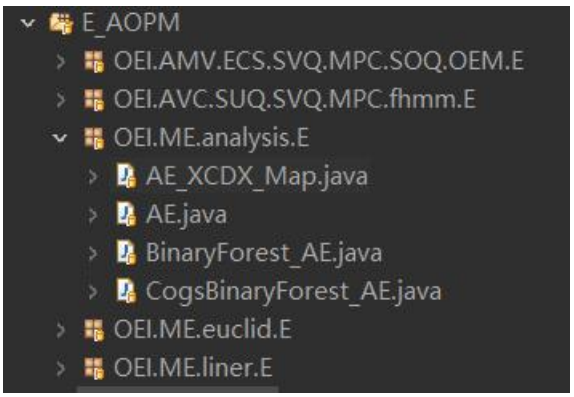
```
                }

                        score[lineCount]= Math.abs(score[lineCount]/ 45 - 1);//nor 均值进行了优化 20210714
                        lineCount++;
                }
        }
        //Deiphring 分解为 Deciphring + Desiphine
        static void scoreDeciphring(Double[] score, String[] nameScore, double[] getStatisticRatio
                        , List<Double[]> fileCells, List<String> fileNames, int pcaScale, int upcaScale, int icaScale, int
ecaScale) {
                int lineCount= 0;
                Iterator<Double[]> iteratorCells= fileCells.iterator();
                Iterator<String> iteratorNames= fileNames.iterator();
                while(iteratorNames.hasNext()) {
                        String name= iteratorNames.next();
                        nameScore[lineCount++]= name.toString();
                }
                lineCount= 0;
                while(iteratorCells.hasNext()) {
                        Double[] Doubles= iteratorCells.next();
                        score[lineCount]= new Double(0);
                        int PCA= 0;
                        int UPCA= 0;
                        int ICA= 0;
                        int ECA= 0;
                        for(int i= 0; i< getStatisticRatio.length; i++) {
                                try {
                                        if(0== Doubles[i]|| 0== getStatisticRatio[i]) {
                                                score[lineCount]+= 10000;
                                                ICA++;
                                                UPCA++;
                                        }else {
                                                Double ratio= (getStatisticRatio[i]* 10000)/ Doubles[i];
                                                if(ratio> 2|| ratio< 0.5) {
                                                        UPCA++;
                                                        score[lineCount]+= 10000;
                                                }else if(ratio< 1.5|| ratio> 0.75) {
                                                        score[lineCount]+= ratio;
                                                        PCA++;
                                                        if(i> 2) {
                                                                ICA++;
                                                        }
                                                        if(ratio< 1.1|| ratio> 0.9) {
                                                                ECA++;
                                                        }
                                                }
                                        }
                                }catch(Exception e) {
                                        System.out.println(Doubles[i]);
                                        score[lineCount]+= 0;
                                }
                        }
                        if(PCA> pcaScale&& UPCA< upcaScale&& ICA> icaScale&& ECA> ecaScale) {
                                score[lineCount]= Math.abs(score[lineCount]/ 45 - 1);//nor 均值进行了优化 20210714
                        }else {
                                score[lineCount]= 99999.0;//nor 均值进行了优化 20210714
                        }
                        lineCount++;
                }
        }
```

4元基枝与元基花

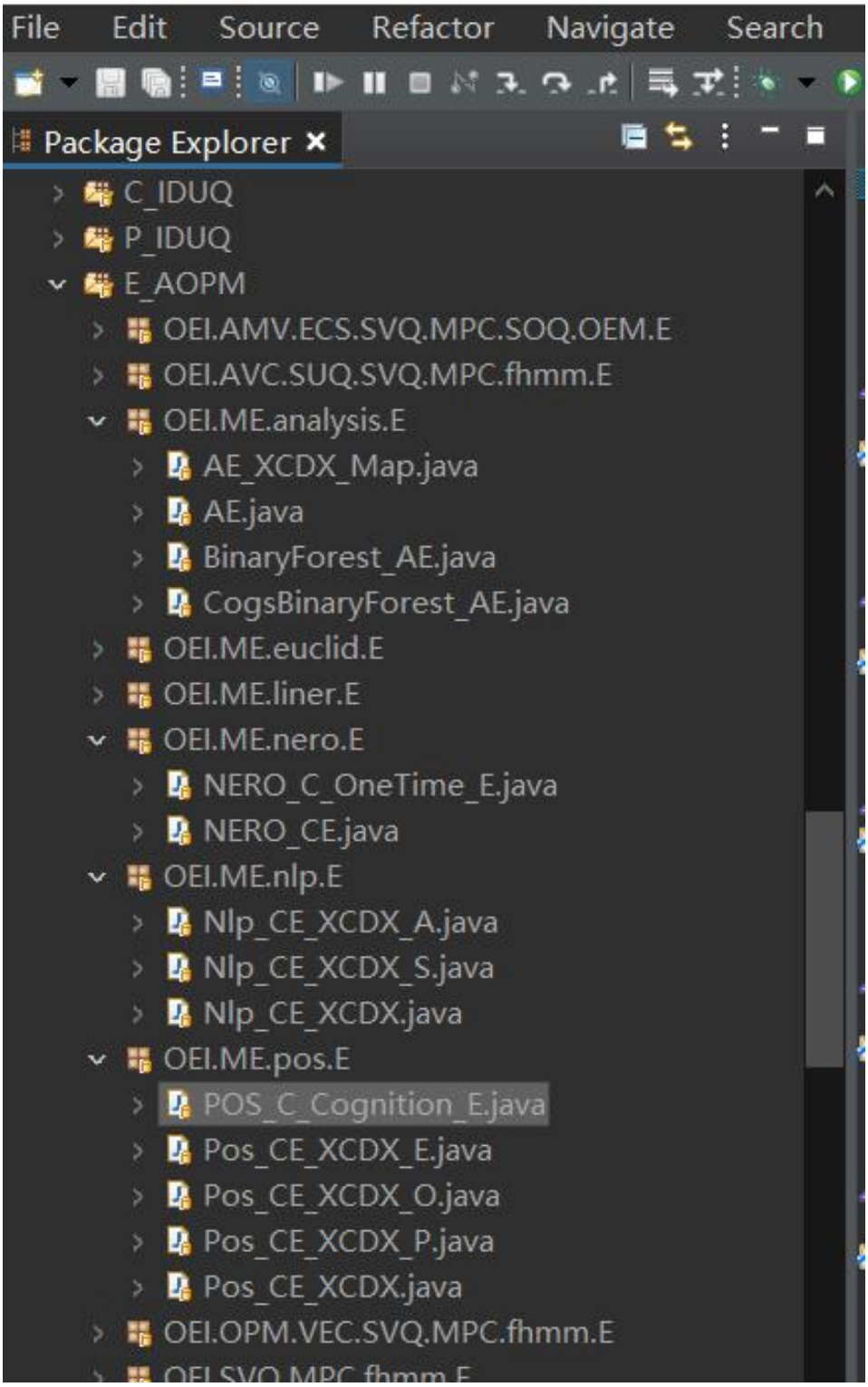


5 2 极速分词





极速新陈代谢分词：核心源码：



```
package OEI.ME.analysis.E;
import java.io.IOException;
import java.util.List;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.FMHMMNode;
import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;
import SVQ.stable.StablePOS;
import ME.utils.WordFrequencyUtil;
import OCI.AVC.SUQ.SVQ.MPC.fhmm.C.EmotionMap;
import OCI.ME.analysis.C.A_XCDX_Map;

import java.util.Iterator;
//PARSER XCDX
//Yaoguang.Luo
//20210702
//一种元基枝 写法结构 处理子类接口 分类.
public class AE_XCDX_Map extends AE implements A_XCDX_Map{
	public Map<String, WordFrequency> parserStringByReturnFrequencyMap(String inputString) {
		Map<String, String> wordsForest= fHMMList.getPosCnToCn();
		Map<String, WordFrequency> outputList = new ConcurrentHashMap<>();
		Map<Long, FMHMMNode> forestRoots= fHMMList.getMap();
		int inputStringLength= inputString.length();
		int forestDepth= StablePOS.INT_ZERO;
		int countInputStringLength;
		StringBuilder[] fixWords = new StringBuilder[StablePOS.INT_TWO];
		fixWords[StablePOS.INT_ZERO] = new StringBuilder();
		fixWords[StablePOS.INT_ONE] = new StringBuilder();
		StringBuilder stringBuilder = new StringBuilder();
		int find = StablePOS.INT_ZERO;
		Here:
			for (int charPosition= StablePOS.INT_ZERO; charPosition< inputStringLength; charPosition
					+= (countInputStringLength== StablePOS.INT_ZERO? StablePOS.INT_ONE:
countInputStringLength)) {
				if(inputString.charAt(charPosition)< StablePOS.INT_ONE_TWO_EIGHT){
					if(fixWords[StablePOS.INT_ZERO].length()> StablePOS.INT_ZERO) {

	if(fixWords[StablePOS.INT_ZERO].charAt(fixWords[StablePOS.INT_ZERO].length()- StablePOS.INT_ONE)
							< StablePOS.INT_ONE_TWO_EIGHT) {

	fixWords[StablePOS.INT_ZERO].append(inputString.charAt(charPosition));
							countInputStringLength= StablePOS.INT_ONE;
							find = StablePOS.INT_ONE;
							continue Here;
						}
						fixWords[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWords[StablePOS.INT_ZERO].length());
					}
					find=StablePOS.INT_ONE;
					fixWords[StablePOS.INT_ZERO].append(inputString.charAt(charPosition));
					countInputStringLength = StablePOS.INT_ONE;
					continue Here;
				}
				if(find == StablePOS.INT_ONE) {
					find = StablePOS.INT_ZERO;
					WordFrequencyUtil.WordFrequencyFindCheck(outputList, fixWords);
				}
				stringBuilder.delete(StablePOS.INT_ZERO, stringBuilder.length());
				stringBuilder =
neroController.getBinaryForestRecurWordOneTime(stringBuilder.append(inputString
						.charAt(charPosition)), inputString, charPosition, inputStringLength,
```

```
forestRoots, forestDepth
                                                , charPosition + StablePOS.INT_ONE);
                                String countWordNode = stringBuilder.toString();
                                int compare = countInputStringLength = countWordNode.length();
                                if (compare == StablePOS.INT_ONE) {
                                        WordFrequencyUtil.WordFrequencyCompareCheck(outputList, fixWords,
countWordNode);
                                        fixWords[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWords[StablePOS.INT_ZERO].length());
                                        fixWords[StablePOS.INT_ZERO].append(countWordNode);
                                        continue Here;
                                }
                                if (compare == StablePOS.INT_TWO) {
                                        countInputStringLength =
nlpController.doSlangPartAndPOSCheckForTwoCharForMap(countInputStringLength
                                                        , outputList, stringBuilder, wordsForest, fixWords, posController);
                                        continue Here;
                                }
                                if (compare == StablePOS.INT_THREE) {
                                        I_FixWords(charPosition, inputString, fixWords);
                                        countInputStringLength =
nlpController.doPOSAndEMMCheckOfThreeForMap(countInputStringLength, outputList
                                                        , wordsForest, stringBuilder, fixWords, posController);
                                        continue Here;
                                }
                                if (compare == StablePOS.INT_FOUR) {
                                        I_FixWords(charPosition, inputString, fixWords);
                                        countInputStringLength =
nlpController.doSlangCheckForMap(countInputStringLength, outputList, stringBuilder
                                                        , wordsForest, fixWords, posController);
                                }
                        }
                return outputList;
        }

        public Map<String, WordFrequency> getWordFrequencyMap(List<String> sets) throws IOException {
                Map<String, WordFrequency> map = new ConcurrentHashMap<>();
                Iterator <String> iterator = sets.iterator();
                Here:
                        while(iterator.hasNext()){
                                String setOfi = iterator.next();
                                if (map.containsKey(setOfi)) {
                                        WordFrequency wordFrequency = map.get(setOfi);
                                        wordFrequency.I_Frequency(wordFrequency.getFrequency() +
StablePOS.INT_ONE);
                                        map.put(setOfi, wordFrequency);
                                        continue Here;
                                }
                                WordFrequency wordFrequency = new WordFrequency();
                                wordFrequency.I_Frequency(StablePOS.INT_ONE);
                                wordFrequency.I_Word(setOfi);
                                map.put(setOfi, wordFrequency);
                        }
                return map;
        }
        //计算函数以后移出 DNA元基组 到RNA.
        public List<WordFrequency> sortWordFrequencyMap(Map<String, WordFrequency> map) throws IOException {
                List<WordFrequency> list = quick6DLuoYaoguangSort.frequencyWordMapToList(map);
                quick6DLuoYaoguangSort.quick6DLuoYaoGuangSortWordFrequency(list, StablePOS.INT_ZERO
                        , list.size() - StablePOS.INT_ONE);
                return list;
        }
```

```
	public Map<Integer, WordFrequency> getWordFrequencyByReturnSortMap(List<String> sets) throws IOException {
		return sortWordFrequencyMapToSortMap(getWordFrequencyMap(sets));
	}

	public Map<Integer, WordFrequency> sortWordFrequencyMapToUnsortMap(Map<String, WordFrequency> map){
		Map<Integer, WordFrequency> listMap = quick6DLuoYaoguangSort.frequencyWordMapToMap(map);
		return listMap;
	}

	public Map<Integer, WordFrequency> sortWordFrequencyMapToSortMap(Map<String, WordFrequency> map){
		Map<Integer, WordFrequency> listMap = quick6DLuoYaoguangSort.frequencyWordMapToMap(map);
		quick6DLuoYaoguangSort.quick6DLuoYaoGuangSortWordFrequency(listMap, StablePOS.INT_ZERO
				, listMap.size() - StablePOS.INT_ONE);
		return listMap;
	}

	public Map<String, WordFrequency> parserMixStringByReturnFrequencyMap(String mixedString) {
		mixedString += StablePOS.SPACE_STRING;
		Map<String, String> wordsForest = fHMMList.getPosCnToCn();
		Map<String, WordFrequency> outputList = new ConcurrentHashMap<>();
		Map<Long, FMHMMNode> forestRoots = fHMMList.getMap();//.getRoot();
		int inputStringLength = mixedString.length();
		int forestDepth = StablePOS.INT_ZERO;
		int countInputStringLength;
		StringBuilder[] fixWords = new StringBuilder[StablePOS.INT_TWO];
		fixWords[StablePOS.INT_ZERO] = new StringBuilder();
		fixWords[StablePOS.INT_ONE] = new StringBuilder();
		StringBuilder stringBuilder = new StringBuilder();
		int find = StablePOS.INT_ZERO;
		Here:
			for (int charPosition = StablePOS.INT_ZERO; charPosition < inputStringLength; charPosition
					+= (countInputStringLength == StablePOS.INT_ZERO ? StablePOS.INT_ONE :
countInputStringLength)) {
				//luan ma
				if(mixedString.charAt(charPosition) < StablePOS.INT_ONE_TWO_EIGHT && charPosition
< mixedString.length()
						- StablePOS.INT_ONE){
					if(find == StablePOS.INT_ZERO) {
						fixWords[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWords[StablePOS.INT_ZERO].length());
					}
					fixWords[StablePOS.INT_ZERO].append(mixedString.charAt(charPosition));
					countInputStringLength = StablePOS.INT_ONE;
					find = StablePOS.INT_ONE;
					continue Here;
				}
				if(find == StablePOS.INT_ONE) {
					find = StablePOS.INT_ZERO;
					Iterator<String> it =
fHMMList.englishStringToWordsList(fixWords[StablePOS.INT_ZERO].toString()).iterator();
					while(it.hasNext()) {
						String temp=it.next();
						if (outputList.containsKey(temp)) {
							WordFrequency wordFrequency = outputList.get(temp);
							wordFrequency.I_Frequency(wordFrequency.getFrequency() +
StablePOS.INT_ONE);
							outputList.put(temp, wordFrequency);
						} else {
							WordFrequency wordFrequency = new WordFrequency();
							wordFrequency.I_Frequency(StablePOS.INT_ONE);
							wordFrequency.I_Word(temp);
							outputList.put(temp, wordFrequency);
						}
```

```
                                        }
                                        fixWords[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWords[StablePOS.INT_ZERO].length());
                                }
                                stringBuilder.delete(StablePOS.INT_ZERO, stringBuilder.length());
                                stringBuilder =
neroController.getBinaryForestRecurWordOneTime(stringBuilder.append(mixedString
                                                .charAt(charPosition)), mixedString, charPosition, inputStringLength,
forestRoots, forestDepth
                                                , charPosition + StablePOS.INT_ONE);

                                String countWordNode = stringBuilder.toString();
                                int compare = countInputStringLength = countWordNode.length();
                                if (compare == StablePOS.INT_TWO) {

                                        countInputStringLength =
nlpController.doSlangPartAndPOSCheckForTwoCharForMap(countInputStringLength
                                                        , outputList, stringBuilder, wordsForest, fixWords, posController);
                                        continue Here;
                                }
                                if (compare == StablePOS.INT_THREE) {
                                        I_FixWords(charPosition, mixedString, fixWords);
                                        countInputStringLength =
nlpController.doPOSAndEMMCheckOfThreeForMap(countInputStringLength, outputList
                                                        , wordsForest, stringBuilder, fixWords, posController);
                                        continue Here;
                                }
                                if (compare == StablePOS.INT_ONE) {
                                        if (outputList.containsKey(countWordNode)) {
                                                WordFrequency wordFrequency = outputList.get(countWordNode);
                                                wordFrequency.I_Frequency(wordFrequency.getFrequency() +
StablePOS.INT_ONE);
                                                outputList.put(countWordNode, wordFrequency);
                                        } else {
                                                WordFrequency wordFrequency = new WordFrequency();
                                                wordFrequency.I_Frequency(StablePOS.INT_ONE);
                                                wordFrequency.I_Word(fixWords[StablePOS.INT_ZERO].toString());
                                                outputList.put(countWordNode, wordFrequency);
                                        }
                                        fixWords[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWords[StablePOS.INT_ZERO].length());
                                        fixWords[StablePOS.INT_ZERO].append(countWordNode);
                                        continue Here;
                                }
                                if (compare == StablePOS.INT_FOUR) {
                                        I_FixWords(charPosition, mixedString, fixWords);
                                        countInputStringLength =
nlpController.doSlangCheckForMap(countInputStringLength, outputList, stringBuilder
                                                        , wordsForest, fixWords, posController);
                                }
                        }
                return outputList;
        }

        public List<WordFrequency> getWordFrequency(List<String> sets) throws IOException {
                return sortWordFrequencyMap(getWordFrequencyMap(sets));
        }

        public EmotionMap getEmotionMap() {
                return this.emotionMap;
        }
```

```
}

、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、

package OEI.ME.analysis.E;
import java.io.IOException;
import java.util.List;
import java.util.Map;
//import java.util.concurrent.ConcurrentHashMap;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.FMHMMNode;
//import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;
import SVQ.stable.StablePOS;
//import ME.utils.WordFrequencyUtil;
import OCI.AVC.SUQ.SVQ.MPC.fhmm.C.EmotionMap;
import OCI.ME.analysis.C.A;
import OCI.ME.liner.C.Quick6DLuoYaoguangSort;
import OCI.ME.nero.C.NERO_C_OneTime;
//import OCI.ME.nlp.C.NLP_C;
//import OCI.ME.pos.C.POS_C;
//import OCI.ME.pos.C.Pos_C_XCDX;
import OCI.ME.pos.C.Pos_C_XCDX_P;
import OCI.SVQ.MPC.fhmm.C.FHMMList;
import OEI.AVC.SUQ.SVQ.MPC.fhmm.E.EmotionMap_E;
import OEI.ME.liner.E.Quick6DLuoYaoguangSort3DMap_E;
import OEI.ME.nero.E.NERO_C_OneTime_E;
//import OEI.ME.nlp.E.NLP_CE;
import OEI.ME.nlp.E.Nlp_CE_XCDX_S;
//import OEI.ME.pos.E.POS_CE;
//import OEI.ME.pos.E.Pos_CE_XCDX;
import OEI.ME.pos.E.Pos_CE_XCDX_P;
import OEI.SVQ.MPC.fhmm.E.FMHMMListOneTime_E;

import java.util.Iterator;
import java.util.LinkedList;

import PEQ.AMV.ECS.test.SensingTest;
public class AE implements A {
	protected FHMMList fHMMList;
	protected NERO_C_OneTime neroController;
	protected Nlp_CE_XCDX_S nlpController;
	protected Pos_C_XCDX_P posController;
	protected Quick6DLuoYaoguangSort quick6DLuoYaoguangSort;
	protected Map<Long, FMHMMNode> forestRoots;
	protected Map<Long, Map<String, String>> wordsForests;
	protected Map<Long, FMHMMNode> []forestsRoots;
	protected Map<String, String> wordsForest;
	protected EmotionMap emotionMap;
	protected SensingTest sensingTest;
	public void IV_() throws IOException {
		this.fHMMList= new FMHMMListOneTime_E();
		fHMMList.index();
		fHMMList.indexPosEnToCn();
		fHMMList.indexPosEnToEn();
		fHMMList.indexEnToCn();
		fHMMList.indexCnToEn();
		fHMMList.indexFullEnToCn();
		fHMMList.indexFullCnToEn();
		neroController= new NERO_C_OneTime_E();
		nlpController= new Nlp_CE_XCDX_S();
		posController= new Pos_CE_XCDX_P();
		quick6DLuoYaoguangSort= new Quick6DLuoYaoguangSort3DMap_E();
```

```
                forestRoots= fHMMList.getMap();
                forestsRoots= fHMMList.getMaps();
                wordsForest= fHMMList.getPosCnToCn();
                wordsForests= fHMMList.getWordsForests();
        }

        public void IV_Mixed() throws IOException {
                this.fHMMList= new FMHMMListOneTime_E();
                fHMMList.indexMixed();
                fHMMList.indexPosEnToCn();
                fHMMList.indexPosEnToEn();
                fHMMList.indexEnToCn();
                fHMMList.indexCnToEn();
                fHMMList.indexFullEnToCn();
                fHMMList.indexFullCnToEn();
                fHMMList.indexFullCnToPy();
                fHMMList.indexFullCnToKo();;
                fHMMList.indexFullCnToJp();;
                fHMMList.indexFullCnToTt();
                fHMMList.indexFullCnToRs();
                fHMMList.indexFullCnToAb();
                neroController= new NERO_C_OneTime_E();
                nlpController= new Nlp_CE_XCDX_S();
                posController= new Pos_CE_XCDX_P();
                quick6DLuoYaoguangSort= new Quick6DLuoYaoguangSort3DMap_E();
                forestRoots= fHMMList.getMap();
                forestsRoots= fHMMList.getMaps();
                wordsForest= fHMMList.getPosCnToCn();
                wordsForests= fHMMList.getWordsForests();
                emotionMap= new EmotionMap_E();
                emotionMap.IV_MotivationMap();
                emotionMap.IV_NegativeMap();
                emotionMap.IV_PositiveMap();
                emotionMap.IV_TrendingMap();
                emotionMap.IV_PredictionMap();
                sensingTest= new SensingTest();
        }

        public List<String> parserMixedString(String mixedString) {
                mixedString+= StablePOS.SPACE_STRING_DISTINCTION;
                int inputStringLength= mixedString.length();
                List<String> outputList = new LinkedList<>();
                int forestDepth = StablePOS.INT_ZERO;
                int countInputStringLength;
                StringBuilder[] fixWords = new StringBuilder[StablePOS.INT_TWO];
                fixWords[StablePOS.INT_ZERO] = new StringBuilder();
                fixWords[StablePOS.INT_ONE] = new StringBuilder();
                StringBuilder stringBuilder = new StringBuilder();
                int find = StablePOS.INT_ZERO;
                Here:
                        for (int charPosition = StablePOS.INT_ZERO;charPosition<inputStringLength;charPosition

        +=(countInputStringLength==StablePOS.INT_ZERO?StablePOS.INT_ONE:countInputStringLength)) {
                                if(mixedString.charAt(charPosition) < StablePOS.INT_TEN_SOUTHANDS && charPosition
< inputStringLength
                                                - StablePOS.INT_ONE){
                                        if(find == StablePOS.INT_ZERO) {
                                                fixWords[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWords[StablePOS.INT_ZERO].length());
                                        }
                                        fixWords[StablePOS.INT_ZERO].append(mixedString.charAt(charPosition));
                                        countInputStringLength = StablePOS.INT_ONE;
                                        find = StablePOS.INT_ONE;
```

```
                                        continue Here;
                                }
                                if(StablePOS.INT_ONE == find) {
                                        find = StablePOS.INT_ZERO;
                                        Iterator<String> it =
fHMMList.englishStringToWordsList(fixWords[StablePOS.INT_ZERO].toString()).iterator();
                                        StringBuilder number= new StringBuilder();
                                        while(it.hasNext()) {
                                                String temp = it.next();
                                                if(StablePOS.NUMBERS.contains(temp)) {
                                                        number.append(temp);
                                                }else {
                                                        if(number.length()>0) {
                                                                outputList.add(number.toString());
                                                                number.delete(0, number.length());
                                                        }
                                                        outputList.add(temp);
                                                }
                                        }
                                        if(number.length()>0) {
                                                outputList.add(number.toString());
                                                number.delete(0, number.length());
                                        }
                                        fixWords[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWords[StablePOS.INT_ZERO].length());
                                }
                                stringBuilder.delete(StablePOS.INT_ZERO, stringBuilder.length());
                                stringBuilder =
neroController.getBinaryForestRecurWordOneTime(stringBuilder.append(mixedString
                                                .charAt(charPosition)), mixedString, charPosition, inputStringLength,
forestRoots, forestDepth
                                                , charPosition + StablePOS.INT_ONE);
                                String countWordNode = stringBuilder.toString();
                                int compare = countInputStringLength = countWordNode.length();
                                if (StablePOS.INT_ONE == compare) {
                                        outputList.add(countWordNode);
                                        fixWords[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWords[StablePOS.INT_ZERO].length());
                                        fixWords[StablePOS.INT_ZERO].append(countWordNode);
                                        continue Here;
                                }
                                if (StablePOS.INT_TWO == compare) {
                                        countInputStringLength =
nlpController.doSlangPartAndPOSCheckForTwoChar(countInputStringLength, outputList
                                                        , stringBuilder, wordsForest, fixWords, posController, charPosition,
mixedString);
                                        continue Here;
                                }
                                if (StablePOS.INT_THREE == compare) {
                                        I_FixWords(charPosition, mixedString, fixWords);
                                        countInputStringLength =
nlpController.doPOSAndEMMCheckOfThree(countInputStringLength, outputList
                                                        , wordsForest, stringBuilder, fixWords, posController, charPosition,
mixedString);
                                        continue Here;
                                }
                                if (StablePOS.INT_FOUR == compare) {
                                        I_FixWords(charPosition, mixedString, fixWords);
                                        countInputStringLength = nlpController.doSlangCheck(countInputStringLength,
outputList, stringBuilder
                                                        , wordsForest, fixWords, posController, charPosition, mixedString);
                                }
                        }
```

```
		return outputList;
	}

	public List<String> parserString(String inputString) {
		List<String> outputList= new LinkedList<>();
		int inputStringLength= inputString.length();
		int forestDepth= StablePOS.INT_ZERO;
		int countInputStringLength;
		StringBuilder[] fixWords= new StringBuilder[StablePOS.INT_TWO];
		fixWords[StablePOS.INT_ZERO]= new StringBuilder();
		fixWords[StablePOS.INT_ONE]= new StringBuilder();
		StringBuilder stringBuilder= new StringBuilder();
		int find= StablePOS.INT_ZERO;
		Here:
			for (int charPosition= StablePOS.INT_ZERO; charPosition< inputStringLength; charPosition
					+= (countInputStringLength!= StablePOS.INT_ZERO? countInputStringLength:
StablePOS.INT_ONE)) {
				if(StablePOS.INT_ONE_TWO_EIGHT> inputString.charAt(charPosition)){
					if(fixWords[StablePOS.INT_ZERO].length()> StablePOS.INT_ZERO) {

	if(fixWords[StablePOS.INT_ZERO].charAt(fixWords[StablePOS.INT_ZERO].length()- StablePOS.INT_ONE)
							< StablePOS.INT_ONE_TWO_EIGHT) {

	fixWords[StablePOS.INT_ZERO].append(inputString.charAt(charPosition));
							countInputStringLength= StablePOS.INT_ONE;




							find= StablePOS.INT_ONE;
							continue Here;
						}
						fixWords[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWords[StablePOS.INT_ZERO].length());
					}
					find= StablePOS.INT_ONE;
					fixWords[StablePOS.INT_ZERO].append(inputString.charAt(charPosition));
					countInputStringLength= StablePOS.INT_ONE;
					continue Here;
				}
				if(find== StablePOS.INT_ONE) {
					find= StablePOS.INT_ZERO;
					outputList.add(fixWords[StablePOS.INT_ZERO].toString());
				}
				stringBuilder.delete(StablePOS.INT_ZERO, stringBuilder.length());
				stringBuilder=
neroController.getBinaryForestRecurWordOneTime(stringBuilder.append(inputString
						.charAt(charPosition)), inputString, charPosition, inputStringLength,
forestRoots, forestDepth
						, charPosition+ StablePOS.INT_ONE);
				String countWordNode= stringBuilder.toString();
				int compare= countInputStringLength= countWordNode.length();
				if (compare== StablePOS.INT_ONE) {
					outputList.add(countWordNode);
					fixWords[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWords[StablePOS.INT_ZERO].length());
					fixWords[StablePOS.INT_ZERO].append(countWordNode);
					continue Here;
				}
				if (compare== StablePOS.INT_TWO) {
					countInputStringLength=
nlpController.doSlangPartAndPOSCheckForTwoChar(countInputStringLength, outputList
							, stringBuilder, wordsForest, fixWords, posController, charPosition,
```

```
inputString);
                                        continue Here;
                                }
                                if (compare== StablePOS.INT_THREE) {
                                        I_FixWords(charPosition, inputString, fixWords);
                                        countInputStringLength=
nlpController.doPOSAndEMMCheckOfThree(countInputStringLength, outputList
                                                        , wordsForest, stringBuilder, fixWords, posController, charPosition,
inputString);
                                        continue Here;
                                }
                                if (compare== StablePOS.INT_FOUR) {
                                        I_FixWords(charPosition, inputString, fixWords);
                                        countInputStringLength= nlpController.doSlangCheck(countInputStringLength,
outputList, stringBuilder
                                                        , wordsForest, fixWords, posController, charPosition, inputString);
                                }
                        }
                return outputList;
        }

        public void I_FixWords(int charPosition, String inputString, StringBuilder[] fixWords) {
                fixWords[StablePOS.INT_ONE].delete(StablePOS.INT_ZERO, fixWords[StablePOS.INT_ONE].length());
                if (charPosition + StablePOS.INT_EIGHT < inputString.length()) {
                        fixWords[StablePOS.INT_ONE].append(inputString.substring(charPosition + StablePOS.INT_THREE
                                , charPosition + StablePOS.INT_EIGHT));
                        return;
                }
                fixWords[StablePOS.INT_ONE].append(inputString.substring(charPosition + StablePOS.INT_THREE
                        , inputString.length()));
        }



        public String[] parserEnglishString(String englishString) {
                String[] words = englishString.replaceAll(StablePOS.NLP_SPASE_REP, StablePOS.SPACE_STRING)
                        .split(StablePOS.SPACE_STRING);
                if(StablePOS.INT_ZERO == words.length ) {
                        return new String[] {StablePOS.SPACE_STRING};
                }
                return words;
        }
}
```

```
package OEI.ME.analysis.E;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.FMHMMNode;
import OCI.ME.analysis.C.BinaryForest_A;
import PEQ.AMV.ECS.test.SensingTest;
import SVQ.stable.StablePOS;

public class BinaryForest_AE extends AE implements BinaryForest_A {

        public void studyNewWord(String study, String token, String posStudy) {
                //learn new word
                FMHMMNode fFHMMNode= forestRoots.get(Long.valueOf(study.charAt(StablePOS.INT_ZERO)));
```

```
		Map<String, Integer> map;
		if(null== fFHMMNode) {
			fFHMMNode= new FMHMMNode();
			map= new ConcurrentHashMap<>();
		}else {
			map= fFHMMNode.getNext();
		}
		map.put(token, StablePOS.INT_ONE);
		fFHMMNode.I_Next(map);
		forestRoots.put(Long.valueOf(study.charAt(StablePOS.INT_ZERO)), fFHMMNode);
		//learn new pos
		fHMMList.studyNewPos(study+token, posStudy);
	}

	@Override
	public Map<String, String> getStudyPos() {
		return fHMMList.getStudyPos();
	}

	@Override
	public Map<String, String> getPinYin() {
		return fHMMList.getFullCnToPy();
	}

	@Override
	public Map<String, String> getCtT() {
		return fHMMList.getFullCnToTt();
	}

	@Override
	public Map<String, String> getCtK() {
		return fHMMList.getFullCnToKo();
	}


	@Override
	public Map<String, String> getCtJ() {
		return fHMMList.getFullCnToJp();
	}

	@Override
	public Map<String, String> getCtR() {
		// TODO Auto-generated method stub
		return fHMMList.getFullCnToRs();
	}

	@Override
	public Map<String, String> getCtA() {
		// TODO Auto-generated method stub
		return fHMMList.getFullCnToAb();
	}


	@Override
	public SensingTest getSensingTest() {
		// TODO Auto-generated method stub
		return this.sensingTest;
	}

	public Map<String, String> getPosEnToCn() {
		return fHMMList.getPosEnToCn();
	}
```

```
	public Map<String, String> getPosEnToEn() {
		return fHMMList.getPosEnToEn();
	}
	@Override
	public Map<String, String> getPosCnToCn() {
		return fHMMList.getPosCnToCn();
	}

	public Map<String, String> getFullEnToCn() {
		return fHMMList.getFullEnToCn();
	}

	public Map<String, String> getFullCnToEn() {
		return fHMMList.getFullCnToEn();
	}

	public Map<String, String> getEnToCn() {
		return fHMMList.getEnToCn();
	}

	public Map<String, String> getCnToEn() {
		return fHMMList.getCnToEn();
	}
}
```

----------==========

```
package OEI.ME.analysis.E;
import java.io.IOException;
import java.util.Iterator;
import java.util.List;
import java.util.Map;



import java.util.concurrent.ConcurrentHashMap;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.FMHMMNode;
import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;
import ME.utils.WordFrequencyUtil;
import OCI.AVC.SUQ.SVQ.MPC.fhmm.C.EmotionMap;
import OCI.ME.analysis.C.CogsBinaryForest_A;
import SVQ.stable.StablePOS;
public class CogsBinaryForest_AE extends BinaryForest_AE implements CogsBinaryForest_A {
	public Map<String, WordFrequency> parserStringByReturnFrequencyMap(String inputString) {
		Map<String, String> wordsForest = fHMMList.getPosCnToCn();
		Map<String, WordFrequency> outputList = new ConcurrentHashMap<>();
		Map<Long, FMHMMNode> forestRoots = fHMMList.getMap();
		int inputStringLength = inputString.length();
		int forestDepth = StablePOS.INT_ZERO;
		int countInputStringLength;
		StringBuilder[] fixWords = new StringBuilder[StablePOS.INT_TWO];
		fixWords[StablePOS.INT_ZERO] = new StringBuilder();
		fixWords[StablePOS.INT_ONE] = new StringBuilder();
		StringBuilder stringBuilder = new StringBuilder();
		int find = StablePOS.INT_ZERO;
		Here:
			for (int charPosition= StablePOS.INT_ZERO; charPosition< inputStringLength; charPosition
```

```
                                        += (countInputStringLength== StablePOS.INT_ZERO? StablePOS.INT_ONE:
countInputStringLength)) {
                                if(inputString.charAt(charPosition)< StablePOS.INT_ONE_TWO_EIGHT){
                                        if(fixWords[StablePOS.INT_ZERO].length()> StablePOS.INT_ZERO) {

        if(fixWords[StablePOS.INT_ZERO].charAt(fixWords[StablePOS.INT_ZERO].length()- StablePOS.INT_ONE)
                                                                < StablePOS.INT_ONE_TWO_EIGHT) {

        fixWords[StablePOS.INT_ZERO].append(inputString.charAt(charPosition));
                                                        countInputStringLength= StablePOS.INT_ONE;
                                                        find = StablePOS.INT_ONE;
                                                        continue Here;
                                                }
                                                fixWords[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWords[StablePOS.INT_ZERO].length());
                                        }
                                        find=StablePOS.INT_ONE;
                                        fixWords[StablePOS.INT_ZERO].append(inputString.charAt(charPosition));
                                        countInputStringLength = StablePOS.INT_ONE;
                                        continue Here;
                                }
                                if(find == StablePOS.INT_ONE) {
                                        find = StablePOS.INT_ZERO;
                                        WordFrequencyUtil.WordFrequencyFindCheck(outputList, fixWords);
                                }
                                stringBuilder.delete(StablePOS.INT_ZERO, stringBuilder.length());
                                stringBuilder =
neroController.getBinaryForestRecurWordOneTime(stringBuilder.append(inputString
                                                        .charAt(charPosition)), inputString, charPosition, inputStringLength,
forestRoots, forestDepth
                                                        , charPosition + StablePOS.INT_ONE);
                                String countWordNode = stringBuilder.toString();
                                int compare = countInputStringLength = countWordNode.length();
                                if (compare == StablePOS.INT_ONE) {
                                        WordFrequencyUtil.WordFrequencyCompareCheck(outputList, fixWords,
countWordNode);
                                        fixWords[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWords[StablePOS.INT_ZERO].length());
                                        fixWords[StablePOS.INT_ZERO].append(countWordNode);
                                        continue Here;
                                }
                                if (compare == StablePOS.INT_TWO) {
                                        countInputStringLength =
nlpController.doSlangPartAndPOSCheckForTwoCharForMap(countInputStringLength
                                                                , outputList, stringBuilder, wordsForest, fixWords, posController);
                                        continue Here;
                                }
                                if (compare == StablePOS.INT_THREE) {
                                        I_FixWords(charPosition, inputString, fixWords);
                                        countInputStringLength =
nlpController.doPOSAndEMMCheckOfThreeForMap(countInputStringLength, outputList
                                                        , wordsForest, stringBuilder, fixWords, posController);
                                        continue Here;
                                }
                                if (compare == StablePOS.INT_FOUR) {
                                        I_FixWords(charPosition, inputString, fixWords);
                                        countInputStringLength =
nlpController.doSlangCheckForMap(countInputStringLength, outputList, stringBuilder
```

```
                                                        , wordsForest, fixWords, posController);
                                }
                        }
                return outputList;
        }

        public Map<String, WordFrequency> getWordFrequencyMap(List<String> sets) throws IOException {
                Map<String, WordFrequency> map = new ConcurrentHashMap<>();
                Iterator <String> iterator = sets.iterator();
                Here:
                        while(iterator.hasNext()){
                                String setOfi = iterator.next();
                                if (map.containsKey(setOfi)) {
                                        WordFrequency wordFrequency = map.get(setOfi);
                                        wordFrequency.I_Frequency(wordFrequency.getFrequency() +
StablePOS.INT_ONE);
                                        map.put(setOfi, wordFrequency);
                                        continue Here;
                                }
                                WordFrequency wordFrequency = new WordFrequency();
                                wordFrequency.I_Frequency(StablePOS.INT_ONE);
                                wordFrequency.I_Word(setOfi);
                                map.put(setOfi, wordFrequency);
                        }
                return map;
        }
        //计算函数以后移出 DNA元基组 到RNA.
        public List<WordFrequency> sortWordFrequencyMap(Map<String, WordFrequency> map) throws IOException {
                List<WordFrequency> list = quick6DLuoYaoguangSort.frequencyWordMapToList(map);
                quick6DLuoYaoguangSort.quick6DLuoYaoGuangSortWordFrequency(list, StablePOS.INT_ZERO
                                , list.size() - StablePOS.INT_ONE);
                return list;
        }

        public Map<Integer, WordFrequency> getWordFrequencyByReturnSortMap(List<String> sets) throws IOException {
                return sortWordFrequencyMapToSortMap(getWordFrequencyMap(sets));
        }

        public Map<Integer, WordFrequency> sortWordFrequencyMapToUnsortMap(Map<String, WordFrequency> map){
                Map<Integer, WordFrequency> listMap = quick6DLuoYaoguangSort.frequencyWordMapToMap(map);
                return listMap;
        }

        public Map<Integer, WordFrequency> sortWordFrequencyMapToSortMap(Map<String, WordFrequency> map){
                Map<Integer, WordFrequency> listMap = quick6DLuoYaoguangSort.frequencyWordMapToMap(map);
                quick6DLuoYaoguangSort.quick6DLuoYaoGuangSortWordFrequency(listMap, StablePOS.INT_ZERO
                                , listMap.size() - StablePOS.INT_ONE);
                return listMap;
        }

        public Map<String, WordFrequency> parserMixStringByReturnFrequencyMap(String mixedString) {




                mixedString += StablePOS.SPACE_STRING;
                Map<String, String> wordsForest = fHMMList.getPosCnToCn();
                Map<String, WordFrequency> outputList = new ConcurrentHashMap<>();
                Map<Long, FMHMMNode> forestRoots = fHMMList.getMap();//.getRoot();
                int inputStringLength = mixedString.length();
                int forestDepth = StablePOS.INT_ZERO;
                int countInputStringLength;
                StringBuilder[] fixWords = new StringBuilder[StablePOS.INT_TWO];
```

```
		fixWords[StablePOS.INT_ZERO] = new StringBuilder();
		fixWords[StablePOS.INT_ONE] = new StringBuilder();
		StringBuilder stringBuilder = new StringBuilder();
		int find = StablePOS.INT_ZERO;
		Here:
			for (int charPosition = StablePOS.INT_ZERO; charPosition < inputStringLength; charPosition
					+= (countInputStringLength == StablePOS.INT_ZERO ? StablePOS.INT_ONE :
countInputStringLength)) {
				//luan ma
				if(mixedString.charAt(charPosition) < StablePOS.INT_ONE_TWO_EIGHT && charPosition
< mixedString.length()
						- StablePOS.INT_ONE){
					if(find == StablePOS.INT_ZERO) {
						fixWords[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWords[StablePOS.INT_ZERO].length());
					}
					fixWords[StablePOS.INT_ZERO].append(mixedString.charAt(charPosition));
					countInputStringLength = StablePOS.INT_ONE;
					find = StablePOS.INT_ONE;
					continue Here;
				}
				if(find == StablePOS.INT_ONE) {
					find = StablePOS.INT_ZERO;
					Iterator<String> it =
fHMMList.englishStringToWordsList(fixWords[StablePOS.INT_ZERO].toString()).iterator();
					while(it.hasNext()) {
						String temp=it.next();
						if (outputList.containsKey(temp)) {
							WordFrequency wordFrequency = outputList.get(temp);
							wordFrequency.I_Frequency(wordFrequency.getFrequency() +
StablePOS.INT_ONE);
							outputList.put(temp, wordFrequency);
						} else {
							WordFrequency wordFrequency = new WordFrequency();
							wordFrequency.I_Frequency(StablePOS.INT_ONE);
							wordFrequency.I_Word(temp);
							outputList.put(temp, wordFrequency);
						}
					}
					fixWords[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWords[StablePOS.INT_ZERO].length());
				}
				stringBuilder.delete(StablePOS.INT_ZERO, stringBuilder.length());
				stringBuilder =
neroController.getBinaryForestRecurWordOneTime(stringBuilder.append(mixedString
						.charAt(charPosition)), mixedString, charPosition, inputStringLength,
forestRoots, forestDepth
						, charPosition + StablePOS.INT_ONE);
				String countWordNode = stringBuilder.toString();
				int compare = countInputStringLength = countWordNode.length();
				if (compare == StablePOS.INT_TWO) {
					countInputStringLength =
nlpController.doSlangPartAndPOSCheckForTwoCharForMap(countInputStringLength
							, outputList, stringBuilder, wordsForest, fixWords, posController);
					continue Here;
				}
				if (compare == StablePOS.INT_THREE) {



					I_FixWords(charPosition, mixedString, fixWords);
```

```
                                        countInputStringLength =
nlpController.doPOSAndEMMCheckOfThreeForMap(countInputStringLength, outputList
                                                        , wordsForest, stringBuilder, fixWords, posController);
                                        continue Here;
                                }
                                if (compare == StablePOS.INT_ONE) {
                                        if (outputList.containsKey(countWordNode)) {
                                                WordFrequency wordFrequency = outputList.get(countWordNode);
                                                wordFrequency.I_Frequency(wordFrequency.getFrequency() +
StablePOS.INT_ONE);
                                                outputList.put(countWordNode, wordFrequency);
                                        } else {
                                                WordFrequency wordFrequency = new WordFrequency();
                                                wordFrequency.I_Frequency(StablePOS.INT_ONE);
                                                wordFrequency.I_Word(fixWords[StablePOS.INT_ZERO].toString());
                                                outputList.put(countWordNode, wordFrequency);
                                        }
                                        fixWords[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWords[StablePOS.INT_ZERO].length());
                                        fixWords[StablePOS.INT_ZERO].append(countWordNode);
                                        continue Here;
                                }
                                if (compare == StablePOS.INT_FOUR) {
                                        I_FixWords(charPosition, mixedString, fixWords);
                                        countInputStringLength =
nlpController.doSlangCheckForMap(countInputStringLength, outputList, stringBuilder
                                                        , wordsForest, fixWords, posController);
                                }
                        }
                return outputList;
        }

        public List<WordFrequency> getWordFrequency(List<String> sets) throws IOException {
                return sortWordFrequencyMap(getWordFrequencyMap(sets));
        }

        public EmotionMap getEmotionMap() {
                return this.emotionMap;
        }

}
```

```
package OEI.ME.nlp.E;
import java.util.List;
import java.util.Map;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;
import SVQ.stable.StablePOS;
import SVQ.stable.StableMaps;
import ME.utils.WordForestUtil;
//import OCI.ME.nlp.C.NLP_C;
import OCI.ME.nlp.C.Nlp_C_XCDX_A;
//import OCI.ME.pos.C.POS_C;
import OCI.ME.pos.C.Pos_C_XCDX_P;
import OCI.ME.pos.C.Pos_C_XCDX_O;
public class Nlp_CE_XCDX_A implements Nlp_C_XCDX_A{
        public int doSlangPartAndPOSCheckForTwoChar(int countInputStringLength, List<String> outputList
                        , StringBuilder stringBuilder, Map<String, String> wordsForest, StringBuilder[] prefixWord
                        , Pos_C_XCDX_P posUtils, int charPosition, String textInputString){
```

```
                String countWordNode= stringBuilder.toString();




                if (prefixWord[StablePOS.INT_ZERO].length()== StablePOS.INT_ZERO){
                        if(StableMaps.CiTwo.containsKey(countWordNode)) {
                                prefixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
prefixWord[StablePOS.INT_ZERO].length());
                                prefixWord[StablePOS.INT_ZERO].append(countWordNode);
                                outputList.add(countWordNode);
                                return countInputStringLength;
                        }
                        prefixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
prefixWord[StablePOS.INT_ZERO].length());
                        prefixWord[StablePOS.INT_ZERO].append(stringBuilder.charAt(StablePOS.INT_ZERO));
                        outputList.add(StablePOS.EMPTY_STRING+ stringBuilder.charAt(StablePOS.INT_ZERO));
                        return countInputStringLength- StablePOS.INT_ONE;
                }
                String[] strings= new String[StablePOS.INT_TWO];
                strings[StablePOS.INT_ZERO]= String.valueOf(countWordNode.charAt(StablePOS.INT_ZERO));
                strings[StablePOS.INT_ONE]= countWordNode;
                if (StableMaps.mingCi.containsKey(strings[StablePOS.INT_ZERO])){
                        countInputStringLength= posUtils.chuLiMingCiOfTwo(wordsForest, outputList,
countInputStringLength
                                        , strings, prefixWord, charPosition, textInputString);
                        return countInputStringLength;
                }
                if (StableMaps.baDongCi.containsKey(strings[StablePOS.INT_ZERO])){
                        countInputStringLength = posUtils.chuLiBaDongCiOfTwo(wordsForest, outputList,
countInputStringLength
                                        , strings, prefixWord);
                        return countInputStringLength;
                }
                if (StableMaps.jieCi.containsKey(strings[StablePOS.INT_ZERO].toString())){
                        if (StableMaps.dongCi.containsKey(prefixWord[StablePOS.INT_ZERO].toString())){
                                if (!StableMaps.jieCi.containsKey(countWordNode)){
                                        countInputStringLength=
((Pos_C_XCDX_O)posUtils).parserFirstCharOfTwo(countInputStringLength, outputList, strings, prefixWord);
                                        return countInputStringLength;
                                }
                        }
                }
                if (StableMaps.CiTwo.containsKey(countWordNode)){
                        prefixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
prefixWord[StablePOS.INT_ZERO].length());
                        prefixWord[StablePOS.INT_ZERO].append(countWordNode);
                        outputList.add(countWordNode);
                        return countInputStringLength;
                }
                countInputStringLength= ((Pos_C_XCDX_O)posUtils).parserFirstCharOfTwo(countInputStringLength,
outputList, strings, prefixWord);
                return countInputStringLength;
        }

        public int doPOSAndEMMCheckOfThree(int countInputLength, List<String> outputList
                        , Map<String, String> wordsForest, StringBuilder stringBuilder, StringBuilder[] prefixWord
                        , Pos_C_XCDX_P posUtils, int charPosition, String textInputString){
                String inputString= stringBuilder.toString();
                if (StableMaps.CiThree.containsKey(inputString)){
                        prefixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
prefixWord[StablePOS.INT_ZERO].length());
```

```
                        prefixWord[StablePOS.INT_ZERO].append(inputString);
                        outputList.add(inputString);
                        return countInputLength;
                }
                String[] strings= new String[StablePOS.INT_FOUR];
                strings[StablePOS.INT_ZERO]= String.valueOf(inputString.charAt(StablePOS.INT_ZERO));
                strings[StablePOS.INT_ONE]= String.valueOf(inputString.charAt(StablePOS.INT_ZERO))
                                + inputString.charAt(StablePOS.INT_ONE);
                strings[StablePOS.INT_TWO]= String.valueOf(inputString.charAt(StablePOS.INT_ONE))
                                + inputString.charAt(StablePOS.INT_TWO);
                strings[StablePOS.INT_THREE]= String.valueOf(inputString.charAt(StablePOS.INT_TWO));



                if (null== prefixWord[StablePOS.INT_ZERO]){
                        if (StableMaps.CiThree.containsKey(inputString)){
                                prefixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
prefixWord[StablePOS.INT_ZERO].length());
                                prefixWord[StablePOS.INT_ZERO].append(inputString);
                                outputList.add(inputString);
                                return countInputLength;
                        }
                        StringBuilder stringsBuilder= new StringBuilder();
                        countInputLength= doSlangPartAndPOSCheckForTwoChar(--countInputLength, outputList
                                        , stringsBuilder.append(strings[StablePOS.INT_ONE]), wordsForest, prefixWord,
posUtils, charPosition, textInputString);
                        return countInputLength;
                }
                if (!StableMaps.CiOne.containsKey(strings[StablePOS.INT_ZERO])){
                        StringBuilder stringsBuilder= new StringBuilder();
                        countInputLength= doSlangPartAndPOSCheckForTwoChar(--countInputLength, outputList
                                        , stringsBuilder.append(strings[StablePOS.INT_ONE]), wordsForest, prefixWord,
posUtils, charPosition, textInputString);
                        return countInputLength;
                }
                if(StableMaps.lianCi.containsKey(strings[StablePOS.INT_THREE])) {
                        countInputLength = posUtils.chuLiLianCiPostFixOfThree(wordsForest, outputList, countInputLength,
strings, prefixWord);
                        return countInputLength;
                }
                if (StableMaps.lianCi.containsKey(strings[StablePOS.INT_ZERO])){
                        countInputLength = posUtils.chuLiLianCiOfThree(wordsForest, outputList, countInputLength, strings,
prefixWord);
                        return countInputLength;
                }
                if (StableMaps.jieCi.containsKey(strings[StablePOS.INT_ZERO])){
                        countInputLength = posUtils.chuLiJieCiOfThree(wordsForest, outputList, countInputLength, strings,
prefixWord);
                        return countInputLength;
                }
                if (StableMaps.zhuCi.containsKey(strings[StablePOS.INT_ZERO])){
                        countInputLength = posUtils.chuLiZhuCiOfThree(wordsForest, outputList, countInputLength, strings,
prefixWord);
                        return countInputLength;
                }
                if (StableMaps.liangCi.containsKey(strings[StablePOS.INT_ZERO])){
                        countInputLength = posUtils.chuLiLiangCiOfThree(wordsForest, outputList, countInputLength, strings,
prefixWord);
                        return countInputLength;
                }
```

```
                if (StableMaps.mingCi.containsKey(strings[StablePOS.INT_ZERO])){
                        countInputLength= posUtils.chuLiMingCiOfThree(wordsForest, outputList, countInputLength, strings,
prefixWord);
                        return countInputLength;
                }
                if (StableMaps.shiTaiCi.containsKey(strings[StablePOS.INT_ZERO])){
                        countInputLength= posUtils.chuLiShiTaiCiOfThree(wordsForest, outputList, countInputLength,
strings, prefixWord);
                        return countInputLength;
                }
                if
(StableMaps.dongCi.containsKey(strings[StablePOS.INT_ZERO])||StableMaps.fuCi.containsKey(strings[StablePOS.INT_ZERO])
){
                        if(StableMaps.zhuCi.containsKey(prefixWord[StablePOS.INT_ZERO].toString())
                                        && (StableMaps.liangCi.containsKey(strings[StablePOS.INT_TWO])
                                        || StableMaps.jieCi.containsKey(strings[StablePOS.INT_TWO]))) {
                                countInputLength = ((Pos_C_XCDX_O)posUtils).parserFirstCharOfThree(countInputLength,
outputList, strings, prefixWord);
                                return countInputLength;



                        }

        if(StableMaps.fuCi.containsKey(strings[StablePOS.INT_TWO])||StableMaps.mingCi.containsKey(strings[StablePOS.IN
T_TWO])
                                        ||StableMaps.daiCi.containsKey(strings[StablePOS.INT_TWO])) {
                                countInputLength = ((Pos_C_XCDX_O)posUtils).parserFirstCharOfThree(countInputLength,
outputList, strings, prefixWord);
                                return countInputLength;
                        }
                }
                if (StableMaps.fuCi.containsKey(strings[StablePOS.INT_ZERO])){
                        countInputLength= posUtils.chuLiFuCiOfThree(wordsForest, outputList, countInputLength, strings,
prefixWord);
                        return countInputLength;
                }
                if(StableMaps.CiTwo.containsKey(strings[StablePOS.INT_ONE])) {
                        StringBuilder stringsBuilder= new StringBuilder();
                        countInputLength= doSlangPartAndPOSCheckForTwoChar(--countInputLength, outputList,
stringsBuilder.append(strings[StablePOS.INT_ONE])
                                        , wordsForest, prefixWord, posUtils, charPosition, textInputString);
                        return countInputLength;
                }
                outputList.add(strings[StablePOS.INT_ZERO]);
                prefixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
prefixWord[StablePOS.INT_ZERO].length());
                prefixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_ZERO]);
                return StablePOS.INT_ONE;
        }


        public int doSlangPartAndPOSCheckForTwoCharForMap(int countInputStringLength, Map<String, WordFrequency>
outputList
                        , StringBuilder stringBuilder, Map<String, String> wordsForest, StringBuilder[] prefixWord
                        , Pos_C_XCDX_P posUtils){
                String countWordNode= stringBuilder.toString();
                if (!wordsForest.containsKey(countWordNode)){
                        WordForestUtil.wordsForestNotContainsKey(outputList, countWordNode, prefixWord);
                        return --countInputStringLength;
                }
                if (prefixWord[StablePOS.INT_ZERO].length()== StablePOS.INT_ZERO){
```

```
                        WordForestUtil.prefixWordEqualZero(outputList, countWordNode, prefixWord);
                        return countInputStringLength;
                }
                String[] strings= new String[StablePOS.INT_TWO];
                strings[StablePOS.INT_ZERO]= String.valueOf(countWordNode.charAt(StablePOS.INT_ZERO));
                strings[StablePOS.INT_ONE]= String.valueOf(countWordNode.charAt(StablePOS.INT_ZERO))
                                + String.valueOf(countWordNode.charAt(StablePOS.INT_ONE));
                if (wordsForest.containsKey(strings[StablePOS.INT_ZERO])){
                        if (wordsForest.get(strings[StablePOS.INT_ZERO]).contains(StablePOS.NLP_CI_MING)){
                                countInputStringLength= posUtils.chuLiMingCiOfTwoForMap(wordsForest, outputList,
countInputStringLength
                                                , strings, prefixWord);
                                return countInputStringLength;
                        }
                }
                if (wordsForest.containsKey(strings[StablePOS.INT_ONE])){
                        WordForestUtil.wordsForestContainsKey(outputList, countWordNode, prefixWord);
                        return countInputStringLength;
                }
                return StablePOS.INT_ZERO;
        }

        public int doPOSAndEMMCheckOfThreeForMap(int countInputLength, Map<String, WordFrequency> outputList
                        , Map<String, String> wordsForest, StringBuilder stringBuilder, StringBuilder[] prefixWord
                        , Pos_C_XCDX_P posUtils){
                String inputString= stringBuilder.toString();
                if (wordsForest.containsKey(inputString)){
                        WordForestUtil.wordsForestContainsKey(outputList, inputString, prefixWord);
                        return countInputLength;




                }
                String[] strings= new String[StablePOS.INT_FOUR];
                strings[StablePOS.INT_ZERO]= String.valueOf(inputString.charAt(StablePOS.INT_ZERO));
                strings[StablePOS.INT_ONE]= String.valueOf(inputString.charAt(StablePOS.INT_ZERO))
                                + inputString.charAt(StablePOS.INT_ONE);
                strings[StablePOS.INT_TWO]= String.valueOf(inputString.charAt(StablePOS.INT_ONE)
                                + inputString.charAt(StablePOS.INT_TWO));
                strings[StablePOS.INT_THREE]= String.valueOf(inputString.charAt(StablePOS.INT_TWO));
                if (null== prefixWord[StablePOS.INT_ZERO]){
                        if (wordsForest.containsKey(inputString)){
                                WordForestUtil.wordsForestContainsKey(outputList, inputString, prefixWord);
                                return countInputLength;
                        }
                        StringBuilder stringsBuilder= new StringBuilder();
                        countInputLength= doSlangPartAndPOSCheckForTwoCharForMap(--countInputLength, outputList
                                        , stringsBuilder.append(strings[StablePOS.INT_ONE]), wordsForest, prefixWord,
posUtils);
                        return countInputLength;
                }
                if (!wordsForest.containsKey(strings[StablePOS.INT_ZERO])){
                        StringBuilder stringsBuilder= new StringBuilder();
                        countInputLength= doSlangPartAndPOSCheckForTwoCharForMap(--countInputLength, outputList
                                        , stringsBuilder.append(strings[StablePOS.INT_ONE]), wordsForest, prefixWord,
posUtils);
                        return countInputLength;
                }
                if (StableMaps.zhuCi.containsKey(strings[StablePOS.INT_ZERO])){
                        countInputLength= posUtils.chuLiZhuCiOfThreeForMap(wordsForest, outputList, countInputLength
                                        , strings, prefixWord);
```

```
                return countInputLength;
            }
            if (StableMaps.liangCi.containsKey(strings[StablePOS.INT_ZERO])){
                countInputLength= posUtils.chuLiLiangCiOfThreeForMap(wordsForest, outputList, countInputLength
                        , strings, prefixWord);
                return countInputLength;
            }
            if (StableMaps.zhuCi.containsKey(strings[StablePOS.INT_ZERO])){
                countInputLength= posUtils.chuLiJieCiOfThreeForMap(wordsForest, outputList, countInputLength
                        , strings, prefixWord);
                return countInputLength;
            }
            if (StableMaps.lianCi.containsKey(strings[StablePOS.INT_ZERO])){
                countInputLength= posUtils.chuLiLianCiOfThreeForMap(wordsForest, outputList, countInputLength
                        , strings, prefixWord);
                return countInputLength;
            }
            StringBuilder stringsBuilder= new StringBuilder();
            countInputLength= doSlangPartAndPOSCheckForTwoCharForMap(--countInputLength, outputList
                    , stringsBuilder.append(strings[StablePOS.INT_ONE]), wordsForest, prefixWord, posUtils);
            return countInputLength;
        }
}
```

```
package OEI.ME.nlp.E;
import java.util.List;
import java.util.Map;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;
import SVQ.stable.StablePOS;
import SVQ.stable.StableMaps;
import ME.utils.WordForestUtil;


//import OCI.ME.nlp.C.NLP_C;
import OCI.ME.nlp.C.Nlp_C_XCDX_S;
//import OCI.ME.pos.C.POS_C;
import OCI.ME.pos.C.Pos_C_XCDX_P;
//import OCI.ME.pos.C.Pos_C_XCDX_O;

//繁衍和继承是新陈代谢的一种体现.
public class Nlp_CE_XCDX_S extends Nlp_CE_XCDX_A implements Nlp_C_XCDX_S{

// 2个月研究发现 词性越来越多, 根据笛摩根定律, 先把未知词汇也添加到条件中. 之后采用 排除法优化.
//          if(StableMaps.jieCi.containsKey(preRegister)|| StableMaps.mingCi.containsKey(preRegister)||
StableMaps.xingRongCi.containsKey(preRegister)
//          || StableMaps.fuCi.containsKey(preRegister)|| StableMaps.dongCi.containsKey(preRegister)||
StableMaps.lianCi.containsKey(preRegister)
//          || StableMaps.liangCi.containsKey(preRegister)|| StableMaps.xingWeiCi.containsKey(preRegister)||
StableMaps.shiTaiCi.containsKey(preRegister)
//          || StableMaps.zhuCi.containsKey(preRegister)) {
//if(StableMaps.mingCi.containsKey(postRegister)|| StableMaps.dongCi.containsKey(postRegister)||
StableMaps.lianCi.containsKey(postRegister)
//                  || StableMaps.xingRongCi.containsKey(postRegister)|| StableMaps.xingWeiCi.containsKey(postRegister)||
StableMaps.liangCi.containsKey(preRegister)
```

```
//                || StableMaps.fuCi.containsKey(postRegister)|| StableMaps.jieCi.containsKey(postRegister)) {
        public int doSlangCheck(int countInputStringLength, List<String> output, StringBuilder stringBuilder,
                        Map<String, String> wordsForest, StringBuilder[] prefixWord, Pos_C_XCDX_P posUtils, int
charPosition, String textInputString){
                String inputString = stringBuilder.toString();
                if (StableMaps.CiFour.containsKey(inputString)){
                        output.add(inputString);
                        prefixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
prefixWord[StablePOS.INT_ZERO].length());
                        prefixWord[StablePOS.INT_ZERO].append(inputString);
                        return countInputStringLength;
                }//will make pre 3 or post 3 check. now finished pre 3 .20190330
                String preRegister= StablePOS.EMPTY_STRING+ inputString.charAt(StablePOS.INT_ZERO)+
inputString.charAt(StablePOS.INT_ONE);
                String inRegister= StablePOS.EMPTY_STRING+ inputString.charAt(StablePOS.INT_ONE)+
inputString.charAt(StablePOS.INT_TWO);
                String postRegister= StablePOS.EMPTY_STRING+ inputString.charAt(StablePOS.INT_TWO)+
inputString.charAt(StablePOS.INT_THREE);
                if(StableMaps.dongCi.containsKey(StablePOS.EMPTY_STRING+ inputString.charAt(StablePOS.INT_THREE)
                + prefixWord[StablePOS.INT_ONE].charAt(StablePOS.INT_ZERO))) {
                        countInputStringLength= doPOSAndEMMCheckOfThree(--countInputStringLength, output,
wordsForest
                                        , stringBuilder.delete(StablePOS.INT_THREE, StablePOS.INT_FOUR), prefixWord,
posUtils, charPosition, textInputString);
                        return countInputStringLength;
                }
                if (StableMaps.CiTwo.containsKey(preRegister)){
                        if (StableMaps.CiTwo.containsKey(postRegister)){
                                String string= StablePOS.EMPTY_STRING+ inputString.charAt(StablePOS.INT_ZERO);
                                if(StableMaps.xingWeiCi.containsKey(prefixWord[StablePOS.INT_ZERO].toString())
                                                &&StableMaps.shiTaiCi.containsKey(string)) {
                                        output.add(string);
                                        prefixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
prefixWord[StablePOS.INT_ZERO].length());
                                        prefixWord[StablePOS.INT_ZERO].append(string);
                                        return countInputStringLength- StablePOS.INT_THREE;
                                }
                                if(StableMaps.zhuCi.containsKey(string)){
                                        String[] strings= new String[StablePOS.INT_FOUR];
                                        strings[StablePOS.INT_ZERO]=
String.valueOf(inputString.charAt(StablePOS.INT_ZERO));
                                        strings[StablePOS.INT_ONE]=
String.valueOf(inputString.charAt(StablePOS.INT_ZERO))
                                                        + inputString.charAt(StablePOS.INT_ONE);
                                        strings[StablePOS.INT_TWO]=
String.valueOf(inputString.charAt(StablePOS.INT_ONE))
                                                        + inputString.charAt(StablePOS.INT_TWO);
                                        strings[StablePOS.INT_THREE]=
String.valueOf(inputString.charAt(StablePOS.INT_TWO));



                                        countInputStringLength= posUtils.chuLiZhuCiOfThree(wordsForest, output,
countInputStringLength-StablePOS.INT_ONE, strings, prefixWord);
                                        return countInputStringLength;
                                }
                                output.add(preRegister);
                                prefixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
prefixWord[StablePOS.INT_ZERO].length());
                                prefixWord[StablePOS.INT_ZERO].append(preRegister);
                                return countInputStringLength-StablePOS.INT_TWO;
```

```
                        }
                }
                if(StableMaps.CiThree.containsKey(preRegister+
inputString.charAt(StablePOS.INT_TWO))&& !StableMaps.CiTwo.containsKey(postRegister)) {
                        prefixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
prefixWord[StablePOS.INT_ZERO].length());
                        prefixWord[StablePOS.INT_ZERO].append(preRegister+ inputString.charAt(StablePOS.INT_TWO));
                        output.add(preRegister+ inputString.charAt(StablePOS.INT_TWO));
                        return countInputStringLength- StablePOS.INT_ONE ;
                }
                if(StableMaps.CiTwo.containsKey(preRegister)&& StableMaps.CiTwo.containsKey(inRegister)) {
                        countInputStringLength= doPOSAndEMMCheckOfThree(--countInputStringLength, output,
wordsForest
                                        , stringBuilder.delete(StablePOS.INT_THREE, StablePOS.INT_FOUR), prefixWord,
posUtils, charPosition, textInputString);
                        return countInputStringLength;
                }
                if(StableMaps.CiTwo.containsKey(preRegister)) {
                        countInputStringLength= doSlangPartAndPOSCheckForTwoChar(countInputStringLength-
StablePOS.INT_TWO, output
                                        , stringBuilder.delete(StablePOS.INT_TWO, StablePOS.INT_FOUR), wordsForest,
prefixWord, posUtils, charPosition, textInputString);
                        return countInputStringLength;
                }
                output.add(StablePOS.EMPTY_STRING+ inputString.charAt(StablePOS.INT_ZERO));
                prefixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
prefixWord[StablePOS.INT_ZERO].length());
                prefixWord[StablePOS.INT_ZERO].append(StablePOS.EMPTY_STRING+
inputString.charAt(StablePOS.INT_ZERO));
                return countInputStringLength= StablePOS.INT_ONE;
        }
//卡诺图化简.PCA阀门分流. 卷积催化, .原来备注这里 ,20190523
//        if(!wordsForest.containsKey(preRegister)&&
(wordsForest.containsKey(inRegister)||wordsForest.containsKey(postRegister))) {
//        if(wordsForest.containsKey(preRegister+ inputString.charAt(StableData.INT_TWO))) {
//                output.add(preRegister+ inputString.charAt(StableData.INT_TWO));
//                prefixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO,
prefixWord[StableData.INT_ZERO].length());
//                prefixWord[StableData.INT_ZERO].append(preRegister+ inputString.charAt(StableData.INT_TWO));
//                return countInputStringLength- StableData.INT_ONE;
//        }
//        output.add(StableData.EMPTY_STRING+ inputString.charAt(StableData.INT_ZERO));
//        prefixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, prefixWord[StableData.INT_ZERO].length());
//        prefixWord[StableData.INT_ZERO].append(StableData.EMPTY_STRING+
inputString.charAt(StableData.INT_ZERO));
//        return countInputStringLength- StableData.INT_THREE;
//}

//if(wordsForest.containsKey(preRegister)&& wordsForest.containsKey(inRegister+
inputString.charAt(StableData.INT_THREE))) {
//        countInputStringLength= doPOSAndEMMCheckOfThree(--countInputStringLength, output, wordsForest
//                        , stringBuilder.delete(StableData.INT_THREE, StableData.INT_FOUR), prefixWord, posUtils,
charPosition, textInputString);
//        return countInputStringLength;
//}
//if(wordsForest.containsKey(preRegister)) {
//        countInputStringLength= doPOSAndEMMCheckOfThree(--countInputStringLength, output, wordsForest
//                        , stringBuilder.delete(StableData.INT_THREE, StableData.INT_FOUR), prefixWord, posUtils,
charPosition, textInputString);
//        return countInputStringLength;
//}
        public int doSlangCheckForMap(int countInputStringLength, List<String> output, StringBuilder stringBuilder
                        , Map<String, String> wordsForest, StringBuilder[] prefixWord, Pos_C_XCDX_P posUtils, int
```

```
charPosition,



String textInputString){
                String inputString= stringBuilder.toString();
                if (wordsForest.containsKey(inputString)){
                        output.add(inputString);
                        prefixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
prefixWord[StablePOS.INT_ZERO].length());
                        prefixWord[StablePOS.INT_ZERO].append(inputString);
                        return countInputStringLength;
                }
                countInputStringLength= doPOSAndEMMCheckOfThree(--countInputStringLength, output, wordsForest
                                , stringBuilder.delete(StablePOS.INT_THREE, StablePOS.INT_FOUR), prefixWord,
posUtils, charPosition, textInputString);
                return countInputStringLength;
        }

        public int doSlangCheckForMap(int countInputStringLength, Map<String, WordFrequency> output, StringBuilder
stringBuilder
                        , Map<String, String> wordsForest, StringBuilder[] prefixWord, Pos_C_XCDX_P posUtils){
                String inputString= stringBuilder.toString();
                if (wordsForest.containsKey(inputString)){
                        WordForestUtil.wordsForestContainsKey(output, inputString, prefixWord);
                        return countInputStringLength;
                }
                if(StableMaps.mingCi.containsKey(StablePOS.EMPTY_STRING+
inputString.charAt(StablePOS.INT_ZERO)+ inputString.charAt(StablePOS.INT_ONE))) {
                        if(StableMaps.mingCi.containsKey(StablePOS.EMPTY_STRING+
inputString.charAt(StablePOS.INT_TWO)+ inputString.charAt(StablePOS.INT_THREE))) {
                                WordForestUtil.wordsForestContainsKey(output, StablePOS.EMPTY_STRING+
inputString.charAt(StablePOS.INT_ZERO)+ inputString.charAt(StablePOS.INT_ONE), prefixWord);
                                return countInputStringLength;
                        }
                }
                countInputStringLength= doPOSAndEMMCheckOfThreeForMap(--countInputStringLength, output,
wordsForest
                                , stringBuilder.delete(StablePOS.INT_THREE, StablePOS.INT_FOUR), prefixWord,
posUtils);
                return countInputStringLength;
        }
}
```

```
package OEI.ME.nlp.E;
import java.util.List;

import java.util.Map;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;
import SVQ.stable.StablePOS;
import SVQ.stable.StableMaps;
import ME.utils.WordForestUtil;
import OCI.ME.nlp.C.NLP_C;
//import OCI.ME.pos.C.POS_C;
import OCI.ME.pos.C.Pos_C_XCDX_P;
import OCI.ME.pos.C.Pos_C_XCDX_O;
public class Nlp_CE_XCDX implements NLP_C{
        public int doSlangPartAndPOSCheckForTwoChar(int countInputStringLength, List<String> outputList
```

```
                        , StringBuilder stringBuilder, Map<String, String> wordsForest, StringBuilder[] prefixWord
                        , Pos_C_XCDX_P posUtils, int charPosition, String textInputString){
                String countWordNode= stringBuilder.toString();
                if (prefixWord[StablePOS.INT_ZERO].length()== StablePOS.INT_ZERO){
                        if(StableMaps.CiTwo.containsKey(countWordNode)) {
                                prefixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
prefixWord[StablePOS.INT_ZERO].length());
                                prefixWord[StablePOS.INT_ZERO].append(countWordNode);
                                outputList.add(countWordNode);
                                return countInputStringLength;



                        }
                        prefixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
prefixWord[StablePOS.INT_ZERO].length());
                        prefixWord[StablePOS.INT_ZERO].append(stringBuilder.charAt(StablePOS.INT_ZERO));
                        outputList.add(StablePOS.EMPTY_STRING+ stringBuilder.charAt(StablePOS.INT_ZERO));
                        return countInputStringLength- StablePOS.INT_ONE;
                }
                String[] strings= new String[StablePOS.INT_TWO];
                strings[StablePOS.INT_ZERO]= String.valueOf(countWordNode.charAt(StablePOS.INT_ZERO));
                strings[StablePOS.INT_ONE]= countWordNode;
                if (StableMaps.mingCi.containsKey(strings[StablePOS.INT_ZERO])){
                        countInputStringLength= posUtils.chuLiMingCiOfTwo(wordsForest, outputList,
countInputStringLength
                                        , strings, prefixWord, charPosition, textInputString);
                        return countInputStringLength;
                }
                if (StableMaps.baDongCi.containsKey(strings[StablePOS.INT_ZERO])){
                        countInputStringLength = posUtils.chuLiBaDongCiOfTwo(wordsForest, outputList,
countInputStringLength
                                        , strings, prefixWord);
                        return countInputStringLength;
                }
                if (StableMaps.jieCi.containsKey(strings[StablePOS.INT_ZERO].toString())){
                        if (StableMaps.dongCi.containsKey(prefixWord[StablePOS.INT_ZERO].toString())){
                                if (!StableMaps.jieCi.containsKey(countWordNode)){
                                        countInputStringLength=
((Pos_C_XCDX_O)posUtils).parserFirstCharOfTwo(countInputStringLength, outputList, strings, prefixWord);
                                        return countInputStringLength;
                                }
                        }
                }
                if (StableMaps.CiTwo.containsKey(countWordNode)){
                        prefixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
prefixWord[StablePOS.INT_ZERO].length());
                        prefixWord[StablePOS.INT_ZERO].append(countWordNode);
                        outputList.add(countWordNode);
                        return countInputStringLength;
                }
                countInputStringLength= ((Pos_C_XCDX_O)posUtils).parserFirstCharOfTwo(countInputStringLength,
outputList, strings, prefixWord);
                return countInputStringLength;
        }

        public int doPOSAndEMMCheckOfThree(int countInputLength, List<String> outputList
                        , Map<String, String> wordsForest, StringBuilder stringBuilder, StringBuilder[] prefixWord
                        , Pos_C_XCDX_P posUtils, int charPosition, String textInputString){
                String inputString= stringBuilder.toString();
                if (StableMaps.CiThree.containsKey(inputString)){
                        prefixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
prefixWord[StablePOS.INT_ZERO].length());
```

```
                    prefixWord[StablePOS.INT_ZERO].append(inputString);
                    outputList.add(inputString);
                    return countInputLength;
                }
                String[] strings= new String[StablePOS.INT_FOUR];
                strings[StablePOS.INT_ZERO]= String.valueOf(inputString.charAt(StablePOS.INT_ZERO));
                strings[StablePOS.INT_ONE]= String.valueOf(inputString.charAt(StablePOS.INT_ZERO))
                        + inputString.charAt(StablePOS.INT_ONE);
                strings[StablePOS.INT_TWO]= String.valueOf(inputString.charAt(StablePOS.INT_ONE))
                        + inputString.charAt(StablePOS.INT_TWO);
                strings[StablePOS.INT_THREE]= String.valueOf(inputString.charAt(StablePOS.INT_TWO));
                if (null== prefixWord[StablePOS.INT_ZERO]){
                    if (StableMaps.CiThree.containsKey(inputString)){
                        prefixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
prefixWord[StablePOS.INT_ZERO].length());
                        prefixWord[StablePOS.INT_ZERO].append(inputString);
                        outputList.add(inputString);
                        return countInputLength;
                    }
                    StringBuilder stringsBuilder= new StringBuilder();



                    countInputLength= doSlangPartAndPOSCheckForTwoChar(--countInputLength, outputList
                            , stringsBuilder.append(strings[StablePOS.INT_ONE]), wordsForest, prefixWord,
posUtils, charPosition, textInputString);
                    return countInputLength;
                }
                if (!StableMaps.CiOne.containsKey(strings[StablePOS.INT_ZERO])){
                    StringBuilder stringsBuilder= new StringBuilder();
                    countInputLength= doSlangPartAndPOSCheckForTwoChar(--countInputLength, outputList
                            , stringsBuilder.append(strings[StablePOS.INT_ONE]), wordsForest, prefixWord,
posUtils, charPosition, textInputString);
                    return countInputLength;
                }
                if(StableMaps.lianCi.containsKey(strings[StablePOS.INT_THREE])) {
                    countInputLength = posUtils.chuLiLianCiPostFixOfThree(wordsForest, outputList, countInputLength,
strings, prefixWord);
                    return countInputLength;
                }
                if (StableMaps.lianCi.containsKey(strings[StablePOS.INT_ZERO])){
                    countInputLength = posUtils.chuLiLianCiOfThree(wordsForest, outputList, countInputLength, strings,
prefixWord);
                    return countInputLength;
                }
                if (StableMaps.jieCi.containsKey(strings[StablePOS.INT_ZERO])){
                    countInputLength = posUtils.chuLiJieCiOfThree(wordsForest, outputList, countInputLength, strings,
prefixWord);
                    return countInputLength;
                }
                if (StableMaps.zhuCi.containsKey(strings[StablePOS.INT_ZERO])){
                    countInputLength = posUtils.chuLiZhuCiOfThree(wordsForest, outputList, countInputLength, strings,
prefixWord);
                    return countInputLength;
                }
                if (StableMaps.liangCi.containsKey(strings[StablePOS.INT_ZERO])){
                    countInputLength = posUtils.chuLiLiangCiOfThree(wordsForest, outputList, countInputLength, strings,
prefixWord);
                    return countInputLength;
                }
                if (StableMaps.mingCi.containsKey(strings[StablePOS.INT_ZERO])){
```

```
                        countInputLength= posUtils.chuLiMingCiOfThree(wordsForest, outputList, countInputLength, strings,
prefixWord);
                        return countInputLength;
                }
                if (StableMaps.shiTaiCi.containsKey(strings[StablePOS.INT_ZERO])){
                        countInputLength= posUtils.chuLiShiTaiCiOfThree(wordsForest, outputList, countInputLength,
strings, prefixWord);
                        return countInputLength;
                }
                if
(StableMaps.dongCi.containsKey(strings[StablePOS.INT_ZERO])||StableMaps.fuCi.containsKey(strings[StablePOS.INT_ZERO])
){
                        if(StableMaps.zhuCi.containsKey(prefixWord[StablePOS.INT_ZERO].toString())
                                        && (StableMaps.liangCi.containsKey(strings[StablePOS.INT_TWO])
                                        || StableMaps.jieCi.containsKey(strings[StablePOS.INT_TWO]))) {
                                countInputLength = ((Pos_C_XCDX_O)posUtils).parserFirstCharOfThree(countInputLength,
outputList, strings, prefixWord);
                                return countInputLength;
                        }

        if(StableMaps.fuCi.containsKey(strings[StablePOS.INT_TWO])||StableMaps.mingCi.containsKey(strings[StablePOS.IN
T_TWO])
                                        ||StableMaps.daiCi.containsKey(strings[StablePOS.INT_TWO])) {
                                countInputLength = ((Pos_C_XCDX_O)posUtils).parserFirstCharOfThree(countInputLength,
outputList, strings, prefixWord);
                                return countInputLength;
                        }
                }
                if (StableMaps.fuCi.containsKey(strings[StablePOS.INT_ZERO])){




                        countInputLength= posUtils.chuLiFuCiOfThree(wordsForest, outputList, countInputLength, strings,
prefixWord);
                        return countInputLength;
                }
                if(StableMaps.CiTwo.containsKey(strings[StablePOS.INT_ONE])) {
                        StringBuilder stringsBuilder= new StringBuilder();
                        countInputLength= doSlangPartAndPOSCheckForTwoChar(--countInputLength, outputList,
stringsBuilder.append(strings[StablePOS.INT_ONE])
                                        , wordsForest, prefixWord, posUtils, charPosition, textInputString);
                        return countInputLength;
                }
                outputList.add(strings[StablePOS.INT_ZERO]);
                prefixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
prefixWord[StablePOS.INT_ZERO].length());
                prefixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_ZERO]);
                return StablePOS.INT_ONE;
        }
// 2个月研究发现 词性越来越多, 根据笛摩根定律, 先把未知词汇也添加到条件中. 之后采用 排除法优化.
//          if(StableMaps.jieCi.containsKey(preRegister)|| StableMaps.mingCi.containsKey(preRegister)||
StableMaps.xingRongCi.containsKey(preRegister)
//          || StableMaps.fuCi.containsKey(preRegister)|| StableMaps.dongCi.containsKey(preRegister)||
StableMaps.lianCi.containsKey(preRegister)
//          || StableMaps.liangCi.containsKey(preRegister)|| StableMaps.xingWeiCi.containsKey(preRegister)||
StableMaps.shiTaiCi.containsKey(preRegister)
//          || StableMaps.zhuCi.containsKey(preRegister)) {
//if(StableMaps.mingCi.containsKey(postRegister)|| StableMaps.dongCi.containsKey(postRegister)||
StableMaps.lianCi.containsKey(postRegister)
//                  || StableMaps.xingRongCi.containsKey(postRegister)|| StableMaps.xingWeiCi.containsKey(postRegister)||
StableMaps.liangCi.containsKey(preRegister)
```

```
//                || StableMaps.fuCi.containsKey(postRegister)|| StableMaps.jieCi.containsKey(postRegister)) {
        public int doSlangCheck(int countInputStringLength, List<String> output, StringBuilder stringBuilder,
                        Map<String, String> wordsForest, StringBuilder[] prefixWord, Pos_C_XCDX_P posUtils, int
charPosition, String textInputString){
                String inputString = stringBuilder.toString();
                if (StableMaps.CiFour.containsKey(inputString)){
                        output.add(inputString);
                        prefixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
prefixWord[StablePOS.INT_ZERO].length());
                        prefixWord[StablePOS.INT_ZERO].append(inputString);
                        return countInputStringLength;
                }//will make pre 3 or post 3 check. now finished pre 3 .20190330
                String preRegister= StablePOS.EMPTY_STRING+ inputString.charAt(StablePOS.INT_ZERO)+
inputString.charAt(StablePOS.INT_ONE);
                String inRegister= StablePOS.EMPTY_STRING+ inputString.charAt(StablePOS.INT_ONE)+
inputString.charAt(StablePOS.INT_TWO);
                String postRegister= StablePOS.EMPTY_STRING+ inputString.charAt(StablePOS.INT_TWO)+
inputString.charAt(StablePOS.INT_THREE);
                if(StableMaps.dongCi.containsKey(StablePOS.EMPTY_STRING+ inputString.charAt(StablePOS.INT_THREE)
                + prefixWord[StablePOS.INT_ONE].charAt(StablePOS.INT_ZERO))) {
                        countInputStringLength= doPOSAndEMMCheckOfThree(--countInputStringLength, output,
wordsForest
                                        , stringBuilder.delete(StablePOS.INT_THREE, StablePOS.INT_FOUR), prefixWord,
posUtils, charPosition, textInputString);
                        return countInputStringLength;
                }
                if (StableMaps.CiTwo.containsKey(preRegister)){
                        if (StableMaps.CiTwo.containsKey(postRegister)){
                                String string= StablePOS.EMPTY_STRING+ inputString.charAt(StablePOS.INT_ZERO);
                                if(StableMaps.xingWeiCi.containsKey(prefixWord[StablePOS.INT_ZERO].toString())
                                                &&StableMaps.shiTaiCi.containsKey(string)) {
                                        output.add(string);
                                        prefixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
prefixWord[StablePOS.INT_ZERO].length());
                                        prefixWord[StablePOS.INT_ZERO].append(string);
                                        return countInputStringLength- StablePOS.INT_THREE;
                                }
                                if(StableMaps.zhuCi.containsKey(string)){
                                        String[] strings= new String[StablePOS.INT_FOUR];



                                        strings[StablePOS.INT_ZERO]=
String.valueOf(inputString.charAt(StablePOS.INT_ZERO));
                                        strings[StablePOS.INT_ONE]=
String.valueOf(inputString.charAt(StablePOS.INT_ZERO))
                                                        + inputString.charAt(StablePOS.INT_ONE);
                                        strings[StablePOS.INT_TWO]=
String.valueOf(inputString.charAt(StablePOS.INT_ONE))
                                                        + inputString.charAt(StablePOS.INT_TWO);
                                        strings[StablePOS.INT_THREE]=
String.valueOf(inputString.charAt(StablePOS.INT_TWO));
                                        countInputStringLength= posUtils.chuLiZhuCiOfThree(wordsForest, output,
countInputStringLength-StablePOS.INT_ONE, strings, prefixWord);
                                        return countInputStringLength;
                                }
                                output.add(preRegister);
                                prefixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
prefixWord[StablePOS.INT_ZERO].length());
                                prefixWord[StablePOS.INT_ZERO].append(preRegister);
                                return countInputStringLength-StablePOS.INT_TWO;
```

```
                        }
                }
                if(StableMaps.CiThree.containsKey(preRegister+
inputString.charAt(StablePOS.INT_TWO))&& !StableMaps.CiTwo.containsKey(postRegister)) {
                        prefixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
prefixWord[StablePOS.INT_ZERO].length());
                        prefixWord[StablePOS.INT_ZERO].append(preRegister+ inputString.charAt(StablePOS.INT_TWO));
                        output.add(preRegister+ inputString.charAt(StablePOS.INT_TWO));
                        return countInputStringLength- StablePOS.INT_ONE ;
                }
                if(StableMaps.CiTwo.containsKey(preRegister)&& StableMaps.CiTwo.containsKey(inRegister)) {
                        countInputStringLength= doPOSAndEMMCheckOfThree(--countInputStringLength, output,
wordsForest
                                        , stringBuilder.delete(StablePOS.INT_THREE, StablePOS.INT_FOUR), prefixWord,
posUtils, charPosition, textInputString);
                        return countInputStringLength;
                }
                if(StableMaps.CiTwo.containsKey(preRegister)) {
                        countInputStringLength= doSlangPartAndPOSCheckForTwoChar(countInputStringLength-
StablePOS.INT_TWO, output
                                        , stringBuilder.delete(StablePOS.INT_TWO, StablePOS.INT_FOUR), wordsForest,
prefixWord, posUtils, charPosition, textInputString);
                        return countInputStringLength;
                }
                output.add(StablePOS.EMPTY_STRING+ inputString.charAt(StablePOS.INT_ZERO));
                prefixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
prefixWord[StablePOS.INT_ZERO].length());
                prefixWord[StablePOS.INT_ZERO].append(StablePOS.EMPTY_STRING+
inputString.charAt(StablePOS.INT_ZERO));
                return countInputStringLength= StablePOS.INT_ONE;
        }
//卡诺图化简.PCA阀门分流. 卷积催化, .原来备注这里 ,20190523
//        if(!wordsForest.containsKey(preRegister)&&
(wordsForest.containsKey(inRegister)||wordsForest.containsKey(postRegister))) {
//        if(wordsForest.containsKey(preRegister+ inputString.charAt(StableData.INT_TWO))) {
//                output.add(preRegister+ inputString.charAt(StableData.INT_TWO));
//                prefixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO,
prefixWord[StableData.INT_ZERO].length());
//                prefixWord[StableData.INT_ZERO].append(preRegister+ inputString.charAt(StableData.INT_TWO));
//                return countInputStringLength- StableData.INT_ONE;
//        }
//        output.add(StableData.EMPTY_STRING+ inputString.charAt(StableData.INT_ZERO));
//        prefixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, prefixWord[StableData.INT_ZERO].length());
//        prefixWord[StableData.INT_ZERO].append(StableData.EMPTY_STRING+
inputString.charAt(StableData.INT_ZERO));
//        return countInputStringLength- StableData.INT_THREE;
//}

//if(wordsForest.containsKey(preRegister)&& wordsForest.containsKey(inRegister+
inputString.charAt(StableData.INT_THREE))) {
//        countInputStringLength= doPOSAndEMMCheckOfThree(--countInputStringLength, output, wordsForest
//                        , stringBuilder.delete(StableData.INT_THREE, StableData.INT_FOUR), prefixWord, posUtils,
charPosition, textInputString);
//        return countInputStringLength;
//}
//if(wordsForest.containsKey(preRegister)) {
//        countInputStringLength= doPOSAndEMMCheckOfThree(--countInputStringLength, output, wordsForest




//                        , stringBuilder.delete(StableData.INT_THREE, StableData.INT_FOUR), prefixWord, posUtils,
charPosition, textInputString);
//        return countInputStringLength;
```

```
//}
	public int doSlangCheckForMap(int countInputStringLength, List<String> output, StringBuilder stringBuilder
			, Map<String, String> wordsForest, StringBuilder[] prefixWord, Pos_C_XCDX_P posUtils, int
charPosition, String textInputString){
		String inputString= stringBuilder.toString();
		if (wordsForest.containsKey(inputString)){
			output.add(inputString);
			prefixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
prefixWord[StablePOS.INT_ZERO].length());
			prefixWord[StablePOS.INT_ZERO].append(inputString);
			return countInputStringLength;
		}
		countInputStringLength= doPOSAndEMMCheckOfThree(--countInputStringLength, output, wordsForest
				, stringBuilder.delete(StablePOS.INT_THREE, StablePOS.INT_FOUR), prefixWord,
posUtils, charPosition, textInputString);
		return countInputStringLength;
	}

	public int doSlangPartAndPOSCheckForTwoCharForMap(int countInputStringLength, Map<String, WordFrequency>
outputList
			, StringBuilder stringBuilder, Map<String, String> wordsForest, StringBuilder[] prefixWord
			, Pos_C_XCDX_P posUtils){
		String countWordNode= stringBuilder.toString();
		if (!wordsForest.containsKey(countWordNode)){
			WordForestUtil.wordsForestNotContainsKey(outputList, countWordNode, prefixWord);
			return --countInputStringLength;
		}
		if (prefixWord[StablePOS.INT_ZERO].length()== StablePOS.INT_ZERO){
			WordForestUtil.prefixWordEqualZero(outputList, countWordNode, prefixWord);
			return countInputStringLength;
		}
		String[] strings= new String[StablePOS.INT_TWO];
		strings[StablePOS.INT_ZERO]= String.valueOf(countWordNode.charAt(StablePOS.INT_ZERO));
		strings[StablePOS.INT_ONE]= String.valueOf(countWordNode.charAt(StablePOS.INT_ZERO))
				+ String.valueOf(countWordNode.charAt(StablePOS.INT_ONE));
		if (wordsForest.containsKey(strings[StablePOS.INT_ZERO])){
			if (wordsForest.get(strings[StablePOS.INT_ZERO]).contains(StablePOS.NLP_CI_MING)){
				countInputStringLength= posUtils.chuLiMingCiOfTwoForMap(wordsForest, outputList,
countInputStringLength
						, strings, prefixWord);
				return countInputStringLength;
			}
		}
		if (wordsForest.containsKey(strings[StablePOS.INT_ONE])){
			WordForestUtil.wordsForestContainsKey(outputList, countWordNode, prefixWord);
			return countInputStringLength;
		}
		return StablePOS.INT_ZERO;
	}

	public int doPOSAndEMMCheckOfThreeForMap(int countInputLength, Map<String, WordFrequency> outputList
			, Map<String, String> wordsForest, StringBuilder stringBuilder, StringBuilder[] prefixWord
			, Pos_C_XCDX_P posUtils){
		String inputString= stringBuilder.toString();
		if (wordsForest.containsKey(inputString)){
			WordForestUtil.wordsForestContainsKey(outputList, inputString, prefixWord);
			return countInputLength;
		}
		String[] strings= new String[StablePOS.INT_FOUR];
		strings[StablePOS.INT_ZERO]= String.valueOf(inputString.charAt(StablePOS.INT_ZERO));
		strings[StablePOS.INT_ONE]= String.valueOf(inputString.charAt(StablePOS.INT_ZERO))
				+ inputString.charAt(StablePOS.INT_ONE);
		strings[StablePOS.INT_TWO]= String.valueOf(inputString.charAt(StablePOS.INT_ONE)
```

```
                                + inputString.charAt(StablePOS.INT_TWO));
                strings[StablePOS.INT_THREE]= String.valueOf(inputString.charAt(StablePOS.INT_TWO));
                if (null== prefixWord[StablePOS.INT_ZERO]){
                        if (wordsForest.containsKey(inputString)){
                                WordForestUtil.wordsForestContainsKey(outputList, inputString, prefixWord);
                                return countInputLength;
                        }
                        StringBuilder stringsBuilder= new StringBuilder();
                        countInputLength= doSlangPartAndPOSCheckForTwoCharForMap(--countInputLength, outputList
                                        , stringsBuilder.append(strings[StablePOS.INT_ONE]), wordsForest, prefixWord,
posUtils);
                        return countInputLength;
                }
                if (!wordsForest.containsKey(strings[StablePOS.INT_ZERO])){
                        StringBuilder stringsBuilder= new StringBuilder();
                        countInputLength= doSlangPartAndPOSCheckForTwoCharForMap(--countInputLength, outputList
                                        , stringsBuilder.append(strings[StablePOS.INT_ONE]), wordsForest, prefixWord,
posUtils);
                        return countInputLength;
                }
                if (StableMaps.zhuCi.containsKey(strings[StablePOS.INT_ZERO])){
                        countInputLength= posUtils.chuLiZhuCiOfThreeForMap(wordsForest, outputList, countInputLength
                                        , strings, prefixWord);
                        return countInputLength;
                }
                if (StableMaps.liangCi.containsKey(strings[StablePOS.INT_ZERO])){
                        countInputLength= posUtils.chuLiLiangCiOfThreeForMap(wordsForest, outputList, countInputLength
                                        , strings, prefixWord);
                        return countInputLength;
                }
                if (StableMaps.zhuCi.containsKey(strings[StablePOS.INT_ZERO])){
                        countInputLength= posUtils.chuLiJieCiOfThreeForMap(wordsForest, outputList, countInputLength
                                        , strings, prefixWord);
                        return countInputLength;
                }
                if (StableMaps.lianCi.containsKey(strings[StablePOS.INT_ZERO])){
                        countInputLength= posUtils.chuLiLianCiOfThreeForMap(wordsForest, outputList, countInputLength
                                        , strings, prefixWord);
                        return countInputLength;
                }
                StringBuilder stringsBuilder= new StringBuilder();
                countInputLength= doSlangPartAndPOSCheckForTwoCharForMap(--countInputLength, outputList
                                , stringsBuilder.append(strings[StablePOS.INT_ONE]), wordsForest, prefixWord, posUtils);
                return countInputLength;
        }

        public int doSlangCheckForMap(int countInputStringLength, Map<String, WordFrequency> output, StringBuilder
stringBuilder
                        , Map<String, String> wordsForest, StringBuilder[] prefixWord, Pos_C_XCDX_P posUtils){
                String inputString= stringBuilder.toString();
                if (wordsForest.containsKey(inputString)){
                        WordForestUtil.wordsForestContainsKey(output, inputString, prefixWord);
                        return countInputStringLength;
                }
                if(StableMaps.mingCi.containsKey(StablePOS.EMPTY_STRING+
inputString.charAt(StablePOS.INT_ZERO)+ inputString.charAt(StablePOS.INT_ONE))) {
                        if(StableMaps.mingCi.containsKey(StablePOS.EMPTY_STRING+
inputString.charAt(StablePOS.INT_TWO)+ inputString.charAt(StablePOS.INT_THREE))) {
                                WordForestUtil.wordsForestContainsKey(output, StablePOS.EMPTY_STRING+
inputString.charAt(StablePOS.INT_ZERO)+ inputString.charAt(StablePOS.INT_ONE), prefixWord);
                                return countInputStringLength;
                        }
```

```
                }
                countInputStringLength= doPOSAndEMMCheckOfThreeForMap(--countInputStringLength, output,
wordsForest
                                , stringBuilder.delete(StablePOS.INT_THREE, StablePOS.INT_FOUR), prefixWord,
posUtils);
                return countInputStringLength;
        }
}
```

```
package OEI.ME.pos.E;

import java.util.List;
import java.util.Map;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;
import OCI.ME.pos.C.POS_C;

public class POS_C_Cognition_E implements POS_C{

        @Override
        public int chuLiBaDongCiOfTwo(Map<String, String> wordsForest, List<String> outputList, int
countInputStringLength,
                        String[] strings, StringBuilder[] prefixWord) {
                // TODO Auto-generated method stub
                return 0;
        }

        @Override
        public int chuLiMingCiOfTwo(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength,
                        String[] strings, StringBuilder[] fixWord, int charPosition, String inputString) {
                // TODO Auto-generated method stub
                return 0;
        }

        @Override
        public void I_FixWordsOfTwo(int charPosition, String inputString, StringBuilder[] fixWords) {
                // TODO Auto-generated method stub

        }

        @Override
        public int parserFirstCharOfTwo(int countInputStringLength, List<String> outputList, String[] strings,
                        StringBuilder[] fixWord) {
                // TODO Auto-generated method stub
                return 0;
        }

        @Override
        public int chuLiLianCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength,
                        String[] strings, StringBuilder[] fixWord) {
                // TODO Auto-generated method stub
                return 0;
        }

        @Override
        public int loopCheckBackFix(StringBuilder[] fixWord, int backPosition, Map<String, String> wordsForest,
                        int countInputStringLength, List<String> outputList, String[] strings, int[] nestCountInputStringLength)
{
```

```
		// TODO Auto-generated method stub
		return 0;
	}

	@Override
	public void didNotFindFirstChar(List<String> outputList, String[] strings, StringBuilder[] fixWord,
			Map<String, String> wordsForest) {
		// TODO Auto-generated method stub

	}

	@Override
	public int parserFirstCharOfThree(int countInputStringLength, List<String> outputList, String[] strings,
			StringBuilder[] fixWord) {









		// TODO Auto-generated method stub
		return 0;
	}

	@Override
	public int parserFirstTwoCharOfThree(int countInputStringLength, List<String> outputList, String[] strings,
			StringBuilder[] fixWord) {
		// TODO Auto-generated method stub
		return 0;
	}

	@Override
	public int chuLiZhuCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength,
			String[] strings, StringBuilder[] fixWord) {
		// TODO Auto-generated method stub
		return 0;
	}

	@Override
	public int chuLiJieCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength,
			String[] strings, StringBuilder[] fixWord) {
		// TODO Auto-generated method stub
		return 0;
	}

	@Override
	public int chuLiLiangCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength,
			String[] strings, StringBuilder[] fixWord) {
		// TODO Auto-generated method stub
		return 0;
	}

	@Override
	public int chuLiMingCiOfTwoForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList,
			int countInputStringLength, String[] strings, StringBuilder[] fixWord) {
		// TODO Auto-generated method stub
		return 0;
	}

	@Override
	public int parserFirstCharOfTwoForMap(int countInputStringLength, Map<String, WordFrequency> outputList,
			String[] strings, StringBuilder[] fixWord, Map<String, String> wordsForest) {
		// TODO Auto-generated method stub
```

```
        return 0;
    }

    @Override
    public int chuLiLiangCiOfThreeForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList,
            int countInputStringLength, String[] strings, StringBuilder[] fixWord) {
        // TODO Auto-generated method stub
        return 0;
    }

    @Override
    public int chuLiJieCiOfThreeForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList,
            int countInputStringLength, String[] strings, StringBuilder[] fixWord) {
        // TODO Auto-generated method stub
        return 0;
    }

    @Override
    public int chuLiLianCiOfThreeForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList,
            int countInputStringLength, String[] strings, StringBuilder[] fixWord) {




        // TODO Auto-generated method stub
        return 0;
    }

    @Override
    public int loopCheckBackFixForMap(StringBuilder[] fixWord, int backPosition, Map<String, String> wordsForest,
            int countInputStringLength, Map<String, WordFrequency> outputList, String[] strings,
            int[] nestCountInputStringLength) {
        // TODO Auto-generated method stub
        return 0;
    }

    @Override
    public int chuLiZhuCiOfThreeForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList,
            int countInputStringLength, String[] strings, StringBuilder[] fixWord) {
        // TODO Auto-generated method stub
        return 0;
    }

    @Override
    public void didNotFindFirstCharForMap(Map<String, WordFrequency> outputList, String[] strings,
            StringBuilder[] fixWord, Map<String, String> wordsForest) {
        // TODO Auto-generated method stub

    }

    @Override
    public int parserFirstCharOfThreeForMap(int countInputStringLength, Map<String, WordFrequency> outputList,
            String[] strings, StringBuilder[] fixWord, Map<String, String> wordsForest) {
        // TODO Auto-generated method stub
        return 0;
    }

    @Override
    public int chuLiMingCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength,
            String[] strings, StringBuilder[] fixWord) {
        // TODO Auto-generated method stub
        return 0;
    }
```

```
	@Override
	public int chuLiShiTaiCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputLength,
			String[] strings, StringBuilder[] prefixWord) {
		// TODO Auto-generated method stub
		return 0;
	}

	@Override
	public int chuLiFuCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputLength,
			String[] strings, StringBuilder[] prefixWord) {
		// TODO Auto-generated method stub
		return 0;
	}

	@Override
	public int chuLiLianCiPostFixOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputLength,
			String[] strings, StringBuilder[] prefixWord) {
		// TODO Auto-generated method stub
		return 0;
	}

}
```

```
package OEI.ME.pos.E;

import java.util.List;
import java.util.Map;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;
import SVQ.stable.StablePOS;
import SVQ.stable.StableMaps;
import OCI.ME.pos.C.Pos_C_XCDX_E;

public class Pos_CE_XCDX_E extends Pos_CE_XCDX_O implements Pos_C_XCDX_E{

	public void I_FixWordsOfTwo(int charPosition, String inputString, StringBuilder[] fixWords) {
		fixWords[StablePOS.INT_ONE].delete(StablePOS.INT_ZERO, fixWords[StablePOS.INT_ONE].length());
		if (charPosition+ StablePOS.INT_SEVEN < inputString.length()) {
			fixWords[StablePOS.INT_ONE].append(inputString.substring(charPosition + StablePOS.INT_TWO
					, charPosition + StablePOS.INT_SEVEN));
			return;
		}
		fixWords[StablePOS.INT_ONE].append(inputString.substring(charPosition + StablePOS.INT_TWO
				, inputString.length()));
	}

	public int loopCheckBackFix(StringBuilder[] fixWord, int backPosition, Map<String, String> wordsForest
			, int countInputStringLength, List<String> outputList, String[] strings, int[]
nestCountInputStringLength){
		String charPositionAtFixWord = StablePOS.EMPTY_STRING +
fixWord[StablePOS.INT_ONE].charAt(backPosition);
		if (wordsForest.containsKey(charPositionAtFixWord) &&
(StableMaps.zhuCi.containsKey(charPositionAtFixWord)
				|| StableMaps.shengLueCi.containsKey(charPositionAtFixWord)||
```

```
StableMaps.fuCi.containsKey(charPositionAtFixWord))){

        if(!wordsForest.get(fixWord[StablePOS.INT_ZERO].toString()).contains(StablePOS.NLP_CI_SHENG_LUE)
                                        && wordsForest.get(charPositionAtFixWord).contains(StablePOS.NLP_CI_FU)){
                                return StablePOS.INT_ERROR;
                        }
                        nestCountInputStringLength[StablePOS.INT_ZERO]=
parserFirstCharOfThree(countInputStringLength, outputList
                                        , strings, fixWord);
                        return StablePOS.INT_RIGHT;
                }
                return StablePOS.INT_ERROR;
        }

        public void didNotFindFirstChar(List<String> outputList, String[] strings, StringBuilder[] fixWord
                        , Map<String, String> wordsForest){
                if(!wordsForest.containsKey(strings[StablePOS.INT_TWO])){
                        if(wordsForest.containsKey(strings[StablePOS.INT_ONE])){
                                outputList.add(strings[StablePOS.INT_ONE]);
                                outputList.add(strings[StablePOS.INT_THREE]);
                                fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWord[StablePOS.INT_ZERO].length());
                                fixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_THREE]);
                        }
                        return;
                }
                if (StableMaps.fuCi.containsKey(strings[StablePOS.INT_TWO])){
                        outputList.add(strings[StablePOS.INT_ZERO]);
                        outputList.add(strings[StablePOS.INT_TWO]);
                        fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWord[StablePOS.INT_ZERO].length());
                        fixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_TWO]);
                        return;
                }
                if(StableMaps.CiTwo.containsKey(strings[StablePOS.INT_ONE])){









                        outputList.add(strings[StablePOS.INT_ONE]);
                        outputList.add(strings[StablePOS.INT_THREE]);
                        fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWord[StablePOS.INT_ZERO].length());
                        fixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_THREE]);
                }
        }

        public int loopCheckBackFixForMap(StringBuilder[] fixWord, int backPosition, Map<String, String> wordsForest
                        , int countInputStringLength, Map<String, WordFrequency> outputList, String[] strings, int[]
nestCountInputStringLength){
                String charPositionAtFixWord= StablePOS.EMPTY_STRING+
fixWord[StablePOS.INT_ONE].charAt(backPosition);
                if (wordsForest.containsKey(charPositionAtFixWord)&&
(StableMaps.zhuCi.containsKey(charPositionAtFixWord)
                                        || wordsForest.get(charPositionAtFixWord).contains(StablePOS.NLP_CI_SHENG_LUE))){
                        nestCountInputStringLength[StablePOS.INT_ZERO]=
parserFirstCharOfThreeForMap(countInputStringLength, outputList
                                        , strings, fixWord, wordsForest);
                        return StablePOS.INT_RIGHT;
                }
                return StablePOS.INT_ERROR;
```

```
	}
	public void didNotFindFirstCharForMap(Map<String, WordFrequency> outputList, String[] strings, StringBuilder[]
fixWord
			, Map<String, String> wordsForest){
		if(!wordsForest.containsKey(strings[StablePOS.INT_TWO])){
			return;
		}
		if (StableMaps.fuCi.containsKey(strings[StablePOS.INT_TWO])){
			if (outputList.containsKey(strings[StablePOS.INT_ZERO])){
				WordFrequency wordFrequency= outputList.get(strings[StablePOS.INT_ZERO]);
				wordFrequency.I_Frequency(wordFrequency.getFrequency()+ StablePOS.INT_ONE);
				outputList.put(strings[StablePOS.INT_ZERO], wordFrequency);
			} else{
				WordFrequency wordFrequency= new WordFrequency();
				wordFrequency.I_Frequency(StablePOS.INT_ONE);
				wordFrequency.I_Word(strings[StablePOS.INT_ZERO]);
				outputList.put(strings[StablePOS.INT_ZERO], wordFrequency);
			}
			if (outputList.containsKey(strings[StablePOS.INT_TWO])){
				WordFrequency wordFrequency= outputList.get(strings[StablePOS.INT_TWO]);
				wordFrequency.I_Frequency(wordFrequency.getFrequency()+ StablePOS.INT_ONE);
				outputList.put(strings[StablePOS.INT_TWO], wordFrequency);
			} else{
				WordFrequency wordFrequency= new WordFrequency();
				wordFrequency.I_Frequency(StablePOS.INT_ONE);
				wordFrequency.I_Word(strings[StablePOS.INT_TWO]);
				outputList.put(strings[StablePOS.INT_TWO], wordFrequency);
			}
			fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWord[StablePOS.INT_ZERO].length());
			fixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_TWO]);
		}
	}
}
```

```
package OEI.ME.pos.E;

import java.util.List;
import java.util.Map;




import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;
import SVQ.stable.StablePOS;
import OCI.ME.pos.C.Pos_C_XCDX_O;

///
/// O  o
/// E  e
/// P  p
public class Pos_CE_XCDX_O implements Pos_C_XCDX_O{

	public int parserFirstCharOfTwo(int countInputStringLength, List<String> outputList, String[] strings
			, StringBuilder[] fixWord){
		outputList.add(strings[StablePOS.INT_ZERO]);
		String postNext=StablePOS.EMPTY_STRING + strings[StablePOS.INT_ONE].charAt(StablePOS.INT_ONE);
		outputList.add(postNext);
		fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO, fixWord[StablePOS.INT_ZERO].length());
```

```
                fixWord[StablePOS.INT_ZERO].append(postNext);
                return countInputStringLength;
        }

        public int parserFirstCharOfThree(int countInputStringLength, List<String> outputList, String[] strings
                        , StringBuilder[] fixWord){
                outputList.add(strings[StablePOS.INT_ZERO]);
                outputList.add(strings[StablePOS.INT_TWO]);
                fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO, fixWord[StablePOS.INT_ZERO].length());
                fixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_TWO]);
                return countInputStringLength;
        }

        public int parserFirstTwoCharOfThree(int countInputStringLength, List<String> outputList, String[] strings
                        , StringBuilder[] fixWord){
                outputList.add(strings[StablePOS.INT_ONE]);
                outputList.add(strings[StablePOS.INT_THREE]);
                fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO, fixWord[StablePOS.INT_ZERO].length());
                fixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_THREE]);
                return countInputStringLength;
        }

        public int parserFirstCharOfTwoForMap(int countInputStringLength, Map<String, WordFrequency> outputList, String[]
strings
                        , StringBuilder[] fixWord, Map<String, String> wordsForest){
                countInputStringLength -= StablePOS.INT_TWO;
                if (outputList.containsKey(strings[StablePOS.INT_ZERO])){
                        WordFrequency wordFrequency = outputList.get(strings[StablePOS.INT_ZERO]);
                        wordFrequency.I_Frequency(wordFrequency.getFrequency() + StablePOS.INT_ONE);
                        outputList.put(strings[StablePOS.INT_ZERO], wordFrequency);
                } else{
                        WordFrequency wordFrequency = new WordFrequency();
                        wordFrequency.I_Frequency(StablePOS.INT_ONE);
                        wordFrequency.I_Word(strings[StablePOS.INT_ZERO]);
                        outputList.put(strings[StablePOS.INT_ZERO], wordFrequency);
                }
                fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO, fixWord[StablePOS.INT_ZERO].length());
                fixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_ZERO]);
                countInputStringLength += StablePOS.INT_ONE;
                return countInputStringLength;
        }

        public int parserFirstCharOfThreeForMap(int countInputStringLength, Map<String, WordFrequency> outputList, String[]
strings
                        , StringBuilder[] fixWord, Map<String, String> wordsForest){
                countInputStringLength-= StablePOS.INT_THREE;
                if (outputList.containsKey(strings[StablePOS.INT_ZERO])){
                        WordFrequency wordFrequency = outputList.get(strings[StablePOS.INT_ZERO]);
                        wordFrequency.I_Frequency(wordFrequency.getFrequency()+ StablePOS.INT_ONE);
                        outputList.put(strings[StablePOS.INT_ZERO], wordFrequency);




                } else{
                        WordFrequency wordFrequency= new WordFrequency();
                        wordFrequency.I_Frequency(StablePOS.INT_ONE);
                        wordFrequency.I_Word(strings[StablePOS.INT_ZERO]);
                        outputList.put(strings[StablePOS.INT_ZERO], wordFrequency);
                }
                fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO, fixWord[StablePOS.INT_ZERO].length());
                fixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_ZERO]);
```

```
                countInputStringLength ++;
                if (wordsForest.containsKey(strings[StablePOS.INT_TWO])){
                        if (outputList.containsKey(strings[StablePOS.INT_TWO])){
                                WordFrequency wordFrequency= outputList.get(strings[StablePOS.INT_TWO]);
                                wordFrequency.I_Frequency(wordFrequency.getFrequency() + StablePOS.INT_ONE);
                                outputList.put(strings[StablePOS.INT_TWO], wordFrequency);
                        } else{
                                WordFrequency wordFrequency= new WordFrequency();
                                wordFrequency.I_Frequency(StablePOS.INT_ONE);
                                wordFrequency.I_Word(strings[StablePOS.INT_TWO]);
                                outputList.put(strings[StablePOS.INT_TWO], wordFrequency);
                        }
                        fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWord[StablePOS.INT_ZERO].length());
                        fixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_TWO]);
                        countInputStringLength+= StablePOS.INT_TWO;
                        return countInputStringLength;
                }
                return countInputStringLength;
        }
}
```

```
package OEI.ME.pos.E;

import java.util.List;
import java.util.Map;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;
import OCI.ME.pos.C.Pos_C_XCDX_P;
import SVQ.stable.StablePOS;
import SVQ.stable.StableMaps;

public class Pos_CE_XCDX_P extends Pos_CE_XCDX_E implements Pos_C_XCDX_P{
        public int chuLiBaDongCiOfTwo(Map<String, String> wordsForest, List<String> outputList, int
countInputStringLength,
                        String[] strings, StringBuilder[] prefixWord){
                if (!wordsForest.containsKey(prefixWord[StablePOS.INT_ZERO].toString())){
                        return countInputStringLength;
                }
                if (StableMaps.daiCi.containsKey(prefixWord[StablePOS.INT_ZERO].toString())
                                ||StableMaps.fuCi.containsKey(prefixWord[StablePOS.INT_ZERO].toString())) {
                        countInputStringLength = parserFirstCharOfTwo(countInputStringLength, outputList, strings,
prefixWord);
                        return countInputStringLength;
                }
                if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT_ONE])){
                        outputList.add(strings[StablePOS.INT_ONE]);
                        prefixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
prefixWord[StablePOS.INT_ZERO].length());
                        prefixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_ONE]);
                        return countInputStringLength;
                }
                return countInputStringLength- StablePOS.INT_TWO;
        }

        public int chuLiMingCiOfTwo(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength
                        , String[] strings, StringBuilder[] fixWord, int charPosition, String inputString){
                if (wordsForest.containsKey(fixWord[StablePOS.INT_ZERO].toString())){
```

```
                        if (StableMaps.liangCi.containsKey(fixWord[StablePOS.INT_ZERO].toString())){
                                countInputStringLength= parserFirstCharOfTwo(countInputStringLength, outputList, strings,
fixWord);
                                return countInputStringLength;
                        }
                        if (StableMaps.dongCi.containsKey(fixWord[StablePOS.INT_ZERO].toString())
                                        ||StableMaps.xingRongCi.containsKey(fixWord[StablePOS.INT_ZERO].toString())
                                        ||StableMaps.mingCi.containsKey(fixWord[StablePOS.INT_ZERO].toString())
                                        ||StableMaps.zhuCi.containsKey(fixWord[StablePOS.INT_ZERO].toString())
                                        ||StableMaps.liangCi.containsKey(fixWord[StablePOS.INT_ZERO].toString())){
                                if(StableMaps.CiTwo.containsKey(strings[StablePOS.INT_ONE])) {
                                        fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWord[StablePOS.INT_ZERO].length());
                                        fixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_ONE]);
                                        outputList.add(strings[StablePOS.INT_ONE]);
                                        return countInputStringLength;
                                }
                                countInputStringLength= parserFirstCharOfTwo(countInputStringLength, outputList, strings,
fixWord);
                                return countInputStringLength;
                        }
                        I_FixWordsOfTwo(charPosition, inputString, fixWord);
                        if (StablePOS.INT_ZERO< fixWord[StablePOS.INT_ONE].length()&&
StableMaps.fuCi.containsKey(StablePOS.EMPTY_STRING
                                        + fixWord[StablePOS.INT_ONE].toString().charAt(StablePOS.INT_ZERO))){
                                countInputStringLength= parserFirstCharOfTwo(countInputStringLength, outputList, strings,
fixWord);
                                return countInputStringLength;
                        }
                        if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT_ONE])){
                                outputList.add(strings[StablePOS.INT_ONE]);
                                fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWord[StablePOS.INT_ZERO].length());
                                fixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_ONE]);
                                return countInputStringLength;
                        }
                        countInputStringLength= parserFirstCharOfTwo(countInputStringLength, outputList, strings, fixWord);
                        return countInputStringLength;
                }
                return countInputStringLength;
        }

        public int chuLiLianCiPostFixOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputLength,
                        String[] strings, StringBuilder[] prefixWord) {
                if (StableMaps.lianCi.containsKey(strings[StablePOS.INT_TWO])){
                        countInputLength= parserFirstCharOfThree(countInputLength, outputList, strings, prefixWord);
                        return countInputLength;
                }
                if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT_ONE])){
                        countInputLength= parserFirstTwoCharOfThree(countInputLength, outputList, strings, prefixWord);
                        return countInputLength;
                }
                return countInputLength;
        }

        public int chuLiLianCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength
                        , String[] strings, StringBuilder[] fixWord){
                if (outputList.size() == StablePOS.INT_ZERO){
                        didNotFindFirstChar(outputList, strings, fixWord, wordsForest);
                        return countInputStringLength;
                }
                if (wordsForest.containsKey(fixWord[StablePOS.INT_ZERO].toString()) &&
```

```
(StableMaps.mingCi.containsKey(fixWord[StablePOS.INT_ZERO].toString())
                                || StableMaps.dongCi.containsKey(fixWord[StablePOS.INT_ZERO].toString())||
StableMaps.fuCi.containsKey(fixWord[StablePOS.INT_ZERO].toString())
                                || StableMaps.daiCi.containsKey(fixWord[StablePOS.INT_ZERO].toString()) ||
StableMaps.weiCi.containsKey(fixWord[StablePOS.INT_ZERO].toString()))){



                        countInputStringLength = parserFirstCharOfThree(countInputStringLength, outputList, strings,
fixWord);
                        return countInputStringLength;
                }
                if (wordsForest.containsKey(fixWord[StablePOS.INT_ZERO].toString()) &&
(StableMaps.zhuCi.containsKey(fixWord[StablePOS.INT_ZERO].toString())
                                || StableMaps.shengLueCi.containsKey(fixWord[StablePOS.INT_ZERO].toString()))){
                        for (int BackPosition = StablePOS.INT_ZERO; BackPosition <
fixWord[StablePOS.INT_ONE].length(); BackPosition++){
                                int[] nestCountInputStringLength = new int[StablePOS.INT_ONE];
                                int result = loopCheckBackFix(fixWord, BackPosition, wordsForest, countInputStringLength,
outputList, strings
                                                , nestCountInputStringLength);
                                if (result == StablePOS.INT_RIGHT){
                                        return nestCountInputStringLength[StablePOS.INT_ZERO];
                                }
                        }
                        if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT_ONE])){
                                outputList.add(strings[StablePOS.INT_ONE]);
                                fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWord[StablePOS.INT_ZERO].length());
                                fixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_ONE]);
                                return countInputStringLength- StablePOS.INT_ONE;
                        }
                        return countInputStringLength- StablePOS.INT_THREE;
                }
                if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT_ONE])){
                        outputList.add(strings[StablePOS.INT_ONE]);
                        fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWord[StablePOS.INT_ZERO].length());
                        return countInputStringLength- StablePOS.INT_ONE;
                }
                return countInputStringLength- StablePOS.INT_THREE;
        }

        public int chuLiZhuCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength
                        , String[] strings, StringBuilder[] fixWord){
                if (StablePOS.INT_ZERO== outputList.size()){
                        didNotFindFirstChar(outputList, strings, fixWord, wordsForest);
                        return countInputStringLength;
                }
                if (wordsForest.containsKey(fixWord[StablePOS.INT_ZERO].toString())){
                        if (StableMaps.dongCi.containsKey(fixWord[StablePOS.INT_ZERO].toString())){
                                if(wordsForest.containsKey(strings[StablePOS.INT_TWO])) {
                                        countInputStringLength = parserFirstCharOfThree(countInputStringLength,
outputList, strings, fixWord);
                                        return countInputStringLength;
                                }
                                outputList.add(strings[StablePOS.INT_ZERO]);
                                fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWord[StablePOS.INT_ZERO].length());
                                fixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_ZERO]);
                                return    countInputStringLength= StablePOS.INT_ONE;
```

```
                        } else if(fixWord[StablePOS.INT_ONE].length()> StablePOS.INT_ONE) {
                                String firstChar= StablePOS.EMPTY_STRING+
fixWord[StablePOS.INT_ONE].toString().charAt(StablePOS.INT_ZERO);
                                String secondChar= StablePOS.EMPTY_STRING+
fixWord[StablePOS.INT_ONE].toString().charAt(StablePOS.INT_ONE);
                                if(!StableMaps.fuCi.containsKey(firstChar)&& !StableMaps.fuCi.containsKey(secondChar)
                                                &&!StableMaps.fuCi.containsKey(firstChar+ secondChar)) {
                                        if(wordsForest.containsKey(firstChar)&& wordsForest.containsKey(secondChar)) {
                                                outputList.add(strings[StablePOS.INT_ZERO]);
                                                fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWord[StablePOS.INT_ZERO].length());
                                                fixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_ZERO]);
                                                countInputStringLength= StablePOS.INT_ONE;




                                                if(wordsForest.containsKey(strings[StablePOS.INT_TWO])) {
                                                        outputList.add(strings[StablePOS.INT_TWO]);
                                                        fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWord[StablePOS.INT_ZERO].length());

        fixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_TWO]);
                                                                countInputStringLength= StablePOS.INT_THREE;
                                                }
                                                return countInputStringLength;
                                        }
                                }
                        }
                        if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT_ONE])){
                                outputList.add(strings[StablePOS.INT_ONE]);
                                fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWord[StablePOS.INT_ZERO].length());
                                fixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_ONE]);
                                return countInputStringLength- StablePOS.INT_ONE;
                        }
                        if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT_TWO])){
                                outputList.add(strings[StablePOS.INT_ZERO]);
                                outputList.add(strings[StablePOS.INT_TWO]);
                                fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWord[StablePOS.INT_ZERO].length());
                                fixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_TWO]);
                                return countInputStringLength;
                        }
                }
                return countInputStringLength;
        }

        public int chuLiJieCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength
                        , String[] strings, StringBuilder[] fixWord){
                if (StablePOS.INT_ZERO== outputList.size()&& (wordsForest.get(strings[StablePOS.INT_TWO])
                                .contains(StablePOS.NLP_CI_WEI))){
                        outputList.add(strings[StablePOS.INT_ZERO]);
                        outputList.add(strings[StablePOS.INT_TWO]);
                        fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWord[StablePOS.INT_ZERO].length());
                        fixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_TWO]);
                        return countInputStringLength;
                }
                if (outputList.size() > StablePOS.INT_ZERO&&
wordsForest.containsKey(fixWord[StablePOS.INT_ZERO].toString())){
                        if (StableMaps.qingTaiCi.containsKey(fixWord[StablePOS.INT_ZERO].toString())
```

```
                                        || StableMaps.weiCi.containsKey(fixWord[StablePOS.INT_ZERO].toString())
                                        || StableMaps.lianCi.containsKey(fixWord[StablePOS.INT_ZERO].toString())){
                                countInputStringLength= parserFirstCharOfThree(countInputStringLength, outputList, strings,
fixWord);
                                return countInputStringLength;
                        } else{
                                if(StableMaps.dongCi.containsKey(fixWord[StablePOS.INT_ZERO].toString())) {
                                        if(StableMaps.xingWeiCi.containsKey(strings[StablePOS.INT_ONE])
                                                                        ||
StableMaps.xingRongCi.containsKey(strings[StablePOS.INT_ONE])) {
                                                countInputStringLength=
parserFirstTwoCharOfThree(countInputStringLength, outputList, strings, fixWord);
                                                return countInputStringLength;
                                        }
                                }
                                if (StableMaps.mingCi.containsKey(strings[StablePOS.INT_TWO])){
                                        outputList.add(strings[StablePOS.INT_ZERO]);
                                        outputList.add(strings[StablePOS.INT_TWO]);
                                        fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWord[StablePOS.INT_ZERO].length());
                                        fixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_TWO]);
                                        return countInputStringLength;
                                }else if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT_ONE])){
                                        if(StableMaps.jieCi.containsKey(strings[StablePOS.INT_ONE])) {




                                                countInputStringLength=
parserFirstTwoCharOfThree(countInputStringLength, outputList, strings, fixWord);
                                                return countInputStringLength;
                                        }
                                        outputList.add(strings[StablePOS.INT_ONE]);
                                        fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWord[StablePOS.INT_ZERO].length());
                                        fixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_ONE]);
                                        countInputStringLength = StablePOS.INT_TWO;
                                        return countInputStringLength;
                                }else if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT_TWO])){
                                        outputList.add(strings[StablePOS.INT_ZERO]);
                                        outputList.add(strings[StablePOS.INT_TWO]);
                                        fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWord[StablePOS.INT_ZERO].length());
                                        fixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_TWO]);
                                        return countInputStringLength;
                                }
                        }
                }
                if(StableMaps.jieCi.containsKey(strings[StablePOS.INT_ONE])) {
                        countInputStringLength= parserFirstTwoCharOfThree(countInputStringLength, outputList, strings,
fixWord);
                        return countInputStringLength;
                }
                outputList.add(strings[StablePOS.INT_ZERO]);
                if(StableMaps.CiTwo.containsKey(strings[StablePOS.INT_TWO])) {
                        outputList.add(strings[StablePOS.INT_TWO]);
                        fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWord[StablePOS.INT_ZERO].length());
                        fixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_TWO]);
                        return countInputStringLength;
                }
                fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO, fixWord[StablePOS.INT_ZERO].length());
```

```
                fixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_ZERO]);
                return countInputStringLength= StablePOS.INT_ONE;
        }

        public int chuLiLiangCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength
                        , String[] strings, StringBuilder[] fixWord){
                if (wordsForest.containsKey(fixWord[StablePOS.INT_ZERO].toString())){
                        if (StableMaps.mingCi.containsKey(fixWord[StablePOS.INT_ZERO].toString())||
StableMaps.daiCi.containsKey(fixWord[StablePOS.INT_ZERO].toString())){
                                countInputStringLength = parserFirstCharOfThree(countInputStringLength, outputList,
strings, fixWord);
                                return countInputStringLength;
                        }
                        if (StableMaps.liangCi.containsKey(strings[StablePOS.INT_ONE])){
                                outputList.add(strings[StablePOS.INT_ONE]);
                                fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWord[StablePOS.INT_ZERO].length());
                                fixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_ONE]);
                                return StablePOS.INT_TWO;
                        }
                        if
((StableMaps.xingWeiCi.containsKey(fixWord[StablePOS.INT_ZERO].toString())||StableMaps.xingRongCi.containsKey(fixWor
d[StablePOS.INT_ZERO].toString()))
                                        && StableMaps.mingCi.containsKey(strings[StablePOS.INT_ONE])){
                                outputList.add(strings[StablePOS.INT_ONE]);
                                fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWord[StablePOS.INT_ZERO].length());
                                fixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_ONE]);
                                return StablePOS.INT_TWO;
                        }
                }
                fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO, fixWord[StablePOS.INT_ZERO].length());
                fixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_ZERO]);




                outputList.add(strings[StablePOS.INT_ZERO]);
                if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT_TWO])){
                        outputList.add(strings[StablePOS.INT_TWO]);
                        fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWord[StablePOS.INT_ZERO].length());
                        fixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_TWO]);
                        return StablePOS.INT_THREE;
                }
                return  StablePOS.INT_ONE;
        }

        public int chuLiMingCiOfTwoForMap(Map<String, String> wordsForest,  Map<String, WordFrequency> outputList, int
countInputStringLength
                        , String[] strings, StringBuilder[] fixWord){
                if (wordsForest.containsKey(fixWord[StablePOS.INT_ZERO].toString())){
                        if (StableMaps.liangCi.containsKey(fixWord[StablePOS.INT_ZERO].toString())){
                                countInputStringLength = parserFirstCharOfTwoForMap(countInputStringLength, outputList,
strings, fixWord
                                                , wordsForest);
                                return countInputStringLength;
                        }
                        countInputStringLength -= StablePOS.INT_TWO;
                        if (wordsForest.containsKey(strings[StablePOS.INT_ONE])){
                                if (outputList.containsKey(strings[StablePOS.INT_ONE])){
                                        WordFrequency wordFrequency = outputList.get(strings[StablePOS.INT_ONE]);
```

```
                                        wordFrequency.I_Frequency(wordFrequency.getFrequency() +
StablePOS.INT_ONE);
                                        outputList.put(strings[StablePOS.INT_ONE], wordFrequency);
                                } else{
                                        WordFrequency wordFrequency = new WordFrequency();
                                        wordFrequency.I_Frequency(StablePOS.INT_ONE);
                                        wordFrequency.I_Word(strings[StablePOS.INT_ONE]);
                                        outputList.put(strings[StablePOS.INT_ONE], wordFrequency);
                                }
                                fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWord[StablePOS.INT_ZERO].length());
                                fixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_ONE]);
                                countInputStringLength += StablePOS.INT_TWO;
                        }
                        return countInputStringLength;
                }
                return countInputStringLength;
        }

        public int chuLiLiangCiOfThreeForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList,
int countInputStringLength
                        , String[] strings, StringBuilder[] fixWord){
                if (wordsForest.containsKey(fixWord[StablePOS.INT_ZERO].toString())){
                        if (StableMaps.mingCi.containsKey(fixWord[StablePOS.INT_ZERO].toString())||
StableMaps.daiCi.containsKey(fixWord[StablePOS.INT_ZERO].toString())){
                                countInputStringLength= parserFirstCharOfThreeForMap(countInputStringLength, outputList,
strings, fixWord
                                                , wordsForest);
                                return countInputStringLength;
                        }
                        countInputStringLength -= StablePOS.INT_THREE;
                        if (wordsForest.containsKey(strings[StablePOS.INT_ONE])){
                                if (outputList.containsKey(strings[StablePOS.INT_ONE])){
                                        WordFrequency wordFrequency = outputList.get(strings[StablePOS.INT_ONE]);
                                        wordFrequency.I_Frequency(wordFrequency.getFrequency() +
StablePOS.INT_ONE);
                                        outputList.put(strings[StablePOS.INT_ONE], wordFrequency);
                                } else{
                                        WordFrequency wordFrequency = new WordFrequency();
                                        wordFrequency.I_Frequency(StablePOS.INT_ONE);
                                        wordFrequency.I_Word(strings[StablePOS.INT_ONE]);
                                        outputList.put(strings[StablePOS.INT_ONE], wordFrequency);



                                }
                                fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWord[StablePOS.INT_ZERO].length());
                                fixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_ONE]);
                                countInputStringLength += StablePOS.INT_TWO;
                        }
                        return countInputStringLength;
                }
                return countInputStringLength;
        }

        public int chuLiJieCiOfThreeForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList, int
countInputStringLength
                        , String[] strings, StringBuilder[] fixWord){
                if (outputList.size()== StablePOS.INT_ZERO&&
StableMaps.weiCi.containsKey(strings[StablePOS.INT_TWO])){
                        if (outputList.containsKey(strings[StablePOS.INT_ZERO])){
```

```
                                WordFrequency wordFrequency= outputList.get(strings[StablePOS.INT_ZERO]);
                                wordFrequency.I_Frequency(wordFrequency.getFrequency()+ StablePOS.INT_ONE);
                                outputList.put(strings[StablePOS.INT_ZERO], wordFrequency);
                        } else{
                                WordFrequency wordFrequency= new WordFrequency();
                                wordFrequency.I_Frequency(StablePOS.INT_ONE);
                                wordFrequency.I_Word(strings[StablePOS.INT_ZERO]);
                                outputList.put(strings[StablePOS.INT_ZERO], wordFrequency);
                        }
                        if (outputList.containsKey(strings[StablePOS.INT_TWO])){
                                WordFrequency wordFrequency= outputList.get(strings[StablePOS.INT_TWO]);
                                wordFrequency.I_Frequency(wordFrequency.getFrequency()+ StablePOS.INT_ONE);
                                outputList.put(strings[StablePOS.INT_TWO], wordFrequency);
                        } else{
                                WordFrequency wordFrequency= new WordFrequency();
                                wordFrequency.I_Frequency(StablePOS.INT_ONE);
                                wordFrequency.I_Word(strings[StablePOS.INT_TWO]);
                                outputList.put(strings[StablePOS.INT_TWO], wordFrequency);
                        }
                        fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWord[StablePOS.INT_ZERO].length());
                        fixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_TWO]);
                        return countInputStringLength;
                }
                if (outputList.size()> StablePOS.INT_ZERO&&
wordsForest.containsKey(fixWord[StablePOS.INT_ZERO].toString())){
                        if (StableMaps.lianCi.containsKey(fixWord[StablePOS.INT_ZERO].toString())||
StableMaps.qingTaiCi.containsKey(fixWord[StablePOS.INT_ZERO].toString())
                                        || StableMaps.weiCi.containsKey(fixWord[StablePOS.INT_ZERO].toString())){
                                countInputStringLength= parserFirstCharOfThreeForMap(countInputStringLength, outputList,
strings, fixWord, wordsForest);
                                return countInputStringLength;
                        } else{
                                countInputStringLength-= StablePOS.INT_THREE;
                                if (wordsForest.containsKey(strings[StablePOS.INT_ONE])){
                                        if (outputList.containsKey(strings[StablePOS.INT_ONE])){
                                                WordFrequency wordFrequency=
outputList.get(strings[StablePOS.INT_ONE]);
                                                wordFrequency.I_Frequency(wordFrequency.getFrequency()+
StablePOS.INT_ONE);
                                                outputList.put(strings[StablePOS.INT_ONE], wordFrequency);
                                        } else{
                                                WordFrequency wordFrequency= new WordFrequency();
                                                wordFrequency.I_Frequency(StablePOS.INT_ONE);
                                                wordFrequency.I_Word(strings[StablePOS.INT_ONE]);
                                                outputList.put(strings[StablePOS.INT_ONE], wordFrequency);
                                        }
                                        fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWord[StablePOS.INT_ZERO].length());
                                        fixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_ONE]);
                                        countInputStringLength+= StablePOS.INT_TWO;
                                }
                                return countInputStringLength;



                        }
                }
                return countInputStringLength;
        }

        public int chuLiLianCiOfThreeForMap(Map<String, String> wordsForest, Map<String, WordFrequency>  outputList, int
```

```
countInputStringLength
                        , String[] strings, StringBuilder[] fixWord){
                if (outputList.size()== StablePOS.INT_ZERO){
                        didNotFindFirstCharForMap(outputList, strings, fixWord, wordsForest);
                        return countInputStringLength;
                }
                if (wordsForest.containsKey(fixWord[StablePOS.INT_ZERO].toString())
                                && (StableMaps.mingCi.containsKey(fixWord[StablePOS.INT_ZERO].toString())
                                                ||
StableMaps.daiCi.containsKey(fixWord[StablePOS.INT_ZERO].toString())
                                                ||
StableMaps.weiCi.containsKey(fixWord[StablePOS.INT_ZERO].toString())
                                                ||
StableMaps.dongCi.containsKey(fixWord[StablePOS.INT_ZERO].toString())
                                                ||
StableMaps.fuCi.containsKey(fixWord[StablePOS.INT_ZERO].toString()))){
                        countInputStringLength = parserFirstCharOfThreeForMap(countInputStringLength, outputList, strings,
fixWord, wordsForest);
                        return countInputStringLength;
                }
                if (wordsForest.containsKey(fixWord[StablePOS.INT_ZERO].toString())
                                && (StableMaps.zhuCi.containsKey(fixWord[StablePOS.INT_ZERO].toString())
                                                ||
StableMaps.shengLueCi.containsKey(fixWord[StablePOS.INT_ZERO].toString()))){
                        for (int BackPosition= StablePOS.INT_ZERO; BackPosition< fixWord[StablePOS.INT_ONE].length();
BackPosition++){
                                int[] nestCountInputStringLength = new int[StablePOS.INT_ONE];
                                int result= loopCheckBackFixForMap(fixWord, BackPosition, wordsForest,
countInputStringLength, outputList, strings
                                                , nestCountInputStringLength);
                                if (result== StablePOS.INT_RIGHT){
                                        return nestCountInputStringLength[StablePOS.INT_ZERO];
                                }
                        }
                        countInputStringLength-= StablePOS.INT_THREE;
                        if (wordsForest.containsKey(strings[StablePOS.INT_ONE])){
                                if (outputList.containsKey(strings[StablePOS.INT_ONE])){
                                        WordFrequency wordFrequency= outputList.get(strings[StablePOS.INT_ONE]);
                                        wordFrequency.I_Frequency(wordFrequency.getFrequency() +
StablePOS.INT_ONE);
                                        outputList.put(strings[StablePOS.INT_ONE], wordFrequency);
                                } else{
                                        WordFrequency wordFrequency = new WordFrequency();
                                        wordFrequency.I_Frequency(StablePOS.INT_ONE);
                                        wordFrequency.I_Word(strings[StablePOS.INT_ONE]);
                                        outputList.put(strings[StablePOS.INT_ONE], wordFrequency);
                                }
                                fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWord[StablePOS.INT_ZERO].length());
                                fixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_ONE]);
                                countInputStringLength += StablePOS.INT_TWO;
                        }
                        return countInputStringLength;
                }
                countInputStringLength-= StablePOS.INT_THREE;
                if (wordsForest.containsKey(strings[StablePOS.INT_ONE])){
                        if (outputList.containsKey(strings[StablePOS.INT_ONE])){
                                WordFrequency wordFrequency = outputList.get(strings[StablePOS.INT_ONE]);
                                wordFrequency.I_Frequency(wordFrequency.getFrequency()+ StablePOS.INT_ONE);
                                outputList.put(strings[StablePOS.INT_ONE], wordFrequency);
                        } else{
                                WordFrequency wordFrequency= new WordFrequency();
                                wordFrequency.I_Frequency(StablePOS.INT_ONE);
```

```
                                wordFrequency.I_Word(strings[StablePOS.INT_ONE]);
                                outputList.put(strings[StablePOS.INT_ONE], wordFrequency);




                        }
                        fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWord[StablePOS.INT_ZERO].length());
                        countInputStringLength+= StablePOS.INT_TWO;
                }
                return countInputStringLength;
        }

        public int chuLiZhuCiOfThreeForMap(Map<String, String> wordsForest, Map<String, WordFrequency>  outputList, int
countInputStringLength
                        , String[] strings, StringBuilder[] fixWord){
                if (StablePOS.INT_ZERO== outputList.size()){
                        didNotFindFirstCharForMap(outputList, strings, fixWord, wordsForest);
                        return countInputStringLength;
                }
                if (wordsForest.containsKey(fixWord[StablePOS.INT_ZERO].toString())){
                        if (StableMaps.dongCi.containsKey(fixWord[StablePOS.INT_ZERO].toString())){
                                countInputStringLength= parserFirstCharOfThreeForMap(countInputStringLength, outputList,
strings, fixWord, wordsForest);
                                return countInputStringLength;
                        } else{
                                countInputStringLength-= StablePOS.INT_THREE;
                                if (wordsForest.containsKey(strings[StablePOS.INT_ONE])){
                                        if (outputList.containsKey(strings[StablePOS.INT_ONE])){
                                                WordFrequency wordFrequency=
outputList.get(strings[StablePOS.INT_ONE]);
                                                wordFrequency.I_Frequency(wordFrequency.getFrequency()+
StablePOS.INT_ONE);
                                                outputList.put(strings[StablePOS.INT_ONE], wordFrequency);
                                        } else{
                                                WordFrequency wordFrequency= new WordFrequency();
                                                wordFrequency.I_Frequency(StablePOS.INT_ONE);
                                                wordFrequency.I_Word(strings[StablePOS.INT_ONE]);
                                                outputList.put(strings[StablePOS.INT_ONE], wordFrequency);
                                        }
                                        fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWord[StablePOS.INT_ZERO].length());
                                        fixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_ONE]);
                                        countInputStringLength+= StablePOS.INT_TWO;
                                }
                                return countInputStringLength;
                        }
                }
                return countInputStringLength;
        }

        public int chuLiMingCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength,
                        String[] strings, StringBuilder[] fixWord){
                if (StableMaps.xingWeiCi.containsKey(fixWord[StablePOS.INT_ZERO].toString())|| StableMaps.mingCi
                                .containsKey(fixWord[StablePOS.INT_ZERO].toString())){
                        if(StableMaps.dongCi.containsKey(strings[StablePOS.INT_TWO])){
                                countInputStringLength= parserFirstCharOfThree(countInputStringLength, outputList, strings,
fixWord);
                                return countInputStringLength;
                        }
                        if(StableMaps.dongCi.containsKey(strings[StablePOS.INT_THREE])){
```

```
                                if(StableMaps.fuCi.containsKey(StablePOS.EMPTY_STRING+ (0==
fixWord[StablePOS.INT_ONE].length()? "@^_^@": fixWord[StablePOS.INT_ONE].charAt(StablePOS.INT_ZERO)))){
                                        if(StableMaps.dongCi.containsKey(strings[StablePOS.INT_ONE])||
StableMaps.qingTaiCi.containsKey(strings[StablePOS.INT_ONE])) {
                                                countInputStringLength=
parserFirstTwoCharOfThree(countInputStringLength, outputList, strings, fixWord);
                                                return countInputStringLength;
                                        }
                                        outputList.add(strings[StablePOS.INT_ZERO]);
                                        if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT_TWO])){
                                                outputList.add(strings[StablePOS.INT_TWO]);
                                                fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWord[StablePOS.INT_ZERO].length());
                                                fixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_TWO]);
                                                return countInputStringLength;
                                        }
                                        fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWord[StablePOS.INT_ZERO].length());
                                        fixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_ZERO]);
                                        return countInputStringLength= StablePOS.INT_ONE;
                                }
                                if(!StableMaps.dingMingCi.containsKey(strings[StablePOS.INT_ZERO])){
                                        if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT_ONE])){
                                                countInputStringLength=
parserFirstTwoCharOfThree(countInputStringLength, outputList, strings, fixWord);
                                                return countInputStringLength;
                                        }
                                        outputList.add(strings[StablePOS.INT_ZERO]);
                                        fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWord[StablePOS.INT_ZERO].length());
                                        fixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_ZERO]);
                                        return countInputStringLength= StablePOS.INT_ONE;
                                }
                        }
                        if(StableMaps.mingCi.containsKey(strings[StablePOS.INT_TWO])){
                                if(StablePOS.INT_ZERO< fixWord[StablePOS.INT_ONE].length()&&
StableMaps.zhuCi.containsKey(StablePOS.EMPTY_STRING
                                        + fixWord[StablePOS.INT_ONE].charAt(StablePOS.INT_ZERO))){
                                        if(StableMaps.CiTwo.containsKey(strings[StablePOS.INT_ONE])) {

        if(!StableMaps.dongCi.containsKey(fixWord[StablePOS.INT_ZERO].toString())){
                                                        countInputStringLength=
parserFirstCharOfThree(countInputStringLength, outputList, strings, fixWord);
                                                        return countInputStringLength;
                                                }
                                                countInputStringLength=
parserFirstTwoCharOfThree(countInputStringLength, outputList, strings, fixWord);
                                                return countInputStringLength;
                                        }
                                        outputList.add(strings[StablePOS.INT_ZERO]);
                                        fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWord[StablePOS.INT_ZERO].length());
                                        fixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_ZERO]);
                                        return countInputStringLength= StablePOS.INT_ONE;
                                }
                                countInputStringLength= parserFirstCharOfThree(countInputStringLength, outputList, strings,
fixWord);
                                return countInputStringLength;
```

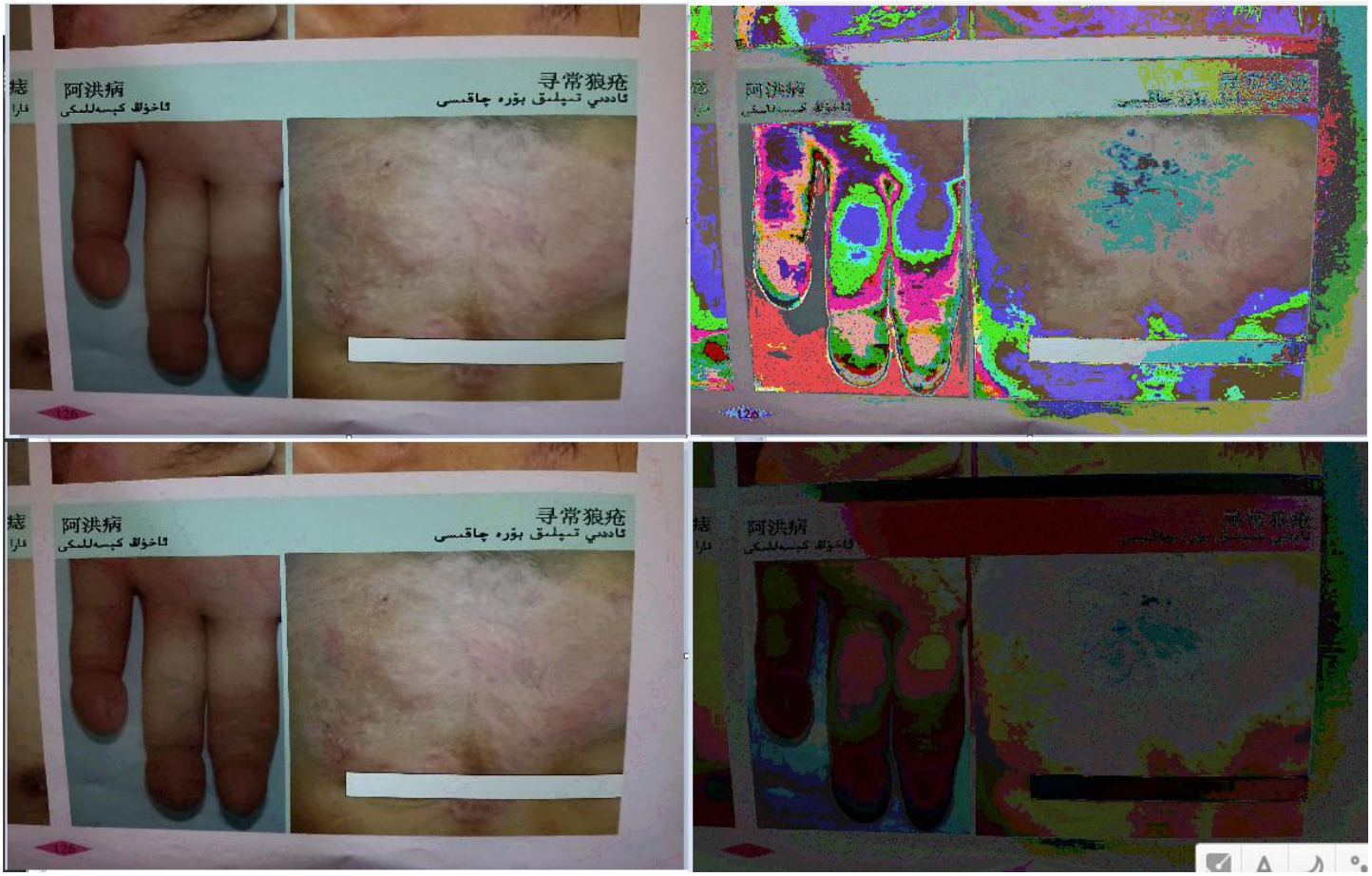
```
                        }
                        if(StableMaps.mingCi.containsKey(strings[StablePOS.INT_ONE])||
StableMaps.fuCi.containsKey(strings[StablePOS.INT_ONE])){
                                countInputStringLength= parserFirstTwoCharOfThree(countInputStringLength, outputList,
strings, fixWord);
                                return countInputStringLength;
                        }
                        if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT_TWO])){
                                countInputStringLength= parserFirstCharOfThree(countInputStringLength, outputList, strings,
fixWord);
                                return countInputStringLength;
                        }
                        outputList.add(strings[StablePOS.INT_ZERO]);
                        fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWord[StablePOS.INT_ZERO].length());
                        fixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_ZERO]);
                        countInputStringLength= StablePOS.INT_ONE;
                        return countInputStringLength;
                }
                if(StableMaps.dongCi.containsKey(strings[StablePOS.INT_THREE])){
                        if(StableMaps.dongCi.containsKey(strings[StablePOS.INT_TWO])){
                                if(StableMaps.mingCi.containsKey(strings[StablePOS.INT_ZERO])) {



                                        countInputStringLength= parserFirstCharOfThree(countInputStringLength,
outputList, strings, fixWord);
                                        return countInputStringLength;
                                }
                        }
                        if(null!= fixWord[StablePOS.INT_ZERO]&&
StablePOS.INT_ZERO<fixWord[StablePOS.INT_ZERO].length()){
                                if(StableMaps.zhuCi.containsKey(StablePOS.EMPTY_STRING+
fixWord[StablePOS.INT_ZERO].charAt(StablePOS.INT_ZERO))){
                                        if(!StableMaps.mingCi.containsKey(strings[StablePOS.INT_ONE])) {
                                                countInputStringLength= parserFirstCharOfThree(countInputStringLength,
outputList, strings, fixWord);
                                                return countInputStringLength;
                                        }
                                }
                        }
                        if(StableMaps.CiTwo.containsKey(strings[StablePOS.INT_ONE])) {
                                countInputStringLength= parserFirstTwoCharOfThree(countInputStringLength, outputList,
strings, fixWord);
                                return countInputStringLength;
                        }
                        countInputStringLength= parserFirstCharOfThree(countInputStringLength, outputList, strings,
fixWord);
                        return countInputStringLength;
                }
                if(StableMaps.mingCi.containsKey(strings[StablePOS.INT_TWO])){
                        if(StablePOS.INT_ZERO< fixWord[StablePOS.INT_ONE].length()&&
StableMaps.zhuCi.containsKey(StablePOS.EMPTY_STRING
                                        + fixWord[StablePOS.INT_ONE].charAt(StablePOS.INT_ZERO))){
                                if(StableMaps.CiTwo.containsKey(strings[StablePOS.INT_ONE])){
                                        if(!StableMaps.dongCi.containsKey(fixWord[StablePOS.INT_ZERO].toString())){
                                                countInputStringLength= parserFirstCharOfThree(countInputStringLength,
outputList, strings, fixWord);
                                                return countInputStringLength;
                                        }
                                        countInputStringLength= parserFirstTwoCharOfThree(countInputStringLength,
outputList, strings, fixWord);
```

```
                                        return countInputStringLength;
                                }
                                countInputStringLength= parserFirstCharOfThree(countInputStringLength, outputList, strings,
fixWord);
                                return countInputStringLength;
                        }
                        if(StablePOS.INT_ZERO< fixWord[StablePOS.INT_ONE].length()&&
StableMaps.dingMingCi.containsKey(StablePOS.EMPTY_STRING
                                        + fixWord[StablePOS.INT_ONE].charAt(StablePOS.INT_ZERO))){
                                countInputStringLength= parserFirstTwoCharOfThree(countInputStringLength, outputList,
strings, fixWord);
                                return countInputStringLength;
                        }
                        if(StableMaps.dongCi.containsKey(fixWord[StablePOS.INT_ZERO].toString())){
                                if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT_ONE])){
                                        countInputStringLength= parserFirstTwoCharOfThree(countInputStringLength,
outputList, strings, fixWord);
                                        return countInputStringLength;
                                }
                        }
                        countInputStringLength= parserFirstCharOfThree(countInputStringLength, outputList, strings,
fixWord);
                        return countInputStringLength;
                }
                if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT_ONE])){
                        outputList.add(strings[StablePOS.INT_ONE]);
                        fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWord[StablePOS.INT_ZERO].length());
                        fixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_ONE]);
                        countInputStringLength= StablePOS.INT_TWO;
                        return countInputStringLength;



                }
                outputList.add(strings[StablePOS.INT_ZERO]);
                if(StableMaps.CiTwo.containsKey(strings[StablePOS.INT_TWO])) {
                        outputList.add(strings[StablePOS.INT_TWO]);
                        fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
fixWord[StablePOS.INT_ZERO].length());
                        fixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_TWO]);
                        return countInputStringLength;
                }
                fixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO, fixWord[StablePOS.INT_ZERO].length());
                fixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_ZERO]);
                return countInputStringLength= StablePOS.INT_ONE;
        }

        public int chuLiShiTaiCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputLength,
                        String[] strings, StringBuilder[] prefixWord) {
                if ((StableMaps.mingCi.containsKey(strings[StablePOS.INT_TWO].toString()))
                                && (StableMaps.jieCi.containsKey(prefixWord[StablePOS.INT_ZERO].toString())
                                                ||
StableMaps.xingWeiCi.containsKey(prefixWord[StablePOS.INT_ZERO].toString())
                                                ||
StableMaps.dongCi.containsKey(prefixWord[StablePOS.INT_ZERO].toString()))){
                        countInputLength= parserFirstCharOfThree(countInputLength, outputList, strings, prefixWord);
                        return countInputLength;
                }
                if (StableMaps.dongCi.containsKey(strings[StablePOS.INT_TWO].toString())
                                ||StableMaps.liangCi.containsKey(strings[StablePOS.INT_TWO].toString())) {
                        countInputLength= parserFirstCharOfThree(countInputLength, outputList, strings, prefixWord);
```

```
                        return countInputLength;
                }
                if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT_ONE])){
                        outputList.add(strings[StablePOS.INT_ONE]);
                        prefixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
prefixWord[StablePOS.INT_ZERO].length());
                        prefixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_ONE]);
                        countInputLength= StablePOS.INT_TWO;
                        return countInputLength;
                }
                outputList.add(strings[StablePOS.INT_ZERO]);
                if(StableMaps.CiTwo.containsKey(strings[StablePOS.INT_TWO])) {
                        outputList.add(strings[StablePOS.INT_TWO]);
                        prefixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
prefixWord[StablePOS.INT_ZERO].length());
                        prefixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_TWO]);
                        return countInputLength;
                }
                prefixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
prefixWord[StablePOS.INT_ZERO].length());
                prefixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_ZERO]);
                return countInputLength= StablePOS.INT_ONE;
        }

        public int chuLiFuCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputLength,
                        String[] strings, StringBuilder[] prefixWord) {
                if (StableMaps.fuCi.containsKey(strings[StablePOS.INT_TWO].toString())) {
                        if (StableMaps.fuCi.containsKey(prefixWord[StablePOS.INT_ZERO].toString())) {
                                countInputLength= parserFirstCharOfThree(countInputLength, outputList, strings,
prefixWord);
                                return countInputLength;
                        }
                }
                if (StableMaps.dongCi.containsKey(strings[StablePOS.INT_TWO].toString())) {
                        if (StableMaps.zhuCi.containsKey(StablePOS.EMPTY_STRING+
prefixWord[StablePOS.INT_ONE].charAt(StablePOS.INT_ZERO))
                                        || StableMaps.daiCi.containsKey(StablePOS.EMPTY_STRING+
prefixWord[StablePOS.INT_ONE].charAt(StablePOS.INT_ZERO))) {
                                countInputLength= parserFirstCharOfThree(countInputLength, outputList, strings,
prefixWord);
                                return countInputLength;
                        }
                }
                if(StableMaps.CiTwo.containsKey(strings[StablePOS.INT_ONE])) {
                        countInputLength= parserFirstTwoCharOfThree(countInputLength, outputList, strings, prefixWord);




                        return countInputLength;
                }
                outputList.add(strings[StablePOS.INT_ZERO]);
                prefixWord[StablePOS.INT_ZERO].delete(StablePOS.INT_ZERO,
prefixWord[StablePOS.INT_ZERO].length());
                prefixWord[StablePOS.INT_ZERO].append(strings[StablePOS.INT_ZERO]);
                return StablePOS.INT_ONE;
        }
}
```

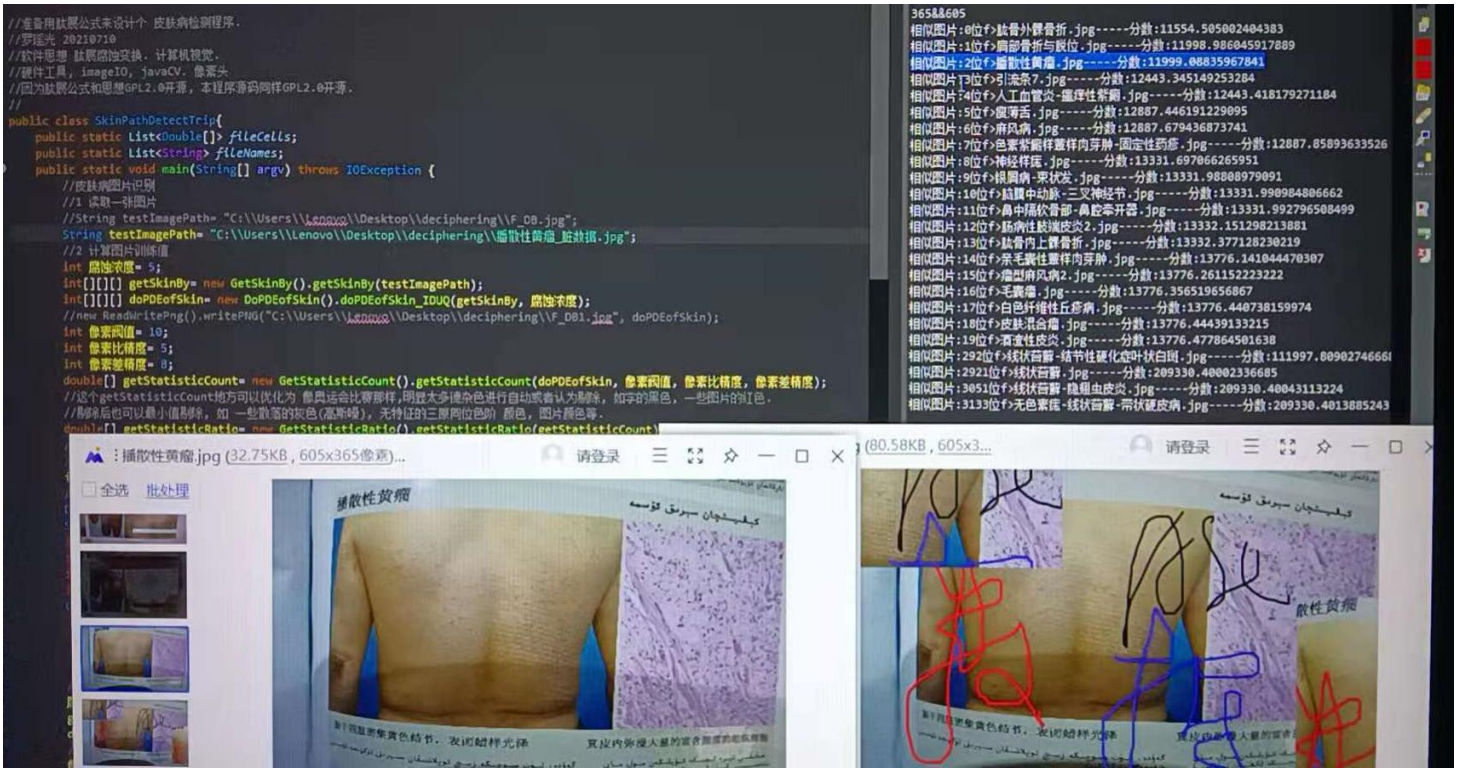
```
package OEI.ME.pos.E;

import OCI.ME.pos.C.Pos_C_XCDX;
import OCI.ME.pos.C.Pos_C_XCDX_E;
import OCI.ME.pos.C.Pos_C_XCDX_O;
import OCI.ME.pos.C.Pos_C_XCDX_P;

public class Pos_CE_XCDX implements Pos_C_XCDX{
	public Pos_C_XCDX_O pos_C_XCDX_O;
	public Pos_C_XCDX_P pos_C_XCDX_P;
	public Pos_C_XCDX_E pos_C_XCDX_E;

	public Pos_CE_XCDX() {
		pos_C_XCDX_O= new Pos_CE_XCDX_O();
		pos_C_XCDX_P= new Pos_CE_XCDX_P();
		pos_C_XCDX_E= new Pos_CE_XCDX_E();
	}
}
```

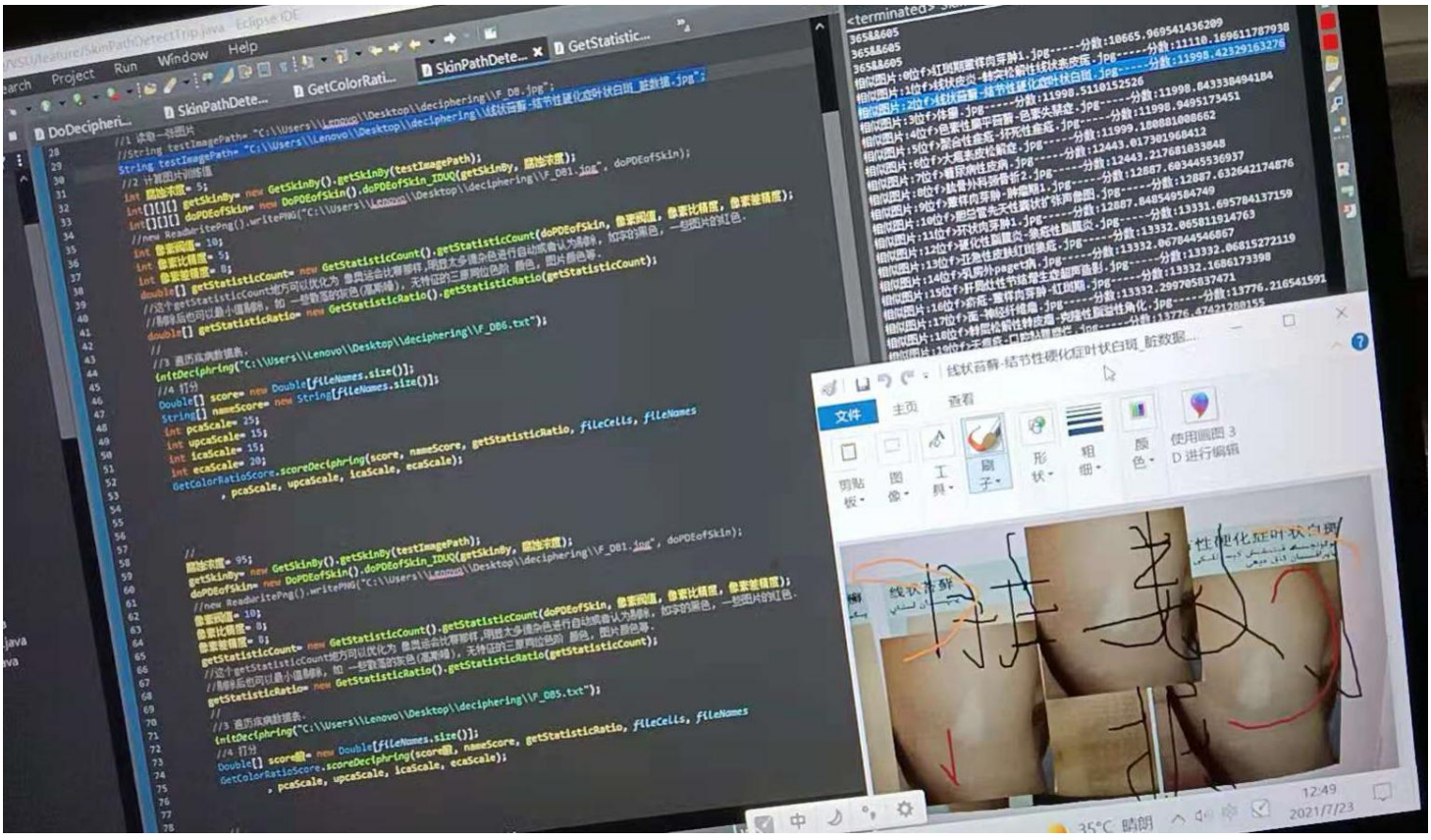
6 四进制，十七进制肽腐蚀算法对比



左上原图 右上 十七进制八元腐蚀
左下十七进制四元腐蚀 右下 四进制四元腐蚀

7 图片识别读脏能力展示



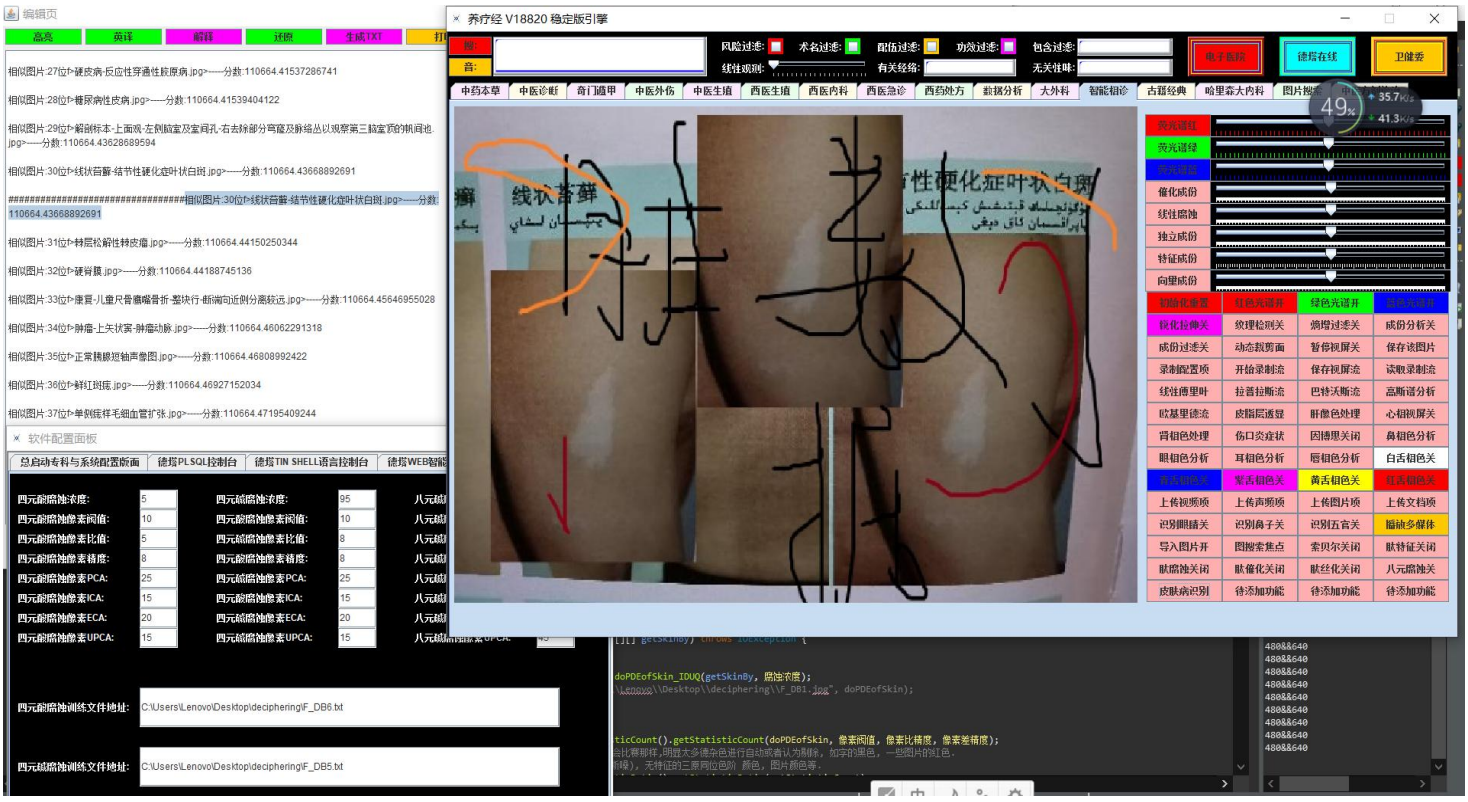


对应完整源码
```
package ISQ.VSU.feature;

import java.io.BufferedReader;

import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Map;

import ESU.array.Double_ESU;
import OEU.LYG4DQS4D.LYG9DWithDoubleTopSort4D;

//准备用肽展公式来设计个 皮肤病检测程序.
//罗瑶光 20210710
//软件思想 肽展腐蚀变换. 计算机视觉.
//硬件工具, imageIO, javaCV. 像素头
//因为肽展公式和思想GPL2.0开源, 本程序源码同样GPL2.0开源.
//
public class SkinPathDetectTrip{
	public static List<Double[]> fileCells;
	public static List<String> fileNames;
	public static void main(String[] argv) throws IOException {
		//皮肤病图片识别
		//1 读取一张图片
		//String testImagePath= "C:\\Users\\Lenovo\\Desktop\\deciphering\\F_DB.jpg";
		String testImagePath= "C:\\Users\\Lenovo\\Desktop\\deciphering\\播散性黄瘤_脏数据.jpg";
		//2 计算图片训练值
		int 腐蚀浓度= 5;
		int[][][] getSkinBy= new GetSkinBy().getSkinBy(testImagePath);
		int[][][] doPDEofSkin= new DoPDEofSkin().doPDEofSkin_IDUQ(getSkinBy, 腐蚀浓度);
		//new ReadWritePng().writePNG("C:\\Users\\Lenovo\\Desktop\\deciphering\\F_DB1.jpg",
doPDEofSkin);
		int 像素阀值= 10;
		int 像素比精度= 5;
		int 像素差精度= 8;
		double[] getStatisticCount= new GetStatisticCount().getStatisticCount(doPDEofSkin, 像素阀值, 像
素比精度, 像素差精度);
		//这个getStatisticCount地方可以优化为 像奥运会比赛那样,明显太多德杂色进行自动或者认为剔
除, 如字的黑色, 一些图片的红色.
		//剔除后也可以最小值剔除, 如 一些散落的灰色(高斯噪), 无特征的三原同位色阶 颜色, 图片颜
色等.
		double[] getStatisticRatio= new GetStatisticRatio().getStatisticRatio(getStatisticCount);
		//
		//3 遍历疾病数据表.
		initDeciphring("C:\\Users\\Lenovo\\Desktop\\deciphering\\F_DB6.txt");
```

```
//4 打分
                Double[] score= new Double[fileNames.size()];
                String[] nameScore= new String[fileNames.size()];
                int pcaScale= 25;
                int upcaScale= 15;
                int icaScale= 15;
                int ecaScale= 20;
                GetColorRatioScore.scoreDeciphring(score, nameScore, getStatisticRatio, fileCells, fileNames
                                , pcaScale, upcaScale, icaScale, ecaScale);



                //
                腐蚀浓度= 95;
                getSkinBy= new GetSkinBy().getSkinBy(testImagePath);
                doPDEofSkin= new DoPDEofSkin().doPDEofSkin_IDUQ(getSkinBy, 腐蚀浓度);
                //new ReadWritePng().writePNG("C:\\Users\\Lenovo\\Desktop\\deciphering\\F_DB1.jpg",
doPDEofSkin);
                像素阀值= 10;
                像素比精度= 8;
                像素差精度= 8;
                getStatisticCount= new GetStatisticCount().getStatisticCount(doPDEofSkin, 像素阀值, 像素比精度,
像素差精度);
                //这个getStatisticCount地方可以优化为 像奥运会比赛那样,明显太多德杂色进行自动或者认为剔
除, 如字的黑色, 一些图片的红色.
                //剔除后也可以最小值剔除, 如 一些散落的灰色(高斯噪), 无特征的三原同位色阶 颜色, 图片颜
色等.
                getStatisticRatio= new GetStatisticRatio().getStatisticRatio(getStatisticCount);
                //
                //3 遍历疾病数据表.
                initDeciphring("C:\\Users\\Lenovo\\Desktop\\deciphering\\F_DB5.txt");
                //4 打分
                Double[] score酸= new Double[fileNames.size()];
                GetColorRatioScore.scoreDeciphring(score酸, nameScore, getStatisticRatio, fileCells, fileNames
                                , pcaScale, upcaScale, icaScale, ecaScale);



                //
                //
                腐蚀浓度= 95;
                getSkinBy= new GetSkinBy().getSkinBy(testImagePath);
                doPDEofSkin= new DoPDEofSkin().doPDEofSkin_AOPM_VECS_IDUQ_TXH(getSkinBy, 腐蚀浓
度);
                //new ReadWritePng().writePNG("C:\\Users\\Lenovo\\Desktop\\deciphering\\F_DB_17.jpg",
doPDEofSkin);
                像素阀值= 10;
                像素比精度= 2;
                像素差精度= 8;
                getStatisticCount= new GetStatisticCount().getStatisticCount(doPDEofSkin, 像素阀值, 像素比精度,
像素差精度);
                //这个getStatisticCount地方可以优化为 像奥运会比赛那样,明显太多德杂色进行自动或者认为剔
除, 如字的黑色, 一些图片的红色.
```

```
//剔除后也可以最小值剔除, 如 一些散落的灰色(高斯噪), 无特征的三原同位色阶 颜色, 图片颜色等.
		getStatisticRatio= new GetStatisticRatio().getStatisticRatio(getStatisticCount);
		//
		//3 遍历疾病数据表.
		initDeciphring("C:\\Users\\Lenovo\\Desktop\\deciphering\\F_DB6_17.txt");



//4 打分
		Double[] score_17= new Double[fileNames.size()];
		pcaScale= 1;
		upcaScale= 45;
		icaScale= 1;
		ecaScale= 1;
		GetColorRatioScore.scoreDeciphring(score_17, nameScore, getStatisticRatio, fileCells, fileNames
				, pcaScale, upcaScale, icaScale, ecaScale);
		//融合
		for(int i= 0; i< score酸.length; i++) {
			score[i]+= score酸[i]+ score_17[i];
		}
		//5 筛选
		double[] scoreDouble= new double[score.length];
		for(int i= 0; i< scoreDouble.length; i++) {
			scoreDouble[i]= score[i];//
		}
		//改成map
		Map<Double, Map<String, Boolean>> map= Double_ESU.getMapFromDoubleStringArray(score,
nameScore);
		int 递归深度= 70; //避免同值冗余内存高峰
		int 堆栈广度= 7; //避免堆栈浪费计算高峰
		new LYG9DWithDoubleTopSort4D().sort(scoreDouble, 堆栈广度, 递归深度);
		//new Quick9DLYGWithString_ESU().sort(scoreInt, nameScore);

		//6 推荐
		Here:
		for(int i= 0; i< scoreDouble.length; i++) {
			if(!map.containsKey(scoreDouble[i])) {
				i++;
				continue Here;
			}
			Iterator<String> iterator= map.get(scoreDouble[i]).keySet().iterator();
			while(iterator.hasNext()) {
				String string= iterator.next();
				if(string.contains("线状苔藓")) {
					System.out.println("相似图片:"+ i+ "位"+ string+ "-----分数:"+
scoreDouble[i]);
				}else{
					if(i< 20) {
						System.out.println("相似图片:"+ i+ "位"+ string+ "-----分数:"+
scoreDouble[i]);
					}
				}
			}
			map.remove(scoreDouble[i]);
		//	System.out.println("相似图片:"+ nameScore[i]+ "-----分数:"+ scoreInt[i]);
```

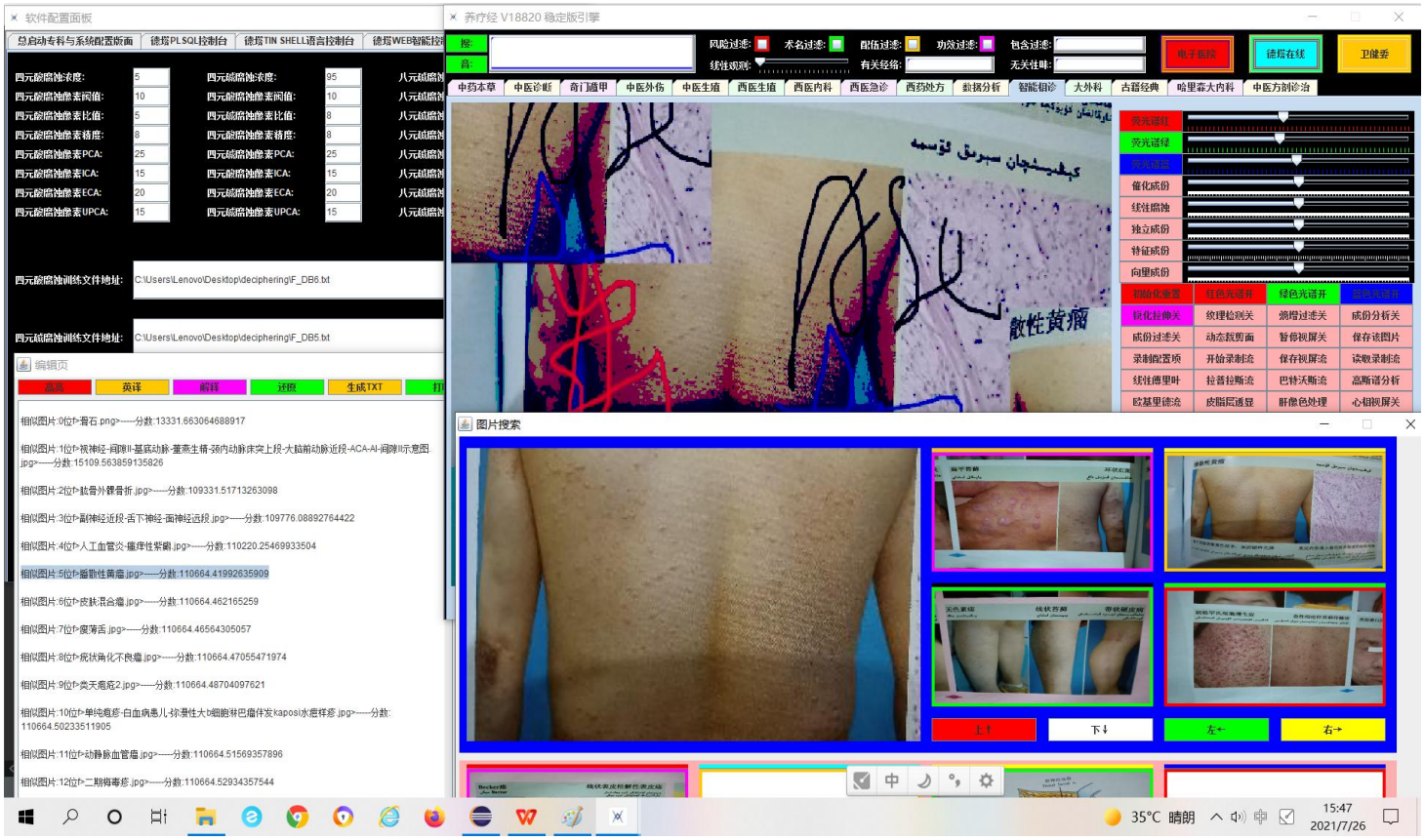
```
                }
        }


        private static void initDeciphring(String string) throws NumberFormatException, IOException {
                fileNames= new ArrayList<>();
                fileCells= new ArrayList<>();
                String ctempString;
                InputStream inputStream= new FileInputStream(string);
                BufferedReader cReader= new BufferedReader(new InputStreamReader(inputStream, "GBK"));
                while ((ctempString= cReader.readLine())!= null) {
                        String[] lineCell= ctempString.split(">d>");
                        if(lineCell.length> 1) {
                                fileNames.add(lineCell[0]);
                                Double[] lineDouble= new Double[lineCell.length- 1];
                                for(int i= 0; i< lineDouble.length; i++) {
                                        lineDouble[i]= Double.valueOf(lineCell[i+ 1]);
                                }
                                fileCells.add(lineDouble);
                        }
                }
                cReader.close();
        }
}
```



多诊断页综合作业- 线性苔藓极脏数据识别。

示例图片太多 适当保留些 多诊断页综合作业- 散播黄瘤 极脏数据识别

结论，3个训练数据文档 做成身份证头像那样格式化采样，精度可以翻100~10000倍。目前是手机随便拍照的采样，连书的框边都在图片里。读脏能力如图 都特别强悍。 不多解释了。



多诊断页综合作业- 散播黄瘤 极脏数据识别

```
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;
    }
    //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);
                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 {
                        OIQ[0][k]= IDUQ.charAt(k);
                    }
                }
                ps[i][j]= new InitonsPDS().DO_ACP_IDV_17(OIQ, 17);
            }
        }
        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 injectPDI(String pdi) {

        return pdi;
}
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;
}
//      //阿拉伯数字变元基数字
//      decadeToPDS.initonsSet.put("0", "0");
//      decadeToPDS.initonsSet.put("1", "1");
//      decadeToPDS.initonsSet.put("2", "2");
```

```
//      decadeToPDS.initonsSet.put("3", "3");
//      decadeToPDS.initonsSet.put("4", "4");
//      decadeToPDS.initonsSet.put("5", "5");
//      decadeToPDS.initonsSet.put("6", "6");
//      decadeToPDS.initonsSet.put("7", "7");
//      decadeToPDS.initonsSet.put("8", "8");
//      decadeToPDS.initonsSet.put("9", "9");
//      decadeToPDS.initonsSet.put("10", "A");
//      decadeToPDS.initonsSet.put("11", "B");
//      decadeToPDS.initonsSet.put("12", "C");
//      decadeToPDS.initonsSet.put("13", "D");
//      decadeToPDS.initonsSet.put("14", "E");
//      decadeToPDS.initonsSet.put("15", "F");
//      decadeToPDS.initonsSet.put("16", "G");
//
//      //元基数字变阿拉伯数字
//      decadeToPDS.numberSet.put("0", 0);
//      decadeToPDS.numberSet.put("1", 1);
//      decadeToPDS.numberSet.put("2", 2);
//      decadeToPDS.numberSet.put("3", 3);
//      decadeToPDS.numberSet.put("4", 4);
//      decadeToPDS.numberSet.put("5", 5);
//      decadeToPDS.numberSet.put("6", 6);
//      decadeToPDS.numberSet.put("7", 7);
//      decadeToPDS.numberSet.put("8", 8);
//      decadeToPDS.numberSet.put("9", 9);
//      decadeToPDS.numberSet.put("A", 10);
//      decadeToPDS.numberSet.put("B", 11);
//      decadeToPDS.numberSet.put("C", 12);
//      decadeToPDS.numberSet.put("D", 13);
//      decadeToPDS.numberSet.put("E", 14);
//      decadeToPDS.numberSet.put("F", 15);
//      decadeToPDS.numberSet.put("G", 16);
//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");
```

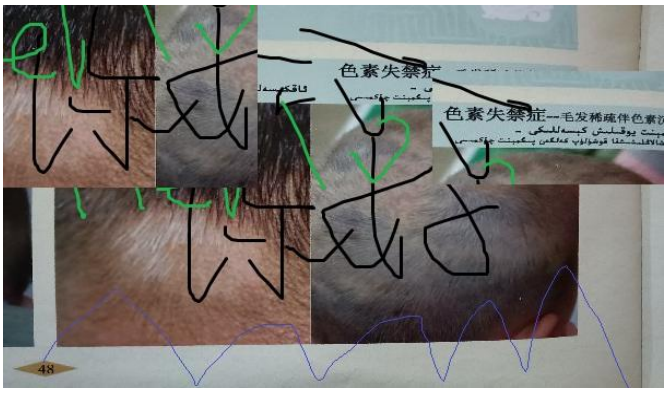
```
        initonsSet.put("10", "A");
        initonsSet.put("11", "B");
        initonsSet.put("12", "C");
        initonsSet.put("13", "D");
        initonsSet.put("14", "E");
        initonsSet.put("15", "F");
        initonsSet.put("16", "G");

        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_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_M_SVQ_PDS_OEU(String SQA) {
        String[] PDS= new String[]{"AOPM", "VECS", "DIUQ", "HTX"};
        String ISQ_PSD= "";
```

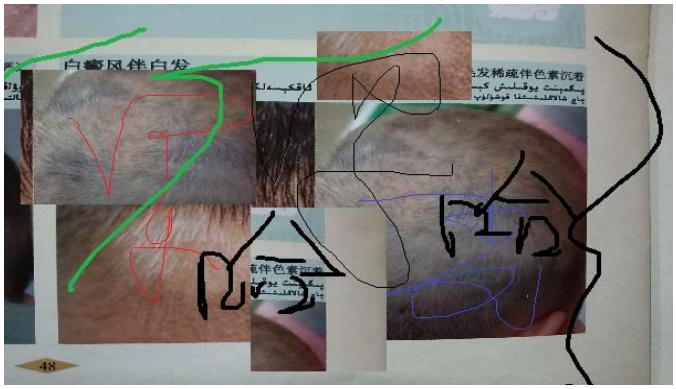
```
        while(injectPDI(ISQ_PSD).length()> 0) {
            ISQ_PSD+= PDSEncode(SQA);
        }
        return ISQ_PSD;
    }
    public String ESU_ECS_SVQ_PDS_OEU(String SQA) {
        String[] PDS= new String[]{"AOPM", "VECS"};
        String ISQ_PSD= "";
        while(injectPDI(ISQ_PSD).length()> 0) {
            ISQ_PSD+= PDSEncode(SQA);
        }
        return ISQ_PSD;
    }
    public String ESU_P_SVQ_PDS_OEU(String SQA) {
        String[] PDS= new String[]{"AOPM", "VECS", "DIUQ", "HTX"};
        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;
    }
    public String EUP_QD_PQI(String AEP, int QSV) {
        String[][] CED= new String[][] {{"A","O","P","M"}, {"V","E","C","S"}, {"D","I","U","Q"}};
        String PM_ISQ= "";
        return AEP;
    }
    public String EUP_QD_PQU(String AEP, int QSU) {
        String[][] CED= new String[][] {{"D","I","U","Q"}, {"H","T","X"}};
        String PM_ISQ= "";
        return AEP;
    }
    public String ESU_M_SVQ_PDS_OEU_M(String SQA) {
        String[] PDS= new String[]{"AOPM"};
        String ISQ_PSD= "";
        while(injectPDI(ISQ_PSD).length()> 0) {
            ISQ_PSD+= PDSEncode(SQA);
```

```
        }
        return ISQ_PSD;
    }
    public String ESU_M_SVQ_PDS_OEU_P(String SQA) {
        String[] PDS= new String[]{"VECS"};
        String ISQ_PSD= "";
        while(injectPDI(ISQ_PSD).length()> 0) {
            ISQ_PSD+= PDSEncode(SQA);
        }
        return ISQ_PSD;
    }
    public String ESU_M_SVQ_PDS_OEU_O(String SQA) {
        String[] PDS= new String[]{"DIUQ"};
        String ISQ_PSD= "";
        while(injectPDI(ISQ_PSD).length()> 0) {
            ISQ_PSD+= PDSEncode(SQA);
        }
        return ISQ_PSD;
    }
    public String ESU_M_SVQ_PDS_OEU_A(String SQA) {
        String[] PDS= new String[]{"HTX"};
        String ISQ_PSD= "";
        while(injectPDI(ISQ_PSD).length()> 0) {
            ISQ_PSD+= PDSEncode(SQA);
        }
        return ISQ_PSD;
    }
    public String ESU_M_SVQ_PDS_OEU_OA(String SQA) {
        //AEC.VSQ.IC.IE
        return SQA;
    }
}
```

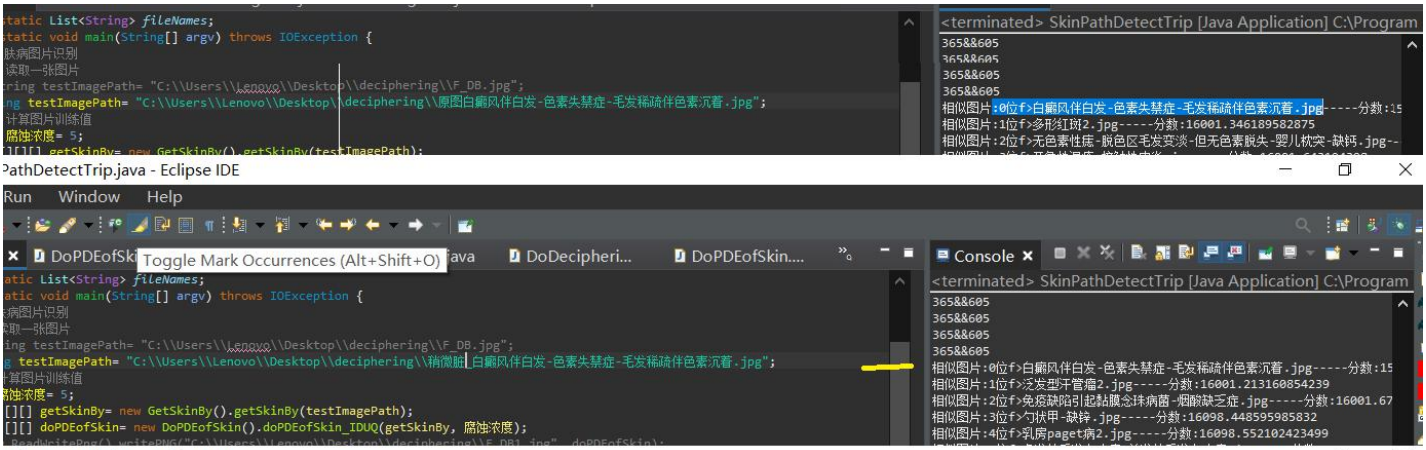
展示：　　原图　　稍微脏

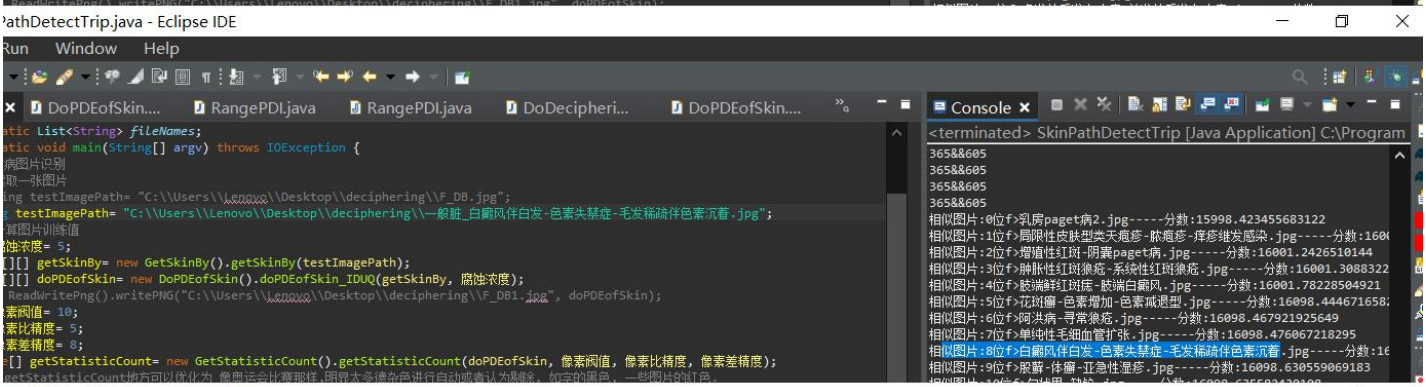
一般脏　　脏

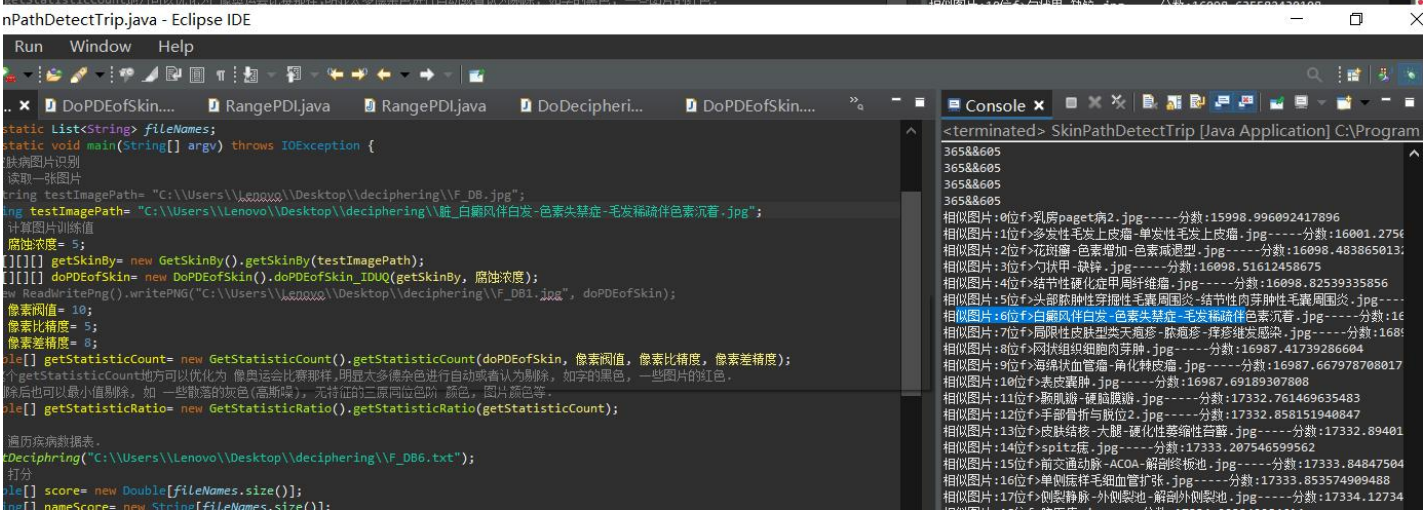




读脏识别结果：分别在第 1 位， 第 1 位， 第 8 位， 第 7 位 如下所示

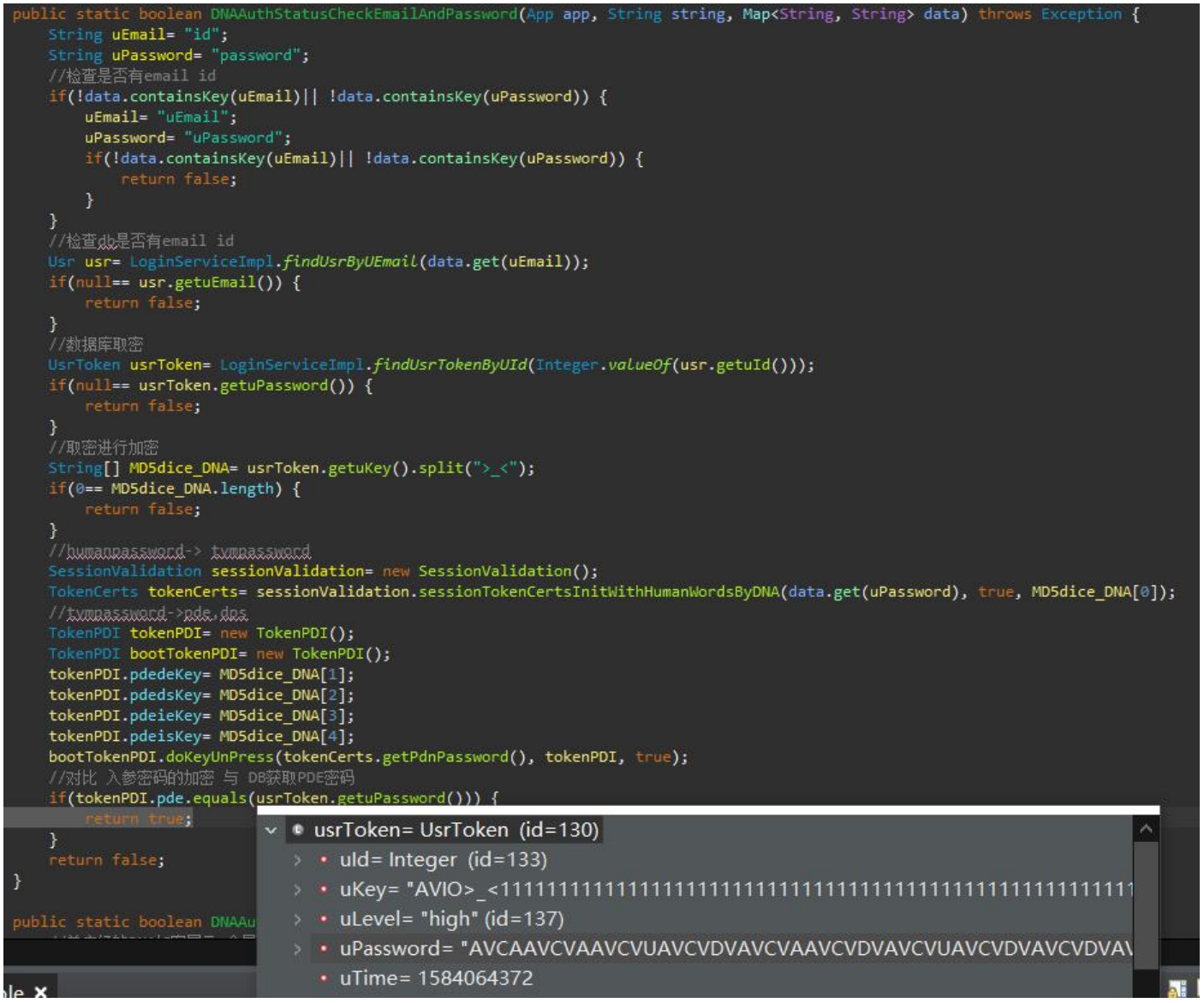






```
//DNA token session status 源码
//写个新的DNA status 跟之前的区分开来
   public static boolean DNAAuthStatusCheckEmailAndPassword(App app, String string,
   Map<String, String> data) throws Exception {
         String uEmail= "id";
         String uPassword= "password";
         //检查是否有email id
         if(!data.containsKey(uEmail)|| !data.containsKey(uPassword)) {
               uEmail= "uEmail";
               uPassword= "uPassword";
               if(!data.containsKey(uEmail)|| !data.containsKey(uPassword)) {
                     return false;
               }
         }
         //检查db是否有email id
         Usr usr= LoginServiceImpl.findUsrByUEmail(data.get(uEmail));
         if(null== usr.getuEmail()) {
               if(null!= data.get("uType")) {
                     if(data.get("uType").equals("register")) {
                           return true;//202108030254 罗瑶光 区别register添加模拟
                     }
               }
               return false;
         }
         //数据库取密
         UsrToken usrToken=
   LoginServiceImpl.findUsrTokenByUId(Integer.valueOf(usr.getuId()));
         if(null== usrToken.getuPassword()) {
               return false;
         }
         //取密进行加密
         String[] MD5dice_DNA= usrToken.getuKey().split(">_<");
         if(0== MD5dice_DNA.length) {
               return false;
         }
         //humanpassword-> tvmpassword
         SessionValidation sessionValidation= new SessionValidation();
```

```
TokenCerts tokenCerts=
   sessionValidation.sessionTokenCertsInitWithHumanWordsByDNA(data.get(uPassword), true,
   MD5dice_DNA[0]);
         //tvmpassword->pde,dps
         TokenPDI tokenPDI= new TokenPDI();
         TokenPDI bootTokenPDI= new TokenPDI();
         tokenPDI.pdedeKey= MD5dice_DNA[1];
         tokenPDI.pdedsKey= MD5dice_DNA[2];
         tokenPDI.pdeieKey= MD5dice_DNA[3];
         tokenPDI.pdeisKey= MD5dice_DNA[4];
         bootTokenPDI.doKeyUnPress(tokenCerts.getPdnPassword(), tokenPDI, true);
         //对比 入参密码的加密 与 DB获取PDE密码
         if(tokenPDI.pde.equals(usrToken.getuPassword())) {
               return true;
         }
         return false;
   }

//DNA token session status 展示
```

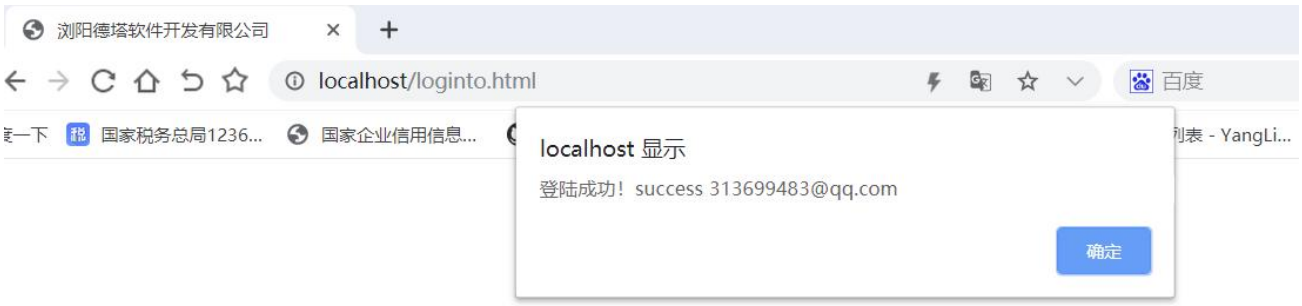


//DNA token session login 源码

```
//不在老函数里面改，重命名一个新的DNA开头 罗瑶光
//数据库token的存根是pde，token是pds
//token验证是输入pds，比对的pde
//这个函数设计完后我就设计register，注册一个pde存根，这样就可以验证这个登陆函数了。
//罗瑶光 20210731
public static Map<String, Object> transactionLoginDB_DNA(String uEmail, String uPassword)throws Exception {
        Usr usr = LoginServiceImpl.findUsrByUEmail(uEmail);
        UsrToken usrToken = LoginServiceImpl.findUsrTokenByUId(usr.getuId());
        //DNA SESSION 替换
        //老表的 getuKey 就是dna的getPdnLock，我要做的就是把pds4个概率钥匙也合并。
       //先定个格式为 >_< 分开 如 lock>_<de>_<ds>_<ie>_<is, split就好处理了。
        String[] MD5dice_DNA= usrToken.getuKey().split(">_<");
        SessionValidation sessionValidation= new SessionValidation();
        TokenCerts tokenCerts= sessionValidation.sessionTokenCertsInitWithHumanWordsByDNA(uPassword, true,
MD5dice_DNA[0]);
        TokenPDI pDE_RNA_Formular= new TokenPDI();
        TokenPDI pDE_RNA_Formular1= new TokenPDI();
        pDE_RNA_Formular1.pdedeKey= MD5dice_DNA[1];
        pDE_RNA_Formular1.pdedsKey= MD5dice_DNA[2];
```

```
        pDE_RNA_Formular1.pdeieKey= MD5dice_DNA[3];
        pDE_RNA_Formular1.pdeisKey= MD5dice_DNA[4];


        pDE_RNA_Formular.doKeyUnPress(tokenCerts.getPdnPassword(), pDE_RNA_Formular1, true);
        System.out.println("pds--3>"+ pDE_RNA_Formular1.pds);
        //
        if (!pDE_RNA_Formular1.pde.equals(usrToken.getuPassword())) {
                Map<String, Object> out = new HashMap<>();
                out.put("loginInfo", "unsuccess");
                out.put("returnResult", "unsuccess");
                return out;
        }

        sessionValidation= new SessionValidation();
        tokenCerts= sessionValidation.sessionTokenCertsInitWithHumanWordsByDNA(uPassword, false, null);
        PEU.P.dna.Token token= sessionValidation.sessionInitByTokenPDICertsDNA(tokenCerts);
        String passwordString= String_ESU.charsetSwap(token.getmPassword(), "GBK", "GBK");
        String passwordEncoder= String_ESU.stringToURIencode(passwordString, "UTF8");
        System.out.println("pds--1>"+ token.getmPasswordPDS());
        String passwordPDE= passwordEncoder.toString();
        String de= token.getUpdsde(); //对应PDS 概率钥匙加密
        String ds= token.getUpdsds(); //对应PDS 概率钥匙加密
        String ie= token.getUpdsie(); //对应PDS 概率钥匙加密
        String is= token.getUpdsis(); //对应PDS 概率钥匙加密
        String lock= tokenCerts.getPdnLock()+ ">_<"+ de+ ">_<"+ ds+ ">_<"+ ie+ ">_<"+ is;//对应非对称的筛子模拟
上锁

        //Token token = TokenUtil.getNewTokenFromUsrAndUsrToken(usr, usrToken);
        //去掉概率钥匙密码PDE, 仅仅时间 pds 和 id 即可
        token.I_Updsie("");
        token.I_Updsis("");
        token.I_Updsde("");
        token.I_Updsds("");
        token.I_uKey("");
        token.I_mPassword("");

        //// 仅仅含有时间, EMAIL 和 pds 3个
        token.I_uEmail(uEmail);
        String json = new Gson().toJson(token);
        String jsonToken = StringUtil.encode(json);
        LoginServiceImpl.U_UsrTokenByUId(usr.getuId(), lock, passwordPDE, token.getuTime()/1000);
        Map<String, Object> out= new HashMap<>();
        out.put("userToken", jsonToken);// 仅仅含有时间, EMAIL 和 pds 3个
        out.put("userEmail", uEmail);
        out.put("loginInfo", "success");
        return out;
}
```

//DNA token session login 展示



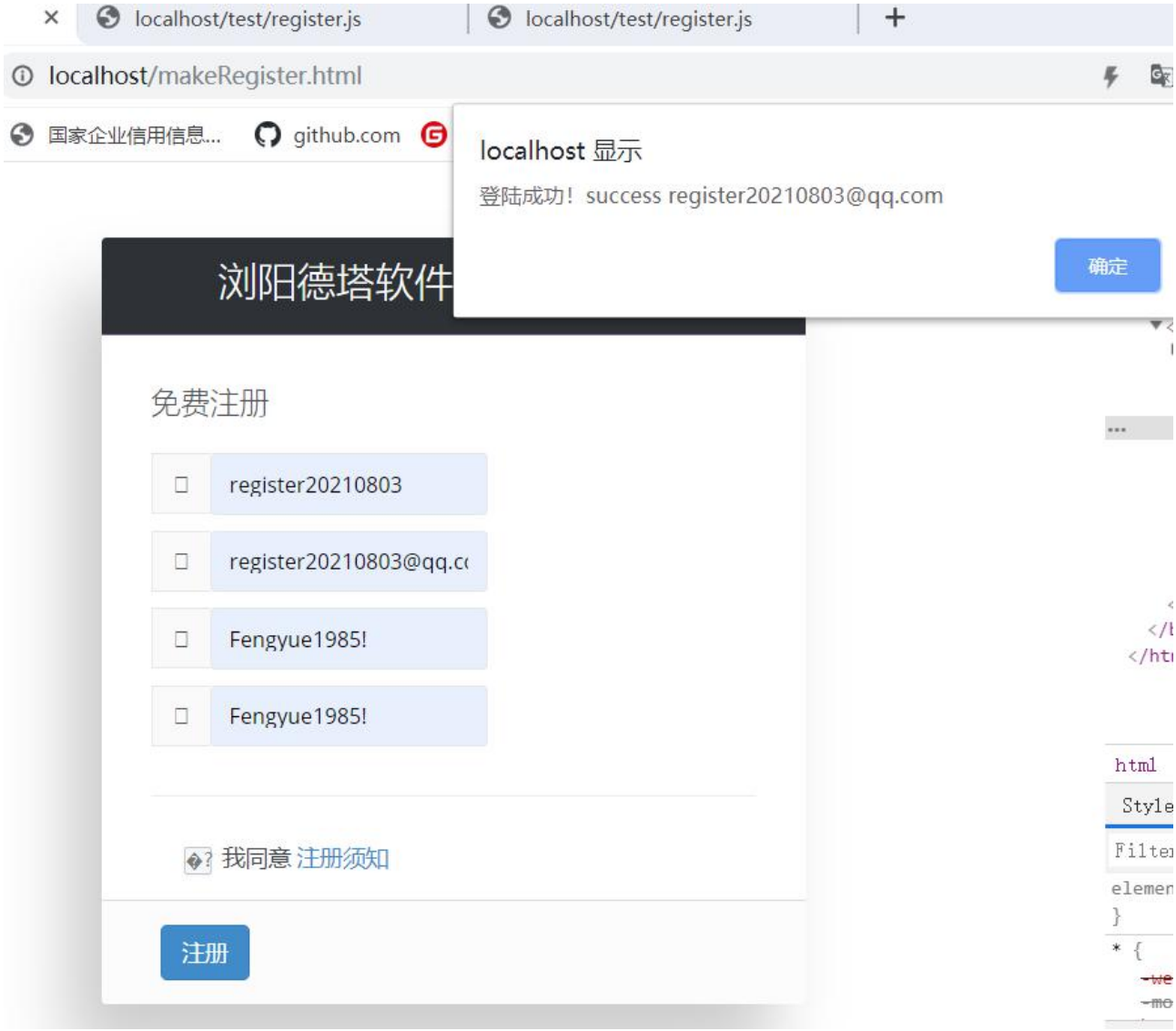
//DNA token session register 源码

```
	//注册函数也用dna设计先分开来。
	@SuppressWarnings("unused")
	public static Map<String, Object> transactionRegisterDB_DNA(String uEmail, String uEmailEnsure, String
uName, String uPassword,
			String uPassWDEnsure, String uAddress, String uPhone, String uWeChat, String uQq, String
uAge,
			String uSex) throws Exception {
		Usr usr = LoginServiceImpl.findUsrByUEmail(uEmail);
		if(usr.getuEmail()!=null) {
			Map<String, Object> out = new HashMap<>();
			out.put("loginInfo", "unsuccess");
			out.put("returnResult", "unsuccess");
			return out;
		}
		JSONObject jsobj=new JSONObject();
		jsobj.put("u_email", uEmail);
		jsobj.put("u_name", uName);
		//用dna代替
		//jsobj.put("u_password", TokenUtil.getSecondMD5Password(uPassword));
		//模拟加个测试账号: 313699483@QQ.COM, 密码: Fengyue1985!
		String id= uEmail;
		String idString= String_ESU.charsetSwap(id, "GBK", "GBK");
		String idEncoder= String_ESU.stringToURIencode(idString, "UTF8");
		String password= uPassword;
```

```
//加密
                SessionValidation sessionValidation= new SessionValidation();
                TokenCerts tokenCerts=
sessionValidation.sessionTokenCertsInitWithHumanWordsByDNA(password, false, null);
                PEU.P.dna.Token token= sessionValidation.sessionInitByTokenPDICertsDNA(tokenCerts);
                String passwordString= String_ESU.charsetSwap(token.getmPassword(), "GBK", "GBK");//pde
                String passwordEncoder= String_ESU.stringToURIencode(passwordString, "UTF8");
                System.out.println("pds--1>"+ tokenCerts.getPds());
                jsobj.put("u_password", token.getmPassword());//pde
                jsobj.put("u_address", uAddress);
                jsobj.put("u_phone", uPhone);
                jsobj.put("u_weChat", uWeChat);
                jsobj.put("u_qq", uQq);
                jsobj.put("u_age", uAge);
                jsobj.put("u_sex", uSex);
                jsobj.put("u_id", "random");
                LoginServiceImpl.IU_RowByTablePath("backend", "usr", jsobj);
                usr = LoginServiceImpl.findUsrByUEmail(uEmail);
                JSONObject jsobjToken= new JSONObject();
                jsobjToken.put("u_id", usr.getuId());
                jsobjToken.put("u_level", "low");

                String de= token.getUpdsde(); //对应PDS 概率钥匙加密
                String ds= token.getUpdsds(); //对应PDS 概率钥匙加密
                String ie= token.getUpdsie(); //对应PDS 概率钥匙加密
                String is= token.getUpdsis(); //对应PDS 概率钥匙加密
                String lock= tokenCerts.getPdnLock()+ ">_<"+ de+ ">_<"+ ds+ ">_<"+ ie+ ">_<"+ is;//对应非对称
的筛子模拟上锁

                jsobjToken.put("baseName", "backend");
                jsobjToken.put("u_key", lock);
                jsobjToken.put("u_password", token.getmPassword());
//              jsobjToken.put("u_password", TokenUtil.getSecondMD5Password(uPassword));
                LoginServiceImpl.IU_RowByTablePath("backend", "usrToken", jsobjToken);
                return transactionLoginDB_DNA(uEmail, uPassword);
        }
```

Register.js

```
        $http.get('register?uEmail=' + encodeURIComponent(email)
                        + '&uName=' + encodeURIComponent(usrname)
                        + '&uType=' + encodeURIComponent("register") //加个标识，稍后优化 罗瑶光 20210803
                        + '&uPassword=' + encodeURIComponent(password))
                        .then(function successCallback(response) {//DNA token session register 展示
```


localhost/test/register.js
localhost/test/register.js
localhost/makeRegister.html
国家企业信用信息...
github.com
localhost 显示
登陆成功！success register20210803@qq.com
确定
浏阳德塔软件
免费注册
register20210803
register20210803@qq.c
Fengyue1985!
Fengyue1985!
我同意 注册须知
注册
html
Style

PdeSwap Test DEMO

```
package VPC.transaction;
import MSU.AMS.VQS.SQV.SI.OSU.SMV.http.SessionValidation;
import OSI.OPE.SI.SD.SU.SQ.ASU.OSU.PSU.MSU.AVQ.ASQ.ASU.MPE.procedure.pde.FullDNATokenPDI;
import OSI.OPE.SI.SD.SU.SQ.ASU.OSU.PSU.MSU.AVQ.ASQ.ASU.MPE.procedure.pde.Initon;
import OSI.OPE.SI.SD.SU.SQ.ASU.OSU.PSU.MSU.AVQ.ASQ.ASU.MPE.procedure.pde.InitonLinkDNA;
import OSI.OPE.SI.SD.SU.SQ.ASU.OSU.PSU.MSU.AVQ.ASQ.ASU.MPE.procedure.pde.TokenPDI;
import PEU.P.dna.TokenCerts;
//把 token pdi 例子进行拆分成 3 个函数，之后方便集成到 data swap 包中。
//肽展作者 罗瑶光
//编码作者 罗瑶光
//见 VPC.transaction
//见 DNA 元基催化与肽计算 第 841 页公式
//20210828
public class PdeSwap{
    public static String PdcToPde(String pdc, String lock, String de, String ds, String ie, String is) {
        SessionValidation sessionValidation= new SessionValidation();
        TokenCerts tokenCerts= sessionValidation.sessionTokenCertsInitWithHumanWordsByDNA(pdc, true, lock);
        TokenPDI pDE_RNA_Formular= new TokenPDI();
        TokenPDI pDE_RNA_Formular1= new TokenPDI();
        pDE_RNA_Formular1.pdedeKey= de;
        pDE_RNA_Formular1.pdedsKey= ds;
        pDE_RNA_Formular1.pdeieKey= ie;
        pDE_RNA_Formular1.pdeisKey= is;
        pDE_RNA_Formular.doKeyUnPress(tokenCerts.getPdnPassword(), pDE_RNA_Formular1, true);
        //System.out.println("pde--3>"+ pDE_RNA_Formular1.pde);
        return pDE_RNA_Formular1.pde;
    }
    public static String PdcToPds(String pdc, String lock, String de, String ds, String ie, String is) {
        SessionValidation sessionValidation= new SessionValidation();
        TokenCerts tokenCerts= sessionValidation.sessionTokenCertsInitWithHumanWordsByDNA(pdc, true, lock);
        TokenPDI pDE_RNA_Formular= new TokenPDI();
        TokenPDI pDE_RNA_Formular1= new TokenPDI();
        pDE_RNA_Formular1.pdedeKey= de;
        pDE_RNA_Formular1.pdedsKey= ds;
        pDE_RNA_Formular1.pdeieKey= ie;
        pDE_RNA_Formular1.pdeisKey= is;
        pDE_RNA_Formular.doKeyUnPress(tokenCerts.getPdnPassword(), pDE_RNA_Formular1, true);
        //System.out.println("pds--3>"+ pDE_RNA_Formular1.pds);
        return pDE_RNA_Formular1.pds;
    }
    // MPOASCEV
    public static String PdeToPds(String pds, String lock, String de, String ds, String ie, String is) {
        FullDNATokenPDI pDE_RNA_Formular2= new FullDNATokenPDI();
        pDE_RNA_Formular2.pdeieKey= ie;
        pDE_RNA_Formular2.pdeisKey= is;
        pDE_RNA_Formular2.pdedeKey= de;
        pDE_RNA_Formular2.pdedsKey= ds;
```

```
        System.out.println("准备计算元基 DNA 序列："+ pds);
        //pds~pde
        Initon[] initon= new Initon[pds.length()];
        for(int i= 0; i< pds.length(); i+ + ) {
            if(initon[i]= = null) {
                initon[i]= new Initon();
            }
            initon[i].I_Initon(""+ pds.charAt(i));
            if(i+ 1< pds.length()) {
                if(initon[i+ 1]= = null) {
                    initon[i+ 1]= new Initon();
                }
                initon[i].next= initon[i+ 1];
                initon[i+ 1].prev= initon[i];
            }
        }
        InitonLinkDNA initonLinkDNA= new InitonLinkDNA();
        Initon InitonPDEM= pDE_RNA_Formular2.doDecrementM(initon[0], initonLinkDNA, pDE_RNA_Formular2);
        Initon InitonPDEP= pDE_RNA_Formular2.doDecrementP(InitonPDEM, initonLinkDNA, pDE_RNA_Formular2);
        Initon InitonPDEO= pDE_RNA_Formular2.doDecrementO(InitonPDEP, initonLinkDNA, pDE_RNA_Formular2);
        Initon InitonPDEA= pDE_RNA_Formular2.doDecrementA(InitonPDEO, initonLinkDNA, pDE_RNA_Formular2);
        Initon InitonPDES= pDE_RNA_Formular2.doDecrementS(InitonPDEA, initonLinkDNA
            , pDE_RNA_Formular2, true);
        Initon InitonPDEC= pDE_RNA_Formular2.doDecrementC(InitonPDES, initonLinkDNA, pDE_RNA_Formular2);
        Initon InitonPDEE= pDE_RNA_Formular2.doDecrementE(InitonPDEC, initonLinkDNA
            , pDE_RNA_Formular2, true);
        Initon InitonPDEV= pDE_RNA_Formular2.doDecrementV(InitonPDEE, initonLinkDNA, pDE_RNA_Formular2);
        while(InitonPDEV.hasNext()) {
            pDE_RNA_Formular2.pds+ = InitonPDEV.getStore();
            InitonPDEV= InitonPDEV.next;
        }
        pDE_RNA_Formular2.pds+ = InitonPDEV.getStore();
        while(InitonPDEV.hasPrev()) {
            InitonPDEV= InitonPDEV.prev;
        }
        return pDE_RNA_Formular2.pds;
    }
    //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。
    // VECSAOPM
    public static String PdsToPde(String pds, String lock, String de, String ds, String ie, String is) {
        FullDNATokenPDI pDE_RNA_Formular2= new FullDNATokenPDI();
        pDE_RNA_Formular2.pdeieKey= ie;
        pDE_RNA_Formular2.pdeisKey= is;
        pDE_RNA_Formular2.pdedeKey= de;
        pDE_RNA_Formular2.pdedsKey= ds;
        System.out.println("准备计算元基 DNA 序列："+ pds);
        //pds~pde
```

```
        Initon[] initon= new Initon[pds.length()];
        for(int i= 0; i< pds.length(); i+ + ) {
            if(initon[i]= = null) {
                initon[i]= new Initon();
            }
            initon[i].I_Initon(""+ pds.charAt(i));
            if(i+ 1< pds.length()) {
                if(initon[i+ 1]= = null) {
                    initon[i+ 1]= new Initon();
                }
                initon[i].next= initon[i+ 1];
                initon[i+ 1].prev= initon[i];
            }
        }
        InitonLinkDNA initonLinkDNA= new InitonLinkDNA();
        Initon InitonPDE1V= pDE_RNA_Formular2.doIncrementV(initon[0], initonLinkDNA, pDE_RNA_Formular2);
        Initon InitonPDE1E= pDE_RNA_Formular2.doIncrementE(InitonPDE1V, initonLinkDNA
            , pDE_RNA_Formular2, true);
        Initon InitonPDE1C= pDE_RNA_Formular2.doIncrementC(InitonPDE1E, initonLinkDNA, pDE_RNA_Formular2);
        Initon InitonPDE2S= pDE_RNA_Formular2.doIncrementS(InitonPDE1C, initonLinkDNA
            , pDE_RNA_Formular2, true);
        Initon InitonPDE1A= pDE_RNA_Formular2.doIncrementA(InitonPDE2S, initonLinkDNA, pDE_RNA_Formular2);
        Initon InitonPDE1O= pDE_RNA_Formular2.doIncrementO(InitonPDE1A, initonLinkDNA, pDE_RNA_Formular2);
        Initon InitonPDE1P= pDE_RNA_Formular2.doIncrementP(InitonPDE1O, initonLinkDNA, pDE_RNA_Formular2);
        Initon InitonPDE1M= pDE_RNA_Formular2.doIncrementM(InitonPDE1P, initonLinkDNA, pDE_RNA_Formular2);
        while(InitonPDE1M.hasNext()) {
            pDE_RNA_Formular2.pde+ = InitonPDE1M.getStore();
            InitonPDE1M= InitonPDE1M.next;
        }
        pDE_RNA_Formular2.pde+ = InitonPDE1M.getStore();
        while(InitonPDE1M.hasPrev()) {
            InitonPDE1M= InitonPDE1M.prev;
        }
        return pDE_RNA_Formular2.pde;
    }
    public static void main(String[] argv) {
        FullDNATokenPDI pDE_RNA_FullFormular= new FullDNATokenPDI();
        String a= "luoyaoguang";
        pDE_RNA_FullFormular.encodeDNA(a);
        @SuppressWarnings("unused")
        String initonKeys= "EIU/0.6/EDU/0.4/si/0.3/sq/0.7/EIU/0.5/EDU/0.5/si/0.632/sq/0.368";
        pDE_RNA_FullFormular.key[0]= 0.6;
        pDE_RNA_FullFormular.key[1]= 0.3;
        pDE_RNA_FullFormular.key[2]= 0.5;
        pDE_RNA_FullFormular.key[3]= 0.632;
        pDE_RNA_FullFormular.text= "控制吸收";
        pDE_RNA_FullFormular.pdw= pDE_RNA_FullFormular.initonSect(pDE_RNA_FullFormular.text);
        System.out.println("原文: "+ pDE_RNA_FullFormular.text);
        //pDE_RNA_FullFormular.pdw= "字典保密：MSIOCUOCIPCUPCI";
```

```
        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];
        for(i= 0; i< pDE_RNA_FullFormular.pdw.length(); i+ + ) {
            pDE_RNA_FullFormular.code+ = pDE_RNA_FullFormular.lock + pDE_RNA_FullFormular.pdw.charAt(i);
        }
        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);
        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= PdeSwap.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= PdeSwap.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);
    }
}
```

德塔 华瑞集 养疗经 软件工程类 源码引用综合表

工程类名

DNA 元基催化版本:德塔行为分析图灵机

引用

罗瑶光，中华人民共和国 国家版权局 软著登字第 3951366 号 德塔自然语言图灵系统 V10.6.1

CNN 思想 提出者 Yann LeCun，Wei Zhang，Alexander Waibel 等

ANN 思想 数据挖掘教材

RNN 思想 提出者 M. I. Jordan，Jeffrey Elman

HMM 思想 提出者 隐.马尔可夫

申明

GNU GPL 2.0 协议

工程类名

DNA 元基催化版本:德塔精神分析图灵机

引用

罗瑶光，中华人民共和国 国家版权局 软著登字第 3951366 号 德塔自然语言图灵系统 V10.6.1

CNN 思想 提出者 Yann LeCun，Wei Zhang，Alexander Waibel 等

ANN 思想 数据挖掘教材

RNN 思想 提出者 M. I. Jordan，Jeffrey Elman

HMM 思想 提出者 隐.马尔可夫

申明

GNU GPL 2.0 协议

工程类名

DNA 元基催化版本:德塔语言图灵机

引用

罗瑶光，中华人民共和国 国家版权局 软著登字第 3951366 号 德塔自然语言图灵系统 V10.6.1

CNN 思想 提出者 Yann LeCun，Wei Zhang，Alexander Waibel 等

ANN 思想 数据挖掘教材

RNN 思想 提出者 M. I. Jordan，Jeffrey Elman

HMM 思想 提出者 隐.马尔可夫

申明

GNU GPL 2.0 协议

工程类名

DNA 元基催化版本:德塔意识图灵机

引用

罗瑶光，中华人民共和国 国家版权局 软著登字第 3951366 号 德塔自然语言图灵系统 V10.6.1申明

GNU GPL 2.0 协议

工程类名

DNA 元基催化:德塔 DNA 元基催化与肽计算第三修订版综合

引用

罗瑶光，罗荣武，中华人民共和国 国家版权局 国作登字-2021-A-00097017 DNA 元基编码 类人 DNA 与 神经元基于催化算子映射编码方式 V_1.2.2

罗瑶光，罗荣武，中华人民共和国 国家版权局 国作登字-2021-L-00103660 DNA 元基催化与肽计算第二卷养疗经应用研究

20210305

罗瑶光，中华人民共和国 国家版权局 国作登字 2021-A-00042587 DNA 元基公式 AOPM VECS IDUQ 肽展公式推导与元基编码进化计算以及它的应用发现 1.2.2
罗瑶光，中华人民共和国 国家版权局 国作登字 2021-A-00042586 DNA 元基解码 DNA 催化与肽展计算和AOPM-TXH-VECS-IDUQ 元基解码 V013_026 中文版本
申明
GNU GPL 2.0 协议

工程类名
DNA 元基催化版本:数据预测包
技术和引用
罗瑶光，中华人民共和国 国家版权局 软著登字第 5447819 号 数据预测引擎系统 V1.0.0
坐标熵增化:欧基里德
点线面知识:解析几何教材
三维空间坐标计算:立体几何教材
极限小区间叠加:高等数学教材
欧拉图论:离散数学教材
正交概率差值:数据挖掘教材
申明
GNU GPL 2.0 协议

工程类名
DNA 元基催化版本:肽计算催化函数集
引用
罗瑶光，中华人民共和国 国家版权局 国作登字 2021-A-00042586 DNA 元基解码 DNA 催化与肽展计算和AOPM-TXH-VECS-IDUQ 元基解码 V013_026 中文版本
罗瑶光，罗荣武，中华人民共和国 国家版权局 国作登字-2021-L-00103660 DNA 元基催化与肽计算第二卷养疗经应用研究20210305
申明
GNU GPL 2.0 协议

工程类名
DNA 元基催化版本:Unicorn ETL
申明
GNU GPL 2.0 协议
罗瑶光, 中华人民共和国 国家版权局 软著登字第 4240558 号 德塔 ETL 人工智能可视化数据流分析引擎系统 V1.0.2
ETL 节点的分类命名采用 KNIME, ORANGE, KETTEL, Weka, ORACLE 等统一标准分类函数: config 配置,execute 运行, show or view 查看.
神经元节点我设计的外观来自牛津大学的牛顿.霍华德教授发布的神经细胞图片的外观获得的灵感(该神经元图片展示的是人类真实的神经元.).
申明
GNU GPL 2.0 协议

工程类名
DNA 元基催化版本:肽计算催化函数集
使用须知
引用
罗瑶光，中华人民共和国 国家版权局 国作登字 2021-A-00042586 DNA 元基解码 DNA 催化与肽展计算 和AOPM-TXH-VECS-IDUQ 元基解码 V013_026 中文版本
罗瑶光，罗荣武，中华人民共和国 国家版权局 国作登字-2021-L-00103660 DNA 元基催化与肽计算第二卷养疗经应用研究

20210305
申明
GNU GPL 2.0 协议

工程类名
DNA 元基催化版本:德塔 Socket 流可编程数据库
引用
罗瑶光，中华人民共和国 国家版权局 软著登字第 4317518 号 德塔 Socket 流可编程数据库语言引擎系统 V1.0.0
TCPIP 标准协议下研发的.
申明
GNU GPL 2.0 协议

工程类名
DNA 元基催化版本:Socket 流 PLSQL Socket 数据库引擎
引用
DNA 元基催化版本:数据智慧语言
罗瑶光, 中华人民共和国 国家版权局 软著登字第 4317518 号 德塔 Socket 流可编程数据库语言引擎系统 V1.0.0
电子书籍来源:
1 人类 5000 年文明历史中出现的古籍经典.
2 作者目前电脑里的医学电子书 下载主要来自 微信的 qq 群下载(免费)，百度的文库下载(花钱)，豆丁的文库下载(花钱)，和父亲给作者的医学教材.
3 特别感谢其中 豆丁的文库上医学书籍内容最完整，也最严谨.
申明
GNU GPL 2.0 协议

工程类名
DNA 元基催化版本:VPCS 操作类汇总聚集接口
引用
1 文件读写操作, C 语言程序设计, Java How To Program 6th
2 罗瑶光, 关于VPCS, 中华人民共和国 国家版权局 国作登字-2021-A-00097017 DNA 元基编码 类人 DNA 与神经元基于催化算子映射编码方式 V_1.2.2
申明
GNU GPL 2.0 协议

工程类名
DNA 元基催化版本: Unicorn UI
引用
AWT, JDK1.6 Resource, Sun Technology.
作者编码出现的问题 2014 年+ 在 谷歌搜索 StackOverflows 英文网站上找解决方案. 如 Jdk DEMO 的JButton 对象的 动画继承.
罗瑶光，中华人民共和国 国家版权局 软著登字第 4240558 号 德塔 ETL 人工智能可视化数据流分析引擎系统 V1.0.2
申明
GNU GPL 2.0 协议

工程类名
DNA 元基催化版本:VPCS WEB 2.0
引用
CGI/TCPIP
罗瑶光，中华人民共和国 国家版权局 软著登字第 4317518 号 德塔 Socket 流可编程数据库语言引擎系统V1.0.0
罗瑶光，罗荣武，中华人民共和国 国家版权局 国作登字-2021-A-00097017 DNA 元基编码 类人 DNA 与 神经元基于催化算子映

射编码方式 V_1.2.2
罗瑶光，罗荣武，中华人民共和国 国家版权局 国作登字-2021-L-00103660 DNA 元基催化与肽计算第二卷养疗经应用研究20210305

工程类名
DNA 元基催化版本:数据智慧语言
引用
罗瑶光，中华人民共和国 国家版权局 软著登字第 4317518 号 德塔 Socket 流可编程数据库语言引擎系统 V1.0.0
罗瑶光，中华人民共和国 国家版权局 软著登字第 4240558 号 德塔 ETL 人工智能可视化数据流分析引擎系统 V1.0.2
罗瑶光，罗荣武，中华人民共和国 国家版权局 国作登字-2021-A-00097017 DNA 元基编码 类人 DNA 与 神经元基于催化算子映射编码方式 V_1.2.2
罗瑶光，中华人民共和国 国家版权局 国作登字 2021-A-00042587 DNA 元基公式 AOPM VECS IDUQ 肽展公式推导与元基编码进化计算以及它的应用发现 1.2.2
PLSQL 思想定义见:Oracle 数据库对 SQL 语句的扩展.(因为 可编程数据库思想关键字词汇被 Oracle 首发了，所以就引用下甲骨文．作者人生的所有实际工作从没有用过甲骨文的 PLSQL 技术.)
ORM 思想：作者最早接触是在蓝汛, Kiyor 给作者展示 golang 有 mysql 的 orm API 包 的 UnSQL 语法句型，演示的很酷炫．作者在 INTEL 实际工作了 2 个月用 Hibernate ORM UNSQL API 操作 MSSQL，当时感觉没有完整的文档，研发相当痛苦．
申明
GNU GPL 2.0 协议

工程类名
DNA 元基催化:德塔极速分词包
引用
罗瑶光，中华人民共和国 国家版权局 软著登字第 3951366 号 德塔自然语言图灵系统 V10.6.1
CNN 思想 提出者 Yann LeCun，Wei Zhang，Alexander Waibel 等
ANN 思想 数据挖掘教材
RNN 思想 提出者 M. I. Jordan，Jeffrey Elman
HMM 思想 提出者 隐.马尔可夫
增加注释下：我的 6 万中英文语义词库因为 有涉及 新华字典的词语录入．没有在 德塔自然语言图灵系统 V10.6.1 个人软著的申请资源中.
1 用复旦大学的免费分词软件(LGPL-3.0 license)进行将词语的词性标注.
2 用百度,谷歌,有道,免费在线翻译进行词语的翻译，特别感谢有道，翻译质量最高.
3 6 万词库有 2 万是 百度上买了新华字典电子书录词语，感谢新华字典. (仅仅录了 2 万个词语的名而已．词语解释,词语造句，偏旁部首，同义反义词汇等全部没有抄录．我当时用浏览器中 英文单词对应的中文翻译记录的同义词汇)
4 另外 4 万是我在新华字典的分词后 出现的未知词汇不断的记录统计下，然后增加的.
4.1 Github 竟然还是卡阿卡的，我就再细节溯源.我在百度文库上不单买了新华字典的词语，我还买了成语词汇列表和国家中文语言过级的词汇(不多,几百几千个而已),我都只录了词语而已．4.1.1 再说一个细节，记得当时我买了有一个词汇 txt 竟然是一行显示,我还因此把算法按行读改成了按字节 buffer 读．可以查嘛．4.2 我的购买方式是微信类扫码百度文库包月能免费下载买的.(提交于 2021年5月7日 04:14我的支付宝虽然一年多没用了，现在一直没有注销.国家可查询，大家再看我后注销支付宝的时间，因为注销健康宝的时候给我显示欠费一分钱，永远消不掉，我的信用分就一直无法申诉修改的自动减少。于是把支付宝注销了就没有健康宝的任何信息了。避免猫腻 git log ： https://github.com/yaoguangluo/YangLiaoJing_HuaRuiJi/commit/d9f860b87e29c33e246cb0ca5265db76041695d2)
5 一部份词语是未知的单字进行同词性连着进行合并出来的新词并记录的.
申明
GNU GPL 2.0 协议 100% 源码全部开源发布，避嫌．申明下 1 我的分词方法函数在 2019 年 4 月 3 号就写完了，2019 年 5 月 28 号就下证了．2 复旦大学的邹锡鹏 在同年 7 月才提出新的分词方法.(刚浏览器搜的时候弹出这个信息，我没细看) 3 我的源码已经全部开源，双方 git 都有开源, giff 一下，是否逻辑相同.就是了,新的分词方法和我

相同就是 邹锡鹏抄袭. 4 我的源码 4 月 3 号的 和 复旦的逻辑相同 就是 罗瑶光抄袭. 简单的很. 注明我的分词当时每秒达到 1800 万中文分词速度. 当时 lucene 中文分词速度每秒 20 万.(刚看了下,复旦那篇论文Multi-Criteria Chinese Word Segmentation with Transformer 采用的是机器学习的训练分词, 和本人的 按人类语言语法定义分词 不同) 罗瑶光 20210507
申明
GNU GPL 2.0 协议

工程类名
DNA 元基催化版本:养疗经 与 华瑞集 数据库应用接口文档
引用
罗瑶光, 中华人民共和国 国家版权局 软著登字第 4317518 号 德塔 Socket 流可编程数据库语言引擎系统 V1.0.0
DNA 元基催化与肽计算 第一卷 page306, 307, 308, 309
申明
GNU GPL 2.0 协议

工程类名
DNA 元基催化版本:养疗经 与 华瑞集 数据对象类操作包
引用
罗瑶光, 中华人民共和国 国家版权局 软著登字第 4607950 号 德塔数据结构变量快速转换 V1.0
申明
GNU GPL 2.0 协议

工程类名
DNA 元基催化版本:VPCS 线程类汇总聚集接口
引用
1 进程, 操作系统教材.
2 多线程, 操作系统教材.
3 罗瑶光, 罗荣武, VPCS, 中华人民共和国 国家版权局 国作登字-2021-A-00097017 DNA 元基编码类人 DNA 与 神经元基于催化算子映射编码方式 V_1.2.2
申明
GNU GPL 2.0 协议

工程类名
DNA 元基催化版本:数据变换引擎
引用
罗瑶光, 中华人民共和国 国家版权局 软著登字第 4607950 号 德塔数据结构变量快速转换 V1.0
JSON API 来自 Google Gson CSV API 来自 POI 区别下我给章鑫杰写的源码(我在美国 2014 年后有一次问赵川,我的 KNIME 节点资源文件不见了, 他说也没保留, 给我 exe, 我听了很痛苦, 说我再写个. 就自己用 POI 又写了个.)记得我当时对话场景是用 qq 聊天的.另外股市的数据变换我贴出来是前年钟意来我家一次装了'暴雪'后, 我的股市分析源码莫名丢失,我当天随即把股市源码备份包进行 git 发布. 避免猫腻.(当天我 Windows 操作系统有 360 杀毒 和 360 安全卫士.)
申明
GNU GPL 2.0 协议

工程类名
DNA 元基催化版本:小波分析
引用
圆周率: 祖冲之
小波叠加分治法: 高斯代数, 牛顿微积分法
国:傅里叶

FFT 蝶形运算思想: J.W.库利
FFT 蝶形运算思想: J.W.图基
快速傅里叶
离散余弦变换
申明
GNU GPL 2.0 协议 函数的由来是作者在 2014 年后用 java.math 的数据包计算傅里叶变换,发现该数学包的写法,接口规范化了，不能满足作者要的自适应计算,于是阅读 source 包，把离散余弦函数拿出来,重新做成卷积离散函数变换.基础来自 2012~2013 年 作者在路德大学 与 Renhart 教授探讨为什么在 2kpi/2pi 周期卷积 cosh 变换 2 维图片，噪声那么大,不是书上输出图形 当时教授说不是滤波,不要把卷积核用图片长与宽以代替,太大了.
对话地址:进门右边大教室黑板后面的办公室内. 教材<计算机视觉>绿皮书，教材里面有明确的 DCT 思想伪编码.
在溯源下为什么从 java.math 包提取:因为 apache 的 java.math 是 jdk 开源原生组件，最近成为 commons.math，我提取了什么: 提取了 cos 的 for loop，把 0n 的 n 拿出来 作为参数，这样就能控制卷积子核的大小. 因为 java.math 当时固定了大小,我用会失真. 就这么回事我再溯源下,当时 java.math 的傅里叶可不是我这样用的，当时记得好像是用的是 vector，用 while 循环 记得是，记得就没有 N, ,我就用 N 代替了，这个 N 是怎么来的,我再溯源,记得是我买了一本书，等我找下,当时数据处理我记得我 refer 了那一页找打了 springer 的 盲信号处理,理论与实践，红皮书，教材第 273 页 9.8.6 公式例子.离散余弦变换和 9.8.5 离散傅里叶变换 DFT, ,这本书记得是很久前 20142015 年前后 在长沙近方台新华书店买的，教材里面有各种明确的 DCT 公式.

工程类名
DNA 元基催化版本:线性卷积包
引用
罗瑶光, 中华人民共和国 国家版权局 软著登字第 4584594 号 Java 数据分析算法引擎系统 V1.0.0
罗瑶光, 罗荣武，中华人民共和国 国家版权局 国作登字-2021-A-00097017 DNA 元基编码 类人 DNA 与 神经元基于催化算子映射编码方式 V_1.2.2
东尼·霍尔的快速排序 1-4 代思想; 泡沫，粘贴，插入，堆排序等思想，均来自算法导论教材.
索贝尔，高斯，拉普拉斯, Emboss 思想等，均来自计算机视觉教材.
DFT, FFT 思想 来自 傅里叶，库力，图基.
java sound API Demo 来自 stackflows, Sun Technology.
树和图论思想来自 离散数学教材.
Cache 缓存的甲骨文和 SUN 的官方 demo 函数 来自 亚米的王浩雷 介绍我三级缓存思想，我用谷歌搜索的.
GZIP 来自哈夫曼的编码思想. 提出者是 Jean-loup Gailly 和 Mark Adler
SOUND 来自 JDK sound 原生组件 demo.
三维的 JOGL 画图函数因为不是 ETL 肽节点.所以我从数据智能 ETL 包 并到 该数据处理包下
申明
GNU GPL 2.0 协议

工程类名
DNA 元基催化版本:养疗经 与 华瑞集 后端 Rest 接口文档
引用
罗瑶光, 中华人民共和国 国家版权局 软著登字第 3951366 号 德塔自然语言图灵系统 V10.6.1
罗瑶光, 中华人民共和国 国家版权局 软著登字第 4317518 号 德塔 Socket 流可编程数据库语言引擎系统 V1.0.0
罗瑶光, 罗荣武, 中华人民共和国 国家版权局 国作登字-2021-A-00097017 DNA 元基编码 类人 DNA 与 神经元基于催化算子映射编码方式 V_1.2.2
VPCS Public 规范
TCP/IP 规范
HTTP 规范
Restful 规范
申明
GNU GPL 2.0 协议