北京大學
PEKING UNIVERSITY

Java 程序设计

WordNet 大作业报告

苑　斌
刘自然
孙唯童
李汪洋

July 5, 2015

目录

1 WordNet 项目简介

1.1 Wordnet

WordNet 是由 Princeton 大学的心理学家，语言学家和计算机工程师联合设计的一种基于认知语言学的英语词典。它不是光把单词以字母顺序排列，而且按照单词的意义组成一个“单词的网络”。

它是一个覆盖范围宽广的英语词汇语义网。名词，动词，形容词和副词各自被组织成一个同义词的网络，每个同义词集合都代表一个基本的语义概念，并且这些集合之间也由各种关系连接。（一个多义词将出现在它的每个意思的同义词集合中）。在 WordNet 的第一版中（标记为 1.x），四种不同词性的网络之间并无连接。WordNet 的名词网络是第一个发展起来的，正因如此，我们下面将要讨论的大部分学者的工作都仅限于名词网络。

名词网络的主干是蕴涵关系的层次 (上位/下位关系)，它占据了关系中的将近 80%。层次中的最顶层是 11 个抽象概念，称为基本类别始点（unique beginners），例如实体（entity，“有生命的或无生命的具体存在”），心理特征（psychological feature，“生命有机体的精神上的特征）。名词层次中最深的层次是 16 个节点。

1.2 需求分析

此次大作业要求我们在 Wordnet 的基础上重新开发了一个新型的单词查询器。其中包括功能为单词查询，按照名词，形容词，副词，动词，查询。并且将界面做的更加好看。并且支持同义词，反义词，上义词，下义词查询。并且这些词的关系是以图的方式呈现在我们的面前。

也就是说要设计一个优秀的 wordnet 浏览器，要做到以下几点：

1. 读入文件 data, index
2. 根据 index 建立单词的属性
3. 根据 data 获取每个单词的近义词、反义词、下义词、上义词
4. 建立相关界面（包括图形呈现区域，选择区域，词性按钮，关系按钮，查询框，输出框）
5. 实现每个按钮的事件（输入单词按回车显示相应图形，并且显示相关解释，在词性按钮切换的情况下能够重新查询按钮）
6. 实现图的绘制
7. 实现历史查询结果功能
8. 实现相应的数据结构来实现单词查询
9. 实现图片截取功能

针对以上要求，我们做出了如下的设计：首先，用 JAVA 的 SWING 库建立一个基本的电子词典查询界面，输入单词后，能快速显示词义、例句、上下义词、近反义词等信息，为了便于用户操作，除了点击“查询”按钮能激活查询动作外，还对输入框添加了回车事件监听，使得用户输完单词后，敲回车键也能立刻完成查询。其次，我们使用了特殊的数据结构保存数据，快捷的查询算法实现查询，使查询信息能“瞬间”呈现在输出框中。然后，我们为了更好的展示 wordnet 词库中词的关系，在每一个被查单词输出框后面，附加了一个词网关系图，能让与该单词构成近义、反义、上下义关系的词以该词为中心形成一个树形结构图，并添加了保存词网关系图的功能。此外，在细节方面，我们还实现了记录查询历史、查特定词性的词义、智能匹配查询等等人性化的设计。我们还不断优化缩减代码，让这款浏览器小而精致。我们希望通过以上设计，能实现一个集用户友好、功能强大、小巧轻便于一身的 wordnet 浏览器。

1.3 任务分工

- **苑斌**：Data 与 index 文件处理，数据结构编写，查询单词接口编写
- **刘自然**：字典文件以及数据读入处理
- **孙唯童**：GUI 设计与制作，合并文件
- **李汪洋**：数据可视化显示，文档整理

2 实现简介以及接口说明

2.1 数据读入

ReadFile.java 类用于数据的读取和保存。

要将文件的每行作为一个 string 保存，所以首先根据每个文件的行数开相应大小的 string 类型数组。八个文件的行数分别为 117127，81456，11518，13680，22170，18906，4630，3673，依此，在类 ReadFile 中首先声明 8 个表示文件行数的常量：

```
static final int INDEX_NOUN_LINES = 117200;
static final int DATA_NOUN_LINES = 81500;
static final int INDEX_VERB_LINES = 11600;
static final int DATA_VERB_LINES = 13700;
static final int INDEX_ADJ_LINES = 22200;
static final int DATA_ADJ_LINES = 19000;
static final int INDEX_ADV_LINES = 4700;
static final int DATA_ADV_LINES = 3700;
```

然后在全局声明了 index_noun 等八个数组。

ReadFile 类的构造函数如下：

```
public ReadFile() throws IOException{
        getNoun();
        getVerb();
        getAdj();
        getAdv();
}
```

在构造函数 ReadFile() 中，分别调用 getNoun(), getVerb(),getAdj(),getAdv() 函数进行名词、动词、形容词、副词的读取，这样实例化该类即可完成文件的读入工作。

在每个读文件函数中，选择用 BufferedReader 进行文件的读取：

```
BufferedReader br =
        new BufferedReader(new FileReader("data/index.noun"));
```

由于每个数据文件中，前 29 行用于注释说明，是无用信息，故应过滤。所以使用

```
for(int j = 0; j < 29; j++)
        br.readLine();
```

忽略前 29 行的字符串。接下来用 while 循环一直读到文件的结尾，并将每行数据保存在前面声明的八个 string 数组中。最后，不忘关闭刚刚打开的文件流。

2.2 数据存储

我以每个单词封装了一个类。此类的名称为 word。此外我将同一个语义关系也封装了一个类，名称为 synnet。

我们此次用到的数据包括两种文件，一种是 data，一种是 index. 其中 index 为每个单词的索引，形如如下所示：

```
aback r 2 0 2 0 00076140 00076057
```

其中第一项表示单词名称，第二项表示单词类型，第三项表示有多少个项集，第四项表示有几种关系，第五项依然是项集数，第六项为关系的 offset, 比如在此实例中，单词为 aback，类型为 adj, 共有两个项集，在 data 中的位置分别为 00076140 00076057.

其中 data 文件如下所示：

```
00001740 03 n 01 entity 0 003 ~ 00001930 n 0000 ~ 00002119 n 0000 ~
    04372113 n 0000 | that which is perceived or known or inferred to have
     its own distinct existence (living or nonliving)
```

其中第一项表示为 offset，第二项表示在哪个 file 中，第三个依然表示单词属性（名词，形容词，动词，副词）。第四个表示有几个单词，格式为十六进制。第五个表示 n 个相同意义的单词接下来就是关系。第七个为关系的数目，接下来就是关系的表示，接下来是 n 个关系其中我们提取了四种比较典型的关系作为此次展示的目标，它们分别为同义词关系，反义词关系，上义词关系以及下义词关系，其中同义词反义词很容易理解。上义词则表示的是某种物体的统称，比如说苹果，香蕉的统称就是水果。下义词表示更加具体的描述，比如苹果就是水果的下义词，他们的关系应该是一棵树，具有很明显的上下级关系。每一项都显示出来了关系类型，以及相应的 offset。最后一项为解释。

我在处理这张表的时候是先读入 index，接下来按照 index 中的每个 offset 来得到相应的关系。然后分别在 data 文件中去寻找相应的 offset，找到之后将这些关系加入到该 word 当中去。其中单词我用一个数组来存储，并且这个数组按照单词名称来排序，在搜索的时候我利用二分检索，这样可以提高搜索的效率。并且每个单词类中我存储了四种关系，分别为同义词，反义词，上义词和下义词。相当于利用一个邻接表来存储图结构。

附上每个类以及相关方法说明：

1. DealWithData 类型说明 (用于查询单词，传递结果的类)

```
//方法：

//用于确定每个index文件中有多少条目。并初始化size
public void Start()

//将每个同义的项集加入到word的关系当中去。 第一个参数为要添加的单词，
    第二个参数为一个关系的数组，第三个参数为要加入的单词类型
public void PutSynnetIntoWord(word cur, Vector<synnet> syn,String
    charac)

//打印出每个单词所对应的关系，用于调试使用
public void printall(word tep)

//按照名称和词性来查询单词，返回一个word类型
public word getwordaccordname(String name, String charac)

//根据一个单词中关系的offset来查询得到一个关系的数组
public Vector<synnet> FindTheSynnet(word root,String characteristic)

//根据offset返回一个synnet
public synnet FindSynnetAccordOffset (String offset ,String
    BinarySearchLine)
```

```
20
21 //判断当前的offset和一行中第一个项（也是Offset）的大小关系，若为真，
       则表示第一项大于第二项，若为假，则表示第二项小于第一项
22 public boolean compareStringLine(String cur ,String line)
23
24 //利用二分法得到一个单词，但并未得到其相应的关系
25 public word FindAccordName(String name,String characteristic )
```

2. relation 类型说明

```
1  //相关成员变量
2  public class relation extends Object{
3          public  String offset;//地址偏移
4          public  String type;//词性
5          public  String rela;//关系
6          public  String reindex;//关系类型
7          public  String[] name;//名字
8  }
9
10 //synnet关系集合
11
12 //data中的名词
13 static public synnet[] dn = new synnet[ReadFile.DATA_NOUN_LINES];
14 //data中的形容词
15 static public   synnet[] dadj = new synnet[ReadFile.DATA_ADJ_LINES];
16 //data中的副词
17 static public   synnet[] dadv = new synnet[ReadFile.DATA_ADV_LINES];
18 //data中的动词
19 static public   synnet[] dv = new synnet[ReadFile.DATA_VERB_LINES];
20
21 //data 形容词中项集的个数
22 static public int dadjsize = ReadFile.data_adj.length;
23 //data 副词中项集的个数
24 static public int dadvsize = ReadFile.data_adv.length;
25 //data 动词词中项集的个数
26 static public int dvsize = ReadFile.data_verb.length;
27 //data 名词中项集的个数
```

```
28 static public int dnsize =  ReadFile.data_noun.length;
29
30 private String offset;  //位置
31 private String fnum;    //文件标号？
32 private String type;    //类型
33 private int w_cnt;      //词语个数
34 private String[] words; //词语
35 private  String[] words_num;    //词语的十六进制
36 private int RelaNum;    //关系数目
37
38 relation[] Rela;        //关系列表
39
40 String frame;   //句型
41 String gloss;   //注释或者例句
42
43 //方法：
44 public synnet(){}// TODO Auto-generated constructor stub
45 public void Setoffset(String off){}//设置偏移量
46 public void Setfnum(String num){}//设置文件位置
47 public void Settype(String rhs){}//设置类型
48 public void SetWordNum(int num){}//设置词语个数
49 public void SetRelationNum(int num){}//关系个数
50 public void Setgloss(String rhs){}//添加注释，解释
51 public String Gettype(){}//返回类型
52 public String Getoffset(){}//返回偏移量
53 public String GetFileNum(){}//返回文件编号
54 public int GetWordNum(){}//返回单词个数
55 public String[] GetWords(){}//返回单词们
56 public String[] GetWordshex(){}//返回单词的十六进制
57 public int GetRelaNum(){}//返回关系个数
58 public relation[] GetRelation(){}//返回关系
```

3. word 类型说明

```
1 //相关成员变量
2
3 static int InSize = ReadFile.INDEX_NOUN_LINES;  //index 中的名词 个数
```

```
static int IvSize = ReadFile.INDEX_VERB_LINES;  //index中的动词个数
static int IadjSize = ReadFile.INDEX_ADJ_LINES; //index中的形容词个数
static int IadvSize = ReadFile.INDEX_ADV_LINES; //index中的副词个数

//index中的名词
public static word[] In = new word[ReadFile.INDEX_NOUN_LINES];
//index中的动词
static public word[] Iv = new word[ReadFile.INDEX_VERB_LINES];
//index中的形容词
static public word[] Iadj = new word[ReadFile.INDEX_ADJ_LINES];
//index中的副词
static public word[] Iadv = new word[ReadFile.INDEX_ADV_LINES];

private String name;//单词名字
private String type;//类型 四种 n,adj,adv,v
private int relationnum;//关系数目
private int synnetnum;//义项数目
private int frequence;//频次

Vector<String> Relations = new Vector<String>();

String items[] ;
ArrayList<synnet> antonymy = new ArrayList<synnet>();   //反义词指针
ArrayList<synnet> hypermymy = new ArrayList<synnet>();  //上位词指针
ArrayList<synnet> hyponymy = new ArrayList<synnet>();   //下位词指针
ArrayList<String> synsets = new ArrayList<String>();    //同义词集合

public boolean isvisit;//判断是否已经添加关系进入此单词。

//相关实现方法:
public boolean Hassyn(){}//返回是否有同义词
public boolean Hasanto(){}//是否有反义词
public boolean Hashyper(){}//返回是否有上位词
public boolean Hashyponymy(){}//返回是否有下义词
public int compareTo(Object o){}比较函数
public String Getname(){}//返回该单词的名称
```

```
40 public String GetType(){}//返回该单词的类型
41 public int GetSynnetNum(){}得到关系数目
42 public void SetType(String ty)//设置单词类型
43 public void SetName(String na){}//设置单词名称
44 public void Setsynnetnum(int num){}//设置单词关系数目
```

2.3 图形界面部分

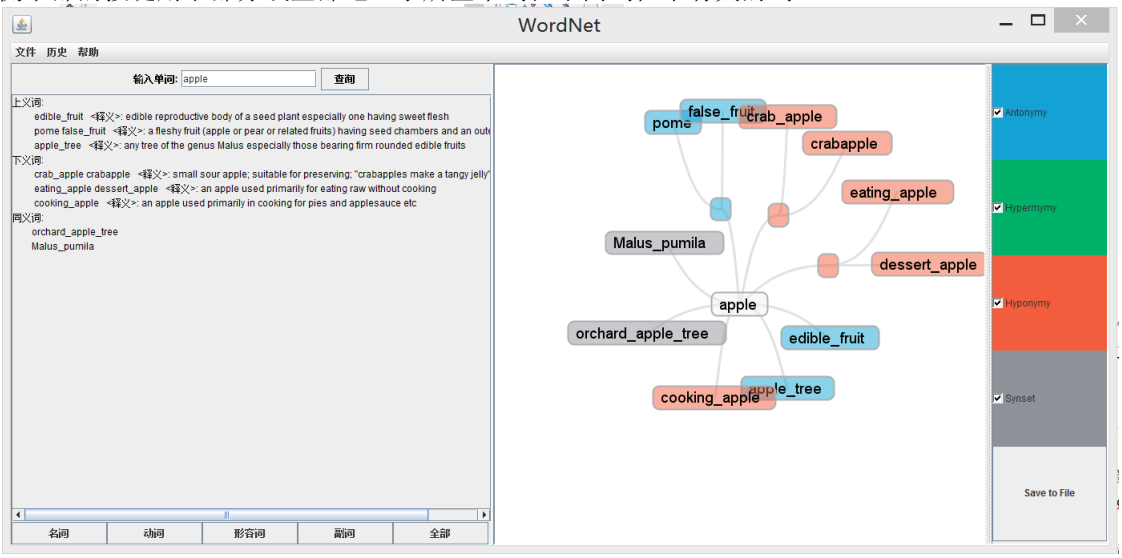
带有顶部菜单的 JFrame 构成图形界面的主体，JFrame 中由 JSplitPane 分割出左右两部分，右侧部分在一 JPanel 中通过 GraphView 显示树状图，具体在 2.4 部分阐述。左侧布置于一顶级 JPanel 上，分上中下三个部分——上部为提示、单词输入和处理栏，中部为文字结果显示，下部为词性选择按钮。

JFrame 顶菜单栏分文件、历史、帮助三个部分。“文件”菜单有两个下拉选项用于退出以及保存右侧树状图。“历史”菜单可记录五个以内的不同单词输入记录，帮助快速定位之前的查询。“帮助”菜单用于显示一些版本信息。

左侧顶的文本输入框用于输入待查询单词。“查询”键监听“文本输入框”中的回车或其上点击，之后：

- 查找单词打印在左侧文本框中
- 在右侧图中展示与其有关系的词
- 在“历史”栏记录所查单词

左侧下部的按键用于部分或全部地显示所查单词在不同词性下有关的词。



涉及类及相关方法说明：

- Frame 类，用于提供操作界面，提示用户操作并响应用户要求

```
//界面组成（成员变量）：
private JFrame f = new JFrame( "WordNet" );
private JSplitPane  mainSplitPane;//分割出
private JPanel pTop = new JPanel();//左顶文字输入栏
private JPanel pBottom = new JPanel();//左底按钮
private JPanel pTree = new JPanel();//树状图
//##显示树的类，由  GraphView 实现
private GraphView netplace = new GraphView(null);
//绘制左侧窗口背景的Panel，暂未设置背景
public JPanel background = new JPanel();
private String contest;//输入到文本框的内容

//相关实现方法：

//将各部件添加到所属位置并提供相关监听
public void init()throws IOException
//主函数，运行init()
public static void main( String [] args )throws IOException
//根据contest内容查询，并显示结果
public void  displayText(String contest,int intcharac)throws
    IOException

public void  antonymytext(word tep)throws IOException//反义词打印
public void  hypermymytext(word tep)throws IOException//上义词打印
public void  hyponymytext(word tep)throws IOException//下义词打印
public void  synsetstext(word tep)throws IOException//同义词打印

private void sureExit()//√判断是否要退出
```

2.4 图像呈现处理

使用第三方开源库 Prefuse 进行实现，将词与词之间复杂的网状关系以图的形式直接进行呈现。主要是运用 Prefuse 支持，绘制图形，并维护 HashMap 来对“边”、“点”等进行维护管理，确保在图形绘制的过程中速度最快。同时，图形中的节点还支持交互功能，主要功能包括，左键单击则旋转图形，将点击节点做为中心节点展示；左键双击，将所选词进行展开，进一步扩展图形；右键单击则会自动对图形进行缩放，已达到最好的显示效果等，并且通过 Prefuse 的过滤器功能，可以按照需

要选择关系进行显示。涉及类及相关方法说明：

- GraphView 类，扩展 JPanel，内部包含图像

```
//成员函数：

//定义Prefuse中的字段名称
private static final String graph = "graph";
private static final String nodes = "graph.nodes";
private static final String edges = "graph.edges";
private static final String label1 = "name";
private static final String label2 = "id";

//Relationship表示各种关系的显示情况
private boolean[] Relationship = {true, true, true, true, true};

//表示各种关系的名称
private String[] RelationshipName = {"反义词", "上义词", "下义词", "
    同义词"};

//主体显示对象
private Graph g;
private Display display;
protected Visualization m_vis = new Visualization();

//用于扩展单词查询
private DealWithData SearchWord = new DealWithData();

//设置默认颜色
private final int[] FillColor = new int[] {
            ColorLib.rgba(242, 242, 242, 127), ColorLib.rgba(20, 162,
                212, 127),
            ColorLib.rgba(0, 177, 106, 127), ColorLib.rgba(242, 94,
                61, 127),
            ColorLib.rgba(142, 146, 152, 127)};

//用于维护点边关系
private HashMap<String,Node> OriginNode = new HashMap<String,Node>();
```

```
32 private HashMap<String,Node> AntonymyNode = new HashMap<String,Node
       >();
33 private HashMap<String,Node> HypermymyNode = new HashMap<String,Node
       >();
34 private HashMap<String,Node> HyponymyNode = new HashMap<String,Node
       >();
35 private HashMap<String,Node> SynsetsNode = new HashMap<String,Node>()
       ;
36
37
38 //相关实现方法:
39 public Display getDisplay(){}//对外提供获取内部图形引用的接口
40 public void setWord(word wd){}//重置图形为新的单词查询
```

3 使用说明

- **概述**：显示界面主体分两部分，左侧查询与文字显示和右侧的树图显示，可以按照客户需求通过拖拽竖条改变各部分所占比例。
- **查询功能**：使用时在左上部文本框中输入所查单词，敲击回车或点击查询键，既在左侧文本框中显示该词各词性的上义词、下义词、同义词、反义词。其中上下义词中将语义相近的词在一行打印，之间空格分割，并对其各含义进行了解释。默认只输出名词的语义。
- **分词性显示**：通过单击左下侧各按键，选择部分或全部的显示查询单词的相关语义。如，在 apple 下，单击“名词”键，将显示 apple 的所有名词的语义关系。如果内容为空，则显示为空。
- **树状图语义选择性显示**：勾选最右侧 checkbox，按需求显示特定的语义关系。
- **历史回查**：顶部菜单栏“历史”-> 选择之前查过的单词，避免重复输入。
- **左键点击图中节点**：旋转图使选中目标至于中间，并查询其释义
- **左键双击图中节点**：在图中按照此节点进一步展开
- **左键单击图中节点**：自动进行缩放
- **滚轮滑动**：对图形进行缩放
- **保存树状图**：

1. 右下角“保存树状图”按钮 -> 选择保存的位置 -> 命名 -> 保存。

2. 顶部菜单栏“文件”->“保存树状图”-> 选择保存的位置 -> 命名 -> 保存。

4 总结与感想

4.1 苑斌

通过此次大作业我学习到了很多的知识，熟悉了 java 语言的相关接口，以及对 java 泛型使用的了解。熟悉了如何使用 java 的 compareTo 来对数组排序。在熟悉接口的同时，也了解了封装的概念，对于每个类（比如 word 和 synnet）。我向外部提供了接口，使得不需要通过成员变量的直接访问来设置成员变量。同时我对于 index 中的每个单词都进行 ASCII 码的排序，这样在查找的时候就可以通过二分查找找到单词了。算是一个小小的优化吧。在这次大作业的过程中，我还发现如果一开始将 data 中的每一项都加入到 word 当中去，那么就会出现堆溢出的问题。于是我就换了一种策略，每次在查询的时候才将 data 中的项集加入到 word 当中去。这样极大的节省了内存空间的占用。算是第二个小小的优化吧。通过这次 java 的大作业，我熟悉了一般使用 java 写程序的方法，熟悉了图形界面的相关内容，知道了可以调用外部库来优化我们的程序。可以说，在此次大作业中，我收获颇多。

4.2 刘自然

Java 程序设计这门课是我本学期最喜欢的一门课之一，对于一个热爱 CODING 的人，接触一门新的语言就像探险家发现一片新的大陆一样。JAVA 语言的严谨、健壮，还有面向对象程序设计的优雅，令我深深为这门语言着迷。

编写 Wordnet 浏览器是这学期最好玩的一门大作业，充满挑战而又新鲜有趣。从开始的毫无头绪，到初具雏形的设计方案，再到实现时的困难重重，最后到收获成功的喜不自胜，我们小组一步一个脚印，一行一行码出了令我们满意的果实。在这次大作业中，我除了收获到一个自己编写的实用的电子词典，还学到了很多课堂之外的东西。

我懂得了一个好的程序员出了要具有卓越的 CODING 能力，还要有过人的交流技巧。首先，你要学会表达自己的想法。大型的项目需要团队的合作才能完成，即使你能以一己之力构建出整个项目的体系架构，要想实现他们，特别是雕琢那些细节，一个人也是不可能完成的。这个时候，你需要把你脑海中的蓝图用生动的语言表达出来，让你的团队能理解能接受。这样，项目才能最终得以实现。其次，交流也表现在代码风格中。好代码的标准之一就是简明易懂，这次大作业才让我理解了这条要求的重要性。面对同伴看不懂的代码，那种抓耳挠腮焦头烂额的感觉会让人非常沮丧。经过这次大作业的磨练，我的代码风格提升了一个台阶，除了详细的注释，我还养成了写说明文档的习惯。我以后一定会继续坚持这个好的传统。

这次大作业还提升了我自学的能力。找源码、看 API、看博客……自学而来的知识掌握得更加牢固，通过查资料解决 BUG 后的自豪感也更加令人振奋。

然而这次大作业也有小小的遗憾。由于分工原因，我的任务比较轻，只负责编写文件读入。于是，我只写了很小的一部分代码。而且由于接口的原因，我写的两个类没有被收录到最终的代码里，只保留了一个类。这也让我知道了在大项目的建设当中，并不是你所有的代码都能被加入，有时候会“腰斩”你一半的努力。总的来说，这次大作业也坚定了我以后继续学习 JAVA 的决心。写代码的快乐，收货成果的满足感，将激励我在 JAVA 学习之路上越走越远！

4.3 孙唯童

在学习 java 的过程中，觉得实践是最重要的环节。很多以为已经清楚的了解的东西，在一开始拿起来动手的时候想从脑海中提取出相关知识点，然而却常常找不到或是仅仅得到一点残缺的记忆。在不断的作业与练习中，通过反复查找 api 并使用，很多东西才慢慢记住。

但记住所有语句并不是 java 学习的目标，要写出好的程序，不是单单依赖记住了多少类，而是在于对于整体和细节的宏观把握。整体在于知道类的大框架，其间的继承与使用关系，对于相似或有继承关系的类能快速进行类比和使用。细节在于了解类的一些细节实现和其作用。

很多时候对于 java，知道有什么比是什么更有用，这是因为其中的规则和关于细节的问题往往可以通过搜索解决，而搜索一个已知的类的细则要远简单于描述一个抽象的功能并搜索出预期结果。半成品的积累和了解面的拓展对于 java 编程意义重大。往往我们不倾向于重复工作，而是充分利用以前的工作，这样，才能把更多的精力投入到建设性工作中去。

一学期的学习，还仅仅是皮毛，java 学习日久天长。

4.4 李汪洋

我觉得对这门课的感触有很多吧，不光光是又学会了一门语言，或者说又获取了一些编程经验之类，我觉得在这门课里我得到的最大收获就是三个值得信赖的同伴，苑斌、刘自然和孙唯童学长，有趣的是，我们这一组没有一个人是纯正的信科人，我、苑斌和刘自然都是从其他院系转到信科来的，而孙唯童则是计算机双学位。刚开始分工的时候，我分到了一般认为是比较困难的数据可视化部分，还在想自己能不能搞定这些事，Prefuse 是我在这次大作业中主要使用的库，然而最大的问题就是 Prefuse 在国内使用的很少，几乎很难找到像样的中文资料，甚至可以说搜罗了很久网上也就只有两篇关于 Prefuse 的中文介绍，有的还只是翻译了官方文档中的一些解释罢了，在这种情况下，我被迫自己的看 Prefuse 的英文文档，一般去参照示例程序进行学习，期间遇到了很多的困难，刚开始的效果也不尽如人意，至少我自己都看不下去了，不过同伴们在 DDL 这件事上可以说是对我格外容忍了，我向来是最后一个完成大家预定的计划的，不过他们也并不怪我。或许课程的学习只有一个学期，不过结识的新朋友却是长久的。