大美 mixer　简书>>https://www.jianshu.com/u/84f0ebbbac87
Github>>https://github.com/lmh760008522　**CSDN**>>https://blog.csdn.net/qq_34746896

《DIY 我的北斗地图》

设计文档

目　　录

1　引 言

1.1 项目简要介绍

本系统名为“DIY 我的北斗地图”，提供的是基于 swing 的 Java 应用程序。DIY，是“Do it Yourself”的英文缩写。简单来说,DIY 就是自己动手制作,没有专业资质的限制,想做就做是每个人都可以利用 DIY 做出一份私人订制表达自我的“产品”来。

该项目主要分为两部分。第一部分是查看地图，用户可以根据自己的需要放大或缩小自己需要的地图;第二部分是添加地点,用户可以根据自己的喜好在地图上任意添加地点标识,随时记录自己的行为动作以及任何感想。

1.2 项目背景

每个人的潜意识都是以自我为中心的。我们在看照片的时候往往习惯性地第一眼就关注照片中的自己是否令人满意。对于产品来说，一切功能、逻辑、UI、交互和内容甚至运营也需要围绕用户的自我意识展开。

DIY 最初兴起于电脑的拼装,随后逐渐演绎成为一种流行生活方式，并形成了一种“自己去做，自己体验，挑战自我，享受其中的快乐”的积极向上的精神。DIY 没有所谓的高手和菜鸟之分，任何人任何时间只要想自己动手去做，就能享受其中的快乐，获得最适合自己的满意的成果。

DIY 的存在似乎总是开始于人们想省钱的心理。为了少付出，人们难免要选择"自己做"，而放弃"别人/机器做"。十几年前的"打家具热"着实让许多人过了一把"DIY 瘾"，就是因为不少人算了经济帐后，觉得自己做比买、比请人做都实惠，才纷纷自己动手做将起来。由于规模经营使得工业化生产成本远低于人工生产成本，花费不高还方便，人们可能就不会死抱住 DIY 不放了。

但是当人们看腻了市场上工业产品的千篇一律,或千篇一律的市场的市场产品无法满足自己家的特殊需要，"Do it Yourself"（我自己动手做）的念头就可能油然而生。做你需要的，做你想要的，做市场上绝无仅有、独一无二的你自己的作品，成为 DIY 更高层次的追求。

它在过程中给予人的满足则更重要。工业化生产的确已经日臻完美，越来越多的人已不可能也没必要去掌握旧日能工巧匠的手艺了。毋需自己动手做，却总仿佛缺了些什么，就象缺少了运动，人就是这么怪，需要贴近自然，需要运动劳作，需要在运动劳作中享受生活的乐趣。这，是 DIY 又一高层次的追求。

DIY，因为人的需要而存在。

1.3 项目的创新点

本系统利用北斗的地理位置信息，让更多的人了解北斗，支持我国的航天事业。

本系统倡导 DIY 精神，让用户可以根据自己的情况在地图上私人定制地点，同时记录下自己的感想，实现自己的创作，获得一定程度上的自我满足。

1.4 定义

名 称	描 述
Swing	Swing 是一个用于开发 Java 应用程序用户界面的开发工具包。以抽象窗口工具包（AWT）为基础使跨平台应用程序可以使用任何可插拔的外观风格。Swing 开发人员只用很少的代码就可以利用 Swing 丰富、灵活的功能和模块化组件来创建优雅的用户界面。
Windows	Microsoft Windows,是美国微软公司研发的一套操作系统，它问世于 1985 年，起初仅仅是 Microsoft-DOS 模拟环境，后续的系统版本由于微软不断的更新升级，不但易用，也慢慢的成为家家户户人们最喜爱的操作系统。
Derby	Apache Derby 是一个完全用 java 编写的数据库，Derby 是一个 Open source 的产品，基于 Apache License 2.0 分发。Apache Derby 非常小巧，核心部分 derby.jar 只有 2M，所以既可以做为单独的数据库服务器使用，也可以内嵌在应用程序中使用。Cognos 8 BI 的 Content Store 默认就是使用的 Derby 数据库，可以在 Cognos8 的安装目录下看到一个叫 derby10.1.2.1 的目录，就是内嵌的 10.1.2.1 版本的 derby。

1.5 参考资料

[1]昊斯特曼 (Horstmann Gay S.) , Gary Cornell; 《Java 核心技术》; 电子工业出版社
[2]Bruce Eckel; 《Thinking in Java 》; Prentice Hall
[3]Pearson Education,Inc; 《敏捷软件开发:原则、模式与实践》; 清华大学出版社
[4]周志明 ; 《深入理解 Java 虚拟机》; 机械工业出版社
[5]Joshua Bloch ; 《effective java》; Addison-Wesley Professional
[6]Bruce Eckel; 《Java 编程思想(第 4 版)》; 机械工业出版社
[7]Martin Fowler ; 《重构:改善既有代码的设计》; 碁峰
[8]Mark Allen Weiss; 《数据结构与算法分析》; 机械工业出版社
[9]弗罗斯特; 《数据库设计与开发》; 清华大学
[10]Abraham Silberschatz; 《数据库系统概念》; 机械工业出版社
[11]http://blog.csdn.net/chaiqunxing51/article/details/53735592
[12]http://yanguz123.iteye.com/blog/1819642
[13]http://wanggangwork.blog.51cto.com/6481803/1362389/
[14]http://www.cnblogs.com/azhqiang/p/3876214.html
[15]百度百科——DIY
[16]凤凰网——中国北斗卫星导航系统
[17]中国北斗物联网

[18]北斗网

2 任 务 概 述

2.1 目标

本系统专门为我国北斗事业的宣传和热衷于 DIY 的用户而设计的，旨在充分发挥“自己去做，自己体验，挑战自我，享受其中的快乐”的积极向上的精神。系统采用方便的基于 swing 的 java 应用程序，使用户能方便的定制可以展现自己个性的地图，从而获得自我满足感。

2.2 运行环境

1.硬件环境

CPU：建议 PII300 以上

RAM：128M 以上，建议 256M 以上

DISK：100M 以上的可用硬盘安装空间

2.软件环境

运行平台：Windows8.1 操作系统

运行环境：Sun JDK 1.8

数据库服务器：derby

开发技术平台：Eclipse Neon.1

3 功能需求

1.提供 Swing 应用程序界面访问系统。

2.提供用户使用指南页面，方便用户了解并使用系统，有助于新手快速上手。

3.提供用户测试样例说明，方便用户更好地深入系统操作。

4.提供地图的更换功能，方便用户应用于不同的场景。

5.提供地图的区域显示功能，方便用户随时了解当前地点所在的大致位置。

6.提供地图的缩放功能，便于用户查看和标记。

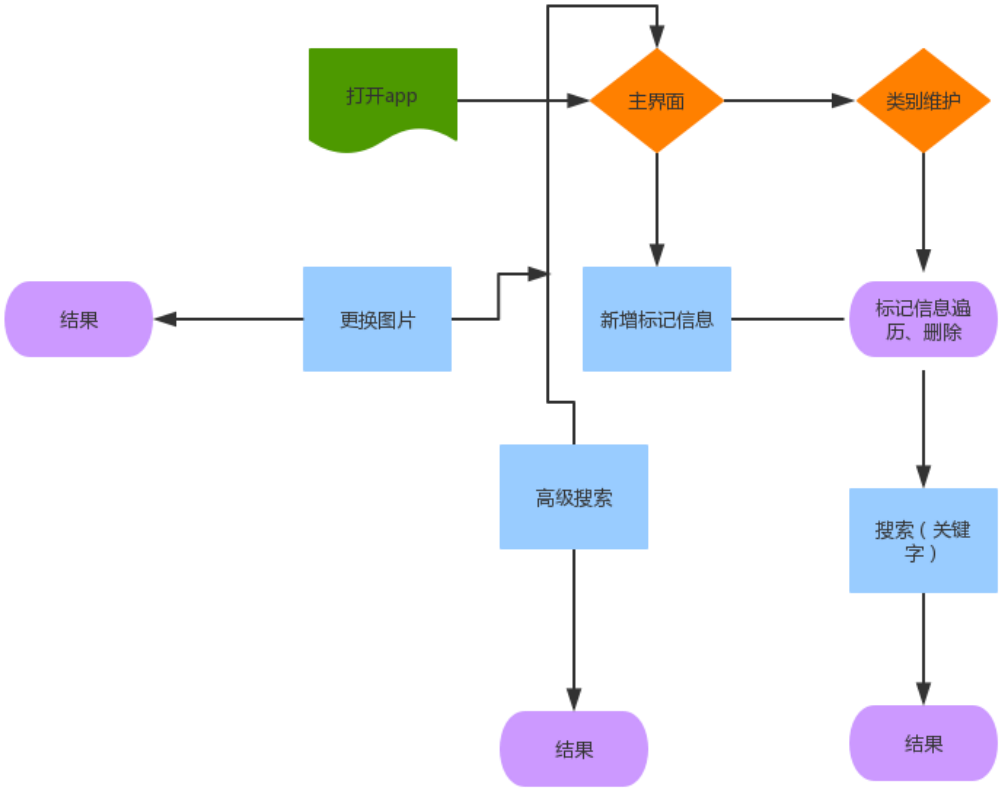
7.提供地图的维护功能，让用户能够轻松的进行个人需求化地图定制。

8.提供地图被标记点的普通搜索和精确查找功能，能够让用户快速锁定想要的地点。

9.提供自主添加地图标识功能，用户可以添加标记、类型、日期、说明等，实现 DIY 功能，实现对自己喜爱偏好设置。

4 总 体 设 计

4.1 基本设计概念和处理流程



简要说明：

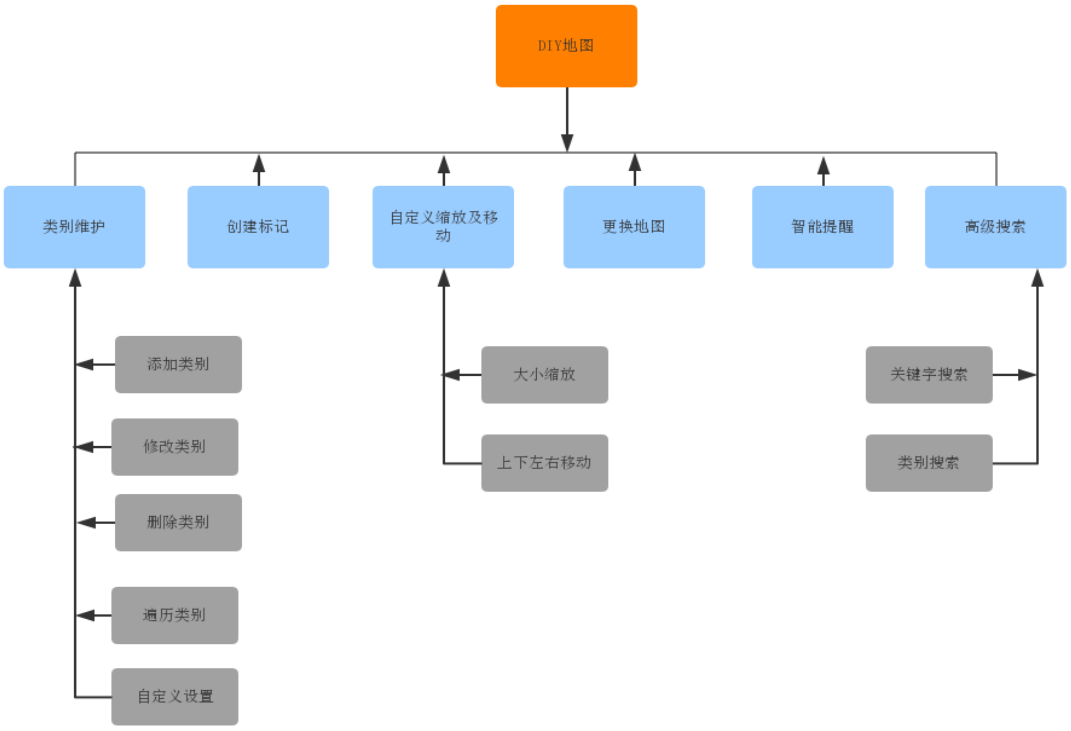
主界面：用户通过主界面可以实现查看地图、地图的左右移动与放大缩小；

更换图片：用户可以自主更换地图；

高级搜索：可以根据多个条件限制进行搜索已添加的地点；

新增标记信息：用户可以随意增加信息，比如类别、日期等等。

4.2 整体结构



各模块功能说明：

a)**类别维护模块：**用户可以对类别信息进行操作，使用个性化的名词或者短语定制属于自己的类别信息

- **添加类别：**用户可以添加类别信息
- **修改类别：**用户可以修改类别信息
- **删除类别：**用户可以删除类别信息
- **遍历类别：**用户可以对现有的类别信息进行遍历

b)**创建标记模块：**用户可以在地图的任意位置创建属于自己的标记，也可以对现有的标记进行操作。旨在增强用户的个性化需求，定制属于自己的地图。

c)**自定义移动及其缩放模块：**用户可以对地图进行大小缩放，根据不同的比例查看当前位置的具体信息。也可以上下左右移动视角，遍及地图的每一个角落。

d)**更换地图模块：**用户可以对现有的地图进行更换，拓展自己的 app 程序。

e)**智能提醒模块：**真实化地图的各种应用场景，采取智能提醒的模块，用户点击某一点，可以提供语音输出。（未完善）

f)**高级搜索模块：**标记添加完后，在地图上很难进行查询，本程序提供了搜索功能，在文本框中输入搜索关键字，单击“搜索”按钮，可按标记名称进行搜索，单击“高级”按钮，打开“高级搜索”窗口

5 核心技术以及算法

地图缩放算法

```
package com.mwq.map.tool;
import java.awt.Image;
import java.awt.image.BufferedImage;
import java.io.IOException;
import javax.imageio.ImageIO;
import javax.swing.ImageIcon;
/**
   *圆梦北斗小队
   * @author Administrator
   */
public class MapProcessor {
private BufferedImage map;// 地图对象
private int viewportWidth;// 地图显示区的宽度
private int viewportHeight;// 地图显示区的高度
private int showCenterX;// 地图视觉中心的水平坐标
private int showCenterY;// 地图视觉中心的垂直坐标
private int cutMapWidth;// 截取地图的宽度
private int cutMapHeight;// 截取地图的高度
private float scale;// 缩放比例
public MapProcessor(String mapPath, int viewportWidth, int viewportHeight,
int scale) {
this.viewportWidth = viewportWidth;
this.viewportHeight = viewportHeight;
replaceMap(mapPath);
dealWithScale(scale);
dealWithCutMapSize();
}
public void replaceMap(String mapPath) {
try {
    map = ImageIO.read(MapProcessor.class.getResource(mapPath));
} catch (IOException e) {
    e.printStackTrace();
}
this.showCenterX = map.getWidth() / 2;
this.showCenterY = map.getHeight() / 2;
}
public Image zoom(int width, int height) {
return map.getScaledInstance(width, height, Image.SCALE_DEFAULT);
}
```

```
    private ImageIcon cut() {
        BufferedImage subimage = map.getSubimage(
                showCenterX - cutMapWidth / 2, showCenterY - cutMapHeight / 2,//
    开始截取点的坐标
                cutMapWidth, cutMapHeight);// 截取地图的大小
        return new ImageIcon(subimage.getScaledInstance(
                viewportWidth, viewportHeight, BufferedImage.SCALE_DEFAULT));//
    返回当前显示的地图
    }
    public ImageIcon adjustWindow(int viewportWidth, int viewportHeight) {
        this.viewportWidth = viewportWidth;// 设置视口宽度
        this.viewportHeight = viewportHeight;// 设置视口高度
        dealWithCutMapSize();// 处理需要截取地图的大小
        validateShowCenterX();// 验证地图视觉中心的水平坐标
        validateShowCenterY();// 验证地图视觉中心的垂直坐标
        return this.cut();// 返回当前显示的地图
    }
    public ImageIcon adjustScale(int scale) {
        float forestallScale = this.scale;// 调整前的比例
        dealWithScale(scale);// 处理比例
        dealWithCutMapSize();// 处理需要截取地图的大小
        if (this.scale < forestallScale) {// 比例缩小，地图的视觉中心可能变化
            validateShowCenterX();// 验证地图视觉中心的水平坐标
            validateShowCenterY();// 验证地图视觉中心的垂直坐标
        }
        return this.cut();// 返回当前显示的地图
    }
    public ImageIcon moveOfHorizontal(int distance) {
        if (scale == 0) {// 不缩放
            this.showCenterX += distance;
        } else {
            if (scale > 0) {// 放大
                this.showCenterX += distance / scale;
            } else {// 缩小
                this.showCenterX += distance * -scale;
            }
        }
        validateShowCenterX();// 验证地图视觉中心的水平坐标
        return this.cut();// 返回当前显示的地图
    }
    public ImageIcon moveOfVertical(int distance) {
        if (scale == 0) {// 不缩放
            this.showCenterY += distance;
        } else {
```

```
            if (scale > 0) {// 放大
                this.showCenterY += distance / scale;
            } else {// 缩小
                this.showCenterY += distance * -scale;
            }
        }
        validateShowCenterY();// 验证地图视觉中心的垂直坐标
        return this.cut();// 返回当前显示的地图
    }
    public void revert() {
        this.showCenterX = map.getWidth() / 2;// 设置地图视觉中心的水平坐标
        this.showCenterY = map.getHeight() / 2;// 设置地图视觉中心的垂直坐标
    }
    public void adjustShowCenter(int x, int y) {
        this.showCenterX = x;
        this.showCenterY = y;
        validateShowCenterX();
        validateShowCenterY();
    }
    private void dealWithScale(int scale) {
        if (scale == 0) {// 不缩放
            this.scale = 0;
        } else {
            this.scale = scale / 100f;
            if (scale > 0) {// 放大
                this.scale += 1;
            } else {// 缩小
                this.scale -= 1;
            }
        }
    }
    private void validateShowCenterX() {
        int w = cutMapWidth / 2;
        if (cutMapWidth % 2 != 0) {
            w += 1;
        }
        if (showCenterX < w) {// 从地图左边缘开始截取
            showCenterX = w;
        } else {
            if (showCenterX > map.getWidth() - w) {// 截取至地图右边缘
                showCenterX = map.getWidth() - w;
            }
        }
    }
```

```
private void validateShowCenterY() {
    int h = cutMapHeight / 2;
    if (cutMapHeight % 2 != 0) {
        h += 1;
    }
    if (showCenterY < h) {// 从地图上边缘开始截取
        showCenterY = h;
    } else {
        if (showCenterY > map.getHeight() - h) {// 截取至地图下边缘
            showCenterY = map.getHeight() - h;
        }
    }
}
private void dealWithCutMapSize() {
    if (scale == 0) {// 不缩放
        cutMapWidth = viewportWidth;
        cutMapHeight = viewportHeight;
    } else {
        if (scale > 0) {// 放大
            cutMapWidth = (int) (viewportWidth / scale);
            cutMapHeight = (int) (viewportHeight / scale);
        } else {// 缩小
            cutMapWidth = (int) (viewportWidth * -scale);
            cutMapHeight = (int) (viewportHeight * -scale);
        }
    }
}
public BufferedImage getMap() {
    return map;
}
public int getShowCenterX() {
    return showCenterX;
}
public int getShowCenterY() {
    return showCenterY;
}
public int getCutMapWidth() {
    return cutMapWidth;
}
public int getCutMapHeight() {
    return cutMapHeight;
}
public float getScale() {
    return scale;
```

```
}
public int parseScale(double scale) {
    if (scale == 0) {
        return 0;
    } else {
        if (scale > 0) {
            return (int) ((scale - 1) * 100);
        } else {
            return (int) ((scale + 1) * 100);
        }
    }
}
private int rightClickX;
private int rightClickY;
private int rightClickToMapX;
private int rightClickToMapY;
public void rightClick(int x, int y) {
    rightClickX = rightClickToMapX = x;
    rightClickY = rightClickToMapY = y;
    if (scale != 0) {
        if (scale > 0) {
            rightClickToMapX = (int) (rightClickToMapX / scale);
            rightClickToMapY = (int) (rightClickToMapY / scale);
        } else {
            rightClickToMapX = (int) (rightClickToMapX * -scale);
            rightClickToMapY = (int) (rightClickToMapY * -scale);
        }
    }
    rightClickToMapX += showCenterX - cutMapWidth / 2;
    rightClickToMapY += showCenterY - cutMapHeight / 2;
}
public int getRightClickX() {
    return rightClickX;
}
public int getRightClickY() {
    return rightClickY;
}
public int getRightClickToMapX() {
    return rightClickToMapX;
}
public int getRightClickToMapY() {
    return rightClickToMapY;
}
}
```

6 接 口 设 计

6.1 外部接口

本桌面应用程序共提供了一个主页面，采用了较为简洁的方式，将程序的绝大部分功能直观的展现个给了用户。整体结构继承了 javax.swing 下的 JFrame,JPanel 和 JDialog 等组件，使本应用程序不再依赖于本地平台的 GUI，实现了跨平台的一致性。以下是详细的说明：

JFrame ——允许程序员把其他组件添加到它里面，把它们组织起来，并把它们呈现给用户。为了最大程度地简化组件，在独立于操作系统的 Swing 组件与实际运行这些组件的操作系统之间，JFrame 起着桥梁的作用。JFrame 在本机操作系统中是以窗口的形式注册的，这么做之后，就可以得到许多熟悉的操作系统窗口的特性：最小化/最大化、改变大小、移动。

JPanel 是 Java 图形用户界面(GUI)工具包 swing 中的面板容器类，包含在 javax.swing 包中，是一种轻量级容器，可以加入到 JFrame 窗体中。本应用程序将主页面的 JFrame 容器分为了 leftPanel，adjustMapPanel，rightPanel，searchPanel，searchConditionPanel，searchButtonPanel，managePanel 等几个小容器。

JDialog 它一般是一个临时的窗口，主要用于显示提示信息或接受用户输入。

采用微软团队开发的 Toolkit 工具，时刻保持应用程序处于屏幕的居中位置。

6.2　内部接口

1.数据处理：

采用 derby 开源数据库，使用其内嵌的 JDBC 驱动，把 Derby 嵌入到基于 Java 的应用程序中。默认静态方法加载数据库驱动，并且在数据库不存在的情况下创建数据库

驱动：org.apache.derby.jdbc.EmbeddedDriver

协议：jdbc:derby:db_map

2.地图缩放：

采用类似比例尺的方法实现地图的缩放功能，并且不断的验证地图视觉中心的水平坐标和垂直坐标，防止位置的偏移。

部分属性如下：

```
BufferedImage map;// 地图对象
viewportWidth;// 地图显示区的宽度
viewportHeight;// 地图显示区的高度
showCenterX;// 地图视觉中心的水平坐标
showCenterY;// 地图视觉中心的垂直坐标
cutMapWidth;// 截取地图的宽度
cutMapHeight;// 截取地图的高度
scale;// 缩放比例
```

7 数据结构设计

7.1 数据对象与形成的数据结构

1.数据对象

（1）数据项名：id
　　数据类型：int
（2）数据项名：name
　　数据类型：varchar
（3）数据项名：father_id
　　数据类型：int
（4）数据项名：tb_sign
　　数据类型：varchar
（5）数据项名：sort_id
　　数据类型：int

2.数据结构

数据结构名：数据表
含义说明：记录用户添加标记时所需要的信息
组成：标记名称、所属类型、创建日期、标记说明

表名. data

字段	含义	类型	空值	默认
name	标记名称	int（10）	not null	无
type	所属类型	varchar（30）	not null	无
date	创建日期	varchar（20）		无
state	标记说明	varchar(20)		无

8 使用说明

8.1 安装和初始化

步骤一：把 demo 文件夹拷贝到 Windows 平台的电脑上。
步骤二：将安装配置包导入 eclipse 中。
步骤三：用户自定义类库加入 derby.jar
步骤四：在 eclipse 中运行即可

8.2 软件主要功能的使用说明

运行程序后，打开程序主界面，如图 1 所示。在界面下方可调整地图的显示比例及位置。

图 1　程序主界面

单击“维护类别”按钮，打开“维护类别”窗口，如图 2 所示，在这里可以添加、修改、删除类别信息。

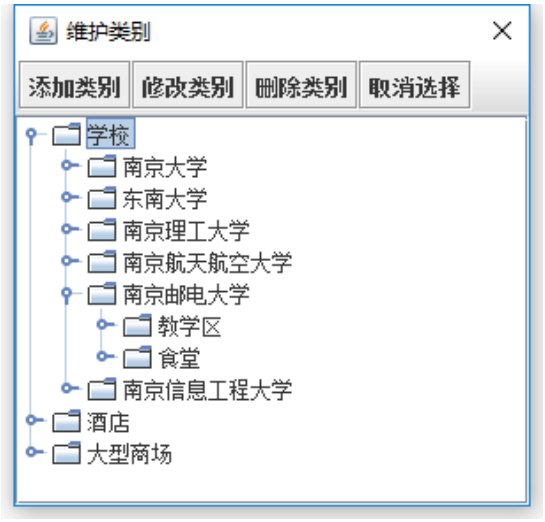


图 2　“维护类别”窗口

添加完类别后，在地图上单击鼠标右键，弹出如图 3 所示的菜单，选择“创建标记”命令，打开“创建标记”窗口，如图 4 所示。

图 3　创建标记

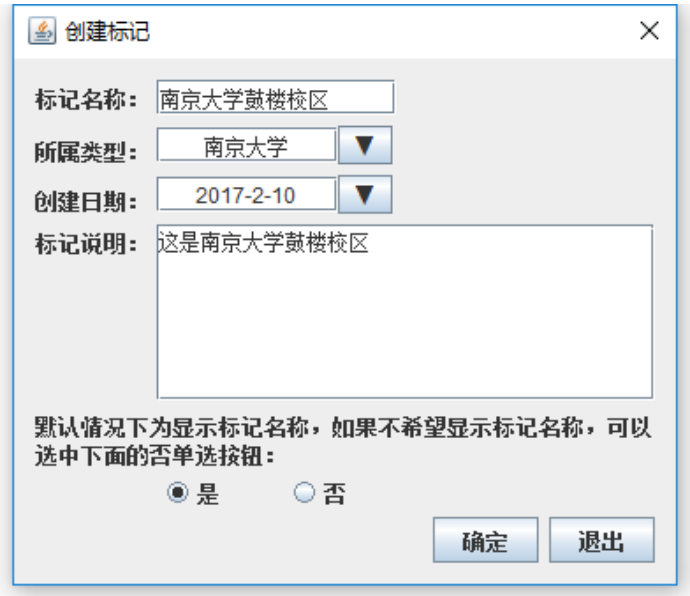


图 4　创建标记窗口

标记创建完后，在地图上即可查看到，如图 5 所示。

图 5　创建的新标记

右键单击标记圆点处，弹出如图 6 所示的菜单，可以查看、修改及删除标记。

图 6　查看、修改及删除标记菜单

标记添加完后，在地图上很难进行查询，本程序提供了搜索功能，在文本框中输入搜索关键字，单击“搜索”按钮，可按标记名称进行搜索，单击“高级”按钮，打开“高级搜索”窗口，如图 7 所示。

图 7（1）　普通搜索窗口

图 7（2）　高级搜索窗口

搜索到的标记会在列表中显示，双击标记名称，可快速在地图上定位到该标记位置，如图 8 所示。

图 8　定位标记位置

单击“更换地图”按钮，弹出“友情提示”窗口，如图 9 所示，在这里选择“是”或者“否”，可进行地图的更换。

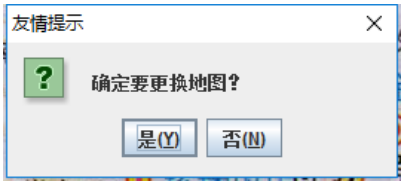


图 9 友情提示

选择“是”按钮，会跳出更换地图选择窗口，如图 10 所示，选择相应的本地图片可将地图更换。

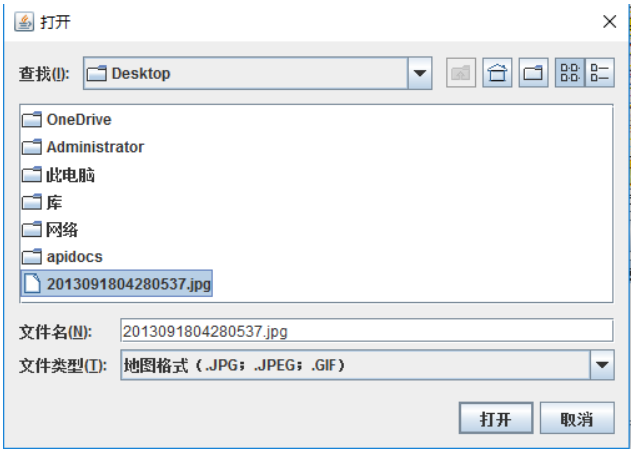


图 10 更换地图选择窗

更换后的地图，如图 1.11 所示

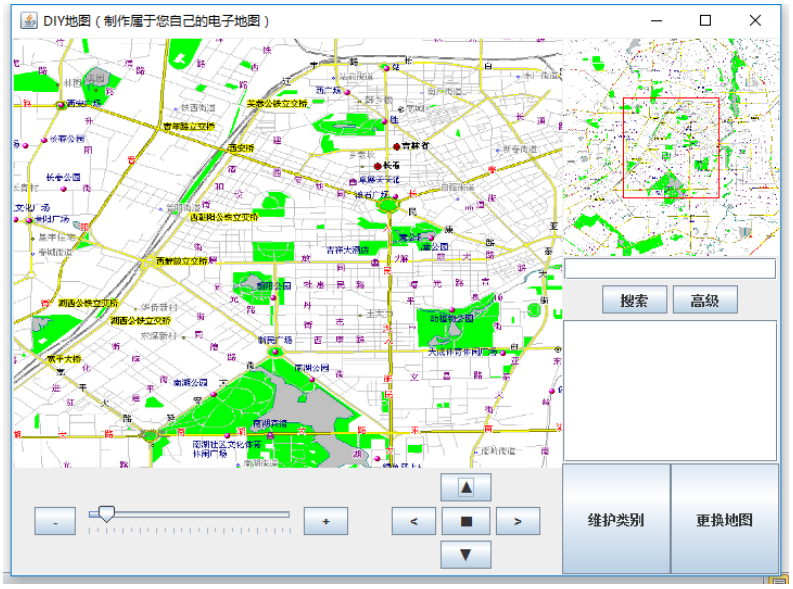


图 11 更换后的地图

9 结 束 语

本系统于 2016 年 12 月 30 日提出构思方案,经过调查讨论,在 2017 年 1 月 17 日正式开始研发.研发人员主要有刘美含、孙鹏飞、代保金、褚洪建，于 2017 年 2 月 18 日基本完成。

在这次项目中,我们首次提出要使用基于 swing 的 java 应用程序,并以 Windows 作为系统平台,经过不断的学习和试验,慢慢总结出很多关于 java 的编程经验以及 derby 数据库的使用方法，使平时课堂学习学位所用。另外,在开发中，我们收集和整理许多关于北斗卫星的信息，北斗应用的多元化不得不让我们为祖国而感到自豪。在这过程中，我们也学会如何自主思考，自主寻求方案以解决难题。

最后，要感谢每一位开发人员全程的投入和辛勤的付出。随着我国科技的发展和进步，我相信我国的航天事业将更加辉煌！