# shapeFile

## 1.1 shapefile简介

ESRI Shapefile, 是美国环境系统研究所公司（ESRI）开发的一种空间数据开放格式，属于一种矢量图形格式，目前，该文件格式已经成为了地理信息软件界的一个开放标准，用于根据几何体对象：点、折线、多边形，实现描述各类地理信息和空间对象的属性。

## 1.2 shapefile组成

Shapefile一般由多个文件共同组成，同时文件必须放在同一文件夹，一般通过命名相互关联。

Shapefile可选的组成文件格式共有11种，有8种为可选，储存包括空间索引，几何体索引等等信息，另外3种为必须文件，主要功能如下：

1) .shp — 图形格式，用于保存元素的几何实体。

2) .shx — 图形索引格式。几何体位置索引，记录每一个几何体在shp文件之中的位置，能够加快向前或向后搜索一个几何体的效率。

3) .dbf — 属性数据格式，以dBase IV的数据表格式存储每个几何形状的属性数据。

## 1.3 打开shapefile

浏览shapefile,我们一般需要下载ESRI的软件ArcMap，这是ArcGis桌面端三大组件之一，属于付费软件，下载破解和安装可参考下面网站：

下载地址：

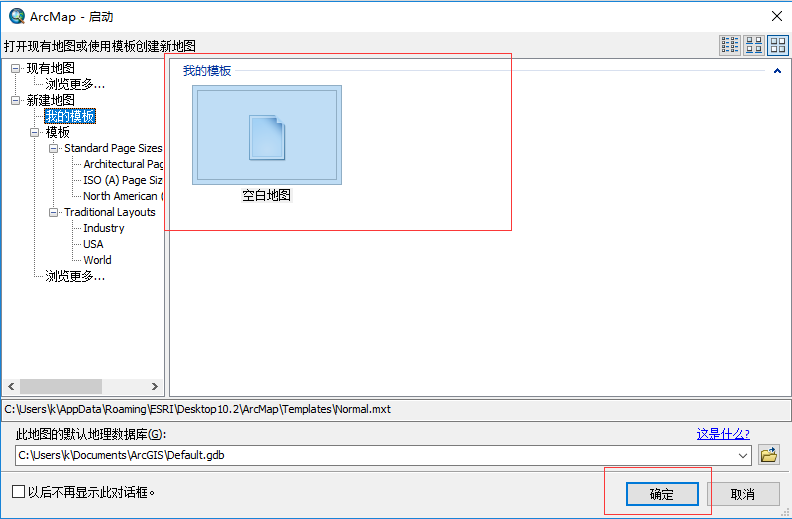
<https://pan.baidu.com/s/1kWWd09L>

密码：hjxy

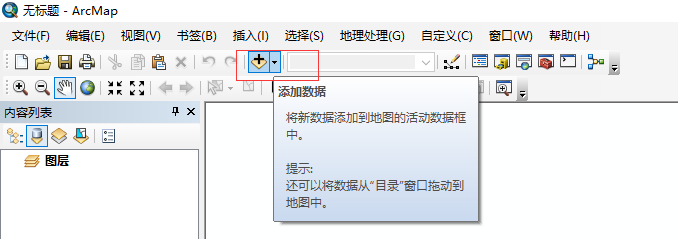
破解教程：

<https://jingyan.baidu.com/article/e73e26c0cb5c1324adb6a791.html>

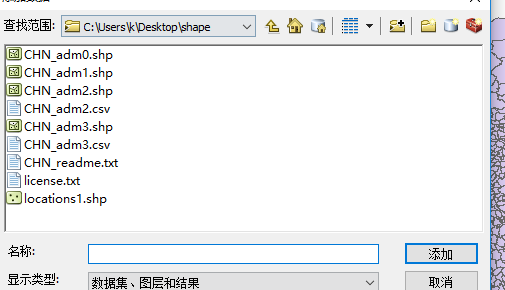
安装完成后，点击arcgis，系统会提醒我们需要先先创建一个arcgis图层，如下图使用最简单的空白模板即可。



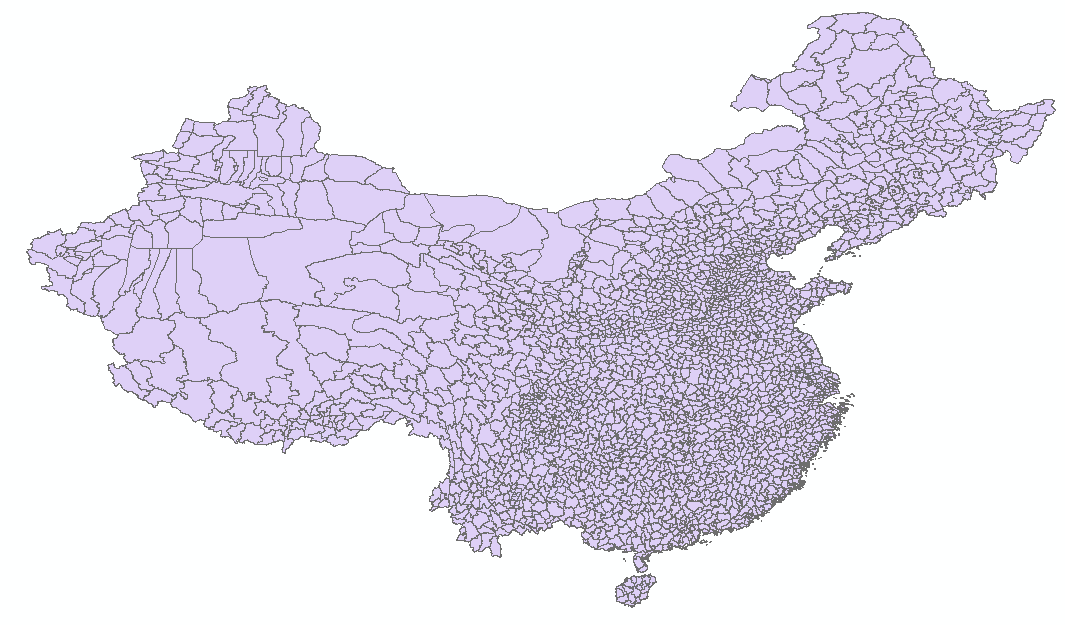
随后，如下图所示点击添加数据



选择需要打开的shapefile主文件.shp

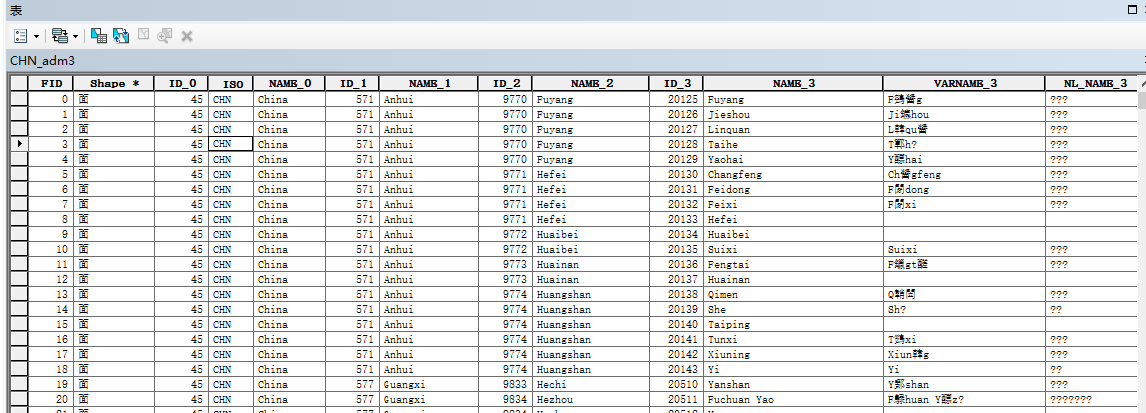


打开shapefile



## 1.4 简单了解shapefile

Shapefile文件通过点、线、面图形化展示地理信息，他可以按比例放大缩小展示。

同时我们在图层右侧选择shape文件，右键打开属性我们可以看到，arcgis将以表格形式把shapefile所包含的信息以表格形式展示出来。

我们的目的是编辑shapfile，其实就相当于我们需要编辑这一张表格，他的字段，数据都是可以编辑的。

# Geotools

2.1 **geotools简介**

编辑shapfile需要如arcGis一样打开shapfile文件，把几个文件的内容属性相互关联起来，进行编辑修改。这是比较繁杂的过程，我们可以把这些过程交给java开源工具geotools，我们只需要调用接口进行编辑即可。

Geotools是从1999年开始编写，至今保持更新，相对的，他的功能较为繁杂，包含多个jar包，项目也较大，需要根据需要对其进行精简。

更多关于geotools的资料可以到官网：<http://geotools.org> 进行了解，同时geotools项目也将在官网进行下载。

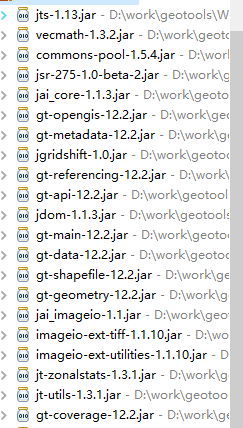
# Geotools读写shapefile

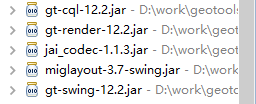
## 3.1 geotools jar包筛选

在官网我们可以下载到geotools整个项目和geotools的各个jar包及文档，考虑我们项目使用java7和稳定性，我们选用v12版本。

我们可以看到geotools已经发展了数十年，他的jar包数量非常庞大，有上百个之多，针对项目内所使用的书写功能，我们需要对官网下载的bin包内的jar包进行筛选。

筛选后jar包引用如下图：





## 读取shapefile内容

代码如下：

**import** java.io.File;

**import** java.nio.charset.Charset;

**import** java.util.Iterator;

**import** java.util.Map;

**import** org.geotools.data.shapefile.ShapefileDataStore;

**import** org.geotools.data.shapefile.ShapefileDataStoreFactory;

**import** org.geotools.data.simple.SimpleFeatureIterator;

**import** org.geotools.data.simple.SimpleFeatureSource;

**import** org.opengis.feature.Property;

**import** org.opengis.feature.simple.SimpleFeature;

**import** org.opengis.feature.type.Name;

**import** org.opengis.feature.type.PropertyDescriptor;

**import** org.opengis.feature.type.PropertyType;

**public** **class** ReadShape {

@SuppressWarnings("unused")

**public** **static** **void** main(String[] args) {

ShapefileDataStoreFactory dataStoreFactory = **new** ShapefileDataStoreFactory();

**try** {

ShapefileDataStore sds = (ShapefileDataStore)dataStoreFactory.createDataStore(**new** File("C:\\Users\\k\\Desktop\\shape\\CHN\_adm2.shp").toURI().toURL());

sds.setCharset(Charset.*forName*("GBK"));

SimpleFeatureSource featureSource = sds.getFeatureSource();

SimpleFeatureIterator itertor = featureSource.getFeatures().features();

**while**(itertor.hasNext()) {

SimpleFeature feature = itertor.next();

Iterator<Property> it = feature.getProperties().iterator();

**while**(it.hasNext()) {

Property pro = it.next();

PropertyType type = pro.getType();

Name name = pro.getName();

PropertyDescriptor descriptor = pro.getDescriptor();

Map<Object, Object> userData = pro.getUserData();

Object value = pro.getValue();

}

}

itertor.close();

} **catch** (Exception e) {

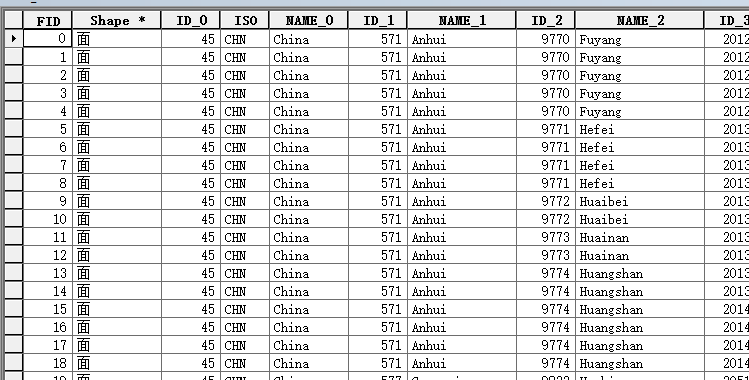
e.printStackTrace();

}

}

}

根据上述代码，参考之前提到的arcgis的属性框（如下图所示）



Geotools读取shapefile是一列一列进行遍历的，从shape开始往后遍历，那么我们只要取出数据就讲可以进行加工。

## 写入shapefile内容

写入shapefile内容，官方推荐是以csv格式文件进行写入，这在官网<http://docs.geotools.org/latest/userguide/tutorial/feature/csv2shp.html> 有较为详尽的介绍。同时，还可以通过下列直接写入固定数据的形式写入内容。由于geotools发展历程长远，版本过多，下述实现只能在v12后的版本可以运行。

Shapefile写入点：

**import** java.io.File;

**import** java.io.IOException;

**import** java.io.Serializable;

**import** java.util.ArrayList;

**import** java.util.HashMap;

**import** java.util.List;

**import** java.util.Map;

**import** org.geotools.data.DefaultTransaction;

**import** org.geotools.data.Transaction;

**import** org.geotools.data.collection.ListFeatureCollection;

**import** org.geotools.data.shapefile.ShapefileDataStore;

**import** org.geotools.data.shapefile.ShapefileDataStoreFactory;

**import** org.geotools.data.simple.SimpleFeatureCollection;

**import** org.geotools.data.simple.SimpleFeatureSource;

**import** org.geotools.data.simple.SimpleFeatureStore;

**import** org.geotools.feature.simple.SimpleFeatureBuilder;

**import** org.geotools.feature.simple.SimpleFeatureTypeBuilder;

**import** org.geotools.geometry.jts.JTSFactoryFinder;

**import** org.geotools.referencing.crs.DefaultGeographicCRS;

**import** org.geotools.swing.data.JFileDataStoreChooser;

**import** org.opengis.feature.simple.SimpleFeature;

**import** org.opengis.feature.simple.SimpleFeatureType;

**import** com.vividsolutions.jts.geom.Coordinate;

**import** com.vividsolutions.jts.geom.GeometryFactory;

**import** com.vividsolutions.jts.geom.Point;

**public** **class** SimpleWrite {

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

SimpleFeatureTypeBuilder builder = **new** SimpleFeatureTypeBuilder();

builder.setName("Location");

builder.setCRS(DefaultGeographicCRS.*WGS84*); // <-坐标参考系统

// 往订单中填加属性

builder.add("the\_geom", Point.**class**);

builder.length(15).add("Name", String.**class**); // <- 15名称字段

builder.add("number",Integer.**class**);

// 构建类型

**final** SimpleFeatureType TYPE = builder.buildFeatureType();

List<SimpleFeature> features = **new** ArrayList<>();

/\*

\* GeometryFactory将用于创建每个要素的几何属性，

\*使用Point对象作为位置。

\*/

GeometryFactory geometryFactory = JTSFactoryFinder.*getGeometryFactory*();

SimpleFeatureBuilder featureBuilder = **new** SimpleFeatureBuilder(TYPE);

Point point = geometryFactory.createPoint(**new** Coordinate(116.46, 39.92));

featureBuilder.add(point);

featureBuilder.add("test");

featureBuilder.add(22);

SimpleFeature feature = featureBuilder.buildFeature(**null**);

features.add(feature);

/\*

\*获取输出文件名并创建新的shapefile

\*/

String path = "D://test//";

String newPath =path+ "11.shp";

JFileDataStoreChooser chooser = **new** JFileDataStoreChooser("shp");

chooser.setDialogTitle("Save shapefile");

chooser.setSelectedFile(**new** File(newPath));

File newFile = chooser.getSelectedFile();

ShapefileDataStoreFactory dataStoreFactory = **new** ShapefileDataStoreFactory();

Map<String, Serializable> params = **new** HashMap<>();

params.put("url", newFile.toURI().toURL());

params.put("create spatial index", Boolean.*TRUE*);

ShapefileDataStore newDataStore = (ShapefileDataStore) dataStoreFactory.createNewDataStore(params);

/\*

\* TYPE用作模板来描述文件内容

\*/

newDataStore.createSchema(TYPE);

/\*

\* 将特征写入shapefile

\*/

Transaction transaction = **new** DefaultTransaction("create");

String typeName = newDataStore.getTypeNames()[0];

SimpleFeatureSource featureSource = newDataStore.getFeatureSource(typeName);

SimpleFeatureType SHAPE\_TYPE = featureSource.getSchema();

/\*

\* Shapefile格式有一些限制：

\* - “the\_geom”始终是第一个，用于几何属性名称

\* - “the\_geom”必须是Point，MultiPoint，MuiltiLineString，MultiPolygon类型

\* - 属性名称的长度是有限的

\* - 不支持所有数据类型Timestamp表示为Date）

\*

\*每个数据存储都有不同的限制，因此请检查生成的SimpleFeatureType。

\*/

System.*out*.println("SHAPE:"+SHAPE\_TYPE);

**if** (featureSource **instanceof** SimpleFeatureStore) {

SimpleFeatureStore featureStore = (SimpleFeatureStore) featureSource;

/\*

\* SimpleFeatureStore有一个方法来添加来自

\* SimpleFeatureCollection对象的特征，所以我们使用ListFeatureCollection

\*类来包装我们的特征列表。

\*/

SimpleFeatureCollection collection = **new** ListFeatureCollection(TYPE, features);

featureStore.setTransaction(transaction);

**try** {

featureStore.addFeatures(collection);

transaction.commit();

} **catch** (Exception problem) {

problem.printStackTrace();

transaction.rollback();

} **finally** {

transaction.close();

}

}

}

}

# 4. CAD

CAD原指辅助设计人员进行图形设计的软件，由于AutoCAD的普及，现在通常指autoCAD软件生成的文件。这类文件包含三种格式：

（1）DWG：是原始图纸格式，包含了图纸所有的信息，Autodesk公司出于安全考虑没有给出详细的格式说明。

（2）DWF：比DWG文件小很多，用于其他用户浏览，添加备注等，不能编辑。

（3）DXF：是用于和其他CAD系统交换数据的文件格式，也是AutoCAD用于外部信息交互的格式，分二进制和ASCII格式，AutoCAD的帮助里包含了DXF文件的详细描述。

同时需要知道的是，以上三个格式，只需要在保存时选定格式，以上三种文件格式可以实现互转的。

# 5 Ycad

## 5.1 ycad简介

Autocad使用 c++进行开发且历史悠久，在这一领域使用java进行二次开发的案例几乎没有。针对cad文件的三种格式，要实现对于cad内容解析，在java领域只能针对dxf这一种文件格式根据adobe公司官方的网站上的dxf参考指南：<http://docs.autodesk.com/ACD/2011/CHS/filesDXF/WSfacf1429558a55de185c428100849a0ab7-5f35.htm> 上提供的编码解读，进行内容的解析。

这一块的工作，我们可以交给ycad进行。

Ycad项目可以在 <https://sourceforge.net/projects/ycad/> 了解与下载，ycad是历史悠久的项目，但是缺乏入门的文档，且长时间不曾维护，针对我们的使用我们需要对他的源码进行简单的解读，以便于修改并融入我们的项目。

## 5.2 dxf简单说明

在进行ycad源码解读前，我们需要对dxf这一文件格式进行简单的说明。在上述我们已经说过，更详尽的解读可以参考adobe在官网提供的dxf参考指南，我们这里只做简单入门对dxf的说明。

首先，我们需要知道的是，dxf可以通过文本编辑器，如notepad，vscode等等打开，比较推荐使用vscode这一类稍重量级的文本编辑器打开，因为dxf文件内容极多，普通文本编辑器容易卡死。

如下图使用vscode打开dxf后，我们就可以进行解读，需要知道的是，ycad的运行思路也正是建立在这一种解读之上。



下面，我们将对dxf文件简单介绍：

**1、 Dxf文件中的最小单元**

Dxf文件的最小单元是”码---值”. 码和值各占一行.  “码”代表某种”意思”;  “值”代表这个”意思”的”值”。

例如

2                                            //2 是”码”, 这个”码”在这里的意思代表”段名”

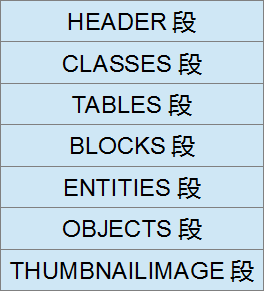
HEARER                               //这里, “段名”的值是” HEARER”

同一个”码”在不同的上下文代表不同的意思

.

**2、Dxf文件的整体结构**

dxf文件就是由这7个段组成：



HEADER，包含了一系列和图纸相关的变量信息，每个变量由给出变量名称的组码 9 指定，其后是提供变量值的组。比如AutoCAD版本，坐标系的最小、最大值等。

CLASSES，包含了在BLOCKS，ENTITIES和OBJECTS段用到的类的定义，比如LWPOLYLINE

TABLES，包含各种表，比如图层（Layer）、线条类型（LTYPE）等；每个表可以包含多个条目

BLOCKS，包含构成图形中每个块参照的块定义和图形图元，由一系列Entity组成

ENTITIES，包含各种图形对象，也叫图元（Entity），比如点（POINT）、线（LINE），圆（CIRCLE），弧（ARC），多边形（LWPOLYLINE）等，是我们解析的重点

OBJECTS，包括非图形对象的数据，供 AutoLISP 以及 ObjectARX 应用程序所使用

THUMBNAILIMAGE，包含DXF文件的缩略图

**3、dxf文件中的字符格式结构**

**0                                            //0--SECTION码值代表"段开始"**

**SECTION**

**2                                            //2--段名码值代表"段名";这里是HEADER段**

**HEADER                              //例如2--HEADER (HEADER段);2--BLOCKS(BLOCKS段)**

**码                                          //各段的不同的"码--值"**

**值**

**码**

**值**

**...**

**...**

**0                                            //0--ENDSEC码值代表"段结束"**

**ENDSEC**

**0                                            //0--SECTION码值代表"段开始"**

**SECTION**

**2                                            //2--段名码值代表"段名";这里是CLASSES段**

**CLASSES                              //例如2--HEADER (HEADER段);2--BLOCKS(BLOCKS段)**

**码                                          //各段的不同的"码--值"**

**值**

**码**

**值**

**...**

**...**

**0                                            //0--ENDSEC码值代表"段结束"**

**ENDSEC**

**...**

**......**

**其他段**

**......**

**0                                            //0--EOF码值代表"整个文件结束"**

**EOF**

**其中每个段在文件中的结构都是这样**:

0                                            // 0--SECTION码值代表"段开始"

SECTION

2                                            // 2--段名码值代表"段名";这里是HEADER段

HEADER                               // 例如2--HEADER (HEADER段); 2--BLOCKS(BLOCKS段)

码                                          //各段的不同的"码--值"

值

码

值

...

...

0                                            // 0--ENDSEC码值代表"段结束"

ENDSEC

**4、ENTITIES 段**

ENTITIES 段包含图形数据. 例如LINE 直线, ARC圆弧

（1）ENTITIES段中图元数据说明

看图元数据(LINE和ARC)在文件中的格式

0                                                               // 0--图元码值代表"新图元开始"

LINE                                                         //0--LINE代表直线

5                                                               // 5--值代表"句柄"

D640

8                                                               // 8--值代表"图层名"

BOT-AL-FRAME

6                                                               //具体其他的"码值"意思可参考DXF的说明文档,因为太多了

CONTINUOUS

10

2232.4029275168591

20

1056.158312627997

30

0.0

11

2256.9029153003012

21

1056.158315364697

31

0.0

0                                                               //0--图元码值代表"新图元开始"

ARC                                                         //0--ARC代表圆弧

5

D641

8

BOT-AL-FRAME

6

CONTINUOUS

10

1870.4529275168611

20

1059.908312627997

30

0.0

40

3.750000000000032

50

138.18968510434721

51

270.0

（2）LINE和ARC图元

LINE图元中, 包含两个端点的坐标值; ARC图元中包含圆心, 半径, 起始角度等. 每中图元都有它对应的”码—值”数据.

5、其他段

和ENTITIES段一样, 其他段也有它的数据和结构, 都可以参考DXF的说明文档来了解. 但是无论有多少段,这些段的结构式怎样的, 最小单元都是由”码—值”的形式组成的。

Dxf的文件格式大致如上述所示，但是更详尽的码和值还需要参考官方文档的解读指南。

## 5.3 ycad简单源码解读

简单对有ycad项目进行阅读后，我们不难看出ycad属于桌面级应用，读取cad后其同时还使用gui实现的回显。我们并不需要gui这一块，所以首先，我们需要定位到项目的启动：

在com.ysystems.ycad.app.ycadv包下Ycadv类内，我们可以找到整个程序的入口main函数，如下图所示：

**public** **static** **void** main(String argv[]){

String[]filename = **new** String[1];

filename[0] = "src=C:/Users/k/Desktop/cadd/color.dxf";

// Instantiate the Applet (processes argv)

Ycadv vapplet = **new** Ycadv(filename);

// Create a Frame for the Applet

YcadvFrame vframe = **new** YcadvFrame("Ycadv", vapplet);

// Add the Applet to the Frame (Frame's use BorderLayout)

vframe.add("Center", vapplet);

// Init and start the Applet

// **TODO**: bail if init errors

vapplet.init();

// Resize the Frame according to user parameters

//d System.out.println("++Ycadv:w=" + vapplet.getParamWidth() + ",h=" + vapplet.getParamHeight());

vframe.~~resize~~(vapplet.getParamWidth(), vapplet.getParamHeight()); // calls Component.resize(int width, int height)

// Show and start the Applet

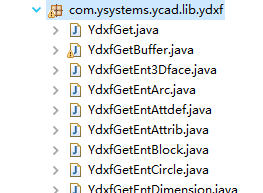
vframe.~~show~~();

vapplet.start();

}

我们需要把针对gui的vframe这一块的启动和展示注释或者删除，只留下对文件解析的过程即可。

接着往下阅读源码，我们将会发现，对文件的解析工作，ycad实现于com.ysystems.ycad.lib.ydxf 包下的各种各样的get类如下图所示：



通过YdxfGet这一个入口，这些类通过码进行区分，也通过码根据dxf解读规则进行解读。

## 5.4 根据需求修改源码

Ycad对于dxf的解析，首先经过：YdxfGet这个类，在这个类里通过get()方法，判断出现在读入的对象属于header、table、block还是entities并创建对应的对象，并根据对应对象的get()进行更深入的解析。

我们已经知道，我们需要的entities中的数据，而我们需要读取的内容在cad中几乎都是以折线形式展示，那么要获取该类数据，我们只需要在知道该类线的特征，例如实线、虚线、颜色,或者通过标注名称，修改该类源码或其他方式定位到该线段，就可以获取到这一条线段的数据。

# 6. 存在的问题

1. 如果ycad与shapeFile的文件交互，目前点、线信息是可行的，但是当涉及到多边形，需要考虑cad与shapfile的作图规则异同，目前不能解决。

2. ycad可以实现对dxf内容阅读、修改，但是如果生成一个dxf还不能实现。