Shapefile 格式说明及读写代码示例

Shape files 数据说明

Shape files 是 ESRI 提供的一种矢量数据格式，它没有拓扑信息，一个 Shape files 由一组文件组成，其中必要的基本文件包括坐标文件（ .shp ）、索引文件（ .shx ）和属性文件（ .dbf ）三个文件。

坐标文件的结构说明

坐标文件 (.shp) 用于记录空间坐标信息。它由头文件和实体信息两部分构成（如图 2.1 所示）。

坐标文件的文件头

坐标文件的文件头是一个长度固定 (100 bytes) 的记录段，一共有 9 个 int 型和 7 个 double 型数据，主要记录内容见表 2.2 。

文件头	
记录头	记录内容
记录头	记录内容
记录头	记录内容
记录头	记录内容

图 2.1 坐标文件的结构

起始位置	名称	数值	类型	位序

0	File Code	9994	Integer	big
4	Unused	0	Integer	big
8	Unused	0	Integer	big
12	Unused	0	Integer	big
16	Unused	0	Integer	big
20	Unused	0	Integer	big
24	文件长度	文件的实际长度	Integer	big
28	版本号	1000	Integer	Little
32	几何类型	表示这个 Shapefile 文件所记录的空间数据的几何类型	Integer	Little
36	Xmin	空间数据所占空间范围的 X 方向最小值	Double	Little
44	Ymin	空间数据所占空间范围的 Y 方向最小值	Double	Little
52	Xmax	空间数据所占空间范围的 X 方向最大值	Double	Little
60	Ymax	空间数据所占空间范围的 Y 方向最大值	Double	Little
68*	Zmin	空间数据所占空间范围的 Z	Double	Little

		方向最小值		
76*	Zmax	空间数据所占空间范围的 Z 方向最大值	Double	Little
84*	Mmin	最小 Measure 值	Double	Little
92*	Mmax	最大 Measure 值	Double	Little

表 2.2shapefiles 头文件表

注：最后 4 个加星号特别标示的四个数据只有当这个 Shapefile 文件包含 Z 方向 坐标或者具有 Measure 值时才有值，否则为 0.0 。所谓 Measure 值，是用于存储需要的 附加数据，可以用来记录各种数据，例如权值、道路长度等信息。

位序

细心的读者会注意到表 2.2 中的数值的位序有 Little 和 big 的区别，对于位序是 big 的数据我们在读取时要小心。通常，数据的位序都是 Little ，但在有些情况下可能会是 big ，二者的区别在于它们位序的顺序相反。一个位序为 big 的数据，如果我们想得到它的真实数值，需要将它的位序转换成 Little 即可。转换原理非常简单，就是交换字节顺序，下面是作者实现的在两者间进行转换的程序，代码如下：

```
// 位序转换程序
unsigned long OnChangeByteOrder (int indata)
{
    char ss[8];
```

```
    char ee[8];
    unsigned long val = unsigned long(indata);
    _ultoa( val, ss, 16 );// 将十六进制的数 (val) 转到一个字
符串 (ss) 中
    int i;
    int length=strlen(ss);
    if(length!=8)
    {
        for(i=0;i<8-length;i++)
            ee[i]='0';
        for(i=0;i<length;i++)
            ee[i+8-length]=ss[i];
        for(i=0;i<8;i++)
            ss[i]=ee[i];
  }
    ////****** 进行倒序
    int t;
    t     =ss[0];
    ss[0]      =ss[6];
    ss[6]      =t;
    t     =ss[1];
    ss[1]      =ss[7];
```

```
    ss[7]      =t;
    t     =ss[2];
    ss[2]      =ss[4];
    ss[4]      =t;
    t     =ss[3];
    ss[3]      =ss[5];
    ss[5]      =t;
  ////******
    //****** 将存有十六进制数 (val) 的字符串 (ss) 中的十六进制数转成十进制数
    int value=0;
    for(i=0;i<8;i++)
    {
        int k;
        CString mass;
        mass=ss[i];
        if(ss[i]=='a' ||
           ss[i]=='b' ||
           ss[i]=='c' ||
           ss[i]=='d' ||
           ss[i]=='e' ||
           ss[i]=='f')
```

```
            k=10+ss[i]-'a';
        else
            sscanf(mass,"%d",&k);
        value=value+int(k*pow(16,7-i));
    }
    return (value);
}
```

Shapefile 文件支持的几何类型（ ShapeType ）

Shapefile 文件所支持的几何类型如表 2.3 所示：

编号	几何类型
0	Null Shape （表示这个 Shapefile 文件不含坐标）
1	Point （表示 Shapefile 文件记录的是点状目标，但不是多点）
3	PolyLine（表示 Shapefile 文件记录的是线状目标）
5	Polygon （表示 Shapefile 文件记录的是面状目标）
8	MultiPoint（表示 Shapefile 文件记录的是多点，即点集合）
11	PointZ （表示 Shapefile 文件记录的是三

	维点状目标）
13	PolyLineZ （表示 Shapefile 文件记录的是三维线状目标）
15	PolygonZ （表示 Shapefile 文件记录的是三维面状目标）
18	MultiPointZ （表示 Shapefile 文件记录的是三维点集合目标）
21	PointM （表示含有 Measure 值的点状目标）
23	PolyLineM （表示含有 Measure 值的线状目标）
25	PolygonM （表示含有 Measure 值的面状目标）
28	MultiPointM （表示含有 Measure 值的多点目标）
31	MultiPatch （表示复合目标）

表 2.3shapefiles 文件支持的几何类型

对于一个不是记录 Null Shape 类型的 Shapefile 文件，它所记录的空间目标的几何类型必须一致，不能在一个 Shapefile 文件中同时记录两种不同类型的几何目标。

读取坐标文件（ .shp ）的文件头的代码 如下：

void OnReadShp （ CString ShpFileName ）

```
{
    FILE*   m_ShpFile_fp;       //****Shp 文件指针
// 打开坐标文件
    if((m_ShpFile_fp=fopen(ShpFileName,"rb"))==NULL)
    {
        return;
    }
    // 读取坐标文件头的内容开始
    int FileCode;
    int Unused;
    int FileLength;
    int Version;
    int ShapeType;
    double Xmin;
    double Ymin;
    double Xmax;
    double Ymax;
    double Zmin;
    double Zmax;
    double Mmin;
    double Mmax;
    fread(&FileCode,    sizeof(int),   1,m_ShpFile_fp);
```

```
        FileCode = OnChangeByteOrder(FileCode);
        for(i=0;i<5;i++)
              fread(&Unused,sizeof(int),    1,m_ShpFile_fp);
      fread(&FileLength,    sizeof(int),    1,m_ShpFile_fp);
       FileLength       = OnChangeByteOrder(FileLength);
        fread(&Version,          sizeof(int),    1,m_ShpFile_fp)
;
        fread(&ShapeType,     sizeof(int),    1,m_ShpFile_fp);
        fread(&Xmin,          sizeof(double),1,m_ShpFile_fp);
        fread(&Ymin,          sizeof(double),1,m_ShpFile_fp);
        fread(&Xmax,          sizeof(double),1,m_ShpFile_fp);
        fread(&Ymax,          sizeof(double),1,m_ShpFile_fp);
        fread(&Zmin,          sizeof(double),1,m_ShpFile_fp);
        fread(&Zmax,          sizeof(double),1,m_ShpFile_fp);
        fread(&Mmin,          sizeof(double),1,m_ShpFile_fp);
        fread(&Mmax,          sizeof(double),1,m_ShpFile_fp);
        // 读取坐标文件头的内容结束
        // 根据几何类型读取实体信息
}
```

实体信息的内容

实体信息负责记录坐标信息，它以记录段为基本单位，每一个记录段记录一个地理实体目标的坐标信息，每个记录段分为记录头和记录内容两部分。

记录头的内容包括记录号（ Record Number ）和坐标记录长度 (Content Length) 两个记录项。它们的位序都是 big 。记录号（ Record Number ）和坐标记录长度 (Content Length) 两个记录项都是 int 型，并且 shapefile 文件中的记录号都是从 1 开始的。

记录内容包括目标的几何类型（ ShapeType ）和具体的坐标记录 (X 、 Y) ，记录内容因要素几何类型的不同其具体的内容及格式都有所不同。下面分别介绍点状目标（ Point ）、线状目标（ PolyLine ）和面状目标（ Polygon ）三种几何类型的 .shp 文件的记录内容：

点状目标

shapefile 中的点状目标由一对 X 、 Y 坐标构成，坐标值为双精度型（ double ）。点状目标的记录内容如表 2.4 ：

记录项	数值	数据类型	长度	个数	位序
几何类型（ ShapeType ）	1 （表示点状目标）	int 型	4	1	Little
X 方向坐标	X 方向坐标值	double 型	8	1	Little

Y 方向坐标	Y 方向坐标值	double 型	8	1	Little

下面是 **读取点状目标的记录内容的代码：**

```
OnReadPointShp(CString ShpFileName)
{
    // 打开坐标文件
    ……
    // 读取坐标文件头的内容开始
    ……
    // 读取点状目标的实体信息
    int RecordNumber;
    int ContentLength;
    int num   =0;
    while((fread(&RecordNumber,   sizeof(int),   1,ShpF
ile_fp)!=0))
    {
        num++;
         fread(&ContentLength,sizeof(int),   1,ShpFile_fp)
;
         RecordNumber      =
OnChangeByteOrder(RecordNumber);
```

```
            ContentLength        =
OnChangeByteOrder(ContentLength);
            int shapeType;
            double x;
      double y;
            fread(&shapeType, sizeof(int),    1,ShpFile_fp);
            fread(&x, sizeof(double),    1,ShpFile_fp);
            fread(&y, sizeof(double),    1,ShpFile_fp);
}
}
```

线状目标

shapefile 中的线状目标是由一系列点坐标串构成，一个线目标可能包括多个子线段，子线段之间可以是相离的，同时子线段之间也可以相交。 Shapefile 允许出现多个坐标完全相同的连续点，当读取文件时一定要注意这种情况,但是不允许出现某个退化的、长度为 0 的子线段出现。线状目标的记录内容如表 2.5 ：

记录项	数值	数据类型	长度	个数	位序
几何类型（ ShapeType ）	3（表示线状目标）	int 型	4	1	Littl e

坐标范围（ Box ）	表示当前线目标的坐标范围	double 型	32	4	Little
子线段个数（ NumParts ）	表示构成当前线目标的子线段的个数	int 型	4	1	Little
坐标点数（ NumPoints ）	表示构成当前线目标所包含的坐标点	int 型	4	1	Little

	个数				
Parts 数组	记录了每个子线段的坐标在 Points 数组中的起始位置	int 型	4×NumParts	NumParts	Littl e
Points 数组	记录了所有的坐标信息	Point 型	根据点个数来确定	NumPoints	Littl e

表 2.5 线状目标的记录内容

具体的数据结构如下：

```
PolyLine
{
```

Double[4] Box // 当前线状目标的坐标范围

Integer NumParts // 当前线目标所包含的子线段的个数

Integer NumPoints // 当前线目标所包含的顶点个数

Integer[NumParts] Parts // 每个子线段的第一个坐标点在 Points 的位置

Point[NumPoints] Points // 记录所有坐标点的数组

}

这些记录项的具体含义如下：

Box 记录了当前的线目标的坐标范围，它是一个 double 型的数组，按照 Xmin 、 Ymin 、 Xmax 、 Ymax 的顺序记录了坐标范围；

NumParts 记录了当前线目标所包含的子线段的个数；

NumPoints 记录了当前线目标的坐标点总数；

Parts 记录了每个子线段的第一个坐标点在坐标数组 points 中的位置，以便读取数据；

Points 是用于存放当前线目标的 X 、 Y 坐标的数组。

下面是读取线状目标的记录内容的代码：

OnReadLineShp(CString ShpFileName)

```
{
    // 打开坐标文件
    ……
    // 读取坐标文件头的内容开始
    ……
    // 读取线状目标的实体信息
    int RecordNumber;
    int ContentLength;
    int num   =0;
    while((fread(&RecordNumber,   sizeof(int),   1,ShpF
ile_fp)!=0))
{
        fread(&ContentLength,sizeof(int),   1,ShpFile_fp)
;
        RecordNumber      = OnChangeByteOrder
(RecordNumber);
        ContentLength       = OnChangeByteOrder
(ContentLength);
        int shapeType;
        double Box[4];
        int NumParts;
        int NumPoints;
```

```
            int *Parts;
            fread(&shapeType,    sizeof(int),    1,ShpFile_fp);
    // 读 Box
            for(i=0;i<4;i++)
                  fread(Box+i,     sizeof(double),1,ShpFile_fp)
;   // 读 NumParts 和 NumPoints
            fread(&NumParts,     sizeof(int),    1,ShpFile_fp)
;
            fread(&NumPoints,    sizeof(int),    1,ShpFile_fp);
    // 读 Parts 和 Points
            Parts   = new int[NumParts];
            for(i=0;i<NumParts;i++)
fread(Parts+i,    sizeof(int),    1,ShpFile_fp);
            int pointNum;
            for(i=0;i<NumParts;i++)
            {
                  if(i!=NumParts-1)
                        pointNum       =Parts[i+1]-Parts[i];
                  else
                        pointNum       =NumPoints-Parts[i];
                  double *PointsX;
                  double *PointsY;
```

```
                PointsX =new double[pointNum];
                PointsY =new double[pointNum];
            for(j=0;j<pointNum;j++)
                {
                    fread(PointsX+j,
sizeof(double),1,ShpFile_fp);
                    fread(PointsY+j,
sizeof(double),1,ShpFile_fp);
                }
                delete[] PointsX;
                delete[] PointsY;
            }
            delete[] Parts;
        }
}
```

面状目标

shapefile 中的面状目标是由多个子环构成，每个子环是由至少四个顶点构成的封闭的、无自相交现象的环。对于含有岛的多边形，构成它的环有内外环之分，每个环的顶点的排列顺序或者方向说明了这个环到底是内环还是外环。一个内环的顶点是按照逆时针顺序排列

的；而对于外环，它的顶点排列顺序是顺时针方向。如果一个多边形只由一个环构成，那么它的顶点排列顺序肯定是顺时针方向。

每条多边形记录的数据结构与线目标的数据结构完全相同，

```
Polygon
{
Double[4]              Box              // 当前面状目标的坐标范围
Integer                NumParts         // 当前面目标所包含的子环的个数
Integer                NumPoints        // 构成当前面状目标的所有顶点的个数
Integer[NumParts]  Parts                // 每个子环的第一个坐标点在 Points 的位置
Point[NumPoints]   Points               // 记录所有坐标点的数组
}
```

对于一个 shapefile 中的多边形，它必须满足下面三个条件：

构成多边形的每个子环都必须是闭合的，即每个子环的第一个顶点跟最后一个顶点是同一个点；

每个子环在 Points 数组中的排列顺序并不重要，但每个子环的顶点必须按照一定的顺序连续排列；

存储在 shapefile 中的多边形必须是干净的。所谓一个干净的多边形，它必须满足两点：

没有自相交现象。这就要求任何一个子环不能跟其它的子环相交，共线的现 象也将被当作相交。但是允许两个子环的顶点重合；

对于一个不含岛的多边形或者是含岛的多边形的外环，它们的顶点排列顺序必须是顺时针方向；而对于内环，它的排列顺序必须是逆时针方向。所谓的“ 脏多边形 ”就是指顶点排列顺序为顺时针的内环。

图 2.2 中的多边形是一个典型的例子。这个多边形包括一个岛，所有顶点的个数为 8 。 NumParts 等于 2 ， NumPoints 等于 10 。请注意内环（岛）的顶点的排列顺序是逆时针的（如图 2.3 所示）

表 2.6 ：面状目标的记录内容

记录项	数值	数据类型	长度	个数	位序
几何类型（ ShapeType ）	5（表示面状目标）	int 型	4	1	Littl e
坐标范围（ Box ）	表示当前	doubl e 型	32	4	Littl e

	面目标的坐标范围				
子线段个数（NumParts）	表示构成当前面状目标的子环的个数	int 型	4	1	Littl e
坐标点数（NumPoints）	表示构成当前面状目标所包含的坐标点个数	int 型	4	1	Littl e

Parts 数组	记录了每个子环的坐标在 Points 数组中的起始位置	int 型	4×NumParts	NumParts	Little
Points 数组	记录了所有的坐标信息	Point 型	根据点个数来确定	NumPoints	Little

下面是 读取面状目标的记录内容的代码：

```
void OnReadAreaShp(CString ShpFileName)
{
    // 打开坐标文件
    ……
```

```
        // 读取坐标文件头的内容开始
        ……
        // 读取面状目标的实体信息
        int RecordNumber;
        int ContentLength;
        while((fread(&RecordNumber,  sizeof(int),  1,m_S
hpFile_fp)!=0))
        {
            fread(&ContentLength,sizeof(int),  1,m_ShpFile
_fp);
            RecordNumber      = OnChangeByteOrder
(RecordNumber);
            ContentLength      = OnChangeByteOrder
(ContentLength);
            int shapeType;
            double Box[4];
            int NumParts;
            int NumPoints;
            int *Parts;
            fread(&shapeType,  sizeof(int),  1,m_ShpFile
_fp);
            // 读 Box
```

```
            for(i=0;i<4;i++)
          fread(Box+i,      sizeof(double),1,m_ShpFile_fp);
            // 读 NumParts 和 NumPoints
            fread(&NumParts,      sizeof(int),    1,m_ShpFile_
fp);
            fread(&NumPoints,     sizeof(int),    1,m_ShpFile
_fp);
            // 读 Parts 和 Points
            Parts        =new int[NumParts];
            for(i=0;i<NumParts;i++)
                  fread(Parts+i,    sizeof(int),    1,m_ShpFile_
fp);
            int pointNum;
            int xx;
            int yy;
            for(i=0;i<NumParts;i++)
            {
                  if(i!=NumParts-1)
                        pointNum        =Parts[i+1]-Parts[i];
           else
                        pointNum        =NumPoints-Parts[i];
```

```
                double *PointsX;
                double *PointsY;

                PointsX =new double[pointNum];
                PointsY =new double[pointNum];

                for(j=0;j<pointNum;j++)
                {
                    fread(PointsX+j,
sizeof(double),1,m_ShpFile_fp);
                    fread(PointsY+j,
sizeof(double),1,m_ShpFile_fp);
                }
                delete[] PointsX;
                delete[] PointsY;
            }
            delete[] Parts;
        }
}
```

属性文件的结构说明

属性文件 (.dbf) 用于记录属性信息。它是一个标准的 DBF 文件，也是由头文件和实体信息两部分构成。

文件头
记录 1
记录 2
记录 3
记录 4

……

记录 n

属性文件的文件头

其中文件头部分的长度是不定长的，它主要对 DBF 文件作了一些总体说明（见表 2.7 ），其中最主要的是对这个 DBF 文件的记录项的信息进行了详细地描述，比如对每个记录项的名称、数据类型、长度等信息都有具体的说明。

在文件中的位置	内容	说明
0	1 个字节	表示当前的版本信息
1 — 3	3 个字节	表示最近的更新日期，按照 YYMMDD 格式。
4 — 7	1 个 32 位数	文件中的记录条数。
8 —	1 个 16 位	文件头中的字节数。

9	数	
10 — 11	1 个 16 位数	一条记录中的字节长度。
12 — 13	2 个字节	保留字节，用于以后添加新的说明性信息时使用，这里用 0 来填写。
14	1 个字节	表示未完成的操作。
15	1 个字节	dBASE IV 编密码标记。
16 — 27	12 个字节	保留字节，用于多用户处理时使用。
28	1 个字节	DBF 文件的 MDX 标识。在创建一个 DBF 表时，如果使用了 MDX 格式的索引文件，那么 DBF 表的表头中的这个字节就自动被设置了一个标志，当你下次试图重新打开这个 DBF 表的时候，数据引擎会自动识别这个标志，如果此标志为真，则数据引擎将试图打开相应的 MDX 文件。
29	1 个字节	Language driver ID.
30 — 31	2 个字节	保留字节，用于以后添加新的说明性信息时使用，这里用 0 来填写。
32 — X	（ n*32 ）个字节	记录项信息描述数组。 n 表示记录项的个数。这个数组的结构在表 2.8 中有详细的解释。

X ＋ 1	1 个字节	作为记录项终止标识。

表 2.7 属性文件（ .dbf ）的文件头

位置	内容	说明
0 － 10	11 个字节	记录项名称，是 ASCII 码值。
11	1 个字节	记录项的数据类型，是 ASCII 码值。（ B 、 C 、 D 、 G 、 L 、 M 和 N ，具体的解释见表 2.9 ）。
12 － 15	4 个字节	保留字节，用于以后添加新的说明性信息时使用，这里用 0 来填写。
16	1 个字节	记录项长度，二进制型。
17	1 个字节	记录项的精度，二进制型。
18 － 19	2 个字节	保留字节，用于以后添加新的说明性信息时使用，这里用 0 来填写。
20	1 个字节	工作区 ID 。
21 － 30	10 个字节	保留字节，用于以后添加新的说明性信息时使用，这里用 0 来填写。

31	1 个字节	MDX 标识。如果存在一个 MDX 格式的索引文件，那么这个记录项为真，否则为空。

表 2.8 记录项信息描述

代码	数据类型	允许输入的数据
B	二进制型	各种字符。
C	字符型	各种字符。
D	日期型	用于区分年、月、日的数字和一个字符，内部存储按照 YYYYMMDD 格式。
G	(General or OLE)	各种字符。
N	数值型 (Numeric)	- . 0 1 2 3 4 5 6 7 8 9
L	逻辑型（ Logical ）	? Y y N n T t F f (? 表示没有初始化) 。
M	(Memo)	各种字符。

属性文件的实体信息

实体信息部分就是一条条属性记录，每条记录都是由若干个记录项构成，因此只要依次循环读取每条记录就可以了。

一个读取 dbf 文件的例子

假设要读取一个名为 soil 的 dbf 文件（存储了土地利用信息），它含有 8 个记录项，记录项信息如表 2.10 所示：

记录项名称	数据类型	长度	小数位数
Area	数值型（ double ）	31	15
Perimeter	数值型（ double ）	31	15
soils_	数值型（ int ）	11	0
soils_id	数值型（ int ）	11	0
soil_code	字符型（ character ）	3	无
Suit	字符型（ character ）	1	无
Centroid_x	数值型（ double ）	31	15
Centroid_y	数值型（ double ）	31	15

表 2.10dbf 文件中的数据类型

下面是读取这个 dbf 文件的代码：

```
void OnReadDbf(CString DbfFileName)
{
    FILE*    m_DbfFile_fp;        //****Dbf 文件指针
    // 打开 dbf 文件
    if((m_DbfFile_fp=fopen(DbfFileName,"rb"))==NULL)
```

```
{
    return;
}
int i,j;
//////**** 读取 dbf 文件的文件头  开始
BYTE version;
fread(&version,    1,   1,m_DbfFile_fp);

BYTE date[3];
for(i=0;i<3;i++)
{
     fread(date+i,    1,   1,m_DbfFile_fp);
}
int RecordNum;          //******
fread(&RecordNum,        sizeof(int),   1,m_DbfFile_
fp);
short HeaderByteNum;
fread(&HeaderByteNum, sizeof(short),
1,m_DbfFile_fp);
short RecordByteNum
fread(&RecordByteNum, sizeof(short),
1,m_DbfFile_fp);
```

```
        short Reserved1;
        fread(&Reserved1,     sizeof(short), 1,m_DbfFile_fp);

        BYTE Flag4s;
        fread(&Flag4s,                sizeof(BYTE), 1,m_DbfFile
_fp);
     BYTE EncrypteFlag;
        fread(&EncrypteFlag,          sizeof(BYTE), 1,m_Dbf
File_fp);

        for(i=0;i<3;i++)
        {
             fread(&Unused,        sizeof(int),    1,m_DbfFile_
fp);
        }
        BYTE MDXFlag;
        fread(&MDXFlag,     sizeof(BYTE), 1,m_DbfFile_fp);

        BYTE LDriID;
        fread(&LDriID,                sizeof(BYTE), 1,m_DbfFile_
fp);
        short Reserved2;
```

```
        fread(&Reserved2,     sizeof(short), 1,m_DbfFile_fp);
        BYTE name[11];
        BYTE fieldType;
        int Reserved3;
        BYTE fieldLength;
        BYTE decimalCount;
        short Reserved4;
        BYTE workID;
        short Reserved5[5];
        BYTE mDXFlag1;
        int fieldscount;
        fieldscount = (HeaderByteNum - 32) / 32;
        // 读取记录项信息一共有 8 个记录项
      for(i=0;i< HeaderByteNum;i++)
        {
              //FieldName----11    bytes
              fread(name,     11, 1,m_DbfFile_fp);
              //FieldType----1      bytes
              fread(&fieldType,    sizeof(BYTE),
1,m_DbfFile_fp);
              //Reserved3----4      bytes
              Reserved3       =0;
```

```
            fread(&Reserved3, sizeof(int), 1,m_DbfFile_fp);
            //FieldLength--1      bytes
            fread(&fieldLength,sizeof(BYTE),
1,m_DbfFile_fp);
            //DecimalCount-1    bytes
            fread(&decimalCount,sizeof(BYTE),
1,m_DbfFile_fp);
            //Reserved4----2      bytes
            Reserved4       =0;
            fread(&Reserved4,
sizeof(short), 1,m_DbfFile_fp);
            //WorkID-------1     bytes
          fread(&workID,            sizeof(BYTE),
1,m_DbfFile_fp);
            //Reserved5----10    bytes
            for(j=0;j<5;j++)
            {
                  fread(Reserved5+j,sizeof(short), 1,m_DbfF
ile_fp);
            }
            //MDXFlag1-----1 bytes
```

```
            fread(&mDXFlag1,        sizeof(BYTE),
1,m_DbfFile_fp);
        }
        BYTE terminator;
        fread(&terminator,
    sizeof(BYTE), 1,m_DbfFile_fp);        // 读取 dbf 文件头结
束
        double Area,Perimeter,Centroid_y,Centroid_x;
        int Soils_,Soils_id;
        CString Soil_code,suit;
        BYTE    deleteFlag;
        char media[31];
        // 读取 dbf 文件记录   开始
        for(i=0;i<RecordNum;i++)
        {
            fread(&deleteFlag, sizeof(BYTE),
1,m_DbfFile_fp);              // 读取 Area double
       for(j=0;j<31;j++)
                fread(media+j, sizeof(char),
1,m_DbfFile_fp);
            Area =atof(media);              // 读取 Perimeter
double
```

```
            for(j=0;j<31;j++)
                fread(media+j, sizeof(char),
1,m_DbfFile_fp);
            Perimeter =atof(media);          // 读取
soils_ int
            for(j=0;j<31;j++)
                strcpy(media+j,"");
            for(j=0;j<11;j++)
                fread(media+j, sizeof(char),
1,m_DbfFile_fp);
            Soils_      =atoi(media);          // 读取
Soils_id int
            for(j=0;j<31;j++)
                strcpy(media+j,"");
        for(j=0;j<11;j++)
                fread(media+j, sizeof(char),
1,m_DbfFile_fp);
            Soils_id    =atoi(media);
            // 读取 soil_code string
            for(j=0;j<31;j++)
                strcpy(media+j,"");
            for(j=0;j<3;j++)
```

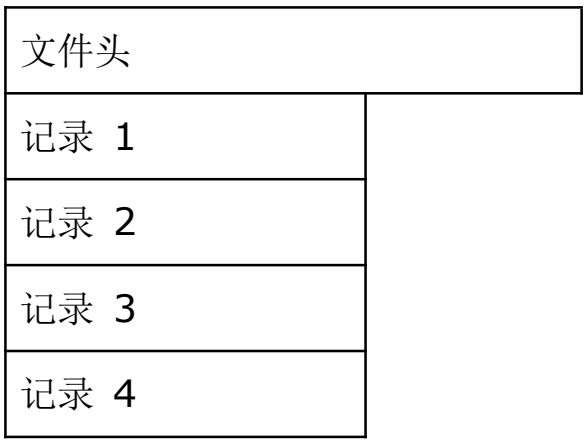
```
                fread(media+j, sizeof(char),
1,m_DbfFile_fp);
            Soil_code       =media;             // 读取 suit
string
            for(j=0;j<31;j++)
                strcpy(media+j,"");
            for(j=0;j<1;j++)
                fread(media+j, sizeof(char),
1,m_DbfFile_fp);
            suit =media;            // 读取 Centroid_y double
          for(j=0;j<31;j++)
                strcpy(media+j,"");
            for(j=0;j<31;j++)
                fread(media+j, sizeof(char),
1,m_DbfFile_fp);
            Centroid_y      =atof(media);           // 读取
Centroid_x double
            for(j=0;j<31;j++)
                strcpy(media+j,"");
            for(j=0;j<31;j++)
                fread(media+j, sizeof(char),
1,m_DbfFile_fp);
```

```
            Centroid_x      =atof(media);
        }
        // 读取 dbf 文件记录  结束
}
```

索引文件的结构说明

索引文件（ .shx ）主要包含坐标文件的索引信息，文件中每个记录包含对应的坐标文件记录距离坐标文件的文件头的偏移量。通过索引文件可以很方便地在坐标文件中定位到指定目标的坐标信息。

索引文件也是由头文件和实体信息两部分构成（如图 2.5 ），其中文件头部分是一个长度固定 (100 bytes) 的记录段，其内容与坐标文件的文件头基本一致。它的实体信息以记录为基本单位，每一条记录包括偏移量（ offset ）和记录段长度（ Content Length ）两个记录项，它们的位序都是 big ，两个记录项都是 int 型。



……

……

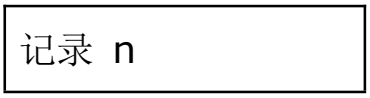


图 2.5 索引文件的结构

记录项	数值	数据类型	长度	个数	位序
位移量（Offset）	表示坐标文件中的对应记录的起始位置相对于坐标文件起始位置的位移量。	int 型	4	1	Big
记录长度（Content Length）	表示坐标文件中的对应记录的长度。	int 型	4	1	Big

表 2.11 索引文件的记录内容

下面是一段读取索引文件的代码：

```
void OnReadShx （ CString ShxFileName ）
{
    FILE*   m_ShxFile_fp;       //****Shx 文件指针
    // 打开索引文件
    if((m_ShxFile_fp=fopen(ShxFileName,"rb"))==NULL)
    {
        return;
    }
```

```
    // 读取索引文件头的内容开始
    int FileCode;
    int Unused;
    int FileLength;
    int Version;
   int ShapeType;
    double Xmin;
    double Ymin;
    double Xmax;
    double Ymax;
    double Zmin;
    double Zmax;
    double Mmin;
    double Mmax;
    fread(&FileCode,    sizeof(int),   1,m_ShxFile_fp);
    FileCode = OnChangeByteOrder(FileCode);
    for(i=0;i<5;i++)
        fread(&Unused,sizeof(int),   1,m_ShxFile_fp);
    fread(&FileLength,   sizeof(int),   1,m_ShxFile_fp);
    FileLength      = OnChangeByteOrder(FileLength);
    fread(&Version,          sizeof(int),   1,m_ShxFile_fp)
;
```

```
        fread(&ShapeType,    sizeof(int),   1,m_ShxFile_fp);
        fread(&Xmin,         sizeof(double),1,m_ShxFile_fp);
        fread(&Ymin,         sizeof(double),1,m_ShxFile_fp);
        fread(&Xmax,         sizeof(double),1,m_ShxFile_fp);
        fread(&Ymax,         sizeof(double),1,m_ShxFile_fp);
       fread(&Zmin,         sizeof(double),1,m_ShxFile_fp);
        fread(&Zmax,         sizeof(double),1,m_ShxFile_fp);
        fread(&Mmin,         sizeof(double),1,m_ShxFile_fp);
        fread(&Mmax,         sizeof(double),1,m_ShxFile_fp);
        // 读取索引文件头的内容结束


            int Offset, ContentLength;
        // 读取实体信息
        while((fread(&Offset,    sizeof(int),   1,
m_ShxFile_fp)!=0))
        {
            fread(&ContentLength,sizeof(int),   1,
m_ShxFile_fp);
            Offset            = OnChangeByteOrder(Offset);
            ContentLength       =
OnChangeByteOrder(ContentLength);
        }
```

}

小结

本节介绍了 MapObjects 支持的各种数据，并详细介绍了 shapefiles 的文件结构，同时给出了读取 shapefiles 的坐标文件（ .shp ）、属性文件（ .dbf ）和索引文件（ .shx ）的程序，给出这些程序的目的在于让读者通过这些例子深入掌握 shapefiles 文件的格式，进而具备将特定格式的数据文件转换成 shapefiles 文件的能力。