中地软件系列丛书

MAPGIS7.0 二次开发教程

—基础篇（C++版）

中地数码科技有限公司

2006 年 4 月 武汉

内容提要

《MAPGIS7.0 二次开发教程-基础篇(C++版)》是根据最新推出的 MAPGIS7.0 软件平台编写而成，主要内容包含了 MAPGIS7.0 数据模型的讲解和怎样利用 MAPGIS7.0 提供的框架组件搭建自己的应用系统两部分。

本书作为 MAPGIS7.0 地理信息系统系列产品配套使用手册，供使用 MAPGIS7.0 地理信息系统进行二次开发的用户参考。

前　言

在国家“十五”863项目的支持下，历经5年的科技攻关，由中地数码科技有限公司开发的具有完全自主版权的第一套“分布式超大型GIS平台软件MAPGIS7.0”已经研制成功。MAPGIS7.0是属于最新的“第四代GIS”软件产品，具备“纵向多层，横向网格”的分布式体系结构，采用“面向服务”的最新设计思想，支持局域和广域网络环境下空间信息网格（SIG）的分布式计算，实现了面向空间实体及其关系的数据组织、高效海量空间数据的存储与索引、大尺度多维动态空间信息数据库、三维实体建模和分析，具有TB级空间数据处理能力、支持分布式空间信息分发与共享、网络化空间信息服务，支持Unix/Linux大型服务器，支持海量、分布式的国家空间基础设施建设。

《MAPGIS7.0二次开发教程-基础篇(C++版)》是根据最新推出的MAPGIS7.0软件平台编写而成，主要内容包含了MAPGIS7.0数据模型的讲解和怎样利用MAPGIS7.0提供的框架组件搭建自己的应用系统两部分。

本书共分为十二章：

第一部分是数据存储部分（第一章到第七章），通过示例展示了地理数据库、要素类、简单要素类、对象类、关系类、注记类等，规则各个类之间的逻辑关系和使用这些数据模型应注意的问题。

第二部分是基于插件的应用框架部分（第八章到第十四章），包含应用框架、地图文档、地图可视化、基本显示、系统库管理、图形编辑、空间分析组件的逻辑关系和应用这些组件来搭建自己应用框架的示例代码。

参加本书编写的人员主要是MAPGIS7.0的软件开发工程师和二次开发技术支持工程师。由于时间仓促，书中难免存在错误和不当之处，敬请广大用户及读者提出宝贵意见和建议，以利改进。

中地软件丛书编委会

2006年4月

目　　录

第 1 章 MAPGIS7 空间数据管理

1.1 面向实体的空间数据模型

经过几十年的发展，今天的 GIS 系统已经具备了较强的数据存贮、管理和输入输出功能，但目前大多数的 GIS 仍然是以数据为中心的，在完整表达客观地理世界、进行高层次的空间分析和直接提出决策方案的能力方面还远远不够，导致这种情况的根本原因在于现有 GIS 的数据模型不能准确地表达客观地理世界。

MAPGIS7 的空间数据模型将现实世界中的各种现象抽象为对象、关系和规则，各种行为（操作）基于对象、关系和规则，模型更接近人类面向实体的思维方式。该模型还综合了面向图形的空间数据模型的特点，使得模型表达能力强，广泛适应 GIS 的各种应用。该模型具有以下特点：

1. 真正的面向地理实体，全面支持对象、类、子类、子类型、关系、有效性规则、数据集、地理数据库等概念；
2. 对象类型覆盖 GIS 和 CAD 对模型的双重要求，包括：要素类、对象类、关系类、注记类、修饰类、动态类、几何网络；
3. 具备类视图概念，可通过属性条件、空间条件和子类型条件定义要素类视图、对象类视图、注记类视图和动态类视图。
4. 要素可描述任意几何复杂度的实体，如水系。
5. 完善的关系定义，可表达实体间的空间关系、拓扑关系和非空间关系。空间关系按照 9 交模型定义；拓扑关系支持结构表达方式和空间规则表达方式；完整地支持 4 类非空间关系，包括关联关系、继承关系(完全继承或部分继承)、组合关系（聚集关系或组成关系）、依赖关系。
6. 支持关系多重性，包括 1－1、1－M、N－M。
7. 支持有效性规则的定义和维护，包括定义域规则、关系规则、拓扑规则、空间规则、网络连接规则。
8. 支持多层次数据组织，包括地理数据库、数据集、数据包、类、几何元素、几何实体、几何数据，如图 1-1 所示。
9. 几何数据支持向量表示法和解析表示法，包括折线、圆、椭圆、弧、矩形、样条、bezier 曲线等形态。能够支持规划设计等应用领域。

1.2 实体表达及分类

1.2.1 对象

在 MAPGIS7.0 中，现实世界中的实体用对象表示。诸如房子、湖泊或顾客之类的实体，均可用对象表示。对象有属性、行为和一定的规则，以记录的形式存储对象。对象是各种实体一般性的抽象，特殊性对象包括要素、关系、注记、修饰符、轨迹、连接边、连接点等。

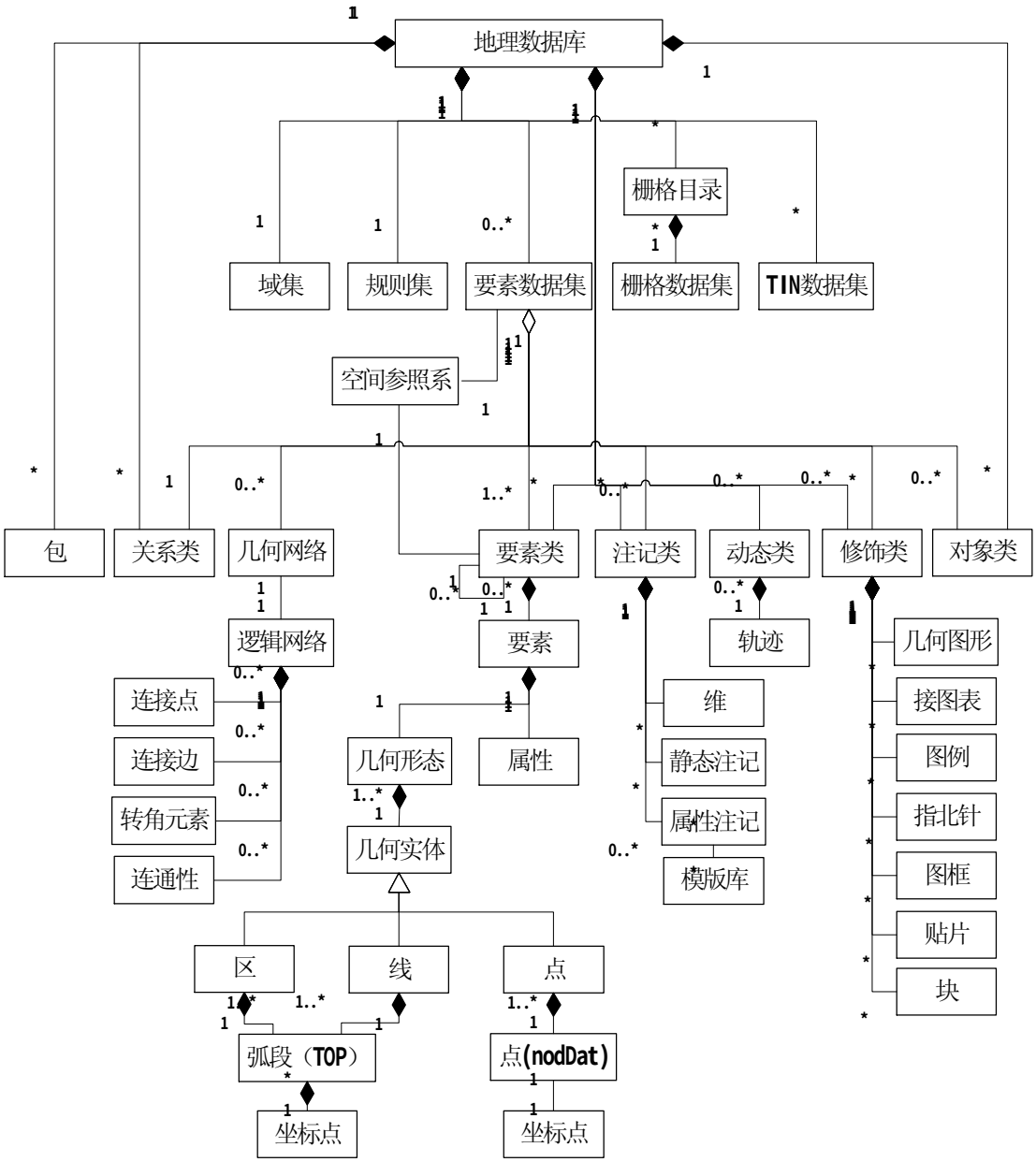


图 1-1 MAPGIS7.0 面向实体的空间数据

1.2.2 对象类型、子类型

根据对象的行为和属性可以将对象划分成不同的类型，具有相同行为和属性的对象构成对象类，特殊的对象类包括要素类、关系类、注记类、修饰类、动态类、几何网络。不特别声明的情况下，对象类是指没有空间特征的同类对象集。

子类型是对象类的轻量级分类，以表达相似对象，如供水管网中区分钢管、塑料管、水泥管。不同类或子类型的对象可以有不同的属性缺省值和属性域。

1.2.3 对象类

对象类是具有相同行为和属性的对象的集合。在空间数据模型中，一般情况下，对象类是指没有几何特征的对象（如房屋所有者、表格记录等）的集合；在忽略对象特殊性的情况下，对象类可以指任意一种类型的对象集。

1.2.4 要素类

- 要素类是具有相同属性的要素的集合，是一种特殊的对象类。
- 要素是现实世界中现象的抽象，往往用于表达某种类型的地理实体，如道路、学校等。要素是真实世界中的地理对象在地图上的表示，要素具有几何和属性。如果该要素与地球上的某一地理位置相关，则该要素就为地理要素。
- 要素类是相同类型要素的集合。
- 要素按其数据组织方式的不同可以分为简单要素模型和复杂要素模型。

1. 简单要素模型

基本概念

1. 点 ：0 维几何，表示坐标空间中的一个单独位置。点有一个 x 坐标值和一个 y 坐标值。
2. 线 ：是一维的几何实体。用来表示较狭窄的要素。包括折线和弧线。
3. 曲线 ：1 维几何基形，表示一条线的连续影像。注：曲线的边界是该曲线起止端点的集合。如果该曲线是一个环，两端点是相同的，该曲线（是拓扑闭合）被认为没有边界。第一个点称为起始点，最后一个点称为终止点。该曲线的连通性由“一条线的连续影像” 子句保证。拓扑理论认为一个连通的集合的连续影像是连通的。
4. 曲面 ：是一个 2 维几何对象。OpenGIS 抽象规范将简单表面定义为由一个简单碎片（patch）组成， 该碎片(patch) 与一个外部边界和 0 个或更多的内部边界相联系。
5. 多边形 ：是一个平面表面，通过一个外部边界和 0 个或更多的内部边界定义。每个内部边界定义一个多边形的一个洞（hole）。
   多边形的断言（assertion）（定义有效多边形的规则）是：
   - 多边形是拓扑闭合的。
   - 多边形的边界是由一组线性环组成的，这些线性环构成多边形的外部边界和内部边界。
   - 在边界交叉处不会出现两个环，在多边形的边界中的环可能在一点上相交，但只可能相切。
   - 多边形不可能有分割线（cut lines），刺穿(spikes)或者小孔(punctures)。
   - 每个多边形的内部是一组相连的点。
   - 带有一个或者多个洞（hole）的多边形的外部是不相连的。每个洞（hole）定义一个外部的相连组件。
6. 多点 ：多点是 0 维几何集。多点的组成元素限制为点。这些点不是相连或有序的。如果多点中没有两个相同的点（有相同的坐标值），那么多点是简单的。
7. 多线 ：是一个 1 维几何集，它的组成元素是线。
8. 多面 ：是一个 2 维几何集，组成元素是表面（surface）。多面中的任意两个表面的内部都可能相交。多面的任意两个组成元素的边界都只可能相交于有限多个点。
9. 多多边形：多多边形是一个由多边形组成的多面。
   多多边形的断言（assertion）（定义有效多多边形的规则）为：
   - 组成多多边形的两个多边形的内部不可能相交。
   - 组成多多边形的两个多边形不可能相交（cross），但可能与有限个点相连。（注意，穿过在上面的第一条中就被禁止了）
   - 多多边形被定义为拓扑闭合的。
   - 多多边形不能有分割线（cut lines），刺穿（spikes）或者小孔（punctures）。一个多多边形是一个规则的，闭合的点集。

- Ø 带有 1 个以上多边形的多多边形的内部的是不连通的，多多边形的内部的连通组件的数量就等于多多边形中的多边形的数量。一个多多边形的边界是一组对应于它的成员多边形的边界的封闭曲线。多多边形的边界中的每一条曲线都在一个成员多边形的边界中，而成员多边形边界的每条曲线都是在多多边形的边界中的。

10. 子元素：指一个元素 c 是另一个元素（它的父元素）p 的内容，但不是 p 中的任何其他元素的内容。多点、多线、多多边形拥有子元素。子元素即构成多点、多线、多多边形的点、线、多边形。子元素的个数即构成多点、多线、多多边形的点、线、多边形的个数。
11. 解析线：解析线指由解析线参数控制的曲线。如弧线、圆心半径园、3 点圆、椭圆、圆心半径起始角终止角（顺时针旋转）”弧、3 点弧、矩形（对角点＋角度）、矩形（单点＋长度宽度＋角度）、样条曲线、Bezier 曲线。

2 数据模型

简单要素模型支持的几何形态有：

1. 点和多点

单个点或多个点构成的几何实体（支持三维数据）。

2. 线

支持折线，弧线、圆心半径圆、3 点圆、椭圆、圆心半径起始角终止角（顺时针旋转）”弧、3 点弧、矩形（对角点＋角度）、矩形（单点＋长度宽度＋角度）、样条曲线、Bezier 曲线。

3. 多线

由多个符合 2 的线构成。

4. 多边形

支持由符合 2 的线构成的多边形。

5. 多多边形

由多个符合 3 的多边形构成。

2 数据组织结构图

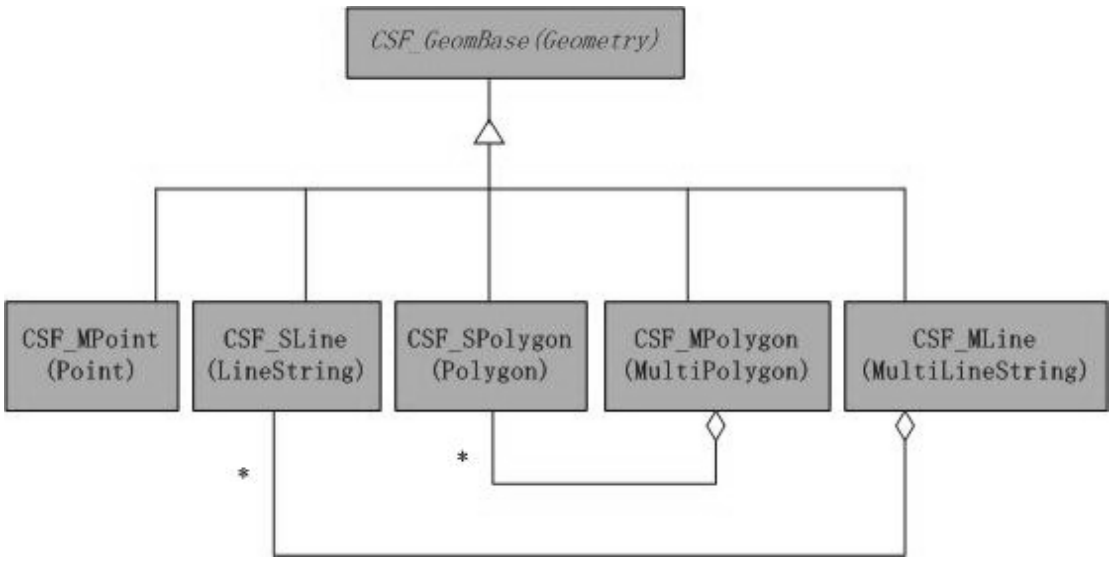


图 1-2　简单要素类数据组织结构图

2. 复杂要素模型

n 基本概念

一、属性（Attribute）

要素的特性（characteristic）。

二、几何实体

地理对象的外观特征或可视化形状。地理对象可以用三种几何实体表示在地图上：点、线、多边形。

三、弧段

空间数据层中的坐标点序列，包括：折线、圆、椭圆、弧、矩形、样条、贝齐尔等。

四、点

空间数据层中单一坐标点，点分为“一般点”、“实结点”、“虚结点”。

虚结点：为了实现连通性由系统内部产生的点。它没有对应的地物，也不能引用。

五、假点

空间数据层中的引用点。

六、假弧段

空间数据层中的引用弧段。

n 概念层次

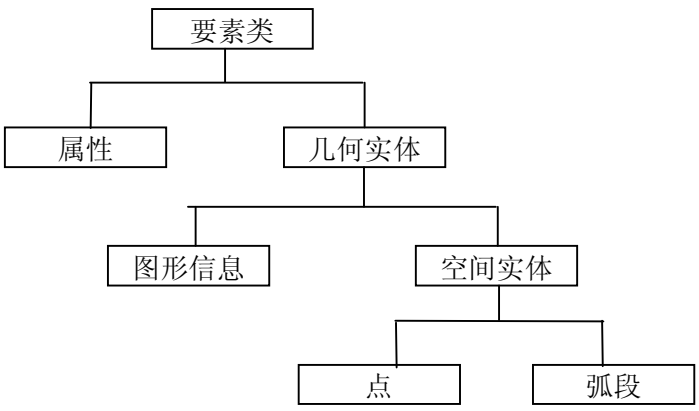


图 1-3　复杂要素概念层次图

- 数据组织结构图

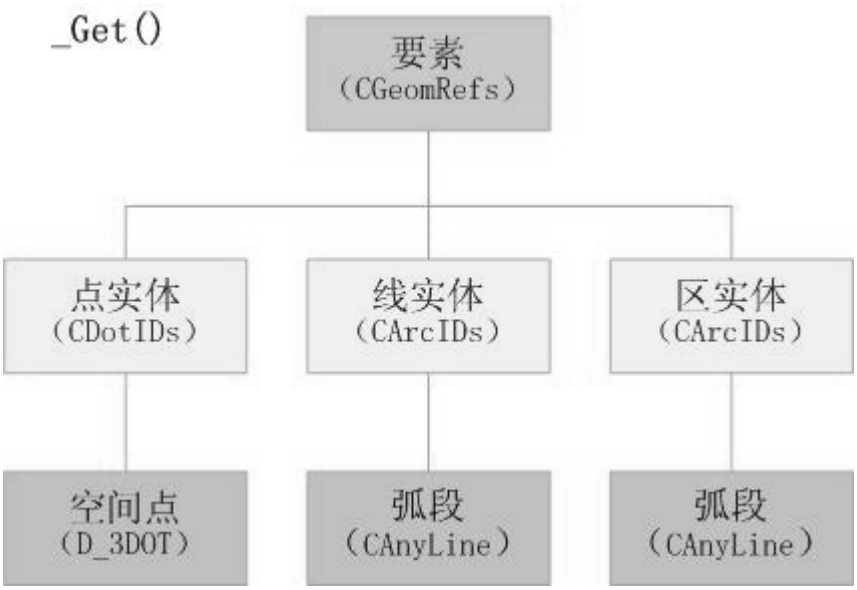


图 1-4　复杂要素数据组织结构图

1.2.5 关系类

现实世界中的各种现象是普遍联系的，而联系本身也是一种特殊现象，具有多种表现形式。在面向实体的空间数据模型中，对象之间的联系被称作关系，是一种特殊的对象。

房屋所有者和房屋之间的产权关系，具有公共边界的行政区之间的相邻关系，甲乙双方之间的合同关系，都是对象之间关系的实例。

在该模型中，关系被分为空间关系和非空间关系。其中：

（1）空间关系与对象的位置和形态等空间特性有关，包括距离关系和拓扑关系。拓扑关系如水管和阀门的连接关系、两条道路的相交关系；

（2）非空间关系是对象属性之间存在的关系，如甲乙方之间的合同关系。

关系类是关系的集合，一般在对象类、要素类、注记类、修饰类的任意两者之间建立关系类。

1.2.6 注记类

注记是一种标识要素的描述性文本，分为静态注记、属性注记和维注记。其中：

(1) 文本注记是一种内容和位置固定的注记，包括注记和版面。

(2) 属性注记的内容来自要素的属性值，显示属性注记时，动态地将属性值填入注记模板。因此也称为动态注记，属性注记直接和它要标注的要素相关联，移动要素时，注记跟随移动，注记的生命期受该要素的生命期控制。

(3) 维注记是一种特殊类型的地图注记，仅用来表示特定的长度和距离。维分为平行维和线性维。平行维与基线平行，表示真实距离；线性维可以是垂直、水平或旋转的，并不表示真实距离。

注记的集合构成注记类。

1.2.7 修饰类

用于存储修饰地图或者辅助制图的要素，包括几何图形、接图表、图例、指北针、图框、贴片和块。

1.2.8 动态类

动态类是一种特殊的对象类，是空间位置随时间变化的动态对象的集合。
动态对象的位置随时间变化形成轨迹
动态类中记录轨迹的信息，包括 x、y、z、t 和属性。

1.3 类的命名规则

§ .....Cls (类)
CFeatureCls 要素类
CSFeatureCls 简单要素类
CObjectCls 对象类
CRelationCls 关系类
CAnnotationCls 注记类
CDynamicCls 动态类
CRuleCls 规则类
CDressCls 修饰类

§ .....Set (集)
CFeatureSet 要素集
CSFeatureSet 简单要素集
CObjectSet 对象集
CAnnotationSet 注记集
CDynamicSet 动态集
CGeomLinSet 线状几何实体集
CGeomRegSet 面状几何实体集
CGeomPntSet 点状几何实体集

§ .....s (集、多是相同元素的聚集)
CPoints 多点

1.4 简单要素数据模型导引

下面示例是在一个简单要素类里添加一个简单要素对象。

```
//例 1-1 简单要素
void AppendSFeature(CSFeatureCls *ptSFCls)
{
    CAnyLine        line;      // 任意线（几何实体）
    CRecord         linAtt;    // 属性
    LIN_INFO        linInfo;   // 图形参数
    memset(&linInfo,0,sizeof(LIN_INFO));
    CATT_STRU       attStru;
```

```
    //设置要素属性
    ptSFCls->att_GetStru(attStru);
    linAtt.SetStru(attStru);
    for(int i = 0;i<attStru.hd.numbfield;i++)
        linAtt.SetFldFromStr(i,"Zondy");

    //设置要素几何形态
    line.m_linType = ARC_TYPE_BROKEN_LINE;
    D_DOT* dotset = new D_DOT[2];
    dotset[0].x = 0; dotset[0].y = 0;
    dotset[1].x = 50; dotset[1].y = 50;
    line.m_varLin.Set(dotset,2);
    linInfo.linstyID = 6;

    //在简单要素类里添加要素
    TYPE_LIN_ID SfeaID = ptSFCls->line_Append(&line,&linAtt,&linInfo);
    delete []dotset;
}
```

定义的 D_DOT*决定了这条折线的坐标序列，CAnyLine 是几何实体封装类，这里是任意线，在 basclass70.h 头文件中可以看到其他几何实体封装类的声明，从前面的章节了解到简单要素模型支持的几何形态有 1）点和多点 2）线 3）多线 4）多边形 5）多多边形，
这些几何封装都是继承于 CGeomBase，有 CPoints, CMultiLine, CPolygon 等。这些几何形态+属性构成了简单要素。

以上的示例是添加一个简单要素到简单要素层，由于应用层次不同，在 MAPGIS7 中还有一种数据模型，就是要素类里的要素，要素具有拓扑信息，可以把任意的点、线、面组合成一个要素，满足更复杂的应用。

下一节具体介绍 MAPGIS7 中的几何实体封装类。

1.5 几何实体封装类

几何实体包括：点、多点、折线、解析线、多线、多边型、多多边型，这些几何实体类都继承 CGeomBase 类，下面是 CGeomBase 的声明（部分）

```
//几何封装类的基类
class BCLS_EXPORT_CLASS CGeomBase
{
public:
    CGeomBase();
```

```
    virtual ~CGeomBase();

    virtual short type()=0;          //成功返回 type>=1;失败返回 0
    virtual long    Save(CSmartFile *ptFile, long off)=0; //返回数据长度
    ……
}
```

CGeomBase 提供了 10 个实现类的公共的接口，并都设置为虚函数，在例 1-1 中 line_Append 需要传入的参数是 CGeomBase* 类型，定义一个 CAnyLine line；由于 CAnyLine 继承于 CGeomBase，传入&line，这种向上类型转换是安全的，几何封装类每实现类的接口很相似，这里举个多线的事例，其他的参考相对应的开发手册。

```
//例 1-2 几何封装类
CMultiLine      ptMultLin;
D_DOT           dot[2];

CGeomBasePTR    *ptLine;
ptLine = new CGeomBasePTR[2];

dot[0].x = 0; dot[0].y = 0;
dot[1].x = 0; dot[1].y = 100;
ptLine[0] = new CAnyLine;
ptLine[1] = new CAnyLine;
((CAnyLine*)ptLine[0])->m_varLin.Append(dot,2);
dot[0].x = 0; dot[0].y = 100;
dot[1].x = 100; dot[1].y = 100;
((CAnyLine*)ptLine[1])->m_varLin.Append(dot,2);

ptMultLin.Set(ptLine,2);

delete   (CAnyLine*) ptLine[0];
delete   (CAnyLine*) ptLine[1];
delete   []ptLine;
```

上面的示例是在一个 CMultiLine（多线类）里添加两条 AnyLine，首先申请了可以存放两条线的存储空间，new CGeomBasePTR[2]，通过 Append 方法存放，存放的时候进行了类型转换。

根据这个示例，了解到多线的实现，其他例如多点和多多边型，开发时可以参照开发手册和头文件。

图 1-1（MAPGIS7.0 面向实体的空间数据模型）层次最高的是地理数据库，地理数据库是面向实体空间数据模型的全局视图，完整地、一致地表达了被描述区域的地理模型。下一章将介绍地理数据库。

第 2 章 地理数据库

2.1 存储策略

地理数据库存储和管理地理数据。MAPGIS7.0 地理数据库采取基于文件和基于商业数据库两种存储策略。由于这两种存储策略支持相同的空间数据模型，因此在文件和数据库之间能够实现无损的平滑的数据迁移；同时，两种策略具有共同的平台，这使得上层软件不需要因为数据迁移而改变。

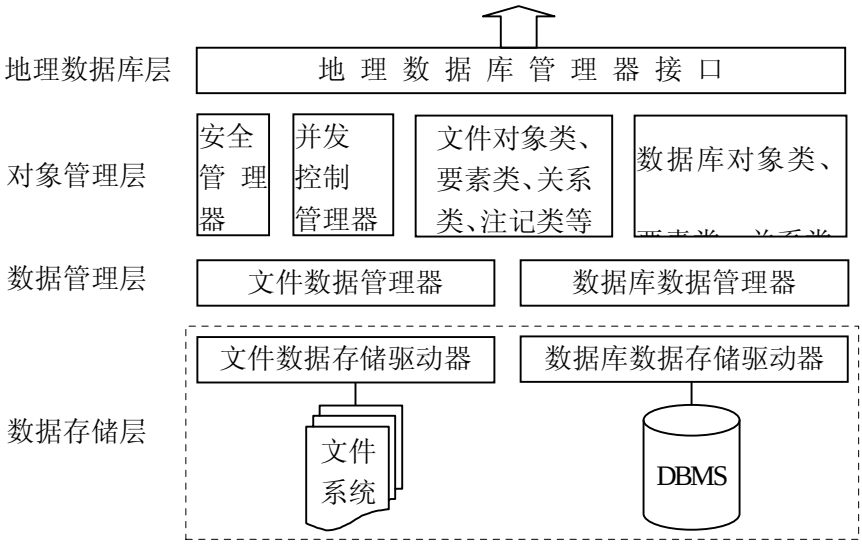


图 2-1 MAPGIS7.0 存储策略概念图

针对不同的应用规模和应用阶段，给用户提供了多种**最佳的性价比**和**最大的投资收益率**选择方案。例如：

- 应用规模小的用户、二次开发团体、教学单位、数据累积规模较小的用户都可选择基于文件的存储策略，以节省昂贵的商业数据库费用；
- 大型、超大型应用可选择基于商业数据库的存储策略；
- 分多个阶段进行开发的应用，在前期阶段，数据规模较小，用户不多，在后期阶段数据规模大，用户多，则可先采用文件存储策略，再购买适当许可数的商业数据库和服务器设备、以后根据数据规模和业务情况再增加数据库许可数和服务器等软硬件设备。这不仅提高了用户的资金利用率，而且在软硬件性能迅速提高，价格江河日下的今天，让用户享受到多重好处。

2.2 地理数据库的数据组织

MAPGIS7 按照“地理数据库－数据集－类”这几个层次组织数据，以满足不同应用领域对不同专题数据的组织和管理需要。

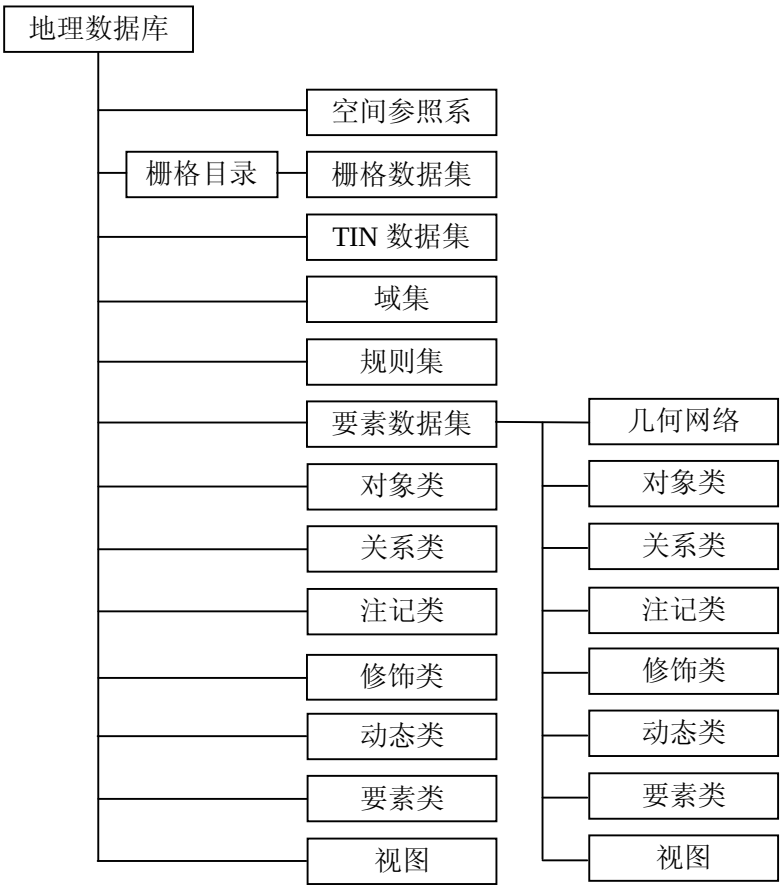


图 2-2　地理数据库数据组织

其中数据集是地理数据库中若干不同对象类的集合，通过命名数据集提供了一种数据分类视图，便于数据组织、管理和授权。根据不同的用途，数据集分为：要素数据集、栅格目录、栅格数据集、TIN数据集。

(1) 要素数据集

要素数据集是地理数据库中具有相同空间参照系的要素类、对象类、关系类、注记类、修饰类、动态类、几何网络的集合；

在一个要素数据集下存放不同的类，有利于以下情况的数据组织：

- 专题归类：当不同的要素类属于同一范畴时，可将其组织到一个要素数据集中。例如：全国范围内某种比例尺的地质图数据。
- 共享几何特征：如用地、水系、行政区界等不同的要素类需要共享公共几何边界时，将其组织到一个要素数据集中，当移动其中的一个要素时，公共的部分也要求一起移动,并可保持这种公共边的关系不变。
- 创建几何网络：同一个几何网络中作为边要素和点要素的各种要素类，必须组织到一个要素数据集中，以便于管理和建立网络模型。例如：通讯网络中，各种交换机、交接箱、路由器、电缆、光缆等，分别对应点要素类和线要素类，在通讯网络建模时,将其全部考虑到通讯网络对应的几何网络模型中。

(2) 栅格目录

栅格目录用于管理有相同空间参照系的多幅栅格数据，各栅格数据在物理上独立存储，易于更新，常用于管理更新周期快、数据量较大的影像数据。同时，栅格目录也可实现栅格数据和栅格数据集的混合管理，其中目录项既可以是单幅栅格数据，也可以是地理数据库中已经存在的栅格数据集，具有数据

组织灵活、层次清晰的特点。

(3) 栅格数据集

栅格数据集用于管理具有相同空间参照系的一幅或多幅镶嵌而成的栅格影像数据，物理上真正实现数据的无缝存储，适合管理 Dem 等空间连续分布、频繁用于分析的栅格数据类型。由于物理上的无缝拼接，因此以栅格数据集为基础的各种栅格数据空间分析具有速度快、精度较高的特点。

(4) TIN 数据集

包含一系列含有 X,Y 和 Z 坐标的点集和由它们构建的三角网集。相对栅格数据模型而言，TIN 数据集提供了显示、分析地表和其他表面模型的另外一种数据组织形式。

2.3 如何创建地理数据库

例 2-1 创建基于文件的地理数据库

```
void   OnCreateGdb(){
    CGDataBase* GDateBase;
    CGDBServer* ptWA;
    char szName[] = "My TestGDB";
    HDF_CREATE_MSG Msg;

    memset(&Msg,0,sizeof(HDF_CREATE_MSG));
    strcpy(Msg.HDFName,"D:\\MyTestGDB");
    Msg.extendInfo.extendMode = 0;
    Msg.extendInfo.isExtendable = 1;
    Msg.extendInfo.maxFileSize = 0;
    Msg.size = 10;

    ptWA = new CGDBServer;
    GDateBase = new CGDataBase;

    if(ptWA->Connect("MapGisLocal")>0)
        GDateBase->Create(ptWA,szName,&Msg,1);
        ptWA->Disconnect();

    delete ptWA; ptWA = NULL;
    delete GDateBase; GDateBase = NULL;
}
```

以上的示例是用 CGDataBase 的 Create 方法在 D 分区上创建一个名为 MyTestGDB 地理数据库，在创建之前需要连接到 MAPGIS 本地的数据源 MapGisLocal 上，当然也可以通过其它数据源来创建地理数据库，建立 MAPGIS7 数据源的方法在以后的章节介绍，HDF_CREATE_MSG 结构决定了地理数据库的特性，其中的 size 是 HDF 文件的大小，以 M 为单位，HDFName 是数据库物理数据文件的名称（需带全路径），extendInfo 是数据库的扩展信息，如 maxFileSize 表示允许增长到的最大文件大小，0 表示不受限制。

MAPGIS7 地理数据库连接数据源的方式包括：LOCAL，WINDOWS 服务，ORACLE，ODBC，

WEB_SERVICE，通过 Cln_config.h 里的 cfg_ModSvcInfo 函数来创建这些数据源，下面是创建时的对话框。

图 2-3　cfg_ModSvcInfo 函数调用成功的对话框

2.4 打开地理数据库

例 2-2 打开地理数据库

```
void OnOpengdb()
{
    CGDBServer    GDBSvr;
    CGDataBase   GDB;
    GDBPATHINFO path;
    char        user[30]="";
    char        pwsd[64]="";

    long ptSelFlag = XCLS_TYPE_GDB;
    long selN = 1;
    if(ShowGDBShellDlg(path,&ptSelFlag,selN)>0)
    {
        GetLoginInfo(path.svcName,user,30,pwsd,64);
        if(GDBSvr.Connect(path.svcName,user,pwsd)>0)
```

```
            if(GDB.Open(&GDBSvr,path.gdbName)>0)
            MessageBox("GDB 打开成功!");
    }
}
```

例 2-2 通过显示 GDB 目录的对话框函数 ShowGDBShellDlg 打开地理数据库。

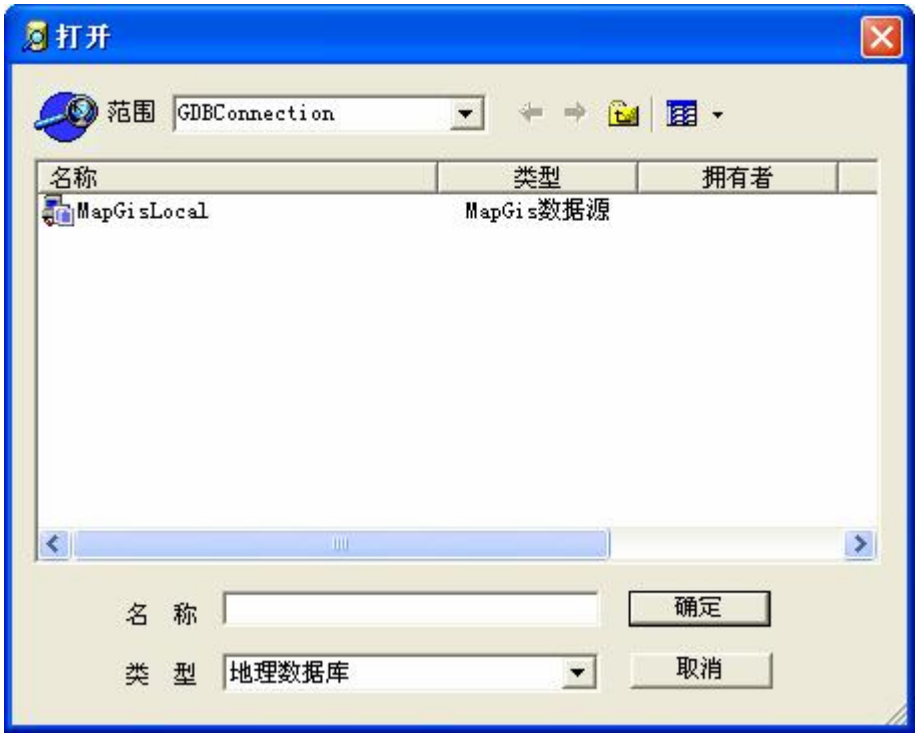


图 2-4ShowGDBShellDlg 函数的执行结果

ptSelFlag 是对话框中显示的文件类型，XCLS_TYPE_GDB 代表地理数据库，其他类型定义参阅 basdefine70.h，GDBPATHINFO 是 GDB 路径信息，示例中的 svcName 和 gdbName 分别是服务名和 GDB 名，通过服务名连接成功后就可以打开相应的 GDB。

2.5 小结

地理数据库类声明（部分）

```
class CLN_BAS70_EXPORT_CLASS CGDataBase
{
public:
//1. HDF 操作函数，hdfType＝HDF_DATA/HDF_LOG 分别标识“数据 HDF”和“日志 HDF”
    long hdf_GetNum(char hdfType, short *ptHDFNum);
//2. 空间参照系操作函数
    long sRef_GetNum(short type, long *n);
//3. 要素数据集操作函数
    TYPE_DS_ID fds_Create(char *dsName, TYPE_XCLS_ID srID);
//4. 要素类操作函数
```

```
long fcls_GetNum(TYPE_XCLS_ID dsID, short type, short clsORView, long *ptNum);
//5. “对象类”操作函数
//6. “注记类”操作函数
……
}
```

地理数据库类 CGDataBase 的声明中包含了对空间参照系操作函数，要素数据集操作函数，“对象类”操作函数，说明地理数据库是面向实体空间数据模型的全局视图，完整地、一致地表达了被描述区域的地理模型。一个地理数据库包括 1 个全局的空间参照系、1 个域集、1 个规则集、多个数据集、多数据包和各种对象类。大家可以回顾图 2-2　地理数据库数据组织图，下一章开始介绍地理数据库里的管理的简单要素类和要素数据集。

第 3 章 简单要素类

3.1 简单要素类概述

- 要素是现实世界中现象的抽象，往往用于表达某种类型的地理实体，要素具有几何和属性。
- 要素类是相同类型要素的集合。
- 要素按其数据组织方式的不同可以分为简单要素模型和复杂要素模型。

MAPGIS7 中定义一个简单要素表示一个独立的实体，不存储拓扑信息。

下面是 cln_bas70.h 中简单要素类的声明（部分）

```
class CLN_BAS70_EXPORT_CLASS CSFeatureCls : public CSFeatureSpace, public CAttributeWrite, public CSubTypeWrite, public CVersionCls, public CValidation
{
public:
//CAttributeSpace
    virtual long att_Update(    TYPE_OBJ_ID ID, CRecord &rcd);
//①简单要素类操作
TYPE_FCLS_ID    cls_Create(CGDataBase    *ptGDB,    char    *ptFclsName,    TYPE_XCLS_ID    dsID,
                TYPE_XCLS_ID gNetID, TYPE_XCLS_ID srID,
                char *ptAliasName, char *ptModelName, char *ptSubTypeField, CATT_STRU *fclsStru, char
                fclsType=FCLS_SUNION_TYPE, long scale=1);
//④点或多点要素操作函数
TYPE_PNT_ID pnt_Append(CGeomBase *pnt, CRecord *att=NULL, PNT_INFO *inf=NULL);
PNT_INFO_ID    pinf_Append(PNT_INFO &inf);
//⑤线或多线要素操作函数
//⑥多边形或多多边形要素操作函数
};
```

列出的几个成员函数，att_Update 是属性更新，pnt_Append 是添加点和多点要素，pinf_Append 是添加点和多点的图形参数，可以看出简单要素具有几何形态和属性。第一章中示例 1-1，1-2 分别介绍了简单要素的组成和几何实体的封装，在简单要素类里面添加几何实体都是用几何实体的基类类型 CGeomBase。

3.2 简单要素类的操作举例

3.2.1 创建简单要素类

例 3-1 创建简单要素类（以混合要素类为例）

```
void    OnCREATECSFeaCls()
{
    CGDBServer              GDBSvr;
```

```
CGDataBase              *ptGDB; //存储该要素类的地理数据库
GDBPATHINFO             path;       //地理数据库路径信息
long                    ptSelFlag=XCLS_TYPE_GDB;      //文件类型为数据库
long                    selN=1;                       //文件类型个数
char                    user[30]="";
char                    pwsd[64]="";

CSFeatureCls            *ptSFCls=NULL;                //简单要素类对象指针
char                    SFclsName[]="TestSFeaCls";    //简单要素类名称
/*简单要素类所属的要素数据集的ID，dsID=0表示要创建的要素类
不属于某个要素数据集。*/
long            DsID=0;
long            NetID=0;                        //简单要素类所在的几何网络的ID
long            srID=0;                         //空间参照系的id
char            AliasName[]="MyAliasName";//简单要素类的别名，不能超过128个字节
char            ModelName[]="MyModelName";  //模板名称，不能超过128个字节
char            *ptSubTypeField=NULL;//子类型的字段名，可为NULL，表示不划分子类型
//简单要素类的属性结构，该参数可以为NULL，表示没有属性
//如要创建属性结构，参考要素类的创建示例
CATT_STRU       *fclsStru=NULL;

char                    fclsType=FCLS_SUNION_TYPE;      //混合要素类
TYPE_FCLS_ID    fclsid=0;                               //简单要素类ID;

ptGDB = new CGDataBase;
ptSFCls = new CSFeatureCls;

if(ShowGDBShellDlg(path,&ptSelFlag,selN,0,"选择类")>0)  //地理数据库目录选择对话框
{
    GetLoginInfo(path.svcName,user,30,pwsd,64);         //得到对话框中的用户名和密码
    if(GDBSvr.Connect(path.svcName,user,pwsd)>0)        //服务器连接
        if(ptGDB->Open(&GDBSvr,path.gdbName)>0)         //打开地理数据库
        {
            fclsid=ptSFCls->cls_Create(ptGDB, SFclsName, DsID, NetID, selN,
                AliasName, ModelName, ptSubTypeField, fclsStru,fclsType);
        }
}

if(fclsid>0)
    AfxMessageBox("简单要素类创建成功！");
else
    AfxMessageBox("简单要素类创建失败！");

delete ptSFCls; ptSFCls=NULL;
```

```
    delete ptGDB;  ptGDB=NULL;

}
```

示例通过 ShowGDBShellDlg 打开地理数据库，打开数据库的过程参阅地理数据库一章，CSFeatureCls（简单要素类）cls_Create 成员函数创建一个简单要素类，fclsType 是简单要素类的类型，简单要素类有四种类型，分别是点（FCLS_SPNT_TYPE），线（FCLS_SLIN_TYPE），面（FCLS_SREG_TYPE），和混合要素（FCLS_SUNION_TYPE）。

3.2.2 添加一个要素

例 3-2：添加一个要素（以几何形态为区（只有一圈弧段）的要素为例）

```
void     OnAppendsfeature(CSFeatureCls *ptSFCls)
{
    CAnyPolygon       reg;                          //多边形对象指针
    D_DOT             plydot[5];                    //坐标点序列
    TYPE_REG_ID       objid;                        //多边形要素的id
    REG_INFO          rinf;                         //多边形图形参数
    long              ptNe[2];                    //记录多边形每条线上点数的数组
    CGeomBasePTR      *ptLine;

    memset(&rinf,0,sizeof(REG_INFO));
    rinf.libID = 0;

    rinf.fillclr = 6;                               //区域填充色
    rinf.endclr = 6;                                //区域填充结束色
    rinf.patID = 1632;
    rinf.pathei = 5;                                //图案高
    rinf.patwid = 5;                                //图案宽
    rinf.patclr = 5;                        //图案颜色，图案颜色固定，即使渐变填充时也不变。
    rinf.outpenw = 2;                               //图案笔宽

    ptNe[0]    = 1;                                 //第一圈的弧段数
   ptNe[1] = 0;

    //构造多边型坐标
    plydot[0].x=0;        plydot[0].y=0;
    plydot[1].x=100;      plydot[1].y=0;
    plydot[2].x=100;      plydot[2].y=100;
    plydot[3].x=0;        plydot[3].y=100;
    plydot[4].x=0;        plydot[4].y=0;

    ptLine = new CGeomBasePTR[2];
    memset(ptLine,0,sizeof(CGeomBasePTR)*2);

    ptLine[0] = new CAnyLine;
```

```
    if(((CAnyLine*)(ptLine[0]))->m_varLin.Append(plydot,5,2)>0)
    {
        reg.Set(ptLine,ptNe,1);
        objid = ptSFCls->polygon_Append(&reg,NULL,&rinf);   //添加多边形要素到简单要素类
        if(objid>0)
            AfxMessageBox("添加区要素成功！");
    }
    delete (CAnyLine*)ptLine[0];
    delete[] ptLine;       ptLine = NULL;
}
```

简单要素类 CSFeatureCls 通过 polygon_Append 成员函数添加几何形态为区的要素，第一个参数 CGeomBase *，定义一个 CAnyPolygon 的对象，把地址传入。ptLine 指向组成多边形边界上的线的数组，这里的示例虽然只有一条线作为区的边界，但实际应用中组成区边界的线往往是多条，这里申请可以存放两条线指针的存储空间只是为了说明这个问题。

polygon_Append 成员函数的第二个参数是 CRecord 类型，CRecord 对象是属性记录对象，大家会在对象类的章节看到相应的介绍，这里给 NULL，添加的要素就不具有属性记录。

第三个参数是图形参数结构，记录了图案库的编号和图案高宽等信息，具体的参见 basdefine70.h.

3.2.3 简单要素类查询

例 3-3 矩形查询示例（以区几何形态的要素类型为例）

```
void OnRectAsk(CSFeatureCls *ptSFCls)
{
    D_RECT      rect;
    long        rtn=-1;
    CSFeatureSet *ptSet=NULL;
    TYPE_OBJ_ID objid;

    //构造矩形范围
    rect.xmin = -100;       rect.ymin = -100;
    rect.xmax = 1000;       rect.ymax = 1000;
    ptSet = new CSFeatureSet;

    rtn = ptSFCls->RectSelect(AnyPolygon_Type,&rect,ptSet);      //矩形查询
    if(rtn>0)
    {
        if(ptSet->GetObjCount()>0)
        {
            ptSet->MoveFirst();
            do {
                ptSet->GetObjID(&objid);
            } while(ptSet->MoveNext()!=END_OF_SET);
        }
    }
```

```
    else
        AfxMessageBox("查询结果集为空！");

    delete ptSet;          ptSet = NULL;
}
```

矩形查询，结果放在简单要素集 CSFeatureSet 中，要遍历简单要素集，首先把光标移动到开始的地方，再顺序向下移动光标直到取到最后。

第 4 章 要素类

4.1 要素模型概论

前一章介绍的简单要素类，简单要素的几何形态点、多点、线、多线、多边形、多多边形的实现都是继承于 CGeomBase。MAPGIS7 中另外一种数据模型是复杂要素，
不同于简单要素的模型。

1. 复杂要素的层次图：

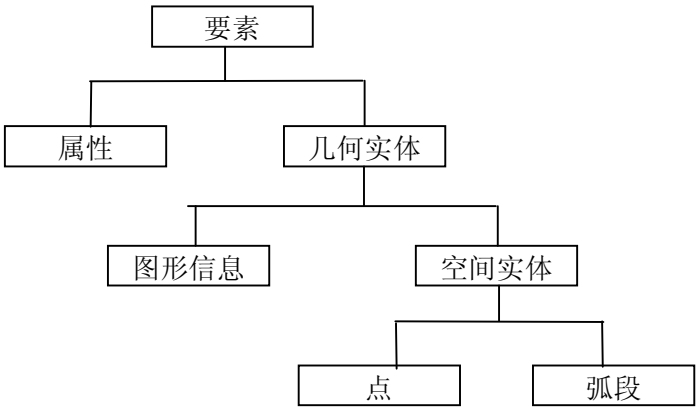


图 4-1 复杂要素概念层次图

2. 数据组织结构图

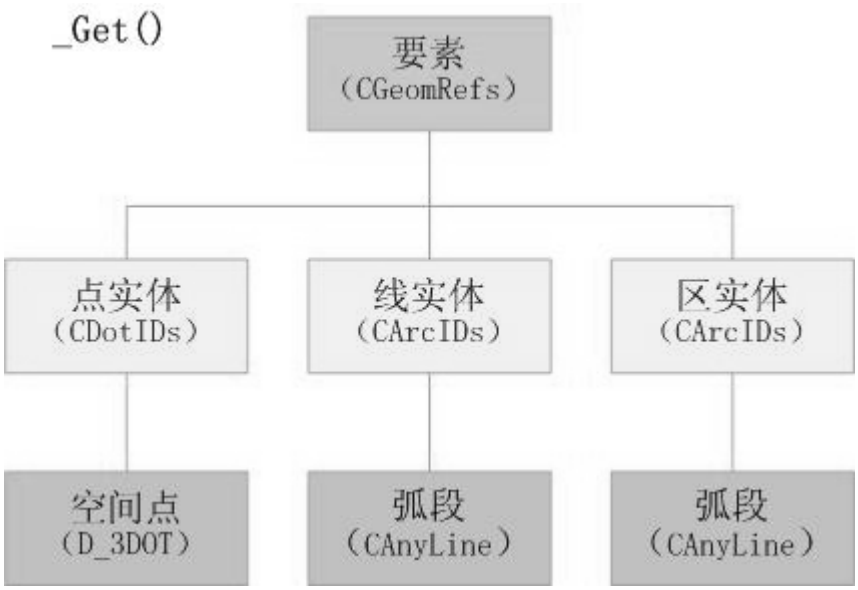


图 4-2 复杂要素数据组织结构图

- 复杂要素的结构图中空间数据层由点和弧段组成，点空间数据分为“一般点”、“实结点”、“虚

结点”。点的坐标存储在 D_3DOT 结构中，其中虚结点是为了实现连通性由系统内部产生的点，它没有对应的地物，也不能引用。弧段是空间数据层中的坐标点序列，包括折线、圆、椭圆、弧、矩形、样条、贝齐尔等，用 CAnyLine 对象来存储。

- 复杂要素的结构图中的几何数据层由空间实体+图形信息组成，图形信息在简单要素类一章中已有介绍，几何数据是通过引用空间数据来实现，比如我们要描述一面状地物，组成面的边界由多条线构成，在简单要素类里我们是用多边形（CAnyPolygon）的实现，复杂要素类的实现不同，先看下面（CArcIDs）的声明：

```
typedef CSmartArray<INT64>          CInt64IDs;
typedef CInt64IDs    CArcIDs;
```

可以把 CSmartArray 看做是记录空间数据（弧段）ID 的一个数组，数组里每个元素就是空间数据的 ID，ID 是 INT64 类型。

- 复杂要素的结构图中的要素层要引用下一层次的几何数据，靠的是CGeomRefs类。

```
class BCLS_EXPORT_CLASS CGeomRefs
{
public:
GEOM_REF *New(long bufLen);
}
```

上面是 CGeomRefs 的声明的部分，里面的 GEOM_REF 结构的声明如下：

```
typedef struct GEOM_REF_stru
{
public:
        GEOM_REF_stru(){
        geomType=GEOM_UNKNOWN_TYPE;  //几何实体类型：点、线、区
        fclsID    = 0;                    //该几何实体所在的要素类 ID
        geomID    = 0;                    //该几何实体的 ID
        infID     =0;                     //图形信息 ID
            }
}GEOM_REF;
```

几何实体所在的要素类 ID，该几何实体的 ID，这些组合形成了要素，

下面的章节会详细介绍如何添加要素到要素类中去。

4.2 如何创建要素类

例 4-1 创建要素类

```
void OnCreatecfls()
{
    CGDBServer          GDBSvr;
    CGDataBase          *ptGDB;                 //存储该要素类的地理数据库
    GDBPATHINFO         path;                   //地理数据库路径信息
    long            ptSelFlag=XCLS_TYPE_GDB;  //文件类型为数据库
    long            selN=1;                         //文件类型个数
```

```
char            user[30]="";
char            pwsd[64]="";

char            fclsType=FCLS_UNION_TYPE;       //要素类的类型    TYPE_FCLS_ID
fclsID;                                 //要素类的 ID
CFeatureCls         *ptFCls=NULL;                   //要素类对象
char            CFlsName[]="MyCFls";            //要素类的名字
long            DsID=0;                         /*要素类所属的要素数据集的 ID，dsID=0
表示要创建的要素类不属于某个要素数据集。*/
long            NetID=0;                //要素类所在的几何网络的 ID
long            SrID=0;                 //空间参照系的 ID，<=0 表示使用缺省参照系
char            AliasName[]="MyAliasName";//要素类的别名，不能超过 128 个字节
char            ModelName[]="MyModelName"; //模板名称，不能超过 128 个字节
char            *ptSubTypeField=NULL;           //子类型的字段名

CATT_STRU           *fclsStru=NULL;   //要素类的属性结构
CFIELD_HEAD         *field=NULL;

//以下是添加字段要属性结构中
field = new CFIELD_HEAD;
fclsStru = new CATT_STRU;
lstrcpy(field->fieldname,_T("MyAtt"));                  //字段名
field->fieldtype        = STR_TYPE;                     //字段类型
field->msk_leng         = 50;                           //字段字符长度
field->edit_enable      =1;                             //字段是否可编辑
field->ptc_pos          = -1;                       //字段在属性结构中的序号，新字段为-1
fclsStru->AppendFld(1,field);                           //添加字段
//字段添加完成

ptGDB = new CGDataBase;
ptFCls = new CFeatureCls;

if(ShowGDBShellDlg(path,&ptSelFlag,selN,0,"选择类")>0) //地理数据库目录选择对话框
{
    GetLoginInfo(path.svcName,user,30,pwsd,64);   //得到对话框中的用户名和密码
    if(GDBSvr.Connect(path.svcName,user,pwsd)>0)            //服务器连接
    {
        if(ptGDB->Open(&GDBSvr,path.gdbName)>0)     //打开地理数据库
            //在打开的地理数据库中创建要素类
            fclsID=ptFCls->fcls_Create(ptGDB,CFlsName,DsID,NetID,SrID,
            AliasName,ModelName,
            ptSubTypeField,fclsStru,fclsType);
    }
}
```

```
    if(fclsID>0)
    {
        ptFCls->fcls_Close();
        AfxMessageBox("创建成功！");
    }
    else
        AfxMessageBox("创建失败！");

    delete ptGDB;   ptGDB=NULL;
    delete ptFCls;    ptFCls = NULL;
    delete field;       field = NULL;
    delete fclsStru; fclsStru = NULL;

}
```

创建要素类和创建简单要素过程相似，这里添加了属性结构的部分，CATT_STRU 是要素类的属性结构，CFIELD_HEAD 是每个字段的具体信息，字段的类型，长度，是否可编辑都记录在 CFIELD_HEAD 里，通过 AppendFld 方法添加到 CATT_STRU 中。

4.3 如何添加要素到要素类。

如果现在需要添加一多边形地物，从复杂要素的模型我们知道首先需要空间数据（线），然后是几何实体（多线），最后是要素，下面是要素类的声明（部分）：

```
class CLN_BAS70_EXPORT_CLASS CFeatureCls : public CFeatureSpace, public CAttributeWrite, public CSubTypeWrite, public CVersionCls, public CValidation
{
    //3. 空间数据数据操作函数：
    TYPE_DOT_ID dot_Append(const D_3DOT &xy, CArcIDs *dat=NULL, char isVirtualNod=0, long userCode=-1);
    TYPE_ARC_ID arc_Append(CAnyLine &arc, TYPE_DOT_ID *ptStNod=NULL, TYPE_DOT_ID *ptEndNod=NULL, long userCode=-1);

    //2. 几何实体数据操作函数：
    TYPE_GEOM_ID gpnt_Append(CDotIDs &dat, PNT_INFO *inf=NULL, long userCode=-1);
    TYPE_GEOM_ID glin_Append( CArcIDs &dat, LIN_INFO *inf=NULL, long userCode=-1);
    TYPE_GEOM_ID greg_Append( CArcIDs   &rdata, REG_INFO *inf=NULL, long userCode=-1);

    //1. 要素操作函数
    TYPE_FID   f_Append(CFeature   &ft);            //添加要素：包括要素的几何引用信息和属性记录。
}
```

dot_Append，arc_Append 添加空间数据，gpnt_Append，glin_Append，greg_Append 添加几何数据，

包含图形参数，f_Append 添加要素，包含属性。
下面举个添加区要素到要素类的示例。
例 4-2 添加要素

```
void WINAPI  AppendFeature(CFeatureCls *ptFCls)
{
    CAnyLine          arc;
    CArcIDs           ArcIDs;
    REG_INFO          rinf;
    D_DOT             xy1[3];
    D_DOT             xy2[2];
    CFeature          feature;
    long          fclsID;

    xy1[0].x = 0;   xy1[0].y = 100;
    xy1[1].x = 0;   xy1[1].y = 0;
    xy1[2].x = 100;     xy1[2].y = 0;

    xy2[0].x = 100;     xy2[0].y = 0;
    xy2[1].x = 0;   xy2[1].y = 100;

    arc.m_linType = ARC_TYPE_BROKEN_LINE;
    arc.m_varLin.Set(&xy1,3);

    TYPE_ARC_ID arcid = ptFCls->arc_Append(arc);
    arc.m_varLin.Set(&xy2,2);
    TYPE_ARC_ID arcid1 = ptFCls->arc_Append(arc);

    ArcIDs.Add(arcid);
    ArcIDs.Add(arcid1);

    memset(&rinf,0,sizeof(REG_INFO));
    rinf.libID = 0;

    rinf.fillclr = 6;                                   //区域填充色
    rinf.endclr = 0;                                    //区域填充结束色
    rinf.patID = 8;
    rinf.pathei = 5;                                    //图案高
    rinf.patwid = 5;                                    //图案宽

    REG_INFO_ID    rinfID=ptFCls->rinf_Append(rinf);
    TYPE_GEOM_ID gregID=ptFCls->greg_Append(ArcIDs,rinfID);

    fclsID    =ptFCls->GetSpaceID();
    feature.m_geom.New(1);
```

```
    feature.m_geom[0].geomType=GEOM_REG_TYPE;
    feature.m_geom[0].geomID = gregID;
    feature.m_geom[0].fclsID = fclsID;
    feature.m_geom[0].infID = rinfID;

    CATT_STRU  stru;
    if(ptFCls->att_GetStru(stru)>0)
    {
        feature.m_att.SetStru(stru);
        for(int i=0;i<stru.hd.numbfield;i++)
            feature.m_att.SetFldFromStr(i,"Test");
    }

    TYPE_FID fID = ptFCls->f_Append(feature);
}
```

定义 CAnyLine 作为空间数据弧段，通过 arc_Append 添加两条弧段到要素类，通过 greg_Append 函数引用这两条弧段的 ID 和图形参数构成几何实体，同时生成几何实体的 ID：gregID，最后引用 gregID 和要素的属性完成要素的添加，这个示例可以帮助理解复杂要素的概念层次图。

4.4 要素类的其它操作

- 取要素信息操作

  理解了复杂要素的模型的数据组织结构后，我们可以通过相应的成员函数来获取要素的引用信息和坐标，或者用更新函数来更改要素的引用信息。

例 4-3 取要素引用信息和坐标

```
Void GetFeaturePos(CFeatureCls* fCls, TYPE_FID fID)
{
    //一、直接通过要素类成员函数 f_GetGeomPos 取坐标
    CPolyGeometry               geoms;
    fCls->f_GetGeomPos(fID,geoms);
    D_DOT *xy = geoms.ptXY();           //D_DOT *xy 是要素的空间坐标信息

    //二、通过要素类成员函数 f_Get 取(几何引用－>空间引用)
    CFeature      f;
    GEOM_REF      *ptGeomRef;           //要素的几何实体信息
    long          geomNum;              //要素的几何实体的个数
    CDotIDs           dat;              //几何引用的空间点数据对象 ID 序列
    CArcIDs           ldat,rdat;        //几何引用的空间弧段数据对象 ID 序列

    int               j,k;
```

```
int                 num;
D_3DOT              dot;
CVarLine    ldot,rdot;                  //线的折线坐标

if(fCls->f_Get(fID,f)>0)                //得到一个要素
{
    geomNum = f.m_geom.GetGeomNum();        //取要素引用几何实体的个数
    ptGeomRef = f.m_geom.GetBufPtr();       //取要素的引用几何实体的信息
    //循环操作几何实体
    for(int i=0;i<geomNum;i++)
    {
        switch(ptGeomRef[i].geomType)
        {
            case GEOM_PNT_TYPE:             //判断几何实体的类型
                //取点实体引用的空间点数据对象 ID 序列
                fCls->gpnt_Get(ptGeomRef[i].geomID,dat);
                num = dat.GetSize();
                for(j=0;j<num;j++)
                {
                    //dot 是点数据对象 ID 为 dat[j]的坐标
                    fCls->dot_GetPos(dat[j],dot);
                }
                break;
            case GEOM_LIN_TYPE:
                //取线实体引用的空间弧段数据对象 ID 序列
                fCls->glin_Get(ptGeomRef[i].geomID,ldat);
                num = ldat.GetSize();
                if(num!=0)
                {
                for(j=0;j<num;j++)
                {
                    fCls->arc_GetPos(abs(ldat[j]),ldot);    //取弧段坐标信息

                    for(k=0;k<ldot.dotNum();k++)
                    {
                        double x = ldot.GetX(k);        //x,y,z 是坐标信息
                        double y = ldot.GetY(k);
                        double z = ldot.GetZ(k);
                    }
                }
                }
                break;
            case GEOM_REG_TYPE:
                //取区实体引用的空间弧段数据对象 ID 序列
```

```
                    fCls->greg_Get(ptGeomRef[i].geomID,rdat);
                    num = rdat.GetSize();
                    if(num!=0)
                    {
                    for(j=0;j<num;j++)
                    {
                        fCls->arc_GetPos(abs(rdat[j]),rdot);     //取弧段坐标信息
                        for(k=0;k<rdot.dotNum();k++)
                        {
                            double x = rdot.GetX(k);             //x,y,z 是坐标信息
                            double y = rdot.GetY(k);
                            double z = rdot.GetZ(k);
                        }

                    }
                    }
                    break;
                default:
                    ;
            }

        }
    }
}
```

示例中取要素的坐标有两种方法，一种是通过 f_GetGeomPos 直接返回坐标信息，另外一种通过取要素的引用信息取空间数据的坐标点序列。从 f_Get 到 greg_Get 到 arc_GetPos 一层层的往下取。

属性操作

例 4-4 更新要素的属性

```
void UpdateFeatureAtt(CFeatureCls* fCls, TYPE_FID fID)
{
    CFeature    f;                                          //要素
    CRecord     attRcd;                                     //属性记录
    CATT_STRU   stru;                                       //属性结构
    char        strtemp[]="test";

    if(fCls->f_Get(fID,f)>0)
    {
        if(f.m_att.GetStru(stru)>0)                         //得到要素的属性结构
        {
            for(int i=0;i<stru.hd.numbfield;i++)            //循环每个字段
            {
                long rtn = f.m_att.SetFldFromStr(i,strtemp); //设字段的值
```

```
                attRcd = f.m_att;
                fCls->att_Update(fID,attRcd);          //更新要素的属性
            }
        }
    }
}
```

更新要素属性，GetStru 得到属性结构，循环设置每个字段的值，修改后的 f.m_att 用 att_Update 更新。

- 查询

  查询接口提供了 f_AttSelect（属性查询）f_RegSelect（区域查询） f_RectSelect（矩形范围查询） f_Near（临近查询）四种查询方式，这里举属性查询的示例。

例 4-5 属性查询

```
void FAttSelect(CFeatureCls* fCls)
{
    char            condition[]="test='test'";
    long            subTypeCode=-1;  //子类型字段的值，该参数缺省为-1
    char            orderByFld[]="OID";                    //结果排序字段名
    char            isAsc=1;
    CATT_STRU       *ptStru=NULL;
    CConditionOBJ   *ptConObj=NULL;                        //查询条件
    CFeatureSet     *ptSet=NULL;                           //返回查询结果集
    TYPE_OBJ_ID     objid;

    ptConObj = new CConditionOBJ;
    ptSet = new CFeatureSet;
    ptConObj->SetAttCondition(condition);


    if(fCls->f_AttSelect(ptConObj, ptSet, subTypeCode, orderByFld, isAsc)>0)    //属性查询
    {
        if(ptSet->GetObjCount()>0)
        {
            ptSet->MoveFirst();
            do {
                ptSet->GetObjID(&objid);                   //取结果集里的所有对象
            } while(ptSet->MoveNext()!=END_OF_SET);
        }
    }

    delete ptConObj;    ptConObj=NULL;
    delete ptSet;       ptSet=NULL;
```

}

示例是属性条件查询，属性条件记录到 CConditionOBJ 对象里，满足此条件的要素存放在要素集 CFeatureSet 中，可以通过光标的移动来得到结果集里的所有对象的 ID。

4.5 小结

简单要素类是直接存储简单点，简单线，简单多边型。要素类是按照要素—几何实体—空间数据三个层次存储空间信息，有更强的表达能力，可描述几何形态较为复杂的地理实体。
要素类弧段的拓扑信息用 arc_WriteTop 写入，具体的参阅要素类声明。

第 5 章 对象类和关系类

5.1 对象类

对象类是具有相同行为和属性的对象的集合。在空间数据模型中，一般情况下，对象类是指没有几何特征的对象（如房屋所有者、表格记录等）的集合；

5.1.1 对象类名词解释

先看对象类的声明（部分）

```
class CLN_BAS70_EXPORT_CLASS CObjectCls : public CAttributeSpace, public CAttributeWrite, public
        CSubTypeWrite, public CVersionCls, public CValidation
{
    //添加
    virtual TYPE_OBJ_ID att_Append(CATT_STRU *stru, char *att);
    virtual TYPE_OBJ_ID att_Append(CRecord &rcd);
    //属性结构
    virtual long att_SetStru(CATT_STRU &stru);
    virtual long att_GetStru(CATT_STRU &stru);
    //子类型
    long subT_SetSubTypeField(char *ptFldName);
}
```

CATT_STRU 和 CRecord 在以前的章节里曾经使用过，CATT_STRU 是属性结构，CRecord 是属性记录。如果我们需要添加一个字段到属性结构中可以这样做：

```
    CATT_STRU           *fclsStru=NULL;                 //要素类的属性结构
    CFIELD_HEAD         *field=NULL;

    //以下是添加字段要属性结构中
    field = new CFIELD_HEAD;
    fclsStru = new CATT_STRU;
    lstrcpy(field->fieldname,_T("MyAtt"));                  //字段名
    field->fieldtype        = STR_TYPE;                     //字段类型
    field->msk_leng          = 50;                          //字段字符长度
    field->edit_enable     =1;                              //字段是否可编辑
    field->ptc_pos           = -1;                      //字段在属性结构中的序号，新字段为-1
    fclsStru->AppendFld(1,field);                           //添加字段
    //字段添加完成
```

CFIELD_HEAD 记录每个字段的具体信息，包括字段名、字段类型，字段的长度等，如果需要添加

多个字段，可以定义 CFIELD_HEAD 的数组，数组的每个元素都是单个字段的具体信息。在 CATT_STRU 类中还有个成员变量 CINFO_HEAD，CINFO_HEAD 记录了属性结构的整体信息，如属性结构拥有字段的个数，要遍历取每个字段就可以根据字段的个数。

CRecord 是属性记录，可以根据字段序号和字段名两种方式设置字段的值。

5.1.2 对象类示例

例 5-1-1 取对象类属性结构

```
void   OnGetCObjAttStru(CObjectCls* ptCObjectCls)
{
    CATT_STRU  stru;                          //对象类属性结构
    CString        str;

    if(ptCObjectCls->att_GetStru(stru)>0)                    //取属性结构
    {
        for(int i=0;i<stru.hd.numbfield;i++)                 //循环字段
        {
            str.Format("%s",stru.fldEntry[i].fieldname);     //取字段名
            MessageBox(str);
        }
    }
}
```

att_GetStru 方法返回对象类的属性结构，属性结构中的 hd.numbfield 是当前结构中字段的个数，由于字段的序号是从 0 开始，循环每个字段，取出序号为 i 的字段名（stru.fldEntry[i].fieldname）。fldEntry 就是前面所讲的 CFIELD_HEAD 对象。

例 5-1-2 设置对象类属性结构

```
void   OnSetcobjattstur(CObjectCls* ptCObjectCls)
{
    CATT_STRU       stru;                                    //对象类的属性结构
    CFIELD_HEAD         fldhead;                             //字段信息
    CExtFIELD_HEAD      extfldhead(LONG_TYPE,1);             //字段的扩展信息
    TYPE_XCLS_ID    dmn_ID = 1;                              //域集 ID

    lstrcpy(fldhead.fieldname,_T("APPENDFIELD"));            //字段名
    fldhead.fieldtype = LONG_TYPE;                           //字段类型
    fldhead.ptc_pos = -1;                                //字段在属性结构中的序号
    fldhead.msk_leng = 10;                                   //字段字符长度

    lstrcpy(extfldhead.m_alias,_T("MyField"));               //字段扩展信息
    extfldhead.m_IsNull = 1;
    extfldhead.SetDefVal(10);
    extfldhead.SetMinVal(1);
    extfldhead.SetMaxVal(90);
```

```
    extfldhead.m_dmnID = dmn_ID;                                    //所属域集 ID


    fldhead.SetExtFIELD_HEAD(&extfldhead);
    if(stru.AppendFld(1,&fldhead)>0)                                //添加字段
    {
        if(ptCObjectCls->att_SetStru(stru)>0)                       //设置对象类属性结构
        AfxMessageBox("对象类属性结构创建成功！");
    }

}
```

att_ SetStru 方法设置属性结构，先向属性结构中加入一个长整型的字段，并且设置了字段的扩展信息，扩展信息限制此字段的值在 1 到 90 之间。

■ 例 5-1-3 添加记录（所有字段）

```
void   OnAppendRecordALL(CObjectCls* ptCObjectCls)
{

    CATT_STRU  stru;
    TIME_STRU  timeval;                                     //时间记录结构体
    short          sht=100;
    char       str[]="zondy";
    char       chr='A';
    __int64        i64=1000;
    DATE_STRU  date;


    //设置时间
    timeval.hour = 23;
    timeval.sec = 30;
    timeval.min = 20;

    date.day = 23;
    date.mon=1;
    date.year=2006;

    if(ptCObjectCls->att_GetStru(stru)>0)               //取对象类的属性结构
    {
        CRecord        rcd(&stru);
        for(short i=0;i<stru.hd.numbfield;i++)
        {
            switch(stru.fldEntry[i].fieldtype)          //判断字段的类型
```

```
            {
            case LONG_TYPE:
                rcd.SetLongVal(i,1000);                 //设置长整型字段的值
                break;
            case TIME_TYPE:
                rcd.SetTimeVal(i,timeval);              //设置时间型字段的值
                break;
            case SHORT_TYPE:
                rcd.SetShortVal(i,sht);                 //设置短整型字段的值
                break;
            case BYTE_TYPE:                                 //设置字节类型字段的值
                rcd.SetCharVal(i,chr);
                break;
            case INT64_TYPE:
                rcd.SetInt64Val(i,i64);                 //设置 64 位长整型字段的值
                break;
            case DATE_TYPE:                                 //设置日期型字段的值
                rcd.SetDateVal(i,date);
                break;
            default:
                rcd.SetFldNull(i);
            }
        }
        if(ptCObjectCls->att_Append(rcd)>0)             //添加记录到对象类中
            AfxMessageBox("记录添加成功！");

    }
}
```

att_Append 添加属性记录到对象类中，需要判断字段的具体类型来调用相对应的设字段值的函数，stru.fldEntry[i].fieldtype 是字段的类型，在 CRecord 的构造函数中传入属性结构对象，如果是传入对象类的属性结构，添加的属性记录是所有字段都赋值，也可以传入对象类属性结构的部分给部分字段赋值。

l 例 5-1-4 记录查询

```
void OnRecordSelect(CObjectCls* ptCObjectCls)
{
    //查询（有条件表达式对话框）
    TYPE_OBJ_ID         ObjID;                                  //对象 ID
    CConditionOBJ   condition;                                  //查找条件对象
    CObjectSet          ObjSet;                                 //对象集合
    CATT_STRU       stru;
    char            expStr[255];
    long            len=255;
```

```
    memset(&stru,0,sizeof(CATT_STRU));

    ptCObjectCls->att_GetStru(stru);
    if(Tool_ShowAttExpSel(stru,"输入表达式",expStr,&len)>0)
    {
        condition.SetAttCondition(expStr);
        if(ptCObjectCls->Select(&condition,&ObjSet)>0)
        {
            if(ObjSet .GetObjCount()>0)
            {
                ObjSet .MoveFirst();
                do {
                    ObjSet .GetObjID(&ObjID);          //取结果集里的所有对象
                } while(ptSet.MoveNext()!=END_OF_SET);
            }
    }
}
```

Tool_ShowAttExpSel 弹出属性条件表达式输入对话框，输入的条件表达式用 CConditionOBJ 对象存放，对象类的 Select 方法返回满足条件的结果集，光标的移动循环取结果集的对象 ID。

子类型

子类型是对象类的轻量级分类，以表达相似对象，如供水管网中区分钢管、塑料管、水泥管。不同类或子类型的对象可以有不同的属性缺省值和属性域。

对象类的子类型是让你为对象的不同类别指定不同简单行为的特殊的属性。对象类的所有子类型共享相同的属性集；可以为子类型中的每一个字段定义不同的属性值域；也可以为子类型中的每一个字段定义不同的缺省值；

例 5-1-5 设置子类型

```
void    OnSetSubTypeFld(CObjectCls* ptCObjectCls)
{
    CATT_STRU  stru;
    if(ptCObjectCls->att_GetStru(stru)>0)
    {
        //设置子类型对象，此字段类型必须是长整或短整型
        //参数传 NULL 或""则取消子类型字段设置
        if(ptCObjectCls->subT_SetSubTypeField(stru.fldEntry[0].fieldname)>0)
            AfxMessageBox("子类型对象设置成功！");
    }
}
```

通过 subT_SetSubTypeField 为属性结构中第一个字段设置子类型，设置子类型的字段必须是长整或短整型，如果要取消子类型字段设置，参数传入 NULL。

5.2 关系类

5.2.1 关系类名词解释

- 关系

关系是指地理数据库中两个或多个对象之间的联系（association）或连接（link）。

关系可以存在于空间对象之间、非空间对象之间、空间和非空间对象之间，对象之间的关系可分为空间关系和非空间关系。

空间关系是与实体的空间位置或形态引起的空间拓扑关系，包括：分离、包含、相接、相等、相交、覆盖等九种。

非空间关系是对象的语义引起的关系，包括：关联、继承(完全、部分)、组合（聚集、组成）、依赖。

非空间关系具有多重性：1－1、1－M、N－M。

关系的集合称为关系类。

- 关联关系

关联关系也叫一般关系。如果多个对象之间互相独立，不存在依赖，则这种关系称为一般关系。一般关系有 1-1、1-M 和 N-M 的映射。见下图：

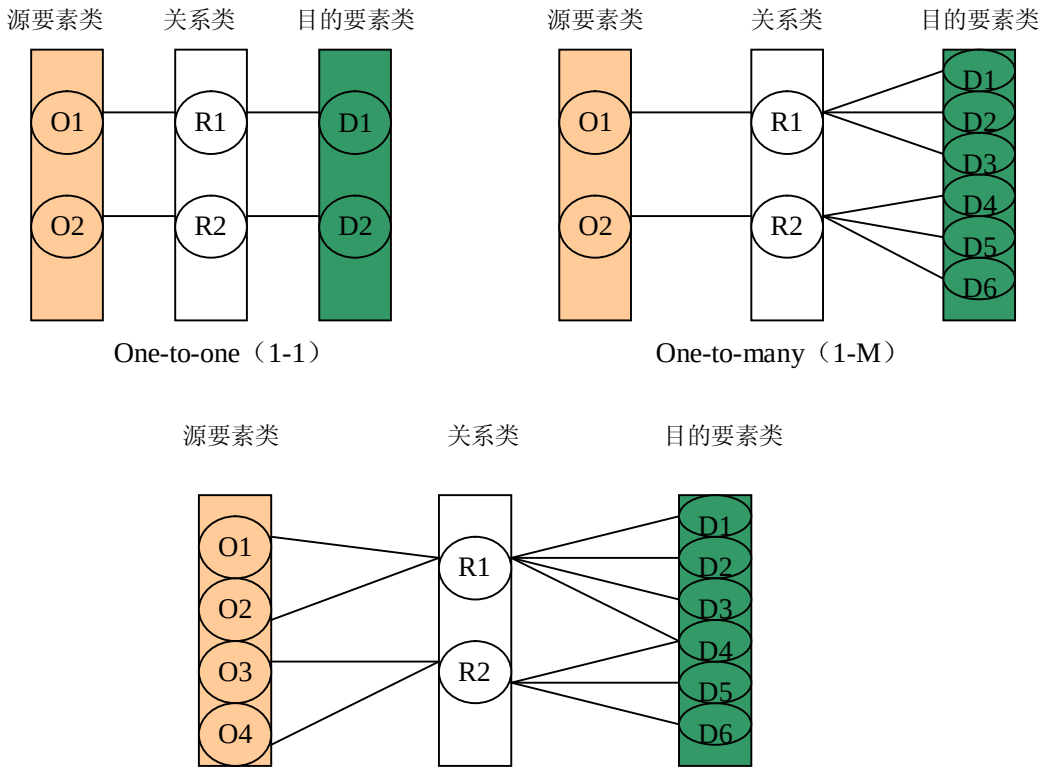


图 5-1　1-1、1-M 和 N-M 的映射图

- 继承关系

继承关系是指目的对象类继承原始对象类的某些或全部特性。继承关系分为部分继承和完全继承。

部分继承是指子类继承了父类的部分特性。

完全继承是指子类继承了父类的全部特性。

- 组合关系

如果一个对象控制其它对象的生命，则这种关系称为组合关系。例如：电线杆（原始类）和变压器（目的类）是一个一对多的组合关系，一旦电线杆被删除，一个删除消息就会发送给与它相关联的变压器，随之变压器也从其要素类中被删除。
组合关系的映射只能是 1-1 或 1-M。
组合关系分为聚集和组成。
聚集是指组合体与各部分具有不同的生命期。
组成是指组合体与各部分具有相同的生命期（同生共死）。

- 依赖关系

依赖关系由对象的语义引起，但与空间有关，如要素类 1 中的弧段 A 引用了要素类 2 中的弧段 B，则成为 A 依赖于 B，如行政边界以河流中心线为准。

- N—M 的关系和属性化的关系

N-M 的关系需要独立的表格（关系表）来存储关系。
属性化（表格化）的关系是指两对象间的关系还需要用额外的属性字段来表现它们间的联系，这种关系也需要建立独立的表格（关系表）。例如：在土地和所有者的关系中，若某个所有者只拥有某块土地 20%的股份，对于这个属性，它即不能存在于所有者要素类中，也不能存在于土地要素类中，它是在两个要素类有了关系之后才存在，所以我们需要创建一个独立的表格，来存放这样的属性。

- 标签

标签分为前向标签和后向标签。这两个标签用于描述原始对象类和目的对象类之间的关系，例如：定义前向标签为“管理”，后向标签为“被管理”。

- 通知

通知是作用在原始类和目的类上的一个管理机制，它分为四种类型：不通知、向前通知、向后通知和互相通知。

5.2.2 如何创建关系

例 5-2-1 创建关系类

```
Void OnCreatecrelationcls()
{
    CRelationCls  *ptCRelationCls;                          //关系类对象指针
    CGDBServer        GDBSvr;
    CGDataBase        *ptGDB=NULL;
    GDBPATHINFO          path;
    long              ptSelFlag=XCLS_TYPE_GDB;
    char              user[30]="";
    char              pwsd[64]="";
    char               ptRclsName[]="MyRelationCls";          //关系类名称
    long              dsID=0;                               //要素数据集 ID
    TYPE_XCLS_ID   origClsID=4;                             //原始类 ID
    TYPE_XCLS_ID   destClsID=5;                             //目的类 ID
    short                origClsType=XCLS_TYPE_FCLS;        //原始类类型码
```

```
short           destClsType=XCLS_TYPE_FCLS;          //目的类类型码
char        ptFwardLabel[]="FwardLabel";             //向前标签
char        ptBwardLabel[]="BwardLabel";             //向后标签
short           cardinality=REL_CARD_1_1;            //映射类型
short           notification=REL_NOTIFY_BOTH;        //通知类型
char        relType=REL_TYPE_ASSOCIATE;              //关系类类型
char        isAttributed=0;                        //指定关系类是否属性化
CATT_STRU       *stru=NULL;                          //属性结构
char        origPKey[]="OID";                        //原始类主键
char        destPKey[]="OID";                        //目的类主键
char        origFKey[]="q";                          //原始类外键
char        destFKey[]="";                           //目的类外键
TYPE_OCLS_ID    clsid=0;

ptGDB = new CGDataBase;
ptCRelationCls = new CRelationCls;

if(ShowGDBShellDlg(path,&ptSelFlag,1)>0)
{
    GetLoginInfo(path.svcName,user,30,pwsd,64);
    if(GDBSvr.Connect(path.svcName,user,pwsd)>0)
        if(ptGDB->Open(&GDBSvr,path.gdbName)>0)          //打开地理数据库
        {
            //创建关系类
            clsid = ptCRelationCls->Create(ptGDB, ptRclsName, dsID,
                origClsID, destClsID, origClsType, destClsType,
                ptFwardLabel, ptBwardLabel, cardinality,
                notification, relType, isAttributed,
                stru, origPKey, destPKey,
                origFKey, destFKey);
        }
}

if(clsid>0)
    AfxMessageBox("关系类创建成功！");
else
    AfxMessageBox("关系类创建失败！");
```

```
    delete    ptCRelationCls;    ptCRelationCls=NULL;
    delete    ptGDB;             ptGDB=NULL;
}
```

在原始类 origClsID 和目的类 destClsID 创建一对一的关联关系。

例 5-2-2 取关系类型

```
void   OnGetCRelationclsType(CRelationCls* ptCRelationCls)
{
    char      crlsType;
    if(ptCRelationCls->GetRelationType(crlsType)>0)
    {
        switch(crlsType)
        {
        case REL_TYPE_ASSOCIATE:
            AfxMessageBox("关联关系");
            break;
        case REL_TYPE_COMPOSITE:
            AfxMessageBox("组合关系");
            break;
        case REL_TYPE_DEPENDENCE:
            AfxMessageBox("依赖关系/引用关系");
            break;
        case REL_TYPE_INHERITED:
            AfxMessageBox("继承关系");
            break;
        case REL_TYPE_META:
            AfxMessageBox("元关系");
            break;
        default:
            ;
        }
    }
}
```

第 6 章 注记类

6.1 注记类名词解释

- 注记

表现地理现象的地理要素除了有几何形状和空间位置外，还有一些描述文本，例如表示城市的要素类具有与其名称相关的文本，通常将这些文本称为注释。在地理数据库中注记是在制图显示时用来标注要素的文本，它可以确定位置或者识别要素。注记按类型分为静态注记、属性注记和维注记三种。

- 静态注记

静态注记的文字内容和注记位置均由用户输入。根据版式的不同，静态注记又分为静态文本注记和 HTML 版面注记。

- 静态文本注记

静态注记的文字内容和注记位置均由用户输入。根据版式的不同，静态注记又分为静态文本注记和 HTML 版面注记。

- HTML 版面注记

HTML 版面注记也是一种静态注记，它可以用来描述地图上某些现象的信息。HTML 所代表的意义是静态超文本标记语言，它是全球广域网上描述网页内容和外观的标准。标记描述了每个网页上的组件，例如文本段落，表格或图像。对于地图上需要用一段文字描述的信息，就可以用 HTML 版面注记来标注。用 HTML 方便了控制版面的格式。

- 属性注记

属性注记是地理数据库中与要素属性字段相关联的注记。属性注记所在的注记类与同一个要素数据集下的一个(且只能是一个)要素类相关联，每一个属性注记和要素类中的一个要素相关联。属性注记的文本可以是要素属性表中的一个字段或者多个字段合成的文本信息。

属性注记的特性有如下一些：

1．属性注记类与某个要素类相关联。用户可以选定要素类的全部或者部分要素，在关联的属性注记类中为这些要素添加相应的记录。

2．当要素类的要素移动时，属性注记类的注记也随之移动，属性注记的这种行为在注记类创建的时候就产生了，并且是默认存在的。

3．当我们改变要素的某个属性，该属性正是要素与属性注记相关联的字段，那么属性注记的文本内容也会随之发生改变。

4．当要素被删除时，与之建立关系的属性注记也同时被删除；若与要素对应的注记被删除，则要素不受影响。

5．属性注记类的文本内容不能被修改。

- 维

维是一种特殊的地图注记，用来表示地图上的长度或者距离，一个维注记可能表示一个建筑或小区某一边界的长度，或者表示两个要素之间的距离，例如一个消防栓到某一建筑拐角的距离。

6.2 注记类示例

例 6-1 添加注记

```
void OnAppendAnnotation(CAnnotationCls* ptCAnnotationCls)
{
/*添加文本注记*************************************

    D_DOT          xy;                          //注记位置
    CAnnDat        dat;                        //注记信息封装类
    TYPE_FID       fID=0;                       //与注记关联的要素 id
    CRecord        *ptRcd=NULL;                 //注记的属性记录（可以为 NULL，CRecord
用法可以参照对象类一章）
    char           ptTxt[]="Zondy";             //注记的内容
    ANN_STR_INFO   strinfo;                     //注记的可视化信息
    memset(&strinfo,0,sizeof(ANN_STR_INFO));

    //初始化注记的可视化信息结构
    strinfo.height=5;
    strinfo.width=5;
    strinfo.iclr=5;

    xy.x=100;
    xy.y=100;

    dat.m_type = ANN_TYPE_STR;       //注记类型为字符串注记类型（必须先指定类型）
    dat.SetTxt(ptTxt);
    dat.SetInfo((char*)&strinfo);

    TYPE_AID AID = ptCAnnotationCls->Append(&xy, dat, fID, ptRcd);     //添加注记
    if(AID>0)
        AfxMessageBox("添加注记成功！");
    else
        AfxMessageBox("添加注记失败！");
    ************************************************************/
```

```
///*添加 html 注记**********************
    D_DOT           xy;                         //注记位置
    CAnnDat         dat;                        //注记信息封装类
    TYPE_FID        fID=0;                      //与注记关联的要素 id
    CRecord         *ptRcd=NULL;                    //注记的属性记录（可以为 NULL，CRecord
用法可以参照对象类一章）
    char            ptTxt[]="<p><strong><em><u><strike><font
color='#ff0080'>zondy</font></strike></u></em></strong></p>";
    ANN_HTML_INFO   htmlinfo;                   //注记的可视化信息
    TMPL_INFO       tmplinfo;
    memset(&htmlinfo,0,sizeof(ANN_HTML_INFO));
    memset(&tmplinfo,0,sizeof(TMPL_INFO));

    //初始化注记的可视化信息结构
    htmlinfo.height=20;
    htmlinfo.width=20;

    //初始化注记模板信息
    tmplinfo.iclr=5;

    xy.x=100;
    xy.y=100;

    dat.m_type = ANN_TYPE_HTML;     //注记类型为 HTML 注记类型（必须先指定类型）
    dat.SetTxt(ptTxt);
    dat.SetInfo((char*)&htmlinfo);
    dat.SetTmplInfo(tmplinfo);

    TYPE_AID AID = ptCAnnotationCls->Append(&xy, dat, fID, ptRcd);     //添加注记
    if(AID>0)
        AfxMessageBox("添加注记成功！");
    else
        AfxMessageBox("添加注记失败！");

//*********************************/
}
```

添加注记到指定的注记类里，注记支持 html 语言，对注记的格式和版面可以用 html 语句来控制，上面的示例分别添加了文本注记和 html 注记。

例 6-1 取注记信息

```
Void OnGetAnnotationInfo(CAnnotationCls* ptCAnnotationCls,TYPE_AID id)
{
    D_DOT       xy;                                     //注记坐标
    D_RECT      rc;                                     //注记外包矩形
    TYPE_FID    fID;                                  //和注记相关的要素 ID
    CAnnDat     dat;                                 //注记数据信息（具体参阅 basclass70.h）
    char        *str;


    if(ptCAnnotationCls)
    {
        if(ptCAnnotationCls->Get(id,&xy, &rc, &fID,dat)>0)        //取注记信息
            str = dat.GetTxt();                    //CAnnDat 参阅 basclass70.h 或添加注记代码
            CString cstr;
            cstr.Format("取注记信息成功！内容是%s",str);
            MessageBox(cstr);
    }

}
```

第 7 章 规则和 GSQL

7.1 规则名词解释

对象特性的一个特殊表现是某些属性的取值往往存在边界条件，对象之间的关系（包括空间关系）甚至关系本身存在某种约束条件。所有这些限制条件统称为有效性规则。MAPGIS7 中，有效性规则分为 4 种类型：属性规则，空间规则，连接规则，关系规则。有效性规则可以作用在类上，也可以作用在子类型上。

- 属性规则与定义域

属性规则用于约定某个字段的缺省值，限定取值范围，设置合并和拆分策略。属性规则通过“定义域”来表达，取值范围分连续型和离散型，相应地把定义域分为范围域和编码域。

范围域适用于数值型、日期型、时间型等可连续取值的字段类型，编码域除了可以适用于连续取值类型外，还可用于字符串等类型的字段。

合并和拆分策略定义要素合并和拆分时属性字段的变化规则，合并策略包括：缺省、累加、加权平均，拆分策略包括：缺省、复制、按比例。如地块合并，合并后的要素属性“地价”可定义为“累加”策略。

- 空间规则

空间规则作用于要素类或要素类之间，用于限定要素在某个空间参照系中的相互关系。空间规则如：

(1) 要素类中每条弧段只能作为两个多边形的边界；
(2) 要素类中多边形之间不能重叠；
(3) 要素类中多边形之间不能有缝隙；
(4) “城镇”要素必须落在“行政区”要素内部；
(5) 不能有悬挂线；
(6) 线不能自相交；

- 关系规则

关系规则随着关系的产生而产生，用于限定对象之间关系映射的数目。例如：原始类和目的类之间建立了 N－M 的关系，则通过关系规则可以限定关系的原始对象数是 1－3，目的对象数是*－5，即原始类中的每个对象与目的类中至少 1 个、最多 3 个对象建立关系；而目的类中的对象可以和原始对象没关系，但最多只能与 5 个原始对象有关系。

- 连接规则

连接规则主要使用在几何网络中，用以约束可能和其它要素相连的网络要素的类型，以及可能和其他任何特殊类型相连的要素的数量。有两种类型的连接规则：边对边连接规则、点对边连接规则。

边对边规则约束了哪一种类型的边通过一组结点可以与另一种类型的边相连。

点对边规则约束了哪一种边类型可以和哪一种结点类型相连。

7.2 规则示例

例 7-1 要素类添加拓扑规则()

```
void CCAnnotationClsView::OnAddRule(CFeatureCls* ptCFeaCls)
{
    CTopRule      toprule;                                          //拓扑规则类

    X_CLS_FCLS fclInf;
    CGDataBase   *GDB;                                    //地理数据库
    GDB = ptCFeaCls->GetGDataBase();                      //取要素类的地理数据库
    if(ptCFeaCls->fcls_GetInfo(fclInf)>0)
    {
        if(fclInf.ftype==FCLS_LIN_TYPE)          //判断要素类的类型是否是线要素类
        {
            toprule.origClsID=fclInf.ID;                  //定义原始类的 ID
            toprule.destClsID=fclInf.ID;                  //定义目的类的 ID
            toprule.topType=LINE_NO_INTERSECTORINTERIORTOUCH;    //定义拓扑规则的类型
            TYPE_RULE_ID ruleID = GDB->rule_Add(fclInf.ID,&toprule);//添加规则
            if(ruleID>0)
                AfxMessageBox("拓扑规则添加成功");
        }
    }
}
```

规则定义后用规则有效性检查来查找不符合要求的对象。

例 7-2 规则有效性检查

```
void   OnValidateRule(CFeatureCls* ptCFeaCls,TYPE_RULE_ID ruleID)
{
    CSmartArray<RULE_ERR_INFO>      err;                  //不符合规则的 ID
    RULE_ERR_INFO                   errinfo;
    if(ptCFeaCls->rule_ValidateCls(ruleID,err)>0)         //规则有效性检查
    {
        for(int i=0;i<err.GetSize();i++)                  //循环取不符合规则的 ID
            errinfo = err.GetAt(i);
    }
}
```

7.3 GSQL

空间数据查询语言（Geometric Structured Query Language），是对数据库标准查询语言 SQL 的扩展，以仿照 SQL 脚本的形式实现空间数据的查询和分析。

- 属性查询

任何一个 GIS 软件都要管理两类数据：空间数据和属性数据。属性数据是反映事物相关特性的数据，用来描述对象的特征，属性数据通过对象类或要素类的属性表进行表示，以属性字段和属性记录的形式体现，对这些属性数据的查询称为属性查询。

- 空间查询

空间数据就是在某个空间框架（例如地球表面）中对象的位置数据，是反映事物地理位置有关信息的数据与的属性数据一起共同说明对象的总体特征。空间数据是与实体的地理位置有关的查询称为空间查询。

- 空间分析

利用计算机对数字地图进行分析。空间分析是 GIS 的核心和灵魂，是 GIS 区别于一般的信息系统、CAD 或者电子地图系统的主要标志之一。空间分析包括以下的内容：基于空间关系的查询、空间量算、缓冲区分析、网络分析、叠加分析、空间统计分类分析等。由于网络分析单独成为一个独立的模块，因此，本 GSQL 查询分析器中并不包含网络分析。

- 语法规则

编辑语句时必须遵守的规范。因为 GSQL 是一种通过解释方式来执行的语言，因此，它必须遵从特定的解释规范才能被正确执行。GSQL 语句的语法规则以 SQL 的语法规则为基础进行扩展，加入了对空间数据类型和空间分析函数的支持，SQL 的语法收录在 MAPGIS7 使用手册中。

例 7-3 GSQL

```
void OnGsqExecute(CGDataBase *ptOriGDB,CGDataBase *ptDesGDB)
{
    //需要执行的 GSQL 语句
    char gsql[]="SET @CLS-SC-ROAD = SELECT ATT FROM FCLS.SC-ROALK AS ROALK WHERE RN LIKE 'G%' ORDER BY RN;";
    char errStr[255];
    CGSQLBlock gsqlblock;
    if(gsqlblock.gsql_CreatEnv(ptoriGDB,ptdesGDB)>0)                    //分配 GSQL 环境
        if(gsqlblock.gsql_Prepare(gsql,sizeof(gsql))>0)                 //准备执行 GSQL 语句
            if(gsqlblock.gsql_Execute(0)>0)                             //执行 GSQL 语句
            {
                gsqlblock.gsql_ReleaseEnv();                            //释放 GSQL 环境
                AfxMessageBox("GSQL 执行成功！");
            }
            else
            {
                gsqlblock.gsql_GetErrStr(errStr);                       //得到执行 GSQL 语句的错误信息

                AfxMessageBox(errStr);
                gsqlblock.gsql_ReleaseEnv();

            }

    }
```

第 8 章 应用框架

8.1 概述

MAPGIS7.0 采用全组件化结构，为了方便应用软件开发，系统设计了一个全新的应用开发框架模型，使用的当前非常流行的平台+插件组成框架，插件思想贯穿整个系统。这种全新开发框架模型的最大特性是可实现动态挂接符合 MAPGIS7.0 接口标准的功能模块，使系统具有很大程度的灵活性和可扩展性。通过各子系统的分解，细化，建立关系图，接口层次图，接口方法属性定义，用来指导编码测试。

8.1.1 模块命名规则

（1） 插件接口定义-------前缀 MPI；

（2） 界面接口--------前缀 WPI；

（3） 组件命名--------前缀 CGis；

（4） 界面扩展库类命名--------前缀 GUI；

（5） Plugins--------插件；

其他遵循 ZD－编程规范。

8.1.2 主界面中各对象的功能与操作方式

（1） 菜单条、工具条：执行应用程序 基本地命令响应；

（2） 工作区：数据目录组织、浏览、管理；文档目录结构管理及目录基本操作（添加、删除操作）管理；

（3） 主视图区：地图图像显示、属性浏览、元数据浏览、表格数据浏览等；

（4） 工具箱：功能类似工具条，停靠在框架右边，便于用户操作；

（5） 属性区：提供图元属性浏览，用户自定义数据浏览；

（6） 输出区：提供操作信息的输入，操作结果的显示；

（7） 帮助区：给用户提供及时的帮助信息。

8.2 结构图

应用框架（AppLoader）部分的结构图(图 8.1)如下所示

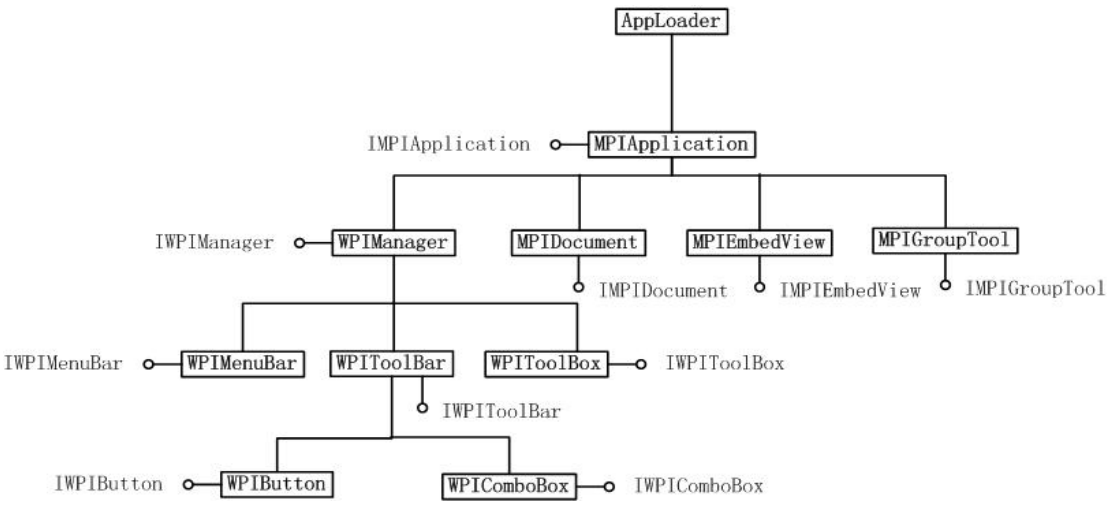


图 8.1　应用框架部分结构图

8.3 主要模块及接口说明

1. MPIFrame 模块

该模块主要定义了框架的接口部分，实现了应用程序的动态加载和管理,主要包括以下几个接口:

IMPIApplication	插件的管理模块，实现插件之间的交互和配置.
IMPIDocument	主要定义了文档的接口.
IMPIEmbedView	有视图类型插件的接口定义.
IMPIResource	获取插件的资源.
IMPIGroupTool	组工具类型插件接口的定义.
IMPITool	工具接口的定义.
IObjectCategory	插件识别，加载接口.
IMPICommand	命令响应接口.

2. AppFrame 模块

从 MPIApplication 接口继承的有关 MapGis7.0 的管理部分,包含接口:IGisAppFrame.

3. WPIFrame 模块

该模块主要实现对应用框架所加载的工具,视图等插件的界面进行定制与管理,如:工具栏的标题的设置等等.

IWPIPropertyPage	属性页
IWPIButton	按钮
IWPIComboButton	组合框
IWPIToolBar	工具条
IWPIManager	界面管理器

4. Definterface

基本和 7.0 相关的数据结构和类型声明.

TagFeatureSet 定义:

```
typedef struct
{
    LPFeatureCls   lpFtureCls;   //要素类指针
    LPFeatureSet   lpFtureSet;   //要素集合指针
        }TagFeatureSet;
```

Envelope 定义:

```
typedef struct
{
        double XMin;        //范围 X 方向最小值
```

```
        double XMax;        //范围 X 方向最大值
        double YMin;        //范围 Y 方向最小值
        double YMax;        //范围 Y 方向最大值
    }Envelope;
```

以下数据结构定义参见开发手册:

Graph_Style 定义:
Enum_SelSetType 定义:
Num_GeomType 定义:
Enum_SpacType 定义:
Enum_DateType 定义:
ViewType 定义:
UNION_GeomType 定义:
GeomSetStru 定义:
SpacSetStru 定义:
Enum_Layer_Travel 定义:
SymbolizeMethod 定义:
AlignType 定义:
GeomType 定义:
UnionDispInfo 定义:
DispInfo 定义:
StruGradeDisplayInfo 定义:

8.4 应用实例

1. 插件的管理

- 功能说明

对 mapgis7.0 框架与插件的开发方式有比较系统化和层次化的认识,学会插件的加载、卸载以及其他相关操作.

图 8.2 插件管理工具

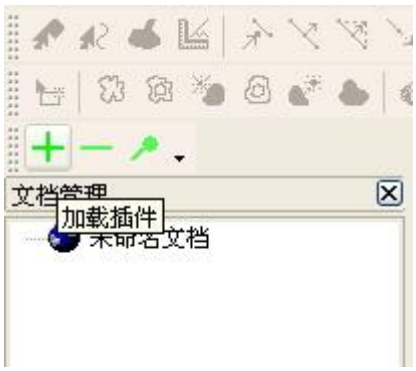


图 8.3 插件管理工具加载后的应用框架

本例中插件管理工具条一共包括三个按钮项,其中分别实现插件的加载、卸载以及浏览.其中加载插件以加载符号库管理插件为例;卸载插件以卸载编辑扩展工具条为例;浏览插件会将用户已加载的所有插件的 CLSID 列出.

- 基本思路

IMPIApplication 接口提供了基本的插件的管理方法,用户可以根据实际需要,通过该接口提供的方法对插进实现交互管理.

用户首先需要创建一个 ATL COM 工程,添加一个 New ATL Object,并根据实际需要来增加接口!步骤如下所示:

a. 新建工程

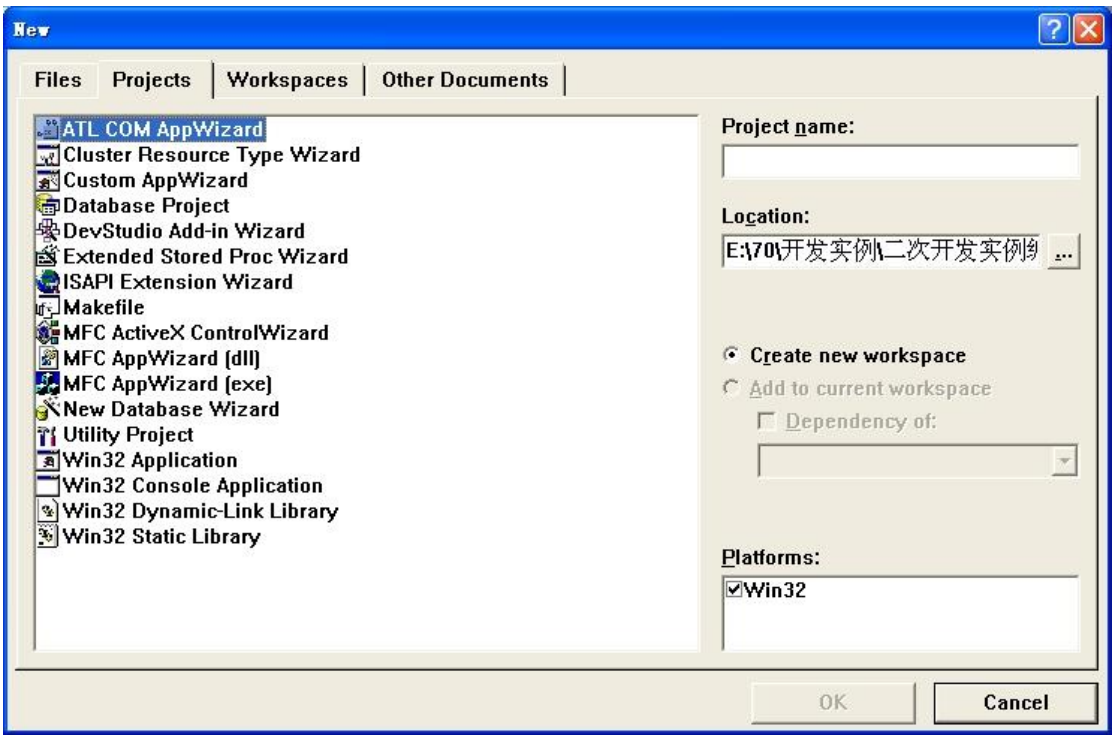


图 8.4 新建一个 ATL COM 应用程序(步骤一)

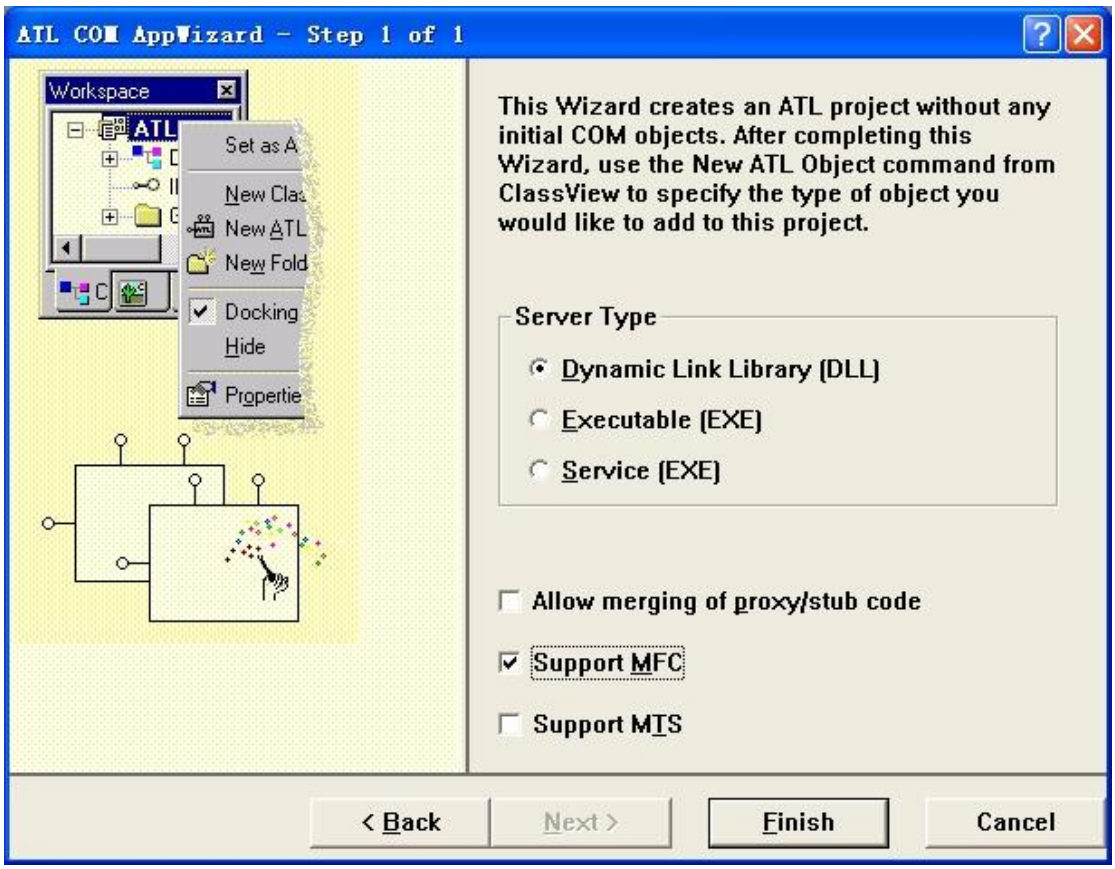


图 8.5 新建一个 ATL COM 应用程序(步骤二)

b. 添加对象

图 8.6 添加对象

图 8.7 添加插件管理接口后的类视图

```
import "oaidl.idl";
import "ocidl.idl";
    [
        object,
        uuid(A3B71FBB-87CF-4F04-AE66-B048E9E6CF21),

        helpstring("IManageObject Interface"),
        pointer_default(unique)
    ]
    interface IManageObject : IUnknown
    {
        [propget, helpstring("property Active")] HRESULT Active([out, retval] BOOL *pVal);
        [propput, helpstring("property Active")] HRESULT Active([in] BOOL newVal);
    };
```

图 8.8 接口*.idl 文件部分截图

c. 添加接口

```
#ifndef __MANAGEOBJECT_H_
#define __MANAGEOBJECT_H_

#include "resource.h"       // main symbols
#include "mpiframe.h"
#include "appframe.h"
#include "cln_bas70.h"
#include "basdefine70.h"
#include "EditExtension.h"

/////////////////////////////////////////////////////////////////////////////
// CManageObject
EXTERN_C const GUID CAITID_MPI_GROUPTOOL;

class ATL_NO_VTABLE CManageObject :
    public CComObjectRootEx<CComSingleThreadModel>,
    public CComCoClass<CManageObject, &CLSID_ManageObject>,
    public IManageObject,
    public IMPIGroupTool,
    public IMPICommand
```

图 8.9 类的说明

```
    DECLARE_REGISTRY_RESOURCEID(IDR_MANAGEOBJECT)

    DECLARE_PROTECT_FINAL_CONSTRUCT()

    BEGIN_CATEGORY_MAP(CManageObject)
    IMPLEMENTED_CATEGORY(CAITID_MPI_GROUPTOOL)
    END_CATEGORY_MAP()

    BEGIN_COM_MAP(CManageObject)
        COM_INTERFACE_ENTRY(IManageObject)
        COM_INTERFACE_ENTRY(IMPIGroupTool)
        COM_INTERFACE_ENTRY(IMPICommand)
        COM_INTERFACE_ENTRY(IObjectCategory)
    END_COM_MAP()
```

图 8.10 增加接口

需要增加 IMPIGroupTool 组工具管理接口、IMPICommand 命令响应接口,

IObjectCategory 插件的识别、加载接口.

用户还需要实现上述接口下的所有虚函数,对应的接口函数可查看相关的*.h 文 件,本例中需要实现的接口函数如图 8.11:

```
// IMPICommand
public:
    STDMETHOD(OnCommand)(UINT cmmID,LONG idExten,LONG lParam,BSTR text);
    STDMETHOD(get_Resource)(UINT iIndex,IMPIResource* *pVal);

//IObjectCategory
public:
    STDMETHOD(GetObjectCategory)(IID* idCategoryBase,CLSID* clsidCategoryImpl);
    STDMETHOD(OnConnect)();
    STDMETHOD(OnDisconnect)();
    STDMETHOD(OnCreate)(IMPIApplication* pMpiApplication);
//IMPIGroupTool
public:
    STDMETHOD(OnWinMessage)(WIN_MESSAGE_PTR ptrMessage,short *sFinshed);
    STDMETHOD(get_State)(Enum_ToolStates *pVal);
    STDMETHOD(put_State)(Enum_ToolStates newVal);
    STDMETHOD(Active)(BOOL bActive);
//IManageObject
public:
    STDMETHOD(get_Active)(/*[out, retval]*/ BOOL *pVal);
    STDMETHOD(put_Active)(/*[in]*/ BOOL newVal);
```

图 8.11 插件接口函数

(1) 部分代码

```
IMPIApplication *m_pMpiApplication;     //IMPIApplication 指针通过 OnCreat 接口
                                         //函数获取得到
//加载插件
if (m_pMpiApplication)
{
    IObjectCategory* pObject=NULL;//定义插件管理接口
    // SymEditTools.SymEditToolBar.1 为符号颜色库管理插件对应的注册表项
    // CLSID
    //取插件的 CLSID
    HRESULT hr= CLSIDFromString(A2W("SymEditTools.SymEditToolBar.1"),
                    &m_objectclsid);
    if (SUCCEEDED(hr))
    {
        //加载插件
        m_pMpiApplication->LoadObject(m_objectclsid,0,&pObject);
        if(pObject)
        {
            //释放接口
            pObject->Release();
            pObject=NULL;
        }
    }
}

//卸载插件
if (m_pMpiApplication)
```

```
{
    IObjectCategory* pObject=NULL;
    HRESULT hr;
    //查询 pEnum 接口,pEnum 为已经加载的插件接口
    hr =m_pMpiApplication->QueryInterface(IID_IMPIAddinEnum,
                                          (void**)&pEnum);
    if(pEnum)
    {
        pEnum->Reset(IID_IEditExtension);//起始位置为编辑扩展插件
        pEnum->Next(&pObject);          //取编辑扩展插件对象
        if (!pObject)
        {
            MessageBox(NULL,"未加载编辑扩展插件!!","提示",MB_OK);
            return ;
        }
        //卸载插件
        m_pMpiApplication->UnLoadObject(m_objectcate);
        //释放接口
        pObject ->Release();
        pObject =NULL;
        pEnum->Release();
    }
}

//循环取插件对象
if (m_pMpiApplication)
{
    LPWSTR m_progID;
    CString   m_string;
    IObjectCategory* pObject=0;
    IID                 idCategoryBase;        //返回插件的类型
    CLSID               clsidCategoryImpl;     //实现对象的 CLSID
    //查询 IMPIAddinEnum 接口
    m_pMpiApplication->QueryInterface(IID_IMPIAddinEnum,
                                      (void**)&pEnum);
    if(pEnum)
    {
        pEnum->Reset(IID_NULL);//从所有已加载的第一个插件开始
        pEnum->Next(&pObject); //取第一个 IobjectCategory 对象
        //循环取所有已经加载的插件对象
        while(pObject)
        {
            //取插件管理接口
            pObject->GetObjectCategory(&idCategoryBase,
```

```
                                                    &clsidCategoryImpl);
                    ProgIDFromCLSID(clsidCategoryImpl,&m_progID);
                    m_string=m_string + W2A(m_progID)+"\n";
                    pObject->Release();
                    //取下一个插件对象
                    pEnum->Next(&pObject);
                }
                //释放接口
                pObject->Release();pObject=NULL;
                MessageBox(NULL,m_string,"已加载插件列表",MB_OK);
                pEnum->Release();
            }
        }
```

2. 用户工具插件的编写

- **u** 功能说明

mapgis7.0 框架+插件的开发模式使得用户自定义插件成为一种极其重要的二次开发形式,用户可以根据需要集成现有功能或者新增其他功能,通过实现 mapgis7.0 标准插件接口,将这些功能模块以插件的形式添加到应用框架中,对数据进行交互操作,具有极大的灵活性和可扩展性.

本例在建立的 DemoGroupTool 工程中添加两个 simple object,其中 IdemoToolbar 用来实现整个功能模块插件的管理与控制,IGetAtt 实现要素类的 OID 的查询.

本例用户自定义插件(ToolBar)主要包括以下几个功能按钮:

a. 帮助按钮.
b. 打印按钮.
c. 关于按钮.
d. 浏览编辑当前激活图层信息按钮.
e. 通过鼠标拾取要素,浏览要素的属性及属性结构.

- **u** 基本思路

用户自定义插件只要支持应用框架定义的接口，其就可以在应用框架中很好的运行,用户一般需要实现以下几个接口函数:

a. //单一功能插件,主要如 Toolbar,Menu 等,实现按钮的响应事件.
::OnCommand(UINT cmmID,LONG idExten,LONG lParam,BSTR text)
//视图类插件,创建视图,定义视图的位置及其他属性
::OnCreateView(WIN_HANDLE container,WIN_HANDLE* lEmbedViewWnd)

b. ::get_Resource(UINT iIndex,IMPIResource* *pVal)
该函数主要用来获取用户定义插件的资源.

c. //功能插件
::OnWinMessage(WIN_MESSAGE_PTR ptrMessage,short * sFinshed)
该函数用来接收消息
//视图插件
::OnMessage(LONG message,LONG lParam,LONG*lOption,IObjectCategory* pSender)
该函数用来定义视图插件的消息机制.
其他步骤参照实例 1 来完成

- **u** 部分代码(以功能模块插件为例)

```
//DemoToolbar.h 中定义成员变量
IMPIApplication *m_pMpiApplication;//应用框架
IMPITool* m_ActiveTool;
IGisDocument* pGisDoc = NULL;//定义地图文档接口
IMap* pMap = NULL ;//定义 map 接口
IMapLayer     *pMapLayer=NULL;//定义图层接口
IFeatureLayer *pFeatureLayer=NULL;//要素类图层接口
CFeatureCls          *m_Fcls=NULL;//要素类
IMPIDocument* m_pMpiDocument;//地图文档

// DemoToolbar.cpp 的部分代码
//获取资源
STDMETHODIMP CDemoToolbar::get_Resource(UINT iIndex,IMPIResource* *pVal)
{
    USES_CONVERSION;
    if(iIndex != 0)
        return S_FALSE;
    //定义资源接口
    IMPIResource* pResource =NULL;
    //创建 IMPIResource
    ::CoCreateInstance(CLSID_MPIResource,NULL,
        CLSCTX_INPROC_SERVER,
        IID_IMPIResource,
        (void**)&pResource);
    if(pResource)
    {
        //定制资源信息
        pResource->put_ResourceHandle((LONG)_Module.m_hInstResource);
        pResource->put_Caption(A2W("示例工具条"));
        // IDR_TOOLBAR_IMAGES 为已定义好的 ToolBar 资源
        pResource->put_ToolbarResID((LONG)IDR_TOOLBAR_IMAGES);
        //资源类型为 TOOLBAR
        pResource->put_ResourceMask(ENUM_RESOURCE_TOOLBAR);
        *pVal = pResource;
        return S_OK;
    }
    else
    {
        *pVal = NULL;
        return S_FALSE;
    }
    return S_OK;
}
```

```
//按钮响应事件
STDMETHODIMP CDemoToolbar::OnCommand(UINT cmmID,
                                     LONG idExten,LONG lParam,BSTR text)
{
    USES_CONVERSION;
    char clsstr[50];
    CLSID tool_clsid;
    HRESULT hr;
    BOOL    bHitZoom = FALSE;
    m_cmdID  = cmmID;
    switch(cmmID)
    {
    case ID_APP_ABOUT:
        //关于功能略
        break;
    case ID_FILE_PRINT:
        //打印功能略
        break;
    case ID_HELP_INDEX:
        //帮助略
        break;
    case ID_GET_LAYERINFO:    //获取激活图层的信息,添加 CFclsInfo 对话框类
        //取地图文档
        if (m_pMpiDocument)
        {
            CFclsInfo    dlg;
            //查询 IGisDocument 接口
            m_pMpiDocument->QueryInterface(IID_IGisDocument,
            (void**)&pGisDoc);
            if (pGisDoc)
            {
                //取当前激活地图
                pGisDoc->get_CurMap(&pMap);
                if (pMap)
                {
                    //取当前激活图层
                    pMap->get_ActiveLayer(&pMapLayer);
                    if (pMapLayer)
                    {
                        //取激活图层对应的要素类
                        pFeatureLayer=(IFeatureLayer*)pMapLayer;
                        pFeatureLayer->get_FeatureClass(&pFeatureVal);
                        if (pFeatureVal)
```

```
                    {
                        m_Fcls=new CFeatureCls;
                        m_Fcls=(CFeatureCls*)pFeatureVal;
                        if (m_Fcls)
                        {
                            //取要素类信息
                            m_Fcls->fcls_GetInfo(m_FclsInfo);
                            dlg.m_name=m_FclsInfo.name;
                            dlg.m_owner=m_FclsInfo.owner;
                            dlg.m_creatTime.Format("%d\-%d\-%d",
                                    m_FclsInfo.createTime.year,
                                    m_FclsInfo.createTime.month,
                                    m_FclsInfo.createTime.day);
                            dlg.m_srID=m_FclsInfo.srID;
                            dlg.m_dsID=m_FclsInfo.dsID;
                            dlg.m_aliasName=m_FclsInfo.aliasName;
                            dlg.DoModal();
                        }
                    }
                    //释放接口
                    pMapLayer->Release();
                    pMapLayer=NULL;
                }
                pMap->Release();
                pMap=NULL;
            }
            pGisDoc->Release();
            pGisDoc=NULL;
        }
    }
    break;
case ID_BUTTONTEST:
    //略
    break;
case ID_MOUSE_TEST:
    //略
    break;
case ID_GET_ATT: //取要素类 OID
    //注意: DemoGroupTool.GetAtt.1 为定义的获取要素类 OID 接口 CLSID
    //包含 IMPITool,及 IGetAtt 两个接口
    strcpy(clsstr, " DemoGroupTool.GetAtt.1 ");
    bHitZoom=true;
    break;
default:
```

```
            break;
        }
        if(bHitZoom)
        {
            if(m_ActiveTool)
            {
                m_ActiveTool->Release();
                m_ActiveTool = NULL;
            }
            //取 Tool 的 CLSID
            CLSIDFromString(A2W(clsstr),&tool_clsid);
            //创建 IMPITool
            hr = CoCreateInstance(tool_clsid,
                NULL,
                CLSCTX_SERVER,
                IID_IMPITool,
                (void**) &m_ActiveTool );

            if(!m_ActiveTool)
                return S_FALSE;
            //调用 OnCreat 函数
            m_ActiveTool->OnCreate(m_pMpiApplication);
        }
        return S_OK;
}

// 实现 OnCreate 接口函数
STDMETHODIMP CDemoToolbar::OnCreate(IMPIApplication* pMpiApplication)
{
        if(pMpiApplication)
        {
            //取到应用框架及对应的地图文档
            m_pMpiApplication = pMpiApplication;
            pMpiApplication->get_Document(&m_pMpiDocument);
            if(m_pMpiDocument)
                m_pMpiDocument->Release(); //不计数
        }
        return S_OK;
}

// 实现 OnWinMessage 接口函数
STDMETHODIMP CDemoToolbar::OnWinMessage(WIN_MESSAGE_PTR
                                                ptrMessage,short * sFinshed)
{
```

```
if(!ptrMessage)
    return S_FALSE;
LONG xPos =0;
LONG yPos=0 ;
MSG* pMsg = (MSG*)ptrMessage;
RECT    rc;
//取当前窗口范围
::GetClientRect(pMsg->hwnd,&rc);
    switch (pMsg->message)
    {
    case WM_KEYDOWN:
        break;
    case WM_LBUTTONDOWN:
        //传入的鼠标位置
        xPos = LOWORD(pMsg->lParam);
        yPos = HIWORD(pMsg->lParam);
        if(m_ActiveTool)
        {
            //这里传入的坐标为设备坐标
            m_ActiveTool->OnLMouseDown(0,0,xPos,rc.bottom-yPos);
        }
        break;
    case WM_RBUTTONDOWN:
        break;
    case WM_RBUTTONUP:
        if (m_ActiveTool)
        {
            *sFinshed = 1;
        }
        break;
    case WM_MOUSEMOVE:
        xPos = LOWORD(pMsg->lParam);
        yPos = HIWORD(pMsg->lParam);
        if(m_ActiveTool)
            //这里传入的坐标为设备坐标
            m_ActiveTool->OnMouseMove(0,0,xPos,rc.bottom-yPos);
        break;
    case WM_LBUTTONUP:
        xPos = LOWORD(pMsg->lParam);
        yPos = HIWORD(pMsg->lParam);
        if(m_ActiveTool)
        {
            //这里传入的坐标为设备坐标
            m_ActiveTool->OnLMouseUp(0,0,xPos,rc.bottom-yPos);
```

```
                }
                break;
            }
        return S_OK;
}

//GetAtt.cpp 部分代码
#include "stdafx.h"
#include "DemoGroupTool.h"
#include "GetAtt.h"
#include "DemoToolbar.h"

//实现 OnCreate
STDMETHODIMP CGetAtt::OnCreate(IMPIApplication* pMpiApplication)
{
    m_pGisDoc = NULL ;
    IMPIDocument      *pMpiDocument;
    pMpiApplication->get_Document(&pMpiDocument);   //取地图文档
    //取应用框架接口
    pMpiApplication->QueryInterface(IID_IGisAppFrame,(void**)&pApp);
    if(pMpiDocument)
    {
        //查询 IGisDocument 接口
        pMpiDocument->QueryInterface(IID_IGisDocument,(void**)&m_pGisDoc);
    }
    return E_NOTIMPL;
}

// 响应鼠标事件查询要素类
STDMETHODIMP CTeatViewAtt::OnLMouseUp( UINT Button, UINT Shift,
                                                    int X, int Y)
{
    IMap      *pMap;
    long pVal;
    IMapLayer     *pLayer;
    D_DOT         pos;
    CFeatureSet   *pSet;

    if    (GetTransformation(&pTans)==S_OK);   //调用 GetTransformation 函数取
                                               //转换接口
    {
        if (pTans)
        {
            pTans->DpToLp(X,Y,&(pos.x),&(pos.y));//设备坐标转换为逻辑坐标
```

```
            pTans->Release();pTans=NULL;//释放接口
        }
    }
    if (m_pGisDoc)
    {
        //获取当前激活地图
        m_pGisDoc->get_CurMap(&pMap);
        if (pMap)
        {
            //获取当前激活图层
            pMap->get_ActiveLayer(&pLayer);
            m_fclslayer=(IFeatureLayer*)pLayer;
            if (m_fclslayer)
            {
                //取要素图层对应的要素类
                m_fclslayer->get_FeatureClass(&pVal);
                m_fcls=new CFeatureCls;
                m_fcls=(CFeatureCls*)pVal;
                char name[128]="";

                //创建结果集
                pSet=new CFeatureSet;
                //矩形查询
                m_fcls->fcls_GetName(name,128);
                long ri=m_fcls->f_Near(&pos,pSet,20,20);
                if (ri)
                {
                    TYPE_OBJ_ID      id;
                    CATT_STRU stru;
                    char resultType=-1;
                    CRecord         rcd;
                    long num=pSet->GetObjCount();
                    if (num>0)
                    {
                        //取结果集的第一个位置
                        pSet->MoveFirst(-1);
                        //取要素类的 OID
                        pSet->GetObjID(&id);
                        CString    str;
                        str.Format("您选择的要素类为%d",id);
                        MessageBox(NULL,str,"结果",MB_OK);
                    }
                    delete pSet;pSet=NULL;
                }
```

```
                    m_fclslayer->Release();m_fclslayer=NULL;
                }
                pMap->Release();pMap=NULL;
            }
        }
        return E_NOTIMPL;
    }

    //实现 GetTransformation 函数,主要用来取转换接口
    HRESULT CTeatViewAtt::GetTransformation(ITransformation** pTans)
    {
        if(pApp)
        {
            //取转换接口
            pApp->get_TransformationObj(VM_DATEVIEW,pTans);
            pApp->Release();
            pApp=NULL;
            return S_OK;
        }
        return S_FALSE;
    }
```

3. 用户视图插件的编写

- **u** 功能说明

mapgis70 二次开发用户可根据实际需要增加视图类插件,IMPIEmbedView 接口主要 用 来 管 理视图类插件的创建、加载以及消息映射等，用户通过该接口提供的接口函数可 实现视图的定制.

本例中创建了一个 TreeCtrl 视图,打开地图文档,以及添加地图、删除地图、添加 图层、删除图层操作时视图显示相应的树型结构.

- **u** 基本思路

建立工程及添加接口的方法参见实例 1.

```
DECLARE_REGISTRY_RESOURCEID(IDR_TESTEMBEDVIEW)

DECLARE_PROTECT_FINAL_CONSTRUCT()

BEGIN_CATEGORY_MAP(CTestEmbedView)
IMPLEMENTED_CATEGORY(CAITID_MPI_EMBEDVIEW)
END_CATEGORY_MAP()

BEGIN_COM_MAP(CTestEmbedView)
    COM_INTERFACE_ENTRY(ITestEmbedView)
    COM_INTERFACE_ENTRY(IMPIEmbedView)
    COM_INTERFACE_ENTRY(IObjectCategory)
    COM_INTERFACE_ENTRY(IMessagePort)
END_COM_MAP()
```

图 1.12 视图类插件接口

其中 IMPIEmbedView 主要用来创建视图,管理视图的加载等操作,IMessagePort 实 现 插 件 的消息映射,IObjectCagory 接口实现插件的加载、卸载等.

用户需要实现的接口函数可参照实例一,查看相应的*.h 文件来完成!

- 部分代码

```
//获取资源
STDMETHODIMP CTestEmbedView::get_Resource(UINT iIndex,
                                          IMPIResource** pResource)
{
    USES_CONVERSION;
    HRESULT hr ;
    //创建资源接口
    hr=CoCreateInstance(CLSID_MPIResource,
                        0,
                        CLSCTX_INPROC_SERVER,
                        IID_IMPIResource,
                        (void**)pResource);
    if(FAILED(hr))
        *pResource = 0;
    else
    {
        //定制资源信息
        (*pResource)->put_Caption(A2W("测试视图"));
        (*pResource)->put_ResourceHandle((LONG)_Module.m_hInstResource);
        (*pResource)->put_ImageResID((LONG)IDB_ICON);//IDI_PROPERTIES_BAR
    }
    return S_OK;
}

//创建视图 m_wndPropList 为 CtreeCtrl 类型变量
STDMETHODIMP CTestEmbedView::OnCreateView(WIN_HANDLE container,WIN_HANDLE*
                                          lEmbedViewWnd)
{
    HWND hWnd = (HWND )container;
    CWnd* pParentWnd =(CWnd*)CWnd::FromHandle(hWnd);
    m_parentWnd = pParentWnd;
    //创建视图
    m_wndPropList.Create(WS_CHILD | WS_VISIBLE , CRect(0,0,0,0), pParentWnd,
                (UINT)-1);//| WS_TABSTOP | WS_BORDER
    if(m_wndPropList.GetSafeHwnd())
    {
        *lEmbedViewWnd = (WIN_HANDLE)m_wndPropList.GetSafeHwnd();
    }
    else
        *lEmbedViewWnd = NULL;
    return S_OK;
```

```
}

//响应消息函数
STDMETHODIMP CTestEmbedView::OnMessage(LONG message,
                        LONG lParam,LONG*wParam,IObjectCategory* pSender)
{
    USES_CONVERSION;
    short      sflg;
    BSTR     m_mapname;     //地图名称
    BSTR     m_layername; //图层名
    BSTR      m_docname;     //地图文档名
    HTREEITEM hTDOC=NULL, hTMAP=NULL, hTLAYER=NULL;   //定义 HTREEITEM 变量

    //创建 IGroupLayer 接口
    CoCreateInstance(CLSID_GroupLayer,0,CLSCTX_INPROC,IID_IMapLayer,
                                                    (void**)&pGroupLay);
    switch(message) {
    //打开地图文档调用
    case MSG_GIS_OPENDOCUMENT:
        if (pMpiDoc)
        {
            //取地图文档
            pMpiDoc->QueryInterface(IID_IGisDocument,(void**)&pGisDoc);
            if (pGisDoc)
            {
                pGisDoc->get_FileName(&m_docname);   //取文档名称
                //添加根节点
hTDOC=m_wndPropList.InsertItem(W2A(m_docname),
                                            NULL,NULL,TVI_ROOT,TVI_LAST);
                //设置节点数据项
                m_wndPropList.SetItemData(hTDOC,(DWORD)pGisDoc);
                //遍历地图
                HRESULT hr= pGisDoc->EnumMap(FIRST,&pMap,&sflg);
                while (hr==S_OK && sflg>0)
                {
                    pMap->get_BasicLayer(&pTempLayer);
                    if (pTempLayer)
                    {
                        //取地图对应的组图层
        pTempLayer->QueryInterface(IID_IMapLayer,
                                                    (void**)&pGroupLay);
                        //取地图名称
                        pMap->get_Name(&m_mapname);
                        //添加子节点
```

```
        hTMAP=m_wndPropList.InsertItem(W2A(m_mapname),
                                        NULL,NULL,hTDOC,TVI_LAST);
                        //设置节点数据项
                        m_wndPropList.SetItemData(hTMAP,(DWORD)pGroupLay);
                        //遍历图层
                        pMap->EnumLayer(FIRST,&pMapLayer);
                        while (pMapLayer)
                        {
                            //取图层名称
                            pMapLayer->get_LayerName(&m_layername);
                            //添加子节点
                        hTLAYER=m_wndPropList.InsertItem(W2A(m_layername),
                                    NULL,NULL,hTMAP,TVI_LAST);
                            //设置节点数据项
            m_wndPropList.SetItemData(hTLAYER,
                                        (DWORD)pMapLayer);
                            pMap->EnumLayer(NEXT,&pMapLayer);
                        }
                    }
                    HRESULT hr=pGisDoc->EnumMap(NEXT,&pMap,&sflg);
                }
                //设置数型控件的样式
                DWORD dwStyle=GetWindowLong(m_wndPropList.GetSafeHwnd(),
                                        GWL_STYLE);
                dwStyle |=TVS_HASLINES+TVS_HASBUTTONS+TVS_LINESATROOT;
                SetWindowLong(m_wndPropList.GetSafeHwnd(),
                                                GWL_STYLE,dwStyle);
                //更新控件
                m_wndPropList.SetRedraw(true);
            }
        }
        break;
    //关闭地图文档时调用
    case MSG_GIS_CLOSEDOCUMENT:
    //删除所有节点
        m_wndPropList.DeleteAllItems();
        break;
    //添加地图时调用
    case MSG_ADDMAP:
        pMap=(IMap*)lParam;
        if (pMap)
        {
            pMap->get_BasicLayer(&pTempLayer);
            if (pTempLayer)
```

```
            {
                //取地图的组图层
                pTempLayer->QueryInterface(IID_IMapLayer,
                                                (void**)&pGroupLay);
                HTREEITEM hTADDMAP;   //定义添加地图增加的节点
                pMap->get_Name(&m_mapname);
                if (m_wndPropList.GetRootItem())
                {
                    //增加节点
                    hTADDMAP=m_wndPropList.InsertItem(W2A(m_mapname),NULL,
                                NULL,m_wndPropList.GetRootItem(),TVI_LAST);
                    //设置节点数据项
                    m_wndPropList.SetItemData(hTADDMAP,(DWORD)pGroupLay);
                    //遍历图层
                    pMap->EnumLayer(FIRST,&pMapLayer);
                    while (pMapLayer)
                    {
                        //取图层名称
                        pMapLayer->get_LayerName(&m_layername);
                        //增加子节点
                        hTLAYER= m_wndPropList.InsertItem(W2A(m_layername),
                                                NULL,NULL,hTADDMAP,TVI_LAST);
                        //设置节点数据项
                        m_wndPropList.SetItemData(hTLAYER,
                                                        (DWORD)pMapLayer);
                        pMap->EnumLayer(NEXT,&pMapLayer);
                    }
                }
            }
            m_wndPropList.SetRedraw(true);
        }
        break;
    //删除地图时调用
    case MSG_DELMAP:
        pMap=(IMap*)lParam;
        if (pMap)
        {
            DWORD pItem;
            HTREEITEM hTDELMAP=NULL;     //定义删除图层对应的节点
            pMap->get_BasicLayer(&pTempLayer);
            if (pTempLayer)
            {
                //取地图的组图层
                pTempLayer->QueryInterface(IID_IMapLayer,
```

```
                                                    (void**)&pGroupLay);
                    //循环取每个节点
                    hTDELMAP=m_wndPropList.GetChildItem(
                                                    m_wndPropList.GetRootItem());
                    while (hTDELMAP)
                    {
                        //取节点数据
                        pItem=m_wndPropList.GetItemData(hTDELMAP);
                        //判断该节点数据与要删除的地图组图层是否一致
                        if (pGroupLay==(IGroupLayer*)pItem)
                        {
                            //删除节点
                            m_wndPropList.DeleteItem(hTDELMAP);
                            break;
                        }
                        //取下一个节点
                        hTDELMAP=m_wndPropList.GetNextItem(hTDELMAP,TVGN_NEXT);
                    }
                }
                m_wndPropList.SetRedraw(true);
            }
            break;
        //添加图层时调用,参见实例,这里不列出
        case MSG_ADDLAYER:
            break;
        //删除图层时调用,参见实例,这里不列出
        case MSG_DELLAYER:
            break;
        }
        return S_OK;
    }
```

第 9 章 地图文档

9.1 概述

9.1.1 定义、缩写

（1）**地图文档** GisDocument。地图文档是地图的一种数据的综合表现和管理形式，存储了组成地图的各种制图元素，包括标题、指北针、图例、比例尺、布局、数据窗体、图层等，但图层只是作为地理数据的一种引用，指向位于本地或者网络数据库中的地理数据集，并不存储地理数据；

（2）**排版、布局** Layout。为用户最后出版地图的版面控制提供功能和方法；

（3）**排版框** LayoutFrame。排版框用以把出版时需要的各种制图元素和版面有机的结合起来，一个排版框绑定一个元素；

（4）**地图** Map。用户感兴趣的一些对地理数据引用构成的图层的集合。地图的主要作用是集中的管理这些独立的图层，为用户归纳，综合分析地理数据等提供手段；

（5）**图层** Layer。图层是对一类具有相似特性的地理数据的引用；

（6）**组图层** GroupLayer。把用户感兴趣的一组特定图层集合在一起，集中管理；

（7）**要素层** FeatureLayer。用以对地理数据库中要素类的引用形成的图层的管理；

（8）**符号化（规则化）** Symbolization。对要素层中的要素指定规则，按特定规则显示和分析提供支持；

（9）**地图环境** MapSurround。每一个 Map 都有一个地图环境，描述与 Map 相关的一些环境信息，如图例表示、比例尺表示、指北针表示等；

（10）**制图元素** CartoElement。构成地图环境的一些元素，包括图例元素、比例尺元素、指北针元素等。

9.2 结构图

地图文档部分分文档管理和符号化显示两个模块。文档管理模块的结构图如图 9-1 所示：

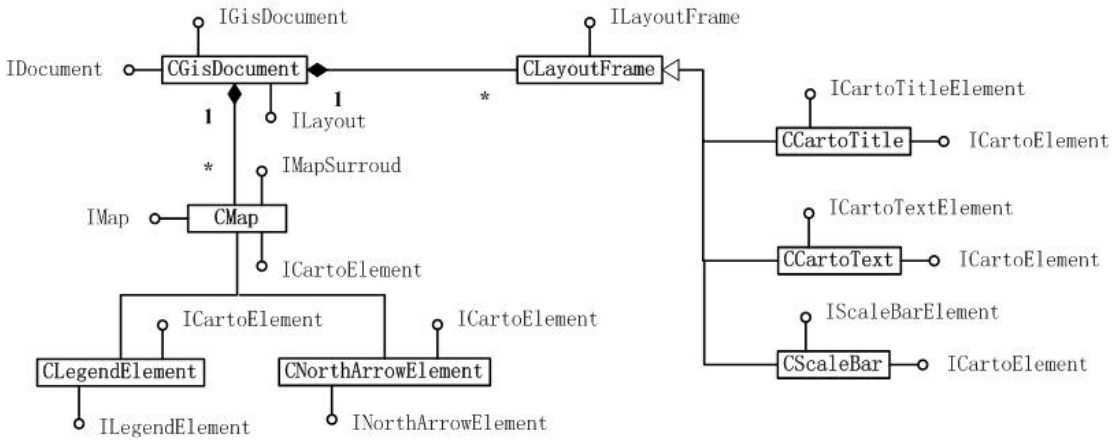


图 9-1 文档管理结构图

符号化显示模块的结构图如图 9-2 所示：

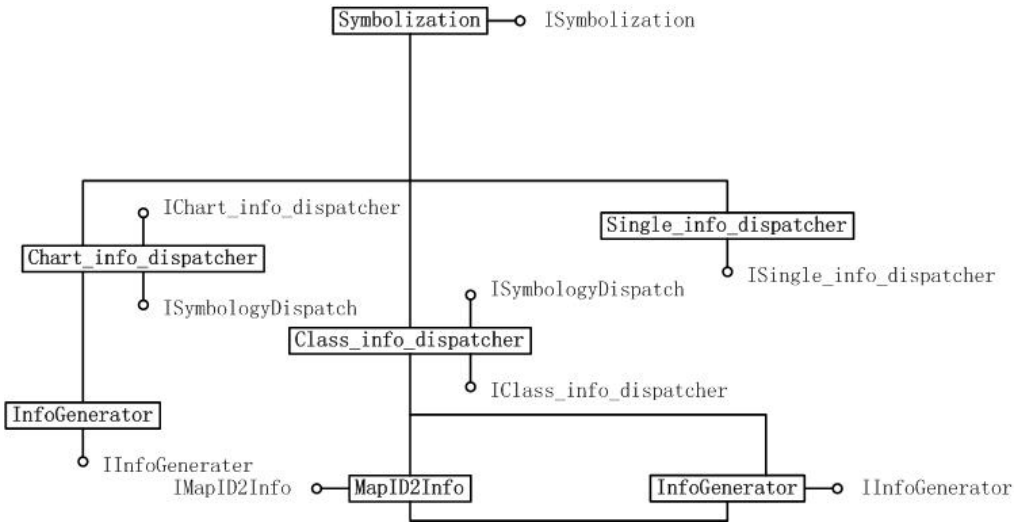


图 9-2 符号化显示结构图

9.3 主要模块及接口说明

1. GisMap 模块 IMap 实现地图数据的控制和保存
   - IMapSelection 地图数据选择接口
   - ILayerSelection 图层选择接口
   - IMapSelectTool 地图选择工具接口
   - IMapLayer 所有图层的基接口
   - ILayerSelectTool 图层选择工具接口
   - IFeatureLayer 要素图层实现一个要素类的数据管理和基本操作
   - IMapSurround 地图制图元素，包括指北针、专题图、图例、比例尺等
   - IGroupLayer 组图层管理所属的多个图层
   - IAnnotationLayer 注记图层实现注记类的管理和基本操作
   - ILayerLogin 图层登陆和退出接口的定义
   - ISFeatureLayer 简单要素图层实现简单要素类的管理和基本操作
   - IGeomNetLayer 几何网络图层实现几何网络类的管理和基本操作
2. CartoElement 模块
   - IMICartoElement 制图元素包括指北针、专题图、比例尺等
   - IMapSeIScaleBarElement 比例尺
   - INorthArrowElement 指北针
   - IGraphElement 专题图
3. GISDocument 模块
   - IGisDocument 实现地图文档数据的加载和保存
   - IGisTemplate 地图文档模版的应用和管理
4. 符号化显示模块

   ISymbolization 符号化实现的接口 ISymbologyDispatch 符号化分配接口

IMapID2Info	类图形信息接口
IClass_info_dispatcher	统计信息接口
ISingle_info_dispatcher	单符号化接口
IInfoGenerater	图表信息生成接口
IChart_info_dispatcher	图表分类信息接口

9.4 MAPGIS70 的 VC++开发环境介绍(MFC AppWizard exe 工程)

前面一章我们介绍了有关 mapgis70 进行插件开发的一些环境设置(ATL COM 工程),本节将对 mapgis70 二次开发 MFC 应用程序工程环境设置作进一步的说明,在以后的章节中将不在说明:

(1) 建立一个 Visual C++ MFC 应用程序工程.

(2) 继承 CGxViewEx 视图类及 CMapGisView 视图类.

CGxViewEx 视图类主要用来进行 mapgis70 的视图管理,包括如刻度尺,滚动条等,CMapGisView 用来显示数据及其他窗口操作等.

具体操作如下:

a. 打开工程中的视图类的两个文件(*View.h,*View.cpp),用 Visual C++查找替换 工 具 (Edit 功能菜单下的 Replace 命令或热键 Ctrl+H),分别将两个文件中的 CView 字符创全 部替换为 CGxViewEx,替换完毕,在工程视图类的定义前面加入#include "GxViewex.h", 这 样就完成 CGxViewEx 视图类的替换了.

b. 调用 CGxViewEx 的 OnDraw(pDC)函数,以完成绘图工作的预先处理,具体操作方 法:在视图类的 OnDraw(CDC*pDC)函数中加入语句 CGxViewEx::OnDraw(pDC)

```
void CTestView::OnDraw(CDC* pDC)
{
    CTest_MapViewDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // TODO: add draw code for native data here
    CGxViewEx::OnDraw(pDC);
}
```

c. 用户新建一个视图类,如 CMyView(从 CView 中继承),参照步骤 a 将 CView 字符创,替换为 CMapGisView,替换完毕,在工程视图类的定义前面加入#include "MapGisView.h", 这样就完成 CMapGisView 视图类的替换了.

d. 参考步骤 b 在 c 中新建的视图类的 OnDraw(CDC*pDC)函数中加入语句 CMapGisView::OnDraw(pDC):

```
void CMyView::OnDraw(CDC* pDC)
{
    CDocument* pDoc = GetDocument();
    // TODO: add draw code here
    CMapGisView::OnDraw(pDC);
  }
```

e. 在工程视图 CTestView 创建的同时,创建 CMyView

```
int CTestView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    SetMapViewRuntime(RUNTIME_CLASS(CMyView));
```

```
    if (CGxViewEx::OnCreate(lpCreateStruct) == -1)
        return -1;
    return 0;
}
```

注意:用户也可直接通过 CGxViewEx::GetMapGisView()得到一个 CMapGisView 指 针 ! 在 此 , 并不作硬性规定必须建立一个从 CMapGisView 继承的新的视图类!

9.5 应用实例

一. 地图文档的基本操作

(1) 功能说明

实现地图文档的加载、保存、获取地图文档的基本信息以及地图的创建、删除、激活等功能.

(2) 基本思路

通过 IGisAppFrame 接口获取地图文档接口 IgisDocument,该接口下提供了地图文档基本操作的接口函数.

(3) 部分代码

```
//地图文档的打开
void CMyView::OnOpenMapDocument()
{
    IGisAppFrame       *pApp; //应用框架接口
    IGisDocument*pGisDoc;   //地图文档接口
    short sflg;
    //获取应用框架接口
    GetGisAppFrame(&pApp);
    if(pApp)
    {
        //获取地图文档接口
        pApp->get_GisDocument(&pGisDoc);
        if(pGisDoc)
        {
          //第一个参数给 0,弹出对话框
            pGisDoc->Open(0,OPEN_NORMAL,&sflg);
            if (sflg)
                MessageBox("打开地图文档成功!!!");
            else
            {
                //释放接口
                pGisDoc->Release();
                pGisDoc=NULL;
            }
        }
    }
```

```
}

//地图文档的保存
void CMyView::OnSaveMapDocunment()
{
	if (pGisDoc)
	{
		pGisDoc->Save();		//保存
	}
	else
	{
		MessageBox("未打开地图文档!!!");
	}
}

//遍历地图文档中的地图
void CMyView::OnViewAllMap()
{
 USES_CONVERSION;
 IMap	*pTempMap=0;
 short	sflg;
 short	pLayerIndex=0;
 if (pGisDoc)
 {
	ULONG pMapCount;	//地图文档中地图的数目
	pGisDoc->get_MapCount(&pMapCount); //取地图数
	if (!pMapCount)
	{
		MessageBox("该文档中没有地图!!!");
		return;
	}
	//从地图文档中的第一个 map 开始遍历
	pGisDoc->EnumMap(FIRST,&pTempMap,&sflg);
	while(sflg>0)
	{
		LPWSTR  pVal;
		Long	pLayercount;
		pTempMap->get_Name(&pVal);//获取地图名
		pTempMap->get_LayerCount(&pLayercount);//取地图中图层的数目
		CString Info;
		Info.Format("第%d 号地图信息:\n 地图名为:%s\n 地图中包含的图层数
				为:%d\n",pLayerIndex,W2CA(pVal),pLayercount);
		MessageBox(Info);
		pTempMap->Release();
```

```
            pTempMap=NULL;
            //取下一个 map
            pGisDoc->EnumMap(NEXT,&pTempMap,&sflg);
            pLayerIndex++;
        }
    }
    else
    {
        MessageBox("未打开文档!!!");
        return;
    }
}
```

二. 地图及图层的基本操作

(1) 功能说明

实现地图的创建、添加、保存、浏览及图层的基本操作。

(2) 基本思路

Imap 及 ImapLayer 接口提供了常用的地图及图层操作的接口函数。

(3) 部分代码

```
//地图的创建
void CMyView::OnCreatMap()
{
 LPWSTR m_NewMapName;
 IMap     *p_NewMap;
 short    sflg;
 USES_CONVERSION;
//地图名
 m_NewMapName=A2W("TestMap");
 if (pGisDoc)
 {
     //创建地图
     pGisDoc->CreateMap(m_NewMapName,&p_NewMap,&sflg);
     p_NewMap->put_Visible(true); //设置地图为可见
     if (sflg)
     {
         MessageBox("创建成功!!!");
         pGisDoc->Save();
         p_NewMap->Release();
         p_NewMap=NULL;
     }
     else
         MessageBox("创建失败!!!");
 }
```

```
 else
 {
      MessageBox("未打开文档!!!");
 }
}

//添加图层
void CMyView::OnAddLayer()
{
 IMap      *pTempMap=0;
 IFeatureLayer* pFtLayer = NULL;
 USES_CONVERSION;
 CoCreateInstance(CLSID_FeatureLayer,0,CLSCTX_INPROC,IID_IMapLayer,
                                    (void**)&pMapLayer);   //创建 ImapLayer 接口
 if (pGisDoc)
 {
      //打开要素类
      OnOpenFcls();
      pGisDoc->get_CurMap(&pTempMap);   //取当前激活地图
      if (m_ptFCls)
           {
                char m_ptFClsname[256]="";
                D_RECT m_rc;
                Envelope newVal;
                m_ptFCls->fcls_GetName(m_ptFClsname,256); //名称
                m_ptFCls->fcls_GetRange(m_rc);   //范围
                newVal.XMax=m_rc.xmax;
                newVal.XMin=m_rc.xmin;
                newVal.YMin=m_rc.ymin;
                newVal.YMax=m_rc.ymax;
                if (pMapLayer)
                {
                     //设置图层的基本信息
                     pMapLayer->QueryInterface(IID_IMapLayer,
                                                 (void**)&pFtLayer);
                     //设置数据源信息,图层保存的数据源一般与当前地图文档一致
                     pFtLayer->SetGDBLogInfo(A2W(path.svcName),path.gdbID,
                                          A2W(user),A2W(pwsd),(long)xclsID);
                     //设置图层名
                     pMapLayer->put_LayerName(A2W(m_ptFClsname));
                     //设置图层范围
                     pMapLayer->put_LayerBound(newVal);
                     pMapLayer->put_Editable(true);   //设置为可编辑
```

```
                    pMapLayer->put_Visible(true);     //设置为可见
                    pMapLayer->put_MaxScale(100000);   //设置最大显示比例
                    pMapLayer->put_MinScale(0);       //设置最小显示比例
                    //绑定要素类
                    pMapLayer->Attach_XClsHandle((LONG)m_ptFCls);
                    //添加图层
                    if ((pTempMap->AddLayer(pMapLayer))==S_OK)
                    {
                        MessageBox("添加新图层成功!!!");
                        pGisDoc->Save();
                        pMapLayer->Release();
                        pMapLayer=NULL;
                    }
                }
            }
 }
 else
 {
      MessageBox("未打开地图文档!!!");
 }
}

//实现 OnOpenFcls,打开要素类
void CMyView::OnOpenFcls()
{
 CGDBServer        m_GDBSvr=NULL;   //数据源
 CGDataBase         *m_ptGDB=NULL;   //地理数据库
 CFeatureCls        *m_ptFCls=NULL;     //要素类
 long            flag=0;
 long            ptSelFlag;            //文件类型
 long            selN=1;               //文件类型个数
 long            type;             //打开的文件类型
 GDBPATHINFO          path;
 char            user[30]= "";
 char            pwsd[64]= "";

 m_ptGDB = new CGDataBase;//地理数据库对象
 ptSelFlag = XCLS_TYPE_FCLS;//设定弹出的对话框为选择要素对话框
 if(ShowGDBShellDlg(path,&ptSelFlag,selN,0,"选择类")<=0)//弹出选择要素对话框
```

```
        return;
    GetLoginInfo(path.svcName,user,30,pwsd,64);//获取数据源的信息
    if(m_GDBSvr.Connect(path.svcName,user,pwsd)<=0)//连接数据源
        return;
    if(m_ptGDB->Open(&m_GDBSvr,path.gdbName)<=0)//打开数据库
        return;
    type    = path.xclsInf[0].xclsType;
    xclsID = path.xclsInf[0].xclsID;

    if(type !=XCLS_TYPE_FCLS)
        return;
    m_ptFCls=new CFeatureCls;//要素类
    flag=m_ptFCls->fcls_Open(m_ptGDB,xclsID,1);                    //打开要素类
      if (flag>0)
       {
            MessageBox("要素类已经打开");
       }
       else
       {
            MessageBox("不能打开该要素类");
       }
  }
```

三.自定义图层

(1) 功能说明

mapgis70 支持自定义图层,用户可在自定义图层中存储自定义的数据内容,如:文本和其他文句类型,用户可以通过编码来给定自定义图层的属性,或附加其他功能菜单 等等多重操作.

本例中,在自定义图层中添加文本注释,同时添加了一个属性页以及两个功能菜单!

(2) 基本思路

a.自定义图层的接口必须继承自 IMapLayer，并实现 IMapLayer 的所有接口函数:

```
STDMETHOD (get_MinScale)(double* pVal);
STDMETHOD (put_MinScale)(double newVal);
STDMETHOD (get_MaxScale)(double *pVal);
STDMETHOD (put_MaxScale)(double newVal);
STDMETHOD (get_LayerName)(BSTR *pVal);
STDMETHOD (put_LayerName)(BSTR newVal);
STDMETHOD (get_AreaOfInterest)(Envelope *pVal);
STDMETHOD (get_Cached)(BOOL *pVal);
STDMETHOD (put_Cached)(BOOL newVal);
```

```
STDMETHOD (put_SpatialReference)(long newVal);
STDMETHOD (get_Valid)(BOOL *pVal);
STDMETHOD (get_Visible)( BOOL *pVal);
STDMETHOD (put_Visible)(BOOL newVal);
STDMETHOD (Draw)(short typeFlg,IDisplay *pDisplay,ICancel *pCancel);
STDMETHOD (get_LayerIndex)(short *pVal);
STDMETHOD (put_LayerIndex)(short newVal);
STDMETHOD (get_LayerBound)(Envelope *pVal);
STDMETHOD (put_LayerBound)(Envelope newVal);
STDMETHOD (ConnectData)(short *pVal);
STDMETHOD (get_Editable)(BOOL *pVal);
STDMETHOD (put_Editable)(BOOL newVal);
STDMETHOD (get_Active)(BOOL *pVal);
STDMETHOD (put_Active)(BOOL newVal);
```

b.自定义图层必须实现 IPersistStream 接口，以支持自定义图层自身对象的存取需要。

```
//IPersist，是IPersistStream的父接口
STDMETHOD (GetClassID)(CLSID *pClassID);
```

说明：通过 GetClassID 让 Map 正确的识别和创建自定义图层的组件对象

```
//IPersistStream
STDMETHOD (IsDirty)();
```

说明：通过IsDirty判断该自定义图层是否存在脏数据以决定是否存储该自定义图层 组件对象的数据或者更新。

```
STDMETHOD (Load)(IStream *pStm);
```

说明：通过Load读取该自定义图层自身对象数据。

```
STDMETHOD (Save)(IStream *pStm,BOOL fClearDirty);
```

说明：通过Save存储该定义图层的自身对象数据。

```
STDMETHOD (GetSizeMax)(ULARGE_INTEGER *pcbSize);
```

说明：通过 GetSizeMax 获得需要存放该自定义图层的数据所需要的最大空间，以加快存取速度。

c.自定义图层必须在组件注册信息中添加 IMapLayer 的分类信息，通过使用以下语 句

```
BEGIN_CATEGORY_MAP(CCustomXXXXXLayer)
    IMPLEMENTED_CATEGORY(CATID_IMapLayer)
END_CATEGORY_MAP()
```

进行组件分类注册，其中 CATID_IMapLayer 的定义包含在 CATID.h 的头文件中。

(3) 部分代码

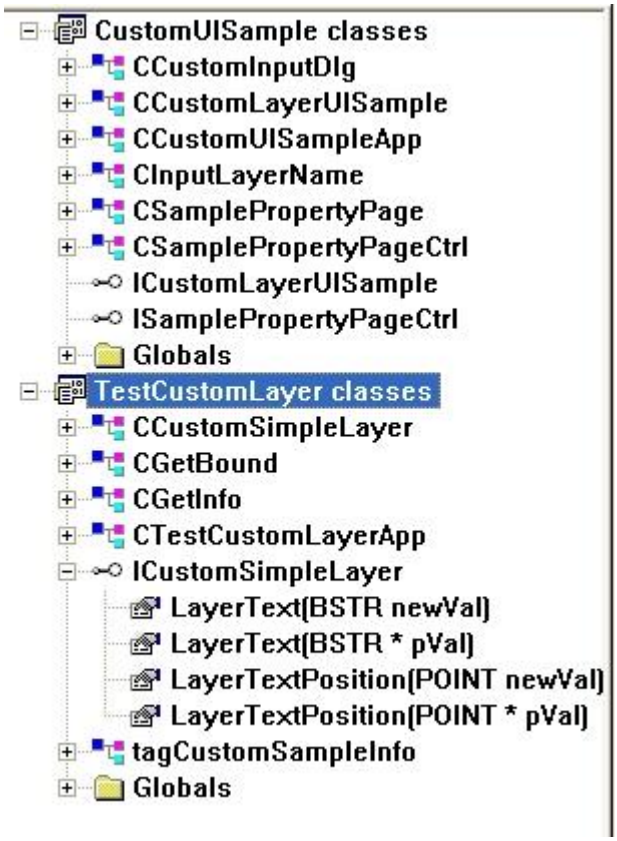


图 9.3 自定义图层类视图

```
import "oaidl.idl";
import "ocidl.idl";
import "GisMap.idl";
    [
        object,
        uuid(B848ADCB-E46E-46AE-A9BA-BA91ECE10219),

        helpstring("ICustomSimpleLayer InterFace"),
        pointer_default(unique)
    ]
    interface ICustomSimpleLayer : IMapLayer
    {
        [propget, helpstring("Layer Text")] HRESULT LayerText([out, retval] BSTR *pVal);
        [propput, helpstring("Layer Text")] HRESULT LayerText([in] BSTR newVal);
        [propget, helpstring("Layer Text Position")] HRESULT LayerTextPosition([out, retval] POINT *pVal);
        [propput, helpstring("Layer Text Position")] HRESULT LayerTextPosition([in] POINT newVal);
    };
```

图 9.4 自定义图层接口定义四个属性

```
BEGIN_COM_MAP(CCustomSimpleLayer)
    COM_INTERFACE_ENTRY(ICustomSimpleLayer)
    COM_INTERFACE_ENTRY(IMapLayer)
    COM_INTERFACE_ENTRY(IPersistStream)
    COM_INTERFACE_ENTRY(IMPICommand2)
    COM_INTERFACE_ENTRY(IMPICommand)
END_COM_MAP()
```

图 9.5 用户添加接口

实例中添加 ImapLayer 接口实现图层的管理,IPersistStream 接口实现自定义图 层 自身对象的存取，IMPIComman 及 IMPIComman2 接口实现资源按钮响应事件等操作！

```
//取自定义图层的 CLSID
STDMETHODIMP CCustomSimpleLayer::GetClassName(GUID* name)
{
    if(name)
    {
        *name=CLSID_CustomSimpleLayer;
        return S_OK;
    }
    return E_POINTER;
}

//所添加的两个菜单项
STDMETHODIMP CCustomSimpleLayer::OnCommand(UINT idCommand,
                                    LONGidExten,LONG lParam,BSTR text)
{
    switch(idCommand)
    {
    case ID_COMMAND1:
        break;
    case ID_COMMAND2:
        break;
    }
    return S_OK;
}

//取 UICLSID,在 UICLASS 中将定义图层的属性,名称等属性!
STDMETHODIMP CCustomSimpleLayer::get_UIClsID(BSTR* UIClassID)
{
    USES_CONVERSION;
    *UIClassID = ::SysAllocString(A2W(
                    "CustomUISample.CustomLayerUISample.1"));
    return S_OK;
}

//初始化图层时调用
STDMETHODIMP CCustomLayerUISample::ShowInitDataUI(long
        hParent,IMapLayer* pMapLayer, long* sflg)
{
    USES_CONVERSION;
    CLocalResource local(_Module.m_hInstResource);
    CCustomInputDlg dlg;     //用来输入图层的基本信息
    CInputLayerName  dlg1;   //用来输入图层名
    if(dlg.DoModal() == IDOK)
```

```
    {
        ICustomSimpleLayer* pCustomLayer = NULL;
        POINT point;
        BSTR bstr;
        BSTR m_layername;
        if(SUCCEEDED(pMapLayer->QueryInterface(
                IID_ICustomSimpleLayer,(void**)&pCustomLayer)))
        {
        //初始化图层内容,位置等信息!
            point.x = dlg.m_nTextX;
            point.y = dlg.m_nTextY;
            bstr = dlg.m_csText.AllocSysString();
            pCustomLayer->put_LayerTextPosition(point);     //注释的位置
            pCustomLayer->put_LayerText(bstr);       //注释内容
            ::SysFreeString(bstr);
            bstr = NULL;
            if (dlg1.DoModal()==IDOK)
                {
                //图层名设置
                m_layername=A2W(dlg1.m_LayerName);
                pCustomLayer->put_LayerName(m_layername);   //设置图层名
                *sflg=1;
                }
        };
    }
    return S_OK;
}

//CustomSimpleLayer.rgs 文件,自定义图层的注册
HKEY_CURRENT_USER
{
    SOFTWARE
    {
        MapGis
        {
            FrameWork70
            {
            Layers
                {
                    TestCustomLayer.CustomSimpleLayer.1 =
                            b 'F85BCEF88AFF4FEDAF5A244DE4652042'
                    {
                        val descript = s '测试示例图层'
                        val otherkey = s 'customkey'
```

```
                                restrict
                                {

                                }
                            }
                        }
                    }
                }
            }
        }
```

自定义图层加载到地图中的效果如下:

图 9.6 自定义图层注册后出现在图层类型列表中

图 9.7 初始化图层信息

图 9.8 输入图层名

图 9.9 自定义图层加载到地图中的效果

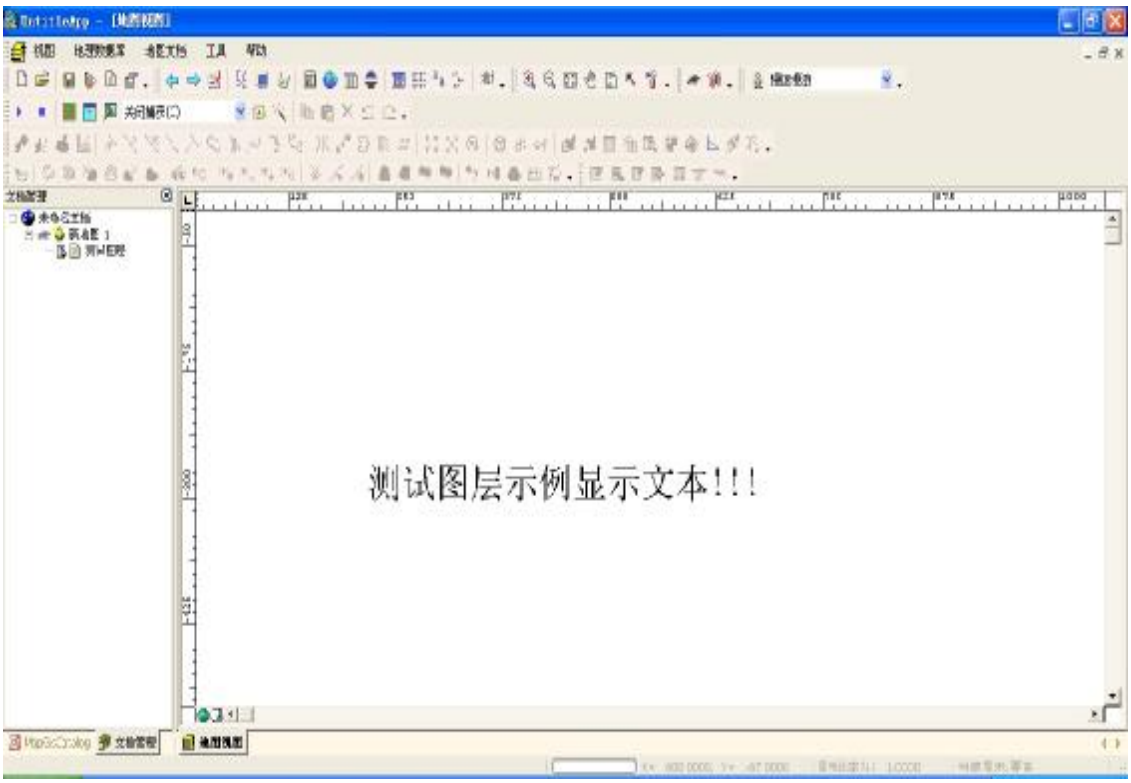


图 9.10 显示图层中的文本数据

第 10 章 地图可视化

10.1 概述

本章将介绍地图数据在视窗中的可视化，数据显示模式和版面显示模式，视口和排版框定义，放大、缩小地图等基本操作，以及与地图数据显示相关的标尺，滚动条，状态栏和制图元素等的基

10.1.1 定义、缩写

地图文档 ： 地图窗口的客户区

视口 ： 指地图排版（LayOut）中排版框（LayOutFrame）所对应的客户区的区域，也可泛指常规显示下的地图窗口客户区或是地图版面模式下的客户区.参照坐标系 ： 多个地图使用不同的坐标系，在一个视图上显示这些地图时需要做投影变换，被选来作为基准的坐标系就是参照坐标系.窗口坐标系 ： 原点在客户区左上角，X 轴正方向朝右，Y 轴正方向朝下.设备坐标系 ： 原点在客户区左下角，X 轴正方向朝右，Y 轴正方向朝上.逻辑坐标系 ： 也就是地图使用的数据坐标系或是地理坐标系，地图数据实际使用的坐标系数据视图模式 ： 地图的常规显示，显示的窗口范围一般是地图视图客户区范围.版面视图模式 ： 地图排版显示，版面中可显示多个地图排版框，指北针，图例等.坐标转换对象 ： 管理显示参数，设备参数，参照坐标系，各种坐标参数之间的转换

10.2 结构图

地图可视化部分的结构图如图所示:

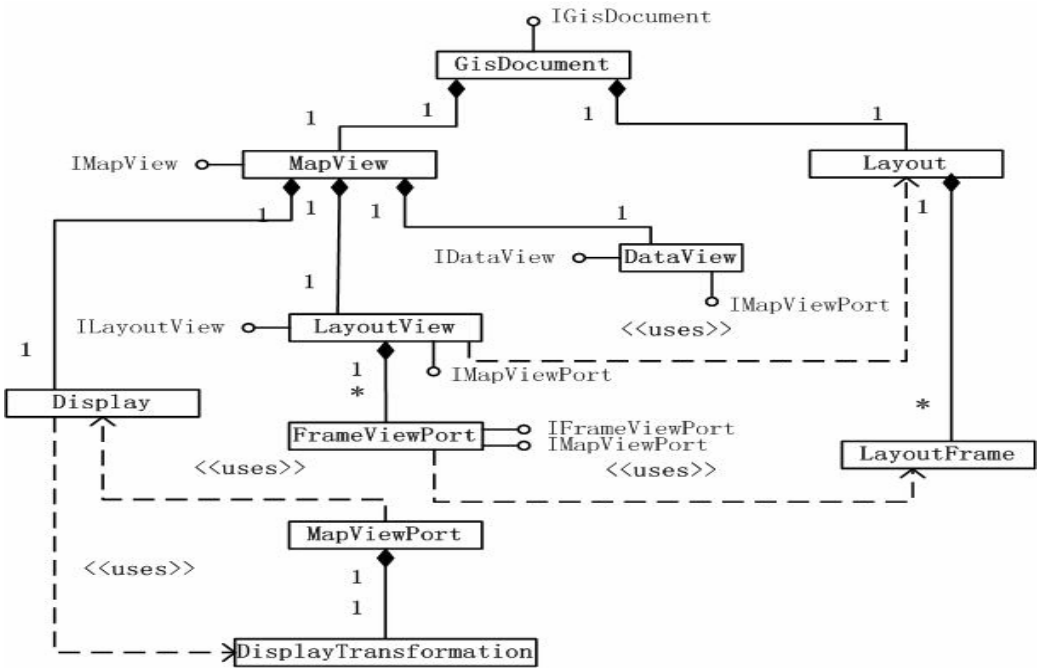


图 10.1 地图可视化部分的结构图

10.3 主要模块及接口说明

1. CMapGisView 模块

该类是显示地图基础视窗类，提供了基本的地图显示，基本的地图窗口的操作以及绑定地图视窗类到指定视窗或对话框上的功能。

主要方法:

(1)窗口操作函数 :包括对窗口的放大 、缩小、更新、复位、移动、选择数 据集合、清空集合、显示 上一级窗口、显示模式的设置等操作。

(2)基本显示函数

(3)窗口绑定函数

(4)设置地图范围

(5)取用户设置的数据集合链表接口

(6)显示用户添加的地图数据

(7)删除用户记录的数据

(8)设置是否显示滚动条标志

2. GisRender 模块

该模块提供了基本的显示函数和创建各种驱动函数，如显示驱动、图像文件驱动，打印驱动，还提供了与要素相关的显示函数. 主要接口说明:

IDisplayDrv	显示驱动接口.主要包含创建和销毁 BMP、WIN 打印和栅格文件的驱动操作。
IDisplay	基本显示接口.各种元素（要素、几何、空间数据）的显示和绘制及参数的设定 。
IScreenDisplay	屏幕显示接口.窗口相关信息的获取和设置，如窗口句柄等。
IDisplayOption	显示参数接口.提供了修改显示的各种控制参数。
ITransformation	坐标转换接口.实现了各种坐标之间的转化操作以及设置投影参照系、地图显示参数等。

3. GisViewPort 模块

该模块定义了数据视图和版面视图，并提供两种视图模式下相应的操作

主要接口说明:

IdataViewRender	提供了加亮或擦除给定单个元素和集合的显示接口。
ImapView	与地图文档相关联的地图视图接口，管理数据视图和版面视图。
IDataView	管理常规数据显示的接口，提供常规地图数据的显示与管理 。
ILayoutView	版面视图接口，管理版面及排版框。
IFrameViewPort	与排版框对应的排版框视口，提供排版框视口的数据显示如：排版框视口的绘制、参数设置等操作。
ImapViewPort	视口基类接口，提供地图视口的数据显示等操作，包括视口数据的绘制、更新视口、放大缩小显示图形的参数和附加窗口等。

4. ToolExtension 模块

该模块实现了几个基本的功能组件，包括选择工具、放大地图、缩小地图、移动地图功能。

主要接口说明:

IZoomIn	放大接口
IRestoreTool	复位接口
IZoomOut	缩小接口

IMoveTool　　移动工具
ISelectTool　　选择工具接口

5. GisLayout 模块

该模块实现了排版框的创建、绘制及其相关参数设置。

主要接口说明:

IlayoutFrame　　取排版框 ID、设置和获得排版框的范围和边界参数操作。
Ilayout　　版面框的创建、删除、遍历以及获得版面框的信息。

6. CartoElement 模块

该模块实现了地图修饰类图例、专题图、指北针、比例尺等的创建、绘制、显示及其属性的修改。

主要接口说明:

ICartoElement　　修饰类的创建、绘制、更新、以及显示范围、名称、显示模式的设置等操作。
ILegendElement　　图例和图例标题的设置和获得；图例属性参数的设置和获得。
IScaleBarElement　　比例尺模板和比例尺的创建；获得比例尺模板及比例尺的信息的操作。
INorthArrowElement　　指北针的创建；指北针模板和指北针信息的获得。
IGraphElement　　专题图的创建和信息获得。

7. ViewOptionParam 模块

该模块实现了视图显示参数的设置

10.4 应用实例

1. 地图的显示

(1) 功能说明

打开地图文档,在视窗中显示当前激活地图.

(2) 基本思路

建立工程的视图类继承 CMapGisView 类.

(3) 部分代码

```
//CMyView drawing
void CMyView::OnDraw(CDC* pDC)
{
    CDocument* pDoc = GetDocument();
    CMapGisView::OnDraw(pDC);
}

//打开地图文档并显示
void CMyView::OnOpenMapDocument()
{
    //取应用框架接口
    GetGisAppFrame(&pApp);
    short sflg;
```

```
    if(pApp)
    {
    //取地图文档
    pApp->get_GisDocument(&pGisDoc);
    if(pGisDoc)
    {
        打开地图文档
        pGisDoc->Open(0,OPEN_NORMAL,&sflg);
        if (sflg)
        {
            MessageBox("打开地图文档成功!!!");
            //取地图文档中的当前地图
            pGisDoc->get_CurMap(&pMap);
            if(pMap)
            {
                //添加显示,并调整显示范围
                AppendUserMap(pMap);
                SetUserObjectRange();
                RestoreWnd();
            }
        }
        else
        {
        //释放接口
        pGisDoc->Release();
        pGisDoc=NULL;
        }
    }
    }
}
```

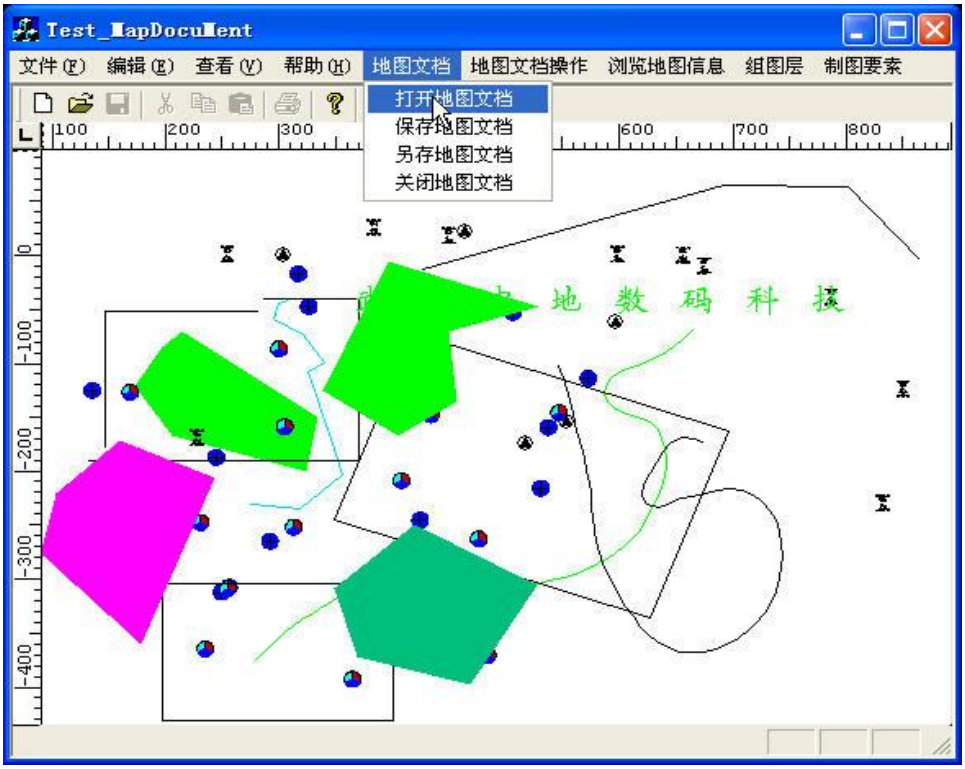


图 10.2 显示地图文档

2. 通过 Idisplay 显示接口显示要素

(1) 功能说明

添加打开的要素类在视图窗口.

(2) 基本思路

GisRender 模块提供了多种显示接口,通过定义好的接口函数可以达到显示要素、几何、空间数据以及其他注释等等.

(3) 部分代码

```
//实现当前视图类的 OnOwnerDraw
void CMyView::OnOwnerDraw(IDisplay *pDisplay)
{
    Envelope    pDspEnv;
    D_RECT      rc;
    short       srtflg;
    CMapGisView::OnOwnerDraw(pDisplay);

    if (pDisplay)
    {
```

```
            //m_ptFCl 为打开的要素类,参见前一掌 OnOpenFcls()方法
            if (m_ptFCls)
            {
                //设置显示范围
                m_ptFCls->GetRange(rc);
                pDspEnv.XMin=rc.xmin;
                pDspEnv.XMax=rc.xmax;
                pDspEnv.YMin=rc.ymin;
                pDspEnv.YMax=rc.ymax;
                pDisplay->DispFeatureCls((long)m_ptFCls,&pDspEnv,&srtflg);

            }
        }
    }
```

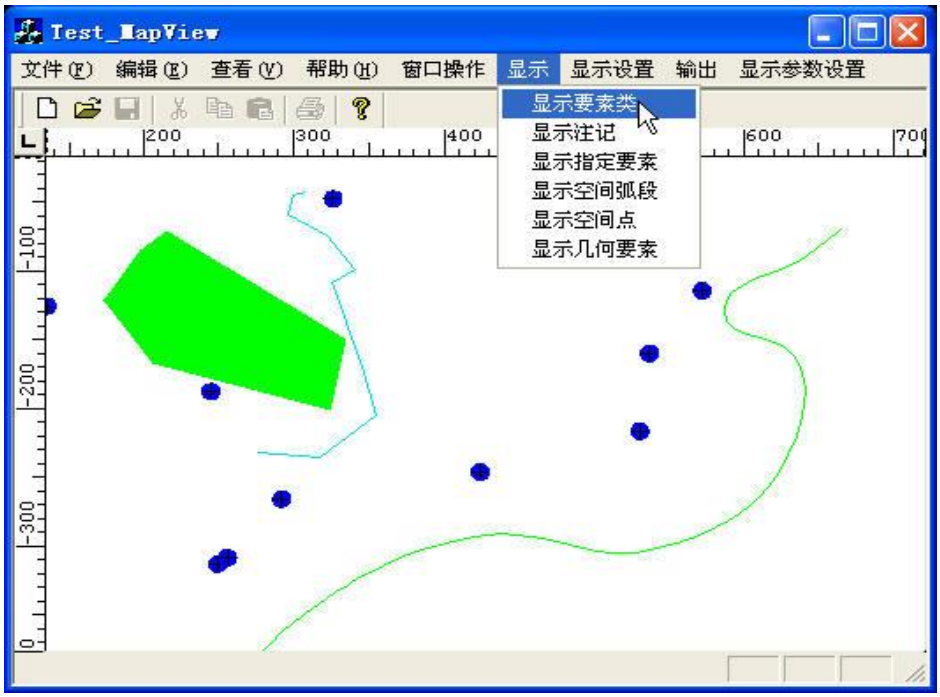


图 10.3 显示要素类

3. 输出 BMP、GIF 图像

(1) 功能说明

将当前视图中显示的数据输出为 BMP、GIF 等格式的图像.

(2) 基本思路

创建输出 BMP 图像驱动,将当前显示视图显示的数据在位图设备中绘制,从而输出 位图.

(3) 部分代码

```
//函数第一个参数为输出图像的名称,第二个参数为位图的格式,0/1 分别表示 BMP/GIF    //格式
void CMyView::OnPutOut(CString m_Filename,short ImgType)
{
USES_CONVERSION;
IDisplayDrv* pDisplayDrv=0;   //显示驱动接口
short          sflg;
ITransformation    *pTransformation;   //转换接口
Envelope       envolopedev;
Envelope       envolope;
short              sflag;

OnOpenFCLS();
CommandFlag=1;
if (m_ptFCls)
{
    if (pDisplay)
    {
        //查询驱动接口并创建 BMP 驱动
        pDisplay->QueryInterface(IID_IDisplayDrv,(LPVOID*)&pDisplayDrv);
        pDisplayDrv->CreateBMPDrv(A2W(m_Filename),800,600,
                                                ImgType,0,NULL,&sflg);
        if (sflg)
        {
            pDisplay->get_DisplayTransformation(&pTransformation);
            envolopedev.XMin=0;
            envolopedev.YMin=0;
            envolopedev.XMax=800;
            envolopedev.YMax=600;
            //设置设备范围
            pTransformation->SetDeviceRect(&envolopedev,&sflag);
            if(sflag !=      1)
            {
                AfxMessageBox("设置设备范围不成功");
                pDisplayDrv->DestroyBMPDrv();       //关闭位图设备
                return ;
            }
            D_RECT rc;
            m_ptFCls->GetRange(rc);
            envolope.XMin=rc.xmin;
            envolope.XMax=rc.xmax;
            envolope.YMin=rc.ymin;
            envolope.YMax=rc.ymax;
            pTransformation->SetDispRect(&envolope,&sflag);   //显示指定区域
            if(sflag !=1)
```

```
                {
                    AfxMessageBox("设置显示范围不成功");
                    pDisplayDrv->DestroyBMPDrv();       //关闭位图设备
                    return ;
                }
                //在位图设备中画要素类
                pDisplay->DispFeatureCls((long)m_ptFCls,&envolopedev,&sflag);
                if(sflag !=       1)
                {
                    AfxMessageBox("显示要素类不成功");
                    pDisplayDrv->DestroyBMPDrv();       //关闭位图设备
                    return ;
                }
                pDisplayDrv->DestroyBMPDrv();       //关闭位图设备
                pDisplayDrv->Release();             //释放接口
                pDisplayDrv = 0;
                pTransformation->Release();
                pTransformation = 0;
            }
        }
    }
    }
```

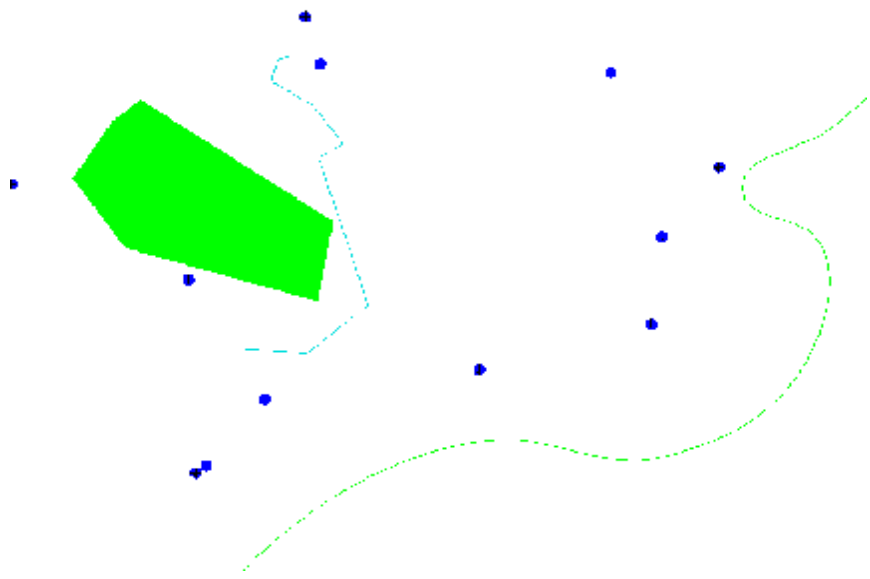


图 10.4 输出的 GIF 图像

4. 基本图形的绘制

(1) 功能说明

使用显示接口 IdisPlay 实现基本图形的绘制,包括点、折线、曲线、圆、弧、字符　串等等.

(2) 基本思路

首先通过取显示接口以及转换接口,转换得到设备坐标;然后调用显示接口下提供 的函数进行基本图像的绘制!

(3) 部分代码

```
//画点
void CMyView::OnDrawCrossDot()
{
ITransformation *pTans=NULL;
D_DOT           pos;

//取显示接口
GetDispObj(&pDisplay);
//取转换接口
if (pDisplay)
{
    pDisplay->get_DisplayTransformation(&pTans);
    if (pTans)
    {
        //逻辑坐标转换得到设备坐标
        pTans->LpToDp2(200,200,&pos.x,&pos.y);
        //画点
        pDisplay->DrawCrossDot(pos.x,pos.y,6);
        RestoreWnd();
        //释放接口
        pTans->Release();
        pTans=NULL;
    }
    pDisplay->Release();
    pDisplay=NULL;
}
}
//画线
void CMyView::OnLineTo()
{
ITransformation *pTans=NULL;
D_DOT           pos1;
D_DOT           pos2;

//取显示接口
GetDispObj(&pDisplay);
if (pDisplay)
{
    //取转换接口
    pDisplay->get_DisplayTransformation(&pTans);
```

```
    if (pTans)
    {
        //设置画笔
        pDisplay->SetPen(5,7);
        //逻辑坐标转换为设备坐标
        pTans->LpToDp2(100,100,&pos1.x,&pos1.y);
        pTans->LpToDp2(300,300,&pos2.x,&pos2.y);
        //画线
        pDisplay->MoveTo(pos1.x,pos1.y);
        pDisplay->LineTo(pos2.x,pos2.y);
        RestoreWnd();
        //释放接口
        pTans->Release();
        pTans=NULL;
    }
pDisplay->Release();
pDisplay=NULL;
}
}

//画圆
void CMyView::OnCircle()
{
ITransformation *pTans=NULL;
D_DOT           pos;

//取显示接口
GetDispObj(&pDisplay);
if (pDisplay)
{

//取转换接口
pDisplay->get_DisplayTransformation(&pTans);
if (pTans)
{
    //设置画笔
    pDisplay->SetPen(5,7);
    //逻辑坐标转换为设备坐标
    pTans->LpToDp2(300,300,&pos.x,&pos.y);
    //画圆
    pDisplay->MoveTo(pos.x,pos.y);
    pDisplay->Circle(100);
    RestoreWnd();
    //释放接口
```

```
        pTans->Release();
        pTans=NULL;
}
pDisplay->Release();
pDisplay=NULL;
}
}

//画贝赛尔曲线
void CMyView::OnBzier()
{
ITransformation *pTans=NULL;
D_DOT               posset[3];

//取显示接口
GetDispObj(&pDisplay);
//设置控制点坐标
posset[0].x=100;posset[0].y=100;
posset[1].x=300;posset[1].y=200;
posset[2].x=400;posset[2].y=300;
if (pDisplay)
{

//取转换接口
pDisplay->get_DisplayTransformation(&pTans);
if (pTans)
{
        //设置画笔
        pDisplay->SetPen(5,7);
        //逻辑坐标转换为设备坐标
        pTans->LpToDpArray((LPDotSet)posset,3);
        //画曲线
        pDisplay->Bzier((LPDotSet)posset,3,20);
        RestoreWnd();
        //释放接口
        pTans->Release();
        pTans=NULL;
}
pDisplay->Release();
pDisplay=NULL;
}
}
//画填充圆
void CMyView::OnCircleFill()
```

```
{
ITransformation *pTans=NULL;
D_DOT              pos;
//取显示接口
GetDispObj(&pDisplay);

if (pDisplay)
{

//取转换接口
pDisplay->get_DisplayTransformation(&pTans);
if (pTans)
{
    //设置画笔
    pDisplay->SetBrush(7,10,10,10,30);
    //逻辑坐标转换为设备坐标
    pTans->LpToDp2(300,300,&pos.x,&pos.y);
    //画填充圆
    pDisplay->MoveTo(pos.x,pos.y);
    pDisplay->CircleFill(100);
    RestoreWnd();
    //释放接口
    pTans->Release();
    pTans=NULL;
}
pDisplay->Release();
pDisplay=NULL;
}
}

//画区
void CMyView::OnPolyPolyGon()
{
ITransformation *pTans=NULL;
D_DOT              posset[4];
LPLongLst          lne[5];
int                na;

//取显示接口
GetDispObj(&pDisplay);

posset[0].x=100;posset[0].y=100;
posset[1].x=300;posset[1].y=200;
posset[2].x=400;posset[2].y=300;
```

```
posset[3].x=100;posset[3].y=100;
lne[0]=4;
if (pDisplay)
{

//取转换接口
pDisplay->get_DisplayTransformation(&pTans);
if (pTans)
{
    //设置画刷
    pDisplay->SetBrush(7,10,10,10,0);
    //逻辑坐标转换为设备坐标
    pTans->LpToDpArray((LPDotSet)posset,4);
    //画区
    pDisplay->PolyPolyGon((LPDotSet)posset,(LPLongLst)lne,1);
    RestoreWnd();
    //释放接口
    pTans->Release();
    pTans=NULL;
}
pDisplay->Release();
pDisplay=NULL;
}
}

//输出字符串
void CMyView::OnStringOut()
{
USES_CONVERSION;
ITransformation *pTans=NULL;
D_DOT               pos;
char            str[128]="";

strcpy(str,"武汉中地数码科技有限公司");
//取显示接口
GetDispObj(&pDisplay);
if (pDisplay)
{

//取转换接口
pDisplay->get_DisplayTransformation(&pTans);
if (pTans)
{
    //逻辑坐标转换为设备坐标
```

```
        pTans->LpToDp2(100,100,&pos.x,&pos.y);
        //设置画笔
        pDisplay->SetPen(5,7);
        //输出字符串
        pDisplay->StringOut(pos.x,pos.y,50,50,A2W(str),0,0,0,0,0);
        RestoreWnd();
        //释放接口
        pTans->Release();
        pTans=NULL;
    }
    pDisplay->Release();
    pDisplay=NULL;
    }
    }
```

第 11 章 基本显示

11.1 结构图

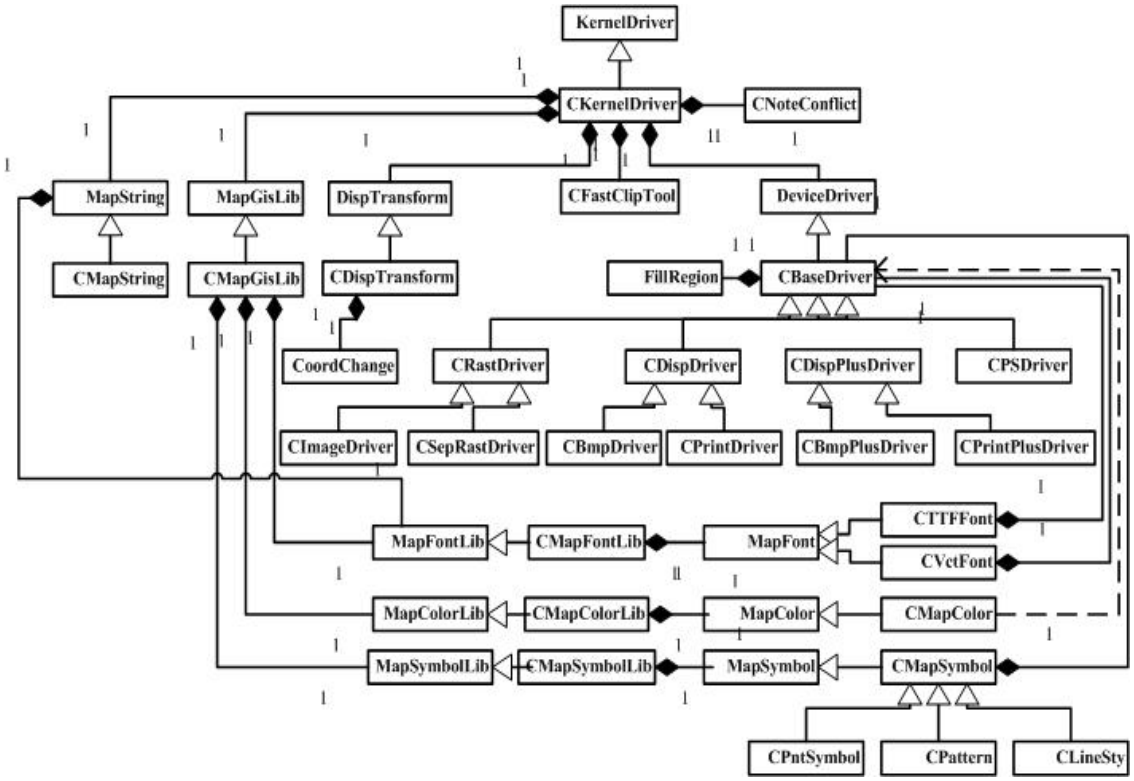


图 11-1 基本显示和系统库管理结构图

11.2 主要模块及接口说明

1. DisplayDriver 模块

   提供了六类功能函数：

   （**1**）设备初始化及结束处理；

   （**2**）打印输出使用的虚函数；

   （**3**）设置输出文件层；

   （**4**）设置显示笔或刷子的状态；

   （**5**）基本显示函数以及被基本函数延伸调用的函数；

   （**6**）释放自身资源。

2. KernelDriver 模块

   提供参数设置函数和基本显示函数的接口

   可以分为八类功能函数：

   （**1**）设备绑定和取消设备绑定

（2）变换参数处理
（3）设备初始化及结束处理
（4）输出使用的虚函数
（5）设置输出文件层
（6）输出参数设置
（7）基本显示函数
（8）设置显示状态控制

3. 驱动创建模块

```
//创建显示对象
KernelDriver* CreateDispDriver(HWND hwnd,HDC hdc);
//创建 GdiPlus 显示对象
KernelDriver* CreateDispPlusDriver(HWND hwnd,HDC hdc);
//创建位图高质量驱动对象
KernelDriver* CreateBmpPlusDriver(char* FileName,short wid,short hei,
                    short type=0, short GifTansparentflg=0,long lBkClr =9 );
//创建 windows 高质量打印驱动对象
KernelDriver* CreateWinPrintPlusDriver(D_RECT LPRect,D_RECT DPRect,
                                                    HDC PrintDC);
//创建 SVG 驱动对象
KernelDriver* CreateSVGDrv(char* FileName, short wid, short hei);
//创建 windows 打印对象
KernelDriver* CreateWinPrintDriver(D_RECT LPRect,   D_RECT DPRect,
                                                    HDC PrintDC);
//创建位图对象
KernelDriver* CreateBmpDriver(char* FileName, short wid, short hei,
                    short type=0, short GifTansparentflg=0,long lBkClr= 9);
//创建光栅对象(内存方式)
KernelDriver *CreateMemRastDriver(double wid, double hei, D_RECT DPRect,
                                                    HDC PrintDC,short mode);
//创建光栅对象(文件方式)
KernelDriver *CreateFileRastDriver(char *FileName, short resolution,
          short mode, double wid/*=0*/, double hei/*=0*/,D_RECT PageRect);
//创建一个 PSDriver
KernelDriver *CreatePSDriver(char *FileName, D_RECT *rect);
//创建一个 EPSDriver
KernelDriver *CreateEPSDriver(char *FileName, D_RECT *rect);
```

4. DeviceDriver 对象

定义纯虚函数

5. DispTransform 对象

可以分为四类功能函数：

（1）设置或获取显示范围、视口范围；
（2）置、取转换参数；
（3）逻辑坐标与设备坐标互换；
（4）显示范围及参数控制。

6. CNoteConflict 对象

提供解决注记冲突的监测和记

11.3 应用实例

基本图形的绘制

(1) 主要功能

通过创建显示驱动,进行基本图形的绘制.

(2) 基本思路

KernelDriver 模块提供了基本的图形绘制方法,包括子图、线型、图案、三点弧等 等， 通 过 CreateDispDriver 创建显示驱动，返回 KernelDriver 对象.

(3) 部分代码

```
//创建显示驱动
void CMyView::OnCreatDispDriver()
{
    KernelDriver        *m_KernelDriver=NULL;
    CDC                 *pDC;
    pDC=GetDC();
    m_KernelDriver = CreateDispDriver(m_hWnd,pDC->m_hDC); //创建驱动对象
    if (m_KernelDriver)
    {
        MessageBox("创建成功!!!");
    }
}

//画直线
void CMyView::OnDrawBeeLine()
{
ITransformation         *pTransformation; //转换接口
CDC                     *pDC;
KernelDriver            *m_KernelDriver=NULL;
D_DOT                   pos1;
D_DOT                   pos2;
IDisplay                *pDisplay;

    pos1.x=100;pos1.y=100;
    pos2.x=300;pos2.y=300;

    //设置画笔模式
    m_KernelDriver->SetPen(5,6);
    //画线
    m_KernelDriver->MoveTo(pos1);
    m_KernelDriver->LineTo(pos2);
    m_KernelDriver->Free();
    m_KernelDriver=NULL;
}
```

```
ReleaseDC(pDC);
}

//视图范围内交互绘制图形(以画折线为例,部分变量定义参见实例)
//鼠标左键按下事件
void CMyView::OnLButtonDown(UINT nFlags, CPoint point)
{
    BOOL    m_LButtonDown=TRUE;
    m_End.x=point.x;m_End.y=point.y;
    m_Start.x=point.x;m_Start.y=point.y;
    m_KernelDriver->Line(m_Start.x,m_Start.y,m_End.x,m_End.y,7);

    CMapGisView::OnLButtonDown(nFlags, point);
}

//鼠标 move 事件
void CMyView::OnMouseMove(UINT nFlags, CPoint point)
{
    if (m_LButtonDown)
    {
        pos2.x=point.x;pos2.y=point.y;
        m_KernelDriver->SetPenMode(7);
        //异或线搽掉前一次画出来的线
        m_KernelDriver->Line(m_Start.x,m_Start.y,m_End.x,m_End.y,6);
        m_End=pos2;
        //画新的线
        m_KernelDriver->Line(m_Start.x,m_Start.y,m_End.x,m_End.y,6);
    }
    CMapGisView::OnMouseMove(nFlags, point);
}

//鼠标左键弹起事件
void CMyView::OnLButtonUp(UINT nFlags, CPoint point)
{
    if (AppendRectFlag)
    {
        m_LButtonDown=false;
    }

    CMapGisView::OnLButtonUp(nFlags, point);
}

//鼠标右键弹起事件
void CMyView::OnRButtonUp(UINT nFlags, CPoint point)
{
    pos2.x=point.x;pos2.y=point.y;
    m_KernelDriver->Line(m_Start.x,m_Start.y,pos2.x,pos2.y,7);
    m_LButtonDown=false;
    CMapGisView::OnRButtonUp(nFlags, point);
}
```

第 12 章 系统库管理

12.1 概述

本章描述了 MapGIS7.0 系统库管理模块设计函数。主要包括以下内容：系统库中的数据存储和相关的管理操作。

12.2 主要模块及接口说明

1. MapFont 模块
   字体接口
   主要方法参见开发手册

2. MapFontLib 模块
   字体库接口
   提供了两类功能函数：
   （**1**）基本库打开关闭函数；
   （**2**）库操作辅助功能。
   主要方法参见开发手册.

3. CMapFontLib 模块
   字体库对象
   提供了两类功能函数：
   （**1**）字体号与字体对象间的相互获取；
   （**2**）打开和关闭字库、Vct 字库、TTF 字库；
   （**3**）创建和销毁所有字体的 LOGFONT；
   （**4**）得到字库类型标志。
   主要方法参见开发手册

4. CMapFontFile 模块
   本模块主要提供了 TTF 字库的文件读取和写入的功能函数
   主要方法参见开发手册

5. MapSymbol 模块
   符号对象提供符号的基本操作函数。主要包括：
   （**1**）符号头信息的获取与设置；
   （**2**）符号的修改（图元的添加、删除、更新，符号定位点的修改）、符号图元信息的获取；
   （**3**）符号的显示；
   （**4**）符号的导入、导出。
   主要方法参见开发手册

6. MapSymbolLib 模块

符号库对象提供的功能主要包括：

（1）符号库的创建、打开、保存；

（2）符号库信息的提取：符号库名、符号库描述、符号分类数、符号数等；

（3）符号库中符号的添加、删除、修改等；

（4）导入导出函数；

（5）分类函数。

主要方法参见开发手册.

7. ConvertSymbol 模块

主要提供符号库的转换功能.

主要方法参见开发手册.

8. GetNewSym 模块

本模块提供 6.x－7.0 的符号转换函数

主要方法参见开发手册

9. CMapSymbol 模块

符号库

主要内容

（1）点、线、区的基本结构；

（2）符号信息和图元的相关操作；

（3）点状符号的相关操作；

（4）画刷的建立；

（5）线状符号的相关操作；

（6）删格符号的相关操作；

主要方法参见开发手册.

10. MapColor 模块

颜色库

11. MapColorLib 模块

提供了四类功能函数：

（1）使用颜色库函数；

（2）颜色空间转换函数；

（3）打开关闭颜色库函数；

（4）颜色库修改维护函数。

主要方法参见开发手册.

12. MapColorExt 模块

颜色转换

主要方法参见开发手册.

13. SymLibUI_out 模块

提供了编辑系统库的输出接口。

主要方法参见开发手册.

12.3 应用实例

一. 基本的系统库操作方法

(1) 功能说明

打开符号库、颜色库、字体库，从中提取符号、颜色、字体进行图形的绘制.

(2) 基本思路

MapSymbolLib、MapColorLib、MapFontLib 提供了打开符号库、颜色库、字体库的 操 作方法，用户可直接使用这些模块下对应的方法打开系统库.

(3) 部分代码

```
//打开符号库
void CMyView::OnOpenGisLib()
{

MapSymbolLib    *pMapsymboilib=NULL;
pMapsymboilib=GetSymbolLib();                //获取符号库
if (pMapsymboilib)
{
    long m_totalNum;
    long m_classNum;
    CString    str;
    m_totalNum=pMapsymboilib->GetSymbolNum(); //取符号数
    pMapsymboilib->GetClassNum();
    m_classNum=pMapsymboilib->GetClassNum(); //取分类数
}
}

//打开字体库
void CMyView::OnOpenFontLib()
{

MapFontLib    *pMapfontlib=NULL;
pMapfontlib=GetMapFontLib();                //获取字体库
if (pMapfontlib)
{
    char *m_fontName="";
    CString    str;
    pMapfontlib->GetFontName(0,&m_fontName);    //取第一号字体的名称
    str.Format("字体库第一号为:%s",m_fontName);
```

```
        MessageBox(str);
    }
    }

    //打开颜色库
    void CMyView::OnOpenColorLib()
    {
    MapColorLib  *pMapcolorlib=NULL;
    pMapcolorlib=GetMapColorLib();                  //获取颜色库
    if (pMapcolorlib)
    {
        CString   str;
        long num=pMapcolorlib->GetColorNum();   //取颜色总数
        str.Format("颜色库总数为:%d\n",num);
        MessageBox(str);
    }
    }
```

二. 添加、删除符号的操作方法

(1) 功能说明

完成在当前符号库中添加一个新的符号,以及删除当前符号库中已有的符号.

(2) 基本思路

添加一个新的符号需要记录符号的坐标信息以及图形的参数信息,这里所说的坐标 信息指的是在符号编辑框中的坐标位置,一个符号可以由若干部分组成,组成一个符号 的基本图形要素为一个 SYMBOLITEM 对象.

(3) 部分代码

```
    //在当前符号库中添加一个新的符号
    void CMyView::OnAppenfASymbol()
    {
    MapSymbolLib      *pMapsymbollib=NULL;
    CMapSymbol        *pMapsymbol;
    SYMBOLHEAD            m_symbohead;
    SYMBOLITEM            pItem;
    BYTE                  *pData=NULL;
    S_DOT                 m_pos[2];
    long              m_symboNo=0;
    long                  sflg=0;
    pMapsymbollib=GetSymbolLib();               //获取符号库
    if(pMapsymbollib)
    {
        pMapsymbol=new CMapSymbol(NULL);
        m_symboNo=pMapsymbollib->GetBaseNum();          //获取当前符号库中的基数
        long rtl=pMapsymbol->GetSymbolHead(&m_symbohead); //取符号头对象
```

```
    if (rtl)
    {
        //定义符号头信息,包括符号的类型、名称、分类码以及定位点等
        m_symbohead.sSymbolType = SYMBOL_PNT|SYMBOL_TYPE_VECTOR;
        lstrcpy(m_symbohead.chSymbolName,"测试符号");
        m_symbohead.sClassCode =0;
        m_symbohead.sOrgX=5000,m_symbohead.sOrgY=5000;
        //将新添加的符号添加在符号库的最后一个位置
        m_symbohead.lSymbolNo=m_symboNo+1;
        sflg=pMapsymbol->SetSymbolHead(&m_symbohead);
        if (sflg)
        {
            //指定位置信息
            m_pos[0].x=5000,m_pos[0].y=5000;
            m_pos[1].x=8000,m_pos[1].y=5000;
            pData = (BYTE *)new S_DOT[2];
            memcpy(pData,(BYTE *)m_pos,sizeof(S_DOT)*2);

            //指定 pItem 的图形参数
            pItem.bItemType =6;             //类型为圆
            pItem.sInterPenWid=100;     //内部笔宽
            pItem.sDataLen=2;               //数据长度
            pItem.lInterColor=6;    //内部颜色
            pItem.bOuterColor = 6; //外部颜色
            pItem.bOuterPen = 6;//外部笔宽

            if (pMapsymbol)
            {
                //添加组成符号的基本要素项,如果组成该符号的项有多个,如有圆、
                //折线等,则需要添加多个 pItem
                pMapsymbol->AppendItem(&pItem,pData);
                long rtn=pMapsymbollib->WriteSymbol(pMapsymbol);
                if (rtn)
                {
                    MessageBox("添加符号成功!!!");
                }
                delete      pMapsymbol;
            }
            delete[] pData;
        }
    }
}
}
```

```
//删除当前符号库中的一个符号
void CMyView::OnDeletASymbol()
{
MapSymbolLib      *pMapsymbollib=NULL;
MapSymbol         *pMapsymbol=NULL;

pMapsymbollib=GetSymbolLib();
if (pMapsymbollib)
{
    long rtl=pMapsymbollib->DeleteSymbol(0);        //给定要删除的符号的编号
    if (rtl)
    {
        MessageBox("删除符号成功!!!");
    }
}
}
```

第 13 章 图形编辑

13.1 概述

13.1.1 模块命名规则

（1） 编辑工具由 EditTool 前缀开始；
（2） 交互工具由 IATool 前缀开始。

13.1.2 定义缩写词

（1） 弧段：几何空间中的坐标点序列，包括折线、圆、椭圆、弧、矩形、样条、贝齐尔等；
（2） 点 ：几何空间中单一坐标点；点分为“一般点”、“实结点”、“虚结点”；
（3） 几何实体 ：地理对象的外观特征或可视化形状。（地理对象可以用三种几何实体表示在地图上： 点、线、多边形）；
（4） 空间实体 ：组成几何实体的元素，存储坐标点。空间实体分为弧段和点两种类型；
（5） 要素：
现实世界中现象的抽象；
真实世界中的地理对象在地图上的表示；
要素具有几何和属性；
地理数据库中有几何类型字段的对象；

13.1.3 结构图

编辑模块由编辑控制、通用编辑、拓扑编辑和注记编辑四个部分组成，其结构关系如下图：

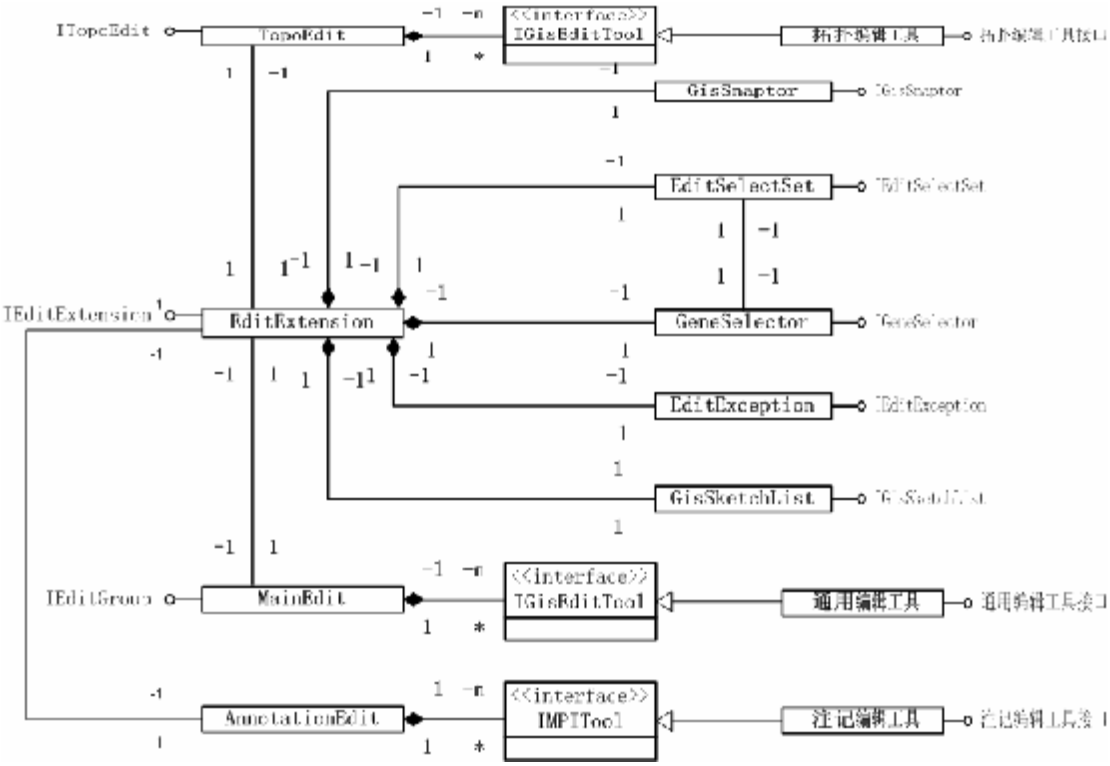


图 6.1 编辑模块结构图

13.2 主要模块及接口说明

1. 编辑控制

IobjectSnapped	捕获对象
IgisSnaptor	捕获工具
IgisSketchList	编辑草图
IeditSelectSet	编辑选择集
IeditExtension	编辑扩展
IGeneSelectorView	通用选择视图
IeditException	编辑异常提示
IeditExceptionBar	编辑提示栏
IEmbedLBView	图例板视图插件

2. 通用编辑

IeditGroup	编辑组
IGisEditTool	编辑工具
IGisIATool	交互工具

3. 拓扑编辑

ItopoEditGroup　　拓扑编辑工具组

4. 注记编辑

IAnnEditor　　注记编辑器

5. EditDev 模块

EditDev 提供了通用编辑、拓扑编辑以及注记编辑的常用功能，用户可直接使用该模块下提供的方法进行如：输入线、拓扑构建等操作.

13.3 应用实例

基本图形编辑

EditDev 模块下提供了大部分的图形编辑功能,用户可直接调用,不需要给定入口参数,具体实例这里不做演示.使用时,需继承 EditDevView 视图类.

详细方法可参见开发手册或查询 EditDev.h 头文件下的内容.

```
//MyView.h
class CMyView : public CEditDevView
{
    protected:
    CMyView();                  // protected constructor used by dynamic creation
    DECLARE_DYNCREATE(CMyView)

    // Attributes
    public:
    ……
    ……
}

void CMyView::OnDraw(CDC* pDC)
{
    CDocument* pDoc = GetDocument();
    // TODO: add draw code here
    CEditDevView::OnDraw(pDC);
}
```

用户从 CeditDevView 类派生一个视图类 CmyView 类,重载 CEditDevView::OnDraw(pDC),直接调用 CeditDevView 类中的函数就可以实现基本的图形编辑功能!

```
//增加节点
void CMyView::OnAddNode()
{
	AddNode();
}
//靠近线
void CMyView::OnAnearOtherLine()
{
	AnearOtherLine();
}
//自动接边
void CMyView::OnAutoJoinEdge()
{
	AutoJoinEdge();
}
……
```

第 14 章 空间分析

14.1 概述

14.1.1 概述及结构图

空间分析是 GIS 中最重要的内容之一,体系了 GIS 的本质

空间分析的主要内容:

- 拓扑分析:包括空间图形数据的拓扑运算,即旋转变换 、比例尺变换、二维及三维显示和几何元素计算等。
- 属性分析：包括数据检索、逻辑与数学运算、重分类和统计分析等。
- 拓扑与属性的联合分析:包括与拓扑相关的数据检索、叠置处理、区域分析、领域分析、网络分析、形状探测、瘦化处理和空间内插等。

本文档描述 MapGIS7.0 的空间分析模块设计。结构图如下：

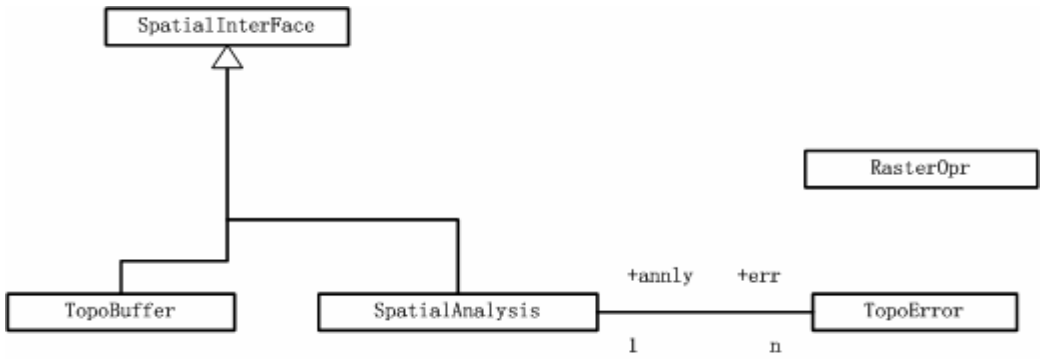


图 14.1 空间分析模块

14.1.2 空间分析主要功能说明

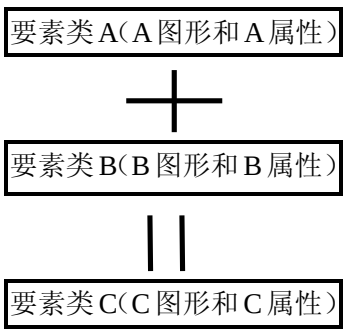
- 叠加分析：包括区对区叠加分析、线对区叠加分析、点对区叠加分析、区对点叠加分析和点对线叠加分析。
- 缓冲区分析：包括点 BUFFER 分析、线 BUFFER 分析、区 BUFFER 分析。

设有原要素类 A 和 B，叠加结果为要素类 C，其中：

A 要素类属性为： 缺省字段， F1

B 要素类属性为： 缺省字段， F2

叠加过程如下图所示：



其中 C 要素类的图形类型和 A 要素类相同，而属性则是 A 要素类与 B 要素类属性连接的结果。

14.1.3 叠加分析

（a） 区对区叠加分析

包括合并、相交、相减、判别四种方式。叠加结果用限影表示，叠加结果的属性为：

标志码、面积、周长，F1、区号、F2

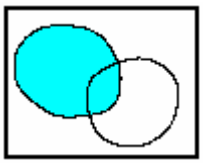
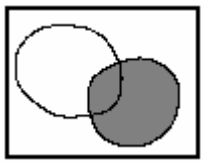
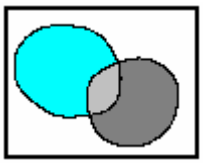
其中区号为第二个要素类的区号（标志码）。

合并：属于 A 或属于 B 的区域。

标志码	面积	周长	F1
1	320.5	61.2	a

标志码	面积	周长	F2
1	280.7	50.1	b

标志码	面积	周长	F1	区号	F2
1	198.2	51.3	a		
2	122.3	42.1	a	1	b
3	158.4	53.4		1	b

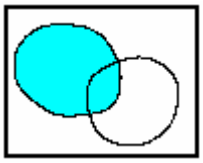
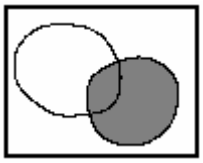
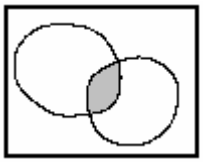




相交：属于 A 且属于 B 的区域。

标志码	面积	周长	F1
1	320.5	61.2	a

标志码	面积	周长	F2
1	280.7	50.1	b

标志码	面积	周长	F1	区号	F2
1	122.3	42.1	a	1	b

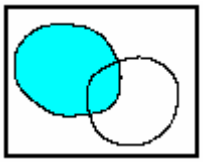
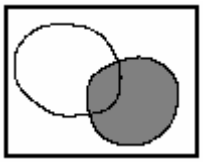
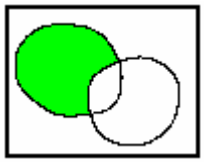




相减：属于 A 不属于 B 的区域。

标志码	面积	周长	F1
1	320.5	61.2	a

标志码	面积	周长	F2
1	280.7	50.1	b

标志码	面积	周长	F1	区号	F2
1	198.2	51.3	a		

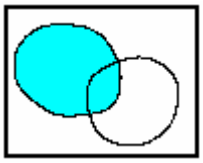
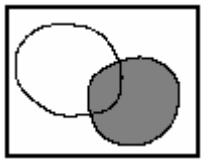
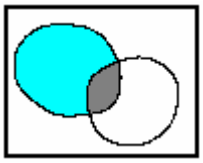




判别：属于 A 的区域。

标志码	面积	周长	F1
1	320.5	61.2	a

标志码	面积	周长	F2
1	280.7	50.1	b

标志码	面积	周长	F1	区号	F2
1	198.2	51.3	a		
2	122.3	42.1	a	1	b

（b） 线对区叠加分析

包括相交、判别、相减三两种方式，叠加结果要素类仍然是线要素类，叠加结果的属性为：

标志码、线长度、F1、区号、F2

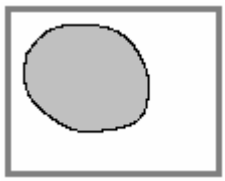
线图元用标号表示。

相交：穿过区域的线段部分

标志码	线长度	F1
1	167.0	a

标志码	面积	周长	F2
1	320.5	61.2	b

标志码	线长度	F1	区号	F2
1	80.8	a	1	b

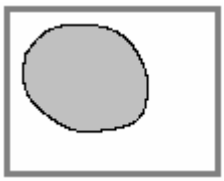




判别：所有线图元

标志码	线长度	F1
1	167.0	a

标志码	面积	周长	F2
1	320.5	61.2	b

标志码	线长度	F1	区号	F2
1	32.2	a		
2	80.8	a	1	b
3	44.0	a		

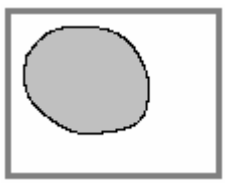




相减：区域以外的线段

标志码	线长度	F1
1	167.0	a

标志码	面积	周长	F2
1	320.5	61.2	b

标志码	线长度	F1
1	32.2	a
3	44.0	a


（c） 点对区叠加分析

包括相交、判别、相减三种方式，叠加结果要素类仍然是点要素类，结果属性为：

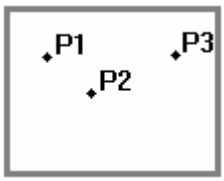
标志码、F1、区号、F2

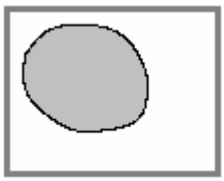
相交：落在区域上的点。

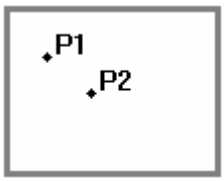
标志码	F1
1	p1
2	p2
3	p3

标志码	面积	周长	F2
1	320.5	61.2	b

标志码	F1	区号	F2
1	p1	1	b
2	p2	1	b





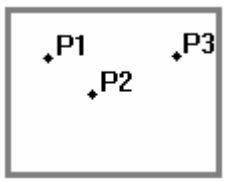


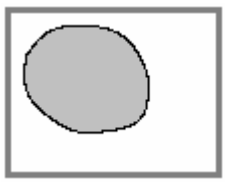
判别：所有点图元

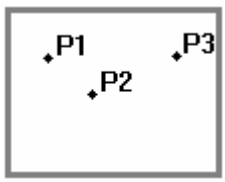
标志码	F1
1	p1
2	p2
3	p3

标志码	面积	周长	F2
1	320.5	61.2	b

标志码	F1	区号	F2
1	p1	1	b
2	p2	1	b
3	p3		





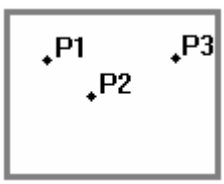


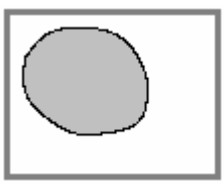
相减：区域以外的点图元

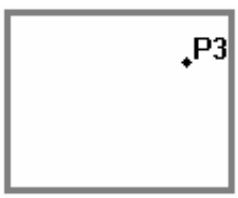
标志码	F1
1	p1
2	p2
3	p3

标志码	面积	周长	F2
1	320.5	61.2	b

标志码	F1
1	p3







（d） 区对点叠加分析

分为相减和相交两种方式。叠加结果为区要素类，结果属性和原始区要素类相同。

相交：保留那些有点落在上面的区域。

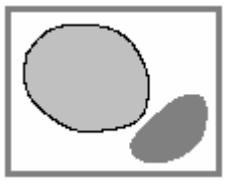
标志码	面积	周长	F1
1	320.5	61.2	b

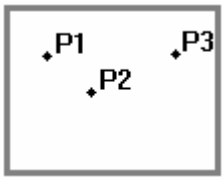
标志码	F2
1	p1

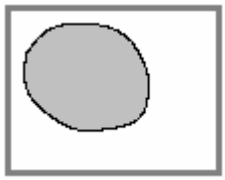
标志码	面积	周长	F1
1	320.5	61.2	b

2	150.7	45.1	c

2	p2
3	p3





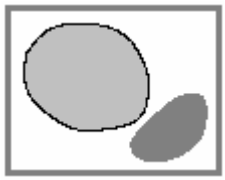


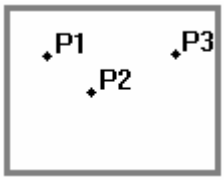
相减：保留那些没有点落在上面的区域。

标志码	面积	周长	F1
1	320.5	61.2	b
2	150.7	45.1	c

标志码	F2
1	p1
2	p2
3	p3

标志码	面积	周长	F1
2	150.7	45.1	c





（e） 点对线叠加分析

叠加结果为点要素类，该方法保留所有点，找到距离某点最近的线并计算出点线之间的距离，然后将线号和点线距离记录到属性中。

点线距离定义如下：

对任意点 D 和曲线 L，假设 L 由 n 个离散点 d[0]、d[1]、d[2]…d[n]构成，则 D 到 d[0]、d[1]…d[n]的距离分别为 S0、S1…Sn，D 到直线段（d[0]，d[1]）、（d[1]，d[2]）…（d[n-1]，d[n]）的法线距离分别为：

$$li=\begin{cases} D\text{到}(d[i-1],d[i])\text{的法线距离} & \text{若法线距离存在} \\ \infty & \text{若法线距离不存在} \end{cases}$$

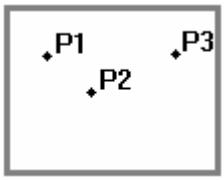
那么点 D 到曲线 L 的距离 S=min（S0，S1，S2……Sn，l1，l2…ln）。

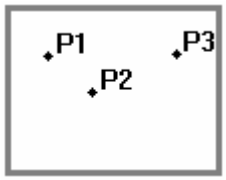
叠加结果的属性为： 标志码、F1、线号、点线距离、F2

标志码	F1
1	p1
2	p2
3	p3

标志码	线长度	F2
1	23.6	l1
2	67.0	l2

标志码	F1	线号	点线距离	F2
1	p1	1	5.2	l1
2	p2	1	4.4	l1
3	p3	2	3.8	l2



14.2 接口说明

SpatialInterFace	空间分析界面接口
TopoError	拓扑错误接口
TopoBuffer	buffer 分析接口
SpatialAnalysis	空间分析接口
RasterOpr	矢量光栅化处理接口

创建接口的 API 函数:

1、创建空间分析接口

```
long __SPC_ANLY70_EXPORT_CLASS CreateSpatialAnalysis( SpatialAnalysis** pSpatial );
```

2、创建 buffer 分析接口

```
long __SPC_ANLY70_EXPORT_CLASS CreateTopoBuffer( TopoBuffer** pTopoBuf );
```

3、创建拓扑错误接口

```
long __SPC_ANLY70_EXPORT_CLASS CreateTopoError( TopoError** pTopoErr );
```

4、创建矢量光栅化处理接口

```
long __SPC_ANLY70_EXPORT_CLASS CreateRasterOpr( RasterOpr** pRaster );
```

14.3 应用实例

空间分析

- 功能说明

实现要素与要素之间的叠加分析(本例以区对区的空间分析为例);实现要素类的缓 冲区分析;实现要素类的裁剪分析以及要素类的空间几何查询.

- 基本思路

空间分析接口 SpatialAnalysis 提供的要素类之间的叠加分析方法,用户可针对打开 的要素类进行要素类之间的相交,相减等分析.

buffer 分析接口 TopoBuffer 提供了要素的缓冲区分析,包括缓冲区分析的参数设置 (如 buffer 分析半径等参数),用户可根据实际需要进行单边,双边缓冲区分析.

空间几何查询分析是指通过判断要素类 A 与要素类 B 的空间关系(相等、相离、包 含、包含于等)比较得到 A 要素类中满足条件的要素，如 A 中有部分要素类包含于 B 中 的要素类,则得到的结果要素类就是 A 中满足条件的这部分要素!

- 部分代码

```
//区对区的叠加分析
```

```
//其中 opfun 表示叠加方式,如相交,相减等!
//ptFclsname 表示叠加分析得到的结果要素的名称
void CMyView::OnSpatialAnalysis(short opfun,char* ptFclsname)
{
SpatialAnalysis        *m_pSpati=NULL; //空间分析对象
CFeatureCls            *m_pfSrc0=NULL;
CFeatureCls            *m_pfSrc=NULL;
CFeatureCls            *m_pfDes=NULL;
long                   sflg=0;

//打开要素类 A
m_pfSrc0=OnOpenFcls();
if (!m_pfSrc0)
{
    delete m_pfSrc0,m_pfSrc0=NULL;
    return;
}
//打开要素类 B
m_pfSrc=OnOpenFcls();
if (!m_pfSrc)
{
    delete m_pfSrc,m_pfSrc=NULL;
    return;
}
//创建结果要素类
m_pfDes=new CFeatureCls();
m_pfDes->fcls_Create(m_ptGDB,ptFclsname,0,0,0,NULL,NULL,NULL,NULL,FCLS_REG
        _TYPE);
//创建空间分析对象
long rtl=CreateSpatialAnalysis(&m_pSpati);
if (rtl)
{
    //进行空间叠加分析
    sflg=m_pSpati->OverLay(m_pfSrc0,m_pfSrc,m_pfDes,opfun,NULL);
    if (sflg)
    {
        MessageBox("叠加成功!!!");
    }
```

```
    }
    //释放空间
    delete m_pSpati,m_pSpati=NULL;
    delete m_pfSrc0,m_pfSrc0=NULL;
    delete m_pfSrc,m_pfSrc=NULL;
    delete m_pfDes,m_pfDes=NULL;
}

    //缓冲区分析
    void CMyView::OnConBufferAnalysis()
    {
    TopoBuffer              *m_pBuffer=NULL;
    CFeatureCls             *m_pfSrc=NULL;
    CFeatureCls             *m_pfDes=NULL;
    char                ptFclsname[128];

    char    user[30]="";
    char    pwsd[64]="";
    long    flag=0;
    long    ptSelFlag;          //文件类型
    long    type;           //打开的文件类型
    long    xclsID;             //打开的类 ID

    m_ptGDB = new CGDataBase;
    ptSelFlag=XCLS_TYPE_FCLS;    //弹出对话框为选择要素类对话框
    //弹出选择要素类对话框
    if(ShowGDBShellDlg(path,&ptSelFlag,1,0,"选择类")<=0)
        return  ;
    //获取数据源信息
    GetLoginInfo(path.svcName,user,30,pwsd,64);
    //连接数据源
    if(m_GDBSvr.Connect(path.svcName,user,pwsd)<=0)
        return  ;
    //打开数据库
    if(m_ptGDB->Open(&m_GDBSvr,path.gdbName)<=0)
        return  ;
    type    = path.xclsInf[0].xclsType;
    xclsID = path.xclsInf[0].xclsID;
```

```
//判断打开的是否是要素类
if(type !=XCLS_TYPE_FCLS)
    return;
m_pfSrc=new CFeatureCls;
flag=m_pfSrc->fcls_Open(m_ptGDB,xclsID,1);  //打开要素类

if (!flag)
{
    delete m_pfSrc,m_pfSrc=NULL;
    return;
}
//定义缓冲区分析结果要素类
lstrcpy(ptFclsname,"buffer 结果要素") ;
m_pfDes=new CFeatureCls();
m_pfDes->fcls_Create(m_ptGDB,ptFclsname,0,0,0,NULL,NULL,NULL,NULL,FCLS_REG
                                        _TYPE);
//创建缓冲区分析对象
long rtl=CreateTopoBuffer(&m_pBuffer);
if (rtl)
{
    //设置缓冲区分析的参数
    m_pBuffer->SetSideType(FULL_SIDE);
    m_pBuffer->SetKnobFlg(CIRCLE_ANGLE);
    m_pBuffer->SetBufRadius(1,1);
    m_pBuffer->SetBufMethod(RASTER_BUF);
    //创建缓冲区分析进度条
    m_pBuffer->CreateProgressBar("缓冲区分析");
    m_pBuffer->EnableShowProgress(true);
    //固定半径缓冲区分析
    long sflg=m_pBuffer->ConstantBufRadius(m_pfSrc,m_pfDes,NULL);
    //结束进度条
    m_pBuffer->EnableShowProgress(false);
    if (sflg)
    {
        MessageBox("BUFFER 分析完成!!!");
    }
    //释放空间
    delete m_pBuffer,m_pBuffer=NULL;
```

```
        delete m_pfSrc,m_pfSrc=NULL;
        delete m_pfDes,m_pfDes=NULL;

    }

    }

    //空间裁剪分析
    void CMyView::OnClip()
    {
    SpatialAnalysis         *m_pSpati=NULL;
    CFeatureCls             *m_pfSrc0=NULL;
    CFeatureCls             *m_pfSrc=NULL;
    CFeatureCls             *m_pfDes=NULL;
    long                sflg=0;
    char                ptFclsname[128];
    char                m_type1;
    //打开被裁剪要素类
    m_pfSrc0=OpenFcls();
    if (!m_pfSrc0)
    {
        delete m_pfSrc0,m_pfSrc0=NULL;
        return;
    }
    //取得当前打开要素类类型,结果要素类应与该要类型一致!
    m_pfSrc0->fcls_GetType(&m_type1);
    //打开裁剪框要素类
    m_pfSrc=OpenFcls();
    if (!m_pfSrc)
    {
        delete m_pfSrc,m_pfSrc=NULL;
        return;
    }
    char m_type;
    //判断裁剪框要素类是否为面要素类活着混合要素类,裁剪框要素类必须要为面要素类    // 活 着 混合要素类
    m_pfSrc->fcls_GetType(&m_type);
    if (m_type!=6 && m_type!=0)
```

```
{
    MessageBox("裁减框必须为面要素或者混合要素!");
    return;
}
//创建结果要素类
m_pfDes=new CFeatureCls();
lstrcpy(ptFclsname,"裁剪结果要素");
m_pfDes->fcls_Create(m_ptGDB,ptFclsname,0,0,0,NULL,NULL,NULL,NULL,m_type1);
//创建空间分析对象
long rtl=CreateSpatialAnalysis(&m_pSpati);
if (rtl)
{
    //进行空间裁剪分析
    sflg=m_pSpati->Clip(m_pfSrc0,m_pfSrc,m_pfDes,OVLY_INCLIP,NULL);
    if (sflg)
    {
        MessageBox("裁剪成功!!!");
    }
    //释放空间
    delete m_pSpati,m_pSpati=NULL;
    delete m_pfSrc0,m_pfSrc0=NULL;
    delete m_pfSrc,m_pfSrc=NULL;
    delete    m_pfDes,m_pfDes=NULL;

}
}

//空间几何查询,opfun 为空间查询方法,如:相交,相离,包含等
void CMyView::OnSpaGeoSql(short opfun)
{
SpatialAnalysis         *m_pSpati=NULL;
CFeatureCls             *m_pfSrc0=NULL;
CFeatureCls             *m_pfSrc=NULL;
CFeatureCls             *m_pfDes=NULL;
long                    sflg=0;
char                    ptFclsname[128];
char                    m_type;
//打开待查询要素类
```

```
m_pfSrc0=OpenFcls();
if (!m_pfSrc0)
{
	delete m_pfSrc0,m_pfSrc0=NULL;
	return;
}
//获取打开的要素类类型,结果要素类的类型应与该类型一致!
m_pfSrc0->fcls_GetType(&m_type);
//打开查询要素类
m_pfSrc=OpenFcls();
if (!m_pfSrc)
{
	delete m_pfSrc,m_pfSrc=NULL;
	return;
}
//创建结果要素类
m_pfDes=new CFeatureCls();
lstrcpy(ptFclsname,"查询结果要素");
m_pfDes->fcls_Create(m_ptGDB,ptFclsname,0,0,0,NULL,NULL,NULL,NULL,m_type);
//创建空间分析对象
long rtl=CreateSpatialAnalysis(&m_pSpati);
if (rtl)
{
	//进行空间几个查询
	sflg=m_pSpati->SpaGeoSql(m_pfSrc0,m_pfSrc,m_pfDes,opfun,0);
	if (sflg)
	{
		MessageBox("查询完成!!!");
	}
	//释放空间
	delete m_pSpati,m_pSpati=NULL;
	delete m_pfSrc0,m_pfSrc0=NULL;
	delete m_pfSrc,m_pfSrc=NULL;
	delete	m_pfDes,m_pfDes=NULL;
}

}
```