第6章

空间数据管理

6.1 SDE及空间数据

6.1.1 SDE介绍

ArcSDE是数据库系统中管理地理数据库的接口，通过该接口可以往关系数据中加入空间数据，提供地理要素的空间位置及形状等信息，是ArcGIS与关系数据库之间的GIS通道。它允许用户在多种数据管理系统中管理地理信息，并使所有的ArcGIS应用程序都能够使用这些数据。

ArcSDE是多用户ArcGIS系统的一个关键部件，它为DBMS提供了一个开放的接口，允许ArcGIS在多种数据库平台上管理地理信息，这些平台包括Oracle、Oracle with Spatial/Locator、Microsoft SQL Server、IBM DB2和Informix。通过ArcSDE，ArcGIS可以在DBMS中轻而易举地管理一个共享的、多用户的空间数据库。ArcSDE的具体功能如下。

（1）高性能的DBMS通道

ArcSDE是多种DBMS的通道。它本身并非一个关系数据库或数据存储模型。它是一个能在多种DBMS平台上提供高级的、高性能的GIS数据管理的接口。

（2）开放的DBMS支持

ArcSDE允许在多种DBMS中管理地理信息：Oracle、Oracle with Spatial or Locator、Microsoft SQL Server、Informix以及IBM DB2。

多用户ArcSDE为用户提供了大型空间数据库支持，并且支持多用户编辑。

（3）连续、可伸缩的数据库

ArcSDE可以支持海量的空间数据库和任意数量的用户，直至DBMS的上限。GIS工作流和长事务处理GIS中的数据管理工作流，例如多用户编辑、历史数据管理、check-out/check-in，以及松散耦合的数据复制等，都依赖于长事务处理和版本管理。ArcSDE为DBMS提供了这种支持。

（4）丰富的地理信息数据模型

ArcSDE保证了存储于DBMS中的矢量和栅格几何数据的高度完整性。这些数据包括：矢量

和栅格几何图形，支持 x、y、z 和 x、y、z、m 的坐标，以及曲线、立体、多行栅格、拓扑、网络、注记、元数据、空间处理模型、地图、图层，等等。

（5）灵活的配置

ArcSDE 通道可以让用户在客户端应用程序内，或跨网络、跨计算机地对应用服务器进行多种多层结构的方案配置。ArcSDE 支持 Windows、UNIX、Linux 等多种操作系统。

ArcSDE 能够让同样的功能在所有的 DBMS 上得到实现。尽管所有的关系数据库都支持 SQL，并能使用相似的方法处理简单的 SQL，但是不同数据库的数据库服务器在实现细节上却有着显著的差别，这些差别包括性能和索引、支持的数据类型、集成管理工具和复杂查询的执行，还包括在 DBMS 中对空间数据类型的支持。

标准的 SQL 并不支持空间数据。ISO SQL/MM Spatial 和 OGC 的简单要素 SQL 规范扩展了 SQL，并且为不同的矢量数据定义了标准的 SQL 支持。 DB2 和 Informix 直接支持这些 SQL 类型。Oracle 使用的是自己的标准，其空间类型系统是核心数据库系统上的一个独立的可选扩展。而微软的 SQL Server 不提供空间类型的支持。ArcSDE 不但灵活地支持了每个 DBMS 提供的独特功能，而且能为底层 DBMS 提供它们所不具备的功能的支持。

ArcSDE 支持高性能的空间数据的管理，它支持的数据库如下。

① Oracle（带压缩二进制）

② Oracle（带 Locator 和 Spatial）

③ 微软 SQL Server（带压缩二进制）

④ IBM DB2（带 Spatial Extender）

⑤ IBM Informix（带 Spatial Database）

ArcSDE 是为了解决 DBMS 的多样性和复杂性而存在的。ArcSDE 的体系结构给用户提供了巨大的灵活性，它允许用户自由地选择 DBMS 来存储空间数据。ArcSDE 分摊了 DBMS 和 GIS 之间对管理空间数据的职责，对空间数据的管理职责是由 GIS 软件和常规 DBMS 软件所共同承担的。某些空间数据的管理功能，例如磁盘存储、属性类型定义、查询处理，以及多用户事务处理等，是由 DBMS 来完成的。当然一些 DBMS 引擎本身也扩展了对空间数据的支持，它们具备索引和搜索的功能。GIS 软件负责为特定的 DBMS 提供各种地理数据的表达。从实际效用上看，DBMS 是被作为一个空间数据的实现机制而存在的。

ArcSDE 是基于多层体系结构的（应用和存储）。数据的存储和提取由存储层（DBMS）实现，而高端的数据整合和数据处理功能则由应用层（ArcGIS）提供。ArcSDE 支持 ArcGIS 应用层，并提供 DBMS 通道技术，使得空间数据可以存储于多种 DBMS 中。ArcSDE 用于高效地存储、索引、访问和维护 DBMS 中的矢量、栅格、元数据及其他空间数据。ArcSDE 同时能保证所有的 GIS 功能可用，而无需考虑底层的 DBMS。使用 ArcSDE，用户在 DBMS 中即可有效地管理自己的地理数据资源。ArcSDE 使用 DBMS 支持的数据类型，以表格的形式管理底层存储的空间数据，并可使用 SQL 在 DBMS 中访问这些数据。ArcSDE 同时也提供了开放的客户端开发接口（C API 和 Java API），通过这些接口，用户定制的应用程序也可以完全访问底层的空间数据表。

6.1.2 空间数据

空间数据（Spatial Data）是指用来表示空间实体的位置、形状、大小及其分布特征诸多方面信息的数据。它可以用来描述来自现实世界的目标，它具有定位、定性、时间和空间关系等特性。

定位是指在一个已知的坐标系里，空间目标都具有唯一的空间位置；定性是指有关空间目标的自然属性，它伴随着目标的地理位置；时间是指空间目标是随时间的变化而变化；空间关系通常又称拓扑关系。

空间数据适用于描述所有呈二维、三维甚至多维分布的关于区域的现象。空间数据不仅能够表示实体本身的空间位置及形态信息，而且还有表示实体属性和空间关系(如拓扑关系)的信息。在空间数据中不可再分最小单元现象称为空间实体，空间实体是对存在于这个自然世界中地理实体的抽象，主要包括点、线、面以及实体等基本类型。如把一根电线杆抽象成为一个点，该点可以包含电线杆所处的位置信息、电线杆的高度信息和其他一些相关信息；可以把一条道路抽象为一条线，该线可以包含这条道路的长度、宽度、起点、终点，以及道路等级等相关信息；可以把一个湖泊抽象为一个面，该面可以包含湖泊的周长、面积和湖水的质量信息等。在空间对象建立后，还可以进一步定义其相互之间的关系，这种相互关系被称为“空间关系”，又称为“拓扑关系”，如可以定义点－线关系、线－线关系、点－面关系等。因此可以说空间数据是一种可以用点、线、面以及实体等基本空间数据结构来表示人们赖以生存的自然世界的数据。

归纳起来它具有以下 5 个基本特征。

（1）空间特征

每个空间对象都具有空间坐标，即空间对象隐含了空间分布特征。这意味着在空间数据组织方面，要考虑它的空间分布特征。除了通用性数据库管理系统或文件系统关键字的索引和辅关键字索引以外，一般需要建立空间索引。

（2）非结构化特征

在当前通用的关系数据库管理系统中，数据记录一般是结构化的。即它满足关系数据模型的第一范式要求，每一条记录是定长的，数据项表达的只能是原子数据，不允许嵌套记录。而空间数据则不能满足这种结构化要求。若将一条记录表达一个空间对象，它的数据项可能是变长的。例如，1 条弧段的坐标，其长度是不可限定的，它可能是 2 对坐标，也可能是 10 万对坐标。其二，1 个对象可能包含另外的 1 个或多个对象，例如 1 个多边形，它可能含有多条弧段。若 1 条记录表示 1 条弧段，在这种情况下，1 条多边形的记录就可能嵌套多条弧段的记录，所以它不满足关系数据模型的范式要求，这也就是为什么空间图形数据难以直接采用通用的关系数据管理系统的主要原因。

（3）空间关系特征

空间数据除了前面所说的空间坐标隐含了空间分布关系外，空间数据中记录的拓扑信息则表达了多种空间关系。这种拓扑数据结构一方面方便了空间数据的查询和空间分析，另一方面也给空间数据的一致性和完整性维护增加了复杂性。特别是有些几何对象，没有直接记录空间坐标的信息。如拓扑的面状目标，仅记录了组成它的弧段的标识，因而进行查找、显示和分析操作时都要操纵和检索多个数据文件方能得以实现。

（4）分类编码特征

一般而言，每一个空间对象都有一个分类编码，而这种分类编码往往属于国家标准、行业标准或地区标准，每一种地物的类型在某个 GIS 中的属性项个数是相同的。因而在许多情况下，一种地物类型对应于一个属性数据表文件。当然，如果几种地物类型的属性项相同，多种地物类型也可以共用一个属性数据表文件。

（5）海量数据特征

空间数据量是巨大的，通常称为海量数据。之所以称为海量数据，是指它的数据量比一般的通用数据库要大得多。一个城市地理信息系统的数据量可能达几十 GB，如果考虑影像数据的存

储，可能达几百个 GB。这样的数据量在城市管理的其他数据库中是很少见的。正因为空间数据量大，所以需要在二维空间上划分块或者图幅，在垂直方向上划分层来进行组织。

空间数据库系统是一个存储空间和非空间数据的数据库系统，其数据模型和查询语言能支持空间数据类型和空间索引，并且提供进行空间查询和其他空间分析的方法。空间数据库中存储的信息包含两部分：一部分是和空间有关的信息，如点、线、矩形、多边形、多面体等占有空间的对象，称之为空间数据；另一部分是与空间数据有关的各个属性，如点所表示的城市的面积、人口等，称之为非空间数据（属性数据）。

6.2 空间数据库及组织

空间数据库系统是描述、存储和处理空间数据及其属性数据的数据库系统。空间数据库是随着地理信息系统的开发和应用而发展起来的数据库新技术。目前，空间数据库系统尚不是独立存在的系统，它与应用紧密结合，大多数是作为地理信息系统的基础和核心的形式出现。由于空间数据的复杂性和特殊性，一般的商用数据库管理系统难以满足要求。因而，围绕空间数据管理方法，出现了几种不同的组织方式。当前空间数据库的组织方式主要有两种：混合型空间数据库和集成型空间数据库，它们的区别在于是否对空间数据和属性数据进行一体化组织。

6.2.1 混合型空间数据库

目前大多数的地理信息开发平台就是采用混合结构型结构，即非空间数据存储在关系数据库中，空间数据存放在系统文件中。在空间数据库技术领域，比较有名的是 MapInfo 公司和 ESRI 公司，其中的许多产品都是采用混合型的空间数据库，如商用系统 MapInfo 系列，ESRI 的 ArcView、ArcInfo 系列等。这些产品分别使用不同的模块存储空间数据和属性数据。其空间数据在垂直方向上是分层进行组织的，空间数据在水平方向上是按图幅组织的。不同的是：MapInfo 公司设计的产品中每一个图层由 5 个文件组成，而 ESRI 公司设计的产品中每一个图层只由 3 个文件组成。

这种混合管理的模式（即用文件系统管理几何图形数据，用商用关系数据库管理系统管理属性数据），它们之间是通过目标标识或者内部连接码进行连接。在这种管理模式中，几何图形数据与属性数据除了它们的 ObjectID 作为连接关键字段以外，两者几乎是独立地组织、管理与检索。就几何图形而言，由于 GIS 系统采用高级语言编程，可以直接操纵数据文件，所以图形用户界面与图形文件处理是一体的，中间没有裂缝。但对属性数据来说，则因系统和历史发展而异。早期系统中的属性数据必须通过关系数据库管理系统，图形处理的用户界面和属性的用户界面是分开的，它们只是通过一个内部码连接，导致这种连接方式的主要原因是早期的数据库管理系统不提供编程的高级语言，如 Fortran 或 C 的接口，而只能采用数据库操纵语言。这样通常要同时启动两个系统（GIS 图形系统和关系数据库管理系统），甚至两个系统来回切换，使用起来很不方便。

最近几年，随着数据库技术的发展，越来越多的数据库管理系统提供有高级编程语言 C 和 Fortran 等接口，使得地理信息系统可以在 C 语言的环境下直接操纵属性数据，并通过 C 语言的对话框和列表框显示属性数据，或通过对话框输入 SQL 语句，并将该语句通过 C 语言与数据库的接口查询属性数据库，在 GIS 的用户界面下显示查询结果。对于这种工作模式，并不需要启动一个完整的数据库管理系统，用户甚至不知道何时调用了关系数据库管理系统，图形数据和属

性数据的查询与维护完全在一个界面之下。在 ODBC（开放性数据库连接协议）推出之前，每个数据库厂商提供一套自己的与高级语言的接口程序，这样 GIS 软件商就要针对每个数据库开发一套与 GIS 的接口程序，所以往往在数据库的使用上会受到限制。

在推出了 ODBC 之后，GIS 软件商只要开发 GIS 与 ODBC 的接口软件，就可以将属性数据与任何一个支持 ODBC 协议的关系数据库管理系统连接。无论是通过 C，还是通过 ODBC 与关系数据库连接，GIS 用户都是在一个界面下处理图形和属性数据，它比前面分开的界面要方便得多。但这种采用文件与关系数据库管理系统的混合管理模式，还不能说建立了真正意义上的空间数据库管理系统，因为文件管理系统的功能较弱，特别是在数据的安全性、一致性、完整性、并发控制以及数据损坏后的恢复等方面缺少基本的功能。多用户操作的并发控制比起商用数据库管理系统来要逊色得多，因而 GIS 软件商一直在寻找采用商用数据库管理系统来同时管理图形和属性数据。

文件系统结构简单，在数据存取过程中几乎没有额外开销，并且可以按照用户的需求任意定制数据存储格式或存储复杂数据结构。但其缺点也很明显，即数据冗余度大、难以共享数据、容易造成数据的不一致性、程序与数据缺乏独立性、系统不易扩充。

这些产品虽然在功能上达到了空间信息处理的要求，但由于没有从 DBMS 的核心部分，即数据结构部分来考虑空间信息的特殊性，所以处理的效率受到了很大的限制。

6.2.2 集成型空间数据库

近来国外提出的一种集成型结构，是将所有的数据都存储于一个数据库中。由于其采用的数据库原型不同，所以集成性空间数据库的类型也大不相同。

（1）全关系型数据库模型

全关系型空间数据库管理系统是指图形和属性数据都用现有的关系数据库管理系统管理。关系数据库管理系统的软件厂商不作任何扩展，由 GIS 软件商在此基础上进行开发，使之不仅能管理结构化的属性数据，而且能管理非结构化的图形数据。其设计思想是把构成空间对象的每个基本元素（点、线段）对作为一行记录存入表中，通过关系模型来维护其空间属性的内部结构以及空间对象之间的拓扑关系。

用关系数据库管理系统管理图形数据的模式有两种，一种是基于关系模型的方式，图形数据按照关系数据模型组织。这种组织方式由于涉及一系列关系连接运算，相当费时，因此在处理空间目标方面效率不高。另一种方式是将图形数据的变长部分处理成 Binary 二进制块 Block 字段。目前大部分关系数据库管理系统都提供了二进制块的字段域，以便管理多媒体数据或可变长文本字符。GIS 利用这种功能，通常把图形的坐标数据，当做一个二进制块，交由关系数据库管理系统存储和管理。这种存储方式虽然省去了前面所述的大量关系连接操作，但是二进制块的读写效率要比定长的属性字段慢得多，特别是涉及对象的嵌套时，速度则更慢。

关系型数据库（RDB）有强大的关系理论支持，无疑是数据库产品中最成熟的。由于它是基于几种简单的基本类型和简单的二维表，因此它能完成非常复杂的关系操作，并能得到几乎任何期望的信息组合。它的优点也是很明显的，首先，物理数据存储与逻辑数据库结构的独立性；其次，多样、简易的数据访问能力，从而提供了高效数据访问的可能性；第三，相当灵活的数据库设计；第四，数据存储和冗余的减少。

采用关系数据库来管理地理信息是 GIS 与数据库结合的初步尝试，但其中存在两个难题：

首先，难以在关系数据库中存储空间数据，为了存储和表示结构复杂多变的空间数据，需要设计复杂的 E－R 模型，并在数据库中存放大量的表，这样会增大系统的复杂性，而且还无法利用数据库提供的索引机制；其次，难以保持地理信息的数据一致性，由于地理信息的空间数据和属性数据通常不能存放在一张表里，因此这样会割裂逻辑上为一个整体的地理信息，而且拓扑关系也难以维护。

由于存在上述难题，虽然关系数据库在与 GIS 的结合中做出了有益的尝试，但最终还是没能成为主流。为此，随着面向对象型数据库（OODB）的出现与发展，人们提出了面向对象型数据库与 GIS 相结合的方案。

（2）面向对象数据库

面向对象模型最适于空间数据的表达和管理，它不仅支持变长记录，而且支持对象的嵌套、信息的继承与聚集。面向对象的空间数据库管理系统允许用户定义对象和对象的数据结构以及它的操作。这样，我们就可以将空间对象根据 GIS 的需要，定义出合适的数据结构和一组操作。这种空间数据结构可以是不带拓扑关系的面条数据结构，也可以是拓扑数据结构，当采用拓扑数据结构时，往往会涉及对象的嵌套、对象的连接和对象与信息的聚集。

虽然面向对象数据库屏蔽了原有的关系理论，具有完全面向对象的特征，具有很强的理论优势，但其商业化产品则侧重于程序设计语言对复杂数据的访问，同时还存在查询表达能力不强、查询效率比较低的问题。而且由于其中的许多技术细节还不够成熟，因此目前 OODB 还难以用于与地理信息系统应用相关的领域。

（3）对象—关系模型数据库

对象－关系模型数据库（ORDB）是面向对象思想与数据库管理系统相结合的一个折中产物，它与 OODB 的区别在于，它没有抛弃关系理论，而是在关系模型的基础上，进一步支持了面向对象思想。这样不但继承了关系模型的优良特性，并兼容了原有的使用关系数据库的软件系统；而且提供了支持面向对象的类型与接口，也满足了面向对象软件系统的要求。不论在科学研究中，还是在实际应用中，它都占有重要的地位，是当前数据管理技术的中流砥柱。ORDB 在与 GIS 结合的过程中，具有如下明显的优势：① 支持基本类型扩充。② 支持复杂对象。③ 支持继承。④ 支持规则。这样，通过对对象存储访问的支持，不仅解决了关系型数据库所面临的难题，同时通过兼容关系数据库系统，保证了对现有运行系统升级的可能性。而最为重要的是，ORDB 已经拥有比较成熟的商业化产品，这为它与 GIS 的结合打下了良好的基础。

对象—关系模型把关系理论与面向对象思想有机结合，不但继承了关系系统优良的特性，还提供使用面向对象方法建模的能力，满足了用户对数据库表达能力的要求。

① 空间扩展模型

此方案是基于对象－关系模型数据库而建立的。它通过建立一个空间对象表达模型，把空间数据和属性数据存储统一在 DBMS 之下，从而把 GIS 的数据管理部分的功能统一到 DBMS 之中，由 DBMS 完成数据层功能。其模型的重点在数据库上，希望通过增强数据库的空间功能而从根本上解决问题。这种扩展的空间对象管理模块，主要解决了空间数据变长记录的管理。由于由数据库软件商进行扩展，因此效率要比前面所述的二进制块的管理高得多。空间扩展模型体现出了两个优势：① 由数据库厂商支持的数据分析操作可以得到数据库系统内部优化机制的支持，因而效率较高。② 该模型在 DBMS 层为访问空间对象提供了统一的接口，也方便了系统从 RDB 到 ORDB 的移植。

但是，它还存在着技术不成熟的问题，如它仍然没有解决对象的嵌套问题，空间数据结构也不能由用户任意定义，不能提供对象—关系系统应具有的所有功能。同时空间扩展模型是数据库

厂商提供的解决方案，它希望通过增强数据库的空间功能来从根本上解决问题。由于该方案是由数据库厂商开发的，因此能够比较充分地利用底层数据库，从而达到比较好的性能。但是，数据库的空间扩展毕竟是数据库系统的附属产品，它不但继承了底层数据库的优势，也受到底层数据库的制约。例如，只有几个大的数据库厂商提供对空间扩展模型的支持，而这些数据库产品又只能应用于比较大的应用系统中，这样空间扩展模型也就难以投入到中小应用中去。因此这种模型的使用仍然受到了一定程度的限制。

② 基于中间件的数据库模型

此方案也是基于对象－关系模型数据库而建立的，其本质就是在数据库和最终用户之间增加中间件，通过中间件层实现从空间对象模型到数据库存储的映射，从而避免了对 DBMS 内核的直接修改，以消除数据库和最终用户间接口的差异，这样一方面能充分利用数据库提供的功能，另一方面也能满足用户最终的需求。这种模型的典型例子是 ESRI 的 ArcSDE。

中间件的主要任务就是分析并执行空间对象访问命令，为 GIS 应用提供一个一致且稳定的接口。基于中间件的数据库模型一般包括用户、中间件、数据存储层等 3 个层次。其中，用户使用地理信息，数据存储层负责存储地理信息，而中间件则负责用户与数据库之间的数据访问管理。对上层（GIS 应用及 SDSS 应用）的接口，提供完整的空间对象管理功能，统一管理空间属性和用户属性。对下层（数据库）的接口，则将空间属性与用户属性分开管理，空间属性的管理功能由中间件和 DBMS 共同实现，用户属性的管理功能则由 DBMS 直接实现。由于支持面向对象方法，因此中间件为空间对象提供了统一的访问接口，同时中间件可以实现 DBMS 不能完成但效率不高的部分。

基于中间件的数据库模型则是 GIS 厂商提供的解决方案，它希望通过提供灵活方便的接口来适应多变的应用需求，即通过设计中间件来封装数据管理部分，以提供对多种类型数据源的支持。这虽然降低了 GIS 应用开发的复杂度，但是，增加中间件也势必在空间数据存取中增加了额外开销，从而影响系统的整体效率，另外，构造并维护中间件的人力物力投入也不容忽视。与空间扩展模型相比，中间件模型是一种以性能换功能的方法，但它在当前的应用中却又是相当有效的。其特点也是很明显的，主要表现在以下几个方面。

- GIS 应用接口稳定，使得基于它的 GIS 应用也具有较强的稳定性。
- 易于裁剪，采用构件方式实现可以进行多层次的裁剪，以取得良好的性能/功能比。
- 数据库接口统一，不依赖于特定的数据库产品。
- 易于扩展，可以针对效率问题在中间件层提供一定的优化，如智能 CACHE 技术、空间索引过滤和查询优化，可以针对领域应用扩展功能，比如为了满足空间数据库在 SDSS 中的特殊要求，还可以在中间件层添加辅助空间数据挖掘的调用模块和支持空间知识库系统的调用模块等。

从本质上看，基于中间件的数据库模型和空间扩展模型的目的都是一样的，即都要为 GIS 应用提供方便高效的服务，都要提供一个空间对象的统一访问接口。而它们的区别在于，基于中间件的数据库模型是对应用需求的直接响应，也是空间扩展模型的基础，而空间扩展模型则是基于中间件的数据库模型的总结与改进。相比之下，由于基于中间件的数据库模型具有灵活的结构，所以它不但在研究方面很有潜力，而且可以方便地实验各种数据管理技术。

6.3　空间数据模型

空间数据模型是地理信息系统的基础，它不仅决定了系统数据管理的有效性，而且是系统灵

活性的关键。空间数据模型是在实体概念的基础上发展起来的，它包含两个基本内容，即实体组和它们之间的相关关系。实体和相关关系可以通过性质和属性来说明。空间数据模型可以被定义为一组由相关关系联系在一起的实体集。

结合空间数据的具体特点进行空间数据模型的设计，是地理信息系统的关键。空间数据模型的设计，主要是构建一个能够用真实世界的抽象提取来代表该真实世界的模型。由于空间数据模型的设计与计算机硬件、系统软件和工具软件的发展现状密切相关，所以，就目前的发展现状而言，很难用一个统一的数据模型来表达复杂多变的地理空间实体。例如，某些空间数据模型可能很适合于绘图，但它们对于空间分析来说效率却十分低；有些数据模型有利于空间分析，但对图形的处理则不理想。

目前，与 GIS 设计有关的空间数据模型主要有矢量模型、栅格模型、数字高程模型、面向对象模型、矢量和栅格的混合数据模型等。前 4 种模型属于定向性模型，在模型设计时只包括与应用目标有关的实体及其相互关系，而混合模型的设计则包括所有能够指出的实体及其相互关系。就目前的应用现状而言，矢量模型、栅格模型、数字高程模型已相当成熟（目前成熟的商业化 GIS 主要是采用这 3 类模型），而其他模型，特别是混合模型则处于大力发展之中。

6.3.1 矢量模型（vector model）

矢量模型是利用边界或表面来表达空间目标对象的面或体要素，通过记录目标的边界，同时采用标识符（Identifier）表达它的属性来描述空间对象实体。矢量模型能够方便地用于比例尺变换、投影变换以及图形的输入和输出。矢量模型处理的空间图形实体是点（point）、线（line）和面（area）。矢量模型的基本类型起源于“Spaghetti”模型。在 Spaghetti 模型中，点用空间坐标对表示，线由一串坐标对表示，面则是由线形成的闭合多边形。CAD 等绘图系统大多采用 Spaghetti 模型。

GIS 的矢量数据模型与 Spaghetti 模型的主要区别是：前者通过拓扑结构数据来描述空间目标之间的空间关系，而后者则没有。在矢量模型中，拓扑关系是进行空间分析的关键。在 GIS 的拓扑数据模型中，与点、线、面相对应的空间图形实体主要有结点（node）、弧段（arc）和多边形（polygon），多边形的边界被分割成一系列的弧和结点，结点、弧、多边形间的空间关系在数据结构或属性表中加以定义。GIS 的矢量数据模型具有以下几个特点。

① 通过对结点、弧、多边形拓扑关系的描述，相邻弧段的公用结点和相邻多边形的公用弧段，在计算机中只需记录一次。而在 Spaghetti 模型中，记录的次数则大于 1。

② 空间图形实体的拓扑关系，如拓扑邻接、拓扑关联、拓扑包含等，不会随着诸如移动、缩放、旋转等变换而变化，而空间坐标及一些几何属性（如面积、周长、方向等）则会受到影响。

③ 一般情况下，通过矢量模型所表达的空间图形实体数据文件占用的存储空间要比栅格模型小。

④ 能够精确地表达图形目标，精确地计算空间目标的参数（如周长、面积）。

6.3.2 栅格模型（raster model）

栅格模型直接采用面域或空域枚举来直接描述空间目标对象。在栅格模型中，点（点状符号）是由一个或多个像元，线是由一串彼此相连的像元构成。在栅格模型中，每一像元的大小是一致

的（一般是正方形），而且每一个栅格像元层记录着不同的属性（如植被类型等）。像元的位置由纵横坐标（行列）决定。所以，每个像元的空间坐标不一定要直接记录，因为像元记录的顺序已经隐含了空间坐标。

栅格模型具有以下几个特点。

① 栅格的空间分辨率是指一个像元在地面所代表的实际面积大小（一个正方形的面积）。

② 对于同一幅图形或图像来说，随着分辨率的增大，存储空间也随之增大。例如每一像元占用一个字节，而且分辨率为 100m，那么，一个面积为 10km×10km=100km 的区域就有 1000×1000=1000000 个像元，所占存储空间为 1000000 个字节；如果分辨率为 10m，那么，同样面积的区域就有 10000×10000=1 亿个像元，所占存储空间近 100MB。

③ 表达空间目标、计算空间实体相关参数的精度与分辨率密切相关，分辨率越高，精度越高。

④ 常适合进行空间分析。例如，同一地区多幅遥感图像的叠加操作等。

⑤ 不适合进行比例尺变化和投影变换等。

6.3.3 数字高程模型（DEM，Digital Elevation Model）

数字高程模型是采用规则或不规则多边形拟合面状空间对象的表面，主要是对数字高程表面的描述。根据多边形的形状，可以把数字高程模型分为两种，即网格模型和不规则三角网模型。

（1）网格模型(Grid model)

与栅格模型相似，同样是直接采用面域或空域枚举来描述空间目标对象。一般情况下，栅格模型的每一像元或像元的中心点代表一定面积范围内空间对象或实体的各种空间几何特征和属性几何特征，而网格模型通常以行列的交点特征值代表交点附近空间对象或实体的各种空间几何特征和属性几何特征。栅格模型主要用于图像的分析和处理，而网格模型则主要进行等值线的自动生成，坡度、坡向的分析等。栅格模型处理的数据主要来源于航空、航天摄影以及视频图像等，而网格模型则主要来源于原始空间数据的插值。

（2）TIN 模型

TIN 模型是利用不规则三角形来描述数字高程表面。在 TIN 模型中，同样可以建立三角形顶点（数据点）、三角形边、三角形个体间的拓扑关系。如果建立了 TIN 模型图形实体（三角形顶点、三角形边、三角形）的拓扑关系，将大大加快处理三角形的速度。

归纳起来，数字高程模型的主要优点是能够方便地进行空间分析和计算，如对地表坡度、坡向的计算等。

6.3.4 面向对象的数据模型（Object-Oriented Data Model）

面向对象表示方法的最大优点是便于表达复杂的目标。面向对象的方法为数据模型的建立提供了分类、概括、联合和聚集等 4 种数据处理技术，这些技术对复杂空间数据的表达较为理想。在概括、联合、聚集等技术的运用中，都要涉及对象或类型的属性值或属性结构在不同级或者层之间的传递或继承。为此，面向对象的方法提供了继承和传播两种工具。

在类型的层次结构中，子类的属性结构或操作方法可以部分地从超类中获取，这就是继承。继承可以减少数据冗余，并且有助于保持数据的完整性。

无论是联合还是聚集，它们都有一个共同之处，那就是将一组对象合并成为一个更加复杂的

对象，这一类复杂对象的属性值来源于两个方面：

① 一部分属性值由该复杂对象本身定义，与构成它的成员对象（组件对象）无关；

② 另一部分属性值依赖于这些成员对象。

为此，复杂对象必须具备获取成员对象属性值的能力。传播正是一种用来描述复杂对象的依赖性并获取成员对象属性值的工具，这种工具所基于的最基本的原理就是成员对象的相关属性只能存储一次，然后再将这些属性值传给复杂对象，这样当成员对象的属性值被改变后，复杂对象的属性值则无需修改。显然，这种工具的使用大大地减少了数据冗余量，并保证了数据的一致性。

通过采用继承和传播两种工具，将使得原本十分复杂的描述变得自然和简单。例如，一个城市由若干个区组成，因此，城市的人口应为各区人口的总和。对于城市这样一个复杂对象，它的人口数由它的成员对象，即各区的人口数求和派生而成。

6.3.5 混合数据模型（Hybrid Model）

由于矢量模型、栅格模型、数字高程模型在处理空间对象时都有各自的优缺点，所以，能否在一个统一模型的基础上充分利用相关模型的优点就是目前 GIS 界研究的方向之一。到目前为止，有代表性的研究成果主要有 TIN 与矢量的一体化模型，栅格与矢量一体化的多级网格模型。下面简单介绍一些这方面的内容。

（1）矢量与 TIN 的一体化的数据模型

发展矢量与 TIN 一体化数据模型的主要原因是：它既能发挥 TIN 的空间分析和计算功能，又能方便地查询属性信息。因为 TIN 的三角形顶点和边（如测量点、河流走向线、陆上径流路径、封闭洼地边界线等）可能为矢量图形的特征点和线的一部分，所以它们的系统代码或内部码是完全一致的。模型将空间数据类型抽象为点状目标、线状目标、面状目标，这 3 类图形目标又通过特殊点、结点、一般线段、弧以及多边形等加以表达。

TIN 与矢量之间存在着部分（part of）或包含的关系，实际上也是通过这种关系实现一体化的。由于 TIN 是以矢量系统的矢量图形为基础生成的，所以，单个三角形顶点、边、三角形本身包含的空间区域可能是矢量系统点、线、面状图形目标的一部分。如果在 TIN 和矢量数据结构中保持点和顶点、线和边、面和三角形有关标识号的统一，那么，就能实现 TIN 和矢量的一体化。

（2）多级网格模型

多级网格模型的本质是在基本网格的基础上细分为 256×256 或 16×16 个网格，然后基本网格和细分网格都采用四叉树的编码方法去表达点、线、面的有关参数。具体来讲，就是要遵循如下 3 点约定：

① 点状地物是仅有空间位置，没有形状和面积，在计算机内仅有一个位置数据；

② 线性地物是有形状，没有面积，在计算机内由一组原子填满路径；

③ 面状地物是有形状和面积，在计算机内是由一组填满路径的原子表达的边界线。

6.4 Geodatabase 体系结构

6.4.1 Geodatabase 介绍

随着关系数据库技术的完善，变长信息的存储成为可能，人们开始采用关系数据库技术管理

空间数据，并逐渐成为主导。利用关系数据库管理空间几何信息、属性信息、拓扑信息、三维信息以及多媒体信息，使得 GIS 软件能够充分地利用商用数据库中已经成熟的众多特征，如快速索引、数据完整性和一致性、并发控制等机制。在此情况下，ESRI 公司推出了 Geodatabase，该数据模型采用全关系数据库管理空间数据，与各种商用关系数据库兼容。Geodatabase 充分利用面向对象技术，将空间要素的属性和行为有机地结合在一起，采用现代化的、标准的方式实现对象，对象都定义为一个 COM 组件，并允许用户在这些基本模型的基础上扩展自己的面向对象数据模型。

Geodatabase 是面向对象的空间数据存储模型，ArcSDE 是 ESRI 公司针对空间数据的存储问题推出的一套空间数据库管理软件。通过 SDE，用户可以将多种数据产品按照 Geodatabase 模型存储于商用数据库系统中，并获得管理和检索服务。

Geodatabase 提供了可伸缩的空间数据存储方案，提供了两种存储方式：personal database 和 Enterprise Geodatabase。Personal database 适用于单机环境下的 GIS 应用，实际上就是一个 Access 数据库，Personal database 通过 Microsoft Jet Engine 将空间数据存放在 Access 数据库中。Enterprise Geodatabase 主要用于在网络环境下工作的 GIS，适合大型数据库在网络环境下进行多用户并发操作和版本控制。GIS 用户可以通过建立版本来并发访问 Geodatabase 中的空间数据。

Geodatabase 提供了 4 种空间数据的表述形式：描述要素的矢量数据、描述影像和网格的栅格数据、描述表面不规则三角网络、用于地址定位的 Addresses 和 Locator。

6.4.2 Geodatabase 的体系结构

Geodatabase 的基本体系结构包括要素数据集、栅格数据集、TIN 数据集、独立的对象类、独立的关系类和属性域等。其中要素数据集又由对象类、要素类、关系类等构成。

（1）对象类（Object Class）

对象类表示的是非空间实体，不在地图上直接表示，但与地图上的直接表示的图形要素关联。对象类可以组织到一个要素数据集中，也可以单独存在。

（2）要素类（Feature Class）

要素类表示的是具有相同几何形状的空间实体。要素类可以独立存在，也可以具有拓扑关系。当不同的要素类之间存在拓扑关系时，应组织到一个要素数据集中。要素类可细分为点状要素类、线状要素类、面状要素类等。

要素类与对象类的主要区别在于：要素类中存储了空间信息，而对象类中却没有。

（3）要素数据集（Feature Datasets）

要素数据集由一组具有相同空间参考（spatial reference）的要素类组成，用于存放矢量数据。

（4）栅格数据集（Raster Datasets）

栅格数据集用于存放栅格数据，可以支持海量栅格数据，支持影像镶嵌。可以通过建立“金字塔”索引，从而提高海量栅格数据的检索速度和显示效率。

（5）TIN 数据集（Tin Datasets）

TIN 数据集由一系列不规则的三角形构成，这些不规则的三角形表示了地表的起伏。

（6）关系类（Relationship Class）

关系类用于定义两个不同的要素类或对象类之间的关联关系。

（7）属性域（Domain）

属性域用于定义属性的有效取值范围，可以是连续的变化区间，也可以是离散的。

（8）几何网络（Geometric Network）

几何网络是在若干要素类的基础上建立的一个新的类。几何网络包括网络的边要素和点要素，例如供水系统中的几何网络，水管是边要素，点要素就是阀门。

6.5 Geodatabase 对象模型

6.5.1 Geodatabase 中的主要类

Geodatabase 中的主要类可以分为工作区部分和数据集部分，这两部分包含了 Geodatabase 中最基础、最主要的类。工作区部分主要负责对 Geodatabase 中的各种数据源进行宏观管理，而数据集部分主要是用于对数据库中各种具体的数据进行描述和管理。

（1）WorkspaceFactory 类

该类用于生成 Workspace，该类允许用户通过事先指定的连接属性连接一个 Workspace，这些连接属性一般使用 PropertySet 对象定义，并且可以存放在一个连接文件中。WorkspaceFactory 是一个可创建的对象，具有一个连接池，该连接池存放了与当前应用连接、处于激活状态的 Workspace。该类还提供了浏览、管理基于文件系统的 Workspace 的方法，同时还提供了管理远程数据库中的数据的功能。

该类实现了 IWorkspaceFactory 和 IWorkspaceFactory2 两个接口，提供了用于创建、打开 Workspace 和获取一个 Workspacefactory 的属性的方法。该类还派生 ArcInfoWorkspaceFactory、CadWorkspaceFactory、IMsWorkspaceFactory、ShapefileworkspaceFactory、SdeWorkspaceFactory 和 AccessWorkspaceFactory 等。

IworkspaceFactory 接口提供了创建和打开 Workspace 的方法，同时提供了获取 Workspace 的相关属性信息的方法。该接口提供了 10 个方法和两个属性。如 Copy 方法，用于将一个 Workspace 复制到指定的目录；GetWorkspaceName 方法，用于获得一个具有指定文件名列表对应的 Workspace 的名称；Move 方法，用于将一个 Workspace 移到指定的文件夹下；Open 方法，用于打开一个具有指定连接属性的 Workspace，这个方法用于打开 ArcSDE 数据库；OpenFromFile 方法，用于根据指定的文件名或者目录打开一个 Workspace，这个方法通常用于打开文件系统工作区或本地数据库工作区，也可以根据连接属性文件打开 SDE 数据库工作区。

IWorkspaceFactory2 接口的作用和 IWorkspaceFactory 接口基本一致，只是这个接口另外提供了直接根据一个描述连接属性的字符串来打开一个 Workspace 的方法 OpenFromString。

（2）Workspace 类

Workspace 是一个用于存放空间数据和非空间数据的容器，可以存放 FeatureDatasets、Raster datasets 和 Tables 等各种数据。一个 Workspace 就是一个数据集集合，该类提供了列举、添加、删除、复制和更改它所包含的数据集的方法，还提供了创建数据集的方法。根据枚举 esriWorkspaceType，Workspace 可以分为 esriFileSystemWorkspace（文件系统工作区）、esriLocalDatabaseworkspace（本地数据库工作区）、esriRemoteDatabaseWorkspace（分布式远程数据库工作区）等。Shapefiles 和 ArcInfo workspaces 就是文件系统工作区类型的 Workspace。该类实现了 IWorkspace、IWorkspace2、IWorkspaceDomains2 等众多接口。

一个 Workspace 可以被看做是文件系统中的目录，也可以被看做一个关系数据库。在访问一个 Workspace 中的数据库之前，用户首先必须打开这个 Workspace。用户不能直接创建一个

Workspace，一般是由 WorkspaceFactory 生成。

IWorkspace 接口提供了获取 Workspace 各种信息的方法，该接口提供了 3 个方法和 6 个属性。如 Exists 用于判断当前 Workspace 是否存在，IsDirectoy 用于判断当前 Workspace 是否是文件系统下的一个目录，WorkspaceFactory 用于获取当前 Workspace 对应的 WorkspaceFactory。

（3）FeatureDataset 组件类

FeatureDataset 组件类是一个用来存储具有与该 FeatureDataset 同空间参考的多个 Feature Class 的数据集。FeatureDataset 还可以存储 Geometric Network 和 Relationship Class。该类实现了 IFeatureDataset 接口，该接口提供继承 IDataset 接口。IFeatureDataset 提供了在一个 Dataset 中创建一个新的 Feature Class 的功能，共有 7 个方法和 9 个属性，如 CreateFeatureClass 方法用于创建一个新的 Feature Class。

6.5.2 Geodatabase 中的其他常用类

Geodatabase 的 OMD 中还提供了一些常用类，如与空间数据入库相关的类、与查询相关的类。

空间入库相关类还同时实现了各种数据格式的转换，如 FeatureDataConverter、FieldChecker、EnumFieldError 等。其中 FeatureDataConverter 类是最核心的类，所有的数据导入功能都是通过这个类实现的。FeatureDataConverter 类主要用于实现类似 ArcCatalog 的功能；FieldChecker 类用于对属性字段进行有效性判定；EnumFieldError 类是一个枚举，用来记录所有数据转换失败的无效字段的名称。

FeatureDataConverter 类实现了 IfeatureDataConverter 和 IFeatureDataConverter2 两个接口。IFeatureDataConverter 接口用于对不同格式的数据进行相互转换，提供了 3 个方法，如 ConvertFeatureClass 用于将 FeatureClass 导入到数据库中，ConvertFeatureDataset 用于将要素数据集导入到数据库中，ConvertTable 用于将一个表格数据导入到数据库中。IFeatureDataConverter2 接口在功能上基本与 IFeatureDataConverter 接口一致。

FieldChecker 类实现了 IFieldChecker 接口，用于对属性字段进行有效性判定，如 Validate 方法用于检查一系列属性字段的有效性。

查询相关组件类有 QueryFilter 和 SpatialFilter，这两个类将在第 10 章介绍。

6.6 Geodatabase 的使用与开发

6.6.1 空间数据库连接

空间数据库企业级 Geodatabase 的连接主要是使用 IworkspaceFactory、Iworkspace 和 IpropertySet 等 3 个接口，通过 IPropertySet 设置数据库的连接属性，由 IworkspaceFactory 创建一个 IWorkspace 的接口指针并返回该指针。6.6.3 小节的代码演示了如何进行空间数据库的连接。

6.6.2 创建新的数据集

数据集是由一组具有相同空间参考的要素类组成，用于存放数据。创建新的数据集主要使用

IWorkspace、IFeatureWorkspace、IfeatureDataset 和 IspatialReference 等 4 个接口，其中 IWorkspace 用于生成 IFeatureWorkspace，ISpatialReference 用于设置要素数据集的空间参考属性（该属性为可选属性），整个过程返回 IFeatureDataset 接口指针，即新生成的要素数据集。

6.6.3 空间数据的入库

本小节通过一个例子来演示如何加载数据到空间数据中。

新建一个 VS2005 工程，工程命名为“lesson1”，在 FORM1 窗体上添加一个 Mapcontrol 控件，两个 GroupBox 控件，7 个 Label 控件，7 个 TextBox 控件，4 个 Button 控件，然后修改 GroupBox、Label 和 Button 的 Text 属性，效果如图 6-1 所示。

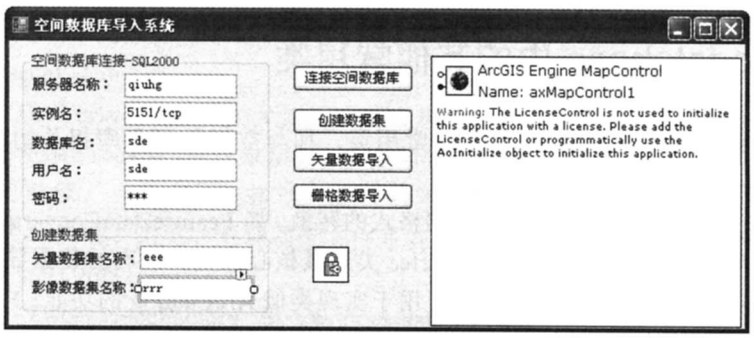


图 6-1　工程 lseeon1 添加控件效果

例子代码使用 SDE 空间数据库，因此 ArcEngine 的 Lisence 应使用企业数据库类型，在 Form1_Load 事件中初始化 lisence，不能使用 lisence 控件设置，否则提示没有许可 lisence。全部实例代码如下。

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using ESRI.ArcGIS.esriSystem;
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.Controls;
using ESRI.ArcGIS.SystemUI;
using ESRI.ArcGIS.Geometry;
using ESRI.ArcGIS.Geodatabase;
using ESRI.ArcGIS.DataSourcesFile;
using ESRI.ArcGIS.Display;
using ESRI.ArcGIS.DataSourcesGDB;
using ESRI.ArcGIS.DataSourcesRaster;

namespace lesson1
{
    public partial class Form1 : Form
```

```
{
    public Form1()
    {
        InitializeComponent();
    }

    private void Form1_Load(object sender, EventArgs e)
    {
        IAoInitialize pao = new AoInitializeClass();
        pao.Initialize(esriLicenseProductCode.esriLicenseProductCodeEngineGeoDB);

    }
    //  工作空间
    IWorkspace workspace;
    //矢量数据工作空间
    IFeatureWorkspace featureWorkspace;
    //影像数据工作空间
    IRasterWorkspaceEx rasterWorkspace;
    //矢量数据集
    IFeatureDataset featureDataset;
    //影像数据集
    IRasterDataset rasterDataset;
    private void button1_Click(object sender, EventArgs e)
    {
       // SDE 空间连接属性
        IPropertySet propertySet = new PropertySetClass();
        propertySet.SetProperty("server", this.textBox1.Text );
        propertySet.SetProperty("instance",this.textBox2.Text );
        propertySet.SetProperty ("database",this.textBox3.Text );
        propertySet.SetProperty("user", this.textBox4.Text );
        propertySet.SetProperty ("password",this.textBox5.Text );
        propertySet.SetProperty("version", "SDE.DEFAULT");
        IWorkspaceFactory workspaceFactory = new SdeWorkspaceFactory();
       //打开 SDE 工作空间

        workspace = workspaceFactory.Open(propertySet, 0);
        MessageBox.Show("连接 SDE 空间数据库成功");

    }
    //创建数据集（矢量数据集和影像数据集）
    private void button2_Click(object sender, EventArgs e)
    {
        featureWorkspace =workspace as IFeatureWorkspace ;
        rasterWorkspace = workspace as IRasterWorkspaceEx;
        //定义空间参考
        ISpatialReferenceFactory spatialReferenceFactory = new SpatialReference
EnvironmentClass();
        ISpatialReference   spatialReference   =   spatialReferenceFactory.
CreateGeographic CoordinateSystem((int)esriSRGeoCSType.esriSRGeoCS_Beijing1954);
        spatialReference.SetDomain(-1000, -1000, 1000, 1000);
```

```
            IEnumDatasetName enumDatasetName;
            IDatasetName   datasetName;
            string dsName = "";
            enumDatasetName = workspace.get_DatasetNames(esriDatasetType.esriDTFeature
Dataset);
            datasetName = enumDatasetName.Next();
            bool isExist = false;
            //创建矢量数据集
            dsName = "SDE."+this.textBox6.Text;
            while (datasetName != null)
            {
                if (datasetName.Name == dsName)
                {
                    isExist = true;
                }
                datasetName = enumDatasetName.Next();
            }
            if (isExist ==false  )
            {
                featureDataset   =   featureWorkspace.CreateFeatureDataset(this.
textBox6.Text, spatialReference);
            }
            //创建影像数据集
            isExist = false;
            enumDatasetName = workspace.get_DatasetNames(esriDatasetType.esriDTRaster
Dataset );
            datasetName = enumDatasetName.Next();

            dsName = "SDE." + this.textBox6.Text;
            while (datasetName != null)
            {
                if (datasetName.Name == dsName)
                {
                    isExist = true;
                }
                datasetName = enumDatasetName.Next();
            }
            if (isExist == false)
            {
                //设置存储参数
                IRasterStorageDef rasterStorageDef = new RasterStorageDefClass();
                rasterStorageDef.CompressionType   =   esriRasterCompressionType.
esriRaster CompressionUncompressed;
                rasterStorageDef.PyramidLevel = 1;
                rasterStorageDef.PyramidResampleType   =   rstResamplingTypes.RSP_
Bilinear Interpolation;
                rasterStorageDef.TileHeight = 128;
                rasterStorageDef.TileWidth = 128;
                //设置坐标系统
                IRasterDef rasterDef = new RasterDefClass();
                ISpatialReference rasterDpatialRefrence = new UnknownCoordinateSystem
```

```
Class();
            rasterDef.SpatialReference = rasterDpatialRefrence;

            IGeometryDef geometryDef = new GeometryDefClass();
            IGeometryDefEdit geometryDefedit = (IGeometryDefEdit)geometryDef;
            geometryDefedit.AvgNumPoints_2 = 5;
            geometryDefedit.GridCount_2 = 1;
            geometryDefedit.set_GridSize(0, 1000);
            geometryDefedit.GeometryType_2 = esriGeometryType.esriGeometry Polygon;
            ISpatialReference spatialReference2 = new UnknownCoordinateSystem Class();
            geometryDefedit.SpatialReference_2 = spatialReference2;
            rasterDataset = rasterWorkspace.CreateRasterDataset(this.textBox7.
            Text, 1, rstPixelType.PT_LONG, rasterStorageDef, "DEFAULTS", rasterDef,
            geometry Def);

        }

    }
    //加载矢量数据到 SDE 数据库
    private void button3_Click(object sender, EventArgs e)
    {
        featureWorkspace = workspace as IFeatureWorkspace;
        this.openFileDialog1.Filter = "shp file (*.shp)|*.shp";
        this.openFileDialog1.Title = "打开矢量数据";
        this.openFileDialog1.Multiselect = false;
        string fileName = "";
        if (this.openFileDialog1.ShowDialog() == DialogResult.OK)
        {
            fileName  = this.openFileDialog1.FileName;
            string filepath;
            string file;
            int lastIndex;
            lastIndex = fileName.LastIndexOf(@"\");
            filepath = fileName.Substring(0, lastIndex );
            file = fileName.Substring(lastIndex+1);
            //读取 SHP 数据
            IWorkspaceFactory shpwpf = new ShapefileWorkspaceFactoryClass();
            IWorkspace shpwp = shpwpf.OpenFromFile(filepath, 0);
            IFeatureWorkspace shpfwp = shpwp as IFeatureWorkspace;
            IFeatureClass shpfc = shpfwp.OpenFeatureClass(file);

            //导入 SDE 数据库
            IFeatureClass sdeFeatureClass=null;
            IFeatureClassDescription featureClassDescription = new
            FeatureClass DescriptionClass();
            IObjectClassDescription objectClassDescription =
            featureClassDescription as IObjectClassDescription;
            IFields fields = shpfc.Fields;
            IFieldChecker fieldChecker = new FieldCheckerClass();
            IEnumFieldError enumFieldError = null;
            IFields validateFields = null;
```

```
                fieldChecker.ValidateWorkspace = featureWorkspace as IWorkspace;
                fieldChecker.Validate(fields, out enumFieldError, out validateFields);
                featureDataset = featureWorkspace.OpenFeatureDataset(this.textBox6.
                Text);
                try
                {
                    sdeFeatureClass = featureWorkspace.OpenFeatureClass(shpfc.Alias Name);
                }
                catch (Exception  ex)
                {
                }
                //在 SDE 数据库中创建矢量数据集
                if (sdeFeatureClass == null)
                {
                    sdeFeatureClass = featureDataset.CreateFeatureClass(shpfc.
AliasName, validateFields, objectClassDescription.InstanceCLSID, objectClassDescription.
ClassExtension CLSID, shpfc.FeatureType, shpfc.ShapeFieldName, "");
                }
                IFeatureCursor featureCursor = shpfc.Search(null, true);
                IFeature feature = featureCursor.NextFeature();
                IFeatureCursor sdeFeatureCursor= sdeFeatureClass.Insert(true);
                IFeatureBuffer sdeFeatureBuffer;
                //添加实体对象
                while (feature != null)
                {
                    sdeFeatureBuffer = sdeFeatureClass.CreateFeatureBuffer();
                    IField shpField = new FieldClass();
                    IFields shpFields = feature.Fields;
                    for (int i = 0; i < shpFields.FieldCount; i++)
                    {
                        shpField = shpFields.get_Field(i);
                        int index = sdeFeatureBuffer.Fields.FindField(shpField.Name);
                        if (index != -1)
                        {
                            sdeFeatureBuffer.set_Value(index, feature.get_Value(i));
                        }
                    }
                    sdeFeatureCursor.InsertFeature(sdeFeatureBuffer);
                    sdeFeatureCursor.Flush();
                    feature = featureCursor.NextFeature();
                }
                //加载数据到 Mapcontrol
                IFeatureLayer sdeFeatureLayer = new FeatureLayerClass();
                sdeFeatureLayer.FeatureClass = sdeFeatureClass;
                this.axMapControl1.Map.AddLayer(sdeFeatureLayer as ILayer);
                this.axMapControl1.Extent = this.axMapControl1.FullExtent;
                this.axMapControl1.Refresh();
            }

        }
        //加载影像数据到 SDE 数据库
```

```
private void button4_Click(object sender, EventArgs e)
{
    this.openFileDialog1.Filter = "TIFF file (*.tif)|*.tif";
    this.openFileDialog1.Title = "打开影像数据";
    this.openFileDialog1.Multiselect = false;
    string fileName = "";
    if (this.openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        fileName = this.openFileDialog1.FileName;
        string filepath;
        string file;
        int lastIndex;
        lastIndex = fileName.LastIndexOf(@"\");
        filepath = fileName.Substring(0, lastIndex);
        file = fileName.Substring(lastIndex + 1);

        //导入 SDE 数据库
        rasterWorkspace = workspace as IRasterWorkspaceEx;
        IWorkspaceFactory tifwpf = new RasterWorkspaceFactoryClass();
        IWorkspace tifwp = tifwpf.OpenFromFile(filepath, 0);
        IRasterWorkspace tifrwp = tifwp as IRasterWorkspace;
        IRasterDataset rasterDataset = tifrwp.OpenRasterDataset(file);
        IRasterDataset sdeRasterDataset = null;
        lastIndex = file.LastIndexOf(@".");
        file = file.Substring(0, lastIndex);
        try
        {
            sdeRasterDataset = rasterWorkspace.OpenRasterDataset(file);
        }
        catch (Exception Ex)
        {
        }
        if (sdeRasterDataset == null)
        {
            IGeoDataset geoDataset =rasterDataset  as IGeoDataset ;
            IRasterSdeServerOperation rasterSdeServeroperation ;

            IBasicRasterSdeConnection sdeCon = new BasicRasterSdeLoader();
            IPropertySet propertySet = new PropertySetClass();
            propertySet = workspace.ConnectionProperties;
            //建立与 SDE 数据库的连接
            sdeCon.ServerName = propertySet.GetProperty("server").ToString();
            sdeCon.Instance = propertySet.GetProperty("instance").ToString();
            sdeCon.UserName = propertySet.GetProperty("user").ToString();
            sdeCon.Password = "sde";
            sdeCon.Database = propertySet.GetProperty("database").ToString();
            sdeCon.SdeRasterName = file;
            sdeCon.InputRasterName = fileName;
            rasterSdeServeroperation = sdeCon as IRasterSdeServerOperation;
            //保存影像数据到 SDE 数据库中
            rasterSdeServeroperation.Create ();
```

```
                    rasterSdeServeroperation.Update ();
                    rasterSdeServeroperation.ComputeStatistics ();
                    IRasterLayer rasterLayer=new RasterLayerClass ();
                    sdeRasterDataset = rasterWorkspace.OpenRasterDataset(file);
                    rasterLayer.CreateFromDataset(sdeRasterDataset);
                    this.axMapControl1.Map.AddLayer(rasterLayer as ILayer);
                    this.axMapControl1.Extent = this.axMapControl1.FullExtent;
                    this.axMapControl1.Refresh();
                }
            }
        }
    }
}
```

6.7 本章小结

空间数据库是地理信息系统的基础，空间数据库性能优劣直接关系到系统的性能。通过几十年的发展，空间数据库也逐步发展成熟，本章介绍空间数据库特征、并着重介绍了空间数据库的组织：混合型数据库和集成型空间数据库。空间数据库只管理和装载空间数据的容器，空间数据具有多种模型：矢量模型、栅格模型、数字高程模型、面向对象模型、混合数据模型。这些数据模型都有着各自的优点和缺点，在空间数据库的构建时应综合考虑这些模型的优劣，并进行权衡考虑。

ESRI 的 Geodatabase 是面向对象的空间数据存储模型，该模型提供了 4 种空间数据的表述形式：描述要素的矢量数据、描述影像和格网的栅格数据、描述表面不规则三角网络、用于地址定位的 Addresses 和 Locator。也是初学者应重点掌握的内容。