



---

# UDBX 开放数据格式白皮书

---

UDBX Open Data Format White Paper

**(Version 1.0 Beta)**

超图研究院

2020 年 9 月

目录

- 1. 概述..... 1
  - 1.1 基本约定..... 1
  - 1.2 总体结构..... 2
- 2. 系统表 ..... 3
  - 2.1 SpatialLite 系统表 ..... 4
  - 2.2 SuperMap 系统表..... 5
- 3. 数据表 ..... 11
  - 3.1 矢量数据集 ..... 11
  - 3.2 栅格数据集 ..... 15
- 4. 对象存储结构 ..... 17
  - 4.1 基本类型..... 17
  - 4.2 SpatialLite 的简单对象 ..... 19
  - 4.3 CAD 数据集中存储的对象..... 22
  - 4.4 文本对象..... 27
  - 4.5 三维模型对象..... 28
  - 4.6 栅格块存储 ..... 36
  - 4.7 其它对象..... 37
- 5. 应用指南..... 39

# 1. 概述

UDBX (Universal Database Extension) 是一种文件型的开放式数据格式，由超图软件提出，支持全空间数据的高效存储与管理，为空间数据的共享和交换提供开放、便捷的解决方案。

UDBX 的特点主要包括：

- 全空间数据支持：包括二三维一体化及矢量/栅格一体化的数据存储和管理；
- 单文件存储：采用单文件存储，便于数据拷贝和分发；
- 并发支持：一个 UDBX 数据可同时被多个用户并发访问；
- 跨平台支持：支持桌面端、服务器端、移动端高效读写；
- 海量数据支持：单个 UDBX 文件支持超大规模数据存储；
- Unicode 支持：支持 Unicode 编码存储多国语言。

## 1.1 基本约定

本文出现的术语或缩略语遵循以下约定：

- (1) OGC：开放地理空间信息联盟 (Open Geospatial Consortium)；
- (2) OGC 简单对象模型规范：即 OGC Simple feature access - Part 1: Common architecture 规范中规定的简单对象模型；
- (3) SQLite：一种开源的轻量级关系型数据库；
- (4) SpatiaLite：基于 SQLite 的空间扩展引擎，可以存储和管理符合 OGC 简单对象模型规范的空间数据；
- (5) proj.4：OSGeo 的开源 GIS 工具，专注于地图的坐标系表达以及转换；
- (6) WKT：即 OGC 的 Well Known Text；
- (7) OGDC：开放式空间数据库互访开发标准接口 (Open Geospatial Database Connectivity)，基于国家标准《地理空间数据库访问接口》(GB/T30320-2013)的一套 C++ 接口，目的是实现不同格式空间数据库的互联互通；
- (8) 数据集：由具有相同坐标系的同种类型数据组成的数据集；
- (9) 矢量数据集：一组具有相同 Geometry 类型 (CAD 数据集除外)、相同的空间参照系和相同的字段信息结构的矢量数据要素的集合；
- (10) 栅格数据集：具有长度和宽度范围的一组有规则的空间阵列数据的集合，这些阵列数据有相同的空间参照系，阵列数据值即代表对应空间位置的属性且所有值描述的是同类属性，比如温度、高程值或其它；
- (11) 数据源：由各种类型的数据集组成的数据集集合。

关于版本号：

- (1) 建议 SpatiaLite 采用 4.3 版本；
- (2) 建议 SQLite 采用 3.17.0 版本。

关于字段/数据类型：

- (1) 本文系统表 (见第 2 节) 及数据表 (见第 3 节) 表格中的“字段类型”为 SQLite 的字段类型，其中 INTEGER 类型默认为 32 位整型；
- (2) UDBX 提供的数据集字段信息，是基于 SQLite 字段类型的扩展，具体描述见

2.2.2.2 小节;

- (3) 对象的二进制流存储采用基本数据类型描述，各类型的含义、取值范围、占用字节数等见 4.1.1 小节。

## 1.2 总体结构

UDBX 采用 SQLite 数据库存储空间数据，文件扩展名为“.udbx”。存储规则上，分为两大类：Spatialite 存储和 SuperMap 自定义存储，Spatialite 存储点、线、面等基本矢量类型，其它数据类型为 SuperMap 自定义的存储格式。

UDBX 中的表格，分为两大类：系统表和数据表。系统表用于管理 UDBX 的数据集，因此，系统表又区分为 Spatialite 的系统表和 SuperMap 自定义的系统表；数据表用于存储不同的类型数据集，一个数据集对应一张或多张表格。UDBX 中表结构关系见图 1。

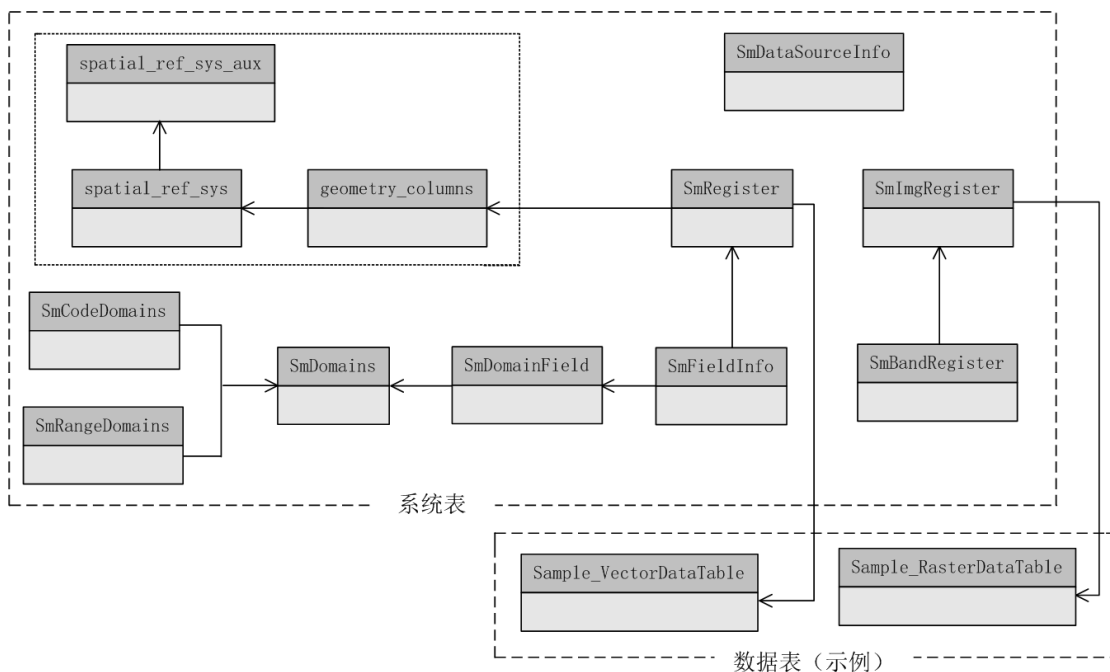


图 1 UDBX 表结构关系图

一个 udbx 文件是一个数据源，存储多个数据集。数据集包括两大类：矢量数据集和栅格数据集。矢量数据集存储具有相同坐标系和属性信息的空间对象，栅格数据集则按块存储和组织连续的场数据。数据集类型见表 1。

表 1 数据集类型说明

数据集类型	枚举值	描述
Tabular	0	属性表，不存储几何对象
Point	1	二维点数据集
PointZ	101	三维点数据集
Line	3	二维线数据集
LineZ	103	三维线数据集
Region	5	二维面数据集
RegionZ	105	三维面数据集
Text	7	文本对象

CAD	149	复合数据集，存储多种几何对象
Network	4	二维网络数据集
Network3D	205	三维网络数据集
Model	203	三维模型数据集
Image	88	多波段影像数据集
Grid	83	Grid 数据集，对应数字表面模型，像素值代表地表特征值
VoxelGrid	89	体元栅格数据集
Mosaic	206	镶嵌数据集

矢量数据集存储的 Geometry 类型见表 2。其中，二维/三维的点、线、面数据集存储的对象采用 Spatialite 的点、线、面对象，对象命名采用"GAIA"前缀标识；其它对象类型用"Geo"作为前缀标识。各 Geometry 对应的存储结构见第 4 小节。

表 2 Geometry 类型说明

Geometry 类型	枚举值	描述
GAIAPoint	1	二维点对象
GAIAPolygon	3	二维面对象
GAIAMultiLineString	5	二维线，可带子对象
GAIAMultiPolygon	6	二维面，可带子对象
GAIAPointZ	1001	三维点对象
GAIAMultiLineStringZ	1005	三维线，可带子对象
GAIAMultiPolygonZ	1006	三维面，可带子对象
GeoPoint3D	101	三维点
GeoLine3D	103	三维线，可带子对象
GeoRegion3D	105	三维面，可带子对象
GeoText	7	文本对象
GeoModel3D	1218	三维模型对象
GeoRect	12	矩形/斜矩形
GeoRectRound	13	圆角矩形
GeoCircle	15	圆
GeoEllipse	20	椭圆
GeoPie	21	扇面
GeoArc	24	圆弧
GeoEllipticArc	25	椭圆弧
GeoCardinal	27	Cardinal 曲线
GeoCurve	28	自由曲线
GeoBSpline	29	B 样条曲线

## 2. 系统表

系统表用于存储和管理数据源中的数据集信息，系统表分为 Spatialite 定义的系统表和 SuperMap 定义的系统表。

## 2.1 SpatiaLite 系统表

UDBX 采用 SpatiaLite 管理点、线、面数据集，因此主要涉及 SpatiaLite 的坐标系和矢量数据集相关系统表。

### 2.1.1 坐标系信息表

坐标系信息存储在 spatial\_ref\_sys 和 spatial\_ref\_sys\_aux 中，两张表通过 srid 关联。

表 3 spatial\_ref\_sys 表的字段信息

字段名	字段类型	是否允许空值	描述
srid	INTEGER	N	主键； 坐标系的唯一标识
auth_name	TEXT	N	定义该坐标系的作者/官方名称
auth_srid	INTEGER	N	该坐标系的内部标识
ref_sys_name	TEXT	N	坐标系名字
proj4text	TEXT	N	用 proj.4 <sup>[1]</sup> 文本格式表示的坐标系
srsrtext	TEXT	N	用 wkt <sup>[2]</sup> 表示的坐标系

表 4 spatial\_ref\_sys\_aux 表的字段信息

字段名	字段类型	是否允许空值	描述
srid	INTEGER	N	主键； 外键，与 spatial_ref_sys (srid)关联 坐标系的唯一标识
is_geographic	INTEGER	Y	是否是地理坐标系
has_flipped_axes	INTEGER	Y	坐标轴是否翻转
spheroid	TEXT	Y	参考椭球体
prime_meridian	TEXT	Y	中央子午线
datum	TEXT	Y	大地基准面
projection	TEXT	Y	投影方式
unit	TEXT	Y	坐标系单位
axis_1_name	TEXT	Y	主轴名称
axis_1_orientation	TEXT	Y	主轴朝向
axis_2_name	TEXT	Y	副轴名称
axis_2_orientation	TEXT	Y	副轴朝向

### 2.1.2 矢量数据集系统表

SpatiaLite 的矢量数据集系统表信息存储在 geometry\_columns 表中。

表 5 geometry\_columns 表的字段信息

字段名	字段类型	是否允许空值	描述
f_table_name	TEXT	N	数据表的名称

f_geometry_column	TEXT	N	数据表中 geometry 列名; 联合主键 (f_table_name, f_geometry_column)
geometry_type	INTEGER	N	geometry 类型, 见表 2
coord_dimension	TEXT	N	geometry 坐标的维度
srid	TEXT	N	坐标系标识, 与 spatial_ref_sys 表的 srid 字段关联
spatial_index_enabled	INTEGER	N	是否建立了空间索引; 取值: 0 表示无索引; 1 表示 R 树索引

## 2.2 SuperMap 系统表

SuperMap 自定义系统表, 包括数据源描述信息表、矢量数据集系统表、栅格数据集系统表。

### 2.2.1 数据源描述信息表

SmDataSourceInfo 表存储数据源的基本描述信息, 见表 6。

表 6 SmDataSourceInfo 表的字段信息

字段名	字段类型	是否允许空值	描述
SmFlag	INTEGER	N	数据源 ID 标识, 主键
SmVersion	INTEGER	Y	版本号。当前版本号为 10
SmDsDescription	TEXT	Y	数据源描述信息
SmProjectInfo	BLOB	Y	数据源的坐标系信息, 见 4.7.1
SmLastUpdateTime	DATE	N	数据源的最后更新时间
SmDataFormat	INTEGER	N	数据存储格式。 当前值为 0, 表示按 UTF8 编码存储

### 2.2.2 矢量数据集系统表

矢量数据集系统表包括数据集注册表、字段信息表、值域信息表。

#### 2.2.2.1 矢量数据集注册表

矢量数据集的注册信息记录在 SmRegister 表中, 包括数据集名称、对应的表名、数据集类型、父子数据集关系等, 见表 7。

表 7 SmRegister 表的字段信息

字段名	字段类型	是否允许空值	描述
SmDatasetID	INTEGER	N	主键; 数据集 ID

SmDatasetName	TEXT	Y	数据集名字
SmTableName	TEXT	Y	数据表名字
SmOption	INTEGER	Y	数据集的选项信息，记录是否带金字塔、是否压缩等，内部使用
SmEncType	INTEGER	Y	预留字段
SmParentDTID	INTEGER	N	父数据集 ID，可以为空
SmDatasetType	INTEGER	Y	数据集类型；枚举值见表 1
SmObjectCount	INTEGER	N	对象个数，即数据表的记录个数
SmLeft	REAL	Y	数据集的地理范围：左
SmRight	REAL	Y	数据集的地理范围：右
SmTop	REAL	Y	数据集的地理范围：下
SmBottom	REAL	Y	数据集的地理范围：上
SmIDColName	TEXT	Y	数据表对象 ID 列名
SmGeoColName	TEXT	Y	数据表几何对象列名
SmMinZ	REAL	Y	数据集最小高度，三维数据集适用
SmMaxZ	REAL	Y	数据集最大高度，三维数据集适用
SmSRID	INTEGER	Y	坐标系 ID，与 spatial_ref_sys 表的 srid 关联；如果该字段值为空，则取 SmProjectInfo 的值
SmIndexType	INTEGER	Y	空间索引类型；取值范围{0, 2}，0 表示无索引；2 表示 R 树索引
SmToleranceFuzzy	REAL	Y	结点捕捉容限值。拓扑处理/编辑时使用
SmToleranceDAngle	REAL	Y	角度容限值。拓扑处理/编辑时使用
SmToleranceNodeSnap	REAL	Y	长悬线容限值。拓扑处理/编辑时使用
SmToleranceSmallPolygon	REAL	Y	最小多边形容限值。拓扑处理/编辑时使用
SmToleranceGrain	REAL	Y	
SmMaxGeometrySize	INTEGER	N	几何对象二进制流最大字节数
SmOptimizeCount	INTEGER	N	预留字段
SmOptimizeRatio	REAL	Y	预留字段
SmDescription	TEXT	Y	数据集描述信息
SmExtInfo	TEXT	Y	数据集用户自定义扩展信息
SmCreateTime	DATETIME	Y	数据集创建时间
SmLastUpdateTime	DATETIME	Y	数据集最后更新时间
SmProjectInfo	BLOB	Y	数据集坐标系，见 4.7.1。如果指定了该值，则以数据集的坐标系为准，忽略数据源的坐标系

### 2.2.2.2 数据集字段信息表

矢量数据集的字段信息存储在 SmFieldInfo 表中，见表 8，主要记录每个数据集有哪些字段（与数据集主表的字段对应），每个字段的别名、对应 SuperMap 的字段类型等信息。



表 8 SmFieldInfo 表的字段信息

字段名	字段类型	是否允许空值	描述
SmID	INTEGER	N	主键
SmDatasetID	INTEGER	Y	所属数据集的 ID; 对应 SmRegister 的 SmDatasetID 字段
SmFieldName	TEXT	Y	字段名
SmFieldCaption	TEXT	Y	字段别名
SmFieldType	INTEGER	Y	对应 SuperMap 的字段类型, 见表 9 表 9 SuperMap 的字段类型
SmFieldFormat	TEXT	Y	字段值的格式化字符串
SmFieldSign	INTEGER	Y	字段标识, 见表 10
SmFieldDomain	TEXT	Y	废弃字段
SmFieldUpdatable	INTEGER	Y	字段值是否可修改
SmFieldRequired	INTEGER	Y	是否必填字段
SmFieldDefaultValue	TEXT	Y	字段默认值
SmFieldSize	INTEGER	Y	字段长度

表 9 SuperMap 的字段类型

枚举值	字段类型	字节长度	取值范围	描述
0	Unknown	/		无效值
1	Boolean	1	0 1	布尔型
2	Byte	1	[0, 255]	无符号单字节
3	Int16	2	[-32768, 32767]	短整型
4	Int32	4	[-2147483648, 2147483647]	整型
16	Int64	8	$[-2^{63}, (2^{63}-1)]$	长整型
6	Float	4	$[-3.4 \times 10^{38}, 3.4 \times 10^{38}]$	单精度浮点型
7	Double	8	$[-1.7 \times 10^{308}, 1.7 \times 10^{308}]$	双精度
10	Text	外部指定	/	不定长字符串
127	NText	外部指定	/	宽字节不定长字符串类型
18	Char	外部指定	/	定长字符串
8	Date	/	/	日期型, 年、月、日。 不带时间
22	Time	/	/	时间型, 时、分、秒, 不带日期
23	TimeStamp	/		时间戳型, 年、月、 日、时、分、秒
9	Binary	外部指定	/	固定长度二进制类型
11	LongBinary	外部指定	/	不定长二进制类型

表 10 SuperMap 的字段标识

枚举值	描述
0	普通字段

1	网络数据集的节点 ID，默认是 SmNodeID 字段
2	网络数据集的边的起点字段，默认是 SmFNode 字段
3	网络数据集的边的终点字段，默认是 SmTNode 字段
4	网络数据集边的 ID 字段
11	对象 ID 字段
12	几何对象字段
50	用户自定义字段标识起始值

### 2.2.2.3 值域信息表

值域分为范围值域（见表 11）和枚举值域（见表 12），用于约束字段的取值。

SmDomains 表汇总了两类值域的信息，见表 13；SmDomainField 记录了字段及其应用的值域规则，见表 14。

表 11 SmRangeDomains 表的字段信息

字段名	字段类型	是否允许空值	描述
DomainID	INTEGER	N	主键
FieldType	INTEGER	N	字段类型
DomainRangeInfos	BLOB	Y	范围值域规则对象，见 4.7.2

表 12 SmCodeDomains 表的字段信息

字段名	字段类型	是否允许空值	描述
DomainID	INTEGER	N	主键
FieldType	INTEGER	N	字段类型
DomainCodeInfos	BLOB	Y	枚举值域规则对象，见 4.7.2

表 13 SmDomains 表的字段信息

字段名	字段类型	是否允许空值	描述
DomainID	INTEGER	N	主键； 值域规则 ID
DomainName	TEXT	N	值域名称
DomainDescription	TEXT	N	描述信息
DomainType	INTEGER	N	值域类型,取值： 1 表示范围值域，在范围内为合法； 3 表示范围值域，不在范围内为合法； 2 表示合法枚举值域，取枚举值为合法； 4 表示非法枚举值域，取枚举值为非法。

表 14 SmDomainField 表的字段信息

字段名	字段类型	是否允许空值	描述
DatasetID	INTEGER	N	数据集 ID； 与 SmRegister 表的 SmDatasetID 字段关联
FieldName	TEXT	N	应用值域规则的数据集的字段名；

			联合主键(DatasetID,FieldName), 与 SmFieldInfo 表的 (SmDatasetID, SmFieldName)关联, 见表 8
DomainID	INTEGER	N	值域规则 ID, 与表 13 的 DomainID 关联

## 2.2.3 栅格数据集系统表

栅格数据集由两张系统表管理: SmImgRegister 存储栅格数据集信息; SmBandRegister 存储栅格数据集的波段/层信息, 以及对应的金字塔数据集信息。

### 2.2.3.1 栅格数据集注册表

SmImgRegister 的字段信息见表 15。

表 15 SmImgRegister 表的字段信息

字段名	字段类型	是否允许空值	描述
SmDatasetID	INTEGER	N	主键; 数据集 ID
SmDatasetName	TEXT	N	数据集名字
SmTableName	TEXT	N	数据集表名
SmDatasetType	INTEGER	N	数据集类型, 枚举值见表 1
SmWidth	INTEGER	Y	数据集宽度, 像素单位
SmHeight	INTEGER	Y	数据集高度, 像素单位
SmeBlockSize	INTEGER	Y	存储块大小, 以像素为单位, 长宽一致; 取值: 64   128   256   1024
SmColorSpace	INTEGER	Y	废弃字段
SmGeoLeft	REAL	Y	地理范围: 左
SmGeoTop	REAL	Y	地理范围: 上
SmGeoRight	REAL	Y	地理范围: 右
SmGeoBottom	REAL	Y	地理范围: 下
SmCreateTime	DATE	N	数据集创建时间
SmCreator	TEXT	N	创建者
SmDescription	TEXT	Y	数据集描述信息
SmClipRegion	BLOB	Y	裁剪多边形; 显示时仅多边形内部区域可见
SmExtInfo	TEXT	Y	数据集扩展描述信息
SmStatisticsInfo	TEXT	Y	统计信息
SmProjectInfo	BLOB	Y	坐标系信息, 见 4.7.1

### 2.2.3.2 波段信息注册表

表 16 SmBandRegister 表的字段信息

字段名	字段类型	是否允许空值	描述
SmBandID	INTEGER	N	主键; 波段 ID

SmDatasetID	INTEGER	N	所属数据集 ID，与 SmImgRegister 的 SmDatasetID 关联
SmBandIndex	INTEGER	N	波段顺序编号
SmBandName	TEXT	N	波段名称
SmBandAvail	INTEGER	N	是否可用； 取值 1：可用；0：不可用
SmOption	INTEGER	Y	数据集选项信息，内部使用，记录是否带有金字塔等信息
SmScalar	INTEGER	Y	预留字段
SmEncType	INTEGER	N	Block 的压缩编码方式，见表 17
SmPixelFormat	INTEGER	N	像素格式，见表 18
SmMaxBlockSize	INTEGER	Y	存储块的最大大小； 单位：字节
SmMinZ	REAL	Y	像素值的最小值
SmMaxZ	REAL	Y	像素值的最大值
SmAltitude	REAL	Y	该层数据代表的高度； 外部设置
SmPyramid	TEXT	Y	父数据集名； 如果不为空，则为金字塔数据集
SmPyramidLevel	INTEGER	N	金字塔层 ID； 如果大于 0，则为对应的金字塔层
SmCreator	TEXT	N	创建者
SmCreateTime	DATE	N	创建时间
SmNoValue	REAL	Y	无值
SmPalette	BLOB	Y	调色板，即颜色对照表，可视化时使用。存储为 Color 数组，见 4.1.10。

表 17 Block 压缩编码方式

枚举值	含义	精度损失
0	不使用压缩编码	无损
8	DCT，离散余弦压缩	有损
9	SGL，游程编码压缩	无损
11	LZW 压缩	无损
12	PNG 压缩	无损

表 18 像素格式

枚举值	含义
1	MONO，单值
4	4 位数值
8	8 位无符号
80	8 位有符号
16	16 位有符号
160	16 位无符号

24	24 位真彩色
32	32 位增强真彩色
320	32 位有符号整型
321	32 位无符号整型
64	64 位有符号长整型
3200	32 位浮点型
6400	64 位双精度浮点型

### 3. 数据表

数据表用于存储实际数据，每个数据集对应一张或多张表格，按系统表的管理范围，划分为矢量数据集和栅格数据集两大类。

#### 3.1 矢量数据集

矢量数据集包括：属性数据集、二维/三维的点/线/面数据集、文本数据集、CAD 数据集、二维/三维网络数据集、三维模型数据集。

矢量数据集的空间索引，用 SQLite 的空间索引机制，相关描述可参见其官方文档。

##### 3.1.1 属性数据集

属性数据集不存储空间数据，数据表对应的系统字段见表 19。

表 19 属性数据集系统字段

字段名	字段类型	是否允许空值	描述
SmID	INTEGER	N	主键；对象的唯一标识
SmUserID	INTEGER	Y	用户自定义 ID 值

##### 3.1.2 二维/三维点数据集

二维点数据集和三维点数据集系统字段相同，见表 20。其中，SmGeometry 字段存储点对象的类型由 geometry\_columns 的 geometry\_type 字段决定，见表 5。

表 20 点数据集系统字段

字段名	字段类型	是否允许空值	描述
SmID	INTEGER	N	主键；对象的唯一标识
SmUserID	INTEGER	Y	用户自定义 ID 值
SmGeometry	POINT	N	存储 GAIAPoint 或 GAIAPointZ 对象， 存储结构见 4.2.1 和 4.2.2

### 3.1.3 二维/三维线数据集

二维线数据集和三维线数据集系统字段相同，见表 21。其中，SmGeometry 字段存储线对象的具体类型由 geometry\_columns 的 geometry\_type 字段决定，见表 5。

表 21 线数据集系统字段

字段名	字段类型	是否允许空值	描述
SmID	INTEGER	N	主键；对象的唯一标识
SmUserID	INTEGER	Y	用户自定义 ID 值
SmLength	REAL	N	线对象的长度；单位为米
SmTopoError	INTEGER	N	拓扑容限； 线数据集拓扑处理时使用
SmGeometry	MULTILINESTRING	N	存储 GAIAMultiLineString 或 GAIAMultiLineStringZ 对象，存储结构见 4.2.3 和 4.2.4

### 3.1.4 二维/三维面数据集

二维面数据集和三维面数据集系统字段相同，见表 22。其中，SmGeometry 字段存储面对象的具体类型由 geometry\_columns 的 geometry\_type 字段决定，见表 5。

表 22 面数据集系统字段

字段名	字段类型	是否允许空值	描述
SmID	INTEGER	N	主键；对象的唯一标识
SmUserID	INTEGER	Y	用户自定义 ID 值
SmArea	REAL	N	面对象的面积；单位为平方米
SmPerimeter	REAL	N	面对象的周长；单位为米
SmGeometry	MULTIPOLYGON	N	存储 GAIAMultiPolygon 或 GAIAMultiPolygon Z 对 象，存储结构见 4.2.6 和 4.2.7 小节。

### 3.1.5 文本数据集

文本数据集系统字段见表 23，一个文本数据集对应一张数据表。

表 23 文本数据集系统字段

字段名	字段类型	是否允许空值	描述
SmID	INTEGER	N	主键；二维文本对象的唯一标识
SmUserID	INTEGER	Y	用户自定义 ID 值
SmGeometry	BLOB	Y	GeoText 的二进制流数据，见 4.3.8 小节
SmIndexKey	POLYGON	Y	对象的范围，存储为 GAIAPolygon 对象见 4.2.5 小 节；空间索引维护时使用

### 3.1.6 CAD 数据集

CAD 数据集可存储多种几何对象类型, 见 4.3 小节。一个 CAD 数据集对应一张数据表, 见表 24。

表 24 CAD 数据集系统字段

字段名	字段类型	是否允许空值	描述
SmID	INTEGER	N	主键; 对象的唯一标识
SmUserID	INTEGER	Y	用户自定义 ID 值
SmGeoType	INTEGER	N	Geometry 类型, 见表 25
SmGeometry	BLOB	Y	SuperMap 简单对象的二进制流, 见 4.3 小节
SmIndexKey	POLYGON	Y	SuperMap 简单对象的范围, 见 4.2.5 小节, 空间索引维护时使用

表 25 CAD 数据集存储的对象类型

Geometry 类型	枚举值	描述
GeoPoint	1	二维点
GeoLine	3	二维线, 可带子对象
GeoRegion	5	二维面, 可带子对象
GeoText	7	文本, 见 4.4 小节
GeoRect	12	矩形/斜矩形
GeoRectRound	13	圆角矩形
GeoCircle	15	圆
GeoEllipse	20	椭圆
GeoPie	21	扇面
GeoArc	24	圆弧
GeoEllipticArc	25	椭圆弧
GeoCardinal	27	Cardinal 曲线
GeoCurve	28	自由曲线
GeoBSpline	29	B 样曲线
GeoPoint3D	101	三维点
GeoLine3D	103	三维线, 可带子对象
GeoRegion3D	105	三维面, 可带子对象

### 3.1.7 二维/三维网络数据集

二维网络数据集和三维网络数据集存储方式相同, 由主表和子表组成, 主表存储网络数据集的边以及结点连接信息, 子表存储网络数据集的结点, 见表 26 和表 27。

如果是三维网络数据集, 则主表的 SmGeometry 字段存储三维线对象, 子表对应存储三维点对象。

表 26 网络数据集主表的系统字段

字段名	字段类型	是否允许空值	描述
-----	------	--------	----

SmID	INTEGER	N	主键；线对象的唯一标识
SmUserID	INTEGER	Y	用户自定义 ID 值
SmEdgeID	INTEGER	N	边 ID
SmFNode	INTEGER	Y	起点结点 ID
SmTNode	INTEGER	Y	终点结点 ID
SmResistanceA	REAL	Y	正方向阻力
SmResistanceB	REAL	Y	负方向阻力
SmTopoError	INTEGER	N	拓扑容限；线数据集拓扑处理时使用
SmLength	REAL	N	线对象的长度；单位为米
SmGeometry	MULTILINESTRING	N	GAIAMultiLineString 或 GAIAMultiLineStringZ 对象，存储结构见 4.2.3 和 4.2.4

表 27 网络数据集子表的系统字段

字段名	字段类型	是否允许空值	描述
SmID	INTEGER	N	主键； 点对象的唯一标识
SmUserID	INTEGER	Y	用户自定义 ID 值
SmNodeID	INTEGER	Y	结点 ID
SmGeometry	POINT	N	GAIAPoint 或 GAIAPointZ 对象，存储结构见 4.2.1 和 4.2.2

### 3.1.8 三维模型数据集

三维模型数据集存储三维模型对象 GeoModel3D，对象逻辑结构及存储结构 4.3 小节。

一个三维模型数据集对应两张数据表：主表和子表，见表 28 和表 29。主表存储模型的结构信息，子表存储模型关联的实体对象。主表存储的模型结构中记录了模型对象引用的实体对象名字，基于实体对象名字的 64 位 HashCode 编码，存储在子表的 SmHashCode 字段中。

表 28 三维模型数据集主表的系统字段

字段名	字段类型	是否允许空值	描述
SmID	INTEGER	N	主键；模型对象的唯一标识
SmUserID	INTEGER	Y	用户自定义 ID 值
SmGeometry	BLOB	Y	GeoModel3D 的二进制流数据，见 4.5.1 小节
SmIndexKey	POLYGON	Y	模型对象的范围，见 4.2.5 小节； 空间索引维护时使用

表 29 三维模型数据集子表的系统字段

字段名	字段类型	是否允许空值	描述
SmID	INTEGER	N	主键； 模型对象的唯一标识
SmHashCode	INTEGER	Y	唯一约束；



SmGeometry	BLOB	Y	实体对象的二进制流数据，见 4.5.3 小节
------------	------	---	------------------------

## 3.2 栅格数据集

栅格数据集存储二维或三维空间中的场数据，包括 Image 数据集、Grid 数据集、体元栅格（VoxelGrid）数据集和镶嵌数据集。

Image 数据集、Grid 数据集和 VoxelGrid 数据集的数据组织和存储方式相同，

### (1) 原始数据

原始数据存储对应一张数据表，即主表。其存储的基本单元是块（Block），块大小记录在 SmImgRegister 的 SmeBlockSize 字段中，见表 15；每个块代表固定长、宽的矩阵数据块，对应主表中的一行记录，见表 30。

### (2) 金字塔

金字塔数据可以有多层，每层金字塔对应一个子数据集，每个子数据集是与主数据集同类型的栅格数据集，其父子关系由系统表 SmBandRegister 维护，见表 16。

表 30 栅格数据表的字段信息

字段名	字段类型	是否允许空值	描述
SmRow	INTEGER	N	块的行号
SmColumn	INTEGER	N	块的列号
SmBandID	INTEGER	N	栅格数据层 ID。 如果是 Grid 数据集则默认为 0； 如果是 Image 数据集则为波段号； 如果是体元栅格数据集则为层 ID； 联合主键：(SmRow,SmColumn,SmBandID)
SmSize	INTEGER	N	块数据字节流大小
SmBand	LONGBLOB	Y	块数据，存储内容见 4.6 小节

镶嵌数据集采用元数据+原始影像文件的方式管理海量影像数据，见 3.2.4 小节。

### 3.2.1 Image 数据集

Image 数据集用于存储影像数据，可以有一个或多个波段，像素值代表颜色值。单波段 Image 数据集可以存储黑白度的灰度值，或 RGB 合成像素值的彩色值。

影像数据一般比较大，通常采用 DCT（Discrete Cosine Transform）。DCT 为离散余弦编码，是一种广泛应用于图像压缩的变换编码方法，该方法有很高的压缩率和性能，但数据精度会有损失。如果希望保持数据精度，则可以采用 PNG、LZW 压缩编码，见 4.6 小节。

### 3.2.2 Grid 数据集

Grid 数据集存储 DTM（Digital Terrain Model，数字地面模型）数据。像素值可表示地理实体或地理现象，如高程值、土壤类型、土地利用类型、岩层深度等。基于 Grid 数据集

可以进行栅格数据统计、代数运算等以数学分析和图形处理为主的计算。

Grid 数据集可采用的压缩编码方式为 LZW 和 SGL，见 4.6 小节。SGL 是 SuperMap 自定义的一种压缩存储格式，是改进的 LZW (Lempel\_Zir\_Welch) 编码方式，该编码方式不仅可以对重复数据起到压缩作用，还可以对不重复数据进行压缩操作。

### 3.2.3 VoxelGrid 数据集

VoxelGrid 数据集，存储连续、非匀质的三维空间属性场数据，如空中电磁信号场、空气与水体污染场、地下地质属性场等。

对连续的三维场数据重采样或插值成不同层的数据，每一层数据用 Block 方式存储，高度特征值标识记录在 SmBandRegister 表的 SmAltitude 字段，见表 16。

### 3.2.4 镶嵌数据集

镶嵌数据集主表记录在 SmImgRegter 系统表中，每个镶嵌数据集挂接了三个子数据集，分别代表轮廓、边界和裁剪。三个子数据集都为面数据集，表的命名规则为 (tablename 是镶嵌数据集主表名)：

- 轮廓子数据集：tablename\_F
- 边界子数据集：tablename\_B
- 裁剪子数据集：tablename\_C

轮廓子数据集是镶嵌数据集存储和组织影像文件的基础，是一个面数据集，每一个面对象对应单幅影像的地理范围，通过轮廓可以全局浏览影像的分布情况及覆盖情况，见表 31。

表 31 镶嵌数据集的轮廓子数据集表

字段名	字段类型	是否允许空值	描述
SmID	INTEGER	N	主键； 对象 ID
SmUserID	INTEGER	Y	用户自定义 ID 值
SmArea	REAL	N	面对象的面积
SmPerimeter	REAL	N	面对象周长
SmGeometry	MULTIPOLYGON	N	几何对象
SmFileName	TEXT	Y	挂接的文件名字
SmMinPS	REAL	Y	影像文件最小显示分辨率
SmMaxPS	REAL	Y	影像文件最大显示分辨率
SmLowPS	REAL	Y	影像原始分辨率
SmHighPS	REAL	Y	影像金字塔分辨率
SmCategory	INTEGER	Y	是否是原始文件。取值： 1 表示是原始文件 2 表示是概视图
SmPath	TEXT	Y	影像文件路径
SmInfo	TEXT	Y	影像文件统计信息
SmZOrder	INTEGER	Y	预留字段

SmFileHash	TEXT	Y	文件对应的 Hash 值
SmOverviewLevel	INTEGER	Y	金字塔层级，原始文件为 0

边界子数据集存储镶嵌数据集的显示范围，表的字段与二维面数据集相同。

裁剪子数据集存储每幅影像的显示范围，表的字段除了二维面数据集的字段，还包括 FootprintID，INTEGER 类型，与轮廓子数据集的 SmID 字段关联。

## 4. 对象存储结构

本节描述了 UDBX 中各种对象的二进制存储结构，字节序为 Little-Endian，即低位字节排放在内存的低地址端。

### 4.1 基本类型

描述对象存储的结构的基本类型包括：基本数据类型及一些常用的对象类型。

#### 4.1.1 基本数据类型

对象存储结构中涉及的基本数据类型见表 32。

表 32 基本数据类型及描述

类型	字节数	取值范围	描述
byte	1	[0, 255]	单字节
bool	1	0 1	布尔型
int16	2	[-32768, 32767]	短整型
uint16	2	[0, 65535]	无符号短整型
int32	4	[-2147483648, 2147483647]	整型
uint32	4	[0, 4294967295]	无符号整型
int64	8	$[-2^{63}, (2^{63}-1)]$	长整型
uint64	8	$[0, (2^{64}-1)]$	无符号长整型
float	4	$[-3.4 \times 10^{38}, 3.4 \times 10^{38}]$	单精度浮点型
double	8	$[-1.7 \times 10^{308}, 1.7 \times 10^{308}]$	双精度浮点型
wchar	2	--	宽字符类型

#### 4.1.2 String

对象存储结构中涉及的字符串数据类型用 String 对象描述，采用 Unicode 编码，字符集规定为 UTF8，存储结构如下：

```
String {
    int32      length;      //字节数
    byte      str[length]; //数据内容
}
```

### 4.1.3 Point

二维坐标点，用 x、y 表示：

```
Point {           //点坐标对象
    double  x;
    double  y;
}
```

### 4.1.4 PointZ

三维坐标点，用 x、y、z 表示：

```
PointZ {          //点坐标对象
    double  x;
    double  y;
    double  z;
}
```

### 4.1.5 Rect

矩形，用左上角点和右下角点表示：

```
Rect {           //矩形
    Point pntLB;   //左下角点
    Point pntRT;   //右上角点
}
```

### 4.1.6 BoundingBox

包围盒，用最大、最小向量表示

```
BoundingBox {
    PointZ boxMax;   //包围盒最大向量
    PointZ boxMin;   //包围盒最小向量
}
```

### 4.1.7 Ring

环形，由点组成首尾相连的环状。

```
Ring {           //由点组成的环形
    int32         numPoints;   //点个数
    Point[]        pnts[numPoints]; //点坐标
}
```

### 4.1.8 RingZ

环形，由三维点组成首尾相连的环状。

```
RingZ {
    int32      numPoints;      //由点组成的环形
    PointZ[]   pnts[numPoints]; //点个数
    //点坐标
}
```

### 4.1.9 Vector3D

三维向量，存储结构同 PointZ，见 4.1.4。

### 4.1.10 Color

颜色，由 rgba 四个分量组成的 uint32 值。

```
Color {
    byte      a;      // alpha 值
    byte      b;      // blue 值
    byte      g;      // green 值
    byte      r;      // red 值
}
```

## 4.2 SpatialLite 的简单对象

SpatialLite 的简单对象类型即二维/三维的点、线、面类型。

GAIAInfo 是 SpatialLite 各类简单对象存储的头部信息，存储结构如下：

```
GAIAInfo {
    //几何对象的基本信息
    static byte   byteOrdering = 1; //字节序：小端存储
    int32         srid;             //坐标系 ID
    Rect          mbr;             //对象的坐标范围
    static byte   gaiaMBR=0x7c;    //MBR 结束标识
}
```

### 4.2.1 GAIAPoint

SpatialLite 的二维点对象：

```
GAIAPoint {
    static byte   gaiaStart = 0x00; //二进制流开始标记
    GAIAInfo      info;           //几何对象的基本信息
    static int32   geoType = 1;    //Geometry 类型标识
    Point          geoPnt;         //点对象的坐标值
    static byte   gaiaEnd = 0xFE;  //二进制流结束标记
}
```

## 4.2.2 GAIAPointZ

SpatiaLite 的三维点对象:

```
GAIAPointZ {
    static byte    gaiaStart = 0x00;    //二进制流开始标记
    GAIAGeoInfo    info;                //几何对象的基本信息
    static int32    geoType = 1001;     //Geometry 类型标识
    PointZ          geoPntZ;            //点对象的坐标值
    static byte    gaiaEnd = 0xFE;      //二进制流结束标记
}
```

## 4.2.3 GAIAMultiLineString

SpatiaLite 的二维多线对象:

```
GAIAMultiLineString {
    static byte    gaiaStart = 0x00;    //二进制流开始标记
    GAIAGeoInfo    info;                //几何对象的基本信息
    static int32    geoType = 5;        //Geometry 类型标识
    int32          numLineStrings;      //子对象个数
    LineStringEntity[] lineStrings[numLineStrings]; // LineString 的几何数据
    static byte    gaiaEnd = 0xFE;      //二进制流结束标记
}

LineStringEntity {
    static byte    gaiaEntityMark = 0x69; //子对象标识
    static int32    geoType = 2;          //Geometry 类型标识
    int32          numPoints;             //点个数
    Point[]         pnts[numPoints];      //每个点的坐标值
}
```

## 4.2.4 GAIAMultiLineStringZ

SpatiaLite 的三维多线对象:

```
GAIAMultiLineStringZ {
    static byte    gaiaStart = 0x00;    //二进制流开始标记
    GAIAGeoInfo    info;                //几何对象的基本信息
    static int32    geoType = 1005;     //Geometry 类型标识
    int32          numLineStrings;      //子对象个数
    LineStringZEntity[] lineStrings[numLineStrings]; // LineString 的几何数据
    static byte    gaiaEnd = 0xFE;      //二进制流结束标记
}

LineStringZEntity {
    static byte    gaiaEntityMark = 0x69; //子对象标识
    static int32    geoType = 1002;      //Geometry 类型标识
}
```

```

        int32          numPoints;          //点个数
        PointZ[]      pnts[numPoints];    //每个点的坐标值
    }

```

## 4.2.5 GAIAPolygon

SpatiaLite 的二维面对象：

```

GAIAPolygon {
    static byte    gaiaStart = 0x00;      //二进制流开始标记
    GAIAGeoInfo    info;                  //几何对象的基本信息
    PolygonData    data;                  //Polygon 的几何数据
    static byte    gaiaEnd = 0xFE;        //二进制流结束标记
}

PolygonData {                             //Polygon 的几何数据
    static int32   geoType = 3;           //Geometry 类型标识
    int32          numInteriors;          //内环个数
    Ring           exteriorRing;          //外环对象
    Ring[]         interiorRings[numInteriors]; //内环对象
}

```

## 4.2.6 GAIAMultiPolygon

SpatiaLite 的二维多面对象：

```

GAIAMultiPolygon {
    static byte    gaiaStart = 0x00;      //二进制流开始标记
    GAIAGeoInfo    info;                  //几何对象的基本信息
    static int32   geoType = 6;           //Geometry 类型标识
    int32          numPolygon;            //子对象个数
    PolygonEntity[] polygons[numPolygon]; //子对象数据
}

PolygonEntity {
    static byte    gaiaEntityMark = 0x69; //子对象标识
    PolygonData    data;                  //子对象数据
}

```

## 4.2.7 GAIAMultiPolygonZ

SpatiaLite 的三维面对象：

```

GAIAMultiPolygonZ {
    static byte    gaiaStart = 0x00;      //二进制流开始标记
    GAIAGeoInfo    info;                  //几何对象的基本信息
}

```

```

        static int32      geoType = 1006;           //Geometry 类型标识
        int32            numPolygon;               //子对象个数
        PolygonEntity[]  polygons[numPolygon];     //子对象数据
    }

    PolygonEntity {
        static byte      gaiaEntityMark = 0x69;     //子对象标识
        PolygonZData     data;                     //子对象数据
    }

    PolygonZData {                                //PolygonZ 的几何数据
        static int32     geoType = 1003;           //Geometry 类型标识
        int32            numInteriors;              //内环个数
        RingZ            exteriorRing;              //外环对象
        RingZ[]          interiorRings[numInteriors]; //内环对象
    }

```

## 4.3 CAD 数据集中存储的对象

CAD 数据集可以存储二维/三维点/线/面和参数化等空间对象（见表 25），还可以存储文本对象（见 4.4 小节）。与其它数据集不同，CAD 数据集可以存储对象风格，以 GeoHeader 的形式存储在对象头部，结构如下：

```

GeoHeader {
    int32      geoType;           //对象类型，见表 25
    int32      styleSize;         //对象风格占用字节数
    Style      style;             //风格内容，见 4.3.1 小节
}

```

### 4.3.1 Style

Style 按对象的维度，分为点符号、线符号和面填充风格（符号的 ID 需要配合 SuperMap 的符号库使用）。

#### 4.3.1.1 StyleMarker

点符号。

```

StyleMarker {
    int32      length;           //字节流长度
    int32      markerStyle;      //点符号在符号库中的 ID
    int32      markerSize;       //符号大小，精度 0.1mm
    int32      markerAngle;      //符号旋转角度，单位 0.1 度
    Color      markerColor;      //符号颜色
    int32      markerwidth;      //符号宽度
    int32      markerHeight;     //符号高度
}

```



```

byte    reservedLength;           //预留字节的长度
byte[]   reservedData[reservedLength+4]; //预留数据
byte    fillOpaqueRate;          //填充透明度
byte    fillGradientType;        //渐变填充类型
int16    fillAngle;              //填充角度
int16    fillCenterOffsetX;       //填充中心点水平偏移百分比
int16    fillCenterOffsetY;       //填充中心点垂直偏移百分比
Color    fillBackcolor;          //填充背景色
}

```

#### 4.3.1.2 StyleLine

线符号。

```

StyleLine {
    int32    lineStyle;           //线符号在符号库中 ID 号
    int32    lineWidth;          //线宽, 单位 0.1mm
    Color    lineColor;          //线颜色
    byte    reservedLength;       //预留字节的长度
    byte[]   reservedData[reservedLength+4]; //预留数据
}

```

#### 4.3.1.3 StyleFill

面填充风格。

```

StyleFill {
    int32    lineStyle;           //线符号在符号库中的 ID
    int32    lineWidth;          //线宽, 精度 0.1mm
    Color    lineColor;          //线颜色
    int32    fillStyle;          //填充符号在符号库中的 ID
    Color    fillForecolor;       //填充前景色
    Color    fillBackcolor;       //填充背景色
    byte    fillOpaquerate;       //填充透明度
    byte    fillGradientType;     //渐变填充类型
    int16    fillAngle;           //填充角度, 精度 0.1°
    int16    fillCenterOffsetX;    //填充中心点水平偏移百分比
    int16    fillCenterOffsetY;    //填充中心点垂直偏移百分比
    byte    reserved1Length;       //预留字节的长度
    byte[]   reserved1Data[reserved1Length+4]; //预留数据
    byte    reserved2Length;       //预留字节的长度
    byte[]   reserved2Data[reserved2Length+4]; //预留数据
}

```

### 4.3.2 GeoPoint

二维点对象。

```
GeoPoint {
    GeoHeader    header;
    Point        pnt;        // 点坐标值
}
```

### 4.3.3 GeoLine

二维线对象。

```
GeoLine {
    GeoHeader    header;
    uint32       numSub;           //子对象个数
    int32        subPointCount[numSub]; //每个子对象点个数
    Point[]      pnts[allPntCount]; //allPntCount 为所有子对象点坐标个数之和
}
```

### 4.3.4 GeoRegion

二维面对象。

```
GeoRegion {
    GeoHeader    header;
    uint32       numSub;           //子对象个数
    int32        subPointCount[numSub]; //每个子对象点个数
    Point[]      pnts[allPntCount]; //allPntCount 为所有子对象点坐标个数之和
}
```

### 4.3.5 GeoPoint3D

三维点。

```
GeoPoint3D {
    GeoHeader    header;
    PointZ       pnt;        //点坐标
}
```

### 4.3.6 GeoLine3D

三维线。

```
GeoLine3D {
    GeoHeader    header;
```

```

        uint32      numSub;                //子对象个数
        int32       subPointCount[numSub]; //每个子对象点个数
        PointZ[]    pnts[allPntCount];    //allPntCount 为所有子对象点坐标个数
        之和
    }

```

### 4.3.7 GeoRegion3D

三维面。

```

GeoRegion3D {
    GeoHeader header;
    uint32      numSub;                //子对象个数
    int32       subPointCount[numSub]; //每个子对象点个数
    PointZ[]    pnts[allPntCount];    //allPntCount 为所有子对象点坐标个数之和
}

```

### 4.3.8 GeoRect

矩形对象。

```

GeoRect {
    GeoHeader header;
    Point      pntCenter;    //中心点坐标值
    double     width;        //宽度
    double     height;       //高度
    int32      angle;        //旋转角度乘 10 的四舍五入整型值
    static int32 reserved=0;  //预留
}

```

### 4.3.9 GeoRectRound

圆角矩形。

```

GeoRectRound {
    GeoHeader header;
    Point      pntCenter;    //中心点坐标值
    double     width;        //宽度
    double     height;       //高度
    int32      angle;        //旋转角度乘 10 的四舍五入整型值
    static int32 reserved=0;  //预留
    double     radiusX;      //圆角长半轴
    double     radiusY;      //圆角短半轴
}

```

### 4.3.10 GeoCircle

圆。

```
GeoCircle {
    GeoHeader header;
    Point      pntCenter;           //中心点坐标值
    double     radius;              //半径
}
```

### 4.3.11 GeoEllipse

椭圆。

```
GeoEllipse {
    GeoHeader header;
    Point      pntCenter;           //中心点坐标值
    double     semimajoraxis;       //长半轴
    double     semiminoraxis;      //短半轴
    int32      angle;              //旋转角度乘 10 的四舍五入整型值
    static int32 reserved=0;       //预留
}
```

### 4.3.12 GeoPie

扇面。

```
GeoPie {
    GeoHeader header;
    Point      pntCenter;           //中心点坐标值
    double     semimajoraxis;       //长半轴
    double     semiminoraxis;      //短半轴
    int32      rotationangle;       //旋转角度乘 10 的四舍五入整型值
    int32      startangle;          //起始角度乘 10 的四舍五入整型值
    int32      endangle;            //终止角度乘 10 的四舍五入整型值
    static int32 reserved=0;       //预留
}
```

### 4.3.13 GeoArc

圆弧。

```
GeoArc {
    GeoHeader header;
    Point      pntStart;            //起始点坐标值
    Point      pntMiddle;          //中间点坐标值
    Point      pntEnd;             //终止点坐标值
}
```

```
}
```

#### 4.3.14 GeoEllipticArc

椭圆弧。

```
GeoEllipticArc {
    GeoHeader header;
    Point pntCenter; //中心点坐标值
    double semimajoraxis; //长半轴
    double semiminoraxis; //短半轴
    int32 rotationangle; //旋转角度乘 10 的四舍五入整型值
    int32 startangle; //起始角度乘 10 的四舍五入整型值
    int32 endangle; //终止角度乘 10 的四舍五入整型值
    static int32 reserved=0; //预留
}
```

#### 4.3.15 曲线

曲线包括 Cardinal 曲线、自由曲线和 B 样条曲线。其存储结构相同：

```
CurveObject {
    GeoHeader header;
    uint32 numPnts; //曲线控制点的个数
    Point[] pnts[numPnts]; //曲线控制点的坐标
}
```

### 4.4 文本对象

文本对象可以存储在文本数据集或者 CAD 数据集中。如果存储在文本数据集中，则 GeoHeader 的 styleSize 为 0。

```
GeoText {
    GeoHeader header; //对象头部，见 4.3 小节
    int32 subCount; //文本子对象个数
    TextStyle textStyle; //文本风格
    GeoSubText subTexts[subCount]; //文本子对象
}

GeoSubText {
    Point pntAnchor; //定位点
    int32 subAngle; //旋转角度，实际旋转角度乘 10 的四舍五入整型值
    static int32 reserved = 0; //预留
    String subText; // 文本子对象的文本内容
}
```

```

TextStyle {
    Color          color;          //文本颜色
    TextStyleBit   textStyleBit;   //文本按位风格
    Color          bgColor;        //文本背景颜色
    double         fontWidth;      //文本字体宽度
    double         fontHeight;     //文本字体高度
    Point          pntAnchor;      //文本定位点
    String         faceName;       //文本字体名称
}

```

```

TextStyleBit {
    byte          fixedSize;       //固定大小
    byte          weight;          //笔画宽度
    byte          styleFlag;       //粗体斜体下划线等
    byte          alignFlag;       //文本对齐方式
}

```

其中，styleFlag 用 8 位表示是否带有某种风格标识，从低位到高位分别表示：阴影、轮廓线、背景不透明、固定大小、删除线、下划线、斜体、粗体；

alignFlag 分为两部分，高 4 位预留，低位 4 位表示字体对齐方式，取值含义：

- 0：左上； 1：中上； 2：右上；
- 6：左下； 7：中下； 8：右下；
- 9：左中； 10：中中； 11：右中。

## 4.5 三维模型对象

三维模型对象（GeoModel3D）由带局部坐标系的模型对象（ModelNode）及其放置的位置、姿态等信息组成，其组织结构见图 2。

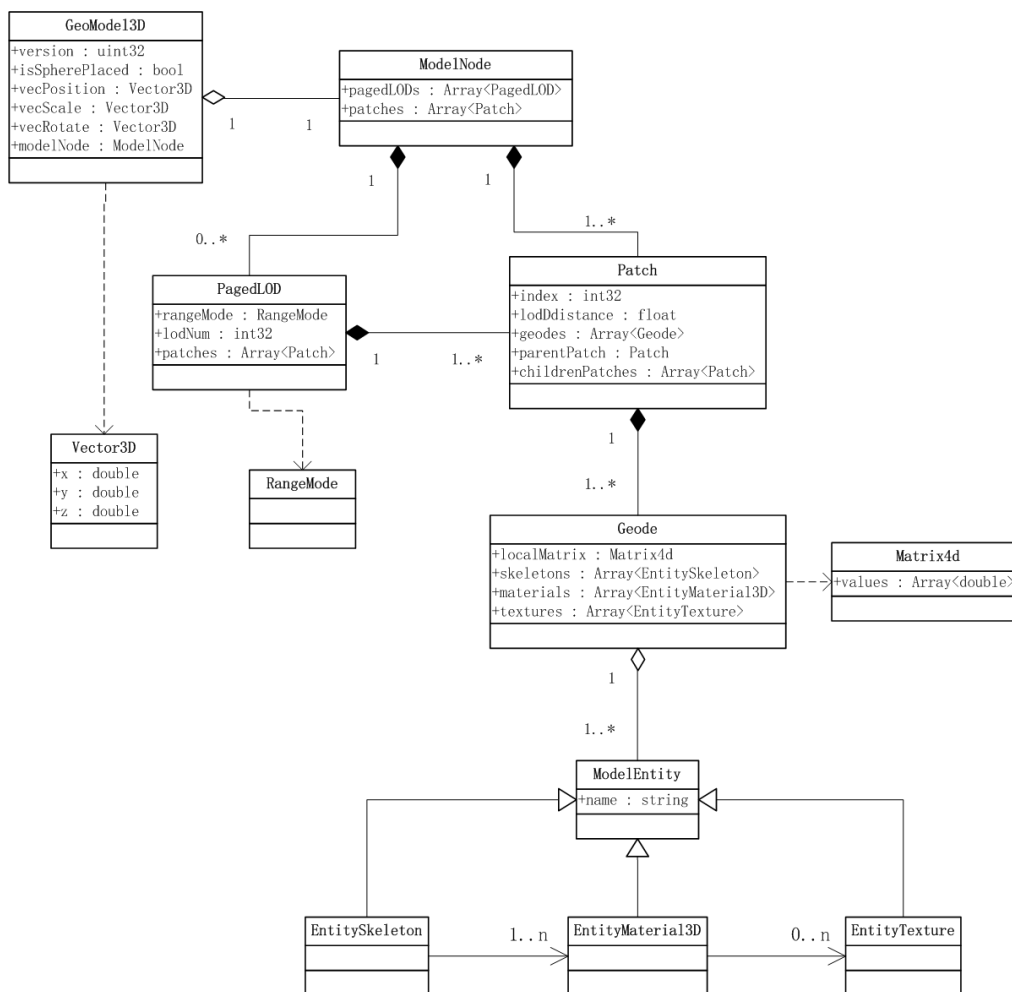


图 2 GeoModel3D 对象组织结构

ModelNode 由精细层和 LOD 层（用 PagedLOD 表示，可选）数据组成，精细层和 LOD 层的基本组成单元均为 Patch；每个 Patch 包含多个 Geode；Geode 是一个数据包，由实体对象（用 ModelEntity）组成，通过 Geode 上的矩阵，可以把相同的实体放在不同的位置，实现模型数据的实例化存储。

ModelEntity 的子类包括骨架（EntitySkeleton）、材质（EntityMaterial3D）和纹理（EntityTexture）。

在存储策略上，GeoModel3D 存储在主表中，Geode 仅存储实体对象的名字；实体对象单独存储在数据集子表中，基于实体对象名字的 64 位 HashCode 编码作为对象的 ID。

### 4.5.1 GeoModel3D

```

GeoModel3D {
    static int32    type=1218;           //对象类型
    static int32    hasStyle=0;         //是否有风格
    uint32          version;            //对象版本号
    bool            isSpherePlaced;     //是否放置到球面
    Vector3D        vecPosition;        //对象位置
    Vector3D        vecScale;           //对象缩放值
    Vector3D        vecRotate;          //对象旋转值

```

```

        BoundingBox bbox;                //包围盒
        ModelNode modelNode;
    }

```

## 4.5.2 ModelNode

```

ModelNode {
    int32      numLODs;                //LOD 层数
    PagedLOD   pagedLODs[numLODs];    // LOD 层数据
    int32      numPatches;             //精细层 Patch 个数
    Patch      patches[numPatches];    //精细层 Patch 数据
}

PagedLOD {
    RangeMode  rangeMode;              //切换范围模式, 存储为 int16
    int32      lodNum;                 //LOD 层号
    int32      numPatches;             //本层的 Patche 个数
    Patch      patches[numPatches];    //Patch 数据
}

Enum RangeMode {                      // LOD 切换模式
    DISTANCE_FROM_EYE_POINT = 0,      // 根据到相机的距离切换
    PIXEL_SIZE_ON_SCREEN = 1          // 根据屏幕像素大小切换
}

Patch {
    float      lodDdistance;            //切换距离
    int32      index;                  //当前数据层 Patch 的索引号
    int32      parentIndex;            //父节点索引号,-1 时代表没有父节点
    int32      numChildren;            //子节点个数
    int32      childrenIndexes[numChildren]; //子节点索引号
    int32      geodeCount;             //Geode 个数
    Geode      geodes[geodeCount];     //各 Geode 数据
}

Geode {
    Matrix4d   localMatrix;            //矩阵信息
    int32      numSkeletons;            //骨架个数
    String     skeletonNames[numSkeletons] //骨架名字
    int32      numMaterials;           //材质个数
    String     materialNames[numMaterials] //材质名字
    int32      numTextures;            //纹理个数
    String     textureNames[numTextures] //纹理名字
}

```



```
Matrix4d {           //4*4 矩阵，行主序
    double values[16];
}
```

### 4.5.3 ModelEntity

#### 4.5.3.1 EntitySkeleton

```
EntitySkeleton {
    String          name;           //骨架名
    String          materialName;   //关联的材质名
    BoundingBox     bbox;           //包围盒
    Matrix4d        localMatrix;    //模型矩阵
    VertexDataPackage dataPack;     //顶点数据
    int32           numIndexpacks;   //索引包个数
    IndexPackage     indexPacks[numIndexpacks]; //索引包数组
}

VertexDataPackage {
    //顶点属性，与 VertexOptions 中的枚举值按位运算得出
    int32    vertexOptions;
    uint16   numDim;        //顶点坐标维度
    uint32   numVertexes;   //顶点个数
    uint16   vertexStride;  //顶点坐标在数组中的偏移量
    //顶点坐标数据。当 vertexOptions 具备 VO_VERTEX_DOUBLE 属性则 double，否则为 float
    variant  vertexData[numVertexes * numDim];
    uint32   numNormals;    //法向量个数
    uint16   normalStride   //法向量在数组中的偏移
    float    normalData[numNormals * numDim]; //法向量数据
    uint32   numColors;     //顶点颜色个数
    uint16   colorStride;   //颜色在数组中的偏移
    uint32   colorData[colorCount]; //顶点颜色，4 字节存储 R/G/B/A
    int32    numTextures;   //纹理通道个数
    TextureCoord textureCoords[numTextures]; //纹理坐标数据
}

Enum VertexOptions {           //顶点数据的属性
    VO_NORMALS = 1,            //包含法线
    VO_TEXTURE_COORDS = 2,     //包含纹理坐标
    VO_DIFFUSE_COLOURS = 4,     //包含顶点颜色
    VO_SPECULAR_COLOURS = 8,    //包含顶点 secondColor
}
```

```

VO_BLEND_WEIGHTS = 16,           //使用权重值计算
VO_USE_SINGLE_COLOR = 32,        //仅采用一种颜色绘制
VO_USE_POINT_SMOOTHING = 64,     //启动点反走样
VO_MATERIAL = 128,               //使用材质
VO_TEXTURE_COLOR = 256,          //使用纹理颜色
VO_VERTEX_DOUBLE = 512,          //顶点坐标为高精度 double
VO_TEXTURE_COORD_Z_IS_MATRIX = 1024, //表示顶点属性的 Z 值是一个矩阵
};

```

```

TextureCoord {                    //纹理坐标
    uint16    dimension;           //纹理坐标维度
    uint32    numCoords;           //纹理坐标个数
    uint16    stride;              //偏移值
    float     coordData[numCoords * dimension]; //坐标值
}

```

```

IndexPackage {
    uint32      numIndexes;         //索引个数
    IndexType    type;              //索引数据类型, 存储为 int32
    bool         isUseIndex;        //是否使用索引
    OperationType operationType;     //顶点的组织方式, 存储为 int32
    //索引数据, 当 type 为 IT_32BIT 或者 IT_32BIT_2 时, variant 为 uint32; 否则为
    uint16
    variant      indexData[indexesCount];
    int32        numPass;           //使用的 Pass 个数
    String       passNames[numPass]; //使用的 Pass 的名称数组
}

```

```

Enum IndexType {
    IT_16BIT = 0,           //索引值采用 uint16 表示
    IT_32BIT = 1,           //索引值采用 uint32 表示
    IT_16BIT_2 = 2,         //带属性索引, 索引值采用 uint16 表示
    IT_32BIT_2 = 3,         //带属性索引, 索引值采用 uint32 表示
}

```

```

Enum OperationType {            //顶点的组织方式
    OT_POINT_LIST = 1,          //单个点
    OT_LINE_LIST = 2,           //两点线
    OT_LINE_STRIP = 3,          //线串
    OT_TRIANGLE_LIST = 4,       //三角形
    OT_TRIANGLE_STRIP = 5,      //条带三角形
    OT_TRIANGLE_FAN = 6,        //扇面三角形构成
    OT_QUAD_STRIP = 8,          //条带四边形
    OT_QUAD_LIST = 9,           //四边形串, 不共享边
}

```

```

        OT_POLYGON = 10,                //多边形
    }

```

### 4.5.3.2 EntityMaterial3D

```

EntityMaterial3D {
    double    version;                //版本号
    String    name;                   //材质名
    String    groupName;              //材质所在组名
    EffectType effectType;            //特效材质类型，存储为 int32
    int32     numTechnique;           //Technique 个数
    Technique techniques[numTechnique]; //Technique 数据
}

Enum EffectType {                    //特效材质枚举
    NONE = 0,                        //无特效
    WATER = 1,                      //水面特效
}

Technique {
    String    name;                   //Technique 名字
    String    schemeName;            //Technique 所属的 scheme 名字
    String    lodIndex;              //Technique 所使用的 LOD 层索引
    String    mShadowCasterMaterialName; //阴影投射的材质名字
    String    mShadowReceiverMaterialName; //阴影接收的材质名字
    int32     numPass;               //pass 个数
    Pass      passes[numPass];       //绑定的所有 pass
}

Pass {
    String    name;                   //pass 名字
    PolygonMode polygonMode;          //绘制模式，存储为 int32
    CullingMode cullMode;             //裁剪模式，存储为 int32
    bool       lightEnabled;          //设置光照是否开启
    uint32     reserved;              //未使用
    bool       reserved;              //未使用
    float      pointSize;             //点尺寸大小
    float      pointMinSize;          //点最小尺寸
    float      pointMaxSize;          //点最大尺寸
    int16      reserved;              //未使用
    double     reserved [3];          //未使用
    SmoothHintMode pntSmoothHintMode; //线平滑方式，存储为 int32
    SmoothHintMode lineSmoothHintMode; //点平滑方式，存储为 int32
    uint32     ambient;               //环境光
}

```

```

uint32 diffuse; //散射光
uint32 specular; //反射光
uint32 selfIllumination; //自发光
uint32 materialColor; //材质颜色
float shininess; //发光, 影响发射光点的大小
uint32 tracking; //顶点颜色跟踪
bool receiveShadow; //是否接收阴影
bool colorWrite; //颜色是否能够写入
float alphaReject; //Alpha 测试参考值
CompareFunction alphaRejectFunc; //Alpha 测试方法, 存储为 int32
bool reserved; //未使用
bool transparentSorting; //透明物体深度排序
bool reserved; //未使用
bool depthCheck; //是否进行深度测试
bool depthWrite; //渲染时是否进行深度写入
CompareFunction depthBufferFunc; //深度测试方法, 存储为 int32
float constantPolygonOffset; //多边形偏移量常量部分
float slopeScalePolygonOffset; //多边形偏移量深度坡度因子部分
float reserved; //未使用
bool blendAlpha; //是否进行 Alpha 混合
String vertexProgram; //顶点着色器的名字
String fragmentProgram; //片元着色器的名字
String geometryProgram; //几何着色器的名字
String shadowCasterVertexProgram; //阴影投射顶点着色器的名字
String shadowReceiverVertexProgram; //阴影接收顶点着色器的名字
String shadowReceiverFragmentProgram; //阴影接收片元着色器的名字
int32 numTextureUnitState; //纹理单元个数
TextureUnitState textureUnitStates[numTextureUnitState]; //关联的纹理单元
int32 textureZType[numTextureUnitState]; //各纹理 Z 通道
}

```

```

Enum PolygonMode { //渲染引擎用的多边形显示模式
    PM_POINTS = 1, //仅显示点
    PM_WIREFRAME = 2, //仅显示线框
    PM_SOLID = 3 //显示实体
}

```

```

Enum CullingMode { //渲染引擎用的的裁剪模式
    CULL_NONE = 1, //不进行裁剪
    CULL_CLOCKWISE = 2, //顺时针方向被裁剪
    CULL_ANTICLOCKWISE = 3 //逆时针方向被裁减
}

```

```

Enum SmoothHintMode { //图像绘制的反走样模式

```

```

    SHM_NONE = 0,                //不使用抗锯齿
    SHM_DONT_CARE = 1,           //由 OpenGL 决定达到点/线的平滑效果
    SHM_FASTEST = 2,             //运行速度最快
    SHM_NICEST = 3               //显示效果最好
}

Enum CompareFunction {           //各类测试的比较方式
    CMPF_ALWAYS_FAIL = 0,        //从不通过测试
    CMPF_ALWAYS_PASS = 1,        //总是通过测试
    CMPF_LESS = 2,               //只有参考值<缓冲区标记值时才通过
    CMPF_LESS_EQUAL = 3,         //只有参考值<=缓冲区标记值时才通过
    CMPF_EQUAL = 4,              //只有参考值=缓冲区标记值时才通过
    CMPF_NOT_EQUAL = 5,          //只有参考值!=缓冲区标记值时才通过
    CMPF_GREATER_EQUAL = 6,       //只有参考值>=缓冲区标记值时才通过
    CMPF_GREATER = 7             //只有参考值>缓冲区标记值时才通过
}

TextureUnitState {
    String    name;                //纹理单元状态名字
    String    textureNameAlias;    //纹理别名
    String    textureName;         //纹理单元使用的纹理名称
    String    cubicTextureName;    //立方体纹理名
    uint32    reserved;            //未使用
    TextureAddressingMode modeU;    //纹理坐标寻址模式 U 方向, 存储为 int32
    TextureAddressingMode modeV;    //纹理坐标寻址模式 V 方向, 存储为 int32
    TextureAddressingMode modeW;    //纹理坐标寻址模式 W 方向, 存储为 int32
    FilterOptions minFilter;        //缩小时的滤波类型, 存储为 int32
    FilterOptions maxFilter;        //放大时的滤波类型, 字节 int32
    FilterOptions mipFilter;        //Mipmap 时滤波类型, 字节 int32
    double     UScale;              //纹理 U 的缩放
    double     VScale;              //纹理 V 的缩放
    bool        EnvironmentMapEnabled; // 是否启用环境映射
    int32        reserved;          //未使用
    Matrix4d     texModMatrix;      //纹理矩阵
}

Enum TextureAddressingMode {      //纹理寻址模式
    TAM_WRAP,        //重复贴图
    TAM_MIRROR,       //对称翻转
    TAM_CLAMP,        //边缘像素填充所有大于 1 的纹理坐标, 边缘拉长
    TAM_BORDER,       //不在[0,1]范围内的纹理坐标使用用户指定的边缘颜色
}

Enum FilterOptions {              //纹理或者 mipmap 的滤波模式

```

```

    FO_NONE = 0,           //无过滤
    FO_POINT = 1,          //临近采样
    FO_LINEAR = 2,         //线性采样
    FO_TRILINEAR = 3,      //三线性采样
    FO_ANISOTROPIC = 4     //类似线性采样，考虑纹理角度，各向异性，未使用
}

```

### 4.5.3.3 EntityTexture

```

EntityTexture {
    String name;           //纹理名字
    bool mipmap;           //是否带 mipmap
    int32 level;           //mipmap 层级
    TextureData textureData; //纹理数据
}

TextureData {
    static uint32 compressType=14; //纹理压缩类型，存储为 uint32
    uint32 width;                 //纹理宽度
    uint32 height;                //纹理高度
    PixelFormat format;           //纹理像素格式，存储为 int32
    uint32 size;                  //数据流字节长度
    int32 zipSize;                //zip 压缩后大小
    uchar data[zipSize]          //zip 压缩后数据
}

Enum PixelFormat {           //纹理像素格式
    PF_BYTE_RGB = 11,         //3 字节像素，每个颜色占一个字节
    PF_BYTE_BGR = 10,         //3 字节像素，每个颜色占一个字节
    PF_BYTE_BGRA = 12,        //4 字节像素，每个颜色和 alpha 各占一个字节
    PF_BYTE_RGBA = 13,        //4 字节像素，每个颜色和 alpha 各占一个字节
}

```

## 4.6 栅格块存储

栅格块 (Block) 存储与压缩编码方式 (见表 17) 相关。Block 的二进制流大小写入数据表的 SmSize 字段，Block 数据写入 SmBand，见表 30。

### 1) 无压缩编码

Block 按实际长、宽逐像素存储值，行主序。像素值的类型及存储方式见表 18。

### 2) DCT 压缩

采用 jpeglib 开源库 (版本号 6b) 对原始 Block 数据流进行压缩。

### 3) SGL 压缩

按 X 方向 16 个像素, Y 方向 4 个像素对 Block 划分 N 个 Tile, 每个 Tile 内部进行游程编码。

```
SGLBlock {
    int16      numTiles;          //Tile 个数
    static byte sizeX = 16;       //Tile 在 X 方向像素数
    static byte sizeY = 4;       //Tile 在 X 方向像素数
    SGLTile[]  tileData[numTiles]; //每个 Tile 压缩后的数据
}
SGLTile {
    int16      dataLenth;         //数据流长度, 字节单位
    byte       bitCountDiff;      //编码值中, Diff 值占用位数
    byte       byteCountMin;      //Tile 的最小值占用字节数
    byte[]     minValueByByte[byteCountMin]; //Tile 的最小值
    ValuePack[] valuePacks[]; //游程编码值
}
ValuePack {
    byte       tagCount;          //值的个数。如果该值最高位为 1, 则表示无值, 低四位为无值个数, 没有 values 值; 如果该值第 3 位为 1, 则表示 values 为不重复的值, 个数为低两位取整
    byte[]     values[bitCountDiff]; //原值与最小值的差按占用字节数写入
}
```

### 4) LZW 压缩

目前是采用 zlib 开源库 (版本号 1.2.2) 对原始 Block 数据流进行压缩。

### 5) PNG 压缩

采用 libpng 开源库 (版本号 1.2.6) 对原始 Block 数据流进行压缩。

## 4.7 其它对象

### 4.7.1 坐标系对象

坐标系对象的存储结构如下:

```
ProjectInfo {
    int32      prjCoordSysType;    //投影坐标系类型
    int32      geoCoordSysType;    //地理坐标系类型
    int32      projectionType;      //投影方式类型
    int32      datumType;          //datum 类型
    int32      spheroidType;        //椭球体类型
    int32      primeMeridianType;  //中央子午线类型
}
```

```

int32      reserved;           //预留
int32      unit;              //坐标系单位
double     falseEasting;      //水平偏移量
double     falseNorthing;     //垂直偏移量
double     centralMeridian;    //中央经线
double     centralParallel;    //原点纬线
double     standardParallel1;  //标准纬线 1
double     standardParallel2;  //标准纬线 2
double     scaleFactor;        //比例因子
double     azimuth;           //方位角
double     firstPointLongitude; //第一点经线
double     secondPointLongitude; //第二点经线
double     axis;              //椭圆长半轴
double     flatten;           //椭圆扁率
double     primeMeridian;      //中央子午线值
double[]    reserved[2];      //预留
String      prjCoordSysName;    // 投影坐标系名称
String      geoCoordSysName;    // 地理坐标系名称
String      spheroidName;       // 椭球体名称
String      datumName;         // datum 名称
uint32      epsgCode;          // EPSG 编号
double      rectifiedAngle;     // 纠正角
}

```

相关描述可参考 OGDC，见 5 小节。其中 prjCoordSysType、geoCoordSysType、projectionType、datumType、spheroidType、primeMeridianType 的枚举值对应的含义可参考 Projection/UGPJCon.h 文件；单位值 unit 可参考 Base/ogdcdefs.h。

## 4.7.2 值域规则对象

值域规则对象分为范围值域 (DomainRangeInfos) 和枚举值域 (DomainCodeInfos)。

```

DomainRangeInfos {
    int32      numRanges;           //范围区间的个数
    DomainRange [] ranges[numRanges]; //各区间对象
}

DomainRange {
    DomainRangeType rangeType; //区间类型
    //区间值类型由字段类型决定：
    //字段类型 Int16、Int32，对应 variant 为 int32；
    //字段类型 Int64，对应 variant 为 int64；
    //字段类型 Float、Double，对应 variant 为 double
    variant     leftValue;         //区间左值
    variant     rightValue;        //区间右值
}

```



```

Enum DomainRangeType {
    CloseClose = 1,    //左值取闭区间，右值取闭区间
    OpenClose  = 2,    //左开，右闭
    CloseOpen  = 3,    //左闭，右开
    OpenOpen   = 4     //左开，右开
}

DomainCodeInfos {
    int32          numCodes;           //枚举值的个数
    DomainCode[]   ranges[numCodes];  //各枚举对象
}

DomainCode {
    //值类型由字段类型决定：
    //字段类型 Boolean、Byte、Int32，对应 variant 为 int32
    //字段类型 Int16，对应 variant 为 int16
    //字段类型 Int64，对应 variant 为 int64
    //字段类型 Float、Double，对应 variant 为 double
    //字段类型 Text、NText，对应 variant 为 String
    Variant      codeValue;           //枚举值
    String       codeDescription;     //枚举值说明
}

```

## 5. 应用指南

超图软件自 SuperMap 9D（2019）开始全面支持 UDBX 格式，包括组件、桌面、移动端和服务端系列产品。

此外，OGDC 也提供了对 UDBX 的支持，基于一套规范的 C++ 接口对 UDBX 数据进行读写操作，链接：<https://github.com/SuperMap/OGDC>。