第18章

其他经验技巧

18.1 ArcEngine 中的先闪烁后刷新现象

在应用程序开发的过程中，有的时候需要对实体对象进行闪烁和刷新，如果采用下面的代码：

```
m_activeView.Refresh();
m_mapControl.FlashShape ( feature );
```

在执行时图形会先闪烁，然后才刷新背景。显然这不是我们想要的结果，原因在于图形显示和闪烁不在同一个线程中执行，刷新背景比较复杂，需要的时间较长，而闪烁图形的动作很快就能完成。而使用 m_activeView.UpdateWindow 后，就可以让代码在此等待，等到刷新确实搞定了，才开始执行 FlashShape。代码如下：

```
m_activeView.Refresh();
m_activeView.UpdateWindow();      //这条语句是关键
m_mapControl.FlashShape(feature);
```

18.2 ArcEngine 中几种数据的删除方法和性能比较

在查询结果中删除数据：

```
private void Deletel(IFeatureClass PFeatureclass)
    {
      IQueryFilter pQueryFilter = new QueryFilterClass();
      pQueryFilter.WhereClause = "objectID<=" + DeleteNum;
      IFeatureCursor pFeatureCursor = PFeatureclass.Search(pQueryFilter, false);
      IFeature pFeature = pFeatureCursor.NextFeature();
      while (pFeature != null)
      {
        pFeature.Delete();
```

```
            pFeature = pFeatureCursor.NextFeature();
          }
          System.Runtime.InteropServices.Marshal.ReleaseComObject(pQueryFilter);
        }
```

更新游标删除：

```
    private void Delete2(IFeatureClass PFeatureclass)
        {
          IQueryFilter pQueryFilter = new QueryFilterClass();
          pQueryFilter.WhereClause = "objectID<=" + DeleteNum;
          IFeatureCursor pFeatureCursor = PFeatureclass.Update(pQueryFilter, false);
          IFeature pFeature = pFeatureCursor.NextFeature();
          while (pFeature != null)
          {
            pFeatureCursor.DeleteFeature();
            pFeature = pFeatureCursor.NextFeature();
          }
          System.Runtime.InteropServices.Marshal.ReleaseComObject(pQueryFilter);
        }
```

使用 DeleteSearchedRows 删除：

```
    private void Delete4(IFeatureClass PFeatureclass)
        {
          IQueryFilter pQueryFilter = new QueryFilterClass();
          pQueryFilter.WhereClause = "objectID<=" + DeleteNum;
          ITable pTable = PFeatureclass as ITable;
          pTable.DeleteSearchedRows(pQueryFilter);
          System.Runtime.InteropServices.Marshal.ReleaseComObject(pQueryFilter);
        }
```

使用 ExecuteSQL 删除：

```
    private void Delete4(IFeatureClass PFeatureclass)
        {
          IDataset pDataset = PFeatureclass as IDataset;
          pDataset.Workspace.ExecuteSQL("delete from " + PFeatureclass.AliasName + "
where objectid<=" + DeleteNum);
        }
```

测试代码：

```
    IFeatureLayer pFeatureLayer = axMapControl1.Map.get_Layer(0) as IFeatureLayer;
    IFeatureClass PFeatureClass = pFeatureLayer.FeatureClass;
    System.Diagnostics.Stopwatch MyWatch = new System.Diagnostics.Stopwatch();
    MyWatch.Start();
    Delete1(PFeatureClass)
    //Delete2(PFeatureClass);
    //Delete3(PFeatureClass);
    //Delete4(PFeatureClass);
    //Delete5(PFeatureClass);
    MyWatch.Stop();
```

```
MessageBox.Show("删除时间:" + MyWatch.ElapsedMilliseconds.ToString() + "毫秒");
```

测试结果如表 18-1 所示。

表 18-1　　测试结果

测试方法	第 1 次时间（单位 ms）	第 2 次时间（单位 ms）
1	5214	5735
2	299	290
3	59	28
4	26	26

结论：使用 ExecuteSQL，可以直接进行数据的删除操作，所以删除最快，数据库的使用效率最高。DeleteSearchedRows 和 ExecuteSQL 属于批量删除，性能较优。而在查询结果中删除，速度最慢。

18.3　数据游标

ArcEngine 使用何种游标(Cursor)来管理记录子集，取决于数据源。ICursor 和 IFeatureCursor 一样，ICursor 用于操作表，而 IfeatureCursor 则用于操作要素类。即 Cursor 是一种为了特定的目的——操作存储在传统数据库表中的记录子集而建立的类结构；FeatureCursor 的记录子集则是存储在 Shapefile 文件、个人 Geodatabase 和企业 Geodatabase 中。

Cursor 和 FeatureCursor 有 3 种类型的游标，常用的是 SearchCursor，用于查询操作以返回一个满足查询条件的记录子集，该游标是一种只读的游标，用于遍历获取的信息。InsertCursor 专门用于往一个表中插入一条记录，UpdateCursor 则用于更新或删除记录。

Cursor 类用于产生一个与数据库表进行交互的对象，是一个非实例化对象，因此必须使用另一个对象来获得一个 Cursor 类的实例。在 ArcEngine 中，表类用于产生一个 Cursor 类的实例。表类包含有 3 种方法，能够产生一个 Cursor 类的实例。如 ITable 接口拥有 3 种方法，能够返回特定类型的 Cursor。ITable 接口的 Search、Insert 和 Update 等方法能够用于返回 Cursor 实例，这些方法的名字与返回的 Cursor 类型相对应。

FeatureCursor 与 Cursor 类相似，其区别在于前者是操作地理数据集，而后者是操作传统数据库表。FeatureCursor 类也是通过一个 FeatureClass 对象的方法产生的非实例化对象，IFeatureClass 接口也包含 Search、Insert 和 Update 等方法，用于返回一个 IFeatureCursor 实例。

18.4　投影变换

投影变换大多采用 IGeometry 接口的 Project 方法，这个方法简单，但是是使用 GeoStar 组件做投影变换的最慢的方法。在 GeoStar 的投影变换组件中，有一个 GeoProjectedCoordinateSystem 组件类，该类实现 IGeoProjectedCoordinateSystem2 接口，通过该接口可以进行高效的投影变换。具体步骤如下：

创建 GeoProjectedCoordinateSystem 组件；

获取源和目标空间参考对象；

获取要变换的 Geometry 对象；

通过 Geometry 得到坐标串和解释串；

用 GeoProjectedCoordinateSystem 组件进行投影变换。

一般情况下，线和面对象的 Geometry 使用这种方法投影变换，加上读写的时间，比使用 Geometry 的 Project 方法要快 15 到 20 倍。

18.5 ITopologicalOperator

在空间分析中通常要使用 ITopologicalOperator 接口，但是这个接口出现问题的几率也是非常高。有的时候既没有报错，又没有结果，比如 merge 方法。下面的代码演示了如何处理这类问题。

```
ICursor  cursor ;
featureSelect.SelectionSet.Search(null,false,out  cursor );
IFeatureCursor featureCursor =cursor as IFeatureCursor ;
IFeature  featureFirst =featureCursor.NextFeature();
m_EditWorkspace.StartEditOperation();
IGeometry geometryFirst =featureFirst.Shape;
ITopologicalOperator2  topo =(ITopologicalOperator2) geometryFirst;
//这里是重点，先强制检查下面 3 个步骤，然后进行操作，成功的可能性高一些
topo.IsKnownSimple_2=false;
topo.Simplify();
geometryFirst.SnapToSpatialReference();
//这是准备合并的图斑使用的
ITopologicalOperator2  topo2;
IGeometry  geometryNext;
IFeature featureNext=featureCursor.NextFeature();
geometryNext=featureNext.Shape;
While(featureNext!=null)
{
    topo2=geometryNext as ITopologiccalOperator2;
    top2.IsKnownSimple_2=false;
    geometryNext.SnapTOSpatialReference();
    geometryFirst=topo.Union(geometryNext);
    featureNext=featureCursor.NextFeature();
    geometryNext=featureNext.Shape;
}
```

18.6 缓冲区查询

缓冲区查询是 GIS 应用程序中最常使用的功能，但是经常会出现缓冲区查询的图形参数是正确的，但却出现了异常的情况，为此（系统性能许可的情况下）可以采用一些多次执行缓冲区查询的方式来避免出现这类问题。

比如写个缓冲区查询方法，在内部做 10 次循环，如果缓冲区出现异常，则扩大缓冲区半径增加 0.1 倍，直到得到正确的结果。

18.7 插入记录效率

往要素类中插入记录的方法有两种，一种是 IFeature.Store，另一种是 IFeatureCursor.Insert（IfeatureBuffer）和 IFeatureCursor.Flush 方法。由于后者使用了缓存，因此速度比前者快。