Open CASCADE Foundation Classes – Basics

Open CASCADE 基础库之基本功能

eryar@163.com

本章主要介绍 ***Open CASCADE*** 的基本功能，如内存管理(***memory management***)、手柄编程(***programming with handles***)、基本类型(***primitive types***)、异常处理(***exception handling***)、泛型编程(***genericity by downcasting***)、***Plug-in*** 的创建等……

一、数据类型 ***Data Types***

1．1　基本类型 ***Primitive Types***

基本类型是由语言定义的且他们是由值控制的。有些基本类型从类 ***Storable*** 继承而来。这就意味着他们可以在持久性对象中使用，或者被包含在对象的方法中，或者作为对象内部的一部分。由类 ***Standard_Storable*** 派生的类有：

- ***Boolean***: 用来表示逻辑数据。只有两种状态：***Standard_True*** 和 ***Standard_False***;
- ***Character***: 用来表示任意 ***ASCII*** 字符；
- ***ExtCharacter***: 用来表示字符的扩展；
- ***Integer***: 用来表示整数；
- ***Real***: 用来表示实数；
- ***ShortReal***: 用来表示实数的另一种选择，精度要低；

也有非存储类型，如：

- ***CString***:
- ***ExtString***:
- ***Address***:

下表所示为 ***Open CASCADE*** 中基本类型与 ***C++***中基本类型对应表：

C++ Types	***Open CASCADE Types***
int	***Standard_Integer***
double	***Standard_Real***
float	***Standard_ShortReal***
unsigned int	***Standard_Boolean*** ***Standard_False*** = 0; ***Standard_True*** = 1;
char	***Standard_Character***
short	***Standard_ExtCharacter***
char*	***Standard_CString***
void*	***Standard_Address***
short*	***Standard_ExtString***

 使用上述基本类型的注意事项：

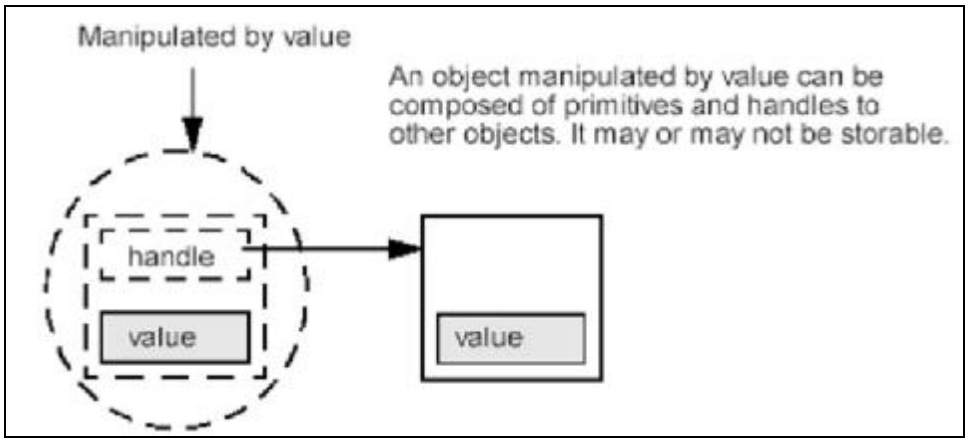
具体请参考《***Open CASCADE Foundation Classes User's Guide***》;

1．2　由值控制的类型

由值控制的变量类型分为三类：

- 基本类型；
- 枚举类型；
- 不是由类 ***Standard_Persistent*** 直接或间接派生的类；

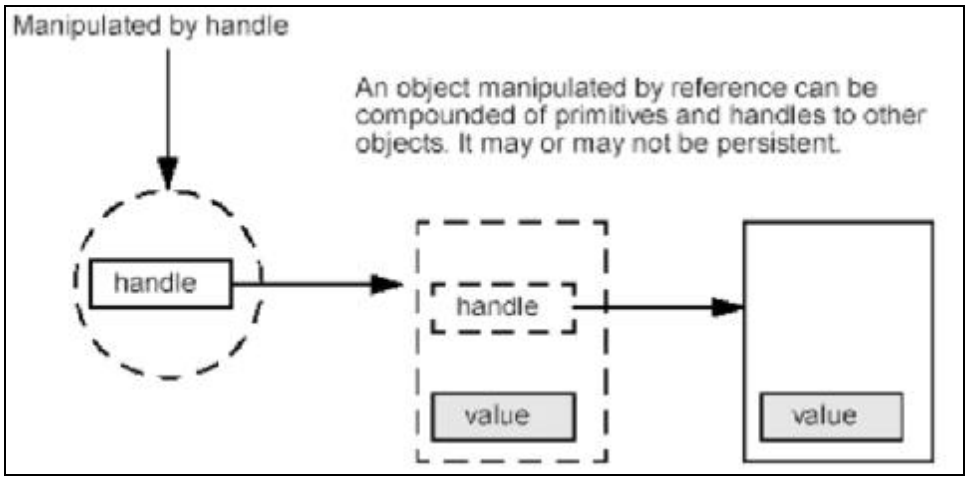
由值控制的变量比由手柄控制的变量更直接，更快，就是不能保存到文件。



1．3　由引用控制的类型

由手柄控制的变量分为两类：

- 由类 ***Persistent*** 派生的类，这种类可以保存到文件；
- 由类 ***Transient*** 派生的类；



1．4　属性的结论

	Manipulated by handle	Manipulated by value
storable	***Persistent***	**Primitive**, **Storable** (storable if nested in a persistent class)
temporary	**Transient**	**Other**

Figure 6. Summary of the relationship for the various data types between how they are handled and their storability.

二、手柄编程 ***Programming with Handles***

2．1 手柄定义 ***Handle Definition***

手柄与 ***C++***的指针(***pointer***)类似。几个手柄可以引用同一个对象，同样地，一个手柄也可以引用了几个对象，但是一次只能是一个对象。为了访问引用的对象，必须先解除引用(***de-referenced***)，就像使用 ***C++***的指针一样。临时变量和持久变量既可以是由值控制的也可以是由手柄控制的。引用非持久性对象的手柄称为不可存储手柄。因此，持久性对象不能包含不可存储手柄。

类的组织：使用手柄的类的对象可以是持久的也可是临时的。从类 ***Standard_Transient*** 继承的类的实例是临时的，而从类 ***Standard_Persistent*** 继承的类的实例是持久的。本章只讨论临时类及其相关的手柄。持久类及其相关的手柄的组织与此类似。

类 ***Standard_Transient*** 是 ***Open CASCADE*** 中由手柄控制的类层次结构中的基类。它有引用计数部分，所有的子类都继承此部分。当使用 ***Handle***()类时，就可知引手柄引用实例的数量。

直接或间接由类 ***Transient*** 派生的类，***CDL extractor*** 将创建类的相应的手柄 ***Handle***()，类名是相同的，只是在类名前加上了“***Handle_****”。***Open CASCADE*** 提供预处理器的宏定义 ***Handle***()，用来产生相应类的带手柄的类。

使用手柄：在对临时对象执行任何操作之前，你必须声明手柄。如：若点(***Point***)和线(***Line***)是从包 ***Geom*** 中定义的两个临时对象，代码如下：

Example

```
Handle(Geom_Point) p1, p2;
Handle(Geom_Line) aLine;
```

声明手柄创建了一个未指向任何对象的空手柄。手柄可以通过其方法 ***IsNull***()来验证。使用手柄无效，可使用方法 ***Nullify***()。

只要类型兼容，既可以从创建新的对象或通过赋值来初始化手柄。手柄仅用于共享的对象。对于所有的本地操作，建议使用由值控制的类。

2．2 类型管理 ***Type Management***

Open CASCADE 提供一种描述数据类型层次的通用方式，并且可以运行时检查对象类型，与 ***C++***的 ***RTTI*** 类似。对于从类 ***Standard_Transient*** 继承的每个类，***CDL extractor*** 从类 ***Standard_Type*** 创建代码。由类 ***Standard_Transient*** 派生的类的虚函数 ***DynamicType***()返回一个实例。通过虚函数 ***IsKind***()来检查给定的对象是不是指定的类型。

2．3 使用手柄创建对象 ***Using Handles to Create Objects***

创建由手柄控制的对象，声明手柄并使用标准 ***C++***的 ***new*** 操作符，紧随其后调用构造函数。

```
Handle (Geom_CartesianPoint) p;
p = new Geom_CartesianPoint (0, 0, 0);
```

与指针不同的是，手柄不需要 ***delete***。因为当手柄引用的对象为零时，对象将会被自动释放。

2．4 方法调用 ***Invoking Methods***

当你使用手柄的时候，就跟使用 ***C***++的指针一样。调用手柄引用对象的方法使用操作符->。检查或修改手柄的状态，通过操作符点.来实现。下例所示为访问一个点对象的坐标：

Example

```
Handle (Geom_CartesianPoint) centre;
Standard_Real x, y, z;
if (centre.IsNull()) {
  centre = new PGeom_CartesianPoint (0, 0, 0);
}
centre->Coord(x, y, z);
```

下例所示为如何检查笛卡尔坐标点的类型：

Example

```
Handle(Standard_Transient) p = new Geom_CartesianPoint(0.,0.,0.);
if ( p->DynamicType() == STANDARD_TYPE(Geom_CartesianPoint) )
  cout << "Type check OK" << endl;
else
  cout << "Type check FAILED" << endl;
```

当调用一个空 ***Null*** 手柄时，***NullObject*** 异常将会产生。

调用类方法：类方法就是 ***C***++类中的静态函数。即用类名加上“::”和方法名来调用。

Example: finding the maximum degree of a Bezier curve:

```
Standard_Integer n;
n = Geom_BezierCurve::MaxDegree();
```

2．5 手柄释放 ***Handle De-allocation***

在删除一个对象之前，必须其没有被引用。为了减少管理对象生命周期的编程工作量，***Open CASCADE*** 中对象的删除函数是由手柄控制类的引用计数(***reference counter***)来确保。手柄就是用来管理引用计数，当对象不再引用时将会调用 ***delete*** 将其删除。当是 ***Standard_Transient*** 的子类时，通常不需要直接使用 ***delete*** 操作符。当对相同的对象使用 ***new*** 时，引用计数将会增加。当手柄被销毁、置为空或重新赋值，引用计数将会减少。当引用计数为 0 时对象将会自动调用 ***delete*** 操作符。内存分配的原理如下所示：

Example

```
...
{
Handle (TColStd_HSequenceOfInteger) H1 = new TColStd_HSequenceOfInteger;

  // H1 has one reference and corresponds to 48 bytes of memory
  {
    Handle (TColStd_HSequenceOfInteger) H2;
    H2 = H1; // H1 has two references
    if (argc == 3) {
      Handle (TColStd_HSequenceOfInteger) H3;
      H3 = H1;
      // Here, H1 has three references
      ...
    }
    // Here, H1 has two references
  }

  // Here, H1 has 1 reference
}
// Here, H1 has no reference and the referred
// TColStd_HSequenceOfInteger object is deleted
```

Cycles

本段内容不清楚，具体内容请参考原文。

2．6 不使用 ***CDL*** 创建类 ***Creating Transient Classes without CDL***

尽管可用 ***CDL extractor*** 生成手柄类及其相关 ***C***++代码，然而也可不用 ***CDL*** 管理手柄。为此，在文件 ***Standard_DefineHandle.hxx*** 中提供了几个宏定义：

DECLARE_STANDARD_HANDLE(***class_name***, ***ancestor_name***) 这个宏定义了以 ***class_name*** 为类名并继承类 ***ancestor_name*** 的手柄类。这个宏必须放在头文件中，且基类必须是可用的。

IMPLEMENT_STANDARD_HANDLE(***class_name***, ***ancestor_name***) 这个宏实现了转换方法 ***DownCast***()，应该在 ***C***++文件中使用。

DEFINE_STANDARD_RTTI(***class_name***) 这个宏声明方法需要 ***RTTI*** 支持，应该在类的 ***public*** 中使用。

IMPLEMENT_STANDARD_RTTIEXT(***class_name***, ***ancestor_name***) 实现上面的方法。

注：在使用这些宏的时候，必须确保参数的正确性，特别是父类的名字。否则定义将会不正

确，且编译也不会报错。

Example:

Appli_ExtSurface.hxx file:

```
#include <Geom_Surface.hxx>
class Appli_ExtSurface : public Geom_Surface
{
. . .
public:
  DEFINE_STANDARD_RTTI(Appli_ExtSurface)
}
DECLARE_STANDARD_HANDLE(Appli_ExtSurface,Geom_Surface)
```

Appli_ExtSurface.cxx file:

```
#include <Appli_ExtSurface.hxx>
IMPLEMENT_STANDARD_HANDLE(Appli_ExtSurface,Geom_Surface)
IMPLEMENT_STANDARD_RTTIEXT(Appli_ExtSurface,Geom_Surface)
```

三、内存管理 ***Memory Management in Open CASCADE***

在几何建模的过程中，程序创建和删除相当数量的 ***C++***对象在动态内存中，也就是堆中(***heap***)。在这种情况下，标准函数管理内存的性能可能不足够。所以，***Open CASCADE*** 在标准包中实现了内存的管理。

3．1 用法 ***Usage***

使用 ***Open CASCADE*** 内存管理只需要在 ***C*** 中使用 ***malloc***() 的地方使用 ***Standard***::***Allocate***()；在使用 ***free***()的地方使用 ***Standard***::***Free***()；在使用 ***realloc***()的地方使用 ***Standard***::***Reallocate***()。

在***C++***中，类的操作符 ***new***()和***delete***()已经定义了在申请内存时使用***Standard***::***Allocate***()并在释放时使用 ***Standard***::***Free***()。所以，类所有的对象的内存都将由 ***Open CASCADE*** 的内存管理器来管理。

CDL extractor 为所有的类定义了 ***new***()和 ***delete***()。所以，所有的 ***Open CASCADE*** 的类（小部分除外）都使用 ***Open CASCADE*** 的内存管理器。

由于操作符 ***new***()和 ***delete***()被继承，所以，所有从 ***Open CASCADE*** 派生的类，所有从 ***Standard_Transient*** 类派生的类都是由内存管理器管理。

注：若重载了部分从 ***Standard_Transient*** 类派生类的 ***new***()和 ***delete***()，尽管不推荐这样做，方法 ***Delete***()必须重定义，以便对这样的指针使用 ***delete*** 操作。这将确保合适的 ***delete***()函数将会被调用，即使是由手柄控制的对象。

3．2 配置内存管理器 ***Configuring Memory Manager***

Open CASCADE 内存管理器可以被配置以便对不同的内存区域使用不同的优化技术，或者根本不使用任何优化而直接使用 ***C*** 的 ***malloc***()和 ***free***()函数。配置方法为修改环境变量的值：

- ***MMGT_OPT***:
- ***MMGT_CLEAR***:
- ***MMGT_CELLSIZE***:
- ***MMGT_NBPAGES***:
- ***MMGT_THRESHOLD***:
- ***MMGT_REENTRANT***:

3．3 实现细节 ***Implementation details***

本段内容请参考原文。

四、异常处理 ***Exception Handling***

异常处理提供了一种从指定点转换到其他点的一种方法。一个方法可能会产生一个异常，将程序从正常执行处转换到捕捉异常处。***Open CASCADE*** 提供了异常类的层次，其基类是包 ***Standard*** 中的 ***Standard_Failure***。***CDL extractor*** 使用标准接口生成异常类。

Open CASCADE 也提供将系统信号的转换成异常的支持，如数除 0 这样所有的情况都可以用安全、统一的方法来处理了。但是为了支持不同的平台，也使用了一些特殊的方法。

如下内容为在 ***Open CASCADE*** 中使用异常处理的推荐方法。

4．1 产生异常 ***Raising an Exception***

类 ***C++***的语法：产生适当的异常将需要调用指定类型的 ***Raise***()方法。

Example

```
DomainError::Raise("Cannot cope with this condition");
```

产生了一个 ***DomainError*** 类型的异常并可附上相关信息“***Cannot cope with this condition***”，信息字符串是可选的。这个异常可以被捕捉 ***DomainError*** 类型的 ***Handler*** 捕捉到：

```
try {
  OCC_CATCH_SIGNALS
  // try block
}
catch(DomainError) {
// handle DomainError exceptions here
}
```

正常使用异常处理：异常不能被当作编程技巧来替代“***goto***”，但可作为防止方法被误用的一种方法。

五、***Plug-In*** 管理

本段内容请参考原文。

六、结论

本章内容介绍了一些 ***C***++编程的知识及 ***Open CASCADE*** 对 ***C***++的包装。其中，手柄编程是 ***C***++中常见的方法，这种引用计数的方式使内存的管理更方便。

eryar@163.***com***

Pudongxin Shanghai China

2012-8-23