Conversion Operators in OpenCascade

eryar@163.com

Abstract. C++ lets us redefine the meaning of the operators when applied to objects. It also lets us define conversion operations for class types. Class-type conversions are used like the built-in conversions to implicitly convert an object of one type to another type when needed. A conversion operator provides a way for you to define how an object can be converted automatically to a different type. The paper gives some conversion operators examples in OpenCascade.

Key words. OpenCascade, Conversion Operators, Operator overloading

1. Introduction

C++允许我们重新定义操作符用于类类型对象时的含义。如果需要，可以像内置转换那样使用类类型转换，将一个类型对象隐式转换到另一类型。如在 OpenCascade 中经常看到如下类似的代码：

```
TopoDS_Shape theSphere = BRepPrimAPI_MakeSphere(1.0);
```

其中，BRepPrimAPI_MakeSphere 也是一个类，直接赋值给了另一个类 TopoDS_Shape 的对象 theSphere。第一次这么来用的时候有些困惑，不知道你有没有这样的疑问，不管你有没有，反正我是有的（Just kidding）。后来才知道，这就是一种重载方式，重载了类型转换操作符（Conversion Operator）。

使用类型转换操作符在将一种类型转换到另一种类型时，感觉自然。当类较多且经常需要进行类型之间的转换时，定义类型转换操作符还是很方便的。本文结合 OpenCascade 程序来体验使用类型转换操作符带来的便利。

2. Conversion Operators

转换操作符（Conversion Operators）提供了从一种对象类型自动转换到另一种类型的方式。一个经典例子就是自定义字符串类，但是可以将这个自定义的字符串类当作函数参数传给 const char*类型的函数，如标准 C 中的一些函数：strcmp(), strlen()。示例程序如下所示：

```
class MyString
{
public:
    MyString(const char* string);

    // convert MyString to a C-style string.
    operator const char*() { return mBuffer; }

private:
    char* mBuffer;
    int mLength;
};

// MyString objects get automatically converted to const char*
MyString mystr("Haggis");
int same = strcmp(mystr, "Edible");
int len = strlen(mystr);
```

转换操作符是一种特殊的类成员函数。它定义将类类型值转换为其他类型值的转换。转换操作符在类定义体内声明，在关键字 operator 之后跟着转换的目标类型。转换操作符的通用形式为：

```
    operator type();
```

转换函数必须是成员函数，不能指定返回类型，且形参表必须为空。因为转换的目标类型已经出现在转换操作符中了，所以就不需要重复定义返回值类型了。

3. Conversion Operators in OpenCascade

OpenCascade 中很多地方用到了转换操作符，如将生成的基本实体转换成其他拓朴类型时就用到了转换操作符，程序代码如下所示：

```
/*
*      Copyright (c) 2014 eryar All Rights Reserved.
*
*                File : Main.cpp
*              Author : eryar@163.com
*                Date : 2014-04-12 18:02
*             Version : V1.0
*
*      Description : Learn Conversion Operators in OpenCascade.
*
*         Key words : OpenCascade, Conversion Operators
*
*/

#define WNT
#include <BRepPrimAPI_MakeSphere.hxx>

#pragma comment(lib, "TKernel.lib")
#pragma comment(lib, "TKMath.lib")
#pragma comment(lib, "TKBRep.lib")
#pragma comment(lib, "TKPrim.lib")
#pragma comment(lib, "TKTopAlgo.lib")

void TestConversionOperators(void)
{
	TopoDS_Shape theSphereShape = BRepPrimAPI_MakeSphere(1.0);
	TopoDS_Solid theSphereSolid = BRepPrimAPI_MakeSphere(1.0);
	TopoDS_Shell theSphereShell = BRepPrimAPI_MakeSphere(1.0);
	TopoDS_Face theSphereFace = BRepPrimAPI_MakeSphere(1.0);

	// error C2440: 'initializing' : cannot convert
	// from 'BRepPrimAPI_MakeSphere' to 'TopoDS_Wire'
	//TopoDS_Wire theSphereWire = BRepPrimAPI_MakeSphere(1.0);
}

int main(int argc, char* argv[])
{
	TestConversionOperators();

	return 0;
}
```

如上代码所示，可以将类 BRepPrimAPI_MakeSphere 自动转换成 TopoDS_Shape, TopoDS_Solid, TopoDS_Shell, TopoDS_Face，但是不能自动转换成 TopoDS_Wire。这是因为在其父类 BRepPrimAPI_MakeOneAxis 中定义这些转换操作符，代码如下所示：

```
//! The abstract class MakeOneAxis is the root class of <br>
//! algorithms used to construct rotational primitives. <br>
class BRepPrimAPI_MakeOneAxis    : public BRepBuilderAPI_MakeShape {
public:

  DEFINE_STANDARD_ALLOC
```

```
   //! The inherited commands should provide the algorithm. <br>
//!                 Returned as a pointer. <br>
   Standard_EXPORT     virtual    Standard_Address OneAxis()    = 0;
   //! Stores the solid in myShape. <br>
   Standard_EXPORT     virtual    void Build() ;
   //! Returns the lateral face of the rotational primitive. <br>
//! <br>
   Standard_EXPORT       const TopoDS_Face& Face() ;
Standard_EXPORT operator TopoDS_Face();
   //! Returns the constructed rotational primitive as a shell. <br>
   Standard_EXPORT       const TopoDS_Shell& Shell() ;
Standard_EXPORT operator TopoDS_Shell();
   //! Returns the constructed rotational primitive as a solid. <br>
   Standard_EXPORT       const TopoDS_Solid& Solid() ;
Standard_EXPORT operator TopoDS_Solid();

protected:

private:
};
```

由上述代码可知，当将 BRepPrimAPI_MakeSphere 赋值给 TopoDS_Shape 时，会调用 operator TopoDS_Shape()转换操作符的转换函数；当赋值给 TopoDS_Shell 时，会调用 operator TopoDS_Shell()转换函数，等等。未定义的转换类型是不允许自动转换的，如 TopoDS_Wire。

使用这些转换操作符使不同类型之间的类型转换很自然直观，看上去就像调用了一个函数。

类型之间的转换当然还有其他方法，如给转换的目标类型增加一个构造函数来实现。但是使用构造函数来转换不能转换成基本类型，如 int, double 等；还有个不足之处就是要修改转换目标类的声明文件来增加一个构造函数。没有转换操作符来得自然，方便。

4. Conclusion

当需要在不同类型之间进行类型转换时，可以使用转换操作符（Conversion Operators）。使用转换操作符的方式别其他方法要简单直观。

由于 OpenCascade 中类型比较多，且经常需要要不同类型之间进行转换操作，所以将一些常用的转换定义成转换操作符还是很方便的。

5. References

1. Bjarne Stroustrup. The C++ programming language. Higher Education Press. 2009
2. Stanley B. Lippman, Josee Lajoie, Barbara E. Moo. C++ Primer. Addison Wesley. 2005
3. Martin Reddy. API Design for C++. Morgan Kaufmann. 2011