OPEN CASCADE 学习笔记
——句柄

著: Roman Lygin

译： George Feng

这是一篇关于开源三维建模软件 OPEN CASCADE 内核的博文：ROMAN LYGIN 是 OPEN CASCADE 的前程序开发员和项目经理，曾经写过许多关于该开源软件开发包的深入文章，可以在他的博客(HTTP://OPENCASCADE.BLOGSPOT.COM)上面找到这些文章。

序

在 OpenCascade 的论坛上知道了 Roman Lygin 在他的博客上写了 Open CASCADE notes 系列文章，但是却无法访问他的博客，幸而百度文库已经收录了 Topology and Geometry 和 Surface Modeling 两篇文章，拜读之后获益良多。如果大家发现文中翻译有错误或不足之处，望不吝赐教，可以发到我的邮箱 fenghongkui@sina.com.cn ，十分感谢。

2012 年 6 月 28 日星期四

译者案：句柄是某个对象的一种标号，它是一个数值，它不是指针，根据指针可以知道对象在内存中的位置，但是通过句柄却不能直接更改内存中的对象，这事实上对于内存中的对象形成了一种保护。中文中句柄这个词应该是来源于手柄的演化，利用手柄可以拎起来皮箱、包裹、茶壶等，就是说通过手柄可以控制一个对象，而计算机语言中的句柄却是用来“拎”(间接访问)内存中或者其他设备对象的。

我的第一篇文章，引言

我也成为一个博友了，这是我第一次发博文，所以我对这种发文方式没有概念，

也不知道是否有效。让我们走着瞧吧。

我曾经在 Open CASCADE(OCC) 工作了 7 年时间，2004 年离开，然后去了 Intel，在那一直工作到现在。在 OCC 的那些年的经历是非常珍贵的，我做过软件开发和项目管理，客户关系维护，与来自于各地和具有各种文化背景的人一起工作。

从 1997 年开始，我就在 CAD 数据交换小组作为软件开发工程师工作，我觉得这非常幸运，因为这使我可以学习到很多 OCC 代码，因为数据交换使用了大量的 OCC 算法。假如你曾经看到了 IGES、STEP、Shape Healing 模块的源代码，你可能已经看到过我的全名 (例如，在文件 IGESToBRep_IGESBoundary.cxx 中) 或者首字母名字 “ rln ” (例如，在文件 ShapeFix_Wire.cxx 中)。每一个 Open CASCADE 员工都有一个首字母名字，在内部广泛使用 (我记得几分钟的会议就会有这样的事情 “ 出席人员： ABV，PDN，GKA，SMH，SZV，等等 ”)。RLN 就是我的。另外，这非常有趣，但是在 Intel 公司中没有这样的命名方法，没有像 OCC 那样几乎都采用三字母命名。在 Intel，人们使用名字 (first names)，有时候也用姓的首字母，例如， Roman S，特别是在有好几个 Roman 时，就更要使用这种加上姓的首字母的方式。

我仍然与 Open CASCADE 小组人员保持着密切的联系， 跟哪的很多人都是朋友。即使是在 Intel，我还时常深入研究 OCC，查看有什么地方进行了更新，而且还找机会开发一些东西 (几个星期前， 我还做了 OCC 多线程的开发框架原型系统)。

那么为什么要开这个博客呢？目的是什么？

- 分享我的想法。有时候看到论坛上的问题，我忍不住要回帖，或者看一些原型系统顾不上吃中午饭。 你能想象一个疯子要在家中开发一个商业软件， 因为他的工作不允许他在工作时间开发他的软件。那就是我在 2001 年的样子，当时我换成了管理工作。尽管 OCC 有很多缺陷，但我始终认为 OCC 是一个伟大的产品，但是它被大大低估了。它没有获得它应该有的知名度。
- 分享知识同时帮助大家。 在论坛上同样的问题被一遍又一遍的提出来， 这些问题我都可以在几秒钟之内回答。 参考文档也不够用， 有时候概述是更好解决问题的方法。

首先要解决什么问题呢？我想应该先弄清楚什么人要看这些教程， 还要清楚怎么使用这些教程。我拜访过很多用户，看到过很多印象深刻的基于 OCC 的应用程序，我真想大喊一声 “ 你怎么能那样写程序呢？ ” 。所以，假如有人原意分享他

们的技巧， 我非常乐意跟他们一块学习。 保持一定的编写软件的技术是非常重要的。

我想应该将我的笔记分成许多短文 (或许也不算短)，这样在论坛上会使这些文章变得更为有趣。 假如你有兴趣， 可以给我建议应该怎么做才能使文章变得更为有趣。

几点需要澄清的地方：

- 该博客是可以互动的。请大家积极评论，欢迎发表各种看法和经验。假如有人想要添加自己的文章，那就更好了。
- 我并不知道 OCC 中的所有细节 (恐怕也没有一个这样的人)。总有照顾不到的地方，我还没有涉及到。但是如果需要我可以深入研究，并分享一些看法。再强调一下，假如有人对 OCC 有更深入的认识，他们的贡献也非常有价值。
- 当然，这个博客是我个人的， 并没有得到 Open CASCADE 或者其他方面资助。

谢谢！

Roman

POSTED BY ROMAN LYGIN AT 19:49, 2008-11-19 

第 1 节 句柄类的结构

让我们开始讲句柄 (handle) 的第一篇文章，假如你想要在 Open CASCADE 上开发软件的话，这篇文章虽然比较简单但是非常重要。
你可能已经注意到许多类直接或者从它的父类间接继承了 Standard_Transient ，例如， Geom_Surface 和 AIS_InteractiveObject ，这样在程序中就可以使用 Handle(Geom_Surface) 。

Open CASCADE 尽力不使用指针 (至少在 API 中不使用)。在任何需要共享对象的地方，都使用句柄。

句柄是非常常见的概念，通常称为智能指针。例如， Boost 库(www.boost.org) 有好几个类用来处理智能指针， 其中的 intrusive_ptr 是最接近 OCC 中的 Handle 的。 QT(www.trolltech.com) 中也有类似功能的 QPointer 。

句柄提供了一种方法自动引用对象， 而不用考虑令人头痛的删除问题。 当最后一个指向对象的句柄销毁的时候， 句柄指向的对象 (从 Standard_Transient 派生出来的)将会销毁。

除了上面的功能，句柄也提供了一种安全的类型识别和对象的向下转换(downcasting)。可以看看 Foundation Classes User's Guide 上面详细讲了这种机制。

Boost 和 QT 都在模板中定义了智能指针，Open CASCADE 没有这样做。Open CASCADE 使用两个平行的显示类结构，一个从 Standard_Transient 继承，另一个从 Handle_Standard_Transient 继承。当 CDL 解析器生成头文件时，或者使用宏定义 DEFINE_STANDARD_HANDLE 时，就生成了一个新的句柄类。我相信选择这种句柄结构是由于历时原因造成的，在 1990 年代的时候，老的编译器对模板的支持不好。由于同样的原因，TCollection 类也是基于 #defines 的(直到 2002 年，出现了 NCollection，这个类是基于模板的)。

事实上，必须得说一下，还有另外一个平行结构——称为持久(persistence) 类 Standard_Persistent。但是由于它们几乎从来都不使用，在这篇文章中我们可以忽略它们。但是在这讲的东西，都可以直接应用到这些持久类中。

为了生成你自己的句柄类，你需要使用在 Standard_DefineHandle.hxx 的宏定义：

```
DECLARE_STANDARD_HANDLE(class_name,ancestor_name)
DEFINE_STANDARD_RTTI(class_name)
```

```
IMPLEMENT_STANDARD_HANDLE(class_name,ancestor_name)
IMPLEMENT_STANDARD_RTTIEXT(class_name,ancestor_name)
```

例如：

```
DEFINE_STANDARD_HANDLE(OCC_UT_Id,Standard_Transient)
class OCC_UT_Id : public Standard_Transient
{
public:
OCC_UT_Id() : myId (0) {}
OCC_UT_Id(const Standard_Integer theId) : myId (theId) {}
Standard_Integer Id() const { return myId; }
private:
Standard_Integer myId;
public:
DEFINE_STANDARD_RTTI(OCC_UT_Id)
};

IMPLEMENT_STANDARD_HANDLE(OCC_UT_Id,Standard_Transient)
IMPLEMENT_STANDARD_RTTIEXT(OCC_UT_Id,Standard_Transient)
```

大概你已经注意到 DEFINE_STANDARD_RTTI 在类中转换成为成员函数时，不需要任何参数，它是用来保持一致性的。

直到 Open CASCADE 6.3.0 时，Open CASCADE CDL 解析器都为句柄生成显示代码，例如 Handle_Geom_Surface.hxx 。为此最近我给 Open CASCADE 公司的人员送了一个修改版本， 这样它就可以生成宏， 使用头文件使得程序更为干净。让他们自己去改吧。
另外一个值得注意的地方是，我在论坛上看到有人将宏
IMPLEMENT_STANDARD_RTTI 与下面的宏，
IMPLEMENT_STANDARD_TYPE,
IMPLEMENT_STANDARD_SUPERTYPE,
IMPLEMENT_STANDARD_SUPERTYPE_ARRAY ，等一起使用，尽力模仿 CDL 解析器 (它放到子目录中 (dry subdirectory) 的东西)。不要受此困惑，IMPLEMENT_STANDARD_RTTIEXT 将会做所有的工作，它将会使你的代码变得干净。

待续... ...
POSTED BY ROMAN LYGIN AT 20:22, 2008-11-20

第 2 节 Standard_Transient 和 MMgt_TShared

接上节... ...

Standard_Transient 和 MMgt_TShared
你可能已经注意到了，大多数句柄类都是从 MMgt_TShared 继承来的，而 MMgt_TShared 继承了 Standard_Transient ，但是它没有添加任何变量。 在 cdl 文件中的注释声称它支持引用计数。引用计数在 Standard_Transient 的内部。我仔细研究了 Open CASCADE 3.0 (在 1999 年发布，第一个开源发布版本)，也是这样的。所以这种改变发生在至少 10 年前，但是 MMgt_TShared 这个遗弃的孤儿仍然活着。所以我现在只使用 Standard_Transient ，而且建议 OCC 的人员将 MMgt_TShared 最终移除。所以你可能会想要将 Standard_Transient 作为你的基类。

*循环依赖 *
假如你有一个循环依赖的话 (例如一个对象引用另一个对象)，这样底层的实体 (underlying objects) 将不能销毁， 这将会引起内存泄漏。 例如下面的类会引起循

环依赖。

```
DEFINE_STANDARD_HANDLE(OCC_UT_Employee,Standard_Transient)
class OCC_UT_Employee : public Standard_Transient
{
public:
OCC_UT_Employee (const Handle(OCC_UT_Employee)& theBoss) : myBoss
(theBoss) {}
const Handle(OCC_UT_Employee)& Boss() const { return myBoss; }
void AddDirectReport (const Handle(OCC_UT_Employee)& theReport);

private:
Handle(OCC_UT_Employee) myBoss;
TColStd_ListOfTransient myDirectReports;
public:
DEFINE_STANDARD_RTTI(OCC_UT_Employee)
};
```

正如 Foundation Classes User ’s Guide所建议的，在某些情况下你可以使用指针(例如，OCC_UT_Employee* myBoss)，或者首先将某些引用置空(myBoss.Nullify)。

使用句柄的代价 (Handle overhead)

正像这个世界上的任何事情一样，你必须为使用句柄的方便付出代价。任何赋值或者删除导致执行好几条指令 (比较，函数调用，分支等)。记着这个。假如你在对性能要求苛刻的地方使用句柄，要好好想想你怎么优化指令。下面将介绍几个我想到的优化程序时需要注意的几个地方。

1. 首先使用 DownCast() 得到期望的数据类型，然后在一个循环种使用它，比较下面的两种实现方式：

```
static void CheckEmployee (const Handle(Standard_Transient)&
theEmployee)
{
for (int i = 0 ; i < aNbCycles; i++) {
```

```
Handle(OCC_UT_Employee) anEmp =
Handle(OCC_UT_Employee)::DownCast (theEmployee);
}
}

static void CheckEmployee (const Handle(Standard_Transient)&
theEmployee)
{
Handle(OCC_UT_Employee) anEmployee =
Handle(OCC_UT_Employee)::DownCast (theEmployee);
for (int i = 0 ; i < aNbCycles; i++) {
Handle(OCC_UT_Employee) anEmp = anEmployee;
}
}
```

当循环次数次数 aNbCycles 达到 10,000,000 时，在我的笔记本上， 前者花费 1.6 秒，后者花费 0.42 秒，或者 3.8X 倍速。

2. 避免复制 (Avoid copies)

在任何可能的情况下，类方法应该返回 const Handle()& ，并将其赋值给同样类型的值，而不是简单的使用 Handle() 。思考一下下面的两个例子：

a. 假如你在 OCC_UT_Employee 类种定义了 Boss() 函数：

```
Handle(OCC_UT_Employee) Boss() const { return myBoss; }
```

而且以下面的方式使用它：

```
static void CheckBoss (const Handle(Standard_Transient)& theEmployee)
{
Handle(OCC_UT_Employee) anEmployee =
Handle(OCC_UT_Employee)::DownCast (theEmployee);
for (int i = 0 ; i < aNbCycles; i++) {
Handle(OCC_UT_Employee) aBoss = anEmployee->Boss();
}
}
```

b. 现在转换到 const Handle()&:

```
const Handle(OCC_UT_Employee)& Boss() const { return myBoss; }
...
static void CheckBoss (const Handle(Standard_Transient)& theEmployee)
{
Handle(OCC_UT_Employee) anEmployee =
Handle(OCC_UT_Employee)::DownCast (theEmployee);
for (int i = 0 ; i < aNbCycles; i++) {
const Handle(OCC_UT_Employee)& aBoss = anEmployee->Boss();
}
}
```

速度从 0.26 秒提高到了 0.03 秒，或者说 8.5x 倍速！然而，必须承认， Open CASCADE 自己的代码中也有这样的问题。 OCC 的开发人员，如果你读到了这篇文章，一定要注意了。

待续... ...

POSTED BY ROMAN LYGIN AT 23:40, 2008-11-20

第 3 节使用句柄的代价

接上节... ...

3. 在要求苛刻的地方避免使用 DownCast()
比较下面代码：
a.

```
Handle(Standard_Transient) aTransient = new OCC_UT_Id(1);
for (i = 0 ; i < aNbCycles; i++) {
anId = Handle(OCC_UT_Id)::DownCast (aTransient)->Id();
}
```

b.

```
for (i = 0 ; i < aNbCycles; i++) {
Handle(OCC_UT_Id)& anIdHTmp = *((Handle(OCC_UT_Id)*) &aTransient);
anId = anIdHTmp->Id();
}
```

c. for (i = 0 ; i < aNbCycles; i++) {

```
OCC_UT_Id* anIdPTmp = (OCC_UT_Id*)aTransient.Access();
anId = anIdPTmp->Id();
}
```

a. 花费了 1.75 秒，b 和 c 大约 0.04 秒，或者说提高了 40 倍速度 (在某些情况下达到了 100 倍速度)!

顺便说一下，在 BRep_Tool 中可以找到 b 种类型的用法。

然而一定要注意一些可能出现问题的情况。 除非你可以进行有效的检查， 或者你可以安全的使用它，否则不要使用直接转换。例如，下面的代码会抛出异常，而不是返回一个空句柄。

```
TopoDS_Face aFace;
TopLoc_Location aLoc;
Handle(Geom_Surface) aSurf = BRep_Tool::Surface (aFace, aLoc);
```

这些是我自己的观点，如果有人分享他们的观点，我非常乐意聆听。

由于 Handles() 是使用最为广泛的类，它们的实现必须完美无暇。关于这点，我想假如使用 UndefinedHandleAddress (等于 0xfefd0000 或者 64 位平台的 0xfefdfefdfefd0000 参见 Hande_Standard_Transient.hxx) 来表示空句柄当然是正确的。但是，简单的 0 也足够了，而且这能够节省比较操作的时间。是否有人原意来做个实验？ OCC 的同仁们？

*考虑多线程 *

总之，还是要说一下， Handle() 目前为止还不是线程安全的。 需要使用临界区 (例如 Standard_Mutex) 来保护句柄实例，从而可以并发访问。否则，会引发数据读写竞争问题 (data race) ，这是由并发访问非保护数据引起的，这可能导致引用计数被破坏，从而进一步引起内存泄漏，访问冲突或者其他令人头疼的问题。

好了，就这些了，伙计们。那么，这篇文章有多大的用处呢？请将你的评论和想法告诉我。这对我非常重要，谢谢！

Roman

POSTED BY ROMAN LYGIN AT 12:54, 2008-11-21