DM-Pillar 视频教程脚本

第一部分 环境配置

1-1 DM-Pillar 使用入门

大家从官网上下好我们的产品后，需要先安装 VS，我们的这套动态库是支持 VS2019 及以上的编译器的。

下载完毕后先看一下我们产品的组成，我们的官方可以下下来三个压缩包，分别是 DM demo、DM 帮助文件、DM 核心库，解压后可以看到有四个文件夹，demo 中是我们的测试程序，Pillar 中是我们核心库的头文件和动态库，然后是接口手册和使用手册，那我们以一个程序为例，看一下如何使用我们的 DM 的动态库，先打开 VS，我们创建一个新项目，这里选择控制台应用，选默认的路径，再设置一下 64 位，配置一下项目，首先设置一下头文件的目录，把头文件中的五个文件夹的目录都放在其中，然后在配置一下库目录，也就是 dll 所在位置，设置一下依赖项名称，这下我们配置就完成了。那么我们写一个程序测试一下，我们以 PLRMathVector2D 二维向量类为例，我们创建一个向量然后进行向量数乘，然后最后看一下结果对不对，首先我包含一下这个类的头文件，然后为了方便，我们直接用 using namespace 求解的命名空间，然后我们声明一个 PLRMathVector2D 类的一个对象，然后我们使用 SetCoord 函数设置向量值，之后我们再声明一个新的对象 vec2，它的值是 vec 变量乘以 2，然后我们声明两个变量 a 和 b，然后使用 GetCoord 函数获取 a 和 b 的值，最后将 a 和 b 的值打印出来，看是否是预期的 2 和 4，我们运行一下看看，结果是正确的，符合我们的预期。

```
#include "PLRMathVector2D.h"

using namespace QiuJiePillar;
```

```
int main()
{
    PLRMathVector2D vec;
    vec.SetCoord(1, 2);
    PLRMathVector2D vec2 = 2*vec;
    double a, b;
    vec2.GetCoord(a, b);
    std::cout << a << " " << b << std::endl;
    }
```

1-2 Demo 程序开发环境搭建

下面介绍 Demo 程序的开发环境

需要安装 Visual Studio 2019。

Visual Studio 2019 的 qt 插件。

qt，其中 Demo 程序中使用的 qt 库是 6.3.1 版本。

Demo 程序还使用了 vtk 开源库，版本是 9.2.0。

还有 SARibbon 开源库。

OpenCasCade 开源库，这个的版本是 7.6.0。

以上三个开源库的 Visual Studio 2019 版本我们已经编译好，附加在 Demo 源码包中。

1-3 Demo 程序的结构

下面介绍 Demo 程序的结构

Demo 程序有 QJApp、GUIMain、QJEdApi、DataModel、test_Create、test_Interface 六个主要模块。

QJApp 是 Demo 程序的启动模块，启动后加载主窗口。

GUIMain 是 Demo 程序的主界面模块，程序启动后看到的界面就在该模块中实现，主界面用 qt 实现，采用 ribbon 风格。

QjEdAPI 是注册本地命令和事件的模块，命令名与命令函数在该模块中关联。

DataModel 是一个简易的数据模型模块，为命令函数中生成的几何实体提供添加、删除、隐藏、显示等基础服务。

test_Create 是命令插件模块，命令函数在该模块中实现。

test_Interface 是操作按钮界面模块。

1-4 Demo 程序中新添加一个命令

为了帮助用户方便的试用几何内核功能，需要 Demo 程序比较容易扩展，我们采用在插件中注册命令的方式实现这个目的。

下面介绍怎样在 Demo 程序中添加一个新的命令。

添加新命令需要以下三个步骤。

添加命令按钮。

在插件中实现命令函数。

在插件中注册命令。

结合程序，具体说明一下。

首先，在 GUIMain 模块的 GUIMain 类中有四个创建命令按钮的函数。

分别对应 ribbon 风格的大图标，中图标，小图标，以及下拉弹出式的图标组。

例如，在“线框”面板中添加“画圆”命令按钮。

重新编译后运行，可以看到命令面板中多了一个命令按钮。

需要注意创建按钮函数的第四个参数，这是命令名，后面注册命令的时候会用到。

第二步，在 test_Create 工程中实现命令函数

第三步，注册命令。

我们利用动态链接库的入口函数注册关联命令。

找到 test_Create 的 DllMain，通过 CommandStation 的 RegisterCommand 函数注册命令。

第一个参数是命令名，这个名字就是创建按钮函数时使用的第四个参数。

第二个参数是在第二步中实现的命令函数名。

经过以上三步操作，我们就在 Demo 程序中新添加了一个命令。

重新编译后运行，可以看到命令已经联通了。

1- 5 Demo 程序中命令执行流程

下面介绍 Demo 程序的命令执行流程。

操作按钮的槽函数是 GUIMain 类的 OnExecute。

OnExecute 中提取按钮的命令字符串，用解析出来的命令字符串作参数，调用 CommandStation 类的 Run 函数。

Run 函数先查命令字符串对应的回调函数，如果找到就调用回调函数执行该命令。回调函数中生成几何实体，使用 DataModel 类提供的接口服务完成几何实体的添加、删除、显示、隐藏等功能。

第二部分 功能演示

2-1 创建立方体

创建立方体我们可以直接参考测试程序中的 CreateFreeformSolidCube 函数，其核心函数是 PLRCreateSolidCube。

我们打开 test_create 项目看一下，我们进入 CreateFreeformSolidCube 这个函数看一下实现，首先，我们获取一下工厂指针，由于所有的 实体对象都是由工厂生成，所以除了数学计算相关类来说，我们一般都要先获取一下工厂指针，然后判断一下获取指针是否失败，接下来我们创建四个点，分别是原点、x 轴端点、y 轴端点、z 轴端点，然后我们调用 PLRCreateSolidCube 函数生成立方体，我们看一下这个函数，这个函数的参数都有标注，如果创建成功最后会生成一个 PLRBody 的指针，然后判断一下获取指针是否失败，最后呢，我们清空一下界面中的实体对象，再将 PLRBody 实体指针添加到可视化界面当中，我们这里注意一下，AddGeometry 这个函数虽然可以接收 PLRTopObjct 类型的指针，但一般来说，我们直接传入它的子类 PLRBody 的指针即可，那现在我们运行一下看看结果，我们点击示例中的立方体，可以看到生成成功了，我们看一下结果。

2-2 创建圆柱面

创建圆柱面我们可以直接参考测试程序中的 CreateCylinder 函数，其核心函数是 CreateCylinder，除此之外，相关重要核心函数还有生成片体的函数 PLRCreateSkinBody。

我们打开 test_create 项目看一下 CreateCylinder 函数，首先，我们获取一下工厂指针，然后

判断一下获取指针是否失败，接下来我们需要设置一下创建圆柱面所需的参数，看一下生成圆柱面的CreateCylinder 函数的所需参数，它需要局部坐标系、圆柱半径、轴线起点、轴线终点、起始角、终止角，除了局部坐标系以外都是都是基础类型，我们可以直接设置，局部坐标系我们需要单独在创建一下，我们设置圆柱半径为 10，轴线起点位置为 0，轴线重点位置为 50，起始角为 0 度，终止角为360 度，也就是一个完整的圆柱面，然后我们设置局部坐标系，看到 set 函数的参数都有标注，我们需要分别设置坐标系的原点和三个方向向量，然后将这些参数传到 CreateCylinder 函数当中，获取了圆柱面的指针，为了能够显示出来，我们还需要将圆柱面变成一个片体，这里使用PLRCreateSkinBody 函数，这个函数可以直接传入圆柱面的指针获取结果，然后判断一下获取指针是否失败，最后呢，我们清空一下界面中的实体对象，再将 PLRBody 实体指针添加到可视化界面当中，我们看一下结果。

2-3 创建圆环面

创建圆环面我们可以直接参考测试程序中的 CreateTorusPillar 函数，其核心函数是 CreateTorus。

我们打开 test_create 项目看一下，进入 CreateTorusPillar 函数，首先，我们获取一下工厂指针，然后判断一下获取指针是否失败，接下来我们需要设置一下创建圆环面所需的参数，看一下生成圆环面的 CreateTorus 函数，它需要的参数是局部坐标系、主环半径、主环起始角、主环终止角、次环半径、选择保留的环侧，除了局部坐标系以外都是基础类型，我们可以直接设置，首先我们先创建坐标系原点（0,0,0），然后以该点为原点创建单位坐标系，设置主环半径为 80，起始角为 0 度，终止角为 360 度，次环半径为 40，保留环侧标志为真，将这些参数传给 CreateTorus 函数，获取圆环面的指针，为了能够显示出来，我们还需要将圆环面变成一个片体，使用 PLRCreateSkinBody 函数，然后判断一下获取指针是否失败，最后我们清空一下界面中的实体对象，再将 PLRBody 实体指针添加到可视化界面当中，我们看一下结果。

2-4 创建球面

创建球面我们可以直接参考测试程序中的 CreateSphere 函数，其核心函数是 PLRCreateSolidSphere。

我们打开 test_create 项目看一下，进入 CreateSphere 函数，首先，我们获取一下工厂指针，然后判断一下获取指针是否失败，之后我们定义一下球面的球心为（0,0,0），然后使用 PLRCreateSolidSphere 函数来创建球体，我们看一下这个函数，它的参数是球心，球的半径，那把半径设为 10，获取球面的 PLRBody 指针，然后判断一下指针结果是否失败，最后清空一下界面中的实体对象，再将球面的 PLRBody 实体指针添加到可视化界面当中，我们看一下结果。

2-5 创建棱锥

创建棱锥我们可以直接参考测试程序中的 CreatePyramid 函数，其核心函数是 PLRCreateSolidPyramid，除此之外，其他重要核心函数还有生成线框的函数 PLRCreateWireBody 和生成线段的函数 CreateLine。

我们打开 test_create 项目看一下，进入 CreatePyramid 函数，写代码之前我们先构思一下如何创建棱锥，我们先看一下创建棱锥的函数 PLRCreateSolidPyramid，它的输入参数包括一个底面和一个顶点，而如何创建一个底面，我们考虑先创建一条线段，然后运用线拉伸成面，然后看一下拉伸函数 CreateExtrude，它实际调用的是核心拉伸函数 PLRCreateTopOpExtrude，这个函数需要一个轮廓和一个方向和拉伸的起始和终止位置，CreateExtrude 这个函数是一个为了方便而在 test_create 中将拉伸核心函数包装起来的函数，具体使用方法可以参考后面的拉伸章节，那怎么创建这个轮廓，我们考虑使用 PLRCreateWireBody 函数,这个函数可以把几何对象 PLRCurve 转化为拓扑对象 PLRBody，那想获取 PLRCurve 对象，这里我们使用 CreateLine 函数，它的返回值是 PLRCure 的子类能够符合要求，那我们这下就可以开始写代码了，首先，我们获取一下工厂指针，然后判断一下获取指针是否失败，这个函数可以把那我们先定义边的左端点为(25, 25, 0)，右端点为(-25, 25, 0)，定义棱锥的顶点为(0, 0, -50)，定义拉伸方向为（0,-1,0），然后使用 CreateLine 函数将左右端点相连生成一条线段，将 PLRLine 的指针转换为它的父类指针 PLRCurve，使用 PLRCreateWireBody 函数将线段转为 PLRBody 对象,然后使用 CreateExtrude 沿定义好的方向从 0 拉伸到 50，从拉伸结果中获取底面的 PLRFace 对象，并传入创建棱锥的函数 PLRCreateSolidPyramid 获得结果，然后判断一下指针结果是否失败，最后清空一下界面中的实体对象，再将棱锥的 PLRBody 实体指针添加到可视化界面当中，我们看一下结果。

2-6 拉伸

拉伸我们可以直接参考测试程序中的 CreateExtrudesPointToLine 对应点拉伸，CreateExtrudesLineToFace 对应线拉伸，CreateExtrudesFaceToBody 对应面拉伸，拉伸的核心函数是 PLRCreateTopOpExtrude，在测试程序中为了方便使用又包装了一层函数为 CreateExtrudesOperator。

我们打开 test_create 项目看一下,这三种拉伸都用的一个核心函数即 PLRCreateTopOpExtrude，我们看一下这个函数，它需要轮廓，拉伸的起始位置和终止位置，由于调用方式没有区别，所以为了在三个函数中统一使用，将相关操作包装为 CreateExtrudesOperator，它的输入参数与 PLRCreateTopOpExtrude 函数没有区别，那我们看一下这个函数的实现，首先判断一下传入参数是否有空，然后将参数给到 PLRCreateTopOpExtrude 函数创建拉伸操作指针，注意这里像拉伸这样的拓扑操作都需要调用 Run 函数后才会拉伸成功，而且调用 GetResult 函数方法才会获得拓扑操作的结果，而且由于拓扑操作指针与实体对象指针不同，并不是由工厂统一管理，所以使用完毕之后需要删除，不然会出现内存泄漏,这里介绍完后后续章节将不再赘述,介绍完 CreateExtrudesOperator 函数后，我们先看一下点拉伸 CreateExtrudesPointToLine，先创建一个拓扑的（0,0,0）点对象，使用 CreatePointBody 函数，然后从点（0,0,0）沿（1,0,0）方向从位置 10 拉伸到 50 进行拉伸获取线的对象，接着我们看一下线拉伸 CreateExtrudesLineToFace，首先创建一条从（0,0,0）到（50,0,0）的线段，然后使用 PLRCreateWireBody 函数将线段转换为拓扑 PLRBody 对象，在将线段沿（0，1,0）方向从 15 到 50 位置拉伸成面，最后我们看一下面拉伸 CreateExtrudesFaceToBody，首先要构造面，这里我们使用封闭曲线生成面的方法创建三角面进行拉伸，我们先创建三角面的三个顶点(0, 0, 0)，(0, 0, 20)，(10, 10, 0)，使用 CreateLine 函数两两创建线段，将三条线段放在线段数组中传入 PLRCreateSkinBody 函数获取片体，然后将片体沿（0，1，0）方向从 20 位置到 60 位置进行拉伸，那这样就完成了，我们看一下结果。

2-7 厚曲面

厚曲面我们可以直接参考测试程序中的加厚平面 CreateThickPlaneSurface 函数，其核心函数为 PLRCreateTopOpThickenSkin，为了使用方便测试程序中又包装了一层函数为 CreateThickSurfaceOperator。

我们打开 test_create 项目看一下，我们先看一下厚曲面的核心函数，它的核心函数是

PLRCreateTopOpThickenSkin，我们看一下这个函数，它需要厚曲面的面，加厚起始位置，加厚终止位置，加厚方向倍数，和拉伸类似，其操作流程可以参考教程中的拉伸那一章，我们将这个操作包装为 CreateThickSurfaceOperator 函数，它的输入参数和 PLRCreateTopOpThickenSkin 没有区别，这里除了需要厚曲面的面以外都是基本函数，所以我们需要首先生成厚曲面的面，那么我们从头看一下加厚平面 CreateThickPlaneSurface 的函数实现，首先获取工厂指针，判断获取成功与否，创建四个点（0,0,0），（0,10,0），（0,0,15），（0,10,15），按顺序两两相连使用 CreateLine 函数生成线段，创建数组填入线段指针，通过 PLRCreateSkinBody 函数通过四条封闭的线段创建包含一个面的片体，然后使用 CreateThickSurfaceOperator 函数对这个面做从 8 到 15 的位置做厚曲面，最后呢，我们清空一下界面中的实体对象，再将 PLRBody 实体指针添加到可视化界面当中，我们看一下结果。

2-8 加厚

加厚我们可以直接参考测试程序中的加厚 ThicknessCubeToCuboidPLR 函数，其核心函数是 PLRCreateTopOpSolidThicken，除此之外，还有一个重要函数是添加加厚面函数 Append。

我们打开 test_create 项目看一下，我们先看一下这个核心函数 PLRCreateTopOpSolidThicken，它需要加厚的实体，那我们需要创建一个立方体，那我们从头看一下加厚的示例，首先获取工厂指针，判断获取成功与否，创建立方体的原点（0,0,0）、x 轴端点（100,0,0）、y 轴端点（0,100,0）、z 轴端点（0,0,100），然后使用 PLRCreateSolidCube 函数根据 4 个端点创建立方体，并判断获取立方体指针成功与否，然后获取需要加厚的面，将面添加到一个 vector 容器中，然后使用 PLRCreateTopOpSolidThicken 创建加厚操作指针，并且调用 Append 函数把包含加厚的面的 vector 容器，加厚 100 厚度填入其中，运行 Run 函数，通过 GetResult 函数获取加厚后的实体，为了进行对比，我们对加厚前的实体进行坐标变换，声明变换向量为（100,100,100），然后调用 PLRCreateTopOpTransform 函数进行坐标变换，运行 Run 函数，然后通过 GetResult 函数获取坐标变换的实体，最后呢，我们清空一下界面中的实体对象，再将 PLRBody 实体指针和坐标变换的实体指针添加到可视化界面当中，我们看一下结果。

2-9 旋转

旋转我们可以直接参考测试程序中的由线旋转成面 CreateLineToPlane 的函数和

由面旋转成体 CreatePlaneToSolid 的函数，其核心函数是 PLRCreateTopOpRevol。

我们打开 test_create 项目看一下，我们先看一下旋转的核心函数 PLRCreateTopOpRevol，它需要旋转的轮廓、旋转轴、开始角度、结束角度，除了基础类型以外，我们还需要创建旋转的轮廓和旋转轴，那么我们先从由线旋转成面 CreateLineToPlane 的函数开始看，先通过 CreateCircle 创建一个 0 到 180 度的半圆的轮廓，然后通过 PLRCreateWireBody 函数创建一个包含一条线的线框，然后设置局部坐标系，将线框、局部坐标系，从 0 旋转到 180 度传入到 PLRCreateTopOpRevol 函数当中，然后执行旋转，并获取旋转结果的 PLRBody 对象，释放旋转操作指针，为了做一个对比，将线框和旋转对象分开，我们对旋转结果做坐标变换，我们使用 SetSymmetry 设置一个镜像面，然后将执行坐标变换，并获取坐标变换的结果，释放坐标变换指针，最后呢，我们清空一下界面中的实体对象，再将 PLRBody 实体指针和坐标变换的实体指针添加到可视化界面当中，这样由线旋转成面就完成了。再看一下由面旋转成体的 CreatePlaneToSolid 函数，与由线旋转成面类似都是用的是旋转的核心函数 PLRCreateTopOpRevol，只是传入的轮廓从线变成了面，要先要生成三角面，我们看一下具体实现，我们先定义三角面的三个点（0，0，0），（50，0，0），（0，30，0），然后通过 CreateLine 函数两两创建线段，并存入指针数组中，并调用 PLRCreateSkinBody 函数将三条封闭曲线创建包含一个面的片体，接下来的步骤与由线旋转成面的操作类似，为了进行对比用坐标变换将三角面进行位置变换，然后通过旋转获取旋转的面的 PLRBody 结果，我们看一下结果。

2-10 倒角

倒角我们可以直接参考测试程序中的立方体倒角 CreateChamferCuboidExample 和圆柱体倒角 CreateChamferCylinderExample，其核心函数是 PLRCreateTopOpChamfer，在测试程序中为了方便又包装了一层函数为 CreateEdgeChamfer 函数，除此之外，还有一个重要的核心类 PLRChamferParams 倒角参数类。

我们打开 test_create 项目看一下，这两个倒角示例都使用了一个核心函数即 PLRCreateTopOpChamfer 创建倒角操作，我们看一下这个函数，输入参数需要倒角实体的指针，再看一下倒角操作的 Append 函数,它需要的输入参数为倒角参数对象,而倒角参数类 PLRChamferParams，它的构造需要倒角的参考面的指针列表，倒角类型，倒角类型的第一个参数和第二个参数，以及传播模式，具体第一个参数和第二个参数是什么可以参考倒角类型的注释来看，不同的类型参数的代表意义是不同的，传播模式一般使用默认的即可，剩下要创建的非基础类型只剩需要倒角的参考面的指针

列表了，为了在两个示例中统一使用，将倒角部分的操作包装为 CreateEdgeChamfer 函数，它的输入参数与倒角操作和倒角参数的创建所需的参数没有区别，那我们看一下这个包装函数的具体实现，首先判断一下传入的参数是否为空,然后根据传入的倒角参数类的参数创建倒角参数 PLRChamferParams 类的对象，然后判断一下创建的成功与否，然后传入需要倒角的实体对象然后使用 PLRCreateTopOpChamfer 函数创建倒角操作指针，判断一下倒角操作指针是否创建成功，然后调用 Append 将创建好的倒角参数对象传递给倒角操作指针，调用 Run 函数执行倒角操作，然后使用 GetResult 函数获取倒角操作结果，最后将倒角操作指针和倒角参数指针都释放掉，注意这两个操作相关的指针工厂并不管理，工厂只管理模型实体，最后返回倒角结果，那看完包装函数，我们看一下具体的示例，先看立方体倒角函数 CreateChamferCuboidExample，首先获取工厂指针，并判断工厂指针获取失败与否，然后调用 PLRCreateSolidCube 函数根据输入的四个点（0,0,0），（20,0,0），（0,20,0），（0,0,20）创建需要进行倒角的立方体，然后为了进行对比倒角结果，我们在创建一个相同大小的参照立方体，接下来选出一条边来进行倒角，然后选择倒角类型为 kPLRTopd1a2，即要倒角的边与生成的倒角面和参考面的交点之间的距离为 5,生成的倒角面和其他面之间的夹角为 60 度，将这些参数传入 CreateEdgeChamfer 函数当中获得倒角结果的 PLRBody 对象，最后呢，我们清空一下界面中的实体对象，再将 PLRBody 实体指针和坐标变换的实体指针添加到可视化界面当中，这样立方体倒角就完成了。我们在看一下圆柱体倒角，与立方体倒角类似，都是使用的 CreateEdgeChamfer 函数进行倒角，只是使用的 PLRCreateSolidCylinder 函数创建圆柱体后进行的倒角，并且对圆柱的上下两面的圆形的边都进行了倒角，我们看一下结果。

2-11 抽壳

抽壳我们可以直接参考测试程序中的立方体抽壳 CreateShellCuboidExample 和圆柱体抽壳 CreateShellCylinderExample，其核心函数是 PLRCreateTopOpSolidShell，在测试程序中为了方便又包装了一层函数为 CreateShellOperator 函数，除此之外，还有一个重要的核心函数添加移除面函数 Append。

我们打开 test_create 项目看一下，这两个抽壳示例都使用了一个核心函数即 PLRCreateTopOpSolidShell 函数,我们看一下这个函数,它的输入参数需要抽壳操作实体的指针，抽壳的内侧厚度，抽壳的外部厚度，再看一下抽壳的 Append 函数，它需要输入抽壳的移除面，这些

参数非基本类型的参数有需要抽壳操作的实体的指针和抽壳的移除面，这两个需要创建，而为了在两个示例中统一使用，将抽壳部分的操作包装为 CreateShellOperator 函数，而它的输入参数与抽壳操作的所需的参数没有区别，那接下来我们看一下包装函数的具体实现，首先判断一下传入的参数是否为空，然后将传入的抽壳参数赋予 PLRCreateTopOpSolidShell 函数创建抽壳操作指针，之后判断一下抽壳操作指针创建成功与否，使用 Append 函数添加抽壳移除面，用 Run 函数执行抽壳操作并通过 GetResult 函数获取抽壳结果，最后释放一下抽壳操作指针，返回抽壳结果实体指针，那看完包装函数，我们看一下具体示例。我们先看一下立方体抽壳 CreateShellCuboidExample 函数，首先我们获取工厂指针，判断指针获取成功与否，使用 PLRCreateSolidCube 函数根据输入的四个点（0,0,0），（20,0,0），（0,20,0），（0,0,20）创建立方体实体对象指针，为了进行对比，我们再创建一个参照立方体，之后获取需要抽壳的移除面，然后传入参数包括创建的立方体实体对象指针，获取的移除面，设置内部抽壳厚度为-2，外部抽壳厚度为-1，执行抽壳的包装函数 CreateShellOperator 获取抽壳结果的 PLRBody 对象，最后呢，我们清空一下界面中的实体对象，再将 PLRBody 实体指针和作对比的参照实体指针添加到可视化界面当中，这样立方体抽壳就完成了，与之类似，圆柱体抽壳 CreateShellCylinderExample 函数的流程也一样，只是抽壳的实体指针的创建是通过 PLRCreateSolidCylinder 函数以（0,0,0）到（0,0,50）为轴，半径为 20 创建的实体对象，那我们看一下结果。

2-12 面分割

面分割我们可以直接参考测试程序中的面分割 CreateSimpleSplit，其核心函数是 PLRCreateTopOpSplit。

我们打开 test_create 项目看一下，我们先看一下核心函数 PLRCreateTopOpSplit，它的输入的参数需要分割的实体、保留的部分、用来分割的实体，其中分割的实体和用来分割的实体需要创建，我们进入 CreateSimpleSplit 看一下，首先我们获取工厂指针，判断指针获取成功与否，然后创建分割圆和参照圆的所在的平面，然后通过 CreateCircle 创建分割圆和参照圆，用 PLRCreateSkinBody 函数创建包含圆的片体，之后通过 CreateLine 函数将（-20,0,0）和（10,0,0）创建分割线，同时创建参照线，并通过 PLRCreateWireBody 将分割线和参照线创建为线框，然后传入分割线和分割圆调用 PLRCreateTopOpSplit 函数创建分割操作指针，生成分割操作指针后调用 Run 方法执行分割操作，再调用 GetResult 方法获得分割的 PLRBody 结果，然后释放分割操作指针，最后清空一下界面中的实体

对象，再将分割结果、分割线、参照实体和参照的分割线添加到可视化界面当中，我们看一下结果。

2-13 体分割

体分割我们可以直接参考测试程序中的立方体分割 CreateCubeSplit 和圆柱体分割 CreateCylinderSplit，其核心函数是 PLRCreateTopOpSolidSplit，除此之外，其他重要的核心函数是添加分割对象的函数 SetSplit。

我们打开 test_create 项目看一下,先看一下体分割的核心函数即 PLRCreateTopOpSolidSplit，我们看一下这个函数，输入的参数包括需要被分割的实体、保留的部分，再看一下分割操作指针的 SetSplit 函数，它需要分割的实体对象，相关函数一共有被分割实体和分割的实体两个非基本类型要被创建。回到立方体分割 CreateCubeSplit 的示例看一下，首先获取工厂指针，判断获取成功与否，而为了创建被分割的立方体和要分割的实体，这里使用拉伸来实现，定义四个点与两个方向向量，调用 CreateLine 函数将前两个点创建为线段，调用 PLRCreateWireBody 函数创建包含该线段的片体，用以拉伸为一个面，再继续将面拉伸成立方体，接下来同样的从后两个点的线段开始拉伸为一个面，这里就是重点了，传入立方体，并选择保留的部分，调用 PLRCreateTopOpSolidSplit 函数创建分割操作指针，判断分割操作指针创建成功与否，调用 SetSplit 函数传入要分割的面，使用 Run 函数执行分割操作，GetResult 函数获取分割后的 PLRBody 实体，再释放分割操作指针，为了进行对比，调用坐标变换的 PLRCreateTopOpTransform 函数，设置平移量为（70,0,0），创建坐标变换后的实体，最后清空一下界面中的实体，将切割后的实体和参照实体添加到可视化界面当中。再看一下圆柱体分割 CreateCylinderSplit 示例，与立方体分割类似，只是圆柱体分割是通过 PLRCreateSolidCylinder 函数，传入两个点和半径 15 创建的圆柱体，通过 CreatePlane 函数传入三个点创建的分割面，其他的操作都类似，我们看一下结果。

2-14 修剪

修剪我们可以直接参考测试程序中的修剪 CreateTopTrim，其核心函数是 PLRCreateTopOpTrimSkin。

我们打开 test_create 项目看一下，先看一下这个核心函数 PLRCreateTopOpTrimSkin，输入的参数包括被修剪的实体，用来修剪的实体，保留的部分，其中被修剪的实体和用来修剪的实体是非基

础类型，需要被创建。我们回到修剪示例函数 CreateTopTrim 当中，首先获取工厂指针，判断获取成功与否，然后创建两个平面，使用 CreateCircle 函数创建两个半径为 10 的圆形线，再使用 PLRCreateSkinBody 函数将两条圆形线变成包含一个面的片体，一个作为被修剪的实体，一个作为对比参照使用，之后通过 CreateLine 函数将四个点连成两条线段，使用 PLRCreateWireBody 函数创建包含线段的线框，在使用 CreateExtrude 的拉伸包装函数创建修剪的长方形面和参照的长方形面，接下来调用 PLRCreateTopOpTrimSkin，传入被修剪的实体，修剪的实体，选择保留被修剪体的右边和修剪体的左边来创建修剪操作指针，运行 Run 函数来执行修剪操作，然后调用使用 GetResult 函数获取结果，最后清空一下界面中的实体，将切割后的实体和参照实体添到可视化界面当中，我们看一下结果。

2-15 拔模

拔模我们可以直接参考测试程序中的立方体拔模 CreateRectangleDraftShow 和圆柱体拔模 CreateCylinderDraftShow，其核心函数是 PLRCreateTopOpDraft，其它重要相关核心函数是添加拔模参数函数 AddDraftParams，拔模与其它的操作不一样，更加复杂，除了拔模操作类以外还有多个相关类，包括拔模角度类 PLRDraftAngle，可变角度拔模参数类 PLRDraftVariableAngleData，拔模参数类 PLRDraftParams。

我们打开 test_create 项目看一下,先看一下拔模的核心函数 PLRCreateTopOpDraft，这个函数需要拔模实体，之后拔模操作的 AddDraftParams 函数需要拔模参数对象，获取拔模参数较为复杂，所以将整个过程包装为 GetDraftDomain 函数，我们看一下它的实现，传入被拔模的实体对象和拔模角度，它会先取传入的拔模实体的一个面作为拔模面，另一个面作为中性面，对拔模角度赋值，通过 GetBoundingBoxUV 函数获取中性面的包围盒，然后获取包围盒中心位置的法矢作为拔模方向，传入拔模面和角度生成 PLRDraftAngle 对象，再通过 PLRDraftAngle 生成可变角度拔模参数 PLRDraftVariableAngleData，最后通过拔模方向，中性面，可变角度拔模参数生成拔模参数 PLRDraftParams。总结一下，即想拔模需要先创建拔模参数 PLRDraftParams 对象，想创建拔模参数，需要给出中性面和拔模方向以及可变角度拔模参数 PLRDraftVariableAngleData 对象，想生成可变角度拔模参数需要先生成 PLRDraftAngle 拔模角度对象，想生成拔模角度对象需要传入参数

拔模面和拔模角度，这样拔模参数就创建成功了。捋顺了拔模操作的整个流程之后我们看一下立方体拔模 CreateRectangleDraftShow 函数，首先获取工厂指针，判断获取成功与否，通过 CreateRectangle 创建一个立方体，这个函数是一个包装函数，会创建一个标准的立方体，然后调用包装函数 GetDraftDomain 创建拔模参数，设置拔模角度为-10，之后调用核心函数 PLRCreateTopOpDraft 创建拔模操作指针，判断拔模指针创建成功与否，调用拔模操作的 AddDraftParams 函数添加创建的拔模参数，然后调用 Run 函数执行拔模操作，使用 GetResult 函数获取拔模结果，完成后释放拔模操作指针和拔模参数指针，再调用 CreateRectangle 函数创建一个标准立方体作为对比，最后清空一下界面中的实体，将切割后的实体和参照实体添加到可视化界面当中。在看一下圆柱体拔模 CreateCylinderDraftShow，与立方体拔模类似，只是调用了 CreateDraftBodyToCylinder 生成了圆柱体进行的拔模，我们看一下结果。

2-16 倒圆角

倒圆角我们可以直接参考测试程序中的对立方体倒圆角 CreateRectangleFillet 和对圆柱体倒圆角 CreateCylinderFillet，其核心函数是 PLRCreateTopOpFillet，其它重要相关核心函数是添加倒圆角参数函数 Append，倒圆角操作与其它操作不一样，更加复杂，除了倒圆角操作类以外还有多个相关类，包括边导圆参数类 PLREdgeFilletParams，可变圆角半径类 PLRFilletVariableRadius。

我们打开 test_create 项目看一下，先看一下倒圆角的核心函数 PLRCreateTopOpFillet，这个函数需要倒圆角实体对象，之后倒圆角操作的 Append 函数需要边倒圆参数 PLREdgeFilletParams 对象，我们看一下，该类有两种构造函数，这两种构造函数都需要 PLRFilletVariableRadius 可变圆角半径对象，只是数量不同，代表是固定半径倒圆角还是可变半径倒圆角，可变圆角半径对象所需参数则是基本类型和倒圆角的边。总结一下，即想倒圆角，需要先创建可变圆角半径对象 PLRFilletVariableRadius 定义要倒圆角的半径和边，然后通过一个或多个可变圆角半径对象构造 PLREdgeFilletParams 边倒圆参数对象，然后调用 PLRCreateTopOpFillet 创建倒圆角操作对象，使用 Append 函数添加边倒圆对象完成倒圆角。捋顺了倒圆角操作的整个流程之后我们看一下立方体倒圆角 CreateRectangleFillet 函数，首先获取工厂指针，判断获取成功与否，通过包装函数 CreateRectangle 创建一个标准立方体，然后在立方体中选一条边作为倒圆角的边，简单设计一下让

倒圆角的半径变化两次，并设定每次变化的半径，之后调用核心函数 PLRCreateTopOpFillet 创建倒圆角操作指针，调用倒圆角操作的 Append 函数添加边倒圆参数对象，调用 Run 函数执行倒圆角操作，使用 GetResult 函数获取倒圆角结果，完成后释放边倒圆参数对象和倒圆角操作指针，再调用 CreateRectangle 函数创建一个标准立方体作为对比，最后清空一下界面中的实体，将倒圆角后的实体对象和倒圆角前的是实体对象添加到可视化界面当中。再看一下对圆柱体倒圆角 CreateCylinderFillet，与立方体倒圆角类似，只是边倒圆参数类构造时使用了一个可变半径倒圆角对象，即固定半径倒圆角，我们看一下结果。

2-17 投影

投影我们可以直接参考测试程序中的点到线的投影 CreatePointOnCurveProjectionExample，点到面的投影 CreatePointOnSurfaceProjectionExample，线到面的投影 CreateCurveOnSurfaceProjectionExample，这三个示例函数使用的核心函数不同，但又有相似之处，即同一函数名参数不同的重载，所以一起来看，该核心函数为 PLRCreateProjection，但其核心函数返回值不同，分别对应点到线的投影类 PLRGeoOpProjectionPtCrv、点到面的投影类 PLRGeoOpProjectionPtSur、线到面的投影类 PLRGeoOpProjectionCrvSur。

我们打开 test_create 项目看一下，我们先看投影的核心函数 PLRCreateProjection，对于三种投影方式都有两个共同的参数为投影的方向向量 PLRMathDirection 对象和 PLRSkillValue 枚举值，这两个值可以直接使用默认的，如有修改需求可以查看注释，不同的参数中，点到线的投影的输入参数为 PLRPoint 投影点对象和 PLRCurve 投影曲线对象，返回值为点到曲线的投影类 PLRGeoOpProjectionPtCrv 对象；点到面的投影的输入参数为 PLRPoint 投影点对象，PLRSurface 投影面对象，返回值为点到面的投影类 PLRGeoOpProjectionPtSur 对象；线到面的投影的输入参数为 PLRCurve 投影曲线对象，PLRCrvLimits 投影曲线的参数区间对象，PLRSurface 投影曲面对象，PLRSurLimits 投影曲面的参数范围对象，返回值为线到面的投影类 PLRGeoOpProjectionCrvSur 对象。

以点到线的投影中的包装函数 CreatePointOnCurveProjection 为例看一下函数的使用，首先通过传入的点和线调用 PLRCreateProjection 函数生成投影操作指针，然后根据获取投影操作的点集数量函数 GetNumberOfPoints 的结果是否为零判断是否存在投影，如果存在投影，调用 NextPoint 函数使得内部指针指向下一个结果点，然后调用 GetCartesianPoint 函数获取投影点的结果，调用 CreatePointBody 函数创建包含投影点的拓扑点，最后释放投影操作指针，点到面的投影中的包装函

数 CreatePointOnSurfaceProjection 和线到面的投影中的包装函数 CreateCurveOnSurfaceProjection 也是类似的，就不详细说了，可以直接看代码中的注释。我们在以点到线的投影为例从头看一下投影的整个过程，首先获取工厂指针，判断成功与否，创建 3 个点，分别为（0,0,0），（1,1,0），（2,4,0），以这三个点调用 CreateSplineCurve 创建三次样条曲线的函数生成三次样条曲线，再使用 PLRCreateWireBody 函数创建包含样条线的线框，使用 CreateCartesianPoint 创建投影点，CreatePointBody 函数创建包含投影点的拓扑点，需要的参数创建完毕后调用点到线的投影的包装函数 CreatePointOnCurveProjection 获取投影结果，最后清空一下界面中的实体，将投影实体对象和投影前的实体对象添加到可视化界面当中。点到面的投影和线到面的投影与之类似，只是参数有细微差别，最后我们看一下结果。

2-18 阵列

阵列我们可以直接参考测试程序中的实体的矩形阵列 CreateRectanglePatternFromBodyExample,实体的圆形阵列 CreateCirclePatternFromBodyExample，曲面的矩形阵列 CreateRectanglePatternFromBodyByFaceBodyExample，曲面的圆形阵列 CreateCirclePatternFromBodyByFaceBodyExample，其核心函数为 PLRCreateTopOpPattern 函数，除此之外，还有其他重要的核心函数 SetRectangularPattern 设置为矩形阵列和 SetCircularPattern 设置为圆周阵列。

我们打开 test_create 项目看一下，先看一下其核心函数 PLRCreateTopOpPattern，它的输入的参数包括待阵列的实体对象，用于与阵列后的体进行布尔运算的实体，布尔操作类型，其中待阵列的实体对象，用于与阵列后的体进行布尔运算的实体对象需要生成，所以以实体的矩形阵列 CreateRectanglePatternFromBodyExample 这个函数为例看一下使用，首先获取工厂指针，判断成功与否，之后以（-500，-500，-500）为原点，原点到（-500，-500,500）为 x 轴，原点到（-500，500，-500）为 y 轴，原点到（500，-500，-500）为 z 轴创建立方体，这个立方体将作为需要进行阵列的实体对象，然后以（0,0，-500）（0,0，500）之间的连线做轴线，半径 20 生成圆柱体，调用 PLRCreateTopPattern 函数创建在立方体上做阵列的布尔差运算，之后调用 SetRectangularPattern 函数设置矩形阵列沿（0,1,0）（1,0,0）方向相邻 60 距离每行每列设置 20 个实体，设置完成后调用 Run 函数执行阵列操作，GetResult 函数获取阵列结果实体对象，获取到阵列结果后释放阵列操作指针，最后清空一下界面中的实体，将阵列结果对象添加到可视化界面当中。实体的圆形阵列、曲面的

矩阵阵列、曲面的圆形阵列与之类似，只是实现过程 有细微差别，最后我们看一下结果。

2-19 等距

等距我们可以直接参考测试程序中的平面等距曲面 PlaneSurfaceEquidistanceShow、锥面等距曲面 ConeSurfaceEquidistanceShow、样条面等距曲面 SplineSurfaceEquidistanceShow，其核心函数是曲面等距 PLRCreateGeoOpOffsetSur。

我们打开 test_create 项目看一下，这几个示例函数使用的核心函数相同，且使用方法类似，所以定义了 CreateSurfaceEquidistance 函数来对核心函数进行了包装，我们先看一下这个函数，输入参数参数包括原片体和等距距离，在进入函数内部看一下等距的操作流程，首先获取传入片体的曲面，调用 Getlimits 函数获取曲面的参数范围，然后将曲面，等距距离，容差传入 PLRCreateGeoOpOffsetSur 核心函数中，创建等距曲面操作指针，然后调用 SetLimits 设置参数范围与原曲面相同，调用 Run 函数执行等距操作，调用 GetOffsetSurface 函数获取等距结果，然后使用 PLRCreateSkinBody 创建包含等距结果曲面的片体，然后释放等距曲面的操作指针，这里需要注意与其他拓扑操作的不同点，一个是需要设置曲面的参数范围，一个是获取结果的函数不是 GetReasult 而是 GetOffsetSurface。那么接下来我们看一下具体的实现，以平面等距曲面 PlaneSurfaceEquidistanceShow 函数为例，首先获取工厂指针，判断成功与否，调用 CreatePlaneBody 函数创建标准平面，然后调用 CreateSurfaceEquidistance 根据传入的标准曲面做等距操作获取结果曲面，最后清空一下界面中的实体，将标准曲面对象和等距结果曲面对象添加到可视化界面当中。锥面等距曲面和样条面等距曲面与之类似，最后我们看一下结果。

2-20 基本扫掠

基本扫掠即一条引导线一个轮廓的扫掠我们可以直接参考测试程序中的 CreateSweepExample01-12，其核心函数是创建扫掠操作 PLRCreateFrFTopologicalSweep。

我们打开 test_create 项目看一下，我们先找其中最典型的 CreateSweepExample01 看一下，首先获取工厂指针，判断获取成功与否，声明导线的指针和轮廓的指针，调用 CreateLine 用（0,0,0）和（70,0,0）的连线创建 PLRCurve 曲线指针，调用 PLRCreateWireBody 创建包含导线的 PLRBody 线

框，创建完导线后再创建轮廓，依旧是调用 CreateLine 用（0,0,0）和（0,50,0）的连线创建 PLRCurve 曲线指针，调用 PLRCreateWireBody 创建包含导线的 PLRBody 线框，两个都创建完成后调用核心函数 PLRCreateFrFTopologicalSweep 传入导线和轮廓创建扫掠操作指针，调用 Run 函数执行扫掠，调用 GetResult 函数获取扫掠结果，之后释放扫掠操作指针，为了与导线和轮廓区分开来，将扫掠结果进行坐标变换，这里使用 PLRCreateTopOpTransform 函数创建坐标变换操作指针，SetTranslation 函数设置平移量，调用 Run 函数执行坐标变换，GetResult 函数获取坐标变换结果，释放坐标变换操作的指针，最后清空一下界面中的实体，将导线和轮廓对象以及扫掠结果对象添加到可视化界面当中。其他的函数与 CreateSweepExample01 的区别仅是变换导线和轮廓，操作没有变化，最后我们看一下结果。

2-21 其他方式扫掠

其他方式扫掠包括显式扫掠、直线扫掠、圆弧扫掠、二次曲线扫掠，我们先看对应的核心函数，显式扫掠的核心函数是 PLRCreateFrFTopologicalSweep，直线扫掠的核心函数是 PLRCreateFrFTopologicalSegmentSweep，圆弧扫掠的核心函数是 PLRCreateFrFTopologicalCircleSweep，二次曲线扫掠操作的核心函数是 PLRCreateFrFTopologicalConicSweep。

我们打开 test_create 项目看一下，我们先看一下核心函数的区别，通过输入参数可以发现除了显式扫掠之外，其他三种扫掠方式的输入参数相同，只需要输入导线，而显示扫掠则不但需要导线，还需要轮廓，并需要指定扫掠类型，事实上显式扫掠可以实现另外三种扫掠的功能，而反过来则不能，另外三种扫掠方式可以看做是规定了轮廓类型的显式扫掠，所以我们就只讲最通用的显示扫掠的操作。具体的实现可以参考测试程序中的显示扫掠的 CreateSweepExplicitTwoGuides 和 CreateSweepExplicitDraft，直线扫掠的 CreateSweepSegmentTwoGuides、CreateSweepSegmentLimitMiddle、CreateSweepSegmentSection，圆弧扫掠的 CreateSweepCircleThreeGuides、CreateSweepCircleTwoPointsRadii、CreateSweepCircleCenterRadii、CreateSweepCircleGuideSection，以及二次曲线扫掠的 CreateSweepConicTwoGuides、CreateSweepConicThreeGuides。我们以显示扫掠的一个轮廓两条导线的 CreateSweepExplicitTwoGuides 函数为例看一下其他方式扫掠的操作过程，首先获取工厂指针，判断获取成功与否，准备根据（0,0,0），（0,0,20），（40,0,0）三个节点的点集形成的三次样条曲

线作为轮廓，根据（0,180,20），（0,300,10）与（0,0,0）三个节点的点集形成的三次样条曲线和根据（40,0,0）（40,350,0）两点形成的线段作为引导线，之后调用 PLRCreateFrFTopologicalSweep 创建显式扫掠操作指针，传入准备轮廓和引导线，扫掠类型选择多截面，用 Run 函数执行扫掠操作，使用 GetResult 获取扫掠结果，并释放扫掠操作指针，为了进行对比扫掠的参数和结果，对扫掠结果进行坐标变换，这里使用了三维坐标变换类 PLRMathTransformation 和 PLRCreateTopOpTransform 创建坐标变换操作函数进行了坐标变换，最后清空一下界面中的实体，将导线和轮廓对象以及扫掠结果对象添加到可视化界面当中。其他的扫掠方式区别在于传入的参数不同，就不再一一介绍，最后我们看一下结果。

第三部分 实例演示

3-1 圆盘

这是圆盘，即零件 19 的最终显示结果，那么如何创建这样的模型呢，先分析一下圆盘的具体形状，再考虑一下创建思路，圆盘的最外层的轮廓是一个扁平的圆柱体，该圆柱体的边有倒角，往里还有一圈空心圆柱体形状的棱，该棱的外边和内部与圆盘连接处有倒角，圆盘的中心处是个圆柱体形状的洞，洞内的中心还有一圈空心圆柱体的槽，除此之外，在圆盘最外边的轮廓和棱之间还有六个均匀排列的圆柱体形状的洞，该洞分两层，上层半径较小，下层半径较大，根据圆盘的形状特点考虑通过创建圆柱体和布尔操作相结合的方式组成基本形状，然后对所需要的边进行倒角，至于六个洞考虑阵列的方式通过布尔差的方式创造出来。接下来看一下具体实现，首先通过 GetFactory 函数获取工厂指针，然后通过 PLRCreateSolidCylinder 函数创建圆柱体，分别创建最外层轮廓的圆柱体、棱的外侧圆柱体、棱的内侧圆柱体、中间的洞的圆柱体、中间洞的棱的圆柱体，外层六个洞的上层圆柱体和六个洞的下层圆柱体，之后使用布尔操作 CreateTopBoolean 将外层六个洞的上层圆柱体和下层圆柱体结合在一起，然后开始倒角，调用 CreateEdgeChamfer 函数对最外层轮廓的圆柱体、棱的外侧圆柱体和棱的内侧圆柱体，洞中间的棱圆柱体进行倒角，之后调用

PLRCreateTopOpPattern 函数对六个洞的圆柱体的结合体进行阵列操作，调用 SetCircularPattern 设置以（0,0,10）到（0,0,1）为轴，切向六十度旋转出 6 个实体对象，所有的图形都处理完毕后，使用布尔的并与差操作对各个圆柱体进行组合加减，并将最终结果添加到界面当中。

3-2 水杯

这是水杯，即零件 59 的最终显示结果，那么如何创建这样的模型呢，先分析一下水杯的具体形状，在考虑一下创建思路，杯子从形状上看，可以分为杯底、杯壁和把手，把手又可以分为两部分，一部分是直的部分，一部分是弯曲的部分，杯底考虑圆柱体倒圆角，杯壁由于是弯曲向外突出的，考虑使用样条线拉伸后旋转 360 度生成，而把手的两部分都考虑使用椭圆轮廓和导线扫掠而成。接下来看一下具体实现，首先通过 GetFactory 函数获取工厂指针，然后通过 PLRCreateSolidCylinder 函数创建杯底圆柱体，厚度 3，半径 22，之后对杯底圆柱体的下边倒圆角，圆角半径为 3，这样杯底就完成了，接下来使用 CreateSplineCurve 函数将杯壁外侧三个特征点（22,0,3），（26,0,21），（30,0,45）相连为三次样条曲线，之后使用 CreateExtrude 函数对该三次样条曲线沿（-1,0,0）拉伸 2 的长度为杯壁的横截面，在调用 PLRCreateTopOpRevol 函数将杯壁的横截面旋转 360 度形成杯壁，下面就开始生成把手，先考虑直的部分，调用 CreateLine 函数从（22.5，0,8）到（39.2,0,21.49）生成把手直的导线，接下来调用 CreateEllipse 函数设置长半轴为 3，短半轴为 1.5 在以（22.5,0,8）为原点，（0,1，0）为 x 轴方向，（0,0,-1）为 y 轴方向生成把手的椭圆横截面，使用 PLRCreateFrFTopologicalSweep 将椭圆横截面沿导线扫掠获取把手直的部分，直的部分创建完成后考虑弯曲的部分，调用 CreateCircle 函数在以（33,0,28）为原点（-1,0,0）为 x 轴方向（0,0,1）为 y 轴方向的平面上以 9 为半径从弧度 1 到弧

度 4 生成圆弧导线，接下来调用 CreateEllipse 函数设置长半轴为 3，短半轴为 1.5 在以（39.2,0,21.49）为原点，（0，1，0）为 x 轴方向，（0，0，-1）为 y 轴方向生成把手的椭圆横截面，最后在使用 PLRCreateFrFTopologicalSweep 将椭圆横截面沿导线扫掠获取把手弯曲的部分，这样水杯的所有部分都已经生成完毕了。

3-3 瓶子

这是瓶子，即零件 71 的最终显示结果，那么如何创建这样的模型呢，先分析一下瓶子的具体形状，在考虑一下创建思路，瓶子从形状看，可以分为瓶壁和瓶底，瓶底考虑正方形面拉伸成长方体即可，瓶壁的上方是个圆形，下方是个方形，考虑使用多截面扫掠，而且瓶壁的弯曲程度比较特殊，在上方和下方的切线都与 z 轴平行，考虑扫掠的时候需要支撑面控制扫掠路线。接下来看一下具体实现，首先通过 GetFactory 函数获取工厂指针，通过 CreateCircle 函数以（0,0,0）为原点，（1,0,0）为 x 轴，（0,1,0）为 y 轴，半径 9 生成瓶口的圆形轮廓，调用 PLRCreateTopOpExtrude 沿 z 轴方向拉伸为瓶壁上方瓶口位置的支撑面，接下来将（-15，-15,30），（15，-15,30），（15,15,30），（-15,15,30）两两相连成为正方形轮廓的四条线段，使用 PLRCreateTopOpJoin 函数将四条线段接合为相连的正方形轮廓，接着同样调用 PLRCreateTopOpExtrude 沿 z 轴方向拉伸为瓶壁下方瓶底位置的支撑面，接下来准备扫掠，但是上方的轮廓和下方轮廓还没有设置闭合点，为了扫掠结果正确，调用 PLRTopInsertClosureVertex 函数将瓶壁的上下轮廓沿闭合点打断为新的轮廓，接下来调用 PLRCreateFrFTopologicalSweep 函数沿瓶壁的上下两个轮廓进行扫掠，并设置支撑面，这样就能获取瓶壁的外边的面，为了获取完整的瓶壁还需要将该面进行加厚，调用 PLRCreateTopOpThickenSkin 函数向内加厚 1.5，这样瓶壁就完成了，然后将瓶底的正方形轮廓使用 PLRCreateSkinBody 函数填充为面，然后沿（0,0,1）方面调用 PLRCreateTopOpExtrude 函数拉伸 6 长度为长方体，最后使用 PLRCreateTopOpBoolean 函数做布尔并操作，将瓶壁和瓶底结合在一起，这样瓶子就生成完毕了。

结束