**2、****VTK编译安装**

学习VTK最好是从下载VTK源码，然后自己编译开始。编译VTK是很简单的一件事情，最重要的是把准备工作做好。本章将会详细演示如何编译安装VTK。

这里提到的VTK开发环境为：Windows 7旗舰版64位操作系统，Visual Studio 2008集成开发环境，CMake 2.8.6，Qt 4.8.2以及VTK 5.10 Release版本。后续所有的程序示例代码都是在这个环境下测试通过的。

**2.1        编译VTK的准备工作**

**首先，你的计算机必须安装了VisualStudio 2008**。如果采用其他集成开发环境(IDE)的，也必须先安装对应的IDE工具。编译64位版本的VTK之前，在安装Visual Studio 2008的时候，需要安装相应的64位组件。

**其次，安装CMake**。CMake下载地址：<http://www.cmake.org/cmake/resources/software.html>。Windows平台下只要下载文件cmake-2.8.6-win32-x86.exe即可。安装完CMake以后，运行界面如图2.1所示。

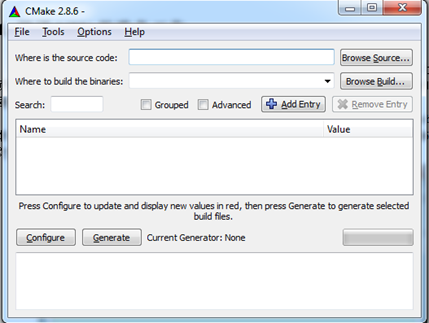


图2.1CMake界面

如果你在这之前没有听说过CMake，也不知道CMake如何使用，先不要着急。假如你实在比较急的话，可以先去Google或者百度一下CMake到底是个什么东西。

接着，如果你想自己编译VTK类库的帮助文档的话，还需要安装Doxygen工具。**这一步不是必须的**。

此外，如果你想使用Tcl、[**Python**](http://lib.csdn.net/base/11)或者[**Java**](http://lib.csdn.net/base/17)等语言开发的话，相应的工具也必须先安装。相关的文件可以从以下页面下载到：Tcl/Tk下载地址：<http://www.tcl.tk/software/tcltk/>；Python下载地址：<http://www.python.org/>；Java下载地址：<http://www.java.com/>。

**最后，确定你想使用哪种工具作为界面开发，比如MFC、Qt、FLTK等等**。以这个系列教程为例，采用Qt作为GUI(用户图形界面)开发工具，那么在安装VTK之前还必须安装Qt，Windows平台下的Qt Opensource编译版本可以从以下页面下载(<http://qt-project.org/downloads/>和<http://code.google.com/p/qt-msvc-installer/downloads/list>)。

换言之，如果你想使用C++作为开发语言，采用Qt作为GUI开发工具，但不想编译VTK的帮助文档(因为已经有现成的，编译帮助文档的时间也会比较长)，那么**在编译VTK之前，只要安装VisualStudio 2008、CMake和Qt即可**。

**2.2        编译VTK的详细步骤**

将下载到的VTK源码文件vtk-5.10.0.zip解压到某个磁盘下，比如，解压到D:\Toolkits\VTK\VTK-5.10。然后，建立一个空的文件夹，叫什么名字无所谓，只要不含有中文即可(为了让文件名看起来规整统一，新建的文件夹命名为VTK-5.10-bin，完整的路径为：D:\Toolkits\VTK\VTK-5.10-bin)。接着解压下载的vtkdata-5.10.0.zip文件，比如解压到：D:\Toolkits\VTK\VTKData (如果你没有下载vtkdata-5.10.0.zip文件，这一步也可以跳过)。

然后，打开CMake(界面如图2.1所示)，在CMake界面上的“Where is the source code”文本框里输入前一步解压vtk-5.10.0.zip之后的路径，即D:\Toolkits\VTK\VTK-5.10。也就是这个文本框应该输入VTK源码目录里最外层的CMakeLists.txt文件所在的路径。接着，在“Where to build the binaries”文本框里输入前一步新建的空文件夹的路径，即D:\Toolkits\VTK\VTK-5.10-bin，这个文本框所输的路径可以跟“Where is the source code”的一样，分开两个不同的路径的好处是后续编译过程生成的文件不会跟VTK的源码混合在一起，避免对源码目录的“污染”，这也是后续内容介绍的CMake的“in-place”和“out-of-place”的区别。

以上两个文本框输入完之后，按“Configure”按钮，会弹出如图2.2所示的对话框，根据你自己的需要以及你计算机已经安装的IDE、编译器等选择适当的选项即可，比如，使用Visual Studio 2008，准备编译64位的VTK，则可以选择“Visual Studio 9 2008 Win64”选项(如果选择错误，要重新选择IDE选项的话，停止CMake的配置或者生成过程，点击CMake的File菜单下的Delete Cache之后，再重新用CMake配置VTK的时候就会重新弹出图2.2的对话框)，然后确定，CMake即开始根据具体的平台环境配置VTK工程(配置时间会因硬件配置的不同而不同，一般需要几分钟时间)。

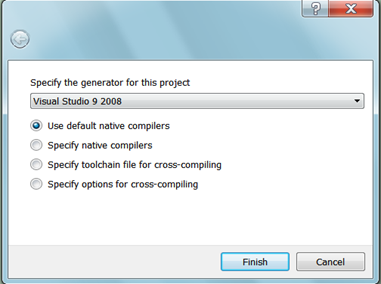


图2.2CMake选择编译环境的对话框

配置完成后，CMake界面如图2.3所示。

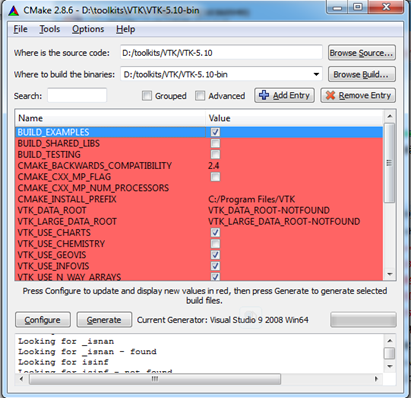


图2.3首次配置完VTK后的界面

关于VTK的一些CMake配置选项的说明：

**BUILD\_EXAMPLES** —默认是关闭的，如果打开这个选项，则会编译VTK例子，同时VTK编译所需的时间也较长，占用的磁盘空间也较大。对于初学者来说，建议把这个选项打开。

**BUILD\_SHARED\_LIBS** —默认是关闭的，意味着VTK是静态编译；如果打开这个选项，则是动态编译。

**“静态编译”与“动态编译”的区别**：静态编译就是在编译的时候把所有的模块都编译进可执行文件(exe)里去，当启动这个可执行文件时所有的模块都已加载进来。动态编译则是编译的时候需要的模块都没有编译进去，一般情况下可以把这些模块都编译成动态链接库DLL，启动程序（初始化）的时候这些模块不会被加载，运行的时候用到那个模块就调用哪个DLL文件。静态链接库编译相当于你带着一个工具包到处跑，遇到有需要的地方不需要周围的环境提供相应的工具，自己用自己工具包的工具就行了，所以当环境发生变化可以尽可能的无视；动态链接库编译相当于不带任何东西，走到哪是哪。这两者的区别显然就是前者重量增加了，即程序的体积会比后者的大。

所以，究竟是用“静态编译”还是“动态编译”，关键看自己的需要。对于VTK初学者而言，所涉及到的工程可能都比较小，建议用“静态编译”，也方便把VTK程序移植到其他没有安装VTK的计算机上运行。本书的VTK类库是采用静态编译。

**BUILD\_TESTING** —默认是打开的，表示会编译VTK的测试程序。VTK里每个类都有对应的程序文件对该类进行测试，对于初学者而言，可以关闭掉这个选项，后续如有需要，可以再打开此选项。

**VTK\_DATA\_ROOT** —指向VTKData所在的路径，即第1章里下载的文件vtkdata-5.10.0.zip解压后的路径。一般情况下，VTK能够自动搜索到，如果没有搜索到可以手动输入，在运行VTK的例子时需要用到这些数据。

**CMAKE\_INSTALL\_PREFIX** —这个选项表示VTK的安装路径，默认的路径是：C:/Program Files/VTK。该选项的值可不作更改，按默认值即可。

**VTK\_USE\_QT** —是否使用Qt。在这个系列里，我们使用Qt作为GUI开发工具，所以勾上该选项。

**VTK\_WRAP\_JAVA**、**VTK\_WRAP\_PYTHON**、**VTK\_WRAP\_TCL** —这三个选项是供准备使用Java、Python或者Tcl语言开发VTK工程的用户选择的。VTK由两个子系统组成，分别是C++类库和提供给Java、Tcl和Python来操作该类库的解释器工具，如果勾上以上三个选项，即会编译这些解释器工具。

勾选图2.3的“Advance”复选框，可以在CMake界面上看到更多的选项，各选项分别表示什么意思，可以把鼠标在该选项名字停留片刻，即会弹出关于该选项的提示信息。另外“Advance”视图下以下几个选项需要说明一下：

**BUILD\_DOCUMENTATION** —默认是关闭的，如果打开这个选项，则会编译VTK帮助文档，由于这个文档VTK已提供下载，所以可以不选。

**VTK\_USE\_GUISUPPORT** — VTK是否支持GUI，Windows下如果需要使用MFC开发的，需要勾选上这个选项。本教程在演示的时候也会把这个选项勾选，如果你确定不使用MFC，可以不用选择该选项。

定制每个选项的值以后，按“Configure”按钮，继续配置，CMake界面如图2.4所示。

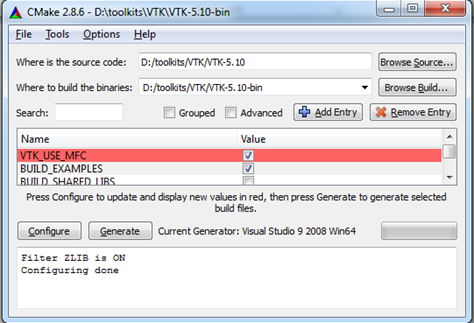


图2.4定制完CMake选项，再Configure后的界面

由于在前一步里勾选了VTK\_USE\_GUISUPPORT，所以Configure以后会有新的红色选项**VTK\_USE\_MFC**出现，如果确定要使用MFC，需要把这个选项再次勾上。

继续按“Configure”按钮，直到没有红色的选项出现，然后按“Generate”按钮，开始生成VTK工程文件。这一步完成以后，打开在“Where to build thebinaries”里输入的路径，即D:\Toolkits\VTK\VTK-5.10-bin，可以看到如图2.5所示的目录结构。

接着双击该目录下的VTK.sln文件，打开VTK工程，如图2.6所示。Visual Studio 2008默认的编译版本是Debug，对于初学者来说，最好选择该版本进行编译，以方便后续程序的调试。由于我们在CMake最开始选择编译环境时选择的是“Visual Studio 9 2008 Win64”，所以Visual Studio 2008上显示的就是“X64”版本，即64位的。选择Visual Studio 2008里的Build菜单->Build Solution(或者按F7键，关于快捷键可能有些版本的VS2008会不一样)，开始编译VTK。计算机的配置情况不同，所需的编译时间也不一样，如果计算机配置还过得去的话，图2.6的118个工程编译下来的时间大约需要半小时左右。

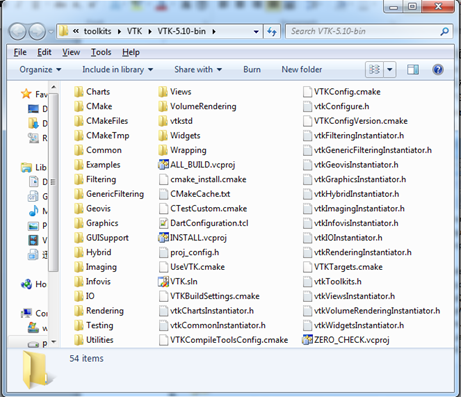


图2.5用CMake配置完VTK生成的工程文件

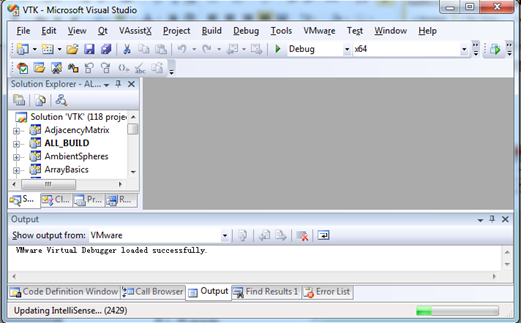


图2.6打开VTK.sln后的界面，根据CMake的选项不同，该项目所包含的工程数目也不相同。该图显示的选择编译Example，没有选择编译Testing，一共有118个工程。

编译完成后，对应的Debug目录会生成如下文件：

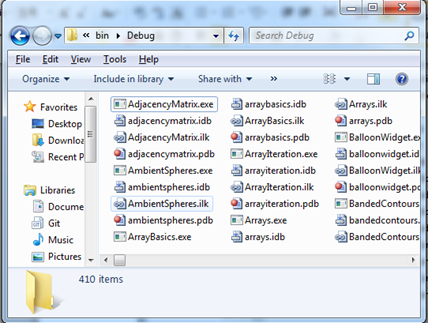


图2.7VTK Debug版本下生成的文件

如果采用动态编译，在图2.7所在的目录，会生成对应的动态链接库DLL文件。至此，VTK编译完成。如果你还想把VTK相关的头文件、lib文件等提取出来，以便用于其他项目的开发，可以编译Solution下的“INSTALL”工程，右击该工程，选择菜单：“Project Only” -> “Build Only INSTALL”，VTK里所有的头文件以及相关的库文件等，即会被提取到你在CMAKE\_INSTALL\_PREFIX里指定的路径里，默认的路径是：C:/Program Files/VTK。

**注意**：由于Windows 7有管理员权限的问题，如果直接双击VTK.sln，然后编译INSTALL工程的话，会提示不能在C:\Program Files\下创建目录等错误，此时可以先关闭掉VTK工程，在“开始”->“搜索程序及文件”一栏输入Visual Studio 2008，然后右击该程序，选择“管理员权限运行”Visual Studio 2008，接着在Visual Studio 2008下通过菜单打开VTK.sln文件，再重新编译INSTALL工程。

至此，就成功地编译并安装好了VTK。如果在编译安装VTK的过程中出现什么问题，可以在VTK users邮件列表上提问。

**2.3        创建一个简单的VTK程序**

在2.2一节里，我们详细讲解了如何编译、安装VTK，那么如何测试VTK到底有没有正确地安装，或者说怎么使用我们在前面编译出来的VTK函数库呢？

首先我们需要写一个CMakeLists.txt文件。前面的内容我们已经接触了CMake，也用CMake来配置过VTK工程，下面我们先看看CMake的介绍。

**2.3.1 什么是CMake？**

对于每个使用VTK的开发人员来说，必须认识的一个工具就是CMake，CMake的产生与发展也与VTK息息相关。以下一段内容摘自维基百科，主要是关于CMake的历史：

“CMake是为了解决美国国家医学图书馆出资的Visible Human Project项目下的 Insight Segmentation andRegistration Toolkit (ITK)软件的跨平台构建的需求而创造出来的，其设计受到了Ken Martin开发的pcmaker的影响。pcmaker 当初则是为了支援Visualization Toolkit (VTK)这个开源的三维图形和视觉系统才出现的，现在VTK也采用了CMake。”

从以上关于CMake的介绍可以知道，CMake其实就是一个跨平台的工程构建工具，可以根据不同的平台生成与平台相关的工程配置文件，比如Windows平台采用Visual Studio，则可以生成\*.dsw/\*.sln等项目文件。利用CMake可以管理大型的项目，VTK就是使用了CMake作为项目管理工具。同时CMake也简化了工程构建过程，只要给工程里的每个目录都写一个CMakeLists.txt，就可以生成出该工程的编译文件。CMake支持in-place构建(也就是生成的二进制文件跟源文件在同一个目录)和out-of-place构建(编译链接生成的二进制文件和源文件分别在不同的目录，前面我们讲VTK编译过程时就是采用这种构建方式)两种工程构建方式。

CMake有自己的语言和语法，用CMake对工程进行管理的过程，就是编写CMakeLists.txt脚本文件的过程，原则上要求工程里的每一个目录都包含一个同名的文件，而且这个文件的名字只能是：CMakeLists.txt。假如写成cmakelists.txt，由于Windows不区别文件名大小写，所以可以通过；但如果在别的平台，如Ubuntu，用CMake构建工程时就会提示找不到CMakeLists.txt，所以建议，不管在哪个平台下，都使用CMakeLists.txt这个文件名，注意大小写。

**2.3.2 写一个简单的CMakeLists.txt脚本文件**

为了测试是否成功安装了VTK，我们可以建立一个简单的VTK工程进行试验。本书里提到的所有示例都是用CMake进行管理，因此需要先写一个CMakeLists.txt文件。同样先新建一个文件夹(为便于本教程后续示例工程文件的管理，我们在前面VTK的安装目录里新建一个名为Examples的文件夹，Examples文件夹里存放本教程里提到的所有程序示例，每个程序示例的命名风格为XXX\_ProjectName，XXX表示示例所在的章节编号，ProjectName为工程的名字，比如以下测试示例命名为：2.3.2\_TestVTKInstall，在本教程完整的路径为：D:\Toolkits\VTK\Examples\2.3.2\_TestVTKInstall，接下来的内容，你可以暂时不要急着问为什么，先按照描述步骤一步一步跟着试做一遍，后面的内容会逐行代码地解释，所以暂时你先“知其然”，后续的内容会让你“知其所以然”)。

然后在该目录下新建一个名为**CMakeLists.txt**的记事本文件。输入内容为：

cmake\_minimum\_required(VERSION2.8)

project(TestVTKInstall)

find\_package(VTKREQUIRED)

include(${VTK\_USE\_FILE})

add\_executable(${PROJECT\_NAME}TestVTKInstall.cpp)

target\_link\_libraries(${PROJECT\_NAME}vtkRendering vtkCommon)

接着在2.3.2\_TestVTKInstall目录下新建一个cpp文件，名字名为**TestVTKInstall.cpp**，输入内容为：

#include"vtkRenderWindow.h"

#include"vtkSmartPointer.h"

intmain()

{

 vtkSmartPointer<vtkRenderWindow> renWin= vtkSmartPointer<vtkRenderWindow>::New();

 renWin->Render();

 std::cin.get();

 return 0;

}

也就是在D:\Toolkits\VTK\Examples\2.3.2\_TestVTKInstall目录下有两个文件，分别为CMakeLists.txt和TestVTKInstall.cpp。打开CMake程序，在CMake的“Where is the source code”一栏输入路径：D:\Toolkits\VTK\Examples\2.3.2\_TestVTKInstall，在“Where to build the binaries”一栏输入路径：D:\Toolkits\VTK\Examples\2.3.2\_TestVTKInstall\bin，接着按“Configure”按钮，CMake会弹出图2.8的对话框。

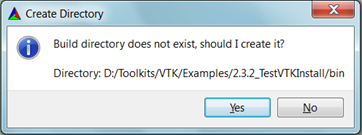


图2.8CMake询问是否创建不存在的路径

选择“Yes”，然后再选择我们准备采用的编译环境“Visual Studio 2008Win64”，CMake开始配置工程。这个工程非常小，很快就配置完成，接着在CMake界面上会出现一些红色的选项，如果没有勾选“Advance”视图，默认显示出来的选项只有两项，分别是：**CMAKE\_INSTALL\_PREFIX**以及**VTK\_DIR**，前一个选项我们已经知道是什么意思，**VTK\_DIR**这个选项就是指向编译的VTK目录，即D:/Toolkits/VTK/VTK-5.10-bin，准确地说，这个路径就是VTKConfig.cmake文件所在的完整路径。对于CMAKE\_INSTALL\_PREFIX这个选项默认的值都是C:\Program Files\XXX (XXX指的就是你在CMakeLists.txt里的project(XXX)命令里填写的工程名字)。对于VTK\_DIR，一般情况下，在你编译完VTK以后，用CMake配置VTK的工程时，会自动找到这个路径，如果CMake找不到，或者找到的不是你想要的VTK版本(假如你的计算机已经编译多个版本的VTK时)，可以通过“Browse Build…”按钮选择需要的VTK编译路径，或者直接输入该路径。

设置完选项的值以后，再次“Configure”，直到没有红色的选项出现，最后“Generate”。完成以后打开在“Whereto build the binaries”一栏指定的路径D:\Toolkits\VTK\Examples\2.3.2\_TestVTKInstall\bin，生成的文件如图2.9所示。

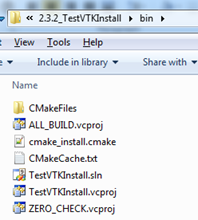


图2.9用CMake构建TestVTKInstall工程所生成的文件

打开我们熟悉的\*.sln文件，即TestVTKInstall.sln，按键盘F7键开始编译工程，完成以后按F5运行，如果没有其他意外，会弹出图2.10的对话框。

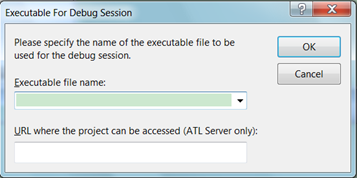


图2.10Executable For Debug Session对话框

对于不熟悉VisualStudio 2008的初学者而言，会以为这是一种错误，其实只是项目默认的启动工程没有生成可执行文件而已。右击“Solution Explorer”里的“TestVTKInstall”工程，然后选择“Set as StartUp project”，再次F5，运行界面如图2.11所示。靠前的是VTK窗口，靠后的是控制台窗口，用CMake构建的工程，默认的都是带控制台窗口的，方便输出调试信息。

如果到这里，你的程序的运行结果也跟图2.11的类似，说明你的计算机已经成功安装了VTK。

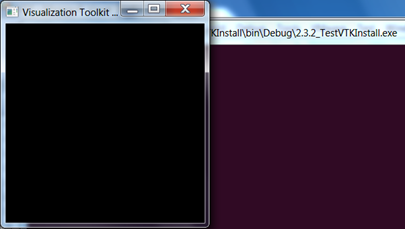


图2.11示例2.3.2\_TestVTKInstall运行界面

**2.3.3 CMake的几个常用命令**

为便于描述，我们把2.3.2节里的CMakeLists.txt的内容再列出来，并标上行号：

   1: cmake\_minimum\_required(VERSION 2.8)

   2: project(TestVTKInstall)

   3: find\_package(VTK REQUIRED)

   4: include(${VTK\_USE\_FILE})

   5: add\_executable(${PROJECT\_NAME} TestVTKInstall.cpp)

   6: target\_link\_libraries(${PROJECT\_NAME} vtkRendering vtkCommon)

第1行用到的CMake命令**cmake\_minimum\_required**，该命令完整的形式为：

**cmake\_minimum\_required**(VERSION

major[.minor[.patch[.tweak]]]

[FATAL\_ERROR])

命令说明：用于指定构建工程时所需的CMake版本要求。参数VERSION，必须的关键字，且为大写，(注：CMake的命令名是不区分大小写的，为了统一描述，本书里所有的CMake命令都以小写书写，但是CMake命令的参数关键字，如VERSION，要求必须大写。)第二个参数为指定CMake的版本号，**CMake最新的版本是2.8.10，2012年11月7日发布**。第三个参数为可选参数，且为内置的关键字“FATAL\_ERROR”。如果构建工程所用的CMake版本没有达到要求，配置过程就会弹出图2.12所示的错误提示信息，终止工程构建过程。

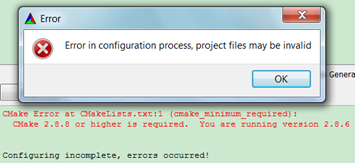


图2.12CMake构建工程时的错误提示信息

第2行，**project**命令。完整语法格式为：

**project**(projectname[CXX] [C] [Java])

用该命令指定工程名称，可指定工程支持的语言，支持语言的参数为可选。默认支持C\C++。该命令还隐含两个CMake变量：**<projectname>\_BINARY\_DIR**以及**<projectname>\_SOURCE\_DIR**，在这个例子里就是TestVTKInstall\_BINARY\_DIR以及TestVTKInstall\_SOURCE\_DIR。同时CMake也预定义了**PROJECT\_BINARY\_DIR**和**PROJECT\_SOURCE\_DIR**变量，他们的值分别跟<projectname>\_BINARY\_DIR与<projectname>\_SOURCE\_DIR一致。为了统一起见，以后直接使用PROJECT\_BINARY\_DIR和PROJECT\_SOURCE\_DIR分别表示工程的编译路径和源码路径，即这个例子里的“D:\Toolkits\VTK\Examples\2.3.2\_TestVTKInstall\bin”和“D:\Toolkits\VTK\Examples\2.3.2\_TestVTKInstall”，因为这样即使修改了工程名称，也不会影响这两个变量。

project命令的第一个参数还隐含了另外一个变量：**PROJECT\_NAME**，我们在第5、6行分别引用了这个变量：${PROJECT\_NAME}。注意：CMake使用“${变量}”的形式来获取该变量的值，也就是说，在这个例子里，${PROJECT\_NAME}就相当于工程名：“TestVTKInstall”。

第3行find\_package命令，完整语法格式为：

**find\_package**( <package>

[version]

[EXACT]

[QUIET]

[[REQUIRED|COMPONENTS][components...]]

[NO\_POLICY\_SCOPE])

find\_package命令用于搜索并加载外部工程，隐含的变量为**<package>\_FOUND**，用于标示是否搜索到所需的工程。参数[REQUIRED]表示所要搜索的外部工程对本工程来说是必须的，如果没有搜索到，CMake会终止整个工程构建过程。对VTK为例，find\_package命令搜索的就是VTK的配置文件VTKConfig.cmake。我们在VTK附带的例子目录里的CMakeLists.txt有以下的CMake脚本语句：

(摘自VTK-5.10\Examples\Tutorial\Step1\Cxx\CMakeListx.txt)：

find\_package(VTKREQUIRED)

if(NOTVTK\_USE\_RENDERING)

message(FATAL\_ERROR"Example ${PROJECT\_NAME} requires VTK\_USE\_RENDERING.")

endif(NOTVTK\_USE\_RENDERING)

其实这四行脚本的作用跟第3行实现的功能是一模一样的。关于find\_package命令其他参数的意义可以参考CMake帮助文件(CMake安装目录下的doc文件夹下有文档文件)。

第4行include命令，完整的语法为：

**include**( <file|module>

[OPTIONAL]

[RESULT\_VARIABLE<VAR>]

[NO\_POLICY\_SCOPE])

指定载入一个文件或者模块，如果指定的是模块，那么将在CMAKE\_MODULE\_PATH中搜索这个模块并载入，在本例中，指定的是VTK模块，则会在CMAKE\_MODULE\_PATH中搜索VTK模块并载入，变量CMAKE\_MODULE\_PATH指的是搜索CMake模块的目录，安装安CMake以后，在CMake的安装目录下(本机是安装在C:\Program Files (x86)\CMake 2.8\)可以找到CMake已经定义的模块，路径为：C:\Program Files (x86)\CMake2.8\share\cmake-2.8\Modules，在该目录下，有FindVTK.cmake文件。在这个文件里我们发现就有变量VTK\_USE\_FILE的说明信息。

VTK编译目录(D:/Toolkits/VTK/VTK-5.10-bin/)下的VTKConfig.cmake文件里我们可以看到变量VTK\_USE\_FILE定义为：

#The location of the UseVTK.cmake file.

SET(VTK\_USE\_FILE"D:/Toolkits/VTK/VTK-5.10-bin/UseVTK.cmake")

换言之，include (${VTK\_USE\_FILE})命令就是包含UseVTK.cmake文件。

第5行**add\_executable**命令，完整语法为：

**add\_executable**(<name>

[WIN32]

[MACOSX\_BUNDLE]

[EXCLUDE\_FROM\_ALL]

source1source2 ... sourceN)

定义这个工程会生成一个文件名为<name>的可执行文件(本例中，我们使用变量${PROJECT\_NAME}的值来指定即将生成的可执行文件的名字)，相关的源文件通过source1 source2 … sourceN列出(如果工程里有多个源文件，源文件之间用空格或者回车隔开)，如果有多个源文件，也可以先用set命令定义一个变量，然后再用取变量值的操作符${}获取到源文件列表，比如，某工程有source1.cpp、source2.cpp和source3.cpp三个文件，可以写成：

set (projectname\_srcsource1.cpp source2.cpp source3.cpp)

add\_executable(projectname ${projectname\_src})

与下行是等价的：

add\_executable(projectname source1.cpp source2.cpp source3.cpp)

第6行target\_link\_libraries命令，完整语法为：

**target\_link\_libraries**(<target>

[item1[item2 [...]]]

[[debug|optimized|general]<item>] ...)

指定生成可执行文件时需要链接哪些文件。参数<target>的名称必须与第5行指定的<name>一致。本例中，我们同样采用${PROJECT\_NAME}来获取需要的名称，并且指定需要链接的函数库为vtkRendering，在写这些链接的函数库时不需要带“.lib”的后缀。

可能这个时候，你会问：为什么就知道要链接vtkRendering.lib这个文件呢？

TestVTKInstall.cpp文件里，我们使用到了vtkRenderWindow和vtkSmartPointer这个两个类，查找这两个类的头文件所在的路径，分别为：D:\Toolkits\VTK\VTK-5.10\Rendering和D:\Toolkits\VTK\VTK-5.10\Common。于是我们可以暂时做一个猜测：因为VTK里所有的类都是以“vtk”开头，那VTK生成的函数库应该也以“vtk”开头，而我们要用到的两个类vtkRenderWindow和vtkSmartPointer的头文件vtkRenderWindow.h和vtkSmartPointer.h又分别在文件夹Rendering和Common里，刚好在VTK编译的目录里我们能找到vtkRendering.lib和vtkCommon.lib这两个文件，于是是否可以断定我们要用到的两个类vtkRenderWindow和vtkSmartPointer的接口就是分别定义在vtkRendering.lib和vtkCommon.lib里的？最后我们做出艰难的决定：在CMakeLists.txt的target\_link\_libraries里，要链接的函数库就是vtkRendering和vtkCommon。随着对VTK的深入了解，你会发现这种猜测是对的！

如果你好奇心比较强的话，在VTKConfig.cmake这个文件里(用记事本打开)，可以找到以下语句：

SET(VTK\_LIBRARIES"vtkCommon;vtkFiltering;vtkImaging;vtkGraphics;vtkGenericFiltering;vtkIO;vtkRendering;vtkVolumeRendering;vtkHybrid;vtkWidgets;vtkInfovis;vtkGeovis;vtkViews;vtkCharts")

上面的内容我们已经讲过，set命令是用来定义某个变量的，而取变量的值我们可以用符号${}，那是不是我只要在target\_link\_libraries的最后，不用一一列出所要链接的函数库，而直接引用变量值：${VTK\_LIBRARIES}来代替列出的“vtkRenderingvtkCommon”呢？答案也是可行的。

到此为止，就应该能知道CMake常用的六个命令**cmake\_minimum\_required**、**project**、**find\_package**、**include**、**add\_executable**和**target\_link\_libraries**的“所以然”了，在这个CMakeLists.txt文件的六行代码里，除了第1行关于CMake版本要求的可以省略之外，其他的5行都是必须的。

**2.3.4 一个简单的VTK工程**

回头再看看TestVTKInstall.cpp里的代码，同样标上行号：

|  |
| --- |
| 示例2.3.2\_TestVTKInstall |

  1:  #include"vtkRenderWindow.h"

  2:  #include"vtkSmartPointer.h"

  3:  int main()

  4:  {

  5:  vtkSmartPointer<vtkRenderWindow> renWin =vtkSmartPointer<vtkRenderWindow>::New();

  6:   renWin->Render();

  7:

  8:   std::cin.get();

  9:   return 0;

 10:  }

第1、2行，包含头文件，因为要用到VTK里的vtkRenderWindow和vtkSmartPointer两个类，所以包含相应的头文件。VTK对类的命名都是以小写的vtk开头，每个类的关键字的首字母大写。

第5行，用智能指针定义了一个类型为vtkRenderWindow的对象，这是VTK的类实例化对象的基本方法。因为VTK里每个类的构造函数都定义为保护成员，因此你不能够用以下的语句来定义一个VTK对象：

vtkClassExampleinstance; //vtkClassExample这个类当然是不存在的，只是说明问题而已

要不然会提示如下的错误：

errorC2248: vtkClassExample:: vtkClassExample: cannot access protected memberdeclared in class vtkClassExample

所以，要构造VTK的对象可以用第5行的方法，或者用以下的方法：

vtkRenderWindow\*renWin = vtkRenderWindow::New();

至于为什么，后面的内容会让你再“知其所以然”。

第6行，调用vtkRenderWindow里的方法显示并渲染VTK窗口。

第8行，没有其他特别的意义，只不过是让程序暂停下来，等待接受用户的输入，目的是想让你看看VTK窗口到底是长什么样子，你可以把它注释掉，看看它会不会一闪而过。

这个程序非常简单，就一个VTK窗口，其他什么也没有。但它确实是一个VTK的工程，至少使用了两个VTK类，调用了一个VTK的方法。在后面的章节里，你还会经常与这个窗口打交道。

**2.4        本章小结**

这章一开始我们为安装VTK作了非常充分的准备工作，了解了在编译安装VTK之前需要先安装哪些软件。然后，我们一步一步地演示了如何编译VTK，这个过程还是非常简单的。最后，安装完VTK，我们通过一个非常简单的VTK小程序——显示一个VTK窗口，来测试VTK是否成功安装。通过这个小程序，我们学习了CMakeLists.txt脚本的写法，并掌握了6个CMake命令，分别是：cmake\_minimum\_required、project、find\_package、include、add\_executable和target\_link\_libraries。

**2.5        本章参考资料**

[1]静态编译与动态编译：<http://cau99.blog.51cto.com/1855224/358797>

[2]CMake介绍：<http://en.wikipedia.org/wiki/CMake>

[3]CMake实践：<http://sewm.pku.edu.cn/src/paradise/reference/CMake%20Practice.pdf>

**==========欢迎转载，转载时请保留该声明信息==========**

**版权归**[**@东灵工作室**](http://weibo.com/dolingstudio)**所有，更多信息请访问**[**东灵工作室**](http://www.doling.net/)

**教程系列导航：**[**http://blog.csdn.net/www\_doling\_net/article/details/8763686**](http://blog.csdn.net/www_doling_net/article/details/8763686)

**================================================**

来源： <http://blog.csdn.net/www_doling_net/article/details/8532742>