Python 在扫描仪驱动程序发布编译中的应用①

喻武龙[1]，翁省辉[2]，李克勤[1]

[1](北京理工大学珠海学院 信息学院，珠海 519085)
[2](北京理工大学珠海学院 计算机学院，珠海 519085)

摘 要：由于扫描仪驱动程序的复杂性，驱动程序的编译及打包过程比较繁琐。以一个实际的扫描仪驱动程序为例，阐述了利用 Python 实现自动化编译的解决方案。该方案大大提高了编译的自动化程度，确保了驱动程序版本发布的速度与质量。

关键词：Python；扫描仪驱动程序；TWAIN；自动化

Application of Python to the Scanner Driver Automation Build

YU Wu-Long[1], WENG Sheng-Hui[2], LI Ke-Qin[1]

[1](School of Information, Beijing Institute of Technology(Zhuhai), Zhuhai 519085, China)
[2](School of Computing Science, Beijing Institute of Technology(Zhuhai), Zhuhai 519085, China)

Abstract: As the complexity of the scanner driver, it is cumbersome to build and package the scanner driver.The paper takes a practical scanner driver as the example to describe the solution of automation build by using Python. This solution can promote the degree of automation greatly,at the same time it can ensure the speed and quality of scanner driver release.

Key words: Python; scanner driver; TWAIN; automation

1 概述

Windows 下的扫描仪驱动程序主要采用 Twain 或者 WIA 协议。本文中的扫描仪驱动程序是一款在市场上有广泛应用的、同时支持 Twain 和 WIA 协议的驱动程序，它的总体结构图如图 1 所示。从图中可以看出，该驱动程序包括两个部分：Twain 和 WIA。其中 Twain 又分为三个模块：Protocol 模块用来实现 Twain 协议；Twain UI 模块提供用户界面。

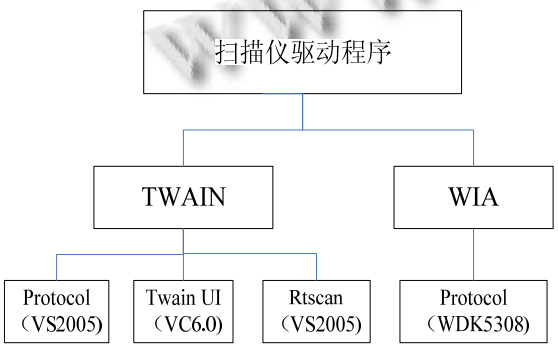

图 1 扫描仪驱动整体架构图

目前，扫描仪驱动程序的版本发布效率低下而且容易出错，主要表现在以下几个方面：第一，每个模块都需要手动编译，效率低下而且容易遗漏；第二，缺乏检查机制，无法保证编译得到的文件是否正确（如版本号等）。第三，打包时，需要将很多相关文件拷贝至一个文件夹内，而且不同的 Brand 需要拷贝不同的文件，手工拷贝极易遗漏或出错。

Python 是一种功能强大而完善的面向对象的解释性的通用型语言[1]。它具有具有语法简单、免费开源、面向对象、可扩展性强以及丰富而强大的类库等优点，非常适合作为脚本语言简化大量繁琐手工操作[2]。因此，本文提出了一个基于 python 的扫描仪驱动程序自动化编译脚本的解决方案，并对其中的关键技术进行了研究。

2 自动化编译脚本的整体设计

自动化编译脚本的整体设计如图 2 所示。它包括

① 收稿时间:2011-08-16;收到修改稿时间:2011-09-19

了一个主脚本、配置各个 brand 的脚本、编译源代码脚本、用来定义变量的脚本以及打包生成可执行文件的脚本。

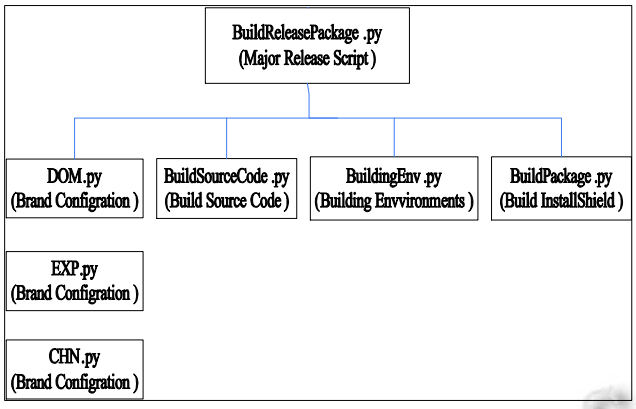

图 2　自动化编译脚本的整体设计

自动化编译脚本的设计流程如图 3 所示。其流程如下：首先编译前检查编译环境是否有效，例如是否安装了对应的编译工具，如有效则通过注册表取得对应的安装路径；之后开始编译各个模块的源代码；编译完成之后则检查生成的 dll 是否正确；最后更新各个 Brand 的配置文件再通过 InstallShield 生成各个 Brand 的可执行文件。

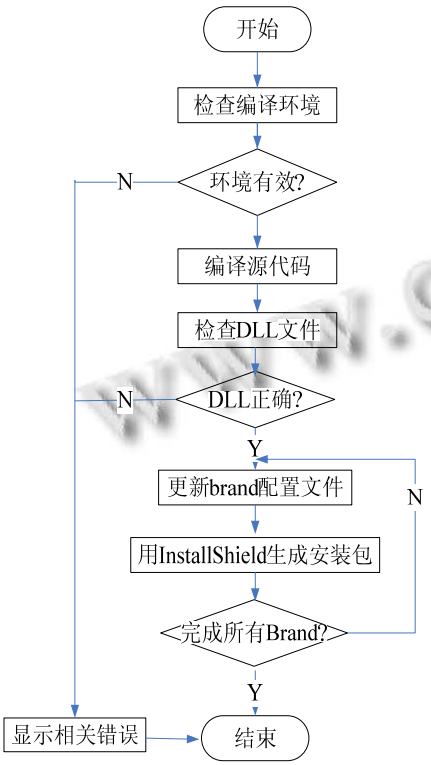

图 3　自动化编译脚本的设计流程

3　自动化编译脚本的具体实现

自动化编译脚本的具体实现主要包括获得编译工具的安装路径、vs2005 及 vc6.0 工程的自动化编译、WDK 工程的自动化编译、各个 Dll 的检查、InstallShield 工程的自动化编译以及有关文件的拷贝。Python 实现文件拷贝比较容易，在此不做阐述。

3.1　获得编译工具的安装路径

编译脚本首先要进行编译前检查，主要是通过查询注册表实现。Python 查询注册表有两种方法：一种是通过内置模块 _winreg[3]；另一种是 Win32 Extension For Python 的 win32api 模块[4]。后者是第三方提供的，需要进行额外的安装，考虑到以后脚本的移植问题，笔者的实现采用前者。关键代码如下：

```
def GetBuildCmdsFromRegistry(keyName,
valueName,outPutPath):
try:
key=_winreg.OpenKey(_winreg.HKEY_LOCAL_
MACHINE,keyName, _winreg.KEY_READ)
except:
raw_input ('Checking Environment failed! %s is not
found in your computer.'%keyName)
sys.exit()
outPutPath,type=_winreg.QueryValueEx(key,value
Name)
_winreg.CloseKey(key)
return outPutPath
```

3.2　vs2005 及 vc6.0 工程的自动化编译

vs2005 及 vc6.0 是微软提供给开发者的 IDE。它们既可以工作在图形模式下,也可以工作在命令模式下。vs2005 及 vc6.0 工程的自动化编译就是工作在命令模式下。

vs2005 提供了 Devenv 命令[5]，它不仅可以设置集成开发环境的各个选项，而且还支持从命令行生成、调试和部署项目。本文使用 Devenv 进行命令行编译。其用法如下：

devenv solutionfile.sln /build solutionconfig [/project projectnameorfile [/projectconfig name]]

其中，SolutionName 必选，解决方案文件的完整路径和名称；SolnConfigName 必选，用于生成在 Solution Name 中命名的解决方案的解决方案配置名称；/project ProjName 可选，解决方案内的一个项目文件的路径和名

称；/projectconfig ProjConfigName 可选，在重新生成命名的 /project 时要使用的项目生成配置的名称。

使用示例如下：

"C:\Program Files\Microsoft Visual Studio 8\Common7\IDE\devenv"

"D:\My Documents\Visual Studio 2005\Projects\test\test.sln" /build " Release|Win32"

如需编译 64 位 dll，则只需将最后一个参数由“Release|Win32”改为“Release|X64”。

类似地，VC6.0 工程的命令行编译可以使用 msdev 命令。其用法如下：

msdev FileName [/MAKE "ProjectName–ConfigName | ALL"] [/REBUILD /CLEAN /NORECURSE /OUT LogFile/USEENV]

使用示例如下：

"C:\Program Files\Microsoft Visual Studio\Common\MSDev98\Bin\msdev"

"C:\Program Files\Microsoft Visual Studio\MyProjects\test\test.dsw" /make "test - Win32 Release"

如果该工作区有多个工程时，可以将最后一个参数改为"all"。

值得说明的是：msdev 及 devenv 在使用过程中应该使用绝对路径，其绝对路径可通过调用 3.1 节中 GetBuildCmdsFromRegistry 获得。

3.3 WDK 工程的自动化编译

WDK 工程的编译首先要进入到相应的编译环境中，如编译 64 位时，需进入 Windows Vista and Windows server Longhorn 中的 Windows Vista and Windows server Longhorn x64 Free Build Environment；而编译 32 位动态库时需进入 Windows Vista and Windows server Longhorn x86 Free Build Environment。WDK 工程的自动化编译则是利用一条命令先进入编译环境然后再编译。关键代码如下：

```
wiaCmd="C:\\WINDOWS\\system32\\cmd.exe /c "
wiaSrcFile= "..\\ wia\\wiascanr"
WDK_BASE=GetBuildCmdsFromRegistry('SOFTWARE\\Microsoft\\WINDDK\\5308\\SETUP','BUILD', WDK_BASE)
wia32ReleaseEnv='\"'+WDK_BASE+'bin\\setenv.bat '+WDK_BASE+' fre WLH'
wia64ReleaseEnv='\"'+WDK_BASE+'bin\\setenv.bat '+WDK_BASE+' fre AMD64 WLH'
Disk,Path=os.path.splitdrive(os.getcwd())
WDK32BuildCmd=wiaCmd+wia32ReleaseEnv+'&&%s&&cd'%(Disk)+wiaSrcFile + ‘&&build -g –c’
WDK64BuildCmd=wiaCmd+wia64ReleaseEnv+'&&%s&&cd'%(Disk)+wiaSrcFile+‘&&build-g–c’
os.system(WDK32BuildCmd)
os.system(WDK64BuildCmd)
```

3.4 Dll 的检查

Dll 的检查包括两个方面：一、检查是否生成所有需要的动态库文件，可以通过 os.path.isfile 来判断文件是否存在；二、检查动态库的版本号是否为本次发行的版本号，win32api 为我们提供了获取动态库版本号的方法，关键代码如下：

```
from win32api import GetFileVersionInfo, LOWORD, HIWORD
import os,sys,redef GetDLLVersion(filename):
res = 0
try:
info = GetFileVersionInfo (filename, "\\")
ms = info['FileVersionMS']
ls = info['FileVersionLS']
Version="Version:",HIWORD(ms),".",LOWORD(ms),".",HIWORD(ls),".",LOWORD(ls)
except:
res = -1
print "Invalid DLL!"
return res
```

3.5 InstallShield 工程的自动化编译

类似 VC6.0 与 VS2005，它也是先查找注册表得到 InstallShield 安装的绝对路径，然后通过 ISCmdBld.exe 来实现。关键代码如下：

```
InstallShieldBuilder=GetBuildCmdsFromRegistry('SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\App Paths\\ISCmdBld.exe','',InstallShieldBuilder)
BuildingCmd="\""+InstallShieldBuilder+"\" -s -p "+path;
os.system(BuildingCmd)
```

其中 path 为 InstallShield 工程路径。

(下转第 206 页)

与第一次融合类似，以表5为基础数据，计算出各传感器的综合支持程度 $r=[0.000 \quad 0.529 \quad 0.471]^T$，相应地，计算出合并传感器3和2后的目标支持度和基本概率赋值，再与传感器4融合，得出 $m_3 \oplus m_2 \oplus m_4(B)=0.066$，由于传感器1的支持程度为0，传感器组2 ,3 ,4 的决策值0.066便为该次测量支持假设 H_1 的程度，即 $u(B)=0.066$。

与计算支持假设 H_1 类似，可计算出该次测量对假设 H_0 的支持程度为 $u(A)=0.913$，对假设 H_2 的支持程度为 $u(U)=1-\max(u(A),u(B))=0.087$ (见表 6)表 6

表 6　两种方法融合结果比较

假设	支持程度(FSB-DB)	支持程度(FTB)
物体为 A	0.913	0.806
物体为 B	0.066	0.115
物体为 U	0.087	0.194

通过融合结果可以看出：对 A 的概率分配函数值最大，最终判定目标是物体 A。FTB 融合算法对目标的支持程度从大到小分别为物体 A，不明物 U 和物体 B，结果与本文一致，不过 FTB 方法最终得到的融合数据为 0.806，精度明显低于 0.913，由此可见，FSB-DS 算法的准确率比 FTB 融合算法要高出 13.3%，由 FSB-DB 算法得出的目标物体的支持程度是非目标物体的十几倍，相较于 FTB 融合算法，FSB-DB 算法具有明显的优越性（印证了前文的算法分析）。由于 FTB 融合算法没有考虑多传感器的相互关系，计算传感器的支持程度时绝对化和主观化，才会导致结果误差偏大。

6　结论

多传感器对某一参数进行测量时，因受传感器自身因素和环境的影响，会有不同的测量结果，在对测量数据融合时，确定各传感器的可靠程度及数据融合方法至关重要[6]。本文提出的 FSB-DS 信息融合方法简单易行，客观反映了各传感器之间的相互支持程度，能有效避免计算过程中的绝对化，使得融合结果更加精确。

参考文献

1 Keller J. Joining sensors through data fusion. Military and Aerospace Electronics, 2008,19(11):18–23.

2 Li XL, Cao JN. Coordinated workload scheduling in hierarchical sensor networks for data fusion applications. Journal of Ccomputer Science and Technology, 2008,23(3):355–364.

3 Francis BA. Convergence in the boundary layer for singular perturbed equations. Automatica, 1982,18(2):57–62.

4 韩峰,杨万海,袁晓光.基于模糊集合的证据理论信息融合方法.控制与决策,2010,25(3):449–452.

5 王婷杰,施惠昌.一种基于模糊理论的一致性数据融合方法.传感器技术,1999,18(6):50–53.

6 刘晓光,胡学钢.D–S 证据理论在决策支持系统中的应用.计算机系统应用,2010,19(10):112–116.

(上接第 243 页)

4　结语

扫描仪驱动程序的编译与发布非常繁琐，它首先需要编译几种类型的工程文件，然后要为各个 brand 将数量庞大的文件拷贝至一起，最后再通过编译 InstallShield 工程打包成一个可执行文件。本文将 Python 引入到扫描仪驱动程序的编译与发布的流程中，提出了一种可以大大简化发布人员工作量的解决方案。用户只需从服务上下载有关文件，然后运行主脚本，输入本次发布的版本号，脚本便会一直运行，直到生成所有的发布包。实践证明，该方案能够极大地提高发布效率，大大减少了发布时间，同时还避免了人工失误。此外，该方案只需稍作修改便可应用于其他应用程序的编译与发布，特别是当发布包非常庞大时，该方案的效果更加明显。

参考文献

1 Chun WJ. Core Python Programming. 2nd Ed. Prentice Hall, 2006–09.

2 蒋崇武,刘斌,王轶辰,胡璇.基于 Python 的实时嵌入式软件测试脚本.计算机工程,2009,35(15):64–66.

3 The_winreg module. http://effbot.org/librarybook/winreg.htm.

4 Python for Windows extensions. http://python.net/crew/mhammond/win32

5 Devenv Command Line Switches.