基于 C++和 Python 混合编程的 WORD 文档操作方法

陶　诚，陆从珍
（中国电子科技集团公司第 28 研究所，南京，210007）

摘　要：介绍了基于 C++和 Python 混合语言编程的 WORD 文档操作技术及其实现，针对 C++开发的原系统对 WORD 文档操作不方便的问题，利用 Python 脚本灵活高效的特点，给出了一种在 C++动态库中嵌入 Python 调用的方法。最后展示了一个项目中实现的 WORD 文档表格化输出接口以及相应的操作 WORD 文档的 Python 脚本，解决了实际项目中的困难。

关键词：C++；Python；混合编程；WORD
中图分类号：TP273

0　引　言

在许多应用软件开发中，会出现在主要功能上需求相同，但在少数功能上需求迥异的情况，甚至会出现需求反复变化的状况。如果针对每一次需求的变化单独实现一个版本，会给日后的版本维护带来繁重的工作量。用脚本语言编写的程序虽然在性能上不如系统编程语言，但在代码修改、项目部署、运行配置方便性等方面的优势是系统编程语言所不具备的。在系统编程语言中使用嵌入式脚本语言既可以利用系统编程语言的各种优点，又可以方便程序配置甚至让专业用户修改逻辑，提供二次开发的平台。项目只要提供一批安全的核心接口给嵌入式脚本系统，就可以任由用户在合理范围进行个性的开发，实现不同的功能。

对于主流使用 C/C++来进行软件开发的系统，已有的程序大都是用 C++来开发的，积累了大量的代码。但是在对 WORD 文档的操作上，已有的封装类功能不够全面，维护修改都比较困难[1]。由于对 WORD 文档的操作都是基于组件对象模型（Component Object Model，COM）接口来实现的，也就是在后台调用 WORD 进行处理，此时脚本语言的性能不再是瓶颈，而其开发效率高、易于部署升级等特点可以为系统所利用。

混合编程作为一项利用不同编程语言特点满足系统需求的技术，在不同的研究领域中得到了广泛的应用[2~4]。Python 是一种功能强大的脚本语言，它所具有的多种特性使其非常适用于混合语言编程。本文提出了利用 Python 脚本语言的灵活性来增强 C/C++开发系统的功能，以满足用户频繁多变需求的新方法，有效地克服了 C/C++语言开发难度大、周期长带来的困难；给出了 C++动态库中嵌入 Python 解释器的具体方法，并基于 COM 接口开发了一个操作 WORD 文档输出表格化数据的应用。

1　基于 C++和 Python 的混合编程方法

1.1　Python 语言概述

Python 是一种解释型、交互式、面向对象、动态语义、语法优美的脚本语言，自从 1989 年由 Guido Van Rossum 设计出来后，经过几十年的发展，已经同 Tcl、Perl 一起，成为目前应用最广的三种跨平台脚本语言。Python 的主要特点有：

（1）免费开源

Python 是 FLOSS（自由/开放源码软件）之一。简单地说，你可以自由地发布这个软件的拷贝，阅读它的源代码，对它做改动，把它的一部分用于新的自由软件中。

收稿日期：2014-07-25

（2）高层语言

当你用 Python 语言编写程序的时候，你无需考虑如何管理你的程序使用的内存一类的底层细节。

（3）可移植性

由于它的开源本质，Python 已经被移植在许多平台上（经过改动使它能够工作在不同平台上）。这些系统包括 Linux、Windows、Solaris、Mac OS、Pocket PC、Symbian、Android 等。

（4）面向对象

Python 提供类、类的继承、类的私有和公有属性、异常处理等完善的对面向对象方法的支持。

（5）可嵌入性

可以方便地把 Python 嵌入 C/C++程序中，为程序提供脚本功能。

（6）可扩展性

如果希望一段关键代码运行得更快或者希望某些算法不公开，可以把部分程序用 C/C++编写，然后在 Python 程序中使用它们。

1.2　C++和 Python 混合编程方法

有两种最基本的方法来集成 C++和 Python——扩展和嵌入。扩展是指用类似 C/C++等的系统语言实现 Python 的扩展模块，然后从 Python 中调用这些模块的功能。嵌入是指将 Python 解释器嵌入到应用程序中，使用应用程序可以解释执行 Python 语言写成的脚本语言。扩展和嵌入都是通过 Python 的 C 语言应用程序编程接口来进行的，C 和 Python 之间的交互主要是数据格式的转换和异常的处理。

对于用 C++开发的主系统，有两种方式来调用 Python 执行脚本：一是通过系统命令调用，二是通过嵌入 Python 解释器直接调用。前者的参数传递只能通过命令行参数的形式，也就是字符串的形式，能力十分有限；后者使用 Python 内置的高级数据结构，可以传递十分复杂的数据。此外，前者在运行时要求系统提供 Python 解释器的运行环境，需要通过安装 Python 来实现，在产品级系统开发中是不现实的；后者运行时只要提供动态库和压缩包即可，方便部署，适用于产品级系统的开发。

综上所述，本文采用将 Python 解释器嵌入 C++动态库的方式来实现 C++和 Python 的混合编程。主系统通过动态库的导出函数直接调用，可以做到无缝衔接。

1.3　嵌入 Python 解释器

在 C/C++中嵌入 Python 解释器，通常需要如图 1 所示几个步骤，简单介绍如下：

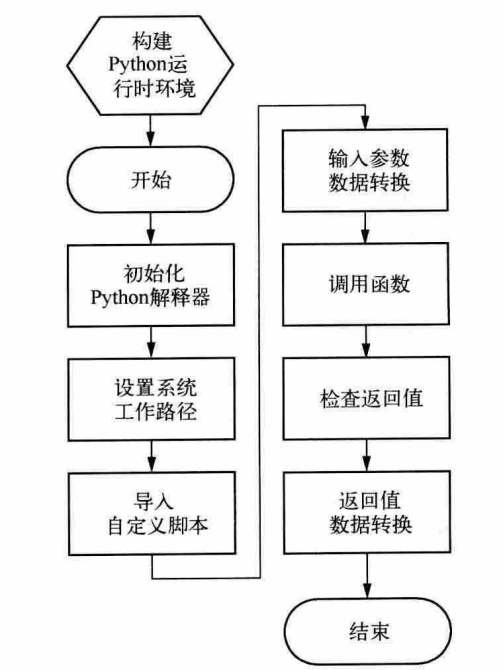

图 1　嵌入 Python 解释器步骤

（1）构建 Python 运行时环境

对于不同版本（Release 版或 Debug 版）的应用程序，需要不同版本（Release 版或 Debug 版）的 Python 动态库，一般的 Python 发行版本已经包含了 Release 版的 Python 动态库，而 Debug 版的 Python 动态库，则要从 Python 源码自己编译。

除了动态库外，还需要将要用到的 Python 库（纯 Python 语言实现部分）打包压缩为一个 zip 文件，供运行时调用。这部分工作可以使用 Py2exe 等打包工具，或者自己直接压缩相应目录来完成。

（2）初始化 Python 解释器

通过调用 Python 的 C 语言 API 函数 Py_Initialize，来完成 Python 解释器的初始化。

（3）设置系统工作路径

通过调用 Python 的 C 语言 API 函数 PyRun_SimpleString，设置系统路径，为后续模块和脚本的调用做好铺垫。

(4) 导入自定义脚本

通过调用Python的C语言API函数PyImport__ImportModule,导入自定义的Python脚本,其中包含了所有的处理代码。

(5) 输入参数数据转换

通过调用Python的C语言API函数Py__BuildValue,将C++数据类型的输入参数转换为Python可接受的输入参数。这是工作量最大的部分,也是最容易出错的地方,下一节将做进一步深入的探讨。

(6) 调用函数

通过调用Python的C语言API函数PyObject__CallObject,执行自定义脚本中的函数,将控制权交给Python解释器,待Python脚本函数执行完毕后返回。

(7) 检查返回值

检查PyObject_CallObject函数的返回值,按照事先的约定,对返回值进行检查,根据检查结果进行后续处理。

(8) 返回值数据转换

根据步骤7的结果,将返回值转换为C++数据类型。

步骤1的工作在实际编码前完成,只需执行一次。步骤2和3在应用程序第一次调用嵌入式Python脚本时执行一次,后续可以通过Python的C语言API函数Py_IsInitialized来判断是否已经初始化,避免重复。其他步骤在每一次调用嵌入式Python脚本时都要执行。

1.4 数据转换与传递

如前所述,C++和Python混合编程的主要工作就是在C++类型数据和Python类型数据之间进行转换。C++的主要数据类型包括整型、浮点型、字符串等,而Python作为高级语言,除了这些基本数据类型外,还包括列表、元组、字典等复杂的数据类型,C++中的结构体数据类型由于涉及字节对齐的问题,直接转换到Python中不好处理,一般都是转换为Python中的字典数据类型再进行处理。

将C++数据类型转换为Python数据类型,通过调用Python的C语言API函数Py_BuildValue实现。下面给出常用的参数如表1所示。

表1　Python与C++数据类型对照

参数	Python数据类型	C++数据类型
s	(*string*)	[*char**]
s#	(*string*)	[*char**, *int*]
i	(*integer*)	[*int*]
b	(*integer*)	[*char*]
h	(*integer*)	[*short int*]
l	(*integer*)	[*long int*]
B	(*integer*)	[*unsigned char*]
H	(*integer*)	[*unsigned short int*]
I	(*integer/long*)	[*unsigned int*]
k	(*integer/long*)	[*unsigned long*]
c	(*string of length* 1)	[*char*]
d	(*float*)	[*double*]
f	(*float*)	[*float*]
(*items*)	(*tuple*)	[*matching-items*]
[*items*]	(*list*)	[*matching-items*]
{*items*}	(*dictionary*)	[*matching-items*]

将Python数据类型转换为C++数据类型,对于嵌入Python的应用来说,只有在调用Python脚本函数返回,检查和转换返回值时需要使用。由于返回值的类型都是PyObject的指针,因此需要先用Py*_Check函数判断返回值的类型,然后调用相应类型的函数进行转换。下面给出常用的几对检查函数和转换函数如表2所示。

表2　常用检查函数和转换函数

检查函数	转换函数	数据类型
PyString_Check	PyString_AsString	字符串
PyInt_Check	PyInt_AsLong	整型
PyFloat_Check	PyFloat_AsDouble	浮点型
PyTuple_Check	PyTuple_GetItem	元组
PyList_Check	PyList_GetItem	列表
PyDict_Check	PyDict_GetItem	字典

另一个需要注意的问题是引用计数。因为Python中的垃圾收集(Garbage Collection,GC)是根据对象的引用计数来实现的,所以如果对象不能正确的增加/减少引用计数,会引起内存泄漏或野指针

问题，造成系统的崩溃。

在 Python 的 C 语言 API 中提供了 Py_INCREF 和 Py_DECREF 两个宏来增加和减少引用计数，Py_DECREF 在引用计数为 0 时自动调用 free 函数来释放内存。现在的问题变成了何时调用这两个宏。

这里引入两个概念：拥有（owns）和借用（borrow）。对象不可以“拥有”，但对象的引用可以“拥有”，这样对象的引用计数就变成了“拥有”对象引用的计数。对象引用的拥有者不再需要该对象时，必须调用 Py_DECREF。与此相对的，对象的引用也可以被“借用”，借用者在该对象引用的拥有者调用 Py_DECREF 之前能使用该对象引用，借用者不允许调用 Py_DECREF。“借用”的优点是不用担心何时调用 Py_DECREF，即不用担心内存泄漏的问题；缺点是在某些微妙的场景下，一旦在拥有者调用 Py_DECREF 后使用该对象引用，就会出现野指针的问题。

在 Python 的 C 语言 API 中，对象的引用在函数传入传出时，其拥有权是否会传递，要具体查看每一个函数的接口说明，不能一概而论。

2　Python 脚本操作 WORD 文档的方法

对 Word 文档的操作有两大类方法：一是通过直接读写 Word 文档来实现，二是通过 Word 提供的 COM 接口来实现。前者要求对 Word 文档的存储格式非常清楚，由于 Word 文档的存储格式不是开放的，且有各种不同的版本格式，连微软自己有时都不能保证前后兼容性，要想弄清楚的难度可想而知。当然现在存在一些第三方的库，在一定程度上弄清楚了 Word 文档的存储格式，能提供基本的操作，但稳定性和可靠性都难以保证。后者可以利用 Word 程序提供的所有功能，由微软官方提供支持，文档齐全。

Python 语言对 COM 接口调用有完善的封装[5]，与 MSDN（微软开发者网络）文档中的 Word 对象模型高度一致，不用考虑底层复杂的操作，大大提高了开发效率。

下面首先简单介绍 WORD 对象模型中各类主要对象，然后介绍在 Python 脚本中通过 COM 接口操作 WORD 文档的方法。

2.1　WORD 对象模型概述

WORD 对象模型中提供了数百个可以交互的对象，它们是按照层次顺序排列的，如图 2 所示。

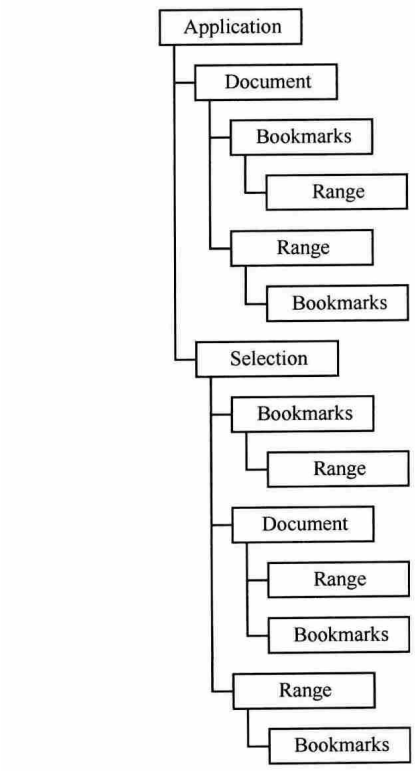

图 2　Word 对象模型

Application 对象表示整个应用程序，它的所有成员通常作为一个整体应用于 Word，可以使用该对象的属性和方法来控制 Word 环境。Document 对象表示单个 Word 文档及其所有内容。Selection 对象表示当前选择的区域，Range 对象表示文档中的一个连续的区域，与 Selection 对象不同的是它在文档中是不可见的，是操作 Word 文档时最常用的对象。此外，常用的对象还有操作表格的 Table 对象和 Cell 对象以及设置字体的 Font 对象。

2.2　用 Python 脚本操作 WORD

2.2.1　Python 脚本的基本知识

下面简单介绍 Python 脚本的基本知识，更全面的内容请参考文献[1]。

（1）作为动态类型语言，不需要预先声明变量的类型。变量的类型在赋值时被初始化，变量名区分大小写。

（2）数据类型除了整型、浮点型、字符串常见类型外，还有列表、元组、字典等功能强大的类型。

（3）字符串可以用成对的单引号、双引号、三引

号(三个连续的单引号或双引号)来表示。其中,三引号支持多行。字符串支持索引运算符[]和切片运算符[:]。

(4) 注释语句从"#"开始,直到一行的结束。

(5) 代码块通过缩进表达式表达代码逻辑,而不是使用大括号。支持制表符和空格缩进,但不能混用。

(6) 控制语句有 if/elif/else、while、for,没有 switch/case。

(7) 采用包的方式管理模块,通过 import 导入其他模块中的函数、类和变量。

2.2.2 Python 脚本调用 COM 接口

用 Python 脚本操作 Word,需要用到第三方库 win32com[6],它提供了对 COM 接口的良好封装。通常包含以下步骤:

(1) 导入 win32com 模块;

(2) 创建 Word 对象;

(3) 创建 Document 对象;

(4) 使用 Range 对象操作 Word 文档;

(5) 保存并关闭 Document 对象;

(6) 关闭 Word 对象。

3 输出表格化数据实例

3.1 数据输出接口定义

在项目中,需要将飞行计划以表格的形式输出到 Word 文档,采用独立动态库(HgScript)来封装。这里的接口包含两个部分:一个是应用程序调用 HgScript 的接口,一个是 HgScript 调用自定义 Python 脚本函数的接口。

对于前者,为了避免飞行计划结构变化(添加或删除结构体成员)对数据输出接口的影响,采用字符串(string)来存储单条飞行计划,用不可见的控制字符分隔一条飞行计划中的不同成员,用动态数组(vector)存放所有的飞行计划;用一个字符串变量存储输出的格式类型。动态库接口函数声明如下:

*BOOL Export To Word (const char * psz Plan Type, const vector<string> &vHgPlan);*

对于后者,采用列表按序存储飞行计划,用字符串存储输出的格式类型。Python 脚本接口函数声明如下:

def Export To Word (plan Type, plan Content)

在动态库中,利用 2.3 节、2.4 节所述方法来实现数据的转换和传递,这里不再赘述。

3.2 用 Python 脚本控制输出

根据输出格式类型调用不同的处理函数,以表格化的形式输出飞行计划到 Word 文档。不同的处理函数大体步骤相同,仅在一些输出参数选择上略有不同。下面给出处理函数内部的核心代码:

```
def 输出处理函数(doc,planContent):
    rng=doc.Range()
    #插入文档标题
    title=time.strftime("%m月%d日 XXXX计划报告表\n")
    rng.InsertAfter(title)
    #设置字体等参数
    rng=doc.Paragrahs(1).Range
    rng.Font.Size=20
    rng.Font.Name='宋体'
    rng.Font.Color=wdColorRed
    rng.ParagrahFormat.Alignment=wdAlignParagraphCenter
    #插入表头
    tbl=rng.Tables.Add(Range=doc.Paragrahs.Item(2).Range,NumRows=1,NumCols=10)
    #设置列宽
    tbl.Columns.Item(1).SetWidth(11)
    …
    #设置行高
    tbl.Rows.SetHeight(18)
    #解析计划内容
    plans=ParsePlanContent(planContent)
    #填写表头内容
    tbl.Cell(1,1).Range.Text='XX'
    …
    #插入计划内容
    iRow=1
    for plan in plans:
        #表格添加新行
        tbl.Rows.Add()
```

```
            iRow=iRow+1
            if'xx'in plan.keys():
                tbl.Cell(iRow,1).Range.Text=
plan['xx']
            ...
        return True
```

4　结束语

根据系统输出 Word 文档格式要求高、变化频繁的需要，采用 C++语言和 Python 语言混合编程开发了 Word 文档输出软件。实践证明，采用嵌入 Python 解释器的方式，运行时不依赖 Python 环境，适合产品级开发，同时能利用 Python 脚本语言高效灵活的特点，完成操作 Word 文档输出的功能。

Python 作为一种功能强大的脚本语言，开发效率很高，在其运行速度不是瓶颈时，完全可以作为 C/C++等编程语言的有效补充，为整个系统提供更强的灵活性。

参　考　文　献

[1] 肖雪飞，欧勇恒. 提高雷达航迹与飞行计划相关率方法[J]. 指挥信息系统与技术，2012，3(6)：30-35.

[2] 李慧，韩一平，华彩成. VisualC++与 Fortran 混合编程在电磁散射中的应用[J]. 微波学报，2012，S1：315-318.

[3] 范德胜，孟祥锋，杨修伦. 基于相移干涉术的光学信息隐藏系统的软件实现[J]. 物理学报，2012，61(24)：258-264.

[4] 黄斌，王永生. MFC 与 Simulink 的集成及其在喷水推进船加速过程控制中的应用[J]. 上海交通大学学报，2010，44(9)：1206-1210，1216.

[5] Python Software Foundation. Python v2.7.2 documentation. Jun 12，2011.

[6] Mark Hammond. Python and COM，2011.

陶诚(1980—)，男，工程师，主要研究领域为信息系统。

The C++ and Python Based Hybrid Programming Method for Word Document Operation

Tao Cheng，Lu CongZhen

(The 28th Research Institute of China Electronics Technology Group Corporation，Nanjing 210007，China)

Abstract：Compared with other modern script languages，C++ language is not handy to operate Word document. In this article，a C++ and Python based hybrid programming method for Word document operation is introduced and implemented. Making use of Python's agile and efficient feature，a method that embedding python in C++ dynamic link library is presented. The design measure and technique has been applied in real system.

Key words：C++；Python；Hybrid programming；Word