# Python嵌入C++接口总结

## 零、目录

1. 环境配置
2. 程序架构
3. Python和C++里的数据结构切换
4. 遇到的一些问题

## 一、环境配置

1、 首先在properties里面的include.path添加python/include

2、 其次在properties里的library.path添加python/libs



3、 将python里的python2.6.lib修改成python26\_d.lib，因为原来库文件里面的那个

lib不是debug版的，需要只需要修改名字就行了；或者修改头文件里面的配置

python2.6.lib在python安装目录里的python/libs里面

4、 然后在properties里面的linker——input，additional path 添加python26\_d.lib;

 



## 二、程序结构

1、首先要添加头文件：#include<Python.h>

2、在要调用python文件的地方，要启动python解释器，在用完python解释器后，要释放

python解释器，于是调用python文件要在启动Python解释器和和释放 之间；

下面是一般的程序架构：

//使用python之前，要调用Py\_Initialize();这个函数进行初始化

Py\_Initialize();

//这是直接执行python语句，如果在调用的python文件里面已经有的话就不需要了。

PyRun\_SimpleString("import sys");

PyRun\_SimpleString("sys.path.append('./')");//把当前目录设为当前路径添加python文件

加载python文件

PyObject \* pModule = NULL, \* first, \* second;

pModule =PyImport\_ImportModule("add\_item");//这里是要调用的文件名

加载无参数函数

second= PyObject\_GetAttrString(pModule, "show");//这里是要调用的函数名

PyEval\_CallObject(second, NULL);//调用函数

设置函数的参数堆栈

pArgs = PyTuple\_New(2);

PyTuple\_SetItem(pArgs, 0, pList);

PyTuple\_SetItem(pArgs, 1, pDict);

first = PyObject\_AttrString(pModule, “parameter ”);

PyEval\_CallObject(first, pArgs);

。 。 。 。 。

Py\_Finalize();//调用Py\_Finalize，这个根Py\_Initialize相对应的。

## 三、Python和C++里的数据结构切换

Python提供了6中基本数据结构：

 整型，浮点型，字符串，元组，列表，字典；

**构造python的基本数据类型：**

Py\_BuildValue("s", name)；

Py\_BuildValue("i", name)；

Py\_BuildValue("f", name)；

返回一个PyObject \*

**元组：**

添加元素：

pArgs = PyTuple\_New(2);

 PyTuple\_SetItem(pArgs, 0, pList);

 PyTuple\_SetItem(pArgs, 1, pDict);

注意：元组在赋值之后就不能改变，要改变的话只能重新生成

**列表：**

添加元素：

 PyList\_SetItem(pList, 0, Py\_BuildValue("s",x\_name));

 PyList\_SetItem(pList, 1, Py\_BuildValue("s",y\_name));

返回列表长度：

 PyList\_GET\_SIZE(list)

返回列表第I个元素：

PyList\_GET\_ITEM(list，i)

返回PYTOHN类型的C++中的字符串：

 PyString\_AsString（PyObject \* string）

添加一项：

PyList\_Append(pList, Py\_BuildValue("s", name));

 插入一项：

 PyList\_insert(pList,I, Py\_BuildValue("s", name));

**字典：**

 PyObject\* pDict = PyDict\_New();

 PyDict\_SetItemString(pDict, "first", Py\_BuildValue("i", 1));

 PyDict\_SetItemString(pDict, "second", Py\_BuildValue("i", 1));

## 四、遇到的一些问题

 1、首先python文件里面只能有函数，和一些常量值，比如字符串，整型，常量字典，不可以有可变的变量，以为C不会给python文件分配内存；如果在程序里，试图去改变python文件里面的常量，那么这个函数就不会执行下去，所以在编写python接口程序时候，需要分配内存的变量需要在c++程序里面定义，然后通过接口传递给函数：所以接口的重要行，全局变量什么的都一定不要出现；

 2、python文件里面的格式比较严格，不能有注释，不能有空行

附录：

1. C++的调用程序；
2. 被调用的python程序

# C++程序：

#include<Python.h>//前面所做的一切配置都是为了调用这个头文件和相关库

#include<iostream>

using namespace std;

int main()

{

 Py\_Initialize();//使用python之前，要调用Py\_Initialize();这个函数进行初始化

 PyRun\_SimpleString("import sys");

 PyRun\_SimpleString("sys.path.append('./')");

 PyObject \* pModule = NULL, \* first = NULL, \* second = NULL;//声明变量

 pModule =PyImport\_ImportModule("add\_item");//这里是要调用的文件名

 //无参数调用

 second= PyObject\_GetAttrString(pModule, "show");//这里是要调用的函数名

 PyEval\_CallObject(second, NULL);//调用函数

 //有参数函数调用

 //PyObject\* pDict = PyModule\_GetDict(pModule);

 first= PyObject\_GetAttrString(pModule, "add\_item");//这里是要调用的函数名

 PyObject \* pArgs = NULL, \* pList = NULL;

 //构造列表

 pList = PyList\_New(2);

 PyList\_SetItem(pList, 0, Py\_BuildValue("s",x\_name));

 PyList\_SetItem(pList, 1, Py\_BuildValue("s",y\_name));

 PyList\_Append(pList, Py\_BuildValue("s", "the third"));

 //构造字典

 PyObject\* pDict = PyDict\_New();

 PyDict\_SetItemString(pDict, "first", Py\_BuildValue("i", 1));

 PyDict\_SetItemString(pDict, "second", Py\_BuildValue("i", 1));

 //设置参数，用元组封装

 pArgs = PyTuple\_New(2);

 PyTuple\_SetItem(pArgs, 0, pList);

 PyTuple\_SetItem(pArgs, 1, pDict);

 PyEval\_CallObject(first, pArgs);//调用函数

 //PyEval\_CallObject(first,NULL);

 Py\_Finalize();//调用Py\_Finalize，这个根Py\_Initialize相对应的。

 system("pause");

 return 0;

}

PYTHON 程序：

def add\_item(mylist, mydict):

 print "have reach this"

 print mylist,'\n',mydict

def show():

 print "can work"