2、C++ 代码编写

官方文档：

Python/C API Reference Manual - Python 3.10.4 documentation

Extending and Embedding the Python Interpreter - Python 3.10.4 documentation

包含 #include "Python.h"

1. 函数介绍

- Py_SetPythonHome(L"D:\\Pythonzheng"); //导入python解释器
- Py_Initialize(); //初始化Python环境
- Py_IsInitialized() // 判断是否初始化成功
- PyRun_SimpleString("import sys"); // 向python解释器中添加代码
- PyRun_SimpleString("sys.path.append('D:/daima/5.Visual Studio 2017')"); //添加python代码目录
- PyObject * pModule = PyImport_ImportModule("t2"); //导入py文件 如引入失败 返回空
- !pModule // 引入失败 此值为True
- PyObject * pFunc = NULL; // 定义引入函数
- pFunc = PyObject_GetAttrString(pModule, "pp"); //调用的函数名为 pp
- PyObject * args = Py_BuildValue("(ii)", 28, 103); //给python函数参数赋值
- PyObject * pRet = PyObject_CallObject(pFunc, args); //调用函数 得到返回值
- int res = 0;
- PyArg_Parse(pRet, "i", &res); //转换返回类型
- cout << "res:" << res << endl; // 得到返回值

其他：

变量转化

- PyObject* Py_BuildValue(const char, …) *新建变量*
- *PyObject* PyString_FromString(char* moduleName) C字符串转换为str
- PyObject* PyLong_FromDouble(char* moduleName) long转换为double

执行

- PyRun_SimpleString(const char) *高层调用python语句*

- *PyObject* PyImport_Import(PyObject* pModule) 从模块名获取模块
- PyObject* PyObject_GetAttrString(PyObject* pModule, char* funcName) 从模块获取函数指针
- int PyCallable_Check(PyObject* pFunc) 检测函数有效性
- PyObject* PyTuple_New(int argNum) 创建元组，大小与函数参数个数一致
- int PyTuple_SetItem(PyObject* pArgs, Py_ssize_t argNum, PyObject* pValue) 设置元组值
- PyObject* PyObject_CallObject(PyObject* pFunc, PyObject* pArgs) 传入函数及参数，执行，获取

返回值解析

- int **PyArg_Parse**(PyObject* pVlaue, const char* format, …)
- int **PyArg_ParseTuple**(PyObject *arg, char *format, …)

2. 函数调用

a. 无参 有返回值

C++

```
//调用无参 但有返回值
PyObject * pp = PyEval_CallObject(pFunc, NULL); //返回值给pp Object对象
int res = 0; //接收数据 int类型
PyArg_Parse(pp, "i", &res); //类型转换 i就是整形
cout << "res:" << res << endl;//输出结果
```

b. 无参 无返回值

C++

```
PyEval_CallObject(pFunc, NULL);  //直接调用
```

c. 有参 无返回值

C++

```
PyObject* pArgs = PyTuple_New(2);
PyTuple_SetItem(pArgs, 0, Py_BuildValue("i", 2)); //0: 表示序号。第一个参数。
PyTuple_SetItem(pArgs, 1, Py_BuildValue("i", 4)); //1: 也表示序号。第二个参数。i: 表示传入的参数类型是int类型。
//PyObject * args = Py_BuildValue("(ii)", 28, 103);         //给python函数参数赋值

PyEval_CallObject(pFunc, pArgs);  //无返回类型
```

d. 有参 有返回值

C++

```
PyObject* pArgs = PyTuple_New(2);
PyTuple_SetItem(pArgs, 0, Py_BuildValue("i", 2)); //0: 表示序号。第一个参数。
PyTuple_SetItem(pArgs, 1, Py_BuildValue("i", 4)); //1: 也表示序号。第二个参数。i: 表示传入的参数类型是int类型。
//PyObject * args = Py_BuildValue("(ii)", 28, 103);         //给python函数参数赋值

PyObject * pp = PyEval_CallObject(pFunc, pArgs);  //无返回类型
int res = 0;
PyArg_Parse(pp, "i", &res);//转换返回类型
cout << "res:" << res << endl;//输出结果
```

3. 函数调用例子

C++

```
#include<python.h>
#include<iostream>
#include <stdlib.h>
using namespace std;

int main()
{

        //1、引入解释器环境 初始化
        Py_SetPythonHome(L"D:\\Pythonzheng");    //导入python解释器
        Py_Initialize();                         //初始化Python环境
        if (!Py_IsInitialized())
        {
                cout << "初始化失败! " << endl;
                return 0;
```

```
        }
        cout << "Python初始化成功" << endl;


        //2、引入python代码目录
        PyRun_SimpleString("import sys");                    // 添加项目所属 目录
        PyRun_SimpleString("sys.path.append('D:/daima/5.Visual Studio
2017')");

        //3、引入python代码文件
        PyObject * pModule = PyImport_ImportModule("t2");  //导入py文件
        if (!pModule)
        {
                cout << "调用的Python代码文件没找到" << endl;
                return 0;
        }
        cout << "成功调用的Python代码文件!" << endl;

        //4、引入python代码中的函数
        PyObject * pFunc = NULL;
// 定义引入函数
        pFunc = PyObject_GetAttrString(pModule, "pp");                  //调用的
函数名为 pp
        PyObject * args = Py_BuildValue("(ii)", 28, 103);         //给python函数
参数赋值
        PyObject * pRet = PyObject_CallObject(pFunc, args);         //调用函数 得
到返回值

        int res = 0;
        PyArg_Parse(pRet, "i", &res);
//转换返回类型
        cout << "res:" << res << endl;
//输出结果


        //5、终结python环境
        Py_Finalize();                  //反初始化，结束时调用

        system("pause");
        return 0;
}
```

4. 类调用

C++

```cpp
PyObject* pClass;
PyObject* pDict;
PyObject* pInstance;
PyObject* pClassArgs;
PyObject* pResults;

//拿到pModule里的所有类和函数定义
pDict = PyModule_GetDict(pModule);
//找到名为Executor的类
pClass = PyDict_GetItemString(pDict, "Executor");
//设置类初始化需要的参数
//初始化需要当前文件夹下的配置文件config.txt ?
pClassArgs = Py_BuildValue("(s)", "./config.txt");
//初始化Executor，建立实例pInstance
pInstance = PyInstance_New(pClass, pClassArgs, NULL);
// 执行pInstance.func(12345)
pResults = PyObject_CallMethod(pInstance, "func", "(i)", 12345);
```

5. C++格式码 / Python类型 / C++类型 对应表

FormatCode	Python	C++
s	str	char*
z	str / None	char* / Null
i	int	int
l	long	long
c	str	char
d	float	double
D	complex	Py_Complex*
O	(any)	PyObject*
S	str	PyStringObject*

6. PyArg_ParseTuple返回值解析用法：

C++

```cpp
int ok;
int i, j;
long k, l;
char *s;
int size;

ok = PyArg_ParseTuple(args, ""); /* No arguments */
        /* Python call: f() */
ok = PyArg_ParseTuple(args, "s", &s); /* A string */
        /* Possible Python call: f('whoops!') */
ok = PyArg_ParseTuple(args, "lls", &k, &l, &s); /* Two longs and a string */
        /* Possible Python call: f(1, 2, 'three') */
ok = PyArg_ParseTuple(args, "(ii)s#", &i, &j, &s, &size);
        /* A pair of ints and a string, whose size is also returned */
        /* Possible Python call: f((1, 2), 'three') */
{
    char *file;
    char *mode = "r";
    int bufsize = 0;
    ok = PyArg_ParseTuple(args, "s|si", &file, &mode, &bufsize);
    /* A string, and optionally another string and an integer */
    /* Possible Python calls:
       f('spam')
       f('spam', 'w')
       f('spam', 'wb', 100000) */
}
{
    int left, top, right, bottom, h, v;
    ok = PyArg_ParseTuple(args, "((ii)(ii))(ii)",
             &left, &top, &right, &bottom, &h, &v);
    /* A rectangle and a point */
    /* Possible Python call:
       f(((0, 0), (400, 300)), (10, 10)) */
}
{
    Py_complex c;
    ok = PyArg_ParseTuple(args, "D:myfunction", &c);
    /* a complex, also providing a function name for errors */
    /* Possible Python call: myfunction(1+2j) */
}
```