内嵌python,有手就行

今天我们通过将`yt-dlp`封装成一个exe来学习如何将`python`内嵌到你的应用程序

下载源码

我们下载`yt-dlp`的源码

```
git clone https://github.com/yt-dlp/yt-dlp.git
```

我们先查看源码的入口文件__main__.py

__main__.py

```
#!/usr/bin/env python3

# Execute with
# $ python -m yt_dlp

import sys

if __package__ is None and not getattr(sys, 'frozen', False):
    # direct call of __main__.py
    import os.path
    path = os.path.realpath(os.path.abspath(__file__))
    sys.path.insert(0, os.path.dirname(os.path.dirname(path)))

import yt_dlp

if __name__ == '__main__':
    yt_dlp.main()
```

很明显我们要调用其功能的话，只需要导入`yt-dlp`，并调用`main`函数即可

前期准备

c++调用python的原理大概是将python当做一个c++库来调用，那么我需要哪些组件呢，这里以python3.7.4为例

- 准备bin目录

进入python安装目录，我们将除了`Lib`、`include`、`libs`文件夹的所有文件都复制到你项目文件夹的`sdk/win/bin`文件夹中

将python安装目录`Lib`文件夹除了site-packages文件夹的所有文件进行zip压缩，命名为python37.zip,放到项目文件夹的`sdk/win/bin/Lib`

进入`yt-dlp`文件夹,使用python3.7.4创建并进入虚拟环境`venv`,使用`pip install -r requirements.txt`安装`yt-dlp`所需的第三方库，将`venv\Lib\site-packages`文件夹复制到项目文件夹的`sdk/win/bin`

最后将`yt_dlp`目录复制到项目文件夹的`sdk/win/bin`

- 准备头文件目录

将python安装目录`include`文件夹内的所有文件复制到项目文件夹的`sdk/include/python`，并将`pyconfig.h`的部分代码注释掉

```
#ifdef _DEBUG
       #define Py_DEBUG
#endif
```

同时将：`pragma comment(lib,"python37_d.lib")`修改为:`pragma comment(lib,"python37.lib")`

- 准备库目录

将python安装目录libs文件夹内的`python3.lib`、`python37.lib`复制到项目文件夹的`sdk/win/lib`

构建项目

我这里使用`cmake`构建项目

```
cmake_minimum_required(VERSION 3.0.0 FATAL_ERROR)

project(cYt-dlp)

if(NOT CMAKE_BUILD_TYPE)
    set(CMAKE_BUILD_TYPE "Debug")
endif()

# 设置c++17
if(MSVC)
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} /std:c++17")
else() # gcc clang
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++17")
endif(MSVC)


if(MSVC)
set(PLATFORM win)
set(LINK_LIBRARY_DIR lib)
set(LINK_LIBRARY python37 python3)
endif()

add_executable(cYt-dlp main.cpp)

#添加头文件目录
target_include_directories(cYt-dlp PUBLIC ${CMAKE_SOURCE_DIR}/sdk/include)

# 添加库目录
target_link_directories(cYt-dlp PUBLIC
${CMAKE_SOURCE_DIR}/sdk/${PLATFORM}/${LINK_LIBRARY_DIR})


# 链接动态库
```

```
target_link_libraries(cYt-dlp PUBLIC ${LINK_LIBRARY})


if(MSVC)
    # 后处理脚本，将sdk bin目录复制到生成文件夹
    add_custom_command(
    TARGET cYt-dlp POST_BUILD
    COMMAND  xcopy "\"${CMAKE_SOURCE_DIR}/sdk/win/bin\"" "$(Outdir)" /E /Y
    )

    # 设置vs调试目录
    set_target_properties(cYt-dlp PROPERTIES VS_DEBUGGER_WORKING_DIRECTORY
"$(OutDir)")
endif()
```

编写代码

- 设置Home路径

```
/**
    设置python37.dll的所在路径
**/
void Py_SetPythonHome(const wchar_t *home)
```

```
wstring wStrExeDir;
wchar_t wszExeName[MAX_PATH] = { 0, };

::GetModuleFileNameW(NULL, wszExeName, MAX_PATH);
wStrExeDir = wszExeName;
wStrExeDir = wStrExeDir.substr(0, wStrExeDir.find_last_of(L"\\"));

//设置Home路径
Py_SetPythonHome(wStrExeDir.c_str());
```

这里设置exe所在目录即可

- 初始化

```
Py_Finalize();
```

- 传递命令行参数

```
    PyObject* sys = PyImport_ImportModule("sys");
    PyObject* arglist = PyList_New(argc);
    for (int i = 0; i < argc; i++) {
        PyObject* arg = PyUnicode_DecodeFSDefault(argv[i]);
        PyList_SetItem(arglist, i, arg);
    }
    PyObject_SetAttrString(sys, "argv", arglist);
    Py_DECREF(arglist);
    Py_DECREF(sys);
```

这里是写死的，不用讲解

- 调用目标函数

```
/*
    导入模块，相当于python语法里的import 语句
*/
PyObject* PyImport_ImportModule(const char *name)
```

```
/**
    获取对象属性
**/
 PyObject* PyObject_GetAttrString(PyObject *o, const char *attr_name)
```

```
/*
    调用Python函数
*/
PyObject* PyObject_CallObject(PyObject *callable, PyObject *args)
```

```
char* result;
PyObject* pModule = PyImport_ImportModule("yt_dlp");
if (pModule == NULL) {
    PyErr_Print();
    cout << "module not found" << endl;
    return -1;
}

PyObject* pFunc = PyObject_GetAttrString(pModule, "main");
if (!pFunc || !PyCallable_Check(pFunc)) {
    PyErr_Print();
    cout << "not found function init" << endl;
    return -1;
}

PyObject* pReturn = PyObject_CallObject(pFunc, NULL);
PyErr_Print();
```

- 释放资源

```
Py_Finalize();
```

> 可以查看官方文档

最后完整代码如下

`main.cpp`

```
#include <Windows.h>
#include <iostream>
#include <python/Python.h>

using namespace std;

int main(int argc,char ** argv) {
```

```
    wstring wStrExeDir;
    wchar_t wszExeName[MAX_PATH] = { 0, };

    ::GetModuleFileNameW(NULL, wszExeName, MAX_PATH);
    wStrExeDir = wszExeName;
    wStrExeDir = wStrExeDir.substr(0, wStrExeDir.find_last_of(L"\\"));

    //设置Home路径
    Py_SetPythonHome(wStrExeDir.c_str());

    //初始化
    Py_Initialize();

    //传递命令行参数
    PyObject* sys = PyImport_ImportModule("sys");
    PyObject* arglist = PyList_New(argc);
    for (int i = 0; i < argc; i++) {
        PyObject* arg = PyUnicode_DecodeFSDefault(argv[i]);
        PyList_SetItem(arglist, i, arg);
    }
    PyObject_SetAttrString(sys, "argv", arglist);
    Py_DECREF(arglist);
    Py_DECREF(sys);

    //调用目标函数
    char* result;
    PyObject* pModule = PyImport_ImportModule("yt_dlp");
    if (pModule == NULL) {
        PyErr_Print();
        cout << "module not found" << endl;
        return -1;
    }

    PyObject* pFunc = PyObject_GetAttrString(pModule, "main");
    if (!pFunc || !PyCallable_Check(pFunc)) {
        PyErr_Print();
        cout << "not found function init" << endl;
        return -1;
    }

    PyObject* pReturn = PyObject_CallObject(pFunc, NULL);
    PyErr_Print();


    //释放资源
    Py_Finalize();

    return 0;
}
```