doi：10.11823/j.issn.1674－5795.2018.02.14

# 基于 SWIG 的 Python 仪器驱动封装技术

马宇，叶卫东
（北京航空航天大学，北京 100191）

**摘　要**：Python 是交互式的高级脚本语言，语法简单，开发效率高。但 Python 不支持内存指针，不便于仪器编程。首先对比了 Python 调用仪器驱动的主要方案，之后以某多功能数据采集模块为例，介绍了用 SWIG（Simple Wrapper and Interface Generator）开源工具将其驱动封装为 Python 拓展模块的主要步骤，并实现了远程、跨平台调用。最后设计性能对比测试，定量分析了 Python 程序中调用仪器驱动的运行效率。

**关键词**：Python；SWIG；仪器驱动；自动测试系统；虚拟仪器
**中图分类号**：TP311.1　　**文献标识码**：A　　**文章编号**：1674－5795(2018)02－0054－05

## Instrument Driver Wrapping in Python Based on SWIG

MA Yu，YE Weidong
（Beihang University，Beijing 100191，China）

**Abstract**：Python is an interactive high－level scripting language with simple syntax and high development efficiency. But Python does not support memory pointers，and is not convenient for instrument programming. This paper introduces and compares several methods to invoke instrument driver in Python，and takes one multi－function data acquisition card as an example to show how to wrap customized instrument driver into Python's extension module utilizing SWIG（Simple Wrapper and Interface Generator）. Remote and cross－platform API calling is also achieved in the process. Finally，the efficiency of invoking instrument driver in Python is quantitatively analyzed through a performance benchmark test.

**Key words**：Python；SWIG；instrument driver；automatic testing system；virtual instrument

## 0　引言

虚拟仪器是自动测试系统的基础，测试软件是虚拟仪器的核心。目前虚拟仪器和测试软件常见的开发平台包括：NI 公司的 LabVIEW，LabWindows／CVI，C#，C++和 Java 等。LabVIEW 和 LabWindows 具有丰富的图形化测试控件库；C#便于开发 Windows 图形界面程序；C，C++和 Java 是目前使用最广的编程语言。

在实际的产品测试开发中，这些平台或编程语言也存在一些局限。比如，LabVIEW 图形化语言不便于代码管理和维护，LabWindows／CVI 使用面向过程的 C 语言，模块化开发需要大量的编程技巧，开发效率较低，手工管理内存容易导致软件缺陷（如缓冲区溢出）。设计功能较复杂的测试软件时，静态语言在不重新编译的情况下难以对软件的功能进行动态配置。

Python 是面向对象的高级编程语言，动态类型、自动内存管理、解释执行、原生跨平台，可拓展性极强，具有丰富的开源库，能快速实现应用程序所需的各种功能[1]。Python 在仪器编程方面已有少量应用[2]，主要障碍是大量仪器没有提供 Python 的编程接口。SWIG（Simple Wrapper and Interface Generator）是跨语言接口转换工具，支持 Python/Perl/ PHP 等动态脚本语言与 C，C++，C#，Java 等静态编译型语言之间的接口转换[3]，Python 中的很多拓展库实际上来自 SWIG 对 C 库的封装。

本文采用 Python 设计和开发自动测试软件，提出将 SWIG 用于仪器驱动的跨语言、跨平台封装，弥补 Python 在仪器编程方面的短板，希望能够促进 Python 在虚拟仪器和自动测试领域的推广和应用。

## 1　系统组成

### 1.1　硬件结构

整个测试系统由测试计算机、测试服务器和控制主机组成，通过交换机组网，如图 1 所示。测试计算机采用 RS－232，GPIB 或 PXI 等测试总线连接测试资源（模块化仪器或可程控的台式仪器），通过仪器驱动程序控制仪器设备；测试服务器运行数据库和测试应用软件；控制主机实现人机交互，对整个测试系统进行控制。

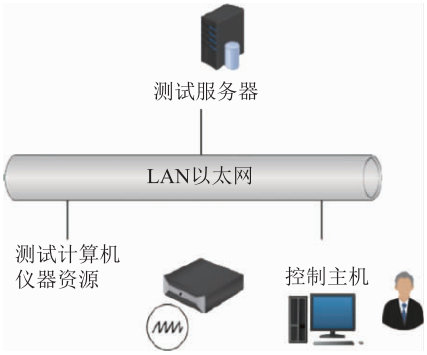


图 1　系统硬件组成

可根据测试需求，调整和缩放系统规模，如将三者合为一体，即以传统的单机方式运行整个测试系统。

### 1.2　软件组成

用 Python 开发自动测试软件，其基本层次结构如图 2 所示。

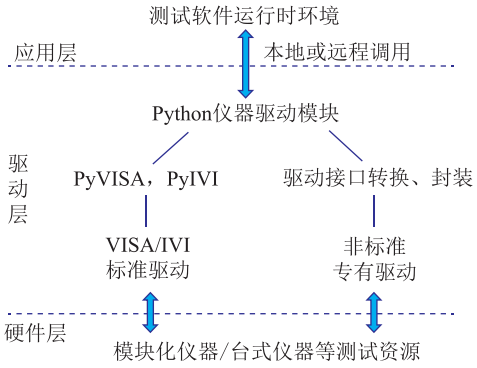


图 2　系统软件层次结构

最上层为测试应用层，负责测试用例执行、数据存储和分析、测试报告生成等具体的测试业务。应用层之下为仪器驱动层，在 Python 仪器驱动模块中封装和调用底层硬件的 API，对测试资源进行配置和管理。

对于支持 NI-VISA 和 NI-IVI 标准驱动的测试设备，Python 中的开源拓展库 pyvisa 和 pyivi 分别提供了二者的 API 封装，可以直接调用已封装好的与具体仪器无关的函数接口或可互换类驱动接口[4]。

但实际的测试需求往往丰富多变，因为各种原因，测试系统还会使用很多缺少 VISA 或 IVI 驱动支持的测试仪器，如各种总线通讯接口卡、数据采集卡、多功能复合仪器，以及一些自研或定制的非标准设备，有时也包括一些暂时无法升级的老旧测试设备。这类不被 VISA 或 IVI 驱动支持的测试资源，常被称作专有设备或非标设备。

将 Python 作为测试开发平台的主要技术障碍，就是如何在 Python 中对这类专有设备进行编程控制。

## 2　非标仪器驱动的封装和调用

仪器驱动通常采用 C 或 C++ 编写，一般会以 C 语言动态链接库的形式发布，并提供头文件、库文件等供二次开发。常见的仪器驱动程序(或 SDK)中包含的文件类型及作用，如表 1 所示。

表 1　非标仪器设备驱动程序的常见结构

| 文件类型 | 说明 |
|---|---|
| 动态链接库(*.dll) | 仪器所有功能的底层函数、常量、变量等资源，以动态链接方式加载到内存中运行，供其他程序调用 |
| 引入库/附加库(*.lib) | DLL 动态库导出的函数名称、内存地址 |
| 静态链接库(*.lib) | 功能同动态链接库，但采用静态链接 |
| Windows 驱动(*.sys) | Windows 硬件驱动文件 |
| 配置工具等(*.exe) | 仪器的参数信息 |
| 头文件(*.h) | API 函数、常量、变量等的定义和声明 |

用 C，C++ 调用专有仪器驱动的 API 函数时，一般只需要正确设置编译器的链接路径和链接方式。在 C#中，需要用 declare 语句对 API 函数的参数和返回值类型等进行声明，之后方可调用仪器 API 进行编程。有些厂商提供了 LabVIEW，C#等环境的驱动接口声明代码(或 SDK)，以简化编程工作。

Python 语言在虚拟仪器开发中使用比例比较低，大多数仪器厂商并没有为仪器提供 Python SDK 支持。Python 不支持指针操作，完全使用引用类型表示变量、参数等(传递内存地址而不是值拷贝)，编译成中间二进制字节码后通过解释器解释运行。Python 与 C 语言虽然语句类似，但在数据类型、内存操作、设计模式、运行方式等方面存在很大的差异，Python 无法简单直接地调用针对 C 语言编程而设计的仪器驱动。

本文对在 Python 中跨语言调用仪器驱动程序，进行了技术研究和方案验证。

### 2.1　仪器驱动调用和封装

Python 本身是开放、可拓展的，除了可使用大量的第三方开源拓展库，还可自行编写 Python 拓展模块。借助于一些开源库，可以实现在 Python 脚本中调用外部 DLL 动态链接库中的 C 程序。

1)ctypes 库，使用方式与 C#中非托管方式调用 DLL 类似，手工编写接口代码、声明每个 API 函数的参数和返回值类型。该方案简单易行，但工作量大、难以自动化处理，适合 API 函数较少的情况。

2)libcffi 库提供了比 ctypes 更友好的编程接口，用

更少的代码可完成同样的功能。

3)用 CPython 将外部 DLL 库封装为原生的 Python 拓展模块。该方案需要编写 C 代码，将仪器驱动头文件中定义的各种 API 函数、数据类型等转换为相应的 Python 对象。由于需要了解 Python 解释器的底层实现机制，工作量和开发难度都很大。

4)SWIG 可自动解析 C 或C++ 的代码和头文件，提取 API 函数的参数类型、返回值类型，自动生成 CPython 接口转换代码。该方案通用性和自动化程度较高，只要熟悉 SWIG 的配置语法，无需手工编写底层的转换代码，即可快速批量地进行 API 封装。Python 中的 pyivi 模块，实际上就是 NI-IVI 的 SWIG 封装。

在以上 4 种方案中，可能并不存在绝对的最佳方案。自动测试系统经常搭配使用模块化仪器、台式仪器以及其他程控测试资源，并根据测试需求灵活地增加或置换仪器。在选择驱动封装方案时，建议根据驱动 API 的数量和复杂度，结合开发人员对相关工具的熟悉程度，使用 ctypes，libcffi 或 SWIG。

本文所设计的测试系统中使用了较多的非标仪器，API 数量和类型都比较丰富，因此使用 SWIG 对仪器驱动进行封装，通过自动化处理提高开发效率。

### 2.1 用 SWIG 封装仪器驱动为 Python 拓展模块

本文以某公司的多功能数据采集模块 PXI - nuDAQ2206 为例(以下简称 DAQ2206)，简要介绍将其厂商专有驱动用 SWIG 转换为 Python 拓展模块的关键步骤。

DAQ2206 为 PXI 模块仪器，测试资源包括 64 个 AD 通道，2 个 DA 通道，2 个定时/计数器和 24 个 IO 口。在工业控制、自动测试中使用多功能复合测试设备，可以提高资源密度、减小设备体积。但此类多功能复合设备，往往缺少 VISA 和 IVI 驱动的支持。

公司提供了专有驱动包 D2K_DASK，支持包括 DAQ2206 在内的多种模块化仪器。用 SWIG 对其进行接口封装的核心工作是按照 SWIG 库的配置语法，在拓展名为 *.i 的 API 接口描述文件中对特殊的 API 参数类型进行声明，以便于 SWIG 能够正确地进行类型转换和封装。示例如下(片段)：

```
%module D2KDASK
%include"D2K_DASK.h"
%include"typemaps.i"
%apply int *OUTPUT{BOOLEAN *Stopped, U32 *AccessCnt};
%apply double *OUTPUT{F64 *voltage};
```

首先用%include 指令包含驱动 API 的头文件。一般情况下 SWIG 能自动识别大部分函数原型、变量和常量[5]，并将其转换为相应的 Python 对象。Python 采用动态数据类型和自动内存管理，无法通过指针直接操作内存，所以仪器驱动中经常使用的指针类型的参数通常需要特殊处理。

仪器 API 大多将操作的状态码作为返回值，但由于 C / C++ 函数不支持多返回值，为了输出额外的数据，一般会使用指针作为参数、间接地绕开这一限制。以函数 D2K_AI_AsyncCheck 为例，其函数原型为 I16 D2K_AI_AsyncCheck (U16 CardNumber, BOOLEAN * Stopped, U32 * AccessCnt)。其中，指针类型的参数 BOOLEAN * Stopped 和 U32 * AccessCnt，被用于输出数据采集状态和已采集数据的个数。该 API 函数实际上并不关心这些形参的初始值，只是单向地将输出数据写入指针所指向的内存。

借助于 SWIG 的指针类型处理模块 typemaps.i，通过指令%apply int *OUTPUT{BOOLEAN *Stopped, U32 *AccessCnt}，可将对应的参数声明为 Python 整数类型，参数用途为 OUTPUT。

编写好 SWIG 接口文件后，调用 swig 命令可自动生成接口的包装代码(如 D2KDASK_wrap.c)，将其编译为动态链接库(Windows 下还需要修改拓展名为 *.pyd)，即得到仪器驱动的 Python 拓展模块。在 Python 中可直接 import 导入仪器驱动拓展模块。受益于 Python 语法简单、多返回值、动态类型、自动内存管理等特性，无需繁琐的定义、声明和底层操作，可以简洁、自然地调用驱动模块中的资源，如：

```
>>> from D2K_DASK import D2K_AI_AsyncCheck
>>> status, stop, count = D2K_AI_AsyncCheck(0)
>>> status, stop, count
(0, 1, 50)
```

以上只是使用 SWIG 封装仪器驱动的简单示例。除 typemaps.i 外，SWIG 还提供了 windows.i，cpointer.i，carrays.i，cstring.i，cmalloc.i，cdata.i 等拓展库，能够处理 Windows 编程中使用的各种头文件，在 Python 脚本中操作函数指针、数组、字符串、结构体和联合体等 C 语言数据类型，通过 malloc 动态申请和释放内存，直接对内存进行不受保护地读写。SWIG 大大丰富和扩充了 Python 的底层编程能力，基本能满足用 Python 进行仪器编程的需求。

### 2.2 进一步功能封装

将仪器驱动封装为 Python 模块后，还可参考 IVI

可互换类驱动的实现机制，利用 Python 面向对象的特性，将具体的底层 API 操作封装在类内部，对外抽象出与仪器无关的高级操作接口，逐步将测试软件与底层仪器 API 解耦，提高仪器的可互换性。

### 2.3　实现分布式和跨平台调用

设计和开发测试系统时，有时需要在测试软件中集中管理和操作连接到多台测试计算机的仪器资源。可能的原因包括：计算机接口类型和数量有限，仪器设备空间分布较广，系统中不同的测试设备所要求的软件运行环境无法统一等。随着计算机软硬件平台不断升级，测试设备会逐渐过时，在需要对老旧的测试系统进行升级维护时，上述问题可能会更加突出[6]。将仪器设备接入到多台测试计算机后，传统的测试开发平台或编程语言往往难以用比较简单的方式，解决在分布式、跨平台的环境下，对仪器驱动进行远程操作和远程调试等问题。

Python 具有数量众多且功能强大的网络编程库。其中，RPyC(Remote Python Call)库采用对象代理(Object Proxying)技术，可以像操作本地对象一样操作远程主机上的 Python 模块和程序。Python 作为弱类型的动态语言，允许在运行时修改和替换对象，该技术被称为“猴子补丁”(MonkeyPatch)，可用于在不改变源码的情况下、对软件功能进行追加或变更。

结合 RPyC 库和“猴子补丁”，通过本地测试软件中的代理对象，可通过 RPyC 库，透明地操作远程计算机所连接的测试设备，如图 3 所示。

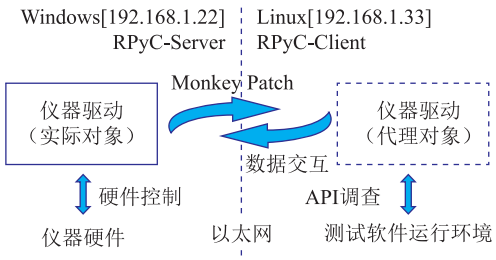


图 3　实现远程、跨平台调用仪器驱动

```
>>> import rpyc
>>> server_ip = '192. 168. 1. 22'
>>> server = rpyc. classic. connect( host = server_ip, port
= 18812)
>>> D2K_AI_AsyncCheck = \
    server. D2K_DASK. D2K_AI_AsyncCheck
>>> status, stop, count = D2K_AI_AsyncCheck(0)
```

在实现远程调用的同时，以上方案还支持跨平台操作，即远程计算机和本地计算机可分别使用不同类型的操作系统(Windows/Linux/Unix 等)，不同版本的 Python，SWIG。

使用 pyvisa，pyivi 以及 SWIG 封装的 Python 拓展调用仪器驱动，结合 RPyC 和“猴子补丁”，可将单机测试软件无缝迁移到分布式、跨平台的网络环境中。该方案能非常好地解决测试软件设计中远程调试、远程操作以及运行环境无法统一的问题，为老旧测试设备的联网升级改造、多台测试机的组网运行，提供了一种简单易行的技术方案。

## 3　性能测试与分析

Python 脚本在运行时首先会被编译为中间字节码，再通过解释器解释执行，执行过程中解释器会进行大量的类型检查、自省等操作，导致 Python 代码的运行效率和实时性表现较差[7]。虚拟仪器程序和自动测试软件，对运行性能和实时性往往有较高的要求。调用经 SWIG 封装的仪器驱动模块时，隐含的类型转换、数据拷贝等跨语言调用，必然会引入一定的封装延迟。因此，有必要定量测量和研究经 SWIG 封装后的 Python 仪器驱动模块的运行效率。

本文选取了 DAQ2206 的 5 个 API 函数，分别用 C 语言直接调用和用 Python 调用经 SWIG 封装后的驱动模块，统计执行耗时、计算封装开销。

在 C 程序中精确测量时间的原理是：北桥提供了高精度性能计数器，调用 QueryPerformanceCounter 和 QueryPerformanceFrequency 这两个 API 可分别获取其计数值和计数频率。在被测函数前后插桩、获取计数差值，除以计数频率，即得到函数的执行耗时。

表 2　测试环境

| 项目 | 本地测试环境 | 远程测试环境 |
|---|---|---|
| 操作系统 | Windows7(32 位) | Ubuntu 16. 04(64 位) |
| 硬件 | 凌华嵌入式控制器 cPCI－2510 | 桌面 PC |
| CPU | Intel Core i7－2610UE @ 1. 5 GHz | Intel Core i7－2600 @ 3. 4 GHz |
| Python | v2. 7. 13(32 位) | v3. 5. 2(64 位) |
| SWIG | v3. 0. 10 | v3. 0. 10 |
| 网络 | 100 Mbps 有线局域网 | |

嵌入式控制器 cPCI－2510 计数频率测得约为 1. 46 MHz(硬件决定)，相当于计时分辨力可达 1 μs 以上。测试次数 1000 次，数据统计如表 3 所示。

表 3 性能测试数据 μs

| API 函数 | 运行时间（C 语言） | 运行时间（Python） | 封装耗时 | |
|---|---|---|---|---|
| | | | 平均值 | 标准差 |
| Register_Card | 430421.5 | 430435.2 | 13.7 | 3.4 |
| Release_Card | 119739.0 | 119754.5 | 15.5 | 3.2 |
| AI_ReadChannel | 29.0 | 32.0 | 3.0 | 1.0 |
| AI_VoltScale | 0.3 | 2.8 | 2.5 | 0.5 |
| AI_AsyncCheck | 7.4 | 10.5 | 3.1 | 2.3 |

可以看出，SWIG 接口转换引入的封装耗时约为 2 ~ 15 μs。Register_Card，Release_Card 的延迟相对其他 API 较高，但由于只在初始化阶段和程序退出时调用，封装开销对性能的影响基本可忽略。其他 3 个 API 函数的封装开销为 2 ~ 3 μs，与 API 本身的执行时间没有明显关联。受操作系统任务调度影响，Windows 软件的实时性指标往往只能达到 10 ms 左右，因此微秒级的调用开销一般不会对测试任务产生严重影响。

在对程序性能要求非常严格的场合，不建议非常频繁（如每秒数千次以上）地调用 SWIG 封装后的 API 函数（如 AI_VoltScale）。此时，可以将最耗时的底层关键代码用 C 语言实现，一并编译、封装到 Python 仪器驱动拓展模块中，作为整体进行调用，这样既可以使用 Python 语言进行高效率的开发，也不会由于 SWIG 封装和 Python 解释运行而导致软件整体的实时性受到破坏。

在 100 Mbps 局域网环境下，远程调用驱动 API 会再引入约 2 ms 的传输延迟，可能对软件的执行效率产生一定的影响。因此，远程调用一般更适用于软件调试、低速数据采集等实时性要求相对较低的场景。

## 4 总结

测试软件是虚拟仪器和自动测试系统的核心，传统的测试开发平台使用中存在较大的局限性。本文通过 SWIG 将仪器驱动程序转换为 Python 拓展模块，弥补了 Python 在底层编程方面的不足，解决了用 Python 进行仪器编程的主要障碍。

受益于 Python 的自动内存管理、动态类型、面向对象以及丰富的拓展库，用 Python 开发测试软件，可提高编程效率，降低在分布式、跨平台的环境下设计和开发测试软件的难度，缩短复杂测控系统的开发时间，一定程度上也有助于提高仪器的可互换性。在对软件实时性、仪器操作性能等有较高要求的场合，可采用 C 语言和 Python 混合编程，在软件开发效率和运行效率之间，取得比较好的平衡。

## 参 考 文 献


[1] 丁未. 将工业与科技世界的运行统一在 Python 语言的开源框架中[J]. 中国仪器仪表，2013(08)：23 - 25.

[2] Hughes J M. 真实世界的 Python 仪器监控：Real world instrumentation with Python：数据采集与控制系统自动化[M]. 北京：电子工业出版社，2013.

[3] Beazley D M. SWIG：An Easy to Use Tool for Integrating Scripting Languages with C and C ++ [C]// Usenix Tcl/tk Workshop，1996.

[4] 黄建军，李宥谋，刘婧，等. 基于 Python 语言的自动化测试系统的设计与实现[J]. 现代电子技术，2017，40(4)：39 - 43.

[5] Beazley D M. Automated scientific software scripting with SWIG[J]. Future Generation Computer Systems，2003，19(5)：599 - 609.

[6] Weltzin C，Schlonsky S. Reducing obsolescence of Linux - based ATEs with virtualization[J]. Instrumentation & Measurement Magazine IEEE，2011，14(4)：1 - 3.

[7] 范浩杰. 面向 Python 程序源代码的分析与编译优化研究[D]. 北京：北京信息科技大学，2015.


**收稿日期**：2017 - 09 - 18

**马宇**（1992 - ），男，陕西榆林人，硕士，主要研究方向为嵌入式系统、数据采集和自动测试系统软件开发。

**叶卫东**（1957 - ），男，广东惠阳人，副教授，博士，主要研究方向为嵌入式计算机系统、分布式计算机系统、数据采集系统、自动测试系统、大型建筑的综合健康管理系统、无线传感器网络、网络时间同步技术、LED 大屏幕显示技术、高精度信号调理技术等。

1990 年于北京航空航天大学获测试计量技术及仪器专业硕士学位，2010 年获得北京航空航天大学检测技术与自动化装置专业的博士学位。承担的本科课程有《传感器信号调理电路设计与仿真》《检测技术与自动化装置专业课程设计》《检测技术与自动化装置专业综合实验》。主讲的研究生课程有《嵌入式计算机系统设计》《MOTOROLA 单片机系列实验》《数据采集系统设计》。科研方面共主持参加各类研发项目 50 余项，其中获得部级二等奖 2 项，三等奖 1 项，发表论文 30 余篇，其中 EI 检索 8 篇，合作出版论著 3 篇。参与《数据采集系统检定规范》的制定和 LXI 总线标准的翻译工作。