**AUBOPE插件开发**

目录

[一 开发框架 1](#_Toc7050)

[1.1. 插件管理器目录结构 1](#_Toc19302)

[1.2. 插件依赖关系文件 2](#_Toc13242)

[1.3. 插件MetaData配置文件 2](#_Toc31098)

[1.4. 插件qmake文件 3](#_Toc14519)

[1.5. 插件头文件 4](#_Toc25462)

[1.6. 插件共享文件夹 9](#_Toc7172)

[二 脚本结构 10](#_Toc12647)

[2.1. 脚本通用接口 10](#_Toc14279)

[2.2. 输出参数协议 10](#_Toc1533)

[2.3. 输出参数举例 11](#_Toc6281)

[2.4. 脚本通用接口例程 13](#_Toc17871)

[三 应用案例 13](#_Toc14830)

# 开发框架

示教器通过插件管理器统一管理所有为示教器服务的插件。

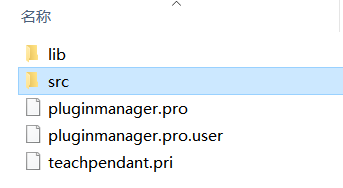
示教器的运行目录规定了lib/teachpendannt/plugins为示教器的插件库目录，share/teachpendant/plugins为示教器的共享文件根目录。

插件管理器会深度遍历检索该目录中所有以.so为后缀的示教器插件。检索过程中会根据插件的MetaData（元数据，下文中有详细描述）获取插件的名称，版本号等信息，之后实例化插件对象，创建并加载插件窗口。

下文中以Template插件为例进行介绍，这里的”Template”，即是插件库名称，又是插件运行目录名称，还是插件名称。

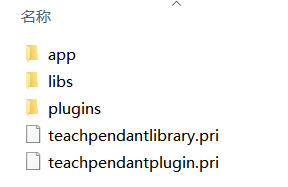
## 插件管理器目录结构

pluginmanager根目录结构如下：



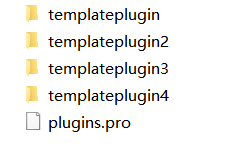
该目录中我们仅需要了解src目录的结构。

Src目录结构为：

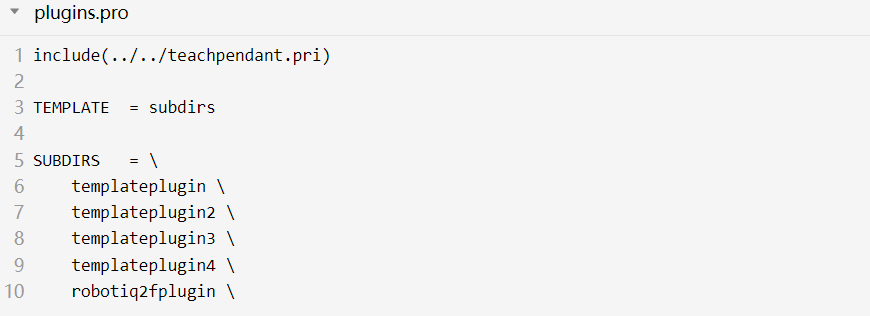


该目录中我们仅需要了解plugins目录的结构。

plugins目录结构为：



这个目录是插件开发的主目录，本例程中我们添加了4个插件，插件目录名如上图所示，plugins.pro是一个插件的总qmake文件，该文件内容如下：

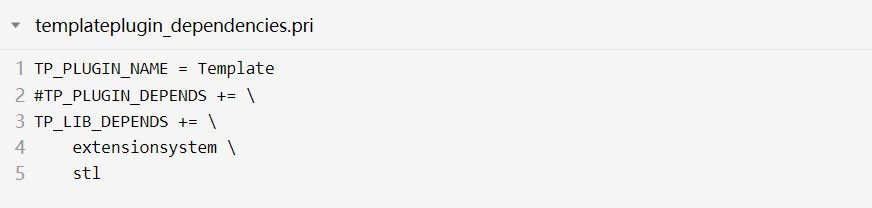


这里仅需要修改SUBDIRS中的插件目录即可。每一项插件目录代表一个示教器插件。

## 插件依赖关系文件

该文件名称命名规范为”%PluginDirName%\_dependencies.pri”.

在Template插件中该插件目录名为”templateplugin”，所以该插件对应的依赖关系文件名称为”templateplugin\_dependencies.pri”。该文件内容为：

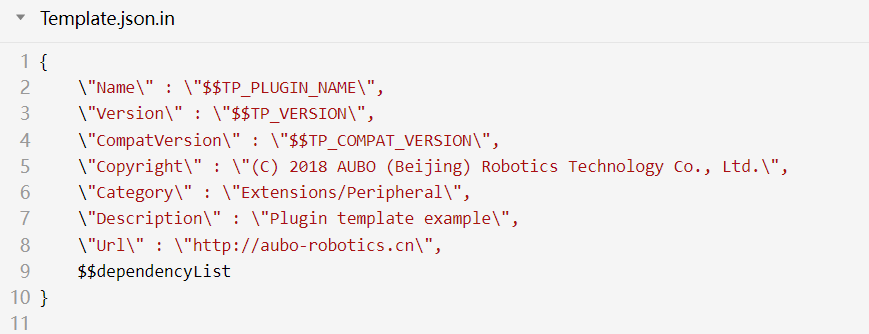


这里仅需要修改TP\_PLUGIN\_NAME变量的值，其余部分保持原状。

**TP\_PLUGIN\_NAME变量值尤其重要**，它即是插件库名称”lib%TP\_PLUFIN\_NAME%.so)”（对应于Template插件库名为”libTemplate.so”），又是插件共享文件目录”share/teachpendant/plugins/%TP\_PLUGIN\_NAME%”（对应于Template插件的共享文件目录为”share/teachpendant/plugins/Template”），还是插件名称（这里的插件名称是指下文中调用脚本通用接口时所使用的插件名称）。

## 插件MetaData配置文件

该文件名称命名规范为”%TP\_PLUGIN\_NAME%json.in”.在Template插件中该插件元数据配置文件名称为”Template.json.in”。该文件的内容为：



**注意：该文件的后缀名为”.json.in”，不是标注的.json**

本文件仅需要修改Category选项即可，**Category变量值尤其重要**，Categotry值以”/”作为分隔符，可以理解为两级目录名，”Template.json.in”中的Category值为”Extension/Peripheral”，第一级目录名”Extensions”对应与示教器的TabWidget中的Extension选项，**第一级目录名不得修改**，**必须为”Extensions**”。第二级目录名为”Peripheral对应与Extensions”选项中的左侧抽屉Peripheal，该抽屉的第一个格子Template对应于本文件中的Name的值$$TP\_PLUGIN\_NAME，这里该值为上文templateplugin\_dependencies.pri文件配置的”Template”

Description/Url值为该插件的描述信息，示教器不解析，仅作为备注使用。其余值不需要做修改。

## 插件qmake文件

该文件名称命名规范为”%PluginDirName%.pro”.在Template插件中该文件名称为”templateplugin.pro”。文件内容如下：

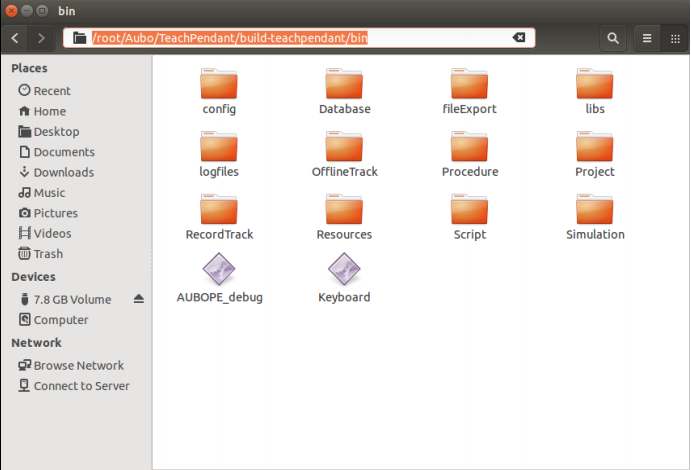


**其中尤其重要的是第一行include(../../teachpendantplugin.pri)，必须加，而且必须加在该文件的第一行。**

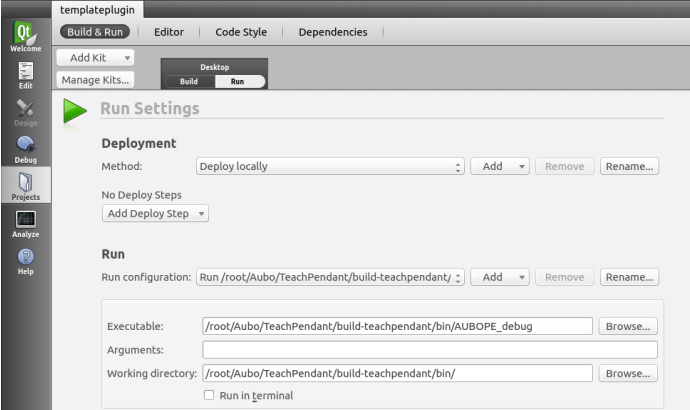
其中**最后一行为该插件的编译目录**，如果独立测试则注释该行，如果与示教器进行联调则打开改行，将DESTDIR的值设置为实际的示教器运行目录中该插件所在的运行目录”lib/teachpendant/plugins/%TP\_PLUGIN\_NAME%”.

例如下图使用的示教器运行目录为：

"/root/Aubo/TeachPendant/build-teachpendant/bin"



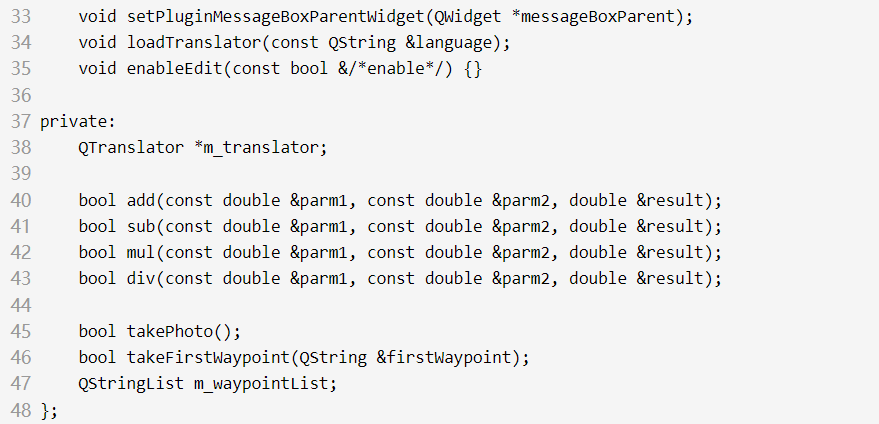
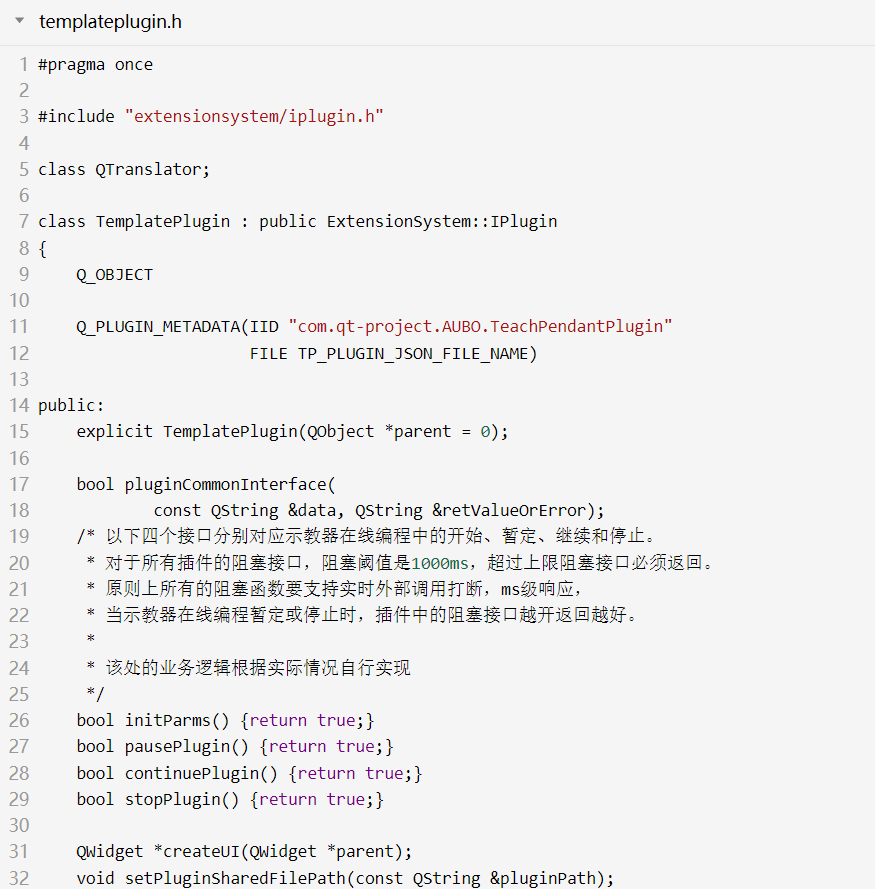
在与示教器联调时，将插件配置为下图所示：



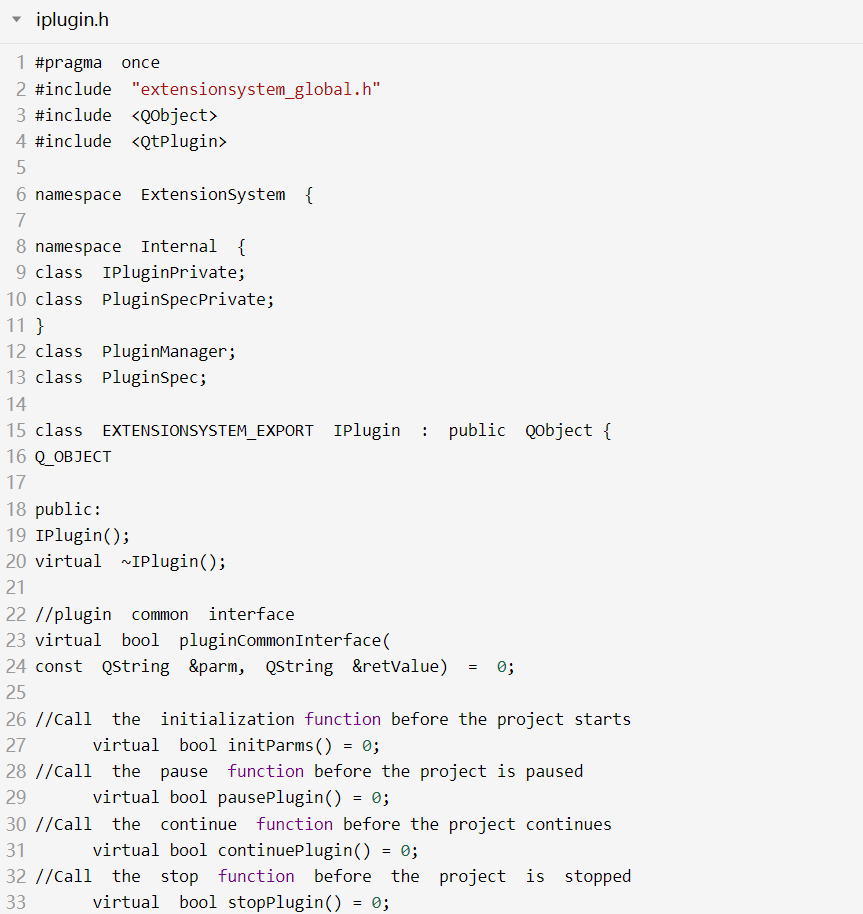
其余的中间部分根据实际开发需求进行修改。

## 插件头文件

该文件名称命名规范为”%PluginDirName%.h”。在Template插件中该插件目录名为”templateplugin”，所以该插件对应的头文件名称为”templateplugin.h”，该文件内容如下：



**注：插件头文件必须继承子插件管理器的抽象接口类ExtensionSystem::IPlugin。**其中 ExtensionSystem是一个插件管理器的命名空间，IPlugin是插件抽象接口类，该文件内容如下：



插件头文件需要实现 IPlugin 类中的全部纯虚函数。下面对纯虚函数逐一介绍：

|  |  |
| --- | --- |
| **Bool pluginCommonInterface(const QString &data, QString &retValueO rError)** | |
| 功能 | 插件通用接口，由脚本通用接口 script\_common\_interface 调用 |
| 参数 | data 为从脚本通用接口 script\_common\_interface 中传入的输入参数 |
| retValueOrError，如果函数调用成功， 表示函数的输出参数； 如果函  数调用失败，表示函数的错误信息。 |
| 返回 | 调用函数的结果 |

|  |  |
| --- | --- |
| **bool initParms(void)** | |
| 功能 | 在示教器工程 Start 之前调用该函数。初始化示教器已加载插件的参  数。 |
| 参数 | 无 |
| 返回 | 若调用成功返回 true，若调用失败返回 false。 |

|  |  |
| --- | --- |
| **bool pausePlugin(void)** | |
| 功能 | 在示教器工程 Pause 之前调用该函数。插件需要确保内部逻辑可以  正常暂停。 |
| 参数 | 无 |
| 返回 | 若调用成功返回 true，若调用失败返回 false。 |

|  |  |
| --- | --- |
| **bool continuePlugin(void)** | |
| 功能 | 在示教器工程 Continue 之前调用该函数。插件需要确保内部逻辑可  以正常继续。 |
| 参数 | 无 |
| 返回 | 若调用成功返回 true，若调用失败返回 false。 |

|  |  |
| --- | --- |
| **bool stopPlugin(void)** | |
| 功能 | 在示教器工程 Stop 之前调用该函数。插件需要确保内部逻辑可以在  5s 内正常停止。 |
| 参数 | 无 |
| 返回 | 若调用成功返回 true，若调用失败返回 false。 |

|  |  |
| --- | --- |
| **QWidget \*createUI(QWidget \*parent)** | |
| 功能 | 创建并返回插件窗口。 |
| 参数 | **parent** 插件窗口的父类 |
| 返回 | 插件窗口 |

|  |  |
| --- | --- |
| **void** setPluginSharedFilePath**(const QString &pluginPath)** | |
| 功能 | 设置插件相对于示教器程序的相对路径。 |
| 参数 | **pluginPath** 插件共享文件相对于示教器程序的相对路径。 |
| 返回 | 无 |

|  |  |
| --- | --- |
| **void setPluginMessageBoxParentWidget(QWidget \*messageBoxParent)** | |
| 功能 | 设置插件 MessageBox 窗口的父类 |
| 参数 | **messageBoxParent** 插件 MessageBox 窗口的父类 |
| 返回 | 无 |

|  |  |
| --- | --- |
| **void loadTranslator(const QString &language)** | |
| 功能 | 加载插件翻译文件 |
| 参数 | **language** 标准语言代码， 例如： en\_GB zh\_CN zh\_TW fr\_FR de\_D  E |
| 返回 | 无 |

|  |  |
| --- | --- |
| **void enableEdit(const bool &enable)** | |
| 功能 | 使能插件 UI 编辑， 当示教器工程处于运行态时， 会失能插件 UI，当  退出运行态时，会使能插件 UI。 |
| 参数 | Enable 使能插件 UI 编辑 |
| 返回 | 无 |

## 插件共享文件夹

示教器的运行目录规定了 share/teachpendant/plugins 为示教器的共享文件根 目录。对应于每一个插件， 共享文件目录为"share/teachpendant/plugins/%TP\_PLU GIN\_NAME%" 。例如 Template 插件的共享文件目录为"share/teachpendant/plugins/Template"。

这里指的共享文件目录示教器会通过 setPluginSharedFilePath 接口函数以相对于示教器可执行程序路径的相对路径形式给出。例如示教器可执行程序路径为： "/root/Aubo/TeachPendant/build-teachpendant/bin"，Template 插件的共享文件路径为： "/root/Aubo/TeachPendant/build-teachpendant/share/teachpendant/plugins/Tempate"，

所以 setPluginSharedFilePath 接口函数给出的路径为"../share/teachpendant/plugins/Template"。

在 Template 插件中， 我们没有共享文件。例程中在"share/teachpendant/plugins/Template" 目录下创建了一个 doc 文件夹，仅作示例。

# 脚本结构

示教器的运行时是由 Lua 脚本控制的，示教器的lua 接口中封装了与机械臂本体相关的接口函数，并提供了调用示教器插件的脚本通用接口。

示教器工程中，每一个主程序和线程均独立开辟了一个lua 运行时，每个运行时通过《示教器 V4.x 脚本接口手册》中的接口控制本体和插件。插件开发人员需要确保插件对示教器暴露的接口均是线程安全的，并且插件的 UI 需要做到毫秒级响应。

## 脚本通用接口

|  |  |
| --- | --- |
| **variable script\_common\_interface(string pluginName, string data)** | |
| 功能 | 脚本通用接口，该函数中根据插件名称调用对应插件的脚本通用接  口 pluginCommonInterface。 |
| 参数 | pluginName插件名称，由插件依赖关系文件中的 TP\_PLUGIN\_NAME 定义。 |
| data输入参数，函数不对 data做任何解析，仅根据 pluginType参数  调用相对应插件的插件通用接口，并将 data作为输入参数。 |
| 返回 | 任意类型。  通过调用插件通用接口 pluginCommonInterface 返回的输出参数， 即  上文中的 retValueOrError 参数。  脚本通用接口调用插件通用接口。如果成功，按照输出参数协议（见 下文）解析 retValueOrError参数，并返回到 Lua脚本的运行时工作栈；如果失败，将 retValueOrError 参数作为错误信息处理。 |
| 示例 | addResult=script\_common\_interface("Template","add|8.8,1.2") print("addResult:"..addResult) |

## 输出参数协议

输出参数支持 Table(包含 key:value)多层嵌套(最多 5 层),多参数返回。

Table 中的 key 只能为 string 、int 、double 类型。

关于 table 中 Key-Value 说明：

table 内全部 value 均有默认的 index 作为 key,

如果某个键值对显示的给定了 key,那么使用显示给定的 key.

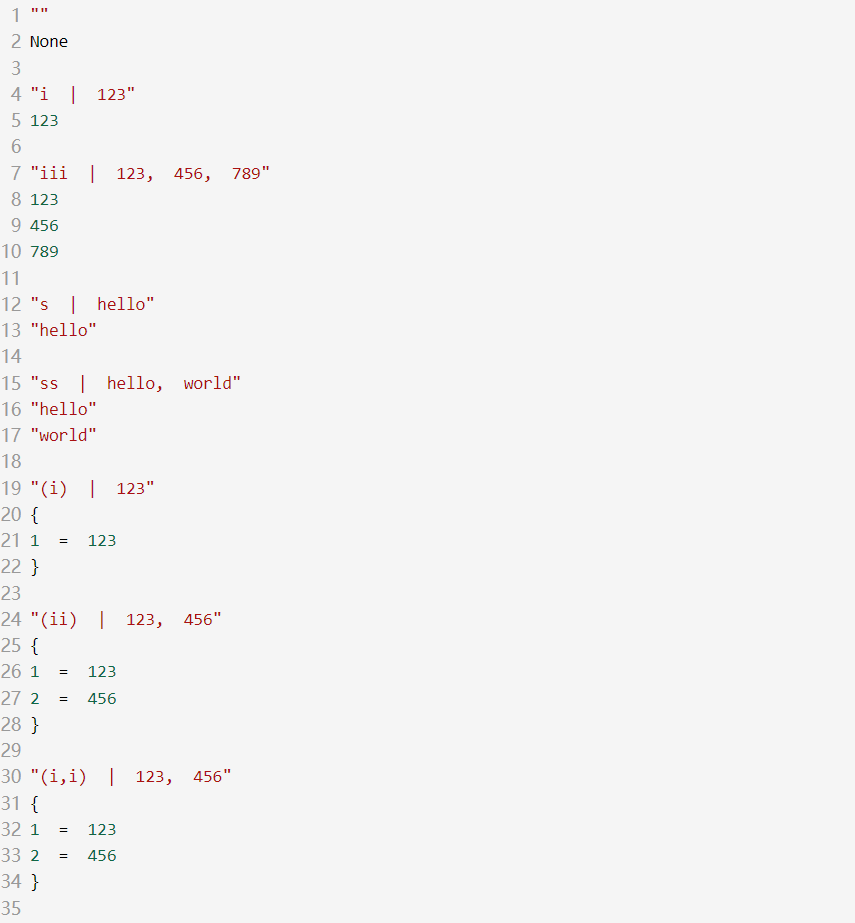
table 中每个键值对之间仅支持逗号或者空格分隔符,不加分隔符也可以.

|  |  |
| --- | --- |
| Lua 基础数据类型 | format 字符 |
| table | () |
| int | i |
| double | d |
| string | s |
| bool | b |

返回值格式为 ： "format|data1,data2,data3..."

## 输出参数举例

常规用法：



多类型多返回值：

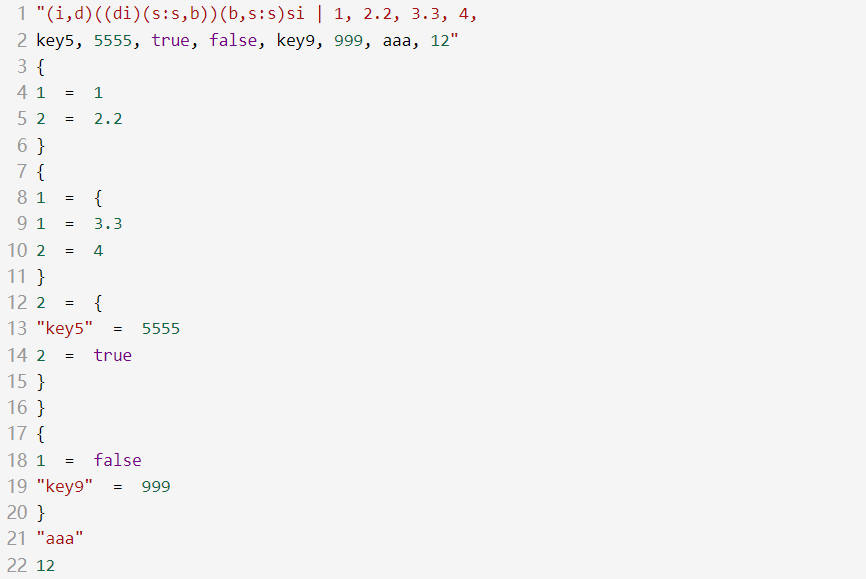
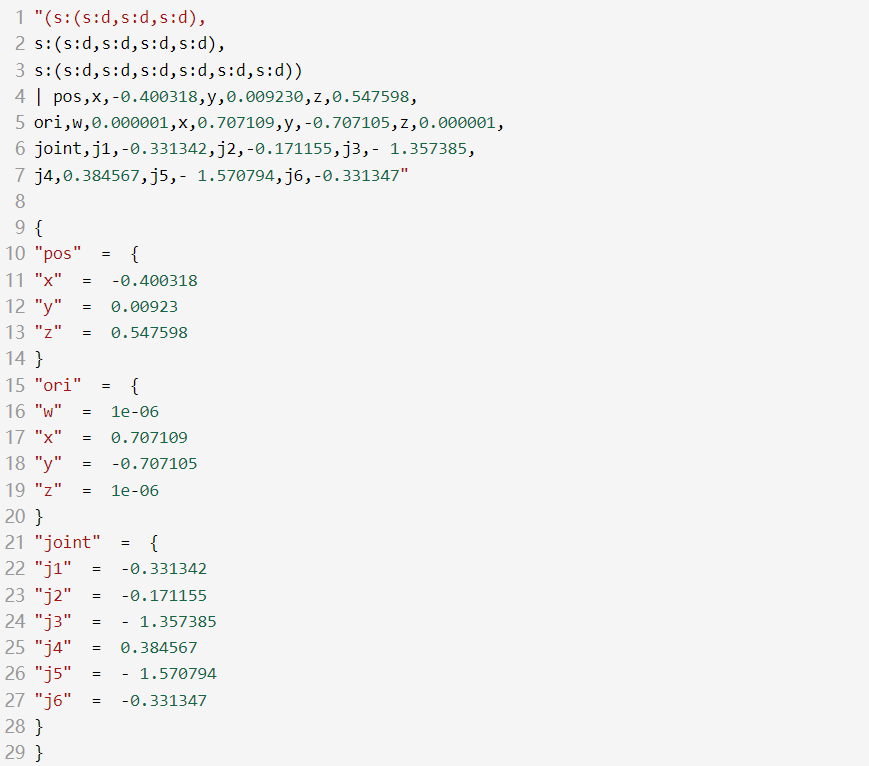


Table 嵌套(全部显示指定 key)



## 脚本通用接口例程



# 应用案例

本例中添加了4个插件，目录名分别为：templateplugin，templateplugin2，templateplugin3，templateplugin4。配置信息如下表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **插件名称** | **依赖关系文件名** | **TP\_PLUGIN\_NAME值** | **MetaData配置文件名** | **category值** |
| Template | templateplugin\_dependencies.pri | Template | Template.json.in | Extensions/Peripheral |
| Template2 | Templateplugin2\_dependencies.pri | Template2 | Template2.json.in | Extensions/Technology |
| Template3 | Templateplugin3\_dependencies.pri | Template3 | Template3.json.in | Extensions/Peripheral |
| Template4 | Templateplugin4\_dependencies.pri | Template4 | Template4.json.in | Extensions/Technology |

示教器会将这四个插件加载到TabWidget中的Extensions 选项中，将Template和 Template3加载到抽屉Peripheral中，将Template2和Template4加载到抽屉Technology 中。

运行效果如下图:

