ARCS脚本编程

目录

1. ARCS 脚本简介

ARCS的脚本编程是指用脚本语言来控制机器人。基于LUA，遵循LUA语言的语法规则。支持示教器脚本编程，支持SDK发送本地脚本程序到控制器。

2. LUA语法

LUA语法教程可参考 https://www.runoob.com/lua/lua-tutorial.html

2.1 常用变量类型

类型	说明	示例
nil	表示一个无效值，在条件表达式中相当于 false。	nil
Boolean	true或false。	true false
number	数字。包括整型、浮点型。	1 -5 3.1415
string	字符串。 一对双引号或者单引号之间的字符。	“abc” ‘abc’
function	自定义函数。	print
table	表。用{}表示。	{1, 2, 3}

2.2 自定义函数

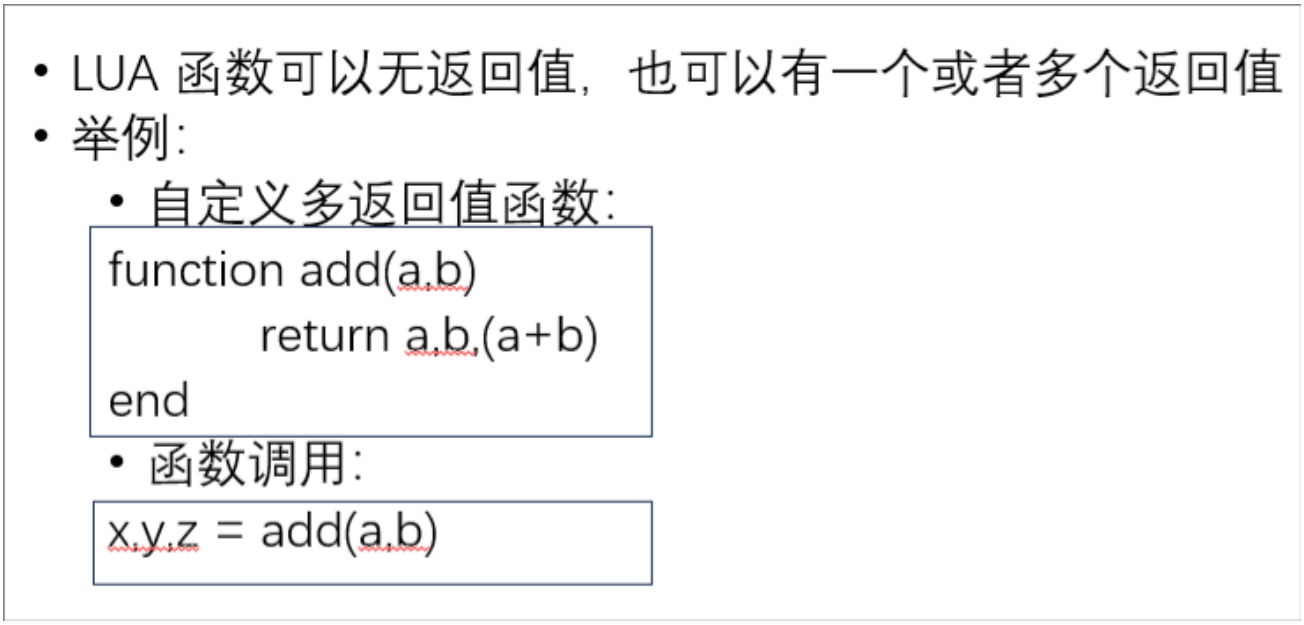
- LUA 函数可以无返回值，也可以有一个或者多个返回值
- 举例：
  - 自定义多返回值函数：

```
function add(a,b)
        return a,b,(a+b)
end
```

  - 函数调用：

```
x,y,z = add(a,b)
```

2.3 条件和循环语句



3. 查看ARCS脚本API定义

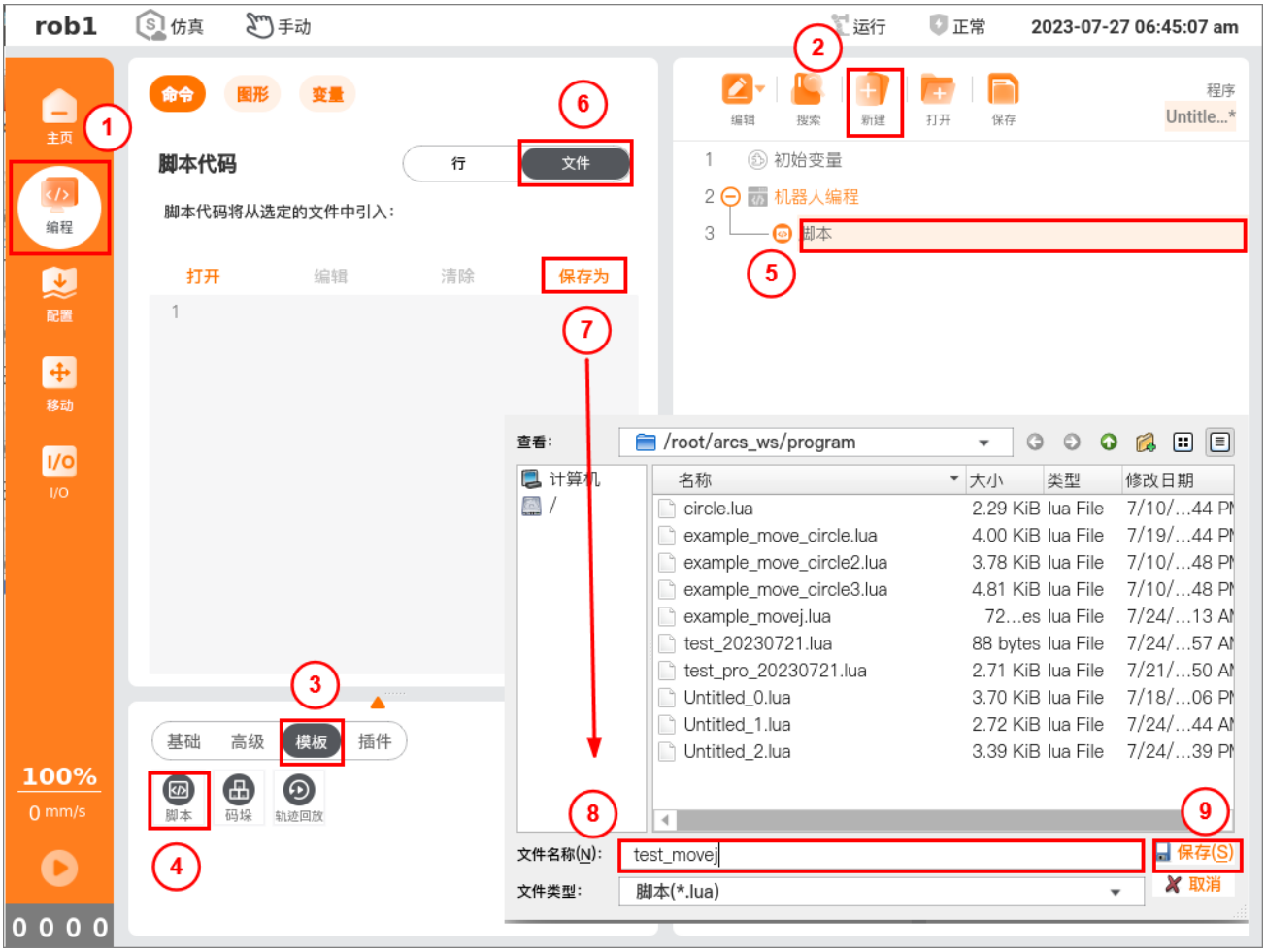
和C++SDK 的API一样，ARCS脚本API定义在SDK软件包的include文件夹的头文件里。

以motion_control.h中moveJoint接口为例，注释中的Lua函数原型就是Lua函数定义。

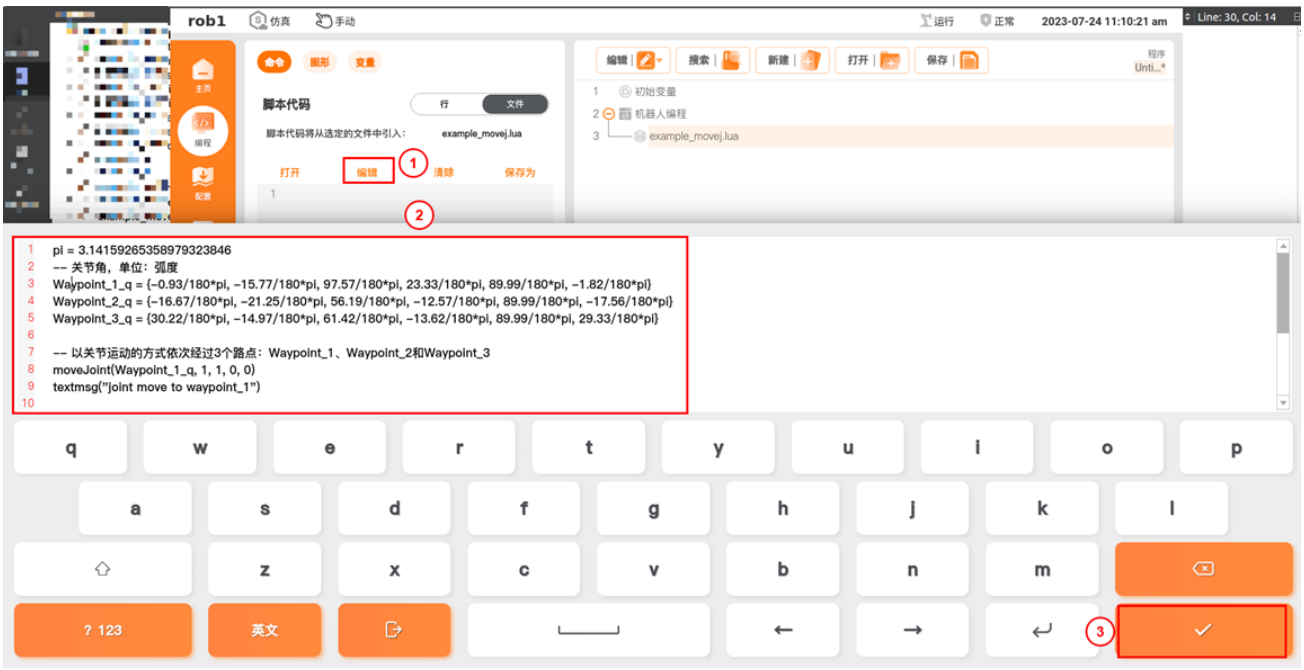
4. 在示教器中新建和编写脚本的操作步骤

注：示教器中的.pro文件和.lua文件均保存在 /root/arcs_ws/program路径下

1. 在示教器中新建一个脚本，命名为test_movej，点击保存



2. 点击编辑，编写test_movej.lua程序



test_movej.lua代码：

```
    pi = 3.14159265358979323846
    -- 关节角，单位：弧度
    Waypoint_1_q = {-0.93/180*pi, -15.77/180*pi, 97.57/180*pi, 23.33/180*pi, 89.99/180*pi, -1.82/180*pi}
    Waypoint_2_q = {-16.67/180*pi, -21.25/180*pi, 56.19/180*pi, -12.57/180*pi, 89.99/180*pi, -17.56/180*pi}
    Waypoint_3_q = {30.22/180*pi, -14.97/180*pi, 61.42/180*pi, -13.62/180*pi, 89.99/180*pi, 29.33/180*pi}

    -- 设置运动速度比率
    setSpeedFraction(0.8)
    -- 以关节运动的方式依次经过3个路点：Waypoint_1、Waypoint_2和Waypoint_3
    moveJoint(Waypoint_1_q, 1, 1, 0, 0)
    -- 将信息打印到aubo_control日志中
    textmsg("joint move to waypoint_1")

    moveJoint(Waypoint_2_q, 1, 1, 0, 0)
    textmsg("joint move to waypoint_2")

    moveJoint(Waypoint_3_q, 1, 1, 0, 0)
    textmsg("joint move to waypoint_3")

```

3. 保存.pro工程，命名为pro_movej。注意.pro文件名不能和.lua文件名相同。

因为在保存.pro工程的时候，同时会在/root/arcs_ws/program路径下生成一个与.pro文件名相同的.lua文件，机器人实际执行的是这个lua文件。


rob1
仿真
手动
运行
正常
2023-07-27 07:01:10 am
主页
编程
配置
移动
I/O
命令
图形
变量
脚本代码
行
文件
脚本代码将从选定的文件中引入：
test_movej.lua
打开
编辑
清除
保存为
1 pi = 3.14159265358979323846
2 -- 关节角，单位：弧度
3 Waypoint_1_q = {-0.93/180*pi, -15.77/180*pi, 97.5
4 Waypoint_2_q = {-16.67/180*pi, -21.25/180*pi, 56.
5 Waypoint_3_q = {30.22/180*pi, -14.97/180*pi, 61.4
6
7 -- 以关节运动的方式依次经过3个路点：Waypoint_1、
8 moveJoint(Waypoint_1_q, 1, 1, 0, 0)
9 textmsg("joint move to waypoint_1")
10
11 moveJoint(Waypoint_2_q, 1, 1, 0, 0)
12 textmsg("joint move to waypoint_2")
13
14 moveJoint(Waypoint_3_q, 1, 1, 0, 0)
15 textmsg("joint move to waypoint_3")
基础
高级
模板
插件
脚本
码垛
轨迹回放
100%
0 mm/s
编辑
搜索
新建
打开
保存
程序
pro_m...
1 初始变量
2 机器人编程
3 test_movej.lua
1
2
rob1
仿真
手动
运行
正常
2023-07-27 07:02:09 am
1 local aubo = require('aubo')
2 local sched = sched or aubo.sched
3 local math = aubo.math or math
4
5 local sleep = sched.sleep
6 local thread = sched.thread
7 local sync = sched.sync
8 local run = sched.run
9 local kill = sched.kill
10 local halt = sched.halt
11 function p_pro_movej()
12 local _ENV = sched.select_robot(1)
13 setCollisionStopType(0)
14 setCollisionLevel(6)
15 setHomePosition({0,-0.2617993877991494,1.7453292519943
16 modbusDeleteAllSignals()
17 setDigitalInputActionDefault()
18 setDigitalOutputRunstateDefault()
19 setPayload(0, {0,0,0}, {0,0,0}, {0,0,0,0,0,0,0,0,0})
20 setTcpOffset({0,0,0,0,0,0})
21 setToolVoltageOutputDomain(0)
22 setToolIoInput(1, true)
23 setToolIoInput(2, true)
24 setToolIoInput(3, true)
25 setToolIoInput(0, false)
26 setPlanContext(sched.current_thread_id(), 1, "初始变量")
27 u57fau5ea7 = {0,0,0,0,0,0}
机器人运行工程实质上是
运行这个脚本文件

所以，如果在工程中添加的.lua程序的文件名与.pro工程相同的话，在某些情况下，很有可能覆盖掉这个机器人实质要运行的脚本文件，导致工程无法运行。

4. 运行工程

5. SDK脚本编程

5.1 介绍

ARCS SDK中的SCRIPT模块可以发送本地脚本程序至机器人控制器，来完成对机器人的控制。SCRIPT模块的接口定义在script.h中，其中sendFile和sendString接口可以实现机器人运行本地脚本。

script.h deprecated

```
/**
 * 发送脚本文件
 *
 * 远程调用本地的脚本
 *
 * @param path 脚本文件在本地的路径
 * @retval 0 成功
 * @retval -1 失败
 */
int sendFile(const std::string &path);

/**
 * 发送脚本内容
 *
 * 调用本地的脚本
 *
 * @param script 脚本内容
 * @retval 0 成功
 * @retval -1 失败
 */
int sendString(const std::string &script);
```

调用本地的脚本

5.2 SDK脚本编程操作步骤

1. 在本地编写Lua脚本程序。

（1）程序开头要导入aubo等库

```
1 local aubo = require('aubo')
2 local sched = sched or aubo.sched
3 local math = aubo.math or math
4
5 local sleep = sched.sleep
6 local thread = sched.thread
7 local sync = sched.sync
8 local run = sched.run
9 local kill = sched.kill
10 local halt = sched.halt
11 function p_pro_movej()
12   local _ENV = sched.select_robot(1)
13   setCollisionStopType(0)
14   setCollisionLevel(6)
15   setHomePosition({0,-0.2617993877991494,1.74532925199433,0.4363323129985824,1.57
16   modbusDeleteAllSignals()
```

（2）程序结尾要添加启动插件

```
67 local app = {}
68
69
70 app.PRIORITY = 1000 -- set the app priority, which determines app execution order
71 app.VERSION = "0.1"
72 app.VENDOR = "Aubo Robotics"
73
74 function app:start(api)
75   --
76   self.api = api
77   print("start---")
78   p_Untitled_2()   被调用的函数
79 end
80
81 function app:robot_error_handler(name, err)
82   --
83   print("An error hanppen to robot "..name)
84 end
85
86 -- return our app object
87 return app
```

（3）之前在示教器脚本编写的test_movej.lua可以修改如下，保存为test_movej2.lua

```lua
local aubo = require('aubo')
local sched = sched or aubo.sched
local math = aubo.math or math

local sleep = sched.sleep
local aubo = require('aubo')
local sched = sched or aubo.sched
local math = aubo.math or math

local sleep = sched.sleep
local thread = sched.thread
local sync = sched.sync
local run = sched.run
local kill = sched.kill
local halt = sched.halt

function p_movej()
    local _ENV = sched.select_robot(1)
    pi = 3.14159265358979323846

    -- 关节角，单位：弧度
    Waypoint_1_q = {-0.93/180*pi, -15.77/180*pi, 97.57/180*pi,
23.33/180*pi, 89.99/180*pi, -1.82/180*pi}
```

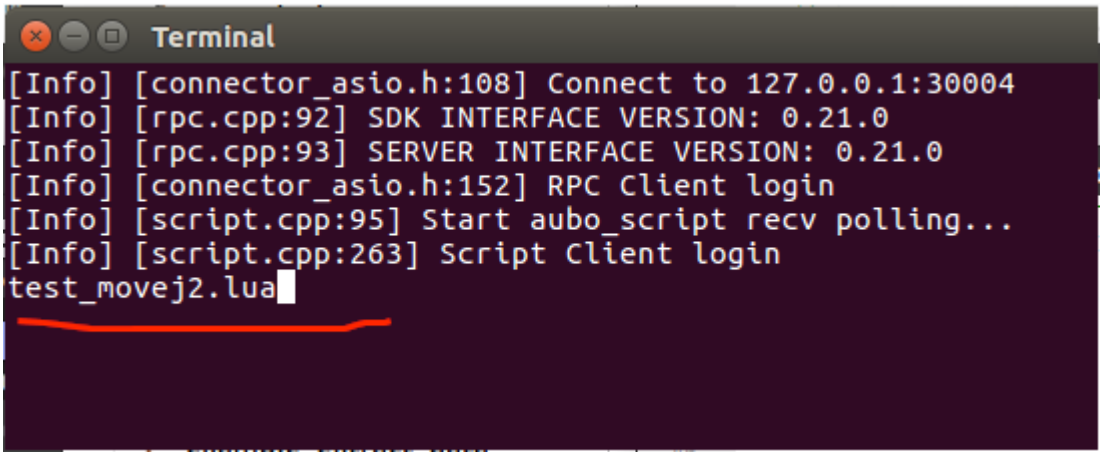
```
23      Waypoint_2_q = {-16.67/180*pi, -21.25/180*pi, 56.19/180*pi,
    -12.57/180*pi, 89.99/180*pi, -17.56/180*pi}
24      Waypoint_3_q = {30.22/180*pi, -14.97/180*pi, 61.42/180*pi,
    -13.62/180*pi, 89.99/180*pi, 29.33/180*pi}
25
26      -- 设置运动速度比率
27      setSpeedFraction(0.8)
28      -- 以关节运动的方式依次经过3个路点：Waypoint_1、Waypoint_2和Waypoint_3
29      moveJoint(Waypoint_1_q, 1, 1, 0, 0)
30      -- 将信息打印到aubo_control日志中
31      textmsg("joint move to waypoint_1")
32
33      moveJoint(Waypoint_2_q, 1, 1, 0, 0)
34      textmsg("joint move to waypoint_2")
35
36      moveJoint(Waypoint_3_q, 1, 1, 0, 0)
37      textmsg("joint move to waypoint_3")
38  end
39
40
41  local app = {}
42
43
44  app.PRIORITY = 1000 -- set the app priority, which determines app
    execution order
45  app.VERSION = "0.1"
46  app.VENDOR = "Aubo Robotics"
47
48  function app:start(api)
49    --
50    self.api = api
51    print("start---")
52    p_movej()
53  end
54
55  function app:robot_error_handler(name, err)
56    --
57    print("An error hanppen to robot "..name)
58  end
59
60  -- return our app object
61  return app
62
63
64
```

或者也可以直接用前面示教器脚本编程中的pro_movej.lua文件。

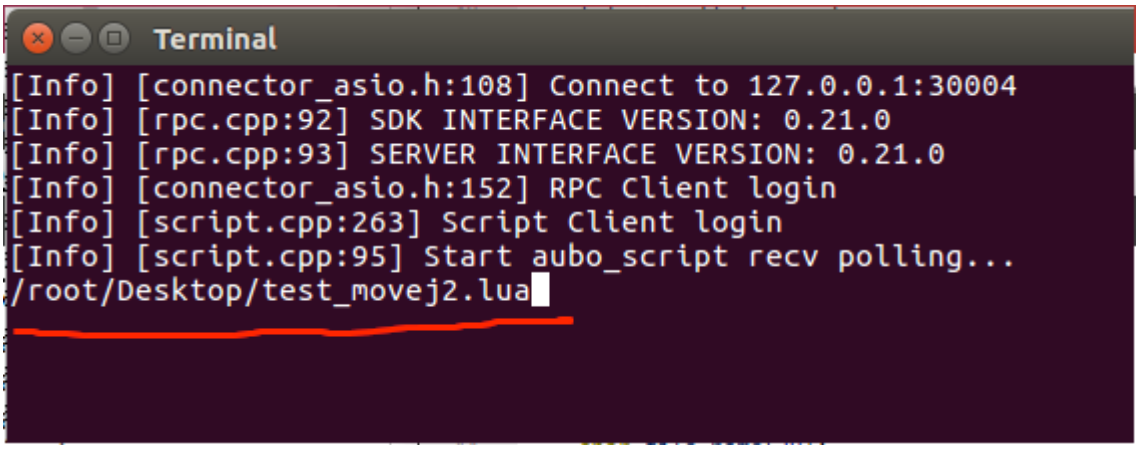
2. 用QtCreator 打开 example_script.cpp 示例，将test_movej2.lua或者pro_movej.lua拷贝至虚拟机中。在运行代码前要确保机器人已上电，处于运行状态。或者在example_script.cpp增加机器人上电的代码。

这里有两种方法去执行test_movej2.lua文件：

方法1：将lua文件放到程序的可执行目录中，即example/c++/build/bin文件夹中，运行example_script.cpp代码之后，在终端输入test_movej2.lua即可



方法2：将lua文件放到非可执行目录中，例如桌面，那么运行example_script.cpp代码之后，在终端输入test_movej2.lua的路径（/root/Desktop/test_movej2.lua）即可



3. 由于example_script.cpp示例中没有用到rpc服务，所以可以将代码中的相关部分注释掉或去掉。

```cpp
int main(int argc, char **argv)
{
#ifdef WIN32
    // 将Windows控制台输出代码页设置为 UTF-8
    SetConsoleOutputCP(CP_UTF8);
#endif
    //    auto rpc = std::make_shared<RpcClient>();
    //    // 接口调用: 连接到 RPC 服务
    //    rpc->connect(LOCAL_IP, 30004);
    //    // 接口调用: 登录
    //    rpc->login("aubo", "123456");

    auto script = std::make_shared<ScriptClient>();
    // 接口调用: 连接到 SCRIPT 服务
    script->connect(LOCAL_IP, 30002);
    // 接口调用: 登录
    script->login("aubo", "123456");

    // 输入脚本文件名
    char file_name[20];
    cin >> file_name;

    // 打开文件
    ifstream file;
    file.open(file_name);
    // 如果打开文件失败，则退出程序
    if (!file) {
        cout << "open fail." << endl;
        exit(1);
    }

    // 读取并打印脚本文件中的内容
    std::string str_all;
    while (!file.eof()) {
        std::string str_line;
        getline(file, str_line);
        str_all += str_line;
        str_all += "\n";
    }
    str_all += "\r\n\r\n";
    cout << "脚本内容:" << endl << str_all << endl;
    file.close();

    //    rpc->getRuntimeMachine()->abort();
    // 发送脚本内容到控制器
    script->sendString(str_all);

    // 增加阻塞来保证执行完脚本
    while (1) {
        std::this_thread::sleep_for(std::chrono::seconds(1));
    }

    return 0;
}
```