DragonMAT 二次开发 SDK

使用指南

版本号：**1.1**

发布时间：**2021-01-15**

版本历史

版本	日期	责任人	版本描述
1.1	2021-01-15	AWA1695	1.增加第 1 章。 2.更新文档模板和排版。
1.0	2020-08-04	KPA0398	创建文档。

目录

版本历史 ii
目录 iii
1 前言 1
1.1 文档简介 1
1.2 目标读者 1
1.3 适用范围 1
1.4 文档约定 1
1.4.1 标志说明 1
2 概述 2
3 使用操作说明 3
3.1 所用到的插件库 3
3.2 插件的使用说明 3
3.2.1 头文件包含 3
3.2.2 定义全局变量 4
3.2.3 调用 LoadPluginCenter 4
3.2.4 初始化 Plugin_info 4
3.2.5 调用 LoadLibrary 5
3.2.6 QueryInterface 获取接口 5
4 软件应用场景和架构图 6
5 接口说明 7
6 信息号说明 10
7 目录说明 12
8 驱动安装 13

1 前言

1.1 文档简介

本文档介绍 DragonMAT 二次开发包的使用方法。

1.2 目标读者

希望对 DragonMAT 进行二次开发的开发者。

1.3 适用范围

开发环境：Windows 10 + Visual Studio 2015

1.4 文档约定

1.4.1 标志说明

本文档采用各种醒目的标志来表示在操作过程中应该特别注意的地方，这些标志的含义如下：

标识	说明
 警告	该标志后的说明应给予格外关注，如果不遵守，可能会导致人员受伤或死亡。
 注意	提醒操作中应注意的事项。不当的操作可能会损坏器件，影响可靠性、降低性能等。
 说明	为准确理解文中指令、正确实施操作而提供的补充或强调信息。
 窍门	一些容易忽视的小功能、技巧。了解这些功能或技巧能帮助解决特定问题或者节省操作时间。

2 概述

工厂自动化测试工具开发包（MAT SDK）是用来提供自动化测试工具二次开发的软件开发包。

作用：能够在 PC 端通过 ADB（Android Debug Bridge）通讯方式抓取保存在待测设备中的测试项，并且按照顺序运行测试项上的测试功能。其中自动测试项可全自动完成，而手动测试项，例如播放声音，测试员听到声音后通过点击界面上的“通过”按键来完成测试。测试项全部测试完成后，将测试结果显示出来并保存成文件记录。

3 使用操作说明

3.1 所用到的插件库

plgvector.dll---调用接口 LoadPluginCenter() 作用：插件管理

AdbSocket.dll---调用接口 GetAdbInterface() 作用：ADB 操作

usbbinder.dll---调用接口 GetUSBInterface() 作用：USB 设备和窗口绑定

awatserver.dll---调用接口 GetAATPInterface() 作用：PC 端和设备端通讯协议

Langplgex.dll--调用接口 GetLang() 作用：多国语言

loghelper.dll--调用接口 awlog() 作用：日志分级打印

注意

前面 4 个是开发中必需用到的，后面 2 各根据需求使用。

3.2 插件的使用说明

3.2.1 头文件包含

说明

参考 demo 中的 StdAfx.h

```
// TODO: 在此处引用程序需要的其他头文件
#include "../../../Plugin/duilib-master/DuiLib/UIlib.h" //注：如果使用其他界面库可不包含
#include "../includes/lang_if.h"
#include "../includes/IAWLog.h"
#include "../includes/IAWUSBBind.h"
#include "../includes/IAdbSocket.h"
#include "../includes/IAWATServer.h"
#include "InterfaceUtil.h"
#include "SimpleInstance.h"

//引用外部的 lib 文件
using namespace DuiLib;
#ifdef _DEBUG #pragma comment(lib, "..\\Lib\\DuiLib_d.lib")
# pragma comment(lib, "..\\Lib\\lib_json_d.lib")
#else
#pragma comment(lib, "..\\Lib\\DuiLib.lib")
#pragma comment(lib, "..\\Lib\\lib_json.lib")
#endif

#pragma comment(lib, "User32.lib")
```

```
#pragma comment(lib,"ws2_32.lib")
#pragma comment(lib,"ole32.lib")
#pragma comment(lib,"Shell32.lib")
#pragma comment(lib,"comdlg32.lib")

#include "common/framework/platform_init.h"
#include <stdio.h>
#include "kernel/os/os.h"
```

3.2.2 定义全局变量

说明

参考 interfaceUtil.cpp

```
Plugin_Man_Fun* CInterfaceUtil::st_pPlgMan(NULL);
IShellAdb* CInterfaceUtil::st_pAdb(NULL);
IAATPServer* CInterfaceUtil::st_pAATP(NULL);
IAWUSBBinder* CInterfaceUtil::st_pUsbBinder(NULL);
Lang_Plg_Interface* CInterfaceUtil::st_pLang(NULL);
```

3.2.3 调用 LoadPluginCenter

调用 LoadPluginCenter 返回插件管理器接口，类型为 Plugin_Man_Fun。

```
st_pPlgMan = LoadPluginCenter();
```

3.2.4 初始化 Plugin_info

初始化 Plugin_info,准备注册要使用的插件。

```
Plugin_info plgInfo;
LoadPlugin(&plgInfo, LIVE_PROC_PLG_ID, LIVE_PROC_PLG_NAME);
```

LIVE_PROC_PLG_ID ——插件的 ID

LIVE_PROC_PLG_NAME ——插件的插件名称

以上两个宏定义在插件接口文件中声明。

示例：

```
Plugin_info plgInfo[] = {
            {ADBSOCKET_PLG_ID, ADBSOCKET_PLG_NAME},
            {AATP_PLG_ID, AATP_PLG_NAME},
            { USBBIND_PLG_ID, USBBIND_PLG_NAME },
            { LANG_PLG_ID_EX, LANG_PLG_NAME_EX },
};

st_pPlgMan = LoadPluginCenter();
  ASSERT(st_pPlgMan);

int ns = sizeof(plgInfo)/ sizeof(Plugin_info);
for(int i = 0; i < ns ; i++){
      if (!st_pPlgMan->LoadPlugin(plgInfo[i])){
```

```
            CDuiString msg;
            msg.Format(_T("Load plugin :\"%s\" failed!,errcode:%d"),plgInfo[i].szModualName,
            GetLastError());
            OutputDebugString(msg);
            return __LINE__;
        }
}
```

3.2.5 调用 LoadLibrary

在 LoadPluginCenter 中调用 LoadLibrary 加载插件，示例：

```
TCHAR szPath[MAX_PATH] = _T("");
GetAbsPathName(PLUGIN_VECTOR_NAME, szPath);
HMODULE hMod = LoadLibrary(szPath);
```

3.2.6 QueryInterface 获取接口

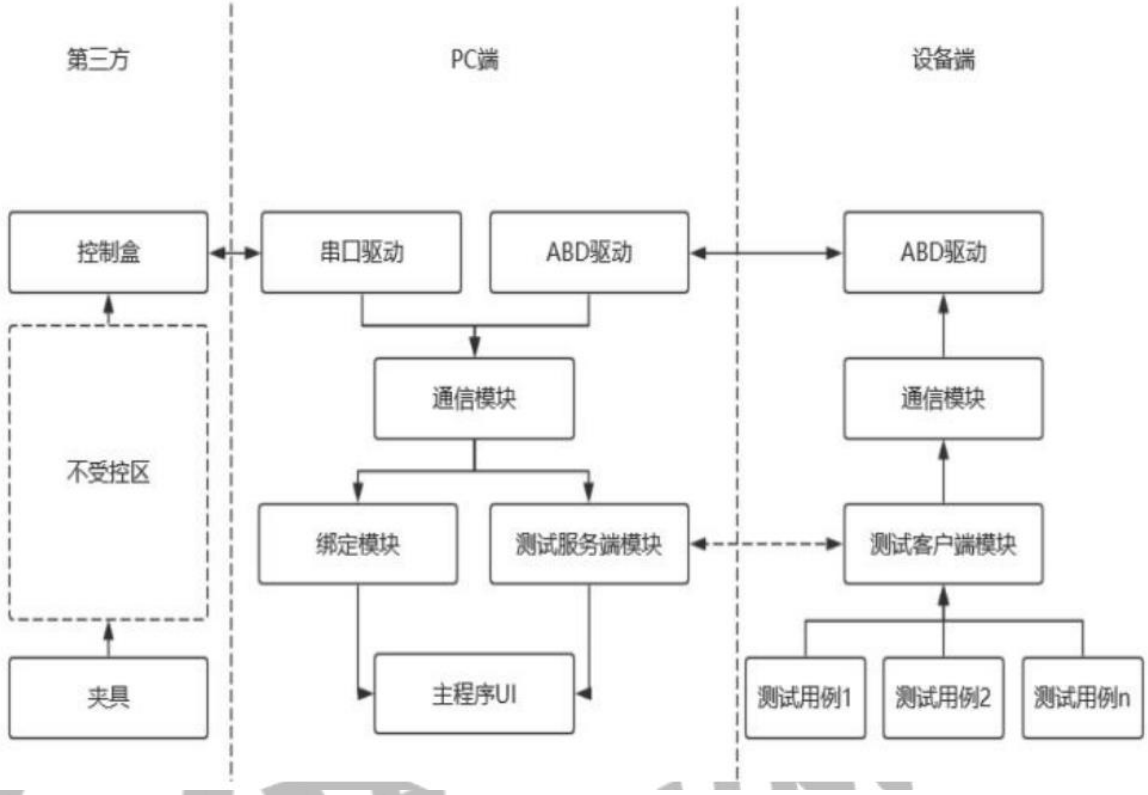
通过 Plugin_Man_Fun 的 QueryInterface 获取相应的接口，示例：

```
st_pAdb = (IShellAdb*)st_pPlgMan->QueryInterface(ADBSOCKET_PLG_ID);
st_pUsbBinder = (IAWUSBBinder*)st_pPlgMan->QueryInterface(USBBIND_PLG_ID);
st_pAATP = (IAATPServer*)st_pPlgMan->QueryInterface(AATP_PLG_ID);
st_pNet = (IAWNet*)st_pPlgMan->QueryInterface(NET_PLG_ID);
st_pReport = (IAWReport*)st_pPlgMan->QueryInterface(AWREPORT_PLUGIN_ID);
st_pLang = (Lang_Plg_Interface*)st_pPlgMan->QueryInterface(LANG_PLG_ID_EX);
```

4 软件应用场景和架构图

软件架构图如图 4-1 所示。

图 **4-1** 软件架构示意图



本文档主要聚焦 PC 端开发，主程序 UI 以 DUILIB 为例。（串口部分可与第三方合作商自行拓展）。

5 接口说明

在测试服务端这一层，建立多个待测设备实例，在实例中通过如下接口开始/停止测试。

1.PC 端和设备端通讯插件接口

```
typedef struct _TAATPServer
{
   AATP_Init Init;
   AATP_CreateServer CreateServer;
   AATP_ReleaseServer ReleaseServer;
   AATP_StartListen StartListen;
   AATP_StopListen StopListen;
   AATP_SendRequest SendRequest;
}IAATPServer;
```

2.初始化服务

```
GetAATPInterface()->Init(GetAdbInterface(), NULL);
```

3.启动服务

```
HAATPServer CreateServer(int handle, const char* szAuthor, const char* szTime, const TAATPClient* pDev, int iPort)
```

4.开始测试

```
int StartListen(HAATPServer hServer, const char* szXml, AATP_ListenHandle fnHandle)
```

5.发送请求（留给测试失败时，双击重测的接口）

```
int SendRequest(HAATPServer hServer, const TAATPRequest* pRequest)
```

6.停止测试

```
int StopListen(HAATPServer hServer)
```

7.关闭服务

```
int ReleaseServer(HAATPServer hServer)
```

8.USB 绑定插件接口

```
typedef struct _tag_IAWUSBBinder
{
   IAWUsbbinder_InitBinder InitBinder;
   IAWUsbbinder_ReleaseBinder ReleaseBinder;
   IAWUsbbinder_AddFliter AddFliter;
   IAWUsbbinder_FliterUSBEvent FliterUSBEvent;
   IAWUsbbinder_GetDeviceCount GetDeviceCount;
   IAWUsbbinder_GetDeviceByIndex GetDeviceByIndex;
   IAWUsbbinder_Update Update;
   IAWUsbbinder_SendFliterMsg SendFliterMsg;
   IAWUsbbinder_GetAllPluginedDevices GetAllPluginedDevices;
}IAWUSBBinder;
```

9.初始化绑定器

```
LRESULT InitBinder(HWND hWnd, void* pAdb, void* pData, IAWUsbbinder_SendMessage cb);
```

10.释放绑定资源

```
LRESULT ReleaseBinder();
```

11.增加消息过滤通知者

```
LRESULT AddFliter(IAWUsbbinder_Fliter pfnFliter);
```

12.过滤 USB WIN32 消息

```
LRESULT FliterUSBEvent(HWND hWnd, UINT message,WPARAM wParam, LPARAM lParam);
```

13.获取设备数量

```
int GetDeviceCount();
```

14.根据索引获取设备信息

```
TUSBDev* GetDeviceByIndex(int nIndex);
```

15.更新设备绑定信息

```
void Update(int nIndex);
```

16.发送消息

```
Int SendFliterMsg(int Event, TUSBDev* pWork);
```

17.获取所有已插入设备的信息

```
Int GetAllPluginedDevices(unsigned int uPid, unsigned int uVid, TAWUSBDeviceArray* pArray);
```

18.ADB 插件接口

```
typedef struct _tag_IShellAdb
{
    pIAdbSocket_InitializeEvn InitializeEvn;
    pIAdbSocket_ExecuteCmd ExecuteCmd;
    pIAdbSocket_FreeBuff FreeBuff;
    pIAdbSocket_RawSetup RawSetup;
    pIAdbSocket_Setup Setup;
    pIAdbSocket_AdbConnect AdbConnect;
    pIAdbSocket_AdbSend AdbSend;
    pIAdbSocket_AdbRecv AdbRecv;
    pIAdbSocket_Pull Pull;
    pIAdbSocket_Push Push;
    pIAdbSocket_Close Close;
    pIAdbSocket_AdbDevices AdbDevices;
    pIAdbSocket_WaitForDevice WaitForDevice;
}IShellAdb;
```

19.初始化 ADB 环境

```
int InitializeEvn();
```

20.执行命令

```
int ExecuteCmd(const char* commandstr,int nTimeOut, char** pOut, int* nOut);
```

21.释放 ExecuteCmd 接口返回的指针缓存

```
int FreeBuff(char* pBuff);
```

22.创建一个 adb forward 实例

```
HADB RawSetup(const char* szId, int nPort, const char* szExec, const char* flag);
```

23.已弃用

```
Setup（）；
```

24.连接设备端端口

```
int AdbConnect(HADB hAdb);
```

25.发送 SOCKET 数据

```
int AdbSend(HADB hAdb, const char* szData, int nLen, int nTimeout);
```

26.接收 SOCKET 数据

```
int AdbRecv(HADB hAdb, char* szData, int nRecvLen, int nTimeout);
```

27.ADB PULL

```
int Pull(HADB hAdb, const char* szRemote, const char* szLocal);
```

28.ADB PUSH

```
int Push(HADB hAdb, const char* szLocal, const char* szRemote);
```

29.释放一个 adb forward 实例

```
int Close(HADB hAdb);
```

30.封装 adb devices

```
int AdbDevices(TAdbDeviceArray* pArray);
```

31.封装 adb wait-for-device

```
int WaitForDevice(HADB hAdb, int nTimeOut);
```

6 信息号说明

请先参考《DragonMAT 交互协议》，了解交互流程和事件定义。

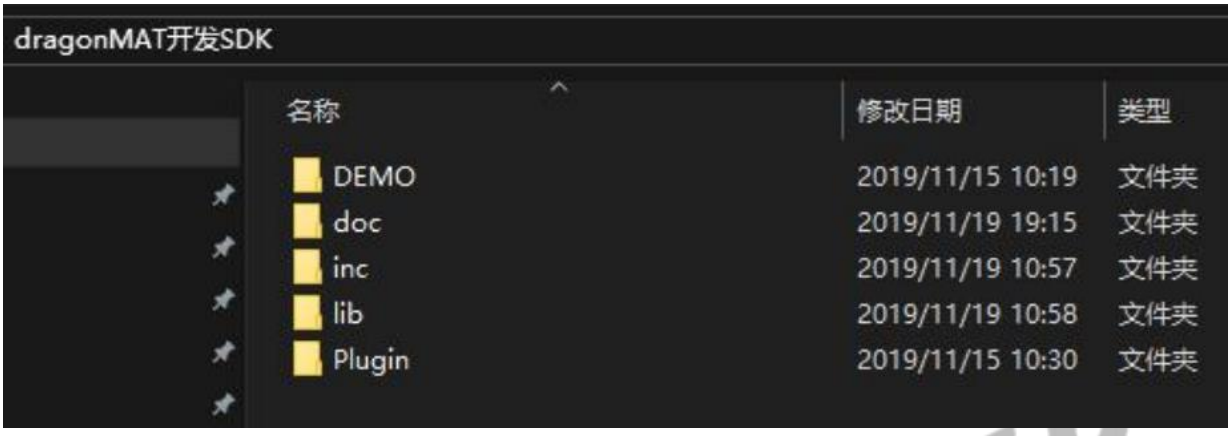
```
#define  AATP_MSG_LOST_HEARTBEAT  (0x0001L)  //丢失心跳包
#define  AATP_MSG_BAD_REQ_FORMAT  (0x0002L)  //错误的请求指令
#define  AATP_MSG_HANDLE_ERROR  (0x0003L)  //未使用
#define  AATP_MSG_LISTEN_EXIT  (0x0004L)  //未使用
#define  AATP_MSG_EVENT_TIP  (0x0005L)  //处理 TIP 类型事件
#define  AATP_MSG_EVENT_SELECT  (0x0006L)  //处理 SELECT 类型事件
#define  AATP_MSG_EVENT_START_ONE  (0x0007L)  //启动一组测试
#define  AATP_MSG_EVENT_END_ONE  (0x0008L)  //结束一组测试
#define  AATP_MSG_EVENT_OPERATOR  (0x0009L)  //刷新 TIP 上的文字和颜色
#define  AATP_MSG_EVENT_FINISH  (0x000AL)  //测试完成
#define  AATP_MSG_EVENT_START_CMB  (0x000BL)  //未使用
#define  AATP_MSG_EVENT_END_CMB  (0x000CL)  //未使用
#define  AATP_MSG_EVENT_DOWNLOAD  (0x000DL)  //PC 下载文件到设备
#define  AATP_MSG_EVENT_DOWNLOAD_GO  (0x000EL)  //未使用
#define  AATP_MSG_EVENT_UPLOAD  (0x000FL)  //设备上传文件到 PC
#define  AATP_MSG_EVENT_NET_ERR  (0x0010L)  //网络错误
#define  AATP_MSG_START_APK_FAILED  (0x0011L)  //启动 APK 失败
#define  AATP_MSG_CONNECT_FAILED  (0x0012L)  //连接失败
#define  AATP_MSG_CONNECT_HB_FAILED  (0x0013L)  //连接心跳异常
#define  AATP_MSG_START_TEST  (0x0014L)  //开始测试
#define  AATP_MSG_SELECT_TIMEOUT  (0x0015L)  //选择界面超时
#define  AATP_MSG_LOST_PLUGIN  (0x0016L)  //目标路径插件不存在
#define  AATP_MSG_PLUGIN_ERR  (0x0017L)  //插件无法加载
#define  AATP_MSG_PLUGIN_INIT_ERR  (0x0018L)  //插件预处理失败,报出失败号 //v2.0
#define  AATP_MSG_EVENT_GET_INFO  (0x0019L)  //获取事件信息
#define  AATP_MSG_RELEASE  (0x0020L)  //事件释放
#define  AATP_MSG_OPERATOR_EXCEPTION  (0x0021L)  //操作状态异常
#define  AATP_MSG_OPERATOR_ACTION_OK  (0x0022L)  //操作状态 OK
#define  AATP_MSG_OPERATOR_POSTDO_OK  (0x0023L)  //上一次操作状态 OK
#define  AATP_MSG_EVENT_SHOWRESOURCE  (0x0024L)  //显示资源（例如图片）
#define  AATP_MSG_EVENT_EDIT  (0x0025L)  //编辑框事件
#define  AATP_MSG_EDIT_TIMEOUT  (0x0026L)  //编辑框界面输入超时
#define  AATP_MSG_UPDATE_CONFIG  (0x0027L)  //更新测试用例 //v3.0
#define  AATP_MSG_SESUPEND_BEGIN  (0x0028L)  //保持当前界面状态开始
#define  AATP_MSG_SESUPEND_TIME  (0x0029L)  //保持界面状态时间
#define  AATP_MSG_SESUPEND_END  (0x0030L)  //保持当前界面状态结束
```

```
#define    AATP_MSG_COMMON_RESPONSE    (0x003AL)    //普通应答  //v3.1 FOR SCAN
#define    AATP_MSG_EVENT_SCAN    (0x003BL)    //扫码事件
#define    AATP_MSG_SCAN_RESPONSE    (0x003CL)    //未使用
```

7 目录说明

SDK 包包含目录如图 7-1 所示。

图 **7-1 SDK** 包目录示意图



1. DEMO 目录---DEMO 程序源代码。

2. doc 目录--说明文档目录。

3. Inc 目录--项目需要的头文件。

4. Lib 目录--项目所需要的库文件

5. Plugin 目录--放有第三方 DUILIB 界面库和使用文档。

8 驱动安装

工具需要依赖 ADB 命令工作，使用前请自行安装 ADB 驱动程序。