**WxBox - 详细需求分析文档**

**(v0.1.4)**

**修订历史记录**

|  |  |  |  |
| --- | --- | --- | --- |
| 修改日期 | 版本  （指的是文档的版本号） | 说明 | 作者 |
| 2021-12-25 | v0.1.0 | 文档初始化 | 胡智钦 |
| 2021-12-29 | v0.1.1 | 迭代需求 | 胡智钦 |
| 2021-01-06 | v0.1.2 | 迭代需求，补充WxBoxServer与WxBoxClient | 胡智钦 |
| 2021-01-11 | v0.1.3 | 迭代需求，加入WxBot模块注入 | 胡智钦 |
| 2021-01-16 | v0.1.4 | 加入Plugin | 胡智钦 |
|  |  |  |  |
|  |  |  |  |

目录

[1 引言 4](#_Toc93675183)

[2 WxBox项目 4](#_Toc93675184)

[2.1 非功能性需求 4](#_Toc93675185)

[2.2 功能性需求 4](#_Toc93675186)

[2.2.1 根据宏定义检测运行的操作系统【Proj-FR-1】 4](#_Toc93675187)

[3 WxBox模块 5](#_Toc93675188)

[3.1 非功能性需求 5](#_Toc93675189)

[3.2 功能性需求 5](#_Toc93675190)

[3.2.1 获取微信安装目录【WxBox-FR-1】 6](#_Toc93675191)

[3.2.2 检查微信安装目录是否有效【WxBox-FR-2】 7](#_Toc93675192)

[3.2.3 获取微信版本号【WxBox-FR-3】 7](#_Toc93675193)

[3.2.4 获取系统中的微信进程列表【WxBox-FR-4】 7](#_Toc93675194)

[3.2.5 根据窗口句柄获取所在进程【WxBox-FR-5】 7](#_Toc93675195)

[3.2.6 解除微信单例【WxBox-FR-6】 8](#_Toc93675196)

[3.2.7 定义Hook特征结构【WxBox-FR-7】 8](#_Toc93675197)

[3.2.8 检查指定微信进程是否可用【WxBox-FR-8】 11](#_Toc93675198)

[3.2.9 WxBox Server【WxBox-FR-9】 11](#_Toc93675199)

[3.2.10 模块注入与卸载【WxBox-FR-10】 21](#_Toc93675200)

[3.2.11 强制重置PluginVirtualMachine【WxBox-FR-11】 21](#_Toc93675201)

[4 WxBot模块 22](#_Toc93675202)

[4.1 非功能性需求 22](#_Toc93675203)

[4.2 功能性需求 22](#_Toc93675204)

[4.2.1 Plugin【WxBot-FR-1】 22](#_Toc93675205)

[4.2.2 Plugin命令语法测试工具【WxBot-FR-2】 26](#_Toc93675206)

# 引言

这个文档用于描述和记录WxBox模块与WxBot模块的需求，并对所有需求进行详细的分析以及设计。这个文档将不断的迭代，不需要一开始就全部补完。

# WxBox项目

## 非功能性需求

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 功能名称 | 简要描述 | 依赖 | 优先级 |
| Proj-NFR-1 |  |  |  |  |

## 功能性需求

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 功能名称 | 简要描述 | 依赖 | 优先级 |
| Proj-FR-1 | [识别操作系统](#_根据宏定义检测运行的操作系统) | 根据宏定义检测运行的操作系统 |  | 必须 |

### 根据宏定义检测运行的操作系统【Proj-FR-1】

这个通过meson.build构建脚本在configure的时候根据所在操作系统生成config.h文件，在项目中可以通过引入这个头文件，用WXBOX\_PLATFORM宏来判断是什么操作系统，这个宏是一个整型值，它的所有可能值如下：

|  |  |
| --- | --- |
| 宏 | 值 |
| WXBOX\_UNSUPPORT\_OS | 0 |
| WXBOX\_WINDOWS\_OS | 1 |
| WXBOX\_MAC\_OS | 2 |

# WxBox模块

## 非功能性需求

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 功能名称 | 简要描述 | 依赖 | 优先级 |
| WxBox-NFR-1 |  |  |  |  |

## 功能性需求

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 功能名称 | 简要描述 | 依赖 | 优先级 |
| WxBox-FR-1 | [获取微信安装目录](#_获取微信安装目录) | 获取微信安装目录，能应对正常和非常规安装微信的环境 |  | 必须 |
| WxBox-FR-2 | [检查微信安装目录是否有效](#_检查指定的安装路径是否有效【WxBox-FR-2】) | 检查指定的微信安装路径是否有效 | WxBox-FR-1 | 必须 |
| WxBox-FR-3 | [获取微信版本号](#_获取微信版本号) | 主要以关键模块的版本号来作为微信的版本号，比如Windows下使用WeChatWin.dll的文件版本号 | WxBox-FR-1 | 必须 |
| WxBox-FR-4 | [获取系统中的微信进程列表](#_获取系统中微信进程列表【WxBox-FR-4】) | 获取系统中所有进程的信息 |  | 必须 |
| WxBox-FR-5 | [识别窗口进程](#_根据窗口句柄获取所在进程) | 根据窗口的句柄获取它所属的进程信息 |  | 可选 |
| WxBox-FR-6 | [解除微信单例](#_解除微信单例【WxBox-FR-6】) | 通过WxBox启动新的微信实例 | WxBox-FR-3 | 必须 |
| WxBox-FR-7 | [定义Hook特征结构](#_定义Hook特征结构【WxBox-FR-7】) | 定义已知版本信息的Hook点与特征信息的存储方式 |  | 必须 |
| WxBox-FR-8 | [检查指定微信进程是否可用](#_检查指定微信进程是否可用【WxBox-FR-8】) | 检查指定微信进程对于WxBot需要的所有API是否都能定位到Hook点 | WxBox-FR-7 | 必须 |
| WxBox-FR-9 | [WxBox Server](#_WxBox_Server【WxBox-FR-9】) | 与WxBot交互 |  | 必须 |
| WxBox-FR-10 | [模块注入与卸载](#_模块注入与卸载【WxBox-FR-10】) | dll/so模块注入与卸载 |  | 必须 |
| WxBox-FR-11 | [强制重置PluginVirtualMachine](#_强制重置PluginVirtualMachine【WxBox-FR-1) | 向所有WxBoxClient发出指令，强制重置PluginVirtualMachine |  | 必须 |
| WxBox-FR-12 | 微信多开 |  |  | 必须 |

### 获取微信安装目录【WxBox-FR-1】

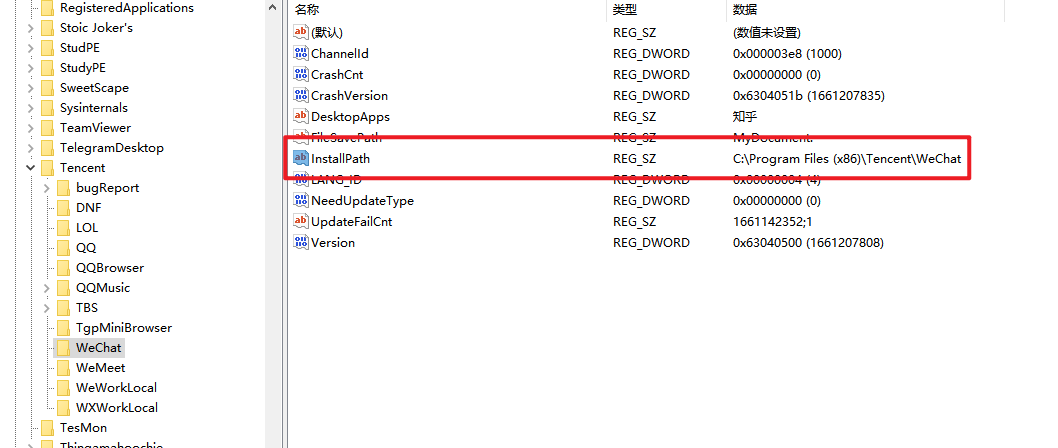
#### Windows

##### 正常安装

在Windows下如果是正常安装的话，可以在注册表的以下位置获取到微信的安装路径

HKEY\_CURRENT\_USER\SOFTWARE\Tencent\WeChat#InstallPath

这个InstallPath是一个REG\_SZ类型的值



【微信安装目录在注册表中的位置】

关于注册表的Windows API可以看MSDN文档：<https://docs.microsoft.com/zh-cn/windows/win32/sysinfo/registry-functions>。

主要使用的API是[RegGetValue](https://docs.microsoft.com/zh-cn/windows/win32/api/winreg/nf-winreg-reggetvaluea)。

##### 非常规安装

如果微信安装路径在注册表中找不到的话，可能是系统上并没有安装微信，也可能使用的不是常规的安装方法，比如绿色版之类的。这时候让用户手动指定微信的安装位置。

#### Mac OS

### 检查微信安装目录是否有效【WxBox-FR-2】

#### Windows

在Windows下检查安装目录是否存在WeChat.exe和WeChatWin.dll，以此来决定安装目录是否有效。

#### Mac OS

### 获取微信版本号【WxBox-FR-3】

#### Windows

在Windows下以WeChatWin.dll的文件版本来作为微信的版本号。相关函数API有：[GetFileVersionInfoSizeA](https://docs.microsoft.com/en-us/windows/win32/api/winver/nf-winver-getfileversioninfosizea)、[GetFileVersionInfo](https://docs.microsoft.com/en-us/windows/win32/api/winver/nf-winver-getfileversioninfoa)和[VerQueryValueA](https://docs.microsoft.com/en-us/windows/win32/api/winver/nf-winver-verqueryvaluea)。相关结构体：[VS\_FIXEDFILEINFO](https://docs.microsoft.com/en-us/windows/win32/api/verrsrc/ns-verrsrc-vs_fixedfileinfo)。

#### Mac OS

### 获取系统中的微信进程列表【WxBox-FR-4】

#### Windows

在Windows下可以用快照来获取进程列表，关键函数是CreateToolhelp32Snapshot，MSDN文档：[CreateToolhelp32Snapshot](https://docs.microsoft.com/en-us/windows/win32/api/tlhelp32/nf-tlhelp32-createtoolhelp32snapshot)。

#### Mac OS

### 根据窗口句柄获取所在进程【WxBox-FR-5】

#### Windows

在Windows下可以用WindowFromPoint来根据屏幕上的坐标来获取窗口句柄，用GetWindowThreadProcessId来获取该窗口句柄所属的进程，MSDN文档：[WindowFromPoint](https://docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-windowfrompoint)，[GetWindowThreadProcessId](https://docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-getwindowthreadprocessid)。

#### Mac OS

### 解除微信单例【WxBox-FR-6】

#### Windows

在Windows下，做法是这样的：首先需要获取微信的安装目录并确保安装目录是有效的，根据安装目录可以拼接出WeChat.exe的绝对路径，然后通过CreateProcess以DEBUG\_ONLY\_THIS\_PROCESS模式来启动一个WeChat.exe进程并Attach住，这里需要注意因为微信是多进程模式的，所以必须用DEBUG\_ONLY\_THIS\_PROCESS而不是DEBUG\_PROCESS。

Attach住之后就可以进入调试循环了，需要关心的只有一个事件，那就是LOAD\_DLL\_DEBUG\_EVENT事件，即dll的加载事件，这是因为WeChatWin.dll是动态加载的，而我们要做解除微信单例，需要到操作就在WeChatWin.dll上，所以在LOAD\_DLL\_DEBUG\_EVENT事件处识别到加载WeChatWin.dll模块，就可以执行解除操作了。

接下来分为两种情况：一种是对于已知版本，这种是知道了修改点的；第二种是未知版本，需要用特征去定位修改点。

这里以第二种情况来说明做法，首先使用TitanEngine的FindEx方法来在WeChat.exe进程的WeChatWin.dll模块上去搜索特征字符串“**\_WeChat\_App\_Instance\_Identity\_Mutex\_Name**”的VA地址，接下来继续在WeChatWin.dll模块上用FindEx方法搜索引用这个VA地址的地方，这里需要补充说明一下，“**\_WeChat\_App\_Instance\_Identity\_Mutex\_Name**”这种全局字符串在push、mov这些指令的使用方式是这样的，编译生成的exe文件，push、mov后跟的是全局变量的一个RVA地址，在PE可执行文件加载到内存后，这个RVA地址会被重定位成一个VA地址，所以我们在内存上搜索的是VA地址而不是RVA。

定位到引用“**\_WeChat\_App\_Instance\_Identity\_Mutex\_Name**”的位置后，还需要往回定位到这个函数的入口点，找到后那么修改点就找到了。

接下来使用frida-gum的GumX86Writer来生成机器指令，汇编代码是move eax, 0; ret（对于这里其实不需要用到GumX86Writer，直接使用[0xB8, 0x00, 0x00, 0x00, 0x00,0xC3]就可以了）。

最后就是把机器指令替换到找到的函数入口点上，然后脱离WeChat.exe的调试（这里暂时来看必须使用这种调试挂起的方式）。

相关API文档资料：

* [CreateProcess](https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createprocessa)
* [Process Creation Flags](https://docs.microsoft.com/en-us/windows/win32/procthread/process-creation-flags)
* [Creating a Basic Debugger](https://docs.microsoft.com/en-us/windows/win32/debug/creating-a-basic-debuggerhttps:/docs.microsoft.com/en-us/windows/win32/debug/creating-a-basic-debugger)
* [Writing the Debugger's Main Loop](https://docs.microsoft.com/en-us/windows/win32/debug/writing-the-debugger-s-main-loop)
* [WaitForDebugEvent](https://docs.microsoft.com/en-us/windows/win32/api/debugapi/nf-debugapi-waitfordebugevent)
* [ContinueDebugEvent](https://docs.microsoft.com/en-us/windows/win32/api/debugapi/nf-debugapi-continuedebugevent)
* [LOAD\_DLL\_DEBUG\_INFO structure](https://docs.microsoft.com/en-us/windows/win32/api/minwinbase/ns-minwinbase-load_dll_debug_info)
* [x64dbg/TitanEngine](https://github.com/x64dbg/TitanEngine)
* [gumx86writer](https://github.com/frida/frida-gum/blob/main/gum/arch-x86/gumx86writer.h)

#### Mac OS

### 定义Hook特征结构【WxBox-FR-7】

所有的预定义的特征信息使用yaml存储起来。

#### 存储已知微信版本的Hook点方式

对于已知版本的微信的所有Hook点，直接存储对应Hook点的RVA，例如：以下存储Windows微信版本3.4.5.27的Hook点信息。

absolute:

  '3.4.5.27':

    CheckAppSingleton:

      EntryRVA: 0x00C45E20

    FetchGlobalContactContextAddress:

      EntryRVA: 0x00406540

    InitWeChatContactItem:

      EntryRVA: 0x005B7460

    DeinitWeChatContactItem:

      EntryRVA: 0x005B7D40

    FindAndDeepCopyWeChatContactItemWithWXIDWrapper:

      EntryRVA: 0x005BD9A0

    FetchGlobalProfileContext:

      EntryRVA: 0x0009C7B0

    HandleRawMessages:

      EntryRVA: 0x0046BF93

    HandleReceivedMessages:

      EntryRVA: 0x0047152C

    WXSendTextMessage:

      EntryRVA: 0x00442540

    FetchGlobalSendMessageContext:

      EntryRVA: 0x000989A0

    WXSendFileMessage:

      EntryRVA: 0x0043DC20

#### 定义特征信息

特征信息是给未知微信版本定位Hook点用的。分为两步：第一步（扫描）是根据“扫描”特征进行扫描定位到Hook点附近的位置；第二步（定位）是根据“定位”特征从第一步找到的位置来实际定位到Hook点。

##### 扫描（Scan）

扫描方式（ScanType）分为三种：

* ref：例如：根据字符串提供的二进制流，定位到引用这个字符串的位置
* multiPushRef：例如：根据多个字符串的连续引用（连续的push <字符串地址>指令），定位到这个连续引用的起始位置
* instruction：例如：直接根据指令的二进制流，定位到这段指令的起始位置

ref和multiPushRef需要的扫描需要执行两次，而instruction只需要一次。

所有扫描方式的共同参数

|  |  |
| --- | --- |
| 参数名 | 说明 |
| ScanType[必须]<string> | 扫描方式 |

不同类型各自独有的参数：

|  |  |  |
| --- | --- | --- |
| 类型 | 参数说明 | |
| ref | 参数名 | 说明 |
| RefFeatureStream[必须]<uint8\_t[]> | 引用的字节流，例如：单字节字符串的字节流，宽字节字符串的字节流 |
| RefBackExtralInstruction[可选] <uint8\_t[]> | 在找到的VA前附加指令作为Scan的新模式 |
| RefFrontExtralInstruction[可选] <uint8\_t[]> | 在找到的VA后附加指令作为Scan的新模式 |
| multiPushRef | 参数名 | 说明 |
| PushInstruction[必须]<uint8\_t[]> | push指令前缀，这里固定是[0x68] |
| RefFeatureStreams[必须]<uint8\_t[][]> | 引用的多个字节流数组，这是个二维数组，例如：多个字符串字节流的数组 |
| RefFeatureStreamsOffset [必须]<uint8\_t[]> | 给找到的FeatureStreams的偏移 |
| instruction | 参数名 | 说明 |
| InstructionFeatureStream[必须]<uint8\_t[]> | 指令的二进制流 |

##### 定位（Locate）

定位动作（LocateAction）分为四种：

* back：往低地址定位
* front：往高地址定位
* backThenFront：分两步，先往低地址定位，然后再往高地址定位，这里用到两个特征
* backMultiTimes：使用相同的特征，多次往低地址定位

所有定位动作的共同参数

|  |  |
| --- | --- |
| 参数名 | 说明 |
| LocateAction[必须]<string> | 定位动作类型 |
| LocateActionFeatureStream[必须]<uint8\_t[]> | 定位使用的字节流 |
| LocateActionRange[必须]<int> | 定位动作的范围 |
| HookPointOffset[必须]<int> | 定位执行到最后，进行偏移然后得到实际的Hook点 |

不同类型各自独有的参数：

|  |  |  |
| --- | --- | --- |
| 类型 | 参数说明 | |
| back | 参数名 | 说明 |
|  |  |
| front | 参数名 | 说明 |
|  |  |
| backThenFront | 参数名 | 说明 |
| ThenLocateActionFeatureStream[必须]<uint8\_t[]> | 执行front定位使用的字节流 |
| backMultiTimes | 参数名 | 说明 |
| LocateActionExecuteTimes [必须]<int> | back定位动作的重复执行次数 |

### 检查指定微信进程是否可用【WxBox-FR-8】

注入WxBot之前需要检查指定微信进程对于WxBot需要的所有API是否都能定位到Hook点。方法是先检查进程是否存在WxBot模块，如果存在直接调用WxBot来获取Hook点（如果已经注入了WxBot模块那么已经是做完Hook了的），如果不存在那么提取微信版本号，然后检查是否为已知版本，如果是未知版本则使用特征进行扫描，如果所有API都能定位到则，这个微信进程能使用WxBot。

检查完之后还需要把找到的所有Hook点的VA收集起来返回给调用者。

**note: 检查是否已注入WxBot模块现在先不做**

### WxBox Server【WxBox-FR-9】

使用gRPC+protobuf的双向流来让WxBox和WxBot进行交互，采用C/S模式，分为WxBoxServer和WxBoxClient，WxBoxServer运行在WxBox端，WxBoxClient运行在多个WxBot模块上，由WxBot（WxBoxClient）主动连接WxBox（WxBoxServer）。WxBoxServer或者WxBoxClient在处理消息时，都要尽量不进行耗时长的处理。

#### Server端

##### 组成

* WxBoxServer：WxBox主服务循环
* WxBotEndPoint：每个WxBotEndPoint代表一个WxBoxClient，通过WxBotEndPoint与WxBoxClient通信
* Callback：处理消息的回调，通过Qt的信号槽实现

##### 线程安全

在需要的地方使用读写锁来互斥访问。比如：客户端登记需要互斥

##### WxBoxServer服务循环

采用生产者/消费者模型维护一个消息队列，由客户端发送过来的消息会经过这里投放给WxBox，而WxBox向客户端发起的消息也都会经过这里发送给客户端。另外一些管理操作也传递到消息队列中统一处理，例如wxbox主动关闭客户端的连接。一些管理操作不需要通过消息队列，例如，客户端握手时登记endpoint，客户端主动退出后删除登记

##### 客户端管理

###### Handshake

* 客户端使用handshake时指定的pid来标识，如果pid无效或者不是微信进程那么handshake失败
* map<pid, WxBotEndpoint>
* 客户端登记发生在Handshake通过时
* 同一个进程不能有两个WxBoxClient存在，所以如果有特定pid的客户端已经连接并且未断开连接的话，那么这个handshake会失败
* 检查和登记这些都是由WxBotEndPoint交给WxBoxServer来做的，需要注意实际登记时需要互斥写入
* 客户端刚连接时（未握手前），先加入等待队列，握手后再移出等待队列，这是为了防止连接后不进行握手，导致WxBox时等待客户端结束导致无法正常关闭

###### 连接状态

* WxBoxClient主动连接（handshake）
* WxBoxServer验证handshake是否能通过（例如：验证WxBoxClient所在进程是否为微信进程）
* WxBoxClient主动正常退出
* WxBoxClient非正常退出（例如：WxBot所在进程被强制结束掉）
* WxBoxServer主动结束客户端的连接
* WxBoxServer非正常结束客户端的连接（例如：WxBox进程被强制结束掉）
* WxBoxClient存活状态（是否连接正常，可正常通信）

##### 消息

不管server还是client，任何时候只有一个消息在处理

##### 读与写

* WxBoxServer与WxBoxClient全双工通讯，WxBoxServer能主动向WxBoxClient发送请求，反过来WxBoxClient也能向WxBoxServer发送请求
* WxBoxServer接收到客户端发送的消息时，会进入WxBotEndPoint的OnReadDone事件，这时候需要把消息拷贝投递到WxBoxServer消息队列中，WxBoxServer的服务循环会经过Qt的信号把它投送给WxBox的信号槽处理（信号槽的连接方式使用QueuedConnection）
* WxBox给客户端发送消息，先由WxBox向WxBoxServer消息队列投送消息，客户端的标识使用pid，WxBoxServer会在消息循环中把消息实际发送出去

#### Client端

与WxBoxServer的做法差不多

##### 组成

* WxBoxClient：消息循环
* WxBoxEndPoint：通过WxBoxEndPoint与WxBoxServer进行通信
* Callback：处理消息的回调

##### Ping

测试与WxBoxServer的连通性

##### Handshake

通过Handshake与WxBoxServer建立连接

##### 读与写

类似WxBoxServer的做法

##### Wait

等待WxBoxClient循环结束

##### 检测WxBoxServer连接断开

ClientBidiReactor在Read和Write结束后会进入OnReadDone或OnWriteDone，它们的原型是bool OnReadDone(bool ok)、bool OnWriteDone(bool ok)，当ok为false时表明读或写失败，可以用这个来作为WxBoxServer断开连接（正常或非正常断开）的标志，只要读或写失败那么就认为WxBoxServer断开了，并且将进入断开重连状态。

##### 主动断开连接

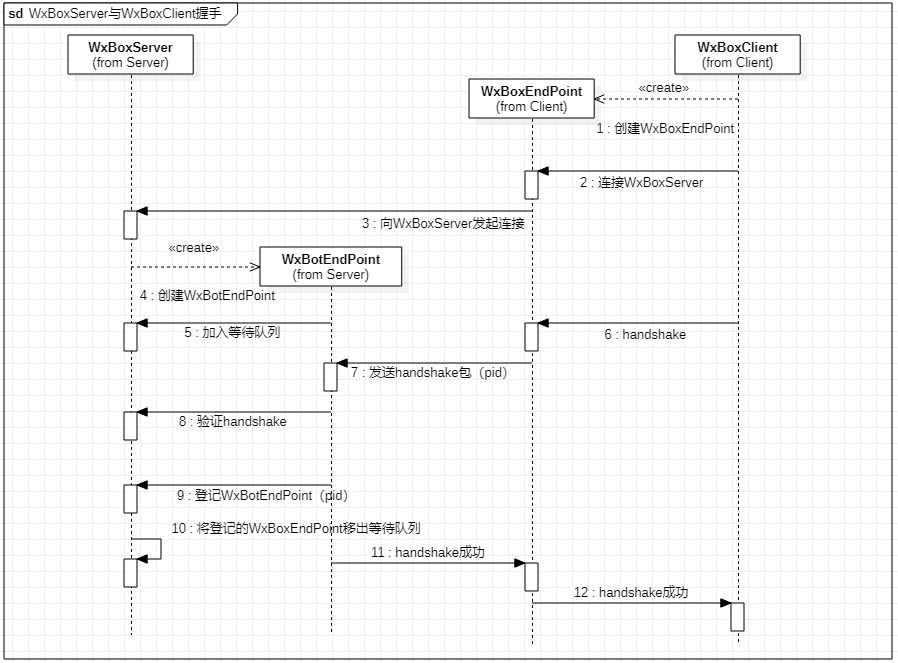
ClientBidiReactor在没有Read和Write时要保持存活状态，需要调用AddHold，那么断开连接就调用RemoveHold，然后等Read和Write全部结束就可以了

##### WxBoxServer断开重连

在WxBoxServer关闭后（例如：WxBox进程被强制退出），WxBot需要每隔一段时间尝试重新连接WxBoxServer

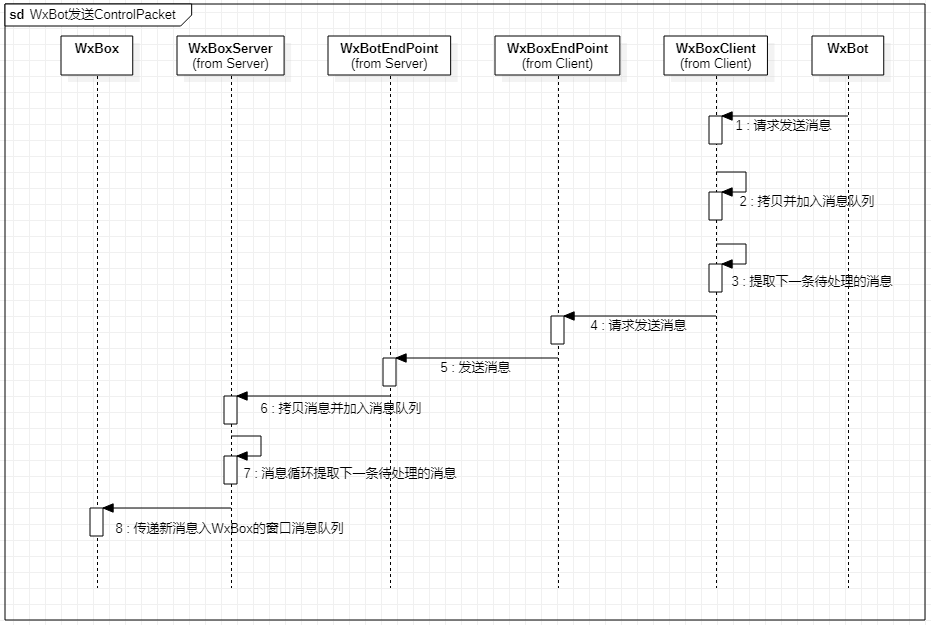
#### Server与Client的交互流程

##### 握手



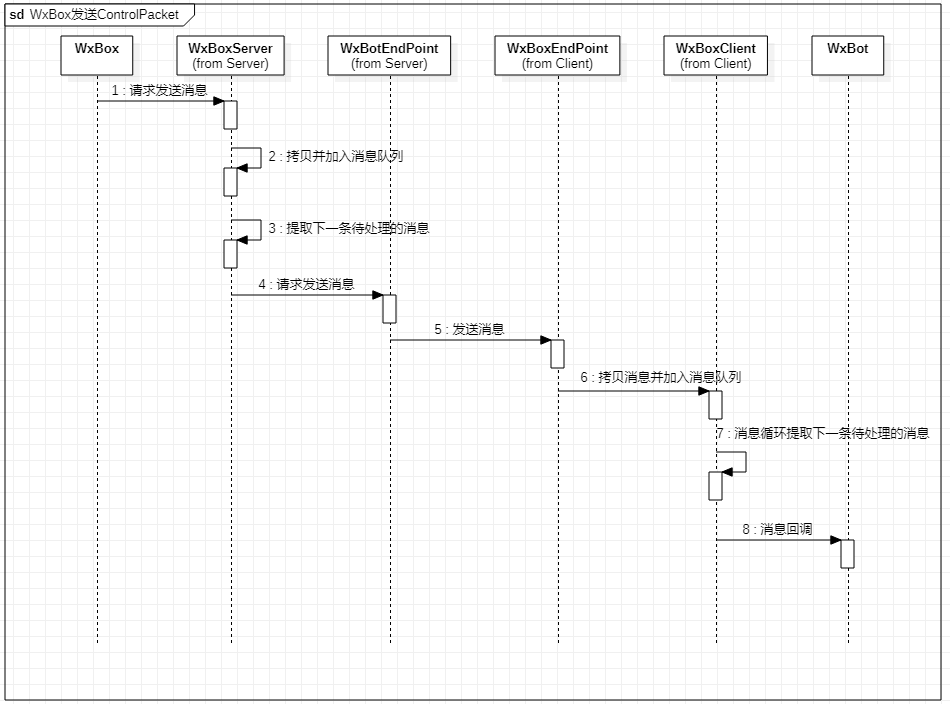
【WxBoxServer与WxBoxClient握手交互图】

##### Client发送ControlPacket



【WxBot向WxBox发送消息】

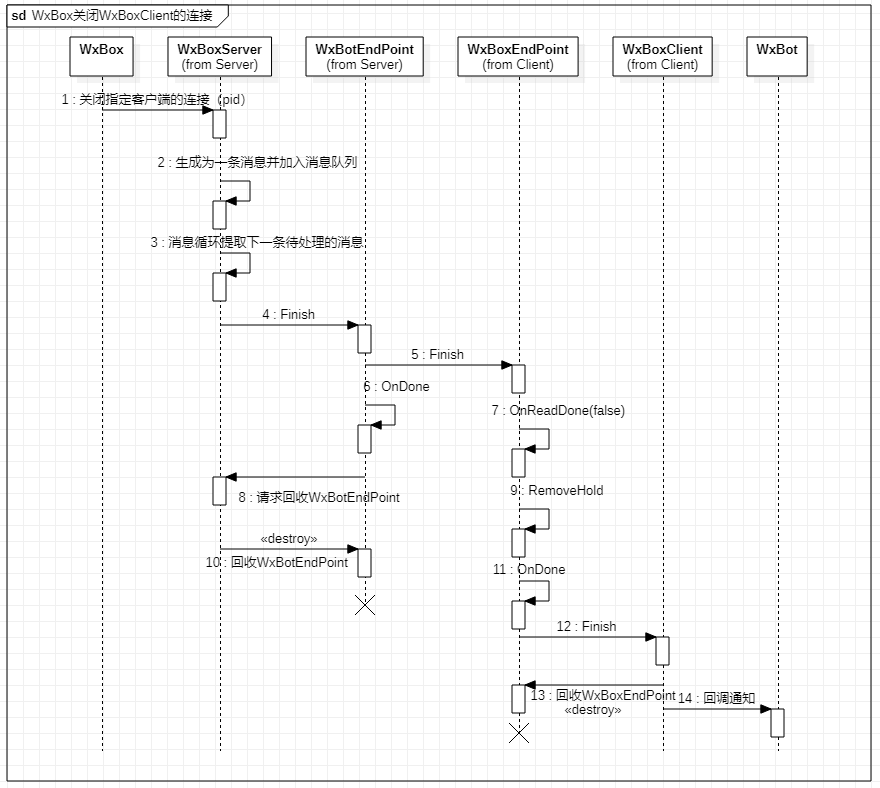
##### Server发送ControlPacket



【WxBox向WxBot发送消息】

##### WxBox关闭WxBoxClient的连接

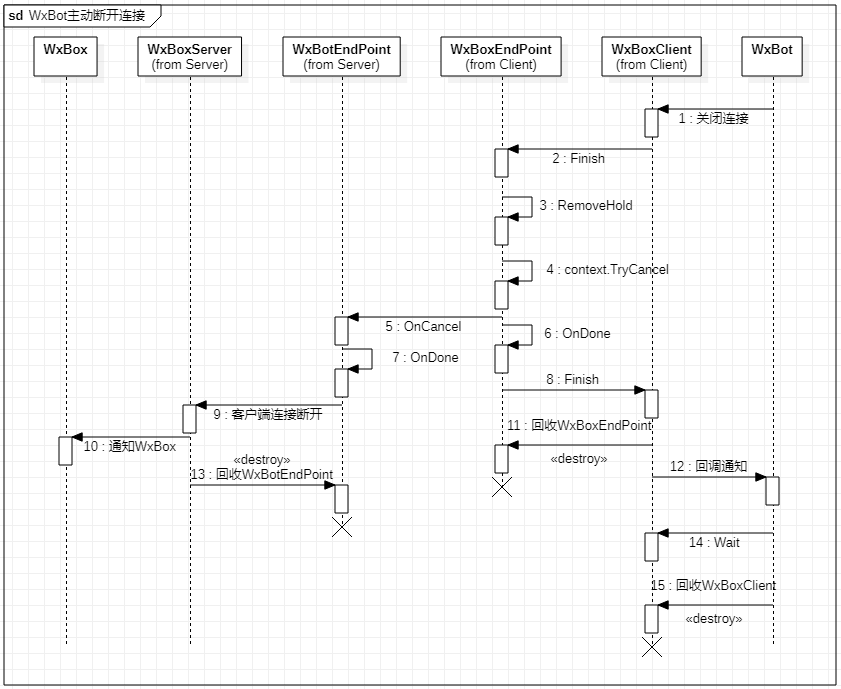
这只会发生在WxBox关闭WxBoxServer服务，请求所有客户端退出连接时发生，并不会单独断开一个客户端的连接，因为这样做没有意义，如果客户端发现Server断开了之后，会定时尝试连接Server。



【WxBox关闭WxBoxClient的连接】

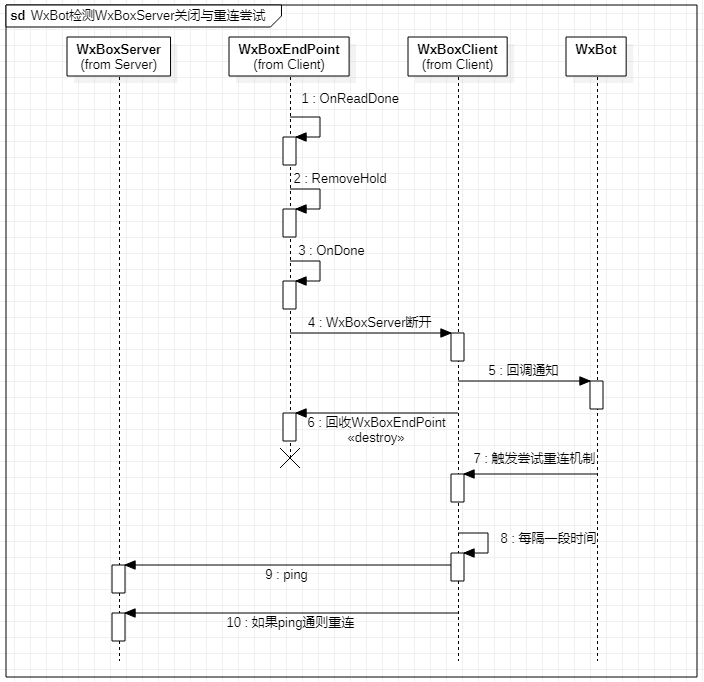
##### WxBot主动断开连接

这只会发生在WxBot模块的宿主进程即将关闭，不然不会主动断开连接的。



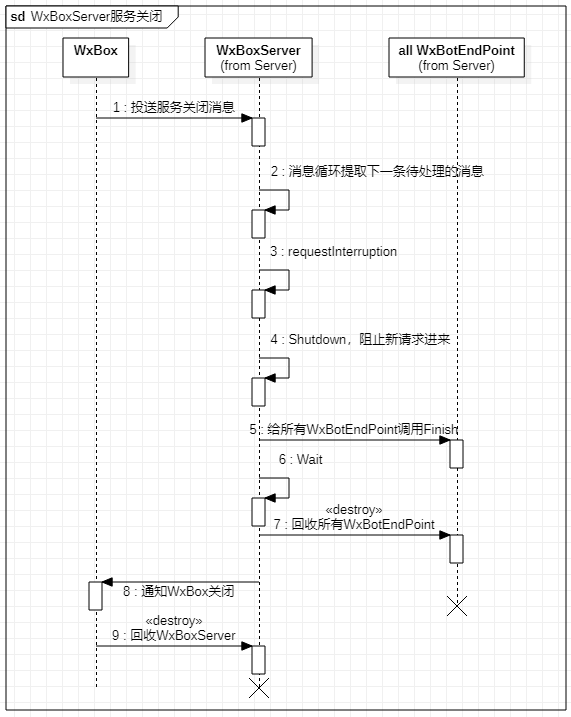
【WxBot主动断开连接】

##### WxBot检测WxBoxServer关闭与重连尝试



【WxBot检测WxBoxServer关闭与重连尝试】

##### WxBoxServer服务关闭



【WxBoxServer服务关闭】

### 模块注入与卸载【WxBox-FR-10】

【deprecated：这部分使用原生的系统API重写，不使用frida进行注入】

frida-core支持注入模块，但是有一个问题，就是它是在注入成功并执行完目标方法后就卸载模块了，跟需求不太符合，所以这里这一块还是直接重新写。这里附上使用frida-core注入的例子：

#include <frida-core-devkit-15.1.14-windows-x86/frida-core.h>

TEST(hook, DISABLED\_use\_frida)

{

    frida\_init();

    FridaInjector\* injector = frida\_injector\_new();

    GError\* error = nullptr;

    auto    e     = frida\_injector\_inject\_library\_file\_sync(injector, 628036, "forhook.dll", "SayHi", "", nullptr, &error);

    frida\_injector\_close\_sync(injector, nullptr, nullptr);

    frida\_unref(injector);

    frida\_deinit();

}

#### Windows

方法很简单，这里简述下注入方法：

1. 加载dll模块使用LoadLibraryA，通过GetProcAddress在kernel32模块获取LoadLibraryA函数的入口地址，在本进程获取到的这个函数的地址对于其它进程是一样的
2. 在目标进程分配两个页，一个页放自定义数据，一个放后续需要注入的代码
3. 使用CreateRemoteThread以LoadLibraryA为入口地址在目标进程创建一个远程线程，这个函数调用会返回一个远程线程的句柄
4. 使用CreateRemoteThread返回的句柄调用WaitForSingleObject等待注入完毕
5. 把自定义的一个naked函数用类似的方式注入的目标进程中，用新远程线程调用自定义方法

卸载方法：同样是创建远程线程，不过改为FreeLibrary用来卸载模块。

#### Mac OS

### 强制重置PluginVirtualMachine【WxBox-FR-11】

由于监控文件更新可能会漏掉，所以WxBox可以向所有WxBoxClient发出指令强制重置PluginVirtualMachine。

# WxBot模块

## 非功能性需求

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 功能名称 | 简要描述 | 依赖 | 优先级 |
| WxBot-NFR-1 |  |  |  |  |

## 功能性需求

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 功能名称 | 简要描述 | 依赖 | 优先级 |
| WxBot-FR-1 | [Plugin](#_Plugin【WxBot-FR-1】) | 实现lua扩展 |  | 必须 |
| WxBot-FR-2 | [Plugin命令语法测试工具](#_Plugin命令语法测试工具【WxBot-FR-2】) | 提供一个交互式Plugin命令执行的语法测试工具 |  | 可选 |

### Plugin【WxBot-FR-1】

使用C构建的lua来实现WxBox的Plugin，要注意的是，项目中的lua是C语言的构建版本而不是C++，所以“让出”、“错误处理”等操作用的是setjmp、longjmp来完成长跳转的，这会破坏栈帧结构，像在自定义封装的lua C函数内使用了C++的局部对象，那么是否能够调用对象的析构函数这是无法保证的，默认情况下，在Visual Studio编译环境由于编译器有补充了一些处理，所以可以调用析构函数，但是在Linux环境就不行。所以在编写lua扩展的部分使用C++类对象或者其它C++特性的话一定要谨慎。

#### Plugin提供的主要功能

* + 执行命令（只能通过filehelper微信联系人或者WxBox窗口界面上执行命令，另外对于模块调用，只能执行Plugin模块或者WxBox内部模块的方法，不过全局函数可以随意调用）
  + hook接收的消息
  + hook发送出去的消息

#### Plugin机制

WxBox的Plugin运行环境，我们把它称为PluginVirtualMachine，它维护单个lua虚拟机跑单线程任务处理，所有脚本、任务都经由它处理，PluginVirtualMachine存在一个任务队列，当队列没有消息时所在线程将休眠，采用的也是生产者-消费者模式，一次只处理一个任务，所有任务都在lua的保护模式下运行，当异常发生时能更好的让出，另外为了避免任务中有死循环或者存在耗时十分长的任务，对每个任务都有超时处理，如果任务运行时间超过WXBOX\_LUA \_PLUGIN \_TASK\_TIMEOUT那么直接结束掉该任务。

#### Plugin类型

* + .lua纯文本脚本
  + 二进制lua脚本：Windows下以dll形式发布，Mac以so形式发布

每个.lua文件代表一个模块，不存在同名的模块。纯文本和二进制的它们都放在同一个目录下（plugins根目录），二进制的Plugin主要以辅助为主，尽量用纯文本。WxBox只会自动加载.lua纯文本的脚本，二进制的扩展按需要在.lua中加载。

#### Plugin编写规则

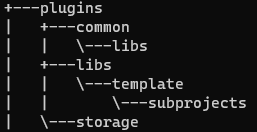
* 扩展的文件名要与.lua内注册的模块名要相同（包括大小写）；
* 扩展内所有的全局对象（包括函数、变量等）都要包含在扩展的“名字空间”内；
* 别在全局范围内执行任何语句。

尽量按照以上的规则编写.lua脚本。

#### Storage

lua可以自由的在storage文件夹内存储配置文件和数据文件，尽量每个模块对应一个文件夹。

#### Plugins目录结构



【plugins目录结构】

* +---plugins：存放所有包括纯文本和二进制的扩展；
* | +---common：存放“第三方”的lua库，比如：yaml库；
* | | \---libs：存放“第三方”的动态lua库
* | +---libs：存放二进制的lua扩展
* | | \---template：一个二进制扩展的模板，要想编写二进制的扩展可以参考这个模板。
* | | \---subprojects
* | \---storage：供所有脚本存储配置信息、数据；

#### Plugin热更新（HotPatching）机制

监控plugins文件夹的变化，每次新变化（只针对plugins根目录下的所有\*.lua文件），更新分为三种情况，一种是加入新的扩展，第二种是脚本被删除，第三种是脚本内容发生变化，。

所有plugin提供load、unload和prereload方法。

* + - 对于新加入的扩展，直接加载进来之后调用扩展的load方法；
    - 对于被删除的扩展会执行扩展的unload方法，然后把扩展导入的模块删除；
    - 对于第三种情况会先执行原本脚本的prereload方法，然后把扩展导入的模块删除，最后把脚本加载进来并执行模块的load方法。

当检测到plugins文件夹内扩展出现变化后不是马上应用更新的，当出现更新时，会先等待一段时间（WXBOX\_LUA\_PLUGIN\_HOT\_PATCHING\_INTERVAL毫秒），在WXBOX\_LUA\_PLUGIN\_HOT\_PATCHING\_INTERVAL时间内如果再次出现更新那么把等待时间刷新。每次的更新信息都会先记录下来，当等待完WXBOX\_LUA\_PLUGIN\_HOT\_PATCHING\_INTERVAL后，会检查所有的更新信息来执行热更新。

由于监控文件更新可能会漏掉，所以WxBox可以向所有WxBoxClient发出指令强制重置PluginVirtualMachine。另外PluginVirtualMachine通过文件监控来确定更新时机，但是将自己收集更新信息，这是为了避免出现遗漏。

在Windows上这里用的是ReadDirectoryChangesW API来监控文件夹内的文件变化，不过这个API不能保证100%接收到所有的更新事件，要获取更准确的更新事件可以考虑“Change Journal”，不够暂时来说ReadDirectoryChangesW是够用的，因为ReadDirectoryChangesW虽然不能保证接收所有的文件更新事件，但是在多个文件更新时，最起码能够接收到一个事件，在接收到消息后，PluginVirtualMachine重新扫描整个plugins目录就可以了。

#### Plugin事件

* load：扩展加载事件，返回true表示初始化成功，返回false表示初始化失败；
* prereload：扩展重载前事件，没有返回值；
* unload：扩展卸载事件，没有返回值。
* receive\_text\_message：微信接收文本消息，接收一个PluginEventModel，没有返回值
* send\_text\_message：微信发送文本消息，接收一个PluginEventModel，没有返回值

note:

* 不能通过调用语句来调用Plugin的事件方法。
* load、prereload和unload事件处理不接收参数，其它参数都会传递一个PluginEventModel

#### Plugin任务超时机制

在所有需要超时机制的lua调用前，先启动一个CheckTimeOut任务，然后再执行lua调用。再CheckTimeOut中，会检查是否超时，如果超时就通过lua\_sethook（mask使用LUA\_MASKLINE，即执行一条语句前捕获）设置lua钩子函数。在钩子函数中抛出错误，那么lua调用就会让出并返回。

#### Plugin调用语法（BNF）

statement ::= prefixexp methodcall

prefixexp ::= “**>>**”

methodcall ::= method | method “**:**” arglist

method ::= methodname | modulename “**.**” methodname

arglist ::= expr {“**,**” expr}

expr ::= **false** | **true** | Numeral | LiteralString ~~| expr op expr~~

~~op ::= “~~**~~+~~**~~” | “~~**~~-~~**~~“ | “~~**~~\*~~**~~” | “~~**~~/~~**~~”~~

note:

* 后面划线的部分先不实现
* 现在字符串字面量不支持使用转义字符，字符串可以用”或者’来作为界符
* Numberal只支持10进制的正整数

例子：

带参数

>>exec:”文本”,1234

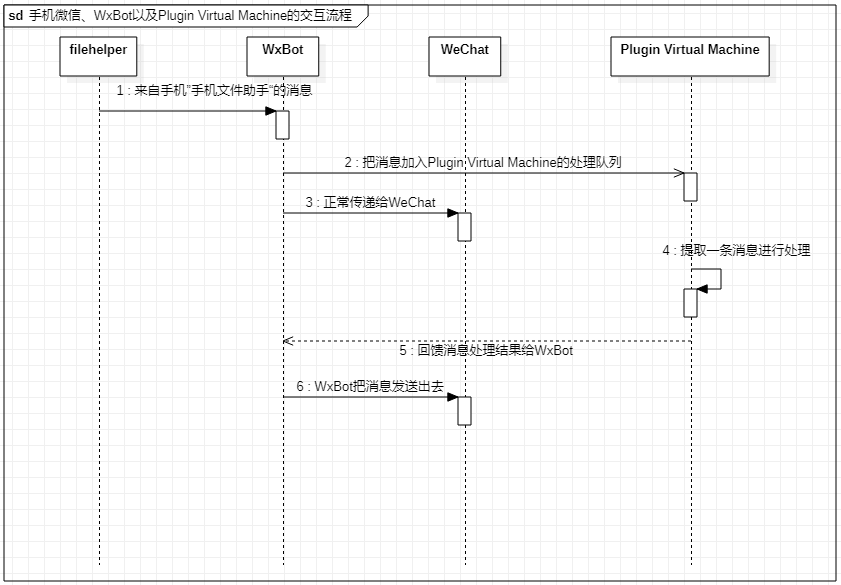
不带参数

>>shutdown

特定模块的方法，退出登陆，不带参数

>>wx.logout

#### Plugin交互图



【手机微信、WxBot以及Plugin Virtual Machine的交互流程】

### Plugin命令语法测试工具【WxBot-FR-2】

提供一个交互式的Plugin命令执行工具，用于测试命令调用的语法