Cosmos大核SDK接入文档（宿主程序）

1. 文档概述

本文档提供Cosmos大核SDK的接入指南，帮助开发者快速集成SDK，实现与Cosmos系统的通信和功能调用。

2. 环境要求

2.1 支持平台

- **Windows**（`_WIN32`，`_WINDLL`）
- **Linux**（`__linux__`，`__GNUC__ >= 4`）

2.2 跨平台支持

CosmosHost提供了完整的跨平台支持，包括：

- 动态库加载封装（Windows: `LoadLibraryExA` / Linux: `dlopen`）
- 路径处理工具（自动处理路径分隔符，使用 `platform::path_join`）
- 字符编码转换（Windows: GBK↔UTF-8 / Linux: 直接透传，使用 `platform::gbk_to_utf8` 和 `platform::utf8_to_gbk`）
- 跨平台定时器实现（使用 `platform::PeriodicTimer`）
- 可执行文件路径获取（使用 `platform::executable_dir`）

2.3 依赖项

- C/C++编译器（MSVC/GCC/Clang）
- CMake（推荐）

3. SDK集成

3.1 下载SDK

sdk主要包含以下文件：

- `Cosmos.Product.Sdk.h`（核心头文件）
- `CosmosSDK.dll`（Windows动态库）
- `libCosmosSDK.so`（Linux动态库）
- `osslsigncode.dll`（`CosmosSDK.dll`依赖库，仅Windows）

3.2 配置项目

CMake配置示例（按照提供的demo配置）

```
cmake_minimum_required(VERSION 3.10)

project(CosmosHostDemo)
```

```
# 设置SDK路径
set(COSMOS_SDK_PATH "./")

# 包含头文件
include_directories(${COSMOS_SDK_PATH}/include)
# 包含rapidjson头文件
include_directories(${COSMOS_SDK_PATH}/include/rapidjson)

set(SRC_FILE
    "CCosmosHostApi.h"
    "main.cpp"
    "CCosmosHostApi.cpp"
    "platform.h"
    "platform.cpp"
)

add_executable(CosmosHostDemo
    ${SRC_FILE}
)

set_target_properties(CosmosHostDemo PROPERTIES
    CXX_STANDARD 17
    CXX_STANDARD_REQUIRED YES
)

if (WIN32)
    target_compile_definitions(CosmosHostDemo PRIVATE _CRT_SECURE_NO_WARNINGS)
else()
    find_package(Threads REQUIRED)
    target_link_libraries(CosmosHostDemo PRIVATE Threads::Threads)
    if (UNIX)
        # dlopen/dlsym
        target_link_libraries(CosmosHostDemo PRIVATE dl)
    endif()
endif()
```

4. 核心API使用

4.1 初始化环境

```
#include "platform.h"  // 跨平台支持头文件

// 获取SDK库名称（跨平台）
static std::string GetSdkLibraryName() {
#if defined(_WIN32)
    return "CosmosSDK.dll";
#else
    return "libCosmosSDK.so";
#endif
}

// 获取主程序名称（跨平台）
static std::string GetMainAppName() {
#if defined(_WIN32)
```

```
    return "Cosmos.MainApp.exe";
#else
    return "Cosmos.MainApp";
#endif
}

// 在构造函数中初始化
CCosmosApi::CCosmosApi()
{
    // 获取可执行文件所在目录（跨平台）
    std::string strPath = platform::executable_dir();
    std::string strSdkPath = platform::path_join(platform::path_join(strPath,
"Cosmos"), GetSdkLibraryName());

    // 加载并校验宿主sdk函数指针（跨平台）
    platform::DynamicLibrary sdkModule;
    if (!sdkModule.load(strSdkPath))
    {
        return;
    }

    /// 获取cosmos引擎提供方法
    auto Call_Cosmos_InitializeEnvironment =
        reinterpret_cast<Cosmos_InitializeEnvironmentDelegate>
(sdkModule.symbol("Cosmos_InitializeEnvironment")); //初始化环境变量函数
    m_pRelease =
        reinterpret_cast<Cosmos_UninitializeEnvironmentDelegate>
(sdkModule.symbol("Cosmos_UninitializeEnvironment")); //去初始化环境变量，宿主关闭时使
用

    /// 保存cosmos引擎提供的回调函数，集体代表含义，参考成员变量声明
    m_pNotify = reinterpret_cast<Cosmos_NotifyDelegate>
(sdkModule.symbol("Cosmos_Notify"));
    m_pNotifyRelease = reinterpret_cast<Cosmos_ReleaseResultDelegate>
(sdkModule.symbol("Cosmos_ReleaseResult"));
    m_pInvoke = reinterpret_cast<Cosmos_InvokeDelegate>
(sdkModule.symbol("Cosmos_Invoke"));
    m_pInvokeRelease = reinterpret_cast<Cosmos_ReleaseInvokeResponseDelegate>
(sdkModule.symbol("Cosmos_ReleaseInvokeResponse"));
    m_pSubscribe = reinterpret_cast<Cosmos_SubscribeDelegate>
(sdkModule.symbol("Cosmos_Subscribe"));
    m_pSubscribeRelease =
reinterpret_cast<Cosmos_ReleaseSubscribeResponseDelegate>
(sdkModule.symbol("Cosmos_ReleaseSubscribeResponse"));
    m_pPush = reinterpret_cast<Cosmos_PushSubscriptionDataDelegate>
(sdkModule.symbol("Cosmos_PushSubscriptionData"));
    m_pUnsubscribe = reinterpret_cast<Cosmos_UnsubscribeDelegate>
(sdkModule.symbol("Cosmos_Unsubscribe"));

    //初始化客户端参数
    auto clientParameters = new Cosmos_ClientParameters;
    memset(clientParameters, 0, sizeof(Cosmos_ClientParameters));
    static std::string mainAppPath = platform::path_join("./Cosmos",
GetMainAppName());
    clientParameters->CosmosMainAppPath = (char*)(mainAppPath.c_str());     //设置
Cosmos引擎所在位置(可以填绝对位置)
```

```
    //设置行情账户信息和宿主账户信息
    rapidjson::Document docProduct, docMarketAccount;
    std::string dataProduct, dataMarketAccount;
    //设置宿主账户信息
    {
        docProduct.SetObject();
        docProduct.AddMember(rapidjson::StringRef("Account"), "test",
docProduct.GetAllocator());                                 //当前宿主的账号
        docProduct.AddMember(rapidjson::StringRef("Token"), "123",
docProduct.GetAllocator());                                   //本次登录的
        docProduct.AddMember(rapidjson::StringRef("Password"), "123123",
docProduct.GetAllocator());                               //密码
        docProduct.AddMember(rapidjson::StringRef("ProductID"), "test",
docProduct.GetAllocator());                                //当前宿主的产品id

        //下面三个配置建议设置成配置项读取，部署在不同环境的信息会发生变化
        docProduct.AddMember(rapidjson::StringRef("SpiderUrl"),
"https://unitetest.chinastock.com.cn:8081", docProduct.GetAllocator());      //应
用市场采集用户行为地址
        docProduct.AddMember(rapidjson::StringRef("Ip"), "10.4.124.34",
docProduct.GetAllocator());                               //消息中心的ip
        docProduct.AddMember(rapidjson::StringRef("Port"), 9999,
docProduct.GetAllocator());                                   //消息中心的端口

        rapidjson::StringBuffer strBuf;
        rapidjson::Writer<rapidjson::StringBuffer> writer(strBuf);
        docProduct.Accept(writer);
        dataProduct = platform::gbk_to_utf8(strBuf.GetString());
        dataProduct = base64_encode(dataProduct);
    }

    //设置行情账户信息,现在账号不进行校验，可以随便填入
    {
        docMarketAccount.SetObject();
        docMarketAccount.AddMember(rapidjson::StringRef("Account"), "test",
docProduct.GetAllocator());                                 //行情账号
        docMarketAccount.AddMember(rapidjson::StringRef("Md5"), "123",
docProduct.GetAllocator());                                     //行情账号密码的md5值

        rapidjson::StringBuffer strBuf;
        rapidjson::Writer<rapidjson::StringBuffer> writer(strBuf);
        docMarketAccount.Accept(writer);
        dataMarketAccount = platform::gbk_to_utf8(strBuf.GetString());
        dataMarketAccount = base64_encode(dataMarketAccount);
    }

    std::string strId = "HostDemo-```"+ dataMarketAccount + "```" + dataProduct +
"```";
    clientParameters->Id = strId.c_str();
//产品id，改字段由三部分组成宿主的产品id + 行情账户信息 + 宿主账户信息，参数之间使用"```"做分
隔
    clientParameters->Version = "0.0.0.1";
 //设置客户端版本号
    clientParameters->HighDpiMode = Comos_GuiHighDpiMode::SystemAware;
 //高清屏配置
```

```
    //开发者参数
    auto developerParameters = new Cosmos_DeveloperParameter;
    memset(developerParameters, 0, sizeof(Cosmos_DeveloperParameter));
    developerParameters->AppProviderMode =
"nuget;https://unitetest.chinastock.com.cn:453/v3/index.json";  //设置组件来源,当为
nuget时，分号后面的路径为应用市场访问地址
    developerParameters->RuntimeMode = "debug";                                    //
运行模式（Cosmos引擎内部使用，开发时选择debug、发行版本选择release）
    developerParameters->GuiMode = "show";
 //应用商店是否显示（改应用商店非用户使用的应用商店，开发时可选择show、发行版本选择hide）

    //cef参数参数设置，如果存在cef的组件，需要设置该配置，不然无法运行cef组件
    auto webViewParameters = new Cosmos_WebViewParameters;
    memset(webViewParameters, 0, sizeof(Cosmos_WebViewParameters));
    webViewParameters->CefDirectory = "C:/Users/ThsQstudio";
              //cef动态库路径
    webViewParameters->CefResourcesDirectory = "C:/Users/ThsQstudio/Resources";
             //cef资源路径
    webViewParameters->CefLocaleDirectory =
"C:/Users/ThsQstudio/Resources/locales";           //cef字体包路径

    //向cosmos引擎提供宿主回调函数
    auto responsibility = new Cosmos_Responsibility;
    responsibility->Cosmos_NotifyHandler = Cosmos_Notify_Callback;
//向Cosmos引擎注册notify函数
    responsibility->Cosmos_ReleaseResultHandler = Cosmos_ReleaseResult_Callback;
    responsibility->Cosmos_InvokeHandler = Cosmos_Invoke_Callback;
//向Cosmos引擎注册invoke函数
    responsibility->Cosmos_ReleaseInvokeResponseHandler =
Cosmos_ReleaseInvokeResponse_Callback;
    responsibility->Cosmos_SubscribeHandler = Cosmos_Subscribe_Callback;
//向Cosmos引擎注册订阅函数
    responsibility->Cosmos_UnsubscribeHandler = Cosmos_UnSubscribe_Callback;
//向Cosmos引擎注册取消订阅函数
    responsibility->Cosmos_PushSubscriptionDataHandler =
Cosmos_PushSubscriptionData_Callback;
    responsibility->Cosmos_ReleaseSubscribeResponseHandler =
Cosmos_ReleaseSubscribeResponse_Callback;

    //设置cosmos引擎的环境参数
    m_pEnvironment = new Cosmos_EnvironmentCreationParameters;
    m_pEnvironment->Responsibility = responsibility;
    m_pEnvironment->ClientParameters = clientParameters;
    m_pEnvironment->DeveloperParameter = developerParameters;
    m_pEnvironment->WebViewParameters = webViewParameters;

    //调用Cosmos引擎初始化函数，启动Cosmos引擎
    auto pRes = Call_Cosmos_InitializeEnvironment(nullptr, m_pEnvironment,
nullptr);
    if (pRes->Code == 200)
    {
        printf("启动成功\n");

        //启动成功，开启定时器，模拟订阅推送（跨平台后台线程）
```

```
        m_timer = std::make_unique<platform::PeriodicTimer>
(std::chrono::milliseconds(100), []() {
            CCosmosApi::GetInstance()->SimulatePush();
        });
    }
    else
    {
        printf("启动失败：code：%d, msg:%s\n", pRes->Code, pRes->Message);
    }
}
```

4.3 发送请求

```
std::string CCosmosApi::Invoke(const std::string& strMethod, const std::string&
strRequest)
{
    std::string strReuslt ="";
    if (m_pInvoke)
    {
        std::string strUtf8Method = platform::gbk_to_utf8(strMethod);
        std::string strUtfRequest = platform::gbk_to_utf8(strRequest);
        Cosmos_InvokeRequest* pRequest = new Cosmos_InvokeRequest;

        pRequest->Method = strUtf8Method.c_str();
        pRequest->Parameters = strUtfRequest.c_str();
        auto pRes = m_pInvoke(nullptr, pRequest);

        if (pRes)
        {
            rapidjson::Document doc;
            doc.SetObject();
            doc.AddMember(rapidjson::StringRef("Code"), pRes->Result->Code,
doc.GetAllocator());
            doc.AddMember(rapidjson::StringRef("Data"),
rapidjson::StringRef(pRes->DataFrame->Data), doc.GetAllocator());

            rapidjson::StringBuffer strBuf;
            rapidjson::Writer<rapidjson::StringBuffer> writer(strBuf);
            doc.Accept(writer);
            strReuslt = platform::utf8_to_gbk(strBuf.GetString());

            //释放对方应答数据
            m_pInvokeRelease(nullptr, pRes);
        }
    }
    return strReuslt;
}
```

4.2 发起通知

```
void CCosmosApi::Notify(const std::string& strTopic, const std::string&
strNotify)
{
    std::string strUtf8Topic = platform::gbk_to_utf8(strTopic);
```

```
    std::string strUtfNotify = platform::gbk_to_utf8(strNotify);
    Cosmos_NotifyRequest *pNotify = new Cosmos_NotifyRequest;

    pNotify->Topic = strUtf8Topic.c_str();
    pNotify->Message = strUtfNotify.c_str();
    pNotify->RoutingKey = "";

    auto pRes = m_pNotify(nullptr, pNotify);
    if (pRes)
    {
        //释放内存
        m_pNotifyRelease(nullptr, pRes);
    }
}
```

4.4 订阅数据

```
//该方法组件引擎暂未实现
std::string CCosmosApi::SubScribe(const std::string& strTopic, const std::string&
strSubscribe)
{
    std::string strUtf8Topic = platform::gbk_to_utf8(strTopic);
    std::string strUtfSubscribe = platform::gbk_to_utf8(strSubscribe);
    Cosmos_SubscribeRequest* pSub = new Cosmos_SubscribeRequest;
    pSub->Topic = strUtf8Topic.c_str();
    pSub->Parameters = strUtfSubscribe.c_str();
    std::string strUUid = "";

    auto pRes = m_pSubscribe(nullptr, pSub);
    if (pRes)
    {
        //订阅成功
        if (pRes->Result->Code == 200)
        {
            strUUid = pRes->Subscription->SubscriptionId;
        }
        //释放对方应答
        m_pSubscribeRelease(nullptr, pRes);
    }
    return strUUid;
}
```

4.5 清理资源

```
// 在析构函数中自动清理资源
CCosmosApi::~CCosmosApi()
{
    if (m_timer) m_timer->stop();
    SAFE_DELETE(m_pEnvironment->Responsibility);
    SAFE_DELETE(m_pEnvironment->ClientParameters);
    SAFE_DELETE(m_pEnvironment->DeveloperParameter);
    SAFE_DELETE(m_pEnvironment->WebViewParameters);
```

```
    SAFE_DELETE(m_pEnvironment);
    if (m_pRelease)
    {
        m_pRelease();
    }
}

// 或者通过Close方法手动清理
void CCosmosApi::Close()
{
    SAFE_DELETE(m_instance);
}
```

5 Cosmos提供业务接口

5.1 invoke方法（同步调用）

基础协议

```
{
    "Action"         : "Invoke",      // 调用方式。invoke：同步调用 notify：异步调用，具体填入查看提供的方法
    "ActionInstance" : "11111111",    // 调用者对象实例
    "ActionContext"  :                // 调用参数
    {
        "Invoker"    : "00000000",    // 被调用对象，0代表小核引擎，其他为组件实例
        "Function"   : "xxxx",        // 方法名
        "Parameters" :                // 请求参数
        {
        }
    }
```

5.1.1 创建组件（方法名：CreateWidget）

请求协议如下:

```
{
    "Action"         : "Invoke",      // 行为：调用
    "ActionInstance" : "11111111",     // 调用者生成，用于Trace
    "ActionContext"  :
    {
        "Invoker"    : "00000000",    // 0代表小核引擎
        "Function"   : "CreateWidget", // CreateWidget代表构造Widget
        "Parameters" :
        {
            "AppGuid" : "appguid",  // appguid
            "WidgetGuid" : "{9C42AA03-9066-4648-AAF2-B46B70775AB4}",  // GUID是指Widget的唯一编号，提交时生成。目前版本由开发者写在元数据里。
            "WidgetVersion" : "1.0.0",  // Widget版本
            "WidgetPreference" :
            {
                "ParentHandle" :      "12345678", // 父窗口句柄,十进制数字字符串
                "TitleBarVisibility" : "Visible",  // 标题栏可见性 (Visible, Collapsed, Hidden)
```

```
                "WindowVisibility"    : "Visible",    // 窗口可见性 (Visible,
Collapsed, Hidden)
                "ResizeMode"          : "NoResize",  // 大小变更模式 "NoResize",
"CanResize"
                "WidgetWidth": 920,       // 弹窗宽度
                "WidgetHeight" :680,    // 弹窗高度
                "BorderThickness"     : "1",          // 边框宽度(100% DPI下的像素值,
随DPI自增)
                "ModalModeParentHandle" : "87654321", // 模态模式父窗口句柄,十进制数字
字符串，当指定该字符串时，Widget以模态方式呈现
                "CalibrationDpi" : 96, // 父窗口初始DPI值，常见值：96、144、192 ,
                "WindowFontWeight": 400, // 字体粗细:
https://learn.microsoft.com/en-us/dotnet/api/system.windows.fontweights?
view=windowsdesktop-7.0#remarks
                "WindowTop":  100.0, // 窗口顶部位置，double类型，逻辑单位随DPI自增;
WindowTop 和 WindowLeft 均不传时，默认居中显示
                "WindowLeft":  100.0 // 窗口左边位置，double类型，逻辑单位随DPI自增
            }
        }
    }
}
```

应答如下:

```
{
    "Action"         : "InvokeReturn",
    "ActionInstance" : "11111111",         // 将调用时的Instance回传
    "ActionContext"  :
    {
        "Return"     :
        {
            "WidgetHandle" : "66666666"   // Widget对象句柄（注意不是HWND）
            "WindowHandle" : "77777777"   // HWND 窗体句柄
        }
    }
}
```

示例:

5.1.2 创建页面（方法名：CreatePage）

通过读取一个xml文件路径返回一个布局完整的页面

请求协议如下:

```
{
    "Action"         : "Invoke",       // 行为：调用
    "ActionInstance" : "11111111",      // 调用者生成，用于Trace
    "ActionContext"  :
    {
        "Invoker"    : "00000000",     // 0代表小核引擎
        "Function"   : "CreatePage", // CreateWidget代表构造Widget
        "Parameters" :
        {
            "XmlPath" : "xxx",  //填入xml文件路径（绝对路径）
```

```
            "ParentHandle" :      "12345678", // 父窗口句柄,十进制数字字符串
            "TitleBarVisibility" : "Visible",  // 标题栏可见性 (Visible, Collapsed,
Hidden)
            "WindowVisibility"   : "Visible",   // 窗口可见性 (Visible, Collapsed,
Hidden)
            "ResizeMode"         : "NoResize",  // 大小变更模式 "NoResize",
"CanResize"
            "WidgetWidth": 920,      // 弹窗宽度
            "WidgetHeight" :680,   // 弹窗高度
            "BorderThickness"    : "1",          // 边框宽度(100% DPI下的像素值，随DPI
自增）
            "ModalModeParentHandle" : "87654321", // 模态模式父窗口句柄,十进制数字字符
串，当指定该字符串时，Widget以模态方式呈现
            "CalibrationDpi" : 96, // 父窗口初始DPI值，常见值：96、144、192 ，
            "WindowFontWeight": 400, // 字体粗细： https://learn.microsoft.com/en-
us/dotnet/api/system.windows.fontweights?view=windowsdesktop-7.0#remarks

            "WindowTop":  100.0, // 窗口顶部位置，double类型，逻辑单位随DPI自增；
WindowTop 和 WindowLeft 均不传时，默认居中显示
            "WindowLeft":  100.0 // 窗口左边位置，double类型，逻辑单位随DPI自增
        }
    }
}
```

应答如下：

```
{
    "Action"         : "InvokeReturn",
    "ActionInstance" : "11111111",        // 将调用时的Instance回传
    "ActionContext"  :
    {
        "Return"     :
        {
            "WidgetHandle" : "66666666"   // Widget对象句柄（注意不是HWND）
            "WindowHandle" : "77777777"   // HWND 窗体句柄
        }
    }
}
```

示例：

5.1.3 关闭组件（方法名：DestroyWidget）

请求如下：

```
{
    "Action"         : "Invoke",      // 行为：调用
    "ActionInstance" : "11111111",     // 调用者生成，用于Trace
    "ActionContext"  :
    {
        "Invoker"    : "00000000",    // 0代表小核引擎
        "Function"   : "DestroyWidget",
        "Parameters" :
        {
            "WidgetHandle" : "66666666",   // Widget对象句柄（注意不是HWND）
```

```
            "WindowHandle" : "77777777"    // HWND 窗体句柄
        }
    }
}
```

应答如下：

```
{
    "Action"         : "InvokeReturn",
    "ActionInstance" : "11111111",    // 将调用时的Instance回传
    "ActionContext"  :
    {
        "Return"     :  null
    }
}
```

5.1.4 关闭小核（方法名：ShutdownCosmos）

请求如下

```
{
    "Action"         : "Invoke",       // 行为：调用
    "ActionInstance" : "11111111",      // 调用者生成，用于Trace
    "ActionContext"  :
    {
        "Invoker"    : "00000000",     // 0代表小核引擎
        "Function"   : "ShutdownCosmos",
        "Parameters" :
        {

        }
    }
}
```

应答如下：

```
{
    "Action"         : "InvokeReturn",
    "ActionInstance" : "22222222",
    "ActionContext"  :
    {
        "Return"     :  null // 无返回内容
    }
}
```

5.1.5 更新主题颜色（方法名：SetUserExperience）

请求如下：

```
{
    "Action"         : "Invoke",       // 行为：调用
    "ActionInstance" : "11111111",      // 调用者生成，用于Trace
    "ActionContext"  :
    {
        "Invoker"    : "00000000",     // 0代表小核引擎
        "Function"   : "SetUserExperience",
        "Parameters" :
        {
             "ColorScheme" : "Dark",    // 主题色：Dark, Light, 可传NotChanged
        }
    }
}
```

应答如下：

```
{
    "Action"         : "InvokeReturn",
    "ActionInstance" : "22222222",
    "ActionContext"  :
    {
        "Return"     :  null // 无返回内容
    }
}
```

5.1.6 获取资源字典（方法名：GetResouceDictionary）

请求示例：

```
{
    "Action"         : "Invoke",
    "ActionInstance" : "TestInstance",
    "ActionContext"  :
    {
        "Invoker"    : "00000000",
        "Function"   : "GetResouceDictionary",
        "Parameters" :
        {
            "Color" : "Dark"  // 可选参数，指定颜色主题（Dark/Light）
        }
    }
}
```

出参：Dictionary<string, object> dictionary

```
{
    "Action"         : "InvokeReturn",
    "ActionInstance" : "TestInstance",
    "ActionContext"  :
    {
        "Return" :
        {
            "Color":
```

```
            {
                "Dark":{
                "color-green5":"#FF357A4F",
                ...
                },
                "Light":{
                    "color-green5":"#FF9BC7AA",
                    ...
                }
            },
            "FontFamily":{
                "Font-Common":"Microsoft Yahei"
            },
            "FontSize":{
                "FontSize-Large":16.0,
                "FontSize-Normal":12.0,
                "FontSize-Medium":14.0
            }
        }
    }
}
```

5.1.7 添加快捷键（方法名：AddShortcut）

虚拟按键映射参考：虚拟按键映射表

方法：public async Task<Result> FromProductCall_AddShortcut(ActionInvoke actionInvoke)
映射：AddShortcut
入参：
ShortcutId：　快捷键唯一ID
Description：快捷键功能；
ShortCut:{　　快捷键按键映射结构体
　Modifiers, 辅助按键映射
　Key　　虚拟按键映射
}
请求示例：

```
{
    "Action"         : "Invoke",
    "ActionInstance" : "TestInstance",
    "ActionContext"  :
    {
        "Invoker"    : "00000000",
        "Function"   : "AddShortcut",
        "Parameters" :[
            {
                "ShortcutId": "1",
                "Description": "截图",
                "ShortCut":{
                     // windows辅助键
                     // MOD_ALT      0x0001
                     // MOD_CONTROL  0x0002
                     // MOD_NOREPEAT 0x4000
                     // MOD_SHIFT    0x0004
```

```
                    // MOD_WIN      0x0008
                    "Modifiers":1,
                    "Key":65
                }
            },
            {
                "ShortcutId": "2",
                "Description": "锁定",
                "ShortCut":{
                    // windows辅助键
                    // MOD_ALT      0x0001
                    // MOD_CONTROL  0x0002
                    // MOD_NOREPEAT 0x4000
                    // MOD_SHIFT    0x0004
                    // MOD_WIN      0x0008
                    "Modifiers":2,
                    "Key":76
                }
            }
        ]
    }
}
```

出参:

```
{
    "Action"         : "InvokeReturn",
    "ActionInstance" : "TestInstance",
    "ActionContext"  :
    {
        "Return" : [
            {
                "ShortcutId": "1",
                "Description": "截图",
                "ShortCut":{
                    "Modifiers":1,
                    "Key":65
                },
                "IsSuccessful":true,
                "ErrorMessage":null
            },
            {
                "ShortcutId": "2",
                "Description": "锁定",
                "ShortCut":{
                    "Modifiers":2,
                    "Key":76
                },
                "IsSuccessful":false,
                "ErrorMessage":"Is already been registered"
            }
        ]
    }
}
```

5.1.8 删除快捷键（方法名：DelShortcut）

方法：public async Task<Result> FromProductCall_DelShortcut(ActionInvoke actionInvoke)
映射：DelShortcut
请求示例：

```
{
    "Action"         : "Invoke",
    "ActionInstance" : "TestInstance",
    "ActionContext"  :
    {
        "Invoker"    : "00000000",
        "Function"   : "DelShortcut",
        "Parameters" :[
            {
                "ShortcutId": "1",
                "Description": "截图",
                "ShortCut":{
                    "Modifiers":1,
                    "Key":65
                }
            },
            {
                "ShortcutId": "2",
                "Description": "锁定",
                "ShortCut":{
                    "Modifiers":2,
                    "Key":76
                }
            }
        ]
    }
}
```

出参：

```
{
    "Action"         : "InvokeReturn",
    "ActionInstance" : "TestInstance",
    "ActionContext"  :
    {
        "Return" : [
            {
                "ShortcutId": "1",
                "Description": "截图",
                "ShortCut":{
                    "Modifiers":1,
                    "Key":65
                },
                "IsSuccessful":true,
                "ErrorMessage":null
            },
            {
                "ShortcutId": "2",
                "Description": "锁定",
```

```
                "ShortCut":{
                    "Modifiers":2,
                    "Key":76
                },
                "IsSuccessful":true,
                "ErrorMessage":null
            }
        ]
    }
}
```

5.1.9 获取快捷键列表（方法名：GetShortcut）

方法：public async Task<Result> FromProductCall_GetShortcut(ActionInvoke actionInvoke)
映射：GetShortcut
请求示例：

```
{
    "Action"         : "Invoke",
    "ActionInstance" : "TestInstance",
    "ActionContext"  :
    {
        "Invoker"    : "00000000",
        "Function"   : "GetShortcut",
        "Parameters" : {}
    }
}
```

出参：List ListShortCut

```
{
    "Action"         : "InvokeReturn",
    "ActionInstance" : "TestInstance",
    "ActionContext"  :
    {
        "Return" :{
            "Result":{
                "1":{
                    "ShortCutId":"1",
                    "ShortCut":{
                        "Modifiers":2,
                        "Key":76
                    },
                    "Description":"",
                },
                "2":{
                    ...
                }
            }
        }
    }
}
```

5.2 Notify方法（异步调用）

基础协议

```
{
    "Action"         : "Notify",      // 调用方式。invoke：同步调用 notify：异步调用，具体填入查看提供的方法
    "ActionInstance" : "11111111",    // 调用者对象实例
    "ActionContext"  :                // 调用参数
    {
        "Notifier"    : "00000000",   // 被通知对象，0代表小核引擎，其他为组件实例
        "NotifyType"   : "xxxx",      // 通知方法
        "Parameters" :                // 请求参数
        {
        }
    }
```

5.2.1 更新组件属性（方法名：SetWidgetPreference）

请求如下：

```
{
    "Action"         : "Notify",      // 行为：通知
    "ActionInstance" : "11111111",     // 调用者生成，用于Trace
    "ActionContext"  :
    {
        "Notifier"    : "00000000",    // 0代表小核引擎
        "NotifyType"   : "SetWidgetPreference", // 设置组件窗口属性
        "Parameters" :
        {
              "WidgetHandle" :      "12345678", // 必填；Widget对象句柄

              "WindowFontWeight": 400, // 非必填：字体粗细：https://learn.microsoft.com/en-us/dotnet/api/system.windows.fontweights?view=windowsdesktop-7.0#remarks
              "WindowTop":  100.0, // 非必填;窗口顶部位置;double类型，逻辑单位随DPI自增
              "WindowLeft":  100.0, // 非必填;窗口左边位置，double类型，逻辑单位随DPI自增
              "CalibrationDpi" : 96, // 父窗口初始DPI值，默认:96;常见值：96、144、192,
              "WindowWidth": 920,       //非必填；弹窗宽度
              "WindowHeight" :680,   // 非必填;弹窗高度
              "WindowVisibility"   : "Visible"   // 窗口可见性 (Visible, Collapsed, Hidden)
        }
    }
}
```

5.2.2 批量更新组件属性（方法名：SetRangeWidgetPreference）

请求如下：

```
{
    "Action"         : "Notify",       // 行为：通知
    "ActionInstance" : "11111111",      // 调用者生成，用于Trace
    "ActionContext"  :
    {
        "Notifier"    : "00000000",     // 0代表小核引擎
        "NotifyType"   : "SetRangeWidgetPreference", // 设置插件窗口属性
        "Parameters" :
        [{
              "WidgetHandle" :      "12345678", // 必填；Widget对象句柄

              "WindowFontWeight": 400, // 非必填：字体粗细：
https://learn.microsoft.com/en-us/dotnet/api/system.windows.fontweights?
view=windowsdesktop-7.0#remarks
              "WindowTop":  100.0, // 非必填;窗口顶部位置;double类型，逻辑单位随DPI自增
              "WindowLeft":  100.0, // 非必填;窗口左边位置，double类型，逻辑单位随DPI自
增
              "CalibrationDpi" : 96, // 父窗口初始DPI值，默认:96;常见值：96、144、192
,
              "WindowWidth": 920,       //非必填；弹窗宽度
              "WindowHeight" :680,    // 非必填;弹窗高度
              "WindowVisibility"    : "Visible"    // 窗口可见性（Visible，Collapsed，
Hidden）
        }]
    }
}
```

5.2.3 更新账户信息（方法名：SetAccount）

请求如下：

```
{
    "Action"         : "Notify",       // 行为：通知
    "ActionInstance" : "11111111",      // 调用者生成，用于Trace
    "ActionContext"  :
    {
        "Notifier"    : "00000000",     // 0代表小核引擎
        "NotifyType"   : "SetAccount", // 设置插件窗口属性
        "Parameters" :
        {
              "Token" : "12345678", // 操作员token
              "Account": "22112", // 账号
              "Password": "密码" // 密码
        }
    }
}
```

6. 示例

参考：宿主demo

7. 错误处理

所有API返回`Cosmos_Result`结构，包含：

- `Code`（错误吗，成功为200，其他都为请求错误）
- `Message`（错误信息）

建议在调用API后检查`Code`，并根据错误码进行相应处理。

8. 注意事项

1. **线程安全**：SDK默认非线程安全，多线程环境下需自行加锁。
2. **资源释放**：在析构函数中自动调用`m_pRelease()`（即`Cosmos_UninitializeEnvironment`）避免内存泄漏，或通过`Close()`方法手动清理。
3. **回调函数**：订阅回调应在主线程处理，避免阻塞。
4. **编码转换**：cosmos内部接受和发送出来的数据都为utf8编码。需要使用`platform::gbk_to_utf8`和`platform::utf8_to_gbk`进行编码转换（Windows平台会自动转换，Linux平台直接透传）。
5. **跨平台路径**：使用`platform::path_join`进行路径拼接，避免硬编码路径分隔符。
6. **定时器**：使用`platform::PeriodicTimer`实现跨平台定时器，避免使用平台特定的定时器API。