# API文档

## 前言

miniblink使用了wke的接口。wke的相关介绍可以google一下。

总的来说，miniblink的接口是纯C导出，只要使用wke.h即可加载。无需.lib。

* 另外，请勿跨线程调用所有接口（除非接口有特殊申明）
* 所有接口如果返回的是const utf8\*，const wchar\_t\*类型的字符串，均不需要手动释放
* miniblink暂时只支持windows系统

## API

### unsigned int wkeVersion();

获取目前api版本号

参数：无

### const utf8\* wkeVersionString();

获取版本字符串

参数：无

### void wkeSetWkeDllPath(const wchar\_t\* dllPath)

设置miniblink的全路径+文件名

参数：

dllPath：dll的全路径，注意是全路径

### void wkeGC(wkeWebView webView, long delayMs);

延迟让miniblink垃圾回收

参数：略

### void wkeSetFileSystem(WKE\_FILE\_OPEN pfnOpen, WKE\_FILE\_CLOSE pfnClose, WKE\_FILE\_SIZE pfnSize, WKE\_FILE\_READ pfnRead, WKE\_FILE\_SEEK pfnSeek)

wke遗留的接口，设置文件相关的hook函数。可以作为拦截对文件的操作。现在已被更先进的函数取代

参数：略

### bool wkeIsLoaded(wkeWebView webView);

（被废弃）获取webView是否处于加载状态，还是加载完成。

参数：略

### bool wkeIsLoadingSucceeded(wkeWebView webView)

（被废弃）判断webview是否加载成功

参数：略

### bool wkeIsLoadFailed(wkeWebView webView)

（被废弃）判断webview是否加载失败

参数：略

### bool wkeIsLoadComplete(wkeWebView webView)

（被废弃）判断webview是否加载完成

参数：略

注意：wkeIsLoadComdivlete等效于wkeIsLoadFailed、wkeIsLoadingSucceeded其中任何一个返回true

### bool wkeIsDocumentReady(wkeWebView webView)

DOM文档结构是否加载完成。和上述几个接口不同，上述几个接口是网络相关的判断。

参数：略

### void wkeStopLoading(wkeWebView webView)

停止加载页面

参数：略

### bool wkeReload(wkeWebView webView)

重新加载页面

参数：略

### const utf8\* wkeGetTitle(wkeWebView webView)

获取页面标题

参数：略

### const wchar\_t\* wkeGetTitleW(wkeWebView webView)

获取页面标题的宽字节版本

参数：略

### void wkeResize(wkeWebView webView, int w, int h)

重新设置页面的宽高。如果webView是带窗口模式的，会设置真窗口的宽高。

参数：略

### int wkeGetWidth(wkeWebView webView)

获取页面宽度

参数：略

### int wkeGetHeight(wkeWebView webView)

获取页面高度

参数：略

### int wkeGetContentWidth(wkeWebView webView)

获取网页排版出来的宽度

参数：略

### int wkeGetContentHeight(wkeWebView webView)

获取网页排版出来的高度

参数：略

### void wkeSetDirty(wkeWebView webView, bool dirty)

（被废弃）设置页面为脏状态，在下一帧将强制重绘页面

参数：略

### bool wkeIsDirty(wkeWebView webView)

（被废弃）获取页面脏状态

参数：略

### void wkeAddDirtyArea(wkeWebView webView, int x, int y, int w, int h)

（被废弃）添加页面脏矩形

参数：略

### void wkeLayoutIfNeeded(wkeWebView webView)

（被废弃）强制让页面重新排版

参数：略

### void wkePaint2(wkeWebView webView, void\* bits, int bufWid, int bufHei, int xDst, int yDst, int w, int h, int xSrc, int ySrc, bool bCopyAlpha)

参数：

|  |  |
| --- | --- |
| bits | 外部申请并传递给mb的buffer，大小是bufWid \* bufHei \* 4 字节 |
| bufWid、bufHei | bits的宽高 |
| xDst、yDst | 绘制到bits的哪个坐标 |
| w、h、xSrc、ySrc | mb需要取的画面的起始坐标 |
| bCopyAlpha | 是否拷贝画面的透明度值 |

注意：此函数一般给3d游戏使用。另外频繁使用此接口并拷贝像素有性能问题。最好用wkeGetViewDC再去拷贝dc。

### void wkePaint(wkeWebView webView, void\* bits, int pitch)

获取页面的像素的简化版函数。

参数：

bits：外部申请并传递给mb的buffer，大小是webview宽度 \* 高度 \* 4 字节。  
pitch：填0即可。这个参数玩过directX的人应该懂

### void wkeRepaintIfNeeded(wkeWebView webView)

（被废弃）强制重绘页面

参数：略

### HDC wkeGetViewDC(wkeWebView webView)

获取webview的DC

参数：略

### HWND wkeGetHostHWND(wkeWebView webView)

获取webveiw对应的窗口句柄。实现和wkeGetWindowHandle完全相同

参数：略

### bool wkeCanGoBack(wkeWebView webView)

页面是否可以后退

参数：略

### bool wkeGoBack(wkeWebView webView)

强制让页面后退

参数：略

### bool wkeCanGoForward(wkeWebView webView)

略

参数：略

### bool wkeGoForward(wkeWebView webView)

略

参数：略

### void wkeEditorSelectAll(wkeWebView webView)

给webview发送全选命令

参数：略

### void wkeEditorUnSelect(wkeWebView webView)

略

参数：略

### void wkeEditorCopy(wkeWebView webView)

拷贝页面里被选中的字符串

参数：略

### void wkeEditorCut(wkeWebView webView)

略

参数：略

### void wkeEditorDelete(wkeWebView webView)

略

参数：略

### void wkeEditorUndo(wkeWebView webView)

略

参数：略

### void wkeEditorRedo(wkeWebView webView)

略

参数：略

### const wchar\_t \* wkeGetCookieW(wkeWebView webView)

获取页面的cookie

参数：略

### const utf8\* wkeGetCookie(wkeWebView webView)

获取页面的cookie

参数：略

### void wkeSetCookie(wkeWebView webView, const utf8\* url, const utf8\* cookie)

设置页面cookie。

参数：略

注意：cookie必须符合curl的cookie写法。至于curl的cookie怎么写，请自行搜索

### void wkeVisitAllCookie(void\* params, wkeCookieVisitor visitor)

通过访问器visitor访问所有cookie。

参数：

visitor：访问器。

### void wkePerformCookieCommand(wkeCookieCommand command)

通过设置mb内置的curl来操作cookie。

参数：

command：

wkeCookieCommandClearAllCookies: 内部实际执行了curl\_easy\_setopt(curl, CURLOPT\_COOKIELIST, "ALL");  
wkeCookieCommandClearSessionCookies: curl\_easy\_setopt(curl, CURLOPT\_COOKIELIST, "SESS");  
wkeCookieCommandFlushCookiesToFile: curl\_easy\_setopt(curl, CURLOPT\_COOKIELIST, "FLUSH");  
wkeCookieCommandReloadCookiesFromFile: curl\_easy\_setopt(curl, CURLOPT\_COOKIELIST, "RELOAD");

注意：这个接口只是调用curl设置命令，并不会去修改js里的内容

### void wkeSetCookieEnabled(wkeWebView webView, bool enable)

开启或关闭cookie

参数：略

注意：这个接口只是影响blink，并不会设置curl。所以还是会生成curl的cookie文件

### bool wkeIsCookieEnabled(wkeWebView webView)

略

参数：略

### void wkeSetCookieJarPath(wkeWebView webView, const WCHAR\* path)

设置cookie的本地文件目录。默认是当前目录。cookies存在当前目录的“cookie.dat”里

参数：略

### void wkeSetCookieJarFullPath(wkeWebView webView, const WCHAR\* path)

设置cookie的全路径+文件名，如“c:\mb\cookie.dat”

参数：略

### void wkeSetLocalStorageFullPath(wkeWebView webView, const WCHAR\* path)

设置local storage的全路径。如“c:\mb\LocalStorage\”

参数：略

注意：这个接口只能接受目录。

### void wkeSetMediaVolume(wkeWebView webView, float volume)

设置音量，未实现

参数：略

### float wkeGetMediaVolume(wkeWebView webView)

获取音量，未实现

参数：略

### bool wkeFireMouseEvent(wkeWebView webView, unsigned int message, int x, int y, unsigned int flags)

向mb发送鼠标消息

参数：

message：可取WM\_MOUSELEAVE等Windows相关鼠标消息  
x、y：坐标  
flags：可取值有WKE\_CONTROL、WKE\_SHIFT、WKE\_LBUTTON、WKE\_MBUTTON、WKE\_RBUTTON，可通过“或”操作并联。

### bool wkeFireContextMenuEvent(wkeWebView webView, int x, int y, unsigned int flags)

向mb发送菜单消息（未实现）

参数：略

### bool wkeFireMouseWheelEvent(wkeWebView webView, int x, int y, int delta, unsigned int flags)

向mb发送滚轮消息，用法和参数类似wkeFireMouseEvent。

参数：略

### bool wkeFireKeyUpEvent(wkeWebView webView, unsigned int virtualKeyCode, unsigned int flags, bool systemKey)

向mb发送WM\_KEYUP消息，

参数：

virtualKeyCode：见https://msdn.microsoft.com/en-us/library/windows/desktop/dd375731(v=vs.85).aspx flags：可取值有WKE\_REPEAT、WKE\_EXTENDED，可通过“或”操作并联。 systemKey：暂时没用

### bool wkeFireKeyDownEvent(wkeWebView webView, unsigned int virtualKeyCode, unsigned int flags, bool systemKey)

略

参数：略

### bool wkeFireKeyPressEvent(wkeWebView webView, unsigned int charCode, unsigned int flags, bool systemKey)

略

参数：

charCode：WM\_CHAR消息的The character code of the key.见https://msdn.microsoft.com/en-us/library/windows/desktop/ms646276(v=vs.85).aspx

### bool wkeFireWindowsMessage(wkeWebView webView, HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam, LRESULT\* result)

向mb发送任意windows消息。不过目前mb主要用来处理光标相关。mb在无窗口模式下，要响应光标事件，需要通过本函数手动发送光标消息

参数：略

### void wkeSetFocus(wkeWebView webView)

设置webview是焦点态。如果webveiw关联了窗口，窗口也会有焦点

参数：略

### void wkeKillFocus(wkeWebView webView)

略

参数：略

### wkeRect wkeGetCaretRect(wkeWebView webView)

获取编辑框的那个游标的位置

参数：略

### jsValue wkeRunJS(wkeWebView webView, const utf8\* script)

运行一段js。返回js的值jsValue。jsValue是个封装了内部v8各种类型的类，如果需要获取详细信息，有jsXXX相关接口可以调用。见下述。

参数：略

### jsValue wkeRunJSW(wkeWebView webView, const wchar\_t\* script)

同上。注意，此函数以及wkeRunJS，执行的js，也就是script，是在一个闭包中

参数：略

### jsExecState wkeGlobalExec(wkeWebView webView)

获取页面主frame的jsExecState。jsExecState是什么，见下述。

参数：略

### void wkeSleep(wkeWebView webView)

暂未实现

参数：略

### void wkeWake(wkeWebView webView)

略

参数：略

### bool wkeIsAwake(wkeWebView webView)

暂未实现

参数：略

### void wkeSetZoomFactor(wkeWebView webView, float factor)

设置页面缩放系数，默认是1

参数：略

### float wkeGetZoomFactor(wkeWebView webView)

略

参数：略

### void wkeSetEditable(wkeWebView webView, bool editable)

未实现

参数：略

### void wkeOnTitleChanged(wkeWebView webView, wkeTitleChangedCallback callback, void\* callbackParam)

设置标题变化的通知回调

参数：typedef void(\*wkeTitleChangedCallback)(wkeWebView webView, void\* param, const wkeString title);

title：标题的字符串封装。wkeString怎么用，见相关接口。 param：通过wkeOnTitleChanged的callbackParam设置

### void wkeOnMouseOverUrlChanged(wkeWebView webView, wkeTitleChangedCallback callback, void\* callbackParam)

鼠标划过的元素，如果是，则调用此回调，并发送a标签的url

参数：略

### void wkeOnURLChanged(wkeWebView webView, wkeURLChangedCallback callback, void\* callbackParam)

url改变回调

参数：略

### void wkeOnURLChanged2(wkeWebView webView, wkeURLChangedCallback2 callback, void\* callbackParam)

和上个接口不同的是，回调多了个参数

参数：typedef void(\*wkeURLChangedCallback2)(wkeWebView webView, void\* param, wkeWebFrameHandle frameId, const wkeString url)

frameId：表示frame的id。有相关接口可以判断这个frameId是否是主frame

### void wkeOnPaintUpdated(wkeWebView webView, wkePaintUpdatedCallback callback, void\* callbackParam)

页面有任何需要刷新的地方，将调用此回调

参数：typedef void(\*wkePaintUpdatedCallback)(wkeWebView webView, void\* param, const HDC hdc, int x, int y, int cx, int cy)

x、y、cx、cy表示刷新的区域矩形

### void wkeOnPaintBitUpdated(wkeWebView webView, wkePaintBitUpdatedCallback callback, void\* callbackParam)

同上。不同的是回调过来的是填充好像素的buffer，而不是DC。方便嵌入到游戏中做离屏渲染

参数：略

### void wkeOnAlertBox(wkeWebView webView, wkeAlertBoxCallback callback, void\* callbackParam)

网页调用alert会走到这个接口填入的回调

参数：略

### void wkeOnConfirmBox(wkeWebView webView, wkeConfirmBoxCallback callback, void\* callbackParam)

略

参数：略

### void wkeOnPromptBox(wkeWebView webView, wkePromptBoxCallback callback, void\* callbackParam)

略

参数：略

### void wkeOnNavigation(wkeWebView webView, wkeNavigationCallback callback, void\* param)

网页开始浏览将触发回调

参数：typedef bool(\*wkeNavigationCallback)(wkeWebView webView, void\* param, wkeNavigationType navigationType, const wkeString url);

wkeNavigationType: 表示浏览触发的原因。可以取的值有：

WKE\_NAVIGATION\_TYPE\_LINKCLICK：点击a标签触发  
WKE\_NAVIGATION\_TYPE\_FORMSUBMITTE：点击form触发  
WKE\_NAVIGATION\_TYPE\_BACKFORWARD：前进后退触发  
WKE\_NAVIGATION\_TYPE\_RELOAD：重新加载触发

注意：wkeNavigationCallback回调的返回值，如果是true，表示可以继续进行浏览，false表示阻止本次浏览。

### void wkeOnCreateView(wkeWebView webView, wkeCreateViewCallback callback, void\* param)

网页点击a标签创建新窗口时将触发回调

参数：typedef wkeWebView(\*wkeCreateViewCallback)(wkeWebView webView, void\* param, wkeNavigationType navigationType, const wkeString url, const wkeWindowFeatures\* windowFeatures);

### void wkeOnDocumentReady(wkeWebView webView, wkeDocumentReadyCallback callback, void\* param)

对应js里的body onload事件

参数：略

### void wkeOnDocumentReady2(wkeWebView webView, wkeDocumentReady2Callback callback, void\* param)

同上。区别是wkeDocumentReady2Callback多了wkeWebFrameHandle frameId参数。可以判断是否是主frame

参数：略

### void wkeOnDownload(wkeWebView webView, wkeDownloadCallback callback, void\* param)

页面下载事件回调。点击某些链接，触发下载会调用

参数：略

### void wkeNetOnResponse(wkeWebView webView, wkeNetResponseCallback callback, void\* param)

一个网络请求发送后，收到服务器response触发回调

参数：略

### void wkeOnConsole(wkeWebView webView, wkeConsoleCallback callback, void\* param)

网页调用console触发

参数：略

### void wkeSetUIThreadCallback(wkeWebView webView, wkeCallUiThread callback, void\* param)

暂时未实现

参数：略

### void wkeOnLoadUrlBegin(wkeWebView webView, wkeLoadUrlBeginCallback callback, void\* callbackParam)

任何网络请求发起前会触发此回调

参数：typedef bool(\*wkeLoadUrlBeginCallback)(wkeWebView webView, void\* param, const char \*url, void \*job)

注意：  
1，此回调功能强大，在回调里，如果对job设置了wkeNetHookRequest，则表示mb会缓存获取到的网络数据，并在这次网络请求 结束后调用wkeOnLoadUrlEnd设置的回调，同时传递缓存的数据。在此期间，mb不会处理网络数据。  
2，如果在wkeLoadUrlBeginCallback里没设置wkeNetHookRequest，则不会触发wkeOnLoadUrlEnd回调。  
3，如果wkeLoadUrlBeginCallback回调里返回true，表示mb不处理此网络请求（既不会发送网络请求）。返回false，表示mb依然会发送网络请求。  
用法举例：  
假如需要hook百度某个url（如httdiv://baidu.com/a.js）,替换为本地c:\b.js，则可以这样实现：

void readJsFile(const wchar\_t\* divath, std::vector\* buffer) {

HANDLE hFile = CreateFileW(divath, GENERIC\_READ, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (INVALID\_HANDLE\_VALUE == hFile) {

DebugBreak();

return;

}

DWORD fileSizeHigh;

const DWORD bufferSize = ::GetFileSize(hFile, &amdiv;fileSizeHigh);

DWORD numberOfBytesRead = 0;

buffer->resize(bufferSize);

BOOL b = ::ReadFile(hFile, &amdiv;buffer->at(0), bufferSize, &amdiv;numberOfBytesRead, nulldivtr);

::CloseHandle(hFile);

}

static bool HookUrl(void\* job, const char\* url, const char\* hookUrl, const wchar\_t\* localFile, const char\* mime) {

if (0 != strstr(url, hookUrl)) {

wkeNetSetMIMETydive(job, (char\*)mime); // "text/html" "text/javascridivt"

std::vector buffer;

ReadJsFile(localFile, &amdiv;buffer);

wkeNetSetData(job, &amdiv;buffer[0], buffer.size());

return true;

}

return false;

}

bool handleLoadUrlBegin(wkeWebView webView, void\* divaram, const char\* url, void\* job) {

if (HookUrl(job, url, "httdiv://baidu.com/a.js", L"c:\\b.js", "text/javascridivt"))

return true;

return false;

}

如果需要拿到httdiv://baidu.com/a.js的真实网络数据再修改，则可以：

bool handleLoadUrlBegin(wkeWebView webView, void\* divaram, const char\* url, void\* job) {

if (0 != strstr(url, "httdiv://baidu.com/a.js")) {

wkeNetHookRequest(job);

return false;

}

return false;

}

void handleLoadUrlEnd(wkeWebView webView, void\* divaram, const char\* url, void\* job, void\* buf, int len) {

char code[] = "console.log('test')";

wkeNetSetData(job, code, sizeof(code));

}

### void wkeOnLoadUrlEnd(wkeWebView webView, wkeLoadUrlEndCallback callback, void\* callbackParam)

见wkeOnLoadUrlBegin的描述

参数：略

### void wkeOnDidCreateScriptContext(wkeWebView webView, wkeDidCreateScriptContextCallback callback, void\* callbackParam)

javascript的v8执行环境被创建时触发此回调

参数：略

注意：每个frame创建时都会触发此回调

### void wkeOnWillReleaseScriptContext(wkeWebView webView, wkeWillReleaseScriptContextCallback callback, void\* callbackParam)

每个frame的javascript的v8执行环境被关闭时触发此回调

参数：略

### void wkeOnWillMediaLoad(wkeWebView webView, wkeWillMediaLoadCallback callback, void\* callbackParam)

video等多媒体标签创建时触发此回调

参数：略

### bool wkeIsMainFrame(wkeWebView webView, wkeWebFrameHandle frameId)

判断frameId是否是主frame

参数：略

### wkeWebFrameHandle wkeWebFrameGetMainFrame(wkeWebView webView)

获取主frame的句柄

参数：略

### jsValue wkeRunJsByFrame(wkeWebView webView, wkeWebFrameHandle frameId, const utf8\* script, bool isInClosure)

运行js在指定的frame上，通过frameId

参数：isInClosure表示是否在外层包个function() {}形式的闭包

注意：如果需要返回值，在isInClosure为true时，需要写return，为false则不用

### const utf8\* wkeGetFrameUrl(wkeWebView webView, wkeWebFrameHandle frameId)

获取frame对应的url

参数：略

### const utf8\* wkeGetString(const wkeString s)

获取wkeString结构体对应的字符串，utf8编码

参数：略

### const wchar\_t\* wkeGetStringW(const wkeString string)const wchar\_t\* wkeGetStringW(const wkeString string)

获取wkeString结构体对应的字符串，utf16编码

参数：略

### void wkeSetString(wkeString string, const utf8\* str, size\_t len)

设置wkeString结构体对应的字符串，utf8编码

参数：略

### void wkeSetStringW(wkeString string, const wchar\_t\* str, size\_t len)

设置wkeString结构体对应的字符串，utf16编码

参数：略

### wkeString wkeCreateStringW(const wchar\_t\* str, size\_t len)

通过utf16编码的字符串，创建一个wkeString

参数：略

### void wkeDeleteString(wkeString str)

析构这个wkeString

参数：略

### void wkeSetUserKeyValue(wkeWebView webView, const char\* key, void\* value)

对webView设置一个key value键值对。可以用来保存用户自己定义的任何指针

参数：略

### void\* wkeGetUserKeyValue(wkeWebView webView, const char\* key)

略

参数：略

### int wkeGetCursorInfoType(wkeWebView webView)

略

参数：略

### wkeWebView wkeCreateWebView()

创建一个webview，但不创建真窗口。一般用在离屏渲染里，如游戏

参数：略

### void wkeDestroyWebView(wkeWebView webView)

效果同wkeDestroyWebWindow

参数：略

### wkeWebView wkeCreateWebWindow(wkeWindowType type, HWND parent, int x, int y, int width, int height)

创建一个带真实窗口的wkeWebView

参数：wkeWindowType

WKE\_WINDOW\_TYPE\_POPUP：      普通窗口  
WKE\_WINDOW\_TYPE\_TRANSPARENT：透明窗口。mb内部通过layer window实现  
WKE\_WINDOW\_TYPE\_CONTROL：    嵌入在父窗口里的子窗口。此时parent需要被设置

### void wkeDestroyWebWindow(wkeWebView webWindow)

销毁wkeWebView对应的所有数据结构，包括真实窗口等

参数：略

### HWND wkeGetWindowHandle(wkeWebView webWindow)

获取窗口对应的真实句柄。和wkeGetHostHWND的实现完全相同

参数：略

### void wkeOnWindowClosing(wkeWebView webWindow, wkeWindowClosingCallback callback, void\* param)

wkeWebView如果是真窗口模式，则在收到WM\_CLODE消息时触发此回调。可以通过在回调中返回false拒绝关闭窗口

参数：略

### void wkeOnWindowDestroy(wkeWebView webWindow, wkeWindowDestroyCallback callback, void\* param)

窗口即将被销毁时触发回调。不像wkeOnWindowClosing，这个操作无法取消

参数：略

### void wkeShowWindow(wkeWebView webWindow, bool showFlag)

略

参数：略

### void wkeEnableWindow(wkeWebView webWindow, bool enableFlag)

略

参数：略

### void wkeMoveWindow(wkeWebView webWindow, int x, int y, int width, int height)

略

参数：略

### void wkeMoveToCenter(wkeWebView webWindow)

窗口在父窗口或屏幕里居中

参数：略

### void wkeResizeWindow(wkeWebView webWindow, int width, int height)

resize窗口，和wkeResize效果一样

参数：略

### void wkeSetWindowTitle(wkeWebView webWindow, const utf8\* title)

略

参数：略

### void wkeSetWindowTitle(wkeWebView webWindow, const wchar\_t\* title)

略

参数：略

### void wkeSetDeviceParameter(wkeWebView webView, const char\* device, const char\* paramStr, int paramInt, float paramFloat)

设置mb模拟的硬件设备环境。主要用在伪装手机设备场景

参数：

device：设备的字符串。可取值有：

|  |  |
| --- | --- |
| "navigator.maxTouchPoints" | 此时 paramInt 需要被设置，表示 touch 的点数 |
| "navigator.platform" | 此时 paramStr 需要被设置，表示js里获取的 navigator.platform字符串 |
| "navigator.hardwareConcurrency" | 此时 paramInt 需要被设置，表示js里获取的 navigator.hardwareConcurrency 整数值 |
| "screen.width" | 此时 paramInt 需要被设置，表示js里获取的 screen.width 整数值 |
| "screen.height" | 此时 paramInt 需要被设置，表示js里获取的 screen.height 整数值 |
| "screen.availWidth" | 此时 paramInt 需要被设置，表示js里获取的 screen.availWidth 整数值 |
| "screen.availHeight" | 此时 paramInt 需要被设置，表示js里获取的 screen.availHeight 整数值 |
| "screen.pixelDepth" | 此时 paramInt 需要被设置，表示js里获取的 screen.pixelDepth 整数值 |
| "screen.pixelDepth" | 目前等价于"screen.pixelDepth" |
| "window.devicePixelRatio" | 同上 |

### void wkeInit()

初始化整个mb。此句必须在所有mb api前最先调用。并且所有mb api必须和调用wkeInit的线程为同个线程

参数：略

### void wkeShutdown()

关闭整个mb。调用后，所有mb api将不能再被调用，否则会有崩溃

参数：略

### void wkeInitialize()

效果和wkeInit一模一样

参数：略

### void wkeSetProxy(const wkeProxy\* proxy)

设置整个mb的代码。此句是全局生效

参数：略

### void wkeSetViewProxy(wkeWebView webView, wkeProxy\* proxy)

设置整个mb的代码。此句是针对特定webview生效

参数：略

### void wkeConfigure(const wkeSettings\* settings)

设置一些配置项

参数：

typedef struct { wkeProxy proxy; unsigned int mask; } wkeSettings; mask可以取：

WKE\_SETTING\_PROXY：效果和wkeSetProxy一样，通过proxy设置  
WKE\_SETTING\_PAINTCALLBACK\_IN\_OTHER\_THREAD：这是个高级用法，开启时，on paint回调会在另外个线程（其实就是渲染线程）。  
这个是用来实现多线程上屏功能，性能更快。

### bool wkeIsInitialize()

略

参数：略

### void wkeFinalize()

效果和wkeShutdown一样

参数：略

### void wkeSetMemoryCacheEnable(wkeWebView webView, bool b)

开启内存缓存。网页的图片等都会在内存缓存里。关闭后，内存使用会降低一些，但容易引起一些问题，如果不懂怎么用，最好别开

参数：略

### void wkeSetTouchEnabled(wkeWebView webView, bool b)

开启触屏模式。开启后，鼠标消息将自动转换成触屏消息

参数：略

### void wkeSetMouseEnabled(wkeWebView webView, bool b)

开启关闭鼠标消息，可以在开启触屏后，关闭鼠标消息

参数：略

### void wkeSetNavigationToNewWindowEnable(wkeWebView webView, bool b)

关闭后，点a标签将不会弹出新窗口，而是在本窗口跳转

参数：略

### void wkeSetCspCheckEnable(wkeWebView webView, bool b)

关闭后，跨域检查将被禁止，此时可以做任何跨域操作，如跨域ajax，跨域设置iframe

参数：略

### void wkeSetNpapiPluginsEnabled(wkeWebView webView, bool b)

开启关闭npapi插件，如flash

参数：略

### void wkeSetHeadlessEnabled(wkeWebView webView, bool b)

开启无头模式。开启后，将不会渲染页面，提升了网页性能。此功能方便用来实现一些爬虫，或者刷单工具

参数：略

### void wkeSetDebugConfig(wkeWebView webView, const char\* debugString, const char\* param)

开启一些实验性选项。

参数：

debugString：

|  |  |
| --- | --- |
| "showDevTools" | 开启开发者工具，此时param要填写开发者工具的资源路径，如file:///c:/miniblink-release/front\_end/inspector.html。注意param此时必须是utf8编码 |
| "wakeMinInterval" | 设置帧率，默认值是10，值越大帧率越低 |
| "drawMinInterval" | 设置帧率，默认值是3，值越大帧率越低 |
| "antiAlias" | 设置抗锯齿渲染。param必须设置为"1" |
| "minimumFontSize" | 最小字体 |
| "minimumLogicalFontSize" | 最小逻辑字体 |
| "defaultFontSize" | 默认字体 |
| "defaultFixedFontSize" | 默认fixed字体 |

### void wkeSetHandle(wkeWebView webView, HWND wnd)

设置wkeWebView对应的窗口句柄。

参数：略

注意：只有在无窗口模式下才能使用。如果是用wkeCreateWebWindow创建的webview，已经自带窗口句柄了。

### void wkeSetHandleOffset(wkeWebView webView, int x, int y)

设置无窗口模式下的绘制偏移。在某些情况下（主要是离屏模式），绘制的地方不在真窗口的(0, 0)处，就需要手动调用此接口

参数：略

### void wkeSetViewSettings(wkeWebView webView, const wkeViewSettings\* settings)

设置一些webview相关的设置.目前只有背景颜色可以设置

参数：略

### void wkeSetTransparent(wkeWebView webView, bool transparent)

通知无窗口模式下，webview开启透明模式。

参数：略

### bool wkeIsTransparent(wkeWebView webView)

判断窗口是否是分层窗口（layer window）

参数：略

### void wkeSetUserAgent(wkeWebView webView, const utf8\* userAgent)

设置webview的UA

参数：略

### void wkeSetUserAgentW(wkeWebView webView, const wchar\_t\* userAgent)

略

参数：略

### const utf8\* wkeGetUserAgent(wkeWebView webView)

获取webview的UA

参数：略

### void wkeLoadURL(wkeWebView webView, const utf8\* url)

加载url。url必须是网络路径，如"http://qq.com/"

参数：略

### void wkeLoadW(wkeWebView webView, const wchar\_t\* url)

略

参数：略

### void wkeLoadHTML(wkeWebView webView, const utf8\* html)

加载一段html

参数：略

注意：如果html里有相对路径，则是相对exe所在目录的路径

### void wkeLoadHtmlWithBaseUrl(wkeWebView webView, const utf8\* html, const utf8\* baseUrl)

加载一段html，但可以指定baseURL，也就是相对于哪个目录的url

参数：略

### void wkeLoadFile(wkeWebView webView, const utf8\* filename)

略

参数：略

### const utf8\* wkeGetURL(wkeWebView webView)

获取webview主frame的url

参数：略

### void wkeNetSetHTTPHeaderField(void\* jobPtr, wchar\_t\* key, wchar\_t\* value, bool response)

在wkeOnLoadUrlBegin回调里调用，表示设置http请求（或者file:///协议）的 http header field。response一直要被设置成false

参数：略

### void wkeNetSetMIMEType(void\* jobPtr, char\* type)

在wkeOnLoadUrlBegin回调里调用，表示设置http请求的MIME type

参数：略

### const char\* wkeNetGetMIMEType(void\* jobPtr, wkeString mime)

略

参数：第2个参数可以传nullptr

### void wkeNetSetData(void\* jobPtr, void\* buf, int len)

在wkeOnLoadUrlEnd里被调用，表示设置hook后缓存的数据

参数：略

### void wkeNetCancelRequest(void\* jobPtr)

在wkeOnLoadUrlBegin回调里调用，设置后，此请求将被取消。

参数：略

### int wkeNetGetFavicon(wkeWebView webView, wkeOnNetGetFavicon callback, void\* param)

获取favicon。

参数：略

注意：此接口必须在wkeOnLoadingFinish回调里调用。可以用下面方式来判断是否是主frame的LoadingFinish:

wkeTempCallbackInfo\* temInfo = wkeGetTempCallbackInfo(webView);

if (::wkeIsMainFrame(webView, temInfo->frame)) {

    ::wkeNetGetFavicon(webView, HandleFaviconReceived, divaram);

}

### void wkeNetHoldJobToAsynCommit(void\* jobPtr)

高级用法。在wkeOnLoadUrlBegin回调里调用。 有时候，wkeOnLoadUrlBegin里拦截到一个请求后，不能马上判断出结果。此时可以调用本接口，然后在 异步的某个时刻，调用wkeNetContinueJob来让此请求继续进行

参数：略

### wkeRequestType wkeNetGetRequestMethod(void \*jobPtr)

获取此请求的method，如post还是get

参数：略

### wkePostBodyElements\* wkeNetGetPostBody(void \*jobPtr)

获取此请求中的post数据。只有当请求是post时才有效果

参数：略

### wkePostBodyElements\* wkeNetCreatePostBodyElements(wkeWebView webView, size\_t length)

略

参数：略

### void wkeNetFreePostBodyElements(wkePostBodyElements\* elements)

略

参数：略

### void wkeNetFreePostBodyElements(wkePostBodyElements\* elements)

略

参数：略

### wkePostBodyElement\* wkeNetCreatePostBodyElement(wkeWebView webView)

略

参数：略

### void wkeNetFreePostBodyElement(wkePostBodyElement\* element)

这四个接口要结合起来使用。 当wkeOnLoadUrlBegin里判断是post时，可以通过wkeNetCreatePostBodyElements来创建一个新的post数据包。 然后wkeNetFreePostBodyElements来释放原post数据。

参数：略

### int jsArgCount(jsExecState es)

获取es里存的参数个数。一般是在绑定的js调用c++回调里使用，判断js传递了多少参数给c++

参数：略

### jsType jsArgType(jsExecState es, int argIdx)

判断第argIdx个参数的参数类型。argIdx从是个0开始计数的值。如果超出jsArgCount返回的值，将发生崩溃

参数：略

### jsValue jsArg(jsExecState es, int argIdx)

获取第argIdx对应的参数的jsValue值。

参数：略

### jsType jsTypeOf(jsValue v)

获取v对应的类型。

参数：略

### bool jsIsNumber(jsValue v)

判断v是否为数字

参数：略

### bool jsIsString(jsValue v)

略

参数：略

### bool jsIsBoolean(jsValue v)

略

参数：略

### bool jsIsObject(jsValue v)

当v不是数字、字符串、undefined、null、函数的时候，此接口返回true

参数：略

### bool jsIsTrue(jsValue v)

如果v本身是个布尔值，返回对应的true或者false；如果是个对象（JSTYPE\_OBJECT），返回false（这里注意）

参数：略

### bool jsIsFalse(jsValue v)

等价于!jsIsTrue(v)

参数：略

### int jsToInt(jsExecState es, jsValue v)

如果v是个整形或者浮点，返回相应值（如果是浮点，返回取整后的值）。如果是其他类型，返回0（这里注意）

参数：略

### double jsToDouble(jsExecState es, jsValue v)

如果v是个浮点形，返回相应值。如果是其他类型，返回0.0（这里注意）

参数：略

### const wchar\_t\* jsToTempStringW(jsExecState es, jsValue v)

如果v是个字符串，返回相应值。如果是其他类型，返回L""（这里注意） 另外，返回的字符串不需要外部释放。mb会在下一帧自动释放

参数：略

### const utf8\* jsToTempString(jsExecState es, jsValue v)

同上

参数：略

### const utf8\* jsToString(jsExecState es, jsValue v)

同上，只是返回的是utf8编码

参数：略

### const wchar\_t\* jsToStringW(jsExecState es, jsValue v)

略

参数：略

### jsValue jsInt(int n)

创建建一个int型的jsValue，注意是创建

参数：略

### jsValue jsString(jsExecState es, const utf8\* str)

构建一个utf8编码的字符串的的jsValue。str会在内部拷贝保存，注意是创建

参数：略

### jsValue jsArrayBuffer(jsExecState es, char \* buffer, size\_t size)

构建一个js的arraybuffer类型的jaValue。主要用来处理一些二进制数据，注意是创建

参数：略

### wkeMemBuf\* jsGetArrayBuffer(jsExecState es, jsValue value)

获取一个js的arraybuffer类型的数据。主要用来处理一些二进制数据

参数：略

### jsValue jsEmptyObject(jsExecState es)

构建一个临时js object的jsValue，注意是创建

参数：略

### jsValue jsEvalW(jsExecState es, const wchar\_t\* str)

执行一段js，并返回值。

参数：略

注意：str的代码会在mb内部自动被包裹在一个function(){}中。所以使用的变量会被隔离 注意：要获取返回值，请写return。这和wke不太一样。wke不需要写retrun

### jsValue jsEvalExW(jsExecState es, const wchar\_t\* str, bool isInClosure)

和上述接口的区别是，isInClosure表示是否要包裹一层function(){}。jsEvalW相当于jsEvalExW(es, str, false)

参数：略

注意：如果需要返回值，在isInClosure为true时，需要写return，为false则不用

### jsValue jsCall(jsExecState es, jsValue func, jsValue thisValue, jsValue\* args, int argCount)

调用一个func对应的js函数。如果此js函数是成员函数，则需要填thisValue。 否则可以传jsUndefined。args是个数组，个数由argCount控制。 func可以是从js里取的，也可以是自行构造的。

参数：略

### jsValue jsCallGlobal(jsExecState es, jsValue func, jsValue\* args, int argCount)

调用window上的全局函数

参数：略

### jsValue jsGet(jsExecState es, jsValue object, const char\* prop)

如果object是个js的object，则获取prop指定的属性。如果object不是js object类型，则返回jsUndefined

参数：略

### void jsSet(jsExecState es, jsValue object, const char\* prop, jsValue value)

设置object的属性

参数：略

### jsValue jsGetGlobal(jsExecState es, const char\* prop)

获取window上的属性

参数：略

### void jsSetGlobal(jsExecState es, const char\* prop, jsValue v)

设置window上的属性

参数：略

### jsValue jsGetAt(jsExecState es, jsValue object, int index)

设置js arrary的第index个成员的值，object必须是js array才有用，否则会返回jsUndefined

参数：略

### void jsSetAt(jsExecState es, jsValue object, int index, jsValue value)

设置js arrary的第index个成员的值，object必须是js array才有用。

参数：略

### jsKeys\* jsGetKeys(jsExecState es, jsValue object)

获取object有哪些key

参数：略

### int jsGetLength(jsExecState es, jsValue object)

获取js arrary的长度，object必须是js array才有用。

参数：略

### void jsSetLength(jsExecState es, jsValue object, int length)

设置js arrary的长度，object必须是js array才有用。

参数：略

### wkeWebView jsGetWebView(jsExecState es)

获取es对应的webview

参数：略

### void jsGC()

强制垃圾回收

参数：略

### void fastcall jsBindFunction(const char\* name, jsNativeFunction fn, unsigned int argCount)

绑定一个全局函数到主frame的window上。

参数：略

注意：此接口只能绑定主frame，并且特别需要注意的是，因为历史原因，此接口是fastcall调用约定！（但wkeJsBindFunction不是）  
另外此接口和wkeJsBindFunction必须在webview创建前调用  
使用示例：

c++里：

--------

jsValue JS\_CALL onNativeFunction(jsExecState es) {

    jsValue v = jsArg(es, 0);

    const wchar\_t\* str = jsToTemdivStringW(es, v);

    OutdivutdebugStringW(str);

}

jsBindFunction("testCall", onNativeFunction， 1);

js里：

--------

window.testCall('testStrt');

### void jsBindGetter(const char\* name, jsNativeFunction fn)

对js winows绑定一个属性访问器，在js里windows.XXX这种形式调用时，fn会被调用

参数：略

示例：jsBindGetter("XXX")

### void jsBindSetter(const char\* name, jsNativeFunction fn)

对js winows绑定一个属性设置器。

参数：略

### void wkeJsBindFunction(const char\* name, wkeJsNativeFunction fn, void\* param, unsigned int argCount)

和jsBindFunction功能类似，但更方便一点，可以传一个param做自定义数据。

参数：略

此接口和wkeJsBindFunction必须在webview创建前调用

### jsValue jsObject(jsExecState es, jsData\* data)

构建一个js Objcet，可以传递给js使用。

参数：

typedef jsValue(\*jsGetPropertyCallback)(jsExecState es, jsValue object, const char\* propertyName);  
  属性访问器。在js里XXX.yyy这样调用时，jsGetPropertyCallback会被触发，并且propertyName的值为yyy  
typedef bool(\*jsSetPropertyCallback)(jsExecState es, jsValue object, const char\* propertyName, jsValue value);  
  属性设置器。在js里XXX.yyy=1这样调用时，jsSetPropertyCallback会被触发，并且propertyName的值为yyy，value的值是字符串类型。可以通过之前提到的字符串操作接口来获取  
typedef jsValue(\*jsCallAsFunctionCallback)(jsExecState es, jsValue object, jsValue\* args, int argCount);  
  在js里XXX()这样调用时会触发。  
typedef void(\*jsFinalizeCallback)(struct tagjsData\* data);  
  在js里没人引用，且垃圾回收时会触发

### jsValue jsFunction(jsExecState es, jsData\* data)

创建一个主frame的全局函数。jsData的用法如上。js调用：XXX() 此时jsData的callAsFunction触发。 其实jsFunction和jsObject功能基本类似。且jsObject的功能更强大一些

参数：略

### jsData\* jsGetData(jsExecState es, jsValue value)

获取jsObject或jsFunction创建的jsValue对应的jsData指针。

参数：略

### jsExceptionInfo\* jsGetLastErrorIfException(jsExecState es);

当wkeRunJs、jsCall等接口调用时，如果执行的js代码有异常，此接口将获取到异常信息。否则返回nullptr。

参数：略