ChakraCore-JSRT

行之 / 2018-07-17 13:47:32 / 浏览数 3509 技术文章 技术文章 顶(0) 踩(0)

JSRT概观

JSRT API提供了一种将ChakraCore的嵌入程序，并且可以使用JavaScript的方法。

示例代码

目录：\test\native-tests\test-shared-basic\sample.cpp

```
#include "stdafx.h"
// Fixed by PR: https://github.com/Microsoft/ChakraCore/pull/2511
// #include <stdint.h>     // To work around issue #2510 temporarily:
//                         // https://github.com/Microsoft/ChakraCore/issues/2510
#include "ChakraCore.h"
#include <string>
#include <iostream>
using namespace std;
int main()
{
   JsRuntimeHandle runtime;
   JsContextRef context;
   JsValueRef result;
   unsigned currentSourceContext = 0;
   // Your script; try replace hello-world with something else
   wstring script = L"(()=>{return \'Hello world!\';})()";
   // Create a runtime.
   JsCreateRuntime(JsRuntimeAttributeNone, nullptr, &runtime);
   // Create an execution context.
   JsCreateContext(runtime, &context);
   // Now set the current execution context.
   JsSetCurrentContext(context);
   // Run the script.
   JsRunScript(script.c_str(), currentSourceContext++, L"", &result);
   // Convert your script result to String in JavaScript; redundant if your script returns a String
   JsValueRef resultJSString;
   JsConvertValueToString(result, &resultJSString);
   // Project script result back to C++.
   const wchar_t *resultWC;
   size_t stringLength;
   JsStringToPointer(resultJSString, &resultWC, &stringLength);
   wstring resultW(resultWC);
   cout << string(resultW.begin(), resultW.end()) << endl;
   system("pause");
   // Dispose runtime
   JsSetCurrentContext(JS_INVALID_REFERENCE);
   JsDisposeRuntime(runtime);
   return 0;
}
```

概念

了解如何使用JSRT API托管JavaScript引擎需要只知道两个关键概念：runtimes和execution contexts。

- 一个runtime表示一个完整的JavaScript执行环境。 每个runtime都会创建自己的garbage-collected heap，默认情况下，它有自己的实时编译器（JIT）线程和垃圾收集器（GC）线程。
- execution contexts表示一个拥有和别的execution contexts不同的JavaScript全局对象的JavaScript执行环境。
- 一个runtime可能包含多个execution contexts，在这种情况下，所有的execution context共享与runtime相关联的JIT thread和GC thread。
- runtimes表示单个线程执行。一次只能有一个runtime处于active状态，并且runtime一次只能在一个线程上处于活跃状态。runtimes是rental threaded，所以如果一个runtime在当前在线程上不处于active状态值，它可以被任何没有active runtime的线程调用。
- execution contexts与特定的runtime绑定，并且在这个runtime内执行代码。与runtimes不同的是，多个execution contexts可以同时在一个线程上处于active状态。因此一个host可以调用execution contexts，execution contexts可以回调host，并且host可以调用多个execution contexts。

- 事实上，除非host需要在分离的环境中运行代码，不然就可以使用单个execution context。同样除非主机需要同时运行多段代码，不然只要运行单个runtime就足够了。
- 只要不被设置成runtime的当前context或者有一个正数的引用计数，contexts都被GC收集。（更多的时候使用JsAddRef比JsRelease多）

在JSAPI中，JSRuntime是表示JavaScript引擎实例的顶级对象。程序通常只有一个JSRuntime，即使它有很多线程。JSRuntime是JavaScript对象的universe，一个JSRunti

所有JavaScript代码和大多数JSAPI的调用都在JSContext中运行。JSContext是JSRuntime的子代。一个Context可以运行script。每一个Context都包含全局对象和执行堆栈

创建运行时runtime

目录：\lib\Jsrt\

JsCreateRuntime

```
CHAKRA_API JsCreateRuntime(_In_ JsRuntimeAttributes attributes, _In_opt_ JsThreadServiceCallback threadService, _Out_ JsRuntim
{
   return CreateRuntimeCore(attributes,
       nullptr /*optRecordUri*/, 0 /*optRecordUriCount */, false /*isRecord*/, false /*isReplay*/, false /*isDebug*/,
       UINT_MAX /*optSnapInterval*/, UINT_MAX /*optLogLength*/,
       nullptr, nullptr, nullptr, nullptr, /*TTD IO handlers*/
       threadService, runtimeHandle);
}
```

参数：

attributes

创建runtime时需要的属性，ChakraCore中的定义如下：

```
typedef enum _JsRuntimeAttributes
 {
     /// ■■■■■■■
     JsRuntimeAttributeNone = 0x00000000,
     /// runtime■■■■■■■■■■■■■■■■■■■■■■
     JsRuntimeAttributeDisableBackgroundWork = 0x00000001,
     /// runtime■■■■■■■script interruption■■runtime■■script interruption■■■■■■■■■■■■■
     JsRuntimeAttributeAllowScriptInterrupt = 0x00000002,
     /// Host■■■■<c>Jsidle</c>,■■idle■■■■■runtime■■■■■■■■■■■■■
     JsRuntimeAttributeEnableIdleProcessing = 0x00000004,
     /// runtime■■■■native code
     JsRuntimeAttributeDisableNativeCodeGeneration = 0x00000008,
     /// ■■<c>eval</c> or <c>function</c>■■■■■■■■■■.
     JsRuntimeAttributeDisableEval = 0x00000010,
     /// runtime■■■■■■■experimental■■.
     JsRuntimeAttributeEnableExperimentalFeatures = 0x00000020,
     /// ■■JsSetException■■■■■■■■script debugger■■■■■■■■■■■■■■■■■■■■■
     JsRuntimeAttributeDispatchSetExceptionsToDebugger = 0x00000040,
     ///■■■■ ■■■■■■■■■
     ///■OOM■■■Failfast■■
     JsRuntimeAttributeDisableFatalOnOOM = 0x00000080,
 ///■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■WPA■■■■■JavaScript■■■■■■■■■■■■■■■■■■thunks■■■■■■■■■■■■■thunk■■■■■■■
     JsRuntimeAttributeDisableExecutablePageAllocation = 0x00000100,

 } JsRuntimeAttributes;
```

theadService

runtime的线程服务，可以为NULL

runtimeHandle

创建runtime

return：

- 如果操作成功则代码为JsNoError，否则为失败代码。

CreateRuntimeCore：

```
JsErrorCode CreateRuntimeCore(_In_ JsRuntimeAttributes attributes,
   _In_opt_ const char* optTTUri, size_t optTTUriCount, bool isRecord, bool isReplay, bool isDebug,
   _In_ UINT32 snapInterval, _In_ UINT32 snapHistoryLength,
   _In_opt_ TTDOpenResourceStreamCallback openResourceStream, _In_opt_ JsTTDReadBytesFromStreamCallback readBytesFromStream,
   _In_opt_ JsTTDWriteBytesToStreamCallback writeBytesToStream, _In_opt_ JsTTDFlushAndCloseStreamCallback flushAndCloseStream,
   _In_opt_ JsThreadServiceCallback threadService, _Out_ JsRuntimeHandle *runtimeHandle)
{
   VALIDATE_ENTER_CURRENT_THREAD();  //■■■■■■■■■■■■■■■

   PARAM_NOT_NULL(runtimeHandle); //■■■■■■■■■■■■■
   *runtimeHandle = nullptr;    //■■■■■■■0


   //■■■■■arributes■■■■■■■■■■■■■■■■■■runtime
   JsErrorCode runtimeResult = GlobalAPIWrapper_NoRecord([&]() -> JsErrorCode {
       const JsRuntimeAttributes JsRuntimeAttributesAll =
           (JsRuntimeAttributes)(
           JsRuntimeAttributeDisableBackgroundWork |
           JsRuntimeAttributeAllowScriptInterrupt |
           JsRuntimeAttributeEnableIdleProcessing |
           JsRuntimeAttributeDisableEval |
           JsRuntimeAttributeDisableNativeCodeGeneration |
           JsRuntimeAttributeDisableExecutablePageAllocation |
           JsRuntimeAttributeEnableExperimentalFeatures |
           JsRuntimeAttributeDispatchSetExceptionsToDebugger |
           JsRuntimeAttributeDisableFatalOnOOM
#ifdef ENABLE_DEBUG_CONFIG_OPTIONS
           | JsRuntimeAttributeSerializeLibraryByteCode
#endif
       );

       Assert((attributes & ~JsRuntimeAttributesAll) == 0);
       if ((attributes & ~JsRuntimeAttributesAll) != 0)
       {
           return JsErrorInvalidArgument;
       }
       CreateFileMapping(INVALID_HANDLE_VALUE, nullptr, PAGE_READWRITE, 0, 0, nullptr);
       AllocationPolicyManager * policyManager = HeapNew(AllocationPolicyManager, (attributes & JsRuntimeAttributeDisableBackg
       bool enableExperimentalFeatures = (attributes & JsRuntimeAttributeEnableExperimentalFeatures) != 0;
       ThreadContext * threadContext = HeapNew(ThreadContext, policyManager, threadService, enableExperimentalFeatures);

       if (((attributes & JsRuntimeAttributeDisableBackgroundWork) != 0)
#ifdef ENABLE_DEBUG_CONFIG_OPTIONS
           && !Js::Configuration::Global.flags.ConcurrentRuntime
#endif
           )
       {
           threadContext->OptimizeForManyInstances(true);
#if ENABLE_NATIVE_CODEGEN
           threadContext->EnableBgJit(false);
#endif
       }

       if (!threadContext->IsRentalThreadingEnabledInJSRT()
#ifdef ENABLE_DEBUG_CONFIG_OPTIONS
           || Js::Configuration::Global.flags.DisableRentalThreading
#endif
           )
       {
           threadContext->SetIsThreadBound();
       }

       if (attributes & JsRuntimeAttributeAllowScriptInterrupt)
       {
           threadContext->SetThreadContextFlag(ThreadContextFlagCanDisableExecution);
       }

       if (attributes & JsRuntimeAttributeDisableEval)
       {
           threadContext->SetThreadContextFlag(ThreadContextFlagEvalDisabled);
       }

       if (attributes & JsRuntimeAttributeDisableNativeCodeGeneration)
       {
           threadContext->SetThreadContextFlag(ThreadContextFlagNoJIT);
       }

       if (attributes & JsRuntimeAttributeDisableExecutablePageAllocation)
       {
           threadContext->SetThreadContextFlag(ThreadContextFlagNoJIT);
           threadContext->SetThreadContextFlag(ThreadContextFlagNoDynamicThunks);
       }

       if (attributes & JsRuntimeAttributeDisableFatalOnOOM)
       {
           threadContext->SetThreadContextFlag(ThreadContextFlagDisableFatalOnOOM);
       }

#ifdef ENABLE_DEBUG_CONFIG_OPTIONS
       if (Js::Configuration::Global.flags.PrimeRecycler)
       {
           threadContext->EnsureRecycler()->Prime();
       }
#endif

       bool enableIdle = (attributes & JsRuntimeAttributeEnableIdleProcessing) == JsRuntimeAttributeEnableIdleProcessing;
       bool dispatchExceptions = (attributes & JsRuntimeAttributeDispatchSetExceptionsToDebugger) == JsRuntimeAttributeDispatc

       JsrtRuntime * runtime = HeapNew(JsrtRuntime, threadContext, enableIdle, dispatchExceptions);
       threadContext->SetCurrentThreadId(ThreadContext::NoThread);
       *runtimeHandle = runtime->ToHandle();    //■■JsrtRuntime■■■■
#ifdef ENABLE_DEBUG_CONFIG_OPTIONS
       runtime->SetSerializeByteCodeForLibrary((attributes & JsRuntimeAttributeSerializeLibraryByteCode) != 0);
#endif

       return JsNoError;
   });
   ...

   return runtimeResult;
}
```

可以看到在创建JsrtRunTime的时候先创建了ThreadContext，而在\lib\Runtime\Base\ThreadContext.cpp里则看到了JsrtRunTime更多的成员属性，像是ThreadContex

TTD(Time Travel Debugging)：

是一个工具。可以用来记录正在执行的进程，可以使用TTD对象来查找加载特定代码模块的时间或查找所有异常。

目录：\lib\Common\CommonDfines.h

```
////////
//Time Travel flags
//Include TTD code in the build when building for Chakra (except NT/Edge) or for debug/test builds
#if defined(ENABLE_SCRIPT_DEBUGGING) && (!defined(NTBUILD) || defined(ENABLE_DEBUG_CONFIG_OPTIONS))
#define ENABLE_TTD 1
#else
#define ENABLE_TTD 0
#endif
```

创建执行上下文（Execution context）

目录：\lib\Jsrt\jsrt.cpp

JsCreateContext:

```
CHAKRA_API JsCreateContext(_In_ JsRuntimeHandle runtimeHandle, _Out_ JsContextRef *newContext)
{
   return GlobalAPIWrapper([&](TTDRecorder& _actionEntryPopper) -> JsErrorCode {
       PARAM_NOT_NULL(newContext);
```

```
        VALIDATE_INCOMING_RUNTIME_HANDLE(runtimeHandle);

        bool inRecord = false;
        bool activelyRecording = false;
        bool inReplay = false;

#if ENABLE_TTD
        JsrtRuntime * runtime = JsrtRuntime::FromHandle(runtimeHandle);
        //■■Runtime■■■■■■■
        // ■■■ \lib\Jsrt\JsrtRuntime.h
        // static JsrtRuntime * FromHandle(JsRuntimeHandle runtimeHandle)
        // {
        //      JsrtRuntime * runtime = static_cast<JsrtRuntime *>(runtimeHandle);
        //      runtime->threadContext->ValidateThreadContext();
        //      return runtime;
        //}

        //■■■ \lib\Runtime\Base\ThreadContext.cpp

        //void ThreadContext::ValidateThreadContext()
        //{
        //      #if DBG
        //    // verify the runtime pointer is valid.
        //{
        //      BOOL found = FALSE;
        //      AutoCriticalSection autocs(ThreadContext::GetCriticalSection());
        //      ThreadContext* currentThreadContext = GetThreadContextList();
        //      while (currentThreadContext)
        //      {
        //          if (currentThreadContext == this)
        //          {
        //              return;
        //          }
        //          currentThreadContext = currentThreadContext->Next();
        //      }
        //      AssertMsg(found, "invalid thread context");
        //}
        //       #endif
        //}



        ThreadContext * threadContext = runtime->GetThreadContext();  // ■■■■■■■■runtime   ThreadContext * GetThreadContext()
        if(threadContext->IsRuntimeInTTDMode() && threadContext->TTDContext->GetActiveScriptContext() != nullptr)
        {
            Js::ScriptContext* currentCtx = threadContext->TTDContext->GetActiveScriptContext();//■■ScriptContext
            inRecord = currentCtx->IsTTDRecordModeEnabled();
            activelyRecording = currentCtx->ShouldPerformRecordAction();
            inReplay = currentCtx->IsTTDReplayModeEnabled();
        }
#endif

        return CreateContextCore(runtimeHandle, _actionEntryPopper, inRecord, activelyRecording, inReplay, newContext);
    });
}
```

目录：\lib\Jsrt\Jsrtc.pp

CreateContextCore:

```
//A create context function that we can funnel to for regular and record or debug aware creation
JsErrorCode CreateContextCore(_In_ JsRuntimeHandle runtimeHandle, _In_ TTDRecorder& _actionEntryPopper, _In_ bool inRecordMode
{
    JsrtRuntime * runtime = JsrtRuntime::FromHandle(runtimeHandle);
    ThreadContext * threadContext = runtime->GetThreadContext();  //■■ThreadContext

    //■ThreadContext■■■■■■■
    if(threadContext->GetRecycler() && threadContext->GetRecycler()->IsHeapEnumInProgress())
    {
        return JsErrorHeapEnumInProgress;
```

```
   }
   else if(threadContext->IsInThreadServiceCallback())
   {
       return JsErrorInThreadServiceCallback;
   }

   ThreadContextScope scope(threadContext);  //■■■■runtime

   if(!scope.IsValid())
   {
       return JsErrorWrongThread;
   }
   ...

   JsrtContext * context = JsrtContext::New(runtime);

#if ENABLE_TTD
   if(inRecordMode | inReplayMode)
   {
       Js::ScriptContext* scriptContext = context->GetScriptContext();  //■■ScriptContext
       HostScriptContextCallbackFunctor callbackFunctor((FinalizableObject*)context, (void*)runtime, &OnScriptLoad_TTDCallback
   ...
   *newContext = (JsContextRef)context;
   return JsNoError;
}
```

可以看到在创建JsrtContext的时候，都回创建一个叫做ScriptContext的结构：

目录： \lib\Runtime\Base\ScriptContext.cpp

可以找到类似于SetGlobalObject()，GetGlobalObject()等成员函数，上面代码的调试模式也是通过ScriptContext设置，说明ScriptContext是JsrtContext的具体实现。

设置当前的执行上下文

目录：\lib\Jsrt\Jsrt.cpp

JsSetCurrentContext:

设置当前线程正确的script context

```
CHAKRA_API JsSetCurrentContext(_In_ JsContextRef newContext)
{
   VALIDATE_ENTER_CURRENT_THREAD(); //■■■■■■■■■■xia

   return GlobalAPIWrapper([&] (TTDRecorder& _actionEntryPopper) -> JsErrorCode {
       JsrtContext *currentContext = JsrtContext::GetCurrent();  //■■■■JsrtContext
       Recycler* recycler = currentContext != nullptr ? currentContext->GetScriptContext()->GetRecycler() : nullptr;   //■■al

#if ENABLE_TTD
       Js::ScriptContext* newScriptContext = newContext != nullptr ? static_cast<JsrtContext*>(newContext)->GetScriptContext()
       Js::ScriptContext* oldScriptContext = currentContext != nullptr ? static_cast<JsrtContext*>(currentContext)->GetScriptC

       if(newScriptContext == nullptr)
       {
           if(oldScriptContext == nullptr)
           {
               ; //■■■■ScriptContext■■■ScriptContext■■■■■■■■■
           }
           else
           {
               if(oldScriptContext->IsTTDRecordModeEnabled()) //■■TTD■■■■■■
               {
                   //already know newScriptContext != oldScriptContext so don't check again
                   if(oldScriptContext->ShouldPerformRecordAction())  //■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
                   {
                       oldScriptContext->GetThreadContext()->TTDLog->RecordJsRTSetCurrentContext(_actionEntryPopper, nullptr);
                   }

                   oldScriptContext->GetThreadContext()->TTDContext->SetActiveScriptContext(nullptr);
               }
```

```
            }
        }
        else
        {
            if(newScriptContext->IsTTDRecordModeEnabled())
            {
                if(newScriptContext != oldScriptContext && newScriptContext->ShouldPerformRecordAction())
                {
                    newScriptContext->GetThreadContext()->TTDLog->RecordJsRTSetCurrentContext(_actionEntryPopper, newScriptCont
                }

                newScriptContext->GetThreadContext()->TTDContext->SetActiveScriptContext(newScriptContext);    //■■■■■■■■■■■■
            }
        }
#endif
        //■■■■■■■■■■■
        if (currentContext && recycler->IsHeapEnumInProgress())
        {  //A heap enumeration is currently underway in the script context.
            return JsErrorHeapEnumInProgress;
        }
        else if (currentContext && currentContext->GetRuntime()->GetThreadContext()->IsInThreadServiceCallback())
        {  //■■■■■■■■■■■■■■
            return JsErrorInThreadServiceCallback;
        }

        if (!JsrtContext::TrySetCurrent((JsrtContext *)newContext))
        {  //■■■■■■■■■■■Host API■
            return JsErrorWrongThread;
        }

        return JsNoError;
    });
}
```

执行语句

目录：\lib\Jsrt\Jsrt.cpp

JsRunScript:

执行一个Script

```
CHAKRA_API JsRunScript(_In_z_ const WCHAR * script, _In_ JsSourceContext sourceContext,
   _In_z_ const WCHAR *sourceUrl, _Out_ JsValueRef * result)
{
   return RunScriptCore(script, sourceContext, sourceUrl, false,
       JsParseScriptAttributeNone, false /*isModule*/, result);
}
```

参数：

script

需要执行的script

sourceContext

标识脚本的cookie

sourceUrl

script的位置

result

存储script的结果，可以为NULL

实际上执行真正的RunScriptCore()还要经过一层：

```
JsErrorCode RunScriptCore(const char *script, JsSourceContext sourceContext,
   const char *sourceUrl, bool parseOnly, JsParseScriptAttributes parseAttributes,
   bool isSourceModule, JsValueRef *result)     //parseOnly == true;
{
   utf8::NarrowToWide url((LPCSTR)sourceUrl); //■■■■■■
   if (!url)
   {
       return JsErrorOutOfMemory;
   }

   return RunScriptCore(nullptr, reinterpret_cast<const byte*>(script), strlen(script),
       LoadScriptFlag_Utf8Source, sourceContext, url, parseOnly, parseAttributes,
       isSourceModule, result);
}

JsErrorCode RunScriptCore(const WCHAR *script, JsSourceContext sourceContext,
   const WCHAR *sourceUrl, bool parseOnly, JsParseScriptAttributes parseAttributes,
   bool isSourceModule, JsValueRef *result)
{  //NULL
   return RunScriptCore(nullptr, reinterpret_cast<const byte*>(script),
       wcslen(script) * sizeof(WCHAR),
       LoadScriptFlag_None, sourceContext, sourceUrl, parseOnly,
       parseAttributes, isSourceModule, result);
}
```

接下来是真正执行内容的RunScriptCore:

```
JsErrorCode RunScriptCore(JsValueRef scriptSource, const byte *script, size_t cb,
   LoadScriptFlag loadScriptFlag, JsSourceContext sourceContext,
   const WCHAR *sourceUrl, bool parseOnly, JsParseScriptAttributes parseAttributes,
   bool isSourceModule, JsValueRef *result)
{
   Js::JavascriptFunction *scriptFunction;
   CompileScriptException se;

   JsErrorCode errorCode = ContextAPINoScriptWrapper([&](Js::ScriptContext * scriptContext, TTDRecorder& _actionEntryPopper) -
       PARAM_NOT_NULL(script);
       PARAM_NOT_NULL(sourceUrl);

       SourceContextInfo * sourceContextInfo = scriptContext->GetSourceContextInfo(sourceContext, nullptr);

       if (sourceContextInfo == nullptr)
       {
           sourceContextInfo = scriptContext->CreateSourceContextInfo(sourceContext, sourceUrl, wcslen(sourceUrl), nullptr);
       }

       const int chsize = (loadScriptFlag & LoadScriptFlag_Utf8Source) ?
                           sizeof(utf8char_t) : sizeof(WCHAR);

       //create for every source buffer passed by host
       SRCINFO si = {
           /* sourceContextInfo   */ sourceContextInfo,
           /* dlnHost             */ 0,
           /* ulColumnHost        */ 0,
           /* lnMinHost           */ 0,
           /* ichMinHost          */ 0,
           /* ichLimHost          */ static_cast<ULONG>(cb / chsize), // OK to truncate since this is used to limit sourceText
           /* ulCharOffset        */ 0,
           /* mod                 */ kmodGlobal,
           /* grfsi               */ 0
       };

       Js::Utf8SourceInfo* utf8SourceInfo = nullptr;
       if (result != nullptr)
       {
           loadScriptFlag = (LoadScriptFlag)(loadScriptFlag | LoadScriptFlag_Expression);
       }
       bool isLibraryCode = (parseAttributes & JsParseScriptAttributeLibraryCode) == JsParseScriptAttributeLibraryCode;
       if (isLibraryCode)
       {
            loadScriptFlag = (LoadScriptFlag)(loadScriptFlag | LoadScriptFlag_LibraryCode);
        }
        if (isSourceModule)
        {
            loadScriptFlag = (LoadScriptFlag)(loadScriptFlag | LoadScriptFlag_Module);
        }

#if ENABLE_TTD

                                             ...
#endif

       //loadScript■■■■■■■■■■■■■■■■■■LoadScriptInternal■■■■■■■■■ParseTree■■
        scriptFunction = scriptContext->LoadScript(script, cb,
            &si, &se, &utf8SourceInfo,
            Js::Constants::GlobalCode, loadScriptFlag, scriptSource);

#if ENABLE_TTD
                                                ...
#endif

                                                ...
//■■■■■■■■
#ifdef ENABLE_DEBUG_CONFIG_OPTIONS
                                                 ...
#endif

#if ENABLE_TTD
                                                 ...
#endif
             //■■
            Js::Var varResult = scriptFunction->CallRootFunction(args, scriptContext, true);
            if (result != nullptr)
            {
                *result = varResult;
            }

#if ENABLE_TTD
                                                ...
#endif
        }
        return JsNoError;
    });
}
```

JsConvertValueToString：

使用标准JavaScript语义将值转换为字符串。

```
CHAKRA_API JsConvertValueToString(_In_ JsValueRef value, _Out_ JsValueRef *result)
{
   PARAM_NOT_NULL(result);
   *result = nullptr;

   if (value != nullptr && Js::JavascriptString::Is(value))
   {
        return ContextAPINoScriptWrapper([&](Js::ScriptContext *scriptContext, TTDRecorder& _actionEntryPopper) -> JsErrorCode
            PERFORM_JSRT_TTD_RECORD_ACTION(scriptContext, RecordJsRTVarToStringConversion, (Js::Var)value);
            VALIDATE_INCOMING_REFERENCE(value, scriptContext);

            *result = value;

            PERFORM_JSRT_TTD_RECORD_ACTION_RESULT(scriptContext, result);

            return JsNoError;
        });
   }
                                                 ...
}
```

将脚本返回为C++

JsStringToPointer：

检索字符串值的字符串指针。

```
CHAKRA_API JsStringToPointer(_In_ JsValueRef stringValue, _Outptr_result_buffer_(*stringLength) const WCHAR **stringPtr, _Out_
{
   VALIDATE_JSREF(stringValue);
   PARAM_NOT_NULL(stringPtr);
   *stringPtr = nullptr;
   PARAM_NOT_NULL(stringLength);
   *stringLength = 0;

   if (!Js::JavascriptString::Is(stringValue))
   {
       return JsErrorInvalidArgument;
   }

   return GlobalAPIWrapper_NoRecord([&]() -> JsErrorCode {
       Js::JavascriptString *jsString = Js::JavascriptString::FromVar(stringValue);

       *stringPtr = jsString->GetSz();
       *stringLength = jsString->GetLength();
       return JsNoError;
   });
}
```

处理runtime

JsDisposeRuntime:

```
CHAKRA_API JsDisposeRuntime(_In_ JsRuntimeHandle runtimeHandle)
{
   return GlobalAPIWrapper_NoRecord([&] () -> JsErrorCode {
       VALIDATE_INCOMING_RUNTIME_HANDLE(runtimeHandle);

       JsrtRuntime * runtime = JsrtRuntime::FromHandle(runtimeHandle);
       ThreadContext * threadContext = runtime->GetThreadContext();
       ThreadContextScope scope(threadContext);

       // We should not dispose if the runtime is being used.
       if (!scope.IsValid() ||
           scope.WasInUse() ||
           (threadContext->GetRecycler() && threadContext->GetRecycler()->IsHeapEnumInProgress()))
       {
           return JsErrorRuntimeInUse;
       }
       else if (threadContext->IsInThreadServiceCallback())
       {
           return JsErrorInThreadServiceCallback;
       }

       // Invoke and clear the callbacks while the contexts and runtime are still available
       {
           Recycler* recycler = threadContext->GetRecycler();
           if (recycler != nullptr)
           {
               recycler->ClearObjectBeforeCollectCallbacks();
           }
       }
#ifdef ENABLE_SCRIPT_DEBUGGING
                                        ...
#endif
       Js::ScriptContext *scriptContext;
       for (scriptContext = threadContext->GetScriptContextList(); scriptContext; scriptContext = scriptContext->next)
       {
#ifdef ENABLE_SCRIPT_DEBUGGING
                                       ...
#endif
            scriptContext->MarkForClose();
        }

        // Close any open Contexts.
        // We need to do this before recycler shutdown, because ScriptEngine->Close won't work then.
        runtime->CloseContexts();

#ifdef ENABLE_SCRIPT_DEBUGGING
                                        ...
#endif

#if defined(CHECK_MEMORY_LEAK) || defined(LEAK_REPORT)
        bool doFinalGC = false;

#if defined(LEAK_REPORT)
        if (Js::Configuration::Global.flags.IsEnabled(Js::LeakReportFlag))
        {
            doFinalGC = true;
        }
#endif

#if defined(CHECK_MEMORY_LEAK)
        if (Js::Configuration::Global.flags.CheckMemoryLeak)
        {
            doFinalGC = true;
        }
#endif

        if (doFinalGC)
        {
            Recycler *recycler = threadContext->GetRecycler();
            if (recycler)
            {
                recycler->EnsureNotCollecting();
                recycler->CollectNow<CollectNowFinalGC>();
                Assert(!recycler->CollectionInProgress());
            }
        }
#endif
        //■■■■■free■■
        runtime->SetBeforeCollectCallback(nullptr, nullptr);
        threadContext->CloseForJSRT();
        HeapDelete(threadContext);

        HeapDelete(runtime);

        scope.Invalidate();

        return JsNoError;
    });
}
```

参考链接：

- https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey/JSAPI_reference/JSRuntime
- https://github.com/Microsoft/ChakraCore/wiki/JavaScript-Runtime-%28JSRT%29-Overview
- https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/time-travel-debugging-overview

结语

对源码的简单分析和阅读，有些解释直接就丢在代码里了，不对之处望大佬们指出。

点击收藏 | 0 关注 | 1
上一篇：Pwn2Own 2018 Safa... 下一篇：[红日安全]代码审计Day1 - ...

1. 0 条回复

- 动动手指，沙发就是你的了！