SpiderMonkey 를 이용한 개발 일기

정 원교

2006.6.9

서울 하늘 아래서

email : weongyo@gmail.org

요 약

이 문서는 제가 SpiderMonkey 를 이용하여 HTML 문서 Parser 를 만들면서 격었던 일들에 대해서 기록한 문서입니다. 이 문서를 통해서 어떻게 SpiderMonkey 를 build 하고 사용하였는지 등등에 대해서 기술할 것입니다. 그리고 이 문서를 계기로 국내에 많은 SpiderMonkey 개발자가 생겼으며 좋겠으며 많이 알려지고 사용되어졌음 하는 바램이 있습니다.

차 례

제 1 절 SpiderMonkey 란?

SpiderMonkey 는 C 언어로 작성된 JavaScript 처리 엔진입니다. Mozilla 재단에 의해 관리 되고 있으며, Netscape 와 Firefox 의 JavaScript 처리를 맡고 있습니다.

즉, 웹 페이지에 존재하는 JavaScript 를 해석하고 compile 해서 사용자가 입력하는 이벤트를 처리하거나 실행하게 됩니다.

1.1 하지만...

Embed 되어지는 SpiderMonkey 는 브라우저와는 전혀 상관없는 존재입니다. 단지 JavaScript 문법을 따르고 있을 뿐입니다. JavaScript 라면 생각나는 document 오브젝트나 window 오브젝트 조차 선언되어 있지 않습니다. 매우 깨끗한 상태이기 때문에 여러분이 SpiderMonkey 엔진을 어떻게 사용할 것인가에 따라 매우 색상이 달라질 수 있습니다.

embedding language 라고 해도 과언이 아닙니다.

제 2 절 이 문서가 포함하고 있는 내용은?

영문으로 작성된 SpiderMonkey 사용 문서에 대해서는 공식 홈페이지 http://www.mozilla.org/js/spidermonkey/ 에 방문하셔서 보실 수 있습니다.

이 문서는 제가 SpiderMonkey 를 이용하여 프로그램에 들어갈 컴포넌트를 개발하면서 직면했던 여러가지 문제점이나 삽질기 등등의 경우에 대해서 기술할 예정이며 spidermonkey 와 관련된 이야기로만 구성할 것입니다.

SpiderMonkey 를 이용하여 자신의 혹은 프로젝트의 프로그램에 embed 를 고려 중인 분들에게 조금이나마 도움이 되었으면 합니다.

제 3 절 나는 무엇을 개발하였는가?

세부적인 이야기를 언급하기 이전에 제가 SpiderMonkey 를 이용하여 어떤 것을 개발하였는지에 대해서 말하는 것이 여러분들이 이 글을 읽으면서 앞으로 어떤 이야기들이 어떻게 나올지 예상할 수 있게 할 것입니다.

제가 맡은 업무는 HTML 문서를 해석하여 그 내부에 포함되어 있는 URL link 를 추출하는 이야기로는 간단한? 일이었습니다.

URL Link 를 성공적으로 추출하기 위해서는 아래와 같은 일들을 parser 가 제대로 처리할 수 있어야 했습니다.

- HTML 문서를 제대로 처리할 수 있어야 했습니다. 각각의 tag 및 그 내부에 선언되어 질 수 있는 attribute 들에 대해서 유연하게 가져올 수 있어야 했습니다. a 태그나 img 태그 내부 혹은 가운데 존재할 수 있는 URL link 를 가져오는 것이 저의 일이었기 때문입니다.
- JavaScript 를 완전하게 처리할 수 있어야 했습니다. 외국 사이트의 경우, JavaScript 를 이용한 링크 처리는 거의 하지 않는 반면에, 한국 사이트의 경우, JavaScript 를 이용하여 link 를 거는 행위가 매우 빈번했기 때문입니다. 즉, 애플리케이션이 제대로 된 URL link 를 가져오기 위해서는 그리고 한국 사이트를 위해서는 반드시 JavaScript 처리는 필수 였습니다.
- Flash 파일 내부에 존재할 수 있는 URL link 혹은 JavaScript 구문을 처리할 수 있어야 했습니다. HTML 문서상에서 보일 수 있는 link 수집이 주 목적이었기 때문에 현재 많이 사용되고 있는 SWF 파일 또한 예외가 아니었습니다. 하지만 이 부분은 이 문서의 범위를 넘어서기 때문에 더 이상 언급이 안될 것입니다.

단순히 HTML Parser 와 JavaScript 엔진을 같이 사용한다고 모든 일이 끝나는 것이 아니었습니다. HTML 문서와 JavaScript 가 잘 조합되어 처리가 될려면 아래와 같은 사항들을 고려해 놓아야 하기 때문입니다.

- HTML 의 DOM (Document Object Model) 을 구성해 주어야 했습니다. 그렇지 않으면 JavaScript 코드내에서 exception 이 발생하게 됩니다. HTML 과 JavaScript 가 어떻게 연동되며 브라우저 상에서 행동하게 되는지에 대한 지식이 필요합니다. 예를 들어, frame 태그를 이용한 화면이 있다고 가정을 해봅시다. 각각의 프레임마다 이름이 있으며 이 프레임 윈도우마다 많은 내부 DOM Tree 가 존재할 것입니다. 이러한 구성이 제대로 작동하지 않을 경우, URL link 추출은 실패로 끝나게 되는 것입니다.
- URL link 를 성공적으로 추출하기 위해서는 마치 브라우저에서 하나 하나 클릭한 것과 같은 효과를 주어야 하기 때문에, 브라우저에서 지원하는 JavaScript Object 들을 가상적으로 만들어져야 하며 Internet Explorer 와 Firefox 가 지원하는 Object 들 중에는 공통적인 것들도 있지만 Internet Explorer 에만 있는 Object 들도 있기 때문에 이에 대한 고려도 해주어야 했습니다.

제 4 절 왜 SpiderMonkey 를 선택하였는가?

처음 HTML 문서에서 URL link 를 제대로 추출할 수 있는 라이브러리를 작성해야 한다고 생각했을 때, 조금은 난감하였습니다. 처리되는 속도를 우선적으로 고려해야 하고, 디버깅이 용이해야 할 것 같고, 시간이 그렇게 넉넉한 것도 아니고, 가장 중요한 것으로는 아직 HTML + JavaScript 파서를 개발한 경력이 없기 때문이었습니다. 무엇부터

시작해야 할지 모를 때, Google 를 이용한 자료 조사를 통해서 나름대로의 배경 지식을 쌓게 되었으며, 생각보다 그렇게 쉬운 일이 아님을 깨닳게 되었습니다.

저는 윈도우즈 개발자가 아니었기 때문에 마이크로소프트사에서 제공하는 라이브러리 혹은 컴포넌트를 이용한 개발에는 익숙하지 않았으며 Visual Studio 를 이용한 개발보다 gcc + emacs + linux 가 더욱 더 익숙했기 때문에 나름대로 unix 맛이 나는 JavaScript 엔진을 구해야 할 것 같다는 생각을 하였습니다.

"JavaScript 엔진을 이용한 개발"을 생각하지 못했던 시절에는 Internet Explorer Browser 컴포넌트 (윈도우즈 개발에 익숙지 못하기 때문에, 표현을 이렇게 밖에 못해 죄송합니다.) 를 사용하여 각각의 중요 지점을 hooking 을 해서 사용하도록 하는 의견이 있었습니다. 외국 타사의 제품이 이를 통해서 구현된 사례가 있었기 때문입니다. 하지만 HTML 을 처리하는 속도가 느렸으며 무엇인가 완전해 보이지 않았으며 왠지 꽁수를 사용하는 듯한 기분이 들었습니다. 중요한 것은 개발을 해야 할 제가 전혀 지식이 없다는 것이었습니다.

위 내용 다음의 의견이 (어떻게 만들어야 할지 모를 때) Gecko 를 이용한 개발 방법이었습니다. Layout Engine 이라는 개념이 무엇인지 모를 때 단순히 DOM 이라는 것을 표현해 준다는 말만 듣고 나름대로 분석해 보았던 엔진입니다. 하지만 저의 미천한 C++ 실력과 국내 자료의 부족 그리고 위에서 언급했던 IE Browser 컴포넌트를 사용하는 것과 다름없는 행위를 해야 한다는 결론에 도달하였을 때 포기를 하게 되었습니다.

그리고 마지막으로 Gecko 를 통해서 알게된 SpiderMonkey 를 사용하기로 했습니다. 비록 브라우저가 지원해야 Object 들을 모두 작성해 주어야 제대로 돌아갈 수 있다는 단점이 있었지만 아래와 같은 장점이 있었습니다.

- Gecko 보다 작은 범위의 프로젝트였기 때문에 컴퓨터 실력이 미천한 저로써는 감사할 따름이었습니다. 프로젝트의 규모가 크면 클수록 분석하는데 걸리는 시간이 커지는 것은 자명한 것이며, 이 프로젝트의 전체 나무를 볼 수 있을 것 같은 자신감을 반비례해서 작아지게 되더군요.
- C 언어로 작성되어 있었기 때문에, C++ 에는 아직 서툰 저에겐 안성맞춤이었으며 디버깅과 분석이 좀 더 쉬워 보였습니다. 물론 C 로 작성되어 있어서 전체 나무를 보지 못하면 전혀 분석 안될 때도 있는데, 바로 SpiderMonkey 도 이 경우 였습니다. 내가 가장 잘 할 수 있다는 C 언어 코드를 보고도 전체가 이해 안될 때의 좌절감은 사람을 "내가 왜 컴퓨터를 하고 있나?"라는 생각에 도달하게 해줍니다.
- 작성 완료된 라이브러리는 프로그램의 한 부분으로 들어가야 하는데, 여러 thread 가 동시에 라이브러리를 load 하고 사용하기 때문에 반드시 thread-safe 해야 했습니다.
- 메모리 leak 이 절대로 있으면 안되었습니다. 프로그램의 특성상 몇 시간 혹은 몇 일을 계속 작업을 수행할 수 있기 때문에 라이브러리 내에 포함된 루틴의 메모리 leak 으로 프로그램이 죽을 수는 없는 일이었습니다.
- 믿음직해야 했습니다. 이 세상에는 정말 대단한? 웹 프로그래머들이 많아서 도대체 왜 이렇게 HTML 문서 및 JavaScript 들의 구성을 신기하게 해 놓았는지 못 믿겨질 때가 매우 간혹 있습니다. 프로그램 전체의 안정성을 위해서는 라이브러리의 안정성이 전제되어야 했기 때문에 정말 신기하게 짜놓은 그리고 exception 일어날 것만 같은 코드에도 exception 이 일어나지 않는 JavaScript 엔진이 필요했습니다. SpiderMonkey 의 경우, firefox 의 기본 JavaScript 엔진이며 전세계 많은 사람들로 부터 feedback 을 받았을 거라 믿었기 때문에 안심하고 사용할 수 있었습니다.

해당 라이브러리에 대한 개발이 완료된 이 시점에서 그 동안 SpiderMonkey 의 안정성에 대해 돌이켜 생각해 보면, "대단하다"라는 것입니다. 지금까지 수많은 웹 페이지들을 제가 작성한 라이브러리를 사용하여 분석해 보았지만 SpiderMonkey 자체의 문제는 한번도 만난 적이 없었습니다.

오히려 저의 SpiderMonkey 에 대한 이해 부족 및 잘못된 사용으로 발생하는 exception 혹은 segment fault 였던 것입니다.

제 5 절 Build 하기

아래에서는 SpiderMonkey 를 Build 위해서 사용했던 방법에 대해서 설명하도록 하겠습니다.

5.1 개발 환경

프로젝트로 개발되던 프로그램의 경우, Win32 환경에서 돌아가야 했기 때문에 원하는 라이브러리의 형태는 반드시 DLL 이여야 했습니다. 윈도우즈 환경에서의 개발이 익숙하지 않았던 저로써는 최대한 unix 환경같이 DLL 을 만들 수 있는 쪽으로 머리를 써야 했었습니다.

윈도우즈에서 unix 와 같은 개발 환경을 만들 수 있는 패키지로는 제가 알기로 두 가지가 있었는데, cygwin 과 MinGW 였습니다.

처음에는 cygwin 을 이용하여 개발을 하면 될 것 같다는 막연한 생각으로 설치하고 설정하고 build 하였지만 돈을 지불하지 않고 배포할 경우 cygwin.dll 과 같은 파일을 같이 배포해야 한다는 부분과 만들어진 최종 결과물 dll 이 프로젝트 프로그램 (Delphi 로 제작되었습니다.) 제대로 동작한다는 보장이 없었기 때문에, 그리고 unix system call 을 가상으로 흉내내준다는 그 부분이 조금은 불안 불안해 보였습니다. 즉, 왠지 깔끔하게 보이지 않았던 느낌이 cygwin 대신 MinGW 를 사용하게 된 원인이라고 생각됩니다.

마지막으로 생각했던 부분이 MinGW 였습니다. msys 라는 unix 같은 shell 환경을 제공해 주었고 의존적인 library 도 없었기 때문에 cygwin 보다 깔끔해 보였습니다. 하지만 MinGW 를 쓰기로 마음 먹은 그 때에는 안정성 및 호환성에 대해서는 신뢰가 아직은 없었던 때입니다.

automake 와 autoconf 를 이용하여 build 하였었는데, 가장 황당했던 기억은 output DLL 파일 이름이 확장자 .dll 이 아니라 .exe 로 만들어졌던 것입니다. 이게 정말 제대로 build 된 DLL 파일인지 제대로 load 되어질 수 있는 DLL 인지 확신할 수 없었기 때문에 처음엔 대단히 당황했었습니다. 몇 번의 테스트를 거치고 파일 포맷을 분석하고 난 후에야 겨우 믿게 된 것입니다.

이 이후로는 계속 .exe 를 .dll 로 이름을 바꿔줬던 삽질을 하게 되었습니다.

5.2 MinGW 를 이용하여 build 하는 방법

MinGW 를 이용한 SpiderMonkey build 에 대해서는 google 검색을 통해서 쉽게 찾으실 수 있습니다.

제가 찾은 곳은 redwiki.net 사이트에 있는 build 방법입니다. 아래의 주소에서 자세한 내용을 확인할 수 있습니다.

```
http://www.redwiki.net/wiki/wiki.php/spidermonkey/MinGW빌드하기
```

조심해야 할 부분은 “주의사항”에 나와 있는 컴파일 옵션 부분입니다. 그리고 만약 SpiderMonkey 를 thread-safe 해야 하는 곳에 embed 해서 사용해야 할 경우, 반드시 JS_THREADSAFE 를 define 해주셔야 합니다. 즉, 아래와

같이 옵션을 추가해 주셔야 합니다.

```
CFLAGS = -O2 -DXP_WIN -DEXPORT_JS_API -D_WINDOWS -DWIN32 -D_MINGW \
         -DJS_THREADSAFE -s
```

위 옵션을 주지 않으면 당연히 segment fault 를 여러분의 프로그램에서 만나실 수 있을 것입니다.

자세한 thread-safe 형태의 JavaScript 엔진을 build 하기 위해서는 아래의 5.4 절을 반드시 읽으시기 바랍니다.

5.3 Visual Studio 를 이용하여 build 하는 방법

이 방법은 매우 쉬울 것입니다. 제가 알기로는 현재 SpiderMonkey 배포판에는 Visual Studio 를 위한 "프로젝트 파일"이 이미 존재하기 때문에 단순히 프로젝트를 열어서 build 하시면 바로 Debug/Release 바이너리를 쉽게 얻으실 수 있습니다.

위 섹션에서 언급하였듯이 JS_THREADSAFE 옵션이 필요할 경우, 꼭 설정을 변경하여 build 하시기 바랍니다.

5.4 Thread-Safe 하게 build 하는 방법

SpiderMonkey 를 thread-safe 하게 build 하기 위해서는 Netscape Portable Runtime (NSPR) 라이브러리가 반드시 필요합니다. Locking 부분을 위해서 사용되는 것이 주 목적으로 알고 있습니다.

NSPR 을 사용하지 않고 SpiderMonkey 를 thread-safe 하게 만드는 방법은 없으며 아래와 같이 함으로써 힘들지 않게 build 하실 수 있을 것입니다.

1. http://www.mozilla.org/projects/nspr/ 에 방문하시면 NSPR 의 최신 정보를 얻으실 수 있습니다.
2. NSPR 의 경우, 바이너리와 소스를 같이 배포하기 때문에 자신의 시스템에 맞는 바이너리를 받으시기 바랍니다.
3. 압축을 푸시면 include 디렉토리와 lib 디렉토리가 존재하는데, include 디렉토리의 경우 SpiderMonkey 의 compile option 에 포함되도록 하시고 link 하실 때는 lib 디렉토리를 참조할 수 있도록 설정을 변경해 주시기 바랍니다. 그럼 어렵지 않게 thread-safe 한 바이너리를 얻으실 수 있을 것입니다.

제 6 절 Link 하기

5 절에서 보았던 방법으로 성공적으로 build 를 하였다면 이제 프로그램에 link 를 거는 것이 목표입니다. Link 를 성공적으로 걸기 위해서는 3 가지 요소가 필요하다고 할 수 있는데,

- SpiderMonkey 의 include 파일들
- js32.lib 파일 (SpiderMonkey 를 build 하면 같이 생성되는 파일들 중 하나)

- js32.dll 파일

만약 thread-safe 하게 build 를 하였다면 libnspr4.lib 파일과 libnspr4.dll 파일, 마지막으로 NSPR 라이브러리의 include 파일들이 별도로 필요할 것입니다.

저의 개발 환경은 위에서 말했듯이 MinGW + automake + autoconf 를 이용하여 DLL 을 만들어 사용하였기 때문에 아래에서는 Visual Studio 를 이용한 방법에 대해서 언급하지 못함을 알려 드립니다. (아래에서 언급되는 방법은 정상적인 방법이 아닐수도 있으며 개인적인 경험을 바탕으로 기술되었음을 미리 말해드립니다. 그리고 설명이 틀릴수도 있습니다. 추가/수정/삭제 의견이 있으시면 저의 개인메일로 연락 바랍니다.)

autoconf 를 이용한 작업의 경우, configure.in 파일 작성이 제일 중요한데 DLL 을 작성하기 위해서는 아래의 옵션을 주어야 합니다.

```
AC_LIBTOOL_WIN32_DLL
```

나머지 Makefile.am 혹은 Makefile.in 파일의 구성은 기존 unix shared library 구성과 거의 같지만 오직 하나만 다른 점이 있습니다. 그것은 Windows 용 DLL 의 경우, undefined symbol 을 허용하지 않는다는 것입니다. 즉 DLL 파일에 선언되는 함수들은 모두 서로 유기적으로 연결되어져야 한다는 뜻입니다. 그래서 LDFLAGS 에 반드시 아래의 옵션을 선언해 주시기 바랍니다.

```
-no-undefined
```

그리고 작성하신 라이브러리에서 선언하고 사용하셨던 함수들이 모두 제대로 link 되도록 하시기 바랍니다.

마지막으로 NSPR DLL (libnspr4.lib) 파일과 SpiderMonkey DLL (js32.lib) 파일을 링크하는 부분인데, 저의 경험으로 미루어보면 이 부분이 가장 헤맷던 부분으로 기억이 됩니다. 제 경우, Makefile.am 파일을 작성하여 Makefile.in 으로 변환하여 DLL 작성을 하였지만 Makefile.am 파일에 해당 파일들에 대한 링크를 걸어줄 경우, configure 스크립트의 문제로 인한 것인지 libtool 스크립트의 문제로 인한 것인지 확신할 수 없지만 항상 경고가 발생하고 linking 이 제대로 안되는 것이었습니다.

이를 해결하기 위해서 configure 스크립트나 libtool 스크립트의 이곳 저곳을 고쳐보았지만 적절한 해결책을 찾지 못하였습니다.

그래서 마지막으로 사용했던 방법이 libtool 스크립트를 수정하여 오류나는 부분 (항상 마지막 linking 쪽에서의 문제였습니다.) 에 아래와 같이 수동으로 link 할 부분들을 정의해 주었습니다.

```
-mconsole -mwindows -mole -lole32 -lws2_32 -lstdc++ -lodbc32
-loleaut32 -lmapi32 -lcomdlg32 -luuid
../lib/libnspr4.lib ../lib/js32.lib
```

위의 옵션들을 넣어줘야 할 부분에 대해서 조금 더 설명을 한다면 아래와 같이 찾으실 수 있을 거라 믿습니다.

- libtool 을 editor 로 열고 문자열 “1000000” 가 처음 발경되는 곳입니다. 변수 archive_cmds 내부에 위의 옵션들을 넣으시면 될 것이라 믿습니다.
- libtool 스크립트에 따라 조금씩 내용이 다를 수 있기 때문에 확신할 수는 없지만 제가 사용했던 libtool 파일의 경우, 205 번째 줄이 수정했던 곳입니다.

위와 같이 함으로써 저는 제가 작성중이었던 라이브러리에 SpiderMonkey 를 thread-safe 한 바이너리로 linking 할 수 있었습니다.

제 7 절 본격적인 프로그래밍

이제 조금씩 SpiderMonkey 를 사용하는 방법에 대해서 알아 보도록 하겠습니다. 기본적인 라이브러리의 사용법에 대해서는 모질라 공식 홈페이지를 방문하시면 쉽게 얻어실 수 있습니다. 하지만 처음 사용하시는 분들에 대한 guide 만 존재하지 자세한 설명은 없을 것입니다.

가장 자세한 사용법이 소개되어 있는 것은 아마도 SpiderMonkey 를 JavaScript 엔진으로 사용한 프로그램일 것입니다. 대표적인 프로그램이 Firefox 이며 이 소스를 분석하는 것이 가장 구하기 쉬우나 가장 분석하기 어려운 방법일 것입니다. Firefox 소스의 크기 및 컴파일 시간 (보통 1 2시간 넘습니다.) 그리고 분석할 수 있을 것이라는 자신감은 분석을 하면 할수록 점점 사라져 갈 것이 분명합니다.

만약 친철하게 SpiderMonkey 홈페이지에 open-source 로 공개된 SpiderMonkey 엔진을 도입한 application 목록이 잘 정리되어 있다면 좋겠지만 홈페이지에서는 참고할 예제들이 넉넉하지 않은 것이 현실입니다.

예제를 얻는 가장 좋은 방법은 SpiderMonkey 라이브러리의 함수 이름을 google 에서 찾아가면서 조금씩 이해하는 것입니다. 하지만 대부분의 SpiderMonkey 자료들이 영어로 작성되어 있다는 사실을 알기 바랍니다.

이 문서를 작성하는 이유 중의 하나가 SpiderMonkey 에 대한 간단한 소개글은 있지만 제대로 된 사용기가 없어 많은 개발자들이 접근하기 힘들게 생각하거나 좋은 라이브러리를 사용할 기회를 놓치게 되는 것을 안타깝게 생각했기 때문입니다. 어쩌다가 저의 일이 되어 나름대로 SpiderMonkey 를 사용해 보고 그 경험을 여러분들과 공유하고자 하니 꼭 여러분들께 도움이 되었으면 좋겠습니다.

다만 기본적인 SpiderMonkey 에 대한 자료는 많기 때문에, 그 부분에 대해서는 인터넷 상에서 다른 문서들을 참고하셔서 익히시기 바랍니다. 이 문서에서는 제가 격었던 나름대로의 고생들을 정리해서 기술할 생각입니다.

7.1 처음 시작하기

SpiderMonkey 를 처음 접하시는 분들의 첫번째 출발점은 바로 공식 홈페이지에 있는 문서들입니다. 그곳에서 볼 수 있는 “JS Embedder’s Guide”는 대략적인 개념을 잡는데 유용한 문서입니다. 이 문서를 통해서 SpiderMonkey 가 전체적으로 어떻게 수행되는지 보실 수 있을 것입니다.

```
http://www.mozilla.org/js/spidermonkey/apidoc/jsguide.html
```

그럼 guide 를 끝내고 봐야 할 것은 무엇일까요? 아마 실제로 프로그래밍을 해야하실 분들은 실제 사용 예제일 것입니다. “JS Embedder’s Guide”의 아쉬운 점은 예제가 상세하게 표현되어있지 않다는 것입니다. 하지만 홈페이지에 있는 다른 문서들 또한 상황은 마찬가지 인 것 같습니다. 그 중 가장 도움이 되는 문서는 “JS Embedder’s Reference” 입니다. 그 이유는 각각의 함수들의 역활과 인자들, 반환값에 대해서 자세히 기술되어 있기 때문입니다. 그렇지만 전혀 예제가 포함되어 있지 않기 때문에 처음 접하시는 분들과 영어가 익숙하지 않는 분들은 조금 헤맬 것 같네요.

저의 경우, “JS Embedder’s Guide” 를 읽고도 개념이 제대로 잡히지 않아 여러분 읽었습니다. 그리고 google 에서 실제 함수들을 대상으로 검색하여 예제가 어떻게 돌아가는지 살펴보곤 했습니다. 지금은 SpiderMonkey 가 얼마나 사용되고 있는지 모르겠지만 그 때 (1년 혹은 2 년전?) 는 google 에서 나온 결과 조차 쓸만한게 거의 없어 고생을 했었습니다. 그런 경험 때문에 저에게 가장 유용했던 문서는 “JS Embedder’s Reference” 이었음은 다시 강조해도 부족하지 않습니다.

7.2 SpiderMonkey 가 기본으로 가지고 있는 능력

JavaScript 엔진인 SpiderMonkey 는 JavaScript 문법을 기본적으로 지원합니다. JS 1.0 부터 JS 1.5 까지 지원하고 있는 것으로 알고 있으며 JS 1.3 이후 버전부터는 ECMAScript-262 와 호환되고 있음을 알고 있습니다.

그리고 최소한의 object 들을 가지고 있습니다. 이것을 built-in 오브젝트라고 하는데, Array 와 Boolean, Date, Math, Number, String 가 그것입니다.

이 외에는 전혀 아무것도 선언되어 있지 않다는 것을 말해 드립니다. 이제 SpiderMonkey 에서 정의되어 있는 함수들을 사용하여 새로운 오브젝트를 만들어야 합니다. 이 오브젝트를 어떻게 구성하느냐에 따라 사용에 대한 색깔이 달라지게 될 것입니다.

7.3 Runtime 과 Context

SpiderMonkey 를 이용하여 코딩을 할 때, 처음 겪게 되는 과정입니다. 함수 `JS_NewRuntime ()` 와 `JS_NewContext ()` 를 이용하여 결과적으로 Object 들이 들어갈 Context 를 만드는 부분입니다.

엔진에서 사용되는 개념의 포괄적인 면을 보았을 때, 아래와 같이 표현할 수 있을 듯 합니다.

```
Runtime > Context > Object > Function 혹은 Property
```

Runtime 을 제외한 나머지 부분들은 모두 상위 개념에 여러 개가 들어갈 수 있습니다. 예를 들면, 여러 개의 Context 가 Runtime 내부에 존재할 수 있다는 말이며, Object 또한 Context 에 여러 개가 존재할 수 있습니다.

하지만 Object 가 Runtime 에 들어간다거나 Function 이 Context 에 들어가거나 할 수는 없습니다.

```
JSRuntime *rt;
JSContext *cx;
size_t iRuntimeSize = 1024 * 1024 * 8;
size_t gStackChunkSize = 1024*8;

rt = JS_NewRuntime (iRuntimeSize);
if (!(rt))
  return -1;

cx = JS_NewContext (rt, gStackChunkSize);
if (!(cx))
  return -1;

JS_SetErrorReporter (cx, my_ErrorReporter);
```

위의 구문이 가장 기본적인 시작입니다. (함수들이 모두 메모리를 할당하여 자료 구조를 유지하기 때문에 항상 메모리 해제를 염두에 두시고 개발하시기 바랍니다. 위의 경우 각각 메모리 해제 함수들이 존재합니다.) `JS_NewRuntime ()` 의 첫번째 인자는 Garbage Collection 이 실행할 최소 메모리 할당량을 가르킵니다. 각각의 상속 관계를 잘 살펴보시기 바랍니다.

마지막으로 `JS_SetErrorReporter ()` 함수는 디버깅을 위해서 매우 중요한 함수입니다. 처음해 보는 개발 조건과 환경에서 가장 도움이 되는 존재일 것입니다. 자세한 내용은 8.3 절 “JS exception 처리하기” 를 보시기 바랍니다.

7.4 Private 값 설정하기

pthread 를 이용하여 개발을 해 본 사람이라면 한번쯤은 경험해 봤을 private value 넘기기 부분입니다. `JS_SetContextPrivate ()` 함수를 이용하여 Context 에 넘길 값을 설정하게 됩니다. 그리고 `JS_GetContextPrivate ()` 함수를 이용하여 받아 오게 됩니다.

Context 에 들어가게 되는 Object 들과 Function, Property 들은 공통적으로 갖는 특징이 있는데, 모두 JSContext 변수를 인자로 받는 다는 뜻입니다. 즉, `JS_SetContextPrivate ()` 에서 설정한 값은 Object 나 Function, Property 모든 곳에서 접근할 수 있다는 뜻을 의미합니다. 그렇기 때문에 설정할 값을 신중하게 선택하는 것이 바람직 합니다.

저의 경우, thread-safe 하게 프로그래밍을 하기 위해서 하나의 구조체에 모든 정보를 관리하였기 때문에, 그 구조체에 대한 포인터를 넘김으로써 모든 정보에 접근할 수 있었습니다.

7.5 Object 만들기

이제 Context 에 하나의 Object 를 만드는 부분에 대해서 알아 보도록 하겠습니다. Object 에는 두가지가 있는데, 하나는 일반적인 Object 이고 나머지는 Array Object 입니다.

일반적인 새로운 Object 를 만들 때는 JS_NewObject () 함수와 JS_DefineObject () 함수를 이용하여 만들어지게 됩니다. 첫번째, JS_NewObject () 함수의 경우, Context 내에 처음 Object 를 만들 때 사용되었으며, 이미 존재하는 Object 의 Property 로써 새로운 Object 를 연결시킬 때는 JS_DefineObject () 함수를 사용하였습니다.

만약 Array Object 를 만들고자 한다면 JS_NewArrayObject () 함수를 사용할 수 있습니다. 각각의 element 들을 설정할 수 있도록 하는 JS_SetElement () 나 기타 JS_...Element () 함수들의 사용법에 대해서 알기 바랍니다.

Object 는 Class 와 개념이 조금은 다른 것임을 알아야 합니다. 이름이 ActiveX 라는 class 를 만들었다면 아래의 구문과 같이 변수내에 Class 의 Object 를 초기화하여 할당할 수 있습니다.

```
var a = new ActiveX ("UUID");
```

하지만 위에서 언급된 함수들에 의해 만들어지는 Object 는 초기화할 때, Function 과 Property 를 정해 놓고 만들기 때문에 위의 예제처럼 동적으로 초기화하여 할당하거나 그럴수는 없습니다.

7.6 Function 과 Property

Class 혹은 Object 를 선언할 때, 내부에 선언되어질 Property 혹은 Function 을 정의할 수 있습니다. 이러한 과정은 위 7.5 절에서 만들어진 Object 를 기반으로 선언되어지게 되는데, 구조체 JSClass 와 JSFunctionSpec, JSPropertySpec 를 선언하면서 이루어지게 됩니다.

만들어진 Object 를 대상으로 JS_DefineProperties () 함수와 JS_DefineFunctions () 함수를 이용하여 선언되어지게 됩니다.

아래는 Property 의 값에 변동이 생겼을 때, 호출되게 되는 callback prototype 입니다.

```
JSBool
propertyname (cx, obj, id, val)
     JSContext *cx;
     JSObject *obj;
     jsval id, *val;
```

아래는 Function 이 호출되었을 때, 자동으로 호출되게 되는 callback prototype 입니다.

```
JSBool
funcname (cx, obj, argc, argv, rval)
     JSContext *cx;
     JSObject *obj;
     uintN argc;
     jsval *argv, *rval;
```

위의 인자 선언에서 알 수 있듯이, 모두 공통적으로 JSContext 와 JSObject 를 넘기는 것을 알 수 있습니다. 앞에서 언급하였듯이 사용자가 선언한 private value 를 JSContext 를 통해서 얻으실 수 있습니다. Thread-safe 하게 개발하기 위해서는 이 부분을 중요하게 생각해야 합니다.

만약 이미 만들어진 Object 내에 새로운 Property 를 설정 혹은 가져 오고자 한다면, `JS_SetProperty ()` 와 `JS_GetProperty ()` 함수의 사용에 대해 익숙해 지시기 바랍니다. 이것은 Object 를 다룰 때 사용되는 것입니다.

7.7 값 (value) 다루기

개념없이 SpiderMonkey 를 사용하면서 가장 까다롭게 생각했었던 부분이었습니다만, 알고 나면 별로 두렵지 않은 부분입니다. SpiderMonkey 에서 가장 중요한 구조체는 아마도 jsval 일 것입니다. 이 구조체는 매우 다양한 얼굴을 가지고 있습니다. `JSVAL_TO_...` 매크로를 이용하여 다양한 모습을 변할 수 있기 때문입니다.

즉, 제가 이해하기로는 jsval 는 JavaScript 에서 표현하는 값들을 모두 표현할 수 있는 구조체입니다. 그래서 다양한 모습으로 변신할 수 있습니다. 또한 `JS_ValueTo... ()` 함수를 이용하여 변신할 수도 있습니다.

Function 의 callback prototype 에서 함수의 인자를 전달할 때도 jsval 가 사용되며, Property 의 callback prototype 또한 jsval 를 사용하여 전달됨을 알 수 있을 것입니다. 이러한 jsval 을 얼마나 잘 사용하느냐에 따라 SpiderMonkey 의 이해도가 그에 따라 높아져 갈 것입니다.

만약 현재 처리하고자 하는 jsval 가 어떤 것인지 확신할 수 없을 경우나, 잘못하여 exception 이나 fault 를 일으킬지 걱정되시는 분들은 `JS_ValueToString ()` 함수와 `JS_GetStringBytes ()` 함수의 조합으로 화면에 직접 찍어보실 수 있습니다. `JS_ValueToString ()` 함수의 경우, jsval 가 문자열이 아닐지라도 적당한 문자열로 변환하여 화면에 찍을 수 있도록 할 것입니다. 그러니 exception 걱정하지 마시고 열심히 디버깅하실 수 있을 것입니다.

7.8 JavaScript 코드 실행하기

사용자가 입력한 혹은 실행하고자 원하는 코드를 실제로 실행하고자 할 때를 고려해 보도록 하겠습니다. 다른 embed 언어와 마찬가지로 string 형태로 실행하고자 하는 JavaScript 코드를 전달해야 합니다.

JavaScript 코드를 실행하는 과정은 다음과 같이 실행됩니다.

1. `JS_BufferIsCompilableUnit ()` 함수를 통해서 현재 compile 하고자 하는 JavaScript 코드가 compile 가능한지 아닌지를 살펴보게 됩니다. 말은 그럴듯 해 보이지만, 완벽한 것이 아님을 말씀드리고 싶습니다. 단

순히 token 이 열리고 잘 닫혔는지, 구문이 제대로 완성되어 있는지만 검사할 뿐입니다. 절대적으로 신뢰할 수는 없습니다. 하지만 없으면 허전한 존재이기도 합니다.

2. `JS_ClearPendingException ()` 함수를 통해서 이전에 compile 에서 연기되어 있는 exception 정보를 깨끗히 정리하게 됩니다.

3. 위의 과정이 끝나면 실제로 `JS_CompileScript ()` 함수를 통해서 컴파일하게 됩니다. 이것은 단순히 실행하기 전에 JavaScript 코드를 opcode 수준으로 바꾸는 것을 의미하는 것이지 실제 실행하는 것이 아닙니다. 만약 compile 된 코드에 대한 decompile 을 하여 그 값을 화면에 출력하고 싶다면 `JS_DecompileScript ()` 함수를 사용할 수 있을 것입니다.

4. 이제 마지막으로 실행하기를 원한다면 `JS_ExecuteScript ()` 함수를 통해서 코드를 실행하게 됩니다. 만약 제대로 실행이 된다면 그에 대한 결과가 반환되어지게 되며, 실행 과정에서 exception 이 발생하였다면 `JS_SetErrorReporter ()` 함수에서 선언해줬던 callback 함수가 호출되어 개발자로 하여금 exception 을 다룰 기회를 줄 것입니다.

5. 마지마그로 `JS_DestroyScript ()` 함수로 메모리 해제를 하게 됩니다.

Context 에서 모든 정보가 관리가 되기 때문에, JavaScript 코드를 실행한 결과를 계속 유지가 되게 됩니다. 그래서 한번에 JavaScript 코드를 실행해도 되지만 조금씩 나누어서 실행을 해도 그 결과는 같게 될 것입니다. 이러한 정보들은 Context 를 해제함으로써 모두 사라지게 되는 것입니다.

제 8 절 디버깅하기

이제부터 SpiderMonkey 를 프로그래밍할 때 어떻게 쉽게 디버깅할 것인가를 알아보도록 하겠습니다.

SpiderMonkey 소스를 다운로드 받아 압축을 해제하면 두 디렉토리가 보이는데, 하나는 jsd 디렉토리이고 나머지가 src 디렉토리입니다. 아래에서 언급되는 내용은 jsd 에 관한 내용이 아님을 알려 드립니다. 제가 이해하기로는 jsd 는 compile 된 JavaScript opcode 를 디버깅할 때 사용되는 API 로 알고 있습니다. 그래서 저의 개발에서는 사용되지 않았으며 그에 대한 경험이 없습니다.

SpiderMonkey 를 사용할 때, 디버깅한다고 할 경우는 아래와 같은 상황들이 있을 수 있을 꺼라 예상됩니다.

- Function callback 혹은 Property callback 내부에서 segment fault 혹은 exception 이 발생할 경우
- JavaScript 코드가 실행되면서 중간에 값의 변동을 화면에 출력한다거나 추적하고 싶을 경우
- JavaScript 코드가 실행 전/실행 후의 값을 화면에 출력하고 싶을 경우.
- JavaScript 코드가 실행되면서 exception 이 발생할 경우

아래에서는 위의 여러 경우의 상황에 대해서 어떻게 대처해야 할지 알아보도록 하겠습니다.

8.1 GDB 로 디버깅하기

SpiderMonkey 관련 컴포넌트를 작성할 때 가장 힘들었던 부분은 '연동'이었습니다. 프로그램 소스 코드는 delphi 로 작성되어 있었고 제 컴포넌트는 C 언어로 짜여있다 보니까, delphi 쪽에서 동적으로 DLL 을 load 를 해서 symbol 을 찾아 함수를 호출하게 된 것입니다. 그래서 프로그램을 디버깅할 때는 delphi 측에서 보았을 때는 제 코드를 디버깅할 수 없는 사태가 빚어졌고, 저는 제 컴포넌트 테스트를 할 수 없었던 것입니다.

개발자에게 디버깅 환경이 제대로 갖추어지지 않는 다는 사실은 '너 좀 삽질해야 정신을 차릴 꺼야'라고 말하는 것과 같습니다. printf 로 할 수 있는 디버깅은 한계가 있고 끝없는 삽질에 삽질을 더할 뿐입니다.

제가 경험했던 것이 바로 그 끝없는 삽질이었고 최악의 디버깅 환경이었던 것입니다. 그 개발 이후로 절대 다른 언어간 통합해서 프로그래밍을 짜지 않겠다고 다짐을 하곤 했습니다. 모두 디버깅의 악몽 때문입니다.

결국 제가 짠 코드를 테스트하기 위해서 라이브러리 API 를 사용하는 sample 코드를 작성할 수 밖에 없었으며, 그것을 GDB 로 테스트함으로써 나름대로 어려움을 헤쳐나갈 수 있었습니다.

MinGW 상에서 GDB 를 사용할 경우, 가끔씩 defined-symbol 을 못 찾을 때가 있습니다. 제 경우가 gdb 를 실행시키고 DLL 에 정의된 symbol 에 대한 breakpoint 를 정의할 때 제대로 찾지 못한 경우가 있었는데, 그럴 경우는 'b main' 를 통해서 main 함수까지 실행시킨 후, 다시 원하는 DLL 내부 symbol 에 breakpoint 를 설정할 경우, 제대로 되었습니다.

Context 에 들어있는 Object 들에 대한 디버깅 (즉, 값 내용 화면 출력이나 값 설정이나 여러가지 일을 하고 싶을 때) 을 수행하고 싶을 때는 JavaScript Console 을 만들 필요가 있을 것입니다. 여기서 JavaScript Console 이라고 하면 Firefox 를 떠 올리시면 될 것입니다.

Firefox 의 경우, 메뉴 Tools 밑에 JavaScript Console 이라는 메뉴가 있습니다. 이 Console 을 이용하여 화면에 HTML 문서에서 설정된 값을 출력하거나 하는 여러가지 일들을 수행할 수 있습니다.

이와 마찬가지로 여러분이 구성한 Object 들을 제대로 테스트할 수 있도록 하기 위해서는 Console 구성이 필요할 것으로 보입니다. GDB 내에서 'p' 명령을 통해서 Console 을 호출할 수 있도록 하면 될 것입니다.

저의 경우, JavaScript Console 을 간단한 shell 형식으로 만들었으며, GDB 를 통해서 디버깅을 하면서 Context 에 설정된 값을 살펴보고 쉽을 때, '`p JavaScriptConsole (...)`'와 같이 명령을 내려 현재까지 설정된 값을 살펴보곤 했습니다. Context 에 대한 포인터만 있으면 되기 때문에 어디서든 호출해서 살펴볼 수 있을 것이며 편리할 것입니다. 만드실 때는 exit 구문을 두어 다시 GDB 로 돌아갈 수 있도록 해주는 센스를 발휘하시기 바랍니다.

8.2 메세지 출력하기

라이브러리를 만들어 보신 분들은 다들 아시겠지만, 전형적인 디버깅 방법 중에 하나는 메세지를 파일에 출력하여 그 순서를 살펴봄으로써 디버깅하는 것입니다.

파일을 통한 디버깅 방법을 수행할 때, 가끔 문제가 있는 부분은 `JS_DecompileScript ()` 함수를 이용하여 compile 된 스크립트를 decompile 하여 파일에 뿌릴 때입니다. 만약 decompile 된 스크립트의 크기가 생각보다 커질 경우, 파일에 출력된 형태는 특정 크기 이상은 글씨가 깨져 출력된다는 점입니다.

대부분 사용되는 방법이기 때문에 길게 언급하지는 않겠습니다.

8.3 JS exception 오류 다루기

마지막으로 SpiderMoneky 가 JavaScript 코드를 실행할 때 exception 이 발생했을 때, 그에 대한 error 를 사용자가 지정한 루틴으로 보고할 수 있도록 하는 기능에 대해서 알바도록 하겠습니다.

사용자 callback 은 앞에서 언급하였듯이 `JS_SetErrorReporter ()` 함수를 사용하여 지정할 수 있습니다. 아래는 callback 함수의 prototype 입니다.

```
void
callbackfunc (cx, msg, report)
     JSContext *cx;
     const char *message;
     JSErrorReport *report;
```

JSErrorReport 구조체를 통해서 오류가 발생한 파일 이름과, 줄 번호 등등 여러 error 관련 정보를 볼 수 있습니다. Exception 에 대한 flag 들은 `JSREPORT_IS_...` 매크로를 통해서 확인할 수 있으며 오류가 어떤 함수들로부터 시작되고 끝났는지에 관한 backtrace 할 수 있습니다. 이 부분은 Pending Exception 받아와 stack 에 관한 정보를 출력함으로써 이루어지게 됩니다.

이 부분은 정말 중요한 부분이라고 생각합니다. 라이브러리를 작성하면서 그리고 내부 Object 들은 손수 하나 하나 작성하면서, DOM Tree 가 제대로 작성되고 있는지 테스트를 하면서 이 callback 함수의 도움을 많이 받았습니다.

JavaScript 엔진에서 발생하는 exception 메세지를 보고 하나씩 대처하는 것이 빠른 개발을 보장해 준다고 생각합니다.

디버깅으로 고생했던 저에겐 이 기능이 많은 도움이 되었으며, 그 때 자세한 JSD 의 기능에 대해서 테스트해 보지 않은 것이 조금은 아쉬움으로 남아 있습니다.

제 9 절 마치며

SpiderMonkey 를 가지고 더 이상 작업할 일은 없지만 그 동안 이 라이브러리를 사용하면서 고생했었던 기억들이 많이 납니다. 가장 치명적이었던 기억은 HTML 프레임을 처리해야 했던 기억인데, 하나의 Context 만으로 처리할려고 고집 부리다 수많은 알기도 힘든 segment fault 기억들.. 그리고 프로그램의 메인 소스는 delphi 로 작성되고 제가 맡은 부분은 c 로 짜여 있음으로 인해 발생한 엄청 힘든 debugging 과정들. 정말 다시는 이종 언어로 프로그램을 작성하지 않겠다고 결심을 하게끔 했습니다.

하지만 그렇게 힘들게 만들어진 제품이 국내에서 인정받으며 팔리고 있는 현실을 볼 때 나름대로 보람되고 저 자신도 도움이 되는 사람일 수 있구나 라고 느끼곤 합니다.

SpiderMonkey 를 어떻게 활용할 것인가는 여러분의 몫입니다만 확실한 것은 매우 신뢰할 수 있는 embedding 언어라는 것. 그리고 확장성 또한 좋다는 것. 등등 여러 가지가 있습니다.

최소한 제가 제가 맡은 과제를 해결함에 있어서 SpiderMonkey 를 JavaScript 처리 엔진으로 도입한 것을 전혀 후회하지 않습니다.

다만 가끔씩 들리는 사장님의 “Visual Basic Script 처리를 할 수 있게 해야지!”라는 말을 들을 때 마다 조금 찔리지만 말입니다. (웹 페이지에 VBScript 쓴 웹 개발자들이 미워요. ;-))

매우 좋은 엔진이니 꼭 한번들 경험해 보시기 바랍니다.

질문이나 혹은 문서의 개선에 대한 의견이 있으신 분은 언제든지 환영합니다. 저의 email 주소로 메일 주시면 답변 드리겠습니다.