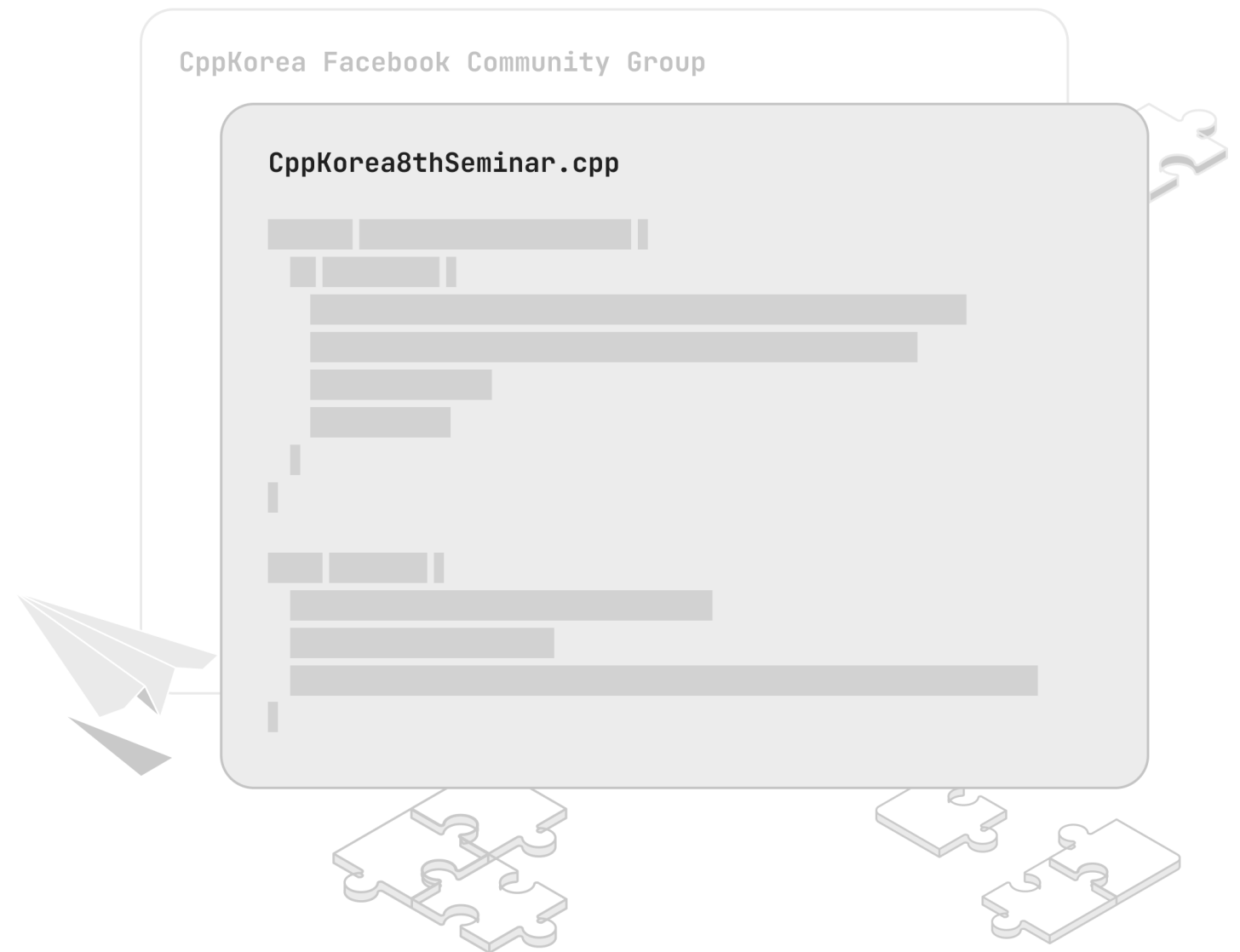


쉽게 따라하는 V8 임베딩

C++에서 자바스크립트 돌리기



DGIST 기초학부 김찬중
freiye.paxbun@gmail.com
<https://github.com/paxbun>



JavaScript 101

—
HTML, CSS와 함께 웹을 구성하는 세 언어 중 하나

버튼을 누르면 메뉴를 표시한다

스크롤을 내리면 새 데이터를 불러온다

검색창에 텍스트를 입력하면 서버와 통신 후 예상 결과를 미리 보여준다

서버 개발에도 사용

런타임 – Node.js, Deno

백엔드 프레임워크 – Next.js, Express.js, ...

클라우드 서버리스 컴퓨팅 (FaaS) – AWS Lambda, Azure Functions, ...

JavaScript 101

—
각 브라우저에는 자바스크립트 엔진이 포함되어있다

Chromium - V8

Firefox – SpiderMonkey

WebKit – JavaScriptCore

Edge Legacy - ChakraCore

브라우저가 아닌 환경에서도 자바스크립트 엔진을 쓸 수 있다

Node.js, Deno는 둘 다 V8 기반

React Native는 JavaScriptCore 사용

JavaScript 101

—
Hello, world!

```
alert("Hello, world!"); // Hello, world! 메시지 박스 표시  
main() // 세미콜론 생략 가능
```

```
// 함수 호이스팅  
function main() {  
    console.log("Hello, world!"); // 콘솔에 Hello, world! 출력  
}
```

```
let a = 6;  
console.log(a / 5); // 1.2
```

JavaScript 101

함수가 일급 객체

자바스크립트에서 :

// 함수를 반환하는 자바스크립트 함수

```
function makeMultiplier(factor) {  
    return function (num) {  
        return factor * num;  
    };  
}
```

```
let multiplyByFive = makeMultiplier(5);  
alert(multiplyByFive(15)); // 75
```

C++에서 :

// 함수 객체를 반환하는 C++ 함수

```
auto MakeMultiplier(double factor) {  
    return [factor](double num) {  
        return factor * num;  
    };  
}
```

```
auto multiplyByFive = MakeMultiplier(5);  
std::cout << multiplyByFive(15); // 75
```

JavaScript 101

Claudio De Sio on Twitter: "Thanks for inventing #javascript! ;-)" - <https://twitter.com/cdesio/status/1013166206877163520>

> 9999999999999999	> true===1
< 10000000000000000	< false
> 0.5+0.1==0.6	> (!+[]+[]+![]).length
< true	< 9
> 0.1+0.2==0.3	> 9+"1"
< false	< "91"
> Math.max()	> 91-"1"
< -Infinity	< 90
> Math.min()	> []==0
< Infinity	< true
> []+[]	
< ""	
> []+{}	
< "[object Object]"	
> {}+[]	
< 0	
> true+true+true===3	
< true	



JavaScript 101

동적타입, 약타입, 덕타입

동적타입 – 변수에 타입이 지정되지 않는다

예) 어떤 변수에 숫자를 대입한 직후 객체를 대입할 수 있다.

```
let a = 5; a = {}; // OK
```

약타입 – 연산 시 타입이 비교적 덜 중요하게 여겨진다

예) 숫자 + 문자열 → 문자열 연결 숫자 - 문자열 → 빨셈 수행

예) "12" == 12 → true "12" === 12 → false

덕타입 – 적절한 메소드와 프로퍼티만 갖추면 어디든지 들어갈 수 있다

예) { value, done: boolean } 을 반환하는 메소드 next()가 있으면 이터레이터로 동작

8개의 타입만 존재한다

프로그래머가 사용자 정의 타입을 만들 수 없다

Undefined, Null, Boolean, String, Symbol, Number, BigInt, Object

JavaScript 101

상속을 구현하는 프로토타입

```
let a = { name: "a" };  
let b = {  
  value: 15,  
  name: "b",  
  introduce() { console.log(this.name); }  
};
```

```
console.log(a.value); // undefined  
a.__proto__ = b;  
console.log(a.value); // 15
```

```
a.introduce(); // a  
b.introduce(); // b
```


JavaScript 101

클래스 \subset 함수

function으로 클래스 만들기 :

```
function Person(name) {  
  this.name = name;  
}  
Person.prototype.introduce = function() {  
  alert(`Hello, my name is ${this.name}!`);  
}
```

```
// Hello, my name is Minsu!  
new Person("Minsu").introduce();
```

```
// function  
alert(typeof Person);
```

class로 클래스 만들기 (ES6) :

```
class Person {  
  constructor(name) { this.name = name; }  
  introduce() {  
    alert(`Hello, my name is ${this.name}!`);  
  }  
}
```

```
// Hello, my name is Minsu!  
new Person("Minsu").introduce();
```

```
// function  
alert(typeof Person);
```

JavaScript 101

화살표 함수

그 함수가 생성된 시점의 `this`를 캡처하는 함수

```
let global = this;
function normal() { return this; }
let arrow = () => { return this; };
function anotherArrow() { return () => this; }
```

```
let a = { normal, arrow, anotherArrow };
```

```
console.log(a === a.normal());    // true
console.log(a === a.arrow());     // false
console.log(global === a.arrow()); // true
console.log(a === a.anotherArrow()); // true
```

JavaScript 101

항상 싱글스레드로 동작한다

자바스크립트에서 :

```
var sum = 0;
for (let i = 1; i <= 100; ++i) {
  // 1초 후에 실행
  setTimeout(() => {
    sum += i;
  }, 1000);
}
// 2초 후에 결과 표시, 항상 5050
setTimeout(() => console.log(sum), 2000);
```

C++에서 :

```
int sum = 0;
for (int i = 1; i <= 100; ++i) {
  // 1초 후에 실행
  std::thread([&sum, i]() {
    std::this_thread::sleep_for(1s);
    sum += i;
  }).detach();
}
// 2초 후에 결과 표시, 5050 아닐 수도 있음
std::thread([&sum]() {
  std::this_thread::sleep_for(2s);
  std::cout << sum;
}).join();
```

JavaScript 101

항상 싱글스레드로 동작한다

자바스크립트에서 :

```
var sum = 0;
for (let i = 1; i <= 100; ++i) {
  // 1초 후에 실행
  setTimeout(() => {
    sum += i;
  }, 1000);
}
// 2초 후에 결과 표시, 항상 5050
setTimeout(() => console.log(sum), 2000);
```

태스크 큐

```
sum += 1;
```

JavaScript 101

항상 싱글스레드로 동작한다

자바스크립트에서 :

```
var sum = 0;
for (let i = 1; i <= 100; ++i) {
  // 1초 후에 실행
  setTimeout(() => {
    sum += i;
  }, 1000);
}
// 2초 후에 결과 표시, 항상 5050
setTimeout(() => console.log(sum), 2000);
```

태스크 큐

sum += 1;

sum += 2;

JavaScript 101

항상 싱글스레드로 동작한다

자바스크립트에서 :

```
var sum = 0;
for (let i = 1; i <= 100; ++i) {
  // 1초 후에 실행
  setTimeout(() => {
    sum += i;
  }, 1000);
}
// 2초 후에 결과 표시, 항상 5050
setTimeout(() => console.log(sum), 2000);
```

태스크 큐

sum += 1;

sum += 2;

...

sum += 100;

JavaScript 101

항상 싱글스레드로 동작한다

자바스크립트에서 :

```
var sum = 0;
for (let i = 1; i <= 100; ++i) {
  // 1초 후에 실행
  setTimeout(() => {
    sum += i;
  }, 1000);
}
// 2초 후에 결과 표시, 항상 5050
setTimeout(() => console.log(sum), 2000);
```

태스크 큐

sum += 1;

sum += 2;

...

sum += 100;

console.log(sum)

JavaScript 101



항상 싱글스레드로 동작한다

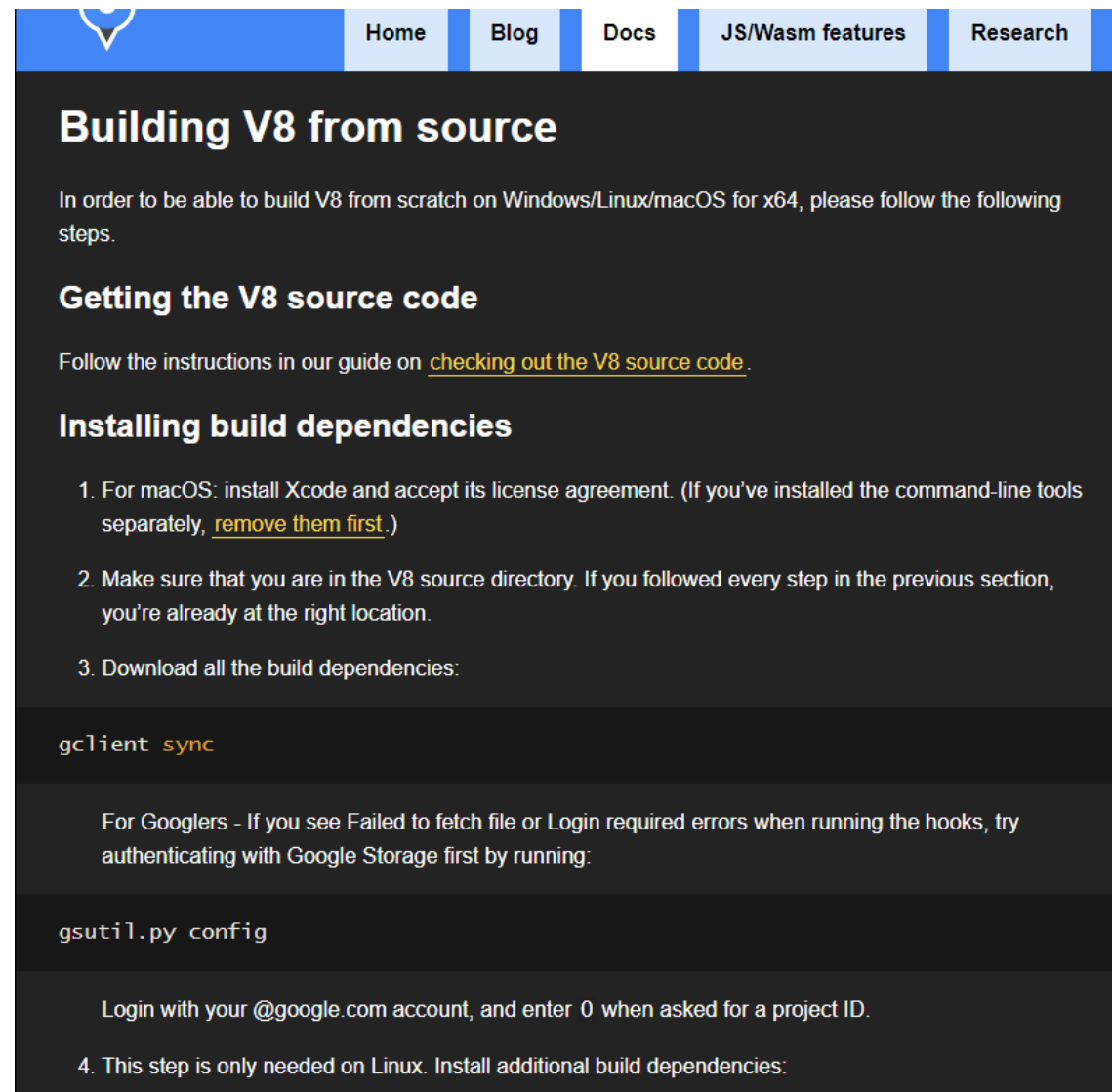
// 영원히 출력되지 않는다

```
setTimeout(() => console.log("Hello"), 0);
```

```
while (true);
```


V8 빌드하기

Building V8 from source – <https://v8.dev/docs/build>



The screenshot shows the V8 documentation website with a blue header containing navigation links: Home, Blog, Docs, JS/Wasm features, and Research. The main content area is dark-themed and titled "Building V8 from source". It provides instructions for building V8 from scratch on Windows/Linux/macOS for x64. The steps include getting the source code, installing build dependencies, and running commands like `gclient sync` and `gsutil.py config`. A note mentions that for Google users, authentication with Google Storage might be required. The page is partially cut off at the bottom.

Building V8 from source

In order to be able to build V8 from scratch on Windows/Linux/macOS for x64, please follow the following steps.

Getting the V8 source code

Follow the instructions in our guide on [checking out the V8 source code](#).

Installing build dependencies

1. For macOS: install Xcode and accept its license agreement. (If you've installed the command-line tools separately, [remove them first](#).)
2. Make sure that you are in the V8 source directory. If you followed every step in the previous section, you're already at the right location.
3. Download all the build dependencies:

```
gclient sync
```

For Googlers - If you see Failed to fetch file or Login required errors when running the hooks, try authenticating with Google Storage first by running:

```
gsutil.py config
```

Login with your @google.com account, and enter 0 when asked for a project ID.

4. This step is only needed on Linux. Install additional build dependencies:

V8 빌드하기

소스 코드 다운로드 및 빌드에 필요한 프로그램 설치

depot_tools 다운로드

리눅스 / 맥

```
git clone https://chromium.googlesource.com/chromium/tools/depot_tools.git
```

윈도우

https://storage.googleapis.com/chrome-infra/depot_tools.zip

depot_tools가 있는 폴더 PATH에 추가

리눅스 / 맥

```
export PATH=/path/to/depot_tools:$PATH
```

윈도우

```
set PATH=C:\path\to\depot_tools;%PATH%
```

V8 빌드하기

소스 코드 다운로드 및 빌드에 필요한 프로그램 설치

gclient 실행 후 원하는 폴더에 V8 소스 코드 다운로드 (윈도우에선 반드시 cmd로 수행)

```
gclient  
cd /path/to/v8  
fetch v8  
cd v8
```

원하는 버전의 V8 사용하기

```
git checkout tags/8.7.220.3
```

V8 빌드하기

소스 코드 다운로드 및 빌드에 필요한 프로그램 설치

빌드에 필요한 프로그램 설치

윈도우 / 맥

`gclient sync`

리눅스

`gclient sync && ./build/install-build-deps.sh`

윈도우 환경변수 설정

`set DEPOT_TOOLS_WIN_TOOLCHAIN=0`

V8 빌드하기

소스 코드 다운로드 및 빌드에 필요한 프로그램 설치

윈도우 Visual Studio 2017 또는 2019에서 다음 컴포넌트 설치

Desktop development with C++

MFC/ATL support

Windows 10 SDK

윈도우 Debugging Tools for Windows 설치

Settings – Apps – Apps & features – Windows Software Development Kit

Modify – Change – Change

Debugging Tools For Windows 선택 후 Change 선택

V8 빌드하기

빌드 및 링크

빌드 스크립트 생성

```
gn gen out/x64.Debug
```

사용 가능한 gn 인자 목록 표시

```
gn args out/x64.Debug --list
```

V8 빌드하기

빌드 및 링크

윈도 gn 인자 설정

MSVC, 디버그 모드, v8_monolith

```
is_clang = false
is_component_build = false
is_debug = true
target_cpu = "x64"
use_custom_libcxx = false
use_lld = false
v8_monolithic = true
v8_use_external_startup_data = false
```

우분투 gn 인자 설정

Clang, 디버그 모드, v8_monolith

```
is_clang = true
is_component_build = false
is_debug = true
target_cpu = "x64"
use_custom_libcxx = false
use_llvm = true
v8_monolithic = true
v8_use_external_startup_data = false
```

V8 빌드하기

빌드 및 링크

빌드

```
ninja -C out/x64.Debug v8_monolith
```

전처리기 정의 복사 (./out/x64.Debug/obj/v8_monolith.ninja의 첫 줄)

```
head -n 1 ./out/x64.Debug/obj/v8_monolith.ninja | sed 's/defines = //'
```


V8 빌드하기

빌드 및 링크

윈도우 CMake

```
link_libraries(v8_monolith winmm dbghelp)
add_compile_definitions(
  -D_ITERATOR_DEBUG_LEVEL=0
  ${V8_COMPILE_DEFINITIONS}
)
```

리눅스 / 맥 CMake

```
link_libraries(v8_monolith pthread)
add_compile_definitions(${V8_COMPILE_DEFINITIONS})
```

V8 빌드하기

빌드 및 링크

윈도우 CMake (/MD /MDd 를 /MT /MTd 로 변경)

```
set(V8_COMPILER_FLAGS
  CMAKE_CXX_FLAGS
  CMAKE_CXX_FLAGS_DEBUG
  CMAKE_CXX_FLAGS_RELEASE
  CMAKE_CXX_FLAGS_MINSIZEREL
  CMAKE_CXX_FLAGS_RELWITHDEBINFO
  CMAKE_C_FLAGS
  CMAKE_C_FLAGS_DEBUG
  CMAKE_C_FLAGS_RELEASE
  CMAKE_C_FLAGS_MINSIZEREL
  CMAKE_C_FLAGS_RELWITHDEBINFO
)
foreach(V8_COMPILER_FLAG ${V8_COMPILER_FLAGS})
  string(REPLACE "/MD" "/MT" ${V8_COMPILER_FLAG} "${${V8_COMPILER_FLAG}}")
  set(${V8_COMPILER_FLAG} "${${V8_COMPILER_FLAG}}" PARENT_SCOPE)
endforeach()
```

V8 빌드하기

paxbun/v8-cmake: Download and Build V8 using CMake automatically - <https://github.com/paxbun/v8-cmake>

V8 for CMake

Download and Build V8 using CMake automatically.

Prerequisites

- Visual Studio 2017/2019 with Desktop development with C++
- Debugging Tools for Windows
- gcc or clang (on Ubuntu)
- Python 3
- Python 2.7 (on Ubuntu)
- CMake 3.13 or higher

Examples

Release mode, Windows

```
set(V8_VERSION "8.7.220.3")
set(V8_CONFIG_STR "x64.release")
set(V8_IS_DEBUG OFF)
set(V8_TARGET_NAME "v8_monolith")
set(V8_IS_COMPONENT_BUILD OFF)
set(V8_IS_CLANG OFF)
set(V8_TARGET_CPU "x64")
set(V8_USE_CUSTOM_LIBCXX OFF)
set(V8_USE_LLD OFF)
set(V8_V8_MONOLITHIC ON)
set(V8_V8_USE_EXTERNAL_STARTUP_DATA OFF)

add_subdirectory(v8-cmake)

target_include_directories(cc3 PRIVATE ${V8_INCLUDE_DIRS})
target_link_directories(cc3 PRIVATE ${V8_LIBRARY_DIRS})
target_link_libraries(cc3 v8_monolith winmm dbghelp)
target_compile_definitions(cc3 PRIVATE -D_ITERATOR_DEBUG_LEVEL=0 ${V8_COMPILE_DEFINITIONS})
```

V8으로 자바스크립트 코드 실행하기

CppKorea/CppKoreaSeminar8th - <https://github.com/CppKorea/CppKoreaSeminar8th>

README.md

쉽게 따라하는 V8 임베딩

V8은 구글이 Chromium을 개발하면서 함께 개발한 자바스크립트 엔진입니다. Node.js 및 Deno와 같은 런타임을 개발하는데도 사용되었고, 크로스플랫폼을 지원합니다. 이 발표에선 V8을 다운받아 빌드한 후, C++ 프로그램에 임베딩하는 방법을 소개합니다.

목차

- V8 소개
- V8 소스코드 다운로드 및 빌드하는 법
- V8 링크하기
- 자바스크립트 소개
- V8으로 자바스크립트 코드 실행하기
- C++ 함수를 자바스크립트에 노출하기
- ES6 모듈 코드 실행하기
- C++ 함수로 이루어진 ES6 모듈 만들기
- setTimeout 구현하기
- 자바스크립트 Promise 활용하기
- C++ 클래스를 자바스크립트에 노출하기

예제 코드 빌드하기

맥을 사용하시는 분들은 주의 사항을 먼저 확인해주세요.

빌드에는 다음 프로그램이 필요합니다.

윈도우에서:

- Visual Studio 2017/2019 with Desktop development with C++

V8으로 자바스크립트 코드 실행하기

VM 인스턴스를 나타내는 v8::Isolate

각각의 Isolate 객체는 서로 완전히 독립되어있다
어떤 Isolate에서 만들어진 객체를 다른 Isolate에서 사용해서는 안된다
하나의 Isolate는 동시에 한 개의 스레드에서만 동시에 접근 가능하다

```
#include <libplatform/libplatform.h>
#include <v8.h>

// v8_use_external_startup_data = false 로 빌드하였으므로 호출 생략
// v8::V8::InitializeICUDefaultLocation(argv[0]);
// v8::V8::InitializeExternalStartupData(argv[0]);

// v8::V8::Initialize가 호출되기 전에 platform 객체가 설정되어야 한다
std::unique_ptr<v8::Platform> platform = v8::platform::NewDefaultPlatform();
v8::V8::InitializePlatform(platform.get());

// Isolate를 생성하기 전에 반드시 호출되어야 한다
v8::V8::Initialize();

// Isolate 객체 생성
v8::Isolate::CreateParams createParams;
createParams.array_buffer_allocator = v8::ArrayBuffer::Allocator::NewDefaultAllocator();
v8::Isolate* isolate = v8::Isolate::New(createParams);
```

V8으로 자바스크립트 코드 실행하기

함수 스택에서 객체를 관리하는 v8::HandleScope

생성된 이후 만들어진 객체들은 모두 v8::HandleScope가 관리한다
소멸된 후엔 v8::HandleScope가 관리하던 v8::Local<T> 핸들을 GC가 추적하지 않는다
이후 v8::Local<T>에 접근하는 것은 UB
v8::HandleScope를 동적 생성하면 안된다

```
// 생성자에서 v8::Isolate::Enter를, 소멸자에서 v8::Isolate::Exit를 호출해준다
v8::Isolate::Scope isolateScope { isolate };

// handleScope가 소멸되거나, 새로운 v8::HandleScope 인스턴스가 생성되기 전까지
// 모든 v8::Local<T> 핸들은 handleScope가 관리한다
v8::HandleScope handleScope { isolate };

v8::Local<v8::Value> value = MakeNewValue(), value2;
{
    // handleScope2가 소멸될 때까지 생성된 모든 v8::Local<T> 핸들은 handleScope2가 관리한다
    v8::HandleScope handleScope2 { isolate };
    // 값을 새로 생성한다
    value2 = MakeNewValue();
}
value->DoSomeOperation(); // Ok
value2->DoSomeOperation(); // UB
```

V8으로 자바스크립트 코드 실행하기

힙에 있는 객체를 나타내는 핸들

`v8::Local<T>`

`v8::HandleScope`가 관리하는 객체를 나타내는 핸들
`v8::HandleScope`가 소멸한 이후 `v8::Local<T>`에 접근하는 것은 UB

`v8::PersistentBase<T>`

`v8::HandleScope`가 관리하지 않는 객체를 나타내는 핸들
`v8::PersistentBase<T>::Reset`이 호출될 때까지 살아남는다
`v8::PersistentBase<T>::SetWeak`으로 finalization 콜백을 설정할 수 있다

`v8::Eternal<T>`

절대로 소멸되지 않을 객체를 나타내는 핸들

V8으로 자바스크립트 코드 실행하기

실행 콘텍스트를 나타내는 v8::Context

전역 객체에 대한 정보를 가지고 있다

SecurityToken 설정을 통해 다른 Context에서 생성된 객체를 가져오는 것을 막을 수 있다 (참고: Same-origin policy)
Context를 생성하기 전 전역 객체를 미리 생성할 수 있다

```
v8::Local<v8::Context> context = v8::Context::New(isolate);
{
    // 생성자에서 v8::Context::Enter를, 소멸자에서 v8::Context::Exit를 호출해준다
    v8::Context::Scope contextScope { context };
    // function foo() { console.log('Hello, world!'); } 컴파일 후 실행
    // foo(); 컴파일 후 실행
    // Hello, world! 출력
}
```



```
v8::Local<v8::Context> context2 = v8::Context::New(isolate);
{
    v8::Context::Scope contextScope { context2 };
    // foo(); 컴파일 후 실행
    // Undefined reference to foo
}
```


V8으로 자바스크립트 코드 실행하기

실패할 수 있는 연산의 결과를 나타내는 v8::MaybeLocal<T>

일부 연산은 실패할 수 있다

예) 코드를 컴파일한다 → 코드 문법이 맞으면 성공, 문법이 틀리면 실패

예) 코드를 실행한다 → 예외가 던져지지 않으면 성공, 예외가 던져지면 실패

실패할 수 있는 연산의 결과는 항상 v8::Maybe<T> 또는 v8::MaybeLocal<T>

v8::MaybeLocal<T>::IsEmpty 를 호출하여 연산이 성공했는지 실패했는지 확인한다

실패한 결과에 v8::MaybeLocal<T>::ToLocalChecked를 호출하면 런타임 에러가 발생한다

```
std::string code;
// 새 문자열을 생성하는 연산은 문자열이 너무 길면 실패한다
v8::MaybeLocal<v8::String> maybeSource = v8::String::NewFromUtf8(isolate, code.c_str());
// 연산이 실패했다면
if (maybeSource.IsEmpty()) {
    std::cout << "Given string is too long!";
    return;
}
// 연산이 성공했다면 값을 가져온다
v8::Local<v8::String> source = maybeSource.ToLocalChecked();
```

V8으로 자바스크립트 코드 실행하기

예외 처리 로직을 지원하는 v8::TryCatch

연산이 실패했다면 예외가 던져진다
던져진 예외는 v8::TryCatch를 이용해 가져온다
임베더가 예외를 던질 땐 v8::Isolate::ThrowException을 이용한다

```
// 예외를 던지는 함수
void SomeEmbedderFunction() {
    v8::Local<v8::String> msg = v8::String::NewFromUtf8Literal(isolate, "Runtime error!");
    isolate->ThrowException(v8::Exception::Error(msg));
    return;
}

// someEmbedderFunction(); 컴파일
v8::Local<v8::String> source = v8::String::NewFromUtf8Literal(isolate, "someEmbedderFunction();");
v8::Local<v8::Script> script = v8::Script::Compile(context, source).ToLocalChecked();

// 코드 실행
v8::TryCatch      tryCatch { isolate };
v8::MaybeLocal<v8::Value> maybeResult = script->Run(context);
assert(maybeResult.IsEmpty());

// 던져진 예외 처리
v8::String::Utf8Value utf8(isolate, tryCatch.Exception());
assert(strcmp("Error: Runtime error!", *utf8) == 0);
```

V8으로 자바스크립트 코드 실행하기

Examples/1-run-javascript

```
Microsoft Visual Studio Debug Console

> let a = 5a;
>
(Compile error) SyntaxError: Invalid or unexpected token
> let a = 6;
> let a = 7;
>
(Compile error) SyntaxError: Identifier 'a' has already been declared
> var sum = 0;
> for (let i = 1; i <= 100; ++i) sum += i;
> console.log(sum);
>
5050
> .exit

C:\Users\freiy\Desktop\easy-v8-embedding\out\build\x64-Debug\Examples\1-run-javascript\1-run-javascript.exe (process 29080) exited with code 0.
```

C++ 함수를 자바스크립트에 노출하기

자바스크립트에 노출할 함수 만들기

자바스크립트에 노출할 함수의 시그니처:

```
void (const v8::FunctionCallbackInfo<v8::Value>&)
```

// 주어진 경로에 있는 파일을 읽어 문자열로 반환하는 함수

```
void ReadStringFromFile(v8::FunctionCallbackInfo<v8::Value> const& info) {
```

```
    // 첫번째 인자를 문자열로 가져온다
```

```
    v8::String::Utf8Value filePath { isolate, info[0] };
```

```
    // 파일 읽기
```

```
    std::string content(fileSize, ' ');
```

```
    std::ifstream ifs { *filePath };
```

```
    ifs.read(std::addressof(content[0]), fileSize);
```

```
    // 반환 값 설정
```

```
    v8::Local<v8::String> contentValue
```

```
        = v8::String::NewFromUtf8(isolate, content.c_str()).ToLocalChecked();
```

```
    info.GetReturnValue().Set(contentValue);
```

```
}
```

C++ 함수를 자바스크립트에 노출하기

함수를 만들 때 사용되는 v8::FunctionTemplate

v8::FunctionTemplate::GetFunction을 통해 v8::Function을 생성한다
v8::FunctionTemplate을 이용해 생성할 수 있는 함수 인스턴스는 컨텍스트 하나 당 하나씩이다

```
// ReadStringFromFile을 노출하는 v8::FunctionTemplate 생성
```

```
v8::Local<v8::FunctionTemplate> readStringFromFile  
= v8::FunctionTemplate::New(isolate, ReadStringFromFile);
```

```
// 함수 객체 생성
```

```
v8::Local<v8::Function> functionInstance  
= readStringFromFile.GetFunction(context);
```

C++ 함수를 자바스크립트에 노출하기

객체를 만들 때 사용되는 v8::ObjectTemplate

v8::ObjectTemplate::NewInstance를 통해 v8::Object를 생성한다

V8::FunctionTemplate을 v8::ObjectTemplate에 넣을 수 있다

v8::Context 를 새로 생성할 때 v8::ObjectTemplate를 넣어 전역에 함수와 변수를 미리 정의할 수 있다

```
// readStringFromFile 프로퍼티에 ReadStringFromFile을 가지는 v8::ObjectTemplate 생성
```

```
v8::Local<v8::ObjectTemplate> global = v8::ObjectTemplate::New(isolate);
```

```
global->Set(isolate, "readStringFromFile", readStringFromFile);
```

```
// 위에서 생성한 v8::ObjectTemplate를 전역 객체로 가지는 콘텍스트 생성
```

```
v8::Local<v8::Context> context = v8::Context::New(isolate, nullptr, global);
```

```
// readStringFromFile("path/to/file") 사용 가능
```

C++ 함수를 자바스크립트에 노출하기

console.log 임베딩

v8::debug::ConsoleDelegate를 상속하여 구현한다
console.clear 같은 다른 함수들도 임베딩할 수 있다

```
#include <src/debug/debug-interface.h>
```

```
struct ConsoleDelegate : public v8::debug::ConsoleDelegate {  
    v8::Isolate* isolate;  
    ConsoleDelegate(v8::Isolate* isolate) : isolate { isolate } {}  
    virtual void Log(const v8::debug::ConsoleCallArguments& args,  
                    const v8::debug::ConsoleContext& context) override {  
        for (int i = 0; i < args.Length(); ++i)  
            std::cout << *v8::String::Utf8Value { isolate, args[i] } << std::endl;  
    }  
};
```

```
ConsoleDelegate delegate { isolate };
```

```
v8::debug::SetConsoleDelegate(isolate, &delegate);
```

C++ 함수를 자바스크립트에 노출하기

console 정의 제거

console을 사용하고 싶지 않다면 전역에서 console을 제거할 수 있다

```
v8::Local<v8::Context> context = v8::Context::New();
```

```
// 전역 객체에서 console 프로퍼티 제거
```

```
context->Global()->Delete(  
    context, v8::String::NewFromUtf8Literal(isolate, "console"));
```


C++ 함수를 자바스크립트에 노출하기

Examples/2-expose-cpp-to-javascript

Microsoft Visual Studio Debug Console

```
> console.log(readStringFromFile("C:/Windows/win.ini"))  
>  
; for 16-bit app support  
[fonts]  
[extensions]  
[mci extensions]  
[files]  
[Mail]  
MAPI=1  
  
> .exit
```

```
C:\Users\freiy\Desktop\easy-v8-embedding\out\build\x64-Debug\Examp  
javascript\2-expose-cpp-to-javascript.exe (process 27328) exited wi
```

win.ini - Notepad

File Edit Format View Help

```
; for 16-bit app support  
[fonts]  
[extensions]  
[mci extensions]  
[files]  
[Mail]  
MAPI=1
```

Ln 1, 100% Windows (CRLF) UTF-8

ES6 모듈 사용하기

ECMAScript 표준

JavaScript의 문법은 ECMA-262에서 정한다
ECMA-262에서 정하는 스펙 이름이 ECMAScript
ECMA-262 6판을 줄여서 ES2015 또는 ES6라고 부른다

ES6 모듈

CommonJS, AMD 등 다양한 모듈 구현 방식이 존재했다
외부적인 구현 없이 기본적으로 모듈을 지원하기 위해 추가된 문법

ES6 모듈 사용하기

ES6 모듈 문법

math.mjs

```
export function add(lhs, rhs) {  
  return lhs + rhs;  
}  
  
export default function hello() {  
  console.log("Hello, world!");  
}
```

index.mjs

```
// export default된 것 가져오기 (이름 일치할 필요 없음)  
import hello from "math.mjs";  
// export된 것 가져오기 (이름 일치해야 함)  
import { add } from "math.mjs";  
// 이렇게 다른 이름으로 가져올 수 있음  
import { add as foo } from "math.mjs";  
// 이렇게 한번에 가져올 수 있음  
import hello, { add } from "math.mjs";  
  
console.log(add(2, 5)); // 7  
hello(); // Hello, world!  
  
// 모듈 자체를 다 갖고 올 수 있음  
import hello, * as math from "math.mjs";  
console.log(math.add(2, 5)); // 7  
hello(); // Hello, world!
```

ES6 모듈 사용하기

ES6 모듈을 나타내는 v8::Module

모듈 소스 컴파일 :

```
v8::Local<v8::String> source; // 모듈 소스
v8::Local<v8::String> moduleName; // 모듈 이름
```

```
v8::ScriptCompiler::Source script {
  source,
  v8::ScriptOrigin {
    moduleName,
    v8::Integer::New(isolate, 0),
    v8::Integer::New(isolate, 0),
    v8::False(isolate),
    v8::Integer::New(isolate, 0),
    v8::Null(isolate),
    v8::False(isolate),
    v8::False(isolate),
    v8::True(isolate),
  },
};
```

```
v8::Local<v8::Module> module
= v8::ScriptCompiler::CompileModule(
  isolate, &script).ToLocalChecked();
```

일반 소스 컴파일 :

```
v8::Local<v8::String> source; // 소스
v8::Local<v8::Script> script
= v8::Script::Compile(context, source);
```

ES6 모듈 사용하기

모듈 초기화

v8::Module::InstantiateModule 로 초기화
이 때 그 모듈이 import하는 모듈들도 찾아준다

```
std::map<std::string, v8::Local<v8::Module>> modules;

v8::MaybeLocal<v8::Module> ModuleResolveCallback(v8::Local<v8::Context> context,
                                                  v8::Local<v8::String> specifier,
                                                  v8::Local<v8::Module> referrer)
{
    v8::String::Utf8Value specifierName { isolate, specifier };
    if (auto it = modules.find(*specifierName); it != modules.end()) return it->second;

    v8::Local<v8::String> msg = v8::String::NewFromUtf8Literal(isolate, "No such module exists");
    isolate->ThrowException(v8::Exception::Error(msg));
    return v8::MaybeLocal<v8::Module> {};
}

v8::Local<v8::Module> module
    = v8::ScriptCompiler::CompileModule(isolate, &Script)
        .ToLocalChecked();

module->InstantiateModule(context, ModuleResolveCallback);
modules.insert(std::make_pair(moduleName, module));
```

ES6 모듈 사용하기

모듈 실행

v8::Module::Evaluate 로 실행

```
v8::Local<v8::Value> result = module->Evaluate(context).ToLocalChecked();
```

ES6 모듈 사용하기

Examples/3-use-es6-modules

```
Microsoft Visual Studio Debug Console

> export function add(lhs, rhs) {
>   return lhs + rhs;
> }
> export default function hello() {
>   console.log("Hello, world!");
> }
> .module My Math Module
>
> import hello, { add as foo } from "My Math Module";
> hello();
> foo(2, 5);
> .module Main
Hello, world!
7
> .exit

C:\Users\freiy\Desktop\easy-v8-embedding\out\build\x64-Debug\Examples\3-use-es6-modules\3-use-es6-modules.exe (process 28620) exited with code 0.
```

C++ 함수로 이루어진 ES6 모듈 만들기

v8::Module::CreateSyntheticModule

C++에서 모듈의 export를 정할 수 있는 기능

```
v8::MaybeLocal<v8::Value> SyntheticModuleCallback(v8::Local<v8::Context> context,
                                                    v8::Local<v8::Module> module) {
    // CreateSyntheticModule에 넘겨줬던 이름만 사용 가능
    v8::Local<v8::Function> myFunc = v8::Function::New(context, MyFunc).ToLocalChecked();
    v8::Local<v8::String> exportName = v8::String::NewFromUtf8Literal(isolate, "myFunc");
    module->SetSyntheticModuleExport(exportName, myFunc);
    return v8::True(isolate);
}

// 모듈 이름
v8::Local<v8::String> moduleName;
// 모듈에서 export하는 것들 이름
std::vector<v8::Local<v8::String>> moduleExportNames {
    v8::String::NewFromUtf8Literal(isolate, "myFunc")
};
// 모듈 생성
v8::Local<v8::Module> module
    = v8::Module::CreateSyntheticModule(isolate, moduleName, moduleExportNames, SyntheticModuleCallback);
```


C++ 함수로 이루어진 ES6 모듈 만들기

v8::Module::CreateSyntheticModule

default export도 같은 방법으로 사용

```
v8::MaybeLocal<v8::Value> SyntheticModuleCallback(v8::Local<v8::Context> context,
                                                    v8::Local<v8::Module> module) {
    // CreateSyntheticModule에 넘겨줬던 이름만 사용 가능
    v8::Local<v8::Function> myFunc = v8::Function::New(context, MyFunc).ToLocalChecked();
    v8::Local<v8::String> exportName = v8::String::NewFromUtf8Literal(isolate, "default");
    module->SetSyntheticModuleExport(exportName, myFunc);
    return v8::True(isolate);
}

// 모듈 이름
v8::Local<v8::String> moduleName;
// 모듈에서 export하는 것들 이름
std::vector<v8::Local<v8::String>> moduleExportNames {
    v8::String::NewFromUtf8Literal(isolate, "default")
};
// 모듈 생성
v8::Local<v8::Module> module
    = v8::Module::CreateSyntheticModule(isolate, moduleName, moduleExportNames, SyntheticModuleCallback);
```

C++ 함수로 이루어진 ES6 모듈 만들기

Examples/4-es6-modules-with-cpp



```
Microsoft Visual Studio Debug Console

> import { readStringFromFile } from "fileUtils";
> console.log(readStringFromFile("C:/Windows/win.ini"));
> .module main
; for 16-bit app support
[fonts]
[extensions]
[mci extensions]
[files]
[Mail]
MAPI=1

> .exit

C:\Users\freiy\Desktop\easy-v8-embedding\out\build\x64-Debug\Examples\4-es6-modules-with-cpp\4-es6-modules-with-cpp.exe (process 10828)
exited with code 0.
```

setTimeout 구현하기

setTimeout(callback, ms, ...args)

callback: 실행할 함수

ms: 몇 초 뒤에 실행하는가 (단위 밀리초)

arg: 함수에 넘길 인자 목록

```
var sum = 0;
for (let i = 1; i <= 100; ++i) {
  // 1초 후에 실행
  setTimeout(() => {
    sum += i;
  }, 1000);
}
// 2초 후에 결과 표시, 항상 5050
setTimeout(() => console.log(sum), 2000);
```

setTimeout 구현하기

Move만 가능한 v8::Global<T>

v8::PersistentBase<T>를 상속하는 클래스들 중 하나 (cf. v8::Persistent<T>)
v8::HandleScope가 관리하지 않는 핸들을 만들 수 있다

```
v8::Global<v8::Value> global;  
void SomeFunc(v8::FunctionCallbackInfo<v8::Value> const& info) { global = info[0]; }
```

// someFunc(5); 컴파일 후 실행

```
v8::Local<v8::Value> value = global.Get(isolate);  
assert(value->Int32Value(context).ToChecked() == 5);  
global.Reset();
```

setTimeout 구현하기

```
struct TimeoutQueueItem {
    chrono::time_point<chrono::steady_clock> firesAt;    // 함수가 실행되는 시점
    v8::Global<v8::Function>      functionToCall; // 실행할 함수
    std::vector<v8::Global<v8::Value>> arguments; // 함수에 넘겨질 인자 목록
};

using TimeoutQueueItemPtr = std::shared_ptr<TimeoutQueueItem>;

struct TimeoutQueueItemComparator {
    // 실행되는 시점으로 비교
    constexpr bool operator()(TimeoutQueueItemPtr const& lhs,
                               TimeoutQueueItemPtr const& rhs) const {
        return lhs->firesAt > rhs->firesAt;
    }
};

std::priority_queue<TimeoutQueueItemPtr,
    std::vector<TimeoutQueueItemPtr>,
    TimeoutQueueItemComparator> timeoutQueue;
```

setTimeout 구현하기

```
void SetTimeout(v8::FunctionCallbackInfo<v8::Value> const& info) {  
    // 함수가 실행될 시점 계산  
    double milli    = info[1]->NumberValue(isolate->GetCurrentContext()).ToChecked();  
    auto duration   = chrono::duration<double, std::milli> { milli };  
    auto willFireAt = chrono::static_clock::now() + duration;  
  
    // 인자들을 v8::Global로 변환  
    std::vector<v8::Global<v8::Value>> arguments;  
    for (int i = 2; i < info.Length(); ++i)  
        arguments.push_back(v8::Global<v8::Value> { isolate, info[i] });  
  
    // 우선순위 큐에 넣기  
    timeoutQueue.push(TimeoutQueueItemPtr {  
        new TimeoutQueueItem {  
            chrono::time_point_cast<chrono::milliseconds>(willFireAt),  
            v8::Global<v8::Function> { isolate, info[0].As<v8::Function>() },  
            std::move(arguments),  
        },  
    });  
}
```

setTimeout 구현하기

```
// 스크립트 컴파일 후 실행

// 큐가 비어있지 않다면 (스크립트 실행 중 setTimeout을 실행했다면)
while (!timeoutQueue.empty())
{
    // 맨 처음으로 실행해야하는 함수를 가져온다
    TimeoutQueueItem item = std::move(*timeoutQueue.top());
    timeoutQueue.pop();

    // 나중에 실행해야한다면 기다린다
    if (item.firesAt >= chrono::static_clock::now()) std::this_thread::sleep_until(item.firesAt);

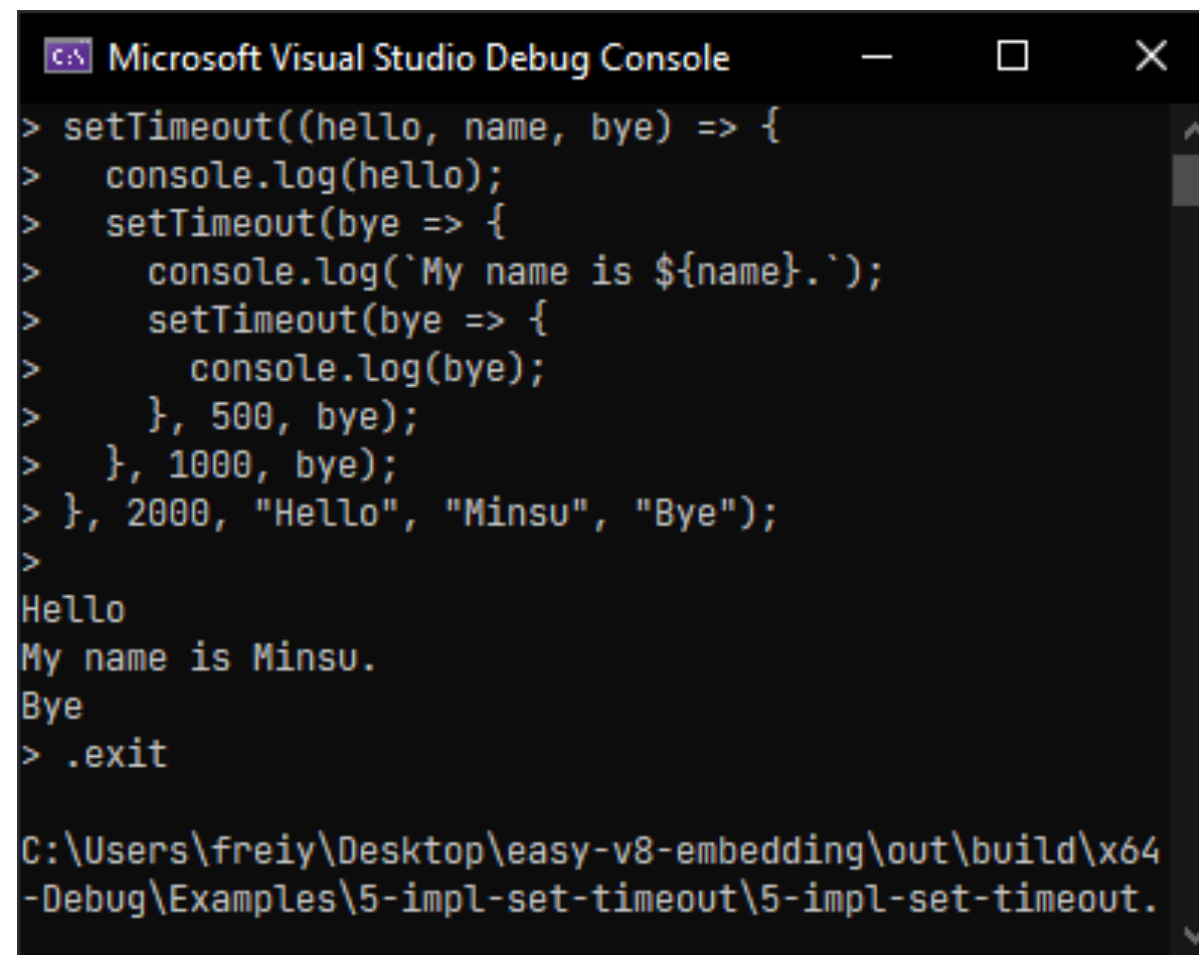
    // 인자들을 v8::Local로 변환
    std::vector<v8::Local<v8::Value>> arguments;
    for (auto& argument : item.arguments) arguments.push_back(argument.Get(isolate));

    // 함수 실행
    item.functionToCall.Get(isolate)->CallAsFunction(
        context, context->Global(), (int)arguments.size(), arguments.data());

    // TimeoutQueueItem의 소멸자가 호출되면서 v8::PersistentBase::Reset 호출
}
```

setTimeout 구현하기

Examples/5-impl-set-timeout



```
Microsoft Visual Studio Debug Console

> setTimeout((hello, name, bye) => {
>   console.log(hello);
>   setTimeout(bye => {
>     console.log(`My name is ${name}.`);
>     setTimeout(bye => {
>       console.log(bye);
>     }, 500, bye);
>   }, 1000, bye);
> }, 2000, "Hello", "Minsu", "Bye");
>
Hello
My name is Minsu.
Bye
> .exit

C:\Users\freiy\Desktop\easy-v8-embedding\out\build\x64-Debug\Examples\5-impl-set-timeout\5-impl-set-timeout.
```


Promise 활용하기

비동기 연산의 결과를 나타내는 Promise

연산의 결과가 함수 호출 직후 바로 나오지 않을 수 있다

예) fetch로 서버에 요청을 보내면 응답이 되돌아오기 전에 값이 반환된다

예) setTimeout을 사용하면 주어진 함수가 n초 뒤에 실행된다

```
new Promise(resolve => {
  console.log("Promise executor");
  setTimeout(() => {
    // resolve가 호출되면 Promise의 상태가
    // pending에서 fulfilled로 바뀐다
    resolve("Bye, world!");
  }, 2000);
}).then(message => {
  // then은 resolve가 호출됐을 때 실행될 콜백을 설정한다
  console.log(message);
  // then에 넣은 콜백은 값을 반환할 수 있다.
  return "Foo, world!";
}).then(message => {
  // then을 계속해서 이어쓸 수 있다
  console.log(message);
});
```

```
console.log("Hello, world!");
```

```
new Promise((resolve, reject) => {
  // reject가 호출되면 Promise의 상태가
  // pending에서 rejected로 바뀐다
  reject("hello");
})
// reject가 호출되면 then에 넘겨진 부분은 실행되지 않는다
.then(message => console.log(message))
// reject가 호출되면 catch에 넘겨진 부분이 실행된다
.catch(message => alert(message));
```

Promise 활용하기

비동기 연산의 결과를 나타내는 Promise

연산의 결과가 함수 호출 직후 바로 나오지 않을 수 있다

예) fetch로 서버에 요청을 보내면 응답이 되돌아오기 전에 값이 반환된다

예) setTimeout을 사용하면 주어진 함수가 n초 뒤에 실행된다

```
new Promise(resolve => {  
  console.log("Promise executor");  
  setTimeout(() => {  
    // resolve가 호출되면 Promise의 상태가  
    // pending에서 fulfilled로 바뀐다  
    resolve("Bye, world!");  
  }, 2000);  
}).then(message => {  
  // then은 resolve가 호출됐을 때 실행될 콜백을 설정한다  
  console.log(message);  
  // then에 넣은 콜백은 값을 반환할 수 있다.  
  return "Foo, world!";  
}).then(message => {  
  // then을 계속해서 이어쓸 수 있다  
  console.log(message);  
});
```

```
console.log("Hello, world!");
```

```
new Promise((resolve, reject) => {  
  // reject가 호출되면 Promise의 상태가  
  // pending에서 rejected로 바뀐다  
  reject("hello");  
})  
// reject가 호출되면 then에 넘겨진 부분은 실행되지 않는다  
.then(message => console.log(message))  
// reject가 호출되면 catch에 넘겨진 부분이 실행된다  
.catch(message => alert(message));
```

Promise 활용하기

비동기 연산의 결과를 나타내는 Promise

연산의 결과가 함수 호출 직후 바로 나오지 않을 수 있다

예) fetch로 서버에 요청을 보내면 응답이 되돌아오기 전에 값이 반환된다

예) setTimeout을 사용하면 주어진 함수가 n초 뒤에 실행된다

```
new Promise(resolve => {
  console.log("Promise executor");
  setTimeout(() => {
    // resolve가 호출되면 Promise의 상태가
    // pending에서 fulfilled로 바뀐다
    resolve("Bye, world!");
  }, 2000);
}).then(message => {
  // then은 resolve가 호출됐을 때 실행될 콜백을 설정한다
  console.log(message);
  // then에 넣은 콜백은 값을 반환할 수 있다.
  return "Foo, world!";
}).then(message => {
  // then을 계속해서 이어쓸 수 있다
  console.log(message);
});
```

```
console.log("Hello, world!");
```

```
new Promise((resolve, reject) => {
  // reject가 호출되면 Promise의 상태가
  // pending에서 rejected로 바뀐다
  reject("hello");
})
// reject가 호출되면 then에 넘겨진 부분은 실행되지 않는다
.then(message => console.log(message))
// reject가 호출되면 catch에 넘겨진 부분이 실행된다
.catch(message => alert(message));
```

Promise 활용하기

비동기 연산의 결과를 나타내는 Promise

연산의 결과가 함수 호출 직후 바로 나오지 않을 수 있다

예) fetch로 서버에 요청을 보내면 응답이 되돌아오기 전에 값이 반환된다

예) setTimeout을 사용하면 주어진 함수가 n초 뒤에 실행된다

```
new Promise(resolve => {  
  console.log("Promise executor");  
  setTimeout(() => {  
    // resolve가 호출되면 Promise의 상태가  
    // pending에서 fulfilled로 바뀐다  
    resolve("Bye, world!");  
  }, 2000);  
}).then(message => {  
  // then은 resolve가 호출됐을 때 실행될 콜백을 설정한다  
  console.log(message);  
  // then에 넣은 콜백은 값을 반환할 수 있다.  
  return "Foo, world!";  
}).then(message => {  
  // then을 계속해서 이어쓸 수 있다  
  console.log(message);  
});
```

```
console.log("Hello, world!");
```

```
new Promise((resolve, reject) => {  
  // reject가 호출되면 Promise의 상태가  
  // pending에서 rejected로 바뀐다  
  reject("hello");  
})  
// reject가 호출되면 then에 넘겨진 부분은 실행되지 않는다  
.then(message => console.log(message))  
// reject가 호출되면 catch에 넘겨진 부분이 실행된다  
.catch(message => alert(message));
```

Promise 활용하기

비동기 연산의 결과를 나타내는 Promise

연산의 결과가 함수 호출 직후 바로 나오지 않을 수 있다

예) fetch로 서버에 요청을 보내면 응답이 되돌아오기 전에 값이 반환된다

예) setTimeout을 사용하면 주어진 함수가 n초 뒤에 실행된다

```
new Promise(resolve => {
  console.log("Promise executor");
  setTimeout(() => {
    // resolve가 호출되면 Promise의 상태가
    // pending에서 fulfilled로 바뀐다
    resolve("Bye, world!");
  }, 2000);
}).then(message => {
  // then은 resolve가 호출됐을 때 실행될 콜백을 설정한다
  console.log(message);
  // then에 넣은 콜백은 값을 반환할 수 있다.
  return "Foo, world!";
}).then(message => {
  // then을 계속해서 이어쓸 수 있다
  console.log(message);
});
```

```
console.log("Hello, world!");
```

```
new Promise((resolve, reject) => {
  // reject가 호출되면 Promise의 상태가
  // pending에서 rejected로 바뀐다
  reject("hello");
})
// reject가 호출되면 then에 넘겨진 부분은 실행되지 않는다
.then(message => console.log(message))
// reject가 호출되면 catch에 넘겨진 부분이 실행된다
.catch(message => alert(message));
```

Promise 활용하기

비동기 연산의 결과를 나타내는 Promise

연산의 결과가 함수 호출 직후 바로 나오지 않을 수 있다

예) fetch로 서버에 요청을 보내면 응답이 되돌아오기 전에 값이 반환된다

예) setTimeout을 사용하면 주어진 함수가 n초 뒤에 실행된다

```
new Promise(resolve => {
  console.log("Promise executor");
  setTimeout(() => {
    // resolve가 호출되면 Promise의 상태가
    // pending에서 fulfilled로 바뀐다
    resolve("Bye, world!");
  }, 2000);
}).then(message => {
  // then은 resolve가 호출됐을 때 실행될 콜백을 설정한다
  console.log(message);
  // then에 넣은 콜백은 값을 반환할 수 있다.
  return "Foo, world!";
}).then(message => {
  // then을 계속해서 이어쓸 수 있다
  console.log(message);
});
```

```
console.log("Hello, world!");
```

```
new Promise((resolve, reject) => {
  // reject가 호출되면 Promise의 상태가
  // pending에서 rejected로 바뀐다
  reject("hello");
})
// reject가 호출되면 then에 넘겨진 부분은 실행되지 않는다
.then(message => console.log(message))
// reject가 호출되면 catch에 넘겨진 부분이 실행된다
.catch(message => alert(message));
```

Promise 활용하기

비동기 연산의 결과를 나타내는 Promise

연산의 결과가 함수 호출 직후 바로 나오지 않을 수 있다

예) fetch로 서버에 요청을 보내면 응답이 되돌아오기 전에 값이 반환된다

예) setTimeout을 사용하면 주어진 함수가 n초 뒤에 실행된다

```
new Promise(resolve => {
  console.log("Promise executor");
  setTimeout(() => {
    // resolve가 호출되면 Promise의 상태가
    // pending에서 fulfilled로 바뀐다
    resolve("Bye, world!");
  }, 2000);
}).then(message => {
  // then은 resolve가 호출됐을 때 실행될 콜백을 설정한다
  console.log(message);
  // then에 넣은 콜백은 값을 반환할 수 있다.
  return "Foo, world!";
}).then(message => {
  // then을 계속해서 이어쓸 수 있다
  console.log(message);
});
```

```
console.log("Hello, world!");
```

```
new Promise((resolve, reject) => {
  // reject가 호출되면 Promise의 상태가
  // pending에서 rejected로 바뀐다
  reject("hello");
})
// reject가 호출되면 then에 넘겨진 부분은 실행되지 않는다
.then(message => console.log(message))
// reject가 호출되면 catch에 넘겨진 부분이 실행된다
.catch(message => alert(message));
```

Promise 활용하기

비동기 연산의 결과를 나타내는 Promise

연산의 결과가 함수 호출 직후 바로 나오지 않을 수 있다

예) fetch로 서버에 요청을 보내면 응답이 되돌아오기 전에 값이 반환된다

예) setTimeout을 사용하면 주어진 함수가 n초 뒤에 실행된다

```
new Promise(resolve => {
  console.log("Promise executor");
  setTimeout(() => {
    // resolve가 호출되면 Promise의 상태가
    // pending에서 fulfilled로 바뀐다
    resolve("Bye, world!");
  }, 2000);
}).then(message => {
  // then은 resolve가 호출됐을 때 실행될 콜백을 설정한다
  console.log(message);
  // then에 넣은 콜백은 값을 반환할 수 있다.
  return "Foo, world!";
}).then(message => {
  // then을 계속해서 이어쓸 수 있다
  console.log(message);
});
```

```
console.log("Hello, world!");
```

```
new Promise((resolve, reject) => {
  // reject가 호출되면 Promise의 상태가
  // pending에서 rejected로 바뀐다
  reject("hello");
})
// reject가 호출되면 then에 넘겨진 부분은 실행되지 않는다
.then(message => console.log(message))
// reject가 호출되면 catch에 넘겨진 부분이 실행된다
.catch(message => alert(message));
```


Promise 활용하기

async/await으로 Promise를 더 간편하게 쓸 수 있다

async → 함수 내에서 await을 사용할 수 있도록 해준다 (Promise 반환)

await → Promise의 상태가 fulfilled 또는 rejected가 될 때까지 기다린다

async/await을 사용했을 때 :

// 주어진 시간 이후 fulfilled가 되는 Promise를 반환하는 함수

```
function sleep(ms) {  
  return new Promise(  
    resolve => setTimeout(resolve, ms);  
  )  
}
```

```
async function introduce() {  
  console.log("Hello.");  
  await sleep(1000);  
  console.log("My name is Minsu.");  
  await sleep(2000);  
  return "Bye.";  
}
```

```
introduce().then(message => console.log(message));
```

async/await을 사용하지 않았을 때 :

// 주어진 시간 이후 fulfilled가 되는 Promise를 반환하는 함수

```
function sleep(ms) {  
  return new Promise(  
    resolve => setTimeout(resolve, ms);  
  )  
}
```

```
function introduce() {  
  console.log("Hello.");  
  return sleep(1000).then(() => {  
    console.log("My name is Minsu.");  
    return sleep(2000).then(() => {  
      return "Bye.";  
    })  
  })  
}
```

```
introduce().then(message => console.log(message));
```

Promise 활용하기

Promise를 resolve할 수 있는 v8::Promise::Resolver

v8::Promise::Resolver::Resolve로 Promise를 fulfilled된 상태로 만든다
v8::Promise::Resolver::Reject로 Promise를 rejected된 상태로 만든다
생성시 pending 상태의 v8::Promise를 함께 생성한다

// v8::Promise::Resolver 생성

```
v8::Local<v8::Promise::Resolver> resolver  
    = v8::Promise::Resolver::New(context).ToLocalChecked();
```

// Resolver와 함께 생성된 Promise 가져오기

```
v8::Local<v8::Promise> promise = resolver->GetPromise();
```

// Promise의 상태를 fulfilled로 바꾼다

```
resolver->Resolve(  
    context,  
    v8::String::NewFromUtf8Literal("Hello, world!").ToChecked());
```

Promise 활용하기

sleep 함수 만들기

setTimeout 구현 :

```
struct TimeoutQueueItem {  
    chrono::time_point<chrono::steady_clock> firesAt;  
    v8::Global<v8::Function> functionToCall;  
    std::vector<v8::Global<v8::Value>> arguments;  
};
```

```
using TimeoutQueueItemPtr  
    = std::shared_ptr<TimeoutQueueItem>;
```

```
struct TimeoutQueueItemComparator {  
    constexpr bool operator()(  
        TimeoutQueueItemPtr const& lhs,  
        TimeoutQueueItemPtr const& rhs) const {  
        return lhs->firesAt > rhs->firesAt;  
    }  
};
```

sleep 구현 :

```
struct SleepQueueItem {  
    chrono::time_point<chrono::steady_clock> firesAt;  
    v8::Global<v8::Promise::Resolver> resolver;  
};
```

```
using SleepQueueItemPtr  
    = std::shared_ptr<SleepQueueItem>;
```

```
struct SleepQueueItemComparator {  
    constexpr bool operator()(  
        SleepQueueItemPtr const& lhs,  
        SleepQueueItemPtr const& rhs) const {  
        return lhs->firesAt > rhs->firesAt;  
    }  
};
```

Promise 활용하기

sleep 함수 만들기

setTimeout 구현 :

```
// 인자들을 v8::Global로 변환
std::vector<v8::Global<v8::Value>> arguments;
for (int i = 2; i < info.Length(); ++i)
    arguments.push_back(
        v8::Global<v8::Value> { isolate, info[i] });

timeoutQueue.push(TimeoutQueueItemPtr {
    new TimeoutQueueItem {
        chrono::time_point_cast<chrono::milliseconds>
            (willFireAt),
        // 인자와 함수를 함께 큐에 넣음
        v8::Global<v8::Function>(isolate, function),
        std::move(arguments),
    },
});
```

sleep 구현 :

```
// Resolver 생성
v8::Local<v8::Promise::Resolver> resolver
    = v8::Promise::Resolver::New(context).ToLocalChecked();

sleepQueue.push(SleepQueueItemPtr {
    new SleepQueueItem {
        chrono::time_point_cast<chrono::milliseconds>
            (willFireAt),
        // 함수 대신 resolver를 v8::Global로 변환
        v8::Global<v8::Promise::Resolver> { isolate, resolver },
    },
});

// Promise 반환
info.GetReturnValue().Set(resolver->GetPromise());
```

Promise 활용하기

sleep 함수 만들기

setTimeout 구현 :

```
while (!timeoutQueue.empty()) {
    // 큐에서 실행할 함수 꺼내기
    auto item = std::move(*timeoutQueue.top());
    timeoutQueue.pop();
    // 실행 시점이 나중이라면 기다린다
    if (item.firesAt >= chrono::static_clock::now())
        std::this_thread::sleep_until(item.firesAt);
    // 인자들을 v8::Local로 변환한다 (중략)
    // 함수를 실행한다
    item.functionToCall.Get(isolate)->CallAsFunction(
        context,
        context->Global(),
        (int)arguments.size(),
        arguments.data());
}
```

sleep 구현 :

```
while (!sleepQueue.empty()) {
    // 큐에서 resolve할 Resolver 꺼내기
    auto item = std::move(*sleepQueue.top());
    sleepQueue.pop();

    // 실행 시점이 나중이라면 기다린다
    if (item.firesAt >= chrono::static_clock::now())
        std::this_thread::sleep_until(item.firesAt);

    // Promise의 상태를 fulfilled로 변경한다
    item.resolver.Get(isolate)->Resolve(
        context, v8::Undefined(isolate)).ToChecked();
}
```

Promise 활용하기

Examples/6-promises

```
Microsoft Visual Studio Debug Console

> async function introduce() {
>   console.log("Hello.");
>   await sleep(1000);
>   console.log("My name is Minsu.");
>   await sleep(2000);
>   return "Bye";
> }
> introduce().then(message => console.log(message));
>
Hello.
Execution complete: [object Promise]
sleep resolved
My name is Minsu.
sleep resolved
Bye
> .exit

C:\Users\freiy\Desktop\easy-v8-embedding\out\build\x64-Debug\Examples\6-promises\6-promises.exe (process 7172) exited with code 0.
```

Promise 활용하기

MicrotaskQueue

마이크로태스크 : 태스크가 실행되기 전에 실행되는 태스크
Promise.then에 넘겨진 콜백은 태스크 큐가 아니라 마이크로태스크 큐에 push된다

// 수동으로 큐를 pop 해주는 마이크로태스크 큐 생성

```
std::unique_ptr<v8::MicrotaskQueue> microtaskQueue  
    = v8::MicrotaskQueue::New(isolate, v8::MicrotasksPolicy::kExplicit);
```

// Context 생성할 때 넘겨줄 수 있다

```
v8::Local<v8::Context> context = v8::Context::New(  
    isolate, nullptr, global,  
    v8::MaybeLocal<v8::Value> {},  
    v8::DeserializeInternalFieldsCallback {},  
    microtaskQueue.get());
```

// 스크립트 실행 (종락)

// 마이크로태스크 큐에 쌓인 태스크 모두 실행

```
microtaskQueue->PerformCheckpoint(isolate);
```

C++ 클래스를 자바스크립트에서 사용하기

클래스 c 함수

function으로 클래스 만들기 :

```
function Person(name) { this.name = name; }  
  
Person.prototype.introduce = function() {  
    alert(`Hello, my name is ${this.name}!`);  
}
```

```
// Hello, my name is Minsu!  
new Person("Minsu").introduce();
```

```
// function  
alert(typeof Person);
```

class로 클래스 만들기 (ES6) :

```
class Person {  
    constructor(name) { this.name = name; }  
    introduce() {  
        alert(`Hello, my name is ${this.name}!`);  
    }  
}
```

```
// Hello, my name is Minsu!  
new Person("Minsu").introduce();
```

```
// function  
alert(typeof Person);
```


C++ 클래스를 자바스크립트에서 사용하기

스크립트에서 접근하지 못하는 필드 만들기

v8::ObjectTemplate::SetInternalFieldCount로 필드의 개수 설정
v8::Object::SetInternalField로 v8::Local<v8::Value>를 필드에 넣는다
v8::Object::GetInternalField로 넣었던 값을 가져온다

v8::Object::SetAlignedPointerInInternalField로 포인터 하나를 필드에 넣는다
V8::Object::GetAlignedPointerFromInternalField로 넣었던 포인터를 가져온다
저장할 포인터는 2 바이트 단위로 정렬되어(2-byte-aligned)있어야 한다

```
v8::Local<v8::ObjectTemplate> object = v8::ObjectTemplate::New(isolate);
object->SetInternalFieldCount(1);

v8::Local<v8::Object> instance = object->NewInstance(context).ToLocalChecked();
instance->SetInternalField(0, v8::String::NewFromUtf8Literal(isolate, "Hello"));

char buffer[6];

v8::Local<v8::String> value = instance->GetInternalField(0).As<v8::String>();
value->WriteUtf8(isolate, buffer);
assert(strcmp(buffer, "Hello") == 0);
```

C++ 클래스를 자바스크립트에서 사용하기

SetWeak으로 finalization 콜백 설정하기

Finalization : GC에서 객체를 수집하는 시점
파일, 메모리 등의 자원을 가지고 있는 객체가 자원을 정리할 수 있게 해준다
Finalizer는 언제 호출될 지 알 수 없으며, 호출되지 않을 수도 있다
객체가 가지고 있던 자원을 정리하는 방법을 따로 마련해야한다 (cf. C# IDisposable.Dispose)
ClearWeak을 사용하면 GC가 객체를 수집하는 것을 막을 수 있다

```
struct Person {  
    std::string name;  
    v8::Global<v8::Object> global;  
  
    Person(v8::Isolate* isolate, v8::Local<v8::Object> local, const char* name)  
        : name { name }, global { isolate, local }  
    {  
        // Finalization 콜백 설정  
        global.SetWeak(this, WeakHandler, v8::WeakCallbackType::kParameter);  
    }  
  
    static void WeakHandler(v8::WeakCallbackInfo<Person> const& data) {  
        // Person::~Person 호출  
        delete data.GetParameter();  
    }  
};
```

C++ 클래스를 자바스크립트에서 사용하기

v8::FunctionCallbackInfo<T>::This()

함수 내에서 this 값을 가져오는데 사용

```
// 생성자
void Constructor(v8::FunctionCallbackInfo<v8::Value> const& info)
{
    v8::Local<v8::Value> name = info[0];
    info.This()->SetAlignedPointerInInternalField(0, new Person {
        isolate,
        info.This(),
        *v8::String::Utf8Value { isolate, name },
    });
}

// `Hello, my name is ${this.name}.` 출력
void Introduce(v8::FunctionCallbackInfo<v8::Value> const& info) {
    void* ptr = info.This()->GetAlignedPointerFromInternalField(0);
    Person* person = static_cast<Person*>(ptr);
    std::cout << "Hello, my name is " << person->name << ".";
}
```

C++ 클래스를 자바스크립트에서 사용하기

v8::FunctionTemplate으로 생성자 만들기

v8::Context를 생성하기 전 클래스를 미리 전역에 정의하기 위해 v8::FunctionTemplate를 사용할 수 있다

```
// Person 생성자
v8::Local<v8::FunctionTemplate> constructor = v8::FunctionTemplate::New(isolate, Constructor);
// Person.prototype
v8::Local<v8::ObjectTemplate> prototype = constructor->PrototypeTemplate();
// Person.prototype.introduce = function() { ... }
prototype->Set(isolate, "introduce", v8::FunctionTemplate::New(isolate, Introduce));

// 포인터를 저장하기 위해 internal 필드 개수 1로 설정
v8::Local<v8::ObjectTemplate> instance = constructor->InstanceTemplate();
instance->SetInternalFieldCount(1);

// 전역에 Person 생성자 정의
v8::Local<v8::ObjectTemplate> global = v8::ObjectTemplate::New(isolate);
global->Set(isolate, "Person", constructor);
```

C++ 클래스를 자바스크립트에서 사용하기

getter, setter

프로퍼티를 가져오거나 프로퍼티에 값을 설정할 때 대신 호출될 함수 (cf. C# 프로퍼티)

```
const obj = {  
  get foo() {  
    console.log("get foo()");  
    return 15;  
  },  
  set foo(value) {  
    console.log(`set foo(${value})`);  
  }  
};
```

```
obj.foo = 25;  
console.log(obj.foo);
```

```
// set foo(25)  
// get foo()  
// 15
```

C++ 클래스를 자바스크립트에서 사용하기

Accessor로 멤버 변수에 접근할 수 있도록 하기

```
// Person::name을 반환한다
void GetName(v8::Local<v8::String>          property,
             const v8::PropertyCallbackInfo<v8::Value>& info) {
    void* ptr = info.This()->GetAlignedPointerFromInternalField(0);
    Person* person = static_cast<Person*>(ptr);
    info.GetReturnValue().Set(
        v8::String::NewFromUtf8(isolate, person->name.c_str()).ToLocalChecked());
}

// Person::name에 값을 설정한다
void SetName(v8::Local<v8::String>          property,
             v8::Local<v8::Value>          value,
             const v8::PropertyCallbackInfo<void>& info) {
    void* ptr = info.This()->GetAlignedPointerFromInternalField(0);
    Person* person = static_cast<Person*>(ptr);
    person->name = *v8::String::Utf8Value(isolate, value);
}

// get name(), set name() 설정
v8::Local<v8::FunctionTemplate> constructor = v8::FunctionTemplate::New(isolate, Constructor);
v8::Local<v8::ObjectTemplate> prototype = constructor->PrototypeTemplate();
prototype->SetAccessor(v8::String::NewFromUtf8Literal(isolate, "name"), GetName, SetName);
```

C++ 클래스를 자바스크립트에서 사용하기

프로그램 종료시 생성됐던 객체들을 모두 소멸시키기

v8::PersistentBase::SetWrapperClassId 로 wrapper class ID를 설정할 수 있다
V8::PersistentHandleVisitor를 상속하여 모든 v8::Persistent 핸들을 방문할 수 있다

```
uint16_t Person::Tag = /* Some constant here */;

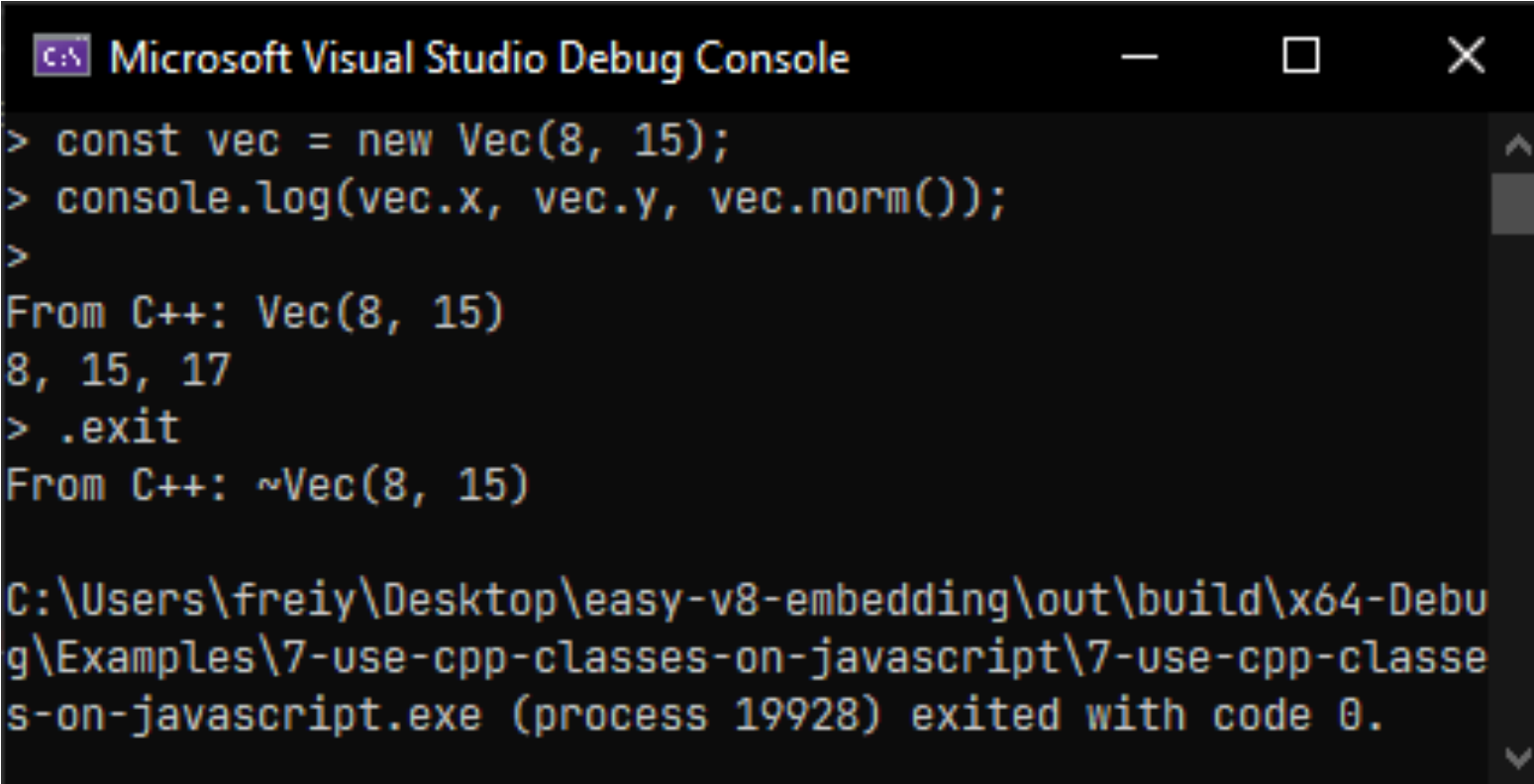
Person::Person() {
    global.SetWeak(this, WeakHandler, v8::WeakCallbackType::kParameter);
    // wrapper class ID를 설정한다
    global.SetWrapperClassId(Person::Tag);
}

struct HandleVisitor : public v8::PersistentHandleVisitor {
    // 핸들을 방문할 때 wrapper class ID가 함께 넘어온다
    virtual void VisitPersistentHandle(v8::Persistent<v8::Value>* value, uint16_t classId) override {
        v8::Local<v8::Context> current = _isolate->GetCurrentContext();
        if (classId == Person::Tag) {
            auto local = value->Get(_isolate);
            void* ptr = local.As<v8::Object>()->GetAlignedPointerFromInternalField(0);
            delete static_cast<Person*>(ptr);
        }
    }
};

// 모든 핸들 방문
Person::HandleVisitor visitor { isolate };
isolate->VisitHandlesWithClassIds(&visitor);
```

C++ 클래스를 자바스크립트에서 사용하기

Examples/7-use-cpp-classes-on-javascript



```
Microsoft Visual Studio Debug Console

> const vec = new Vec(8, 15);
> console.log(vec.x, vec.y, vec.norm());
>
From C++: Vec(8, 15)
8, 15, 17
> .exit
From C++: ~Vec(8, 15)

C:\Users\freiy\Desktop\easy-v8-embedding\out\build\x64-Debug\Examples\7-use-cpp-classes-on-javascript\7-use-cpp-classes-on-javascript.exe (process 19928) exited with code 0.
```


번외

반환 값 하나를 내보낼 수 있는 EscapableHandleScope

HandleScope가 소멸되면 새로 생성된 v8::Local<T>에 접근할 수 없다
HandleScope 밖에 있는 HandleScope로 값을 내보내는 기능이 필요하다
→ EscapableHandleScope를 사용한다

EscapableHandleScope를 사용했을 때 :

```
v8::Local<v8::Value> SomeFunc() {  
  v8::EscapableHandleScope handleScope { isolate };  
  v8::Local<v8::Value> newValue = MakeNewValue();  
  // newValue 핸들을 바깥쪽 HandleScope으로 이동시킨다  
  return handleScope.Escape(newValue);  
}
```

```
v8::Local<v8::Value> value = SomeFunc();  
value->DoSomeOperation(); // Ok
```

HandleScope만 사용했을 때 :

```
v8::Local<v8::Value> SomeFunc() {  
  v8::HandleScope handleScope { isolate };  
  v8::Local<v8::Value> newValue = MakeNewValue();  
  
  return newValue;  
}
```

```
v8::Local<v8::Value> value = SomeFunc();  
value->DoSomeOperation(); // UB
```

번외

포인터를 감싸는 v8::External

v8::Value의 자식 클래스 → v8::Local<v8::Value>에 v8::Local<v8::External>를 대입할 수 있다
v8::External::New로 포인터를 하나 넘길 수 있다
v8::External::Value로 포인터를 다시 가져올 수 있다

```
v8::Local<v8::External> external  
= v8::External::New(isolate, new std::string("Hello"));  
  
assert(*static_cast<std::string*>(external->Value()) == "Hello");
```

번외

임베더 데이터를 저장하는 기능

v8::Context::SetAlignedPointerInEmbedderData로 포인터 하나를 저장할 수 있다
v8::Context::GetAlignedPointerFromEmbedderData로 저장한 포인터를 꺼내올 수 있다
저장할 포인터는 2 바이트 단위로 정렬되어(2-byte-aligned)있어야 한다

v8::Context::SetEmbedderData로 v8::Local<v8::Value> 하나를 저장할 수 있다
꺼내올 때는 v8::Context::GetEmbedderData를 사용한다
사용법은 SetAlignedPointerInEmbedderData와 동일하다

// index에 큰 값이 들어오면 v8이 알아서 저장공간을 늘려준다

```
context->SetAlignedPointerInEmbedderData(  
    /* index */ 200, new std::string("Hello, world!"));
```

// 저장한 포인터를 가져온다

```
void* ptr = context->GetAlignedPointerFromEmbedderData(200);  
assert(*static_cast<std::string*>(ptr) == "Hello, world!");
```

번외

WebAssembly / V8 비교

V8 : C++로 만든 ECMAScript와 WebAssembly를 구현한 자바스크립트 엔진

WebAssembly : 스택 기반 가상 머신 명령어 포맷

C++, C, Rust 등 다양한 언어를 WebAssembly를 타겟으로 컴파일 할 수 있다

주요 브라우저들이 모두 WebAssembly 지원

자바스크립트에서 WebAssembly API를 이용하여 사용 가능

브라우저 없이도, 자바스크립트 없이도 WebAssembly 사용 가능 (wasmtime)

<https://webassembly.studio/?f=vjpohwvki5r>

참고자료

Claudio De Sio on Twitter: "Thanks for inventing #javascript! ;-)"
- <https://twitter.com/cdesio/status/1013166206877163520>

Building V8 from source
- <https://v8.dev/docs/build>

Getting started with embedding V8
- <https://v8.dev/docs/embed>

V8 wrapped objects lifecycle
- <https://itnext.io/v8-wrapped-objects-lifecycle-42272de712e0>

Node.js JavaScript runtime
- <https://github.com/nodejs/node>

Promise() constructor
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise/Promise

ECMA-262
- <https://www.ecma-international.org/publications-and-standards/standards/ecma-262/>

V8 API Reference Guide
- <https://v8docs.nodesource.com/node-15.0/>

CppKorea Facebook Community Group

CppKorea8thSeminar.cpp - Session End

쉽게 따라하는 V8 임베딩

C++에서 자바스크립트
돌리기