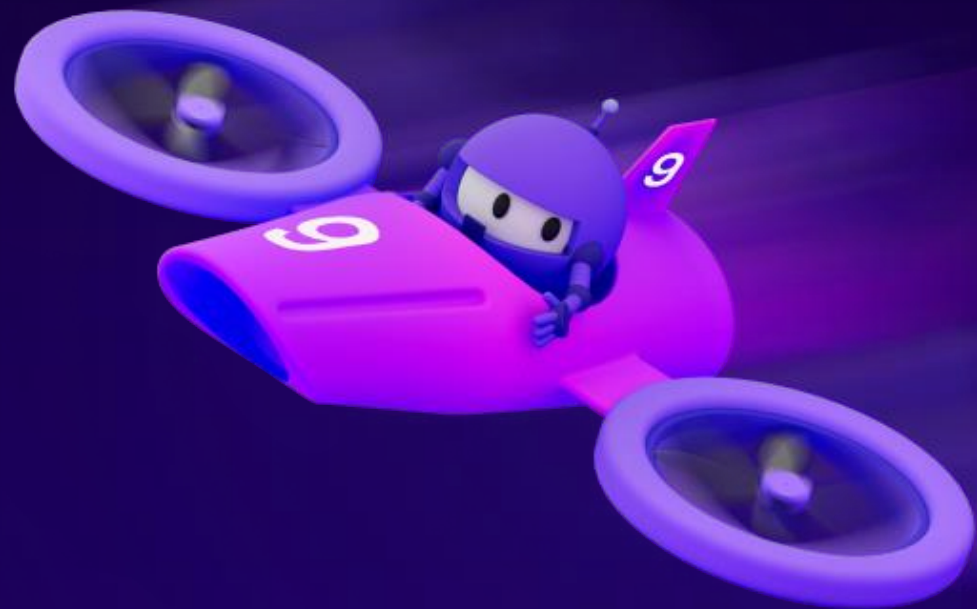


.NET + Node.js

괴롭지만 함께라면 행복할거야

박상현



Agenda

- 발표자 소개
- Node.js 란?
- Node.js C++ Addon
- 네이티브 브릿지로서의 .NET
- 나야, Node API for .NET
- P/Invoke(데모)
- JavaScript에서 C# 메소드 호출하기(데모)
- TypeScript를 위한 타입 생성(데모)
- Node.js를 .NET에 임베딩(데모)
- .NET AOT(데모)

발표자

박상현

✓ 경력

- ✓ 現 소프트웨어 엔지니어 @비공개 스타트업
- ✓ 前 삼성탈레스/한화시스템 수석 연구원

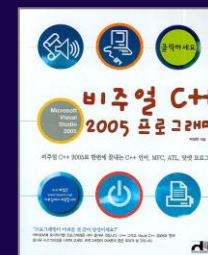
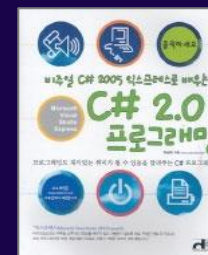
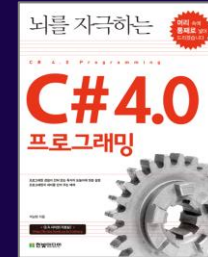
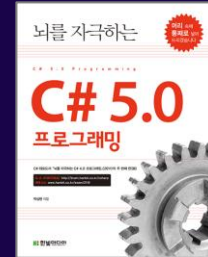
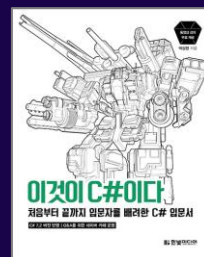
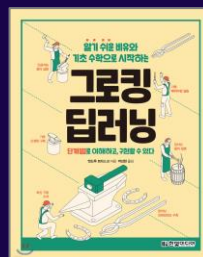
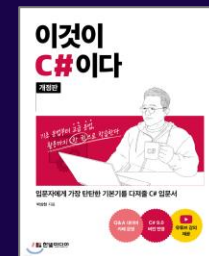
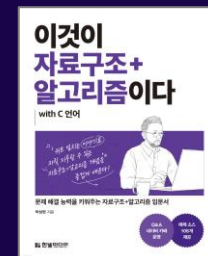
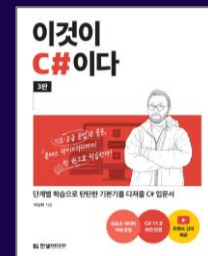
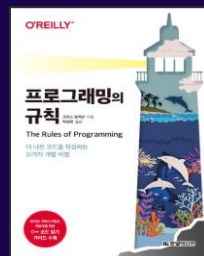
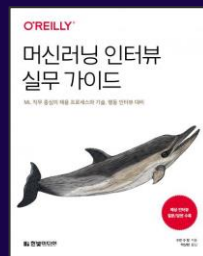
✓ 역서 및 저서

- ✓ <이것이 C#이다> 외 13권
- ✓ 2010, <뇌를 자극하는 알고리즘> 대한민국 학술원 우수학술도서
- ✓ 2020, <그로킹 딥러닝> 세종 학술부문 우수 도서

✓ 연락처

✉ seanlab@gmail.com

in <https://www.linkedin.com/in/swseanlab/>



다음과 같은 고민을
하시는 분들에게



Node.js → .NET 마이그레이션

- Node.js로 작성된 애플리케이션을 .NET 기반으로 옮기는 작업을 가능하게 함.
- 보다 강력한 타이핑을 통한 코드 유지 보수성 향상.
- .NET에서 제공하는 풍부한 라이브러리와 상용/오픈소스 라이브러리 및 강력한 IDE 지원 활용.

JavaScript의 네이티브 코드 접근

- JavaScript 코드가 OS 레벨/네이티브 코드에 접근할 수 있게 함.
- 기존의 네이티브 라이브러리(예: OpenCV 등)를 JavaScript에서 사용 가능.
- C/C++ 또는 Rust를 사용하지 않고 C#만으로 JavaScript 코드에서 네이티브 코드에 접근할 수 있게 함.

.NET → Node.js 마이그레이션

- .NET(C#, VB 등)으로 작성된 코드를 Node.js 기반으로 이관하는 작업을 가능하게 함.
- JavaScript 기반의 넓은 생태계(NPM 모듈, 프론트엔드/백엔드 통일) 활용 가능.

Node.js + .NET 상호운용

- Node.js와 .NET 코드를 '동시에' 활용할 수 있도록 상호운용성을 제공할 수 있음.
- .NET 라이브러리(예: 시스템 API, NuGet 패키지)의 재사용과 Node.js 생태계(NPM 모듈)를 동시에 누림.
- 프로젝트 전환 없이 필요한 부분만 Node.js 프로젝트에 .NET을 도입하거나, .NET 프로젝트에 Node.js를 도입할 수 있음.



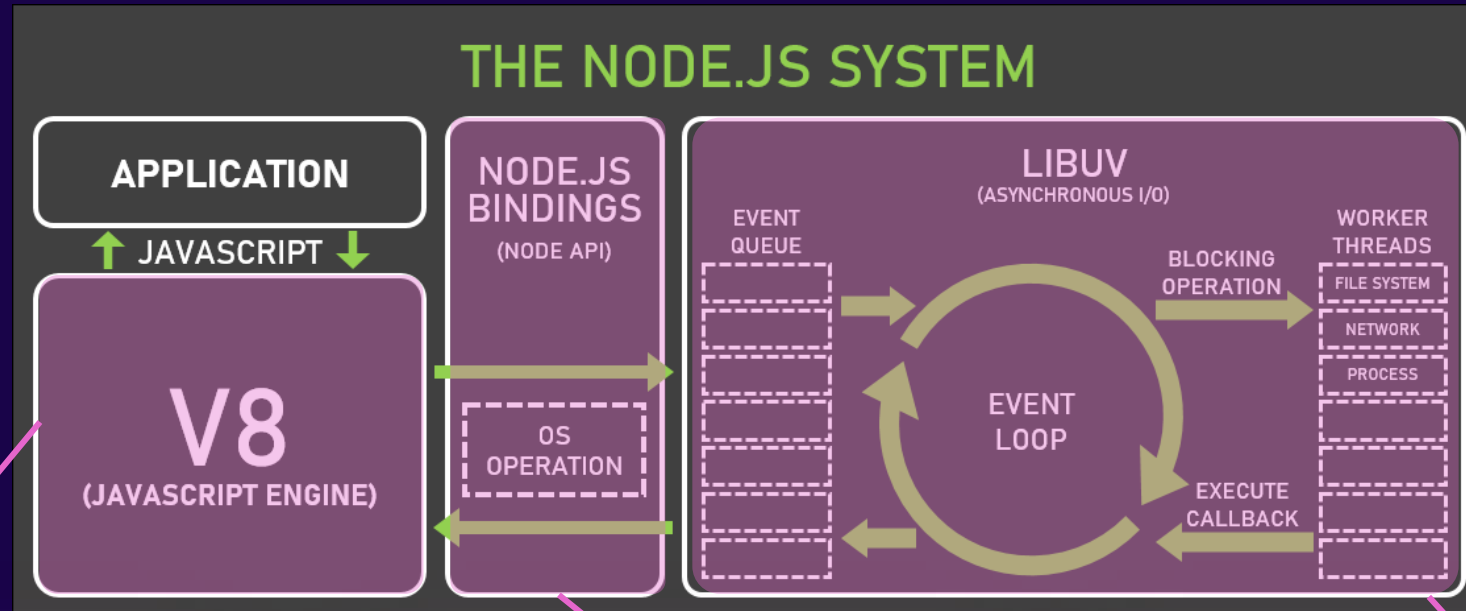
Node.js

Node.js란?

- 서버사이드 자바스크립트 런타임
 - ✓ 웹 브라우저가 아닌 서버 사이드에서 JavaScript 코드를 구동.
- 크로스 플랫폼 지원
 - ✓ *Windows, Linux, macOS* 등 다양한 OS에서 동작.
 - ✓ ARM, x86 등 다양한 아키텍처 지원.
 - ✓ 개발 및 배포 환경에 제약이 적음.
- 단일 이벤트 루프와 비동기(Non-blocking) I/O
 - ✓ Libuv 라이브러리를 활용하여 단일 이벤트 루프 기반으로 비동기 I/O 작업을 처리하며, 높은 동시성을 제공.
- NPM(Node Package Manager) 생태계
 - ✓ 방대한 오픈소스 모듈을 쉽게 설치 및 사용 가능.
 - ✓ 프론트엔드, 백엔드 모두에서 JavaScript 라이브러리 재사용 용이.

Node.js의 두 심장: V8과 libuv

출처: <https://www.tutorialandexample.com/node-js-event-loop>



JavaScript를 컴파일하고 실행

Node.js Binding(예: fs)에 의존하지 않는 코드는 이 단계에서 실행이 완료됨

- 예: 산술 연산, 변수 선언, 함수 호출 등.

네이티브 코드 실행

네트워킹(net, dgram, http), 파일 I/O(fs) 등 Node.js가 제공하는 빌트인 바인딩을 실행함.

이들 바인딩은 libuv의 함수를 호출하여 이벤트 큐에 비동기 I/O 작업을 입력함

- 예: fs.readFile 호출 → uv_fs_read를 통해 비동기 파일 읽기 작업 등록.

이벤트 큐와 스레드 풀

Node.js 바인딩에서 이벤트 큐에 입력한 비동기 I/O작업을 스레드 풀을 이용하여 처리.

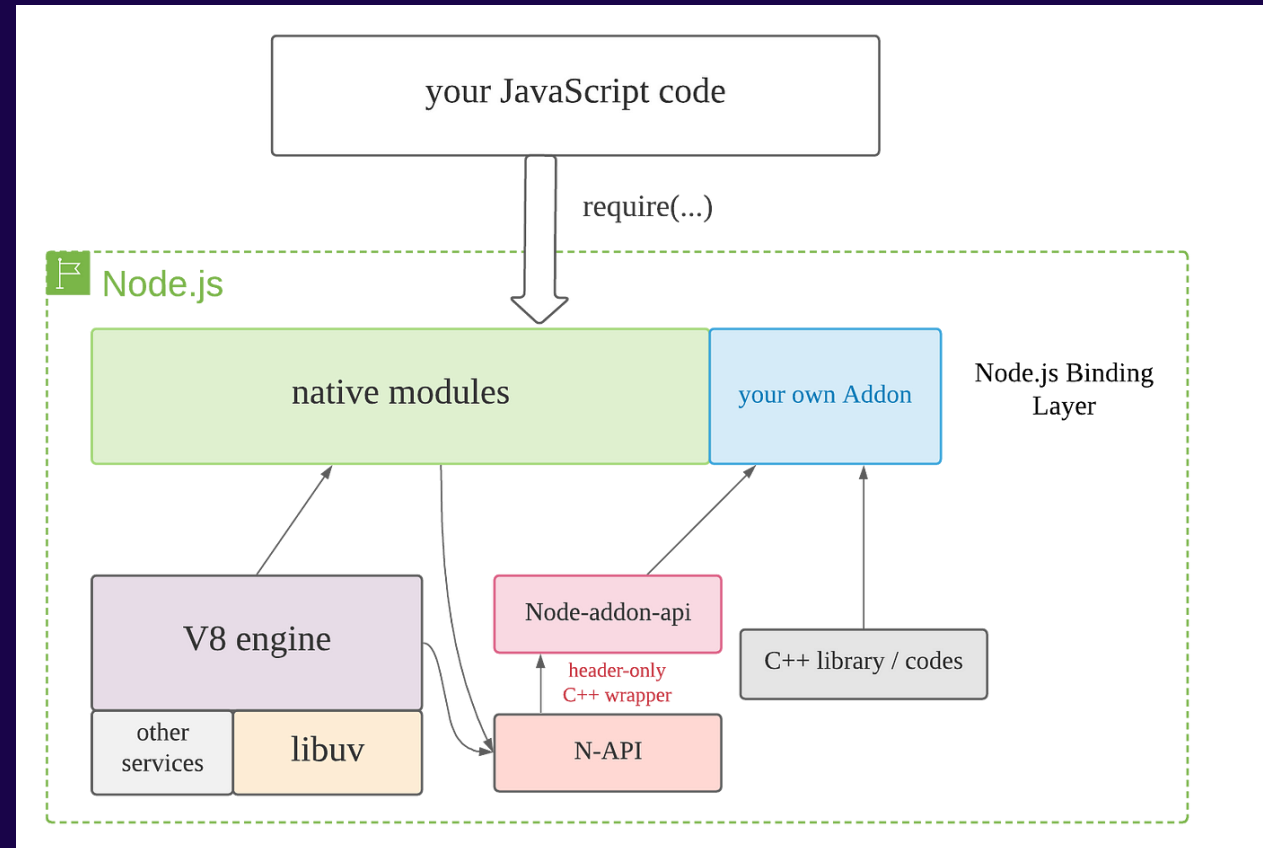
- I/O 작업: OS 레벨 이벤트 메커니즘을 활용 (e.g., epoll, kqueue, IOCP).

- CPU 작업: Libuv 스레드 풀을 사용하여 실행.(e.g. 암호화)

처리가 완료되면 콜백을 통해 실행 결과가 JavaScript 세계로 전달되게 함.

C++ Addon: Node.js 내장 모듈이 2% 부족할 때

- “Addon은 C++로 작성된 동적 링크 공유 객체” (Node.js 공식 문서)
- 동적 링크 공유 객체라는 이름을 OS 별로 풀어보면:
 - 윈도우 OS에서는 동적 링크 라이브러리(.DLL)
 - 리눅스에서는 공유 객체(.so)
 - macOS에서는 동적 라이브러리(.dylib)
- Addon으로 빌드했을 때 확장자는 .node으로 변경됨
- JavaScript 코드에서는 require() 함수를 이용하여 로딩



출처: <https://medium.com/ai-innovation/a-guide-for-javascript-developers-to-build-c-add-ons-with-node-addon-api-28c84a0c0cb1>

언제 Node.js C++ Addon을 구현해야 할까?

- CPU 바운드 작업이 많은 Node.js 애플리케이션의 성능을 끌어올려야 할 때
- 의존하는 SDK 또는 라이브러리가 C/C++만 지원할 때
- OS의 기능에 접근하고자 할 때
- 로봇, IoT 디바이스의 하드웨어를 제어하고자 할 때
- ...

Node.js C++ Addon을 만드는 세 가지 방법

1. C++용 V8 API, libuv API, Node.js API를 직접 사용

- ✓ 주의: V8 API는 Breaking Change가 많음

2. Native abstractions for Node.js(a.k.a nan)

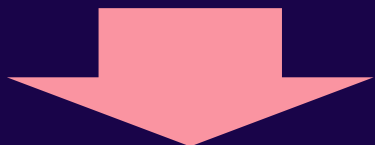
- ✓ V8의 변화에 따라 Addon을 수정해야 하는 문제를 제거하고자 만들어진 추상화 라이브러리
- ✓ Node.js 버전이 바뀌면 nan도 업데이트 될 수 있으며, 이 경우 Addon을 수정하거나 다시 빌드해야 함

3. ★ Node-API(a.k.a N-API) ★

- ✓ Node.js 버전에 관계 없이 사용할 수 있는 ABI(Application Binary Interface) 라이브러리
- ✓ 신규 Node.js 버전이 나오더라도 Addon을 수정하거나 다시 빌드할 필요 없음

N-API: “편의가 목적이라고 한적은 없다.”

- N-API의 목적은 Node.js 버전, 컴파일러에 관계 없이 ABI 안정성을 제공하는 것
- Node-API는 C 스타일의 API로 이뤄져 있음
- 메모리 관리와 에러 핸들링에서 많은 주의가 필요



그래서 **node-addon-api** 를 사용합니다.

- N-API의 C++ Wrapper
- 코드의 양을 획기적으로 줄임
- 당연히 C API를 통해 제공되는 ABI 안정성의 혜택은 그대로



N-API와 node-addon-api 코드 비교

N-API	node-addon-api
<pre>napi_status status; napi_value object, string; status = napi_create_object(env, &object); if (status != napi_ok) { napi_throw_error(env, ...); return; } status = napi_create_string_utf8(env, "bar", NAPI_AUTO_LENGTH, &string); if (status != napi_ok) { napi_throw_error(env, ...); return; } status = napi_set_named_property(env, object, "foo", string); if (status != napi_ok) { napi_throw_error(env, ...); return; }</pre>	<pre>Object obj = Object::New(env); obj["foo"] = String::New(env, "bar");</pre>

우리 .NET이 네이티브를 너무
잘해요

.NET/네이티브 통합

Windows, Linux,
macOS 에서
사용 가능

P/Invoke

간단한 선언으로 C 함수 호출

COM Interop

TLB를 바탕으로 interop 어셈블리 자동 생성

C++/CLI

C++ 코드, 네이티브 라이브러리, COM
컴포넌트 모두 통합 가능

Windows 전용

데모1: P/Invoke (macOS/Silicon)

```
[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Ansi)]
public struct Utsname
{
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 256)]
    public string sysname;
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 256)]
    public string nodename;
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 256)]
    public string release;
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 256)]
    public string version;
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 256)]
    public string machine;
}
```

```
[DllImport("libc")]
public static extern int uname(out Utsname buf);
```

```
Utsname uts;
if (uname(out uts) == 0)
{
    Console.WriteLine($"System Name: {uts.sysname}");
    Console.WriteLine($"Node Name: {uts.nodename}");
    Console.WriteLine($"Release: {uts.release}");
    Console.WriteLine($"Version: {uts.version}");
    Console.WriteLine($"Machine: {uts.machine}");
}
else
{
    Console.WriteLine("uname call failed");
}
```

macOS에서 libc는
/usr/lib/libSystem.dylib 에 있음

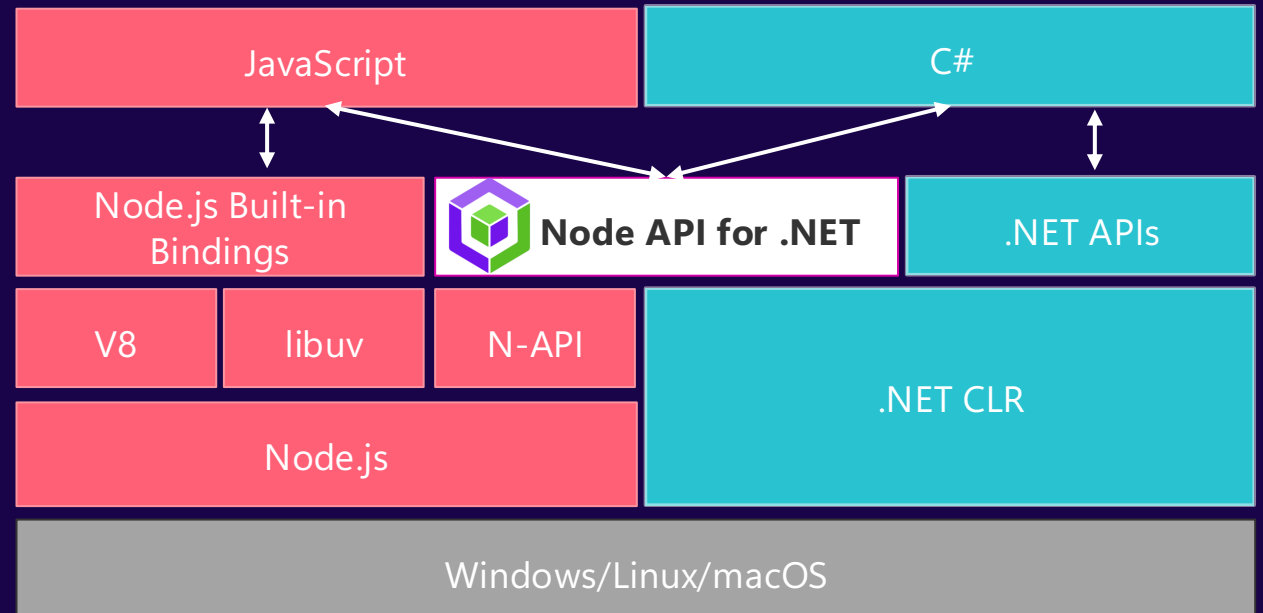
나야,  Node API for .NET.

Node API for .NET



Node API for .NET 은 한 프로세스 안에서 .NET과 JavaScript 간의 고급 상호운용을 가능하게 합니다.

* 출처: Node API for .NET README.md



시나리오에 따라 NPM또는 Nuget 사용

NPM

- <https://www.npmjs.com/package/node-api-dotnet>
- JavaScript에서 .NET 코드를 호출하는 경우에 사용

Nuget

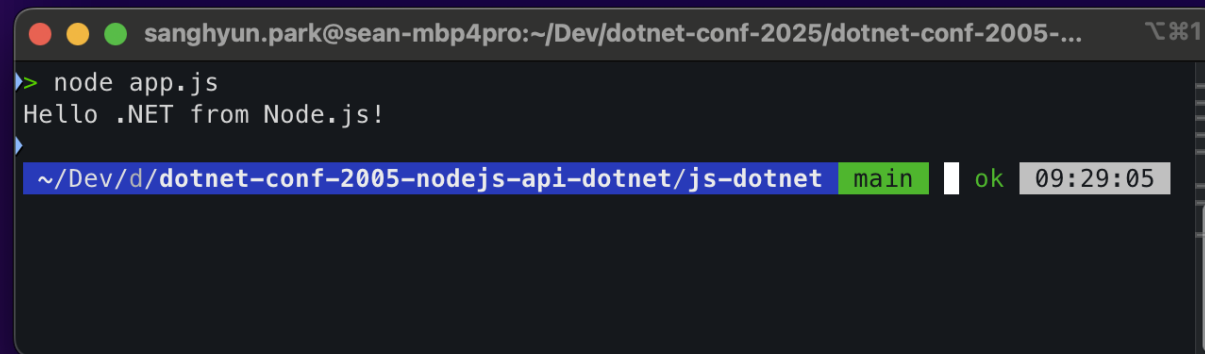
- <https://www.nuget.org/packages/Microsoft.JavaScript.NodeApi/>
- .NET에서 JavaScript 코드를 호출하는 경우
- .NET에서 Node.js용 Addon 모듈을 작성하는 경우

리포지토리: <https://github.com/microsoft/node-api-dotnet>

JavaScript에서 .NET 메소드 호출

데모2: JavaScript에서 .NET 메소드 호출 #1

```
/*  
https://github.com/sean-lab/dotnet-conf-2025-nodejs-api-dotnet/blob/main/js-dotnet/app.js  
*/  
  
import dotnet from 'node-api-dotnet/net8.0';  
  
const Console = dotnet.System.Console;  
Console.WriteLine('Hello .NET from Node.js!');
```



A terminal window with a dark background. The title bar shows 'sanghyun.park@sean-mbp4pro:~/Dev/dotnet-conf-2025/dotnet-conf-2005-...'. The prompt is '>'. The command 'node app.js' has been entered and executed. The output is 'Hello .NET from Node.js!'. Below the command line, there is a status bar showing the file path '~/Dev/d/dotnet-conf-2005-nodejs-api-dotnet/js-dotnet', the branch 'main', a green circle icon, the word 'ok', and a timestamp '09:29:05'.

```
sanghyun.park@sean-mbp4pro:~/Dev/dotnet-conf-2025/dotnet-conf-2005-...  
> node app.js  
Hello .NET from Node.js!  
~/Dev/d/dotnet-conf-2005-nodejs-api-dotnet/js-dotnet main ok 09:29:05
```

데모3: JavaScript에서 .NET 메소드 호출 #2-1

```
/*  
https://github.com/sean-lab/dotnet-conf-2025-nodejs-api-  
dotnet/blob/main/pinvoke-macos/SystemInfo.cs  
*/
```

```
[JSExport]  
public class SystemInfo  
{  
    public string? sysname { get; set; }  
    public string? nodename { get; set; }  
    public string? release { get; set; }  
    public string? version { get; set; }  
    public string? machine { get; set; }  
}
```

```
[JSExport]  
public static class LibcWrapper  
{  
    public static SystemInfo Uname()  
    {  
        Utsname uts;  
        if (uname(out uts) == 0)  
        {  
            return new SystemInfo  
            {  
                sysname = uts.sysname,  
                nodename = uts.nodename,  
                // ...  
            };  
        }  
        else  
        {  
            throw new JSException("uname call failed");  
        }  
    }  
}
```

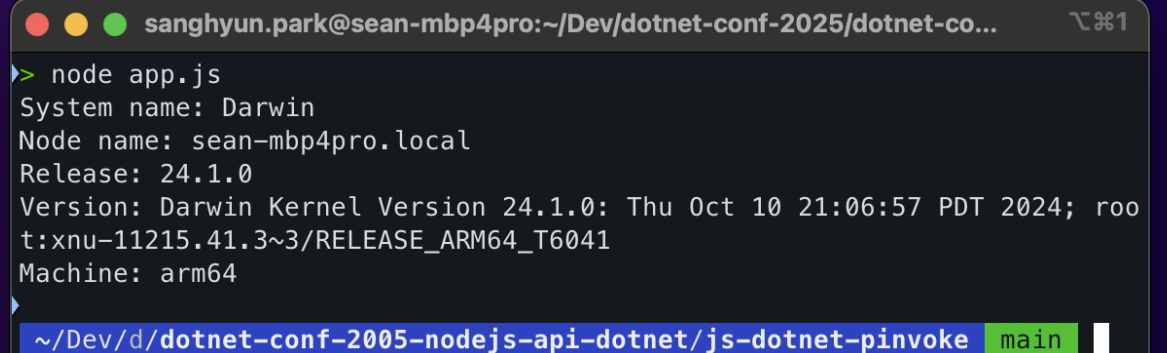
데모3: JavaScript에서 .NET 메소드 호출 #2-2

```
/*
https://github.com/sean-lab/dotnet-conf-2025-nodejs-api-dotnet/blob/main/js-
dotnet-pinvoke/app.js
*/
import dotnet from 'node-api-dotnet';

const PinvokeMacosModule
= dotnet.require(
  '../pinvoke-macos/bin/Debug/net8.0/pinvoke-macos.dll');

const systemInfo = PinvokeMacosModule.LibcWrapper.uname();

console.log(`System name: ${systemInfo.sysname}`);
console.log(`Node name: ${systemInfo.nodename}`);
console.log(`Release: ${systemInfo.release}`);
console.log(`Version: ${systemInfo.version}`);
console.log(`Machine: ${systemInfo.machine}`);
```



A terminal window showing the execution of a Node.js script. The command `node app.js` is entered, and the output displays system information: `System name: Darwin`, `Node name: sean-mbp4pro.local`, `Release: 24.1.0`, `Version: Darwin Kernel Version 24.1.0: Thu Oct 10 21:06:57 PDT 2024; root:xnu-11215.41.3~3/RELEASE_ARM64_T6041`, and `Machine: arm64`. The terminal title bar shows the user `sanghyun.park` at `sean-mbp4pro` in the directory `~/Dev/dotnet-conf-2025-dotnet-co...`. The current file being edited is `~/Dev/d/dotnet-conf-2005-nodejs-api-dotnet/js-dotnet-pinvoke` in the `main` branch.

```
sanghyun.park@sean-mbp4pro:~/Dev/dotnet-conf-2025-dotnet-co...
> node app.js
System name: Darwin
Node name: sean-mbp4pro.local
Release: 24.1.0
Version: Darwin Kernel Version 24.1.0: Thu Oct 10 21:06:57 PDT 2024; root:xnu-11215.41.3~3/RELEASE_ARM64_T6041
Machine: arm64
~/Dev/d/dotnet-conf-2005-nodejs-api-dotnet/js-dotnet-pinvoke main
```


TypeScript를 위한 타입 생성

데모4: TypeScript에서 Intellisense 사용하기

```
/*
https://github.com/sean-lab/dotnet-conf-2025-nodejs-api-dotnet/blob/main/ts-dotnet-pinvoke/app.js
*/
import dotnet from 'node-api-dotnet/net8.0';
import { LibcWrapper, SystemInfo } from
'../../pinvoke-macos/bin/Debug/net8.0/pinvoke-macos';

const PinvokeModule = dotnet.require(
'../../pinvoke-macos/bin/Debug/net8.0/pinvoke-macos.dll') as {
  LibcWrapper: typeof LibcWrapper;
};

const systemInfo: SystemInfo = PinvokeModule.LibcWrapper.uname();

console.log(`System name: ${systemInfo.sysname}`);
console.log(`Node name: ${systemInfo.nodename}`);
console.log(`Release: ${systemInfo.release}`);
console.log(`Version: ${systemInfo.version}`);
console.log(`Machine: ${systemInfo.machine}`);
```

.d.ts 파일 생성

Intellisense 활성화

```
pinvoke-macos
├── bin/Debug/net8.0
│   ├── import.cjs
│   ├── pinvoke-macos.cjs
│   └── pinvoke-macos.d.ts
├── pinvoke-macos.deps.json
├── pinvoke-macos.dll
├── pinvoke-macos.mjs
├── pinvoke-macos.pdb
├── obj
├── pinvoke-macos.csproj
├── pinvoke-macos.sln
├── SystemInfo.cs
├── pinvoke-macos-sample
├── ts-dotnet
└── ts-dotnet-intellisense

export class SystemInfo {
  constructor();

  sysname?: string;

  nodename?: string;

  release?: string;

  version?: string;

  machine?: string;
}

export namespace LibcWrapper {
  export function uname(): SystemInfo;
}
```

```
console.log(`Machine: ${systemInfo.machine}`);
```

- machine?
- nodename?
- release?
- sysname?
- version?

Node.js를 .NET에 임베딩

데모5: C#에서 JavaScript 코드 호출 (1/2)

1. libnode 빌드 (macOS 기준)

```
mkdir libnode
cd libnode
git clone https://github.com/jasongin/nodejs -b napi-libnode-v20.9.0 --single-branch .
./configure --shared; make -j4
```

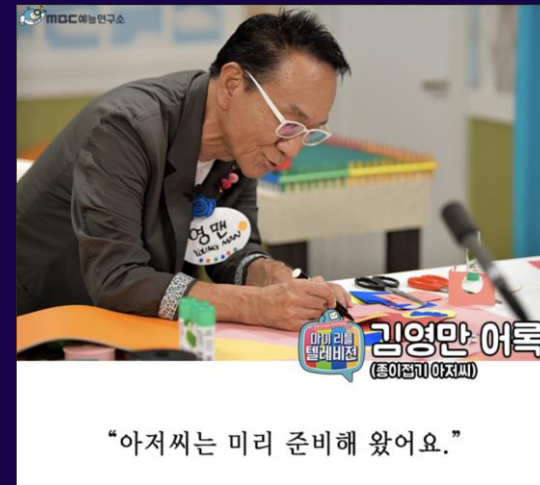
2. libnode.dylib 복사

```
cp out/release/...dylib dotnet-js
```

3. .NET 프로젝트 생성 및 Microsoft.JavaScript.NodeApi 의존성 추가

```
dotnet new console
dotnet add package --prerelease Microsoft.JavaScript.NodeApi
dotnet add package --prerelease Microsoft.JavaScript.NodeApi.Generator
```

Node.js의 코어 기능을 라이브러리 형태로
제공하여, 다른 애플리케이션에서
임베딩할 수 있도록 설계된 동적
라이브러리(Electron.js 등에서 사용 중)




“아저씨는 미리 준비해 왔어요.”

(M4 Pro에서 30분 넘게 소요...)

데모5: C#에서 JavaScript 코드 호출 (2/2)

4. C#에서 libnode.dylib을 로드 하고 JavaScript 코드 실행

```
string baseDir = Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location);  
NodejsPlatform nodejsPlatform = new(Path.Combine(baseDir, "libnode.115.dylib"));  
var nodejs = nodejsPlatform.CreateEnvironment(baseDir);  
  
await nodejs.RunAsync(async () =>  
{  
    var script = "x = 10;";  
    JSValue.RunScript(script);  
  
    var x = (int)JSValue.Global["x"];  
    Console.WriteLine($"Access x in JS from .NET: {x}");  
  
    x = x * x;  
    JSValue.Global.SetProperty("x", x);  
    JSValue.RunScript("console.log(`Access x in JS from JS: ${x}`);");  
  
    await Task.CompletedTask;  
});
```



The terminal window shows the command `dotnet run` being executed. The output displays the results of JavaScript code running within a .NET environment, showing the value of `x` being accessed from .NET and then modified within JavaScript.

```
sanghyun.park@sean-mbp4pro:~/dEv/dotnet-conf-2025/dotnet-conf-200...  
~/dEv/d/dotnet-conf-2005-nodejs-api-dotnet/dotnet-js main ?1 dotnet run  
Access x in JS from .NET: 10  
Access x in JS from JS: 100  
~/dEv/d/dotnet-conf-2005-nodejs-api-dotnet/dotnet-js main ?1
```

AOT로 .NET Addon 만들기

.NET Native AOT로 Native Addon 만들기

- 대상 프레임워크는 .NET 8 이상만 지원
- 프로젝트 파일(.csproj)에 AOT 퍼블리싱 프로퍼티 추가

```
<PublishAot>true</PublishAot>  
<PublishNodeModule>true</PublishNodeModule>  
<PublishDir>bin</PublishDir> >
```

- “dotnet publish” 명령어로 .node 생성
- JavaScript에서 require문을 이용하여 import
- NPM 패키지로도 배포 가능!

데모6: AOT로 Node.js Addon 만들기 (1/2)

1. .NET 프로젝트 생성 및 Microsoft.JavaScript.NodeApi 의존성 추가

```
dotnet new classlib
```

```
dotnet add package --prerelease Microsoft.JavaScript.NodeApi
```

```
dotnet add package --prerelease Microsoft.JavaScript.NodeApi.Generator
```

2. .csproj Property Group 수정

```
<PublishAot>true</PublishAot>
```

```
<PublishNodeModule>true</PublishNodeModule>
```

```
<PublishDir>bin</PublishDir>
```

```
<RuntimeIdentifier>osx-arm64</RuntimeIdentifier> <!-- macOS Silicon -->
```


데모6: AOT로 Node.js Addon 만들기 (2/2)

3. .csproj Property Group 수정

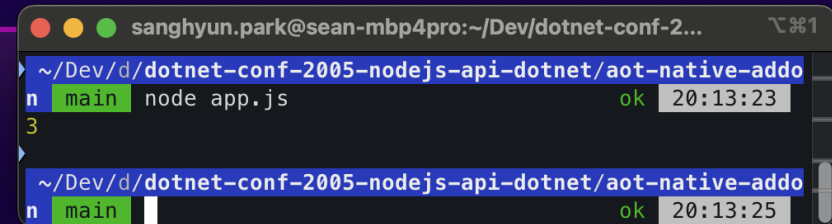
```
using Microsoft.JavaScript.NodeApi;  
  
[JSExport]  
public class Calculator  
{  
    public static int Add(int a, int b)  
    {  
        return a + b;  
    }  
}
```

4. dotnet publish

```
▼ aot-native-addon  
  ▼ bin  
    > aot-native-addon.dylib.dSYM  
    > Release  
    JS aot-native-addon.cjs  
    TS aot-native-addon.d.ts  
    JS aot-native-addon.mjs  
    aot-native-addon.node
```

5. app.js 작성

```
const MyLib = require('./bin/aot-native-addon.node');  
  
console.log(MyLib.Calculator.add(1, 2));
```

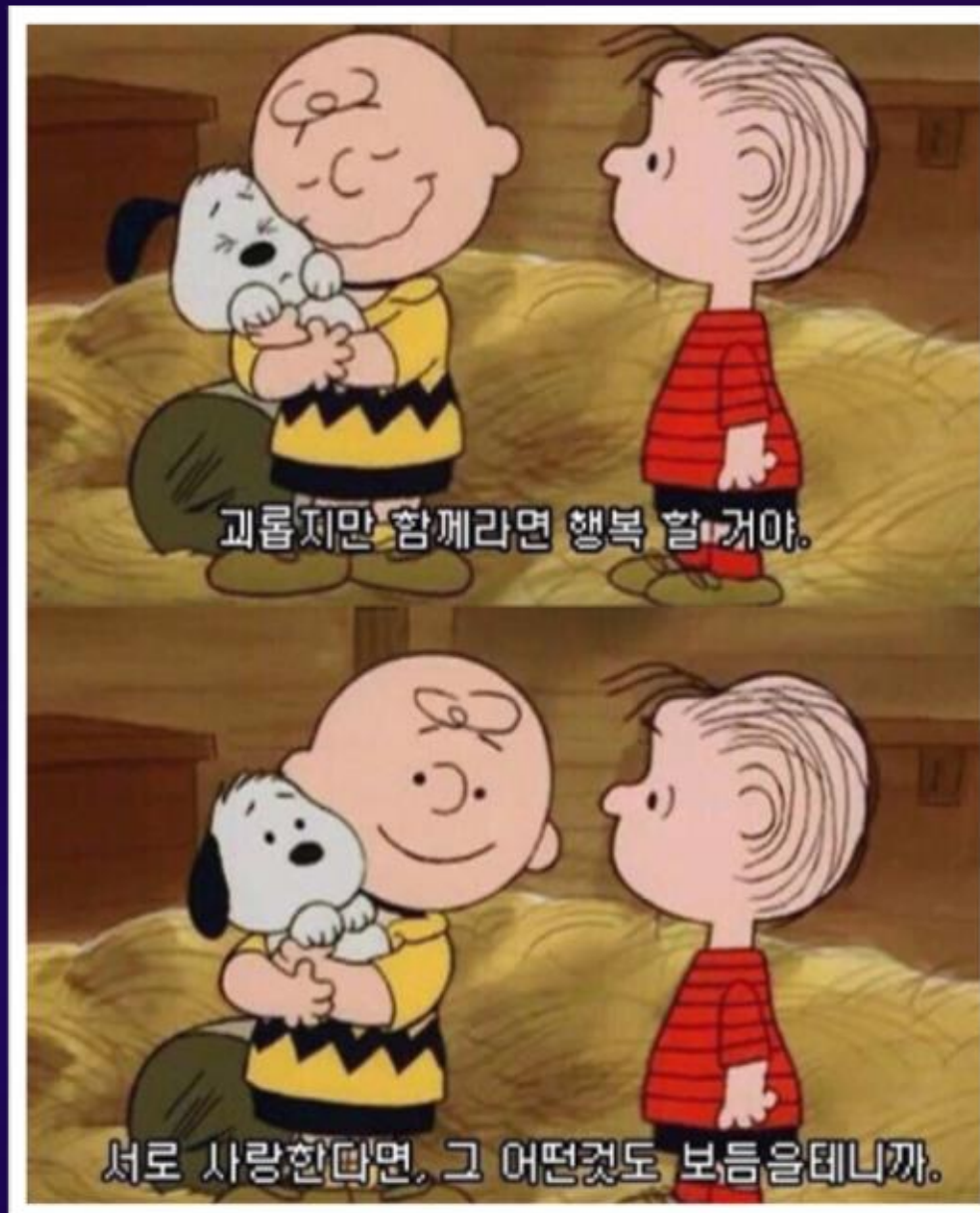


```
sanghyun.park@sean-mbp4pro:~/Dev/dotnet-conf-2...  
~/Dev/d/dotnet-conf-2005-nodejs-api-dotnet/aot-native-addon  
n main node app.js ok 20:13:23  
3  
~/Dev/d/dotnet-conf-2005-nodejs-api-dotnet/aot-native-addon  
n main ok 20:13:25
```

참고자료

- **node-api-dotnet 프로젝트**
 - <https://microsoft.github.io/node-api-dotnet/>
- **예제 코드**
 - <https://github.com/sean-lab/dotnet-conf-2025-nodejs-api-dotnet>
- **JavaScript에서 System API 사용하기**
 - https://docs.google.com/presentation/d/12mer6_AnOO4_5sG72W0hWEyW-p7eQXKgt-Md9xHLCd8/edit?usp=sharing

Q/A



감사합니다

