11. TypeScript, WASM

2023학년 2학기 웹응용프로그래밍

권 동 현

CSLAB

Contents

- Introduction to TypeScript
- Introduction to WASM

Introduction to TypeScript

CSLAB

TypeScript

- JavaScript + Type
  - In JavaScript, types exist, but they are not explicitly specified.
- Through a compiler called Tsc, TypeScript can be transpiled into JavaScript.
  - Since Node can only execute JS, TS code needs to be transpiled into JS before execution.
  - When you install the TypeScript package, the compiler is installed alongside.
  - It operates using the "tsc" command.
  - It runs according to the settings specified in tsconfig.json.
- While there is a TypeScript runtime called Deno, it is not yet widely adopted.

콘솔

```
$ npm init -y
$ npm i typescript
$ npx tsc --init
```

tsconfig.json

- refer to https://www.typescriptlang.org/tsconfig

tsconfig.json

```
{
  "compilerOptions": {
    "target": "es2016",
    "module": "commonjs",
    "esModuleInterop": true,
    "forceConsistentCasingInFileNames": true,
    "strict": true,
    "skipLibCheck": true
*/
  }
}
```

- **target**: 결과물의 문법을 어떤 버전의 자바스크립트 코드로 만들어낼지 정합니다. es2016은 2016년 자바스크립트 버전입니다.
- **module**: 결과물의 모듈 시스템을 어떤 종류로 할지 정합니다. 노드라면 commonjs를 하면 되고, 최신 브라우저에서는 es2022를 하면 됩니다.
- **esModuleInterop**: CommonJS 모듈도 ECMAScript 모듈처럼 인식하게 해줍니다. true로 하면 됩니다.
- **forceConsistentCasingInFileNames**: true이면 모듈을 import할 때 파일명의 대소문자가 정확히 일치해야 합니다. 어차피 리눅스나 맥에서는 파일명 대소문자를 구분하므로 켜두는 것이 좋습니다.
- **strict**: 엄격한 타입 검사를 할지 정합니다. true로 하지 않으면 타입스크립트를 사용하는 의미가 퇴색됩니다.
- **skipLibCheck**: true이면 모든 라이브러리의 타입을 검사하는 대신 내가 직접적으로 사용하는 라이브러리의 타입만 검사해 시간을 아낄 수 있습니다.

Create ts file

- Write compare.js and index.ts.
  - After running npx tsc, index.js should be generated.

compare.js

```
let a  = 'hello';
a = 'world';
```

index.ts

```
let a  = 'hello';
a = 'world';
```

ts 파일은 tsc 명령어를 통해 js로 변환해야 합니다.

콘솔

```
$ npx tsc
```

index.js

```
"use strict";
let a = 'hello';
a = 'world';
```

Error case

- Errors are displayed in the console or editor where the 'tsc' command is entered.

compare.js

```
let a  = 'hello';
a = 123;
```

index.ts

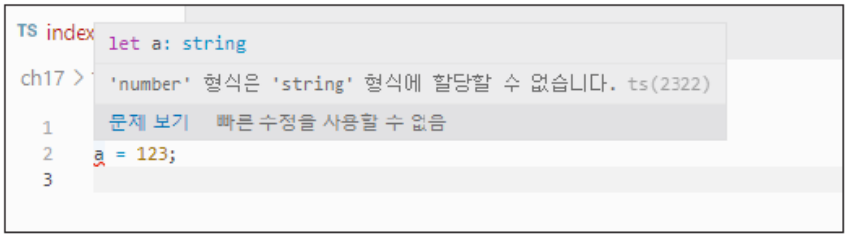
```
let a  = 'hello';
a = 123;
```

콘솔

```
$ npx tsc
index.ts:2:1 - error TS2322: Type 'number' is not assignable to type 'string'.

  ~


Found 1 error in index.ts:2
```

그림 17-3 VS Code에서도 에러가 표시됩니다.

그림 17-4 빨간 줄 위에 마우스 커서를 올리면 에러 메시지를 볼 수 있습니다.



Error case

- tsc separates type checking and code generation functionalities. Even if one fails, the other continues.
  - To perform only type checking, use 'tsc --noEmit'

콘솔

```
$ npx tsc --noEmit
```

Explicitly specifying types (typing)

- Adding types to variables, function parameters, and return values.
  - The part after the colon represents the type

compare.js

```
let a = true;
const b = { hello: 'world' };

function add(x, y) { return x + y };
const minus = (x, y) => x - y;
```

index.ts

```
let a: boolean = true;
const b: { hello: string } = { hello: 'world' };

function add(x: number, y: number): number { return x + y };
const minus = (x: number, y: number): number => x - y;
```

Explicitly specifying types

표 17-1 타입으로 사용 가능한 값

분류	설명	예시
기본형	string, number, boolean, symbol, object, undefined, null, bigint	`const a: string = 'hello';` `const b: object = { hello: 'ts' };`
배열	타입 뒤에 [ ]를 붙임. 길이가 고정된 배열이면 [ ] 안에 타입을 적음	`const a: string[] = ['hello', 'ts', 'js'];` `const b: [number, string] = [123, 'node'];`
상수	1, 'hello', true 등의 고정값	`const a: 1 = 1;`
변수	typeof 변수로 해당 변수의 타입 사용 가능	`let a = 'hello';` `const b: typeof a = 'ts';`
클래스	클래스 이름을 그대로 인스턴스의 타입으로 사용 가능	`class A {}` `const a = new A();`

type 선언	다른 타입들을 조합해서 새로운 타입 생성 가능. 유니언(|), 인터섹션(&) 사용 가능	`const a: { hello: string } = { hello: 'ts' };`
		`type B = { wow: number };`
		`const b: B = { wow: 123 };`
		`type C = string | number;`
		`let c: C = 123;`

인터페이스	interface 선언	`interface A { hello: string, wow: number }` `const a: A = { hello: 'ts', wow: 123 }`

Without specifying types

- Removing types results in TS7006 error.
  - Cases arise where types are automatically inferred, as with 'a' and 'b.'
  - If inference is challenging, the type becomes 'any' (avoid using 'any').

index.ts

```
let a = true;
const b = { hello: 'world' };

function add(x, y) { return x + y };
const minus = (x, y) => x - y;
```

콘솔

```
$ npx tsc --noEmit
index.ts:4:14 - error TS7006: Parameter 'x' implicitly has an 'any' type.
4 function add(x, y) { return x + y };
..
Found 4 errors in the same file, starting at: index.ts:4
```

Introduction to WASM

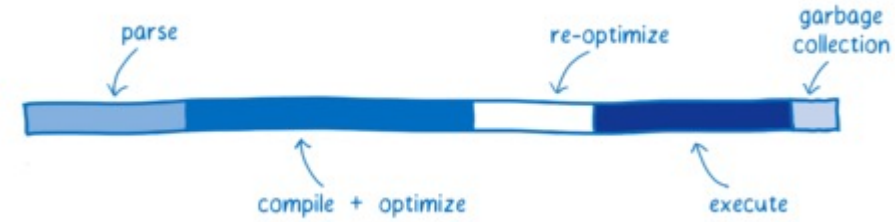
What is WASM?

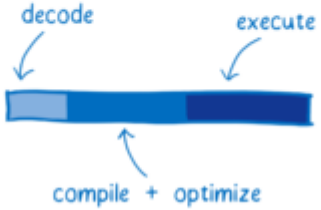
- Web Assembly (WASM)
  - A binary instruction format that serves as a portable compilation target for high-level programming languages like C, C++, and Rust
  - is designed to be a low-level virtual machine that runs in web browsers, enabling the execution of code at near-native speed

- Key features of WASM
  - Performance
    - allows code to be executed at speeds close to native machine code
  - Portability
    - platform-independent, meaning that once compiled to WASM, code can run on any device or platform that supports WebAssembly without modification
  - Security
    - WebAssembly runs in a sandboxed environment, providing a level of security to prevent malicious code from causing harm to the host system
  - Interoperability
    - WebAssembly is designed to work seamlessly with JavaScript, allowing developers to use both languages together in web applications.
  - Broad Industry Supports
    - WebAssembly is supported by major web browsers

What makes WASM fast?

- JS vs WASM
  - Fetching and Decoding(Parsing)
    - WASM is more compact than JS
    - Decoding WASM takes less time than parsing JaS
  - Compiling and Optimizing
    - WASM is closer to machine code than JS
    - WASM already has gone through optimization at compile time
  - Reoptimizing
    - WASM does not need reoptimization
  - Executing
  - Garbage collection

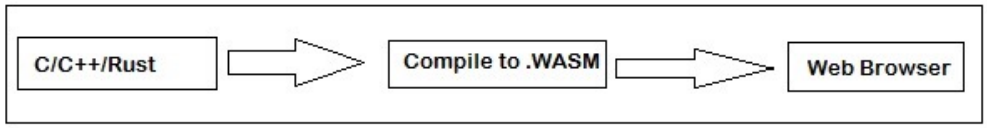




출처: https://hacks.mozilla.org/2017/02/what-makes-webassembly-fast/

How to use WASM

- Source code -> .WASM -> Machine code (in browser)
  - WASM: binary format
  - WAT: human readable format



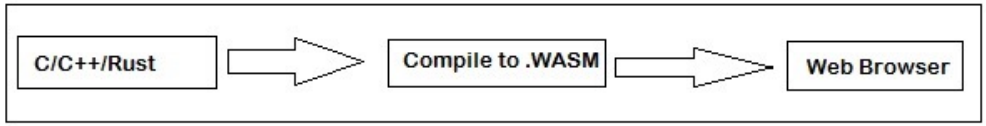

```c
#include <stdio.h>

int main(){
  printf("hello world");
}
```

```
(module
 (type $FUNCSIG$ii (func (param i32) (result i32)))
 (type $FUNCSIG$iii (func (param i32 i32) (result i32)))
 (import "env" "printf" (func $printf (param i32 i32) (result i32)))
 (table 0 anyfunc)
 (memory $0 1)
 (data (i32.const 16) "hello world\00")
 (export "memory" (memory $0))
 (export "main" (func $main))
 (func $main (; 1 ;) (result i32)
  (drop
   (call $printf
    (i32.const 16)
    (i32.const 0)
   )
  )
  (i32.const 0)
 )
)
```

참조: https://mbebenita.github.io/WasmExplorer/

How to use WASM

- Source code -> .WASM -> Machine code (in browser)




```
(module
 (type $FUNCSIG$ii (func (param i32) (result i32)))
 (type $FUNCSIG$iii (func (param i32 i32) (result i32)))
 (import "env" "printf" (func $printf (param i32 i32) (result i32)))
 (table 0 anyfunc)
 (memory $0 1)
 (data (i32.const 16) "hello world\00")
 (export "memory" (memory $0))
 (export "main" (func $main))
 (func $main (; 1 ;) (result i32)
  (drop
   (call $printf
    (i32.const 16)
    (i32.const 0)
   )
  )
  (i32.const 0)
 )
)
```

```
wasm-function[1]:
  sub rsp, 0x18                          ; 0x000000 48 83 ec 18
  cmp qword ptr [r14 + 0x28], rsp        ; 0x000004 49 39 66 28
  jae 0x58                               ; 0x000008 0f 83 4a 00 00 00
 0x00000e:
  mov edi, 0x10                          ; 0x00000e bf 10 00 00 00
  xor esi, esi                           ; 0x000013 33 f6
  mov qword ptr [rsp], r14               ; 0x000015 4c 89 34 24
  mov rax, qword ptr [r14 + 0x30]        ; 0x000019 49 8b 46 30
  mov r14, qword ptr [r14 + 0x38]        ; 0x00001d 4d 8b 76 38
  mov r15, qword ptr [r14 + 0x18]        ; 0x000021 4d 8b 7e 18
  call rax                               ; 0x000025 ff d0
  mov r14, qword ptr [rsp]               ; 0x000027 4c 8b 34 24
  mov r15, qword ptr [r14 + 0x18]        ; 0x00002b 4d 8b 7e 18
  xor eax, eax                           ; 0x00002f 33 c0
  nop                                    ; 0x000031 66 90
  add rsp, 0x18                          ; 0x000033 48 83 c4 18
  ret                                    ; 0x000037 c3

wasm-function[0]:
  sub rsp, 0x18                          ; 0x000000 48 83 ec 18
  mov qword ptr [rsp], r14               ; 0x000004 4c 89 34 24
  mov rax, qword ptr [r14 + 0x30]        ; 0x000008 49 8b 46 30
  mov r14, qword ptr [r14 + 0x38]        ; 0x00000c 4d 8b 76 38
  mov r15, qword ptr [r14 + 0x18]        ; 0x000010 4d 8b 7e 18
  call rax                               ; 0x000014 ff d0
  mov r14, qword ptr [rsp]               ; 0x000016 4c 8b 34 24
  mov r15, qword ptr [r14 + 0x18]        ; 0x00001a 4d 8b 7e 18
  nop                                    ; 0x00001e 66 90
  add rsp, 0x18                          ; 0x000020 48 83 c4 18
  ret                                    ; 0x000024 c3
```

참조: https://mbebenita.github.io/WasmExplorer/