2장 타입스크립트 활용

2.1 컴파일러

타입스크립트는 자바스크립트와 자바와는 다른 다른 방식으로 동작한다.

프로그램을 어떻게 동작할까?

- 프로그램을 프로그래머가 작성한 **텍스트 파일**로 구성된다.
- **텍스트 파일**을 **컴파일러**(Compiler)로 파싱한다.
  - AST(Abstract Syntax Tree) 자료구조로 변환한다.
  - AST를 바이트코드(bytecode)로 변환한다.
- 런타임(runtime)이라는 프로그램에 바이트 코드를 입력하고 평가한다.

타입스크립트 컴파일러(TSC)

바이트 코드 대신 **자바스크립트 코드**로 변환시킨다.

타입스크립트는 언제 안전해지는가?

- 타입스크립트 컴파일러는 AST를 만들어 결과를 만들기 전에 타입 확인을 거친다. → 컴파일 과정에서 타입 검사기(typechecker)가 AST를 확인 할 때 검사한다.

컴파일 과정

TypeScript

1. TS Source → TS AST
2. **타입 검사기(Type Checker)**가 TS AST를 확인
3. TS AST → JS Source

JavaScript

1. JS Source → JS AST
2. JS AST → bytecode
3. 런타임에서 bytecode를 평가

- TSC가 2에서 타입을 검사하고, 컴파일 이후에는 타입을 확인하지 않는다. → 즉, 타입 정보는 최종적으로 만들어지는 프로그램에 영향을 주지 않는다. ⇒ 오직 2단계의 타입 확인 용도로 사용된다.

2.2 타입 시스템

타입시스템이란?

타입 검사기가 프로그램 타입 할당에 사용하는 규칙의 집합

타입 시스템의 두 종류

1. 어떤 타입을 사용하는지 컴파일러에 ***명시적으로 알려주는*** 타입 시스템
2. 자동으로 ***타입을 추론하는*** 타입 시스템

타입스크립트는 두 시스템의 영향을 받았다

- 개발자는 타입을 **명시**, **추론** 방식을 선택 할 수 있다.
  - **타입 명시하기**
    - 어노테이션을 이용하면 타입스크립트로 명시적으로 지정할 수 있다.

```
const a: number = 1;
const b: string = "hi";
const c: boolean[] = [true, false];
```

  - **타입 추론하기**
    - 어노테이션을 사용하지 않으면 타입스크립트가 타입을 추론한다.

```
const a = 1;
const b = "hi";
const c = [true, false];
```

→ 어노테이션을 사용하지 않아도 타입 지정 결과는 달라지지 않는다. ⇒ 타입을 추론은 코드를 줄이고, 개발자의 노동을 줄이므로 보통 생략된다.

2.2.1 타입스크립트 VS 자바스크립트

타입 시스템 기능	자바스크립트	타입스크립트
타입을 어떻게 결정되는가?	동적	정적
자동으로 타입이 변환되는가?	O	X(대부분)
언제 타입을 검사하는가?	런타임	컴파일 타임
에러 검출 시점	런타임(대부분)	컴파일 타임(대부분)

타입을 어떻게 결정되는가? (동적타입 vs 점진적타입)

- **자바스크립트**
  - 동적 타입 바인딩(dynamic type binding)이다. → 프로그램을 실행해야만 데이터 타입을 알 수 있다.
- **타입스크립트**
  - 점진적으로 타입을 확인하는(gradually typed) 언어이다. → 모든 타입을 알고 있으면 최상의 결과가 나오지만, 반드시 알아야하는 것은 아니다. → 자바스크립트 코드를 타입스크립트로 마이그레이션할 때 특히 유용하다.

자동으로 타입이 변환되는가?

- 자바스크립트
  - 약한 타입(weekly typed) 언어이다.
  - 유효하지 않은 연산을 수행하면, 변환시켜 주어진 정보 기반의 최상의 결과를 도출한다.

```
3 + [1]; //31
```

    - 다음과 같은 예시를 확인하자.
      a. 숫자 3과 배열 [1] 연산이 주어진다.
      b. `+` 연산을 사용하였으므로 자바스크립트는 두 값을 연결한다.

c. `3` 을 “3” 문자열로 **암묵적인 변환**을 시행한다.

d. `[1]` 을 “1” 문자열로 **암묵적인 변환**을 시행한다.

e. 변환된 두 문자열을 붙여서 “31” 을 도출한다. ⇒ 즉 아래와 같은 연산이 수행된다.

```
(3).toString() + [1].toString(); //31
```

- 타입스크립트
  - 강한 타입(strongly typed) 언어이다.
  - 변수 및 기타 데이터 구조에 특정 형식을 기반으로 선언할 수 있다.
  - 유효하지 않은 작업 발견 즉시 불평한다.
    - 예시 — Operator '+' cannot be applied to types 'number' and 'number[]'. ⇒자바스크립트의 암묵적 변환으로 문제 원인이 추적하기 어려운 상황을 해결 할 수 있다.

언제 타입을 검사하는가?

- 자바스크립트
  - 타입을 변환하는 노력을하고, 대부분의 상황에서 타입을 따지지 않는다.
- 타입 스크립트
  - 컴파일 타임에 타입 확인기(typechecker)가 코드 타입을 확인한다.
  - 코드를 실행하기 전에 에러를 검출 할 수 있다.

에러는 언제 검출되는가?

- **자바스크립트**
  - **런타임 시점**에 예외를 던지거나 암묵적 형변환을 수행한다. ⇒ 프로그램을 실행해야 문제를 확인할 수 있다.
- **타입스크립트**
  - **컴파일 타임**에 문법 에러와 타입 관련 에러를 검출한다. → 코드 편집기에서 에러를 바로 확인할 수 있다.

2.3 코드 편집기 설정

- TSC는 타입스크립트로 구현된 명령행 도구이다. (TSC는 자체 호스팅 컴파일러 혹은, 자신을 컴파일러 하는 특별한 종류의 컴파일러이다.)
  - 실행시 NodeJS가 필요하다. Node.js

i. 공식 웹사이트를 참고하여 Node.js를 설정한다..

ii. NPM(프로젝트 의존성, 빌드 관리자)을 이용하여 TSC와 TSLint를 설치한다.

```
npm init
npm install --save-dev typescript tslint @types/node
```

— TSC, TELint, NodeJS용 타입 선언 설치

2.3.1 tsconfig.json

- 루트 디렉토리에 tsconfig.json이 존재해야한다.
- 컴파일에 필요한 루트 파일과 컴파일러 옵션을 지정한다.
  - 타입스크립트 내장 명령으로 자동 설정 할 수 있다.

```
./node_modules/.bin/tsc --init
```

  - 예시

```
{
  "compilerOptions": {
    /* Language and Environment */
    "lib": [
      "es2015"
    ] /* Specify a set of bundled library declaration files that describe the target runtime e
    "module": "commonjs" /* Specify what module code is generated. */,
    "outDir": "./dist" /* Specify an output folder for all emitted files. */,
    "sourceMap": true /* Create source map files for emitted JavaScript files. */,
    "strict": true /* Enable all strict type-checking options. */,
    "target": "es2016" /* Set the JavaScript language version for emitted JavaScript and inclu
  },
  "include": ["src"] /* TSC가 타입스크립트 파일을 찾을 디렉터리*/
}
```

→ 옵션을 추가할 수 있다. → 옵션을 바꿀 이유는 자주 발생하지 않는다.

i. 브라우저용 타입스크립트를 위해 lib에 'dom'을 추가

ii. 자바스크립트를 타입스크립트로 마이그레이션하는 과정에서 엄격함을 조절

- 자세한 옵션은 공식문서에서 확인할 수 있다. Documentation - What is a tsconfig.json
- 명령행 옵션으로 제어할 수 있으며 해당 커맨드로 확인 가능하다. —> `./node_modules/.bin/tsc --help`

2.3.2 tslint.json

- 타입스크립트 스타일 규약을 정의한다.
  - 탭, 공백 사용 등을 결정한다.
- 선택사항이지만 일관된 코딩 스타일을 사용하도록 강력히 권장된다.
  - 코드 리뷰 시 동료 개발자와의 소모적인 논쟁을 줄일 수 있다.
- 명령행으로 생성할 수 있다.

```
./node_modules/.bin/tslint --init
```

→ 해당 명령행 입력시 아래 옵션으로 생성된다.

```
{
    "defaultSeverity": "error",
    "extends": [
        "tslint:recommended"
    ],
    "jsRules": {},
    "rules": {},
    "rulesDirectory": []
}
```

해당 옵션은 TSlint 문서에서 확인할 수 있다. TSLint core rules

2.4 index.ts

1. tsconfig.json 과 tslint.json이 설정 후
2. src 디렉터리에 index.ts를 추가한다.

   → 내부에 console.log 등을 입력
3. 타입스크립트를 컴파일하고 실행한다.

```
# TSC로 타입스크립트를 컴파일 한다.
./node_modules/.bin/tsc
# NodeJS로 코드를 실행한다.
node ./dist/index.js
```