알아 두셔야 할 사항

- 책은 타입스크립트 버전 3.7.4를 기준으로 만들어 졌지만, 이 PPT는 버전 4.3.5를 기준으로 작성 되었습니다. 이는 책 코드의 5% 정도가 타입스크립트 버전 4에서 정상적으로 컴파일 되지 않기 때문입니다.
- PPT 내용 중 비주얼 스튜디오 코드 화면을 캡쳐 한 것은 버전 4들 대상으로 코드가 변한 것을 반영한 것이고, PDF를 캡처한 것은 책의 코드가 버전 4에서도 정상적으로 컴파일 된다는 것을 반영한 것입니다.

- 타입스크립트는 tsconfig.json 파일 내용에 매우 민감합니다. 그리고 tsc --init으로 생성된 tsconfig.json 파일은 가장 엄격하게 코드를 컴파일 합니다. 이는 버그가 있는 자바스크립트 코드들을 컴파일 시점에서 잡아주게 하기 위해서 입니다.
- 이에 따라 가능한 이 책의 02-2절에 기술된 타입스크립트 초보자용 tsconfig.json 파일을 사용하시기 바랍니다.

타입스크립트 개발 시 알아 둬야 할 부분

- 타입 에러가 나는 소스 파일 맨 위에 // @ts-nocheck 주석을 붙임
- 특정 줄에서 나는 경우, 해당 줄 앞에 // @ts-ignore 주석을 붙임

- [참고] //@ts-nocheck 주석은 컴파일러의 타입 체크 기능을 무력화 시키므로 가능한 사용을 자제해야 합니다. 하지만 알 수 없는 타입 오류로 인하여 더이상 진도를 나갈 수 가 없을 때의 가장 효과적인 임시 방편 대책입니다.

- [참고] 타입스크립트 컴파일러는 '@types/패키지이름' 형태의 타입 라이브러리가 반드시 필요합니다. 그리고 어떤 패키지들의(이 책의 경우 09장의 ramda 패키지) 타입 라이브러리들은 잘못 구현된 부분이 있거나, 도저히 타입스크립트로 옮길 수 없는 기능을 담고 있기도 합니다.
- 사실 이 경우 유일한 해결책이 // @ts-ignore 주석입니다.

타입스크립트 개발 시 알아 둬야 할 부분

타입 오류가 날 때의 대책

- **// @ts-nocheck 주석(6번 줄)으로 파일 전체의 타입 체크를 안 하게 하여 컴파일 오류를 방지한 예**

ch03-3 > TS Person1.ts > ...

```
1   /*
2   이 코드는 ts2564 오류가 납니다. 그리고 그 이유는 코드를 이런식으로 작성하지 말라는 의미입니다.
3   하지만, 이 코드는 교육용 이므로, 여기서는 @ts-nocheck 을 사용하여
4   컴파일러로 하여금 너무 엄격하게 코드를 해석하지 않도록 했습니다.
5   */
6   // @ts-nocheck
7
8   class Person1 {
9     name: string
10    age?: number
11  }
12  let jack1: Person1 = new Person1()
13  jack1.name = 'Jack'
14  jack1.age = 32
15  console.log(jack1)
```

타입스크립트 개발 시 알아 둬야 할 부분

타입 오류가 날 때의 대책

- // @ts-ignore 주석(19번 줄)으로 20번 줄에서 발생하는 컴파일 오류를 방지한 예

```
10  const indexedMap = R.addIndex(R.map)
11
12  return R.pipe(
13    R.trim,
14    R.split(delim),
15    R.map(R.toLower),
16    indexedMap(
17      (value: string, index: number) => (index > 0 ? makeFirstToCapital(value) : value)
18    ),
19    // @ts-ignore
20    R.join('') // 배열을 다시 문자열로 변환합니다.
21  ) as StringToStringFunc
```

02

타입스크립트 프로젝트 생성과 관리

02-1 타입스크립트 프로젝트 만들기

이 절의 목적

- 타입스크립트를 개발 언어로 하는 Node.js 프로젝트를 생성하는 법 알기
- package.json 파일의 용도를 익히기
- 타인이 생성한 타입스크립트 프로젝트의 사용법 익히기
- chance, ramda 와 같은 외부 오픈 소스 패키지들의 설치법 익히기
- tsconfig.json 파일의 용도 익히기
- 개발 모드 와 배포 모드 익히기

02-1 타입스크립트 프로젝트 만들기

타입스크립트 프로젝트란?

- Node.js 프로젝트에 개발 언어만 타입 스크립트를 사용하는 프로젝트

Node.js 프로젝트란?

- 프로젝트 디렉터리에 package.json 파일이 있는 프로젝트
- package.json 파일은 'npm init --y' 명령으로 생성
- 타인이 만든 프로젝트인 경우 'npm install' 명령으로 package.json 파일에 기술된 패키지들을 설치 필요

```
npm init --y
Wrote to C:\work\typescript\ch02-1\package.json:


{
  "name": "ch02-1",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

02-1 타입스크립트 프로젝트 만들기

프로젝트 생성자 관점에서 패키지 설치하기

- 'npm install' 혹은 'npm i' 명령에 옵션 주기

패키지 설치 명령 옵션

npm i 옵션	의미	단축 명령
--save	프로젝트를 실행할 때 필요한 패키지로 설치합니다. 패키지 정보가 package.json 파일의 'dependencies' 항목에 등록됩니다	-S
--save-dev	프로젝트를 개발할 때만 필요한 패키지로 설치합니다. 패키지 정보가 package.json 파일의 'devDependencies' 항목에 등록됩니다	-D

02-1 타입스크립트 프로젝트 만들기

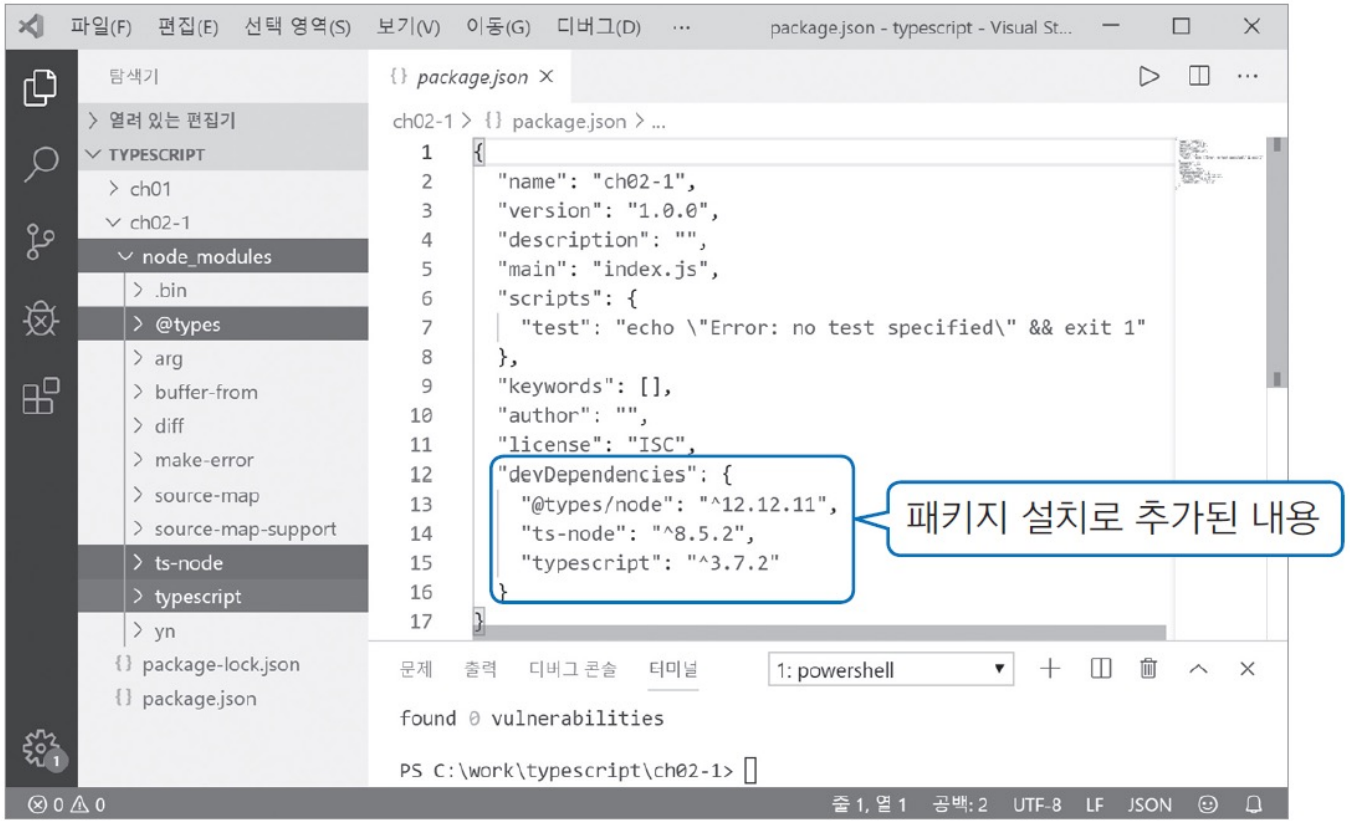
타입스크립트 관련 패키지 설치하기

- 컴파일러와 ts-node 설치하기

```
> npm i -D typescript ts-node
```

- 타입라이브러리 설치하기

```
> npm i -D @types/node
```



02-1 타입스크립트 프로젝트 만들기

tsconfig.json 파일 만들기

- tsconfig.json - 타입스크립트 컴파일러 tsc 의 설정 파일
-
```
> tsc --init
message TS6071: Successfully created a tsconfig.json file.
```

- 이 책에서 사용하는 tsconfig.json 파일 내용

ch02-1 > tsconfig.json > ...

```
{
  "compilerOptions": {
    "module": "commonjs",
    "esModuleInterop": true,
    "target": "es2015",
    "moduleResolution": "node",
    "outDir": "dist",
    "baseUrl": ".",
    "sourceMap": true,
    "downlevelIteration": true,
    "noImplicitAny": false,
    "paths": { "*": ["node_modules/*"] }
  },
  "include": ["src/**/*"]
}
```

- 참고: 책과 달리 "target" 키의 값을 "es5"가 아니라, "es2015"로 설정해야 최신 버전 tsc의 컴파일 오류를 방지할 수 있음

02-1 타입스크립트 프로젝트 만들기

src 디렉터리와 소스 파일 만들기

- tsconfig.json 14행에 include 항목 ["src/**/ *"] 값의 의미- package.json 파일이 있는 '.' 디렉터리 기준으로 ./src 와 ./src/utils 처럼 src의 모든 서브 디렉터리들에 타입스크립트 소스 파일이 있다는 뜻

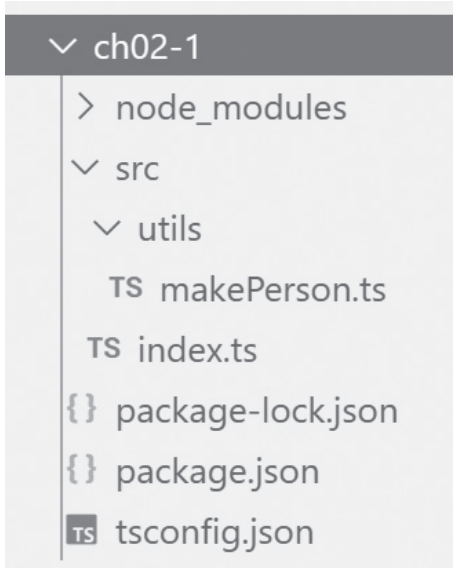
- tsconfig.json 설정대로 프로젝트를 구성하고자 다음 명령으로 src/utils 디렉터리를 생성합니다.

- 다음 'mkdir -p'명령은 src 디렉터리와 src/utils 디렉터리를 동시에 만드는 명령

```
> mkdir -p src/utils
```

- 다음 'touch' 명령은 src/index.ts 파일과 src/utils/makePerson.ts 파일을 동시에 생성하는 명령

```
> touch src/index.ts src/utils/makePerson.ts
```



02-1 타입스크립트 프로젝트 만들기

비주얼 스튜디오 코드(vscode) 편집기에서 코드 작성해 보기

• ch02-1/src/utils/makePerson.ts

```
01: export function makePerson(name: string, age: number) {
02:   return {name: name, age: age}
03: }
04: export function testMakePerson() {
05:   console.log(
06:     makePerson('Jane', 22),
07:     makePerson('Jack', 33)
08:   )
09: }
```

• ch02-1/src/index.ts

```
01: import {testMakePerson} from './utils/makePerson'
02: testMakePerson()
```

시작 소스 파일명을 index로 짓는 이유

node나 ts-node로 소스 파일을 실행하려면 `ts-node ./src/index.ts` 명령을 사용합니다. 하지만 소스 파일명이 index이면 파일명을 생략하고 단순히 `ts-node ./src`로 실행할 수 있습니다. 이 때문에 프로젝트의 시작 함수(엔트리 함수라고 합니다)가 있는 소스 파일명은 보통 index로 짓습니다.

02-1 타입스크립트 프로젝트 만들기

개발 모드와 배포 모드 이해하기

- 개발 모드(development mode) - 제품을 개발 할 때의 모드. 코드가 정상적으로 작성되었는지를 빨리 확인하는 것이 가장 중요한 모드. ts-node 사용.
- 배포 모드(production mode) - 개발 된 제품을 실제 서비스할 때의 모드. 타입스크립트 코드를 자바스크립트 코드로 바꿔(빌드) node.js 로 동작시키는 모드
- 개발 할 때는 ts-node가 편하지만, 배포할 때는 node로 동작시키는 것이 실행 속도가 빠름에 유의

- 옆 package.json 파일의 scripts 항목의 "dev"는 개발 모드일 때 사용하는 명령. 반면에 "build"는 배포 모드에서 사용하는 명령

ch02-1 > {} package.json > ...

```
1  {
2    "name": "02-1",
3    "version": "1.0.0",
4    "description": "",
5    "main": "src/index.js",
     ▷ 디버그
6    "scripts": {
7      "dev": "ts-node src",
8      "build": "tsc && node dist"
9    },
10   "keywords": [],
11   "author": "",
12   "license": "ISC",
13   "devDependencies": {
14     "@types/node": "^16.0.0",
15     "ts-node": "^10.0.0",
16     "typescript": "^4.3.5"
17   }
18 }
```

02-1 타입스크립트 프로젝트 만들기

개발 모드

- **scripts 항목의 dev 명령은 다음처럼 사용**

```
> npm run dev
> ch02-1@1.0.0 dev C:\work\typescript\ch02-1
> ts-node src
{ name: 'Jane', age: 22 } { name: 'Jack', age: 33 }
```

dev 명령에 정의된 명령

코드 실행 결과

배포 모드

- **scripts 항목의 build 명령은 다음처럼 사용**

```
> npm run build
> ch02-1@1.0.0 build C:\work\typescript\ch02-1
> tsc && node dist
{ name: 'Jane', age: 22 } { name: 'Jack', age: 33 }
```

build 명령에 정의된 명령

코드 실행 결과

02

타입스크립트 프로젝트 생성과 관리

이지스퍼블리싱(주)

02-2 모듈 이해하기

이 절의 목적

- 타입스크립트의 모듈 개념 익히기
- 타입스크립트 언어의 import 와 export 키워드 익히기

02-2 모듈 이해하기

모듈이란?

- .ts 파일 확장자
  - 타입스크립트 코드는 반드시 파일 확장자가 .ts 여여 함(예: index.ts)
- 모듈(module)
  - 타입스크립트에서는 index.ts처럼 .ts 확장자를 가진 파일을 모듈이라고 함
  - 다만 .ts 파일이 모듈이기 위해서는 코드에 반드시 import 혹은 export 구문이 있어야 함
- import 와 export 키워드
  - export 키워드 - 모듈의 기능을 제공할 때 사용
  - import 키워드 - 다른 모듈의 기능을 사용하려 할 때 사용

02-2 모듈 이해하기

export 키워드 사용 예

- 다음 Person.ts 파일은 export 키워드를 사용하여 IPerson 과 makePerson이라는 기능을 제공하고 있으므로 모듈

export 키워드 추가 • ch02-2/src/person/Person.ts

```
01: let MAX_AGE = 100
02:
03: export interface IPerson {
04:   name: string
05:   age: number
06: }
... ...생략...
16: export const makePerson = (name: string,
17:     age: number = makeRandomNumber()): IPerson => ({name, age})
```

02-2 모듈 이해하기

import 키워드 사용 예

- **import 구문 -** `import { 심벌 목록 } from '파일의 상대 경로'`

- **import 구문 사용 예 - Person.ts 모듈이 export 한 IPerson과 makePerson을 import 하는 예**

import 문 추가 • ch02-2/src/index.ts

```
01: import {IPerson, makePerson} from './person/Person'
02:
03: const testMakePerson = (): void => {
04:   let jane: IPerson = makePerson('Jane')
05:   let jack: IPerson = makePerson('Jack')
06:   console.log(jane, jack)
07: }
08:
09: testMakePerson()
```

02-2 모듈 이해하기

import * as 구문

- **다른 모듈이 export 한 기능을 한꺼번에 import 하고 싶을 때 사용**

```
import * as 심벌 from '파일 상대 경로'
```

makeRandomNumber.ts 파일이 export 한 모든 기능을 심볼 U로 import

U.makeRandomNumber처럼 심볼.기능 형태로 사용

import * as 구문 추가 • ch02-2/src/person/Person.ts

```
01: import * as U from '../utils/makeRandomNumber'
02:
03: export interface IPerson {
04:   name: string
05:   age: number
06: }
07:
08: class Person implements IPerson {
09:   constructor(public name: string, public age: number) {}
10: }
11:
12: export const makePerson = (name: string,
13:     age: number = U.makeRandomNumber()): IPerson => ({name, age});
```

02-2 모듈 이해하기

export default 키워드

- 자바스크립트와의 호환을 위해 만들어진 구문
- 한 모듈이 내보내는 기능 중 오직 한 개에만 붙일 수 있음
- import 문으로 불러올 때 중괄호 {} 없이 사용할 수 있게 함
- export default와 export 혼합 사용 가능
- export default는 파일에서 단 한번만 사용할 수 있음

export default

export

• ch02-2/src/person/Person.ts

```
01: import {makeRandomNumber} from '../utils/makeRandomNumber'
02: import IPerson from './IPerson'
03:
04: export default class Person implements IPerson {
05:   constructor(public name: string, public age: number = makeRandomNumber()) {}
06: }
07:
08: export const makePerson = (name: string,
09:     age:number = makeRandomNumber()): IPerson => ({name, age})
```

02-2 모듈 이해하기

외부 패키지 설치하기

- 다음은 chance 와 ramda란 이름의 두 개의 외부 패키지를 설치하는 명령

```
> npm i -S chance ramda
> npm i -D @types/chance @types/ramda
```

- [참고] -S 옵션은 생략 가능

• ch02-2/package.json

```
01: {
...   ...생략...
12:   "devDependencies": {
13:     "@types/chance": "^1.0.7",
14:     "@types/node": "^12.12.5",
15:     "@types/ramda": "^0.26.36",
16:     "ts-node": "^8.4.1"",
17:     "typescript": "^3.7.4"
18:   },
19:   "dependencies": {
20:     "chance": "^1.1.3",
21:     "ramda": "^0.26.1"
22:   }
23: }
24:
```

-D 옵션으로 설치한 패키지들

-S 옵션으로 설치한 패키지들

02-2 모듈 이해하기

외부 패키지를 사용할 때 import 문

- **import 문의 from에 '파일 상대 경로'가 아니라 패키지 이름을 사용**

chance 패키지 제공 기능을 Chance라는 이름으로 import 한 예

• ch02-2/src/index.ts

```
01: import IPerson from './person/IPerson'
02: import Person from './person/Person'
03: import Chance from 'chance'
04: import * as R from 'ramda'
05:
06: const chance = new Chance()
07: let persons: IPerson[] = R.range(0, 2)
08:     .map((n: number) => new Person(chance.name(), chance.age()))
09: console.log(persons)
```

ramda 패키지 제공 기능을 R이라는 이름으로 import 한 예

02

타입스크립트 프로젝트 생성과 관리

02-1 타입스크립트 프로젝트 만들기

02-2 모듈 이해하기

02-3 tsconfig.json 파일 살펴보기

02-3 tsconfig.json 파일 살펴보기

이 절의 목적

- tsconfig.json 파일의 다양한 항목들의 각각의 의미에 대해 이해한다

02-3 tsconfig.json 파일 살펴보기

- tsconfig.json 파일의 역할
  - 개발 모드 때 사용하는 ts-node는 내부적으로 타입스크립트 컴파일러 tsc를 호출함
  - tsc는 아무런 옵션 없이 호출하면 기본 모드(default mode)로 동작
  - 만일 tsc --init 명령으로 tsconfig.json 파일을 생성하면, 이 파일은 가장 엄격하게 설정된 옵션을 가지게 됨
  - 이는 소스 코드에 타입 체크를 매우 엄격하게 적용하여, 혹시 모를 오류를 만드는 코드들을 컴파일 타입에서 걸러 주기 위함 임
  - react.js 등 프런트엔드 프레임워크들은 이 보다는 훨씬 완화된 설정을 한 tsconfig.json 파일을 사용함
  - 타입스크립트 초보자들은 절대 tsc --init 명령으로 생성된 tsconfig.json 파일을 그대로 사용하면 안됨

02-3 tsconfig.json 파일 살펴보기

- tsc --help 명령
  - 타입스크립트 컴파일러 tsc 가 구체적으로 어떤 옵션을 줄 수 있는지 알고 싶을 때 사용

```
> tsc --help
Version 3.7.4
Syntax:   tsc [options] [file...]
Examples: tsc hello.ts
          tsc --outFile file.js file.ts
          tsc @args.txt
          tsc --build tsconfig.json
Options:
 -h, --help                                 Print this message.
 -w, --watch                                Watch input files.
...생략...
```

tsc 명령 형식

02-3 tsconfig.json 파일 살펴보기

- tsconfig.json 파일 구조
  - tsconfig.json 항목은 크게 compilerOptions 와 include 항목으로 구성
  - compilerOptions 항목 - 타입스크립트 소스 코드를 컴파일 할 때 적용할 옵션들을 설정 하는 목적
  - include 항목 - 컴파일 해야 할 소스 파일들이 있는 디렉터리 위치 설정
-

• tsconfig.json

```
01: {
02:   "compilerOptions": {
03:     "module": "commonjs",
04:     "esModuleInterop": true,
05:     "target": "es5",
06:     "moduleResolution": "node",
07:     "outDir": "dist",
08:     "baseUrl": ".",
09:     "sourceMap": true,
10:     "downlevelIteration": true,
11:     "noImplicitAny": false,
12:     "paths": { "*": ["node_modules/*"] }
13:   },
14:   "include": ["src/**/*"]
15: }
```

02-3 tsconfig.json 파일 살펴보기

- module 키
  - 타입스크립트 소스코드가 컴파일되어 만들어진 ES5 자바스크립트 코드는 웹 브라우저와 노드제이에스 양쪽에서 모두 동작해야 함
  - 그런데 웹 브라우저와 노드제이에스는 물리적으로 동작하는 방식이 달라서 여러 개의 파일(즉 모듈)로 분할된 자바스크립트 코드 또한 웹 브라우저와 노드제이에스 양쪽에서 각각 다르게 동작
  - 자바스크립트 모듈은 웹 브라우저에서는 AMD(asynchronous module definition) 방식으로 동작하고, 웹 브라우저가 아닌 환경에서는 CommonJS 방식으로 동작

- module 키에 설정할 수 있는 값
  - 웹 브라우저에서 동작: `amd`
  - 노드제이에스에서 동작: `commonjs`

02-3 tsconfig.json 파일 살펴보기

- moduleResolution 키
  - module 키 값이 commonjs이면 Node.js에서 동작하는 것을 의미하므로, moduleResolution 키 값은 항상 node
  - module 키 값이 amd이면 moduleResolution 키 값은 classic으로 설정

02-3 tsconfig.json 파일 살펴보기

- target 키
  - 트랜스파일 할 대상 자바스크립트 버전을 설정하는 용도의 키
  - 보통 es5를 키 값으로 설정
  - 만약 06장의 생성기(generator)기능을 사용하는 경우 es2015를 키 값으로 설정해야 함

02-3 tsconfig.json 파일 살펴보기

- baseUrl과 outDir 키
  - baseUrl과 outDir 키에는 트랜스파일된 ES5 자바스크립트 파일을 저장하는 디렉터리를 설정
  - tsc는 tsconfig.json 파일이 있는 디렉터리에서 실행되므로 현재 디렉터리(current directory)를 의미하는 "."로 baseUrl 키 값을 설정하는 것이 보통
  - OutDir 키는 baseUrl 설정 값을 기준으로 했을 때 하위 디렉터리의 이름을 설정
  - OutDir 키에 dist라는 값을 설정하면 빌드 된 결과가 dist 디렉터리에 만들어짐

02-3 tsconfig.json 파일 살펴보기

- paths 키
  - 소스 파일의 import 문에서 from 부분을 해석할 때 찾아야 하는 디렉터리를 설정하는 용도의 키
  - import 문이 찾아야 하는 소스가 외부 패키지면 node_modules 디렉터리에서 찾아야 하므로 키 값에 node_modules/*도 포함해야 함

02-3 tsconfig.json 파일 살펴보기

- esModuleInterop 키
  - 오픈소스 자바스크립트 패키지들 중에는 웹 브라우저에서만 동작한다는 가정으로 만들어진 것들이 존재
  - 이들은 CommonJS 방식으로 동작하는 타입스크립트 코드에 혼란을 줄 수 있음
  - 02-2절에서 사용해 본 chance가 바로 AMD 방식을 전제로 해서 구현된 라이브러리
  - 따라서 chance 패키지가 동작하려면 esModuleInterop 키 값을 반드시 true로 설정해야 함

02-3 tsconfig.json 파일 살펴보기

- sourceMap 키
  - sourceMap 키 값이 true이면 트랜스파일 디렉터리에는 .js 파일 이외에도 .js.map 파일이 생성됨
  - .js.map 파일을 소스맵(source map) 파일이라고 함
  - 소스맵 파일은 변환된 자바스크립트 코드가 타입스크립트 코드의 어디에 해당하는지를 알려주는 용도
  - 소스맵 파일은 주로 디버깅할 때 사용

02-3 tsconfig.json 파일 살펴보기

- downlevelIteration 키
  - 06장의 생성기(generator)라는 타입스크립트 구문이 정상적으로 동작하려면 이 키 값을 반드시 true로 설정해야 함
  - 또한 target 키 값은 es2015로 설정해야 함

02-3 tsconfig.json 파일 살펴보기

- noImplicitAny 키
  - 타입스크립트 컴파일러는 기본적으로 f(a, b)처럼 매개변수 a, b에 타입을 명시하지 않은 코드일 경우 f(a: any, b: any)처럼 암시적으로 any 타입을 설정한 것으로 간주
  - 또한 이런 형태의 코드는 타입스크립트 언어의 장점을 사용하는 것이 아니므로 다음처럼 코드에 문제가 있음을 알려줌

02-3 tsconfig.json 파일 살펴보기

- noImplicitAny 키 계속
  - 그런데 이 오류는 타입스크립트를 처음 배우는 사람을 매우 혼란스럽게 함
  - 이 책은 타입스크립트 초보자들에게 혼란을 줄이고자 이 키 값을 false로 설정하여 사용
  - 이렇게 하면 타입을 지정하지 않더라도 문제로 인식하지 않기 때문
  - 이 키 값을 true로 하여 사용하려면 이 책의 10장 까지의 이해가 필요함

감사합니다

저는 타입스크립트를 좋아합니다. 최고죠.

라이언 달(Ryan Dahl, Node.js 과 Deno 창시자)