TypeScript

TypeScript 주요 특징

1. 개발 환경이 풍부하다.

마이크로 소프트에서 개발한 altJS로, 표준 Windows 통합 개발 환경 Visual Studio (https://code.visualstudio.com/)가 TypeScript를 표준 채택 있으며, 구문 강조, 인텔리 센스, 디버그 기능을 갖춘 환경이 준비되어 있습니다.

뿐만 아니라, Atom, SublimeText, Visual Studio Code (VSCode) 등 주요 코드 편집기가 TypeScript에 대응하여, 비 Windows 환경에서도 TypeScript 도입이 쉬워졌습니다.

TypeScript 주요 특징

2. JavaScript의 슈퍼 세트이다.

TypeScript를 한마디로 말하면 정적 타입 (Type) 시스템과 클래스 기반 객체 지향 구문을 갖춘 JavaScript의 상위 집합입니다.

JavaScript 구문을 확장한 것으로, 본래의 JavaScript 코드가 거의 그대로 작동하고, TypeScript 구문으로 다시 쓰는 것도 쉽습니다.

JavaScript를 어느 정도 다뤄 온 사람이라면, C#이나 Java 모두 유사한 구문을 가지고 있으므로 이러한 언어를 이용한 적이 있는 사람도 낯설지 않을 것입니다.

TypeScript 주요 특징

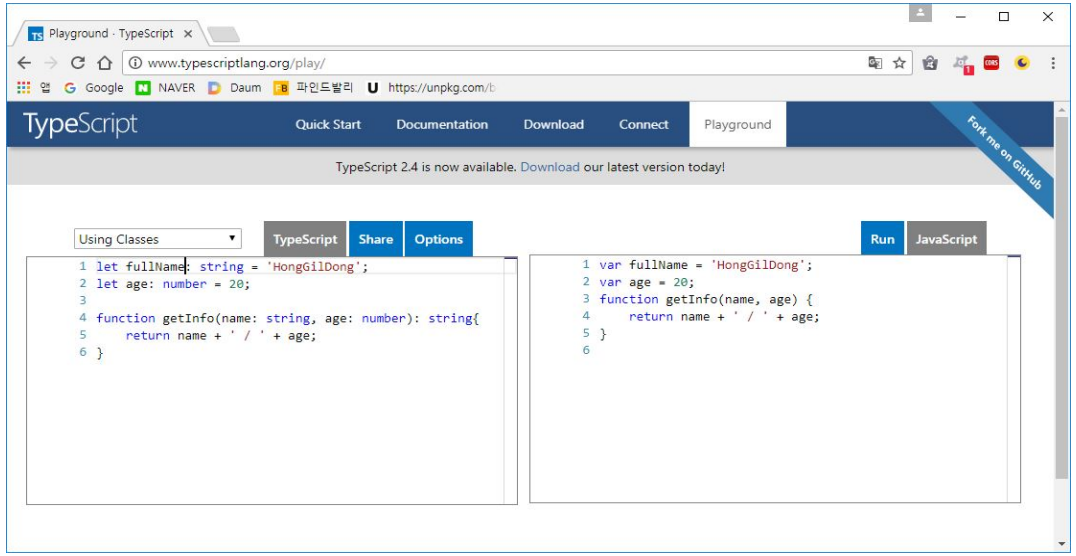
3. ECMAScript 2015의 사양에 대응.

ECMAScript 2015 (통칭, ECMAScript 6)는 2015년 6월 17일에 채택된 JavaScript의 최신 사양입니다. 집필 시점에서는 브라우저 지원 상황은 완전하다고는 말할 수 없지만, TypeScript는 빠르게 ECMAScript 2015의 사양이 채택되어 사용할 수 있습니다 (클래스 기반 객체 지향 구문도 ECMAScript 2015에서 채택 된 새로운 문법입니다).

TypeScript Playground

TypeScript Playground

TypeScript 경험 해 보고 싶거나, 최신 기능을 배우고 싶은 사람은 브라우저에서 실행되는 인터프리터 "TypeScript Playground"(http://www.typescriptlang.org/play/)가 좋습니다.

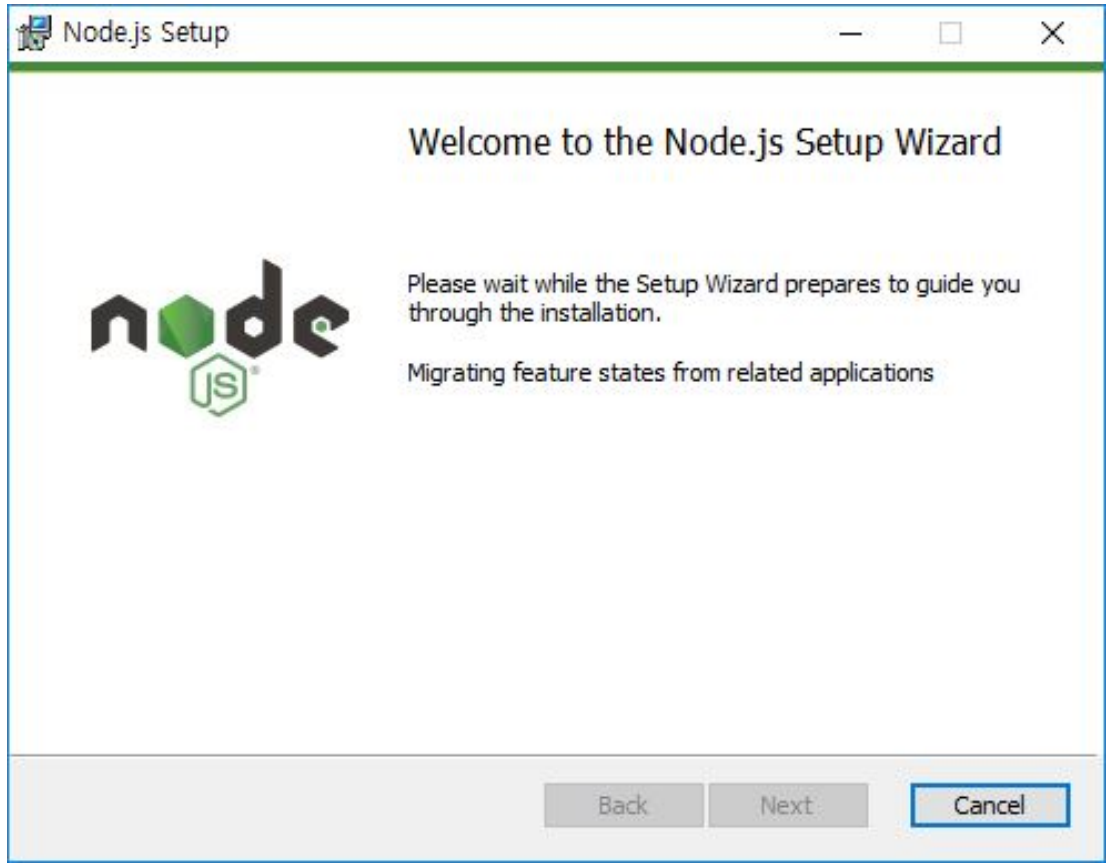


명령 줄에서 실행

tsc 명령을 사용하여 명령 줄에서 TypeScript 컴파일을 수행 할 수 있습니다. Playground은 어디 까지나 테스트에 대한 환경이므로, 실제 개발을 진행 시에는 최소한 명령 줄에서 컴파일 할 수 있는 환경을 구현 할 필요가 있습니다.

tsc 명령은 Node.js(https://nodejs.org/)에서 작동합니다.

Node.js 설치


Node.js Setup
Welcome to the Node.js Setup Wizard
Please wait while the Setup Wizard prepares to guide you through the installation.
Migrating feature states from related applications
Back
Next
Cancel

TypeScript 설치

설치

TypeScript를 npm(Node Package Manager)로 설치할 수 있습니다. 명령 프롬프트에서 다음 명령을 실행하십시오. -g 옵션은 라이브러리를 글로벌로 설치하라는 의미입니다.

$ npm install –g typescript

TypeScript 컴파일 실행

설치하여 tsc 명령을 실행 할 수 있습니다. 예를 들어 hello.ts를 컴파일하려면 다음과 같이합니다.

$ tsc hello.ts

컴파일 된 .js 파일은 hello.js가 됩니다.

여러 .ts 파일을 컴파일 및 결합한다면 - outFile 옵션을 지정합니다. 다음은 hello.ts / world.ts를 정리해 컴파일하고 결과를 helloWorld.js로 출력합니다.

$ tsc --outFile helloWorld.js hello.ts world.ts

TypeScript 컴파일 옵션

-w / --watch

파일의 변경을 감시, 변경 사항이 있으면 자동으로 컴파일 실행

--outDir 폴더명

JS 파일을 출력할 폴더 명을 지정.

--outFile 파일명

출력될 JS 파일명을 지정. (컴파일이 끝난 파일을 1개로 연결

TypeScript 컴파일 옵션

-p 폴더명

컴파일 대상의 프로젝트(tsconfig.json을 포함한 폴더)

--target version

Target이 되는 ECMAScript의 버전

(ES3 | ES5 | ES6 | ES2015 | ES2016 | ESNext 등). 기본 값은 ES3.

--help

Help를 표시.

변수 - let / var 명령

TypeScript에서는 변수를 이용하기 전에 var 또는 let 명령으로 선언 해야합니다. (JavaScript와 같이 선언되지 않은 변수에 대해 값을 할당 할 수 없습니다).

[구문] 변수 선언

let name: type = initial;

name: 변수의 이름

type: 데이터 형

initial: 초기 값

변수 Type

변수를 선언 할 때 지정할 수 있는 형식에는 다음과 같은 것이 있습니다.

number: 수치 형 (정수, 부동 소수점)

string: 문자열 형

boolean: 진위 형 (true, false)

symbol: symbol 형

any: 임의의 형

object type: 뒤에서 설명

변수 Type

TypeScript에서는 변수 선언으로 형태가 생략 된 경우에도 초기 값에서 형식을 유추.

```
let firstName = 'name';

firstName = 'Hong';

firstName = 50;                    // Error
```

변수 Type

형을 생략하고 초기 값도 생략한 경우에는 변수 any 형으로 지정.

```
let lastName;                    // 초기화 안함
lastName = 'GilDong';
lastName = 20;                    // Error 아님
```

함수

정의하는 방법은 JavaScript과 동일하나 매개변수의 타입과 리턴 타입이 존재한다.

정의

```
function(매개변수1: 타입, 매개변수2: 타입, ..): 리턴값{
        기술할 명령..;
}
```

리턴 할 값이 없는 경우 void

함수

매개변수의 초기 값을 설정할 수 있다. 초기 값이 있는 매개변수는 정의한 매개변수의 가장 뒤에 기술해야 한다.

정의

```
function(매개변수1: 타입, 매개변수2: 타입=Value): void{
        기술할 명령..;
}
```

함수

함수 호출 시, 매개변수를 생략 가능하도록 할 수 있다. 생략되는 매개변수는 정의한 매개변수의 가장 뒤에 기술해야 한다.

정의

```
function(매개변수1: 타입, 매개변수?: 타입): void{
        기술할 명령..;
}
```

Class

ES2015부터 사용이 가능하다.

Java 프로그램과 동일하게 접근제한자, 생성자 메서드, 상속, 인터페이스 등을 지원한다

```
class ClassName {
        접근제한자 변수명: 변수타입;

        constructor(매개변수: 매개변수 타입..) { …}

        접근제한자 methodName( ) { … }
}
```

Class 상속

extends 키워드를 이용하여 단일 상속을 구현한다.

```
class ClassName extends SuperClassName{
        접근제한자 변수명: 변수타입;

        constructor(매개변수: 매개변수 타입..) {
                super(매겨변수, ...)
        }
        접근제한자 methodName( ) { ... }
}
```

Interface

Interface는 두 가지 목적으로 사용된다.

1. Object의 타입을 정의 할 목적으로 사용
2. class에서 추상 메서드 구현 등, Java 프로 그램과 동일한 사용 방법

1. Interface를 이용한 Object Type 설정(접근 제한자를 기술하지 않는다)

```
interface  InterfaceName {
        변수명: 변수타입;
        변수명: 변수타입;
        …
}
```

Interface

Interface는 두 가지 목적으로 사용된다.

1. Object의 타입을 정의 할 목적으로 사용
2. class에서 추상 메서드 구현 등, Java 프로 그램과 동일한 사용 방법

3. Interface를 이용한 추상메서드 설정(접근 제한자를 기술하지 않는다)

```
Interface  InterfaceName {
        methodName( ) { … };
        methodName( ) { … };
        …
}
```