타입스크립트 #0

작성자 : 이종호

작성일자 : 2021 – 01 -31

# 타입스크립트란 무엇일까 ?

타입스크립트는 2014년도에 발표한 자바스크립트의 super set 언어이며, 마이크로소프트에서 지원하는 자바스크립트에는 존재하지 않던 타입을 지원하는 언어입니다. 타입스크립트는 2019년 Pythone에 비하여, 낮은 인기를 보여주었지만, 2020년 하반기 pythone을 제치고 가장 인기 있는 언어 2위를 차지하였습니다. 타입스크립트가 인기 있는 이유 중 하나는 자바스크립트를 그대로 사용 할 수 있으면서도 런타임에 발생하는 에러를 최대한 줄일 수 있기 때문입니다.

자바스크립트는 시간이 지나며 기존에 비하여, 너무나도 많은 발전을 해왔고 다른 언어에 비하여 독특한 동작 방식과 강력한 패키지 모듈인 NPM과 YARN, 방대한 자료로 인하여 기존 자바스크립트 사용자가 다른 언어를 학습하거나, 타 언어 사용자의 유입이 매우 어려운 언어가 되어버렸습니다. 특히 변수에 타입을 지정하지 않고, 배열의 방식이 Array List인 점으로 인하여, 런타임 이후 어떠한 에러가 발생할 지 예측하는 것이 어려웠기 때문입니다.

# 타입스크립트를 왜 써야 하는 것인가 ?

OOP를 지원하는 언어인 C# , C++, JAVA와 같은 언어는 interface 혹은 abstract를 사용하여 어떤 class를 설계 할 때 interface에 맞추어 변수명, 프로퍼티명, 메서드명, 타입의 차이로 인한 에러를 컴파일 전에 방지 할 수 있게 됩니다.

현재 자바스크립트의 개발 환경은 과거 웹마스터라는 직군이 혼자서 서버부터 모든 페이지를 설계하고 구현하던 모습이 아닌, 다양한 사람들과 협업을 통하여 하나의 프로그램을 구현합니다. 그렇지만, 자바스크립트는 기존 OOP언어에서 지원하는 정적 타입이나, interface와 같은 것들을 지원하지 않기 때문에 설계와 기획에서 문서화된 interface를 따르게 됩니다. 그렇지만 사람은 언제나 실수 할 수 있는 법이기에, 변수 명이나, 데이터 타입을 인터페이스에 맞게 작성하지 않을 수 있습니다. 이런 경우 자바스크립트의 동적 타입 할당이나, 관대함으로 인하여, 런타임 오류로 이어 질 수 있습니다.

다음과 같은 예를 보겠습니다.

const isLogin = false;

const fruites = ["apple mango","mango apple",1]

document.getElementById("button").addEventListener('onClick',()=>{

for(let i=0; i < fruites.length; i++){

fruites[i].slice(1);

}

})

이 예제에서는 실수로 truites 배열 마지막에 string이 아닌 number타입의 1을 집어 넣었습니다. 위의 JS파일을 html파일에 임포트 시킨다면 브라우저에서는 아무런 에러를 발생시키지 않을 것입니다. 그러나, 버튼을 클릭하여 string타입의 정적 메서드인 slice를 1이라는 숫자에서 사용하게 되는 경우 에러를 발생 시킬 것 입니다. 즉 개발 당시에는 문제가 없다고 판단한 프로그램이 사용 시 치명적인 오류로 이어 질 수 있습니다. 또한 존재하지 않는 변수명으로 인한 에러도 많이 발생합니다.

다음 예를 보겠습니다.

const apple = "";

document.getElementById("button").addEventListener('onClick',()=>{

alert(applee);

})

Apple이라는 변수를 만들었지만 실수로 버튼을 클릭 시 발생하는 콜백에서 잘못된 변수명을 입력하게 되면 버튼을 클릭하기 전까지는 에러를 확인 할 수 없습니다.

타입 스크립트는 이러한 자바스크립트 개발자가 미쳐 발견하지 못한 타입체크, 변수체크를 자바스크립트로 변환 해주기 전에 지원함으로써 런타임시 발생할 수 있는 오류를 최대한 줄이도록 도와줍니다.

# 왜 정적 검사를 해야하는 것인가 ?

현재까지 여러분은 JEST와 같은 테스트 도구들을 이용하여, 자신이 개발한 프로그램을 검사하거나, 하지 않았을 겁니다. 적어도 콘솔 창으로 디버깅을 진행하였을 겁니다. 이러한 테스트는 프로그램의 오류를 최대한 잡아내려는 노력이 밑바탕 되었을 겁니다. 그리고 이런 테스트의 커버리지를 높일 수록 프로그램의 에러는 줄어들 것 입니다. 그러나, 테스트를 진행하는 것은 기획, 설계, 구현과 마찬가지로 시간과 인력을 크게 소모하는 작업이며, 어느 개발자도 자신이 만든 프로그램을 QA에게 전달하여, 혹독한 테스트를 받는 것을 원치 않을 겁니다. 또한 클라이언트는 빠른 출시를 원하고, 테스트에 들어가는 비용을 최대한 줄이기 위하여 커버리지를 높이지 않습니다. 그렇다면 여러분의 테스트를 거치지 않은 코드는 언제 어디서 터질지 모르는 시한폭탄과 같은 프로그램입니다.

중요한 자원인 시간을 줄이기 위해서 테스트의 커버리지를 높이지는 않지만, 자신의 프로그램이 시한폭탄과 같이 언제 터질지 모르고 마음을 졸이지 않고 대기하는 사람은 별로 없을 것 입니다. 이 때 타입스크립트는 시한폭탄과 같은 우리의 프로그램에서 걱정을 덜어줄 수 있는 테스트에 대한 큰 대안으로 작용 할 것 입니다. (물론 완벽한 프로그램을 위하여 테스트를 진행해야 합니다.)

# 타입스크립트를 위해 설치 해야 하는 것

NODE.js

NPM

타입스크립트 설치 npm install typescript -g 맥 사용자의 경우 전역 설치를 위해 sudo를 붙여주여야 한다.

# 타입스크립트 시작하기

const helloToUser = "반갑습니다. 자스민 여러분";

console.log(helloToUser)

/typescript/practice/app.js

터미널 , COMMAND

maeg@maeg-ui-MacBookPro jasmin-ex % tsc

hello.js 파일이 새롭게 생성이 됩니다.

var helloToUser = "반갑습니다. 자스민 여러분";

console.log(helloToUser);

현재 너무 간단한 방법으로 js파일이 생성이 되었습니다.

# ES6로 변환하기

현재 만들어진 hello.js는 const가 var로 변경된 것을 볼 수 있습니다. 이는 변환 과정에서 tsc모듈이 ES5이전으로 변경하는 것을 알 수 있는데, 이를 tsconfig.json파일로 쉽게 변경 가능합니다.

tsc --init

터미널, COMMAND

명령어 입력시 tsconfig.json파일이 새로 생성되는데, target란을 보면 es5로 지정되어 있지만 옆 주석을 보면 ES3가 기본 타겟으로 되어 있는 것을 확인 할 수 있습니다. 그래서 다음과 같이 tsconfigfile의 target과 lib부분을 변경 합니다 .

"target": "ES2015",

"module": "commonjs",

"lib": [

"es5"],

이후 tsc 파일명이 아닌 tsc 명령어를 입력하면 hello.js파일의 var가 const로 변경되는것을 확인 할 수 있습니다.

# 변수에 타입 입력 해보기

/practice/hello.ts

const helloToUser : string = "반갑습니다. 자스민 여러분";

console.log(helloToUser);

변수에 타입을 지정하려면 식별자 오른쪽에 : type 이라고 작성 하시면 됩니다. 현재 누가봐도 string임을 아는데 굳이 string을 지정해줄 필요가 있나요 ? 라고 물어본다면, 개발자의 실수를 예를 들 수 있습니다.

const helloToUser : string = 1;

console.log(helloToUser);

어떤 개발자가 실수로 입력 도중 string타입이 아닌 Number형태의 1을 helloToUser에 할당하였습니다. 이후 개발자가 프로젝트 빌드를 위해 트랜스 파일링을 진행하면 다음과 같이 잘 못된 데이터 타입의 값을 할당하려 시도 했다고, 경고메세지를 보낼 것 입니다. 이로 인해, 잘 못된 타입으로 인한 실수를 미연에 방지 할 수 있습니다.

# 매개변수에 타입 입력 해보기

/practice/hello.ts 파일을 다음과 같이 작성해보도록 하겠습니다.

const addFunc = (a, b) => console.log(a + b);

addFunc(10,20);

전혀 문제 없는 간단한 함수 이지만 어째서인지, a와 b가 any타입인거 같다고 js로의 트랜스 파일링을 거부합니다. 이 오류의 문제를 해결하는 방법은 ADD함수의 매개변수로 전달되는 a와 b의 타입을 특정 할 수 없어 오류가 생길 것 같으니, 매개변수의 타입을 지정 해주면 됩니다.

const addFunc = (a : number, b : number) => console.log(a + b);

addFunc(10,20);

# 본격 적으로 타입스크립트 공부하기 # 타입스크립트의 기본 타입

Boolean : true , false

Number : 10진수, 16진수, 2진수, 8진수등 포함

String : “” , ‘’ , ``

Array : [] ,type [], Array<type>

튜플 타입 : [type,type]

튜플 타입은 조금 생소 하실 분들이 많으리라 생각이 듭니다. 이는 주로 서버로부터 전달된 응답이 테이블 형태로 그대로 올 경우에, 볼 수 있습니다.

Any : any

알지 못하는 타입인 경우 사용 할 수 있습니다. 주로 서드파티 라이브러리등에서 온 객체들이 이 타입을 주로 사용 하게 될 것입니다. 이는 제이쿼리 객체나, moment같은 객체를 표현하기 적합합니다.

Void : void

Void는 undefined와 null을 의미 합니다. 단순히 사이드 이펙트만 행하는 함수의 경우, return 값이 undefined가 되기 때문에 void속성을 표기해주면 좋습니다.

Null , undefined : null과 undefined타입 또한 존재하지만, void로 표시하면 null과 undefined를 행할 수 있어 편리합니다 .

Object : Obejct

Object는 원시타입인 number, string, Boolean, symbol, null, undefined가 아닌 나머지를 의미합니다.

# 타입 단언

타입 단언은 조금 특별한 방법입니다. 가끔은 컴파일러 보다 개발자가 값에 대하여 더 잘 아는 경우가 존재합니다. 타입 단언은 어떤 값이 특정 값이니, 컴파일러는 걱정하지 않아도 된다 . 라고 표현 할 수 있습니다.

이 방법은 두가지가 존재하는데, 꺽쇠를 사용하는 angle-braket문법과 as를 사용하는 as문법이 존재합니다.

const helloToUser : string = "hi !";

const strLength : number = helloToUser.length;

//이건 string의 length가 number 타입을 반환하니 이건 괜찮다.

const crazyThings\_1 = (strLength).map(()=>{

return "???"

})

//이 경우는 strLength에 맵 메서드가 존재하지 않아 컴파일 도중에 에러가 발생한다.

const crazyThings\_2 = (<any>strLength).map(()=>{

return "???"

})

//타입 단언을 사용하면, 컴파일 단계에서 개발자가 특정 검사를 진행 했다고 인지하고 패스 한다.

const crazyThings\_3 = (strLength as any).map(()=>{

return "???"

})

//이 방법은 as문법으로 JSX와 typescript를 병행하는 경우에 사용된다.