Outlines

TypeScript

TypeScript

Agenda

Anders Hejlsberg
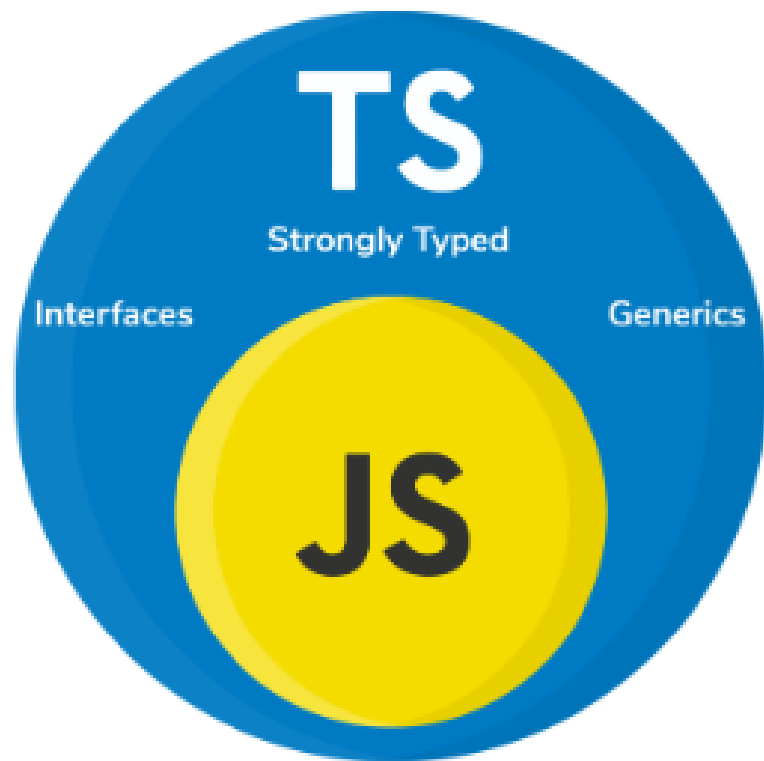
2012 , Microsoft

JavaScript Superset

Add new features & advantages to JavaScript

Language building up on JavaScript

Browser CAN'T execute it!

Typescript Vs Javascript

- TypesScript is an Object oriented programming language whereas JavaScript is a scripting language (with support for object oriented programming).

- TypeScript has static typing whereas JavaScript does not.

- TypeScript uses types and interfaces to describe how data is being used.

- TypeScript has interfaces which are a powerful way to define contracts within your code.

- TypeScript supports optional parameters for functions where JavaScript does not.

✓ **Transpiling allows you to generate ECMAScript**

✓ **Typescript supports JS libraries and API documentation**

✓ **Typescript introduces static typing**

✓ **Typescript uses NPM**

✓ **Typescript is easier to maintain**

✓ **Typescript makes it easier to use React, Angular, and Vue.**

Installation

The following tools you need to setup to start with TypeScript:
- Node js
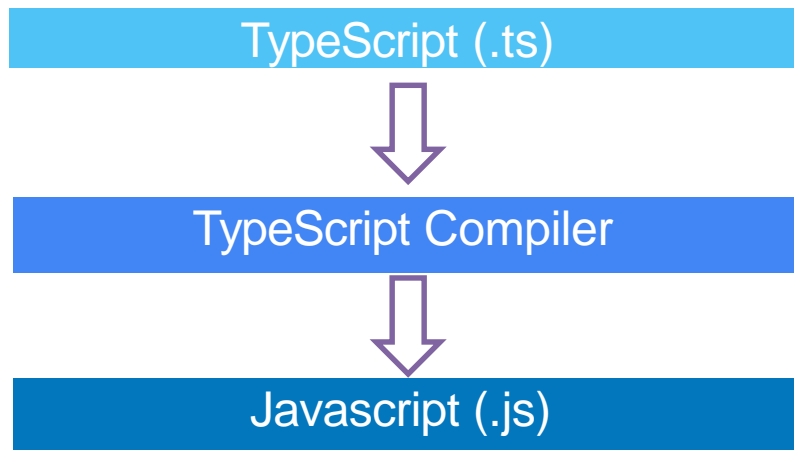- TypeScript compiler
- IDE (VsCode)

```
npm install -g typescript
```

```
tsc -v
```

```
Version 4.0.2
```

TypeScript Compilers

To compile TS code, we run the command tsc filename.ts. This will create a JS file of the same name, so we can eventually use it on the browser.

```
TypeScript (.ts)
        ↓
TypeScript Compiler
        ↓
Javascript (.js)
```

```
let message: string = 'Hello, World!';
console.log(message);
```

compile the app.ts file

```
tsc app.ts
```

EXPLORER

∨ HELLOWORLD

app.js

app.ts
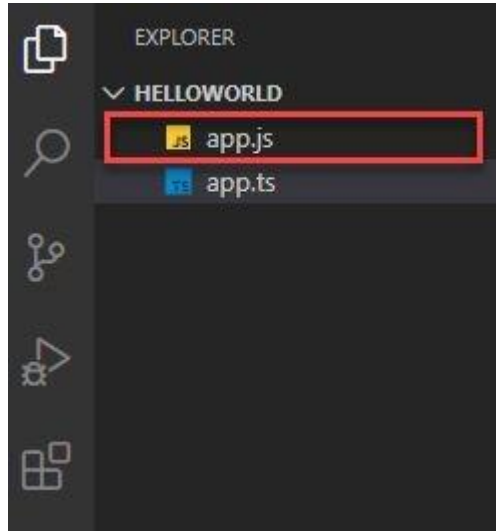
```
node app.js
```

You will see the output as
Hello, World!

JAVASCRIPT

```javascript
class Person {
  constructor(firstName, lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
  }

  getFullName() {
    return this.firstName + " " + this.lastName;
  }
}

const person = new Person("Monster", "lessons");
```

TYPESCRIPT

```typescript
class Person {
  firstName: string;
  lastName: string;

  constructor(firstName: string, lastName: string) {
    this.firstName = firstName;
    this.lastName = lastName;
  }

  getFullName(): string {
    return this.firstName + " " + this.lastName;
  }
}

const person = new Person("Monster", "lessons");
```

TypeScript project configuration :

- TypeScript uses a configuration file named tsconfig.json that holds a number of compilation options.
- We can use the tsc command-line compiler to generate a tsconfig.json file by using the --init command

  tsc –init

  tsc --target

TypeScript project configuration :

Option for `target` Property	Version of JavaScript
`"ES3"`	ECMAScript 3 (ES3)
`"ES5"`	ECMAScript 5 (ES5)
`"ES6"` or `"ES2015"`	ECMAScript 2015 (ES6)
`"ES2016"`	ECMAScript 2016 (ES7)
`"ES2017"`	ECMAScript 2017 (ES8)
`"ES2018"`	ECMAScript 2018 (ES9)
`"ES2019"`	ECMAScript 2019 (ES10)
`"ES2020"`	ECMAScript 2020 (ES11)
`"ESnext"`	Latest version

Watching files for changes :

- TypeScript also has a handy option that will watch an entire directory tree and if a file changes, it will automatically recompile the entire project.

tsc -w hello.ts

```
  1 let hello: string = "world";
  2 hello = [];
         [tsserver 2322] [E] Type 'undefined[]' is not assignable to type 'string'.
```

Basic Type

TypeScript inherits the built-in types from JavaScript. TypeScript types is categorized into:
- **Primitive type**
- **Objective Type**

Primitive types

The following illustrates the primitive types in TypeScript:

Name	Description
string	represents text data
number	represents numeric values
boolean	has true and false values
null	has one value: null
undefined	has one value: undefined . It is a default value of an uninitialized variable
symbol	represents a unique constant value

Object types:

- Function
- Arrays
- Classes
- Objects
- Tuples
- Enum

Function

TypeScript functions are the building blocks of readable, maintainable, and reusable code.

```typescript
const getFullName = (name: string, surname: string): string => {
  return name + " " + surname;
};

console.log(getFullName("Moster", "Lessons"));
```

```
4 const getFullName = (name: string, surname: string) => {
3   return name + " " + surname;
2 };
1
5 console.log(getFullName(true, ["foo"]));
        [tsserver 2345] [E] Argument of type 'true' is not assignable to parameter of
            type 'string'.
```

```typescript
function addNumbers(a: number, b: number): number {
    return a + b;
}


function addStrings(a: string, b: string): string {
    return a + b;
}
```

```typescript
function add(a: number | string, b: number | string): number | string {
    if (typeof a === 'number' && typeof b === 'number')
        return a + b;


    if (typeof a === 'string' && typeof b === 'string')
        return a + b;
}
```

Overloading function

```typescript
function add(a: number, b: number): number;
function add(a: string, b: string): string;
function add(a: any, b: any): any {
    return a + b;
}
```

```typescript
                    function add(a: number, b: number): number (+1 overload)
let result = add(10, 20);
```

ES5

```javascript
function Person(ssn, firstName, lastName) {

    this.ssn = ssn;

    this.firstName = firstName;

    this.lastName = lastName;

}
```

```javascript
Person.prototype.getFullName = function () {

    return `${this.firstName} ${this.lastName}`;

}
```

```javascript
let person = new Person('171-28-0926','John','Doe');

console.log(person.getFullName());
```

ES6

```javascript
class Person {

    ssn;

    firstName;

    lastName;


    constructor(ssn, firstName, lastName) {

        this.ssn = ssn;

        this.firstName = firstName;

        this.lastName = lastName;

    }


    getFullName() {

        return `${this.firstName} ${this.lastName}`;

    }

}
```

```javascript
let person = new Person('171-28-0926','John','Doe');
console.log(person.getFullName());
```

TypeScript

```typescript
class Person {
    ssn: string;
    firstName: string;
    lastName: string;

    constructor(ssn: string, firstName: string, lastName: string) {
        this.ssn = ssn;
        this.firstName = firstName;
        this.lastName = lastName;
    }

    getFullName(): string {
        return `${this.firstName} ${this.lastName}`;
    }
}
```

```typescript
let person = new Person(171280926, 'John', 'Doe');
```

```typescript
class Person {

    constructor(private firstName: string, private lastName: string) {

        this.firstName = firstName;

        this.lastName = lastName;

    }

    getFullName(): string {

        return `${this.firstName} ${this.lastName}`;

    }

    describe(): string {

        return `This is ${this.firstName} ${this.lastName}.`;

    }

}
```

```typescript
class Employee extends Person {

    constructor(

        firstName: string,

        lastName: string,

        private jobTitle: string) {


        // call the constructor of the Person class:

        super(firstName, lastName);

    }

}
```

```typescript
function getFullName(person: {
    firstName: string;
    lastName: string
}) {
    return `${person.firstName} ${person.lastName}`;
}

let person = {
    firstName: 'John',
    lastName: 'Doe'
};

console.log(getFullName(person));
```

```typescript
interface Person {
    firstName: string;
    lastName: string;
}
```

```typescript
function getFullName(person: Person) {
    return `${person.firstName} ${person.lastName}`;
}

let john = {
    firstName: 'John',
    lastName: 'Doe'
};

console.log(getFullName(john));
```

```
interface Mailable {
    send(email: string): boolean
    queue(email: string): boolean
}
```

```
interface FutureMailable extends Mailable {
    later(email: string, after: number): boolean
}
```

```
later(email: string, after: number): void
```

```typescript
function getRandomNumberElement(items: number[]): number {
    let randomIndex = Math.floor(Math.random() * items.length);
    return items[randomIndex];
}
```

```typescript
function getRandomStringElement(items: string[]): string {
    let randomIndex = Math.floor(Math.random() * items.length);
    return items[randomIndex];
}
```

```typescript
let numbers = [1, 5, 7, 4, 2, 9];
console.log(getRandomNumberElement(numbers));
```

```typescript
let colors = ['red', 'green', 'blue'];
console.log(getRandomStringElement(colors));
```

Using the any type

```typescript
function getRandomAnyElement(items: any[]): any {
    let randomIndex = Math.floor(Math.random() * items.length);
    return items[randomIndex];
}
```

```typescript
let numbers = [1, 5, 7, 4, 2, 9];
let colors = ['red', 'green', 'blue'];

console.log(getRandomAnyElement(numbers));
console.log(getRandomAnyElement(colors));
```

TypeScript Generic comes to rescue

```typescript
function getRandomElement<T>(items: T[]): T {

    let randomIndex = Math.floor(Math.random() * items.length);

    return items[randomIndex];

}
```

```typescript
let numbers = [1, 5, 7, 4, 2, 9];

let randomEle = getRandomElement(numbers);

console.log(randomEle);
```

```typescript
let numbers = [1, 5, 7, 4, 2, 9];

let returnElem: string;

returnElem = getRandomElement(numbers);  // compiler error
```

```typescript
function merge<U, V>(obj1: U, obj2: V) {
    return {
        ...obj1,
        ...obj2
    };
}
```

```typescript
let person = merge(
    { name: 'John' },
    { age: 25 }
);


console.log(result);
```

```
{ name: 'John', age: 25 }
```

```
let person = merge(
    { name: 'John' },
    25
);


console.log(person);
```

```
{ name: 'John' }
```

```typescript
function merge<U extends object, V extends object>(obj1: U, obj2: V) {
    return {
        ...obj1,
        ...obj2
    };
}
```

```typescript
let person = merge(
    { name: 'John' },
    25
);
```

```
Argument of type '25' is not assignable to parameter of type 'object'.
```

```typescript
interface Pair<K, V> {
    key: K;
    value: V;
}
```

```typescript
let month: Pair<string, number> = {
    key: 'Jan',
    value: 1
};

console.log(month);
```

Without enum

```
     const statuses = {
1      notStarted: 0,
2      inProgress: 1,
3      done: 2,
4    };
5
6    console.log(statuses.inProgress);
```

With enum

```
enum Status {
  NotStarted,
  InProgress,
  Done,
}

console.log(Status.InProgress);
```

```typescript
enum Status {
  NotStarted,
  InProgress,
  Done,
}

let notStrartedStatus: Status = Status.NotStarted;

notStrartedStatus = "foo";
          [tsserver 2322] [E] Type '"foo"' is not assignable to type 'Status'.
```

Conclusion

-TypeScript simplifies JavaScript code making it easier to read and understand.

- It gives us all the benefits of ES6, plus more productivity.

- Help us avoid painful bugs by type checking.

- Structural, rather than nominal

Any Questions?

References

- [prototype inheritance](#)
- [ES6 allowed you to define a class](#)
- https://www.typescripttutorial.net/
- https://www.typescriptlang.org/
- TypeScript [type annotations](#)

TypeScript

Installing and Compiling TypeScript

npm install -g typescript

tsc app.ts

```
export {};
function welcomePerson(person) {
  console.log(`Hey ${person.firstName} ${person.lastName}`);
  return `Hey ${person.firstName} ${person.lastName}`;
}

const james = {
  firstName: "James",
  lastName: "Quick"
};
welcomePerson(james);
```

```typescript
export {};
function welcomePerson(person: Person) {
  console.log(`Hey ${person.firstName} ${person.lastName}`);
  return `Hey ${person.firstName} ${person.lastName}`;
}
const james = {
  firstName: "James",
  lastName: "Quick"
};
welcomePerson(james);
interface Person {
  firstName: string;
  lastName: string;
}
```

Creating a TypeScript Config File

tsc --init