

TypeScript Introduction

Agenda

Datum	Onderwerp
1-2-2017	TypeScript Introduction
15-2-2017	Advanced TypeScript
1-3-2017	Angular Introduction
15-3-2017	Angular Building Blocks
29-3-2017	Components
12-4-2017	Ajax
26-4-2017	Data Entry
10-5-2017	Single Page Applications

Downloads: <http://bit.ly/centric-ts1>

What are we going to cover

What is TypeScript?

Why use TypeScript instead of ECMAScript.

Understand the main TypeScript language constructs:

- Type inference
- Classes
- Structural typing
- Type definitions

What is TypeScript?

TypeScript is a typed superset of ECMAScript 2015.

- Created at Microsoft by Anders Hejlsberg.

TypeScript is open source with an Apache 2 license.

Almost all valid JavaScript is also valid TypeScript.

Browsers don't understand the TypeScript language.

- You need to transpile it to ECMAScript first

Many open source projects use TypeScript

Angular

- Build with the TypeScript language
- Strongly encourages application developers to do the same

RxJS 5

- Build with the TypeScript language

MobX

- Build with the TypeScript language

Ionic

- Build with the TypeScript language
- Encourages application developers to do the same

Electron builder

- Build with the TypeScript language

Installing TypeScript

The command-line compiler is a Node.js package.

- Use **npm install -g typescript**
- Execute the **tsc** command to run the compiler

Many plugins for various editors

- Visual Studio 2013/2015/2017
- Visual Studio Code
- Sublime Text
- Atom
- WebStorm

Configuring the TypeScript compiler

Either use command line parameters with **tsc**.

- Easy and quick to do
- Harder to share configuration between tools like **tslint**

Alternatively create a **tsconfig.json** configuration file.

- Create one reusable configuration
- The TypeScript compiler uses it no matter how it was called

tsconfig.json

```
{
  "compilerOptions": {
    "module": "system",
    "noImplicitAny": true,
    "outFile": "../../built/local/tsc.js",
    "sourceMap": true
  },
  "include": [
    "src/**/*.ts"
  ],
  "exclude": [
    "node_modules"
  ]
}
```


TypeScript is superset of ECMAScript

Almost all valid JavaScript is also valid TypeScript.

- With a few exceptions due to type inference

TypeScript adds optional static typing.

TypeScript supports almost all ECMAScript 2015/2016 features.

- And some features further in the future

The TypeScript compiler compiles down to ECMAScript 3, 5 or 2015.

ECMAScript that causes errors in TypeScript

```
var person = {  
    age: 25  
};
```

```
// error TS2339: Property 'name' does not exist  
on type '{ age: number; }'.
```

```
person.name = 'John';
```

```
var element = document.getElementById('name');
```

```
// error TS2339: Property 'value' does not exist  
on type 'HTMLElement'.
```

```
console.log(element.value);
```

Converted to valid TypeScript

```
var person: any = {  
    age: 25  
};
```

```
person.name = 'John';
```

```
var element =  
    <HTMLInputElement>document.getElementById('name');  
console.log(element.value);
```

Basic Types

TypeScript supports all ECMAScript types.

- Boolean, number, string, object, array, null and undefined

And adds support for a number of extra types:

- Tuple
- Enum
- Any
- Void

Type inference

TypeScript tries to determine variable type based on usage.

With the any type you can declare variables to be dynamic and type less.

Type inference

```
// data is a number  
// Same as let data: number = 42;
```

```
let data = 42;
```

```
function subtract(x, y) {  
    return x - y;  
}
```

```
// difference is inferred to be a number  
// because of the - operator
```

```
let difference = subtract(1, 2);
```

```
function add(x: number, y: number) {  
    return x + y;  
}
```

```
// sum is inferred to be a number because of adding two numbers
```

```
let sum = add(1, 2);
```

Fat arrow functions

Shorthand notation to define functions.

- Syntactically very similar to C# lambda expressions

Preserve the same reference of this as the outer function.

- Done using an ECMAScript closure

Classes

TypeScript **classes** are very similar to standard ECMAScript 2015 classes.

- Constructor, inheritance, super etc.

TypeScript does add **private** and **readonly** modifiers but these only apply to TypeScript.

- Still publicly accessible in the generated JavaScript
- Use getter and setter to prevent that

Classes

```
class Cat {  
    name: string;  
  
    constructor(name: string) {  
        this.name = name;  
    }  
  
    sleep() {  
        console.log(this.name, 'is sleeping');  
    }  
}  
  
const zorro = new Cat('Zorro');  
zorro.sleep();
```

Parameter properties

```
class Cat {  
    constructor(private readonly name: string) {  
    }  
  
    sleep() {  
        console.log(this.name, 'is sleeping');  
    }  
}
```

```
const zorro = new Cat('Zorro');  
zorro.sleep();
```

Inheritance

Classes can be derived from other classes.

- Only single inheritance is possible

Mixins can be used for the same purpose

Inheritance

```
class Animal {  
    constructor(private name: string) {  
    }  
}
```

```
class Cat extends Animal {  
    constructor(name: string) {  
        super(name);  
    }  
}
```

Interfaces

TypeScript **interfaces** can be used to describe the shape of objects.

- Do not contain any actual code

These interfaces are only used at **compile time** by TypeScript.

- No JavaScript is generated for them

Interface names do not start with an I!

Interfaces

```
interface CatLike {  
    sleep()  
}
```

```
class Cat implements CatLike {  
    constructor(private readonly name: string) {}  
  
    sleep() {  
        console.log(this.name, 'is sleeping');  
    }  
}  
  
const zorro: CatLike = new Cat('Zorro');  
zorro.sleep();
```

Structural typing

The TypeScript compiler actually uses **structural typing**.

The compiler checks if the required type shapes match.

- Interfaces are just a named type descriptions

You can also describe the expected shape inline if you prefer.

- Interfaces and classes make a type more reusable than inlining

Structural typing

```
// Be like a CatLike interface
function beforeDinner(cat: CatLike) { cat.sleep(); }

// Be like a Cat
function eatDinner(cat: Cat) { cat.sleep(); }

// Have a sleep() function
function afterDinner(cat: {sleep()}) { cat.sleep(); }

const zorro = new Cat('Zorro');

beforeDinner(zorro);
eatDinner(zorro)
afterDinner(zorro);
```


Generics

Generic classes and functions are more flexible in their usage.

- Create types that depend on other types

You can add **constraints** to generics as needed.

Generics

```
class Cat {  
    constructor(public name : string) {}  
}  
class Fish {}  
class PetShop <T extends {name : string}> {  
    sell(pet : T) {  
        console.log('Selling', pet.name);  
    }  
}  
  
const shop = new PetShop();  
shop.sell(new Cat('Zorro'));  
  
// error TS2345: Argument of type 'Fish' is not  
// assignable to parameter of type '{ name: string; }'.  
// Property 'name' is missing in type 'Fish'.  
shop.sell(new Fish());
```

Working with existing libraries

Use type definitions with the public interface.

- Provides compile time checking for jQuery, AngularJS etc.

DefinitelyTyped is the main repository.

- <https://github.com/borisyankov/DefinitelyTyped>

Conclusion

TypeScript is a superset of ECMAScript 2015.

- Adds type annotations and checking
- And much more

Easy to get started with.

- Most valid JavaScript is also valid TypeScript
- Just rename your files and use the tsc compiler

Using TypeScript can reduce the number of potential bugs.

- Type inference and checking warns for possible errors

Makes writing large JavaScript applications a lot easier and safer.