Introduction to TypeScript

Course Map: Day 1

1. ASP.NET Core Architecture
2. Getting Started with ASP.NET Core
3. Design Patterns, Unit Testing
4. Entity Framework Core

Course Map: Day 2

5. Introduction to TypeScript

6. Using VS Code for TypeScript

7. Angular 2 Architecture

8. Using Angular CLI for Client Apps

Agenda

- History and importance of JavaScript
- What is TypeScript?
- Installing TypeScript
- Compiling TypeScript to JavaScript
- TypeScript language basics
- TypeScript type system

Get the Bits

github.com / tonysneed /

Kliant.AspNetCore-Angular

Importance of JavaScript

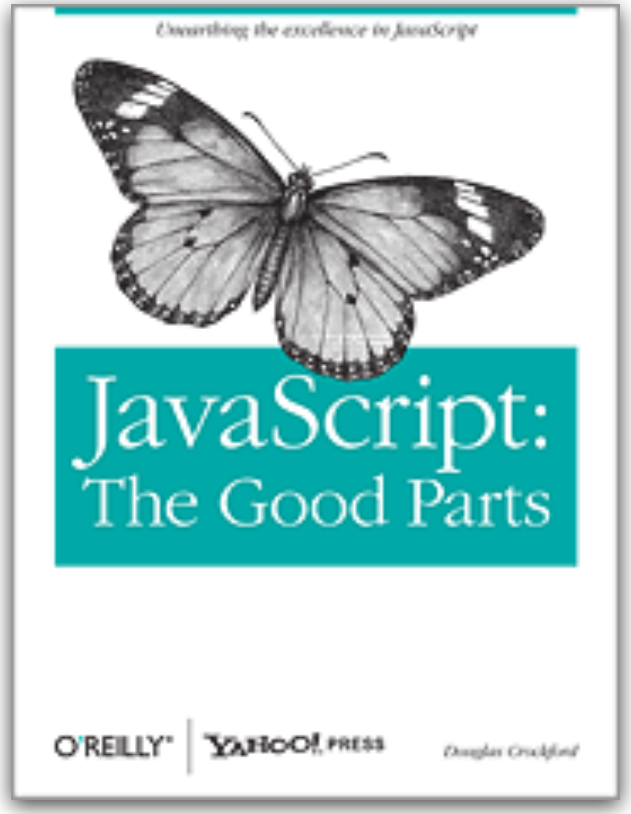
- Server Apps
- Web Browsers
- Cross-Platform Desktop
- Mobile Apps

History of JavaScript

- **1995**: JavaScript 1.0
  - Invented by Brendan Eich in 10 days
  - Has nothing to do with Java
- **1999**: ECMAScript 3
- **2009**: ECMAScript 5 (compatible with most browsers)
- **2015**: ECMAScript 2015 / ES6 (object-oriented features)
- **2016**: ECMAScript 2016 / ES7 (decorators, async/await)

Lack of Static Types

- **Type safety** and compile-time **syntax checking**
- Lack of **intellisense**
- **Large** code bases are difficult to **maintain**



TypeScript to the Rescue!

- Compile-time **syntax checking**
- **Intellisense**
- Code **refactoring**

What is TypeScript?

"TypeScript is a **superset** of JavaScript
that compiles to **plain** JavaScript."

http://www.typescriptlang.org

TypeScript Features

- Optional type annotations
- Interfaces and Generics
- Intellisense for JavaScript libraries
- Parity with latest JavaScript version: ECMAScript 2015
- Proposed features from future versions of JavaScript

Installing TypeScript using Node

1. Install Node.js
    - Has runtime built on Chrome's V8 engine
    - Uses event-driven, non-blocking I/O model
    - Includes Node Package Manager (NPM)
2. Install TypeScript globally using NPM

Installing TypeScript using Node


```
npm install -g typescript
```

Compiling TypeScript to JavaScript

```
// hello.ts

console.log("Hello TypeScript!");
```

```
tsc hello.ts
```

```
node hello.js

Hello TypeScript!
```

Demo: Getting Started with TypeScript

Declaring variables with var

- Function-scoped
- Hoisted to the top of the function

```
function foo(init) {
    if (init) {
        var x = 10; // Hoisted to top
    }
    return x;       // Scoped to containing function
}
foo(true);  // returns 10
foo(false); // returns undefined
```

Declaring variables with let

- Block-scoped
- Not visible outside of enclosing block

```
function foo(init) {
    if (init) {
        let x = 10; // Scoped to if block
    }
    return x;       // error TS2304: Cannot find name 'x'
}
```

TypeScript Basic Types

Type	Description
Boolean	Simple true/false
Number	Floating point values
String	Textual data
Array	Single or multi-dimensional arrays of values
Tuple	Array where element types are known
Enum	Numeric values with friendly names
Any	Dynamic content
Void	Return type of functions that do not return a value
Null	Indicates no value
Undefined	Indicates value not assigned

Type Annotations

- Constructs may by **optionally annotated** to indicate types
  - Compiler can check for **type safety**

```
function foo(text: string): number {
    // Type annotations allow compiler to help
    let len: number;
    len = text.length; // Statement completion
    return len;
}

let n: number = foo("Hello"); // returns 5
```

Template Strings

- String literals can contain **embedded expressions**
  - Surrounded by **backticks**
  - Expressions in form: **${expr}**

```
function foo() {
    let first = "James";
    let last = "Bond";
    let name = `The name is ${last}, ${first} ${last}.`;
    return name;
}
```

Nominal Typing

- Compatibility is based on **type names**
  - Student must **explicitly** extend Person

```
class Person {                          // C#: base class
    public string Name;
}
class Student : Person {                // C#: subclass extends base class
    public float Gpa;
}

Person person = new Student();          // Student is compatible with Person
```

Structural a.k.a Duck Typing

Need something that quacks?

A variable's **runtime** value determines its **behavior**

Structural Typing in TypeScript

- Types are compatible if they have **some members** in common

```
class Person {                          // Person and Student are structurally
compatible
    name: string;
}
class Student {
    name: string;
    gpa: number;
}
let p: Person = new Student();     // Student has a name – good enough!
```

Problem: any can be anything

- Can be useful to specify **more than one** type as valid

```
// Use any to add a number or a string
function addNumString(item: any, value: number) {
    return item + value; }

// Passing number or string make sense
let result1: any = addnumstring(2, 2);    // 4
let result2: any = addnumstring("2", 2);  // "22"

// Passing boolean has unexpected results
let result3: any = addnumstring(true, 2); // 3
```

Type Guards: typeof operator

- Use **typeof** operator to test for types at **runtime**

```
function addNumString(item: any, value: number): any {    // Test for number or string
    if (typeof item === "number") {
        let sum: number = item + value;
        return sum; }
    if (typeof item === "string") {
        let concat: string = item + value;
        return contat; }
    else {
        throw new Error("Wrong type!"); } }

let result3: any = addnumstring(true, 2);                    // Runtime error
```

Union Types

- Increase type safety by replacing any with a **union** type

```
function addNumString(item: number | string, value: number): {
    if (typeof item === "number") {
        let sum: number = item + value;
        return sum; }
    if (typeof item === "string") {
        let concat: string = item + value;
        return contat; }
}

let result3: any = addnumstring(true, 2);      // Compile error
```

Type Assertions and 'as' Operator

- Use **type assertions** to check for type member and then use it

```
function getActor(): Person | Dog { return new Dog() }
let actor = getActor();
if (actor.talk) {    // Compile error!
    actor.talk(); }

if ((<Person>actor).talk) {
    (<Person>actor).talk(); }

if ((actor as Person).talk) {
    (actor as Person).talk(); }
```

Type Guards: instanceof Operator

- Use **instanceof** operator to determine if object is a specific type

```
// Student is a Person, but not a Dog
class Person { }
class Student extends Person { }
class Dog { }

let s = new Student();

s instanceof Student; // True
s instanceof Person;  // True
s instanceof Dog;     // False
```

Demo: TypeScript Language Basics

Questions?