

Motivation TypeScript

Alex Thalhammer

Outlook

- Overview & Motivation TypeScript
- Angular Introduction

Overview



What is TypeScript?

- Superset of EcmaScript 6+ (2015+)
- Compiles to EcmaScript 6 (2015) or 5 (→ Differential loading for IE11)
- Introduces static typing (like known from C++, Java and others)
- Advantages
- Handy Code Completion (IDE)
- Makes Refactoring Easier

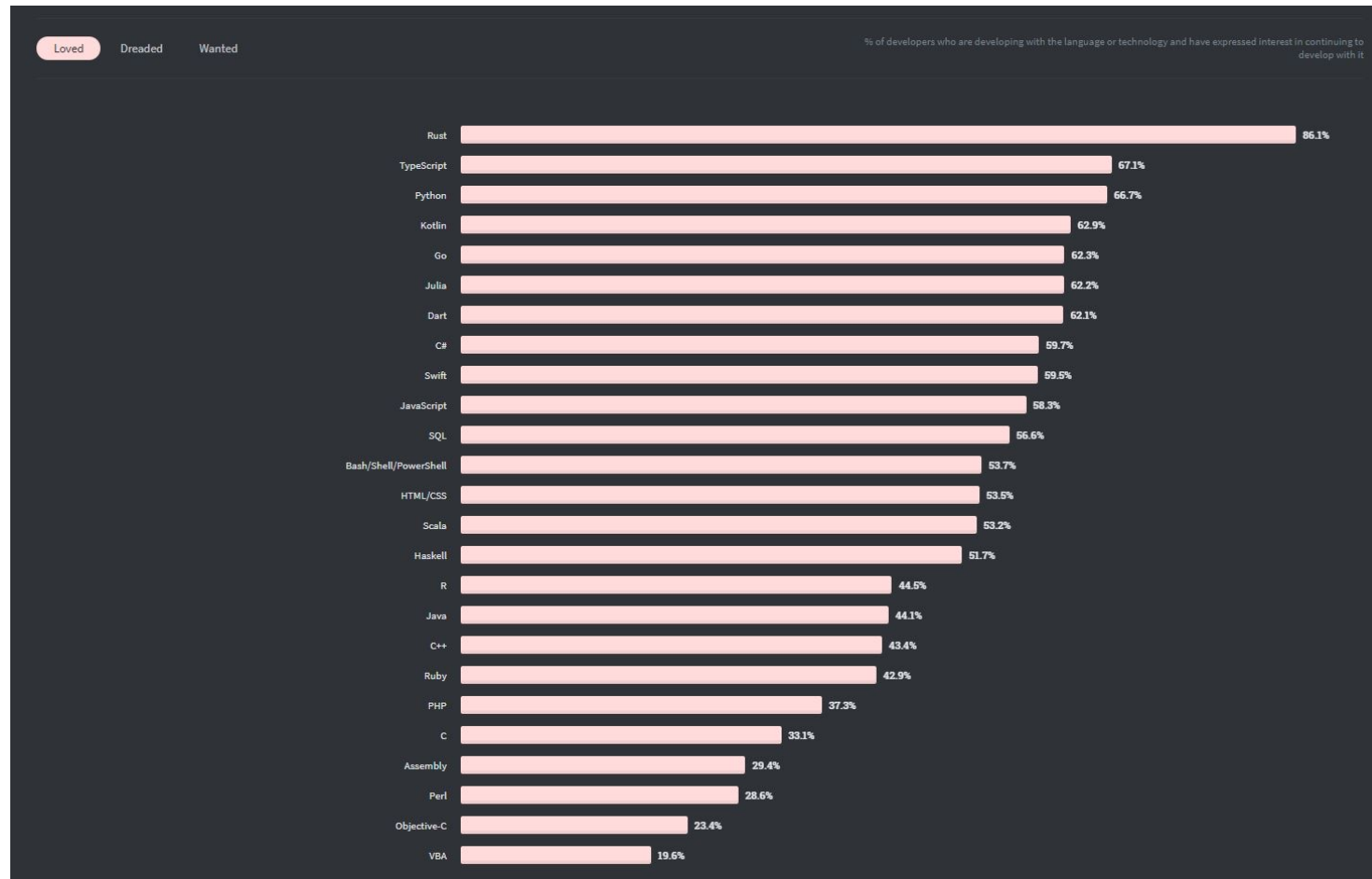


ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE



SOFTWARE
ARCHITECT

Devs <3 TypeScript (StackOverflow Survey'20)

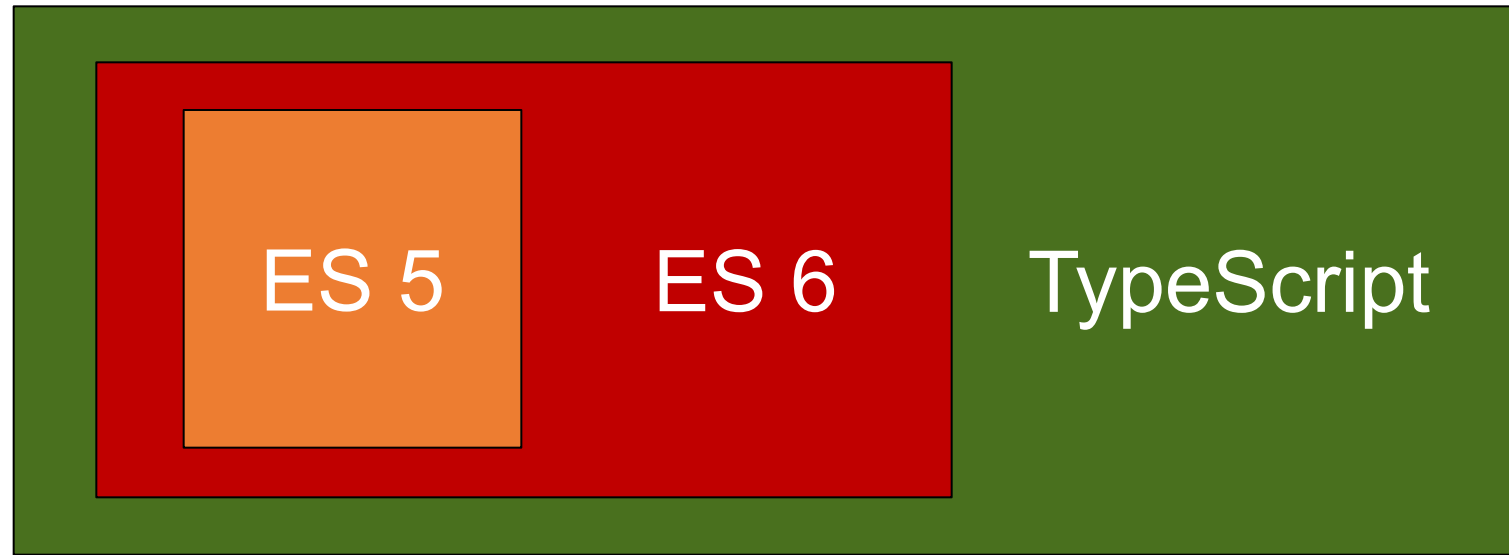


ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE



SOFTWARE
ARCHITECT

TypeScript & ES6



←
compilation

How can we use TypeScript?

- Compiler in terminal / shell
- Integration in VS Code or WebStorm
 - <http://www.typescriptlang.org/>
- Framework support
 - Angular
 - React
 - VueJS

Data types

| | | | |
|---------|------------|---------|-----------|
| number | string | boolean | function |
| object | any | null | undefined |
| classes | interfaces | enums | |



Access Modifier

public

protected

private

readonly

public is standard



A small example

A small example

```
export class Flight {  
  
    constructor(id: number) {  
        this.id = id;  
    }  
  
    public id: number; // int + double  
    protected from: string;  
    private to: string;  
    readonly date: string; // ISO date  
}
```



A small example

```
export class Flight {  
  
  constructor(public id: number) {  
    this.id = id;  
  }  
  
  public id: number; // int + double  
  protected from: string;  
  private to: string;  
  readonly date: string; // ISO date  
}
```



A small example

```
export class Flight {  
  
    constructor(public id: number) {}  
  
    protected from: string;  
    private to: string;  
    readonly date: string; // ISO date  
}
```



Another example

```
// flight-manager.ts
import { Flight } from './flight';

export class FlightManager {

    constructor(private cache: Array<Flight>) {}

    search(from: string, to: string): Flight[] { [...] }

    get count(): number { return this.cache.length; }
    get flights(): Flight[] { return this.cache; }
    set flights(c: Flight[]): void { this.cache = c; }
}
```

Inheritance

```
export class ExtendedFlugManager extends FlugManager {  
  
    constructor(cache: Flight[]) {  
        super(cache);  
    }  
  
    // Overwrite methods  
    search() {  
        [...]  
  
        return super.search();  
    }  
  
}
```

} Optionally



Project setup & ecosystem

Typical projekt struktur

| | |
|-------------------|--|
| src | • Source code (TypeScript, HTML, SCSS) |
| dist | • Dist |
| node_modules | • Libs |
| package.json | • Pointer to Libs and Scripts |
| tsconfig.json | • Config of TypeScript-Compiler |
| webpack.config.js | • Config of Bundler (webpack) |



Webpack

What is webpack?

- Build tool, bundling solution
- De facto Standard
 - Angular CLI, React, Vue
- Supports Node packages (→ de facto Standard)
- We need to customize webpack to support our MFE (more later)

webpack.dev.config

```
const commonConfig = require('./webpack.common.js');

module.exports = webpackMerge(commonConfig, {

  devtool: 'cheap-module-source-map',

  plugins: [
    new DefinePlugin({
      'ENV': '"development"'
    })
  ],

  devServer: {
    port: 8080
  }

});
```




NPM or Yarn & package.json

package.json

```
{  
  "dependencies": {  
    "@angular/common": "2.0.0",  
    [...]  
  },  
  [...]  
}
```



package.json

~ tilde

```
{
  "dependencies": {
    "@angular/common": "~2.0.0",
    [...]
  },
  [...]
}
```



package.json

```
{  
  "dependencies": {  
    "@angular/common": "~2.0.0",  
    [...]  
  },  
  [...]  
}
```



package.json

```
{
  "dependencies": {
    "@angular/common": "^2.0.0",
    [...]
  },
  [...]
}
```

^ caret



package.json

```
{  
  "dependencies": {  
    "@angular/common": "^2.0.0",  
    [...]  
  },  
  [...]  
}
```



package.json

```
{  
  "dependencies": {  
    "@angular/common": "^2.0.0",  
    [...]  
  },  
  "devDependencies": {  
    "webpack": "^1.12.9",  
    "webpack-dev-server": "^1.14.0",  
    [...]  
  },  
  [...]  
}
```



package.json

```
{
  "dependencies": {
    "@angular/common": "^2.0.0",
    [...]
  },
  "devDependencies": {
    "webpack": "^1.12.9",
    "webpack-dev-server": "^1.14.0",
    [...]
  },
  "scripts": {
    "webpack": "webpack"
    "start": "webpack-dev-server",
  },
  [...]
}
```



package.json

```
{
  "dependencies": {
    "@angular/common": "^2.0.0",
    [...]
  },
  "devDependencies": {
    "webpack": "^1.12.9",
    "webpack-dev-server": "^1.14.0",
    [...]
  },
  "scripts": {
    "webpack": "webpack --config webpack.dev.js"
    "start": "webpack-dev-server",
  },
  [...]
}
```



package.json

```
{
  "dependencies": {
    "@angular/common": "^2.0.0",
    [...]
  },
  "devDependencies": {
    "webpack": "^1.12.9",
    "webpack-dev-server": "^1.14.0",
    [...]
  },
  "scripts": {
    "webpack": "webpack",
    "start": "webpack-dev-server",
  },
  [...]
}
```

npm install

npm run webpack

npm start

TypeScript
Speedrun ;-)



Types



Types

let name: **string** = "Max Muster";

let plz: **number** = 12345;

let autor: **boolean** = true;



Implicit types

```
let name = "Max Muster"; // string
```

```
let plz = 12345; // number
```

```
let autor = true; // boolean
```

Only these 3 types!



Any

```
let website2: any = "http://www.softwarearchitekt.at";  
website2 = 1;
```

Try to avoid as much as possible!



Union-Types

```
let nameOrNumber: string | number;
```

```
nameOrNumber = "Max";
```

```
nameOrNumber = 17;
```



Union-Types as parameter

```
function showItem(speed: number | string) {  
  if (typeof speed === 'number') {  
    [...]  
  }  
  else if (typeof speed === 'string') {  
    [...]  
  }  
}
```



Lambda statements

```
function filter(input: number[], callback: (item: number) => boolean): number[] {  
  
    const result: number[] = []; // new Array<number>();  
  
    for (let i: number = 0; i < input.length; i++) {  
        if (callback(input[i])) result.push(input[i]);  
    }  
  
    return result;  
}
```

```
let result = filter([1, 2, 3, 4, 5, 6], (item: number) => item % 2 === 0);
```



Type Assertions



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE



SOFTWARE
ARCHITECT

Type Assertions

```
let k: Contact = new Client(123, "Max Muster", "Essen");
```


Type Assertions

```
let k: Contact = new Client(123, "Max Muster", "Essen");
```

```
[...]
```

```
let art = k.clientAttr; // won't work
```

Type Casting

```
let k: Contact = new Client(123, "Max Muster", "Essen");
```

```
[...]
```

```
let client = k as Client;
```

```
let art = client.clientAttr; // OK
```



Type Casting (Alternative)

```
let k: Contact = new Client(123, "Max Muster", "Essen");
```

```
[...]
```

```
let client = <Client>k;
```

```
let art = client.clientAttr; // OK
```



Interfaces

```
interface IContact {  
    id: number;  
    name: string;  
    location?: string;  
    plz: number;  
    date: any;  
    getInfo(): string;  
}
```

```
class Contact implements IContact {  
    [...]  
}
```



Generics

```
class ReadOnly<T> {  
    private data: T;  
    constructor(data: T) {  
        this.data = data;  
    }  
    public getData(): T {  
        return this.data;  
    }  
}
```

```
let readOnlyNumber = new ReadOnly<number>(42);  
console.debug(readOnlyNumber.getData());
```



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE



SOFTWARE
ARCHITECT

Functions



Functions

```
function sayHello(name: string = "noname"): void {  
    console.debug("Hallo " + name);  
}
```

```
sayHello("Max");
```

```
sayHello();
```



Optional Parameters

```
function sayHello(name?: string): void {  
    if (name) {  
        console.debug("Hallo " + name);  
    } else {  
        console.debug("Hallo!");  
    }  
}
```



Enums



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE



SOFTWARE
ARCHITECT

Enums

- `enum Direction { UP, DOWN, LEFT, RIGHT }`
`let d = Direction.UP;`
- `enum Direction { UP = 7, DOWN, LEFT, RIGHT }`
- `type Direction = 'UP' | 'DOWN' | 'LEFT' | 'RIGHT';`
`let d: Direction = 'UP';`



Recommendations on TypeScript

Use prettier!

Use ESLint (standard since NG 11)

- Avoid any!
- Use strict typing!
- Except implicit types 😊
- Also for functions params and return types!



Demo

Show prettier and ESLint



Now you know TypeScript

- Use it and you're gonna love it
 - If you have used JS before



- Any questions left over?



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE



SOFTWARE
ARCHITECT