TYPESCRIPT

ppedv AG, Vadzim Naumchyk

INHALT

facts

anhang

GETTING STARTED

#TYPESCRIPT > IDEA

'#ts'

WAS IST TS

- eine Programmiersprache
- Obermenge von JavaScript
- aktuelle Version 4.0.5 (27.10.2020) #checkForUpdates

WOZU IST TS

- um Fehler abzufangen (durch Typisierung und Code-Analyse-Tools)
- um Code-Patterns besser umzusetzen
- um schneller zu programmieren
- um schneller zu debuggen
- um Interfaces nutzen zu können
- andere Erweiterungen

STARTING LINKS

OFFIZIELLE QUELLEN

- HOMEPAGE https://www.typescriptlang.org/
- DOCS https://www.typescriptlang.org/docs/home.html
- CODE https://github.com/microsoft/TypeScript
- BLOG https://devblogs.microsoft.com/typescript/

STARTING TOOLS

- Visual Studio Code (VSC)
- nodejs & npmjs (npm i -g typescript)
- TypeScript Compiler tsc (tsc yourfile.ts)
- VSC Erweiterungen
    - TSLint / ESLint (muss für TS noch eingestellt werden)
    - open in browser
    - live server (auto update im Browser bei Änderungen in HTML,JS oder CSS)
    - JavaScript Snippets
    - auto rename tag von Jun Han

TS INSTALL

```
npm install -g typescript
npm init -y (oder strg ö) //für package.json
npm install @types/node -save-dev
```

tsconfig.json anlegen mit

```
{
    "compilerOptions": {
        "module": "commonjs",
        "sourceMap": true,
        "watch": true
    }
}
```

TS UPDATE

```
npm install -g typescript@latest
```

TS USE

TS TO JS

1. um ein TS-Projekt mit tsconfig.json zu initializieren

```
tsc --init
```

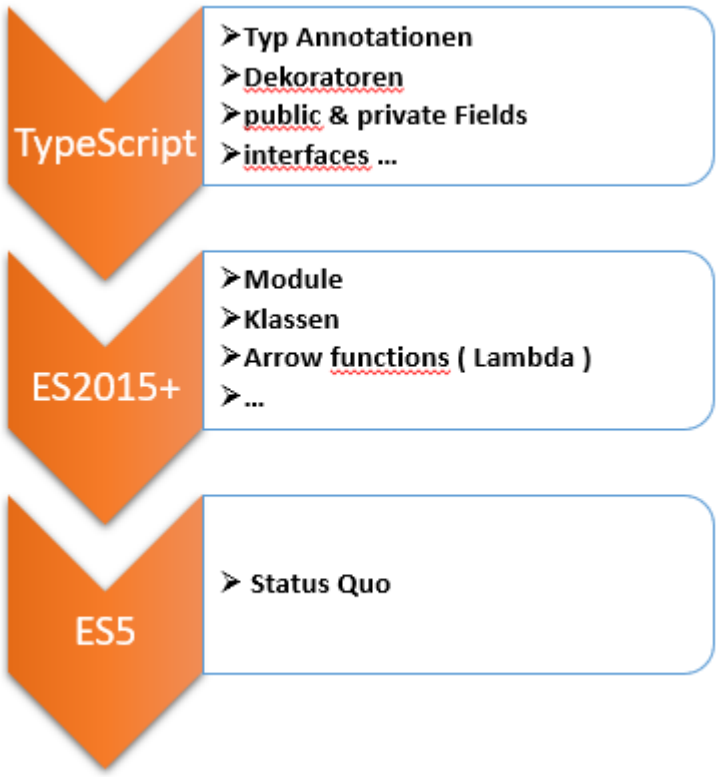
2. ts-Datei anlegen mit gewünschtem Code
3. im Terminal `tsc dateiname.ts` oder in ts-Datei `strg shift b` – Datei builden
4. die erstellte js-Datei darf jetzt in html verlinkt werden

---

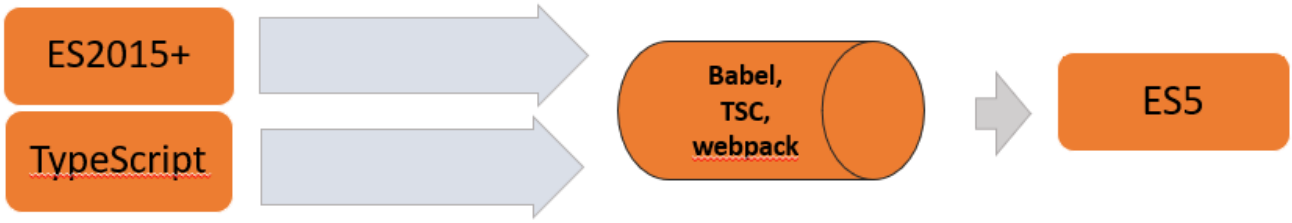
zurück zu ABLAUF.md

TS > ECOSYSTEM

JS SUPERSET



COMPILING



#TSC > IDEA

WAS IST TSC

- ein command line interface Utensil
- tsc: TypeScript Compiler

TSCONFIG.JSON

WAS WIRD UNTER TSCONFIG GEMEINT

eine .json-Datei mit Konfiguration für den Compiler

Anwesenheit der tsconfig.json-Datei in einem Verzeichnes markiert es als Root von einem TS-Projekt.

COMPILER CONFIGURATION

```
"compileOnSave": false,
"compilerOptions": {},
"files": [ "core.ts", "types.ts" ],
"include": [ "src/**/*" ],
"exclude": [ "node_modules", "**/*.spec.ts" ]
```

COMPILER OPTIONS

```
"compilerOptions": {
    "baseUrl": "./",
    "outDir": "./dist/out-tsc",
    "sourceMap": true,
    "declaration": false,
    "downlevelIteration": true,
    "experimentalDecorators": true,
    "module": "esnext",
    "moduleResolution": "node",
    "noImplicitAny": true,
    "removeComments": true,
    "preserveConstEnums": true,
    "sourceMap": true,
    "importHelpers": true,
    "target": "es2015",
    "typeRoots": [
      "node_modules/@types"
    ],
    "lib": [
      "es2018",
      "dom"
    ]
}
```

TYPES

TYPE SYSTEM > INTRO

Static Typing

- Datentypen können angegeben werden und unterstützen insbesondere die Entwicklungsumgebung:
  - Auto-Vervollständigung
  - Fehlermeldungen bei nicht passenden Datentypen
- Beim build: TypeScript wird in JavaScript übersetzt, alle Typeninformationen gehen dabei verloren

PRIMITIVES

same as in JS:

```
let completed: boolean = false;
let age: number = 32;
let name: string = 'Andreas';
```

ANY

any: lässt alle Typen zu

```
let inputBox: any = document.querySelector('#inputbox');
```

VOID

- void: umfasst undefined und null
- nützlich in den Funktionen ohne `return` Anweisung

UNION TYPE

```
export type message = string | null | undefined
```

TYPE ALIASES

```
type C = { a: string, b?: number }
function f({ a, b }: C): void {
```

```
// ...
}
```

TYPE ASSERTION > ANGLE BRACKETS & AS

Zwei Wege

1. angle-brackets syntax
2. as-syntax

```
// 1.
let someValue: any = "this is a string";
let strLength: number = (<string>someValue).length;
// 2.
let someValue: any = "this is a string";
let strLength: number = (someValue as string).length;
```

NON NULLABLE TYPES

Normalerweise akzeptieren alle Datentypen in TS neben eigenen Werten noch 'null' und 'undefined'. Ist das nicht gewünscht, dann siehe hier:

https://github.com/Microsoft/TypeScript/pull/7140

FUNCTIONS

RETURNING NO DATA TYPE

```
function warnUser(): void {
  alert('this is a warning message');
}
```

ARGS & RETURN TYPES

Wir können Parametertypen und Rückgabetypen angeben

```
function repeatString(text: string, times: number): string {
  return ...;
}
```

OPT ARGS

Optionale Parameter:

```
function buildName(
    firstName: string, lastName?: string
): string
{
  return firstName + ' ' + lastName;
}
```

DATA STRUCTURES

OBJECT

Beispiele:

```
let p: {name: string, age: number} = getPerson();
console.log(p.age);
// oder
interface Person {
  name: string,
  age: number
}
let p: Person = getPerson();
```

ARRAYS

Beispiele:

```
let names: string[] = ['Anna', 'Bernhard', 'Caro'];
let amounts: number[] = [3, 10, 23];
let list: any[] = [1, true, "free"];
// alternative with generic array type
let names: Array<string> = ['Anna', 'Bernhard', 'Caro'];
```

OPERATORS > KEYOF

- index type query operator
- für jeden Typ T, `keyof T` ist Union von bekannten, öffentlichen Eigenschaftsnamen von T

```
interface Car {
manufacturer: string;
model: string;
year: number;
}
let carProps: keyof Car; // the union of ('manufacturer' | 'model' | 'year')
```

INTERFACES

#INTERFACE > IDEA

WAS IST EIN INTERFACE

WOZU IST EIN INTERFACE

in TS ist es möglich, eine Variable vom angelegten Interface zu deklarieren. Man braucht also keine Klasse, die dieses Interface implementiert diese Variable ist kompatibel mit Objekten und mit Klassen, die (u.a.) die gleichen Member haben wie das Interface

```
interface Person {
    firstName: string;
    lastName: string;
}
function greetUser(person: Person) {
    return "Hello, " + person.firstName + " " + person.lastName;
}
let user = { firstName: "Max", lastName: "Mustermann"};
document.body.textContent = greetUser(user);
```

READONLY

With readonly the TypeScript compiler checks for unintended property mutation

```
interface Todo {
    readonly text: string;
    readonly done: boolean;
}
```

CLASSES

TS > CLASSES > SHORTHAND

Beim Gebrauch von `public` bei den Argumenten im Konstruktor werden die entsprechenden Eigenschaften automatisch angelegt.

```
class Student {
    fullName: string;
    constructor(public firstName: string, public middleInitial: string, public
lastName: string) {
        this.fullName = firstName + " " + middleInitial +
                " " + lastName;
    }
}
```

GENERICS

EXAMPLES

type variable (normally T)

```
function identity<T>(arg: T): T { return arg; }
let output = identity<string>("myString");
function getProperty<T, K extends keyof T>(obj: T, key: K) { return obj[key]; }
```

DECORATORS

#DECORATORS > IDEA

WAS IST EIN DEKORATOR

- eine Funktion

WOZU IST EIN DEKORATOR

- Mit Dekoratoren lassen sich Funktionen und Klassen nach ihrer Erstellung verändern

```
// Hypothetischer cache-Decorator,
// der die Resultate eines Funktionsaufrufs speichert
import { cache } from 'cache.js';
@cache
function getResults() {
  return this.results;
}
```

DECORATORS > EXAMPLE

In Angular werden Decorators verwendet, um Metadaten zu einer Klasse zu ergänzen:

```
@Component({
    selector: 'app-root',
    templateUrl: './app.component.html',
    styleUrls: ['./app.component.css']
})
export class AppComponent {
    name = 'Vadzim';
}
```

SCOPES

PUBLIC & PRIVATE

Private & Public Properties

```
class ClockComponent {
  private formatTime(time) {
    return …
  }
  public start() {
    …
  }
}
```

SCOPES IN CONSTRUCTOR

```
class Person {
  constructor(public name: string, public age: number) {}
}

// Kurzform für:

class Person {
  name: string;
  age: number;
  constructor(name: string, age: number) {
    this.name = name;
    this.age = age;
  }
}
```

FACTS

TS VERSIONS

Datum	Version
2012, 1. Okt	erschienen
2014, 6. Okt	v. 1.1
2016, 22. Sep	v. 2.0
2018, 30. Jul	v. 3.0
...	...
2019, 9. Jul	v. 3.5.3
2019, 28. Aug	v. 3.6
2019, 10. Sep	v. 3.6.3

TS > COMMUNITY

TWITTER https://twitter.com/typescript/

ANHANG

HASHTAGS

In die Suchfunktion: # und etwas aus der Liste:

- decorator
- seitV3.5
- seitV3.6
- TS
- TSC