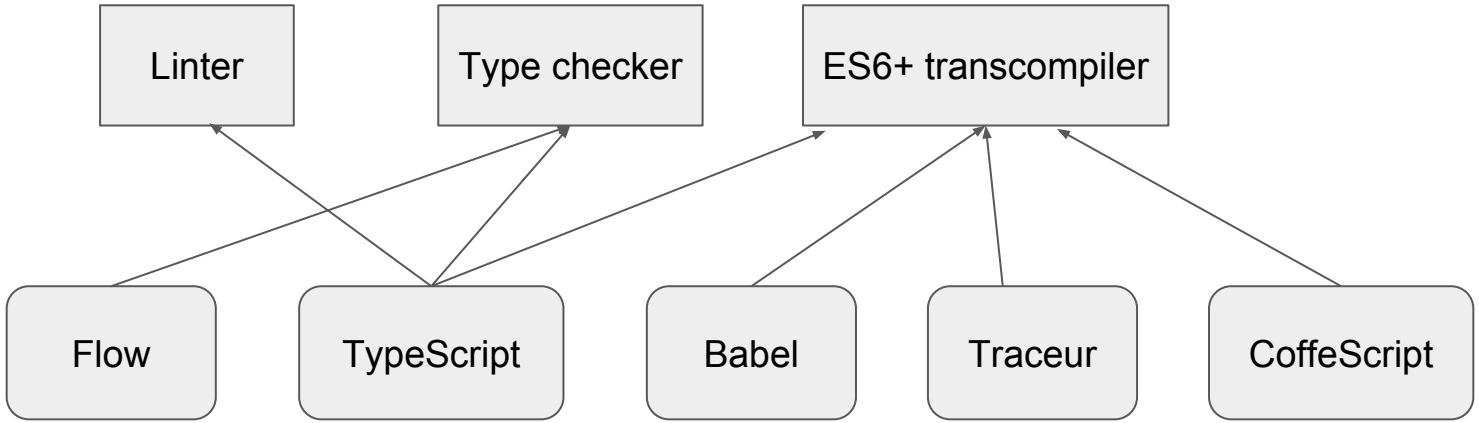
TypeScript

interesting features and how-to’s, from the trenches experience with migrating JS code to TS

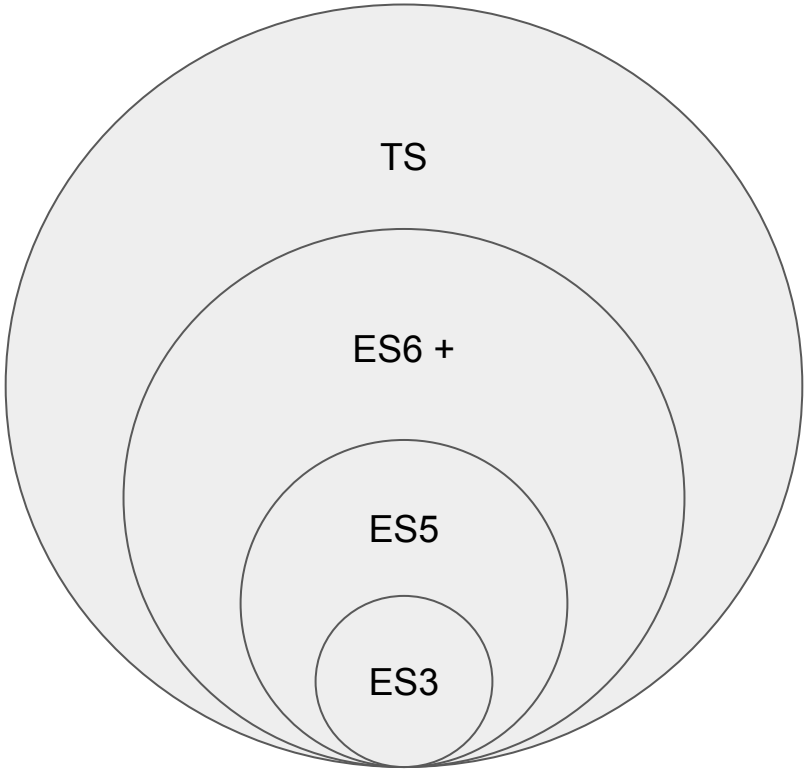
about me


```
let me = {
    name : "Dusan",
    surname : "Rostar",
    birth: {
        city: "Zvolen",
        year: 1981
    },
    school: {
        name: "Zilinska univerzita",
        year: 2005 // 12 y on various dev positions
    },
    work: {
        currentEmployer: "Asseco Solutions SK/DE"
    },
    twitter: "https://twitter.com/rostacik",
    blog: "http://rostacik.net/",
    linkedin: "https://www.linkedin.com/in/rostar/"
};
```

main features of TS



where it fits in the big picture



what is TypeScript

- is (typed) superset of JS (all your JS is already TS code)
- was created and is maintained by MS on GitHub
- brings (compile time) types to JS (no part of TS is used at run time)
- brings new ES6(ES2015, 2016, 2017) features to JS of your choice
- brings classes and modules to JS (if targeting older ES versions)
- can target different ES versions (ES3, ES5, ES6 + up)
- brings IntelliSense to compatible IDEs for all your code
- supports JS for client and server side (browser and node)
- brings some patterns to your JS
- has support for your IDE of choice
- built in support for React
- https://github.com/Microsoft/TypeScript

how to install TypeScript

- install TypeScript tooling for Visual Studio 201x - find TypeScript tooling on https://blogs.msdn.microsoft.com/typescript/ page with annoucncement
- get Visual Studio Code IDE (VS Code ships with TypeScript inside)
- NPM - npm install -g typescript (global node package)
- play with TS in your browser - https://www.typescriptlang.org/play/ or https://jsfiddle.net/

how to setup build

full VS

- let full VS do the build (full VS can only build all .ts files at the same time in one go)

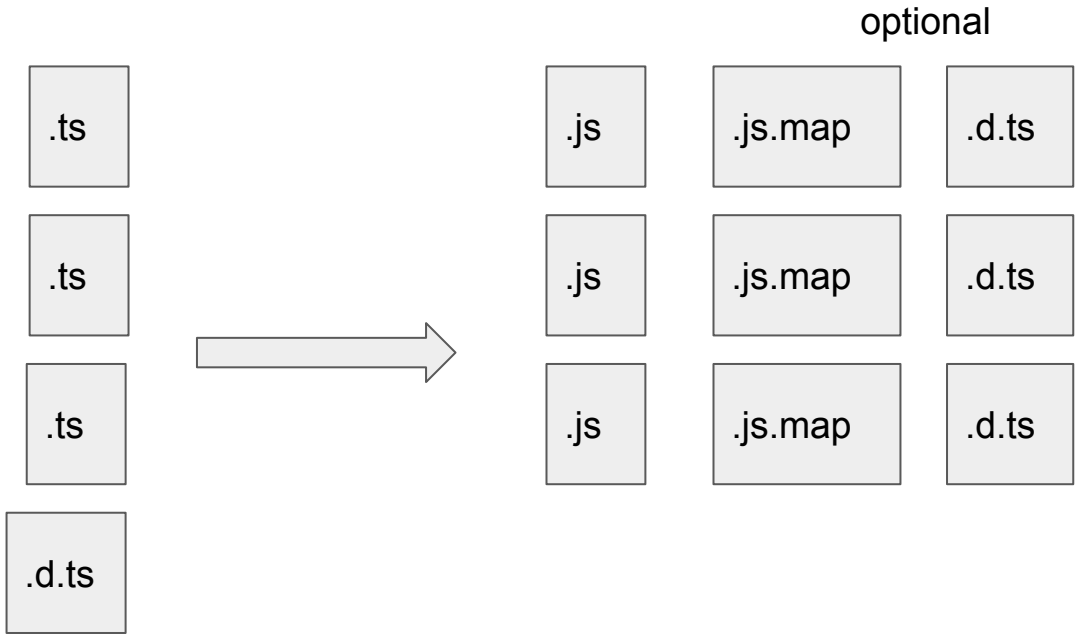
VS Code

- let VS Code build .ts files with built in TS compiler
- setup build task (VS Code task) and install **TS as npm package**

cmd line

- setup build pipeline in some task manager like **gulp, grunt**
- setup build in package manager like **browserify, webpack**

what happens during build time



how can we distribute code written in TS

- hand over all .ts files (consumer will have to build .js from .ts everytime)
- hand over generated .js files + generated .d.ts files (TS will get definitions from .d.ts files, .js files will be used at runtime)

types from 3rd. party libs

- from TS2+ all .d.ts for external code are in https://www.npmjs.com/~types
- npm package @types/{libNameHere} should be in package.json as devDependency
- if there is no .d.ts file for your lib, you have to generate one (https://www.npmjs.com/package/dts-gen)
- TS is distributed with DOM .d.ts library so it knows what is document or window object and it knows their parameters/interfaces

basic TS concepts - Types

let done**: boolean** = true;

basic types : boolean, number, string, Array / Array<T>, tuple - [T1, T2, ….], enum, any, void, null, undefined, never

advanced types : intersection type = U & T, union type = U | T

typeguards : small custom fn that will decide about the type, or typeof, or instanceof

string literal type : exact string values (“val1” | “val2” | “val3”), used to distinguish overloads

Interfaces

- holds shapes of object
- doesn’t compile to any JS
- most of the time descriptions of objects “sit” in .d.ts files
- describe shape of 3rd party JS code in our project or shape of class that will implement interface

Classes

- classes will come with ES6 to all browsers, but TS transcompiles now,
- great for devs comming from Java, C# world,
- supports inheritance (extends keyword),
- supports public, private and protected modifiers (public by default),
- supports abstract classes
- supports readonly modifier

Some cool types features

Union types - object can be of T1 or T2

```
let a: string | Array<string> = "a";
a = ["b", "c"];
```

Intersection types - object can be T1 and T2 at the same time

```
interface IHasProp1 {
    prop1: string;
}

interface IHasProp2 {
    prop2: number;
}

let o: IHasProp1 & IHasProp2;
```

ES6+ stuff - async / await

https://twitter.com/manekinekko/status/855824609299636230

- async / await is based on awaiting of function returning Promise<T>
- Promises as such are also supported (chaining of .then)
- dependent on setting in tsconfig target can be ES5 (based on state machine) or ES6 (yield - generator functions) = different JS (ES6 is more readable)

linter stuff

- control flow analysis (unreachable code)
- noImplicitAny - Warn on expressions and declarations with an implied 'any' type
- noImplicitThis - Raise error on 'this' expressions with an implied any type
- noUnusedLocals - Report errors on unused locals. Requires TypeScript version 2.0 or later
- noUnusedParameters - Report errors on unused parameters. Requires TypeScript version 2.0 or later
- noImplicitReturns - Report error when not all code paths in function return a value
- noFallthroughCasesInSwitch - Report errors for fallthrough cases in switch statement
- strictNullChecks - Enable strict null checks

sample repo (web or node)

project root

node_modules

@types

node/jquery/chai/mocha/etc…

typescript (compiler)

.ts files

tsconfig.json

package.json

how to convert existing JS code base

“older” style : every JS is TS = rename .js to .ts and ~~run away~~ / ~~watch it burn~~ iterate

- rename .js to .ts
- decide how TS build will be done
- add .d.ts for 3rd party libraries
- add tsconfig.json

PS> node .\node_modules\gulp\bin\gulp.js someTSBuildTaskName --color | Select-String -NotMatch "TS2365" | Select-String -NotMatch "TS2345"

http://rostacik.net/2016/11/10/how-to-filter-particular-typescript-errors-in-build-result/

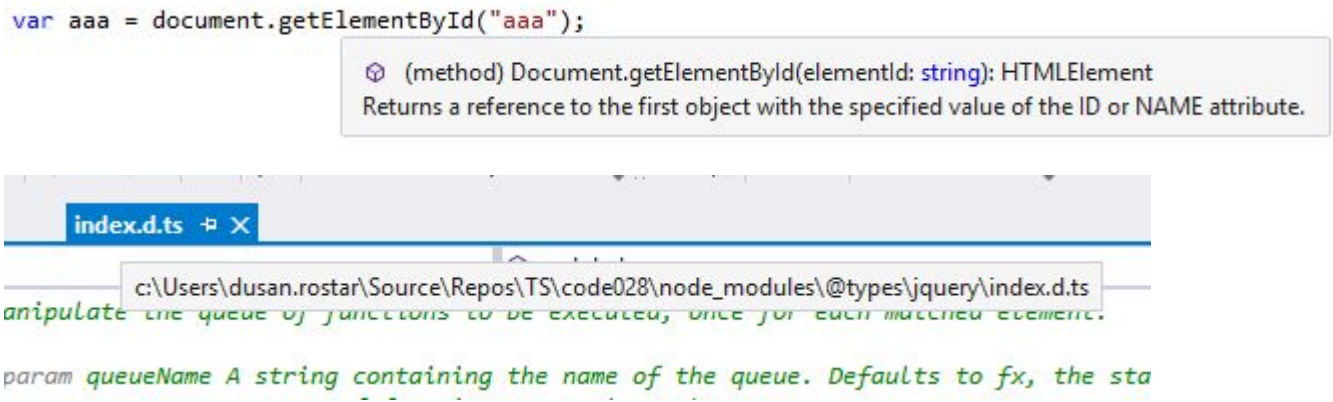
how to convert existing JS code base - “newer style”

from TS 2.3.2 + VS Code , TS is able to bring (checks) types to plain JS files = no need to convert them to TS

https://code.visualstudio.com/updates/v1_12#_type-checking-for-javascript-files

- per file - `// @ts-check`
- whole project - setting in VS Code or adding jsconfig.json

types in JS in full VS



full VS is loading type information from its own cache and from @types folder in node_modules folder in the repo.

conversion pitfalls - optional param

```
function optionalParam(name: string, surname: string, age?: number) {
    console.log(age);
}

optionalParam("name", "surname");
```

it’s OK to not pass the optional param, but it’s not OK to not check if we really have it in the function

parameter with default value

old code :

```
if (param == null) {
        param = someVariable;
}
```

new code :

```
function giveMeName(name: string = "Jozef") {

   console.log(name);

}
```

automagicall event: Event parameter

```
function myFn()
    console.log(event.srcElement);
}
```

var event: Event

in IE and Chrome we always have

just adding property on some object

no modules

```
window.iAmProp = "aaaa";
```

any

Property 'iAmProp' does not exist on type 'Window'.

```
interface Window {
  iAmProp: string;
}

window.iAmProp = "aaaa";
```

(property) Window.iAmProp: string

with modules

```
let windExt: (Window & IHasProp) = <any>window;
windExt.iAmProp
```

(property) IHasProp.iAmProp: string

different types of same variable

- most of the time we fall back to type coercion (one operand will be converted)

we can

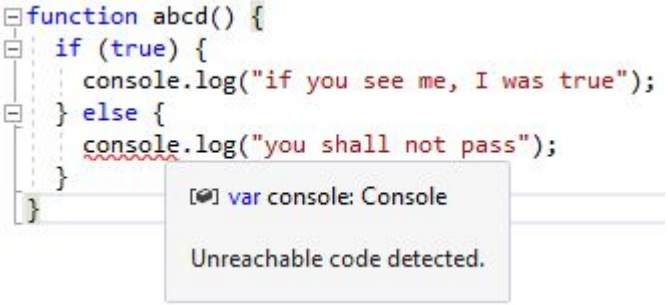
- convert by hand String(iAmNumber) === “iAmString”
- make var of type number | string;

most often seen in “accumulator” variables - temp var for strings, arrays, whathaveyou

undeclared vars

variable without var, where you have to find it using is was intention or not.

hidden gems (errors)



accessing stuff from DOM

```
let iAmInput = document.getElementById("itsInputTime");
iAmInput.value;
```

any

Property 'value' does not exist on type 'HTMLElement'.

```
(<HTMLInputElement>document.getElementById("itsInputTime")).value
```

(property) HTMLInputElement.value: string
Returns the value of the data at the cursor's current position.

```
let iAmInput = <HTMLInputElement>document.getElementById("itsInputTime");
iAmInput.value;
```

(property) HTMLInputElement.value: string
Returns the value of the data at the cursor's current position.

sending any to fn with typed param

```
var fnParam: any = 99999;

function iWantString(param: string): void {
        let res = param.split(",");
}

iWantString(fnParam);
```

catch param is of type any

```
function fnWithTry() {
    try {
        console.log("abcd");
    } catch (e) {
        console.log(e);
    }
}
```

(local var) e: any

https://github.com/Microsoft/TypeScript/issues/8677

in old code you might miss some vendor specific props on Error object

async event handlers

```
function someAsyncFunction(message: string): Promise<boolean> {
    let promise = new Promise<boolean>(function (resolve, reject) {
        let res = confirm(message);
        console.log(`someAsyncFunction : ${res}`);
        resolve(res);
    });

    return promise;
}

$("#someButton").click(async () => {
    try {
        await someAsyncFunction("abcd");
    } catch (e) {
        console.log((<Error>e).message);
    }
});
```

await call in “root” of .ts file

```
1  await someAsyncFunction("aaa");
2
3  (async () => {
4      await someAsyncFunction("aaa");
5  })();
```

can I await sync function?

yep, you can. function needs to be marked async to await inside. if sync fn is awaited, result will be turned to async and resolved immediately.

needed if we have variable what is sync or async function with same signature.

```
var f: (() => any) | (() => Promise<any>);
```

every .ts file for every web page must be module

without modules there will be name clashes or you would have to build each .ts for each page as separate build.

I would advice to use webpack or browserify to automate these builds.