Typescript - deeper dive

How does TypeScript do it?

Bartlomiej Brzezianski

github.com/barbrzez

TypeScript compiler

TypeScript can compile to

- ES3 (very old JavaScript specification)
- ES5 (classic JavaScript, supported by all browsers now)
- ES6 (modern JavaScript, supported by node.js and most - but not all - browsers)

When compiling, it produces JS files and (optionally) type definitions.

Block scopes / function scopes / hoisting

Pure ES5 and older JavaScript uses only two type of scopes - global and function scope.

Additionally, the JavaScript engines hoists variable declarations.

This can be a source of unexpected problems.

TypeScript deals with it and allows us not to worry about it.

This is slightly different with ES6 and let keyword.

JavaScript

```
function DoSomething(a) {
  var calculationValue = 5;
  if (a > 0) {
    var calculationValue = Math.sqrt(a);
    window.setTimeout(function() {
      console.log(calculationValue);
    }, 100);
  }
  calculationValue /= 2;
  return calculationValue;
}
```

Hoisted JavaScript

```
function DoSomething(a) {
  var calculationValue;
  calculationValue = 5;
  if (/* … */) {
    calculationValue = /* … */;
    /* … */
      console.log(calculationValue);
    /* … */
  }
  calculationValue /= 2;
  return calculationValue;
}
```

TypeScript saves the day

We don’t really have to worry about hoisting anymore.

We can use block scope without having to worry about side effects.

Private / protected / public fields and functions

Fields and functions can be private, protected or public (kind of…).

There is a well hidden option to create internal field.

What is a Namespace

A namespace in TypeScript is just an object that contains exported classes, functions and constants.

When you import a definition from a different namespace, you actually reference JavaScript object.

Inheritance, interfaces, as keyword, instanceof

TypeScript creates an __extends function that simulates object oriented inheritance in the prototypal inheritance world of JavaScript.

Interfaces, similar to the ‘as’ keyword are compile time things only.

Therefore you can’t really use the ‘instanceof’ with the interfaces.

More information

https://basarat.gitbooks.io/typescript/

https://www.typescriptlang.org/docs/home.html

Thank you :)

Bartlomiej Brzezianski

github.com/barbrzez