Thursday, March 30, 2017

Es6

Es6 and typeScript

Javascript is also known as “EcmaScript”, and “Es6” is the new version of it bringing more functionality to the table, is not widely supported even today so you have to transpire it to “Es5” by your transpolar of choice in this case TypeScript.

Es6

New features:

- Classes
- Arrow Functions
- Template Strings
- Inheritance
- Constants and Block Scoped Variables
- Spread and Rest operators
- Destructuring
- Modules

Lets start with the first one :

Classes

They are used to describe the blueprint of an object and transform prototypical inheritance model functioning more like a traditional class-based language.

Example:

```
class Car {
	constructor() {  // A constructor
	}
	drive() { // A method
	}
}
```

We can create objects this way by simple assigning them the class:

```
let ferrari = new Car();

ferrari.drive();
```

When using something like this we are using the word “this” in an implicit way, “this” is going to doing a reference to the object to the left of the dot, in this case it references the object “ferrari”

You can also change the caller context by indicating to which object “this” is going to be doing a reference by using Function.prototype.call or Function.prototype.bind or for the last Function.prototype.apply. More information about this one here.

Arrow Functions

Es6 offers some new syntax for dealing with “this”, and part of this syntax are the “arrow functions” that brings easier ways of coding functions.

The arrow functions can be used to define anonymous functions in a simpler way.

For example:

```
	array.forEach( function( element ) {
		console.log(element);
	})
```

this can be a lot simpler:

```
	array.forEach(( element ) => {
		console.log(element);
	})
```

And it also works with functions that return something! making something like this :

```
array.map( function (elementToReturn) {
	return elementToReturn + 1;
})
```

Look like this :

```
array.map( (elementToReturn) => elementToReturn + 1;)
```

But all of this cases have a difference you cant use “this”, “arguments”, “super”, “new.target” because arrow functions do not set a local copy of them. When using “this” inside an arrow function Javascript uses the “this” from the outer scope.

Example:

```
class car {
	constructor ( anArray ) {
		this.types = Array.isArray(anArray) ? anArray : [];
	}

	showType () {
		this.types.forEach( (type, i) => {
			console.log(type + “is a type with index” + i + “inside an array of” +
this.types.length + “elements”);
			// “this” in this case works ! referring to the object “types” outside the
scope.
		} )
	}
}
```

Template Strings

instead of doing something like this:

```
var a = “hello”

var b = “jhon”

console.log(a + “my name is ” + b);
```

i can do something like this

```
console.log(`${a} my name is ${b}`);
```

so its much easier to code and to read;

Inheritance

Es6 provides a syntactic sugarer attempting to alleviate the issues with using prototypical inheritance present in Es5.

Here enters “Subclassing”, we can use “extends” keyword to tell an object to inherit another.

Example:

```
class Vehicle {
	constructor (speed) {
		this.speed = speed;
	}

	ride() {
		console.log(“im riding this vehicle”);
	}
}

//Subclass

class Boat extends Vehicle {
	constructor (speed) {
		super(speed);
	}

	rideAcrossSea() {
		console.log(“im riding through water”);
	}
}
```

Delegation

Its another way of inheritance where one objects asks another to run a function for him.

```
class Vehicle {
	constructor (speed) {
		this.speed = speed;
	}

	ride() {
		console.log(“im riding this vehicle”);
	}
}
```

```
//Subclass

class Boat{
	constructor (vehicle) {
		this.vehicle = vehicle;
	}

	ride() {
		this.vehicle.ride();
	}

	rideAcrossSea() {
		console.log(“im riding through water”);
	}
}

const vehicle = new Vehicle(24);
const boat = new Boat(vehicle);

boat.ride();
boat.rideAcrossSea();
```

Constants and Block Scoped Variables

Es6 introduces the concept of block scoping, here instead of “var” we use “let” and “const”, when we define a variable that we know its going to change we can use “let”, if its a variable that its not going to change we can use “const”, this way we can use variables that accept Read/Write operations on them with “let” and Read only variables with “const” BUT theres a catch with the last one, on some scenarios this can change:

Const object literal:

When we assign our constant variable an object and we change one of its properties, this is going to be accepted.

```
const variable = {
	property1: “hello”
}

variable.property1 = “hi” // it works
```

Const Reference To An Object:

When we reference an object with our variable, we can change one of its properties, and its going to be accepted.

```
let variable = {
	property1: “hello”
}

const variable2 = variable;
```

variable2.property1 = “hi” // it works;

Spread and Rest operators

Spread allows in-place expansion of an expression for the following cases:

1- Array.
2- Function Call.
3- Multiple variable deconstructing.

While Rest on the contrary collects an indefinite number of comma separated expression into an array.

Spread:

const addToArray = (a,b,c) => a + b + c;
let args = [3,5,6];

addToArray(…args); // this is the same as doing `addToArray(args[0], args[1], args[2])` or `addToArray.apply(null,args);

You can also use it outside functions for example:

let array1 = [‘3’ , ’4’ , ’5’];

let array2 = [‘1’ , ’2’ , …arra1 , ’5’ , ’7’]; // { ‘1’ , ’2’ , ’3’ , ’4’ , ’5’ , ’6’ , ’7’ }

Rest :

They are used as indefinite number of arguments passed to a function.

```
function add ( a, b ) {
        return a + b;
}

function addWithRest ( …numbers ) {
        return numbers[0] + numbers[1];
}
```

Destructuring

Is the way of quickly extract data out of an object or array without too much code.

for example:

let foo = [‘1’ , ’2’ , ’3’];

let one = foo[0];
let two = foo[1];
let three = foo[2];

into ->

```
let foo = [ ‘1’ , ’2’ , ’3’ ];
let [ one, two , three ] = foo;

console.log(one); // ‘1’
```

You can use it with objects on the same way which is really useful.

Modules

Its a file that allows code and data to be isolated, in other languages its called a package or library.

All the code can only be accessed inside the module because it has file scope, to share it outside the module it needs to be exported, this can be done by using the “export” keyword.

example:

```
//inside general.js file
export const a = “hello”
```

Web browsers doesn't have the concept of modules so we need to use a tool like “Webpack” to bring a javascript module that can be used by other javascript code.