Productividad con

Y

Jose Ignacio Paris Prieto
https://www.linkedin.com/in/jiparis
@jiparis

Capítulo 1

- *Javascript no es mi lenguaje favorito*
- *Typescript al rescate*
- *Herramientas*
- *Tipos*
- *Clases y herencia*
- *Interfaces*
- *Funciones*
- *Módulos*

JavaScript no es mi lenguaje favorito

Porque ...

... no está tipado
... sintaxis laxa → errores indetectables
... WAT Programming: http://bit.ly/watProg

```
[ ] + [ ] → ""
[ ] + { } → [Object Object]
{ } + [ ] → 0
{ } + { } → NaN
```

Test on Chrome console (F12)

JavaScript no es mi lenguaje favorito

Pero …

… es el lenguaje más universal jamás creado
… ejecuta increiblemente rápido en navegadores modernos

Principales pecados de JavaScript

- Variables globales
- `with` (eliminado en ES5)
- `eval()` (algunos lo llaman `evil()`)
- Type coercion (`boolean == integer`)
- Bloques sin scope `{}`
- `;` opcionales
- hoisting
- …

Principales errores al programar javascript

- ¿Cómo funciona `this`?
- ¿Cómo funciona la herencia por prototipo?
- Hoisting y scopes

JavaScript Hoisting

```
x = 5; // Assign 5 to x

elem = document.getElementById("demo");
elem.innerHTML = x;

var x; // Declare x !!!
```

JavaScript Scopes

```
var arr = [1,2,3];
var out = [];
for(var i = 0; i<arr.length;i++) {
   var item = arr[i];
   out.push(function(){ alert(item); });
}

out.forEach(function(func){ func(); });
```

¿Qué ocurre con este código?

JavaScript Scopes

```
var arr = [1,2,3];
var out = [];
for(var i = 0; i<arr.length;i++) {
    (function(value){
        var item = value;
        out.push(function(){ alert(item); });
    })(arr[i]);
}

out.forEach(function(func){ func(); });
```

Sólo las funciones crean scopes

JavaScript `this`

```
var o = {
  prop: 37,
  f: function() {
    return this.prop;
  }
};

console.log(o.f()); // logs 37
```

Event handlers:

```
var o = {
  prop: 37,
  f: function() {
    console.log(this.tagName);
  }
};

var elements = document.getElementsByTagName('*');

for(var i=0 ; i<elements.length ; i++){
  elements[i].addEventListener('click', o.f, false);
}
```

`this` toma el valor de quien llama a la función

Herencia por prototipado en JavaScript

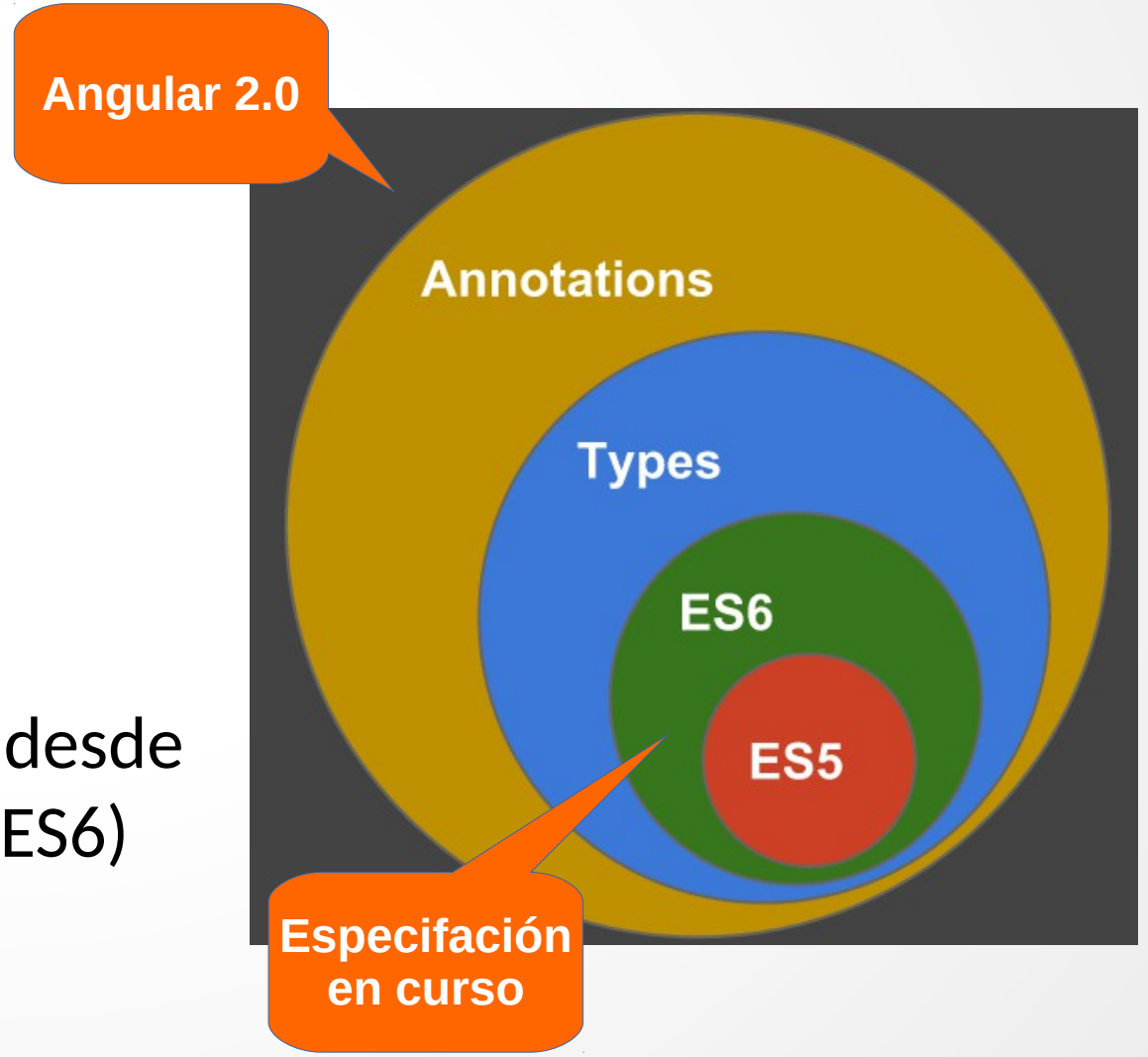
- En JavaScript no hay clases
- No hay `super` o `parent`
- No hay **`extends`** ni **`implement`**

```
base = function(){};

base.prototype.msg = "Hello"; // en objeto

obj1 = new base();
obj2 = new base();

obj1.msg === obj2.msg; // true
```

TypeScript al rescate

Javascript con:

- Tipos (estáticos, sin runtime)
- Clases y herencia
- Módulos
- Una nueva sintaxis más rica y familiar
- Totalmente compatible con ES5
- Hasta la fecha, lo más parecido a ES6 (desde la 1.5, aportará tipos y anotaciones a ES6)



Lo mejor de TypeScript

- Excelente integración con entornos de desarrollo:
  Visual Studio Express 2013, JetBrains WebStorm, Sublime Text ...
- Depuración: Chrome Dev Tools
- Productividad extrema (frente a JavaScript):
  Control de errores en tiempo de compilación
- Interoperabilidad con JavaScript (Es JavaScript)

Herramientas

Desarrollo

- VS 2013 for Web
- Chrome

Tool chain para **command line**:

- NodeJS
  npm install typescript -g

Por ahora no será necesario, pero sí más adelante

Comenzamos

#00

TypeScript es JavaScript

```
function greeting(name) {
    var el =
document.getElementById("content");
    el.innerHTML = "Hello " + name + "!";


greeting("world");
```

```
function greeting(name) {
    var el =
document.getElementById("content");
    el.innerHTML = "Hello " + name + "!";
}

greeting("world");
//# sourceMappingURL=00.js.map
```

Tipos

#01

```
function showArea(shape: string, width: number, height: number) {
    var area = width * height;
    var message = "I'm a " + shape + " with an area of " + area + " cm squared.";


    var el = document.getElementById("content");
    el.innerHTML = message;
}

showArea("rectangle", 12, 44);
```

Clases

#02

```
class Shape{
    ...

    constructor(shape: string, width: number, height:
number) {
        this.mShape = shape;
        this.mWidth = width;
        this.mHeight = height;
    }

    public showArea() {
        var area = this.mWidth * this.mHeight;
        ...
    }
}
```

Constructor opcional

Métodos
(public o private)

Herencia

#03

```
class Shape03{
    constructor(private type: string) {
    }

    public showMessage(area: number) {
    ...
    }
}

class Square extends Shape03 {
    public constructor(private width: number, private height:
number) {
        super("square");
    }

    public showArea() {
        var area = this.width * this.height;
        this.showMessage(area);
    }
}
```

```
var oneSquare = new Square(12,
44);
var otherSquare = new Square(2,
8);
oneSquare.showArea();
otherSquare.showArea();
```

Interfaces

#04

```
interface IShape {
    calculateArea(): number;
    showArea();
}
```

```
class Square extends Shape {
    public constructor(private width: number, private
number) {
        super("square");
    }

    calculateArea():number {
        return this.width * this.height;
    }
}
```

```
var newSq: IShape = new Square(5, 6);
newSq.showArea();

var newSq: IShape = new Triangle(5, 6);
newTr.showArea()
```

Una nota sobre las 2 formas de definir funciones

#05

1

```
prototypeMethod(): void {
    console.log("this method goes to the prototype");
}
```

```
MyClass.prototype.prototypeMethod = function () {
    console.log("this method goes to the prototype");
};
```

2

```
propertyMethod = () : void => {
    console.log("Hello from property");
}
```

Arrow syntax

```
this.propertyMethod = function () {
    console.log("Hello from property");
};
```

Una nota sobre las 2 formas de definir funciones

#05

1

```
prototypeMethod(): void {
    console.log("this method goes to the prototype");
}
```

Mayor eficiencia (el prototipo se comparte)
`this` se refiere a quien llama al método (normalmente el propio objeto)

2

```
propertyMethod = () : void => {
    console.log("Hello from property");
}
```

Usar siempre para callbacks de eventos

Menor eficiente (una “instancia” del método por cada objeto)
`this` se refiere **siempre** al objeto (su valor se copia previamente)

Modulos

#06

```
module Leccion06 {
    export interface IShape {
        calculateArea(): number;

        showArea();
    }
    …
}
/// <reference path="Shape.ts" />

module Leccion06 {
    export class Rectangle extends Leccion06.BaseShape {
        public constructor(private width: number, private height:
number) {
            super("square");
        }

        calculateArea(): number {
            return this.width * this.height;
        }
    }
}
```

Usar export para declarar elementos públicos

Referencias necesarias para command line, aunque VS no sequejará

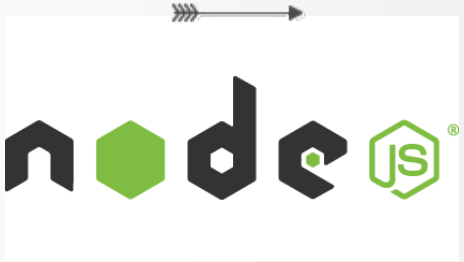
Namespaces

Módulos externos

TypeScript se integra con gestores de módulos:

- AMD (RequireJS)
- CommonJS (NodeJS)

```
export module Leccion06 {
    ...
}
```



Pero para proyectos web grandes es contraproducente

- **Cientos de .js son cientos de llamadas HTTP**
- **Solución: cachear y minimizar en local (con `grunt requirejs`)**

- **Pero TypeScript ya lo hace !!!**

```
tsc --out myapp.js app.ts
```

Librerías JavaScript externas

TypeScript interopera con JavaScript, pero ...
... ¿qué hay con el tipado? ¿se pierde lo bueno de TypeScript?

TypeScript permite declarar "librerías de entorno"

Se referencian como cualquier otro módulo

```
/// <reference path="../definitions/angular.d.ts" />
```

https://github.com/borisyankov/DefinitelyTyped/
http://definitelytyped.org/

Librerías JavaScript externas

TypeScript interopera con JavaScript, pero ...
... ¿qué hay con el tipado? ¿se pierde lo bueno de TypeScript?

TypeScript permite declarar "librerías de entorno"

Se referencian como cualquier otro módulo

```
/// <reference path="../definitions/angular.d.ts" />
```

https://github.com/borisyankov/DefinitelyTyped/
http://definitelytyped.org/