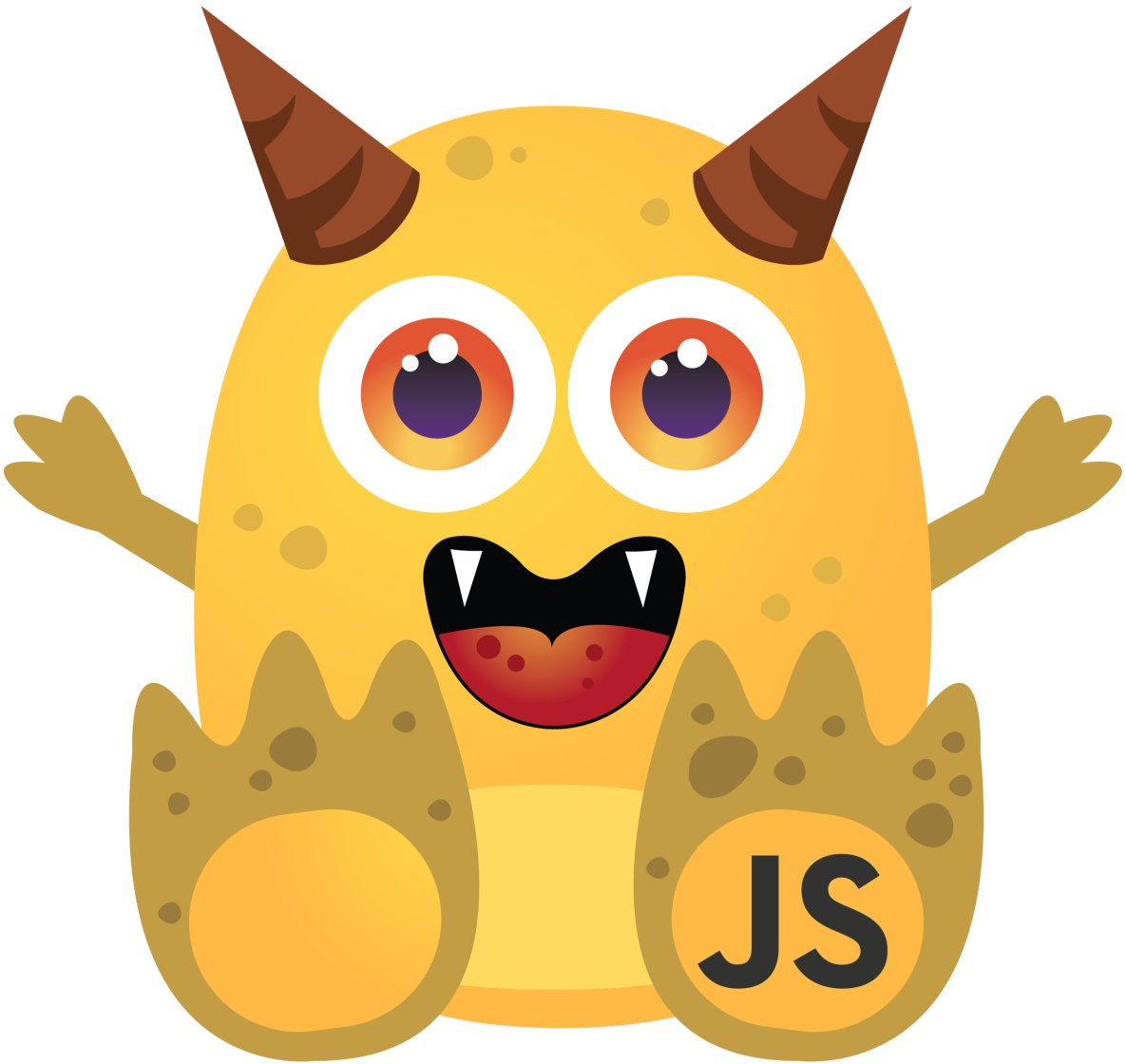
NA CO SI DÁT POZOR V JAVASCRIPTU?

Milan Lempera @milanlempera
Víťa Plšek @winsik

Angular.cz


JS

STRICT MODE

volitelně aktivovatelná omezenější varianta JS

```
'use strict';   // strict mode pro celý soubor

a = 11;         // ReferenceError: a is not defined


function sum(a, a, c) {   // SyntaxError:
  //    Strict mode function may not have duplicate parameter names

  return a + b + c;
}

var someNumber = 010;  // SyntaxError:
  //    Octal literals are not allowed in strict mode.
```

V čem se JS liší od ostatních jazyků?

FUNCTION

- funkce jsou objektovým typem
- každá funkce je instancí typu Function

function statement

```
function hello() {
    console.log('Hello!');
}

hello();
```

function expression

```
var hello2 = function() {
  console.log('Hello!');
};

hello2();
```

FUNCTIONS ARE FIRST-CLASS OBJECTS

```
var hello = function() {
  console.log('Hello!');
};

hello();    // 'Hello!'


var helloAlias = hello;
helloAlias();   // 'Hello!'

anotherFunction(hello);
```

```
var getFunction = function() {
  return function() {
    console.log('Hello function');
  };
};


var hello = getFunction();

hello();  // 'Hello function'
```

FUNCTION - NEEXISTUJE PŘETĚŽOVÁNÍ

```
function test(a, b) {
  ...
}

function test(a, b, c) {  // tato definice "přepíše" původní
  ...
}
```

FUNCTION - ARGUMENTS

```
function test(a, b) {
  console.log('a:', a);
  console.log('b:', b);
  console.log('arguments:', arguments);
}

test(1, 2);
//  a: 1
//  b: 2
//  arguments: [1, 2]
```

arguments je array-like objekt obsahující parametry předané funkci při volání

FUNCTION - PARAMETRY

```
test(1);
//  a: 1
//  b: undefined
//  arguments: [1]

test(1, 2, 3);
//  a: 1
//  b: 2
//  arguments: [1, 2, 3]
```

FUNCTION - DEFAULTNÍ HODNOTY

```
function test(a) {

  a = a || "defaultValue";


}
```

Boolean	true	false
String	libovolný neprázdný řetězec	""
Number	libovolné nenulové číslo	0, NaN
Object	libovolný objekt	null
Undefined	-	undefined

FUNCTION - DEFAULTNÍ HODNOTY

```
function test(a, b) {

  a = a || "defaultValue";

  b = typeof b !== 'undefined' ?  b : 1;

}
```

OBJEKTOVĚ ORIENTOVANÉ PROGRAMOVÁNÍ

JS MÁ OBJEKTY

ALE NEMÁ TŘÍDY

OBJECT LITERAL

```
var dog = {
  name: 'Teggi',
  speak: function() {
    return "Haf";
  }
};
```

- vytváří instanci objektu
- jde o anonymní objekt

KONSTRUKTOR

```
var Dog = function(name) {
  this.name = name;

  this.speak = function() {
    return "Haf";
  };
}

var dog = new Dog("Teggi");
```

- konstrukční funkce
- název vždy velkým písmenem (konvence)
- voláme s operátorem new

KONSTRUKTOR

```
var Dog = function(name) {
  this.name = name;
  this.speak = function() {
    return "Haf";
  };
}
```

```
var dog1 = new Dog("Teggi");
var dog2 = new Dog("Rek");
var dog3 = new Dog("Sára");
var dog4 = new Dog("Zára");
var dog5 = new Dog("Dag");
...
```

každá instance bude mít vlastní metodu speak

```
dog1.speak === dog2.speak;   // false
dog1.speak === dog3.speak;   // false
dog1.speak === dog4.speak;   // false
dog1.speak === dog5.speak;   // false
```

PROTOTYPE

- vlastnost každé funkce
- obsahuje vlastnosti dostupné každé instanci odvozené z této funkce
- vlastnosti prototypu jsou společné všem instancím

KONSTRUKTOR + PROTOTYPE

```
var Dog = function(name) {
  this.name = name;
};

Dog.prototype.speak = function() {
  return "Haf";
};
```

```
var dog1 = new Dog("Teggi");
var dog2 = new Dog("Rek");
var dog3 = new Dog("Sára");

dog1.speak();     // "Haf"

dog1.speak === dog2.speak;   // true
dog1.speak === dog3.speak;   // true
```

DĚDIČNOST

- v JS používáme prototypovou dědičnost
- zřetězení prototypů od potomka k rodiči/rodičům

PROTOTYPE CHAIN

```
var Dog = function(name) {
  this.name = name;
};

Dog.prototype.speak = function() {
  return "Haf";
};

var dog1 = new Dog("Rek");
dog1 instanceof Dog   // true
dog1 instanceof Object // true

dog1            // {name: "Rek"}

var p1 = Object.getPrototypeOf(dog1)    // Dog.prototype
var p2 = Object.getPrototypeOf(p2)      // Object.prototype
var p3 = Object.getPrototypeOf(p3)      // null
```

DĚDIČNOST

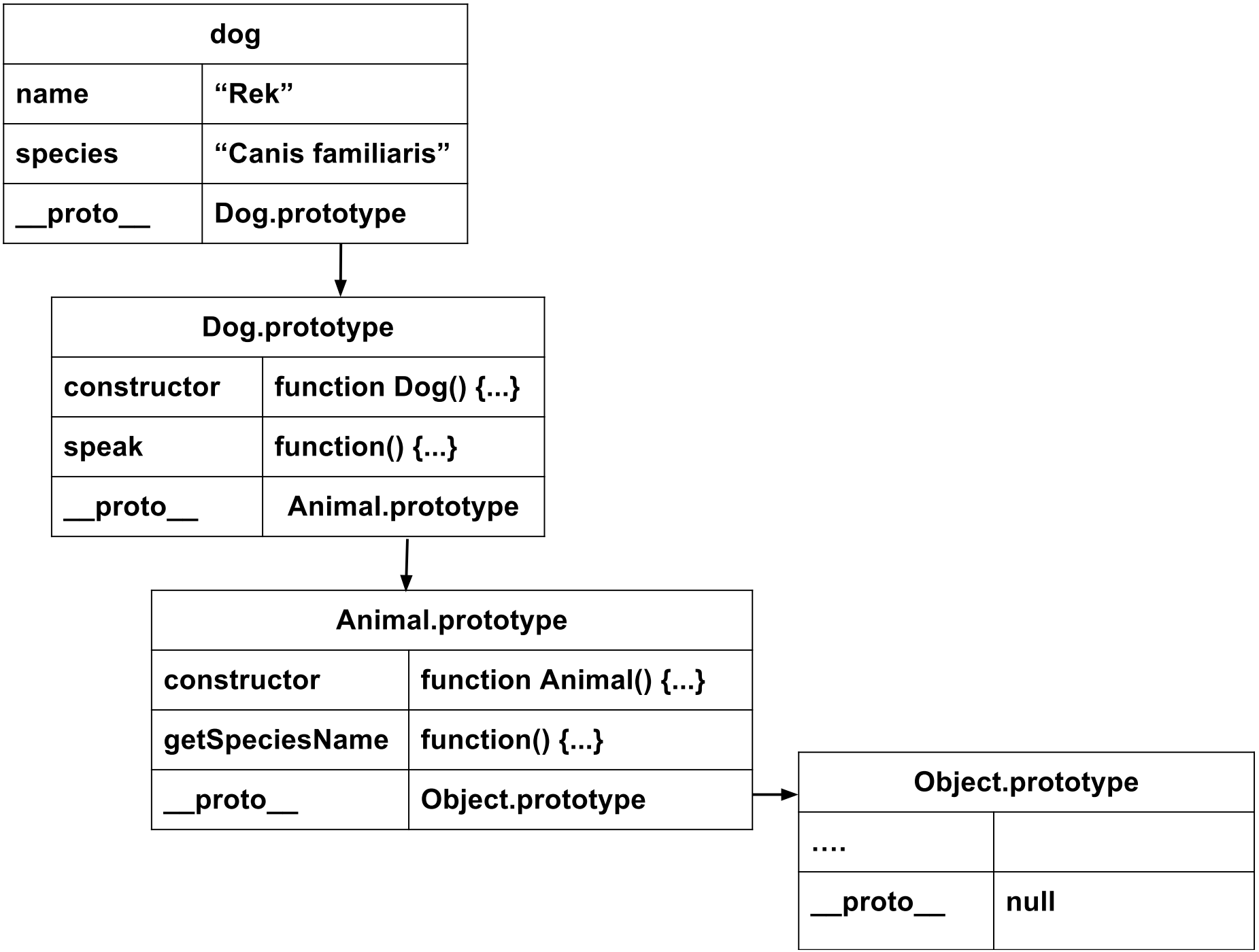
```
var Animal = function(species) {
  this.species = species;
}

Animal.prototype.getSpeciesName = function() {
  return this.species;
}
```

```
var Dog = function(name) {
  Animal.call(this, "Canis familiaris");
  this.name = name;
};

Dog.prototype = Object.create(Animal.prototype);
Dog.prototype.constructor = Dog;

Dog.prototype.speak = function() {
  return "Haf";
};
```


dog
name
“Rek”
species
“Canis familiaris”
__proto__
Dog.prototype
Dog.prototype
constructor
function Dog() {...}
speak
function() {...}
__proto__
Animal.prototype
Animal.prototype
constructor
function Animal() {...}
getSpeciesName
function() {...}
__proto__
Object.prototype
Object.prototype
....
__proto__
null

MODIFIKÁTORY PŘÍSTUPU

JS NEMÁ

- v JS je vše public
- vše můžete přepsat

  i vlasnosti nativních objektů

OBJECT + CLOSURE

```
var dogFactory = function(dogName) {
  var name = dogName;

  function speak() {
    return "Haf";
  };

  return {
    getName: function() {
      return name;
    },
    speak: speak
  }
}

var dog = dogFactory("Teggi");
```

- není to constructor, ale factory
- volá se bez new, nefunguje instanceOf
- využívá closure

CLOSURE

```
var genFrom10 = generator(10);
genFrom10()   // 10
genFrom10()   // 11
genFrom10()   // 12
```

```
var generator = function(initialNumber) {
  var current = initialNumber;

  return function () {
    return current++;
  };

};
```

OBJECT + CLOSURE

```
var dogFactory = function(dogName) {
  var name = dogName;

  function speak() {
    return "Haf";
  };

  return {
    getName: function() {
      return name;
    },
    speak: speak
  }
}

var dog = dogFactory("Teggi");
```

Nepoužívejte closure jako náhradu private.

THIS V JS

THIS A OBJEKT

```
var object = {
  prop: 42,
  method: function() {
    console.log(this.prop);
  }
};

console.log(object.method());    // 42
```

THIS A OBJEKT

```
function independent() {
  console.log(this.prop);
}

var object = {
  prop: 42,
};

object.method = independent;

console.log(object.method());    // 42
```

This je objekt na kterém je funkce volaná

- Vazba na this není danná definicí, ale pouze voláním.
- Proto nezáleží na tom, jestli funkce vznikla v rámci objektu, nebo jak k němu byla přiřazena.

THIS V ASYNCHRONNÍCH OPERACÍCH

```
var object = {
  prop: 42,
  method: function() {
    console.log(this.prop);
  }
};

setTimeout(object.method, 1000);

// po jedné vteřině vypíše "undefined"
```

```
setTimeout(function() {
  object.method();
}, 1000);
```

```
setTimeout(object.method.bind(object), 1000);

//  fce.bind(thisArg[, arg1[, arg2[, ...]]])  : Function
```

PROBLÉMY S ČÍSLY

OPERÁTORY A DYNAMICKÉ TYPY

Operátor sčítání / operátor zřetězení

```
"Angular" + ".cz" = "Angular.cz"

35 + 7 = 42
```

Význam operátoru závisí na kontextu

```
 1   + "1" = "11"
"1" +  1   = "11"
```

Pokud je jeden z operandů string, jedná se o zřetězení

NUMBER - SPECIÁLNÍ HODNOTY

```
"Barcamp " + 2015 = "Barcamp 2015"
```

```
"Barcamp " - 2015 = NaN
"Barcamp " * 2015 = NaN
"Barcamp " / 2015 = NaN
```

NaN - Not a Number.

```
10 / 0 = Infinity
```

Dělení nulou nevyvolá žádnou chybu.

A TO JE VŠECHNO?

NENÍ, ALE TOTO BYLA NEJPROBLEMATIČTĚJŠÍ MÍSTA

JS není takové strašidlo, když mu rozumíte!

ANGULAR.CZ/BARCAMP-HK