Javascript - Object

phuong.vu@htklabs.com

Agenda

- Javascript Concepts
- Javascript Types
- Objects - Properties, Methods, Definitions
- Function Object

Javascript Concepts

- Object-Oriented
  - JavaScript is an object-oriented language.
  - Functions are objects, too. They can have properties and methods.
  - Two main types of objects:
    - Native: further be categorized as built-in (for example, Array, Date) or user-defined (var o = {};)
    - Host: Defined by the host environment, for example, window and all the DOM objects
- No classes: There are no classes in JavaScript
- Prototype: is an object (not a class or anything special) and every function has a prototype property . Prototypes are used for inheritance.

Types

- **Primitive types (5):** Number, String, Boolean, Null, Undefined.
- **Object types**: Object, Array, RegExp, Function, Date, Math

Objects

- In JavaScript, objects are king. If you understand objects, you understand JavaScript.
- In JavaScript, almost "everything" is an object.
- Only five primitive types are not objects: *number, string, boolean, null,* and *undefined*, and the first three have corresponding object representation in the form of primitive wrappers.
- ***Number, string,*** and ***boolean*** primitive values are easily converted to objects.
- Dates, Maths, Regular expressions, Arrays, Functions are always objects.
- Objects are objects.
- Objects are Variables Containing Variables

Objects are Variables Containing Variables

- JavaScript variables can contain single values:
  - Ex: `var person = "John Doe";`
- Objects are variables too. But objects can contain many values. A JavaScript object is an unordered collection of variables called named values.
  - Ex: `var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};`

Object Properties

- The named values, in JavaScript objects, are called properties.
- Properties can usually be changed, added, and deleted, but some are read only.
- The syntax for accessing the property of an object are:
  - *objectName.property // person.age*
  - *objectName["property"] // person["age"]*
  - *objectName[expression] // x = "age"; person[x]*

Object Methods

- Methods are actions that can be performed on objects.
- Object properties can be both primitive values, other objects, and functions.
- An object method is an object property containing a function definition.
- Create an object method with the following syntax:
  - *methodName : function() { code lines }*
  - *objectName.methodName()*

Object Prototypes

- Every JavaScript object has a prototype. The prototype is also an object.
- All JavaScript objects inherit their properties and methods from their prototype.
- The standard way to create an object prototype is to use an object constructor function
    - Ex:
```
function Person(first, last, age, eyecolor) {
    this.firstName = first;
    this.lastName = last;
    this.age = age;
    this.eyeColor = eyecolor;
}
```

Object Definitions - 3 ways

- Define and create a single object, using an object literal.
  - Ex: `var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};`
- Define and create a single object, with the keyword new.
  - Ex: `var person = new Object();`
- Define an object constructor, and then create objects of the constructed type.
  - Ex:

```
function Person(first, last, age, eye) {
    this.firstName = first;
    this.lastName = last;
    this.age = age;
    this.eyeColor = eye;
    }
    var myFather = new Person("John", "Doe", 50, "blue");
```

Built-in JavaScript Constructors

- new Object(); // A new Object object
- new String(); // A new String object
- new Number(); // A new Number object
- new Boolean() // A new Boolean object
- new Array(); // A new Array object
- new RegExp(); // A new RegExp object
- new Function(); // A new Function object
- new Date(); // A new Date object

Built-in constructors (avoid)	Literals and primitives (prefer)
`var o = new Object();`	`var o = {};`
`var a = new Array();`	`var a = [];`

`var re = new RegExp(`
`  "[a-z]",`
`  "g"`
`);`
	`var re = /[a-z]/g;`

`var s = new String();`	`var s = "";`
`var n = new Number();`	`var n = 0;`
`var b = new Boolean();`	`var b = false;`

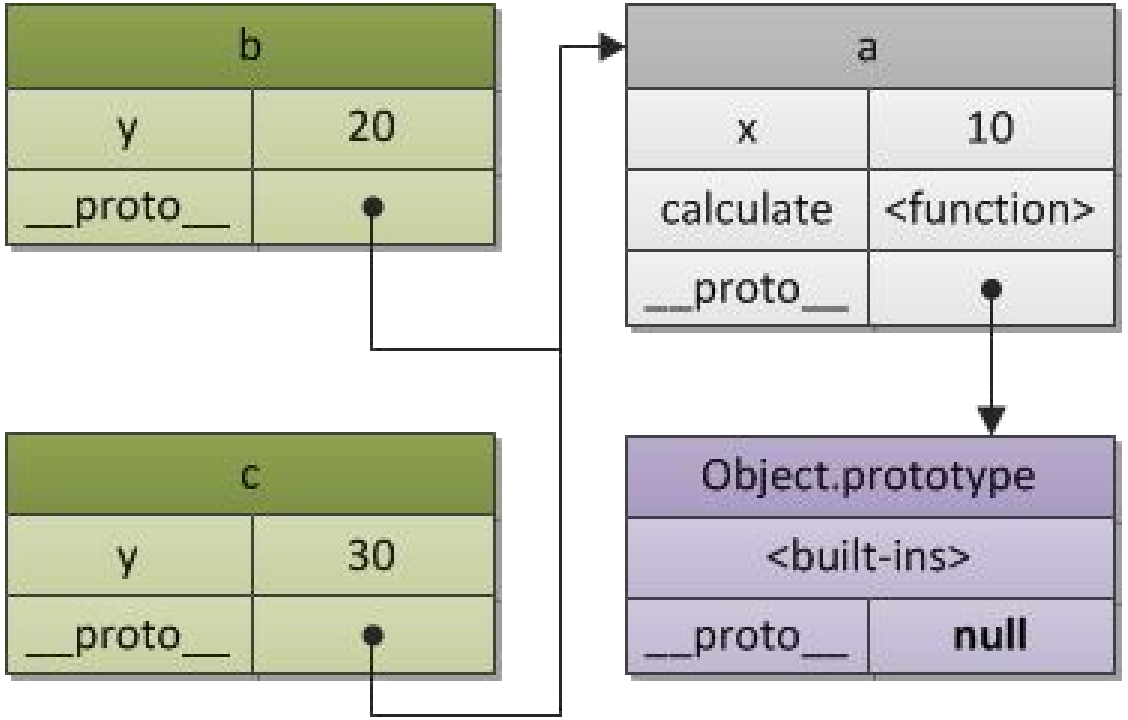
`throw new Error("uh-oh");`	`throw {`
	`  name: "Error",`
	`  message: "uh-oh"`
	`};`
	
	... or
	
	`throw Error("uh-oh");`

Object with prototype

```
var a = {
  x: 10,
  calculate: function (z) {
    return this.x + this.y + z;
  }
};

var b = {
  y: 20,
  __proto__: a
};

var c = {
  y: 30,
  __proto__: a
};
```



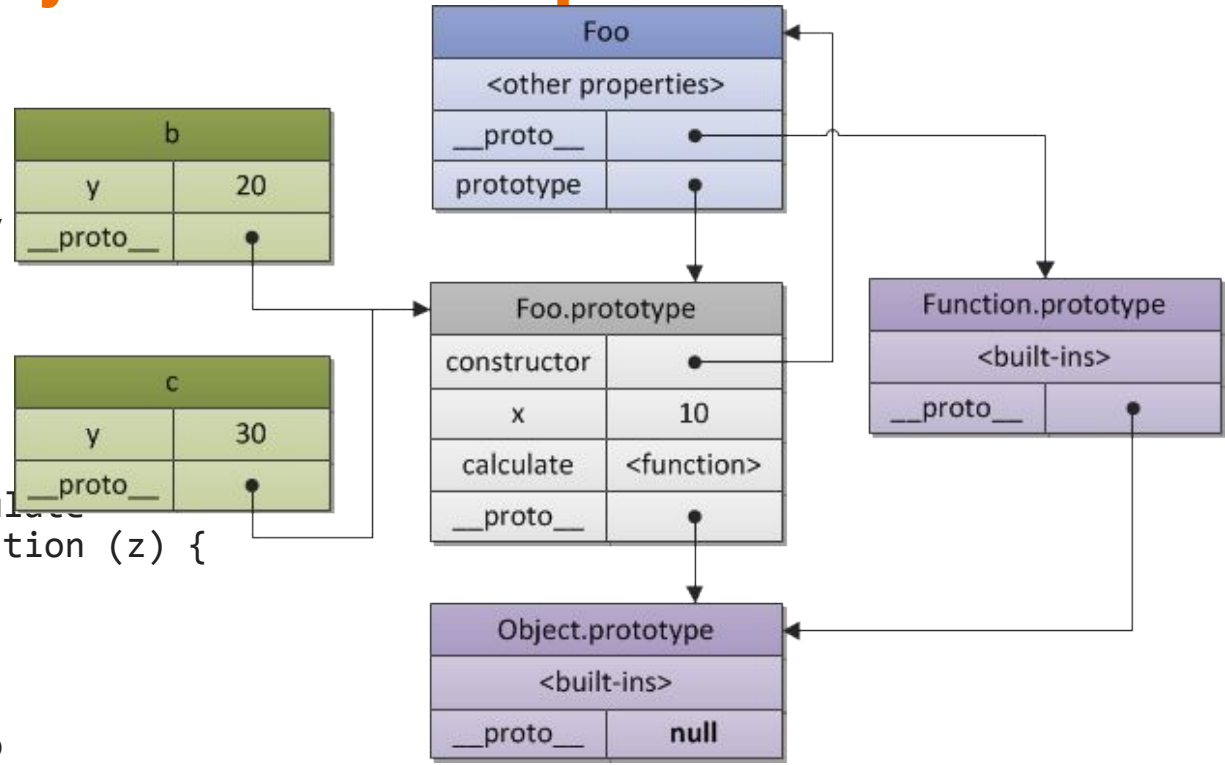
Constructor and Object relationship

```
// a constructor function
function Foo(y) {
  // creation own "y" property
  this.y = y;
}

// inherited property "x"
Foo.prototype.x = 10;

// and inherited method "calculate"
Foo.prototype.calculate = function (z) {
  return this.x + this.y + z;
};

// now create our "b" and "c"
// objects using "pattern" Foo
var b = new Foo(20);
var c = new Foo(30);
```



Function Objects

- Can be created dynamically at runtime, during the execution of the program.
- Can be assigned to variables, can have their references copied to other variables, can be augmented, and, except for a few special cases, can be deleted.
- Can be passed as arguments to other functions and can also be returned by other functions.
- Can have their own properties and methods.

Important features of functions in JavaScript

- Functions are first-class objects; they can be passed around as values and augmented with properties and methods.
- Functions provide local scope, which other curly braces do not. Also something to keep in mind is that declarations of local variables get hoisted to the top of the local scope.

Functions Declarations

- Defines a named function variable without requiring variable assignment.
- Occur as standalone constructs and cannot be nested within non-function blocks.
- Think of them as siblings of Variable Declarations. Just as Variable Declarations must start with “var”, Function Declarations must begin with “function”.
- Ex:

```
function bar() {
        return 3;
}
```

Function Expressions

- Defines a function as a part of a larger expression syntax (typically a variable assignment).
- Functions defined via Functions Expressions can be named or anonymous.
- Function Expressions must not start with “function”.
- Ex:

```
//anonymous function expression
var a = function() {return 3;}
//named function expression
var a = function bar() {return 3;}
//self invoking function expression
(function sayHello() {alert("hello!");})();
```

Callbacks Pattern

- Functions are objects, which means that they can be passed as arguments to other functions.
- When you pass the function (***B***) as a parameter to the other function (***A***) then at some point ***A*** is likely to execute (or call) ***B***. In this case ***B*** is called a callback function or simply a callback.
- Ex:

```
function writeCode(callback) {
        // do something...
        callback();
        // ...
}
function introduceBugs() {// ... make bugs }
writeCode(introduceBugs);
```

Returning Functions

- Functions are objects, so they can be used as return values.
- This means that a function doesn't need to return some sort of data value or array of data as a result of its execution.
- A function can return another more specialized function, or it can create another function on-demand, depending on some inputs.
- Ex:

```
var setup = function () {
        alert(1);
        return function () { alert(2)}
};
// using the setup function
var my = setup(); // alerts 1
my(); // alerts 2
```

Assignment

https://docs.google.com/document/d/1jj1IWvw9_Wfd9qAdLJIMiyXXwmdy3HPcaoMgfg7YgZc/edit

Q&A

Thanks for your listening!