JS: History and Philosophy	Object Creation	Constructor/Methods	Prototypes/Inheritance

Lecture 2: JavaScript - Objects and Prototypes CPEN400A - Building Modern Web Applications - Winter 2020-1

Karthik Pattabiraman

The University of British Columbia Department of Electrical and Computer Engineering Vancouver, Canada



▲□▶ ▲□▶ ▲□▶ ▲□▶ □ のQで



Thursday September 21, 2017

Object Creation

Constructor/Methods

Prototypes/Inheritance

イロト 不得 トイヨト イヨト

э

SQA

Javascript: History and Philosophy



1 Javascript: History and Philosophy

- 2 Object Creation in Javascript
- Object Constructor and Methods
- Prototypes and Inheritance
- 5 Type-Checking and Reflection



- Invented in 10 days by Brendan Eich at Nescape in May 1995 as part of the Navigator 2.0 browser
 - Based on Self, but dressed up to look like Java
 - Standardized by committee in 2000 as ECMAScript



Brendan Eich (Inventor of JavaScript):

JavaScript (JS) had to "look like Java" only less so, be Java's dumb kid brother or boy-hostage sidekick. Plus, I had to be done **in ten days** or something worse than JS would have happened

Object Creation

Constructor/Methods

Prototypes/Inheritance

Reflection 00000000000000000



Javascript: Philosophy

- Everything is an object
 - Includes functions, non-primitive types etc.
 - Even the class of an object is an object !
- Nothing has a type
 - Or its type is what you want it to be (duck typing)
 - No compile-time checking (unless in strict mode)
 - Runtime type errors can occur
- Programmers make mistakes anyways
 - If an exception is thrown, do not terminate program (artifact of browsers, rather than JS)
- Code is no different from data
 - So we can use 'eval' to convert data to code
- Function's can be called with fewer or more arguments than needed (variadic functions)

JS: History and Philosophy Object 000● 0000

▲ロ ▶ ▲周 ▶ ▲ 国 ▶ ▲ 国 ▶ ● の Q @



This lecture

- We'll learn about Objects and Classes the "old way" (without ES6)
- ES6 makes it much simpler to declare and use objects, but
- it's just syntactic sugar around the old way of doing things
- Not understanding objects from the ground up can result in nasty surprises
- Things will make a lot more sense if we go from the old way

Object Creation

Constructor/Methods

Prototypes/Inheritance

イロト 不得 トイヨト イヨト

э

Sac



Object Creation in Javascript

- Javascript: History and Philosophy
- Object Creation in Javascript
- Object Constructor and Methods
- Prototypes and Inheritance
- 5 Type-Checking and Reflection

Object Creation

Constructor/Methods

Prototypes/Inheritance

What is an Object in JS ?



- Container of properties, where each property has a name and a value, and is mutable
 - Property names can be any string, including the empty string
 - Property values can be anything except undefined
- What are not objects ?
 - Primitive types such as numbers, booleans, strings
 - null and undefined these are special types

What about classes ?

- There are no classes in JavaScript, as we understand them in languages such as Java
- "What ? How can we have objects without classes ?"
 - Objects use what are known as prototypes
 - An object can inherit the properties of another object using prototype linkage (more later)

Object Creation

Constructor/Methods

Prototypes/Inheritance

Reflection

▲□▶ ▲□▶ ▲□▶ ▲□▶ □ のQで



Example of Object Creation

```
1 // Initializing an empty object
2 var empty_object = {};
3 
4 // Object with two attributes
5 var name = {
6 firstName: "Karthik",
7 lastName: "Pattabiraman";
8 };
```

NOTE

You don't need a quote around firstName and lastName as they're valid JavaScript identifiers



- What if you write name["middleName"]?
 - Returns undefined. Later use of this value will result in an "TypeError" exception being thrown

▲□▶ ▲□▶ ▲□▶ ▲□▶ □ のQで

▲□▶ ▲□▶ ▲□▶ ▲□▶ □ のQで

```
1 name["firstName"] = "Different firstName";
2 name.lastName = "Different lastName";
```

- What happens if the property is not present ?
 - It'll get added to the object with the value
- In short, objects behave like hash tables in JS

Object Creation

Constructor/Methods

Prototypes/Inheritance

▲ロ ▶ ▲周 ▶ ▲ 国 ▶ ▲ 国 ▶ ● の Q @

Objects are passed by REFERENCE !



- In JavaScript, objects are passed by REFRENCE
 - No copies are ever made unless explicitly asked
 - i.e., JSON.parse(JSON.stringify(obj))
 - Changes made in one instance are instantly visible in all instances as it is by reference

Object Creation

Constructor/Methods

Prototypes/Inheritance

イロト 不得 トイヨト イヨト

э

SQA

Object Constructor and Methods



- Javascript: History and Philosophy
- Object Creation in Javascript
- 3 Object Constructor and Methods
- Prototypes and Inheritance
- 5 Type-Checking and Reflection

Object Creation

Constructor/Methods

Prototypes/Inheritance

Reflection 00000000000000

How to create an object ?



- Define the object type by writing a "Constructor function"
 - By convention, use a capital letter as first letter
 - $\bullet~$ Use "this" within function to initialize properties
- Call constructor function with the new operator and pass it the values to initialize
 - Forgetting the 'new' can have unexpected effects
- 'new' operator to create an object of instance 'Object', which is a global, unique JavaScript object

Object Creation using New

```
1 var Person = function(firstName, lastName, gender)
        {
2         this.firstName= firstName;
3         this.lastName = lastName;
4         this.gender = gender;
5    }
6    var p = new Person("John", "Smith", "Male");
```

0000	000000	00000	000000000000000	00000000000000000
JS: History and Philosophy 0000	Object Creation	Constructor/Methods	Prototypes/Inheritance	Reflection 0000000000000



▲□▶ ▲□▶ ▲□▶ ▲□▶ □ のQで

- It's a reference to the current object, and is valid only inside the object
- Need to explicitly use this to reference the object's fields and methods
 - Forgetting this means you'll create new local vars
 - Can be stored in ordinary local variables
 - Cannot be modified from within the object

Prototypes/Inheritance

Reflection 00000000000000

▲□▶ ▲□▶ ▲□▶ ▲□▶ □ のQで



Constructors

- Using the new operator as we've seen
- this is set to the new object that was created
 - Automatically returned unless the constructor chooses to return another object (non-primitive)
- Bad things can happen if you forget the 'new' before the call to the constructor (Later)

JS: History and Philosophy	Object Creation	Constructor/Methods 0000●0	Prototypes/Inheritance	Reflection 00000000000000000
Object Method	ds			16

- Functions that are associated with an object
- Like any other field of the object and invoked as object.methodName()
 - Example: Polygon.draw(10, 100);
 - this is automatically defined inside the method
 - Must be explicitly added to the object

```
1 this.dist = function(point) {
2
3 return Math.sqrt( (this.x - point.x)
4 * (this.x - point.x)
5 + (this.y - point.y)
6 * (this.y - point.y));
7 }
```

NOTE

this is bound to the object on which it is invoked

ヘロマ ヘビマ ヘヨマ

Sac

3

JS: History and Philosophy Object Creation Constructor/Methods 00000 000000 000000 Prototypes/Inheritance

Reflection 000000000000000

▲ロ ▶ ▲周 ▶ ▲ 国 ▶ ▲ 国 ▶ ● の Q @





- Simply say object.methodName(parameters)
- Example: p1.dist(p2);
- this is bound to the object on which it is called. In the example, this = p1. This binding occurs at invocation time (late binding).

Object Creation

Constructor/Methods

Prototypes/Inheritance

イロト 不良 とうせん ほうし

э

Sac

Reflection 000000000000000



Prototypes and Inheritance

- Javascript: History and Philosophy
- 2 Object Creation in Javascript
- Object Constructor and Methods
- Prototypes and Inheritance
- 5 Type-Checking and Reflection

▲ロ ▶ ▲周 ▶ ▲ 国 ▶ ▲ 国 ▶ ● の Q @



Object Prototype

- Every object has a field called Prototype
 - Prototype is a pointer to the object the object is created from (i.e., the class object)
 - Changing the **prototype object** instantly changes all instances of the object
- The default prototype value for a given object is Object
 - Can be changed when using new or Object.create to construct the object



 In the previous example, what is the prototype value of a "Person" object ?

▲□▶ ▲□▶ ▲□▶ ▲□▶ □ のQで

```
1 var p = new Person("John", "Smith", "Male");
2 console.log( Object.getPrototypeOf(p) );
```

• What will happen if we do the following instead

1 console.log(Object.getPrototypeOf(Person));

JS: History and Philosophy Object Creation Constructor/Methods Prototypes/Inheritance Reflection



▲ロ ▶ ▲周 ▶ ▲ 国 ▶ ▲ 国 ▶ ● の Q @

Prototype Field

- Prototypes of objects created through {} (object literal syntax) is
 - Object.prototype
- Prototype of objects created using new Object
 - Object.prototype
- Prototype of objects created using new and constructors functions (e.g., Person)
 - Prototype field set according to the constructor function (if object) (e.g., Person)
 - Object.prototype (otherwise)



- Initializes a new native object
- Sets the object's "prototype" field to the constructor function's prototype field
 - In Chrome (V8 engine), the prototype of an object instance o is accessible through the hidden property o.___proto___.
 - Direct usage should be avoided! Use instead Object.getPrototypeOf(o)
 - If it's not an Object, sets it to Object.prototype
 - i.e., Object.create(null)
- Calls the constructor function, with the object as this
 - Any fields initialized by the function are added to this
 - Returns the object created **if and only if** the constructor function returns a primitive type (i.e., number, boolean, etc.). Ideally, the constructor function shouldn't return anything!

Object Creation

Constructor/Methods

Prototypes/Inheritance

Reflection 000000000000000

▲□▶ ▲□▶ ▲□▶ ▲□▶ □ のQで





- An object's prototype object is just another object (typically). So it can be modified too.
- We can add properties to prototype objects the property becomes instantly visible in all instances of that prototype (even if they were created before the property was added)
 - Reflects in all descendant objects as well (later)

Prototype Modification: Example



▲ロ ▶ ▲周 ▶ ▲ 国 ▶ ▲ 国 ▶ ● の Q @

Object Creation

Constructor/Methods

Prototypes/Inheritance

Reflection 000000000000000

▲□▶ ▲□▶ ▲□▶ ▲□▶ □ のQで



Delegation with Prototypes

- When you lookup an Object's property, and the property is not defined in the Object,
 - It checks if the Object's prototype is a valid object
 - If so, it does the lookup on the prototype object
 - If it finds the property, it returns it
 - Otherwise, it recursively repeats the above process till it encounters Object.prototype
 - If it doesn't find the property even after all this, it returns undefined

Prototypes/Inheritance



Prototype Inheritance

- Due to Delegation, Prototypes can be used for (simulating) inheritance in JavaScript
 - Set the prototype field of the child object to that of the parent object
 - Any access to child object's properties will first check the child object (so it can over-ride them)
 - If it can't find the property in the child object, it looks up the parent object specified in prototype
 - This process carries on recursively till the top of the prototype chain is reached (Object.prototype)



```
1 var Employee = function(firstName, lastName, Gender, title)
        {
            Person.call( this, firstName, lastName, Gender );
            this.title = title;
        }
        Employee.prototype = new Person();
            /* Why should you create a new person object ? */
        Employee.prototype.constructor = Employee;
10
11 var emp = new Employee("ABC", "XYZ", "Male", "Manager");
```

▲ロ ▶ ▲周 ▶ ▲ 国 ▶ ▲ 国 ▶ ● の Q @



- Creates a new object with the specified prototype object and properties
- proto parameter must be null or an object
 - Throws TypeError otherwise

Object.create Argument

- Can specify initialization parameters directly in Object.create as an (optional) 2nd argument
- var $e = Object.create(Person, { Title: {value: "Manager" }})$
 - We can specify other elements such as enumerable, configurable etc. (more later)



▲□▶ ▲□▶ ▲□▶ ▲□▶ □ のQで

```
var Person = \{
 1
2
3
       firstName: "John";
       lastName: "Smith";
4
       gender: "Male";
5
       print : function() {
6
7
8
9
          console.log( "Person : " + this.firstName
                    + this.lastName + this.gender;
    };
10
    var e = Object.create( Person );
11
    e.title = "Manager";
```

JS: History and Philosophy Object Creation Constructor/Methods Prototypes/Inheritance

Reflection 0000000000000000

▲□▶ ▲□▶ ▲□▶ ▲□▶ □ のQで



Design Tips

- Object.create might be cleaner in some situations, rather than using new and .prototype (no need for artificial objects)
- With new, you need to remember to use this and also NOT return an object in the constructor
 - Otherwise, bad things can happen
- Object.create allows you to create objects without running their constructor functions
 - Need to run your constructor manually if you want
 - i.e., Person.call(p2, "Bob")

Object Creation

Constructor/Methods

Prototypes/Inheritance



Class Activity

 Construct a class hierarchy with the following properties using both pseudo-class inheritance (through constructors) and prototypal inheritance (thro' Object.create). Add an area method and a toString prototype function to all the objects.

 $\mathsf{Point}\ \{\ \mathsf{x},\ \mathsf{y}\ \} \Rightarrow \mathsf{Circle}\ \{\ \mathsf{x},\ \mathsf{y}\ \mathsf{,r}\ \} \Rightarrow \mathsf{Ellipse}\ \{\ \mathsf{x},\ \mathsf{y},\ \mathsf{r},\ \mathsf{r2}\ \}$

Object Creation

Constructor/Methods

Prototypes/Inheritance

イロト 不得 トイヨト イヨト

э

SQA



Type-Checking and Reflection

- Javascript: History and Philosophy
- Object Creation in Javascript
- Object Constructor and Methods
- Prototypes and Inheritance
- 5 Type-Checking and Reflection

Object Creation

Constructor/Methods

Prototypes/Inheritance

▲□▶ ▲□▶ ▲□▶ ▲□▶ □ のQで



Reflection and Type-Checking

- In JS, you can query an object for its type, prototype, and properties at runtime
 - To get the Prototype: getPrototypeOf()
 - To get the type of: typeof
 - To check if it's of certain instance: instanceof
 - To check if it has a certain property: in
 - To check if it has a property, and the property was not inherited through the prototype chain: hasOwnProperty()

Object Creation

Constructor/Methods

Prototypes/Inheritance

▲ロ ▶ ▲周 ▶ ▲ 国 ▶ ▲ 国 ▶ ● の Q @





• Can be used for both primitive types and objects

```
1 typeof( Person.firstName ) => String
2 typeof( Person.lastName ) => String
3 typeof( Person.age ) => Number
4 typeof(Person.constructor) => function (prototype)
5 typeof(Person.toString) => function (from Object)
6 typeof(Person.middleName) => undefined
```



• Checks if an object has in its prototype chain the prototype property of the constructor

```
object instanceof constructor => Boolean
1
2
3
   // Example:
4
   var p = new Person( /* ... */ );
5
6
   var e = new Employee( /* ... */ );
7
   p instanceof Person; // True
8
9
   p instanceof Employee; // False
   e instanceof Person; // True
10
   e instanceof Employee; // True
11
   p instanceof Object; // True
12
   e instanceof Object; // True
```

◆□▶ ◆□▶ ◆臣▶ ◆臣▶ ○臣 - のへで



- Gets an object's prototype (From the prototype field) Object.getPrototypeOf(Obj)
 - Equivalent of 'super' in languages like Java
- Notice the differences between invoking getPrototypeOf on an object constructed using the "associative array" syntax vs through a constructor!

▲□▶ ▲□▶ ▲□▶ ▲□▶ □ のQで

```
1 var proto = {};
2 var obj = Object.create(proto);
3 Object.getPrototypeOf(obj); // proto
4 Object.getPrototypeOf(proto); // Object
```

▲ロ ▶ ▲周 ▶ ▲ 国 ▶ ▲ 国 ▶ ● の Q @



in operator

- Tests if an object o has property p
 - Checks both object and its prototype chain

```
1 var p = new Person( /* ... */ );
2 var e = new Employee( /* ... */ );
3 
4 "firstName" in p; // True
5 "lastName" in e; // True
6 "Title" in p; // False
```

Prototypes/Inheritance

Reflection 0000000000000

▲ロ ▶ ▲周 ▶ ▲ 国 ▶ ▲ 国 ▶ ● の Q @



hasOwnProperty

- Only checks the object's properties itself
 - Does not follow the prototype chain
 - Useful to know if an object has overridden a property or introduced a new one

```
1 var p = new Employee( /* ... */ );
2 p.hasOwnProperty("Title") // True
3 p.hasOwnProperty("FirstName") // False
```



• Go over the fields of an object and perform some action(s) on them (e.g., print them)

▲□▶ ▲□▶ ▲□▶ ▲□▶ □ のQで

• Can use hasOwnProperty as a filter if needed

```
1 var name;
2 for (name in obj) {
3 if (typeof(obj[name]) != "function") {
4 document.writeln(name + " : " + obj[name]);
5 }
6 }
```



Removing an Object's Property



▲□▶ ▲□▶ ▲□▶ ▲□▶ □ のQで

• To remove a property from an object if it has one (not removed from its prototype), use:

delete object.property-name

• Properties inherited from the prototype cannot be deleted unless the object had overriden them.

```
var e = new Employee( /* ... */ );
2
  delete e. Title; // Title is removed from e
```

Object Creation

Constructor/Methods

Prototypes/Inheritance

Reflection 000000000●0000



Object Property Types

- Properties of an object can be configured to have the following attributes (or not):
 - Enumerable: Show up during enumeration(for.. in)
 - Configurable: Can be removed using delete, and the attributes can be changed after creation
 - Writeable: Can be modified after creation
- By default, all properties of an object are enumerable, configurable and writeable



• Can be done during Object creation with Object.create

```
1 var e = Object.create( Person,
2 { Title: {value: "Manager",
3 enumerable: true,
4 configurable: true,
5 writeable: false
6 } );
```

• Can be done after creation using Object.defineProperty

▲□▶ ▲□▶ ▲□▶ ▲□▶ □ のQで

```
1 Object.defineProperty( Employee, "Title",
2 {value: "Manager",
3 enumerable: true,
4 configurable: true,
5 writable: false});
```

Prototypes/Inheritance

▲ロ ▶ ▲周 ▶ ▲ 国 ▶ ▲ 国 ▶ ● の Q @



Design Guidelines

- Use for. . . in loops to iterate over object's properties to make the code extensible
 - Avoid hardcoding property names if possible
 - Use instanceof rather than getPrototypeOf
- Try to fix the attributes of a property at object creation time. With very few exceptions, there is no need to change a property's attribute.

Object Creation

Constructor/Methods

Prototypes/Inheritance

Reflection ○○○○○○○○○○○○●○

▲ロ ▶ ▲周 ▶ ▲ 国 ▶ ▲ 国 ▶ ● の Q @



Class Activity

- Write a function to iterate over the properties of a given object, and identify those properties that it inherited from its prototype AND overrode it with its own values
 - Do not consider functions

Object Creation

Constructor/Methods

Prototypes/Inheritance

Reflection 0000000000000000●

▲ロ ▶ ▲周 ▶ ▲ 国 ▶ ▲ 国 ▶ ● の Q @



Table of Contents

- 1 Javascript: History and Philosophy
- Object Creation in Javascript
- 3 Object Constructor and Methods
- Prototypes and Inheritance
- 5 Type-Checking and Reflection