Lecture 6: Prototypes and Reflection

Building Modern Web Applications
Vancouver Summer Program 2018 (Package E)

Karthik Pattabiraman, Julien Gascon-Samson

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

Electrical and Computer Engineering

Monday, July 30, 2018

Prototypes and Inheritance

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

Object Prototype: Example

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

```
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) );
```

Prototype Field

- Prototypes of objects created through {} 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)

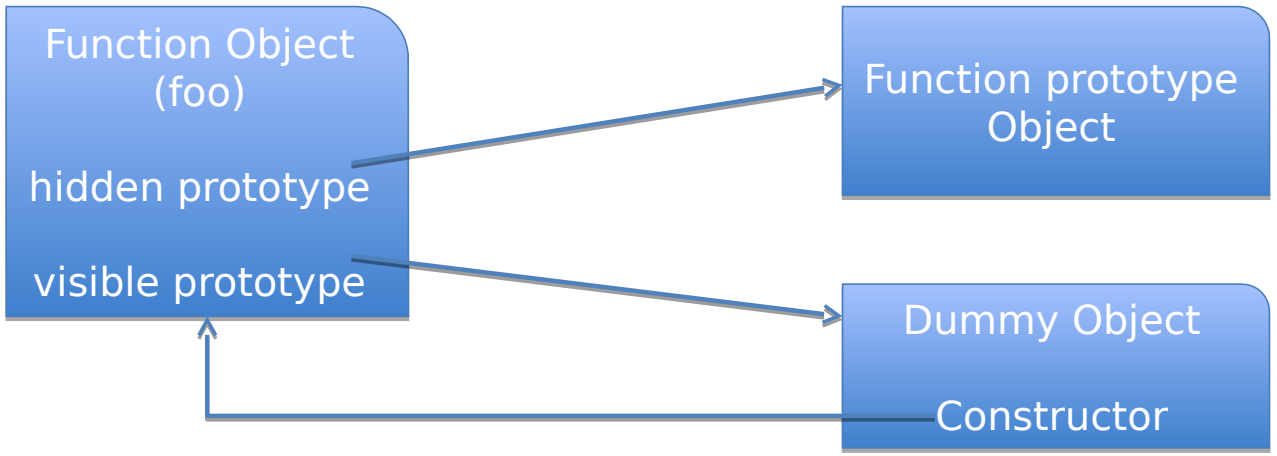
What 'new' really does?

- 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!

Functions are Objects too !

- Every function is an instance of a Function object, which is itself derived from Object
- A function object has two prototype fields:
  - A hidden prototype field to Function.prototype, which in turn links to Object.prototype
  - A visible prototype field (Function.prototype) which points to an Object whose constructor function points to the function itself !

What's really going on ?



- Why is it done in this convoluted way ?

Reason: Constructors

- In JavaScript, Functions can be used as constructors for Object creation (new operator)
  - However, JS engine does not know ahead of time which functions are constructors and which aren't
  - For the constructor functions, the (visible) prototype is copied to the new object's prototype
  - New object's prototype's constructor is thus set to the constructor function that created the object

Example


```
1  function Point( x, y) {
2        this.x = x; this.y = y;
3
4  };
5
6
7  var p1 = new Point(2,3);
8  var p2 = new Point(5,7);
9
10 console.log(Object.getPrototypeOf(p1) == Object.
       getPrototypeOf(p2));
11 console.log(Object.getPrototypeOf(p1).constructor);
```

Adding Functions to Prototype

- Functions can also be added to the Prototype object of an object
  - These will be applied to all instances of the object
  - Can be overridden by individual objects if needed

```
1  Point.prototype.toString = function( ) {
2          return "(" + this.x + " , " + this.y + ")";
3  }
```

Prototype Functions: Example


```
1  var p1 = new Person("John", "Smith", "Male");
2
3  Person.prototype.print = function() {
4     console.log( "Person: " + this.firstName
5           + this.lastName + this.gender + "\n");
6  }
7
8  var p2 = new Person("Linda", "James", "Female");
9  p1.print();
10 p2.print();
```

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

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)

Prototype Inheritance: Example


```
1  var Employee = function(firstName, lastName, Gender, title)
       {
2      Person.call( this, firstName, lastName, Gender );
3      this.title = title;
4  }
5
6  Employee.prototype = new Person();
7      /* Why should you create a new person object ? */
8
9  Employee.prototype.constructor = Employee;
10
11 var emp = new Employee("ABC", "XYZ", "Male", "Manager");
```

Object.create(proto)

- Creates a new object with the specified prototype object and properties
- Defined as a property of the Object – so available to all objects in JavaScript
- 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)

Prototype Inheritance with *Object.create*: Example


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

Class Activity 1 – Pseudo-Class Inheritance




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

Point { x, y } ⇒ Circle { x, y ,r } ⇒ Ellipse { x, y, r, r2 }

Class Activity 2 – Prototypal Inheritance

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

Point { x, y } ⇒ Circle { x, y ,r } ⇒ Ellipse { x, y, r, r2 }

Type-Checking and Reflection

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()

typeof

- 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
```

instanceof

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

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

getPrototypeOf

- 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!

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

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
```

Iterating over an Object's fields

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

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

```
1  delete object.property-name
```

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

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

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

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

Table of Contents