JS JavaScript Objects

Asst.Prof. Dr. Umaporn Supasitthimethee
ผศ.ดร.อุมาพร สุภสิทธิเมธี

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects

JavaScript: The Definitive Guide, Seventh Edition, by David Flanagan

JavaScript Objects

- ECMAScript **objects as hash tables**: nothing more than a grouping of name-value pairs where the value may be **data or a function.**
- **An object** is an **unordered collection of properties**
- An object is a **composite value**: it aggregates multiple values (primitive values or other objects) and allows you to store and retrieve those values by name.
- Property names are usually S*trings* or can also be *Symbols.*
- No object may have two properties with the same name.
- JavaScript objects are dynamic—properties can usually be added and deleted
- It is possible to create an instance of an "implicit" class without the need to actually create the class.

JavaScript Object Examples

```
//Simple Object
let student = {
name: 'Bob',
age: 32,
gender: 'male',
```

```
//Object Value is array
let profile = {
 id: 123,
 interests: [ 'music', 'skiing']
}
```

```
//Aggregated Object
 let book  = { isbn: 123456789,
                 title: "JavaScript",
                 author: {
                    firstname: "Umaporn",
                    lastname: "Sup"
                 }
                 };
```

Object Passing to functions by reference

- Objects are ***mutable*** and manipulated by reference rather than by value.

//04_Objects/script2.js

- `let point = { x:10, y: 20 };`
- `let newPoint = point;`
- `newPoint.x = 30;`
- `console.log (point) //{x:30, y:20);`

Understanding Objects

```
//create object without class
//The function distance does not care
//whether the arguments are an instance of the class Point

function distance(p1, p2) {
  console.log(typeof p1); //object
  console.log(typeof p2); //object
  // ** - The exponentiation assignment operator
  return Math.sqrt((p1.x - p2.x) ** 2 + (p1.y - p2.y) ** 2);
}

console.log(distance({ x: 1, y: 1 }, { x: 2, y: 2 })); //1.4142135623730951
```

Shorthand Object Methods

- When function is defined as a property of an object, we call that function *a method*
- Prior to ES6

```
let square ={
    area: function(){ return this.side * this.side;},
    side: 10
};
square.area() //=>100
```

- In ES6, the object literal syntax has been extended to allow a shortcut where the function keyword and the colon are omitted,

```
let square ={
    area (){ return this.side * this.side;},
    side: 10
};
square.area() //=>100
```

Understanding Object Creation

1. Simplest form with **object literals**, object literal is a comma-separated list of {name: value} pairs.

```
let point = {x:10, y: 20};
```

2. with the **new** operator. Objects created using the new keyword and a constructor invocation use the value of the prototype property of the constructor function as their prototype.

```
let person = new Object();
let a = new Array();
let p = new Point();
```

3. with the **Object.create()** function. Creates a new object, with specified prototypes.

```
let o = Object.create({x: 1, y: 2});
let p = Object.create(o);
console.log(p.x); //1
console.log(p.y); //2
```

Understanding Object Creation

//new Object()

```
let person = new Object();
person.name = "Adam";
person.age = 29;

person.greeting = function(){
       console.log("Hello" + person.name);
};

//{ name: 'Adam', age: 29, greeting: [Function] }
```

//Object Literal

```
let person={}; //no property
console.log(typeof person) //object
```

```
let person = {
    name: "Adam", //property key: value
    age: 29,
    greeting(){ //property function
       console.log("Hello" + person.name);
    }
};
```

```
//both can call greeting() function
person.greeting();
```

```
Hello, Adam
```

Object literals

- The easiest way to create an object is to include an object literal in your JavaScript code.
- In its simplest form, an object literal is a comma-separated list of colon-separated `name: value` pairs, enclosed within curly braces {}.
- A **property name** is a *JavaScript identifier* or a *string literal*
- A **property value** is any JavaScript expression; the value of the expression (it may be a primitive value or an object value) becomes the value of the property

Object literal Examples

```
let empty={};
let point = {x:0, y:0};
let p2={x: point.x, y: point.y+1};
let book ={
   "main title": "JavaScript", //These property names include spaces
   "sub-title": "The Definitive Guide", //and hyphens, so use string literals
   for: "all audiences",
   author:{
          firstname: "David",
          Surname: "Flanagan"
   }
};
```

Getting, Setting , Creating Object Properties

- To obtain the value of a property, use the dot (.) or square bracket([]) operators

```
let book = {  isbn: 123456789,
              title: "JavaScript",
              author:{
                firstname: "Umaporn",
                lastname: "Sup"
              }
};
```

```
object.property
object["property"]
```

- with the[] array notation, the name of the property is expressed as a string.
- Strings are JavaScript data types, so they can be manipulated and created while a program is running.

```
//getting object property
console.log (book.isbn);
console.log(book["title"]);
Console.log(book['author']['firstname’])
//setting object property
book.author.firstname = "Uma";
//create new object property
book["publishedYear"]=2000;
//or book.publishedYear=2000;
```

```
//console.log(book)
{
  isbn: 123456789,
  title: 'JavaScript',
  author: { firstname: 'Uma', lastname: 'Sup' },
  publishedYear: 2000
}
```

Create class and constructor functions (ES6)

```
//Recommendation to wrapping property and function within a single unit
//A class can be composed of the class's constructor method, instance methods, getters,
setters, and static class methods.

class Rectangle{
  constructor(width, height){ //invoke by new operator
    // Everything added to 'this' will exist on each individual instance
     this._width=width;
     this._height=height;
  }
  // Everything defined in the class body is defined on the class prototype object and
sharing between instances.
  area(){
     return this._width*this._height;
  }
}
```

```
let rec1=new Rectangle (2, 3);
console.log(rec1.area()); //6
```

//04_Objects/script1.js

```
class Rectangle {
  constructor(width, height) {
    this._width = width;
    this._height = height;
  }
    area() {
    return this._width * this._height;
  }
  get width() {
    return this._width;
  }
  set width(newWidth) {
    this._width = newWidth;
  }
  get height() {
    return this._height;
  }
  set height(newHeight) {
    this._height = newHeight;
  }
  toString(){
    return "width = "+this._width + ", height = "+this._height +
    ", area = " + this.area();
  }
}
```

```
//Get and Set
//ES6 classes brings a new syntax for getters and
setters on object properties.
//Get and set allows us to run code on the reading or
writing of a property.
//use _ convention to create a backing field to store
our name property. Without this every time get or set is
called it would cause a stack overflow.
```

```
let rec1=new Rectangle (2.5, 3.5);
console.log(rec1.width);  //2.5
console.log(rec1.height); //3.5
console.log(rec1.area()); //8.75
console.log(rec1.toString());
//width = 2.5, height = 3.5, area = 8.75
```

```
//create object with Function Constructor Pattern (ES5)
//It is also possible to define custom constructors, in the form of a function,
//that define properties and methods for your own type of object.
//By convention, constructor functions always begin with an uppercase letter,
//whereas non constructor functions begin with a lowercase letter.
```

//04_Objects/script4.js

```
function Person(name, age, job){
 this.name = name;
 this.age = age;
 this.job = job;
 this.sayName = function() {
  console.log(this.name);
 };
}

let person1 = new Person("Pot", 40, "Tester");
let person2 = new Person("Joe", 20, "Doctor");

person1.sayName(); //"Pot"
person2.sayName(); //"Joe
```

Any function that is called with the **new** operator acts as a constructor, whereas any function called without it acts just as you would expect a normal function call to act.

```
Person.prototype.greeting = function () {
  return `Hello, ${this.name}`;
};

console.log(person1.greeting());
```

Object Prototypes

- **Prototypes** are the mechanism by which JavaScript objects inherit features from one another.
- **JavaScript** is often described as a prototype-based language — to provide inheritance, objects can have a prototype object, which acts as a template object that it inherits methods and properties from.

Prototype Chaining

- ECMA-262 describes **prototype chaining** as the primary method of **inheritance i**n ECMAScript.
- The object created by **new** Object() or **object literal** inherit from `Object.prototype`
- Similarly, the object created by `new Array()` uses `Array.prototype` as its prototype, and the object created by `new Date()` uses `Date.prototype` as its prototype.
- `Date.prototype` inherits properties from `Object.prototype`, so a `Date` object created by `new Date()` inherits properties from both `Date.prototype` and `Object.prototype`.
- This linked series of prototype objects is known as a ***prototype chain.***

Prototype Chaining

- JavaScript objects have a set of **"own properties"** and they also inherit a set of properties from their prototype object.

```
let o = {};                  // o inherits object methods from Object.prototype
o.x = 1;                     // and it now has an own property x.
let p = Object.create(o);    // p inherits properties from o and Object.prototype
p.y = 2;                     // and has an own property y.
let q = Object.create(p);    // q inherits properties from p, o, and Object.prototype
q.z = 3;                     // and has an own property z.
let f = q.toString();        // toString is inherited from Object.prototype
q.x + q.y                    // => 3; x and y are inherited from o and p
```

Prototype Chaining

`prototypeObj.isPrototypeOf(object)`

object - the object whose prototype chain will be searched.

Return a Boolean indicating whether the calling object lies in the prototype chain of the specified object.

```
//define our own class and
//constructor functions
class Rectangle{
  constructor(width, height){
    this._width=width;
    this._height=height;
  }
  area(){
    return this._width*this._height;
  }
}
let rec1=new Rectangle (2, 3);
console.log(rec1.area()); //6
```

```
//create object with Object.create()
let square = Object.create(rec1);
square.perimeter = function() {
  return 4 * this.width;
}
console.log(square.width); //2
console.log(square.height);//3
console.log(square.area());//6
console.log(square.perimeter());//8
console.log(Object.prototype.isPrototypeOf(rec1));//true
console.log(Rectangle.prototype.isPrototypeOf(square));//true
console.log(Object.prototype.isPrototypeOf(square));//true
```

How to Compare Objects in JavaScript

1. Referential equality: ==, ===, Object.is()
2. Manual comparison of properties' values.
3. Shallow Equality check the properties' values for equality.

Referential equality

- Both are the same object means both object point to the same object instances.
- Three ways to compare objects:
  - The strict equality operator ===
  - The loose equality operator ==
  - Object.is() function

//04_Objects/script3.js

```
//Object Comparing
let student = { id: 1, name: "Joe" };
let newStudent = { id: 2, name: "Joe" };
let oldStudent = { id: 1, name: "Joe" };
let alumniStudent = student;
```

```
if (student == alumniStudent) { //true
  console.log("student equals to alumni student by ==");
  //student equals to alumni student by ==
}
if (student == newStudent) { //false
  console.log("student equals alumni student by ==");
}
if (student === alumniStudent) { //true
  console.log("student strictly equals to alumni student");
  //student strictly equals to alumni student.
}

if (student === newStudent) { //false
  console.log("student strictly equals to new student by ==");
}
```

```
//Object Comparing
let student = { id: 1, name: "Joe" };
let newStudent = { id: 2, name: "Joe" };
let oldStudent = { id: 1, name: "Joe" };
let alumniStudent = student;
```

//04_Objects/script3.js

```
/*The Object.is()method determines whether two values are the same value without
type conversion.  Both the same object means both object have same reference*/

  if (Object.is(student, alumniStudent)) { //true
    console.log("student equals to alumni student by Object.is()");
    //student equals to alumni student by Object.is()
  }
  if (Object.is(student, newStudent)) { //false
    console.log("student equals to new student by Object.is()");
  }

  if (Object.is(student, oldStudent)) { //false
    console.log("student equals to old student by Object.is()");
  }
```

Manual Comparison

- A manual comparison of properties' values.

//04_Objects/script3.js

```
//compare properties manually
function isStudentEqual(object1, object2) {
  return object1.id === object2.id;
}
```

```
console.log(isStudentEqual(student, oldStudent)); //true
console.log(isStudentEqual(student, alumniStudent));//true
```

Shallow Equality

`Object.keys(obj)`

//04_Objects/script3.js

obj - the object of which the enumerable's own properties are to be returned. Return an array of strings that represent all the enumerable properties of the given object.

```
//3. Shallow Equality
let book1 = {
  isbn: 123456789,
  title: "JavaScript",
};

let book2 = {
  isbn: 123456789,
  title: "JavaScript",
};
```

```
function shallowEquality(object1, object2){
  const keys1=Object.keys(object1);
  const keys2=Object.keys(object2);

  if(keys1.length !== keys2.length){
    return false;
  }
  for(let key of keys1){
    if(object1[key] !== object2[key] ){
      return false;
    }

  }
  return true;
}
```

```
console.log("shallow equality: " + shallowEquality(book1, book2)); //true
```

JSON – JavaScript Object Notation

- JavaScript Object Notation (JSON) is a standard text-based format for representing structured data based on JavaScript object syntax.
- It is commonly used for transmitting data in web applications (e.g., sending some data from the server to the client, so it can be displayed on a web page, or vice versa).
- Even though it closely resembles JavaScript object literal syntax, it can be used independently from JavaScript, and many programming environments feature the ability to read (parse) and generate JSON.
- A JSON string can be stored in its own file, which is basically just a text file with an extension of .json, and a MIME type of application/json.

JSON structure

- **JSON is a string** whose format very much resembles JavaScript object literal format.
- JSON **requires double quotes** to be used around strings and property names. **Single quotes are not valid** other than surrounding the entire JSON string.
- You can include the same basic data types inside JSON as you can in a standard JavaScript object — strings, numbers, arrays, booleans, and other object literals.
- JSON is purely a string with a specified data format — **it contains only properties, no methods**.
- We can also convert arrays to/from JSON.

```
{
  "squadName": "Super hero squad",
  "homeTown": "Metro City",
  "formed": 2016,
  "secretBase": "Super tower",
  "active": true,
  "members": [
    {
      "name": "Molecule Man",
      "age": 29,
      "secretIdentity": "Dan Jukes",
      "powers": [
        "Radiation resistance",
        "Turning tiny",
        "Radiation blast"
      ]
    },
    {
      "name": "Madame Uppercut",
      "age": 39,
      "secretIdentity": "Jane Wilson",
      "powers": [
        "Million tonne punch",
        "Damage resistance",
        "Superhuman reflexes"
      ]
    },
    {
      "name": "Eternal Flame",
      "age": 1000000,
      "secretIdentity": "Unknown",
      "powers": [
        "Immortality",
        "Heat Immunity",
        "Inferno",
        "Teleportation",
        "Interdimensional travel"
      ]
    }
  ]
}
```

```
[
  {
    "name": "Molecule Man",
    "age": 29,
    "secretIdentity": "Dan Jukes",
    "powers": [
      "Radiation resistance",
      "Turning tiny",
      "Radiation blast"
    ]
  },
  {
    "name": "Madame Uppercut",
    "age": 39,
    "secretIdentity": "Jane Wilson",
    "powers": [
      "Million tonne punch",
      "Damage resistance",
      "Superhuman reflexes"
    ]
  }
]
```

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON