JS: OOP & Design Patterns

Denis Mantsevich

Overview

- Объектно-ориентированный
- Нет классов
- Функция это объект
- Все свойства и методы объекта общедоступны
- Объекты можно изменять в любой момент времени
- Предпочтение отдавайте приему составления объектов, а не наследованию
- Прототип - это объект
- others...

OOP. Functions

- Создают локальную область видимости
- Могут выступать в качестве конструкторов объектов
- Могут создаваться динамически в процессе выполнения программы
- Могут присваиваться переменным, ссылки на них могут копироваться в другие переменные, могут быть расширены дополнительными свойствами и, за исключением некоторых особых случаев, могут быть удалены
- Могут передаваться как аргументы другим функциям и могут возвращаться другими функциями
- Могут иметь собственные свойства и методы

OOP. Constructor

```
// Constructor
var Car = function (brand, sn) {
    // Properties
    this.brand = brand;
    this.sn = sn;
};
// Method
Car.prototype.beep = function () {
    console.log(this.brand + ": Beep!");
    return this;
};

var toyota = new Car("Toyota", "L87WX459087T3");
toyota.beep(); // Toyota: Beep!

var lada = new Car("Lada", "A12OP593K87TZC");
// Override
lada.beep = function () {
    console.log("Does not work");
    return this;
};

lada.beep(); //Does not work
toyota.beep(); // Toyota: Beep!

var vwPolo = Car("VW", "B01MNRT4123Y5"); // Error. this === window.
```

OOP. Private properties

```
1  // Constructor
2  var Car = function (brand, sn) {
3      // Properties
4      this.brand = brand;
5      // Private props start with "_"
6      this._sn = sn;
7  }
8  // Methods
9  Car.prototype.beep = function () {
10     console.log(this.brand + ": Beep!");
11     return this;
12 }
13 Car.prototype.getSN = function () {
14     return this._sn;
15 }
16
17 var toyota = new Car("Toyota", "L87WX459087T3");
18 toyota.getSN(); // "L87WX459087T3"
19 console.log(toyota.sn);// undefined
20 console.log(toyota._sn);// "L87WX459087T3"
```

OOP. Private methods

```
// Constructor
var Car = (function () {
    // Clojure
    var getWheelsInfo = function () {
        return this.brand + " have " + this.wheels + " wheels";
    };
    var CarConstr = function (brand, sn, wheels) {
        // Properties
        this.wheels = wheels || 4;
        this.brand = brand;
        // Private props start with "_"
        this._sn = sn;
    }
    CarConstr.prototype.beep = function () {
        console.log(this.brand + ": Beep!");
        return this;
    }
    CarConstr.prototype.info = function () {
        return getWheelsInfo.call(this);
    }
    return CarConstr;
}());

var toyota = new Car("Toyota", "L87WX459087T3", 6);
toyota.beep(); // // Toyota: Beep!
toyota.info();// Toyota have 6 wheels
toyota.getWheelsInfo();// Error
getWheelsInfo();// Error
```

OOP. Static

```
// Constructor
var Car = function (brand, sn) {
    // Properties
    this.brand = brand;
    // Private props start with "_"
    this._sn = sn;
    Car.cars.push(this);
}
// Methods
Car.prototype.beep = function () {
    console.log(this.brand + ": Beep!");
    return this;
}
Car.cars = [];
Car.MAX_SPEED = 1000;
Car.GetAll = function () {
    return Car.cars;
}

var toyota = new Car("Toyota", "L87WX459087T3");
var vwPolo = new Car("VW", "RA7PV45394PTRT72");
console.log(Car.GetAll(), Car.MAX_SPEED); // Return [toyora, vwPolo], 1000;
```

OOP. Inheritance.

```
1  // Для прототипа дочернего класса устанавливаем
2  // ссылку на прототип родителя.
3  // Самый оптимальный вариант. При изменении
4  // прототипа дочернего конструктора, прототип
5  // родителя не изменяется
6  function inherit(C, P) {
7      var F = function () {};
8      F.prototype = P.prototype;
9      C.prototype = new F();
10 }
```

DP. Single Global Var

```
(function (root) {
    // Local scope. We can define variables.
    var App = {},
        settings = {
            name: "Hello App"
            //...
        }; // This is private variables

    // ... we init out app

    // ... we can add some public methods & properties
    App.config = function (key) {
        return settings[key];
    };
    App.VERSION = 1.0;

    root.MYAPP = App; // Export our App to global scope.

}(window));

// use our app
consoel.log(MYAPP.config("name") + MYAPP.VERSION); // Hello App 1.0
console.log(settings); // Error
```

DP. Namespaces

```
(function (root) {
    // Local scope. We can define variables.
    var App = {},
        modules = {}; // This is private variables
    // Import/Export modules mechanism
    App.Module = function (namespace, mdl) {
        var ns = modules, keys = namespace.split("."), i = 0, l = keys.length - 1;
        if (typeof mdl === "undefined") { // Define module
            for (; (i <= l) && ns; ns = ns[keys[i]], i++);
            return ns;
        } else { // Export module
            for (; i < l; ns = (ns[keys[i]] || (ns[keys[i]] = {})), i++);
            if (typeof ns[keys[i]] === "undefined") {
                ns[keys[i]] = mdl;
            } else {
                throw new Error("Module is exists.");
            }
        }
        return this;
    };
    root.MYAPP = App; // Export out App to global scope.
}(window));
// Examples
MYAPP.Module("core.events.dom", {}); // Define module
MYAPP.Module("core.events.internal", {status: "ready"}); // Define module
MYAPP.Module("core.string", String); // Define module
var status = MYAPP.Module("core.events.internal").status; // Import module. Result: "ready"
```

DP. Memorization

```
1  // func - функция которую нужно закешировать, hasher - хеш-функция
2  var memoize = function (func, hasher) {
3      var memo = {};
4      hasher || (hasher = function () {
5          return [].join.call(arguments, "&");
6      });
7      return function () {
8          var key = hasher.apply(this, arguments);
9          if (typeof memo[key] !== "undefined") {
10             return memo[key]
11         } else {
12             return (memo[key] = func.apply(this, arguments));
13         }
14     };
15 };
16 var fibonacci = memoize(function (n) {
17     return n < 2 ? n : fibonacci(n - 1) + fibonacci(n - 2);
18 });
19 fibonacci(3);//Calc
20 fibonacci(105);//Calc, but for 3 get from cache
21 fibonacci(3);//Get from cache
22 fibonacci(100);//Get from cache (Why?)
```

DP. Currying

```
1  // Если функцию приходиться часто вызывать с одинаковыми параметрами,
2  //то можно использовать механизм каррировая
3  var curring = function (func) {
4      var args = [].slice.call(arguments, 1);
5      return function () {
6          return func.apply(null, args.concat([].slice.call(arguments, 0)));
7      }
8  };
9   // Определяем параметры по-умолчанию (в данном случае 10)
10 var plus10 = curring(function (x, y) {
11     return x + y;
12 }, 10);
13  plus10(5); //15
14 plus10(-1); //9
15 // Дефолтые параметры 10,3
16 var plus10mult3 = curring(function (x, y, z) {
17     return (x + z) * y;
18 }, 10, 3);
19  plus10mult3(5); //45
20  plus10mult3(-1); //27
```

DP. Extend

```
1  // Копирование свойств из одного объекта в другой.
2  function extend(parent, child) {
3      var i;
4      child = child || {};
5      for (i in parent) {
6          if (parent.hasOwnProperty(i)) {
7              child[i] = parent[i];
8          }
9      }
10     return child;
11 }
```

DP. Bind

```
1  // Назначить неизменяемый контекст вызова.
2  if (typeof Function.prototype.bind === 'undefined') {
3      Function.prototype.bind = function (thisArg) {
4          var fn = this,
5              slice = Array.prototype.slice,
6              args = slice.call(arguments, 1);
7          return function () {
8              return fn.apply(thisArg, args.concat(slice.call(arguments)));
9          };
10     };
11 }
12 var t = {
13         name: "Obj1"
14     },
15     z = {
16         name: "Obj1"
17     },
18     say = (function () {
19         return this.name;
20     }).bind(t);
21 say(); // Obj1
22 say.call(z); // Obj1
```

DP. Singleton

```
// Просто создать объект с помощью литерала
var App = {};

// Для конструктора
function Logger() {
    // имеется ли экземпляр, созданный ранее?
    if (typeof Logger.instance === 'object') {
        return Logger.instance;
    }
    // ... создать новый экземпляр
    // сохранить его
    Logger.instance = this;
    // неявный возврат экземпляра:
    return this;
}
Logger.getInstance = function(){
    return new Logger();
};

var uni = new Logger();
var uni2 = new Logger();
var uni3 = Logger.getInstance();
console.log(uni === uni2 === uni3); // true
```

DP. Factory

```
// родительский конструктор
function CarMaker() {}
// методы дочерних конструкторов
CarMaker.prototype.drive = function () {
    return "Vroom, I have " + this.doors + " doors";
};
// статический фабричный метод
CarMaker.factory = function (type) {
    var constructorName = type,
        newcar;
    if (typeof CarMaker[constructorName] !== "function") {
        throw new Error("Constructor is not exists");
    }
    if (typeof CarMaker[constructorName].prototype.drive !== "function") {
        // Устанавливаем наследование
        CarMaker[constructorName].prototype = new CarMaker();
    }
    newcar = new CarMaker[constructorName]();
    return newcar;
};
// специализированные конструкторы
CarMaker.Compact = function () {
    this.doors = 4;
};
CarMaker.Convertible = function () {
    this.doors = 2;
};
var newCar = CarMaker.factory("Compact");
```

DP. Events. Sub/Pub. Observe

```
// Конструктор
var Events = {
    on: function (name, callback) {
        if (name && typeof callback == 'function') {
            this.events(name).push(callback);
        }
        return this;
    },
    off: function (name, callback) {
        var events = this.events(name), i = events.length;
        while (i--) {
            if (events[i] === callback) {
                events.splice(i, 1);
                return this;
            }
        }
        return this;
    },
    events: function (name) {
        this._events || (this._events = {});
        return name ? (this._events[name] || (this._events[name] = [])) : [];
    },
    trigger: function (name, params) {
        params = params || {};
        var events = this.events(name);
        for (var i = 0, l = events.length; i < l; i++) { events[i].call(this, params); }
        return this;
    }
};
```

DP. Others

- Прокси объект
- Шаблон делегирования
- Строитель
- Отложенная инициализация
- Фасад
- Стратегия
- Декоратор
- Итератор
- Медиатор
- others...

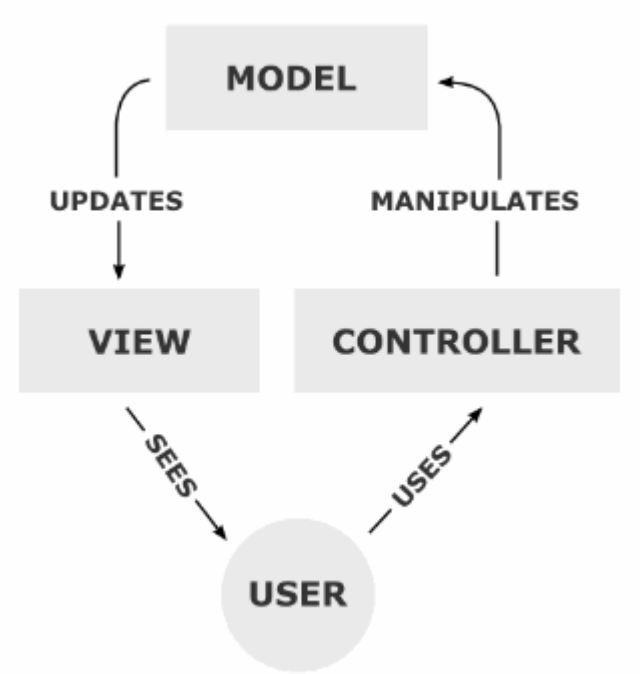
DP. Modules. AMD & Common JS

- **Asynchronous Module Definition** - Асинхронная загрузка. Используется на клиенте. Каждый модуль находиться в отдельном файле. Загрузить модуль можно с помощью ***require***. Объявить модуль и зависимости можно с помощью ***define.***
  See: RequireJS
- **Common JS** - Синхронная загрузка. Используется на серверной части. Каждый модуль находиться в отдельном файле. Загрузить модуль можно с помощью ***require***. Чтобы объявить модуль нужно экспортировать его возможности с помощью объекта ***exports***.
  See: CommonJS

DP. Model-View-Controller (MVC)

Концепция **MVC** позволяет разделить данные, представление и обработку действий пользователя на три отдельных компонента:

- ***Модель*** (англ. Model). Модель предоставляет знания: данные и методы работы с этими данными, реагирует на запросы, изменяя своё состояние. Не содержит информации, как эти знания можно визуализировать.
- ***Представление, вид*** (англ. View). Отвечает за отображение информации (визуализацию). Часто в качестве представления выступает форма (окно) с графическими элементами.
- ***Контроллер*** (англ. Controller). Обеспечивает связь между пользователем и системой: контролирует ввод данных пользователем и использует модель и представление для реализации необходимой реакции.



Simple Templating

```
var template = function (str){
        // Создаем новую функцию, которая пробегает по всем свойствам объекта и
        // подставляет в нужные места
        return new Function("obj",
                "var p=[],print=function(){p.push.apply(p,arguments);};" +
                    // Позволяет создать локальную область видимости,
                    // где свойства объектов станут переменными
                    "with(obj){p.push('" +
                    // Заменяем в шаблоне все "магические" теги
                    str
                        .replace(/[\r\t\n]/g, " ")
                        .split("<%").join("\t")
                        .replace(/((^|%>)[^\t]*)'/g, "$1\r")
                        .replace(/\t=(.*?)%>/g, "',$1,'")
                        .split("\t").join("');")
                        .split("%>").join("p.push('")
                        .split("\r").join("\\'")
                    + "');}return p.join('');");
};
```

Simple Templating

```
<script id="search-result" type="text/template">
    // Шаблон
        <h1>Результаты поиска</h1>
        <% if (items.length){ %>
            <div class="results-description">Всего найдено товаров: <%=totalCount %> </div>
            <ul class="catalog">
                <% for(var i, l = items.length; i<l; i++) { %>
                    <li class="item-<%= i %>">
                        <img src="<%= items[i].img %>" alt="<%= items[i].name %>"/>
                        <span class="name"><%= items[i].name %></span>
                    </li>
                <% } %>
            </ul>
        <% } else { %>
            <div class="empty">По вашему запросу ничего не найдено.</div>
        <% } %>
</script>
<script type="text/javascript">
    // Использование шаблона
    var searchResult = {
            totalCount: 100,
            items: [
                { img: "path/to/image_nokia.jpg", name: "Nokia Lumia 320" },
                { img: "path/to/image_sony.jpg", name: "iPhone 4" }
                //...
            ]
        },
        searchTemplate = template(document.getElementById("search-result").innerHTML);
    console.log(searchTemplate(searchResult)); // Получим результат. Можем вывести его в нужную колонку
</script>
```