Объекты в JS. Дескрипторы свойств и прототипное наследование.

Объект (Object) – это единственный непримитивный тип данных в JS, который представляет собой ассоциативный массив, то есть такую структуру данных, в которой информация хранится в виде пар «ключ-значение». Такая структура данных удобна, когда требуется хранить значения, ключи которых не могут быть приведены к непрерывной области значений индексов (для хранения в массиве), а также тогда, когда семантически удобнее использовать именованные ключи (например, при создании констант конфигурации статусов и т. д.). Ещё одно преимущество объектов заключается в том, что они лежат в основе синтаксиса JSON (JavaScript Object Notation), что упрощает их взаимную конвертацию (сериализацию и десериализацию) при работе с HTTP-запросами.

Если Вы ещё недостаточно хорошо разбираетесь с основами объектов JS, а именно: синтаксис объектов; инициализация и объявление объекта через литерал и конструктор new Object(); обращение к ключу и запись ключа объекта через точку (dot notation) и квадратные скобки (bracket notation); удаление ключа объекта и т. д., то прежде чем идти дальше, рекомендуется прочитать статью [1].

Объекты в JS передаются в качестве значения переменной по ссылке. Следовательно, каждое объявление или инициализация переменной типа Object связывает эту переменную с объектом с уникальным адресом в памяти, даже если свойства этих объектов полностью идентичны. Следующий скриншот показывает, что будет при сравнении таких переменных (взят из статьи [2]):

```JS
// Создаётся один объект
const book = { title: 'Дюна' }

// Создаётся другой объект
const anotherBook = { title: 'Дюна' }

console.log(book === anotherBook)
// false
```

Методом в JS называется функция, которая одновременно является свойством объекта:

```
var myObj = {
  myMethod: function (params) {
    // ...do something
  },
};
```

В JS существует ключевое слово «this», которое имеет смысл только внутри методов объекта, а также в функциях-конструкторах. «This» определяется только в момент вызова метода или функции (а не объявления!) и принимает в качестве своего значения объект, в котором метод был вызван, либо объект, который был создан с помощью функции-конструктора.

В примере ниже вызов метода person1.introduceSelf выведет в консоль «Hi! I'm Chris.», а вызов person2.introduceSelf выведет в консоль «Hi! I'm Deepti.» (пример взят из [1]):

```
const person1 = {
  name: "Chris",
  introduceSelf() {
    console.log(`Hi! I'm ${this.name}.`);
  },
};

const person2 = {
  name: "Deepti",
  introduceSelf() {
    console.log(`Hi! I'm ${this.name}.`);
  },
};
```

Аналогично в случае функции-конструктора:

```
function Person(name) {
  this.name = name;
  this.introduceSelf = function () {
    console.log(`Hi! I'm ${this.name}.`);
  };
}

const salva = new Person("Salva");
salva.introduceSelf();
// "Hi! I'm Salva."

const frankie = new Person("Frankie");
frankie.introduceSelf();
// "Hi! I'm Frankie."
```

В остальных случаях «this» равен undefined в строгом режиме исполнения кода, а в нестрогом – глобальному объекту окружения (host environment): window в случае браузера и global в случае Node.js.

Объекту можно задать свойства-геттеры (getters) и свойства-сеттеры (setters). Для краткости будем их называть просто геттеры и сеттеры соответственно. Геттер – это метод объекта, который возвращает значение некоторого свойства этого же объекта. Сеттер – это метод объекта, который устанавливает переданное в качестве аргумента значение некоторому свойству этого же объекта. Ниже представлен пример использования геттеров и сеттеров в объектах (взят в статье [3]):

```
const myObj = {
  a: 7,
  get b() {
    return this.a + 1;
  },
  set c(x) {
    this.a = x / 2;
  },
};
console.log(myObj.a); // 7
console.log(myObj.b); // 8, returned from the get b() method
myObj.c = 50; // Calls the set c(x) method
console.log(myObj.a); // 25
```

Геттеры и сеттеры полезны при форматировании и преобразовании свойств и инкапсуляции этой логики. Например, на их основе построена реактивность фреймворка Vue 2.

Согласно спецификации ECMAScript [4], каждое свойство объекта является либо data property, либо accessor property. Data property представляет собой пару ключ-значение (data и value) с атрибутами configurable, writable и enumerable. Accessor property определяется геттером или сеттером (или ими обоими одновременно), а также атрибутами configurable и enumerable.

Атрибут configurable (boolean) отвечает за возможность изменять атрибуты configurable, writable и enumerable. Также этот атрибут отвечает за возможность удалить свойство объекта. Если configurable === false, то свойство нельзя удалить или изменить его атрибуты (даже сам configurable).

Атрибут writable (boolean) отвечает за возможность менять значение свойства (value). Если writable === false, то менять нельзя.

Атрибут enumerable (boolean) отвечает за возможность свойства быть «видимым» для итераторов: цикла for…in и метода Object.keys(). Если enumerable === false, то свойство невидимо для итератора.

Изменить конфигурацию атрибутов можно с помощью метода Object.defineProperty() или Object.defineProperties() для нескольких

свойств одновременно. Более подробно об этом Вы можете прочитать в статье [5].

Ещё одно немаловажное свойство всех объектов – это прототипное наследование. Когда объект создается посредством конструктора (через оператор new), то этот объект наследует свойства другого объекта, который указан в свойстве [[Prototype]] конструктора (хоть конструктор и функция, но как мы знаем, это тоже объект и поэтому тоже имеет свойства). Это происходит в случае, если свойство [[Prototype]] указано в конструкторе.

Прототипное наследование заключается в следующем: при обращении к некоторому свойству (ключу) объекта, если оно отсутствует в «собственных» (own) свойствах объекта, продолжается поиск этого свойства в прототипе этого объекта. Если оно найдено в прототипе, то возвращается его значение как обычно. Иначе, поиск продолжается в прототипе прототипа и так далее рекурсивно по цепочке прототипов. Обратиться к прототипу объекта можно через свойство «__proto__».

Отметим, что в цикле for … in, итерирование происходит по всей цепочке прототипов. Если требуется итерирование по «собственным» свойствам объекта, что следует взять массив ключей Object.keys(), куда попадают только собственные свойства, либо осуществить проверку с помощью метода hasOwnProperty().

Если Вы не работали с прототипным наследованием объектов, рекомендуем прочесть статью [7]. Также настоятельно рекомендуем прочитать небольшой фрагмент текста из спецификации ECMAScript [6], где хорошо объясняется концепция работы объектов и наследования прототипов.

Напоследок, ещё одно важное замечание, касающееся изменения прототипов: никогда не изменяйте прототип объекта вручную. Если Вам это требуется, лучше создайте новый объект с нужным прототипом. Это связано со значительным ухудшением производительности движков JS в любом веб-браузере. Подробнее об этом в предупреждении на странице [8].

Вопросы и задания

1. Пусть имеется массив events следующего вида (кол-во элементов может быть больше):

```
const events = [
    {
        name: "OpenOffice Day",
        date: "2025-09-18T09:30:00.000+03:00",
    },
    {
        name: "Weekend",
        date: "2025-06-10T23:59:00.000+03:00",
    },
    {
        name: "New Year",
        date: "2025-12-31T23:59:00.000+03:00",
    },
    {
        name: "Halloween",
        date: "2025-10-31T00:00:00.000+03:00",
    },
    {
        name: "Something else then Halloween",
        date: "2025-10-31T00:00:00.000+03:00",
    },
];
```

Требуется написать функцию, которая принимает в качестве аргумента массив events и возвращает объект следующего вида

```
{
    "OpenOffice Day": "2025-09-18T09:30:00.000+03:00",
    "Weekend": "2025-06-10T23:59:00.000+03:00",
    "New Year": "2025-12-31T23:59:00.000+03:00",
    "Halloween": "2025-10-31T00:00:00.000+03:00",
    "Something else then Halloween":
        "2025-10-31T00:00:00.000+03:00",
};
```

где ключом является поле «name», а значением – поле «date». Сможете ли написать эту функцию в однострочном виде (one-liner)?

2. Продолжите выполнение предыдущего задания. Напишите функцию, которая принимает в качестве аргумента объект из предыдущего задания и возвращает объект, ключами которого являются месяца, а значениями – массивы с именами событий, соответствующих данному месяцу:

```
{
    "Sep": [ "OpenOffice Day" ],
    "Jun": [ "Weekend" ],
    "Dec": [ "New Year" ],
    "Oct": [ "Halloween", "Something else then Halloween" ]
}
```

   Для имён используйте следующие сокращения: "Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", Nov", "Dec". Подберите оптимальную структуру данных для хранения названий месяцев в отношении данной задачи.

3. Допустим Вам необходимо создать объект в файле конфигурации, который не должен быть изменён случайно в процессе разработки приложения другими разработчиками. Как бы Вы реализовали такой объект, если требуется, чтобы не было возможности добавлять новые свойства, изменять значения существующих свойств, конфигурировать и удалять имеющиеся свойства?

4. Предыдущее задание с теми же требованиями, кроме того, что у объекта должна быть возможность изменять значения существующих свойств.

5. Напишите функцию, которая принимает некоторый объект конечной вложенности и возвращает новый объект одинарной вложенности. Например, при передаче в данную функцию объекта вида

```
{
    prop_1: 10,
    prop_2: {
        prop_2_1: 20,
        props_2_2: 30,
    },
    prop_3: {
        prop_3: {
            prop_3: 13,
        },
        prop_2_2: 15
    },
};
```

должен быть возвращён объект вида

```
{
    prop_1: 10,
    prop_2_1: 20,
    prop_2_2: 15,
    prop_3: 13,
};
```

где повторяющиеся свойства перезаписываются на самое последнее вхождение.

6. *Выполните предыдущее задание, но уже с учётом цепочки прототипов переданного в функцию объекта. Цепочка прототипов предполагается конечной.

Ссылки

[1]. https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Scripting/Object_basics

[2]. https://doka.guide/js/object/

[3]. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_objects

[4]. https://tc39.es/ecma262/multipage/ecmascript-data-types-and-values.html#sec-object-type

[5]. https://doka.guide/js/descriptors/

[6]. https://tc39.es/ecma262/multipage/overview.html#sec-objects

[7]. https://learn.javascript.ru/prototype-inheritance

[8]. https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Object/setPrototypeOf