JavaScript Base

Занятие 5. Объекты

IT Education Academy

Содержание

- Понятие «объект»
- Способы создания объектов
- Свойства объектов
- Методы объектов
- Ключевое слово «this»

Понятие «объект»

Объекты в **JavaScript**, как и во многих других языках программирования, похожи на объекты реальной жизни. Концепцию объектов JavaScript легче понять, проводя паралелли с реально существующими в жизни объектами.

В JavaScript, объект это самостоятельная единица, имеющая свойства и определенный тип. Сравним, например, с чашкой. У чашки есть цвет, форма, вес, материал, из которого она сделана, и т.д. Точно так же, объекты JavaScript имеют свойства, которые определяют их характеристики.

Понятие «объект»

JavaScript спроектирован на основе простой парадигмы. В основе концепции лежат простые объекты. **Объект** — это набор свойств, и свойство состоит из имени и значения, ассоциированного и этим именем. Значением свойства может быть функция, которую можно назвать **методом** объекта. В дополнение к встроенным в браузер объектам, вы можете определить свои собственные объекты.

Объекты в JavaScript сочетают в себе два важных функционала:

- Первый — это ассоциативный массив: структура, пригодная для хранения любых данных.
- Второй — языковые возможности для объектно-ориентированного программирования.

Способы создания объектов

Пустой объект можно создать следующими способами:

```
1. obj = new Object();
2. obj = {}; // пустые фигурные скобки
```

Обычно все пользуются синтаксисом 2, т.к. он короче.

Объект может содержать в себе любые значения, которые называются свойствами объекта. Доступ к свойствам осуществляется по имени свойства (иногда говорят «по ключу»).

Свойства объектов

Свойство объекта можно понимать как переменную, закрепленную за объектом. Свойства объекта определяют его характеристики.

```
var myCar = {};
myCar.maker = "Ford";
myCar.model = "Mustang";
myCar.year = 1969;
```

```
var myCar = new Object();
myCar["maker"] = "Ford";
myCar["model"] = "Mustang";
myCar["year"] = 1969;
```

Чтобы прочитать значения также обращаются через точку или квадратные скобки.

```
alert( myCar.maker + ' ' + myCar.model
); // "Ford Mustang"
```

```
alert( myCar["maker"] + ' ' + myCar
["model"] ); // "Ford Mustang"
```

Удаление осуществляется оператором **delete**

```
delete myCar.year;
```

Свойства объектов

Разница между использованием через точку и квадратные скобки следующая: квадратные скобки позволяют использовать в качестве имени свойства любую строку.

```
var person = {};
Person['preferred music style'] =
'Jazz';
```

Квадратные скобки также позволяют обратиться к свойству, имя которого хранится в переменной

```
person.age = 25;
var key = 'age';
alert( person[key] ); // person['age']
```

Вообще, если имя свойства хранится в переменной (`var key = "age"`), то единственный способ к нему обратиться — это квадратные скобки `person[key]`.

Доступ через точку используется, если мы на этапе написания программы уже знаем название свойства. А если оно будет определено по ходу выполнения, например, введено посетителем и записано в переменную, то единственный выбор — квадратные скобки.

Задача 1

1. Создайте пустой объект user
2. Добавьте свойство firstname = “Василий”
3. Добавьте свойство lastname = “Васильев”
4. Добавьте свойство age = 29
5. Замените значения свойств на Иван Иванов соответственно
6. Удалите свойство age

Объявление со свойствами

Объект можно заполнить значениями при создании, указав их в фигурных скобках: `{ ключ1: значение1, ключ2: значение2, ... }`.

Такой синтаксис называется **литеральным** (англ. **literal**).

```
var menuSetup = {
  width: 300,
  height: 200,
  title: "Menu"
};
```

Названия свойств можно перечислять как в кавычках, так и без, если они удовлетворяют ограничениям для имён переменных.

```
var menuSetup = {
  width: 300,
  'height': 200,
  "has submenu": true
};
```

Значения свойств

В качестве значения можно тут же указать и другой объект

```
var user = {
  name: "Genny",
  age: 25,
  size: {
    top: 90,
    middle: 60,
    bottom: 90
  }
}

alert(user.name) // "Genny"
alert(user.size.top) // 90
```

Проверка наличия свойства

Иногда бывает нужно проверить, есть ли в объекте свойство с определенным ключом.

Для этого есть особый оператор: "in".

Его синтаксис: "prop" in obj, причем имя свойства — в виде строки, например

```
if ("name" in user) {
  alert( "person has name" );
}
```

Впрочем, чаще используется другой способ — сравнение значения с undefined.

Дело в том, что в JavaScript можно обратиться к любому свойству объекта, даже если его нет. Ошибки не будет.

Но если свойство не существует, то вернется специальное значение **undefined**.

```
alert( user.name === undefined ); // false
```

Задача 2

1. Создать объект car через литерал с такими свойствами:
    a. Model
    b. Year
    c. Manufacturer
2. Переназначить year
3. Проверить наличие свойств:
    a. Model
    b. Manufacturer
    c. Body

Перебор свойств объекта

Для перебора всех свойств из объекта используется цикл по свойствам `for..in`. Эта синтаксическая конструкция отличается от рассмотренного ранее цикла `for(;;)`.

```
for (var key in obj) {
  /*...делать что-то с obj[key]...*/
}
```

Обратите внимание, мы использовали квадратные скобки `menu[key]`. Как уже говорилось, если имя свойства хранится в переменной, то обратиться к нему можно только так, не через точку.

```
var menu = {
  width: 300,
  height: 200,
  title: "Menu"
};

for (var key in menu) {
  // этот код будет вызван для каждого свойства объекта
  // ..и выведет имя свойства и его значение
  alert( "Key: " + key + " value: " + menu[key] );
}
```

Порядок перебора свойств

Порядок перебора соответствует порядку объявления для нечисловых ключей, а числовые — сортируются (в современных браузерах).

```
var user = {
  firstname: "Rob",
  lastname: "Stark"
};
user.age = 16;
for (var prop in user) {
  alert( prop ); // firstname, lastname, age
}
```

Задача 3

Вывести все свойства объекта из предыдущей задачи в столбик так:

Имя свойства: Значение

Методы объектов

При объявлении объекта можно указать свойство-функцию.

```
var user = {
  name: 'Robert',

  // method
  sayHi: function() {
    alert( 'Hello!' );
  }
};

// call
user.sayHi();
```

Свойства-функции называют «методами» объектов. Их можно добавлять и удалять в любой момент, в том числе и явным присваиванием.

```
var user = {
  name: 'Robert'
};
user.sayHi = function() {
  alert('Hello!');
};

// Call method
user.sayHi();
```

Ключевое слово «this»

Для доступа к текущему объекту из метода используется ключевое слово **this**.

```
var user = {
  name: 'Robert',

  sayHi: function() {
    alert( this.name );
  }
};

user.sayHi(); // sayHi in user context
```

Использование **this** гарантирует, что функция работает именно с тем объектом, в контексте которого вызвана.

Контекст **this** никак не привязан к функции, даже если она создана в объявлении объекта. Чтобы **this** передался, нужно вызвать функцию именно через точку (или квадратные скобки).

Копирование объекта по ссылке

В переменной, которой присвоен объект, хранится не сам объект, а «адрес его места в памяти», иными словами — «ссылка» на него.

При копировании переменной с объектом — копируется эта ссылка, а объект по-прежнему остается в единственном экземпляре.

Так как объект всего один, то изменения через любую переменную видны в других переменных.

```
var user = { name: 'Rob' };

var admin = user;

admin.name = 'John';
alert(user.name); // 'John'
```

Задача 4

1. Создать объект person со свойством name и методом getName, который возвращает имя. **Literal**
2. Добавить (не в литерал) метод sayHello, которое выводит «Привет имя!»

Преобразования объектов

Бывают операции, при которых объект должен быть преобразован в примитив.

Например:

- **Строковое преобразование** — если объект выводится через alert(obj).
- **Численное преобразование** — при арифметических операциях, сравнении с примитивом.
- **Логическое преобразование** — при if(obj) и других логических операциях.

Логическое преобразование

Любой объект в логическом контексте — true, даже если это пустой массив `[]` или объект `{}`.

```
if ({} && []) {
  alert( "All objects - true!" ); // сработает
}
```

Строковое преобразование

Стандартным строковым представлением пользовательского объекта является строка "[object Object]"

```
var user = {
  firstName: 'Robert'
};

alert( user ); // [object Object]
```

Такой вывод объекта не содержит интересной информации. Поэтому имеет смысл его поменять на что-то более полезное.

Если в объекте присутствует метод **toString**, который возвращает примитив, то он используется для преобразования.

```
var user = {
  firstName: 'Robert',

  toString: function() {
    return 'User ' + this.firstName;
  }
};

alert( user );  // User Robert
```

Численное преобразование

Для численного преобразования объекта используется метод valueOf, а если его нет — то toString.

```
var room = {
  number: 777,
  valueOf: function() { return this.
number; },
  toString: function() { return this.
number; }
};
alert( +room );  // 777, valueOf
delete room.valueOf; // valueOf удалён
alert( +room );  // 777, toString
```

При строковом преобразовании объекта используется его метод **toString**. Он должен возвращать примитивное значение, причём не обязательно именно строку.

Для численного преобразования используется метод **valueOf**, который также может возвратить любое примитивное значение. У большинства объектов **valueOf** не работает (возвращает сам объект и потому игнорируется), при этом для численного преобразования используется **toString**.

Задача 5

К объекту из предыдущей задачи добавить метод toString который возвращает строку вида: «Person: имя».

Method chaining

Method chaining — это последовательный вызов методов через точку.

Для того, чтобы это работало, методы должны возвращать объёкт (через this).

Работает это так: когда методы завершают свою работу, они возвращают объект, у которого они изначально были вызваны. Мы можем немедленно вызвать любой из методов объекта.

```
var road = {
      distance: 0,
      goAhead: function(){
            this.distance++;
            return this;
      },
      goBack: function(){
            this.distance--;
            return this;
      },
      getDistance: function(){
            return this.distance;
      }
}
alert(road.goAhead().goAhead().goBack().
getDistance());
```

Задача 6

Дополнить предыдущую задачу так, чтобы методы можно было вызвать «паровозиком» (method chaining). И вызвать из так.

Ссылки

- https://learn.javascript.ru/object
- https://learn.javascript.ru/object-for-in
- https://learn.javascript.ru/object-reference
- https://learn.javascript.ru/object-methods
- https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide/Working_with_Objects
- https://learn.javascript.ru/object-conversion

Вопросы?