JS Fundamentals
Objects

Автор презентации: Макухина Марина

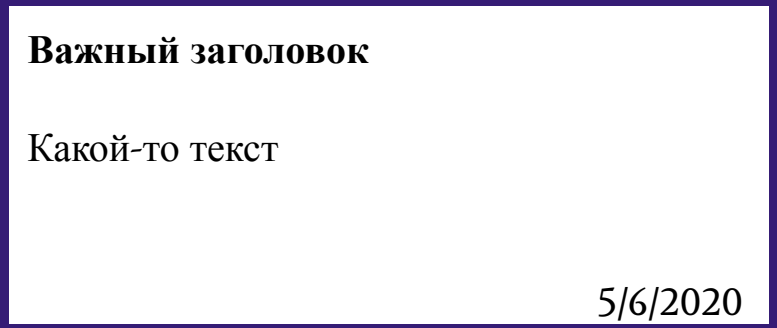
Определение объекта

Объекты — общие структурные элементы, на которых строится большая часть JS.

Простые примитивы (такие как *string*, *boolean*, *number*, *null* и *undefined*) сами по себе объектами не являются.

Null иногда причисляют к объектному типу, но это недоразумение происходит от ошибки в языке, из-за которой *typeof null* неправильно возвращает строку *"object"*. В действительности *null* является отдельным примитивным типом.

Применения объектов



Предположим, заказчик просит реализовать некое хранилище для своих заметок по типу Google Keep. Заметка должна содержать заголовок, текст и информацию о дате последнего изменения заметки.

Заказчик должен иметь возможность изменять или удалять заголовок или текст заметки.

Начать реализацию задачи можно с создания объекта *note*.

Создание объекта

Существуют различные способы создания новых объектов:

используя литеральный синтаксис, благодаря которому определяем и создаем объект в одном операторе.

```
const note = {
  "title": "Важный заголовок",
  "body": "Какой-то текст",
  "edited at": new Date()
}
```

используя ключевое слово *new*.

```
const note = new Object();
note.title = "Важный заголовок";
note.body = "Какой-то текст";
note["edited at"] = new Date();
```

Свойства

Свойство – это пара «ключ: значение», где ключ – это строка, а значение может быть чем угодно. В данном случае нам необходимы следующие свойства: *title*, *body* и *edited at*.

Свойства могут быть в виде строк или символов Если в качестве свойства используется любое другое значение, оно будет преобразовано в строку.

```
console.log(note.title);
console.log(note["edited at"]);
```

Для обращения к свойствам можно использовать запись через точку. Для свойств, имена которых состоят из нескольких слов, доступ к значениею осуществляется через квадратные скобки.

Проверка существования свойства

Чтобы узнать, какими свойствами обладает объект, можно использовать метод *Object.keys()*.

```
alert(Object.keys(note))
// title,body,edited at
```

При обращении к свойству, которого нет, возвращается undefined.

```
console.log(note.author);
// undefined
```

Для проверки существования свойства в объекте используется специальный оператор *in*.

```
console.log("title" in note);
// true
```

Если свойство объекта было инициализировано значением undefined, то оператор in вернёт true при проверке на наличие этого свойства у объекта.

```
note.body = undefined;
console.log("body" in note);
// true
```

Создание, изменение и удаление свойств

Чтобы создать новое свойство или изменить существующее, достаточно просто присвоить значение, предварительно обратившись к нему через точку или через квадратные скобки.

```
note.author = "Иван";
note["body"] = "Новый текст";
console.log(note.author); // "Иван"
console.log(note.body);
// "Новый текст"
```

Присваивание свойству значения undefined не удаляет само свойство. Для удаление используется ключевое слово delete, которое удаляет как значение свойства, так и само свойство.

```
delete note.author;
console.log(note.author);
// undefined
console.log("author" in note);
// false
```

Перебор свойств объекта

Цикл *for...in* используется для перебора всех свойств объекта, в том числе функции и свойства прототипов. Он пройдёт по каждому отдельному элементу, вызывая на каждом шаге ключ. Цикл проходит по свойствам в произвольном порядке.

```
for (let property in note) {
    console.log(property + " is "
       + note[property]);
}
// title is Важный заголовок
```

* Цикл *for...of* не может быть использован для объектов, так как выполняет обход по элементам коллекций (итерируемых объектов, например, Array, String, и т.д.). На каждом шаге итерации будет вызвано значение, а не ключ.

Упорядочение свойств объекта

Свойства в JS сортируются по дате создания. Исключением являются свойства с целочисленными ключами – они упорядочены по возрастанию.

Если объект содержит разные типы ключей, первыми будут размещены целочисленные.

Если ключ 07 записан без кавычек, то он будет обработан как целочисленное значение. Если такое значение записано в кавычках – как строка.

```
let colors = {
    "1": "red",
    07: "violet",
    "4.0": "green",
    5: "blue",
    "03": "yellow",
    "2": "orange",
    "6": "indigo"
  };
  for (let code in colors) {
    alert(code);
  }
// 1  2  5  6  7  4.0  03
```

Методы Object для получения ключей и значений

Метод *Object.keys* возвращает массив строковых элементов, соответствующих именам перечисляемых свойств, найденных непосредственно в самом объекте.

```
let user = { name: "Kate", age: 19 };
console.log(Object.keys(user)); // [ 'name', 'age' ]
```

Метод *Object.values* возвращает массив, чьи элементы это значения перечисляемых свойств найденных в объекте.

```
console.log(Object.values(user));
// [ 'Kate', 19 ]
```

Метод *Object.values* удобно использовать для перебора значений свойств в цикле, что служит очень удобной заменой циклу *for...in*. Разница между циклом и методом в том, что цикл перечисляет свойства и из цепочки прототипов.

Методы объектов

Метод – свойство, содержащее определение функции. Есть 2 способа записи:

```
const passport = {
 code: "1234",
 getId: function () {
  return "FA" + this.code;
 }
};
```

```
const passport = {
 code: "1234",
 getId() {
  return "FA" + this.code;
 }
};
```

Чтобы вызвать функцию объекта, необходимо указать имя свойство со скобками.
Если вызвать такое свойство без скобок – оно вернет определение функции.

```
alert(passport.getId());  // FA1234
```

Сравнение объектов

Объекты имеют ссылочный тип. Два отдельных объекта никогда не будут равными, даже если они имеют равный набор свойств.

В результате присвоения одного объекта другому получатся две ссылки на одну и ту же область памяти. При сравнение двух таких переменных будет получено *true*.

```
let bird1 = { name: "sparrow" };
let bird2 = bird1;

bird2.name = "magpie";

console.log(bird1 == bird2);
// true
console.log(bird1.name);
// magpie
```

Копирование объектов

Если объект состоит только из примитивных данных, то для копирования можно использовать метод *Object.assign* или *spread operator.*

Оба способа позволяют копировать свойства из нескольких объектов в один.

```
let book1 = { name: "Cinderella" };
let book2 = Object.assign({}, book);
book2.name = "Snow White";
console.log(book2);
//{name: "Snow White"}
let info = {
  author: "Grimm",
};
let bookInfo = { ...book2, ...info };
console.log(bookInfo);
//{name: "Snow White", author: "Grimm"}
```

Поверхностная копия

Метод *Object.assign(...)*, как и *spread operator*, делает только поверхностную копию: для тех свойств объекта, которые сами являются объектами, копируются только ссылки. Кроме того, данный способ копирует из исходных объектов в новый объект только перечисляемые и собственные свойства.

```
let day = { name: "Mon", date: { id: 1 } };
Object.defineProperty(day, "isFirst", { value: true, enumerable: false });
let dayCopy = Object.assign({}, day);
dayCopy.name = "Tue";
dayCopy.date.id = 2;
console.log(day);      // {name: "Mon", date: {id: 2}, isFirst: true}
console.log(dayCopy); // {name: "Tue", date: {id: 2}}
```

Глубокая копия

Глубокая копия продублирует каждый объект на пути копирования.

Для этого необходимо преобразовать объект JavaScript в строку JSON с помощью метода *JSON.stringify()*, а затем выполнить обратное преобразование с помощью *JSON.parse()*. Метод *JSON.stringify()* пропустит значение undefined, функцию и символ.

```
let day = { name: "Mon", date: { id: 1 }, prnt() { return "test"; }};
let dayCopy = JSON.parse( JSON.stringify (day));
dayCopy.date.id = 2;
console.log(day); // {name: "Mon", date: {id: 1}, prnt: ƒ}
console.log(dayCopy); // {name: "Mon", date: {id: 2}}
```

Полезные ссылки

https://learn.javascript.ru/object

https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Object

https://learn.javascript.ru/object-copy

https://learn.javascript.ru/keys-values-entries

https://learn.javascript.ru/json

На этом всё!

•••

Спасибо за внимание