JS Fundamentals
Built-in objects, Arrays
...

Автор презентации: Макухина Марина

Встроенные объекты Number, Boolean, String

Числа, логические значения и строки и имеют соответствующие типы объектов (*Number*, *Boolean*, *String*) которые принято также называть объектными обертками.

Объектные обертки служат очень важной цели. Примитивные значения не имеют свойств или методов, так что для обращения к методам *length* или *toString* необходима объектная обертка для значения. JavaScript автоматически упаковывает примитивные значения для обслуживания таких обращений:

```
let value = "Hello";
console.log(value.length);  // 5
console.log(value.toUpperCase());  // HELLO
```

Создание объектных оберток

Объектные обертки создаются через вызов соответствующего метода: *Number()*, *Boolean()*, *String()*. Значение, переданное первым параметром, при необходимости будет преобразовано к соответствующему типу.

В случае использование объекта *Number*, если аргумент не может быть преобразован в число, возвращается *NaN*.

```
let notNum = Number("Hello");
console.log(notNum);  // Nan

let num = Number("0001.2");
console.log(num);  // 1.2
```

Тонкости работы с объектом Boolean

При создании объектной обертки *Boolean* происходит преобразование к логическому значению. Если переданное значение опущено или равно пустой строке, *0, -0, false, null, undefined, NaN*, объект будет иметь значение *false*. Все остальные значения создают объект с начальным значением, равным *true*.

Любой объект, чьё значение не является равным *undefined* или *null*, включая сам объект *Boolean* со значением *false*, является “истинными”.

```
let value = new Boolean(false);
if (value) {
  console.log("Should not be printed");
  // Should not be printed
}
```

Явное использование объектных оберток

Когда в программе происходит попытка обратиться к методу строки *value*, интерпретатор JavaScript преобразует строковое значение в объект, как если бы был выполнен вызов *new String(value)*. Этот объект используется интерпретатором для доступа к строковым методам. После обращения к методу этот объект уничтожается.

```
let value = "Hello";
console.log(value.length);
```

Не стоит использовать конструкции вида *new String(“Hello”)*, *new Number(1)*, *new Boolean(true)*, так как неявная упаковка внутри браузера будет происходить быстрее, чем прямое использованием объектной формы. Для приведенного примера лучше использовать литералы-примитивы *“Hello”*, *1* и *true*.

Методы String

Метод	Описание
concat()	Объединяет текст двух строк и возвращает новую строку.
split()	Разбивает строку на массив строк, разделенных указанной строкой на подстроки.
toLowerCase()	Возвращает строковое значение с символами в нижнем регистре.
toUpperCase()	Возвращает строковое значение с символами в верхнем регистре.
replace()	Возвращает новую строку с некоторыми или всеми сопоставлениями с шаблоном, замененными на заменитель.
search()	Выполняет поиск совпадения регулярного выражения со строкой.
trim()	Обрезает пробельные символы в начале и в конце строки.

Массивы

Массивы (*arrays*) – это объекты специального типа. Содержимое массива упорядочено, а ключи являются числовыми и последовательными. Массивы представляют собой контейнеры для значений любого типа (*string*, *number*, *boolean*, *object* и т.п.), в том числе другого массива.

Существует два варианта синтаксиса для создания массива:

```
let array = new Array();
```

```
let array = [];
```

При инициализации массива, в скобках можно сразу указать начальные значения элементов:

```
let array = new Array("one", "two");
```

```
let array = ["one", "two"];
```

Особенности работы с массивом

Размер массива не ограничен: можно добавить или удалить элементы в любое время. Элементы массива нумеруются с нуля. Первый элемент в массиве – элемент с индексом 0.

Узнать количество элементов массива можно с помощью свойства *length*.

```
console.log(array.length);  // 2
```

Получить элемент массива можно указав его номер в квадратных скобках. Для изменения значения элемента массива достаточно присвоить ему новое значение:

```
array[0] = 1;
console.log(array[0]); // 1
```

Манипулирование содержимым массива

Добавить новый элемент массива можно обратившись к несуществующему элементу через индекс *(arr.length - 1)* и присвоив ему значение.

```
let colors = ["red", "green", "blue"];
colors[3] = "white";
console.log(colors);
// ["pink", "green", "blue", "white"]
```

Если создать элемент с индексом, намного превышающим длину массива, то будут образованы «дыры» в виде пустых элементов. Массивы, содержащие как минимум один «пустой элемент», часто называются «разреженными» (*sparse*).

```
colors[6] = "black";
console.log(colors.length);  // 7
console.log(colors);
// ["red", "green", "blue", undefined, undefined, undefined, "black"]
```

Добавление и удаление элементов массива

Добавить новый элемент в конец массива можно с помощью метода *push*. Метод *pop* удаляет последний элемент из массива. Добавить элемент в начало массива позволяет метод *unshift* (первый элемент сдвигается на вторую позицию). Метод shift удаляет из массива первый элемент (второй элемент становится первым).

```
let prime = [3, 5, 7, 179];
console.log(prime.pop());  // 179
console.log(prime.push(11)); // 4
console.log(prime); // [3, 5, 7, 11]
```

```
console.log(prime.shift()); // 3
console.log(prime); // [5, 7, 11]
console.log(prime.unshift( 2)); // 4
console.log(prime) // [2, 5, 7, 11]
```

Методы *push* и *unshift* возвращают новую длину массива после добавления нового элемента, а pop и shift возвращают удаленный элемент.

Добавление нескольких элементов в массив

Метод *concat* добавляет в массив несколько элементов и возвращает его копию.

```
const numbers = [1, 2];
const newNumbers = numbers.concat( 3, 4, ["400", "500"]);
console.log(newNumbers);  //  [1, 2, 3, 4, "400", "500"]
```

Метод *concat* не изменяет исходный массив, а лишь возвращает поверхностную копию, содержащую копии тех элементов, что были объединены с исходным массивом.

Метод *concat* копирует ссылки на объекты в новый массив. И оригинал, и новый массив ссылаются на один и тот же объект. Для примитивных типов метод копирует значения в новый массив.

Цикл while

Циклом называется многократное выполнение одних и тех же действий. Цикл — это та часть кода программы, которая выполняется заданное количество раз.

Цикл *while* повторяет код, пока выполняется его условие (condition). В приведенном примере, цикл будет выполняться до тех пор, пока значение *number* будет меньше 6. Внутри цикла происходит увеличение *number* на единицу в каждой итерации. Если упустить этот шаг, цикл будет выполняться бесконечно, поскольку условие всегда останется истинным.

```
let number = 1;
while (number < 6) {
  console.log(number);
  number++;
}
// 1, 2, 3, 4, 5
```

Проверку условия можно разместить под телом цикла, используя специальный синтаксис *do ... while.*

Цикл for

Цикл *for* чрезвычайно гибок и он даже может заменить цикл *while* или *do ... while*. Этот цикл лучше всего подходит для случая, когда необходимо фиксированное количество циклов (особенно когда необходимо знать номер текущего цикла).

Цикл for состоит из трех частей: инициализация (*i = 0*), условие (*i < 3*) и выражение (*i ++*). Это нечто, что не может быть создано с помощью цикла *while*, здесь вся информация цикла удобно размещается в одном месте.

```
for (let i = 0; i < 3; i++) {
  console.log("Iteration: " + i);
  // Iteration: 0
  // Iteration: 1
  // Iteration: 2
}
```

Общепринятое соглашение: использовать переменную *i* в цикле *for* независимо от того, что подсчитывается, хотя допустимо использовать любое имя переменной.

Перебор элементов массива

Есть несколько способов, с помощью которых можно перебрать элементы массива:

1. Цикл *for*. Проверка осуществляется на длину массива.

```
let arr = ["a", "b", "c"];
for (let i = 0; i < arr.length; i++) {
  console.log(array[i]);
} // a b c
```

2. Оператор *for ... of*. При итерациях используется значения массива.

```
for (let value of arr) {
  value += 1;
  console.log(value);
} // a1, b1, c1
```

Кроме того, существует также оператор *for ... in*, который используется для прохода через перечисляемые (*enumerable*) свойства объекта. Он пройдёт по каждому отдельному элементу, вызывая на каждом шаге ключ. Этот подход не рекомендуется использовать для массивов, где важен порядок индексов.

Прерывание и переход к следующей итерации

Существуют два оператора, позволяющие изменить нормальный ход управления потоком в цикле. Оператор *break* позволяет немедленно прерывать цикл. Использование оператора *continue* позволяет перейти к следующему шагу в цикле.

```
for (;;) {
  console.log("I am working");
  let value = prompt("Finish? (y)");
  if (value === "y") {
    break;
  }
}
console.log("Next step");
```

```
for (let i = 0; i < 3; i++) {
  let value = prompt("Print? (y)");
  if (value !== "y") {
    continue;
  }
  console.log(i);
}
console.log("Next step");
```

Методы Array

Метод	Описание
isArray()	Возвращает true, если значение является массивом, иначе возвращает false.
reverse()	Переворачивает порядок элементов в массиве — первый элемент становится последним, а последний — первым.
sort()	Сортирует элементы массива на месте и возвращает отсортированный массив
join()	Объединяет все элементы массива в строку.
every()	Возвращает true, если каждый элемент в массиве удовлетворяет условию проверяющей функции.

Некоторые методы принимают в качестве аргументов функции, вызываемые при обработке массива. Более детально они будут рассмотрены на следующих уроках.

Примеры использование методов массива

Узнать, является ли какое-то значение массивом, можно с помощью метода *isArray*.

```
let isArray = Array.isArray([1]);
console.log(isArray); // true
```

Метод reverse() на месте переставляет элементы массива, на котором он был вызван, изменяет массив и возвращает ссылку на него.

```
const arr = ['one', 'two', 'three'];
arr.reverse();
console.log(arr); // [ 'three', 'two', 'one' ]
```

Полезные ссылки

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

https://learn.javascript.ru/array

https://learn.javascript.ru/while-for

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

На этом всё!

•••

Спасибо за внимание