Дополнительные возможности синтаксиса JavaScript

Как и любой язык программирования, *JavaScript* менялся со временем. С момента появления в 1995 новые возможности добавлялись каждые несколько лет. В 1997 появился *ECMAScript*, его целью было направить развитие *JavaScript* в нужное русло. Выходили новые версии — ES3, ES5, ES6 и так далее, но промежутки между выходом версий длились по 6, а то и по 10 лет. Новая модель стандарта языка предполагает делать маленькие изменения каждый год вместо того, чтобы накопить огромное количество изменений и выпустить их все за раз, как это было с ES6. В ES6 настолько сильно поменялся стандарт языка, что выпускались отдельные справочники именно по ES6 (к примеру, «You Don't Know JS: ES6 & Beyond»).

Версия ES6 вышла в июне 2015 года. Это принесло некую путаницу в название пакета, ведь ES6 и ES2015 — это одно и то же. А поскольку по новой модели стандарта языка обновления выпускались каждый год, ES6 переименовали в ES2015, чтобы отражать год релиза. И все следующие версии теперь называются в соответствии с годом их выпуска.

При разборе примеров я навыке мы будем использовать синтаксис и конструкции, которые появились в ES2015: `const`/`let`, интерполяция, оператор spread, деструктурирующее присваивание, стрелочные функции и другие. Чтобы лучше понимать код и, главное, применять его на практике, рассмотрим некоторые из этих конструкций более детально. Начнём с шаблонных строк и интерполяции.

Шаблонные строки и интерполяция

Шаблонные строки или шаблонные литералы — это строковые литералы, которые допускают использование выражений внутри. С ними вы можете записывать строковые литералы в несколько строк и использовать строковую интерполяцию.

Пример литерала на несколько строк:

```
console.log(`Освещена последняя сосна.
Под нее тёмный краж пушится,
Сейчас погаснет и она.
День конченый — не повторится.`);
```

Интерполяция строк уже давно поддерживается в других языках программирования, к примеру, в *PHP* и *C#*. Вообще интерполяция — это замена переменных-заполнителей или выражений в строке на их значения. Переменные-заполнители и выражения передаются в строку в виде `${...}`.

Например:

```
const value = 13;
console.log(`Квадрат числа ${value} равен ${value*value}`);
// Квадрат числа 13 равен 169
```

В выражениях можно использовать условия, например так:

```
<p class="description short"></p>
```

```
const description = document.querySelector('.description');

console.log(`<a class="more" href="#">${description.classList.contains('short') ? 'Читать дальше' :
'Скрыть описание'}</a>`);
// "<a class="more" href="#">Читать дальше</a>"
```

Больше информации о строковых литералах можно найти в спецификации.

Деструктурирующее присваивание

Деструктурирующее присваивание (или деструктуризация) в выражениях *JavaScript* используется, чтобы было проще извлекать данные из массивов или объектов. Разберём на примерах. Начнём с массива.

```
const weekDays = ['Понедельник', 'Вторник', 'Среда', 'Четверг', 'Пятница', 'Суббота', 'Воскресенье'];

// без деструктуризации
const monday = weekDays[0];
const tuesday = weekDays[1];
const wednesday = weekDays[2];
const thursday = weekDays[3];
const friday = weekDays[4];
const saturday = weekDays[5];
const sunday = weekDays[6];

// с деструктуризацией
const [monday, tuesday, wednesday, thursday, friday, saturday, sunday] = weekDays;
```

С помощью деструктуризации можно быстро поменять местами значения переменных.

```
let first = 1;
let last = 3;

[first, last] = [last, first];

// теперь first содержит значение 3, а last - 1
```

Если массивы деструктуризация разбирает по элементам, то объекты распадаются на свойства.

Допустим, так:

```
const fullName = {name: 'Пётр', surname: 'Петров'};
const {name, surname} = fullName;
console.log(`${name} ${surname}`);
// Пётр Петров
```

С помощью деструктурирующего присваивания можно легко разбирать в *JavaScript*-коде data-атрибуты. К примеру, у нас есть следующая разметка:

```
<p>Температура воды:</p>
<meter value="0" max="100" low="10" high="60" data-description="Низкая"></meter>
<meter value="30" max="100" low="10" high="60" data-description="Нормальная"></meter>
<meter value="80" max="100" low="10" high="60" data-description="Горячая"></meter>
<meter value="100" max="100"  data-description="Кипяток"></meter>
```

Получим значение data-атрибута `data-description` для тега `<meter>`:

```
const element = document.querySelector('meter');
const {description} = element.dataset;

console.log(`${description}`);
// Низкая
```

Подробнее деструктурирующее присваивание описывается в спецификации.

Оператор spread

Оператор *spread* (`...`) или распаковку аргументов удобно использовать при вызове функций с большим числом параметров. Чтобы не перечислять параметры функции через запятую, можно поместить их в массив, а затем распаковать за счёт оператора `...`.

Допустим, у нас есть функция для склеивания строк, входящие параметры передаются через запятую:

```
const concat = (str1, str2, str3, str4, str5, str6, str7, str8) => {
 return str1 + str2 + str3 + str4 + str5 + str6 + str7 + str8;
};

const countdown = [
  'Ты кавай,',
  'Я кавай,',
  'Мы орём на весь трамвай,',
  'Манга,',
  'Неки, ',
  'Семен-ай,',
  'Дэсу,',
  'Ты води давай!!!'];

console.log(concat(...countdown));

// результат:
// Ты кавай,Я кавай,Мы орём на весь трамвай,Манга,Неки,Семен-ай,Дэсу,Ты води давай!!!
```

Стрелочные функции

У стрелочных функций в *JavaScript* есть два преимущества перед классическими функциями — это более короткий синтаксис и отсутствие собственного контекста `this`. Про `this` мы рассказываем в тренажёре «Программирование на JavaScript». Плюсы более компактного синтаксиса рассмотрим на примерах.

```
const elements = [
  'Hydrogen',
  'Helium',
  'Lithium',
  'Beryllium'
];

elements.map(function(element) {
  return element.length;
});
// Это выражение вернёт массив [8, 6, 7, 9]
```

Функцию в параметрах метода `map` можно записать как стрелочную:

```
elements.map((element) => {
  return element.length;
}); // Получим тот же результат, массив [8, 6, 7, 9]
```

Если единственный оператор в выражении стрелочной функции это `return`, то его можно удалить, а вместе с ним и окружающие фигурные скобки. Так запись функции станет ещё компактнее:

```
elements.map(element => element.length); // [8, 6, 7, 9]
```

А теперь попробуем переписать объявление функции:

```
function calendar() {
  return '3 сентября';
};
console.log(calendar());

// эта запись эквивалентна
calendar = () => '3 сентября';
console.log(calendar());
```

И для функциональных выражений применим стрелочную функцию:

```
const multiply = function (a, b) {return a * b};
console.log(multiply(2, 2));

// эта запись эквивалентна
const multiply = (a, b) => a * b;
console.log(multiply(2, 2));
```

Обо всех особенностях стрелочных функций читайте в спецификации.

Ознакомились со статьёй?

Сохранить прогресс

Общая теория | Обработка данных