Остаточные параметры и оператор расширения

Многие встроенные функции JavaScript поддерживают произвольное количество аргументов.

Например:

- `Math.max(arg1, arg2, ..., argN)` – вычисляет максимальное число из переданных.
- `Object.assign(dest, src1, ..., srcN)` – копирует свойства из исходных объектов `src1..N` в целевой объект `dest`.
- ...и так далее.

В этой главе мы узнаем, как сделать то же самое с нашими собственными функциями и как передавать таким функциям параметры в виде массива.

Остаточные параметры (...)

Вызывать функцию можно с любым количеством аргументов независимо от того, как она была определена.

Например:

```
1  function sum(a, b) {
2    return a + b;
3  }
4
5  alert( sum(1, 2, 3, 4, 5) );
```

Лишние аргументы не вызовут ошибку. Но, конечно, посчитаются только первые два.

Остаточные параметры могут быть обозначены через три точки `...`. Буквально это значит: «собери оставшиеся параметры и положи их в массив».

Например, соберём все аргументы в массив `args`:

```
1  function sumAll(...args) { // args – имя массива
2    let sum = 0;
3
4    for (let arg of args) sum += arg;
5
6    return sum;
7  }
8
9  alert( sumAll(1) ); // 1
10 alert( sumAll(1, 2) ); // 3
11 alert( sumAll(1, 2, 3) ); // 6
```

Мы можем положить первые несколько параметров в переменные, а остальные – собрать в массив.

В примере ниже первые два аргумента функции станут именем и фамилией, а третий и последующие превратятся в массив `titles`:

```
1  function showName(firstName, lastName, ...titles) {
2    alert( firstName + ' ' + lastName ); // Юлий Цезарь
3
```


```
  // Оставшиеся параметры пойдут в массив
  // titles = ["Консул", "Император"]
  alert( titles[0] ); // Консул
  alert( titles[1] ); // Император
  alert( titles.length ); // 2
}

showName("Юлий", "Цезарь", "Консул", "Император");
```

Остаточные параметры должны располагаться в конце

Остаточные параметры собирают все остальные аргументы, поэтому бессмысленно писать что-либо после них. Это вызовет ошибку:

```
function f(arg1, ...rest, arg2) { // arg2 после ..
  // Ошибка
}
```

`...rest` должен всегда быть последним.

Переменная "arguments"

Все аргументы функции находятся в псевдомассиве `arguments` под своими порядковыми номерами.

Например:

```
function showName() {
  alert( arguments.length );
  alert( arguments[0] );
  alert( arguments[1] );

  // Объект arguments можно перебирать
  // for (let arg of arguments) alert(arg);
}

// Вывод: 2, Юлий, Цезарь
showName("Юлий", "Цезарь");

// Вывод: 1, Илья, undefined (второго аргумента нет)
showName("Илья");
```

Раньше в языке не было остаточных параметров, и получить все аргументы функции можно было только с помощью `arguments`. Этот способ всё ещё работает, мы можем найти его в старом коде.

Но у него есть один недостаток. Хотя `arguments` похож на массив, и его тоже можно перебирать, это всё же не массив. Он не поддерживает методы массивов, поэтому мы не можем, например, вызвать `arguments.map(...)`.

К тому же, `arguments` всегда содержит все аргументы функции — мы не можем получить их часть. А остаточные параметры позволяют это сделать.

Соответственно, для более удобной работы с аргументами лучше использовать остаточные параметры.

Стрелочные функции не имеют "arguments"

Если мы обратимся к `arguments` из стрелочной функции, то получим аргументы внешней «нормальной» функции.

Пример:

```
function f() {
  let showArg = () => alert(arguments[0]);
  showArg(2);
}

f(1); // 1
```

Как мы помним, у стрелочных функций нет собственного `this`. Теперь мы знаем, что нет и своего объекта `arguments`.

Оператор расширения

Мы узнали, как получить массив из списка параметров.

Но иногда нужно сделать в точности противоположное.

Например, есть встроенная функция Math.max. Она возвращает наибольшее число из списка:

```
alert( Math.max(3, 5, 1) ); // 5
```

Допустим, у нас есть массив чисел `[3, 5, 1]`. Как вызвать для него `Math.max`?

Просто так их не вставишь — `Math.max` ожидает получить список чисел, а не один массив.

```
let arr = [3, 5, 1];

alert( Math.max(arr) ); // NaN
```

Конечно, мы можем вводить числа вручную: `Math.max(arr[0], arr[1], arr[2])`. Но, во-первых, это плохо выглядит, а, во-вторых, мы не всегда знаем, сколько будет аргументов. Их может быть как очень много, так и не быть совсем.

И тут нам поможет *оператор расширения*. Он похож на остаточные параметры – тоже использует `...`, но делает совершенно противоположное.

Когда `...arr` используется при вызове функции, он «расширяет» перебираемый объект `arr` в список аргументов.

Для `Math.max`:

```
let arr = [3, 5, 1];

alert( Math.max(...arr) ); // 5 (оператор "раскрывает"
```

Этим же способом мы можем передать несколько итерируемых объектов:

```
let arr1 = [1, -2, 3, 4];
let arr2 = [8, 3, -8, 1];

alert( Math.max(...arr1, ...arr2) ); // 8
```

Мы даже можем комбинировать оператор расширения с обычными значениями:

```
let arr1 = [1, -2, 3, 4];
let arr2 = [8, 3, -8, 1];

alert( Math.max(1, ...arr1, 2, ...arr2, 25) ); // 25
```

Оператор расширения можно использовать и для слияния массивов:

```
let arr = [3, 5, 1];
let arr2 = [8, 9, 15];

let merged = [0, ...arr, 2, ...arr2];

alert(merged); // 0,3,5,1,2,8,9,15 (0, затем arr, затем
```

В примерах выше мы использовали массив, чтобы продемонстрировать свойства оператора расширения, но он работает с любым перебираемым объектом.

Например, оператор расширения подойдёт для того, чтобы превратить строку в массив символов:

```
let str = "Привет";

alert( [...str] ); // П,р,и,в,е,т
```

Посмотрим, что происходит. Под капотом оператор расширения использует итераторы, чтобы перебирать элементы. Так же, как это делает `for..of`.

Цикл `for..of` перебирает строку как последовательность символов, поэтому из `...str` получается `"П", "р", "и", "в", "е", "т"`. Получившиеся символы собираются в массив при помощи стандартного объявления массива: `[...str]`.

Для этой задачи мы можем использовать и `Array.from`. Он тоже преобразует перебираемый объект (такой как строка) в массив:

```
let str = "Привет";

// Array.from преобразует перебираемый объект в массив
alert( Array.from(str) ); // П,р,и,в,е,т
```

Результат аналогичен `[...str]`.

Но между `Array.from(obj)` и `[...obj]` есть разница:

- `Array.from` работает как с псевдомассивами, так и с итерируемыми объектами
- Оператор расширения работает только с итерируемыми объектами

Выходит, что если нужно сделать из чего угодно массив, то `Array.from` — более универсальный метод.

Итого

Когда мы видим `"..."` в коде, это могут быть как остаточные параметры, так и оператор расширения.

Как отличить их друг от друга:

- Если `...` располагается в конце списка аргументов функции, то это «остаточные параметры». Он собирает остальные неуказанные аргументы и делает из них массив.
- Если `...` встретился в вызове функции или где-либо ещё, то это «оператор расширения». Он извлекает элементы из массива.

Раздел

Продвинутая работа с функциями

Навигация по уроку

Поделиться

Редактировать на GitHub

Полезно запомнить:

- Остаточные параметры используются, чтобы создавать новые функции с неопределённым числом аргументов.
- С помощью оператора расширения можно вставить массив в функцию, которая по умолчанию работает с обычным списком аргументов.

Вместе эти конструкции помогают легко преобразовывать наборы значений в массивы и обратно.

К аргументам функции можно обращаться и по-старому — через псевдомассив `arguments`.

Комментарии

перед тем как писать...