|  |  |
| --- | --- |
| **Spread syntax** (...) allows an iterable such as an array expression or string to be expanded in places where zero or more arguments (for function calls) or elements (for array literals) are expected, or an object expression to be expanded in places where zero or more key-value pairs (for object literals) are expected. | Синтаксис распространения (...) позволяет расширить итерацию, такую ​​как выражение массива или строка, в местах, где ожидается ноль или более аргументов (для вызовов функций) или элементов (для литералов массива), или выражение объекта, которое должно быть развернуто в места, где ожидается ноль или более пар ключ-значение (для объектных литералов). |
| Spread syntax can be used when all elements from an object or array need to be included in a list of some kind. | Синтаксис распространения можно использовать, когда все элементы объекта или массива необходимо включить в какой-либо список. |
| In the above example, the defined function takes x, y, and z as arguments and returns the sum of these values. An array value is also defined. | В приведенном выше примере определенная функция принимает x, y и z в качестве аргументов и возвращает сумму этих значений. Также определяется значение массива. |
| When we invoke the function, we pass it all the values in the array using the spread syntax and the array name — ...numbers. | Когда мы вызываем функцию, мы передаем ей все значения в массиве, используя синтаксис распространения и имя массива - ... числа. |
| If the array contained more than three numbers, e.g. [1, 2, 3, 4], then it would still work fine, except that all four would be passed, but only the first three would be used unless you added more arguments to the function, e.g.: | Если в массиве более трех чисел, например [1, 2, 3, 4], тогда он все равно будет работать нормально, за исключением того, что все четыре будут переданы, но будут использоваться только первые три, если вы не добавите дополнительные аргументы в функцию, например: |
| The above example is somewhat rigid; the real value in spread syntax is that it works with the same value, no matter how many elements are contained in the object, array, etc. | Приведенный выше пример несколько жесток; реальное значение синтаксиса распространения состоит в том, что он работает с одним и тем же значением, независимо от того, сколько элементов содержится в объекте, массиве и т. д. |
| It is commonly used when you want to add a new item to a local data store, or display all stored items plus a new addition. A very simple version of this kind of action could look like so: | Если вы используете конструктор RegExp со строковым литералом, помните, что обратная косая черта является escape-символом в строковых литералах, поэтому, чтобы использовать его в регулярном выражении, вам нужно экранировать его на уровне строкового литерала. / a \ \* b / и new RegExp («a \\ \* b») создают одно и то же выражение, которое ищет «a», за которым следует буквальный «\*», за которым следует «b» |
| In the above example you can rerun the last line as many times as you like, to keep adding an additional 12 to the end of the array. | В приведенном выше примере вы можете повторно запускать последнюю строку сколько угодно раз, чтобы продолжать добавлять еще 12 в конец массива. |
| Rest syntax looks exactly like spread syntax. In a way, rest syntax is the opposite of spread syntax. Spread syntax "expands" an array into its elements, while rest syntax collects multiple elements and "condenses" them into a single element. See [rest parameters](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/rest_parameters). | В приведенном выше примере повторно запустить пример сколько угодно раз. |
| If you need to use any of the special characters literally (actually searching for a "\*", for instance), you must escape it by putting a backslash in front of it. For instance, to search for "a" followed by "\*" followed by "b", you'd use /a\\*b/ — the backslash "escapes" the "\*", making it literal instead of special. | Если вам нужно использовать какой-либо из специальных символов буквально (например, при поиске "\*"), вы должны избежать его, поставив перед ним обратную косую черту. Например, для поиска «a», за которым следует «\*», за которым следует «b», вы должны использовать / a \ \* b / - обратная косая черта «экранирует» «\*», делая его буквальным, а не специальным. |
| Similarly, if you're writing a regular expression literal and need to match a slash ("/"), you need to escape that (otherwise, it terminates the pattern). For instance, to search for the string "/example/" followed by one or more alphabetic characters, you'd use /\/example\/[a-z]+/i—the backslashes before each slash make them literal. | Точно так же, если вы пишете литерал регулярного выражения и вам нужно сопоставить косую черту ("/"), вам нужно убрать его (в противном случае он завершает шаблон). Например, для поиска строки «/ example /», за которой следует один или несколько буквенных символов, вы должны использовать / \ / example \ / [a-z] + / i - обратная косая черта перед каждой косой чертой делает их буквальными. |
| To match a literal backslash, you need to escape the backslash. For instance, to match the string "C:\" where "C" can be any letter, you'd use /[A-Z]:\\/ — the first backslash escapes the one after it, so the expression searches for a single literal backslash. | Чтобы соответствовать буквальной обратной косой черте, вам нужно избегать обратной косой черты. Например, чтобы соответствовать строке «C: \», где «C» может быть любой буквой, вы должны использовать / [AZ]: \\ / - первая обратная косая черта экранирует следующую после нее, поэтому выражение ищет одну буквальная обратная косая черта. |
| If using the RegExp constructor with a string literal, remember that the backslash is an escape in string literals, so to use it in the regular expression, you need to escape it at the string literal level. /a\\*b/ and new RegExp("a\\\*b") create the same expression, which searches for "a" followed by a literal "\*" followed by "b". | Если вы используете конструктор RegExp со строковым литералом, помните, что обратная косая черта является escape-символом в строковых литералах, поэтому, чтобы использовать его в регулярном выражении, вам нужно экранировать его на уровне строкового литерала. / a \ \* b / и new RegExp («a \\ \* b») создают одно и то же выражение, которое ищет «a», за которым следует буквальный «\*», за которым следует «b». |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |