



Операторы языка JavaScript

Арифметические операторы. Инкремент и декремент

- ♦ В языке JavaScript присутствуют стандартные для языков программирования арифметические операторы, позволяющие производить вычисления с численными и строковыми значениями (для строк только оператор +).
- ♦ К арифметическим операторам JavaScript относятся: + (сложение), - (вычитание), * (умножение), / (деление).
- ♦ В дополнение к ним присутствует оператор взятия остатка от деления %. Все указанные операторы являются бинарными (в том смысле, что принимают два значения и возвращают одно). Кроме указанных операторов, существует еще и унарный оператор -, инвертирующий значение аргумента (например -123, -val).
- ♦ В JavaScript предусмотрена также удобная возможность записи выражений вида $i = i + 1$, $i = i - 1$, $i = i * j$ и пр., где i - произвольная переменная, а j - произвольное выражение.
- ♦ Первые два выражения сокращенно записываются как инкремент и декремент: $i++$ и $i--$. Третье выражение и подобные ему можно сократить, применив следующие операторы:
 - ♦ • оператор -=, то есть $i = i - j$ эквивалентно $i -= j$;
 - ♦ • оператор +=, то есть $i = i + j$ эквивалентно $i += j$;
 - ♦ • оператор *=, то есть $i = i * j$ эквивалентно $i *= j$;
 - ♦ • оператор /=, то есть $i = i / j$ эквивалентно $i /= j$;
 - ♦ • оператор %=, то есть $i = i \% j$ эквивалентно $i \% = j$.
- ♦ Кроме того, предусмотрены соответствующие операторы &=, ^=, |= для двоичных операторов и <<=, >>=, >>>= для операторов сдвига.

Логические операторы и операторы сравнения

- ◆ Оператор сравнения сравнивает свои операнды и возвращает логическое значение, базируясь на истинности сравнения. Операнды могут быть числами, строками, логическими величинами или объектами. Строки сравниваются на основании стандартного лексикографического порядка, используя Unicode-значения. В большинстве случаев, если операнды имеют разный тип, то JavaScript пробует преобразовать их в тип, подходящий для сравнения. Такое поведение обычно происходит при сравнении числовых операндов. Единственным исключением из данного правила является сравнение с использованием операторов `===` и `!==`, которые производят строгое сравнение на равенство или неравенство. Эти операторы не пытаются преобразовать операнды перед их сравнением. Следующая таблица описывает операторы сравнения в контексте следующего примера кода:

```
var var1 = 3, var2 = 4;
```

Арифметические операторы

- ◆ Арифметические операторы используют в качестве своих операндов числа (также литералы или переменные) и в качестве результата возвращают одно числовое значение. Стандартными арифметическими операторами являются сложение (+), вычитание (-), умножение (*), и деление (/). При работе с числами с плавающей точкой эти операторы работают аналогично их работе в большинстве других языках программирования (обратите внимание, что деление на ноль возвращает бесконечность Infinity). Например:
- ◆ `console.log(1 / 2); /* возвращает 0.5 */`
- ◆ `Console.log(1 / 2 === 1.0 / 2.0); /* возвращает true */`

Битовые (поразрядные) операторы

- ◆ Битовые операторы обрабатывают свои операнды как последовательности из 32 бит (нулей и единиц), а не как десятичные, шестнадцатеричные или восьмеричные числа. Например, десятичное число 9 имеет двоичное представление 1001. Битовые операторы выполняют операции над таким двоичным представлением, но результат возвращают как обычное числовое значение JavaScript.

Следующая таблица обобщает битовые операторы JavaScript.

◆ Битовые операторы

Оператор	Использование	Описание
Побитовое И	<code>a & b</code>	Возвращает единицу в каждой битовой позиции, для которой соответствующие биты обоих операндов являются единицами.
Побитовое ИЛИ	<code>a b</code>	Возвращает единицу в каждой битовой позиции, для которой один из соответствующих битов или оба бита операндов являются единицами.
Исключающее ИЛИ	<code>a ^ b</code>	Возвращает единицу в каждой битовой позиции, для которой только один из соответствующих битов операндов является единицей.
Побитовое НЕ	<code>~ a</code>	Заменяет биты операнда на противоположные.
Сдвиг влево	<code>a << b</code>	Сдвигает <code>a</code> в двоичном представлении на <code>b</code> бит влево, добавляя справа нули.
Сдвиг вправо с переносом знака	<code>a >> b</code>	Сдвигает <code>a</code> в двоичном представлении на <code>b</code> бит вправо, отбрасывая сдвигаемые биты.
Сдвиг вправо с заполнением нулями	<code>a >>> b</code>	Сдвигает <code>a</code> в двоичном представлении на <code>b</code> бит вправо, отбрасывая сдвигаемые биты и добавляя слева нули.

Примеры работы битовых операторов

Выражение	Результат	Двоичное описание
$15 \& 9$	9	$1111 \& 1001 = 1001$
$15 9$	15	$1111 1001 = 1111$
$15 \wedge 9$	6	$1111 \wedge 1001 = 0110$
~ 15	-16	$\sim 00000000 \dots 00001111 = 1111 1111 \dots 11110000$
~ 9	-10	$\sim 00000000 \dots 0000 1001 = 1111 1111 \dots 1111 0110$

Логические операторы

Логические операторы обычно используются с булевыми (логическими) значениями; при этом возвращаемое ими значение также является булевым. Однако операторы `&&` и `||` фактически возвращают значение одного из операндов, поэтому, если эти операторы используются с небулевыми величинами, то возвращаемая ими величина также может быть не булевой. Логические операторы описаны в следующей таблице.

◆ Сокращённая оценка

Так как логические выражения вычисляются слева направо, они проверяются на возможность выполнения сокращённой оценки с использованием следующих правил:

`false && anything` - сокращение с результатом `false`.

`true || anything` - сокращение с результатом `true`.

Правила логики гарантируют, что данные вычисления всегда корректны. Обратите внимание, что часть "anything" представленных выше выражений не вычисляется, таким образом удастся избежать любых побочных эффектов вычисления данной части.

Оператор запятая

- ◆ Оператор запятая (,) просто вычисляет оба операнда и возвращает значение последнего операнда. Данный оператор в основном используется внутри цикла for, что позволяет при каждом прохождении цикла одновременно обновлять значения нескольких переменных.
- ◆ Например, если a является двумерным массивом, каждая строка которого содержит 10 элементов, то следующий код с использованием оператора запятая позволяет выполнять одновременное приращение двух переменных. Данный код выводит на экран значения диагональных элементов массива:

```
for (var i = 0, j = 9; i <= 9; i++, j--)  
    document.writeln("a[" + i + "][" + j + "] = " + a[i][j]);
```

Операторы отношения

Оператор отношения сравнивает свои операнды и возвращает результат сравнения в виде булева значения.

◆ Оператор `in`

Оператор `in` возвращает `true`, если указанный объект имеет указанное свойство. Синтаксис оператора:

`propNameOrNumber in objectName`

где `propNameOrNumber` - строка или числовое выражение, представляющее имя свойства или индекс массива, а `objectName` - имя объекта.

Приоритет операторов

- ◆ Приоритет операторов определяет порядок их выполнения при вычислении выражения. Вы можете влиять на приоритет операторов с помощью скобок.
- ◆ Приведенная ниже таблица описывает приоритет операторов от наивысшего до низшего.

Тип оператора	Операторы
свойство объекта	<code>.</code> <code>[]</code>
вызов, создание экземпляра объекта	<code>()</code> <code>new</code>
отрицание, инкремент	<code>!</code> <code>~</code> <code>-</code> <code>++</code> <code>--</code> <code>typeof</code> <code>void</code> <code>delete</code>
умножение, деление	<code>*</code> <code>/</code> <code>%</code>
сложение, вычитание	<code>+</code> <code>-</code>
побитовый сдвиг	<code><<</code> <code>>></code> <code>>>></code>
сравнение, вхождение	<code><</code> <code><=</code> <code>></code> <code>>=</code> <code>in</code> <code>instanceof</code>
равенство	<code>==</code> <code>!=</code> <code>===</code> <code>!==</code>
битовое-и	<code>&</code>
битовое-исключающее-или	<code>^</code>
битовое-или	<code> </code>
логическое-и	<code>&&</code>
логическое-или	<code> </code>
условный (тернарный) оператор	<code>?:</code>
присваивание	<code>=</code> <code>+=</code> <code>-=</code> <code>*=</code> <code>/=</code> <code>%=</code> <code><<=</code> <code>>>=</code> <code>>>>=</code> <code>&=</code> <code>^=</code> <code> =</code>
запятая	<code>,</code>

Основные выражения

Базовые ключевые слова и основные выражения в JavaScript.

◆ Оператор *this*

Используйте ключевое слово `this` для указания на текущий объект. В общем случае `this` указывает на вызываемый объект, которому принадлежит данный метод. Используйте `this` следующим образом:

```
this["propertyName"]
```

```
this.propertyName
```

Левосторонние выражения

Значениям слева назначаются значения справа.

◆ **New**

Вы можете использовать оператор `new` для создания экземпляра объекта пользовательского типа или одного из встроенных объектов. Используйте оператор `new` следующим образом:

```
var objectName = new objectType([param1, param2, ..., paramN]);
```

◆ **Super**

Ключевое слово используется, чтобы вызывать функции родительского объекта. Это полезно и с классами для вызова конструктора родителя, например.

```
super([arguments]); // вызывает конструктор родителя.  
super.functionOnParent([arguments]);
```

Вопросы на закрепление:

- ◆ 1. Как работает JavaScript, - однопоточный язык программирования. Это означает, что у него один стек вызовов. Таким образом, в некий момент времени он может выполнять лишь какую-то одну задачу. Стек вызовов — это структура данных, которая, говоря упрощённо, записывает сведения о месте в программе, где мы находимся.
- ◆ 2. Структура сценария JavaScript, - Сценарием JavaScript считается фрагмент кода, расположенный между дескрипторами `<SCRIPT>` и `</SCRIPT>`:
 - ◆ Текст HTML-документа
 - ◆ `<SCRIPT>`
 - ◆ Код сценария
 - ◆ `</SCRIPT>`
 - ◆ Текст HTML-документа
- ◆ 3. Перечислите операторы JavaScript? -
 - ◆ Операторы присваивания
 - ◆ Операторы сравнения
 - ◆ Арифметические операторы
 - ◆ Бинарные операторы
 - ◆ Логические операторы
 - ◆ Строковые операторы
 - ◆ Условный (тернарный) оператор
 - ◆ Оператор запятая
 - ◆ Унарные операторы
 - ◆ Операторы отношения
 - ◆ Приоритет операторов