18 сентября 2020 г.

Операторы сравнения

Многие операторы сравнения известны нам из математики.

В JavaScript они записываются так:

- Больше/меньше: `a > b`, `a < b`.
- Больше/меньше или равно: `a >= b`, `a <= b`.
- Равно: `a == b`. Обратите внимание, для сравнения используется двойной знак равенства `==`. Один знак равенства `a = b` означал бы присваивание.
- Не равно. В математике обозначается символом `≠`, но в JavaScript записывается как `a != b`.

В этом разделе мы больше узнаем про то, какие бывают сравнения, как язык с ними работает и к каким неожиданностям мы должны быть готовы.

В конце вы найдёте хороший рецепт того, как избегать «причуд» сравнения в JavaScript.

Результат сравнения имеет логический тип

Все операторы сравнения возвращают значение логического типа:

- `true` – означает «да», «верно», «истина».
- `false` – означает «нет», «неверно», «ложь».

Например:

```
1  alert( 2 > 1 );  // true (верно)
2  alert( 2 == 1 ); // false (неверно)
3  alert( 2 != 1 ); // true (верно)
```

Результат сравнения можно присвоить переменной, как и любое значение:

```
1  let result = 5 > 4; // результат сравнения присваивается переменной result
2  alert( result ); // true
```

Сравнение строк

Чтобы определить, что одна строка больше другой, JavaScript использует «алфавитный» или «лексикографический» порядок.

Другими словами, строки сравниваются посимвольно.

Например:

```
alert( 'Я' > 'А' ); // true
alert( 'Коты' > 'Кода' ); // true
alert( 'Сонный' > 'Сон' ); // true
```

Алгоритм сравнения двух строк довольно прост:

1. Сначала сравниваются первые символы строк.
2. Если первый символ первой строки больше (меньше), чем первый символ второй, то первая строка больше (меньше) второй. Сравнение завершено.
3. Если первые символы равны, то таким же образом сравниваются уже вторые символы строк.
4. Сравнение продолжается, пока не закончится одна из строк.
5. Если обе строки заканчиваются одновременно, то они равны. Иначе, большей считается более длинная строка.

В примерах выше сравнение `'Я' > 'А'` завершится на первом шаге, тогда как строки `'Коты'` и `'Кода'` будут сравниваться посимвольно:

1. `К` равна `К`.
2. `о` равна `о`.
3. `т` больше, чем `д`. На этом сравнение заканчивается. Первая строка больше.

> **Используется кодировка Unicode, а не настоящий алфавит**
>
> Приведённый выше алгоритм сравнения похож на алгоритм, используемый в словарях и телефонных книгах, но между ними есть и различия.
>
> Например, в JavaScript имеет значение регистр символов. Заглавная буква `"А"` не равна строчной `"а"`. Какая же из них больше? Строчная `"а"`. Почему? Потому что строчные буквы имеют больший код во внутренней таблице кодирования, которую использует JavaScript (Unicode). Мы ещё поговорим о внутреннем представлении строк и его влиянии в главе Строки.

Сравнение разных типов

При сравнении значений разных типов JavaScript приводит каждое из них к числу.

Например:

```
alert( '2' > 1 ); // true, строка '2' становится числом 2
alert( '01' == 1 ); // true, строка '01' становится числом 1
```

Логическое значение `true` становится `1`, а `false` – `0`.

Например:

```
alert( true == 1 ); // true
alert( false == 0 ); // true
```

Забавное следствие

Возможна следующая ситуация:

- Два значения равны.
- Одно из них `true` как логическое значение, другое – `false`.

Например:

```
let a = 0;
alert( Boolean(a) ); // false

let b = "0";
alert( Boolean(b) ); // true

alert(a == b); // true!
```

С точки зрения JavaScript, результат ожидаем. Равенство преобразует значения, используя числовое преобразование, поэтому `"0"` становится `0`. В то время как явное преобразование с помощью `Boolean` использует другой набор правил.

Строгое сравнение

Использование обычного сравнения `==` может вызывать проблемы. Например, оно не отличает `0` от `false`:

```
alert( 0 == false ); // true
```

Та же проблема с пустой строкой:

```
alert( '' == false ); // true
```

Это происходит из-за того, что операнды разных типов преобразуются оператором `==` к числу. В итоге, и пустая строка, и `false` становятся нулём.

Как же тогда отличать `0` от `false`?

Оператор строгого равенства `===` проверяет равенство без приведения типов.

Другими словами, если `a` и `b` имеют разные типы, то проверка `a === b` немедленно возвращает `false` без попытки их преобразования.

Давайте проверим:

```
alert( 0 === false ); // false, так как сравниваются разные типы
```

Ещё есть оператор строгого неравенства `!==`, аналогичный `!=`.

Оператор строгого равенства дольше писать, но он делает код более очевидным и оставляет меньше места для ошибок.

Сравнение с null и undefined

Поведение `null` и `undefined` при сравнении с другими значениями — особое:

При строгом равенстве `===`

Эти значения различны, так как различны их типы.

```
1  alert( null === undefined ); // false
```

При нестрогом равенстве `==`

Эти значения равны друг другу и не равны никаким другим значениям. Это специальное правило языка.

```
1  alert( null == undefined ); // true
```

При использовании математических операторов и других операторов сравнения `< > <= >=`

Значения `null/undefined` преобразуются к числам: `null` становится `0`, а `undefined` – `NaN`.

Посмотрим, какие забавные вещи случаются, когда мы применяем эти правила. И, что более важно, как избежать ошибок при их использовании.

Странный результат сравнения null и 0

Сравним `null` с нулём:

```
1  alert( null > 0 );  // (1) false
2  alert( null == 0 ); // (2) false
3  alert( null >= 0 ); // (3) true
```

С точки зрения математики это странно. Результат последнего сравнения говорит о том, что "`null` больше или равно нулю", тогда результат одного из сравнений выше должен быть `true`, но они оба ложны.

Причина в том, что нестрогое равенство и сравнения `> < >= <=` работают по-разному. Сравнения преобразуют `null` в число, рассматривая его как `0`. Поэтому выражение (3) `null >= 0` истинно, а `null > 0` ложно.

С другой стороны, для нестрогого равенства `==` значений `undefined` и `null` действует особое правило: эти значения ни к чему не приводятся, они равны друг другу и не равны ничему другому. Поэтому (2) `null == 0` ложно.

Несравненное значение undefined

Значение `undefined` несравнимо с другими значениями:

```
1  alert( undefined > 0 ); // false (1)
2  alert( undefined < 0 ); // false (2)
3  alert( undefined == 0 ); // false (3)
```

Почему же сравнение `undefined` с нулём всегда ложно?

На это есть следующие причины:

- Сравнения `(1)` и `(2)` возвращают `false`, потому что `undefined` преобразуется в `NaN`, а `NaN` – это специальное числовое значение, которое возвращает `false` при любых сравнениях.
- Нестрогое равенство `(3)` возвращает `false`, потому что `undefined` равно только `null`, `undefined` и ничему больше.

Как избежать проблем

Зачем мы рассмотрели все эти примеры? Должны ли мы постоянно помнить обо всех этих особенностях? Не обязательно. Со временем все они станут вам знакомы, но можно избежать проблем, если следовать надёжным правилам:

- Относитесь очень осторожно к любому сравнению с `undefined/null`, кроме случаев строгого равенства `===`.
- Не используйте сравнения `>= > < <=` с переменными, которые могут принимать значения `null/undefined`, разве что вы полностью уверены в том, что делаете. Если переменная может принимать эти значения, то добавьте для них отдельные проверки.

Итого

- Операторы сравнения возвращают значения логического типа.
- Строки сравниваются посимвольно в лексикографическом порядке.
- Значения разных типов при сравнении приводятся к числу. Исключением является сравнение с помощью операторов строгого равенства/неравенства.
- Значения `null` и `undefined` равны `==` друг другу и не равны любому другому значению.
- Будьте осторожны при использовании операторов сравнений вроде `>` и `<` с переменными, которые могут принимать значения `null/undefined`. Хорошей идеей будет сделать отдельную проверку на `null/undefined`.

Задачи

Операторы сравнения

важность: 5

Каким будет результат этих выражений?

```
1  5 > 4
2  "ананас" > "яблоко"
3  "2" > "12"
4  undefined == null
5  undefined === null
6  null == "\n0\n"
7  null === +"\n0\n"
```

решение

Предыдущий урок | Следующий урок

Поделиться

Карта учебника

Комментарии

- Если вам кажется, что в статье что-то не так - вместо комментария напишите на GitHub.
- Для одной строки кода используйте тег `<code>`, для нескольких строк кода — тег `<pre>`, если больше 10 строк — ссылку на песочницу (plnkr, JSBin, codepen...)
- Если что-то непонятно в статье — пишите, что именно и с какого места.

о проекте связаться с нами пользовательское соглашение политика конфиденциальности