Интернет-технологии и распределённая обработка данных

Лекция 11

Выражения и операторы JavaScript

1. Выражения в JavaScript
2. Арифметические операторы , операторы инкремента и декремента
3. Операторы присваивания
4. Операторы сравнения и логические операторы
5. Побитовые операторы
6. Специальные операторы
7. Приоритет операторов
8. Взаимодействие с пользователем: alert, prompt, confirm

Выражения

- *Выражение* – последовательность символов JavaScript, которая может быть **вычислена** транслятором для получения **значения**.

- Иногда сам **процесс вычисления** важнее полученного значения, которое может вовсе не использоваться – в этом случае говорят об использовании **побочного эффекта** выражения


переменная
операнды
var x = 50 + "day";
операторы

Разновидности выражений

1. Первичные выражения
2. Инициализаторы
3. Выражения обращения к свойствам
4. Выражения вызова
5. Выражения создания объектов
6. Выражения с операторами

Первичные выражения

Такие выражения не включают более простых подвыражений.

1. Литералы для number, string, регулярных выражений.

2. Некоторые ключевые слова: `true`, `false`, `null` (мы их рассматривали как литералы), `this` – контекст вызова функции («текущий объект»).

3. Ссылки на переменные.

```
12.3          // Числовой литерал
"текст"       // Строковый литерал
/regexp/      // Литерал регулярного выражения

this  // Возвращает "текущий" объект
true  // Возвращает логическое значение true
false // Возвращает логическое значение false
null  // Возвращает значение null

i             // Возвращает значение переменной i
sum           // Возвращает значение переменной sum
undefined     // глобальная переменная, а не ключевое слово, как
null
```

Инициализаторы

1. Порождающие объект («литералы объектов») – т.е. *значением выражения является новый объект*.

```
var point= {x:2.3, y:-4.2};   // объект с 2-я свойствами x и y
```

2. Порождающие массив («литералы массивов»).

```
var arr = [1, 2+3]; // Массив из 2-х элементов.
                    //Первый элемент – 1, второй – 5
```

3. Определяющие функцию («литералы функций»).

```
var f = function (x) { return x > 0; } ;
```

Выражения обращения к свойствам

Значение = *значение свойства* или *элемент массива*.

1. *выражение.идентификатор*
2. *выражение*[*выражение*]

```
point.x      // значение = 2.3
point["x"] // значение = 2.3
arr[1]       // значение = 5
```

Выражения вызова

Состоит из:

Выражения, возвращающего функцию

Круглые скобки

В скобках через запятую – выражения для аргументов

```
func(1, 2);
arr.sort();
```

Значением является: то, что вернула функция (или undefined)

Выражения создания объекта

```
new Point(2, 3)
```

Выражение создаёт новый объект, затем передаёт этот объект указанной функции (*конструктору*) в качестве `this`. *Значением* является:

- созданный объект (если конструктор ничего не возвращает или возвращает не объект)

или

- объект, возвращённый конструктором.

Операторы

Использование операторов – основной способ конструирования выражений

Характеристики любого оператора:

- количество *операндов*
- ассоциативность
- тип операндов
- приоритет

- **унарный**
- **бинарный**
- **тернарный**

«Левостороннее выражение»

- это выражения, которые могут стоять слева от оператора присваивания.

Означает одно из трёх:

1. Переменная
2. Свойство объекта
3. Элемент массива

Арифметические операторы

`x + y` `x - y` `x * y` `x / y` `x % y`

число, число → число

Традиционная семантика (за исключением +).

Операнды преобразуются к числам, результатом является число.

Любая арифметическая операция с `NaN` даёт в итоге `NaN`, также если преобразование типов невозможно.

Конкатенация строк

```
x + y
```

строка, строка → строка

Это сцепление строк. При необходимости операнды преобразуются к строке.

```
alert( '1' + 2 ); // "12«
alert( 2 + '1' ); // "21"
```

Конкатенация vs. сложение

`x + y`

Конкатенация «предпочитаема». Если хотя бы один из операндов строка, то будет конкатенация строк. Не важно слева или справа строка.

Внимание: если один из операндов объект, то у объекта вызывается метод `valueOf()`, и дальнейшее поведение операции `+` зависит от типа возвращённого значения.

Конкатенация vs. сложение

```
var obj = {
  toString: function() {
    return "[object MyObject]";
  },
  valueOf: function() {
    return 17;
  }
};

console.log(obj + 3);     // число 20
console.log(obj + "3");   // строка 173
```

```
var obj = {
  toString: function() {
    return "[object MyObject]";
  },
  valueOf: function() {
    return 17;
  }
};

console.log(obj + 3);    //
число 20
console.log(obj + "3");  //
строка 173
```

```
var obj = {
  toString: function() {
    return "[object MyObject]";
  },
  valueOf: function() {
    return "17";
  }
};

console.log(obj + 3);    //
строка 173
console.log(obj + "3");  //
строка 173
```

Унарный плюс и унарный минус

```
+x          -y
```

число → число

Унарный плюс – преобразование операнда в число – в этом его польза.

Операторы присваивания

x = y

левостороннее выражение, любой → любой

Выполняется присваивание переменной или свойству. Значением выражения является значение y.

Существует «присваивание с операцией»: += -= *= /= %= <<= >>= >>>= &= |= ^=

x += 10; // равнозначно x = x + 10;

```
var a, b, c;
a = b = c = 2 + 2;
alert( a ); // 4
alert( b ); // 4
alert( c ); // 4
```

```
var a = 1;
var b = 2;
var c = 3 - (a = b + 1);
alert( a ); // 3
alert( c ); // 0
```

```
var n = 2;
n *= 3 + 5;
alert( n ); // 16  (n = 2 * 8)
```

//присваивание выполнится после других операций

Инкремент и декремент

`++x` `x++` `--y` `y--`

левостороннее выражение → число

Оператор ++ преобразует свой операнд в число, прибавляет единицу, помещает новое значение обратно в операнд.

Оператор -- работает по аналогичной схеме.

Возвращаемое значение операторов зависит от позиции:

x++ значение операнда **после преобразования в число**, но **до увеличения**.

++x значение операнда **после увеличения**.

```
var x  = 5, z, s, k, l;
z = ++x * 2; /*  z = 12, x = 6,  x сначала увеличивается на 1,
                потом  умножение */
var y=5;   s = y++ * 2;    /*  s = 10, y = 6, сначала умножение,
                потом в y увеличенное на 1 значение */
var d=5;   k = --d * 2; //  k = 8, d = 4
var c=5;   l = c-- * 2; //  l = 10, c = 4
```

Поразрядные битовые операторы

Оператор	Описание
~	Поразрядная инверсия (унарный)
<<	Сдвиг влево
>>	Сдвиг вправо с сохранением знака
>>>	Сдвиг вправо с заполнением нулями
&	Поразрядное И
^	Поразрядное ИСКЛЮЧАЮЩЕЕ ИЛИ
|	Поразрядное ИЛИ

Операторы работают с 32-х разрядными числами. Если нужно, выполняется преобразование операндов к числу, отбрасывание дробной части и «лишних» старших битов. Значения `NaN`, `Infinity` и `-Infinity` преобразуются в `0`.

Этим свойством операторов иногда пользуются, чтобы преобразовать типы.

Почему побитовые операции в примерах ниже не меняют число? Что они делают внутри? $123_{10}=1111011_{2}$

```
alert( 123 ^ 0 ); // 123

alert( 0 ^ 123 ); // 123

alert( ~~123 ); // 123
```

Операторы сравнения

Проверка идентичности

`x === y` `x !== y`

любой, любой → логический

Проверка без преобразования типов. Более строгая, чем у операторов `==` и `!=`.

true, если операнды равны без преобразования типов

Проверка идентичности

Операнды имеют разные типы: **не идентичны**

Один или оба `NaN`: **не идентичны** (`x!==x` это `true` только для `NaN`)

Оба операнда `null` или оба операнда `undefined`: **идентичны**

Оба операнда `true` или оба операнда `false`: **идентичны**

Оба операнда являются равными числами : **идентичны**

`0` и `-0`: **идентичны**

Оба операнда являются равными строками: **идентичны**

Оба операнда ссылаются на один и тот же объект: **идентичны**

Проверка равенства

`x == y` `x != y`

любой, любой → логический

Проверка на равенство с предварительной попыткой приведения типов.

«Злые близнецы» для === и !==

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

Та же ситуация с пустой строкой:

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

Когда нужно отличить 0 от false:

```
alert( 0 === false );    // false, т.к. типы различны
```

Проверка равенства

Операнды имеют одинаковые типы: выполняется проверка на идентичность

Одно значение `null`, а второе `undefined`: **равны**

Строка и число: строка преобразуется в число, затем сравнение

Операнд-`boolean`: `true` заменяем на `1`, `false` на `0`, затем сравнение

Операнд-объект: преобразуется в число, затем сравнение

Любые другие комбинации не являются равными

x < y x > y x <= y x >= y

число, число → логический

строка, строка → логический

Операторы сравнения работают для чисел и строк. Сравнение чисел «предпочитаемо» -- сравнение строк, только если оба операнда являются строками.

Сравнение с NaN всегда возвращает false.

```
alert( 'Б' > 'А' ); // true
alert( 'Вася' > 'Ваня' ); // true, т.к. 'с' > 'н'
alert( 'а' > 'Я' ); // true,
alert( 'Привет' > 'Прив' ); // true, так как 'е' больше чем "ничего".
alert( "2" > "14" ); // true
alert( +"2" > +"14" ); // false
```

null и undefined равны == друг другу и не равны чему бы то ни было ещё

```
alert( null > 0 ); // false

alert( null == 0 ); // false

//null не больше и не равен нулю

alert(null >= 0); // true
```

Сравнение приводит к числу, получается ноль.

А при проверке равенства значения null и undefined они равны друг другу, но не равны чему-то ещё

```
alert( undefined > 0 ); // false
alert( undefined < 0 ); // false
//undefined при преобразовании к числу даёт NaN.
```

по стандарту сравнения ==, <, >, <=, >= и даже === с NaN возвращают false

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

в стандарте – undefined равно лишь null и ничему другому.

любые сравнения с undefined/null, кроме точного ===, следует делать с осторожностью.

Логические операторы: || (ИЛИ), && (И) и ! (НЕ)

Логическое ИЛИ

- `x || y`
- *любой, любой → любой*
- Вычисляем значение выражения `x`.
- Если значение «истинно», возвращаем его (не `true`, а именно значение выражения `x`).
- Иначе вычисляем значение выражения `y` и возвращаем это значение.

```
alert( true || true ); // true
alert( false || true ); // true
alert( true || false ); // true
alert( false || false ); // false

var x;
true || (x = 1);
alert(x); // undefined, x не присвоен

var x;
false || (x = 1);
alert(x); // 1
```

|| запинается на «правде» – вычисляет значения до первого true

```
alert( 1 || 0 ); // 1

alert( true || 'неважно что' ); // true

alert( null || 1 ); // 1

alert( undefined || 0 ); // 0
```

Используют для выбора первого «истинного» значения из списка:

```
var undef; // переменная не присвоена, т.е. равна undefined

var zero = 0; var emptyStr = ""; var msg = "Привет!";

var result = undef || zero || emptyStr || msg || 0; alert( result );

// выведет "Привет!" - первое значение, которое является true
```

Если все значения «ложные», то || возвратит последнее из них:

```
alert( undefined || ' ' || false || 0 ); // 0
```

Логическое И

`x && y`

любой, любой → любой

Вычисляем значение выражения `x`.

Если значение «ложно», возвращаем его и конец вычислений

(не `false`, а **именно значение выражения `x`**).

Иначе вычисляем значение выражения `y` и возвращаем это значение.

```
alert( true && true ); // true
alert( false && true ); // false
alert( true && false ); // false
alert( false && false ); // false
```

//&& запинается на «лжи».

```
// Первый аргумент - true, возвращается второй аргумент
alert( 1 && 0 ); // 0
alert( 1 && 5 ); // 5
// Первый аргумент - false, возвращается, второй игнорируется
alert( null && 5 ); // null
alert( 0 && "не важно" ); // 0
alert( 1 && 2 && null && 3 ); // null
alert( 1 && 2 && 3 ); // 3
```

Приоритет оператора && больше, чем ||

```
alert( 5 || 1 && 0 ); // 5


var x = 1; (x > 0) && alert( 'Больше' );
//можно , но не рекомендуется
```

Логическое НЕ

.! x

любой → логический

Операнд преобразуется в логическое значение, которое затем инвертируется и возвращается.

Дважды применив этот оператор, можно преобразовать любое значение x в его логический эквивалент.

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

двойное НЕ используют для преобразования значений к логическому типу:

```
alert( !!"строка" ); // true
alert( !!null ); // false
```

Тернарный оператор

```
x ? y : z
```

логический, любой, любой → любой

Значение выражения x преобразуется к типу boolean. Если получилось `true` – вычисляем и возвращаем y, иначе вычисляем и возвращаем z.

Rez=условие ? ifTrue : ifFalse);

Оператор typeof

- `typeof x` или `typeof(x)`

любой → строка

Возвращает строку с именем «типа» операнда `x`.

Оператор typeof

x	typeof x
undefined	"undefined"
null	"object"
true *или* false	"boolean"
любое число	"number"
любая строка	"string"
любая функция	"function"
любой объект, не являющийся функцией	"object"

Оператор instanceof – проверка типа объекта

является ли объект экземпляром определенного класса

```
obj instanceof Constr
```

объект, функция → логический

Возвращает `true`, если объект `obj` (или его прототип) был создан с использованием функции-конструктора `Constr`.

```
var d = new Date();  // конструктор Date()
d instanceof Date;   // true
d instanceof Object; // true
// все (обычные) объекты - это экземпляры Object
```

Оператор in – проверка наличия свойства

```
prop in obj
```

строка, объект → логический

Возвращает `true`, если объект `obj` (или его прототип) содержит свойство с именем `prop`.

```
// Массивы
var trees = new Array("redwood", "bay", "cedar", "oak", "maple");
0 in trees // true
3 in trees // true
6 in trees // false
"bay" in trees // false (вы должны указать индекс элемента
// в массиве, а не значение в этом индексе)
"length" in trees // true (length является свойством Array)
// Уже существующие объекты
"PI" in Math // true
// Пользовательские объекты
var mycar = {make: "Honda", model: "Accord", year: 1998};
"make" in mycar // true
"model" in mycar // true
```

Оператор delete

```
delete obj.x
```

левостороннее выражение → логический

Пытается удалить свойство из объекта или элемент из массива (при этом в массиве образуется «дырка»).

В случае успешного удаления возвращает `true` (обычно значение игнорируется – важен побочный эффект).

Не все свойства могут быть удалены!

Оператор delete

```
var o = {x: 1, y: 2};   // определить объект
delete o.x;             // удалить его свойство true
"x" in o;               // => false

var a = [1,2,3];        // создать массив
delete a[2];            // удалить последний элемент true
2 in a;                 // => false
a.length;               // => 3: длина не изменилась
```

Оператор delete

- В строгом режиме:
  - при попытке удаления несуществующего свойства – ошибка TypeError.
  - при попытке применения не к свойству и не к элементу массива – ошибка SyntaxError.

Оператор void вычисляет переданное *выражение* и возвращает undefined

```
void x
```

любой → `undefined`

Вычисляет значение, отбрасывает его, возвращает `undefined`.

Обычно: в URL-адресах вида `javascript:`, где позволяет вычислить выражение ради его побочного эффекта, не отображая вычисленное значение в браузере.

```
<a href="javascript: void window.open();">Новое окно</a>
```

Оператор , (запятая)

`x, y`

любой, любой → любой

Вычисляет левый операнд, вычисляет правый операнд и возвращает значение правого операнда.

Обычно: в циклах для инициализации счётчиков

```
for (var i = 0, j = 10; i < j; i++, j--)
    alert(i + j);
```

Приоритет операторов

определяет порядок, в котором операторы выполняются. Операторы с более высоким приоритетом выполняются первыми.

Приоритет операторов можно изменить при помощи скобок (и).

Выражение вызова и выражение обращения к свойству имеют более высокий приоритет, чем все операторы в таблице.

Приоритет операторов	Операторы
1	++ и --, + и – (унарные), ~, !, delete, typeof, void
2	*, /, %
3	+ и – (бинарные)
4	<<, >>, >>>
5	<, <=, >, >=, instanceof, in
6	==, !=, ===, !==
7	&
8	^
9	|
10	&&
11	||
12	?:
13	=
14	,

взаимодействие с пользователем: alert, prompt, confirm

alert(сообщение) – выводит на экран модальное окно с сообщением и приостанавливает выполнение скрипта, пока пользователь не нажмёт «ОК».

```
alert( "Привет" );
```

prompt – выводит модальное окно с заголовком title, полем для ввода текста, заполненным строкой по умолчанию default и кнопками OK/CANCEL.

```
result = prompt(title, default); /*Пользователь должен либо что-то ввести и нажать OK, либо отменить ввод кликом на CANCEL или нажатием Esc на клавиатуре.*/
```

Вызов prompt возвращает то, что ввёл посетитель – строку или специальное значение null, если ввод отменён.

```
var years = prompt('Сколько вам лет?', 100); //100 – default
alert('Вам ' + years + ' лет!')
```

confirm выводит окно с вопросом question с двумя кнопками: OK и CANCEL.

```
result = confirm(question);
```

Результатом будет true при нажатии OK и false – при CANCEL(Esc).

Например:

```
var isAdmin = confirm("Вы - администратор?");
alert( isAdmin );
```