**Преобразование объектов: toString и valueOf**

Ранее, в главе [Преобразование типов для примитивов](http://learn.javascript.ru/types-conversion) мы рассматривали преобразование типов для примитивов. Теперь добавим в нашу картину мира объекты.

Бывают операции, при которых объект должен быть преобразован в примитив.

Например:

* Строковое преобразование – если объект выводится через alert(obj).
* Численное преобразование – при арифметических операциях, сравнении с примитивом.
* Логическое преобразование – при if(obj) и других логических операциях.

Рассмотрим эти преобразования по очереди.

**[Логическое преобразование](http://learn.javascript.ru/object-conversion" \l "logicheskoe-preobrazovanie)**

Проще всего – с логическим преобразованием.

**Любой объект в логическом контексте – true, даже если это пустой массив [] или объект {}.**

if ({} && []) {

alert( "Все объекты - true!" ); // alert сработает

}

**[Строковое преобразование](http://learn.javascript.ru/object-conversion" \l "strokovoe-preobrazovanie)**

Строковое преобразование проще всего увидеть, если вывести объект при помощи alert:

var user = {

firstName: 'Василий'

};

alert( user ); // [object Object]

Как видно, содержимое объекта не вывелось. Это потому, что стандартным строковым представлением пользовательского объекта является строка "[object Object]".

Такой вывод объекта не содержит интересной информации. Поэтому имеет смысл его поменять на что-то более полезное.

**Если в объекте присутствует метод toString, который возвращает примитив, то он используется для преобразования.**

var user = {

firstName: 'Василий',

toString: function() {

return 'Пользователь ' + this.firstName;

}

};

alert( user ); // Пользователь Василий

**Результатом toString может быть любой примитив**

Метод toString не обязан возвращать именно строку.

Его результат может быть любого примитивного типа. Например, это может быть число, как в примере ниже:

var obj = {

toString: function() {

return 123;

}

};

alert( obj ); // 123

Поэтому мы и называем его здесь *«строковое преобразование»*, а не «преобразование к строке».

Все объекты, включая встроенные, имеют свои реализации метода toString, например:

alert( [1, 2] ); // toString для массивов выводит список элементов "1,2"

alert( new Date ); // toString для дат выводит дату в виде строки

alert( function() {} ); // toString для функции выводит её код

**[Численное преобразование](http://learn.javascript.ru/object-conversion" \l "chislennoe-preobrazovanie)**

Для численного преобразования объекта используется метод valueOf, а если его нет – то toString:

var room = {

number: 777,

valueOf: function() { return this.number; },

toString: function() { return this.number; }

};

alert( +room ); // 777, вызвался valueOf

delete room.valueOf; // valueOf удалён

alert( +room ); // 777, вызвался toString

Метод valueOf обязан возвращать примитивное значение, иначе его результат будет проигнорирован. При этом – не обязательно числовое.

**У большинства объектов нет valueOf**

У большинства встроенных объектов такого valueOf нет, поэтому численное и строковое преобразования для них работают одинаково.

Исключением является объект Date, который поддерживает оба типа преобразований:

alert( new Date() ); // toString: Дата в виде читаемой строки

alert( +new Date() ); // valueOf: кол-во миллисекунд, прошедших с 01.01.1970

**Детали спецификации**

Если посмотреть в стандарт, то в пункте [15.2.4.4](http://es5.github.com/x15.2.html#x15.2.4.4) говорится о том, что valueOf есть у любых объектов. Но он ничего не делает, просто возвращает сам объект (непримитивное значение!), а потому игнорируется.

**[Две стадии преобразования](http://learn.javascript.ru/object-conversion" \l "dve-stadii-preobrazovaniya)**

Итак, объект преобразован в примитив при помощи toString или valueOf.

Но на этом преобразования не обязательно заканчиваются. Вполне возможно, что в процессе вычислений этот примитив будет преобразован во что-то другое.

Например, рассмотрим применение к объекту операции ==:

var obj = {

valueOf: function() {

return 1;

}

};

alert( obj == true ); // true

Объект obj был сначала преобразован в примитив, используя численное преобразование, получилось 1 == true.

Далее, так как значения всё ещё разных типов, применяются правила преобразования примитивов, результат: true.

То же самое – при сложении с объектом при помощи +:

var obj = {

valueOf: function() {

return 1;

}

};

alert( obj + "test" ); // 1test

Или вот, для разности объектов:

var a = {

valueOf: function() {

return "1";

}

};

var b = {

valueOf: function() {

return "2";

}

};

alert( a + b ); // "12"

alert( a - b ); // "1" - "2" = -1

**Исключение: Date**

Объект Date по историческим причинам является исключением.

Бинарный оператор плюс + обычно использует численное преобразование и метод valueOf. Как мы уже знаем, если подходящего valueOf нет (а его нет у большинства объектов), то используется toString, так что в итоге преобразование происходит к строке. Но если есть valueOf, то используется valueOf. Выше в примере как раз a + b это демонстрируют.

У объектов Date есть и valueOf – возвращает количество миллисекунд, и toString – возвращает строку с датой.

…Но оператор + для Date использует именно toString (хотя должен бы valueOf).

Это и есть исключение:

// бинарный плюс для даты toString, для остальных объектов valueOf

alert( new Date + "" ); // "строка даты"

Других подобных исключений нет.

**Как испугать Java-разработчика**

В языке Java (это не JavaScript, другой язык, здесь приведён для примера) логические значения можно создавать, используя синтаксис new Boolean(true/false), например new Boolean(true).

В JavaScript тоже есть подобная возможность, которая возвращает «объектную обёртку» для логического значения.

Эта возможность давно существует лишь для совместимости, она и не используется на практике, поскольку приводит к странным результатам. Некоторые из них могут сильно удивить человека, не привыкшего к JavaScript, например:

var value = new Boolean(false);

if (value) {

alert( true ); // сработает!

}

Почему запустился alert? Ведь в if находится false… Проверим:

var value = new Boolean(false);

alert( value ); // выводит false, все ок..

if (value) {

alert( true ); // ..но тогда почему выполняется alert в if ?!?

}

Дело в том, что new Boolean – это не примитивное значение, а объект. Поэтому в логическом контексте он преобразуется к true, в результате работает первый пример.

А второй пример вызывает alert, который преобразует объект к строке, и он становится "false".

**В JavaScript вызовы new Boolean/String/Number не используются, а используются простые вызовы соответствующих функций, они преобразуют значение в примитив нужного типа, например Boolean(val) === !!val.**

**[Итого](http://learn.javascript.ru/object-conversion" \l "itogo)**

* В логическом контексте объект – всегда true.
* При строковом преобразовании объекта используется его метод toString. Он должен возвращать примитивное значение, причём не обязательно именно строку.
* Для численного преобразования используется метод valueOf, который также может возвратить любое примитивное значение. У большинства объектов valueOf не работает (возвращает сам объект и потому игнорируется), при этом для численного преобразования используется toString.

Полный алгоритм преобразований есть в спецификации ECMAScript, смотрите пункты [11.8.5](http://es5.github.com/x11.html#x11.8.5), [11.9.3](http://es5.github.com/x11.html#x11.9.3), а также [9.1](http://es5.github.com/x9.html#x9.1) и [9.3](http://es5.github.com/x9.html#x9.3).

Заметим, для полноты картины, что некоторые тесты знаний в интернет предлагают вопросы типа:

{}[0] // чему равно?

{} + {} // а так?

Если вы запустите эти выражения в консоли, то результат может показаться странным. Подвох здесь в том, что если фигурные скобки {...} идут не в выражении, а в основном потоке кода, то JavaScript считает, что это не объект, а «блок кода» (как if, for, но без оператора просто группировка команд вместе используется редко).

Вот блок кода с командой:

{

alert("Блок")

}

А если команду изъять, то будет пустой блок {}, который ничего не делает. Два примера выше как раз содержат пустой блок в начале, который ничего не делает. Иначе говоря:

{}[0] // то же что и: [0]

{} + {} // то же что и: + {}

То есть, такие вопросы – не на преобразование типов, а на понимание, что если { ... } находится вне выражений, то это не объект, а блок.