**Дата и Время**

Для работы с датой и временем в JavaScript используются объекты [Date](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Date/).

**[Создание](http://learn.javascript.ru/datetime" \l "sozdanie)**

Для создания нового объекта типа Date используется один из синтаксисов:

**new Date()**

Создает объект Date с текущей датой и временем:

var now = new Date();

alert( now );

**new Date(milliseconds)**

Создает объект Date, значение которого равно количеству миллисекунд (1/1000 секунды), прошедших с 1 января 1970 года GMT+0.

// 24 часа после 01.01.1970 GMT+0

var Jan02\_1970 = new Date(3600 \* 24 \* 1000);

alert( Jan02\_1970 );

**new Date(datestring)**

Если единственный аргумент – строка, используется вызов Date.parse (см. далее) для чтения даты из неё.

**new Date(year, month, date, hours, minutes, seconds, ms)**

Дату можно создать, используя компоненты в местной временной зоне. Для этого формата обязательны только первые два аргумента. Отсутствующие параметры, начиная с hours считаются равными нулю, а date – единице.

Заметим:

* Год year должен быть из 4 цифр.
* Отсчет месяцев month начинается с нуля 0.

Например:

new Date(2011, 0, 1, 0, 0, 0, 0); // // 1 января 2011, 00:00:00

new Date(2011, 0, 1); // то же самое, часы/секунды по умолчанию равны 0

Дата задана с точностью до миллисекунд:

var date = new Date(2011, 0, 1, 2, 3, 4, 567);

alert( date ); // 1.01.2011, 02:03:04.567

## [Получение компонентов даты](http://learn.javascript.ru/datetime" \l "poluchenie-komponentov-daty)

Для доступа к компонентам даты-времени объекта Date используются следующие методы:

**getFullYear()**

Получить год (из 4 цифр)

**getMonth()**

Получить месяц, **от 0 до 11**.

**getDate()**

Получить число месяца, от 1 до 31.

**getHours(), getMinutes(), getSeconds(), getMilliseconds()**

Получить соответствующие компоненты.

**Не getYear(), а getFullYear()**

Некоторые браузеры реализуют нестандартный метод getYear(). Где-то он возвращает только две цифры из года, где-то четыре. Так или иначе, этот метод отсутствует в стандарте JavaScript. Не используйте его. Для получения года есть getFullYear().

Дополнительно можно получить день недели:

**getDay()**

Получить номер дня в неделе. Неделя в JavaScript начинается с воскресенья, так что результат будет числом **от 0(воскресенье) до 6(суббота)**.

**Все методы, указанные выше, возвращают результат для местной временной зоны.**

Существуют также UTC-варианты этих методов, возвращающие день, месяц, год и т.п. для зоны GMT+0 (UTC): getUTCFullYear(), getUTCMonth(), getUTCDay(). То есть, сразу после "get" вставляется "UTC".

Если ваше локальное время сдвинуто относительно UTC, то следующий код покажет разные часы:

// текущая дата

var date = new Date();

// час в текущей временной зоне

alert( date.getHours() );

// сколько сейчас времени в Лондоне?

// час в зоне GMT+0

alert( date.getUTCHours() );

Кроме описанных выше, существуют два специальных метода без UTC-варианта:

**getTime()**

Возвращает число миллисекунд, прошедших с 1 января 1970 года GMT+0, то есть того же вида, который используется в конструкторе new Date(milliseconds).

**getTimezoneOffset()**

Возвращает разницу между местным и UTC-временем, в минутах.

alert( new Date().getTimezoneOffset() ); // Для GMT-1 выведет 60

## 

## [Установка компонентов даты](http://learn.javascript.ru/datetime#ustanovka-komponentov-daty)

Следующие методы позволяют устанавливать компоненты даты и времени:

* setFullYear(year [, month, date])
* setMonth(month [, date])
* setDate(date)
* setHours(hour [, min, sec, ms])
* setMinutes(min [, sec, ms])
* setSeconds(sec [, ms])
* setMilliseconds(ms)
* setTime(milliseconds) (устанавливает всю дату по миллисекундам с 01.01.1970 UTC)

Все они, кроме setTime(), обладают также UTC-вариантом, например: setUTCHours().

Как видно, некоторые методы могут устанавливать несколько компонентов даты одновременно, в частности, setHours. При этом если какая-то компонента не указана, она не меняется. Например:

var today = new Date;

today.setHours(0);

alert( today ); // сегодня, но час изменён на 0

today.setHours(0, 0, 0, 0);

alert( today ); // сегодня, ровно 00:00:00.

### [Автоисправление даты](http://learn.javascript.ru/datetime" \l "avtoispravlenie-daty)

Автоисправление – очень удобное свойство объектов Date. Оно заключается в том, что можно устанавливать заведомо некорректные компоненты (например 32 января), а объект сам себя поправит.

var d = new Date(2013, 0, 32); // 32 января 2013 ?!?

alert(d); // ... это 1 февраля 2013!

**Неправильные компоненты даты автоматически распределяются по остальным.**

Например, нужно увеличить на 2 дня дату «28 февраля 2011». Может быть так, что это будет 2 марта, а может быть и 1 марта, если год високосный. Но нам обо всем этом думать не нужно. Просто прибавляем два дня. Остальное сделает Date:

var d = new Date(2011, 1, 28);

d.setDate(d.getDate() + 2);

alert( d ); // 2 марта, 2011

Также это используют для получения даты, отдаленной от имеющейся на нужный промежуток времени. Например, получим дату на 70 секунд большую текущей:

var d = new Date();

d.setSeconds(d.getSeconds() + 70);

alert( d ); // выведет корректную дату

Можно установить и нулевые, и даже отрицательные компоненты. Например:

var d = new Date;

d.setDate(1); // поставить первое число месяца

alert( d );

d.setDate(0); // нулевого числа нет, будет последнее число предыдущего месяца

alert( d );

var d = new Date;

d.setDate(-1); // предпоследнее число предыдущего месяца

alert( d );

### [Преобразование к числу, разность дат](http://learn.javascript.ru/datetime" \l "preobrazovanie-k-chislu-raznost-dat)

Когда объект Date используется в числовом контексте, он преобразуется в количество миллисекунд:

alert(+new Date) // +date то же самое, что: +date.valueOf()

**Важный побочный эффект: даты можно вычитать, результат вычитания объектов Date – их временная разница, в миллисекундах**.

Это используют для измерения времени:

var start = new Date; // засекли время

// что-то сделать

for (var i = 0; i < 100000; i++) {

var doSomething = i \* i \* i;

}

var end = new Date; // конец измерения

alert( "Цикл занял " + (end - start) + " ms" );

### [Бенчмаркинг](http://learn.javascript.ru/datetime" \l "benchmarking)

Допустим, у нас есть несколько вариантов решения задачи, каждый описан функцией.

Как узнать, какой быстрее?

Для примера возьмем две функции, которые бегают по массиву:

function walkIn(arr) {

for (var key in arr) arr[key]++

}

function walkLength(arr) {

for (var i = 0; i < arr.length; i++) arr[i]++;

}

Чтобы померить, какая из них быстрее, нельзя запустить один раз walkIn, один раз walkLength и замерить разницу. Одноразовый запуск ненадежен, любая мини-помеха исказит результат.

Для правильного бенчмаркинга функция запускается много раз, чтобы сам тест занял существенное время. Это сведет влияние помех к минимуму. Сложную функцию можно запускать 100 раз, простую – 1000 раз…

Померяем, какая из функций быстрее:

var arr = [];

for (var i = 0; i < 1000; i++) arr[i] = 0;

function walkIn(arr) {

for (var key in arr) arr[key]++;

}

function walkLength(arr) {

for (var i = 0; i < arr.length; i++) arr[i]++;

}

function bench(f) {

var date = new Date();

for (var i = 0; i < 10000; i++) f(arr);

return new Date() - date;

}

alert( 'Время walkIn: ' + bench(walkIn) + 'мс' );

alert( 'Время walkLength: ' + bench(walkLength) + 'мс' );

Теперь представим себе, что во время первого бенчмаркинга bench(walkIn) компьютер что-то делал параллельно важное (вдруг) и это занимало ресурсы, а во время второго – перестал. Реальная ситуация? Конечно реальна, особенно на современных ОС, где много процессов одновременно.

**Гораздо более надёжные результаты можно получить, если весь пакет тестов прогнать много раз.**

var arr = [];

for (var i = 0; i < 1000; i++) arr[i] = 0;

function walkIn(arr) {

for (var key in arr) arr[key]++;

}

function walkLength(arr) {

for (var i = 0; i < arr.length; i++) arr[i]++;

}

function bench(f) {

var date = new Date();

for (var i = 0; i < 1000; i++) f(arr);

return new Date() - date;

}

// bench для каждого теста запустим много раз, чередуя

var timeIn = 0,

timeLength = 0;

for (var i = 0; i < 100; i++) {

timeIn += bench(walkIn);

timeLength += bench(walkLength);

}

alert( 'Время walkIn: ' + timeIn + 'мс' );

alert( 'Время walkLength: ' + timeLength + 'мс' );

**Более точное время с performance.now()**

В современных браузерах (кроме IE9-) вызов [performance.now()](https://developer.mozilla.org/en-US/docs/Web/API/performance.now) возвращает количество миллисекунд, прошедшее с начала загрузки страницы. Причём именно с самого начала, до того, как загрузился HTML-файл, если точнее – с момента выгрузки предыдущей страницы из памяти.

Так что это время включает в себя всё, включая начальное обращение к серверу.

Его можно посмотреть в любом месте страницы, даже в <head>, чтобы узнать, сколько времени потребовалось браузеру, чтобы до него добраться, включая загрузку HTML.

Возвращаемое значение измеряется в миллисекундах, но дополнительно имеет точность 3 знака после запятой (до миллионных долей секунды!), поэтому можно использовать его и для более точного бенчмаркинга в том числе.

**console.time(метка) и console.timeEnd(метка)**

Для измерения с одновременным выводом результатов в консоли есть методы:

* console.time(метка) – включить внутренний хронометр браузера с меткой.
* console.timeEnd(метка) – выключить внутренний хронометр браузера с меткой и вывести результат.

Параметр "метка" используется для идентификации таймера, чтобы можно было делать много замеров одновременно и даже вкладывать измерения друг в друга.

В коде ниже таймеры walkIn, walkLength – конкретные тесты, а таймер «All Benchmarks» – время «на всё про всё»:

var arr = [];

for (var i = 0; i < 1000; i++) arr[i] = 0;

function walkIn(arr) {

for (var key in arr) arr[key]++;

}

function walkLength(arr) {

for (var i = 0; i < arr.length; i++) arr[i]++;

}

function bench(f) {

for (var i = 0; i < 10000; i++) f(arr);

}

console.time("All Benchmarks");

console.time("walkIn");

bench(walkIn);

console.timeEnd("walkIn");

console.time("walkLength");

bench(walkLength);

console.timeEnd("walkLength");

console.timeEnd("All Benchmarks");

**При запуске этого примера нужно открыть консоль, иначе вы ничего не увидите.**

**Внимание, оптимизатор!**

Современные интерпретаторы JavaScript делают массу оптимизаций, например:

1. Автоматически выносят инвариант, то есть постоянное в цикле значение типа arr.length, за пределы цикла.
2. Стараются понять, значения какого типа хранит данная переменная или массив, какую структуру имеет объект и, исходя из этого, оптимизировать внутренние алгоритмы.
3. Выполняют простейшие операции, например сложение явно заданных чисел и строк, на этапе компиляции.
4. Могут обнаружить, что некий код, например присваивание к неиспользуемой локальной переменной, ни на что не влияет и вообще исключить его из выполнения, хотя делают это редко.

Эти оптимизации могут влиять на результаты тестов, поэтому измерять скорость базовых операций JavaScript («проводить микробенчмаркинг») до того, как вы изучите внутренности JavaScript-интерпретаторов и поймёте, что они реально делают на таком коде, не рекомендуется.

## [Форматирование и вывод дат](http://learn.javascript.ru/datetime" \l "formatirovanie-i-vyvod-dat)

Во всех браузерах, кроме IE10-, поддерживается новый стандарт [Ecma 402](http://www.ecma-international.org/publications/standards/Ecma-402.htm), который добавляет специальные методы для форматирования дат.

Это делается вызовом date.toLocaleString(локаль, опции), в котором можно задать много настроек. Он позволяет указать, какие параметры даты нужно вывести, и ряд настроек вывода, после чего интерпретатор сам сформирует строку.

Пример с почти всеми параметрами даты и русским, затем английским (США) форматированием:

var date = new Date(2014, 11, 31, 12, 30, 0);

var options = {

era: 'long',

year: 'numeric',

month: 'long',

day: 'numeric',

weekday: 'long',

timezone: 'UTC',

hour: 'numeric',

minute: 'numeric',

second: 'numeric'

};

alert( date.toLocaleString("ru", options) ); // среда, 31 декабря 2014 г. н.э. 12:30:00

alert( date.toLocaleString("en-US", options) ); // Wednesday, December 31, 2014 Anno Domini 12:30:00 PM

Вы сможете подробно узнать о них в статье [Intl: интернационализация в JavaScript](http://learn.javascript.ru/intl), которая посвящена этому стандарту.

**Методы вывода без локализации:**

toString(), toDateString(), toTimeString() Возвращают стандартное строчное представление, не заданное жёстко в стандарте, а зависящее от браузера. Единственное требование к нему – читаемость человеком. Метод toString возвращает дату целиком, toDateString() и toTimeString() – только дату и время соответственно.

var d = new Date();

alert( d.toString() ); // вывод, похожий на 'Wed Jan 26 2011 16:40:50 GMT+0300'

toUTCString() То же самое, что toString(), но дата в зоне UTC.

toISOString() Возвращает дату в формате ISO Детали формата будут далее. Поддерживается современными браузерами, не поддерживается IE8-.

var d = new Date();

alert( d.toISOString() ); // вывод, похожий на '2011-01-26T13:51:50.417Z'

Если хочется иметь большую гибкость и кросс-браузерность, то также можно воспользоваться специальной библиотекой, например [Moment.JS](http://momentjs.com/) или написать свою функцию форматирования.

**[Разбор строки, Date.parse](http://learn.javascript.ru/datetime" \l "razbor-stroki-date-parse)**

Все современные браузеры, включая IE9+, понимают даты в упрощённом формате ISO 8601 Extended.

Этот формат выглядит так: YYYY-MM-DDTHH:mm:ss.sssZ, где:

* YYYY-MM-DD – дата в формате год-месяц-день.
* Обычный символ T используется как разделитель.
* HH:mm:ss.sss – время: часы-минуты-секунды-миллисекунды.
* Часть 'Z' обозначает временную зону – в формате +-hh:mm, либо символ Z, обозначающий UTC. По стандарту её можно не указывать, тогда UTC, но в Safari с этим ошибка, так что лучше указывать всегда.

Также возможны укороченные варианты, например YYYY-MM-DD или YYYY-MM или даже только YYYY.

Метод Date.parse(str) разбирает строку str в таком формате и возвращает соответствующее ей количество миллисекунд. Если это невозможно, Date.parse возвращает NaN.

Например:

var msUTC = Date.parse('2012-01-26T13:51:50.417Z'); // зона UTC

alert( msUTC ); // 1327571510417 (число миллисекунд)

С таймзоной -07:00 GMT:

var ms = Date.parse('2012-01-26T13:51:50.417-07:00');

alert( ms ); // 1327611110417 (число миллисекунд)

**Формат дат для IE8-**

До появления спецификации ECMAScript 5 формат не был стандартизован, и браузеры, включая IE8-, имели свои собственные форматы дат. Частично, эти форматы пересекаются.

Например, код ниже работает везде, включая старые IE:

var ms = Date.parse("January 26, 2011 13:51:50");

alert( ms );

Вы также можете почитать о старых форматах IE в документации к методу [MSDN Date.parse](http://msdn.microsoft.com/en-us/library/k4w173wk%28v=vs.85%29.aspx).

Конечно же, сейчас лучше использовать современный формат. Если же нужна поддержка IE8-, то метод Date.parse, как и ряд других современных методов, добавляется библиотекой [es5-shim](https://github.com/kriskowal/es5-shim).

## [Метод Date.now()](http://learn.javascript.ru/datetime" \l "metod-date-now)

Метод Date.now() возвращает дату сразу в виде миллисекунд.

Технически, он аналогичен вызову +new Date(), но в отличие от него не создаёт промежуточный объект даты, а поэтому – во много раз быстрее.

Его использование особенно рекомендуется там, где производительность при работе с датами критична. Обычно это не на веб-страницах, а, к примеру, в разработке игр на JavaScript.

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

* Дата и время представлены в JavaScript одним объектом: [Date](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Date/). Создать «только время» при этом нельзя, оно должно быть с датой. Список методов Date вы можете найти в справочнике [Date](http://javascript.ru/Date) или выше.
* Отсчёт месяцев начинается с нуля.
* Отсчёт дней недели (для getDay()) тоже начинается с нуля (и это воскресенье).
* Объект Date удобен тем, что автокорректируется. Благодаря этому легко сдвигать даты.
* При преобразовании к числу объект Date даёт количество миллисекунд, прошедших с 1 января 1970 UTC. Побочное следствие – даты можно вычитать, результатом будет разница в миллисекундах.
* Для получения текущей даты в миллисекундах лучше использовать Date.now(), чтобы не создавать лишний объект Date (кроме IE8-)
* Для бенчмаркинга лучше использовать performance.now() (кроме IE9-), он в 1000 раз точнее.