Sep 22 · 7 min read

API командной строки(консоли): Справка комманд

Command Line API Reference—оригинал статьи.

API консоли содержит коллекцию полезных функций для выполнения простых задач: выделение и испектирование DOM элементов, вывод данных в читабельном формате, остановка и запуск профайлера, и мониторинг DOM событий.

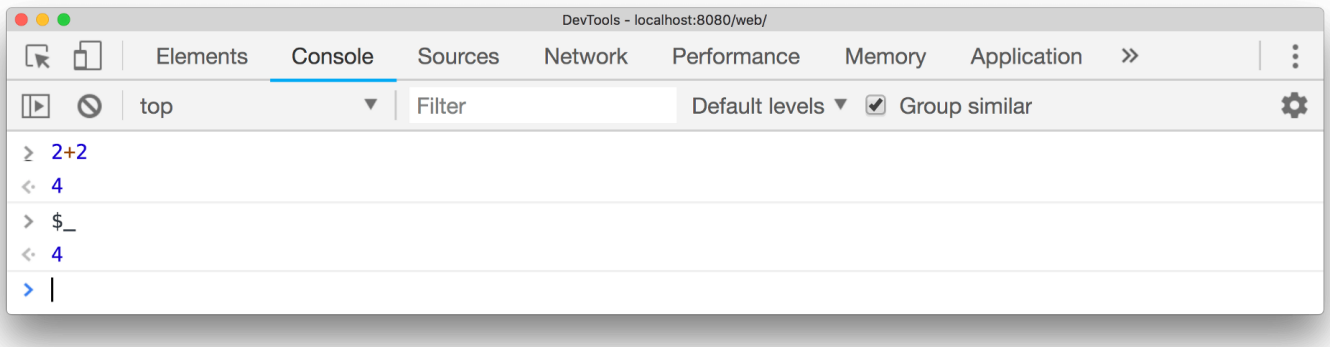
> *Это API доступно только из консоли. Вы не сможете получитить доступ к нему из скриптов страницы.*

> *Если вы ищите функции которые работают с консолью (функции которые начинаются с* `console.*` *), ознакомьтесь с Console API.*

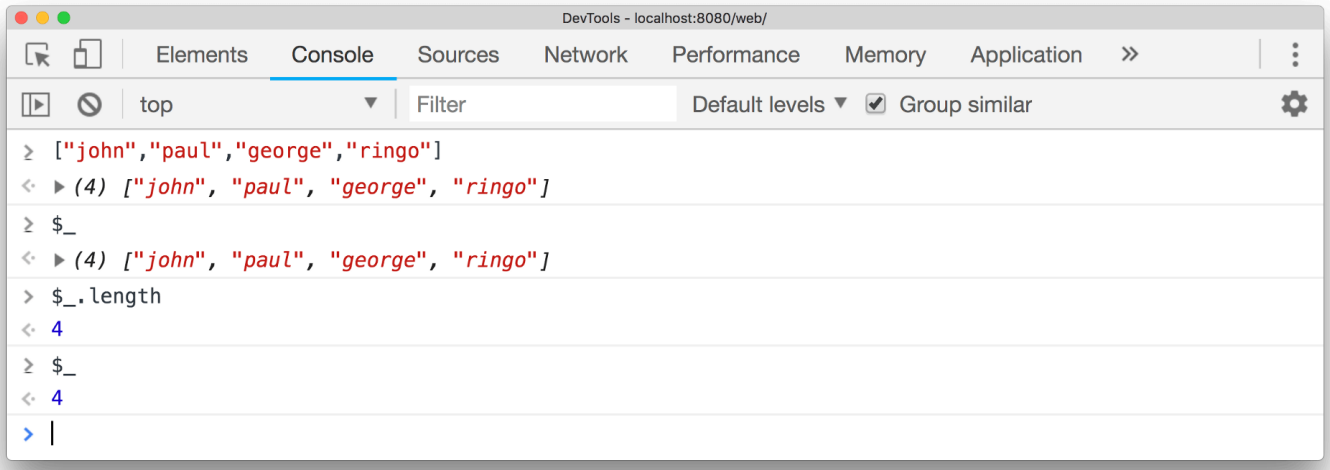
$_

`$_` возвращает последнее выполненное выражение.

В следующем примере, выполняется простое выражение (`2 + 2`). `$_` свойство содержит результат этого выражения.



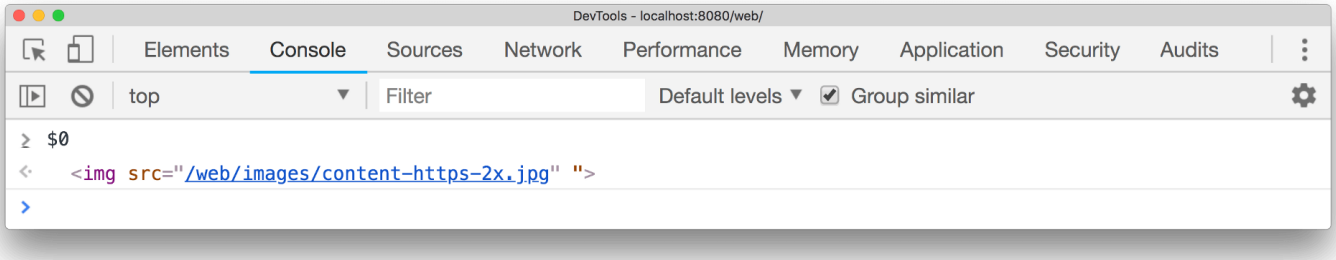
В следующем примере, в консоль вводится массив имен. Выполнение `$_.length` для нахождения длины массива, приведет к изменению содержимого `$_` , что в итоге будет равно 4:



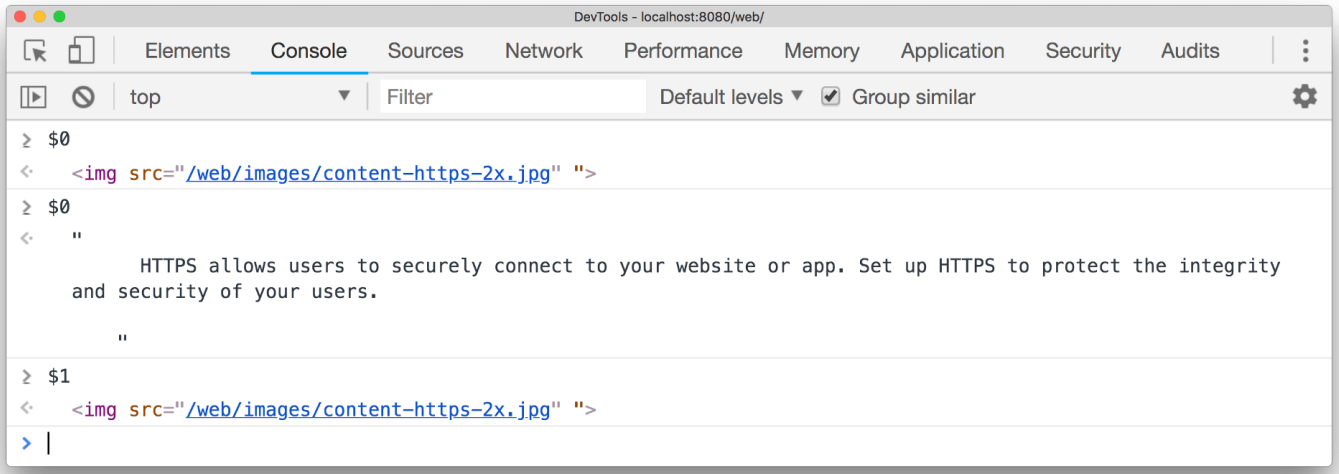
$0 — $4

Команды `$0` , `$1` , `$2` , `$3` и `$4` содержат последние 5 проинспектированных DOM элементов через панель Elements или последние 5 JavaScript обектов выбранные в панели Profiles. `$0` возвращает самый последний выделенный элемент или JavaScript объект, `$1` возвращает предыдущий выделенный элемент и так далее.

В следующем примере, выделенный через панель Elements тег `img` , `$0` содержит ссылку на тег `img` так как это последний выбранный элемент:



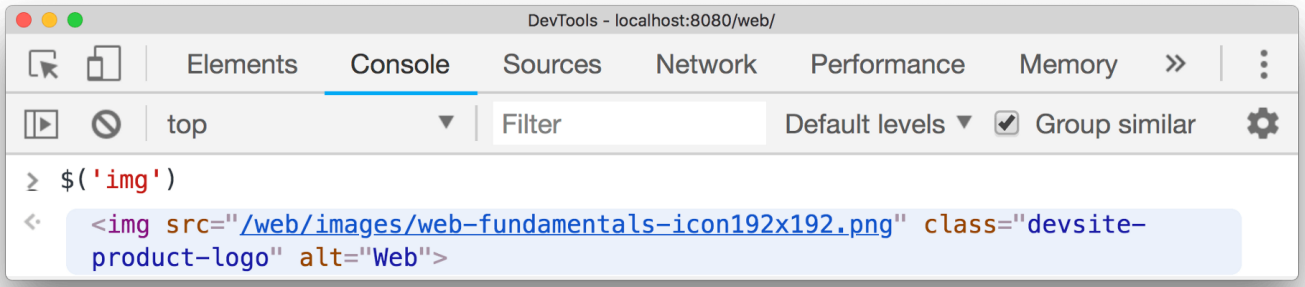
На изображении ниже показан другой элемент выбранный на той же странице. Теперь `$0` возвращает недавно выбранный элемент, а `$1` возвращает предыдущий выбранный элемент, то есть тег `img` .



$(selector, [startNode])

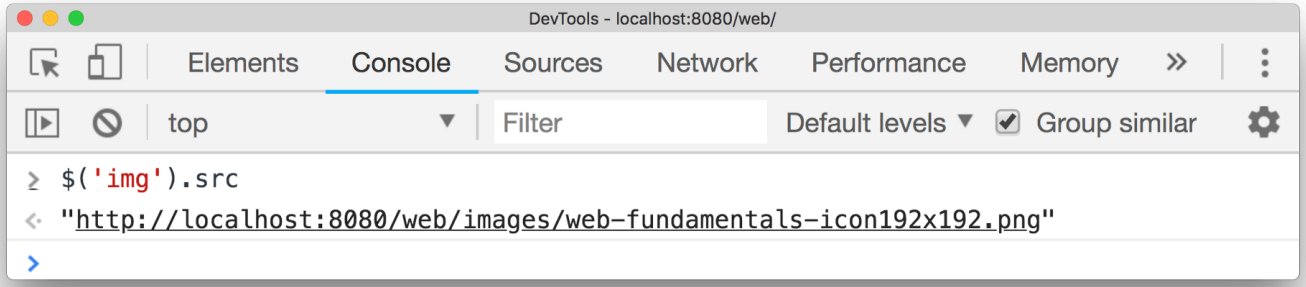
`$(selector)` возвращает первый DOM элемент подходящий под переданный CSS селектор. Эта функция алиас для document.querySelector().

В следеющем примере возвращается ссылка на первый элемент `img` в странице:



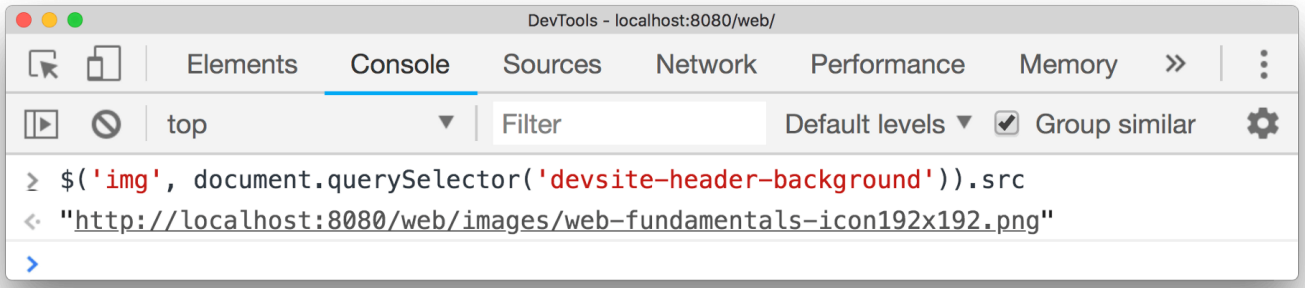
Клик правой кнопки мыши по результату и выбирете “Reveal in Elements Panel”, чтобы найти элемент в DOM или “Scroll in to View”, чтобы перейти к этому элементу на странице.

Следующий пример, возвращает ссылку на выбранный элемент и выводит значение его `src` атрибута:



Эта функция также принимает второй параметр, `startNode` , элемент или узел(node) в котором нужно искать элемент. По умолчанию это параметр равен `document` .

В следующем примере возвращается ссылка на первый элемент выбранного, в данный момент, узла и выводит его `src` :



Переводчик: во втором параметре по идее должен быть класс **.devsite-header-background**. Возможно опечатались, хотя консоль не поругалась

> *Если вы используете такую библиотеку как Jquery, которая использует знак* `$` *, эта функциональность переписывается и*

> *будет использоваться реализация библиотеки*

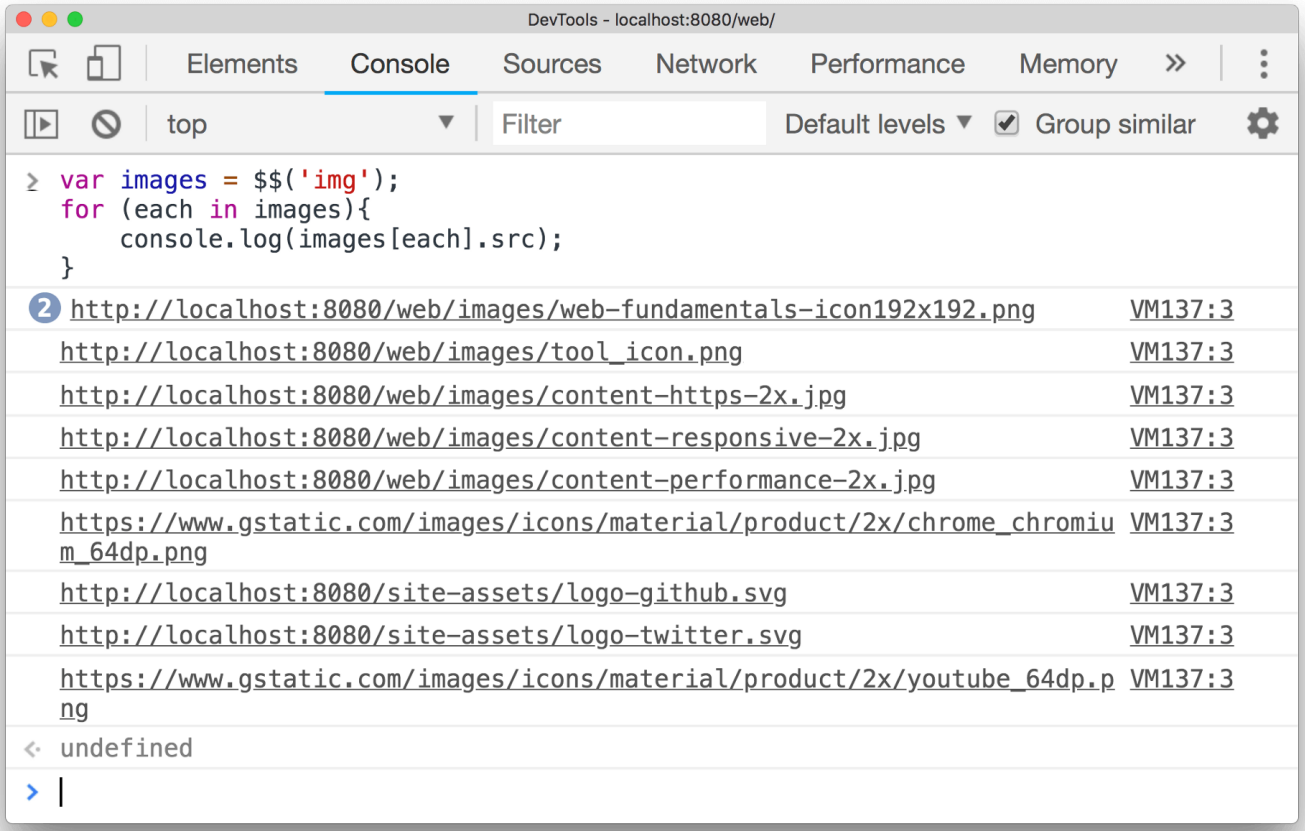
$$(selector, [startNode])

`$$(selector)` возвращает массив всех элементов подходящих под CSS селектор. Эта команда эквивалента document.querySelectorAll().

Следующий пример использует `$$()` для получения массива всех `<img>` элементов на странице и выводит значение атрибута `src` каждого элемента:

```
var images = $$('img');

for (each in images) {
  console.log(images[each].src);
}
```



`$$()` так же принимает параметр `startNode` и работает точно так же как и предыдущая функция.

Далее вы видете измененную версию предыдущего примера, здесь мы используем `$$()` для получения массива элементов в переданном узле:

```
var images = $$(
  'img',
  document.querySelector('.devsite-header-background')
);
```

```
for (each in images) {
  console.log(images[each].src);
}
```

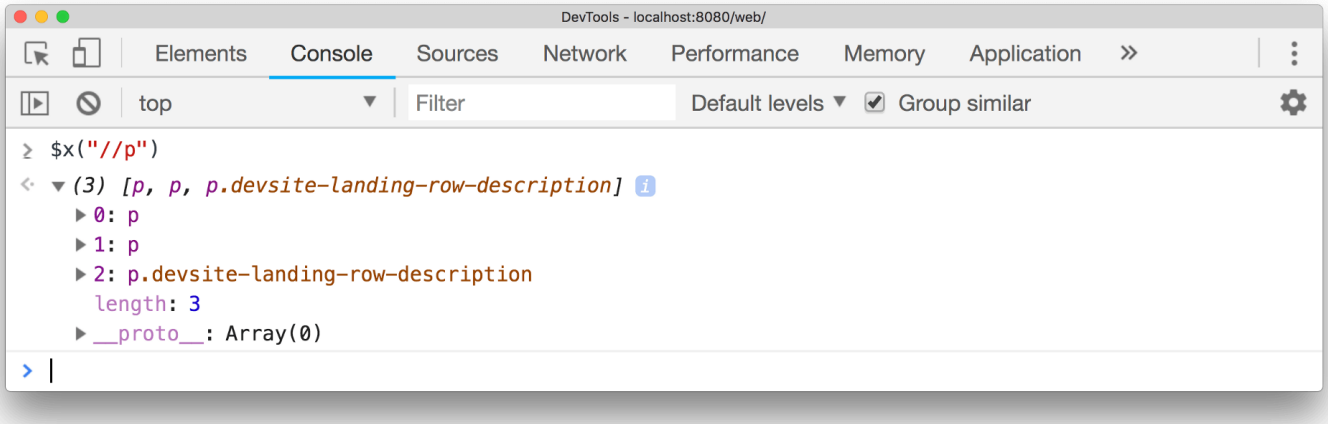
> *Используйте* `Shift + Enter` *в консоле, чтобы перейти на новую строку без выполнения написанного*

$x(path, [startNode])

`$x(path)` вернет массив DOM элементов которые соответвуют XPath выражению.

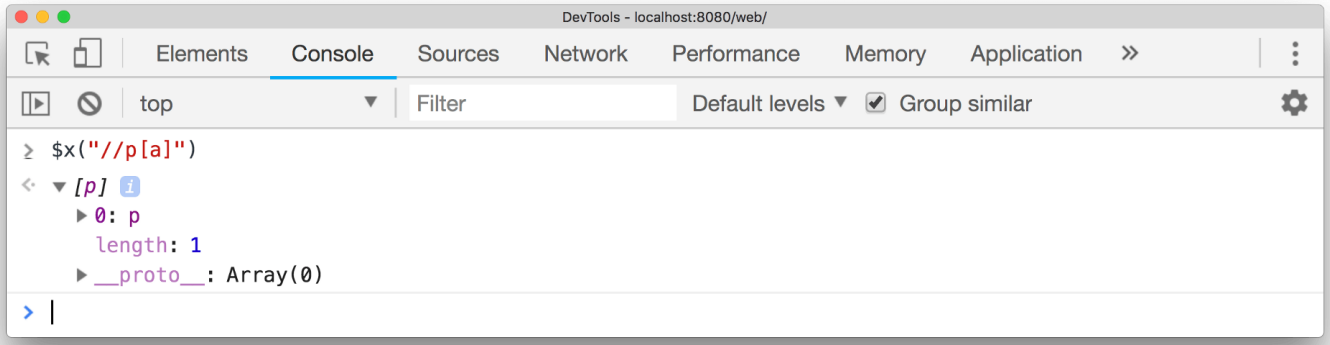
Следующий пример вовращает все теги `<p>` на странице:

```
$x("//p")
```

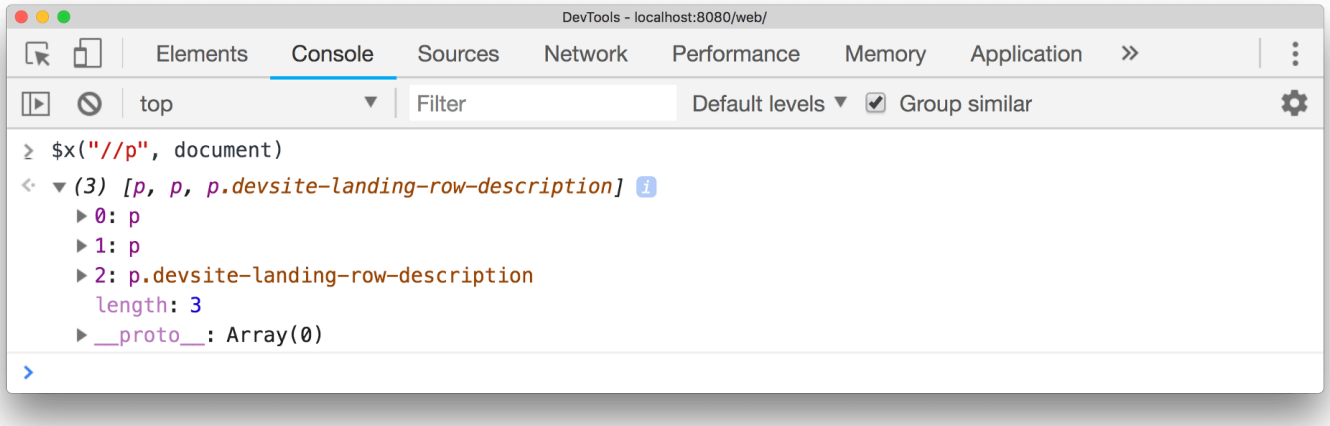


Следующий пример вернет все `<p>` у которых имеется дочерний элемент `<a>`:

```
$x("//p[a]")
```



Так же как и у других функций селекторов, `$x(path)` принимает второй параметр, `startNode`, который указывает в каком узле искать элементы.



clear()

`clear()` просто очищает консоль

```
clear();
```

copy(object)

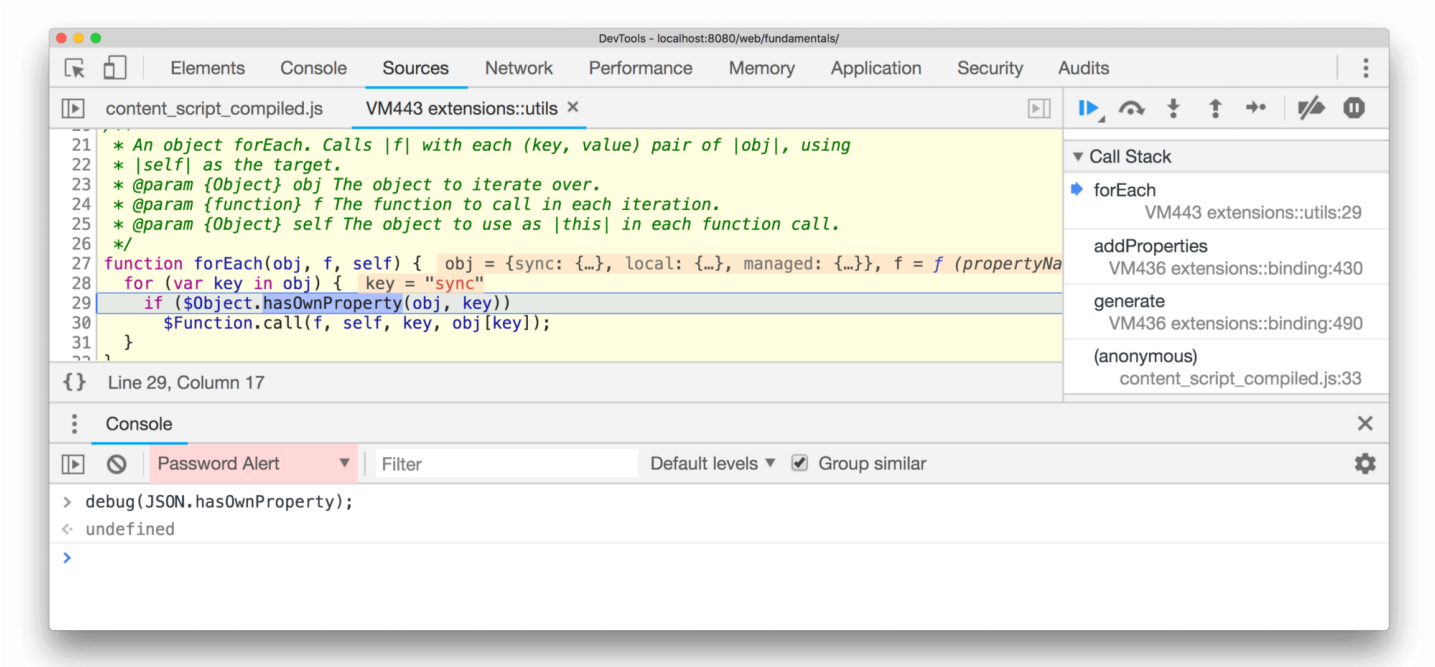
`copy(object)` копирует переданный объект как строку в буфер обмена

```
copy($0);
```

debug(function)

После вызова функции `debug` с переданным значем, открывается панель Sources и начинается режим отладки

```
debug(getData);
```



Используйте `undebug(fn)` , чтобы убрать breakpoint или используйте интерфейс отладчика для удаления breakpoint`ов.

Если хотите больше узнать о breakpoint`ax, то читайте Отладка вашего кода с помощью Breakpoint`ов

dir(object)

`dir(object)` выводит объектно-стилизованный лист всех свойств переданного объекта. `dir(object)` алиас метода `console.dir()`

Следующий пример показывает разницу между выполнением `document.body` в строке консоли и использованием функции `dir()` для вывода того же элемента

```
document.body;
dir(document.body);
```

Больше информации можете узнать по ссылке `console.dir()` из Console API

dirxml(object)

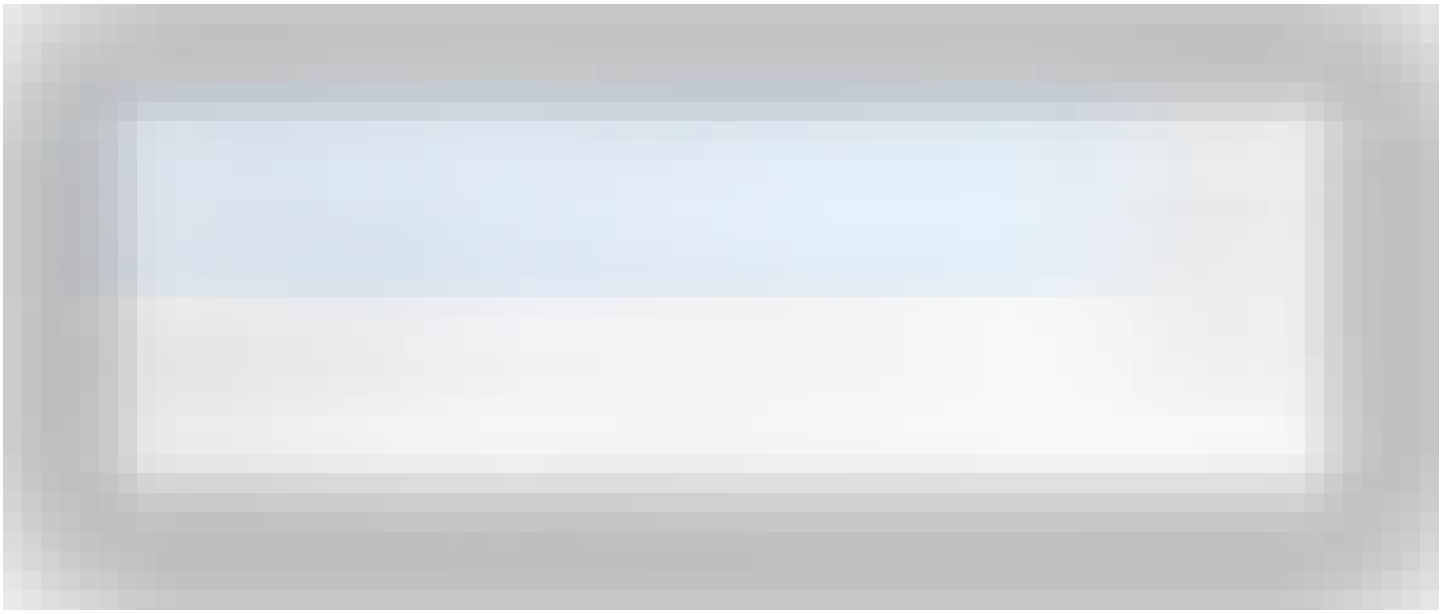
`dirxml(object)` выводит XML стилизованый объект, как это выглядит в панели Elements. Этот метод алиас для console.dirxml()

inspect(object/function)

`inspect(object/function)` открывает и выделяет переданный элемент или обект в необходимой панели: если это DOM элемент, то откроется панель Elements или если это JavaScript объект, то откроется панель Profiles.

В следующем примере в параметр функции передается `document.body` и открывается панель Elements:

```
inspect(document.body);
```

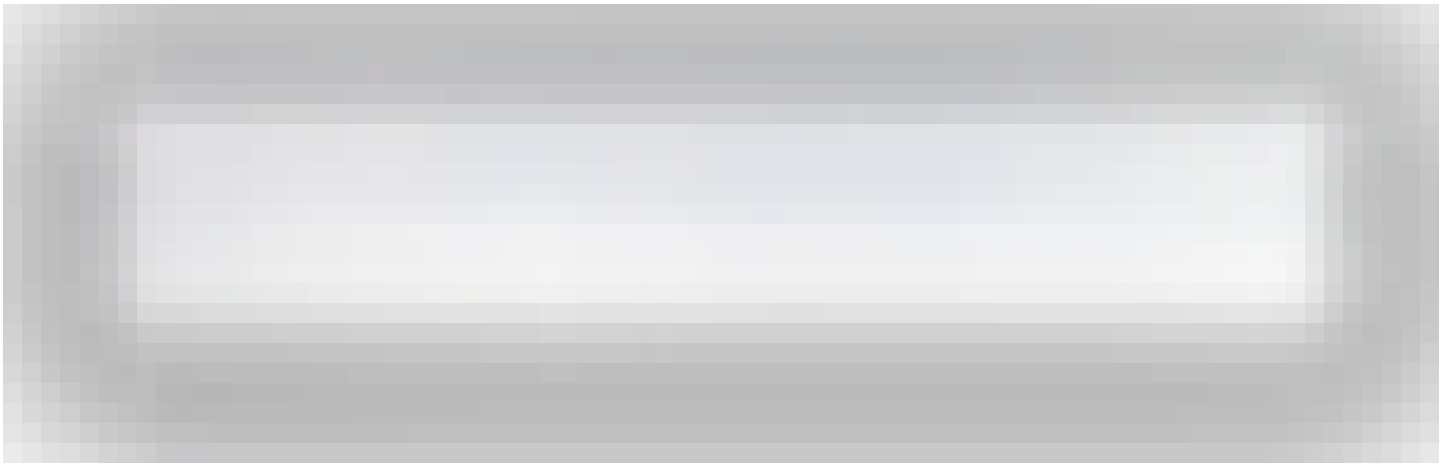


При передаче функции в inspect , функция открывает документ в панели Sources

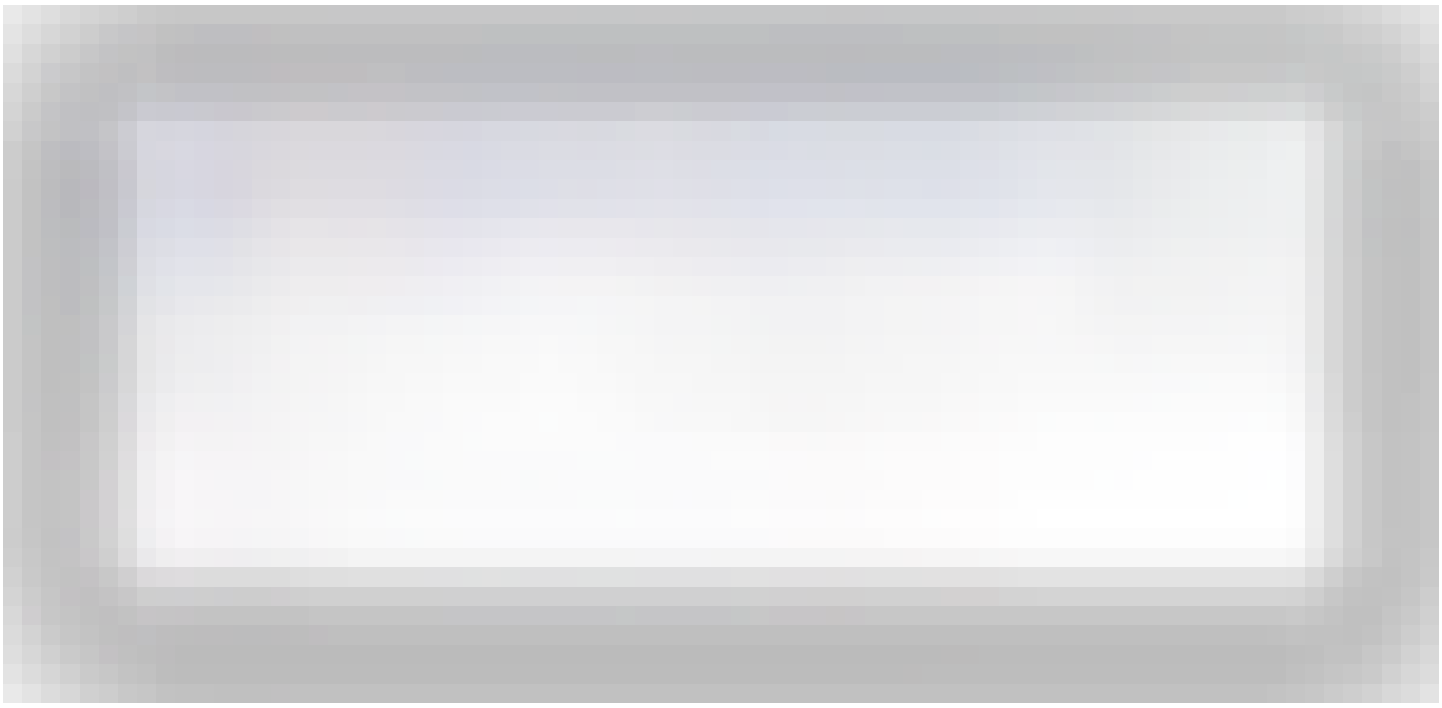
getEventListeners(object)

`getEventListeners(object)` возвращает слушателей событий для переданного объекта. В возвращенном объекте ключами являются типы событий (например, `click` или `keydown`) , а в их значениях содержится массив функций-слушателей. Элементы каждого массива это объекты в которых описаны слушатели. Например, в следующем списке перечислены все зарегистрированные слушатели документа(страницы):

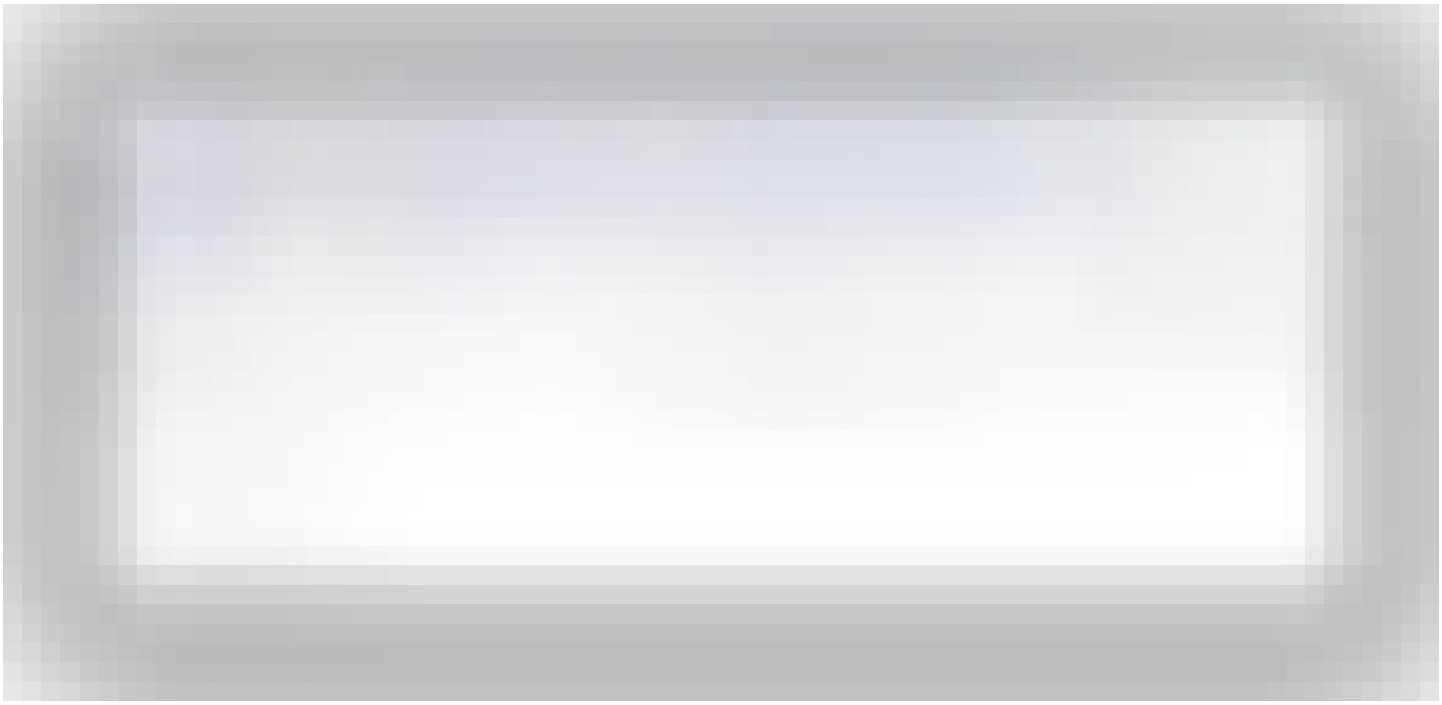
```
getEventListeners(document);
```



Если в переданном объекте, слушателей больше чем один, тогда в значении содержится массив из слушателей. В следующем примере, для типа `click` зарегистрировано два слушателя:



Вы можете просматривать каждый объект и их свойства:



keys(object)

`keys(object)` возвращает массив содержащий ключи переданного в него объекта. Чтобы получить массив значений объекта, используйте `values()` .

Например, предположим у нас есть следующий объект:

```
var player1 = {
  "name": "Ted",
  "level": 42
}
```

Если предположить что `player1` был объявлен в глобальной области выдимости, то вызвав `keys(player1)` и `values(player1)` в консоли, мы получим:

monitor(function)

При выполнении переданной функции выводится сообщение с названием функции и аргументами, которые были переданы

```
function sum(x, y) {
  return x + y;
}
```

```
monitor(sum);
```

Ипользуйте `unmonitor(function)` для прекращения мониторинга

monitorEvents(object[, events])

Когда указанное событие происходит на переданном элементе, в консоли выводится обект события. Вы можете передать: одно событие , массив событий или передать название общего события, которое предоставляет коллекцию более частных

событий (например, “key” и в консоль будет выводится все события связанные с клавишами). Смотрите примеры ниже.

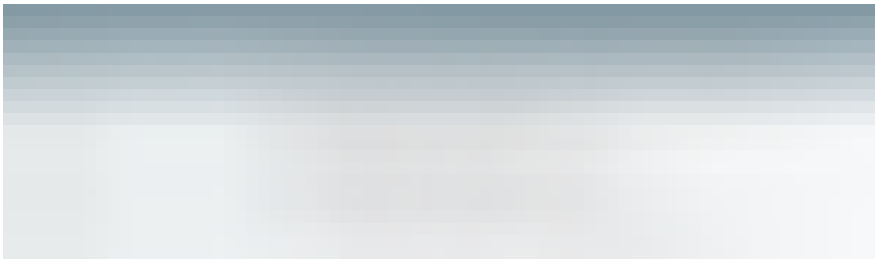
В следующем примере, мониторится событие “resize” для объекта window и каждый раз когда будет происходить событие, объект этого события будет выводиться в консоль.

```
monitorEvents(window, "resize");
```

Следующий код мониторит события “resize” и “scroll” у объета window:

```
monitorEvents(window, ["resize", "scroll"])
```

Ниже можете посмотреть на таблицу **общих** событий:

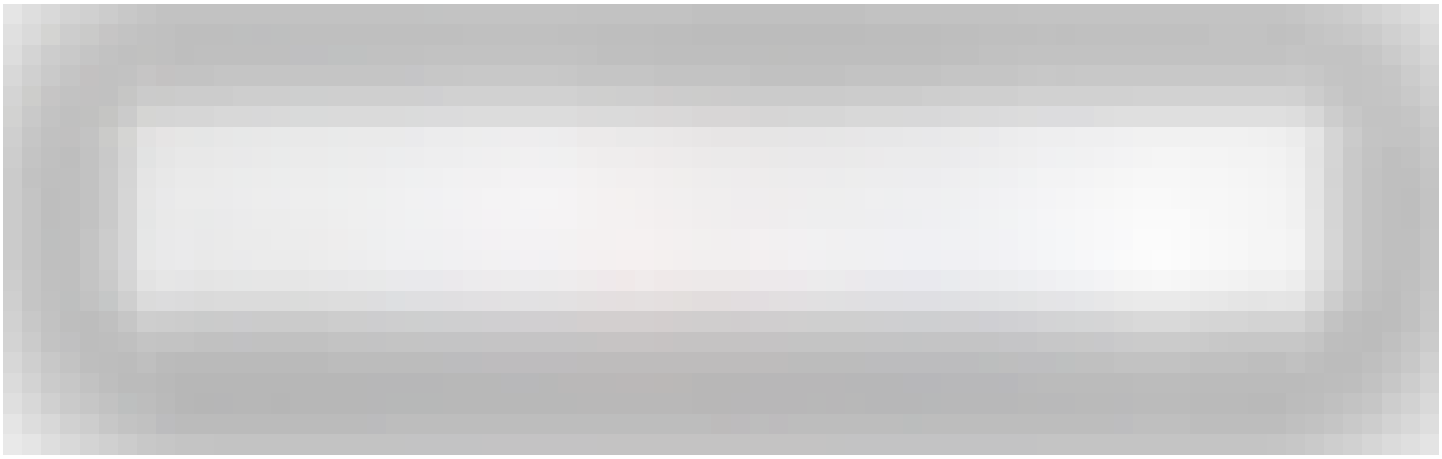


Например, ниже вторым параметром функции мы передаем общий тип события “key”, который соответвует всем событиям

связанным с клавишами, а первым параметром элемент который мы выбрали из Elements панели

```
monitorEvents($0, "key");
```

Ниже приведен пример мониторинга события “key”:



profile([name]) и profileEnd([name])

`profile()` запускает JavaScript CPU ссесию профилирования с переданным именем. `profileEnd()` соответсвенно останавливает профилирование и выводит результат в Profile панель. (Читайте так же Ускорение выполнения JavaScript)

Для старта профилирования:

```
profile("My profile")
```

Для завершения профилирования и вывода результата в Profile панель

```
profileEnd("My profile")
```

Профайлер так же может быть вложенным, следующий код работает в любом порядке:

```
profile('A');
profile('B');
profileEnd('A');
profileEnd('B');
```

Результат в Profile панели:

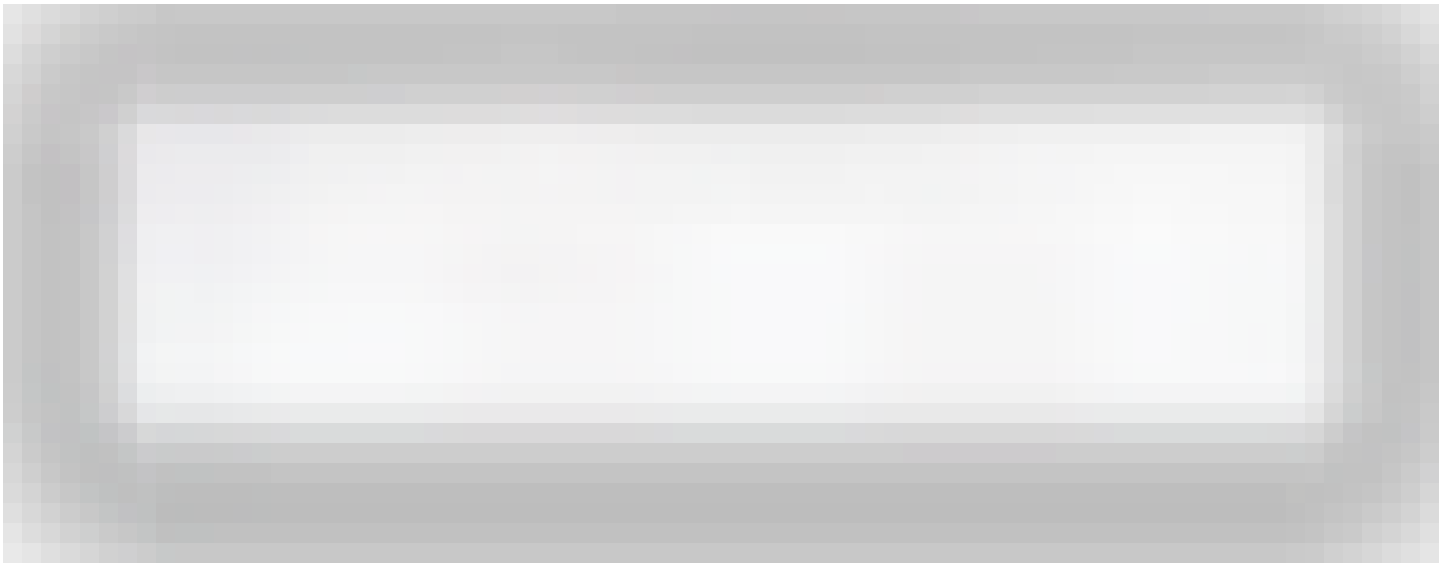
> *Несколько CPU профайлеров могут работать одновременно и вам не надо закрывать их в той же последовательности в какой вы их создавали*

table(data[, columns])

Данные объектов или массивов можно выводит в виде таблицы с помощью передачи первым параметром данных и вторым количества колонок.

```
var names = {
  0: { firstName: "John", lastName: "Smith" },
  1: { firstName: "Jane", lastName: "Doe" }
};

table(names);
```



undebug(function)

`undebug(function)` прекращает режим отладки для переданной в нее функции:

```
undebug(getData);
```

unmonitor(function)

`unmonitor(function)` прекращает режим мониторинга для переданной функции:

```
unmonitor(getData);
```

unmonitorEvents(object[, events])

`unmonitorEvents(object[, events])` прекращает мониторинг событий для переданного объекта. Например, следующий код прекращает мониториг всех событий для свойства `window`

```
unmonitorEvents(window);
```

Вы так же можете прекратить мониторинг для определенных событий переданного объекта, следующий код запускает мониторинг общего события “mouse” и сразу же прекращает мониторинг события более частного события “mousemove”.

```
monitorEvents($0, "mouse");
unmonitorEvents($0, "mousemove");
```

values(object)

`values(object)` возвращает в виде массива значения переданного объекта:

```
values(object);
```

. . .

Авторы: Andi Smith , Meggin Kearney

Переводчик: **Sam Bulatov**