**[FRONTENDER MAGAZINE](https://frontender.info/)**

**Использование консоли Google Chrome**

Консоль предоставляет две функции для разработчиков для тестирования веб-страниц и приложений:

* в консоль можно выводить отладочную информацию, используя такие методы [Console API](https://developers.google.com/chrome-developer-tools/docs/console-api), как [console.log()](https://developers.google.com/chrome-developer-tools/docs/console-api#consolelogobject_object) и [console.profile()](https://developers.google.com/chrome-developer-tools/docs/console-api#consoleprofilelabel).
* оболочка для работы в консоли, в которой вы можете вводить команды, в частности используя автодополнение, и оперировать document и Chrome DevTools. Вы можете выполнять JavaScript-выражения прямо в консоли и использовать методы [Command Line API](https://developers.google.com/chrome-developer-tools/docs/commandline-api), например, [$()](https://developers.google.com/chrome-developer-tools/docs/commandline-api#selector) для создания выборки элементов, или [profile()](https://developers.google.com/chrome-developer-tools/docs/commandline-api#profile) для запуска CPU-профайлера.

В статье мы рассмотрим наиболее распространённые способы использования [Console API](https://developers.google.com/chrome-developer-tools/docs/console-api) и [Command Line API](https://developers.google.com/chrome-developer-tools/docs/commandline-api). Более подробно о них вы сможете узнать в документации.

* [Базовые операции](https://frontender.info/chrome-developer-tools/#--1)
  + [Открытие консоли](https://frontender.info/chrome-developer-tools/#--2)
  + [Очистка консоли](https://frontender.info/chrome-developer-tools/#--3)
  + [Настройки консоли](https://frontender.info/chrome-developer-tools/#--4)
* [Использование API консоли](https://frontender.info/chrome-developer-tools/#-api-)
  + [Вывод в консоль](https://frontender.info/chrome-developer-tools/#--)
  + [Ошибки и предупреждения](https://frontender.info/chrome-developer-tools/#---1)
  + [Проверки](https://frontender.info/chrome-developer-tools/#assertions)
  + [Фильтрация вывода в консоли](https://frontender.info/chrome-developer-tools/#---)
  + [Группирование вывода](https://frontender.info/chrome-developer-tools/#--5)
  + [Замена строк и их форматирование](https://frontender.info/chrome-developer-tools/#----)
  + [Представление DOM-элементов как JavaScript-объекты](https://frontender.info/chrome-developer-tools/#-dom---javascript-)
  + [Стилизация вывода консоли с помощью CSS](https://frontender.info/chrome-developer-tools/#-----css)
  + [Измерение временных затрат](https://frontender.info/chrome-developer-tools/#---2)
  + [Корреляция с панелью Timeline](https://frontender.info/chrome-developer-tools/#---timeline)
  + [Создание точек останова](https://frontender.info/chrome-developer-tools/#---3)
* [Использование the Command Line API](https://frontender.info/chrome-developer-tools/#-command-line-api)
  + [Выполнение выражений](https://frontender.info/chrome-developer-tools/#--6)
  + [Выделение элементов](https://frontender.info/chrome-developer-tools/#--7)
  + [Инспектирование DOM-элементов и JavaScript объектов](https://frontender.info/chrome-developer-tools/#-dom---javascript--1)
  + [Доступ к недавним элементам или объектам](https://frontender.info/chrome-developer-tools/#-----)
  + [Отслеживание событий](https://frontender.info/chrome-developer-tools/#--8)
  + [Контроль за CPU-профайлером](https://frontender.info/chrome-developer-tools/#--cpu-)

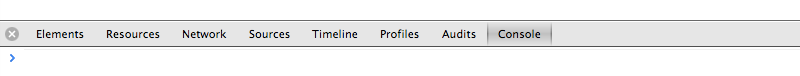
**Базовые операции**

**Открытие консоли**

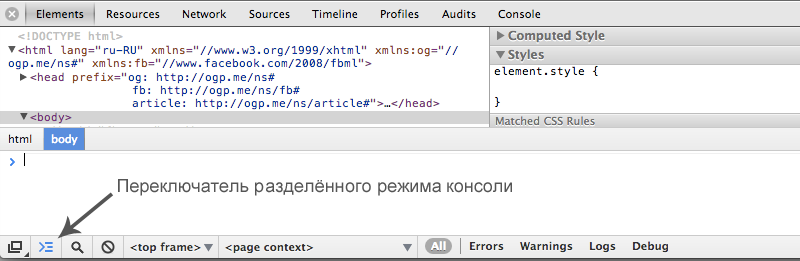
Консоль в Chrome DevTools доступна в двух вариантах: вкладка Console и в виде разделённой версии, доступной из любой другой вкладки.

Для того, чтобы открыть вкладку Console вы можете:

* использовать хоткей **Command - Option - J** (Mac) или **Control - Shift - J** (Windows/Linux);
* выбрать пункт меню **View > Developer > JavaScript Console**.



Для того, чтобы вызвать или скрыть разделённую версию консоли в других вкладках, нажмите клавишу **Esc** или кликните на иконку **Show/Hide Console** в нижнем левом углу DevTools. Скриншот показывает разделённый вариант консоли во вкладке Elements.



**Очистка консоли**

Для очистки консоли:

* через контекстное меню вкладки консоли (клик правой кнопки мыши) выберите пункт **Clear Console**.
* Введите [clear()](https://developers.google.com/chrome-developer-tools/docs/commandline-api#clear) — команду из Command Line API в консоли.
* Вызовите [console.clear()](https://developers.google.com/chrome-developer-tools/docs/console-api#consoleclear) (команду из Console API) из скрипта.
* Используйте хоткеи **⌘K** или **⌃L** (Mac) **Control - L** (Windows и Linux).

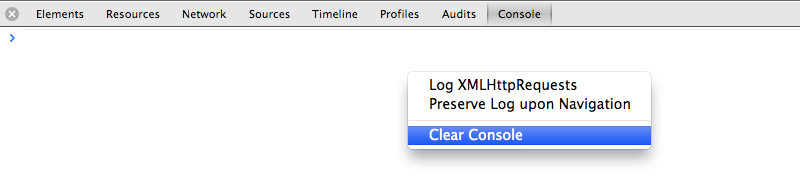
По умолчанию, история консоли очищается при переходе на другую страницу. Вы можете отменить очистку включив **Preserve log upon navigation** в разделе консоли в настройках DevTools (см. [настройки Консоли](https://developers.google.com/chrome-developer-tools/docs/console#consolepreferences)).

**Настройки консоли**

Консоль имеет две главные опции, которые вы можете настраивать в главной вкладке настроек DevTools:

* **Log XMLHTTPRequests** — определяет, логировать ли XMLHTTPRequest в панели консоли.
* **Preserve log upon navigation** — определяет, сбрасываться ли истории консоли при переходе на новую страницу. По умолчанию обе из этих опций отключены.

Вы можете поменять эти настройки в контекстном меню консоли, кликнув правой кнопкой мыши.



**Использование API консоли**

Console API — набор методов, доступных из объекта console, объявленного из DevTools. Одно из главных назначений API — [логировать информацию](https://developers.google.com/chrome-developer-tools/docs/console#writing_to_the_console) (значение свойства, целый объект или DOM-элемент) в консоль во время работы вашего приложения. Вы также можете группировать вывод, чтобы избавиться от визуального мусора.

**Вывод в консоль**

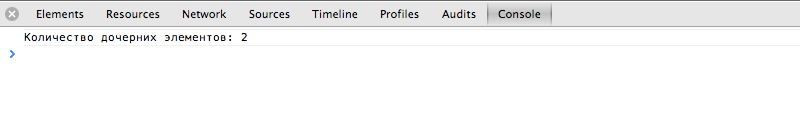
Метод [console.log()](https://developers.google.com/chrome-developer-tools/docs/console-api#consolelogobject_object) принимает один и более параметров и выводит их текущие значения на консоль. Например:

**var** a = document.createElement('p');

a.appendChild(document.createTextNode('foo'));

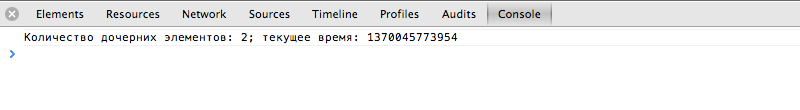
a.appendChild(document.createTextNode('bar'));

console.log("Количество дочерних элементов: " + a.childNodes.length);



Вместо того, чтобы конкатенировать параметры функции с помощью оператора + (как показано выше), вы можете ввести каждый параметр друг за другом через запятую и их значения будут выведены в одну строку, разделённые пробелом.

console.log("Количество дочерних элементов: ", a.childNodes.length, "; текущее время: ", Date.now() );



**Ошибки и предупреждения**

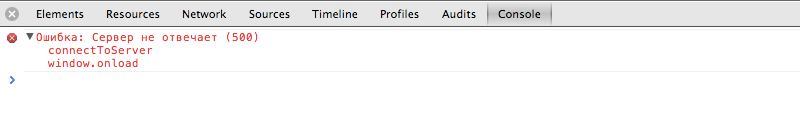
Метод [console.error()](https://developers.google.com/chrome-developer-tools/docs/console-api#consoleerrorobject_object) выводит красную иконку рядом с сообщением красного цвета.

**function** **connectToServer**() {

console.error("Ошибка: %s (%i)", "Сервер не отвечает",500);

}

connectToServer();

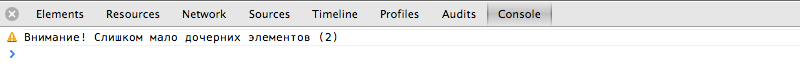


Метод [console.warn()](https://developers.google.com/chrome-developer-tools/docs/console-api#consolewarnobject_object) выводит жёлтую иконку рядом с текстом сообщения.

**if**(a.childNodes.length < 3 ) {

console.warn('Внимание! Слишком мало дочерних элементов (%d)', a.childNodes.length);

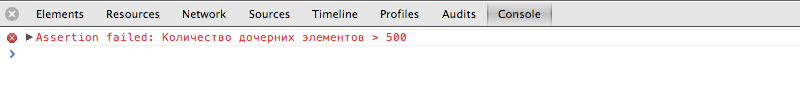
}



**Проверки**

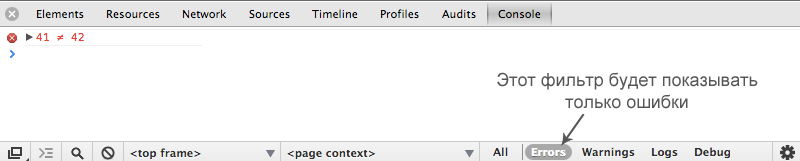
Метод [console.assert()](https://developers.google.com/chrome-developer-tools/docs/console-api#consoleassertexpression_object) выводит сообщение об ошибке (это второй аргумент) только в том случае, если первый аргумент равен false. К примеру, в следующем примере сообщение об ошибке появится, только если количество дочерних элементов DOM-элемента listбольше пятисот.

**console**.assert(**list**.childNodes.length < 500, "Количество дочерних элементов > 500");



**Фильтрация вывода в консоли**

Вы можете быстро фильтровать сообщения в консоли по их типу (уровню) — ошибки, предупреждения и стандартный лог — выбрав один из доступных опций внизу консоли:



Возможные фильтры:

* **All** — без фильтрации.
* **Errors** — сообщения console.error().
* **Warnings** — сообщения console.warn().
* **Logs** — сообщения console.log(), console.info() и console.debug().
* **Debug** — сообщения console.timeEnd() и остальных функций консольного вывода.

**Группирование вывода**

Вы можете визуально группировать вывод в консоли с помощью команд [console.group()](https://developers.google.com/chrome-developer-tools/docs/console-api#consolegroupobject_object) и [groupEnd()](https://developers.google.com/chrome-developer-tools/docs/console-api#consolegroupend).

**var** user = "Вася Пупкин", authenticated = false;

console.group("Этап аутентификации");

console.log("Аутентификация пользователя '%s'", user);

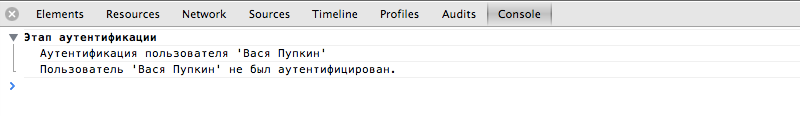
*// Код авторизации…*

**if** (!authenticated) {

console.log("Пользователь '%s' не был аутентифицирован.", user)

}

console.groupEnd();



Также вы можете вкладывать группы логов друг в друга. В следующем примере группа логов создаётся для этапа аутентификации в процессе логина. Если пользователь аутентифицирован, то создаётся вложенная группа для этапа авторизации.

**var** user = "Вася Пупкин", authenticated = true, authorized = true;

*// Внешняя группа*

console.group("Аутентификация пользователя '%s'", user);

**if** (authenticated) {

console.log("Пользователь '%s' был аутентифицирован.", user)

*// Начало вложенной группы*

console.group("Авторизация пользователя '%s'", user);

**if** (authorized) {

console.log("Пользователь '%s' был авторизован.", user)

}

*// Конец вложенной группы*

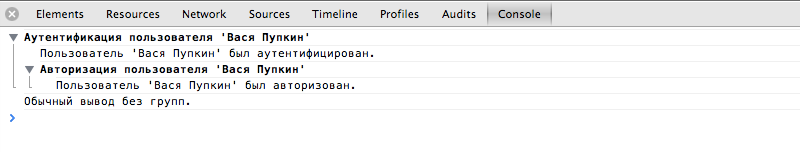
console.groupEnd();

}

*// Конец внешней группы*

console.groupEnd();

console.log("Обычный вывод без групп.");



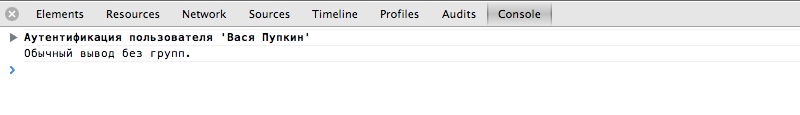
Для создания изначально свёрнутой группы используйте [console.groupCollapsed()](https://developers.google.com/chrome-developer-tools/docs/console-api#consolegroupcollapsed) вместо console.group():

console.groupCollapsed("Аутентификация пользователя '%s'", user);

**if** (authenticated) {

*//…*

}



**Замена строк и их форматирование**

Первый параметр, передаваемый в любой метод консоли (например, log() или error()), может содержать *модификаторы форматирования*. Модификатор вывода состоит из символа %, сразу за которым следует буква, сообщающая о том, какое форматирование должно быть применено (например, %s — для строк). Модификатор форматирования определяет, куда подставить значение, переданное из следующих параметров функции.

В следующем примере используется строчный и числовой модификаторы %s (string) и %d(decimal) для замены значений в выводимой строке.

console.log("%s купил %d бочонков мёда", "Саша", "100");

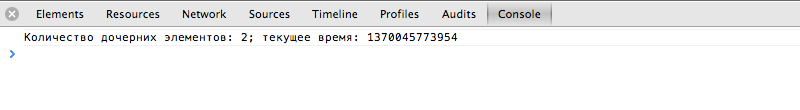
Результатом будет «Саша купил 100 бочонков мёда».

Приведённая таблица содержит поддерживаемые модификаторы форматирования и их значения:

| **Модификатор форматирования** | **Описание** |
| --- | --- |
| %s | Форматирует значение как строку. |
| %d или %i | Форматирует значение как целое число (decimal и integer). |
| %f | Форматирует объект как число с плавающей точкой. |
| %o | Форматирует значение как DOM-элемент (также как в панели Elements). |
| %O | Форматирует значение как JavaScript-объект. |
| %c | Применяет переданные в качестве второго аргумента CSS-стили к выводимой строке. |

В следующем примере модификатор форматирования %d заменяется на значение document.childNodes.length и форматируется как целое число; модификатор %fзаменяется на значение, возвращаемое Date.now() и форматируется как число с плавающей точкой.

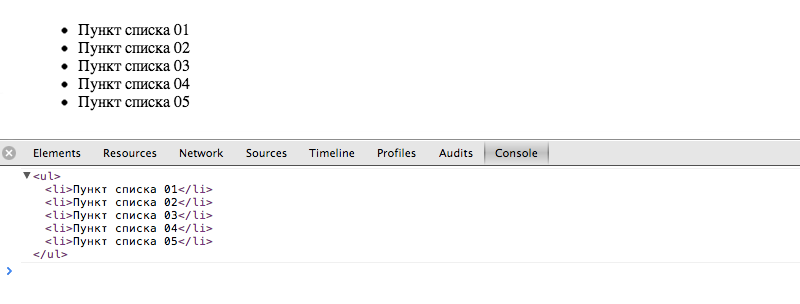
**console**.log("Количество дочерних элементов: %**d**; текущее время: %**f**", **a**.childNodes.length, **Date**.now() );



**Представление DOM-элементов как JavaScript-объекты**

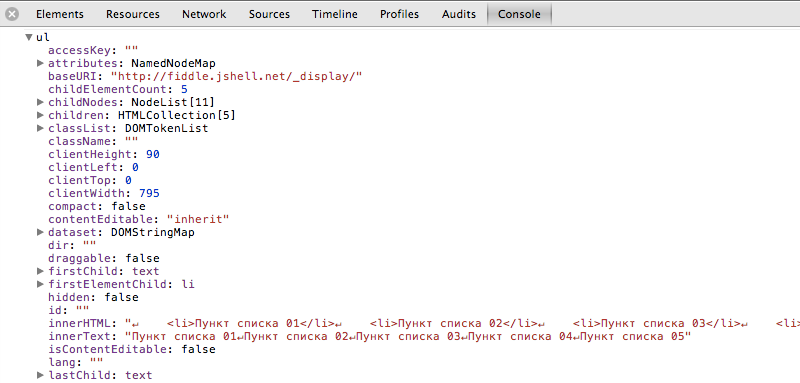
По умолчанию, когда вы логируете DOM-элемент в консоль, он выводится в XML- формате, как в панели Elements:

**console**.log(**document**.body.firstElementChild);



Вы можете вывести DOM-элемент в JavaScript-представлении с помощью метода console.dir():

**console**.dir(**document**.body.firstElementChild);



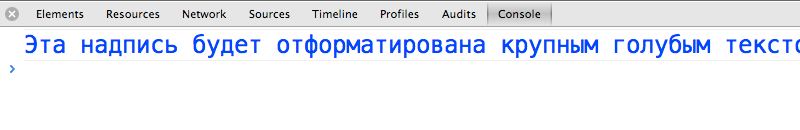
Точно также вы можете использовать [модификатор вывода](https://developers.google.com/chrome-developer-tools/docs/console#string_substitution_and_formatting) %0 в методе console.log():

console.log("%O", document.body.firstElementChild);

**Стилизация вывода консоли с помощью CSS**

Можно использовать модификатор %c, чтобы применить СSS-правила, к любой строке, выводимой с помощью [console.log()](https://developers.google.com/chrome-developer-tools/docs/console#writingtotheconsole) или похожих методов.

console.log("%cЭта надпись будет отформатирована крупным голубым текстом", "color: blue; font-size: x-large");



**Измерение временных затрат**

Методы [console.time()](https://developers.google.com/chrome-developer-tools/docs/console-api#consoletimelabel) и [console.timeEnd()](https://developers.google.com/chrome-developer-tools/docs/console-api#consoletimeendlabel) используются для измерения того, как много времени потребовалось для выполнения скрипта. console.time() вызывается для запуска таймера, а console.timeEnd() — для его остановки. Время, прошедшее между вызовами этих функций, будет выведено в консоль.

console.time("Инициализация массива");

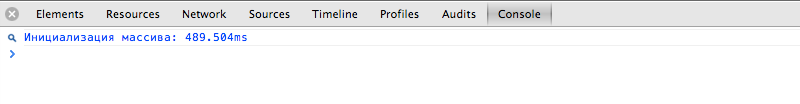
**var** array= **new** Array(1000000);

**for** (**var** i = array.length - 1; i >= 0; i--) {

array[i] = **new** Object();

};

console.timeEnd("Инициализация массива");



Внимание: необходимо использовать одинаковые строки в параметрах console.time() и timeEnd() для ожидаемой работы таймера — считайте эту строку его уникальным идентификатором.

**Корреляция с панелью Timeline**

Панель [Timeline](https://developers.google.com/chrome-developer-tools/docs/timeline) предоставляет подробный обзор того, куда было потрачено время работы вашего приложения. Метод [console.timeStamp()](https://developers.google.com/chrome-developer-tools/docs/console-api#consoletimestamplabel) создаёт отдельную отметку в момент своего исполнения. Это помогает легко и непринуждённо соотносить события в приложении с браузерными событиями *reflow* и *repaint*.

Внимание: метод console.timeStamp() выполняется только при записи событий в панели Timeline.

В следующем примере в панели Timeline появляется отметка «Adding result» в тот момент, когда поток выполнения программы доходит до console.timeStamp("Adding result")

**function** **AddResult**(name, result) {

console.timeStamp("Добавление результатов");

**var** text = name + ': ' + result;

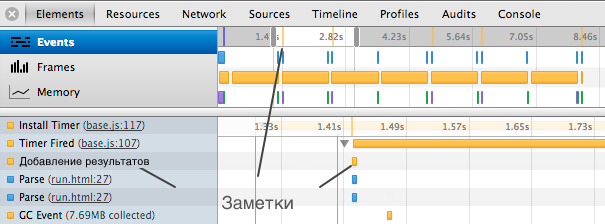
**var** results = document.getElementById("results");

results.innerHTML += (text + "<br>");

}

Как проиллюстрировано в скриншоте, вызов timeStamp() отмечен в следующих местах:

* жёлтая вертикальная линия в панели Timeline.
* Запись добавлена в список записанных событий.



**Создание точек останова**

Вы можете начать отладку вашего кода, вызвав команду [debugger](https://developers.google.com/chrome-developer-tools/docs/console-api#debugger). К примеру, в следующем коде отладка начинается сразу после вызова метода brightness():

**function** **brightness**() {

**debugger**;

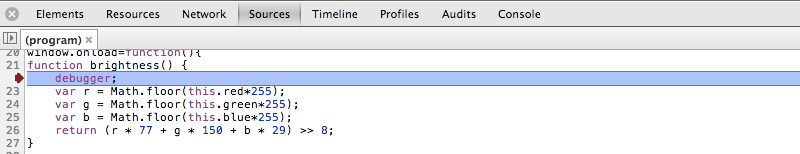
**var** r = Math.floor(**this**.red\*255);

**var** g = Math.floor(**this**.green\*255);

**var** b = Math.floor(**this**.blue\*255);

**return** (r \* 77 + g \* 150 + b \* 29) >> 8;

}



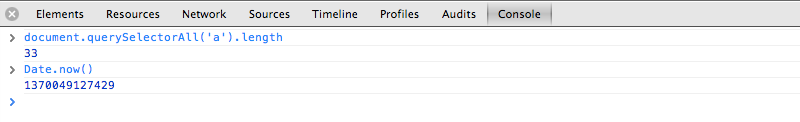
**Использование Command Line API**

Кроме того, что консоль — это место вывода логов вашего приложения, вы можете вводить в нее команды, определенные в [Command Line API](https://developers.google.com/chrome-developer-tools/docs/commandline-api). Это API дает следующие возможности:

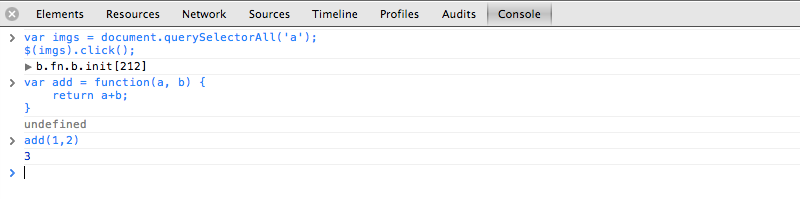
* удобные функции для выделения DOM-элементов.
* Методы управления CPU-профайлером.
* Псевдонимы для некоторых методов Console API.
* Отслеживание событий.
* Просмотр обработчиков событий объекта.

**Выполнение выражений**

Консоль выполнит любой JavaScript-код, который вы в неё введёте, сразу после нажатия кнопки Return или Enter. В консоли действует автодополнение и дополнение по табу. По мере ввода выражений и свойств вам предлагают возможные варианты. Если существуют несколько одинаково начинающихся свойств, вы можете выбирать между ними с помощью кнопки Tab. Нажав стрелку вправо вы выберете текущую подсказку. Если подсказка одна, то нажатие Tab тоже приведет к ее выбору.



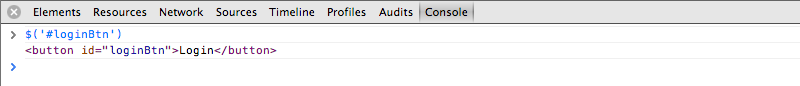
Для того, чтобы вводить многострочные JavaScript-выражения, используйте сочетание клавиш Shift+Enter для перевода строки.



**Выделение элементов**

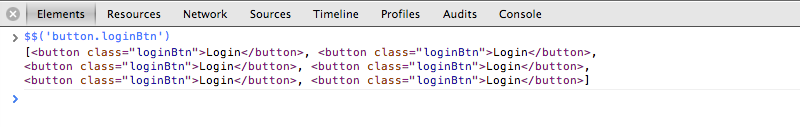
Command Line API предоставляет набор методов для доступа к DOM-элементам в вашем приложении. К примеру, метод [$()](https://developers.google.com/chrome-developer-tools/docs/commandline-api#selector) возвращает первый элемент, соответствующий объявленному CSS-селектору, идентично с [document.querySelector()](http://docs.webplatform.org/wiki/css/selectors_api/querySelector). Следующий код вернёт первый элемент с ID «loginBtn».

$('#loginBtn');



Метод [$$()](https://developers.google.com/chrome-developer-tools/docs/commandline-api#selector_1) возвращает массив элементов, соответствующих указанному CSS- селектору, идентично [document.querySelectorAll()](http://docs.webplatform.org/wiki/css/selectors_api/querySelectorAll). Чтобы получить все кнопки с классом loginBtn, нужно ввести:

$$('button.loginBtn');



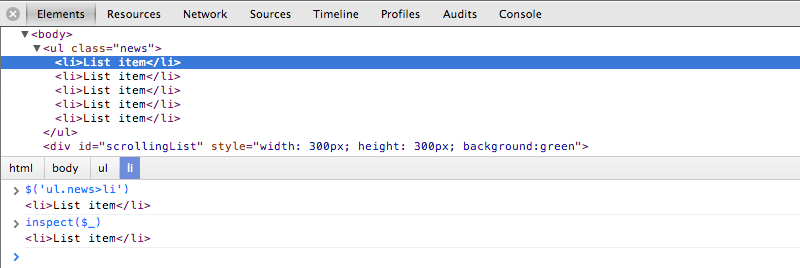
И, наконец, метод [x()](https://developers.google.com/chrome-developer-tools/docs/commandline-api#xpath) принимает XPath-путь в качестве параметра и возвращает массив элементов, соответствующих этому пути. Например, этот код вернёт все элементы <script>, являющиеся дочерними по отношению к элементу <body>:

$x('/html/body/script');

**Инспектирование DOM-элементов и объектов**

Метод [inspect()](https://developers.google.com/chrome-developer-tools/docs/commandline-api#inspectobject) принимает ссылку на DOM-элемент (или объект) в качестве параметра и отображает элемент или объект в соответствующей панели: DOM-элемент в панели Elements и JavaScript-объект в панели Profile.

К примеру, в следующем скриншоте функция $() использована, чтобы получить ссылку на элемент <li>. Затем последнее исполненное выражение ([$\_](https://developers.google.com/chrome-developer-tools/docs/commandline-api#_)) передаётся в inspect(), чтобы открыть этот элемент в панели Elements.



**Доступ к недавно вызванным DOM-элементам или объектам**

Часто во время тестирования вы выбираете DOM-элементы либо непосредственно в панели Elements, либо используя соответствующий инструмент (иконка — увеличительное стекло), чтобы работать с этими элементами. Также вы можете выбрать снимок использования памяти в панели Profiles для дальнейшего изучения этого объекта.

Консоль запоминает последние пять элементов (или объектов), которые вы выбирали, и к ним можно обратиться используя свойства [$0, $1, $2, $3](https://developers.google.com/chrome-developer-tools/docs/commandline-api#0_-_4) и [$4](https://developers.google.com/chrome-developer-tools/docs/commandline-api#0_-_4). Последний выбранный элемент или объект доступен как **$0**, второй — **$1** и так далее.

Следующий скриншот показывает значения этих свойств после выбора трех различных элементов подряд из панели Elements:

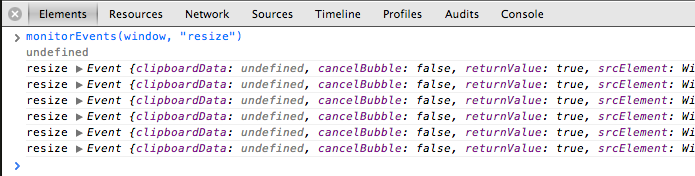


Внимание: В любом случае вы можете кликнуть правой кнопкой мыши или кликнуть с зажатой клавишей Control в консоли и выбрать пункт «Reveal in Elements Panel»

**Отслеживание событий**

Метод [monitorEvents()](https://developers.google.com/chrome-developer-tools/docs/commandline-api#monitoreventsobject_events) позволяет отслеживать определенные события объекта. При возникновении события оно выводится в консоль. Вы определяете объект и отслеживаемые события. Например, следующий код позволяет отслеживать событие "resize" объекта окна.

monitorEvents(window, "resize");



Чтобы отслеживать несколько событий одновременно, можно передать в качестве второго аргумента массив имен событий. Следующий код отслеживает одновременно события “mousedown” и “mouseup” элемента body:

monitorEvents(document.body, ["mousedown", "mouseup"]);

Кроме того, вы можете передать один из поддерживаемых «типов событий», которые DevTools сами преобразуют в реальные имена событий. Например, тип события touch позволит отслеживать события touchstart, touchend, touchmove, и touchcancel.

monitorEvents($('#scrollBar'), "touch");

Чтобы узнать, какие типы событий поддерживаются — ознакомьтесь с [monitorEvents()](https://developers.google.com/chrome-developer-tools/docs/commandline-api#monitoreventsobject_events) из Console API.

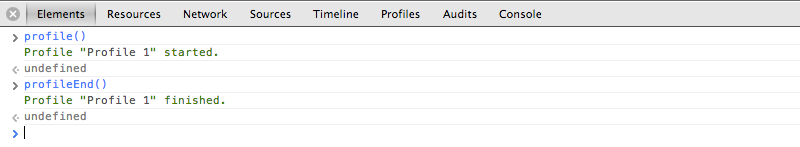
Чтобы прекратить отслеживать событие вызовите unmonitorEvents() с объектом в качестве аргумента.

unmonitorEvents(window);

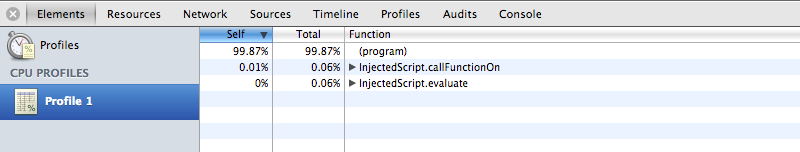
**Контроль за CPU-профайлером**

С помощью методов [profile()](https://developers.google.com/chrome-developer-tools/docs/commandline-api#profilename) и [profileEnd()](https://developers.google.com/chrome-developer-tools/docs/commandline-api#profileendname) можно создавать JavaScript профили CPU. По желанию можно задать профилю имя.

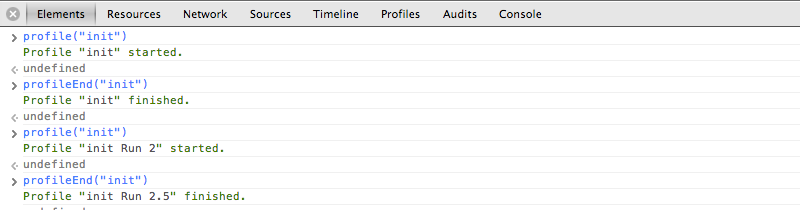
Ниже вы видите пример создания нового профиля с именем назначенным по умолчанию:



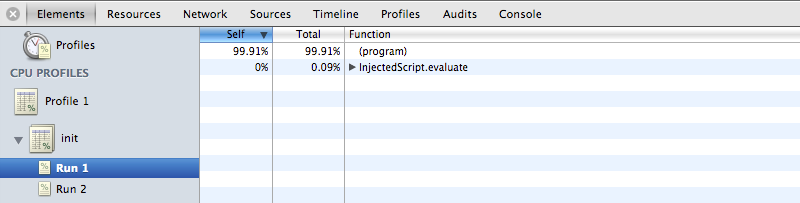
Новый профиль появляется в панели Profiles с именем Profile 1:



Если вы обозначите имя для нового профиля, то оно будет использоваться в качестве его заголовка. Если вы создаете множество профилей с одинаковыми именами, то они группируются под одним заголовком:



Результат в панели Profiles:



Профили CPU могут быть вложенными, например:

**profile**("A");**profile**("B");**profileEnd**("B")**profileEnd**("A")

Вызовы методов остановки и запуска профилирования не обязательно должны быть корректно вложены друг в друга. Например, этот пример будет работать так же, как и предыдущий:

**profile**("A");**profile**("B"); **profileEnd**("A");**profileEnd**("B");