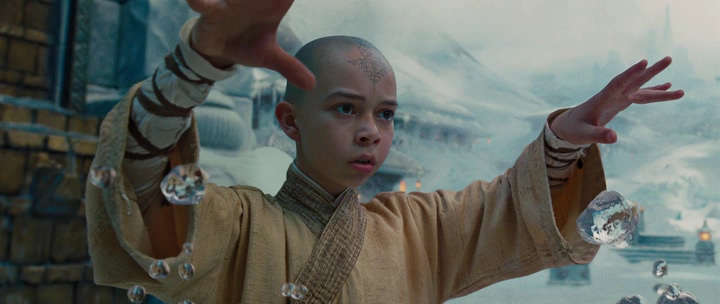
Консоль разработчика Google Chrome: десять неочевидных полезностей

[Автор оригинала: Swagat Kumar Swain](https://medium.freecodecamp.com/10-tips-to-maximize-your-javascript-debugging-experience-b69a75859329" \l ".tq5dgdfbc" \o "Автор оригинала: Swagat Kumar Swain" \t "_blank)

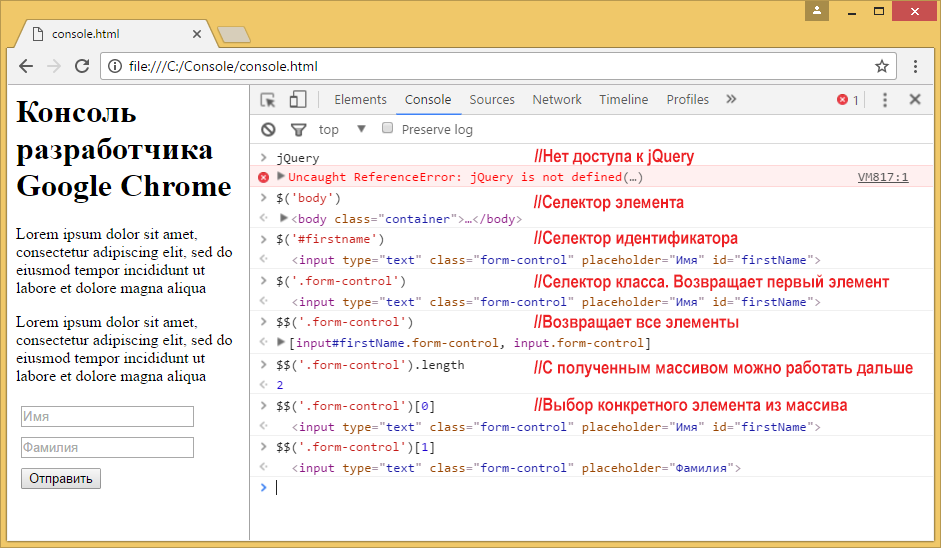
* [Блог компании RUVDS.com](https://habr.com/ru/company/ruvds/),
* [Разработка веб-сайтов](https://habr.com/ru/hub/webdev/),
* [JavaScript](https://habr.com/ru/hub/javascript/),
* [Google Chrome](https://habr.com/ru/hub/google_chrome/),
* [HTML](https://habr.com/ru/hub/html5/)
* Перевод

Как с помощью консоли разработчика превратить Google Chrome в подобие текстового редактора? Какой смысл в ней приобретает знакомый многим по jQuery значок **$**? Как вывести в консоль набор значений, оформленный в виде вполне приличной таблицы? Если сходу ответы на эти вопросы в голову не приходят, значит вкладка **Console** из инструментов разработчика Chrome ещё не раскрылась перед вами во всей красе.

[[](https://habrahabr.ru/company/ruvds/blog/316132/)](https://habrahabr.ru/company/ruvds/blog/316132/)

На первый взгляд, перед нами – вполне обычная JavaScript-консоль, которая годится только на то, чтобы выводить в неё логи ответов серверов или значения переменных. Я, кстати, так ей и пользовался, когда только начал программировать. Однако, со временем набрался опыта, подучился, и неожиданно для себя обнаружил, что консоль Chrome умеет много такого, о чём я и не догадывался. Хочу об этом сегодня рассказать. Да, если вы читаете сейчас не на мобильнике, можете тут же всё это и попробовать.

1. Выбор элементов DOM

Если вы знакомы с jQuery, не мне вам рассказывать о важности конструкций вроде **$(‘.class’)** и **$(‘id’)**. Для тех, кто не в курсе, поясню, что они позволяют выбирать элементы DOM, указывая назначенные им классы и идентификаторы. Консоль разработчика обладает похожей функциональностью. Здесь «$», однако, отношения к jQuery не имеет, хотя делает, в сущности, то же самое. Это – псевдоним для функции **document.querySelector()**.  
  
Команды вида **$(‘tagName’)**, **$(‘.class’)**, **$(‘#id’)** и **$(‘.class #id’)** возвращают первый элемент DOM, совпадающий с селектором. При этом, если в документе доступна jQuery, её «$» данный функционал консоли перекроет.  
  
Есть здесь и ещё одна конструкция: **$$**. Её использование выглядит как **$$(‘tagName’)** или **$$(‘.class’)**. Она позволяет выбрать все элементы DOM, соответствующие селектору и поместить их в массив. Работа с ним ничем не отличается от других массивов. Для того, чтобы выбрать конкретный элемент, можно обратиться к нему по индексу.  
  
Например, команда **$$(‘.className’)** предоставит нам массив всех элементов страницы с указанным при её вызове именем класса. Команды **$$(‘.className’)[0]** и **$$(‘.className’)[1]** дадут доступ, соответственно, к первому и второму элементу полученного массива.  
  
  
*Эксперименты с командами****$****и****$$***

2. Превращаем браузер в текстовый редактор

Вам приходилось ловить себя на мысли о том, что хорошо было бы править текст отлаживаемой веб-страницы прямо в браузере? Если да – значит вам понравится команда, показанная ниже.

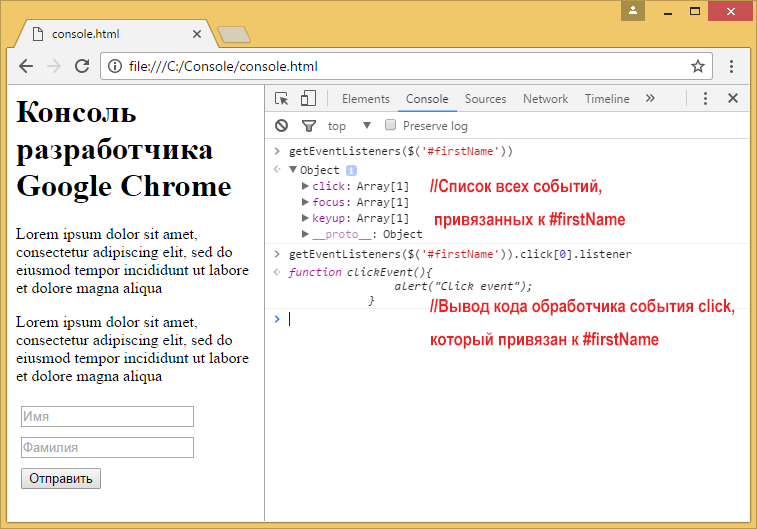
document.body.contentEditable=true

После её исполнения в консоли, документ, открытый в браузере, можно редактировать без необходимости поисков нужного фрагмента в HTML-коде.

3. Поиск обработчиков событий, привязанных к элементу

В процессе отладки может понадобиться найти обработчики событий, привязанные к элементам. С помощью консоли сделать это очень просто. Достаточно воспользоваться такой командой:

getEventListeners($(‘selector’))

В результате её выполнения будет выдан массив объектов, содержащий список событий, на которые может реагировать элемент.  
  
  
*Обработчики событий*  
  
Для того, чтобы найти обработчик для конкретного события, можно использовать такую конструкцию:

getEventListeners($(‘selector’)).eventName[0].listener

Эта команда выведет код функции-обработчика события. Здесь **eventName[0]** – это массив, который содержит все события конкретного типа. Например, на практике это может выглядеть так:

getEventListeners($(‘*#firstName’)).click[0].listener*

В результате мы получим код функции, связанной с событием **click** элемента с идентификатором **firstName**.

4. Мониторинг событий

Если вы хотите понаблюдать за возникновением событий, привязанных к конкретному элементу DOM, консоль в этом поможет. Вот несколько команд, которые можно использовать для мониторинга событий.

* Команда**monitorEvents($(‘selector’))** позволяет организовать мониторинг всех событий, связанных с элементом, которому соответствует селектор. При возникновении события делается запись в консоли. Например, команда **monitorEvents($(‘#firstName’))** позволит логировать все события, связанные с элементом, идентификатор которого – **firstName**.
* Команда**monitorEvents($(‘selector’),’eventName’)** похожа на предыдущую, но она нацелена на конкретное событие. Здесь, помимо селектора элемента, функции передаётся и имя события. Такая команда позволит вывести в консоль данные о возникновении одного события. Например, команда **monitorEvents($(‘#firstName’),’click’)** выведет сведения только по событию **click**элемента с идентификатором **firstName**.
* Команда **monitorEvents($(‘selector’),[‘eventName1’,’eventName3',….])** позволяет наблюдать за несколькими выбранными событиями. Здесь в функцию передаётся строковой массив, который содержит имена событий. Например, такая команда: **monitorEvents($(‘#firstName’),[‘click’,’focus’])**будет выводить в консоль сведения о событиях **click** и **focus** для элемента с идентификатором **firstName**.
* Команда **unmonitorEvents($(‘selector’))** позволяет прекратить мониторинг и логирование событий в консоли.

5. Измерение времени выполнения фрагмента кода

В консоли Chrome доступна функция вида **console.time(‘labelName’)**, которая принимает в качестве аргумента метку и запускает таймер. Ещё одна функция, **console.timeEnd(‘labelName’)**, останавливает таймер, которому назначена переданная ей метка. Вот пример использования этих функций:

console.time('myTime'); *//Запускает таймер с меткой myTime*

console.timeEnd('myTime'); *//Останавливает таймер с меткой myTime*

*//Вывод: myTime:123.00 ms*

Вышеприведённый пример позволяет узнать время между запуском и остановкой таймера. То же самое можно сделать внутри JavaScript-программы и найти время выполнения фрагмента кода.  
  
Cкажем, мне нужно выяснить длительность исполнения цикла. Сделать это можно так:

console.time('myTime'); *// Запускает таймер с меткой myTime*

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

 2+4+5;

}

console.timeEnd('mytime'); *// Останавливает таймер с меткой myTime*

*//Вывод - myTime:12345.00 ms*

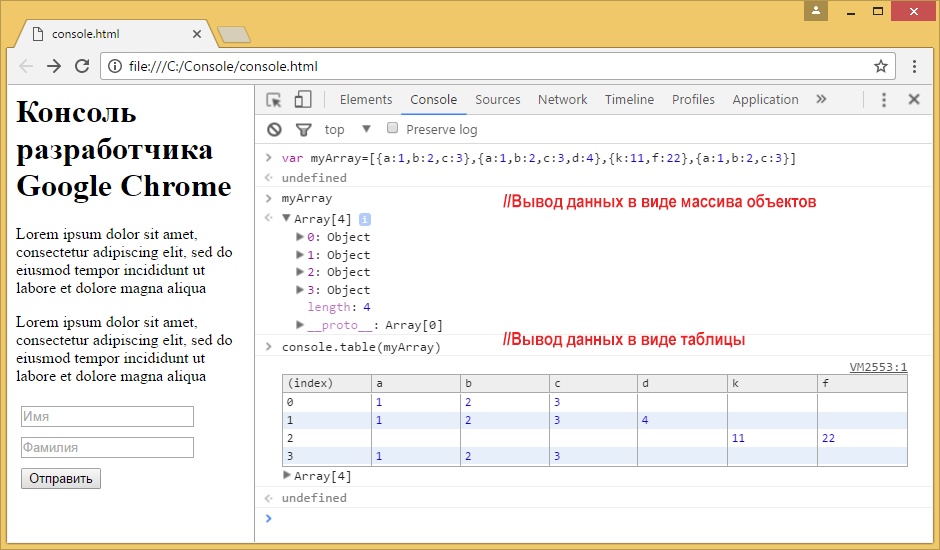
6. Вывод значений переменных в виде таблиц

Предположим, у нас имеется такой массив объектов:

var myArray=[{a:1,b:2,c:3},{a:1,b:2,c:3,d:4},{k:11,f:22},{a:1,b:2,c:3}]

Если вывести его в консоли, получится иерархическая структура в виде, собственно, массива объектов. Это – полезная возможность, ветви структуры можно разворачивать, просматривая содержимое объектов. Однако, при таком подходе сложно понять, например, как соотносятся свойства похожих элементов. Для того, чтобы с подобными данными было удобнее работать, их можно преобразовать к табличному виду. Для этого служит такая команда:

console.table(variableName)

Она позволяет вывести переменную и все её свойства в виде таблицы. Вот, как это выглядит.  
  
  
*Вывод массива объектов в виде таблицы*

7. Просмотр кода элемента

Быстро перейти к коду элемента из консоли можно с помощью следующих команд:

* Команда **inspect($(‘selector’))** позволяет открыть код элемента, соответствующего селектору, в панели **Elements** инструментов разработчика Google Chrome. Например, команда **inspect($(‘#firstName’))** позволит просмотреть код элемента с идентификатором **firstName**. Команда **inspect($$(‘a’)[3])** откроет код четвёртой ссылки, которая присутствует в DOM.
* Команды вида **$0**, **$1**, **$2** позволяют быстро переходить к недавно просмотренным элементам. Например, **$0** откроет код самого последнего просмотренного элемента, и так далее.

8. Вывод списка свойств элемента

Если надо просмотреть список свойств элемента, консоль поможет и в этом. Здесь используется такая команда:

dir($(‘selector’))

Она возвращает объект, содержащий свойства, связанные с заданным элементом DOM. Как и в прочих подобных случаях, содержимое этого объекта можно исследовать, просматривая его древовидную структуру.

9. Вызов последнего полученного результата

Консоль можно использовать как калькулятор, это, вероятно, знают все. Но вот то, что она имеет встроенные средства, позволяющие использовать в командах результаты предыдущих вычислений, известно немногим. С помощью конструкции**$\_** можно извлечь из памяти результат предыдущего выражения. Вот как это выглядит:

2+3+4

9 *//- Результат суммирования - 9*

$\_

9 *// Выводится последний полученный результат*

$\_ \* $\_

81 *// Так как последний результат 9, получаем 81*

Math.sqrt($\_)

9 *// Квадратный корень из последнего результата, который был равен 81*

$\_

9 *// Снова получаем 9 – результат предыдущего вычисления*

10. Очистка консоли и памяти

Если нужно очистить консоль и память, воспользуйтесь такой вот простой командой:

clear()

После нажатия на Enter чистая консоль будет готова к новым экспериментам.   
Вот и всё.

11, 12, 13, 14…

Откровенно говоря, это – далеко не всё. Я показал лишь некоторые из неочевидных возможностей консоли Google Chrome. На самом деле, их [намного больше](https://developers.google.com/web/tools/chrome-devtools/console/). Уверен, вы сможете расширить мой список собственными находками.  
  
Надеюсь, мой рассказ помог вам узнать о консоли Chrome что-то полезное, экономящее время, достойное стать частью повседневного арсенала веб-программиста.  
  
А какие инструменты используете вы? Давайте-давайте, делитесь какими-нибудь приколюхами! :)