**Документация методов ядра jQuery:** [**http://api.jquery.com/**](http://api.jquery.com/)

**Версии jQuery:**

* **jQuery 3x -** для современных браузеров: IE9+ и новые
* **jQuery Compact 3** - IE8+ и новые
* **jQuery 2x -** IE9+, старые iOS и Safari
* **jQuery 1x -** IE6+, старые iOS и Safari

**Подключение jQuery**

**Сейчас стало модным подключать библиотеки через CDN:**

* **jQuery:** [**http://code.jquery.com/**](http://code.jquery.com/)
* **Google** [**https://developers.google.com/speed/libraries/**](https://developers.google.com/speed/libraries/)

Что бы не было проблем с загрузкой с CDN, лучше проверять таким образом

<script src="//code.jquery.com/jquery-1.11.3.min.js"></script>

<script>

window.jQuery ||

document.write('<script src="javascript/jquery-.11.3.min.js"></<script>');

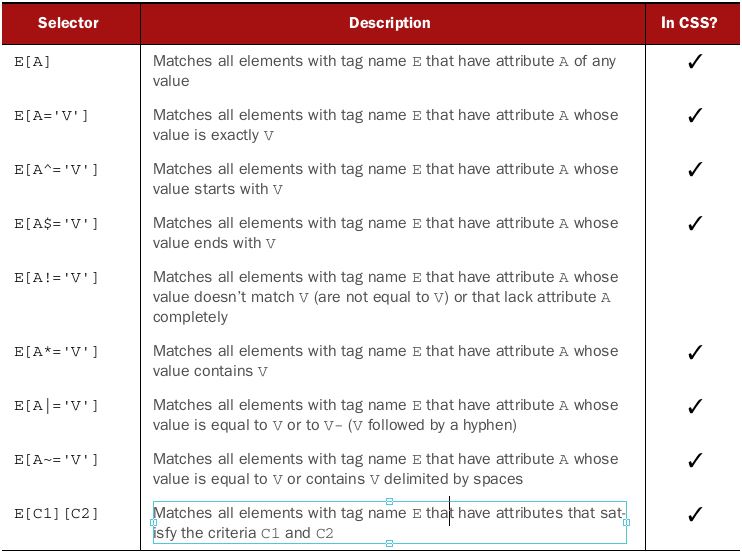
</script>

**Использование jQuery**

$() === jQuery(); //оба варианта идентичны

$(".btn") //выберет все элементы с классом btn, или один, если он только один в DOM

**Селекторы**

****

* выбор сразу нескольких разных элементов:

$('div, span'); //выберет все div и span. Отличается от нижней!

* Выбор дочернего элемента в родителе:

var $myElements = $('p', 'div')

//отберет все p в div. Второй параметр идет как контекст поиска

//переменную рекомендуют начинать с $, что бы было понятно, к чему она принадлежит

* Выбор нестандартного атрибута (поскольку метод нестандартный, не будет работать в связке с другими стандартными селекторами)

$("a[href!='http://jquery.com']") // правильное использование

$("a[href!^='http://']") // неправильное использование

* Метод выбора элемента, который ближе всего соответствует:

var $main = $("div[class|='main']");

/\*выберет div с классом main,

или который начинается с класса main, например class="main-footer"\*/

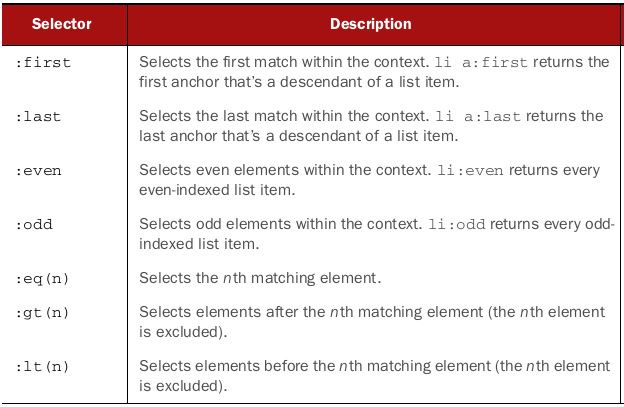
* Выбор по атрибуту:

$span['data-technologies~="javascript"'] // выберет по атрибуту

* Выбор в цепи (chain):

input[type="text"][required]

**Фильтры**

****

**:even** отбирает все четные элементы, начиная с интекса 0 (1, 3, 5...)

**:odd** - тоже самое, наоборот

**p:gt(-2) -** выберет элементы, начиная с 3 с конца

Выбирают один или несколько элементов без учета вложенности

**Фильтры потомков (аналог псевдоклассов в CSS)**

Работают соответственно псведоклассам CSS - индекс начинается с 1, выбирают элементы с учетом вложенности, как в CSS.

Пример для понимания:

<tbody>

<tr>

<td>Java</td>

<td>Static</td>

<td>1995</td>

</tr>

<tr>

<td>Ruby</td>

<td>Dynamic</td>

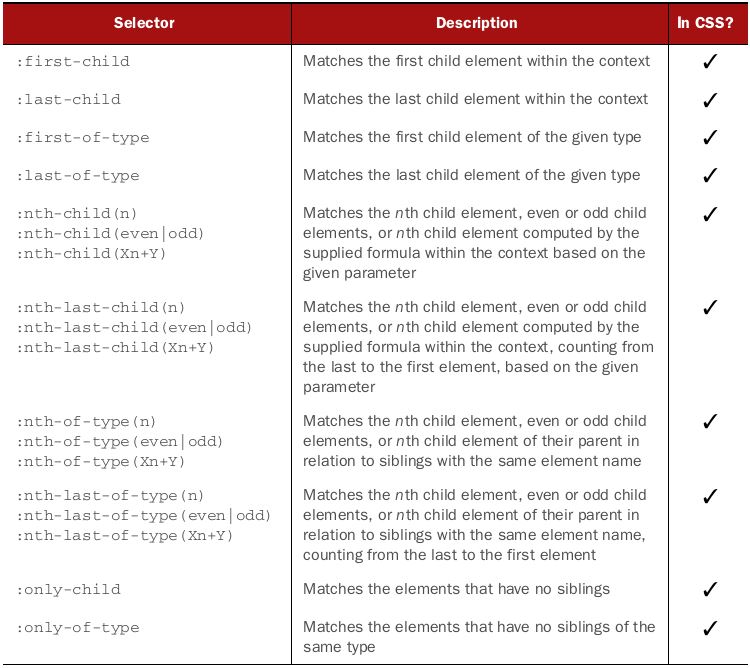
<td>1993</td>

</tr>

</tbody>

#languages td:last-child //выберет два элемента: [1995,1993]

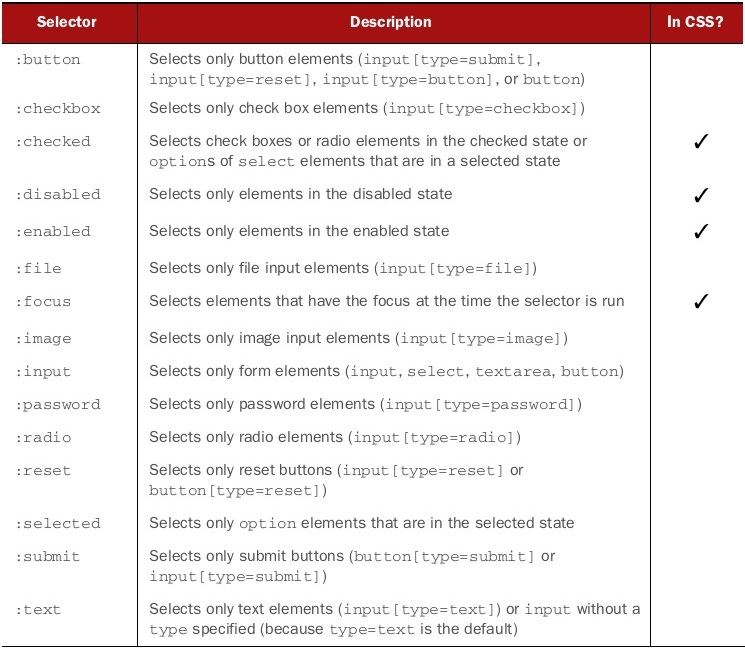
#languages td:last //выберет 1 элемент - самый последний в окне, 1993

****

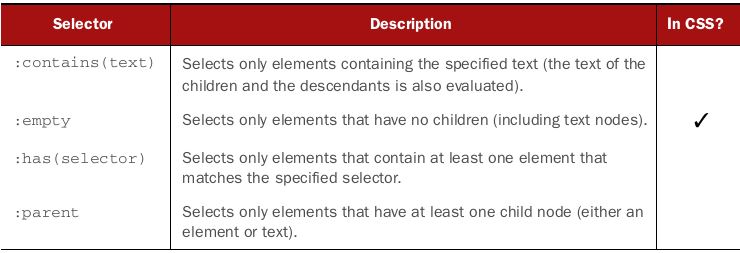
**Фильтры форм**

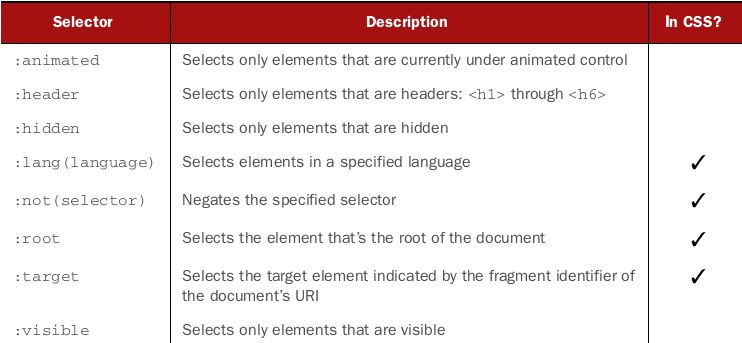
Поддерживается такая запись:

$('input:checkbox:checked:enabled');

****

**Фильтры контента**

****

**Остальные фильтры**  


**Создание своих фильтров**

Взято с css-tricks.

Как вести поиск по странице без учета регистра

<div>john</div>

<div>John</div>

<div>hey hey JOHN hey hey</div>

<script>

/\*функция поиска по страница без учета регистра\*/

$.expr[":"].contain = $.expr.createPseudo(function(arg) {

return function( elem , context, isXml ) {

return $(elem).text().toUpperCase().indexOf(arg.toUpperCase()) >= 0;

};

});

$('div:contain("john")').addClass('bingo');

</script>

**Мой фильтр:**

**$.expr[":"].checkHtml = $.expr.createPseudo(function (arg) {  
 return function (elem, context, isXml) {  
 if(elem.innerHTML.indexOf("3") !== -1){ //если в тексте элемента есть цифра 3  
 return elem.style.outline = "1px solid red"; //обводит элемент красной линией  
 };  
 };  
});  
  
$("body td:checkHtml");**

**Cоздать элемент**

**$(<div>)** - сокращенная запись, идентична $(<div></div>)

$('<img>', //тэг, который создаем

{

src: 'images/little.bear.png', //обьект - атрибуты тэга, в виде ключ\значение

alt: 'Little Bear',

title:'I woof in your general direction',

click: function() {

alert($(this).attr('title'));

}

})

.appendTo('body'); //пихаем в конец боди

**Методы jQuery**

**get(index)** - возвращает один или все соответствующие элементы из jQuery обьекты как **JS-коллекцию.** Если параметры не указаны, возвращает всю коллекцию. Индекс может быть негативным, тогда вернет элемент с конца. Если индекс не верен, вернет undefined.

Поскольку получается jQuery обьект, стандартные методы (например classList.add()) на нем работать не будут

**add(selector, [context]) - крайне похож на обычную $(selector, context) функцию.** Создает новый jQuery обьект, указанный в selector и **добавляет** его к существующей коллекции, указанной ранее. selector может быть как CSS селектор, который выберет и добавит элементы, или же существующий jQuery обьект. В context можно указать селектор, который ограничит выборку. Возвращает копию превоначальной коллекции, но уже с добавлением новых элементов

* selector - Selector|Element|Array|jQuery)

$("img[alt]").add("img[title]") // эквивалент $(" img[alt], img[title]")

Выше метод добавил к коллекции еще одну выборку, и вернул новую коллекцию

**.not(selector)** - работает аналогично :not.Создает копию коллекции без элемента, который передается в селектор. Селектор может принимать CSS селектор, массив, jQuery обьект или фукнцию. Если функция, то она вызывается для каждого элемента, где **this** - отвечает контексту выбранной ранее коллекции. Функция принимает первым аргументом индекс элемента, вторым - текущий элемент. Если функция возвращает true - удаляет элементы.

* selector принимает Selector|Element|Array|jQuery|Function

$(div).not(function(index, elem) {

/\*удалит элементы, у которых есть более двух детей или нечетный индекс \*/

return $(this).children().length > 2 && index % 2 === 0

})

var $fig = $("figure").not(function (index, elem) {

/\*если у элемента индекс 3 или у него нет класса gg - элемент будет удален\*/

return !(index === 3) && !($(elem).hasClass("gg"));

/\*использую $(elem) что бы работал метод .hasClass\*/

});

$fig.addClass("outlined");

**.filter(selector) -** удаляет все элементы, которые не отвечают параметрам, заданным в аргументе. инвертированыйаналог метода **.not(),** разница - удаляет элементы, которые возвращают **false.**

* selector принимает Selector|Element|Array|jQuery|Function. Функция принмает index и element. Возвращает true или false

Возвращает копию первоначальной коллекции без удаленных элементов

$('img').addClass('opacity').filter('[title\*="dog"]').addClass('red-border');

/\*отберет картинки, добавит класс opacity, потом из этой выборки выберет по селектору\*/

**.eq(index) -** возвращает DOM элемент по индексу, аналогично :eq селектору.

var $secondDiv = $('div').eq(1); // аналогично $('div:eq(1)');

Рекомендуется использовать метод **eq()** в целях повышения производительности

**.first()** - возвращает первый элемент. DOM элемент, не jQuery обьект

**.last()** - возвращает последний элемент. DOM элемент, не jQuery обьект

**.toArray()** - превращает jQuery обьект с выбранными элементами в JS коллекцию. Не принимает аргументов

**.index(element)** - ищет указанный элемент в коллекции или jQuery обьекте и возвращает его индекс, или ищет позицию (индекс) элемента среди его родственных элементов. Если элемент не наден, возвращает -1

* element - Selector|Element|jQuery

var index = $('#main-menu > li').index(document.getElementById('blog-link'));

При вызове без аргумента, метод ищет элемент среди элементов своего уровня:

<div id="container">

<p>This is a text</p>

<img src="image.jpg" />

<a href="/">Homepage</a>

<img src="anotherImage.jpg" />

</div>

var index = $('#container > img').index(); /\* 1, второй элемент по индексу. Если элемент содержит в себе подэлементы, вернет всю коллекцию \*/

**.parent([selector]) -** находит родителя строго на один уровень вверх.

$(this).parent()

Для того чтобы получить родителя от родителя нужно применять цепочку вызовов (произвольной глубины):

$(this).parent().parent()....

**.parents([selector]) -** возвращает список всех родительских элементов, и, может содержать селектор, для уточнения набора родителей.

$(this).parents("li.test")

**.closest(selector, [context]) -** возвращает первый ближайший родительский узел или же текущий узел - которые удовлетворяют условию в селекторе (селектор обязательный). Может принимать контекст для уточнения поиска (набор узлов заранее выбранных).

Основные **отличия его от .parents():**

* может вернуть и текущий узел
* обязательный селектор
* возвращает только первый элемент который попал под условие, поиск ведется тоже вверх по дереву.

$(this).closest("li.test") //или $(this).closest("li.test", itemsList)

**.siblings([selector]) -** возвращает набор соседних элементов (тех, которые имеют того же родителя что и текущий), может принимать уточняющий селектор. Не включает сам элемент, с которого начался поиск

$("ul li#id1").siblings()

**.children([selector])** - возвращает обьект с детьми (1 уровень вложенности). Можно отфильтровать селектором. Не возвращает текстовые ноды! (комментарии не увидит)

**.contents()** - возвращает обьект с детьми и текстовыми элементами. Не принимает аргументов

**.find(selector) -** ищет и возвращает нужные элементы, проверяя все уровни вложенности

**.nextUntil(selector, [filter]) -** перебирает и возвращает все последующие элементы одного уровня **(siblings),**  не включая элемента, с которого начался поиск, которые не содержат selector. filter позволяет сузить возвращаемые элементы до нужных. Вообще без аргументов, вернет все последющие элементы

**.prevUntil(selector, [filter]) -** перебирает и возвращает все предыдущие элементы одного уровня **(siblings),**  не включая элемента, с которого начался поиск, которые не содержат selector. filter позволяет сузить возвращаемые элементы до нужных. Вообще без аргументов, вернет всех предыдущие элементы

**.parentsUntil(selector, [filter]) -** возвращает список всех родительских элементов, не включая тех элементов, которые отвечают selector-у. filter позволяет сузить возвращаемые элементы до нужных

**.next([selector])** возвращает набор, состоящий из следующего соседа каждого элемента. Если элемент имеет больше 1 соседа, берет только ближайшего. Если задаем аргумент, вернет соседа, только если он будет отвечать аргументу

**.nextAll([selector]) -** возвращает всех следующих за элементом соседей. Если задан аргумент, вернет только тех, кто ему отвечает

**prev([selector])** возвращает набор, состоящий из предыдущего соседа каждого элемента. Если элемент имеет больше 1 соседа, берет только ближайшего. Если задаем аргумент, вернет соседа, только если он будет отвечать аргументу

**.prevAll([selector]) -** возвращает всех предыдущих от элементом соседей на одном уровне. Если задан аргумент, вернет только тех, кто ему отвечает.

**.offsetParent() -** возвращает ближайшего родителя со значениями position relative/absolutely/fixed, по которому спозиционирован элемент(ы)

**.slice(start, [end]) -** создает и возвращает новую коллекцию, содержащую элементы, которые попадают под индексы, указанные в аргументах

* start - стартовый индекс, с которой будет вырезать
* end - конечная позиция. Если негативная - высчитывает смещение с конца.

**.has(selector|element) -** выбирает элементы у которых есть **потомки**, отвечающих selector

$('div').has('img[alt]'); //выберет img с атрибутом alt

**.map(callback) -** вызывает функцию-коллбэк для каждого элемента в коллекции и собирает возвращенные значения в jQuery обьект. Что бы функция сработала, необходимо, что бы она вернула значение, которое **не null и не undefined**

* callback - функция, которая принимает два аргумента - индекс элемента, начиная с 0 и сам элемент. Элемент установлен как контекст функции (this).

var $allIDs = $('div').map(function() { return this.id; }); //вернет все элементы с ID

**.each(iterator) -** обходит все элементы в коллекции вызывая iterator для каждого. iterator принимает два аргумента - index и элемент как this

$('img').each(function(i, elem){

this.alt = 'Изображение ' + i;

});

**.is(selector) -** возвращает true, если хоть один элемент соответствует selector, иначе - false. Рекомендуется к использованию как очень быстрая.

* selector принимает Selector|Element|Array|jQuery|Function

var hasImage = $('\*').is('img'); // true / false

**.end()** - используется с цепочкой методов jQuery; оканчивает самую последнюю операцию в текущей цепочке и возвращает коллекцию элементов в ее превоначальное состояние.

/\* отбирает все картинки, прячет их, потом берет первоначальную коллекцию и добавляет всем картинкам класс \*/

$('img').filter('[title]').hide().end().addClass('my-class');

**.addBack([selector]) -** добавляет предыдущую коллекцию элементов в стек текущей коллекции, опционально фильтруя через selector

/\*проводит слияние отобранных img с их предками - div и добавляет классы и тем, и другим. Конечная коллекция получается вида: [div, div, div, img, img, img]\*/

$('div').addClass('my').find('img').addClass('red').addBack().addClass('opa');

**Обработка атрибутов и свойств обьектов в jQuery**

**Атрибут** - то, что есть в DOM-дереве. Атрибут всегда строка (**String**)

**Свойство** - то, что браузер строит на основе атрибутов DOM дерева. Извлекаться может строкой, булевым значением, или обьектом.

Обычно, свойства синхронизируются на основе атрибутов, изменяя атрибут изменяется свойство, и наоборот:

<input id="checkbox" type="checkbox" title="checkbox" checked="gotcha!" data-my="skill">

* Если атрибут существует как встроенное (родное) свойство DOM обьекта (например атрибут "title" у input), атрибут будет синхронизирован со свойством.
* Если атрибут существует как встроенное свойство, но оно - булевое значение, то значение будет синхронизировано только в булевом варианте. Например, в примере выше:

input.getAttribute('checked') // извлечет строку "gotcha!"

console.log(checkbox.checked); //извлечет булево значение - true, или false

* Если атрибут не существует как встроенное свойство, будет создан атрибут только в DOM, без создание свойства обьекта:

input.getAttribute('data-my') // skill

input['data-my']; // undefined

* Не все свойства идентичны атрибутам. Например, атрибут если src содержит тот путь, который записан в DOM, то свойство - абсолютный:

<img id="gg" src="images/clouds.jpg">

img.getAttribute("src"); // images/clouds.jpg

img.src; // http://localhost:63342/Project1/images/clouds.jpg

**.attr(name)** - получает значение, назначенное указанному атрибуту, который идет первый в коллекции отобранных элементов. Возвращает undefined, если указанная отобранная коллекция пустая или в ней не существует первого элемента

**.attr(name, value)** - устанавливает указанным атрибутам переданное значение value во всей коллекции.

* name - (String) имя атрибута, который нужно установить
* value - (String|Number|Boolean|Function), если не функция, все остальные элементы переводятся в строку. Функция принимает два параметра: индекс элемента и текущее значение имени атрибута (который указан в name)

$("img").attr("data-mine", function (index, currentValue){

return previousValue + ' I am element ' + index +

' and my name is ' + (this.id || 'unset');

})

**.attr(object with attributes) -** использует свойства, указанные в переданном обьекте что бы установить соответствющие атрибуты всем элементам в коллекции. Если в качестве значения ключа в обьекте принимается функция, будет выполнять функцию так же, как в примере выше

$("input").attr({

value: "sdafasd",

title: function (index, currentVal){

return currentVal + " " + index; } })

**.removeAttr(name) -** удаляет указанные атрибуты из каждого подходящего элемента. Принимает строковое имя атрибута или списка атрибутов, разделенными пробелами.

$('img').removeAttr('title alt');

**Пример замены value на значение placeholder** (для браузеров, у которых нет поддержки placeholder):

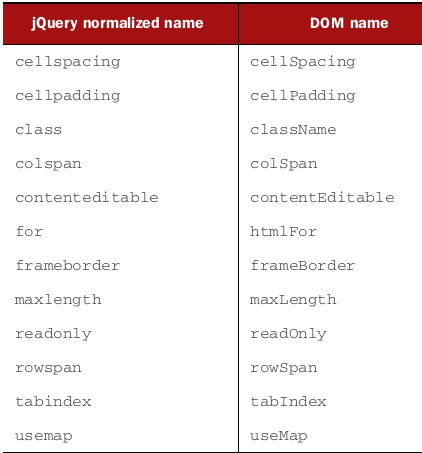
$('input').each(function(index, elm) {

var $element = $(elm); // что бы начали работать методы jQuery

$element.attr('value', $element.attr('placeholder'));

});

**.prop(name)** - аналог attr(), только работает со свойством. Механика работы такова: извлекает из attr() свойство, что бы сфокусировать attr() только на атрибутах

Принимает свойства как DOM, так и специально измененные для простоты jQuery:

**.prop(name, value)** - назначает свойство с нужным значением всем элементам в коллекции

$('#legal-age').prop('checked', true);

**.prop(properties) -** использует свойства и значения, указанные в переданном обьекте, что бы установить соответствующие значения во всех элементах коллекции

$('input:checkbox').prop({

disabled: true,

checked: true

});

**.removeProp(name)** - в отличии от removeAttr(), метод не позволяет использовать список значений, разделенных пробелами. Метод не должен использоваться что бы удалять родные свойства, такие как checked или required, потому что это полностью удалит свойство. Однажды удаленное, оно уже не может быть повторно добавленно в элемент. Что бы изменить текущий статус, нужно задавать false

**Помещение своих данных в элементы**

При хранении своих данных в элементах посредством стандартного JS, есть вероятность утечки данных, например когда хранится ссылка на элемент, который уже не нужен.

jQuery самостоятельно отслеживает и удаляют все свойства jQuery обьекта, который прикреплен к элементу, при условии того, что элемент удален

**.data(name,value)** - добавляет переданное значение в **jQuery хранилище** для всех элементов в коллекции

* name - строковое значение, имя данных. Слова разделять camelCase! При удалении (removeData()), свойства с дефисами видит только в массиве, строкой не воспринимает
* value - любое значение, даже массив или обьект

Метод обрабатывает camel-case как свойства с дефисом:

$('.class').data('lastValue'); //тоже самое, что $('.class').data('last-value');

Так же метод пытается конвертировать значение value в его родной тип:

$('.class').attr('last-value', 10); // typeof $('.class').attr('last-value') == string

$('.class').data('last-value', 10); // typeof $('.class').attr('last-value') == number

undefined не распознается, как значение, но возвращает jQuery обьект

$('#name').data('mandatory', undefined); /\* не изменит значение mandatory, но вернет jQuery обьект, на котором вызывался метод \*/

**.data(object) -** добавляет пары типа ключ-значение данного обьекта в **jQuery хранилище** для всех элементов в коллекции

$("#gg").data({

"data-name": "hide",

"data-action": "scroll",

"data-enabled": true,

});

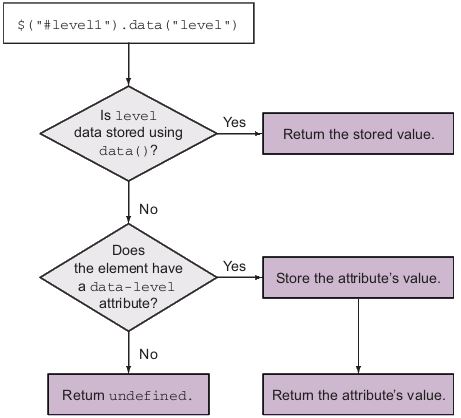
console.log($("#gg").data()); // выведет все ключ/значение

console.log($("#gg").prop("data-enabled")); // undefined, это не свойство элемента

**$.data(DOM elem, name, value) -** низкоуровневый метод работы с DOM-элементами, не принимает jQuery обьекты

$.data(document.getElementById('book'), 'price', 10);

**.data([name])** - извлекает все раннее хранимые данные, которые были записаны через метод **.data(),** если не находит такие данные -ищет **data-\***  атрибуты с таким именем. Как только метод ивзлекает значение data-\* атрибута, сразу записывает его в jQuery ранилище. Следовательно, все последующие вызовы метода больше не извлекают данные из атрибута, даже если изменить последний через метод attr(). Данные извлекаются из первого элемента коллеккции. Если name не задано, метод вернет обьект, в котором хранятся все помещенные раннее данные. Если метод не нашел заданый name, вернет undefined



<input id="level1" type="text" value="I'm a text!" data-custom="foo" />

console.log($('#level1').data('custom')); //foo

console.log($('#level1').attr('data-custom')); //foo

$('#level1').attr('data-custom', 'new value'); // занесли новое значени в хранилище

console.log($('#level1').data('custom')); // foo - взято из хранища jQuery

console.log($('#level1').attr('data-custom')); // new value - взято из атрибута

**Изменения jQuery 3**: все ключи, заданные в data() трансформируются в camelCase:

$('#name').data({'my-property': 'hello'});

console.log($('#name').data()); // myProperty

**.removeData([name])** - удаляет всю ранее помещенные данные с указанным именем во всех элементах jQuery обьекта. Параметр принимает массив или строку с разделенными пробелами именами. Если никаких аргументов не задано, удалит все значения. От себя: если ключи разделены дефисами, видит их только в массиве, строкой воспринимает только camelCase:

$('#gg').data({

"my-property": "hello",

"vedro1": "pustoe",

"lico": "protertoe"

});

console.log($("#gg").data()); // показывает все правильно

$("#gg").removeData('vedro1 my-property'); // удаляем свойства jQuery обьекта

console.log($("#gg").data()); // осталось my-property !

**$.removeData(DOM element, name**) - низкоуровневый метод удаленя данных jQuery-обьекта из DOM-элемента.

**$.hasData(DOM element) -** проверяет, есть ли какие-то данные, в прикрепленном к элементу jQuery обьекту. Возвращает булево значение.

$.hasData(document.getElementById("gg")) //true или false

**Работа со стилями и классами в jQuery**

**.addClass(names) -** добавляет переданные названия классов всем элементам в коллекции. Если передается функция, она вызывается на каждом элементе и возвращает значение как имя класса.

* names - (String|Function) - если передана функция, каждый элемент служит контекстом (this). Функция принимает два параметра: индекс элемента и значение атрибута class

Метод возвращает jQuery-коллекцию

$("figure").addClass("11 22 33"); //добавит всем элементам классы, не удаляя их текущих

**.removeClass(names) -** удаляет указанные классы с каждого элемента коллекции. Аналог addClass(), так же можно передать функцию

**.toggleClass([names], [switch]) -** добавляет указанные классы элементам, которые ими не обладают, или удаляет, если указанный класс есть. Если аргумент не передан, удаляет ВСЕ классы на элементе, при повторном вызове - восстанавливает классы. Если указан только switch аргумент, добавляет или удаляет классы, основываясь на его значении

* name **-** (String|Function). Функция с индексом и текущими классами в качестве аргумента.
* switch - принимает true или false. Если указан true, добавляет тем элементам указанный класс, у которых его нет, если false - удаляет у тех элементов, у которых он есть

Возвращает jQuery коллекцию. Пример использования

$(".social").click(function (){ // обработчик на клик по кнопке

$(".popup", this).toggleClass("hidden"); // ищет в родителе потомка popup

})

Еще пример:

/\* добавить класс, если значение какой-то переменной равно 10 \*/

$('p').toggleClass('hidden', aValue === 10);

**.hasClass(name) -** проверяет, есть ли у коллекции нужный класс. Возвращает true\false

**.css(name, value) / .css(properties) -** метод для установки значений атрибута style на DOM-элементах. Не удаляет уже существующие значения. Opacity работает даже в старых IE

* name - (String), имя свойства CSS, которое нужно установить (например, background-color)
* value - (String|Number|Function) - значение свойства. Если передается номер, jQuery переводит его в строку и добавляет "px" в конец. Что бы добавить значеню другую меру измерения, значение нужно предварительно перевести в строку и добавить соответствующую единицу измерения перед вызовом метода.  
  Функция принимает два аргумента: индекс и текущее значение. Возвращает новое значение
* properties - (Object) - можно задать пары ключ-значение, которые будут скопированы всем элементам в коллекции

Возвращает jQuery коллекцию

$('.expandable').css('width', function(index, currentWidth) {

return parseInt(currentWidth, 10) + 20 \* index; //parseInt что бы удалить "px"

});

$('.expandable').css('width', '+=20'); // тоже самое, но короче

**.css(name) -** извлекает вычисленные значения(е) CSS-свойства, указанного в name, для первого элемента в коллекции

* name (String|Array)

Возвращает вычисленное значение как строку или обьект с парами ключ-значение

var styles = $('.special').css([

'font-size', 'color', 'text-decoration' ]); // хотим получить значения свойств

for(var property in styles) { // перебираем свойста, т.к. вернулся именно обьект

console.log(property + ': ' + styles[property]);

}

**Вычисление размеров**

Все методы могут быть применены только к видимым элементам

**.width(value) / .height(value) -** устанавливает ширину и высоту элементов в коллекции. Без аргумента, извлекает значения ширины\высоты первого элемента коллекции. Этот метод не учитывает значения border и padding элемента. Извлекаемое значение вычисляется в пикселях, несмотря на то, в каких величинах оно задано в стилях и задано ли вообще, и возвращается не стройкой, а в виде Number.

* value - (String|Number|Function), принимает число с указаными единицами измерения, если заходит просто число, устанавливает ему "px"

$('div').width(500); // идентичен записи $('div').css('width', 500);

**.innerWidth(value) / .innerHeight(value) -** аналог .width(), разница в том, что значение извлекается и задается без учета ширины границ, но с учетом padding (padding-box). Если задавать значение, сам рассчитает ширину так, что бы она включала padding:

$(".social").css("padding", 50); // установили отступ 50

$(".social").innerWidth(525); // задали ширину включая padding

console.log($("figure").innerWidth()); // в стили элементу пропишется ширина = 425px

**.outerWidth(value / include margin) / .outerHeight(value / include margin) -** аналог width(), разница в том, что значения включает значения и border, и padding (border-box). Если в getter-версии вместо значения передать булево true / false, соответственно margin будет или не будет включен в расчет

$("figure").outerWidth(); // width + padding + border

$("figure").outerWidth(true); // width + padding + border + margin

**Позиционирование и прокрутка**

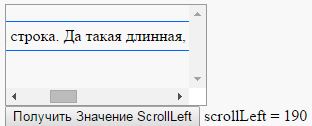
**.offset(coordinates) -** без аргумента, возвращает текущие координаты (в пикселях) первого элемента в коллекции, по отношению к окну браузера. С аргументами, устанавливает координаты всем элементам в коллекции, по отношению к браузеру.

* coordinates - (Object|Function). Принимается обьект с двумя свойствами: top и left, которые и будут новыми координатами. Функция принимает два аргумента: индекс и обьект с двумя свойствами top и left. Функция возвращает обьект, что бы установить новые значения.

Возвращает jQuery-коллекцию

$(".social").offset({top: 200, left: 200});

**.position() -** возвращает положение в пикселях первого элемента колекции к его ближайшему предку, у которого позиция relative, absolute, или fixed

**.scrollLeft(value)** - Если аргумент не задан, возвращает значение горизонтальной прокрутки первого элемента в коллекции. Возвращаемое значение - Number, которо соотносится с .position() в пикселях. Если элемент не прокручен, вернет 0.

Если задан аргумент, прокрутит на указанное расстояние

**.scrollTop(value) -** аналог scrollLeft, но работает с высотой

**Добавление элементов в другие элементы DOM**

**.html(content) -** без аргумента возвращает innerHtml первого элемента в коллекции. С аргументом раздает content всем элементам коллекции

* **content** - (String|Function). Функция принимает аргументами индекс и существующий контент

$('#message').html('<p>Your current balance is <b>1000$</b></p>');

**.text(content) -** без аргументов, возвращает конкатенацию всех текстовых узлов всех элементов в коллекции.

С аргументами, устанавливает переданный content всем элементам коллекции. Если content содержит угловые скобки (<>) и амперсанды(&), эти символы преобразуются в соответсвующие HTML обьекты

* content - (String|Number|Boolean|Function) - все преобразует в строку

Возвращает jQuery коллекцию

**.append(content, [content, content...])**  добавляет в конец контента каждого элемента в коллекции переданные аргументы. Метод принимает произовольное количество аргументов.

Если передается существующий уже элемент, он удаляется со своего старого места, клонируется нужное количество раз и вставляется в новое.

* content - (String|Element|jQuery|Array|Function). Функция принимает индекс и существующий контент каждого входящего элемента в качестве аргументов

Возвращает jQuery коллекцию

**.prepend(content, [content, content...])**  - добавляет в начало контента каждого элемента коллекции переданные аргументы. Идентичен **.append()**

**.before(content, [content, content...])**  - вставляет переданные аргументы в DOM как элементы одного уровня с целевым элементом. Добавляет аргументы перед целевым элементом

$('#description').before($('button')); // вставит button перед description

**.after(content, [content, content...])**  - вставляет переданные аргументы в DOM как элементы одного уровня с целевым элементом. Добавляет аргументы после целевого элемента

**.appendTo(target) -** вставляет каждый элемент в коллекцию отобранных элементов в конец контента указанной target(s).

* **target -** (String|Element|jQuery|Array)

**.prependTo(target)** - аналогичноappendTo(), только добавляет элемент в начало контента

**.insertBefore(target) -** вставляет элемент до указанной target

$('button').insertBefore('#description'); // вставит button перед description

**.insertAfter(target) -** вставляет элемент сразу после указанной target

**Оборачивание элементов (wrapping)**

**.wrap(wrapper) -** оборачивает каждый элемент в jQuery-коллекции в переданный аргумент. Если у аргумента есть контент, обернутые элементы внутри будут размещены после контента

* **wrapper -** (String|Element|jQuery|Function) - строка должна содержать открывающей тег и опционально закрывающий. Если передан селектор, который отобрал больше одного элемента, для оборачивания будет использован только первый в коллекции.  
  Фукции передается только один аргумент - индекс, this которого - текущий элемент. Функция возвращает значение, которое может быть HTML фрагментом или jQuery обьектом

Возвращает jQuery коллекцию

**.wrapAll(wrapper) -** оборачивает все элементы коллекции в один wrapper

**.wrapInner(wrapper) -** оборачивает **контент элементов коллекции**, включая text nodes, в переданный wrapper

**.unwrap() -** удаляет родителя указанных элементов в коллекции. Дочерний элемент, вместе со всеми элементами одного уровня, меняется местами с родительским элементом

В **jQuery 3** добавлена возможность передать аргумент, который содержит например CSS-селектор. Если родитель соответсвует CSS-селектору, он будет удален

**Удаление элементов**

**.remove([selector]) -** удаляет все элементы и их контент в коллекции. Возвращает удаленный элемент, но без прикрепленных eventListeners и jQuery data.

* selector - (String) - фильтрует, какие элементы надо удалить

Возвращает jQuery коллекцию

**.detach([selector]) -** удаляет все элементы и их контент из колекции, возвращает удаленные элементы, сохраняя при этом все привязанные eventListeners и jQuery data.

Метод часто используют, что бы модифицировать элемент по причине увеличения производительности, т.к. изменить удаленный элемент быстрее, чем когда он есть в DOM

Возвращает jQuery коллекцию

**.empty()** - удаляет контент всех DOM элементов в коллекции. Удобно использовать с Ajax, что бы добавить новый контент в какой-то контейнер

Возвращает jQuery коллекцию

var newContent = '<p>Wow, this new content is awesome!</p>';

$('#content').empty().html(newContent)

**Клонирование и замена элементов**

**.clone([copyHandlersAndData[, copyChildrenHandlersAndData]])** - делает глубокую копию элемента в jQuery коллекции и возвращает новую jQuery коллекцию. Элементы и все дочерние элементы скопированы. eventListeners и jQuery data может быть скопирована выборочно, в зависимости от второго аргумента

* copyHandlersAndData (Boolean) - если true, обработчики событий и data скопируется
* copyChildrenHandlersAndData (Boolean) - если true, копирует обработчики событий и data включая всех дочерних элементов. Если не задан, принимает значение первого аргумента метода (вида .clone(true, true))

Возвращает новую jQuery коллекцию

$('ul').clone(true).insertBefore('#here'); // второй аргумент в clone так же будет true

**.replaceWith(content) -** удаляет все элементы и вставляет вместо каждого из них content.

* content (String|Element|Array|jQuery|Function). Функция вызывается с this для каждого элемента, и не принимает аргументов. Возвращает значение как новое содержание

Возвращает jQuery коллекцию, содержащую **удаленные** элементы

$(".list li").replaceWith("<div>заменено</div>")

**.replaceAll(target)** - заменяет каждый элемент в коллекции, переданной в target теми элементами, к которым метод применен

* target - (String|Element|Array|jQuery)

Возвращает jQuery коллекцию, содержащую **вставленные** элементы

$("<div>заменено</div>").replaceAll(".list li") // тоже самое, как и пример выше

**Методы обработки форм**

**.val(value) -** возвращает текущее значение value первого элемента jQuery коллекции. Если первый элемент - <select> и ни какой <option> не выбрано, метод вернет null. Так же, если первый элемент - не элемент формы, вернет пустую строку. Если <select> имеет атрибут multiselect, и хотя бы одна <option> выбрана, вернет массив со значениями выбранных <option>.

Если задан аргумент, устанавливает его значение value всем элементам в коллекции, если передан массив value - любые checkbox, radio buttons, <option> в коллекции становятся checked или selected, если их значение их value отвечает любым value, переданным в аргументе.

* value - (String|Number|Array|Function). В функцию передается два аргумента: индекс и текущее значение, возвращает она новое значение.

getter возвращает выбранные value, setter возвращает jQuery-коллекцию

$('input[type="radio"][name="radio-group"]:checked').val(); // getter

$('input[type="checkbox"], select').val(['one', 'two', 'three']); // setter

**Обработка событий (events)**

Когда на странице происходит какое-то действие, образец обьекта под названием Event передается в обработчик в качестве первого аргумента во всех новых браузера.

В IE 8- по-другому: он переносит экземпляр Event в глобальное свойство: window.evemt

Для проверки, первой инструкций в non-jQuery event handler пишут это:

event = event || window.event

DOM 2 внедрил такое поведение event: при клике на элемент, event сначала стартует от своего корня, потом доходит до элемента, на котором было совершено событие, и после этого опять спускается вниз

Root -> Target называется capture phase. Параметр **true** ловит на этой фазе

Target -> Root называется bubble phase. По умолчанию, 3й аргумент установлен на **false**

Elem.addEventListener(“click”, function go(), **false / true**)

**.on (eventType, selector, data, handler)**

**.on (eventsHash, selector, data) –** прикрепляет обработчик событий с одним или несколькими событиями для выбранных элементов

* eventType - (String) - указывает название эвента или эвентов. Можно указать несколько эвентов, разделив их пробелом. Так же, можно указать так: click.myapp.mymodule
* selector - (String) - можно указать для event delegation, что бы словить евент на тех потомках, которые нужны (например, обработчик на ul, что бы ловить клики на каждом li)
* data - информация, которую можно передать в event как свойства с названием data и сделать доступной для функции-обработчика
* handler - (Function) - функция-обработчик, при вызове принимает Event как первый аргумент, и this вторым аргументом. Вместо функции можно написать false, что бы сработало .preventDefault() и stopPropagation() Так же принимает дополнительные параметры от функций trigger() или triggerHandler().

В jQuery 3 может принимать обьект вместо функции.

* eventHash - (JS Object) евенты можно передавать в виде event: функция-обработчик.

Возвращает jQuery-коллекцию.

//пример 1

$('img').on('click', null, null, function() { ... });

$('img').on('click', function() { ... }); // аналог верхней

//пример 2

$('button').on({

click: function(event) {

console.log('Button clicked!');

},

mouseenter: myFunctionHandler, // задаем в обьекте event и handlers

mouseleave: myFunctionHandler });

**.one(eventType, selector, data, handler), .one(eventsHash, selector, data) -** тоже самое, что .on(), но срабатывает только 1 раз, потом удаляется

**.off(eventType[, selector][, handler]), .off(eventsHash, [, selector]), .off() -** удаляет прослушчики, принимаются аргументы аналог .on(). Без аргументов удаляет все прослушчики на элементе

Если вы пишете плагин или работаете над крупным сайтом, над которым работают другие люди, желательно при установке обработчиков событий использовать пространство имен, чтобы в последствии можно было безболезненно удалять с элементов свои обработчики, не затрагивая чужих. Если указать только пространство имен: **".myPlugin"**, то с выбранных элементов будут удалены обработчики всех типов событий, с указанным пространством имен.

Чтобы удалить делегированные обработчики, необходимо указывать параметр **selector**, в точности совпадающий с таким же параметром, использованным в методе .on() при установке удаляемого теперь обработчика. Если нужно удалить с элементов все делегированные обработчики, не затрагивая обработчики, установленные непосредственно, следует в параметре selector указать специфическое значение — **"\*\*"**

var validate = function() { // обработка события };

// назначаем обработчики в пространстве имён ".validator"

$( "form" ).on( "click.validator mouseenter", "button", validate );

$( "form" ).on( "keypress.validator", "input[type='text']", validate );

// удаляем все обработчики из пространства имён ".validator", mouseenter остался

$( "form" ).off( ".validator" );

**jQuery.Event** имеет стандартные свойства, как и DOM-обьект, хотя сам по себе является отдельной сущностью. Так же он имеет методы:

**preventDefault()** - предотвращает любые события по умолчанию

**stopPropagation()** - предотвращает распространение события на вниз и вверх по дереву

**stopImmediatePropagation() -** останавливает любое дальнейшее распространение события включая события на currentTarget

**isDefaultPrevented() -** вернет true, если preventDefault() был вызван на этом обьекте

**isPropagationStopped()**

**isImmediatePropagationStopped()**

**.trigger(event, data) -** вызывает любые обработчики событий, назначенные для передаваемых событий в eventType для всех элементов

* **event -** (String|jQuery.Event)указывает имя события. Можно указывать именовынные события (click.myFunc) и передавать jQuery.Event
* **data -** (Any) - данные для передачи в обработчики

Возвращает jQuery коллекцию

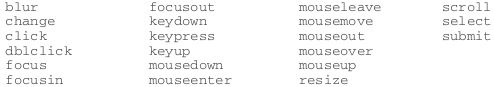
$('#foo').trigger('click', 1, 2, 3); // 1, undefined, undefined

$('#foo').trigger('click', [1, 2, 3]); // 1, 2, 3 передача в массиве работает нормально

**.triggerHandler(event, data) -** вызывает обработчики событий на первом элементе коллекции, как будто под preventDefault() и stopPropagation(). Цепь jQuery после него работать не будет.

Возвращает любое значение, которое возвращает последний вызванный обработчик или undefined

**"event"(data, handler) -** сокращение для on() и trigger() методов

****

Возвращает jQuery коллекцию

$('#btn').click(function() { // привязывает функцию по клику

console.log('Button clicked');

}).click(); // вызывает, как метод .trigger()

**.hover(enterHandler, leaveHandler) , .hover(handler) -** назначает обработчики для mouseenter и mouseleave событий для элементов. Эти обработчики срабатывают только когда когда мышь входит и выходит из области элементов, игнорируя переходы по дочерниим элементам. В чистом браузерном виде:

* mouseover, mouseout - когда мышь входит в дочерний блок, обработчик на родительском блоке срабатывает, как будто мышь вообще вышла за его пределы
* mouseenter, mouseleave - когда мышь входит в дочерний блок, обработчики на родительском срабатывать не будут
* enterHandler, leaveHandler , handler **-** функции, которые будут срабатывать

Возвращает коллекцию jQuery

**Создание custom event**

$('#btn').on('customEvent', function(){

alert('customEvent'); // работает!

});

$('#anotherBtn').click(function() {

$('#btn').trigger('customEvent'); // вручную вызываем кастомное событие

});

**Именованые события (namespaced events)**

$('.my-class').on('click.editMode', myFunction); // присвоили click имя

$('\*').off('.editMode'); // удалили все обработчики с именем editMode

**АНИМАЦИИ И ПЕРЕХОДЫ**

**.hide(duration[, easing] [, callback])**

**.hide()**

Уменьшает ширину и высоту элементов без "display: none", одновременно уменьшая прозрачность, после чего полностью удаляет элементы из потока (display: none). Если вызван без параметров, операция производится моментально.

* duration - (Number|String) длительность эффектов, в ms
* slow =
* normal = 400ms;
* fast =
* easing - (String) тип кривой Безье. По умолчанию - swing
* callback - (Function) функция, которая будет вызвана после анимации. Контекст this устанавливается для элемента, который был анимирован

Возвращает jQuery коллекцию

**.hide( options ) -** передаваемые значения в виде обьекта

* always (Function) - фнкция, которая вызывается, когда анимация выполнена или остановлена без выполнения. Promise обьект передается в функцию разрешенным или запрещенным
* complete (Function) - функция, которая вызывается, когда анимация окончена
* done - (Function) - функция, которая вызывается когда анимация завершена, что оначает что ее Promise обьект разрешен
* duration - время выполнения, в ms
* easing - тип кривой Безье для анимации
* fail - (Function) функция, вызываемая, когда анимация не была завершена. Promise обьект передается в нее запрещенным
* progress (Function) - функция, исполняемая после каждого шага анимации. Функция вызывается только один раз для обьекта анимации, несмотря на количество анимационных свойств
* queue - (Boolean|String) - булево значение, указание на то, будет ли анимация помещена в очередь эффектов. Если *false* - анимация начнется незамедлительно. По умолчанию *true* . В случае если передана строка, анимация добавляется в очередь, представленной в строке. Когда используется своя строка, анимация не начинается автоматически
* specialEasing - (Object) - карта одного или нескольких CSS свойств, значения которых easing на функцию
* start - (Function) - функция, которая вызывается, когда анимация начинается
* step - (Function) - функция, которая исполняется для каждого из анимационных свойств для каждого из анимированных элементов

**.show(duration[, easing] [, callback]) | .show() | .show( options ) -** показывается скрытые элементы. Параметры и действие то же, что и *.hide()* метода

**.toggle(duration[, easing] [, callback]) | .toggle() | .toggle( options ) -** показывает/скрывает элементы. Параметры и действие то же, что и *.hide() \ .show* методов

**.fadeIn() / .fadeOut() / .fadeToggle() -** скрыть / показать элементы без изменения их высоты \ ширины. Просто уменьшает opacity до 0, потом убирает из потока (display: none). Опции такие же, как в методах выше. Скорость по умолчанию - 400

**.fadeTo(duration, opacity, [easing], [callback])** - постепенно увеличивает или уменьшает opacity от текущих настроек к указанным. Никогда не удаляет элемент из потока