(function($, undefined){

// тут тихо и уютно

// мы всегда будем уверены, что $ == jQuery

// a undefined не переопределен ;)

})(jQuery);

a[href] — все ссылки с атрибутом href

a[href=#] — все ссылки с href=#

a[href~=#] — все ссылки с # где-то в href

a[hreflang|=en] — все ссылки, для которых hreflang начинается с en и обрезается по

символу «-» — en, en-US, en-UK

a[href^=http] — ссылки начинающиеся с http

a[href\*="google.com"] — ссылки на погуглить

a[href$=.pdf] — ссылки на PDF файлы (по идее)

:first-child – первый дочерний элемент

:last-child – последний дочерний элемент

n это как i=0 в for, и i++

:nth-child(2n+1) – выборка элементов по несложному уравнению –

подробнее можно прочитать в статье «Как работает nth-child»

[http://web-standards.ru/articles/nth-child/]

:not(…) – выбрать те, что не подпадают под вложенную выборку

**jQuery: поиск элементов**

Обычный же вызов elem[0].children[0] на порядки обгонит все вышеприведённые способы, особенно если детей много.

Простейший поиск элементов выглядит так:

|  |
| --- |
| var result = $('...CSS-селектор...'); |

Например, $("div a") — ссылки a внутри div.

Возможны и более сложные запросы, например:

|  |  |
| --- | --- |
| var list = $('li > a:odd:not([href^="http://"])'); | |
| // расшифровка: нечётные (:odd) ссылки внутри li, |

|  |
| --- |
| // кроме тех (:not), у которых атрибут href начинается c http:// |

**Для поиска внутри какого-либо элемента — можно передать его вторым аргументом $.**

Например, найдём все a внутри #menu:

|  |
| --- |
| var menu = document.getElementById('menu'); |
| $('a', menu); // поиск аналогичен menu.querySelectorAll('a') | |

## [Метод each](http://learn.javascript.ru/jquery-search" \l "%D0%BC%D0%B5%D1%82%D0%BE%D0%B4-each)

Для более удобного перебора у jQuery-коллекции есть метод [each](http://api.jquery.com/each/). Его синтаксис похож на [forEach](https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Array/forEach) массива:

|  |  |
| --- | --- |
| .each( function(index, item) ) | |
| var links = $('li a'); // найти все ссылки на странице внутри LI | | |
|  |

|  |
| --- |
| links.each(function(i, a) { |
| alert(i + ': ' + a.innerHTML); | |

|  |
| --- |
|  |
| if (i == 1) return false; // стоп на элементе коллекции с индексом 1 | |

|  |
| --- |
| }); |

**При переборе each, текущий элемент передаётся как this.**

Можно было бы использовать это, чтобы сделать код короче:

|  |
| --- |
| $('li a').each(function(i) { |
| alert(i + ': ' + this.innerHTML); | |

|  |
| --- |
|  |
| if (i == 1) return false; | |

|  |
| --- |
| }); |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

**Для получения одного элемента из jQuery-коллекции есть три способа:**

**Метод** [**get(индекс)**](http://api.jquery.com/get/)**, работает так же, как прямой доступ:**

|  |
| --- |
| alert( $('body').get(0) ); // BODY |

**Прямой доступ по номеру:**

|  |
| --- |
| alert( $('body')[0] ); // BODY |

Это — самый быстрый и прямой доступ, он использует внутреннюю структуру jQuery-коллекции: элементы там хранятся по индексам. Но этим он нарушает принцип инкапсуляции.

 **Метод** [**eq(индекс)**](http://api.jquery.com/eq/) **возвращает коллекцию из одного элемента — с данным номером.**

Он отличается от метода [get(индекс)](http://api.jquery.com/get/) и прямого обращения по индексу тем, что возвращает именно jQuery-коллекцию с одним элементом, а не сам элемент.

Например:

|  |  |
| --- | --- |
| // DOM-элемент для первой ссылки | |
| $('a').get(0); |

|  |
| --- |
|  |
| // jQuery-объект из одного элемента: первой ссылки | |

|  |
| --- |
| $('a').eq(0); |

Во втором случае вызов eq создаёт новую jQuery-коллекцию, добавляет в нее нулевой элемент и возвращает. Это удобно, если мы хотим дальше работать с этим элементом, используя методы jQuery.

Если элемента с таким номером нет — eq возвратит пустую коллекцию.

## [«Сцепление» вызовов](http://learn.javascript.ru/jquery-search" \l "%D1%81%D1%86%D0%B5%D0%BF%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5-%D0%B2%D1%8B%D0%B7%D0%BE%D0%B2%D0%BE%D0%B2)

**Почти все методы jQuery-объекта возвращают jQuery-объект — либо новый, либо тот на котором вызваны.**

Это называют *«сцепление»* вызовов, или *«чейнинг»* (от англ. «chaining»):

|  |  |
| --- | --- |
| $('li a[href$=".pdf"]') // ссылки, оканчивающиеся на pdf | |
| .each(function() { |

|  |  |
| --- | --- |
| this.className = "pdf"; // дать им класс | |
| }) |

|  |
| --- |
| .each(function() { |
| alert(this.href); // вывести | |

|  |
| --- |
| }); |

## [Шпаргалка по методам](http://learn.javascript.ru/jquery-traversal" \l "%D1%88%D0%BF%D0%B0%D1%80%D0%B3%D0%B0%D0%BB%D0%BA%D0%B0-%D0%BF%D0%BE-%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D0%B0%D0%BC)

## [Шпаргалка по методам](http://learn.javascript.ru/jquery-traversal#%D1%88%D0%BF%D0%B0%D1%80%D0%B3%D0%B0%D0%BB%D0%BA%D0%B0-%D0%BF%D0%BE-%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D0%B0%D0%BC)



## [По текущей коллекции](http://learn.javascript.ru/jquery-traversal" \l "%D0%BF%D0%BE-%D1%82%D0%B5%D0%BA%D1%83%D1%89%D0%B5%D0%B9-%D0%BA%D0%BE%D0%BB%D0%BB%D0%B5%D0%BA%D1%86%D0%B8%D0%B8)

[filter(фильтр)](http://api.jquery.com/filter), [not(фильтр)](http://api.jquery.com/not), [has(фильтр)](http://api.jquery.com/has)

Возвращают отфильтрованную коллекцию, содержащую только элементы…

* для filter — подходящие под фильтр,
* для not — не подходящие под фильтр,
* для has — содержащие потомка, подходящего под фильтр.

|  |  |
| --- | --- |
| list.filter(':even').css('color', 'blue');// ФИЛЬТР: чётные | |
| list.not(':even').css('color', 'green');  // НЕ чётные |

|  |  |  |
| --- | --- | --- |
| list.has('em').css('background', 'pink'); // СОДЕРЖАТ em | | |
| [eq(n)](http://api.jquery.com/eq), [first](http://api.jquery.com/first), [last](http://api.jquery.com/last), [slice(start, end)](http://api.jquery.com/slice)  Возвращает новую коллекцию…   * eq(n) — с одним элементом по его номеру, * first/last — из первого/последнего элемента, * slice(start[, end]) — из элементов с номера start до end. * $( 'div' ).slice( 1, 8).css( 'background', 'yellow' );   [is(фильтр)](http://api.jquery.com/is)  Возвращает true, если *какой-нибудь* элемент из коллекции подходит под фильтр. | | |
| alert( list.is('li') ); // true |
| alert( list.is('.other') ); // true | |

|  |
| --- |
| alert( list.is(':hidden') ); // false |
|  |

[add(элементы)](http://api.jquery.com/add)

Возвращает новую коллекцию из элементов текущей и новых, по селектору.

|  |  |
| --- | --- |
| var elems = $('ul'); | |
|  |

|  |
| --- |
| // можно добавить элемент DOM |
| var li = document.getElementsByTagName('li')[0]; | |

|  |
| --- |
| elems = elems.add( li ); |
| alert(elems.length); // 2 | |

|  |
| --- |
|  |
| elems = elems.add( '.other' ); // можно - по селектору | |

|  |  |
| --- | --- |
| alert(elems.length); // 3 | |
|  |

|  |  |
| --- | --- |
| elems = elems.add( $('li') );  // можно - коллекцию | |
| alert(elems.length); // 5, дубликаты удаляются |

|  |
| --- |
| </script> |

Важно: эта функция не меняет текущую коллекцию, она создаёт новую из существующих и добавленных элементов.

[map(функция)](http://api.jquery.com/add)

Этот метод стоит особняком. Он не меняет коллекцию, не ищет в ней, а пропускает её через функцию и возвращает результаты.

|  |
| --- |
| $('code') |
| .css('color', 'red') // подсветить CODE красным | |

|  |
| --- |
| .map(function() { |
| return this.parentNode; // получить коллекцию родителей CODE | |

|  |
| --- |
| }) |
| .css('color', 'green'); // подсветить их зелёным | |

|  |
| --- |
| </script> |

Заметим две приятные особенности:

* Большинство методов, которые осуществляют фильтрацию, могут принимать как селектор так и фильтрующую функцию. jQuery любит функции.
* Большинство методов, которые принимают элементы, могут получать их в виде jQuery-коллекции или селектора. Главное, чтобы найти по этому можно было.

## [По родителям](http://learn.javascript.ru/jquery-traversal" \l "%D0%BF%D0%BE-%D1%80%D0%BE%D0%B4%D0%B8%D1%82%D0%B5%D0%BB%D1%8F%D0%BC)

[parent()](http://api.jquery.com/parent/), [parents(фильтр)](http://api.jquery.com/parents/), [parentsUntil(стоп, фильтр)](http://api.jquery.com/parentsUntil/)

Родственные связи:

$("\*") // выбор всех элементов

$("article > h2") // выбираем все теги h2 которые являются

// непосредственными потомками тега article

$("article > \*") // выбор всех потомков элементов article

$("article").children()// --

$("p").parent() // выбор всех прямых предков элементов p

$("p").parents() // выбор всех предков элементов p (не понадобится)

$("p").parents("div") // выбор всех предков элемента p которые есть div

// parents принимает в качестве параметра селектор

Родители — один parent, все parents(фильтр) (с фильтром по селектору) и до определённого parentsUntil(где остановиться, селектор для элементов).

$( "p" ).parent( ".selected" ).css( "background", "yellow" );

var parentTag = $( this ).parent().get( 0 ).tagName;

функция возвращает всех родителей их имена тегов, и заносится в массив который преобразуется в строку через запятую

var parentEls = $( "b" ).parents()

.map(function() {

return this.tagName;

})

.get()

.join( ", " );

Возвращает количество родителей div

$( "div" ).css( "border-color", "white" );

var len = $( "span.selected" )

.parents( "div" )

.css( "border", "2px red solid" )

.length;

var link = $('a[href\*="blog"]'); // ссылка, атрибут href содержит "blog"

[closest(фильтр, элемент-контейнер)](http://api.jquery.com/closest/)

Ищет одного, ближайшего, родителя, подходящего под селектор.

Второй, необязательный, аргумент элемент-контейнер, если он передан, ограничивает поиск. jQuery будет идти вверх до тех пор, пока не встретит этот DOM-элемент.

var ul = link.closest('ul'); // ближайший сверху UL

ul = link.closest('ul', li); // ближайший сверху UL, но внутри LI

## [По потомкам](http://learn.javascript.ru/jquery-traversal" \l "%D0%BF%D0%BE-%D0%BF%D0%BE%D1%82%D0%BE%D0%BC%D0%BA%D0%B0%D0%BC)

|  |  |  |  |
| --- | --- | --- | --- |
| [find(селектор)](http://api.jquery.com/find/)  // для каждого LI искать CODE среди потомков, вернуть их коллекцию | | | |
| $('li').find('code');  $("article h2") // выбираем все теги h2 внутри тега article  $("div article h2") // выбираем все теги h2 внутри тега article  // внутри тега div, в доме который построил Джек  $("article").find("h2") // аналогично примерам выше  $("div").find("article").find("h2") //  [children(селектор)](http://api.jquery.com/children/), [contents()](http://api.jquery.com/contents/)  Выбирает детей по селектору, без аргументов — всех детей: | | |
| $('li') |
| .children('code') // вернуть всех детей LI, подходящих под селектор code | |

Метод [contents()](http://api.jquery.com/contents/) — также возвращает детей, но в отличие от children — узлы всех типов, включая текстовые и комментарии, а не только узлы-элементы.

## [По соседям](http://learn.javascript.ru/jquery-traversal" \l "%D0%BF%D0%BE-%D1%81%D0%BE%D1%81%D0%B5%D0%B4%D1%8F%D0%BC)

[prev()](http://api.jquery.com/prev), [prevAll(фильтр)](http://api.jquery.com/prevAll), [prevUntil(элемент, фильтр)](http://api.jquery.com/prevUntil)

Получить левого соседа prev, всех левых соседей prevAll или всех левых соседей prevUntil до указанного (элемент), подходящих под фильтр.

$("h1 + h2") // выбор всех h2 элементов, перед которыми идут h1

// элементы (у нас только один такой)

$("#stick ~ article") // выбор всех article элементов после элемента

// c id=stick

$("#stick").prev() // выбор предыдущего элемента от найденного

$("#stick").next() // выбор следующего элемента от найденного

[next()](http://api.jquery.com/next), [nextAll(фильтр)](http://api.jquery.com/nextAll), [nextUntil(элемент, фильтр)](http://api.jquery.com/nextUntil)

То же самое, но правые соседи.

[siblings()](http://api.jquery.com/siblings)

Получить коллекцию всех соседей.

## [Стек селекторов: методы end и addBack](http://learn.javascript.ru/jquery-traversal" \l "%D1%81%D1%82%D0%B5%D0%BA-%D1%81%D0%B5%D0%BB%D0%B5%D0%BA%D1%82%D0%BE%D1%80%D0%BE%D0%B2-%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D1%8B-end-%D0%B8-addback)

**Все методы не влияют на текущую коллекцию, они создают и возвращают новую.**

|  |
| --- |
| $('form') |
| .find(':checkbox') // нашли чекбоксы внутри | |

|  |  |
| --- | --- |
| .each(function() { ... }) // обработали | |
| .end() // вернулись обратно на форму |

|  |
| --- |
| .find(':radio') // поискали другие элементы внутри.. |
| .each(function() { ... }); // сделали с ними что-то ещё | |

## [Классы и стили](http://learn.javascript.ru/jquery-dom" \l "%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D1%8B-%D0%B8-%D1%81%D1%82%D0%B8%D0%BB%D0%B8)

* addClass, hasClass, removeClass

attr  
prop  
css

Таблица методов:

.append(parent)

Вставить последним потомком parent

.prepend(parent)

Вставить первым потомком parent

.insertAfter(sibling)

Вставить сразу после элемента sibling

.insertBefore(sibling)

Вставить сразу перед элементом sibling

.remove()

Удалить элемент из DOM, удалить связанные с ним данные и обработчики событий.

.detach()

Удалить элемент из DOM, не удаляя связанные с ним данные и обработчики.

.empty()

Удалить всех потомков вызовом remove

Зеркальные методы:

* append → appendTo
* prepend → prependTo
* insertAfter → after
* insertBefore → before

## [jQuery чеклист](http://learn.javascript.ru/widgets-intro" \l "jquery-%D1%87%D0%B5%D0%BA%D0%BB%D0%B8%D1%81%D1%82)

Мы будем использовать следующие возможности jQuery:

* Создание элементов: простая форма $("<div/>") и расширенная $("<div/>", {html: ...}).
* Поиск элементов: $(".selector", context) и elem.find(".selector").
* Навигация по DOM: closest, parent…
* Размеры и метрики: offset, width/height, outerWidth/outerHeight, scrollTop/scrollLeft.
* События: обработчики и делегирование с [jQuery.on](http://api.jquery.com/on/), неймспейсы в событиях.
* Манипуляции с DOM/CSS: append/prepend/и т.п., метод css.
* Для задач на анимацию: animate (таких задач мало, так что не обязательно).

**Вёрстка должна быть семантической, использовать соответствующие смыслу информации теги и классы.**

## [Внутренний и внешний интерфейс](http://learn.javascript.ru/internal-external-interface#%D0%B2%D0%BD%D1%83%D1%82%D1%80%D0%B5%D0%BD%D0%BD%D0%B8%D0%B9-%D0%B8-%D0%B2%D0%BD%D0%B5%D1%88%D0%BD%D0%B8%D0%B9-%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81)

* Внутренний интерфейс — это свойства и методы, доступ к которым может быть осуществлен только из других методов объекта.
* Внешний интерфейс — это свойства и методы, доступные снаружи объекта.
* **Приватный метод объявляется как локальная функция.**
* Например, выделим обработчик клика в приватный метод onTitleClick:
* [показать чистый исходник в новом окне](http://learn.javascript.ru/internal-external-interface#viewSource)[Скрыть/показать номера строк](http://learn.javascript.ru/internal-external-interface#noList)[печать кода с сохранением подсветки](http://learn.javascript.ru/internal-external-interface#printSource)

|  |  |
| --- | --- |
| function Menu(options) { | |
| var self = this; |

|  |
| --- |
|  |
| var elem = options.elem; | |

|  |
| --- |
|  |
| elem.on('click', '.menu-title', onTitleClick); | |

|  |
| --- |
|  |
| function onTitleClick(e) { | |

|  |  |
| --- | --- |
| if (elem.hasClass('menu-open')) { | |
| self.close(); |

|  |
| --- |
| } else { |
| self.open(); | |

|  |  |
| --- | --- |
| } | |
| } |

|  |
| --- |
|  |
| self.open = function() { | |

|  |  |
| --- | --- |
| elem.addClass('open'); | |
| }; |

|  |
| --- |
|  |
| this.close = function() { | |

|  |  |
| --- | --- |
| elem.removeClass('open'); | |
| }; |

|  |
| --- |
|  |
| } | |
| Передаю объект анонимный  var foodMenu = new Menu({  elem: $('#food-menu')  });  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  И получаю его свойство elem  var elem = options.elem;  // отмена выделения при клике на меню отличная штука  elem.on('mousedown selectstart', false);  elem.on('click', '.menu-title', onTitleClick); | |

**Анимация**

display и переключают его из текущего

состояния в none и обратно

// скроем все картинки

$('img').hide();

// теперь вернём их на место

$('img').show();

$('img').hide('slow');

// теперь вернём их на место, чуть быстрее

$('img').show(400);

вторым параметром в приведенных методах может быть

callback-функция – она будет выполнена по окончанию анимации элементов:

// скрываем все картинки

$('img').hide('slow', function(){

// опосля отображаем alert

alert("Images was hidden");

});

Идём дальше – у нас на очереди набор методов slide – slideUp(), slideDown() и

slideToggle(). Их поведение схоже с предыдущими функциями, но анимация будет

затрагивать лишь высоту блоков slide.html

fadeIn(*duration, callback*) – изменяет opacity от 0 до предыдущего

fadeOut(*duration, callback*) – изменяет opacity от текущего до 0

// fadeOut()

$('article img').animate({

'opacity':'hide'

})

// fadeIn()

$('article img').animate({

'opacity':'show'

})

fadeToggle(*duration, callback*) – переключатель между In и Out

fadeTo(*duration, opacity, callback*) – изменяет opacity до требуемого значения

animate.

Принимает параметры!

params – CSS свойства – с этим мы уже познакомились

duration – скорость анимации – тоже упоминалась ранее, указывается в

миллисекундах, или используя ключевые слова "fast" или "slow"

easing – указываем какую функцию будем использовать для изменения значений

callback – функция, которая будет вызвана после окончания анимации

(см. пример animate.html):

// останавливаем выполнение текущей анимации

$('#box').stop();

// останавливаем выполнение текущей анимации

// и всех последующих (чистим очередь)

$('#box').stop(true);

// останавливаем выполнение текущей анимации и всех последующих

// но применяем результат текущей

$('#box').stop(true, true);

animate.queue.html

jQuery.fx.off = true; отключить всю анимацию

**Манипуляции с DOM**

after(*content*) — вставляет контент после каждого элемента из выборки, т.е. если

вы встречаете строку $("p").after("<hr/>"), читайте её как «после каждого

параграфа будет вставлена линия»

insertAfter(*element*) — вставляет элементы из выборки после каждого элемента

переданного в качестве аргумента, т.е. если вы встречаете строку

$("<hr/>").insertAfter("p") – читайте её как «линия будет вставлена после

каждого параграфа»

**( ".inner" ).after( "<p>Test</p>" );**

<div class="container">

<h2>**Greetings**</h2>

<div class="inner">**Hello**</div>

<p>**Test**</p>

<div class="inner">**Goodbye**</div>

<p>**Test**</p>

</div>

*— Хм, а я разницы не увидел! — тут всё легко, присмотритесь:*

*$("после чего добавляем").after("что добавляем")*

*$("что добавляем").insertAfter("после чего добавляем")*

before(*content*) — вставляет контент перед каждым элементом из выборки

insertBefore(*element*) — вставляет элементы из выборки перед каждым

элементом переданным в качестве аргумента

append(*content*) — вставляет контент в конец каждого элемента из выборки, т.е.

строку кода $("p").append("<hr/>"), следует читать как «в конец каждого

параграфа будет добавлена линия»

appendTo(*element*) — вставляет выбранный контент в конец каждого элемента

переданного в качестве аргумента: $("<hr/>").appendTo("p") — «линия будет

добавлена в конец каждого параграфа»

*$("куда добавляем").*append*("что добавляем")*

*$("что добавляем").*appendTo*("куда добавляем")*

prepend(*content*) — вставляет контент в начало каждого элемента из выборки

prependTo(*element*) — вставляет выбранный контент в начало каждого элемента

переданного в качестве аргумента

replaceWith(*content*) – заменяет найденные элементы новым содержимым

replaceAll(*target*) – вставляет контент в замен найденному

*$("что").*replaceWith*("на что меняем")*

*$("что").*replaceAll*("вместо чего")*

wrap(*element*) – оборачиваем каждый найденный элемент новым элементом, т.е.

мы конфеты из коробки заворачиваем в фантики

wrapAll(*element*) – оборачивает найденные элементы новым элементом, мы

берём все конфеты, и заворачиваем в один большой фантик

wrapInner(*element*) – оборачивает контент каждого найденного элемента новым

элементом, берём конфеты, убираем фантики, заворачиваем в свой фантик, и сверху

заворачиваем в родной фантик

unwrap() – удаляет родительский элемент у найденных элементов, фантики вон

clone(*withDataAndEvents*) – клонирует выбранные элементы, для дальнейшей

вставки копий назад в DOM, позволяет так же копировать и обработчики событий

detach() – удаляет элемент из DOM, но при этом сохраняет все данные о нём в

jQuery, следует использовать, если надо удалить элемент, а потом вернуть его

обратно

empty() – удаляет текст и дочерние DOM элементы

remove() – удаляет элемент из DOM, насовсем

html() – вернёт HTML заданного элемента

html(*newHtml*) – заменит HTML в заданном элементе

text() – вернёт текст заданного элемента, если внутри элемента будут другие HTML

тэги, то вернётся сборная солянка из текста всех элементов

text(*newText*) – заменит текст внутри выбранных элементов, при попытке вставить

таким образом HTML, будет получен текст, где тэги будут приведены к HTML entities:

*$("div").*text*("Some <strong>text</strong>")*

*>> Some &lt;strong&gt;text&lt;/strong&gt;*

*Получение координат объекта*

offset() – вернёт позицию DOM элемента относительно document'а, данные будут

получены в виде объекта: { top: 10, left: 30 }

offset(*{ top: 10, left: 30 }*) – устанавливаем расположение DOM элемента по

указанным координатам

position() – вернёт позицию DOM элемента относительно родительского элемента

height() – возвращает высоту элемента за вычетом отступов и границ; если у нас

несколько элементов в выборке, вернётся первый; значение, в отличии от метода

css('height'), возвращается без указания единиц измерения

height(*height*) — устанавливает высоту всех элементов в выборке, если значение

высоты передано без указания единиц измерения, то это будут px

// в качестве памятки, взято из мануала

$(window).height(); // высота окна

$(document).height(); // высота HTML документа

width() и width(*width*) – ведут себя аналогично методу height(), но работают с

шириной элемента

*Методы height() и width()* ***не изменяют*** *своего поведения в зависимости*

*от выбранной блочной модели, т.е. они всегда возвращают параметры*

*области внутри margin, padding и border'а элемента.*

innerHeight() и innerWidth() – вернут соответственно высоту и ширину элемента,

включая padding

outerHeight() и outerWidth() – вернут высоту и ширину элемента, включая

padding и border

outerHeight(*true*) и outerWidth(*true*) – высота и ширина, включая padding,

border и margin



scrollLeft() – возвращает значение «проскроленности» по горизонтали первого

элемента из выборки

scrollLeft(*value*) – устанавливает значение горизонтального скрола для каждого

элемента из выборки

scrollTop() – возвращает значение «проскроленности» по вертикали первого

элемента из выборки

scrollTop(*value*) – устанавливает значение вертикального скрола для каждого

элемента из выборки

*Значение scrollTop и scrollLeft поддаются анимации и не работают для*

*спрятанных элементов DOM*

**Работа с формами**

change — изменение значения элемента

submit — отправка формы

<form action="/save/">

<input type="text" name="name" value="Ivan"/>

<select name="role">

<option>User</option>

<option>Admin</option>

</select>

<input type="submit"/>

</form>

$('form').submit(function(){

// чуть позже расскажу подробнее о AJAX

$.post(

$(this).attr('action'), // ссылка куда отправляем данные

$(this).serialize() // данные формы

);

// отключаем действие по умолчанию

return false;

});

serialize() – он в ответе за «сбор» данных с формы в удобном для

передачи данных формате:

name=Ivan&role=Admin

Так же есть метод serializeArray() –он собранные данные представляет в виде объекта:

[

{

name:"name",

value:"Ivan"

},

{

name:"role",

value:"Admin"

},

]

val() – получение значения первого элемента формы из выборки

val(*value*) – установка значение всем элементам формы из выборки

проверки введение текста в формах

if ($(this).find('input[name=user]').val() == '') {

alert('Введите имя пользователя');

return false;

radiobutton'ами

$('input[type=radio][name=choose][value=2]').prop('checked', true)

С checbox'ами чуть-чуть попроще:

$('input[name=check] ').prop('checked', true)

Проверяем «чекнутость» простым скриптом:

$('input[name=check] ').prop('checked')

// или чуть более наглядным способом

$('input[name=check] ').is(':checked')

А если потребуется изменить список? Есть на все случаи жизни:

// возьмём список заранее, поберегу чернила

var $select = $('form select[name=Role]');

// добавить новый элемент в выпадающий список

$select.append('<option>Manager</option>');

// выбрать необходимый элемент

$select.val('Value 1');

// или по порядковому номеру, начиная с 0

$select.find('option:eq(2)').prop('selected', true);

// очищаем список

66

$select.remove('option');

// преобразуем в multiple

// не забываем, что имя такого селекта, должно быть с [], т.е.

// myselect[], иначе сервер получит, лишь одно значение

$('select').attr('size',

$('select option').length

)

$('select').attr('multiple', true)

// всё хорошо – отправляем запрос на сервер

$.post(

$(this).attr('action'), // ссылка куда отправляем данные

$(this).serialize() // данные формы

);

return false;

});