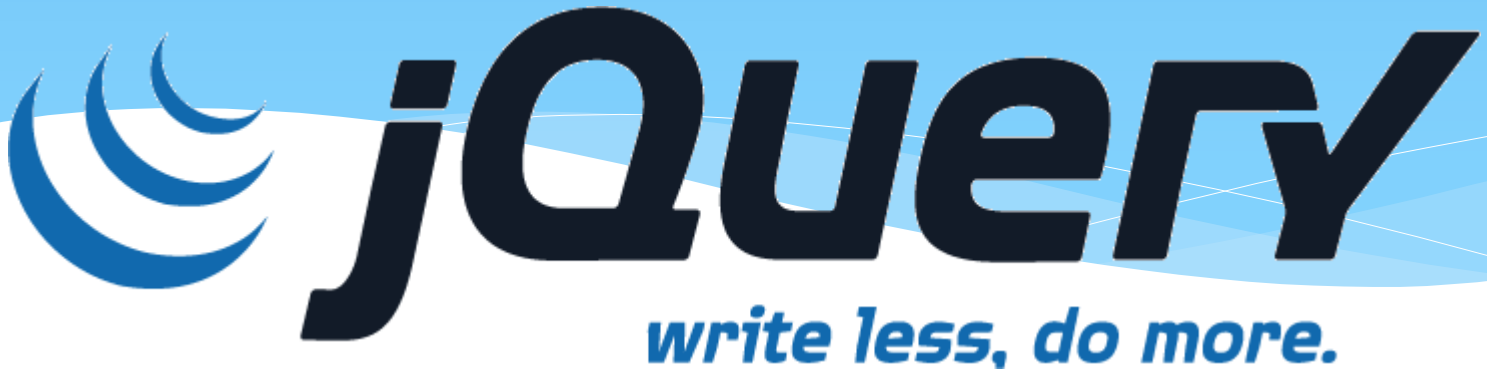
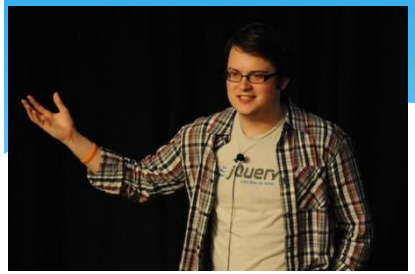
Основы jQuery



История появления jQuery



Возвратимся назад в те далекие времена, когда **jQuery** еще не появился на мировой сцене. Датой возникновения jQuery считается 2006 год. До этого момента разработчикам приходилось тратить много времени на то, чтобы создать динамическую страницу, опираясь лишь на средства Javascript. Куча циклов **while** и **foreach**, хоровод инструкций **if\else** — все это никак не отвечало требованиям оптимизации кода.

В те времена из-за плохой совместимости браузеров нередко приходилось создавать отдельные скрипты для Firefox и IE. Существовало множество способов «обмануть» браузеры, заставить их одинаково реагировать на события. Целью такого подхода являлось сглаживание различий в представлении одной и той же веб-страницы разными браузерами.

Вместо того, чтобы активно изучать Javascript, многие разработчики решили попросту отказаться от его применения, тем самым избежав различных проблем. Однако некоторые энтузиасты не стали безрассудно исключать этот перспективный вариант. Так на горизонте возник **jQuery**.

История появления jQuery

HTML был одной из первых вещей, которую Джон Резиг освоил, когда он только начал заниматься программированием. Резиг программировал на QBasic, когда один его знакомый показал ему, как создать веб-страницу (используя Angelfire), а также основы HTML. Отец подарил ему на Рождество две книги по HTML. Именно тогда, когда он только начал программировать на Visual Basic, HTML и веб-дизайн очень заинтересовали его.

Но страсть к **JavaScript** пришла значительно позже, примерно в 2004 году. Тогда Резиг получал степень в области компьютерных наук и работал на полставки в местной фирме Brand Logic. Он занимался дизайном сайта, в котором создавался пользовательский скроллинг. Джон был разочарован и расстроен, особенно потому, что использовал код других разработчиков, после чего решил серьёзно изучить JavaScript. Изучив, пришёл к выводам, что **JavaScript** — это простой, но изящный язык, который является невероятно мощным для решения многих задач. В течение следующей пары лет Джон создал множество различных JavaScript-приложений, прежде чем закончить создание **jQuery**. Основной целью создания **jQuery** Резиг видел возможность закодировать многоразовые куски кода, которые позволят упростить JavaScript и использовать их так, чтобы не беспокоиться о кросс-браузерных вопросах. Библиотека была представлена общественности на компьютерной конференции «BarCamp» в Нью-Йорке в 2006 году.

С чего начинался jQuery

1. jQuery изначально назывался JSelect.

Но доменное имя **JSelect.com** было занято, а **JQuery.com** — нет. Первая буква в названии стала писаться строчной несколько позже, а сначала это был "**JQuery**".

2. jQuery создавался без системы управления версиями.

Сегодня невозможно представить ни один серьезный open source проект, который бы выпускался без использования **системы управления версиями** (VCS). Но в то время автор jQuery даже и не думал использовать VCS для своего проекта! Справедливости ради отметим, что тогда даже большие проекты не использовали VCS. jQuery перешел на систему контроля версий (это был SVN) только с появлением первых контрибьютеров.

3. Название jQuery было «украдено» из другого проекта.

Название «jQuery» также носила SQL библиотека на Java. Но Джон Резиг узнал о конфликте имен только когда с ним связались создатели «Java-jQuery». Джон вспоминает, что тогда расстроился из-за всей этой путаницы.

С чего начинался jQuery

4. На jQuery сильно повлияли другие проекты с открытым исходном кодом.

В комментариях Джон упоминает несколько проектов, которые оказали наиболее сильное влияние на разработку jQuery:

- Prototype JavaScript framework
- moo.fx
- XPath
- Behaviour.js

Из всех этих проектов до сих пор активен только *Prototype* (последняя версия **1.7.3** от 22.09.15). Остальные проекты больше не разрабатываются. *XPath* перешел в статус рекомендаций W3C еще в 1999 и с тех пор не изменялся, в отличии от HTML и CSS.

5. Первым плагином jQuery стал «JSON for jQuery».

Созданный в январе 2006 года плагин предоставлял простой способ работы с JSON-данными. Также плагин показал, что у jQuery большие перспективы именно из-за расширяемости его функционала с помощью плагинов. В дальнейшем одной из причин успеха jQuery стало большое количество плагинов от сторонних разработчиков.

JSON

JSON (англ. *JavaScript Object Notation*) — текстовый формат обмена данными, основанный на JavaScript и обычно используемый именно с этим языком. Как и многие другие текстовые форматы, JSON легко читается людьми. Несмотря на происхождение от JavaScript (точнее, от подмножества языка стандарта ECMA-262 1999 года), формат считается независимым от языка и может использоваться практически с любым языком программирования. Для многих языков существует готовый код для создания и обработки данных в формате JSON.
Следующий пример показывает JSON-представление объекта, описывающего человека. В объекте есть строковые поля имени и фамилии, объект, описывающий адрес, и массив, содержащий список телефонов. Как видно из примера, значение может представлять собой вложенную структуру.

```
{
    "firstName": "Иван",
    "lastName": "Иванов",
    "address": {
        "streetAddress": "Московское ш., 101, кв.101",
        "city": "Ленинград",
        "postalCode": 101101
    },
    "phoneNumbers": [
        "812 123-1234",
        "916 123-4567"
    ]
}
```

JSON
JavaScript Object Notation

```
<person>
  <firstName>Иван</firstName>
  <lastName>Иванов</lastName>
  <address>
    <streetAddress>Московское ш., 101, кв.101</streetAddress>
    <city>Ленинград</city>
    <postalCode>101101</postalCode>
  </address>
  <phoneNumbers>
    <phoneNumber>812 123-1234</phoneNumber>
    <phoneNumber>916 123-4567</phoneNumber>
  </phoneNumbers>
</person>
```

XML

С чего начинался jQuery

6. JSLint использовался в jQuery для контроля качества.

В своих современных комментариях к коду Джон Резиг отметил места, которые были, по его мнению, недостаточно хороши. Эта проблема в дальнейшем решалась с помощью JSLint — инструменту для анализа JS кода.

7. jQuery изначально не имел поддержки Ajax.

Асинхронные веб-приложения были в моде в начале-середине 2000-х. Сегодня многие веб-приложения работают асинхронно и этим уже никого не удивишь. Но тогда это была революционная идея, и многие веб-разработчики спешили запрыгнуть на подножку этого поезда. Все мы хотели избежать полной перезагрузки наших страниц, и обновлять только ту часть, которая требуется. Удивительно, но jQuery не имел поддержки Ajax в своей первой версии.

8. API jQuery 2006 года до сих пор поддерживается.

Ваши любимые методы jQuery, такие как .css(), .toggle(), .show() и .hide(), были в самой первой версии и все еще поддерживаются API. И за эти 9 лет в них не сломалась совместимость со старыми версиями. «Есть большая вероятность, что если вы возьмете код, который использует jQuery 2006 года, и подключите к нему современный jQuery, то он все еще будет работать», — говорит Джон.

С чего начинался jQuery

9. В коде jQuery не использовались фигурные скобки, если они были не обязательны.

Считается хорошей практикой использовать фигурные скобки в блочных конструкциях, даже когда этого не требует JavaScript. Использование фигурных скобок предотвращает ошибки, особенно при работе в команде. JSLint, о котором упоминалось выше, считает ошибкой отсутствие фигурных скобок. Однако в первой версии jQuery Джон предпочитал опускать опциональные фигурные скобки. «Мне действительно не нравились лишние скобки, — объясняет Джон, — Этот стиль кода мучил всех нас и стал причиной многих логических ошибок в дальнейшем.»

```
if ( !b )
  for ( var j in a )
    this.style[j] = a[j];
else
  this.style[a] = b;
```

```
if ( !b ) {
  for ( var j in a ) {
    this.style[j] = a[j];
  }
}
else {
  this.style[a] = b;
}
```

jQuery === JavaScript?

Поскольку **jQuery** повсеместно распространена, то вы, возможно, не вполне представляете, где заканчивается JavaScript и начинается **jQuery**. Для многих веб-дизайнеров и начинающих разработчиков HTML/CSS, библиотека **jQuery** — это первый контакт с языком программирования JavaScript. Поэтому **jQuery** иногда путают с JavaScript.

1) Давайте оговоримся, что **JavaScript** – это не **jQuery** и даже не сам **DOM API**. **jQuery** – это сторонняя свободная библиотека, написанная на JavaScript и поддерживаемая целым сообществом разработчиков. Кроме того, jQuery не относится к числу стандартов тех организаций (напр., W3C), которые пишут спецификации HTML, CSS или DOM.

2) Не забывайте, что **jQuery** служит прежде всего как «сахар» и используется поверх DOM API. Этот сахар помогает работать с интерфейсом DOM, который печально известен своей сложностью и обилием багов.

3) **jQuery** – это просто полезная библиотека, которой можно пользоваться при написании сценариев для HTML-элементов. На практике большинство разработчиков прибегают к ней при DOM-скриптинге, поскольку ее API позволяет решить больше задач меньшим количеством кода.

4) Библиотека jQuery и ее плагины используются разработчиками так широко, что такой код часто нахваливают как самые востребованные сценарии во всем Вебе.

Введение в jQuery

jQuery — библиотека JavaScript, фокусирующаяся на взаимодействии JavaScript и HTML. **Библиотека jQuery** помогает легко получать доступ к любому элементу DOM, обращаться к атрибутам и содержимому элементов DOM, манипулировать ими. Также библиотека jQuery предоставляет удобный API для работы с AJAX. Сейчас разработка jQuery ведется командой jQuery во главе с Джоном Резигом. Две базовые концепции, на которых основана jQuery, таковы: **“найди и сделай”** и **“пиши меньше, делай больше”**.

Текущие версии: **1.12.4** и **3.2.1** (*not supported IE6,7,8*) (от 20 Марта 2017 года)

jQuery библиотека содержит следующий функционал:

- *операции с HTML/DOM (манипулирование компонентами HTML/DOM)*
- *операции с CSS-селекторами*
- *HTML-обработчики событий*
- *Эффекты анимации*
- *AJAX*
- *Utilities*

jQuery упрощает работу с **JavaScript**, а также вызовы **AJAX** и **DOM**-манипуляции. Есть много библиотек **JavaScript**, но **jQuery** , является самой популярной и используемой за счет своей расширяемости.

Сравнение кода

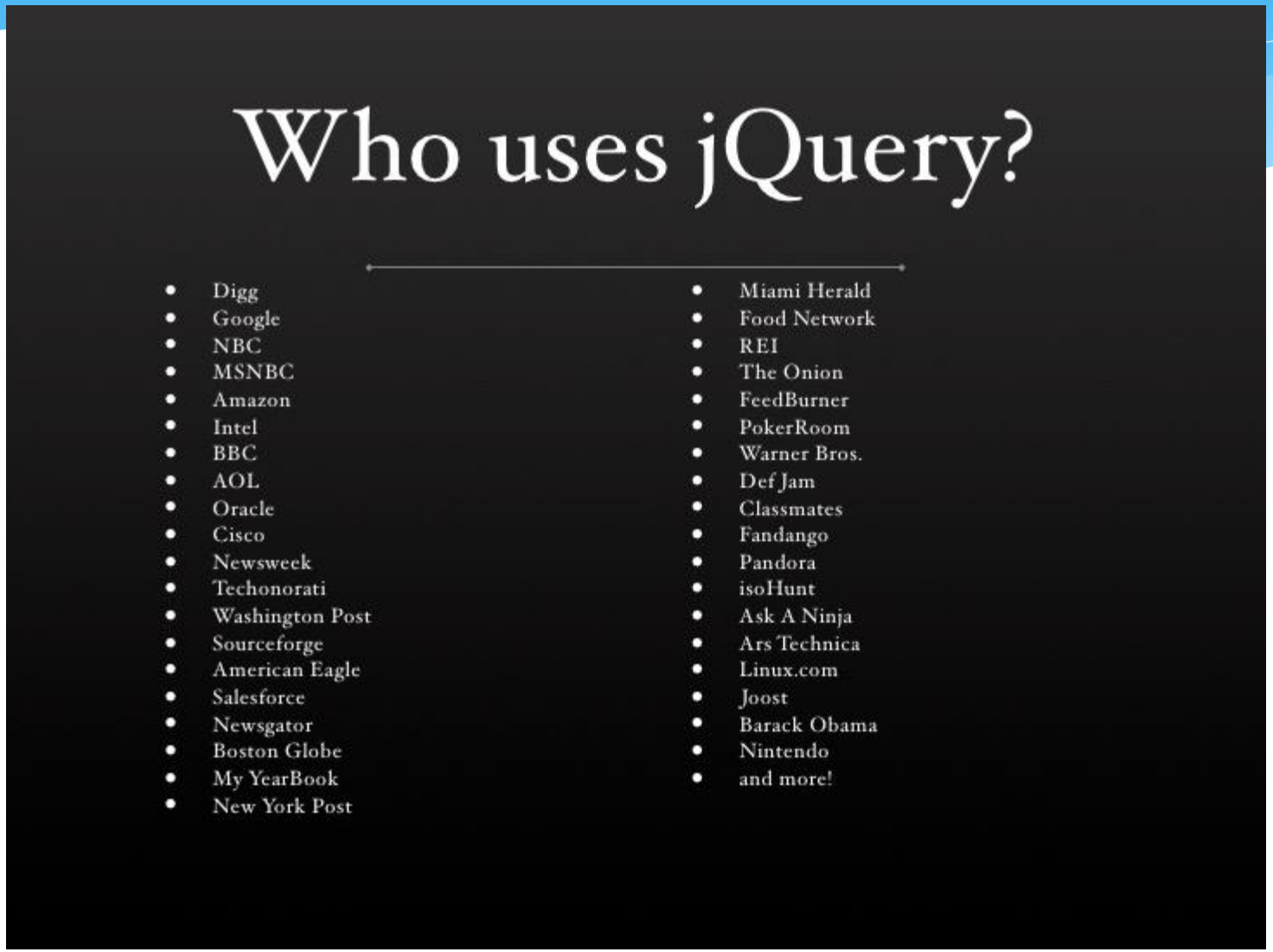
Name	JavaScript	jQuery

Events	document.addEventListener('DOM', function() {
	// code
	});
	
	[].forEach.call(document.querySelectorAll('a'), function(el) {
	el.addEventListener('click', function() {
	// code
	})
	});
		$(document).ready(function() {
		// code
		});
		
		$('a').click(function() {
		// code
		});

Selectors	var divs = document.querySelectorAll('div')	var divs = $('div');
Attributes	document.querySelector('img').setAttribute('alt', 'My image')	$('img').filter(':first').attr('alt', 'My image')

Manipulation	document.body.appendChild(document.createElement('p'));
	
	var wrap = document.getElementById('wrap');
	while (wrap.firstChild)
	wrap.removeChild(wrap.firstChild)
		$('body').append($('<p/>'))
		
		$('#wrap').empty();

Кто использует jQuery?



Начало работы с библиотекой jQuery

Библиотеку **jQuery** можно скачать с сайта **http://jquery.com**, а можно вставить в документ, используя известные интернет-адреса:

- **По адресу** ***http://code.jquery.com/jquery-latest.js*** – доступна всегда последняя версия.
- **С Google:** ***https://developers.google.com/speed/libraries/devguide?hl=ru#jquery*** можно загрузить любую из не слишком старых версий. Синтаксис такой: src="http://ajax.googleapis.com/ajax/libs/jquery/1.8.3/jquery.min.js", где 1.8.3 — версия, причём можно указать её приблизительно: 1.8 означает последнюю версию вида 1.8.*, а 1 — последнюю версию вида 1.*. Файл jquery.min.js обозначает сжатый код, а jquery.js — несжатый, для удобства отладки.
- ***Либо с Microsoft CDN:*** ***src="http://ajax.aspnetcdn.com/ajax/jQuery/jquery-3.1.1.js"***

jQuery синтаксис

Базовая команда для библиотеки выглядит как:

$(селектор).action(), где

- **$** - предписание использовать jQuery;
- **(селектор)** - это "запрос или элементы поиска" в HTML элементах страницы;
- **action()** - это действия, которые должны быть выполнены над найденными элементами (это те элементы, которые удовлетворяют условиям селектора).

Например:

$(this).hide() – скрывает текущий элемент (где this – это указатель на текущий элемент, позволяет делать код универсальным за счет того, что не надо писать здесь имя или id элемента, над которым будет производится действие hide()).

$("p").hide() – скрывает все <p> элементы на странице.

$(".test").hide() – скрывает все элементы на странице, которые ассоциированы с классом "test".

$("#test").hide() – скрывает все элементы на странице, у которых id="test".

jQuery синтаксис

Работу с jQuery можно разделить на 2 типа:

Получение jQuery-объекта с помощью функции $().

Например, передав в неё CSS-селектор, можно получить jQuery-объект всех элементов HTML, попадающих под критерий и далее работать с ними с помощью различных методов jQuery-объекта. В случае, если метод не должен возвращать какого-либо значения, он возвращает ссылку на jQuery объект, что позволяет вести цепочку вызовов методов согласно концепции текучего интерфейса (*fluent interface*).

Вызов глобальных методов у объекта $

Например, удобных итераторов по массиву.

Как же все-таки работает **jQuery**?

```
<head>
<script type="text/javascript" src="jquery.js"></script>

<script type="text/javascript">
$(document).ready(function(){

    $(".button").click(function(){
        $("#panel").slideDown("slow");
    });

});
</script>
</head>
```

подключаем jQuery

Событие "ready" (функция будет выполнена, когда DOM будет готов

Эту часть можно вынести во внешний файл

К чему Вы хотите привязать Вашу функцию?
Это может быть class, ID, selector (DIV,H1,P...)

Эта функция будет вызвана по событию "click" на элементе с классом "button"

Что же будет происходить с #panel?
Элемент будет медленно опускаться вниз

Чем мы будем оперировать?
Элементом с ID = panel

```
$("#panel")
```

Для квотирования могут использоваться как одинарные так и двойные кавычки:
("".class"") == ('.class')

Текучий интерфейс (*fluent interface*)

Текучий интерфейс (**англ.** *fluent interface*) в разработке программного обеспечения — способ реализации объектно-ориентированного API, нацеленный на повышение читабельности исходного кода программы.

Текучий интерфейс хорош тем, что упрощается множественный вызов методов одного объекта. Обычно это реализуется использованием цепочки методов, передающих контекст вызова следующему звену (но текучий интерфейс влечет за собой нечто большее, чем просто цепочку методов). Обычно, этот контекст:

- *определен с помощью значения, возвращаемого методом;*
- *наследуется (в качестве нового контекста используется предыдущий);*
- *прекращается возвращением ничего не значащего значения (void).*

Такой стиль косвенно полезен повышением наглядности и интуитивности кода. Однако может весьма пагубно сказаться на отладке, если цепочка действует как одно выражение, куда отладчик не всегда может установить промежуточную точку останова.

Пример реализации класса с текучим интерфейсом в JavaScript

```
var Car = (function(){

    var speed, color, doors, pub;

    function setSpeed(new_speed) {
        speed = new_speed;
        return pub;
    }

    function setColor(new_color) {
        color = new_color;
        return pub;
    }

    function setDoors(new_doors) {
        doors = new_doors;
        return pub;
    }

    pub = {
        'setSpeed': setSpeed,
        'setColor': setColor,
        'setDoors': setDoors,
    };

    return pub;

})

// Обычная реализация
myCar2 = Car();
myCar2.setSpeed(100);
myCar2.setColor('blue');
myCar2.setDoors(5);

// Текучий интерфейс
myCar = Car();
myCar.setSpeed(100).setColor('blue').setDoors(5);
```

Текучий интерфейс (*fluent interface*)

Пример реализации класса с текучим интерфейсом в jQuery

```
var $ = function(selector) {
    if(this.$) {
        return new $(selector);
    }
    if(typeof selector == "string") {
        this.init = document.getElementById(selector);
    }
};

$.prototype = {
    text: function(text) {
        if(!text){
           this.init.innerHTML;
        }
        this.init.innerHTML = text;
        return this;
    },
    css: function(style) {
        for(var i in style){
           this.init.style[i] = style[i];
        }
        return this;
    }
};
//пример использования:
$('div').text('div').css({color: "red"});
```

Событие **Ready** у объекта страницы **Document**

Вы, заметите, что в большинстве примеров jQuery-методы находятся внутри события документа **Ready()**:

```
$(document).ready (function(){
// jQuery-методы размещаем здесь...
});
```

Это необходимо для предотвращения любых срабатываний jQuery - кода, прежде чем документ не закончит полную загрузку.

Это хорошая практика, чтобы дождаться, пока документ будет полностью **загружен** и **готов** *до работы с ним*. Это также позволяет вам сформировать свой JavaScript код в головной части, прежде чем тело документа.

Селекторы **#id** и **.class**

Селектор jQuery **#id** использует **id** атрибут в HTML-тегах, чтобы найти определенный элемент. **Id** должен быть уникальным внутри всей страницы, если вы хотите найти с его помощью конкретный уникальный элемент.

Чтобы найти элемент с помощью **id**, то перед названием искомого идентификатора ставится знак **#**, например:

```
$("#test")
```

Селектор jQuery **class** находит элементы определенного класса.
Для поиска элементов определенного класса указывается перед названием точка, например:

```
$(".test")
```

Другие примеры jQuery селекторов

Пример	Описание
$("*")	Будут выбраны все элементы присутствующие на странице.
$("#el1")	Будет выбран элемент с id=el1.
$(".el1")	Будут выбраны все элементы на странице с class=el1.
$("div")	Будут выбраны все элементы div на странице.
$("div, #el1, .el1")	Будут выбраны все элементы div, элемент с id="el1" и все элементы с class="el1" на странице.
$("div.el1")	Будут выбраны все элементы div на странице атрибут class которых равен el1.
$("p > div")	Будут выбраны все элементы потомки div родительского элемента p.
$("[src]")	Будут выбраны все элементы на странице имеющие атрибут src.
$("[src='wisdomweb.gif']")	Будут выбраны все элементы на странице со значениями атрибута src="wisdomweb.gif".
$("[src!='wisdomweb.gif']")	Будут выбраны все элементы на странице со значениями атрибута src не равными "wisdomweb.gif".
$("[src^='wisdomweb']")	Будут выбраны все элементы на странице со значениями атрибута src начинающимися на wisdomweb (например wisdomweb.ru, wisdomweb.gif и т.д.).
$("[src$='.gif']")	Будут выбраны все элементы на странице со значениями атрибута src заканчивающимися на .gif.
$("[src*='wisdomweb']")	Будут выбраны все элементы на странице со значениями атрибута src содержащими wisdomweb.
$(":input")	Будут выбраны все элементы input на странице.
$(":button")	Будут выбраны все элементы input на странице с атрибутом type=button.
$(":text")	Будут выбраны все элементы input на странице с атрибутом type=text.
$(":password")	Будут выбраны все элементы input на странице с атрибутом type=password.
$(":radio")	Будут выбраны все элементы input на странице с атрибутом type=radio.
$(":checkbox")	Будут выбраны все элементы input на странице с атрибутом type=checkbox.
$(":reset")	Будут выбраны все элементы input на странице с атрибутом type=reset.

Другие примеры jQuery селекторов

Пример	Описание
$(":image")	Будут выбраны все элементы input на странице с атрибутом type=image.
$(":file")	Будут выбраны все элементы input на странице с атрибутом type=file.
$("div:first")	Будет выбран первый div элемент на странице.
$("div:last")	Будет выбран последний div элемент на странице.
$("li:even")	Будут выбраны все элементы списка с четными индексами. Так как нумерация индексов элементов начинается с 0, то по сути будут выбраны нечетные элементы т.е. 1й, 3й, 5й элементы списка и т.д.
$("li:odd")	Будет выбраны все элементы с нечетными индексами. Так как нумерация индексов элементов начинается с 0, то по сути будут выбраны четные элементы т.е. 2й, 4й, 6й элементы списка и т.д.
$("li:eq(3)")	Будет выбран 4 элемент списка (нумерация элементов в списке начинается с нуля).
$("li:gt(1)")	Будет выбраны все элементы списка индекс которых больше 1 (т.е. все элементы списка начиная с 3 элемента).
$("li:lt(2)")	Будет выбраны все элементы списка индекс которых меньше 2 (т.е. первые 2 элемента списка).
$(":header")	Будет выбраны все элементы на странице являющиеся заголовками (т.е. элементы h1, h2, h5 и т.д.).
$(":animated")	Будет выбраны все анимированные элементы, которые находятся на странице.
$(":contains('wisdomweb')")	Будет выбраны все элементы содержащие строку wisdomweb.
$(":empty")	Будет выбраны все элементы не имеющие узлов потомков.
$(":hidden")	Будет выбраны все скрытые элементы на странице.
$(":visible")	Будет выбраны все видимые элементы находящиеся на странице.

Другие примеры jQuery селекторов

Пример	Описание
$("*")	Выбирает все элементы
$(this)	Выбирает текущий HTML-элемент
$("p.intro")	Выбирает все **<p>** элементы с class="intro"
$("p:first")	Выбирает первый **<p>** элемент
$("ul li:first")	Выбирает первый ** ** элемент из первого ** **
$("ul li:first-child")	Выбирает первый ** ** элемент из каждого ** **
$("[href]")	Выбирает все элементы с атрибутом **href**
$("a[target='_blank']")	Выбирает все **<a>** элементы с целевым атрибутом, чье значение равно "_blank"
$("a[target!='_blank']")	Выбирает все **<a>** элементы с целевым атрибутом, чье значение НЕ равно "_blank"
$(":button")	Выбирает все **<button>** элементы и **<input>** элементы с type="button"
$("tr:even")	Выбирает все чётные **<tr>** элементы
$("tr:odd")	Выбирает все нечётные **<tr>** элементы

Пример 1:

Задача: Выбрать все элементы li и обвести их черной однопиксельной рамкой. Элементам li элемента ul с классом pr присвоить фоновый цвет #aeafae

Решение:

```
$(function () {
        $('li', 'ul').css('border','1px solid #000000');
        $( "li", "ul.pr" ).css('background','#aeafae');
 });
```

Пример 2:

Задача: Создать страницу с текстом. По клику этот текст отображается в модальном окне (alert). Для смены текста в jQuery существует метод text(). Если ему не передавать никаких параметров (ничего не писать в круглых скобках), то он просто вернет текст, содержащийся в элементе, к которому применен. Если же в скобках что то указать, то текст элемента будет изменен на значение, указанное в параметрах метода.

Решение:

```
$(function () {
        $('.wrap p').click(function() {
             $('.wrap p').text('   а это наш текст ');
          });
});
```

Пример 3:

Задача: Скрыть блок с классом first, содержащий текстовый блок, если будет осуществлен клик на этом блоке.

Решение:

```
$(function () {
        $('div.first').click(function() {
                $(this).hide();
        });
});
```

Методы манипуляции элементами

`.is()` //вернёт значение **true**, если среди отобранных ранее элементов присутствует хотя бы один, указанный в параметре и **false** - если такого элемента нет.

`.not()` // исключит из отобранных ранее элементы, которые указаны в параметре.

`.eq(index)` // выберет элемент из отобранных ранее, с указанным в параметре индексом.

`.slice(startIndex, endIndex)` // выберет элементы из ранее отобранных, индекс которых находится в диапазоне, указанном в параметре (отсчет начинается с 0). При указании отрицательных значений в параметре, отсчет начинается с 1. Второй параметр не обязателен.

`.filter()` // выберет элемент из отобранных ранее, которые соответствуют указанному в параметре условию.

`.filter(fn)` // выберет элементы из отобранных ранее, которые соответствуют значению, возвращенному из функции (при условии **true**), указанной в параметре.

jQuery «+» и «-»

+

1. Облегчает манипулирование моделью документа DOM.
2. Множество различных эффектов
3. Выполнение Ajax-запросов
4. Кросс-браузерная совместимость
5. Модульность jQuery, Вы можете нарастить функционал, подключая модули.
6. Простота использованиям и синтаксиса
7. Легкость разработки
8. Цепочки методов
9. CSS-селекторы – как объекты DOM
10. Великолепная документация

-

1. Скорость выполнения. Да, чистый JavaScript работает быстрее (в умелых руках), но говорить о быстродействии не совсем правильно (растет скорость компьютеров, да и сами разработчики постоянно наращивают быстродействие самой библиотеки)
2. Это размер библиотеки. Размер jQuery библиотеки порядка 84 кБ. Не так уж и много, но все же. Опять же это спорный минус при современных скоростях интернета или при использовании CDN.