Антон Шевчук

jQuery

учебник для начинающих

1.0.6
2016

Об авторе

Антон Шевчук родился в 1983 году в небольшом городке Ахтырка, Украина, отец — инженер-конструктор, мать — врач-терапевт. В 2000-м году закончил общеобразовательную школу №2. В том же году поступил в Харьковский Национальный Университет Радиоэлектроники на факультет КИУ на специальность системное программирование. Во время учебы проживал в общежитии №8 и был там известен под ником «Dark Lord». После получения диплома в 2005-м, остался в Харькове и устроился на работу в компанию NIX Solutions Ltd.

Ведет блог своего имени http://anton.shevchuk.name. На данный момент, продолжает работать в компании NIX Solutions Ltd в должности технического эксперта отдела PHP. Знает PHP, увлекается JavaScript, завёл себе open-source проекты. Является одним из организаторов конференций ThinkPHP в Харькове.

О книге

Данный учебник подойдет как для начинающих web-разработчиков, так и для продвинутых JavaScript программистов, желающих освоить новую для себя библиотеку.

В учебнике принято следующее форматирование:

> **важно** — запомни, запиши, и всегда применяй

> `<html>` — исходный код в тексте выделяется monotype шрифтом

```
// листинг исходного кода будет набран monotype шрифтом с отступом
// также будет использоваться элементарная подсветка синтаксиса

function () {
      setTimeOut("alert('Hello World!')", 5000);
}
```

> *информация, которую стоит принять к сведению,*
> *или, возможно, лишь прочитать и улыбнуться,*
> *будет оформлена в блок текста с отступом и выделена курсивом*

Примеры кода вы сможете найти в репозитории автора на GitHub'e — https://github.com/AntonShevchuk/jquery-for-beginners, вживую пощупать — на персональном блоге http://anton.shevchuk.name/book/code/

Условия распространения

- Данная книга распространяется бесплатно и доступна для загрузки на странице http://anton.shevchuk.name/jquery-book/
- Если хотите поделиться книгой с другом — отправьте ему ссылку на страницу http://anton.shevchuk.name/jquery-book/, любые иные способы «поделиться» нежелательны
- За полное или частичное копирование материалов без согласования с автором вам будет стыдно

Оглавление

0% О HTML, CSS и JavaScript

Начнём знакомство с jQuery с повторения (или изучения) основ правильного употребления связки HTML и CSS с небольшой примесью JavaScript.

> *Если не хотите упасть в грязь лицом перед коллегами — то не пропускайте данную главу «мимо ушей».*

HTML — о нём стоит помнить две вещи – **семантический** и **правильный**.

Семантическая вёрстка

Семантическая вёрстка HTML документа подразумевает использование тегов по прямому назначению, т.е. если вам необходим заголовок — то вот тег `<h1>` и собратья, необходима табличное представление данных — используйте тег `<table>` и только его.

> *Иногда, избавляясь от табличной верстки, доходит до абсурда, и тег `<table>` становится изгоем, и его перестают использовать даже для табличного представления данных, не стоит повторять эту ошибку.*

Забегая чуть-чуть вперёд, стоит упомянуть теги из спецификации HTML5: `<article>`, `<aside>`, `<header>`, `<footer>`, `<menu>`, `<section>` и т.д. — используйте их, не бойтесь.

> *Не бояться — это правильно, но использовать тоже надо с умом, рекомендую ресурс http://html5doctor.com/ — очень хорошо и подробно расписано о нововведениях спецификации HTML5.*
>
> *И еще парочка интересных ресурсов в нагрузку:*
>
> - *http://htmlbook.ru/html5 — неплохо, и на русском*
> - *http://www.html5rocks.com/en/ — тут целое сообщество*

Старайтесь избегать избыточных элементов на странице, большинство HTML страниц грешат лишними блочными элементами:

```
<div id="header">
	<div id="logo">
		<h1><a href="/">Мой блог</a></h1>
	</div>
	<div id="description">
		<h2>Тут я делюсь своими мыслями</h2>
	</div>
</div>
```

Данную конструкцию можно легко упростить, и при этом код станет более читаемым, изменения в CSS будут минимальными (или даже не потребуются):

```
<header>
	<h1>
		<a href="/">Мой блог</a>
	</h1>
	<h2>Тут я делюсь своими мыслями</h2>
</header>
```

В английском языке есть термин «divits» – сим термином награждают HTML- разметку с чрезмерным использованием div'ов без потребности, я же обзываю такие творения «дивными». Обычно таким грешат новички, которые для применения стилей CSS оборачивают элементы в div'ы, что и приводит к их размножению без нужды.

Ещё одним обязательным пунктом для создания «правильного» HTML является использование названий классов и идентификаторов, которые однозначно говорят нам о содержимом элемента, а не о каких либо нюансах оформления, приведу пример:

Плохо	
`red, green и т.д.`	в какой-то момент захотите перекрасить, и элемент с классом «red» будет синего цвета
`wide, small и т.д.`	сегодня широкий, а завтра?
`h90w490`	наверное, это элемент с высотой 90px и шириной 490px, или я ошибаюсь?
`b_1, ax_9`	эти название тоже ни о чем не говорят

`color1, color2 и т.д.`	иногда встречается для «скинованных» сайтов, но создают такие классы из лени
`element1...20`	такое тоже встречается, и ничем хорошим не пахнет

Ну и примеры правильного именования:

Хорошо	
logo, content	логотип, основной контент
menu, submenu	меню и подменю
even, odd	чётный и нечётный элементы списка
paginator	постраничная навигация
copyright	копирайт

Есть ещё один момент – это форматирование HTML и CSS кода, я не буду заострять на нём внимание, но весь код в книге будет отформатирован отступами, и, возможно, это даст свои плоды в ваших творениях.

Валидный HTML

Зеленый маркер W3C validator'a – это правильно, и к этому надо стремится, так что не забывайте закрывать теги, да прописывать обязательные параметры, приведу пример HTML кода, в котором допущено 6 ошибок (согласно спецификации HTML5), найдите их:

```
<!DOCTYPE html>
<html>
<body>
      <h2>Lorem ipsum
      <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
      Nunc urna metus, ultricies eu, congue vel, laoreet id, justo.
      Aliquam fermentum adipiscing pede. Suspendisse dapibus ornare
      quam. Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
      <p>
      <a href="/index.php?mod=default&act=image">
            <img src="/img001.jpg">
      </a>
</body>
</html>
```

Если сомневаетесь в своих знаниях, то проверьте данный код на странице https://validator.w3.org/nu/#textarea

CSS-правила и селекторы

Теперь приступим к CSS, и начнём, пожалуй, с расшифровки аббревиатуры CSS это Cascading Style Sheets, дословно «каскадная таблица стилей», но:

— *Почему же она называется каскадной?* — этот вопрос я часто задаю на собеседованиях претендентам. Ответом же будет аналогия, ибо она незыблема как перпендикулярная лягушка: представьте себе каскад водопада, вот вы стоите на одной из ступенек каскада с чернильницей в руках, и выливаете ее содержимое в воду — вся вода ниже по каскаду будет окрашиваться в цвет чернил. Применив эту аналогию на HTML — создавая правило для элемента, вы автоматически применяете его на все дочерние элементы (конечно, не все правила так работают, но о них позже) — это наследование стилей. Теперь, если таких умников с чернильницей больше чем один, и цвета разные, то в результате получим смешение цветов, но это в жизни, а в CSS работают правила приоритетов, если кратко и по делу:

- самый низкий приоритет имеют стили браузера по умолчанию — в разных браузерах они могут отличаться, поэтому придумали CSS Reset (гуглится и юзается), и все будут на равных
- чуть важнее — стили заданные пользователем в недрах настроек браузера, в жизни встречаются редко
- самые важные — стили автора странички, но и там всё идёт по порядку
  - самый низкий приоритет у тех, что лежат во внешнем подключённом файле
  - затем те, что встроили внутрь HTML с помощью тега `<style>`
  - потом те, что захардкодили плохие люди (не вы, вы так делать не будете) в атрибуте «`style`»
  - самый высокий приоритет у правил с меткой «`!important`»
  - при равенстве приоритетов, тапки у того, кто объявлен последним

Если голова ещё не бо-бо, то я также упомяну, что при расчёте, чьи же правила главней, ещё анализируется специфичность селекторов, и тут считается следующим образом:

> расчёт происходит по четырём весовым позициям `[0:0:0:0]`
>
> стили заданные в атрибуте «`style=""`» имеют наибольший приоритет и получают еденицу по первой позиции — `[1:0:0:0]`
>
> за каждый идентификатор элемента — `[0:1:0:0] (#id)`
> за каждый класс, либо псевдо класс — `[0:0:1:0] (.my, :pseudo)`
> за каждый тег — `[0:0:0:1] (div, a)`
> при этом `[1:0:0:0] > [0:x:y:z] > [0:0:x:y] > [0:0:0:x]`.

Пример селекторов, выстроенных по приоритету (первые важнее):

```
#my p#id – [0:2:0:1]
#my #id  – [0:2:0:0]
#my p    – [0:1:0:1]
#id      – [0:1:0:0]

.wrapper .content p – [0:0:2:1]
.content div p      – [0:0:1:2]
.content p          – [0:0:1:1]
P                   – [0:0:0:1]
```

HTML-код для иллюстрации специфичности из предыдущего примера (см. css.priority.html):

```
<div class="wrapper">
      <div id="my" class="content">
            <p id="id">
                  Lorem ipsum dolor sit amet, consectetuer...
            </p>
      </div>
</div>
```

При равенстве счета — последний главный.

> *Говорят, что правило с 255 классами будет выше по приоритету, нежели правило с одним «`id`», но я надеюсь, такого кода в реальности не существует*

Вот такое краткое вступительное слово, но пора вернуться к jQuery. Так вот, работая с jQuery, вы должны «на отлично» читать правила CSS, а также уметь составлять CSS-селекторы для поиска необходимых элементов на странице. Но давайте обо всем по порядку, возьмём следующий простенький пример вполне семантического HTML (см. html.example.html):

```
<!DOCTYPE html>
<html dir="ltr" lang="en-US">
<head>
	<meta charset="UTF-8"/>
	<title>Page Title</title>
	<link rel="profile" href="http://gmpg.org/xfn/11"/>
	<style type="text/css">
		body {
			font: 62.5%/1.6 Verdana, Tahoma, sans-serif;
			color: #333333;
		}
		h1, h2 {
			color: #ff6600;
		}
		header, main, footer {
			margin: 30px auto;
			width: 600px;
		}
		#content {
			padding: 8px;
		}
		.box {
			border:1px solid #ccc;
			border-radius:4px;
			box-shadow:0 0 2px #ccc;
		}
	</style>
</head>
<body>
	<header>
		<h1>Page Title</h1>
		<p>Page Description</p>
	</header>
	<main id="content" class="wrapper box">
	<article>
		<h2>Article Title</h2>
		<p>
		Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
		Nunc urna metus, ultricies eu, congue vel, laoreet...
		</p>
	</article>
	<article>
		<h2>Article Title</h2>
		<p>
		Morbi malesuada, ante at feugiat tincidunt, enim massa
		gravida metus, commodo lacinia massa diam vel eros...
		</p>
	</article>
	</main>
	<footer>&copy;copyright 2016</footer>
</body>
</html>
```

Это пример простого и правильного HTML5 с небольшим добавлением CSS3. Давайте разберём селекторы в приведённом CSS-коде (я умышленно не выносил CSS в отдельный файл, ибо так наглядней):

`body`	— данные правила будут применены к тегу `<body>` и всем его потомкам, запомните: настройки шрифтов распространяются вниз «по каскаду»
`h1,h2`	— мы выбираем теги `<h1>` и `<h2>`, и устанавливаем цвет шрифта для данных тегов и их потомков
`#content`	— выбираем элемент с «`id="content"`», настройки отступов не распространяются на потомков, они будут изменяться только для данного элемента
`.box`	— выбираем элементы с «`class="box"`», и изменяем внешний вид границ элементов с заданным классом

Теперь подробнее и с усложнёнными примерами:

`h1`	ищем элементы по имени тега
`#container`	ищем элемент по идентификатору «`id=container`» (**идентификатор уникален**, значит, на странице он должен быть только один)
`div#container`	ищем `<div>` с идентификатором container, но предыдущий селектор работает быстрее, но этот важнее
`.news`	выбираем элементы по имени класса «`class="news"`»
`div.news`	все элементы `<div>` с классом «`news`» (так работает быстрее в IE8, т.к. в нём не реализован метод «`getElementsByClassName()`»)
`#wrap .post`	ищем все элементы с классом «`post`» внутри элемента с «`id = wrap`»
`.cls1.cls2`	выбираем элементы с двумя классами «`class="cls1 cls2"`»
`h1,h2,.posts`	перечисление селекторов, выберем всё перечисленное
`.post > h2`	выбираем элементы `<h2>`, которые являются непосредственными потомками элемента с классом «`post`»
`a + span`	будут выбраны все элементы `<span>` следующие сразу за элементом `<a>`

`a[href^=http]`	будут выбраны все элементы `<a>` у которых атрибут «`href`» начинается с http (предположительно, все внешние ссылки)

Это отнюдь не весь список, описание же всех CSS3 селекторов можно найти на соответствующей страничке W3C:
http://www.w3.org/TR/css3-selectors/

*40% задач, которые вы будете решать с помощью jQuery, сводятся к поиску необходимого элемента на странице, так что **знание CSS селекторов обязательно**. Вот еще кусочек CSS для тренировки, напишите соответствующий ему HTML (это тоже вопрос с собеседования ;):*

```
#my p.announce, .tt.pm li li a:hover+span { color: #f00; }
```

Пишите прям тут:

CSS. Погружение

> *Этот раздел будет полезен начинающим верстальщикам, и тем, кто захочет сделать чуть больше, нежели вывод окна сообщения по клику.*

О форматировании

Нет, я не властен над собой, и таки приведу в качестве примера CSS форматирование, которое я использую:

```
/*header*/
header {
	margin-bottom: 16px;
	font-weight: 400;
}
	header h1 {
		color: #999;
	}
	header p {
		font-size: 1.4em;
		margin-top: 0;
	}
/*/header*/
```

Почему это хорошо:

- такой CSS легко читается
- есть идентификатор начала и конца блока (можно быстро найти необходимую часть даже в очень большом CSS файле используя поиск по метке «*header»)
- подобное форматирование явно указывает на вложенность элементов
- и можно легко проследить наследование свойств

Я не настаиваю на своём варианте, но хотелось, чтобы вы приняли на вооружение один из многих стандартов форматирования, и всегда следовали ему.

> *Когда станете матёрыми front-end разработчиками, то познаете всю силу CSS-препроцессоров, а пока слушайте и запоминайте.*

Именование классов и идентификаторов

Я уже затрагивал эту тему, когда рассказывал о релевантности HTML, так вот – имена классов могут быть даже такими: «`b-service-list__column b-service-list__column_right`» и это будет круто, и «must be» – но лишь в рамках действительно больших проектов, и собственно чего я распинаюсь, задам исходную точку для изучения – информации там еще на одну книгу ;):

— «Что такое БЭМ» — [https://ru.bem.info/methodology/]

> *Обязательно ознакомьтесь с принципами БЭМ – это полезно для расширения кругозора, и прокачки скилов*

О цветах

В WEB используется цветовая модель RGB, я не открою вам Америку, если расскажу, что красный цвет можно записать не только как «`red`», но и ещё несколькими способами:

```
p { color: red }
p { color: #ff0000 }
p { color: #f00 } /* сокращенная запись, экономит 3 байта */
p { color: rgb(255, 0, 0) }
```

> *Теперь вы без запинки должны назвать цвета `#f00`, `#0f0`, `#00f`, а те, у кого по рисованию было «отлично», назовут и `#ff0`, `#0ff` и `#f0f` ;)*

С появлением CSS3, указывая цвет, мы также можем задать значение α-канала, т.е. прозрачность:

```
p { color: rgba(255, 0, 0, 1) }   /* обычный текст */
p { color: rgba(255, 0, 0, 0.5) } /* полупрозрачный текст */
```

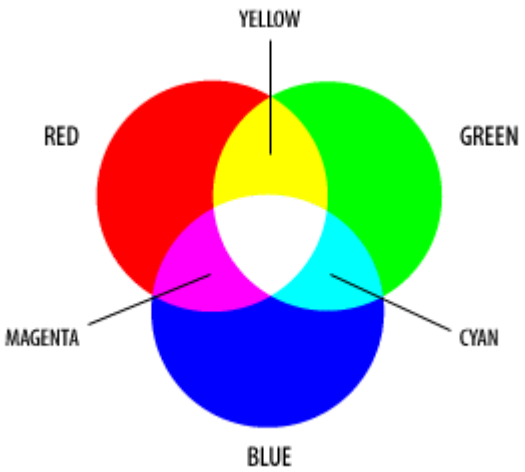
Ещё одна примочка CSS3 – это возможность использования цветовой модели HSL (hue saturation lightness – тон, насыщенность и светлота) и HSLA (HSL + α-канал):

```
p { color: hsl(  0, 100%, 50%) } /* красный */
p { color: hsl(120, 100%, 50%) } /* зелёный */
p { color: hsl(240, 100%, 50%) } /* синий */
p { color: hsla( 0, 100%, 50%, 0.5) } /* полупрозрачный красный */
```

Для перевода из HSL в RGB существует простой алгоритм, но пока не стоит им себя грузить.

Да кто этим HSL пользуется? Не морочьте себе голову!

Для тех, кого вопрос со смешанием каналов RGB поставил в тупик, то вот вам наглядное руководство:



Блочные и строчные элементы

> *Опять я буду ссылаться на чей-то учебник — на этот раз от Ивана Сагалаева — http://softwaremaniacs.org/blog/category/primer/, и пусть вас не смущают даты написания статей — они повествуют об основах и актуальность они не потеряют ещё очень долго*

Возможно, вы ещё не знаете, но HTML теги делятся на блочные (block) и строчные (inline). Блочными элементами называют те, которые отображаются как прямоугольник, они занимают всю доступную ширину и их высота определяется содержимым. Блочные теги по умолчанию начинаются и заканчиваются новой строкой — это `<div>`, `<h1>` и собратья, `<p>` и другие.
Если хотите, чтобы ваш HTML оставался валидным, следите за тем, чтобы блочные элементы не располагались внутри строчных элементов. Внутри строчных тегов может быть либо текст, либо другие строчные элементы.

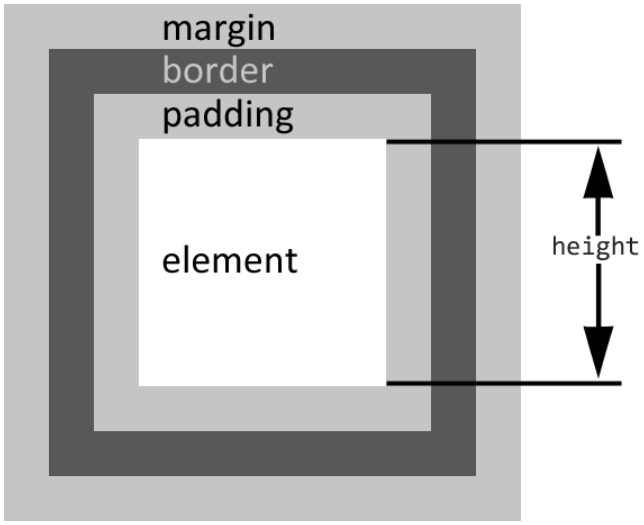
> *Одна из самых часто встречаемых ошибок, это оборачивание заголовка в ссылку: `<a href="#"><h1>Название статьи</h1></a>`, не допускайте подобные промахи.*
>
> *Хотя если мы ориентируемся на HTML5 – то тег `<a>` теперь может быть блочным элементом, и приведённый пример будет валидным.*
>
> *Ага, вот такой я не последовательный.*

По теме:

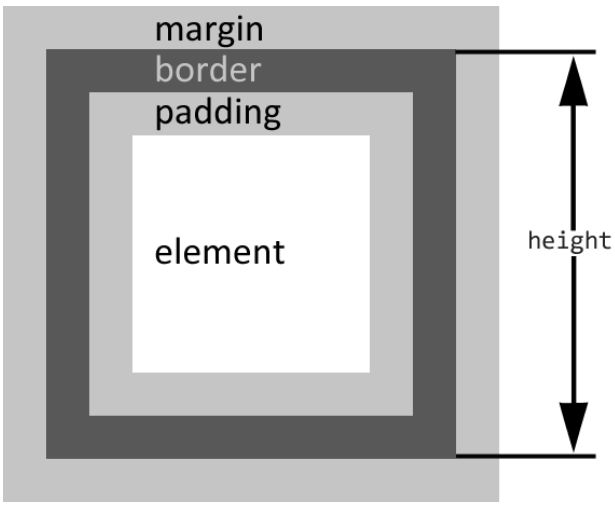
- «Inline Elements List and What's New in HTML5» [http://www.tutorialchip.com/tutorials/inline-elements-list-whats-new-in-html5/]
- «HTML5 Block Level Elements: Complete List» [http://www.tutorialchip.com/tutorials/html5-block-level-elements-complete-list/]
- «Раскладка в CSS: поток» [http://softwaremaniacs.org/blog/2005/08/27/css-layout-flow/]

О размерах блочных элементов

Ещё хотел отдельно остановиться на вычислении ширины и высоты блочных элементов, ведь тут есть один нюанс, по умолчанию, высота и ширина элементов считается без учёта толщины границ и внутренних отступов, т.е. как-то так:



Эта блочная модель называется «content-box», и вот в CSS3 появилась возможность изменять блочную модель, указывая атрибут «box-sizing». Отлично, теперь мы можем выбирать между двумя значениями «content-box» и «border-box», первый я уже описал, а вот второй вычисляет высоту и ширину включая внутренние отступы и толщину границ:



Такая блочная модель была свойственна IE6 в «quirks mode»

Полезные статьи по теме:

— «Блочные элементы» [http://htmlbook.ru/content/blochnye-elementy]
— «Встроенные элементы» [http://htmlbook.ru/content/vstroennye-elementy]

Плавающие элементы

Я бы хотел ещё рассказать о CSS свойстве «`float`», но боюсь, рассказ будет долгим и утомительным, но если кратко: если вы указываете элементу свойство «`float`», то:

— наш элемент будет смещён по горизонтали, и «прилипнет» к указанному краю родительского элемента
— если это был блочный элемент, то теперь он не будет занимать всю ширину родительского элемента, и освободит место
— если следом идут блочные элементы, то они займут его место
— если следом идут строчные элементы, то они будут обтекать наш элемент со свободной стороны

Это поведение «по умолчанию», как это выглядит в живую можно посмотреть на примере css.float.html. Тут главное надо понимать происходящее, и уметь управлять, если конечно вы хотите хоть чуть-чуть научиться верстать :)

Жизненно необходимая информация для верстальщиков:

— «Раскладка в CSS: float» [http://softwaremaniacs.org/blog/2005/12/01/css-layout-float/]

Позиционирование

Дам лишь вводную по «`position`» – у него бывает лишь четыре значения:

`static` — положение дел «по умолчанию», блоки ложатся друг за другом, сверху вниз, по порядку, без отклонений

`absolute` — блок позиционируется согласно заданным координатам

`fixed` — похоже на «`absolute`», с той лишь разницей, что блок не будет скролиться

`relative` — такой блок можно сдвигать относительно места где он расположен, ну и все внутренние «абсолютные» блоки будут использовать данный элемент как точку отсчета при позиционировании по координатам

Для самостоятельного изучения:

— «Раскладка в CSS: позиционирование» [http://softwaremaniacs.org/blog/2005/08/03/css-layout-positioning/]

Разделяй и властвуй

Тут стоит запомнить несколько простых правил:

— выносим JavaScript во внешние файлы
— никогда не пишем inline обработчиков событий «`onclick="some()"`»
— выносим CSS из HTML во внешние файлы
— никогда не пишем inline стилей «`style="color:red"`»
— и ещё разок для закрепления – не пишем inline!

Теперь приведу код, за который следует что-нибудь ломать (это пример плохого кода, уточняю для тех, кто не понимает намёков):

```
<script>
      function doSomething(){ /* … */ }
      /* раздается хруст сломанных костей запястья, чтобы не печатал */
</script>
<style>
      p { line-height:20px; }
      /* крххххх… берцовая кость, и на работу уже не пойдет */
</style>
<div style="color:red;font-size:1.2em">
      <p onclick="doSomething();">Lorem ipsum dolor sit amet...</p>
      <!-- тыдыщь, головой об стол… насмерть, как жест милосердия -->
</div>
```

Неясно, почему же это плохо? Похоже, вам просто не приходилось менять дизайн для уже готового сайта :) Проясню суть проблемы: вам ставят задачу – «*надо поменять цвет шрифта для всех страниц сайта*», коих может быть три десятка. Это могут быть не только HTML-файлы, а страницы какого-то

шаблонизатора, разбросанные по двум десяткам папок (и это еще не самый плохой вариант). И тут появляется он — красный абзац. Вероятность услышать «слова поддержки» в адрес автора сего кода будет стремиться к единице. Насчет inline-обработчиков событий ситуация похожа, вот представьте себе — пишите вы JavaScript код, всё отлично, всё получается, пока вы не пытаетесь кликнуть по красному абзацу, он оказывается вам не подвластен, и живёт своей собственной жизнью, игнорируя все ваши усилия. Вы смотрите код, и опять кто-то услышит эти слова...

Применив четыре правила «красного абзаца» у вас должен будет получиться чистый и предсказуемый HTML код:

```
<div id="abzac">
      <p>Lorem ipsum dolor sit amet...</p>
</div>
```

Стили можно будет легко повесить на `<div>` с идентификатором, как собственно и обработчик событий для нашего параграфа.

> *Абзац — не параграф, но для красного словца, и лёгкости усвоения сгодится*

Немного о JavaScript

В данный раздел я вынес ту информацию о JavaScript, которую необходимо знать, чтобы у вас не возникало «детских» проблем с использованием jQuery. Если у вас есть опыт работы с JavaScript — то листайте далее.

Изучать хотите JavaScript и jQuery? Так силу познайте инструмента истинного:

- *Developer Tools для Chrome, Safari и других webkit-based браузеров*
- *FireBug для FireFox*
- *DragonFly для Opera*
- *Developer Tools для Microsoft Edge*

Список не полон, но console там есть, применять её надо уметь

О форматировании

Хотел бы сразу обратить внимание на форматирование JavaScript кода. Мой опыт мне подсказывает — лучше всего спроецировать стандарты форматирования основного языка разработки на прикладной — JavaScript, а если вы хотите чего-нить из глобального и общепринятого, то я для вас уже погуглил:

- «jQuery Core Style Guidelines» [http://contribute.jquery.org/style-guide/js/]
- «Airbnb JavaScript Style Guide() {» [https://github.com/airbnb/javascript]
- «Как писать неподдерживаемый код?» – вредные советы от Ильи [http://learn.javascript.ru/write-unmain-code]

В довесок поделюсь небольшим советом: все переменные, содержащие объект jQuery, лучше всего именовать, начиная с символа «$». Поверьте, такая небольшая хитрость экономит много времени.

И ещё — в конце каждой строки я ставлю точку с запятой, сам JavaScript этого не требует, но и лишним не будет.

Основы JavaScript

Переменные

Первое с чем столкнёмся – это объявление переменных:

```
var name = "Ivan";
var age = 32;
```

Всё просто, объявляем переменную, используя ключевое слово «`var`».

> *Можно, конечно же, и без «`var`», но делать я вам это настоятельно не рекомендую, т.к. могут возникнуть непредвиденные проблемы в коде о чём чуть позже расскажу.*

> *Да, да, JavaScript относится к языкам с динамической типизацией, и нам нет нужды указывать тип данных при объявлении переменных, вы даже можете устроить «holy war» по этому поводу, но делайте это локально, не выходя за рамки своей черепной коробки.*

На имена переменных наложено два ограничение:

- имя может состоять из букв, цифр, и символов «$» и «_»
- первый символ не должен быть цифрой

Учтите, регистр букв имеет значение:

```
var company = "Facebook";
// совсем другая «компания»
var Company = "Google";
```

Хочу так же вас познакомить с таким нововведением ECMAScript-2015 (в дальнейшем ES-2015) как объявление переменных с использованием конструкции «`let`»:

```
let company = "Facebook";
```

Внешне, от «var» не сильно отличается, зато какая разница в поведении:

- область видимости переменной «let» ограничена блоком {...}, в отличии от «var», которая видна везде внутри функции:

```
var a = 0;
if (true) {
      var a = 1000;
      alert(a);  // 1000
}
alert(a);  // 1000
```

А теперь сравните с поведением «let»:

```
let a = 0;
if (true) {
      let a = 1000;
      alert(a);  // 1000
}
alert(a);  // 0
```

- переменная «let» видна только после объявления:

```
alert(a);  // undefined
var a = 0;

alert(b);  // error: 'b' is not defined
let b = 0;
```

- переменную «let» нельзя объявить повторно:

```
var a;
var a;  // ок

let b;
let b;  // error: 'b' has already been declared
```

— внутри цикла переменная «let» будет объявлена новая для каждой итерации:

```
for (var i = 0; i < 10; i++) { /* … */ }
alert(i); // 10

for (let j = 0; j < 10; j++) { /* … */ }
alert(j); // error: 'j' is not defined
```

Константы

В JavaScript'е до ES-2015 не было констант, но поскольку необходимость в них всё же была и до того, то была неглассная договорённость: переменные, набранные в верхнем регистре через подчёркивание, не изменять:

```
var USER_STATUS_ACTIVE = 1;
var USER_STATUS_BANNED = 2;
```

Константы необходимы, чтобы избежать появления «magic numbers», сравните следующий код «`if(status==2)`» — о чём тут речь мало кому будет понятно, а вот код «`if(status==USER_STATUS_BANNED)`» уже более информативный

Если же говорить о эре ES-2015, то тут уже используем «const», а об остальном уже позаботится сам JavaScript:

```
const USER_STATUS_ACTIVE = 1;
USER_STATUS_ACTIVE = 2;  // error: assignment to constant variable
```

Да-да, я повторюсь, это наглядный пример правильного именования констант — имя однозначно указывает на хранимое значение — «статус активного пользователя»

Отдельно стоит заметить, константа не позволяет изменять саму переменную, но если мы присвоим константе объект или массив, то с ним вы сможете делать что душа пожелает:

```
const USER_STATUS_ACTIVE = 1;
const USER_STATUS_BANNED = 2;
const USER = {
      name: "mr.Smith",
      status: USER_STATUS_BANNED
};

USER.status = USER_STATUS_ACTIVE;  // ok
USER = {name: "mr.Wesson"};        // error
```

Вот и получился «читаемый» код, мотайте на ус

Типы данных

В JavaScript не так уж и много типов данных:

— `number` — целое или дробное число:

```
var answer = 42;
var pi = 3.1415;
```

также существуют следующие специальные значения:

`NaN (not-a-number)` — результат числовой операции, которая завершилась ошибкой, запустите следующий код в консоли:

```
Math.sqrt(-5);
```

но учтите:

```
(NaN == NaN) == false;
  isNaN(NaN) == true;
```

`Infinity` — за гранью `1.7976931348623157E+10308` (т.е. больше)

`-Infinity` — за гранью `-1.7976931348623157E+10308` (т.е. меньше)

— `string` — строка, заключается в кавычки:

```
var str = "Hello World!";
```

в JavaScript нет разницы между двойными кавычками и одинарными (привет PHP)

— `boolean` — булево значение, т.е. или «`true`» или «`false`»

```
var result = true;
```

— `object` — это объекты, на них остановлюсь подробнее чуть позже…
— `symbol` — тип данных из ES-2015, служит для создания уникальных идентификаторов (о нём рассказа не будет, не в ходу ещё)
— `null` — специальное значение для определения «пустоты»

```
var result = null;
```

— `undefined` — ещё одно специальное значение, для «неопределенности», используется как значение неопределённой или несуществующей переменной, например, если переменная объявлена, но значение ей ещё не присвоено:

```
// переменная есть, но нет значения
var a;
alert(a);  // undefined
if (typeof a == "undefined") {
      alert("variable is undefined");
}

// или может совсем не быть переменной
if (window["a"] == undefined) {
      alert("variable does not exist");
}
```

Во втором примере нас может ожидать сюрприз, если кто определит переменную «undefined», как обойти такую «неприятность» я ещё расскажу.

Массивы

Массив — это коллекция данных с числовыми индексами. Данные могут быть любого типа, в качестве примера, я приведу один из самых простых вариантов — массив со строками:

```
                0       1       2
var users = ["Ivan", "Petr", "Serg"]
```

Нумерация массивов начинается с «0», так что для получения первого элемента вам потребуется следующий код:

```
alert(users[0]);      // выведет Ivan
```

Размер массива хранится в свойстве length:

```
alert(users.length);  // выведет 3

a[3] = "Danylo";
alert(users.length);  // выведет 4
```

В действительности «length» возвращает индекс последнего элемента массива+1, так что не попадитесь:

```
var a = [];
a[4] = 10;
alert(a.length); // выведет 5;
```

Для перебора массива лучше всего использовать цикл «for(;;)»:

```
for (let i = 0; i < users.length; i++) {
      alert(users[i]);  // последовательно выведет Ivan, Petr и Serg
}
```

Для работы с последними элементами массива следует использовать методы «push()» и «pop()»:

```
users.push("Sidorov");        // добавляем элемент в конец массива
var sidorov = users.pop();    // удаляем и возвращаем последний элемент
```

Для работы с первыми элементами массива следует использовать методы «unshift()» и «shift()»:

```
users.unshift("Sidorov");     // добавляем элемент в начало массива
var sidorov = users.shift();  // удаляем и возвращаем первый элемент
```

Последние два метода работают медленно, т.к. перестраивают весь массив

Функции

С функциями в JavaScript'е всё просто, вот вам элементарный пример:

```
function hello() {
      alert("Hello world");
}
```

Просто, пока не заговорить об анонимных функциях…

Анонимные функции

В JavaScript можно создавать анонимную функцию (т.е. функцию без имени), для этого достаточно слегка изменить предыдущую конструкцию:

```
function() {
      alert("Hello world");
}
```

Так как функция это вполне себе объект, то её можно присвоить переменной, и (или) передать в качестве параметра в другую функцию:

```
var myAlert = function(name) {
      alert("Hello " + name);
}
function helloMike(myFunc) {   // тут функция передаётся как параметр
      myFunc("Mike");          // а тут мы её вызываем
}
helloMike(myAlert);
```

Анонимную функцию можно создать и тут же вызвать с необходимыми параметрами:

```
(function(name) {
      alert("Hello " + name);
})("Mike");
```

Это не сложно, скоро вы к ним привыкните, и вам их будет недоставать в других языках.

Ах, этот ES-2015, он принёс нам сокращенную запись для анонимных функций — функции-стрелки:

```
// была простая анонимная функция
var inc = function (x) {
      return x+1;
}
// стала запись в одну строчку
var inc = x => x+1;

// была функция с несколькими аргументами
var sum = function (a, b) {
      return a+b;
}
// стала запись в одну строчку
var sum = (a, b) => a+b;
```

Данная запись конечно же удобная, но не стоит увлекаться, если перестараться, то ваш код станет нечитаем, и по итогу вы можете услышать много не лестных выражений в свой адрес.

Объекты

На объекты в JavaScript возложено две роли:

- хранилище данных
- функционал объекта

Первое предназначение можно описать следующим кодом:

```
var user = {
      name: "Ivan",
      age: 32
};
alert(user.name); // Ivan
alert(user.age);  // 32
```

Это фактически реализация key-value хранилища, или хэша, или ассоциативного массива, или …, ну вы поняли, названий много, но в JavaScript'е это объект, и запись выше – это JSON – JavaScript Object Notation (хоть и с небольшими оговорками).

Для перебора такого хранилища можно использовать цикл «`for(.. in ..)`»:

```
for (let prop in user) {
      alert(prop + "=" + user[prop]);  // выведет name=Ivan
                                       // затем age=32
}
```

С объектами, конструкторами и т.д. в JavaScript посложнее будет, хотя для понимания не так уж и много надо, запоминайте: любая функция вызванная с использованием ключевого слова «`new`» возвращает нам объект, а сама становится конструктором данного объекта:

```
function User(name) {
      this.name = name;
      this.status = USER_STATUS_ACTIVE;
}
var me = new User("Anton");
```

Поведение функции «`User()`» при использовании «`new`» слегка изменится:

1. Данная конструкция создаст новый, пустой объект
2. Ключевое слово «this» получит ссылку на этот объект
3. Функция выполнится и возможно изменит объект через «this» (как в примере выше)
4. Функция вернёт «this» (по умолчанию)

Результатом выполнения кода будет следующий объект:

```
{name: "Anton", status: 1 }
```

Опять отправлю читать про ES-2015, в данном стандарте появилась конструкция «class», что по сути — синтаксический сахар для JavaScript'а — специально для тех, кто любит C-подобные языки программирования — https://learn.javascript.ru/es-class.

Область видимости и чудо this

Для тех, кто только начинает своё знакомство с JavaScript я расскажу следующие нюансы:

— когда вы объявляете переменную или функцию, то она становится частью «window»:

```
var a = 1234;
alert(window["a"]); // => 1234
function myLog(message) {
	alert(message); // => 1234
}
window["myLog"](a);
```

— когда искомая переменная не найдена в текущей области видимости, то её поиски будут продолжены в области видимости родительской функции:

```
var a = 1234;
(function(){
	var b = 4321;
	(function() {
		var c = 1111;
		alert((a+b)/c); // => 5
	})();
})();
```

— чудо-переменная «this» всегда указывает на текущий объект вызывающий функцию (поскольку по умолчанию все переменные и функции попадают в «window», то «this == window»):

```
var a = 1234;
function myLog() {
      alert(this);    // => window
      alert(this.a);  // => 1234
}
```

— контекст «this» можно изменить используя функции «bind()», «call()», и «apply()»

Всё что касается «window» относится лишь к браузерам, но поскольку книга о jQuery, то иное поведение я и не рассматриваю, но вот так прозрачно намекаю, что оно есть ;)

Замыкания

Изучив замыкания, можно понять много магии в JavaScript'е. Приведу пример кода с пояснениями:

```
var a = 1234;
var myFunc = function(){
      var b = 4321;
      var c = 1111;
      return function() {
            return ((a+b)/c);
      };
};
var anotherFunc = myFunc();   // myFunc возвращает анонимную функцию
                              // с «замкнутыми» значениями c и b
alert(anotherFunc()); // => 5
```

Что же тут происходит: функция, объявленная внутри другой функции, имеет доступ к переменным родительской функции. Повтыкайте в код, пока вас не осенит, о чём я тут толкую.

Хорошая задачка, которая в полной мере даёт понимание сути проблемы: «Армия функций» [https://learn.javascript.ru/task/make-army]

Рекомендуемые статьи по теме:

- «Привязка контекста и карринг: "bind"» [https://learn.javascript.ru/bind]
- «Явное указание this: "call", "apply"» [https://learn.javascript.ru/call-apply]
- «Функции "изнутри", замыкания» [http://learn.javascript.ru/closures]
- «Использование замыканий» [http://learn.javascript.ru/closures-usage]
- «Closures: Front to Back» [http://net.tutsplus.com/tutorials/javascript-ajax/closures-front-to-back/]

Вводная по JavaScript затянулась, лучше не поленитесь, и изучите весь учебник от Ильи Кантора — http://learn.javascript.ru/.

10% Подключаем, находим, готовим

Базу подготовили, и теперь пора перейти к непосредственному изучению jQuery. Всё начинается с подключения библиотеки. И уже на этом этапе мы можем пойти несколькими путями:

1. Скачиваем jQuery с домашней странице проекта (http://jquery.com/) и положим рядышком с нашей HTML страничкой (советую скачать development версию — всегда интересно покопаться в исходном коде :):

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

   Данный способ хорош для работы в offline, или при медленном соединении с интернетом. Отдельно стоит обратить внимание на путь — скрипты в отдельной папке, и это не случайно, нужно приучать себя к порядку.

2. Используем CDN (предпочитаю сервис от Google, хотя есть и Microsoft и Яндекс, и ещё универсальный http://cdnjs.com/, последний, кстати, размещает много популярных плагинов, за что отдельное спасибо):

```
<head>
      <script type="text/javascript"
      src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.0/jquery.m
      in.js"></script>
</head>
```

   Небольшие пояснения: CDN достаточно умная штука, при таком запросе библиотеки jQuery вам вернутся HTTP заголовки, в которых будет сказано, что «протухнет» этот файл лишь через год. Если же вы запросите файл по адресу «`jquery/3.1/jquery.min.js`», то вам вернётся последняя доступная версия библиотеки из ветки 3.1 — на момент написания сих строк это была версия 3.1.0, при этом в заголовках «`expires`» будет стоять текущая дата, и кэш будет жить лишь час.

Есть CDN предоставляемый и самими разработчиками jQuery, но он отнюдь не такой продвинутый как у Google, и на моей памяти у него были проблемы со стабильностью, так что будьте аккуратней при работе с ним – http://code.jquery.com/

3. С использованием менеджера пакетов bower.io устанавливаем искомую библиотеку. Данный менеджер пакетов позволяет устанавливать очень много разнообразных библиотек и пакетов, зацените список – http://sindresorhus.com/bower-components/

   Зачем я упоминаю про данный менеджер пакетов? Ну может кто из вас окажется ну очень любопытным и осилит работу с ним самостоятельно :)

Будь готов

Теперь пора приступить к работе — возьмём какой-нибудь элемент на страничке и попробуем его изменить. Для этого в <head> вставим следующий код (пример странички ищите ранее):

```
<script>
	// мы пытаемся найти все элементы <h2> на странице
	// и изменить цвет шрифта на красный
	jQuery("h2").css("color", "red");
</script>
```

Только подобный код ничего не сделает, так как, на момент выполнения, на странице ещё не будет тегов <h2>, слишком рано выполняется скрипт, до загрузки всего HTML документа. Для того, чтобы код сработал верно, мы должны либо поместить код в самый низ страницы (главное после искомого <h2>), либо использовать метод «.ready()» для отслеживания события «load» нашего «document»:

```
<script>
	// ждём загрузки всего документа
	// после этого будет выполнена анонимная функция
	// которую мы передали в качестве параметра
	jQuery(document).ready(function(){
		jQuery("h2").css("color", "red");
	});
</script>
```

Также можно использовать сокращённый вариант без явного вызова метода «.ready()»:

```
<script>
	$(function(){
		$("h2").css("color", "red");
	});
</script>
```

Последний вариант стоит причислить к «best practices», ведь в jQuery 3.0 метод «`.ready()`» уже помечен как deprecated.

Вы можете создать сколь угодно много подобных функций, не обязательно весь необходимый функционал помещать в одну.

`$()` — это синоним для «`jQuery()`», чтобы у вас не возникало конфликтов с другими ~~странами~~ библиотеками за использование `$`, советую ваш код оборачивать в анонимную функцию следующего вида (best practice):

```
(function($, undefined){
      // тут тихо и уютно
      // мы всегда будем уверены, что $ === jQuery
      // а undefined не переопределен ;)
})(jQuery);
```

Наглядный код в [ready.html]

Селекторы

Как я уже говорил ранее, в поиске элементов на странице заключается практически половина успешной работы с jQuery. Так что приступим к поискам по документу (чтобы не листать, пусть пример HTML будет и тут):

```
<!DOCTYPE html>
<html dir="ltr" lang="en-US">
<head>
      <meta charset="UTF-8"/>
      <title>Page Title</title>
</head>
<body>
      <header>
            <h1>Page Title</h1>
            <p>Page Description</p>
      </header>
      <div id="content" class="wrapper box">
      <article>
            <h2>Article Title</h2>
            <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
            Nunc urna metus, ultricies eu, congue vel, laoreet...</p>
      </article>
      <article>
            <h2>Article Title</h2>
            <p>Morbi malesuada, ante at feugiat tincidunt, enim massa
            gravida metus, commodo lacinia massa diam vel eros...</p>
      </article>
      </div>
      <footer>&copy;copyright 2014</footer>
</body>
</html>
```

А теперь приступим к выборкам — выбор элементов по «`id`» либо «`className`» аналогично используемым в CSS:

```
$("#content")     // выбираем элемент с id=content
$("div#content")  // выбираем div с id=content (хотя и без div работает)
$(".wrapper")     // выбираем элементы с class=wrapper
$("div.wrapper")  // выбираем div'ы с class=wrapper
$(".wrapper.box") // выбираем элементы с class=wrapper и box
$("h2")           // выбираем все теги h2
$("h1, h2")       // выбираем все теги h1 и h2
```

Используйте валидные имена классов и идентификаторов

Теперь вспомним, что мы в DOMe не одни, это таки иерархическая структура:

```
$("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")  //
```

У нас есть соседи:

```
$("h1 + p")        // выбор всех p элементов, перед которыми есть h1
                   // элементы (у нас только один такой)

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

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

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
```

Если хотите поиграться с селекторами от души — то для этого я припас для вас соответствующую страничку — [css.selectors.html]

Поиск по атрибутам

Ещё со времён CSS2 была возможность найти элемент с определёнными атрибутами, в CSS3 добавили ещё возможностей по поиску:

`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)

Заглянув внутрь jQuery вы скорей всего найдёте то место, где ваше выражение будет анализироваться с помощью регулярных выражений, по этой причине в селекторах необходимо экранировать специальные символы используя обратный слеш «\\»:

```
$("a[href^=\\/]").addClass("internal");
```

Поиск по дочерним элементам

Хотелось бы еще обратить внимание на селекторы из спецификации CSS3 [http://www.w3.org/TR/css3-selectors/] — много интересных:

`:first-child` — первый дочерний элемент
`:last-child` — последний дочерний элемент
`: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

Но поскольку не все браузеры знакомы с CSS3-селекторами, то мы можем использовать jQuery для назначения стилей:

```
$("div:last-child").addClass("last-paragraph");
```

Sizzle

> *Пропустите это раздел, и вернитесь к нему тогда, когда вас заинтересует, как происходит поиск элементов внутри «$»*

В качестве «поисковика» по элементам DOM'a jQuery использует библиотеку Sizzle. Данная библиотека одно время была неотъемлемой частью jQuery, затем «отпочковалась» в отдельный проект, который с радостью использует как сам jQuery, так и Dojo Toolkit. Но хватит лирики, давайте перейдем непосредственно к поиску, для оного в JavaScript'e предусмотрены следующие методы (не в jQuery, не в Sizzle, а в JavaScript'e):

- `getElementById(id)` — поиск по «`id="…"`»
- `getElementsByName(name)` — поиск по «`name="name"`» и «`id="name"`»
- `getElementsByClassName(class)` — поиск по «`class="class"`»
- `getElementsByTagName(tag)` — поиск по тегу
- `querySelectorAll(selector)` — поиск по произвольному CSS селектору

Пробежавшись беглым взглядом по списку, можно заметить метод «`querySelectorAll()`» – он универсален и действительно удобен, да, именно этот метод библиотека пытается вызвать, когда вы скармливаете что-то в качестве селектора в jQuery, но данный метод иногда нас подводит, и тогда на сцену выходит Sizzle во всей красе, вооруженный всеми упомянутыми методами, да еще со своим уникальным арсеналом:

```
if (document.querySelectorAll) (function(){
	var oldSelect = select
	/* ... */
	select = function( selector, context, results, seed, xml ) {
		// используем querySelectorAll когда нет фильтров в запросе,
		// когда это запрос не по xml объекту,
		// и когда не обнаружено проблем с запросом
		// еще есть пару проверок, которые я опустил для наглядности
		try {
			push.apply(
				results,
				slice.call(context.querySelectorAll( selector ), 0)
			);
			return results;
		} catch(qsaError) { /* подвёл, опять, ну сколько можно */ }
		/* ... */
```

```
            // выход Sizzle
            return oldSelect( selector, context, results, seed, xml );
        };
});
```

Но давайте уже рассмотрим, как Sizzle ищет в DOM'e, если таки метод «`document.querySelectorAll()`» споткнулся. Начнём с разбора алгоритма работы на следующем примере:

```
$("thead > .active, tbody > .active")
```

1. Получить первое выражение до запятой: «`thead > .active`»
2. Разбить на кусочки: «`thead`», «`>`», «`.active`»
3. Найти исходное множество элементов по последнему кусочку (все «`.active`»)
4. Фильтровать справа налево (все «`.active`» которые находятся непосредственно в «`thead`»)
5. Искать следующий запрос после запятой (возвращаемся к первому пункту)
6. Оставить только уникальные элементы в результате

Глядя на данный алгоритм вы должны заметить, что **правила читаются справа на лево**!

Копаем глубже. При анализе даже самого маленького выражения есть несколько нюансов на которые стоит обратить внимание – первый – это приоритет поиска, он различен для различных браузеров (в зависимости от их возможностей, или «невозможностей»):

```
order: new RegExp( "ID|TAG" +
        (assertUsableName ? "|NAME" : "") +
        (assertUsableClassName ? "|CLASS" : "")
)
```

Не обращайте внимание на RegExp – это внутренняя кухня Sizzle

Таким образом, рассматривая выражение «div#my», Sizzle найдёт вначале элемент с «id="my"», а потом уже проверит на соответствие с . Хорошо, это вроде как фильтрация, и она тоже соблюдает очерёдность – это второй нюанс:

```
preFilter: {
      "ATTR": function (match) { /* ... */ },
      "CHILD": function (match) { /* ... */ },
      "PSEUDO": function (match) { /* ... */ },
},
filter: {
      "ID": function (id) { /* ... */ },
      "TAG": function (nodeName) { /* ... */ },
      "CLASS": function (className) { /* ... */ },
      "ATTR": function (name, operator, check) { /* ... */ },
      "CHILD": function (type, argument, first, last) { /* ... */ },
      "PSEUDO": function (pseudo, argument, context, xml) { /* ... */ }
}
```

Третий нюанс – это относительные фильтры, которые работают сразу с двумя элементами (они вам знакомы по CSS селекторам):

```
relative: {
      ">": { dir: "parentNode", first: true },
      " ": { dir: "parentNode" },
      "+": { dir: "previousSibling", first: true },
      "~": { dir: "previousSibling" }
},
```

Ой, зачем я вас всем этим гружу? Почитайте лучше об оптимизации запросов абзацем ниже.

Официальная документация по библиотеки Sizzle доступна на GitHub'е проекта:

— «Sizzle Documentation»

[https://github.com/jquery/sizzle/wiki/Sizzle-Documentation]

Оптимизируем выборки

Ну перво-наперво вам следует запомнить, что

> ***результаты поиска не кэшируются, каждый раз, запрашивая элементы по селектору, вы инициируете поиск элементов снова и снова***

Взглянув на алгоритм работы Sizzle, сходу напрашиваются несколько советов об оптимизации по работе с выборками:

1. Сохранять результаты поиска (исходя из постулата выше):

```
// было
$("a.button").addClass("active");
      /* ... */
$("a.button").click(function(){ /* ... */ });

// стало
var $button = $("a.button");
$button.addClass("active");
      /* ... .*/
$button.click(function(){ /* ... */ });
```

Правильная IDE о подобных вещах знает, и будет вам время от времени подсказывать ;)

2. Или использовать цепочки вызовов (что по сути аналогично первому правилу):

```
// было
$("a.button").addClass("active");
$("a.button").click(function(){ /* ... */ });

// стало
$("a.button").addClass("active")
             .click(function(){ /* ... */ });
```

3. Использовать «context» (это такой второй параметр при выборе по селектору):

```
// было
$(".content a.button");

// стало
$("a.button", ".content");
$(".content").find("a.button",); // чуток быстрее
```

4. Разбивать запрос на более простые составные части используя «context», и сохранять промежуточные данные (как следствие из правил №1 и №3)

```
// было
$(".content a.button");
$(".content h3.title");

// стало
var $content = $(".content")
$content.find("a.button");
$content.find("h3.title");
```

5. Использовать более «съедобные» селекторы дабы помочь методу «.querySelectorAll()», т.е. если у вас нет уверенности в правильности написания селектора, или сомневаетесь в том, что все браузеры поддерживают необходимый CSS селектор, то лучше разделить «сложный» селектор на несколько более простых:

```
// было
$(".content div input:disabled");

// стало
$(".content div").find("input:disabled");
```

6. Не использовать jQuery, а работать с «native» функциями JavaScript'a

Есть ещё один пункт – выбирать самый быстрый селектор из возможных, но тут без хорошего багажа знаний не обойтись, так что дерзайте, пробуйте и присылайте ваши примеры.

Для наглядности лучше всего взглянуть на сравнительный тест sizzle.html (данный тест был изначально разработан Ильёй Кантором для мастер-класса по JavaScript и jQuery)

> *Маленькая хитрость от создателей jQuery – запросы по id элемента не доходят до Sizzle, а скармливаются «`document.getElementById()`» в качестве параметра:*

```
$("#content") -> document.getElementById("content");
```

Примеры оптимизаций

Выбор по идентификатору элемента — самый быстрый из возможных, старайтесь использовать оный:

```
// внутри одна регулярочка + getElementById()
$("#content")

// а вот так ещё быстрее
$(document.getElementById("content"))

// но экономия незначительна
// а удобство использования стремится к нулю
```

Селектор «`div#content`» работает на порядок медленнее, нежели поиск лишь по идентификатору – «`#content`», но и он имеет право на существование в случае, если ваш скрипт используется на нескольких страницах, а логика требует лишь обрабатывать поведение для элемента `<div>`. Данный селектор можно представить в двух вариантах:

```
// getElementById() + фильтрация
$("#content").filter("div");

// оставляем как есть и надеемся на QuerySelectorAll()
$("div#content");
```

В результате тестирования получаем следующий расклад:

— пример с использованием «`.filter()`» работает быстрее в браузерах Chrome, Firefox и IE9.0+
— оба способа работают наравне в браузерах IE8.0 и мобильном Safari
— второй пример работает в два раза быстрее в последних версиях Opera

Выводы делаем сами.

20% Атрибуты элементов и CSS

В предыдущих примерах мы уже изменяли CSS-свойства DOM-элементов, используя одноименный метод «`.css()`», но это далеко не всё. Теперь копнём поглубже, чтобы не штурмовать форумы банальными вопросами ;)

Копать начнём с более досконального изучения метода «`.css()`»:

`css(property)` — получение значения CSS свойства
`css(property, value)` — установка значения CSS свойства
`css({key: value, key:value})` — установка нескольких значений
`css(property, function(index, value) { return value })` — тут для установки значения используется функция обратного вызова (в просторечии — callback-функция), `index` это порядковый номер элемента в выборке, `value` — старое значение свойства

Метод «`.css()`» возвращает текущее значение, а не прописанное в CSS файле по указанному селектору

Примеры использования (css.html):

```
$("#my").css('color')          // получаем значение цвета шрифта
$("#my").css('color', 'red')   // устанавливаем значение цвета шрифта
// установка нескольких значений
$("#my").css({
      'color':'red',
      'font-size':'14px',
      'margin-left':'10px'
})
// альтернативный способ
$("#my").css({
      color:'red',
      fontSize:'14px',
      marginLeft:'10px',
})
// используя функцию обратного вызова
$("#my").css('height', function(i, value){
      return parseFloat(value) * 1.2;
})
```

Ну, вроде с CSS разобрались, хотя нет — стоит ещё описать манипуляции с классами, тоже из разряда первичных навыков:

`addClass(className)` — добавление класса элементу

`addClass(function(index, currentClass){ return className })` — добавление класса используя функцию обратного вызова

`hasClass(className)` — проверка на причастность к определённому классу

`removeClass(className)` — удаление класса

`removeClass(function(index, currentClass){ return className })` — удаление класса используя функцию обратного вызова

`toggleClass(className)` — переключение класса

`toggleClass(className, switch)` — переключение класса по флагу switch

`toggleClass(function(index, currentClass, switch){ return className }, switch)` — переключение класса используя функцию обратного вызова

В приведённых методах в качестве «`className`» может быть строка содержащая список классов через пробел.

Мне ни разу не приходилось использовать данные методы с функциями обратного вызова, и лишь единожды пригодился флаг «switch», так что не заморачивайтесь всё это запоминать, да и в дальнейшем, цитируя руководство по jQuery, я буду сознательно опускать некоторые «возможности»

Но хватит заниматься переводом официальной документации, перейдём к наглядным примерам (class.html):

```
// добавляем несколько классов за раз
$("#my").addClass('active notice')

// переключаем несколько классов
$("#my").toggleClass('active notice')
```

```
// работает вот так (похоже на классовый XOR):
<div id="my" class="active notice">  →  <div id="my" class="">
<div id="my" class="active">   →  <div id="my" class="notice">
<div id="my" class="">  →  <div id="my" class="active notice">

// аналогично предыдущему примеру
$("#my").toggleClass('active')
$("#my").toggleClass('notice')

// проверяем наличие класса(-ов)
$("#my").hasClass('active')

// удаляем несколько классов за раз
$("#my").removeClass('active notice')
```

Также, стоит вспомнить, что у DOM элементов бывают атрибуты отличные от класса, и мы их тоже можем изменять, для этого нам потребуются следующие методы:

`attr(attrName)` — получение значения атрибута

`attr(attrName, attrValue)` — установка значения атрибута (также можно использовать hash, либо функцию обратного вызова)

`removeAttr(attrName)` — удаление атрибута

Атрибуты – это всё то, что мы видим внутри угловых скобочек, когда пишем HTML код:

```
<!-- В данном примере это href, title, class -->
<a href="#top" title="anchor" class="simple">To Top</a>
```

Атрибуты, с которыми вам чаще других придётся сталкиваться:

```
// получение альтернативного текста картинки
var altText = $('img').attr('alt')

// изменение адреса картинки
$('img').attr('src', '/images/default.png')
```

```
// работаем со ссылками
$('a#my').attr({
      'href':'http://anton.shevchuk.name',
      'title':'My Personal Blog',
});
```

Кроме атрибутов, также есть свойства элементов, к ним относится «selectedIndex», «tagName», «nodeName», «nodeType», «ownerDocument», «defaultChecked» и «defaultSelected». Ну вроде бы список невелик, можно и запомнить. Для работы со свойствами используем методы из семейства «.prop()»:

`prop(propName)` — получение значения свойства

`prop(propName, propValue)` — установка значения свойства (также можно использовать hash, либо функцию обратного вызова)

`removeProp(propName)` — удаление свойства (скорей всего никогда не понадобится)

А теперь выключите музыку, и запомните следующее – **для отключения элементов формы, и для проверки/изменения состояния чекбоксов мы всегда используем метод «.prop()»,** пусть вас не смущает наличие одноименных атрибутов в HTML (это я про «disabled» и «checked»), используем «.prop()» и точка (наглядный пример property.html)

30% События

Прежде чем приступить к прочтению данной главы, стоит определиться, что же из себя представляют события web-страницы. Так вот – события – это любые действия пользователя, будь то ввод данных с клавиатуры, проматывание страницы или передвижения мышки, и конечно же «клики».

> *А ещё существуют события создаваемые скриптами, и их обработчики – триггеры и хэндлеры, но о них чуть позже.*

jQuery работает практически со всеми событиями в JavaScript'e, приведу список оных с небольшими пояснениями:

`change` — изменение значения элемента (значение, при потери фокуса, элемента отличается от изначального, при получении фокуса)

`click` — клик по элементу (порядок событий: «`mousedown`» → «`mouseup`» → «`click`»)

`dblclick` — двойной щелчок мышки

`resize` — изменение размеров элементов

`scroll` — скроллинг элемента

`select` — выбор текста для «`input[type=text]`» и «`textarea`»

`submit` — отправка формы

`focus` — фокус на элементе, актуально для «`input[type=text]`», но в современных браузерах работает и с другими элементами

`blur` — фокус ушёл с элемента, актуально только для элементов «`input[type=text]`» — срабатывает при клике по другому элементу на странице или по событию клавиатуры (к примеру переключение по tab'y)

`focusin` — фокус на элементе, данное событие срабатывает на предке элемента, для которого произошло событие «`focus`»

`focusout` — фокус ушёл с элемента, данное событие срабатывает на предке элемента, для которого произошло событие «`blur`»

`keydown` — нажатие клавиши на клавиатуре

`keypress` — нажатие клавиши на клавиатуре, последовательность «`keydown` → `keypress` → `keyup`»

`keyup` — отжатие клавиши на клавиатуре

`mousedown` — нажатие клавиши мыши

`mouseup` — отжатие клавиши мыши

`mousemove` — движение курсора

`mouseenter` — наведение курсора на элемент, не срабатывает при переходе фокуса на дочерние элементы

`mouseleave` — вывод курсора из элемента, не срабатывает при переходе фокуса на дочерние элементы

`mouseover` — наведение курсора на элемент

`mouseout` — вывод курсора из элемента

> *Опробовать события можно на примере с [событиями мышки] и [элементами формы]. Для большинства событий существуют «shorthand» методы, так для отслеживания «`click`» можно использовать «`.click()`» :)*

Вызов большинства из перечисленных событий можно эмулировать непосредственно из самого скрипта:

```
<script>
	$("#menu li a").click()
	// или используя метод trigger
	$("#menu li a").trigger("click")
</script>
```

Теперь стоит рассказать немного и об обработчиках событий, для примера возьму код строчкой выше, и слегка его модифицирую:

```
$("#menu li a").click(function(event){
	alert("Hello!")
})
```

Теперь кликнув по ссылке вы увидите приветствие и после закрытия оного браузер перейдет по ссылке указанной в атрибуте «href». Но это не совсем то, что мне хотелось – надо было лишь вывести текст, и никуда не уходить. Ага, для этого стоит отменить действие по умолчанию:

```
$("#menu li a").click(function(event){
	alert("Hello!");
	event.preventDefault();
})
```

Теперь перехода нет, т.к. метод «event.preventDefault()» предотвращает данное действие. Но вот если кто-то повесит ещё один обработчик на само меню?

```
$("#menu").click(function(event){
	alert("Menu!");
})
```

В результате мы получим два сообщения, но почему? Если у вас возникает подобный вопрос, значит вы еще не знакомы с тем, как обрабатываются события. Попробую кратенько дать вводную, когда вы кликаете на элементе в DOM дереве, то происходит «погружение» события – т.е. вначале все родительские элементы могут обработать «клик», и лишь потом он доберётся до элемента по которому был совершён, но и это еще не всё, затем событие начинает проделывать обратный путь – «всплывает», давая тем самым второй шанс родительским элементам обработать событие.

> *Но не так всё гладко, у нас же есть IE, который принципиально не работает с «погружением», поэтому все решили идти по пути наименьшего сопротивления и обрабатывают события лишь на этапе «всплытия».*

> *Рекомендую к прочтению статью «Всплытие и перехват» из уже упомянутого учебника Кантора*

Хорошо, вроде бы понятно, теперь вернёмся к нашему примеру, и пытаемся понять что же у нас происходит – у нас есть обработчик клика для ссылки и непосредственно для самого меню, в котором эта ссылка находится. Теперь кликая по ссылке, срабатывает обработчик события на ссылке, и затем событие всплывает до меню, и срабатывает его обработчик события «`click`». Но это не совсем желаемый результат, и для борьбы с подобным вредительством, необходимо останавливать «всплытие» событий:

```
$("#menu li a").click(function(event){
	alert("Hello!");
	event.preventDefault();
	event.stopPropagation();
})
```

Для ускорения разработки в jQuery есть быстрый способ вызова этих двух методов за раз:

```
$("#menu li a").click(function(event){
	return false; // вот это он :)
})
```

Теперь у вас есть достаточный багаж знаний, чтобы легко манипулировать событиями на странице. Хотя я добавлю еще немного — для того, чтобы сработал лишь ваш обработчик события, можно использовать метод «`event.stopImmediatePropagation()`»:

```
$("#menu li a").click(function(event){
	alert("Hello!");
	event.stopImmediatePropagation();
	return false;
})
$("#menu li a").click(function(event){
	alert("Hello again!");
	return false;
})
```

В данном примере, при клике по ссылке будет выведено лишь одно сообщение. И да, порядок имеет значение.

Учимся рулить

Мы уже успели познакомиться с методом «`.click()`», в действительности этот метод представляет из себя обёртку для вызова «`.on()`» и «`.trigger()`»:

```
if (arguments.length > 0) {
      this.on("click", null, data, fn ) :
} else {
      this.trigger("click");
}
```

Ой, код я чуть-чуть изменил — для читаемости, если же любопытство восторжествует, то ищите в исходном коде по строке «`dblclick`»

Ну так давайте же попробуем без этих обёрток:

```
// вешаем обработчик
$('.class').on('click', function(){
      // что-то делаем
});
// вызываем обработчик
$('.class').trigger('click');
// отключаем обработчик
$('.class').off('click');
```

Можно повесить обработчик событий практически на любой объект:

```
// проще некуда
var obj = {
      test:function() {
            console.log('obj.test');
      }
}

// создаём обработчик произвольного события someEvent
$(obj).on('someEvent', function(){
      console.log('obj.someEvent');
      this.test();
});

// инициируем событие someEvent
```

```
$(obj).trigger('someEvent');

// полюбопытствуем
console.log(obj);
```

Скопируйте приведенный код в консоль и запустите, я думаю вам будет интересно ;)

Пространство имен

Как вы уже узнали, когда мы хотим создать/удалить свой обработчик событий, мы пишем следующий код:

```
// создаем свой обработчик
$('.class').on('click', function(){
      // что-то делаем
});

// удаляем все обработчики
$('.class').off();
```

Но как всегда, есть ситуации когда нам необходимо отключить не все обработчики (как пример, надо отключить обработку какого-то контрола определенным плагином), в этом случае нам на помощь приходят пространства имен, использовать их достаточно легко:

```
// создаём обработчик
$('.class').on('click.namespace', function(){
      // что-то делаем
});

// вызываем обработчик
$('.class').trigger('click.namespace');

// вызываем все обработчики без пространства имён
$('.class').trigger('click!');

// удаляем все обработчики click в данном пространстве имён
$('.class').off('click.namespace');
```

Еще примерчик, вешаем обработчик, который выводит текст в консоль:

```
$('.class').on('click.namespace', function(){
      console.log('bang');
});

// вызываем событие, наш обработчик сработает
$('.class').trigger('click.namespace');
// тоже работает
$('.class').trigger('click');
// событие из другого пространства имён, наш обработчик не будет вызван
$('.class').trigger('click.other');
```

Также, есть поддержка нескольких пространств имён:

```
$('.class').on('click.a.b', function(){
      // для пространства имён a и b
});

// вызываем обработчик из пространства a
$('.class').trigger('click.a');
// отменяем обработчик click для пространства b
$('.class').off('click.b');
```

Можно одним махом удалить все обработчики с определенного пространства имен:

```
// обработчик клика
$('.class').on('click.namespace', function(){});
// обработчик фокус
$('.class').on('blur.namespace', function(){});
// передумали, и все отменили
$('.class').off('.namespace');
```

Официальная документация скудна на этот счёт, и я надеюсь мой пример поможет лучше разобраться в данном вопросе (events.namespace.html).

«Живые» события

> *Я тут немного забегу вперёд, так что если чего стало непонятно, отложите данный раздел «на потом».*

Стоит обратить внимание на еще одну задачку, которая очень часто ставится перед разработчиком – это добавление обработчиков событий для элементов, которые добавляются на страницу динамически. Пожалуй, надо привести пример подобной задачи:

« – У нас есть HTML страница, на которой все внутренние ссылки будут подгружаться AJAX'ом, данное утверждение справедливо и для подгружаемого HTML'а тоже»

Первое условие решается просто:

```
$('a[href^=\\/]').on('click', function() {
      var url = $(this).attr('href');
      $('body').load(url + ' body > *');
      return false;
});
```

> *Для наглядности, условимся, что внутренние ссылки содержат относительные пути от корня сайта.*

Со вторым условием чуть-чуть посложнее ситуация, но тоже вполне решаема:

```
$('body').on('click', 'a[href^=\\/]', function() {
      var url = $(this).attr('href');
      $('body').load(url + ' body > *');
      return false;
});
```

Отличий не так уж и много, проясню происходящее:

- первым делом на элемент <body> будет повешен обработчик события «click»
- данный обработчик будет срабатывать только в том случае, когда событие будет относится к элементу <a>

Работа данной схемы базируется на «всплытии» событий, так что используя метод «event.stopPropagation()» вы сможете предотвратить выполнение «живых» обработчиков

> *Лирическое отступление к истории: жил да был когда-то плагин для jQuery, назывался «.live()», позволял он вешать обработчики на элементы DOM дерева которых ещё нет (подгружаемые AJAX'ом или ещё как), а потом ~~он умер~~ его внесли в само ядро. Метод «.live()» к тому времени работал лишь с «document». Затем появился метод «.delegate()» который научился вешать обработчик на произвольный элемент, а затем и он был поглощён методом «.on()». Так что не пугайтесь сильно, если встретите старый метод «.live()», адаптировать под новые версии jQuery его будет не так уж и сложно (ну я на это надеюсь)*

Оптимизация

Неявным бонусом от использования «живых» событий можно считать возможность оптимизации, о «подходящих» случаях я и расскажу.

Случай первый, банальный – представьте себе таблицу на тысячу строк да десяток столбцов, а теперь попытайтесь подсчитать, сколько памяти скушают обработчики события «`click`» для каждой ячейки? Вот-вот, стоит это переписать в один обработчик:

```
$('table').on('click', 'td', function() { /* ... */ }
```

Случай второй, надуманный – необходимо записывать действия пользователя на странице, т.е. отслеживать клики по бессчётному количеству объектов:

```
$('body').on('click', '*', function() {
      console.info("Click on "+this.tagName);
});
```

Пример работы данного «надуманного» варианта можно посмотреть на странице events.optimization.html

Touch события

Смартфоны с большим сенсорным экраном — это уже норма жизни, и любому web-разработчику рано или поздно потребуется разрабатывать интерфейсы с поддержкой «touch» событий. На этот случай в JavaScript'е предусмотрены следующие события:

`touchstart` — событие схоже с «`mousedown`», происходит при касании пальцем экрана

`touchend` — убираем палец с экрана, ака «`mouseup`»

`touchmove` — водим пальцем по экрану — «`mousemove`»

`touchcancel` — странное событие, отмена «`touch`» до того, как палец был убран

О том как с ними работать, можно подчерпнуть из отличной статьи на английском языке — «Touching and Gesturing on iPhone, Android, and More» [http://www.sitepen.com/blog/?p=3425] (хоть рассказ там и о Dojo Toolkit).

В jQuery Mobile работа с touch событиями идёт «из коробки». Чтобы jQuery UI заставить работать с touch событиями следует использовать библиотеку jQuery UI Touch Punch [http://touchpunch.furf.com/] Пробуйте, но учтите, без touch устройства разработка интерфейсов для подобных устройств – нонсенс (*англ. nonsense*).

40% Анимация

Библиотека jQuery позволяет очень легко анимировать DOM элементы, для этого предусмотрено несколько функций, но обо всём по порядку, начнём с простого «`.hide()`» и «`.show()`», эти два метода соответственно скрывают либо отображают элементы:

```
// скроем все картинки
$('img').hide();

// теперь вернём их на место
$('img').show();
```

Данные вызовы оперируют лишь CSS атрибутом «`display`» и переключают его из текущего состояния в «`none`» и обратно. В качестве первого параметра можно задать скорость анимации, для этого можно использовать одно из зарезервированных слов «`slow`» или «`fast`», либо же указывать скорость в миллисекундах (1000 мс = 1 сек):

```
// медленно спускаемся с горы и… скрываем все картинки
//    slow == 600
//    fast == 200
$('img').hide('slow');

// теперь вернём их на место, чуть быстрее
$('img').show(400);
```

В таком случае, исчезновение элементов будет сопровождаться анимацией атрибутов «`width`», «`height`», «`opacity`» и прочих (см. пример в hide.html). В довесок к этим двум методам есть еще метод «`.toggle()`», он работает как переключатель «`hide → show`» или «`show → hide`».

Теперь идём немножко дальше – 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");
});
```

Приведу иллюстрацию для наглядности процесса анимации:

Рисунок 1 — анимация show()

Анимацию таких атрибутов как «height», «width» и «opacity» видно невооружённым взглядом, в действительности же это далеко не всё, заглянув внутрь jQuery можно увидеть, что так же изменяются внутренние и внешние отступы – «padding» и «margin» – так что не стоит об этом забывать.

Идём дальше – у нас на очереди набор методов из семейства slide – «.slideUp()», «.slideDown()» и «.slideToggle()». Их поведение схоже с предыдущими функциями, но анимация будет затрагивать лишь высоту блоков – смотрим пример slide.html (ну и иллюстрации так же есть):

Рисунок 2 — анимация slideDown()

Прежде чем перейти к десерту упомяну семейство функций fade — они манипулируют лишь «opacity»:

`fadeIn(duration, callback)` – изменяет «opacity» от `0` до предыдущего

`fadeOut(duration, callback)` – изменяет «opacity» от текущего до `0`

`fadeToggle(duration, callback)` – переключатель между «In» и «Out»

`fadeTo(duration, opacity, callback)` – изменяет значение «opacity» до требуемого значения

А теперь самое сладкое – все эффекты анимации в jQuery крутятся вокруг метода «`.animate()`». Данная функция берет один или несколько CSS-свойств элемента и изменяет их от исходного до заданного за N-ое количество итераций (количество итераций зависит от указанного времени, но не реже одной итерации в 13мс, если я правильно накопал это значение). Ну что-же, от слов к делу, попробуем реализовать функции «`.fadeIn()`» и «`.fadeout()`» с помощью «`.animate()`» (см. пример animate.html):

```
// fadeOut()
$('article img').animate({
     'opacity':'hide'
})
// fadeIn()
$('article img').animate({
    'opacity':'show'
})
```

Всё просто, давайте-ка теперь усложним задачу – изменим размер блоков и прозрачность:

```
// значения указанных свойств будут плавно изменяться
// от текущих до заданных
$('article img').animate({
    'opacity':0.5,
    'height':'50px',
    'width':'250px'
})
```

Как видите – тоже несложно, теперь попробуем отталкиваться от текущих значений, а не задавать необходимые:

```
// изменяем, шаг за шагом
$('article img').animate({
    'opacity':'-=0.1',
    'height':'+=10px'
})
```

Поиграли и хватит, пора усложнить вам жизнь – у метода «`.animate()`» может быть более одного параметра, и пора приступить к их разбору. Набор параметров может быть разным, приведу первый, тот, что попроще:

`params` – CSS свойства – с этим мы уже познакомились

`duration` – скорость анимации, уже 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

Из приведённых параметров нам только «`easing`» не встречался ранее – я его берёг на сейчас – этот параметр указывает, какая функция будет использоваться для процесса анимации значений. Это могут быть линейные, квадратичные, кубически и любые другие функции. «Из коробки» мы можем выбрать лишь между «`linear`» и «`swing`»:

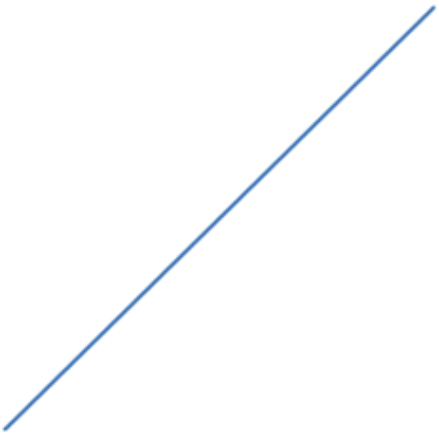


Рисунок 3 — easing «linear»

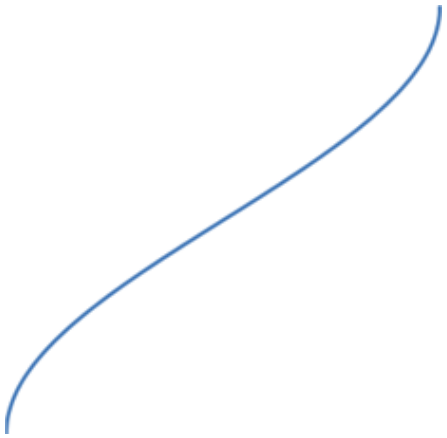


Рисунок 4 — easing «swing»

Заглянув в код jQuery мы легко найдём соответствующий код:

```
linear: function(p) {
      return p;
},
swing: function(p) {
      return 0.5 - Math.cos( p*Math.PI ) / 2;
}
```

p – коэффициент прохождения анимации, изменяется от 0 до 1

Сложно? Хотите больше и сразу? Тогда ищите easing plugin на странице http://gsgd.co.uk/sandbox/jquery/easing/, он действительно из разряда «must have».

Подключайте и используйте одну из трёх десятков функций easing (наглядно, с иллюстрациями – animate.easing.html, а так же http://easings.net/)

Но давайте вернёмся к методу «`.animate()`», которая в качестве параметров может принимать ещё один набор параметров, который уже не будет казаться таким простым:

`params` – CSS свойства (уже было)

`options` – тут целый набор возможностей, часть уже описывалась ранее:

- `duration` – скорость анимации
- `easing` – функция («`linear`» или «`swing`»)
- `complete` – функция, которая будет вызвана после окончания анимации
- `step` – функция, которая будет вызвана на каждом шаге анимации, о ней расскажу чуть ниже
- `queue` – флаг/параметр очереди, чуть позже опишу подробнее
- `specialEasing` – хэш в котором можно описать какую именно easing функцию следует использовать для изменения определённых параметров

Step-by-step

Хотелось бы отдельно остановиться на функции «step», и для наглядности приведу пример реализации подобной функции, которая отображает текущее значение анимированного параметра:

```
var customStep = function(now, obj) {
      obj.elem;    // объект анимации
      obj.prop;    // параметр, который анимируется
      obj.start;   // начальное значение
      obj.end;     // конечное значение
      obj.pos;     // коэффициент, изменяется от 0 до 1
      obj.options; // опции настроек анимации

      now; // текущее значение анимированного параметра, вычисляется как
           // now = (obj.end - obj.start) * obj.pos

      $(this).html(obj.prop +': '+now+obj.unit);  // вывод текста
}

$("#box").animate({height: "+=10px"}, {step: customStep});
```

Мне ни разу не приходилось использовать step-функции, лишь только для примера

В очередь…©

Немного об очередности работы метода «.animate()» – большинство читателей, наверное, уже знакомо с организацией последовательной анимации – для этого мы можем использовать цепочку вызовов:

```
$('#box ')
      // говорим что меняем
      .animate({left:'+=100'})
      // следующий вызов добавляется в очередь на выполнение
      .animate({top:'+=100'})
```

Для параллельного запуска анимации, необходимо будет внести следующие изменения:

```
$('#box')
      // говорим что меняем
      .animate({left:'+=100'})
      // следующий вызов будет игнорировать очередь
      .animate({top:'+=100'}, {queue:false})
```

Есть ещё чудесная функция «.stop()», которая позволяет остановить текущую анимацию на полпути, а так же почистить очередь при необходимости. Для обеспечения различного поведения функции, она принимает три параметра:

queue	— имя очереди; для работы с очередью анимации «fx» данный параметр опускаем («fx» – очередь по умолчанию)
clearQueue	— флаг очистки очереди
jumpToEnd	— применить результат анимации али нет

```
// 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);

// останавливаем выполнение только текущей анимации
// и применяем её результат
$('#box').stop(false, true);
```

Пример есть, и требует ваших проб и ошибок – animate.queue.html

> *Заметка на будущее: если вы сделали выпадающее меню, и поигравшись с мышкой оно продолжает выпадать и исчезать, то значит надо вставить «`.stop()`» в обработчик события*

По умолчанию вся анимация над объектом складывается в очередь «`fx`», но с версии 1.7 можно указывать произвольную очередь:

```
$('#box')
      .animate({'left':'-=100'}, {queue:'x'}) // составляем очередь X
      .dequeue('x')                           // запускаем очередь X

$('#box').stop('x')  // останавливаем анимацию в очереди X
```

Для чего нам может понадобиться произвольная очередь? Да для распараллеливания анимации, чтобы мы могли запустить одну очередь анимации, и в любой другой момент запустить другую очередь, возможно, это заклад под игры? Но чего гадать, давайте смотреть — animate.game.html

Отключение

Иногда требуется отключить всю анимацию (к примеру, для отладки) воспользуйтесь следующей конструкцией:

```
jQuery.fx.off = true;
```

50% Манипуляции с DOM

А теперь я буду долго и нудно рассказывать о том, как с помощью jQuery можно изменять DOM дерево на странице, т.е. добавлять и удалять элементы, но чего это я, глава в действительности не будет объёмной :)

Начнём с создания элементов для последующей работы с ними, документация нам заботливо сообщает, что тут всё просто:

```
var $myDiv = $('<div id="my" class="some"></div>')
```

Этот пример вполне рабочий, да вот только производительностью он блистать не будет, ведь внутри будет всё это разбираться с помощью метода «`jQuery.parseHTML()`», который совсем не быстрый. Но мы можем помочь парсеру если атрибуты элемента будем передавать вторым параметром:

```
var $myDiv = $('<div>', {'id':'my', 'class':'some'})
```

Можем сделать ещё проще:

```
var $myDiv = $('<div>').attr({'id':'my', 'class':'some'});
```

И этот способ будет работать даже быстрее (ну совсем капельку), но почему? Для того, чтобы ответить на данный вопрос – загляните в код jQuery, в самую главную функцию «`init()`», в её коде можно найти алгоритм разбора предыдущего примера:

1. Парсим строку, и создаём DOM элемент в jQuery обёртке
2. Заходим в **цикл** обработки переданных параметров:
    a. Проверяем, а нет ли функции у нашего элемента с таким названием
    b. Если нет, то устанавливаем атрибут элемента используя метод «`.attr()`»

Выводы делайте сами, где мы тут время потеряли :)

Ну и на последок опишу самый быстрый способ, который я часто использую:

```
var myDiv = document.createElement('div');
    myDiv.id = 'my';
    myDiv.className = 'some';
```

Да, это и есть «чистый» JavaScript, но как по мне – в данном случае он не менее удобен любых фреймворков. И вот вам домашнее задание – оптимизируйте такой скрипт:

```
$('<div id="my"><div id="precious">Ring</div></div>')
```

Выполняйте тут, бумага стерпит:

Все необходимые нам методы собраны в одном разделе документации – Manipulation, с некоторыми из них мы уже познакомились, и осталось совсем чуть-чуть:

`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»

— 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 выбранным элементом

`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»

Опять про разницу:

```
$("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

Так, с этим кусочком документации вроде как разобрались, опять же – почувствуйте разницу перечисленных методов, ведь дальше будут ещё:

`replaceWith(content)` – заменяет найденные элементы новым

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

```
$("что-то находим").replaceWith("на что меняем")
$("что вставляем").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;
```

Переварили? Хорошо, теперь настал черёд методов, которые работают с размерами, и знают координаты элементов:

Но прежде чем продолжить, хотелось бы освежить в памяти информацию о вычислении высоты и ширины блочных элементов ;)

`offset()` – вернёт позицию DOM элемента относительно document'a, 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`»

Для наглядности различий между методами «`.height()`», «`.innerHeight()`» и «`.outerHeight()`» я создал страничку height.html, а ещё переделал несколько картинок из официальной документации в одну полноценную иллюстрацию:

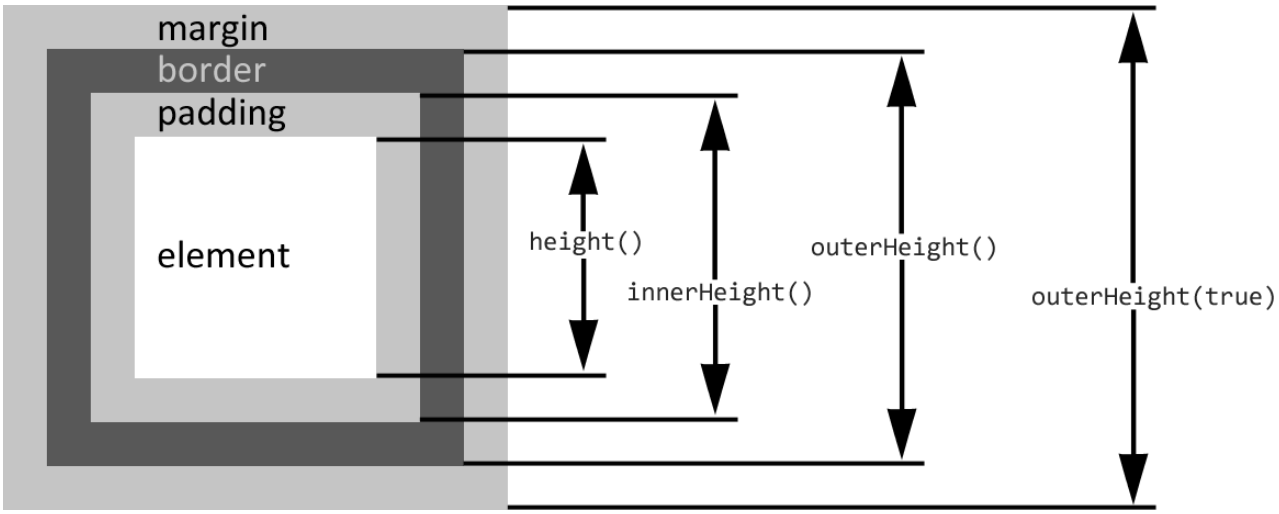


Рисунок 5 — блочная модель

Ну и последняя пара методов:

`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*

Методов реально много, я и сам не всегда помню что и для чего (особенно это касается wrap-семейства), так что не утруждайте себя запоминанием всего перечисленного, главное помнить что таковые имеются и держать под рукой документацию

60% Работа с формами

Формы – это, пожалуй, один из самых нелюбимых элементов на странице – пока настроишь внешний вид, потом еще проверь, что ввели нерадивые пользователи да выведи им информацию о допущенных ошибках, и в конце концов отправляешь на сервер данные с чувством облегчения от проделанной кропотливой работы. Так вот – о том, что поможет в этой самой работе я и буду рассказывать.

Для начала, стоит напомнить события с которыми чаще всего придётся работать:

`change` — изменение значения элемента

`submit` — отправка формы

В каких же случаях они нам помогут? Да всё просто – отслеживание change позволяет обрабатывать такие события как изменение selectbox'a, или radiobutton'a, что потребуется для динамического изменения формы. И самый простой пример тому – это на странице регистрации выбор страны, затем по выбранной стране должен быть подгружен список регионов, по региону – список городов и так далее. Отслеживание submit потребуется для проверки правильности заполнения формы, а так же для отправки формы посредством AJAX. Форму возьмём попроще:

```
<form action="/save/">
      <input type="text" name="name" value="Ivan"/>
      <select name="role">
            <option>User</option>
            <option>Admin</option>
      </select>
      <input type="submit"/>
</form>
```

А примеры будут идти в обратном порядке, вот отправка формы AJAX'ом по ссылке из «`action`»:

```
$('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;
});
```

Вот и первый метод – «`.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"
	},
]
```

Теперь стоит добавить в данный код немного проверки данных:

```
$('form').submit(function(){
	if ($(this).find('input[name=name]').val() == '') {
		alert('Введите имя пользователя');
		return false;
	}
	// кусок кода с отправкой
	// ...
});
```

Вот еще один метод, который нам будет частенько нужен:

> `val()` – получение значения первого элемента формы из выборки
>
> `val(value)` – установка значение всем элементам формы из выборки

Данный метод отлично работает практически со всеми элементами формы, вот только с radiobutton'ами установить значение таким образом не получится, тут потребуется небольшой workaround:

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

> *Можно конечно же использовать и метод «`.click()`», дабы эмулировать выбор необходимо пункта, но это вызовет все обработчики события «`click`», что не желательно*

С checkbox'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')
```

Проверять и отправлять форму AJAX'ом теперь умеем, теперь осталось решить вопрос с динамическим изменением формы, и для этого у нас уже есть все необходимые знания, вот, к примеру, добавление выпадающего списка:

```
$('form').append('<select name="some"></select>');
```

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
$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)
```

Хорошо, работать с формой теперь можем, осталось прикрутить более вменяемый вывод ошибок (да-да, за «`alert()`» да по рукам):

```
if ($(this).find('input[name=user]').val() == '') {
      $(this).find('input[name=user]')
            .before('<div class="error">Введите имя</div>');
      return false;
}
```

При повторной отправки формы не забудьте убрать сообщения оставшиеся от предыдущей проверки:

```
$(this).find('.error').remove()
```

Теперь можно объединить кусочки кода и получить следующий вариант:

```
$('form').submit(function(){
	// чистим ошибки
	$(this).find('.error').remove();
	// проверяем поля формы
	if ($(this).find('input[type=name]').val() == '') {
		$(this).find('input[name=user]')
			.before('<div class="error">Введите имя</div>');
		return false;
	}
	// всё хорошо – отправляем запрос на сервер
	$.post(
		$(this).attr('action'),  // ссылка куда отправляем данные
		$(this).serialize()      // данные формы
	);
	return false;
});
```

Теперь стоит вернуться к списку событий формы, и перечислить недостающие:

`focus` — фокус на элементе, для работы с данным событием так же есть «shorthand» метод «`.focus()`»; потребуется, если надо вывести подсказку к элементу формы при наведении

`blur` — фокус ушёл с элемента + метод «`.blur()`»; пригодится при валидации формы по мере заполнения полей

`select` — выбор текста в «`textarea`» и «`input[type=text]`» + метод «`.select()`»; если соберётесь разрабатывать свой WYSIWYG, то познакомитесь очень плотно

`submit` — отправка формы + метод «`.submit()`»; этот метод будете использовать частенько

Примеры работы данных методов доступны на странице form.html

Вот так мы и расправились с «ужасными» формами, возможно я ещё приведу несколько примеров из реальной жизни, но это будет уже в последующих версиях данного учебника :)

70% AJAX

Что такое AJAX я думаю рассказывать не стоит, ибо с приходом веб-два-нуля большинство пользователей уже воротят носом от перезагрузок страниц целиком, а с появлением jQuery реализация упростилась в разы…

Начнем с самого простого – загрузка HTML кода в необходимый нам DOM элемент на странице. Для этой цели нам подойдет метод «`.load()`». Данный метод может принимать следующие параметры:

`url` – запрашиваемой страницы

`data` – передаваемые данные (необязательный параметр)

`callback` – функция которая будет вызвана при завершении запроса к серверу (необязательный параметр)

Теперь на примерах:

```
// в элемент с id=content будет вставлен весь HTML с указанной страницы
$("#content").load("/get-my-page.html");

// в элемент с id=content будет вставлен HTML с указанной страницы
// выбранный по указанному селектору #wrapper
$("#content").load("/get-my-page.html #wrapper");

// передаем данные на сервер
$("#content").load("/get-my-page.html", {id:42});

// обрабатываем полученные данные
$("#content").load("/get-my-page.html", function(){
      alert("Ничего оригинальней не придумал");
});
```

Из моего опыта работы – вам очень часто придётся пользоваться методом «`.load()`» как описано в первом примере, а еще советую запомнить второй пример, он может выручить, когда надо реализовать загрузку AJAX'ом, а доступа к сервер-сайду у вас нет или он ограничен.

Живой пример можно пощупать на страничке – ajax.load.html

Следующий метод с которым я вас познакомлю будет «`$.ajax()`» – собственно, он тут главный, и все остальные AJAX методы являются лишь обёрткой (и «`.load()`» в том же числе). Метод «`$.ajax()`» принимает в качестве параметра пачку настроек и URL куда стучаться, приведу пример аналогичный вызову «`.load()`»:

```
$.ajax({
	url: "/get-my-page.html", // указываем URL
	method: "GET",            // HTTP метод, по умолчанию GET
	data: {"id": 42},         // данные, которые отправляем на сервер
	dataType: "html",         // тип данных загружаемых с сервера
	success: function (data) {
		// вешаем свой обработчик события success
		$("#content").html(data)
	}
});
```

Тут мы обрабатывали HTML ответ от сервера – это хорошо когда нам полстраницы обновить надо, но данные лучше передавать в «правильном» формате – это XML – понятно, структурировано, и избыточно, и как-то не совсем JavaScript-way, и поэтому наш выбор – это JSON (wikipedia в помощь):

```
{
	"note": {
		"time":"2012.09.21 13:11:15",
		"text":"Рассказать про JSONP"
	}
}
```

Фактически это и есть JavaScript код как есть (JavaScript Object Notation если быть придирчиво точным), при этом формат уже распространён настолько, что работа с данными в другом формате уже не комильфо.

> *Жизнь не стоит на месте, есть и более удобные форматы, но не в JavaScript'е :)*

Для загрузки JSON существует быстрая функция-синоним – «`jQuery.getJSON()`» – в качестве обязательного параметра лишь ссылка, куда стучимся, опционально можно указать данные, для передачи на сервер и функцию обратного вызова

Нельзя просто так взять и описать все возможные параметры для вызова «`$.ajax()`», таки стоит держать официальный мануал под рукой – http://api.jquery.com/jQuery.ajax/

Ещё есть пару-тройку функций, которые стоит упомянуть:

`get(url, data, success, dataType)` — загружает данные методом GET
`post(url, data, success, dataType)` — загружает данные методом POST
`getScript(url, success)` — загружает JavaScript с сервера методом GET

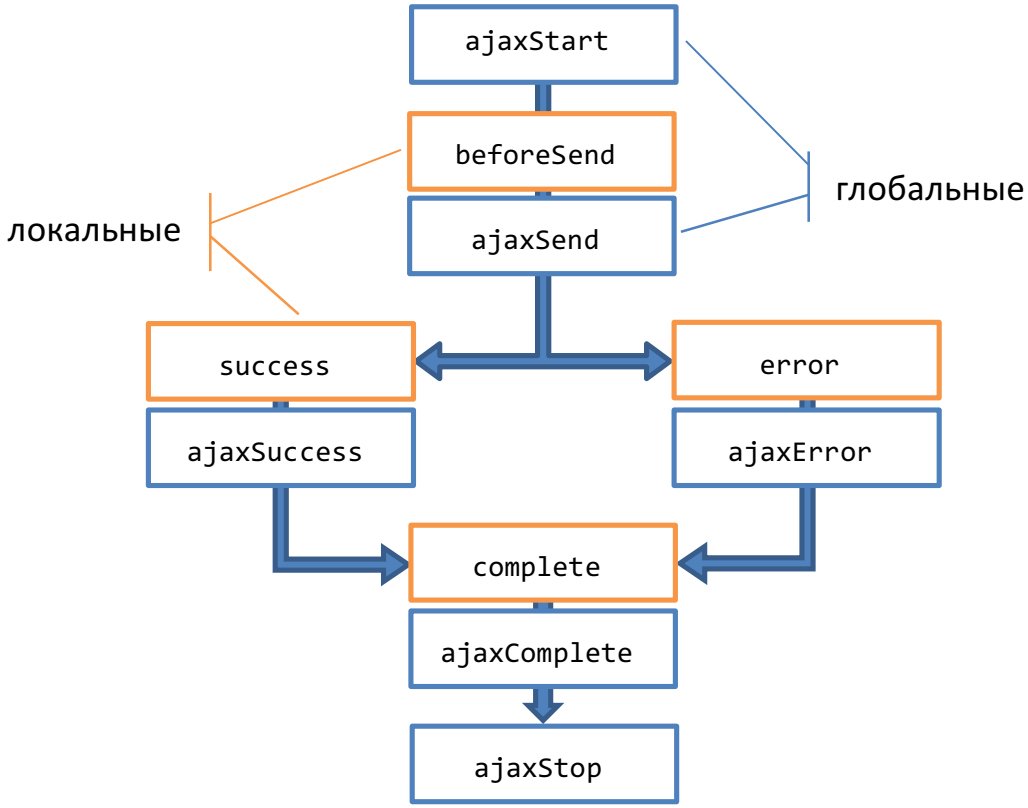
Все они, как я уже говорил ранее, лишь обёртки над вызовом «`$.ajax()`», в чём не сложно убедиться заглянув в исходный код библиотеки, и там, на строчках с 9 137 по 9 156, вы найдёте реализацию методов «`$.get()`» и «`$.post()`» :)

Данная информация актуальна для jQuery версии 3.1.0

Я неспроста оставляю так много места под ваши заметки:..

Обработчики AJAX событий

Для удобства разработки, AJAX запросы бросают несколько событий, и их естественно можно и нужно обрабатывать. jQuery позволяет обрабатывать эти события для каждого AJAX запроса в отдельности, либо глобально. Приведу схемку на которой наглядно видно порядок возникновения событий в jQuery:



Вот и полный список событий с небольшими ремарками:

`ajaxStart` — данное событие возникает в случае когда побежал первый AJAX запрос, и при этом других активных AJAX запросов в данный момент нет

`beforeSend` — возникает до отправки запроса, и позволяет редактировать `XMLHttpRequest`, локальное событие

`ajaxSend` — возникает до отправки запроса, как и «`beforeSend`»

`success` — возникает по возвращению ответа, когда в ответе нет ошибок, локальное событие

`ajaxSuccess` — возникает по возвращению ответа, аналогично событию «`success`»

`error` — возникает в случае ошибки, локальное событие

`ajaxError` — возникает в случае ошибки

`complete` — возникает по завершению текущего AJAX запроса при любом раскладе, локальное событие

`ajaxComplete` — глобальное событие, аналогичное complete

`ajaxStop` — данное событие возникает в случае, когда больше нету активных запросов

Пример для отображения элемента с «`id="loading"`» во время выполнения любого AJAX запроса (т.е. мы обрабатываем глобальное событие):

```
$("#loading").on("ajaxSend", function(){
	$(this).show(); // показываем элемент
}).on("ajaxComplete", function(){
	$(this).hide(); // скрываем элемент
});
```

Это задачка по юзабилити – мы всегда должны держать пользователя сайта в курсе дела о происходящем на странице, и отправка AJAX запроса тоже попадает под разряд «must know». Подобное решение у вас будет практически на любом сайте, где ходит AJAX

Для локальных событий – вносим изменения в опции метода «`.ajax()`»:

```
$.ajax({
	beforeSend: function(){
		// данный обработчик будет вызван
		// перед отправкой данного AJAX запроса
	},
	success: function(){
		// а этот при удачном завершении
	},
	error: function(){
		// этот при возникновении ошибки
	},
	complete: function(){
		// и по завершению запроса (удачном или нет)
	}
});
```

Можно глобальные обработчики отключить принудительно используя флаг «global», для этого выставляем его в «false», и пишем функционал в обход событий «ajaxStart» и «ajaxStop»:

```
$.ajax({
	global: false,
	beforeSend: function(){
		// ...
	},
	success: function(){
		// ...
	},
	error: function(){
		// ...
	},
	complete: function(){
		// ...
	}
});
```

Данная опция частенько помогает избежать конфликтов при работе с AJAX запросами, но не стоит ей злоупотреблять.

JSONP

JSONP – это наш старый знакомый JSON с прослойкой в виде callback функции O_o. Да ладно, давайте на примерах, вот как у нас выглядит ответ сервера в формате JSON:

```
{
	"note": {
		"time":"2012.09.21 13:12:42",
		"text":"Рассказать зачем нужен JSONP"
	}
}
```

Хорошо, когда у нас эти данные приходят с нашего сервера – обработаем, и всё будет чики-пики, но а если нам потребуется заполучить данные с другого сервера, то политика безопасности в браузерах не позволит отправить XMLHTTPRequest на другой сервер, и надо уже будет что-то придумывать. Можно чуть-чуть напрячься и вспомнить, что подключать JavaScript с другого сервера то мы можем, и он будет выполнен. Вот она – зацепка-то, а если подключаемый скрипт будет содержать вызов нашей функции с подготовленными данными – то это уже что-то:

```
alertMe({
	"note": {
		"time":"2012.09.21 13:13:13",
		"text":"Каков же профит от использования JSONP?"
	}
})
```

Таким образом, описав в своём коде функцию «`alertMe()`» мы сможем обработать данные с удаленного сервера. Зачастую, сервера ловят параметр «`callback`» или «`jsonp`», и используют его как имя функции обёртки:

```
<script type="text/javascript"
	src="http://domain.com/getUsers/?callback=alertMe">
</script>
```

Ну это было предыстория, теперь вернёмся к jQuery и методу «`.ajax()`»:

```
$.ajax({
	url: "http://domain.com/getUsers/?callback=?", // указываем URL
	dataType: "jsonp",
	success: function (data) {
		// обрабатываем данные
	}
});
```

В запрашиваемом URL наблюдательный читатель заметит незаконченную структуру «`callback=?`», так вот вместо «`?`» будет подставлено имя ново сгенерированной функции, внутри которой будет осуществляться вызов функции «`success()`». Вместо этой прокси-функции можно использовать и свою функцию, достаточно указать её имя в качестве параметра «`jsonpCallback`» при вызове «`$.ajax()`». Оба этих подхода есть в примере ajax.jsonp.html.

> *А еще стоит упомянуть, что можно указать как обзывается callback-параметр используя параметр «`jsonp`», таким образом указав «`jsonp:"my"`» в URL будет добавлена структура «`my=?`»*

На данный момент достаточно много сервисов предоставляют API с поддержкой JSONP:

- Yahoo – поиск и большинство сервисов
- Flickr – работа с поиском данного сервиса
- MediaWiki – соответственно и производные – Wikipedia, Wiktionary
- CNET – поиск по новостному порталу

Использование подобного подхода позволяет обходить ограничения накладываемые сервисами на количество запросов с одного IP, плюс не грузит сервер дополнительной работой по проксированию запросов от пользователя к сервисам.

К сожалению, многие провайдеры сервисов (такие как Google) отказывают от предоставления доступа к их API с использованием JSONP.

Лечим JavaScript зависимость

Любовь к AJAX бывает чрезмерной, и в погоне к Web2.0 (3.0, 4.0, … — желаемое подчерк-нуть) мы создаём сайты в которых все наши действия бегут через XMLHTTPRequest. Нет, это конечно не плохо — снижаем нагрузку на сервер, канал и т.д. и т.п., но есть одно «но» — у нас есть поисковые машины, которые не озадачивают себя выполнением JavaScript кода, а контент, спрятанный за AJAX запросом, им отдать всё таки нужно. Следовательно, у нас возникает необходимость дублирования навигации (это как минимум) для клиентов без JavaScript.

> *Стоит помнить, что есть ещё пользователи, у которых отключён JavaScript в браузере (или даже не поддерживается, привет тебе, рысь), но эти знают, что делают. А есть ещё скрипты, которые ломаются, и не дают обычным пользователям воспользоваться навигацией по сайту, а пользователей это очень сильно расстраивает, так что эта глава не «просто так».*

Как же всё это обойти и на грабли не наступить? Да всё очень просто — создавайте обычную навигацию, которую вы бы делали не слышав ни разу о AJAX и компании:

```
<ul class="navigation">
      <li><a href="/">Home</a></li>
      <li><a href="/about.html">About Us</a></li>
      <li><a href="/contact.html">Contact Us</a></li>
</ul>
<section id="content"><!-- Content --></section>
```

Данный пример работает у нас без JavaScript'а, все страницы в нашем меню используют один и тот же шаблон для вывода информации, и по факту у нас изменяется лишь содержимое `<div>` с «`id="content"`». Теперь приступим к загрузке контента посредством AJAX – для этого добавим следующий код:

```
$(function() {
	// вешаем обработчик на все ссылки в нашем меню navigation
	$("ul.navigation a").click(function(){
	var url = $(this).attr("href"); // возьмем ссылку
	url =+ "?ajax=true";	// добавим к ней параметр ajax=true

	$("#content").load(url); // загружаем обновлённое содержимое
	return false; // возвращаем false
			//  - дабы не сработал переход по ссылке
	});
});
```

В данном примере мы предполагаем, что сервер, видя параметр «`ajax=true`» вернет нам не полностью всю страницу, а лишь обновление для искомого элемента `<div id="content">`.

> *Конечно, сервер должен быть умнее и не требовать явного указания для использования AJAX'а, а должен вполне удовлетвориться, словив header «X_REQUESTED_WITH» со значением «XMLHttpRequest». Большинство современных фреймворков для web-разработки с этим справляются «из коробки».*

Если же управлять поведением сервера проблематично, и он упёрто отправляет нам всю страницу целиком, то можно написать следующий код:

```
$(function() {
	// вешаем обработчик на все ссылки в нашем меню navigation
	$("ul.navigation a").click(function(){
		var url = $(this).attr("href"); // возьмем ссылку

		// загружаем страницу целиком, но в наш контейнер вставляем
		// лишь содержимое #content загружаемой страницы
		$("#content").load(url + " #content > *");
		return false;	// возвращаем false
	});
});
```

Если в подгружаемом содержимом так же есть ссылки – то вы уже должны знать как «оживить» события.

Прокачиваем AJAX

У нас есть три способа для «прокачки» AJAX'a в jQuery: это создание префильтров, добавление новых конверторов и транспортов.

Префильтры

Префильтр – это функция, которая будет вызвана до «ajaxStart», в ней вы сможете изменить как объект «jqXHR», так и любые сопутствующие настройки:

```
// регистрация AJAX префильтра
$.ajaxPrefilter(function( options, originalOptions, jqXHR ) {
	// наши манипуляции над настройками и jqXHR
});
```

Для чего всё это? Да вот простая задачка – не ждать «старый» AJAX ответ, если мы запрашиваем URL заново:

```
// коллекция текущих запросов
var currentRequests = {};
$.ajaxPrefilter(function( options, originalOptions, jqXHR ) {
	// наша произвольная настройка
	if ( options.abortOnRetry ) {
		if ( currentRequests[ options.url ] ) {
			// отменяем старый запрос
			currentRequests[ options.url ].abort();
		}
		currentRequests[ options.url ] = jqXHR;
	}
});

// вызов с использованием фильтра
$.ajax({
	/* ... */
	abortOnRetry: true
})
```

Ещё можно изменить опции вызова, вот пример который по флагу «crossDomain» пересылает запрос на заранее подготовленную проксирующую страницу на нашем сервере:

```
$.ajaxPrefilter(function( options ) {
	if ( options.crossDomain ) {
		options.url = "/proxy/" + encodeURIComponent( options.url );
		options.crossDomain = false;
	}
});
```

Префильтры можно «вешать» на определенный тип «dataType» (т.е. в зависимости от ожидаемого типа данных от сервера будут срабатывать различные фильтры):

```
$.ajaxPrefilter("json script", function(options, original, jqXHR) {
	/* ... */
});
```

Ну и последнее, для переключение «dataType» на какой-нить другой нам достаточно будет вернуть необходимое значение:

```
$.ajaxPrefilter(function( options ) {
	// это наша функция-детектор необходимых URL
	if ( isActuallyScript( options.url ) ) {
		// теперь «ждём» script
		return "script";
	}
});
```

Будьте очень осторожны когда оперируете глобальными настройками, да ещё через такую неявную фичу как фильтры – задокументируйте подобные подходы в сопроводительной документации, иначе разработчики которые будут в дальнейшем сопровождать ваш код будут сильно ругаться (в качестве оных можете оказаться и вы сами, ну через пару месяцев)

Конверторы

Конвертор – функция обратного вызова, которая вызывается в том случае, когда полученный типа данных не совпадает с ожидаемым (т.е. «dataType» указан неверно).

Всё конверторы хранятся в глобальных настройках «ajaxSettings»:

```
// формат ключа "из_формата в_формат"
// в качестве входного формата можно использовать "*"
converters: {
      "* text": window.String,  // что угодно приводим к тексту
      "text html": true,  // текст к html (флаг true == без изменений)
      "text json": jQuery.parseJSON,  // текст к JSON
      "text xml": jQuery.parseXML  // разбираем текст как xml
}
```

Для расширения набора конверторов потребуется функция «$.ajaxSetup()»:

```
$.ajaxSetup({
      converters: {
            "text mydatatype": function( textValue ) {
                  if ( valid( textValue ) ) {
                        // разбор пришедших данных
                        return mydatatypeValue;
                  } else {
                        // возникла ошибка
                        throw exceptionObject;
                  }
            }
      }
});
```

Имена «dataType» должны всегда быть в нижнем регистре

Конверторы следует использовать, если требуется внедрить произвольные форматы «dataType», или для конвертации данных в нужный формат. Необходимый «dataType» указываем при вызове метода «$.ajax()»:

```
$.ajax( url, { dataType: "mydatatype" });
```

Конверторы можно задавать так же непосредственно при вызове «$.ajax()», дабы не засорять общие настройки:

```
$.ajax( url, {
	dataType: "xml text mydatatype",
	converters: {
		"xml text": function( xmlValue ) {
			// получаем необходимые данные из XML
			return textValue;
		}
	}
});
```

Чуть-чуть пояснений – мы запрашиваем «XML», который конвертируем в текст, который будет передан в наш конвертор из «text» в «mydatatype»

Транспорт

Использование своего транспорта – это крайняя мера, прибегайте к ней, только в том случае если с поставленной задачей нельзя справиться с использованием префильтров и конверторов

Транспорт – это объект, который предоставляет два метода – «send()» и «abort()» – они будут использоваться внутри метода «$.ajax()». Для регистрации своего метода транспортировки следует использовать метод «$.ajaxTransport()», будет это выглядеть как-то так:

```
$.ajaxTransport( function( options, originalOptions, jqXHR ) {
	if ( /* transportCanHandleRequest */ ) {
		return {
			send: function( headers, completeCallback ) {
				/* отправляем запрос */
			},
			abort: function() {
				/* отменяем запрос */
			}
		};
	}
});
```

Проясню чуток параметры с которыми будем работать:

`options`	– настройки запроса (то что указываем при вызове «`$.ajax()`»)
`originalOptions`	– «чистые» настройки, даже без учёта изменений «по умолчанию»
`jqXHR`	– объект «`jQuery XMLHttpRequest`»
`headers`	– заголовки запроса в виде связки ключ-значение
`completeCallback`	– функция обратного вызова, её следует использовать для оповещения о завершении запроса

Функция «`completeCallback()`» имеет следующую сигнатуру:

```
function ( status, statusText, responses, headers ) {
    /* какой-то код */
}
```

где:

`status`	– HTTP статус ответа.
`statusText`	– текстовая интерпретация ответа
`responses`	– это объект содержащий ответы сервера во всех форматах, которые поддерживает транспорт, для примера: родной «`XMLHttpRequest`» будет выглядеть как «`{ xml: XMLData, text: textData }`» при запросе XML документа; опционален
`headers`	– строка содержащая заголовки ответа сервера, ну если конечно транспорт может их получить (вот например метод «`XMLHttpRequest.getAllResponseHeaders()`» это осилит); опционален

Как и префильтры, транспорт можно привязывать к определенному типу запрашиваемых данных:

```
$.ajaxTransport( "script", function( options, originalOptions, jqXHR ) {
    /* привязываемся лишь к script*/
});
```

А теперь мега-напряг – пример транспорта «image»:

```
$.ajaxTransport( "image", function( options ) {
	if (options.type === "GET" && options.async ) {
		var image;
		return {
			send: function( _ , callback ) {
				image = new Image();
				function done( status ) { // подготовим
					if ( image ) {
					var statusText =
						( status == 200 ) ? "success" : "error",
					tmp = image;
					image = image.onreadystatechange = image.onerror
						= image.onload = null;
					callback( status, statusText, { image: tmp } );
					}
				}
				image.onreadystatechange = image.onload = function() {
					done( 200 );
				};
				image.onerror = function() {
					done( 404 );
				};
				image.src = options.url;
			},
			abort: function() {
				if ( image ) {
				image = image.onreadystatechange = image.onerror
					= image.onload = null;
				}
			}
		}; // /return
	} // /if
}); // /ajaxTransport
```

Рабочий пример вы сможете найти в файле ajax.transport.html, но я хотел бы ещё раз напомнить, что это «advanced level», и данный раздел лишний в учебнике «для начинающих».

По следам официальной документации:

– «Extending Ajax: Prefilters, Converters, and Transports» [http://api.jquery.com/extending-ajax/]

80% Объект Deferred и побратимы

> *С jQuery версии 3.x, Deferred объект стал совместим с Promise из ES-2015 (т.н. ES6), так что практически всё, что относится к Promise верно и для Deferred.*

Работа с объектом «Deferred» это уже высший пилотаж, это «mad skills» заставлять асинхронный JavaScript работать так, как нам хочется. Давайте посмотрим как он работает (данный код можно скопировать в консоль и выполнить на любой странице, где подключен jQuery 3.x):

```
// инициализация Deferred объекта
// статус «ожидает выполнение»
var D = $.Deferred();

// подключаем обработчики
D.then(function() { console.log("first") });
D.then(function() { console.log("second") });

// изменяем статус на «fulfilled» - «выполнен успешно»
// для этого вызываем resolve()
// наши обработчики будут вызваны в порядке очереди,
// но они не ждут друг-друга
D.resolve();

// данный обработчик подключён слишком поздно, и будет вызван сразу
D.then(function() { console.log("third") });
```

Если всё это перевести на человеческий язык, то получится следующий сценарий:

- если всё будет хорошо, тогда выполни вот эту функцию и выведи «first»
- и ещё вот эту функцию — «second»
- `resolve()` — мы узнали, всё хорошо
- ~~если~~ всё хорошо, выполняем функцию и выводим «third»

Кроме сценариев с «happy end», есть ещё и грустные истории, когда всё пошло не так как нам бы хотелось:

```
// инициализация Deferred объекта
var D = $.Deferred();

// подключаем обработчики
D.then(function() { console.log("done") });
D.catch(function() { console.log("fail") });

// изменяем статус на «rejected» - «выполнен с ошибкой»
D.reject();

// в консоли нас будет ожидать лишь «fail» :(
```

Получается так:

- если всё будет хорошо, тогда выполни вот эту функцию — «done»
- если всё будет плохо, тогда вот эта функция выведет «fail»
- ой, всё плохо

В действительности метод «`.then()`», позволяет вешать одновременно как обработчики для положительного сценария, так и для варианта с ошибкой:

```
D.then(function(result) { console.log("done") },
       function(error) { console.log("fail") });
```

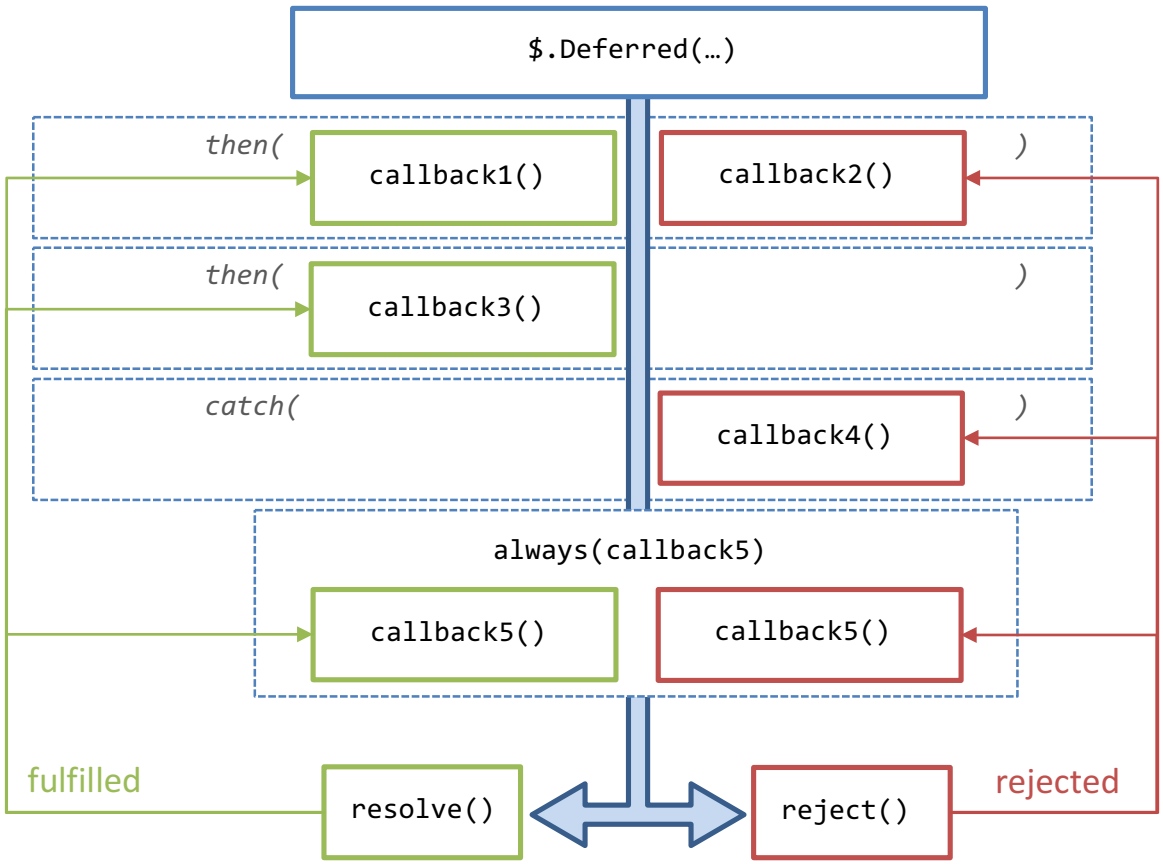
Становится ли от этого код читаемым – сомневаюсь, но такой вариант существует и используется повсеместно, я же предпочитаю для отлова ошибок использовать метод «`.catch()`», который по своей сути лишь сокращенная запись для «`.then(null, fn)`»:

```
var D = $.Deferred();

// подключаем обработчик ошибок через then()
D.then(null, function() { console.log("fail") });
// подключаем обработчик ошибок через catch()
D.catch(function() { console.log("again fail") });
```

Ещё упомяну метод «.always()» – он добавляет обработчики, которые будут выполнены вне зависимости от случившегося (в дейсвтительности, внутри происходит вызов «.done(arguments)» и «.fail(arguments)»).

Чтобы не путаться в перечисленных методах приведу блок-схему:



При вызове «.resolve()» и «.reject()» можно передать произвольные данные в зарегистрированные callback-функции для дальнейшей работы. Кроме того, существуют еще методы «.resolveWith()» и «.rejectWith()», они позволяют изменять контекст вызываемых callback-функции (т.е. внутри них «this» будет смотреть на указанный контекст).

Отдельно хотел отметить, что если вы собираетесь передать Deferred объект «на сторону», чтобы «там» могли повесить свои обработчики событий, но не хотите потерять контроль, то возвращайте не сам объект, а результат выполнения метода «.promise()» – это фактически будет искомый объект в режиме «read only».

А ещё, кроме поведения «ждём чуда», с помощью Deferred можно выстраивать цепочки вызовов – «живые очереди»:

```
$.ajax('ajax/example.json')
	.then(function(){
		// подождём окончания AJAX запроса
		return $('article img').slideUp(2000).promise()
	})
	.then(function(){
		// подождём пока спрячутся картинки
		return $('article p').slideUp(2000).promise()
	})
	.then(function(){
		// подождём пока спрячутся параграфы
		return $('article').hide(2000).promise()
	})
	.then(function(){
		// всё сделано шеф
	});
```

Подобное поведение можно воспроизвести используя лишь animate, но нам же хочется заглянуть чуть-чуть поглубже — [deferred.pipe.html]() (до версии 1.8 тут шла речь о методе «`.pipe()`», а теперь о «`.then()`»)

В данном примере мы вызываем метод «`.then()`», которому скормлена callback-функция, которая должна возвращать объект Promise, это необходимо для соблюдения порядка в очереди – попробуйте убрать в примере один «`return`», и вы заметите, что следующая анимация наступит не дождавшись завершения предыдущей.

На этом возможности Deferred ещё не завершились, есть ещё связка методов «`.notify()`» и «`.progress()`» – первый шлёт послания в callback-функции, которые зарегистрированы с помощью второго. Приведу наглядный код для демонстрации (копи-паст в консоль, и смотрите что получается):

```
var D = $.Deferred();
var money = 100; // наш бюджет
// съём денежки
D.progress(function($){
	console.log(money + " - " + $ + " = " + (money-$));
```

```
        money -= $;
        if (money < 0) { // деньги закончились
                D.reject();
        }
});

// тратим деньги
setTimeout(function(){ D.notify(40); },  500); // покупка 1
setTimeout(function(){ D.notify(50); }, 1000); // покупка 2
setTimeout(function(){ D.notify(30); }, 1500); // покупка 3

D.then(function(){ console.info("All Ok") });
D.catch(function(){ console.error("Insufficient Funds") });
```

Испытайте всю мощь Deferred в примере [deferred.html]

Теперь покажу хитрый метод «`$.when()`»:

```
$.when(
        $.ajax("/ajax/example.json"),
        $("article").slideUp(200)
).then(function(){
        alert("All done");
}, function(){
        alert("Something wrong");
})
```

Поясню происходящее – AJAX запрос и анимация стартуют одновременно, когда и тот и другой завершат свою работу, будет вызвана функция, которую мы передаём в качестве аргумента в метод «`.then()`» (одна из двух, в зависимости от исхода происходящего). Для обеспечения работы этой «магии» методы «`$.when()`», «`$.ajax()`» и «`.animate()`» реализуют интерфейс Deferred. Пример работы на странице when.html

Теперь можно и заумно – метод «`.when()`» возвращает проекцию Deferred объекта, принимает в качестве параметров произвольное множество Deferred объектов, когда все из них отработают, объект «when» изменит своё состояние в «выполнено», с последующим вызовом всех подписавшихся.

Callbacks

Callbacks – это крутой объект – он позволяет составлять списки функций обратного вызова, а также даёт бразды правления над ними. Работать с ним проще нежели с Deferred, тут нет разделения на позитивный и негативный сценарии, лишь стек функций, который будет выполнен по команде «`.fire()`»:

```
var C = $.Callbacks();
C.add(function(msg) {
      console.log(msg+" first")
});
C.add(function(msg) {
      console.log(msg+" second")
});

C.fire("Go");
>>>
      Go first
      Go second
```

— А в чём сила, брат?
— В аргументах

По умолчанию, вы можете прямо из консоли вызвать метод «`.fire()`» снова и снова, и будете получать один и тот же результат раз за разом. А можно задать поведение Callbacks через флаги:

`once` — все функции будут вызваны единожды (аналогично как в объекте Deferred).

`memory` — сохранять значение с последнего вызова «`.fire()`», и скармливать его в ново-зарегистрированные функции обратного вызова, и лишь потом обрабатывает новое значение (в Deferred именно так).

`unique` — список функций обратного вызова фильтруется по уникальности

`stopOnFalse` — как только какая-нить функция вернёт «`false`», процесс запуска остановится

Наверное, будет лучше с примерами, вот «once»:

```
var C = $.Callbacks("once");
C.add(function(msg) {
      console.log(msg+" first")
});
C.add(function(msg) {
      console.log(msg+" second")
});
C.fire("Go");
C.fire("Again"); // не даст результата, только Go
>>>
      Go first
      Go second
```

С «memory» посложнее, будьте внимательней:

```
var C = $.Callbacks("memory");
C.add(function(msg) {
      console.log(msg+" first")
});
C.fire("Go");
C.add(function(msg) {
      console.log(msg+" second")
});
C.fire("Again");
>>>
      Go first
      Go second // без флага, этой строчки не было бы
      Again first
      Again second
```

Пример с уникальностью прост до безобразия:

```
var C = $.Callbacks("unique");
var func = function(msg) {
      console.log(msg+" first")
};
C.add(func);
C.add(func);  // эта строка не повлияет на результат
C.fire("Go"); // только Go first
>>>
      Go first
```

Флаг «stopOnFalse»:

```
var C = $.Callbacks("stopOnFalse");
C.add(function(msg) {
      console.log(msg+" first");
      return false; // вот он – роковой false
      });
C.add(function(msg) { console.log(msg+" second") });
C.fire("Go"); // только Go first
>>>
      Go first
```

Перечисленные флаги можно комбинировать и получать интересные результаты, а можно не получать, а лишь посмотреть на пример callbacks.html

Из истории: объект «Deferred» отпочковался от метода «$.ajax()» в результате рефакторинга версии 1.5. Шло время, появлялись новые версии jQuery, и вот новый виток рефакторинга – результатом стало отделение «Callbacks» от «Deferred» в версии 1.7, таким образом в текущей версии библиотеки метод «$.ajax()» работает с объектом «Deferred», который является надстройкой над «Callbacks». Дабы не вносить путаницу в терминологию, я использую определение «Deferred Callbacks» и при работе с «Callbacks», ибо колбэков много, и каждый раз уточнять, что я говорю именно «о том самом» — дело достаточно утомительное.

Статьи по данной теме:

- «Что такое этот новый jQuery.Callbacks Object» [http://habrahabr.ru/post/135821/]
- «jQuery Deferred Object (подробное описание)» [http://habrahabr.ru/post/113073/]
- «Async JS: The Power of $.deffered» [http://www.html5rocks.com/en/tutorials/async/deferred/]

90% Пишем свой плагин

jQuery плагин

Для начала вспомним, для чего нам нужны плагины? Мой ответ — создание повторно используемого кода, и да — с удобным интерфейсом. Давайте напишем такой код, вот простая задачка: «По клику на параграф, текст должен измениться на красный»

JavaScript и даже не jQuery

Дабы не забывать истоков — начнем с реализации на нативном JavaScript'e:

```
var loader = function () {
	// находим все параграфы
	var para = document.getElementsByTagName('P');
	// перебираем все, и вешаем обработчик
	for (var i=0,size=para.length;i<size;i++) {
		// обработчик
		para[i].onclick = function() {
			this.style.color = "#FF0000";
		}
	}
}

// естественно, весь код должен работать после загрузки всей страницы
document.addEventListener("DOMContentLoaded", loader, false);
```

Данный код не является кроссбраузерным, и написан с целью лишний раз подчеркнуть удобство использования библиотеки ;)

jQuery, но еще не плагин

Теперь можно этот код упростить, подключаем jQuery и получаем следующий вариант:

```
$(function(){
	$('p').click(function(){
		$(this).css('color', '#ff0000');
	})
});
```

Таки jQuery плагин

С поставленной задачей мы справились, но где тут повторное использование кода? Или если нам надо не в красный, а в зеленый перекрасить? Вот тут начинается самое интересное, чтобы написать простой плагин достаточно расширить объект «`$.fn`»:

```
$.fn.mySimplePlugin = function() {
	$(this).click(function(){
		$(this).css('color', '#ff0000');
	})
}
```

Если же писать более грамотно, то нам необходимо ограничить переменную `$` только нашим плагином, а так же возвращать «`this`», чтобы можно было использовать цепочки вызовов (т.н. «`chaining`»), делается это следующим образом:

```
(function($) {
	$.fn.mySimplePlugin = function(){
		// код плагина
		return this;
	};
})(jQuery);
```

Внесу небольшое пояснение о происходящей тут «магии», код «(function($){…})(jQuery)» создает анонимную функцию, и тут же вызывает ее, передавая в качестве параметра объект jQuery, таким образом внутри анонимной функции мы можем использовать алиас $ не боясь за конфликты с другими библиотеками — так как теперь $ находится лишь в области видимости нашей функции, и мы имеем полный контроль над ней. Если у вас возникло ощущение дежавю – то всё верно, я об этом уже рассказывал

Добавим опцию по выбору цвета и получим рабочий плагин (см. plugin.global.html):

```
(function($) {
	// значение по умолчанию - ЗЕЛЁНЫЙ
	var defaults = { color:'green' };
	// актуальные настройки, глобальные
	var options;
	$.fn.mySimplePlugin = function(params){
		// при многократном вызове настройки будут сохранятся
		// и замещаться при необходимости
		options = $.extend({}, defaults, options, params);
		$(this).click(function(){
			$(this).css('color', options.color);
		});
		return this;
	};
})(jQuery);
```

Вызов:

```
// первый вызов
$('p:first,p:last').mySimplePlugin();
// второй вызов
$('p:eq(1)').mySimplePlugin({ color: 'red' });
```

В результате работы данного плагина, каждый клик будет изменять цвет параграфа на красный, т.к. мы используем глобальную переменную для хранения настроек, то второй вызов плагина изменят значение для всех элементов.

Можно внести небольшие изменения, и разделить настройки для каждого вызова (см. plugin.html):

```
// актуальные настройки, будут индивидуальными при каждом запуске
var options = $.extend({}, defaults, params);
```

А разница то в одном «var». Мне даже сложно себе представить как много часов убито в поисках потерянного «var» в JavaScript'e, будьте внимательны

Работаем с коллекциями объектов

Тут все просто, достаточно запомнить — «this» содержит jQuery объект с коллекцией всех элементов, т.е.:

```
$.fn.mySimplePlugin = function(){
      console.log(this); // это jQuery объект
      console.log(this.length); // число элементов в выборке
};
```

Если мы хотим обрабатывать каждый элемент то соорудим следующую конструкцию внутри нашего плагина:

```
// необходимо обработать каждый элемент в коллекции
return this.each(function(){
      $(this).click(function(){
            $(this).css('color', options.color);
      });
});
// предыдущий вариант немного избыточен,
// т.к. внутри функции click и так есть перебор элементов
return this.click(function(){
      $(this).css('color', options.color);
});
```

Опять же напомню, если ваш плагин не должен что-то возвращать по вашей задумке — возвращайте «this» — цепочки вызовов в jQuery это часть магии, не стоит её разрушать. Методы «.each()» и «.click()» возвращают объект jQuery.

Публичные методы

Так, у нас написан крутой плагин, надо бы ему еще докрутить функционала, пусть цвет регулируется несколькими кнопками на сайте. Для этого нам понадобится некий метод «color», который и будет в ответе за всё. Сейчас приведу пример кода готового плагина — будем курить вместе (обращайте внимание на комментарии):

```
// настройки со значением по умолчанию
var defaults = { color:'green' };

// наши будущие публичные методы
var methods = {
	// инициализация плагина
	init: function(params) {

		// настройки, будут индивидуальными при каждом запуске
		var options = $.extend({}, defaults, params);

		// инициализируем лишь единожды
		if (!this.data('mySimplePlugin')) {
			// закинем настройки в реестр data
			this.data('mySimplePlugin', options);
			// добавим событий
			this.on('click.mySimplePlugin', function(){
				$(this).css('color', options.color);
			});
		}
		return this;
	},

	// изменяем цвет в реестре
	color: function(color) {
		var options = $(this).data('mySimplePlugin');
		options.color = color;
		$(this).data('mySimplePlugin', options);
	},

	// сброс цвета элементов
	reset: function() {
		$(this).css('color', 'black');
	}
};
```

```
$.fn.mySimplePlugin = function(method){
	// немного магии
	if ( methods[method] ) {
		// если запрашиваемый метод существует, мы его вызываем
		// все параметры, кроме имени метода прийдут в метод
		// this так же перекочует в метод
		return methods[ method ].apply( this, Array.prototype.slice.call(
		arguments, 1 ));
	} else if ( typeof method === 'object' || ! method ) {
		// если первым параметром идет объект, либо совсем пусто
		// выполняем метод init
		return methods.init.apply( this, arguments );
	} else {
		// если ничего не получилось
		$.error('Метод "' +  method + '" в плагине не найден');
	}
};
```

Теперь еще небольшой пример использование данных методов:

```
// вызов без параметров - будет вызван init
$('p').mySimplePlugin();

// вызов метода color и передача цвета в качестве параметров
$('p').mySimplePlugin('color', '#FFFF00');

// вызов метода reset
$('p').mySimplePlugin('reset');
```

Для понимания данного кусочка кода, вы должны разобраться лишь с переменной «`arguments`», и с методом «`apply()`». Тут им целые статьи посвятили, дерзайте:

— *http://www.seifi.org/?p=673*
— *https://learn.javascript.ru/arguments-pseudoarray*
— *https://learn.javascript.ru/call-apply*

О обработчиках событий

Если ваш плагин вешает какой-либо обработчик, то лучше всего (читай всегда) данный обработчик повесить в своём собственном namespace:

```
return this.on("click.mySimplePlugin", function(){
      $(this).css('color', options.color);
});
```

Данный финт позволит в любой момент убрать все ваши обработчики, или вызвать только ваш, что очень удобно:

```
// вызовем лишь наш обработчик
$('p').trigger("click.mySimplePlugin");

// убираем все наши обработчики
$('p').off(".mySimplePlugin");
```

Дежавю? Ок!

На этом об обычных плагинах всё, хотя дам ещё чуток информации к размышлению, но на английском:

— «Essential jQuery Plugin Patterns» [http://coding.smashingmagazine.com/?p=115389]

Data

Если по какой-то причине вы еще не знакомы с «.data()» — то советую прочитать документацию и усвоить незамедлительно. Если же в двух словах — это реестр данных, и все данные привязанные к какому-либо элементу лучше хранить в нём, это же правило касается и плагинов. Если вам надо сохранить состояние плагина — используйте «.data()», если необходим кеш — используйте «.data()», если вам необходимо сохранить … ну думаю понятно. Приведу еще примерчик связанный с инициализацией:

```
function() {  // функция init
      var init = $(this).data('mySimplePlugin');
      if (init) {
            return this;
      } else {
            $(this).data('mySimplePlugin', true);
            return this.on('click.mySimplePlugin', function(){
                  $(this).css('color', options.color);
            });
      }
}
```

По стечению обстоятельств, в HTML5 появились data-атрибуты, и для доступа к ним jQuery использует тот же метод «.data()», но вот дела, «jQuery.data()» – не манипулирует атрибутами HTML, а работает со своим реестром, и лишь при отсутствии там данных пытается заполучить атрибут «data-*», не попадитесь:

```
<div id="my" data-foo="bar"></div>
      $("#my").data("foo");      // >>> bar
      $("#my").attr("data-foo"); // >>> bar

$("#my").data("foo", "xyz");
      $("#my").data("foo");      // >>> xyz
      $("#my").attr("data-foo"); // >>> bar

$("#my").attr("data-foo", "def");
      $("#my").data("foo");      // >>> xyz
      $("#my").attr("data-foo"); // >>> def
      <div id="my" data-foo="def"></div>
```

Animate

> *Информация в данном разделе актуальна для jQuery версии 1.8 и выше, если вас заинтересуют возможности расширения для более старых версий, то читайте мою статью «Пишем плагины анимации»*

Для начала затравка – метод «`.animate()`» манипулирует объектом «`jQuery.Animation`», который предусматривает следующие точки для расширения функционала:

— jQuery.Tween.propHooks
— jQuery.Animation.preFilter
— jQuery.Animation.tweener

Начну рассказ с «`jQuery.Tween.propHooks`», т.к. уже есть плагины, в код которых можно заглянуть :) Для пущей наглядности я возьму достаточно тривиальную задачу – заставим плавно изменить цвет шрифта для заданного набора элементов:

```
$('p').animate({color:'#ff0000'});
```

Приведенный выше код не даст никакого эффекта, т.к. свойство «`color`» библиотека из коробки не анимирует, но это можно исправить – надо лишь прокачать «`jQuery.Tween.propHooks`»:

```
$.Tween.propHooks.color = {
      get: function(tween) {
            return tween.elem.style.color;
      }
      set: function(tween) {
            tween.easing;  // текущий easing
            tween.elem;    // испытуемый элемент
            tween.options; // настройки анимация
            tween.pos;     // текущий прогресс
            tween.prop;    // свойство которое изменяем
            tween.start;   // начальные значения
            tween.now;     // текущее значение
            tween.end;     // желаемое результирующие значения
            tween.unit;    // еденицы измерения
      }
}
```

Этот код ещё не работает, это наши кирпичики, с их помощью будем строить нашу анимацию. Перед работой стоит заглянуть внутрь каждого из приведенных свойств:

```
console.log(tween);
>>>
      easing: "swing"
      elem: HTMLParagraphElement
      end: "#ff0000"
      now: "NaNrgb(0, 0, 0)"
      options: Object
            complete: function (){}
            duration: 1000
            old: false
            queue: "fx"
            specialEasing: Object
      pos: 1
      prop: "color"
      start: "rgb(0, 0, 0)"
      unit: "px"
```

В консоле у нас будет очень много данных, т.к. приведённый метод вызывается N кол-во раз, в зависимости от продолжительности анимации, при этом «`tween.pos`» постепенно наращивает своё значение с `0` до `1`. По умолчанию, наращивание происходит линейно, если надо как-то иначе — то стоит посмотреть на easing плагин или дочитать раздел до конца (об этом я уже упоминал в главе Анимация)

Даже при таком раскладе мы уже можем изменять выбранный элемент (путём манипуляций над «`tween.elem`»), но есть более удобный способ – можно установить свойство «`run`» объекта «`tween`»:

```
$.Tween.propHooks.color = {
      set: function(tween) {
            // тут будет инициализация
            tween.run = function(progress) {
                  // тут код отвечающий за измение свойств элемента
            }
      }
}
```

Получившийся код будет работать следующим образом:

1. единожды будет вызвана функция «set»
2. функция «run()» будет вызвана N-раз, при этом «progress» будет себя вести аналогично «tween.pos»

Теперь, возвращаясь к изначальной задачи по изменению цвета можно наворотить следующий код:

```
$.Tween.propHooks.color = {
	set: function(tween) {
		// приводим начальное и конечное значения к единому формату
		// #FF0000 == [255,0,0]
		tween.start = parseColor(tween.start);
		tween.end = parseColor(tween.end);
		tween.run = function(progress) {
			tween.elem.style['color'] =
				// вычисляем промежуточное значение
				buildColor(tween.start, tween.end, progress);
		}
	}
}
```

Код функций «parseColor()» и «buildColor()» вы найдёте в листинге на странице color.html

Результатом станет плавное перетекание исходного цвета к красному (#F00 == #FF0000 == 255,0,0), вживую можно посмотреть на странице color.html

В плагине jQuery Color для решения поставленной задачи использовали jQuery.cssHooks, но мы же не ищем лёгких путей.

Ещё хотел было рассказать про префильтры анимации, но – документации нет, а как использовать «в жизни» – я не догадался, но чуть-чуть информации таки накопал (код можно найти в функции «Animation»):

```
jQuery.Animation.prefilter(function(element, props, opts) {
	// deffered объект animate
	element; // искомый элемент
	props;   // настройки анимации из animate()
	opts;    // опции анимации
	// отключаем анимацию при попытке анимировать высоту элемента
	if (props['height'] != undefined) {
		return this;
	}
});
```

Пример можно пощупать [animate.prefilter.html]

Про «jQuery.Animation.tweener» так же много не расскажешь, но пример получилось сделать чуток интересней – приведённый код позволяет анимировать ширину и высоту объекта по заданной диагонали:

Осторожно, для понимания происходящего потребуются знания геометрии за 8-ой класс

```
// создаём поддержку нового свойства для анимации – diagonal
jQuery.Animation.tweener( "diagonal", function( property, value ) {
	// создаём tween объект
	var tween = this.createTween( property, value );
	// промежуточные вычисления и данные
	var a = jQuery.css(tween.elem, 'width', true );
	var b = jQuery.css(tween.elem, 'height', true );
	var c = Math.sqrt(a*a + b*b), sinA = a/c, sinB = b/c;
	tween.start = c;
	tween.end = value;
	tween.run = function(progress) {
		// вычисление искомого значения – новое значение гипотенузы
		var hyp = this.start + ((this.end - this.start) * progress);
		// непосредственно измение свойств элемента
		tween.elem.style.width = sinA*hyp + tween.unit;  // ширина
		tween.elem.style.height = sinB*hyp + tween.unit; // высота
	};
	return tween;
});
```

Пример работы [animate.tweener.html]

Easing

Теперь опять обратимся к easing'у – приведу пример произвольной функции, которой будет следовать анимация. Дабы особо не фантазировать – я взял пример из статьи на вездесущем хабре о анимации в MooTools фреймворке [http://habrahabr.ru/post/43379/] – наглядный пример с сердцебиением, которое описывается следующими функциями:

$$y = \begin{cases} x^4 * 49.4 & x < 0.3 \\ 9x - 2.3 & x < 0.4 \\ -13x + 6.5 & x < 0.5 \\ 4x - 2 & x < 0.6 \\ 0.4 & x < 0.7 \\ 4x - 2.4 & x < 0.75 \\ -4x + 3.6 & x < 0.8 \\ 1 - \sin(\operatorname{acos}(x)) & x \geq 0.8 \end{cases}$$

В расширении функционала easing нет ничего военного:

```
$.extend($.easing, {
	/**
	 * Heart Beat
	 *
	 * @param x progress
	 * @link http://habrahabr.ru/blogs/mootools/43379/
	 */
	heart:function(x) {
		if (x < 0.3)  return Math.pow(x, 4) * 49.4;
		if (x < 0.4)  return 9 * x - 2.3;
		if (x < 0.5)  return -13 * x + 6.5;
		if (x < 0.6)  return 4 * x - 2;
		if (x < 0.7)  return 0.4;
		if (x < 0.75) return 4 * x - 2.4;
		if (x < 0.8)  return -4 * x + 3.6;
		if (x >= 0.8) return 1 - Math.sin(Math.acos(x));
	}
});
```

Чуть-чуть пояснений, конструкция «$.extend({}, {})» «смешивает» объекты:

```
$.extend({name:"Anton"}, {location:"Kharkiv"});
>>>
      {
            name:"Anton",
            location:"Kharkiv"
      }
$.extend({name:"Anton", location:"Kharkiv"}, {location:"Kyiv"});
>>>
      {
            name:"Anton",
            location:"Kyiv"
      }
```

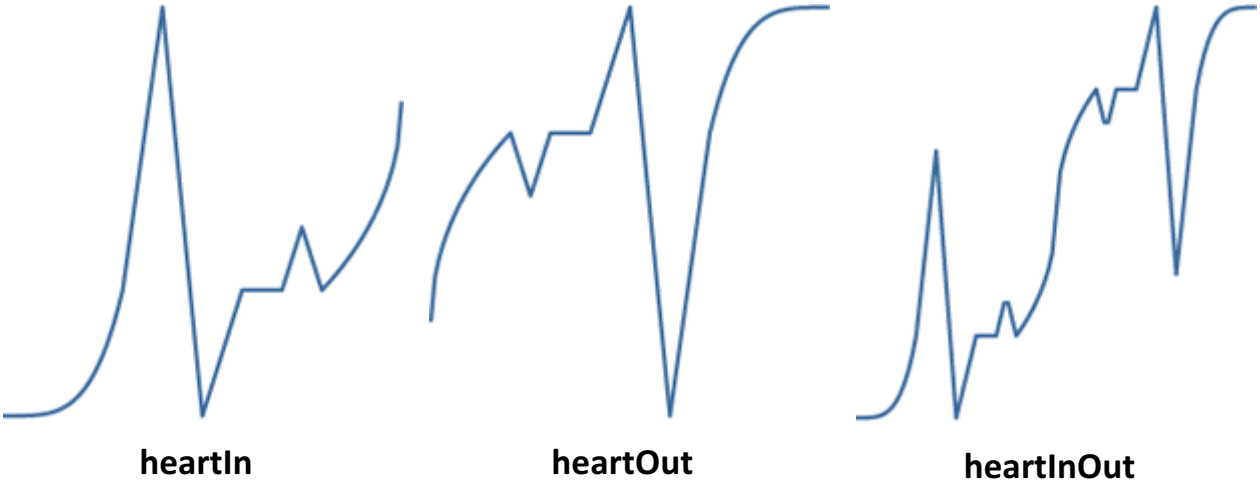
Таким образом мы «вмешиваем» новый метод к существующему объекту «$.easing»; согласно коду, наш метод принимает в качестве параметра лишь одно значение:

x – коэффициент прохождения анимации, изменяется от 0 до 1, дробное

Результат конечно интересен, но его можно ещё чуть-чуть расширить дополнительными функциями (развернем и скомбинируем):

```
heartIn: function (x) {
      return $.easing.heart(x);
},
heartOut: function (x) {
      return $.easing.heart(1 - x);
},
heartInOut: function (x) {
      if (x < 0.5) return $.easing.heartIn(x);
      return $.easing.heartOut(x);
}
```

Получим следующие производные функции:



Работать с данным творением надо следующим образом:

```
$("#my").animate({height:"+200px"}, 2000, "heartIn"); // вот оно
```

Пример работы данной функции можно пощупать на странице easing.html

Sizzle

Когда я рассказывал о Sizzle я решил вас не грузить возможностями по расширению библиотеки, но вот время настало… В Sizzle можно расширять много чего:

— Sizzle.selectors.match
— Sizzle.selectors.find
— Sizzle.selectors.filter
— Sizzle.selectors.attrHandle
— Sizzle.selectors.pseudos

Но браться мы будем лишь за расширение псевдо-селекторов, наподобие:

```
$("div:animated"); // поиск анимированных элементов
$("div:hidden");   // поиск скрытых элементов div
$("div:visible");  // поиск видимых элементов div
```

Почему я привел только эти фильтры? Всё просто — только они не входят в Sizzle, и относятся лишь к jQuery, именно такие плагины мы будем тренироваться разрабатывать. Начнем с кода фильтра «`:visible`»:

```
// пример для расширения Sizzle внутри jQuery
// для расширения самого Sizzle нужен чуть-чуть другой код
jQuery.expr.pseudos.visible = function( elem ) {
      // проверяем ширину и высоту каждого элемента в выборке
      return !!( elem.offsetWidth || elem.offsetHeight);
};
```

Выглядит данный код несложно, но, пожалуй, я таки дам каркас для нового фильтра и добавлю чуть-чуть пояснений:

```
$.extend($.expr.pseudos, {
      /**
       * @param element   DOM элемент
       * @param i         порядковый номер элемента
       * @param match     объект матчинга регулярного выражения
       * @param elements  массив всех найденных DOM элементов
       */
      test: function(element, i, match, elements) {
            /* тут будет наш код, и будет решать кто виноват */
            return true || false; // выносим вердикт
      }
})
```

Ну теперь попробуем решить следующую задачку:

> *— Необходимо выделить ссылки в тексте в зависимости от её типа: внешняя, внутренняя или якорь*

Для решения лучше всего подошли бы фильтры для селекторов следующего вида:

```
$("a:internal");
$("a:anchor");
$("a:external");
```

Поскольку «из коробки» данный функционал не доступен, мы напишем его сами, для этого нам понадобится не так уж и много (пример лишь для последнего «`:external`», рабочий код на странице sizzle.filter.html):

```
$.extend($.expr.pseudos, {
	// определения внешней ссылки
	// нам понадобится лишь DOM Element
	external: function(element) {
		// а у нас ссылка?
		if (element.tagName.toUpperCase() != 'A') return false;
		// есть ли атрибут href
		if (element.getAttribute('href')) {
			var href = element.getAttribute('href');
			// отсекаем ненужное
			if ((href.indexOf('/') === 0) // внутренняя ссылка
			  || (href.indexOf('#') === 0) // якорь
				// наш домен по http:// или https://
			  || (href.indexOf(window.location.hostname) === 7)
			  || (href.indexOf(window.location.hostname) === 8)
			) {
				return false; // мимо
			} else {
				return true;  // да, мы нашли внешние ссылки
			}
		} else {
			return false;
		}
	}
})
```

ВСЕГДА используйте фильтр вместе с HTML тэгом который ищите:

```
$("tag:filter")
```

Это один из пунктов оптимизации работы с фильтрами jQuery, иначе ваш фильтр будет обрабатывать все DOM элементы на странице, а это может очень сильно сказаться на производительности. Если же у вас несколько тегов, то пишите уж лучше так — «`$("tag1:filter, tag2:filter, tag3:filter")`», или ещё лучше через вызов метода «`.filter()`».

– «Sizzle Documentation» — скудненькая официальная документация [https://github.com/jquery/sizzle/wiki/Sizzle-Documentation]

100% Последняя глава

Вы думаете, я ещё чему-то смогу вас научить? Сомневаюсь. Я думаю, эту главу вы напишите сами. И если вам будет не в лом, то даже пришлёте её мне на почтовый ящик Anton.Shevchuk@gmail.com

Дополнение

jQuery-inlog

Еще чуть-чуть о полезном инструментарии: есть такой классный плагин – jQuery-inlog [http://prinzhorn.github.com/jquery-inlog/] – основное его назначение — дать нам чуть-чуть больше понимания о происходящем внутри самого jQuery, вот кусочек HTML:

```
<body>
      <div class="bar">
            <div class="bar">
                  <div id="foo"></div>
            </div>
      </div>
      <div id="bacon"></div>
</body>
```

А вот и код, который его обслуживает:

```
$l(true);
$("#foo").parents(".bar").next().prev().parent().fadeOut();
$l(false);
```

Какие-то странные манипуляции, для какого же элемента будет применён метод «`.fadeOut()`»? Для выяснения оного наш код обёрнут в вызов метода «`$l()`». «`$l()`» — это и есть собственно вызов плагина, результат его работы можно найти в консоли:

```
$("#foo") ↷ ⊞ [ div#foo ]
parents(".bar") ↷ ⊞ [ div.bar, div.bar ]
next() ↷ ⊞ [ div#bacon ]
prev() ↷ ⊞ [ div.bar ]
parent() ↷ ⊞ [ body ]
fadeOut() ↷ ⊞ [ body ]
```

У данного плагина есть ещё настройки, которые регулируют объём информации выводимой в консоль.

Пример и скриншот взять с официальной документации по плагину

jQuery UI

jQuery UI представляет из себя набор виджетов и плагинов от самих разработчиков jQuery. По моему мнению, данный инструмент необходимо изучить настолько, насколько это требуется чтобы не писать свои «велосипеды». Скачать-почитать о данной надстройке над jQuery можно на домашней страницы проекта – http://jqueryui.com/

Что нам необходимо знать о виджетах и плагинах? Первое – это какие они есть, и второе – как работают. На этих двух моментах я и постараюсь остановиться.

Интерактивность

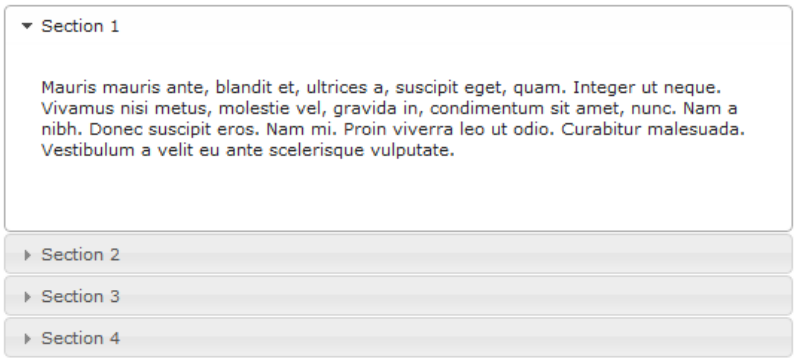
Начну с полезных плагинов, которые могут упростить жизнь при создании интерактивных интерфейсов:

- Draggable [http://jqueryui.com/position/] – данный компонент позволяет сделать любой DOM элемент перетаскиваемым при помощи мыши
- Droppable [http://jqueryui.com/droppable/] – это логичное продолжение draggable компонента, необходим для работы с контейнерами, внутрь которых можно перетаскивать элементы
- Resizable [http://jqueryui.com/resizable/] – как следует из название – даёт возможность растягивать любые DOM элементы
- Selectable [http://jqueryui.com/selectable/] – позволяет организовать «выбор» элементов, удобно использовать для организации менеджмента картинок
- Sortable [http://jqueryui.com/sortable/] – сортировка DOM элементов

Виджеты

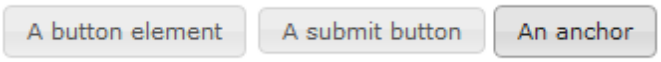
Виджеты – это уже комплексное решение содержащие не только JavaScript код, но и некую HTML и CSS реализацию:

— Accordion – данный виджет следует использовать если у вас уже используется jQuery UI в проекте, сам по себе основной его функционал можно реализовать в несколько строк (посмотреть можно в accordion.html)



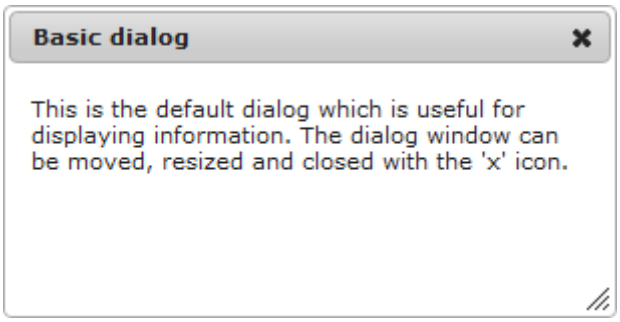
— Autocomplete – как и следует из название, данный виджет отвечает за добавление функции автодополнения к полям ввода, естественно с поддержкой AJAX

— Button – создание кнопок используя JavaScript – ещё тот моветон, но возможно пригодится, если вы сильно завязались на jQuery UI:



— Datepicker – если ваш браузер не поддерживает в полной мере спецификацию HTML5 и `<input type="date"/>` в частности, то потребуется эмуляция данной возможности с помощью виджета:

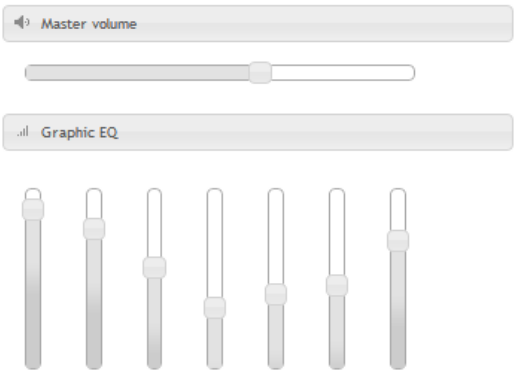
— Dialog – виджет предназначенный для создания слегка неуклюжих диалоговых окон:



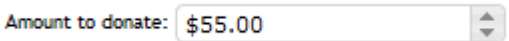
— Menu – создание меню из списка, с поддержкой вложенности

— Progressbar – название говорит само за себя, и да в HTML5 он тоже включён:

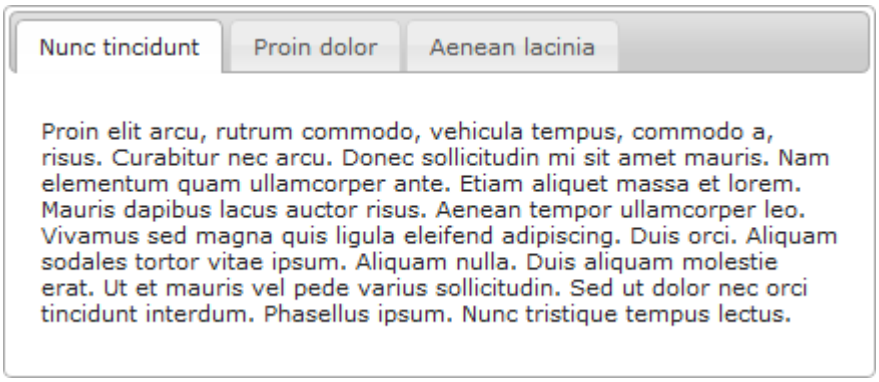
— Slider – ещё один виджет для устаревших браузеров:



— Spinner – ещё один удобный контрол для форм, опять же – в HTML5 уже есть:



— Tabs – они же табы – достаточно популярный элемент в web-разработке, и так же как и «accordion» вполне заменяем простым кодом (см. tabs.html)



— Tooltip – вот и последний виджет – всплывающие подсказки, простой и должен быть востребован, ну а там жизнь покажет

На этом обзор виджетов можно считать законченным, вернёмся к плагинам.

> *Все виджеты и плагины завязаны на ядро jQuery UI, но есть так же зависимости между самими плагинами и стоит о них помнить. Но не переживайте – при сборке jQuery UI пакета все зависимости проверяются автоматически, т.е. когда вам потребуется доселе неподключенный виджет, лучше скачать сборку заново.*

Утилиты

Утилит у нас не много – вот полезный плагин – position, который позволяет контролировать положение DOM элементов – http://jqueryui.com/position/, а ещё есть фабрика по созданию виджетов, но о ней я расскажу чуть попозже.

Эффекты

Среди эффектов предоставляемых jQuery UI я выделяю четыре пункта:

- Анимация цвета
- Анимация изменения классов
- Набор эффектов
- Расширения возможностей easing

За анимацию цвета отвечает компонент «Effects Core», который позволяет анимировать изменения цвета посредством использования функции «`.animate()`»:

```
$("#my").animate({ backgroundColor: "black" }, 1000);
```

Да-да, jQuery из коробки не умеет этого делать, а вот jQuery UI позволяет анимировать следующие параметры:

- backgroundColor
- borderBottomColor
- borderLeftColor
- borderRightColor
- borderTopColor
- color
- outlineColor

Ещё одной возможностью заключенной в «Effects Core» является анимация изменений класса DOM элемента, т.е. когда вы будете присваивать новый класс элементу, то вместо обычного моментального применения новых CSS свойств вы будете наблюдать анимацию этих свойств от текущих до заданных в присваиваемом классе. Для использования данного функционала нам потребуются старые знакомые – методы «`.addClass()`», «`.toggleClass()`» и «`.removeClass()`», с одной лишь разницей – при вызове метода вторым параметром должна быть указана скорость анимации:

```
$("#my").addClass("active", 1000);
$("#my").toggleClass("active", 1000);
$("#my").removeClass("active", 1000);
```

Если из предыдущего абзаца у вас не возникло понимания сути происходящего, то этот код для вас:

```
<style>
	#my {
		font-size:14px;
	}
	#my.active {
		font-size:20px;
	}
</style>
<script>
	$(function (){
		$("#my").addClass("active", 1000);
		// тут получается аналогично следующему вызову
		$("#my").animate({"font-size":"20px"}, 1000);
	});
</script>
```

А ещё появляется метод «`.switchClass()`», который заменяет один класс другим, но мне он ни разу не пригодился.

О наборе эффектов я не буду долго рассказывать, их лучше посмотреть в действии на странице http://jqueryui.com/effect/. Для работы с эффектами появляется метод «`.effect()`», но сам по себе его лучше не использовать, ведь UI расширил функционал встроенных методов «`.show()`», «`.hide()`» и «`.toggle()`», теперь, передав в качестве параметра скорости анимации названия эффекта вы получите необходимый результат:

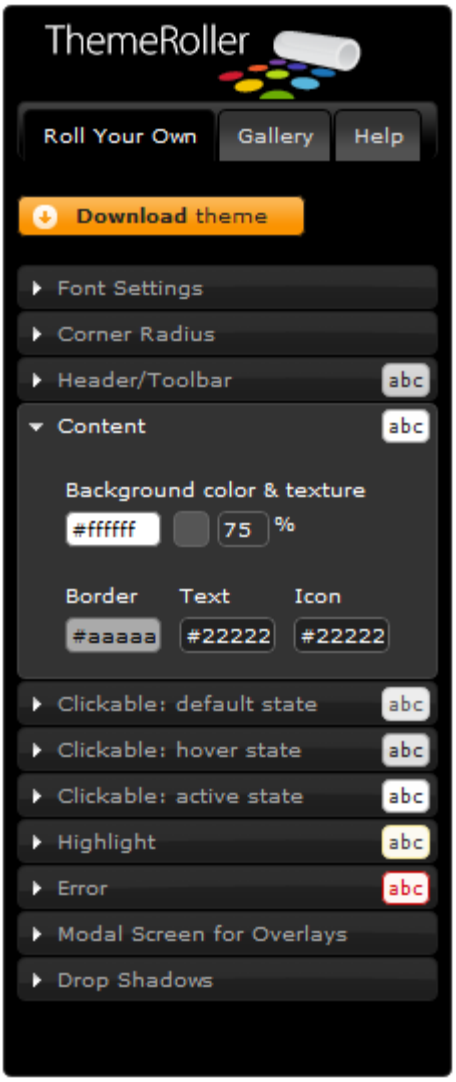
```
$("#my").hide("puff");
$("#my").show("transfer");
$("#my").toggle("explode");
```

Приведу список эффектов, может кто запомнит: blind, bounce, clip, drop, explode, fold, highlight, puff, pulsate, scale, shake, size, slide, transfer.

Помните, я в главе о анимации рассказывал о easing и одноименном плагине для jQuery? Так вот UI тоже расширяет easing, так что подключив его можно отключать плагин. И да, этот функционал завязан лишь на «Effects Core».

Темы

Одной из самых замечательных особенностей jQuery UI является возможность менять «шкурки» всех виджетов разом, и для этого даже предусмотрена специальная утилита – ThemeRoller [http://jqueryui.com/themeroller/]:



Если в какой-то момент времени потребуется внести изменения в тему, то откройте файл jquery-ui-#.#.##-custom.css и найдёте строчку начинающуюся с текста «To view and modify this theme, visit http://...» и таки пройдите по указанной ссылке, и уже используя ThemeRoller внесите необходимые изменения.

Пишем свой виджет

Отправной точкой при написания виджета для jQuery UI для вас будет официальная документация, но поскольку со знанием английского не у всех сложилось, то я постараюсь перевести и адаптировать информацию изложенную в ней.

Первое, о чём стоит рассказать, это то, что правила написания плагинов для jQuery слишком вальяжны, что не способствует их качеству. При создании jQuery UI, походу, решили пойти путём стандартизации процесса написания плагинов и виджетов, я не могу сказать насколько задумка удалась, но стало явно лучше чем было. Начну с описания каркаса для вашего виджета:

```
$.widget("book.expose", {
	// настройки по умолчанию
	options: {
		color: "red"
	},
	// инициализация widget
	// вносим изменения в DOM и вешаем обработчики
	_create: function() {
		this.element;   // искомый объект в jQuery обёртке
		this.name;      // имя - expose
		this.namespace; // пространство – book
		this.element.on("click."+this.eventNamespace, function(){
			console.log("click");
		})
	},
	// метод отвечает за применение настроек
	_setOption: function( key, value ) {
		// применяем изменения настроек
		this._super("_setOption", key, value );
	},
	// метод _destroy должен быть антиподом к _create
	// он должен убрать все изменения внесенные изменения в DOM
	// и убрать все обработчики, если таковые были
	_destroy: function() {
		this.element.off('.'+this.eventNamespace);
	}
});
```

Поясню для тех кто не прочёл комментарии:

`options` – хранилище настроек виджета для конкретного элемента

`_create()` – отвечает за инициализацию виджета – тут должны происходить изменения в DOM'е, и «вешаться» обработчики событий

`_destroy()` – антипод для «`_create()`» – должен подчистить всё, что мы намусорили

`_setOption(key, value)` – данный метод будет вызван при попытке изменить какие-либо настройки:

```
$("#my").expose({key:value})
```

Наблюдательный глаз заметит, что все перечисленные методы начинаются со знака подчёркивания – это такой способ выделить «приватные» методы, которые недоступны для запуска, и если мы попытаемся запустить «`$('#my').expose('_destroy')`», то получим ошибку. Но учтите – это лишь договорённость, соблюдайте, её!

Для обхода договорённости о приватности можно использовать метод «`data()`»:

```
$("#my").data("expose")._destroy() // место для смайла «(evil)»
```

В данном примере, я постарался задать хороший тон написания виджетов – я «повесил» обработчики событий в namespace, это даст в дальнейшем возможность контролировать происходящее без необходимости залазить в код виджета, это «true story».

Код описанный в методе «`_destroy()`» – избыточен, т.к. он и так выполняется в публичном «`destroy()`», приведён тут для наглядности.

А для ленивых, чтобы не прописывать каждый раз «`eventNamespace`» в обработчиках событий, разработчики добавили в версии 1.9.0 два метода: «`_on()`» и «`_off()`», первый принимает два параметра:

— DOM элемент, или селектор, или jQuery объект

— набор обработчиков событий в виде объекта

Все перечисленные события будут «висеть» в пространстве «eventNamespace», т.е. результат будет предположительно одинаковым:

```
this._on(this.element, {
	mouseover:function(event) {
		console.log("Hello mouse");
	},
	mouseout:function(event) {
		console.log("Bye mouse");
	}
});
```

Второй метод – «_off()» – позволяет выборочно отключать обработчики:

```
this._off(this.element, "mouseout click");
```

Ну каркас баркасом, пора переходить к функционалу – добавим произвольную функцию с произвольным функционалом:

```
callMe:function(){
	console.log("Allo?");
}
```

К данной функции мы легко сможем обращаться как из других методов виджета так и извне:

```
// изнутри
this.callMe()

// извне
$("#my").expose("callMe")
```

Если ваша функция принимает параметры, то передача оных осуществляется следующим способом:

```
$("#my").expose("callMe", "Hello!")
```

Если вы хотите достучаться в обработчике событий до метода виджета, то не забудьте про область видимости переменных, и сделайте следующий манёвр:

```
_create: function() {
      var self = this;   // вот он!
      this.element.on("click."+this.eventNamespace, function(){
            // тут используем self, т.к. this уже указывает на
            // элемент по которому кликаем
            self.callMe();
      })
},
```

Хорошо идём, теперь поговорим о событиях – для более гибкой разработки и внедрения виджетов предусмотрен функционал по созданию произвольных событий и их «прослушиванию»:

```
// инициируем событие
this._trigger("incomingCall");

// подписываемся на событие при инициализации виджета
$("#my").expose({
      incommingCall: function(ev) {
            console.log("din-don");
      }
})

// или после, используя в качестве имени события
// имя виджета + имя события
$("#my").on("exposeincomingCall", function(){
      console.log("tru-lya-lya")
});
```

Материала много, я понимаю, но ещё добавлю описание нескольких методов которые можно вызвать из самого виджета:

`_delay()` – данная функция работает как «`setTimeout()`», вот только контекст переданной функции будет указывать на сам виджет (это чтобы не заморачиваться с областью видимости)

`_hoverable()` и `_focusable()` – данным методам необходимо скармливать элементы для которых необходимо отслеживать события «hover» и «focus», чтобы автоматически добавит к ним классы «ui-state-hover» и «ui-state-focus» при наступлении оных

`_hide()` и `_show()` – эти два метода появились в версии 1.9.0, они созданы дабы стандартизировать поведение виджетов при использовании методов анимации, настройки принято прятать в опциях под ключами «hide» и «show» соответственно. Использовать методы следует следующим образом:

```
options: {
	hide: {
		effect: "slideDown",  // настройки эквиваленты вызову
		duration: 500         // .slideDown( 500)
	}
}
// внутри виджета следует использовать вызовы _hide() и _show()
this._hide( this.element, this.options.hide, function() {
	// это наша функция обратного вызова
	console.log('спрятали');
});
```

Существует ещё пару методов, которые реализованы за нас:

```
enable: function() {
	return this._setOption( "disabled", false );
},
	disable: function() {
	return this._setOption( "disabled", true );
},
```

Фактически, данный функции создают синоним для вызова:

```
$("#my").expose({ "disabled": true }) // или false
```

Наша задача сводится лишь к отслеживанию данного флага в методе «`_setOption()`».

Примеру быть – widget.html, возможно этот виджет и не будет популярен, зато он наглядно демонстрирует как создавать виджеты для jQuery UI.

> *Будьте внимательны, с выходом jQuery UI версии 1.9.0 были внесены правки в Widget API, следовательно, большинство доступной информации устарело, так что читайте официальную документацию [http://wiki.jqueryui.com/w/page/12138135/Widget%20factory], а ещё лучше – заглядывайте в код готовых виджетов «от производителя»*

Информация по теме разработки виджетов:

- «The jQuery UI Widget Factory. WAT?» – эта документации актуальна [http://ajpiano.com/widgetfactory/]
- «Understanding jQuery UI widgets: A tutorial» [http://bililite.com/blog/understanding-jquery-ui-widgets-a-tutorial/]
- «Tips for Developing jQuery UI 1.8 Widgets» [http://www.erichynds.com/jquery/tips-for-developing-jquery-ui-widgets/]
- «Coding your First jQuery UI Plugin» [http://net.tutsplus.com/tutorials/javascript-ajax/coding-your-first-jquery-ui-plugin/]

jQuery Tools

> *Данный набор утилит достаточно долго не обновлялся, аж с марта 2012-го года, но он ещё не канул в лету, работа идёт, и обещают выпустить вторую версию.*

Альтернатива jQuery UI, хотя я бы назвал полезным дополнением. Библиотека jQuery Tools состоит из компонентов, которые разделяются на три типа:

- UI Tools – интерактивные компоненты интерфейса
- Form Tools – компоненты по работе с формами
- Toolbox – всяко-разно и полезно

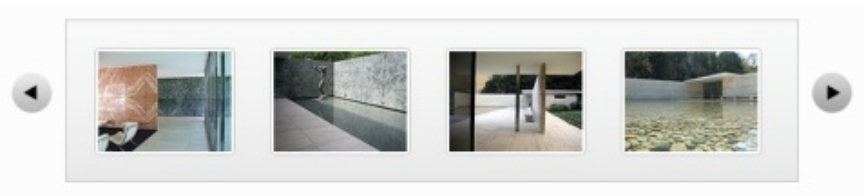
UI Tools

Tabs – аналогично jQuery UI по назначению, скромнее по функционалу, плюс еще есть возможность настраивать как «accordion»:

Tooltip – всплывающие подсказки, я не уверен в юзабельности оных, но может для кого-то данный компонент окажется полезным, ведь его так же включили в последний jQuery UI:

Overlay – создание всплывающих модальных окошек, полезный компонент, аля «lightbox»:

Scrollable – компонент для создания «карусели» из картинок, может пригодиться в любом онлайн-магазине:



Form Tools

Validator – позволяет на декларативном уровне задавать правила проверки вводимых значений, что бывает очень полезно, и крайне удобно (совместим с HTML5).

RangeInput – аналогичен jQuery UI Slider

DateInput – аналогичен jQuery UI Datepicker

Toolbox

Expose – пожалуй самый замечательный компонент из всего набора – он позволяет выделять элемент на странице, путём затемнение остальных элементов; это трудно описать, лучше посмотрите на демо:

http://jquerytools.github.io/demos/toolbox/expose/index.html

На этом, пожалуй, стоит закончить этот обзор.

jQWidgets

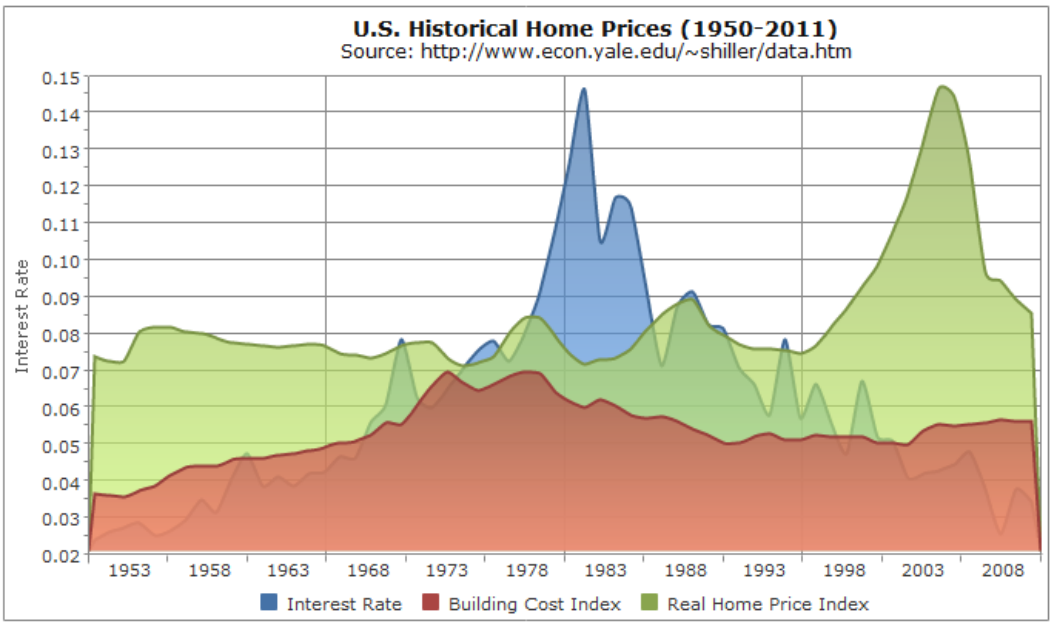
jQWidgets – это не просто библиотека, а целый комбайн с кучей полезных виджетов, да их на данный момент насчитывается тридцать четыре штуки, тут только привести список и скриншот на каждый – уже пол книги будет, так что обойдёмся кратеньким резюме на каждый виджет. Начну обзор с комплексных и «тяжёлых» виджетов, к ним картинки я таки приложу.

jqxGrid – это датагрид с кучей полезных и не очень примочек, тут понятное дело есть постраничная навигация, сортировки, фильтрации и группировки, только по этому виджету уже можно здоровенный талмуд написать:

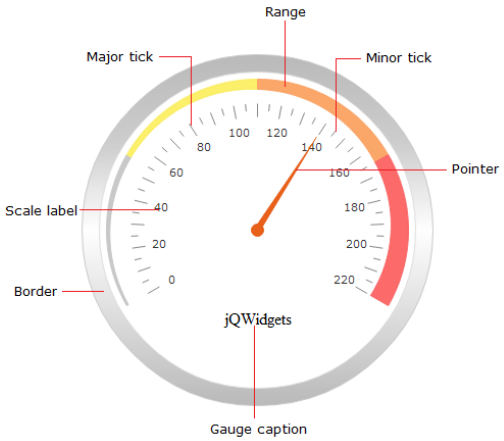
Product Name	Quantity per Unit	Unit Price	Units In Stock	Discontinued
Chai	10 boxes x 20 bags	$18.00	39	☐
Chang	24 - 12 oz bottles	$19.00	17	☐
Aniseed Syrup	12 - 550 ml bottles	$10.00	13	☐
Chef Anton's Cajun Seasoning	48 - 6 oz jars	$22.00	53	☐
Chef Anton's Gumbo Mix	36 boxes	$21.35	0	☑
Grandma's Boysenberry Spread	12 - 8 oz jars	$25.00	120	☐
Uncle Bob's Organic Dried Pears	12 - 1 lb pkgs.	$30.00	15	☐
Northwoods Cranberry Sauce	12 - 12 oz jars	$40.00	6	☐
Mishi Kobe Niku	18 - 500 g pkgs.	$97.00	29	☑
Ikura	12 - 200 ml jars	$31.00	31	☐

Go to: 0 Show rows: 10 1-10 of 77

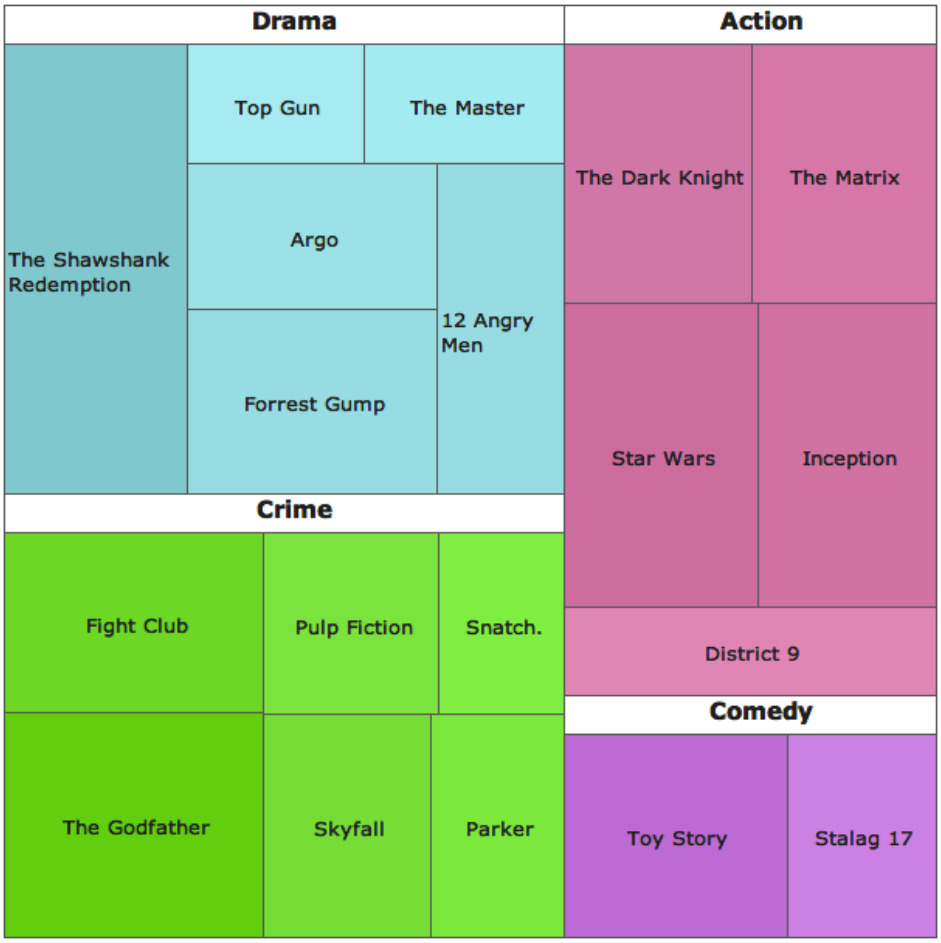
jqxChart – виджет для построения разнообразных графиков с помощью HTML, CSS и JavaScript, сделано всё очень и очень культурно, особенно выделю функцию по сохранению графика как картинки, иногда очень её не хватает:



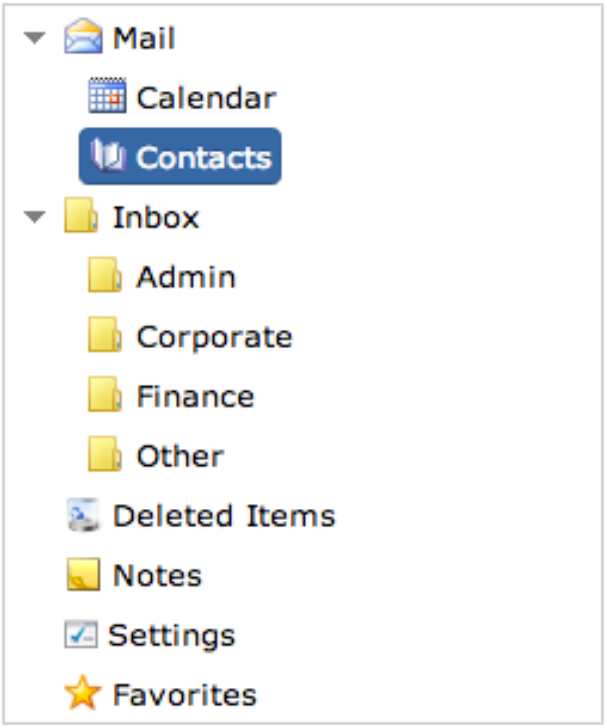
jqxGauge – этот виджет не часто встретишь и в даже более именитых фреймворках, но по сути – это некий измеритель, т.е. с его помощью можно нарисовать спидометр, манометр, термометр или ещё какой другой измерительный прибор с произвольной шкалой:

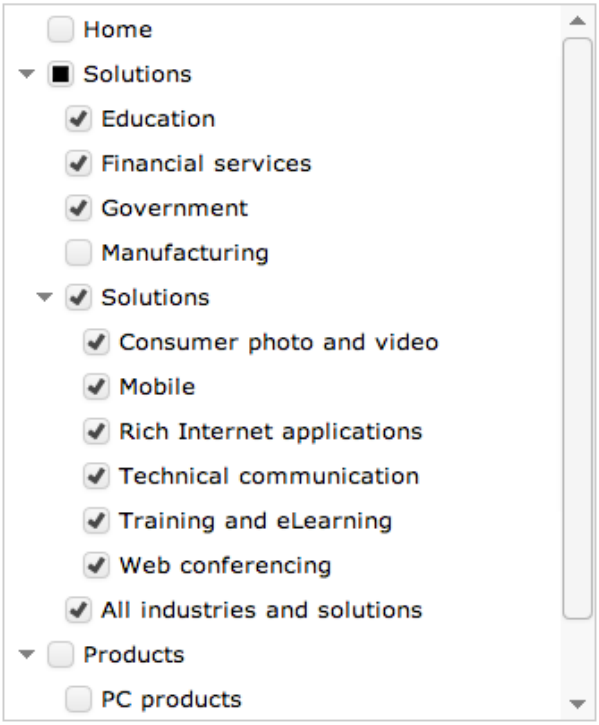


jqxTreeMap – ещё один редкий вид, скорее даже уникальный, с его помощью можно построить связанное дерево в виде организованных прямоугольничков, если ничего не понятно, то лучше просмотреть демку, ну и скриншот прилагаю:



jqxTree – это уже не столь экзотический виджет, как понятно из названия, будем садить деревья:





На этом обзор «крутых» виджетов можно заканчивать, углубляться в скучные и обыденный обёртки над элементами форм мне не хочется, замечу лишь, что во многом данный фреймворк обходит jQuery UI, но не всё так радужно в этом королевстве:

> *Данный фреймворк распространяется под лицензией Creative Commons Attribution-NonCommercial 3.0 License, которая предусматри-вает бесплатное использование библиотеки для не коммерческих проектов, иначе – смотрите расценки.*

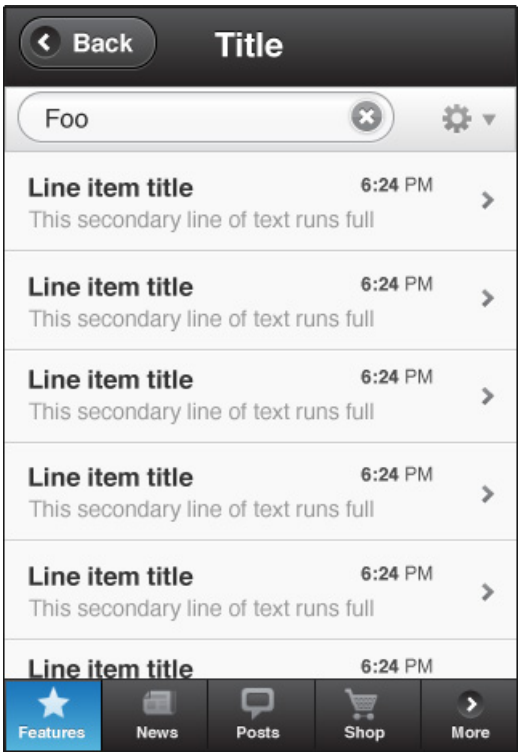
Ещё стоит упомянуть одну приятную особенность – это возможность легкой интеграции с MVVM фреймворком Knockout, но это уже другая история

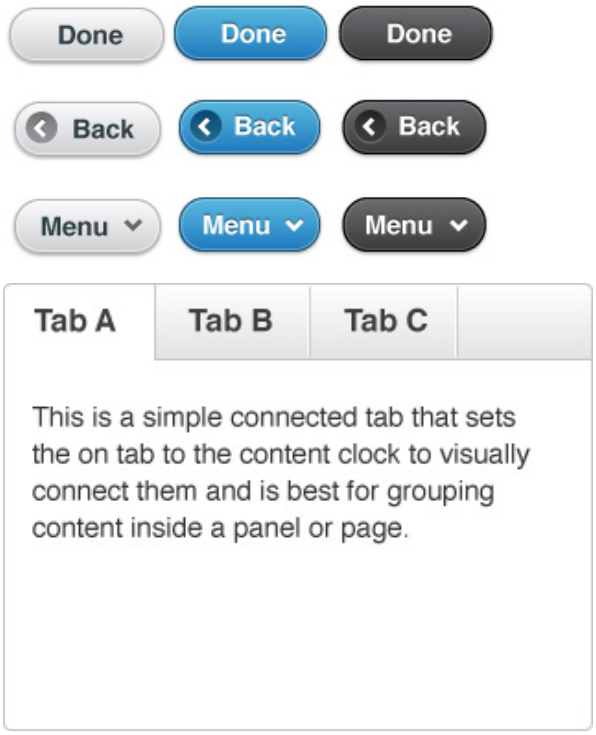
jQuery Mobile

А вот это вполне самостоятельный продукт, и как следует из названия предназначен для создания интерфейсов для мобильных устройств с поддержкой «Touch Screen». Этот фреймворк хорошо документирован, с кучей примеров, которые можно пощупать на http://jquerymobile.com/

> *Данный фреймворк хорошо подходит для создания мобильных версий сайтов, но при этом он будет выглядеть как мобильное приложение, хотя нет – сайт будет выглядеть как сайт на jQuery Mobile. Насколько это хорошо или плохо мне судить сложно, скажем так – это востребовано.*

Я не буду рассматривать все компоненты данного фреймворка, приведу лишь некоторые скриншоты (взяты с официального сайта http://jquerymobile.com/):





> *Для jQMobile существует свой собственный ThemeRoller:*
>
> *http://jquerymobile.com/themeroller/*

И еще, посоветую обратить внимание на API данного фреймворка, без изучения оного вам будет затруднительно создать действительно интересные приложения.

Еще плагины

Хотелось бы порекомендовать еще несколько плагинов, которые стоит всегда держать под рукой:

color — если потребуется анимация цвета фона, либо шрифта, либо еще чего-нибудь (как альтернатива jQuery UI) [https://github.com/jquery/jquery-color]

cookie — удобная работа с «печеньками» в браузере [https://plugins.jquery.com/cookie/]

easing — расширяем стандартный набор функций easing (об этом я рассказывал в главе анимация) [http://gsgd.co.uk/sandbox/jquery/easing/]

form — упрощает работу с формами, сам уже давно не пользуюсь, но для быстрого старта самое оно [http://malsup.com/jquery/form/]

hotkeys — название говорит само за себя [https://github.com/jeresig/jquery.hotkeys]

shadow animation — как следует из названия – плагин для анимации теней [http://www.bitstorm.org/jquery/shadow-animation/]

jQuery Transition Events – поддержка CSS Transition из JS [https://github.com/ai/transition-events]

Redactor – просто отличный WYSIWYG редактор, лёгкий, быстрый, и не бесплатный [http://redactorjs.com/]

Обновление на версию 2.x

Чем же нам грозит новая версия? Ну нам то боятся её не стоит, а вот пользователи старых версий Internet Explorer не смогут больше насладиться удобством столь популярной библиотеки. Да, да, именно так, из второй версии убрали поддержку IE с 6-ой по 8-ую. Благодаря этой «оптимизации» библиотека похудела на 10%.

> *Если более современные IE работают в режиме совместимости со старыми версиями, то jQuery не будет заморачиваться, и откажется работать. Избежать это поможет мета-тег X-UA-Compatible или заголовок HTTP.*
>
> *В скором времени «под нож» пойдут и другие устаревающие браузеры, будьте бдительны Android/WebKit 2.x, за вами уже выехали.*

Если вас не зацепила эта диета, то можно пойти дальше, и собрать jQuery только из того, что вам действительно необходимо, что для этого потребуется – читайте в подробном пошаговом руководстве, которому и перевод-то не требуется.

Если вас не впечатляет этот список измений, то это лишь благодаря труду разработчиков jQuery, которые постарались обеспечить полную совместимость API веток 1.x и 2.x, а вот это уже действительно впечатляет.

> *Ну ладно, не всё так гладко с совместимостью, был этот путь тернист, и проблемы с обновлением будут, вот только искать причину нужно будет в списке изменений версии 1.9, именно эта версия стала поворотной в вопросе совместимости веток.*

- «How to build your own jQuery» [https://github.com/jquery/jquery/]
- «jQuery Core 1.9 Upgrade Guide» [http://jquery.com/upgrade-guide/1.9/]

Обновление на версию 3.x

В первую очередь данное обновление коснулось старых версий браузеров, точнее не коснулось, теперь поддерживаются следующие версии:

- Internet Explorer — поддерживается с 9ой
- Chrome, Edge, Firefox и Safari — благодаря встроенной системе обновлений в эти браузеры поддерживается текущая и предыдущая версии браузеров
- Opera — в фаворитах не ходит, поэтому поддерживают только текущую версию
- Safari Mobile iOS — с 7ой и старше
- Android — с 4ой и старше

Кроме этого, было внесено много изменений, которые ломают обратную совместимость с предыдущими версиями, все их перечеслять не буду, остановлюсь на тех, с которыми уже пришлось столкнуться.

> *Большинство приведенных изменений касается не только версии 3.x, но старых веток, начиная с версий 1.12.x и 2.2.x*

AJAX

— Метод «`jQuery.ajax()`» теперь совместим с Promise, и вы можете использовать методы «`.then()`» и «`.catch()`»:

```
$.ajax({url: "/get-my-page.html" /* и т.д. */ })
 .then(function() { /* всё ОК */ })
 .catch(function() { /* ошибка */ })
;
```

— Добавлена новая сигнатура для вызова двух AJAX-методов «`$.get(settings)`» и «`$.post(settings)`», теперь настройки совместимы с «`$.ajax(settings)`».

— При подключении скриптов с другого домена теперь в обязательном порядке требует указания «`dataType: "script"`»

— При выполнении AJAX-запросов и указании URL с хэшем, последний более не обрезается, а отправляется на сервер как есть.

Атрибуты

— Раньше метод «`.removeAttr()`» для true-false атрибутов, таких как «`checked`», «`selected`» и «`readonly`», втихую выставлял соответсвующее свойство DOM элемента в «`false`», теперь извольте делать сие ручками:

```
$("input[type=email]").removeAttr("readonly").prop("readonly", false)
```

— Если вызовите метод «`.val()`» для мультиселекта, в котором ничего не выбрано, то получите в ответ пустой массив, а раньше был «`null`»
— Для SVG заработали методы по манипуляции с классами (хотя полноценной поддержки SVG в jQuery нет).

Core

— Ядро jQuery теперь запускается в strict mode.
— Обработчики события document-ready теперь запускаются асинхронно, т.е. теперь если какой-то обработчик сфейлил, то это не повлияет на запуск других обработчиков.
— Метод «`$.isNumeric()`» более не пробует кастовать метод «`.toString()`» для произвольных объектов (кому это надо было?).
— Методы «`.width()`», «`.height()`» и так далее, раньше вызвав их для пустой коллекции объектов мы получали «`null`», теперь «`undefined`».
— Официально добавлен promise «`jQuery.ready`», который очень удобно заюзать вместе с «`$.when()`»:

```
$.when($.ready, $.getScript("script.js") ).then(function() {
      // документ готов, и скрипт script.js подгружен
}).catch( function() {
      // ошибка
});
```

— Метод «`jQuery.unique()`» переименован в «`jQuery.uniqueSort()`»
— Метод «`jQuery.parseJSON()`» устарел, переходите на «`JSON.parse()`»

— Внутреннее хранилище данных перешло на camelCase, это важно для тех, кто использовал его, не прибегая к методу «`.data()`»

Объект Deferred

Как я уже упоминал ранее, объект Deferred теперь совместим с Promise из ES-2015, а это нам грозит переписыванием методов «`.done()`» на «`.then()`» и «`.fail()`» на «`.catch()`».

Второй важный момент — callback-функции, согласно спецификации ES-2015, должны принимать только один аргумент: для успешного выполнения это некий результат выполнения, в случае возникновения ошибки функция это будет сама ошибка. Если у вас не получается так сделать, то старые функции «`.done()`» и «`.fail()`» всё ещё остаются с нами, хоть чую и их скоро выпилят:

```
// было
$.get("/get-my-page.html")
 .done(function(data, textStatus, jqXHR) { /* всё ОК */ })
 .fail(function(jqXHR, textStatus, errorThrown) { /* ошибка */ });

// стало
$.get("/get-my-page.html")
 .then(function(data) { /* всё ОК */ })
 .catch(function(error) { /* ошибка */ });
```

Размеры

Небольшие изменения постигли функции «`.width()`», «`.height()`», «`.css("width")`», и «`.css("height")`», теперь они могут возвращать не только integer значение высоты и ширины, но и float, такая точность связана с переходом на использование «`getBoundingClientRect()`».

Ещё момент, вызов «`$(window).outerWidth()`» и «`$(window).outerHeight()`» теперь будут включать в себя размеры скролбаров окна.

Эффекты

Анимация переехала на использование requestAnimationFrame API, так что теперь всё стало быстрее и красивее.

Функции «.show()», «.hide()» и «.toggle()» научились запоминать предыдущее состояние CSS-свойства «display».

Количество аргументов easing-функций сократили до одного аргумента, который содержит прогресс анимации от 0 до 1.

События

Удалены shorthand-методы для следующих событий: «.load()», «.unload()» и «.error()», связано данное изменение с конфликтами возникающие при использовании данных методов, так что переписывайте на «.on()»:

```
// было
$("img").load(fn)
// стало
$("img").on("load", fn).
```

Удалено синтетическое событие «ready», так что «.on("ready", fn)» более не работает, используйте синтаксис «$(fn)».

Делегированные события, в случае если их пытаются повесить с использованием неправильных селекторов теперь будут сразу ругаться и выбрасывать ошибку. Дебажить станет легче:

```
// пример сломанного селектора div:not
$("body").on("click", "div:not", e => false);
```

Селекторы

За селекторы «:hidden» и «:visible» теперь отвечает «getClientRects()», если у запрашиваемого элемента есть layout box, значит он считается видимым, как результат пустой или
 теперь считаются видимыми.

Кривые селекторы «$("#")» и «.find("#")» теперь будут вызывать ошибку.

Описал многое, но не всё, полное руководство доступно на официальном сайте:

– «jQuery Core 3.0 Upgrade Guide»
  [https://jquery.com/upgrade-guide/3.0/]

История изменений

1.0.0 alpha	Октябрь, 2012	Первая публикация «для своих», всего 95 скачиваний, вычитка началась
1.0.0 beta		Исправление опечаток и грамматических ошибок. Публикация на Хабре и более 22 000 скачиваний
1.0.0		Релиз :) Исправлено множество опечаток и несколько ссылок. Добавлена информация о быстродействии селекторов вида «`div#id`», спасибо хабра-пользователю skorney за тестирование. Более 2 000 скачиваний
1.0.1	Ноябрь, 2012	Ещё несколько мелких правок. Спасибо читателям за отзывы. 23 000+
1.0.2	Январь, 2013	И ещё… Опубликовал в ePub формате, выложил на scribd, получилось более 10 000 скачиваний и более 10 000 прочтений на scribd
1.0.3	Сентябрь, 2013	Обновил книгу вслед за обновлением jQuery, добавил информацию о изменениях в версии 2.x относительно 1.x. Добавил описание виджетов jQWidgets. Удалил описание событий для метода «`data()`». Примеры кода используемые в учебнике теперь доступны на GitHub, так что жду pull-запросов: https://github.com/AntonShevchuk/jquery-for-beginners. Счётчик прочтений на scribd перевалил за 20 тысяч
1.0.4	Май, 2014	Оптимизация шрифтов под печать формата A5. Тираж!
1.0.5	Март, 2015	Мелкие исправления и уточнения. ~22 000 скачиваний
1.0.6	Сентябрь, 2016	Обновлена документация с учётом выхода jQuery 3.x. Добавлена информация по миграции на jQuery 3.x. Добавлена информация о нововведениях ES-2015.

Благодарности

Спасибо моей компании NIX Solutions Ltd за моральную и материальную поддержку. Спасибо моим коллегам, которым первым пришлось читать эту книгу и вносить корректировки.

Хочу еще сказать спасибо Илье Кантору за его отличные мастер-классы по JavaScript'у (это не реклама, они действительно хороши), которые помогли мне в лучшей степени познать мир этого замечательного языка.

Огромное спасибо моей супруге Оле и сыну Данилу за терпение, что его хватило не выгонять меня из-за компьютера и из дому.