Антон Шевчук

*jQuery*

*посібник для початківців*

1.0.6

2016

**Про автора**

Антон Шевчук народився в 1983 році в невеликому містечку Охтирка, Україна, батько - інженер-конструктор, мати - лікар-терапевт. У 2000-му році закінчив загальноосвітню школу №2. У тому ж році вступив до Харківський Національний Університет Радіоелектроніки на факультет КВУ на спеціальність системне програмування. Під час навчання проживав у гуртожитку №8 і був там відомий під ніком «Dark Lord». Після отримання диплома в 2005-му, залишився в Харкові і влаштувався на роботу в компанію NIX Solutions Ltd.

Веде блог свого імені [http://anton.shevchuk.name.](http://anton.shevchuk.name/) На даний момент, продовжує працювати в компанії NIX Solutions Ltd на посаді технічного експерта відділу PHP. Знає PHP, захоплюється JavaScript, завів собі open-source проекты. Є одним з організаторів конференцій ThinkPHP в Харкові.

**Про книгу**

Даний підручник підійде як для початківців web-розробників, так і для просунутих JavaScript програмістів, бажаючих освоїти нову для себе бібліотеку.

У підручнику прийнято наступне форматування:

**важливо** - запам'ятай, запиши, і завжди застосовуй

<html> — вихідний код в тексті виділяється monotype шрифтом

*// лістинг вихідного коду буде набрано monotype шрифтом з відступом*

*// також буде використовуватися елементарне підсвічування синтаксису*

function () {

setTimeOut("alert('Hello World!')", 5000);

}

*інформація, яку варто взяти до відома, або, можливо, лише прочитати і посміхнутися,*

*буде оформлена в блок тексту з відступом і виділена курсивом*

Приклади коду ви зможете знайти в репозиторії автора на GitHub'е — https://github.com/AntonShevchuk/jquery-for-beginners, вживу — на персональному блозі <http://anton.shevchuk.name/book/code/>

**Умови розповсюдження**

* Даная книга розповсюджується безкоштовно і доступна для завантаження на сторінці <http://anton.shevchuk.name/jquery-book/>
* Якщо хочете поділитися книгою з другом – надішліть йому посилання на сторінку [http://anton.shevchuk.name/jquery-book/,](http://anton.shevchuk.name/jquery-book/) будь-які інші способи «поділитися» не бажані
* За повне або часткове копіювання матеріалів без узгодження з автором Вам буде соромно

**Зміст**

[Про автора 2](#_Toc27561800)

[Про книгу 3](#_Toc27561801)

[Умови розповсюдження 4](#_Toc27561802)

[0% Про HTML, CSS і JavaScript 6](#_Toc27561803)

[10% Підключаємо,знаходимо, готуємо 36](#_Toc27561821)

[20% Атрибути елементів і CSS 50](#_Toc27561828)

[30% Події 54](#_Toc27561829)

[40% Анімація 63](#_Toc27561835)

[50% Маніпуляції с DOM 71](#_Toc27561839)

[60% Работа з формами 78](#_Toc27561840)

[70% AJAX 83](#_Toc27561841)

[80% Об'єкт Deferred і побратими 99](#_Toc27561842)

[90% Пишемо свій плагін 107](#_Toc27561843)

[100% Останній розліл 125](#_Toc27561844)

[Доповнення 126](#_Toc27561845)

[Історія змін 152](#_Toc27561846)

[Подяки 153](#_Toc27561847)

**0% Про HTML, CSS і JavaScript**

Почнемо знайомство з jQuery з повторення (або вивчення) основ правильного вживання зв'язків HTML і CSS з невеликою домішкою JavaScript.

*Якщо не хочете впасти в бруд обличчям перед колегами – то не пропускайте цей розділ «повз вуха»*

HTML — про нього варто пам’ятати дві речі– **семантичний** та **правильний**.

**Семантична верстка**

Під семантичною версткою HTML документа мається на увазі використання тегів за прямим призначенням, тобто якщо вам потрібний заголовок то в цьому вам допоможе тег <h1> і його побратими, необхідне табличне представлення даних – використовують тег <table> і тільки його

*Іноді, позбавляючись від табличної верстки, доходить до абсурду, і тег*

*<table> стає ізгоєм, і його перестають використовувати навіть для табличного представлення даних, не варто повторювати цю помилку.*

Забігаючи наперед, варто згадати за теги специфікації HTML5:

<article>, <aside>, <header>, <footer>, <menu>, <section> и т.д. — не бійтесь їх використовувати

*Не боятися — це правильно, але використовувати теж треба з розумом, рекомендую ресурс* [*http://html5doctor.com/*](http://html5doctor.com/) *— дуже добре і детально розписано про нововведення специфікації HTML5.*

*І ще декілька цікавих ресурсів в навантаження:*

* + [*http://htmlbook.ru/html5*](http://htmlbook.ru/html5) *— не погано, але російською*
  + [*http://www.html5rocks.com/en/*](http://www.html5rocks.com/en/) *— тут ціле співтовариство*

Намагайтесь уникати надлишкових елементів на сторінці, більшість HTML сторінок грішать зайвими блоковими елементами:

<div id="header">

<div id="logo">

<h1><a href="/">Мій блог</a></h1>

</div>

<div id="description">

<h2>Тут я ділюсь своїми думками</h2>

</div>

</div>

Дану конструкцію можна легко спростити, і при цьому код стане більш читабельним, зміни в CSS будуть мінімальними (або навіть не будуть потрібні):

<header>

<h1>

</h1>

<a href="/">Мій блог</a>

<h2>Тут я ділюсь своїми думками</h2>

</header>

*В англ. мові є термін «divits» – цим терміном нагороджують HTML- розмітку з надмірним використанням div’ів без потреби, я ж називаю такі творіння «дивними». Зазвичай таким грішать новачки,які для застосування стилів CSS обертають елементи в div’и, что и приводит к их размножению без нужды.*

Ще одним обов’язковим пунктом для створення «правильного» HTML є використання назв класів та індентифікаторів, які однозначно говорять нам про вміст елемента, а не про будь-яких нюансах оформлення, наведу приклад :

|  |  |
| --- | --- |
| **Погано** | |
| red, green і т.д. | В якийсь момент захочете перефарбувати, ы елемент з класом «red» буде синього кольору |
| wide, small и т.д. | Сьогодні широкий, а завтра? |
| h90w490 | Напевно це елемент з висотою 90px і шириною 490px, чи я помиляюсь? |
| b\_1, ax\_9 | Ці назви також ні про що не свідчать |

|  |  |
| --- | --- |
| color1,color2и т.д. | Іноді зустрічається для «скінованних» сайтів, але створюють такі класи з ліні |
| element1...20 | Таке також трапляється, але ні про що хороше не свідчить |

Ну і приклади правильного іменування:

|  |  |
| --- | --- |
| **Добре** | |
| logo, content | логотип, основний контент |
| menu, submenu | меню і підменю |
| even, odd | Парні і непарні елементи списку |
| paginator | Посторінкова навігація |
| copyright | Копірайт |

*Є ще один момент - це форматування HTML і CSS коду, я не буду загострювати на ньому увагу, але весь код в книзі буде відформатований відступами, і, можливо, це дасть свої плоди в ваших витворах.*

**Валідний HTML**

Зелений маркер W3C validator'а – це правильно, і до цього потрібно прагнути, так що не забувайте закривати теги та прописувати обов’язкові параметри, наведу приклад HTML коду, в якому допущено 6 помилок (згідно специфікації HTML5), знайдіть їх:

<!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>

*Якщо сумніваєтесь у своїх знаннях, то перевірте даний код на сторінці https://validator.w3.org/nu/#textarea*

**CSS-правила і селектори**

Тепер приступимо до CSS, і почнемо, мабуть, з розшифровки абревіатури CSS це Cascading Style Sheets, дослівно «каскадна таблиця стилів», але:

—Чому ж вона називається каскадною? - це питання я часто задаю на співбесідах претендентам. Відповіддю ж буде аналогія, бо вона непорушна як перпендикулярна жаба: уявіть собі каскад водоспаду, ось ви стоїте на одній із сходинок каскаду з чорнильницею в руках, і виливаєте її вміст в воду - вся вода нижче по каскаду буде фарбуватися в колір чорнила. Застосувавши цю аналогію на HTML -створюючи правило для елемента, ви автоматично використовуєте його на всі дочірні елементи (звичайно, не всі правила так працюють, але про них пізніше) - це успадкування стилів. Тепер, якщо таких розумників з чорнильницею більше ніж один, і кольори різні, то в результаті отримаємо змішання кольорі, але це в життя, а в CSS працюють правила пріоритетів, якщо коротко і по справі:

* Самий низький пріорітет мають стилі браузера за замовчуванням – в різних браузерах вони можуть відрізнятися, тому придумали CSS Reset (Гугл ), і всі будуть на рівних;
* Трохи важливіше – стилі задані користувачем в корінні налаштувань браузера, в житті зустрічаються рідко
* найважливіші – стилі автора сторінки, але і там все йде по порядку
* найнижчий пріоритет у тих, що розташовані в зовнішньому підключеному файлі
* потім ті, що розміщенні всередині HTML з допомогою тега <style>
* потім ті, що захардкодили погані люди (не ви, ви так робити не будете) в атрибуті «style»
* найвищий пріорітет у правил з міткою «!important»
* за однакової кількості пріорітетів, тапки у того, хто оголошений останнім

Якщо голова ще не бо-бо, то я також згадаю, що при розрахунку, чиї ж правила головніше, ще аналізується специфіка селектор, і тут вважається таким чином:

розрахунок відбувається за чотирма ваговими позиціями [0: 0: 0: 0]

стилі задані в атрибуті « style =" " » мають найбільший пріоритет і отримують одиницю по першій позиції - [1: 0: 0: 0]

за кожен ідентифікатор елемента - [0: 1: 0: 0] (#id)

за кожен клас, або псевдо клас - [0: 0: 1: 0] (.my, : pseudo)

за кожен тег - [0: 0: 0: 1] (div, a)

при цьому [1: 0: 0: 0]> [0: x: y: z]> [0: 0: x: y]> [0: 0: 0: x].

Приклад селекторів, збудованих за пріоритетом (перші важливіше):

#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 ілюстрації специфічності попереднього прикладу (див. css.priority.html):

<div class="wrapper">

<div id="my" class="content">

<p id="id">

Lorem ipsum dolor sit amet, consectetuer...

</p>

</div>

</div>

У разі рівного розподілу рахунку – останній головний.

*Кажуть, що правило з 255 класами буде вище за пріоритетом, ніж правило з одним « id », але я сподіваюся, такого коду в реальності не існує.*

Ось таке коротке вступне слово, але пора повернутися до jQuery. Так ось, працюючи з jQuery, ви повинні «на відмінно» читати правила CSS, а також вміти складати CSS- селектори для пошуку необхідних елементів на сторінці. Але давайте про все по порядку, візьмемо наступний простенький приклад цілком семантичного 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"](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>

Це приклад простого і правильного HTML5 з невеликим додаванням CSS3. Давайте розберемо селектори в наведеному CSS-коді (я навмисне не виносив CSS в окремий файл, бо так наглядніше:

Body — дані правила будуть застосовані до тегу <body> і всім його нащадкам, запам'ятайте: настройки шрифтів поширюються вниз «по каскаду»

h1,h2 — ми вибираємо теги <h1> і <h2> , і встановлюємо колір шрифту для даних тегів і їх нащадків

#content— вибираємо елемент з « id =" content " », настройки відступів не поширюються на нащадків, вони будуть змінюватися тольки для даного елемента

.box - вибираємо елементи з « class =" box " », і змінюємо зовнішній вигляд кордонів елементів із заданим класом

Тепер докладніше із ускладненими прикладами:

|  |  |
| --- | --- |
| h1 | шукаємо елементи на ім'я тега |
| #container | шукаємо елемент по ідентифікатору **« id = container »** ( ідентифікатор унікальний , значить, на сторінці він повинен бути тільки один) |
| div # container | шукаємо <div> c ідентифікатором container, але попередній селектор працює швидше, але цей важливіше |
| .news | вибираємо елементи на ім'я класу « class =" news " » |
| div.news | всі елементи <div> c класом « news » (так працює швидше в IE8, тому що в ньому не реалізований метод « getElementsByClassName () ») |
| #wrap .post | шукаємо всі елементи з класом « post » всередині елемента з « id = wrap » |
| .cls1.cls2 | вибираємо елементи з двома класами « class =" cls1 cls2 " » |
| h1, h2, .posts | перерахування селектор, виберемо все перераховане |
| .post> h2 | вибираємо елементи <h2> , які є безпосередніми нащадками елемента з класом « post » |
| a + span | будуть обрані всі елементи <span> наступні відразу за елементом  <a> |
| a[href^=http] | Будуть вибрані всі елементи <a> у яких «href» починається з http |

*Це аж ніяк не весь список, опис же всіх CSS3 селектор можна знайти на відповідній сторінці W3C:* [*http://www.w3.org/TR/css3-selectors/*](http://www.w3.org/TR/css3-selectors/)

*40% завдань, які ви будете вирішувати за допомогою jQuery, зводяться до пошуку необхідного елемента на сторінці, так що****знання CSS селекторів обов'язково****. Ось ще шматочок CSS для тренування, напишіть відповідний йому HTML (це теж питання з співбесіди;):*

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

*Пишіть прям тут:*

**CSS. Занурення**

*Цей розділ буде корисний починаючим верстальщикам, і тим, хто захоче зробити трохи більше, ніж висновок вікна повідомлення при натисканні.*

**Про форматуванні**

Ні, я не владний над собою, і таки наведу як приклад CSS форматування, яке я використовую:

*/\*header\*/*

header {

margin-bottom: 16px; font-weight: 400;

}

header h1 {

color: #999;

}

header p {

font-size: 1.4em; margin-top: 0;

}

*/\*/header\*/*

Чому це добре:

- такий CSS легко читається

- є ідентифікатор початку і кінця блоку (можна швидко знайти необхідну частину навіть в дуже великому CSS файлі використовуючи пошук по мітці « \* header»)

- подібне форматування явно вказує на вкладеність елементів

- і можна легко простежити успадкування властивостей

Я не наполягаю на своєму варіанті, але хотілося, щоб ви взяли на озброєння один з багатьох стандартів форматування, і завжди слідували йому.

*Коли станете запеклими front-end розробниками, то пізнаєте всю силу CSS-препроцесорів, а поки слухайте і запам'ятовуйте..*

**Іменування класів і ідентифікаторів**

Я вже торкався цієї теми, коли розповідав про релевантності HTML, так ось - імена класів можуть бути навіть такими: « b-service-list column b-service- list column\_right » і це буде круто, і «must be» - але лише в рамках дійсно великих проектів, і власне чого я розпинаюсь, задам вихідну точку для вивчення - інформації там ще на одну книгу;):

- «Що таке БЕМ»— [https://ru.bem.info/methodology/]

*Обов'язково ознайомтеся з принципами БЕМ - це корисно для розширення кругозору, і прокачування скилов*

**Про кольори**

У WEB використовується колірна модель RGB , я не відкрию вам Америку, якщо розповім, що червоний колір можна записати не тільки як « red », а й ще декількома способами:

p { color: red }

p { color: #ff0000 }

p { color: #f00 } */\* скоросений запис, економить 3 байта \*/*

p { color: rgb(255, 0, 0) }

*Тепер ви без запинки повинні назвати кольори # f00 , # 0f0 , # 00f , а ті, у кого з малювання було «відмінно», назвуть і # ff0 , # 0ff і # f0f ;)*

З появою CSS3, вказуючи колір, ми також можемо поставити значення α-каналу, тобто прозорість:

p { color : rgba ( 255 , 0 , 0 , 1 ) } */ \* звичайний текст \* /*

p { color : rgba ( 255 , 0 , 0 , 0.5 )} */ \* напівпрозорий текст \* /*

Ще одна примочка CSS3 - це можливість використання колірної моделі HSL (hue saturation lightness - тон, насиченість і світлин) і HSLA (HSL + α- канал):

p { color : hsl ( 0 , 100% , 50% )} */ \* червоний \* /*

p { color : hsl ( 120 , 100% , 50% )} */ \* зелений \* /*

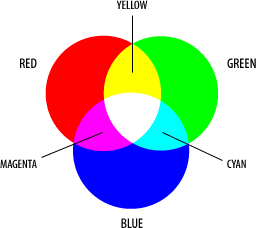
p { color : hsl ( 240 , 100% , 50% )} */ \* синій \* /*

p { color : hsla ( 0 , 100% , 50% , 0.5 )} */ \* напівпрозорий червоний \* /*

Для перекладу з HSL в RGB існує простий алгоритм, але поки не варто їм себе вантажити.

*Так хто цим HSL користується? Чи не мороичте собі голову!*

Для тих, кого питання змішання каналів RGB поставив в глухий кут, то ось вам наглядний посібник:



**Блокові і рядкові елементи**

*Знову я буду посилатися на чийсь підручник - на цей раз від Івана Сагалаєва -*[*http://softwaremaniacs.org/blog/category/primer/ ,*](https://translate.google.com/translate?hl=uk&prev=_t&sl=ru&tl=uk&u=http://softwaremaniacs.org/blog/category/primer/)*і нехай вас не бентежать дати написання статей - вони оповідають про основи і актуальність вони не втратять ще дуже довго .*

Можливо, ви ще не знаєте, але HTML теги діляться на блокові (block) і рядкові (inline). Блоковими елементами називають ті, які відображаються як прямокутник, вони займають всю доступну ширину і їх висота визначається вмістом. Блокові теги за замовчуванням починаються і закінчуються новим рядком - це <div> , <h1> і побратими, <p> і інші.

Якщо хочете, щоб ваш HTML залишався дійсним, стежте за тим, щоб блокові елементи не розташовувалися всередині строкових елементів. Усередині строкових тегів може бути або текст, або інші строкові елементи

*.*

*Одна з найбільш поширених помилок, це обертання заголовка на посилання: <a href="#"> <h1> Назва статті </ h1> </a> , не допускайте подібні промахи.*

*Хоча якщо ми орієнтуємося на HTML5 - то тег <a> тепер може бути блоковим елементом, і наведений приклад буде дійсним.*

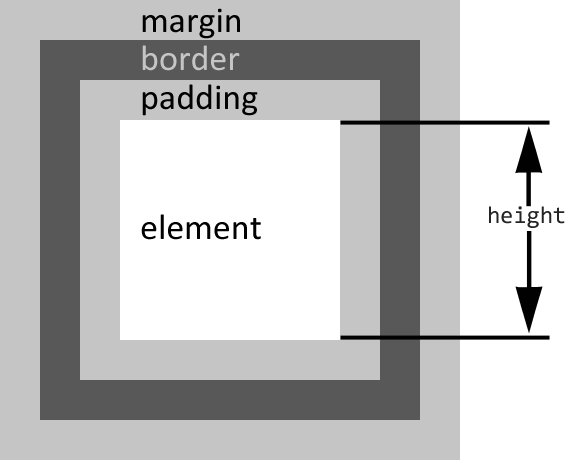
*Ага, ось такий я не послідовний .*

По темі:

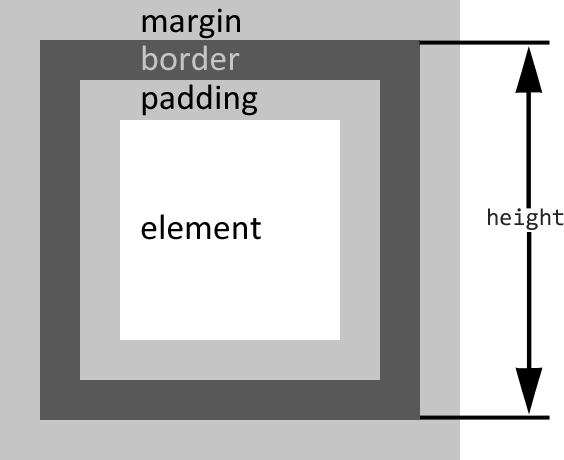
* «Inline Elements List and What’s New in HTML5» [<http://www.tutorialchip.com/tutorials/inline-elements-list-whats-new-> in-html5/]
* «HTML5 Block Level Elements: Complete List» [<http://www.tutorialchip.com/tutorials/html5-block-level-elements-> complete-list/]
* «Раскладка в CSS: поток» [<http://softwaremaniacs.org/blog/2005/08/27/css-layout-flow/>]

**Про розміри блокових елементів**

Ще хотів окремо зупинитися на обчисленні ширини і висоти блокових елементів, адже тут є один нюанс, за замовчуванням, висота і ширина елементів вважається без урахування товщини кордонів і внутрішніх відступів, тобто якось так:



Ця блокова модель називається « content-box », і ось в CSS3 з'явилася можливість змінювати блокову модель, вказуючи атрибут « box-sizing » . Дуже добре, тепер ми можемо вибирати між двома значеннями « content-box » і « border-box », перший я вже описав, а ось другий обчислює висоту і ширину включаючи внутрішні відступи і товщину кордонів:



*Така блокова модель була властива IE6 в «quirks mode»*

Корисні статті по темі:

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

**Плаваючі елементи**

Я б хотів ще розповісти про CSS властивості « float », але боюся, розповідь буде довгим і виснажливим, але якщо коротко: якщо ви вказуєте елементу властивість « Float », то:

* наш елемент буде зміщений по горизонталі, і «прилипне» до зазначеного краю батьківського елементу
* якщо це був блоковий елемент, то тепер він не буде займати всю ширину батьківського елемента, і звільнить місце
* якщо слідом йдуть блокові елементи, то вони займуть його місце
* - якщо слідом йдуть строкові елементи, то вони будуть обтікати наш елемент з вільною боку

Це поведінка «за замовчуванням», як це виглядає в живу можна подивитися на прикладі css.float.html . Тут головне треба розуміти те, що відбувається, і вміти керувати, якщо звичайно ви хочете хоч трохи навчитися верстати :)

Життєво необхідна інформація для верстальників:

- «Розкладка в CSS: float» [ [http://softwaremaniacs.org/blog/2005/12/01/css-layout-float/](https://translate.google.com/translate?hl=uk&prev=_t&sl=ru&tl=uk&u=http://softwaremaniacs.org/blog/2005/12/01/css-layout-float/) ]

**Позиціонування**

Дам лише вступну по « position » - у нього буває лише чотири значення:

static - стан справ «за замовчуванням», блоки лягають один за одним, зверху вниз, по порядку, без відхилень

absolute - блок позиціонується згідно із заданими координатами

fixed - схоже на « absolute », але різниця в тому, що блок не буде «скролитись»

relative - такий блок можна зрушувати щодо місця де він розташований, ну і всі внутрішні «абсолютні» блоки будуть використовувати даний елемент як точку відліку при позиціонуванні за координатами

Для самостійного вивчення:

- «Розкладка в CSS: позиціювання» [

- [<http://softwaremaniacs.org/blog/2005/08/03/css-layout-positioning/>]

**Розділяй і володарюй**

Тут варто запам’ятати декілька простих правил:

* виносимо JavaScript в зовнішні файли
* ніколи не пишемо inline обробників подій « onclick =" some () " »
* виносимо CSS з HTML в зовнішні файли
* ніколи не пишемо inline стилів « style =" color: red " »
* і ще разок для закріплення - не пишемо inline!

Тепер наведу для прикладу код, за який слід що-небуть ламати (це приклад поганого коду,уточнюю для тих хто не розуміє натяків):

<script>

function doSomething(){ */\* … \*/* }

*/\* лунає хрускіт зламаних кісток зап’ястя,щоб не друкував \*/*

</script>

<style>

p { line-height:20px; }

*/\* крхххх… гомілкова кістка, і на роботу вже не піде\*/*

</style>

<div style="color:red;font-size:1.2em">

<p onclick="doSomething();">Lorem ipsum dolor sit amet...</p>

*<!-- тидищ, головой об стіл… на смерть, як жест милосердя -->*

</div>

Неясно, чому ж це погано? Схоже, вам просто не доводилося міняти дизайн для вже готового сайту :) Проясню суть проблеми: вам ставлять завданя -

« Треба поміняти колір шрифту для всіх сторінок сайту », яких може бути три десятка. Це можуть бути НЕ тільки HTML-файли, а сторінки якогось

шаблонізатора

розкидані по двом десяткам папок (і це ще не найгірший варіант). І тут з'являється він - червоний абзац. імовірність почути

«Слова підтримки» на адресу автора цього коду буде прагнути до одиниці. Щодо inline-обробників подій ситуація схожа, ось уявіть собі - пишите ви JavaScript код, все відмінно, все виходить, поки ви не намагаєтеся клікнути по червоному абзацу, він виявляється вам НЕ підвладний, і живе своїм власним життям, ігноруючи всі ваші зусилля. Ви дивитесь код, і знову хтось почує ці слова ...

Застосувавши чотири правила «червоного абзацу» у вас повинен буде вийти чистий і передбачуваний HTML код:

<div id="abzac">

<p>Lorem ipsum dolor sit amet...</p>

</div>

Стилі можна буде легко повісити на <div> з ідентифікатором, як власне і обробник подій для нашого параграфа.

*Абзац - НЕ параграф, але для красного слівця, і легкості засвоєння згодиться*

**Трохи про JavaScript**

*В даний розділ я виніс ту інформацію про JavaScript, яку необхідно знати, щоб у вас не виникало «дитячих» проблем з використанням jQuery. Якщо у вас є досвід роботи з JavaScript - то перегортайте далі.*

*Хочете вивчати JavaScript і jQuery? Так силу познайте інструмента істинного:*

* + *Developer Tools для Chrome, Safari та інших webkit-based браузерів*
  + *FireBug для FireFox*
  + *DragonFly для Opera*
  + *Developer Tools для Microsoft Edge*

*Список не повний, но console там є, треба вміти його застосовувати*

**Про форматування**

Хотів би відразу звернути увагу на форматування JavaScript коду. Мій досвід мені підказує - найкраще спроектувати стандарти форматування основної мови розробки на прикладну - JavaScript, а якщо ви хочете чогось глобального і загальноприйнятого, то я для вас вже погуглив:

* «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>]

*На додачу поділюся невеликою порадою: всі змінні, що містять об'єкт jQuery, найкраще називати, починаючи з символу «$». Повірте, така невелика хитрість економить багато часу.*

*І ще - в кінці кожного рядка я ставлю крапку з комою, сам JavaScript цього не вимагає, а й зайвим не буде.*

**Основи JavaScript**

**Змінні**

Перше з чим зіткнемося - це оголошення змінних:

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

Все просто, оголошуємо змінну, використовуючи ключове слово « var ».

*Можна, звичайно ж, і без « var », але робити я вам це наполегливо не рекомендую, тому що можуть виникнути непередбачувані проблеми в коді про що трохи пізніше розповім.*

*Так, так, JavaScript відноситься до мов з динамічною типізацією, і нам немає потреби вказувати тип даних при оголошенні змінних, ви навіть можете влаштувати «holy war» з цього приводу, але робіть це локально, не виходячи за рамки своєї черепної коробки.*

На ім’я змінних наклаадено два:

* ім'я може складатися з букв, цифр, і символів « $ » і « \_ »
* перший символ не повинен бути цифрою

Врахуйте, регістр букв має значення:

var company = "Facebook";

*// зовсім інша «компанія»*

var Company = "Google";

Хочу також вас познайомити з таким нововведенням ECMAScript-2015 (надалі ES-2015) як оголошення змінних з використанням конструкції « let »:

let company = "Facebook";

Зовні, від « var » не сильно відрізняється, зате яка різниця в поведінці:

- область видимості змінної « let » обмежена блоком {...} , на відміну від « var », яку видно всюди всередині функції:

var a = 0; if (true) {

var a = 1000; alert(a); *// 1000*

}

alert(a); *// 1000*

А тепер зрівняйте з поведінкою «let»:

let a = 0; if (true) {

let a = 1000; alert(a); *// 1000*

}

alert(a); *// 0*

* змінна «let» видна тільки після оголошення:

alert(a); *// undefined*

var a = 0;

alert(b); *// error: 'b' is not defined*

let b = 0;

* змінну «let» не можна оголосити повторно:

var a;

var a; *// ок*

let b;

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

* всередині циклу змінна «let» буде оголошена нова для кожної ітерації:

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

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

**Константи**

У JavaScript'е до ES-2015 не було констант, але оскільки необхідність в них все ж була і до того, то була негласна домовленість: змінні, набрані у верхньому регістрі через підкреслення, не змінювати:

var USER\_STATUS\_ACTIVE = 1; var USER\_STATUS\_BANNED = 2;

Константи необхідні, щоб уникнути появи «magic numbers», порівняйте наступний код « if (status == 2) » - про що тут мова мало кому буде зрозуміло, а ось код « if (status == USER\_STATUS\_BANNED) » вже більше інформативний

Якщо ж говорити про ері ES-2015 року, то тут вже використовуємо « const », а про решту вже подбає сам JavaScript:

const USER\_STATUS\_ACTIVE = 1;

USER\_STATUS\_ACTIVE = 2; *// error: assignment to constant variable*

*Так-так, я повторюся, це наглядний приклад правильного іменування констант - ім'я однозначно вказує на збережене значення -*

*«Статус активного користувача»*

Окремо варто зауважити, константа не дозволяє змінювати саму змінну, але якщо ми дамо константі об'єкт або масив, то з ним ви зможете робити що душа забажає:

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*

*Ось і вийшов «читабельний» код, «мотайте на вус»*

**Типи даних**

В JavaScript не так вже й багато типів даних:

* number - ціле або дробове число:

var answer = 42; var pi = 3.1415;

також існують такі спеціальні значення:

NaN (not-a-number) — результат числовий операції, яка завершилася помилкою, запустіть наступний код в консолі:

Math.sqrt(-5);

але врахуйте :

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

Infinity — за межею 1.7976931348623157E+10308 (тобто більше)

-Infinity — за межею -1.7976931348623157E+10308 (тобто менше)

* string — рядок, береться в лапки «»:

var str = "Hello World!";

*в JavaScript немає різниці між подвійними лапками і одинарними (привіт PHP)*

* boolean - логічне значення, тобто або « true » або « false »

var result = true ;

* object - це об'єкти, на них зупинюся детальніше трохи пізніше ...
* symbol - тип даних з ES-2015 року, служить для створення унікальних ідентифікаторів (про нього розповіді не буде, не в ходу ще)
* null - спеціальне значення для визначення «порожнечі»
* var result = null ;

* undefined - ще одне спеціальне значення, для «невизначеності», використовується як значення невизначеної або неіснуючої змінної, наприклад, якщо змінна оголошена, але значення їй ще не присвоєно:

*// змінна є,але немає значення*

var a;

alert(a); *// undefined*

if (typeof a == "undefined") { alert("variable is undefined");

}

*// чи може зовсім не бути змінної*

if (window["a"] == undefined) { alert("variable does not exist");

}

*У другому прикладі нас може очікувати сюрприз, якщо хтось визначить змінну « undefined », як обійти таку  «Неприємність» я ще розповім.*

**Масиви**

Масив - це колекція даних з числовими індексами. Дані можуть бути будь-якого типу, в якості прикладу, я приведу один з найпростіших варіантів

- масив з рядками:

0 1 2

var users = ["Ivan", "Petr", "Serg"]

Нумерація масивів починається з « 0 », так що для отримання першого елемента вам потрібно наступний код:

alert(users[0]); *// виведе Ivan*

Розмір масиву зберігається у властивості length:

alert(users.length); *// виведе 3*

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

*В дійсності « length » повертає індекс останнього елемента масиву + 1, так що не попадіться:*

var a = [];

a[4] = 10;

alert(a.length); *// виведе 5;*

Для перебору масиву найкраще використовувати цикл « for (;;) »:

for (let i = 0; i < users.length; i++) {

alert(users[i]); *// послідовно виведе Ivan, Petr и Serg*

}

Для роботи з останніми елементами масиву слід використовувати методи

« Push () » і « pop () »:

users.push("Sidorov"); *// додаємо елемент в кінець масиву*

var sidorov = users.pop(); *// видаляємо та повертаємо останній*

Для роботи з першими елементами масиву слід використовувати методи

« Unshift () » і « shift () »:

users.unshift("Sidorov"); *// додаємо елемент в початок масиву*

var sidorov = users.shift(); *// видаляєм та повертаємо перший елемент*

*Останні два методи працюють повільно, тому що перебудовують весь масив*

**Функції**

З функціями в JavaScript'е все просто, ось вам елементарний приклад:

function hello() {

alert("Hello world");

}

*Просто, поки не заговорити про анонімні функції…*

**Анонімні функції**

В JavaScript можна створювати анонімну функцію (тобто функцію без імені), для цього досить злегка змінити попередню конструкцію:

function() {

alert("Hello world");

}

Так як функція це цілком собі об'єкт, то її можна привласнити змінній, і (або) передати в якості параметра в іншу функцію:

var myAlert = function(name) { alert("Hello " + name);

}

function helloMike(myFunc) { *// тут функція передаєттся як параметр*

myFunc("Mike"); *// а тут мы її викликаємо*

}

helloMike(myAlert);

Анонімну функцію можна створити і тут же викликати з необхідними параметрами:

(function(name) {

alert("Hello " + name);

})("Mike");

Це НЕ складно, скоро ви до них звикнете, і вам їх буде бракувати в інших мовах.

Ах, цей ES-2015 року, він приніс нам скорочений запис для анонімних функцій

- функції-стрілки:

*// була проста анонімна функція*

var inc = function (x) { return x+1;

}

*// стала запис в один рядок*

var inc = x => x+1;

*// була функція з декількома аргументами*

var sum = function (a, b) { return a+b;

}

*// стала запис в один рядок*

var sum = (a, b) => a+b;

Даний запис звичайно ж зручний, але НЕ варто захоплюватися, якщо перестаратися, то ваш код стане нечитабельний, і за підсумком ви можете почути багато не приємних виразів в свій адрес.

**Об’єкти**

На об'єкти в JavaScript покладено дві ролі:

- сховище даних

- функціонал об'єкта

Перше призначення можна описати наступним кодом:

var user = {

name: "Ivan", age: 32

};

alert(user.name); *// Ivan*

alert(user.age); *// 32*

Це фактично реалізація key-value сховища, або кеша, або асоціативного масиву, або ..., ну ви зрозуміли, назв багато, але в JavaScript'і це об'єкт, і запис вище - це JSON - JavaScript Object Notation (хоч і з невеликими застереженнями) .

Для перебору такого сховища можна використовувати цикл « for (.. in ..) »:

for (let prop in user) {

alert(prop + "=" + user[prop]); *// виведе name=Ivan*

*// потім age=32*

}

З об'єктами, конструкторами і т.д. в JavaScript складніше буде, хоча для розуміння не так вже й багато треба, запам'ятовуйте: будь-яка функція викликана з використанням ключового слова « new » повертає нам об'єкт, а сама стає конструктором даного об'єкта:

function User(name) { this.name = name;

this.status = USER\_STATUS\_ACTIVE;

}

var me = new User("Anton");

Поведенінка функції «User()» при використанні «new» зміниться:

1. Дана конструкція, створить новий пустий об’єкт
2. Ключеве слово «this» получить посилання на цей об’єкт
3. Функція вииконається і можливо змінить об’єкт через «this» (як в прикладі вище)
4. Функція поверне «this» (за замовчуванням) Результатом буде код:

{name: "Anton", status: 1 }

*Знову відправлю читати про ES-2015 року, в даному стандарті з'явилася конструкція «class», що по суті - синтаксичний цукор для JavaScript'а - спеціально для тих, хто любить C-подібні мови програмування—* [*https://learn.javascript.ru/es-class*](https://learn.javascript.ru/es-class)*.*

**Область видимості і диво this**

Для тих, хто тільки починає своє знайомство з JavaScript я розповім наступні нюанси:

- коли ви оголошуєте змінну або функцію, то вона стає частиною « Window »:

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

alert(message); *// => 1234*

}

window["myLog"](a);

* коли шукана змінна не знайдена в поточній області видимості, то її пошуки будуть продовжені в області видимості батьківської функції:

var a = 1234; (function(){

var b = 4321; (function() {

var c = 1111; alert((a+b)/c); *// => 5*

})();

})();

* чудо-змінна « this » завжди вказує на поточний об'єкт викликає функцію (оскільки за замовчуванням всі змінні і функції потрапляють в « window », то « this == window »):

var a = 1234; function myLog() {

alert(this); *// => window*

alert(this.a); *// => 1234*

}

* контекст « this » можна змінити використовуючи функції « bind () », « call () », і « apply () »

*Все що стосується « window » відноситься лише до браузерів, але оскільки книга про jQuery, то іншу поведінку я і не розглядаю, але ось так прозоро натякаю, що воно є;)*

**Замикання**

Вивчивши замикання, можна зрозуміти багато магії в JavaScript'і. Наведу приклад коду з поясненнями:

var a = 1234;

var myFunc = function(){ var b = 4321;

var c = 1111; return function() {

return ((a+b)/c);

};

};

var anotherFunc = myFunc(); *// myFunc повертає анонімну функцію*

*// з «замкнутыми» значеннями c і b*

alert(anotherFunc()); *// => 5*

Що ж тут відбувається: функція, оголошена всередині іншої функції, має доступ до змінних батьківської функції. Втикайте в код, поки вас не осінить, про що я тут тлумачу.

*Гарне завдання, яке в повній мірі дає розуміння суті проблеми: «Армія функцій» [https://learn.javascript.ru/task/make-army*]

Рекомендовані статті по темі:

- «Прив'язка контексту і каррінг:" bind "»

* [https://learn.javascript.ru/bind]
* «Явна вказівка this: "call", "apply"» [https://learn.javascript.ru/call-apply]
* «Функції "зсередини", замикання» [<http://learn.javascript.ru/closures>]
* «Використання замикань» [<http://learn.javascript.ru/closures-usage>]
* «Closures: Front to Back»

[<http://net.tutsplus.com/tutorials/javascript-ajax/closures-front-to-back/>]

Вступна по JavaScript затягнулася, краще не полінуйтеся, і вивчіть весь підручник від Іллі Кантора— [http://learn.javascript.ru/.](http://learn.javascript.ru/)

**10% Підключаємо,знаходимо, готуємо**

Базу підготували, і тепер настав час перейти до безпосереднього вивчення jQuery. Все починається з підключення бібліотеки. І вже на цьому етапі ми можемо піти декількома шляхами:

* 1. Завантажуємо jQuery з домашньої сторінки (<http://jquery.com/)>

і покладемо поряд з нашою HTML сторінкою (раджу скачати development версію - завжди цікаво покопатися в вихідному коді

:):

<head>

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

</head>

*Даний спосіб хороший для роботи в offline, або при повільному з'єднанні з інтернетом. Окремо варто звернути увагу на шлях - скрипти в окремій папці, і це не випадково, потрібно привчати себе до порядку.*

* 1. Використовуємо CDN (надаю перевагу від Google, хоча є і Microsoft і Яндекс, і ще універсальний [http://cdnjs.com/,](http://cdnjs.com/) останній, о речі, розміщує багато популярних плагінів за що окреме спасибі):

<head>

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

</head>

*Невеликі пояснення: CDN досить розумна штука, при такому запиті бібліотеки jQuery вам повернуться HTTP заголовки, в яких буде сказано, що «протухне» цей файл лише через рік. Якщо ж ви запросите файл     адресою « Jquery / 3.1 / jquery.min.js » , то вам повернеться остання доступна версія бібліотеки з гілки 3.1 - на момент написання цих рядків це була версія 3.1.0, при цьому в заголовках « Expires » буде стояти поточна дата, і кеш буде жити лише годину. Є CDN який надається і самими розробниками jQuery, але він аж ніяк НЕ такий просунутий як у Google, і на моїй пам'яті у нього були проблеми зі стабільністю, так що будьте акуратніше при роботі з ним–* [*http://code.jquery.com/*](http://code.jquery.com/)

* 1. З використанням менеджера пакетів bower.io встановлюємо потрібну бібліотеку.Даний менеджер пакетів дозволяє встановлювати дуже багато різноманітних бібліотек і пакетів, оцініть список – <http://sindresorhus.com/bower-components/>

*Навіщо я згадую про цей менеджер пакетів? Ну може хто з вас виявиться ну дуже цікавим і подужає роботу з ним самостійно :)*

**Будь готовий**

Тепер настав час приступити до роботи - візьмемо який-небудь елемент на сторінці і спробуємо його змінити. Для цього в <head> вставимо наступний код (приклад сторінки шукайте раніше ):

<Script>

*// ми намагаємося знайти всі елементи <h2> на сторінці*

*// і змінити колір шрифту на червоний*

jQuery ( "h2" ). css ( "color" , "red" );

</ Script>

Тільки подібний код нічого не зробить, тому що, на момент виконання, на сторінці ще не буде тегів <h2> , занадто рано виконується скрипт, до завантаження всього HTML документа. Для того, щоб код спрацював правильно, ми повинні або помістити код в самий низ сторінки (головне після шуканого<h2> ), або використовувати метод « .ready () » для відстеження події « load » нашого « document » :

<script>

*// чекаємо завантаження всього документа*

*// після цього буде виконана анонімна функція*

*// яку ми передали в якості параметра*

jQuery(document).ready(function(){

jQuery("h2").css("color", "red");

});

</script>

Також можна використовувати скорочений варіант без явного виклику методу

« .Ready ()» :

<script>

$(function(){

$("h2").css("color", "red");

});

</script>

*Останній варіант варто зарахувати до «best practices», адже в jQuery*

*3.0 метод « .ready () » вже позначений як deprecated.*

Ви можете створити скільки завгодно багато подібних функцій, НЕ обов'язково весь необхідний функціонал поміщати в одну.

*$ () - це синонім для « jQuery () », щоб у вас не виникало конфліктів з іншими ~~країнами~~бібліотеками за використання $ , раджу ваш код обертати в анонімну функцію наступного виду (best practice):*

(function($, undefined){

*// тут тихо і затишно*

*// ми завжди будемо впевнені, що $ === jQuery*

*// a undefined НЕ перевизначений;)*

})(jQuery);

*Наочний код в ready.html*

**Селектори**

Як я вже говорив раніше, в пошуку елементів на сторінці полягає практично половина успішної роботи з jQuery. Так що приступимо до пошуків по документу (щоб не гортати, нехай приклад HTML буде і 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 тепер приступимо до вибірок - вибір елементів по « id » або « className » аналогічно використовуваним в CSS:

$ ( "#Content" ) *// вибираємо елемент з id = content*

$ ( "Div # content" ) *// вибираємо div з id = content (хоча і без div працює)*

$ ( ".Wrapper" ) *// вибираємо елементи з class = wrapper*

$ ( "Div.wrapper" ) *// вибираємо div ' и з class = wrapper*

$ ( ".Wrapper.box" ) *// вибираємо елементи з class = wrapper і box*

$ ( "H2" ) *// вибираємо все теги h2*

$ ( "H1, h2" ) *// вибираємо все теги h1 і h2*

*Використовуйте валідні імена класів і ідентифікаторів*

Тепер згадаємо, що ми в DOMе не одні, це таки ієрархічна структура:

$ ( "Article h2" ) *// вибираємо все теги h2 всередині тега article*

$ ( "Div article h2" ) *// вибираємо все теги h2 всередині тега article*

*// всередині тега div, в будинку який побудував Джек*

$ ( "Article" ). find ( "h2" ) *// аналогічно прикладам вище*

$ ( "Div" ). find ( "article" ). find ( "h2" ) *//*

У нас є сусіди:

$ ( "H1 + p" ) *// вибір всіх p елементів, перед якими є h1*

*// елементи (у нас тільки один такий)*

$ ( "#Stick ~ article" ) *// вибір всіх article елементів після елемента*

*// c id = stick*

$ ( "#Stick" ). prev () *// вибір попереднього елемента від знайденого*

$ ( "#Stick" ). next () *// вибір наступного елемента від знайденого*

Споріднені зв'язки:

$ ( "\*" ) *// вибір всіх елементів*

$ ( "Article > h2" ) *// вибираємо все теги h2 які є*

*// безпосередніми нащадками тега article*

$ ( "Article > \*" ) *// вибір всіх нащадків елементів article*

$ ( "Article" ). children ()

$ ( "P" ). parent () *// вибір всіх прямих предків елементів p*

$ ( "P" ). parents () *// вибір всіх предків елементів p (Не знадобиться)*

$ ( "P" ). parents ( "div" ) *// вибір всіх предків елемента p які є div*

*// parents приймає як параметр селектор*

*Якщо хочете погратися з селекторами від душі - то для цього я припас для вас відповідну сторінку — css.selectors.html*

**Пошук за атрибутами**

Ще з часів CSS2 була можливість знайти елемент з певними атрибутами, в CSS3 додали ще можливостей щодо пошуку:

a [href] - всі посилання з атрибутом « href »

a [href = #] - всі посилання з « href = # »

a [href ~ = #] - всі посилання з « # » десь в « href »

a [hreflang | = en] - всі посилання, для яких hreflang починається з « En » і обрізається по символу « - » - « en », « en-US », « en-UK » a [href ^ = http] - посилання починаються з « http »

a [href \* = "google.com" ] - посилання на погуглити

a [href $ =. pdf] - посилання на PDF файли (по ідеї)

*Зазирнувши всередину jQuery ви скоріше за все знайдете те місце, де ваш вираз буде аналізуватися за допомогою регулярних виразів, з цієї причини в селекторах необхідно екранувати спеціальні символи використовуючи зворотний слеш «*\\ *»:*

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

**Пошук по дочірнім елементам**

Хотілося б ще звернути увагу на селектори з специфікації CSS3 [<http://www.w3.org/TR/css3-selectors/>] — багато цікавих:

: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»

: first-child - перший дочірній елемент

: last-child - останній дочірній елемент

: nth-child (2n + 1) - вибірка елементів по нескладному рівняння докладніше можна прочитати в статті «Як працює nth-child»[<http://web-standards.ru/articles/nth-child/>]

:not(…) — вибрати ті що не підпадають під вкладену вибірку

Але оскільки не всі браузери знайомі з CSS3-селекторами, то ми можемо використовувати jQuery для призначення стилів:

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

**Sizzle**

*Пропустіть цей розділ, і поверніться до нього тоді, коли вас зацікавить, як відбувається пошук елементів всередині « $ »*

Як «пошукача» за елементами DOM'а jQuery використовує бібліотеку Sizzle. Дана бібліотека у свій час була невід'ємною частиною jQuery, потім

«Відбрунькувалися» в окремий проект, який з радістю використовує як сам jQuery, так і Dojo Toolkit. Але вистачить лірики, давайте перейдемо безпосередньо до пошуку, в JavaScript'і передбачені наступні методи (не в jQuery, не в Sizzle, а в JavaScript'і):

getElementById ( *id*) - пошук по « id =" ... " »

getElementsByName(*namе*) - пошук по « name ="name" » і « id =" name " »

getElementsByClassName ( *class*) - пошук по « class =" class " »

getElementsByTagName ( *tag*) - пошук по тегу

querySelectorAll ( *selector*) - пошук за довільним CSS селектором

Пробігшись швидким поглядом по списку, можна помітити метод

« QuerySelectorAll () » - він універсальний і справді зручний, так, саме цей метод бібліотека намагається викликати, коли ви згодовуєте щось в якості селектора в jQuery, але даний метод іноді нас підводить, і тоді на сцену виходить Sizzle у всій красі, озброєний усіма згаданими методами, та ще й зі своїм унікальним арсеналом:

if (document.querySelectorAll)

( function () { var oldSelect = select

*/ \* ... \* /*

select = function (selector, context, results, seed, xml) {

*// використовуємо querySelectorAll коли немає фільтрів в запиті,*

*// коли це запит не по xml об'єкту,*

*// і коли не виявлено проблем із запитом*

*// ще є пару перевірок, які я опустив для наочності*

try {

push. apply (

results,

slice. call (context. querySelectorAll (selector), 0 )

);

return results;

} Catch (qsaError) { */ \* підвів , знову , ну скільки можна \* /*}

*/ \* ... \* /*

};

});

*// вихід Sizzle*

return oldSelect (selector, context, results, seed, xml);

Але давайте вже розглянемо, як Sizzle шукає в DOM'е, якщо все таки метод

« Document.querySelectorAll () » спіткнувся. Почнемо з розбору алгоритму роботи на наступному прикладі:

$ ( "Thead> .active, tbody> .active" )

1. Отримати перший вираз до коми: « thead> .active »

2. Розбити на шматочки: « thead », « > », « .active »

3. Знайти вихідне безліч елементів за останнім шматочком (все

« .Active »)

4. Фільтрувати справа наліво (всі « .active » які знаходяться безпосередньо в « thead »)

5. Шукати наступний запит після коми (повертаємося до першого пункту)

6. Залишити тільки унікальні елементи в результаті

Дивлячись на даний алгоритм ви повинні помітити, що **правила читаються справа на ліво**!

Копаємо глибше. При аналізі навіть самого маленького вираження є кілька нюансів на які варто звернути увагу - перший - це пріоритет пошуку, він різний для різних браузерів (в залежності від їх можливостей, або «неможливостей»):

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

)

*Чи не звертайте увагу на RegExp - це внутрішня кухня Sizzle*

Таким чином, розглядаючи вираз « div # my », Sizzle знайде спочатку елемент з « id =" my " », а потім вже перевірить на відповідність з <div> . Добре, це ніби як фільтрація, і вона теж дотримується черговість - це другий 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) { */ \* ... \* /*}

}

Третій нюанс - це відносні фільтри, які працюють відразу з двома елементами (вони вам знайомі по CSS селекторів):

relative: {

">" : {Dir: "parentNode" , first: true },

"" : {Dir: "parentNode" },

"+" : {Dir: "previousSibling" , first: true }, "~" : {dir: "previousSibling" }

},

*Ой, навіщо я вас всіх цим вантажу? Почитайте краще про оптимізацію запитів абзацом нижче.*

Офіційна документація по бібліотеці Sizzle доступна на GitHub'е проекту:

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

**Оптимізуємо вибірки**

Ну насамперед вам слід запам'ятати, що

**результати не кешуються, кожен раз, запитуючи елементи по селектору, ви ініціюєте пошук елементів знову і знову**

Поглянувши на алгоритм роботи Sizzle, відразу напрошуються кілька порад про оптимізацію по роботі з вибірками:

1. Зберігати результати пошуку (виходячи з постулату вище):

*// було*

$ ( "A.button" ). addClass ( "active" );

*/ \* ... \* /*

$ ( "A.button" ). click ( function () { */ \* ... \* /*});

*// стало*

var $ button = $ ( "a.button" );

$ button. addClass ( "active" );

*/ \* .... \* /*

$ button. click ( function () { */ \* ... \* /*});

*Правильна IDE про подібні речі знає, і буде вам час від часу нагадувати;)*

2. Або використовувати ланцюжок викликів (що по суті аналогічно першому правилу):

*// було*

$ ( "A.button" ). addClass ( "active" );

$ ( "A.button" ). click ( function () { */ \* ... \* /*});

*// стало*

$ ( "A.button" ). addClass ( "active" )

. click ( function () { */ \* ... \* /*});

3. Використовувати « context » (це такий другий параметр при виборі по селектору):

*// було*

$ ( ".Content a.button" );

*// стало*

$ ( "A.button" , ".content" );

$ ( ".Content" ). find ( "a.button" ,); *// трохи швидше*

4. Розбивати запит на більш прості складові частини використовуючи « context », і зберігати проміжні дані (як наслідок з правил №1 і №3)

*// було*

$ ( ".Content a.button" );

$ ( ".Content h3.title" );

*// стало*

var $ content = $ ( ".content" )

$ content. find ( "a.button" );

$ content. find ( "h3.title" );

5. Використовувати більш «їстівні» селектори щоб допомогти методу

« .QuerySelectorAll () », тобто якщо у вас немає впевненості в правильності написання селектора, або сумніваєтеся в тому, що всі браузери підтримують необхідний CSS селектор, то краще розділити

«Складний» селектор на кілька простіших:

*// було*

$ ( ".Content div input: disabled" );

*// стало*

$ ( ".Content div" ). find ( "input: disabled" );

6. Чи не використовувати jQuery, а працювати з «native» функціями JavaScript'у

*Є ще один пункт - вибирати найшвидший селектор з можливих, але тут без гарного багажу знань НЕ обійтися, так що дерзайте, пробуйте і надсилайте ваші приклади.*

Для наочності найкраще поглянути на порівняльний тест sizzle.html (даний тест був спочатку розроблений Іллею Кантором для майстер-класу по JavaScript і jQuery )

*Маленька хитрість від творців jQuery - запити по id елемента не доходять до Sizzle, а зїдаються « document.getElementById () » в якості параметра:*

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

**Приклади оптимізацій**

Вибір за ідентифікатором елемента - найшвидший з можливих, намагайтеся використовувати його:

*// всередині одна регулярочка + getElementById ()*

$ ( "#Content" )

*// а ось так ще швидше*

$ (Document. GetElementById ( "content" ))

*// але економія незначна*

*// а зручність використання прагне до нуля*

Селектор « div # content » працює на порядок повільніше, ніж пошук лише за ідентифікатором - « #content », але і він має право на існування в разі, якщо ваш скрипт використовується на декількох сторінках, а логіка вимагає лише обробляти поведінку для елемента <div > . Даний селектор можна уявити в двох варіантах:

*// getElementById () + фільтрація*

$ ( "#Content" ). filter ( "div" );

*// залишаємо як є і сподіваємося на QuerySelectorAll ()*

$ ( "Div # content" );

В результаті тестування одержуємо наступний розклад:

- приклад з використанням « .filter () » працює швидше в браузерах Chrome, Firefox і IE9.0 +

- обидва способи працюють нарівні в браузерах IE8.0 і мобільному Safari

- другий приклад працює в два рази швидше в останніх версіях Opera

Висновки робимо самі.

**20% Атрибути елементів і CSS**

У попередніх прикладах ми вже змінювали CSS-властивості DOM-елементів, використовуючи однойменний метод « .css () », але це далеко не все. Тепер копнемо глибше, щоб не штурмувати форуми банальними питаннями;)

Копати почнемо з більш досконального вивчення методу « .css () »:

css ( *property*) - отримання значення CSS властивості css ( *property*, *value*) - установка значення CSS властивості css ({ *key*: *value*, *key*: *value*}) - установка декількох значень

css ( *property*, function ( *index*, *value*) { return *value*}) - тут для установки значення використовується функція зворотного виклику (в просторіччі

- callback-функція), index це порядковий номер елемента в вибірці, value - старе значення властивості

*Метод « .css () » повертає поточне значення, а не прописане в CSS файлі за вказаним селектору*

Приклади використання ( css.html ):

$ ( "#My" ). css ( 'color' ) *// отримуємо значення кольору шрифту*

$ ( "#My" ). css ( 'color' , 'red' ) *// встановлюємо значення кольору шрифту*

*// установка декількох значень*

$ ( "#My" ). css ({

'color' : 'red' ,

'font-size' : '14px' , 'margin-left' : '10px'

})

*// альтернативний спосіб*

$ ( "#My" ). css ({

color: 'red' , fontSize: '14px' , marginLeft: '10px' ,

})

*// використовуючи функцію зворотного виклику*

$ ( "#My" ). css ( 'height' , function (i, value) { return parseFloat (value) \* 1.2 ;

})

Ну, начебто з CSS розібралися, хоча ні - варто ще описати маніпуляції з класами, теж з розряду первинних навичок:

addClass ( *className*) - додавання класу елементу

addClass ( function ( *index*, *currentClass*) { return *className*}) - додавання класу використовуючи функцію зворотного виклику

hasClass ( *className*) - перевірка на причетність до певного класу

removeClass ( *className*) - видалення класу

removeClass ( function ( *index*, *currentClass*) { return *className*}) - видалення класу використовуючи функцію зворотного виклику

toggleClass ( *className*) - перемикання класу

toggleClass ( *className, switch*) - перемикання класу по прапору switch

toggleClass ( function ( *index*, *currentClass, switch*) { return *className*}, *switch*) - перемикання класу використовуючи функцію зворотнього виклику

*У наведених методах як « className » може бути, що рядок містить список класів через пробіл.*

*Мені ні разу НЕ доводилося використовувати дані методи з функціями зворотного виклику, і лише одного разу став у нагоді прапор «switch», так що не заморачуйтесь все це запам'ятовувати, та й надалі, цитуючи керівництво по jQuery, я буду свідомо опускати деякі «можливості»*

Але вистачить займатися перекладом офіційної документації, перейдемо до наочних прикладів ( class.html ):

*// додаємо кілька класів за раз*

$ ( "#My" ). addClass ( 'active notice' )

*// перемикаємо кілька класів*

$ ( "#My" ). toggleClass ( 'active notice' )

*// працює ось так (схоже на класовий 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" >

*// аналогічно до попереднього прикладу*

$ ( "#My" ). toggleClass ( 'active' )

$ ( "#My" ). toggleClass ( 'notice' )

*// перевіряємо наявність класу (-ів)*

$ ( "#My" ). hasClass ( 'active' )

*// видаляємо кілька класів за раз*

$ ( "#My" ). removeClass ( 'active notice' )

Також, варто згадати, що у DOM елементів бувають атрибути відмінні від класу, і ми їх теж можемо змінювати, для цього нам потрібні наступні методи:

attr ( *attrName*) - отримання значення атрибута

attr ( *attrName, attrValue*) - установка значення атрибута (також можна використовувати hash, яку функцію зворотного виклику)

removeAttr ( *attrName*) - видалення атрибута

Атрибути - це все те, що ми бачимо усередині кутових лапок, коли пишемо HTML код:

*<! - В даному прикладі це href, title, class ->*

<a href= "#top" title= "anchor" class= "simple"> To Top </a>

Атрибути, з якими вам частіше за інших доведеться стикатися:

*// отримання альтернативного тексту картинки*

var altText = $ ( 'img' ). attr ( 'alt' )

*// зміна адреси картинки*

$ ( 'Img' ) .attr ( 'src' , '/images/default.png' )

*// працюємо з посиланнями*

$ ( 'A # my' ). attr ({

'href' : 'http://anton.shevchuk.name' , 'title' : 'My Personal Blog' ,

});

Крім атрибутів, також є властивості елементів, до них відноситься

« SelectedIndex », « tagName », « nodeName », « nodeType » , « ownerDocument »,

« DefaultChecked » і « defaultSelected ». Ну начебто список невеликий, можна і запам'ятати. Для роботи з властивостями використовуємо методи з сімейства

« .Prop () » :

prop ( *propName*) - отримання значення властивості

prop ( *propName, propValue*) - установка значення властивості (також можна використовувати hash, яку функцію зворотного виклику)

removeProp ( *propName*) - видалення властивості (скоріше за все ніколи не знадобиться)

А тепер вимкніть музику, і запам'ятайте наступне - **для відключення елементів форми, і для перевірки / зміни стану чекбоксів ми завжди використовуємо метод « .prop () »,**нехай вас не бентежить наявність однойменних атрибутів в HTML (це я про « disabled » і « checked »), використовуємо « .Prop () » і крапка (наочний приклад property.html )

**30% Події**

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».

Перш ніж приступити до прочитання даної глави, варто визначитися, що ж із себе представляють події web-сторінки. Так ось - події - це будь-які дії користувача, будь то введення даних з клавіатури, скролінг сторінки і звичайно ж «кліки».

*А ще існують події створювані скриптами, і їх обробники - тригери і хендлери, але про них трохи пізніше.*

jQuery працює практично зі всіма подіями в JavaScript'і, наведу список одих з невеликими поясненнями:

change - зміна значення елемента (значення, при втрати фокусу, елемента відрізняється від початкового, при отриманні фокусу)

click - клік по елементу (порядок подій: « mousedown » →

« Mouseup » → « click »)

dblclick - подвійне клацання мишки

resize - зміна розмірів елементів

scroll - скролінг елемента

select - вибір тексту для « input [type = text] » і « textarea »

submit - відправка форми

focus - фокус на елементі, актуально для « input [type = text] », але в сучасних браузерах працює і з іншими елементами

blur - фокус пішов з елемента, актуально тільки для елементів

« Input [type = text] » - спрацьовує при кліці по іншому елементу на сторінці або за подією клавіатури (наприклад перемикання по tab'у)

focusin - фокус на елементі, дана подія спрацьовує на «предке» елемента, для якого відбулася подія « focus »

focusout - фокус пішов з елемента, дана подія спрацьовує на «предка» елемента, для якого відбулася подія « blur »

keydown - натискання клавіші на клавіатурі

keypress - натискання клавіші на клавіатурі, послідовність

« Keydown → keypress → keyup » keyup - віджимання клавіші на клавіатурі

mousedown - натискання клавіші миші

mouseup - віджимання кнопки миші

 mousemove - рух курсора

mouseenter - наведення курсору на елемент, не спрацьовує при переході фокусу на дочірні елементи

mouseleave - висновок курсору з елемента, НЕ спрацьовує при переході фокусу на дочірні елементи

mouseover - наведення курсору на елемент

mouseout - висновок курсору з елемента

*Випробувати події можна на прикладі з подіями мишки і елементами форми . Для більшості подій існують                       «Shorthand» методи, так для відстеження « click » можна використовувати « .click () » :)*

Виклик більшості з перерахованих подій можна «емулювати» безпосередньо з самого скрипта:

<Script>

$ ( "#Menu li a" ). click ()

*// або використовуючи метод trigger*

$ ( "#Menu li a" ). trigger ( "click" )

</ Script>

Тепер варто розповісти трохи і про обробника подій, для прикладу візьму код рядком вище, і злегка його модифікую:

$ ( "#Menu li a" ). click ( function (event) { alert ( "Hello!" )

})

Тепер клікнувши на посилання ви побачите привітання і після закриття браузер перейде за посиланням зазначеної в атрибуті « href ». Але це не зовсім те, що мені хотілося - треба було лише вивести текст, і нікуди НЕ йти. Ага, для цього варто скасувати дію по замовчуванням:

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

})

Тепер переходу немає, тому що метод « event.preventDefault () » запобігає дана дія. Але от якщо хтось повісить ще один обробник на саме меню?

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

})

В результаті ми отримаємо два повідомлення, але чому? Якщо у вас виникає таке запитання, значить ви ще не знайомі з тим, як оброблюються події. Спробую коротенько дати вступну, коли ви натискаєте на елемент в DOM дереві, то відбувається «занурення» події - тобто спочатку всі батьківські елементи можуть обробити «клік», і лише потім він добереться до елемента за яким було здійснено, але і це ще не все, потім подія починає проробляти зворотний шлях - «спливає», даючи тим самим другий шанс батьківським елементам обробити подію.

*Але не так все гладко, у нас же є IE, який принципово не працює з «зануренням», тому всі вирішили йти по шляху найменшого опору і обробляють події лише на етапі*

*«Спливання».*

*Рекомендую до прочитання статтю « Всплиття і перехоплення » з уже згаданого підручника Кантора*

Добре, начебто зрозуміло, тепер повернемося до нашого прикладу, і намагаємося зрозуміти що ж у нас відбувається - у нас є обробник кліка для посилання і безпосередньо для самого меню, в якому це посилання знаходиться. Тепер клікаючи по посиланню, спрацьовує обробник події на засланні, і потім подія спливає до меню, і спрацьовує його обробник події « click ». Але це не зовсім бажаний результат, і для боротьби з подібним шкідництвом, необхідно зупиняти «спливання» подій:

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

})

Для прискорення розробки в jQuery є швидкий спосіб виклику цих двох методів за раз:

$ ( "#Menu li a" ). click ( function (event) { return false ; *// ось це він :)*

})

Тепер у вас є достатній багаж знань, щоб легко маніпулювати подіями на сторінці. Хоча я додам ще трохи - для того, щоб спрацював лише ваш обробник події, можна використовувати 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 даному прикладі, при кліці по посиланню буде виведено лише одне повідомлення. І так, порядок має значення.

**Вчимося керувати**

Ми вже встигли познайомитися з методом « .click () », в дійсності цей метод представляє з себе обгортку для виклику « .on () » і « .trigger () »:

if (arguments.length> 0 ) {this. on ( "click" , null, data, fn):

} Else {

this. trigger ( "click" );

}

*Ой, код я трохи змінив - для читання, якщо ж цікавість «восторжествує», то шукайте в вихідному коді по рядку « dblclick »*

Ну так давайте ж спробуємо без цих обгорток:

*// вішаємо обробник*

$ ( '.Class' ). on ( 'click' , function () {

*// щось робимо*

});

*// викликаємо обробник*

$ ( '.Class' ). trigger ( 'click' );

*// відключаємо обробник*

$ ( '.Class' ). off ( 'click' );

Можна повісити обробник подій практично на будь-який об'єкт:

*// простіше нікуди*

var obj = {

test: function () {

console. log ( 'obj.test' );

}

}

*// створюємо обробник довільного події someEvent*

$ (Obj). on ( 'someEvent' , function () { console. log ( 'obj.someEvent' ); this. test ();

});

*// ініціюємо подія someEvent*

$ (Obj). trigger ( 'someEvent' );

*// полюбопитствуем*

console. log (obj);

*Скопіюйте наведений код в консоль і запустіть, я думаю вам буде цікаво;)*

**Простір імен**

Як ви вже дізналися, коли ми хочемо створити / видалити свій обробник подій, ми пишемо наступний код:

*// створюємо свій обробник*

$ ( '.Class' ). on ( 'click' , function () {

*// щось робимо*

});

*// видаляємо все обробники*

$ ( '.Class' ). off ();

Але як завжди, є ситуації коли нам необхідно відключити не всі обробники (як приклад, треба скасувати обробку якогось контрола певним плагіном), в цьому випадку нам на допомогу приходять простори імен, використовувати їх досить легко:

*// створюємо обробник*

$ ( '.Class' ). on ( 'click.namespace' , function () {

*// щось робимо*

});

*// викликаємо обробник*

$ ( '.Class' ). trigger ( 'click.namespace' );

*// викликаємо всі обробники без простору імен*

$ ( '.Class' ). trigger ( 'click!' );

*// видаляємо все обробники click в даному просторі імен*

$ ( '.Class' ). off ( 'click.namespace' );

Ще приклад, вішаємо обробник, який виводить текст в консоль:

$ ( '.Class' ). on ( 'click.namespace' , function () {console. log ( 'bang' );

});

*// викликаємо подія, наш обробник спрацює*

$ ( '.Class' ). trigger ( 'click.namespace' );

*// теж працює*

$ ( '.Class' ). trigger ( 'click' );

*// подія з іншого простору імен, наш обробник НЕ буде викликаний*

$ ( '.Class' ). trigger ( 'click.other' );

Також, є підтримка декількох просторів імен:

$ ( '.Class' ). on ( 'click.a.b' , function () {

*// для простору імен a і b*

});

*// викликаємо обробник з простору a*

$ ( '.Class' ). trigger ( 'click.a' );

*// скасовуємо обробник click для простору b*

$ ( '.Class' ). off ( 'click.b' );

Можна одним махом видалити всі обробники з певного простору імен:

*// обробник кліка*

$ ( '.Class' ). on ( 'click.namespace' , function () {});

*// обробник фокус*

$ ( '.Class' ). on ( 'blur.namespace' , function () {});

*// передумали, і все скасували*

$ ( '.Class' ). off ( '.namespace' );

Офіційна документація мізерна на цей рахунок, і я сподіваюся мій приклад допоможе краще розібратися в даному питанні ( events.namespace.html ).

**«Живі» події**

*Я тут трохи забіжу вперед, так що якщо чогось стало незрозуміло, відкладіть цей розділ «на потім».*

Варто звернути увагу на ще одну задачку, яка дуже часто ставиться перед розробником - це додавання обробників подій для елементів, які додаються на сторінку динамічно. Мабуть, треба навести приклад такого завдання:

*«- У нас є HTML сторінка, на якій всі внутрішні посилання будуть довантажуватися AJAX'ом, дане твердження справедливо і для довантажувати HTML'а теж»*

Перша умова вирішується просто:

$ ( 'A [href ^ = \\ /]' ). on ( 'click' , function () { var url = $ (this). attr ( 'href' );

$ ( 'Body' ). load (url + 'body> \*' ); return false ;

});

*Для наочності, домовимося, що внутрішні посилання містять відносні шляхи від кореня сайту.*

З другою умовою трохи складніше ситуація, але теж цілком вирішувана:

$ ( 'Body' ). on ( 'click' , 'a [href ^ = \\ /]' , function () { var url = $ (this). attr ( 'href' );

$ ( 'Body' ). load (url + 'body> \*' ); return false ;

});

Відмінностей не так вже й багато, проясню, що відбувається:

- насамперед на елемент <body> буде повішений обробник події

« Click »

- даний обробник буде спрацьовувати тільки в тому випадку, коли подія буде відноситься до елементу <a>

Робота даної схеми базується на «спливанні» подій, так що використовуючи метод « event.stopPropagation () » ви зможете запобігти виконанню

«Живих» оброблювачів

*Ліричний відступ до історії: жив собі колись плагін для jQuery, називався « . live () », дозволяв він вішати обробники на елементи DOM дерева яких ще немає (підкачуємі AJAX'ом або ще як), а потім ~~він помер~~його внесли в саме ядро. Метод « .live () » на той час працював лише з « document ». Потім з'явився метод*

*« .Delegate () » який навчився вішати обробник на довільний елемент, а потім і він був поглинений методом « .on () ». Так що не лякайтеся сильно, якщо зустрінете старий метод « .live () », адаптувати під нові версії jQuery його буде не так вже й складно (ну я на це сподіваюся)*

**Оптимізація**

Неявним бонусом від використання «живих» подій можна вважати можливість оптимізації, про «потрібні» випадки я і розповім.

Випадок перший, банальний - уявіть собі таблицю на тисячу рядків і десяток стовпців, а тепер спробуйте підрахувати, скільки пам'яті з'їдять обробники події « click » для кожної клітинки? Ось-ось, варто це переписати в один обробник:

$ ( 'Table' ). on ( 'click' , 'td' , function () { */ \* ... \* /*}

Випадок другий, надуманий - необхідно записувати дії користувача на сторінці, тобто відслідковувати кліки по незліченній кількості об'єктів:

$ ( 'Body' ). on ( 'click' , '\*' , function () {console. info ( "Click on" + this.tagName);

});

Приклад роботи даного «надуманого» варіанту можна подивитися на сторінці events.optimization.html

**Touch події**

Смартфони з великим сенсорним екраном - це вже норма життя, і будь-якого web-розробнику рано чи пізно буде потрібно розробляти інтерфейси з підтримкою «touch» подій. На цей випадок у JavaScript'і передбачені наступні події:

touchstart - подія схоже з « mousedown », відбувається при торканні пальцем екрану

touchend - прибираємо палець з екрана, ака « mouseup »

touchmove - водимо пальцем по екрану - « mousemove »

touchcancel - дивна подія, скасування « touch » до того, як палець був прибраний

Про те як з ними працювати, можна отримати з відмінною статті англійською мовою - «Touching and Gesturing on iPhone, Android, and More» [ [http://www.sitepen.com/blog/?p=3425](https://translate.google.com/translate?hl=uk&prev=_t&sl=ru&tl=uk&u=http://www.sitepen.com/blog/%3Fp%3D3425) ] (хоч розповідь там і про Dojo Toolkit).

В jQuery Mobile робота з touch подіями йде «з коробки». Щоб jQuery UI змусити працювати з touch подіями слід використовувати бібліотеку jQuery UI Touch Punch [ [http://touchpunch.furf.com/](https://translate.google.com/translate?hl=uk&prev=_t&sl=ru&tl=uk&u=http://touchpunch.furf.com/) ] Пробуйте, але врахуйте, без touch пристрої розробка інтерфейсів для подібних пристроїв - нонсенс ( *англ. Nonsense*).

**40% Анімація**

Бібліотека jQuery дозволяє дуже легко анімувати DOM елементи, для цього є кілька шляхів, але про все по порядку, почнемо з простого « .hide () » і « .show () », ці два методу відповідно приховують або відображають елементи:

*// приховуємо всі картинки*

$ ( 'Img' ). hide ();

*// тепер повернемо їх на місце*

$ ( 'Img' ). show ();

Дані виклики оперують лише CSS атрибутом « display » і перемикають його з поточного стану в « none » і назад. В якості першого параметра можна задати швидкість анімації, для цього можна використовувати одне із зарезервованих слів « slow » або « fast », або ж вказувати швидкість в мілісекундах (1000 мс = 1 сек):

*// повільно спускаємося з гори і ... приховуємо всі картинки*

*//****slow == 600***

*//****fast == 200***

$ ( 'Img' ). hide ( 'slow' );

*// тепер повернемо їх на місце, трохи швидше*

$ ( 'Img' ). show ( 400 );

В такому випадку, зникнення елементів буде супроводжуватися анімацією атрибутів « width », « height », « opacity » та інших (див. Приклад в hide.html ). На додачу до цих двох методів є ще метод « .toggle () », він працює як перемикач « hide → show » або « show → hide ».

Тепер йдемо трошки далі - другим параметром в наведених методах може бути callback-функція - вона буде виконана після закінчення анімації елементів:

*// приховуємо всі картинки*

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

*// після відображаємо alert*

alert ( "Images was hidden" );

});

Наведу ілюстрацію для наглядності процесу анімації:



**Рисунок 1 — анимация show()**

*Анімацію таких атрибутів як « height », « width » і « opacity » видно неозброєним поглядом, в дійсності ж це далеко не все, заглянувши всередину jQuery можна побачити, що так само змінюються внутрішні і зовнішні відступи - « padding » і « margin » - так що не варто про це забувати.*

Йдемо далі - у нас на черзі набір методів з сімейства slide -

« .SlideUp () », « .slideDown () » і « .slideToggle () ». Їх поведінка схожа з попередніми функціями, але анімація буде зачіпати лише висоту блоків

- дивимося приклад slide.html (ну і ілюстрації так само є):



**Рисунок 2 — анимация slideDown()**

Перш ніж перейти до десерту згадаю сімейство функцій fade - вони маніпулюють лише « opacity »:

fadeIn ( *duration, callback*) - змінює « opacity » від 0 до попереднього fadeOut ( *duration, callback*) - змінює « opacity » від поточного до 0 fadeToggle ( *duration, callback*) - перемикач між « In » та « Out »

fadeTo ( *duration, opacity, callback*) - змінює значення « opacity » до необхідного значення

А тепер саме солодке - всі ефекти анімації в jQuery крутяться навколо методу « .animate () ». Ця функція бере один або кілька CSS-властивостей елемента і змінює їх від вихідного до заданого за N-ну кількість ітерацій (кількість ітерацій залежить від зазначеного часу, але не рідше однієї ітерації в 13мс, якщо я правильно знайшов це значення). Ну що-ж, від слів до справи, спробуємо реалізувати функції « .fadeIn () » і « .fadeout () » за допомогою « .animate () » ( див . Приклад animate.html ):

*// fadeOut ()*

$ ( 'Article img' ). animate ({ 'opacity' : 'hide'

})

*// fadeIn ()*

$ ( 'Article img' ). animate ({ 'opacity' : 'show'

})

Все просто, давайте тепер ускладнимо завдання - змінимо розмір блоків і прозорість:

*// значення зазначених властивостей будуть плавно змінюватися*

*// від поточних до заданих*

$ ( 'Article img' ). animate ({ 'opacity' : 0.5 ,

'height' : '50px' ,

'width' : '250px'

})

Як бачите - теж нескладно, тепер спробуємо відштовхуватися від поточних значень, а не ставити необхідні:

*// змінюємо, крок за кроком*

$ ( 'Article img' ). animate ({ 'opacity' : '- = 0.1' , 'height' : '+ = 10px'

})

Погралися і вистачить, пора ускладнити вам життя - у методу « .animate () » може бути більш одного параметра, і пора приступити до їх розбору. Набір параметрів може бути різним, наведу перший, той, що простіше:

params - CSS властивості - з цим ми вже познайомилися

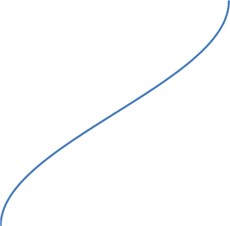
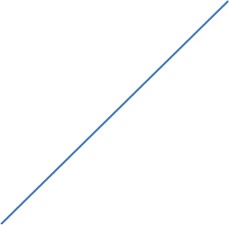
duration    - швидкість анімації, вже згадувалася раніше, вказується в мілісекундах, або використовуючи ключові слова « fast » або

« Slow »

easing - вказуємо яку функцію використовуватимемо для зміни значень

callback - функція, яка буде викликана після закінчення анімації

З наведених параметрів нам тільки « easing " не зустрічався раніше - я його беріг на зараз - цей параметр вказує, яка функція буде використовуватися для процесу анімації значень. Це можуть бути лінійні, квадратичні, кубічно і будь-які інші функції. «З коробки» ми можемо вибрати лише між « linear » і « swing »:



**Рисунок 3 — easing «linear» Рисунок 4 — easing «swing»**

Заглянувши в код jQuery ми легко знайдемо відповідний код:

linear: function (p) { return p;

},

swing: function (p) {

return 0.5 - Math. cos (p \* Math.PI) / 2 ;

}

*p - коефіцієнт проходження анімації, змінюється від 0 до 1*

*Складно? Хочете більше і відразу? Тоді шукайте easing plugin на сторінці*[*http://gsgd.co.uk/sandbox/jquery/easing/ ,*](https://translate.google.com/translate?hl=uk&prev=_t&sl=ru&tl=uk&u=http://gsgd.co.uk/sandbox/jquery/easing/)*він дійсно з розряду «must have».*

*Підключайте і використовуйте одну з трьох десятків функцій easing (наочно, з ілюстраціями - animate.easing.html , а так само*[*http://easings.net/ )*](https://translate.google.com/translate?hl=uk&prev=_t&sl=ru&tl=uk&u=http://easings.net/))

Але давайте повернемося до методу « .animate () », яка в якості параметрів може приймати ще один набір параметрів, який вже не буде здаватися таким простим:

params - CSS властивості (вже було)

options - тут цілий набір можливостей, частина вже описувалася раніше:

duration - швидкість анімації

easing - функція ( « linear » або « swing »)

complete - функція, яка буде викликана після закінчення анімації

step - функція, яка буде викликана на кожному кроці анімації, про неї розповім трохи нижче

queue - прапор / параметр черги, трохи пізніше опишу детальніше

specialEasing - хеш в якому можна описати яку саме easing функцію слід використовувати для зміни певних параметрів

**Step-by-step**

Хотілося б окремо зупинитися на функції « step », і для наочності наведу приклад реалізації подібної функції, яка відображає поточне значення анімованого параметра:

var customStep = function (now, obj) {obj.elem; *// об'єкт анімації*

obj.prop; *// параметр, який анімується*

obj.start; *// початкове значення*

obj.end; *// кінцеве значення*

obj.pos; *// коефіцієнт, змінюється від 0 до 1*

obj.options; *// опції налаштувань анімації*

now *; // поточне значення анімованого параметра, обчислюється як*

*// now = (obj.end - obj.start) \* obj.pos*

$ (This). html (obj.prop + ': ' + now + obj.unit); *// вивід тексту*

}

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

*Мені жодного разу не доводилося використовувати step-функції, лише тільки для прикладу*

**У чергу ... ©**

Трохи про черговість роботи методу « .animate () » - більшість читачів, напевно, вже знайоме з організацією послідовної анімації - для цього ми можемо використовувати ланцюжок викликів:

$ ( '#Box' )

*// говоримо що міняємо*

. animate ({left: '+ = 100' })

*// наступний виклик додається в чергу на виконання*

. animate ({top: '+ = 100' })

Для паралельного запуску анімації, необхідно буде внести наступні зміни :

$ ( '#Box' )

*// говоримо що міняємо*

. animate ({left: '+ = 100' })

*// наступний виклик буде ігнорувати чергу*

. animate ({top: '+ = 100' }, {queue: false })

Є ще чудова функція « .stop () », яка дозволяє зупинити поточну анімацію на півдорозі, а так же почистити чергу при необхідності. Для забезпечення такої різниці у поведінці функції, вона приймає три параметри:

queue - ім'я черги; для роботи з чергою анімації « fx » даний параметр опускаємо ( « fx » - черга за замовчуванням)

clearQueue - прапор очищення черги

jumpToEnd - застосувати результат анімації

*// зупиняємо виконання поточної анімації*

$ ( '#Box' ). stop ();

*// зупиняємо виконання поточної анімації*

*// і всіх наступних (чистимо чергу)*

$ ( '#Box' ). stop ( true );

*// зупиняємо виконання поточної анімації і всіх наступних*

*// але застосовуємо результат поточної*

$ ( '#Box' ). stop ( true , true );

*// зупиняємо виконання тільки поточної анімації*

*// і застосовуємо її результат*

$ ( '#Box' ). stop ( false , true );

Приклад є, і вимагає ваших спроб і помилок - animate.queue.html

*Замітка на майбутнє: якщо ви зробили меню, що випадає, і погравшись з мишкою воно продовжує випадати і зникати, то значить треба вставити « .stop () » в обробник події*

За замовчуванням вся анімація над об'єктом складається в чергу « fx », але з версії 1.7 можна вказувати довільну чергу:

$ ( '#Box' )

. animate ({ 'left' : '- = 100' }, {queue: 'x' }) *// складаємо чергу X*

. dequeue ( 'x' ) *// запускаємо чергу X*

$ ( '#Box' ). stop ( 'x' ) *// зупиняємо анімацію в черзі X*

Для чого нам може знадобитися довільна черга? Так для розпаралелювання анімації, щоб ми могли запустити одну чергу анімації, і в будь-який інший момент запустити іншу чергу, можливо, це заклад під гри? Але чого гадати, давайте дивитися - animate.game.html

**Відключення**

Іноді потрібно відключити всю анімацію (наприклад, для налагодження) скористайтеся наступною конструкцією:

jQuery.fx.off = true ;

**50% Маніпуляції с DOM**

А тепер я буду довго і нудно розповідати про те, як з допомогою jQuery можна змінювати DOM дерево на сторінці, тобто додавати і видаляти елементи, але чого це я, глава насправді не буде об'ємною :)

Почнемо зі створення елементів для подальшої роботи з ними, документація нам дбайливо повідомляє, що тут все просто:

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

Цей факт цілком робочий, та ось тільки продуктивністю він блищати не буде, адже всередині буде все це розбиратися за допомогою методу

« JQuery.parseHTML () », який зовсім не швидкий. Але ми можемо допомогти парсеру якщо атрибути елемента будемо передавати другим параметром:

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

Можемо зробити ще простіше :

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

І цей спосіб буде працювати навіть швидше (ну зовсім крапельку), але чому? Для того, щоб відповісти на дане питання - загляньте в код jQuery, в найголовнішу функцію « init () », в її коді можна знайти алгоритм розбору попереднього прикладу:

1. Парс рядок, і створюємо DOM елемент в jQuery обгортці

2. Заходимо в **цикл**обробки переданих параметрів:

a. Перевіряємо, а чи немає функції у нашого елемента з такою назвою

b. Якщо немає, то встановлюємо атрибут елемента використовуючи метод

« .Attr () »

Висновки робіть самі, дe ми тут час втратили :)

Ну і на останок опишу найшвидший спосіб, який я часто використовую:

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

myDiv.className = 'some' ;

Так, це і є «чистий» JavaScript, але як на мене - в даному випадку він не менш зручний будь-яких фреймворків. І ось вам домашнє завдання - оптимізуйте такий скрипт:

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

*Виконуйте тут, папір стерпить:*

Всі необхідні нам методи зібрані в одному розділі документації - Manipulation , з деякими з них ми вже познайомилися, і залишилося зовсім трохи:

after ( *content*) - вставляє контент після кожного елемента з вибірки, тобто якщо ви зустрічаєте рядок « $ ( " p " ). after ( "<hr />" ) », читайте її як

«Після кожного параграфа буде вставлена ​​лінія»

insertAfter ( *element*) - вставляє елементи з вибірки після кожного елемента переданого в якості аргументу, тобто якщо ви зустрічаєте рядок « $ ( " <hr /> " ). insertAfter ( "p" ) »- читайте її як« лінія буде вставлена після кожного параграфа »

*- Хм, а я різниці не побачив! - тут все легко, придивіться:*

*$ ( "Після чого додаємо" ). after ( "що додаємо" )*

*$ ( "Що додаємо" ). insertAfter ( "після чого додаємо" )*

before ( *content*) - вставляє контент перед кожним обраним елементом

insertBefore ( *element*) - вставляє елементи з вибірки перед кожним елементом переданим в якості аргументу

append ( *content*) - вставляє контент в кінець кожного елемента з вибірки, тобто рядок коду « $ ( " p " ). append ( "<hr />" ) », слід читати як« в кінець кожного параграфа буде додана лінія »

appendTo ( *element*) - вставляє обраний контент в кінець кожного елемента переданого в якості аргументу: « $ ( " <hr /> " ). appendTo ( "p" ) »

- «лінія буде додана в кінець кожного параграфа»

*Знову про різницю:*

*$ ( "Куди додаємо" ).*append *( "що додаємо" )*

*$ ( "Що додаємо" ).*appendTo *( "куди додаємо" )*

prepend ( *content*) - вставляє контент в початок кожного елемента з вибірки

prependTo ( *element*) - вставляє обраний контент в початок кожного елемента переданого в якості аргументу

Так, з цим шматочком документації ніби як розібралися, знову ж таки - відчуйте різницю перерахованих методів, адже далі будуть ще:

replaceWith ( *content*) - замінює знайдені елементи новим

replaceAll ( *target*) - вставляє контент в замін знайденому

*$ ( "Щось знаходимо" ).*replaceWith *( "на що міняємо" )*

*$ ( "Що вставляємо" ).*replaceAll *( "замість чого" )*

wrap ( *element*) - обертаємо кожен знайдений елемент новим елементом, тобто ми цукерки з коробки загортаємо в фантики

wrapAll ( *element*) - обертає знайдені елементи новим елементом, ми беремо всі цукерки, і загортаємо в один великий фантик

wrapInner ( *element*) - обертає контент кожного знайденого елемента новим елементом, беремо цукерки, прибираємо фантики, загортаємо в свій фантик, і зверху загортаємо в рідній фантик

unwrap () - видаляє батьківський елемент у знайдених елементів, фантики геть

clone ( *withDataAndEvents*) - клонує вибрані елементи, для подальшої вставки копій назад в DOM, дозволяє так само копіювати і обробники подій

detach () - видаляє елемент з DOM, але при цьому зберігає всі дані про нього в jQuery, слід використовувати, якщо треба видалити елемент, а потім повернути його назад

empty () - видаляє текст і дочірні DOM елементи

remove () - видаляє елемент з DOM, назовсім

html () - поверне HTML заданого елемента

html ( *newHtml*) - замінить HTML в заданому елементі

text() – поверне текст заданого елемента, якщо всередині елемента будуть інші HTML теги, то повернеться збірна солянка з тексту всіх елементів

text(*newText*) – замінить текст всередині обраних елементів, при спробі вставити таким чином HTML, буде отриманий текст, де теги будуть приведені до HTML entities:

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

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

Переварили? Добре, тепер настала черга методів, які працюють з розмірами, і знають координати елементів:

*Але перш ніж продовжити, хотілося б освіжити в пам'яті інформацію про обчислення висоти і ширини блокових елементів;)*

offset() – поверне позицію DOM елемента щодо document'а, дані будуть отримані у вигляді об'єкта: «{ top: 10, left: 30 }»

offset(*{ top: 10, left: 30 }*) – встановлюємо розташування DOM елемента за вказаними координатами

position() – поверне позицію DOM елемента щодо батьківського елементу

height() – повертає висоту елемента за вирахуванням відступів і кордонів; якщо у нас кілька елементів у вибірці, повернеться перший; значення, на відміну від методу «css ('height')», повертається без вказівки одиниць виміру

height(*height*) — встановлює висоту всіх елементів у вибірці, якщо значення висоти передано без вказівки одиниць виміру, то це будуть «px»

* *як нагадування, взято з мануала*

$(window).height(); *// висота вікна*

$(document).height(); *//* *висота* *HTML* *документу*

width() и width(*width*) – поводяться аналогічно методу «.height()», але працюють з шириною елемента

*Методи «.height()» и «.width()»* ***не змінюють*** *своєї поведінки в залежності від обраної блокової моделі, тобто вони завжди повертають параметри області всередині margin, padding і border'а елемента.*

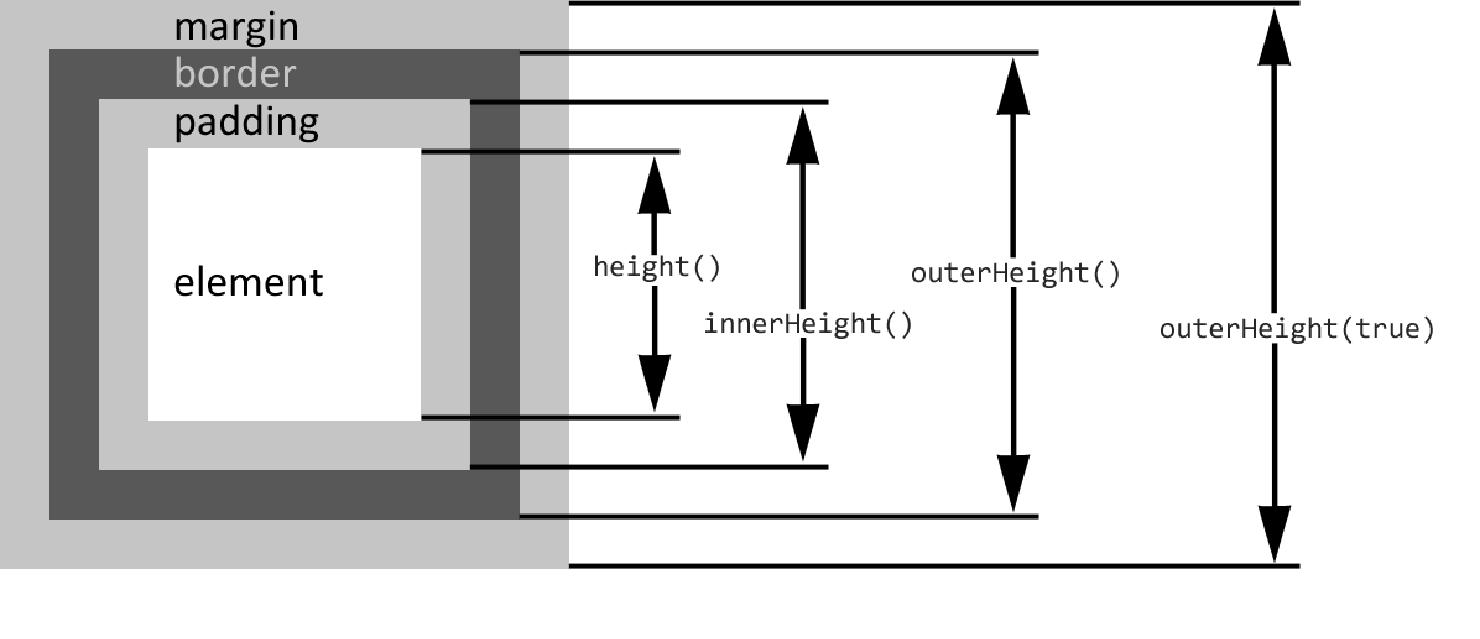
innerHeight() і innerWidth() – повернуть відповідно висоту і ширину елемента, включаючи «padding»

outerHeight() і outerWidth() – повернуть висоту і ширину елемента, включаючи

«padding» и «border»

outerHeight(*true*) і outerWidth(*true*) – висота і ширина, включаючи «padding», «border» і «margin»

Для наочності відмінностей між методами «.height()», «.innerHeight()» та «.outerHeight()» я створив сторінку height.html, а ще переробив кілька картинок з офіційної документації в одну повноцінну ілюстрацію:



**Рисунок 5 — блочна модель**

Ну і остання пара методів:

scrollLeft() – повертає значення «проскроленності» по горизонталі першого елемента з вибірки

scrollLeft(*value*) – встановлює значення горизонтального скрол для кожного елемента з вибірки

scrollTop() – повертає значення «проскроленності» по вертикалі першого елемента з вибірки

scrollTop(*value*) – встановлює значення вертикального скролу для кожного елемента з вибірки

*Значення «scrollTop» и «scrollLeft» піддаються анімації і не працюють для захованих елементів DOM*

Методів реально багато, я і сам не завжди пам'ятаю що і для чого (особливо це стосується wrap-сімейства), так що не перевантажуйте себе запам'ятовуванням усього перерахованого, головне пам'ятати що такі є і тримати під рукою документацію

**60% Работа з формами**

Форми - це, мабуть, один з найменш улюблених елементів на сторінці - поки налаштуєш зовнішній вигляд, потім ще перевір, що ввели недбайливі користувачі та виведи їм інформацію про допущені помилки, і в кінці кінців відправляєш на сервер дані з почуттям полегшення від виконаної кропіткої роботи . Так ось - про те, що допоможе в цій самій роботі я і буду розповідати.

Для початку, варто нагадати події з якими найчастіше доведеться працювати:

change — зміна значення елемента

submit — відправка форми

* яких же випадках вони нам допоможуть? Та все просто - відстеження change дозволяє обробляти такі події як зміна selectbox'а, або radiobutton'а, що буде потрібно для динамічної зміни форми. І найпростіший приклад тому - це на сторінці реєстрації вибір країни, потім по обраній країні повинен бути довантажуючи список регіонів, по регіону - список міст і так далі. Відстеження submit потрібно для перевірки правильності заповнення форми, а так само для відправки форми за допомогою AJAX. Форму візьмемо простіше:

<form action="/save/">

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

<option>User</option>

<option>Admin</option>

</select>

<input type="submit"/>

</form>

* приклади йтимуть в зворотному порядку, ось відправка форми AJAX'ом по посиланню з «action»:

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

* *трохи пізніше розповім докладніше про AJAX* $.post(

$(this).attr('action'), *// посилання куди відправляємо дані*

$(this).serialize() *// дані форми*

);

*// відключаємо дію за замовчуванням*

returnfalse;

});

Ось і перший метод– «.serialize()» – він у відповіді за «збір» даних з форми в зручному для передачі даних форматі:

name=Ivan&role=Admin

Також є метод «.serializeArray()» – він зібрані дані представляє у вигляді об'єкта:

[

{

name:"name",

value:"Ivan"

},

{

name:"role",

value:"Admin"

},

]

Тепер варто додати в даний код трохи перевірки даних:

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

if ($(this).find('input[name=name]').val() == '') { alert('Введите имя пользователя');

return false;

}

*// частина коду з відправкою// ...*

});

Ось ще один метод, який нам буде частенько потрібен:

val() – отримання значення першого елемента форми з вибірки

val(*value*) – установка значення всіх елементів форми з вибірки

Даний метод відмінно працює практично з усіма елементами форми, ось тільки з radiobutton'амі встановити значення таким чином не вийде, тут потрібно невеликий workaround:

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

*Можна звичайно ж використовувати і метод «.click()», щоб емулювати вибір необхідно пункту, але це викличе всі обробники події «click», що не бажано*

З checkbox'ами трохи простіше:

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

Перевіряємо «чекнутість» простим скриптом:

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

* *або трохи більше наочним способом*

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

Перевороти і відправляти форму AJAX'ом тепер вміємо, тепер залишилося вирішити питання з динамічною зміною форми, і для цього у нас вже є всі необхідні знання, ось, наприклад, додавання списку:

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

А якщо буде потрібно змінити список? Є на всі випадки життя:

* *візьмемо список заздалегідь, побережу чорнило*

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

* *додати новий об'єкт в список, що випадає* $select.append('<option>Manager</option>');
* *вибрати необхідний елемент*

$select.val('Value 1');

* *або за порядковим номером, починаючи з 0*

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

* *очищуємо список*

$select.remove('option');

* *перетворимо в multiple*
* *не забуваємо, що ім'я такого селекта, має бути з [], тобто.*
* *myselect[], інакше сервер отримає, лише одне значення*

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

$('select option').length

)

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

Добре, працювати з формою тепер можемо, залишилося прикрутити більш розсудливий висновок помилок (так-так, за «alert()» да по руках):

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

.before('<div class="error">Введите имя</div>'); return false;

}

При повторній відправки форми не забудьте прибрати повідомлення залишилися від попередньої перевірки:

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

Тепер можна об'єднати шматочки коду і отримати наступний варіант:

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

* *чистимо помилки*

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

* *проверяємо поля форми*

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

.before('<div class="error">Введите имя</div>'); return false;

}

* *все добре - відправляємо запит на сервер* $.post(

$(this).attr('action'), *//* *посилання куди відправляємо дані*

$(this).serialize() *// дані форми*

);

return false;

});

Тепер варто повернутися до списку подій форми, і перерахувати відсутні:

focus — фокус на елементі, для роботи з даною подією так само є «shorthand» метод «.focus()»; буде потрібно, якщо треба вивести підказку до елементу форми при наведенні

blur — фокус пішов з елемента + метод «.blur()»; стане в нагоді при валідації форми у міру заповнення полів

select — вибір тексту в «textarea» и «input[type=text]» + метод

«.select()»; якщо зберетеся розробляти свій WYSIWYG, то познайомитеся дуже щільно

submit — відпправка форми + метод «.submit()»; цей метод будете використовувати частенько

*Приклади роботи даних методів доступні на сторінці form.html*

Ось так ми і розправилися з «жахливими» формами, можливо я ще наведу кілька прикладів з реального життя, але це буде вже в наступних версіях даного підручника :)

**70% AJAX**

Що таке AJAX я думаю розповідати не варто, бо з приходом веб-два-нуля більшість користувачів вже вернуть носом від перезавантажень сторінок цілком, а з появою jQuery реалізація спростилася в рази ...

Почнемо з найпростішого - завантаження HTML коду в необхідний нам DOM елемент на сторінці. Для цієї мети нам підійде метод «.load ()». Даний метод може приймати такі параметри:

url – запитуваної сторінки

data – передані дані (необов'язковий параметр)

callback – функція яка буде викликана при завершенні запиту до сервера (необов'язковий параметр)

Тепер на прикладах:

* *в елемент з id=content буде вставлений весь HTML з вказаною сторінки*

$("#content").load("/get-my-page.html");

* *в елемент з id=content буде вставлений HTML з вказаною сторінки*
* *обраний за вказаною селектору #wrapper*

$("#content").load("/get-my-page.html #wrapper");

* *передаємо дані на сервер*

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

* *обробляємо отримані дані*

$("#content").load("/get-my-page.html", function(){ alert("Ничего оригинальней не

придумал");

});

* *мого досвіду роботи - вам дуже часто доведеться користуватися методом «.load ()» як описано в першому прикладі, а ще раджу запам'ятати другий приклад, він може виручити, коли треба реалізувати завантаження AJAX'ом, а доступу до сервера-сайду у вас немає або він обмежений.*

*Живий приклад можна побачити на сторінці – ajax.load.html*

Наступний метод з яким я вас познайомлю буде «$.ajax()» – власне, він тут головний, і всі інші AJAX методи є лише обгорткою (і «.load()» в тому ж числі). Метод «$.ajax()» приймає як параметр пачку налаштувань і URL куди стукати, наведу приклад аналогічний викликом «.load()»:

$.ajax({

url: "/get-my-page.html", *//* *вказуємо* *URL*

|  |  |  |
| --- | --- | --- |
| method: "GET", | *// HTTP метод, за замовчуванням GET* | |
| data: {"id": 42}, | *//* | *дані, які відправляємо на сервер* |
| dataType: "html", | *//* | *тип даних загружених з серверу* |

success: function (data) {

* *вішаємо свій обробник події success* $("#content").html(data)

}

});

Тут ми обробляли HTML відповідь від сервера - це добре коли нам півсторінки оновити треба, але дані краще передавати в «правильному» форматі - це XML - зрозуміло, структуровано, і надлишково, і якось не зовсім JavaScript-way, і тому наш вибір - це JSON (wikipedia в помощь):

{

"note": {

"time":"2012.09.21 13:11:15",

"text":"Разказати про JSONP"

}

}

Фактично це і є JavaScript код як є (JavaScript Object Notation якщо бути прискіпливо точним), при цьому формат вже поширений настільки, що робота з даними в іншому форматі вже не комільфо.

*Життя не стоїть на місці, є і більш зручні формати, але не в JavaScript’е :)*

Для завантаження JSON існує швидка функція-синонім – «jQuery.getJSON()»

– в якості обов'язкового параметра лише посилання, куди стукаємо, опціонально можна вказати дані, для передачі на сервер і функцію зворотного виклику

*Не можна просто так взяти і описати всі можливі параметри для виклику «$.ajax()», таки варто тримати офіційний мануал під рукою*

*– http://api.jquery.com/jQuery.ajax/*

Ще є пару-трійку функцій, які варто згадати:

get(*url, data, success, dataType*) — завантажує дані методом GET post(*url, data, success, dataType*) — завантажує дані методом POST

getScript(*url, success*) — загружає JavaScript з сервера методом GET

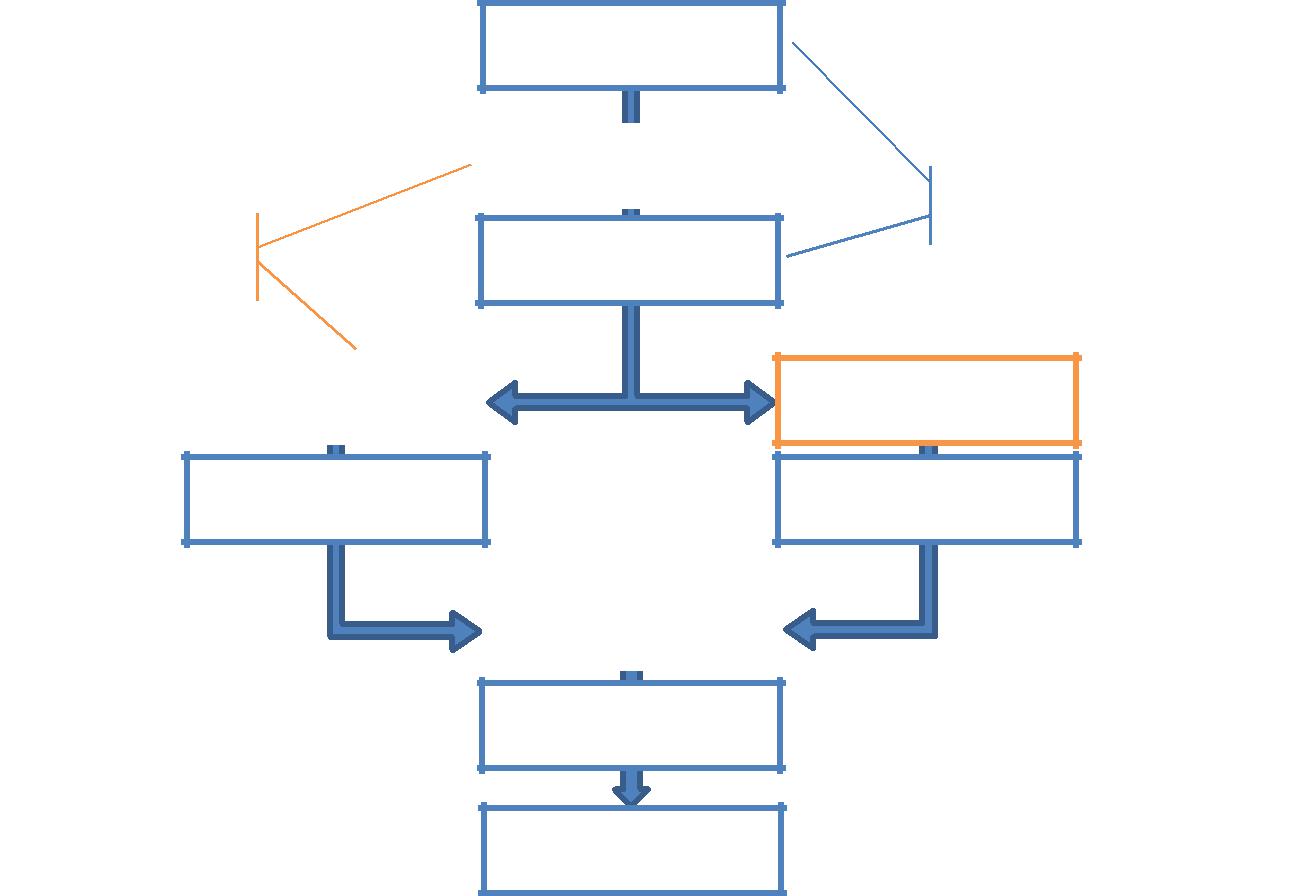
Всі вони, як я вже говорив раніше, лише обгортки над викликом «$.ajax()», в чому не складно переконатися заглянувши в вихідний код бібліотеки, і там, на стрічках з 9 137 до 9 156, ви знайдете реалізацію методів «$.get()» та

«$.post()» :)

*Дана інформація актуальна для jQuery версії 3.1.0*

**Обробники AJAX подій**

Для зручності розробки, AJAX запити кидають кілька подій, і їх природно можна і потрібно обробляти. jQuery дозволяє обробляти ці події для кожного AJAX запиту окремо, або глобально. Наведу схемку на якій наочно видно порядок виникнення подій в jQuery:



ajaxStart

beforeSend

глобальні

локальні ajaxSend

|  |  |
| --- | --- |
| success | error |
| ajaxSuccess | ajaxError |

complete

ajaxComplete

ajaxStop

Ось і повний список подій з невеликими застереженнями:

ajaxStart — дана подія виникає в разі коли побіг перший AJAX запит, і при цьому інших активних AJAX запитів в даний момент немає

beforeSend — виникає до відправлення запиту, і дозволяє редагувати XMLHttpRequest, локальна подія

ajaxSend — виникає до відправлення запиту, як і «beforeSend»

success — виникає після повернення відповіді, коли у відповіді немає помилок, локальна подія

ajaxSuccess — виникає після повернення відповіді, аналогічно

події «success»

error — виникає в разі помилки, локальна подія

ajaxError — виникає в разі помилки

complete — виникає по завершенню поточного AJAX запиту під

час будь-якого розкладу, локальна подія

ajaxComplete — глобальна подія, аналогічне complete

ajaxStop — дана подія виникає в разі, коли більше немає активних запитів

Приклад для відображення елемента з «id="loading"» під час виконання будь-якого AJAX запиту (тобто ми обробляємо глобальна подія):

$("#loading").on("ajaxSend", function(){

$(this).show(); *//* *показує елемент*

}).on("ajaxComplete", function(){

$(this).hide(); *//* *пріховує елемент*

});

*Це завдання з юзабіліті - ми завжди повинні тримати користувача сайту в курсі справи про те, що відбувається на сторінці, і відправка AJAX запиту теж потрапляє під розряд «must know». Подібне рішення у вас буде практично на будь-якому сайті, де ходить AJAX*

Для локальних подій - вносимо зміни в опції методу «.ajax()»:

$.ajax({

beforeSend: function(){

* *даний обробник буде викликаний*
* *перед відправкою даного AJAX запиту*

},

success: function(){

* *а цей при вдалому завершенні*

},

error: function(){

*// цей при виникненні помилки*

},

complete: function(){

* *і по завершенню запиту (вдалому чи ні)*

}

});

Можна глобальні обробники відключити примусово використовуючи прапор «global», для цього виставляємо його в «false», і пишемо функціонал в обхід подій «ajaxStart» та «ajaxStop»:

$.ajax({

global: false,

beforeSend: function(){

*// ...*

},

success: function(){

*// ...*

},

error: function(){

*// ...*

},

complete: function(){

*// ...*

}

});

*Дана опція частенько допомагає уникнути конфліктів при роботі з AJAX запитами, але не варто їй зловживати.*

**JSONP**

JSONP – це наш старий знайомий JSON з прошарком у вигляді callback функції О\_о. Гаразд, давайте на прикладах, ось як у нас виглядає відповідь сервера у форматі JSON:

{

"note": {

"time":"2012.09.21 13:12:42",

"text":" Розповісти навіщо потрібен JSONP"

}

}

Добре, коли у нас ці дані приходять з нашого сервера - опрацюємо, і все буде чікі-піки, але а якщо нам буде потрібно дістати дані з іншого сервера, то політика безпеки в браузерах не дозволить відправити XMLHTTPRequest на інший сервер, і треба вже буде що -то придумувати. Можна трохи напружитися і пригадати, що підключати JavaScript з іншого сервера то ми можемо, і він буде виконаний. Ось вона - зачіпка-то, а якщо підключається скрипт буде містити виклик нашій функції з підготовленими даними - то це вже щось:

alertMe({

"note": {

"time":"2012.09.21 13:13:13",

"text":" Який же профіт від використання JSONP?"

}

})

Таким чином, описавши в своєму коді функцію «alertMe()» ми зможемо обробити дані з віддаленого сервера. Найчастіше, сервера ловлять параметр «callback» або «jsonp», і використовують його як ім'я функції обгортки:

<script type="text/javascript"

src="http://domain.com/getUsers/?callback=alertMe">

</script>

Ну це було передісторія, тепер повернемося до jQuery і методу «.ajax()»:

$.ajax({

url: "http://domain.com/getUsers/?callback=?", *//* *вказуємо* *URL*

dataType: "jsonp",

success: function (data) {

*// обробляємо дані*}

});

* запитуваній URL спостережливий читач помітить незакінчену структуру «callback=?», так ось замість «?» буде підставлена ім'я ново згенерованої функції, всередині якої буде здійснюватися виклик функції «success()». Замість цієї проксі-функції можна використовувати і свою функцію, досить вказати її ім'я в якості параметра «jsonpCallback» при виклику «$.ajax()». Обидва ці підходи є в прикладі ajax.jsonp.html.
  + *ще варто згадати, що можна вказати як обзивається callback-параметр використовуючи параметр «jsonp», таким чином вказавши «jsonp:"my"» в URL буде додана структура «my=?»*

На даний момент досить багато сервісів надають API з підтримкою JSONP:

— Yahoo – пошук і більшість сервісів

— Flickr – робота з пошуком даного сервісу

— MediaWiki – відповідно і похідні – Wikipedia, Wiktionary

— CNET – пошук по новинному порталу

Використання подібного підходу дозволяє обходити обмеження накладаються сервісами на кількість запитів з одного IP, плюс не грузить сервер додатковою роботою по проксінг запитів від користувача до сервісів.

На жаль, багато провайдерів сервісів (такі як Google) відмовляють від надання доступу до їх API з використанням JSONP.

**Лікуємо JavaScript залежність**

Любов до AJAX буває надмірною, і в гонитві до Web2.0 (3.0, 4.0, ... - бажане підкреслити) ми створюємо сайти в яких всі наші дії біжать через XMLHTTPRequest. Ні, це звичайно не погано - знижуємо навантаження на сервер, канал і т.д. і т.п., але є одне «але» - у нас є пошукові машини, що не спантеличують себе виконанням JavaScript коду, а контент, захований за AJAX запитом, їм віддати все таки потрібно. Отже, у нас виникає необхідність дублювання навігації (це як мінімум) для клієнтів без JavaScript.

*Варто пам'ятати, що є ще користувачі, у яких відключений JavaScript в браузері (або навіть не підтримується, привіт тобі, рись), але ці знають, що роблять. А є ще скрипти, які ламаються, і не дають звичайним користувачам скористатися навігацією по сайту, а користувачів це дуже сильно засмучує, так що ця глава не "просто так».*

Як же все це обійти і на граблі не наступити? Так все дуже просто - створюйте звичайну навігацію, яку ви б робили не чувши ні разу про AJAX і компанії:

<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>

Даний приклад працює у нас без JavaScript'a, всі сторінки в нашому меню використовують один і той же шаблон для виведення інформації, і за фактом у нас змінюється лише вміст <div> з «id="content"». Тепер приступимо до завантаження контенту за допомогою AJAX - для цього додамо наступний код:

$(function() {

* *вішаємо обробник на всі посилання в нашому меню navigation* $("ul.navigation a").click(function(){

var url = $(this).attr("href"); *//* *візьмемо посилання*

url =+ "?ajax=true";*// додамо до неї параметр ajax=true*

$("#content").load(url); *//* *завантажуємо оновлене вміст* return false; *//* *повертаємо* *false*

*// - щоб не спрацював перехід по посиланню*

});

});

* даному прикладі ми припускаємо, що сервер, бачачи параметр «ajax=true» поверне нам не повністю всю сторінку, а лише оновлення для шуканого

елемента <div id="content">.

*Звичайно, сервер повинен бути розумніший і не вимагати явного вказівки для використання AJAX'а, а повинен цілком задовольнитися,*

*слова* *header «X\_REQUESTED\_WITH»* *зі значенням* *«XMLHttpRequest».*

*Більшість сучасних фреймворків для web-розробки з цим справляються «з коробки».*

Якщо ж керувати поведінкою сервера проблематично, і він упертий відправляє нам всю сторінку цілком, то можна написати наступний код:

$(function() {

* *вішаємо обробник на всі посилання в нашому меню navigation* $("ul.navigation a").click(function(){

var url = $(this).attr("href"); *//* *візьмемо посилання*

* + *завантажуємо сторінку цілком, але в наш контейнер вставляємо*
  + *лише вміст #content завантажується сторінки*

$("#content").load(url + " #content > \*"); return false; *//* *повертаємо*

});

});

Якщо в довантажувати вмісті так само є посилання - то ви вже повинні знати як

«оживити» події.

**Прокачуємо AJAX**

* нас є три способи для «прокачування» AJAX'а в jQuery: це створення префільтри, додавання нових конверторів та транспортів.

**Префільтри**

Префільтр - це функція, яка буде викликана до «ajaxStart», в ній ви зможете змінити як об'єкт «jqXHR», так і будь-які супутні настройки:

* *реєстрація AJAX префільтр*

$.ajaxPrefilter(function( options, originalOptions, jqXHR ) { *// наші маніпуляції над настройками і jqXHR*

});

Для чого все це? Так ось проста задача - не чекати «стару» AJAX відповідь, якщо ми запитуємо URL заново:

* *колекція поточних запитів* var currentRequests = {};

$.ajaxPrefilter(function( options, originalOptions, jqXHR ) {

* + *наша довільна настройка*

if ( options.abortOnRetry ) {

if ( currentRequests[ options.url ] ) {

* *скасовуємо старий запит* currentRequests[ options.url ].abort();

}

currentRequests[ options.url ] = jqXHR;

}

});

* *виклик з використанням фільтра* $.ajax({

*/\* ... \*/* abortOnRetry: true

})

Ще можна змінити опції виклику, ось приклад який по прапору

«crossDomain» пересилає запит на заздалегідь підготовлену проксірующую сторінку на нашому сервері:

$.ajaxPrefilter(function( options ) {

if ( options.crossDomain ) {

options.url = "/proxy/" + encodeURIComponent( options.url ); options.crossDomain = false;

}

});

Префільтри можна «вішати» на певний тип «dataType» (тобто в залежності від очікуваного типу даних від сервера будуть спрацьовувати різні фільтри):

$.ajaxPrefilter("json script", function(options, original, jqXHR) {

*/\* ... \*/*

});

Ну і останнє, для перемикання «dataType» на какой-нить інший нам досить буде повернути необхідне значення:

$.ajaxPrefilter(function( options ) {

* *це наша функція-детектор необхідних URL* if ( isActuallyScript( options.url ) ) {
* *тепер «чекаємо» script* return "script";

}

});

*Будьте дуже обережні коли оперуєте глобальними настройками, та ще через таку неявну фичу як фільтри - інвентаризують подібні підходи в супровідній документації, інакше розробники які будуть в подальшому супроводжувати ваш код будуть сильно лаятись(в якості них можете опинитися і ви самі, ну через пару місяців)*

**Конвертори**

Конвертор - функція зворотного виклику, яка викликається в тому випадку, коли отриманий типу даних не збігається з очікуваним (тобто. «dataType» вказаний невірно).

Все конвертори зберігаються в глобальних налаштуваннях «ajaxSettings»:

* *формат ключа "із\_формата в\_формат "*
* *в якості вхідного формату можна використовувати "\*"*

converters: {

*"\* text": window.String, // що завгодно приводимо до тексту*

*"text html": true,* *// текст к html (прапор true == без змін)*

*"text json": jQuery.parseJSON,* *// текст до JSON*

*"text xml": jQuery.parseXML* *// розбираємо текст як xml*

}

Для розширення набору конверторів потрібно функція «$.ajaxSetup()»:

$.ajaxSetup({

converters: {

"text mydatatype": function( textValue ) { if ( valid( textValue ) ) {

* + *розбір даних, що прийшли* return mydatatypeValue;

} else {

* + *виникла помилка*

throw exceptionObject;

}

}

}

});

*Імена «dataType» повинні завжди бути в нижньому регістрі*

Конвертори слід використовувати, якщо потрібно впровадити довільні формати «dataType», або для конвертації даних в потрібний формат. необхідний «dataType» вказуємо при виклику методу «$.ajax()»:

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

Конвертори можна задавати так само безпосередньо при виклику «$.ajax()», щоб не засмічувати загальні налаштування:

$.ajax( url, {

dataType: "xml text mydatatype",

converters: {

"xml text": function( xmlValue ) {

* *отримуємо необхідні дані з XML* return textValue;

}

}

});

*Чуть-чуть пояснень - ми запитуємо «XML», який конвертуємо в текст, який буде переданий в наш конвертор із «text» в*

*«mydatatype»*

**Транспорт**

*Використання свого транспорту - це крайній захід, вдавайтеся до неї, тільки в тому випадку якщо з поставленим завданням не можна впоратися з використанням префільтри і конверторів*

Транспорт - це об'єкт, який надає два методи – «send()» та

«abort()» – вони будуть використовуватися всередині методу «$.ajax()». Для реєстрації свого методу транспортування слід використовувати метод «$.ajaxTransport()», буде це виглядати якось так:

$.ajaxTransport( function( options, originalOptions, jqXHR ) {

if ( */\* transportCanHandleRequest \*/* ) {

return {

send: function( headers, completeCallback ) {

*/\* відправляємо запит \*/*

},

abort: function() {

*/ \* Скасовуємо запит \* /*}

};

}

});

Проясню трохи параметри з якими будемо працювати:

options

– настройки запиту (то що вказуємо при виклику «$.ajax()»)

originalOptions

– «чисті »настройки, навіть без урахування змін« за замовчуванням»

jqXHR

– об'єкт «jQuery XMLHttpRequest»

headers

– заголовки запиту у вигляді зв'язки ключ-значення

completeCallback – функція зворотного виклику, її слід використовувати для оповіщення про завершення запиту

Функція «completeCallback()» має наступну сигнатуру:

function ( status, statusText, responses, headers ) {

*/\* якийсь код \*/*

}

де:

status

– HTTP статус відповіді.

statusText

– текстова інтерпретація відповіді

responses

– це об'єкт який містить відповіді сервера у всіх форматах,

які підтримує транспорт, для прикладу: рідний

«XMLHttpRequest» буде виглядати як «{ xml: XMLData, text:

textData }» при запиті XML документа; опціональний

headers

– рядок містить заголовки відповіді сервера, ну якщо звичайно транспорт може їх отримати (ось наприклад метод «XMLHttpRequest.getAllResponseHeaders()» це подужає); опціональний

Як і префільтри, транспорт можна прив'язувати до певного типу запитуваних даних:

$.ajaxTransport( "script", function( options, originalOptions, jqXHR ) {

*/\* прив'язуємося лише до script\*/*

});

А тепер мега-напряг - приклад транспорту «image»:

$.ajaxTransport( "image", function( options ) {

if (options.type === "GET" && options.async ) { var image;

return {

send: function( \_ , callback ) {

image = new Image();

function done( status ) { *//* *підготуємо*

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*

Робочий приклад ви зможете знайти в файлі ajax.transport.html, але я хотів би ще раз нагадати, що це «advanced level», і цей розділ зайвий в підручнику «для початківців».

Слідами офіційної документації:

— «Extending Ajax: Prefilters, Converters, and Transports»

[http://api.jquery.com/extending-ajax/]

**80% Об'єкт Deferred і побратими**

* *jQuery версії 3.x, Deferred об'єкт став сумісний з Promise з ES-2015 (т.зв. ES6), так що практично все, що відноситься до Promise вірно і для Deferred.*

Робота з об'єктом «Deferred» це вже вищий пілотаж, це «mad skills» змушувати асинхронний JavaScript працювати так, як нам хочеться. Давайте подивимося як він працює (даний код можна скопіювати в консоль і виконати на будь-якій сторінці, де підключений jQuery 3.x):

* *ініціалізація Deferred об'єкта*
* *статус «очікує виконання»*

var D = $.Deferred();

* *підключаємо обробники*

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

* *змінюємо статус на «fulfilled» - «виконаний успішно»*
* *для цього викликаємо resolve()*
* *наші обробники будуть викликані в порядку черги,*
* *але вони не чекають один одного*

D.resolve();

* *даний обробник підключений занадто пізно, і буде викликаний відразу*

D.then(function() { console.log("third") });

Якщо все це перевести на людську мову, то вийде наступний сценарій:

* якщо все буде добре, тоді виконай ось цю функцію і виведи «first»
* і ще ось цю функцію — «second»

resolve() — ми дізналися, все добре

якщо все добре, виконуємо функцію і виводимо «third»

Крім сценаріїв з «happy end», є ще й сумні історії, коли все пішло не так як нам би хотілося:

* *инициализация Deferred объекта* var D = $.Deferred();
* *підключаємо обробники*

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

* *змінюємо статус на «rejected» - «виконаний з помилкою»*

D.reject();

* *в консолі нас чекатиме лише «fail» :(*

Виходить так:

— якщо все буде добре, тоді виконай ось цю функцію — «done»

— якщо все буде погано, тоді ось ця функція виведе «fail»

— ой, все погано

Насправді метод «.then()», дозволяє вішати одночасно як обробники для позитивного сценарію, так і для варіанту з помилкою:

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

Чи стає від цього код читабельним - сумніваюся, але такий варіант існує і використовується повсюдно, я ж вважаю за краще для вилову помилок використовувати метод «.catch()», який за своєю суттю лише скорочений запис для «.then(null, fn)»:

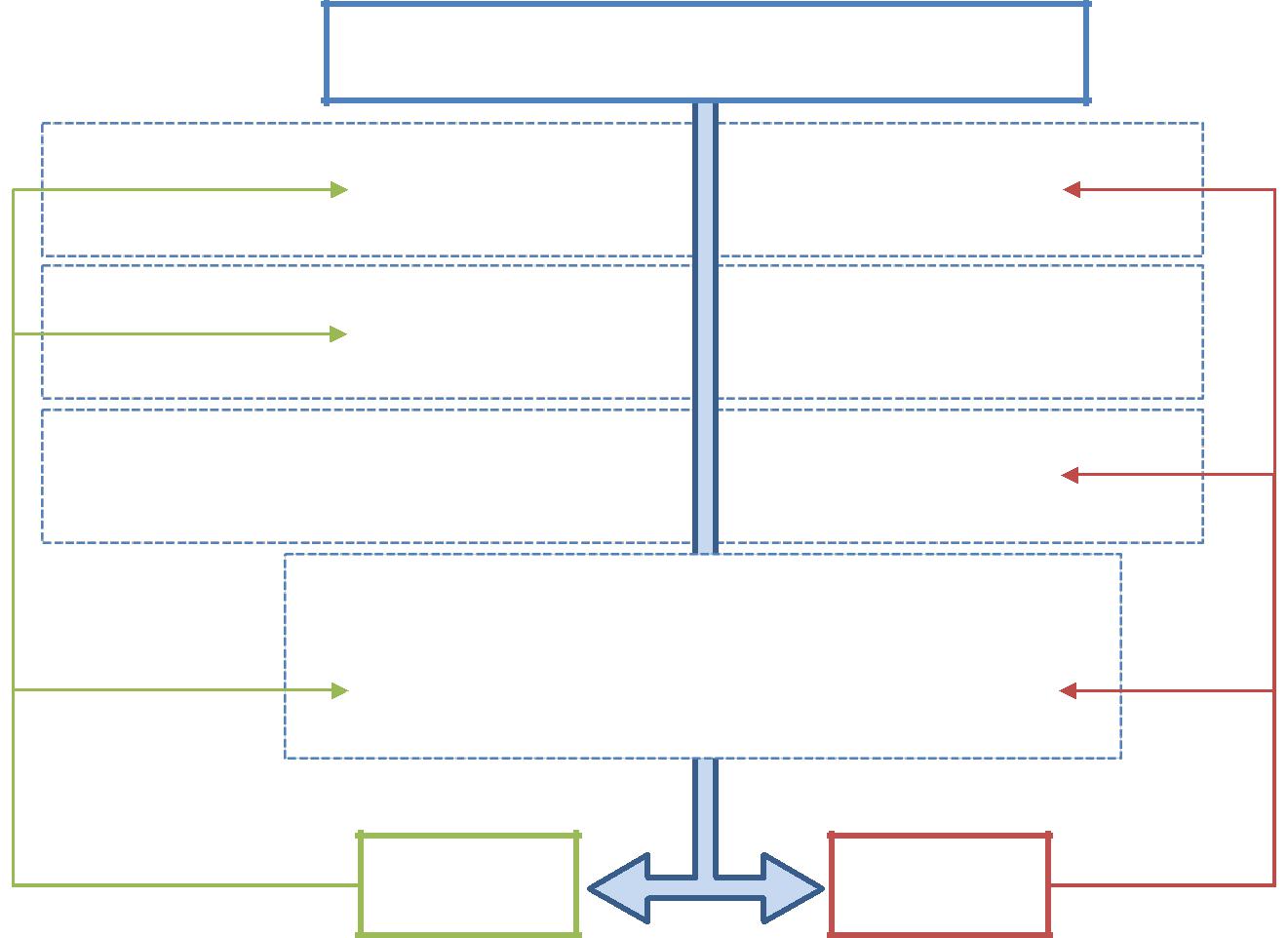
var D = $.Deferred();

* *підключити обробник помилок через then()*
* *підключити обробник помилок через catch()*

D.catch(function() { console.log("again fail") });

Ще згадаю метод «.always()» – він додає обробники, які будуть виконані незалежно від того, що сталося (у дейсвтітельності, всередині відбувається виклик «.done(arguments)» та «.fail(arguments)»).

Щоб не плутатися в перерахованих методах приведу блок-схему:



$.Deferred(…)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *then(* | callback1() |  | callback2() | *)* |  |
|  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
| *then(* | callback3() | *)* | | |  |
|  |  |  |  |  |
|  |  |  |  |  |  |
| *catch(* | |  |  |  |  |
|  |  | *)* |  |
|  |  |  | callback4() |  |  |
|  |  |  |  |  |  |
|  | always(callback5) | | | |  |
|  |  |  |  |  |  |
|  | callback5() |  | callback5() |  |  |
|  |  |  |  |  |  |

|  |  |
| --- | --- |
| fulfilled | rejected |

resolve() reject()

При виклику «.resolve()» и «.reject()» можна передати довільні дані в зареєстровані callback-функції для подальшої роботи. Крім того, існують ще методи «.resolveWith()» та «.rejectWith()», вони дозволяють змінювати контекст викликаються callback-функції (тобто всередині них «this» буде дивитися на вказаний контекст).

Окремо хотів відзначити, що якщо ви збираєтеся передати Deferred об'єкт

«На сторону», щоб «там» могли повісити свої обробники подій, але не хочете втратити контроль, то повертайте сам об'єкт, а результат виконання методу «.promise()» – це фактично буде шуканий об'єкт в режимі «read only».

* ще, крім поведінки «чекаємо дива», за допомогою Deferred можна вибудовувати ланцюжки викликів - «живі черги»:

$.ajax('ajax/example.json')

.then(function(){

* *почекаємо закінчення AJAX запиту*

return $('article img').slideUp(2000).promise()

})

.then(function(){

* *почекаємо поки сховаються картинки*

return $('article p').slideUp(2000).promise()

})

.then(function(){

* *подождём пока спрячутся параграфы*

return $('article').hide(2000).promise()

})

.then(function(){

* *Все зроблено шеф*

*Таку поведінку можна відтворити використовуючи лише animate, але нам же хочеться зазирнути трохи глибше — deferred.pipe.html (До версії 1.8 тут йшлося про метод «.pipe()», а тепер про «.then()»)*

* даному прикладі ми викликаємо метод «.then()», якому згодована callback-функція, яка повинна повертати об'єкт Promise, це необхідно для дотримання порядку в черзі - спробуйте прибрати в прикладі один «return», і ви помітите, що наступна анімація настане не дочекавшись завершення попередньої.

На цьому можливості Deferred ще не завершилися, є ще зв'язка методів «.notify()» та «.progress()» – перший шле послання в callback-функції, які зареєстровані за допомогою другого. Наведу наочний код для демонстрації (копі-паст в консоль, і дивіться що виходить):

var D = $.Deferred();

var money = 100; *//* *наший бюджет*

* *знімання готівки*

D.progress(function($){

console.log(money + " - " + $ + " = " + (money-$));

money -= $;

if (money < 0) { *//* *гроші закінчились*

D.reject();

}

});

* *тратимо гроші*

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") });

*Випробуйте всю міць Deferred в прикладі 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, що відбувається - AJAX запит і анімація стартують одночасно, коли і той і інший завершать свою роботу, буде викликана функція, яку ми передаємо

* якості аргументу в метод «.then()» (Одна з двох, залежно від результату, що відбувається). Для забезпечення роботи цієї «магії» методи «$.when()», «$.ajax()» та «.animate()» реалізують інтерфейс Deferred. Приклад роботи на сторінці when.html

*Тепер можна і незрозумілою - метод «.when ()» повертає проекцію Deferred об'єкта, приймає в якості параметрів довільне безліч Deferred об'єктів, коли все з них відпрацюють, об'єкт*

*«When» змінить свій стан в «виконано», з подальшим викликом всіх тих, хто підписався .*

**Callbacks**

Callbacks - це крутий об'єкт - він дозволяє складати списки функцій зворотного виклику, а також дає кермо влади над ними. Працювати з ним простіше ніж з Deferred, тут немає поділу на позитивний і негативний сценарії, лише стек функцій, який буде виконаний по команді «.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 чому сила, брат?*

*— В аргументах*

За замовчуванням, ви можете прямо з консолі викликати метод «.fire()» знову і знову, і будете отримувати один і той же результат раз по раз. А можна задати поведінку Callbacks через прапори:

once — всі функції будуть викликані раз (аналогічно як в об'єкті Deferred).

memory — зберігати значення з останнього виклику «.fire()», і згодовувати його в ново-зареєстровані функції зворотного виклику, і лише потім обробляє нове значення (в Deferred саме так).

unique — список функцій зворотного виклику фільтрується по унікальності

stopOnFalse — як тільки якась нитка функція поверне «false», процес запуску зупиниться

Напевно, буде краще з прикладами, ось «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 дасть результату,* *тільки* *Go*

>>>

Go first

Go second

З «memory» складніше, будьте уважніше:

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 прапора,* *цього рядка не було б* Again first Again second

Приклад з унікальністю простий до неподобства:

var C = $.Callbacks("unique");

var func = function(msg) {

console.log(msg+" first")

};

C.add(func);

C.add(func); *//* *цей рядок не вплине на результат* C.fire("Go"); *//* *тільки* *Go first*

>>>

Go first

Прапор «stopOnFalse»:

var C = $.Callbacks("stopOnFalse");

C.add(function(msg) {

console.log(msg+" first");

return false; *//* *ось він* *-* *фатальний* *false* });

C.add(function(msg) { console.log(msg+" second") }); C.fire("Go"); *//* *тільки* *Go first*

>>>

Go first

Перераховані прапори можна комбінувати і отримувати цікаві результати, а можна не отримувати, а лише подивитися на приклад callbacks.html

*З історії: об'єкт «Deferred» отпочковался від методу «$.ajax() »В*

*результаті рефакторінга версії 1.5. Йшов час, з'являлися нові версії jQuery, і ось новий виток рефакторінга - результатом стало*

*відділення «Callbacks» від «Deferred» у версії 1.7, таким чином в*

*поточній версії бібліотеки метод «$.ajax()» працює з об'єктом*

*«Deferred», який є надбудовою над «Callbacks». Щоб не вносити*

*плутанину в термінологію, я використовую визначення «Deferred*

*Callbacks» і при роботі з «Callbacks», бо колбеков багато, і кожен раз*

*уточнювати, що я говорю саме «про той самий» - справа досить*

*обтяжлива.*

Статті на цю тему:

— «Що таке цей новий jQuery.Callbacks Object»

[http://habrahabr.ru/post/135821/]

— «jQuery Deferred Object (докладний опис)»

[http://habrahabr.ru/post/113073/]

— «Async JS: The Power of $.deffered»

[http://www.html5rocks.com/en/tutorials/async/deferred/]

**90% Пишемо свій плагін**

**jQuery плагін**

Для початку згадаємо, для чого нам потрібні плагіни? Моя відповідь - створення повторно використовуваного коду, і так - зі зручним інтерфейсом. Давайте напишемо такий код, ось проста задачка: «По кліку на параграф, текст повинен змінитися на червоний»

**JavaScript і даже не jQuery**

Щоб не забувати витоків - почнемо з реалізації на нативном JavaScript'е:

var loader = function () {

* *знаходимо все параграфи*

var para = document.getElementsByTagName('P');

* *перебираємо все, і вішаємо обробник*

for (var i=0,size=para.length;i<size;i++) {

* *обрабник*

para[i].onclick = function() {

this.style.color = "#FF0000";

}

}

}

* *природно, весь код повинен працювати після завантаження всієї сторінки* document.addEventListener("DOMContentLoaded", loader, false);

*Даний код не є кросбраузерності, і написаний з метою зайвий раз підкреслити зручність використання бібліотеки;)*

**jQuery, но еще не плагин**

Тепер можна цей код спростити, підключаємо jQuery і отримуємо наступний варіант:

$(function(){

$('p').click(function(){

$(this).css('color', '#ff0000');

})

});

**Таки jQuery плагін**

* поставленим завданням ми впоралися, але де тут повторне використання коду? Або якщо нам треба не в червоний, а в зелений перефарбувати? Ось тут починається найцікавіше, щоб написати простий плагін досить розширити об'єкт «$.fn»:

$.fn.mySimplePlugin = function() {

$(this).click(function(){

$(this).css('color', '#ff0000');

})

}

Якщо ж писати більш грамотно, то нам необхідно обмежити змінну $ тільки нашим плагіном, а так само повертати «this», щоб можна було використовувати ланцюжка викликів (т.зв. «chaining»), робиться це в такий спосіб:

(function($) {

$.fn.mySimplePlugin = function(){

* *код плагіну* return this;

};

})(jQuery);

*Внесу невелике пояснення про відбувається тут «магії», код «(function($){…})(jQuery)» створює анонімну функцію, і тут же викликає її, передаючи в якості параметра об'єкт jQuery, таким чином всередині анонімної функції ми можемо використовувати алиас $ не боячись за конфлікти з іншими бібліотеками - так як тепер $ знаходиться лише в області видимості нашої функції, і ми маємо повний контроль над нею. Якщо у вас виникло відчуття дежавю - то все вірно, я про це вже розповідав*

Додамо опцію по вибору кольору і отримаємо робочий плагін (див.plugin.global.html):

(function($) {

* *значення за замовчуванням - ЗЕЛЕНИЙ* var defaults = { color:'green' };
* *актуальні настройки, глобальні*

var options;

$.fn.mySimplePlugin = function(params){

* *при багаторазовому виклику налаштування будуть збережуться*
* *і заміщатися при необхідності*

options = $.extend({}, defaults, options, params); $(this).click(function(){

$(this).css('color', options.color);

});

return this;

};

})(jQuery);

Виклик:

* *перший виклик*

$('p:first,p:last').mySimplePlugin();

* *другий виклик*

$('p:eq(1)').mySimplePlugin({ color: 'red' });

* результаті роботи даного плагіна, кожен клік буде змінювати колір параграфа на червоний, тому що ми використовуємо глобальну змінну для зберігання налаштувань, то другий виклик плагіна змінять значення для всіх елементів.

Можна внести невеликі зміни, і розділити настройки для кожного виклику

(див. plugin.html):

* *актуальні настройки, будуть індивідуальними при кожному запуску* var options = $.extend({}, defaults, params);
* *різниця то в одному «var». Мені навіть складно собі уявити як багато годин вбито в пошуках втраченого «var» в JavaScript'е, будьте уважні*

**Працюємо з колекціями об'єктів**

Тут все просто, достатньо запам'ятати — «this» містить jQuery об'єкт з колекцією всіх елементів, тобто .:

$.fn.mySimplePlugin = function(){

console.log(this); *//* *це* *jQuery* *об'єкт*

console.log(this.length); *//* *число*

*елементів у вибірці*};

Якщо ми хочемо обробляти кожен елемент то зробимо наступну конструкцію усередині нашого плагіна:

* *необхідно обробити кожен елемент в колекції* return this.each(function(){

$(this).click(function(){ $(this).css('color', options.color);

});

});

* *попередній варіант трохи надлишковий,*
* *тому всередині функції click і так є перебір елементів* return this.click(function(){

$(this).css('color', options.color);

});

*Знову ж нагадаю, якщо ваш плагін не повинен щось повертати за вашою задумом - повертайте «this» - ланцюжки викликів в jQuery це частина магії, не варто її руйнувати. методи «.each()» та «.click()» повертають об'єкт jQuery.*

**Публічні методи**

Так, у нас написаний крутий плагін, треба б йому ще докрутити функціоналу, нехай колір регулюється кількома кнопками на сайті. Для цього нам знадобиться якийсь метод «color», який і буде у відповіді за все. Зараз приведу приклад коду готового плагіна - будемо палити разом (звертайте увагу на коментарі):

* *настройки зі значенням за замовчуванням* var defaults = { color:'green' };
* *наші майбутні публічні методи*

var methods = {

* *ініціалізація плагіна* init: function(params) {
  + *настройки, будуть індивідуальними при кожному запуску* var options = $.extend({}, defaults, params);
  + *инициализируем лише одного разу*

if (!this.data('mySimplePlugin')) {

* *закинемо настройки в реєстр data* this.data('mySimplePlugin', options);
* *додамо подій*

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

});

}

return this;

},

* *змінюємо колір в реєстрі* color: function(color) {

var options = $(this).data('mySimplePlugin'); options.color = color; $(this).data('mySimplePlugin', options);

},

* *скидання кольору елементів*

reset: function() {

$(this).css('color', 'black');

}

};

$.fn.mySimplePlugin = function(method){

* *трохи магії*

if ( methods[method] ) {

* *якщо запитуваний метод існує, ми його викликаємо*
* *всі параметри, крім імені методу прийдуть в метод*
* *this так само перекочує в метод*

return methods[ method ].apply( this, Array.prototype.slice.call( arguments, 1 ));

} else if ( typeof method === 'object' || ! method ) {

* + *якщо першим параметром йде об'єкт, або зовсім порожньо*
  + *виконуємо метод init*

return methods.init.apply( this, arguments );

} else {

* + *якщо нічого не вийшло*

$.error('Метод "' + method + '" в плагине не найден');

}

};

Теперь еще небольшой пример использование данных методов:

* *виклик без параметрів - буде викликаний init* $('p').mySimplePlugin();
* *виклик методу color і передача кольору в якості параметрів* $('p').mySimplePlugin('color', '#FFFF00');
* *виклик методу reset*

$('p').mySimplePlugin('reset');

*Для розуміння даного шматочка коду, ви повинні розібратися лише зі змінною «arguments», і з методом «apply ()». Тут їм цілі статті присвятили, дерзайте:*

— *http://www.seifi.org/?p=673*

— *https://learn.javascript.ru/arguments-pseudoarray*

— *https://learn.javascript.ru/call-apply*

**Про обробниках подій**

Якщо ваш плагін вішає будь-якої обробник, то краще за все (читай завжди)

даний обробник повісити в своєму власному namespace:

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

});

Даний фінт дозволить в будь-який момент забрати всі ваші обробники, або викликати тільки ваш, що дуже зручно:

* *викличемо лише наш обробник*

$('p').trigger("click.mySimplePlugin");

* *прибираємо всі наші обробники* $('p').off(".mySimplePlugin");

*Дежавю? Ок!*

На цьому про звичайні плагинах все, хоча дам ще трохи інформації для роздумів, але англійською:

— «Essential jQuery Plugin Patterns»

[http://coding.smashingmagazine.com/?p=115389]

**Data**

Якщо з якоїсь причини ви ще не знайомі з «.data()» — то раджу прочитати документацію і засвоїти негайно. Якщо ж в двох словах - це реєстр даних, і всі дані прив'язані до якого-небудь елементу краще зберігати в ньому, це ж правило стосується і плагінів. Якщо вам треба зберегти стан плагіна - використовуйте «.data()», якщо необхідний кеш - використовуйте «.data()», якщо вам необхідно зберегти ... ну думаю зрозуміло. Наведу ще прімерчік пов'язаний з ініціалізацією:

function() { *// функція 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);

});

}

}

За збігом обставин, в HTML5 з'явилися data-атрибути, і для доступу до них jQuery використовує той же метод «.data()», але ось справи, «jQuery.data()»

– НЕ маніпулює атрибутами HTML, а працює зі своїм реєстром, і лише за відсутності там даних намагається роздобути атрибут «data-\*», не попадіться:

<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**

*Інформація в даному розділі актуальна для jQuery версії 1.8 і вище, якщо вас зацікавлять можливості розширення для більш старих версій, то читайте мою статтю «Пишемо плагіни анімації»*

Для початку запал - метод «.animate()» маніпулює об'єктом «jQuery.Animation»,який передбачає наступні точки для розширення функціоналу:

— jQuery.Tween.propHooks

— jQuery.Animation.preFilter

— jQuery.Animation.tweener

Почну розповідь з «jQuery.Tween.propHooks», тому вже є плагіни, в код яких можна зазирнути :) Для більшої наочності я візьму досить тривіальну завдання

* змусимо плавно змінити колір шрифту для заданого набору елементів:

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

Наведений вище код не дасть ніякого ефекту, тому що властивість «color» бібліотека з коробки не анімує, але це можна виправити - треба лише прокачати «jQuery.Tween.propHooks»:

$.Tween.propHooks.color = {

get: function(tween) {

return *tween.elem.style.color;*

}

set: function(tween) {

tween.easing; *// поточний easing*

tween.elem; *// випробуваний елемент*

tween.options; *//* *настройки анімація*

tween.pos; *// поточний прогрес*

tween.prop; *// властивість яку змінюємо*

tween.start; *// початкове значення*

tween.now; *// поточне значення*

tween.end; *// бажане результуючі значення*

tween.unit; *// одиниці виміру*

}

Цей код ще не працює, це наші цеглинки, з їх допомогою будемо будувати нашу анімацію. Перед роботою варто заглянути всередину кожного з наведених властивостей:

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"

* консолі у нас буде дуже багато даних, тому що наведений метод викликається N к-ть разів, в залежності від тривалості анімації, при цьому «tween.pos» поступово нарощує своє значення з 0 до 1. За замовчуванням, нарощування відбувається лінійно, якщо треба якось інакше - то варто подивитися на easing плагін або дочитати розділ до кінця (про це я вже згадував в розділі Анімація)

Навіть при такому розкладі ми вже можемо змінювати обраний елемент (шляхом маніпуляцій над «tween.elem»), але є більш зручний спосіб - можна встановити властивість «run» об'єкта «tween»:

$.Tween.propHooks.color = {

set: function(tween) {

* *тут буде ініціалізація* tween.run = function(progress) {
  + *тут код відповідає за зміну властивостей елемента*

}

}

}

Одержаний код буде працювати наступним чином:

1. раз буде викликана функція «set»
2. функція «run()» буде викликана N-раз, при цьому «progress» буде себе вести аналогічно «tween.pos»

Тепер, повертаючись до початкової завдання щодо зміни кольору можна навернути наступний код:

$.Tween.propHooks.color = {

set: function(tween) {

* *наводимо початкове і кінцеве значення до єдиного формату*
* *# FF0000 == [255,0,0]*

tween.start = parseColor(tween.start);

tween.end = parseColor(tween.end);

tween.run = function(progress) {

tween.elem.style['color'] =

* *обчислюємо проміжне значення*

buildColor(tween.start, tween.end, progress);

}

}

}

*Код функцій «parseColor ()» і «buildColor ()» ви знайдете в лістингу на сторінці color.html*

Результатом стане плавне перетікання початкового кольору до червоного (#F00

* #FF0000 == 255,0,0), вживу можна подивитися на сторінці color.html
  + *плагіні jQuery Color для вирішення поставленого завдання використовували jQuery.cssHooks, але ми ж не шукаємо легких шляхів.*

Ще хотів було розповісти про префільтри анімації, але - документації немає, а як використовувати «в життя» - я не здогадався, але трохи інформації таки накопав (код можна знайти в функції «Animation»):

jQuery.Animation.prefilter(function(element, props, opts) {

* *deferred об'єкт animate* element; *//* *шуканий елемент*

props; *//* *настройки анімації з* *animate* *()*

opts;*// опції анимації*

* *відключити анімацію при спробі анімувати висоту елемента* if (props['height'] != undefined) {

return this;

}

});

*Приклад можна побачити animate.prefilter.html*

Про «jQuery.Animation.tweener» так само багато не розкажеш, але приклад вийшло зробити трохи цікавіше - приведений код дозволяє анімувати ширину і висоту об'єкта за заданою діагоналі:

*Обережно, для розуміння того, що відбувається потрібні знання геометрії за 8-ий клас*

* *створюємо підтримку нового властивості для анімації – diagonal* jQuery.Animation.tweener( "diagonal", function( property, value ) {
  + *створюємо tween об'єкт*

var tween = this.createTween( property, value );

* *проміжні обчислення і дані*

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) {

* *обчислення шуканого значення - нове значення гіпотенузи* var hyp = this.start + ((this.end - this.start) \* progress);
* *безпосередньо измение властивостей елемента* tween.elem.style.width = sinA\*hyp + tween.unit; *//* *ширина* tween.elem.style.height = sinB\*hyp + tween.unit; *//* *висота*

};

return tween;

});

*Приклади роботи animate.tweener.html*

**Easing**

Тепер знову звернемося до easing'у - приведу приклад довільної функції, якій буде слідувати анімація. Щоб особливо не фантазувати - я взяв приклад зі статті на всюдисущого Хабре o анімації в MooTools фреймворку [http://habrahabr.ru/post/43379/] – наочний приклад з серцебиттям, яке описується такими функціями:

𝑥$ ∗ 49.4 𝑥 < 0.3 9𝑥 − 2.3 𝑥 < 0.4

−13𝑥 + 6.5 𝑥 < 0.5

4𝑥 − 2 𝑥 < 0.6

𝑦 = 0.4 𝑥 < 0.7

4𝑥 − 2.4 𝑥 < 0.75

−4𝑥 + 3.6 𝑥 < 0.8

1 − sin (acos (𝑥)) 𝑥 ≥ 0.8

У розширенні функціоналу easing немає нічого військового:

$.extend($.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 пояснень, конструкція «$.extend({}, {})» «Змішує» об'єкти:

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

>>>

{

name:"Anton",

location:"Kharkiv"

}

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

>>>

{

name:"Anton",

location:"Kyiv"

}

Таким чином ми «втручаємося» новий метод до існуючого об'єкту «$.easing»; згідно з кодом, наш метод приймає як параметр лише одне значення:

x – коефіцієнт проходження анімації, змінюється від 0 до 1, дробове

Результат звичайно цікавий, але його можна ще трохи розширити додатковими функціями (розгорнемо і скомбініруем):

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);

}

Отримаємо наступні похідні функції:



**heartIn** **heartOut** **heartInOut**

Працювати з даними творінням треба в такий спосіб:

$("#my").animate({height:"+200px"}, 2000, "heartIn"); *//* *ось воно*

Приклад роботи даної функції можна побачити на сторінці easing.html

**Sizzle**

Коли я розказував про Sizzle я вирішив вас не вантажити можливостями щодо розширення бібліотеки, але ось час настав ... В Sizzle можна розширювати багато чого:

— Sizzle.selectors.match

— Sizzle.selectors.find

— Sizzle.selectors.filter

— Sizzle.selectors.attrHandle

— Sizzle.selectors.pseudos

Але братися ми будемо лише за розширення псевдо-селектор, на зразок:

$("div:animated"); *//* *пошук анімованих елементів*

$("div:hidden"); *//* *пошук прихованих елементів* *div* $("div:visible"); *//* *пошук видимих елементів* *div*

Чому я навів тільки ці фільтри? Все просто - тільки вони не входять в Sizzle, і відносяться лише до jQuery, саме такі плагіни ми будемо тренуватися розробляти. Почнемо з коду фільтра «:visible»:

* *приклад для розширення Sizzle всередині jQuery*
* *для розширення самого Sizzle потрібен трохи інший код* jQuery.expr.pseudos.visible = function( elem ) {
  + *перевіряємо ширину і висоту кожного елемента у вибірці* return !!( elem.offsetWidth || elem.offsetHeight);

};

Виглядає цей код нескладно, але, мабуть, я таки дам каркас для нового фільтра і додам трохи пояснень:

$.extend($.expr.pseudos, {

*/\*\**

*\* @param element* *DOM елемент*

*\* @param i* *поряковий номер елементу*

* *@param matchоб'єкт матчінга регулярного виразу*
* *@param elements масив всіх знайдених DOM елементів*

*\*/*

test: function(element, i, match, elements) {

*/\* тут буде наш код, і буде вирішувати хто винен \*/* return true || false; *//* *виносимо вердикт*

}

})

Ну тепер спробуємо вирішити таку завдання:

*— Необхідно виділити посилання в тексті в залежності від її типу: зовнішня, внутрішня або якір*

Для вирішення найкраще підійшли б фільтри для селекторів такого вигляду:

$("a:internal");

$("a:anchor");

$("a:external");

Оскільки «з коробки» даний функціонал буде доступний ще ми напишемо його самі, для цього нам знадобиться не так вже й багато (приклад лише для останнього «:external», робочий код на сторінці sizzle.filter.html):

$.extend($.expr.pseudos, {

* *визначення зовнішнього посилання*
* *нам знадобиться лише DOM Element* external: function(element) {
  + *а у нас посиланняа?*

if (element.tagName.toUpperCase() != 'A') return false;

* *чи є атрибут href*

if (element.getAttribute('href')) {

var href = element.getAttribute('href');

* *відсікаємо непотрібне*

if ((href.indexOf('/') === 0) *//* *внутрішнє посилання*

* (href.indexOf('#') === 0) *//* *якір*
  + - *наш домен по http:// или https://*
  + (href.indexOf(window.location.hostname) === 7)
  + (href.indexOf(window.location.hostname) === 8)

) {

return false; *//* *повз* } else {

return true*;* *//* *да,* *ми знайшли зовнішні посилання*

}

} else {

return false;

}

}

})

***ЗАВЖДИ використовуйте фільтр разом з HTML тегом який шукайте:***

***$("tag:filter")***

*Це один з пунктів оптимізації роботи з фільтрами jQuery, інакше ваш фільтр буде обробляти всі DOM елементи на сторінці, а це може дуже сильно позначитися на продуктивності. Якщо ж у вас кілька тегів, то пишіть вже краще так — «$("tag1:filter, tag2:filter,*

*tag3:filter")», або ще краще через виклик методу*

*«.filter()».*

— «Sizzle Documentation» — скудненько офіційна документація

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

**100% Останній розділ**

Ви думаєте, я ще чогось зможу вас навчити? Сумніваюся. Я думаю, цей розділ ви напишіть самі. І якщо вам буде не в лом, то навіть надішлете її мені на поштову скриньку Anton.Shevchuk@gmail.com

**Доповнення**

**jQuery-inlog**

Ще трохи про корисне інструментарії: є такий класний плагін - jQuery-inlog [http://prinzhorn.github.com/jquery-inlog/] – основне його призначення - дати нам трохи більше розуміння про те, що відбувається всередині самого jQuery, ось шматочок HTML:

<body>

<div class="bar">

<div class="bar">

<div id="foo"></div>

</div>

</div>

<div id="bacon"></div>

</body>

А ось і код, який його обслуговує:

$l(true);

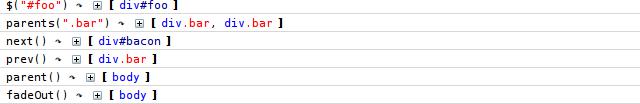
$("#foo").parents(".bar").next().prev().parent().fadeOut(); $l(false);

Якісь дивні маніпуляції, для якого ж елемента буде застосований метод

«.fadeOut()»? Для з'ясування оного наш код обгорнутий в виклик методу

«$l()». «$l()» — це і є власне виклик плагіна, результат його роботи можна

знайти в консолі:



* даного плагіна є ще настройки, які регулюють обсяг інформації, що виводиться в консоль.

*Приклад та скріншот взяти з офіційної документації стосовно плагіну*

**jQuery UI**

jQuery UI являє собою набір віджетів і плагінів від самих розробників jQuery. На мою думку, даний інструмент необхідно вивчити настільки, наскільки це потрібно щоб не писати свої

«Велосипеди». Завантажити-почитати про дану надбудові над jQuery можна на домашньої сторінки проекту – http://jqueryui.com/

Що нам необхідно знати про віджети і плагінах? Перше - це якими вони є, і друге - як працюють. На цих двох моментах я і постараюся зупинитися.

**Інтерактивність**

Почну з корисних плагінів, які можуть спростити життя при створенні інтерактивних інтерфейсів:

— Draggable [http://jqueryui.com/position/] - даний компонент дозволяє зробити будь-який DOM елемент перетягувати за допомогою миші

— Droppable [http://jqueryui.com/droppable/] – це логічне продовження draggable компонента, необхідний для роботи з контейнерами, всередину яких можна перетягувати елементи

— Resizable [http://jqueryui.com/resizable/] – як випливає з назва - дає можливість розтягувати будь DOM елементи

— Selectable [http://jqueryui.com/selectable/] – дозволяє організувати

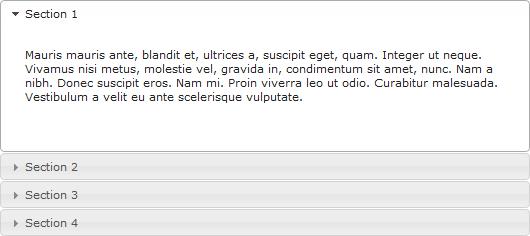
— «Вибір» елементів, зручно використовувати для організації менеджменту картинок

— Sortable [http://jqueryui.com/sortable/] – сортування DOM елементів

**Віджети**

Віджети - це вже комплексне рішення містять не тільки JavaScript код, а й певну HTML і CSS реалізацію:

— Accordion – даний віджет слід використовувати якщо у вас вже використовується jQuery UI в проекті, сам по собі основний його функціонал можна реалізувати в кілька рядків (подивитися можна в accordion.html)



— Autocomplete - як і випливає з назва, даний віджет відповідає за додавання функції автодоповнення до полів вводу, природно з підтримкою AJAX

— Button – створення кнопок використовуючи JavaScript - ще той моветон, але можливо стане в нагоді, якщо ви сильно зав'язалися на jQuery UI:

— Datepicker – якщо ваш браузер не підтримує в повній мірі специфікацію HTML5 і <input type="date"/> зокрема, то буде потрібно емуляція даної можливості за допомогою віджета:



— Dialog – віджет призначений для створення злегка незграбних діалогових вікон:

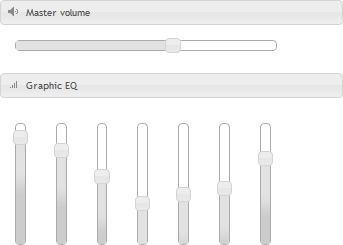


— Menu – створення меню зі списку, з підтримкою вкладеності



— Progressbar – н назва говорить сама за себе, і так в HTML5 він теж включений:

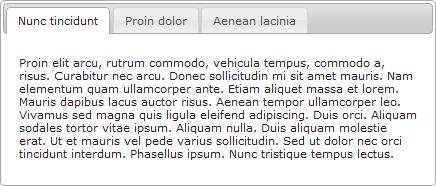
— Slider – ще один віджет для застарілих браузерів:



— Spinner – ще один зручний контрол для форм, знову ж таки - в HTML5 вже є:



— Tabs – вони ж таби - досить популярний елемент в web-розробці, і так само як і «accordion» цілком замінюємо простим кодом (див.tabs.html)



— Tooltip – ось і останній віджет - спливаючі підказки, простий і повинен бути затребуваний, ну а там життя покаже

На цьому огляд віджетів можна вважати закінченим, повернемося до плагінів.

*Всі віджети і плагіни зав'язані на ядро jQuery UI, але є так само залежності між самими плагінами і варто про них пам'ятати. Але не переживайте - при складанні jQuery UI пакету все залежності перевіряються автоматично, тобто коли вам буде потрібно досі непідключений віджет, краще скачати збірку заново.*

**Утиліти**

Утиліт у нас не багато - ось корисний плагін - position, який дозволяє контролювати стан DOM елементів – http://jqueryui.com/position/, а ще є фабрика по створенню віджетів, але про неї я розповім трохи пізніше.

**Ефекти**

Серед ефектів надаються jQuery UI я виділяю чотири пункти:

— Анімація кольору

— Анимация зміни класів

— Набір ефектів

— Разширения можливостей easing

За анімацію кольору відповідає компонент «Effects Core», який дозволяє анімувати зміни кольору за допомогою використання функції«.animate()»:

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

Так-так, jQuery з коробки не вміє цього робити, а ось jQuery UI дозволяє анімувати такі параметри:

— backgroundColor

— borderBottomColor

— borderLeftColor

— borderRightColor

— borderTopColor

— color

— outlineColor

Ще однією можливістю укладеної в «Effects Core» є анімація змін класу DOM

елемента, тобто коли ви будете привласнювати новий клас елементу, то

замість звичайного моментального застосування нових CSS властивостей ви

будете спостерігати анімацію цих властивостей від поточних до заданих в

привласнюються класи. Для використання даного функціоналу нам будуть

потрібні старі знайомі - методи «.addClass()», «.toggleClass()» та

«.removeClass()», з однією лише різницею - при виклику методу другим

параметром повинна бути вказана швидкість анімації:

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

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

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

Якщо з попереднього абзацу у вас не виникло розуміння суті того, що відбувається, то цей код для вас:

<style>

#my {

font-size:14px;

}

#my.active {

font-size:20px;

}

</style>

<script>

$(function (){

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

* *тут виходить аналогічно наступного виклику* $("#my").animate({"font-size":"20px"}, 1000);

});

</script>

* *ще з'являється метод «.switchClass()», який замінює один клас іншим, але мені він ні разу не знадобився.*

Про набір ефектів я не буду довго розповідати, їх краще подивитися в дії на

сторінці http://jqueryui.com/effect/. Для роботи з ефектами з'являється метод

«.effect()», але сам по собі його краще не використовувати, адже UI

розширив функціонал вбудованих методів «.show()», «.hide()» та

«.toggle()», тепер, передавши як параметр швидкості анімації назви ефекту

ви отримаєте необхідний результат:

$("#my").hide("puff");

$("#my").show("transfer");

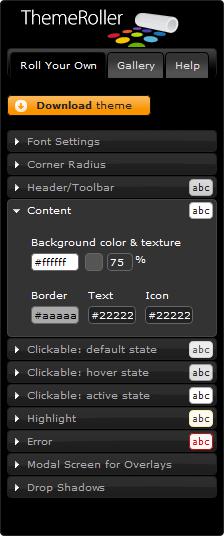
$("#my").toggle("explode");

*Наведу список ефектів, може хто запам'ятає: blind, bounce, clip, drop, explode, fold, highlight, puff, pulsate, scale, shake, size, slide, transfer.*

Пам'ятайте, я в голові про анімації розповідав про easing і однойменному плагін для jQuery? Так ось UI теж розширює easing, так що підключивши його можна відключати плагін. І так, цей функціонал зав'язаний лише на «Effects Core».

**Теми**

Однією з найбільш чудових особливостей jQuery UI є можливість міняти «шкурки» всіх віджетів разом, і для цього навіть передбачена спеціальна утиліта – ThemeRoller [http://jqueryui.com/themeroller/]:



*Якщо в якийсь момент часу буде потрібно внести зміни в тему, то відкрийте файл jquery-ui-#.#.##-custom.css і знайдете рядок починається з тексту «To view and modify this theme, visit http://...» і таки пройдіть по вказаному посиланню, і вже використовуючи ThemeRoller внесіть необхідні зміни.*

**Пишемо свій віджет**

Відправною точкою при написання віджета для jQuery UI для вас буде офіційна документація, але оскільки зі знанням англійської не у всіх склалося, то я постараюся перевести і адаптувати інформацію викладену в ній.

Перше, про що варто розповісти, це те, що правила написання плагінів для jQuery занадто вальяжно, що не сприяє їх якості. При створенні jQuery UI, походу, вирішили піти шляхом стандартизації процесу написання плагінів і віджетів, я не можу сказати наскільки задумка вдалася, але стало явно краще ніж було. Почну з опису каркаса для вашого віджета:

$.widget("book.expose", {

* *настройки за замовчуванням* options: {

color: "red"

},

* *ініціалізація widget*
* *вносимо зміни в DOM і вішаємо обробники*

\_create: function() {

this.element; *//* *шуканий об'єкт в* *jQuery* *обгортці*

this.name; *//* *ім'я* *- expose* this.namespace; *//* *простіро* *–* *book* this.element*.*on*(*"click."+this.eventNamespace, function(){

console.log("click");

})

},

* *метод відповідає за застосування налаштувань*

\_setOption: function( key, value ) {

* *застосовуємо зміни налаштувань*

this.\_super("\_setOption", key, value );

},

* *метод \_destroy повинен бути антиподом до \_create*
* *він повинен прибрати всі зміни внесені зміни в DOM*
* *і прибрати все обробники, якщо такі були*

\_destroy: function() {

this.element.off('.'+this.eventNamespace);

}

});

Поясню для тих хто не прочитав коментарі:

options – сховище налаштувань віджета для конкретного елемента

\_create() – відповідає за ініціалізацію віджета - тут повинні відбуватися зміни в DOM'е, і «вішатися» обробники подій

\_destroy() – антипод для «\_create()» – повинен підчистити все, що ми насмітили

\_setOption(key,value)– даний метод буде викликаний при спробі змінити будь-які налаштування:

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

*Наглядова око помітить, що всі перераховані методи починаються зі знака підкреслення - це такий спосіб виділити «Приватні» методи, які недоступні для запуску, і якщо ми спробуємо запустити «$('#my').expose('\_destroy')», то отримаємо помилку. Але майте на увазі - це лише домовленість, дотримуйтесь, її!*

*Для обходу домовленості про приватністьі можна використовувати*

*метод «data()»:*

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

В даному прикладі, я постарався поставити хороший тон написання віджетів - я «Повісив» обробники подій в namespace, це дасть в подальшому можливість контролювати те, що відбувається без необхідності залазити в код віджету, це «true story».

*Код описаний в методі «\_destroy()» – надлишковий, тому що він і так виконується в публічному «destroy()», наведено тут для наочності.*

* для ледачих, щоб не прописувати кожен раз «eventNamespace» в обробниках подій, розробники додали в версії 1.9.0 два методу:

«\_on()» та «\_off()», перший приймає два параметри:

* + DOM елемент, або селектор, або jQuery об'єкт

— набір обробників подій у вигляді об'єкта

Всі перераховані події будуть «висіти» в просторі «eventNamespace», тобто результат буде приблизно однаковим:

this.\_on(this.element, { mouseover:function(event) { console.log("Hello mouse");

},

mouseout:function(event) {

console.log("Bye mouse");

}

});

Інший метод – «\_off()» – дозволяє вибірково відключати обробники:

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

Ну каркас баркасом, пора переходити до функціоналу - додамо довільну функцію з довільним функціоналом:

callMe:function(){

console.log("Allo?");

}

До даної функції ми легко зможемо звертатися як з інших методів віджета так і ззовні:

* *зсерединии* this.callMe()
* *ззовні*

$("#my").expose("callMe")

Якщо ваша функція приймає параметри, то передача оних здійснюється наступним способом:

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

Якщо ви хочете достукатися в обробнику подій до методу керування, то не забудьте про область видимості змінних, і зробіть наступний маневр:

\_create: function() {

var self = this; *// ось він!*

this.element*.*on*(*"click."+this.eventNamespace, function(){

* *тут використовуємо self, тому що this вже вказує на*
* *елемент по якому натискаємо*

self.callMe();

}*)*

},

Добре йдемо, тепер поговоримо про події - для більш гнучкої розробки і впровадження віджетів передбачений функціонал по створенню довільних подій і їх «прослуховування»:

* *ініціюємо подія*

this.\_trigger("incomingCall");

* *підписуємося на подію при ініціалізації віджета* $("#my").expose({

incommingCall: function(ev) { console.log("din-don");

}

})

* *або після, використовуючи в якості імені події*
* *ім'я віджета + ім'я події*

$("#my").on("exposeincomingCall", function(){ console.log("tru-lya-lya")

});

Матеріалу багато, я розумію, але ще додам опис декількох методів які можна викликати з самого віджета:

\_delay() – дана функція працює як «setTimeout ()», ось тільки контекст переданої функції буде вказувати на сам віджет (це щоб не морочитися з областю видимості)

\_hoverable() та \_focusable() – даними методам необхідно згодовувати елементи для яких необхідно відстежувати події «hover» та

«focus», щоб автоматично додасть до них класи «ui-state-hover» та «ui-state-focus» при настанні оних

\_hide() та \_show() – ці два методи з'явилися у версії 1.9.0, вони створені щоб стандартизувати поведінка віджетів при використанні методів анімації, налаштування прийнято ховати в опціях під ключами

«hide» та «show» відповідно. Використовувати методи слід наступним чином:

options: {

hide: {

effect: "slideDown", *// настройки еквіваленти викликом*

duration: 500 *// .slideDown( 500)*

}

}

* *всередині віджета слід використовувати виклики \_hide() та \_show()* this.\_hide( this.element, this.options.hide, function() {
  + *це наша функція зворотного виклику*

console.log('спрятали');

});

Існує ще пару методів, які реалізовані за нас:

enable: function() {

return this.\_setOption( "disabled", false );

},

disable: function() {

return this.\_setOption( "disabled", true );

},

Фактично, даний функції створюють синонім для виклику:

$("#my").expose({ "disabled": true }) *//* *або* *false*

Наше завдання зводиться лише до відстеження даного прапора в методі

«\_setOption()».

Прикладом буде – widget.html, можливо цей віджет і не буде популярний, зате він наочно демонструє як створювати віджети для jQuery UI.

*Будьте уважні, з виходом jQuery UI версії 1.9.0 були внесені правки в Widget API, отже, більшість доступної інфор мації застаріло, так що*

*читайте офіційну документацію [http://wiki.jqueryui.com/w/page/12138135/Widget%20factory], а ще краще - заглядайте в код готових віджетів «від виробника»*

Інформація по темі розробки віджетів:

— «The jQuery UI Widget Factory. WAT?» – ця документації 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**

*Даний набір утиліт досить довго не оновлювався, аж з березня 2012-го року, але він ще не канув в лету, робота йде, і обіцяють випустити другу версію.*

Альтернатива jQuery UI, хоча я б назвав корисним доповненням. Бібліотека jQuery Tools складається з компонентів, які поділяються на три типи:

— UI Tools – інтерактивні компоненти інтерфейсу

— Form Tools – компоненти по роботі з формами

— Toolbox – всяке різне і корисноUI Tools

Tabs – аналогічно jQuery UI за призначенням, скромніше за функціоналом, плюс ще є можливість налаштовувати як «accordion»:



Tooltip – спливаючі підказки, я не впевнений в юзабельності оних, але може для когось цей компонент виявиться корисним, адже його так само включили

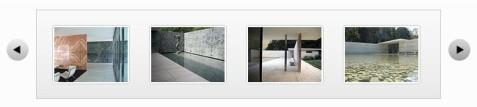
* останній jQuery UI:



Overlay – створення спливаючих модальних віконець, корисний компонент, аля «lightbox»:



Scrollable – компонент для створення «каруселі» з картинок, може стати в нагоді в будь-якому онлайн-магазині:



**Form Tools**

Validator – п озволяет на декларативному рівні задавати правила перевірки введених значень, що буває дуже корисно, і вкрай зручно (сумісний з HTML5).

RangeInput – аналогичний jQuery UI Slider DateInput – аналогичний jQuery UI Datepicker

**Toolbox**

Expose – мабуть найчудовіший компонент з усього набору - він дозволяє виділяти елемент на сторінці, шляхом затемнення інших елементів; це важко описати, краще подивіться на демо:

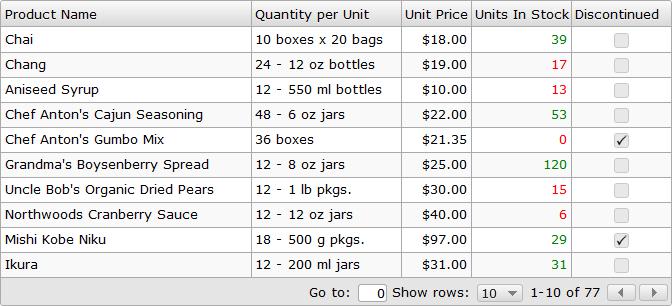
http://jquerytools.github.io/demos/toolbox/expose/index.html

На цьому, мабуть, варто закінчити цей огляд.

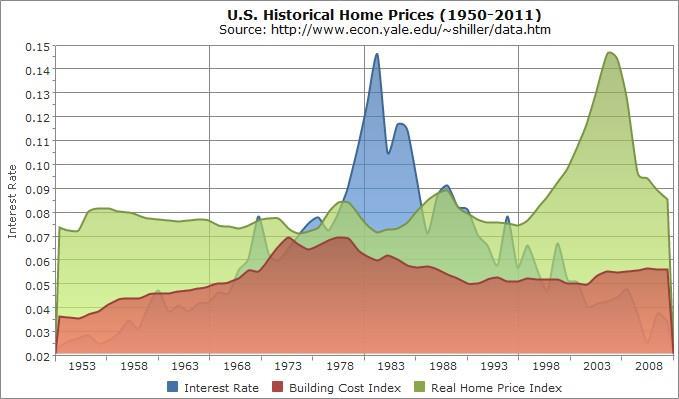
**jQWidgets**

jQWidgets – це не просто бібліотека, а цілий комбайн з купою корисних віджетів, та їх на даний момент налічується тридцять чотири штуки, тут тільки привести список заблокований на кожен - вже пів книги буде, так що обійдемося коротенько резюме на кожен віджет. Почну огляд з комплексних і «важких» віджетів, до них картинки я таки прикладу.

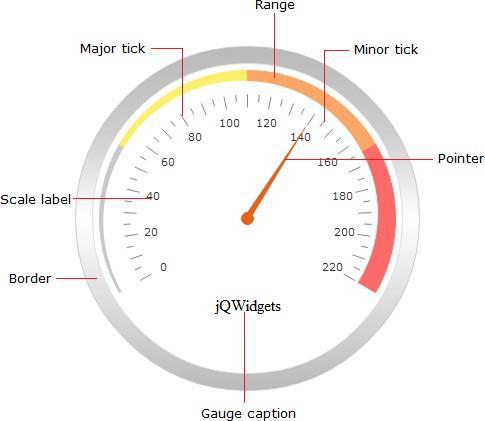
jqxGrid – це датагрід з купою корисних і не дуже примочок, тут ясна річ є посторінкова навігація, сортування, фільтрації та угруповання, тільки з цього віджету вже можна здоровенний талмуд написати:



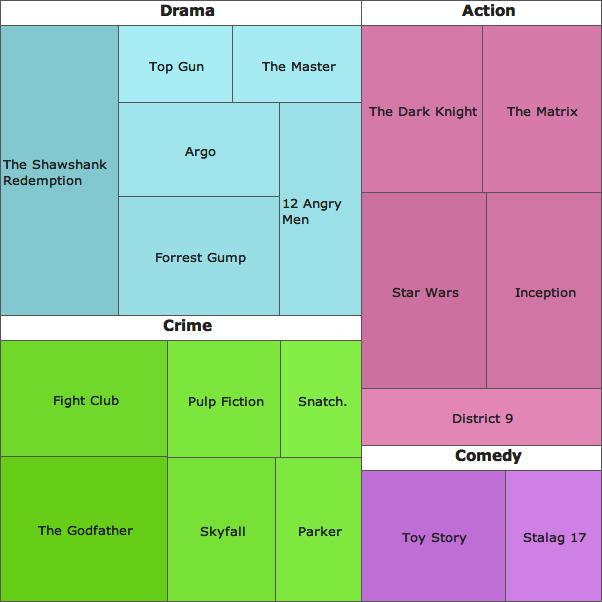
jqxChart – віджет для побудови різноманітних графіків за допомогою HTML, CSS і JavaScript, зроблено все дуже і дуже культурно, особливо виділю функцію зі збереження графіка як картинки, іноді дуже її не вистачає:



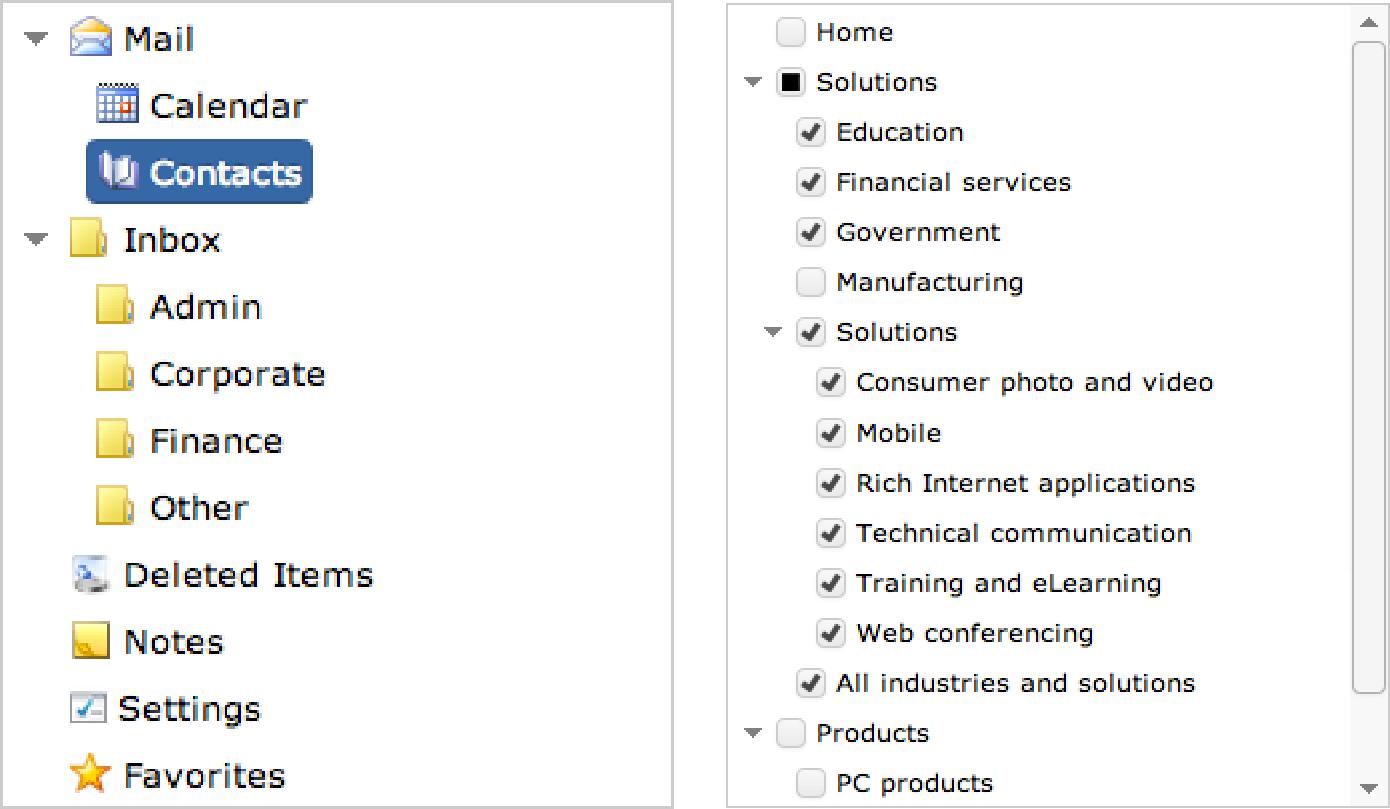
jqxGauge – цей віджет не часто зустрінеш і в навіть більш іменитих фреймворками, але по суті - це якийсь вимірювач, тобто з його допомогою можна намалювати спідометр, манометр, термометр або ще якийсь інший вимірювальний прилад з довільною шкалою:



jqxTreeMap – ще один рідкісний вид, скоріше навіть унікальний, з його допомогою можна побудувати пов'язане дерево у вигляді організованих прямоугольничков, якщо нічого не зрозуміло, то краще переглянути демку, ну і навіщо додаю:



jqxTree – це вже не настільки екзотичний віджет, як зрозуміло з назви, будемо садити дерева:



На цьому огляд «крутих» віджетів можна закінчувати, заглиблюватися в нудні і звичайний обгортки над елементами форм мені не хочеться, зауважу лише, що багато в чому даний фреймворк обходить jQuery UI, але не все так райдужно в цьому королівстві:

*Даний фреймворк поширюється під ліцензією Creative Commons Attribution-NonCommercial 3.0 License, яка передбачати кість безкоштовне використання бібліотеки для некомерційних проектів, інакше - дивіться расценки.*

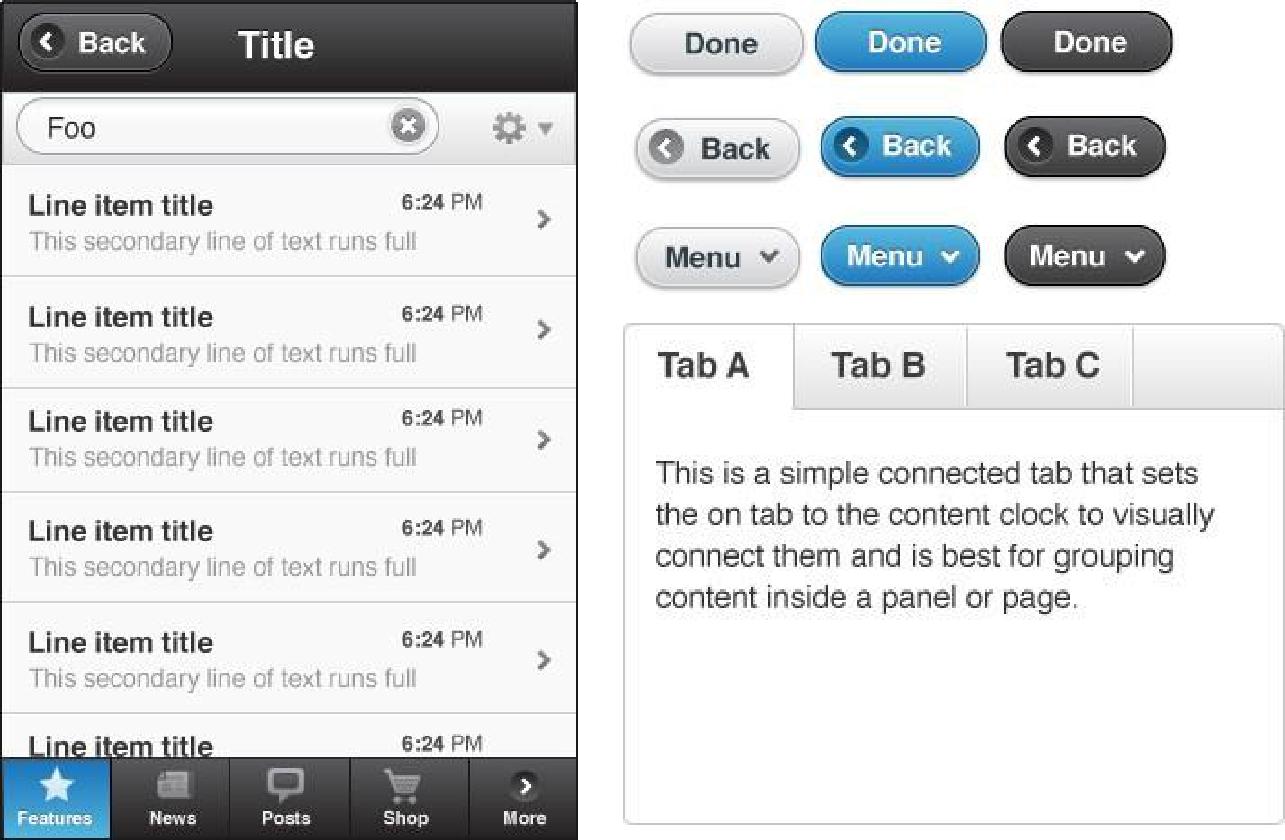
Ще варто згадати одну приємну особливість - це можливість легкої інтеграції з MVVM фреймворком Knockout, але це вже інша історія

**jQuery Mobile**

* ось це цілком самостійний продукт, і як випливає з назви призначений для створення інтерфейсів для мобільних пристроїв з підтримкою «Touch Screen». Цей фреймворк добре документований, з купою прикладів, які можна помацати на http://jquerymobile.com/

*Даний фреймворк добре підходить для створення мобільних версій сайтів, але при цьому він буде виглядати як мобільний додаток, хоча ні - сайт буде виглядати як сайт на jQuery Mobile. Наскільки це добре чи погано мені судити складно, скажімо так - це затребувано.*

* не буду розглядати всі компоненти даного фреймворка, наведу лише деякі скріншоти (взяті з офіційного сайту http://jquerymobile.com/):



*Для jQMobile існує свій власний ThemeRoller:*

*http://jquerymobile.com/themeroller/*

* ще, пораджу звернути увагу на API даного фреймворка, без вивчення оного вам буде важко створити дійсно цікаві програми.

**Ще плагіни**

Хотілося б порекомендувати ще кілька плагінів, які варто завжди тримати під рукою:

color — якщо буде потрібно анімація кольору фону, або шрифту, або ще чого-небудь (як альтернатива jQuery UI) [https://github.com/jquery/jquery-color]

cookie — зручна робота з «печеньки» в браузері [https://plugins.jquery.com/cookie/]

easing — розширюємо стандартний набір функцій easing (про це я

розповідав в розділі анимация)

[http://gsgd.co.uk/sandbox/jquery/easing/]

form — спрощує роботу з формами, сам вже давно не користуюся, але для швидкого старту саме воно [http://malsup.com/jquery/form/]

hotkeys — назва говорить сама за себе

[https://github.com/jeresig/jquery.hotkeys]

shadow animation — як випливає з назви - плагін для анімації тіней

[http://www.bitstorm.org/jquery/shadow-animation/] jQuery

Transition Events – підтримка CSS Transition из JS [https://github.com/ai/transition-events]

Redactor – просто чудовий WYSIWYG редактор, легкий, швидкий, і не безкоштовний

[http://redactorjs.com/]

**Оновлення на версію 2.х**

Чим же нам загрожує нова версія? Ну нам то бояться її не варто, а ось користувачі старих версій Internet Explorer не зможуть більше насолодитися зручністю настільки популярною бібліотеки. Так, так, саме так, з другої версії прибрали підтримку IE з 6-ї по 8-му. Завдяки цій «оптимізації» бібліотека схудла на 10%.

*Якщо більш сучасні IE працюють в режимі сумісності зі старими версіями, то jQuery НЕ буде морочитися, і відмовиться працювати. Уникнути це допоможе мета-тег X-UA-Compatible або заголовок HTTP.*

*Незабаром «під ніж» підуть і інші застарілі браузери, будьте пильні Android / WebKit 2.x, за вами вже виїхали.*

– Якщо вас не зачепила ця дієта, то можна піти далі, і зібрати jQuery тільки з того, що вам дійсно необхідно, що для цього буде потрібно

– - читайте в докладному покроковому керівництві, якому і переклад-то не потрібно.

Якщо вас не вражає цей список змінився, то це лише завдяки праці розробників jQuery, які постаралися забезпечити повну сумісність API гілок 1.х і 2.х, а ось це вже дійсно вражає.

*Ну ладно, не все так гладко з сумісністю, був цей шлях тернистий, і проблеми з оновленням будуть, ось тільки шукати причину потрібно буде в списку змін версії 1.9, саме ця версія стала поворотною в питанні сумісності гілок.*

— «How to build your own jQuery»

[https://github.com/jquery/jquery/]

— «jQuery Core 1.9 Upgrade Guide»

[http://jquery.com/upgrade-guide/1.9/]

**Оновлення на версію 3.х**

* першу чергу це оновлення торкнулося старих версій браузерів, точніше не торкнулося, тепер підтримуються наступні версії:

— Internet Explorer — підтримується з 9-ої

— Chrome, Edge, Firefox и Safari — завдяки вбудованій системі оновлень в ці браузери підтримується поточна і попередня версії браузерів

— Opera — в фаворитах не ходить, тому підтримують тільки поточну версію

— Safari Mobile iOS — c 7ой і старше

— Android — з 4ой і старше

Крім цього, було внесено багато змін, які ламають зворотну сумісність з попередніми версіями, все їх перечеслять не буду, зупинюся на тих, з якими вже довелося зіткнутися.

*Більшість наведених змін стосується не тільки версії 3.x, але старих гілок, починаючи з версій 1.12.x и 2.2.x*

**AJAX**

— Метод «jQuery.ajax()» тепер сумісний з Promise, і ви можете використовувати методи «.then()» и «.catch()»:

$.ajax({url: "/get-my-page.html" */\** *и т.д. \*/* })

.then(function() { */\** *все ОК* *\*/* })

.catch(function() { */\** *полмилка* *\*/* })

;

— - Додана нова сигнатура для виклику двох AJAX-методів

«$.get(settings)» та «$.post(settings)», тепер настройки сумісні з «$.ajax(settings)».

— - При підключенні скриптів з іншого домену тепер в обов'язковому порядку вимагає вказівки «dataType: "script"»

— При виконанні AJAX-запитів і надання URL з хешем, останній паче не обрізається, а відправляється на сервер як є.

**Атрибуты**

— Раніше метод «.removeAttr()» для true-false атрибутів, таких як «checked», «selected» та «readonly», нишком виставляв відповідне властивість DOM елемента в «false», тепер будьте ласкаві робити це ручками:

$("input[type=email]").removeAttr("readonly").prop("readonly", false)

— - Якщо викличте метод «.val()» для мультіселекта, в якому нічого не вибрано, то отримаєте у відповідь порожній масив, а раніше був «null»

— Для SVG заробили методи по маніпуляції з класами (хоча повноцінної підтримки SVG в jQuery нет).

**Core**

— Ядро jQuery тепер запускаєтся в strict mode.

— обробники події document-ready тепер запускаються асинхронно, тобто тепер якщо якийсь оброблювач сфейліл, то це не вплине на запуск інших обробників.

— Метод «$.isNumeric()» більше не намагається кастовать метод «.toString()» для довільних об'єктів (кому це треба було?).

— Методи «.width()», «.height()» і так далі, раніше викликавши їх для порожній колекції об'єктів ми отримували «null», тепер «undefined».

— - Офіційно доданий promise «jQuery.ready», який дуже зручно заюзать

разом с «$.when()»:

$.when($.ready, $.getScript("script.js") ).then(function() {

* *документ готовий, і скрипт script.js довантажуючи*

}).catch( function() {

* + *полмилка*

});

— Метод «jQuery.unique()» перейменований в «jQuery.uniqueSort()»

— Метод «jQuery.parseJSON()» застарів, переходите на «JSON.parse()»

— Внутрішнє сховище даних перейшло на camelCase, це важливо для тих, хто використовував його, не вдаючись до методу «.data()»

**Об'єкт Deferred**

Як я вже згадував раніше, об'єкт Deferred тепер сумісний з Promise з ES- 2015 року, а це нам загрожує переписуванням методів «.done()» на «.then()» та

«.fail()» на «.catch()».

Другий важливий момент - callback-функції, згідно специфікації ES-2015 року, повинні приймати тільки один аргумент: для успішного виконання це якийсь результат виконання, в разі виникнення помилки функція це буде сама помилка. Якщо у вас не виходить так зробити, то старі функції «.done()» та «.fail()» все ще залишаються з нами, хоч чую і їх скоро випив:

* *було*

$.get("/get-my-page.html")

.done(function(data, textStatus, jqXHR) { */\** *все ОК* *\*/* })

.fail(function(jqXHR, textStatus, errorThrown) { */\** *поомилка* *\*/* });

* *стало*

$.get("/get-my-page.html")

.then(function(data) { */\** *все ОК* *\*/* })

.catch(function(error) { */\** *помилка* *\*/* });

**Розміри**

Невеликі зміни спіткали функції «.width()», «.height()»,

«.css("width")», та «.css("height")», тепер вони можуть повертати не тільки integer значення висоти і ширини, а й float, така точність пов'язана з переходом на використання «getBoundingClientRect()».

Ще момент, виклик «$(window).outerWidth()» та «$(window).outerHeight()»

тепер будуть включати в себе розміри скролбаров вікна.

**Ефекти**

Анімація переїхала на використання requestAnimationFrame API, так що тепер все стало швидше і красивіше.

Функції «.show()», «.hide()» та «.toggle()» навчилися запам'ятовувати попередній стан CSS-властивості «display».

Кількість аргументів easing-функцій скоротили до одного аргументу, який містить прогрес анімації від 0 до 1.

**Подія**

Вилучені shorthand-методи для наступних подій: «.load()», «.unload()» та

«.error()», пов'язана дана зміна з конфліктами виникають при використанні даних методів, так що переписуйте на «.on()»:

* *було*

$("img").load(fn)

* *стало*

$("img").on("load", fn).

Вилучено синтетичне подія «ready», так що «.on("ready", fn)» більше не працює, використовуйте синтаксис «$(fn)».

Делеговані події, в разі якщо їх намагаються повісити c використанням неправильних селектор тепер будуть відразу лаятися і викидати помилку. Дебажіть стане легше:

* *приклад зламаного селектора div:not*

$("body").on("click", "div:not", e => false);

**Селектори**

За селектори «:hidden» та «:visible» тепер відповідає «getClientRects()»,

якщо у запитуваної елемента є layout box, значить він вважається видимим, як результат порожній <span> або <br/> тепер вважаються видимими.

криві селектори «$("#")» та «.find("#")» тепер будуть викликати помилку.

Описав багато, але не все, повне керівництво є на офіційному сайті:

— «jQuery Core 3.0 Upgrade Guide»

[https://jquery.com/upgrade-guide/3.0/]

**Історія змін**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1.0.0 | Жовтень, 2012 |  | Перша публікація «для своїх», всього 95 завантажень, | | |  |
| alpha |  |  | вичитка почалася | | |  |
|  |  |  |  |  |  |  |
| 1.0.0 |  |  | Виправлення помилок і граматичних помилок. | | |  |
| beta |  |  | Публікація на Хабре і більше 22 000 завантажень | | |  |
|  |  |  |  |
|  |  |  |  |  |  |  |
| 1.0.0 |  |  | Реліз :) Виправлено безліч помилок і кілька посилань. | | |  |
|  |  |  | Додана інформація про швидкодію селектор виду | | |  |
|  |  |  | «div#id», спасибі хабра- користувачеві skorney за | | |  |
|  |  |  | тестування. Понад 2 000 завантажень | | |  |
|  |  |  |  |  |  |  |
| 1.0.1 | Листопад, 2012 |  | Ще кілька дрібних правок. Дякую читачам за | | |  |
|  |  |  | відгуки.23 000+ | | |  |
|  |  |  |  |  |  |  |
| 1.0.2 | Січень, 2013 |  | І ще ... Опублікував в ePub форматі, виклав на scribd, | | |  |
|  |  |  | вийшло понад 10 000 завантажень і більше 10 000 | | |  |
|  |  |  | прочитань на scribd | | |  |
|  |  |  |  | | |  |
| 1.0.3 | Вересень, 2013 |  | Оновив книгу слідом за оновленням jQuery, додав | | |  |
|  |  |  | інформацію про зміни у версії 2.х щодо 1.Х. Додав | | |  |
|  |  |  | опис віджетів jQWidgets. Видалив опис подій для | | |  |
|  |  |  | методу «data ()». Приклади коду використовуються в | | |  |
|  |  |  | підручнику тепер доступні на GitHub, так що чекаю | | |  |
|  |  |  | pull-запитів: | | |  |
|  |  |  | https://github.com/AntonShevchuk/jquery-for- | | |  |
|  |  |  | beginners. | |  |  |
|  |  |  |  |  | |  |
|  |  |  | Счётчик прочтений на scribd перевалил за 20 тысяч | | |  |
|  |  |  |  | | |  |
| 1.0.4 | Травень, 2014 |  | Оптимізація шрифтів під друк формату А5. тираж! | | |  |
|  |  |  |  | | |  |
| 1.0.5 | Березень, 2015 |  | Дрібні виправлення та уточнення. ~ 22 000 | | |  |
|  |  |  |  | завантажень | |  |
| 1.0.6 | Вересень, 2016 |  | Оновлена документація з урахуванням виходу jQuery | | |  |
|  |  |  | 3.x. Додана інформація по міграції на jQuery 3.x. | | |  |
|  |  |  | Додана інформація про нововведення ES-2015. | | |  |
|  |  |  |  |  |  |  |

**Подяки**

Спасибі моїй компанії NIX Solutions Ltd за моральну і матеріальну підтримку. Дякую моїм колегам, яким першим довелося читати цю книгу і вносити коригування.

Хочу ще сказати спасибі Іллі Кантору за його відмінні майстер-класи по JavaScript'у (це не реклама, вони дійсно гарні), які допомогли мені в кращому ступені пізнати світ цього чудового мови.

Величезне спасибі моїй дружині Олі і синові Данилу за терпіння, що його вистачило не виганяти мене з-за комп'ютера і з дому.