Бесплатная электронная книга

УЧУСЬ jQuery

Free unaffiliated eBook created from
Stack Overflow contributors.

#jquery

Около

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: jquery

It is an unofficial and free jQuery ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official jQuery.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

глава 1: Начало работы с jQuery

замечания

jQuery - это библиотека JavaScript, которая упрощает операции DOM, обработку событий, AJAX и анимацию. Он также заботится о многих проблемах совместимости браузеров в базовых DOM и javascript-движках.

Каждая версия jQuery может быть загружена с https://code.jquery.com/jquery/ в сжатых (минитизированных) и несжатых форматах.

Версии

Версия	Заметки	Дата выхода
1,0	Первый стабильный выпуск	2006-08-26
1,1		2007-01-14
1.2		2007-09-10
1,3	Sizzle введен в ядро	2009-01-14
1.4		2010-01-14
1,5	Отложенное управление обратным вызовом, переписывание модуля ajax	2011-01-31
1,6	Значительный прирост производительности в методах `attr()` и `val()`	2011-05-03
1,7	API новых событий: `on()` и `off()` .	2011-11-03
1,8	Sizzle переписан, улучшена анимация и гибкость `$(html, props)` .	2012-08-09
1,9	Удаление устаревших интерфейсов и очистка кода	2013-01-15
1,10	Исправлены исправления ошибок и отличия от обоих бета-циклов 1.9 и 2.0	2013-05-24
1,11		2014-01-24
1,12		2016-01-08

Версия	Заметки	Дата выхода
2,0	Снижение поддержки IE 6-8 для повышения производительности и уменьшения размера	2013-04-18
2,1		2014-01-24
2,2		2016-01-08
3.0	Массовые ускорения для некоторых пользовательских селекторов jQuery	2016-06-09
3,1	Нет более бесшумных ошибок	2016-07-07

Examples

Пространство имен jQuery («jQuery» и «$»)

`jQuery` является отправной точкой для написания любого кода jQuery. Его можно использовать как функцию `jQuery(...)` или переменную `jQuery.foo` .

`$` является псевдонимом для `jQuery` и они обычно могут быть взаимозаменяемы друг для друга (кроме случаев, когда используется `jQuery.noConflict();` см. « Устранение конфликтов имен»).

Предполагая, что у нас есть этот фрагмент HTML -

```
<div id="demo_div" class="demo"></div>
```

Мы могли бы использовать jQuery для добавления текстового содержимого в этот div. Для этого мы могли бы использовать функцию jQuery `text()` . Это может быть написано с использованием `jQuery` или `$` . т.е. -

```
jQuery("#demo_div").text("Demo Text!");
```

Или же -

```
$("#demo_div").text("Demo Text!");
```

Оба результата приведут к тому же окончательному HTML -

```
<div id="demo_div" class="demo">Demo Text!</div>
```

Поскольку `$` является более кратким, чем `jQuery` это обычно предпочтительный метод

написания кода jQuery.

jQuery использует селектор CSS, а в примере выше использовался селектор идентификаторов. Для получения дополнительной информации о селекторах в jQuery см. Типы селекторов .

Начиная

Создайте файл `hello.html` со следующим содержимым:

```
<!DOCTYPE html>
<html>
<head>
    <title>Hello, World!</title>
</head>
<body>
    <div>
        <p id="hello">Some random text</p>
    </div>
    <script src="https://code.jquery.com/jquery-2.2.4.min.js"></script>
    <script>
        $(document).ready(function() {
            $('#hello').text('Hello, World!');
        });
    </script>
</body>
</html>
```

Живая демонстрация на JSBin

Откройте этот файл в веб-браузере. В результате вы увидите страницу с текстом: `Hello, World!`

Объяснение кода

1. Загружает библиотеку jQuery из CDN jQuery:

   ```
   <script src="https://code.jquery.com/jquery-2.2.4.min.js"></script>
   ```

   Это вводит глобальную переменную `$` , псевдоним для функции `jQuery` и пространства имен.

   > *Имейте в виду, что одна из наиболее распространенных ошибок, возникающих при включении jQuery, не загружает библиотеку перед любыми другими скриптами или библиотеками, которые могут зависеть от нее или использовать ее.*

2. Заменяет функцию, которая будет выполняться, когда DOM (Document Object Model) будет обнаружена как «готовая» jQuery:

```
// When the `document` is `ready`, execute this function `...`
$(document).ready(function() { ... });

// A commonly used shorthand version (behaves the same as the above)
$(function() { ... });
```

3. Когда DOM готов, jQuery выполняет функцию обратного вызова, показанную выше. Внутри нашей функции есть только один вызов, который делает две основные вещи:

    1. Получает элемент с атрибутом `id` равным `hello` (наш селектор `#hello`). Использование селектора в качестве переданного аргумента является ядром функциональности и именования jQuery; вся библиотека существенно эволюционировала от расширения document.querySelectorAll MDN .

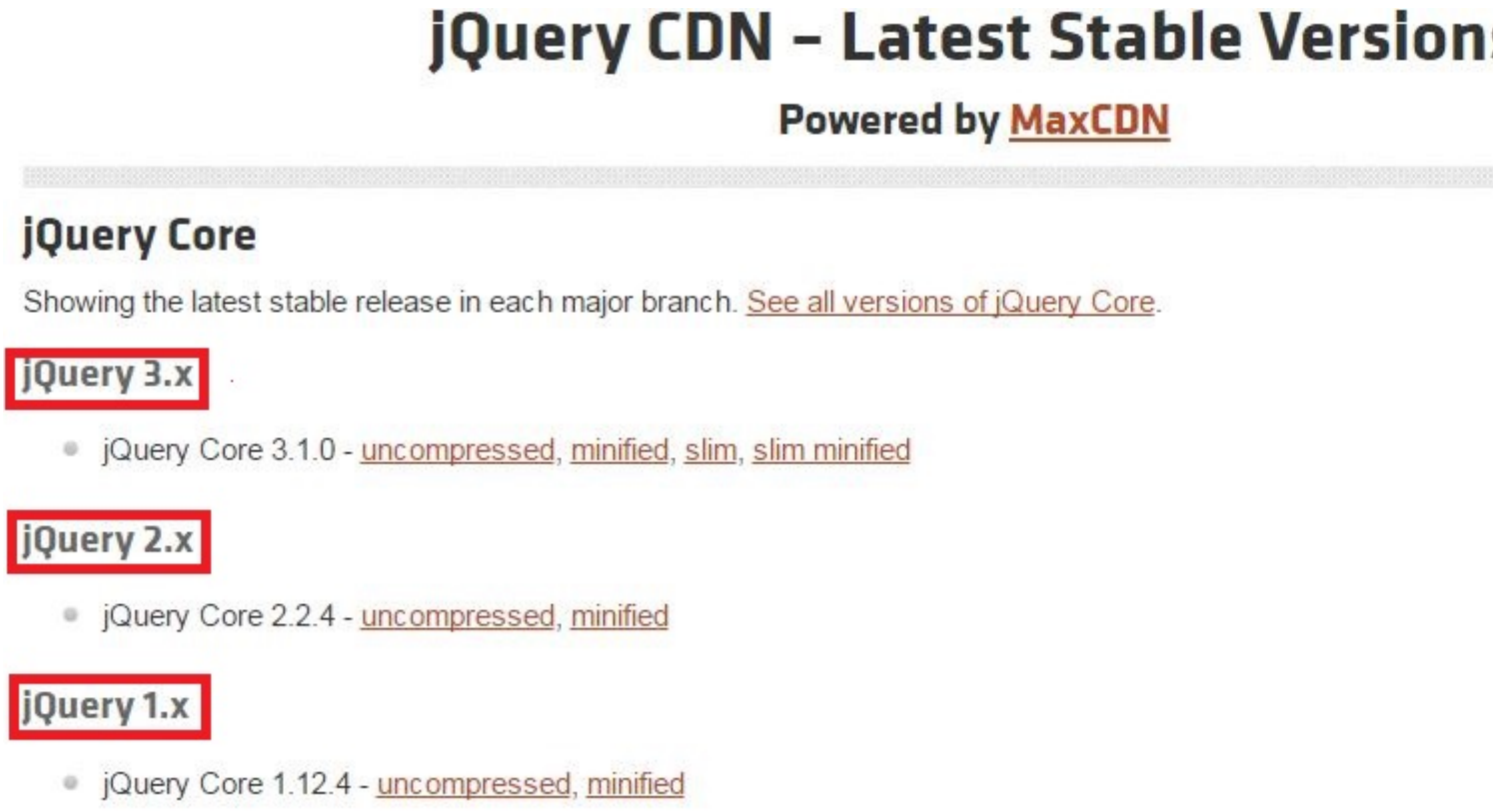
    2. Установите `text()` внутри выбранного элемента `Hello, World!` ,

```
#    ↓ - Pass a `selector` to `$` jQuery, returns our element
$('#hello').text('Hello, World!');
#             ↑ - Set the Text on the element
```

Подробнее см. На странице jQuery - Documentation .

Включить тег script в начало страницы HTML

Чтобы загрузить **jQuery** с официального CDN , перейдите на сайт jQuery. Вы увидите список различных версий и форматов.



Теперь скопируйте источник версии jQuery, который вы хотите загрузить. Предположим, вы хотите загрузить **jQuery 2.X** , щелкните **без сжатия** или **миниатюрный** тег, который покажет вам что-то вроде этого:

Скопируйте полный код (или щелкните значок копирования) и вставьте его в `<head>` или `<body>` вашего html.

Лучшей практикой является загрузка любых внешних библиотек JavaScript в главном теге с помощью атрибута `async` . Вот демонстрация:

```
<!DOCTYPE html>
    <html>
      <head>
         <title>Loading jquery-2.2.4</title>
         <script src="https://code.jquery.com/jquery-2.2.4.min.js" async></script>
      </head>
      <body>
          <p>This page is loaded with jquery.</p>
      </body>
   </html>
```

При использовании атрибута `async` следует осознавать, что библиотеки javascript затем асинхронно загружаются и выполняются как можно скорее. Если две библиотеки включены, где вторая библиотека зависит от первой библиотеки, это случай, если вторая библиотека загружается и выполняется перед первой библиотекой, тогда она может вызывать ошибку, и приложение может сломаться.

Устранение конфликтов пространства имен

Библиотеки, отличные от jQuery, также могут использовать `$` как псевдоним. Это может вызвать помехи между этими библиотеками и jQuery.

Чтобы освободить `$` для использования с другими библиотеками:

```
jQuery.noConflict();
```

После вызова этой функции `$` больше не является псевдонимом для `jQuery` . Тем не менее, вы все равно можете использовать переменную `jQuery` для доступа к функциям jQuery:

```
jQuery('#hello').text('Hello, World!');
```

При желании вы можете назначить другую переменную в качестве псевдонима для jQuery:

```
var jqy = jQuery.noConflict();
jqy('#hello').text('Hello, World!');
```

И наоборот, чтобы другие библиотеки не вмешивались в jQuery, вы можете обернуть свой код jQuery в выведенное сразу выражение функции (IIFE) и передать в `jQuery` в качестве аргумента:

```
(function($) {
    $(document).ready(function() {
        $('#hello').text('Hello, World!');
    });
})(jQuery);
```

Внутри этого IIFE `$` является псевдонимом только для jQuery.

Еще один простой способ **защитить псевдоним `$` jQuery и убедиться, что DOM готов** :

```
jQuery(function( $ ) { // DOM is ready
   // You're now free to use $ alias
   $('#hello').text('Hello, World!');
});
```

Подвести итоги,

- `jQuery.noConflict()` : `$` больше не ссылается на jQuery, а переменная `jQuery` .
- `var jQuery2 = jQuery.noConflict()` - `$` больше не ссылается на jQuery, тогда как переменная `jQuery` делает и переменная `jQuery2` .

Теперь существует третий сценарий. Что делать, если мы хотим, чтобы jQuery был доступен **только в `jQuery2`** ? Использование,

```
var jQuery2 = jQuery.noConflict(true)
```

Это приводит к тому, что ни `$` ни `jQuery` ссылаются на jQuery.

Это полезно, когда несколько версий jQuery должны быть загружены на одну страницу.

```
<script src='https://code.jquery.com/jquery-1.12.4.min.js'></script>
<script>
    var jQuery1 = jQuery.noConflict(true);
</script>
<script src='https://code.jquery.com/jquery-3.1.0.min.js'></script>
<script>
    // Here, jQuery1 refers to jQuery 1.12.4 while, $ and jQuery refers to jQuery 3.1.0.
</script>
```

https://learn.jquery.com/using-jquery-core/avoid-conflicts-other-libraries/

Загрузка jQuery через консоль на странице, у которой ее нет.

Иногда приходится работать со страницами, которые не используют `jQuery` то время как большинство разработчиков используют `jQuery` .

В таких ситуациях можно использовать консоль `Chrome Developer Tools` (`F12`), чтобы вручную добавить `jQuery` на загруженную страницу, выполнив следующие действия:

```
var j = document.createElement('script');
j.onload = function(){ jQuery.noConflict(); };
j.src = "https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js";
document.getElementsByTagName('head')[0].appendChild(j);
```

Версия, которую вы хотите, может отличаться от `1.12.4` (`1.12.4`), вы можете получить ссылку на одну из них, которая вам нужна здесь .

Объект jQuery

Каждый раз, когда вызывается jQuery, используя `$()` или `jQuery()` , внутри он создает `new` экземпляр `jQuery` . Это исходный код, который показывает новый экземпляр:

```
// Define a local copy of jQuery
jQuery = function( selector, context ) {

    // The jQuery object is actually just the init constructor 'enhanced'
    // Need init if jQuery is called (just allow error to be thrown if not included)
    return new jQuery.fn.init( selector, context );
}
```

Внутренне jQuery ссылается на свой прототип как `.fn` , а стиль, используемый здесь для внутренней инициализации объекта jQuery, позволяет показывать этот прототип без явного использования `new` вызывающим.

В дополнение к настройке экземпляра (таким `.each` , `.each` jQuery API, например, `.each` , `children` , `filter` и т. Д.), Внутренне jQuery также создает структуру, подобную массиву,

чтобы соответствовать результату селектора (при условии, что в качестве аргумента было передано что-то другое, кроме ничего, `undefined` , `null` или подобного.). В случае одного элемента эта подобная массиву структура будет содержать только этот элемент.

Простой демонстрацией было бы найти элемент с идентификатором, а затем получить доступ к объекту jQuery, чтобы вернуть базовый элемент DOM (это также будет работать, когда несколько элементов совпадают или присутствуют).

```
var $div = $("#myDiv");//populate the jQuery object with the result of the id selector
var div = $div[0];//access array-like structure of jQuery object to get the DOM Element
```

Загрузка плагинов jQuery с именами

Обычно при загрузке плагинов обязательно включайте плагин *после* jQuery.

```
<script src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
<script src="some-plugin.min.js"></script>
```

Если вы *должны* использовать более одной версии jQuery, то обязательно загрузите плагин (ы) *после* необходимой версии jQuery, за которой следует код, чтобы установить `jQuery.noConflict(true)` ; затем загрузите следующую версию jQuery и связанные с ней плагины:

```
<script src="https://code.jquery.com/jquery-1.7.0.min.js"></script>
<script src="plugin-needs-1.7.min.js"></script>
<script>
// save reference to jQuery v1.7.0
var $oldjq = jQuery.noConflict(true);
</script>
<script src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
<script src="newer-plugin.min.js"></script>
```

Теперь при инициализации плагинов вам нужно будет использовать связанную версию jQuery

```
<script>
// newer jQuery document ready
jQuery(function($){
  // "$" refers to the newer version of jQuery
  // inside of this function

  // initialize newer plugin
  $('#new').newerPlugin();
});

// older jQuery document ready
$oldjq(function($){
  // "$" refers to the older version of jQuery
  // inside of this function

  // initialize plugin needing older jQuery
```

```
  $('#old').olderPlugin();
});
</script>
```

Для инициализации обоих плагинов можно использовать только одну функцию готовности документа, но чтобы избежать путаницы и проблем с каким-либо дополнительным кодом jQuery внутри функции готовности документа, было бы лучше оставить ссылки отдельными.

Прочитайте Начало работы с jQuery онлайн: https://riptutorial.com/ru/jquery/topic/211/начало-работы-с-jquery

глава 2: Ajax

Синтаксис

- var jqXHR = $ .ajax (url [, settings])
- var jqXHR = $ .ajax ([настройки])
- jqXHR.done (функция (data, textStatus, jqXHR) {});
- jqXHR.fail (функция (jqXHR, textStatus, errorThrown) {});
- jqXHR.always (function (jqXHR) {});

параметры

параметр	подробности
URL	Указывает URL-адрес, на который будет отправлен запрос
настройки	объект, содержащий многочисленные значения, которые влияют на поведение запроса
тип	Метод HTTP, который будет использоваться для запроса
данные	Данные, которые будут отправлены по запросу
успех	Функция обратного вызова, вызываемая при успешном выполнении запроса
ошибка	Обратный вызов для обработки ошибки
StatusCode	Объект числовых HTTP-кодов и функций, которые будут вызываться, когда ответ имеет соответствующий код
тип данных	Тип данных, которые вы ожидаете от сервера
Тип содержимого	Тип содержимого данных, отправляемых на сервер. По умолчанию используется «application / x-www-form-urlencoded; charset = UTF-8»
контекст	Определяет контекст, который будет использоваться внутри обратных вызовов, обычно `this` относится к текущей цели.

замечания

AJAX означает **A** синхронный **J** avaScript **и X** ML. AJAX позволяет веб-странице выполнять асинхронный запрос HTTP (AJAX) на сервер и получать ответ, не загружая всю страницу.

Examples

Обработка HTTP-ответов с помощью $ .ajax ()

В дополнении к `.done` , `.fail` и `.always` обещают обратные вызовы, которые срабатывают на основании был ли запрос успешным или нет, есть вариант , чтобы вызвать функцию , когда конкретный код HTTP Status возвращаются с сервера. Это можно сделать с `statusCode` параметра `statusCode` .

```
$.ajax({
    type: {POST or GET or PUT etc.},
    url:  {server.url},
    data: {someData: true},
    statusCode: {
        404: function(responseObject, textStatus, jqXHR) {
            // No content found (404)
            // This code will be executed if the server returns a 404 response
        },
        503: function(responseObject, textStatus, errorThrown) {
            // Service Unavailable (503)
            // This code will be executed if the server returns a 503 response
        }
    }
})
.done(function(data){
    alert(data);
})
.fail(function(jqXHR, textStatus){
    alert('Something went wrong: ' + textStatus);
})
.always(function(jqXHR, textStatus) {
   alert('Ajax request was finished')
});
```

Как официальная документация jQuery гласит:

> Если запрос выполнен успешно, функции кода состояния имеют те же параметры, что и обратный вызов успеха; если это приводит к ошибке (включая перенаправление 3xx), они принимают те же параметры, что и обратный вызов `error` .

Использование Ajax для отправки формы

Иногда у вас может быть форма и вы хотите отправить ее с помощью ajax.

Предположим, у вас есть эта простая форма -

```
<form id="ajax_form" action="form_action.php">
  <label for="name">Name :</label>
  <input name="name" id="name" type="text" />
  <label for="name">Email :</label>
  <input name="email" id="email" type="text" />
```

```
  <input type="submit" value="Submit" />
</form>
```

Следующий код jQuery можно использовать (в вызове `$(document).ready`) -

```
$('#ajax_form').submit(function(event){
  event.preventDefault();
  var $form = $(this);

  $.ajax({
      type: 'POST',
      url: $form.attr('action'),
      data: $form.serialize(),
      success: function(data) {
        // Do something with the response
      },
      error: function(error) {
        // Do something with the error
      }
  });
});
```

объяснение

- `var $form = $(this)` - форма, кэшированная для повторного использования
- `$('#ajax_form').submit(function(event){` - Когда представлена форма с идентификатором «ajax_form», запустите эту функцию и передайте событие в качестве параметра.
- `event.preventDefault();` - Предотвратить отправку формы нормально (в качестве альтернативы мы можем использовать `return false` после инструкции `ajax({});` которая будет иметь тот же эффект)
- `url: $form.attr('action'),` - получить значение атрибута «действие» формы и использовать его для свойства «url».
- `data: $form.serialize(),` - преобразует входные данные в форме в строку, подходящую для отправки на сервер. В этом случае он вернет что-то вроде « name=Bob&email=bob@bobsemailaddress.com»

Отправка данных JSON

jQuery делает обработку *ответов* jSON безболезненной, но требуется немного больше работы, когда данный запрос желает *отправить* данные в формате JSON:

```
 $.ajax("/json-consuming-route", {
      data: JSON.stringify({author: {name: "Bullwinkle J. Moose",
                                     email: "bullwinkle@example.com"} }),
      method: "POST",
      contentType: "application/json"
   });
```

Обратите внимание, что мы указываем правильный `contentType` для данных, которые мы

отправляем; это хорошая практика в целом и может потребоваться для API, который вы отправляете, но он *также* имеет побочный эффект от инструкции jQuery не выполнять преобразование по умолчанию от `%20` до `+` , что и было бы, если `contentType` был слева от значения по умолчанию `application/x-www-form-urlencoded` . Если по какой-то причине вы должны оставить contentType установленным по умолчанию, обязательно установите для параметра `processData` значение false, чтобы предотвратить это.

Вызов `JSON.stringify` можно было бы избежать здесь, но его использование позволяет нам предоставлять данные в виде объекта JavaScript (таким образом, избегая смущающих синтаксических ошибок JSON, таких как отказ от цитирования имен свойств).

Все в одном примере

Ajax Получить:

Решение 1:

```
$.get('url.html', function(data){
    $('#update-box').html(data);
});
```

Решение 2:

```
 $.ajax({
     type: 'GET',
     url: 'url.php',
 }).done(function(data){
     $('#update-box').html(data);
 }).fail(function(jqXHR, textStatus){
     alert('Error occured: ' + textStatus);
 });
```

Ajax Load: еще один метод get ajax для упрощения

```
$('#update-box').load('url.html');
```

.load также можно вызывать с дополнительными данными. Часть данных может быть представлена как строка или объект.

```
$('#update-box').load('url.php', {data: "something"});
$('#update-box').load('url.php', "data=something");
```

Если .load вызывается с помощью метода обратного вызова, запрос на сервер будет сообщением

```
$('#update-box').load('url.php', {data: "something"}, function(resolve){
    //do something
});
```

Ajax Post:

Решение 1:

```
$.post('url.php',
    {date1Name: data1Value, date2Name: data2Value},  //data to be posted
    function(data){
        $('#update-box').html(data);
    }
);
```

Решение 2:

```
$.ajax({
    type: 'Post',
    url: 'url.php',
    data: {date1Name: data1Value, date2Name: data2Value}  //data to be posted
}).done(function(data){
    $('#update-box').html(data);
}).fail(function(jqXHR, textStatus){
    alert('Error occured: ' + textStatus);
});
```

Ajax Post JSON:

```
var postData = {
    Name: name,
    Address: address,
    Phone: phone
};

$.ajax({
     type: "POST",
     url: "url.php",
     dataType: "json",
     data: JSON.stringfy(postData),
     success: function (data) {
         //here variable data is in JSON format
     }
 });
```

Ajax Получить JSON:

Решение 1:

```
$.getJSON('url.php', function(data){
    //here variable data is in JSON format
});
```

Решение 2:

```
 $.ajax({
      type: "Get",
      url: "url.php",
```

```
      dataType: "json",
      data: JSON.stringfy(postData),
      success: function (data) {
          //here variable data is in JSON format
      },
      error: function(jqXHR, textStatus){
          alert('Error occured: ' + textStatus);
      }
  });
```

Ajax Прервать вызов или запрос

```
var xhr = $.ajax({
    type: "POST",
    url: "some.php",
    data: "name=John&location=Boston",
    success: function(msg){
       alert( "Data Saved: " + msg );
    }
});
```

// убиваем запрос

```
xhr.abort()
```

Загрузка файлов Ajax

1. Простой полный пример

Мы могли бы использовать этот примерный код для загрузки файлов, выбранных пользователем каждый раз при создании нового файла.

```
<input type="file" id="file-input" multiple>
```

```
var files;
var fdata = new FormData();
$("#file-input").on("change", function (e) {
    files = this.files;

    $.each(files, function (i, file) {
        fdata.append("file" + i, file);
    });

    fdata.append("FullName", "John Doe");
    fdata.append("Gender", "Male");
    fdata.append("Age", "24");

    $.ajax({
        url: "/Test/Url",
        type: "post",
        data: fdata, //add the FormData object to the data parameter
```

```
        processData: false, //tell jquery not to process data
        contentType: false, //tell jquery not to set content-type
        success: function (response, status, jqxhr) {
            //handle success
        },
        error: function (jqxhr, status, errorMessage) {
            //handle error
        }
    });
});
```

Теперь давайте разберем это и проверим его по частям.

2. Работа с файловыми входами

Этот MDN-документ (с использованием файлов из веб-приложений) хорошо читает о различных методах обработки входных файлов. Некоторые из этих методов также будут использоваться в этом примере.

Прежде чем мы загрузим файлы, мы сначала должны предоставить пользователю способ выбрать файлы, которые они хотят загрузить. Для этого мы будем использовать `file input` . `multiple` свойство позволяет выбирать несколько файлов, их можно удалить, если вы хотите, чтобы пользователь выбирал один файл за раз.

```
<input type="file" id="file-input" multiple>
```

Мы будем использовать `change event` ввода для захвата файлов.

```
var files;
$("#file-input").on("change", function(e){
    files = this.files;
});
```

Внутри функции обработчика мы получаем доступ к файлам через свойство файлов нашего ввода. Это дает нам FileList , который является массивом, подобным объекту.

3. Создание и заполнение формы

Чтобы загрузить файлы с помощью Ajax, мы будем использовать FormData .

```
var fdata = new FormData();
```

FileList, который мы получили на предыдущем шаге, является массивом, подобным объекту, и его можно повторить с использованием различных методов, в том числе для цикла , для ... цикла и jQuery.each . В этом примере мы будем придерживаться jQuery.

```
$.each(files, function(i, file) {
  //...
});
```

Мы будем использовать метод добавления формы FormData для добавления файлов в наш объект formdata.

```
$.each(files, function(i, file) {
  fdata.append("file" + i, file);
});
```

Мы также можем добавить другие данные, которые мы хотим отправить таким же образом. Предположим, мы хотим отправить некоторую личную информацию, которую мы получили от пользователя вместе с файлами. Мы могли бы добавить эту информацию в наш объект formdata.

```
fdata.append("FullName", "John Doe");
fdata.append("Gender", "Male");
fdata.append("Age", "24");
//...
```

4. Отправка файлов с помощью Ajax

```
$.ajax({
    url: "/Test/Url",
    type: "post",
    data: fdata, //add the FormData object to the data parameter
    processData: false, //tell jquery not to process data
    contentType: false, //tell jquery not to set content-type
    success: function (response, status, jqxhr) {
        //handle success
    },
    error: function (jqxhr, status, errorMessage) {
        //handle error
    }
});
```

Мы устанавливаем `processData` свойств `processData` и `contentType` значение `false` . Это делается для того, чтобы файлы могли быть отправлены на сервер и правильно обработаны сервером.

Прочитайте Ajax онлайн: https://riptutorial.com/ru/jquery/topic/316/ajax

глава 3: jQuery Отложенные объекты и обещания

Вступление

Обещания jQuery - это умный способ объединения асинхронных операций в блок-блоке. Это заменяет вставку старых вызовов обратных вызовов, которые не так легко реорганизовываются.

Examples

Создание основных обещаний

Вот очень простой пример функции, которая « *обещает* действовать, когда истекает заданное время». Он делает это, создавая новый объект « `Deferred` , который разрешается позже и возвращает обещание Отсрочка:

```
function waitPromise(milliseconds){

   // Create a new Deferred object using the jQuery static method
   var def = $.Deferred();

   // Do some asynchronous work - in this case a simple timer
   setTimeout(function(){

       // Work completed... resolve the deferred, so it's promise will proceed
       def.resolve();
   }, milliseconds);

   // Immediately return a "promise to proceed when the wait time ends"
   return def.promise();
}
```

И используйте вот так:

```
waitPromise(2000).then(function(){
     console.log("I have waited long enough");
});
```

Асинхронные обещания

Если у вас есть несколько асинхронных задач, которые должны возникать один за другим, вам нужно объединить свои объекты обещаний. Вот простой пример:

```
function First() {
    console.log("Calling Function First");
```

```
    return $.get("/ajax/GetFunction/First");
}

function Second() {
    console.log("Calling Function Second");
    return $.get("/ajax/GetFunction/Second");
}

function Third() {
    console.log("Calling Function Third");
    return $.get("/ajax/GetFunction/Third");
}

function log(results){
    console.log("Result from previous AJAX call: " + results.data);
}

First().done(log)
       .then(Second).done(log)
       .then(Third).done(log);
```

jQuery ajax () успех, ошибка VS .done (), .fail ()

успех и ошибка: обратный вызов **успеха,** который вызывается при успешном завершении запроса Ajax.

Обратный вызов **сбоя,** который вызывается при возникновении ошибки при выполнении запроса.

Пример:

```
 $.ajax({
        url: 'URL',
        type: 'POST',
        data: yourData,
        datatype: 'json',
        success: function (data) { successFunction(data); },
        error: function (jqXHR, textStatus, errorThrown) { errorFunction(); }
    });
```

.done () и .fail ():

.ajax (). done (function (data, textStatus, jqXHR) {}); Заменяет метод .success (), который устарел в jQuery 1.8.Это альтернативная конструкция для функции обратного вызова успеха выше.

.ajax (). fail (function (jqXHR, textStatus, errorThrown) {}); Заменяет метод .error (), который устарел в jQuery 1.8.Это альтернативная конструкция для полной функции обратного вызова выше.

Пример:

```
$.ajax({
    url: 'URL',
    type: 'POST',
    data: yourData,
    datatype: 'json'
})
.done(function (data) { successFunction(data); })
.fail(function (jqXHR, textStatus, errorThrown) { serrorFunction(); });
```

Получите текущее состояние обещания

По умолчанию состояние обещания ожидает, когда оно будет создано. Состояние обещания изменяется, когда отложенный объект, создавший обещание, решает / отклоняет его.

```
var deferred = new $.Deferred();
var d1= deferred.promise({
    prop: "value"
});
var d2= $("div").promise();
var d3= $("div").hide(1000).promise();

console.log(d1.state()); // "pending"
console.log(d2.state()); // "resolved"
console.log(d3.state()); // "pending"
```

Прочитайте jQuery Отложенные объекты и обещания онлайн:
https://riptutorial.com/ru/jquery/topic/8308/jquery-отложенные-объекты-и-обещания

глава 4: Prepend

Examples

Предоставление элемента контейнеру

Решение 1:

```
$('#parent').prepend($('#child'));
```

Решение 2:

```
$('#child').prependTo($('#parent'));
```

Оба решения добавляют элемент `#child` (добавление в начале) к элементу `#parent` .

До:

```
<div id="parent">
  <span>other content</span>
</div>
<div id="child">

</div>
```

После:

```
<div id="parent">
  <div id="child">

  </div>
  <span>other content</span>
</div>
```

Метод подготовки

`prepend()` - вставить содержимое, заданное параметром, в начало каждого элемента в наборе согласованных элементов.

1. `prepend( content [, content ] )`

```
// with html string
jQuery('#parent').prepend('<span>child</span>');
// or you can use jQuery object
jQuery('#parent').prepend($('#child'));
// or you can use comma seperated multiple elements to prepend
jQuery('#parent').prepend($('#child1'),$('#child2'));
```

2. `prepend(function)`

`version: 1.4` JQuery *`version: 1.4`* начиная с *`version: 1.4`* вы можете использовать функцию обратного вызова в качестве аргумента. Где вы можете получить аргументы в качестве позиции индекса элемента в наборе и старого значения HTML элемента. Внутри функции `this` относится к текущему элементу в наборе.

```
jQuery('#parent').prepend(function(i,oldHTML){
     // return the value to be prepend
     return  '<span>child</span>';
});
```

Прочитайте Prepend онлайн: https://riptutorial.com/ru/jquery/topic/1909/prepend

глава 5: Атрибуты

замечания

Функция jQuery `.attr()` получает значение атрибута для **первого** элемента в наборе согласованных элементов или задает один или несколько атрибутов для **каждого** согласованного элемента.

Стоит отметить, что при получении значения атрибута он получает его только от первого элемента, который соответствует селектору (т.е. `$("input").attr("type");` получит только тип первого ввода , если их больше одного)

Однако при настройке атрибута он будет применяться ко всем соответствующим элементам.

Examples

Получить значение атрибута элемента HTML

Когда один параметр передается функции `.attr()` он возвращает значение переданного атрибута для выбранного элемента.

Синтаксис:

```
$([selector]).attr([attribute name]);
```

Пример:

HTML:

```
<a href="/home">Home</a>
```

JQuery:

```
$('a').attr('href');
```

Получение атрибутов `data` :

jQuery предлагает функцию `.data()` для обработки атрибутов данных. `.data` возвращает значение атрибута data для выбранного элемента.

Синтаксис:

```
$([selector]).data([attribute name]);
```

Пример:

Html:

```
<article data-column="3"></article>
```

JQuery:

```
$("article").data("column")
```

Замечания:

> Метод data () jQuery даст вам доступ к атрибутам data- *, НО, он сжимает случай имени атрибута. Ссылка

Установка значения атрибута HTML

Если вы хотите добавить атрибут к некоторому элементу, вы можете использовать функцию `attr(attributeName, attributeValue)` . Например:

```
$('a').attr('title', 'Click me');
```

В этом примере мы добавим текст курсора мыши `"Click me"` ко всем ссылкам на странице.

Эта же функция используется для изменения значений атрибутов.

Удаление атрибута

Чтобы удалить атрибут из элемента, вы можете использовать функцию `.removeAttr(attributeName)` . Например:

```
$('#home').removeAttr('title');
```

Это приведет к удалению атрибута `title` из элемента с идентификатором `home` .

Разница между attr () и prop ()

`attr()` получает / устанавливает атрибут HTML, используя функции DOM `getAttribute()` и `setAttribute()` . `prop()` работает, установив свойство DOM без изменения атрибута. Во многих случаях два являются взаимозаменяемыми, но иногда они нужны друг другу.

Чтобы установить флажок:

```
$('#tosAccept').prop('checked', true); // using attr() won't work properly here
```

Чтобы удалить свойство, вы можете использовать метод `removeProp()` . Аналогично `removeAttr()` удаляет атрибуты.

Прочитайте Атрибуты онлайн: https://riptutorial.com/ru/jquery/topic/4429/атрибуты

глава 6: Видимость элементов

параметры

параметр	подробности
продолжительность	Когда передано, эффекты `.hide()` , `.show()` и `.toggle()` анимируются; элемент (ы) будет постепенно исчезать или исчезать.

Examples

обзор

```
$(element).hide()            // sets display: none
$(element).show()            // sets display to original value
$(element).toggle()          // toggles between the two
$(element).is(':visible')    // returns true or false
$('element:visible')         // matches all elements that are visible
$('element:hidden')          // matches all elements that are hidden

$('element').fadeIn();          // display the element
$('element').fadeOut();          // hide the element

$('element').fadeIn(1000);          // display the element using timer
$('element').fadeOut(1000);          // hide the element using timer

// display the element using timer and a callback function
$('element').fadeIn(1000, function(){
 // code to execute
});

// hide the element using timer and a callback function
$('element').fadeOut(1000, function(){
   // code to execute
});
```

Переключить возможности

Простой случай `toggle()`

```
function toggleBasic() {
  $(".target1").toggle();
}
```

С определенной *продолжительностью*

```
function toggleDuration() {
```

```
  $(".target2").toggle("slow"); // A millisecond duration value is also acceptable
}
```

... и *обратный вызов*

```
function toggleCallback() {
  $(".target3").toggle("slow",function(){alert('now do something');});
}
```

... или с *ослаблением* и обратным вызовом.

```
function toggleEasingAndCallback() {
  // You may use jQueryUI as the core only supports linear and swing easings
  $(".target4").toggle("slow","linear",function(){alert('now do something');});
}
```

... или с различными *вариантами* .

```
function toggleWithOptions() {
  $(".target5").toggle(
    { // See all possible options in: api.jquery.com/toggle/#toggle-options
      duration:1000, // milliseconds
      easing:"linear",
      done:function(){
        alert('now do something');
      }
    }
  );
}
```

Также можно использовать *слайд* как анимацию с `slideToggle()`

```
function toggleSlide() {
  $(".target6").slideToggle(); // Animates from top to bottom, instead of top corner
}
```

... или затухают в / из, изменяя непрозрачность с помощью `fadeToggle()`

```
function toggleFading() {
  $( ".target7" ).fadeToggle("slow")
}
```

... или переключить класс с помощью `toggleClass()`

```
function toggleClass() {
  $(".target8").toggleClass('active');
}
```

Обычным случаем является `toggle()` , чтобы показать один элемент, скрывая другой (тот же класс)

```
function toggleX() {
  $(".targetX").toggle("slow");
}
```

Все приведенные выше примеры можно найти здесь

Прочитайте Видимость элементов онлайн: https://riptutorial.com/ru/jquery/topic/1298/видимость-элементов

глава 7: Каждая функция

Examples

Основное использование

```
// array
var arr = [
   'one',
   'two',
   'three',
   'four'
];
$.each(arr, function (index, value) {
  console.log(value);

  // Will stop running after "three"
  return (value !== 'three');
});
// Outputs: one two three
```

jQuery каждая функция

HTML:

```
<ul>
  <li>Mango</li>
  <li>Book</li>
</ul>
```

Автор сценария:

```
$( "li" ).each(function( index ) {
  console.log( index + ": " + $( this ).text() );
});
```

Таким образом, сообщение регистрируется для каждого элемента в списке:

0: Манго

1: Книга

Прочитайте Каждая функция онлайн: https://riptutorial.com/ru/jquery/topic/10853/каждая-функция

глава 8: Манипуляция CSS

Синтаксис

- .css (cssProperty) // Получить значение свойства rendered CSS
- .css ([cssProperty, ...]) // Получить значения из массива cssProperties
- .css (cssProperty, value) // Установленное значение
- .css ({cssProperty: значение, ...}) // Установить свойства и значения
- .css (cssProperty, function) // Представление cssProperty для функции обратного вызова

замечания

Полученные значения

Если используется единица реагирования (например, `"auto"` , `"%"` , `"vw"` и т. Д.), `.css()` вернет фактическое отображаемое значение в `px`

```
.myElement{ width: 20%; }
```

```
var width = $(".myElement").css("width"); // "123px"
```

Форматирование свойств и значений

Свойства можно определить с помощью **стандартного форматирования CSS как String** или с помощью **camelCase**

```
"margin-bottom"
marginBottom
```

Значения должны быть выражены в String. Числовые значения рассматриваются как `px` единицы внутри jQuery

```
.css(fontSize: "1em")
.css(fontSize: "16px")
.css(fontSize: 16)       // px will be used
```

Начиная с jQuery 3 избегайте использования `.show()` и `.hide()`

Согласно этой статье в блоге jQuery , из-за проблем с накладными расходами и производительности вы больше не должны использовать `.show()` или `.hide()` .

Если у вас есть элементы в таблице стилей, которые установлены для `display: none` , метод `.show()` больше не будет отменять это. Поэтому самым важным правилом перехода на jQuery 3.0 является следующее: Не используйте таблицу стилей для установки значения по умолчанию для `display: none` а затем попытайтесь использовать `.show()` - или любой метод, который показывает элементы, такие как `.slideDown()` и `.fadeIn()` - сделать видимым. Если вам нужен элемент, который должен быть скрыт по умолчанию, лучший способ - добавить к элементу имя класса, например «скрытый», и определить, какой класс будет `display: none` в таблице стилей. Затем вы можете добавить или удалить этот класс, используя `.addClass()` jQuery `.addClass()` и `.removeClass()` для контроля видимости. В качестве альтернативы вы можете иметь вызов `.ready()` обработчика `.hide()` для элементов, прежде чем они будут отображаться на странице. Или, если вы действительно должны сохранить таблицу стилей по умолчанию, вы можете использовать `.css("display", "block")` (или соответствующее отображаемое значение), чтобы переопределить таблицу стилей.

Examples

Установить свойство CSS

Установка только одного стиля:

```
$('#target-element').css('color', '#000000');
```

Установка нескольких стилей одновременно:

```
$('#target-element').css({
    'color': '#000000',
    'font-size': '12pt',
    'float': 'left',
});
```

Получить свойство CSS

Чтобы получить свойство CSS элемента, вы можете использовать метод `.css(propertyName)` :

```
var color    = $('#element').css('color');
var fontSize = $('#element').css('font-size');
```

Инкремент / Уменьшение числовых свойств

Числовые свойства CSS могут быть увеличены и уменьшены с помощью синтаксиса `+=` и `-=` , соответственно, с использованием метода `.css()` :

```
// Increment using the += syntax
$("#target-element").css("font-size", "+=10");

// You can also specify the unit to increment by
$("#target-element").css("width", "+=100pt");
$("#target-element").css("top", "+=30px");
$("#target-element").css("left", "+=3em");

// Decrementing is done by using the -= syntax
$("#target-element").css("height", "-=50pt");
```

CSS - Getters and Setters

CSS Getter

Функция `.css()` **getter** может применяться к каждому элементу DOM на странице следующим образом:

```
// Rendered width in px as a string. ex: `150px`
// Notice the `as a string` designation - if you require a true integer,
// refer to `$.width()` method
$("body").css("width");
```

Эта строка вернет **вычисленную ширину** указанного элемента, каждое свойство CSS, которое вы предоставили в круглых скобках, даст значение свойства для этого элемента DOM `$("selector")` , если вы попросите атрибут CSS, который не существует, вы будет `undefined` как ответ.

Вы также можете вызвать **getter** с массивом атрибутов:

```
$("body").css(["animation","width"]);
```

это вернет объект всех атрибутов со своими значениями:

```
Object {animation: "none 0s ease 0s 1 normal none running", width: "529px"}
```

CSS Setter

`.css()` метод **установки** также может быть применен к каждому элементу DOM на странице.

```
$("selector").css("width", 500);
```

Этот оператор устанавливает `width` `$("selector")` в `500px` и возвращает объект jQuery, чтобы вы могли связать больше методов с указанным селектором.

`.css()` **сеттер** также может быть использована передача объекта свойств CSS и значений:

```
$("body").css({"height": "100px", width:100, "padding-top":40, paddingBottom:"2em"});
```

Все изменения, внесенные сеттером, добавляются к свойству `style` элемента DOM, что влияет на стили элементов (если только значение свойства стиля уже не определено как `!important` другом месте в стилях).

Прочитайте Манипуляция CSS онлайн: https://riptutorial.com/ru/jquery/topic/2732/манипуляция-css

глава 9: Манипуляция DOM

Examples

Создание элементов DOM

Функция `jQuery` (обычно псевдоним как `$`) может использоваться как для выбора элементов, так и для создания новых элементов.

```
var myLink = $('<a href="http://stackexchange.com"></a>');
```

Вы можете опционально передать второй аргумент с атрибутами элемента:

```
var myLink = $('<a>', { 'href': 'http://stackexchange.com' });
```

`'<a>'` -> Первый аргумент указывает тип элемента DOM, который вы хотите создать. В этом примере это якорь, но может быть что угодно в этом списке . См. Спецификацию для ссылки на элемент `a` .

`{ 'href': 'http://stackexchange.com' }` -> второй аргумент - это объект JavaScript, содержащий пары имен / значений атрибутов.

пары «имя»: «значение» будут отображаться между `< >` первого аргумента, например `<a name:value>` который для нашего примера будет `<a href="http://stackexchange.com"></a>`

Манипулирование классами элементов

Предполагая, что страница содержит элемент HTML, такой как:

```
<p class="small-paragraph">
  This is a small <a href="https://en.wikipedia.org/wiki/Paragraph">paragraph</a>
  with a <a class="trusted" href="http://stackexchange.com">link</a> inside.
</p>
```

jQuery предоставляет полезные функции для управления классами DOM, в частности `hasClass()` , `addClass()` , `removeClass()` и `toggleClass()` . Эти функции непосредственно изменяют атрибут `class` согласованных элементов.

```
$('p').hasClass('small-paragraph'); // true
$('p').hasClass('large-paragraph'); // false

// Add a class to all links within paragraphs
$('p a').addClass('untrusted-link-in-paragraph');

// Remove the class from a.trusted
$('a.trusted.untrusted-link-in-paragraph')
```

```
.removeClass('untrusted-link-in-paragraph')
.addClass('trusted-link-in-paragraph');
```

Переключить класс

Учитывая пример разметки, мы можем добавить класс с нашим первым `.toggleClass()` :

```
$(".small-paragraph").toggleClass("pretty");
```

Теперь это вернет `true` : `$(".small-paragraph").hasClass("pretty")`

`toggleClass` обеспечивает тот же эффект с меньшим кодом как:

```
if($(".small-paragraph").hasClass("pretty")){
   $(".small-paragraph").removeClass("pretty");}
else {
   $(".small-paragraph").addClass("pretty"); }
```

toggle Два класса:

```
$(".small-paragraph").toggleClass("pretty cool");
```

Boolean для добавления / удаления классов:

```
$(".small-paragraph").toggleClass("pretty",true); //cannot be truthy/falsey

$(".small-paragraph").toggleClass("pretty",false);
```

Функция для переключения классов (см. Пример ниже, чтобы избежать проблемы)

```
$( "div.surface" ).toggleClass(function() {
  if ( $( this ).parent().is( ".water" ) ) {
    return "wet";
  } else {
    return "dry";
  }
});
```

Используется в примерах:

```
// functions to use in examples
function stringContains(myString, mySubString) {
  return myString.indexOf(mySubString) !== -1;
}
function isOdd(num) { return num % 2;}
var showClass = true; //we want to add the class
```

Примеры:

Используйте индекс элемента для переключения нечетных / четных классов

```
$( "div.gridrow" ).toggleClass(function(index,oldClasses, false), showClass ) {
  showClass
  if ( isOdd(index) ) {
    return "wet";
  } else {
    return "dry";
  }
});
```

Более сложный пример `toggleClass` , учитывая простую разметку сетки

```
<div class="grid">
  <div class="gridrow">row</div>
  <div class="gridrow">row</div>
  <div class="gridrow">row</div>
  <div class="gridrow">row</div>
  <div class="gridrow">row</div>
  <div class="gridrow gridfooter">row but I am footer!</div>
</div>
```

Простые функции для наших примеров:

```
function isOdd(num) {
  return num % 2;
}

function stringContains(myString, mySubString) {
  return myString.indexOf(mySubString) !== -1;
}
var showClass = true; //we want to add the class
```

Добавьте нечетный / четный класс к элементам с классом `gridrow`

```
$("div.gridrow").toggleClass(function(index, oldClasses, showThisClass) {
  if (isOdd(index)) {
    return "odd";
  } else {
    return "even";
  }
  return oldClasses;
}, showClass);
```

Если строка имеет класс `gridfooter` , удалите нечетные / четные классы, сохраните остальные.

```
$("div.gridrow").toggleClass(function(index, oldClasses, showThisClass) {
  var isFooter = stringContains(oldClasses, "gridfooter");
  if (isFooter) {
    oldClasses = oldClasses.replace('even', ' ').replace('odd', ' ');
    $(this).toggleClass("even odd", false);
  }
  return oldClasses;
}, showClass);
```

Возвращаемые классы - это то, что производится. Здесь, если элемент не имеет `gridfooter` , добавьте класс для четного / нечетного. Этот пример иллюстрирует возврат списка классов OLD. Если это `else return oldClasses;` удаляется, добавляются только новые классы, поэтому строка с классом `gridfooter` будет удалена всеми классами, если бы мы не вернули те старые, которые в противном случае были бы отключены (удалены).

```
$("div.gridrow").toggleClass(function(index, oldClasses, showThisClass) {
  var isFooter = stringContains(oldClasses, "gridfooter");
  if (!isFooter) {
    if (isOdd(index)) {
      return "oddLight";
    } else {
      return "evenLight";
    }
  } else return oldClasses;
}, showClass);
```

Другие методы API

jQuery предлагает множество методов, которые могут использоваться для манипуляций с DOM.

Первый - это метод .empty () .

Представьте себе следующую разметку:

```
<div id="content">
  <div>Some text</div>
</div>
```

Вызов `$('#content').empty();` , внутренний div будет удален. Это также может быть достигнуто с помощью `$('#content').html('');` ,

Другой удобной функцией является функция .closest () :

```
<tr id="row_1">
  <td><button type="button" class="delete">Delete</button>
</tr>
```

Если вы хотите найти ближайшую строку к кнопке, нажатой в одной из ячеек строки, вы можете сделать это:

```
$('.delete').click(function() {
  $(this).closest('tr');
});
```

Поскольку, вероятно, будет несколько строк, каждый со своими собственными кнопками `delete` , мы используем `$(this)` внутри функции .click (), чтобы ограничить область нажатием кнопки, которую мы нажали.

Если вы хотите получить `id` строки, содержащей кнопку « `Delete` , которую вы нажали, вы можете сделать что-то вроде этого:

```
$('.delete').click(function() {
  var $row = $(this).closest('tr');
  var id = $row.attr('id');
});
```

Обычно считается хорошей практикой префиксные переменные, содержащие объекты jQuery, с значком `$` (доллар), чтобы дать понять, что такое переменная.

Альтернативой `.closest()` является метод .parents () :

```
$('.delete').click(function() {
  var $row = $(this).parents('tr');
  var id = $row.attr('id');
});
```

и есть также функция .parent () :

```
$('.delete').click(function() {
  var $row = $(this).parent().parent();
  var id = $row.attr('id');
});
```

`.parent()` только поднимается на один уровень дерева DOM, поэтому он довольно негибкий, если вы хотите изменить кнопку удаления, которая должна содержаться в пределах `span` например, селектор jQuery будет разбит.

.html ()

Вы можете использовать этот метод для замены всего HTML в селекторе. Предполагая, что у вас есть элемент html, подобный этому

```
<div class="row">
    <div class="col-md-12">
        <div id="information">
            <p>Old message</p>
        </div>
    </div>
</div>
```

Вы можете использовать `.html()` . удалить и добавить предупреждающий или информационный текст, чтобы предупредить всех пользователей одной строкой.

```
$("#information").html("<p>This is the new alert!</p>");
```

Сортировочные элементы

Чтобы эффективно сортировать элементы (все сразу и с минимальным прерыванием DOM), нам необходимо:

1. **Найти** элементы
2. **Сортировка** на основе заданного условия
3. **Вставить** обратно в DOM

```
<ul id='my-color-list'>
    <li class="disabled">Red</li>
    <li>Green</li>
    <li class="disabled">Purple</li>
    <li>Orange</li>
</ul>
```

1. Найдите их - `.children()` или `.find()`

   Это вернет нам объект, похожий на Array.

   ```
   var $myColorList = $('#my-color-list');

   // Elements one layer deep get .children(), any deeper go with .find()
   var $colors = $myColorList.children('li');
   ```

2. `Array.prototype.sort()` их - `Array.prototype.sort()`

   В настоящее время он настроен для возврата элементов в порядке возрастания на основе содержимого HTML (также их цветов).

   ```
   /**
    * Bind $colors to the sort method so we don't have to travel up
    * all these properties more than once.
    */
   var sortList = Array.prototype.sort.bind($colors);

   sortList(function ( a, b ) {

       // Cache inner content from the first element (a) and the next sibling (b)
       var aText = a.innerHTML;
       var bText = b.innerHTML;

       // Returning -1 will place element `a` before element `b`
       if ( aText < bText ) {
           return -1;
       }

       // Returning 1 will do the opposite
       if ( aText > bText ) {
           return 1;
       }

       // Returning 0 leaves them as-is
       return 0;
   });
   ```

3. Вставьте их - `.append()`

   Обратите внимание, что нам не нужно сначала отделять элементы - `append()` будет перемещать элементы, которые уже существуют в DOM, поэтому у нас не будет дополнительных копий

```
// Put it right back where we got it
$myColorList.append($colors);
```

Сделать это мило

Добавить кнопку сортировки

```
<!-- previous HTML above -->
<button type='button' id='btn-sort'>
    Sort
</button>
```

Установите начальное значение направления сортировки

```
var ascending = true;
```

`sortList()` наши элементы DOM и `sortList()` здесь, чтобы свести к минимуму нашу обработку DOM

```
var $myColorList = $('#my-color-list');
var $colors = $myColorList.children('li');
var sortList = Array.prototype.sort.bind($colors);
```

Заверните все в функции `doSort()`

```
// Put the sortList() and detach/append calls in this portable little thing
var doSort = function ( ascending ) {

    sortList(function ( a, b ) {
        // ...
    });

    $myColorList.append($colors);
};
```

Добавить обработчик кликов для `$('#btn-sort')`

```
$('#btn-sort').on('click', function () {
    // Run the sort and pass in the direction value
    doSort(ascending);

    // Toggle direction and save
    ascending = !ascending;
});
```

Все вместе сейчас

```
var ascending = true;

var $myColorList = $('#my-color-list');
var $colors = $myColorList.children('li');
var sortList = Array.prototype.sort.bind($colors);

var doSort = function ( ascending ) {

    sortList(function ( a, b ) {

        var aText = a.innerHTML;
        var bText = b.innerHTML;

        if ( aText < bText ) {
            return ascending ? -1 : 1;
        }

        if ( aText > bText ) {
            return ascending ? 1 : -1;
        }

        return 0;
    });

    $myColorList.append($colors);

};

$('#btn-sort').on('click', function () {
    doSort(ascending);
    ascending = !ascending;
});
```

бонус

Многоуровневая сортировка (группировка отсортированных элементов)

```
// ...

var doSort = function ( ascending ) {

    sortList(function ( a, b ) {
```

```
        // ...initial sorting...

    }).sort(function ( a, b ) {

        // We take the returned items and sort them once more
        var aClass = a.className;
        var bClass = b.className;

        // Let's group the disabled items together and put them at the end

        /**
         * If the two elements being compared have the same class
         * then there's no need to move anything.
         */
        if ( aClass !== bClass ) {
            return aClass === 'disabled' ? 1 : -1;
        }
        return 0;
    });

    // And finalize with re-insert
    $myColorList.append($colors);
};

// ...
```

Можете ли вы сделать это еще на один шаг?

Добавить еще одну кнопку для переключения сортировки отключенной группы

MDN - Array.prototype.sort ()

Прочитайте Манипуляция DOM онлайн: https://riptutorial.com/ru/jquery/topic/512/манипуляция-dom

глава 10: Метод jQuery .animate ()

Синтаксис

1. (Селектор) .animate ({стили}, {} варианты)

параметры

параметр	подробности
свойства	Объект свойств и значений CSS, которые анимация будет двигаться в направлении
продолжительность	(по умолчанию: 400) Строка или номер, определяющий продолжительность выполнения анимации
ослабление	(по умолчанию: swing) Строка, указывающая, какую функцию ослабления использовать для перехода
полный	Функция для вызова после завершения анимации, которая вызывается один раз для каждого элемента.
Начните	определяет функцию, которая будет выполняться при начале анимации.
шаг	определяет функцию, которая будет выполняться для каждого шага анимации.
очередь	логическое значение, указывающее, следует ли разместить анимацию в очереди эффектов.
прогресс	определяет функцию, которая будет выполняться после каждого шага анимации.
сделанный	определяет функцию, которая будет выполняться при завершении анимации.
потерпеть поражение	задает функцию, которая будет выполнена, если анимация не завершится.
specialEasing	карту одного или нескольких свойств CSS из параметра styles и соответствующие им функции ослабления.
всегда	задает функцию, которая будет выполняться, если анимация

параметр	подробности
	прекратится без завершения.

Examples

Анимация с обратным вызовом

Иногда нам нужно изменить положение слов из одного места в другое или уменьшить размер слов и автоматически изменить цвет слов, чтобы улучшить привлекательность нашего веб-сайта или веб-приложений. JQuery много помогает с этой концепцией, используя `fadeIn(), hide(), slideDown()` но ее функциональность ограничена, и она выполняет только конкретную задачу, которая ему присваивается.

Jquery исправляет эту проблему, предоставляя потрясающий и гибкий метод под названием `.animate()` . Этот метод позволяет настраивать пользовательские анимации, в которых используются свойства css, которые дают разрешение на пролет над границами. например, если мы дадим свойство стиля css как `width:200;` и текущая позиция элемента DOM равна 50, анимированный метод уменьшает текущее значение позиции из заданного значения css и анимирует этот элемент до 150. Но нам не нужно беспокоиться об этой части, потому что движок анимации будет обрабатывать его.

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js"></script>
<script>
    $("#btn1").click(function(){
        $("#box").animate({width: "200px"});
    });
</script>

<button id="btn1">Animate Width</button>
<div id="box" style="background:#98bf21;height:100px;width:100px;margin:6px;"></div>
```

Список свойств стиля css, которые разрешены в `.animate()` .

```
backgroundPositionX, backgroundPositionY, borderWidth, borderBottomWidth, borderLeftWidth,
borderRightWidth, borderTopWidth, borderSpacing, margin, marginBottom, marginLeft,
marginRight, marginTop, outlineWidth, padding, paddingBottom, paddingLeft, paddingRight,
paddingTop, height, width, maxHeight, maxWidth, minHeight, minWidth, fontSize, bottom, left,
right, top, letterSpacing,  wordSpacing, lineHeight, textIndent,
```

Скорость указана в `.animate()` .

```
milliseconds (Ex: 100, 1000, 5000, etc.),
"slow",
"fast"
```

Ослабление, указанное в `.animate()` .

«Качели»
«Линейная»

Вот несколько примеров со сложными вариантами анимации.

Например, 1:

```
$( "#book" ).animate({
  width: [ "toggle", "swing" ],
  height: [ "toggle", "swing" ],
  opacity: "toggle"
 }, 5000, "linear", function() {
    $( this ).after( "<div>Animation complete.</div>" );
});
```

Например, 2:

```
  $("#box").animate({
     height: "300px",
     width: "300px"
     }, {
     duration: 5000,
     easing: "linear",
     complete: function(){
        $(this).after("<p>Animation is complete!</p>");
     }
  });
```

Прочитайте Метод jQuery .animate () онлайн: https://riptutorial.com/ru/jquery/topic/5064/метод-jquery--animate---

глава 11: Перемещение DOM

Examples

Выбрать дочерние элементы элемента

Чтобы выбрать дочерние элементы элемента, вы можете использовать метод `children()` .

```
<div class="parent">
    <h2>A headline</h2>
    <p>Lorem ipsum dolor sit amet...</p>
    <p>Praesent quis dolor turpis...</p>
</div>
```

Измените цвет *всех* `.parent` элементов элемента `.parent` :

```
$('.parent').children().css("color", "green");
```

Метод принимает необязательный аргумент `selector` который может использоваться для фильтрации возвращаемых элементов.

```
// Only get "p" children
$('.parent').children("p").css("color", "green");
```

Итерирование списка элементов jQuery

Когда вам нужно перебрать список элементов jQuery.

Рассмотрим эту структуру DOM:

```
<div class="container">
    <div class="red one">RED 1 Info</div>
    <div class="red two">RED 2 Info</div>
    <div class="red three">RED 3 Info</div>
</div>
```

Чтобы напечатать текст, присутствующий во всех элементах `div` с классом `red` :

```
$(".red").each(function(key, ele){
    var text = $(ele).text();
    console.log(text);
});
```

Подсказка: `key` - это индекс элемента `div.red` мы сейчас итерируем, внутри его родителя. `ele` является HTML элемент, так что мы можем создать объект JQuery из него с помощью `$()` или `jQuery()` , например так: `$(ele)` . После этого мы можем вызвать любой метод jQuery для

объекта, например `css()` или `hide()` и т. Д. В этом примере мы просто вытягиваем текст объекта.

Выбор братьев и сестер

Чтобы выбрать братьев и сестер элемента, вы можете использовать метод `.siblings()` .

Типичный пример, в котором вы хотите изменить братьев и сестер элемента, находится в меню:

```
<ul class="menu">
    <li class="selected">Home</li>
    <li>Blog</li>
    <li>About</li>
</ul>
```

Когда пользователь нажимает на элемент меню, `selected` класс должен быть добавлен к элементу щелчка и удален из его *братьев и сестер* :

```
$(".menu").on("click", "li", function () {
    $(this).addClass("selected");
    $(this).siblings().removeClass("selected");
});
```

Метод принимает необязательный аргумент `selector` , который можно использовать, если вам нужно сузить типы братьев и сестер, которые вы хотите выбрать:

```
$(this).siblings("li").removeClass("selected");
```

ближайший () метод

Возвращает первый элемент, который соответствует селектору, начинающемуся с элемента, и пересекающему дерево DOM.

HTML

```
<div id="abc" class="row">
    <div id="xyz" class="row">
    </div>
    <p id="origin">
      Hello
    </p>
</div>
```

JQuery

```
var target = $('#origin').closest('.row');
console.log("Closest row:", target.attr('id') );
```

```
var target2 = $('#origin').closest('p');
console.log("Closest p:", target2.attr('id') );
```

ВЫХОД

```
"Closest row: abc"
"Closest p: origin"
```

first (): первый метод возвращает первый элемент из согласованного набора элементов.

HTML

```
<div class='.firstExample'>
   <p>This is first paragraph in a div.</p>
   <p>This is second paragraph in a div.</p>
   <p>This is third paragraph in a div.</p>
   <p>This is fourth paragraph in a div.</p>
   <p>This is fifth paragraph in a div.</p>
</div>
```

JQuery

```
var firstParagraph = $("div p").first();
console.log("First paragraph:", firstParagraph.text());
```

Выход:

```
First paragraph: This is first paragraph in a div.
```

Получить следующий элемент

Чтобы получить следующий элемент, вы можете использовать метод `.next()` .

```
<ul>
    <li>Mark</li>
    <li class="anna">Anna</li>
    <li>Paul</li>
</ul>
```

Если вы стоите на элементе «Анна» и хотите получить следующий элемент «Пол», метод `.next()` позволит вам это сделать.

```
// "Paul" now has green text
$(".anna").next().css("color", "green");
```

Метод принимает необязательный аргумент `selector` , который может использоваться, если следующий элемент должен быть определенным типом элемента.

```
// Next element is a "li", "Paul" now has green text
```

```
$(".anna").next("li").css("color", "green");
```

Если следующий элемент не относится к `selector` типа, возвращается пустой набор, и модификации ничего не сделают.

```
// Next element is not a ".mark", nothing will be done in this case
$(".anna").next(".mark").css("color", "green");
```

Получить предыдущий элемент

Чтобы получить предыдущий элемент, вы можете использовать метод `.prev()` .

```
<ul>
    <li>Mark</li>
    <li class="anna">Anna</li>
    <li>Paul</li>
</ul>
```

Если вы стоите на элементе «Анна» и хотите получить предыдущий элемент «Марк», метод `.prev()` позволит вам это сделать.

```
// "Mark" now has green text
$(".anna").prev().css("color", "green");
```

Метод принимает необязательный аргумент `selector` , который может использоваться, если предыдущий элемент должен быть определенным типом элемента.

```
// Previous element is a "li", "Mark" now has green text
$(".anna").prev("li").css("color", "green");
```

Если предыдущий элемент не относится к типу `selector` возвращается пустой набор, и модификации ничего не сделают.

```
// Previous element is not a ".paul", nothing will be done in this case
$(".anna").prev(".paul").css("color", "green");
```

Отфильтровать выделение

Чтобы отфильтровать выделение, вы можете использовать метод `.filter()` .

Метод вызывается по выбору и возвращает новый выбор. Если фильтр соответствует элементу, он добавляется к возвращенному элементу, в противном случае он игнорируется. Если ни один элемент не сопоставляется, возвращается пустой выбор.

HTML-код

Это HTML, который мы будем использовать.

```
<ul>
    <li class="zero">Zero</li>
    <li class="one">One</li>
    <li class="two">Two</li>
    <li class="three">Three</li>
</ul>
```

селектор

Фильтрация с использованием селекторов - один из самых простых способов фильтрации выбора.

```
$("li").filter(":even").css("color", "green"); // Color even elements green
$("li").filter(".one").css("font-weight", "bold"); // Make ".one" bold
```

функция

Фильтрация выделения с использованием функции полезна, если невозможно использовать селектор.

Функция вызывается для каждого элемента в выборе. Если он вернет `true` значение, то элемент будет добавлен в возвращаемый выбор.

```
var selection = $("li").filter(function (index, element) {
    // "index" is the position of the element
    // "element" is the same as "this"
    return $(this).hasClass("two");
});
selection.css("color", "green"); // ".two" will be colored green
```

элементы

Вы можете фильтровать элементы DOM. Если элементы DOM находятся в выборе, они будут включены в возвращаемый выбор.

```
var three = document.getElementsByClassName("three");
$("li").filter(three).css("color", "green");
```

выбор

Вы также можете отфильтровать выделение другим выбором. Если элемент находится в обоих вариантах выбора, он будет включен в возвращаемый выбор.

```
var elems = $(".one, .three");
```

```
$("li").filter(elems).css("color", "green");
```

метод find ()

.find () позволяет нам искать потомков этих элементов в дереве DOM и конструировать новый объект jQuery из соответствующих элементов.

HTML

```
<div class="parent">
   <div class="children" name="first">
       <ul>
           <li>A1</li>
           <li>A2</li>
           <li>A3</li>
       </ul>
   </div>
   <div class="children" name="second">
       <ul>
           <li>B1</li>
           <li>B2</li>
           <li>B3</li>
       </ul>
   </div>
  </div>
```

JQuery

```
$('.parent').find('.children[name="second"] ul li').css('font-weight','bold');
```

Выход

- A1
- A2
- A3

- **B1**
- **Би 2**
- **B3**

Прочитайте Перемещение DOM онлайн: https://riptutorial.com/ru/jquery/topic/1189/перемещение-dom

глава 12: Плагины

Examples

Плагины - Начало работы

API jQuery можно расширить, добавив к его прототипу. Например, существующий API уже имеет множество доступных функций, таких как `.hide()` , `.fadeIn()` , `.hasClass()` и т. Д.

Прототип jQuery `$.fn` через `$.fn` , исходный код содержит строку

```
jQuery.fn = jQuery.prototype
```

Добавление функций в этот прототип позволит этим функциям быть доступными для вызова из любого сконструированного объекта jQuery (который делается неявно с каждым вызовом jQuery или каждый вызов $, если вы предпочитаете).

Построенный объект jQuery будет содержать внутренний массив элементов, основанный на переданном ему селекторе. Например, `$('.active')` построит объект jQuery, который содержит элементы с активным классом во время вызова (как, впрочем, это не живой набор элементов).

`this` значение внутри функции плагин будет ссылаться на созданный объект JQuery. В результате `this` используется для представления согласованного набора.

Основной плагин :

```
$.fn.highlight = function() {
    this.css({ background: "yellow" });
};

// Use example:
$("span").highlight();
```

Пример jsFiddle

Цепь и повторное использование

В отличие от вышеприведенного примера , плагины jQuery, как ожидается, будут **Chainable** .
Это означает возможность объединения нескольких методов в одну и ту же коллекцию элементов, например `$(".warn").append("WARNING! ").css({color:"red"})` (см., Как мы использовали `.css()` после метода `.append()` , оба метода применяются к одной и той же

коллекции `.warn`)

Разрешение использовать один и тот же плагин для разных коллекций, передающих разные параметры настройки, играет важную роль в **настройке** / **повторном использовании**

```
(function($) {
  $.fn.highlight = function( custom ) {

    // Default settings
    var settings = $.extend({
        color : "",              // Default to current text color
        background : "yellow"    // Default to yellow background
    }, custom);

    return this.css({     // `return this` maintains method chainability
        color : settings.color,
        backgroundColor : settings.background
    });

  };
}( jQuery ));


// Use Default settings
$("span").highlight();    // you can chain other methods

// Use Custom settings
$("span").highlight({
    background: "#f00",
    color: "white"
});
```

jsFiddle demo

свобода

Вышеприведенные примеры относятся к пониманию создания основного плагина. Имейте в виду не ограничивать пользователя ограниченным набором параметров настройки.

Скажем, например, вы хотите построить `.highlight()` котором вы можете передать нужный **текст.** Строка, которая будет подсвечена, и позволит максимально свободу стилей:

```
//...
// Default settings
var settings = $.extend({
   text : "",          // text to highlight
   class : "highlight" // reference to CSS class
}, custom);

return this.each(function() {
   // your word highlighting logic here
});
//...
```

пользователь может теперь передать желаемый **текст** и иметь полный контроль над добавленными стилями, используя собственный класс CSS:

```
$("#content").highlight({
    text : "hello",
    class : "makeYellowBig"
});
```

Пример jsFiddle

Метод jQuery.fn.extend ()

Этот метод расширяет объект прототипа jQuery ($ .fn) для предоставления новых настраиваемых методов, которые могут быть привязаны к функции jQuery ().

Например:

```
<div>Hello</div>
<div>World!</div>

<script>
jQuery.fn.extend({
  // returns combination of div texts as a result
  getMessage: function() {
    var result;
    // this keyword corresponds result array of jquery selection
    this.each(function() {
        // $(this) corresponds each div element in this loop
        result = result + " " + $(this).val();
    });
    return result;
  }
});

// newly created .getMessage() method
var message = $("div").getMessage();

// message = Hello World!
console.log(message);
</script>
```

Прочитайте Плагины онлайн: https://riptutorial.com/ru/jquery/topic/1805/плагины

глава 13: Получение и установка ширины и высоты элемента

Examples

Получение и установка ширины и высоты (игнорирование границы)

Получить ширину и высоту:

```
var width = $('#target-element').width();
var height = $('#target-element').height();
```

Установите ширину и высоту:

```
$('#target-element').width(50);
$('#target-element').height(100);
```

Получение и установка innerWidth и innerHeight (игнорирование отступов и границ)

Получить ширину и высоту:

```
var width = $('#target-element').innerWidth();
var height = $('#target-element').innerHeight();
```

Установите ширину и высоту:

```
$('#target-element').innerWidth(50);
$('#target-element').innerHeight(100);
```

Получение и установка внешней ширины и внешней высоты (включая отступы и границы)

Получить ширину и высоту (без поля):

```
var width = $('#target-element').outerWidth();
var height = $('#target-element').outerHeight();
```

Получить ширину и высоту (включая маржу):

```
var width = $('#target-element').outerWidth(true);
var height = $('#target-element').outerHeight(true);
```

Установите ширину и высоту:

```
$('#target-element').outerWidth(50);
$('#target-element').outerHeight(100);
```

Прочитайте Получение и установка ширины и высоты элемента онлайн:
https://riptutorial.com/ru/jquery/topic/2167/получение-и-установка-ширины-и-высоты-элемента

глава 14: присоединять

Синтаксис

1. $ (Селектор) .append (содержание)
2. $ (Содержание) .appendTo (селектор)

параметры

параметры	подробности
содержание	Возможные типы: Элемент, HTML-строка, текст, массив, объект или даже функция, возвращающая строку.

замечания

- .append () & .after () может потенциально выполнить код. Это может произойти путем ввода тегов скрипта или использования атрибутов HTML, которые выполняют код (например,). Не используйте эти методы для вставки строк, полученных из ненадежных источников, таких как параметры запроса URL, файлы cookie или входные данные формы. Это может привести к уязвимостям межсайтового скриптинга (XSS). Удалите или удалите любой пользовательский ввод, прежде чем добавлять контент в документ.
- jQuery официально не поддерживает SVG. Использование методов jQuery в SVG документы, если они явно не документированы для этого метода, могут привести к неожиданному поведению. Примеры методов, поддерживающих SVG, начиная с jQuery 3.0 - это addClass и removeClass.

Examples

Добавление элемента в контейнер

Решение 1:

```
$('#parent').append($('#child'));
```

Решение 2:

```
$('#child').appendTo($('#parent'));
```

Оба решения добавляют элемент `#child` (добавление в конце) к элементу `#parent` .

До:

```
<div id="parent">
  <span>other content</span>
</div>
<div id="child">

</div>
```

После:

```
<div id="parent">
  <span>other content</span>
  <div id="child">

  </div>
</div>
```

Примечание. Когда вы добавляете контент, который уже существует в документе, это содержимое будет удалено из исходного родительского контейнера и добавлено в новый родительский контейнер. Таким образом, вы не можете использовать `.append()` или `.appendTo()` для клонирования элемента. Если вам нужен клон, используйте `.clone()` -> [http://api.jquery.com/clone/][1]

Эффективное последовательное использование .append ()

Начинающийся:

HTML

```
<table id='my-table' width='960' height='500'></table>
```

JS

```
var data = [
    { type: "Name", content: "John Doe" },
    { type: "Birthdate", content: "01/01/1970" },
    { type: "Salary", content: "$40,000,000" },
    // ...300 more rows...
    { type: "Favorite Flavour", content: "Sour" }
];
```

Добавление внутри цикла

Вы получили большой массив данных. Теперь пришло время пройти и отобразить его на странице.

Ваша первая мысль может состоять в том, чтобы сделать что-то вроде этого:

```
var i;                          // <- the current item number
var count = data.length;        // <- the total
var row;                        // <- for holding a reference to our row object

// Loop over the array
for ( i = 0; i < count; ++i ) {
    row = data[ i ];

    // Put the whole row into your table
    $('#my-table').append(
        $('<tr></tr>').append(
            $('<td></td>').html(row.type),
            $('<td></td>').html(row.content)
        )
    );
}
```

Это *совершенно верно* и сделает то, что вы ожидаете, но ...

Не делайте этого.

Помните эти **300 +** строк данных?

Каждый из них заставит браузер повторно рассчитать значения ширины, высоты и позиционирования каждого элемента вместе с любыми другими стилями - если они не разделены границей макета , которые, к сожалению, для этого примера (поскольку они являются потомками элемента `<table>`), они не могут.

В небольших количествах и в нескольких столбцах этот штраф за исполнение, безусловно, будет незначительным. Но мы хотим, чтобы каждая миллисекунда считалась.

Лучшие варианты

1. Добавьте в отдельный массив, добавьте после завершения цикла

```
/**
 * Repeated DOM traversal (following the tree of elements down until you reach
 * what you're looking for - like our <table>) should also be avoided wherever possible.
 */

// Keep the table cached in a variable then use it until you think it's been removed
var $myTable = $('#my-table');
```

```
// To hold our new <tr> jQuery objects
var rowElements = [];

var count = data.length;
var i;
var row;

// Loop over the array
for ( i = 0; i < count; ++i ) {
    rowElements.push(
        $('<tr></tr>').append(
            $('<td></td>').html(row.type),
            $('<td></td>').html(row.content)
        )
    );
}

// Finally, insert ALL rows at once
$myTable.append(rowElements);
```

Из этих опций этот больше всего зависит от jQuery.

---

2. Использование современных методов Array. *

```
var $myTable = $('#my-table');

// Looping with the .map() method
//  - This will give us a brand new array based on the result of our callback function
var rowElements = data.map(function ( row ) {

    // Create a row
    var $row = $('<tr></tr>');

    // Create the columns
    var $type = $('<td></td>').html(row.type);
    var $content = $('<td></td>').html(row.content);

    // Add the columns to the row
    $row.append($type, $content);

    // Add to the newly-generated array
    return $row;
});

// Finally, put ALL of the rows into your table
$myTable.append(rowElements);
```

Функционально эквивалентный тому, который был до него, только легче читать.

---

3. Использование строк HTML (вместо встроенных методов jQuery)

```
// ...
var rowElements = data.map(function ( row ) {
    var rowHTML = '<tr><td>';
    rowHTML += row.type;
    rowHTML += '</td><td>';
    rowHTML += row.content;
    rowHTML += '</td></tr>';
    return rowHTML;
});

// Using .join('') here combines all the separate strings into one
$myTable.append(rowElements.join(''));
```

Отлично, но снова, **не рекомендуется** . Это заставляет jQuery анализировать очень большой объем текста одновременно и не требуется. jQuery очень хорош в том, что он делает при правильном использовании.

---

4. Вручную создавать элементы, добавлять к фрагменту документа

```
var $myTable = $(document.getElementById('my-table'));

/**
 * Create a document fragment to hold our columns
 * - after appending this to each row, it empties itself
 *   so we can re-use it in the next iteration.
 */
var colFragment = document.createDocumentFragment();

/**
 * Loop over the array using .reduce() this time.
 * We get a nice, tidy output without any side-effects.
 *  - In this example, the result will be a
 *    document fragment holding all the <tr> elements.
 */
var rowFragment = data.reduce(function ( fragment, row ) {

    // Create a row
    var rowEl = document.createElement('tr');

    // Create the columns and the inner text nodes
    var typeEl = document.createElement('td');
    var typeText = document.createTextNode(row.type);
    typeEl.appendChild(typeText);

    var contentEl = document.createElement('td');
    var contentText = document.createTextNode(row.content);
    contentEl.appendChild(contentText);

    // Add the columns to the column fragment
    // - this would be useful if columns were iterated over separately
    //   but in this example it's just for show and tell.
    colFragment.appendChild(typeEl);
    colFragment.appendChild(contentEl);
```

```
    rowEl.appendChild(colFragment);

    // Add rowEl to fragment - this acts as a temporary buffer to
    // accumulate multiple DOM nodes before bulk insertion
    fragment.appendChild(rowEl);

    return fragment;
}, document.createDocumentFragment());

// Now dump the whole fragment into your table
$myTable.append(rowFragment);
```

Мой личный фаворит . Это иллюстрирует общее представление о том, что делает jQuery на более низком уровне.

Погружение глубже

- Средство просмотра jQuery
- Array.prototype.join ()
- Array.prototype.map ()
- Array.prototype.reduce ()
- document.createDocumentFragment ()
- document.createTextNode ()
- Основы Google Web - производительность

jQuery append

HTML

```
<p>This is a nice </p>
<p>I like </p>

<ul>
  <li>List item 1</li>
  <li>List item 2</li>
  <li>List item 3</li>
</ul>

<button id="btn-1">Append text</button>
<button id="btn-2">Append list item</button>
```

скрипт

```
$("#btn-1").click(function(){
    $("p").append(" <b>Book</b>.");
});
$("#btn-2").click(function(){
    $("ul").append("<li>Appended list item</li>");
 });
});
```

Прочитайте присоединять онлайн: https://riptutorial.com/ru/jquery/topic/1910/присоединять

глава 15: Селекторы

Вступление

Селектор jQuery выбирает или находит элемент DOM (объект объектной модели) в документе HTML. Он используется для выбора элементов HTML на основе идентификатора, имени, типов, атрибутов, класса и т. Д. Он основан на существующих селекторах CSS.

Синтаксис

- Тег: нет маркера, используйте тег напрямую
- Id: `#id`
- Класс: `.className`
- Атрибут: `[attributeName]`
- Атрибут со значением: `[attributeName ='value']`
- Атрибут начинается со значения `^=` : `[attributeName ^= 'value']`
- Атрибут заканчивается значением `$=` : `[attributeName $='value']`
- Атрибут содержит значение `*=` : `[attributeName *= 'value']`
- Псевдоселектор:: `:pseudo-selector`
- Любой потомок: Пробел между селекторами
- Прямые дети: `>` между селекторами
- Близлежащий брат, следующий за первым: `+`
- Несмещенный родной брат после первого: `~`
- Или: `,` (запятая) между селектором

замечания

При написании `selectors` для `class` или `id` или `attribute` который содержит некоторые специальные символы, такие как

```
! " # $ % & ' ( ) * + , . / : ; < = > ? @ [ \ ] ^ { | } ~
```

символы должны быть экранированы с использованием двух обратных косых черт `\\` .

например.

```
<span id="temp.foobar"><span>
```

селектора будут обрамлены, как,

```
$('#temp\\.foobar')
```

Examples

Типы переключателей

В jQuery вы можете выбирать элементы на странице, используя множество различных свойств элемента, включая:

- Тип
- Учебный класс
- Я БЫ
- Владение атрибутом
- Значение атрибута
- Индексированный селектор
- Псевдо-состояние

Если вы знаете CSS-селектора, вы заметите, что селектора в jQuery одинаковы (с незначительными исключениями).

Возьмем следующий HTML-код:

```
<a href="index.html"></a>                  <!-- 1 -->
<a id="second-link"></a>                   <!-- 2 -->
<a class="example"></a>                    <!-- 3 -->
<a class="example" href="about.html"></a>  <!-- 4 -->
<span class="example"></span>              <!-- 5 -->
```

Выбор по типу:

Следующий селектор jQuery выберет все элементы `<a>` , включая 1, 2, 3 и 4.

```
$("a")
```

Выбор по классам

Следующий селектор jQuery выберет все элементы `example` класса (включая не-элементы), которые представляют собой 3, 4 и 5.

```
$(".example")
```

Выбор по идентификатору

Следующий селектор jQuery выберет элемент с данным идентификатором, который равен 2.

```
$("#second-link")
```

Выбор по владению атрибутом

Следующий селектор jQuery выберет все элементы с определенным атрибутом `href` , включая 1 и 4.

```
$("[href]")
```

Выбор по значению атрибута

Следующий селектор jQuery выберет все элементы, где существует атрибут `href` со значением `index.html` , который равен только 1.

```
$("[href='index.html']")
```

Выбор по индексированной позиции (*индексированный селектор*)

Следующий селектор jQuery будет выбирать только 1, второй `<a>` ie. `second-link` поскольку указанный индекс равен `1` например, `eq(1)` (обратите внимание, что индекс начинается с `0` следовательно, второй выбран здесь!).

```
$("a:eq(1)")
```

Выбор с индексированным исключением

Чтобы исключить элемент, используя его индекс `:not(:eq())`

Следующий выбирает `<a>` элементы, за исключением того, что с `example` класса, который равен 1

```
$("a").not(":eq(0)")
```

Выбор с исключением

Чтобы исключить элемент из выделения, используйте `:not()`

Следующие элементы `<a>` , кроме тех, что указаны в `example` класса, равны 1 и 2.

```
$("a:not(.example)")
```

Выбор по псевдо-состоянию

Вы также можете выбрать в jQuery, используя псевдо-состояния, включая `:first-child` `:last-child` `:first-of-type` `:last-of-type` и т. Д.

Следующий селектор jQuery выберет только первый элемент `<a>` : номер 1.

```
$("a:first-of-type")
```

Объединение селекторов jQuery

Вы также можете увеличить свою специфичность, объединив несколько селекторов jQuery; вы можете комбинировать любое их количество или объединить их все. Вы также можете одновременно выбрать несколько классов, атрибутов и состояний.

```
$("a.class1.class2.class3#someID[attr1][attr2='something'][attr3='something']:first-of-type:first-child")
```

Это позволит выбрать элемент `<a>` :

- Имеет следующие классы: `class1, class2, and class3`
- Имеет следующий идентификатор: `someID`
- Имеет следующий атрибут: `attr1`
- Имеет следующие атрибуты и значения: `attr2` со значением `something` , `attr3` со значением `something`
- Имеет следующие состояния: `first-child` и `first-of-type`

Вы также можете отделить разные селекторы запятой:

```
$("a, .class1, #someID")
```

Это позволит выбрать:

- Все элементы `<a>`
- Все элементы, имеющие класс `class1`
- Элемент с идентификатором id `#someID`

Выбор ребенка и родного брата

Селекторы jQuery обычно соответствуют тем же соглашениям, что и CSS, что позволяет вам выбирать детей и братьев и сестер таким же образом.

- Чтобы выбрать непрямого ребенка, используйте пробел
- Чтобы выбрать прямой дочерний элемент, используйте команду `>`
- Чтобы выбрать соседнего брата, следующего за первым, используйте `+`
- Чтобы выбрать несмежный брат, следующий за первым, используйте `~`

Выбор подстановочных знаков

Могут быть случаи, когда мы хотим выбрать все элементы, но не существует общего свойства для выбора (класс, атрибут и т. Д.). В этом случае мы можем использовать селектор `*` , который просто выбирает все элементы:

```
$('#wrapper *')    // Select all elements inside #wrapper element
```

Объединение селекторов

Рассмотрим следующую структуру DOM

```
<ul class="parentUl">
    <li> Level 1
        <ul class="childUl">
            <li>Level 1-1 <span> Item - 1 </span></li>
            <li>Level 1-1 <span> Item - 2 </span></li>
        </ul>
    </li>
    <li> Level 2
        <ul class="childUl">
            <li>Level 2-1 <span> Item - 1 </span></li>
            <li>Level 2-1 <span> Item - 1 </span></li>
        </ul>
    </li>
</ul>
```

Селекторы потомков и детей

Если родительский `<ul>` - `parentUl` находит своих потомков (`<li>`),

1. Простой **`$('parent child')`**

   >> `$('ul.parentUl li')`

   Это приведет к тому, что все соответствующие потомки указанного предка будут *опущены на все уровни* .

2. `>` - **`$('parent > child')`**

   >> `$('ul.parentUl > li')`

   Это находит всех соответствующих детей (*только 1 уровень вниз*).

3. Контекстный селектор - **`$('child','parent')`**

   >> `$('li','ul.parentUl')`

   Это работает так же, как и выше.

4. `find()` - **`$('parent').find('child')`**

   >> `$('ul.parentUl').find('li')`

   Это работает так же, как и выше.

5. `children()` - **`$('parent').find('child')`**

>> `$('ul.parentUl').children('li')`

Это работает так же, как и выше.

Другие комбинаторы

Селектор групп: ","

Выберите все элементы `<ul>` И все элементы `<li>` И все элементы `<span>` :

```
$('ul, li, span')
```

Селектор мультипликации: "" (без символа)

Выделите все элементы `<ul>` с классом `parentUl` :

```
$('ul.parentUl')
```

Смежный селектор: «+»

Выделите все элементы `<li>` , которые помещаются сразу после другого элемента `<li>` :

```
$('li + li')
```

Общий селектор сиблинга: "~"

Выберите все элементы `<li>` которые являются братьями и сестрами других элементов `<li>` :

```
$('li ~ li')
```

обзор

Элементы могут быть выбраны jQuery с помощью селекторов jQuery . Функция возвращает либо элемент, либо список элементов.

Основные селекторы

```
$("*")                      // All elements
$("div")                    // All <div> elements
$(".blue")                  // All elements with class=blue
$(".blue.red")              // All elements with class=blue AND class=red
$(".blue,.red")             // All elements with class=blue OR class=red
```

```
$("#headline")               // The (first) element with id=headline
$("[href]")                  // All elements with an href attribute
$("[href='example.com']")    // All elements with href=example.com
```

Операторы отношения

```
$("div span")          // All <span>s that are descendants of a <div>
$("div > span")        // All <span>s that are a direct child of a <div>
$("a ~ span")          // All <span>s that are siblings following an <a>
$("a + span")          // All <span>s that are immediately after an <a>
```

Кэширование Селекторы

Каждый раз, когда вы используете селектор в jQuery, DOM ищет элементы, соответствующие вашему запросу. Выполнение этого слишком часто или неоднократно снижает производительность. Если вы ссылаетесь на определенный селектор более одного раза, вы должны добавить его в кэш, назначив его переменной:

```
var nav = $('#navigation');
nav.show();
```

Это заменит:

```
$('#navigation').show();
```

Кэширование этого селектора может оказаться полезным, если вашему веб-сайту часто приходится показывать / скрывать этот элемент. Если имеется несколько элементов с одним и тем же селектором, переменная станет массивом этих элементов:

```
<div class="parent">
    <div class="child">Child 1</div>
    <div class="child">Child 2</div>
</div>

<script>
    var children = $('.child');
    var firstChildText = children[0].text();
    console.log(firstChildText);

    // output: "Child 1"
</script>
```

ПРИМЕЧАНИЕ . Элемент должен существовать в DOM во время его назначения переменной. Если в DOM нет элемента с классом `child` вы будете хранить пустой массив в этой переменной.

```
<div class="parent"></div>

<script>
```

```
    var parent   = $('.parent');
    var children = $('.child');
    console.log(children);

    // output: []

    parent.append('<div class="child">Child 1</div>');
    children = $('.child');
    console.log(children[0].text());

    // output: "Child 1"
</script>
```

Не забудьте переназначить селектор переменной после добавления / удаления элементов в DOM с помощью этого селектора.

Примечание . При кешировании селекторов многие разработчики запустит имя переменной с помощью `$` чтобы обозначить, что переменная является объектом jQuery следующим образом:

```
var $nav = $('#navigation');
$nav.show();
```

Элементы DOM в качестве селекторов

jQuery принимает множество параметров, и один из них является фактическим элементом DOM. Передача элемента DOM в jQuery приведет к тому, что базовая структура массива объекта jQuery удерживает этот элемент.

jQuery обнаружит, что аргумент является элементом DOM, проверяя его nodeType.

Наиболее частое использование элемента DOM происходит в обратных вызовах, где текущий элемент передается конструктору jQuery, чтобы получить доступ к API jQuery.

Например, в `each` обратном вызове (примечание: каждая функция итератора).

```
$(".elements").each(function(){
    //the current element is bound to `this` internally by jQuery when using each
    var currentElement = this;

    //at this point, currentElement (or this) has access to the Native API

    //construct a jQuery object with the currentElement(this)
    var $currentElement = $(this);

    //now $currentElement has access to the jQuery API
});
```

Строки HTML как селекторы

jQuery принимает множество параметров как «селекторов», а одна из них - строка HTML.

Передача строки в jQuery приведет к тому, что базовая структура объекта jQuery, подобная массиву, сохранит полученный построенный HTML.

jQuery использует regex, чтобы определить, является ли строка, передаваемая конструктору, строкой HTML, а также что она *должна* начинаться с `<` . Это регулярное выражение определяется как `rquickExpr = /^(?:\s*(<[\w\W]+>)[^>]*|#([\w-]*))$/` (пояснение в regex101.com).

Наиболее частое использование строки HTML в качестве селектора - это когда набор элементов DOM необходимо создавать только в коде, часто это используется библиотеками для таких вещей, как всплывающие окна Modal.

Например, функция, которая вернула привязку, завернутую в div как шаблон

```
function template(href,text){
    return $("<div><a href='" + href + "'>" + text + "</a></div>");
}
```

Вернул бы объект jQuery

```
<div>
    <a href="google.com">Google</a>
</div>
```

если он называется `template("google.com","Google")` .

Прочитайте Селекторы онлайн: https://riptutorial.com/ru/jquery/topic/389/селекторы

глава 16: событие, подготовленное документами

Examples

Что готово для документа и как его использовать?

Код jQuery часто заверяется в `jQuery(function($) { ... });` так что он запускается только после завершения загрузки DOM.

```
<script type="text/javascript">
  jQuery(function($) {
    // this will set the div's text to "Hello".
    $("#myDiv").text("Hello");
  });
</script>

<div id="myDiv">Text</div>
```

Это важно, потому что jQuery (и JavaScript вообще) не может выбрать элемент DOM, который не был отображен на странице.

```
<script type="text/javascript">
  // no element with id="myDiv" exists at this point, so $("#myDiv") is an
  // empty selection, and this will have no effect
  $("#myDiv").text("Hello");
</script>

<div id="myDiv">Text</div>
```

Обратите внимание: вы можете псевдоним пространства имен jQuery путем передачи пользовательского обработчика в метод `.ready()` . Это полезно для случаев, когда другая библиотека JS использует один и тот же сокращенный псевдоним `$` *jQuery* , который создает конфликт. Чтобы избежать этого конфликта, вы должны вызвать `$.noConflict();` - Это заставляет использовать только пространство имен *jQuery* по умолчанию (вместо коротких `$` alias).
Передав пользовательский обработчик обработчику `.ready()` , вы сможете выбрать имя псевдонима для использования *jQuery* .

```
$.noConflict();

jQuery( document ).ready(function( $ ) {
   // Here we can use '$' as jQuery alias without it conflicting with other
   // libraries that use the same namespace
   $('body').append('<div>Hello</div>')
});
```

```
jQuery( document ).ready(function( jq ) {
   // Here we use a custom jQuery alias 'jq'
   jq('body').append('<div>Hello</div>')
});
```

Вместо того, чтобы просто помещать код jQuery в нижнюю часть страницы, использование функции `$(document).ready` гарантирует, что все HTML-элементы были отображены, и вся документальная модель документа (DOM) готова для выполнения кода JavaScript.

jQuery 2.2.3 и ранее

Все они эквивалентны, код внутри блоков будет работать, когда документ будет готов:

```
$(function() {
  // code
});

$().ready(function() {
  // code
});

$(document).ready(function() {
  // code
});
```

Поскольку они эквивалентны, первая является рекомендуемой формой, следующая версия - это ключевое слово `jQuery` вместо `$` которое дает те же результаты:

```
jQuery(function() {
  // code
});
```

jQuery 3.0

нотация

Начиная с jQuery 3.0 рекомендуется только эта форма:

```
jQuery(function($) {
  // Run when document is ready
  // $ (first argument) will be internal reference to jQuery
  // Never rely on $ being a reference to jQuery in the global namespace
});
```

Все остальные обработчики документа устарели в jQuery 3.0.

Асинхронный

Начиная с jQuery 3.0, готовый обработчик всегда будет вызываться асинхронно . Это

означает, что в приведенном ниже коде журнал «внешний обработчик» всегда будет отображаться первым, независимо от того, готов ли документ к точке выполнения.

```
$(function() {
  console.log("inside handler");
});
console.log("outside handler");
```

> наружный обработчик
> внутренний обработчик

Разница между $ (document) .ready () и $ (window) .load ()

`$(window).load()` **устарел в jQuery версии 1.8 (и полностью удален из jQuery 3.0)** и как таковой больше не должен использоваться. Причины отмены указаны на странице jQuery об этом событии

> Предостережения о событии загрузки при использовании с изображениями
>
> Общей проблемой, которую разработчики пытаются решить с помощью `.load()` является выполнение функции, когда изображение (или коллекция изображений) полностью загружено. Есть несколько известных оговорок с этим, что следует отметить. Это:
>
> - Он не работает последовательно и не надежно кросс-браузер
> - Он не срабатывает корректно в WebKit, если изображение `src` установлено на тот же `src` что и раньше
> - Он неправильно выравнивает дерево DOM
> - Может перестать запускаться для изображений, которые уже хранятся в кеше браузера

Если вы все еще хотите использовать `load()` это описано ниже:

`$(document).ready()` ждет, пока не будет доступна полная DOM - все элементы в HTML были проанализированы и находятся в документе. Однако на данный момент ресурсы, такие как изображения, могут не быть полностью загружены. Если важно подождать, пока все ресурсы будут загружены, `$(window).load()` **и вы будете знать о значительных ограничениях этого события,** то вместо этого можно использовать следующее:

```
$(document).ready(function() {
  console.log($("#my_large_image").height()); // may be 0 because the image isn't available
});

$(window).load(function() {
  console.log($("#my_large_image").height()); // will be correct
});
```

Прикрепление событий и управление DOM внутри ready ()

Пример использования `$(document).ready()` :

1. **Присоединение обработчиков событий**
   Присоединить обработчики событий jQuery

```
$(document).ready(function() {
  $("button").click(function() {
    // Code for the click function
  });
});
```

2. **Запустить код jQuery после создания структуры страницы**

```
jQuery(function($) {
// set the value of an element.
   $("#myElement").val("Hello");
});
```

3. **Манипулировать загруженную структуру DOM**
   Например: скрыть `div` когда страница загружается в первый раз и отображать ее в событии нажатия кнопки

```
$(document).ready(function() {
  $("#toggleDiv").hide();
  $("button").click(function() {
    $("#toggleDiv").show();
  });
});
```

Разница между jQuery (fn) и выполнением кода перед

Использование события, готового к документу, может иметь небольшие недостатки производительности с задержкой до 300 мс. Иногда такое же поведение может быть достигнуто путем выполнения кода непосредственно перед закрывающим `</body>` :

```
<body>
  <span id="greeting"></span> world!
  <script>
    $("#greeting").text("Hello");
  </script>
</body>
```

приведет к аналогичному поведению, но будет выполняться раньше, чем не дожидаться запуска события события, как это происходит в:

```
<head>
  <script>
    jQuery(function($) {
```

```
      $("#greeting").text("Hello");
    });
  </script>
</head>
<body>
  <span id="greeting"></span> world!
</body>
```

Акцент на том, что первый пример основан на ваших знаниях вашей страницы и размещении сценария непосредственно перед закрывающим `</body>` и, в частности, после тега `span` .

Прочитайте событие, подготовленное документами онлайн:
https://riptutorial.com/ru/jquery/topic/500/событие--подготовленное-документами

глава 17: События

замечания

jQuery внутренне обрабатывает события через функцию addEventListener . Это означает, что совершенно законно иметь более одной функции, связанной с одним и тем же событием для одного и того же элемента DOM.

Examples

Прикрепить и отсоединить обработчики событий

Прикрепить обработчик событий

Начиная с версии 1.7 jQuery имеет API событий `.on()` . Таким образом, любое стандартное событие javascript или настраиваемое событие может быть привязано к выбранному в данный момент элементу jQuery. Есть ярлыки, такие как `.click()` , но `.on()` дает вам больше опций.

HTML

```
<button id="foo">bar</button>
```

JQuery

```
$( "#foo" ).on( "click", function() {
  console.log( $( this ).text() ); //bar
});
```

Отсоединить обработчик событий

Естественно, у вас есть возможность отключать события от объектов jQuery. Вы делаете это с помощью `.off( events [, selector ] [, handler ] )` .

HTML

```
<button id="hello">hello</button>
```

JQuery

```
$('#hello').on('click', function(){
    console.log('hello world!');
    $(this).off();
});
```

При нажатии кнопки `$(this)` будет ссылаться на текущий объект jQuery и удалит из него все прикрепленные обработчики событий. Вы также можете указать, какой обработчик должен быть удален.

JQuery

```
$('#hello').on('click', function(){
    console.log('hello world!');
    $(this).off('click');
});

$('#hello').on('mouseenter', function(){
    console.log('you are about to click');
});
```

В этом случае событие `mouseenter` будет функционировать после нажатия.

Делегированные события

Начнем с примера. Вот очень простой пример HTML.

Пример HTML

```
<html>
    <head>
    </head>
    <body>
        <ul>
            <li>
                <a href="some_url/">Link 1</a>
            </li>
            <li>
                <a href="some_url/">Link 2</a>
            </li>
            <li>
                <a href="some_url/">Link 3</a>
            </li>
        </ul>
    </body>
</html>
```

Эта проблема

Теперь в этом примере мы хотим добавить прослушиватель событий ко всем элементам `<a>` . Проблема в том, что список в этом примере является динамическим. Элементы `<li>` добавляются и удаляются с течением времени. Тем не менее, страница не обновляется между изменениями, что позволит нам использовать простые прослушиватели событий щелчка для объектов ссылок (например, `$('a').click()`).

Проблема заключается в том, как добавлять события в элементы `<a>` которые приходят и уходят.

Фоновая информация - распространение событий

Делегированные события возможны только из-за распространения событий (часто называемых пузырьками событий). Каждый раз, когда запускается какое-либо событие, он все время будет пузыриться (до корня документа). Они *делегируют* обработку события не изменяющемуся элементу-предшественнику, отсюда и название «делегированные» события.

Итак, в приведенном выше примере, нажатие ссылки `<a>` element приведет к событию « click» в этих элементах в следующем порядке:

-
- литий
- уль
- тело
- HTML
- корень документа

Решение

Зная, что происходит в пузырьках событий, мы можем поймать одно из необходимых событий, которые распространяются через наш HTML.

Хорошим местом для этого в этом примере является элемент `<ul>` , поскольку этот элемент не является динамическим:

```
$('ul').on('click', 'a', function () {
```

```
  console.log(this.href); // jQuery binds the event function to the targeted DOM element
                          // this way `this` refers to the anchor and not to the list
  // Whatever you want to do when link is clicked
});
```

В приведенном выше:

- У нас есть «ul», который является получателем этого прослушивателя событий
- Первый параметр («click») определяет, какие события мы пытаемся обнаружить.
- Второй параметр ('a') используется, чтобы объявить, откуда должно *происходить* событие (из всех дочерних элементов этого получателя события, ul).
- Наконец, третий параметр - это код, который выполняется, если выполняются требования первого и второго параметров.

Подробнее о том, как работает решение

1. Пользователь кликает элемент `<a>`
2. Это вызывает событие click на элементе `<a>` .
3. Событие начинается в направлении к корню документа.
4. Событие пузырится сначала на элемент `<li>` а затем на элемент `<ul>` .
5. Слушатель событий запускается, поскольку элемент `<ul>` имеет подключенный прослушиватель событий.
6. Слушатель событий сначала обнаруживает событие запуска. Событие барботирования - это «щелчок», и слушатель имеет «щелчок», это пропуск.
7. Проверки прослушивателя пытаются сопоставить второй параметр ('a') с каждым элементом в цепочке пузырьков. Поскольку последний элемент в цепочке - это «a», это соответствует фильтру, и это тоже пропуск.
8. Код в третьем параметре выполняется с использованием согласованного пункта , как это `this` . Если функция не включает вызов `stopPropagation()` , событие будет продолжать распространяться вверх к корню (`document`).

Примечание. Если подходящий не изменяющийся предок недоступен / удобен, вы должны использовать `document` . Как привычка не используют `'body'` по следующим причинам:

- `body` есть ошибка, связанная с стилем, что может означать, что события мыши не пузырятся на нем. Это зависит от браузера и может произойти, когда расчетная высота тела равна 0 (например, когда все дочерние элементы имеют абсолютные позиции). События мыши всегда `document` .
- `document` *всегда* существует для вашего скрипта, поэтому вы можете прикреплять делегированные обработчики для `document` за пределами *обработчика DOM-ready* и быть уверенным, что они все равно будут работать.

Загрузка документа .load ()

Если вы хотите, чтобы ваш сценарий дождался загрузки определенного ресурса, например изображения или PDF, вы можете использовать `.load()` , который является ярлыком для ярлыка для `.on( "load", handler)` .

HTML

```
<img src="image.jpeg" alt="image" id="image">
```

JQuery

```
$( "#image" ).load(function() {
  // run script
});
```

События для повторения элементов без использования идентификаторов

проблема

На странице есть серия повторяющихся элементов, которые вам нужно знать, какое событие произошло, чтобы что-то сделать с этим конкретным экземпляром.

Решение

- Дайте всем общим элементам общий класс
- Применить прослушиватель событий к классу. `this` обработчик внутренних событий является элементом выбора соответствия, произошедшее на
- Перейдите к внешнему самому повторяющему контейнеру для этого экземпляра, начиная с `this`
- Используйте `find()` в этом контейнере, чтобы изолировать другие элементы, специфичные для этого экземпляра

HTML

```
<div class="item-wrapper" data-item_id="346">
   <div class="item"><span class="person">Fred</span></div>
   <div class="item-toolbar">
      <button class="delete">Delete</button>
   </div>
</div>
<div class="item-wrapper" data-item_id="393">
   <div clss="item"><span class="person">Wilma</span></div>
   <div class="item-toolbar">
      <button class="delete">Delete</button>
   </div>
</div>
```

JQuery

```
$(function() {
  $('.delete').on('click', function() {
    // "this" is element event occured on
    var $btn = $(this);
    // traverse to wrapper container
    var $itemWrap = $btn.closest('.item-wrapper');
    // look within wrapper to get person for this button instance
    var person = $itemWrap.find('.person').text();
    // send delete to server and remove from page on success of ajax
    $.post('url/string', { id: $itemWrap.data('item_id')}).done(function(response) {
      $itemWrap.remove()
    }).fail(function() {
      alert('Ooops, not deleted at server');
    });
  });
});
```

originalEvent

Иногда будут свойства, недоступные в событии jQuery. Для доступа к основным свойствам используйте `Event.originalEvent`

Получить направление прокрутки

```
$(document).on("wheel",function(e){
    console.log(e.originalEvent.deltaY)
    // Returns a value between -100 and 100 depending on the direction you are scrolling
})
```

Включение и отключение определенных событий через jQuery. (Именованные слушатели)

Иногда вы хотите отключить всех ранее зарегистрированных слушателей.

```
//Adding a normal click handler
$(document).on("click",function(){
    console.log("Document Clicked 1")
});
//Adding another click handler
$(document).on("click",function(){
    console.log("Document Clicked 2")
});
//Removing all registered handlers.
$(document).off("click")
```

Проблема с этим методом заключается в том, что ВСЕ слушатели, привязанные к `document` другими плагинами и т. Д., Также будут удалены.

Чаще всего, мы хотим отделить всех слушателей, прикрепленных только нами.

Чтобы достичь этого, мы можем связать именованных слушателей,

```
//Add named event listener.
$(document).on("click.mymodule",function(){
    console.log("Document Clicked 1")
});
$(document).on("click.mymodule",function(){
    console.log("Document Clicked 2")
});

//Remove named event listener.
$(document).off("click.mymodule");
```

Это гарантирует, что любой другой прослушиватель кликов не будет непреднамеренно изменен.

Прочитайте События онлайн: https://riptutorial.com/ru/jquery/topic/1321/события

глава 18: Флажок Выбрать все с автоматической проверкой / снятием флажка с другого изменения флажка

Вступление

Я использовал различные примеры и ответы Stackoverflow, чтобы прийти к этому действительно простому примеру о том, как управлять флажком «Выбрать все» в сочетании с автоматической проверкой / снятием флажка, если изменяется статус группы. Ограничение: идентификатор «select all» должен соответствовать именам ввода для создания группы выбора. В этом примере входным элементом выбора всех ID является cbGroup1. Именами ввода являются также cbGroup1

Код очень короткий, а не достаточно, если заявление (время и ресурс).

Examples

2 выберите все флажки с соответствующими ячейками группы

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>

<p>
<input id="cbGroup1" type="checkbox">Select all
<input name="cbGroup1" type="checkbox" value="value1_1">Group1 value 1
<input name="cbGroup1" type="checkbox" value="value1_2">Group1 value 2
<input name="cbGroup1" type="checkbox" value="value1_3">Group1 value 3
</p>
<p>
<input id="cbGroup2" type="checkbox">Select all
<input name="cbGroup2" type="checkbox" value="value2_1">Group2 value 1
<input name="cbGroup2" type="checkbox" value="value2_2">Group2 value 2
<input name="cbGroup2" type="checkbox" value="value2_3">Group2 value 3
</p>

<script type="text/javascript" language="javascript">
    $("input").change(function() {
      $('input[name=\''+this.id+'\']').not(this).prop('checked', this.checked);
      $('#'+this.name).prop('checked', $('input[name=\''+this.name+'\']').length ===
$('input[name=\''+this.name+'\']').filter(':checked').length);
    });
</script>
```

Прочитайте Флажок Выбрать все с автоматической проверкой / снятием флажка с другого изменения флажка онлайн: https://riptutorial.com/ru/jquery/topic/10076/флажок-выбрать-все-с-автоматической-проверкой---снятием-флажка-с-другого-изменения-флажка

кредиты

S. No	Главы	Contributors
1	Начало работы с jQuery	A.J, acdcjunior, amflare, Anil, bwegs, Community, DGS, empiric, Fueled By Coffee, hairboat, Hirshy, Iceman, Igor Raush, J F, jkdev, John C, Kevin B, Kevin Katzke, Kevin Montrose, Luca Putzu, Matas Vaitkevicius, mico, Mottie, Neal, ni8mr, Prateek, RamenChef, Rion Williams, Roko C. Buljan, secelite, Shaunak D, Stephen Leppik, Suganya, Sverri M. Olsen, the12, Travis J, user2314737, Velocibadgery, Yosvel Quintero
2	Ajax	Alon Eitan, amflare, Andrew Brooke, Ashkan Mobayen Khiabani, Athafoud, atilacamurca, Ben H, Cass, Community, csbarnes, Dr. J. Testington, Edathadan Chief aka Arun, empiric, hasan, joe_young, John C, kapantzak, Kiren Siva, Lacrioque, Marimba, Nirav Joshi, Ozan, shaN, Shaunak D, Teo Dragovic, Yosvel Quintero
3	jQuery Отложенные объекты и обещания	Alex, Ashiquzzaman, Gone Coding
4	Prepend	Ashkan Mobayen Khiabani, empiric, J F, Pranav C Balan
5	Атрибуты	A.J, acdcjunior, ban17, ochi, Scimonster
6	Видимость элементов	Alex Char, Paul Roub, Rupali Pemare, The_Outsider, Theodore K., user2314737, Zaz
7	Каждая функция	bipon, Renier
8	Манипуляция CSS	abaracedo, Ashkan Mobayen Khiabani, Brandt Solovij, J F, j08691, Jonathan Michalik, Kevin B, Petroff, Roko C. Buljan, ScientiaEtVeritas, Shlomi Haver, Sorangwala Abbasali, Stephen Leppik, Sverri M. Olsen
9	Манипуляция DOM	Angelos Chalaris, Assimilater, Brock Davis, DawnPaladin, DefyGravity, Deryck, Marimba, Mark Schultheiss, martincarlin87, Neal, Paul Roub, Shaunak D, still_learning
10	Метод jQuery .animate ()	RamenChef, Rust in Peace, Simplans, VJS
11	Перемещение DOM	A.J, charlietfl, Community, dlsso, mark.hch, rmondesilva, SGS

		Venkatesh, sucil, Sverri M. Olsen, The_Outsider, Zaz
12	Плагины	hasan, Roko C. Buljan, Travis J
13	Получение и установка ширины и высоты элемента	Ashkan Mobayen Khiabani
14	присоединять	Ashkan Mobayen Khiabani, bipon, Community, Darshak, Deryck , empiric, Flyer53, J F, JF it, Paul Roub, Pranav C Balan, Proto, Shaunak D
15	Селекторы	alepeino, Alon Eitan, Brock Davis, Castro Roy, David, DelightedD0D, devlin carnate, dlsso, hasan, Iceman, James Donnelly, JLF, John Slegers, kapantzak, Kevin B, Keyslinger, Matas Vaitkevicius, Melanie, MikeC, Nux, Petr R., rbashish, Scimonster, Shaunak D, Shekhar Pankaj, Sorangwala Abbasali, ssb, Sverri M. Olsen, Travis J, whales, WOUNDEDStevenJones , Zaz
16	событие, подготовленное документами	Adjit, Alon Eitan, amflare, charlietfl, Emanuel Vintilă, Igor Raush, J F, jkdev, Joram van den Boezem, Liam, Mark Schultheiss, Melanie, Nhan, Nico Westerdale, Scimonster, secelite, TheDeadMedic, the-noob, URoy
17	События	Adjit, charlietfl, DelightedD0D, doydoy44, empiric, Gone Coding, Horst Jahns, Jatniel Prinsloo, Kevin B, Louis, Luca Putzu, Marimba, NotJustin, SGS Venkatesh, Stephen Leppik, Sunny R Gupta, Washington Guedes, WMios, Zakaria Acharki
18	Флажок Выбрать все с автоматической проверкой / снятием флажка с другого изменения флажка	user1851673