**Тема 10. Введение в jQuery. Выборка элементов. Работа со структурой страницы**

1. **Начало работы с jQuery.**
2. **Функция jQuery.**
3. **Cелекторы.**
4. **Фильтры jQuery.**
5. **Работа с выборкой.**
6. **Фильтрация выборки.**
7. **Поиск в выборке.**
8. **Навигация по странице.**
9. **Атрибуты и свойства элементов.**
10. **Изменение стилей CSS.**
11. **Использование классов в jQuery.**
12. **Содержимое элементов.**
13. **Создание и добавление дочерних элементов.**
14. **Обертывание элементов.**
15. **Добавление сестринских элементов.**
16. **Замена элементов.**
17. **Удаление элементов.**

Содержание данной темы включает материалы, доступные по адресу https://metanit.com.

1. Начало работы с jQuery.

Современное веб-программирование и создание веб-сайтов уже невозможно представить без использования языка JavaScript. Однако в настоящее время, все чаще используется не просто нативный JavaScript, а JavaScript-фреймворки и библиотеки. Одной из таких библиотек является jQuery. jQuery не просто библиотека, она включает в себя библиотеку jquery.ui, предназначенную для создания визуальных интерфейсов, jqyery.mobile, используемую при разработке мобильных сайтов и др.

Преимущества использования jQuery:

* Упрощение работы с кодом. jQuery предлагает простой элегантный синтаксис для манипулирования элементами на веб-странице.
* Расширяемость. Весь код jQuery открыт для просмотра и изменения, и его можно модифицировать. А также можно создавать плагины jQuery.
* Кроссбраузерность. jQuery имеет поддержку большинства известных браузеров.

**Подключение библиотеки jQuery**

Чтобы начать работать с данной библиотекой, надо ее скачать, например, с официального сайта разработчика <https://jquery.com/download/>. На странице загрузок можно найти несколько версий jQuery. На момент составления данного пособия последней доступной версией является 3.4.1. Хотя версии немного отличаются друг от друга, но эти отличия, как правило, не столь существенны, и базовый стержень и общие принципы у большинства версий по сути одни и те же.

Библиотека представлена в двух вариантах - Compressed или Monified (минимизированная) и Uncompressed (обычный). Минимизированные версии предоставляют ту же функциональность, что и обычные, но отличаются тем, что не содержат всяких необязательных символов, наподобие пробелов, комментариев и т.д., и поэтому в своем названии имею суффикс min, например, jquery-3.4.1.min.js. Поскольку они производительнее за счет меньшего объема, их рекомендуется использовать в реальном производстве.

Библиотека jquery подключается также, как и другие файлы javascript. Например:

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title>Мир jQuery</title>  <script src="jquery-3.4.1.min.js"></script>  </head>  <body>  </body>  </html> |

Создадим простую веб-страницу с использованием jquery:

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title>Мир jQuery</title>  <script src="jquery-3.4.1.min.js"></script>  </head>  <body>  <h2>Добро пожаловать в мир jQuery</h2>  <button id="btn1">jQuery</button>  <button id="btn2" onclick="alert('Мир JavaScript'); ">JavaScript</button>    <script type="text/javascript">  $(function(){      $("#btn1").click(function(){          $(this).css('background-color', 'red');          alert('Мир jQuery');      });  });  </script>  </body>  </html> |

Данная веб-страница, с одной стороны, применяет код jquery, с другой стороны, также демонстрирует отличие от использования стандартного кода javascript. На странице определены две кнопки. У одной кнопки определен обработчик onclick в самой разметке кнопки: onclick="alert('Мир JavaScript'); ". Другая кнопка делает практически тоже самое, но только с jquery. Для этой кнопки определен id (id="btn1"), с помощью которого в функции jquery можно ею управлять. В самом низу страницы определена функция jquery.

Выражение $(function(){}); - это и есть краткое определение функции jquery. Эту функцию принято помещать в конце документа, как в данном случае, перед закрывающем тегом body. Данная функция включает весь код на языке javascript, который будет выполняться при загрузке страницы.

Смысл использованного кода состоит в том, что мы получаем элемент кнопки в выражении $("#btn1") и затем вешаем на него обработчик нажатия. Фактически выражение $("#btn1").click будет аналогично использованию обработчика onclick в теле разметки кнопки. При этом не надо никак изменять разметку самой кнопки, что-то туда дописывать. Все делается в функции jquery: устанавливается цвет кнопки: $(this).css('background-color', 'red'); и выводится сообщение на экран.

В предыдущем примере библиотека подключалась непосредственно с локального диска, однако необязательно загружать библиотеку и размещать на локальном диске рядом с прочими файлами. Вместо этого можно использовать сети CDN (Content Delivery Networks). В данном случае сама библиотека будет находится физически в какой-либо сети CDN, и надо лишь указать на нее ссылку. Например, подключим библиотеку jquery, находящуюся в jQuery CDN:

|  |  |
| --- | --- |
|  | <!DOCTYPE html>  <html>  <head>  <title>Мир jQuery</title>  <script src="<https://code.jquery.com/jquery-3.4.1.min.js>"></script>  </head> |

Также можно использовать и другие сети CDN: Google CDN, Microsoft CDN, Yandex CDN.

1. Функция jQuery.

Ключевым моментом в использовании данной библиотеки является функция jQuery. Эта функция запускается поле загрузки страницы. Поскольку, как правило, эта функция содержит код, манипулирующий элементами веб-страницы, то в этом случае надо, чтобы к моменту срабатывания функции jquery все эти элементы были загружены. Поэтому принято помещать эту функцию в самый низ html-страницы, например, перед закрывающим тегом body.

Если же в предыдущем примере с двумя кнопками код jquery размещался бы в секции head, то в этом случае произошла бы ошибка, поскольку к моменту подключения используемая в скрипте кнопка была бы еще не создана.

Функция jQuery имеет следующий синтаксис:

|  |  |
| --- | --- |
|  | jQuery(document).ready(function(){    // здесь код функции.  }); |

Формально описание функции jQuery: jQuery(объект). В данном случае в качестве объекта используется объект document, представляющий фактически всю структуру DOM веб-страницы. И к нему применяется обработчик ready, который и сигнализирует о том, что DOM-модель веб-страницы загружена. В качестве параметра обработчика используется безымянная функция обратного вызова, которая и срабатывает при загрузке веб-страницы.

Есть также еще один способ объявления, который также равнозначен предыдущему:

|  |  |
| --- | --- |
|  | $(document).ready(function(){    // здесь код функции.  }); |

Знак $ представляет псевдоним jQuery. Можно использовать и сокращенные записи функции jQuery:

|  |  |
| --- | --- |
|  | $(function(){    // здесь код функции.  }); |

Или так:

|  |  |
| --- | --- |
|  | jQuery(function(){    // здесь код функции.  }); |

Все приведенные здесь четыре варианта объявления функции jQuery будут равнозначны.

1. Cелекторы.

Одним из важных функциональностей jQuery является выборка элементов. Чтобы что-то делать с элементами, манипулировать ими, применять к ним методы jQuery, надо сначала их получить. Библиотека предоставляет удобный способ выбора элементов, основанный на селекторах. Достаточно передать в функцию jQuery селектор и можно получить нужный элемент, который соответствует данному селектору. Например, если необходимо получить все эелементы img, то можно использовать следующее выражение: $("img"). В данном случае "img" будет выступать в качестве селектора.

Рассмотрим основные селекторы jQuery, которые используются для выборки элементов.

**Базовые селекторы jQuery**

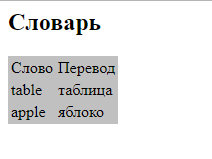
|  |  |  |
| --- | --- | --- |
| **Шаблон селектора** | **Описание** | **Пример** |
| $("\*") | Выборка всех элементов страницы | Например, выражение $("\*").css('background-color', 'red') окрасит все элементы страницы и саму страницу в красный цвет |
| $("Element") | Выборка всех элементов с данным именем тега | $("img") выбирает все элементы img  $("input") выбирает все элементы input |
| $("#id") | Выборка элемента с данным значением атрибута id | $("#btn1") выбирает элемент, у которого значение id равно btn1 (например, <div id="btn1"></div>) |
| $(".class") | Выборка всех элементов с данным значением атрибута class | $(".redStyle") выбирает элементы, у которого значение class равно redStyle (например, <div class="redStyle"></div>) |
| $("selector1,selector2,selectorN") | Выборка всех элементов, которые соответствуют указанным селекторам | Например, у нас есть такой код:  <div class="apple"></div>  <div class="apple"></div>  <div class="orange"></div>  <div class="banana"></div>  то селектор $(".apple, .orange") выберет элементы, выделенные жирным |

**Конкретизация выбора**

Можно использовать сразу несколько селекторов, чтобы более детально настроить выборку элементов. Например, есть такая веб-страница:

|  |  |
| --- | --- |
|  | <!DOCTYPE html>  <html>  <head>  <title>Мир jQuery</title>  </head>  <body>  <h2>Словарь</h2>  <table id="tab">  <tr class="tabhead"><td><p>Слово</p></td><td><p>Перевод</p></td></tr>  <tr><td><p>table</p></td><td><p>таблица</p></td></tr>  <tr><td><p>apple</p></td><td><p>яблоко</p></td></tr>  </table>  <script src="jquery-3.4.1.min.js"></script>  <script type="text/javascript">  jQuery(document).ready(function(){        $("#tab").css('background-color', 'silver');  });  </script>  </body>  </html> |

В данном случае используется селектор #tab, то есть получаем таблицу, которая имеет id=tab. Выражение css('background-color', 'silver') устанавливает цвет элемента - в данном случае серебряный:



Но также можно использовать набор селекторов, чтобы более детально произвести действие. Например, надо, чтобы была закрашена только первая строка, которая имеет атрибут class="tabhead". Тогда можно использовать следующий селектор: $("#tab .tabhead"). В этом случае селектор выберет элемент с class="tabhead", который находится в элементе с id="tab".

При этом можно для сужения выборки использовать множество различных селекторов, например: $("div#menu a.redStyle"). К примеру данный селектор может найти такой элемент:

|  |  |
| --- | --- |
|  | <div id="menu">      <a href="m.tml" class="redStyle">Ссылка на сайт</a>  </div> |

В данном случае применен иерархический селектор, который подразумевает, что элемент по второму селектору должен находиться в элементе по первому селектору. Но кроме того, можно также применить еще ряд иерархических селекторов:

|  |  |  |
| --- | --- | --- |
| **Шаблон селектора** | **Описание** | **Пример** |
| родительский\_селектор > дочерний\_селектор | Выборка всех элементов, которые соответствуют второму селектору и являются дочерними для элементов первого селектора | Например, чтобы получить элемент ссылки в предыдущем примере можно использовать селектор $("#menu > a") |
| селектор1 + селектор2 | Выборка элементов с селектором 'селектор2', которые располагаются непосредственно за элементами 'селектор1'. Причем оба элемента должны находиться на одном уровне | Например, у нас есть следующий блок ссылок:  <div id="menu">  <a href="1.html" class="open">1.html</a>  <a href="2.html" class="closed">2.html</a>  <a href="3.html">3.html</a>  <a href="4.html" class="open">4.html</a>  <a href="5.html" class="closed">5.html</a>  <a href="6.html" class="open">6.html</a>  </div>  Тогда селектор $(".closed + .open") выберет те ссылки, которые выделены жирным, так как эти элементы с атрибутом class="open" идут непосредственно за элементами с атрибутом class="closed". |
| селектор1 ~ селектор2 | Выборка элементов с селектором 'селектор2', которые располагаются на одном уровне с элементами с селектором 'селектор1'. | Например, у нас есть следующий блок ссылок:  <div id="menu">  <a href="1.html" class="open">1.html</a>  <a href="2.html" class="closed">2.html</a>  <a href="3.html">3.html</a>  <a href="4.html" class="open">4.html</a>  <a href="5.html" class="closed">5.html</a>  <a href="6.html" class="open">6.html</a>  </div>  Тогда селектор $(".closed ~ .open") выберет те ссылки, которые выделены жирным, так как эти элементы с атрибутом class="open" находятся на одном уровне с элементами с атрибутом class="closed". |

**Селекторы атрибутов**

Чтобы сузить поиск кроме базовых и иерархических селекторов, рассмотренных выше, можно применять селекторы атрибутов. В данном случае jQuery будет возвращать элементы в зависимости от значений и определения указанных атрибутов.

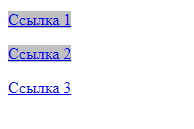
|  |  |  |
| --- | --- | --- |
| **Шаблон селектора** | **Описание** | **Пример** |
| [attr] | Выборка всех элементов с атрибутом attr | Например, выражение $("a[class]") выберет все элементы ссылок, которые имеют атрибут class |
| [attr='value'] | Выборка всех элементов, у которых значение атрибута attr равно value | Например, выражение $("a[class='redStyle']") выберет все элементы ссылок, которые имеют атрибут class="redStyle" |
| [attr^='value'] | Выборка всех элементов, у которых значение атрибута attr начинается со строки value | Например, выражение $("a[class^='red']") выбрать элементы ссылок, которые имеют атрибут class="redStyle", так как слово 'redStyle' начинается на 'red'. |
| [attr$='value'] | Выборка всех элементов, у которых значение атрибута attr оканчивается на строку value | Например, выражение $("a[class$='Style']") выбрать элементы ссылок, которые имеют атрибут class="redStyle", так как слово 'redStyle' оканчивается на 'Style'. |
| [attr~='value'] | Выборка всех элементов, у которых атрибут attr имеет список значений, разделенных пробелом, и одно из этих значений равно value | Например, выражение $("a[class~='redStyle']") выбрать элементы ссылок, которые могут иметь следующее значение атрибута class="redStyle closed". |
| [attr\*='value'] | Выборка всех элементов, у которых значение атрибута attr имеет имеет подстроку value | Например, выражение $("a[class\*='Style']") выбрать элементы ссылок, которые могут иметь следующее значение атрибута class="redStyle1". |
| [attr|='value'] | Выборка всех элементов, у которых значение атрибута attr представляет либо значение value, либо список значений, разделенных дефисами, и первое значение в этом списке равно value | Например, выражение $("a[class|='red']") выбрать элементы ссылок, которые могут иметь следующее значение атрибута class="red-freeStyle-closed". |

**Контекст селекторов**

Выше использовалась такая запись для выборки элемента: $(селектор). Данное выражение просматривало все дерево DOM на соответствие элемента селектору. Однако можно ограничить поиск, применив контекст поиска. Контекст представляет собой селектор, в рамках которого надо производить выборку элементов. Например, есть такая веб-страница:

|  |  |
| --- | --- |
|  | <!DOCTYPE html>  <html>  <head>  <title>Мир jQuery</title>  </head>  <body>  <div id="menu">      <p><a href="1.html" class="open">Ссылка 1</a></p>      <p><a href="2.html" class="open">Ссылка 2</a></p>  </div>  <p><a href="3.html" class="open">Ссылка 3</a></p>    <script src="jquery-3.4.1.min.js"></script>  <script type="text/javascript">  jQuery(document).ready(function(){        $(".open", "div#menu").css('background-color', 'silver');  });  </script>  </body>  </html> |

На странице есть три ссылки, у каждой определен один и тот же класс, но две из них находятся в элементе div и именно их надо получить. Тогда стоит использовать выражение $(".open", "div#menu") – здесь второй параметр-селектор будет являться контекстом выборки. А результатом в данном случае будет следующая страница:

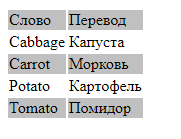


Важно не путать это выражение с выборкой по множеству селекторов, например, $(".open, div#menu ") – это разные выражения, которые будут давать разный результат.

1. Фильтры jQuery.

В дополнение к селекторам применяются фильтры. Фильтры позволяют отфильтровать найденные элементы по определенному принципу. Применять фильтры также просто, как и селекторы. Например, можно получить элементы только с четными индексами:

|  |  |
| --- | --- |
|  | <!DOCTYPE html>  <html>  <head>  <title>Мир jQuery</title>  </head>  <body>  <table>  <tr><td>Слово</td><td>Перевод</td></tr>  <tr><td>Cabbage</td><td>Капуста</td></tr>  <tr><td>Carrot</td><td>Морковь</td></tr>  <tr><td>Potato</td><td>Картофель</td></tr>  <tr><td>Tomato</td><td>Помидор</td></tr>  </table>    <script src="jquery-3.4.1.min.js"></script>  <script type="text/javascript">  jQuery(document).ready(function(){        $("tr:even").css('background-color', 'silver');  });  </script>  </body>  </html> |

В рассматриваемом примере есть таблица, и надо выделить цветом все четные строки. В этом случае стоит применить фильтр :even. Фильтр ставится после селектора: $("tr:even"). В результате все четные строки будут окрашены серым цветом.

Поскольку нумерация объектов начинается с нуля, то поэтому первая строка будет нулевой и, таким образом, попадает в число четных.

Можно выделить следующий набор базовых фильтров:

|  |  |
| --- | --- |
| **Фильтр** | **Описание** |
| :eq(n) | Выбирает n-й элемент выборки (нумерация начинается с нуля) |
| :even | Выбирает элементы с четными номерами |
| :odd | Выбирает элементы с нечетными номерами |
| :first | Выбирает первый элемент выборки |
| :last | Выбирает последний элемент выборки |
| :gt(n) | Выбирает все элементы с номером, большим n |
| :lt(n) | Выбирает все элементы с номером, меньшим n |
| :header | Выбирает все заголовки (h1, h2, h3) |
| :not(селектор) | Выбирает все элементы, которые не соответствуют селектору, указанному в скобках. Например, выражение $("tr:not(.even)") выберет все строки, у которых атрибут class не равен even. |

Специальный тип фильтров – фильтры контента обеспечивают доступ к элементам, имеющим определенное содержимое:

|  |  |
| --- | --- |
| **Фильтр** | **Описание** |
| :contains('text') | Получает все элементы, которые содержат текст text. |
| :has('селектор') | Получает все элементы, которые содержат хотя бы один дочерний элемент, соответствующий селектору |
| :empty | Получает все элементы, которые не имеют дочерних элементов |
| :first-child | Получает все элементы, которые являются первыми дочерними элементами в своих родителях |
| :last-child | Получает все элементы, которые являются последними дочерними элементами в своих родителях |
| :nth-child(n) | Получает все элементы, которые являются n-ными элементами в своих родителях (нумерация идет с единицы) |
| :nth-child(even) | Получает все элементы, которые являются четными элементами в своих родителях (нумерация идет с единицы). Так, в предыдущем примере с таблицей выражение $("tr:nth-child(even)") будет получать все те же элементы, что и выражение $("tr:odd") |
| :nth-child(odd) | Получает все элементы, которые являются нечетными элементами в своих родителях (нумерация идет с единицы) |
| :only-child | Получает все элементы, которые являются единственными дочерними элементами в своих родителях |
| :parent | Получает все элементы, которые имеют, как минимум, один дочерний элемент |

Например, если надо получить все элементы, содержащие текст Капуста, можно применить следующее выражение: $(":contains('Капуста')"). Либо можно с помощью селекторов конкретизировать поиск, например, вести выборку только в строках: $("tr:contains('Капуста')")

И третья группа – фильтры, которые позволяют получить определенные элементы html-форм или используют их атрибуты:

|  |  |
| --- | --- |
| **Фильтр** | **Описание** |
| :button | Получает все элементы button и элементы input с типом button |
| :checkbox | Получает все элементы checkbox |
| :checked | Получает все отмеченные элементы checkbox и radio |
| :disabled | Получает все элементы, которые находятся в неактивном состоянии |
| :enabled | Получает все элементы, которые находятся в активном состоянии |
| :file | Получает все элементы file (input type='file') |
| :input | Получает все элементы input |
| :hidden | Получает все скрытые элементы |
| :password | Получает все элементы password |
| :radio | Получает все элементы radio |
| :reset | Получает все элементы reset |
| :selected | Получает все отмеченные элементы option |
| :submit | Получает все элементы input с типом submit |
| :text | Получает все элементы input с типом text |
| :visible | Получает только видимые элементы |

Можно комбинировать в одном выражении несколько селекторов и фильтров: $('#results:odd:has('img')'). В данном случае осуществляется выбор всех нечетных элементов с id="results", которые содержат элементы img, то есть изображения.

1. Работа с выборкой.

Применение селекторов и фильтров возвращает набор элементов, к которому можно применить различные свойства и методы. Также поскольку набор элементов представляет собой массив, то можно обращаться с ним как с массивом.

Свойство length возвращает количество выбранных элементов: let num = $("tr:nth-child(odd)").length;

Поскольку возвращаемый в процессе выборки набор данных фактически является массивом, то можно получать элементы выборки по индексу: let firstElement = $("tr:nth-child(odd)")[0];. Например:

|  |  |
| --- | --- |
|  | $(function(){      let array = $("tr:even");      for(let i=0;i<array.length;i++){          console.log(i.toString()+". " +array[i].innerHTML);      }  }); |

Альтернативой является использование метода get(index), который в качестве параметра принимает индекс элемента: let elem = $("tr:even").get(0);.

**Функция eq**

Еще одним способом обратиться к элементу является функция eq(index). Это функция в качестве параметра получает индекс элемента и возвращает новый набор, состоящий из одного элемента по данному индексу. Например:

|  |  |
| --- | --- |
|  | $(function(){      let array = $("tr:even");      array.eq(1).css('background-color', 'silver');      array.eq(-1).css('background-color', 'silver');  }); |

Преимуществом такого подхода является то, что можно для результата функции eq использовать методы jQuery (например, css для установки стиля, как в данном случае). Кроме того, можно передать в функцию отрицательное значение. В этом случае отсчет будет производиться с конца.

**Получение первого и последнего элемента**

Поскольку разработчикам довольно часто приходилось в своих задачах получать последний и особенно первый элемент, то разработчики jQuery добавили специальные методы для их выборки: last и first соответственно.

|  |  |
| --- | --- |
|  | $(function(){      let array = $("tr:even");      let firstEl = array.first();      console.log("Первый элемент: " + firstEl.html());      let lastEl = array.last();      console.log("Последний элемент: " + lastEl.html());  }); |

**Перебор элементов выборки**

Кроме перебора в виде обычного массива в цикле, как в вышеприведенном варианте, можно использовать специальный метод each:

|  |  |
| --- | --- |
|  | $(function(){      $("tr:even").each(function(index, elem){          console.log(index + ". " + elem.innerHTML);      });  }); |

В метод each в качестве параметра передается безымянная функция, которая принимает два параметра: index - индекс элемента в наборе и elem - сам элемент.

**Индекс элемента**

Для определения индекса элемента в jQuery предназначен метод index(элемент). В качестве параметра передается элемент, индекс которого надо определить:

|  |  |
| --- | --- |
|  | $(function(){      let array = $("tr:even");      let firstEl = array.first();      let index = array.index(firstEl);      console.log(index); //выведет 0  }); |

Также метод index может принимать в качестве параметра результат выборки функции jQuery:

|  |  |
| --- | --- |
|  | $(function(){      let array = $("tr");      let index = array.index($("tr.tabhead"));      console.log(index);  }); |

**Метод add и добавление новых элементов**

С помощью метода add можно создать новый набор. Данный метод имеет следующие варианты синтаксиса:

* add(селектор): добавление элементов, соответствующих селектору, в набор;
* add(селектор, контекст): добавление элементов, но в данном случае поиск элементов ведется в рамках контекста;
* add(элемент): добавление в набор элемента;
* add(массив элементов): добавление массива элементов;
* add(html): добавление в набор элемента, представленного разметкой html (при этом добавление идет только в набор, а не на страницу);
* add(jQuery): добавление объекта, который является результатом функции jQuery.

Например, создадим начальную выборку из четных строк таблицы и добавим в нее первую строку из нечетных строк таблицы:

|  |  |
| --- | --- |
|  | $(function(){      let array = $("tr:even").add("tr:odd:first");      //данное выражение эквивалентно следующему      // let array = $("tr:even, tr:odd:first");  }); |

В то же время вместо метода add можно применить набор селекторов, который бы дал тот же результат.

1. Фильтрация выборки.

Ранее рассматривались фильтры селекторов, которые помогают произвести более качественную выборку. В jQuery также присутствуют специальные методы, предназначенные для фильтрации, которые фактически имеют тот же самый эффект, что и фильтры селекторов.

**Метод filter**

Метод filter(условие) принимает в качестве параметра условие фильтрации. И если элементы выборки не соответствуют этому условию, то они исключаются из выборки. В зависимости от того, что представляет собой условие, данный метод имеет следующие варианты использования:

* filter(селектор): если объекты выборки не удовлетворяют селектору, то они исключаются;
* filter(элемент): если объекты выборки не представляют данный элемент html, то они исключаются;
* filter(jQuery): если объекты выборки не представляют данный объект jQuery, то они исключаются;
* filter(функция): для каждого элемента выборки вызывается функция, и если эта функция возвращает false, то элемент исключается из выборки

Например, выберем все элементы с атрибутом class="header", которые также являются элементами строк таблицы:

|  |  |
| --- | --- |
|  | $(function(){      //Фильтрация по селектору      let array = $(".header").filter("tr");      array.css('background-color', 'silver');        // Фильтрация по элементу html      let elem = document.getElementsByTagName("tr")[0];      let array1 = $(".header").filter(elem);      array1.css('background-color', 'silver');  }); |

Данный вариант является аналогом выражения $("tr.header"), поэтому того же результата можно добиться, скомбинировав селекторы.

Пример с использованием функции:

|  |  |
| --- | --- |
|  | $(function(){      let array = $("tr").filter(function(){          if($(this).hasClass("header")) { return true;}          else {return false;}      });      array.css('background-color', 'silver');  }); |

Функция вызывается для каждого элемента, а текущий элемент получаем с помощью выражения $(this). Далее с помощью метода hasClass("header") определяем, содержит ли текущий элемент класс header. И если содержит, то возвращаем true.

**Метод not**

Противоположным по действию по сравнению с методом filter является метод not. Он исключает из выборки те элементы, которые соответствуют условию. Данный метод может иметь следующие варианты использования:

* not(селектор): если объекты выборки удовлетворяют селектору, то они исключаются;
* not(элемент): если объекты выборки представляют данный элемент html, то они исключаются;
* not(jQuery): если объекты выборки представляют данный объект jQuery, то они исключаются;
* not(функция): для каждого элемента выборки вызывается функция, и если эта функция возвращает true, то элемент исключается из выборки.

Например, можно в предыдущем примере заменить filter на метод not и тогда получим противоположный результат:

|  |  |
| --- | --- |
|  | $(function(){      let array = $("tr").not(function(){          if($(this).hasClass("header")) { return true;}          else {return false;}      });      array.css('background-color', 'silver');  }); |

**Метод slice**

Метод slice(begin, end) исключает из выборки те элементы, которые не попадают в диапазон, задаваемый параметрами begin и end. Подразумевается, что индексация начинается с 0. Негативные индексы означают, что выборка ведется с конца набора. Необязательный параметр end задает позицию с которой выборка элементов уже прекращена, если этот параметр отсутствует, то выборка ведется до конца набора:

|  |  |
| --- | --- |
|  | $(function(){      let array = $("tr").slice(1,4);      array.css('background-color', 'silver');  }); |

В данном случае итоговая выборка будет содержать 3 элемента tr. Второй вариант функции slice предполагает задание одного параметра – начального индекса, с которого ведется выборка:

|  |  |
| --- | --- |
|  | $(function(){      let array = $("tr").slice(3);      array.css('background-color', 'silver');  }); |

В этом случае в выборку будут включены все объекты, начиная с 4-го.

**Метод has**

Метод has(вложенный элемент) проверяет объект на наличие вложенного элемента. И если объект содержит вложенный элемент, то этот объект остается в выборке. Он может иметь следующие варианты вызова:

* has(селектор): если объекты содержат элемент, соответствующий селектору, то они остаются в выборке;
* has(элемент): если объекты содержат данный элемент html, то они остаются в выборке.

Например, на странице есть список с вложенными списками. И надо выделить тот элемент списка, который имеет вложенный список:

|  |  |
| --- | --- |
|  | <ul>      <li>Пункт 1</li>      <li>Пункт 2          <ul>              <li>Подпункт 2.1</li>              <li>Подпункт 2.2</li>          </ul>      </li>      <li>Пункт 3</li>  </ul> |

Тогда можно использовать следующее выражение: $('li').has('ul').css('background-color', 'silver');

**Преобразование выборки и метод map**

Метод map(callback) предназначен для преобразования элементов выборки с помощью функции, которая передается в параметре callback. А данная функция отбирает элементы из имеющегося для нового набора и на выходе возвращает этот новый набор. Воспользуемся примером с вложенными списками:

|  |  |
| --- | --- |
|  | <ul>      <li>Пункт 1</li>      <li>Пункт 2          <ul>              <li>Подпункт 2.1</li>              <li>Подпункт 2.2</li>          </ul>      </li>      <li>Пункт 3</li>  </ul> |

Теперь отберем только те элементы, которые имеют вложенные списки:

|  |  |
| --- | --- |
|  | $(function(){        let array = $('li').map(function(index,elem){          return $(elem).children()[0];      });        array.each(function(index, elem){          console.log(elem.innerHTML);      });      array.css('background-color', 'silver');  }); |

Функция, передаваемая в метод map, пробегает по всем элементам выборки и принимает два параметра: index - индекс перебираемого элемента в выборке и elem - сам перебираемый элемент. Получив текущий элемент при переборе, можно провести с ним некоторые манипуляции и определить, должен он или какие-то другие элементы попасть в новую выборку.

Так, в данном случае передаем перебираемый элемент в функцию jQuery, чтобы стали доступными методы jQuery, и получаем первый вложенный список: $(elem).children()[0]; (метод children помогает получить вложенные или дочерние элементы).

Далее можно управлять результирующим набором так же, как и любой другой выборкой, например, перебирать, устанавливать свойства и т.д.

**Метод is**

Метод is (условие) определяет, имеются ли в данной выборке объекты, соответствующие условию. В зависимости от условия данный метод имеет следующие варианты использования:

* is(селектор): проверяет, соответствует ли хотя бы один объект выборки селектору;
* is(элемент): проверяет, присутствует ли в выборке данный элемент html;
* is(jQuery): проверяет, присутствует ли в выборке данный объект jQuery;
* is(функция): поиск в выборке объектов, соответствующих данному элементу html.

На выходе метод is возвращает булевое значение: true, если хотя бы один элемент выборки соответствует условию, и false, если такого соответствия не найдено. Например, есть такая разметка:

|  |  |
| --- | --- |
|  | <ul class="menu">      <li>Пункт 1</li>      <li>Пункт 2          <ul class="submenu">              <li>Подпункт 2.1</li>              <li>Подпункт 2.2</li>          </ul>      </li>      <li>Пункт 3</li>  </ul> |

Используем метод is:

|  |  |
| --- | --- |
|  | let lists = $('ul');  if(lists.is('.submenu')){      lists.css('background-color', 'silver');  } |

Поскольку один из элементов списка содержит вложенный подсписок с классом submenu, то метод is взвратит true, и, следовательно, весь список будет закрашен.

1. Поиск в выборке.

Для поиска в выборке предназначен метод find. Он имеет следующие варианты использования:

* find(селектор): поиск в выборке объектов, соответствующих селектору;
* find(элемент): поиск в выборке объектов, соответствующих данному элементу html;
* find(jQuery): поиск в выборке объектов, соответствующих объекту jQuery.

Метод find на выходе возвращает новую выборку. Например, есть следующая разметка меню в виде списков:

|  |  |
| --- | --- |
|  | <ul class="menu">      <li>Пункт 1</li>      <li>Пункт 2          <ul class="submenu">              <li>Подпункт 2.1</li>              <li>Подпункт 2.2</li>          </ul>      </li>      <li>Пункт 3          <ul class="submenu">              <li>Подпункт 3.1</li>              <li>Подпункт 3.2</li>          </ul>      </li>  </ul> |

Теперь последовательно осуществим поиск п селектору, элементу и объекту jQuery:

|  |  |
| --- | --- |
|  | $(function(){      // поиск по селектору      let array0 = $('ul').find('.submenu');      array0.css('background-color', 'silver');        // поиск по элементу      // получаем первый элемент выборки      let elem = $('ul.submenu')[0];      let array1= $('ul').find(elem);      array1.css('background-color', 'gray');        // Поиск объекта jQuery      let jQueryObject = $('ul.submenu');      let array2= $('ul').find(jQueryObject);      array2.css('color', 'blue');  }); |

1. Навигация по странице.

Кроме выборки и фильтрации jQuery предлагает ряд методов для обхода по объектам набора и выбора из них, тех которые нужны.

**Получение дочерних элеметов и метод children**

Для получения дочерних элементов объекта можно использовать метод children([селектор]):

|  |  |
| --- | --- |
|  | <ul class="menu">      <li>Пункт 1</li>      <li>Пункт 2          <ul class="submenu">              <li>Подпункт 2.1</li>              <li>Подпункт 2.2</li>          </ul>      </li>      <li>Пункт 3          <ul>              <li>Подпункт 3.1</li>              <li>Подпункт 3.2</li>          </ul>      </li>  </ul> |

Теперь получим дочерние узлы, как без селектора, так и с применением селектора:

|  |  |
| --- | --- |
|  | // дочерние узлы без селектора  let lists = $('li').children('');  lists.each(function(index, elem){      console.log(elem.innerHTML);  });    //дочерние узлы по селектору  let lists2 = $('li').children('.submenu');  lists2.each(function(index, elem){      console.log(elem.innerHTML);  }); |

В первом случае метод children не использует селектор, поэтому он выбирает все дочерние узла элементов li. То есть результатом будут два элемента ul:

|  |  |
| --- | --- |
|  | <ul class="submenu">      <li>Подпункт 2.1</li>      <li>Подпункт 2.2</li>  </ul>  <ul>      <li>Подпункт 3.1</li>      <li>Подпункт 3.2</li>  </ul> |

Во-втором случае в метод children передается селектор – имя класса дочернего списка, поэтому в этом случае в выборке окажется только первый список, так как только он имеет класс submenu.

**Метод closest**

Метод closest(условие) возвращает ближайшего родителя по отношению к данному элементу. Имеет следующие варианты использования:

* closest(селектор[, контекст]): возвращает ближайшего родителя, соответствующего селектору. В качестве необязательного параметра может принимать контекст, в рамках которого будет проводиться поиск;
* closest(элемент): возвращает ближайшего родителя, соответствующего элементу html;
* closest(jQuery): возвращает ближайшего родителя, соответствующего объекту jQuery.

Итак, найдем ближайших родителей для узлов li:

|  |  |
| --- | --- |
|  | <ul class="menu">      <li>Пункт 1</li>      <li>Пункт 2          <ul class="submenu">              <li>Подпункт 2.1</li>              <li>Подпункт 2.2</li>          </ul>      </li>      <li>Пункт 3          <ul>              <li>Подпункт 3.1</li>              <li>Подпункт 3.2</li>          </ul>      </li>  </ul>  <script type="text/javascript">  $(function(){      let list = $('li').closest('ul.submenu');      list.each(function(index, elem){          console.log(elem.innerHTML);      });  });  </script> |

В итоге метод closest вернет первый вложенный список, поскольку он имеет класс submenu.

**Метод next и последующие элементы**

Для получения следующего по порядку элемента, находящегося на одном уровне с текущим, используется метод next(селектор). Этот метод может принимать необязательный селектор и возвращает элемент, находящийся после текущего.

В данном случае важно понимать, что метод next работает только с элементами одного уровня, или сиблингами (сестрами). Сиблинги (сестры) – это элементы, которые имеют одного общего родителя и находятся на одном уровне. Например:

|  |  |
| --- | --- |
|  | <h2 class="header">Заголовок</h2>  <ul class="menu">      <li class="punkt1">Пункт 1</li>      <li class="punkt2">Пункт 2          <ul class="submenu">              <li>Подпункт 2.1</li>              <li>Подпункт 2.2</li>          </ul>      </li>      <li class="punkt3">Пункт 3</li>  </ul> |

В данном случае сиблингами будут являться заголовок h2 и весь следующий список с классом menu. Если посмотреть вглубь списка, то здесь сиблингами будут элементы списка с классами punkt1, punkt2, punkt3. При этом, поскольку эти элементы находятся с заголовкм h2 на разных уровнях, то по отношению к заголовку они не будут сиблингами.

Применим метод next, получив следующий элемент после первого элемента списка:

|  |  |
| --- | --- |
|  | let lists = $('li.punkt1').next();  lists.each(function(index, elem){      console.log(elem.innerHTML);  }); |

В данном случае метод next вернет элемент <li class="punkt2">, поскольку он идет следующим за элементом li с классом punkt1, который получен в выражении $('li.punkt1').

Можно также получить все следующие элементы одного уровня по отношению к текущему, использовав метод nextAll([селектор]). Этот метод также может принимать в качестве необязательного параметра селектор, сужающий выборку:

|  |  |
| --- | --- |
|  | // без селектора  let lists = $('li.punkt1').nextAll();  lists.each(function(index, elem){      console.log(elem.innerHTML);  });    // c селектором  let lists1 = $('li.punkt1').nextAll('.punkt3');  lists1.each(function(index, elem){      console.log(elem.innerHTML);  }); |

В первом случае в выборке окажутся все элементы одного уровня, которые находятся после элемента li с классом punkt1. Во втором же случае в выборке будут только те следующие элементы, класс которых – punkt3. Кроме того, получить следующие элементы одного уровня с текущим можно с помощью метода nextUntil([селектор]). Если использовать данный метод без селектора, то его действие аналогично методу nextAll.

Если применить селектор, то тогда поиск следующих элементов будет производиться до того элемента, который соответствует селектору. Например:

|  |  |
| --- | --- |
|  | let lists = $('li.punkt1').nextUntil('li.punkt3');  lists.each(function(index, elem){      console.log(elem.innerHTML);  }); |

В данном случае осуществляется поиск всех элементов списка до того элемента, класс которого равен punkt3.

**Метод prev и предыдущие элементы**

На группу методов next/nextAll/nextUntill похожа группа методов prev/prevAll/prevUntil. Метод prev имеет похожее действие на элементы одного уровня, только получает предыдущие элементы по отношению к текущему. Возьмем для примера использованную разметку списком и применим метод prev:

|  |  |
| --- | --- |
|  | let lists = $('li.punkt3').prev();  lists.each(function(index, elem){      console.log(elem.innerHTML);  }); |

Метод prev также может принимать в качестве параметра селектор, сужающий диапазон элементов выборки. Подобно методу nextAll можно использовать метод prevAll([селектор]):

|  |  |
| --- | --- |
|  | // без селектора  let lists = $('li.punkt3').prevAll();  lists.each(function(index, elem){      console.log(elem.innerHTML);  });    // c селектором  let lists1 = $('li.punkt3').prevAll('.punkt1');  lists1.each(function(index, elem){      console.log(elem.innerHTML);  }); |

Аналогично использованию метода nextUntil применение метода prevUntil([селектор]):

|  |  |
| --- | --- |
|  | let lists = $('li.punkt3').prevUntil('li.punkt1');  lists.each(function(index, elem){      console.log(elem.innerHTML);  }); |

**Получение всех элементов одного уровня (сиблингов)**

Метод siblings([селектор]) представляет собой комбинацию методов nextAll и prevAll и позволяет получить всех сиблингов текущего элемента.

|  |  |
| --- | --- |
|  | let lists = $('.punkt2').siblings(); |

В этом случае переменная lists будет содержать набор из предыдущего элемента punkt1 и следующего элемента punkt3.

**Методы parent/parents и получение родителей**

Метод parent([селектор]) возвращает непосредственных родителей для данного элемента. Например, возьмем предыдущую разметку со списками:

|  |  |
| --- | --- |
|  | <body>  <ul class="menu">      <li class="punkt1">Пункт 1</li>      <li class="punkt2">Пункт 2          <ul class="submenu">              <li>Подпункт 2.1</li>              <li>Подпункт 2.2</li>          </ul>      </li>      <li class="punkt3">Пункт 3</li>  </ul>  </body> |

В даном случае для списка menu непосредственным родителем будет элемент body. А для элементов li непосредственным родителем будет элемент ul:

|  |  |
| --- | --- |
|  | let lists = $('.menu').parent(); |

Используя селектор, можно конкретизировать родителя на соответствие данному селектору.

Похожим действием обладает метод parents([селектор]). Только он возвращает не только непосредственных родителей, но и вообще всех родителей вплоть до элемента самого верхнего уровня - то есть элемента html. Например, получим всех родителей для первого элемента списка:

|  |  |
| --- | --- |
|  | $('.punkt1').parents(); |

Это выражение вернет следующую цепочку родителей: ul->body->html.

Можно ограничить возвращаемый набор родителей, использовав селектор или применив еще один метод – метод parentsUntil([селектор]). Этот метод ограничивает восхождение по дереву родителей элементом, соответствующим селетору. Например, чтобы родители не выходили за пределы элемента body:

|  |  |
| --- | --- |
|  | $('.punkt1').parentsUntil('body'); |

1. Атрибуты и свойства элементов.

Библиотека jQuery предлагает инструментарий для манипуляции свойствами и атрибутами элементов.

Атрибуты – это элементы разметки элементов, такие как id, style, class и ряд других. Свойства же представляют элементы объектов javascript.

Несмотря на такое различие, между свойствами и атрибутами имеет сопоставление. Так, атрибут id будет соответствовать свойству id. Например, у нас есть ссылка:

|  |  |
| --- | --- |
|  | <a href="1.html" id="link1">Ссылка</a>  <script type="text/javascript">  $(function(){      let ahref=document.getElementsByTagName('a')[0]      console.log(ahref.id); // выведет link1  });  </script> |

То есть на консоль выведет значение атрибута id. В то же время не для всех атрибутов есть одноименные классы. Так, атрибуту class соответствует свойство className.

**Изменение свойств**

Для работы со свойствами в jQuery имеется метод prop(). Чтобы получить значения свойства, надо передать в этот метод имя свойства. Например, можно получить все адреса ссылок на странице:

|  |  |
| --- | --- |
|  | $('a').each(function(index,elem){      console.log($(elem).prop('href'));  }); |

Чтобы изменить значение свойства, надо просто передать в качестве второго параметра новое значение:

|  |  |
| --- | --- |
|  | $('a').first().prop('href','33.html'); |

После этого первая ссылка на странице будет иметь в качестве адреса страницу 33.html.

**Удаление свойств**

Для удаления свойств можно использовать метод removeProp('имя\_свойства'):

|  |  |
| --- | --- |
|  | $('a').first().removeProp('href'); |

После этого свойству будет присвоено новое значение: undefined, которое будет указывать, свойство не определено.

**Работа с атрибутами**

Работа с атрибутами во многом похожа на работу со свойствами. Так, чтобы получить значение атрибута элемента, надо использовать метод attr('имя\_атрибута'):

|  |  |
| --- | --- |
|  | $('a').each(function(index,elem){      console.log($(elem).attr('href'));  }); |

Действие данного метода аналогично тому, что приводилось для метода prop: вывод на консоль всех адресов ссылок. А передавая в качестве второго параметра в метод attr некоторое значение, можно установить новое значение атрибута:

|  |  |
| --- | --- |
|  | $('a').first().attr('href','33.html'); |

И также для удаления значений атрибутов можно использовать метод removeAttr('имя\_атрибута'):

|  |  |
| --- | --- |
|  | $('a').first().removeAttr('href'); |

Обратите внимание, если в случае с удалением свойства через метод removeProp соответствующий свойству атрибут оставался, только ему присваивалось значение свойства, то есть undefined, то при удалении атрибута через removeAttr атрибут удаляется из разметки элемента.

**Пользовательские атрибуты html5**

В HTML5 была введена такая функциональность, как пользовательские атрибуты. Их суть в том, что можно применить к элементу дополнительные атрибуты, которые будут хранить некоторое дополнительное значение. Подобные атрибуты начинаются с префикса data-, после которого идет собственно название атрибута и его значение. Например, в следующем примере добавим атрибут data-year, который будет хранить год:

|  |  |
| --- | --- |
|  | <ul data-year="2010">      <li>Java</li>      <li>C/C++</li>      <li>PHP</li>  </ul> |

Для работы с подобными атрибутами используется метод data. Он имеет следующие варианты использования:

* data('имя атрибута'): получает значение атрибута первого элемента набора. В качестве параметра передается имя атрибута без префикса data-.

Так, можно получить значение атрибута из предыдущего примера следующим образом:

|  |  |
| --- | --- |
|  | console.log($('ul').data('year')); |

* data(): возвращает объект javascript, который содержит набор атрибутов и их значений в виде пар ключ-значение.

Например, есть элемент с двумя атрибутами:

|  |  |
| --- | --- |
|  | <ul data-year="2010" data-description="lang"> |

Тогда метод data() вернул бы объект {description:'lang', year:'2010'}. И чтобы получить значение отдельного атрибута, можно написать так:

|  |  |
| --- | --- |
|  | console.log($('ul').data().year); |

* data('атрибут', 'новое значение'): устанавливает в качестве значения атрибута строку, переданную в качестве второго параметра:

|  |  |
| --- | --- |
|  | $('ul').first().data('year', '2012'); |

* В качестве нового значения может выступать и целый объект javascript, а данный атрибут будет содержать в качестве значения весь этот объект:

|  |  |
| --- | --- |
|  | $('ul').first().data('lang', { rate: "tiobe", year: 2012 }); |

Если же надо удалить атрибут, то для этого нужно применить метод removeData('название\_атрибута'):

|  |  |
| --- | --- |
|  | $('ul').removeData('year'); |

1. Изменение стилей CSS.

**Получение стиля**

Для работы со стилями применяется метод css(). Чтобы получить значение нужного свойства, надо передать в этот метод в качестве параметра название свойства:

|  |  |
| --- | --- |
|  | console.log($('body').css('font-size')); |

**Изменение стиля**

Чтобы изменить стиль, во-первых, можно передать в метод css в качестве второго параметра новое значение свойства:

|  |  |
| --- | --- |
|  | $('a').css('font-weight', 'bold'); |

Во-вторых, можно изменить стиль с помощью функции, которая также передается в качестве второго параметра метода css. Например, изменим цвет шрифта ссылок:

|  |  |
| --- | --- |
|  | $('a').css('color', function(index, oldValue){          if(oldValue=='rgb(0, 0, 238)')          {return 'red';}          else          {return 'green';}      }); |

В зависимости от того, какое было старое значение oldValue элемента для данного свойства, функция возвращает новое значение для каждого элемента выборки.

В-третьих, можно передать массив свойств для установки:

|  |  |
| --- | --- |
|  | $('a').css({'color':'red', 'cursor':'pointer', 'font-size': '14px'}); |

Здесь в качестве параметра передаем объект javascript, в котором устанавливаем для нужных свойств новые значения.

Иногда необходимо увеличить или уменьшать свойства относительно текущего значения на определенное количество. В этом случае можно написать так:

|  |  |
| --- | --- |
|  | $('a').css({'font-size': '-=1', 'margin-left': '+=10'}); |

**Установка ширины и высоты**

Несмотря на то, что можно манипулировать свойствами width и height через метод css, можно также использовать одноименные методы width() и height():

|  |  |
| --- | --- |
|  | let div = $('div').first();  let newWidth=div.width()+150;  div.width(newWidth);  let newHeight = div.height()+50;  div.height(newHeight); |

1. Использование классов в jQuery.

**Добавление класса**

Для добавления к элементу класса применяется метод addClass('названия\_классов'), который в качестве параметра принимает название класса. Поскольку для одного элемента можно определить несколько классов, то и в данном случае можно передать в метод addClass строку, содержащую несколько классов, разделенных пробелами:

|  |  |
| --- | --- |
|  | $('ul').first().addClass("redStyle");  // добавление двух классов  $('div').first().addClass("black visible"); |

**Удаление класса**

Для удаления класса используется метод removeClass('названия\_классов'). Данный метод также принимает набор имен классов, разделенных пробелами. И если у элемента имеется такой класс, то он удаляется:

|  |  |
| --- | --- |
|  | $('div').first().removeClass("black visible"); |

**Метод hasClass**

Нередко бывает нужно проверить, имеет ли тот или иной элемент какой-нибудь класс, особенно это бывает необходимо, если классы добавляются динамически. В этом случае может помочь метод hasClass('названия\_классов'). И если содержит класс, то данный метод возвращает true, иначе false.

|  |  |
| --- | --- |
|  | if($('ul').first().hasClass("redStyle")){      console.log('Список содержит класс redStyle');  }  else{      console.log('Список не содержит класс redStyle');  } |

**Переключение классов**

Переключение классов осуществляется с помощью метода toggleClass('название\_класса'). В качестве параметра принимается название класса, который будет присвоен элементу. Например, сделаем переключение класса у кнопки по ее нажатию:

|  |  |
| --- | --- |
|  | <style>  .redStyle {color:red;}  </style>  <!------------------------------------------->  <button class="redStyle">Переключить стиль</button>  <script type="text/javascript">  $(function(){          $('button').click(function(){              $(this).toggleClass("redStyle");          }      });  });  </script> |

Здесь кнопке назначен обработчик события click. Сам обработчик передается в виде функции в качестве параметра. В этой функции переключаем класс redStyle с помощью метода toggleClass("redStyle") – если класс есть, он удаляется, а если его нет, то он добавляется.

1. Содержимое элементов.

Библиотека jQuery представляет удобный инструментарий для работы с содержимым элемента в виде методов html и text.

**Работа с разметкой**

Для получения или установки разметки используется метод html, который имеет следующие варианты использования:

* html(): получает разметку html первого элемента в наборе;
* html('новая\_разметка'): устанавливает в качестве разметки элемента код html, переданный в качестве параметра;
* html(функция): установка разметки с помощью функции.

Получим и изменим разметку элемента:

|  |  |
| --- | --- |
|  | // получение разметки для элемента списка и вывод ее в консоль  let html = $('li.punkt2').html();  console.log(html);    // установка новой разметки элемента списка  $('li.punkt1').html("Пункт 1 <ul class='submenu'><li>Подпункт 1.1</li></ul>") |

Новая разметка полностью затирает старую. Если надо просто добавить кусок разметки к старой, то можно присоединить новую разметку к старой:

|  |  |
| --- | --- |
|  | let oldHtml = $('div.header').html();  $('div.header').html(oldHtml+"<p> Новый параграф</p>") |

Также можно использовать функцию, которая пробегает по всем элементам выборки и в качестве параметров принимает текущий индекс элемента в выборке и его текущую разметку. С помощью функции можно переписать предыдущий пример следующим образом:

|  |  |
| --- | --- |
|  | $('div.header').html(function(index,oldValue){      console.log("Старая разметка: " +oldValue);      return oldValue+"<p> Новый параграф</p>";  }); |

**Работа с текстом элемента**

Для работы с тексом используется метод text(), действие которого во многом похоже на действие метода html, только вместо разметки управляем текстом элемента:

* text(): получает текст первого элемента в наборе;
* text('новый\_текст'): устанавливает в качестве текста элемента строку, переданную в качестве параметра;
* text(функция): установка текста с помощью функции.

|  |  |
| --- | --- |
|  | let oldText = $('div.header').text();  console.log(oldText);    $('div.header').text("Название статьи");    // использование функции  $('div').first().text(function(index,oldValue){      console.log("Старый текст: " +oldValue);      return "Новый текст";  }); |

Однако метод text следует использовать с осторожностью. Если элемент имеет внутреннюю разметку, то она полностью замещается новым текстовым содержимым. Поэтому в случае, если надо изменить текст без изменения разметки, то лучше применять другие методы.

**Работа с элементами форм**

Для получения значений элементов форм используется метод val(). По принципу действия он похож на методы html() и text(), только используется для установки значений компонентов форм.

* val(): получает значение первого элемента в наборе;
* val('новое\_значение'): устанавливает в качестве значения элемента строку, переданную в качестве параметра;
* val(функция): установка значения с помощью функции.

|  |  |
| --- | --- |
|  | <input type="text" value="Введите значение" />  <script type="text/javascript">  $(function(){      // получение значения текстового поля      let oldValue = $('input[type="text"]').val();      console.log(oldValue);        $('input[type="text"]').first().val("Введите логин");        // использование функции для установки значения      $('input[type="text"]').first().val(function(index,oldValue){          console.log("Старое значение: " +oldValue);          return "Введите пароль";      });  });  </script> |

1. Создание и добавление дочерних элементов.

**Создание новых элементов**

Для создания новых элементов разметки html можно использовать функцию jQuery, передав ей в качестве параметра код добавляемой разметки:

|  |  |
| --- | --- |
|  | let newList=$('<ul><li>Item1</li><li>Item2</li></ul>');  console.log(newList.html()); |

Здесь создается новый элемент, при этом он пока еще не добавляется в html-разметку веб-страницы.

**Метод clone**

Альтернативным способом создания новых элементов представляет метод clone. Данный метод просто клонирует разметку уже существующего элемента:

|  |  |
| --- | --- |
|  | let newList=$('ul').first().clone();  console.log(newList.html()); |

**Добавление элементов**

Само создание элементов было бы бесполезным, если бы нельзя было их добавить в структуру веб-страницы. Для добавления jQuery предоставляет пару методов append/prepend.

**Метод append**

Метод append добавляет элемент в конец элемента выборки. Он имеет следующие варианты использования:

* append('разметка html'): вставляет указанную в параметре разметку html в конец элемента выборки;
* append(элемент): вставка элемента в конец элемента выборки;
* append(jQuery): вставка объекта jQuery в конец элемента выборки;
* append(функция): в конец элемента выборки вставляется строка с разметкой html, элемент html или объект jQuery, которые возвращаются функцией.

Например, у нас есть следующий список:

|  |  |
| --- | --- |
|  | <ul id="langs">      <li>Java</li>      <li>C/C++</li>      <li>PHP</li>  </ul> |

Применим метод append:

|  |  |
| --- | --- |
|  | // вставка строки кода  $('#langs').append('<li>C#</li>');    // вставка элемента  let jsItem = document.createElement('li');  jsItem.innerHTML='JavaScript';  $('#langs').append(jsItem);    //вставка объекта jQuery  let vbItem = $('<li>Visual Basic</li>');  $('#langs').append(vbItem); |

В результате итоговый список будет выглядеть следующим образом:

|  |  |
| --- | --- |
|  | <ul id="langs">      <li>Java</li>      <li>C/C++</li>      <li>PHP</li>      <li>C#</li>      <li>JavaScript</li>      <li>Visual Basic</li>  </ul> |

Использование функции:

|  |  |
| --- | --- |
|  | $('ul#langs').append(function(index,html){      return $('<li>JavaScript</li>');  }); |

Функция вызывается для каждого элемента выборки и принимает два аргумента: index – индекс текущего перебираемого элемента выборки и html – разметка текущего элемента выборки. На выходе в данном случае возвращается объект jQuery, хотя также можно вернуть и просто строку с разметкой и элемент html.

**Метод prepend**

Метод prepend похож на метод append, только добавляет новый элемент в начало элемента выборки. Он имеет следующие варианты использования:

* prepend('разметка html'): вставляет указанную в параметре разметку html в начало элемента выборки;
* prepend(элемент): вставка элемента в начало элемента выборки;
* prepend(jQuery): вставка объекта jQuery в начало элемента выборки;
* prepend(функция): в начало элемента выборки вставляется строка с разметкой html, элемент html или объект jQuery, которые возвращаются функцией.

Применим метод prepend:

|  |  |
| --- | --- |
|  | // вставка строки кода  $('#langs').prepend('<li>C#</li>');    // вставка элемента  let jsItem = document.createElement('li');  jsItem.innerHTML='JavaScript';  $('#langs').prepend(jsItem);    //вставка объекта jQuery  let vbItem = $('<li>Visual Basic</li>');  $('#langs').prepend(vbItem); |

Применение функции будет аналогично ее использованию в методе append. В результате итоговый список будет выглядеть следующим образом:

|  |  |
| --- | --- |
|  | <ul id="langs">      <li>Visual Basic</li>      <li>JavaScript</li>      <li>C#</li>      <li>Java</li>      <li>C/C++</li>      <li>PHP</li>  </ul> |

**Методы appendTo и prependTo**

Методы appendTo и prependTo добавляют объекты в качестве дочерних к элементу, который указан в параметре метода. Например, выше добавлялись элементы в список. Частично перепишем пример, использовав appendTo и prependTo:

|  |  |
| --- | --- |
|  | // вставка в объект jQuery  $('<li>C#</li>').prependTo('#langs');    // вставка в элемент html  let list = document.getElementById('langs');  $('<li>JavaScript</li>').appendTo(list); |

Методы appendTo и prependTo в качестве параметров могут принимать либо объект jQuery, либо элемент html. В первом случае сначала создаем объект jQuery из разметки ($('<li>C#</li>')) и затем добавляем его в начало элемента, у которого id=langs. Во втором случае, получаем элемент списка с помощью метода JavaScript и затем в конец этого элемента добавляем разметку html с помощью метода appendTo.

1. Обертывание элементов.

В предыдущем вопросе дочерние элементы вставляли в уже существующий элемент. Но также можно создать новый элемент и обернуть в него уже существующие. Для этого используется метод wrap() и похожие методы wrapAll() и wrapInner(). Например, может понадобиться обернуть некоторые элементы в новый элемент div. Для этого и предназначены вышеназванные методы.

**Метод wrap**

Метод wrap обертывает элементы выборки тем элементом, которые передается в качестве аргумента в данный метод. Он имеет следующие варианты использования:

* wrap('разметка html'): обертывает элементы выборки в элемент, создаваемый из разметки html;
* wrap(элемент): обертывает элементы выборки в элемент html, переданный в качестве параметра;
* wrap('селектор'): обертывает элементы выборки в элемент документа, соответствующий селектору;
* wrap(jQuery): обертывает элементы выборки в объект jQuery;
* wrap(функция): обертывает элементы выборки в объект, возвращаемый функцией.

Пусть есть следующая разметка:

|  |  |
| --- | --- |
|  | <ul class="langs">      <li>Java</li>      <li>C/C++</li>      <li>PHP</li>  </ul> |

Обернем список с помощью метода wrap:

|  |  |
| --- | --- |
|  | $('.langs').wrap('<div class="redStyle"></div>'); |

После этого разметка будет выглядеть следующим образом:

|  |  |
| --- | --- |
|  | <div class="redStyle">  <ul class="langs">      <li>Java</li>      <li>C/C++</li>      <li>PHP</li>  </ul>  </div> |
|  |  |

Подобным образом можно использовать и другие способы использования метода wrap:

|  |  |
| --- | --- |
|  | let header = document.getElementById('header');  $('.langs').wrap(header); |

Обратите внимание, что в данном случае список вставляется не в существующий элемент, а просто существующий элемент клонируется, и в эту копию обертывается список.

**Метод wrapAll**

Для обертки всех элементов выборки как единого целого в один элемент используется метод wrapAll. Он имеет похожие варианты использования:

* wrapAll('разметка html'): обертывает все элементы выборки в один элемент, создаваемый из разметки html;
* wrapAll(элемент): обертывает все элементы выборки в один элемент html, переданный в качестве параметра;
* wrapAll('селектор'): обертывает все элементы выборки в один элемент, соответствующий селектору;
* wrapAll(jQuery): обертывает все элементы выборки в один объект jQuery.

Например, имеется следующая разметка:

|  |  |
| --- | --- |
|  | <div class="lang">Java</div>  <div class="lang">C/C++</div>  <div class="lang">PHP</div> |

Обернем все эти элементы в один элемент div:

|  |  |
| --- | --- |
|  | $('.lang').wrapAll('<div class="langs"></div>'); |

После этого разметка будет выглядеть следующим образом:

|  |  |
| --- | --- |
|  | <div class="langs">      <div class="lang">Java</div>      <div class="lang">C/C++</div>      <div class="lang">PHP</div>  </div> |

**Метод wrapInner**

Для обертки содержимого элементов в новый элемент служит метод wrapInner. Он имеет следующие варианты использования:

* wrapInner('разметка html'): обертывает содержимое элементов выборки в элемент, создаваемый из разметки html;
* wrapInner(элемент): обертывает содержимое элементов выборки в элемент html, переданный в качестве параметра;
* wrapInner('селектор'): обертывает содержимое элементов выборки в элемент документа, соответствующий селектору;
* wrapInner(jQuery): обертывает содержимое элементов выборки в объект jQuery;
* wrapInner(функция): обертывает содержимое элементов выборки в объект, возвращаемый функцией.

Исходная разметка будет выглядеть так:

|  |  |
| --- | --- |
|  | <ul>  <li class="lang">Java</li>  <li class="lang">C/C++</li>  <li class="lang">PHP</li>  </ul> |

Обернем содержимое элементов списка в теги <p>:

|  |  |
| --- | --- |
|  | $('li.lang').wrapInner('<p></p>'); |

Финальная разметка будет выглядеть так:

|  |  |
| --- | --- |
|  | <ul>      <li class="lang"><p>Java</p></li>      <li class="lang"><p>C/C++</p></li>      <li class="lang"><p>PHP</p></li>  </ul> |

1. Добавление сестринских элементов.

Если метод wrap имеет дело с родительскими элементами, а методы append/prepend – с дочерними, то для добавления элементов одного уровня или сестер(сиблингов) используются методы before/after/insertBefore/insertAfter.

**Метод before**

Метод before вставляет новый элемент перед каждым элементом выборки. Он имеет следующие способы использования:

* before('разметка html'): добавляет в разметку перед каждым элементом выборки элемент, создаваемый из разметки html;
* before(элемент): добавляет в разметку перед каждым элементом выборки элемент html, переданный в качестве параметра;
* before(jQuery): добавляет в разметку перед каждым элементом выборки в объект jQuery;
* before(функция): добавляет в разметку перед каждым элементом выборки объект, возвращаемый функцией.

Допустим, есть следующий список:

|  |  |
| --- | --- |
|  | <ul>  <li class="lang">Java</li>  <li class="lang">C/C++</li>  <li class="lang">PHP</li>  </ul> |

Используя метод before, можно добавить новый элемент списка в начало:

|  |  |
| --- | --- |
|  | // добавление с помощью кода разметки  $('li.lang').first().before('<li class="lang">JavaScript</li>');    // добавление нового элемента html  let vbItem = document.createElement('li');  vbItem.innerHTML='Visual Basic';  $('li.lang').first().before(vbItem); |

После добавления новые элементы окажутся в начале списка:

|  |  |
| --- | --- |
|  | <ul>  <li class="lang">Visual Basic</li>  <li class="lang">JavaScript</li>  <li class="lang">Java</li>  <li class="lang">C/C++</li>  <li class="lang">PHP</li>  </ul> |

**Метод after**

Похожим образом работает метод after, за тем исключением, что добавление идет в конец списка.

* after('разметка html'): добавляет после каждого элемента выборки элемент, создаваемый из разметки html;
* after(элемент): добавляет после каждого элемента выборки элемент html, переданный в качестве параметра;
* after(jQuery): добавляет после каждого элемента выборки в объект jQuery;
* after(функция): добавляет после каждого элемента выборки объект, возвращаемый функцией.

Возьмем предыдущий пример со списком и добавим в него пару элементов:

|  |  |
| --- | --- |
|  | // добавление с помощью кода разметки  $('li.lang').last().after('<li class="lang">JavaScript</li>');    // Добавление объекта jQuery через клонирование последнего элемента  $('li.lang').last().after($('li.lang').last().clone().html('Visual Basic')); |

**Метод insertBefore**

Метод insertBefore по своему действию похож на метод before, только в качестве параметра он принимает элемент, перед которым будет происходить добавление. Он имеет следующий синтаксис:

* insertBefore('разметка html'): элемент добавляется перед элементом, создаваемым из разметки html;
* insertBefore(элемент): элемент добавляется перед элементом html, который передается в качестве параметра;
* insertBefore(jQuery): элемент добавляется перед объектом jQuery;
* insertBefore('селектор'): элемент добавляется перед объектом, который соответствует селектору.

Фактически это тот же метод before, только здесь объект jQuery и элемент, относительно которого производится вставка, меняются местами:

|  |  |
| --- | --- |
|  | $('<li class="lang">JavaScript</li>').insertBefore('li.lang:first'); |

Здесь элемент <li class="lang">JavaScript</li> будет добавлен перед элементом, который соответствует селекторуli.lang:first

**Метод insertAfter**

Метод insertAfter похож на метод after, только, как и метод insertBefore, в качестве параметра он принимает элемент, после которого будет происходить добавление. Он имеет следующие варианты использования:

* insertAfter('разметка html'): элемент добавляется после элемента, создаваемого из разметки html;
* insertAfter(элемент): элемент добавляется после элемента html, который передается в качестве параметра;
* insertAfter(jQuery): элемент добавляется после объекта jQuery;
* insertAfter('селектор'): элемент добавляется после элемента, который соответствует селектору.

|  |  |
| --- | --- |
|  | $('<li class="lang">JavaScript</li>').insertAfter('li.lang:last'); |

1. Замена элементов.

**Метод replaceWith**

Метод replaceWith замещает внутренне содержимое элемента новым содержимым. Он имеет следующие варианты использования:

* replaceWith('разметка html'): замена содержимого элемента разметкой html;
* replaceWith(элемент): замена содержимого элемента некоторым элементом html;
* replaceWith(jQuery): замена содержимого элемента объектом jQuery;
* replaceWith(функция): замена содержимого элемента объектом, возвращаемым функцией.

Например, заменим первый элемент списка новым содержимым:

|  |  |
| --- | --- |
|  | $('li.lang').first().replaceWith('<li class="lang">JavaScript</li>'); |

Применим функцию для замены содержимого. Допустим, есть следующая разметка:

|  |  |
| --- | --- |
|  | <div id="list">      <div class="lang">Java</div>      <div class="lang">C/C++</div>      <div class="lang">PHP</div>  </div> |

Если надо преобразовать этот элемент div в список, тогда можно написать так:

|  |  |
| --- | --- |
|  | $('div.lang').replaceWith(function() {          return '<li class="lang">'+$(this).text()+"</li>";  });  $('div#list').replaceWith(function() {          return '<ol>'+$(this).html()+"</ol>";  }); |

Функция проходит по всем элементам выборки. Перебираемый элемент можно получить с помощью ключевого слова this. Затем заменяем элементы внутри div на элементы списка. На выходе получится следующая разметка:

|  |  |
| --- | --- |
|  | <ol>      <li class="lang">Java</li>      <li class="lang">C/C++</li>      <li class="lang">PHP</li>  </ol> |

**Метод replaceAll**

Метод replaceAll(замещаемый\_элемент) замещает все элементы, определенные в параметре, новым элементом. В качестве замещаемого элемента можем передавать селектор, элемент html или объект jQuery. Например, заменим все элементы предыдущего списка другим содержимым:

|  |  |
| --- | --- |
|  | $('<li class="lang">Элемент списка</li>').replaceAll('li.lang'); |

В результате получится следующая разметка:

|  |  |
| --- | --- |
|  | <ol>      <li class="lang">Элемент списка</li>      <li class="lang">Элемент списка</li>      <li class="lang">Элемент списка</li>  </ol> |

1. Удаление элементов.

**Метод remove**

Метод remove([селектор]) удаляет элементы из структуры DOM. Например, есть следующий список:

|  |  |
| --- | --- |
|  | <ul>      <li class="lang">Java</li>      <li class="lang">C/C++</li>      <li class="lang">PHP</li>      <li class="lang">JavaScript</li>  </ul> |

Можно удалить все четные элементы списков:

|  |  |
| --- | --- |
|  | $('li:even').remove(); |

А результатом будет следующий список:

|  |  |
| --- | --- |
|  | <ul>      <li class="lang">C/C++</li>      <li class="lang">JavaScript</li>  </ul> |

Также можно бы применить селектор для фильтрации удаляемых элементов:

|  |  |
| --- | --- |
|  | $('li').remove(':even'); |

**Метод detach**

Метод detach аналогичен методу remove за тем исключением, что при удалении объекта сохраняются все связанные с ним данные. Преимущество такого подхода заключается в том, что можно позднее вставить удаленные элемент в другом месте разметки html:

|  |  |
| --- | --- |
|  | // удаляем из списка первый элемент, сохраняя его в переменной item  let item = $('li:first').detach();  // вставляем его в конец списка  $('ul').append(item); |

**Метод empty**

Метод empty очищает содержимое элементов, делая их пустыми:

|  |  |
| --- | --- |
|  | $('ul').empty(); |

В результате все элементы списка будут удалены, и разметка будет содержать пустой список: <ul></ul>