**Тема 12. Эффекты и анимация в jQuery. jQuery UI**

1. **Базовые эффекты.**
2. **Эффекты скольжения.**
3. **Эффекты прозрачности.**
4. **Создание анимации.**
5. **Введение в jQuery UI.**
6. **Виджет button.**
7. **Виджет progressbar.**
8. **Виджет слайдер jQuery.**
9. **Виджет Accordion.**
10. **Виджет Autocomplete.**

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

1. Базовые эффекты.

К базовым эффектам в jQuery относятся эффекты скрытия и отображения элементов, которые достигаются с помощью методов show(), hide() и toggle(). Например, можно скрывать и отображать элементы по клику по кнопке:

|  |  |
| --- | --- |
|  | <ul>      <li>Java</li>      <li>C/C++</li>      <li>JavaScript</li>  </ul>  <button id="show">Показать</button>  <button id="hide">Скрыть</button>  <script type="text/javascript">  $(function() {      $('button#show').click(function(){          $('ul').show();      });      $('button#hide').click(function(){          $('ul').hide();      });  });  </script> |

Таким образом, при нажатии на кнопку "Показать", будет срабатывать метод $('ul').show(), а при нажатии на кнопку "Скрыть", список будет скрыт с помощью метода $('ul').hide(). Тот же самый эффект можно сделать, использовав в обоих случаях метод toggle, который просто переключает видимость:

|  |  |
| --- | --- |
|  | $(function() {      $('button').click(function(){          $('ul').toggle();      });  }); |

Разберем эти методы подробнее.

**Метод hide**

Метод hide присваивает у элементов свойству стиля display значение none, тем самым делая элемент скрытым. То же самое можно проделать с помощью следующего выражения: css('display', 'none'). Метод hide может принимать следующие формы:

* hide(): метод без параметров;
* hide([duration][, complete]): принимает два необязательных параметра. Параметр duration указывает как долго анимация элемента будет длиться. По умолчанию его значение равно 400 миллисекунд.

Параметр complete представляет функцию, вызываемую методом по завершению анимации

* hide([duration] [, easing][, complete]): то же самое, только добавляется параметр easing, который принимает название функции плавности анимации в виде строки. По умолчанию его значение равно "swing".

Например, можно использовать медленную (hide('slow')) или быструю анимацию (hide('fast')), длительность которых равна соответственно 600 и 200 миллисекунд. Например:

|  |  |
| --- | --- |
|  | $('button#hide').click(function(){          $('ul').hide('slow', function(){              alert('Элемент скрыт');          });      }); |

**Метод show**

Метод show отображает элемент, устанавливая то значение для свойства display, которое было до скрытия. Например, если до скрытия элемента у него был определен стиль display: inline, то и после отображения он будет иметь значение inline. Если же стиль не был определен явно, то он будет иметь значение display: block. Метод show имеет те же самые формы, что и метод hide:

* show();
* show([duration][, complete]);
* show([duration] [, easing][, complete]).

Допустим, что есть блок параграфов на странице. Единожды виден только один. И надо с помощью кнопок переключать параграфы (то есть фактически изменять их видимость):

|  |  |
| --- | --- |
|  | <html>  <head>  <meta charset='utf-8'>  <script src="jquery-1.10.1.min.js"></script>  <style>      #langs p {display:none;}      #langs p.first {display:block;}  </style>  </head>  <body>  <h3>Выбранный язык</h3>  <div id="langs">      <p class='first'>Java</p>      <p>C/C++</p>      <p>PHP</p>      <p class='last'>JavaScript</p>  </div>  <button id="prev">Назад</button>  <button id="next">Вперед</button>  <script type="text/javascript">  $(function() {      let current = $("div#langs p").first();      $('#next').click(function(){           $(current).not('.last').hide("fast", function() {              current = $(current).next('p');              $(current).show("fast");          });      });      $('#prev').click(function(){           $(current).not('.first').hide("fast", function() {              current = $(current).prev('p');              $(current).show("fast");          });      });  });  </script>  </body>  </html> |

Сначала устанавливаем стили, чтобы вначале был видимым только один блок. С помощью выражения let current = $("div#langs p").first(); получаем текущий просматриваемый блок. При загрузке страницы это будет самый первый блок или блок с классом first. И далее, используя селекторы и фильтры, переключаем видимость через методы show и hide и устанавливаем новый текущий просматриваемый блок.

**Метод toggle**

Метод toggle сочетает функционал методов hide и show и, если элемент отображается на странице, то ему присваивается стильdisplay: none. А если элемент скрыт и имеет значение display: none, то он отображается (свойство display получает то значение, которое было до скрытия. Если свойство не было определено явно, то оно будет иметь значение display: block).

Метод toggle имеет те же самые формы, что и метод hide:

* toggle();
* toggle([duration][, complete]);
* toggle([duration] [, easing][, complete]).

1. Эффекты скольжения.

Эффекты скольжения позволяют плавно скрыть или раскрыть элемент. Эффекты скольжения реализованы в виде методов slideUp(), slideDown() и slideToggle().

Если метод slideUp скрывает элемент в направлении вверх, как бы скользя, то метод slideDown плавно раскрывает скрытый элемент в направлении вниз. Метод slideToggle комбинирует действие обоих методов: если элемент скрыт, но раскрывается, если раскрыт – скрывается.

Эти методы имеют одинаковые формы использования:

* slideUp/slideDown/slideToggle(): метод без параметров;
* slideUp/slideDown/slideToggle([duration] [, easing][, complete]). Параметр duration указывает как долго сокрытие элемента будет длиться. По умолчанию его значение равно 400 миллисекунд.

Параметр easing, который принимает название функции плавности анимации в виде строки. По умолчанию его значение равно "swing". Также можно использовать значения 'slow' и 'fast', которые соответствуют длительности эффекта в 600 и 200 миллисекунд.

Параметр complete представляет функцию, вызываемую методом после завершения анимации.

Допустим, есть две кнопки. По нажатию на первую кнопку она будет медленно скрываться, а по нажатию на вторую кнопку скрытая первая кнопка будет раскрываться:

|  |  |
| --- | --- |
|  | <button id="slideUp">Скрыть</button>  <button id="slideDown">Раскрыть</button>  <script type="text/javascript">  $(function() {      $('#slideUp').click(function(){           $(this).slideUp();      });      $('#slideDown').click(function(){           $('#slideUp').slideDown();      });  });  </script> |

Подобным образом можно применить метод slideToggle:

|  |  |
| --- | --- |
|  | $(function() {      $('#slideUp').click(function(){           $(this).slideToggle('slow');      });      $('#slideDown').click(function(){           $('#slideUp').slideToggle(2000);      });  }); |

1. Эффекты прозрачности.

Эффекты прозрачности позволяют, плавно изменяя прозрачность элемента, скрыть его или отобразить. Эффекты прозрачности реализованы с помощью методов fadeOut(), fadeIn(), fadeTo() и fadeToggle().

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| fadeOut | Скрывает элемент, уменьшая его прозрачность |
| fadeIn | Отображает элемент, увеличивая его прозрачность |
| fadeToggle | Сочетает методы fadeOut и fadeIn: если прозрачность равна нулю, то элемент отображается. Если элемент непрозрачен, то он скрывается |
| fadeTo | Изменение прозрачности до указанного уровня |

Методы fadeOut(), fadeIn() и fadeToggle() имеют похожие формы использования:

* fadeOut/fadeIn/fadeToggle(): метод без параметров;
* fadeOut/fadeIn/fadeToggle([duration] [, easing][, complete]). Параметр duration указывает как долго изменение прозрачности элемента будет длиться. По умолчанию его значение равно 400 миллисекунд.

Параметр easing, который принимает название функции плавности анимации в виде строки. По умолчанию его значение равно "swing". Также можно использовать значения 'slow' и 'fast', которые соответствуют длительности эффекта в 600 и 200 миллисекунд.

Параметр complete представляет функцию обратного вызова, вызываемую методом по завершении анимации

Метод fadeTo в отличие от других методов еще принимает и параметр opacity – оно принимает значение, до которого надо изменить прозрачность элемента: fadeTo(duration, opacity [, easing][, complete]). В качестве значение opacity принимается значение от 0 (полностью прозрачный) до 1 (полностью видимый).

Допустим, на странице будет изображение и две кнопки, которые будут изменять прозрачность этого изображения:

|  |  |
| --- | --- |
|  | <img src="ars.jpg" id="ars" /><br>  <button id="fadeIn">Отобразить</button>  <button id="fadeOut">Cкрыть</button>  <script type="text/javascript">  $(function() {      $('#ars').fadeTo(2000,0.6);      $('#fadeIn').click(function(){           $('#ars').fadeIn('slow', function(){alert('Отображено');});      });      $('#fadeOut').click(function(){           $('#ars').fadeOut(2000, function(){alert('Скрыто');});      });  });  </script> |

Обратите внимание, что метод fadeIn увеличивает прозрачность до того значения, которое было до использования метода fadeOut, а не обязательно до 1. То есть в данном случае, поскольку в начале был применен метод fadeTo и прозрачность изменена до 0.6, то и метод fadeIn будет увеличивать прозрачность до 0.6.

1. Создание анимации.

В предыдущих вопросах рассматривались специальные методы, которые управляют высотой элемента, прозрачностью, видимостью. Однако эти методы не покрывают все потребности по созданию эффектов и анимации. Нередко требуются более сложные по характеру анимации. И здесь поможет метод animate().

Метод animate() принимает набор свойств элемента, которые затем изменяются, за счет чего достигается анимация.

Данный метод имеет следующую форму использования: animate(properties [,duration] [,easing] [,complete]).

* обязательный параметр properties содержит набор css-свойств, у которых указываются финальные значения;
* параметр duration указывает, как долго будет длиться изменение прозрачности элемента. По умолчанию его значение равно 400 миллисекунд. Также можно использовать значения 'slow' и 'fast', которые соответствуют длительности эффекта в 600 и 200 миллисекунд;
* параметр easing принимает название функции плавности анимации в виде строки. По умолчанию его значение равно "swing";
* параметр complete представляет функцию обратного вызова, вызываемую методом по завершении анимации.

Используя анимацию, важно иметь в виду, что в данном случае можно использовать только те свойства css, которые принимают числовые значения, например, ширина и высота. Другие же свойства, как, например, цвета фона или шрифта, уже так просто нельзя использовать.

Применим простую анимацию к изображению, изменив ряд его свойств:

|  |  |
| --- | --- |
|  | <img src="ars.jpg" id="ars" /><br>  <button id="anim">Анимация</button>  <script type="text/javascript">  $(function() {      $('#anim').click(function(){           $('#ars').animate({ opacity: 0.25,                  'margin-left': '50',                  height: '200'});      });  });  </script> |

В данном случае изменяют свое значение три свойства: opacity, margin-left и height. Сама анимация представляет переход от начальных значений к значениям, указанным для свойств в методе animate.

В предыдущем примере жестко кодируются финальные значения свойств. Так, левый отступ после анимации будет иметь значение 50. Если надо, чтобы движение постоянно продолжалось при нажатии на кнопку, то в этом случае можно использовать относительные значения:

|  |  |
| --- | --- |
|  | $('#ars').animate({              'margin-left': '+=50',              width: '-=10',              height: '-=10'}, 1000);      }); |

Кроме указания относительных значений здесь также использовано время анимации - 1000 миллисекунд.

Чтобы выполнить более детальную настройку анимации можем использовать еще одну форму метода animate: animate(properties, options).

Здесь используется новый параметр – options. В этом параметре можно указать ряд конфигурационных параметров, которые будут использоваться при анимации. Этот параметр принимает следующие опции:

* duration: продолжительность анимации. По умолчанию равна 400 миллисекунд;
* easing: название функции плавности анимации. По умолчанию значение 'swing';
* queue: булевое значение, указывающее, нужно ли поместить анимацию в очередь эффектов. По умолчанию имеет значение true, что значит, что анимация помещается в очередь. Если же присвоить значение false, то анимация будет выполняться немедленно;
* specialEasing: объект javascript, который сопоставляет анимируемые свойства с функциями плавности;
* step: функция, вызываемая для каждого анимируемого свойства каждого участвующего в анимации элемента;
* progress: функция, вызываемая на каждом этапе анимации по одному разу для каждого элемента вне зависимости от количества анимируемых свойств;
* complete: функция вызываемая после завершения анимации;
* done: функция, вызываемая при завершении анимации;
* fail: функция, вызываемая при ошибке в процессе анимации, если анимация не сможет завершится нормальным путем;
* always: функция, вызываемая после завершения анимации вне зависимости, завершится анимация обычным путем или с ошибкой.

Конечно, все опции разом необязательно определять и можно остановиться лишь на нескольких. Например, применим ряд из этих опций:

|  |  |
| --- | --- |
|  | $('#ars').animate({          'margin-left': '+=50',          width: '-=10',          height: '-=10'          },{          duration: 1000,          step: function(now, fx) {              let data = fx.elem.id + ' ' + fx.prop + ': ' + now;              $('body').append('<div>' + data + '</div>');          },          complete: function() {              alert('Анимация завершена');          }  }); |

Здесь использовались три опции: продолжительность анимации и функции по шаговой обработки и завершения.

Весь процесс анимации разбивается на ряд мелких этапов, которые выполняются в течение определенного времени (в данном случае 1000 миллисекунд). На каждом этапе и вызывается функция пошаговой обработки (step: function(now, fx)), причем для каждого анимируемого элемента анимируемого свойства. Она принимает два параметра: now и fx. Now показывает текущее значение на данном этапе анимируемого свойства, а fx содержит данные о анимируемом объекте. Как в данном случае, получаем анимируемое свойство (fx.prop) и анимируемый элемент (fx.elem). И затем эти данные можно использовать по своему усмотрению.

1. Введение в jQuery UI.

Фреймворк jQuery позволяет не только управлять элементами DOM, отправлять запросы AJAX, но представляет неплохой инструмент для построения пользовательского интерфейса с помощью виджетов. Для работы с интерфейсом предназначена специальная библиотека jQuery UI.

Библиотека jQuery UI загружается отдельно от основной библиотеки jQuery: ее можно найти по адресу <https://jqueryui.com/download/>. По этому адресу внизу страницы располагается список тем библиотеки. Темы влияют на оформление применяемых компонентов. Но в начале работы выбор темы не столь важен, поэтому можно оставить стандартную тему UI lightness. Также на странице загрузок можно детализировать загрузку – что будет входить в загружаемый пакет, но можно оставить все установки по умолчанию и внизу страницы нажать кнопку Download для загрузки.

Также можно использовать сети CDN, поскольку библиотека довольно популярная, то она есть во многих CDN.

При распаковке пакета, в нем будут несколько папок и файлов в состав которых включены все опции, что были выбраны на странице загрузки jQueryUI. А также файл index.html, содержащий примеры использования виджетов и логики из библиотеки jQueryUI. Если открыть данный файл в текстовом редакторе, то увидим, что он ссылается на выбранную нами тему, библиотеку jQuery и библиотеку jQuery UI. В начале файла будет подключен файл стилей jquery-ui:

|  |  |
| --- | --- |
|  | <link href="jquery-ui.css" rel="stylesheet"> |

А ближе к концу файла будет поключена библиотека jQuery и библиотека jQuery UI:

|  |  |
| --- | --- |
|  | <script src="external/jquery/jquery.js"></script>  <script src="jquery-ui.js"></script> |

Как правило, чтобы воспользоваться возможностями jquery, нужно будет подключить эти три файла на нашу html-страницу. Их можно подключать как и из пакета, скачанного с официального сайта, так и из сетей CDN.

Чтобы подключить библиотеку из загруженного пакета, надо в распакованном пакете создать следующую страницу html:

|  |  |
| --- | --- |
|  | <html>      <head>          <link rel="stylesheet" type="text/css" href="jquery-ui.css">          <script src="external/jquery/jquery.js"></script>          <script src="jquery-ui.js"></script>      </head>      <body>       <p><a>Hello</a></p>       <script type="text/javascript">       $(function(){          $('a').button().click(function(){alert('Hello jQuery UI');});       });       </script>      </body>  </html> |

Здесь просто стандартный элемент ссылки заменяется на виджет button, и к нему добавляется обработчик нажатия. Сначала обязательно подключается основная библиотека jQuery, поскольку jQuery UI от нее зависит. Можно загрузить из CDN.

1. Виджет button.

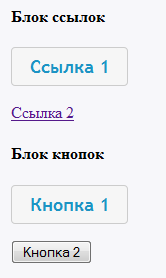
Виджет button создает кнопку и применяется к одиночным элементам. Например:

|  |  |
| --- | --- |
|  | <html>      <head>          <meta charset='utf-8'>          <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.min.css">          <script src="jquery-2.0.2.min.js"></script>          <script src="jquery-ui-1.10.3.custom.min.js"></script>      </head>      <body>          <h4>Блок ссылок</h4>          <p><a class="a\_jqui" href="[https://google.com](https://google.com/)">Ссылка 1</a></p>          <p><a href="[https://google.com](https://google.com/)">Ссылка 2</a></p>            <h4>Блок кнопок</h4>          <p><button class="button\_jqui">Кнопка 1</button></p>          <p><button>Кнопка 2</button></p>           <script type="text/javascript">           $(function(){                $('.a\_jqui').button().click(function(e){                  e.preventDefault();                  alert('Виджет button');              });              $('.button\_jqui').button();           });           </script>      </body>  </html> |

В итоге после действия виджета button первая ссылка получит следующую разметку:

|  |  |
| --- | --- |
|  | <a aria-disabled="false" role="button" class="a\_jqui ui-button ui-widget ui-state-default ui-corner-all ui-button-text-only"  href="[https://google.com](https://google.com/)"><span class="ui-button-text">Ссылка 1</span></a> |

В итоге визуально это будет выглядеть так:



В зависимости от выбранной темы подключаемых стилей jquery-ui внешний вид создаваемых кнопок может немного отличаться, однако можно настроить стиль самостоятельно. Например, на приведенном выше изображении видно, что кнопки слишком большие, и можно сделать их меньше, установив свойства у класса ui-button:

|  |  |
| --- | --- |
|  | <style>      .ui-button {font-size: 13px;}  </style> |

Можно управлять виджетом button через ряд методов:

* button("disable"): делает кнопку недоступной

|  |  |
| --- | --- |
|  | $(function(){        $('.a\_jqui').button().click(function(e){          e.preventDefault();          $('.button\_jqui').button('disable');      });      $('.button\_jqui').button();  }); |

Обратите внимание, что если ранее не был применен к элементам виджет button, то нельзя будет использовать его методы.

* button("enable"): делает кнопку доступной для взаимодействия;
* button("destroy"): удаляет функциональность виджета button у элемента;
* button("widget"): возвращает объект jQuery, представляющий виджет (let widget = $(".button\_jqui").button("widget"););
* button("refresh"): обновляет визуальное состояние элемента;
* button("option"): позволяет получить или установить значения свойств кнопки:

|  |  |
| --- | --- |
|  | <p><button class="get\_option">Получить значение</button></p>  <p><button class="set\_option">Установить значение</button></p>    <script type="text/javascript">  $(function(){        $('button.get\_option').button().click(function(){          // получаем значение          let isDisabled=$('button.get\_option').button('option', 'disabled');          console.log(isDisabled);      });      $('button.set\_option').button().click(function(){          // устанавливаем значение          $('button.set\_option').button('option', 'disabled', true);      });  });  </script> |

Для получения значения в качестве второго параметра после "option" передается название свойства, которое надо получить. А для установки добавляется третий параметр, который указывает на новое значение свойства, переданного во втором параметре.

Кроме методов у виджета button определено также несколько свойств.

* Свойство disabled. Выше уже использовалось это свойство для отключения кнопки. Но кроме того, можно использовать следующий синтаксис для его установки: $('button.get\_option').button({disabled:true}).
* Свойство label: это свойство устанавливает надпись кнопки. ($('button.get\_option').button({label:'Кнопка 1'})). Если в разметке html кнопки задано изначальная надпись, то установка свойства label заменяет ее.
* Свойство icons устанавливает иконку на кнопке. Весь список возможных иконок можно найти на офсайте по адресу<https://api.jqueryui.com/theming/icons/>. В загружаемом пакете с jquery-ui эти иконки находятся в папке css/images.

|  |  |
| --- | --- |
|  | <button class="help">Помощь</button>  <script type="text/javascript">  $(function(){      $('button.help').button({icons: { secondary: "ui-icon-help" }});  });  </script> |

В данном случае устанавливается иконка после текста кнопки:



* Если надо установить иконку перед текстом, то используется опция primary: $('button.help').button({icons: { primary: "ui-icon-help" }});. Также при желании можно задать иконки до и после текста, использовав сразу две опции.
* Свойство showLabel указывает, должна ли отображаться надпись на кнопке.$('button.get\_option').button({showLabel:false}) – в данном случае, если для кнопки установлено свойство icons, то есть заданы иконки, то надпись не будет отображаться, а будет отображаться заданная иконка. Если свойство icons не указано, то свойство showLabel игнорируется.

Если виджет button позволяет создать кнопку из одного элемента, то buttonset применяется для создания группы кнопок. Например, можно создать группу флажков (checkbox) или переключателей (radiobutton). При этом buttonset применяется не к самим кнопкам, а к их контейнерам:

|  |  |
| --- | --- |
|  | <div id="langs">      <input type="radio" id="java" name="lang" checked><label for="java">Java</label>      <input type="radio" id="csharp" name="lang"><label for="csharp">C#</label>      <input type="radio" id="php" name="lang"><label for="php">PHP</label></p>  </div>  <script type="text/javascript">  $(function(){        $('#langs').buttonset();  });  </script> |

Аналогично можно установить виджет и для группы флажков.

1. Виджет progressbar.

Виджет progressbar используется для создания индикатора процесса. Он сообщает пользователю о ходе выполнения какой-нибудь задачи. Например:

|  |  |
| --- | --- |
|  | <div id="progress"></div>  <script type="text/javascript">  $(function(){        $('#progress').progressbar();  });  </script> |

Виджет обладает несколькими свойствами, которые помогают выполнить его настройку:

* value: текущее значение прогрессбара. Например: $('#progress').progressbar({value:33}). По умолчанию значение свойства равно 0 и не может быть выше значения свойства max. Также может принимать булевое значение: если имеет значение false, то прогрессбар имеет неопределенную шкалу значений.
* disabled: отключает прогрессбар, если установлено значение true ($('#progress').progressbar({disabled:true, value:25})). По умолчанию имеет значение false.
* max: указывает максимальное значение прогрессбара. Например, $('#progress').progressbar({max:200}). По умолчанию имеет значение 100.

Прогрессбар имеет следующие методы:

* progressbar("disable"): делает прогрессбар недоступным: $("#progress").progressbar("disable");
* progressbar("enable"): делает прогрессбар доступным для взаимодействия;
* progressbar("destroy"): удаляет функциональность виджета progressbar у элемента;
* progressbar("widget"): возвращает объект jQuery, представляющий виджет (let widget = $("#progress").progressbar("widget"););
* progressbar("option"): позволяет получить или установить значения свойств прогрессбара:

|  |  |
| --- | --- |
|  | <div id="progress"></div>  <p><button id="getVal">Получить значение</button></p>  <p><input type="text" id="val" /> <button id="setVal">Установить значение</button></p>  <script type="text/javascript">  $(function(){        $('#progress').progressbar({value:25});      $('#getVal').button().click(function(){          // получаем значение          let val=$('#progress').progressbar('option', 'value');          alert("Значение индикатора: "+val);      });      $('#setVal').button().click(function(){          // устанавливаем значение          let val = parseInt($('#val').val());          if(val>100 || val<0){              alert('Введено недопустимое значение');          }          else{              $('#progress').progressbar('option', 'value', val);          }      });  });  </script> |

С помощью первой кнопки получаем значение, передавая в качестве второго параметра после "option" название свойства, которое надо получить. А с помощью второй кнопки у свойства прогрессбара устанавливается значение, введенное в текстовое поле – его передаем в качестве третьего параметра метода.

Дополнительно виджет реализует метод progressbar("value"), который возвращает значение свойства value. Передавая в качестве второго параметра числовое значение, мы можем установить значение индикатора: $('#progress').progressbar('value', 35).

Для прогрессбара определены три события:

* change(event, ui): возникает при изменении значения прогрессбара, например:

|  |  |
| --- | --- |
|  | $('#progress').progressbar({value:25, change: function(event, ui){              console.log('value has changed');          }      }); |

* complete(event, ui): возникает, когда значение прогрессбара будет равно max. Например:

|  |  |
| --- | --- |
|  | $('#progress').progressbar({complete: function(event, ui){              console.log('value is equal to maximum');          }      }); |

* create(event, ui): возникает при создании виджета

1. Виджет слайдер jQuery.

Виджет slider создает функциональность ползунка:

|  |  |
| --- | --- |
|  | <div id="slider"></div>  <script type="text/javascript">  $(function(){        $('#slider').slider();  });  </script> |

Используя классы, создаваемые виджетом, можно настроить его стилизацию. При применении виджета к элементу добавляется класс ui-slider. Настроим стилизацию этого класса:

|  |  |
| --- | --- |
|  | <html>  <head>  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.min.css"></style>  <script src="jquery-2.0.2.min.js"></script>  <script src="jquery-ui-1.10.3.custom.min.js"></script>  <style>  .ui-slider {      background: #FF9999;      margin: 10%;      width: 200px;  }  </style>  </head>  <body>  <div id="slider"></div>  <script type="text/javascript">  $(function(){        $('#slider').slider();  });  </script>  </body>  </html> |

Используя следующие свойства слайдера, можно выполнить более точную его настройку:

* aimate: устанавливает скорость анимации. Можно передать в качестве значения названия скоростей, например, "fast" (быстрая анимация) и "slow" (медленная анимация).($('#slider').slider({ animate: 'slow' })). Также можно задать числовое значение, указывающее продолжительность анимации в миллисекундах ($('#slider').slider({ animate: 900 }));
* disabled: при установке значения true делает слайдер недоступным;
* max: максимальное значение слайдера, по умолчанию равно 100;
* min: минимальное значение слайдера, по умолчанию равно 0;
* orientation: ориентация слайдера, может быть горизонтальной (значение по умолчанию – "horizontal"), так и вертикальной ("vertical"). Например: $('#slider').slider({orientation:'vertical'});
* range: указывает, будет ли стилизоваться та часть ползунка, которая была пройдена бегунком. Может принимать булевое значение. Если имеет значение true, то прошедшая часть слайдера стилизуется. По умолчанию имеет значение false, поэтому никакой стилизации не происходит. Также может иметь строковые значения min и max. Если равно min, то стилизация, то есть окрашивание будет производиться сначала отсчетов слайдера, если max – то с конца: $('#slider').slider({range: 'max' });
* step: шаг приращения бегунка слайдера, по умолчанию равен 1;
* value: определяет значение бегунка слайдера;
* values: с помощью этого свойства можно задать у слайдера сразу несколько бегунков: $('#slider').slider({values: [ 10, 25, 50]}).

Слайдер обладает следующими методами:

* slider("disable"): делает слайдер недоступным: $("#slider").slider("disable");
* slider("enable"): делает слайдер доступным для взаимодействия;
* slider("destroy"): удаляет функциональность виджета slider у элемента;
* slider("widget"): возвращает объект jQuery, представляющий виджет (let widget = $("#slider").slider("widget"););
* slider("option"): позволяет получить или установить значения свойств слайдера. Например, установим и получим значение свойства value:

|  |  |
| --- | --- |
|  | $('#slider').slider();  // установка  $('#slider').slider('option', 'value', 30);  // получение значения  let val=$('#slider').slider('option', 'value');  console.log(val); |

* slider("value"): возвращает или устанавливает значение свойства value. Например, получим значение$('#slider').slider('value');. А передавая в качестве второго параметра числовое значение, можно установить значение бегунка: $('#slider').slider('value', 44);

Поскольку у слайдера может быть несколько бегунков, то у метода slider('value') есть аналог, помогающий получить или установить сразу несколько значений - для каждого бегунка. Например, получим все значения:

|  |  |
| --- | --- |
|  | // создание виджета  $('#slider').slider({values: [10,80]});  // установка значений  $('#slider').slider('values', [30,60]);  // получение значений  let vals=$('#slider').slider('values');  console.log(vals); |

Слайдер обладает несколькими событиями, которые помогают отслеживать его изменения:

* change(event, ui): событие возникает, когда пользователь двигает бегунок слайдера, а также когда значение свойстваvalue изменяется программным способом;
* create(event, ui): возникает при создании виджета;
* slide(event, ui): возникает при перемещении мыши вдоль полосы слайдера;
* start(event, ui): событие возникает, когда пользователь начинает перемещать бегунок слайдера;
* stop(event, ui): событие возникает, когда пользователь завершает перемещать бегунок слайдера.

Применим события слайдера:

|  |  |
| --- | --- |
|  | <h3>Слайдер</h3>  <div id="slider"></div> <br />  Текущее значение слайдера: <span id="sliderValue"></span>  <script type="text/javascript">  $(function(){        $('#slider').slider({          change: function (event, ui){              let val = $('#slider').slider('value');              $('#sliderValue').html(val);          },          start: function(event, ui){              console.log('Начало движения слайдера');          },          slide: function(event, ui){              console.log('Перемещение слайдера');          },          stop: function(event, ui){              console.log('Завершение движения слайдера');          }      });  });  </script> |

1. Виджет Accordion.

Виджет accordion используется для создания аккордеонов:

|  |  |
| --- | --- |
|  | <h3>Языки программирования</h3>  <div id="accordion">  <h3>Java</h3>  <div>Описание языка Java</div>  <h3>JavaScript</h3>  <div>Описание языка JavaScript</div>  <h3>C#</h3>  <div>Описание языка C#</div>  </div>    <script type="text/javascript">  $(function(){        $('#accordion').accordion();  });  </script> |

Можно выполнить настройку стилей, применив их к классу ui-accordion. Аккордеон структурно подразделяется на заголовок и основную часть. И кроме стилей аккордеона в целом, также можно настроить по отдельности стили заголовка, применив их к классу ui-accordion-header, и стили основной части - их применяют к классу ui-accordion-content:

|  |  |
| --- | --- |
|  | <html>  <head>  <meta charset='utf-8'>  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.min.css"></style>  <script src="jquery-2.0.2.min.js"></script>  <script src="jquery-ui-1.10.3.custom.min.js"></script>  <style>      .ui-accordion {font-size: 12px;}      .ui-accordion-header {font-size: 13px;}      .ui-accordion-content {font-style: italic;}  </style>  </head>  <body>  <h3>Языки программирования</h3>  <div id="accordion">  <h3>Java</h3>  <div>Описание языка Java</div>  <h3>JavaScript</h3>  <div>Описание языка JavaScript</div>  <h3>C#</h3>  <div>Описание языка C#</div>  </div>  <script type="text/javascript">  $(function(){        $('#accordion').accordion();  });  </script>  </body>  </html> |

Аккордеоны в jquery ui имеет следующие свойства:

* collapsible: при установке этого свойства в true при нажатии на заголовок можно скрывать содержимое блока ($('#accordion').accordion({collapsible:false});). По умолчанию имеет значение false, поэтому скрыть один блок можно только нажав на заголовок другого блока.
* active: указывает, какой блок в текущий момент является активным. Например, $('#accordion').accordion({active:2}); – здесь при создании виджета будет раскрыта третья вкладка, так как нумерация начинается с 0. По умолчанию активной является первая вкладка.
* animate: данное свойство указывает, будет ли виджет анимироваться, а если будет, то как. Данное свойство может принимать ряд значений.

Если свойство имеет значение false, то анимация отключается. Это проявляется в мгновенном открытии новых блоков, а при true подобное открытие происходит плавно.

Также данное свойство может иметь числовое значение, которое устанавливает время анимации в миллисекундах. Значение может представлять строковое название функции плавности анимации, например, linear, swing, easeInCubic и другие функции. Например: $('#accordion').accordion({animate:'easeInCubic'}).

Можно установить в качестве значения объект, включающий как название функции плавности, так и длительность анимации в миллисекундах: $('#accordion').accordion({animate: {easing:'easeInCubic', duration: 800}});.

* disabled: при установке у свойства значения true отключает виджет. По умолчанию имеет значение false.
* event: данное свойство в качестве значения принимает событие, при возникновении которого будет происходить переключение блоков в аккордеоне: $('#accordion').accordion({event:'mouseover'}) – здесь переключение блоков будет осуществляться при наведении мыши (событие mouseover) на заголовок блока. По умолчанию переключение происходит по событию click.
* icons: используя это свойство, можно задать иконку, отображаемую рядом с заголовком блока. По умолчанию это свойство имеет значение {"header": "ui-icon-triangle-1-e", "activeHeader": "ui-icon-triangle-1-s"} – в данном случае опция ui-icon-triangle-1-e указывает иконку для всех заголовков блоков, а опция activeHeader – для активных.

Можно переопределить иконку: $('#accordion').accordion({icons: {"header": "ui-icon-plus", "activeHeader": "ui-icon-minus"}}). Полный список используемых иконок можно найти по адресу <https://api.jqueryui.com/theming/icons/>

Виджет аккордеон имеет следующие методы:

* accordion("disable"): делает аккордеон недоступным ($("#accordion").accordion("disable");)
* accordion("enable"): делает аккордеон доступным для взаимодействия;
* accordion("destroy"): удаляет функциональность аккордеона у элемента;
* accordion("widget"): возвращает объект jQuery, представляющий виджет (let widget = $("#accordion").accordion("widget"););
* accordion("option"): позволяет получить или установить значения свойств аккордеона. Например, установим и получим значение свойства collapsible:

|  |  |
| --- | --- |
|  | $('#accordion').accordion();  // установка  $('#accordion').accordion('option', 'collapsible', true);  // получение значения  let isCollapsible=$('#accordion').accordion('option', 'collapsible');  console.log(isCollapsible); |

Аккордеоны имеют следующие события:

* activate(event, ui): событие возникает после выбора блока в аккордеоне, то есть его активации.

Если параметр event содержит объект события, то параметр ui представляет объект, который содержит информацию об активированном блоке, а также о том блоке, который до этого был активным. Этот параметр имеет следующие свойства:

* newHeader: заголовок блока, который активирован;
* oldHeader: заголовок блока, который до этого был активным;
* newPanel: панель блока, который активирован;
* oldPanel: панель блока, который был активным до выбора текущего активного блока.

Например, получим заголовки текущего активного блока и блока, который ранее был активным:

|  |  |
| --- | --- |
|  | $('#accordion').accordion({      activate: function( event, ui ) {            console.log("Была активной панель: " + $(ui.oldHeader).text());          console.log("Стала активной панель: " + $(ui.newHeader).text());      }  }); |

* create(event, ui): возникает при создании виджета;
* beforeActivate(event, ui): возникает непосредственно перед активацией блока аккордеона.

1. Виджет Autocomplete.

Виджет autocomplete предназначен для создания специальных полей ввода с функцией автозаполнения:

|  |  |
| --- | --- |
|  | <html>  <head>  <meta charset='utf-8'>  <link rel="stylesheet" type="text/css" href="jquery-ui-1.10.3.custom.min.css"></style>  <script src="jquery-2.0.2.min.js"></script>  <script src="jquery-ui-1.10.3.custom.min.js"></script>  </head>  <body>  <input type="text" id="lang" />    <script type="text/javascript">  $(function(){        let langs = ["Java", "JavaScript", "VisualBasic", "PHP", "Pascal"] ;      $('input#lang').autocomplete({          source: langs      });    });  </script>  </body>  </html> |

В качестве источника автозаполнения указываем какой-нибудь массив и потом применяем виджет.

В предыдущем примере источник для списка автозаполнения задавался локально. Однако можно также динамически подключаться к какому-нибудь скрипту на сервере, который бы возвращал список вариантов для автозаполнения. В данном случае это будет скрипт php. Пусть он выглядит следующим образом:

|  |  |
| --- | --- |
|  | <?php      $filter = $\_GET['filter'];        $results = array('Java', 'JavaScript', 'VisualBasic', 'Pascal');      $output='[';      foreach ($results as $key => $value) {          if(substr\_count($value, $filter)>0){              $output .= '{ "label" : "' . $value . '", "value" : "' . $value . '"},';          }      }      // обрезаем последнюю запятую и закрываем кавычки      $output = substr($output,0,-1) . ']';      echo $output;  ?> |

Выходная переменная $output содержит весь список в формате json. По сути каждый элемент списка – это объект, у которого определены два свойства: label (отображаемая метка) и value (само значение). В данном случае оба свойства по значению совпадают. На стороне клиента код получения данных мог бы выглядеть следующим образом:

|  |  |
| --- | --- |
|  | $('input#lang').autocomplete({ source: function(request,response) {        $.getJSON('json.php', { filter : $('input#lang').val() }, function(data){            let suggestions = []; // массив для хранения результатов          $.each(data, function(key, val) {                    suggestions.push(val.value); // добавляем все элементы              });              response(suggestions);          });      }  }); |

Таким образом, динамически получаем введенные в поле ввода символы и передаем их в запросе на сервер. На сервере эти символы используются в качестве фильтра, а в ответ клиенту возвращается набор отфильтрованных записей.

При применении виджета к элементу input к данному элементу добавляется класс ui-autocomplete-input. А область списка автозаполения представляет собой список ul:

|  |  |
| --- | --- |
|  | <ul style="display: none; top: 30px; left: 8px; width: 143px;" tabindex="0" id="ui-id-1"  class="ui-autocomplete ui-front ui-menu ui-widget ui-widget-content ui-corner-all">  <li role="presentation" class="ui-menu-item"><a tabindex="-1" class="ui-corner-all" id="ui-id-9">Java</a></li>  <li role="presentation" class="ui-menu-item"><a tabindex="-1" class="ui-corner-all" id="ui-id-10">JavaScript</a></li>  </ul> |

Изменив стили соответствующих классов, можно настроить отображение списка:

|  |  |
| --- | --- |
|  | <style>      li.ui-menu-item {          font-size: 12px;          font-faminy: Verdana;      }  </style>  </head>  <body>  <input type="text" id="lang" />    <script type="text/javascript">  $(function(){        let langs = ["Java", "JavaScript", "VisualBasic", "PHP", "Pascal"] ;      $('input#lang').autocomplete({          source: langs      });  });  </script>  </body> |

Виджет имеет следующие свойства:

* appendTo: указывает, какой элемент будет включать список для автозаполенния:

|  |  |
| --- | --- |
|  | <input type="text" id="lang" />  <div id="langsList"></div>  <script type="text/javascript">  $(function(){        let langs = ["Java", "JavaScript", "VisualBasic", "PHP", "Pascal"] ;      $('input#lang').autocomplete({ appendTo: "#langsList", source: langs });  });  </script> |

* autoFocus: при установке данного свойства в true выделяется первый элемент списка автозаполнения. По умолчанию свойство имеет значение false.
* delay: указывает в миллисекундах время задержки до показа списка автозаполенния ($('#langs').autocomplete({ delay: 500 });). По умолчанию данное свойство равно 300.
* disabled: при установке у свойства значения true отключает виджет. По умолчанию имеет значение false.
* minLength: указывает на минимальное количество символов, которое должен ввести пользователь, чтобы отобразился список автозаполнения: $('#langs').autocomplete({ minLength: 0 });. По умолчанию данное свойство имеет значение 1.
* position: используя данное свойство, можно задать положение списка на странице. Объект, с помощью которого можно задать положение, использует ряд опций, список которых можно найти на странице <https://api.jqueryui.com/position/>. По умолчанию свойство имеет значение { my: "left top", at: "left bottom", collision: "none" }. Опция my указывает на позицию элемента списка автозаполнения, относительно которой идет выравнивание с элементом ввода. То есть left top выравнивает относительно левого верхнего угла. Опция at указывает уже позицию элемента ввода, относительно которой будет выравниваться список автозаполнения. Опция collision дополнительно настраивает отображение списка.

Можно изменить начальные параметры, например, $('input#lang').autocomplete({ position: { my : "right top", at: "right bottom", collision: 'fit' }, source: langs }).

* source: задает источник для автозаполнения. Это может быть массив, который можно задать в качестве внешнего массива или прямо при свойстве: $('input#lang').autocomplete({ source: ["Java", "JavaScript", "VisualBasic", "PHP", "Pascal"]}).

Виджет autocomplete имеет следующие методы:

* autocomplete("close"): закрывает список ($("#lang").autocomplete("close"););
* autocomplete("disable"): отключает виджет;
* autocomplete("enable"): подключает виджет автозаполнения;
* autocomplete("destroy"): удаляет функциональность автозаполнения у элемента;
* autocomplete("widget"): возвращает объект jQuery, представляющий виджет (let widget = $("#lang").autocomplete("widget"););
* autocomplete("option"): позволяет получить или установить значения свойств виджета. Например, установим и получим значение свойства collapsible:

|  |  |
| --- | --- |
|  | let langs = ["Java", "JavaScript", "VisualBasic", "PHP", "Pascal"] ;  $('input#lang').autocomplete({ source: langs});  $('input#lang').autocomplete( "option", { autoFocus: true } ); |

* autocomplete("search", "value"): производит поиск по списку так с учетом значения value. Например, метод $("#lang").autocomplete("search", "i"); будет эквивалентен действию ввода символа "i" в текстовое поле. И затем буду отображены все элементы списка, в которых присутствует символ "i".

С помощью обработки следующих событий виджета можно произвести определенные действия в ответ на действия пользователя:

* change(event, ui): событие возникает после изменения выбранного элемента.

Параметр event содержит объект события, а параметр ui представляет объект, который содержит информацию о выбранном элементе:

|  |  |
| --- | --- |
|  | let langs = ["Java", "JavaScript", "VisualBasic", "PHP", "Pascal"] ;  $('input#lang').autocomplete({ source: langs});  $('input#lang').autocomplete({      change: function( event, ui ) {          console.log(ui.item.value);      }  }); |

Используя свойство ui.item, получаем выбранный элемент. Элемент определен в виде объекта {label, value}, поэому, чтобы получить значение выбранного элемента, используем свойство value.

* close(event, ui): возникает при закрытии списка автозаполнения;
* create(event, ui): возникает при создании виджета;
* focus(event, ui): возникает при передаче фокуса элементу;
* open(event, ui): возникает при открытии или отображении списка автозаполнения;
* response(event, ui): возникает после завершения поиска, но перед тем, как список результатов автозаполнения появится на экране. Чтобы получить все результаты списка, надо использовать объект ui.content, который представляет массив:

|  |  |
| --- | --- |
|  | let langs = ["Java", "JavaScript", "VisualBasic", "PHP", "Pascal"] ;  $('input#lang').autocomplete({ source: langs});  $('input#lang').autocomplete({      response: function( event, ui ) {          for ( let i = 0; i<ui.content.length; i++ ) {                console.log(ui.content[i].value);          }      }  }); |

* search(event, ui): возникает перед выполнением поиска;
* select(event, ui): возникает при выборе элемента из списка:

|  |  |
| --- | --- |
|  | let langs = ["Java", "JavaScript", "VisualBasic", "PHP", "Pascal"] ;  $('input#lang').autocomplete({ source: langs});  $('input#lang').autocomplete({      select: function( event, ui ) {            console.log(ui.item.value);      }  }); |