**Тема 11. События jQuery и технология AJAX**

1. **Обработчики событий.**
2. **Введение в AJAX.**
3. **Метод load.**
4. **GET- и POST-запросы.**
5. **Получение данных JSON.**

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

1. Обработчики событий.

События позволяют отследить изменение состояния элемента и обработать пользовательский ввод. Используя стандартный код JavaScript можно обработать событие прямо в разметке элемента с помощью соответствующих атрибутов. Например, нажатие кнопки:

|  |  |
| --- | --- |
|  | <button onclick="alert('Привет мир');">Нажать</button> |

Для обработки нажатия кнопки задаются некоторые действия, например, как в данном случае вывод сообщения.

Библиотека jQuery предлагает эффектный способ регистрации обработчиков событий у элементов с помощью метода bind. Метод bind имеет следующий синтаксис: bind('тип\_события', функция\_обработчика\_события). Например, обработаем нажатие кнопки с помощью jQuery:

|  |  |
| --- | --- |
|  | <button class="redStyle">Нажать</button>  <script type="text/javascript">  $(function(){      $('button').bind('click', function(){          alert('Привет мир!');      });  });  </script> |

Используя в обработчике выражение $(this) можно получить элемент, на котором было произведено событие. Используя свойства элемента, можно задать более сложную логику обработки события. Например, на странице есть несколько кнопок, и действия в зависимости от нажатой кнопки будут различаться:

|  |  |
| --- | --- |
|  | <button class="langs">Языки программирования</button>  <script type="text/javascript">  $(function(){      $('button').bind('click', function(){          if($(this).hasClass("langs")){              $('body').append('<ul><li>Java</li><li>JavaScript</li></ul>');          }          alert('Обработчик события click');      });  });  </script> |

Таким образом, если на странице несколько кнопок, то только в случае с одной по нажатию на нее будет добавляться на страницу список. В остальных случаях будет просто выводиться сообщение на экран.

Применяя метод bind, важно учитывать, что он добавляет обработчики для уже имеющихся элементов. К динамически добавленным элементам обработчики применяться не будут.

Если по какой-то причине больше не надо обрабатывать событие, то можно отменить регистрацию связанного с ним обработчика с помощью метода unbind:

|  |  |
| --- | --- |
|  | $('button').bind('click', function(e){      alert('Привет мир!');  });  $('button').unbind(); |

Метод one позволяет задать для элемента обработчик события, который будет срабатывать только один раз. После отработки произойдёт удаление обработчика:

|  |  |
| --- | --- |
|  | <button>Жми1</button>  <button>Жми2</button>  <button>Жми3</button>  <script type="text/javascript">  $(function(){      $('button').one('click', function(e){          $(this).css('color', 'red');          alert('Обработчик для кнопки: '+$(this).text());      });  });  </script> |

В результате обработчик каждой кнопки сработает по разу.

**Параметры обработчика.**

С помощью объекта Event в обработчик события передается ряд параметров, связанных с событием. Затем в обработчике можно извлечь значения этих параметров и как-то применить при обработке.

Некоторые распространенные свойства объекта Event:

* currentTarget: текущий элемент, на котором обрабатывается событие;
* data:представляет собой объект, в который передаются дополнительные данные связанные с событием, например, в методе bind;
* target: элемент, инициировавший событие;
* timestamp: время наступления события;
* pageX: координаты указателя мыши относительно левой границы документа;
* pageY: координаты указателя мыши относительно верхней границы документа;
* which: в обработчиках событий клавиатуры или мыши объект, содержащий данные о нажатой клавише или кнопке;

Например, обработаем нажатие на страницы клавиш клавиатуры и получим нажатую клавишу:

|  |  |
| --- | --- |
|  | $(function(){      $(document).bind('keydown', function(e){          console.log(e.which);      });  }); |

Здесь регистрируется обработчик клавиатуры для всего документа и в случае нажатия на консоль будет выведен числовой код клавиши. Получив данный числовой код, можно произвести некоторые действия.

Используя метод bind можно передать данные в параметр data события в виде необязательного второго параметра:

|  |  |
| --- | --- |
|  | let mes = 'Привет мир!';  $('button').bind('click', mes, function(e){          alert(e.data);      }); |

В данном случае переменная mes и будет передаваться в свойство e.data, из которого потом и сможем получить ее значение.

Также объект Event предоставляет ряд методов:

* preventDefault(): отменяет все действия, которые по умолчанию производит данное событие. Например, при нажатии на ссылку можно с помощью данного метода запретить переход по этой ссылке;
* isDefaultPrevented(): возвращает true, если для данного события уже ранее вызывался метод preventDefault;
* stopPropagation(): запрет всплытия события вверх по иерархии DOM и его последующей обработки;
* isPropagationStopped(): возвращает true, если ранее уже был вызван метод stopPropagation.

**Всплытие событий.**

Допустим, есть следующая структура документа:

|  |  |
| --- | --- |
|  | <body>      <div>          <button>Жми</button>      </div>  </body> |

Таким образом, у нас трехуровневая структура: на самом высоком уровне – элемент body, ниже – элемент div, и в самом низу – элемент button. Теперь добавим для каждого последовательно обработчики нажатия:

|  |  |
| --- | --- |
|  | $('button').bind('click', function(e){          console.log('Событие инициировал элемент ' + e.target.tagName);          console.log('Событие обрабатывается для элемента ' + e.currentTarget.tagName);      });      $('div').bind('click', function(e){          console.log('Событие инициировал элемент ' + e.target.tagName);          console.log('Событие обрабатывается для элемента ' + e.currentTarget.tagName);      });      $('body').bind('click', function(e){          console.log('Событие инициировал элемент ' + e.target.tagName);          console.log('Событие обрабатывается для элемента ' + e.currentTarget.tagName);      }); |

Итак, при нажатии на кнопку сработает не только собственно обработчик кнопки, но и обработчики выше лежащих уровней. Так как нажатие на кнопку – это в то же время нажатие и на элемент div и на элемент body, так как кнопка находится внутри них. Произойдет всплытие событие. То есть с каждым уровнем выше будут просматриваться обработчики события click и, если они имеются, они будут вызываться.

Однако можно запретить всплытие. Например, надо, чтобы срабатывал только обработчик нажатия кнопки. В этом случае нужно применить метод stopPropagation:

|  |  |
| --- | --- |
|  | $('button').bind('click', function(e){      console.log('Событие инициировал элемент ' + e.target.tagName);      console.log('Событие обрабатывается для элемента ' + e.currentTarget.tagName);      e.stopPropagation();  }); |

Теперь обработчики вышележащих уровней не будут задействованы.

**Методы delegate и on.**

Ранее упоминалось, что метод bind добавляет обработчики событий только к тем элементам, которые на момент его действия уже существуют в структуре DOM. При добавлении новых элементов, например, с помощью метода append их события не будут обрабатываться. Но jQuery предоставляет специальный метод delegate, который позволяет создать обработчики и для будущих элементов, которых еще не в структуре DOM. Этот метод имеет следующие варианты использования:

* delegate('селектор','событие', обработчик\_события): добавляет обработчик для события. Для фильтрации элементов, для которых добавляется обработчик, используется селектор;
* delegate('селектор','событие', данные\_события, обработчик\_события): то же самое, что и в предыдущем случае, плюс параметр для передачи данных события в объект Event (e.data).

Применение метода delegate:

|  |  |
| --- | --- |
|  | <button>Жми1</button>  <button>Жми2</button>    <script type="text/javascript">  $(function(){      $('body').delegate('button','click', function(e){          $(this).css('color', 'red');          alert('Обработчик для кнопки: '+$(this).text());      });      $('body').append('<button>Жми3</button>');  });  </script> |

Здесь для всех элементов button, находящихся в элементе body, добавляется обработчик нажатия. И несмотря на то, что уже после этого добавляется новая кнопка, для нее также будет срабатывать обработчик.

Также можно отменить регистрацию обработчика события с помощью метода undelegate:

|  |  |
| --- | --- |
|  | $('body').undelegate('button','click'); |

Метод on во многом является заменой методу delegate и имеет аналогичное действие. Он имеет похожие формы использования:

* on('событие', 'селектор', обработчик\_события);
* on('событие', 'селектор', данные\_события, обработчик\_события).

По сути по сравнению с методом delegate в данном случае меняются лишь параметры селектора и события:

|  |  |
| --- | --- |
|  | $('body').on('click', 'button', function(e){      $(this).css('color', 'red');      alert('Обработчик для кнопки: '+$(this).text());  }); |

Для отмены регистрации обработчиков события можно использовать метод off:

|  |  |
| --- | --- |
|  | // удаление обработчиков события click для кнопок  $('body').off('click','button');  // удаление всех обработчиков для элементов внутри тега body  $('body').off(); |

**Метод trigger.**

Метод trigger используется для вызова обработчиков событий вручную. Например, вызвав метод trigger для нужного элемента можно также и вызвать обработчик события. Он имеет следующие варианты использования:

* trigger('событие'): вызов обработчика для данного события;
* trigger(объект\_Event): вызов обработчика с использованием объекта Event, который содержит данные о том, какой именно обработчик надо вызвать.

Посмотрим на примере. Допустим, есть таблица и два кнопки на форме, которые используют обработчики таблицы:

|  |  |
| --- | --- |
|  | <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>  <br>  <button>Выделить</button>  <button>Снять выделение</button>  <script type="text/javascript">  $(function(){      $('table').on('mouseenter', 'tr', function(e){          $(this).css('background-color', 'silver');      }).on('mouseout', 'tr', function(e){          $(this).css('background-color', 'white');      });        $('button').first().bind('click', function(){          $('tr').trigger('mouseenter');      });        $('button').last().bind('click', function(){          $('tr').trigger('mouseout');      });    });  </script>  </body> |

В примере определены обработчики для строк таблицы для событий mouseenter и mouseout и обработчики для двух кнопок. Поскольку первая кнопка предназначена для выделения строк таблицы, по ее нажатию вызывается обработчик события mouseenter, определенный для строк таблицы. А по нажатию второй кнопки – обработчик события mouseout.

**Специальные методы для обработки событий.**

Кроме методов bind/delegate/on jQuery предоставляет прямые методы для обработки событий. Эти методы, как правило, носят наименование обрабатываемого события, а в качестве параметра принимают функцию обработчика данного события:

* blur: событие потери фокуса элементом;
* change: событие изменение значение элемента;
* click: событие нажатия;
* dblclick: двойное нажатие;
* focus: получение фокуса элементом;
* focusin: получение фокуса элементом или одним из его дочерних элементов;
* focusout: потеря фокуса элементом или одним из его дочерних элементов;
* hover: наведение указателя мыши на элемент и выход за пределы элемента;
* keydown: событие нажатия клавиши клавиатуры;
* keyup: отжатие клавиши;
* mousedown: нажатие мыши на элемент;
* mouseup: отжатие кнопки мыши;
* mouseenter: наведение указателя мыши на элемент;
* mouseleave: выход указателя мыши за границы элемента;
* mousemove: перемещение указателя мыши в пределах элемента;
* mouseout: выход указателя мыши за границы элемента;
* mouseover: наведение указателя мыши на элемент;
* ready: загрузка структуры DOM элемента;
* resize: измение размеров окна браузера;
* scroll: событие прокрутки элемента;
* select: выделение текста;
* submit: отправка формы.

Принцип действий всех вышеперечисленных методов одинаков: если вызывается метод без параметра, то генерируется событие; а если передана в качестве параметра функция – то тем самым устанавливается обработчик для данного события. Например:

|  |  |
| --- | --- |
|  | <button>Кнопка 1</button>  <button>Кнопка 2</button>  <script type="text/javascript">  $(function(){      $('button').first().click(function(){          $(this).css('background-color', 'silver');      });        $('button').last().click(function(){          $('button').first().click();      });  });  </script> |

В данном случае для двух кнопок добавляется обработчики события click, то есть нажатия. Обработчик первой кнопки просто устанавливает цвет нажатой кнопки (которая получена с помощью выражения $(this)). Для второй кнопки также устанавливается обработчик в виде функции, передаваемой в метод click. Только теперь обработчик инициирует событие нажатия для первой кнопки: $('button').first().click();.

Таким образом, в зависимости от задачи и предпочтений можно использовать либо методы bind/delegate/on для регистрации обработчиков событий, либо вышеприведенные методы типа click или mouseenter.

1. **Введение в AJAX.**

Технология AJAX представляет собой технологию асинхронного взаимодействия с сервером. Благодаря данной технологии можно отправлять запросы к серверу и получать от него ответы без перезагрузки страницы в фоновом режиме, создавать насыщенные интерактивные веб-приложения, например, веб-чаты. Функционал асинхронных запросов в браузерах доступен веб-разработчикам через объект XMLHttpRequest.

Для работы с AJAX нужен веб-сервер. Это может быть Node.js, IIS, Apache, nginx. Создадим веб-страничку, где будет использоваться XMLHttpRequest:

|  |  |
| --- | --- |
|  | <html>      <head>          <meta charset='utf-8'>      </head>      <body>          <button onclick="ajaxload();">Загрузить</button>          <div id="news"><h3>Нет новостей</h3></div>          <script type="text/javascript">          function ajaxload(){              let xhr=new XMLHttpRequest();              xhr.onreadystatechange=function(){                  if(this.readyState==4){                      if(this.status >=200 && xhr.status < 300){                          document.getElementById('news').innerHTML=this.responseText;                      }                  }              }              xhr.open('GET', 'ajax.php');              xhr.send();          }          </script>      </body>  </html> |

На странице есть два элемента: кнопка, по нажатию на которую совершается функция ajaxload, и блок div, в который будут загружаться данные при запросе. В функции ajaxload создается объект XMLHttpRequest: let xhr=new XMLHttpRequest();.

Через этот объект будут отправляться запросы. При получении ответа будет срабатывать событие onreadystatechange. Для обработки данного события свойству xhr.onreadystatechange присваивается функция обработки ответа. В функции обработки проверяется на готовность ответа через свойство readyState (состояние this.readyState == 4 означает, что запрос завершен). Далее проверяется статусный код ответа: если сервер возвратил статусный код от 200 до 300, то запрос прошел успешно. Затем передается текст ответа в блок div, который имеется на странице через свойство responseText. В строке xhr.open('GET', 'ajax.php'); устанавливается метод запроса и ресурс, к которому будет идти запрос. В последней строчке xhr.send() отправляем запрос.

Допустим на сервере находится обработчик запросов на языке php, который передает в ответ разметку html:

|  |  |
| --- | --- |
|  | <?php      echo "<h3>ЦСКА победил в чемпионате России</h3><h5>30.05.2013</h5>";  ?> |

Теперь по нажатию по кнопке будет происходить ajax-запрос, результаты которого будут загружаться на страницу.

Перепишем код страницы с использованием jQuery:

|  |  |
| --- | --- |
|  | <html>      <head>          <meta charset='utf-8'>          <script src="jquery-1.10.1.min.js"></script>      </head>      <body>          <button>Загрузить</button>          <div id="news"><h3>Нет новостей</h3></div>          <script type="text/javascript">          $(function(){              $('button').click(function(){                  $('#news').load('ajax.php');              });          });          </script>      </body>  </html> |

Таким образом, код становится чище и проще за счет использования библиотеки jQuery. Достаточно применить метод load, передав в качестве параметра адрес ресурса. И итоговый результат будет тот же самый.

1. **Метод load.**

Метод load загружает код HTML, полученный с сервера, в элемент на странице. Он принимает следующие параметры:

* url: обязательный параметр, содержащий адрес ресурса, к которому будет обращаться запрос;
* data: необязательный параметр, содержащий простой объект javascript или строку, которые будут отправлены на сервер вместе с запросом;
* complete(responseText, textStatus, XMLHttpRequest: необязательный параметр – функция обратного вызова, которая будет выполняться при завершении запроса.

В предыдущем параграфе был показан простейший пример использования метода load:

|  |  |
| --- | --- |
|  | <button>Загрузить</button>  <div id="news"><h3>Нет новостей</h3></div>  <script type="text/javascript">  $(function(){      $('button').click(function(){          $('#news').load('ajax.php');      });  });  </script> |

Если второй параметр data не определен, то запрос выполняется как GET-запрос. Если определен – то осуществляется POST-запрос, а данные передаются как при отправке формы. И, применив соответствующие методы на серверной стороне, можно получить переданные данные. Изменим предыдущий пример:

|  |  |
| --- | --- |
|  | <button>Загрузить</button>  <div id="news"><h3>Нет новостей</h3></div>  <script type="text/javascript">  $(function(){      $('button').click(function(){          $('#news').load('ajax.php', {'event':'Начало чемпионата России', 'date':'13.07.2013'});      });  });  </script> |
|  |  |

Здесь в качестве второго параметра передается объект javascript, в котором определены два свойства: event и date. Теперь получим эти данные на сервере и вернем их в ответ клиенту:

|  |  |
| --- | --- |
|  | <?php      $event=$\_POST['event'];      $date=$\_POST['date'];      echo "<h3>" . $event . "</h3><h5>" . $date . "</h5>";  ?> |

Теперь, при нажатии на кнопку, данные, переданные во втором параметре, возвратятся обратно клиенту.

Применяя третий параметр – функцию обратного вызова complete, можно произвести некоторые дополнительные сопутствующие действия при завершении запроса. Самый простейший пример: вывод уведомления о состоянии запроса:

|  |  |
| --- | --- |
|  | $('#news').load('ajax.php', function(){alert('Запрос выполнен');}); |

1. **GET- и POST-запросы.**

Метод get осуществляет GET-запрос к серверу, то есть все данные запроса передаются в строке запроса. Он принимает следующие параметры:

* url: обязательный параметр, содержащий адрес ресурса, к которому будет обращаться запрос;
* data: необязательный параметр, содержащий простой объект javascript или строку, которые будут отправлены на сервер вместе с запросом;
* success(data, textStatus, jqXHR): необязательный параметр – функция обратного вызова, которая будет выполняться при успешном выполнении запроса. Она может принимать три параметра: data – данные, полученные с сервера, textStatus – статус запроса и jqXHR – специальный объект jQuery, который представляет расширенный вариант объекта XMLHttpRequest;
* dataType: необязательный параметр, содержащий тип данных в виде строки, например, "xml" или "json".

На выходе метод get возвращает объект jqXHR, который будет инкапсулировать данные запроса.

Перепишем пример из предыдущего параграфа с использованием метода get:

|  |  |
| --- | --- |
|  | $(function(){      $('button').click(function(){          $.get('ajax.php', function(data) {                  $('#news').html(data);                  alert('Данные заружены');          });      });  }); |

Здесь использованы два первых параметра: адрес ресурса и функция обратного вызова. В этой функции в качестве параметра data получаем принятые от сервера данные и загружаем их в качестве разметки в элемент выборки ($('#news').html(data);). То же самое можно сделать с помощью метода load.

Используя параметр data, можно отправить дополнительные данные. Например, можно отправить запрос на получение элемента из базы данных по id:

|  |  |
| --- | --- |
|  | $.get('ajax.php', { id: "1"}); |

А поскольку в GET-запросе все данные передаются в строке запроса, то данный код будет аналогичен следующему:

|  |  |
| --- | --- |
|  | $.get('ajax.php?id=1'); |

Соответственно на стороне сервера можно получить этот параметр и произвести с ним какие-либо действия, например, получить элемент из бд по данному id:

|  |  |
| --- | --- |
|  | <?php      $id=$\_GET['id'];  ?> |

Нередко возникает такая ситуация, когда сервер отправляет данные в определенном формате, например, json или xml. Так, php-файл на стороне сервера мог бы выглядеть следующим образом:

|  |  |
| --- | --- |
|  | <?php      echo json\_encode(array("event"=>"Начало чемпионата России","date"=>"13.07.2013"));  ?> |

В данном случае он возвращает объект json. Тогда на клиентской стороне можно явно прописать тип данных, а в функции обработать принятые данные:

|  |  |
| --- | --- |
|  | $.get('ajax.php',      function(data) {              $('#news').empty().append("<h3>"+data.event+"</h3><h5>"+data.date+"</h5>");      },      'json'); |

**Объект jqXHR.**

В функции обратного вызова может использоваться объект jqXHR. По сути с ним так или иначе связаны все методы jQuery, которые предназначены для отправки ajax-запросов. Данный объект представляет собой обертку над стандартным объектом javascript XMLHttpRequest. Для совместимости с XMLHttpRequest jqXHR предлагает нам следующие свойства и методы:

* readyState: возвращает состояние запроса. Имеет значения от 0 (запрос не отправлен) до 4 (запрос завершен);
* status: возвращает код статуса HTTP, который получает от сервера;
* statusText: возвращает статус запроса в виде текста;
* responseXML: возвращает ответ в виде XML;
* responseText: возвращает ответ в виде простого текста;
* setRequestHeader(name, value): устанавливает заголовок запроса name со значением value. Если подобный заголовок уже имеется, то его значение заменяется на value;
* getAllResponseHeaders(): возвращает все заголовки ответа;
* getResponseHeader(name): возвращает заголовок ответа name;
* abort(): прерывает запрос.

Но если для обработки ответа объект XMLHttpRequest требует обработать событие onreadystatechange, то для jqXHR определено сразу несколько методов, которые помогают обработать ответ:

* done(function(data, textStatus, jqXHR)): обрабатывает успешный ответ с сервера;
* fail(function(jqXHR, textStatus, errorThrown)): обрабатывает ответ от сервера, запрос к которому закончился ошибкой;
* always(function(data|jqXHR, textStatus, jqXHR|errorThrown)): обрабатывает как успешные ответы с сервера, так и сообщения с ошибкой;
* then(function(data, textStatus, jqXHR), function(jqXHR, textStatus, errorThrown)): включает функциональность методов done и fail, принимая соответственно две функции обработки ответа.

Итак, ранее использовался метод get следующим образом:

|  |  |
| --- | --- |
|  | $.get('ajax.php', function(data) {                  $('#news').html(data);                  alert('Данные заружены');          }); |

Но так как метод get возвращает объект jqXHR, то можно написать следующим образом, применив метод done:

|  |  |
| --- | --- |
|  | $.get('ajax.php').done(function(data) {                  $('#news').html(data);                  alert('Данные заружены');          }); |

Подобным образом можно применить и другие методы:

|  |  |
| --- | --- |
|  | let jqxhr=$.get('ajax.php');  jqxhr.done(function(data) {          $('#result').html(data);  });  jqxhr.fail(function(data) {          $('#result').html("ОШИБКА!");  }); |

**POST-запросы.**

В отличие от GET-запросов данные POST-запросов передаются не в строке запроса, а в его теле. Распространенным примеров подобных запросов является отправка данных формы на сервер. Для отправки POST-запросов предназначен метод post. Его объявление и использование в целом аналогично методу get. Он принимает следующие параметры:

* url: обязательный параметр, содержащий адрес ресурса, к которому будет обращаться запрос;
* data: необязательный параметр, содержащий простой объект javascript или строку, которые будут отправлены на сервер вместе с запросом;
* success(data, textStatus, jqXHR): необязательный параметр – функция обратного вызова, которая будет выполняться при успешном выполнении запроса. Она может принимать три параметра: data – данные, полученные с сервера, textStatus – статус запроса и jqXHR – специальный объект jQuery, который представляет расширенный вариант объекта XMLHttpRequest;
* dataType: необязательный параметр, содержащий тип данных в виде строки, например, "xml" или "json".

На выходе метод post возвращает объект jqXHR.

Пример использования:

|  |  |
| --- | --- |
|  | $.post('ajax.php', {'login':'1111', 'password' : '2222'},          function(data) {          $('#news').html(data);          }); |

В данном случае в качестве данных передается пароль и логин. На сервере можно получить данные и отправить ответ пользователю:

|  |  |
| --- | --- |
|  | <?php      $login=$\_POST['login'];      $pass=$\_POST['password'];      if($login=="1111" && $pass=="2222"){          echo "Авторизация прошла успешно";      }      else{          echo "Неверно введен логин или пароль";      }  ?> |

Поскольку наиболее часто запрос post используется при отправке данных формы, используем на стороне клиента форму:

|  |  |
| --- | --- |
|  | <html>      <head>          <meta charset='utf-8'>          <script src="jquery-1.10.1.min.js"></script>      </head>      <body>           <form action="ajax.php" id="loginForm">              <input type="text" id="login" placeholder="Логин" /><br><br>              <input type="text" id="password" placeholder="Пароль" /><br><br>              <input type="submit" value="Войти" />          </form>          <div id="result"></div>          <script type="text/javascript">          $("#loginForm").submit(function(event) {              // Предотвращаем обычную отправку формы              event.preventDefault();              $.post('ajax.php', {'login':$('#login').val(), 'password' : $('#password').val()},                      function(data) {                              $('#result').html(data);                      });              });          </script>      </body>  </html> |

Итак, серверная часть, к которой будет обращаться форма – файл ajax.php – остается той же. Только в данном случае теперь для параметра data в методе post данные берем из полей на этой форме. Обратите внимание, что обычная отправка формы блокируется (event.preventDefault();), иначе была бы переадресация.

Посколкьу нередко формы не ограничиваются двумя полями, то проще применять сериализацию формы. Сериализация производится посредством метода serialize и в качестве результата создает объект javascript, где свойствам соответствуют поля формы. И значения хранят эти свойства те же, что имеют соответственные поля формы.

Итак, применим сеиализацию формы:

|  |  |
| --- | --- |
|  | <form action="ajax.php" id="loginForm">      <input type="text" name="login" placeholder="Логин" /><br><br>      <input type="text" name="password" placeholder="Пароль" /><br><br>      <input type="submit" value="Войти" />  </form>  <div id="result"></div>  <script type="text/javascript">  $("#loginForm").submit(function(event) {      // Предотвращаем обычную отправку формы      event.preventDefault();      $.post('ajax.php', $("#loginForm").serialize(),              function(data) {                      $('#result').html(data);              });      });  </script> |

В отличие от предыдущего примера у нас здесь два отличия. Во-первых, поля для ввода имеют атрибут name. При указании параметра data осуществляется сериализация данных формы через метод serialize: $("#loginForm").serialize(). В данном методе в тело запроса передаются параметры. Причем имена параметров – это значения атрибутов name полей ввода. А значения параметров – соответственно введенные значения в текстовые поля. И поэтому с помощью php можно извлечь эти значения: $login=$\_POST['login'].

1. **Получение данных JSON.**

Метод get можно использовать для получения данных в формате json. Однако библиотека jQuery также предоставляет специальный метод getJSON. Этот метод отправляет на сервер GET-запрос и в ответ получает данные json.

Этот метод принимает следующие параметры:

* url: обязательный параметр, содержащий адрес ресурса, к которому будет обращаться запрос;
* data: необязательный параметр, содержащий простой объект javascript или строку, которые будут отправлены на сервер вместе с запросом;
* success(data, textStatus, jqXHR): необязательный параметр, представляющий функцию обратного вызова, которая будет выполняться при успешном выполнении запроса.

На выходе метод getJSON возвращает объект jqXHR, связанный с текущем запросом. Допустим, на сервере есть следующий файл countries.json, который содержит перечисление стран в формате json:

|  |  |
| --- | --- |
|  | {      "Великобритания": 1,      "Китай": 2,      "Россия": 3,      "США": 4,      "Франция": 5  } |

Добавим на страницу выпадающий список (элемент select) и загрузим в него данные из этого файла:

|  |  |
| --- | --- |
|  | <select id="countries">      <option disabled>Выберите страну</option>  </select>  <script type="text/javascript">  $(function(){      $.getJSON('countries.json', function(data) {                  $.each(data, function(key, val) {                      $('#countries').append('<option value="' + val + '">' + key + '</option>');                  });      });  });  </script> |

В качестве значения параметра url используется название файла json. Так как приходящие данные представляют собой набор, то можно перебрать их в цикле each и применить к ним функцию обработки. Обрабатывающая функция вызывается для каждого элемента в наборе и принимает два параметра: название данных или ключ (key) и значение (val). То есть в строке "Россия": 3 "Россия" – это ключ, а 3 – значение.

В данном случае структура данных довольно проста. Теперь посмотрим на использование данных с более сложной структурой. Например, определен такой файл users.json:

|  |  |
| --- | --- |
|  | {      "users": [{          "id" : 1,          "name": "Bill Gates",          "age": 43      }, {          "id" : 2,          "name": "Sergey Brin",          "age": 33      }, {          "id" : 3,          "name": "Larry Page",          "age": 34      }]  } |

Попробуем вывести его в табличку на странице:

|  |  |
| --- | --- |
|  | <table id="users">      <tr><td>Id</td><td>Имя</td><td>Возраст</td><tr>  </table>  <script type="text/javascript">  $(function(){      $.getJSON('users.json', function(data) {              for(let i=0;i<data.users.length;i++){                  $('#users').append('<tr><td>' + data.users[i].id + '</td><td>' + data.users[i].name +                  '</td><td>' + data.users[i].age + '</td><tr>');              }      });  });  </script> |

Используя названия свойств объектов json, можно получить к ним доступ (например, data.users[i].age) и тем самым вывести их значения на страницу.