

Лекция 4. jQuery. jQuery UI

- Это мощная библиотека, которая предоставляет:
 - Мощные средства для работы с DOM
 - Средства для работы с событиями
 - Упрощение работы с AJAX – асинхронными запросами на сервер
 - Promises
- И главное – всё это кросс-браузерно, т.е. работает даже в старых IE
- Хорошее знание данной библиотеки обязательно для веб-разработчика, потому что она используется в 99% проектов

Версии jQuery

- jQuery достаточно активно обновляется, текущая версия 3.x, но при этом из библиотеки была удалена поддержка старых IE
 - jQuery 1.x – старая версия, поддерживает IE 6-8
 - jQuery 3.x – новая версия, не поддерживает IE 6-8
- Выбирайте версию 3.x, если не нужно поддерживать старые IE. Эта версия более легковесная и быстрая, потому что убрано много кода, специфичного для старых браузеров
- Выбирайте версию 1.x если нужна поддержка IE 6-8

Подключение библиотек

- Есть несколько вариантов, как подключить к себе любую библиотеку
 - Просто скачать с сайта, положить в папку с проектом
 - Скачать при помощи менеджера пакетов (рассмотрим позже). Например, **npm** или **bower**
 - Сослаться на CDN - библиотека выложена на специальных публичных серверах
- Пока что воспользуемся первым вариантом, но обычно пользуются вторым

Минифицированная версия, source map

- Обычно, все библиотеки поставляются в двух версиях:
 - Debug (не минифицированная)
 - Min/Release/Dist (минифицированная, чтобы меньше занимать)
- По минифицированным версиям сложно отлаживаться, поэтому часто для разработки используют debug версию
- Для отладки минифицированных версий есть специальное решение – source map файлы
- <https://habrahabr.ru/post/148098/>

Демонстрация

- Скачиваем и подключаем jQuery 3.x

Переменная jQuery. noConflict

- Библиотека добавляет две глобальные переменные: jQuery и \$
- Обе они ссылаются на один и тот же объект, просто имя \$ короче, а jQuery – длинное, но полное имя
- Чаще всего пользуются коротким именем

\$.noConflict

- Некоторые другие JS библиотеки тоже используют \$ как имя своих переменных, поэтому бывает необходимость, чтобы jQuery не затирала переменную \$
- Для этого в jQuery есть функция \$.noConflict:
- ```
<script src="other_lib.js"></script> // определяет $
<script src="jquery.js"></script> // затирает $
<script>
 $.noConflict(); // возвращает значение $ из other_lib
</script>
```



# Document ready

- Со страницей нельзя безопасно работать, пока документ не будет «готов»
- jQuery умеет отслеживать этот момент
- ```
$(document).ready(function() {  
    // Код, работающий с DOM должен быть здесь  
    // или вызываться отсюда  
});
```
- Переменная `document` – глобальная. Она не относится к jQuery, это специальный объект, который обозначает текущий документ (страницу)
- Если код, работающий со страницей писать/вызывать не из `$(document).ready`, то он может работать неправильно

Document ready

- `$(document).ready(function() {
 // код
});`
- Для document ready есть ещё краткий синтаксис:
- `$(function() {
 // код
});`

Window load

- Кроме document ready есть еще событие window load
- `$(window).load(function() {
 // код
});`
- Это событие отрабатывает позже, после того, как вся страница загрузится (включая картинки и iframe'ы)
- Поэтому, обычно, пользуются document ready, потому что оно отрабатывает раньше, и уже сразу скрипт сможет начать делать что-то полезное

Выбор элементов страницы

Выбор элементов на странице

- jQuery предназначен, чтобы работать с элементами страницы
- Чтобы с ними работать, сначала нужно выбрать нужные элементы
- Для выбора jQuery поддерживает все CSS селекторы и несколько дополнительных селекторов
- ```
var links = $("a");
```

 // выбрать все ссылки  

```
var links = $(".class1 .class2");
```

  
// выбрать все элементы с классом class2, которые  
// внутри элементов с классом class1

# Дополнительные селекторы

- Выбор первого элемента  
`var links = $("a:first");`
- Выбор последнего элемента  
`var links = $("a:last");`
- Выбор всех input, select, textarea и button  
`var links = $(":input");`
- Выбор только видимых div элементов  
`var links = $("div:visible");`
- Выбор чекбоксов с проставленной галочкой  
`var links = $(":checked");`
- Выбор только нечетных ссылок  
`var links = $("a:odd");`

# Дополнительные селекторы

- Выбор только четных элементов списка  
`var links = $("li:even");`
- Выбор ссылок, исключая первые три  
`var links = $("a:gt(2)");`  
К этой группе еще относятся `eq`, `lt`
- Выбор только `enabled` элементов  
`var links = $(":enabled");`
- Выбор только `disabled` элементов  
`var links = $(":disabled");`
- Выбор только выбранных опций в `select`  
`var links = $(":selected");`
- И т.д.

# Отрицание :not

- Есть возможность сделать отрицание в селекторе
- `var links = $(":not(li)");` // выбрать все элементы не li
- `var links = $("input:not(:checked)");`  
// выбрать чекбоксы без галочек



# Фильтр :has

- В CSS отсутствуют селекторы, позволяющие выбирать родительские элементы, основываясь на их дочерних элементах
- В jQuery для этого можно пользоваться фильтром :has
- `$(“div:has(span)”)`
- Так будут выбраны все div, внутри которых есть span

# jQuery набор и элементы DOM

- Следует заметить, что результатом поиска по селектору будет объект-обертка jQuery
- Его еще называют **обернутым набором**
- `var links = $("a");`
- Получившаяся переменная `links` является специальным объектом-оберткой над набором нативных DOM-элементов
- Эта обертка предоставляет большое количество функций для работы с выбранными элементами – например, удаление, дальнейший поиск внутри элементов, изменение атрибутов и т.д.

# jQuery набор и элементы DOM

- `var links = $("a");`
- У обертки есть свойство `length`, показывающее, сколько элементов находится в наборе
- `var linksCount = $("a").length;`
- Из обертки можно получить ее DOM элементы, используя синтаксис массива:
- `var domLink = links[0];` // первый DOM элемент
- Из обертки можно получить обертку для конкретного элемента:  
`var link = links.eq(0);`

# Демонстрация

- Пытаемся выбрать элементы и задать CSS параметры в них при помощи jQuery

# Функция add

- Допустим, нам нужно выбрать все ссылки и все элементы списков
- Это можно сделать так:
- `var elements = $("a, li");`
- Либо так при помощи функции add
- `var elements = $("a").add("li");`
- В add можно передавать не только селектор, но и DOM элементы и обернутый набор

# Функция not

- Позволяет наоборот исключить часть элементов из набора
- `var elements = $("a").not(".link1");`
- В not можно передавать не только селектор, но и DOM элементы и обернутый набор

# Функция filter

- Позволяет отфильтровать набор при помощи функции
- ```
var elements = $("a").filter(function(index) {  
    // this будет ссылать на текущий DOM элемент  
    var self = $(this);  
    return self.attr("href").indexOf("jquery") >= 0;  
});
```

Дальнейшая навигация по набору

- Есть много функций для навигации по детям/потомкам/соседям:
- **find, children** – выбор потомков и детей
- **parent, parents** – выбор родителя, родителей
- **closest** - выбор ближайшего родителя, удовлетворяющего селектору
- **next, nextAll, prev, prevAll, siblings** – выбор соседей

Создание, удаление и перемещение элементов

Создание новых HTML-элементов

- Кроме выбора и работы с существующими элементами страницы, jQuery позволяет создавать новые элементы и удалять существующие
- `var el = $("`
- Переменная `el` будет ссылаться на jQuery обертку, готовую быть добавленной на страницу
- Добавим ее на страницу:
- `el.appendTo(document.body);`
`// appendTo добавляет элементы набора в конец`
`// указанного элемента`

Chaining

- Общий принцип библиотеки jQuery – chaining (цепочки)
- При работе с обертками, команды к ним можно составлять в цепочки
- ```
$(“<div></div>”)
 .addClass(“test”)
 .css({ width: “100%” })
 .click(function() {
 alert(“OK”);
 })
 .appendTo(document.body)
```
- Это возможно за счет того, что все эти функции jQuery также выдают обертку-набор

Создали div, дали ему класс test, задали ширину 100% inline-стилем, навесили обработчик клика и добавили в конец документа

# Добавление элементов в другой

- Допустим, у нас уже есть некоторый div, мы хотим добавить внутрь него другой div
- Тогда можно воспользоваться функциями append и prepend
- `var el = $("<div>Text</div>");`  
`$("#myDiv").append(el);`  
*// вставляет переданный элемент как последнего ребенка*
- `var el = $("<div>Text</div>");`  
`$("#myDiv").prepend(el);`  
*// вставляет переданный элемент как первого ребенка*
- При этом, если целевой элемент – один, а добавляемый элемент уже на странице, то он просто перемещается

# Другие полезные функции

- Есть много и других полезных функций для вставки и перемещения элементов
- **before, after, insertBefore, insertAfter**
- Есть метод wrap, который оборачивает указанные элементы указанным тегом
- `$("#a").wrap("<div class='test'></div>");`
- ```
<div class="test">  
  <a href="/index.html">Ссылка</a>  
</div>
```

Еще полезные методы

- Метод `remove` удаляет элементы:
- `$("#a").remove();`
- Метод `detach` выдергивает элементы из DOM, но их можно будет переместить и вставить в другое место:
- `$("#a").detach().appendTo(document.body);`
- Метод `clone` создает копии элементов набора:
- `$("#a").clone().appendTo(document.body);`

Манипулирование свойствами элементов

Getters/setters

- Многие jQuery функции для работы с элементами могут работать как геттеры и как сеттеры, в зависимости от числа аргументов
- Например, функция `attr`, которая работает с атрибутами элементов
- Как сеттер – функция требует 2 аргумента – имя атрибута и новое значение
- `var links = $("a").attr("target", "_blank");`
- Как геттер – требует 1 аргумент, имя атрибута:
- `var target = $("a").attr("target");`

Getters/setters

- Сеттеры устанавливают значение всем элементам в наборе
- `var links = $("a").attr("target", "_blank");`
- Геттер выдает значение только из первого элемента в наборе:
- `var target = $("a").attr("target");`
- Если набор был пустой, то результатом будет `undefined`
- Эти принципы схожи для очень многих jQuery функций, поэтому его нужно запомнить:
- Сеттер меняет все элементы набора, а геттер получает значение из первого элемента набора

removeAttr

- Есть функция для удаления атрибута у всех элементов набора
- `$("a").removeAttr("target");`

Получение/задание innerHtml

- `<div>`
 `Ссылка`
 `</div>`
- Геттер:
- `var html = $("div").html();`
 `// Ссылка`
- Сеттер:
- `var target = $("div").html("");`

Получение/задание text

- `<div>`
 `Ссылка`
 `</div>`
- Геттер:
- `var html = $("a").text();`
 `// Ссылка`
- Сеттер:
- `var target = $("a").text("Новая ссылка");`

Добавление/удаление классов

- Добавление CSS классов элементам:
- `$("#a").addClass("class1");`
`$("#a").addClass("class1 class2");` // несколько классов
- Удаление CSS классов:
- `$("#a").removeClass("class1 class2");`
- Добавить/убрать класс в зависимости от условия:
- `$("#a").toggleClass("visible", $("#a").is(":visible"));`
- Работают над всем набором

Скрытие/показ элементов

- `$(“a”).hide();` // скрыть элементы
`$(“a”).show();` // сделать элементы видимыми
- Показать/скрыть в зависимости от условия:
- `$(“a”).toggle(needToggle);`
// `needToggle` – какая-то наша `boolean` переменная
- Эти функции также применяются ко всему набору

Анимация

- jQuery также предоставляет средства анимации
- По сути анимация – плавное изменение CSS свойств элементов
- Например, изменение прозрачности или размеров
- Демонстрация

Проход по элементам

- Часто нужно пройти по всем элементам набора и выполнить некоторый код
- Обычно для этого пользовались циклами
- Но рекомендуется использовать функцию `each`, которая делает то же самое
- ```
$(“buttons”).each(function(i) {
 $(this).click(function() {
 alert(i);
 });
});
```
- Если из функции вернуть `false`, то это аналогично `break` в цикле



# Работы с формами

# Получение значения элементов

- Чтобы получить или задать значения элементов вроде `input[type='text']`, `textarea`, `select` используется метод `val`:
- `<input type="text" id="name" />`
- `var name = $("#name").val();`  
`$("#name").val("Новый текст");`
- **Важно**, что этот метод **нельзя** применять к чекбоксам и радио-баттонам – он просто вернет значение их атрибута `value`, независимо от того, выбраны флажки или нет

# Получение значения чекбокса

- `<input type="checkbox" id="isMarried" />`
- `var isMarried = $("#isMarried").is(":checked");`  
`$("#isMarried").attr("checked", "checked");`
- Другие варианты:
- `$("#isMarried").prop("checked", true);`  
`$("#isMarried").attr("checked", true);`

# Получение значения радио

- `<input type="radio" value="male" name="gender"/>`  
`<input type="radio" value="female" name="gender"/>`
- `var gender = $("[name='gender']:checked").val();`
- `$("[name='gender'][value='male']")  
 .attr("checked", "checked");`

# Enable/disable

- Для элементов форм часто нужна логика по их активации/деактивации (enable/disable)
- Для этого есть несколько вариантов:
- ```
$(“input”).prop(“disabled”, true);  
$(“input”).attr(“disabled”, “disabled”);  
$(“input”).attr(“disabled”, true);
```
- Для снятия disable:
- ```
$(“input”).prop(“disabled”, false);
$(“input”).removeAttr(“disabled”);
$(“input”).attr(“disabled”, false);
```

**События**

# События

- Графические интерфейсы, в том числе и в браузере, часто основаны на событиях
1. Ждем, пока в интерфейсе что-то произойдет (нажмут кнопку или введут текст и т.д.)
  2. Реагируем на событие, возвращаемся на п.1
- Чтобы иметь возможность реагировать на события, на них нужно подписаться – зарегистрировать функцию-обработчик конкретного события на конкретном элементе

# Старая модель событий

- События в разных браузерах сильно отличаются, это еще одно место, где jQuery берет на себя обеспечение кросс-браузерности
- Навесить обработчики можно через html атрибуты и в JS
- `<button onclick="myFunction()">Кнопка</button>`  
`// myFunction объявлена где-то в нашем скрипте`
- Обработчиком становится не сама функция myFunction, а вот такая функция:
- ```
function(event) {  
    myFunction();  
}
```

Обработчик оборачивается в анонимную функцию и туда передается аргумент event с информацией о событии

Объект event в JS

- В большинстве браузеров туда передается объект Event
- Но в IE он не передается, а просто доступен через `window.event`
- Поэтому в каждом обработчике приходилось писать:

```
if (!event) {  
    event = window.event;  
}
```
- Но и тут проблемы не кончаются:
- Чтобы получить элемент, на котором произошло событие, во всех браузерах нужно обратиться `e.target`
- А в IE – `e.srcElement`

Всплытие событий (bubbling)

- Когда в элементе произошло событие, то механизм обработки событий браузера ищет обработчик в самом элементе и вызывает его
- Но на этом не все
- Далее механизм ищет обработчик этого же события у родителя и вызывает его и т.д.
- Это называется **всплытием события (bubbling)**, т.к. событие распространяется снизу вверх – от потомка к родителям

Остановка всплытия событий

- В некоторых случаях всплытие нужно остановить
- В браузерах, соответствующим стандартам, нужно вызвать метод:
 - `e.stopPropagation();`
- В IE же нужно сделать так:
 - `e.cancelBubble = true;`

Действие по умолчанию

- У некоторых событий есть еще **действие по умолчанию**
- Например, при клике по ссылке `<a>` - действием по умолчанию является переход по адресу
- Или при нажатии на кнопку `submit`, происходит отправка формы
- Есть возможность отменить действие по умолчанию
- Если нужно отменить действие по умолчанию, то надо вернуть `false` из обработчика события

Появление новой модели событий

- Описанный подход устарел
- Одна из его проблем, что можно навесить только один обработчик
- А что, если нужно несколько?
- Поэтому в итоге пришли к шаблону Observer – когда все, кто хочет, подписываются и отписываются от событий

Добавление обработчика

- Теперь обработчики задаются в JS:
- ```
var button = $("button")[0];
button.addEventListener("click", function(e) {
 // код
}, false);
```

  

```
// третий аргумент указывает к какой фазе
// относится событие – см. след. слайд
```

# Новое распространение событий

- Теперь распространение событий делается в два этапа:
  1. Фаза захвата: Сначала оно распространяется сверху вниз, от элемента `html` до целевого элемента
  2. Фаза всплытия: распространяется от целевого элемента к корню – `html`
- В старой модели была только фаза всплытия, обработчик добавляется туда, если третьим аргументом передать `false`
- В новой модели, если передать `true`, обработчик вызовется на фазе захвата. Это редко используемая, но мощная возможность

# Добавление jQuery обработчика

- jQuery предоставляет кросс-браузерное решение для работы с событиями
- Устанавливаются обработчики при помощи функции on:
- ```
$(“button”).on(“click”, function(e) {  
    // обработчик  
});
```
- Для многих функций есть shortcut’ы
- ```
$(“button”).click(function(e) {
 // аналогично
});
```



# Удаление jQuery обработчика

- Чтобы удалить конкретный обработчик, его нужно сначала запомнить в переменную
- `var handler = function(e) {  
 // обработчик  
};  
$("button").on("click", handler); // добавили`
- `$("button").off("click", handler); // убрали`
- Можно убрать все обработчики события:
- `$("button").off("click");`
- Или убрать вообще все обработчики всех событий:
- `$("button").off();`

# Событие change

- Для элементов форм очень полезное событие change
- Оно срабатывает когда пользователь поменяет данные в элементе. Например, текст в поле ввода
- Но оно срабатывает не сразу после нажатия клавиши, а после того, как элемент теряет фокус ввода
- ```
$(".firstName").change(function() {  
    alert("Новое имя = " + $(this).val());  
});
```
- Для немедленных событий пригодятся события клавиатуры – keydown, keyup и др.

Метод trigger

- Можно заставить обработчик выполняться даже если по факту события не произошло
- Например, у нас на странице есть фильтр для данных, который срабатывает по нажатию кнопки
- И мы захотели программно применить фильтр, а не после действий пользователя
- Для этого можно просто заставить вызваться обработчик клика на кнопку
- `$(“button”).trigger(“click”);`
- Причем trigger имитирует и действие по умолчанию, и всплытие события

Метод `triggerHandler`

- Этот метод используется, когда нужно вызвать только обработчик, без действия по умолчанию и без всплытия
- `$(“button”).triggerHandler(“click”);`

Shortcuts для trigger

- Есть краткая версия для метода trigger для многих событий
- Аналогично:
- `$(“button”).trigger(“click”);`
`$(“button”).click();`

Объект jQuery event

- Event в jQuery обработчиках немного отличается
- Полезные методы:
 - `e.preventDefault()` – отменить событие по умолчанию
 - `e.stopPropagation()` – отменить всплытие
 - `e.stopImmediatePropagation()` – отменить всплытие события + отменить выполнение последующих обработчиков на этом элементе
- Если из обработчика события вернуть `false`, то это аналогично вызову `preventDefault` и `stopPropagation`



Что такое AJAX

- AJAX (Asynchronous JavaScript and XML) – средства для отправки запросов на сервер из JS без перезагрузки страницы
- Эта технология в нынешнее время общеупотребима на всех более менее серьезных сайтах и веб-приложениях
- AJAX поддерживается браузерами при помощи специального объекта XHR (XmlHttpRequest)

Объект XHR

- Как обычно, во всех браузерах и старых IE этот объект нужно создавать по-разному
- ```
var xhr;
if (window.ActiveXObject) {
 xhr = new ActiveXObject("Microsoft.XMLHTTP");
} else if (window.XMLHttpRequest) {
 xhr = new XMLHttpRequest();
} else {
 throw new Error("Ajax not supported");
}
```
- jQuery же дает единый подход для работы с AJAX

# JSON

- Несмотря на слово XML в аббревиатуре AJAX, на самом деле могут передаваться любые данные – например, просто текст, JSON и др.
- Рассмотрим формат JSON, который стал очень популярен при использовании с AJAX
- По сути JSON (JavaScript Object Notation) – это просто передача данных в виде JavaScript объектов и массивов
- Примеры:
- `[1, 2, 3]`
- `{"response":{"items":[],"totalCount":0}}`

# JSON

- Как видно, это обычные литералы объектов, массивов, чисел, `boolean`, строк, `null`
- Обычно, для краткости, данные в формате JSON передаются минифицированными – то есть без пробелов и переводов строк
- `{"response":{"items":[],"totalCount":0}}`
- Кроме того, в JSON обязательны двойные кавычки вокруг имен свойств
- Главное достоинство JSON – это полная интеграция с JavaScript. Данные в формате JSON являются корректными JavaScript данными

# Функции для работы с JSON

- В современных браузерах есть две полезные функции для работы с JSON:
  - `JSON.parse(str)` – превращает строку в формате JSON в JavaScript данные
  - `JSON.stringify(obj)` – превращает данные JS в JSON строку
- Примеры:
  - `var obj = JSON.parse('{\"name\":\"Ivan\"}');`  
`console.log(obj.name);` // Ivan
  - `var str = JSON.stringify([1, 2, 3]);`  
// [1, 2, 3]

# JSON в старых браузерах

- В старых браузерах, которые это не поддерживают, подключают специальную библиотеку, например JSON2:
- <https://github.com/douglascrockford/JSON-js/blob/master/json2.js>
- Они добавляют поддержку функций `JSON.stringify` и `JSON.parse`

# Команды GET и POST

- Получать данные с сервера через AJAX можно при помощи HTTP методов GET, либо POST
- У этих методов разная семантика:
  - Запросы GET не должны модифицировать состояние системы. Обычно используются для получения данных с сервера
  - Запросы POST могут модифицировать состояние системы. Обычно используются для внесения данных на сервер, например, для добавления, изменения или удаления данных с сервера
- Важный момент, что GET запросы могут кэшироваться, а POST - нет

# Использование \$.get

- Для отправки GET запроса есть функция \$.get
- \$.get(url, params, callback, type);
  - url – адрес ресурса
  - params – параметры для передачи
  - callback – функция, вызываемая, когда с сервера придет успешный ответ
  - type – необязательный параметр, не будем рассматривать

# Использование \$.get

- `$.get(url, params, callback, type);`
- Загрузили массив телефонов, создали div'ы с ними на странице
- ```
$.get("/getPhones", { page: 1, size: 10}, function(phones) {  
    $.each(phones, function(i, phone) {  
        $("

").text(phone).appendTo(document.body);  
    });  
});


```
- Важно, что запросы выполняются асинхронно – код продолжает исполняться, а callback вызовется только когда придет ответ сервера

Использование \$.post

- \$.post(url, params, callback, type);
- Все аналогично, только делается POST запрос
- \$.post("/deletePhone", { number: "1234"}, function(result) {
 if (result) {
 alert("Номер успешно удален");
 }
});

Метод \$.ajax

- Команды \$.get и \$.post – это всего лишь краткие варианты вызова команды \$.ajax, где некоторые параметры уже заранее заданы
- Но есть возможность полностью управлять отправкой и получением данных
- Выглядит так:
- \$.ajax(options);
- Опций очень много, смотрите документацию

Callback'и \$.ajax

- AJAX запросы не всегда завершаются успешно – на сервере может что-то упасть
- Поэтому недостаточно иметь только обработчик на успешное завершение запроса
- Часто нужен и обработчик на ошибку, чтобы пользователю, например, вывелось сообщение, что произошла ошибка, попробуйте позже
- И еще часто бывает нужен обработчик, который выполняется всегда после запроса, независимо от того успешно он завершился или нет
- Например, при начале загрузки мы показываем «крутилку» - индикатор загрузки, а после запроса мы должны ее убрать

Callback'и \$.ajax

- Для этого в команду \$.ajax можно передать дополнительные callback'и
- `showLoading();` // наша функция, включает «крутилку»
`$.ajax({`
 `url: “/getData”,`
 `success: function(response, textStatus, xhr) {`
 // выполнится если метод отработает успешно
 `},`
 `error: function(xhr, textStatus, errorThrown) {`
 // вызовется в случае ошибки
 `},`
 `complete: function(xhr, textStatus) {`
 `stopLoading();` // выключаем «крутилку»
 `}`
`});`

Promise

- Команды `$.get`, `$.post` и `$.ajax` в качестве результата выдают объект `XHR`, который использовался для данного запроса
- В `jQuery` этот объект имеет несколько особенностей:
 1. У него есть метод `abort()`, который позволяет прервать запрос. Это полезно, если, например, пользователь нажал кнопку, началась загрузка данных, но пользователь захотел ее отменить
 2. Этот объект является `promise`'ом, см. следующий слайд

Promise

- Задавать callback'и в AJAX бывает неудобно
- Допустим, мы хотим добавить несколько success callback'ов
- Или несколько обработчиков ошибок
- Это легко сделать, если пользоваться xhr как promise'ом
- Promise – это такой объект, к которому можно зарегистрировать обработчики на успех, неудачи и которые выполнятся всегда

Promise

- ```
showLoading(); // наша функция, включает «крутилку»
var xhr = $.ajax({ url: "/getData" })
 .done(function(response, textStatus, xhr) {
 // выполнится если метод отработает успешно
 })
 .fail(function(xhr, textStatus, errorThrown) {
 // вызовется в случае ошибки
 })
 .always(function(xhr, textStatus) {
 stopLoading(); // выключаем «крутилку»
 });
```
- Этот синтаксис более удобный и можно добавлять много обработчиков

**Утилитные  
функции**



# Утилитные функции

- В jQuery также есть несколько функций, которые не связаны с DOM элементами
- Они вызываются через точку от \$
- Обрезка пробелов в начале и конце строки:
- `var result = $.trim(" 123 "); // 123`
- Эта функция очень полезна при обработке данных форм, чтобы обрезать лишние пробелы

# Проверка типов

- `$.isFunction(arg)` – выдает `true`, если переданный аргумент является функцией
- `$.isArray(obj)` – выдает `true`, если переданный аргумент является массивом
- `$.isNumeric(obj)` – проверка, что число
- И т.д.

# Копирование свойств

- `$.extend([deep ], target [, object1] [, objectN])`
- Первый аргумент означает глубокое копирование (рекурсивное)
- Объект, переданный первым, получает свойства объектов, переданных дальше
- Часто используется, чтобы дополнить набор настроек по умолчанию переданными пользователем значениями
- Также часто используется для копирования объектов
- `var copy = $.extend(true, {}, someObj);`  
`// глубокая копия объект someObj`

# Проход по массиву и свойствам: each

- Проход по массиву – проходит по элементам массива, для каждого вызывает указанную функцию
- `$.each(array, function(index, value) {  
 // this будет тоже ссылаться на value  
});`
- Если в функции сделать `return false`; то итерирование прервется (аналог `break` в циклах)
- Есть версия для прохода по полям объекта:
- `$.each(obj, function(propName, propValue) {  
});`
- Each рекомендуется вместо циклов, потому что меньше возможности допустить ошибку

# Преобразование - map

- Преобразование – проходит по элементам массива, для каждого вызывает указанную функцию преобразования
- ```
var result = $.map([1, 2, 3, 4], function(value, index) {  
    // заметим, что тут обратный порядок аргументов  
    // по сравнению с each  
    return value * 2;  
});  
// [2, 4, 6, 8]
```
- Если из функции вернуть `null` или `undefined`, то текущий элемент не попадет в результат
- Если функция вернет массив, то в результат попадет не он сам, а все его элементы

Преобразование - map

- Есть версия и для полей объекта
- ```
var obj = { width: 10, height: 20 };
var result = $.map(obj, function(value, propName) {
 // заметим, что тут обратный порядок аргументов
 // по сравнению с each
 return value * 2;
});
// [20, 40]
```
- Результатом будет являться массив из значений
- Эта версия используется редко
- Map также рекомендуется вместо циклов

# Фильтрация - grep

- Фильтрует массив по указанному предикату-функции
- ```
var result = $.grep([1, 2, 3], function(value, index) {  
    return value % 2 === 0;  
});  
// [2]
```

Поиск в массиве - inArray

- Ищет указанное значение в массиве и выдает индекс первого вхождения
- Выдает -1, если ничего не нашлось
- `var index1 = $.inArray(2, [1, 2, 3]); // 1`
`var index2 = $.inArray(4, [1, 2, 3]); // -1`
- Еще можно указать индекс, начиная с которого нужно искать

jQuery пример PhoneBook

- Пример – телефонная книга

jQuery UI

- Библиотека jQuery UI предоставляет следующие возможности:
 - Набор виджетов (элементов управления), которых нет в обычном HTML – календарь, слайдер, autocomplete, диалоги, табы, кнопки и т.д.
 - Дополнительные анимации и эффекты
 - Возможности сортировки, drag'n'drop, изменения размеров, возможность выбора элементов
 - Мощные возможности позиционирования элементов
 - Фабрика виджетов – возможность создавать свои виджеты в стиле jQuery UI

Достоинства jQuery UI

- Достоинства этой библиотеки:
 - Полная бесплатность
 - Контролы в принципе неплохи, но не так, чтобы шикарно
 - Малое количество багов
 - Хорошая документация
 - Популярность – в интернете можно найти много материалов и решений сложных задач
 - Кросс-браузерность

Недостатки jQuery UI

- Но в целом данная библиотека не является каким-то массовым стандартом
- Недостатки:
 - Есть не все нужные виджеты
 - Многие виджеты написаны плохо и имеют утечки памяти
 - Зависимость от jQuery
 - Не всегда легко стилизовать виджеты под требуемый вид

jQuery UI - итого

- Библиотека очень популярна, используется во многих проектах, но не во всех
- Полезно иметь с ней хотя бы некоторое знакомство, но глубоко углубляться не стоит
- Из самых полезных виджетов выделяю:
 - Календарь – тут он неплох, но с утечками памяти
 - Autocomplete
 - Диалог – jQuery UI позволяет вызывать диалог поверх диалога, это редкая фишка в подобных библиотеках

Демонстрация jQuery UI

- Демонстрация
 - Сайт jQuery UI
 - Подключение к проекту
 - Использование виджета

Задачи и материалы

Задача на курс «PhoneBook»

- См. файл

Задача Gabriels

- Теперь, когда мы знаем JS, можно реализовать custom scroll и слайдер
- В качестве слайдера возьмите jQuery UI Slider
- В качестве скролла – найдите и примените любой jQuery плагин сами
- Кстати, слайдер в этом задании необычный. Он должен быть с неравномерным шагом
- Допустимые значения – от 0 до 1000
- От 0 до 100 шаг 10, а дальше – шаг 100
- И не допускаются значения, не кратные 10. Если такое введут, то они должны подправляться скриптом

Материалы для изучения

- <http://learn.jquery.com/>
- <https://jquery.com/>
- <https://jqueryui.com/>
- Бер Бибо, Иегуда Кац.
jQuery. Подробное руководство
по продвинутому JavaScript
(2-е издание, 2012)
- Книга хорошо читается, рекомендую.
Но она по старой версии, так что обязательно смотрите и
документацию

