jQuery plugin

<https://plugins.jquery.com/>

## jQuery

Last week we introduced jQuery and it’s value proposition as a the “swiss army knife” of the DOM, giving us everything we need to create DHTML (ie: selecting elements, watching for / responding to events & updating the DOM). We finished off the lecture by discussing how we can make AJAX requests using jQuery, ie:

$.ajax({

url: "http://localhost:8081/employees",

type: "GET",

contentType: "application/json"

})

.done(**function** (employees) {

console.log(employees)

})

.fail(**function** (err) {

console.log("error: " **+** err.statusText);

});

Today, we will be rounding out our discussion of jQuery by discussing some of jQuery’s **Utility** methods, as well as how to handle **form data** (ie: reading / writing form information). Lastly, we will have a quick discussion on working with **Bootstrap 3 Components** using jQuery:

## Utility Methods

There are a number of [Utility functions / methods](http://api.jquery.com/category/utilities/) included in jQuery (35+). We will only discuss some of the ones that we’re likely to see in this course. However, all of the methods available are useful and we encourage you to **Bookmark** the full list for future reference.

### jQuery.data()

Store arbitrary data associated with the specified element and/or return the value that was set.

See: <http://api.jquery.com/jQuery.data/>

**var** div **=** $( "div" )[ 0 ];

jQuery.data( div, "test", {

first: 16,

last: "pizza!"

});

jQuery.data( div, "test" ).first; *// => 16*

jQuery.data( div, "test" ).last; *// => "pizza!"*

### jQuery.removeData()

Remove a previously-stored piece of data.

See: http://api.jquery.com/jQuery.removeData/

jQuery.removeData( div, "test" );

jQuery.data( div, "test" ).first; *// => TypeError: Cannot read property 'first' of undefined*

### jQuery.contains()

Check to see if a DOM element is a descendant of another DOM element.

See:<http://api.jquery.com/jQuery.contains/>

$.contains( $("ul")[0], $("li")[0] ); *// true*

$.contains( $("li")[0], $("ul")[0] ); *// false*

### jQuery.grep()

Finds the elements of an array which satisfy a filter function. The original array is not affected.

See:<http://api.jquery.com/jQuery.grep/>

**let** arr **=** $.grep(employees, **function**( employee ) {

**return** employee.Position.PositionName **==** "Full Stack Developer";

}); *// => 35 "employee" objects returned*

### jQuery.map()

Translate all items in an array or object to new array of items.

See: <http://api.jquery.com/jQuery.map/>

$.map(employees, **function**( val, i ) {

val.LastName **=** val.LastName **+** "!";

}); *// => transform all last names to end in "!"*

### jQuery.parseHTML()

Parses a string into an array of DOM nodes.

See: <http://api.jquery.com/jQuery.parseHTML/>

**let** htmlArr **=** $.parseHTML( "hello, <b>my name is</b> jQuery." ); *// => 3 elements: 'hello, ', <b> and ' jQuery.'*

## Handling Form Data

jQuery has a very nice interface for handling form values. The following sections illustrate how we can **get**, **set**and correctly **clear** the values of **text** / **password**, **checkbox**, **radiobutton**, **textarea** and **select (single / multiple)** elements.

If you’re following along with the “Code Samples” the following functionality is located in the **“week2/jQuery”**folder

### text / password elements

<input type="text" class="form-control" id="inputEmail" placeholder="Email">

<input type="password" class="form-control" id="inputPassword" placeholder="Password">

Set

$("#inputEmail").val("from jQuery"); *// set to "from jQuery"*

$("#inputPassword").val("abc"); *// set to "abc"*

Get

$("#inputEmail").val();

$("#inputPassword").val();

Clear

$("#inputEmail").val("");

$("#inputPassword").val("");

### checkbox elements

<input type="checkbox" id="checkbox1"> Checkbox

Set

$("#checkbox1").prop("checked", **true**); *// set to "checked"*

Get

$("#checkbox1").prop("checked"); *// true / false*

Clear

$("#checkbox1").prop("checked", **false**);

### radiobutton elements

<input type="radio" name="optionsRadios" id="optionsRadios1" value="option1"> Option One

<input type="radio" name="optionsRadios" id="optionsRadios2" value="option2"> Option Two

Set

$('input[name=optionsRadios][value=' **+** 'option2' **+** ']').prop("checked", **true** ); *// set "option 2" to checked*

Get

$('input[name=optionsRadios]:checked').val(); *// get the value of the checked "optionsRadio"*

Clear

$('input[name=optionsRadios]').prop("checked", **false**);

### textarea elements

<textarea class="form-control" rows="3" id="textarea1"></textarea>

Set

$("#textarea1").val("from jQuery"); *// set to "from jQuery"*

Get

$("#textarea1").val();

Clear

$("#textarea1").val("");

### select (single / multiple) elements

<select class="form-control" id="select1">

<option value="">Please Select</option>

<option value="1">One</option>

<option value="2">Two</option>

<option value="3">Three</option>

<option value="4">Four</option>

<option value="5">Five</option>

</select>

<select multiple="" class="form-control" id="select2">

<option value="1">One</option>

<option value="2">Two</option>

<option value="3">Three</option>

<option value="4">Four</option>

<option value="5">Five</option>

</select>

Set

$("#select1").val(3); *// set to "Three"*

$("#select2").val([4,5]); *// set to "Four" and "Five"*

Get

$("#select1").val();

$("#select2").val();

Clear

$("#select1").val("");

$("#select2").val("");

## jQuery & Bootstrap

The Bootstrap UI framework has become so instrumental in the construction of modern, responsive web apps; largety due to it’s execellent design patterns, modern tooling and wealth of online resources and templates. However, we have only really used it’s CSS (and minimal) JavaScript capabilities. In order to unlock Bootstrap’s full potential as a UI/UX framework, is to familiarize ourselves with the interactive UI components and the jQuery API used to invoke/manipulate them.

As we have seen, we don’t necessairly need to touch any JS code to make use of some of the interactive components. The bootstrap framework uses jQuery in an unobtrusive 눈에 거슬리지 않는 way, by utilizing 활용 ‘**data-**’ attributes, ie:

<button type="button" class="btn btn-primary btn-lg" data-toggle="modal" data-target="#myModal">

Launch demo modal

</button>

However, what if we wish to launch a modal window at a different (unknown) time? For example; when an AJAX request completes? This is where the [Programmatic API](https://getbootstrap.com/docs/3.3/javascript/#js-programmatic-api) comes in to play. It gives us more power and flexability to work with the compoments.

Using the API we can interact with all of Bootstrap’s JavaScript components programmatically (from JavaScript). The following examples illustrate how we can use the API to take advantage of some of Bootstraps most useful components:

### Modal Window

Arguably one of the most important elements in modern UI/UX design is the “modal” window. We can use it to display error/success warnings or focus the user’s attention on a specific task.

See: <https://getbootstrap.com/docs/3.3/javascript/#modals>

$('#myModal').modal({ *// show the modal programmatically*

backdrop: 'static', *// disable clicking on the backdrop to close*

keyboard: **false** *// disable using the keyboard to close*

});

### Button States

Although, technically deprecated, the ‘loading’ and ‘reset’ button states are simple way to programmatically enable/disable a button while at the same time, changing it’s text. This is useful in cases where clicking the button makes an AJAX request and we wish to disable the button until the request has completed.

The ‘toggle’ state is not deprecated and provides a quick way to show a button as ‘pressed’.

See: <https://getbootstrap.com/docs/3.3/javascript/#buttons-stateful> and also: <https://getbootstrap.com/docs/3.3/javascript/#buttons-methods>

$("#loadingDelay").button('loading'); *// switch to a disabled 'loading' state*

$("#loadingDelay").button('reset'); *// reset the button state*

$("#loadingDelay").button('toggle'); *// toggles 'push' state*

### Popovers

Popovers are like a tooltip, however they have their own title bar, and are capable of rendering HTML content. If we wish to use these components, we must ‘opt-in’ by using the following “popover” method in JavaScript. The full list of options provided can be found here: <https://getbootstrap.com/docs/3.3/javascript/#popovers-options>

See: <https://getbootstrap.com/docs/3.3/javascript/#popovers-usage>

$("#dynamicPopover").popover({

template: '<div class="popover" role="tooltip"><div class="arrow"></div>' **+**

'<h3 class="popover-title"></h3><div class="popover-content"></div></div>',

title: 'Tooltip Title',

content: 'Lorem ipsum dolor sit amet',

placement: 'top',

trigger: 'click'

});

### Alerts

Alerts are small messages that take the form of “**success**”, “**information**”, “**warning**”, or “**danger**”. The template for creating alerts is fairly consistant, ie:

**let** alertTemplate **=** '<div class="alert alert-warning alert-dismissible fade in" role="alert">' **+**

'<button type="button" class="close" data-dismiss="alert" aria-label="Close">' **+**

'<span aria-hidden="true">×</span>' **+**

'</button>' **+**

'<strong>Warning</strong>' **+** alertMessage **+**

'</div>';

We can use this to create alert messages on the fly and .append() them to our page whenever we wish to alert the user.

Unlike the modal window, these small messages don’t block the UI and don’t demand any action from the user, except to expictly close them. We can however, get around this limitation by using the .alert(‘close’) method after a certain amount of time:

setTimeout(**function**(){

$warningAlert.alert('close');

},3000);

See: <https://getbootstrap.com/docs/3.3/javascript/#alerts-methods>

© 2017 - Seneca School of ICT