jQuery

Landscape, Margin all 0.1 except bottom 2.5, font, 14

jQuery function: jQuery() OR $()

Document Ready - run when page is fully loaded

* **$**(*function()* { ... }**);** SAME AS $(**document**).**ready**( *function()* { ... } );

Event Handling

* $(“…”)**.on**(“**click/change/mouseover/mouseout/keydown/load/submit**”[, selector], function([*event*]) { $(this).text(); })
* 'this' gives you the reference of the html element of receiving the current event. **$(this)** returns a jQuery wrapper.
* An *event* object is optionally passed as a parameter.
* **[selector]** is required for selecting the **dynamically added** DOM elements

Selector Extensions - <https://api.jquery.com/category/selectors/jquery-selector-extensions/>

* Filter
  + **:contains**(text) // matches elements that contain the given text
  + **:empty** // elements that HAVE NO children (including text nodes)
  + :has(selector) // has the specified element
  + :parent // elements that ARE PARENTS. i.e they have child elements including text nodes
  + :is(selector) // true if found matches among the wrapped set.
  + :not(selector) // a regular filter
  + .not(filter selector/DOM element/array of DOM element) // this is a method filter
  + :header // elements that ARE HEADERS. e.g. h1, h2, h3 ...
  + :animated // elements that are currently being animated
  + **:hidden** // elements that are hidden
  + **:visible** // elements that are visible
  + :selected
* low level filter method for both selector style and programmatic filtering
  + .filter(selector/fn) // alternatively pass a function that returns true if to include, false if to exclude.

Action

Class

* .**removeClass**(‘classname’);
* .**addClass**(‘classname1 classname2 classname3’); // you can add 1+ class names
* .**toggleClass**(‘classname’); // whenever you need to addClass/removeClass on the same classname
* .**hasClass**(‘classname’);

Prop

* **.prop(name); / .prop(name, value);** /.**removeProp**(name);

Adding New HTML Element using jQuery:

* **$(‘<someNewElement > </someNewElement>’).**addClass(‘someClass’);
  + jQuery turns the Html element into a DOM fragment and Select it and return a jQuery object so you can use someAction.
* **Siblings:** $(‘<someNewElement > </someNewElement>’).**insertAfter/Before**(‘#someId’);
* **Child: .prependTo/AppendTo**(‘#someId’);
* Note: .insertAfter/Before, .prependTo/AppendTo will get called 1+ times when you have 1+ selectors

Remove Element

* .**remove**(‘someSelector’);

Content – Get /Set

* .**html**(); / .**html**(“someString”); //including the **tags**
* .**text**(); / .**text**(“someString”); // the **combined** text **without tags**

Value – Get/Set

* **.val()** // get the **input** value of the **1st** matched element
* **.val(val)** // set the input value of **ALL** matched element, including check/select type=radio, type=checkbox & select options

Checkbox & Radio Buttons – [checked]

* **Is Checked?:** $('#checkbox/radiobutton1')**.prop**("**checked**");
* **Checkbox:** $('#checkbox')**.prop**("**checked**", **true/false**);
* **Radiobutton:** $('#radiobutton1')**.prop**("**checked**", **true**); // will **auto unchecked ALL other radio buttons** with the SAME **‘name’** attrib.

Disable Controls – [disabled]

* **Is Disabled?:** .**prop**(“**disabled**”); // test is disabled
* .**prop**(“**disabled**”, **true/false**); // Set disabled or not

Width & Height – get, set

* **width**(), width(value) / **height**(), height(value)

Clone

* **Clone from Template with incremented Id**
  + $('#hiddenTemplate li')**.clone().appendTo**('#myUL')**.prop("id",** "myLI" + cloneIndex).text('Item ' + cloneIndex);

Iteration

* + **$.each(array**, function() { });
  + **$(‘selector returns 1+ elements’).each**(function() { })

[Manipulation](http://docs.jquery.com/Manipulation)

* Wrap - wrap the elements of the matched set with the passed HTML tags or clone of the passed element
  + wrap(open & close html tags/an element to be cloned) // e.g. $('a').wrap('<div></div>') OR $('a').wrap($('div:first')[0])
  + wrapAll / wrapInner
* Replacing
  + replaceWith(***content***)
  + replaceAll(***selector***)
* Remove
  + empty() // remove of **ALL CHILD NODES** of the wrapped set
  + remove([expr]) // remove **ALL ELEMENTS** of the wrapped set.
* Copy - clone(bool) // clone all matched elements including their event handlers if bool=true. returns the newly created wrapped set

Common

* .**hide**(), .**show**() // should **use CSS Class** instead.
* .is(“someSelector”) // to test if the current Element match “someSelector”. e.g. .is(‘:visible’)
* .**toggle**(); // element is toggle between hide and show

[Traversing](http://docs.jquery.com/Traversing)- too much to document here

* **slice**(begin, **end but\_not\_including**) // returns a new subset of **index-based** portion
* relationship - children, contents, next, nextAll, parent, parents, prev, prevAll, siblings
* find(selector) // returns a new wrapped set

Ajax – [stackoverflow1](http://stackoverflow.com/a/8840315/1274961), [stackoverflow2](http://stackoverflow.com/a/15694010/1274961)

* Stub: <https://jsonstub.com>
* **$.ajax({ settings }).done** Requests – [Book Async Javascript](http://1drv.ms/1QAyl0k), [Article](http://1drv.ms/1QsCeqj), [Article](http://1drv.ms/1QsC8z2) 
  + **Pattern**: $.ajax / **.always / .fail / .done / .then**
  + function xhr\_send(url, type) {

**return** **$.ajax**({

**url**: url,

**type**: type,

**contentType**: 'application/json',

**data**: $('#myForm').**serializeFormJSON()** , // data send to server

**// *get*** – **data:** $('#myform')**.serialize(), // e.g.** “key1=value1&key2=value2”

**// *post*** – **data:** **JSON.stringify**($('#myform')**.serializeFormJSON()**), **// e.g.** { key1: 'value1', key2: 'value2' }

**dataType**: **'json'**,

beforeSend: showLoadingImgFn

})

.**always**(function(obja, textStatus, objb) { }) // **return deferred**

.**fail**(function(jqXHR,textStatus,errorThrown) { }); // **return deferred**

}

// **Using** it

$('#submit').on('**click**', function (**event**) {

**event.preventDefault();**

xhr\_send('/api/getitems', 'POST').**done**(function(result,textStatus,jqXHR) {

// Appending <tr> to table

**$.each(**result.**data**, function (i, o) {

$('#resultTbl > tbody**:last-child**').**append**("<tr><td>" + o.Id + "</td><td>" + o.Title + "</td></tr>");

});

}); // return deferred

}

* **Optionally Chain** additional requests using **.then**
  + - $.ajax()/deferred.**then**(function(data,textStatus,jqXHR){}, function(jqXHR,textStatus,errorThrown){})
      * .**then** provides **chaining callbacks with the previous return value** as the argument of the next callback and it **returns a Promise**.
      * .**done** just **attachs more callbacks** which always use the **1st return value** as the argument of all the subsequent callbacks. **Returns Deferred.**
  + **When – when all happens**
    - $.**when**(**deferred1, deferred2, …**).done(function() { }); // accept 1+ deferred child objects, resolved ‘whenAll’ children resolved, rejected ‘whenAny’ children rejected
* **settings** json **data** property
  + **simple**: { name: "John", time: "2pm"}
  + **array**: { 'choices[]': ["Jon", "Susan"] }
  + **Serialize Form input** fields as **Query String**
    - $("#testform")**.serialize()**  // create encoded query string from form elements. e.g. &key1=val1&key2=val2
  + **Serialize Form input fields** as **JSON object** with **[name] attrib** as the **Key**, **[value] attrib** as the **Value** - **serializeFormJSON**

$.fn.**serializeFormJSON** = function () {

var o = {};

var a = this.serializeArray();

$.each(a, function () {

if (o[this.name]) {

if (!o[this.name].push) {

o[this.name] = [o[this.name]];

}

o[this.name].push(this.value || '');

} else {

o[this.name] = this.value || '';

}

});

return o;

};

* **Spinner** [**link**](http://jsfiddle.net/AndrewDryga/GY6LC/)
  + .spinner {

display: inline-block;

opacity: 0;

max-width: 0;

-webkit-transition: opacity 0.25s, max-width 0.45s;

-moz-transition: opacity 0.25s, max-width 0.45s;

-o-transition: opacity 0.25s, max-width 0.45s;

transition: opacity 0.25s, max-width 0.45s;

}

.has-spinner.active {

cursor:progress;

}

.has-spinner.active .spinner {

opacity: 1;

max-width: 50px; /\* More than it will ever come, notice that this affects on animation duration \*/

}

* + <button class="has-spinner"> <span class="spinner"><i class="icon-spin icon-refresh"></i></span> Foo </button>
  + $(this).toggleClass('active');
* **Global ajax event**: attach to the document node: [link](http://snippets.aktagon.com/snippets/204-How-to-display-an-animated-icon-during-Ajax-request-processing)
  + - * $(document).**ajaxError**(function(event, request, **settings**) {

$(this).html("Error requesting page " + **settings**.url + "!");

});

* + - * $(document).**ajaxStart/Send: add Global Spinner \\ tips: use $.active to track the request counter**
      * $(document).**ajaxStop/Complete: remove Global Spinner \\ tips: use $.active to track the request counter**
  + **Returned promise**
  + $(‘…’).**live**(‘**eventname’**, **eventCallback**); // add a handler for all current & future element. Use .**die(‘eventname’, eventCallback)** to unbind the event.

Utilities - [here](http://api.jquery.com/category/utilities/)

* $.**trim**(str)
* $.param(obj)
* $.isArray(obj)
* $.isFunction(obj)
* $.**length** // property. Prefer this to .size()
* $.parseJSON(‘{“key1”:”value1”,“key2”:”value2”}’); // parse string in JSON format
* $.get(index)

Chaining

$("#someId")

**.end()** // to back up the wrapped set to a **previously returned wrapped set** and return it.

**.andSelf()** // **merges 2 previous wrapped sets**.

This syntax **chaining** works because **most jQuery functions return a matched set** as a result. For example, .addClass, .css both return the original matches they acted on. Other functions like .not and .filter are actually **filter functions** – they modify the original matched set and **return a new filtered matched set** instead. There’s also an **.end() function that is used to remove any filters and return to the original matched set** specified at the beginning of the chain. Chaining is a great way to keep code compact, but it’s optional. Chained statements can sometimes be difficult to debug. Not all jQuery functions can are chainable. Some operations result in non-matched sets. For example functions like .val(), .text(), .html() return string values, .width() and .height() return numbers, .position() returns a position object.

Adding new HTML into DOM tree

* there are a number of situations where cloning is very useful. For example, you can use cloning() for a sort of templating mechanism.

var row = $("#gdEntries>tbody>tr:nth-child(2)").clone();

row.find("td:first")  
   .text(new Date().formatDate("MMM dd - hh:mmt"));

row.find("td:nth-child(2)")  
   .html("<b>" + new Date().formatDate("hh:mm:ss") +   
   "</b><br />More text second item here");

row.find("td:last")  
   .text( 199.99.formatNumber("c"));

row.insertBefore("#gdEntries>tbody>tr:nth-child(2)")  
   .show();

* The nice thing with this approach is that you don’t have to create HTML on the client – you’re only injecting data and maybe minimal markup rather than HTML strings because you are cloning an existing node and only ‘filling in the blanks’.