jQuery NOTES

Appending inside a loop

You just received a big array of data. Now it's time to loop through and render it on the page. Your first thought may be to do something like this:

```
var i; // <- the current item number
var count = data.length; // <- the total
var row; // <- for holding a reference to our row object
// Loop over the array
for ( i = 0; i < count; ++i ) {
row = data[ i ];
// Put the whole row into your table
$('#my-table').append(
$('<tr></tr>').append(
$('<td></td>').html(row.type),
$('<td></td>').html(row.content)
)
);
}
```

This is *perfectly valid* and will render exactly what you'd expect, but...

DO NOT do this.

Remember those **300+** rows of data?

Each one will force the browser to re-calculate every element's width, height and positioning values, along with any other styles - unless they are separated by a layout boundary, which unfortunately for this example (as they are descendants of a **<table>** element), they cannot.

At small amounts and few columns, this performance penalty will certainly be negligible. But we want every millisecond to count.

Better options

1. **Add to a separate array, append after loop completes**

```
/**
* Repeated DOM traversal (following the tree of elements down until you reach
* what you're looking for - like our <table>) should also be avoided wherever possible.
*/
// Keep the table cached in a variable then use it until you think it's been removed
var $myTable = $('#my-table');
// To hold our new <tr> jQuery objects
var rowElements = [];
var count = data.length;
var i;
var row;
// Loop over the array
for ( i = 0; i < count; ++i ) {
rowElements.push(
$('<tr></tr>').append(
$('<td></td>').html(row.type),
$('<td></td>').html(row.content)
)
);
}
```

```
// Finally, insert ALL rows at once
$myTable.append(rowElements);
```
Out of these options, this one relies on jQuery the most.

2. **Using modern Array.* methods**

```
var $myTable = $('#my-table');
// Looping with the .map() method
// - This will give us a brand new array based on the result of our callback function
var rowElements = data.map(function ( row ) {
// Create a row
var $row = $('<tr></tr>');
// Create the columns
var $type = $('<td></td>').html(row.type);
var $content = $('<td></td>').html(row.content);
// Add the columns to the row
$row.append($type, $content);
// Add to the newly-generated array
return $row;
});
// Finally, put ALL of the rows into your table
$myTable.append(rowElements);
```
Functionally equivalent to the one before it, only easier to read.

3. **Using strings of HTML (instead of jQuery built-in methods)**

```
// ...
var rowElements = data.map(function ( row ) {
var rowHTML = '<tr><td>';
rowHTML += row.type;
rowHTML += '</td><td>';
rowHTML += row.content;
rowHTML += '</td></tr>';
return rowHTML;
});
// Using .join('') here combines all the separate strings into one
$myTable.append(rowElements.join(''));
```
Perfectly valid but again, **not recommended**. This forces jQuery to parse a very large amount of text at once and is
not necessary. jQuery is very good at what it does when used correctly.

4. **Manually create elements, append to document fragment**

```
var $myTable = $(document.getElementById('my-table'));
/**
* Create a document fragment to hold our columns
* - after appending this to each row, it empties itself
* so we can re-use it in the next iteration.
*/
var colFragment = document.createDocumentFragment();
/**
* Loop over the array using .reduce() this time.
* We get a nice, tidy output without any side-effects.
* - In this example, the result will be a
* document fragment holding all the <tr> elements.
*/
```

```
var rowFragment = data.reduce(function ( fragment, row ) {
// Create a row
var rowEl = document.createElement('tr');
// Create the columns and the inner text nodes
var typeEl = document.createElement('td');
var typeText = document.createTextNode(row.type);
typeEl.appendChild(typeText);
var contentEl = document.createElement('td');
var contentText = document.createTextNode(row.content);
contentEl.appendChild(contentText);
// Add the columns to the column fragment
// - this would be useful if columns were iterated over separately
// but in this example it's just for show and tell.
colFragment.appendChild(typeEl);
colFragment.appendChild(contentEl);
rowEl.appendChild(colFragment);
// Add rowEl to fragment - this acts as a temporary buffer to
// accumulate multiple DOM nodes before bulk insertion
fragment.appendChild(rowEl);
```



```
return fragment;
}, document.createDocumentFragment());
// Now dump the whole fragment into your table
$myTable.append(rowFragment);
```

My personal favorite. This illustrates a general idea of what jQuery does at a lower level.

Dive deeper

jQuery source viewer
Array.prototype.join()
Array.prototype.map()
Array.prototype.reduce()
document.createDocumentFragment()
document.createTextNode()
Google Web Fundamentals - Performance

jQuery append

HTML

```
<p>This is a nice </p>
<p>I like </p>
<ul>
<li>List item 1</li>
<li>List item 2</li>
<li>List item 3</li>
</ul>
<button id="btn-1">Append text</button>
<button id="btn-2">Append list item</button>
```

Script

```
$("#btn-1").click(function(){
$("p").append(" <b>Book</b>.");
});
$("#btn-2").click(function(){
$("ul").append("<li>Appended list item</li>");
});
});
```

Appending an element to a container

Solution 1:

```
$('#parent').append($('#child'));
```

Solution 2:

```
$('#child').appendTo($('#parent'));
```

Both solutions are appending the element #child (adding at the end) to the element #parent.

Before:

```
<div id="parent">
<span>other content</span>
</div>
<div id="child">
</div>
```

After:

```
<div id="parent">
<span>other content</span>
<div id="child">
</div>
</div>
```

Note: When you append content that already exsists in the document, this content will be removed from its
original parent container and appended to the new parent container. So you can't use .append() or .appendTo() to
clone an element. If you need a clone use .clone() -> [http://api.jquery.com/clone/][1]