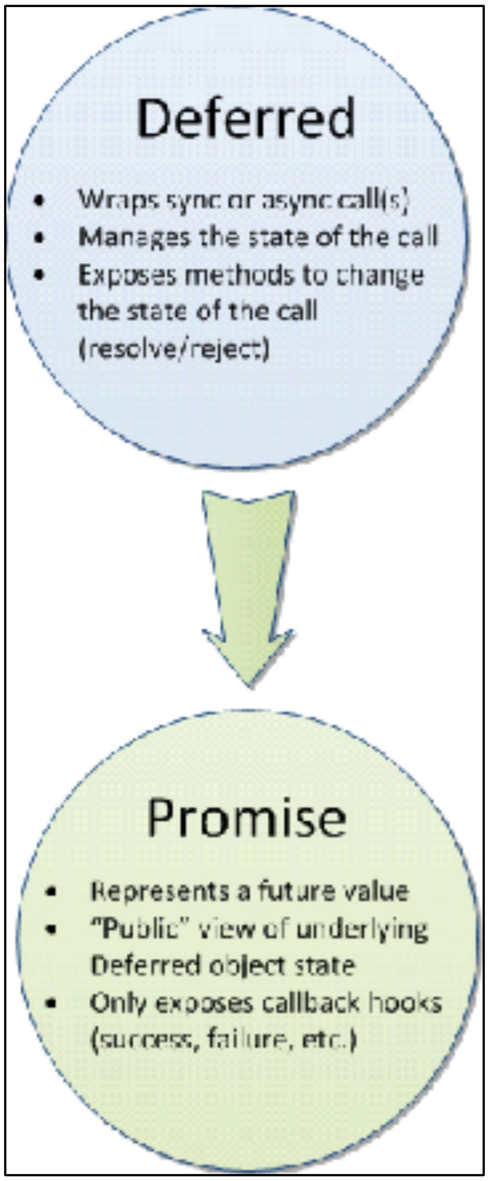
jQuery 201

Deferred and Promise

The deferred object in jQuery is a really great way to deal with synchronous and asynchronous actions with a true pub/sub implementation rather than a single callback. Deferred and promise are important to understand as Ajax returns an object that implements the Promise interface, giving it many of the same methods.


Deferred
Wraps sync or async call(s)
Manages the state of the call
Exposes methods to change the state of the call (resolve/reject)
Promise
Represents a future value
"Public" view of underlying Deferred object state
Only exposes callback hooks (success, failure, etc.)

The general idea is that the Deferred wraps up the state async call, and has the right to *resolve()* the call or *reject()* the call. A deferred object represents work that is not yet finished. Every deferred also has a promise method.

The Promise is the “public” view of the state of the Deferred. It **won’t** allow you to resolve or reject its state, but it will allow you to attach callbacks that are invoked when the state of the Deferred changes. A promise object represents a value that is not yet known.

Both deferred and promise objects own several methods for attaching handlers that get executed as a deferred object’s state changes:

.done() - Executed when the deferred is resolved successfully

.fail() - Executed when the deferred is rejected, great for error handling

.always() - Executed after done or fail, always. Use this for housekeeping functions like removing an ajax loader gif.

.then() - Allows chaining of asynchronous events by returning a new promise object. Same as $.pipe()

http://api.jquery.com/category/deferred-object/

AJAX [http://api.jquery.com/jQuery.ajax/]

$.ajax is a nicely wrapped up API for creating asynchronous calls to the server side. Ajax is essential for calling server side APIs of all return types: json, jsonp, html, xml etc. jQuery itself doesn’t do anything specific to create an AJAX request otherwise known as XHR (XMLHttpRequest), but it does make the creating of said XHR consistent across all browsers.

jQuery	Native
`$.ajax({`	`var r = new XMLHttpRequest();`
`      url: "webservice",`	`r.open("POST", "webservice", true);`
`      type: "POST",`	`r.onreadystatechange = function () {`
`      data: {a:1, b:2, c:3}`	`      if (r.readyState != 4 ||`
`});`	`r.status != 200) return;`
	`      console.log(r.responseText);`
	`};`
	`r.send("a=1&b=2&c=3");`

There are many simplified methods to create XHR objects in jQuery, all of which wrap up

$.ajax()

$.get()
$.getJSON()
$.post()

All of these XHR creation methods return a jqXHR object, which is a promise object. We can either use the callbacks within the $.ajax method or we can use the deferred methods for checking failure and success.

Request Types
The primary or most-commonly-used HTTP verbs (or methods, as they are properly called) are POST, GET, PUT, and DELETE. These correspond to create, read, update, and delete (or CRUD) operations. CRUD is used to separate concerns when interacting with an API.

GET (Read)
The simplest explanation is that a GET request uses the querystring to send data to the server/API. Example: http://api.google.com/search/blah . Browsers impose a query string limit so the amount of data you can pass is limited.

http://www.w3schools.com/tags/ref_httpmethods.asp

POST (Create)
The simplest explanation is that POST submits data to the server through a request packet called post data. Allows for much larger transmission of data.
http://www.w3schools.com/tags/ref_httpmethods.asp

PUT (Update)
Transmit information to the server and tell the server to update or insert it somewhere. The server will determine if it exists, if it does it is expected (under CRUD) to update the existing item, or will insert the data into storage.

DELETE (Delete)
Tells the server to delete something. Generally an ID is all that is passed and the request looks very similar to a GET request.

AJAX Response Data Types
There are a few common AJAX response formats that people use to deliver data. Most of these depend on the type of API, the amount of information returned, the intended use with the data, and the structure of the data on the server side. Providing jQuery with an expected response type helps it determine how to deliver the returned response.

JSON

“JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate…JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.”

source: http://www.json.org/

JSON is pretty much the defacto standard for creating APIs or AJAX responses. It’s easy to use, read, and work with in JavaScript.

JSONP
JSONP or "JSON with padding" is a communication technique used in JavaScript programs running in web browsers to request data from a server in a different domain, something prohibited by typical web browsers because of the same-origin policy.

source: http://en.wikipedia.org/wiki/JSONP

JSONP is necessary until Cross-origin resource sharing (CORS) becomes standard in all browsers. JSONP basically wraps a JSON response in a function call, the name of the function is usually passed to the API via querystring parameter ?callback. The biggest downside to JSONP APIs is that they must be GET requests
XML

- XML stands for EXtensible Markup Language
- XML is a markup language much like HTML
- XML was designed to carry data, not to display data
- XML tags are not predefined. You must define your own tags
- XML is designed to be self-descriptive
- XML is a W3C Recommendation

source: http://www.w3schools.com/xml/xml_whatis.asp

HTML
HTML is used as a response type when you want to return a large amount of html to the page without having to format it on the client side. This is useful if you want consistent html creation/rendering on the client and server without the need to duplicate code. A lot of the times you’ll see HTML return inside of a JSON object so that other data points can be passed back, such as success boolean, status code integer, and more.

$.noConflict()

```
var aQuery = $.noConflict(true);
console.log(aQuery(‘div.arrow’)); // [div.arrow, div.arrow]
console.log(jQuery); // undefined
console.log($); // undefined
```

jQuery.noConflict() allows you to relinquish control of the $ and jQuery variable names (if true is passed to the function). This allows you to rename the jQuery library and/or have 2 instances. This can be very useful for integrating your application into an unknown environment.

jQuery and Real-World Organization

```
var QuoteModule = function(el) {
  this.$container = $(el);
  this.refreshDataUrl = “someapi.mysite.com”;
};

QuoteModule.prototype.init = function() {
  this.bindEvents();
};

QuoteModule.prototype.bindEvents = function() {
  this.$container.on(‘click’, ‘.refresh’, $.proxy(this.refreshData,
this));
};

QuoteModule.prototype.refreshData = function() {
  $.ajax({
    url: this.refreshDataUrl,
    method: “GET”,
    dataType: “json”,
    data: {
      symbol: “msft”
    }
  }).done($.proxy(this._refreshDataSuccess, this));
};

QuoteModule.prototype._refreshDataSuccess = function(data) {
  if (data.success){
    //update data points!
    this.$container.find(“.last-price”).html(data.lastPrice);
  }
};

// Elsewhere in the application…
var mainQuoteModule = new QuoteModule($(“#main-quote-module”));
mainQuoteModule.init();

var secondaryQuoteModule = new QuoteModule($(“#secondary-quote-
module”));
secondaryQuoteModule.init();
```

jQuery is very powerful and helpful, but like any code it can become a mess if not thoughtfully organized. We should typically organize our UI centric client-side code (almost always jQuery based) into prototypical objects. We’ll pass a jQuery object as a parameter to the constructor of this “class”, which represents the containing element of the UI module in question. All events and functionality can then be bound to that specific module. Here are some examples:

Objectives

- How do I handle a successful or failed ajax call?
- When should I use different request types?
- What response data type should I be using most of the time?[1]

[1] Techtonic Group LLC - Proprietary and Confidential