OrchestraCMS Service Requests as Promises

This small library provides a promised-based alternative API to OrchestraCMS's doServiceRequest() method. Instead of providing a single callback function, it uses the promise programming model to handle the request's response.

doServiceRequestAsPromise(), doServiceRequestAsDeferred()

`doServiceRequestAsPromise()` and `doServiceRequestAsDeferred()` are promise-based analogs for `doServiceRequest()`. They take *similar but not identical*, parameters and they immediately invoke the request. Because they return a promise, unlike `doServiceRequest()`, their parameters do **not** include a callback function.

`doServiceRequestAsDeferred()` returns a *jQuery promise*. Use this method if the solution must run on Internet Explorer. If the solution supports only modern browsers, including Edge, using `doServiceRequestAsPromise()` is likely preferred. jQuery promises are similar, but not equivalent, to ECMAScript promises and have different methods and syntax.

```
 // p will be a Promise based on a request against the stg_BlogSupport service class with the action 'getMostRecent'
 var p = doServiceRequestAsPromise({
             service: 'stg_BlogSupport',           // an Apex class that implements cms.ServiceInterface
             action: 'getMostRecent'
         });

// now, do subsequent Promise handling of p as required
p.then(handleRecentBlogResponse);

Promise.all([p, q]).then(handlerForWhenBothPAndQAreFulfilled);


// of course, there may be no need to save the returned Promise in a variable; just handle it as required
doServiceRequestAsPromise({    // returns a promise whose request required DML to modify data
        action: 'postLiked',
        userId: CMS.userid,
        postId: $post.data('contentId')
    },
    'ReadWrite'          // true or any string with 'write' in it allows the request to update Salesforce records
 ).then(handleLikingConfirmation)
 .catch(handleError);
```

doServiceRequestAs…(params, write)

@param params

- JavaScript object whose properties must include provide the default values for subsequent calls to the promise-delivering returned function
    - the property "service" must be provided

        - this must be the name of an Apex class that implements cms.ServiceInterface
    - the string property “action” must be provided
    - additional properties as required by the Execute Request method in the service class

@param write

- defaults to false (DML write operations *not* allowed)
- to enable DML write operations, this parameter must be `true` or, for self-documenting code, a string that contains “write” (in any letter case)

@return

- a Promise or jQuery Deferred’s promise object, depending on which doServiceRequestAs… function was called

createServiceRequestAsPromise(), createServiceRequestAsDeferred()

The functions `createServiceRequestAsPromise()` and `createServiceRequestAsDeferred()` *return a function* that the developer uses as a simplified API to the specified service interface actions. This function may subsequently be invoked with fewer parameters than would otherwise be necessary because the creation of the function included default parameter values that don’t have to be specified again. For example,

```
var myRequest = createServiceRequestAsPromise({
        service: 'stg_BlogSupport',                // an Apex class that implements cms.ServiceInterface
        action: 'getMostRecent'
    });

 // for simplicity, these code examples just get a valid Promise object: p; they do not show how to process it

 var p = myRequest();         // returns a Promise based on a request against the stg_BlogSupport service class.
                              // with the action 'getMostRecent'

 var p = myRequest({category: 'Fitt's Law'}); // returns a Promise for getMostRecent with an additional category param

 var p = myRequest({action: 'getCount'});     // returns a Promise for stg_BlogSupport's getCount action
 var p = myRequest('getCount');               // simplified syntax producing identical results as the above example

 var p = myRequest({                          // returns a promise to a different action that requires DML to modify d
          action: 'postLiked',
          userId: CMS.userid,
          postId: $post.data('contentId')
     },
     'ReadWrite'          // true or any string with 'write' in it allows the request to update Salesforce records
 );


 // get a promise-generating function that communicates with the stg_NewsLoader service class
 var myOtherRequest = createServiceRequestAsPromise('stg_NewsLoader'),

 // returns a Promise based on a request to the stg_NewsLoader service class with the action 'digitalTransformation'
     p = myOtherRequest('digitalTransformation');
```

createServiceRequestAs…(defaultParams, defaultWrite)

@param defaultParams

- JavaScript object whose properties provide the default values for subsequent calls to the promise-delivering returned function
  - typically, the property 'service' will be provided
    - this must be the name of an Apex class that implements cms.ServiceInterface
  - additional properties, such as "action" may also be provided (if a default "action" is not provided here, it must be provided when invoking the function returned by this call)
- alternatively, if defaultParams is passed as a string, it is taken as the name of the service class (with no need to wrap it in as the service property of a JS object)

@param defaultWrite

- defaults to false (DML write operations *not* allowed)
- to enable DML write operations, this parameter must be `true` or, for self-documenting code, a string that contains "write" (in any letter case)

@return

- function _doServiceRequest(params, write)
  - initiates the service request using parameters that extend or replace the default parameters given to the `createServiceRequestAs…` call
    - if params is a string, it is used as the “action” property for the service request
    - returns a Promise or jQuery Deferred’s promise object, depending on which createServiceRequestAs… function was called

Resolving a Service Request Promise

If the service request’s response was valid JSON it will be deserialized into a JavaScript object (unlike `doServiceRequest` which does *not* do JSON parsing). This request response (which may now be a JavaScript object) is used to *resolve the promise*, which means that the response will be the single parameter passed to “then” methods attached to the promise. For example,

```
myRequest('getFeaturedProduct').then(function (response) {
    console.log('The product “%s” is now on sale for $%d', response.name, response.price);
});
/* where response is of the form:
    {
        name: 'Transistor Radio with Morse Code Pad',
        price: 24.99,
        …
    }
*/
```