Branch: master code-institute-stream-1-project / sessions / notes.js

Find file Copy path

LeDanielH moved files to sessions folder bcde236 29 days ago

1 contributor

236 lines (193 sloc) 5.6 KB Raw Blame History

```
1   //promisejs.org
2
3   function waitForMyVar(returnValue){
4           delete window.myVar;
5           var timer = setInterval(function(){
6                   if(window.myVar !== undefined){
7                           clearInterval(timer);
8                           returnValue(window.myVar);
9                   }
10          }, 100);
11
12          return window.myVar;
13  }
14
15  function callMeWhenYouHaveTheValue(value){
16          alert('the value is: '+ value);
17  }
18
19  waitForMyVar(callMeWhenYouHaveTheValue);
20
21  //XHR
22  //https://stream-1-project-ledanielh.c9users.io/#/gigs
23  function getData(url, returnValue){
24          var request = new XMLHttpRequest();
25
26          request.open('GET', url);
27
28          request.onreadystatechange = function(){
29                  if(request.readyState === 4){
30                          //end
31                          returnValue(request.responseText);
32                  }
33          };
34
35          //start
36          request.send(null);
37  }
38
39  function getDataSync(url){
40          var request = new XMLHttpRequest();
41
42          request.open('GET', url, false);
43
44          //start
45          request.send(null);
46
47          return request.responseText;
48  }
49
50
51  // syncrounous
52  var response = getDataSync('/data/json/gigs.json');
53  alert(response);
54
55  getData('/data/json/gigs.json', function(response){
56          alert(response);
57  });
58
59
60  // PROMISES
61
62  // HARDCODED
63  function readJSON(filename){
```

```
  return new Promise(function (fulfill, reject){
    readFile(filename, 'utf8').done(function (res){
      try {
        fulfill(JSON.parse(res));
      } catch (ex) {
        reject(ex);
      }
    }, reject);
  });
}

// <script src="https://www.promisejs.org/polyfills/promise-done-7.0.4.min.js"></script>
// with script tag - shorter
function readJSON(filename){
  return readFile(filename, 'utf8').then(function (res){
    return JSON.parse(res);
  });
}

// even shorter
function readJSON(filename){
  return readFile(filename, 'utf8').then(JSON.parse);
}

// jQuery way
var jQueryPromise = $.ajax('/data.json');
var realPromise = Promise.resolve(jQueryPromise);

// PROPER ANGULAR WAY
$scope.maps = [];
$scope.map = null;
function load(maps){
    $scope.maps = maps;
    $scope.map = maps[0];
}
GigsDataService.maps.query().$promise.then(load);

// OTHER ANGULAR WAYS
// $http.get
// GigsHttpService.getMaps()
//    .then(load);

// GigsDataService.maps.query().$promise.then(function(maps){
//     $scope.maps = maps;
//     $scope.map = maps[0];
// });

// Promise.reject - It's best to always avoid throwing synchronous exceptions in an asychronous method.
var rejectedPromise = Promise.reject(new Error('Whatever'));

// Promise.all - The all function returns a new promise which is fulfilled with an array of fulfillment values for the passed promises or r
function readJsonFiles(filenames) {
  // N.B. passing readJSON as a function, not calling it with `()`
  return Promise.all(filenames.map(readJSON));
}
readJsonFiles(['a.json', 'b.json']).done(function (results) {
  // results is an array of the values stored in a.json and b.json
}, function (err) {
  // If any of the files fails to be read, err is the first error
});

function all(promises) {
  var accumulator = [];
  var ready = Promise.resolve(null);
  promises.forEach(function (promise) {
    ready = ready.then(function () {
      return promise;
    }).then(function (value) {
      accumulator.push(value);
    });
  });
  return ready.then(function () { return accumulator; });
}

// Promise.race makes races like this even easier to run:
function timeout(promise, time) {
  return Promise.race([promise, delay(time).then(function () {
    throw new Error('Operation timed out');
  })]);
}

//generators
```

```
function* demo() {
    var res = yield 10;
    assert(res === 32);
    return 42;
}
var d = demo();
var resA = d.next();
// => {value: 10, done: false}
var resB = d.next(32);
// => {value: 42, done: true}
//if we call d.next() again it throws an error

// deferred pattern
// $q
// var dfd = $q.defer();
function asyncGreet(name) {
  var deferred = $q.defer();

  setTimeout(function() {
    deferred.notify('About to greet ' + name + '.');

    if (okToGreet(name)) {
      deferred.resolve('Hello, ' + name + '!');
    } else {
      deferred.reject('Greeting ' + name + ' is not allowed.');
    }
  }, 1000);

  return deferred.promise;
}

var promise = asyncGreet('Robin Hood');
promise.then(function(greeting) {
  alert('Success: ' + greeting);
}, function(reason) {
  alert('Failed: ' + reason);
}, function(update) {
  alert('Got notification: ' + update);
});

// dfd.resolve
// dfd.reject()

function asyncGreet(name) {
  // perform some asynchronous operation, resolve or reject the promise when appropriate.
  return $q(function(resolve, reject) {
    setTimeout(function() {
      if (okToGreet(name)) {
        resolve('Hello, ' + name + '!');
      } else {
        reject('Greeting ' + name + ' is not allowed.');
      }
    }, 1000);
  });
}

var promise = asyncGreet('Robin Hood');
promise.then(function(greeting) {
  alert('Success: ' + greeting);
}, function(reason) {
  alert('Failed: ' + reason);
});

// new Promise(function(resolve, reject){
//   // what we use reject for
// });

// ou don’t need defer to create a simple-valued promise
// Let’s take a look at this case:

var defer;
defer = $q.defer();
defer.resolve(['detail', 'simple']);
return defer.promise;
// It’s easy to see the author just wanted to create a promise with some hard-coded value. Those 4 lines? They can just be rewritten as:
return $q.when(['detail', 'simple']);
// Same goes for this longer case or that one. $q.when() is perfect for when you want to turn a simple value into a promise.

var myFirstDeferred = $q.defer();
async(
        myFirstDeferred.resolve,
```

```
227         myFirstDeferred.reject
228 );
229
230 var myFirstPromise = myFirstDeferred.promise;
231
232 myFirstPromise.then(function(data) {
233         console.log('My first promise succeeded', data);
234     }, function(error) {
235         console.log('My first promise failed', error);
236 });
```