// for the purpose of this example let's assume that variables `$q` and `okToGreet`

// are available in the current lexical scope (they could have been injected or passed in).

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);

});

The Deferred API

A new instance of deferred is constructed by calling $q.defer().

The purpose of the deferred object is to expose the associated Promise instance as well as APIs that can be used for signaling the successful or unsuccessful completion, as well as the status of the task.

**Methods**

* resolve(value) – resolves the derived promise with the value. If the value is a rejection constructed via $q.reject, the promise will be rejected instead.
* reject(reason) – rejects the derived promise with the reason. This is equivalent to resolving it with a rejection constructed via $q.reject.
* notify(value) - provides updates on the status of the promise's execution. This may be called multiple times before the promise is either resolved or rejected.

**Properties**

* promise – {Promise} – promise object associated with this deferred.

## The Promise API

A new promise instance is created when a deferred instance is created and can be retrieved by calling deferred.promise.

The purpose of the promise object is to allow for interested parties to get access to the result of the deferred task when it completes.

**Methods**

* then(successCallback, [errorCallback], [notifyCallback]) – regardless of when the promise was or will be resolved or rejected, then calls one of the success or error callbacks asynchronously as soon as the result is available. The callbacks are called with a single argument: the result or rejection reason. Additionally, the notify callback may be called zero or more times to provide a progress indication, before the promise is resolved or rejected.

This method returns a new promise which is resolved or rejected via the return value of the successCallback, errorCallback(unless that value is a promise, in which case it is resolved with the value which is resolved in that promise using [promise chaining](http://www.html5rocks.com/en/tutorials/es6/promises/#toc-promises-queues)). It also notifies via the return value of the notifyCallback method. The promise cannot be resolved or rejected from the notifyCallback method. The errorCallback and notifyCallback arguments are optional.

* catch(errorCallback) – shorthand for promise.then(null, errorCallback)
* finally(callback, notifyCallback) – allows you to observe either the fulfillment or rejection of a promise, but to do so without modifying the final value. This is useful to release resources or do some clean-up that needs to be done whether the promise was rejected or resolved. See the [full specification](https://github.com/kriskowal/q/wiki/API-Reference#promisefinallycallback) for more information.
* This lets asynchronous methods return values like synchronous methods: instead of the final value, the asynchronous method returns a promise for the value at some point in the future.
* In simple words “A promise is a word taken for some action, the other party who gave the promise might fulfill it or deny it”. In the case of fulfilling, the promise gets resolved, and in another case, it gets rejected.

|  |
| --- |
| var request = require("request"); |
|  | var userDetails; |
|  |  |
|  | function getData(url) { |
|  | // Setting URL and headers for request |
|  | var options = { |
|  | url: url, |
|  | headers: { |
|  | 'User-Agent': 'request' |
|  | } |
|  | }; |
|  | // Return new promise |
|  | return new Promise(function(resolve, reject) { |
|  | // Do async job |
|  | request.get(options, function(err, resp, body) { |
|  | if (err) { |
|  | reject(err); |
|  | } else { |
|  | resolve(body); |
|  | } |
|  | }) |
|  | }) |
|  | } |
|  |  |
|  | var errHandler = function(err) { |
|  | console.log(err); |
|  | } |
|  |  |
|  | function main() { |
|  | var userProfileURL = "https://api.github.com/users/narenaryan"; |
|  | var dataPromise = getData(userProfileURL); |
|  | // Get user details after that get followers from URL |
|  | dataPromise.then(JSON.parse, errHandler) |
|  | .then(function(result) { |
|  | userDetails = result; |
|  | // Do one more async operation here |
|  | var anotherPromise = getData(userDetails.followers\_url).then(JSON.parse); |
|  | return anotherPromise; |
|  | }, errHandler) |
|  | .then(function(data) { |
|  | console.log(data) |
|  | }, errHandler); |
|  | } |
|  |  |
|  |  |
|  | main(); |