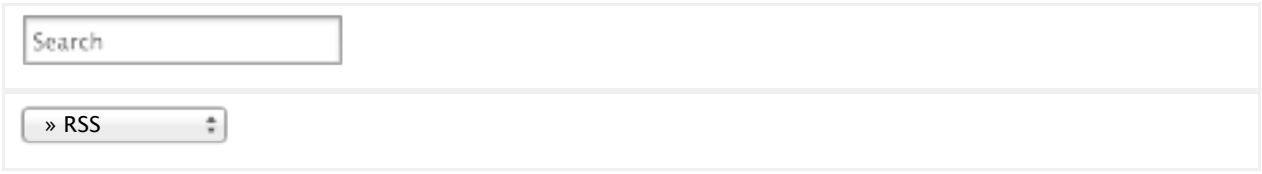
Liam Kaufman

Software Developer and Entrepreneur

- RSS



- About
- Projects
- Blog
- Archives

Using AngularJS Promises

Sep 9th, 2013

In my previous article I discussed the benefits of using dependency injection to make code more testable and modular. In this article I'll focus on using promises within an AngularJS application. This article assume some prior knowledge of promises (a good intro on promises and AngularJS' official documentation).

Promises can be used to unnest asynchronous functions and allows one to chain multiple functions together - increasing readability and making individual functions, within the chain, more reusable.

> Standard Callbacks (no promises)

```
1  function fetchData(id, cb){
2    getDataFromServer(id, function(err, result){
3      if(err){
4        cb(err, null);
5      }else{
6        transformData(result, function(err, transformedResult){
7          if(err){
8            cb(err, null);
9          }else{
10           saveToIndexDB(result, function(err, savedData){
11             cb(err, savedData);
12           });
13         }
14       });
15     }
16   });
17 }
```

Once `getDataFromServer()`, `transformData()` and `saveToIndexDB()` are converted to returning promises we can refactor the above code to:

With Promises

```
function fetchData(id){
  return getDataFromServer(id)
           .then(transformData)
           .then(saveToIndexDB);
}
```

In addition to increasing readability promises can help with error handling, progress updates, and AngularJS templates.

Handling Errors

If `fetchData` is called and an exception is raised in `transformData()` or `saveToIndexDB()`, it will trigger the final error callback.

```
fetchData(1)
  .then(function(result){

  }, function(error){
    // exceptions in transformData, or saveToIndexDB
    // will result in this error callback being called.
  });
```

Unfortunately, if an exception is raised in `getDataFromServer()` it will not trigger the final error callback. This happens because `transformData()` and `saveToIndexDB()` are called within the context of `.then()`, which uses try-catch, and automatically calls `.reject()` on an exception. To bring this behaviour to the first function we can introduce a try-catch block like:

getDataFromServer()

```
function getDataFromServer(id){
  var deferred = $q.defer();

  try{
    // asynchronous function, which calls
    // deferred.resolve() on sucess
  }catch(e){
    deferred.reject(e);
  }

  return deferred.promise;
}
```

While adding try-catch made `getDataFromServer()` less elegant, it makes it more robust and easier to use as the first in a chain of promises.

Using Notify for Progress Updates

A promise can only be resolved, or rejected, once. To provide progress updates, which may happen zero or more times, a promise also includes a notify callback (introduced in AngularJS 1.2+). Notify could be used to provide incremental progress updates on a long running

asynchronous task. Below is an example of a long running function, `processLotsOfData()`, that uses `notify` to provide progress updates.

```
function processLotsOfData(data){
  var output = [],
      deferred = $q.defer(),
      percentComplete = 0;

  for(var i = 0; i < data.length; i++){
    output.push(processDataItem(data[i]));
    percentComplete = (i+1)/data.length * 100;
    deferred.notify(percentComplete);
  }

  deferred.resolve(output);

  return deferred.promise;
};


processLotsOfData(data)
  .then(function(result){
    // success
  }, function(error){
    // error
  }, function(percentComplete){
    $scope.progress = percentComplete;
  });
```

Using the notify function, we can make many updates to the $scope's progress variable before processLotsOfData is resolved (finished), making notify ideal for progress bars.

Unfortunately, using notify in a chain or promises is cumbersome since calls to notify do not bubble up. Every function in the chain would have to manually bubble up notifications, making code a little more difficult to read.

Templates

~~AngularJS templates understand promises and delays their rendering until they're resolved, or rejected~~. AngularJS templates no longer resolve promises - they must be resolved in the controller before they're assigned to the scope. For instance let's say our template looks like:

```
<p>{{bio}}</p>
```

We could do the following in our controller:

```
function getBio(){
  var deferred = $q.defer();
  // async call, resolved after ajax request completes
  return deferred.promise;
};

getBio().then(function(bio){
```

```
8   $scope.bio = bio;
9 });
```

The view renders normally, and when the promise is resolved AngularJS automatically updates the view to include the value resolved in getBio.

Limitations of Promises in AngularJS

When a promise is resolved asynchronously, "in a future turn of the event loop", the .resolve() function must be wrapped in a promise. In the contrived example below, a user would click a button triggering `goodbye()`, which should update the `$scope`'s greeting attribute.

```
1  app.controller('AppCtrl',
2  [    '$scope',
3       '$q',
4       function AppCtrl($scope, $q){
5         $scope.greeting = "hello";
6
7          var updateGreeting = function(message){
8             var deferred = $q.defer();
9
10            setTimeout(function(){
11                deferred.resolve(message);
12            }, 5);
13
14            return deferred.promise;
15         };
16        $scope.goodbye = function(){
17            $scope.greeting = updateGreeting('goodbye');
18        }
19     }
20 ]);
```

Unfortunately, it doesn't work as expected, since the asynchronous event works outside of AngularJS' event loop. The fix for this (besides using AngularJS' setTimemout function), is to wrap the deferred's resolve in `$scope.$apply` to trigger the digest cycle and update the `$scope` accordingly:

```
1 setTimeout(function(){
2   $scope.$apply(function(){
3     deferred.resolve(message);
4   });
5 }, 5)
```

Jim Hoskins goes into more detail on using `$apply`: http://jimhoskins.com/2012/12/17/angularjs-and-apply.html

Conclusions

Using promises is an important part of writing an AngularJS app idiomatically and should help make your code more readable. Understanding their shortcomings, and their strengths make them

much easier to work with.

Posted by Liam Kaufman Sep 9th, 2013

« How AngularJS Made Me a Better Node.js Developer Understanding AngularJS Directives Part 2: ngView »

Comments

Steve · a year ago

Nice topic. About setTimeout, why you just don't use $timeout instead?
In this case you won't need to wrap your code with $scope.$aply

6 · Reply · Share ›

liamks Mod ↗ Steve · a year ago

It was a poorly contrived example :). $timeout would certainly be a better choice, I was just trying to show an asynchronous function that didn't require too much setup.

· Reply · Share ›

Torsten · 3 months ago

Is your notify example really working? I think you have to pack the loop into an asynchronous function. You can not call the notify() before you return the promise.

Here is an example: plnkr.co/edit/CmC1SgOrV3zIWCTm...

· Reply · Share ›

liamks Mod ↗ Torsten · 3 months ago

The example is contrived, but notify can be called within a loop in the above example. However, in reality the actions within the loop would be asyn, so in that case you might have to wrap it's contents in an immediately invoked function expression, or you could pass the promise into the function that does the work on each item and call notify there.

· Reply · Share ›

Pulak · 4 months ago

Article needs to be updated as it is to note that in Angular 1.2 the promise unwrapping feature you mention in the article has been removed.

Modified Controller code:

angular.module('demo1App')

.controller('PromiselimitationCtrl', ['$q', '$timeout','$scope', function ($q, $timeout, $scope) {

$scope.greeting = "hello";

@liamkaufman @liamks Liam Kaufman

Recent Posts

- Smiley Faces in Linux Source Code and Token Statistics
- Understanding AngularJS Directives Part 2: ngView
- Using AngularJS Promises
- How AngularJS Made Me a Better Node.js Developer
- Understanding AngularJS Directives Part 1: ng-repeat and compile

Latest Tweets

- Status updating...

Follow @liamkaufman