**Setting up the Application**

Fortunately, you don’t have to install any additional modules to use $cacheFactory.  Your barebone module definition should suffice.

// Define an angular module for our app

var myApp = angular.module(‘myApp’, []);

**Create the Service Definition**

We will need to define our new cache explicitly.  You’ll have to write your own service to take care of this.

myApp.factory('myCache', function($cacheFactory) {

return $cacheFactory('myData');

});

## Use your new Cache in your Controllers

Now that everything is set up, you should be able to interact with the service and manipulate the new cache with whatever data you want to preserve (assuming it’ll stay static for its intended purpose):

myApp.controller('myController', ['$scope', 'myCache',

function ($scope, myCache) {

  var cache = myCache.get('myData');

  if (cache) { // If there’s something in the cache, use it!

    $scope.variable = cache;

  }

  else { // Otherwise, let’s generate a new instance

    myCache.put(‘myData’, 'This is cached data!');

    $scope.variable = myCache.get('myData');

  }

}

]);

You are provided with two methods: get() and put().  Similar to setting a browser cookie, you can assign any valid javascript variable to a cache object using put() and acquire it at anytime using get().

## Real World Example

Back to the premises I set at the beginning, here’s how it would all work if our application has routing and AJAX calls, and we want to cache the AJAX calls to speed up the routes.

var myApp = angular.module('myApp', ['ngRoute']);

// Define Routing for app

myApp.config(['$routeProvider',

  function($routeProvider) {

    $routeProvider.

      when('/main', {

        templateUrl: 'main.html',

        controller: 'MyMain'

      }).

      otherwise({

        redirectTo: '/main'

      });

    }

]);

// Caching the river...

myApp.factory('myCache', function($cacheFactory) {

return $cacheFactory('myData');

});

// Displays a list of articles in the river

myApp.controller('MyMain', ['$scope', '$http', 'myCache',

function ($scope, $http, myCache) {

   var cache = myCache.get('myData');

   if (cache) {

     $scope.variable = cache;

   }

   else {

     $http.get('http://www.example.com/path/to/api/endpoint')

       .success(function(data) {

         $scope.variable = data;

         myCache.put('myData', data);

       }

    );

  }

}

]);

This is a very elaborate version of the script that allows you finer control over your cache.  However, you can easily eliminate most of the markup in the controller and just use $cacheFactory as an optional parameter in your AJAX request.

myApp.controller('MyMain', ['$scope', '$http', 'myCache',

function ($scope, $http, myCache) {

     $http.get('http://www.example.com/path/to/api/endpoint', {cache: myCache})

       .success(function(data) {

         $scope.variable = data;

       }

    );

  }

}

]);

## Some pointers to keep in mind

(especially if you are transitioning to AngularJS 1.2.x):

* In your controller, you must define you new cache module in the array AND in the function arguments.  It should mirror what you customly wrote in your services (if you’re wondering if that $ is necessary).
* Because we’re using routes in this example, make sure you include ‘ngRoute’ in the module array when you define your app.
* If all goes well, you shouldn’t have to modify your template one bit since you’re merely enhancing the data source and nothing else (one reason why this MVC model is elegant).