Caching

In large, internet-scale web apps, the ability to limit API calls from the client side enables us to create scalable web apps.

Not only does it make the front end appear quicker and more responsive, but it also protects our back end by reducing the amount of work that our back end needs to perform. That way, our back end can serve more client consumers on the front end.

## **What Is a Cache?**

A **Cache** is a component that transparently stores data so that future requests can be served more quickly. It is safe to cache data that doesn’t need to be recomputed often,

Caching works like a big key-value store. There’s a key that points to a cached piece of content. When the content is requested, if the key is found in the cache and is available (i.e., we get a cache hit), then the related content will be served.

If the key is not (i.e., we get a cache miss), then the caching server will need to know how to fetch the data, store it, and return it back to the original requester of the data.

## **Caching through $http**

Angular’s $http service creates a cache with the ID $http (surprise, right?). Enabling the $http request to use this default cache object is simple: The $http method(s) allow us to pass a cache parameter.

### **Default $http Cache**

The default $http cache can be particularly useful when our data doesn’t change very often. We can set it like so:

**$http**({

method: 'GET',

url: '/api/users.json',

cache: true

});

// Or, using the .get helper

**$http**.get('/api/users.json', {

cache: true

});

Now, every request that is made through $http to the URL /api/users.json will be stored in the default $http cache. The key for this request in the $http cache is the full-path URL.

By passing the true parameter in the $http options, we’re telling the $http service to use the default cache. It’s useful to use the default cache if we don’t want to mess with the cache all that often.

We can, however, manipulate the default $http cache if we need to (e.g., if another request we make without caching notifies us of a delta change, we can clear the request in the default $http request).

To reference the $http default request, we simply fetch the cache through the $cacheFactory by ID:

**var** cache = **$cacheFactory**.get('$http');

With the cache in hand, we can do all the normal operations we need and want to do on it, such as retrieve the cached responses, clear items from the cache, or blow away all cached references:

// Fetch the cache for the previous request

**var** usersCache =

cache.get('http://example.com/api/users.json');

// Delete the cache entry for the

// previous request

cache.remove('http://example.com/api/users.json');

// Start over and remove the entire cache

cache.removeAll();

Although we can reference the default cache, it is sometimes more useful to have more control over the cache and create rules around how the cache behaves. We’ll need to create a new cache to use with our requests.

### **Custom Cache**

Telling our $http requests to make requests through our own custom cache is simple. Instead of passing a boolean true with the request, we can pass the instance of the cache.

**var** myCache = **$cacheFactory**.get('myCache');

**$http**({

method: 'GET',

url: '/api/users.json',

cache: myCache

});

// Or, using the .get helper

**$http**.get('/api/users.json', {

cache: myCache

});

Now, instead of using the default cache, $http will use our custom cache.

## **Setting Default Cache for $http**

Although it is easy, it’s not convenient to need to pass an instance of the cache every time we want to make an $http request, especially if we’re using the same cache for every request.

We can set the cache object that $http uses by default through the $httpProvider in a .config() method on our module.

angular.module('myApp', [])

.config(**function**(**$httpProvider**) {

**$httpProvider**.defaults.cache =

**$cacheFactory**('myCache', {capacity: 20});

});

The $http service will no longer use the default cache it creates for us; it will use our own cache, which is effectively now a Least Recently Used (LRU) cache.

An LRU cache keeps only the latest number of caches based upon the capacity of the cache. That is, in our cache that has a capacity of 20, the first 20 requests will be cached, but when the 21st comes in, the least recently requested item will be deleted from the cache. The cache itself takes care of maintaining the details about what to maintain and what to remove.

<http://jsfiddle.net/TheWoollyBully/yTK6f/>