bower install --save angular-cache or npm install --save angular-cache.

angular.module('myApp', ['angular-cache'])

.config(function (CacheFactoryProvider) {

angular.extend(CacheFactoryProvider.defaults, { maxAge: 15 \* 60 \* 1000 });

})

.service('BookService', function (CacheFactory) {

if (!CacheFactory.get('bookCache')) {

// or CacheFactory('bookCache', { ... });

CacheFactory.createCache('bookCache', {

deleteOnExpire: 'aggressive',

recycleFreq: 60000

});

}

var bookCache = CacheFactory.get('bookCache');

return {

findBookById: function (id) {

return $http.get('/api/books/' + id, { cache: bookCache });

}

};

});

**The Basics**

First, inject CacheFactory then create a cache. Let's go:

app.service('myService', function (CacheFactory) {

var profileCache;

// Check to make sure the cache doesn't already exist

if (!CacheFactory.get('profileCache')) {

profileCache = CacheFactory('profileCache');

}

});

Let's add some items to the cache:

profileCache.put('/profiles/34', {

name: 'John',

skills: ['programming', 'piano']

});

profileCache.put('/profiles/22', {

name: 'Sally',

skills: ['marketing', 'climbing', 'painting']

});

Right now, these items will stay in the cache until a page refresh.

Let's have items which are added to profileCache expire after an hour:

profileCache = CacheFactory('profileCache', {

maxAge: 60 \* 60 \* 1000 // 1 hour

});

Perfect. Say we also want the items removed from the cache when they expire:

profileCache = CacheFactory('profileCache', {

maxAge: 60 \* 60 \* 1000 // 1 hour,

deleteOnExpire: 'aggressive'

});

Let's say that when the items do expire, we want to refresh them with new values:

profileCache = CacheFactory('profileCache', {

maxAge: 60 \* 60 \* 1000 // 1 hour,

deleteOnExpire: 'aggressive',

onExpire: function (key, value) {

$http.get(key).success(function (data) {

profileCache.put(key, data);

});

}

});

Or say we want all of our caches to use that configuration as their default:

angular.module('app', ['angular-cache']).config(function (CacheFactoryProvider) {

angular.extend(CacheFactoryProvider.defaults, {

maxAge: 3600000,

deleteOnExpire: 'aggressive',

onExpire: function (key, value) {

var \_this = this; // "this" is the cache in which the item expired

$http.get(key).success(function (data) {

\_this.put(key, data);

});

}

});

});

**Working with a cache**

We can retrieve items from a cache like so:

var profile = profileCache.get('/profiles/34');

profile.name; // 'John'

And get information about items in the cache:

var info = profileCache.info('/profiles/34');

info.isExpired; // false

// etc.

and information about the cache itself:

var info = profileCache.info();

info.size; // 2

info.maxAge; // 3600000

info.deleteOnExpire; // 'aggressive'

// etc.

Items are easily removed, and we can destroy our cache when we're done with it:

profileCache.remove('/profiles/34');

profileCache.get('/profiles/34'); // undefined

profileCache.destroy();

CacheFactory.get('profileCache'); // undefined

**Configuration Options**

These options apply to:

* CacheFactory(cacheId[, options)
* CacheFactory.createCache(cacheId[, options])
* Cache#setOptions(options[, strict])
* Cache#setMaxAge(maxAge), Cache#setOnExpire(onExpire), etc.

**cacheFlushInterval**

If set, remove all items from a cache on an interval after the given number of milliseconds. Default:null.

**capacity**

Maximum number of items a cache can hold. Adding more items than the capacity will cause the cache to operate like an LRU cache, removing the least recently used items to stay under capacity. Default: Number.MAX\_VALUE.

**deleteOnExpire**

Determines the behavior of a cache when an item expires. Default: none.

Possible values:

* none - Cache will do nothing when an item expires.
* passive - Cache will do nothing when an item expires. Expired items will remain in the cache until requested, at which point they are removed, and undefined is returned.
* aggressive - Cache will remove expired items as soon as they are discovered.

**disabled**

Determines whether a cache is disabled. Default: false.

**onExpire**

A callback function to be executed whenever an expired item is removed from a cache when the cache is in passive or aggressive mode. Will be passed the key and value of the expired item.

Will be passed a third done argument if the cache is in passive mode. This allows you to synchronously access the key and value of the expired item when you make the Cache#get(key[, options]) call that is the reason the expired item is being removed in the first place. Default: null.

**maxAge**

The number of milliseconds until a newly inserted item expires. Default: Number.MAX\_VALUE.

**recycleFreq**

Determines how often a cache will scan for expired items when in aggressive mode. Default: 1000(milliseconds).

**storageImpl**

Provide a custom storage medium, e.g. a polyfill for localStorage. Default: null.

Must implement:

* setItem - Same API as localStorage.setItem(key, value)
* getItem - Same API as localStorage.getItem(key)
* removeItem - Same API as localStorage.removeItem(key)

**storageMode**

Determines the storage medium used by a cache. Default: memory.

Possible values:

* memory - Cache will hold data in memory. Data is cleared when the page is refreshed.
* localStorage - Cache will hold data in localStorage if available. Data is *not* cleared when the page is refreshed.
* sessionStorage - Cache will hold data in sessionStorage if available. Data is *not* cleared when the page is refreshed.

**storagePrefix**

Determines the namespace of a cache when storageMode is set to localStorage or sessionStorage. Make it a shorter string to save space. Default: angular-cache.caches..

**storeOnReject**

If inserting a promise into a cache, also insert the rejection value if the promise rejects. Default: false.

**storeOnResolve**

If inserting a promise into a cache, also insert the resolved value if the promise resolves. Default:false.