March 8, 2014 by naorye | Articles in $http, AngularJS, AngularJS Interceptors, JavaScript

Interceptors in AngularJS and Useful Examples

The `$http` service of AngularJS allows us to communicate with a backend and make HTTP requests. There are cases where we want to capture every request and manipulate it before sending it to the server. Other times we would like to capture the response and process it before completing the call. Global http error handling can be also a good example of such need. Interceptors are created exactly for such cases. This article will introduce AngularJS interceptors and will provide some useful examples.

What are Interceptors?

The `$httpProvider` provider contains an array of interceptors. An interceptor is simply a regular service factory that is registered to that array. This is how we create an interceptor:

Interceptor declaration

```
module.factory('myInterceptor', ['$log', function($log) {
    $log.debug('$log is here to show you that this is a regular factory with injectio

    var myInterceptor = {
        ....
        ....
        ....
    };
```

```
    return myInterceptor;
}]);
```

And then add it by it's name to `$httpProvider.interceptors` array:

Add the interceptor to $httpProvider.interceptors

```
module.config(['$httpProvider', function($httpProvider) {
    $httpProvider.interceptors.push('myInterceptor');
}]);
```

Interceptors allow you to:

- **Intercept a request by implementing the `request` function:** This method is called before `$http` sends the request to the backend, so you can modify the configurations and make other actions. This function receives the request configuration object as a parameter and has to return a configuration object or a promise. Returning an invalid configuration object or promise that will be rejected, will make the `$http` call to fail.

- **Intercept a response by implementing the `response` function:** This method is called right after `$http` receives the response from the backend, so you can modify the response and make other actions. This function receives a response object as a parameter and has to return a response object or a promise. The response object includes the request configuration, headers, status and data that returned from the backend. Returning an invalid response object or promise that will be rejected, will make the `$http` call to fail.

- **Intercept request error by implementing the `requestError` function:** Sometimes a request can't be sent or it is rejected by an interceptor. Request error interceptor captures requests that have been canceled by a previous request interceptor. It can be used in order to recover the request and sometimes undo things that have been set up before a request, like removing overlays and loading indicators, enabling buttons and fields and so on.

- **Intercept response error by implementing the `responseError` function:** Sometimes our backend call fails. Other times it might be rejected by a request interceptor or by a previous response interceptor. In those cases, response error interceptor can help us to recover the backend call.

Asynchronous Operations

Sometimes there is a need to make some asynchronous operations inside the interceptor. Luckily AngularJS allows us to return a promise that will be resolved later. This will defer the request sending in case of request interceptor and will defer the response resolving in case of response interceptor.

Make asynchronous operations in request interceptor

```
module.factory('myInterceptor', ['$q', 'someAsyncService', function($q, someAsyncServ
    var requestInterceptor = {
        request: function(config) {
            var deferred = $q.defer();
            someAsyncService.doAsyncOperation().then(function() {
                // Asynchronous operation succeeded, modify config accordingly
                ...
                deferred.resolve(config);
            }, function() {
                // Asynchronous operation failed, modify config accordingly
                ...
                deferred.resolve(config);
            });
            return deferred.promise;
        }
    };

    return requestInterceptor;
}]);
```

In this example, the request interceptor makes an asynchronous operation and updates the config according to the results. Then it continues with the modified config. If `deferred` is rejected, the http request will fail.
The same applies for response interceptor:

Make asynchronous operations in response interceptor

```
module.factory('myInterceptor', ['$q', 'someAsyncService', function($q, someAsyncServ
    var responseInterceptor = {
        response: function(response) {
            var deferred = $q.defer();
            someAsyncService.doAsyncOperation().then(function() {
                // Asynchronous operation succeeded, modify response accordingly
                ...
                deferred.resolve(response);
            }, function() {
                // Asynchronous operation failed, modify response accordingly
                ...
                deferred.resolve(response);
            });
            return deferred.promise;
        }
    };

    return responseInterceptor;
}]);
```

Only when `deferred` is resolved, the request will succeed. If `deferred` is rejected, the request will fail.

Examples

In this section I'll provide some examples to AngularJS Interceptors in order to give a good understanding of how to use them and how they can help you. Keep in mind that the solutions I provide here are not necessarily the best or the most accurate solutions.

Session Injector (request interceptor)

There are two ways of implementing server side authentication. The first one is to use the traditional Cookie-Based Authentication that uses server side cookies to authenticate the user on each request. The other approach is Token-Based Authentication. When the user logs in, he gets `sessionToken` from the backend. This `sessionToken` identifies the user in the server and is sent to the server on each request.
The following `sessionInjector` adds `x-session-token` header to each intercepted request (in case the current user is logged in):

Session Injector

```
module.factory('sessionInjector', ['SessionService', function(SessionService) {
    var sessionInjector = {
        request: function(config) {
            if (!SessionService.isAnonymus) {
                config.headers['x-session-token'] = SessionService.token;
            }
            return config;
        }
    };
    return sessionInjector;
}]);
module.config(['$httpProvider', function($httpProvider) {
    $httpProvider.interceptors.push('sessionInjector');
}]);
```

And now creating a get request:

Creating a request

```
$http.get('https://api.github.com/users/naorye/repos');
```

The configuration object before intercepted by `sessionInjector`:

Before interceptor

```
{
    "transformRequest": [
        null
    ],
    "transformResponse": [
        null
    ],
    "method": "GET",
    "url": "https://api.github.com/users/naorye/repos",
    "headers": {
        "Accept": "application/json, text/plain, */*"
    }
```

```
}
```

The configuration object after intercepted by `sessionInjector`:

After interceptor

```
{
    "transformRequest": [
        null
    ],
    "transformResponse": [
        null
    ],
    "method": "GET",
    "url": "https://api.github.com/users/naorye/repos",
    "headers": {
        "Accept": "application/json, text/plain, */*",
        "x-session-token": 415954427904
    }
}
```


Timestamp Marker (request and response interceptors)

Let's measure the time it takes to get a backend response using interceptors. It is done by adding a timestamp for each request and response:

Timestamp Marker

```
module.factory('timestampMarker', [function() {
    var timestampMarker = {
        request: function(config) {
            config.requestTimestamp = new Date().getTime();
            return config;
        },
        response: function(response) {
            response.config.responseTimestamp = new Date().getTime();
            return response;
        }
    };
    return timestampMarker;
}]);
module.config(['$httpProvider', function($httpProvider) {
    $httpProvider.interceptors.push('timestampMarker');
}]);
```

And now we can do:

Timestamp Marker usage

```
$http.get('https://api.github.com/users/naorye/repos').then(function(response) {
    var time = response.config.responseTimestamp - response.config.requestTimestamp;
    console.log('The request took ' + (time / 1000) + ' seconds.');
});
```

Here you can find an example for the Timestamp Marker.

Request Recover (request error interceptor)

In order to demonstrate a request error interceptor we have to simulate a situation where a previous interceptor rejects the request. Our request error interceptor will get the rejection reason and will recover the request.
Let's create two interceptors: `requestRejector` and `requestRecoverer`.

Request Recoverer

```
module.factory('requestRejector', ['$q', function($q) {
    var requestRejector = {
        request: function(config) {
            return $q.reject('requestRejector');
        }
    };
    return requestRejector;
}]);
module.factory('requestRecoverer', ['$q', function($q) {
    var requestRecoverer = {
        requestError: function(rejectReason) {
            if (rejectReason === 'requestRejector') {
                // Recover the request
                return {
                    transformRequest: [],
                    transformResponse: [],
                    method: 'GET',
                    url: 'https://api.github.com/users/naorye/repos',
                    headers: {
                        Accept: 'application/json, text/plain, */*'
                    }
                };
            } else {
                return $q.reject(rejectReason);
            }
        }
    };
    return requestRecoverer;
}]);
module.config(['$httpProvider', function($httpProvider) {
    $httpProvider.interceptors.push('requestRejector');
    // Removing 'requestRecoverer' will result to failed request
```

```
    $httpProvider.interceptors.push('requestRecoverer');
}]);
```

And now, if we do the following, we will get the log `success` even though `requestRejector` rejected the request:

Request Recoverer example

```
$http.get('https://api.github.com/users/naorye/repos').then(function() {
    console.log('success');
}, function(rejectReason) {
    console.log('failure');
});
```

Here you can find an example for the Request Recover.

Session Recoverer (response error interceptor)

There are times, in our single page application, where the session gets lost. Such situation might happen due to session expiration or a server error. Let's create an interceptor that will recover the session and resend the original request again automatically (for situations where the session expired).
For the example purposes, let's assume that the http status code for session expiration is 419.

Session Recoverer

```
module.factory('sessionRecoverer', ['$q', '$injector', function($q, $injector) {
    var sessionRecoverer = {
        responseError: function(response) {
            // Session has expired
            if (response.status == 419){
                var SessionService = $injector.get('SessionService');
                var $http = $injector.get('$http');
                var deferred = $q.defer();

                // Create a new session (recover the session)
                // We use login method that logs the user in using the current creden
                // returns a promise
                SessionService.login().then(deferred.resolve, deferred.reject);

                // When the session recovered, make the same backend call again and c
                return deferred.promise.then(function() {
                    return $http(response.config);
                });
            }
            return $q.reject(response);
        }
    };
    return sessionRecoverer;
}]);
module.config(['$httpProvider', function($httpProvider) {
    $httpProvider.interceptors.push('sessionRecoverer');
```

```
}]);
```

This way, whenever a backend call fails due to session expiration, `sessionRecoverer` creates a new session and performs the backend call again.

Summary

In this article I explained about AngularJS interceptors. I presented `request`, `response`, `requestError` and `responseError` interceptors and described how and when to use them. I also provided real world useful examples that can help you in your development.

I hope you enjoyed reading this article as much as I enjoyed writing it!
Good Luck!
NaorYe

naorye

← Back Button Behavior on a Page With an iframe • Service Providers in AngularJS + Logger Implementation →

Anil Singh • 8 months ago
thanks
3 • Reply • Share ›

Francisc • a year ago
I have 2 questions regarding the "Session Recoverer" example:

1. Why did you use $injector instead of just injecting $http as a dependency? To avoid circular dependency or something else?

2. What is the point of the SessionService? Would it not have been enough to just return $http(response.config)?

Thanks.
3 • Reply • Share ›

naorye Mod ➚ Francisc • a year ago
Hi Francisc,

Sorry for waiting to my response.
1. Look on the config block. It registers 'sessionRecoverer' into $httpProvider's interceptors list. This means that 'sessionRecoverer' is a dependency for '$http'. Therefore injecting '$http' as a dependency at the declaration level will cause a circular dependency. In order to avoid it, I asked '$http' on runtime.
2. 'sessionRecoverer' should re-login the user in case of session lost. When a session lost, we can't just make the http call again because the session still lost. We need to re-login the user and this is the purpose of 'SessionService'.

I hope I was clear. If you have any other question - please ask.

Thanks,
NaorYe
5 • Reply • Share ›

Ben Brown ➚ naorye • 8 months ago
Found this in a search for a way around circular dependency errors, cheers ^^
• Reply • Share ›

Francisc ➚ naorye • a year ago
Thank you.
• Reply • Share ›

Sreehari Inukollu • 3 months ago
Nice Work
1 • Reply • Share ›

SocialChooozy • 12 hours ago
Real world interceptor use cases. No water, just meat. Thank you soo much!
• Reply • Share ›

DanCoutoS • 4 days ago
First of all, Great article **@naorye** !
And i have some issue... I hope you could help me about it.
Well, I have a webService that uses token on each POST request to authenticate the session...
Let's just say that I have a simple http request...

//$http(someConfigHere).success(DoStuff).error(Don'tDoStuff);

And I Have my interceptor like yours. But it
will inject the new token on rejection config data and on my local storage

Everything
goes according to the plan. My first request goes 401, my injector catches the
401, login again, inject the new token on my previous request and returns a new
http and this request goes 200.

However,
the first request that originally goes 401, still going 401. And, in
background, my new http request made by my injector, goes 200 ._.

There is some way to return the response of http request made by my injector , on a 401 rejected request?

And thx for your attention and Sry by my bad English... (x
• Reply • Share ›

kumar immanuel • 20 days ago
thanks bro...Its helpful...
• Reply • Share ›

Alex Novikov • a month ago
Hi NaorYe,
This is the best article about interceptors I've ever read.
• Reply • Share ›

Alex • a month ago
The best article between 5 that I've read about interceptors.
• Reply • Share ›

Jimit Shah • 2 months ago
Awesome article. Especially the usecase part is too good.
• Reply • Share ›

Yuriy Babenko • 3 months ago
Great stuff. Thank you.
• Reply • Share ›

sujeewa • 4 months ago
Great article .Thank you for sharing this.
• Reply • Share ›

Nick Johnson • 4 months ago
Nice to see a description of what it is, and then some examples of why you might want it. Good article.
• Reply • Share ›

Mohan Radhakrishnan • 5 months ago
This seems to be a good way to intercept cookies and handle the logged-in user's details in angular-cache. Since these two storages are different I need to manually synchronize them with each other. Cookies expire due to inactivity and the cache does not know about them.
• Reply • Share ›

naorye Mod → Mohan Radhakrishnan • 5 months ago
You right. One way to do this is using interceptors.
• Reply • Share ›

Anthony • 6 months ago
Very nice article!!
could you please make an example how you could save a request to be executed only when you have network connectivity? As long as you have no network connectivity it should save the requests.
the detection of connectivity is not a problem, but how do you save a request you made and execute it again once you are connected again?
could you please make an example of this?

thank you
kind regards
• Reply • Share ›

naorye Mod → Anthony • 6 months ago
Anthony, thanks for the feedback.
Please ask questions here: http://www.webdeveasy.com/ask and I'll do my best to answer.

NaorYe
• Reply • Share ›

Didier • 7 months ago
Good article!
• Reply • Share ›

Sergey Romanov • 7 months ago
I have a question. Sometimes my server return error, but only as a JSON. Like in json I can see that this is error, but there is no error by the server. So in my code I always use .success and theer check if response is an error or not. But I would like to be able to use .success and .error for cleaner code.

How can I in inspector, in response, if I identify that response.data is actually a JSON error, to trigger response error so that in the code I could use .error() even though server returned 200 OK..
• Reply • Share ›

naorye Mod → Sergey Romanov • 7 months ago
Hi Sergey,
I've created a question (the first one!) in our new Q&A system. Here is a link to it:
http://www.webdeveasy.com/ask/...

If my answer helps or in case you have other questions, I really appreciate if you log in and mark this answer as helpful / add comments / ask questions.

• Reply • Share ›

vikash kumar • 7 months ago
Very-very nice tutorial.. thank you so much. Keep it up.. :)
• Reply • Share ›

goutham vel • 8 months ago
Just wanted to say thanks for article.
• Reply • Share ›

juristr • 10 months ago
Cool article. I was just wondering whether there is a way to get $http interceptors work with plain standard $.ajax requests as well. Meaning...when you have a 3rd party library which internally uses $.ajax for executing requests. Is there a way to integrate it (through some kind of adapter) s.t. the $http interceptors fire??
• Reply • Share ›

naorye Mod → juristr • 10 months ago
I think that an easy way will be to override $.ajax and implement it using $http.
• Reply • Share ›

juristr → naorye • 10 months ago
Is that the recommended way or how it's normally done? Meanwhile I found that Breeze.js (which has the same "problem") provides the developer some adapters...so I'll see how they solved it..
• Reply • Share ›

naorye Mod → juristr • 10 months ago
Please report back :)
• Reply • Share ›

megapotz • 10 months ago
Amazing! Now it's all clear and I have a perfect understanding on what interceptors are for, and where's the boundary of what should be done with interceptor and transformRequest / transformResponse!
• Reply • Share ›

ruen • 10 months ago
Thank you for this examples. I am now using them for my current project. Especially the "request" part :)
• Reply • Share ›

CYB • 10 months ago
Great article! Thanks!
• Reply • Share ›

nadimtuhin • a year ago
you are the man
• Reply • Share ›

Duc Hong • a year ago
great article about Interceptors, one thing I want to ask is, does this interceptors-thing got called every time an xhr request is called ? In case I just want to modify a little on a specific request, I should be checking the request.url separately so it won't apply the same modification to all requests ?
Thanks
• Reply • Share ›

naorye Mod → Duc Hong • 10 months ago
If you are using $resource, you can mention a response interceptor for specific request (https://docs.angularjs.org/api....
Otherwise, once you called "$httpProvider.interceptors.push('yourInterceptor');", each request will pass through yourInterceptor. One possible solution can be to check the request.url.
• Reply • Share ›

Palanisamy Thangamuthu • a year ago
Thanks for the article, especially those real-world examples are very nice :-)
• Reply • Share ›

Martin Geisler • a year ago
Very nice article! As a small note: the session Recoverer doesn't actually need the $q service if you write it like this (hoping the formatting is okay):

```
if (response.status == 419) {
var SessionService = $injector.get('SessionService');
var $http = $injector.get('$http');
// Create a new session (recover the session)
```

```
// We use login method that logs the user in using the current credentials and
// returns a promise
return SessionService.login().then(function () {
// When the session recovered, make the same backend call again and chain the request
return $http(response.config);
});
}
throw response;
----------------
```

That is, instead of creating a new promisse to return, we can use the promise we get when calling login().then(...). When the fulfillment handler itself a promise, the outer promise will "wait" for this promise and use its value as its own value. That is, a fullfillment handler can either return a value directly or it can return a promise which will be chained into the callback chain automatically.

Also, returning a rejected deferred is the same as throwing the error (interceptors are applied to a promise using then()).

• Reply • Share ›

naorye Mod → Martin Geisler • 10 months ago

You are absolutely right. Still, I remain the code as is because it is not an issue.
You can read more about promises and deferrers here: http://www.webdeveasy.com/java...

• Reply • Share ›

Jinto Jose • a year ago

As mentioned in my previous comment, I created a custom interceptor for aborting angular http and jquery ajax requests. If you can take a look and give your valuable suggestions, that would be great.

http://programmerbuddy.blogspo...

• Reply • Share ›

naorye Mod → Jinto Jose • 10 months ago

Just saw your second comment :) Sorry for the long delay.
I can think of other suitable way to solve your problem (for example using a service that manages all the page data loading, using deferrers and promises). Your solution seems to be a little bit complicate. I would like to see it's code. Can you share a link to your interceptor?

• Reply • Share ›

Jinto Jose → naorye • 10 months ago

Me too missed your reply to 2nd comment :-)
Here is the gist for the code I mentioned.
https://gist.github.com/jintop...

• Reply • Share ›

naorye Mod → Jinto Jose • 10 months ago

As said before, this is not intuitive way, but I can see the advantage of using such an interceptor over a service. This way you can use the regular api of angular ui route where your interceptor takes care of the data synchronization.

Despite all this,my preferred solution for your case is to create a service that responsible for state change. Whenever you changes your state using this service, it fetches the relevant data and then moving to the desired state. Saving a deferred to the last fetch will let you reject it if new route change had made.

Either way, you have to use a solution that fits your needs and logic.

Hope this helps,
NaorYe

• Reply • Share ›

Jinto Jose • a year ago

I want to cancel all old request which are not relevant anymore since user changed the navigation. Using interceptor to handle this scenario would be a good approach? What do you suggest? Thanks in advance

• Reply • Share ›

naorye Mod → Jinto Jose • 10 months ago

What do you mean by "cancel all old requests"? Once a request made by your app, a response will come. Do you want to cancel all responses? To return rejected promise?

• Reply • Share ›

Jinto Jose → naorye • 10 months ago

I came up mainly because of the following scenario.

1. User doing some actions which will trigger an http request.

2. Immediately (before the response came), moves to another route url. For our application, the old request is now not valid anymore.

3. So, on route change, all the request which happened from the previous route will be cancelled, since that request responses are not needed anymore.

Another scenario. Search box

1. User types some text. Request goes with the searchterm

2. Immediately user types again. It causes a race condition.
3. The first search is not valid anymore. So, we can just cancel all previous search request.

Hope this makes sense.

I have now created an interceptor with this approach. Please have a look and let me know your thoughts.

https://gist.github.com/jintop...

Thanks

• Reply • Share ›

naorye Mod → Jinto Jose • 10 months ago

You have two actions:
1. route change
2. new data request
Those two actions has dependency relation, This means, for example, that if one of them fails, the other is not relevant. Therefore you have to bind those two actions together. Currently you made this bind using an interceptor which is not an intuitive way and may be incorrect. I'll look later on your code and write my opinion (in your second post).

NaorYe

• Reply • Share ›

Rahul Ranjith • a year ago

I would like to know one thing whether we can define interceptors as functions like function interc (....) and add that to interceptors list inside a provider..like

App.provider($httpprovider){
function intercp (){
....request :
...response :}
)
$ httpprovider.interceptors.push (intercp)

$ get :function (){

}
)

I am setting interceptors inside provider rather than in the .config .it works for me .in my interceptor I show a spinner in ui using broadcasts

Is this a nice approach ?

• Reply • Share ›

Load more comments