revolunet blog

web technologies for desktop and mobile

- RSS

Search

- Blog
- Archives
- About

Object-oriented AngularJS Services

Feb 14th, 2014 | Comments

Javascript prototypal inheritance can be confusing at first if you come from classical OOP languages, due to Javascript versatility, and the variety of Javascript OOP patterns available. Combined with the new service and factory concepts introduced in AngularJS, implementing OOP in your applications can lead to serious headaches, so i'll try to show you some solutions here.

If you need a step-by-step explanation of the Javascript prototypal inheritance, you can read the great Dr. Axel Rauschmayer JavaScript inheritance by example article.

Once your app grow and your services multiply, you'll quickly feel the need to reuse your code, and to split it in small modules to be able to separate concerns and setup some serious unit testing.

In this post, i'll show how to create a base AngularJS service, based on the Github API, that we'll be able to extend and reuse in different scenarios. We'll also leverage the power of promises chaining to extend the server responses and add additional data before returning the final result.

Something important to note here is that `factories` are useful to define our classes that you can instantiate many times using the `new` keyword, while `services` always create singletons.

Create our base service

Our first service will be responsible of fetching Github basic user data and return the result. We'll use a `factory` instead of a `service`, which will make it easier to instantiate many versions of the service in our application.

```
app.factory('SimpleGithubUser', function($http) {

    var apiUrl = 'https://api.github.com/';

    // instantiate our initial object
    var SimpleGithubUser = function(username) {
        this.username = username;
        this.profile = null;
    };

    // define the getProfile method which will fetch data
    // from GH API and *returns* a promise
    SimpleGithubUser.prototype.getProfile = function() {

        // Generally, javascript callbacks, like here the $http.get callback,
        // change the value of the "this" variable inside it
        // so we need to keep a reference to the current instance "this" :
        var self = this;

        return $http.get(apiUrl + 'users/' + this.username).then(function(response) {

            // when we get the results we store the data in user.profile
            self.profile = response.data

            // promises success should always return something in order to allow chaining
            return response;

        });
    };
    return SimpleGithubUser;
})
```

So we can now easily inject our factory anywhere and use it like this :

```
// we first inject our factory
app.controller('MyCtrl', function(SimpleGithubUser) {
    // instantiate a new user
    var user = new SimpleGithubUser('substack');
    // fetch data and publish on scope
    user.getProfile().then(function() {
        $scope.userLogin = user.profile.login;
    })
});
```

Extending the base service

Now we'd like to attach some additional data to our users. Instead of modiying the original factory, or even worse, duplicate it, we can create another factory that extends the original one, just by using the regular javascript prototypal inheritance. We'll override some methods and use promises chaining to deliver the final data only when all the subsequent requests have been completed.

This has the advantage of encapsulating the logic inside the new service, making it easily testable while keeping your controllers light.

In this example we'll add some data from the Github events API and attach it to the user profile before returning the final result.

```
// we define a new factory and inject our original service so we can extend it properly
app.factory('AdvancedGithubUser', function($http, SimpleGithubUser) {
```

```
3
4     var apiUrl = 'https://api.github.com/';
5
6     // create our new custom object that reuse the original object constructor
7     var AdvancedGithubUser = function() {
8         SimpleGithubUser.apply(this, arguments);
9     };
10
11    // reuse the original object prototype
12    AdvancedGithubUser.prototype = new SimpleGithubUser();
13
14    // define a new internal private method for this object
15    function getUserEvents() {
16        var self = this;
17        return $http.get(apiUrl + 'users/' + this.username + '/events').then(function(response) {
18
19            // attach the events API result to our user profile
20            self.profile.events = response.data;
21
22            // promises should always return a result
23            return response;
24        });
25    }
26
27    // Now let's override our original getProfile method
28    AdvancedGithubUser.prototype.getProfile = function() {
29
30        var self = this;
31
32        // we first call the original getProfile method (aka super method)
33        var originalGetProfile = SimpleGithubUser.prototype.getProfile.apply(this, arguments);
34
35        // we use promises chaining to add additional data
36        return originalGetProfile.then(function() {
37
38            // before returning the result,
39            // call our new private method and bind "this" to "self"
40            // we need to do this because the method is not part of the prototype
41            return getUserEvents.call(self);
42        });
43    };
44    return AdvancedGithubUser;
45 });
```

Usage is exactly the same, except the service added the events data for us :

```
1 // we first inject our factory
2 app.controller('MyCtrl', function(AdvancedGithubUser) {
3     // instantiate a new user
4     var user = new AdvancedGithubUser('substack');
5     // fetch data and publish on scope
6     user.getProfile().then(function() {
7         $scope.userEvents = user.profile.events;
8     })
9 });
```

Create a service instance

Now that you have some solid factories, you can also instantiate some app-wide services that expose pre-configured instances.

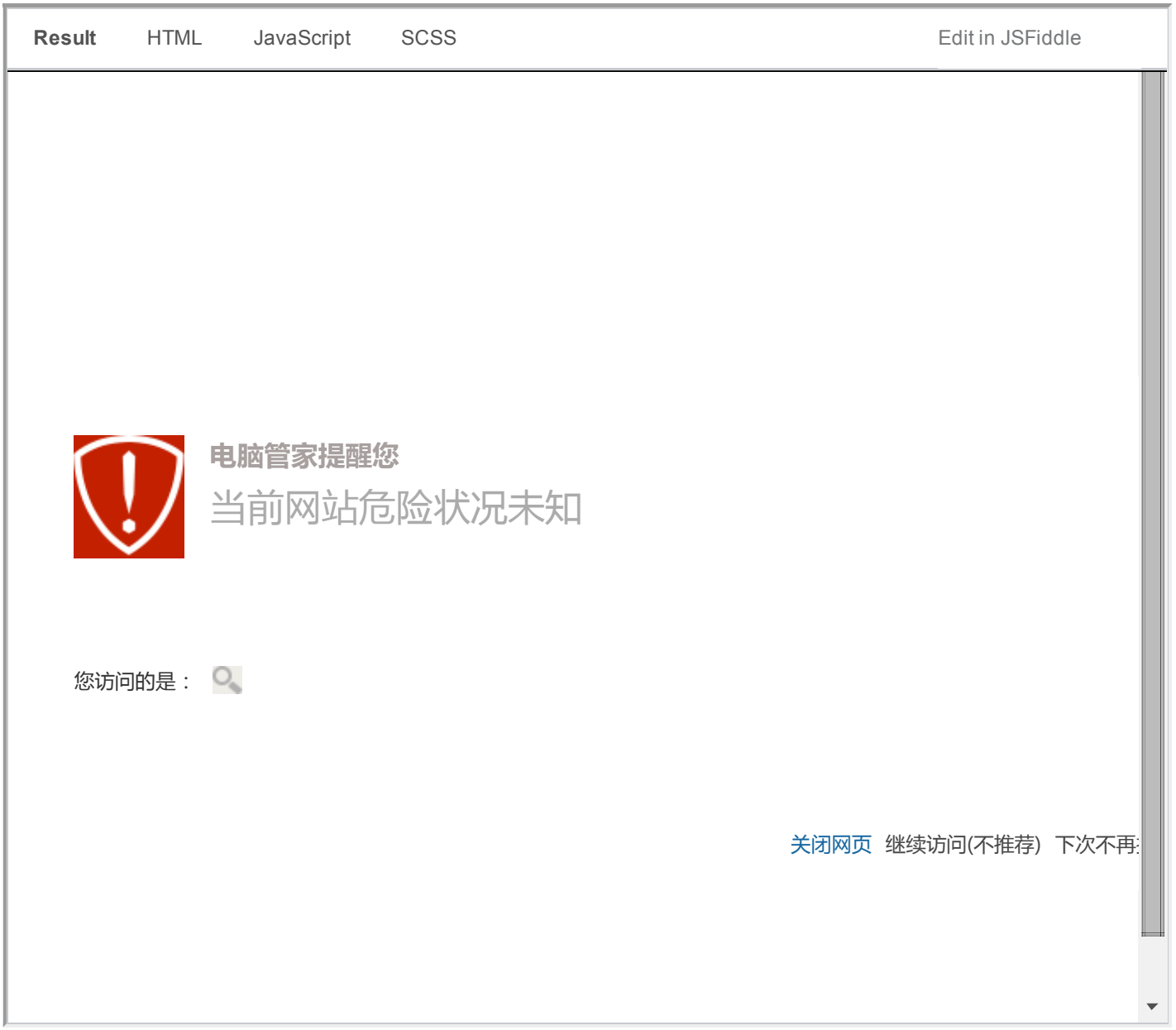
```
1 app.service('MyUserProfile', function(AdvancedGithubUser) {
```

```
    var user = new AdvancedGithubUser('revolunet');
    user.getProfile();
    return user;
});
```

And use it like this :

```
app.controller('MyCtrl', function(MyUserProfile) {
    $scope.user = MyUserProfile;
    alert(MyUserProfile.location);
})
```

Final result

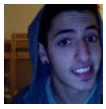


Hope this has been useful to you, please ask below or on twitter @revolunet for any question/suggestion :)

Posted by Julien Bouquillon Feb 14th, 2014 AngularJS

« Unit testing an AngularJS directive Create and host a beautiful website for free using GitHub »

Comments



Simone D'Avico • a year ago

Great solution! Anyway, I think there is a small error. Your solution won't work if SimpleGithubUser needs to be passed some mandatory argument: it would fail where you have

```
AdvancedGithubUser.prototype = new SimpleGithubUser(args);
```

because args would be undefined.

I solved this by changing this line to

```
AdvancedGithubUser.prototype = Object.create(SimpleGithubUser.prototype);
AdvancedGithubUser.prototype.constructor = AdvancedGithubUser;
```

Almost quoting from Speaking Javascript:
Object.create() produces a fresh object whose prototype is SimpleGithubFactory.prototype. We also need to set up the property constructor, because we have replaced the original instance prototype where it had the correct value.

Just wanted to post it here, just in case anyone is having troubles like I had :)

Florian F. • a year ago

One thing that you should be aware of is you're mixing application logic with object instanciation in your controller when you use the 'new' keyword.

What does it mean ? It means you're stuck now if you wan't to mock the AdvancedGithubUser object in your unit tests because you have no control of how the object is instanciated. So by injecting a constructor function in your controller and instantiate manually the object, you lose the power of dependency injection.

Now, I understand that you need to inject the constructor function to functions that need to inherit from it.

So, one solution is to create a Factory. - By Factory, I mean the design pattern as opposed to the angular factory. Note that this confuses a lot of developers that start with Angular as the Angular factory returns a unique instance and the DP returns a new instance at each call) - So the Factory is in charge of instanciating your object and then inject this factory in your controller instead of the constructor function

controller instead of the constructor function.

Here's the updated fiddle : http://jsfiddle.net/fNfTL/2/

Now, If you want to use a mock AdvancedGithubUser objects in your tests, it should be easy because you just have to mock the create function from the Factory.

2 • Reply • Share ›

Julien Bouquillon Florian F. • a year ago

Thanks a lot for taking the time to explain this and make the related fiddle !
Im not sure to get exactly what you mean though about the untestable part. I still can inject a mocked constructor when i test my controller ?

• Reply • Share ›

Valter Julien Bouquillon • 4 months ago

Hi, if you want you can take a look to this talk given by Misko Hevery, It will point you in the Florian F. direction.

• Reply • Share ›

valter Valter • 4 months ago

See the second one.

• Reply • Share ›

sgruhier • a year ago

It's even sexier with CoffeeScript code!

1 • Reply • Share ›

Tom Chen ➚ sgruhier • a year ago

I'd recommend LiveScript (not TypeScript) http://livescript.net/.

▲ | ▼ • Reply • Share ›

mark goldin • 2 months ago

When we extend the base service why do we need to create promises again? All I would want is to extend the base service, and add new server calls and execute them by using some base method.

▲ | ▼ • Reply • Share ›

Mohsen Ghaemmaghami • 3 months ago

Awesome :D
TNX so much!

▲ | ▼ • Reply • Share ›

Jonathas Costa • 6 months ago

What if your getProfile wasn't changed? I tried to call it (from 'base' service), but I get 'TypeError: undefined is not a function'. Do I need to override it anyway?

▲ | ▼ • Reply • Share ›

Jonathas ➚ Jonathas Costa • 6 months ago

Forget it! It is working! :)

▲ | ▼ • Reply • Share ›

Facundo • 6 months ago

Look this!
https://github.com/facka/ngInh...

▲ | ▼ • Reply • Share ›

Ciel • a year ago

Hey... I know you're a busy person, but I'm trying very hard to compile and run your example, and I keep hitting a strange snag that I cannot explain.

I am being told "TypeError: Cannot set property 'events' of undefined"

I have transcribed your code word for word into my own file, and it fails. I have checked it over, and over, and over, and over, and over, and cannot find out the difference other than the fact that I added things to an app variable, instead of chained them like you did. But even if I try to do the chaining, the same error appears.

If I copy and paste your jsFiddle code absolutely verbatim, it works - so there MUST be something I am doing wrong. I've made a topic for it on stackoverflow, here: http://stackoverflow.com/quest...

▲ | ▼ • Reply • Share ›

Julien Bouquillon ➚ Ciel • a year ago

answered on SO :)

▲ | ▼ • Reply • Share ›

Nguyen Tuan • a year ago

great article

• Reply • Share ›

Mathieu ROBIN • a year ago

J'ai jamais pris le temps de te remercier pour cet article, j'avais des petits soucis d'héritage, tu m'as permis de mettre ça au propre ;)

• Reply • Share ›

Julien Bouquillon ➚ Mathieu ROBIN • a year ago

thanks buddy :)

• Reply • Share ›

elijen • a year ago

Can't you use `angular.extend` method for the extending?

http://docs.angularjs.org/api/...

• Reply • Share ›

Julien Bouquillon ➚ elijen • a year ago

it will just copy the attributes you won't be able to traverse the prototype and share methods between instances

• Reply • Share ›

alphaLlama • a year ago

Awesome! I recently started a project and have been using a similar pattern. (I too had to emphasize the diff. between an angular factory vs. a typical OOP factory) The driver for me was inheritance in my services. I was worried i had gone off the deep-end b/c i hadn't seen this done before i ran across this article.

Contrived example how I'm using the pattern with a base Collection and sub-classed FooCollection and BarCollection: http://plnkr.co/edit/lyq4mg?p=....

Testing has been pretty straight forward since I can inject (or mock) whatever instanceFactory service i want.

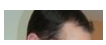
Thanks for posting!

• Reply • Share ›

Kyle Bolton ➚ alphaLlama • a year ago

I like how clean this is, how easy it is to specify which attributes/methods are public, and that it actually follows the factory pattern.
I found a couple issues though:
I don't see how one could implement polymorphism using this pattern. Also, how would one call the parent function from within the child function overriding it?

• Reply • Share ›



KennyDee • a year ago

There is a typo here :
app.controller('MyCtrl', function(UserProfile) {
$scope.user = MyUserProfile;

...

Must be :
app.controller('MyCtrl', function(MyUserProfile) {
$scope.user = MyUserProfile;

Great article by the way :)

• Reply • Share ›

mark goldin ➚ KennyDee • 2 months ago

If I want to use $q.defer() how would I modify your code?

Thanks

• Reply • Share ›

Recent Posts

- Create and host a beautiful website for free using GitHub
- Object-oriented AngularJS services
- Unit testing an AngularJS directive
- Create a reusable AngularJS input form component
- Universal .htaccess CORS support

Tweets de @revolunet

GitHub Repos

- Status updating...

@revolunet on GitHub

Categories

- AngularJS (8)
- apache (1)
- css (1)
- google (1)
- javascript (4)
- mobile (3)
- news (2)

- octopress (1)
- phonegap (2)
- python (1)
- tech (1)
- web (1)