Promises in AngularJS, Explained as a Cartoon

An alternative frontend development article. By Andy Shora

← back to andyshora.com | Tweet 987

One morning, a father says to his son: "Go and get the weather forecast, son!"

Every Sunday morning, a father asks his son to go and find out the weather forecast for the afternoon, by using his super-strong telescope to look across the horizon from the tallest hill by their house. The son promises his dad he will go and get the weather forecast. He creates a promise with his dad at the door when he leaves.

At that point, the dad decides if the weather tomorrow is good, he'll prepare a fishing trip for tomorrow. If it's bad he won't. Also, if the son is unable to get a forecast, he'll stay in as well.

After 30mins or so, the son comes back. Different things happen from week-to-week:

Outcome A) Weather forecast retrieved! Sunshine :-)

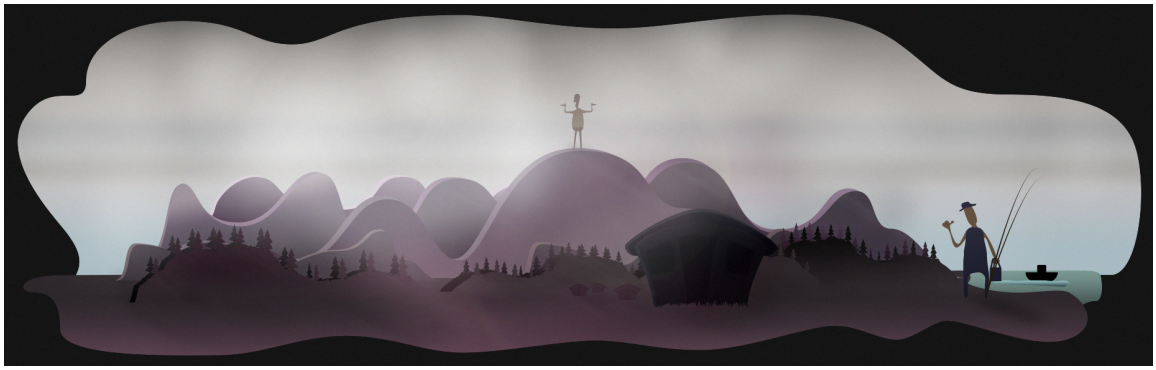
The son succeeded in retrieving the weather forecast, **clear skies and sunshine!** The **promise was fulfilled** (the son kept his promise) and the dad decided to pack up for the fishing trip for Sunday.

Outcome B) Weather forecast retrieved! Cloudy and rain :-(

The son succeeded in retrieving the weather forecast, but it looked like **cloudy and rain**. The **promise was fulfilled** but dad decided to stay in because of the bad weather.

Outcome C) Couldn't get the weather forecast :-/

The son failed to retrieve the forecast, **there was a problem**; it was way too foggy to see what weather was coming over the hills. The promise the son made when he left was broken - the **promise was rejected**! The dad decided to stay in, it wasn't worth the risk.



How does this look in code?

The dad is controlling the logic in this situation, and he's dealing with the Son as if he's a service.

We've already stated the logic, the father asks the son to get the weather forecast, and as the son can't tell him immediately, and the father has other things to do while he waits, the son makes a promise he shall return with the weather. When the dad has the forecast, he'll either pack up the boat, or stay inside. The important thing to note here, is the son's trip up the hill shouldn't 'block' the dad from doing anything, so this is why the situation is perfect for the creation of a promise, which can be resolved (fulfilled or rejected) later on.

Using Angular's then() function we can specify what the Dad needs to do in the event of each outcome. The then() function accepts 2 functions as parameters: a function to be executed when the promise is fulfilled, and a function to be executed when the promise is rejected.

Controller: FatherCtrl

The father is controlling the situation here:

```
// function somewhere in father-controller.js
var makePromiseWithSon = function() {
    // This service's function returns a promise, but we'll deal with
    SonService.getWeather()
        // then() called when son gets back
        .then(function(data) {
            // promise fulfilled
            if (data.forecast==='good') {
                prepareFishingTrip();
            } else {
                prepareSundayRoastDinner();
            }
        }, function(error) {
            // promise rejected, could log the error with: console.log
            prepareSundayRoastDinner();
        });
};
```

Service: SonService

The Son is being used as a service, he climbs the hill and tried to see the weather. We'll suppose when the son is looking through his telescope and looking for the approaching weather, it's analogous to using a weather API, in the sense that it's an asynchronous operation, he may get a variable answer, and there may be a problem (say, a 500

response, foggy skies).

The response from the 'Fishing Weather API' will be returned with the promise, if it was fulfilled. It will be in the format: **{ "forecast": "good" }**

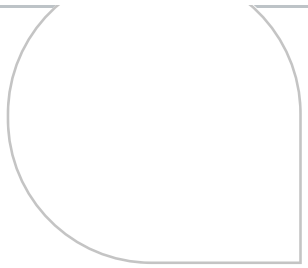
```
app.factory('SonService', function ($http, $q) {
    return {
        getWeather: function() {
            // the $http API is based on the deferred/promise APIs exposed
            // so it returns a promise for us by default
            return $http.get('http://fishing-weather-api.com/sunday/aftern
                .then(function(response) {
                    if (typeof response.data === 'object') {
                        return response.data;
                    } else {
                        // invalid response
                        return $q.reject(response.data);
                    }

                }, function(response) {
                    // something went wrong
                    return $q.reject(response.data);
                });
        }
    };
});
```

Summary

This analogy demonstrates the asynchronous nature of the request the dad makes to his son, for the weather forecast. The dad doesn't want to wait at the door in anticipation when the son leaves, because he has other stuff to do. Instead, he makes a promise at the door, and decides what will happen in either of the 3 scenarios (good weather/bad weather/no forecast). The son immediately gives a promise to his dad when he leaves, and will resolve or reject it on his return.

The son is dealing with an asynchronous service (searching the sky with his telescope/using a weather API) to get data, but all of this is correctly abstracted away from his old man, who doesn't really understand technology!



Hi, I'm **Andy Shora**. I'm a **Frontend Web Developer** based in London and I currently work with some very talented people over at R/GA. I founded Stackey.com - a place to stack things. All work here is my own.

andyshora | fb.com/andyshora | pinterest.com/andyshora
github.com/andyshora | contact me

Illustrations by **Wilf Eddings**. You can find him @iamwilf or at R/GA.

iamwilf | lukeandwilf.com

Kaz · a year ago

I read this to my 8 year old. She understood and and now she's coding this up.

115 · Reply · Share ›

Andy Shora Mod ↗ Kaz · a year ago

This makes me feel warm and fuzzy inside! Thanks, Kaz :-)

29 · Reply · Share ›

Sebastian Gärtner · a year ago

Already got the idea of promises - but it was some kind of a struggle to understand.
With you comic it is really understandable. Thanks!

10 · Reply · Share ›

Robert Pataki ↗ Sebastian Gärtner · a year ago

Same thing here. Somehow never got the grasp of promises in general, but this very article seems to help me a lot to understand the concept (and the code implementation). I also love the invaluable comments (especially the discussion between @thorn and @Esailija), they really add a lot more to the article. Good stuff!

1 · Reply · Share ›

Andy Shora Mod ↗ Sebastian Gärtner · a year ago

Cheers, Sebastian!

· Reply · Share ›

IgorMinar · a year ago

Nice way to explain promises. One correction: when you resolve a promise with a value *it becomes fulfilled* and not resolved.

A promise can be in 3 states: pending, fulfilled or rejected. (resolved is an older term used by Q, which means either fulfilled or rejected, but with Promise A+ spec, resolved is not being used as a term any more).

6 · Reply · Share ›

Andy Shora Mod ↗ IgorMinar · a year ago

Thanks for clearing that up for me, Igor! I've updated the article, nice spot :-)

1 · Reply · Share ›