Promises: resolve is not the opposite of reject

Posted 13 March 2014 and still refusing to do promise-based puns

When I first started working with promises I had the overly simplistic view that passing a value into `reject` would mark the promise as "failed", and passing a value into `resolve` would mark it as "successful". However, the latter isn't always true.

```
new Promise(function(resolve, reject) {
  resolve(something);
}).then(function() {
  console.log("Yey");
}, function() {
  console.log("Boo");
});
```

Even though we aren't calling `reject()`, the rejection callback `console.log("Boo")` will be called if either:

- `something` is not defined, resulting in an error being thrown, which is caught by the promise and turned into a rejection, or
- `something` is a promise that rejects

So:

```
new Promise(function(resolve, reject) {
  resolve(Promise.reject());
}).catch(function() {
  // This is called
});
```

This is a good thing, as it behaves the same as `Promise.resolve()` and the return value from callbacks:

```
var promise1 = Promise.resolve(Promise.reject());
```

```
var promise2 = Promise.resolve().then(function() {
  return Promise.reject();
});
var promise3 = Promise.reject().catch(function() {
  return Promise.reject();
});
```

All promises above are rejected. When you resolve a value with a "then" method, you're deferring the resolution to the eventual non-promise value.

In Practice

You can resolve a value without worrying if it's a value, a promise, or a promise that resolves to a promise that resolves to a promise etc etc.

```
function apiCall(method, params) {
  return new Promise(function(resolve, reject) {
    if (!method) {
      throw TypeError("apiCall: You must provide a method");
    }

    var data = {
      jsonrpc: "2.0",
      method: method
    }

    if (params) {
      data.params = params;
    }

    resolve(postJSON('/my/api/', data));
  });
}
```

Now `apiCall` will reject if `method` isn't provided, or if `postJSON` rejects for whatever reason. You've safely handed off the resolution of the promise onto `postJSON`.

Further reading

- JavaScript promises, there and back again - guide to promises
- ES7 async functions - use promises to make async code even easier

David Fox Powell • a year ago

I think it is worth mentioning that those working with the code

```
var promise1 = Promise.resolve(Promise.reject());
var promise2 = Promise.resolve().then(function() {
  return Promise.reject();
});
var promise3 = Promise.reject().catch(function() {
  return Promise.reject();
});
```

might get confused (I know I did) if they try to copy and paste it into the Chrome console. Essentially, it looks as though an error will be thrown by `Promise.reject()` if it is not immediately used with `then` or `catch`. This might be expected behavior but wanted to mention it to save people time in the future if they encounter this issue. So the above example in the chrome console would have to be (below) to work,

```
var promise1 = Promise.resolve(Promise.reject()).then(() => {}, (err) => {
  console.log(err); //undefined
});
```