JAVASCRIPT PROMISE

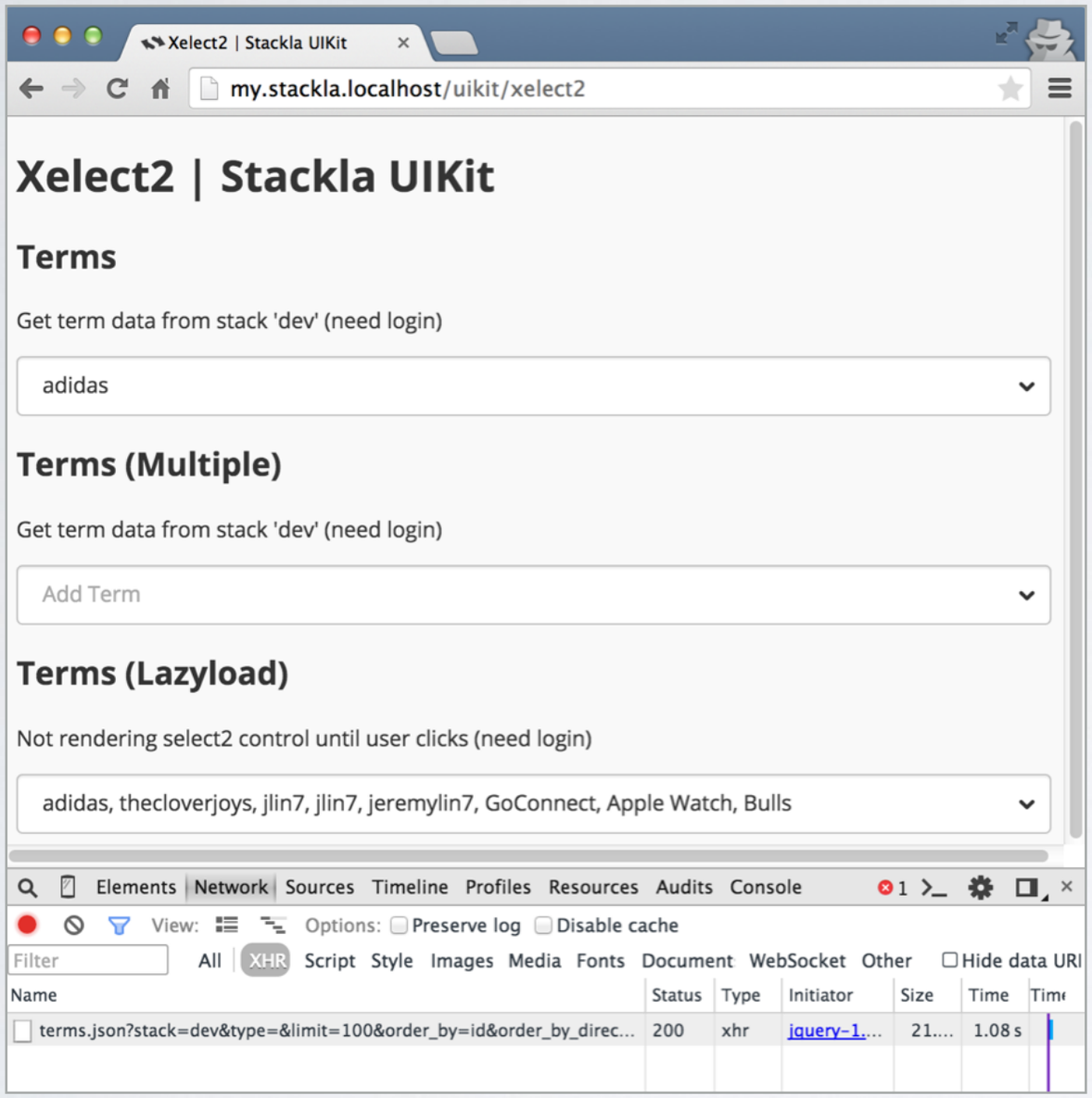
A Good JavaScript Abstraction Pattern

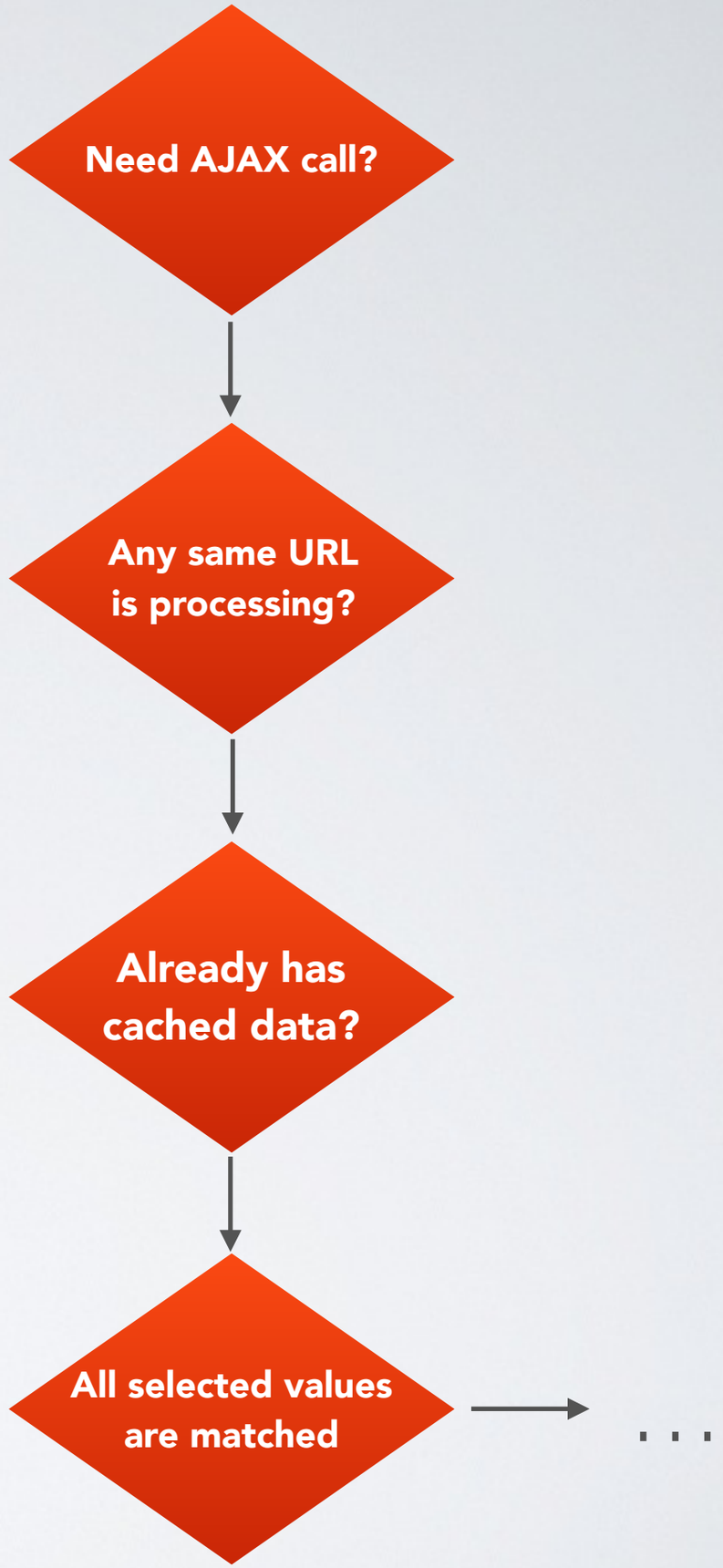
@josephj6802

Stackla

WHY

The reason I decided to investigate Promise…





Render multiple dropdowns
with non-duplicate **AJAX calls**

Stackla

ASYNC & CALLBACK

Handle result when it's done **in the future**

Stackla

Asynchronous Code

Everywhere in JavaScript

Delay

```
setTimeout(function () {
    // do something
}, 1000);
```

AJAX

```
$.ajax('/terms.json', function () {
    // do something after api data
    // being loaded
})
```

Node.js

```
fs.readFile('foo.txt', function () {
    // do something after foo.txt
    // being loaded
})
```

DOM Events

```
$('form').on('submit', function (e) {
    // do something when user submit form
});

$('img').on('load', function (e) {
    // do something when img loaded
});

$('img').on('error', function (e) {
    // do something when img fails loading
});
```

RequireJS

```
require(['lodash', 'jquery'],
    function (_, $) {
    // do something with lodash and jQuery
    }
);
```

Stackla

Asynchronous Code

Everywhere is Callback

Delay

```
setTimeout(function () {
    // do something
}, 1000);
```

AJAX

```
$.ajax('/terms.json', function () {
    // do something after api data
    // being loaded
})
```

Node.js

```
fs.readFile('foo.txt', function () {
    // do something after foo.txt
    // being loaded
})
```

DOM Events

```
$('form').on('submit', function () {
    // do something when user submit form
});

$('img').on('load', function () {
    // do something when img loaded
});

$('img').on('error', function () {
    // do something when img fails loading
});
```

RequireJS

```
require(['lodash', 'jquery'],
    function (_, $) {
    // do something with lodash and jQuery
    }
);
```

Stackla

Asynchronous Code

Everywhere is Callback

Delay

```
setTimeout(function () {
    // do something
}, 1000);
```

DOM Events

```
$('form').on('submit', function () {
    // do something when user submit form
});
$('img').on('load', function () {
    // do something when img loaded
});
$('img').on('error', function () {
    // do something when img fails loading
});
```

AJAX

```
$.ajax('/foo.json').done(function () {
    // do something after api data
    // being loaded
})
```

Node.js

```
fs.readFile('foo.txt', function () {
    // do something after foo.txt
    // being loaded
})
```

RequireJS

```
require(['lodash', 'jquery'],
    function (_, $) {
    // do something with lodash and jQuery
    }
);
```

Nothing wrong with Callback

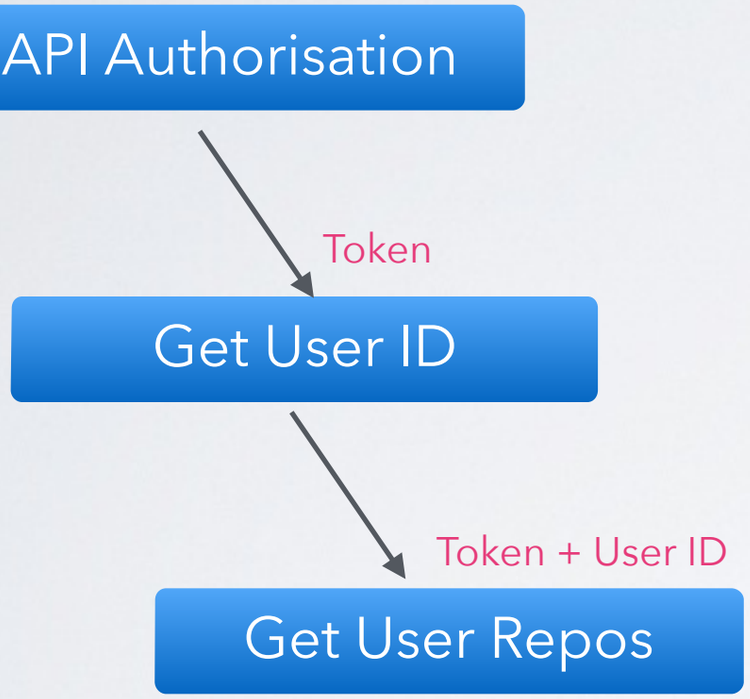
Use it when your scenario is simple

Stackla

Sequencial Requests

Callback Hell

GitHubber#getUserRepos




```
function GitHubber() {}

GitHubber.prototype.getUserRepos = function (name, callback) {

    // Authorisation pass?
    var url = '/api/authorize';
    makeRequest(url, function (data, success) {
        // (Omit) Callback if it fails…
        // Get user ID by name
        url = '/api/getUserInfo/' + name + '?token=' + data.token;
        makeRequest(url, function (data, success) {
            // (Omit) Callback if it fails…
            // Get user's repo by user ID
            url = '/api/getUserRepos?token=…&uid=' + data.uid;
            makeRequest(url, function (data, success) {
                // (Omit) Callback if it fails…
                // Finally success
                callback(data, true);

            });
        });
    });
};
```

Question: How will you refactor it?

Stackla

My Solution

Before understanding Promise

- Break callbacks into **methods with semantic naming**
- Exchange data with **instance variables**
- Make use of **custom events**

I am a big fan of **Custom Events**

Wolfy87/EventEmitter

```
function GitHubber(name) {
    this.name = name;
    this.token = null;
    this.id = null;
    this.repos = [];
    this.steps = ['_authorise', '_getUserInfo', '_getUserRepos'];
}

var proto = {
    _authorise: function () {
        var url = '/api/authorise';
        makeRequest(url, function (data, success) {
            this.token = data.token;
            this.emit('success', ['_authorise']);
        });
    },
    _getUserInfo: function () {
        var url = '/api/getUserInfo/' + this.name +
                  '?token=' + data.token;
        makeRequest(url, function (data, success) {
            this.id = data.id;
            this.emit('success', ['_getUserInfo']);
        });
    },
    _getUserRepos: function () {
        var url = '/api/getRepos/?uid=' + this.id +
                  '?token=' + data.token;
        makeRequest(url, function (data, success) {
            this.repos = data.repos;
            this.emit('success', ['_getUserRepos', this.repos]);
        });
    },
    getUserRepos: function (callback) {
        var that = this;
        that.on('success', function (e, method) {
            var offset = that.steps.indexOf(method);
            if (offset !== that.steps.length - 1) { // Other steps
                that[that.steps[offset + 1]];
            } else { // _getUserRepos
                callback(that.repos);
            }
        });
        that[that.steps[0]]();
    }
};
```

Stackla

My Solution

Before understanding Promise

- Break callbacks into **methods with semantic naming**
- Exchange data with **instance variables**
- Make use of **custom events**

I am a big fan of **Custom Events**

Wolfy87/EventEmitter

```
function GitHubber(name) {
    this.name = name;
    this.token = null;
    this.id = null;
    this.repos = [];
    this.steps = ['_authorise', '_getUserInfo', '_getUserRepos'];
}

var proto = {
    _authorise: function () {
        var url = '/api/authorise';
        makeRequest(url, function (data, success) {
            this.token = data.token;
            this.emit('success', ['_authorise']);
        });
    },
    _getUserInfo: function () {
        var url = '/api/getUserInfo/' + this.name +
                  '?token=' + data.token;
        makeRequest(url, function (data, success) {
            this.id = data.id;
            this.emit('success', ['_getUserInfo']);
        });
    },
    _getUserRepos: function () {
        var url = '/api/getRepos/?uid=' + this.id +
                  '?token=' + data.token;
        makeRequest(url, function (data, success) {
            this.repos = data.repos;
            this.emit('success', ['_getUserRepos']);
        });
    },
    getUserRepos: function (callback) {
        var that = this;
        that.on('success', function (e, method) {
            var offset = that.steps.indexOf(method);
            if (offset !== that.steps.length - 1) { // Other steps
                that[that.steps[offset + 1]];
            } else { // _getUserRepos
                callback(that.repos);
            }
        });
        that[that.steps[0]]();
    }
};
```

Better but still not straightforward

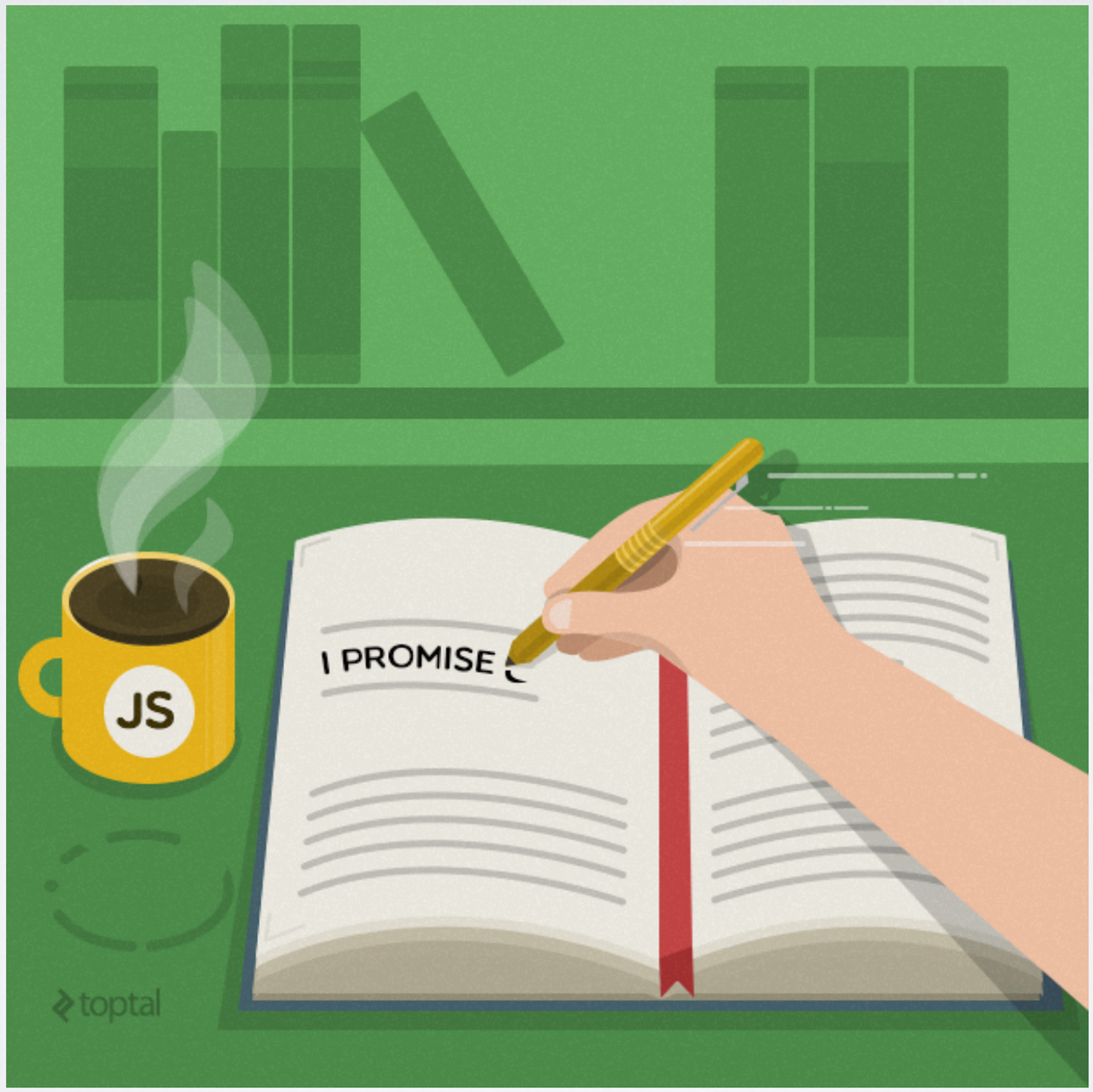
Need read carefully to understand the trick

ex. sequence, error handling, and parallel events

Stackla

JavaScript Promise

Developer's Wonderland



Stackla

PROMISE

NOT another JavaScript framework

- A Programming Pattern
- Specialise on **Asynchronous Code**
- Better **Maintainability**
- Easier for **Scaling**

Stackla

.then

Create A Promise

Returns promise immediately

```
getUserRepos: function () {
    that.repoDeferred = new $.Deferred();
    that.asyncTasks()

    return that.repoDeferred.promise();
},
```

"This task may take a while"

"Our workers will do all tasks for you"

"Keep the ticket for now. I promise you will get a fulfilled or rejected result"

- **Pending**
- **Fulfilled**: that.repoDeferred.**resolve**(data.repos)
- **Rejected**: that.repoDeferred.**reject**('service unavailable')

Stackla

Use Promise

Promise is still callback

```
gitHubber.getUserRepos()
  .then(function (repos) {})
  .catch(function (msg) {});
```

Chain-able **fulfilled** callback

Rejected callback

- **.then(fnFulfilled, fnRejected)**
- **.done(fnFulfilled)**
- **.fail(fnRejected)**

Stackla

Batch Promise

Execute multiple promises together

```
var deferreds = [
    gitHubber.getUserRepos(),
    gitHubber.getUserProfile(),
    gitHubber.getOrgaizations()
];

$.when(deferreds)
    .done(fnFulfilled)
    .fail(fnRejected);
```

Promises Array

All succeeds

One or more fails

Stackla

1st Refactoring

Not attractive… :(

```
function GitHubber(name) {
    this.name = name;
    this.repos = [];
    this.deferreds = {};
}

var proto = {
    _authorise: function () {
        var that = this,
            url = '/api/authorise';

        that.deferreds._authorise = $.Deferreds();
        $.ajax(url, function (data) {
            that.deferreds._authorise.resolve(data);
        });
        return that.deferreds._authorise.promise();
    },
    _getUserInfo: function () {
        var that = this,
            url = '/api/getUserInfo/' + this.name + '?token=' + data.token;

        that.deferreds._getUserInfo = $.Deferreds();
        $.ajax(url, function (data) {
            that.deferreds._getUserInfo.resolve(data.id);
        });
        return that.deferreds._getUserInfo.promise();
    },
    _getUserRepos: function () {
        var that = this,
            url = '/api/getRepos/?uid=' + this.id + '?token=' + data.token;

        that.deferreds._getUserRepos = $.Deferreds();
        $.ajax(url, function (data) {
            that.deferreds._getUserRepos.resolve(data.repos);
        });
        return that.deferreds._getUserRepos.promise();
```

2nd: jQuery Promises

$.ajax() is also a promise object!

```
function GitHubber (name) {
    this.name = name;
    this.token = null;
    that.repos = [];
}

var proto = {
    getUserRepos: function (callback) {
        var that = this,
            deferred = $.Deferred();

        if (that.repos.length) {
            deferred.resolve(that.repos);
            return;
        }

        $.ajax('/api/authorise')
            .then(function (data) {
                that.token = data.token;
                return $.ajax('/api/getUserInfo/' + that.name +'?token=' + data.token);
            })
            .then(function (data) {
                return $.ajax('/api/getRepos/?uid=' + data.uid + '?token=' + that.token);
            })
            .then(function (data) {
                that.repos = data.repos;
                deferred.resolve(data.repos);
            });

        return deferred.promise();
    }
};
```

$.ajax

$.when

$.getJSON

2nd: jQuery Promises

$.ajax() is also a promise object!

```
function GitHubber (name) {
    this.name = name;
    this.token = null;
    that.repos = [];
}

var proto = {
    getUserRepos: function (callback) {
        var that = this,
            deferred = $.Deferred();

        if (that.repos.length) {
            deferred.resolve(that.repos);
            return;
        }

        $.ajax('/api/authorise')
            .then(function (data) {
                that.token = data.token;
                return $.ajax('/api/getUserInfo/' + that.name +'?token=' + data.token);
            })
            .then(function (data) {
                return $.ajax('/api/getRepos/?uid=' + data.uid + '?token=' + that.token);
            })
            .then(function (data) {
                that.repos = data.repos;
                deferred.resolve(data.repos);
            });

        return deferred.promise();
    }
};
```

$.ajax

$.when

$.getJSON

You can reduce huge amount of code by chaining & wrapping promise object properly

Stackla

Promise v.s. Callback

Why Promise?

Promise

```
var promise = $.ajax(url);
promise.done(callback);
```

Callback

```
$.ajax(url, callback)
```

“First-class API for asynchronous tasks”

- **Portability** - async task must be fulfilled or rejected in delegated methods.
- **Consistency** - .resolve(), .reject() , .then(), .done(), .catch(), rejected, fulfilled, pending
- **Chaining** - .then() makes sequential tasks easier to execute
- **Straightforward** - .then() makes our code easier to read

Stackla

Scalability

Using JavaScript Promises to Reason About User Interaction

https://robots.thoughtbot.com/using-javascript-promises-to-reason-about-user-interaction

```
UserSession.signIn()
    .then(this._promisePurchase(video))
```

Abstracted **Session Checking**, **Login Popup**, **Validation**, **Video Purchasing** and **Watching Video** with Promise!

Stackla

With Promise…

- We defines sequence in a very straightforward way (.then)
- Built-in error handling API (.fail)
- Batch execute parallel tasks easily (Promise.all)

Solved a lot of async design issues

Stackla

PROMISE IN THE WILD

Environments and Libraries for Promise

Standard Spec: **Promises/A+**

Stackla

Promise in Browsers

Current aligned | Usage relative | Show all

IE	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
		31					4.1	
8		36					4.3	
9	31	37	7				4.4	
10	37	42	7.1		7.1		4.4.4	
11	38	43	8	29	8.3	8	40	42
Edge	39	44		30				
	40	45		31				
	41	46						

All Modern browsers support Promise

except IE 11 and all its earlier versions

Stackla

Promise & jQuery

$.Deferred

$.when

jQuery's **Deferreds** aren't **Promise/A+** compliant.
Please avoid to use if you want to use Promise extensively.

Stackla

Promise in Node.js

Node.js added native promise
in stable version **0.12**

… comes with **memory leak**

https://github.com/promises-aplus/promises-spec/issues/157

Stackla

Libraries

For both Browser and Node.js

- **Q.js**

  A tool for creating and composing asynchronous promises in JavaScript

- **RSVP.js**

  A lightweight library that provides tools for organising asynchronous code

- **when.js**

  A solid, fast Promises/A+ and when() implementation, plus other async goodies.

- **bluebird (most popular)**

  Bluebird is a full featured promise library with unmatched performance.

Libraries

For both Browser and Node.js

- **Q.js**
  A tool for creating and composing asynchronous promises in JavaScript
- **RSVP.js**
  A lightweight library that provides tools for organising asynchronous code
- **when.js**
  A solid, fast Promises/A+ and when() implementation, plus other async goodies.
- **bluebird (most popular)**
  Bluebird is a full featured promise library with unmatched performance.

Currently you probably need library for polyfills

Use jQuery Deferred with awareness

Q & A

- Promise - A Programming Pattern
- Specialise on **Asynchronous Code**
- Better **Maintainability**
- Easier for **Scaling**

“First-class API for asynchronous tasks”

Stackla