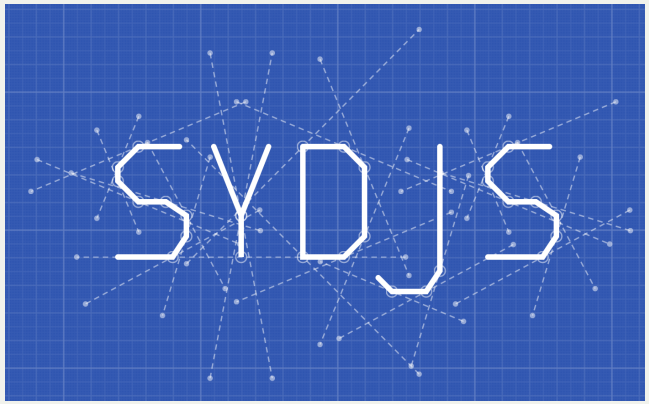
Combining Generators with Promises and Channels

"Don't combine Generators with Promises, combine them with Channels!"

David Nolen



Everyday's jQuery

JQuery AJAX Innocent function

```
function foo() {
    var jqXHR = $.ajax({
        //...
        async: false
    });
    return jqXHR.responseText;
}
```

JQuery AJAX Innocent function

```
function foo() {
    var jqXHR = $.ajax({
        //...
        async: false
    });
    return jqXHR.responseText;
}
```

JQuery AJAX Sync or Async?

```
function foo() {
    var jqXHR = $.ajax({
        //...
        async: false
    });
    return jqXHR.responseText;
}
```

JQuery AJAX Sync or Async?

```
function foo() {
    var jqXHR = $.a
        //...
        async: false
    });
    return jqXHR.responseText;
}
```

SJAX

JQuery AJAX Sync or Async?

```
function
    var
        async: false
    });
    return jqXHR.responseText;
}
```

async (default: `true`)
Type: Boolean
By default, all requests are sent asynchronously (i.e. this is set to `true` by default). If you need synchronous requests, set this option to `false`. Cross-domain requests and `dataType: "jsonp"` requests do not support synchronous operation. Note that synchronous requests may temporarily lock the browser, disabling any actions while the request is active. **As of jQuery 1.8, the use of `async: false` with jqXHR (`$.Deferred`) is deprecated; you must use the success/error/complete callback options instead of the corresponding methods of the jqXHR object such as `jqXHR.done()` or the deprecated `jqXHR.success()`.**

JQuery AJAX Sync or Async?

```
function
    var
                                                t;
}
```

async (default: `true`)
Type: Boolean
By default, all requests are sent asynchronously (i.e. this is set to `true` by default). If you need synchronous requests, set this option to `false`. Cross-domain requests and `dataType: "jsonp"` requests do not support synchronous operation. Note that synchronous requests may temporarily lock the browser, disabling any actions while the request is active. **As of jQuery 1.8, the use of `async: false` with jqXHR (`$.Deferred`) is deprecated; you must use the success/error/complete callback options instead of the corresponding methods of the jqXHR object such as `jqXHR.done()` or the deprecated `jqXHR.success()`.**

jQuery 1.8 Released

Posted on August 9, 2012 by dmethvin

Parallelism
Concurrency

Parallelism in one GIF

Parallelism in one GIF

http://goo.gl/0lmTG6

Parallelism in one GIF

> "Parallelism is about <u>doing</u> lots of things at once"
>
> -- Rob Pike

Concurrency in one GIF

Concurrency in one GIF

http://goo.gl/nzEIOe

Concurrency in one GIF

> "Concurrency is about <u>dealing</u> with lots of things at once"
>
> -- Rob Pike

Concurrency using Generators and Promises

Promises Everyday

Promises Everyday

jQuery
write less, do more.

Download | API Documentation | Blog

.promise()

Categories: Deferred Object

.promise([type] [, target])

Description: *Return a Promise object t*
queued or not, have finished.

.promise([type] [, target])

type (default: `fx`)
Type: String
The type of queue that needs to be ob

Promises Everyday

jQuery
write less, do more.

Download | API Documentation | Blog

.promise()

Categories: Deferred Object

.promise([type] [, target])

Description: *Return a Promise object t*
queued or not, have finished.

.promise([type] [, target])

type (default: `fx`)
Type: String
The type of queue that needs to be ob

ANGULARJS | Home | Learn | De

API Reference / ng / $q

$q

service in module ng

Description

A promise/deferred implementation inspired by Kris Kowal's Q.

Promises Everyday

jQuery
write less, do more.

Download | API Documentation | Blog

.promise()

Categories: Deferred Object

.promise([type] [, target])

Description: *Return a Promise object t*
queued or not, have finished.

.promise([type] [, target])

type (default: `fx`)
Type: String
The type of queue that needs to be ob

then

when.js

When.js is cujoJS's lightweight Promises/A+ and `when()` implementation

ANGULARJS | Home | Learn | De

API Reference / ng / $q

$q

service in module ng

Description

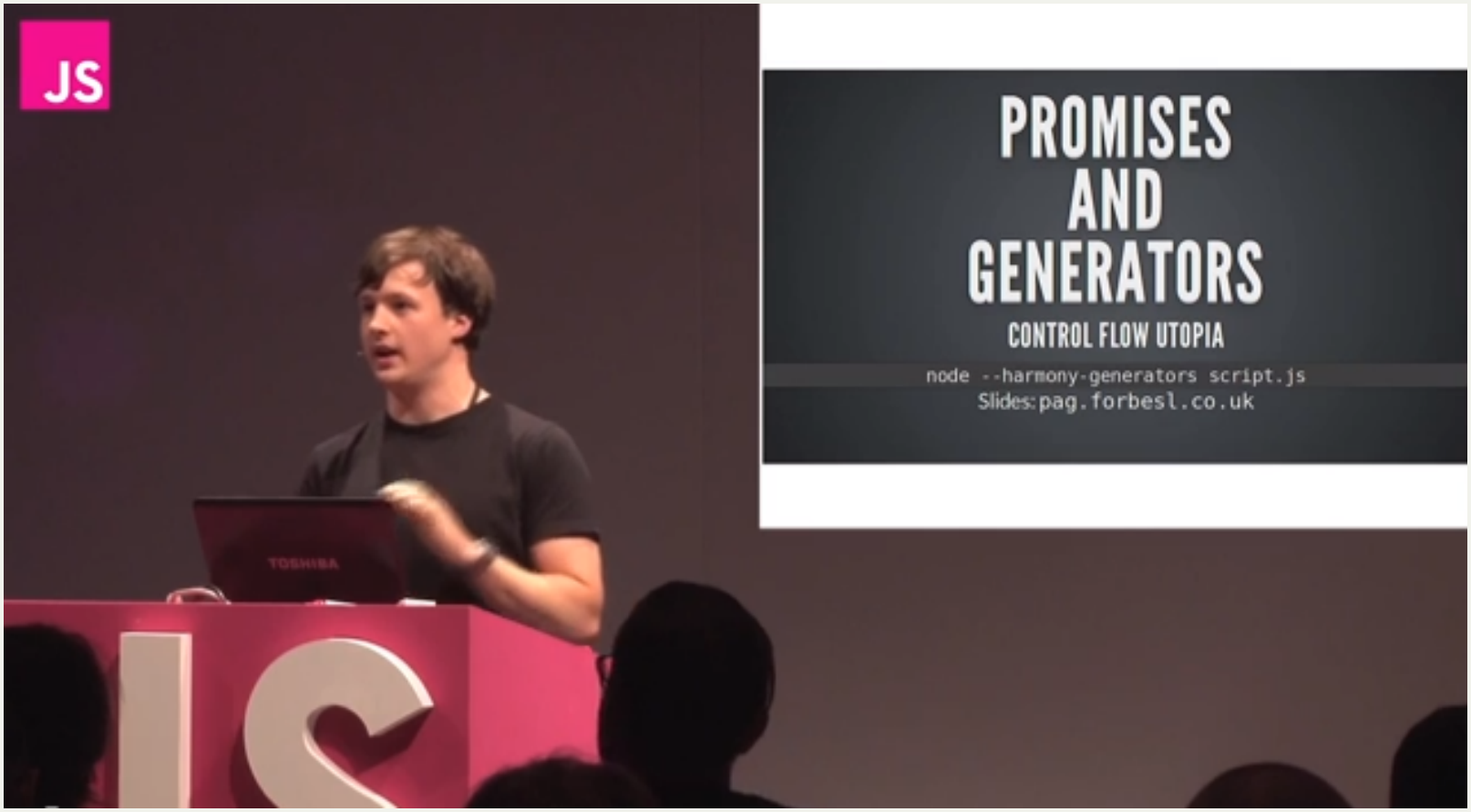
A promise/deferred implementation inspired by Kris Kowal's Q.

Promises Are awesome

- Cleaner method signatures
- Uniform return/error semantics
- Easy composition
- Easy sequential/parallel join
- Always async
- Exception-style error bubbling

credit: http://www.slideshare.net/domenicdenicola/callbacks-promises-and-coroutines-oh-my-the-evolution-of-asynchronicity-in-javascript

Promises and Generators



Forbes Lindesay: Promises and Generators: control flow utopia -- JSConf EU 2013

http://www.youtube.com/watch?v=qbKWsbJ76-s

http://pag.forbeslindesay.co.uk/#/

Concurrency using Generator and Channels

Channels Go-style concurrency

JavaScript Weekly

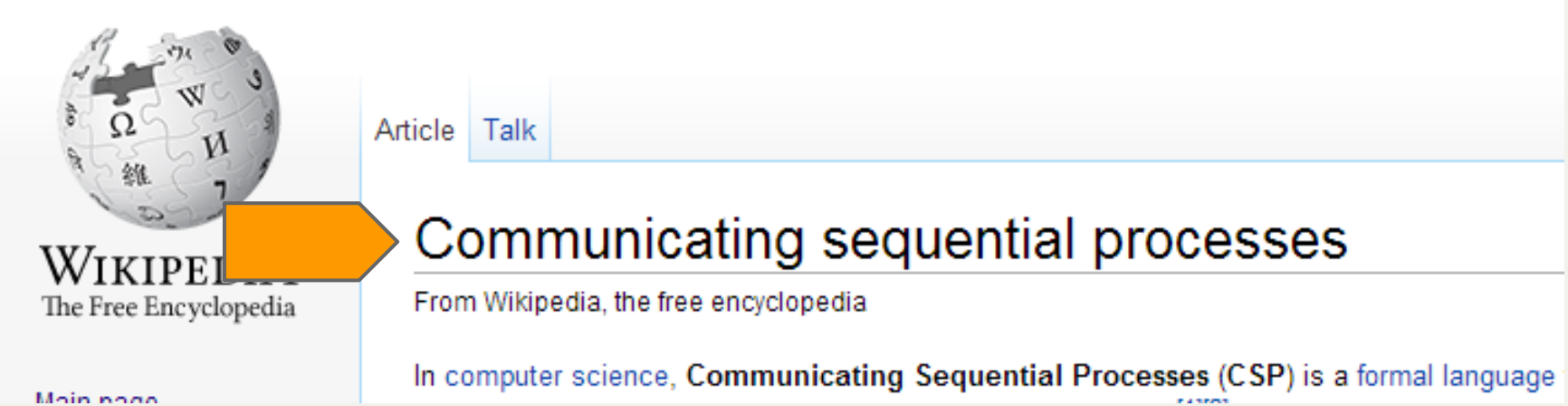
Curated JavaScript news every Friday from Peter Cooper

Issue #145 - August 30, 2013

ES6 Generators Deliver Go-Style Concurrency

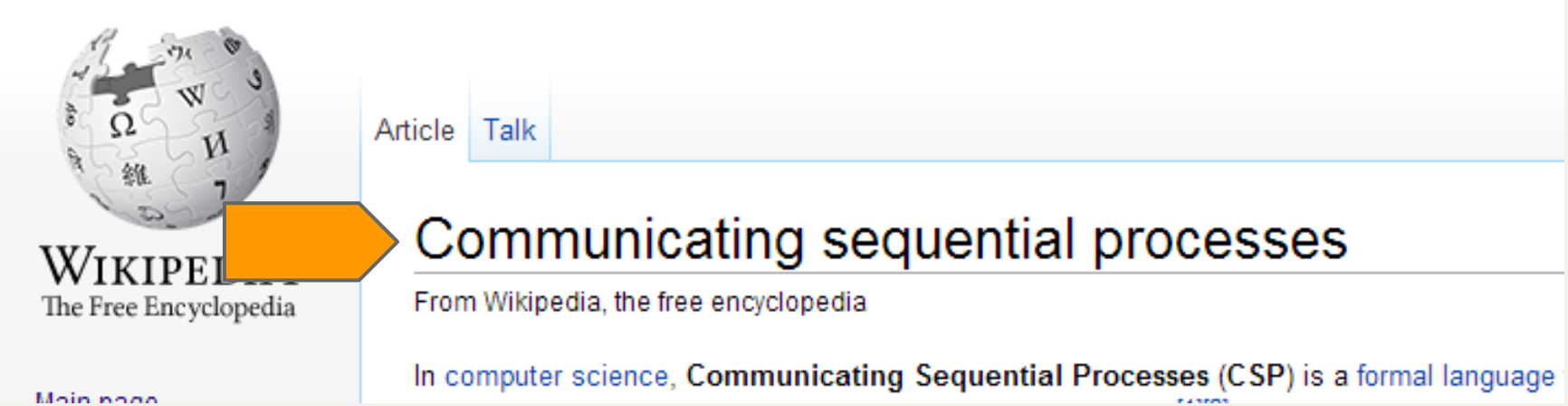
David Nolen shows that one can use ECMAScript 6's generators t
is based on Communicating Sequential Processes) to JavaScript.

Channels defined by CSP



http://en.wikipedia.org/wiki/Communicating_sequential_processes

Channels defined by CSP



"formal language for describing patterns of interaction in concurrent systems"

http://en.wikipedia.org/wiki/Communicating_sequential_processes

Channels CSP implementations

- occam was the first language implementing a CSP model.
- Ease programming language combines the process constructs of CSP with logically shared data
- JCSP is a blending of CSP and occam concepts in a Java thread support API.
- XC is a language developed by XMOS which was heavily influenced by CSP and occam
- Limbo is a language that implements concurrency inside the Inferno operating system, in a style i
- Plan 9 from Bell Labs and Plan 9 from User Space include the libthread library which allows the u
- VerilogCSP is a set of macros added to Verilog HDL to support Communicating Sequential Proce
- SystemVerilogCSP[25] is a package for SystemVerilog that enables abstract CSP-like communic
- Trace monoid and history monoid, the mathematical formalism of which CSP is an example.
- Trace theory, the general theory of traces.
- Go is a programming language by Google incorporating ideas from CSP.[4][26]
- Clojure's Core.async is a library for the clojure programming language based on CSP principles.
- Joyce is a programming language based on the principles of CSP, developed by Brinch Hansen a
- SuperPascal is a programming language also developed by Brinch Hansen, influenced by CSP ar
- Ada implements features of CSP such as the rendezvous.
- DirectShow is the video framework inside DirectX, it uses the CSP concepts to implement the auc

Channels Go-style concurrency

JavaScript Weekly

Curated JavaScript news every Friday from Peter Cooper

Issue #145 - August 30, 2013

ES6 Generators Deliver Go-Style Concurrency

David Nolen shows that one can use ECMAScript 6's generators t
is based on Communicating Sequential Processes) to JavaScript.

Concurrency using Generator and Channels

Producer

Channels Producer

```
var c = [];

go(function* () {
  for(var i = 0; i < 10; i++) {
    yield put(c, i);
    console.log("process one put", i);
  }
  yield put(c, null);
});
```

Channels Producer

```
var c = [];

go(function* () {
  for(var i = 0; i < 10; i++) {
    yield put(c, i);
    console.log("process one put", i);
  }
  yield put(c, null);
});
```

Channels Producer

```
var c = [];


go(function* () {
  for(var i = 0; i < 10; i++) {
    yield put(c, i);
    console.log("process one put", i);
  }
  yield put(c, null);
});
```

Concurrency using Generator and Channels

Consumer

Channels Consumer

```
go(function* () {
  while(true) {
    var val = yield take(c);
    if(val == null) {
      break;
    } else {
      console.log("process two took", val);
    }
  }
});
```

Channels Consumer

```
go(function* () {
  while(true) {
    var val = yield take(c);
    if(val == null) {
      break;
    } else {
      console.log("process two took", val);
    }
  }
});
```

I Want to use

Javascript NOW

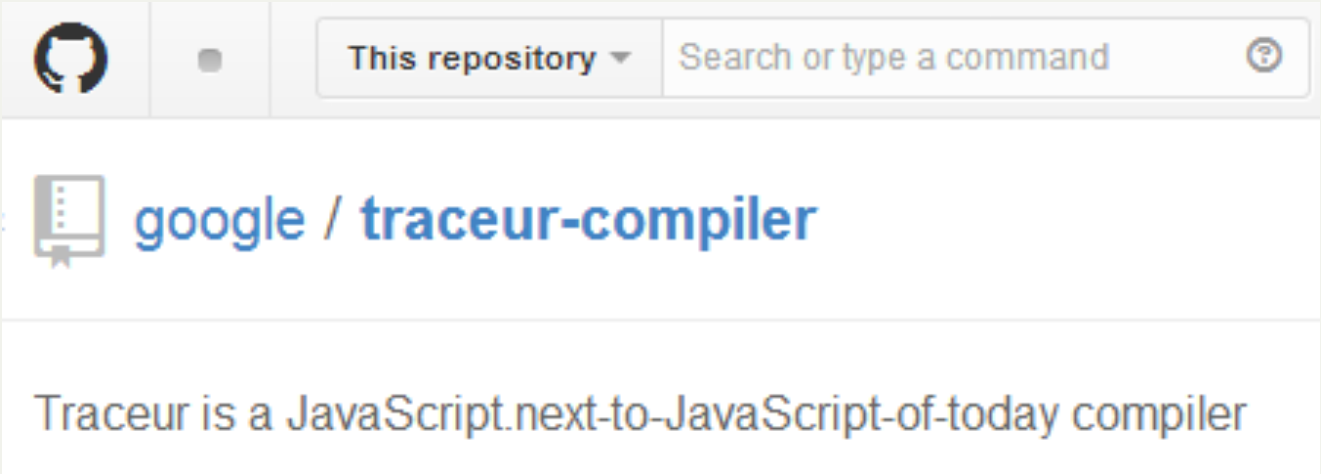
ES6 with Node

```
$ node --v8-options | grep harm
  --harmony_typeof (enable harmony semantics for typeof)
  --harmony_scoping (enable harmony block scoping)
  --harmony_modules (enable harmony modules (implies block
  --harmony_symbols (enable harmony symbols (a.k.a. privat
  --harmony_proxies (enable harmony proxies)
  --harmony_collections (enable harmony collections (sets,
  --harmony_observation (enable harmony object observation
  --harmony_typed_arrays (enable harmony typed arrays)
  --harmony_array_buffer (enable harmony array buffer)
  --harmony_generators (enable harmony generators)
  --harmony_iteration (enable harmony iteration (for-of))
```

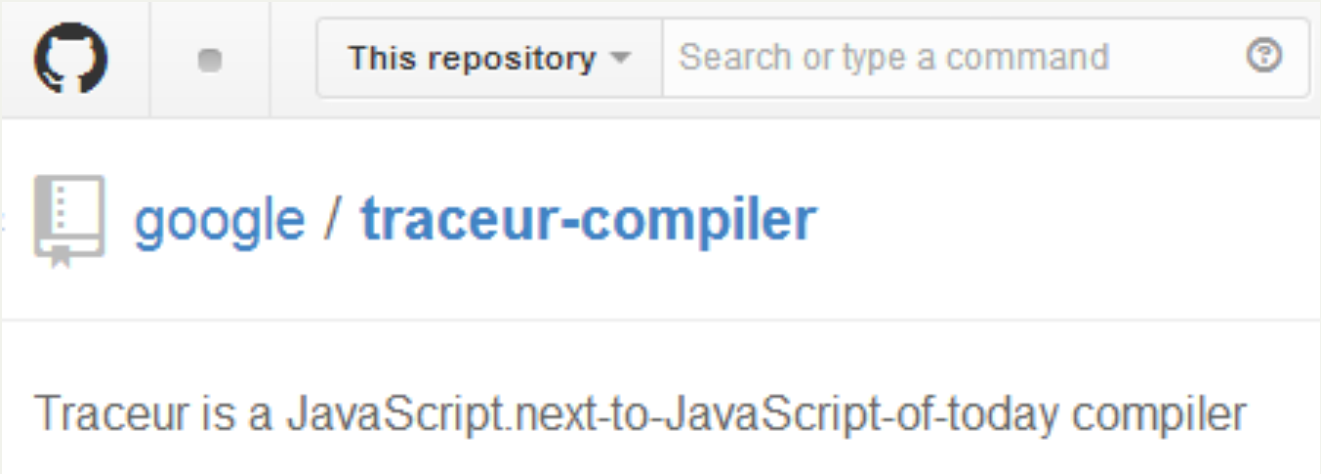
ES6 In the browser

Feature name		Current browser
arrow functions	c	No
class	c	No
let	c	No
const	c	Yes
Generators (yield)		No

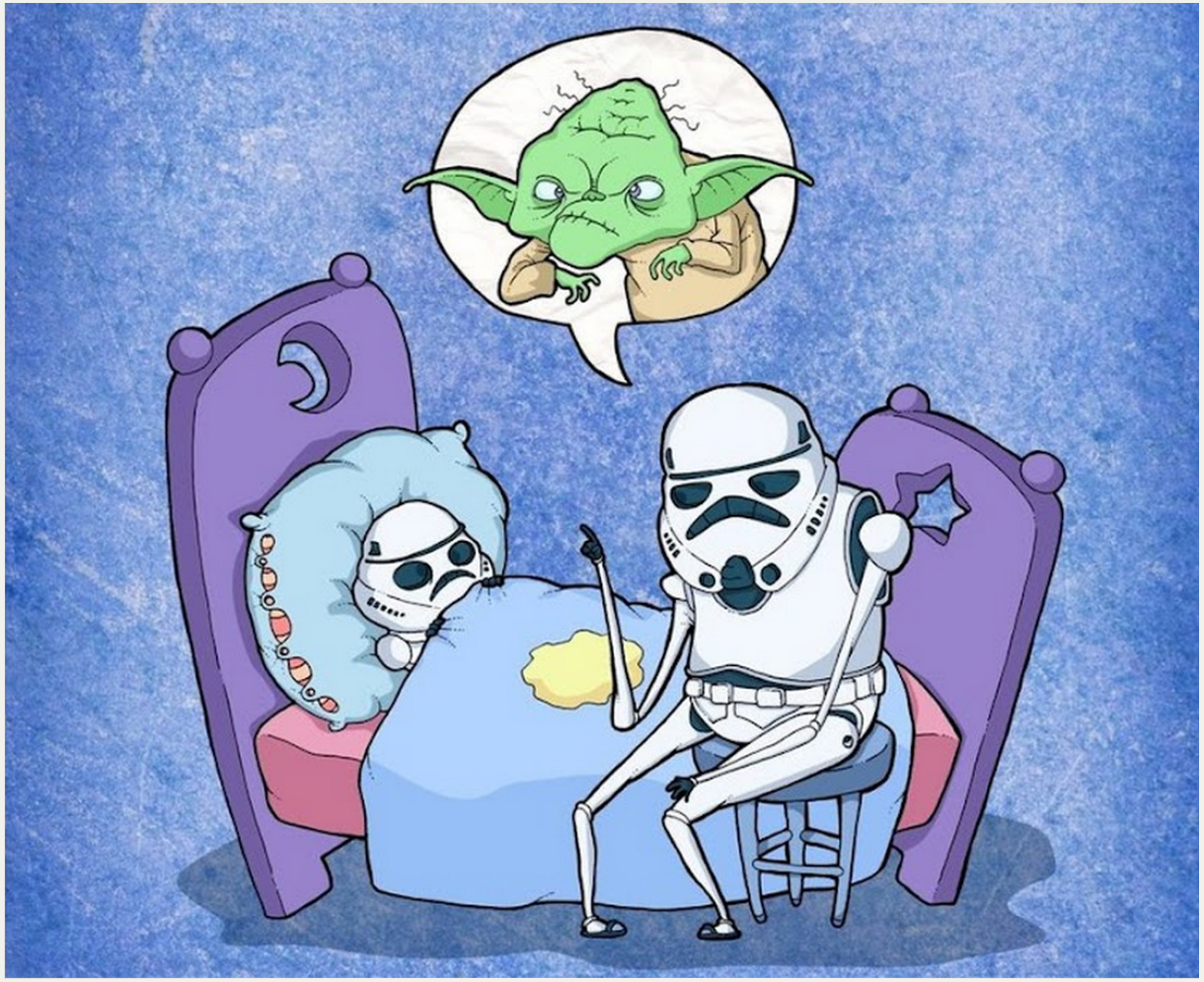
ES6 In the browser



ES6 In the browser



Questions?



Twitter @filippovitale

BitBucket: filippovitale

GitHub: filippovitale

Thank you!

Twitter @filippovitale

BitBucket / GitHub: filippovitale

Resources in random order

- http://swannodette.github.io/2013/08/02/100000-processes/
- http://swannodette.github.io/2013/08/23/make-no-promises/
- http://swannodette.github.io/2013/08/24/es6-generators-and-csp/
- http://concur.rspace.googlecode.com/hg/talk/concur.html#title-slide
- http://stackoverflow.com/q/1050222/81444
- http://kangax.github.io/es5-compat-table/es6/
- http://wiki.ecmascript.org/doku.php?id=harmony:generators
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Iterators
- http://www.slideshare.net/domenicdenicola/es6-the-awesome-parts
- http://www.youtube.com/watch?v=qbKWsbJ76-s
- http://pag.forbeslindesay.co.uk/#/
- http://www.slideshare.net/domenicdenicola/callbacks-promises-and-cor

fin

Generators

FTW

Generators Example

```
function zeroOneTwo() {
  return [0, 1, 2];
}

var array = zeroOneTwo();

for (var i of array) {
  console.log(i);
}
```

Generators Example

```
function* zeroOneTwo() {
  yield 0;
  yield 1;
  yield 2;
}

var generator = zeroOneTwo();

for (var i of generator) {
  console.log(i);
}
```

Generators Example

```
generator.next(); // { value: 0, done: false }
generator.next(); // { value: 1, done: false }
generator.next(); // { value: 2, done: false }
generator.next(); // { value: undefined, done: true }
```

Generators Example .next(value)

```
function* demo() {
  var res = yield 10
  assert(res === 32)
  return 42
}
var d = demo()
var resA = d.next()
// => {value: 10, done: false}
var resB = d.next(32)
// => {value: 42, done: true}
// d.next() - THROWS!!!
```

Generators Async

Generators let us turn

```
function get(filename){
  return readJSON('left.json').then(function (left){
    return readJSON('right.json').then(function (right){
      return {left: left, right: right}
    })
  })
}
```

INTO

```
var get = async(function *(){
  var left = yield readJSON('left.json')
  var right = yield readJSON('right.json')
  return {left: left, right: right}
})
```

Promises and Generators

```
function async(makeGenerator){
  return function (){
    var generator = makeGenerator.apply(this, arguments)

    function handle(result){ // { done: [Boolean], value
      if (result.done) return result.value

      return result.value.then(function (res){
        return handle(generator.next(res))
      }, function (err){
        return handle(generator.throw(err))
      })
    }

    return handle(generator.next())
  }
}
```

Forbes Lindesay: Promises and Generators: control flow utopia -- JSConf EU 2013

http://www.youtube.com/watch?v=qbKWsbJ76-s

http://pag.forbeslindesay.co.uk/#/

Recycle Bin

Actor Model Compared to CSP

Comparison with the Actor Model [edit]

In as much as it is concerned with concurrent processes that exchange messages, the Actor model is broadly similar to CSP. However, the two models make some fundamentally different choices with regard to the primitives they provide:

- CSP processes are anonymous, while actors have identities.
- CSP message-passing fundamentally involves a rendezvous between the processes involved in sending and receiving the message, i.e. the sender cannot transmit a message until the receiver is ready to accept it. In contrast, message-passing in actor systems is fundamentally asynchronous, i.e. message transmission and reception do not have to happen at same time, and senders may transmit messages before receivers are ready to accept them. These approaches may be considered duals of each other, in the sense that rendezvous-based systems can be used to construct buffered communications that behave as asynchronous messaging systems, while asynchronous systems can be used to construct rendezvous-style communications by using a message/acknowledgement protocol to synchronize senders and receivers.
- CSP uses explicit channels for message passing, whereas actor systems transmit messages to named destination actors. These approaches may also be considered duals of each other, in the sense that processes receiving through a single channel effectively have an identity corresponding to that channel, while the name-based coupling between actors may be broken by constructing actors that behave as channels.

XXX YYY

```
console.log("1");

$.get("/echo/2", function (result) {
  console.log(result);
});

console.log("3");
```

XXX YYY

```
console.log("1");

$.get("/echo/2", function (result) {
  console.log(result);
});

console.log("3");

// 1, 3, 2
```

Recycle Bin

“Callbacks are OK for simple operations, but force us into continuation passing style”