Globant

we are ready

Front End Engineer

Introduction to RequireJS/AMD

TOPICS

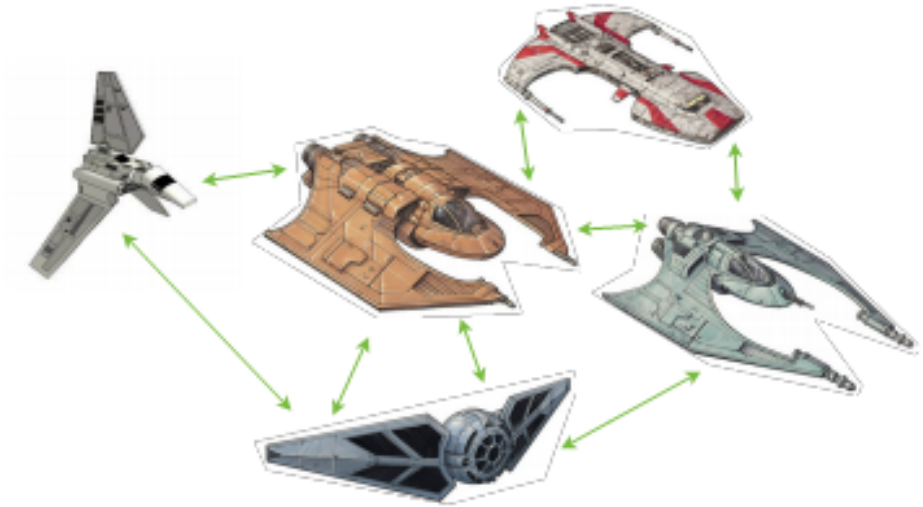
Module Pattern.
AMD
CommonJS
RequireJS

LET US BEGIN

In the beginning, empires generally start out small.

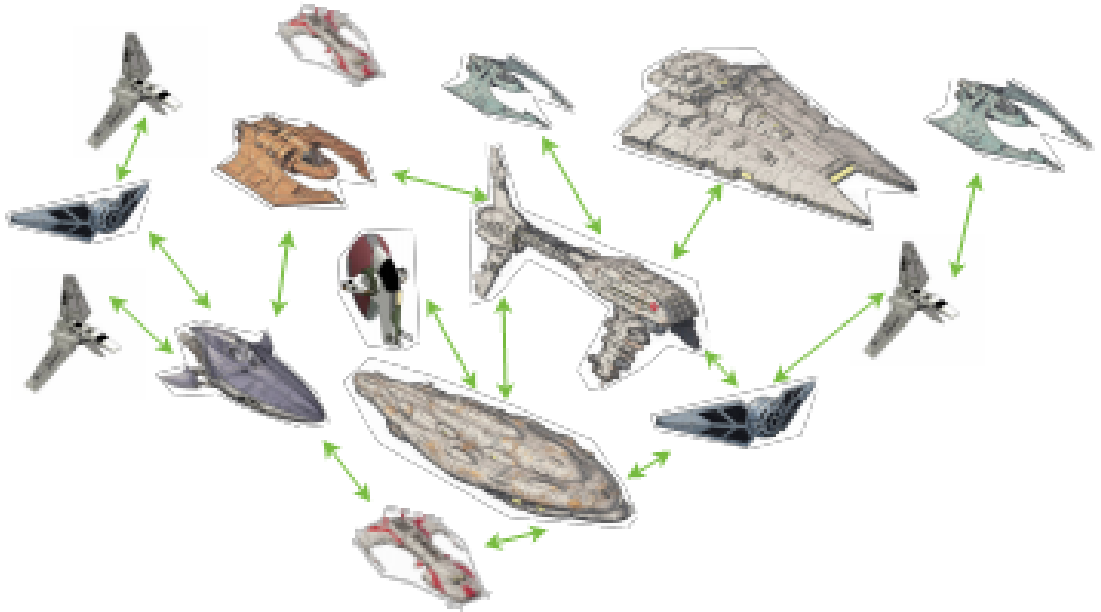
LET US BEGIN

With time, we add a few more ships to our fleet and it begins to grow



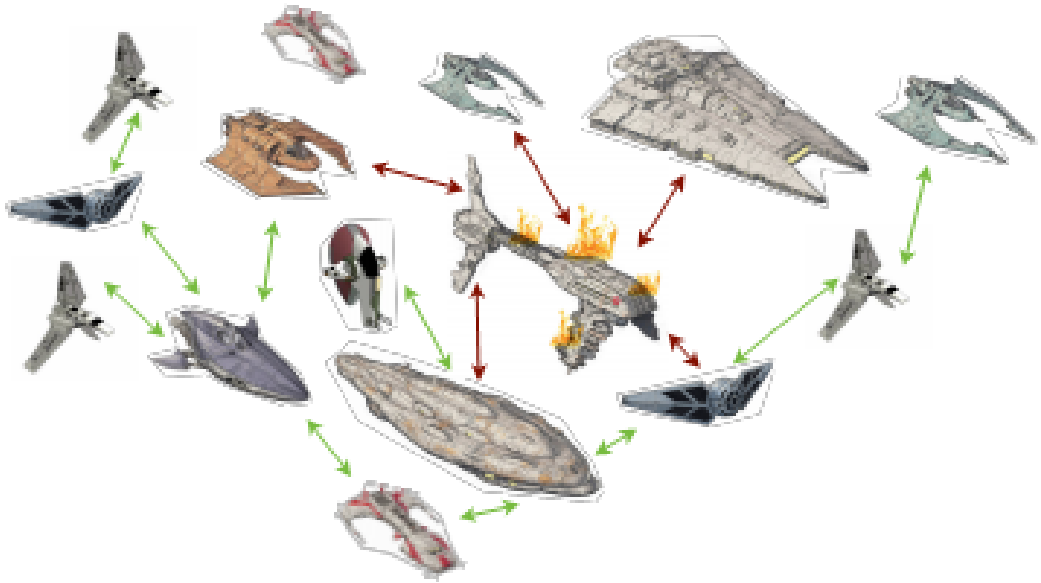
LET US BEGIN

Soon enough, we have so many ships it becomes difficult to handle communication and organization.



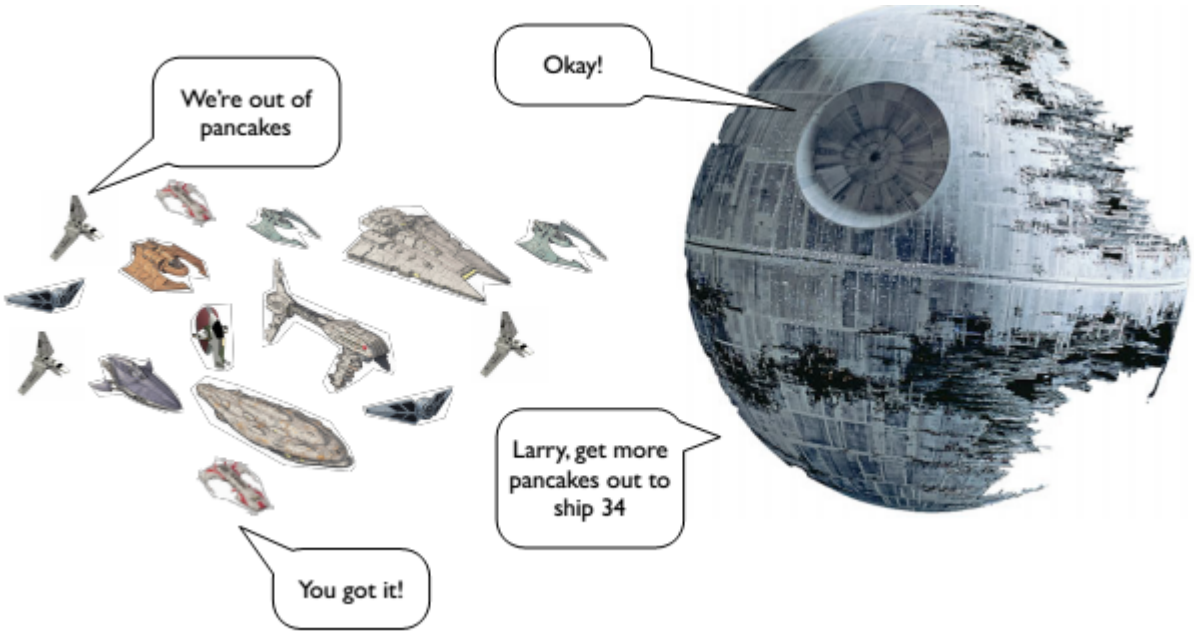
LET US BEGIN

What if all of this grinds to a halt because a ship goes offline? Can everything keep on functioning?



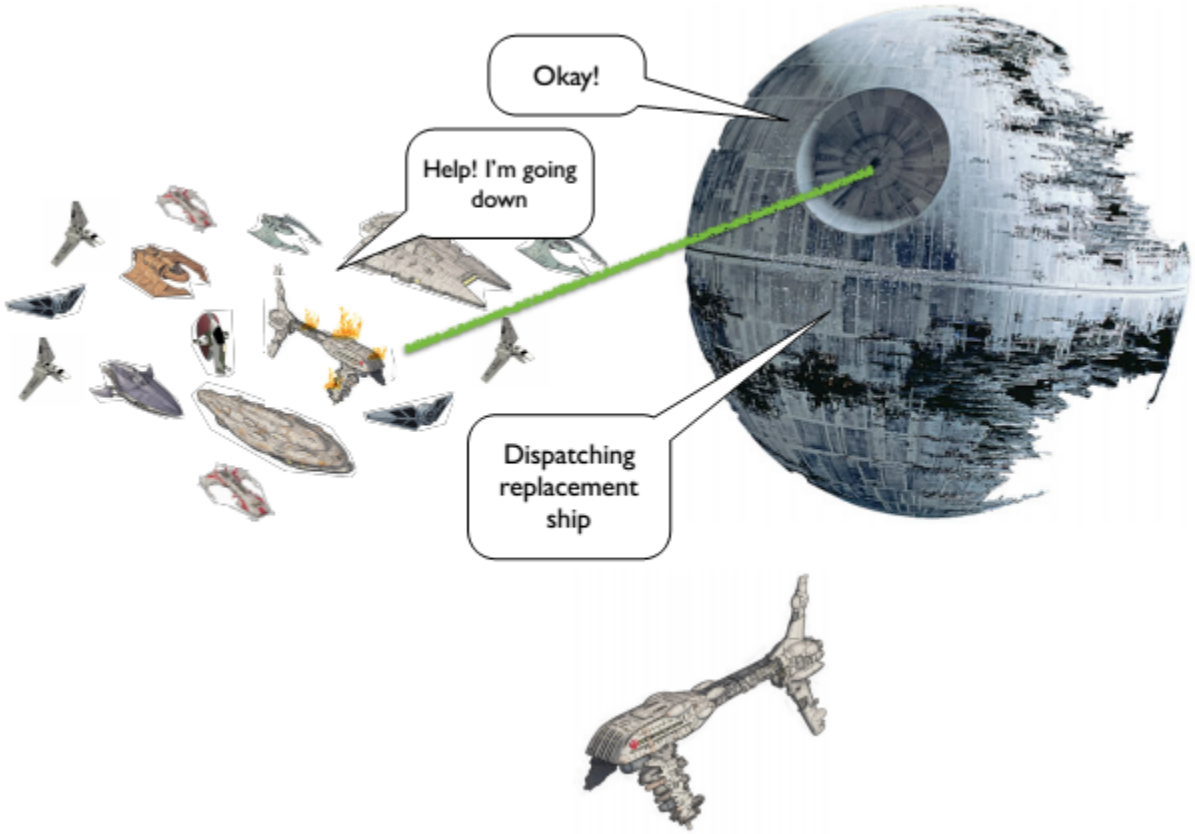
LET US BEGIN

We can introduce a central way of controlling this chaos and solving these problems e.g. the Death Star.



LET US BEGIN

If a ship goes down, the Death Star can respond and react accordingly. e.g. Get rid of the old ship, send a replacement



LET US BEGIN

Think about the future. You should be able to change ‘death stars’ if you find something better..

PATTERNS: A NEW HOPE

The Tools For Our Empire.

These make the architecture proposed possible.

Design Patterns

JavaScript

Scalable Application Architecture

PATTERNS: A NEW HOPE

We're Individuals

"You have your way. I have my way. As for the right way, the correct way, and the only way, it does not exist."

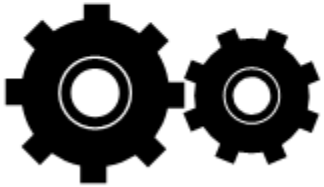
- Friedrich Nietzsche

PATTERNS: A NEW HOPE

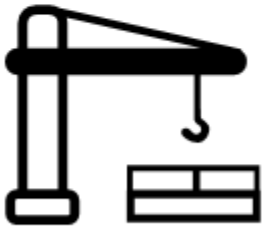
We all do things differently

Each of us have preferences for how we approach..

Solving problems



Structuring solutions



Solving scalability

PATTERNS: A NEW HOPE

Great but can lead to..

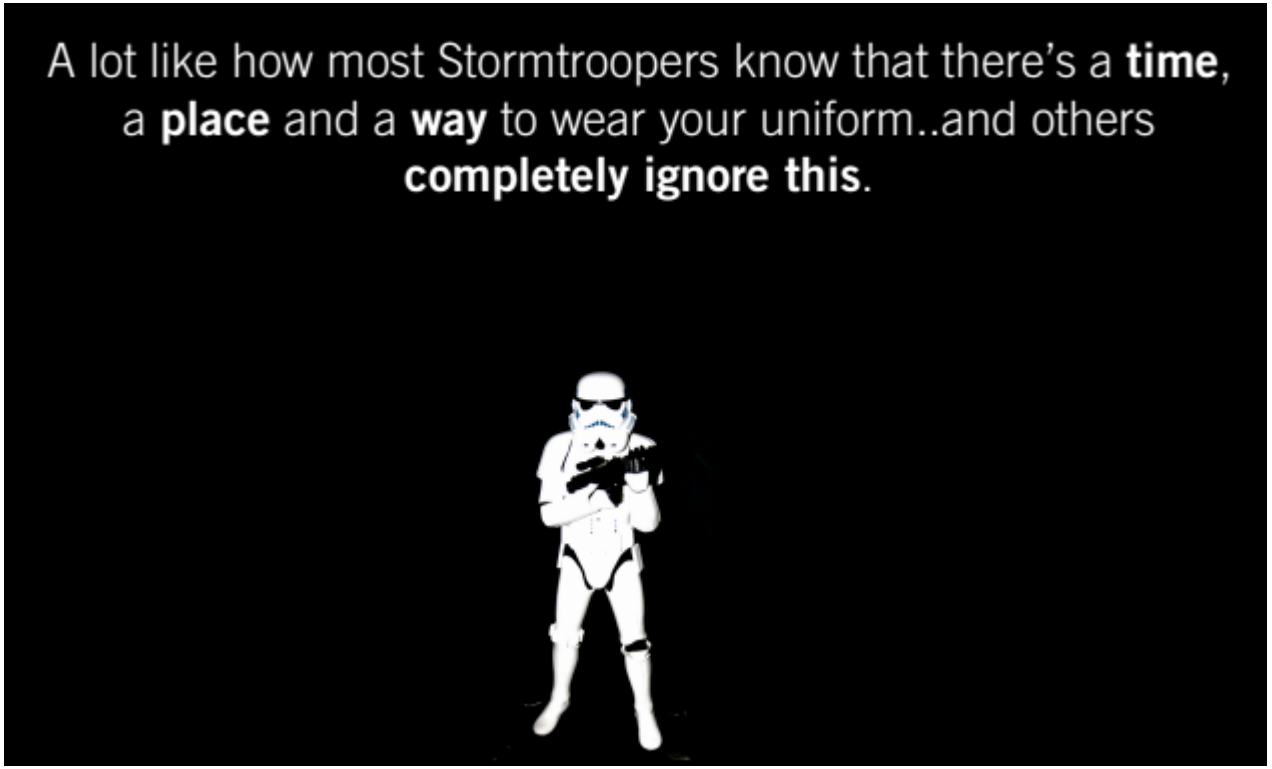
serious problems when working on code to be used by others.

Inconsistent solutions

Inconsistent architecture

Difficult refactoring

PATTERNS: A NEW HOPE



DESIGNING PATTERNS

Reusable solutions that can be applied to commonly occurring problems in software design and architecture.

"We search for some kind of harmony between two intangibles: a form we have not yet designed and a context we cannot properly describe'
- Christopher Alexander, the father of design patterns.

DESIGNING PATTERNS

They're proven

Patterns are generally proven to have successfully solved problems in the past.

Solid

Reliable approaches

Reflect experience

Represent insights

DESIGNING PATTERNS

They're reusable

Patterns can be picked up, improved and adapted without great effort.

Out-of-the-box solutions

Incredibly flexible

Easily adapted

DESIGNING PATTERNS

They're expressive

Patterns provide us a means to describing approaches or structures.

Common vocabulary for expressing solutions elegantly.

Easier than describing syntax and semantics

Problem agnostic

DESIGNING PATTERNS

They offer value

Patterns genuinely can help avoid some of the common pitfalls of development.

Prevent minor issues
that can cause

Major problems
later down the line

THE JAVASCRIPT STRIKES BACK

JAVASCRIPT PATTERNS

Now, you can see:

01 The Module Pattern

http://youtu.be/qrjBYPp0dak

MODULE PATTERN

Stepping stone: IIFE

Immediately invoked function expressions (or self-executing anonymous functions)

```
(function() {
   // code to be immediately invoked
}()); // Crockford recommend this way

(function() {
   // code to be immediately invoked
})(); // This is just as valid


(function( window, document, undefined ){
    //code to be immediately invoked
})( this, this.document);

(function( global, undefined ){
    //code to be immediately invoked
})( this );
```

MODULE PATTERN

This is great, but..

MODULE PATTERN

Privacy In JavaScript

There isn't a *true* sense of it in JavaScript.

No Access Modifiers

Variables & Methods
can't be '**public**'

Variables & Methods
can't be '**private**'

MODULE PATTERN

Simulate privacy

The typical module pattern is where immediately invoked function expressions (IIFEs) use **execution context** to create 'privacy'. Here, **objects** are returned instead of functions.

```
var basketModule = (function() {
    var basket = []; //private
    return { //exposed to public
        addItem: function(values) {
            basket.push(values);
        },
        getItemCount: function() {
            return basket.length;
        },
        getTotal: function(){
            var q = this.getItemCount(),p=0;
            while(q--){
                p+= basket[q].price;
            }
            return p;
        }
    }
}());
```

- In the pattern, variables declared are **only** available **inside** the module.
- Variables defined **within** the **returning object** are available to everyone
- This allows us to **simulate** privacy

MODULE PATTERN

Sample usage

Inside the module, you'll notice we return an object. This gets automatically assigned to basketModule so that you can interact with it as follows:

```
//basketModule is an object with properties which can also be methods
basketModule.addItem({item:'bread',price:0.5});
basketModule.addItem({item:'butter',price:0.3});

console.log(basketModule.getItemCount());
console.log(basketModule.getTotal());

//however, the following will not work:
// (undefined as not inside the returned object)
console.log(basketModule.basket);
//(only exists within the module scope)
console.log(basket);
```

MODULE PATTERN

Module Pattern: Dojo

Dojo attempts to provide 'class'-like functionality through **dojo.declare**, which can be used for amongst other things, creating implementations of the module pattern. Powerful when used with **dojo.provide**.

```
// traditional way
var store = window.store || {};
store.basket = store.basket || {};

// another alternative..
// using dojo.setObject (with basket as a module of the store namespace)
dojo.setObject("store.basket.object", (function() {
    var basket = [];
    function privateMethod() {
        console.log(basket);
    }
    return {
        publicMethod: function(){
                privateMethod();
        }
    };
}()));
```

MODULE PATTERN

Module Pattern: YUI

A YUI module pattern implementation that follows the same general concept.

```
YAHOO.store.basket = function () {

    //"private" variables:
    var myPrivateVar = "I can be accessed only within YAHOO.store.basket .";

    //"private" method:
    var myPrivateMethod = function () {
            YAHOO.log("I can be accessed only from within YAHOO.store.basket");
        }

    return {
        myPublicProperty: "I'm a public property.",
        myPublicMethod: function () {
            YAHOO.log("I'm a public method.");

            //Within basket, I can access "private" vars and methods:
            YAHOO.log(myPrivateVar);
            YAHOO.log(myPrivateMethod());

            //The native scope of myPublicMethod is store so we can
            //access public members using "this":
            YAHOO.log(this.myPublicProperty);
        }
    };

}();
```

MODULE PATTERN

Module Pattern: ExtJS

Another library that can similarly use the module pattern.

```
// define a namespace
Ext.namespace('myNamespace');

// define a module within this namespace
myNameSpace.module = function() {
    // recommended that you don't directly access the DOM
    // from here as elements don't exist yet. Depends on
    // where/how you're waiting for document load.

    // private variables

    // private functions

    // public API
    return {
        // public properties

        // public methods
        init: function() {
            console.log('module initialised successfully');
        }
    };
}(); // end of module
```

Now, you can see:

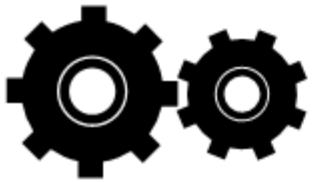
02 The Revealing Module Pattern

http://youtu.be/3T0tQGJ1LUU

MODULE PATTERN: AMD

Better: AMD

Take the concept of reusable JavaScript modules further with the **A**synchronous **M**odule **D**efinition.



Mechanism for defining **asynchronously** loadable modules & dependencies

Non-blocking, parallel loading and well defined.

Stepping-stone to the module system proposed for ES Harmony

AMD PATTERN

AMD: define()

define allows the definition of modules with a signature of
define(**id** /*optional*/, **[dependencies]**, **factory** /*module instantiation fn*/);

```
/* wrapper */
define(
    /*module id*/
    'myModule',

    /*dependencies*/
    ['foo','bar','foobar'],

    /*definition for the module export*/
    function (foo, bar, foobar) {

        /*module object*/
        var module = {};

        /*module methods go here*/
        module.hello = foo.getSomething();
        module.world = bar.doSomething();

        /*return the defined module object*/
        return module;
    }
);
```

AMD PATTERN

```
define(
    'account',
    ['service', 'pubsub'],
    function (service, pubsub) {
        //do something

        //export public APIs
        return {
            signin: function () { /* ... */ },
            signout: function () { /* ... */ },
            getName: function () { /* ... */ },
            setName: function () { /* ... */ }
        };
    }
);
```

Another Way

```
(function () {
    //10000 lines of code

    exports = {
        signin: function () { /* ... */ },
        signout: function () { /* ... */ }
    };

    define('account', function () {
        return exports;
    });
}());
```

AMD PATTERN

AMD: require()

require is used to load code for top-level JS files or inside modules for dynamically fetching dependencies

```
/* top-level: the module exports (one, two) are passed as
 function args to the callback.*/
require(['one', 'two'], function (one, two) {

});

/* inside: complete example */
define('three', ['one', 'two'], function (one, two) {

    /*require('string') can be used inside the function
    to get the module export of a module that has
    already been fetched and evaluated.*/

    var temp = require('one');

    /*This next line would fail*/
    var bad = require('four');

    /* Return a value to define the module export */
    return function () {};
});
```

Now, you can see:

03 Why AMD

http://youtu.be/-TrH8q0YmXE

MODULE PATTERN LARGE APPLICATION

Lots of modules

One file per module
Lots of files:
Performance Issue
Async loading?

Complex Dependency

Load modules by order
Hard to know the order by head and hand

Solution?

Not hard to solve
But there is no standard API

MODULE PATTERN: COMMONJS

Alternative: CommonJS

Another easy to use module system with wide adoption server-side

CommonJS
Working group designing, prototyping, standardizing JS APIs

Format widely accepted on a number of server-side platforms (Node)

Competing standard. Tries to solve a few things AMD doesn't.

CommonJS

by Kevin Dangoor

Volunteers and mailing list

Unify server side JavaScript API

Build JavaScript ecosystem

Not an official organization

COMMONJS

CommonJS Modules

They basically contain two parts: an **exports** object that contains the objects a module wishes to expose and a **require** function that modules can use to import the exports of other modules

```
/* here we achieve compatibility with AMD and CommonJS
using some boilerplate around the CommonJS module format*/
(function(define){
    define(function(require,exports){
        /*module contents*/
        var dep1 = require("foo");
        var dep2 = require("bar");
        exports.hello = function(){...};
        exports.world = function(){...};
    });
})(typeof define=="function"? define:function(factory){factory
(require,exports)});
```

MODULE PATTERN

Better alternative: Universal Module Definition

Defining modules that can work **anywhere** (CommonJS environments such as clients, servers; with script loaders etc). Thx to @KitCambridge for this version.

```
(function (root, Library) {
  // The square bracket notation is used to avoid property munging by the Closure Compiler.
  if (typeof define == "function" && typeof define["amd"] =="object" && define["amd"]) {
    // Export for asynchronous module loaders (e.g., RequireJS, `curl.js`).
    define(["exports"], Library);
  } else {
    // Export for CommonJS environments, web browsers, and JavaScript engines.
    Library = Library(typeof exports == "object" && exports|| (root["Library"] = {
      "noConflict": (function (original) {
        function noConflict() {
          root["Library"] = original;
          // `noConflict` can't be invoked more than once.
          delete Library.noConflict;
          return Library;
        }
        return noConflict;
      })(root["Library"])
    }));
  }
})(this, function (exports) {
  // module code here
  return exports;
});
```

MODULE PATTERN

ES Harmony Modules

A module format proposed for EcmaScript Harmony with goals such as static scoping, simplicity and usability.

```
// Basic module
module SafeWidget {
    import alert from Widget;
    var _private ="someValue";
    // exports
    export var document = {
        write: function(txt) {
            alert('Out of luck, buck');
        },
        ...
    };
}

// Remote module
module JSONTest from 'http://json.org/modules/json2.js';
```

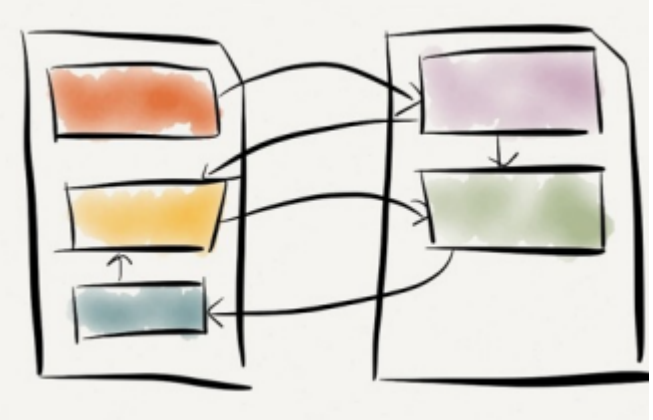
JS is big!!

Web-apps today contain large amounts of JavaScript

Even if you don't run a JS framework, just "some HTML and JS", you of ten have rather large JS files - quite of ten spaghetti-like with a large number of functions calling each other.
New JS-heavy apps of ten use one or more of the JS Frameworks that exist today, e.g. Backbone, Knockout, Ember, SproutCore etc, making it more and more important to think about componentizing and structuring your code.

Dependency management is hard

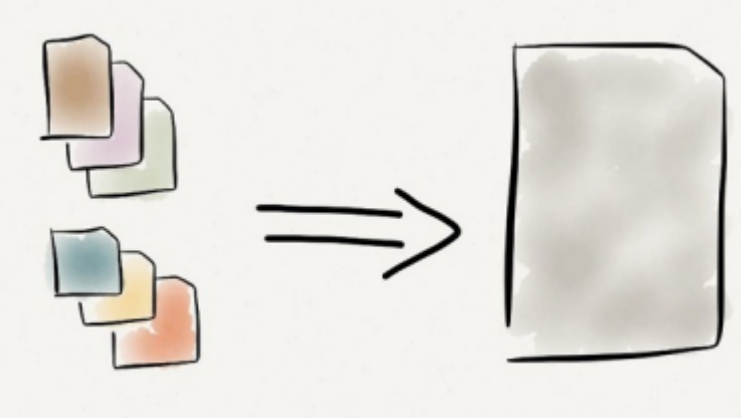
Top-level functions in the global namespace of ten lead to circular dependencies (which aren't easily spotted - you basically need to wade through the JS code to find all dependencies).
2 Files - 5 different functions calling each other



Dev/deploy conflict

Requirements conflict between development and deployment:

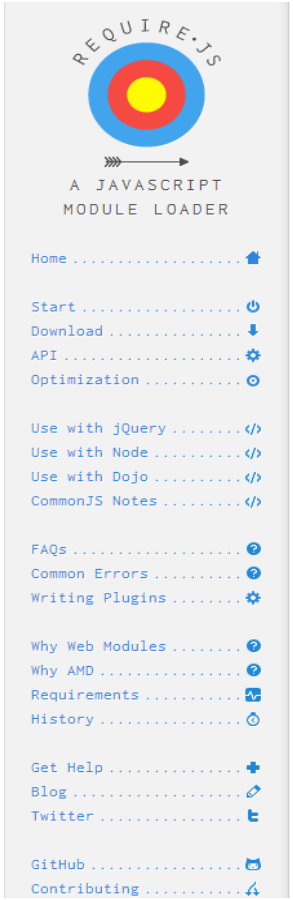
Devs want small discrete files since it's easier to debug and test small units.
Additionally, using large files with a script tag doesn't scale in larger apps with large number of developers: e.g. merge hell in version control



Deployment should be done on large files since we don't want the browser to load lots of files - the latency gives us slowly loaded page

RequireJS

Clientside Dependency Management
AMD/CommonJS implementation



RequireJS

AMD Implementation by James Burke
Async resource loader

Require tries to solve the problems with growing JS code bases by introducing the “module” abstraction.

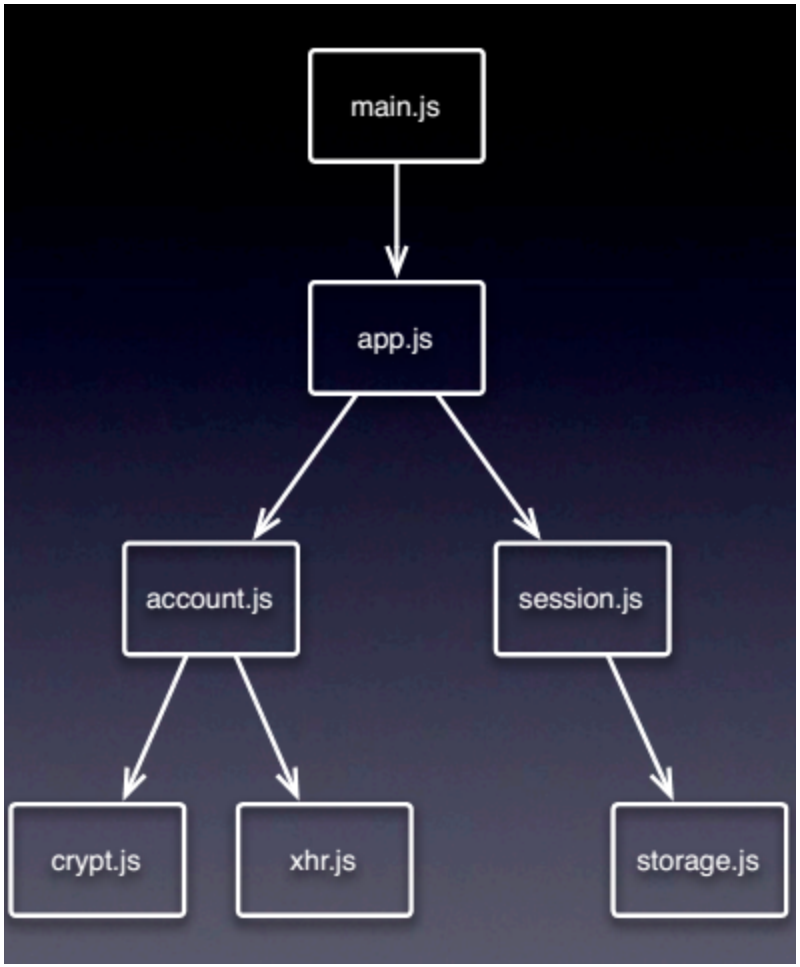
In its core, modules exposes an interface and has dependencies.

RequireJS

You ask RequireJS for a module, and RequireJS traces and instantiates its dependencies (including transitive dependencies) before instantiating the module you wish to use.

Why modules?
Modules are (or, should be)
small => easy to understand => easy to change + easy to test

RequireJS


main.js
app.js
account.js
session.js
crypt.js
xhr.js
storage.js

RequireJS

Using Require.js we define a single entry point on our index page.
We should setup any useful containers that might be used by our Backbone views.

Note: The data-main attribute on our single script tag tells Require.js to load the script located at “js/main.js”. It automatically appends the “.js”

RequireJS

```
1       <!doctype html>
2       <html lang="en">
3       <head>
4         <title>Jackie Chan</title>
5         <!-- Load the script "js/main.js" as our entry point -->
6         <script data-main="js/main" src="js/libs/require/require.js"></script\
7   >
8       </head>
9       <body>
10
11      <div id="container">
12        <div id="menu"></div>
13        <div id="content"></div>
14      </div>
15
16      </body>
17      </html>
```

RequireJS

Unfortunately Backbone.js isn't AMD enabled so I downloaded the community managed repository and patched it on amdjs.
Hopefully if the AMD specification takes off these libraries will add code to allow themselves to be loaded asynchronously. Due to this inconvenience the bootstrap is not as intuitive as it could be.
We also request a module called "app", this will contain the entirety of our application logic.

RequireJS

```
1   // Filename: main.js
2
3   // Require.js allows us to configure shortcut alias
4   // There usage will become more apparent further along in the tutorial.
5   require.config({
6     paths: {
7       jquery: 'libs/jquery/jquery',
8       underscore: 'libs/underscore/underscore',
9       backbone: 'libs/backbone/backbone'
10    }
11
12  });
13
14  require([
15
16    // Load our app module and pass it to our definition function
17    'app',
18  ], function(App){
19    // The "app" dependency is passed in as "App"
20    App.initialize();
21  });
```

RequireJS Features

AMD

Path alias

Circular dependency

Plugins

- order
- Text
- wrap, use
- cs (CoffeeScript)

RequireJS Advantages

No pollution to global scope
Everything is organized in module
Compile CoffeeScript on the fly

Now, you can see:

04 RequireJS

http://youtu.be/-Aa1kmBYg7o

Minefield

Module load fail: hard to debug

- Wrong path
- Use require function
- Plugin error, ex: CoffeeScript syntax error

Example

I don’t work!!!

```
var Employee = require("types/Employee");

function Manager () {
    this.reports = [];
}

Manager.prototype = new Employee();
```

Example

RequireJS

```
define(
    //The name of this module
    "types/Manager",

    //The array of dependencies
    ["types/Employee"],

    function (Employee) {
        function Manager () {
            this.reports = [];
        }
        Manager.prototype = new Employee();
        return Manager;
    }
);
```

Example

Node.js
requirejs in package.json and go!

```
var requirejs = require('requirejs');

requirejs.config({
    nodeRequire: require
});

requirejs(['ideedock'], function(Ideedock) {
    var ideedock = new Ideedock();
    ideedock.initialize();
});
```

Example

The Client

index.html

```
<script data-main="js/main" src="js/lib/require.js"></script>
```

main.js

```
require(['ideedock'], function(Ideedock) {
    ideedock = new Ideedock();
    ideedock.initialize();
});
```

Looks good. However, this is deceptively not that awesome...yet

The Shim

Most of your favorite frameworks, libraries and plugins are not requirablized!

...bring in the shim.

The Shim

Configure the dependencies, exports, and custom initialization for older, traditional "browser globals" scripts that do not use define() to declare the dependencies and set a module value.

The Shim

For "modules" that are just jQuery or Backbone plugins that do not need to export any module value, the shim config can just be an array of dependencies:

```
requirejs.config({
    shim: {
        'jquery.colorize': ['jquery'],
        'jquery.scroll': ['jquery'],
        'backbone.layoutmanager': ['backbone']
    }
});
```

The Shim

```
requirejs.config({
      baseUrl: "js/",
      paths: {
            'backbone': 'http://cdnjs.cloudflare.com/ajax/libs/backbone.js/0.9.2/backbone-min',
            'ejs': 'lib/ejs_production',
            'fancybox': 'fancybox/source/jquery.fancybox',
            'jquery': 'https://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min',
            'jquery.cookie': 'lib/jquery.cookie',
            'jquery.gravatar': 'lib/jquery.gravatar',
            'jquery.timeago': 'lib/jquery.timeago',
            'md5': 'lib/md5-min',
            'sockjs': 'http://cdn.sockjs.org/sockjs-0.3.2.min',
            'stomp': 'lib/stomp',
            'supporters': 'lib/supporters',
            'underscore': 'http://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.3.3/underscore-min'
      },
      shim: {
            'backbone': {
                  deps: ['underscore', 'jquery'],
                  exports: 'Backbone'
            },
            'fancybox': ['jquery'],
            'jquery.cookie': ['jquery'],
            'jquery.gravatar': ['jquery'],
            'jquery.timeago': ['jquery'],
            'stomp': ['sockjs']
      }
});
```

The Shim

The shim config only sets up code relationships. To load modules that are part of or use shim config, a normal require/define call is needed. Setting shim by itself does not trigger code to load.

Only use other "shim" modules as dependencies for shimmed scripts, or AMD libraries that have no dependencies and call define() after they also create a global (like jQuery or lodash). Otherwise, if you use an AMD module as a dependency for a shim config module, after a build, that AMD module may not be evaluated until after the shimmed code in the build executes, and an error will occur.

The ultimate fix is to upgrade all the shimmed code to have optional AMD define() calls.

The Shim

The init function will **not** be called for AMD modules.

Shim config is not supported when running AMD modules in node via RequireJS (it works for optimizer use though). Depending on the module being shimmed, it may fail in Node because Node does not have the same global environment as browsers. As of RequireJS 2.1.7, it will warn you in the console that shim config is not supported, and it may or may not work. If you wish to suppress that message, you can pass
requirejs.config({ suppress: { nodeShim: true }});.

Still Problems

Lots of modules
Lots of files
Lots of requests
Low performance

r.js

Optimization tool

Pack all modules into one file

Minimize the JavaScript file

Usage

```
node r.js -o build.js
```

build.js

```
({
    appDir: "../",
    baseUrl: "scripts",
    dir: "../../appdirectory-build",
    modules: [
        {
            name: "main"
        }
    ]
})
```

What is it?
Combines related scripts together into build layers and minifies them via UglifyJS
Optimizes CSS by inlining CSS files referenced by @import and removing comments.

Getting it
Getting it Node.js installed, yea?
npm install –g requires
node r.js path/tp/buildscript.js

Thank you!

Globant
we are ready