Javascript, The Swiss Army Knife of Programming Languages

David Morcillo

30-11-2013

Outline

Previously on JS Workshop...

Introduction to JS Hello World and Syntax

Good parts Objects, Functions, Inheritance and Arrays

Node.js Javascript platform and npm for back-end dependencies

Bower front-end dependencies

Grunt Javascript task runner

Basic HTML5 Canvas and requestAnimationFrame

Git cheatsheet

`git init` Initialize git repository.

`git add .` Add all changes to stage.

`git commit -am` Commit changes

`git checkout <commit>` Checkout code to specific commit.

`git diff` Show changes between workspace and last commit

`git status -sb` Show current status of workspace and stage

`git log` Show history

Outline

Problem

Including scripts

```
<html>
  <head>
    <script src=‘‘js/game.js’’></script>
    <script src=‘‘js/character.js’’></script>
    <script src=‘‘js/player.js’’></script>
    <script src=‘‘js/enemy.js’’></script>
    <script src=‘‘js/knight.js’’></script>
    <script src=‘‘js/soldier.js’’></script>
    <script src=‘‘js/protector.js’’></script>
    ...
```

Problem

Including scripts

```
<html>
  <head>
    <script src=‘‘js/game.js’’></script>
    <script src=‘‘js/character.js’’></script>
    <script src=‘‘js/player.js’’></script>
    <script src=‘‘js/enemy.js’’></script>
    <script src=‘‘js/knight.js’’></script>
    <script src=‘‘js/soldier.js’’></script>
    <script src=‘‘js/protector.js’’></script>
    ...
```

- We need to remember the inclusion order
- Each module, function or object must be accessible through global scope if we want to use it as a dependency.

Possible solution

index.html

```
<html>
  <head>
    <script src=‘‘js/built.js’’></script>
    ...
```

Gruntfile.js

```
...
    concat: {
        options: {
            separator: ';',
        },
        dist: {
            src: [
                'js/game.js',
                'js/character.js',
                'js/player.js',
                'js/enemy.js',
                'js/knight.js',
                'js/soldier.js',
                'js/protector.js'
            ],
...
```

Require.js

A javascript module loader

RequireJS is a JavaScript file and module loader. It is optimized for in-browser use, but it can be used in other JavaScript environments, like Rhino and Node. Using a modular script loader like RequireJS will improve the speed and quality of your code.

Require.js: an example

Without Require.js

```
var MYGAME = MYGAME || {},
    game   = MYGAME.game,
    entity = MYGAME.entity;

MYGAME.crate = function (spec) {
  // code omitted
};
```

With Require.js

```
define(function (require) {
  var game   = require('game'),
      entity = require('entity'),
      crate;

  crate = function (spec) {
  };

  return crate;
});
```

Require.js: getting started

Get Require.js

Use `bower` to install it as a dependency of your project

Require.js: getting started

Get Require.js

Use `bower` to install it as a dependency of your project

Include it

```
<html>
  <head>
    <script data-main=‘‘scripts/main’’ src=‘‘bower_components/requirejs/require.js’’></script>
    ...
```

Require.js: getting started

Get Require.js

Use `bower` to install it as a dependency of your project

Include it

```
<html>
  <head>
    <script data-main=‘‘scripts/main’’ src=‘‘bower_components/requirejs/require.js’’></script>
    ...
```

Define modules

```
define(function (require) {
    // code omitted
});
```

Require.js: Lab

Exercise

- `git checkout stage_6`
- Install your back-end dependencies with `npm install`
- Install your front-end dependencies with `bower install`
- Start `grunt watch` for auto linting
- Install Require.js and include the main entry point
- Refactor modules and functions using Require.js modules.

Outline

Event Loop

Event Loop

JavaScript has a concurrency model based on an "event loop". This model is quite different than the model in other languages like C or Java.

```
while(queue.waitForMessage()) {
  queue.processNextMessage();
}
```

Event Loop

Event Loop

JavaScript has a concurrency model based on an "event loop". This model is quite different than the model in other languages like C or Java.

```
while(queue.waitForMessage()) {
  queue.processNextMessage();
}
```

A very interesting property of the event loop model is that JavaScript, unlike a lot of other languages, never blocks.

Event Loop

Example

```
var now = +new Date();

setTimeout(function () {
    var dt = (+new Date()) - now;
    console.log('First timeout');
    console.log('Elapsed time: ' + dt + ' milliseconds');
}, 500);

setTimeout(function () {
    var dt = (+new Date()) - now,
        i = 0;
    console.log('Second timeout');
    console.log('Elapsed time: ' + dt + ' milliseconds');
    while(i < 1000000000) {
      i += 1;
    }
}, 250);
```

Event Loop

Example

```
var now = +new Date();

setTimeout(function () {
    var dt = (+new Date()) - now;
    console.log('First timeout');
    console.log('Elapsed time: ' + dt + ' milliseconds');
}, 500);

setTimeout(function () {
    var dt = (+new Date()) - now,
        i = 0;
    console.log('Second timeout');
    console.log('Elapsed time: ' + dt + ' milliseconds');
    while(i < 1000000000) {
      i += 1;
    }
}, 250);
```

Output

```
Second timeout
Elapsed time: 252 milliseconds
First timeout
Elapsed time: 1271 milliseconds
```

DOM Events

When an event happens, the browser sends the event to the related element. If you've set a handler (a function) on that element, it gets called with related event info which means you 'handled' the event.

DOM Events

When an event happens, the browser sends the event to the related element. If you've set a handler (a function) on that element, it gets called with related event info which means you 'handled' the event.

The old way

```
var blueSoldierSeat = document.getElementById('blue-soldier-seat');
blueSoldierSeat.onclick = function (event) {
    // code omitted
};
```

DOM Events

When an event happens, the browser sends the event to the related element. If you've set a handler (a function) on that element, it gets called with related event info which means you 'handled' the event.

The old way

```
var blueSoldierSeat = document.getElementById('blue-soldier-seat');
blueSoldierSeat.onclick = function (event) {
    // code omitted
};
```

The classy way

```
var blueSoldierSeat = document.getElementById('blue-soldier-seat');
blueSoldierSeat.addEventListener('click', function (event) {
    // code omitted
});
```

DOM Events

The jQuery way #1

```
var $blueSoldierSeat = $('#blue-soldier-seat');
$blueSoldierSeat.click(function (event) {
    // code omitted
});
```

DOM Events

The jQuery way #1

```
var $blueSoldierSeat = $('#blue-soldier-seat');
$blueSoldierSeat.click(function (event) {
    // code omitted
});
```

The jQuery way #2

```
var $blueSoldierSeat = $('#blue-soldier-seat');
$blueSoldierSeat.on('click', function (event) {
    // code omitted
});
```

Custom Events

We can use a library or implement our own functions for listening and triggering custom events.

Custom Events

We can use a library or implement our own functions for listening and triggering custom events.

With jQuery on and trigger

```
var tree = {
    apples: 10
};

$(document).on('apple fall', function (event) {
    tree.apples -= 1;
});

$(document).trigger('apple fall');

console.log(tree.apples); // Output: 9
```

Custom Events

With library IndigoUnited/events-emitter

```
// Require.js code omitted
var EventsEmitter  = require('events-emitter/EventsEmitter'),
    emitter        = new EventsEmitter(),
    tree           = {
        apples: 10
    };

emitter.on('apple fall', function (event) {
    tree.apples -= 1;
});

emitter.emit('apple fall');

console.log(tree.apples); // Output: 9
```

Callbacks

Callback function without this

```
function chooseCharacterClass (event) {
    var characterClass = extractCC(event);
    player.characterClass = characterClass;
}

var classButtons = $('.classButton');
classButtons.on('click', chooseCharacterClass);
```

Callbacks

Callback function without this

```
function chooseCharacterClass (event) {
    var characterClass = extractCC(event);
    player.characterClass = characterClass;
}

var classButtons = $('.classButton');
classButtons.on('click', chooseCharacterClass);
```

Callback function with this

```
var player = {
    chooseCharacterClass: function (event) {
        var characterClass = extractCC(event);
        this.characterClass = characterClass; // Problem
    }
};

var classButtons = $('.classButton');
classButtons.on('click', player.chooseCharacterClass); // Warning!
```

Callbacks

A solution using bind

```
var player = {
    chooseCharacterClass: function (event) {
        var characterClass = extractCC(event);
        this.characterClass = characterClass;
    }
};

var classButtons = $('.classButton');
classButtons.on('click', player.chooseCharacterClass.bind(player));
```

Callbacks

A solution using bind

```
var player = {
    chooseCharacterClass: function (event) {
        var characterClass = extractCC(event);
        this.characterClass = characterClass;
    }
};

var classButtons = $('.classButton');
classButtons.on('click', player.chooseCharacterClass.bind(player));
```

Another solution using jQuery proxy

```
var player = {
    chooseCharacterClass: function (event) {
        var characterClass = extractCC(event);
        this.characterClass = characterClass;
    }
};

var classButtons = $('.classButton');
classButtons.on('click', $.proxy(player.chooseCharacterClass, player));
```

Events: Lab

Exercise

- `git checkout stage_7`
- Install your back-end dependencies with `npm install`
- Install your front-end dependencies with `bower install`
- Start `grunt watch` for auto linting
- Find TODOs and complete the exercise.

Outline

Introduction

We are going to do our first steps on a multiplayer game using Websockets. First, we need a web server to serve our web application (our game) and handle communications between all clients connected to the server.

Introduction

We are going to do our first steps on a multiplayer game using Websockets. First, we need a web server to serve our web application (our game) and handle communications between all clients connected to the server.

Express.js

Express is a minimal and flexible node.js web application framework, providing a robust set of features for building single and multi-page, and hybrid web applications.

Express.js: Hello World!

Installation

```
npm install express --save-dev
```

Express.js: Hello World!

Installation

```
npm install express --save-dev
```

Hello World

```
var express = require('express'),
    app     = express();

app.get('/hello.txt', function(req, res){
    var body = 'Hello World';
    res.setHeader('Content-Type', 'text/plain');
    res.setHeader('Content-Length', body.length);
    res.end(body);
});

app.listen(9000);
console.log('Listening on port 9000');
```

Express.js: Hello World!

Installation

```
npm install express --save-dev
```

Hello World

```
var express = require('express'),
    app     = express();

app.get('/hello.txt', function(req, res){
    var body = 'Hello World';
    res.setHeader('Content-Type', 'text/plain');
    res.setHeader('Content-Length', body.length);
    res.end(body);
});

app.listen(9000);
console.log('Listening on port 9000');
```

Run server

```
$ node index.js // Open browser and visit http://localhost:9000/hello.txt
```

Express.js: API

Serving static files with app.use

```
app.use(express.static(__dirname + '/public'));
```

Express.js: API

Serving static files with `app.use`

```
app.use(express.static(__dirname + '/public'));
```

Using a template engine system

Install a template engine system. For example `jade`:

```
$ npm install jade --save-dev
```

Use it on our Express application:

```
app.set("view engine", "jade");
app.set("views", __dirname + "/views");
```

Express.js: API

Defining routes and rendering views

```
app.get('/about', function (req, res) {
    res.render('about');
});

app.get('/credits', function (req, res) {
  res.render('credits', { name: 'test' }); // Pass parameters to the view
});

app.post('/players', function (req, res) {
    res.render('players/show');
});

app.put('/players', function (req, res) {
    res.render('players/show');
});
```

Express.js: API

Parse query parameters

```
// GET /search?q=nintendo
app.get('/search', function (req, res) {
    var q = req.query.q;
});
```

Express.js: API

Parse query parameters

```
// GET /search?q=nintendo
app.get('/search', function (req, res) {
    var q = req.query.q;
});
```

Parse body

First, we need to use `bodyParser` middleware.

```
app.use(express.bodyParser());
```

Then, we can parse body directly:

```
// POST /players player[name]=David
app.post('/players', function (req, res) {
    var playerData = req.body.player,
        playerName = playerData.name;
});
```

Express.js: Lab

- `git checkout stage_8_1`
- Install your back-end dependencies with `npm install`
- Install your front-end dependencies with `bower install`
- Start `grunt watch` for auto linting
- Find TODOs and complete the exercise.

Websockets

Without Websockets we have the limitation of unidirectional communication between server and client. We can emulate some kind of bidirectional communication using AJAX and polling but it's a poor option in real-time applications.

Websockets

Without Websockets we have the limitation of unidirectional communication between server and client. We can emulate some kind of bidirectional communication using AJAX and polling but it's a poor option in real-time applications.

Socket.io

Socket.IO aims to make realtime apps possible in every browser and mobile device, blurring the differences between the different transport mechanisms. It's care-free realtime 100% in JavaScript.

Socket.io: Hello World!

Installation

```
npm install socket.io --save-dev
```

Socket.io: Hello World!

Installation

```
npm install socket.io --save-dev
```

Server-side

```
var express = require('express'),
    http    = require('http'),
    app     = express(),
    server  = http.createServer(app)
    io      = require('socket.io').listen(server);

  // code omitted

  io.sockets.on('connection', function (socket) {
      socket.emit('news', { hello: 'world' });
      socket.on('my other event', function (data) {
          console.log(data);
      });
  });

  // Replace app.listen with this
  server.listen(9000);
```

Socket.io: Hello World!

Require.js configuration

```
requirejs.config({
    // code omitted
    paths: {
        'io': '/socket.io/socket.io'
    }
});
```

Socket.io: Hello World!

Require.js configuration

```
requirejs.config({
    // code omitted
    paths: {
        'io': '/socket.io/socket.io'
    }
});
```

Client-side

```
var io     = require('io'),
    socket = io.connect();

socket.on('news', function (data) {
    console.log(data);
    socket.emit('my other event', { my: 'data' });
});
```

Socket.io: API

Send and receive messages

```
socket.on('message', function (data) {
    var player = data.player;
});

socket.emit('message', { player: 'player1' });
```

Socket.io: API

Send and receive messages

```
socket.on('message', function (data) {
    var player = data.player;
});

socket.emit('message', { player: 'player1' });
```

Broadcast messages

```
// On the server side
socket.broadcast.emit('player logout', { player: 'player1' });
```

Socket.io: API

Store information associated to a client

```
// On the server side
socket.set('playerId', 'player', function () {
    socket.emit('player saved');
});
```

Socket.io: API

Store information associated to a client

```
// On the server side
socket.set('playerId', 'player', function () {
    socket.emit('player saved');
});
```

Retrieve information associated to a client

```
// On the server side
socket.get('playerId', function (err, player) {
    socket.emit('player loaded', { player: player });
});
```

Socket.io: Lab

- `git checkout stage_8_2`
- Install your back-end dependencies with `npm install`
- Install your front-end dependencies with `bower install`
- Start `grunt watch` for auto linting
- Find TODOs and complete the exercise.

Outline

Persistance

We stored player positions on a simple object on the server-side code. If we reboot the server we will lose these values.

Simple object lost when server restarts

```
var playerPositions = {
    player1: { x: 100, y: 200 },
    player2: { x: 100, y: 400 },
    player3: { x: 700, y: 200 },
    player4: { x: 700, y: 400 }
};
```

RedisDB

Website definition

Redis is an open source, BSD licensed, advanced key-value store. It is often referred to as a data structure server since keys can contain strings, hashes, lists, sets and sorted sets.

RedisDB: Installation

From source code

```
$ wget http://download.redis.io/redis-stable.tar.gz
$ tar xvzf redis-stable.tar.gz
$ cd redis-stable
$ make
```

RedisDB: Installation

From source code

```
$ wget http://download.redis.io/redis-stable.tar.gz
$ tar xvzf redis-stable.tar.gz
$ cd redis-stable
$ make
```

Copy executable and default configuration

```
$ sudo cp src/redis-server /usr/local/bin/
$ sudo cp src/redis-cli /usr/local/bin/
$ sudo cp redis.conf /etc/redis.conf
```

RedisDB: Installation

From source code

```
$ wget http://download.redis.io/redis-stable.tar.gz
$ tar xvzf redis-stable.tar.gz
$ cd redis-stable
$ make
```

Copy executable and default configuration

```
$ sudo cp src/redis-server /usr/local/bin/
$ sudo cp src/redis-cli /usr/local/bin/
$ sudo cp redis.conf /etc/redis.conf
```

Run server

```
$ redis-server
[28550] 01 Aug 19:29:28 # Warning: no config file specified, using the default config...
[28550] 01 Aug 19:29:28 * Server started, Redis version 2.2.12
[28550] 01 Aug 19:29:28 * The server is now ready to accept connections on port 6379
...
```

RedisDB: Client

Installation

```
npm install redis --save-dev
```

RedisDB: Client

Installation

```
npm install redis --save-dev
```

Create client on our code

```
var redis = require('redis'),
    redisClient = redis.createClient();
```

RedisDB: Commands

In this workshop we are going to use one Redis data structure: a Hash

RedisDB: Commands

In this workshop we are going to use one Redis data structure: a Hash

Get all fields and values from a hash

```
redisClient.hgetall('myHash', function (err, result) {
    // result is converted on a Javascript object
});
```

Store some fields and values to a hash

```
redisClient.hmset(['myHash', 'key1', 'value1', 'key2', 'value2'], function (err, result) {
   // code omitted
});
```

RedisDB: Lab

- `git checkout stage_9`
- Install your back-end dependencies with `npm install`
- Install your front-end dependencies with `bower install`
- Start `grunt watch` for auto linting
- Find TODOs and complete the exercise.

Outline

Recap

Bibliography

- **Javascript: The Good Parts**. Douglas Crockford
- **Test-Driven Javascript Development**. Christian Johansen
- **Javascript Patterns**. Stoyan Stefanov
- **Testable Javascript**. Mark Ethan Trostler