Node.js Primer

Introduction

PROGRESS EXCHANGE 2014

Your Guides

Richard Key - @busyrich

- Head of Technical Training & Support - Modulus
- >7yrs Web Development Experience
- Degrees in Game Design & Programming
- JavaScript Ninja

Taron Foxworth - @anaptfox

- The Sales Engineer - Modulus
- >3yrs Web Development Experience
- Lover of Wearable Tech
- Exercise Enthusiast

Goals

- what is Node.js
- how Node.js Works
- Node.js Basics
- using Modules
- using streams to pipe data
- building applications with websockets

Preparation

Secure a Text Editor

- Sublime Text
- Atom (Mac)
- Notepad++ (Windows)
- Vim/Vi (old school)

Create Working Directory

```
~/ $ mkdir node-primer
~/ $ cd node-primer
~/node-primer $
```

Check Node.js

```
~/node-primer/ $ node
> 'Hello World!'
'Hello World!'
>
(^C again to quit)
>
~/node-primer/ $
```

Next Up:

What The Heck is Node.js?

What the Heck is Node.js?

PROGRESS EXCHANGE 2014

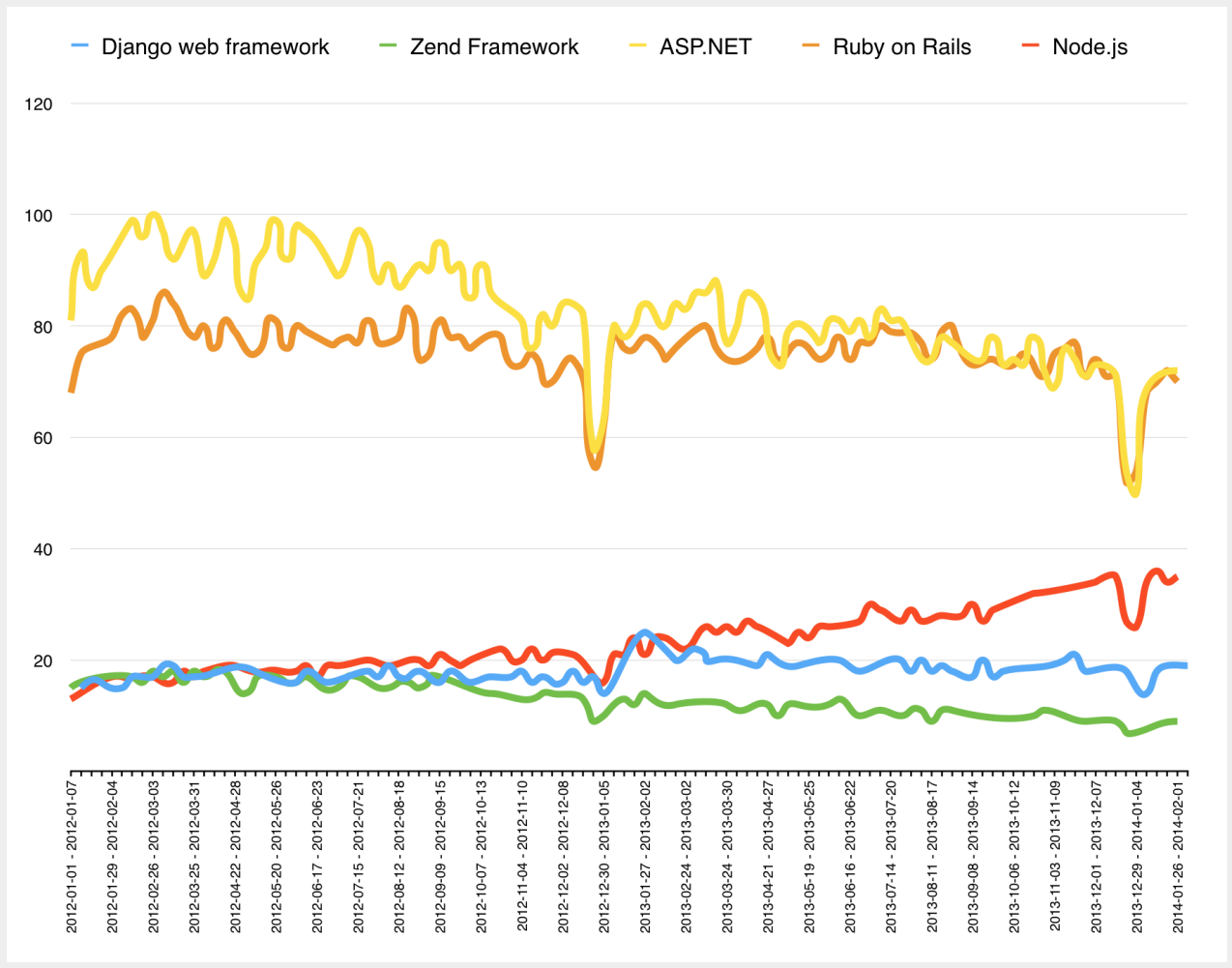
A lightweight, event-driven server-side technology running JavaScript, designed to be asynchronous and non-blocking, built on top of google's V8 engine...what?

Where Node.js Fits In

Language	Saturation
ASP.net	17% of websites
PHP	82%
Ruby	0.5%
Python	0.2%
Node.js	0.1%

Usage data retrieved from W3Techs

Node is Growing



Data retrieved from Google Trends

Google, Microsoft, Yahoo, GE, ebay, Walmart, The New York Times, PayPal, LinkedIn, Klout, Zendesk, rdio...and many more

Some Statistics

- jobs increased by 20,000% (Jan 2014)
- >250 Meetups list Node.js as a topic
- 35,000 downloads/day (Mar 2014)

The Power of Node.js

It's Asynchronous

- execute multiple tasks at the same time
- based on events, not structure
- improves overall performance

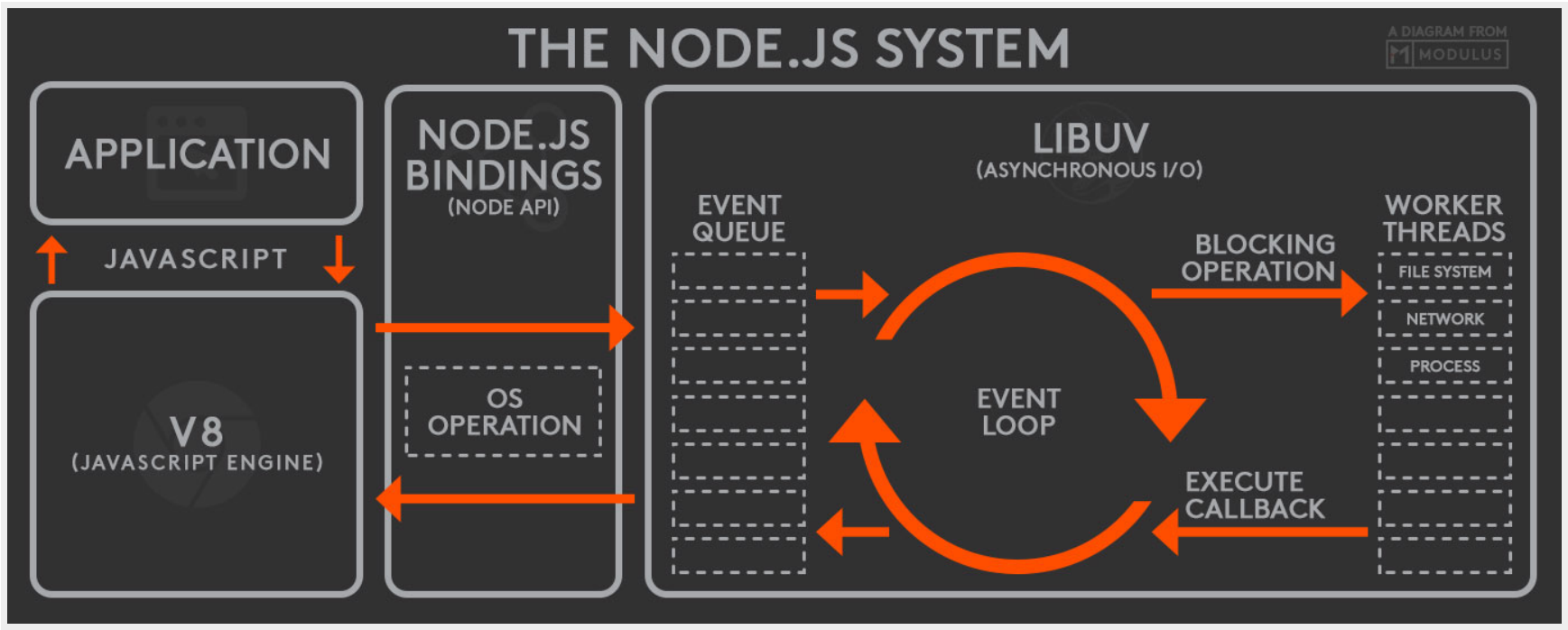
It's Fast

- uses libuv for Asynchronous I/O
- use Google's V8 to run JavaScript
- very little overhead (~20mb of memory)

It's JavaScript!

- familiar for most web developers
- great fit for asynchronous code
- code sharing between client/server

Node's Secret Sauce



Application <=> V8

- javaScript is run directly through V8
- some operations run directly in V8 (object manipulation, simple arithmetic, etc.)
- runs on a single thread

V8 <=> libUV

- OS operations are run via libUV
- passes through the Node.js bindings (Node's API wrapper)
- libUV is the "browser" V8 runs in

libUV's Event Loop

- worker threads run operations from a queue
- fires off **events** when operations finish
- allows **blocking** operations to be asynchronous

Example

```
var v = 2, n = v + 8; //runs in V8

file.get('myfile', function(contents) {
  console.log(contents);
});
// runs via libUV
```

6 lines of JavaScript

Use Cases for Node

API

- handles lots of "little" requests fast
- easily scalable
- large selection of frameworks

Real-time Application

- streams allows piping of data
- asychronous events fits well with real-time
- even more frameworks

Command Line Script

- useful set of built-in modules
- supports pretty much any platform
- if you know Node, you can write a script

Questions?

Next Up: Getting Started

Getting Started with Node.js

PROGRESS EXCHANGE 2014

Hello World

Node Console

```
~/node-primer/ $ node
> 'Hello World!'
'Hello World!'
```

Node `console`

```
> console.log('Hello World!');
Hello World!
undefined
```

More on the `console` object.

Running a file

```
console.log('Hello World!');
```

hello-world.js - 1 lines of JavaScript

```
> node hello-world.js
Hello World!
```

Using Modules

Using `require` to import modules

```
var path = require('path');
console.log(path.resolve('./'));
```

using-require.js - 2 lines of JavaScript

```
~/node-primer/require $ node using-require.js
/Users/richard/node-primer/require
```

More on the core modules.

Using `require` to load files

```
module.exports = {
  name: 'Richard\'s Module',
  version: '0.1.0',
  go: function() {
    return 'Cowabunga dude!';
  }
};
```

my-module.js - 7 lines of JavaScript

```
var richard = require('./my-module');
console.log(richard.go());
```

require-file.js - 2 lines of JavaScript

```
~/node-primer/require $ node require-file.js
Cowabunga dude!
```

`require` just loads files

- all modules are just files
- supports .js, .json, and .node (in that order)
- searches multiple locations
- core modules -> local file -> node_modules

Callbacks

dramatic music

Think in events

Synchronous code

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

for(var i = 0; i < 10; i++) {
  fs.writeFileSync('file' + i + '.txt', 'Hello World!');
  console.log('File ' + i + ' saved.');
}
```

synchronous.js - 6 lines of JavaScript

```
~/node-primer/callbacks $ node synchronous.js
File 0 saved.
File 1 saved.
File 2 saved.
File 3 saved.
File 4 saved.
File 5 saved.
File 6 saved.
File 7 saved.
File 8 saved.
File 9 saved.
```

VS Asynchronous code

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

for(var i = 0; i < 10; i++) {
  fs.writeFile('file' + i + '.txt', 'Hello World!', function(err) {
    console.log('File ' + this + ' saved.');
  }.bind(i));
}
```

asynchronous.js - 7 lines of JavaScript

```
~/node-primer/callbacks $ node asynchronous.js
File 1 saved.
File 3 saved.
File 0 saved.
File 6 saved.
File 5 saved.
File 4 saved.
File 8 saved.
File 7 saved.
File 9 saved.
File 2 saved.
```

Important Differences

- events run simultaneously
- events don't block each other
- code design changes

Your First Server

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

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World\n');
}).listen(1337);

console.log('Server running at http://localhost:1337/');
```

hello-server.js - 6 lines of JavaScript

Questions?

Up Next: NPM

NPM

PROGRESS EXCHANGE 2014

What is NPM?

A module Repository

- 88,000 modules (Aug 14)
- 10-20,000,000 downloads/day (Aug 14)
- thousands of users

Dependency Management

- uses package.json file to install dependencies
- automatic version detection and restriction
- provides lots of other metadata

A CLI Tool

- search for modules
- auto-generate package.json files
- install, and save, modules/dependencies

Basic NPM Usage

Few Important Notes

- NPM comes with Node
- Node and NPM are closely entangled
- NPM is updated via NPM

Installing a Module

```
~/node-primer/npm $ npm install easyseed
npm http GET https://registry.npmjs.org/easyseed
...
npm http GET https://registry.npmjs.org/delayed-stream/-/delayed-stream-0.0.5.tgz
npm http 200 https://registry.npmjs.org/delayed-stream/-/delayed-stream-0.0.5.tgz
easyseed@0.0.1 node_modules/easyseed
└── request@2.22.0 (json-stringify-safe@4.0.0, forever-agent@0.5.2,
aws-sign@0.3.0, qs@0.6.6, tunnel-agent@0.3.0, oauth-sign@0.3.0,
cookie-jar@0.3.0, mime@1.2.11, node-uuid@1.4.1, hawk@0.13.1,
http-signature@0.10.0, form-data@0.0.8)
```

Using a Module

```
var easyseed = require('easyseed');
easyseed.seed.auto();
easyseed.float(function(n) {
  console.log(n);
});
```

using.js - 5 lines of JavaScript

```
~/node-primer/npm $ node using
0.20436576152466984
```

Package.json

Why a Package.json?

- dependency management
- project/module metadata
- organization

Creating a Package.json

```
~/node-primer/npm $ npm init
...
name: (npm) packagejson
...
Is this ok? (yes)
~/node-primer/npm $ ls
node_modules package.json using.js
```

What is in a Package.json?

```
{
  "name": "packagejson",
  "version": "0.0.1",
  "description": "",
  "main": "using.js",
  "dependencies": {
    "easyseed": "^0.0.1"
  },
  "devDependencies": {},
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Richard Key <rich@busyrich.com>",
  "license": "ISC"
}
```

package.json - 15 lines of JavaScript

More of the package.json file format

Saving Dependancies

```
~/node-primer/npm $ npm install uberand --save
npm http GET https://registry.npmjs.org/uberand
...
uberand@0.0.2 node_modules/uberand
├── request@2.12.0
└── vows@0.7.0 (diff@1.0.8, eyes@0.1.8)
```

```
...
"dependencies": {
  "easyseed": "^0.0.1",
  "uberand": "0.0.2"
}
...
```

package.json - 6 lines of JavaScript

More on npm install options

Useful Modules

Web Frameworks

- Express
- Hapi
- Restify

Task Runners/Build Tools

- Grunt
- Gulp
- Broccoli

Utilities

- Async
- lodash
- Node Inspector

All-in-Ones (WARNING!)

- Meteor
- Sails
- MEAN.io

Questions?

Next Up: Your First Server!

Your First Node.js Server

PROGRESS EXCHANGE 2014

Using the HTTP Module

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World\n');
}).listen(1337);

console.log('Server running at http://localhost:1337/');
```

hello-server.js - 6 lines of JavaScript

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});

  if(req.url.toLowerCase() === '/hello' && req.method === 'GET') {
    res.end('World!');
  } else {
    res.end('Try GET /hello.');
  }
}).listen(1337);
console.log('Server running at http://localhost:1337/');
```

http-module.js - 10 lines of JavaScript

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});

  if(req.url.toLowerCase() === '/hello' && req.method === 'GET') {
    res.end('World!');
  } else if(req.method === 'POST') {
    var data = '';

    req.on('data', function(chunk) {
      data += chunk;
    });

    req.on('end', function() {
      res.end(data);
    });
  } else {
    res.end('Try GET /hello or any POST.');
  }
}).listen(1337);
console.log('Server running at http://localhost:1337/');
```

http-module-post.js - 18 lines of JavaScript

Not Ideal...

- very low-level
- not abstracted well
- high maintenance

Using Express

```
~/node-primer/first-server $ npm init
...
~/node-primer/first-server $ npm install express --save
...
~/node-primer/first-server $ npm install body-parser --save
...
~/node-primer/first-server $
```

7 lines

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

app.get('/hello', function(req, res){
  res.send('World!');
});

var server = app.listen(3000, function() {
  console.log('Listening on port %d', server.address().port);
});
```

express-basic.js - 8 lines of JavaScript

```
var express =  require('express'),
    app = express(),
    bodyParser = require('body-parser');

app.use(bodyParser.json());

app.get('/hello', function(req, res){
  res.send('World!');
});

app.post('*', function(req, res) {
  res.send(req.body);
});

var server = app.listen(3000, function() {
  console.log('Listening on port %d', server.address().port);
});
```

express-basic.js - 13 lines of JavaScript

More on Express body parser

Frameworks Are Much Better

- a lot of features
- extensible
- easy to implement
- one for everyone

Building Web APIs

Application Programming Interface

- typically used to access/manage data
- great way to separate concerns
- REST interfaces are the most common

More on APIs and REST

CRUD

- Create, Read, Update, Delete...your data
- POST, GET, PUT, DELETE...requests
- basis of a web API

More on CRUD

CRUD in Express

```
var express =  require('express'),
    app = express(),
    bodyParser = require('body-parser'),
    data = {},
    router = express.Router();

app.use(bodyParser.json())

// router stuff here

var server = app.listen(3000, function() {
  console.log('Listening on port %d', server.address().port);
});
```

express-crud.js - 10 lines of JavaScript

```
router.route('/item/:id')
  .get(function(req, res, next){
    res.send(data[req.params.id] || null);
  })
  .post(function(req, res, next) {
    req.body.id = req.params.id;
    data[req.params.id] = req.body;
    res.send(req.body);
  })
  .put(function(req, res, next) {
    if(req.body && typeof req.body === 'object' &&
      typeof data[req.params.id] === 'object')
    {
      Object.keys(req.body).forEach(function(key) {
        data[req.params.id][key] = req.body[key];
      });
    }

    res.send(data[req.params.id]);
  })
  .delete(function(req, res, next) {
    if(typeof data[req.params.id] === 'object') {
      delete data[req.params.id];
    }

    res.send(true);
  });

app.use(router);
```

express-crud.js - 26 lines of JavaScript

```
var express =  require('express'),
    app = express(),
    bodyParser = require('body-parser'),
    data = {},
    router = express.Router();

app.use(bodyParser.json())

// router stuff here

app.get('/items', function(req, res) {
  res.send(data);
});

var server = app.listen(3000, function() {
  console.log('Listening on port %d', server.address().port);
});
```

express-crud.js - 13 lines of JavaScript

Overview

- full CRUD operation Support
- route to get all data
- uses Express 4's router object
- no data validation (!!!)
- data is stored in memory(!!!)

Express Mini Web API

Goals

1. full CRUD operation Support
2. organization through Modules
3. data validation

```
~/node-primer/first-api $ npm init
...
~/node-primer/first-api $ npm install express --save
...
~/node-primer/first-api $ npm install body-parser --save
...
~/node-primer/first-api $
```

7 lines

```
var express = require('express'),
    app = express(),
    bodyParser = require('body-parser');

app.use(bodyParser.json());

require('./routes/item')(app);

var server = app.listen(3000, function() {
  console.log('Listening on port %d', server.address().port);
});
```

api.js - 8 lines of JavaScript

```
var express = require('express'),
    items = {};

module.exports = function(app) {
  var itemRouter = express.Router();

  itemRouter.param('id', function(req, res, next, id) {
    if(/^\d+$/.test(id)) {
      next();
    } else {
      next(new Error('Item Id must be a number.'));
    }
  });

  itemRouter.route('/item/:id')
    .get(function(req, res, next){
      res.send(items[req.params.id] || null);
    }) // other CRUD routes

  // get all items route

  app.use(itemRouter);
};
```

routes/item.js - 18 lines of JavaScript

API Design Considerations

Organizing Routes

- `/thing/:id` and `/thing`
- use API versioning, IE `/v1/`
- avoid long, confusing routes
- query params are OK for GET requests
- maintain the CRUD to POST, GET, PUT, DELETE relationship
- try to support multiple data formats
- don't be afraid to follow convention

Organizing Files

- use require/modules to your advantage
- single-level folders
- group things by usage, IE models, controllers, routes, etc.
- think in data objects, not "classes"

Real World File Strucutre

```
api
  - models
    - project
    - user
  - controllers
    - project
    - user
  - routes
    - project
    - user
  - lib
    - util.js
    - data.js
  api.js
```

14 lines

Most Importantly...

- KISS your code
- go with what you know
- ask for help
- all else fails, do some google research

Questions?

Streams

PROGRESS EXCHANGE 2014

What are Streams?

What They Do

- "pipe" data from one source to another
- non-buffering (kinda), data literally "streams"
- allows mutations inline
- simple to implement interface

Why Use Them?

- they are MUCH faster
- you can do some cool mutations, inline
- don't hit the disk, IE no I/O required
- you can implement them yourself
- a lot of modules already use streams

Non-Streaming Example

```
var http = require('http'),
    fs = require('fs');

var server = http.createServer(function (req, res) {
  fs.readFile('./users.csv', function (err, data) {
    res.end(data);
  });
});

server.listen(1337);
console.log('Server running at http://localhost:1337/');
```

buffering.js - 9 lines of JavaScript

Streaming Example

```
var http = require('http'),
    fs = require('fs');

var server = http.createServer(function (req, res) {
  fs.createReadStream('./users.csv').pipe(res);
});

server.listen(1337);
console.log('Server running at http://localhost:1337/');
```

streaming.js - 7 lines of JavaScript

Some Practical Examples

Saving Files

```
var request = require('request'),
    fs = require('fs');

request('http://actionholder.com/i/250')
  .pipe(fs.createWriteStream('action.png'));
```

saving-files.js - 4 lines of JavaScript

Inline Compression (Zip)

```
var http = require('http'),
    archiver = require('archiver'),
    fs = require('fs');

var server = http.createServer(function (req, res) {
  res.writeHead(200, {
    'Content-Type': 'application/zip',
    'Content-disposition': 'attachment; filename=users.zip'
  });

  var zip = archiver('zip');
  zip.pipe(res);

  zip.append(fs.createReadStream('./users.csv'), {name:'data/users.csv'})
    .finalize();
});

server.listen(1337);
console.log('Server running at http://localhost:1337/');
```

zipit.js - 16 lines of JavaScript

File Uploads

```
var formidable = require('formidable'),
    http = require('http'),
    uploadPath = './uploads';

var server = http.createServer(function(req, res) {
  if (req.url == '/upload' && req.method.toLowerCase() == 'post') {

    var form = new formidable.IncomingForm({uploadDir:uploadPath});
    form.parse(req, function(err, fields, files) {
      res.writeHead(200, {'content-type': 'text/plain'});
      res.write('received upload:\n\n');
      res.end(JSON.stringify({fields:fields,files:files}, null, 2));
    });

    return;
  }

  res.writeHead(200, {'content-type': 'text/html'});
  res.end(
    '<form action="/upload" enctype="multipart/form-data" method="post">'+
    '<input type="file" name="upload" multiple="multiple"><br>'+
    '<input type="submit" value="Upload">'+
    '</form>'
  );
});

server.listen(1337);
console.log('Server running at http://localhost:1337/');
```

upload.js - 23 lines of JavaScript

Questions?

Next Up: Websockets

Websockets

PROGRESS EXCHANGE 2014

What are Websockets?

- duplex stream, IE two-way connection
- replaces long polling/polling (yuck!)
- all modern browsers support them
- its 100% real-time
- mostly standardized

Good Use Cases

- chat applications
- multiplayer games
- auto-updating web components/feeds
- collaborative interfaces

Socket.IO

Server

```
var fs = require('fs'),
    server = require('http').createServer(function(req, res) {
      fs.createReadStream('./index.html').pipe(res);
    }),
    io = require('socket.io')(server);

server.listen(1337);
console.log('Server running at http://localhost:1337/');

io.on('connection', function (socket) {
  setInterval(function() {
    socket.emit('news', {
      id: Math.floor(Math.random() * 100000),
      time: Math.floor(Date.now() / 1000)
    });
  }, 2000);

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

server.js - 18 lines of JavaScript

Client

```
<script src="/socket.io/socket.io.js"></script>
<script>
  var socket = io('http://localhost:1337');

  socket.on('news', function (data) {
    var span = document.createElement('pre');
    span.innerHTML = JSON.stringify(data, null, 2);
    document.body.appendChild(span);

    socket.emit('my other event', {my:'data'});
  });
</script>
```

index.html - 10 lines of HTML

A Simple Chat Application

Server

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

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

io.sockets.on('connection', function (socket) {
  socket.on('msg', function (data) {
    io.sockets.emit('new', data);
  });
});

server.listen(1337);
console.log('Server running at http://localhost:1337/');
```

server.js - 12 lines of JavaScript

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Simple Chat</title>
    <link rel="stylesheet",type="text/css" href="styles.css"/>
    <script type="text/javascript" src="/socket.io/socket.io.js"></script>
    <script type="text/javascript"
      src="https://ajax.googleapis.com/ajax/libs/jquery/1.8.2/jquery.min.js">
    </script>
    <script type="text/javascript" src="chat.js"></script>
  </head>
  <body>
    <div id="wrapper">
      <h1>Simple Chat</h1>
      <div id="messages"></div>
      <div class="nic">
        Your Name
        <input id="name" name="name" type="text"/>
      </div>
      <textarea id="message"></textarea>
      <input id="send" type="submit" value="Send"/>
    </div>
    <script type="text/javascript">
      $(document).ready(function() {
        Chat.initialize('http://localhost:1337/');
      });
    </script>
  </body>
</html>
```

public/index.html - 29 lines of HTML

Client-side Chat Module

```
(function () {
  window.Chat = {
    socket : null,

    initialize: function(socketURL) {
      // initialize socket.io
    },

    add: function(data) {
      // add message to window
    },

    send: function() {
      // send message
    }
  };
}());
```

public/chat.js - 14 lines of JavaScript

```
(function () {
  window.Chat = {
    //...
    initialize: function(socketURL) {
      this.socket = io.connect(socketURL);

      $('#send').click(function() {
        Chat.send();
      });

      $('#message').keyup(function(evt) {
        if ((evt.keyCode || evt.which) == 13) {
          Chat.send();
          return false;
        }
      });

      this.socket.on('new', this.add);
    },
    //...
  };
}());
```

public/chat.js - 19 lines of JavaScript

```
(function () {
  window.Chat = {
    //...
    add: function(data) {
      var name = data.name || 'anonymous';
      var msg = $('<div class="msg"></div>')
        .append('<span class="name">' + name + '</span>: ')
        .append('<span class="text">' + data.msg + '</span>');

      $('#messages')
        .append(msg)
        .animate({scrollTop: $('#messages').prop('scrollHeight')}, 0);
    },
    //...
  };
}());
```

public/chat.js - 15 lines of JavaScript

```
(function () {
  window.Chat = {
    //...
    send: function() {
      this.socket.emit('msg', {
        name: $('#name').val(),
        msg: $('#message').val()
      });

      $('#message').val('');

      return false;
    },
    //...
  };
}());
```

public/chat.js - 14 lines of JavaScript

Questions?