Node.js

Node.js Overview

- JavaScript runtime environment based on Chrome V8 JavaScript engine
  - Allows JavaScript to run on any computer
    - JavaScript everywhere! On browsers and servers!
  - Intended to run directly on OS, not inside a browser
    - Removes browser-specific JavaScript API
    - Adds support for OS APIs such as file system and network
  - Cross-platform, runs on Linux, Windows, macOS, Solaris, ...
  - Node package manager (NPM) to manage package dependency and installation
- Node.js is *single threaded*
  - No overhead from multi-threading and leads to high-performance
  - But requires asynchronous programming
- Node interactive JavaScript shell

```
$ node
> console.log("Hello world");
Hello world
undefined
> .help
.break     Sometimes you get stuck, this gets you out
.clear     Alias for .break
.editor    Enter editor mode
.exit      Exit the repl
.help      Print this help message
.load      Load JS from a file into the REPL session
.save      Save all evaluated commands in this REPL session to
   a file
> .exit
```

- Example: reply with "Hello world" to any HTTP request on port 3000

```
// ---------- app.js ----------
let http = require("http");

// Create HTTP server and listen on port 3000 for requests
http.createServer((request, response) => {
    response.writeHead(200, {'Content-Type': 'text/plain'});
    response.write("Hello World!\n");
    response.end("URL: " + request.url);
}).listen(3000);
```

```
$ node app.js
```

- Node.js is ***single-threaded***
- Asynchronous-programming
  - Passes a callback, which will be invoked when the server receives a request
  - Nonblocking call
- More on this later

- Node.js everywhere!
  - Node is not a server. It is a general JavaScript runtime platform
  - Node.js is now used for desktop app development, not just for servers
  - Example: Electron, AppJS, . . .

Node package manager (NPM)

- Package manager to help installing and managing Node.js "packages" (or modules)

```
$ npm install express
```

  - Install package locally in the `node_modules/` subdirectory

- `package.json`: a file created in the app root folder to describe package dependency

```
$ npm init -y
$ npm install --save express
```

  - `npm init -y` creates default `package.json`
  - `--save` option adds the installed package to `package.json`

```
// --- package.json ----
{
  "name": "application-name",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
  "dependencies": {
    "express": "4.9.0",
    "cookie-parser": "~1.3.3",
    "ejs": "^1.6.0"
  }
}
```

  - Version: major.minor.patch
    * ~ any patch version is fine
    * ^ an equal or higher version is fine if the major version is the same
  - `npm install` installs all dependencies listed in the file in `node_modules`

- Global mode

```
$ npm install -g nodemon
$ nodemon app.js
```

  - Package `nodemon` supports automatic restart of node after code change

- Global vs local?
  - If a package is used *in the program*, using `require('whatever')`, install it locally
  - If a package is used *in the shell or command line* during development, install it globally

Express ("express")

- A popular node framework for Web server development

  - Provides URL-routing mechanism
  - Integrates HTML *template engine*
  - Integrates *middleware* for complex request handling

- First example

```
// -------- app.js ---------
let express = require('express');
let app = express();

app.get('/', (req, res) => {
    res.send('Hello World!');
});
app.get('/john', (req, res) => {
    res.send('Hello, John!');
});

app.listen(3000);
```

```
$ node app.js
```

  - `app.get(path, callback)` maps `GET` request at `path` to `callback` function
  - `res.send(body)` returns `body` as the response to the request
  - `app.listen(port)` binds to `port`
  - Q: What does the above server do? What URLs does it handle? What response does it generate?

  - Q: Should I install `express` locally or globally?

```
$ npm init -y
$ node install --save express
```

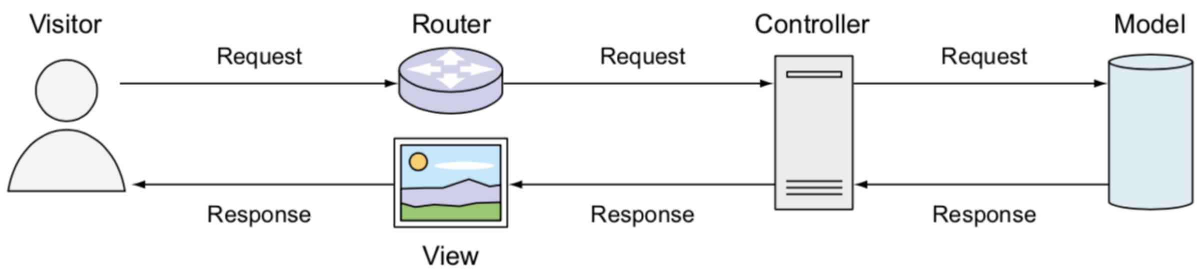

Figure 1: Express-MVC

- Express supports MVC pattern
- Q: How do I map a path to a "controller/function"?

Routing support

- `express.method(path, callback)`
  - Invoke `callback` for a request with `method` and `path`
  - Path interprets `?`, `+`, `*`, and `()` as regular expression operators
    - Exact syntax at https://www.npmjs.com/package/path-to-regexp
  - Example

```
app.get('/list/:username', (req, res) => {
    res.send("Your username is " + req.params.username);
});
```

  - Inside callback, the HTTP request is available in the first parameter `req`:
    - `req.query`: query strings in the URL
    - `req.params`: "/:xxx" parameters in the URL
    - `req.body`: request body
    - `req.cookies`: cookies
  - If a request does not have any matching path, express responds with status "404 Not Found"

View generation

- Response code: `response.status(200)`
- Body: `send()`, `sendFile()`, `render()`, `json()`
  - Raw string: `response.send(body)`
    * Send the string `body` as the response
    * Example: `res.send("Hello, world!")`
  - Static file: `response.sendFile(absolute_path_to_file)`
    * Send a static file from local filesystem
    * Example: `res.sendFile('/User/cho/public/index.html')`
  - JSON: `response.json(object)`
    * Encode JavaScript object `object` in JSON
    * Example: `res.json({title: "Hello", body: "_Love_"})`
  - HTML from template: `response.render(template-file, template-data)`
    * Generate an HTML page from `template-file` using `template-data`
    * *Template engine*: different template engines may be used
      - Examples: Pug (previously Jade), EJS, Mustache, . . .
    * Example: `res.render('index', {title: "Hello"})`
- Redirect: `response.redirect(URL)`
  - Send redirect response
  - Example: `res.redirect('/')`

EJS: Example template engine ("ejs")

- A popular template engine used with Express
  - *Scriptlet tag*
  - `<% ... %>`: javascript for control-flow. no output
  - `<%= ... %>`: print out the result of expression (after HTML escaping)
  - `<%- ... %>`: print out the result (raw output. No HTML escaping)
- Example

```
// ----- index.ejs ------
<!DOCTYPE html>
<html
<head><title><%= title %></title></head>
<body>
```

```
<ul>
<% for (let post of posts) { %>
    <li><%= post.title %></li>
<% } %>
</ul>
</body>
```

```
// ----  app.js ------
let express = require('express');
let app = express();

app.set('view engine', 'ejs');
app.set('views', '.')
app.get('/', (req, res) => {
    res.render('index',
      { title: "Hello",
        posts: [{title: "Title 1"}, {title: "Title 2"}]
      }
    );
});
app.listen(3000);
```

  - To run the above code, we need to install `ejs` module: `npm install --save ejs`

Middleware integration

- `express.use([path,] middleware)` attaches a "`middleware`" to the request handling chain
  - Middleware may take an action based on the request, "transform" the request, serve the response to the request, etc
  - Every request goes through the sequence of middleware, until one serves the response
  - If `path` is specified, the middleware runs only on the request within `path`
- Example

```
let express = require('express');
let path = require('path');
let bodyParser = require('body-parser');

let app = express();

app.use(bodyParser.json());
app.use('/www/', express.static(path.join(__dirname, 'public')
   ));
app.get('/', (req, res) => {
    res.send("Welcome to our Application!");
})
app.listen(3000);
```

- `express.static(absolute_path_to_root_dir)`
    * Middleware for serving static files from a directory
    * Multiple static asset directories can be specified for the same path. The first match is returned
- bodyParser
    * collection of HTTP body parsing middleware

• Creating middleware
- `express.Router()`: creates a modular, mountable route handler inside a middleware
    * Using `router.METHOD(path, callback)`, we can specify further route mapping inside a middleware
    * *Mini app*: a router can be used like a middleware and can be passed as a parameter to `express.use()`
- Example:

```
// ---- app.js --------
let express = require('express');
let birds = require('./birds');

let app = express();
```

```
app.use('/birds', birds);
app.get('/', (req, res) => {
    res.send("Welcome to Birds App!");
})
app.listen(3000);

// ---- birds.js -------
let express = require('express');
let router = express.Router();

router.get('/', function (req, res) {
  res.send('Birds home page');
})
router.get('/about', function (req, res) {
  res.send('About birds');
})

module.exports = router;
```

* Q: What will be returned for `/birds`? `/birds/about`?

Express application generator

- Express application generator can be used to generate a skeleton code for express-based server

```
$ express -e    // generate skeleton code with EJS template
   engine
$ npm install   // install dependent modules
$ npm start     // execute "start" script in package.json
```

```
$ ls
app.js           package-lock.json   routes/
bin/             package.json        t.js
node_modules/    public/             views/
```

- `app.js`: main application
- `public/`: folder for static files
- `routes/`: route handlers
- `views/`: views

```
// app.js ----
var index = require('./routes/index');
var users = require('./routes/users');

var app = express();

// middleware
app.use(logger('dev'));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

app.use('/', index);
app.use('/users', users);

// catch 404 and forward to error handler
app.use(function(req, res, next) {
  var err = new Error('Not Found');
  err.status = 404;
  next(err);
});
```

```
// routes/index.js
var express = require('express');
var router = express.Router();

/* GET home page. */
router.get('/', function(req, res, next) {
  res.render('index', { title: 'Express' });
});
```

```
module.exports = router;
```