ExpressJS

Mendel Rosenblum

Express.js - A web framework for Node.js

Fast, unopinionated, minimalist web framework

Relatively thin layer on top of the base Node.js functionality

What does a web server implementor need?

- **Speak HTTP**: Accept TCP connections, process HTTP request, send HTTP replies

  Node's HTTP module does this
- **Routing**: Map URLs to the web server function for that URL

  Need to support a routing table (like ngRoute in AngularJS or React Router)
- **Middleware support**: Allow request processing layers to be added in

  Make it easy to add custom support for sessions, cookies, security, compression, etc.

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

var expressApp = express();

    // expressApp object has methods for:
```

- Routing HTTP requests
- Rendering HTML (e.g. run a preprocessor like Jade templating engine)
- Configuring middleware and preprocessors

```
expressApp.get('/', function (httpRequest, httpResponse) {
  httpResponse.send('hello world');
});

expressApp.listen(3000);
```

Express routing

- By HTTP method:

```
expressApp.get(urlPath, requestProcessFunction);
expressApp.post(urlPath, requestProcessFunction);
expressApp.put(urlPath, requestProcessFunction);
expressApp.delete(urlPath, requestProcessFunction);
expressApp.all(urlPath, requestProcessFunction);
```

- Many others less frequently used methods
- `urlPath` can contain parameters like ngRoute (e.g. `'/user/:user_id'`)

httpRequest object

```
expressApp.get('/user/:user_id', function (httpRequest, httpResponse) ...
```

- Object with large number of properties

Middleware (like JSON body parser, session manager, etc.) can add properties

`request.params` - Object containing url route params (e.g. `user_id`)
`request.query` - Object containing query params (e.g. &foo=9 ⇒ {foo: '9'})
`request.body` - Object containing the parsed body
`request.get(field)` - Return the value of the specified HTTP header field

httpResponse object

```
expressApp.get('/user/:user_id', function (httpRequest, httpResponse) …
```

- Object with a number of methods for setting HTTP response fields

`response.write(content)` - Build up the response body with `content`
`response.status(code)` - Set the HTTP status code of the reply
`response.set(prop, value)` - Set the response header property to value
`response.end()` - End the request by responding to it
`response.end(msg)` - End the request by responding with msg
`response.send(content)` - Do a write() and end()

- Methods return the response object so they stack (i.e. `return this;`)

```
response.status(code).write(content1).write(content2).end();
```

Middleware

- Give other software the ability to interpose on requests

```
expressApp.all(urlPath, function (request, response, next) {
    // Do whatever processing on request (or setting response)
    next(); // pass control to the next handler
});
```

- Interposing on all request using the route mechanism

```
expressApp.use(function (request, response, next) {...});
```

- Examples:
  - Check to see if user is logged in, otherwise send error response and don't call next()
  - Parse the request body as JSON and attached the object to request.body and call next()
  - Session and cookie management, compression, encryption, etc.

ExpressJS Example: webServer.js from Project #4

```
var express = require('express');
var app = express();                          // Creating an Express "App"

app.use(express.static(__dirname));           // Adding middleware

app.get('/', function (request, response) {     // A simple request handler
   response.send('Simple web server of files from ' + __dirname);
});

app.listen(3000, function () {    // Start Express on the requests
  console.log('Listening at http://localhost:3000 exporting the directory ' +
              __dirname);
});
```

ExpressJS Example: webServer.js from Project #5

```
app.get('/user/list', function (request, response) {
  response.status(200).send(cs142models.userListModel());
  return;
});

app.get('/user/:id', function (request, response) {
  var id = request.params.id;
  var user = cs142models.userModel(id);
  if (user === null) {
    console.log('User with _id:' + id + ' not found.');
    response.status(400).send('Not found');
    return;
  }
  response.status(200).send(user);
  return;
});
```

A Simple Model Fetcher - Fetch from a JSON file

```
expressApp.get("/object/:objid", function (request, response) {
  var dbFile = "DB" + request.params.objid;
  fs.readFile(dbFile, function (error, contents) {
    if (error) {
      response.status(500).send(error.message);
    } else {
      var obj = JSON.parse(contents);  // JSON.parse accepts Buffer types
      obj.date = new Date();
      response.set('Content-Type', 'application/json'); // Same: response.json(obj);
      response.status(200).send(JSON.stringify(obj));
    }
  });
}
```

Note: Make sure you always call end() or send()

A Simple Model Fetcher - Fetch from a JSON file

```
expressApp.get("/object/:objid", function (request, response) {
  var dbFile = "DB" + request.params.objid;
  fs.readFile(dbFile, function (error, contents) {
    if (error) {
      response.status(500).send(error.message);
    } else {
      var obj = JSON.parse(contents);  // JSON.parse accepts Buffer types
      obj.date = new Date();
      response.json(obj);
    }
  });
}
```

Fetching multiple models - Comments of objects

```
app.get("/commentsOf/:objid", function (request, response) {
   var comments = [];
   fs.readFile("DB" + request.params.objid, function (error, contents) {
        var obj = JSON.parse(contents);
        async.each(obj.comments, fetchComments, allDone);
    });
    function fetchComments(commentFile, callback) {
       fs.readFile("DB"+ commentFile, function (error, contents) {
            if (!error) comments.push(JSON.parse(contents));
            callback(error);
       });
     }
     function allDone(error) {
        if (error) responses.status(500).send(error.message); else response.json(comments);
     }
});
```