EXPRESS.JS

Express.js (Express) is a mature web application framework for Node.js built upon Connect.

The Express.js convention is to define the server instance as `app`.

01_hello_express.js:

```
'use strict';

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

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

app.listen(3000, function onListening () {
  console.log('Listening at http://localhost:3000');
});
```

```
$ npm install express connect
```

Express is built upon Connect.

02_hello_connect.js:

```
'use strict';

var http = require('http'),
    connect = require('connect'),
    app = connect();

app.use(function hello(req, res, next) {
  res.end("Hello, Connect.");
});

http.createServer(app).listen(3000);
```

ENVIRONMENT CONFIGURATIONS

DEVELOPMENT CONFIGURATION

```
$ node .
```

Equivalent to:

```
$ NODE_ENV=development node index.js
```

ENVIRONMENT CONFIGURATIONS

PRODUCTION CONFIGURATION

When running Express in production, it is essential to run Express in production mode.

```
$ export NODE_ENV=production
$ node index.js
```

For simplicity's sake:

```
$ NODE_ENV=production node index.js
```

CONFIGURATION

INTRODUCING NCONF

```
$ npm install nconf
```

NCONF: SIMPLIFY CONFIGURATION SOURCES

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

// Load nconf configs
nconf
    .argv()
    .env()
    .file({ file: 'config.json' });

nconf.defaults({
  'port': 1337
});
```

05_express_nconf.js:

```
'use strict';

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

// Load nconf configs
nconf
    .argv()
    .env()
    .file({ file: 'config.json' });

nconf.defaults({
   'port': 1337
});

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

app.listen(nconf.get('port'), function onListening () {
  console.log('Listening at http://localhost:%d', nconf.get('port'));
});
```

MIDDLEWARE

```
var simpleLoggingMiddleware = function (req, res, next) {
  console.log(req.method, req.url);
  next();
};

app.use(simpleLogUrlMiddleware);

app.use(function(req, res, next) {
  res.end("Hello, Connect or Express.");
});
```

03_connect_middleware.js

```
/// simple logging middleware

var simpleLoggingMiddleware = function (req, res, next) {
  console.log(req.method, req.url);
  next();
};

app.use(simpleLoggingMiddleware);

app.use(function hello(req, res, next) {
  res.end("Hello, Connect.");
});

http.createServer(app).listen(3000);
```

04_express_middleware.js

```
/// simple logging middleware in Express

var simpleLoggingMiddleware = function (req, res, next) {
  console.log(req.method, req.url);
  next();
};

// Register the middleware
app.use(simpleLoggingMiddleware);

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

app.listen(3000, function onListening () {
  console.log('Listening at http://localhost:3000');
});
```

BY DEFAULT EXPRESS REGISTERS TWO MIDDLEWARE

```
// implicit middleware
this.use(connect.query());
this.use(middleware.init(this));
```

ORDER MATTERS

Middleware is registered in call order. Consider each `app.use` a call to `array.push()`.

From Connect `lib/proto.js`.

```
// add the middleware
debug('use %s %s', route || '/', fn.name || 'anonymous');
this.stack.push({ route: route, handle: fn });
```

EXPRESS MIDDLEWARE

ORDER MATTERS

express_middleware_order.js

```
/// Register the middleware
app.use(function func1(req, res, next) {
  console.log('func1', req.method, req.url);
  next();
});

app.use(function func2(req, res, next) {
  console.log('func2', req.url);
  next();
});

app.use(function func3(req, res, next) {
  console.log('func3', req.query);
  next();
});

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

app.listen(3000, function onListening() {
  console.log('Listening at http://localhost:3000');
});
```

ROUTES

The popularity of Express.js is largely due to the simple clear API for defining routes and http method handlers.

The simple app.VERB() syntax has been copied by other frameworks such as Restify.

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

Registers a handler for `GET /`.

USING A CONTROLLER

Route handlers quickly become overwhelming, making it hard to organize. Therefore, routes are typically broken down into groups of controller functions.

express_routes_controllers.js

```
/// controllers
var admin = require('./controller_admin'),
    users = require('./controller_users');

// Pass in Express app and augment with routes.
admin(app);
users(app);


app.listen(3000, function onListening () {
  console.log('Listening at http://localhost:3000');
});
```

controller_users.js

```
'use strict';

module.exports = function (server) {

  var users = 'bill jeff erik'.split(' ');

  server.get('/users', function (req, res) {
    res.send(users.join('\n'));
  });

  // Add a user: `curl --data 'user=dshaw' http://localhost:3000/users`
  server.post('/users', function (req, res) {
    if (req.body.user && req.body.user.length > 0) {
      users.push(req.body.user);
      res.send(201);
    } else {
      res.send(400);
    }
  });

};
```

controller_admin.js

```
'use strict';

var requireAuth = require('./require-auth');

module.exports = function adminController(server) {

  server.use('/admin', requireAuth);

  server.get('/admin', function (req, res) {
    res.send('Hello, Admin!');
  });

};
```

EXPLORING THE CONTROLLER FUNCTIONALITY

```
$ curl http://localhost:3000/admin

$ curl http://localhost:3000/users

$ curl --data 'user=dshaw' http://localhost:3000/users

$ curl http://localhost:3000/users
```

APP.ALL

Use when you need send a response to all possible HTTP methods.

```
app.all('/', function hello(req, res){
  res.send('Hello, Everybody!');
});
```

EXPRESS.JS SUPPORTED HTTP METHODS

Let's see what we can use.

METHODS

HTTP methods that node supports

```
$ npm install methods

$ node -p "require('methods')"
```

HANDLING ERRORS

Register error handling middeware.

express_handle_errors.js

```
/// Error handling middleware - must come last in routing setup

// Handle 404
app.use(function(req, res) {
  res.send('404: Not Found', 404);
});

// Handle 500
app.use(function(error, req, res, next) {
  res.send('500: Internal Server Error\n' + error, 500);
});
```

ADDING TEMPLATES

INTRODUCING `RES.RENDER()`

USING DUST.JS

```
$ npm install dustjs-linkedin dustjs-helpers consolidate
```

express_templates.js

```
'use strict';

var express = require('express'),
    dust = require('dustjs-helpers'),
    cons = require('consolidate'),
    app = express();

// Register the dust engine to handle .dust files
// consolidate will look for dust and normalize the renderer signature f
app.engine('dust', cons.dust);
app.set('view engine', 'dust');

app.get('/', function(req, res) {
  res.render('index', {
    title: 'Hello, Dust.js'
  });
});

app.listen(3000, function onListening () {
  console.log('Listening at http://localhost:3000');
});
```

views/index.dust

```
{>layout/}
{<content}
    Additional content from a partial.
{/content}
```

views/layout.dust

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>{title}</title>
    </head>
    <body>
        <h1>{title}</h1>
        {+content}
            Base content.
        {/content}
    </body>
</html>
```

```
$ open http://localhost:3000
```

HANDLING SESSIONS

- Connect's **session** middleware attaches a `session` object to each `request`
- Sessions are identified by a unique cookie
- Session data is stored on the server, not in the cookie

express_sessions.js

```
'use strict';

var express = require('express'),
    cons = require('consolidate'),
    app = express();

require('dustjs-helpers');
app.engine('dust', cons.dust);
app.set('view engine', 'dust');

app.use(express.cookieParser());
app.use(express.session({ secret: 'adorable seamonster' }));

app.get('/', function(req, res) {

  // increment viewCount if it exists, set it to 1 if it doesn't
  req.session.viewCount = (req.session.viewCount || 0) + 1;

  res.render('sessions', {
    viewCount: req.session.viewCount
  });

});

app.listen(3000, function onListening () {
  console.log('Listening at http://localhost:3000');
});
```

Uses `connect.session.MemoryStore` by default

THE PROBLEM WITH IN-MEMORY SESSIONS

```
$ NODE_ENV=production node express_sessions.js

Warning: connection.session() MemoryStore is not
designed for a production environment, as it will leak
memory, and will not scale past a single process.
Listening at http://localhost:3000
```

express_clustered_sessions.js

```
'use strict';

var cluster = require('cluster');

function createServer () {

  /// createServer() to start a worker process

}

if (cluster.isMaster) {
  // start two workers
  cluster.fork();
  cluster.fork();

  // start a new worker when one dies
  cluster.on('exit', function(worker, code, signal) {
    console.log('worker %d died (%s). restarting...', worker.process.pid
    cluster.fork();
  });
} else {
  createServer();
}
```

express_clustered_sessions.js

```
/// createServer() to start a worker process
var express = require('express'),
    cons = require('consolidate'),
    app = express(),
    requests = 0; // keep track of the number of requests

require('dustjs-helpers');
app.engine('dust', cons.dust);
app.set('view engine', 'dust');

app.use(express.cookieParser());
app.use(express.session({ secret: 'adorable seamonster' }));

app.get('/', function(req, res) {
  if (requests++ == 5) // after 5 requests, this child worker dies
    return process.exit(0);

  req.session.viewCount = (req.session.viewCount || 0) + 1;

  console.log('Request handled by PID', process.pid, 'view count =', r

  res.render('sessions', {
    viewCount: req.session.viewCount
  });
});

app.listen(3000, function onListening () {
  console.log(process.pid, 'Listening at http://localhost:3000');
});
```

SCALING SESSIONS

- Use shared-memory: Redis or some other data store
- Session data is saved to the store when updated
- Session data is fetched from the store for each request

express_clustered_redis_sessions.js — requires Redis installed on localhost

```
/// createServer() to start a worker process
var express = require('express'),
    cons = require('consolidate'),
    RedisStore = require('connect-redis')(express), // pass express to
    app = express(),
    requests = 0;

require('dustjs-helpers');
app.engine('dust', cons.dust);
app.set('view engine', 'dust');

app.use(express.cookieParser());
app.use(express.session({ store: new RedisStore(), secret: 'adorable s

app.get('/', function(req, res) {
  if (requests++ == 5)
    return process.exit(0);

  req.session.viewCount = (req.session.viewCount || 0) + 1;

  console.log('Request handled by PID', process.pid, 'view count =', r

  res.render('sessions', {
    viewCount: req.session.viewCount
  });
});

app.listen(3000, function onListening () {
  console.log(process.pid, 'Listening at http://localhost:3000');
});
```

EXPRESS BUILT IN DEBUGGING

Express.js is instrumented with TJ Hollowaychuk's `[debug]` (`https://github.com/visionmedia/debug`) module.

- Set the environment variable `DEBUG` for verbose information about application mode, routing and more.
- Use `express:*` to log everything. Limit as appropriate to `express:application` or `express:router`.

```
$ DEBUG=express:* node express_handle_errors.js
```

```
express:application booting in development mode +0ms
express:router defined get / +0ms
express:router defined get /error +1ms
```

SUMMARY

- Express.js is a web application framework built on Connect.
- Environment configuration is important.
- Composing routes is easy and expressive.
- Be careful not to go too deep with your middleware chain.