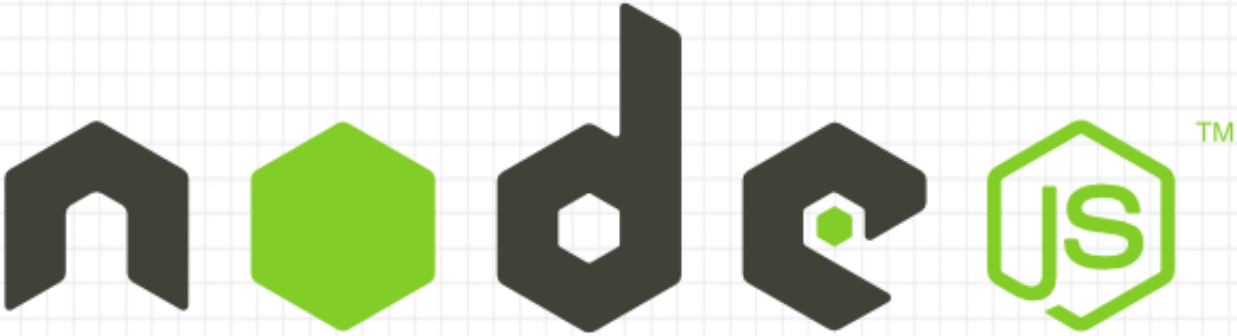


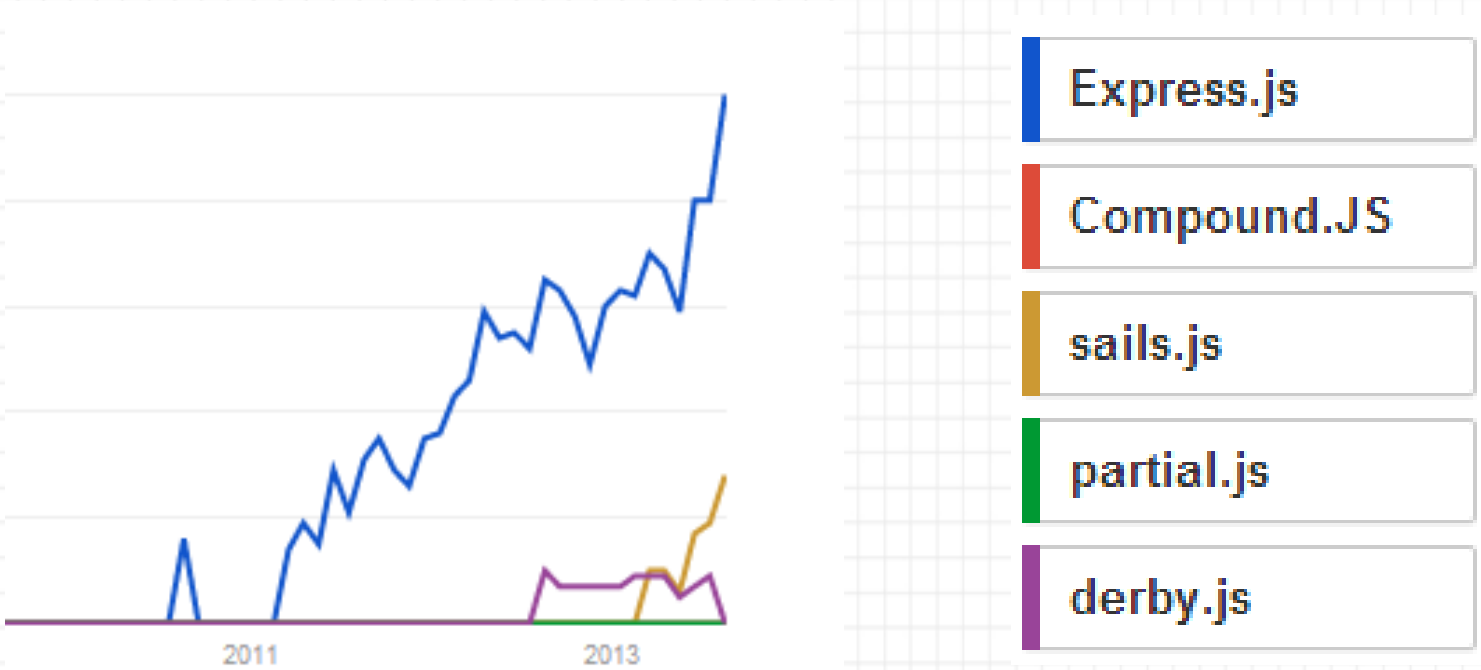
Express Web Application Framework for Node.js

Agenda

- Why Express.js?
- Middleware
- Routing
- Configuration
- Views & Templates

Why Express.js ?

- Providing a robust set of features for building web app's.
  - Configuration
  - Routing
  - Middleware
  - Views & Templates
  - Session
  - Security



Without Express

```
// Require what we need
var http = require("http");

// Build the server
var app = http.createServer(
      function (request, response) {
            response.writeHead(200, {
                  "Content-Type": "text/plain"
            });
            response.end("Hello world!");
      });

// Start that server
app.listen(1337, "localhost");
console.log("Server running at http://localhost:1337/");
```

Pipe Code

How to Route Without Express

```
var http = require("http");
http.createServer(function (req, res) {

    // Homepage
    if (req.url == "/") {
        res.writeHead(200, { "Content-Type": "text/html" });
        res.end("Welcome to the homepage!");
    }// About page
    else if (req.url == "/about") {
        res.writeHead(200, { "Content-Type": "text/html" });
        res.end("Welcome to the about page!");
    }
    // 404'd!
    else {
        res.writeHead(404, { "Content-Type": "text/plain" });
        res.end("404 error! File not found.");
    }

}).listen(1337, "localhost");
```

How to Route With Express

```
//Homepage
app.get('/', function (req, res) {
    res.send("Welcome to the homepage!");
});

// About page
app.get('/about', function (req, res) {
     res.send("Welcome to the about page!");
});
```

Express Routing

Routing

- Routing refers to the definition of end points (URIs) to an application and how it responds to client requests.

- A route is a combination of a URI, a HTTP request method (GET, POST, and so on), and one or more handlers for the endpoint.
- It takes the following structure

  app.METHOD(path, [callback...], callback),

- where app is an instance of express, METHOD is an HTTP request method, path is a path on the server, and callback is the function executed when the route is matched.

Routing

The following is an example of a very basic route.

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

// respond with "hello world" when a GET request is made to the homepage
app.get('/', function(req, res) {
  res.send('hello world');
});
```

Route Methods

A route method is derived from one of the HTTP methods, and is attached to an instance of `express`.

The following is an example of routes defined for the GET and the POST methods to the root of the app.

```
// GET method route
app.get('/', function (req, res) {
  res.send('GET request to the homepage');
});

// POST method route
app.post('/', function (req, res) {
  res.send('POST request to the homepage');
});
```

Route Methods –all()

- There is a special routing method, app.all(), which is not derived from any HTTP method.
- It is used for loading middleware at a path for all request methods.
- In the following example, the handler will be executed for requests to "/secret" whether using GET, POST, PUT, DELETE, or any other HTTP request method supported in the http module.

```
app.all('/secret', function (req, res, next) {
  console.log('Accessing the secret section ...');
  next(); // pass control to the next handler
});
```

Route Paths

- Route paths, in combination with a request method, define the endpoints at which requests can be made to.
- They can be strings, string patterns, or regular expressions.
- Query strings are not a part of the route path.

Examples of route paths based on strings:

```
// will match request to the root
app.get('/', function (req, res) {
  res.send('root');
});

// will match requests to /about
app.get('/about', function (req, res) {
  res.send('about');
});
```

Route Paths – string pattern

```
// will match acd and abcd
app.get('/ab?cd', function(req, res) {
  res.send('ab?cd');
});

// will match abcd, abbcd, abbbcd, and so on
app.get('/ab+cd', function(req, res) {
  res.send('ab+cd');
});

// will match abcd, abxcd, abRABDOMcd, ab123cd,
app.get('/ab*cd', function(req, res) {
  res.send('ab*cd');
});

// will match /abe and /abcde
app.get('/ab(cd)?e', function(req, res) {
 res.send('ab(cd)?e');
});
```

Route Paths – RegularExp

```
// will match anything with an a in the route name:
app.get(/a/, function(req, res) {
  res.send('/a/');
});

// will match butterfly, dragonfly; but not butterflyman,
app.get(/.*fly$/, function(req, res) {
  res.send('/.*fly$/');
});
```

Route Handlers – multiple callbacks

- A route can be handled using a more than one callback function (make sure to specify the next object):

```
app.get('/example/b', function (req, res, next) {
  console.log('response will be sent by the next function ...');
  next();
}, function (req, res) {
  res.send('Hello from B!');
});
```

Route Handlers – array of callbacks

```
var cb0 = function (req, res, next) {
  console.log('CB0');
  next();
}

var cb1 = function (req, res, next) {
  console.log('CB1');
  next();
}

var cb2 = function (req, res) {
  res.send('Hello from C!');
}

app.get('/example/c', [cb0, cb1, cb2]);
```

Route Methods

Method	Description
res.download()	Prompt a file to be downloaded.
res.end()	End the response process.
res.json()	Send a JSON response.
res.jsonp()	Send a JSON response with JSONP support.
res.redirect()	Redirect a request.
res.render()	Render a view template.
res.send()	Send a response of various types.
res.sendFile	Send a file as an octet stream.
res.sendStatus()	Set the response status code and send its string representation as the response body.

Chainable route handlers

- Chainable route handlers for a route path can be created using app.route()
- Create modular routes and reduce redundancy and typos

```
app.route('/book')
  .get(function(req, res) {
    res.send('Get a random book');
  })
  .post(function(req, res) {
    res.send('Add a book');
  })
  .put(function(req, res) {
    res.send('Update the book');
  });
```

Express.Router

- The express.Router class can be used to create modular mountable route handlers.
- A Router instance is a complete middleware and routing system; for this reason it is often referred to as a “mini-app”.
- The following example creates a router as a module, loads a middleware in it, defines some routes, and mounts it on a path on the main app.

Express.Router

```
var express = require('express');
var router = express.Router();

// middleware specific to this router
router.use(function timeLog(req, res, next) {
  console.log('Time: ', Date.now());
  next();
});
// define the home page route
router.get('/', function(req, res) {
  res.send('Birds home page');
});
// define the about route
router.get('/about', function(req, res) {
  res.send('About birds');
});

module.exports = router;
```

Express.Router

Then, load the router module in the app:

```
var birds = require('./birds');
...
app.use('/birds', birds);
```

The app will now be able to handle requests to `/birds` and `/birds/about`, along with calling the `timeLog` middleware specific to the route.

Express Routing

```
app.get('/users/:id?', function (req, res, next)
{
    var id = req.params.id;
    if (id) {
    // do something
    } else {
        next();
    }
});
```

```
{
    path: '/user/:id?',
    method: 'all' | 'get' | 'post' | 'put' | 'delete',
    callbacks: [ [Function] ],
    keys: [ { name: 'id', optional: true } ],
    regexp: /^\/user(?:\/([^\/]+?))?\/?$/i,
    params: [ id: '12' ]
}
```

demo

Routing

Routing Templates

- One to one

```
app.get('/users/:id?', function (req, res, next) { ... }
```

- One to many

```
'/:controller/:action/:id'
```

demo

Routing

Using Middleware

What is middleware?

- Middleware is a request handler.
  - Routing , Controller, Models, Views… Security


```
function myFunMiddleware(request, response, next) {
    // Do stuff with the request and response.
    // When we're all done, call next() to defer
    // to the next middleware.
    next();
}
```

What is middleware?

- Express is a routing and middleware web framework with minimal functionality of its own:
- An Express application is essentially a series of middleware calls.
- Middleware is a function with access to the request object (req), the response object (res), and the next middleware in the application's request-response cycle, commonly denoted by a variable named next.

What is middleware?

Middleware can:

- Execute any code.
- Make changes to the request and the response objects.
- End the request-response cycle.
- Call the next middleware in the stack.

- If the current middleware does not end the request-response cycle, it must call next() to pass control to the next middleware, otherwise the request will be left hanging.

- An Express application can use the following kinds of middleware:

- Application-level middleware
- Router-level middleware
- Error-handling middleware
- Built-in middleware
- Third-party middleware

Application-level middleware

- Bind application-level middleware to an instance of the app object with app.use() and app.METHOD(), where METHOD is is the HTTP method of the request that it handles, such as GET, PUT, POST, and so on, in lowercase

```
var app = express();

// a middleware with no mount path; gets executed for every request to the app
app.use(function (req, res, next) {
  console.log('Time:', Date.now());
  next();
});

// a middleware mounted on /user/:id; will be executed for any type of HTTP request to /user/:id
app.use('/user/:id', function (req, res, next) {
  console.log('Request Type:', req.method);
  next();
});

// a route and its handler function (middleware system) which handles GET requests to /user/:id
app.get('/user/:id', function (req, res, next) {
  res.send('USER');
});
```

Application-level middleware

Here is an example of loading a series of middleware at a mount point with a mount path:

```
// a middleware sub-stack which prints request info for any type of HTTP request to /user/:id
app.use('/user/:id', function(req, res, next) {
  console.log('Request URL:', req.originalUrl);
  next();
}, function (req, res, next) {
  console.log('Request Type:', req.method);
  next();
});
```

```
// a middleware sub-stack which handles GET requests to /user/:id
app.get('/user/:id', function (req, res, next) {
  console.log('ID:', req.params.id);
  next();
}, function (req, res, next) {
  res.send('User Info');
});
```

Application-level middleware

```
// a middleware sub-stack which handles GET requests to /user/:id
app.get('/user/:id', function (req, res, next) {
  // if user id is 0, skip to the next route
  if (req.params.id == 0) next('route');
  // else pass the control to the next middleware in this stack
  else next(); //
}, function (req, res, next) {
  // render a regular page
  res.render('regular');
});

// handler for /user/:id which renders a special page
app.get('/user/:id', function (req, res, next) {
  res.render('special');
});
```

Router-level middleware

- Router-level middleware works just like application-level middleware except it is bound to an instance of express.Router().

```
var app = express();
var router = express.Router();

// a middleware with no mount path, gets executed for every request to the router
router.use(function (req, res, next) {
  console.log('Time:', Date.now());
  next();
});

// a middleware sub-stack shows request info for any type of HTTP request to /user/:id
router.use('/user/:id', function(req, res, next) {
  console.log('Request URL:', req.originalUrl);
  next();
}, function (req, res, next) {
  console.log('Request Type:', req.method);
  next();
});
```

Router-level middleware

```
// a middleware sub-stack which handles GET requests to /user/:id
router.get('/user/:id', function (req, res, next) {
  // if user id is 0, skip to the next router
  if (req.params.id == 0) next('route');
  // else pass the control to the next middleware in this stack
  else next(); //
}, function (req, res, next) {
  // render a regular page
  res.render('regular');
});

// handler for /user/:id which renders a special page
router.get('/user/:id', function (req, res, next) {
  console.log(req.params.id);
  res.render('special');
});

// mount the router on the app
app.use('/', router);
```

Error -Handling middleware

- Error-handling middleware always takes four arguments. You must provide four arguments to identify it as an error-handling middleware. Even if you don't need to use the next object, you must specify it to maintain the signature, otherwise it will be interpreted as regular middleware and fail to handle errors.

- Define error-handling middleware like other middleware, except with four arguments instead of three, specifically with the signature (err, req, res, next)):

```
app.use(function(err, req, res, next) {
  console.error(err.stack);
  res.status(500).send('Something broke!');
});
```

Built-in middleware

- Except for express.static, all of the middleware previously included with Express' are now in separate modules.

```
var options = {
  dotfiles: 'ignore',
  etag: false,
  extensions: ['htm', 'html'],
  index: false,
  maxAge: '1d',
  redirect: false,
  setHeaders: function (res, path, stat) {
    res.set('x-timestamp', Date.now());
  }
}

app.use(express.static('public', options));
```

```
app.use(express.static('public'));
app.use(express.static('uploads'));
app.use(express.static('files'));
```

Built-in middleware

The `root` argument specifies the root directory from which to serve static assets.

The optional `options` object can have the following properties.

Property	Description
`dotfiles`	Option for serving dotfiles. Possible values are “allow”, “deny”, and “ignore”
`etag`	Enable or disable etag generation
`extensions`	Sets file extension fallbacks.
`index`	Sends directory index file. Set `false` to disable directory indexing.
`lastModified`	Set the `Last-Modified` header to the last modified date of the file on the OS. `true` or `false`.
`maxAge`	Set the max-age property of the Cache-Control header in milliseconds or a stri
`redirect`	Redirect to trailing “/” when the pathname is a directory.
`setHeaders`	Function for setting HTTP headers to serve with the file.

Third-party middleware

Use third-party middleware to add functionality to Express apps.

Install the Node module for the required functionality and load it in your app at the application level

The following example illustrates installing and loading cookie-parsing middleware `cookie-parser`.

```
$ npm install cookie-parser
```

```
var express = require('express');
var app = express();
var cookieParser = require('cookie-parser');

// load the cookie parsing middleware
app.use(cookieParser());
```

Static files

- Serving files, such as images, CSS, JavaScript and other static files is accomplished with the help of a built-in middleware in Express - express.static.

- Pass the name of the directory, which is to be marked as the location of static assets, to the express.

- static middleware to start serving the files directly. For example, if you keep your images, CSS, and JavaScript files in a directory named public, you can do this:

Static files

```
app.use(express.static('public'));
```

Now, you will be able to load the files under the `public` directory:

```
http://localhost:3000/images/kitten.jpg
http://localhost:3000/css/style.css
http://localhost:3000/js/app.js
http://localhost:3000/images/bg.png
http://localhost:3000/hello.html
```

Static files

> The files are looked up relative to the static directory, therefore, the name of the static directory is not a part of the URL.

If you want to use multiple directories as static assets directories, you can call the `express.static` middleware

```
app.use(express.static('public'));
app.use(express.static('files'));
```

The files will be looked up in the order the static directories were set using the `express.static` middleware.

Connect Use Method

- A middleware framework for node.

```
var connect = require("connect");
var http    = require("http");
var app     = connect();
```

```
// Add some middleware
app.use(connect.logger());
app.use(connect.Security);
app.use(connect.Routing);
...
```

```
http.createServer(app).listen(1337);
```

Configuration

- Conditionally invoke callback when **env** matches app.get('env'), aka **process.env.NODE_ENV**.

```
// all environments
app.configure(function() {
    app.set('title', 'My Application');
});

// development only
app.configure('development', function() {
    app.set('db uri', 'localhost/dev');
});

// production only
app.configure('production', function() {
    app.set('db uri', 'n.n.n.n/prod');
});
```

Http Methods

HTTP Methods

- app.get(), app.post(), app.put() & app.delete()
- By default Express does not know what to do with this request body, so we should add the ***bodyParser*** middleware.
- ***bodyParser*** will parse ***application/x-www-form-urlencoded*** and ***application/json*** request bodies and place the variables in ***req.body***

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

PUT Samples

```
<form method="post" action="/">
  <input type="hidden" name="_method" value="put" />
  <input type="text" name="user[name]" />
  <input type="text" name="user[email]" />
  <input type="submit" value="Submit" />
</form>
```

Use POST,
but the node
see it as PUT

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

```
app.put('/', function(){
    console.log(req.body.user);
    res.redirect('back');
});
```

Erro

- Exp ich rec r pas

```
app.error(function (err, req, res, next) {
    if (err instanceof NotFound) {
        res.render('404.jade');
    } else {
        next(err);
    }
});
```

```
function NotFound(msg) {
    this.name = 'NotFound';
    Error.call(this, msg);
    Error.captureStackTrace(this, arguments.callee);
}

NotFound.prototype.__proto__ = Error.prototype;

app.get('/404', function (req, res) {
    throw new NotFound;
});

app.get('/500', function (req, res) {
    throw new Error('keyboard cat!');
});
```

errorHandler Middleware

- Typically defined very last, below any other app.use() calls.

```
app.use(express.bodyParser());
app.use(express.methodOverride());
app.use(app.router);
app.use(function(err, req, res, next){
  // logic
});
```

```
app.use(express.bodyParser());
app.use(express.methodOverride());
app.use(app.router);
app.use(logErrors);
app.use(clientErrorHandler);
app.use(errorHandler);
```

Views

View Template Engines

- Express support many template engines:
  - Haml
  - Jade
  - EJS
  - CoffeeKup
  - jQuery Templates

```
// Start Express
var express = require("express");
var app = express();

// Set the view directory to /views
app.set("views", __dirname + "/views");

// Let's use the Jade templating language
app.set("view engine", "jade");
```

View Rendering

- View filenames take the form "<name>.<engine>", where <engine> is the name of the module that will be required.

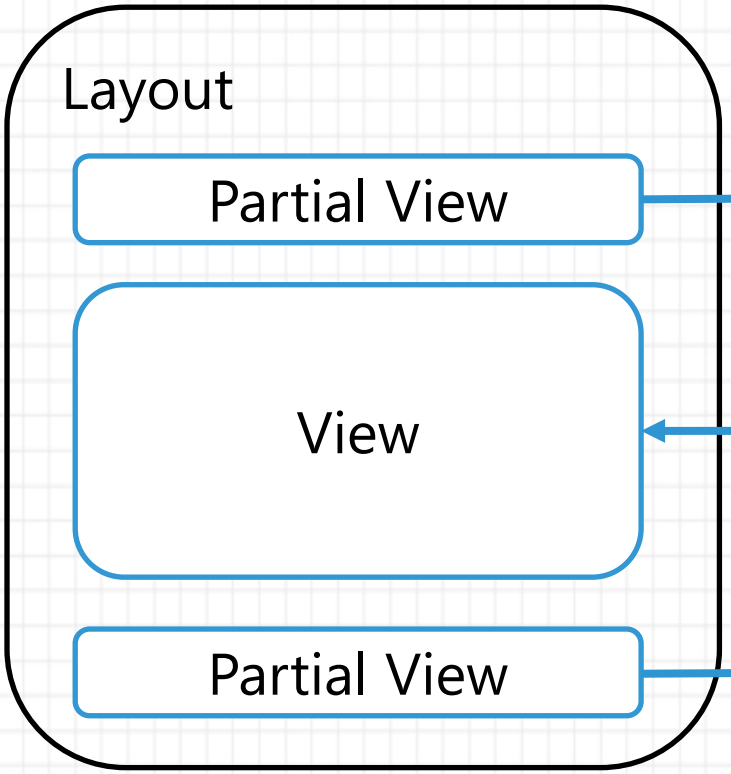
```
app.get('/', function (req, res) {
    res.render( 'index.jade', { title: 'My Site' } );
});
```

Model

View file

View Layout & Partials

- The Express view system has built-in support for partials and collections, which are “mini” views representing a document fragment.



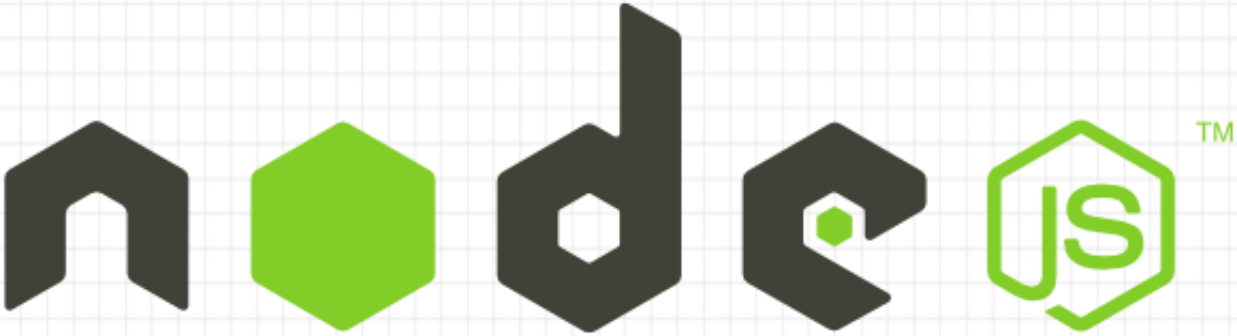

```
partial('header', headerData );
```

```
res.render('page',
  { layout: 'mylayout.jade'
});
```

```
partial('footer', footerData );
```

demo

Views



Thanks