Review of Server-side Development with NodeJS, Express and MongoDB

6/28/2019 Rui Li

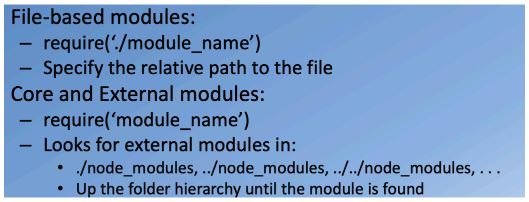
Week1:

1. Node Modules

NodeJS: JavaScript runtime built on Chrome V8 JavaScript Engine
(Utilities written in JavaScript for web development/server-side Development)
NPM: Manages ecosystem of node modules/packages (a package contains JS files and package.json)

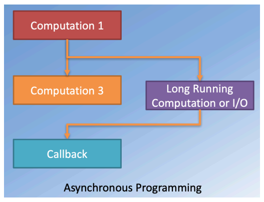
2. Node Modules

CommonJS API: Breaking up your JavaScript application into multiple files
(JavaScript does not define a standard library, CommonJS API defining APIs for common application needs: defines a module format, Node follows the CommonJS module specification.)
Node Modules: Each file in Node application is its own Node module
module variable, module.exports, require
File-based Modules(self-coded) + Core Modules(system) + External Node Modules
Use Node Modules:



File-based modules:
- require('./module_name')
- Specify the relative path to the file

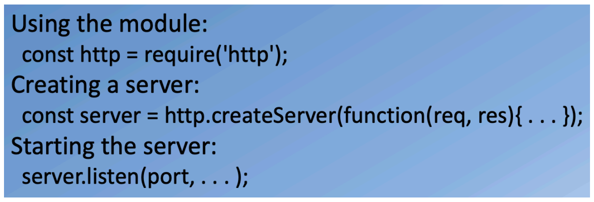
Core and External modules:
- require('module_name')
- Looks for external modules in:
  - ./node_modules, ../node_modules, ../../node_modules, . . .
  - Up the folder hierarchy until the module is found



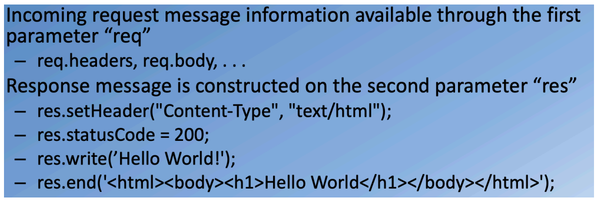
Asynchronous Programming: 把需要耗时间的任务放在后台执行，然后主线继续执行下面的任务，后台任务完成以后给予相应的 callback，在后台完成其支线任务。主线继续完成自己的任务。

3. Node and HTTP

Network operations cause unexpected delays
Hypertext Transfer Protocol (HTTP): A client-server communications protocol, allows retrieving interlinked text documents (hypertext)
HTTP Response: XML or JSON (JavaScript Object Notation)
JSON: Lightweight data interchange format, self-describing and easy to understand
Node HTTP Module: 帮助建立 http server 的一个 node module



Using the module:
const http = require('http');
Creating a server:
const server = http.createServer(function(req, res){ . . . });
Starting the server:
server.listen(port, . . .);



Incoming request message information available through the first parameter "req"
- req.headers, req.body, . . .

Response message is constructed on the second parameter "res"
- res.setHeader("Content-Type", "text/html");
- res.statusCode = 200;
- res.write('Hello World!');
- res.end('<html><body><h1>Hello World</h1></body></html>');

Node path Module: const path = require('path'); path.resolve()/path.extname();
Node fs Module: const fs=require('fs'); fs.exits(filePath,function(exits){...}
POSTMAN: allows you to create HTTP requests and then send them, gives the flexibility of setting up the headers for your http request, and can examine the response.

4. Introduction to Express build a server that serves up the REST API on top of Node.js

Semantic Versioning: <Major Version>.<Minor Version>.<Path>

Express: Fast, unopinionated, minimalist web **framework for Node.js** (a third-party node module/web application framework) 提供一些简便的方法去建立 http server，是一个 node module

Web Services: A **system** designed to support interoperability of systems connected over a network.

(A standardized way for two machines that are connected to the internet to be able to communicate with each other at the application layer level for web-based applications using open standards)

Client-side is facilitated using **REST** where **server** provide a **REST API**

Client can invoke REST API endpoints in order to obtain or upload information from server

Representational State Transfer (REST): A style of software architecture; A collection of network architecture principles which **outline how resources made available on server and accessed from client** (a protocol lives on top of http protocol)

-HTTP methods explicitly
-stateless
-expose directory structure-like URIs
-Transfer using XML, JSON

Express Router: Creates a mini Express application, divide the whole application into several parts according to different route (REST API endpoints). **Supports the specific route endpoint, which this router is going to work, the get/put/post/delete method simply chained into the router, become a group of method implementations.**

Week2:

1. Express Generator

Express Generator: Quick scaffolding tool to generate an Express application skeleton

'Express <App Name>' generates a folder under current folder with the name <App Name>, then use 'npm install' to install modules.

2. Introduction to MongoDB

Databases: store structured information (supports Query, Insert, Update, Delete)

Structured Query Language (SQL): relational databases

NoSQL Databases: scalable, ease of deployment (no object-relation mapping required)

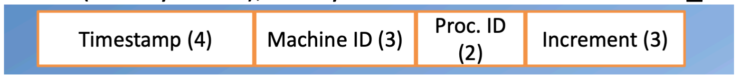
-Document databases (e.g., MongoDB):

Documents (self-contained piece of information) -> Collections -> Database

-Key-value databases (e.g., Redis)
-Column-family databases (e.g., Cassandra)
-Graph databases (e.g., Neo4J)

MongoDB: Document Database, BSON (Binary JSON), every document must have an "_id"

MongoDB ObjectId: 12byte field



3. Node and MongoDB

Node MongoDB Driver: provides a high-level API for a **Node application to interact with the MongoDB server** (inserting, deleting, updating…., supports both callback based and promise based interaction)

Callback Hell: Heavily nested callback code, tame with promises

Promises: supports asynchronous computation, proxy for a value not necessarily know when promise is created (can be chained, can immediately return). **.then + catch** structure

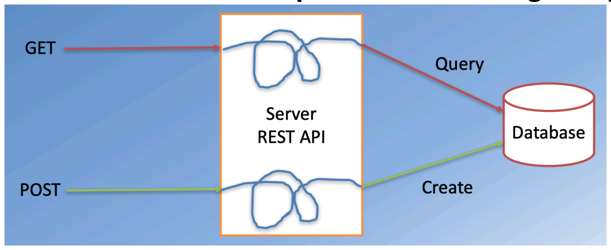
4. Mongoose ODM

MongoDB has no structure imposed on the document

Mongoose: internally use MongoDB driver, can use all its methods
Mongoose ODM: **Adds structure to MongoDB documents through schema**
-Object Data Model
-Object Document Mapping
-Object relational mapping (ORM)
Mongoose Schema: Used to create a Model function, structure of data to be stored (documents fields and types, e.g., String, Number… and subcollection), allows to plugin third-party promise library (e.g., Bluebird)

5. REST API with Express, MongoDB and Mongoose

Goal: integrate the REST API server together with the access to the MongoDB database
Express deal with all **the business logic processing** (service various requests coming to the REST API end points)
MongoDB & Mongoose will **issue the database requests** to the MongoDB (interact with database).



An HTTP request coming in to a REST API end point has to be mapped into a corresponding database operation.
Body Parser: parsed whatever is in the body of the message and load it onto the body property of the request.

Week3:

1. Basic Authentication

HTTP Basic Access Authentication: Method for HTTP user agent to provide username and password with a request, server can challenge a client to authenticate itself (to send username and password in response)
Every request message originating from a client should include the encoded form of the username and password in the request header that goes form the client to the server-side

Server:
```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Basic
```
Client:
```
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```
Authorization Header: "username:password", encoded using Base64

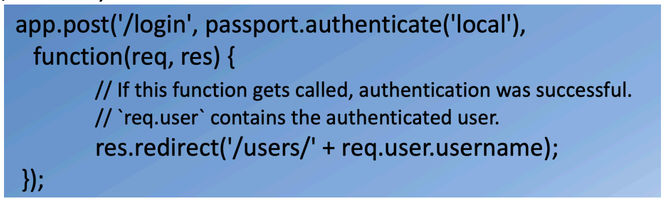
2. Cookies & Express-Session

HTTP Cookies: Small piece of data sent from a web server and stored on the client side, each subsequent request from the client side should include the cookie in the request header
Set cookies: res.cookie(name,value,options)
Parse cookies: var cookieParser = require('cookie-parser'); app.use(cookieParser), attached to req.cookies.name
Signed cookie: signed with a secret key on the server side app.use(cookieParser('secret key') req.signedCookies.name

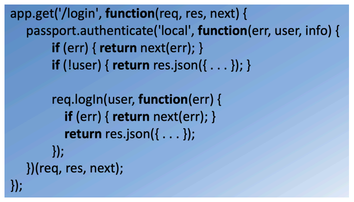
Express-Sessions: track user sessions, combination of cookie with session id and server-side storage of information indexed by session id 一个更大的可以用文件储存 client 信息的库，每次都用 id 从 server 获取

3. User Authentication with Passport

Passport: authentication middleware for Node.js, Modular, flexible (various strategies: Local strategy, OpenID, Oauth)

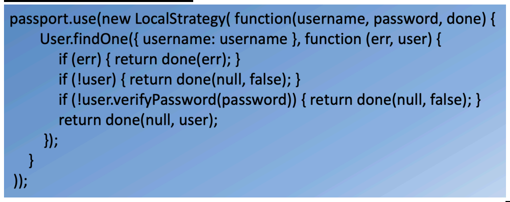


```
app.post('/login', passport.authenticate('local'),
  function(req, res) {
        // If this function gets called, authentication was successful.
        // `req.user` contains the authenticated user.
        res.redirect('/users/' + req.user.username);
 });
```



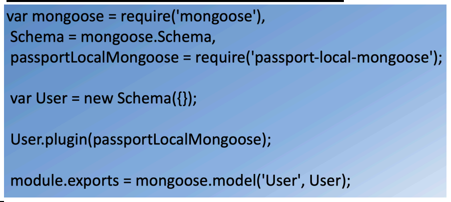
```
app.get('/login', function(req, res, next) {
    passport.authenticate('local', function(err, user, info) {
          if (err) { return next(err); }
          if (!user) { return res.json({ . . . }); }

          req.logIn(user, function(err) {
             if (err) { return next(err); }
             return res.json({ . . . });
          });
    })(req, res, next);
});
```

Passport-Local



```
passport.use(new LocalStrategy( function(username, password, done) {
      User.findOne({ username: username }, function (err, user) {
          if (err) { return done(err); }
          if (!user) { return done(null, false); }
          if (!user.verifyPassword(password)) { return done(null, false); }
          return done(null, user);
      });
  }
));
```

Passport-Local-Mongoose

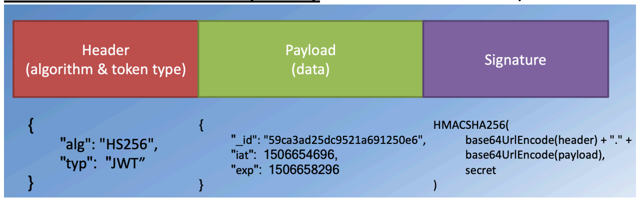


```
var mongoose = require('mongoose'),
Schema = mongoose.Schema,
passportLocalMongoose = require('passport-local-mongoose');

var User = new Schema({});

User.plugin(passportLocalMongoose);

module.exports = mongoose.model('User', User);
```

Cookies+Session Authentication: Cookies used as a storage for session ID that is used as an index into server-side storage of session information.

Token-based Authentication: after validates the credentials sent by client, server creates a signed token and sends it to the client, all subsequent requests from the client should include the token, the server verifies the token and responds with data if validated

JSON Web Tokens (JWT): IETF RFC 7519, self-contained, shareable. (sign(),verify())



```
{
     "alg": "HS256",
     "typ":  "JWT"
}
```

```
{
     "_id": "59ca3ad25dc9521a691250e6",
     "iat":  1506654696,
     "exp":  1506658296
}
```

```
HMACSHA256(
     base64UrlEncode(header) + "." +
     base64UrlEncode(payload),
     secret
)
```

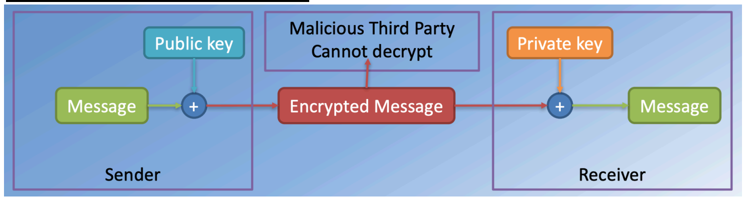
4. Mongoose Population (reference one document from another)

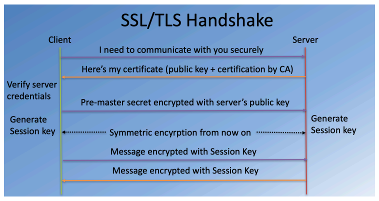
Population: is the process of automatically replacing specified paths within a document with documents from another collection. (cross-reference with ObjectIds)

Week4:

1. HTTPS and Secure Communication

Asymmetric Encryption:




SSL/TLS Handshake
Client
Server
I need to communicate with you securely
Here's my certificate (public key + certification by CA)
Verify server credentials
Pre-master secret encrypted with server's public key
Generate Session key
Symmetric encyrption from now on
Generate Session key
Message encrypted with Session Key
Message encrypted with Session Key

Secure Sockets Layer (SSL) / Transport Layer Security (TLS)

Public key 用于建立连接，连接建立以后用 Private key 继续联系

2. Uploading Files

Through form input: <input type = “file” name=”imageFile”>

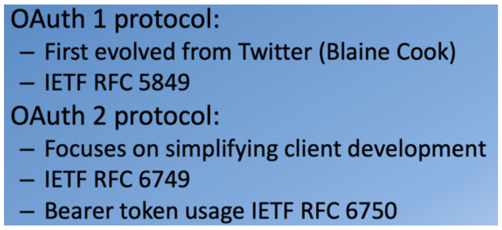
Multer: Node middleware for handling multipart/form data, parses the incoming form data and adds a body object and file/files object to request object.

3. Cross-Origin Resource Sharing（CORS）

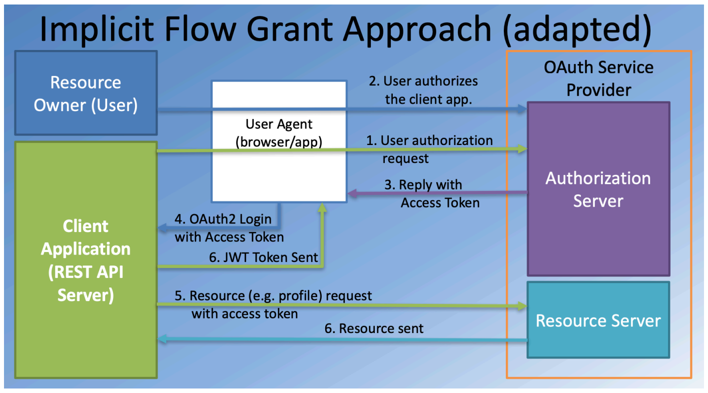
Origin defined by three tuples: Protocol, hostname, port number.
Cross-origin HTTP request: Accessing a resource from a different domain, protocol or port
CORS: mechanism to give web servers cross-domain access controls
(give a new set of HTTP headers that allow servers to describe the set of origins that are permitted to read the information using a web browser)
Simple cross-site Requests + Preflighted Requests + Credentialed Requests
Access-Control-Allow-Origin →*: meaning any origin will be allowed to access this particular resource
Access-Control-Allow-Origin →https://localhost:3443：special origin
Access-Control-Allow-Methods →GET,HEAD,PUT,PATCH,POST,DELETE ：(Preflighting of a request status: 204 No Content)

4. Oauth and User Authentication

Authorization framework based on open standards for Internet users to log into third party websites/apps **using their Social Network accounts**
Oauth 2 Roles: Resource owner + Client Application + Resource Server + Authorization Server


OAuth 1 protocol:
– First evolved from Twitter (Blaine Cook)
– IETF RFC 5849
OAuth 2 protocol:
– Focuses on simplifying client development
– IETF RFC 6749
– Bearer token usage IETF RFC 6750

Authorization Server: Server that issues an access token to the client application to request resource from the resource server
Access token + Refresh token
Register the client application on the Oauth service provider


Implicit Flow Grant Approach (adapted)
Resource Owner (User)
User Agent (browser/app)
2. User authorizes the client app.
1. User authorization request
3. Reply with Access Token
OAuth Service Provider
Authorization Server
Client Application (REST API Server)
4. OAuth2 Login with Access Token
6. JWT Token Sent
5. Resource (e.g. profile) request with access token
6. Resource sent
Resource Server