summer-mvc

Structured, light, spring framework-like web application development framework.

- github : https://github.com/inodient/summer-mvc
- npmjs : https://www.npmjs.com/package/summer-mvc

Introduction

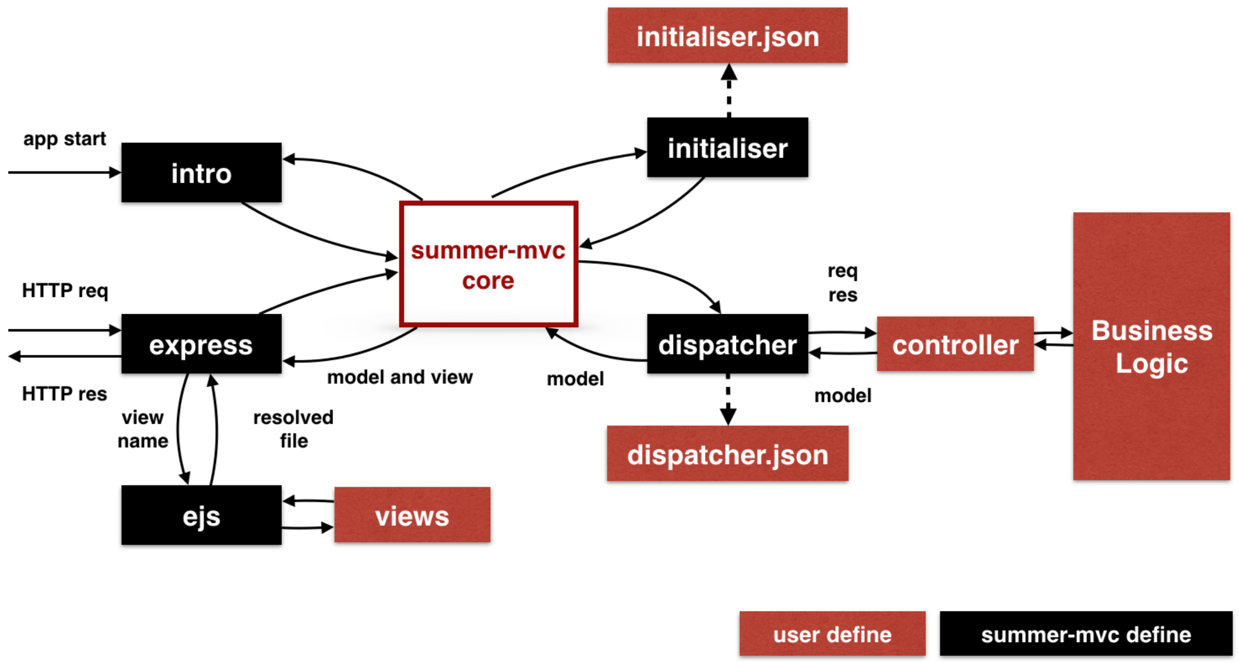
Philosophy

The `summer-mvc` has only one goal to provide very simple way to design web application. As most developer feel comfortable for using the spring framework, the summer-mvc provides spring framework - like environment. `summer-mvc` uses small dispatcher-engine (like spring's dispatcher-servlet), and it can be controlled by JSON files(like spring's xxx-servlet.xml). And you can determine your own controller and views.

Features

- Focus on fast design & development speed
- Use `express` for serving http request, response
- Provide automatic *application-building-machanism*
- Provide structural designing method using **JSON** file format
- Simular architecture with spring framework
- Support 1 view engine (`ejs`), 1 database engine (`mysql`) (v1.1.0)
- Provide easy way to manage cookie, session and database(mysql)

Architecture



The `summer-mvc` consists of 2 parts - ***Initializing part*** and ***execution part***.
As image shown, users can control their web app using **initilizer.json, dispatcher.json, controller and business logic etc**. And use **views** to render results.

initializing part

When administrator starts their web-app, the `summer-mvc` attempt to build simple web application architecture. That time some folders and files will be made automatically. Developer can design whose own web application, correcting ***`initializer.json`*** file.

execution part

When web app receive http request from their clients, the `summer-mvc` send requests to `dispatcher-engine` automatically. And then, `dispatcher-engine` find a controller to resolve request using ***`context_dispatcher.json`***. Developer can design whose own dispatcher, correcting ***`context_dispatcher.json`***. Controller and Business logic is just a part of web app, so these files can contain any logic. But the only rule of controller is that must make `model` object and return this using their callback function.

Getting started

Installing

```
$ npm install summer-mvc
```

Quick Start

In your js file, require module `summer-mvc`. Then `summer-mvc` build architecture, authomatically. (dispatcher, controller, views, queries etc)

```
const mvc = require( "summer-mvc" );
```

And run application. ex) node app.js

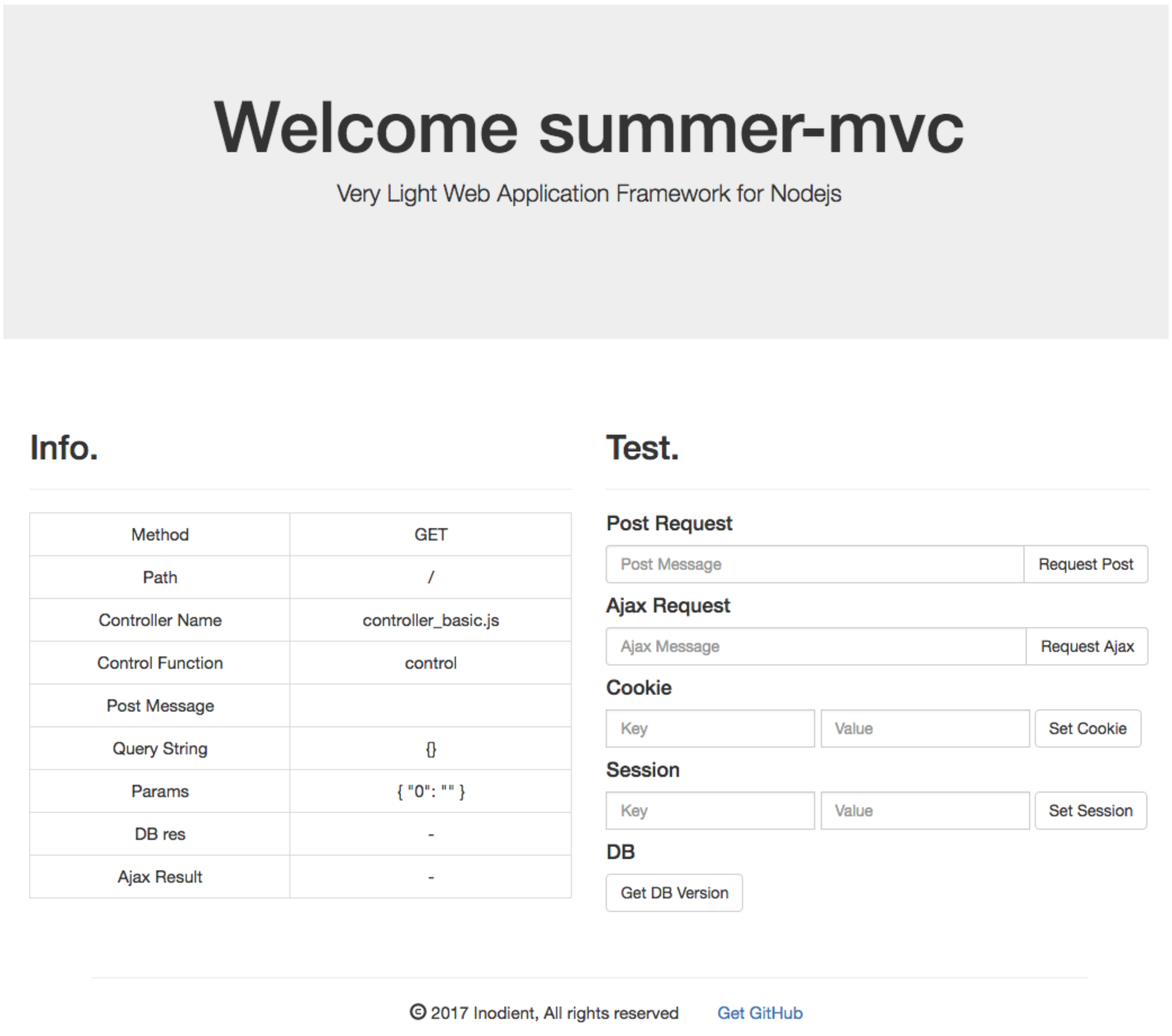
```
node {your-app-js-name}.js
```

Call `basic http request` in your browser. ex) http://localhost:3000
Default port is 3000.

```
http://{your-host-name}:{port}
```

Browser will render *index.ejs*.

index.ejs



Welcome summer-mvc

Very Light Web Application Framework for Nodejs

Info.

Method	GET
Path	/
Controller Name	controller_basic.js
Control Function	control
Post Message	
Query String	{}
Params	{ "0": "" }
DB res	-
Ajax Result	-

Test.

© 2017 Inodient, All rights reserved   Get GitHub

Post Message

Send Post message to `summer-mvc` web app and receive message. Fill in that message in Info.'s Post Message column. Implemented in ***/controller/controller_post.js***

Ajax request

Send Ajax message to `summer-mvc` web app and receive message. Fill in that message in Info.'s Ajax Result column. Implemented in ***/controller/controller_ajax.js***

Cookie

Send Cookie Key / Value to `summer-mvc` web app and receive message. `summer-mvc` set cookie and return message. Fill in that message in Info.'s Query String column. Implemented in ***/controller/controller_cookie_session.js***

session

Send Session Key / Value to `summer-mvc` web app and receive message. `summer-mvc` set session and return message. Fill in that message in Info.'s Query String column. Implemented in ***/controller/controller_cookie_session.js***

DB

Send db query request to `summer-mvc` web app and receive message. `summer-mvc` execute query and return results. Fill in that results in Info.'s DB res column. Implemented in ***/controller/controller_db.js***.
(✓ Install mysql first.)

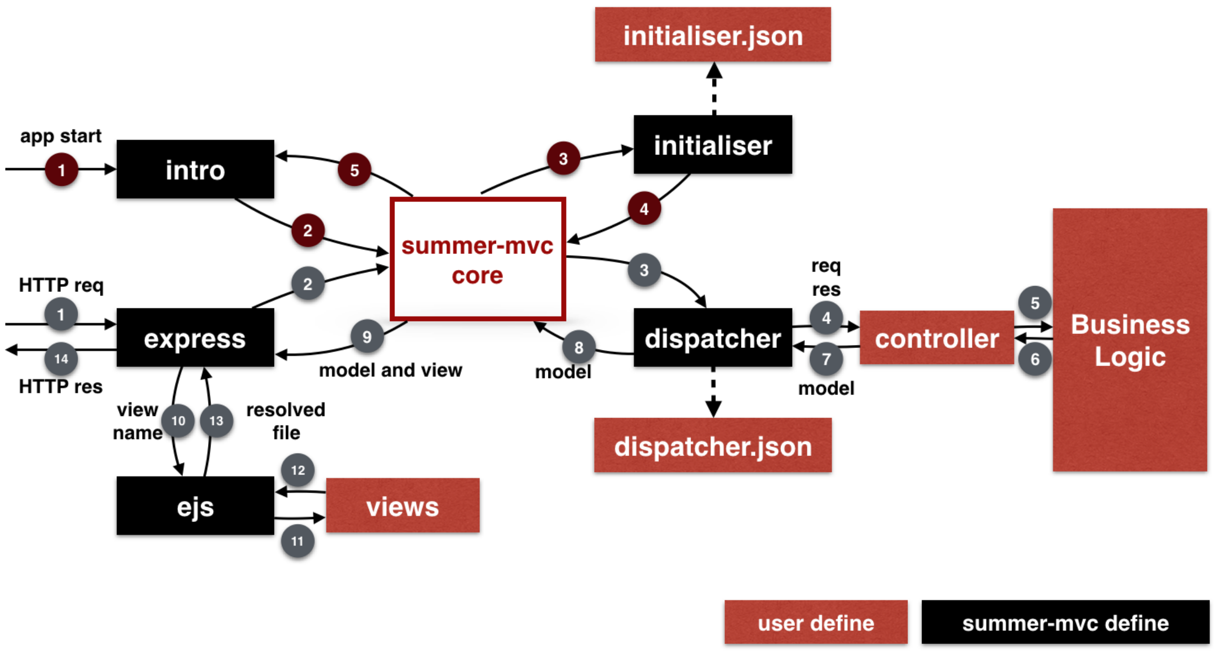
Others

Method, Path, Controller Name, Control Function, Params are basic informations of `summer-mvc` web app.

Guide

The `summer-mvc` provides some methodologies to design web application quickly. Developers only design and implement **initializer.json, dispatcher.json, some controllers, business logics and views**. This concepts are exactly matched with ***MVC (Model-View-Controller)*** pattern. Developers only design and implement MVC and its connections. (like spring framework)

Execution Process



(1, 2) When administrator starts app, (3, 4, 5) **summer-mvc-core** calls initializer and build web application architecture, automatically. **[dispatcher, properties, controller, views, queries]**. Developers can control this process using and correcting ***initializer.json*** file.

(1, 2) When clients send http request, (3, 4) **summer-mvc-core** calls dispatcher and send request to controller. Controllers are implemented as simple Nodejs codes and can use business logics. (5, 6, 7, 8) After execution controllers return **model** object. (9, 10, 11, 12) Finally **summer-mvc** returns **model-and-view (mav)** to **express** and **ejs** modules. (13, 14) Then clients are able to receive http response. Developers can control this process using and correcting ***dispatcher.json*** file.

Automatic Building Mechanism : initializer.json

initializer.json contains basic web application information. When initializing, `summer-mvc` authomatically create and copy folders and files using ***context_architecture*** properties. Developers rewrite this file freely, but please don't delete ***dispatcher / views informations***. Developer can also assign ***static_folders*** for servicing express routers.

> - port : service port (default : 3000)
> - context_architecture : architecture of web application
> - views / options : defaults view engine and options. (**Please don't change**)
> - statis_folders : defaults express static folders (***app.use(express.static(XX));***)

```
{
  "port" : 3000,
  "context_architecture" : [
    {
      "folder" : "dispatcher",
      "files" : [ "context_dispatcher.json" ]
    },
    {
      "folder" : "controller",
      "files" : [ "controller_basic.js" ]
    },
    {
      "folder" : "views",
      "files" : [ "index.ejs", "error.html" ]
    },
    {
      "folder" : "properties",
      "files" : [ "initializer.json", "db.json" ]
    },
    {
      "folder" : "queries",
      "files" : [ "query.json" ]
    }
  ],
  "views" : {
    "engine" : "ejs"
  },
  "static_folders" : [ "img", "lib", "css" ]
}
```

/properties/initializer.json

Design Dispatcher : dispatcher.json

dispatcher.json is the most important part of `summer-mvc` as it has information of http servicing. Developers rewrite this file freely.

> "GET", "POST" : method of http request
>
> - id : identifier (doesn't use for system but fill it)
> - path : URI's resource paths
> - controllerJS : indicate controller js file (../controller)
> - controlFunction : name of control function in controllerJS file
> - view : indicate ejs view file (../views)

```
{
  "GET" : [
    {
      "id": "index",
      "path": "/",
      "controllerJS": "controller_basic.js",
      "controlFunction": "control",
      "viewPath": "",
      "view": "index.ejs"
    },
    {
      "id": "setCookie",
      "path": "/setCookie",
      "controllerJS": "controller_cookie_session.js",
      "controlFunction": "control",
      "viewPath": "",
      "view": "index.ejs"
    }
  ],
  "POST" : [
    {
      "id": "indexPost",
      "path": "/",
      "controllerJS": "controller_post.js",
      "controlFunction": "control",
      "viewPath": "",
      "view": "index.ejs"
    }
  ]
}
```

/dispatcher/context_dispatcher.json

Implements Controller

Developers can implements their own controllers. These must be assigned in *context_dispatcher.json*. As Nodejs allow so many types of implementation style, developer can implement their code freely. The only one rule is that must return ***model*** object using callback function. `summer-mvc` call controller function automatically, but contol function has 3 mandatory parameters.

> - req : request object comes from client
> - res : response object send to client (managed by `your application`)
> - callback : **1st param - err, 2nd param - model object**

```
exports.control = function( req, res, callback ){
  var model = {};
  var err = null;

  model.result = "This is sample code";
  callback( err, model );
}
```

A. dispatched
As `summer-mvc`'s dispatcher automatically assign controller, developers can only assign path and controller in ***context_dispatcher.json***.

```
{
  "GET" : [
    {
      "id": "index",
      "path": "/index/welcome",
      "controllerJS": "controller1.js",
      "controlFunction": "control1",
      "viewPath": "",
      "view": "index.ejs"
    },
    {
      "id": "index",
      "path": "/index/welcome2",
      "controllerJS": "controller2.js",
      "controlFunction": "control2",
      "viewPath": "",
      "view": "index.ejs"
    }
  ]
}
```

B. Use Controller

As controller has ***req*** object, developer can catch request ***path*** directly. Developer make branch codes to control parsed paths.

In *context_dispatcher.json*,

```
{
  "GET" : [
    {
      "id": "index",
      "path": "/index",
      "controllerJS": "controller.js",
      "controlFunction": "control",
      "viewPath": "",
      "view": "index.ejs"
    }
  ]
}
```

In *controller.js*,

```
exports.control = function( req, res, callback ){
  var model = {};
  var err = null;

  var path = req._parsedUrl.pathname;
  path.replace( "/index", "" );

  if( req._parsedUrl.pathname === "/welcome1" ){
    //execute business logic 01
  } else if( req._parsedUrl.pathname === "/welcome2" ){
    //execute business logic 02
  }

  callback( err, model );
}
```

Implements Views

`summer-mvc`'s default view engine is ejs. And `summer-mvc` has **views** folder. Developers create *.ejs files in that folder. As controller return **model** object, *.ejs file must use **model** object to render results.

In *controller.js*,

```
exports.control = function( req, res, callback ){
  var model = {};
  var err = null;

  model.result = "This is sample page";

  callback( err, model );
}
```

In *index.ejs*,

```
<tr>
   <td>Result Message</td>
   <td><%= result %></td>
</tr>
```

APIs

`summer-mvc` provides some APIs for web application especailly **cookie, session** and **db**.

setCookie(cookieKey, cookieValue)

```
let connection = new connectionHandler( req, res );
connection.setCookie( "cookie_Key", "cookie_Value" );
```

getCookie(cookieKey)

```
let connection = new connectionHandler( req, res );
connection.getCookie( "cookie_Key" );
```

clearCookie(cookieKey)

```
let connection = new connectionHandler( req, res );
connection.clearCookie( "cookie_Key" );
```

setSession(sessionKey, sessionValue)

```
let connection = new connectionHandler( req, res );
connection.setSession( "session_Key", "session_Value" );
```

getSession(sessionKey)

```
let connection = new connectionHandler( req, res );
connection.getSession( "session_Key" );
```

destroySession()

```
let connection = new connectionHandler( req, res );
connection.destroySession();
```

executeQuery(queryId[, params, ...], callback) : mysql only

> queryId : identifier of query (queries/query.json's id)
>
> params : prepared statement's empty values
>
> callback : contains 3 parameters - err, results(rs), fields(columns)

```
let db = new dbHandler();
db.executeQuery( "getMySqlVersion", function( err, results, fields ){
    console.log( results );
} );
```

getQueryString(queryId[, params]) : mysql only

> queryId : identifier of query (queries/query.json's id)
>
> params : prepared statement's empty values

```
let db = new dbHandler();
db.executeQuery( "getMySqlVersion" )
```

Sample Architecture

When initializing completed, developer can find some sample codes as follow. Please use this samples code and exercise.

Name	Type	Usage
✚ dispatcher	folder	containing dispatcher json files
context_dispatcher.json	file	containing http req/res paths and assign controller & views
✚ properties	folder	containing defaults setting json files (initializer.json, db.json)
initializer.json	file	containing default web application configuration (port, static_folder etc)
db.json	file	containing db connection informations (host, user, password etc)
✚ controller	folder	containing controller js files
✚ views	folder	containing ejs view files
✚ queries	folder	containing query json files

Parent	Name	Type	Usage
controller	controller_basic.js	file	sample basic controller
controller	controller_ajax.js	file	sample ajax controller
controller	controller_cookie_session.js	file	sample cookie, session controller
controller	controller_db.js	file	sample db controller
controller	controller_post.js	file	sample post controller
views	index.ejs	file	sample index ejs page
views	error.html	file	sample error html page
queries	query.json	file	sample query information file

People

The original author of `summer-mvc` is ***Changho Kang (Inno)***.
`summer-mvc` is the first open source project of **Inno**, some other projects will be released continuously. If have any questions or opinions, please contact **Inno** using email (inodient@gmail.com).

Version Information

v 1.0.0 Initial version
v 1.0.1 Bug fixed
v 1.0.2 Bug fixed
v 1.1.0 But fixed, Document Updated
v 1.1.1 Current Version

License