

spark-notes

November 1, 2018

Contents

I	Project Setup	3
1	Dependencies	4
1.1	Maven	4
II	Server Operations	5
2	init()	6
3	stop()	7
III	Routes	8
4	Overview	9
5	Named Paramaters	10
5.1	splat()	10
6	Grouping Routes	11
IV	Request	12
7	List of Methods	13
V	Respond	15
8	List of Methods	16
9	Querrey Maps, Cookies, Sessions, Halting	17
9.1	Query Maps	17
9.2	Cookies	17
9.3	Sessions	17

9.4 Halting	18
10 Filters	19
10.1 Before	19
10.2 After	19
10.3 afterAfter	19
10.4 Filters Taking Patterns	20
 VI TBC	 21

Part I

Project Setup

Chapter 1

Dependencies

1.1 Maven

```
<dependency>
  <groupId>com.sparkjava</groupId>
  <artifactId>spark-core</artifactId>
  <version>2.7.2</version>
</dependency>
```

Part II

Server Operations

Chapter 2

init()

Default behaviour if it fails:

```
private Consumer<Exception> initExceptionHandler = (e) -> {  
    LOG.error("ignite failed", e);  
    System.exit(100);  
};
```

The behaviour can be customised by invoking
`initExceptionHandler(e -> {});`

Chapter 3

stop()

Shuts down the server.

Part III

Routes

Chapter 4

Overview

- they are made of three peces:
 - **a verb** - *get, post put, delete, head, trace, connect, options*
 - **a path** (*URL*)
 - **a callback** - (*request, response*) ->()
- they are matched in the order they are defined
- **static import is recommended** for improved readability

4.1 Methods

4.1.1 Methods

Methods

get() - shows something.

post() - creates something

put() - updates something

delete() - self-explanatory

options() - 'appease' something

Chapter 5

Named Paramaters

- preceded by a colon
- accessed by *params* method on *request* object.

```
get("/path/:param", (req, res) -> {  
  return "Hello " + req.params(":param");  
});
```

5.1 splat()

It works like a combination of wildcard and vararg. *splat()* return an arrays of those parameters.

```
get("/path/*", (req, res) -> {  
  return "Number of splat parameters is " + req.splat().length;  
});
```

Chapter 6

Grouping Routes

We need to call a *path(..)* method:

```
path("/api", () -> {
  before("/*", (q, a) -> log.info("Received api call"));
  path("/email", () -> {
    post("/add",      EmailApi.addEmail);
    put("/change",    EmailApi.changeEmail);
    delete("/remove", EmailApi.deleteEmail);
  });

  path("/username", () -> {
    post("/add",      UserApi.addUsername);
    put("/change",    UserApi.changeUsername);
    delete("/remove", UserApi.deleteUsername);
  });
});
```

Part IV

Request

Chapter 7

List of Methods

<code>request.attributes();</code>	the attributes list
<code>request.attribute("foo");</code>	value of foo attribute
<code>request.attribute("A", "V");</code>	sets value of attribute A to V
<code>request.body();</code>	request body sent by the client
<code>request.bodyAsBytes();</code>	request body as bytes
<code>request.contentLength();</code>	length of request body
<code>request.contentType();</code>	content type of request.body
<code>request.contextPath();</code>	the context path, e.g. <code>"/hello"</code>
<code>request.cookies();</code>	request cookies sent by the client
<code>request.headers();</code>	the HTTP header list
<code>request.headers("BAR")</code>	value of BAR header
<code>request.host</code>	the host, e.g. <code>"example.com"</code>
<code>request.</code>	client IP address
<code>request.params("foo")</code>	value of foo path parameter
<code>request.params();</code>	map with all parameters
<code>request.pathInfo();</code>	the path info
<code>request.port();</code>	the server port
<code>request.protocol();</code>	the protocol, e.g. <code>HTTP/1.1</code>
<code>request.queryMap();</code>	the query map
<code>request.queryMap("foo");</code>	query map for a certain parameter
<code>request.queryParams();</code>	the query param list
<code>request.queryParams("FOO");</code>	value of FOO query param
<code>request.queryParamsValues("FOO")</code>	all values of FOO query param
<code>request.raw();</code>	raw request handed in by Jetty
<code>request.requestMethod();</code>	The HTTP method (GET, ..etc)
<code>request.scheme();</code>	<code>"http"</code>
<code>request.servletPath();</code>	the servlet path, e.g. <code>/result.jsp</code>
<code>request.session();</code>	session management
<code>request.splat();</code>	splat (*) parameters
<code>request.uri();</code>	the uri, e.g. <code>"http://example.com/foo"</code>
<code>request.url();</code>	the url. e.g. <code>"http://example.com/foo"</code>
<code>request.userAgent();</code>	user agent

Part V

Respond

Chapter 8

List of Methods

<code>response.body();</code>	get response content
<code>response.body("Hello");</code>	sets content to Hello
<code>response.header("FOO", "bar");</code>	sets header FOO with value bar
<code>response.raw();</code>	raw response handed in by Jetty
<code>response.redirect("/example");</code>	browser redirect to /example
<code>response.status();</code>	get the response status
<code>response.status(401);</code>	set status code to 401
<code>response.type();</code>	get the content type
<code>response.type("text/xml");</code>	set content type to text/xml

Chapter 9

Query Maps, Cookies, Sessions, Halting

<http://sparkjava.com/documentation#sessions>

9.1 Query Maps

```
request.queryMap().get("user", "name").value();
request.queryMap().get("user").get("name").value();
request.queryMap("user").get("age").integerValue();
request.queryMap("user").toMap();
```

9.2 Cookies

```
request.cookies();           // get map of all request cookies
request.cookie("foo")       // access request cookie by name
response.cookie("foo", "bar"); // set cookie with a value
response.cookie("foo", "bar", 3600); // set cookie with a max-age
response.cookie("foo", "bar", 3600, true); // secure cookie
response.removeCookie("foo"); // remove cookie
```

9.3 Sessions

```
// create and return session
request.session(true);

// Get session attribute 'user'
request.session().attribute("user");

// Set session attribute 'user'
```

```
request.session().attribute("user","foo");

// Remove session attribute 'user'
request.session().removeAttribute("user");
request.session().attributes();           // Get all session attributes
request.session().id();                   // Get session id
request.session().isNew();                 // Check if session is new
request.session().raw();                   // Return servlet object
```

9.4 Halting

Stops immediately a request within a filter or route.

```
halt();                                // halt
halt(401);                             // halt with status
halt("Body Message");                  // halt with message
halt(401, "Go away!");                 // halt with status and message
```

Chapter 10

Filters

10.1 Before

They are evaluated **before each request**:

```
before((request, response) -> {  
    boolean authenticated;  
  
    // ... check if authenticated  
  
    if (!authenticated) {  
        halt(401, "You are not welcome here");  
    }  
});
```

10.2 After

They are evaluated **after each request**:

```
after((request, response) -> {  
    response.header("foo", "set by after filter");  
});
```

10.3 afterAfter

Works similar to 'finally' blocks:

```
afterAfter((request, response) -> {  
    response.header("foo", "set by afterAfter filter");  
});
```

10.4 Filters Taking Patterns

They are evaluated **only if request path matches the pattern**:

```
before("/protected/*", (request, response) -> {  
    // ... check if authenticated  
  
    halt(401, "Go Away!");  
});
```

Part VI

TBC

<http://sparkjava.com/documentation#response-transformer>