Spark

Prof. Igor Avila Pereira
igor.pereira@riogrande.ifrs.edu.br

Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul (IFRS)
Câmpus Rio Grande
Divisão de Computação

Spark

O que é Spark?

É um framework web Java simples e leve construído para o desenvolvimento rápido.

Objetivo

Fornecer uma alternativa para os desenvolvedores que querem, ou que são obrigados a, desenvolver alguma aplicação web em Java.

Spark

Características

- Prima por ser um framework o mais simples e direto possível, sem a necessidade de complicados arquivos XML de configuração, permitindo o desenvolvimento de aplicações web de forma rápida em Java puro com o mínimo esforço.
- É um paradigma totalmente diferente quando comparado com o uso excessivo de anotações para realizar coisas bastante trivial visto em outros frameworks web.
- **Pré-requisitos:** jdk $>= 8$

Spark: Hello World

```java
import static spark.Spark.*;

public class HelloWorld {
    public static void main(String[] args) {
        get("/hello", (req, res) -> "Hello World");
    }
}
```

Figura : Exemplo: Código Java

```
http://localhost:4567/hello
```

Figura : Exemplo: Como o código deve ser acessado pelo **browser**

Exemplos

```
// matches "GET /hello/foo" and "GET /hello/bar"
// request.params(":name") is 'foo' or 'bar'
get("/hello/:name", (request, response) -> {
    return "Hello: " + request.params(":name");
});
```

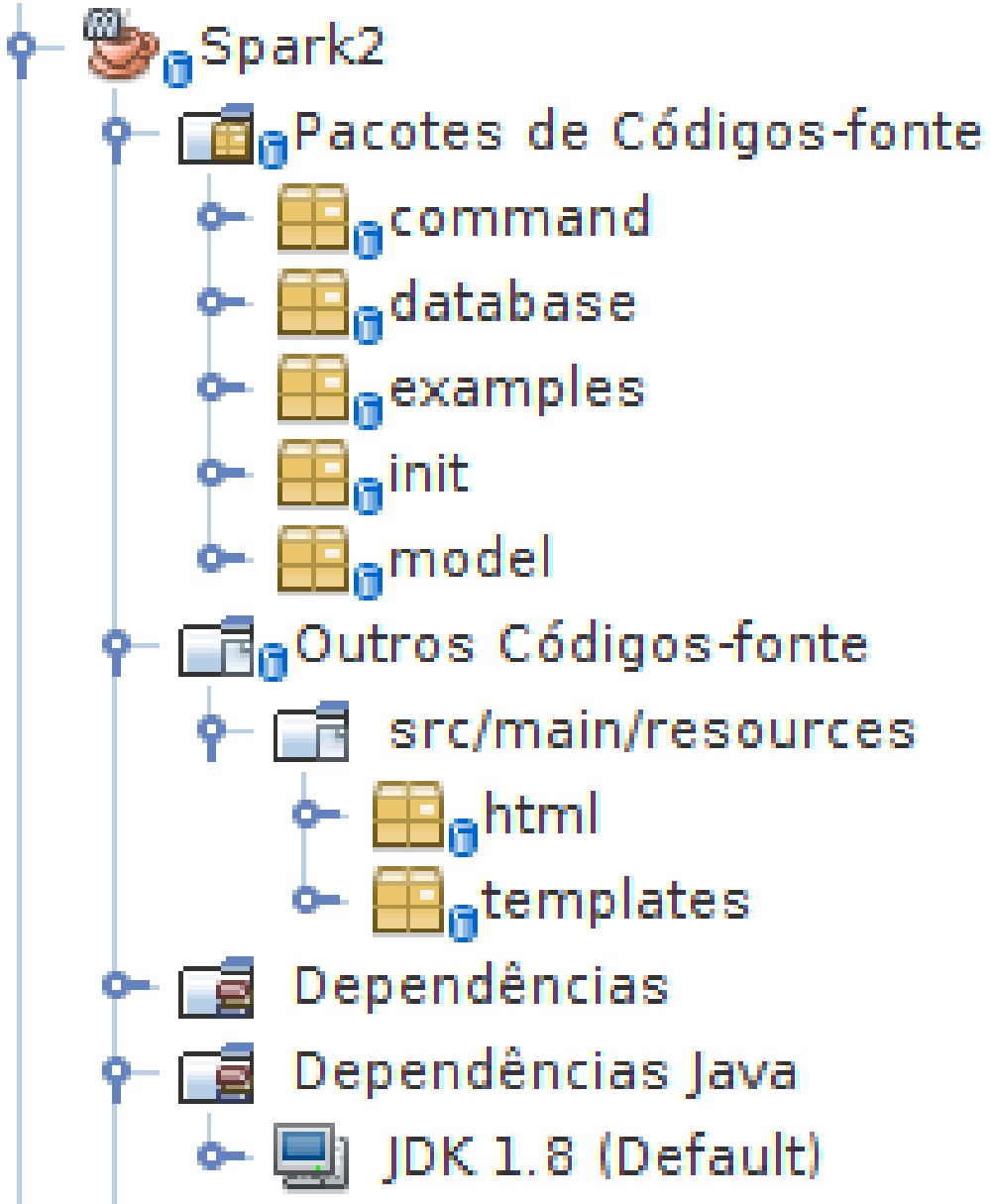
Figura : Exemplo 1

```
// matches "GET /say/hello/to/world"
// request.splat()[0] is 'hello' and request.splat()[1] 'world'
get("/say/*/to/*", (request, response) -> {
    return "Number of splat parameters: " + request.splat().length;
});
```

Figura : Exemplo 2

Spark: Projeto

Diretórios



Spark: Projeto

Diretórios

- **command**: Diretório que armazena a lógica das funcionalidades do projeto
- **database**: diretório responsável por armazenar as classes de conexão e os DAOs
- **examples**: Diretório que contém exemplos
- **init**: Diretório responsável pelo Main e por definir as rotas do sistema.
- **model**: Diretório que armazena as classes de modelo (ou negócio) da aplicação

Spark: Rotas

```
public class Main {

    public static void main(String[] args) {

        staticFileLocation("/html");

        // index
        get("/", new TemplateViewRoute() {
            @Override
            public ModelAndView handle(Request request, Response response) {
                return new ModelAndView(new ListCommand(request, response).getMap(), "index.html");
            }
        }, new MustacheTemplateEngine());

        // delete
        get("/delete/:id", new TemplateViewRoute() {
            @Override
            public ModelAndView handle(Request request, Response response) {
                return new ModelAndView(new DeleteCommand(request, response), "");
            }
        }, new MustacheTemplateEngine());
```

Spark: Rotas

- **staticFileLocation("/html")**: Define o diretório que contém os arquivos estáticos (ex: arquivos html).
- **get("/")**: Define a rota inicial do projeto (*index*)
- **new ListCommand(request, response).getMap()**: Define o command responsável pela lógica da requisição.
  - Obs: **getMap()**: Método que deve ser invocado quando há alguma variável dinâmica que deve ser encaminhada para o template.
  - No caso do **new DeleteCommand()** o método **getMap()** não é utilizado.
- **"index.html"**: Template que será chamado após a execução do command.

Spark: Rotas

- **get:("/delete/:id")**: rota que define a URL que deleta um registro recebendo o parâmetro id por get.
- **new DeleteCommand(request, response)**: command responsável por deletar um registro identificado pela parâmetro **id**.
  - Neste caso o command **DeleteCommand** não exige nenhum template. Neste caso, o command redireciona (após execução) novamente para a rota inicial.

Spark: Commands

```
public class ListCommand extends Command {

    public ListCommand(Request request, Response response) {
        super(request, response);
        map.put("name", "Seja bem vindo!!!");
        ArrayList<Pessoa> pessoas;
        try {
            pessoas = new PessoaDAO().select();
            if (pessoas.size()> 0) {
                map.put("pessoas", pessoas);
            }
        } catch (SQLException ex) {
            Logger.getLogger(ListCommand.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

}
```

Figura : ListCommand

- Para encaminhar algum conteúdo para o template é preciso utilizar o método **map.put("nome", variável)**.
  - **map.put("name", "Seja bem vindo")**
    - Enviando uma String **name** para o template
  - **map.put("pessoas", pessoas)**
    - Enviando um ArrayList **pessoas** para o template

Spark: Template

```
<p style="color: red;"> oi, {{name}}! (testando o css) </p>
<form action="/delete_multiple" method="post">
    <table>
        {{#pessoas}}
        <tr>
            <td> <input type="checkbox" name="id" value="{{id}}"> </td>
            <td> <a href="/delete/{{id}}"> Deletar </a> </td>
            <td> <a href="/screen_update/{{id}}"> Editar </a> </td>
            <td>{{nome}}</td>
            <td>{{sobrenome}}</td>
        </tr>
        {{/pessoas}}
    </table> <br>
    <input type="submit" value="Excluir Multiplo (selecione os checkboxes)">
</form>
```

Figura : Template index.html

- Variáveis setadas no command podem ser utilizadas no template, respeitando a sintaxe {{**nome**}}
- Coleções (como ArrayList) podem ser percorridos através da seguinte sintaxe:
  - {{**#nome**}} (início) e {{**/nome**}} (fim)

Spark: Template

```
public class DeleteCommand extends Command {

    public DeleteCommand(Request request, Response response) {
        super(request, response);
        try {
            new PessoaDAO().delete(Integer.parseInt(request.params(":id")));
        } catch (SQLException ex) {
            Logger.getLogger(ListCommand.class.getName()).log(Level.SEVERE, null, ex);
        }
        response.redirect("/");
    }

}
```

Figura : Command DeleteCommand

- **request.params(":id")**: método que obtém o valor do parâmetro id vindo por GET
- **response.redirect("/")**: método de redirecionamento.
  - Neste caso está redirecionando para a rota definida por "/"

Spark: Rota

```
// tela inserir
get("/screen_insert", new TemplateViewRoute() {
    @Override
    public ModelAndView handle(Request request, Response response) {
        return new ModelAndView(new Command(request, response), "screen_insert.html");
    }
}, new MustacheTemplateEngine());
```

Figura : Rota Tela Inserir

- No caso de um simples redirecionamento há 2 opções:
  1. Criar uma rota, instanciando a *super* classe Command (*pai* de todo Command).
  2. Ou utilizar um link simples para algum arquivo estático (presente no diretório de arquivos estáticos).
- A rota redireciona para o template screen_insert.html

Spark: Screen Insert Template

```
<!DOCTYPE html>
<html>
    <head>
        <title>TODO supply a title</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
        <form action="http://localhost:4567/insert/" method="post">
            Nome: <input name="nome" value="">
            Sobrenome: <input name="sobrenome" value="">
            <input type="submit" value="Enviar">
        </form>
    </body>
</html>
```

Figura : screen_insert.html

- Formulário responsável pela adição.
  - O formulário possui 2 *inputs*:
    - **nome**
    - **sobrenome**

Spark: Rota /insert

```
post("/insert/", new TemplateViewRoute() {
    @Override
    public ModelAndView handle(Request request, Response response) {
        return new ModelAndView(new InsertCommand(request, response).getMap(), "message.html");
    }
}, new MustacheTemplateEngine());
```

Figura : Rota /insert

- Rota responsável por receber os parâmetros vindos do formulário e persistir um novo registro
- O command **InsertCommand** é o command responsável por receber os parâmetros, instanciar o objeto DAO pertinente e atribuir uma mensagem ao usuário.
  - Observe o uso do método **getMap()**.
    - Assim a mensagem (setado no command) será encaminhada para o template message.html

Spark: Rota /insert

```
public class InsertCommand extends Command {

    public InsertCommand(Request request, Response response) {
        super(request, response);
        new PessoaDAO().insert(new Pessoa(request.queryParams("nome"), request.queryParams("sobrenome")));
        map.put("message", "Voce acaba de inserir o usuario: " + request.queryParams("nome") + " " + request.queryParams("sobrenome"));
    }


}
```

Figura : InsertCommand

- **request.queryParams("nome")** contém o valor do campo nome do formulário preenchido pelo usuário
- **request.queryParams("sobrenome")** contém o valor do campo sobrenome do formulário preenchido pelo usuário
- Obs: O **request.queryParams** deve ser utilizado em requisições do tipo POST

Spark: Request

Demais métodos da classe Request.

```
request.body();                 // request body sent by the client
request.cookies();              // request cookies sent by the client
request.contentLength();        // length of request body
request.contentType();          // content type of request.body
request.headers();              // the HTTP header list
request.headers("BAR");         // value of BAR header
request.attributes();           // the attributes list
request.attribute("foo");       // value of foo attribute
request.attribute("A", "V");    // sets value of attribute A to V
request.host();                 // "example.com"
request.ip();                   // client IP address
request.pathInfo();             // the path info
request.params("foo");          // value of foo path parameter
request.params();               // map with all parameters
request.port();                 // the server port
request.queryMap();             // the query map
request.queryMap("foo");        // query map for a certain parameter
request.queryParams("FOO");     // value of FOO query param
request.queryParams();          // the query param list
request.raw();                  // raw request handed in by Jetty
request.requestMethod();        // The HTTP method (GET, ..etc)
request.scheme();               // "http"
request.session();              // session management
```

Spark: Request

Continuação...

```
request.splat();                // splat (*) parameters
request.url();                  // "http://example.com/foo"
request.userAgent();            // user agent
```

Figura : Request

Spark: Response

Demais métodos da classe Response.

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

Figura : Response

Spark: Arquivos Estáticos

```
staticFileLocation("/public"); // Static files
```

Figura : Definir o diretório dos arquivos estáticos **internos** ao projeto*

```
externalStaticFileLocation("/var/www/public"); // Static files
```

Figura : Definir o diretório dos arquivos estáticos **externos** ao projeto

*No projeto disponibilizado, definiu-se o diretório /html

Spark: Filtros

Filtros são **pseudo-rotas** que podem ocorrer antes ou depois de uma rota.

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

Figura : Filtro before - Exemplo: Autenticação de Usuários

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

Figura : Filtro after

Examplos e Tutorials

Exemplos

https://github.com/perwendel/spark/blob/master/README.md#examples

Vídeos Tutoriais

https://sparktutorials.github.io/2015/08/04/spark-video-tutorials.html

Spark

Prof. Igor Avila Pereira
igor.pereira@riogrande.ifrs.edu.br

Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul (IFRS)
Câmpus Rio Grande
Divisão de Computação