

Sistemas Operacionais II N - Atividade de programação guiada 2

Instituto de Informática, Universidade Federal do Rio Grande do Sul
INF01151 - Sistemas Operacionais II N 2021/1
Henry Bernardo Kochenborger de Avila 00301161

October 12, 2021

1 Servidor

A aplicação¹ servidor é composta por uma classe representando um servidor que faz uso do conceito de *webservices* para intermediar a interação com os clientes da aplicação. Sendo assim, para definir um *endpoint* para a aplicação, foi utilizado a anotação *Path* com valor */calculadora*.

```
@Path("calculadora")  
public class CalculadoraRest
```

No que se trata da parte funcional do servidor, é feito o uso da classe *Calculadora* que dispõe de atributos que indicam as respostas sobre duas operações disponíveis pela classe: a soma e multiplicação de dois inteiros.

```
@XmlElement  
public class Calculadora {  
  
    private String operador;  
    private int operando1;  
    private int operando2;  
    private int resultado;  
    private String erro;  
  
    public Calculadora () {}  
  
    public Calculadora(int op1, int op2, String oper) {  
        this.operando1 = op1;  
        this.operando2 = op2;  
        this.operador = oper;  
        this.erro = "";  
        this.resultado = 0;  
        if (oper == "+")  
            this.resultado = op1 + op2;  
        else if (oper == "*")  
            this.resultado = op1 * op2;  
        else  
            this.erro = "Operação não suportada!"  
    }  
}
```

¹Pode ser verificada no seguinte repositório no *GitHub*: <https://github.com/akahenry/sisop2pg>.

A classe *CalculadoraRest* funciona como mediadora entre o cliente e a classe *Calculadora*. Como a classe *Calculadora* não retorna os resultados das operações através de métodos e sim através dos atributos da classe, então as respostas das operações são dadas ao usuários através de uma serialização de uma instância da classe que tratou indiretamente a chamada feita pelo cliente. Por este motivo, é necessário que a classe *Calculadora* seja serializável (indicado através da anotação *XmlRootElement*).

```
@Path("/somarInt/{a}/{b}")
@Produces(MediaType.APPLICATION_JSON)
@GET
public Calculadora somarInt(@PathParam("a") int a, @PathParam("b") int
    b) {
    return new Calculadora(a, b, "+");
}

@Path("/multiplicarInt/{a}/{b}")
@Produces(MediaType.APPLICATION_JSON)
@GET
public Calculadora multiplicarInt(@PathParam("a") int a, @PathParam("b")
    int b) {
    return new Calculadora(a, b, "*");
}
```

Assim como há a necessidade de indicar o *endpoint* do servidor para acesso ao *webservice*, é necessário indicar também qual é o *path* para acesso aos métodos disponibilizados pelo serviço. Sendo assim, os métodos possuem como anotação o *path* em que poderão ser acessados. Além disso, também há a indicações, através de anotações, sobre quais métodos *HTTP* são suportados (apenas *GET*, neste caso) e qual o tipo da informação retornada pela chamada *HTTP* (dada pela anotação *@Produces*).

2 Cliente

O cliente é composto de apenas uma classe que faz uso da aplicação servidor. Ele realiza a instanciação da conexão com o servidor através de domínios pré-definidos através do uso do *Docker*. Por este motivo, o cliente busca o servidor pelo nome do serviço dado ao *container* que irá servir a aplicação servidor (neste caso *pg2-server*).

```
public class Cliente {

    public static void main(String[] args) {
        ClientConfig config = new DefaultClientConfig();
        Client cliente = Client.create(config);
        WebResource servico = cliente.resource("http://pg2-server:9000/
            calculadora");
        int a = 2, b = 3;

        WebResource servicoSomarInt = servico.path("somarInt").path(a +
            "/" + b);

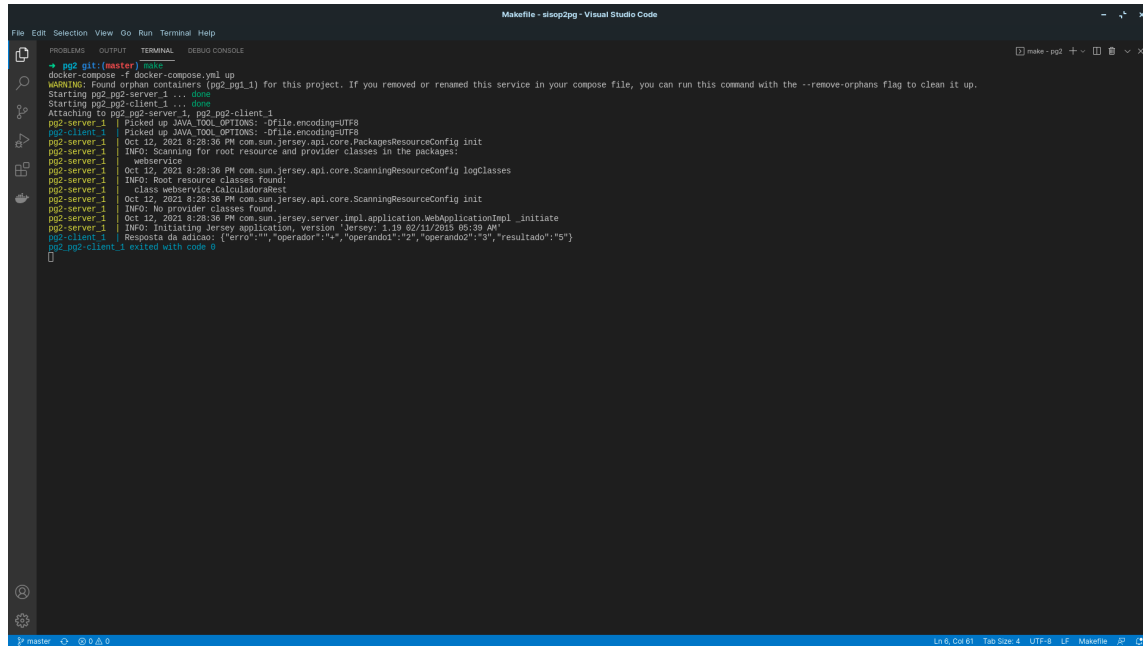
        ClientResponse respostaSomarInt = servicoSomarInt.accept(
            MediaType.APPLICATION_JSON).get(ClientResponse.class);

        String respostaJsonSomarInt = respostaSomarInt.getEntity(String
            .class);
        System.out.println("Resposta da adicao: " +
            respostaJsonSomarInt);
    }
}
```

```
}  
}
```

3 Demonstração

A aplicação foi executada dentro de *containers Docker* para simplificar a instalação dos requisitos.



```
Makefile - sisop2pg - Visual Studio Code  
File Edit Selection View Go Run Terminal Help  
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE  
+ pg2 git:(master) make  
docker-compose -f docker-compose.yml up  
WARNING: Found orphan containers (pg2_pg1_1) for this project. If you removed or renamed this service in your compose file, you can run this command with the --remove-orphans flag to clean it up.  
Starting pg2_pg2-server_1 ... done  
Starting pg2_pg2-client_1 ... done  
Attaching to pg2_pg2-server_1, pg2_pg2-client_1  
pg2_server_1 | Picked up JAVA_TOOL_OPTIONS: -Dfile.encoding=UTF8  
pg2_client_1 | Picked up JAVA_TOOL_OPTIONS: -Dfile.encoding=UTF8  
pg2_server_1 | Oct 12, 2021 8:28:36 PM com.sun.jersey.api.core.PackagesResourceConfig init  
pg2_server_1 | INFO: Scanning for root resource and provider classes in the packages:  
pg2_server_1 | webservice  
pg2_server_1 | Oct 12, 2021 8:28:36 PM com.sun.jersey.api.core.ScanningResourceConfig logClasses  
pg2_server_1 | INFO: Root resource classes found:  
pg2_server_1 | class webservice.CalculadoraRest  
pg2_server_1 | Oct 12, 2021 8:28:36 PM com.sun.jersey.api.core.ScanningResourceConfig init  
pg2_server_1 | INFO: No provider classes found.  
pg2_server_1 | Oct 12, 2021 8:28:36 PM com.sun.jersey.server.impl.application.WebApplicationImpl _initiate  
pg2_server_1 | INFO: Initiating Jersey application, version 'Jersey: 1.9.42/11/05/15 05:39 AM'  
pg2_client_1 | Resposta da adicao: {"erro":"","operador":"+","operando1":"2","operando2":"3","resultado":"5"}  
pg2_pg2-client_1 exited with code 0
```

Figure 1: Execução do servidor seguida da execução do cliente