REST - UMA BREVE INTRODUÇÃO

MCTA025-13 - SISTEMAS DISTRIBUÍDOS

Emilio Francesquini e Fernando Teubl

25 de julho de 2018

Centro de Matemática, Computação e Cognição
Universidade Federal do ABC

UFABC

DISCLAIMER

- Estes slides foram preparados para o curso de **Sistemas Distribuídos na UFABC**.
- Este material pode ser usado livremente desde que sejam mantidos, além deste aviso, os créditos aos autores e instituições.
- Estes slides foram adaptados com base no material disponível em
  - A Brief Introduction to REST - `http://www.infoq.com/articles/rest-introduction`
  - JSR 311: JAX-RS: The Java API for RESTful Web Services - `https://jcp.org/en/jsr/detail?id=311`
  - Jersey - RESTful Web Services in Java - `https://jersey.github.io`
  - Putting Java to REST - `http://java.dzone.com/articles/putting-java-rest`
  - REST Tutorial - `https://www.devmedia.com.br/rest-tutorial/28912`

REST

- **RE**presentational **S**tate **T**ransfer
- Serviços que seguem essa linha são comumente chamados de RESTful
- Modelo apresentado na tese de doutoramento de Roy T. Fielding (UC Irvine, 2000)

REST - PRINCÍPIOS

- Atribuição de identificadores
- Associação de recursos
- Utilização de métodos padrão
- Multiplicidade de representações
- Comunicação sem manutenção de estado

IDENTIFICADORES

- Todas as abstrações do sistema merecem ser identificadas
- Não apenas uma abstração individualmente deve ser identificada, mas qualquer conjunto destas abstrações que faça sentido também pode ser identificado por um id. Por exemplo:
  - Todas as latas de ervilha
  - Todas as pessoas que tem endereço em SP

- Na web, o jeito mais natural de fazer tal identificação é através de URIs
  - `http://example.com/customers/1234`
  - `http://example.com/orders/2007/10/776654`
  - `http://example.com/orders/2007/11`
  - `http://example.com/products?color=green`
- Sem paúra!
  - Anos de prática OO nos ensinaram a não expor os detalhes de implementação, ainda mais ids do BD
  - A idéia toda é a de que os conceitos (recursos) que você desejará expor são geralmente muito mais abrangentes do que uma linha no BD

ASSOCIAÇÃO - HATEOAS

- HATEOAS - Hypermedia As The Engine Of Application State
- Significa que tudo deve ser ligado. Considere o seguinte exemplo de uma resposta a uma requisição de pesquisa por pedidos

```
<pedido ref='http://xyz.com/pedido/1234'>
  <qtd>23</qtd>
  <produto ref='http://abc.com/produtos/4554'/>
  <cliente ref='http://jkl.com/clientes/7654'/>
</pedido>
```

MÉTODOS DE ACESSO PADRONIZADOS

Pode-se imaginar que todo recurso é algo semelhante a:

```
class Resource {
 Resource(URI u);
 Response get();
 Response post(Request r);
 Response put(Request r);
 Response delete();
}
```

Onde:

- GET – Pega uma representação do recurso
- PUT – Cria ou atualiza, se já existir o recurso, com as informações passadas
- DELETE – Apaga o recurso
- POST – Única operação **não idempotente** (ao menos na definição original). Geralmente significa criação de um recurso ou ativação de algum processamento arbitrário.

MÉTODOS DE ACESSO PADRONIZADOS

- Essas restrições devem ser respeitadas pelos serviços RESTful
  - No entanto, na vida real, nem todas essas restrições semânticas são respeitadas e é preciso avaliar cada caso de uso individualmente
- Pode ser difícil imaginar que uma aplicação modelada de uma maneira OO possa ser mapeada desta maneira

REPRESENTADO UM SISTEMA OO

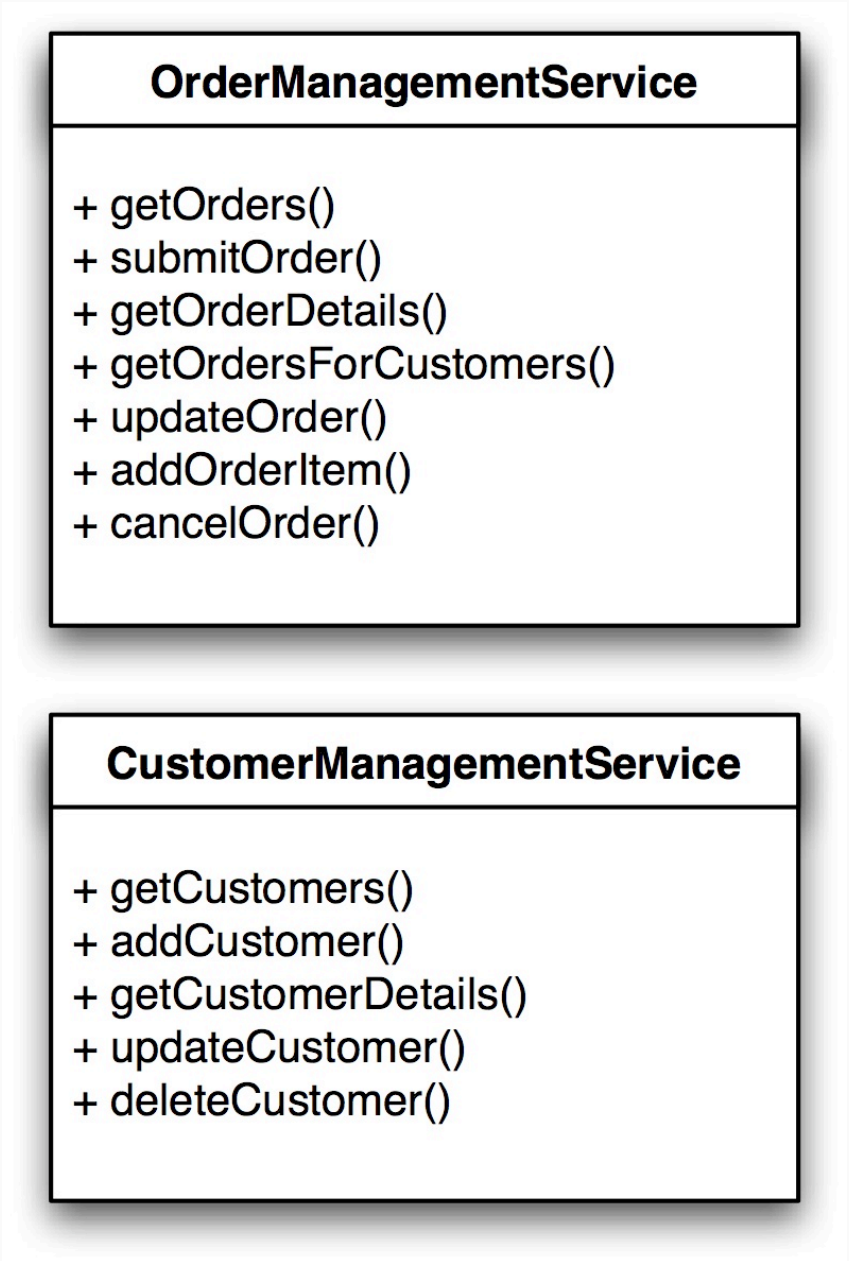


Figura: Tilkov, 2007

REPRESENTADO UM SISTEMA OO

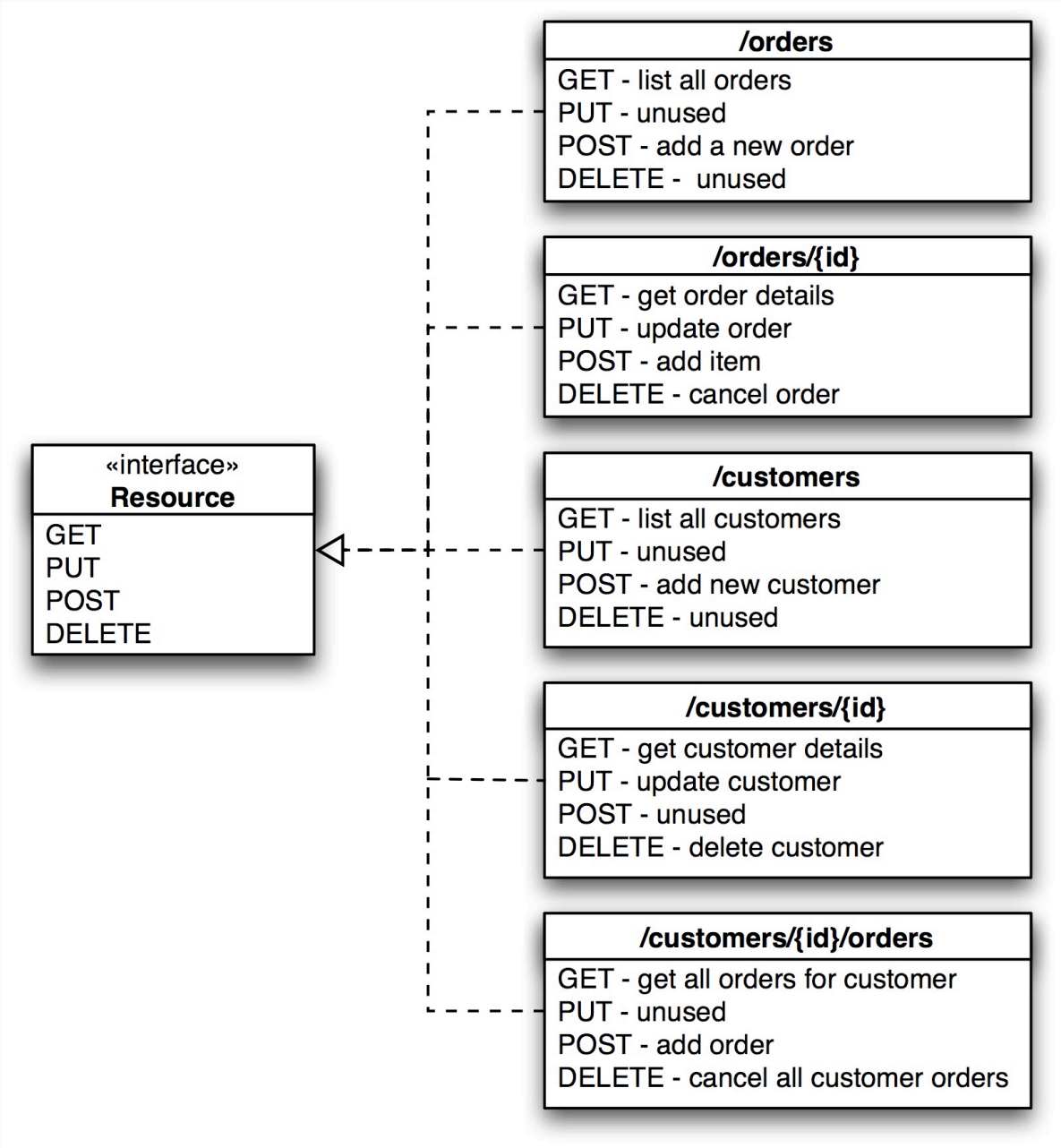


Figura: Tilkov, 2007

MÚLTIPLAS REPRESENTAÇÕES

```
GET /clientes/1234 HTTP/1.1
Host: xyz.com
Accept: application/meucliente+xml
GET /clientes/1234 HTTP/1.1
Host: xyz.com
Accept: text/x-vcard
```

- **Possível uso:** uma mesma API que gera as informações em HTML para uma página ou em um formato XML para ser usada por alguma aplicação

COMUNICAÇÃO SEM MANUTENÇÃO DE ESTADO

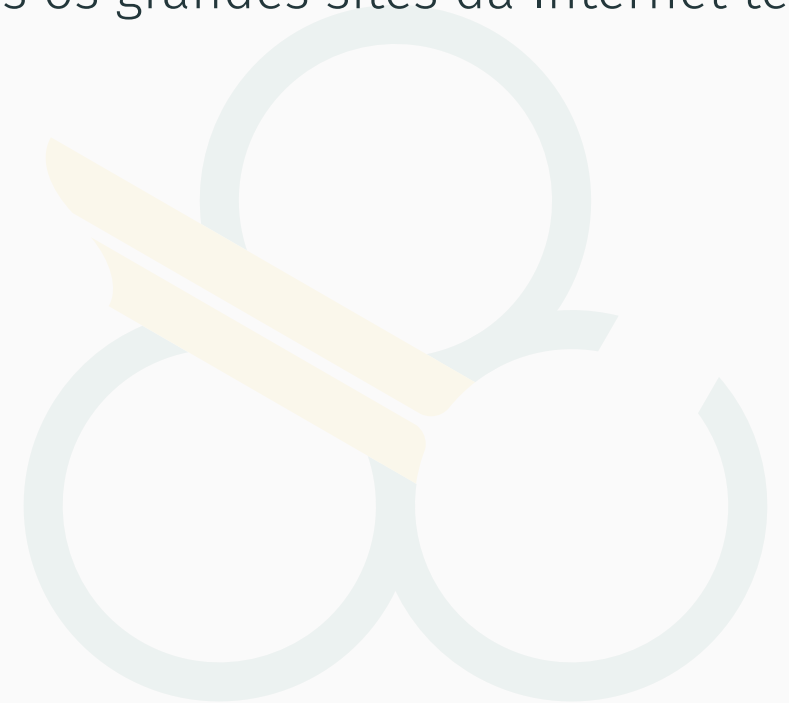
- Depois de ver os princípios anteriores, fica claro como fazer isso
- Atenção: Colocar na URI um ID de sessão não transforma seu web service em um RESTful
- Vantagens
  - Diminuição da carga do servidor
  - Escalabilidade
  - Independência da implementação do servidor
  - Independência entre servidores

- Também tem sido chamado de WebAPI
- Muito usado para fazer mashups
- Um dos pilares para as páginas de conteúdo mais modernas
- Faz a junção das informações direto no cliente e não mais no servidor

- A especificação propriamente dita não diz nada quanto a HTTP
- Também não especifica o número de operações, apenas determina que a interface deve ser padrão
- Não é demais supor que REST e HTTP tenham se influenciado mutuamente já que o Fielding, criador e descritor do REST também tenha definido muitos dos padrões HTTP

Praticamente todos os grandes sites da Internet tem APIs REST.

- Twitter
- Flickr
- Google
- Amazon
- Facebook
- ...



JAX-RS

- Criado para facilitar a implementação de serviços REST em Java
- Baseado em anotações
- Evita a tarefa um tanto entediante de fazer o processamento de URIs, XMLs, ...

REST - JAX-RS - SERVIÇO SIMPLES

```java
@Path("/orders")
public class OrderEntryService {
  @GET
  public String getOrders() {...}
}
```

- Responde por requisições GET no endereço:
  - `http://exemplo.com/orders`

```java
@Path("/orders")
public class OrderEntryService {
  @GET
  public String getOrders(@QueryParam("size")
                          @DefaultValue("50")
                          int size) {
    ...
  }
}
```

- Responde por requisições GET no endereço:
  - http://exemplo.com/orders?size=30

```java
@Path("/orders")
public class OrderEntryService {
   @GET
   @Path{"/{id}"}
   public String getOrder(@PathParam("id")
                          int orderId){
      ...
   }
}
```

- Responde por requisições GET no endereço:
  - `http://exemplo.com/orders/3333`
- Mas não no endereço:
  - `http://exemplo.com/orders/3333/entries`
- Há também a possíbilidade de se usar expressões regulares:

```java
@Path("{id: \\d+}")
```

```
@Path("/orders")
public class OrderEntryService {
  @GET
  @Path("{id}")
  @Produces("application/xml")
  public String getOrder(@PathParam("id")
                         int orderId) {
    ...
  }
}
```

- Só será chamado caso o cliente envie no pedido que ele aceita `application/xml`
- O Firefox, por exemplo, envia como tipos aceitos:

```
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
```

REST - JAX-RS - NEGOCIAÇÃO DE CONTEÚDO

```java
@Path("/orders")
public class OrderEntryService {
  @GET
  @Path("{id}")
  @Produces("application/xml")
  public String getOrder(@PathParam("id") int orderId) {...}

  @GET
  @Path("{id}")
  @Produces("text/html")
  public String getOrderHtml(@PathParam("id") int orderId) {...}

  @GET
  @Path("{id}")
  @Produces("application/json")
  public String getOrderJson(@PathParam("id") int orderId) {...}

}
```

- Em vez de fazermos o empacotamento (*marshalling*) de conteúdo manualmente (note o tipo de retorno sendo `String` nos exemplos), podemos fazer automaticamente

```java
@Path("/orders")
public class OrderEntryService {
  @GET
  @Path("{id}")
  @Produces("application/xml")
  public Order getOrder(@PathParam("id") int orderId) {...}
  ...
}
@XmlRootElement(name="order")
public class Order {
  @XmlElement(name="id")
  int id;
  @XmlElement(name="order-entries")
  List<OderEntry> entries;
  ...
}
```

- O empacotamento também pode ser feito automaticamente caso a classe do objeto devolvido como retorno obedeça as regras de um *bean*, o seja, tenha definido os *getters* e *setters*.

EXEMPLO - CRIAÇÃO DE UMA AGENDA

- Vamos criar uma agenda que guarda
  - Nomes
  - Emails
  - Telefones
- A linguagem será Java utilizando Jersey

- Para criar um projeto Maven básico com tudo o que precisamos execute a seguinte linha de comando em um terminal:

```
$ mvn archetype:generate \
 -DarchetypeArtifactId=jersey-quickstart-grizzly2 \
 -DarchetypeGroupId=org.glassfish.jersey.archetypes \
 -DinteractiveMode=false -DgroupId=sd \
 -DartifactId=servico-agenda -Dpackage=sd \
 -DarchetypeVersion=2.27
```

- Caso receba um erro que o mvn não foi encontrado, instale-o fazendo `sudo apt-get install maven`

- Edite o arquivo `pom.xml` para habilitar o marshalling/unmarshalling automático para Json
  - Basta descomentar as linhas indicadas
- Altere o arquivo `Main.java` para que ele passe a escutar na raiz do nosso servidor (pode alterar a porta também caso necessário.
  - Basta alterar a linha 16 para que fique como abaixo:

```
public static final String BASE_URI =
  "http://localhost:8080/";
```

- O Maven já vai ter criado um serviço simples cuja implementação está em `MyResource.java`

```java
package sd;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("myresource")
public class MyResource {
    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String getIt() {
        return "Got it!";
    }
}
```

EXECUTANDO O PROJETO

- Para executá-lo, basta rodar na raiz do projeto:

```
$ mvn compile
$ mvn exec:java

INFO: [HttpServer] Started.
Jersey app started with WADL available at
http://localhost:8080/application.wadl
Hit enter to stop it...
```

Conteúdo de `http://localhost:8080/application.wadl`

```
<application xmlns="http://wadl.dev.java.net/2009/02">
...
<grammars/>
  <resources base="http://localhost:8080/">
    <resource path="myresource">
      <method id="getIt" name="GET">
        <response>
          <representation mediaType="text/plain"/>
        </response>
      </method>
    </resource>
  </resources>
</application>
```

A classe Email

```
package sd;
public class Email {
    private String tipo;
    private String endereco;
    public Email() {}
    public Email(String tipo, String endereco) {
        this.tipo = tipo;
        this.endereco = endereco;
    }
    public String getTipo() {...}
    public void setTipo(String tipo) {...}
    public String getEndereco() {...}
    public void setEndereco(String endereco) {...}
    public String toString() {...}
}
```

A classe Telefone

```
package sd;
public class Telefone {
    private String tipo;
    private String numero;
    public Telefone() {}
    public Telefone(String tipo, String numero) {
        this.tipo = tipo;
        this.numero = numero;
    }
    public String getTipo() {...}
    public void setTipo(String tipo) {...}
    public String getNumero() {...}
    public void setNumero(String numero) {...}
    public String toString() {...}
}
```

A classe Contato

```
package sd;
...
public class Contato {
    private int id;
    private String nome;
    private List<Telefone> telefones = new ArrayList<Telefone>();
    private List<Email> emails = new ArrayList<Email>();

    public int getId() {...}
    public String getNome() {...}
    public List<Telefone> getTelefones() {...}
    public List<Email> getEmails() {...}
    public void setId(int id) {...}
    public void setNome(String nome) {...}
    public void setTelefones(List<Telefone> telefones) {...}
    public void setEmails(List<Email> emails) {...}
    ...
}
```

A classe ServicoContatos

```
package sd;
...
@Path("contatos")
public class ServicoContatos {
  private static Map<Integer, Contato> contatos =
    new HashMap<Integer, Contato>();
  private static int proximoId = 1;
  private void criaContato(String nome,
                           String emailTipo, String email,
                           String telTipo, String tel) {
    int id = proximoId++;
    Contato contato = new Contato();
    contato.setId(id);
    contato.setNome(nome);
    contato.getEmails().add(new Email(emailTipo, email));
    contato.getTelefones().add(new Telefone(telTipo, tel));
    contatos.put(id, contato);
  }
  ...
```

A classe `ServicoContatos`

```
...
public ServicoContatos() {
  if (contatos.isEmpty()) {
    criaContato("E. Francesquini",
                "Profissional", "e.francesquini@ufbac.edu.br",
                "Comercial", "4996 8327");
    criaContato("F. Ferreira ",
                "Profissional", "fernando.teubl@ufabc.edu.br",
                "Residencial", "1234 5678");
  }
}
...
```

A classe ServicoContatos - GET

```
...
  @GET
  @Produces(MediaType.TEXT_PLAIN)
  public String listPlain() {
      System.out.println("listPlain");
      StringBuffer ret = new StringBuffer();
      for (Contato contato :contatos.values()) {
          ret.append(contato);
          ret.append("\n");
      }
      return ret.toString();
  }

  @GET
  @Produces(MediaType.APPLICATION_JSON)
  public List<Contato> listJson() {
      System.out.println("listJson");
      return new ArrayList(contatos.values());
  }
...
```

A classe ServicoContatos - GET

```
...
@GET
@Produces(MediaType.APPLICATION_JSON)
@Path("/{id}")
public Contato getContato(@PathParam("id") int id) {
    System.out.println("getContato");
    Contato ret = contatos.get(id);
    if (ret == null)
        throw new WebApplicationException(404);
    return ret;
}
...
```

A classe ServicoContatos - GET

```
...
@GET
@Path("/busca/{nome}")
@Produces(MediaType.APPLICATION_JSON)
public List<Contato> findByName(@PathParam("nome") String nome) {
    System.out.println("findByName");
    String lcNome = nome.toLowerCase();
    List<Contato> ret = new ArrayList<Contato>();
    for (Contato contato : contatos.values()) {
        if (contato.getNome() != null
            && contato.getNome().toLowerCase().contains(lcNome)) {
            ret.add(contato);
        }
    }
    return ret;
}
...
```

A classe ServicoContatos - POST

```
...
@POST
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
public Contato add(Contato contato) {
    System.out.println("add");
    if (contato.getNome() == null ||
        contato.getNome().trim().equals("")) {
        throw new WebApplicationException(Response
                                .status(Response.Status.BAD_REQUEST)
                                .entity("Nome é obrigatorio").build());
    }
    contato.setId(proximoId++);
    contatos.put(contato.getId(), contato);
    return contato;
}
...
```

A classe `ServicoContatos` - PUT

```
...
@PUT
@Path("/{id}")
@Consumes(MediaType.APPLICATION_JSON)
public Response update(@PathParam("id") int id, Contato contato) {
    System.out.println("update");
    if (id != contato.getId()) {
        throw new WebApplicationException(Response
                                .status(Response.Status.BAD_REQUEST)
                                .entity("O id deve ser igual").build());
    }
    contatos.put(id, contato);
    return Response.ok().build();
}
...
```

A classe ServicoContatos - DELETE

```
...
@DELETE
public Response delete(@QueryParam("id") int id) {
    System.out.println("delete");
    Contato cont = contatos.get(id);
    if (cont == null) {
        System.out.println("Não achou");
        throw new WebApplicationException(404);
    }
    contatos.remove(id);
    return Response.ok().build();
}
...
```

O WADL

```
<application xmlns="http://wadl.dev.java.net/2009/02">
...
<resources base="http://localhost:8080/">
  <resource path="contatos">
  <method id="listPlain" name="GET">
  <response> <representation mediaType="text/plain"/>  </response>
  </method>
  <method id="listJson" name="GET">
    <response> <representation mediaType="application/json"/> </respon
  </method>
  <method id="add" name="POST">
    <request>
      <representation mediaType="application/json"/>
    </request>
    <response> <representation mediaType="application/json"/> </respon
  </method>
  ...
```

Pode-se acessar diretamente pela linha de comando:

```
$ curl -H "Accept:text/plain" \
  --request GET http://localhost:8080/contatos/
{ 1, E. Francesquini, [{ Profissional:e.francesquini@ufbac.edu.br }],
  [{ Comercial:9876 5432 }] }
{ 2, F. Ferreira , [{ Profissional:fernando.teubl@ufabc.edu.br }],
  [{ Residencial:1234 5678 }] }
$
$ curl -H "Accept:application/json" \
  --request GET http://localhost:8080/contatos/
[{"emails":[{"endereco":"e.francesquini@ufbac.edu.br",
"tipo":"Profissional"}],"id":1,"nome":"E. Francesquini",
"telefones":[{"numero":"9876 5432","tipo":"Comercial"}]},
{"emails":[{"endereco":"fernando.teubl@ufabc.edu.br",
"tipo":"Profissional"}],"id":2,"nome":"F. Ferreira ",
"telefones":[{"numero":"1234 5678","tipo":"Residencial"}]}]
$
```

ACESSANDO O SERVIÇO REST

```
$ curl --request DELETE http://localhost:8080/contatos?id=1
$
$ curl -H "Accept:application/json" \
  --request GET http://localhost:8080/contatos/
[{"emails":[{"endereco":"fernando.teubl@ufabc.edu.br",
"tipo":"Profissional"}],"id":2,"nome":"F. Ferreira ",
"telefones":[{"numero":"1234 5678","tipo":"Residencial"}]}]
$
```

ACESSANDO O SERVIÇO REST

```
$ curl -H "Content-Type:application/json" \
  --request POST http://localhost:8080/contatos/ \
  -d '{"emails":[{"endereco":"maluco@ufabc.edu.br", \
  "tipo":"Pessoal"}], "id":3, "nome":"Maluco UFABC",\
  "telefones":[{"numero":"99999999","tipo":"Celular"}]}'

{"emails":[{"endereco":"maluco@ufabc.edu.br","tipo":"Pessoal"}],
"id":3,"nome":"Maluco UFABC","telefones":[{"numero":"99999999",
"tipo":"Celular"}]}
$
$ curl -H "Accept: application/json" \
  --request GET http://localhost:8080/contatos/
[{"emails":[{"endereco":"fernando.teubl@ufabc.edu.br",
"tipo":"Profissional"}],"id":2,"nome":"F. Ferreira ",
"telefones":[{"numero":"1234 5678","tipo":"Residencial"}]},
{"emails":[{"endereco":"maluco@ufabc.edu.br","tipo":"Pessoal"}],
"id":3,"nome":"Maluco UFABC","telefones":[{"numero":"99999999",
"tipo":"Celular"}]}]
$
```

ACESSO VIA JAVA - VERSÃO NA UNHA

```
public class ClienteREST {
  public static void main(String[] args) {
    ...
    URL url = new URL("http://localhost:8080/contatos");
    HttpURLConnection conn = (HttpURLConnection) url.openConnection();
    conn.setRequestMethod("GET");
    conn.setRequestProperty("Accept", "application/json");
    if (conn.getResponseCode() != 200) {
      throw new RuntimeException("Failed : HTTP error code : "
        + conn.getResponseCode());
    }
    BufferedReader br = new BufferedReader(new InputStreamReader(
      (conn.getInputStream())));
    String output;
    System.out.println("Output from Server .... \n");
    while ((output = br.readLine()) != null) {
      System.out.println(output);
    }
    conn.disconnect();
    ...
  }
}
```

ACESSO VIA JAVA - VERSÃO SIMPLIFICADA

```java
import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder;
import javax.ws.rs.client.WebTarget;

public class ClienteREST {
  public static void main(String[] args) {
    Client c = ClientBuilder.newClient();
    WebTarget target = c.target("http://localhost:8080/");
    String responseMsg = target.path("contatos")
      .request().get(String.class);
    System.out.println(resposeMsg);
  }
}
```

Veja o exemplo de uso no diretório `src/test/` criado pelo Maven.