REST
REpresentational State Transfer

Prof. Dr. Alexandre L'Erario

Introdução

- É um webservice!
  - Objetivo: interoperabilidade entre aplicativos
- Estrutura diferenciada do tradicional SOAP
- Resolve problemas do WebService SOAP
  - Mais leve que o SOAP → menos overhead
- Amplamente utilizado na Internet

REST

- *REpresentational State Transfer*
- Utiliza os métodos existentes do HTTP
- Aplicação que utiliza REST é RESTful
- A informação disponibilizada é um *resource*
  - Como se fosse um objeto

Operações – Baseadas no RDF

HTTP Verb	Behavior
GET	**Returns a serialization of the RDF graph which encodes the state of the given resource**. Content negotiation MUST be performed. In level zero, at least one of application/rdf+xml or text/turtle SHOULD be available from the server.
HEAD	As normal, on the information GET would return. In particular, **metadata may be returned** using Link Headers, indicating, for instance, a SPARQL endpoint which can be used to query the data.
POST	Not specified in general, to allow for application use.
PUT	If the media type of the payload is an RDF graph serialization language, then set the resource state to be as encoded in the serialized RDF graph. Otherwise, not specified in level zero. **Creates the resource, if it does not already exist**. Some resources may be flagged "no clobber", to reject PUT if they already exist; level zero does not specify how to indicate this.
DELETE	**Remove the association between the resource and the URL used in the delete operation**. The server MAY retain the underlying resource, perhaps accessible via a different URL. That is, a successful DELETE removes this one reference to the resource, but does not necessarily affect other references, or the resource itself. Creating multiple references or determining whether they exist is beyond the scope of level zero.
PATCH	**Modify the state of the resource as specified by the payload, according to its media type**. Servers MAY accept SPARQL 1.1 Update (Content-Type: application/sparql-update) on element resource to modify the RDF graph view of the resource state, acting as a SPARQL endpoints with only a default graph.

https://www.w3.org/2001/sw/wiki/REST

Métodos HTTP definidos no rfc2616

O que é RDF

- *Resource Description Framework*
- Semelhante ao modelo Entidade-relacionamento
- “uses *subject* instead of *object* (or *entity*)”
- Auto-descritivo

Rest – Exemplo de representações

```
<lista_alunos>
   <aluno id="1" nome="Alexandre " ra="RA06978">
      <disciplinas>
         <disciplina>
            <nome>Redes de computadores</nome>
         </disciplina>
         <disciplina>
            <nome>Java X</nome>
         </disciplina>
      </disciplinas>
   </aluno>
</lista_alunos>
```

JSON

```
{
   "lista_alunos": {
      "aluno": {
         "disciplinas": {
            "disciplina": [
               {
                  "nome": "Redes de computadores"
               },
               {
                  "nome": "Java X"
               }
            ]
         },
         "_id": "1",
         "_nome": "Alexandre ",
         "_ra": "RA06978"
      }
   }
}
```

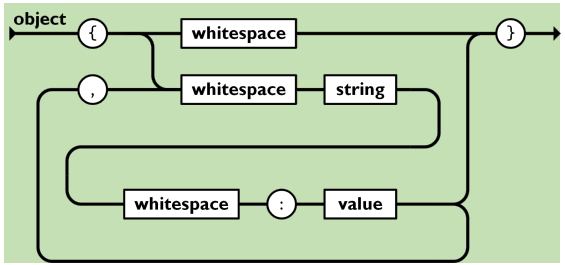
XML

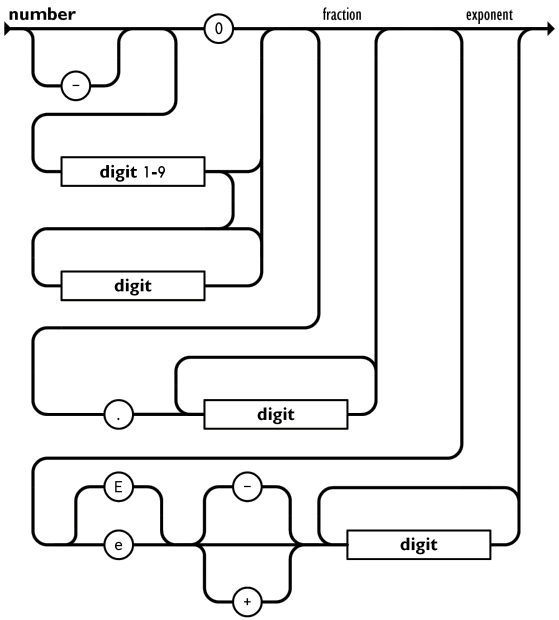
JSON - JavaScript Object Notation

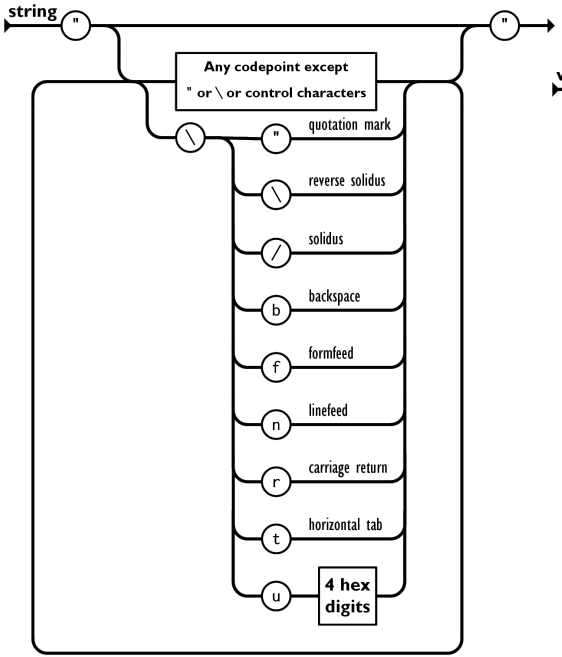
- Representação por pares rótulo:significado.
  - Exemplos: ”email": “alerario@gmail.com”, "ano": 2012
  - Array: [“Assis”, “Cornélio Procópio”, “Londrina”]
  - Objeto

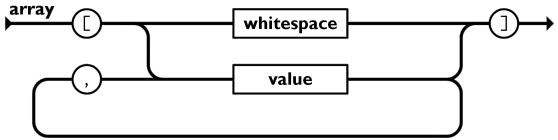
```
filme = {
“titulo”: “JSON não é linguagem!”,
“resumo”: “Representação de informações”,
“ano”: 2022,
“genero”: [“aventura”, “drama”, “ficção”]
}
// para acessar ➔ filme.ano
```

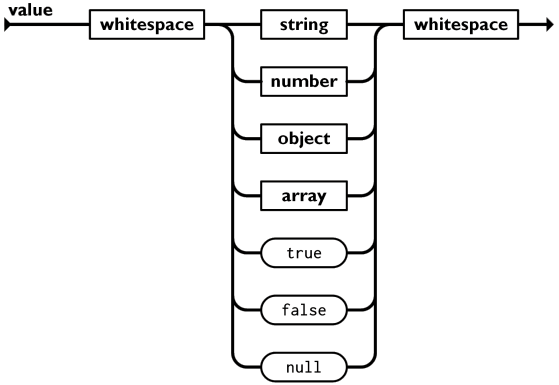
JSON – JavaScript Object Notation (json.org)

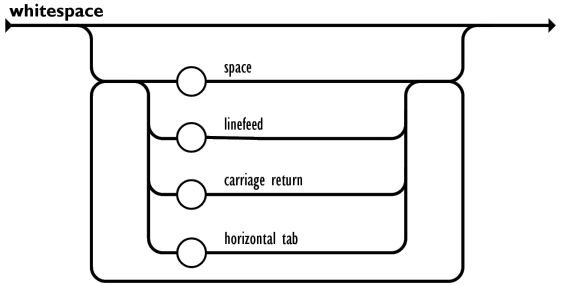












Desenvolver uma aplicação Restful (Passos)

1. Identificar modelo de objetos (*resources*)

2. Criar modelo de URIs. Exemplos:
   http://localhost:8080/aplicativo/servicorest
   http://localhost:8080/aplicativo /servicorest{id_obj}
   http://localhost:8080/aplicativo /servicorest{id_obj}/subservico{id}

3. Definir modelo de representação (XML vs JSON)

4. Associar modelo aos métodos HTTP

WADL

- *Web Application Description Language*
- Descreve em XML a aplicação Web-based
  - Muitas vezes no HTTP
- Descreve:
  - Conjunto de recursos
  - Relação entre recursos
  - Métodos de acesso para cada recurso
  - Formato de representação do recurso

Jakarta RESTful Web Services

- JAX-RS
- https://projects.eclipse.org/projects/ee4j.jaxrs
- Novo pacote: jakarta.ws.rs
- Mantém o uso das *annotations* ☺
  - @ApplicationPath("resources") // definir o uri dos serviços
  - @Path(”service") // definir o uri do serviço
  - @GET, @PUT, @POST, @DELETE ...
- WADL gerado automaticamente
  - http://endereço:porta/resources/application.wadl

Jax-RS - Exemplo

```
//Classe de configuraçao
@ApplicationPath("resources")
public class JakartaRestConfiguration extends Application {

}
```

Define URI de acesso aos recursos

// jakarta.ws.rs.core.Application;

```
//classe de recurso
@Path("rest")
public class JakartaEE9Resource {

    @GET
    @Path("ola/{nome}")
    @Produces(MediaType.TEXT_PLAIN)
    public String oi(@PathParam("nome") String nome){
        return "ola, seja bem vindo " + nome;
    }
```

Define URI de acesso ao serviço

Define a operação (@PUT, @POST, @DELETE)

Adiciona um caminho e parâmetro na URI

Indica o que é produzido pelo método.
Inverso de “@Consumes”

- APPLICATION_JSON
- TEXT_HTML
- TEXT_PLAIN
- TEXT_XML

Obtém o parâmetro {nome}

Continuação do exemplo

Três métodos para incluir uma **Cidade**, sendo:

1. Com parâmetros na URI

```
@PUT
  @Path("cidade/{id}/{nome}")
  @Consumes(MediaType.APPLICATION_JSON)
  public void add(@PathParam("id") int id,@PathParam("nome") String nome){
          Cidade cid = new Cidade();
          cid.setId(id);
          cid.setNome(nome);
          new CrudCidade().add(cid);
  }
```

2. Com um JSON

```
@PUT
  @Path("cidade")
  @Consumes(MediaType.APPLICATION_JSON)
  public void add(JsonObject jsonData){
          Cidade cid = new Cidade();
          cid.setId(jsonData.getInt("codigo"));
          cid.setNome(jsonData.getString("nome"));
          new CrudCidade().add(cid);
  }
```

3. Com um objeto

```
@PUT
  @Path("cidadeobj")
  @Consumes(MediaType.APPLICATION_JSON)
  public void addObj(Cidade cid){
          new CrudCidade().add(cid);
  }
```

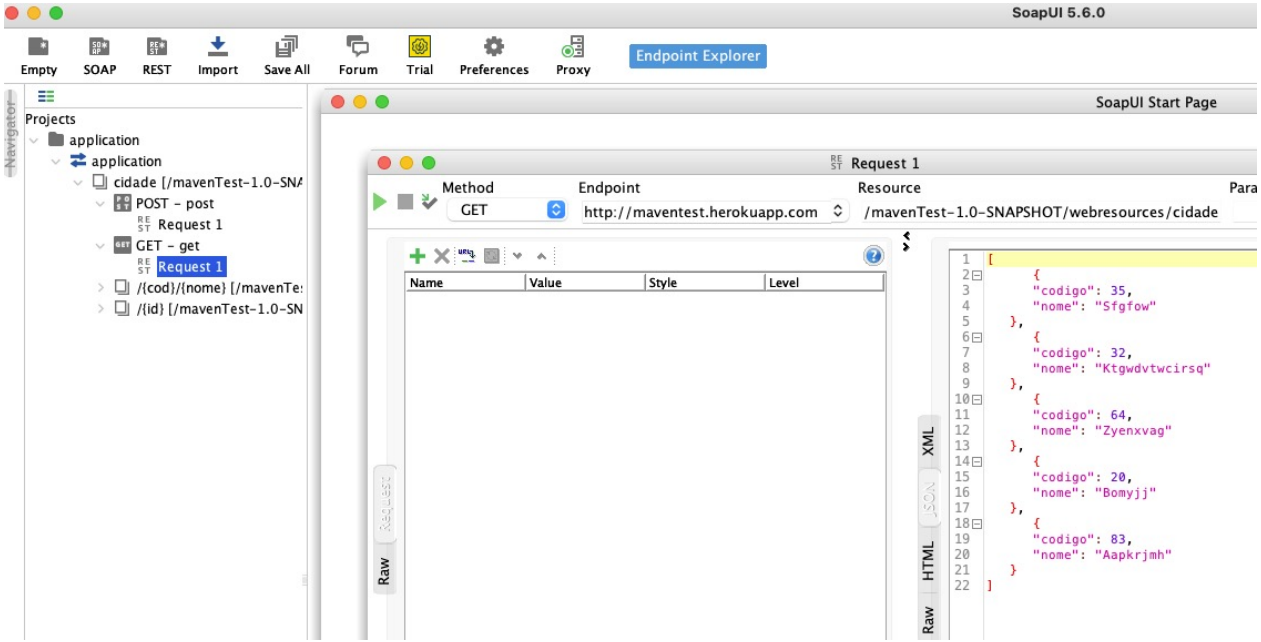
Obter lista de cidades

```
@GET
  @Path("cidades")
  @Produces(MediaType.APPLICATION_JSON)
  public ArrayList<Cidade> getAllCidades(){
     ArrayList<Cidade> lista = new CrudCidade().getAll();
     return lista;
  }
```

Testar serviços Rest

- Aplicativos de teste e análise

SoapUI
Supported by SMARTBEAR



JAX-RS Exemplo de um cliente

```
public static void main(String[] args) {
        RestClient rc = new RestClient();

    //adicionando cidades..
    rc.add(1, "Assis");
    rc.addJson(2, "Cornelio procopio");

    Cidade cid = new Cidade();
    cid.setId(3);
    cid.setNome("Londrina");
    rc.addObj(cid);
```

```
// Listando cidades
    for (Cidade cidade : rc.getCidades()) {
        System.out.println(cidade.getNome());
    }
    // Obter uma cidade

    cid = rc.getCidade(1);
    System.out.println(cid.getNome());
    rc.close();

}
```

JAX-RS cliente – RestClient.java

```
public class RestClient {
   private WebTarget webTarget;
   private Client client;
   private static final String BASE_URI = "http://localhost:8080/RestProvider/resources/";
   public RestClient() {
      client = jakarta.ws.rs.client.ClientBuilder.newClient();
      webTarget = client.target(BASE_URI).path("rest");
   }

   public String ola(String nome) throws ClientErrorException {
      WebTarget resource = webTarget;
      resource = resource.path(
                  java.text.MessageFormat.format("ola/{0}", new Object[]{nome}));
      return resource.request().get(String.class);
   }
   public void close() {
      client.close();
   }
}
```

```
//adicionar apenas com parametros
   public void add(int codigo, String nome) throws ClientErrorException {
      Response response = webTarget.path("cidade")
            .path("/" + codigo + "/" + nome)
            .request()
            .put(Entity.entity("", MediaType.APPLICATION_JSON));
   }

   // adicionar com json
   public void addJson(int codigo, String nome) throws ClientErrorException {
      JsonObject value = Json.createObjectBuilder()
            .add("codigo", codigo)
            .add("nome", nome)
            .build();

      Response response = webTarget.path("cidade")
            .request()
            .put(Entity.entity(value, MediaType.APPLICATION_JSON));

   }

//adicionar com objeto cidade
   public void addObj(Cidade cid) throws ClientErrorException {
      Response response = webTarget.path("cidadeobj")
            .request()
            .put(Entity.entity(cid, MediaType.APPLICATION_JSON));
   }
```

JAX-RS - Cliente

```
//obter cidades
   public ArrayList<Cidade> getCidades() throws ClientErrorException {
      ArrayList resource = webTarget.path("cidades").request(MediaType.APPLICATION_JSON).get(ArrayList.class);

     //converter Arraylist de hashmap para ArrayList de Cidade
      ArrayList<Cidade> lcid = new ArrayList<Cidade>();
      for (Object object : resource) {
         HashMap hm = (HashMap) object;
         Cidade cid= new Cidade();
         cid.setId(((BigDecimal) hm.get("id")).intValue());
         cid.setNome((String) hm.get("nome"));
         lcid.add(cid);
      }
      return lcid;
   }

   //obter uma cidade
   public Cidade getCidade(int id) throws ClientErrorException {
      Cidade resource = webTarget.path("cidade")
            .path("/" + id)
            .request(MediaType.APPLICATION_JSON).get(Cidade.class);
      return resource;
   }
```

Considerações finais

- RESTful : interoperabilidade total e leve
- Demais parâmetros do HTTP também são usados: POST, DELETE