SISTEMAS DISTRIBUÍDOS

Aula 1

REST em Java

JAX-RS (Jersey)

REST : REPRESENTATIONAL STATE TRANSFER

Padrão arquitetural para acesso a informação

Aproximação: uma aplicação vista como uma coleção de recursos

Um recurso é identificado por um URI/URL

Um URL devolve um documento com a representação do recurso

Um URL pode referir uma coleção de recursos

Podem-se fazer referências a outros recursos usando *ligações (links)*

REST: Aspetos a destacar

Protocolo cliente/servidor **stateless**: cada pedido contém toda a informação necessária para ser processado

- Objetivo: tornar o sistema simples.

Interface uniforme: todos os recursos são acedidos por um conjunto de operações HTTP bem definidas

- POST – criar
- GET – obter
- PUT – atualizar/substituir
- DELETE – remover

EXEMPLO: RENDEZVOUS SERVICE

Ideia: Oferecer acesso a um serviço que mantém a localização (URL) de um conjunto de servidores. Os servidores registam o seu Endpoint no serviço e podem encontrar-se mutuamente através dele.

```
/**
 * Interface do servidor que mantem lista de servidores de indexacao.
 */
public interface RendezVousAPI {

    /**
     * Devolve array com a lista de servidores registados.
     */
    Endpoint[] endpoints();

    /**
     * Regista novo servidor.
     */
    void register(String id, Endpoint endpoint);

    /**
     * De-regista servidor, dado o seu id.
     */
    void unregister(String id);
}
```

EXEMPLO: RENDEZVOUS SERVICE

Ideia: Oferecer acesso a um serviço que mantém a localização (URL) de um conjunto de servidores. Os servidores registam o seu Endpoint no serviço e podem encontrar-se mutuamente através dele.

```
public class Endpoint {

    private String url;
    private Map<String, Object> attributes;

    public Endpoint() {
    }

    public Endpoint(String url, Map<String, Object> attributes) {
        this.url = url;
        this.attributes = attributes;
    }
    …
}
```

RENDEZVOUSRESOURCES.JAVA

```
/**
 * Implementação do servidor de rendezvous em REST
 */
@Path("/contacts")
public class RendezVousResources {

    private Map<String, Endpoint> db = new ConcurrentHashMap<>();

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public Endpoint[] endpoints() {
        return db.values().toArray( new Endpoint[ db.size() ]);
    }

    …
```

RENDEZVOUSRESOURCES.JAVA

```
@Path("/contacts")
public class RendezVousResources {

    private Map<String, Endpoint> db = new ConcurrentHashMap<>();

    …

    @POST
    @Path("/{id}")
    @Consumes(MediaType.APPLICATION_JSON)
    public void register( @PathParam("id") String id, Endpoint endpoint) {

        if (db.containsKey(id))
            throw new WebApplicationException( CONFLICT );
        else
            db.put(id, endpoint);
    }
```

RENDEZVOUSRESOURCES.JAVA

```
@Path("/contacts")
public class RendezVousResources {

    private Map<String, Endpoint> db = new ConcurrentHashMap<>();

    …
    @PUT
    @Path("/{id}")
    @Consumes(MediaType.APPLICATION_JSON)
    public void update(@PathParam("id") String id, Endpoint endpoint) {

        if ( ! db.containsKey(id))
            throw new WebApplicationException( NOT_FOUND );
        else
            db.put(id, endpoint);
    }
}
```

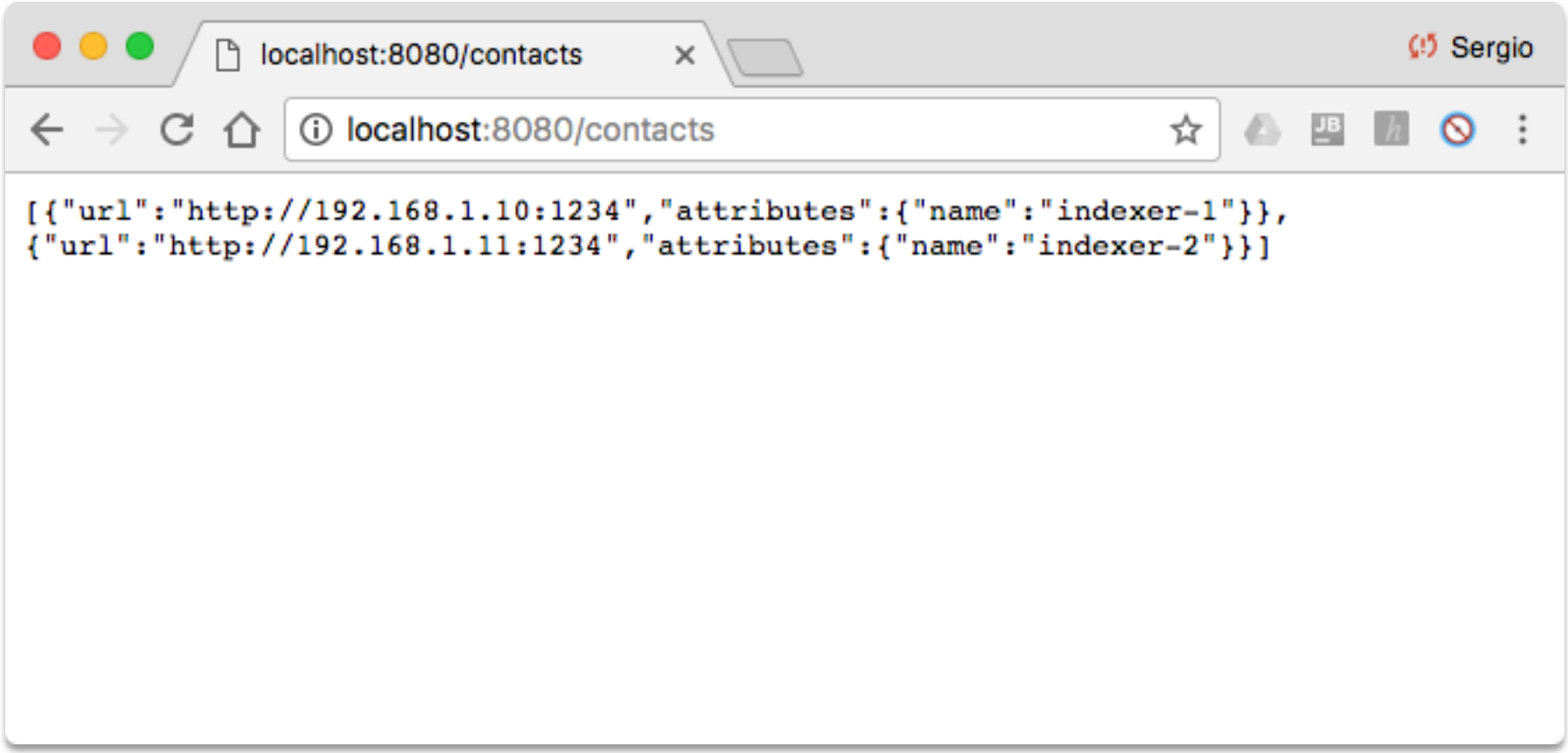
RENDEZVOUSRESOURCES.JAVA

```
@Path("/contacts")
public class RendezVousResources {

    private Map<String, Endpoint> db = new ConcurrentHashMap<>();

    …

    @POST
    @Path("/{id}")
    @Consumes(MediaType.APPLICATION_JSON)
    public void register( @PathParam("id") String id, Endpoint endpoint) {

        if (db.containsKey(id))
            throw new WebApplicationException( CONFLICT );
        else
            db.put(id, endpoint);
    }
```

RendezVousServer.java

```
public class RendezVousServer {
  public static void main(String[] args) throws Exception {
    int port = 8080;
    if( args.length > 0)
       port = Integer.parseInt(args[0]);

    URI baseUri = UriBuilder.fromUri("http://0.0.0.0/")
                            .port(port).build();

    ResourceConfig config = new ResourceConfig();
    config.register( new RendezVousResources() );

    JdkHttpServerFactory.createHttpServer(baseUri, config);

    System.err.println("REST RendezVous Server ready @ " + baseUri);
  }
}
```

O Browser como Cliente REST



LISTENDPOINTS.JAVA (RESULTADO COMO ARRAY)

```
public class ListEndpoints {

    public static void main(String[] args) throws IOException {

        ClientConfig config = new ClientConfig();
        Client client = ClientBuilder.newClient(config);


        URI baseURI = UriBuilder.fromUri("http://localhost:8080/").build();


        WebTarget target = client.target(baseURI);


        Endpoint[] endpoints = target.path("/contacts")
                    .request()
                    .accept(MediaType.APPLICATION_JSON)
                    .get(Endpoint[].class);


        System.err.println("as array: " + Arrays.asList(endpoints));
    }
}
```

LISTENDPOINTS.JAVA (RESULTADO COMO LISTA)

```
public class ListEndpoints {

    public static void main(String[] args) throws IOException {

        ClientConfig config = new ClientConfig();
        Client client = ClientBuilder.newClient(config);


        URI baseURI = UriBuilder.fromUri("http://localhost:8080/").build();


        WebTarget target = client.target(baseURI);


        List<Endpoint> endpoints = target.path("/contacts")
                    .request()
                    .accept(MediaType.APPLICATION_JSON)
                    .get(new GenericType<List<Endpoint>>() {});


        System.err.println("as array: " + endpoints);
    }
}
```

REGISTERENDPOINT.JAVA

```
public class RegisterEndpoint {

   public static void main(String[] args) throws IOException {

      ClientConfig config = new ClientConfig();
      Client client = ClientBuilder.newClient(config);

      URI baseURI = UriBuilder.fromUri("http://localhost:8080/").build();

      WebTarget target = client.target( baseURI );

      Endpoint endpoint = new Endpoint("http://some-server-endpoint-url",
                                                        Collections.emptyMap());

      Response response = target.path("/contacts/" + endpoint.generateId())
                  .request()
                  .post( Entity.entity( endpoint, MediaType.APPLICATION_JSON));

      System.out.println(response.getStatus() );
   }
}
```

CLIENTE: PUT & DELETE

PUT semelhante ao POST, serve para substituir (atualizar) um recurso.

DELETE semelhante ao GET; mas retorna apenas o HTTP status.