Inatel

PÓS-GRADUAÇÃO EM DESENVOLVIMENTO DE APLICAÇÕES PARA DISPOSITIVOS MÓVEIS E CLOUD COMPUTING

Inatel

DM107

Desenvolvimento de Web Services com Autenticação sob plataforma Java e PHP

Profa. Daniela E. C. de Almeida
E-mail: daniela.edvana@inatel.br
Prof. Elton Barbosa
E-mail: elton.barbosa@inatel.br

Agenda

- Introdução;
- Classes de Recurso;
- @Path;
- Encoding;
- @GET, @PUT, @POST, @DELETE ;
- @Produces;
- @Consumes;

Agenda

- Injeções;
- Autenticação.

Introdução

- O desenvolvimento de *Application Programming Interface* (API) RESTful que suporta de forma perfeita a exposição de seus dados em uma variedade de representação de *media types* e abstraia os detalhes de baixo nível não é uma tarefa fácil.

- O *framework* RESTful Jersey possui código aberto e foi criado para fornecer suporte para APIs JAX-RS e servir como implementação de Referência JAX-RS (JSR 311 e JSR 339).

Introdução

- Fornece sua própria API que estende o conjunto de ferramentas JAX-RS com recursos e utilitários adicionais para simplificar ainda mais o serviço RESTful e o desenvolvimento de clientes.

- Expõe inúmeras extensões para que os desenvolvedores possam expandi-lo para atender melhor às suas necessidades.

Classes de Recursos

- *Plain Old Java Object* (POJOs) são anotadas com @Path. Elas têm pelo menos um método anotado com @Path ou uma anotação de designação de método de recurso, como @GET, @PUT, @POST, @DELETE.

@Path

- Um caminho *Uniform Resource Identifier* (URI) relativo.

- Os modelos de caminho URI podem conter variáveis incorporadas na sintaxe URI. Essas variáveis são substituídas em tempo de execução para que um recurso responda a uma solicitação com base no URI substituído. As variáveis são indicadas por chaves.

@Path

- Um valor @Path pode ou não começar com um '/'.
- Um valor @Path pode ou não terminar em '/'.
- Também é possível aplicar o @Path para um método Java.

@Path

- As expressões @Path não se limitam a expressões simples, pode-se usar expressões regulares.

```
@Path("/customers")
public class CustomerResource {
    @GET
    @Path("{id : .+}")
    public String getCustomer(@PathParam("id") String id) {
    ...
    }
    @GET
    @Path("{id : .+}/address")
    public String getAddress(@PathParam("id") String id) {
    ...
    }
}
```

@Path – Regras de Precedência

- A chave primária do tipo é o número de caracteres literais na URI completa. O tipo está em ordem decrescente.
- A chave secundária do tipo é o número de expressões de modelo incorporadas no padrão.
- A chave terciária do tipo é o número de expressões de modelo não padrão. Uma expressão de modelo padrão é aquela que não define uma expressão regular.

@Path – Regras de Precedência

- Essas regras de classificação não são perfeitas. Ainda é possível ter ambigüidades, mas as regras cobrem 90% de casos de uso.

- Se a aplicação tiver ambiguidades de correspondência URI, o *design* da aplicação provavelmente também será complicado e será necessário revisar e refatorar o esquema URI.

Encoding

- A especificação URI só permite certos caracteres dentro de uma *String* URI. Ele também reserva certos caracteres para seu próprio uso.
  - Os caracteres alfabéticos US-ASCII a-z e A-Z são permitidos.
  - Os caracteres de dígito decimal 0-9 são permitidos.
  - Todos esses outros caracteres são permitidos: _- !. ~ '() *.
  - Esses caracteres são permitidos, mas são reservados para sintaxe URI: ,;: $ & + =?/ \ [] @.

@GET, @PUT, @POST, @DELETE

- São anotações de *design* do método de recursos definidas pelo JAX-RS e que correspondem aos métodos HTTP.

```
@PUT
public Response putContainer() {
    System.out.println("PUT CONTAINER " + container);
}
```

@PRODUCES

- É usada para especificar os tipos de representação de *Media Type* que um recurso pode produzir e enviar de volta para o cliente.

```
@Path("/myResource")
@Produces("text/plain")
public class SomeResource {
    @GET
    public String doGetAsPlainText() {
    }

    @GET
    @Produces("text/html")
    public String doGetAsHtml() {
    }
}
```

@PRODUCES

- Se uma classe de recurso for capaz de produzir mais de um tipo de *Media Type*, o método de recurso escolhido corresponderá ao tipo de *Media Type* mais aceitável declarado pelo cliente.

```
@GET
@Produces({"application/xml", "application/json"})
public String doGetAsXmlOrJson() {
    ...
}
```

@PRODUCES

- Opcionalmente, o servidor também pode especificar o fator de qualidade para tipos de *Media Type* individuais. Estes são considerados se vários são igualmente aceitáveis pelo cliente.

```
@GET
@Produces({"application/xml; qs=0.9", "application/json"})
public String doGetAsXmlOrJson() {
    ...
}
```

@CONSUMES

- É usada para especificar os tipos de *Media Type* de representações que podem ser consumidas por um recurso.

```
@POST
@Consumes("text/plain")
public void postClichedMessage(String message) {
    // Store the message
}
```

- @Consumes podem ser aplicados na classe e nos níveis do método e mais de um tipo de *Media Type* pode ser declarado na mesma declaração @Consumes.

Injeções - @PathParam

- Extrai um parâmetro da URI de solicitação que corresponde ao caminho declarado em @Path.

```
@Path("/customers")
public class CustomerResource {
   @Path("{id}")
   @GET
   @Produces("application/xml")
   public StreamingOutput getCustomer(@PathParam("id") int id) {
         ...
   }
}
```

- É possível fazer referência a mais de um parâmetro de caminho URI em seus métodos Java.

Injeções - @QueryParam

- Permite injetar parâmetros de consulta URI individuais em seus parâmetros Java.

```
@Path("/customers")
public class CustomerResource {
        @GET
        @Produces("application/xml")
        public String getCustomers(@QueryParam("start") int start, @QueryParam("size") int
        size) {
        }
}
```

Injeções - @MatrixParam

- São um conjunto arbitrário de pares nome-valor incorporados em um segmento do caminho URI.
- Representa recursos que são endereçáveis pelos seus atributos, bem como pelo seu identificador bruto.

```
@Path("/{make}")
public class CarResource {
    @GET
    @Path("/{model}/{year}")
    @Produces("image/jpeg")
    public Jpeg getPicture(@PathParam("make") String make, @PathParam("model") String
@MatrixParam("color") String color) {
    ...
}
```

Injeções - @HeaderParam

- Usada para injetar valores de cabeçalho de solicitação HTTP.

```
@Path("/myservice")
public class MyService {
        @GET
        @Produces("text/html")
        public String get(@HeaderParam("Referer") String referer) {
        ...
        }
}
```

Injeções - @FormParam

- Usada para acessar os *bodies* da solicitação application / x-www-form-urlencoded.

```
<FORM action="http://example.com/customers" method="post">
First name: <INPUT type="text" name="firstname"><BR>
Last name: <INPUT type="text" name="lastname"><BR>
<INPUT type="submit" value="Send">
</FORM>

@Path("/customers")
        public class CustomerResource {
        @POST
        public void createCustomer(@FormParam("firstname") String first,
        @FormParam("lastname") String last) {
        ...
        }
}
```

Injeções - @CookieParam

- Permite injetar *cookies* enviados por um pedido de cliente para seus métodos de recurso JAX-RS.

```
@Path("/myservice")
public class MyService {
        @GET
        @Produces("text/html")
        public String get(@CookieParam("customerId") int custId) {
        ...
        }
}
```

Injeções - @BeanParam

- Adicionado na especificação JAX-RS 2.0.
- Permite injetar uma classe específica da aplicação cujos métodos ou campos são anotados com qualquer um dos parâmetros de injeção.

```
@POST
public void post(@BeanParam MyBeanParam beanParam, String entity) {
    final String pathParam = beanParam.getPathParam(); // contains injected path parameter "p"
    ...
}
```

Jersey

- **Prática 1 - Criando projeto Jersey com Gradle**

https://github.com/eltonbarbosa/java/tree/base-exemplos

Jersey

- **Prática 2 – Melhorando o exemplo HelloWord**

Autenticação

- Jersey aceita Autenticação HTTP básica e Digest:

- BASIC: As informações de autenticação são enviadas sempre com cada solicitação HTTP. É mais usual. Deve ser combinado com o uso de **Secure Socket Layer** (SSL)/(Transport Layer Security)TLS, pois a senha é enviada somente codificado em BASE64.

Autenticação

- BASIC NON-PREEMPTIVE: As informações de autenticação são adicionadas somente quando o servidor recusa a solicitação com o código de *status* 401 e, em seguida, a solicitação é repetida com informações de autenticação. Tem um impacto negativo no desempenho. Não envia credenciais quando não são necessárias. Deve ser combinado com o uso de SSL/TLS, pois a senha é enviada somente codificado em BASE64.

Autenticação

- DIGEST: Http digest authentication. Não requer o uso de SSL/TLS.

- UNIVERSAL: Combinação de autenticação básica e digest

Autenticação - OAuth

- Define o modelo de autenticação segura em nome de outro usuário.
- Existem duas versões do OAuth no momento: OAuth 1 e OAuth2.
- OAuth 2 é a versão mais recente e não é compatível com a especificação OAuth 1.

Autenticação - OAuth

- OAuth em geral é amplamente utilizado em sites populares, a fim de conceder acesso a uma conta de usuário e recursos associados para um consumidor de terceiros (aplicativo).

- O consumidor então geralmente usa *Web Services RESTful* para acessar os dados do usuário.

Autenticação - OAuth

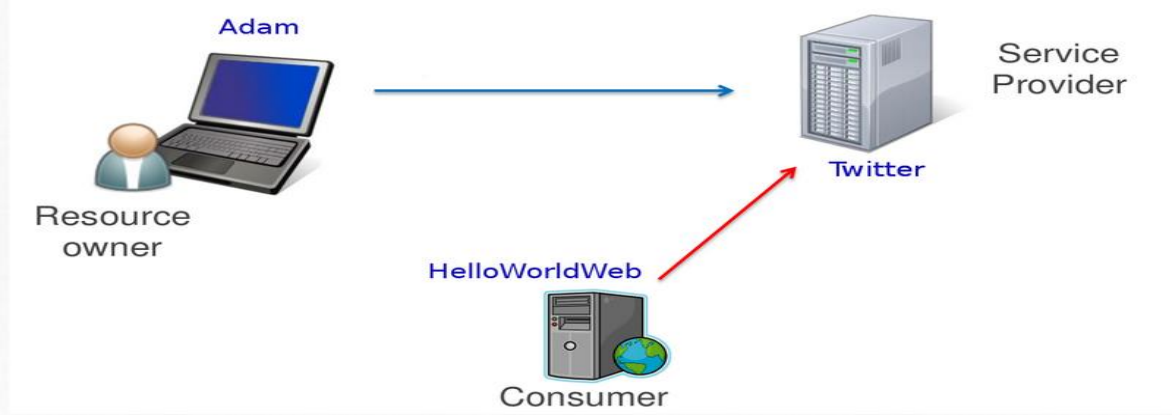


Figura 1 - Suporte OAuth

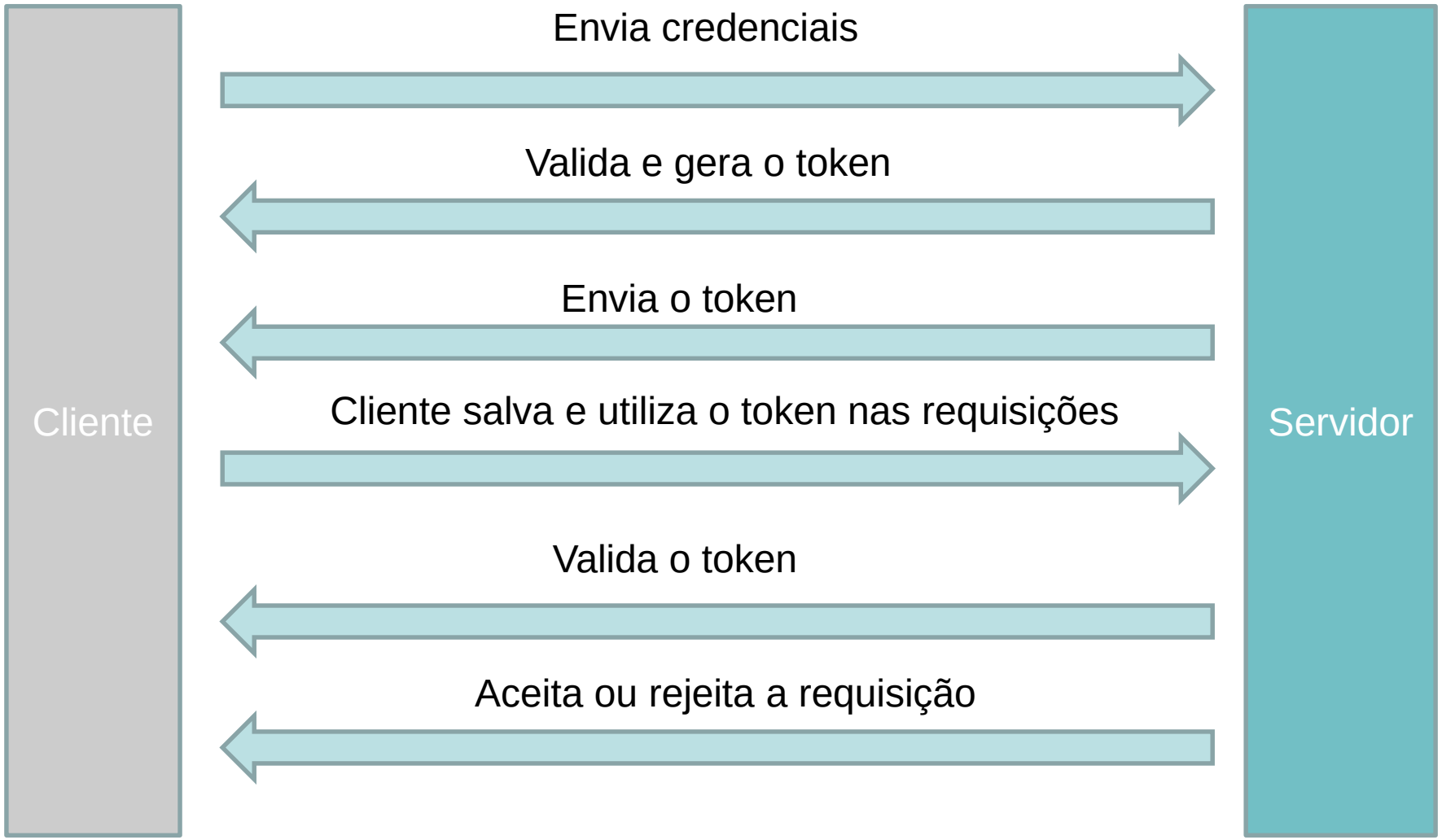
Fonte: Jersey < https://jersey.github.io>

Autenticação - OAuth

- OAuth Jersey atualmente é suportado para os seguintes casos de uso e versões de OAuth:
  - OAuth 1: Cliente (consumidor) e servidor (provedor de serviços);
  - Oauth 2: Cliente (consumidor).

- Com o suporte ao cliente e ao servidor, existem dois cenários suportados:
  - Fluxo de autorização;
  - Autenticação com *token* de acesso (suporte para solicitações autenticadas).

Autenticação baseada em token



Jersey

- **Prática 3 - Adicionando autenticação *basica***

Jersey

- **Prática 4 – Desenvolvimento de serviços REST “Contatos”**

Jersey

- **Prática 5 – Autenticação de API REST baseada em token**

Jersey

- **Prática 6 – Exercícios**

Referências

- Burke, Bill. **Restful Java with JAX-RS 2.0**. 2 ed. O'Reilly, 2007.

- **Jersey RESTful Web Services in Java.** Disponível em < https://jersey.github.io/>. Acesso em 15 Ago. 2017

- Kalali, M.; Metha, B. **Developing RESTful Services with JAX-RS 2.0, WebSockets and JSON**. Birmingham: Packt Publishing, 2016**.**