UNIVERSIDADE DA CORUÑA

Departamento de Ciencias de la Computación y Tecnologías de la Información

Tema 8. Tutorial Avanzado de Jakata RESTFul Web Services (JAX-RS)

Integración de Aplicaciones

Índice

- Aspectos avanzados de JAX-RS (desde v2.0)
  - Soporte para hipermedia (HATEOAS)
  - API de intercepción
- Mediante el ejemplo **`rs-advjaxrstutorial`** se ven en detalle los dos primeros
  - Adicionalmente
    - Tratamiento genérico en la serialización de excepciones
    - Adaptadores para serializar LocalDateTime

Aspectos avanzados de JAX-RS (1)

- Soporte para hipermedia (HATEOAS)
  - Añadida una clase para representar links (Link)
  - Soporte para links transicionales (controles para actuar sobre un recurso)
    - En el servidor es posible añadir links a la hora de construir las respuestas (enviados como cabeceras del mensaje HTTP)
  - Soporte manual para links estructurales (URIs hacia otros recursos)
    - Adaptador JAXB para serializar Links
  - En el cliente es posible “seguir” un Link

Aspectos avanzados de JAX-RS (2)

- API de intercepción
  - Permite tener puntos de extensión estandarizados y bien definidos (tanto en el servidor como en el cliente)
  - Permite añadir aspectos transversales de forma que sean transparentes para el resto del código de la aplicación
    - Autenticación, caching, codificación, etc.

Aspectos avanzados de JAX-RS (3)

- API de intercepción - Filtros
  - Se ejecutan antes o después del procesamiento de una petición o respuesta
    - Antes o después de que se ejecute un método de una clase recurso en el servidor
    - Antes de que se envíe una petición o después de que se reciba la respuesta en el cliente
  - Utilidad, principalmente, para modificar o procesar las cabeceras de las peticiones o respuestas
  - Es posible definir cadenas de filtros y especificar a qué métodos deben aplicarse

Aspectos avanzados de JAX-RS (y 4)

- API de intercepción - Interceptors
  - Relacionados con la serialización y deserialización de los cuerpos de los mensajes HTTP
  - “Envuelven” la ejecución de las instancias correspondientes de **`MessageBodyReader`** y **`MessageBodyWriter`**
  - Pueden utilizarse para generar firmas digitales o para post o pre-procesar el cuerpo del mensaje antes o después de que sea serializado o deserializado (e.g. comprimir/descomprimir)
  - Es posible definir cadenas de Interceptors y especificar a qué métodos deben aplicarse

Ejemplo: **`CatalogService`** (1)

- Estudiaremos el soporte para Hipermedia y la API de intercepción a través de un ejemplo

- El ejemplo se encuentra en el módulo **`rs-advjaxrstutorial`**
  - Tiene los siguientes submódulos
    - **`rs-advjaxrstutorial-service`**
    - **`rs-advjaxrstutorial-client`**
  - No se pretende ilustrar un diseño por capas
    - No se usa una interfaz para exponer la capa modelo
    - No se usa una interfaz para exponer la capa de acceso al servicio
      - El tipo de cliente a utilizar (XML/JSON) se indica a través de una propiedad de configuración (tipo MIME) que lee la clase que accede al servicio

Ejemplo: `CatalogService` (y 2)

- El servicio proporciona operaciones para
  - Encontrar los detalles de un Producto a partir de su identificador
    - Devuelve: Identificador, Nombre, Precio, Descripción, Fecha de creación, Enlace “self”, Enlace “categoría”
  - Encontrar “intervalo de productos” que contengan una cierta palabra clave en su nombre
    - En la cabecera Link devuelve: Enlace “self”, Enlace “next” (opcional), Enlace “previous” (opcional)
    - Por cada producto devuelve: Identificador, Nombre, Precio, Enlace “self”
  - Encontrar los detalles de una Categoría a partir de su identificador
    - Devuelve: Identificador, Nombre, Enlace “self”

Protocolo REST (1)

- Recursos
  - **`/products`** Recurso colección
    - **`GET`**
      - Lista de productos
      - Parámetros
        - » **`keyword`** permite filtrar los productos por palabra clave contenida en el nombre (valor por defecto "" → devuelve todos los productos)
        - » **`startIndex`** y **`count`** permiten especificar el intervalo a recuperar de entre todos los encontrados (valores por defecto 0 y 2)

`/products?keyword=ProductName&startIndex=0&count=5`

Protocolo REST (2)

- Recursos
  - **`/products`** Recurso colección
    - **GET**
      - La información de los productos se representa, en XML, en el siguiente formato

```
<products xmlns="http://ws.udc.es/catalog/xml"
          xmlns:ns2="http://www.w3.org/2005/Atom" >
   <product>
      <product-id>1</product-id>
      <name>Product 1</name>
      <price>10.0</price>
      <n2:link href="http://XXX/products/1" rel="self" title="Self link"
         type="application/xml"/>
   </product>
   <product>
      <product-id>2</product-id>
      <name>Product 2</name>
      <price>20.0</price>
      <n2:link href="http://XXX/products/2" rel="self" title="Self link"
         type="application/xml"/>
   </product>
</products>
```

Protocolo REST (3)

- Listar productos por palabra clave

Petición GET a http://XXX/products?keyword=Product&startIndex=2&count=2
Cabecera Accept: application/xml


```
HTTP/1.1 200 OK
Link: <http://XXX/products/?keyword=prod&startIndex=2&count=2>;
rel="self"; title="Current interval of products";
type="application/xml",
<http://XXX/products/?keyword=prod&startIndex=4&count=2>;
rel="next"; title="Next interval of products";
type="application/xml",
<http://XXX/products/?keyword=prod&startIndex=0&count=2>;
rel="previous"; title="Previous interval of products";
type="application/xml"
...

<?xml version="1.0" encoding="UTF-8"?>
<products xmlns="http://ws.udc.es/products/xml"
          xmlns:ns2="http://www.w3.org/2005/Atom" >
  <product> ... </product>
  <product> ... </product>
</products>
```

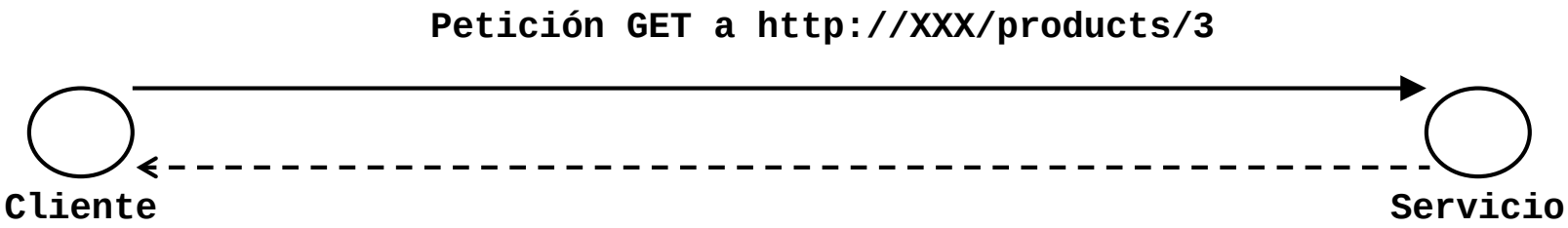
Protocolo REST (4)

- Recursos
  - **`/products/{id}`** Recurso individual por producto
    - **GET**
      - Obtiene la información detallada del producto
      - La información detallada de cada producto se representa, en XML, en el siguiente formato

```
<product-details xmlns="http://ws.udc.es/catalog/xml"
                 xmlns:ns2="http://www.w3.org/2005/Atom" >
   <product-id>3</product-id>
   <name>Product 3</name>
   <price>30.0</price>
   <description>Description of Product 3</description>
   <creationDate>2020-06-28T11:39:41.173696400</creationDate>
   <n2:link href="http://XXX/categories/3" rel="category"
      title="Product category" type="application/xml"/>
   <n2:link href="http://XXX/products/3" rel="self"
      title="Self link" type="application/xml"/>
</product-details>
```

Protocolo REST (5)

- Obtener información de un producto




```
HTTP/1.1 200 OK
…

<?xml version="1.0" encoding="UTF-8"?>
<product-details xmlns="http://ws.udc.es/catalog/xml"
                 xmlns:ns2="http://www.w3.org/2005/Atom" >
   <product-id>3</product-id>
   <name>Product 3</name>
   <price>30.0</price>
   <description>Description of Product 3</description>
   <creationDate>2020-06-28T11:39:41.173696400</creationDate>
   <n2:link href="http://XXX/categories/3" rel="category"
      title="Product category" type="application/xml"/>
   <n2:link href="http://XXX/products/3" rel="self"
      title="Self link" type="application/xml"/>
</product-details>
```

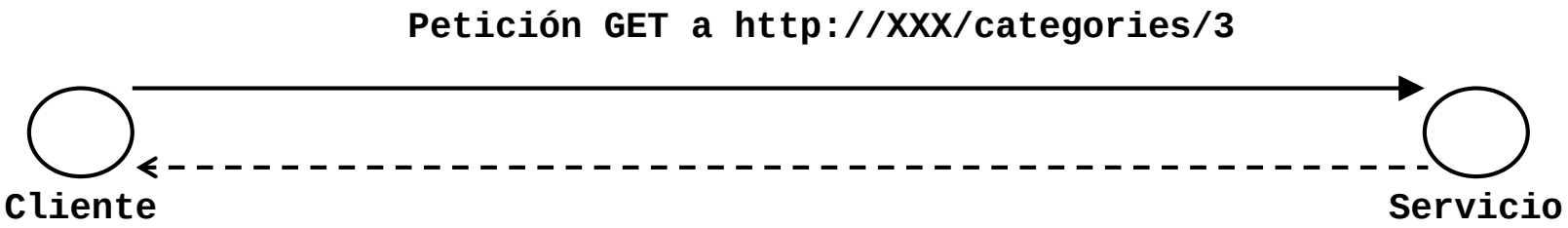
Protocolo REST (6)

- Recursos
  - **`/categories/{id}`** Recurso individual por categoría
    - **GET**
      - Obtiene la información detallada de la categoría
      - La información de cada categoría se representa en XML en el siguiente formato

```
<category xmlns="http://ws.udc.es/catalog/xml"
          xmlns:ns2="http://www.w3.org/2005/Atom" >
   <category-id>3</category-id>
   <name>Category 3</name>
   <n2:link href="http://XXX/categories/3" rel="self"
      title="Self link" type="application/xml"/>
</category>
```

Protocolo REST (7)

- Obtener información de una categoría




```
HTTP/1.1 200 OK
…

<?xml version="1.0" encoding="UTF-8"?>
<category xmlns="http://ws.udc.es/catalog/xml"
          xmlns:ns2="http://www.w3.org/2005/Atom" >
   <category-id>3</category-id>
   <name>Category 3</name>
   <n2:link href="http://XXX/categories/3" rel="self"
      title="Self link" type="application/xml"/>
</category>
```

Protocolo REST (8)

- Para los errores generados por la lógica de la aplicación
  - Se utiliza el código HTTP más próximo a la semántica de la respuesta
    - Recurso no existe: **`404 Not Found`**
      - Similar a **`InstanceNotFoundException`**
  - El cuerpo del mensaje lleva información adicional
    - Representación en XML/JSON de los datos de la excepción
    - Ilustra cómo serializar excepciones de forma genérica de modo que se pueda tratar de forma sencilla el caso de varias excepciones asociadas al mismo código de respuesta de error HTTP

Protocolo REST (y 9)

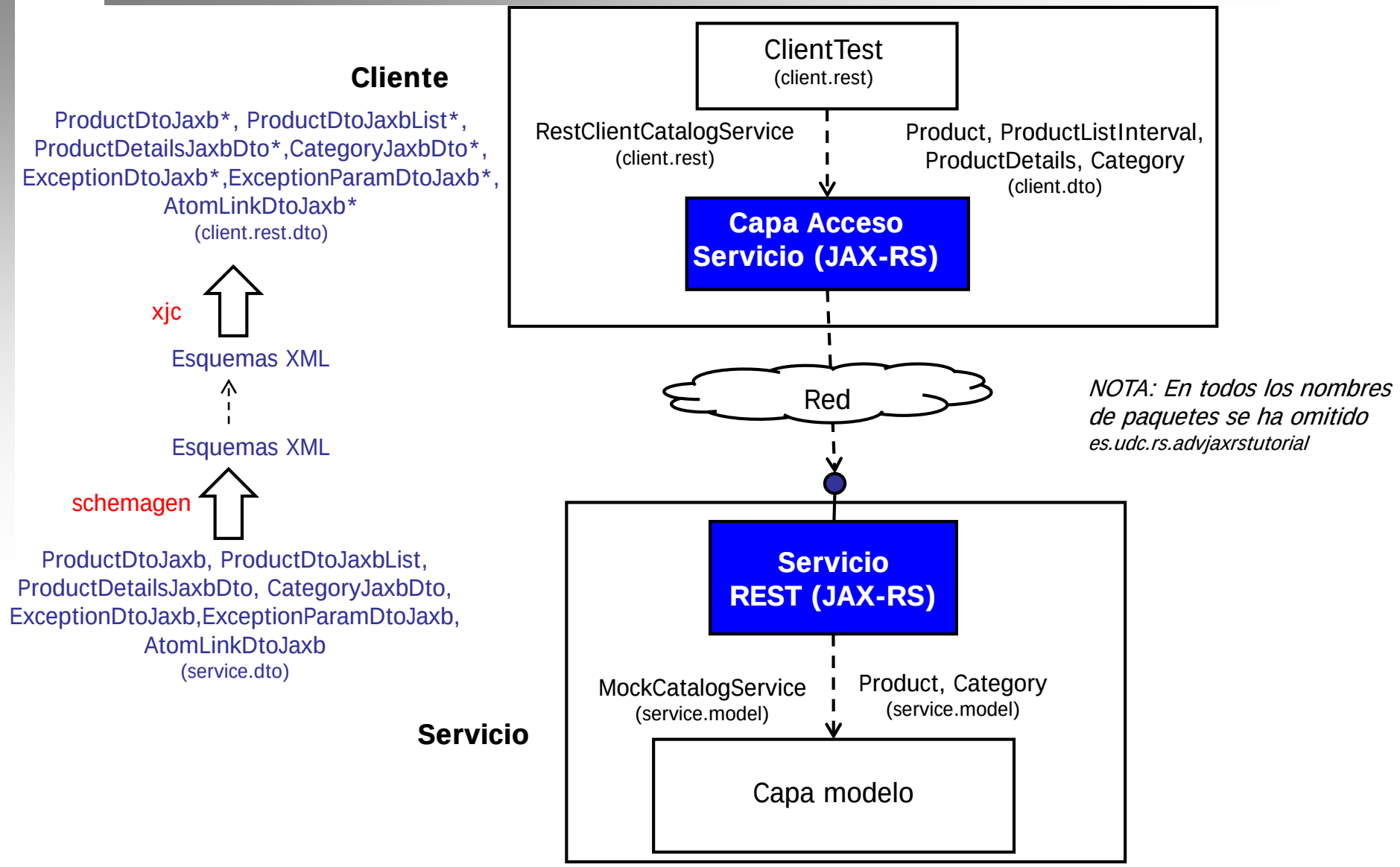
- Ejemplo de formato genérico para errores
  - XML

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<error xmlns="http://ws.udc.es/catalog/xml" errorType="InstanceNotFound">
    <param key="instanceId" value="400"/>
    <param key="instanceType" value="Product"/>
</error>
```

  - JSON

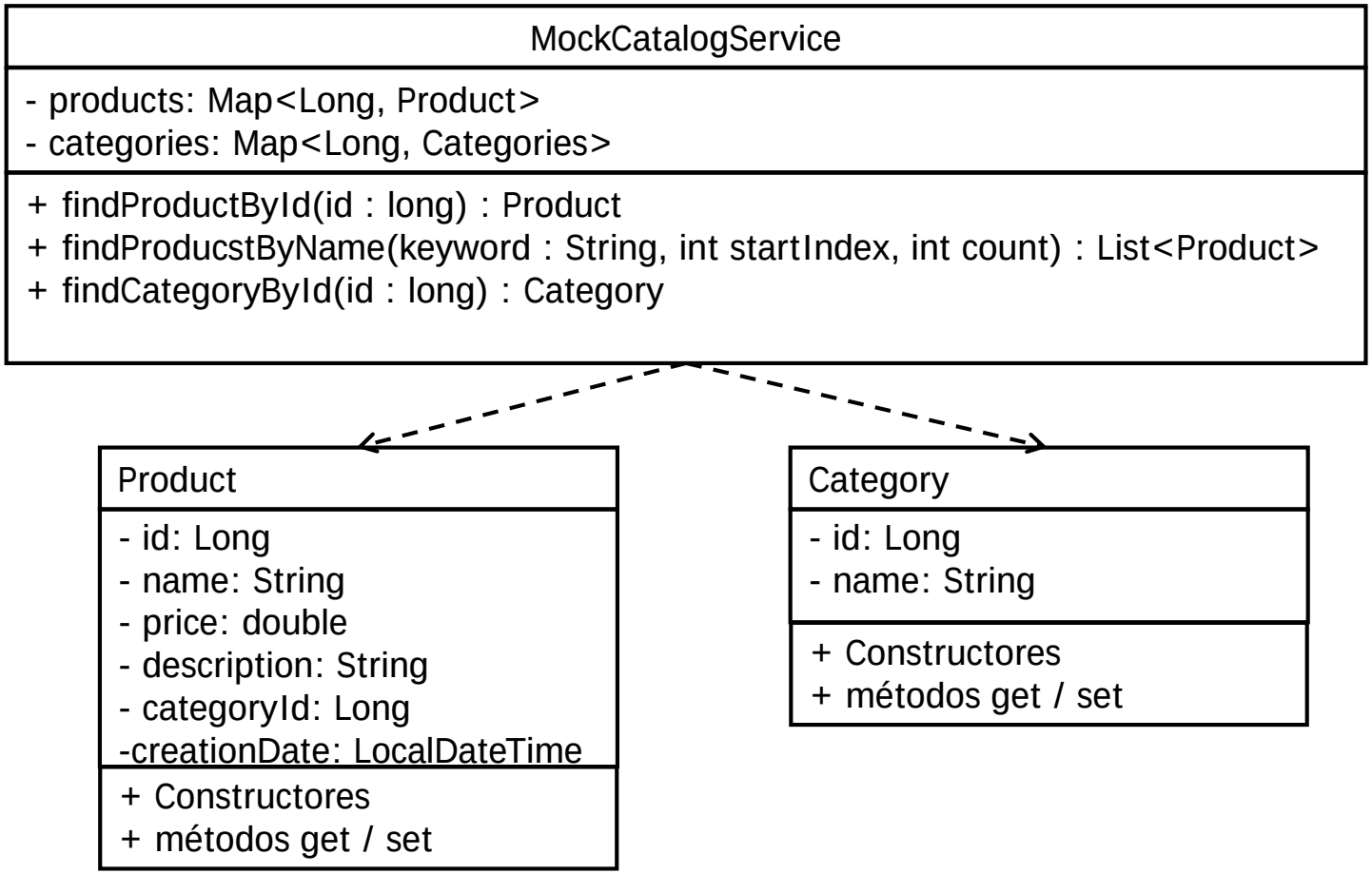
```json
{
    "errorType": "InstanceNotFound",
    "params": [
        {
            "key": "instanceId",
            "value": "400"
        },
        {
            "key": "instanceType",
            "value": "Product"
        }
    ]
}
```

Arquitectura del Tutorial Avanzado de JAX-RS



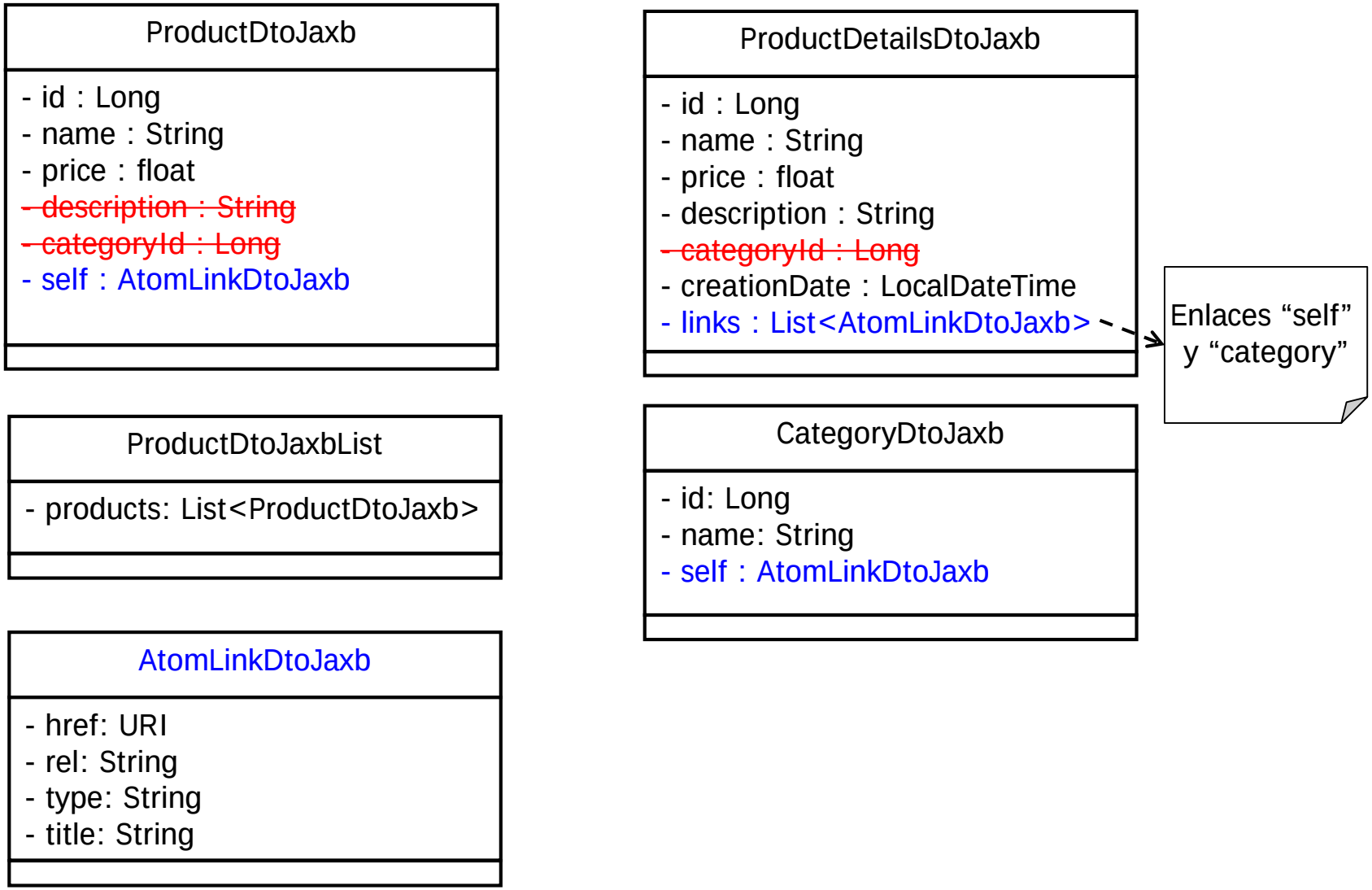
Capa Modelo

- La lógica de negocio se simula a través de la clase **`MockCatalogService`**, que es invocada directamente desde la implementación del servicio



Clases de la Capa del Servicio REST (JAX-RS)

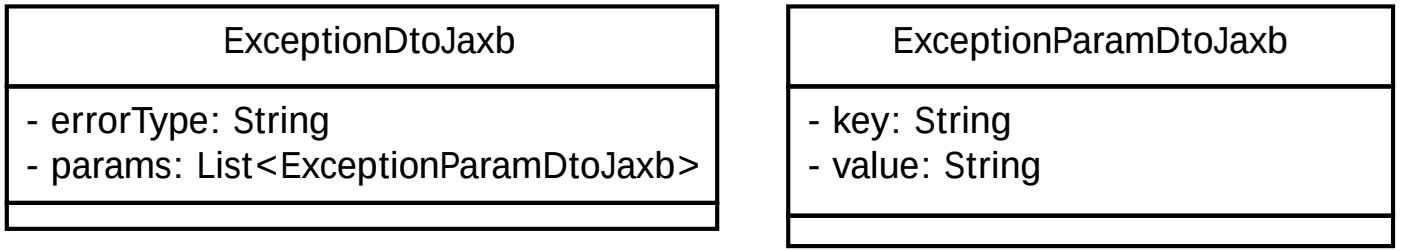
- Se utiliza JAXB para serializarlos a XML



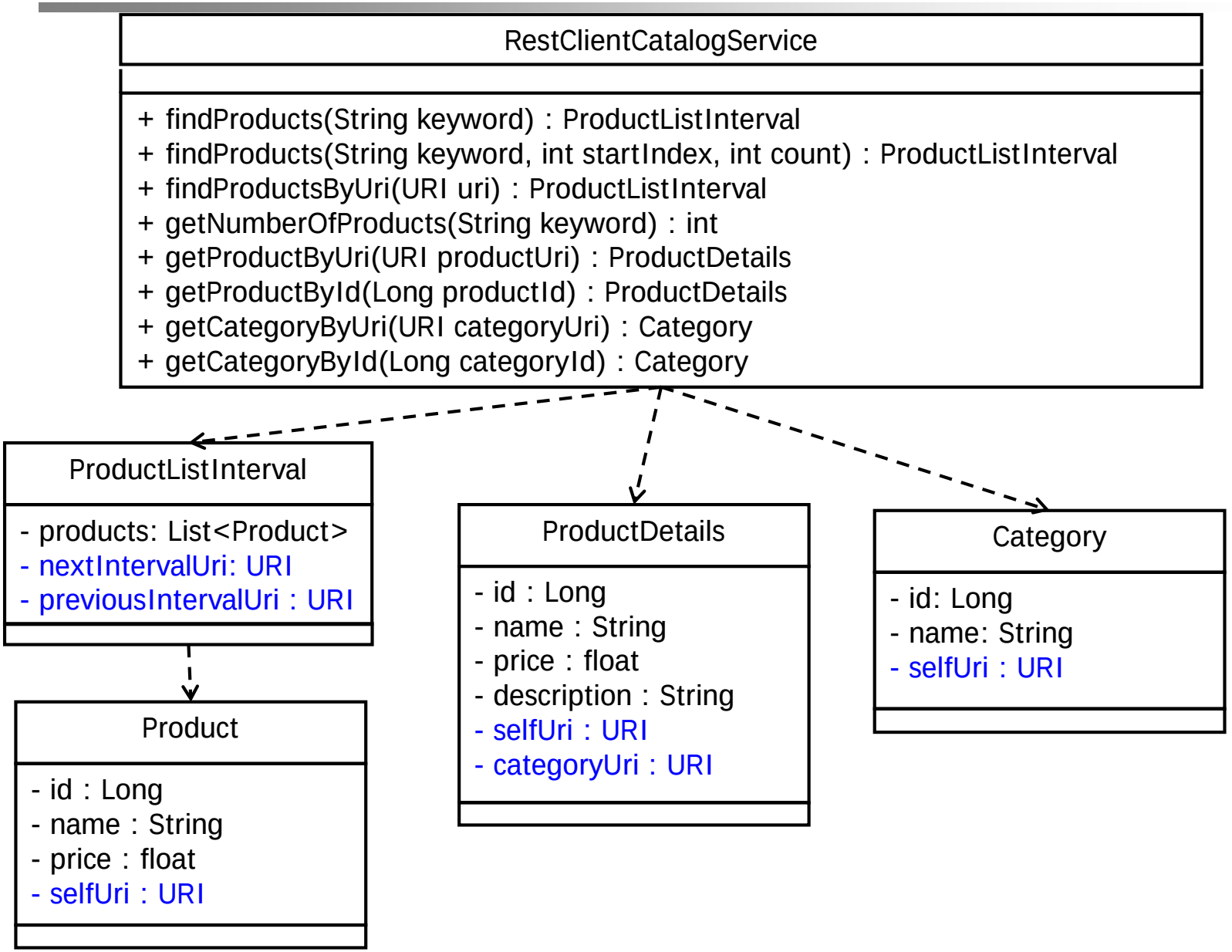
Clases de la Capa de Acceso REST (JAX-RS)

- Se utilizará JAXB para deserializarlos desde XML
  - Generados por el compilador de esquemas, clases equivalentes a las definidas en el servicio

- Adicionalmente, para encapsular los errores (en ambas capas)



Capa de Acceso



Comentarios

- En las clases de la capa del Servicio se utiliza la clase **`Link`** proporcionada por JAX-RS 2.0 para representar enlaces transicionales en cabeceras
  - No se ha utilizado la clase **`Link`** de JAX-RS para los enlaces estructurales por varios motivos
    - En las clases JAXB del servicio es necesario utilizar un adaptador (no directamente tratados por JAXB)
    - En los objetos de la capa de Acceso al Servicio genera una clase especial, **`JaxbLink`**, poco natural
      - Resultantes de ejecutar **`schemagen`** + xjc
    - Incompatibilidades con la mayoría de plugins de JSON (e.g. con Jackson funciona en el servicio pero no en el cliente)
- En las clases ofrecidas por la capa de Acceso al Servicio en el cliente se utilizan URIs para representar enlaces
  - Independencia de la tecnología

MockCatalogService.java (1)

```
public class MockCatalogService {

   private static Map<Long, Product> products =
      new LinkedHashMap<Long, Product>();
   private static Map<Long, Category> categories =
      new HashMap<Long, Category>();
   ...

   static {
      Category category1 = addCategory(
         new Category(null, "Category 1"));
      Category category2 = addCategory(
         new Category(null, "Category 2"));
      ...
      addProduct(new Product(null, "Product 1", 10,
         "Description of Product 1",category1.getId()));
      addProduct(new Product(null, "Product 2", 20,
         "Description of Product 2", category2.getId()));
      ...
   }
```

MockCatalogService.java (y 2)

```
    ...

    private static Category addCategory(Category c) { ... }

    private static Product addProduct(Product p) { ... }

    public static Product findProductById(long id)
        throws InstanceNotFoundException { ... }

    public static List<Product> findProducstByName(String keyword,
        int startIndex, int count) { ... }

    public static Category findCategoryById(long id)
        throws InstanceNotFoundException { ... }

}
```

Comentarios

- Errores
  - Se reutiliza la siguiente excepción definida en **`ws-util`**
    - **`es.udc.ws.util.exceptions.InstanceNotFoundException`**: indica que se intenta hacer algo sobre un objeto que no existe
      - Buscar producto por identificador, buscar categoría por identificador

ProductDtoJaxb.java

```
package es.udc.rs.advjaxrstutorial.service.dto;
...
@XmlRootElement(name = "product")
@XmlType(name = "productType", propOrder = { "id", "name", "price",
     "self" })
public class ProductDtoJaxb {
   @XmlElement(name = "product-id", required = true)
   private Long id;
   @XmlElement(required = true)
   private String name;
   @XmlElement(required = true)
   private double price;
   @XmlElement(name = "link", namespace = "http://www.w3.org/2005/Atom")
   private AtomLinkDtoJaxb self;

   ...
}
```

ProductDetailsDtoJaxb.java

```
package es.udc.rs.advjaxrstutorial.service.dto;
...
@XmlRootElement(name = "product-details")
@XmlType(name = "productDetailsType", propOrder = { "id", "name", "price",
         "description", "creationDate", "links" })
public class ProductDetailsDtoJaxb {
   @XmlElement(name = "product-id", required = true)
   private Long id;
   @XmlElement(required = true)
   private String name;
   @XmlElement(required = true)
   private double price;
   @XmlElement(required = true)
   private String description;
   @XmlElement(required = true)
   @XmlSchemaType(name = "dateTime")
   private LocalDateTime creationDate;

   @XmlElement(name = "link", namespace = "http://www.w3.org/2005/Atom")
   private List<AtomLinkDtoJaxb> links;
   ...
}
```

AtomLinkDtoJaxb.java

```
package es.udc.rs.advjaxrstutorial.service.dto;
...
@XmlType(name = "atomLinkType", namespace = "http://www.w3.org/2005/Atom")
public class AtomLinkDtoJaxb {
  @XmlAttribute(name = "href", required = true)
  private URI href;
  @XmlAttribute(name = "rel",  required = true)
  private String rel;
  @XmlAttribute(name = "type", required = true)
  private String type;
  @XmlAttribute(name = "title", required = true)
  private String title;

  ...
}
```

Exception*DtoJaxb.java

```
package es.udc.rs.advjaxrstutorial.service.dto;
...
@XmlRootElement(name = "error")
@XmlType(name = "errorType")
public class ExceptionDtoJaxb {
   @XmlAttribute(required = true)
   private String errorType;
   @XmlElement(name = "param", required = true)
   private List<ExceptionParamDtoJaxb> params;
...
}


@XmlType(name = "errorParamType")
public class ExceptionParamDtoJaxb {
    @XmlAttribute(required = true)
    private String key;
    @XmlAttribute(required = true)
    private String value;
...
}
```

Soporte para Hipermedia – Servidor (1)

- La clase abstracta **`Link`** permite representar los metadatos incluidos en un link de cabecera o un link Atom
  - **`URI/href`**, **`rel`**, **`title`**, **`type`**
  - También permite añadir parámetros adicionales (para extensiones propietarias)
- Las instancias de la clase **`Link`** se construyen a través de la clase **`Link.Builder`**
  - Tiene métodos para obtener un **`Link`** a partir de una **`URI`**, un **`String`**, un **`UriBuilder`**, otro **`Link`**, etc.)
  - Tiene métodos para establecer las propiedades del link. E.g.
    - **`rel(String rel) : Link.Builder`**
    - **`title(String title) : Link.Builder`**
    - **`type(String type) : Link.Builder`**
    - **`param(String name, String value) : Link.Builder`**
  - Tiene un método para construir el **`Link`**
    - **`build(Object ... values) : Link`**

Soporte para Hipermedia – Servidor (y 2)

- Para añadir uno o más links a las cabeceras de la respuesta se puede llamar al método de la clase **`Response.ResponseBuilder`**
  - **`links(Link... links): Response.ResponseBuilder`**

- Para enlaces estructurales utilizamos nuestra clase **`AtomLinkDtoJaxb`**
  - La alternativa para devolver el contenido de un **`Link`** embebido en el XML de respuesta, cuando se usa JAXB, sería utilizar un adaptador XML de JAXB (**`XmlAdapter`**)
    - JAXB no sabe como serializar la clase **`Link`** a XML
    - Un adaptador XML convierte una instancia de una clase a una instancia de otra que JAXB sí sabe cómo serializar (i.e. esa otra clase estará anotada con las anotaciones de JAXB)
    - Es necesario anotar la propiedad de tipo **`Link`** o **`List<Link>`** con el adaptador **`Link.JaxbAdapter.class`**

Soporte para tipo **`LocalDateTime (1)`**

- Para tratar con el tipo de dato Java LocalDateTime es necesario crear un adaptador específico
  - Tienen que extender la clase **`XmlAdapter<ValueType,BoundType>`**
    - **`abstract marshal(BoundType v): ValueType`**
    - **`abstract unmarshal(ValueType v): BoundType`**
  - Se ha creado la siguiente clase adaptadora, tanto en el **cliente** como en el **servicio**

`es.udc.rs.advjaxrstutorial.*.jaxb.LocalDateTimeXMLAdapter`

  - En el **servicio** se ha registrado la clase adaptadora a nivel de paquete en el fichero **`package-info.java`** (se podría haber asociado a nivel de atributo de clase para un alcance más local)

```
@XmlJavaTypeAdapters({
        @XmlJavaTypeAdapter(type= LocalDateTime.class,
                            value= LocalDateTimeXmlAdapter.class)
})
```

Soporte para tipo **LocalDateTime (y 2)**

- En el **cliente**
  - se ha añadido configuración específica en el fichero de bindings

```
<jxb:globalBindings>
    <xjc:javaType name="java.time.LocalDateTime" xmlType="xs:dateTime"
        adapter="es.udc.rs.advjaxrstutorial.client.rest.jaxb.LocalDateTimeXmlAdapter" />
</jxb:globalBindings>
```

  - y se ha registrado un nuevo módulo (**JavaTimeModule**) en la clase **JaxbJsonContextResolver**

```
this.mapper.registerModule(new JavaTimeModule());
```

- Adicionalmente, necesario añadir una nueva dependencia en el fichero pom.xml
  - Añadida en **rs-advjaxrstutorial**, compartida entre **cliente** y **servicio**

```
<dependency>
    <groupId>com.fasterxml.jackson.datatype</groupId>
    <artifactId>jackson-datatype-jsr310</artifactId>
    <version>${jackson.version}</version>
</dependency>
```

ProductResource.java (1)

```
package es.udc.rs.advjaxrstutorial.service.resources;
...

@Path("products")
public class ProductResource {

   @GET
   @Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
   public Response findProducts(
         @DefaultValue("") @QueryParam("keyword") String keyword,
         @DefaultValue("0") @QueryParam("startIndex") int startIndex,
         @DefaultValue("2") @QueryParam("count") int count,
         @Context UriInfo uriInfo, @Context HttpHeaders headers) {
      List<Product> products = MockCatalogService.
         findProducstByName(keyword, startIndex, count);
      String type = ServiceUtil.getTypeAsStringFromHeaders(headers);

      List<ProductDtoJaxb> productDtos = ProductToProductDtoJaxbConversor.
         toProductDtoJaxb(products, uriInfo.getBaseUri(), type);
```

ProductResource.java (2)

```
    Link selfLink = getSelfLink(uriInfo, keyword, startIndex, count,
       type);
    Link nextLink = getNextLink(uriInfo, keyword, startIndex, count,
       products.size(), type);
    Link previousLink = getPreviousLink(uriInfo, keyword, startIndex,
       count, type);
    ResponseBuilder response = Response.ok(
      new ProductDtoJaxbList(productDtos)).links(selfLink);
    if (nextLink != null) {
       response.links(nextLink);
    }
    if (previousLink != null) {
       response.links(previousLink);
    }
    return response.build();
}
```

ProductResource.java (3)

```
private static Link getNextLink(UriInfo uriInfo, String keyword,
      int startIndex, int count, int numberOfProducts, String type) {
   if (numberOfProducts < count) {
      return null;
   }
   return ServiceUtil.getProductsIntervalLink(uriInfo, keyword,
      startIndex + count, count, "next", "Next interval of products",
      type);
}

private Link getPreviousLink(UriInfo uriInfo, String keyword,
      int startIndex, int count, String type) {
   if (startIndex <= 0) {
      return null;
   }
   startIndex = startIndex - count;
   if (startIndex < 0) {
      startIndex = 0;
   }
   return ServiceUtil.getProductsIntervalLink(uriInfo, keyword,
      startIndex, count, "previous", "Previous interval of products“,
      type);
}
```

ProductResource.java (y 4)

```
    private Link getSelfLink(UriInfo uriInfo, String keyword,
          int startIndex, int count, String type) {
       return ServiceUtil.getProductsIntervalLink(uriInfo, keyword,
          startIndex, count, "self", "Current interval of products",
          type);
    }

    @GET
    @Path("/{id}")
    @Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
    public ProductDetailsDtoJaxb findById(@PathParam("id") long id,
          @Context UriInfo uriInfo, @Context HttpHeaders headers)
          throws InstanceNotFoundException {
       return ProductToProductDtoJaxbConversor.toProductDetailsDtoJaxb(
          MockCatalogService.findProductById(id), uriInfo.getBaseUri(),
          ServiceUtil.getTypeAsStringFromHeaders(headers));
    }

}
```

ServiceUtil.java (1)

```
package es.udc.rs.advjaxrstutorial.service.util;
...
public class ServiceUtil {
   ...
   public static String getTypeAsStringFromHeaders(HttpHeaders headers) {
      ... }

   ...

   public static AtomLinkDtoJaxb getLinkFromUri(URI baseUri,
         Class<?> resourceClass, Object instanceId, String rel,
         String title, String type) {
       Link.Builder linkBuilder = Link.fromPath(baseUri.toString()
          + UriBuilder.fromResource(resourceClass).build().toString() + "/“
           + instanceId)
        .rel(rel).title(title);
       if (type != null) { linkBuilder.type(type); }
       Link link = linkBuilder.build();
       return new AtomLinkDtoJaxb(
        link.getUri(), link.getRel(), link.getType(), link.getTitle());
   }
```

ServiceUtil.java (y 2)

```
    public static Link getProductsIntervalLink(UriInfo uriInfo,
          String keyword,  int startIndex, int count, String rel,
          String title, String type) {
       UriBuilder uriBuilder = uriInfo.getAbsolutePathBuilder()
          .queryParam("keyword", keyword)
          .queryParam("startIndex", startIndex)
          .queryParam("count", count);
       Link.Builder linkBuilder =
          Link.fromUriBuilder(uriBuilder).rel(rel).title(title);
       if (type!=null) {
          linkBuilder.type(type);
       }
       return linkBuilder.build();
    }

}
```

ProductToProductDtoJaxbConversor.java (1)

```
package es.udc.rs.advjaxrstutorial.service.util;

public class ProductToProductDtoJaxbConversor {

   public static List<ProductDtoJaxb> toProductDtoJaxb(List<Product>
         products, URI baseUri, String type) {
     List<ProductDtoJaxb> productDtos = new ArrayList<>(products.size());
     for (Product product : products) {
        productDtos.add(toProductDtoJaxb(product, baseUri, type));
     }
     return productDtos;
   }

   public static ProductDtoJaxb toProductDtoJaxb(Product product, URI
         baseUri, String type) {
      Link selfLink = ServiceUtil.getLinkFromUri(baseUri,
         ProductResource.class, product.getId(), "self", "Self link", type);
      return new ProductDtoJaxb(product.getId(), product.getName(),
         product.getPrice(), selfLink);
   }
```

ProductToProductDtoJaxbConversor.java (y 2)

```
    public static ProductDetailsDtoJaxb toProductDetailsDtoJaxb(
          Product product, URI baseUri, String type) {
       AtomLinkDtoJaxb categoryLink = ServiceUtil.getLinkFromUri(baseUri,
          CategoryResource.class, product.getCategoryId(), "category",
          "Product category", type);
       AtomLinkDtoJaxb selfLink = ServiceUtil.getLinkFromUri(baseUri,
          ProductResource.class, product.getId(), "self", "Self link", type);
       List<AtomLinkDtoJaxb> links = new ArrayList<Link>();
       links.add(categoryLink);
       links.add(selfLink);
       return new ProductDetailsDtoJaxb(product.getId(), product.getName(),
          product.getPrice(), product.getDescription(),
          product.getCreationDate(), links);
    }

}
```

Comentarios (1)

- El método **`getTypeAsStringFromHeaders`** de la clase **`ServiceUtil`** es un método utilidad para obtener el tipo de contenido preferido por el cliente de entre todos los soportados por el servidor (XML o JSON)
  - Será el tipo utilizado para el atributo **`type`** de los enlaces creados

Comentarios (2)

- Los métodos **`getNextLink`**, **`getPreviousLink`** y **`getSelfLink`** de **`ProductResource`** obtienen un **`Link`** para acceder al siguiente, anterior y al propio intervalo respectivamente
  - Usan el método utilidad **`ServiceUtil.getProductsIntervalLink`**
    - Recibe
      - Información de la URI de la petición recibida (**`UriInfo`**)
      - El valor de los parámetros **`keyword`**, **`startIndex`** y **`count`**
      - El valor de los parámetros del link a crear: **`rel`**, **`title`** y **`type`**
    - Construye un **`UriBuilder`** a partir del path absoluto de la URI de la petición recibida y el valor de los parámetros
      - Representa a la URI del enlace
    - A partir del **`UriBuilder`** se construye un **`Link.Builder`**
      - Se le establecen los valores de los parámetros **`rel`**, **`title`** y **`type`**
      - Se construye el **`Link`**
- Los **`Link`** no nulos (**`next`** y **`previous`** pueden serlo) se añaden a las cabeceras de la respuesta

Comentarios (3)

- El método **`toProductDtoJaxb`** del conversor calcula el Link self correspondiente al producto

Comentarios (y 4)

- El método **`toProductDetailsDtoJaxb`** del conversor calcula, además, el **`AtomLinkDtoJaxb category`** (enlace a la categoría a la que pertenece el producto)
  - Usan el método utilidad **`ServiceUtil.getLinkFromUri`**
    - Recibe
      - La URI base de la petición recibida
      - La clase del recurso hacia el que se va a crear el link
      - El identificador de la instancia hacia la que se va a crear el link
      - El valor de los parámetros del link a crear: **`rel`**, **`title`** y **`type`**
    - Construye un **`Link.Builder`** generando la URI a partir de la URI base, el path del recurso y el identificador de la instancia
      - Para obtener el path del recurso se usa **`UriBuilder.fromResource`**
        - » Devuelve, por ejemplo, "**`/products`**" y se convierte a "**`products/`**"
    - Al **`Link.Builder`** se le establecen los valores de los parámetros **`rel`**, **`title`** y **`type`** (si no es nulo) y se construye el **`Link`** (método **`build`**)
    - A continuación se crea el objeto de la clase **`AtomLinkDtoJaxb`**

Soporte para Hipermedia – Cliente (1)

- Links de cabecera
  - La clase **`Response`** tiene métodos para obtener los **`Link`** recibidos en las cabeceras. E.g. a partir del valor del **`rel`** del **`Link`**
    - **`getLink(String rel): Link`**

- Es posible "seguir" un **`Link`**
  - El método **`invocation`** de la clase **`Client`** permite especificar el **`Link`** a seguir y devuelve un **`Builder`**
    - **`invocation(Link link): Builder`**
    - El **`Builder`** devuelto tiene establecido el tipo del **`Link`** como el tipo de contenido aceptado por el cliente (e.g. "application/xml")

Soporte para Hipermedia – Cliente (y 2)

- Links estructurales
  - Es posible utilizar atributos de tipo **`Link`** y anotarlos con el mismo adaptador XML utilizado en la parte servidora
  - Sin embargo, si las clases de la parte cliente las generamos automáticamente con la herramienta **`xjc`** ejecutada sobre los esquemas generados con la herramienta **`schemagen`**, lo que nos generará será una clase equivalente a **`Link.JaxbLink`**
    - No hay manera de especificar en los bindings que el tipo XML compuesto generado para **`Link.JaxbLink`** debe mapearse a **`Link`** utilizando un adaptador XML
    - El elemento **`javaType`** permite especificar como mapear tipos XML a clases Java, pero solamente es aplicable a tipos XML simples
  - Como hemos comentado, hemos decidido utilizar una clase JAXB propia, por lo que en la parte cliente se generará una clase a partir de la que habíamos creado en el servicio

RestClientCatalogService.java (1)

```
package es.udc.rs.advjaxrstutorial.client.rest;
...
public class RestClientCatalogService {
   ...
   private MediaType getMediaType() { ... }

   public ProductListInterval findProducts(String keyword) {
      return findProducts(keyword, 0, 2);
   }


  public ProductListInterval findProducts(String keyword, int startIndex,
    int count) {

    WebTarget wt = getEndpointWebTarget().path("products")
      .queryParam("keyword", keyword)
      .queryParam("startIndex", startIndex)
      .queryParam("count", count);
    return findProducts(wt, this.getMediaType());
}
```

RestClientCatalogService.java (2)

```
public ProductListInterval findProductsByUri(URI uri) {
  WebTarget wt = getClient().target(uri);
  return findProducts(wt, this.getMediaType());
}

private ProductListInterval findProducts(WebTarget wt, MediaType type) {
  Response response = (type != null) ?
    wt.request().accept(type).get() : wt.request().get();
  try {
    validateResponse(Response.Status.OK.getStatusCode(), response);
    ProductDtoJaxbList products = response.readEntity(ProductDtoJaxbList.class);
    return new ProductListInterval(
      ProductToProductDtoJaxbConversor.toProducts(products),
      LinkUtil.getHeaderLinkUri(response, "next"),
      LinkUtil.getHeaderLinkUri(response, "previous"));
  } catch (Exception ex) {
      throw new RuntimeException(ex);
  } finally {
      if (response != null) {
        response.close();
      }
  }
}
```

RestClientCatalogService.java (3)

```
public ProductDetails getProductByUri(URI productUri)
      throws InstanceNotFoundException {
   WebTarget wt = getClient().target(productUri);
   return getProduct(wt, this.getMediaType());
}

public ProductDetails getProductById(Long productId)
      throws InstanceNotFoundException {
   WebTarget wt = getEndpointWebTarget().path("products/{id}")
     .resolveTemplate("id", productId);
   return getProduct(wt, this.getMediaType());
}

private ProductDetails getProduct(WebTarget wt, MediaType type)
      throws InstanceNotFoundException {
   Response response = (type != null) ?
     wt.request().accept(type).get() : wt.request().get();
   try {
     validateResponse(Response.Status.OK.getStatusCode(), response);
     ProductDetailsDtoJaxb product =
       response.readEntity(ProductDetailsDtoJaxb.class);
     return ProductToProductDtoJaxbConversor.toProductDetails(product);
   } catch (InstanceNotFoundException ex) { ... }
```

ProductToProductDtoJaxbConversor.java (1)

```
package es.udc.rs.advjaxrstutorial.client.rest.util;
...
public class ProductToProductDtoJaxbConversor {

   public static Product toProduct(ProductDtoJaxb product) {
      return new Product(product.getId(), product.getName(),
         product.getPrice(), LinkUtil.getLinkUri(product.getSelf()));
   }

   public static List<Product> toProducts(
         ProductDtoJaxbList productListDto) {
     List<ProductDtoJaxb> productDtos = new ArrayList<>(products.size());
     for (Product product : products) {
        productDtos.add(toProductDtoJaxb(product, baseUri, type));
     }
     return productDtos;
   }
```

ProductToProductDtoJaxbConversor.java (y 2)

```
    public static ProductDetails toProductDetails(
          ProductDetailsDtoJaxb product) {
       List<JaxbLink> links = product.getLinks();
       URI catUri = LinkUtil.getLinkUriFromList(links, "category");
       URI selfUri = LinkUtil.getLinkUriFromList(links, "self");
       return new ProductDetails(product.getId(), product.getName(),
             product.getPrice(), product.getDescription(),
             product.getCreationDate(), selfUri, catUri);
       }
    }

}
```

Comentarios (1)

- El método **`getMediaType`** indica el tipo de cliente a utilizar: XML o JSON
- Existen tres versiones públicas del método **`findProducts`**
  - Indicando únicamente la palabra de búsqueda
    - Devuelve el primer intervalo con dos resultados
  - Indicando la palabra de búsqueda y los datos del intervalo (índice del primer resultado y número de resultados)
  - Indicando la URI del intervalo
    - Una vez obtenido un intervalo de resultados podría accederse a través de la URI a los siguientes/anteriores intervalos
- El método **`getHeaderLinkUri`** de **`LinkUtil`** obtiene la URI a la que apunta un link de cabecera con el **`rel`** indicado (o null si tal link no existe en las cabeceras)

Comentarios (y 2)

- Existen dos métodos **`getProductByXXX`** para obtener los detalles de un producto
  - Uno que permite obtenerlo por su identificador
  - Otro que permite obtenerlo por su URI
    - Una vez hecha una búsqueda de productos, se podría acceder a los detalles de cada uno accediendo a la URI de su enlace **`self`**

RestClientCatalogService.java (y 4)

```
public int getNumberOfProducts(String keyword) {
  int numProducts = 0;
  WebTarget wt = getEndpointWebTarget().path("products")
    .queryParam("keyword", keyword);
  Response response = wt.request()
    .accept(this.getMediaType())
    .get();
  try {
    validateResponse(Response.Status.OK.getStatusCode(), response);
    ProductDtoJaxbList products =
      response.readEntity(ProductDtoJaxbList.class);
    numProducts += products.getProducts().size();
    while (response.getLink("next") != null) {
      Response linkResponse = getClient()
        .invocation(response.getLink("next")).get();
      // Validar respuesta linkResponse ...
      products = linkResponse.readEntity(ProductDtoJaxbList.class);
      numProducts += products.getProducts().size();
      response.close();
      response = linkResponse;
    }
    return numProducts;
  } catch (Exception ex) { ...
```

Comentarios (1)

- El método **`getNumberOfProducts`** ilustra cómo seguir enlaces desde el cliente
  - Realiza una primera búsqueda especificando la palabra clave deseada
    - El tamaño del intervalo obtenido será el especificado por defecto en el servidor (en el ejemplo 2)
  - Mientras en la respuesta haya un enlace **`next`**, se accede a ese enlace
  - Mientras tanto va sumando el número de productos obtenidos en cada intervalo

Comentarios (y 2)

- No se muestra la parte relativa al recurso **CategoryResource** por ser similar a lo ya visto

ClientTest.java (1)

```
package es.udc.rs.advjaxrstutorial.client.test;

public class ClientTest {
   RestClientCatalogService client = ...;
   public static void main(String[] args) { ... }

   ...

   private void testIterateProductList(String keyword)
         throws InstanceNotFoundException {
      ProductListInterval productList = client.findProducts(keyword);
      ... // mostrar resultado
      showDetails(productList);
      URI nextIntervalUri = productList.getNextIntervalUri();
      while (nextIntervalUri != null) {
         productList = client.findProductsByUri(nextIntervalUri);
         ... // mostrar resultado
         showDetails(productList);
         nextIntervalUri = productList.getNextIntervalUri();
      }
   }
```

ClientTest.java (y 2)

```
    private void showDetails(ProductListInterval productList)
          throws InstanceNotFoundException {
       for (Product p : productList.getProducts()) {
          ProductDetails productDetails =
          client.getProductByUri(p.getSelfUri());
          ... // mostrar resultado
          Category c = client.getCategoryByUri(
             productDetails.getCategoryUri());
             ... // mostrar resultado
       }
    }

}
```

Comentarios

- El método **`testIterateProductList`**
  - Recibe la palabra clave para realizar la búsqueda de productos
  - Va obteniendo los productos por intervalos, siguiendo la URI del enlace **`next`**
  - Para cada intervalo, lo muestra y a continuación llama al método **`showDetails`**
    - Para cada producto del intervalo se obtienen sus detalles utilizando la URI del enlace **`self`** y se muestran
    - Para cada producto se obtiene la categoría utilizando la URI del enlace **`category`** (contenida en los detalles del producto) y se muestra

Filtros de Servidor (1)

- Filtros de petición y de respuesta
  - Los de petición implementan **`ContainerRequestFilter`**
    - El parámetro de tipo **`ContainerRequestContext`** del método **`filter`** permite obtener, modificar y/o añadir prácticamente cualquier dato de la petición
    - Se dividen en
      - Filtros de PreMatching (anotados con **`@PreMatching`**), que se ejecutan antes de buscar el método que atenderá la petición
      - Filtros de PostMatching que se ejecutan después de buscar el método que atenderá la petición
        - » La URI y el método HTTP de la petición no pueden ser modificados desde estos filtros
  - Los de respuesta implementan **`ContainerResponseFilter`**
    - El parámetro de tipo **`ContainerRequestContext`** del método **`filter`** permite obtener datos de la petición
    - El parámetro de tipo **`ContainerResponseContext`** del método **`filter`** permite obtener, modificar y/o añadir los datos de la respuesta

Filtros de Servidor (y 2)

- Flujo de ejecución de filtros

```
for (filter : preMatchFilters) {
   filter.filter(request);
}
jaxrs_method = match(request);
for (filter : postMatchFilters) {
   filter.filter(request);
}
response = jaxrs_method.invoke();
for (filter : responseFilters) {
   filter.filter(request, response);
}
```

- Los filtros de servidor (al igual que los interceptors) deben anotarse con **@Provider**
- Gestión de Excepciones
  - Si durante el procesamiento de un filtro (o un Interceptor), en el lado servidor, se lanza una Excepción, el entorno de ejecución de JAX-RS la trata igual que si se hubiese lanzado desde un método de un recurso

AuthenticationFilter.java (1)

```
package es.udc.rs.advjaxrstutorial.service.filter;
...

@Priority(Priorities.AUTHENTICATION)
@Provider
@PreMatching
public class AuthenticationFilter implements ContainerRequestFilter {

   @Override
   public void filter(ContainerRequestContext reqContext)
         throws IOException {
      String authHeader = reqContext
         .getHeaderString(HttpHeaders.AUTHORIZATION);
      if (authHeader == null || !verifyCredentials(authHeader)) {
         throw new NotAuthorizedException(
           "MyAuthProtocol realm=\"Catalog Example\"");
      }
   }

   private static boolean verifyCredentials(String authHeader) { ... }
```

CacheControlFilter.java (1)

```
package es.udc.rs.advjaxrstutorial.service.filter;
...
@Priority(Priorities.HEADER_DECORATOR)
@Provider
@Cached
public class CacheControlFilter implements ContainerResponseFilter {

   @Override
   public void filter(ContainerRequestContext reqContext,
         ContainerResponseContext respContext) throws IOException {
      if (reqContext.getMethod().equals("GET")
            && respContext.getStatus() < 400) {
         CacheControl cc = new CacheControl();
         cc.setMaxAge(100);
         respContext.getHeaders().add("Cache-Control",
             RuntimeDelegate.getInstance()
                 .createHeaderDelegate(CacheControl.class).toString(cc)));
      }
   }
}
```

Cached.java (1)

```
package es.udc.rs.advjaxrstutorial.service.filter;
...
@NameBinding
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.TYPE, ElementType.METHOD})
public @interface Cached {
}
```

TrafficLoggerFilter.java (1)

```
package es.udc.rs.advjaxrstutorial.service.filter;
...
@Priority(0)
@PreMatching
@Provider
public class TrafficLoggerFilter implements ContainerRequestFilter,
      ContainerResponseFilter {

   @Override
   public void filter(ContainerRequestContext reqContext,
         ContainerResponseContext respContext) throws IOException {
      logResponse(reqContext, respContext);
   }

   @Override
   public void filter(ContainerRequestContext reqContext)
         throws IOException {
      logRequest(reqContext);
   }
```

TrafficLoggerFilter.java (y 2)

```
    private static void logRequest(ContainerRequestContext reqContext) {
       UriInfo uriInfo = reqContext.getUriInfo();
       String method = reqContext.getMethod();
       MultivaluedMap<String,String> headers = reqContext.getHeaders();
       List<Object> matchedResources = uriInfo.getMatchedResources();
       System.out.println("\n* REQUEST: " + method + " " +
          uriInfo.getRequestUri() + "\n  Headers: " + headers +
          "\n  Matched Resources: " + matchedResources);
    }

    private static void logResponse(ContainerRequestContext reqContext,
          ContainerResponseContext respContext) {
       int status = respContext.getStatus();
       MultivaluedMap<String, String> stringHeaders =
          respContext.getStringHeaders();
       UriInfo uriInfo = reqContext.getUriInfo();
       String method = reqContext.getMethod();
       System.out.println("\n* RESPONSE TO: " + method + " " +
          uriInfo.getRequestUri() + "\n  Status: " + status +
                "\n  Headers: " + stringHeaders);
    }
}
```

Comentarios (1)

- **`AuthenticationFilter`**
  - Filtro de petición de PreMatching que comprueba si en la petición viene la cabecera **`Authorization`**
    - Si viene entonces se comprueba su contenido (método **`verifyCredentials`**)
      - En el ejemplo únicamente se comprueba que contenga un texto “MyAuthProtocol <user>:<password>” teniendo que ser **`<user>`** igual a **`<password>`** para que la autenticación se considere correcta
    - Si no viene la cabecera, o la verificación de credenciales no es correcta entonces lanza la excepción **`NotAuthorizedException`**
      - Hija de **`WebApplicationException`**
      - Establece un código de respuesta 401

Comentarios (2)

- **`CacheControlFilter`**
  - Filtro de respuesta que añade una cabecera **`Cache-Control`** a las respuestas a peticiones GET
    - Comprueba si el método de la petición es GET
    - Añade la cabecera **`Cache-Control`** a la respuesta
  - Es posible indicar que un filtro (o un Interceptor) no se aplique a todos los métodos de los recursos
    - Es necesario crear una anotación y anotarla con la meta-anotación **`@NameBinding`**
    - Debe anotarse el filtro con esa anotación (**`@Cached`** en el ejemplo) y los métodos a los que debe aplicarse el filtro
      - En el ejemplo se supone que solamente se quiere añadir información de cache a las respuestas de detalles de un producto y una categoría, pero no a las búsquedas de productos. E.g.

```
@GET
@Path("/{id}")
@Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
@Cached
public ProductDetailsDtoJaxb findById(@PathParam("id") long id, ...)
   throws InstanceNotFoundException { ... }
```

Comentarios (y 3)

- **`TrafficLoggerFilter`**
  - Filtro de petición y respuesta (implementa ambas interfaces)
    - Loguea ciertos datos de las peticiones
      - Como está marcado como un filtro de PreMatching, puede comprobarse que **`matchedResources`** siempre será **`null`**
    - Loguea ciertos datos de las respuestas (tanto de la petición a la que se está respondiendo como de la respuesta enviada)

Orden de Ejecución (1)

- En el ejemplo nos interesa que
  - Para las peticiones se ejecute primero el filtro **`TrafficLoggerFilter`** y después el **`AuthenticationFilter`**
    - Para que las peticiones con información de autenticación errónea también se logueen
  - Para las respuestas nos interesa que se ejecute primero el filtro **`CacheControlFilter`** y después **`TrafficLoggerFilter`**
    - Para que se loguee la información de cabeceras añadida a algunas respuestas

Orden de Ejecución (2)

- Tanto para filtros como para Interceptors es posible indicar el orden en el que deben ejecutarse
  - Se hace asignándoles una prioridad numérica
  - Una de las formas de asignársela es a través de la anotación **`@Priority`**
  - La clase utilidad **`Priorities`** tiene definidas algunas contantes
    - **`AUTHENTICATION = 1000`**
    - **`AUTHORIZATION = 2000`**
    - **`HEADER_DECORATOR = 3000`**
    - **`ENTITY_CODER = 4000`**
    - **`USER = 5000`**
  - Si no se especifica prioridad se asume **`USER`**

Filtros de Servidor - Ejemplo

- Otro ejemplo típico de filtro de servidor de petición es el *tunneling* del método HTTP
  - Algunos firewalls no permiten las operaciones PUT y DELETE
  - Para resolver esta limitación, muchas aplicaciones invocan un GET e indican el método HTTP a utilizar mediante la cabecera **`X-Http-Method-Override`**
  - Debe ser de PreMatching para poder modificar el método de la petición

```
@Provider
@PreMatching
public class HttpMethodOverride implements ContainerRequestFilter {
   public void filter(ContainerRequestContext ctx) throws IOException {
      String methodOverride =
         ctx.getHeaderString("X-Http-Method-Override");
      if (methodOverride != null) ctx.setMethod(methodOverride);
   }
}
```

Filtros de Cliente (1)

- Filtros de petición y de respuesta
  - Los de petición implementan **`ClientRequestFilter`**
    - El parámetro de tipo **`ClientRequestContext`** del método **`filter`** permite obtener, modificar y/o añadir datos de la petición
  - Los de respuesta implementan **`ClientResponseFilter`**
    - El parámetro de tipo **`ClientRequestContext`** del método **`filter`** permite obtener datos de la petición
    - El parámetro de tipo **`ClientResponseContext`** del método **`filter`** permite obtener, modificar y/o añadir datos de la respuesta
- Los filtros de cliente (y también los interceptors)
  - Deben anotarse con **`@Provider`**
  - Deben registrarse explícitamente, igual que cualquier otro tipo de Provider, en un elemento que implemente la interfaz **`Configurable`**. E.g.
    - En la clase **`Client`** si se quieren aplicar a todas las peticiones (opción utilizada en los ejemplos)
    - En la clase **`WebTarget`** si se quieren aplicar solamente a ciertas peticiones

Filtros de Cliente (y 2)

- Gestión de excepciones
  - Si un filtro de cliente (o Interceptor) lanza una excepción
    - Si es una instancia de **`WebApplicationException`**, entonces el entorno de ejecución la *propaga* al código de la aplicación
    - En otro caso se encapsula dentro de una
      - **`ProcessingException`** si es lanzada antes de que la petición se envie al servidor (filtro de petición)
      - **`ResponseProcessingException`** si es lanzada cuando se procesa una respuesta (filtro de respuesta)

AuthenticationFilter.java

```
package es.udc.rs.advjaxrstutorial.client.rest.filter;
...
public class AuthenticationFilter implements ClientRequestFilter {

   private String userName;
   private String password;

   public AuthenticationFilter(String userName, String password) {
      this.userName = userName;
      this.password = password;
   }

   @Override
   public void filter(ClientRequestContext reqContext) throws IOException {
      String headerValue = "MyAuthProtocol " + userName + ":" + password;
      reqContext.getHeaders().putSingle(HttpHeaders.AUTHORIZATION,
         headerValue);
   }

}
```

CheckRequestFilter.java

```
package es.udc.rs.advjaxrstutorial.client.rest.filter;
...
public class CheckRequestFilter implements ClientRequestFilter {

   @Override
   public void filter(ClientRequestContext reqContext) throws IOException {
      if (reqContext.getHeaders().get("Accept") == null) {
         reqContext.abortWith(Response.status(Response.Status.BAD_REQUEST)
            .entity("Accept header must be defined.").build());
      }
   }

}
```

ErrorLoggerFilter.java

```
package es.udc.rs.advjaxrstutorial.client.rest.filter;
...
public class ErrorLoggerFilter implements ClientResponseFilter {
   @Override
   public void filter(ClientRequestContext reqContext,
         ClientResponseContext respContext) throws IOException {
      if (respContext.getStatus() >=
            Response.Status.BAD_REQUEST.getStatusCode()) {
         logResponse(reqContext,respContext);
      }
   }
   private static void logResponse(ClientRequestContext reqContext,
         ClientResponseContext respContext) {
      int status = respContext.getStatus();
      MultivaluedMap<String, String> stringHeaders =
         reqContext.getStringHeaders();
      URI uri = reqContext.getUri();
      String method = reqContext.getMethod();
      System.out.println("\n*** ERROR IN RESPONSE" + "\n  Request: " +
         method + " " + uri + "\n  Headers: " + stringHeaders +
         "\n  Response Status: " + status);
   }
}
```

RestClientCatalogService.java

```
package es.udc.rs.advjaxrstutorial.client.rest;
...
public class RestClientCatalogService {
   private String userName;
   private String password;
   private jakarta.ws.rs.client.Client client = null;

   public RestClientCatalogService(String userName, String password) {
      this.userName = userName;
      this.password = password;
   }

   private Client getClient() {
      if (client == null) {
         client = ClientBuilder.newClient();
         client.register(new AuthenticationFilter(userName, password));
         client.register(CheckRequestFilter.class);
         client.register(ErrorLoggerFilter.class);
         ...
      }
      return client;
   }
```

Comentarios (1)

- **`AuthenticationFilter`**
  - Filtro de petición que en el constructor recibe como parámetros el nombre de usuario y contraseña del usuario que va a interactuar con el servidor
  - Añade a la petición la cabecera **`Authorization`** con el valor "MyAuthProtocol <user>:<password>"
  - Al registrarlo en la clase **`Client`** es necesario registrar una instancia con el usuario y la contraseña establecidos (en lugar de registrar la clase)
    - **`RestClientCatalogService`** se ha modificado para que en su constructor reciba el nombre de usuario y contraseña que se desea utilizar en el filtro de autenticación
    - La propiedad **`client`** y el método **`getClient`** no son **`static`** porque diferentes instancias de **`RestClientCatalogService`** podrán registrar un filtro de autenticación con un nombre de usuario y contraseña diferentes

Comentarios (2)

- **`CheckRequestFilter`**
  - Filtro de petición que comprueba que la petición contenga la cabecera **`Accept`**
  - En caso de que no contenga esa cabecera se devuelve una respuesta al cliente sin llegar a enviar la petición al servidor
    - Se hace a través del método **`abortWith`** de **`ClientRequestContext`**, que recibe como parámetro un objeto de tipo **`Response`** con la respuesta a devolver al cliente
    - En el ejemplo se devuelve un error 400 y en el cuerpo un texto explicativo del error
  - NOTA: Para probar este filtro se ha añadido a la clase **`RestClientCatalogService`** el método **`getProductByIdWithoutType`**
    - Hace una petición para obtener los detalles de un producto a partir de su identificador, pero no especifica ningún tipo de contenido aceptado por el cliente (i.e. no especifica la cabecera **`Accept`**)

Comentarios (y 3)

- **`ErrorLoggerFilter`**
  - Filtro de respuesta que loguea todas las respuestas de error
  - Mira si el código de estado es mayor que 400 y en caso de serlo loguea información sobre
    - La petición realizada: método, URI y cabeceras
    - La respuesta obtenida: código de estado

Interceptors (1)

- Se dividen en Interceptors de lectura y de escritura
  - Se ejecutan únicamente cuando se utiliza un **`MessageBodyReader`** o un **`MessageBodyWriter`** para deserializar o serializar un objeto Java desde o al cuerpo de una petición HTTP
    - Se invocan en la misma pila de llamadas Java
  - Los Interceptor de lectura envuelven la invocación de **`MessageBodyReader.readFrom`** y los Interceptor de escritura envuelven la invocación de **`MessageBodyWriter.writeTo`**
- Se pueden usar tanto en el cliente como en el servidor
  - Se registran igual que los Filtros

Interceptors (2)

- Los Interceptor de escritura implementan **`WriterInterceptor`**
  - El parámetro de tipo **`WriterInterceptorContext`** del método **`aroundWriteTo`** permite
    - Acceder a y modificar las cabeceras del mensaje HTTP
    - Acceder a y modificar la entidad a enviar en el cuerpo del mensaje HTTP
    - Acceder a y modificar el **`OutputStream`** sobre el que se va a serializar la entidad a enviar en el cuerpo del mensaje HTTP
  - La llamada al método **`proceed`** de **`WriterInterceptorContext`** invoca al siguiente Interceptor de escritura registrado o, si no hay más, al método **`writeTo`** del **`MessageBodyWriter`** correspondiente

Interceptors (y 3)

- Los Interceptors de lectura implementan **`ReaderInterceptor`**
  - El parámetro de tipo del **`ReaderInterceptorContext`** del método **`aroundReadFrom`** permite
    - Acceder a y modificar las cabeceras de la petición/respuesta
    - Acceder a y modificar el **`InputStream`** usado para leer el cuerpo del mensaje HTTP
  - La llamada al método **`proceed`** de **`ReaderInterceptorContext`** invoca al siguiente Interceptor de lectura registrado o, si no hay más, al método **`readFrom`** del **`MessageBodyReader`** correspondiente
  - El método **`aroundReadFrom`** debe devolver el objeto resultante de **`deserializar`** el cuerpo del mensaje HTTP
    - Normalmente será el resultado de invocar al método **`proceed`** del parámetro **`ReaderInterceptorContext`**

GzipEncoder.java

```
package es.udc.rs.advjaxrstutorial.service.interceptor;
...
@Provider
public class GzipEncoder implements WriterInterceptor {

   @Override
   public void aroundWriteTo(WriterInterceptorContext ctx)
         throws IOException, WebApplicationException {
      GZIPOutputStream os = new GZIPOutputStream(ctx.getOutputStream());
      ctx.getHeaders().putSingle("Content-Encoding", "gzip");
      ctx.setOutputStream(os);
      ctx.proceed();
   }

}
```

GzipDecoder.java

```
package es.udc.rs.advjaxrstutorial.client.rest.interceptor;
...
@Provider
public class GzipDecoder implements ReaderInterceptor {

   @Override
   public Object aroundReadFrom(ReaderInterceptorContext ctx)
         throws IOException {
      String encoding = ctx.getHeaders().getFirst("Content-Encoding");
      if (!"gzip".equalsIgnoreCase(encoding)) {
         return ctx.proceed();
      }
      GZIPInputStream is = new GZIPInputStream(ctx.getInputStream());
      ctx.setInputStream(is);
      return ctx.proceed();
   }

}
```

Comentarios (1)

- **`GZipEncoder`**
  - Comprime el cuerpo de las peticiones a enviar, con formato gzip
    - Se usa solamente en el servidor porque en nuestro ejemplo el cliente nunca envía nada en el cuerpo de las peticiones
  - Accede a las cabeceras del mensaje y añade la cabecera **`Content-Encoding`** con valor **`gzip`**
  - Sustituye el **`OutputStream`** que se utilizará para escribir la entidad serializada por un **`GzipOutputStream`**
    - Todo el contenido escrito a través de un **`GzipOutputStream`** se comprime en formato gzip

Comentarios (y 2)

- **`GZipDecoder`**
  - Descomprime el cuerpo de las respuestas recibidas con formato gzip
    - Se usa solamente en el cliente porque en nuestro ejemplo el servidor nunca recibe nada en el cuerpo de las peticiones
  - Mira si en el mensaje viene la cabecera **`Content-Encoding`** con valor **`gzip`**
    - Si no viene, devuelve el resultado de invocar a **`proceed`** sobre el **`ReaderInterceptorContext`** (o sea, lo que devuelve el siguiente Interceptor de lectura o el **`MessageBodyReader`** correspondiente)
    - Si viene, sustituye el **`IntputStream`** que se utilizará para leer cuerpo del mensaje por un **`GzipIntputStream`** y devuelve el resultado de invocar a **`proceed`** sobre el **`ReaderInterceptorContext`**
      - Todo el contenido leído a través de un **`GzipIntputStream`** se descomprime asumiendo el formato gzip