Unidad 4 Creación de servicios REST con JAX-RS

1. API REST con Anotaciones de JAX-RS.

JAX-RS define cinco anotaciones que se corresponden con operaciones HTTP específicas:

• @javax.ws.rs.GET

• @javax.ws.rs.PUT

• @javax.ws.rs.POST

• @javax.ws.rs.DELETE

• @javax.ws.rs.HEAD

La anotación @Path

La anotación @Path identifica la "plantilla" de path para la URI del recurso al que se accede y se puede especificar a nivel de clase o a nivel de método de dicho recurso.

El valor de una anotación @Path es una expresión que denota una URI relativa a la URI base del servidor en el que se despliega el recurso, a la raiz del contexto de la aplicación, y al patrón URL al que responde el runtime de JAX-RS.

La anotación @Path no es necesario que contenga una ruta que empiece o termine en el carácter / . El runtime de JAX-RS analiza igualmente la expresión indicada como valor de @Path Para que una clase Java sea identificada como una clase que puede atender peticiones HTTP, ésta tiene que estar anotada con al menos la expresión: @Path("/") . Este tipo de clases se denominan recursos JAX-RS raíz.

Para recibir una petición, un método Java debe tener al menos una anotación de método

HTTP, como por ejemplo @javax.ws.rs.GET.

Usos de las anotaciones @Produces y @Consumes

La información enviada a un recurso y posteriormente devuelta al cliente que realizó la petición se especifica con la cabecera HTTP Media-Type , tanto en la petición como en la respuesta. Como ya hemos visto, podemos especificar que representaciones de los recursos (valor de Media_Type) son capaces de aceptar y/o producir nuestros servicios mediante las siguientes anotaciones:

• javax.ws.rs.Consumes

• javax.ws.rs.Produces

La ausencia de dichas anotaciones es equivalente a incluirlas con el valor de media type / , es decir, su ausencia implica que se soporta (acepta) cualquier tipo de representación.

Anotación @Consumes

Esta anotación funciona conjuntamente con @POST y @PUT. Le indica al framework (librerías JAX-RS) a qué método se debe delegar la petición de entrada. Específicamente, el cliente fija la cabecera HTTP Content-Type y el framework delega la petición al correspondiente método capaz de manejar dicho contenido. Un ejemplo de anotación con @PUT es la siguiente:

```
@Path("/pedidos")
public class PedidoResource {
@PUT
@Consumes("application/xml")
public void modificarPedido(InputStream representation) { }
}
```

Si @Consumes se aplica a la clase, por defecto los métodos de respuesta aceptan los tipos especificados de tipo MIME. Si se aplica a nivel de método, se ignora cualquier anotación @Consumes a nivel de clase para dicho método.

En este ejemplo, le estamos indicando al framework que el método modificarPedido() acepta un input stream cuyo tipo MIME es "application/xml", y que se almacena en la variable representation. Por lo tanto, un cliente que se conecte al servicio web a través de la URI / pedidos debe enviar una petición HTTP PUT conteniendo el valor de application/xml como tipo MIME de la cabecera HTTP Content-Type .

Si no hay métodos de recurso que puedan responder al tipo MIME solicitado, se le devolverá al cliente un código HTTP 415 ("Unsupported Media Type"). Si el método que consume la representación indicada como tipo MIME no devuelve ninguna representación, se enviará un el código HTTP 204 ("No content"). Como por ejemplo sucede en el código siguiente:

```
@POST
@Consumes("application/xml")
public void creaPedido(InputStream pedido) {
// Crea y almacena un nuevo _Pedido_
}
```

Podemos ver que el método "consume" una representación en texto plano, pero devuelve void, es decir, no devuelve ninguna representación.

Un recurso puede aceptar diferentes tipos de "entradas". Así, podemos utilizar la anotación

@PUT con más de un método para gestionar las repuestas con tipos MIME diferentes. Por ejemplo, podríamos tener un método para aceptar estructuras XML, y otro para aceptar estructuras JSON.

estructuras JSON.

```
@Path("/pedidos")
public class PedidoResource {
@PUT
@Consumes("application/xml")
public void modificarPedidoXML(InputStream pedido) { }
@PUT
@Consumes("application/json")
public void modificarPedidoJson(InputStream pedido) { }
}
```

Anotación @Produces

Esta anotación funciona conjuntamente con @GET, @POST y @PUT. Indica al framework qué tipo de representación se envía de vuelta al cliente.

De forma más específica, el cliente envía una petición HTTP junto con una cabecera HTTP Accept que se mapea directamente con el Content-Type que el método produce. Por lo tanto, si el valor de la cabecera Accept HTTP es application/xml, el método que gestiona la petición devuelve un stream de tipo MIME application/xml. Esta anotación también puede utilizarse en más de un método en la misma clase de recurso. Un ejemplo que devuelve representaciones XML y JSON sería el siguiente:

```
@Path("/pedidos")
public class PedidoResource {
@GET
@Produces("application/xml")
public String getPedidoXml() { }
@GET
@Produces("application/json")
public String getPedidoJson() { }
}
```

Se puede declarar más de un tipo en la misma declaración @Produces, como por ejemplo:

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

2. Librería JAXB

JAXB (no confundir con la interfaz de acceso JAXP) es una librería de (Un)-Marshalling.

- El concepto de **Serialización o Marshalling** que ya ha sido introducido, es el proceso de almacenar un conjunto de objetos en un fichero.
- **Unmarshaling** es justo el proceso contrario: convertir en objetos el contenido de un fichero.

JAXB es capaz de obtener de un esquema XML una estructura de clases que le da soporte en Java. Básicamente, la parte más importante aquí es el uso de la clase `javax.xml.bind.JAXBContext`. Esta clase proporciona un marco para la validación, serialización de objetos Java a XML y es el punto de entrada a la API JAXB.

Adaptadores

Al manipular los tipos complejos que pueden no estar disponibles directamente en JAXB tenemos que escribir un adaptador para indicar JAXB cómo manejar el tipo específico.

Anotaciones

Anotaciones utilizadas en JAXB para XML Vamos a enumerar aquí las más importantes:

- `XmlAccessorOrder` : Esta anotación controla el orden de los campos y las propiedades de una clase en la que aparecerá en el XML.
- `XmlAccessorType` : indica si un elemento se debe serializar o no. Se utiliza en combinación con `javax.xml.bind.annotation .XMLAccessType` .
- `XmlAnyAttribute` : Asigna un elemento a un mapa de atributos de comodín.
- `XmlAttribute` : Esta anotación es uno de los básicos y más utilizados. Se asigna un elemento de Java (propiedad, un atributo de campo) a un atributo del nodo XML.
- `XmlElement` : Asigna un elemento de Java para un nodo XML usando el nombre.
- `XmlElementRef` : Asigna un elemento de Java para un nodo XML usando su tipo (diferente a la anterior, donde se utiliza el nombre de mapeo).
- `XmlElementRefs` : las marcas de una propiedad que se refiere a las clases con `XmlElement` o `JAXBElement` .
- `XmlElements` : Este es un contenedor de múltiples `XMLElement` anotaciones.

- `XmlElementWrapper` : Genera una envoltura alrededor de una representación XML, destinado a ser utilizado con las colecciones.
- `XmlEnum` : Proporciona mapeo para una enumeración de una representación XML. Funciona en combinación con `XmlEnumValue` .
- `XmlEnumValue` : asigna una constante para un elemento XML enumeración.
- `XmlID` : Mapea una propiedad con un identificador XML.

- `XmlMixed` : El elemento anotado puede contener contenido mixto. l
- `XmlRootElement` : Este es probablemente el más utilizado dentro de anotación JAXB. Se utiliza para asignar una clase a un elemento XML.
- `XmlSchema` : asigna un paquete a un espacio de nombres XML.
- `XmlSchemaType` : Asigna un tipo Java a un simple esquema de tipo incorporado.
- `XmlType` : Se utiliza para asignar una clase o una enumeración a un tipo en un esquema XML.
- `XmlValue` : Permite asignar una clase a un tipo complejo esquema XML con un simpleContent o un tipo simple esquema XML.