ESCUELA DE SISTEMAS Y TECNOLOGÍAS

Transparencias de **ANALISTA DE SISTEMAS**

***Edición 2020 Materia:* Java Web**

TEMA: Servicios Web

Agenda

- Servicios Web SOAP
- Servicios Web REST

Servicios Web SOAP (1)

- JavaEE incluye la especificación JAX-WS, que define un API para crear y consumir servicios web SOAP.
- METRO es la implementación de referencia de la especificación JAX-WS.
- Este API permite definir los *endpoints* y clientes de una manera sencilla utilizando *annotations*.
- Para crear un servicio web SOAP basta con definir una clase Java con los métodos correspondientes, utilizando las siguientes *annotations*:
  - **@WebService(serviceName = "nombreServicio")**: Define la clase java como un servicio web.
  - **@WebMethod(operationName = "nombreMetodo")**: Define un método de dicha clase como un método de servicio.
  - **@WebParam(name = "nombreParametro")**: Define un parámetro de un método de servicio.

Servicios Web SOAP (2)

- Los tipos de datos utilizados por los métodos de servicio que no correspondan a los tipos primitivos, deben implementar la interfaz Serializable.
- Toda excepción lanzada dentro de un método de servicio (incluyendo a las excepciones personalizadas) será serializada automáticamente a una **SOAP FAULT** que se enviará al cliente.
- Las herramientas de NetBeans se encargarán de referenciar las bibliotecas correspondientes y agregar la información necesaria al *deployment descriptor*.
- Cada servicio web quedará publicado en una *url* que mapea al servlet **com.sun.xml.ws.transport.http.servlet.WSServlet**. Por ej.: .../***MiServicio***

Servicios Web SOAP (3)

- El WSDL del servicio se podrá acceder mediante la misma url con el parámetro *wsdl*: .../**MiServicio?wsdl**
- Dicho servlet se encargará de gestionar el **endpoint** del servicio.
- También se creará el archivo **sun-jaxws.xml** con la información de los **endpoints**.
- Para consumir un servicio web SOAP basta con agregar una referencia de servicio web (*web service client*) al proyecto, indicando la url del **WSDL**.
- A partir de dicho WSDL, NetBeans se encargará de generar automáticamente las clases ***proxy*** que permitirán comunicar el proyecto con el servicio.

Servicios Web SOAP (4)

- Entre esas clases se encuentran las clases del servicio (***NombreServicio_Service***) y del puerto (***NombreServicio***), que nos permitirá invocar a los métodos de servicio:
  - *NombreServicio_Service servicio = new NombreServicio_Service();*
  - *NombreServicio puerto = servicio.getNombreServicioPort();*
  - *puerto.metodoServicio(...);*
- También se generarán automáticamente las clases correspondientes a las excepciones originadas por el servicio (***TipoExcepcion_Exception***).
- En caso de recibir como respuesta una **SOAP FAULT** al momento de invocar un método de servicio, se disparará una excepción de dicho tipo en el cliente.

Servicios Web REST (1)

- JavaEE incluye la especificación JAX-RS, que define un API para crear y consumir servicios web REST.
- JERSEY es la implementación de referencia de la especificación JAX-RS.
- Este API permite definir los recursos y clientes de una manera sencilla utilizando *annotations*.
- Para crear un servicio web REST, lo primero es definir la clase de la aplicación REST que extienda de **javax.ws.rs.core.Application**.
- En dicha clase debe aplicarse la *annotation* @**ApplicationPath**("/**rutaBaseAplicacion**")
- Luego deberán definirse las clases de los servicios que contendrán los métodos para acceder a los recursos.

Servicios Web REST (2)

- En dichas clases se podrán utilizar diferentes *annotations* para que JAX-RS sepa cómo publicar cada recurso.
- Estos métodos de recurso deben retornar un objeto **javax.ws.rs.core.Response**
- **Algunas annotations que puede utilizarse**:
  - **@Path("ruta")**: Si se aplica sobre una clase de servicio define la ruta base (a partir de la ruta base de la aplicación REST) de los recursos correspondientes a sus métodos. Si se aplica sobre un método de recurso define la ruta (a partir de la ruta base de la aplicación y de la ruta base de la clase del servicio) correspondiente al mismo. Puede definir parámetros dentro de la ruta utilizando llaves: "/ruta/{par1}".

Servicios Web REST (3)

- **Algunas annotations que puede utilizarse (cont.)**:
  - **@PathParam**("**nombreParametro**"): Se aplica sobre un parámetro de un método de recurso para indicar que su valor proviene de un parámetro de la ruta definida en el método.
  - **@GET**, **@POST**, **@PUT**, **@DELETE**, etc.: Se aplica sobre un método de recurso para indicar el tipo de *request* al que responde.
  - **@Produces(MediaType.X)**: Se aplica sobre un método de recurso para indicar el tipo de contenido con el que responde (MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML, etc.). Puede recibir un array con varios MediaTypes { MT.X, MT.Y }.
  - **@Consumes(MediaType.X)**: Se aplica sobre un método de recurso para indicar el tipo de contenido que recibe para poblar la información de sus parámetros. Puede recibir un array con varios MediaTypes.

Servicios Web REST (4)

- La clase **Response** dispone de métodos estáticos que permiten construir la respuesta de un método de recurso.
- Cada uno de estos métodos permite configurar algún aspecto de la respuesta, a la vez que retorna un objeto **ResponseBuilder** con toda la configuración realizada hasta el momento. Al finalizar se invoca al método **build**() que retorna el objeto *Response* para devolver en el método de recurso.
- Algunos de estos métodos son:
  - **status(Response.Status.X**): establece el estado de la respuesta http.
  - **entity(Object**): establece el objeto que se enviará serializado (json, xml, etc.) en el cuerpo de la respuesta.

Servicios Web REST (5)

- En caso de producirse algún tipo de error en un método de recurso deberá construirse y lanzarse una **WebApplicationException**, pasándole por parámetro a su constructor, el objeto Response con un estado **Response.Status.INTERNAL_SERVER_ERROR** y la información del error serializada en el cuerpo de respuesta.