UTN SF

SERVICIOS WEB RESTFUL

Servicios web REST

- REST es un estilo arquitectural de definir servicios web que usa los estándares de la web.
- Está disponible en JavaEE desde la versión 6.
- Se adaptaron mejor a los nuevos frameworks, a la evolución de JavaScript y a HTML5 que SOAP.
  - Además son más sencillos de producir y consumir.
- Sus principales principios son:
  - Todo es un recurso que puede ser identificado por una URL
  - Un recurso puede ser representado por múltiples formatos, definido adecuadamente por el tipo MIME correspondiente. Se usan métodos estándar entre el cliente y el servidor para negociar el contenido.
  - Usa todo el protocolo HTTP para intercambiar información.
  - La comunicación es sin estado.

Novedades en JavaEE7

- Java EE 6 añade soporte para servicios web REST a través de la adición de la API Java para Servicios Web REST (JAX-RS).
  - JAX-RS ha estado disponible como una API independiente mucho tiempo, y se convirtió en parte de Java EE en la versión 6.
- JAX-RS 2.0 (JSR 339), es un “major release” el cual pone foco en la integración de REST con las nuevas características de JavaEE7
- Las principales mejoras respecto de JAX-RS 1.x son:
  - *Se incorpora un API de cliente que no existía en versiones anteriores.*
  - *Tiene integración con CDI, por lo que se pueden aplicar filtros o interceptores para realizar procesamiento sobre los request.*
  - *Se incorpora procesamiento asincrónico que permite implementar interfaces con tiempo de polling más largo o server-side push.*
  - *Se integra con Bean Validation*
- Jersey, es la implementación de JAX-RS incluida por GlassFish.

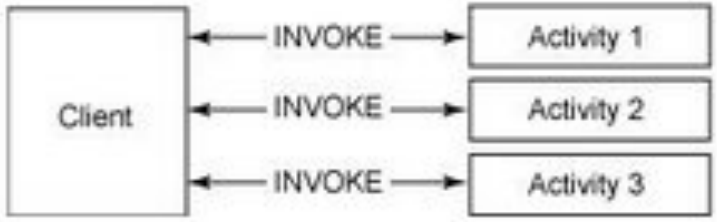
Servicios web basados en Rest

- Proporciona un ambiente de trabajo mucho más liviano y simplificado, dado que elimina todas las cabeceras y archivos adicionales que tiene SOAP, dado que no es necesario interpretar nada más allá de lo que es el protocolo HTTP.
  - útil para dispositivos que tienen ancho de banda acotado como por ejemplo dispositivos móviles.
  - También es mucho más sencilla su integración con AJAX.

Comparación SOAP / RESTful

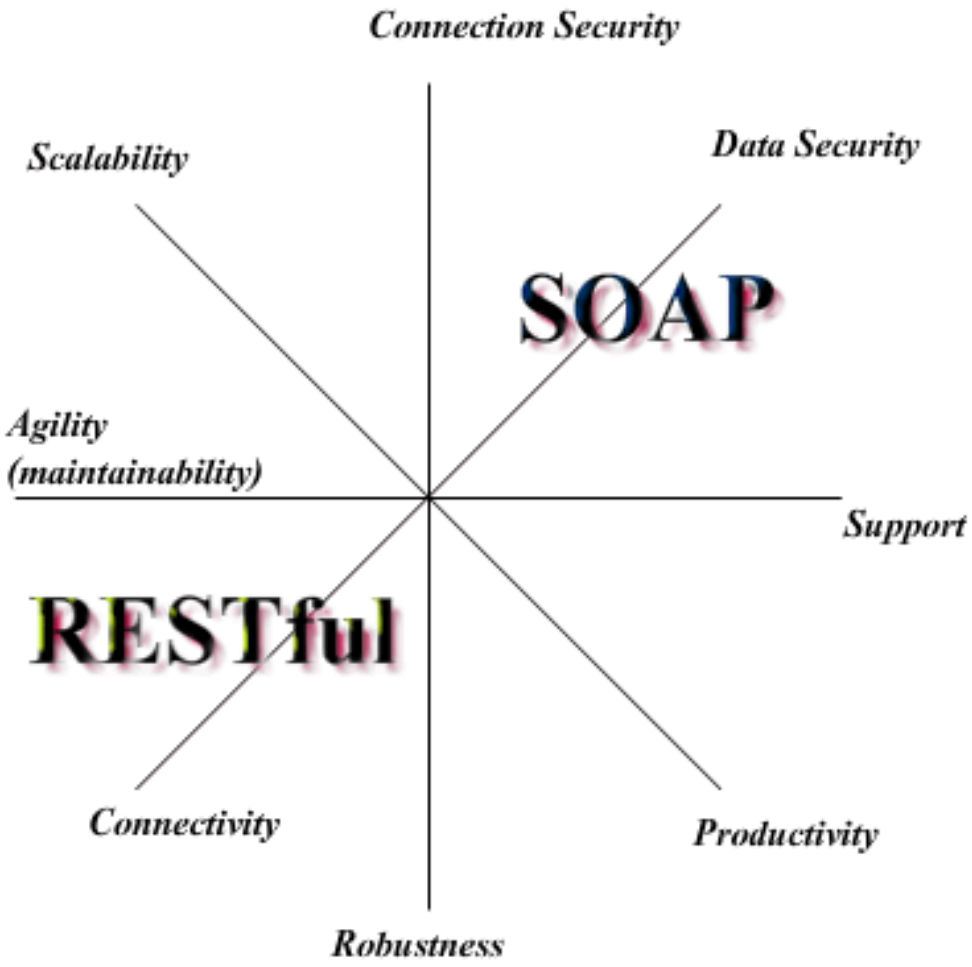
- JAX-WS (SOAP) se utiliza para arquitecturas más complejas donde los requisitos de calidad de servicio y fiabilidad de la información y las operaciones son más altas
- En escenarios donde estas restricciones no son tan fuertes, el uso de JAX-RS (RESTFul) es apropiado dado que como veremos a continuación es mucho más simple y sencillo de desarrollar.

REST-style Web services



SOAP-style Web services

Comparación REST / SOAP



Tipos de Request HTTP que deben soportar

- Los servicios web deben soportar uno o más de los siguientes métodos HTTP
  - GET – por convención una petición HTTP GET se usa para recuperar un recurso existente.
  - POST – por convención una petición HTTP GET se usa para actualizar un recurso existente.
  - DELETE – por convención una petición HTTP GET se usa para borrar un recurso existente.
  - PUT por convención una petición HTTP GET se usa para crear un recurso nuevo.

Equivalencias Métodos HTTP y SQL

Acción	SQL	HTTP
Create	Insert	PUT
Read	Select	GET
Update	Update	POST
Delete	Delete	DELETE

Servicios web REST

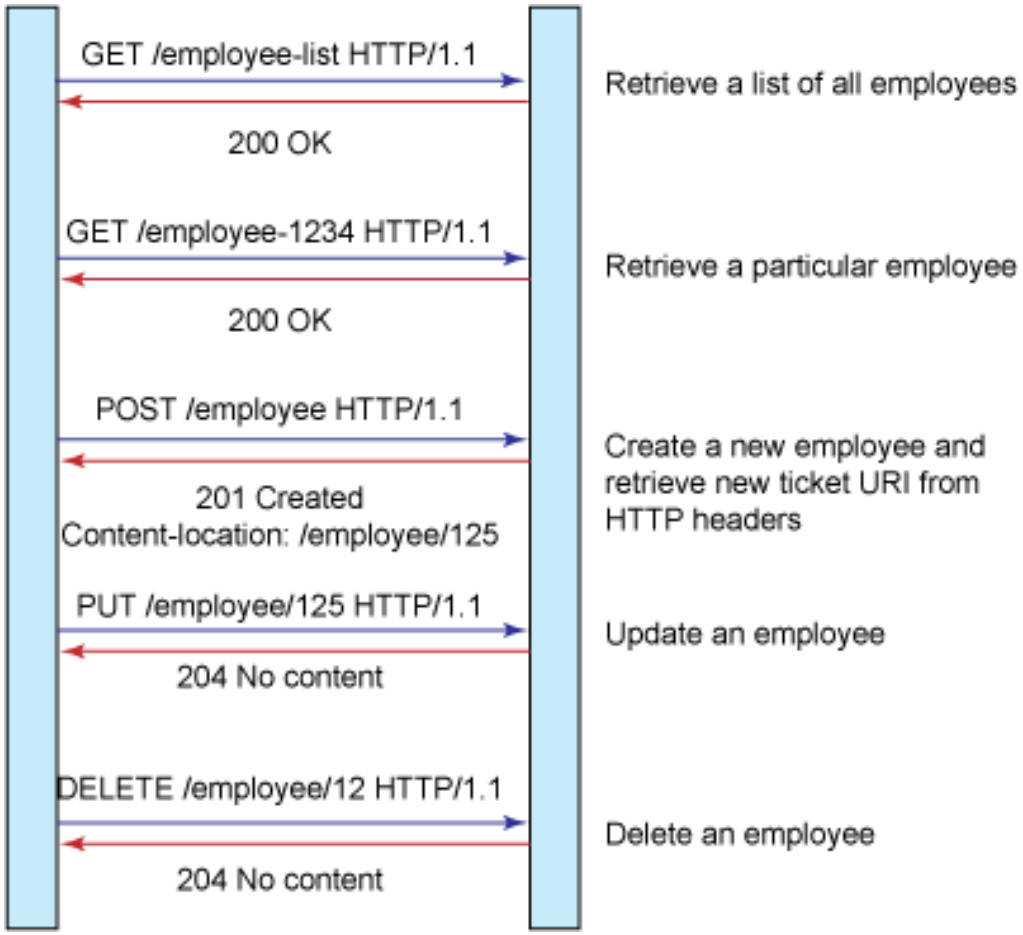
- El desarrollador JavaEE7 puede crear un servicio web anotando simplemente los métodos de una clase administrada por el contenedor.
- Este método es el que será invocado cuando una petición HTTP intente acceder al recurso definido por este método.
- Luego veremos como mediante anotaciones podemos definir la URL asociada a cada método para definirlo como un recurso accesible por clientes.
- Los servicios web RESTFul son muy flexibles. En general pueden producir muchos tipos diferentes de elementos MIME como resultado.

Tipos MIME mas comunes

.mtz	application/metastream
.mzv	application/metastream
.pdf	application/pdf
.png	image/png
.ps	application/postscript
.ra	application/vnd.rn-realmedia
.ram	audio/x-pn-realaudio
.rm	audio/x-pn-realaudio
.shtml	text/x-server-parsed-html
.swf	application/x-shockwave-flash
.tar	application/octet-stream
.tgz	application/octet-stream
.txt	text/plain
.vrml	x-world/x-vrml
.wav	audio/x-wav
.wbmp	image/vnd.wap.wbmp
.wml	text/vnd.wap.wml
.wmlc	application/vnd.wap.wmlc
.wmls	text/vnd.wap.wmlscript
.wmlsc	application/vnd.wap.wmlscriptc
.wrl	x-world/x-vrml
.xls	application/vnd.ms-excel
.xml	text/xml
.zip	application/zip

Servicios web REST

Un servicio web en REST es en realidad es una URL que identifica la ejecución de un método en el servidor. La semántica de este método estará asociada al tipo de request HTTP enviado



Ejemplo de servicio web REST

- Para definir una clase como proveedora de servicios web JAX-RS basta con asignarle a nivel de clase la anotación @Path.
- Esta URL define la ubicación del servicio web para ser ejecutado. Debe ser única.
  - Luego hay que definir mediante anotaciones que métodos HTTP serán implementados.
- El API JAX-RS, soporta las anotaciones @GET, @POST, @PUT, y @DELETE. Decorar un método con alguna de estas anotaciones simplemente hace que el mismo responda a las peticiones HTTP correspondientes.
- Finalmente, cada método puede **producir o consumir** un elemento como respuesta, por ejemplo algún valor que sea un tipo MIME válido y que sea transmitido mediante HTTP como respuesta o como parámetro.

Ejemplo: Código

```
1.  import javax.ws.rs.Consumes;
2.  import javax.ws.rs.DELETE;
3.  import javax.ws.rs.GET;
4.  import javax.ws.rs.POST;
5.  import javax.ws.rs.PUT;
6.  import javax.ws.rs.Path;
7.  import javax.ws.rs.Produces;
8.  @Path("customer")
9.  public class CustomerResource {
10. @PUT
11. @Consumes("text/xml")
12. public void createCustomer(String customerXML){
13. // crear en la base de datos
14. System.out.println("customerXML = " + customerXML);
15. }
```

Ejemplo: Código (cont)

```
1.  @GET
2.  @Produces("text/xml")
3.  public String getCustomer() {
4.  //buscar en la base de datos.
5.  return "<customer>\n«
6.  + "<id>123</id>\n«
7.  + "<firstName>Joseph</firstName>\n«
8.  + "<middleName>William</middleName>\n«
9.  + "<lastName>Graystone</lastName>\n«
10. + "</customer>\n";
11. }
```

Ejemplo: Código (cont)

```
1.  @POST
2.  @Consumes("text/xml")
3.  public void updateCustomer(String customerXML) {
4.  // actualizar en la base de datos.
5.  System.out.println("customerXML = " + customerXML);
    }
6.  @DELETE
7.  @Consumes("text/xml")
8.  public void deleteCustomer(String customerXML){
9.  //borrar de la base de datos
10. System.out.println("customerXML = " + customerXML);
11. }
12. }
13.
```

Análisis del ejemplo

- La anotación @Path a nivel de clase, designa la URL en la cual el servicio web será accesible.
- Para el ejemeplo anterior
  - asumimos que esta clase está desplegada en el servidor local,
  - el puerto HTTP es el puerto 8080,
  - el context root de la aplicación “ejemplo_rest”,
  - todas las peticiones REST son atendidas por el patrón de URL “resources”

  http:// localhost:8080 /ejemplo_rest / resources / customer.
- Así según lo invoquemos con POST, se ejecutará el método “updateContumer” con PUT “createCustomer” y así sucesivamente.

Análisis del ejemplo

- Recordemos que los servicios web REST retornan información a través de HTTP por lo que la información retornada tiene que ser un tipo MIME válido.
- En general, lo que retornan estos servicios web son datos con formato
  - XML, o formato de serialización de objetos en javaScript, JSON.
- Para definir que todos los métodos de una clase consumen y / o producen estos tipos MIME, podemos usar las anotaciones **@Consumes y @Produces** a nivel de clase o de método

Transformar un EJB 3.1 en RESTFul

```
1.  @Path("books")
2.  @Stateless
3.  @Produces({"application/xml", "application/json"})
4.  @Consumes({"application/xml", "application/json"})
5.  public class BookResource {
6.  @PersistenceContext(unitName = "chapter15PU")
7.  private EntityManager em;
8.  @GET
9.  public List<Book> getAllBooks() {
10. Query query = em.createNamedQuery("findAllBooks");
11. List<Book> books = query.getResultList();
12. return books;
13. }
```

Simplemente sumando la anotación @Path sobre una clase java plana que ya esté anotada con @Staless. Transformamos a un EJB, en un componente que no so atiende peticiones RMI sino también responde a servicio web REST

La Anotación @Path

- La anotación @Path, debe poseer un valor de URL relativo al "root" de la aplicación.
  - Se puede aplicar tanto a clases como a nivel de métodos.
- Ejemplo
  - @Path("/customers")
  - public class CustomersResource {
  - @GET
  - public List getCustomers() {// ...}
  - }
- **Invocarlo con GET** http://localhost:8080/miAPP/customers

La Anotación @Path

- También se pueden generar URL de tipo plantillas con variables embebidas, las cuales son evaluadas por el contenedor en tiempo de ejecución.
- Ejemplo: **@Path("/customers/{customername}")**”
  - Si se se invoca como http://localhost:8080/miAPP/customers/martin, genera una variable para el método, que puede estar vinculada a «customername» y que tendrá como valor «martin».

La Anotación @Path

- Si la anotación @Path se marca en el método y en la clase, no se sobrescriben como habitualmente hace Java, sino que se **«concatenan»**
  Si la anotación @Path existe en ambos, el método y la clase, entonces el resultado del path del servicio web, es la concatenación. Entonces en el siguiente ejemplo, el método es ejecutado ante la invocación de “http://localhost: 8080/miAPP/customers/sucursalA”

La Anotación @Path

```
1.  @Path("/items/")
2.  public class ItemsResource {
3.  @GET
4.  public List<Item> getListOfItems() { // ...}
5.  @GET
6.  @Path("{itemid}")
7.  public Item getItem(@PathParam("itemid") String itemid) { // ... }
8.  @GET
9.  @Path("/books/")
10. public List<Book> getListOfBooks() { // ... }
11. @GET
12. @Path("/books/{bookid}")
13. public Book getBook(@PathParam("bookid") String bookid) {// ... }
14. }
```

Se ejecuta con http://..../items

Se ejecuta con http://..../items/9999 y el parámetro «itemid» = 9999 (ejemplo)

Se ejecuta con http://..../items/books/

Se ejecuta con http://..../items/books/6542 y el parámetro «bookid» = 6542 (ejemplo)

Paso de Parámetros

- Como vimos anteriormente, se puede extraer información de la URL, a través de un parámetro.
- Pero hay diferentes formas de pasar parámetros a la ejecución de un método HTTP.
- JAX-RS provee las siguientes anotaciones para pasar parámetros :
  - @QueryParam
  - @MatrixParam
  - @CookieParam
  - @HeaderParam
  - @FormParam

Paso de Parámetros

- @PathParam permite extraer el valor de un template de URL.
  - Por ejemplo de la siguiente URL, http://www.myserver.com/customers/98342
    - extrae el valor 98342
- Ejemplo de uso
  - @Path("/customers")
  - public class CustomersResource {
  - @GET
  - public Customer getCustomer(@**PathParam**("customerid") customerid) {
    // ...}
  - }

Paso de Parámetros

- La anotación @QueryParam extrae el valor de los parámetros de una URL que se ubican luego del signo de preguntas de cierre (?)
- Ejemplo para extraer el parámetro “zip” de la siguiente url
  - http://www.myserver.com/customers?zip=19870 URI:
- Código Ejemplo
  - @Path("/customers")
  - public class CustomersResource {
  - @GET
  - public Customer getCustomerByZipCode**(@QueryParam("zip")** Long zip)
    …
- La conversión a Long la intenta automáticamente el framework.

Paso de Parámetros

- @CookieParam obtiene los valores de una Cookie,
- @HeaderParam obtiene información de encabezado.

```
1.  @Path("/products")
2.  public class ItemsResource {
3.  @GET
4.  public Book getBook(@CookieParam("sessionid") int sessionid) {
5.      // ...
6.  }
7.  }
```

Consumiendo y produciendo tipos MIME

- Con REST, un recurso puede tener varias representaciones
  - Un libro puede ser representado como una página web, datos XML, o una imagen que muestra la portada del libro.
- Las anotaciones @javax.ws.rs.Consumes y @javax.ws.rs.Produces se pueden aplicar a un recurso con varias representaciones posibles.
- En él se definen los tipos MIME de representación intercambiados entre cliente y el servidor.
- El uso de estas anotaciones en un método tiene primacía sobre anotaciones a nivel de clase para un argumento de método o tipo de retorno. (se sobreescriben)
- En ausencia de cualquiera de estas anotaciones, se supone que se brinda soporte para cualquier tipo de MIME (*/*).

Consumir y producir multiples representaciones

- Si un recurso genera más de una representación, se puede indicar en @Produces
  - @GET
  - @Path("{oid}")
  - @Produces({"application/xml", "application/json"})
  - **public** Order getOrder(@PathParam("oid")**int** id) { . . . }
- Si acepta consumer cualquier tipo
  - @POST
  - @Path("{oid}")
  - @Consumes({"application/xml", "application/json"})
  - **public** Order getOrder(@PathParam("oid")**int** id) { . . . }

- JAX-RS define constantes para los principales MediaType

APPLICATION_ATOM_XML	"application/atom+xml"
APPLICATION_FORM_URLENCODED	"application/x-www-form-urlencoded"
APPLICATION_JSON	"application/json"
APPLICATION_OCTET_STREAM	"application/octet-stream"
APPLICATION_SVG_XML	"application/svg+xml"
APPLICATION_XHTML_XML	"application/xhtml+xml"
APPLICATION_XML	"application/xml"
MULTIPART_FORM_DATA	"multipart/form-data"
TEXT_HTML	"text/html"
TEXT_PLAIN	"text/plain"
TEXT_XML	"text/xml"
WILDCARD	"*/*"

Ejemplo

```
1.  @Path("/customer")
2.  @Produces(MediaType.TEXT_PLAIN)
3.  public class CustomerRestService {
4.    @GET
5.    public Response getAsPlainText() {   // ...  }
6.  @GET
7.  @Produces(MediaType.TEXT_HTML)
8.  public Response getAsHtml() {  // ...   }
9.  @GET
10. @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
11. public Response getAsJsonAndXML() {  // ... }
12. @PUT
13. @Consumes(MediaType.TEXT_PLAIN)
14. public void putName(String customer) {
15. // ...
16. }
17. }
```

Tipos MIME de Respuesta (Produces)

- Según el ejemplo anterior, el recurso REST puede retornar más de un tipo MIME para la misma URL, y el mismo método HTTP.
- Aquel cuyo tipo MIME de retorno se ajusta más apropiadamente al que solicitó el cliente en el encabezado HTTP. Cuando el cliente hace una petición solicita uno o más tipos MIME y el que mejor se ajusta es el que se utiliza.
- Además en JavaEE7 se agrega el parámetro qs, que permite especificar nuestra preferencia en la negociación

```
1. @POST
2. @Path("{oid}")
3. @Produces({MediaType.APPLICATION_JSON, qs=0.75 MediaType.APPLICATION_XML})
4. public Response getAsJsonAndXML() {  // ... }
```

```
Accept: application/xml; q=0.75, application/json;
```

Tipos MIME de Respuesta (Produces)

- Si el request de un cliente es “Accept: text/plain” se le dará la representación en ese formato **(@Produces(MediaType.TEXT_PLAIN))**.
- Pero si el cliente puede interpretar 2 formatos, lo puede especificar en HTTP Header, con un valor de calidad de servicio asociado.
- Ejemplo:
  - Accept: text/plain; q=0.8, text/html
- En este encabezado indica que puede interpreter texto/plain y text/html, y prefiere este ultimo pero acepta el anterior si en el servidor está definida la calidad de html con un valor menor a 0.9