JAX

El analizador léxico es conocido como escáner, pues su principal funcionalidad es analizar los lexemas sobre algún patrón definido. Además, su principal funcionalidad es tomar la secuencia de caracteres o símbolos del alfabeto del lenguaje y ubicarlas dentro de las diferentes categorías comúnmente conocidas cono unidades léxicas, que son empleadas por el analizador gramatical para determinar que lo escrito en el programa fuente este correcto o no gramaticalmente, de aquí que algunas de estas unidades léxicas sean rechazadas o removidas por ser un comentario o un espacio en blanco.

¿Qué es el analizador léxico jax?

Jax es un compilador léxico creado en lenguaje Java, que genera un escáner a partir de expresiones regulares que existen por defecto en un archivo de java.

El jax se define como un modelo de programación que simplifica el desarrollo de las aplicaciones mediante el soporte de un modelo basado en las anotaciones estándares para el desarrollo clientes y aplicaciones de servicio web.

¿Cómo se utiliza el jax?

Es encargado de procesar las expresiones regulares en el cual se genera un ficher Java que pueda ser compilado por Java y así crear el escáner.

Los escáneres generados por Jax tienen entradas de búfer de tamaño arbitrario, y es al menos más conveniente para crear las tablas de tokens, Jax utiliza solo 7 bits de caracteres ASCII, y no permite código Unario.

También se puede decir que se usa como la implementación del estándar de programación jax proporciona mejoras para desarrollar clientes y servicios web.

La API de Java para servicios web XML (JAX-WS) simplifica la creación y el despliegue de servicios web y clientes de servicios web. Java SE incluye JAX-WS 2.2.

Compatibilidad con versiones anteriores:
Existen dos anomalías de compatibilidad de fuente pequeña entre las versiones 2.1 y 2.2 de JAX-WS. Los cambios en la API son los siguientes:
En la anotación FaultAction, el tipo del elemento className cambió de ser un simple java.lang.Class de vainilla a ser java.lang.Class <? Extiende java.lang.Exception>.
En la anotación WebServiceRef, el tipo del elemento de valor ahora es más específico. En lugar de ser java.lang.Class, el tipo ahora es java.lang.Class <? Extiende el servicio>.
Debido a estos cambios, algunas aplicaciones que compilan con Java SE 6

pueden no compilarse con Java SE 7. Habrá un error de implementación si se usan tipos incorrectos con la API JAX-WS 2.1. En Java SE 7, estos errores de tipo se detectan en tiempo de compilación en lugar de tiempo de implementación.

Ejemplo:

En el código a continuación, el proxy StockQuoteProvider inyectado debe tener habilitado WS-Addressing como se especifica en la anotación de direccionamiento.

```
clase pública MyClient {
  @Direccionamiento
  @WebServiceRef (StockQuoteService.class)
  StockQuoteProvider privado stockQuoteProvider stock;
  ...
}
```

Si una implementación de JAX-WS encuentra una anotación no admitida o no reconocida anotada con el WebServiceFeatureAnnotation que se especifica con WebServiceRef, DEBE darse un ERROR

Jax

Código generado: Java.

No soporta entornos, está basado en expresiones regulares.

No soporta Unicode.

jax tiene licencia bajo una licencia doble: CDDL 1.1 y GPL 2.0 con excepción de ruta de clase. Eso significa que puede elegir cuál de los dos se adapta mejor a sus necesidades y utilizarlo bajo esos términos.

Ejemplo:

En este ejemplo vamos a usar metros.

sólo nos queda escribir nuestro código. No tenemos más que escribir la clase que será nuestro *Web Service* con sus métodos, que serán los accesibles desde la Web. Para indicar que la clase es un *Web Service* sólo tendremos que ponerle la anotación *@WebService* y a los métodos que queramos que sea accesibles la anotación *@WebMethod*. La clase sería la siguiente

```
package com.chuidiang.ejemplos.jax_ws;

import javax.jws.WebMethod;
import javax.jws.WebService;
```

```
@WebService
public class UnWebService {
    @WebMethod
    public float suma(float a, float b, UnDato c) {
        return a + b;
    }
}
```

Podríamos compilar y generar un fichero .war con esta clase para desplegarla en un Tomcat o similar, pero podemos directamente ponerle un *main()* para hacerla ejecutable. Para hacer pública la clase *UnWebService* como *Web Service* podemos usar la clase *EndPoint* que viene con *jax-ws*. El código quedaría así

```
package com.chuidiang.ejemplos.jax_ws;

import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.xml.ws.Endpoint;

@WebService
public class UnWebService {
    @WebMethod
    public float suma(float a, float b, UnDato c) {
        return a + b;
    }

    public static void main(String[] args) {
        Endpoint.publish("http://localhost:8080/UnWebService", new
UnWebService());
    }
}
```

Ejemplo hacer un cliente java de un web service:

```
package com.chuidiang.ejemplos.metro;

/**
 * Ejemplo simple de cliente de web service. Requiere para
funcionar que el
 * servidor este arrancado.
 *
 * @author chuidiang
 */
public class Main {

   /**
    * @param args
    */
   public static void main(String[] args) {
      // Obtencion del cliente. Sólo funciona si en wsimport usamos
la URL
      // real del fichero wsdl, estilo
http://dominioreal/unwebservice?wsdl
      UnWebServiceService unWebServiceService = new
UnWebServiceService();
      UnWebService unWebService =
unWebServiceService.getUnWebServicePort();

      // Ya podemos usarlo
      System.out.println(unWebService.suma(11.1, 22.2));

   }
}
```

Simplemente instanciamos la clase UnWebServiceService generada por wsimport y le pedimos el getUnWebServicePort() para obtener el UnWebService. Esta clase tiene los mismos métodos que la clase UnWebService del servidor, pero no los implementa, sino que redirige las llamadas al web service.

Una vez obtenida esa clase, sólo tenemos que llamar a sus métodos de forma normal.

Si en vez de usar la URL real del fichero wsdl hubiésemos usado un fichero u otra URL (localhost, por ejemplo), debemos instanciar la clase UnWebServiceService pasando como parámetros en el constructor la URL real del web service.

```
package com.chuidiang.ejemplos.main;

import java.net.MalformedURLException;
import java.net.URL;

import javax.xml.namespace.QName;

import com.chuidiang.ejemplos.metro.UnWebService;
import com.chuidiang.ejemplos.metro.UnWebServiceService;

/**
 * Ejemplo simple de cliente de web service. Requiere para
funcionar que el
 * servidor este arrancado.
 *
 * @author chuidiang
 */
public class Main {

   /**
    * @param args
    * @throws MalformedURLException
    */
   public static void main(String[] args) throws
MalformedURLException {
      // Obtencion del cliente
      UnWebServiceService unWebServiceService = new
UnWebServiceService(
            new URL("http://dominioreal:8080/UnWebService?wsdl"),
// URL real del web service.
            new QName("http://metro.ejemplos.chuidiang.com/",
// copiado del código generado por wsimport
                  "UnWebServiceService"));
      UnWebService unWebService =
unWebServiceService.getUnWebServicePort();
```

```
      // Ya podemos usarlo
      System.out.println(unWebService.suma(11.1, 22.2));

   }
}
```

El segundo parámetro QName se puede copiar de dentro de la clase UnWebServiceService generada por wsimport, en la línea del constructor por defecto.

```
   public UnWebServiceService() {
      super(UNWEBSERVICESERVICE_WSDL_LOCATION, new QName(
            "http://metro.ejemplos.chuidiang.com/",
"UnWebServiceService"));
   }
```