Arquitectura Orientada a Servicios

Servicios Web

Edwin Salvador

12 de junio de 2015

Sesión 10

Contenido I

Contenido I

UDDI

Tecnologías básicas

- Permite localizar WS.

UDDI

Tecnologías básicas

- Permite localizar WS.
- Define la especificación para construir un directorio distribuido de WS.

UDDI

Tecnologías básicas

- Permite localizar WS.
- Define la especificación para construir un directorio distribuido de WS.
- Los datos se almacenan en XML.

UDDI

Tecnologías básicas

- Permite localizar WS.
- Define la especificación para construir un directorio distribuido de WS.
- Los datos se almacenan en XML.
- Almacenan 3 tipos de información:

UDDI

Tecnologías básicas

- Permite localizar WS.
- Define la especificación para construir un directorio distribuido de WS.
- Los datos se almacenan en XML.
- Almacenan 3 tipos de información:
  - **Páginas blancas** datos de organizaciones (dirección, contacto, etc)

UDDI

Tecnologías básicas

- Permite localizar WS.
- Define la especificación para construir un directorio distribuido de WS.
- Los datos se almacenan en XML.
- Almacenan 3 tipos de información:
  - **Páginas blancas** datos de organizaciones (dirección, contacto, etc)
  - **Páginas amarillas** Clasificación de las organizaciones (tipo de industria, zona geográfica, etc.)

UDDI

Tecnologías básicas

- Permite localizar WS.
- Define la especificación para construir un directorio distribuido de WS.
- Los datos se almacenan en XML.
- Almacenan 3 tipos de información:
  - **Páginas blancas** datos de organizaciones (dirección, contacto, etc)
  - **Páginas amarillas** Clasificación de las organizaciones (tipo de industria, zona geográfica, etc.)
  - **Páginas verdes** Información técnica sobre los servicios que ofrecen. Instrucciones para utilizar los servicios (WSDL).

UDDI

Tecnologías básicas

- Permite localizar WS.
- Define la especificación para construir un directorio distribuido de WS.
- Los datos se almacenan en XML.
- Almacenan 3 tipos de información:
  - **Páginas blancas** datos de organizaciones (dirección, contacto, etc)
  - **Páginas amarillas** Clasificación de las organizaciones (tipo de industria, zona geográfica, etc.)
  - **Páginas verdes** Información técnica sobre los servicios que ofrecen. Instrucciones para utilizar los servicios (WSDL).
  - Se puede buscar y publicar servicios a través de una API para trabajar con el registro.

UDDI

Tecnologías básicas

- Permite localizar WS.
- Define la especificación para construir un directorio distribuido de WS.
- Los datos se almacenan en XML.
- Almacenan 3 tipos de información:
  - **Páginas blancas** datos de organizaciones (dirección, contacto, etc)
  - **Páginas amarillas** Clasificación de las organizaciones (tipo de industria, zona geográfica, etc.)
  - **Páginas verdes** Información técnica sobre los servicios que ofrecen. Instrucciones para utilizar los servicios (WSDL).
  - Se puede buscar y publicar servicios a través de una API para trabajar con el registro.
  - Se utilizan mensajes SOAP para acceder al registro.

- Permite localizar WS.
- Define la especificación para construir un directorio distribuido de WS.
- Los datos se almacenan en XML.
- Almacenan 3 tipos de información:
  - **Páginas blancas** datos de organizaciones (dirección, contacto, etc)
  - **Páginas amarillas** Clasificación de las organizaciones (tipo de industria, zona geográfica, etc.)
  - **Páginas verdes** Información técnica sobre los servicios que ofrecen. Instrucciones para utilizar los servicios (WSDL).
  - Se puede buscar y publicar servicios a través de una API para trabajar con el registro.
  - Se utilizan mensajes SOAP para acceder al registro.
  - Especificación y documentación de UDDI en `http://uddi.xml.org/`.

UDDI

- **jUDDI** es un registro *open-source* de Apache.

UDDI

- **jUDDI** es un registro *open-source* de Apache.
- Se puede instalar en cualquier servidor (Tomcat) como una aplicación Java y una base de datos que se pueden instalar en cualquier DBMS (MySQL, Postgres, Oracle, etc.)

Contenido I

Interoperabilidad de los WS: Metro y JAX-WS

- Sun y Microsoft iniciaron proyecto para proporcionar WS que interoperen a través de diferentes plataformas.

Interoperabilidad de los WS: Metro y JAX-WS

- Sun y Microsoft iniciaron proyecto para proporcionar WS que interoperen a través de diferentes plataformas.
- WCF (Windows Communicaction Foundation) es el producto de Microsoft para .NET

Interoperabilidad de los WS: Metro y JAX-WS

- Sun y Microsoft iniciaron proyecto para proporcionar WS que interoperen a través de diferentes plataformas.
- WCF (Windows Communicaction Foundation) es el producto de Microsoft para .NET
- Metro `http://metro.java.net` es el producto de Sun para Java.

Interoperabilidad de los WS: Metro y JAX-WS

- Sun y Microsoft iniciaron proyecto para proporcionar WS que interoperen a través de diferentes plataformas.
- WCF (Windows Communicaction Foundation) es el producto de Microsoft para .NET
- Metro `http://metro.java.net` es el producto de Sun para Java.
- Metro está formada por tres componentes principales:

Interoperabilidad de los WS: Metro y JAX-WS

- Sun y Microsoft iniciaron proyecto para proporcionar WS que interoperen a través de diferentes plataformas.
- WCF (Windows Communicaction Foundation) es el producto de Microsoft para .NET
- Metro `http://metro.java.net` es el producto de Sun para Java.
- Metro está formada por tres componentes principales:
  - **Metro/WSIT 2.1.1**

Interoperabilidad de los WS: Metro y JAX-WS

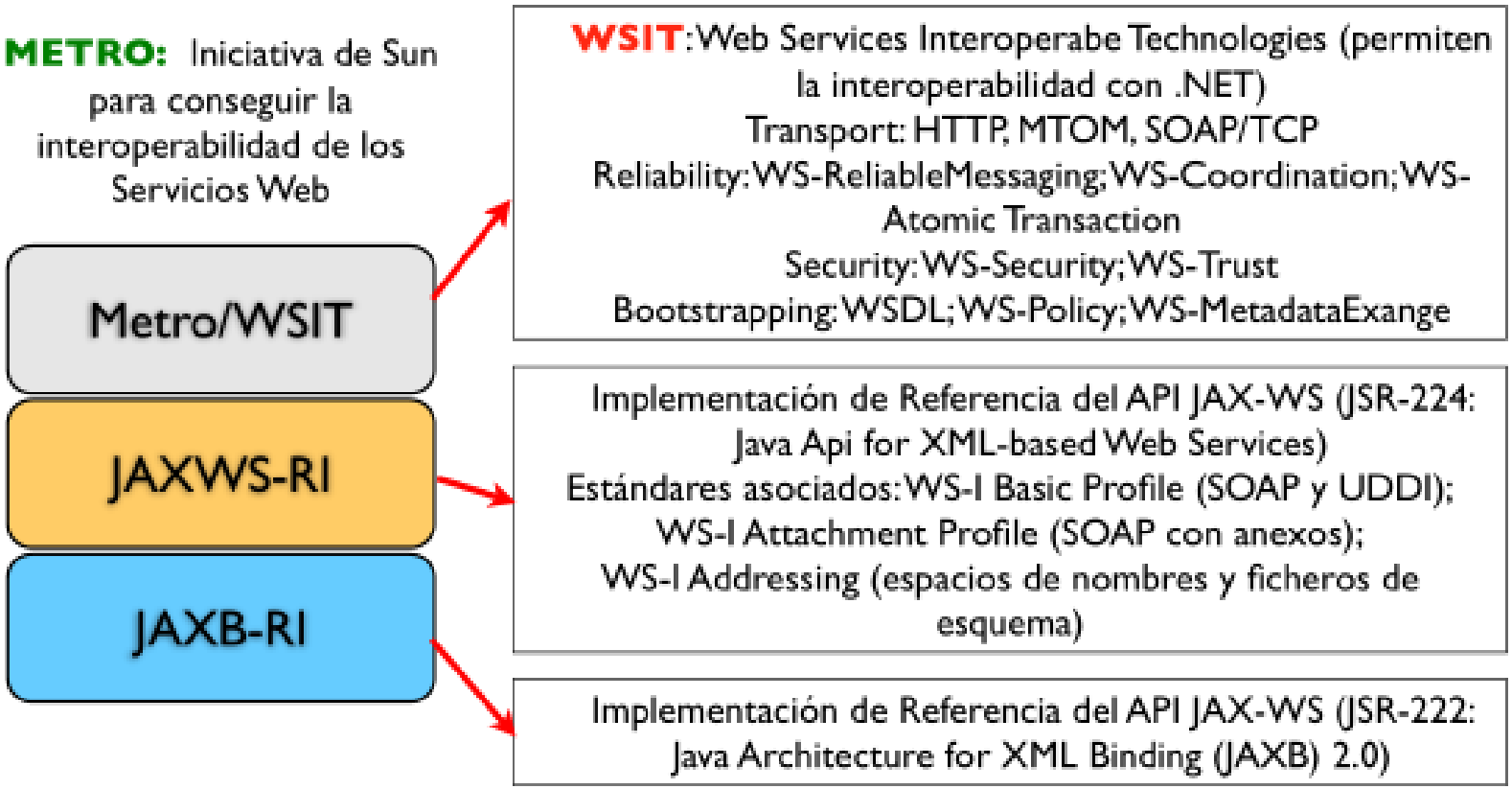
- Sun y Microsoft iniciaron proyecto para proporcionar WS que interoperen a través de diferentes plataformas.
- WCF (Windows Communicaction Foundation) es el producto de Microsoft para .NET
- Metro `http://metro.java.net` es el producto de Sun para Java.
- Metro está formada por tres componentes principales:
  - **Metro/WSIT 2.1.1**
  - JAX-WS RI 2.2.5

Interoperabilidad de los WS: Metro y JAX-WS

- Sun y Microsoft iniciaron proyecto para proporcionar WS que interoperen a través de diferentes plataformas.
- WCF (Windows Communicaction Foundation) es el producto de Microsoft para .NET
- Metro `http://metro.java.net` es el producto de Sun para Java.
- Metro está formada por tres componentes principales:
  - **Metro/WSIT 2.1.1**
  - JAX-WS RI 2.2.5
  - JAXB RI 2.2.4-1

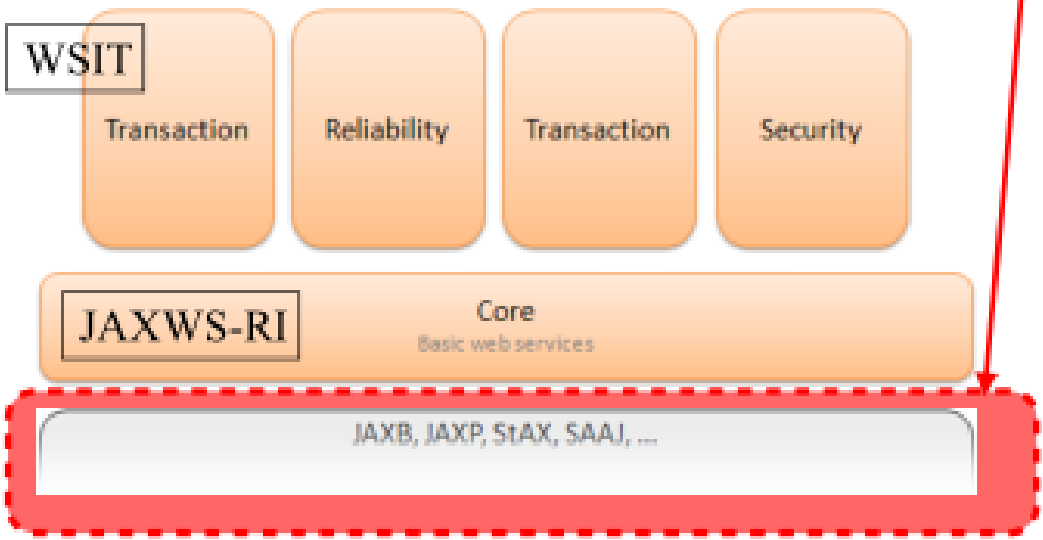
Interoperabilidad de los servicios web (WS-I)

Metro y JAX-WS



Metro

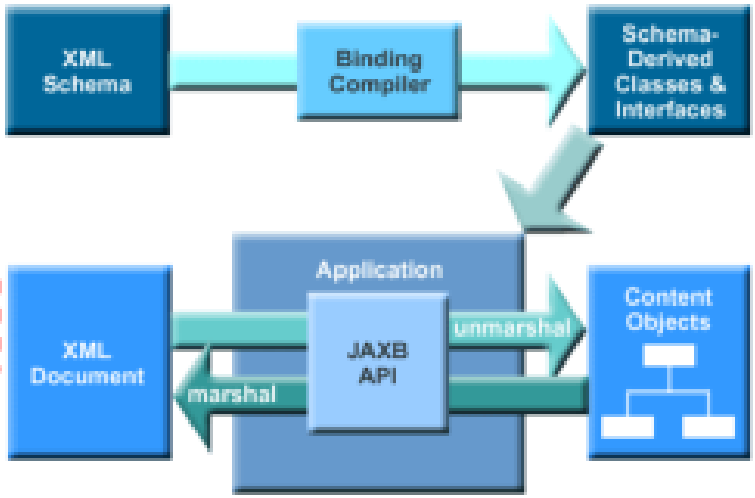
METRO: Iniciativa de Sun para conseguir la interoperabilidad de los Servicios Web



Metro está construido sobre un conjunto de librerías que pueden usarse de forma independiente fuera del contexto de los Servicios Web

JAXB-RI

Implementación de referencia del API **JAXB** (Java Architecture for XML Binding)



Metro

- Con Metro la invocación de una operación de un WS se representa con un protocolo basado en XML.

Metro

- Con Metro la invocación de una operación de un WS se representa con un protocolo basado en XML.
- La especificación SOAP define: *envelope* del mensaje, reglas de modificación y convención para representar invocaciones y respuestas del WS.

Metro

- Con Metro la invocación de una operación de un WS se representa con un protocolo basado en XML.
- La especificación SOAP define: *envelope* del mensaje, reglas de modificación y convención para representar invocaciones y respuestas del WS.
- Las llamadas y respuestas son enviadas como mensajes SOAP (XML) por HTTP.

Metro

- Con Metro la invocación de una operación de un WS se representa con un protocolo basado en XML.
- La especificación SOAP define: *envelope* del mensaje, reglas de modificación y convención para representar invocaciones y respuestas del WS.
- Las llamadas y respuestas son enviadas como mensajes SOAP (XML) por HTTP.
- La API JAX-WS oculta la complejidad del los mensajes SOAP al desarrollador de la aplicación.

Comunicación entre un WS y un cliente mediante JAX-WS

- **En el lado del servidor**, el desarrollador especifica las operaciones del WS definiendo métodos en una interfaz y una o más clases que implementan dichos métodos escrita en Java.

Comunicación entre un WS y un cliente mediante JAX-WS

- **En el lado del servidor**, el desarrollador especifica las operaciones del WS definiendo métodos en una interfaz y una o más clases que implementan dichos métodos escrita en Java.
- **En el lado del cliente** se crea un proxy (objeto local que representa el servicio) y se invoca los métodos sobre el proxy.

Comunicación entre un WS y un cliente mediante JAX-WS

- **En el lado del servidor**, el desarrollador especifica las operaciones del WS definiendo métodos en una interfaz y una o más clases que implementan dichos métodos escrita en Java.
- **En el lado del cliente** se crea un proxy (objeto local que representa el servicio) y se invoca los métodos sobre el proxy.
- JAX-WS se encarga de parsear los mensajes SOAP.

Comunicación entre un WS y un cliente mediante JAX-WS

- **En el lado del servidor**, el desarrollador especifica las operaciones del WS definiendo métodos en una interfaz y una o más clases que implementan dichos métodos escrita en Java.
- **En el lado del cliente** se crea un proxy (objeto local que representa el servicio) y se invoca los métodos sobre el proxy.
- JAX-WS se encarga de parsear los mensajes SOAP.

- Metro y JAX-WS vienen incluidos en Glassfish por lo que lo utilizaremos para implementar los WS.

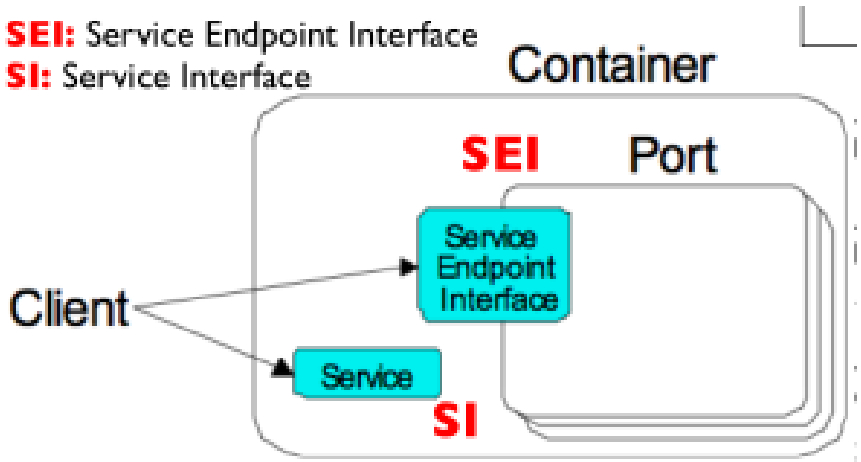
Contenido I

WS desde la vista del cliente

- Un cliente de un WS puede ser otro WS, un componente web o EJB o una aplicación Java o no Java.

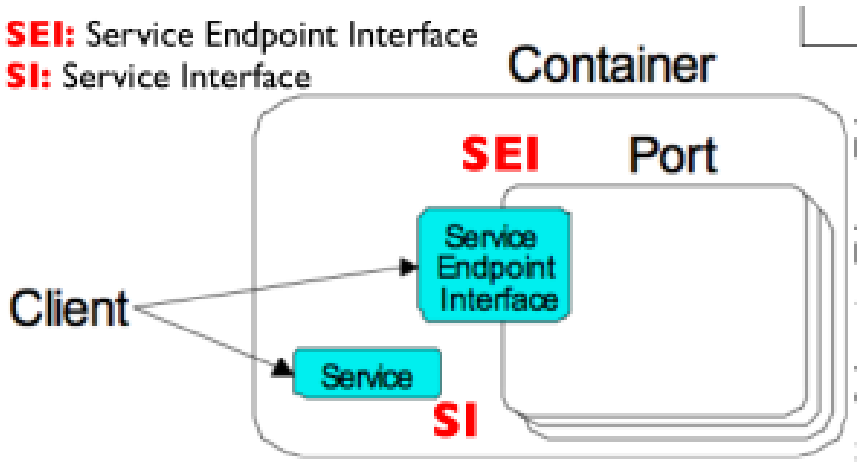
WS desde la vista del cliente

- Un cliente de un WS puede ser otro WS, un componente web o EJB o una aplicación Java o no Java.
- El cliente puede ser remoto y se proporciona una total transferencia al respecto.



WS desde la vista del cliente

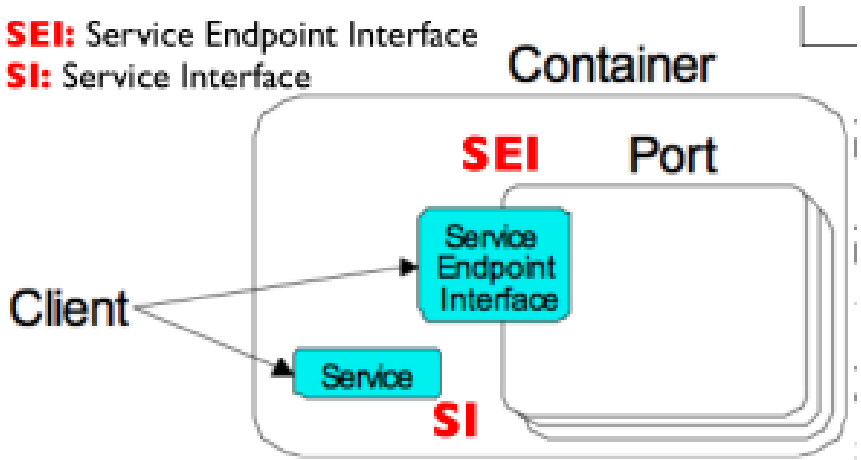
- Un cliente de un WS puede ser otro WS, un componente web o EJB o una aplicación Java o no Java.
- El cliente puede ser remoto y se proporciona una total transferencia al respecto.



  - clase/Interfaz **Service** (SI) define los métodos que un cliente puede utilizar para acceder a un **Port** de un WS.

WS desde la vista del cliente

- Un cliente de un WS puede ser otro WS, un componente web o EJB o una aplicación Java o no Java.
- El cliente puede ser remoto y se proporciona una total transferencia al respecto.



  - clase/Interfaz **Service** (SI) define los métodos que un cliente puede utilizar para acceder a un **Port** de un WS.
  - El cliente accede a una implementación de WS mediante el SEI (especificado por el proveedor). Se puede realizar llamadas a métodos del SEI del Port correspondiente.

WS desde la vista del cliente

- JAX-WS define un modelo de programación donde se mapea un WSDL a Java.

WS desde la vista del cliente

- JAX-WS define un modelo de programación donde se mapea un WSDL a Java.
- Este mapeado se utiliza la seleccionar el Port del servicio que se desea utilizar.

WS desde la vista del cliente

- JAX-WS define un modelo de programación donde se mapea un WSDL a Java.
- Este mapeado se utiliza la seleccionar el Port del servicio que se desea utilizar.
- **wsimport** es la herramienta que proporciona las clases necesarias en el cliente para acceder al WS.

WS desde la vista del cliente

- JAX-WS define un modelo de programación donde se mapea un WSDL a Java.
- Este mapeado se utiliza la seleccionar el Port del servicio que se desea utilizar.
- **wsimport** es la herramienta que proporciona las clases necesarias en el cliente para acceder al WS.
- El cliente solo necesita realizar llamadas sobre la interfaz del *endpoint* del servicio utilizando el PortType correspondiente para acceder al servicio.

Contenido I

Tipos de acceso para invocar JAX-WS

Dos formas de invocar un WS utilizando JAX-WS:

- **Creación de un stub estático**

Tipos de acceso para invocar JAX-WS

Dos formas de invocar un WS utilizando JAX-WS:

- **Creación de un stub estático**
  - Se genera una capa *stub* por debajo del cliente.

Tipos de acceso para invocar JAX-WS

Dos formas de invocar un WS utilizando JAX-WS:

- **Creación de un stub estático**
  - Se genera una capa *stub* por debajo del cliente.
  - Este stub implementará la misma interfaz que el servicio. Esto permitirá acceder al WS desde el cliente.

Tipos de acceso para invocar JAX-WS

Dos formas de invocar un WS utilizando JAX-WS:

- **Creación de un stub estático**
  - Se genera una capa *stub* por debajo del cliente.
  - Este stub implementará la misma interfaz que el servicio. Esto permitirá acceder al WS desde el cliente.
  - Se recomienda contar con herramientas que generen el stub de manera automática.

Tipos de acceso para invocar JAX-WS

Dos formas de invocar un WS utilizando JAX-WS:

- **Creación de un stub estático**
  - Se genera una capa *stub* por debajo del cliente.
  - Este stub implementará la misma interfaz que el servicio. Esto permitirá acceder al WS desde el cliente.
  - Se recomienda contar con herramientas que generen el stub de manera automática.
- **Utilización de la Interfaz de Invocación Dinámica (DII)**

Tipos de acceso para invocar JAX-WS

Dos formas de invocar un WS utilizando JAX-WS:

- **Creación de un stub estático**
  - Se genera una capa *stub* por debajo del cliente.
  - Este stub implementará la misma interfaz que el servicio. Esto permitirá acceder al WS desde el cliente.
  - Se recomienda contar con herramientas que generen el stub de manera automática.
- **Utilización de la Interfaz de Invocación Dinámica (DII)**
  - Permite hacer llamadas a procedimientos del WS dinámicamente sin crear un *stub*.

Tipos de acceso para invocar JAX-WS

Dos formas de invocar un WS utilizando JAX-WS:

- **Creación de un stub estático**
  - Se genera una capa *stub* por debajo del cliente.
  - Este stub implementará la misma interfaz que el servicio. Esto permitirá acceder al WS desde el cliente.
  - Se recomienda contar con herramientas que generen el stub de manera automática.
- **Utilización de la Interfaz de Invocación Dinámica (DII)**
  - Permite hacer llamadas a procedimientos del WS dinámicamente sin crear un *stub*.
  - Se utiliza cuando no se conoce la interfaz del WS con anticipación.

Tipos de acceso para invocar JAX-WS

Dos formas de invocar un WS utilizando JAX-WS:

- **Creación de un stub estático**
  - Se genera una capa *stub* por debajo del cliente.
  - Este stub implementará la misma interfaz que el servicio. Esto permitirá acceder al WS desde el cliente.
  - Se recomienda contar con herramientas que generen el stub de manera automática.
- **Utilización de la Interfaz de Invocación Dinámica (DII)**
  - Permite hacer llamadas a procedimientos del WS dinámicamente sin crear un *stub*.
  - Se utiliza cuando no se conoce la interfaz del WS con anticipación.
  - Se proporciona solo los nombre de los métodos a invocar mediante una cadena de texto.

Tipos de acceso para invocar JAX-WS

Dos formas de invocar un WS utilizando JAX-WS:

- **Creación de un stub estático**
  - Se genera una capa *stub* por debajo del cliente.
  - Este stub implementará la misma interfaz que el servicio. Esto permitirá acceder al WS desde el cliente.
  - Se recomienda contar con herramientas que generen el stub de manera automática.
- **Utilización de la Interfaz de Invocación Dinámica (DII)**
  - Permite hacer llamadas a procedimientos del WS dinámicamente sin crear un *stub*.
  - Se utiliza cuando no se conoce la interfaz del WS con anticipación.
  - Se proporciona solo los nombre de los métodos a invocar mediante una cadena de texto.
  - Podemos utilizar esta interfaz dinámica aunque no contemos con un WDSL que nos proporcione información sobre nuestro servicio. Debemos proporcionar esta información manualmente.

Creación de un cliente web con JAX-WS

Utilizando un stub estático

Pasos:

- Codificar clase cliente, implementando el acceso al servicio web utilizando las interfaces de los stubs que se generarán con **wsimport**.

Creación de un cliente web con JAX-WS

Utilizando un stub estático

Pasos:

- Codificar clase cliente, implementando el acceso al servicio web utilizando las interfaces de los stubs que se generarán con **wsimport**.
- Generar los artefactos necesarios del servicio web para poder conectar con dicho servicio web desde el cliente (mediante wsimport)

Creación de un cliente web con JAX-WS

Utilizando un stub estático

Pasos:

- Codificar clase cliente, implementando el acceso al servicio web utilizando las interfaces de los stubs que se generarán con **wsimport**.
- Generar los artefactos necesarios del servicio web para poder conectar con dicho servicio web desde el cliente (mediante wsimport)
- Compilar la clase cliente (empaquetar y desplegar si es necesario)

Creación de un cliente web con JAX-WS

Utilizando un stub estático

Pasos:

- Codificar clase cliente, implementando el acceso al servicio web utilizando las interfaces de los stubs que se generarán con **wsimport**.
- Generar los artefactos necesarios del servicio web para poder conectar con dicho servicio web desde el cliente (mediante wsimport)
- Compilar la clase cliente (empaquetar y desplegar si es necesario)
- Ejecutar el cliente.

Desplegar el WS en Glassfish

- En Netbeans, en la ventana Services-> Servers -> GlassFish Server 4.1, clic derecho -> Start.

Desplegar el WS en Glassfish

- En Netbeans, en la ventana Services-> Servers -> GlassFish Server 4.1, clic derecho -> Start.
- En Chrome, ir a http://localhost:8080

Desplegar el WS en Glassfish

- En Netbeans, en la ventana Services-> Servers -> GlassFish Server 4.1, clic derecho -> Start.
- En Chrome, ir a http://localhost:8080
- Vamos a `http://localhost:4848`, Desplegar una aplicación y cargamos el archivo HolaMundo.war

Desplegar el WS en Glassfish

- En Netbeans, en la ventana Services-> Servers -> GlassFish Server 4.1, clic derecho -> Start.
- En Chrome, ir a http://localhost:8080
- Vamos a `http://localhost:4848`, Desplegar una aplicación y cargamos el archivo HolaMundo.war
- Vamos a `http://localhost:8080/HolaMundo/hola?wsdl`

Desplegar el WS en Glassfish

- En Netbeans, en la ventana Services-> Servers -> GlassFish Server 4.1, clic derecho -> Start.
- En Chrome, ir a http://localhost:8080
- Vamos a `http://localhost:4848`, Desplegar una aplicación y cargamos el archivo HolaMundo.war
- Vamos a `http://localhost:8080/HolaMundo/hola?wsdl`
- Vamos a `http://localhost:8080/HolaMundo/hola?Tester`

Fichero `hola.wsdl`

- ¿Cuál es la parte abstracta del WSDL?

Fichero `hola.wsdl`

- ¿Cuál es la parte abstracta del WSDL?
- ¿Qué etiqueta nos indica las operaciones que realiza el servicio?

Fichero `hola.wsdl`

- ¿Cuál es la parte abstracta del WSDL?
- ¿Qué etiqueta nos indica las operaciones que realiza el servicio?
- ¿Qué operaciones realiza el servicio?

Fichero `hola.wsdl`

- ¿Cuál es la parte abstracta del WSDL?
- ¿Qué etiqueta nos indica las operaciones que realiza el servicio?
- ¿Qué operaciones realiza el servicio?
- ¿Qué realiza esta operación?

Fichero `hola.wsdl`

- ¿Cuál es la parte abstracta del WSDL?
- ¿Qué etiqueta nos indica las operaciones que realiza el servicio?
- ¿Qué operaciones realiza el servicio?
- ¿Qué realiza esta operación?
- ¿De qué tipo son los mensajes de entrada y salida de esta operación? ¿Dónde están definidos estos tipos?

Fichero `hola.wsdl`

- ¿Cuál es la parte abstracta del WSDL?
- ¿Qué etiqueta nos indica las operaciones que realiza el servicio?
- ¿Qué operaciones realiza el servicio?
- ¿Qué realiza esta operación?
- ¿De qué tipo son los mensajes de entrada y salida de esta operación? ¿Dónde están definidos estos tipos?
- ¿Qué contiene el XSD?

Fichero `hola.wsdl`

- ¿Cuál es la parte abstracta del WSDL?
- ¿Qué etiqueta nos indica las operaciones que realiza el servicio?
- ¿Qué operaciones realiza el servicio?
- ¿Qué realiza esta operación?
- ¿De qué tipo son los mensajes de entrada y salida de esta operación? ¿Dónde están definidos estos tipos?
- ¿Qué contiene el XSD?
- ¿Cuál es la definición concreta del WS?

Fichero `hola.wsdl`

- ¿Cuál es la parte abstracta del WSDL?
- ¿Qué etiqueta nos indica las operaciones que realiza el servicio?
- ¿Qué operaciones realiza el servicio?
- ¿Qué realiza esta operación?
- ¿De qué tipo son los mensajes de entrada y salida de esta operación? ¿Dónde están definidos estos tipos?
- ¿Qué contiene el XSD?
- ¿Cuál es la definición concreta del WS?
- ¿Cuál es el nombre del Port?

Fichero `hola.wsdl`

- ¿Cuál es la parte abstracta del WSDL?
- ¿Qué etiqueta nos indica las operaciones que realiza el servicio?
- ¿Qué operaciones realiza el servicio?
- ¿Qué realiza esta operación?
- ¿De qué tipo son los mensajes de entrada y salida de esta operación? ¿Dónde están definidos estos tipos?
- ¿Qué contiene el XSD?
- ¿Cuál es la definición concreta del WS?
- ¿Cuál es el nombre del Port?
- ¿Cómo accedemos al componente `Port` de nuestro WS?

Invocación de servicios web JAX-WS: comando wsimport

- A partir de JDK se incluye en Java SE la JAX-WS y herramientas para crear WS.

Invocación de servicios web JAX-WS: comando wsimport

- A partir de JDK se incluye en Java SE la JAX-WS y herramientas para crear WS.
- Las clases de la API de JAX-WS están en el paquete javax.xml.ws

Invocación de servicios web JAX-WS: comando wsimport

- A partir de JDK se incluye en Java SE la JAX-WS y herramientas para crear WS.
- Las clases de la API de JAX-WS están en el paquete javax.xml.ws
- Utilizaremos **wsimport** para crear WS. Este tomará como entrada el WSDL del servicio al que queremos acceder y producirá un conjunto de clases Java que nos permitirán acceder al servicio.

Invocación de servicios web JAX-WS: comando wsimport

- A partir de JDK se incluye en Java SE la JAX-WS y herramientas para crear WS.
- Las clases de la API de JAX-WS están en el paquete javax.xml.ws
- Utilizaremos **wsimport** para crear WS. Este tomará como entrada el WSDL del servicio al que queremos acceder y producirá un conjunto de clases Java que nos permitirán acceder al servicio.
- Corremos en la linea de comandos:
  `wsimport -s` <*src.dir*> `-d` <*dest.dir*> `-p` <*pkg*> <*wsdl.uri*>

Invocación de servicios web JAX-WS: comando wsimport

- A partir de JDK se incluye en Java SE la JAX-WS y herramientas para crear WS.
- Las clases de la API de JAX-WS están en el paquete javax.xml.ws
- Utilizaremos **wsimport** para crear WS. Este tomará como entrada el WSDL del servicio al que queremos acceder y producirá un conjunto de clases Java que nos permitirán acceder al servicio.
- Corremos en la linea de comandos:
  `wsimport -s` <*src.dir*> `-d` <*dest.dir*> `-p` <*pkg*> <*wsdl.uri*>
- `wsimport -help`: nos indica que es cada parámetro.

Ejemplo

- Utilizamos el `wsdl` que generamos la clase anterior hello.wsdl (en el repositorio) para crear nuestro cliente del servicio web.

Ejemplo

- Utilizamos el `wsdl` que generamos la clase anterior hello.wsdl (en el repositorio) para crear nuestro cliente del servicio web.
- Creamos un directorio con la siguiente estructura:

Ejemplo

- Utilizamos el `wsdl` que generamos la clase anterior hello.wsdl (en el repositorio) para crear nuestro cliente del servicio web.
- Creamos un directorio con la siguiente estructura:
  - src

Ejemplo

- Utilizamos el `wsdl` que generamos la clase anterior hello.wsdl (en el repositorio) para crear nuestro cliente del servicio web.
- Creamos un directorio con la siguiente estructura:
  - src
  - bin

Ejemplo

- Utilizamos el `wsdl` que generamos la clase anterior hello.wsdl (en el repositorio) para crear nuestro cliente del servicio web.
- Creamos un directorio con la siguiente estructura:
  - src
  - bin
- Ejecutamos en el `cmd`:

```
wsimport -s src -d bin -p soa.ws.hello.stub
http://localhost:8080/HolaMundo/hola?WSDL
```

Ejemplo

- Utilizamos el `wsdl` que generamos la clase anterior hello.wsdl (en el repositorio) para crear nuestro cliente del servicio web.
- Creamos un directorio con la siguiente estructura:
  - src
  - bin
- Ejecutamos en el `cmd`:
  `wsimport -s src -d bin -p soa.ws.hello.stub http://localhost:8080/HolaMundo/hola?WSDL`
- Se generarán varias clases que nos permitirán acceder al WS, invocar sus operaciones desde nuestro cliente.

Ejemplo

- Utilizamos el `wsdl` que generamos la clase anterior hello.wsdl (en el repositorio) para crear nuestro cliente del servicio web.
- Creamos un directorio con la siguiente estructura:
  - src
  - bin
- Ejecutamos en el `cmd`:
  `wsimport -s src -d bin -p soa.ws.hello.stub http://localhost:8080/HolaMundo/hola?WSDL`
- Se generarán varias clases que nos permitirán acceder al WS, invocar sus operaciones desde nuestro cliente.
- Para obtener el *stub* para acceder a `Port` del servicio debemos invocar a la clase generada con el mismo nombre del servicio "HelloService.java" la cual extiende la clase `Service`.

Ejemplo de cliente de WS

HelloWorld.java

```
package soa.ws.hello.stub;

public class HelloWorld {
  public static void main(String[] args){
    HelloService service = new HelloService();
    Hello port = service.getHelloPort();
    System.out.println("Resultado: " + port.sayHello("Hola Mundo"));
  }
```

Invocación de servicios web JAX-WS desde una clase Java con Maven

- En Netbeans: Nuevo Proyecto->Maven->Java Application.

Invocación de servicios web JAX-WS desde una clase Java con Maven

- En Netbeans: Nuevo Proyecto->Maven->Java Application.
- **Project name**: HolaMundoJavaClient, **Group Id**: soaesfot, **Package**: soaesfot.holamundojavaclient

Invocación de servicios web JAX-WS desde una clase Java con Maven

- En Netbeans: Nuevo Proyecto->Maven->Java Application.
- **Project name**: HolaMundoJavaClient, **Group Id**: soaesfot, **Package**: soaesfot.holamundojavaclient
- Project Files->pom.xml

pom.xml

Agregamos:

- `<name>HolaMundoJavaClient</name>`

pom.xml

Agregamos:

- <name>HolaMundoJavaClient</name>

- ```
  <dependencies>
      <dependency>
          <groupId>com.sun.xml.ws</groupId>
          <artifactId>webservices-rt</artifactId>
          <version>1.4</version>
          <scope>compile</scope>
      </dependency>
  </dependencies>

  <build>
    <plugins>

    </plugins>
  </build>
  ```
  ```

Descripción de los plugins a utilizar

- **jaxws-maven-plugin**: para ejecutar la utilidad `wsimport` de JAX-WS y generar los stubs del servicio web.

Descripción de los plugins a utilizar

- **jaxws-maven-plugin**: para ejecutar la utilidad `wsimport` de JAX-WS y generar los stubs del servicio web.
  - Vamos a configurar la ejecución de la goal `wsimport` de forma que, a partir del wsdl situado en: `http://localhost:8080/HolaMundo/hola?WSDL`, genere los stubs necesarios, que por defecto se almacenan en el directorio `target/jaxws/wsimport/java`.

Descripción de los plugins a utilizar

- **jaxws-maven-plugin**: para ejecutar la utilidad `wsimport` de JAX-WS y generar los stubs del servicio web.
  - Vamos a configurar la ejecución de la goal `wsimport` de forma que, a partir del wsdl situado en: `http://localhost:8080/HolaMundo/hola?WSDL`, genere los stubs necesarios, que por defecto se almacenan en el directorio `target/jaxws/wsimport/java`.
  - Por defecto está asociado a la fase generate-sources del ciclo de vida de Maven.

Descripción de los plugins a utilizar

- **jaxws-maven-plugin**: para ejecutar la utilidad `wsimport` de JAX-WS y generar los stubs del servicio web.
  - Vamos a configurar la ejecución de la goal `wsimport` de forma que, a partir del wsdl situado en: `http://localhost:8080/HolaMundo/hola?WSDL`, genere los stubs necesarios, que por defecto se almacenan en el directorio `target/jaxws/wsimport/java`.
  - Por defecto está asociado a la fase generate-sources del ciclo de vida de Maven.
  - Wsimport se ejecutará, antes de compilar los fuentes del proyecto.

Descripción de los plugins a utilizar

- **jaxws-maven-plugin**: para ejecutar la utilidad `wsimport` de JAX-WS y generar los stubs del servicio web.
  - Vamos a configurar la ejecución de la goal `wsimport` de forma que, a partir del wsdl situado en: `http://localhost:8080/HolaMundo/hola?WSDL`, genere los stubs necesarios, que por defecto se almacenan en el directorio `target/jaxws/wsimport/java`.
  - Por defecto está asociado a la fase generate-sources del ciclo de vida de Maven.
  - Wsimport se ejecutará, antes de compilar los fuentes del proyecto.
- **exec-maven-plugin**: para ejecutar la aplicación java desde maven

pom.xml

Dentro de <plugins> añadir:

```
<plugin>
  <groupId>org.jvnet.jax-ws-commons</groupId>
  <artifactId>jaxws-maven-plugin</artifactId>
  <version>2.2</version>
  <executions>
      <execution>
          <goals>
              <goal>wsimport</goal>
          </goals>
      </execution>
  </executions>
```

pom.xml

```
  <configuration>
      <wsdlUrls>
          <wsdlUrl>
              http://localhost:8080/HolaMundo/hola?WSDL
          </wsdlUrl>
      </wsdlUrls>
      <verbose>true</verbose>
  </configuration>
  <dependencies>
      <dependency>
          <groupId>javax.xml</groupId>
          <artifactId>webservices-api</artifactId>
          <version>1.4</version>
      </dependency>
  </dependencies>
</plugin>
```

pom.xml

```
<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>exec-maven-plugin</artifactId>
    <version>1.2.1</version>
    <executions>
        <execution>
            <goals>
                <goal>java</goal>
            </goals>
        </execution>
    </executions>
    <configuration>
        <mainClass>soaesfot.holamundojavaclient.App</mainClass>
    </configuration>
</plugin>
```

GUARDAMOS EL FICHERO

Compilar proyecto Maven

- Antes añadimos a la variable de entorno `PATH` el valor `C:\Program Files\NetBeans 8.0.2\java\maven\bin`. Si esta opción no sirve se debe descargar maven desde `https://maven.apache.org/download.cgi`

Compilar proyecto Maven

- Antes añadimos a la variable de entorno `PATH` el valor `C:\Program Files\NetBeans 8.0.2\java\maven\bin`. Si esta opción no sirve se debe descargar maven desde `https://maven.apache.org/download.cgi`
- En las variables de usuario nos aseguramos que exista la variable `JAVA_HOME` caso contrario la añadimos con el valor `C:\Program Files\Java\jdk1.7.0_17`

Compilar proyecto Maven

- Antes añadimos a la variable de entorno `PATH` el valor `C:\Program Files\NetBeans 8.0.2\java\maven\bin`. Si esta opción no sirve se debe descargar maven desde `https://maven.apache.org/download.cgi`
- En las variables de usuario nos aseguramos que exista la variable `JAVA_HOME` caso contrario la añadimos con el valor `C:\Program Files\Java\jdk1.7.0_17`
- Abrimos el `cmd` y ejecutamos `mvn --version`

Compilar proyecto Maven

- Antes añadimos a la variable de entorno `PATH` el valor `C:\Program Files\NetBeans 8.0.2\java\maven\bin`. Si esta opción no sirve se debe descargar maven desde `https://maven.apache.org/download.cgi`
- En las variables de usuario nos aseguramos que exista la variable `JAVA_HOME` caso contrario la añadimos con el valor `C:\Program Files\Java\jdk1.7.0_17`
- Abrimos el `cmd` y ejecutamos `mvn --version`
- Ejecutamos: `cd` *(carpeta de nuestro proyecto maven de netbeans)*. Clic derecho ->propiedades->Source.

Compilar proyecto Maven

- Antes añadimos a la variable de entorno `PATH` el valor `C:\Program Files\NetBeans 8.0.2\java\maven\bin`. Si esta opción no sirve se debe descargar maven desde `https://maven.apache.org/download.cgi`
- En las variables de usuario nos aseguramos que exista la variable `JAVA_HOME` caso contrario la añadimos con el valor `C:\Program Files\Java\jdk1.7.0_17`
- Abrimos el `cmd` y ejecutamos `mvn --version`
- Ejecutamos: `cd` *(carpeta de nuestro proyecto maven de netbeans)*. Clic derecho ->propiedades->Source.
- En Netbeans: clic derecho en el proyecto-> Clean and Build

Compilar proyecto Maven

- Antes añadimos a la variable de entorno `PATH` el valor `C:\Program Files\NetBeans 8.0.2\java\maven\bin`. Si esta opción no sirve se debe descargar maven desde `https://maven.apache.org/download.cgi`
- En las variables de usuario nos aseguramos que exista la variable `JAVA_HOME` caso contrario la añadimos con el valor `C:\Program Files\Java\jdk1.7.0_17`
- Abrimos el `cmd` y ejecutamos `mvn --version`
- Ejecutamos: `cd` *(carpeta de nuestro proyecto maven de netbeans)*. Clic derecho ->propiedades->Source.
- En Netbeans: clic derecho en el proyecto-> Clean and Build
- En Netbeans: Proyectos->HolaMundoJavaClient->Source Packages->soaesfot->holamundojavaclient->clic derecho->new->Java class. **Name**: App.

Ficheros creados al compilar

- **Hola.java**: contiene la Interfaz del servicio (SEI del servicio)

Ficheros creados al compilar

- **Hola.java**: contiene la Interfaz del servicio (SEI del servicio)
- **Hola_Service.java**: contiene la Clase heredada de Service, que utilizaremos para acceder al componente Port de nuestro servicio web a través del SEI. Para ello debemos utilizar el método "Hola getHolaPort()" de la clase Hola_Service

Ficheros creados al compilar

- **Hola.java**: contiene la Interfaz del servicio (SEI del servicio)
- **Hola_Service.java**: contiene la Clase heredada de Service, que utilizaremos para acceder al componente Port de nuestro servicio web a través del SEI. Para ello debemos utilizar el método "Hola getHolaPort()" de la clase Hola_Service
- **ObjectFactory.java**: Contiene una factoria de métodos para recuperar representaciones java a partir de definiciones XML

Ficheros creados al compilar

- **Hola.java**: contiene la Interfaz del servicio (SEI del servicio)
- **Hola_Service.java**: contiene la Clase heredada de Service, que utilizaremos para acceder al componente Port de nuestro servicio web a través del SEI. Para ello debemos utilizar el método "Hola getHolaPort()" de la clase Hola_Service
- **ObjectFactory.java**: Contiene una factoria de métodos para recuperar representaciones java a partir de definiciones XML
- **Hello.java, HelloResponse.java**: son las clases que representan mensajes de nuestro WSDL

Ficheros creados al compilar

- **Hola.java**: contiene la Interfaz del servicio (SEI del servicio)
- **Hola_Service.java**: contiene la Clase heredada de Service, que utilizaremos para acceder al componente Port de nuestro servicio web a través del SEI. Para ello debemos utilizar el método "Hola getHolaPort()" de la clase Hola_Service
- **ObjectFactory.java**: Contiene una factoria de métodos para recuperar representaciones java a partir de definiciones XML
- **Hello.java, HelloResponse.java**: son las clases que representan mensajes de nuestro WSDL
- **package-info.java**: fichero con información sobre las clases generadas

Clase App.java

```
package soaesfot.holamundojavaclient;
public class App
{
  public static void main( String[] args )
  {
    try {
      //Primero accedemos a un objeto Service
      sw.Hola_Service service = new sw.Hola_Service();
      //a traves de el accedemos al Port
      sw.Hola port = service.getHolaPort();
      java.lang.String name = "amigos de los Servicios Web";
      //utilizamos el Port para llamar al WS a traves del SEI
      java.lang.String result = port.hello(name);
      System.out.println("Result = "+result);
    } catch (Exception ex) {
      // TODO handle custom exceptions here
    }
  }
}
```

Finalmente

- En el `cmd`: `mvn package`

Finalmente

- En el `cmd`: `mvn package`
- `mvn exec:java`

Finalmente

- En el `cmd`: `mvn package`
- `mvn exec:java`
- El resultado debe ser:

```
Result = Hola amigos de los Servicios Web !
```

Invocación de servicios web JAX-WS desde una aplicación Web con Maven

- En Netbeans: Nuevo proyecto -> Maven -> Web Application. **Name:** HolaMundoWebClient, **Group Id:** soaesfot. **Package** soaesfot.holamundowebclient

Invocación de servicios web JAX-WS desde una aplicación Web con Maven

- En Netbeans: Nuevo proyecto -> Maven -> Web Application. **Name:** HolaMundoWebClient, **Group Id:** soaesfot. **Package** soaesfot.holamundowebclient
- GlassFish Server y JavaEE 7 Web

Invocación de servicios web JAX-WS desde una aplicación Web con Maven

- En Netbeans: Nuevo proyecto -> Maven -> Web Application. **Name:** HolaMundoWebClient, **Group Id:** soaesfot. **Package** soaesfot.holamundowebclient
- GlassFish Server y JavaEE 7 Web
- Abrir archivo: Proyectos -> HolaMundoWebClient -> Project Files -> pom.xml

pom.xml

Tenemos las siguientes dependencias y librerías:

- librería **javaee-web-api**: necesaria para utilizar servlets con anotaciones.

pom.xml

Tenemos las siguientes dependencias y librerías:

- librería **javaee-web-api**: necesaria para utilizar servlets con anotaciones.
- plugin **maven-compiler**: necesitamos configurar la versión de los fuentes y ejecutables java, y para compilar las librerías del directorio "endorsed". Dicho directorio contendrá versiones actualizadas de librerías de anotaciones de servicios Web.

pom.xml

Tenemos las siguientes dependencias y librerías:

- librería **javaee-web-api**: necesaria para utilizar servlets con anotaciones.
- plugin **maven-compiler**: necesitamos configurar la versión de los fuentes y ejecutables java, y para compilar las librerías del directorio "endorsed". Dicho directorio contendrá versiones actualizadas de librerías de anotaciones de servicios Web.
- plugin **maven-war**: utilizado para empaquetar nuestro proyecto ignorando el fichero web.xml a la hora de hacer el build de nuestro proyecto.

pom.xml

Tenemos las siguientes dependencias y librerías:

- librería **javaee-web-api**: necesaria para utilizar servlets con anotaciones.
- plugin **maven-compiler**: necesitamos configurar la versión de los fuentes y ejecutables java, y para compilar las librerías del directorio "endorsed". Dicho directorio contendrá versiones actualizadas de librerías de anotaciones de servicios Web.
- plugin **maven-war**: utilizado para empaquetar nuestro proyecto ignorando el fichero web.xml a la hora de hacer el build de nuestro proyecto.
- plugin **maven-dependency**: lo utilizaremos para copiar en el directorio "endorsed" (target/endorsed) la librería javaee-endorsed-api-6.0.jar. Esta librería permite utilizar anotaciones de servicios Web y se utilizará en la librería correspondiente de jax-ws que viene incluida por defecto en jdk.

Plugins extras

Necesarios para desplegar nuestro cliente de WS en glassfish

- plugin **jaxws-maven-plugin**: con el que ejecutaremos la goal `wsimport`, igual que hicimos para el cliente Java del ejemplo anterior.

Plugins extras

Necesarios para desplegar nuestro cliente de WS en glassfish

- plugin **jaxws-maven-plugin**: con el que ejecutaremos la goal `wsimport`, igual que hicimos para el cliente Java del ejemplo anterior.
- plugin **glassfish**: para desplegar el war generado en el servidor de aplicaciones, utilizando la goal glassfish:deploy.

Plugins extras

Necesarios para desplegar nuestro cliente de WS en glassfish

- plugin **jaxws-maven-plugin**: con el que ejecutaremos la goal `wsimport`, igual que hicimos para el cliente Java del ejemplo anterior.
- plugin **glassfish**: para desplegar el war generado en el servidor de aplicaciones, utilizando la goal glassfish:deploy.
  - Para configurar el plugin, vamos a indicar la contraseña utilizando el fichero master-password, que contiene la contraseña codificada del administrador del dominio, y que está situado en el directorio raíz de dicho dominio (verificar con Netbeans).

Plugins extras

Necesarios para desplegar nuestro cliente de WS en glassfish

- plugin **jaxws-maven-plugin**: con el que ejecutaremos la goal `wsimport`, igual que hicimos para el cliente Java del ejemplo anterior.
- plugin **glassfish**: para desplegar el war generado en el servidor de aplicaciones, utilizando la goal glassfish:deploy.
  - Para configurar el plugin, vamos a indicar la contraseña utilizando el fichero master-password, que contiene la contraseña codificada del administrador del dominio, y que está situado en el directorio raíz de dicho dominio (verificar con Netbeans).
  - Por defecto, la contraseña del administrador del dominio es *changeit*, y esta contraseña NO se guarda (es decir, no se crea el master-password).

Plugins extras

Necesarios para desplegar nuestro cliente de WS en glassfish

- plugin **jaxws-maven-plugin**: con el que ejecutaremos la goal `wsimport`, igual que hicimos para el cliente Java del ejemplo anterior.
- plugin **glassfish**: para desplegar el war generado en el servidor de aplicaciones, utilizando la goal glassfish:deploy.
  - Para configurar el plugin, vamos a indicar la contraseña utilizando el fichero master-password, que contiene la contraseña codificada del administrador del dominio, y que está situado en el directorio raíz de dicho dominio (verificar con Netbeans).
  - Por defecto, la contraseña del administrador del dominio es *changeit*, y esta contraseña NO se guarda (es decir, no se crea el master-password).
  - Para crear este fichero tendremos que utilizar los siguientes comandos (modo admin):

```
cd C:/Program Files/glassfish-4.1/bin/

asadmin stop-domain

asadmin change-master-password -savemasterpassword=true
domain1
```

Plugin glassfish

```
<plugin>
    <groupId>org.glassfish.maven.plugin</groupId>
    <artifactId>maven-glassfish-plugin</artifactId>
    <version>2.1</version>
    <configuration>
        <user>admin</user>
        <passwordFile>
            C:\Program Files\glassfish-4.1\glassfish\domains\domain1\master-password
        </passwordFile>
        <glassfishDirectory>C:\Program Files\glassfish-4.1\glassfish</glassfishDirectory>
        <domain>
            <name>domain1</name>
            <adminPort>4848</adminPort>
            <httpPort>8080</httpPort>
        </domain>
        <components>
            <component>
                <name>HolaMundoWebClient</name>
                <artifact>target/HolaMundoWebClient-1.0-SNAPSHOT.war</artifact>
            </component>
        </components>
    </configuration>
</plugin>
```

GUARDAR EL ARCHIVO

Desplegar la aplicación web

- La aplicación está lista para ser desplegada en glassfish

Desplegar la aplicación web

- La aplicación está lista para ser desplegada en glassfish
- Vamos a `http://localhost:4848/` -> Desplegar aplicación. Buscamos el archivo en nuestro proyecto "target/HolaMundoWebClient-1.0-SNAPSHOT.war".

Desplegar la aplicación web

- La aplicación está lista para ser desplegada en glassfish
- Vamos a `http://localhost:4848/` -> Desplegar aplicación. Buscamos el archivo en nuestro proyecto "target/HolaMundoWebClient-1.0-SNAPSHOT.war".
- Desplegar-> launch

Implementación del Servlet

- Clicc derecho en Source Packages -> soaesfot.holamundowebclient. New -> Servlet. **Name** NewServlet.

Implementación del Servlet

- Clicc derecho en Source Packages -> soaesfot.holamundowebclient. New -> Servlet. **Name** NewServlet.
- Copiar contenido del archivo NewServlet.java en el repo.

Implementación del Servlet

- Clicc derecho en Source Packages -> soaesfot.holamundowebclient. New -> Servlet. **Name** NewServlet.
- Copiar contenido del archivo NewServlet.java en el repo.
- Para que nuestro artefacto cambie el nombre por defecto que nos da Maven ("HolaMundoWebClient-1.0-SNAPSHOT") por " HolaMundoWebClient" debemos añadir el archivo `glassfish-web.xml` en `src/main/webapp/WEB-INF/`

Implementación del Servlet

- Clicc derecho en Source Packages -> soaesfot.holamundowebclient. New -> Servlet. **Name** NewServlet.
- Copiar contenido del archivo NewServlet.java en el repo.
- Para que nuestro artefacto cambie el nombre por defecto que nos da Maven ("HolaMundoWebClient-1.0-SNAPSHOT") por " HolaMundoWebClient" debemos añadir el archivo `glassfish-web.xml` en `src/main/webapp/WEB-INF/`
- HolaMundoWebClient -> Web Pages -> WEB-INF. Clic derecho Nuevo -> XML Document.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE glassfish-web-app PUBLIC "-//GlassFish.org//DTD GlassFish Ap
  <glassfish-web-app>
    <context-root>HolaMundoWebClient</context-root>
  </glassfish-web-app>
```

Deplegando la aplicación

- clic derecho en el proyecto-> Clean and build

Deplegando la aplicación

- clic derecho en el proyecto-> Clean and build
- ir a `http://localhost:4848/`

Deplegando la aplicación

- clic derecho en el proyecto-> Clean and build
- ir a `http://localhost:4848/`
- Vamos a `http://localhost:4848/` -> Desplegar aplicación. Buscamos el archivo en nuestro proyecto "target/HolaMundoWebClient-1.0-SNAPSHOT.war".

Deplegando la aplicación

- clic derecho en el proyecto-> Clean and build
- ir a `http://localhost:4848/`
- Vamos a `http://localhost:4848/` -> Desplegar aplicación. Buscamos el archivo en nuestro proyecto "target/HolaMundoWebClient-1.0-SNAPSHOT.war".
- Desplegar-> launch

Deplegando la aplicación

- clic derecho en el proyecto-> Clean and build
- ir a `http://localhost:4848/`
- Vamos a `http://localhost:4848/` -> Desplegar aplicación. Buscamos el archivo en nuestro proyecto "target/HolaMundoWebClient-1.0-SNAPSHOT.war".
- Desplegar-> launch
- ir a `http://localhost:8080/HolaMundoWebClient/Hola`

Pasando el parámetro desde el navegador

- Para pasar el parámetro de entrada directamente en la llamada desde el navegador debemos modificar el servlet.

```
String cadena = request.getParameter("x");
...
out.println("<p>" + port.hello(cadena) + "%lt;/p>");
```

Pasando el parámetro desde el navegador

- Para pasar el parámetro de entrada directamente en la llamada desde el navegador debemos modificar el servlet.

```
String cadena = request.getParameter("x");
...
out.println("<p>" + port.hello(cadena) + "%lt;/p>");
```

- ir a `http://localhost:8080/HolaMundoWebClient/Hola?x=PepePerez`

Pasando el parámetro desde el navegador

- Para pasar el parámetro de entrada directamente en la llamada desde el navegador debemos modificar el servlet.

```
String cadena = request.getParameter("x");
...
out.println("<p>" + port.hello(cadena) + "%lt;/p>");
```

- ir a `http://localhost:8080/HolaMundoWebClient/Hola?x=PepePerez`
- Si deseamos pasar más variables debemos pasarlas de manera `variable=valor` y separadas por `&`

Invocación del servicio Web desde una página JSP

Creamos el archivo `src/main/webapp/index.jsp`

```
<%@page contentType="text/html" pageEncoding="UTF-8" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSP Page</title>
  </head>
  <body>
    <h1>Hola Mundo!</h1>
<%
    try {
      sw.Hola_Service service = new sw.Hola_Service();
      sw.Hola port = service.getHolaPort();
      String name = "amigos de los Servicios Web";
      String result = port.hello(name);
      out.println("Result = "+result);
    } catch (Exception ex) {
      // TODO handle custom exceptions here
    }
%>
  </body>
</html>
```

Desplegar e ingresar a

`http://localhost:8080/HolaMundoWebClient/index.jsp`

Contenido I

Deber

En la carpeta Deber del repo.