Module 13
Implementing Java EE Web Services with JAX-WS

Objectives

- Describe endpoints supported by Java EE 5
- Describe the requirements of the JAX-WS servlet endpoints
- Describe the requirements of JAX-WS EJB endpoints
- Develop web service clients
A web service endpoint is the component that executes as a result of sending a SOAP message to an application server. The Java EE 5 components that can function as endpoints are:

- Servlet components
- Stateless session bean components

Both JAX-WS and JAX-RPC endpoints can be used in Java EE 5. Web services currently under development should use JAX-WS. JAX-WS is an update of JAX-RPC and it simplifies creating and using web services.

A JAX-WS endpoint is not a servlet in the traditional sense. The JAX-WS framework provides a servlet implementation used to handle HTTP requests and process SOAP messages.

A JAX-WS web endpoint:

- Is a standard Java class created just to provide web server functionality
- Is multi-threaded
- Does not require an EJB container
Implementing a JAX-WS Servlet Endpoint

A JAX-WS web endpoint:

- Has a `@javax.jws.WebService` class annotation
- Contains business methods that are public and not final or static
- Has exposed web service methods that are annotated with `@javax.jws.WebMethod`
- Cannot be an abstract or final class
- Requires a default no-arg constructor

Simple Web Component Endpoint Example

```java
package example;
import javax.jws.*;
@WebService
public class SayHello {
    @WebMethod
    public String getGreeting(String name) {
        return "Hello " + name;
    }
}
```
The web.xml file in your web archive (WAR) must be updated to provide a usable URL for the web service.

```xml
<servlet>
    <servlet-name>hello</servlet-name>
    <servlet-class>example.SayHello</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>hello</servlet-name>
    <url-pattern>/sayhello</url-pattern>
</servlet-mapping>
```

Only stateless session beans can be a JAX-WS EJB endpoint. As with web JAX-WS endpoints, the application server provides support for HTTP and SOAP processing, typically with a servlet. A stateless session bean endpoint:

- Is typically an existing session bean already in use by your application.
- Is turned into a web service. Session bean-based web services allow for non-Java client applications.
- Requires an EJB container
JAX-WS EJB Endpoints

A JAX-WS EJB endpoint:

• Has a `@javax.jws.WebService` class annotation
• Contains business methods that are public and not final or static
• Has exposed web service methods that are annotated with `@javax.jws.WebMethod`
• Cannot be an abstract or final class
• Requires a default no-arg constructor
• Must be a stateless session bean

A JAX-WS EJB Endpoint Example

```java
import javax.ejb.*;
import javax.jws. *;

@WebService @Stateless @Remote
public class SayHelloBean
    implements SayHelloInterface {

    @WebMethod
    public String getGreeting(String name) {
        return "Hello " + name;
    }
}
```
JAX-WS Endpoint Life Cycle

JAX-WS endpoints, both web and EJB, can have optional life-cycle methods that are automatically called if present. Any method can be used as a life-cycle method with the correct annotation:

- `@PostConstruct` – Called by the container before the implementing class begins responding to web service clients.
- `@PreDestroy` – Called by the container before the endpoint is removed from operation.

JAX-WS Allowed Data Types

Unlike the JAX-PRC, the JAX-WS does not specify a Java-to-XML binding for data types. The JAXB is specified by JAX-WS as the way in which Java data is converted back and forth to XML.

Basic Java types, such as Strings are supported automatically. However, to return or pass complex object instances, some JAXB programming is required.
Web Service Clients

The key to creating a web service client, in any language, is having a copy of the web service’s WSDL. The WSDL describes the operations, arguments, and return values used in a web service.

The WSDL for a JAX-WS web service is generated automatically when the web service is deployed.

Although not part of the specification, to retrieve the generated WSDL from the Sun Application Server, you can use your browser by requesting a URL similar to:


Developing JAX-WS Clients

To access a web service from a JAX-WS client you need:

- The WSDL
- A Proxy object to handle the creation of SOAP messages and HTTP communication. This proxy type class is known as a Port in JAX-WS.
- To generate the Port class or source code and any other required artifacts for the web service.

The JAX-WS reference implementation and the Sun Java Application Server provide an application known as wsimport to create all the required client code.

The wsimport application can also be run as an Ant task.
A JAX-WS Client Example

In the following example, the Service class is used to instantiate a Port or proxy. The Port handles all SOAP message creation and transmission.

```java
import javax.xml.ws.WebServiceRef;
public class WSTest {
    public WSTest() { }
    public static void main(String[] args) {
        SayHelloService service = new SayHelloService();
        SayHello port = service.getSayHelloPort();
        System.out.println(port.sayHello("Duke"));
    }
}
```

Other Types of Web Service Clients

A JAX-WS web service can be called by any client or platform. The client must:

- Have access to the WSDL file
- Support the correct version of the SOAP specification.

For maximum compatibility, the client should support the WS-I Basic Profile 1.1.