Java API for XML-based RPC (JAX-RPC)

- JAX-RPC hides the details involved in forming the XML for SOAP messages and exchanging those messages with a Web service.
- It allows Java developers to simply make method calls to invoke Web services as if they were simply making a remote procedure call.
- JAX-RPC handles marshalling your method parameters from Java types to XML, forming a SOAP message to package your parameters for transmission, sending the resultant SOAP request message to the Web service you wish to invoke, receiving the Web service’s response message, and unmarshalling the XML from the SOAP response message into Java types.
JAX-RPC, cont.

- A hollow API, like SAAJ, that requires a vendor implementation.
- Under the covers, JAX-RPC version 1.1 uses SAAJ version 1.2 to process SOAP messages.
- With JAX-RPC you can use one-way and request-response message exchange patterns, RPC/Literal, RPC/Encoded, and Document/Literal message styles, and HTTP as the underlying transport protocol.
- JAX-RPC also provides a standard mapping from WSDL service definitions to Java interfaces and from XML data types to Java data types.
- The standard programming model in J2EE for Web service clients and endpoints.
JAX-RPC, cont.

• JAX-RPC is about saving you development time when you want to work with Web services in Java.
• JAX-RPC consists of a client-side programming model and a server-side programming model.
• The client-side programming model provides you the ability to communicate with remote Web services by making method calls to a local object serving as a proxy.
• The server-side programming model allows you to expose Java objects or Enterprise JavaBeans as Web services.
JAX-RPC Server-Side Programming Model

- Allows you to create Web services that are fully interoperable with clients implemented in other languages because it is based on the standard Web service technologies and the WS-I BP.
- Web services implemented by JAX-RPC are referred to as JAX-RPC service endpoints.
- JAX-RPC service endpoints can be implemented as either stateless EJBs (*EJB endpoints*) or plain Java objects that, when deployed, are backed by a Java Servlet and execute in a Java Servlet container (*servlet endpoints*).
- The servlet endpoint is easy to develop because all it requires you to provide is a plain Java object that implements an interface that extends `java.rmi.Remote`. The interface is called the JAX-RPC *service endpoint interface* (*SEI*) and the plain Java object that implements it is referred to as the service implementation.
Servlet Endpoint

- The WSDL service interface definition, which is defined by the `wsdl:portType`, corresponds directly to the JAX-RPC service endpoint interface.
- Ex. RPC-style servlet based service endpoint interface:

  ```java
  public interface MonitorPricingSEI extends java.rmi.Remote {
      public float getMonitorPrice(String productId, String currency)
          throws java.rmi.RemoteException;
  }
  ```

- Ex. Implementation:

  ```java
  public class MonitorPricing_Impl implements MonitorPricingSEI {
      public float getMonitorPrice(String productId, String currency) {
          // To keep this simple, assume the currency is USD and
          // the price of the specified monitor is the following:
          return 700.00f;
      }
  }
  ```
 Servlet Endpoint, cont.

- The SEI is required to extend the `java.rmi.Remote` interface and each of its declared methods must throw the `java.rmi.RemoteException`.
- The SEI defines the Web service operations that will be publicly available once the servlet endpoint is deployed.
- The SEI and implementation classes are deployed to a Java servlet container (like Tomcat) to run.
- On deployment, the servlet container will create a special JAX-RPC Java servlet to handle the requests for the service endpoint. The JAX-RPC servlet will respond to HTTP-based SOAP requests, parse incoming SOAP request messages, marshal the XML data to the appropriate Java types, and invoke the correct method on the service endpoint, passing it the resultant Java types as parameters.
Servlet Endpoint, cont.

• A servlet endpoint has access to the same resources as a normal Java servlet, which includes JDBC DataSources, EJBs, environment variables, and other Web services, plus the ServletContext and the HttpSession.

• Servlet endpoints can use the Java Naming and Directory Interface (JNDI) Environment Naming Context (ENC) to look up resources that have been configured dynamically at runtime or statically via deployment descriptors, and servlet endpoints can implement the javax.xml.rpc.server.ServiceLifecycle interface to get access to the underlying ServletContext.
Servlet Endpoint, cont.

- The `ServiceLifecycle` interface can be implemented by a servlet endpoint to gain access to the JAX-RPC servlet’s context and to perform initialization when the servlet endpoint is created plus cleanup when the servlet endpoint is destroyed.
- The `ServiceLifecycle` interface declares only two methods: `init()` and `destroy()`. The `init()` method is called by the servlet container just after the servlet endpoint is instantiated and before it handles any requests. The `destroy()` method is called by the servlet container just before the servlet endpoint is removed from service.
- You can count on the `init()` method only being called once in a servlet endpoint’s lifecycle.
/** Example getting DataSource from JNDI ENC. */
public class MonitorPricing_Impl implements MonitorPricingSEI,
   javax.xml.rpc.server.ServiceLifecycle
{
   private javax.sql.DataSource dataSource = null;

   // Called at the beginning of a servlet endpoint’s lifecycle
   public void init(Object context) throws javax.xml.rpc.ServiceException
   {
      try {
         javax.naming.InitialContext jndiEnc =
            new javax.naming.InitialContext();
         dataSource = (javax.sql.DataSource)
            jndiEnc.lookup("java:comp/env/jdbc/MonitorDataSource");
      }catch (javax.naming.NamingException e) {
         throw new ServiceException("JNDI lookup failed", e);
      }
   }

   // Called at the end of a servlet endpoint’s lifecycle
   public void destroy() { // Perform some cleanup. }
   ...
Servlet Endpoint, cont.

- The parameter to the `init()` method of the `ServiceLifecycle` interface is an `Object` because the `ServiceLifecycle` interface was intended to be generic enough to be used outside of a J2EE servlet container (like a standalone Axis server). However, inside of a J2EE servlet container you can expect the `Object` passed to `init()` to be of type `ServletEndpointContext`.

- `ServletEndpointContext` provides methods that give you access to the user’s `Principal`, the `HttpSession`, the `ServletContext`, and the SOAP message (via `MessageContext`).

- The servlet endpoint only gets a reference to the `ServletEndpointContext` during the call to `init()`, but the values returned from the methods of `ServletEndpointContext` will change for each request.
import javax.xml.rpc.handler.soap.SOAPMessageContext;
import javax.xml.rpc.server.ServletEndpointContext;
/** Example getting ServletEndpointContext and SOAPMessageContext. */
public class MonitorPricing_Impl implements MonitorPricingSEI,
        javax.xml.rpc.server.ServiceLifecycle {
    private ServletEndpointContext endpointContext = null;

    public void init(Object context) throws javax.xml.rpc.ServiceException
    {
        endpointContext = (ServletEndpointContext) context;
    }
    public void destroy() { // Perform some cleanup. }

    public float getMonitorPrice(String productId, String currency) {
        javax.servlet.http.HttpSession session =
            endpointContext.getHttpSession();
        java.security.Principal principal =
            endpointContext.getUserPrincipal();
        // The SOAPMessageContext will be different for every request.
        SOAPMessageContext soapMsgCntxt = (SOAPMessageContext)
            endpointContext.getMessageContext();
        javax.xml.soap.SOAPMessage soapMessage = soapMsgCntxt.getMessage();
    }
}
EJB Endpoint

- EJB endpoints can take advantage of EJB’s automatic handling of transaction management, persistence, security, and resources.
- A single EJB can be deployed to handle remote, local, and Web service clients simultaneously.
- SOAP, is a stateless protocol, so only stateless session EJBs can be deployed as Web services.
- To expose a stateless session bean as a Web service: define a service endpoint interface, have the stateless session bean implement the interface, deploy it.
- No need for remote interface that extends `EJBOBJECT` or local interface that extends `EJBLocal` or `EJB` home interface.
- Only have to implement the methods of the SEI. SEI methods must throw `RemoteException`, but Bean methods cannot.
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;

/** Example EJB Endpoint that directly implements SEI. */
public class MonitorPricingBean implements SessionBean,
        MonitorPricingSEI {
    public void setSessionContext(SessionContext context) {}
    public void ejbCreate() {}

    public float getMonitorPrice(String productId, String currency) {
        // To keep this simple, assume the currency is USD and
        // the price of the specified monitor is the following:
        return 700.00f;
    }

    public void ejbRemove() {}
    public void ejbActivate() {}
    public void ejbPassivate() {}
}
EJB Endpoint, cont.

- Just like servlet endpoints, EJB endpoints can get a reference to the `javax.xml.rpc.handler.MessageContext`.
- As with servlet endpoints, the `MessageContext` for EJB endpoints is different for each request (different SOAP message).
import javax.xml.rpc.handler.soap.SOAPMessageContext;

public class MonitorPricingBean implements javax.ejb.SessionBean {
    private javax.ejb.SessionContext sessionContext = null;

    public void setSessionContext(javax.ejb.SessionContext context) {
        // Must maintain a reference to SessionContext.
        sessionContext = context;
    }

    public void ejbCreate() {}

    public float getMonitorPrice(String productId, String currency) {
        // The SOAPMessageContext will be different for every request.
        SOAPMessageContext soapMsgCntxt = (SOAPMessageContext)
            sessionContext.getMessageContext();
        javax.xml.soap.SOAPMessage soapMessage =
            soapMsgCntxt.getMessage();

        public void ejbRemove() {}
        public void ejbActivate() {}
        public void ejbPassivate() {}
    }
}