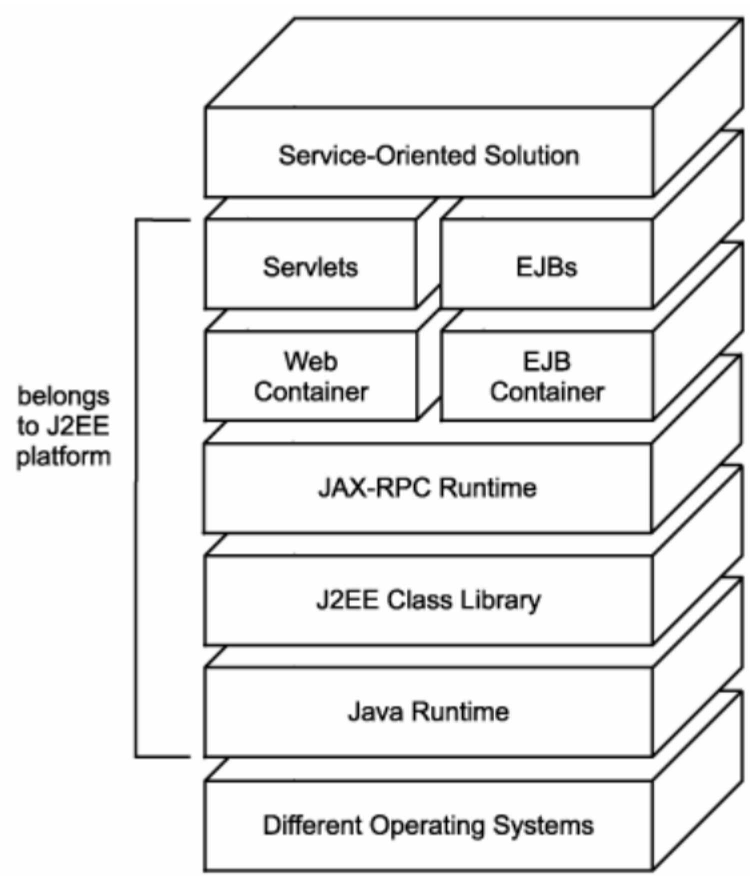
SOA support in J2EE:

The Java 2 Platform is divided into three major development and runtime platforms, each addressing a different type of solution. The Java 2 Platform Standard Edition (J2SE) is designed to support the creation of desktop applications, while the Micro Edition (J2ME) is geared toward applications that run on mobile devices

The Java 2 Platform Enterprise Edition (J2EE) is one of the two primary platforms currently being used to develop enterprise solutions using Web services. The Java 2 Platform Enterprise Edition (J2EE) is built to support large-scale, distributed solutions. It has been used extensively to build traditional n-tier applications with and without Web technologies.

The J2EE development platform consists of numerous composable pieces that can be assembled into full-fledged Web solutions. Let's take a look at some of the technologies more relevant to Web services.

Relevant layers of the J2EE platform as they relate to SOA



Architecture components

J2EE solutions inherently are distributed and therefore componentized. The following types of components can be used to build J2EE Web applications:

> **Java Server Pages (JSPs**) Dynamically generated Web pages hosted by the Web server. JSPs exist as text files comprised of code interspersed with HTML.
>
> **Java Servlets** These components also reside on the Web server and are used to process HTTP request and response exchanges. Unlike JSPs, servlets are compiled programs.
>
> **Enterprise JavaBeans (EJBs)** The business components that perform the bulk of the processing within enterprise solution environments. They are deployed on dedicated application servers and can therefore leverage middleware features, such as transaction support.

While JSPs are of more relevance to establishing the presentation layer of a service-oriented solution, the latter two commonly are used to realize Web services.

Runtime environments

In support of Web services, J2EE provides additional runtime layers that, in turn, supply additional Web services specific APIs (explained later). Most notable is the JAX-RPC runtime, which establishes fundamental services, including support for SOAP communication and WSDL processing.

Java API for XML-based RPC (JAX-RPC) This document defines the JAX-RPC environment and associated core APIs. It also establishes the Service Endpoint Model used to realize the JAX-RPC Service Endpoint, one of the primary types of J2EE Web services

Additionally, implementations of J2EE supply two types of component containers that provide hosting environments geared toward Web services-centric applications that are generally EJB or servlet-based.

- **EJB container** : This container is designed specifically to host EJB components, and it provides a series of enterprise-level services that can be used collectively by EJBs participating in the distributed execution of a business task. Examples of these services include transaction management, concurrency management, operation-level security, and object pooling.
- **Web container:** A Web container can be considered an extension to a Web server and is used to host Java Web applications consisting of JSP or Java servlet components. Web containers provide runtime services geared toward the processing of JSP requests and servlet instances.

EJB and Web containers can host EJB-based or servlet-based J2EE Web services.

Web service execution on both containers is supported by JAX-RPC runtime services. However, it is the vendor-specific container logic that generally determines the shape and form of the system-level message processing logic provided in support of Web services.

J2EE vendors provide containers as part of their server products. A container then establishes the runtime that hosts an instance of the vendor's server software. Examples of currently available containers are Sun ONE (Open Network Environment) Application Server, IBM WebSphere, and Oracle Application Server Containers for J2EE (OC4J)

Programming languages

As its name implies, the Java 2 Platform Enterprise Edition is centered around the Java programming language. Different vendors offer proprietary development products that provide an environment in which the standard Java language can be used to build Web services. Examples of currently available development tools are Rational Application Developer from IBM, Java Studio from Sun Microsystems, and JDeveloper from Oracle.

Java™ API for XML-based web services (JAX-WS):

[illegible]

Java API for XML Web Services (JAX-WS) is the Java standard application program interface (API) for XML Web services. It is used to develop Web services and is a part of the Sun Java development kit (JDK). JAX-WS technology is used with other technologies, either from the core group or more enhanced Web services.

JAX-WS was designed to replace the existing JAX-RPC (remote procedure call). The name was changed to JAX-WS from JAX-RPC to reflect the shift from RPC-style to document-style Web services.

JAX-WS consists of a standardized set of extensions for Java, which enable the development of Java-based Web services through WSDL. Just like JAX-RPC, JAX-WS also uses SOAP to represent a RPC. SOAP includes specifications, encoding rules, important structures, corresponding responses and necessary conventions to do RPCs over the network.

Although SOAP messages are complex, the JAX-WS API hides this complexity from the application developer. On the server side, the developer specifies the web service operations by defining methods in an interface written in the Java programming language. The developer also codes one or more classes that implement those methods. Client programs are also easy to code. A client creates a proxy (a local object representing the service) and then simply invokes methods on the proxy. With JAX-WS, the developer does not generate or parse SOAP messages. It is the JAX-WS runtime system that converts the API calls and responses to and from SOAP messages.

With JAX-WS, clients and web services have a big advantage: the platform independence of the Java programming language. In addition, JAX-WS is not restrictive: A JAX-WS client can access a web service that is not running on the Java platform and vice versa. This flexibility is possible because JAX-WS uses technologies defined by the W3C: HTTP, SOAP, and WSDL. WSDL specifies an XML format for describing a service as a set of endpoints operating on messages.

Java API for XML Based RPC:

JAX-RPC stands for Java API for XML-based RPC. It's an API for building Web services and clients that used remote procedure calls (RPC) and XML. Often used in a distributed client/server model, an RPC mechanism enables clients to execute procedures on other systems.

In JAX-RPC, a remote procedure call is represented by an XML-based protocol such as SOAP. The SOAP specification defines envelope structure, encoding rules, and a convention for representing remote procedure calls and responses. These calls and responses are transmitted as SOAP messages over HTTP.

Although JAX-RPC relies on complex protocols, the API hides this complexity from the application developer. On the server side, the developer specifies the remote procedures by defining methods in an interface written in the Java programming language. The developer also codes one or more classes that implement those methods. Client programs are also easy to code. A client creates a proxy, a local object representing the service, and then simply invokes methods on the proxy.

With JAX-RPC, clients and Web services have a big advantage--the platform independence of the Java programming language. In addition, JAX-RPC is not restrictive: a JAX-RPC client can access a Web service that is not running on the Java platform and vice versa. This flexibility is possible because JAX-RPC uses technologies defined by the World Wide Web Consortium (W3C): HTTP, SOAP, and the Web Service Description Language (WSDL). WSDL specifies an XML format for describing a service as a set of endpoints operating on messages.

A Simple Example: HelloWorld

This example shows you how to use JAX-RPC to create a Web service named `HelloWorld`. A remote client of the `HelloWorld` service can invoke the `sayHello` method, which accepts a string parameter and then returns a string.

HelloWorld at Runtime

Here's a detailed description of what happens at runtime:

1. To call a remote procedure, the `HelloClient` program invokes a method on a stub, a local object that represents the remote service.

1. The stub invokes routines in the JAX-RPC runtime system.

1. The runtime system converts the remote method call into a SOAP message and then transmits the message as an HTTP request.

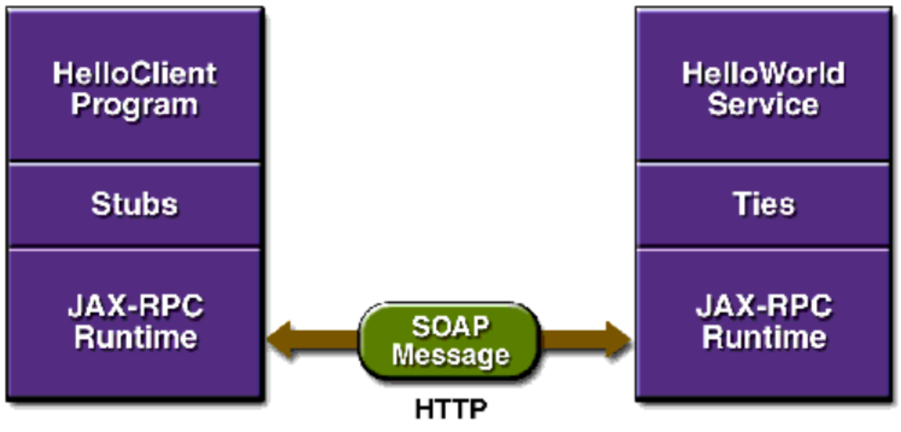
1. When the server receives the HTTP request, the JAX-RPC runtime system extracts the SOAP message from the request and translates it into a method call.

1. The JAX-RPC runtime system invokes the method on the tie object.

1. The tie object invokes the method on the implementation of the `HelloWorld` service.

1. The runtime system on the server converts the method's response into a SOAP message and then transmits the message back to the client as an HTTP response.

1. On the client, the JAX-RPC runtime system extracts the SOAP message from the HTTP response and then translates it into a method response for the `HelloClient` program.



-1 The `HelloWorld` Example at Runtime

Refer the following link for further understanding on JAX-RPC

[A Simple Example: HelloWorld (uccs.edu)]()

Java API for XML Registries (JAXR)

It provides a uniform and standard Java API for accessing different kinds of XML registries.

An XML *registry* is an infrastructure that enables the building, deployment, and discovery of Web services. It is a neutral third party that facilitates dynamic and loosely coupled business-to-business (B2B) interactions. A registry is available to organizations as a shared resource, often in the form of a Web-based service.

A *registry provider* is an implementation of a business registry that conforms to a specification for XML registries.

JAXR enables Java software programmers to use a single, easy-to-use abstraction API to access a variety of XML registries. A unified JAXR information model describes content and metadata within XML registries.

JAXR Architecture

The high-level architecture of JAXR consists of the following parts:

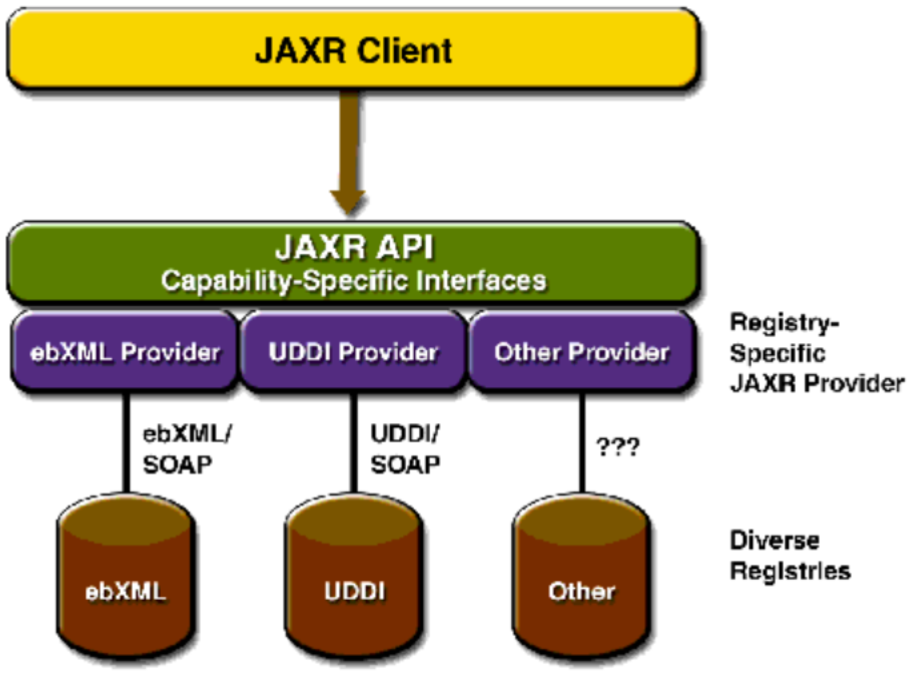
- A *JAXR client*: a client program that uses the JAXR API to access a business registry via a JAXR provider.
- A *JAXR provider*: an implementation of the JAXR API that provides access to a specific registry provider or to a class of registry providers that are based on a common specification.

A JAXR provider implements two main packages:

- `javax.xml.registry`, which consists of the API interfaces and classes that define the registry access interface.
- `javax.xml.registry.infomodel`, which consists of interfaces that define the information model for JAXR. These interfaces define the types of objects that reside in a registry and how they relate to each other. The basic interface in this package is the `RegistryObject` interface. Its subinterfaces include `Organization`, `Service`, and `ServiceBinding`.

When an error occurs, JAXR API methods throw a `JAXRException` or one of its subclasses.

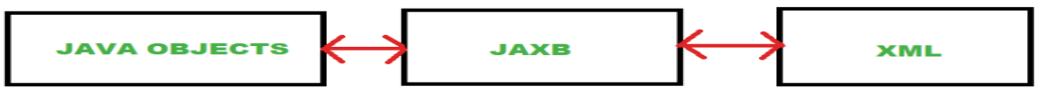
JAXR Architecture



JAVA ARCHITECTURE FOR XML BINDING:

The Java™ Architecture for XML Binding (JAXB) provides an API and tools that automate the mapping between XML documents and Java objects.
The JAXB framework enables developers to perform the following operations:

- Unmarshal(read) XML content into a Java representation
- Access and update the Java representation
- Marshal(write) the Java representation of the XML content into XML content



Java Architecture for XML Binding Functions

JAXB gives Java developers an efficient and standard way of mapping between XML and Java code. Java developers using JAXB are more productive because they can write less code themselves and do not have to be experts in XML. JAXB makes it easier for developers to extend their applications with XML and Web Services technologies.

APPLICATIONS Of JAXB

JAXB framework is helpful to perform the following operations:

- Change XML content into a Java representation.
- Access and update the Java representation
- Change the Java representation into XML content.

Pros and Cons of Using JAXB

Pros:

- Easy to marshal XML file to other data targets like inputStream, DOM node.
- Easy to unmarshal XML file from other data targets.
- No need to be aware of XML parsing techniques.
- simple to use than DOM or SAX parser

Cons:

- JAXB is high layer API so it has less control on parsing than SAX or DOM.
- It is slower than SAX.

1. **Marshalling :** Convert a java Object to xml
2. **UnMarshalling :** Convert xml to java object

Marshalling

Below is the step by step algorithm for converting Java Objects to XML(Marshalling):

1. First Create Java Objects to be Marshalled.
2. Create JAXBContext Object and initializing Marshaller Object.
3. To get the formatted xml output one can set JAXB_FORMATTTED_OUTPUT to True(this Step is optional).

4. Create xml file object by providing location of file as parameter to File Class
5. Call marshal method on Marshaller Object and pass created XML File object to marshal method.
6. Now the XMl file is created.

From Unit 2

Basic Web Services Workflow

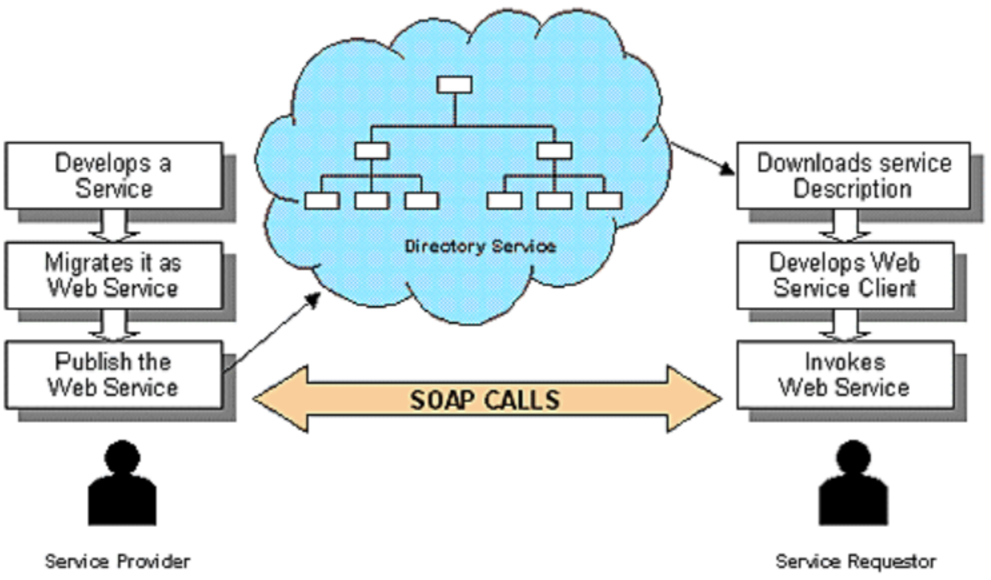
There is a fundamental workflow involved with the description, discovery, and integration of Web services. Understanding this flow is important to appreciate the security concerns I will address later in the article.

Service Provider

- Develops a service (usually a function/procedure or method that may link through several classes and components) in his own language and environment.
- Develops a SOAP layer over and above the service, so that the service can be exposed to SOAP method calls across the Web. In other words, a simple service has become a "Web service."
- Develops a WSDL file, which contains the details of the service, input/output parameters, and so on.
- Publishes his service with a service directory system such as UDDI.

Service Consumer

- Negotiates with the service provider to make use of his service.
- Downloads the WSDL file from the service directory, and understands the details of service invocation.
- Develops a client program that will make use of the service.
- Develops a SOAP layer over the client program so that SOAP calls can be sent and received across the Web. In other words, a simple client has become a Web service client.
- Invokes the service with a SOAP call, as and when required.



WEB SERVICES WORKFLOW