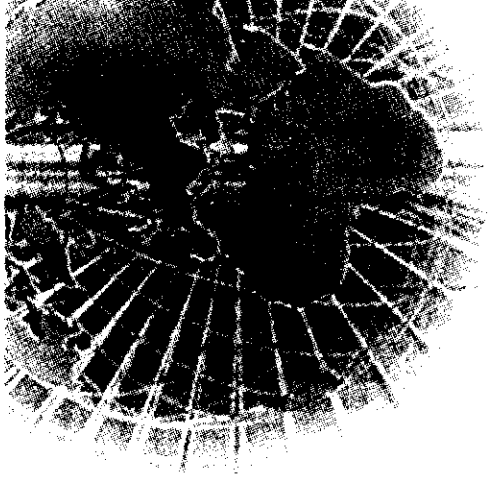


34

Consuming Web Services in AJAX

In the past, the Remote procedures were accessed using platform and language specific protocols, e.g. CORBA used Internet Inter-ORB Protocol to activate remote types, Enterprise JavaBeans needed Remote Method Invocation Protocol, etc. In the same way, each type of architecture would need a tight connection to access a remote source. But with Web services this is not required at all.

Web services present a model by which tasks of e-business processes were distributed widely through Internet. This model is not restricted to a specific business model. Web services are not graphical user interfaces, but can be used into software meant for user interaction. They describe their inputs and outputs in a manner that the second party can predict its functionality, how to call it, and the expected results. The Web services are reusable software components and let the developers reuse basic elements of code made by others. There is a loose connection between these software components, which allow manageable reconfiguration.

An often-cited example of a Web service is that of a stock quote service, in which the request asks for the current price of a specified stock, and the response gives the stock price. This is one of the simplest forms of a Web service. The Web services and consumers of Web services are nothing but typical businesses, making Web services predominantly business-to-business (B-to-B) transactions. Any business can be the provider of Web service, and at the same time be the consumer of other Web services. For example, a wholesale distributor of spices could be in the consumer role when he uses a Web service to check on the availability of vanilla beans and at the same time, in the provider role, when he supplies prospective customers with different vendors' prices for vanilla beans.

Another example of Web service is an auction engine, such as eBay's. This website provides successful auction service. For this auction service, businesses or companies require to build the auction software from start or alternatively send the customers of products to an auction website like eBay. Using Web services, eBay leverage its auction process to other websites and applications for some fee. Businesses only need to subscribe to eBay's Web service and provide some lines of code to their applications to use Web service. Other uses of Web services are payroll management, credit scoring, shipping, business intelligence and mapping services, etc.

Web Services Technologies

Web services use only HTTP which is a protocol all platforms agree upon. Web services are application components designed to support interoperable machine-to-machine interaction over a network. This purpose is achieved using a collection of XML-based standards and protocols, like Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), and Universal Description, Discovery and Integration (UDDI) and Representational State Transfer (REST). We will discuss all of them one-by-one.

SOAP

Simple Object Access Protocol (SOAP) is a protocol which allows applications to exchange information. Unlike the language or platform specific protocols, such as Internet Inter-ORB Protocol (IIOP) or Java Remote Method Protocol (JRMP), the SOAP is not a language or platform specific protocol. It allows communication between applications running on different platforms. SOAP, unlike the IIOP and JRMP which are binary protocols, is a text-based protocol and uses XML-based rule to allow applications interchange information over HTTP. SOAP is based on vendor agnostic technology, namely XML, HTTP, and is shared by all the platforms. It uses the Internet application layer protocol as transport protocol and is used to access Web service.

SOAP Message

SOAP message is a well structured XML document. It is needed for interoperable machine to machine interaction. It inherits all features of XML, like platform independence, human readability, and self documenting format, etc. SOAP messages transform the way business applications communicate over Web with the notion of Web services. The default namespace for SOAP envelope is http://schemas.xmlsoap.org/soap/envelope. Namespace qualified attributes can be used with these elements. SOAP namespaces take care of SOAP message's syntax details. Following are some of the rules that a SOAP message has to abide by:

- It should be written in XML.
- It should use the default namespace for the envelope.

- It should not have a DTD reference.
- It should not have XML processing commands.

SOAP Message Elements

SOAP message elements are used to contain different types of information about a SOAP message and also make it well structured. Before explaining the SOAP message elements, let's have a look at the structure of SOAP message.

The code given in Listing 34.1 presents you a structure of SOAP Message:

Listing 34.1: Structure of SOAP Message

```
<? xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Header>
  ...
  ...
</soap:Header>
<soap:Body>
  ...
  ...
<soap:Fault>
  ...
  ...
</soap:Fault>
</soap:Body>
</soap:Envelope>
```

From seeing this structure, we can deduce that the SOAP message consists of the following elements:

- Envelope element
- Header element
- Body element
- Fault element
- We will now explain them one by one.

The Envelope Element

The Envelope element is the root element of a SOAP message and is an essential element. It recognizes XML document as a SOAP message.

The code given in Listing 34.2 explains how an envelope element looks like:

Listing 34.2: An envelope Element

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  ...
  Message information goes here
  ...
</soap:Envelope>
```

The `xmlns:soap` namespace must have the value `http://www.w3.org/2001/12/soap-envelope`, otherwise the application throws an error and does not accept the message. Any SOAP element can have `encodingStyle` attribute. It defines data types used in the document.

The Header Element

The Header element contains application-related information about SOAP message. It is an optional element and should be the first sub-element of envelope element, if included. The Header element can have attributes, which specify how the recipient processes the message. Header elements act as contracts between the sender and the receiver. The attributes of Header element are as follows:

- actor attribute
- mustUnderstand attribute
- encodingStyle attribute

Let's explore all these elements.

*The **actor** Attribute*

This attribute is used to address the Header element to one or more specific intermediaries.

*The **mustUnderstand** Attribute*

This is used to specify whether the receiver has to process the header.

The code given in Listing 34.3 shows where these attributes are used:

Listing 34.3: Use of Attributes

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Header>
<m:Trans
xmlns:m=http://www.w3schools.com/transaction/
soap:actor="http://www.w3schools.com/appml/">
soap:mustUnderstand="1">
234
</m:Trans>
</soap:Header>
...
...
```

The `mustUnderstand` attribute can accept two values either 0 or 1. We assigned "1" to mustUnderstand attribute which means that the receiver will process the header element. The actor attribute enforces only the header element of this message to reach intermediary represented by http://www.w3schools.com/appml/ URI.

The encodingStyle Attribute

The encodingStyle attribute is used to specify the default namespace for SOAP encoding and data types. In Listing 34.3, http://www.w3.org/2001/12/soap-encoding is the default namespace.

The body Element

It is an essential element. It contains the actual message to be sent to the recipient of the message. The message is application specific and not part of SOAP standards. The code given in Listing 34.4 explains how the body element looks like:

Listing 34.4: The body Element

```
<? xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Body>
  <m:GetPrice xmlns:m="http://www.w3schools.com/prices">
  <m:Item>Shoes</m:Item>
</m:GetPrice>
</soap:Body>
</soap:Envelope>
```

Listing 34.4 requests the price of the Item 'Shoes'. It receives some SOAP response.

The Fault element

An error message from SOAP message is present inside a Fault element. A Fault element can occur once in a SOAP message and, if present, it must be a child element of body element. Table 34.1 lists the sub-elements of the Fault element:

Table 34.1: Sub-elements of Fault Element

Element	Description
`<faultcode>`	It provides a code for identifying the fault. It is a mandatory element
`<faultstring>`	It provides a human-readable description of the fault. It is also a mandatory element
`<faultactor>`	It provides an indication of the system that was responsible for the fault
`<detail>`	It provides information related to faults that occur due to errors associated with the `Body` element of the request message

The Table 34.2 lists the values that the `<faultcode>` element takes with their respective causes:

Table 34.2: Various Fault Code Values and their Respective Causes

Fault code value	Cause
`VersionMismatch`	Invalid namespace is used
`MustUnderstand`	Immediate child element is not understandable
`Client`	Incorrect information in message
`Server`	Internal error

SOAP Example

Let's take an example in which a GetPrice request is sent to a server. The request has a Name parameter. The namespace for the function is `http://www.abc.org/item` and HTTP is the protocol that communicates over TCP/IP. A SOAP request can be HTTP `GET` or `POST` request and the `SOAP` message is a combination of HTTP and XML.

The code given in Listing 34.5 shows an example of SOAP request:

Listing 34.5: The SOAP Request

```
POST /item HTTP/1.1
Host: 189.123.345.239
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 200
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body xmlns:m="http://www.abc.org/item">
  <m:GetPrice>
  <m:Name>Shoes</m:Name>
  </m:GetPrice>
  </soap:Body>
</soap:Envelope>
```

The Response has at least two HTTP headers—Content-Type and Content-Length. The first header represents MIME type and character encoding used for XML body of request or response. The second header tells us the number of bytes in request or response. The code given in Listing 34.6 shows the corresponding SOAP response of this request:

Listing 34.6: The SOAP Response

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body xmlns:m="http://www.abc.org/item">
  <m:GetPriceResponse>
  <m:Price>300.00</m:Price>
  </m:GetPriceResponse>
  </soap:Body>
</soap:Envelope>
```

In this case, the status code of the response is 200 which represents 'success'. Otherwise, the status code sent is 400 which mean that the server is not able to decode the request.

WSDL

In Web services, we try to make it as abstract as possible. Web service registries do not make use of a particular description language essential. Description of service can be written in XML, plain text, etc. WSDL is now becoming a standard for Web service description. Many tools support it and have the capacity to generate WSDL documents from Java code and vice versa. WSDL document consists of the following six elements:

- **Definition**—WSDL document's definitions are root elements, which consist of namespace definitions. WSDL specification recommends the use of XML schema definitions. You can recognize data types of each element during encoding an XML document in this standard.
- **Type**—Types are data type definitions used in message communication with the service. In other words, how are passed parameters to or returned values from methods encoded in neutral way? XML schema has a set of simple types, like strings, double, and you can create complex types based on simple ones. There is another alternative to encode data types inside a WSDL document. WSDL documentation suggests one based on namespace `http://schemas.xmlsoap.org/wsdl/`
- **Message**—Message defines the messages exchanged with the service. Corresponding to each Web service message, the message section contains a message element which further contains sub elements.
- **portType**—WSDL port describes the interfaces exposed by a Web service. It defines Web service, operations to be performed, and messages to be involved.
- **Binding**—Binding specifies the wire protocols used to access each `portType`. A `portType` can have many bindings. In other words, a service might provide access to its methods (operations) via numerous different protocols, such as SOAP, HTTP and RMI-IIOP
- **Service**—Service describes the set of ports to use when invoking a service. A service can have many ports, with each port having a name and a protocol binding.

WSDL Port defines a connection to Web service. It may be understood as a module in programming language. Each operation corresponds to a function in programming language. There are four types of operations defined by WSDL. Table 34.3 lists all these four types of operations:

Table 34.3: Various Operation Types

Operation types	Description
One-way	It represents that the message is only received
Request-Response	It represents that for each request, there is a response
Solicit-Response	It represents the delay in generating response
Notification	It is a message from the server to the client in which the server expects no response

In one way operation, the message is only received and no acknowledgement is generated. The code, given in Listing 34.7 shows one way operation:

Listing 34.7: Part of WSDL Document which Sends a Message

```
<message name="newItemValues">
  <part name="Item" type="xs:string"/>
  <part name="value" type="xs:string"/>
</message>
<portType name="groceryItems">
  <operation name="setItem">
  <input name="newItem" message="newItemValues"/>
  </operation>
</portType >
```

In Listing 34.7, the `groceryItems` port defines a one way operation called `setItem`. The `setItem` operation allows the input of new grocery items messages using a `newItemValues` message with input parameters `Item` and `value`. No output sub element in the operation exists. The code given in Listing 34.8 shows an example of request response operation:

Listing 34.8: Request Response Operation

```
<message name="getItemRequest">
  <part name="Item" type="xs:string"/>
</message>
<message name="getItemResponse">
  <part name="value" type="xs:string"/>
</message>
<portType name="groceryItems">
  <operation name="getItem">
  <input message="getItemRequest"/>
  <output message="getItemResponse"/>
  </operation>
  </portType>
```

In Listing 34.8, the `groceryItems` port defines a request response operation called `getItem`. The `getItem` operation needs an input message called `getItemRequest` with a parameter called `Item` and will return an output message called `getItemResponse` with a parameter called value.

Bindings define the message format and protocol details for a Web service. The code, given in Listing 34.9 shows the usage of bindings:

Listing 34.9: Use of Binding

```
<message name="getItemRequest">
   <part name="Item" type="xs:string"/>
</message>
<message name="getItemResponse">
   <part name="value" type="xs:string"/>
</message>
<portType name="groceryItems">
  <operation name="getItem">
  <input message="getItemRequest"/>
  <output message="getItemResponse"/>
  </operation>
  </portType>
<binding type="groceryItems" name="b">
<soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http" />
<operation>
<soap:operation soapAction="http://abc.com/getItem"/>
<input>
<soap:body use="literal"/>
</input>
<output>
<soap:body use="literal"/>
```

```
</output>
</operation>
</binding>
```

The binding element uses two attributes—name and type. The attribute name specifies the name of the binding and the type of attribute specifies the port for the binding.

The `soap:binding` element also uses two attributes—style and transport. The style value can be one of rpc and document. The transport attribute specifies SOAP protocol to be used. For each operation, the SOAP action must be defined.

The code, given in Listing 34.10 shows the complete WSDL file:

Listing 34.10: The WSDL File

```
<wsdl:definitions name="namemtoken"? targetNamespace="URI">
  <import namespace="URI" location="URI"/> *
  <wsdl:documentation .... /> ?
  <wsdl:types> ?
  <wsdl:documentation .... /> ?
  <xsd:schema .... /> *
  </wsdl:types>
  <wsdl:message name="name"> *
  <wsdl:documentation .... /> ?
  <part name="name" element="name1"? type="name2"?/> *
  </wsdl:message>
  <wsdl:portType name="name"> *
  <wsdl:documentation .... /> ?
  <wsdl:operation name="name"> *
  <wsdl:documentation .... /> ?
  <wsdl:input message="name1"> ?
  <wsdl:documentation .... /> ?
  </wsdl:input>
  <wsdl:output message="name1"> ?
  <wsdl:documentation .... /> ?
  </wsdl:output>
  <wsdl:fault name="name" message="name1"> *
  <wsdl:documentation .... /> ?
  </wsdl:fault>
  </wsdl:operation>
  </wsdl:portType>
  <wsdl:serviceType name="name"> *
  <wsdl:portType name="name1"/> +
  </wsdl:serviceType>
  <wsdl:binding name="name" type="name1"> *
  <wsdl:documentation .... /> ?
  <-- binding details --> *
  <wsdl:operation name="name"> *
  <wsdl:documentation .... /> ?
  <-- binding details --> *
  <wsdl:input> ?
  <wsdl:documentation .... /> ?
  <-- binding details -->
  </wsdl:input>
  <wsdl:output> ?
  <wsdl:documentation .... /> ?
  <-- binding details --> *
  </wsdl:output>
  <wsdl:fault name="name"> *
  <wsdl:documentation .... /> ?
  <-- binding details --> *
  </wsdl:fault>
```

```
    </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="name" serviceType="name1"> *
    <wsdl:documentation .... /> ?
    <wsdl:port name="name" binding="name1"> *
    <wsdl:documentation .... />
    </wsdl:port>
    </wsdl:service>
  </wsdl:definitions>
```

UDDI

UDDI stands for Universal Description Discovery and Integration. It is a directory for storing information about Web services and communicates using SOAP. It uses WSDL to describe interfaces to Web services, e.g. if any industry wants their services to be used by others, it registers its service in UDDI directory. Users then search for the UDDI directory to find the service. When an interface is found, users can communicate with the service.

Earlier, businesses neither had the standard to send information about various products and services nor had the method to integrate different systems. UDDI helps in solving many problems. It enables us to choose the correct business from online ones and then start it. It also tells us how to find new customers and make services in secure environment.

UDDI is a combined effort of many platform and software providers. UDDI can make Web service a resource centric Web service. It has an entity businessEntity which represents an organization. Earlier, businesses were recognized by UUID (Universal Unique Identifier). The UUID is a 128 bit number, which is used to recognize entity (may be business) on internet. For identifying businesses by URI, experts are forced to make businessEntity in the form of an XML document with URI.

UDDI API has the `get_businessDetail` method. This method is redundant and can be removed from API. HTTP also has corresponding GET method. You can perform the same operation by using HTTP GET method. UDDI has many get_methods that work on data objects, such as business services. These data objects can be represented by XML documents and methods can be eliminated. Everything in UDDI database can also be recognized by URI addressable XML resources. Experts are focusing on use of XML in UDDI system. A UDDI entry in one registry points to a UDDI entry in another. When a business changes its information, it registers that change in UDDI. Elements in a single UDDI registry refer to each other, but are not able to refer to objects elsewhere on the web. If the businessEntity documents are written in XML, then it will be easy to add elements and attributes.

Other methods of UDDI can also be removed. The UDDI has the `delete_business` method. HTTP also has corresponding DELETE method. You can perform the same operation by using HTTP DELETE method. You can now perform HTTP delete. The `save_business` method is used for uploading new businesses. A similar method in HTTP is PUT or POST. The `find_business` method is similar to search engine sites. HTTP URI takes a set of search parameters and returns an XML document representing matched entities.

Hence all methods in UDDI API can also be implemented by HTTP-based URIs. This produces another type of architecture for representing Web services of businesses, called REST.

REST

Earlier, business organizations connect to internet using SMTP and FTP clients, and servers to send messages, text files, etc. Now Internet is used to put information of businesses into Web Framework. Technologies used by early Web Framework are HTML, HTTP, and URIs. These technologies are not found successful to integrate Web services with each other. Web services are now based on an architectural style known as Representational State Transfer (REST). The REST architecture is focusing on XML aspects so that Web services can easily integrated with each other.

Roy Fielding's definition of REST is as follows:

"Representational State Transfer is intended to evoke an image of how a well designed Web application behaves: a network of web pages (virtual state machine), where user progresses through an application by selecting links

(state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their uses."

Motivation to define REST is to collect successful characteristics of the Web. There is no such term as REST toolkit or REST standard. It is just an architecture, which you can follow during the designing of your Web services. Examples of REST based Web services are search and online dictionary services. REST uses standards like HTTP, URL, XML/HTML/GIF, text/xml, text/html, image/gif, etc. The main concept of REST based Web service is a single unifying namespace of URIs to recognize resources. URIs recognizes resources, which are logical objects. These resources are sent across the Web in the form of HTTP messages. The entire Web is now considered as a set of resources. A resource can be the one you like, e.g. ABC corp. has declared a resource 123. Users can access it by typing following URL:

http://www.abc.org/product/123

Suppose ABC, Inc has installed some Web services to allow its customers to get a list of Items, get detailed information about specific Items, and finally submit a purchase order. ABC, Inc provides Web service URL of a list of items to the client. Now the client knows the URL to access a XML document containing a list of items. The Web service used for producing a list of items is transparent to the client. Therefore, the organization can change the implementation of this resource. The XML document received by the client is as follows:

```
<?xml version="1.0"?>
<p:Items xmlns:p="http://www.abc.com"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <Item id="123" xlink:href="http://www.abc.com/items/123"/>
  <Item id="124" xlink:href="http://www.abc.com/items/124"/>
  <Item id="125" xlink:href="http://www.abc.com/items/125"/>
  <Item id="126" xlink:href="http://www.abc.com/items/126"/>
</p:Items>
```

The XML document has links to retrieve detailed information about each item. We have URLs for a specific Item. From these URLs we can get detailed information about a particular item. Suppose that the URL is `http://www.abc.com/Items/123`, and then the response document received by client is as follows:

```
<?xml version="1.0"?>
<i:Item xmlns:p="http://www.abc.com"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <Item-ID>123</Item-ID>
  <Name>Bag</Name>
  <Description>This bag is manufactured within organization abc </Description>
  <Specification xlink:href="http://www.abc.com/items/123/specification"/>
  <UnitCost currency="USD">10</UnitCost>
  <Quantity>10</Quantity>
</i:Item>
```

Each response document enables the client to get more information about that particular item. Finally, the user submits a purchase order, which is in the form of XML document, such as `PO.xml`, to the Purchase Order service on the server. After submitting a purchase order, the Purchase Order service also provides a URL so that the client can interact with the service. With the help of this URL, the client can update or edit purchase order later. Hence, the Purchase Order service is shared between the client and server.

Let's now discuss the characteristics of REST based Web services.

REST Web Services Characteristics

Web services based on REST architecture encapsulate a lot of functionality. Here are the characteristics of REST:

- Pull based interaction between the client and server means that the client actively requests changes on server-side, based on some time interval. The client will not automatically get recent changes about some activity continuously changing on server-side. In this interaction, the user interface matters are kept separate from data storage matters. This enhances portability across various platforms
- Stateless nature means that every request from client to server has information required to understand the request and the session state is maintained on client-side

- Responses should be able to cache. Benefit of caching is that it removes the need for further interactions and results in greater efficiency and scalability
- All resources are accessed using uniform interface
- Resources can be named using URL
- Client state can be changed from one to another

REST System is divided into layers and each layer cannot see the functionality of other layers, except that of adjacent layers. There are proxy servers between clients and resources for features like security. It permits to increase client functionality by downloading and executing code in applets.

Elements of REST Architecture

REST does not consider details of component implementation since it is inclined on roles of components, restrictions on their mutual interaction, and representation of important data elements. It includes restrictions upon components, connectors, and data that make base of web architecture.

Data Elements

Unlike the component implementation style where all the data is encapsulated within and hidden by the processing components, the nature and state of data elements is a key aspect of REST. When a link is selected, the information needs to be moved from the location where it is stored to the location where it will be used, in most cases, by a human reader. An architect can do three things with data:

- Display data where it is found and send its image to the receiver.
- Encapsulate data and rendering engine and transfer both to the receiver
- Send raw data with metadata

But each action is not sufficient. REST provides a combination of these actions by understanding shared features of all data types and provide limited common interface. REST components exchange representation of a resource in a format of standard data types. This interface decides whether the representation is in the same format as that of raw source. It permits information hiding through common interface and hence allows the use of encapsulation. Table 34.4 lists the elements of REST:

Table 34.4: Data Elements of REST

[illegible]	[illegible]
Resource	Target of hyperlink
Resource identifier	URL, URI
Representation	JPEG image, HTML document
Representation metadata	Last modified type, content type
Resource metadata	Alternatives, source link
Control data	If-modified-since, cache control

Now let's learn about all these data elements in more detail.

Resource

Resource can be any information like image, person, service, a set of other resources, etc. Resources can be static or dynamic based on the value set investigated after their creation at any time. Semantics of resource play a role in differentiating one resource from another. Version of software product is a dynamic resource because it keeps changing.

Resource Identifiers

As the name indicates, a resource identifier is the identity of resource to recognize it in interaction between various components. The naming authority takes care of keeping semantic validate of resource over time. The quality of identifier depends upon the amount of money spent to maintain its validity.

Representations and Representation metadata

Representation is a sequence of bytes in addition to its metadata to explain those bytes. Representations are used to gain current or expected state of resource. Metadata is in the form of name/value pairs where name is a standard of semantics of value.

Resource metadata

It is the information related to a resource, but not specific to representation. Representation of metadata and resource metadata comes along with response messages.

Control data

It defines what the message between components is meant for. It is used to override the default functionality of connectors, e.g. Cache property can be changed with control data. A representation can tell us about the input data within the user's query form or error status for a response according to control data.

Connectors

REST uses connectors for the processes, like manipulating resources, and exchanging representations. These connectors use general interface for communication between components, which means that the underlying implementation can be changed without affecting users. They also manage network communication to enhance responsiveness. Table 34.5 lists the REST connectors:

Table 34.5: The REST Connectors

Connector	Example
Client	Binwww
Server	Apache API
Cache	Browser cache
Resolver	DNS lookup, bind
Tunnel	SSL

Stateless nature of REST interactions has many benefits, such as decrease in consumption of resources, let interactions process in parallel, intermediary can understand and view request separately and finally reusability of cached response. Connector interface in parameters may involve request control data, identifier of target, etc. The in parameters are made up of request control data and identifier of destination of request. The out parameters are made up of response control data and optional representation.

The client makes a request to begin communication and the server responds to the request and listens to connections. A cache connector is placed inside the interface to client or server connector. It is implemented inside address space of either client or server connector. Cache on client-side is used for network communication and on server for buffer activity of producing a response. A resolver converts the complete resource identifier or part of them into IP address to create connection between components. The URI contains DNS name to recognize naming authority for the resource. The last connector type is tunnel that circulates communication across connection boundary. It disappears when the communication ends on both the ends.

Components

Components are classified on the basis of their role in application action. An origin server uses a server connector to handle namespace from a requested resource. It provides general interface to its services. Table 34.6 lists the REST components:

Table 34.6: The REST Components	
Component	Example
Origin server	Microsoft IIS
Gateway	CGI
Proxy	Netscape proxy
User agent	Internet explorer

A proxy acts as an intermediary interface to provide services, like data translation, performance enhancement, and security enforcement. A Gateway is known as reverse proxy. A user agent makes use of client connector to begin a request.

Rules of REST Web Service Design

The designer takes two approaches to design something. The first approach is to start from a blank slate and create an architecture till it includes all the requirements of the proposed system. The second approach is to start from system requirements and then incrementally apply restrictions to elements of system. The second approach focuses on understanding of system context. REST is an architectural style of web. If the following principles are followed, the User services perform well in web context:

- Recognize all of logical entities such as Items list and detailed item information.
- Each resource is named using a noun URL, e.g. `http://www.abc.com/items/123`
- Classify resources on the basis of whether users can only access the document or are able to modify it.
- Resource representations should be free from side effects.
- Add hyperlinks to get information in details.
- Data to be exposed is designed gradually.
- Specify the format of response data using a schema.
- The methods, which are used for invoking Web services, are explained in detail.

REST Views

By now you must have understood the elements of REST architecture and its design principles. Now let's discuss how elements build up this architecture. There are three types of views used to design principles of REST.

Process View

The process view depicts interaction relationships between components from the path of flow of data. Separation from data storage concerns make component implementation easier, decreases complexity of connector semantics and enhances performance tuning, and scalability of server components. Hierarchical system let proxies, gateways, and firewalls play their role during communication at some point that helps in translation and shared caching.

Connector View

It focuses on an activity of communication between components. To choose a suitable communication mechanism, the client connectors investigate the resource identifier. In case the identifier is a local resource, the client may configure with an annotation filter. REST restricts interface between components and scope of interaction, but does not constrain communication to a specific protocol. Interaction with services offered on servers using protocols ftp and gopher, restrict to semantics of REST connector.

Data View

This view exposes the application state as data flows through the components. It sees an application as structured information and control alternatives by which the user does a required task, e.g. an application for searching for a word in online dictionary. Interaction among components is in the form of messages having

dynamic size. For control semantics, small grain messages are used. For the rest of the application, big grain messages are used.

REST gathers control state into the representations received. The pending requests and topology of connected components define the application state. Application is in a steady state when there are no pending requests and responses to all recent requests are received properly. Received user performance is measured using latency among steady states. Latency is defined as the duration of time between the click on hypermedia link on a web page and the point when the required information for the next page has been displayed. The application state may consist of representations from various servers and is handled by user agent.

Design Decisions

Web service technologies enable you to make interoperable interface for a new or an existing application. Designing Web service for an application is kept separate from designing Business logic of the application. Consider a functionality of Web services, in general, before we look at the issues involved in designing Web service. A typical Web service performs the following tasks:

- Make an interface that clients access to make requests to service
- Publish the service details
- Receive requests from clients
- Build and send response to client
- To accomplish these tasks, the following steps are used for designing a Web service:
- Decisions about making interface for clients include that interface must specify the type of requests by clients to use the service. Other decisions include type of end points, use of SOAP handlers, and effects of design decisions on interoperability.
- Decisions on publishing (to make your service available to clients) the interface include that Web service providers can either restrict their Web services to authorized clients or make it public and register it with public registry.
- Design your service to receive a request from the client.
- Decide how to forward the request to Business logic.
- Decide how Business logic handles the request. There are many factors to be considered before sending a response to the client.

Recover from exceptions

Now let's consider Web service in terms of high level layers as shown in Figure 34.1. Service interaction layer makes end point interface for clients, publishes Web services, and forwards request to Business logic. It also handles format incompatibility through mapping between documents from clients, e.g. XML, and that of Business logic, e.g. objects.

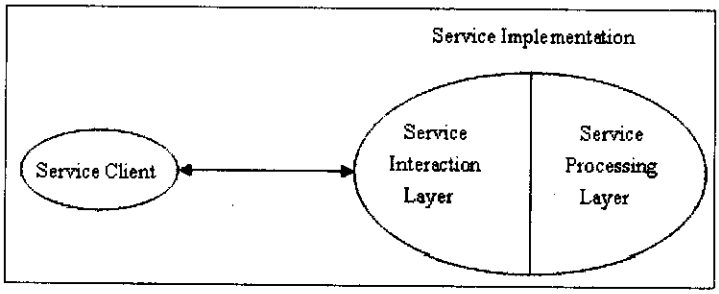


Figure 34.1: Showing Interaction between Service Client and its Implementation

The Service processing layer has Business logic to process requests from service client. It takes care of integration with EIS and another Web service. This partition into layers divides responsibilities and encapsulates pre- and post- processing logic into Service interaction layer.

Service Interaction Layer Design

All clients manipulate the service through entry point interface of service interaction layer. This layer has major responsibilities, like receiving client requests, forwarding requests to suitable Business logic, and making and sending responses. Let's now discuss the various responsibilities that this layer takes.

Designing the Interface

Interface definition of a Web service can be developed in two ways:

- **Java to WSDL** – Begin with collection of Java interfaces of the Web service and creates WSDL of service for clients to use.
- **WSDL to Java** – Begin with a WSDL document having details of Web service interface and use it to create Java interfaces.

Interface Endpoint Type

It depends upon whether the Business logic of the service is nested within web tier or EJB tier. Use JAX-RPC service endpoint in case of web tier, otherwise use EJB service endpoint. Some more points that should be considered while choosing an endpoint are as follows:

- An EJB service endpoint is implemented as a stateless session bean and EJB container serializes the requests. So there is nothing to worry about. For the JAX-RPC endpoint, there requires hard-coded synchronization.
- EJB container handles concurrent client access, but JAX-RPC need to hard code it.
- JAX-RPC executes in a Web container. Here, we cannot start the transaction by declarative means, so we have to use JTA in this case. But the EJB container has container-transaction element for transactional context.
- Different Web service's methods are accessed by different clients. For this, we need to consider the method level access permissions.
- Since JAX-RPC runs in the Web container, it has access to `HttpSession` object. This object can be used to store the client state. On the other hand, EJB service endpoint has no access to Web container state.

Service Particles

Designing a Web service interface includes determining each method's parameter, return values, and generate errors. Business processes that exchange documents result in a coarse grained interface. Coarse grained operations result in lower network overhead and less flexibility. Finer grained operations do exactly the opposite. The design should be a mixture of coarse grained and finer-grained operations. Try to consolidate related operations into a single Web service operation.

Types of Parameters

For designing Web service interface methods, select types of parameters with caution. Figure 34.2 explains how binding takes place between various types of parameters:

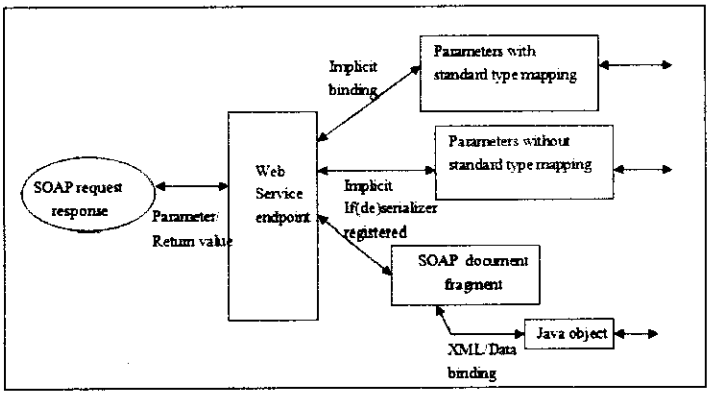


Figure 34.2: Binding Parameters and Returning Values in case of JAX-RPC

Both the method call and its parameters are sent in the form of SOAP message between the client and service. Serialization class is created for conversion among XML and Java representation. SOAP request parameters converted using standard type mapping are bound implicitly. For parameters of method calls in Web service interface, parameters with standard type mappings are selected. Parameters converted with explicit binding need no standard type mapping. Serializers and non-serializers support non-standard type mappings. Alternative to this is to pass parameters as SOAP document fragments in service endpoint interface. Parameters should be mapped to XML and Java objects at appropriate times. There are cases like business-to-business transactions in which XML documents are passed as parameters.

Interfaces with Overloaded Methods

You can overload methods in service interface. For this, there are two approaches used. Consider an example of news services. In a news service, the client can search information by zip code or city name. In case of Java to WSDL, NewsService interface looks like the one shown in the following code:

```
public interface NewsService extends Remote {
  public String getNews(String city) throws RemoteException;
  public String getNews(int zip) throws RemoteException;
}
```

After defining the interface, we run vendor provided tool to create WSDL from interface. The wscompile tool of J2EE 1.4 SDK creates WSDL from NewsService interface. Listing 34.11 has four messages. The first two messages are request and response parts of first SOAP message and the second two messages correspond to the request and response parts of the second SOAP message. The code given in Listing 34.11 shows the use of WeatherService interface with JAVA to WSDL approach:

Listing 34.11: The WSDL File

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="NewsWebService" .......>
  <types/>
  <message name="NewsService_getNews">
        <part name="int_1" type="xsd:int"/>
  </message>
  <message name="NewsService_getNewsResponse">
        <part name="result" type="xsd:string"/>
  </message>
  <message name="NewsService_getNews2">
        <part name="String_1" type="xsd:string"/>
  </message>
  <message name="NewsService_getNews2Response">
        <part name="result" type="xsd:string"/>
  </message>
  ...
</definitions>
```

The second approach is WSDL to Java. Listing 34.12 shows WSDL file for WeatherService interface with overloaded methods avoided. The following WSDL file includes methods `getWeatherByZip` and `getWeatherByCity` which take integer and string parameter, respectively. The code given in Listing 34.12, shows code for the WSDL file:

Listing 34.12: The WSDL File

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="NewsWebService" ...>
  <types/>
  <message name="NewsService_getNewsByZip">
        <part name="int_1" type="xsd:int"/>
  </message>
  <message name="NewsService_getNewsByZipResponse">
        <part name="result" type="xsd:string"/>
  </message>
  <message name="NewsService_getNewsByCity">
```

```
        <part name="String_1" type="xsd:string"/>
    </message>
    <message name="NewsService_getNewsByCityResponse">
        <part name="result" type="xsd:string"/>
    </message>
    ...
</definitions>
```

Each message in the code has a unique name. These different message names convert into overloaded method calls in the interface.

Handling Exceptions

Similar to Java application, Web service application may generate exceptions and these should be caught.

For a Web service, exceptions are also sent to client as part of SOAP messages just like the requests and responses. There are two types of errors thrown by Web services applications—irrecoverable and application-specific. The cause of irrecoverable system error is failure in network connection, and the application-specific error can be anything related to the application. An example of application-specific error in NewsService scenario is `ZipcodenotFoundException`. This kind of error is known as Checked Exception.

The SOAP documentation has a message type fault that allows exceptions to be passed as part of SOAP message. A SOAP fault denotes system-level exceptions, such as `RemoteException`. The WSDL fault denotes service specific exceptions.

Interoperability

WS-I stands for Web services interoperability organization and helps in achieving interoperability between different platforms. WS-I basic profile adheres to standards SOAP, WSDL, UDDI and XML.

WSDL uses two types of messaging styles—the rpc and the document. The `rpc` message style specifies RPC based operation where messages contain function signatures. The document style means operation is document-based where messages contain documents.

WSDL supports two other mechanisms—literal and encoded. If the value of use attribute is literal, it means data is formatted in accordance with the general definitions of WSDL document. If the value is encoded, then the data is formatted on the basis of encodings in URI specified by `encodingStyle` attribute.

How to Receive Requests

In Web service calls, passed parameters can be Java objects or XML documents. In case of Java objects, Web service needs to perform application-specific validation. For XML documents, service endpoint must validate incoming XML document against its schema. If the processing layer handles only objects and the interface accepts XML documents, then map XML documents to objects, as shown in Figure 34.3:

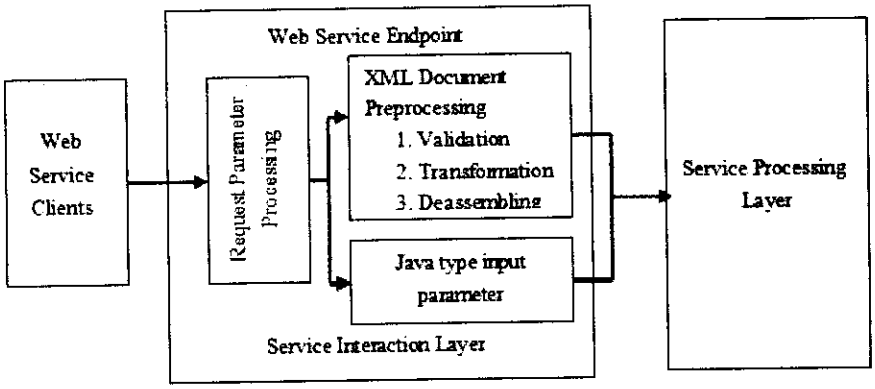


Figure 34.3: Handling Requests in Web Service Interaction Layer

In Figure 34.3, Service endpoint acts as an entry point when handling requests and security techniques; so that only authorized users, which have access rights to the published service, can access service. As said earlier, parameters for Web service method calls can be XML documents or Java Objects. XML documents are validated, transformed, and finally mapped to application specific objects before sending them to the Service Processing Layer. Any exceptions raised, during validation and transformation of XML documents, are handled in the Service Interaction Layer.

How to Delegate Requests

Requests are classified according to the time they take. They are broadly classified into two categories. The first category includes requests that take very less time to process, and therefore, the client may switch to wait state for receiving a response before proceeding further. The second category of requests is those requests that take long time in processing. In this case, the client would not like to wait till the processing is complete.

How to Formulate Response

After the Business logic finishes its processing, the response is generated by constructing the method call return values and output parameters in interaction layer, as shown in Figure 34.4:

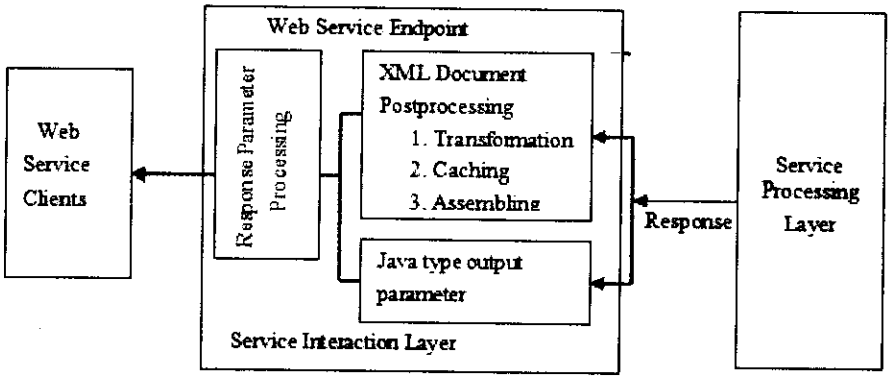


Figure 34.4: Web Service Response Processing

Response assembly and XML document transformations occur in one location in case the document returned to the caller has different schema from internal schema. Data caching is implemented by keeping the functionality near endpoint. Each service will render responses in formats supported by different client types.

Processing Layer Design

Business logic in this layer applies to Web service request. Business logic design issues include where to perform Logic—either in Web tier or in EJB tier, Bean managed persistence or Container managed persistence, etc. Keep processing the layer independent of the interaction layer in order to support a variety of clients, such as web clients, Web service clients, and simple java client. Bind XML documents to Java objects in the interaction layer when the service Business logic does not require to work on XML document. This process occurs in the interaction layer and, hence, the processing layer can support different types or versions of XML schema.

Other Technologies

After discussing Web service related terms and REST architecture, we will concentrate on how Java SE 6 has introduced new APIs and tools to support the creation and implementation of Web Services.

Java API for XML based RPC (JAX-RPC)

Java API for XML-based RPC (JAX-RPC) is a Java API used for developing and using Web services. A collection of procedures that can be called by a remote client over the Internet is an RPC-based Web service. A client written in a language other than Java programming language can access a Web service developed and deployed on the Java platform with JAX-RPC. Conversely, a client written in Java programming language can

communicate with a service that was developed and deployed using some other platform. It is JAX-RPC's support for SOAP and WSDL that makes this interoperability possible. SOAP defines standards for XML messaging and the mapping of data types so that applications adhering to these standards can communicate with each other. JAX-RPC is based on SOAP messaging. A JAX-RPC remote procedure call is implemented as a request-response SOAP message. JAX-RPC's support for WSDL is the other key to interoperability.

Java Architecture for XML Binding (JAXB)

Java Architecture for XML Binding (JAXB) renders a convenient way to bind an XML schema to a representation in Java code. You can easily incorporate XML data and processing functions in applications based on Java technology without having to know much about XML itself.

JAXB is basically a Java technology that enables you to generate Java classes from XML schemas by means of a JAXB binding compiler. The JAXB binding compiler takes XML schemas as input, and then generates a package of Java classes and interfaces that reflect the rules defined in the source schema. These generated classes and interfaces are in turn compiled and combined with a set of common JAXB utility packages to provide a JAXB binding framework. The JAXB binding framework renders methods for unmarshalling XML instance documents into Java content trees and for marshalling Java content trees back into XML instance documents.

Java API for XML Messaging (JAXM)

The Java API for XML Messaging (JAXM) provides a standard way to send XML documents over the Internet from the Java platform. It is based on SOAP 1.1, which defines a basic framework for exchanging XML messages. By adding the protocol's functionality on top of SOAP, JAXM can be extended to work with higher level messaging protocols, such as the one defined in the ebXML (electronic business XML) Message Service Specification.

Java API for XML Registries (JAXR)

To access standard business registries over the Internet, the Java API for XML Registries (JAXR) provides a convenient way. Business registries are often described as electronic yellow pages because they contain listings of businesses and products or services the businesses offer. JAXR gives developers writing applications in the Java programming language, which is an uniform way to use business registries that are based on open standards (such as ebXML) or industry consortium-led specifications (such as UDDI).

Relationship to Other Java APIs

- When we implement JAXR, the other Java APIs can also be used which are necessary to complete the required implementation of the scenario.
- Communication between the JAXR providers and registry providers is through SOAP based RPC, like interface (for example, UDDI). Here, the Java API for XML-based RPC (JAX-RPC) API can be implemented by the JAXR providers.
- For the communication with the registry providers which export XML Messaging-based interface (for example, ebXML), the Java API for XML Messaging (JAXM) can be used in the JAXR providers implementations.
- Processing of XML content retrieved from or submitted to the registry is required by both JAXR providers and JAXR clients. So the Java API for XML Processing (JAXP) and Java Architecture for XML Binding (JAXB) can be used while implementing JAXR providers and JAXR clients.

Java API for XML Parsing

The Java API for XML Processing (JAXP) is used to parse, transform, validate, and query XML documents using an API that is independent of a particular XML processor implementation. JAXP is a standard component in the Java platform. The Java API for XML Processing (JAXP) is for processing XML data using applications written in the Java programming language. JAXP leverages the parser standards Simple API for XML Parsing (SAX) and Document Object Model (DOM) so that you can choose to `parse` your data as a stream of events or build an object representation of it.

Till now we have learnt that Web services is a latest technology that can be used to solve the problem of calling methods on one machine from the other machine where both the machines are using different hardware and software. Now, we will discuss how to create Web service based on JDK SE 6 platform and invoke this Web service with a Web browser based AJAX client.

Creating a Web Service

Let's create a sample Web service and understand how simple and easy it is to develop a Web service using APIs introduced with Java SE 6. The Web service stack provided with JDK 1.6 will be used to create this sample Web service. The JAX-WS tools, like wsimport and wsgen, are part of JDK now. The deployment of Endpoint API is now simplified with the HTTP server, which is included in JDK. For all data binding, the JAXB 2.0 is used which is also a part of Web service API stack. The message processing is done using StAX (Streaming API for XML) and SAAJ (SOAP with Attachments APIs for Java 1.3). The StAX is a Java API containing interfaces that can be implemented by more than one parser.

Creating Web Service EndPoint

A sample application includes a Web service interface and a Web service class which implements a Web service interface. Alternately, we can have a Web service class directly which can be declared using the annotation @WebService. A publisher is created to publish the service on the server and finally, a Web service client is created which can access the Web service and invoke some service operations provided by the Web service.

We have combined the logic of creating a Web service and publishing it on the server in a single class. We start with POJO which is decorated with @WebService. The use of annotation declares that it is going to be a Web Service endpoint.

All the methods which we want to expose as Web service operations are annotated with the @WebMethod annotation. Listing 34.13 demonstrates simple Web service endpoint. The code given in Listing 34.13shows code for creating `MyService.java` file (you can find this file in the Code/AJAX/Chapter 34/web/service folder on the CD):

Listing 34.13: The MyService.java File

```
package service;
import javax.jws.*;
import javax.xml.ws.Endpoint;
@WebService
public class MyService
{
  @WebMethod
  public int sum(int a, int b)
  {
        return a+b;
  }
  @WebMethod
  public String message()
  {
        return "Hello From Web Service!";
  }
  public static void main(String[] args)
  {
  //Creating and Publishing Web Service.
  MyService serv = new MyService();
  Endpoint endpoint = Endpoint.publish("http://localhost:8081/myservice", serv);
  }
}
```

The class is annotated with `@WebService` annotation. The `main()` method of this class is implementing the logic of creating and publishing of Web service on the HTTP server and port number 8081. Save this file in a folder, c:\web\service.

To build and publish this Endpoint, run the following two commands:

The `apt` (Annotation Processing Tool) command is a command line utility used for annotation processing. The things which we escaped from implementing using annotation are implemented by this tool and the related components are generated. Run `apt` command and generate the required wrapper classes like this:

```
C:\web>apt -d sample service/MyService.java
```

Execute the following command to publish this Web service:

```
C:\web>java -cp sample service.MyService
```

Make sure that the c:\web\sample folder is created before these commands are executed. Now we have our Web service endpoint created and deployed. We can access the deployed WSDL by typing the following URL in the Web browser:

```
http://localhost:8081/myservice?wsdl
```

The code given in Listing 34.14 shows you the WSDL of the deployed Web service endpoint:

Listing 34.14: The WSDL File

```
<?xml version="1.0" encoding="UTF-8" ?>
- <definitions xmlns=http://schemas.xmlsoap.org/wsdl/ xmlns:tns="http://service/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" targetNamespace="http://service/"
  name="MyServiceService">
- <types>
- <xsd:schema>
<xsd:import schemaLocation=http://localhost:8081/myservice?xsd=1
  namespace="http://service/" />
  </xsd:schema>
  </types>
- <message name="sum">
  <part element="tns:sum" name="parameters" />
  </message>
- <message name="sumResponse">
  <part element="tns:sumResponse" name="parameters" />
  </message>
- <message name="message">
  <part element="tns:message" name="parameters" />
  </message>
- <message name="messageResponse">
  <part element="tns:messageResponse" name="parameters" />
  </message>
- <portType name="MyService">
- <operation name="sum">
  <input message="tns:sum" />
  <output message="tns:sumResponse" />
  </operation>
- <operation name="message">
  <input message="tns:message" />
  <output message="tns:messageResponse" />
  </operation>
  </portType>
- <binding name="MyServicePortBinding" type="tns:MyService">
  <soap:binding style="document"
 transport="http://schemas.xmlsoap.org/soap/http" />
- <operation name="sum">
  <soap:operation soapAction="" />
- <input>
  <soap:body use="literal" />
  </input>
- <output>
  <soap:body use="literal" />
```

```
    </output>
    </operation>
  - <operation name="message">
    <soap:operation soapAction="" />
  - <input>
    <soap:body use="literal" />
    </input>
  - <output>
    <soap:body use="literal" />
    </output>
    </operation>
    </binding>
  - <service name="MyServiceService">
  - <port name="MyServicePort" binding="tns:MyServicePortBinding">
    <soap:address location="http://localhost:8081/myservice" />
    </port>
    </service>
    </definitions>
```

No deployment descriptor is required and no starting of some container is required. This is all done by JAXWS, which uses HTTP server available in JDK 1.6.

Home Page for Application

The `index.html` is the home page for the application. It has a simple HTML form, which consists of three text fields having names a, b, and result. When the `onClick` event triggers, it invokes the `callAdd` method which extracts values from the first two text boxes, converts them to integers, and then stores into two variables. Now the URL to request the Web service client is made using these variables as query parameters. The code given in Listing 34.15 shows the home page for this application (you can find the index.html file in the Code/AJAX/Chapter 34/MyServiceClient folder on the CD):

Listing 34.15: The index.html File

```
<html>
<head>
  <title>Ajax webServices Example</title>
  <script language="Javascript">
  function postRequest(strURL) {
        var xmlHttp;
        if(window.XMLHttpRequest){ // For Mozilla, Safari, ...
        var xmlHttp = new XMLHttpRequest();
        }
        else if(window.ActiveXObject){ // For Internet Explorer
        var xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
        }
        xmlHttp.open('POST', strURL, true);
        xmlHttp.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
        xmlHttp.onreadystatechange = function(){
              if (xmlHttp.readyState == 4){
                    updatepage(xmlHttp.responseText); }
        }
        xmlHttp.send(strURL);
  }
  function updatepage(str) {
        document.getElementById('result').value = str;
  }
  function callAdd() {
        var a = parseInt(document.f1.a.value);
        var b = parseInt(document.f1.b.value);
        var url = "add.jsp?a=" + a + "&b=" + b + "";
        postRequest(url);
```

```
    }
</script>
</head>
<body>
<h1 align="center"><font color="#000080">Ajax WebServices Example</font></h1>
  <form name="f1">
  <input name="a" id="a" value="">
  <input name="b" id="b" value="">
  <input name="result" type="text" id="result">
  <input type="button" value="ADD" onClick="callAdd()" name="showadd">
  </form>
</body>
</html>
```

Now the `postRequest` method is called with this URL as argument. This method creates an instance of `XMLHttpRequest` object and then the Request is made to the Web service client using this object's `open` and `send` methods. When the server response is completed, the `updatepage` method populates the third text field with the sum of the two entered numbers

Invoking a Web Service

Now, we will develop a client to access this Web service and invoke the exposed Web service operations. The `wsgen` and `wsimport` are Java Web Services tools available with JDK and are used to generate JAX-WS portable artifacts.

Run `wsimport` on the deployed WSDL URL to generate some classes. Then use the `-keep` switch to keep the Java source files for all `.class` files generated and execute the following command (make sure your Web service is deployed):

```
C:\web>wsimport -p client -keep http://localhost:8081/myservice?wsdl
```

See all generated classes and Java source files in the c:\web\client folder. The most important class files are as follows:

- `MyService.java`—Service Endpoint Interface
- `MyServiceService`—Generated Service. It is instantiated to get the proxy

Invoking Endpoint

Now create a client program to invoke the endpoint and execute methods. The `add.jsp` is a Web service client program. It has embedded Java code to invoke the sum method of Web service. The client code creates an instance of the generated service `MyServiceService` and uses this object to get the proxy `myserviceProxy`. This proxy object is used to invoke the exposed Web service operations. The code given in Listing 34.16 shows the code for `add.jsp` page (you can find the add.jsp file in the Code/AJAX/Chapter 34/MyServiceClient folder on the CD):

Listing 34.16: The `add.jsp` Page

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%
try {
  client.MyServiceService service = new client.MyServiceService();
  client.MyService port = service.getMyServicePort();
  int a = Integer.parseInt(request.getParameter("a"));
  int b = Integer.parseInt(request.getParameter("b"));
  int result = port.sum(a, b);
  out.println(result);
  }
  catch (Exception ex1) {
  }
%>
```

Copy the client folder into /WEB-INF/classes directory. Next make a new Web Project and name it `MyServiceClient and deploy` the MyServiceClient folder on Tomcat 6.0 server. Open Internet Explorer and type

http://localhost:8080/MyServiceClient/ on the address bar and enter two numbers23 and 12in the first two text boxes, respectively. Then click the ADD button to populate the sum in the third text field. The result is shown in Figure 34.5.

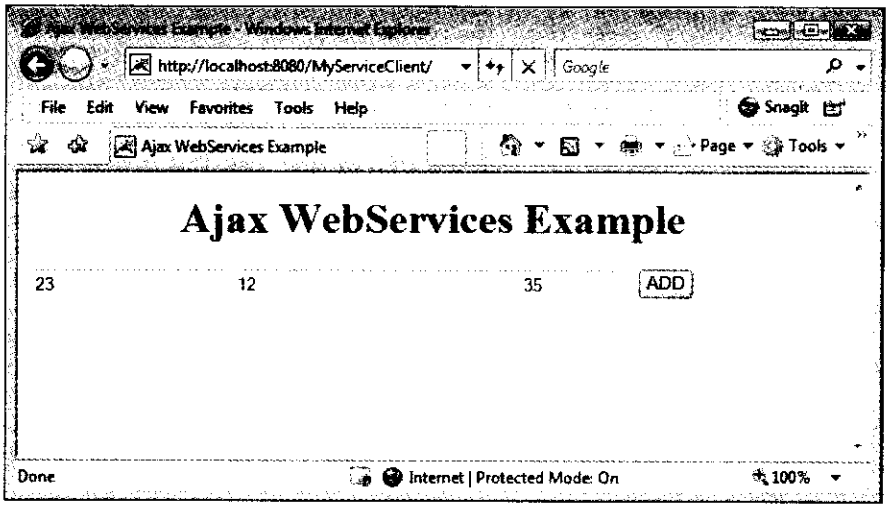


Figure 34.5: Addition of two Numbers using Web services and AJAX

In Figure 34.5, you can see the addition of two numbers.

Cross Domain Web Services

The current style of developing Web applications is the creation of Mashups. This means mixing the content from two different sources. An example of Mashups is a website that combines the content or scripts from many sources. The Mashups applications try to use many Ajax techniques. With the help of them, it becomes sensible to invoke Web service from within your own JavaScript. However, XMLHttpRequest object has one limitation—it is not used to invoke Web services outside its domain. The XMLHttpRequest object doesn't allow calls to be made from code in one domain to a Web service in another. However, these days Mashups are created using APIs, which are made publicly available by companies, such as Google, Flickr, Yahoo, Last.fm, and YouTube. It involves calling Web Services from these APIs.

If you are using Internet Explorer (IE), then the temporary solution for calling cross domain Web services is by configuring the IE as follows:

- Add a trusted site by selecting Tools | Internet Options | Security tab.
- Set the security for the trusted site by selecting Custom level | Miscellaneous section.
- Set the "Access data sources across domains" option to `Enable`. By default, it will be set to `Prompt`.

FireFox doesn't support such type of configuration. Therefore, this solution is not a universal solution.

Another solution for calling cross-domain Web services is getting JavaScript to call a proxy program (either on the client or the server). This proxy program calls the Web service for you. The output can be written to the response stream in the form of text or XML and then it is available using the `responseText` and `responseXML` properties of `XMLHttpRequest`. If your Web service returns JSON (JavaScript Object Notation format), then you can dynamically create a `script` tag and assign the `src` attribute to the location of the Web service. This will work only with JSON data. JSON cannot represent functions or expressions; it can only represent data. It can be easily converted into a JavaScript value, and so is easy to reference via a script. One of the examples of Web services which return JSON data is Yahoo Web services. This solution heavily depends on JSON and so cannot

be used with Web services that don't have a JSON option. Another problem is that the dynamic script doesn't inform whether it is loaded correctly. A further problem is that, in IE, the dynamic loading of scripts stops all other processing that causes a potential memory leak

The preceding discussion shows that there is no one perfect way to call Web services cross domain using Ajax. However, many applications, which invoke cross domain Web services, have been developed. Some of them rely on an ASP.NET server-side proxy as a solution or some prefer the solution that involves JSON.

In the next section, we will discuss how to use Java Web services APIs for developing Web services.

Using APIs

Java Web services Development pack is a software development kit used for developing Web services and other Java applications. In this topic, we are providing a brief coverage on packages that come with Web services developer pack.

javax.xml

It specifies XML constants. These attributes include the prefix to represent default XML namespace, official XML namespace prefix, and specify XML namespace declarations. There are many core XML standards, like extensible markup language v1.1 and v1.0, namespaces, etc. in XML

javax.xml.bind

It makes client applications capable of unmarshalling, marshalling, and validation with its runtime binding framework. The `javax.xml.bind` annotations are used to convert Java objects into XML. Table 34.7 lists the interfaces in the `javax.xml.bind` package:

Table 34.7: Interfaces of the javax.xml.bind Package

Interface	Description
`Element`	It is the element marker interface
`Marshaller`	It controls the process of serializing Java content in the form of trees into XML data
`ParseConversionEvent`	It specifies the error that occurs during conversion of XML data into corresponding Java type
`ValidationEvent`	It specifies the error that occurs during validation in operations, like marshalling, unmarshalling, etc
`Validator`	It is responsible for controlling the validation of content trees during runtime

There is one `JAXBContext` class, which makes client access to JAXB API.

javax.xml.datatype

It provides mapping between XML schema and Java data types. XML schema should have mappings for primitive data types, such as byte, date, integer, long, double etc. JAXB defined mappings for XML schema built-in data types includes:

- `xs:anySimpleType`
- `xs:boolean`
- `xs:byte`
- `xs:decimal`
- `xs:double`
- `xs:float`
- `xs:hexBinary`
- `xs:int`
- `xs:integer`
- `xs:long`

- xs:QName
- xs:short
- xs:string
- xs:unsignedByte
- xs:unsignedInt
- xs:unsignedShort

javax.xml.registry

It contains classes and interfaces of JAXR API. It is formed on ebXML registry information model and also supports registry specifications, such as UDDI. Table 34.8 lists the interfaces of the javax.xml.registry package:

Table 34.8: Interfaces of the javax.xml.registry Package

Interface	Description
CapabilityProfile	It gives details about the capabilities of JAXR provider
Connection	It represents the connection between JAXR client and JAXR provider
DeclarativeQueryManager	It gives the capability to execute declarative queries
FederatedConnection	It denotes a virtual connection to a group of registry providers
JAXRResponse	It denotes JAXR requests and response
Query	It encapsulates a single query in a declarative query language
RegistryService	It is the main interface implemented by a JAXR provider

ConnectionFactory is the base class for creating a JAXR connection.

javax.xml.rpc

RPC means remote procedure calls. This package is used for making function calls and receiving the responses over the network. It contains API classes for programming client. JAX-RPC uses SOAP and HTTP to make remote procedure calls on a network. Table 34.9 lists the interfaces of the javax.xml.rpc package:

Table 34.9: Interfaces of the javax.xml.rpc Package

Interface	Description
Call	It does dynamic invocation of a service endpoint
Service	It represents dynamic proxy for target service endpoint
Stub	It is the base interface for stub classes

javax.xml.soap

This package lets you perform functions, such as create point-to-point connection to particular endpoint, create SOAP messages, add or modify content to header, attachment parts and send SOAP request response message. SOAPPart of SOAPMessage is a DOM-based document and can be accessed using tools and libraries that use DOM. SAAJ APIs are needed to return SAAJ types during examining and accessing DOM tree. SAAJ comes with a package javax.xml.soap, which contains the APIs to handle the SOAP message. SAAJ helps in writing a SOAP messaging application directly, instead of using JAX-RPC. Table 34.10 lists the interfaces of the javax.xml.soap package:

Table 34.10: Interfaces of the javax.xml.soap Package	
Name	It represents an XML name
Node	It represents an element in an XML document
SOAPBody	It represents contents of SOAP body element in a SOAP message
SOAPEnvelope	It represents an envelope containing SOAPHeader and SOAPBody
SOAPHeader	It represents SOAP header element.
SOAPFault	It represents element of SOAPBody object, which describes error information

Table 34.11 lists the various classes of the `javax.xml.soap` package:

Table 34.11: Classes of the javax.xml.soap Package	
AttachmentPart	An object that denotes a single attachment of a SOAPMessage object
MimeHeader	An object that stores MIME header name and its value
SOAPConnection	A Connection object that is used by client to send messages to remote destination
SOAPMessage	It is the root class for all SOAP messages
SOAPPart	An object that stores SOAP related part of SOAPMessage object

javax.xml.transform

It has generic classes for processing transformation instructions and performing transformation from source to result. To pass `Source` and `Result` interface objects to interfaces, `StreamSource` and `Stream Result` classes are used. Table 34.12 lists the interfaces of the javax.xml.transform package:

Table 34.12: Interfaces of the javax.xml.transform Package	
Result	An object that implements this interface contains the information needed to build a transformation result tree
Source	An object that implements this interface contains the information needed to act as source input (XML source or transformation instructions)

Namespaces pose problems when we use XML objects. Qualified names in XML document are prefix names. But these names do not serve as identities of namespaces. It is URI to which prefixes are mapped hold identity. One alternative is creating `QName` object, which consists of all namespace URI, prefix, and local name. To pass namespace values to transformations, use two part string which is made up of namespace URI in curly brackets suffixed by local name. An application can check whether it is null or non-null URI by seeing the first character of this string. For example, for the element defined with `<abc: hav xmlns="http://abc.hav.com/padb/ser.html">`, the qualified name looks like "`{ http://abc.hav.com/padb/ser.html} hav`". Here the prefix is eliminated.

The `javax.xml.transform.dom`, `javax.xml.transform.sax`, and `javax.transform.stream` packages implement DOM, SAX2 and stream specific transformation APIs, respectively.

javax.xml.validation

Validation is a process of verifying that an XML document is an instance of a specified XML schema. The `javax.xml.validation` has API classes for validation of XML documents. Table 34.13 lists the classes of the javax.xml.validation package:

Table 34.13: Classes of the javax.xml.validation Package	
Class	**Description**
SchemaFactory	It is a factory that generates Schema instances
Schema	It is built in representation of grammar used by a XML schema
Validator	It verifies XML document against particular XML schema

The `javax.xml.parsers` and `javax.xml.transform.dom` packages are used in conjunction with validation API. The `javax.xml.parsers` package is used to parse the XML documents and then the `javax.xml.transform.dom` package transforms the parsed XML documents in the form of DOM tree. In the validation process, the first `SchemaFactory` which understands specific types of schema, like WXS schema, is created. Using this factory, the Schema instance is created. Now the newValidator () method of Schema object is invoked and returns an instance of the Validator class. Finally the validate method of Validator class is invoked, which takes an instance of Document object to perform validation. If this method is unable to perform validation, `SAXException` is thrown.

Summary

You have learned Web Services and its relation with AJAX in detail, in this chapter. The chapter started with Web service and then moved towards SOAP, its message elements in detail, syntax of WSDL document and registry standard UDDI. With emphasis on the latest Web services REST architecture in terms of its components and views, we also understood the various design issues that occurs during the design of Web service and the various technologies, like JAXB, JAXR, JAXP, etc. Towards the end of the chapter, we created, implemented, and invoked a Web service using AJAX client.

Quick Revise

Q1. Define Web services.

Ans: Web services present a model by which tasks of e-business processes were distributed widely through Internet. This model is not restricted to a specific business model. Web services are not graphical user interfaces, but can be used into software meant for user interaction. They describe their inputs and outputs in a manner that the second party can predict its functionality, how to call it, and the expected results. The Web services are reusable software components and let the developers reuse basic elements of code made by others. There is a loose connection between these software components, which allow manageable reconfiguration.

Q2. Define SOP.

Ans: Simple Object Access Protocol (SOAP) is a protocol which allows applications to exchange information. Unlike the language or platform specific protocols, such as Internet Inter-ORB Protocol (IIOP) or Java Remote Method Protocol (JRMP), the SOAP is not a language or platform specific protocol. It allows communication between applications running on different platforms. SOAP, unlike the IIOP and JRMP which are binary protocols, is a text-based protocol and uses XML-based rule to allow applications interchange information over HTTP.

Q3. What is Header element. What are the attributes of the Header elements?

Ans: The Header element contains application-related information about SOAP message. It is an optional element and should be the first sub-element of envelope element, if included. The Header element can have attributes, which specify how the recipient processes the message. Header elements act as contracts between the sender and the receiver. The attributes of Header element are as follows:

- actor attribute
- mustUnderstand attribute
- encodingStyle attribute

Q4. Define UDDI.

Ans: UDDI stands for Universal Description Discovery and Integration. It is a directory for storing information about Web services and communicates using SOAP. It uses WSDL to describe interfaces to Web services, e.g. if any industry wants their services to be used by others, it registers its service in UDDI directory. Users then search for the UDDI directory to find the service. When an interface is found, users can communicate with the service.

Q5. Define JAX-RPC.

Ans: Java API for XML-based RPC (JAX-RPC) is a Java API used for developing and using Web services. A collection of procedures that can be called by a remote client over the Internet is an RPC-based Web service. A client written in a language other than Java programming language can access a Web service developed and deployed on the Java platform with JAX-RPC.

Q6. Define JAXM.

Ans: The Java API for XML Messaging (JAXM) provides a standard way to send XML documents over the Internet from the Java platform. It is based on SOAP 1.1, which defines a basic framework for exchanging XML messages. By adding the protocol's functionality on top of SOAP, JAXM can be extended to work with higher level messaging protocols, such as the one defined in the ebXML (electronic business XML) Message Service Specification.

Q7. List the built-in data types of defined by JAXB for XML schema.

Ans: JAXB defined mappings for XML schema built-in data types include:

- xs:anySimpleType
- xs:boolean
- xs:byte
- xs:decimal
- xs:double
- xs:float
- xs:hexBinary
- xs:int
- xs:integer
- xs:long
- xs:QName
- xs:short
- xs:string
- xs:unsignedByte
- xs:unsignedInt
- xs:unsignedShort

Q8. Define the javax.xml.soap package.

Ans: This package lets you perform functions, such as create point-to-point connection to particular endpoint, create SOAP messages, add or modify content to header, attachment parts and send SOAP request response message. `SOAPPart` of `SOAPMessage` is a DOM-based document and can be accessed using tools and libraries that use DOM. SAAJ APIs are needed to return SAAJ types during examining and accessing DOM tree. SAAJ comes with a package `javax.xml.soap`, which contains the APIs to handle the SOAP message.

Q9. List the interfaces of the javax.xml.transform package.

Ans: The interfaces available in the javax.xml.transform package are:

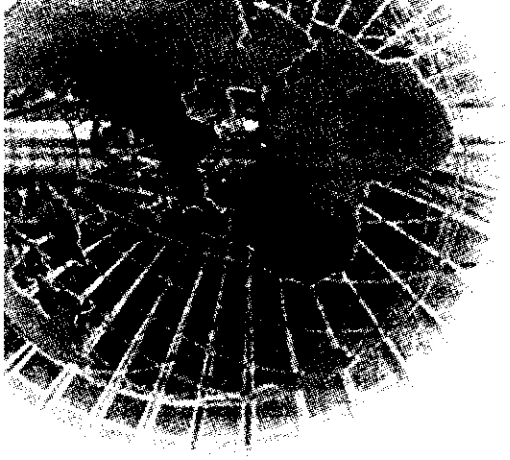
- **Result**—An object that implements this interface contains the information needed to build a transformation result tree.

- **Source**—An object that implements this interface contains the information needed to act as source input (XML source or transformation instructions).

Q10. List the classes of javax.xml.validation package.

Ans: The classes available in javax.xml.validation package are:

- **SchemaFactory**—It is a factory that generates Schema instances.
- **Schema**—It is built in representation of grammar used by a XML schema.
- **Validator**—It verifies XML document against particular XML schema



A

Installing Java Software Development Kit

Java is a programming language, which inherits its object oriented features from C++. Java language was created by a software developer, James Gosling, at Sun Microsystems in 1991. It was first called Oak, which got changed to Java in 1995, when it was first released for public use. The original idea for Java was to create a platform independent and object oriented language that could run on any operating system. Java has a slogan that is 'write once, compile once, and run anywhere'. This slogan means once a program is compiled and run successfully on a platform then that program can easily be executed on any other operating system.

This appendix discusses how to download and install the Java Software Development Kit. Then, we discuss how to create, compile, and run a simple Java program.

Now, let's start with discussing the following features of Java that make it more robust than other languages:

- **Portability**—In the distributed world of Internet, an application developed with the help of a programming language might be accessed on various computers having different kinds of operating systems. But, it's not guaranteed that the application is portable, that is the application runs successfully on all operating systems. To overcome this portability issue, Java introduces a concept of what is known as Bytecode.

  Bytecode is a set of instructions that is generated by Java compiler on compiling a Java program. In other modern programming languages, a program is compiled into an executable code but in Java, a program gets compiled into an intermediate code called Bytecode. This Bytecode then gets executed by Java runtime system called the Java Virtual Machine (JVM). Now, only the JVM needs to be implemented on the system where the Java program has to be executed. JVM is also considered as the Interpreter of Bytecode. In this way, Java has solved the problem of portability.

- **Multithreading**—Java is a programming language designed for the distributed environment of the Internet and for that the concept of multithreading is implemented. This feature helps to write interactive programs wherein multiple tasks can be performed simultaneously; thereby making it a robust programming language.

- **Memory Management**—In Java, all memory management processes are handled automatically. Whenever a program is created, you allocate some memory to the objects used in the program and deallocate (frees) that allocated memory. In Java, you do not need to worry about freeing the memory because Java provides automatic garbage collection that is when the objects are not in use the memory allocated to them will be freed by Java. For example, you have created an array that can store 100 elements, which means you have reserved space for 100 elements. So, when the array completes its functioning and no longer in use then Java frees up the memory allocated to the array. Now, this space can be used by other Java objects. In other programming languages such as C++, you also have to write the code to deallocate the memory, which seems a very tedious process as a programmer needs to remember that memory for which object needs to

be deallocated. Sometimes programmers **deallocate** the memory of the objects that are currently in use and this can harm the process. That's why memory management is a tedious in languages such as a C++ but not in Java because here the memory management is automatic.

- **Security**—Java is a secure language as the programs created in Java are confined to the Java runtime environment that is they can only access that part of your computer hard disk, which is required to execute the program. Java programs are not allowed to access the data outside of Java runtime environment so, downloading Java application through internet won't harm your computer than the applications made in other programming languages can does. That's why Java is a secure language.
- **Distributed**—Java is a distributed language as it can be used to create applications to communicate over the network. Java can communicate over the network because it supports TCP/IP **(Transmission** Control Protocol/Internet Protocol). The TCP/IP protocol is a set of network communication protocol.

The latest release of Java Standard Edition platform is called Java SE 6, Let's now take a look on how to download and install this software package of Java 6. To download Java SDK Standard Edition 1.6, visit Sun Microsystems's website at `http://java.sun.com/javase/downloads/index.jsp`. After downloading the SDK, perform the following steps to install Java SE 6.

1. Run the downloaded SDK executable file. **The Welcome** window **appears,** as shown in Figure A.1:

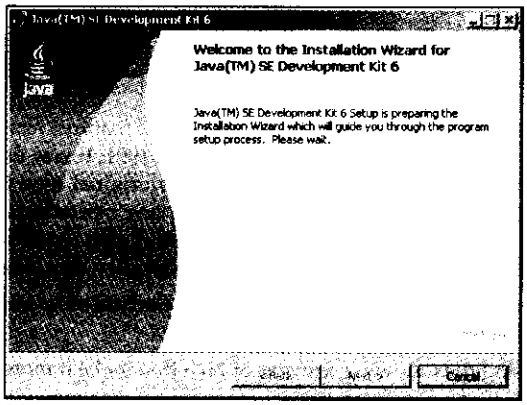


Figure A.1: Displaying the Welcome Window

The welcome window disappears in few moments and a dialog box showing the license agreement appears (Figure A.2).

2. *Click* the Accept button on the License Agreement page to continue the installation process, as shown in Figure A.2:

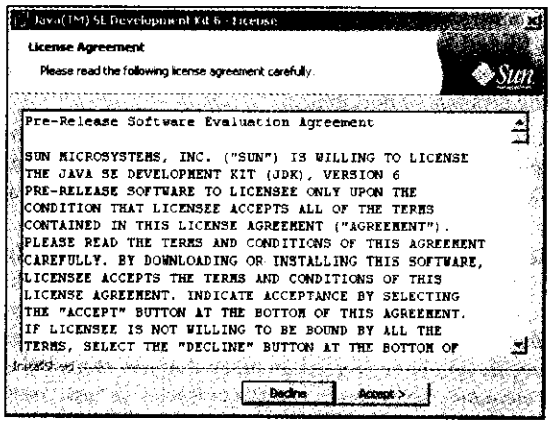


Figure A.2: Displaying the License Agreement Page

The Custom Setup dialog box appears (Figure A.3).

3. Choose a list of components (Figure A.3) that may be installed. Continuing with the Development Tools component, which is selected by default, is sufficient to get started with Java programming.
4. Click the Browse button to change the installation location. In our case, we accept the default installation location.
5. Click the Next button, as shown in Figure A.3:

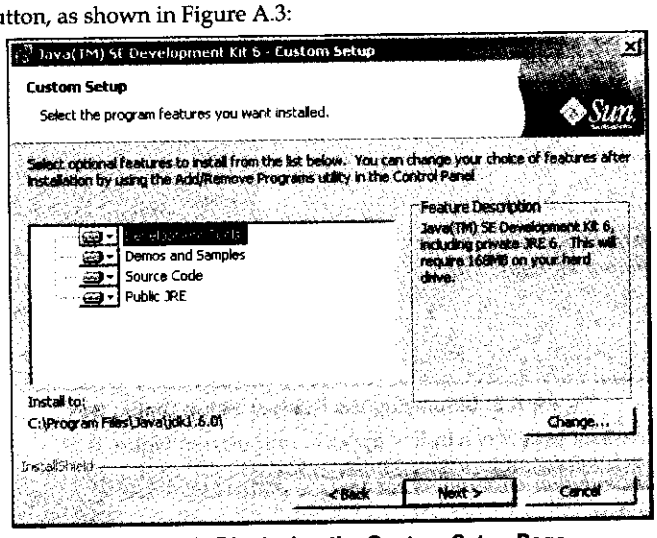


Figure A.3: Displaying the Custom Setup Page

The Installing window appears, as shown in Figure A.4:

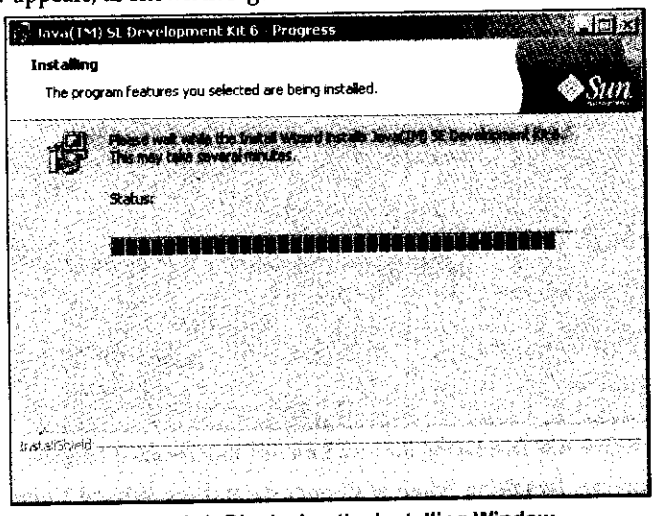


Figure A.4: Displaying the Installing Window

When the progress completes Installing window disappears, Custom Setup dialog box for Java runtime environment appears (Figure A.5).

6. *Click* the Next button to continue the installation process, as shown in Figure A.5:

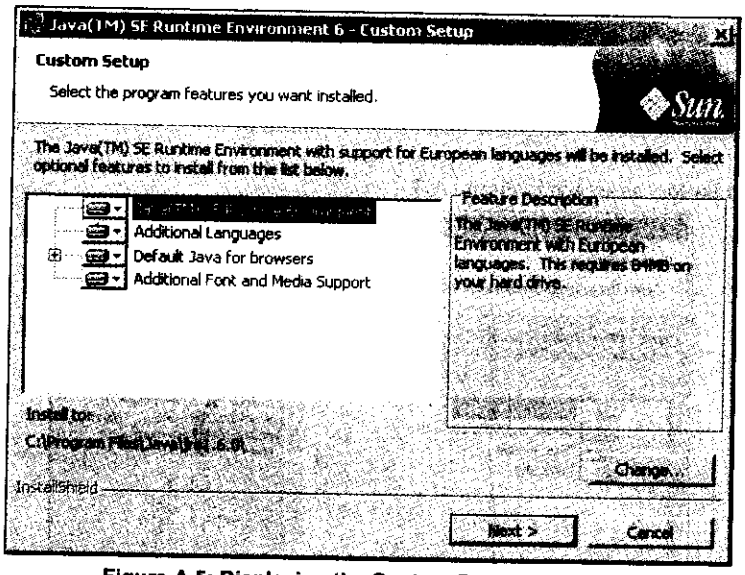


Figure A.5: Displaying the Custom Setup Dialog Box

The Java runtime environment progress window appears as shown in Figure A.6:

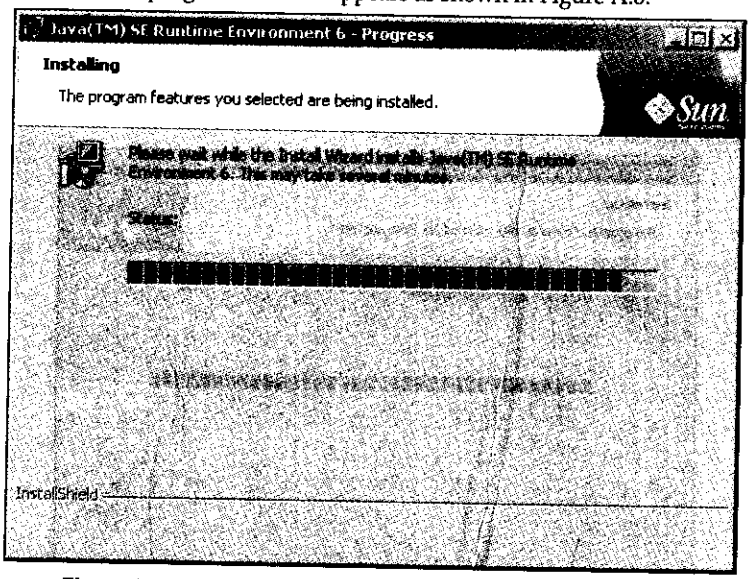


Figure A.6: Displaying the Java Runtime Environment Progress

When the progress completes **Installing** window disappears and the **Wizard Completed** dialog box appears informing you that the installation is completed.

7. Click the **Finish** button on the **Wizard Completed** dialog box to exit from the installation wizard, as shown in Figure A.7:

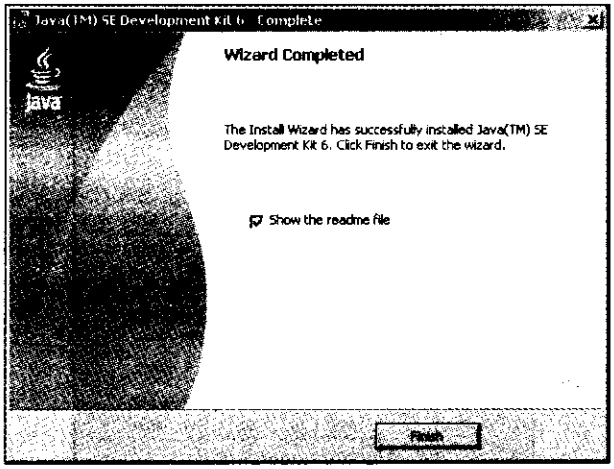


Figure A.7: Displaying the Wizard Completed Page

This is how to install the Java on your system. Now, you can take the advantage of the tools used for compiling and running a Java program (javac and java). The tools used for compiling and running are located in the 'bin' folder of the Java installation directory and these tools are normally operated from a Command Prompt. These tools can however be made available from anywhere on the computer by adding their location to the system path, as shown in the following steps:

1. Select Start→Control Panel→System. The System Properties dialog box appears, as shown in Figure A.8.
2. Click the Advanced tab, as shown in the Figure A.8. The System Properties dialog box appears displaying some advance options including the Environment Variables, as shown in Figure A.8:

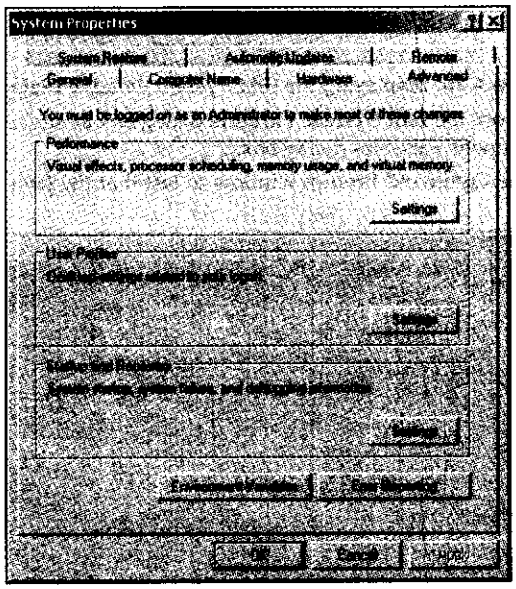


Figure A.8: Displaying the System Properties Dialog Box

3. Click the Environment Variables button, as shown in Figure A.8. The Environment Variables dialog box appears, as shown in Figure A.9.
4. Select the Path Variable given in the System variables, as shown in Figure A.9:

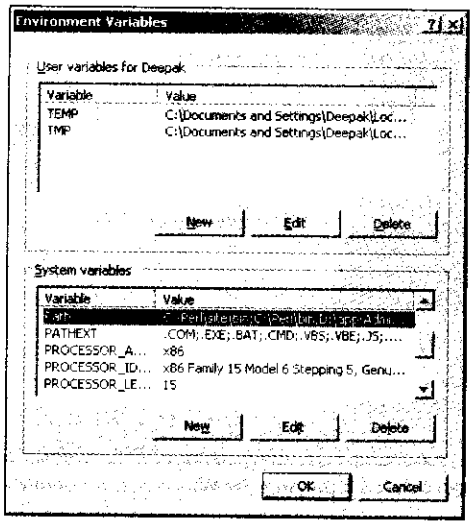


Figure A.9: Displaying the Environment Variables Dialog Box

5. Click the **Edit** button to add the path of the Java bin directory to the end of the list in the Variable value field as shown in Figure A.10:

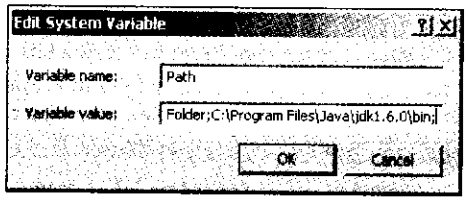


Figure A.10: Displaying the Edit System Variable Dialog Box

6. *Click* the **OK** button to close the **Edit System Variable** dialog box as shown in Figure A.10. And finally click on OK button in Environment Variables dialog box.
7. Type java -version at the Command Prompt window to test that the Java tools can be accessed globally, as shown in Figure A.11:

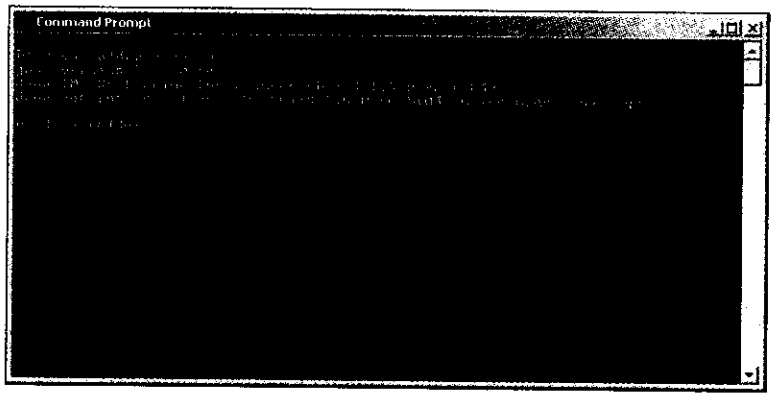


Figure A.11: Displaying the Execution of the Java Command

The output shows that Java has responded correctly to the java-version command, thereby making the Java tools globally accessible. Now, we can continue working with Java. Let's write a Java program.

A Simple Java Program

Let's now create a simple Java program using text editor called Notepad. First of all, put some Java code into the editor such as Notepad to create our first Java program, as shown in the following code snippet:

```
public class HelloWorld
{
  public static void main (String args[])
  {
        System.out.println("Hello World This is our first java program");
  }
}
```

In the above code snippet, the class keyword is used to declare a class called HelloWorld. This name HelloWorld is used as an identifier, which is used to give an appropriate name to a class. Let's take a look at the following line:

```
public static void main (String args[])
```

This is the line at which the program begins executing. It is by calling this `main()` method that all Java programs begin their execution. The line begins with public keyword which is an access specifier, making the `main()` method accessible outside the class in which it is declared.

Static keyword allows `main()` to exist before creating object of the class in which the main method is declared.

void keyword specify that the main ( ) does not return any value.

Now, save the file as `HelloWorld.java` to a desired location on your system. Java programs are saved with the file extension '.java'.

It is to be noted that in Java, name of the file should match with name of the class. Java is a case-sensitive language, that is the word 'hello' and 'Hello' have two different meanings in Java. Once a Java program is written, its time to compile and then run it to show the desired output.

Compiling and Running the Program

To run a Java program, it is necessary to compile it first using Java compiler. This Java compiler is an application named javac.exe located in the Java directory bin folder.

To compile a program, type javac followed by a space and then the name of the file to be compiled, as shown in Figure A.12:

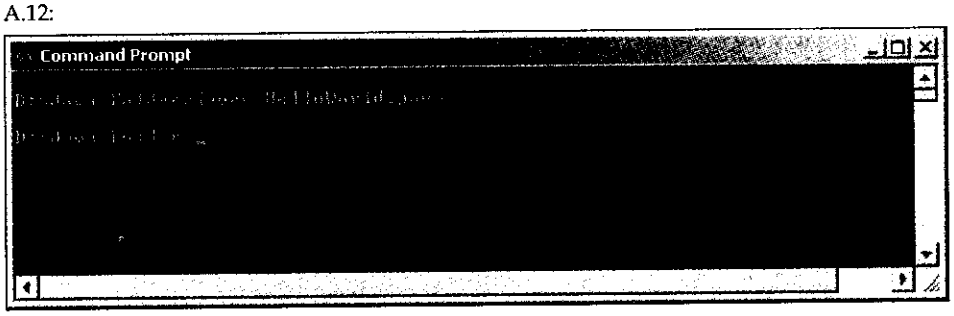


Figure A.12: Displaying the Execution of a Java Program

The errors may occur during compilation due to the following reasons:

- Incorrect syntax
- Inability of compiler to find the source file
- Incorrect file name

Once the source file gets compiled, a new file with the same name as that of the source file followed by an extension '.class' gets created. Now, its time to run this .class file. To do so, SDK provides tool called java.exe, which is an application used to run Java programs. To run the program, type java followed by name of the .class file and press enter to show the output, as shown in Figure A.13:

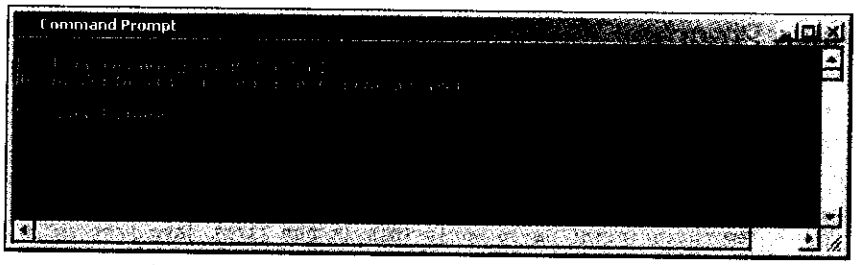
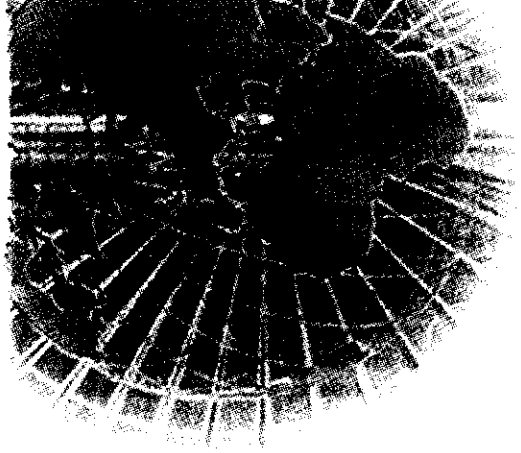


Figure A.13: Displaying the Output of the Java Program

The next appendix describes about the Struts framework.



B

Struts 2–Advancement in Web Technology

In the world of Internet, we require various web-based services for all types of business needs. The software industry is sharply looking at new emerging technologies, design patterns and frameworks, which can bring about an evolution in the development of enterprise-level Web applications. We now have numerous technologies (like Servlet, JSP), different design patterns (like MVC) and various new frameworks to develop a Web application. The name of this new application Framework introduced in this appendix is Struts 2. As you must be aware that a framework is always designed to support the development of an application with its set of APIs, which implement a specific architecture and some reusable components to handle common activities associated with the application. Struts 2 is one such application framework that is about to revolutionize the Web application development.

Struts 2 is not just a revision of Struts 1, it is more than that. Though the working environment, various component designing, configuration, and few other things seem familiar to the Struts 1 developer, the Struts 2 has some architectural differences when compared with Struts 1. In general, Struts 2 Framework implements MVC 2 architecture with centralizing the control using a Front Controller strategy similar to Struts 1, but the basic code of components and their configuration is quite different.

In addition to introducing Struts 2 as a new Web application Framework, we'll get into a detailed discussion of this framework which contains every thing that can make it a preferred choice over other frameworks, like Struts 1. Though Struts 2 is not to replace Struts 1, we now have one more option to choose from while deciding which framework to use for a new Web application development and Struts 2 has all the potential to replace other Web application Frameworks in the industry.

The key features of Struts 2 starting from Interceptors, Results, integration with other popular frameworks and languages through plugins, XWork Validation Framework support, integration with OGNL, and implementation of Inversion of Control (IoC) make this framework stand apart from the other frameworks. We have discussed all these features through this appendix.

Struts 2 is a Web application Framework and will be used for the development of a Web application. This appendix provides a brief discussion over the topics, such as Web application and the technology context which can help you understand architectural decisions taken for the frameworks, like Struts 2. Let's begin our discussion with the term Web application first.

Web Application Framework

All the Web applications have some common features. They execute on a common machine, accept input from common input devices, output data to common output devices, and stores information in a common memory.

An application framework is based on these common aspects and provides a foundation for developing the applications to the developers and alleviates from putting additional efforts for it.

A framework refers to a set of libraries or classes, which are used in the creation of an application. This involves bundling sections of code, which perform a different task, into a framework. With this sort of arrangement, it becomes very easy to design an application by following the framework and reusing those code-sections. The framework that is used for the development of a Web application is known as Web application Framework, such as Struts 1 and Struts 2.

Struts 2 is a Web application Framework which involves tools and libraries for the creation of Web applications. Struts 2 Framework follows Model-View-Controller (MVC) architecture and implements a front controller approach like Struts 1 Framework.

NOTE

Model-View-Controller (MVC) is a pattern used for the creation of Web applications. It solves the problems of decoupling the Business logic and data access code from the Presentation logic. The two different MVC models are MVC 1 and MVC 2. In MVC 1, all requests are handled by JSP, while in MVC 2 it is handled by a Controller servlet. Struts 2 follows the MVC 2 architecture pattern.

Developing a Web application by using Web application Framework, like Struts 2, first requires a layout of the application structure, followed by an incremental addition of framework objects to that structure. This is accomplished by creating the framework tools module. This involves generating framework components by using the wizards and customizing them according to the requirement.

A framework provides some common techniques (Figure B.1) to design an application, which makes the application easier to design, write, and maintain. Figure B.1 shows the common framework strategy:

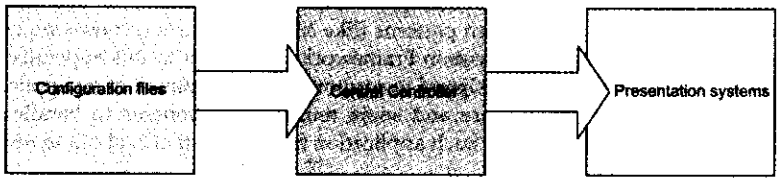


Figure B.1: A Common Framework Strategy

The most common technique that frameworks use includes the following strategies:

- **Configuration files**—Provide the implementation details containing settings specific to an application, such as assembly binding policy, remote objects, etc. Configuration files are read at runtime. These files are external to the application and are not included in the source code.
- **Central Controller**—Provides a way to manage the incoming requests. It receives the requests, processes them, according to a pre-defined logic, and forwards the result to the Presentation systems.
- **Presentation system**—Allows different people to access different parts of the application at the same time. Java developers can work on the Central Controller, while page designers can work on JSPs.

Having understood the concept of a framework, we can now brief that the Web application Framework eases the process of entire Web application development cycle and allows designing complex Web applications in a way that is easily understandable and manageable. The advantages of using a Web application Framework are described as follows:

- **Simplified design and development**—Web application Frameworks greatly simplify the process of development and designing of Web based projects. It provides a foundation, upon which the developers can stand their structures to implement the application very easily. Since the responsibility of developing various application components is given to different members of the team, depending upon their skills, the development process proceeds smoothly.
- **Reduced production time**—The entire Web application is broken down into small components that are developed and tested individually. This sort of arrangement reduces the production time because testing of smaller components requires less time to debug, as compared to testing the entire project at once.

- **Reduced code and effort**—The Web application components that have been tested and debugged, provide a specific functionality can be set aside to be used further when a similar functionality is required in any application. It offers a reduced amount of code and effort involved in the entire development cycle.
- **Improved performance**—Since each component is developed by a specialized person having specialized knowledge and experience in developing similar applications, the best efforts can be provided to application development. The performance of the code improves further by using a powerful set of APIs, provided by the framework, which offers its own features.
- **Easy customization and extension**—In order to customize the application or to extend the capabilities of the application, it is required to make changes in a particular portion of the application, rather than developing an entire new application.
- **Code reuse**—The Web application Framework provides a set of components that are known to work well in other applications. These components can also be used with the next project and by the other members in the organization. New components can also be developed that can be used as reusable code components.

These features of Web application Framework and the way they help in the development of a Web application has made them popular among Web developers. We have a large number of Web application Frameworks available, which are being used in the industry for Web development, like Spring, JSF, Struts 1, WebWork, Struts 2 and many more. Each framework has its own architecture and the way the components are designed and configured are also different. So, instead of having a discussion over the different Web application Frameworks, we can now start our journey with Struts 2.

However, Struts 2 is derived from the WebWork2 Framework which has been in use for the development of Java Web applications. After working for several years, the WebWork2 and Struts communities joined together and created Struts 2. Let's first have a brief introduction of WebWork2 framework.

WebWork2

WebWork2 is an Open Source Framework that is used for creating Java-based Web applications. It is fully based on MVC 2 architecture and is built on a set of Interceptors, Results, and Dispatchers that is integrated on XWork Framework. The View components of WebWork2 include JSP, Velocity, `JasperReports`, `XSLT` and `FreeMarker`. The WebWrok2 Framework provides you the ability of creating your own set of templates by using JSP tags and Velocity macros. It also provides features, such as redirect, request dispatcher results, and multipart file uploading.

WebWork2 provides three features—dispatcher, library of tags to be used in view, and Web-specific results. A dispatcher is responsible for handling client requests by using appropriate actions. By using this technique, the most common way to invoke an action is either via a Servlet or a Filter. The Model is composed of POJO (Plain Old Java Objects) classes. WebWork2 also comes with some built-in JSP tags for creating HTML pages.

History of WebWork 2.0

WebWork 2.0, developed by OpenSymphony, was released in February 2004. WebWork 2.0 is the first release of the complete rewrite of WebWork with implementation on top of XWork. WebWork 2.0 Framework is the result of a combination of earlier versions of WebWork Framework and Xwork Framework. Hence, all the versions of WebWork2 (2.1.x and 2.2.x) support the features of WebWork's previous releases and XWork Framework. XWork is a generic command pattern MVC Framework, which is used to power WebWork as well as other applications. It provides a built-in rich framework, which contains Inversion of Control container, a powerful expression language, validation, data type conversion, and pluggable configuration.

Flow of Execution in WebWork2 Architecture

WebWork2 is based on the Model-View-Controller architecture, which means that the logic is separated from view and the Web application can run on the server by the Controller component. The basic flow of execution in WebWork architecture is shown in Figure B.2:

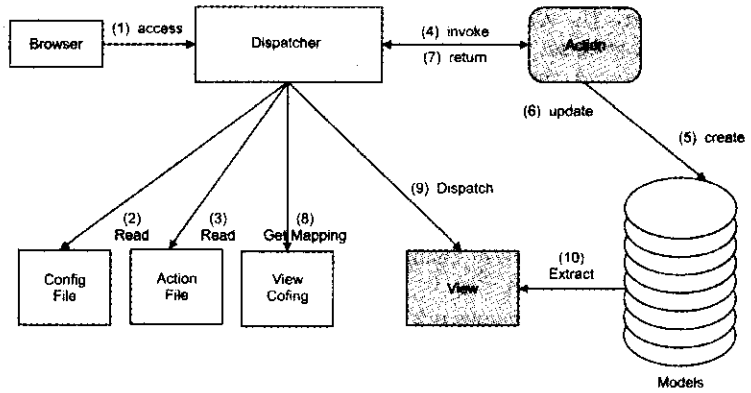


Figure B.2: Flow in WebWork's Architecture

Figure B.2 shows the following the sequence of steps:

1. Browser access dispatcher through a request
2. Dispatcher uses configuration file to get the configuration details
3. Dispatcher reads action configurations to know the appropriate action to be invoked
4. Dispatcher creates an instance of Action class and executes its method
5. Action class creates instance of Model. If the Model is already created, step 6 is taken
6. After creating Model instances, the Action class updates Model for the changes, if any
7. Action returns the result code
8. The dispatcher reads result/view configuration to know appropriate view according to the result code returned by the Action class
9. Dispatcher dispatches to the selected view
10. View further extracts data from the Model and displays it to the user in the appropriate manner

The work flow described through these steps can be compared with what you will study for the Struts 2 based application as the basic concept of dispatcher, action and interceptors have been inherited to Struts 2 from WebWork2.

Features of WebWork2

There are several features of WebWork2, and some of the important features are as follows:

- It is easy to learn. You can start programming using MVC 2 design pattern very easily.
- The interfaces used in WebWork2 are simple.
- WebWork2 provides configuring the Servlet container through `web.xml`, and configuring WebWork through `xwork.xml`.
- Interceptors are one of the most important features of WebWork2. Interceptors allow you to do some processing before and/or after the action and results are executed.
- It is easier to upload a file. WebWork provides an Interceptor, `FileUploadInterceptor`, which provides the retrieval and cleanup of uploaded files. Using this Interceptor, your Action doesn't need to worry about request objects, request wrappers, or even cleaning up the `File` objects.
- WebWork2 creates dynamic web pages without using any Java code in your JSPs, by using simple expression language called the Object Graph Navigation Language (OGNL).
- Velocity is used in place of JSP Scriptlets. However, it is not best practice to write Java code in JSP. WebWork provides Velocity that has a simpler format, which can be used in HTML editors very easily.
- WebWork provides FreeMarker, a template engine which is used to generate text output based on templates. It generates output by using HTML template file and Java objects. It also provides features, like using any JSP tag library.

- WebWork provides an Expression Language (EL) that is a scripting language which allows simple and concise access to JavaBeans, collections, and method calls. It is also used to eliminate the repetitive Java code.

Struts 2

Struts 2 is a free Open Source Framework for creating Java Web applications. It is based on WebWork2 technology. The features of Struts 2 are user interface tag, type conversion, Interceptor, result and validation. Struts 2 is a highly extensible and flexible framework for creating Java-based Web application. It has been developed on the concepts of MVC 2 architecture. The Struts 2 project provides an open-source framework for creating Web applications that easily separate the Presentation layer and Transaction and Data model on different layers. Struts 2 Framework includes a library of mark-up tags, which is used to make dynamic data. To ensure that the output is correct with the set of input data, the tags interact with the framework's validation and internationalization. The tag library can be used with JSP, Velocity, FreeMarker, and other tag libraries, like JSTL and AJAX technology.

Before exploring the features of Struts 2 and understanding how it is different from its previous versions, let's take a look at the history behind it.

Struts 1 was introduced as an idea of using JSPs and Servlets in Web applications to separate the Business logic from the Presentation logic. As time passed, a need for new enhancements and changes was felt in the original design. This forced developers to evolve a next generation framework that could keep up with these changes. There were two candidates at that time that could meet the requirements of the next generation framework—Shale and Struts Ti. Shale is a component-based framework, which has now become a top-level Apache project, whereas Struts Ti continues to follow the Model-View-Controller architecture. Then in March 2002, the WebWork Framework was released. WebWork includes new ideas, concepts and functionality with the original Struts code. In December 2005, WebWorks and the Struts Ti joined hands to develop Struts 2. The request flow in Struts 2 Web application Framework is shown in Figure B.3:

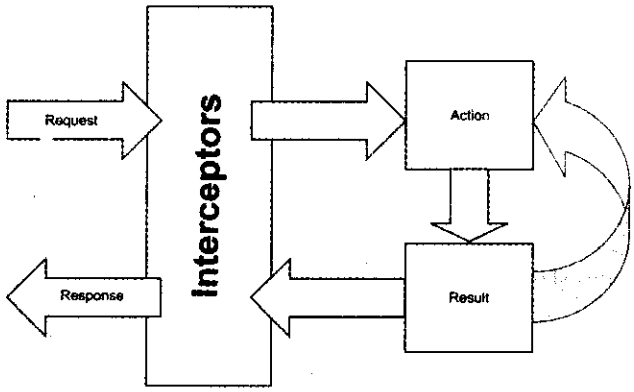


Figure B.3: Request Processing in Struts 2

The processing of a request can be divided into the following steps:

- **Request received**—As a request is received by the framework, it matches it to a configuration so that the required interceptors, Action class, and result class can be invoked.
- **Pre-processing by Interceptors**—Once the configuration is found, the request passes through a series of interceptors. These interceptors provide a pre-processing for the request.
- **Invoke Action class method**—After the pre-processing by the interceptors, a new instance of the Action class is created and the method providing the logic for handling the request is invoked. It is to be noted that in Struts 2, the method to be invoked by the Action class can be specified in the configuration file.
- **Invoke Result class**—Once the action is performed and the result is obtained, a Result class matching the return from processing the actions method is determined. A new instance of this return class is created and invoked.

- **Processing by Interceptors**—The response passes through the interceptors in reverse order to perform any clean-up or additional processing.
- **Responding user**—The control is returned back to the Servlet engine and the result is rendered to the user.

This was how different components interact internally, but the high level design pattern followed by Struts 2 Framework is MVC 2, similar to Struts 1, and the different components represent the different concerns of MVC 2 design pattern. The Model, View and Controller are represented by different Struts 2 components Action, Result and FilterDispatcher, respectively. The MVC pattern implementation through Struts 2 components is shown in Figure B.4:

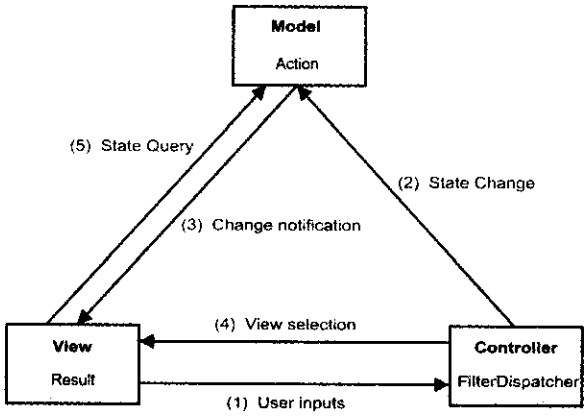


Figure B.4: Struts 2 Components and MVC Pattern Implementation

The work flow is broken into the following steps:

1. The user sends request through a user interface provided by the View, which further passes this request to the Controller, i.e. FilterDispatcher which is a Servlet filter class.
2. The Controller Servlet filter receives the input request coming from the user via the interface provided by the View and instantiates an object of suitable Action class, and executes different methods over this object.
3. If the state of model is changed, then all the associated Views are notified for the changes. Otherwise, this step is skipped.
4. Then the Controller selects the new View to be displayed according to the result code returned by action and executes suitable Result to render the View for the client.
5. The View presents a user interface according to the Model. The data is also contained within the Model i.e. actions. View queries about the state of Model to show the current data which is pulled from the action itself.

Similar to Struts 1, Struts 2 implements front controller approach with MVC 2 pattern, which means there is a single controller component here. All requests are mapped to this front controller, i.e. org.apache.struts2.dispatcher.FilterDispatcher class. The main work of this controller is to map user requests to proper actions. Unlike Struts 1, where actions are used to hold the Business logic and the model part represented by form beans, in Struts 2 the Business logics and model both are implemented as action components. In addition to JSP pages, in Struts 2 View can be implemented using other Presentation layer technologies, like Velocity template or some other presentation layer technology. We will study the working of Struts 2 Framework in detail later in this appendix with the involvement of other Struts 2 components, like Interceptors.

Let's now explore the features of Struts 2, which make it so popular.

Features of Struts 2

Struts 2 was created with the intention of streamlining the entire development cycle, from building, to deploying, to maintaining applications over time. These goals are achieved by Struts by providing the following features supporting building, deploying and maintaining applications:

Build Supporting Features

The Build supporting features of Struts 2 are described as follows:

- New projects can be started easily with the online tutorials and applications provided by various Struts forums.
- Stylesheet-driven form tags provide decreased coding effort and reduce the requirement for input validation.
- Smart checkboxes are provided with the capability to determine if the toggling took place.
- Cancel button can be made to do a different action. Generally, Cancel button is used to stop the current action.
- It supports AJAX tags that provide an appearance similar to standard Struts tag and help to design interactive Web applications.
- It provides integration with the Spring application framework, which is also an open source application framework for the Java platform
- Results obtained after the processing can be processed further for JasperReports, JFreeChart, Action chaining, file downloading, etc.
- It uses JavaBeans for form inputs and putting binary and String properties directly on an Action class.
- There is no need of interfaces because the interfacing among the components is handled by the Controller and any class can be used as an Action class.

Deployment Supporting Features

The deployment supporting features of Struts 2 are as follows:

- It allows framework extension, automatic configuration, and plugins to enhance the capabilities and add new features in future.
- Error is reported precisely by indicating the location and line of an error, which helps in debugging.

Maintenance Supporting Features

- Struts Actions can be tested directly, without using the conventional mock HTTP objects to debug the application.
- Most of the configuration elements have an intelligent default value that can be set only once.
- Controller can be easily customized to handle the request per action. A new class is invoked for handling a new request.
- It provides built-in debugging tools to report problems; hence not much time is wasted in the manual testing.
- It supports JSP, FreeMarker, and Velocity tags that encapsulates the Business logic and requires no need to learn taglib APIs.

Relation between WebWork2 and Struts 2

Struts 2 Framework is based on the WebWork2 Framework. One may think that Struts 2 is an extension of WebWork2, but it is not so. Struts 2 Framework includes features of both WebWork2 and Struts 1 Framework.

Since the arrival of the Apache Struts in year 2000, it has enjoyed a great run among the developer's community. Struts 1 provided a solid framework to organize a mess of JSP and Servlets to develop Web applications, which mostly used server generated HTML with JavaScript for client-side validation. These Web applications were easier to develop and maintain. As time passed, even though the customer's demand for Web applications increased, Struts 1 remained the same. Struts 1 was unable to cope with the growing demands of the customers.

In year 2005, the developer's sat down to discuss the future of Struts project. They came up with Struts Ti proposal. The Struts Ti was also a Struts Framework with some advanced features. But the problem was that Struts 1 was unable to cope with the advancement in technology. So the developers decided to join hands with WebWork developer's team in order to provide a new and efficient framework to the users. The WebWork Framework was part of OpenSymphony and at that time the latest version of this framework was WebWork2. This new framework was meant to provide support for almost all the new technologies which were present at

the time for developing a Web application. The Struts and WebWork was then merged together to provide a new framework called Struts 2.

The Struts 2 Framework is dependent on the WebWork2 Framework. The code of WebWork2 was included in Struts 2. Though Struts 2 has incorporated Struts 1 features, drastic changes have been made in Struts 2 in order to make this framework simpler and easy to use for the developers. It has included features, like Interceptors, Results, etc. from the WebWork2. Several files and packages were included from WebWork2 in Struts 2 Framework. Some elements were included as it is, while others were included with some changes, e.g. names of some packages, files, etc. are changed, but their structure remains same. Some of the features from WebWork were removed. New features were added in Struts 2 Framework, but the core part of this new Struts 2 Framework was based on the WebWork2 Framework.

The architecture of Struts 2 is fully taken from the architecture of WebWork2 Framework. The processing of the request in Struts 2 is almost the same as that in WebWork2. There are a number of extra plug-ins that are included in the Struts 2 as compared to WebWork2 Framework.

Thus, it can be said that Struts 2 Framework is very much dependent on the WebWork Framework.

Changes from WebWork2

Both Struts 2 and WebWork2 are frameworks, which are used to create Web applications based on the MVC architecture. There are a number of changes in Struts 2 Framework as compared to WebWork2 Framework. Some of the features are added or changed, some even removed from the Struts 2 Framework. Also some of the members are renamed in the newly created Struts 2 Framework. But, the basic implementation at most of places is same. To make the picture clear, let's take a look at the changes between WebWork2 and Struts 2 Framework.

There are several features which are changed from WebWork2 while developing Struts 2. Some of them are as follows:

- `ConfigurationManager` is no longer a static factory; an instance is created through Dispatcher. `DispatcherListener` is used for custom configuration.
- The tooltip library used by `xhtml` theme was replaced by Dojo's tooltip component.
- Tiles integration is available as a plug-in.
- Wildcards can be specified in action mappings.
- To allow field errors/action errors to be stored and retrieved through session, `MessageStoreInterceptor` is introduced. Messages are also stored and retrieved through session.

There are several features which are removed from WebWork2 to develop Struts 2. Some of them are as follows:

- **AroundInterceptor**—The `AroundInterceptor` is removed in WebWork2. If you are extending the `AroundInterceptor` in your application, import the class into your source code from WebWork2 and modify it to serve as your base class, or rewrite your Interceptor.
- **oldSyntax**—The oldSyntax is removed from WebWork2.
- **Rich text editor tag**—Rich text editor tag is removed and is replaced by the Dojo's rich text editor.
- **Default method**—`doDefault` is not supported for calling the `default` method.
- **IoC Framework**—The internal IoC framework has been deprecated in WebWork 2.2 and removed in Struts 2. Struts 2 Framework has a Spring plugin for implementing the IoC. This is implemented by using the factory, ObjectFactory.

Table B.1 shows the members that are renamed in Struts 2:

Table B.1: Renamed members in Struts 2

WebWork2	Struts 2
`com.opensymphony.xwork.*`	`com.opensymphony.xwork2.*`
`com.opensymphony.webwork.*`	`org.apache.struts2.*`
`xwork.xml`	`struts.xml`

Table B.1: Renamed members in Struts 2	
[redacted]	[redacted]
webwork.properties	struts.properties
DispatcherUtil	Dispatcher
com.opensymphony.webwork.config.Configuration	org.apache.struts2.config.Settings

In the following example, the tag prefix conventions are changed in Struts 2:

JSP	s:	<s:form ...>
Freemarker	s.	<@s.form ...>
Velocity	s	#sform (...)

Architecture of Struts 2

The architecture of Struts 2 Framework is fully based on MVC architecture. The Struts 2 architecture has a flexible control layer based on standard technologies, like JavaBeans, ResourceBundles, XML, Locales, Results, Interceptors, etc. The architecture of the Struts 2 is shown in Figure B.5:

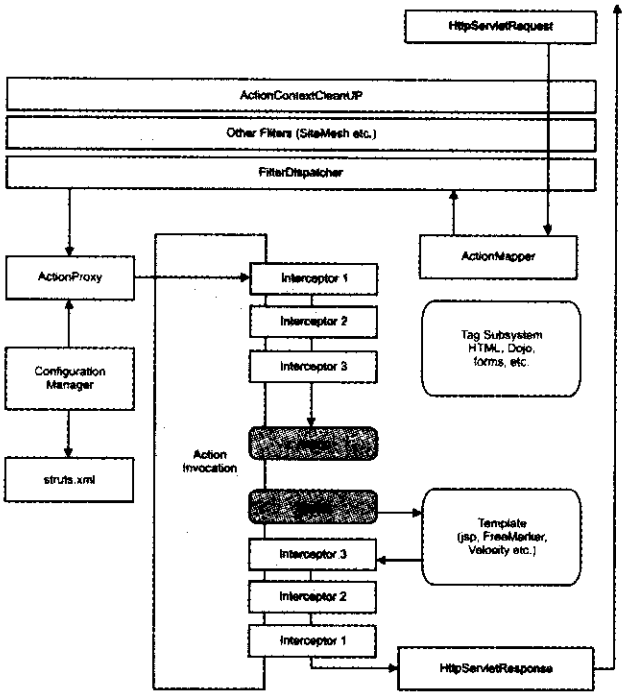


Figure B.5: Architecture of Struts 2 Framework

The various components of Struts 2 can be categorized into the following groups, according to the Struts 2 architecture:

- **Servlet Filter**—The `ActionContextCleanUp`, `FilterDispatcher`, and `SiteMesh` (and other filter) components are treated as Servlet filter component.
- **Struts 2 Core**—`ActionProxy`, `ActionMapper`, `Tagsystem`, `ActionInvocation`, `ConfigurationManager` and Result are treated as Struts 2 Core component.

- **Interceptors**—Interceptors are presented both above the Action component and below the Result components.
- **User created components**—The `struts.xml` file, Action classes, and Templates are treated as user created components. Here the user is allowed to change these configurations.

In the Figure B.5, first the client request passes through the Servlet container, such as Tomcat. Then the client request passes through a chaining system, which involves the three components of Servlet filters.

In the chaining system, first the `HttpServletRequest` goes through the `ActionContextCleanUp` filter. The `ActionContextCleanUp` works with the `FilterDispatcher` and allows integration with `SiteMesh`. Next the `FilterDispatcher` invokes the `ActionMapper` to determine which request should invoke an Action. It also handles execution actions, clean up `ActionContext`, and serves static content, like JavaScript files, CSS files, HTML files, etc. whenever they are required by various parts of Struts 2.

The `ActionMapper` maps HTTP requests and action invocation requests and vice-versa. The `ActionMapper` may return an action or return null value, if no action invocation request matches.

When the `ActionMapper` wants that an Action should be invoked, the `ActionProxy` is called by interacting with the `FilterDispatcher`. The `ActionProxy` obtains the Action class and calls the appropriate method. This method consults the framework Configuration Manager, which is initialized from the `struts.xml` file. After reading the `struts.xml` file, the `ActionProxy` creates an `ActionInvocation` and determines how the Action should be handled. `ActionProxy` encapsulates how an Action should be obtained, and `ActionInvocation` encapsulates how the Action is executed when a request is invoked. Then the `ActionProxy` invokes Interceptors with the help of `ActionInvocation`. The main task of Interceptors is to invoke the Action by using `ActionInvocation`.

Once the Action is determined and executed, the result is generated by using Result component. The Result component lookup data from the Template with the help of Action result code mapped in `struts.xml`. The Template may contain JSP, FreeMarker, or Velocity code which is used to generate the response. The Template uses the Struts Tags, which are provided by the Struts 2 Framework. After collecting data/result from Template, the Action Invocation produces the result with the help of Interceptors.

Before, we conclude our discussion over Struts 2 architecture and how the different events take place one after another when a request is processed in Struts 2 based Web application, let's go through a few more important Struts 2 concepts in brief—Interceptors, ValueStack, and OGNL.

Interceptors

The architecture of Struts 2, shown in Figure B.5, shows a number of interceptors being executed before action and after result is executed. Interceptors are key concepts being utilized in Struts 2 Framework. Interceptors allow you to develop code that can be run before and/or after the execution of an action. Interceptors may also provide for Security checking, Trace Logging, Bottelneck Checking by using the interceptor package. We have a stack of interceptors being executed before action and they can be considered as a kind of request processor in Struts 1. Different sets of interceptors can be used in the stack according to the common functionalities being provided by them. Each interceptor simply defines some common workflow and crosscutting tasks which make them separate from other concerns and makes them reusable.

NOTE

Interceptors have been covered for all its functioning, and configuration details in appendix 4.

ValueStack and OGNL

Some important concepts implemented in Struts 2 are ValueStack and the expression language to interact with this ValueStack, i.e. Object Graph Navigation Language (OGNL). A ValueStack can be defined as a storage structure where the data associated with the current request is stored. As the name suggests, ValueStack is a collection of data stored in a Stack. OGNL expression language is used to retrieve and manipulate data from ValueStack. The data can be set into ValueStack during the pre-processing of the request, manipulated when the Business logic in action is being executed, and read from ValueStack when the Result renders the response for the client.

Comparing Struts 1 with Struts 2

Struts 2 Framework is a new framework, but it can easily be grasped for its new concepts and implementations by comparing it with the most popular Web application Framework, i.e. Struts 1. This comparison is important as both, Struts 1 and Struts 2, share a common implementation in terms of MVC 2 pattern and action based approach in the development of components. There are also some advanced features which are added in the new Struts 2 version, such as Interceptors, XWork Validation Framework support, OGNL support, integration with other frameworks using various plugins, POJO Action, POJO Forms, etc. There are lots of changes which we can observe in Struts 2 Framework. But the basic implementation at most of the places is same as in Struts 1. To make the picture clear, let's take a look at the similarities and differences between Struts 1 and Struts 2 Framework.

Similarity between Struts 1 and Struts 2

Both Web applications framework, i.e. Struts 1 and Struts 2, provide the following key components:

- Both versions follow the MVC 2 architecture.
- Both versions use a request handler that maps Java classes to Web application URIs.
- A response handler that maps logical names to server pages, or to other Web resources in both Struts 1 and 2.
- A tag library helps us to create rich, responsive, form-based applications, and generate dynamic content of web pages in both versions of Struts.

In Struts 2, all the preceding concepts are redesigned and enhanced, but the same architectural hallmarks remained.

We have discussed the similarities between Struts 1 and Struts 2. But the Struts 2 Framework introduces a number of new concepts and type of implementations. Let's now discuss the key changes in Struts 2, as compared to the Struts 1.

Difference between Struts 1 and Struts 2

In addition to various similarities between Struts 1 and Struts 2, there exists various differences as well that gives Struts 2 an edge over the its counterpart. The description of these differences, shown in Table 1.2, is helpful while migrating from Struts 1 to Struts 2. These points help in getting aware of the differences between Struts 1 and Struts 2 components. Further, the comparison makes the benefits of Struts 2 over Struts 1 clear to the reader. Table B.2 describes the differences between Struts 1 and Struts 2:

Table B.2: Difference between Struts 1 and Struts 2

Feature	Struts 1	Struts 2
Action classes	For developing Struts controller component of MVC model, the Action classes are needed, which extends the Abstract base class. This is a common problem in Struts 1 to program abstract classes rather than interfaces	In Struts 2, an Action may implement `com.opensymphony.xwork2.Action` interface along with other interfaces to enable optional and custom services. Struts 2 provide a base `com.opensymphony.xwork2.ActionSupport` class to implement the commonly used interfaces. Thus, Action interface is not required. Any POJO object with a execute signature can be used as a Struts 2 Action object
Binding values into views	To access different objects, Struts 1 uses the standard JSP mechanism for binding objects into page context	The technology ValueStack is used in Struts 2, so that taglibs can access values without coupling your JSP view to the object type it is rendering. Reuse of views to a range of types that may have same property name but different property types are allowed by using `ValueStack` strategy
Servlet Dependency	When an Action is invoked the `HttpServletRequest` and `HttpServletResponse` are passed to the	Struts 2 Actions are not coupled to container. Servlet contexts are represented as simple Maps that allow Actions to be tested in isolation. If

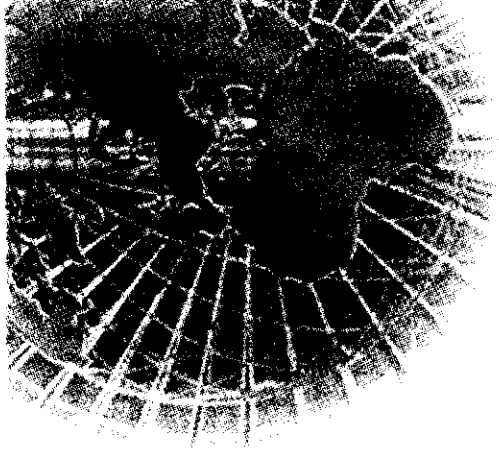
Table B.2: Difference between Struts 1 and Struts 2

Feature	Struts 1	Struts 2
	`execute()` method. That's why Struts 1 Actions have Servlet dependencies	required, the original request and response can still be accessed in Struts 2
Thread Modelling	There is only one instance of a class for handling the entire requests specific to a particular action. Thus Struts 1 Actions are singleton and must be thread-safe. The singleton strategy places restrictions on what can be done by using Struts 1 Actions and thus requires extra care to develop	There are no thread-safety issues because Action objects are instantiated for each request in Struts 2
Testability	The execute method exposes the Servlet API; this is the major hurdle for testing Struts 1 Action	The testing is done by instantiating the Action, setting properties, and invoking methods. Also the dependency injection support also makes the testing simpler
Control of Action Execution	Struts 1 supports separate Request Processors (life-cycles) for each module, but the same life-cycle is shared to all the Actions in the module	Struts 2 supports creation of different life-cycles on a per Action basis via Interceptor Stacks. Whenever needed custom stacks can be created and used with different Actions
Harvesting Input	To capture input, Struts 1 uses an object of `ActionForm`. All ActionForms must extend a base class. Since other JavaBeans cannot be used as ActionForms, developers often create redundant classes to capture input. `DynaBeans` can be used for creating conventional `ActionForm` classes, but here too developers may be redescribing the existing JavaBeans	Struts 2 eliminates the need of second input object by using action properties as input properties. This Action property is accessed from the web page via the taglibs. Struts 2 also support POJO form objects and POJO Action. Input properties may be rich object types with their own properties. Rich object types, which include business or domain objects, can be used as input/output objects
Expression Language (EL)	Struts 1 supports JSTL, so it uses the JSTL EL. The EL has relatively weak collection and indexed property support	The Struts 2 Framework supports Expression Language (EL) that is known as Object Graph Notation Language (OGNL). The EL is more powerful and very flexible. Struts 2 also support JSTL
Validation	Validation in Struts 1 is supported by a validate method on the ActionForm, or by using a Validation Framework thorough a plugin	Validation in Struts 2 is performed by the validate method and the XWork Validation framework. The XWork Validation framework supports the chaining validations into the sub-properties by using the validations defined for the properties class type and the validation context
Type Conversion	The type-conversion is performed by Commons-Beanutils. The properties of Struts 1 ActionForms are usually of String type	The type-conversion is performed by Object Graph Notation Language (ONGL)

This comparison between Struts 1 and Struts 2 is helpful for the developers who have interacted with Struts 1 Framework for Web application development. This explains where Struts 2 is different from Struts 1 and what makes it preferred over Struts 1.

This appendix embarks with Struts 2 Framework and Web application Frameworks, followed by the introduction of Struts 2 Framework—a powerful and flexible Web application Framework—and moves towards a discussion on WebWork2 and Struts 2 that brings various architectural views, which further creates a new terminology, such as Interceptors, Results, ValueStack, OGNL, and more. The appendix concludes with the comparison of Struts 1 and Struts 2.

The next appendix describes Enterprise Java Beans (EJB).



C

Understanding EJB 3.0

An enterprise level application developed on Java Platform, Enterprise Edition (Java EE) should be distributive, transactional, secure, and portable. To ensure that the application supports all these features, developers have to provide extra code and logic for the implementation of various low level interactions, which can distract the developers from the actual business logic implementation. So, to overcome this problem, we use a user-defined component, which is used by the container to ensure security, manage transaction, implement automatic object persistence, and enhance portability.

In this appendix, we are introducing a new technology called server-side component architecture for Java EE, more commonly known as Enterprise JavaBeans (EJB). The use of EJB component helps in easy and fast development of distributed, transactional, secure, and portable Java applications. EJB components and EJB containers used in application servers reduce the efforts of the developers, as they do not need to understand low level transaction and state management details, connection pooling, multithreading, and other similar complex processes related to application development, which are now provided by the container.

The Java EE 5 specification supports Enterprise JavaBeans 3.0 (EJB 3.0), the latest version of EJB. In this appendix, while discussing the EJB technology, we mainly focus on the new concepts introduced by EJB 3.0, and how it has made developing a Web application easier.

EJB 3.0 Fundamentals

Enterprise JavaBeans is a standard architecture used for enterprise level distributed applications that are object-oriented, transaction-oriented, as well as distributed. Implementing the Enterprise JavaBeans architecture means creating enterprise bean components that are managed by EJB container in the application server. The enterprise bean implements the business logic, which can operate on enterprise data and is normally defined through some enterprise bean methods. These enterprise bean methods can be accessed and invoked by remote clients. The behavior of these enterprise beans can be customized at the time of their deployment by changing the entries in the deployment descriptor. The enterprise bean instance is created and managed by the EJB container where it is deployed. In addition to business logic, we can also provide service information by using annotations in the enterprise bean itself or declaratively in the deployment descriptor. The enterprise beans can use the container for all the services defined in the specification. We need to create a client view of the enterprise bean. The client view refers to creating another enterprise component or other Java program, such as applet and Servlet, to access enterprise bean and its business methods. The client view of the enterprise bean is independent of the type of container and server in which the enterprise bean is deployed.

EJB architecture is flexible and helps in implementing the enterprise bean in various scenarios as mentioned here:

- The enterprise bean can be used to provide all stateless services.
- The enterprise bean can be used as a Web service endpoint that provides stateless service.

- The enterprise bean can be used to provide stateless service asynchronously with the arrival of some message.
- The enterprise bean can be used to provide stateful services, where the conversational state of a client is maintained.
- The enterprise bean can be used as an entity that represents a persistent object that is managed automatically for its persistence and relation with other entities.

There are different types of enterprise beans, which can be used in a Web application, depending on the scenarios discussed earlier. Let's start discussing why EJB 3.0 is required and then discuss different EJB concepts and their architecture.

Why EJB 3.0?

The previous version of EJB, EJB 2.1, was powerful enough to support all the needs of a distributed enterprise application, but it was very complex to develop because a developer has to create a number of interfaces and classes for implementing single enterprise bean. In addition to this, all interfaces and classes developed were required to implement interfaces and extend classes from `javax.ejb` package. The client, therefore, had to use JNDI look up to access enterprise bean and invoke its business methods.

The EJB 3.0 introduced with Java EE 5 specification has made Enterprise JavaBeans technology simpler and easier to be implemented in the application. EJB 3.0 needs fewer interfaces and classes as it supports metadata annotation to support dependency injection. The use of default behavior and configuration through annotations has further removed the need of deployment descriptors, and the entity persistence and Object/Relational mapping has also been simplified. In EJB 3.0, since the container has to work more for doing all the processing, it gives developer enough time to concentrate on the implementation of business logic, thereby increasing the productivity of the developer. The EJB 3.0 enhances business logic implementation and speeds up the enterprise bean development.

Some facts about EJB 3.0 are as follows:

- EJB 3.0 leverages Java language metadata, which helps simplify all bean types, and resource accesses. It reduces the need of deployment descriptors but preserves the ability to use XML file as an alternative mechanism and overriding annotations.
- The enterprise beans now resemble Plain Old Java Objects (POJO) and do not need to implement EJB component interfaces.
- All Business interfaces are also Plain Old Java Interfaces (POJI).
- The Home interfaces are not required any more in EJB.
- In EJB 3.0, all callback methods are defined by using annotations, because javax.ejb.EnterpriseBean interfaces are not used in it.
- The persistence model is simplified.
- EJB 3.0 specification always ensures that the EJB 2.1 API is available and supports the reuse of existing components in the new application. EJB 3.0 specification helps in migrating from previous EJB 2.1 application to new EJB 3.0 applications.

EJB 3.0—Architecture and Concepts

The EJB 3.0 architecture comprises the EJB server, EJB container, and EJB client. In this section, we discuss how the container providers can change EJB processing model.

The first model is to process an EJB file to generate deployment artifacts (required interfaces and deployment descriptors) closer to the EJB 2.1 deployment model and then deploy the EJB component in the same way as EJB 2.1 deploys it. Of course, the deployment descriptors and files generated can be non-standard, but they might resemble the deployment descriptors in EJB 2.1. This approach reduces rework for container providers and also reduces the burden of supporting both EJB 2.1 and EJB 3.0-style EJB components. Figure C.1 shows the EJB architecture:

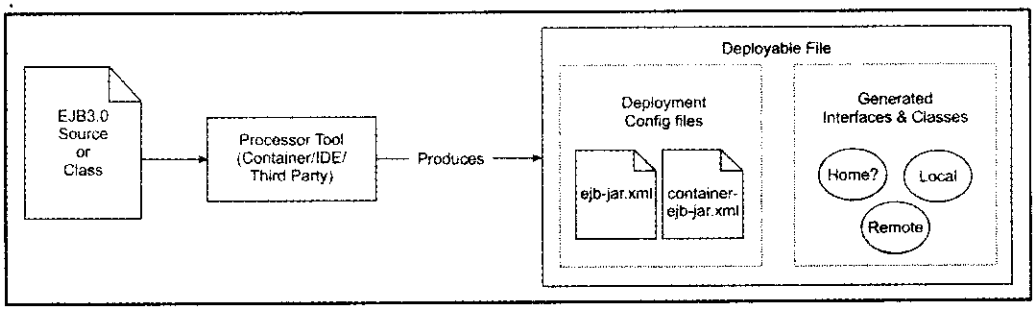


Figure C.1: EJB Architecture

Another EJB 3.0 processing model is a JSP-like (JavaServer Pages) drag-and-drop deployment model. Now you can drop an EJB file into a pre-designated, implementation-defined directory. The container picks it up, processes it, deploys it, and makes it available for use.

Now let's discuss the EJB architecture in detail by understanding EJB server, EJB container, and EJB clients.

EJB Server

An EJB server provides a runtime environment to execute server applications that use enterprise beans. An EJB server is used to create an infrastructure to deploy server applications, also called components. The EJB server provides a JNDI-accessible naming service, manages and co-ordinates the allocation of resources to client applications, provides access to system resources, and provides a transaction service.

In the EJB architecture, servers are basically the resource managers. Every server manages the allocation of resources to the containers it controls. It is the resource manager that determines which and how many of the containers may run within the server. Following points show what a server does in the EJB architecture:

- At the client-level, the server manages all incoming client requests. This also includes providing connections to clients, dispatching client requests to containers, and routing responses back to clients.
- At the container-level, the server manages the processing of the resources that are available to the container. This type of management includes allocating available working processes and threads to the running containers.
- At the service-level, the server verifies that the container has access to shared services, such as a centralized security manager, a pool of JDBC drivers, a transaction service implementation, a global cache manager, and an asynchronous messaging service.

You can run an EJB application on more than one server. So, in addition to allocating resources to the containers managed by the servers, the servers must cooperate as a group to allocate resources efficiently across all the servers in a cluster. This cooperation among the servers involves publishing the load information to a load balancing process, and then assigning the client requests to the least-loaded server to provide the requested services. It also includes system management infrastructure, such as demons, that automatically starts server processes, alerts to a management console when the server load exceeds a certain level, and management interfaces for controlling the resource allocation policies of the server.

EJB Container

The EJB container provides an environment in which one or more enterprise beans can run. This environment is basically a combination of available interfaces and classes that the container uses to support enterprise beans throughout their life-cycle. The EJB container performs the following tasks:

- The EJB container transparently provides services by intercepting all method calls to the EJB components.
- The EJB container provides following control and management services:
  - **Life-cycle management**—Creates and removes enterprise bean instances, handles instance pooling with their activation and passivation.
  - **Naming services**—Provides JNDI registration of EJB when the enterprise bean is loaded.

- **Security checks**—Performs authentication and access control and implements declarative and programmatic security.
- **Persistence management**—Helps in storing the enterprise beans.
- **Transaction coordination**—It is the declarative transaction management
- **Resource pooling**—It is an indirect access to the bean instance

The EJB container simplifies the complex aspects of a distributed application, such as security, transaction coordination, and data persistence. The EJB infrastructure is implemented by EJB container and service providers. This infrastructure deals with the distribution aspects, transaction management, and security aspects of an application. The Java APIs for the infrastructure is defined by EJB specification. Now the developers focus on their area of expertise -- the Business logic -- and leave the infrastructure to the platform specialists. Figure C.2 shows various services provided by EJB container:

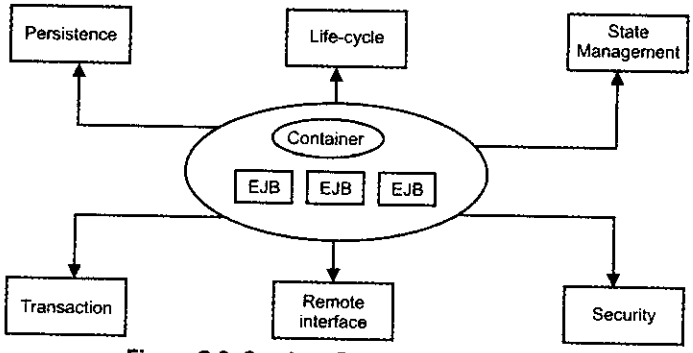


Figure C.2: Services Provided by Container

The responsibilities of an EJB container for each enterprise bean are as follows:

- It registers the object.
- It provides a Remote interface for the object.
- It creates and destroys object instances.
- It checks security for the object.
- It manages the active state for the object (activation/passivation).
- It co-ordinates distributed transactions.
- It persists CMP entity beans, and saves stateful Session beans attributes when passivated.

Whenever a client wants to call business methods of an enterprise bean, it actually connects to the container first. The container then verifies that the call conforms to the semantics specified in the deployment descriptor before dispatching the call to the EJB itself. This mechanism allows the container to control all aspects of the execution of an EJB, including the following:

- **Life-cycle**—The life-cycle methods of the enterprise bean are invoked as per the specification. While managing the life-cycle of an enterprise bean, the container itself enforces security, transaction, and persistence requirements.
- **Security**—When a client attempts to call a method of an EJB, the container looks for the user's permission in an Access Control List (ACL) to verify whether the client has the right to invoke that method. The deployment descriptor object contains declarations about the access control for an EJB's methods.
- **Transactions**—These are the declarations about the transaction restrictions on an EJB's methods included in the deployment descriptor object. When a client attempts to call the method of an EJB, the container is responsible for verifying whether the call obeys the transaction restrictions on that method. The restrictions for execution of a transaction are as follows:
  - It must not execute within a transaction context.
  - It may execute within a transaction context.

- It must execute within a transaction context, and it must execute within a new transaction context.

- **Persistence**—We can use two different types of persistence–one that is managed by bean itself and the other that is managed by the container. In case of the persistence managed by beans itself, the bean needs to implement its own data access code, while in the persistence managed by the container, the enterprise bean delegates data access to the container, thus making the container responsible to implement appropriate data access functioning.

EJB Client

The client code is the code written to execute business logic embedded in enterprise bean and its methods. EJB client can be a simple Java program or a Web client. The end-user runs these clients to get results from the enterprise bean method invocation. An EJB client can use resource injection or JNDI service to access an enterprise bean. The client code can be a simple Java class, Applet, Servlet, or JSP. An enterprise bean is looked up using resource injection or lookup() method, irrespective of the different EJB clients.

Features of EJB 3.0

EJB 3.0 specification addresses all the complexities that it has in its previous versions and solves them by implementing new strategies. The new features added in the EJB 3.0 have changed the way the enterprise beans are developed, configured, and deployed. The extensive use of metadata annotations has reduced the use of different interfaces and classes and deployment descriptors. The interfaces required in an application and deployment descriptor generation are now taken care of by the container. This has relaxed developers from developing unnecessary local or remote home and component interfaces. The callback methods can easily be marked in the enterprise bean now. The enterprise bean has been simplified to a POJO model and all resources to be used by the components are injected using dependency injection. The features of EJB 3.0 have been summarized here:

Annotations

A new program annotation facility has been introduced in the Java Platform, Standard Edition, 5 (Java SE 5). These metadata annotations (or simply the annotations) are inspected at compile time or runtime by the different tools to generate additional constructs, such as deployment descriptors, and to customize the component's runtime behavior. We can annotate class fields, methods, and classes. EJB 3.0 proposed a set of annotations that has made development of enterprise beans easy. We can mark interfaces, classes, fields, and methods created for an EJB implementation with different annotations, such as @Remote, @Stateful, and @Stateless. The annotations indicate that the associated element maintains some application-specific or domain-specific semantics. Annotation should only be provided when defaults cannot be used. For example, for an enterprise bean CustomerBean, we do not need to extend `javax.ejb.EnterpriseBean` type of class; instead, we can use @Stateless annotation to declare it stateless session bean, as shown in the following code snippet:

```
package com.kogent.ejb;
import javax.ejb.Stateless;
@Stateless
public class CustomerBean
{
  public String sayHello()
  {
      return "Hello from CustomerBean.";
  }
}
```

Elimination of Home Interface

In EJB 3.0, all enterprise beans are created homeless. It means that we do not need to create home interface for the enterprise bean. In EJB 2.1, we need to create a home interface by extending `javax.ejb.EJBHome` interface or `javax.ejb.EJBLocalHome` interface and declare life-cycle methods, such as create() and remove(). In EJB 3.0, we do not need to create locale or remote home interfaces for the enterprise bean. The EJB 2.1 APIs are still supported in EJB 3.0 specification and the home interfaces can still be used, which supports reverse compatibility with previous versions.

Elimination of Component Interface

The earlier versions of EJB require the local or remote component interfaces to be created for the given enterprise bean. The local and remote component interfaces need to extend javax.ejb.EJBObject and javax.ejb.EJBLocalObject respectively. The component interfaces were used to declare the business methods that a client could invoke. In EJB 3.0, you do not need to create component interfaces as they have been replaced by business interfaces that declare all business methods to be provided to the client. The business interface can be local or remote, but it is a simple POJI (Plain Old Java Interface).

Callback Methods

In EJB 3.0, we do not need to define all the unnecessary life-cycle callback methods, such as `ejbPassivate()` and `ejbActivate()`, in the enterprise bean class. In the earlier versions, we have to define these callback methods but in EJB 3.0, they have become optional. However, we can define any of these methods in our bean class to which the container can invoke. In EJB 3.0, we can mark an arbitrary method as the callback method, which can listen EJB life-cycle events. We can use different metadata annotations, such as @PreDestroy, @PostConstruct, @PrePassivate, @PostActivate, and @Remove to mark a method as callback method. A sample enterprise bean method marked with @PreDestroy is shown in the following code snippet:

```
@Stateful
public class SomeSessionBean
{
  @PreDestroy
  public void someMethod()
  {
        /*Some Logic to be executed before the destroy() method*/
  }
}
```

Simple POJO Model

EJB 3.0 supports a simple Plain Old Java Object (POJO) model. A simple Java class which does not implement any interface or extends any class can be treated as an enterprise bean. A simple POJO object can be made a powerful component handling concurrency, transactions, and security issues. All these are provided by the container to the simple POJO objects.

The POJO model also helps in creating enterprise bean classes that do not depend on other APIs. The POJO objects help in implementing the unit testing by using frameworks, such as JUnit, without deploying them on a server.

The implementation of POJO model has totally simplified the development of enterprise beans. Now we just need to create simple business interfaces that are the Plain Old Java Interfaces (POJIs), and implement business methods in enterprise bean class which is a Plain Old Java Object (POJO). All the resources are made accessible in the component by resource injection using metadata annotations, and all callback methods can be marked with metadata annotations.

Java Persistence API

EJB 3.0 provides a new Java Persistence API, which further simplifies the programming model for entity persistence. The Object/Relational mapping and all persistence logic is now handled by the container and we just need to provide proper annotations, such as @Id, @Table, @OneToOne, and @OneToMany. These annotations are used to map entities and their relationships to application's database. The long term storage of this large amount of information has been simplified by Java Persistence API. The first release of Java EE 5 introduced entity bean as a solution to Java Persistence. The introduction of the entity bean has prevented the developers to directly deal with the persistence. But the first release of the technology lacked many features, such as annotations, and provoked many problems to the developers. Some of the major problems were: relationships of the entities which had to be managed by the applications, foreign key fields were required to be stored and managed on the bean class. The EJB2.1 specification solved many of the problems by introducing the idea of container-managed entity beans. In the container managed entity bean, the server was responsible for generating subclasses to manage the persistent data. The EJB2.1 specification also introduced the Enterprise Java

Bean Query Language (EJBQL)—a query language designed to create queries for CMP entity beans. This language is similar to SQL but searches for persistent attributes of the enterprise Java bean.

EJB 2.1 specification, despite of improved features than its earlier versions, was still overloaded with the major problems and complexities. The EJB 3.0 provides new Java Persistence API, which simplifies the programming model for entity persistence. Now Object Relational Mapping or Persistence approach uses POJO model instead of abstract persistence schema model. The Object Relational mapping maps entities and their relationships to application's database. EJB 3.0 entity depicts persistent information stored in database by using Container Managed Persistence (CMP) but EJB 2.1 entity bean only represents persistent information stored in the database. The optimistic locking technique that was supported only by TopLink persistence framework is also encapsulated in EJB 3.0 specification. It also enhances R/W access ratios. The optimistic locking technique allows you to use objects in a disconnected model implying that you can change data and their relationships offline and merge data operations into one transaction. We can also create deep and wide entity inheritance hierarchies. Other changes made in the Java Persistence API, which were not there in EJB2.1, are as follows:

- It requires less number of classes and interfaces.
- New EntityManager API, similar to Hibernate, is introduced to perform operations, such as creation, removal, and searching on entity beans.
- The lengthy deployment descriptors have been eliminated through annotations.
- It provides cleaner, easier, and standardized object-relational mapping.
- The need for lookup code has also been eliminated.
- It adds support for inheritance, polymorphism, and polymorphic queries.
- It adds support for named (static) and dynamic queries.
- We can now perform many database related operations, instead of performing only one operation generating primary key.
- It provides a Java Persistence query language—an enhanced EJB QL.
- It makes testing the entities without the EJB container easier. Earlier, the developers need to be aware of deployment platform to test EJBs.

These were changes in the new Java Persistence API, which is introduced with the new entity beans concept.

Dependency Injection

The client, to use the constructed bean in an application, needs to know how to locate and invoke that enterprise bean (constructed bean). The client for an EJB 2.1 session bean gets the reference of the session bean with JNDI. In order to use an enterprise bean, say Calculator bean, the EJB 2.1 client needs to add the following code to locate and invoke an enterprise bean method:

```
Context ic = new InitialContext();
Object obj = ic.lookup ("CalculatorJNDI");
CalculatorHome home =
(CalculatorHome)PortableRemoteObject.narrow(obj, CalculatorHome.class);
Calculator calc = home.create();
```

In the preceding code, the JNDI name of the Calculator bean is "CalculatorJNDI". The local/remote instance is obtained with create() method. But in EJB3.0, the JNDI lookup and create() method invocation is not required. In EJB3.0, a reference to a resource is obtained with dependency injection using annotation. EJB, by implementing dependency injection, conveys to the container that a particular resource is dependent on some resource. A dependency annotation comprises the type of resource, the resource properties, and a name to access the resource. The use of annotations in EJB 3.0 has simplified the task of both the developers and the EJB client. Some examples of dependency annotations are as follows:

```
@EJB (name="sessionBeanName", beanInterface=SessionBeanInterface.class)
@Resource (name="Database", type="javax.sql.DataSource.class")
```

The @EJB annotation used here injects stubs of session bean having name "sessionBeanName". The @Resource annotation is used to inject service object having JNDI name "Database". This name may be present in either global or local JNDI tree.

Dependency annotations may be associated to bean class, its member variables or methods. The information to be specified in dependency annotation depends upon the context and the amount of data to be fetched from that context.

Timer Service

In enterprise applications, sometimes we need to implement time-dependent services. For example, we may need to invoke a particular business method provided by enterprise bean after a given span of time or by invoking them repeatedly after a fix time interval. Before EJB 2.1, developers had to write code manually for building and deploying time-based workflows. However, with EJB 3.0, the task of creating such applications has been considerably simplified. Equipped with annotations and dependency injections, developers can use EJB 3.0 to build and deploy scheduled applications easily.

Interceptors

The interceptors intercept the invocation of business methods of enterprise bean to provide some additional functionality. The interceptors are used with session beans and Message-driven beans. Interceptor methods to be invoked before a business method can be defined in a separate class known as Interceptor class. For example, we always want to validate all passed values to the business methods before the actual logic is executed. We can invoke more than one interceptor on an EJB.

We use @Interceptors annotation to import a chain of interceptors associated with the bean. We can define interceptor methods using @AroundInvoke annotation. The following code snippet is used to add interceptor methods to a bean:

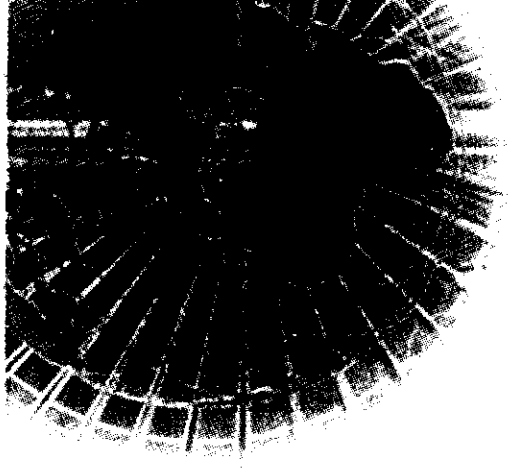
```
@Stateful
@Interceptors({MethodProfiler.class})
public class SomeServiceBean
{
  ...
  ...
}
```

Classifying EJBs

We have listed different scenarios in the enterprise application development where an enterprise bean can be implemented. We have different types of enterprise beans to be implemented in these scenarios. The enterprise beans can be classified into the following three types based on the type of functionality provided by them:

- Session Beans: Holds business processes, such as delivering an order, doing financial calculations for an application. The session beans are reusable components, which provide methods that can be accessed by the client. All the services provided by the session bean are in the form of different methods.
- Message-Driven Beans: Refers to the beans that are associated with some sort of messaging. Messaging is a method of communication between software components or applications. A messaging system is a peer to peer facility in which a messaging client can send messages to other clients, or can receive messages from other clients. Java Messaging Server (JMS) is a core service provided by Java EE application servers. JMS allows asynchronous invocation of different services via messages. JMS clients send messages to server maintained message queues. To monitor message queues, we need a special kind of EJB, called message-driven bean.
- Entity Beans: Represents persistent information stored in the database

This ends up the discussion on EJB.



Index

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z