Unit-5

INTRODUCTION TO AJAX and WEB SERVICES

INTRODUCTION TO AJAX

The underlying technologies behind classic Web applications (HTML) are simple and straight forward. The classic web pages has very little intelligence and lack dynamic and interactive behaviors. Changes in today's web pages are brought by AJAX (Asynchronous JavaScript and XML)

Ajax refers to a set of technologies and techniques that allow web pages be interactive like desktop applications.

AJAX is a new technique for creating better, faster, and more interactive web applications with the help of XML, HTML, CSS, and Java Script. Ajax uses XHTML for content, CSS for presentation, along with Document Object Model and JavaScript for dynamic content display. Conventional web applications transmit information to and from the sever using synchronous requests. It means the user fill out a form, hit submit, and get directed to a new page with new information from the server. With AJAX, when the user hit submit, JavaScript will make a request to the server, interpret the results, and update the current screen. The user would never know that anything was even transmitted to the server.

XML is commonly used as the format for receiving server data. AJAX is a web browser technology independent of web server software. A user can continue to use the application while the client program requests information from the server in the background. In AJAX, clicking is not required; mouse movement is a sufficient event trigger. It is a data-driven technology. AJAX cannot work independently. It is used in combination with other technologies to create interactive webpages. The technologies that support AJAX are: JavaScript, DOM, CSS and XMLHttpRequest.

AJAX is based on the following open standards:

- Browser-based presentation using HTML and Cascading Style Sheets (CSS).

- Data is stored in XML format and fetched from the server.
- Behind-the-scenes data fetching is done using XMLHttpRequest objects in the browser.
- JavaScript to make everything happen.

Asynchronous nature of AJAX

Asynchronous in AJAX means that the script will send a request to the server, and continue the execution without waiting for the reply. As soon as reply is received a browser event is fired, which in turn allows the script to execute associated actions. The client and the server are asynchronous.

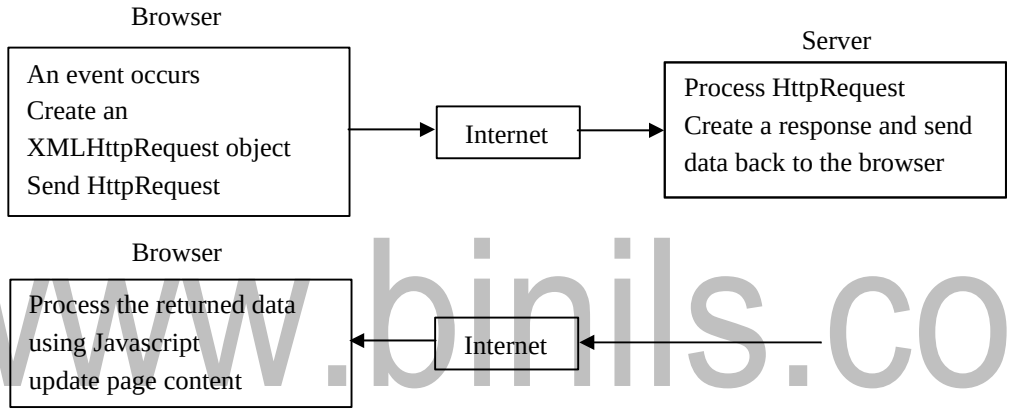
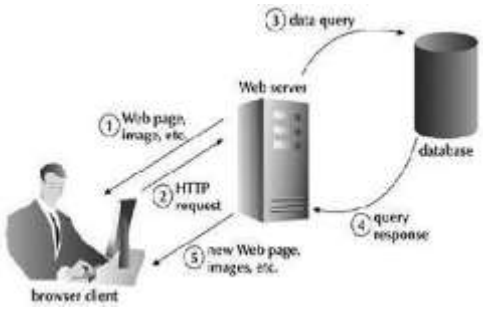
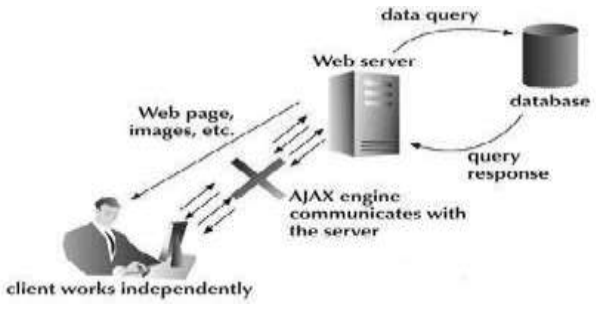


Fig 5.1: Working of AJAX

Synchronous Request	Asynchronous Request
	

5.1 CLIENT SERVER ARCHITECTURE

The Ajax provides a rich and diverse set of product that range from hundreds of suppliers. This diversity provides IT managers and Web developers with the ability to choose the optimal architectural approach and best products among multiple vendors.

Most Ajax technologies transform a platform-independent definition of the application into the appropriate HTML and JavaScript content that is then processed by the browser to deliver a rich user experience. Some Ajax designs perform most of their transformations on the client. Others perform transformations on the server.

Client side vs server side transformations

- **Client-side Ajax transformations**
  - With client-side Ajax, the Ajax engine runs on the client.
  - The server delivers Web content (HTML, CSS, JavaScript, etc.) which is processed by the client-side Ajax engine into revised Web content.

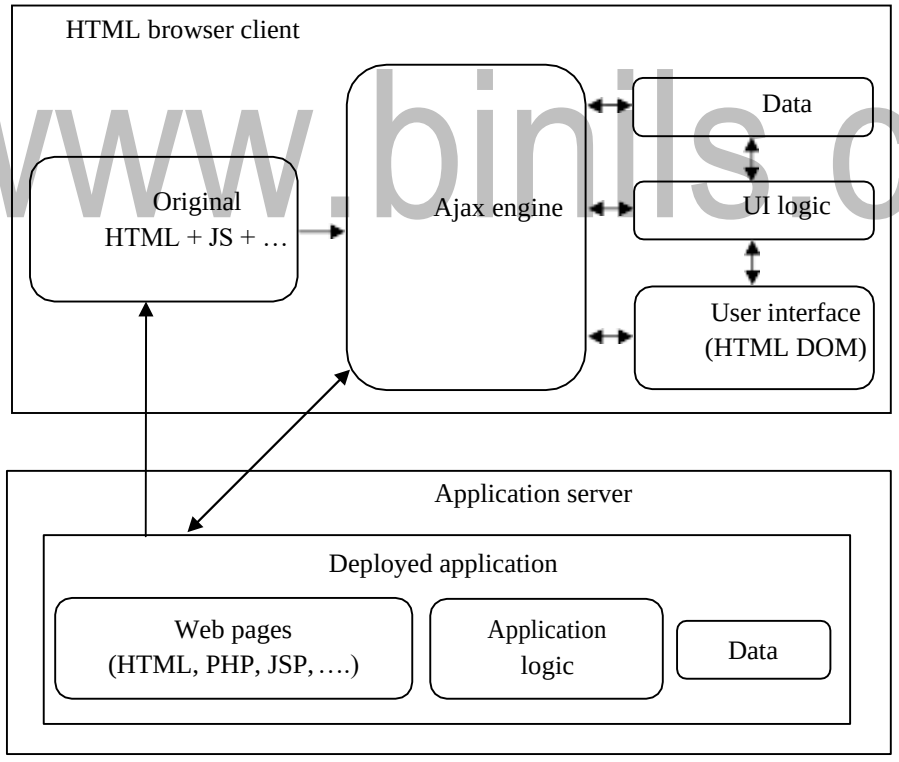


Fig 5.2 AJAX Client transformations

- The browser renders the revised HTML/etc. content that comes out of the Ajax engine.

- With this architecture, the application development team typically provides the following server-side components: Web pages (e.g., *.html, *.php, *.jsp, *.asp), application logic (e.g., Java) and data management (e.g., via a SQL database and/or Web Services)
- The client-side component includes client-side user interface logic, such as event handlers.
- The advantage of this option is the independence from the server side technology.
- The server code creates and serves the page and responds to the client's asynchronous requests.

This way either side can be swapped with another implementation approach**.**

➢ **Server-side transformations**

- For server-side Ajax, an Ajax server component performs most or all of the Ajax transformations.

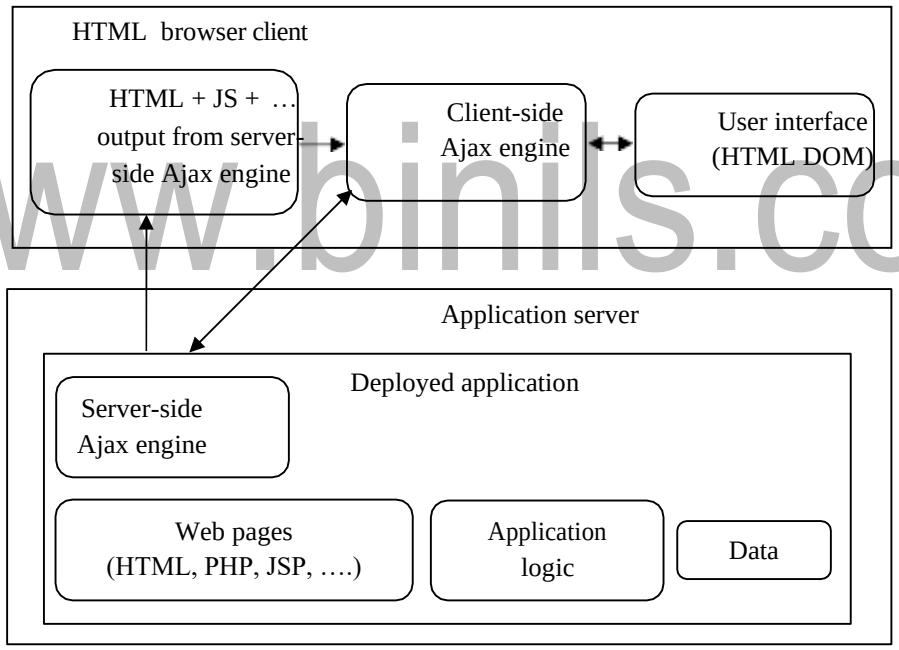


Fig 5.3 AJAX Server transformations

- The server component generates the necessary Web content (HTML, CSS, JavaScript, etc.) to deliver the desired user experience.
- The server-side Ajax toolkit downloads its own client-side Ajax library which communicates directly with the toolkit's server-side Ajax component.

- With this architecture, the application development team typically only provides server-side components (Web pages, application logic, and data management) and entrusts client-side logic to the Ajax toolkit.
- The main benefit of this approach is that it allows the use of server-side languages for debugging, editing, and refactoring tools with which developers are already familiar
- The disadvantage of this approach is the dependence on a particular server-side technology.
- As a general rule, server-side Ajax frameworks expect application code to be written in the server-side language.
- These frameworks typically hide all the JavaScript that runs in the browser inside widgets, including their events.

Single DOM vs Dual DOM

Single DOM

Some Ajax runtime toolkits use a Single-DOM approach where the toolkit uses the browser's DOM for both any original source markup and any HTML+JavaScript that results from the toolkit's Ajax-to-HTML transformation logic. This is same as Fig 5.2 In the above example, the developer is using tree widgets from an Ajax runtime library. The original HTML markup (un-shaded) and the additional HTML markup inserted by the Ajax toolkit (shaded) is given in the left. The DOM objects that correspond to the elements in the HTML markup (e.g., the DOM object that represents a particular element), where JavaScript/DOM objects from unshaded objects correspond to original HTML markup and shaded objects are ones that have been inserted by the Ajax toolkit is given in the right.

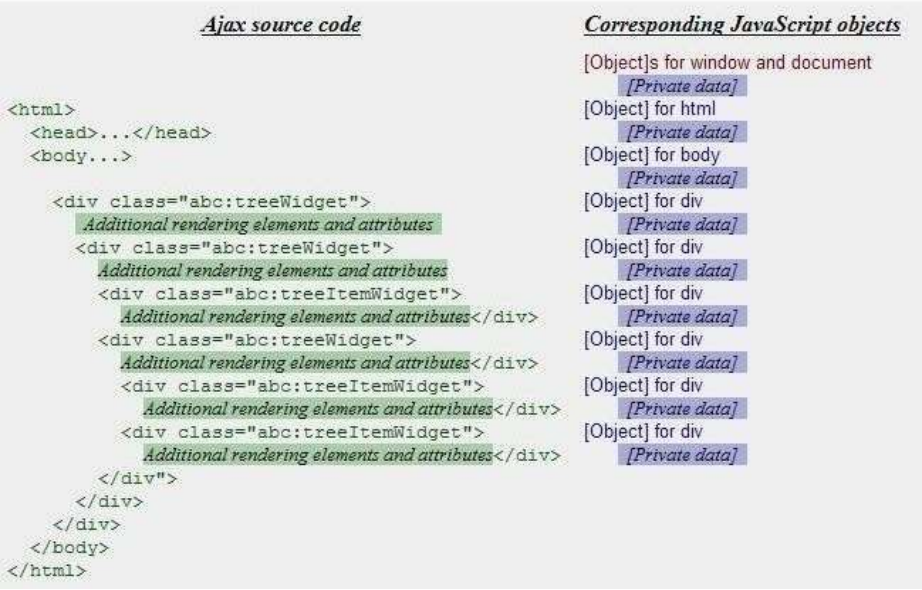


Fig 5.4 Manipulation of Single DOM

Typically, the Ajax toolkit inserts various rendering constructs such as <img> , , and <p> elements, inline within the original HTML markup (e.g., adding child elements to an existing element), thereby providing the various graphics and text necessary to produce the desired visual representation for the tree widgets. The shaded sections on the right reflect the private data that Ajax libraries typically add to various DOM and JavaScript objects in order to store private data, such as run-time state information. The Single-DOM approach is particularly well-suited for situations where the developer is adding Ajax capability within non-Ajax DHTML application.

Dual DOM

Other Ajax runtime toolkits adopt a Dual-DOM approach that separates the data for the Ajax-related markup into an Ajax DOM tree that is kept separate from the original Browser DOM tree. The Dual-DOM approach has two types:

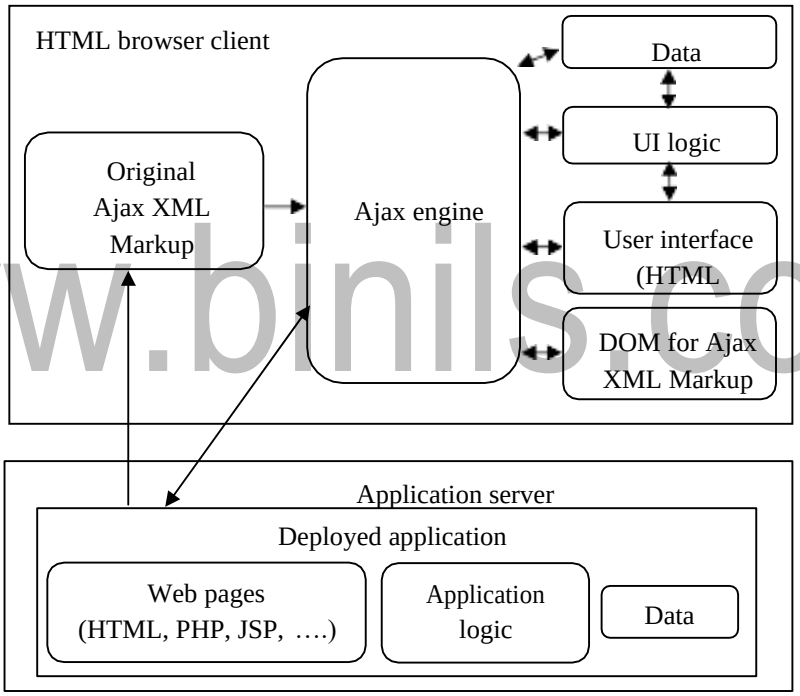


Fig 5.5 Dual DOM

With **Client-side Dual-DOM**, the second DOM tree typically consists of a separate tree of JavaScript objects. With **Server-side Dual-DOM,** the second DOM tree is stored on the server.

Fig 5.6 Manipulation of Dual DOM

There are two DOMs : the **HTML DOM and the XML DOM** corresponding to the Ajax-specific XML markup for the user interface elements.

The above example shows a separate file, "myapp.abc", which contains the user interface definition for the tree widgets, which in this case are to be placed into the HTML tree inside the element with id="abctarget". Even though the example shows the use of a separate file, some Dual-DOM Ajax runtime libraries support inline XML. In either case, a Dual-DOM Ajax runtime library builds a separate DOM tree, typically using its own XML parser rather than relying on the browser's HTML parser.

Sometimes the separate DOM tree is attached to the 'window' or 'document' objects. With this approach, in model view controller (MVC) terms, the Ajax DOM can be thought of as the model, the Browser DOM as the generated view, and the Ajax runtime toolkit as the controller. It is usually necessary to establish bidirectional event listeners between the Ajax DOM and the Browser DOM in order to maintain synchronization. Sometimes having a separate Ajax DOM enables a more complete set of XML and DOM support, such as full support for XML Namespaces, than is possible in the Browser DOM.

Dual-DOM (server-side)

Some Ajax technologies combine server-side Ajax transformations with a Dual-DOM approach. The key difference between Server-side Dual-DOM and Client-side Dual-DOM is that, with Server-side Dual-DOM, the Ajax DOM and most user interface logic is managed on the server. In this scenario, the primary job of the client Ajax engine is to reflect back to the server any interaction state changes, deferring data handling, UI state management and UI update logic to the server. Server-side Dual-DOM enables tight application integration with server-side development technologies such as Java Server Faces (JSF).

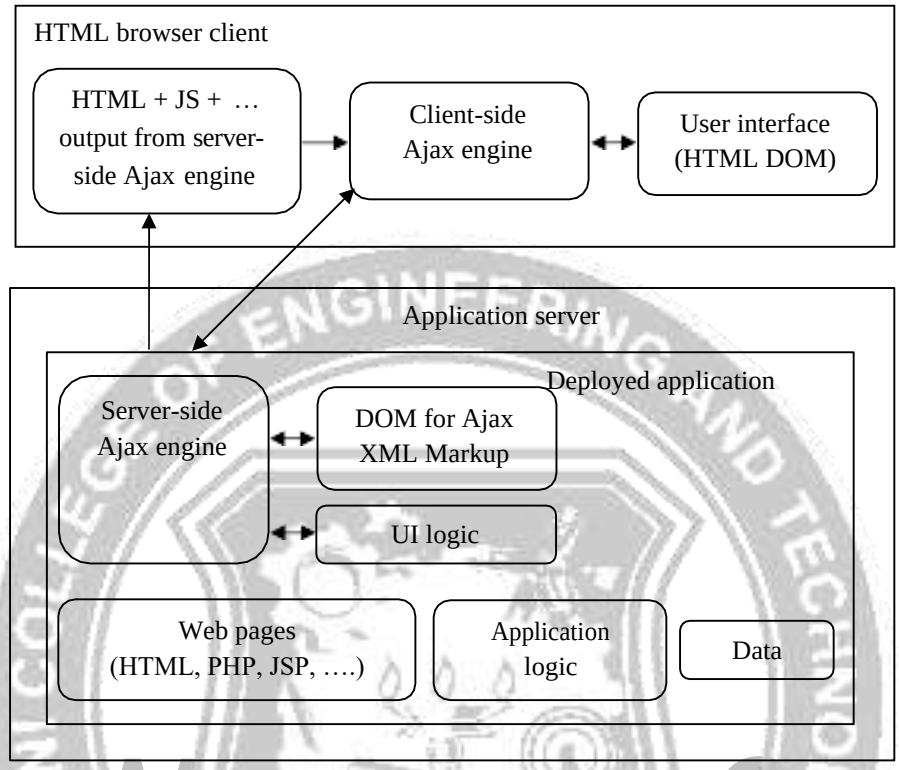


Fig 5.7 Server side Dual DOM

5.4 DEVELOPING A WEB SERVICE

Use Web services tools to discover, create, and publish Web services that are created from Java beans, enterprise beans, and WSDL files.

Creating a Web Service from WSDL

Start from WSDL to build the web service to implement a web service that is already defined either by a standard or an existing instance of the service. In either case, the WSDL already exists. The JAX-WS import tool processes the existing WSDL document, either from a local copy on disk or by retrieving it from a network address or URL. When developing a web service starting from an existing WSDL, the process is actually simpler than starting from

Java. This is because the policy assertions needed to enable various WSIT technologies are already embedded in the WSDL file.

To create a web service from WSDL, create the following source files: WSDL File, Web Service Implementation File, custom-server.xml , web.xml, sun-jaxws.xml, build.xml, build.properties

The following files are standard files required for JAX-WS. custom-server.xml, sun-jaxws.xml and web.xml

The build.xml and build.properties files are standard in any build environment.

WSDL File

```
<wsp:Policywsu:Id="AddNumbers_policy">
<wsp:ExactlyOne> <wsp:All> <wsrm:RMAssertion>
<wsrm:InactivityTimeout Milliseconds="600000"/>
<wsrm:AcknowledgementInterval Milliseconds="200"/>
</wsrm:RMAssertion> </wsp:All> </wsp:ExactlyOne> </wsp:Policy>
```

Web Service Implementation File

AddNumbersImpl.java

```
package fromwsdl.server; import javax.jws.WebService;
import javax.jws.WebMethod;
@WebService (endpointInterface= "fromwsdl.server.AddNumbersPortType")
public class AddNumbersImpl{
  @WebMethod(action="addNumbers")
  public intaddNumbers (int number1, int number2)
```

```
        throw new AddNumbersFault_Exception(message, fault); }
      return number1 + number2; }
    public void oneWayInt(int number) {
  System.out.println("Service received: " + number); } }
```

Publishing a Web service

The Web service, also known as the business service, describes a Web service's endpoint and where its WSDL file resides. The WSDL file lists the operations that service provides.

Prerequisites: Register with a registry, Launch the Web Services Explorer, Add the registry to the Web Services Explorer, Create a Web service, Deploy the Web service, Publish a Business Entity.

Web services are published using two different publication formats: simple and advanced.

Publish a business service using the simple option

- In the Web Services Explorer, select UDDI Main and select the registry to which the users want to publish the business entity.
- In the Actions pane toolbar, click the Publish icon Picture of the Publish icon.
- Select Service and select the Simple radio button.
- Enter the publish URL, your user ID, password, WSDL URL, service name, and service description in the respective fields.
- Click Go to publish your business entity.

Ensure that you select the service document, since the service element is the basis for the Business Service that you will publish. The Web Services Explorer is automatically updated with your published Web service. The registry contains pointers to the URL of the WSDL service document of the Web service. Businesses can now discover and integrate with your Web service.

Publish a Web service using the advanced option:

In the Web Services Explorer, select UDDI Main and select the registry to which the users want to publish the business entity.

- In the Actions pane toolbar, click the Publish icon Picture of the Publish icon.
- Select Service and select the Advanced radio button.
- Enter the publish URL, your user ID, password, and WSDL URL in the respective fields.

- Click Get or Find to associate the service with a business entity.
- Click Get or Find to associate a specific service interface with the service.
- Click Add to create service names.
- Click Add to create service descriptions.
- Click Add to create categories. Enter your service categories. Select a category type from the drop down list.
- Click Browse to open the Categories pane.
- Navigate through the hierarchical taxonomy and select the appropriate classification for your business service, then exit the Categories pane.
- Click Go to publish your business entity.

Testing a web service

The loosely coupled nature of web services and non-existence of a User interface present a challenge to the developers and testers alike. Following are some of the challenges that web service testers have to face: Scalability and Security, Absence of User Interface, Distributed across network and Testing the service

Types of testing

As with traditional applications, there are different sorts of testing that are needed to be carried out in case of web services.

- **Proof of concept Testing**

  Web service is a new concept and because of this we need to make sure that the architecture that we have chosen for our application is a correct one.

- **Functional testing**

  Web service is designed to solve a business problem. It has a predefined function to perform. This type of testing validates whether the service performs the intended function correctly, does it handle the exception conditions gracefully and does it handle the boundary value conditions.

- **Regression testing**

  Regression testing aims to ensure that the web service is still working across builds or releases. This sort of testing needs to be carried out during each release; hence it is an ideal candidate for automation.

- **Load testing**

  Load or stress testing is a test of the performance of the web service when many simultaneous users are accessing the system. The response of the web service must be

consistent and also its performance must not degrade with the increase in the number of users.

- **Unit testing**

  Unit test cases must be written before the application is developed. As and when the application is built, test cases are applied on the code. Hence the functionality is verified as and when we develop the web service.

- **Basic testing**

  The main aim of this testing is to test whether the web service is accessible and can be invoked properly. Main focus in this phase should be to carry out the following procedures.

  - Get the WSDL file and test whether it is well-formed and in compliance with the WSDL specifications published by W3C
  - Using this WSDL file generate the client side stubs that handle the interaction with the web service.
  - Test the web service functionality that is whether the webs ervice responds to the requests submitted to it correctly.
  - Invoke the sample invoker by passing it the parameters required by the web service. Check the response of the web service from a functionality point of view.
  - The sample invoker calls the client stubs which further call the web service
  - The stub constructs the SOAP message from the parameters passed to it and passes this message to the service. This message can be monitored by a Sniffer program like TCP Monitor.
  - If there are any security checks, like username and password we need to test their effectiveness. The intent of this step should be to break in the system and gain unauthorized access.

- **Testing SOA**

  As organizations create a web service interface to their systems and overcome security issues, they will be able to exchange data with business entities such as customers, suppliers and partners in a more uninhibited and loosely coupled manner. For testing such collaborating web services we need to focus on the following:

  - In a system where web services interact with each other, we need to test the 'publish', 'find' and 'bind' capabilities of the constituent web services.
  - A particular SOAP message may typically have a designated recipient, but may also have one or more intermediaries along the message route that take actions based upon the instructions provided to them in the header of the SOAP message. Web services testing must verify the proper functionality of these intermediaries also.

- **Interoperability testing**

  In the loosely coupled environment of a service-oriented architecture, separate sources don't need to know the details of each other's working, but they need to have enough common ground for reliably exchanging messages without error or misunderstanding. Standardized specifications help in creating such a common ground, but differences in implementation may still cause problems in the communication. Interoperability is when services can interact with each other without encountering such problems.

5.6 SIMPLE OBJECT ACCESS PROTOCOL (SOAP)

SOAP is an XML-based protocol for exchanging information between computers.
SOAP is an application of the XML specification.

Features of SOAP:

- SOAP is a communication protocol for Internet
- SOAP can extend HTTP for XML messaging
- SOAP provides data transport for Web services
- SOAP can exchange complete documents or call a remote procedure
- SOAP can be used for broadcasting a message
- SOAP is platform and language independent
- SOAP is the XML way of defining what information gets sent and how
- SOAP enables client applications to easily connect to remote services and invoke remote methods.

SOAPMessage Structure

A SOAP message is an ordinary XML document containing the following elements.

- **Envelope:** (Mandatory) Defines the start and the end of the message.

- **Header:** (Optional) Contains any optional attributes of the message used in processing the message, either at an intermediary point or at the ultimate end point.
- **Body:** (Mandatory) Contains the XML data comprising the message being sent.
- **Fault:** (Optional) An optional Fault element that provides information about errors that occurred while processing the message

SOAP Envelope Element

The SOAP envelope indicates the start and the end of the message so that the receiver knows when an entire message has been received. The SOAP envelope is a packaging mechanism. Every SOAP message has a root Envelope element. Every Envelope element must contain exactly one Body element. If an Envelope contains a Header element, it must contain no more than one, and it must appear as the first child of the Envelope, before the Body. The envelope changes when SOAP versions change. The SOAP envelope is specified using the ENV namespace prefix and the Envelope element. The optional SOAP encoding is also specified using a namespace name and the optional encoding Style element, which could also point to an encoding style other than the SOAP one.

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


SOAP Header Element

The optional Header element offers a flexible framework for specifying additional application-level requirements. For example, the Header element can be used to specify a digital signature for password-protected services; likewise, it can be used to specify an account number for pay-per-use SOAP services.

Header elements can occur multiple times. Headers are intended to add new features and functionality. The SOAP header contains header entries defined in a namespace. The header is encoded as the first immediate child element of the SOAP envelope. When more than one header is defined, all immediate child elements of the SOAP header are interpreted as SOAP header blocks.

SOAP Header element can have following two attributes

a) **Actor attribute:** The SOAP protocol defines a message path as a list of SOAP service nodes. Each of these intermediate nodes can perform some processing and then forward the message to the next node in the chain. By setting the Actor attribute, the client can specify the recipient of the SOAP header.

b) **Must Understand attribute**: Indicates whether a Header element is optional or mandatory. If set to true ie. 1 the recipient must understand and process the Header attribute according to its defined semantics, or return a fault.

SOAP Header

```
<?xml version="1.0"?> <SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://www.w3.org/2001/12/soap-envelope"
SOAP-ENV:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<SOAP-ENV:Header>
<t:Transaction
xmlns:t="http://www.tutorialspoint.com/transaction/"
SOAP-ENV:mustUnderstand="true">5</t:Transaction>
</SOAP-ENV:Header>
....
</SOAP-ENV:Envelope>
```


SOAP Body Element

The SOAP body is a mandatory element which contains the application-defined XML data being exchanged in the SOAP message. The body must be contained within the envelope and must follow any headers that might be defined for the message. The body is defined as a child element of the envelope, and the semantics for the body are defined in the associated SOAP schema. The body contains mandatory information intended for the ultimate receiver of the message.

Example: SOAP Body

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope
.....
<SOAP-ENV:Body>
<m:GetQuotationResponsexmlns:m="http://www.tp.com/Quotation">
```

```
<m:Quotation>This is Qutation</m:Quotation>
</m:GetQuotationResponse> </SOAP-ENV:Body> </SOAP-ENV:Envelope>
```

SOAP Fault Element

When an error occurs during processing, the response to a SOAP message is a SOAP fault element in the body of the message, and the fault is returned to the sender of the SOAP message. The SOAP fault mechanism returns specific information about the error, including a predefined code, a description, the address of the SOAP processor that generated. A SOAP Message can carry only one fault block. Fault element is an optional part of SOAP Message For the HTTP binding, a successful response is linked to the 200 to 299 range of status codes; SOAP fault is linked to the 500 to 599 range of status codes.

Sub Element	Description
<faultCode>	A text code used to indicate a class of errors.
<faultString>	A text message explaining the error
<faultActor>	A text string indicating who caused the fault. This is useful if the SOAP message travels through several nodes in the SOAP message path, and the client needs to know which node caused the error. A node that does not act as the ultimate destination must include a faultActor element.
<detail>	An element used to carry application-specific error messages. The detail element can contain child elements, called detail entries.

SOAP Fault Codes

The faultCode values must be used in the faultcode element when describing faults

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/1999/XMLSchema"> <SOAP-ENV:Body>
<SOAP-ENV:Fault>
<faultcodexsi:type="xsd:string">SOAP-ENV:Client</faultcode>
<faultstringxsi:type="xsd:string">
    Failed to locate method (ValidateCreditCard) in class
    (examplesCreditCard) at /usr/local/ActivePerl-5.6/lib/
```

```
        site_perl/5.6.0/SOAP/Lite.pm line 1555. </faultstring>
 </SOAP-ENV:Fault> </SOAP-ENV:Body> </SOAP-ENV:Envelope>
```

SOAP Encoding

SOAP includes a built-in set of rules for encoding data types. This enables the SOAP message to indicate specific data types, such as integers, floats, doubles, or arrays. SOAP data types are divided into two broad categories: scalar types and compound types. Scalar types contain exactly one value, such as a last name, price, or product description. Compound types contain multiple values, such as a purchase order or a list of stock quotes. Compound types are further subdivided into arrays and structs. Structs contain multiple values, but each element is specified with a unique accessor element.

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://www.w3.org/2001/12/soap-envelope"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<SOAP-ENV:Body>
<ns1:getProductResponse
 xmlns:ns1="urn:examples:productservice"
 SOAP-ENV:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<return xmlns:ns2="urn:examples" xsi:type="ns2:product">
<name xsi:type="xsd:string">Red Hat Linux</name>
<price xsi:type="xsd:double">54.99</price>
<description xsi:type="xsd:string">
      Red Hat Linux Operating System
</description>
<SKU xsi:type="xsd:string">A358185</SKU>
</return> </ns1:getProductResponse> </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```


SOAP

In this example, a GetQuotation request is sent to a SOAP Server over HTTP. The request has a QuotationName parameter, and a Quotation will be returned in the response.

The namespace for the function is defined in "http://www.xyz.org/quotation" address.

SOAP request:

```
POST /Quotation HTTP/1.0
Host: www.xyz.org
Content-Type: text/xml; charset=utf-8
Content-Length: nnn
<?xml version="1.0"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://www.w3.org/2001/12/soap-envelope"
SOAP-ENV:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<SOAP-ENV:Bodyxmlns:m="http://www.xyz.org/quotations">
<m:GetQuotation>
<m:QuotationsName>MiscroSoft</m:QuotationsName>
</m:GetQuotation> </SOAP-ENV:Body> </SOAP-ENV:Envelope>
```

SOAP response:


```
HTTP/1.0 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: nnn
<?xml version="1.0"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://www.w3.org/2001/12/soap-envelope"
SOAP-ENV:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<SOAP-ENV:Bodyxmlns:m="http://www.xyz.org/quotation">
<m:GetQuotationResponse>
<m:Quotation>Here is the quotation</m:Quotation>
</m:GetQuotationResponse> </SOAP-ENV:Body> </SOAP-ENV:Envelope>
```

Advantages and Disadvantages of SOAP

- **Language neutrality:** SOAP can be developed using any language.
- **Interoperability and Platform Independence:** SOAP can be implemented in any language and can be executed in any platform.

- **Simplicity:** SOAP messages are in very simple XML format.
- **Scalability:** SOAP uses HTTP protocol for transport due to which it becomes scalable.

Disadvantages of SOAP

- **Slow:** SOAP uses the XML format which needs to be parsed and is lengthier too which makes SOAP slower than CORBA, RMI or IIOP.
- **WSDL Dependence:** It depends on WSDL and does not have any standardized mechanism for dynamic discovery of the services.

Differences between SOAP and HTTP

SOAP	HTTP
It is a protocol for accessing web services and based on XML.	Http (HyperText Transfer Protocol) is a transfer used protocol, which called a stateless protocol because each command is executed independently, without any knowledge of the commands that came before it.
SOAP provides a way to communicate between applications running on different operating systems, with different technologies and programming languages.	This is the main reason that it is difficult to implement Web sites that react intelligently to user input.

5.5 WEB SERVICE DESCRIPTION LANGUAGE (WSDL)

WSDL stands for Web Services Description Language. It is the standard format for describing a web service. WSDL was developed jointly by Microsoft and IBM.

Features of WSDL

- WSDL is an XML-based protocol for information exchange in decentralized and distributed environments.
- WSDL definitions describe how to access a web service and what operations it will perform.

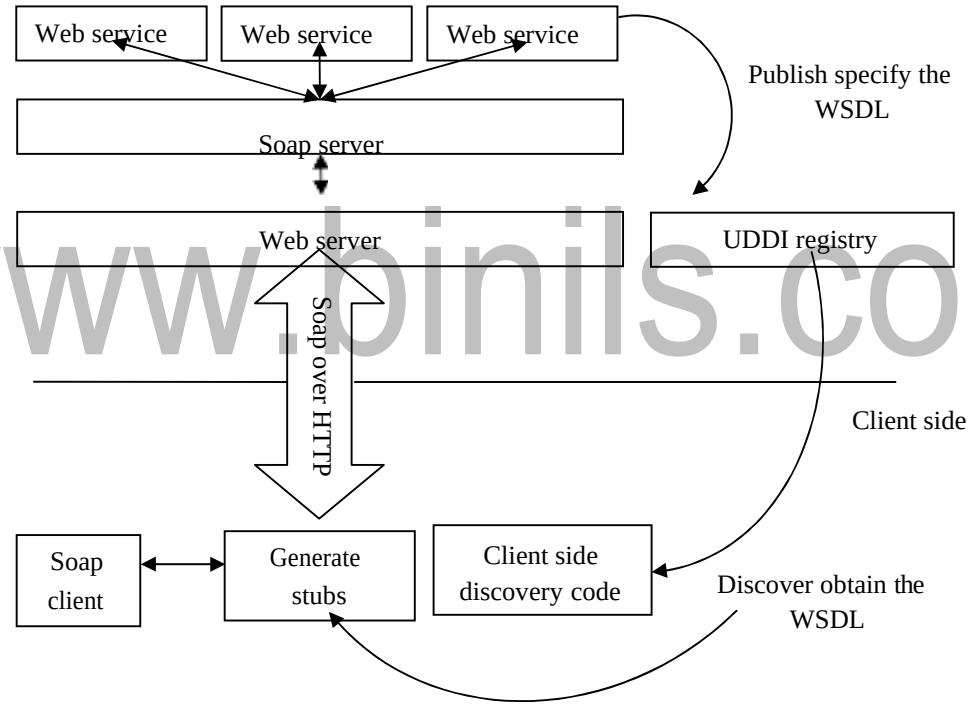


Fig 5.9 Architecture of Web service

- WSDL is a language for describing how to interface with XML-based services.
- WSDL is an integral part of Universal Description, Discovery, and Integration (UDDI), an XML-based worldwide business registry.
- WSDL is the language that UDDI uses.

WSDL addresses this need by defining an XML grammar for describing network services as collections of communication endpoints capable of exchanging messages. WSDL service definitions provide documentation for distributed systems and serve as a recipe for automating the details involved in applications communication.

Steps in providing service:

The following figure illustrates the use of WSDL. At the left is a service provider. At the right is a service consumer. The steps involved in providing and consuming a service are:

- A service provider describes its service using WSDL. This definition is published to a repository of services. The repository could use Universal Description, Discovery, and Integration (UDDI). Other forms of directories could also be used.

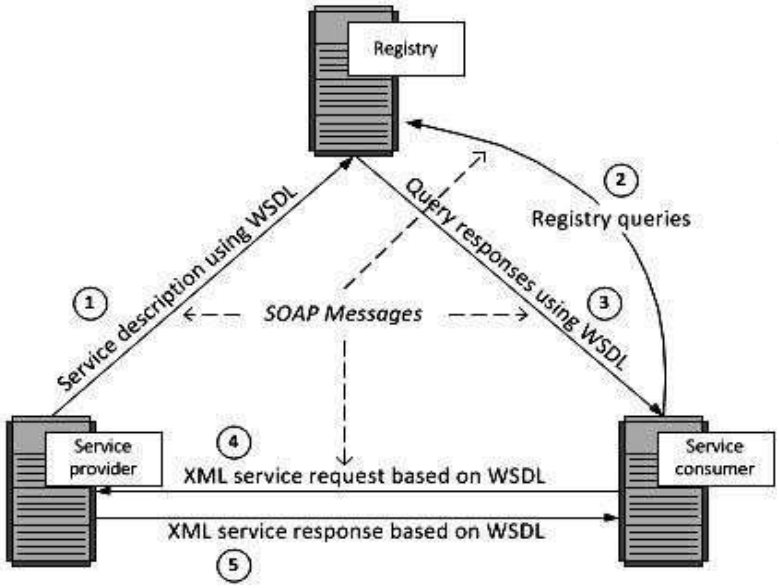


Fig 5.10 WSDL service

- A service consumer issues one or more queries to the repository to locate a service and determine how to communicate with that service.
- Part of the WSDL provided by the service provider is passed to the service consumer. This tells the service consumer what the requests and responses are for the service provider.

- The service consumer uses the WSDL to send a request to the service provider.
- The service provider provides the expected response to the service consumer.

WSDL Document

A WSDL document defines services as collections of network endpoints, or ports. In WSDL, the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. A port is defined by associating a network address with a reusable binding, and a collection of ports define a service.

WSDL Elements

WSDL breaks down web services into three specific, identifiable elements that can be combined or reused once defined: Types, Operations and Binding. A WSDL document has various elements, but they are contained within these three main elements, which can be developed as separate documents and then they can be combined or reused to form complete WSDL files. A WSDL document contains the following elements:

- **Definition**: It is the root element of all WSDL documents. It defines the name of the web service, declares multiple namespaces used throughout the remainder of the document, and contains all the service elements described here.
- **Data types:** The data types to be used in the messages are in the form of XML schemas.
- **Message:** It is an abstract definition of the data, in the form of a message presented either as an entire document or as arguments to be mapped to a method invocation.
- **Operation:** It is the abstract definition of the operation for a message, such as naming a method, message queue, or business process, that will accept and process the message.
- **Port type:** It is an abstract set of operations mapped to one or more end-points, defining the collection of operations for a binding; the collection of operations, as it is abstract, can be mapped to multiple transports through various bindings.
- **Binding:** It is the concrete protocol and data formats for the operations and messages defined for a particular port type.
- **Port:** It is a combination of a binding and a network address, providing the target address of the service communication.
- **Service:** It is a collection of related end-points encompassing the service definitions in the file; the services map the binding to the port and include any extensibility definitions.
- **Documentation:** This element is used to provide human-readable documentation and can be included inside any other WSDL element.
- **Import:** This element is used to import other WSDL documents or XML Schemas.

Document Structure of WSDL

```
<definitions> <types>
definition of types..... </types>
 <message> definition ofa message.....</message>
 <portType> <operation> definition of a operation ..... </operation>
 </portType>
 <binding> definition ofa binding. .... </binding>
 <service> definition of a service..... </service> </definitions>
```

CONSUMING A WEB SERVICE

A web service can be consumed (or called) by a client application. Different types of client applications can consume a web service. In today's software environment, almost every application needs a web service to enhance its functionality. The important advantage of a web service is that it returns its results in xml format, which can be consumed by different types of clients like browser based clients, rich desktop clients, spreadsheets, wireless devices, interactive voice response(IVR) systems and other business applications.

A client application discovers a web service, and then uses services provided by the web service. This process is known as **consuming a Web service.**

Creating Web Ports

Web ports are specially configured ports that you use to consume (call) Web services. A Web port can contain multiple operations that represent a mix of one-way (request only) and two-way (request-response) Web methods. Each operation in a Web port represents one method of a Web service.

Adding Web References

A Web reference is a description of a Web service that is available to the project. A Web reference includes:

- A Universal Resource Locator (URL) for the Web service.
- A WSDL file that offers information about the service such as available methods, ports, and message types.
- A reference map (Reference.map).

When the user add a Web reference, all the Web methods for that Web service must be compatible with the Server.

DATABASE DRIVEN WEB APPLICATION

Web services enable application-to-application interaction over the Web, regardless of platform, language, or data formats. The key ingredients, including Extensible Markup Language (XML), Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), and Universal Description, Discovery, and Integration (UDDI), have been adopted across the entire software industry. The Database Web services technology is a database approach to Web services. It works in the following two directions:

- Accessing database resources as a Web service
- Consuming external Web services from the database

Oracle Database can access Web services through PL/SQL packages and Java classes deployed within the database.

Using Oracle Database as Web Services Provider

Web Services use industry-standard mechanisms to provide easy access to remote content and applications, regardless of the platform and location of the provider and implementation and data format. Client applications can query and retrieve data from Oracle Database and call stored procedures using standard Web service protocols. There is no dependency on Oracle-specific database connectivity protocols. This approach is highly beneficial in heterogeneous, distributed, and disconnected environments. Using Oracle Database as a Web service provider offers the following features:

- Enhances PL/SQL Web services
- Exposes Java in the database as Web services
- Provides SQL query Web services
- Enables DML Web services

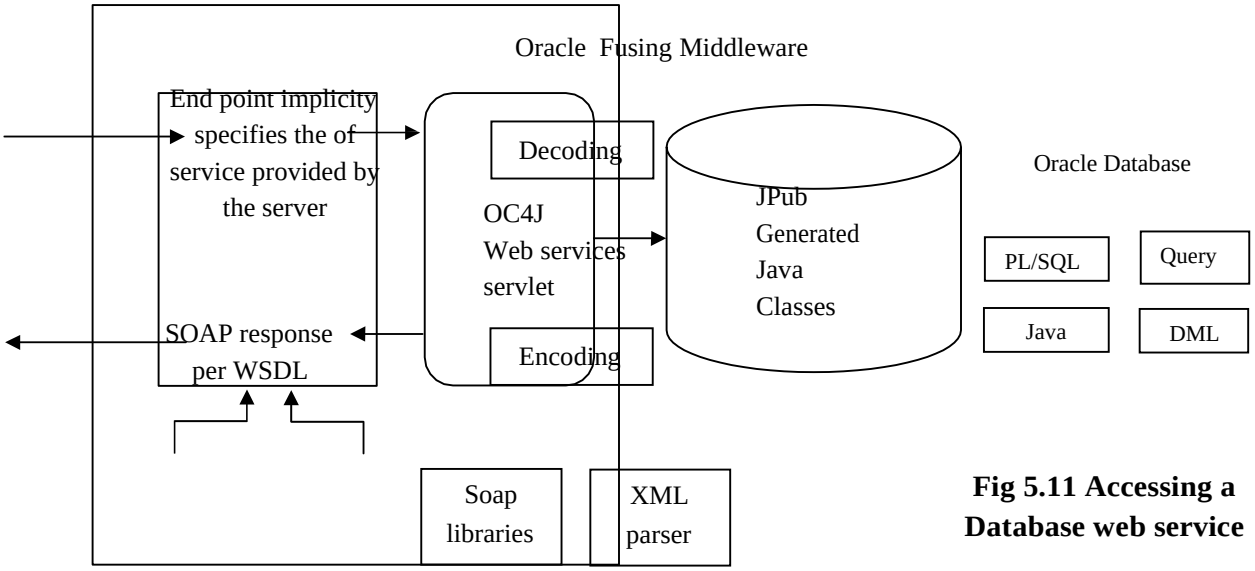


Fig 5.11 Accessing a Database web service

Using Oracle Database as Web Services Consumer

The storage, indexing, and searching capabilities of a relational database can be extended to include semi-structured and non-structured data, including Web services. By calling Web services, the database can track, aggregate, refresh, and query dynamic data produced on-demand, such as stock prices, currency exchange rates, and weather information.

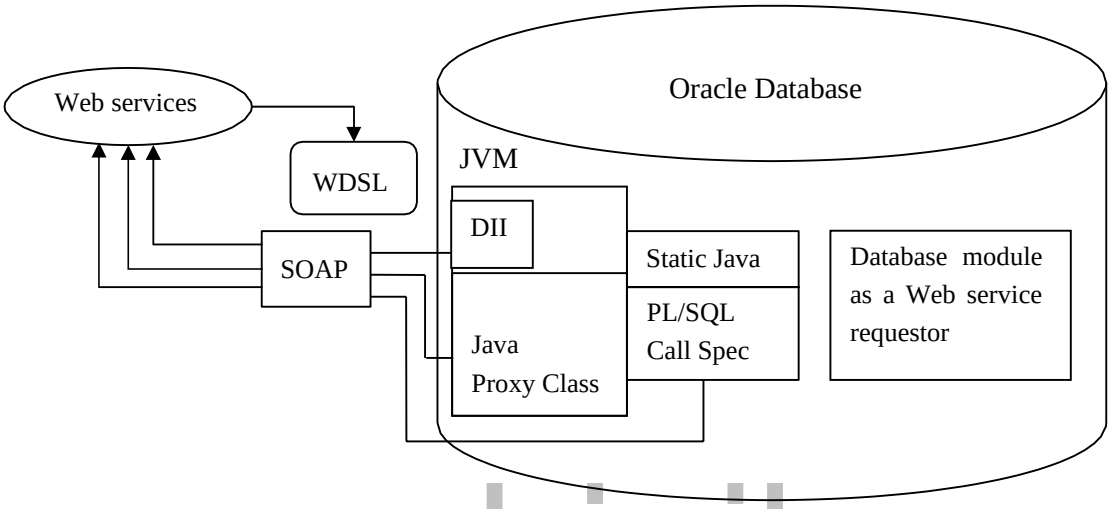


Fig 5.12 Calling web services within a database

Web Service Data Sources (Virtual Table Support)

To access data that is returned from single or multiple Web service invocations, create a virtual table using a Web service data source. This table lets the user to query a set of returned rows as though it were a table.

The client calls a Web service and the results are stored in a virtual table in the database. The result sets can be passed from function to function. This enables the user to set up a sequence of transformation without a table holding intermediate results. By using Web services with the table function, a range of input values can be manipulated fr om single or multiple Web services as a real table.

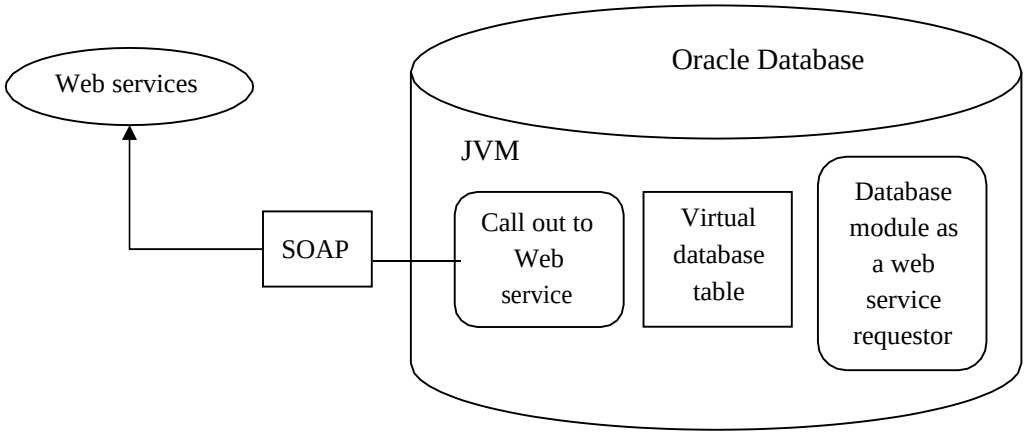


Fig 5.13 Virtual table

5.3 WEB SERVICES

A web service is a collection of open protocols and standards used for exchanging data between applications or systems.

Web services are a technology, a process, and a phenomenon. Web services are built on top of open standards such as TCP/IP, HTTP, Java, HTML, and XML. XML is used to encode all communications to a web service. They include programs, objects, messages, or documents. They are available over the Internet or private (intranet) networks. They are not tied to any one operating system or programming language. They are self-describing through a common XML grammar They are discoverable through a simple find mechanism

Components of Web Services

The basic web services platform is XML + HTTP. All the standard web services work using the following components:

- Java Web Services
- SOAP (Simple Object Access Protocol)
- UDDI (Universal Description, Discovery and Integration)
- WSDL (Web Services Description Language)

Working of Web service

A web service enables communication among various applications by using open standards such as HTML, XML, WSDL, and SOAP. A web service takes the help of:

- XML to tag the data
- SOAP to transfer a message
- WSDL to describe the availability of service.
- Java-based web service can also be built.

Example:

Consider a simple account-management and order processing system. The accounting personnel use a client application built with Visual Basic or JSP to create new accounts and enter new customer orders.

The processing logic for this system is written in Java and resides on a Solaris machine, which maintains a database. The following are the steps to perform an operation in a web service:

- The client program bundles the account registration information into a SOAP message.
- This SOAP message is sent to the web service as the body of an HTTP POST request.
- The web service unpacks the SOAP request and converts it into a command that the application can understand.
- The application processes the information as required and responds with a new unique account number for that customer.
- Then the web service packages the response into another SOAP message, which it sends back to the client program in response to its HTTP request.
- The client program unpacks the SOAP message to obtain the results of the account registration process.

Characteristics of Web Service

- **XML based:** Using XML eliminates any networking, operating system, or platform binding.
- **Loosely Coupled:** A tightly coupled system implies that the client and server logic are closely tied to one another, implying that if one interface changes, the other must be updated. Adopting a loosely coupled architecture tends to make software systems more manageable and allows simpler integration between different systems.
- **Coarse grained:** Web services technology provides a natural way of defining coarse-grained services that access the right amount of business logic.
- **Ability to be Synchronous or Asynchronous:** Synchronicity refers tothe binding of the client to the execution of the service. In synchronous invocations, the client blocks and waits for the service to complete its operation before continuing. Asynchronous operations allow a client to invoke a service and then execute other functions.
- **Supports Remote Procedure Calls (RPCs):** Web services support the transparent exchange of documents to facilitate business integration.

Advantages of Web services: Interoperability, Reusability, Platform independent, Coarse-grained, Low communication cost, Standardized protocol, Exposing the existing function on the network

JAVA WEB SERVICES

Java web services has two APIs: **JAX-WS and JAX-RS**. The java web service application can be accessed by other programming languages such as .Net and PHP. Java web service application perform communication through WSDL (Web Services Description Language).

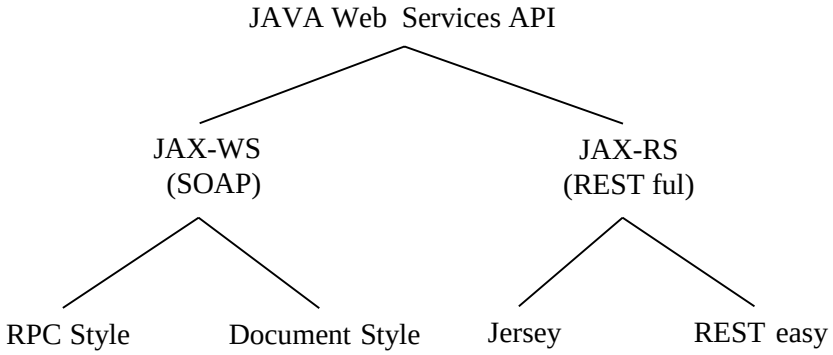


Fig 5.8 Java Web Services API

- **JAX-WS:** for SOAP web services. There are two ways to write JAX-WS application code: by RPC style and Document style.
- **JAX-RS:** for RESTful web services. There are mainly two implementations: Jersey and RESTeasy**.**

JAX-WS

There are two ways to develop JAX-WS example: RPC style and Document style.

1) AX-WS Example RPC Style

Creating JAX-WS example is a easy task because it requires no extra configuration settings. JAX-WS API is inbuilt in JDK, so no extra jar file is needed for it.

This simple application has four files:

- HelloWorld.java
- HelloWorldImpl.java
- Publisher.java
- HelloWorldClient.java

HelloWorld.java

```
import javax.jws.WebMethod;      import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;      import javax.jws.soap.SOAPBinding.Style;
//Service Endpoint Interface      @WebService
@SOAPBinding(style = Style.RPC)
public interface HelloWorld
{      @WebMethod String getHelloWorldAsString(String name);      }
```

HelloWorldImpl.java

```
import javax.jws.WebService;
@WebService(endpointInterface = "com.abc.HelloWorld")
public class HelloWorldImpl implements HelloWorld{
    @Override
    public String getHelloWorldAsString(String name) { //Implementation of the interface
        return "Hello World JAX-WS " + name;    }    }
```

Publisher.java

```
import  javax.xml.ws.Endpoint;       //Endpoint publisher
public class HelloWorldPublisher{
    public static void main(String[] args) {
Endpoint.publish("http://localhost:7779/ws/hello", new HelloWorldImpl());          }      }
```

HelloWorldClient.java

```
import  java.net.URL;      import javax.xml.namespace.QName;
import javax.xml.ws.Service;
public class HelloWorldClient{
    public static void main(String[] args) throws Exception {
    URL url =new URL("http://localhost:7779/ws/hello?wsdl");
    //1st argument service URI, refer to wsdl document above
    //2nd argument is service name, refer to wsdl document above
QNameqname=new QName("http://abc.com/","HelloWorldImplService");
        Service service = Service.create(url, qname);
HelloWorldhello=service.getPort(HelloWorld.class);
System.out.println(hello.getHelloWorldAsString("rpc"));          }      }
```

Hello World JAX-WSrpc

JAX-WS Example Document Style

Like RPC style, we can create JAX-WS example in document style. Use Style.DOCUMENT for @SOAPBinding annotation in place of Style.RPC.

HelloWorld.java

```
import javax.jws.WebMethod; import  javax.jws.WebService;
import javax.jws.soap.SOAPBinding; import javax.jws.soap.SOAPBinding.Style;
//Service Endpoint Interface
@WebService
@SOAPBinding(style = Style.DOCUMENT)
public interface HelloWorld
{ @WebMethod String getHelloWorldAsString(String name); }
```

HelloWorldImpl.java

```
import javax.jws.WebService; //Service Implementation
 @WebService(endpointInterface = "com.abc.HelloWorld")
 public class HelloWorldImpl implements HelloWorld
{ @Override public String getHelloWorldAsString(String name)
{ return "Hello World JAX-WS " + name; } }
```

Publisher.java

```
importjavax.xml.ws.Endpoint;//Endpointpublisher
public class HelloWorldPublisher
{ public static void main(String[] args)
{ Endpoint.publish("http://localhost:7779/ws/hello", new HelloWorldImpl()); } }
```

HelloWorldClient.java


```
import java.net.URL; import javax.xml.namespace.QName;
import javax.xml.ws.Service; public class HelloWorldClient
{ public static void main(String[] args) throws Exception
{ URL url = new URL("http://localhost:7779/ws/hello?wsdl");
//1st argument service URI, refer to wsdl document above
//2nd argument is service name, refer to wsdl document above
QNameqname = new QName("http://abc.com/", "HelloWorldImplService");
Serv
Hello
System.out.println(hello.getHelloWorldAsString("document"));} }
```

RPC style	Document style
RPC style web services use method name and parameters to generate XML structure. The generated WSDL is difficult to be validated against schema.	Document style web services can be validated against predefined schema.

Hello World JAX-WS document

Difference between RPC and Document web services

In RPC style, SOAP message is sent as many elements.	In document style, SOAP message is sent as a single document.
RPC style message is tightly coupled.	Document style message is loosely coupled
In RPC style, SOAP message keeps the operation name.	In Document style, SOAP message loses the operation name.
In RPC style, parameters are sent as discrete values.	In Document style, parameters are sent in XML format.

JAX-RS

There are two main implementation of JAX-RS API: Jersey and RESTEasy.

1) JAX-RS Example Jersey

We can create JAX-RS example by jersey implementation. To do so load jersey jar files or use maven framework. In this example, we are using jersey jar files for using jersey example for JAX-RS.

Here, we create the following four files: Hello.java, web.xml, index.html and HelloWorldClient.java

Hello.java

```
import javax.ws.rs.GET; import javax.ws.rs.Path;
import javax.ws.rs.Produces; import javax.ws.rs.core.MediaType;
@Path("/hello") public class Hello {
 // This method is called if HTML and XML is not requested
 @GET
 @Produces(MediaType.TEXT_PLAIN)
 public String sayPlainTextHello() {
 return  "Hello Jersey Plain";     }
 // This method is called if XML is requested
 @GET
 @Produces(MediaType.TEXT_XML)
 public String sayXMLHello() {
  return "<?xml version=\"1.0\"?>" + "<hello> Hello Jersey" + "</hello>";
 }
  // This method is called if HTML is requested
```

```
 @GET
 @Produces(MediaType.TEXT_HTML)
 public String sayHtmlHello() {
   return "<html> " + "<title>" + "Hello Jersey" + "</title>"
       + "<body><h1>" + "Hello Jersey HTML" + "</h1></body>" + "</html> ";
 }
}
```

Web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
   http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
   id="WebApp_ID" version="3.0">
<servlet>
<servlet-name>Jersey REST Service</servlet-name>
<servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
<init-param>
<param-name>jersey.config.server.provider.packages</param-name>
<param-value>com.abc.rest</param-value>
</init-param>
<load-on-startup>1</load-on-startup> </servlet>
<servlet-mapping>
<servlet-name>Jersey REST Service</servlet-name>
<url-pattern>/rest/*</url-pattern>
</servlet-mapping> </web-app>
```


Index.xml

```
<a href="rest/hello">Click Here</a>
```

Now run this application on server. Here we are using Tomcat server on port 4444. The project name is restfuljersey.

ClientTest.java

```
import java.net.URI;
import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder; import javax.ws.rs.client.WebTarget;
import javax.ws.rs.core.MediaType; import javax.ws.rs.core.UriBuilder;
import org.glassfish.jersey.client.ClientConfig;
public class ClientTest {
  public static void main(String[] args) {
ClientConfigconfig = new ClientConfig();
    Client client =ClientBuilder.newClient(config);
WebTarget target = client.target(getBaseURI());
    //Now printing the server code of different media type
System.out.println(target.path("rest").path("hello").request().accept(MediaType.TEXT_PLAIN).
get(String.class));
System.out.println(target.path("rest").path("hello").request().accept(MediaType.TEXT_XML).
get(String.class));
System.out.println(target.path("rest").path("hello").request().accept(MediaType.TEXT_HTML).
get(String.class));      }
  private static URI getBaseURI() {
    //here server is running on 4444 port number and project name is restfuljersey
    return  UriBuilder.fromUri("http://localhost:4444/restfuljersey").build();      } }
```



```
        Hello Jersey Plain
        <?xml version="1.0"?><hello> Hello Jersey</hello>
<html><title>Hello Jersey</title><body><h1>Hello Jersey HTML</h1></body></html>
```

JAX-RS Annotations

Annotations	Description
Path	It identifies the URI path. It can be specified on class or method.
PathParam	represents the parameter of the URI path.

GET	specifies method responds to GET request.
POST	specifies method responds to POST request.
PUT	specifies method responds to PUT request.
HEAD	specifies method responds to HEAD request.
DELETE	specifies method responds to DELETE request.
OPTIONS	specifies method responds to OPTIONS request.
FormParam	represents the parameter of the form.
QueryParam	represents the parameter of the query string of an URL.
HeaderParam	represents the parameter of the header.
CookieParam	represents the parameter of the cookie.
Produces	defines media type for the response such as XML, PLAIN, JSON etc. It defines the media type that the methods of a resource class or MessageBodyWriter can produce.
Consumes	It defines the media type that the methods of a resource class or MessageBodyReader can produce.

2) RESTful Web Services

REST stands for REpresentational State Transfer.REST is an architectural style not a protocol.

Advantages of RESTful Web Services

- **Fast**: RESTful Web Services are fast because there is no strict specification like SOAP. It consumes less bandwidth and resource.
- **Language and Platform independent**: RESTful web services can be written in any programming language and executed in anyplatform.
- **Can use SOAP**: RESTful web services can use SOAP web services as the implementation.
- **Permits different data format**: RESTful web service permits different data format such as Plain Text, HTML, XML and JSON.

We can download text files, image files, pdffiles, excel files in java by JAX-RS API. To do so we need to write few lines of code only. Here, we are using jersey implementation for developing JAX-RS file download examples. It is important to specify different content type to download different files. The @Produces annotation is used to specify the type of file content.

- @Produces("text/plain"): for downloading text file.
- @Produces("image/png"): for downloading png image file.
- @Produces("application/pdf"): for downloading PDF file.
- @Produces("application/vnd.ms-excel"): for downloading excel file.
- @Produces("application/msword"): for downloading ms word file.

FileDownloadService.java

```
import java.io.File;  import javax.ws.rs.GET;  import javax.ws.rs.Path;
import javax.ws.rs.Produces;  import javax.ws.rs.core.Response;
import javax.ws.rs.core.Response.ResponseBuilder;
@Path("/files")
public class FileDownloadService {
    private static final String FILE_PATH = "c:\\myfile.txt";
    @GET
    @Path("/txt")
    @Produces("text/plain")
    public Response getFile() {
        File file = new File(FILE_PATH);
ResponseBuilder response = Response.ok((Object) file);
response.header("Content-Disposition","attachment; filename=\" javapoint_file1.txt\"");
        return response.build();          } }
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
id="WebApp_ID" version="3.0">
<servlet>
<servlet-name>Jersey REST Service</servlet-name>
```

```
<servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
<init-param>
<param-name>jersey.config.server.provider.packages</param-name>
<param-value>com.javatpoint.rest</param-value>
</init-param>
<load-on-startup>1</load-on-startup> </servlet>
<servlet-mapping>
<servlet-name>Jersey REST Service</servlet-name>
<url-pattern>/rest/*</url-pattern>
</servlet-mapping> </web-app>
```

index.html

```
<a href="rest/files/txt">Download Text File</a>
```

Output:

5.2 XML Http Request Object and CALLBACK()

The XMLHttpRequest object is used to exchange data with a server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page. **XMLHttpRequest (XHR)** is an API that can be used by JavaScript, JScript, VBScript, and other web browser scripting languages to transfer and manipulate XML data to and from a webserver using HTTP, establishing an independent connection channel between a webpage's Client-Side and Server-Side. The data returned from XMLHttpRequest calls will often be provided by back-end databases. Besides XML, XMLHttpRequest can be used to fetch data in other formats, e.g. JSON or even plain text.

Creatingan XMLHttpRequest Object

Syntax:

variable=new XMLHttpRequest(); (new version)

variable=new ActiveXObject("Microsoft.XMLHTTP"); (old version)

Processing Requests in AJAX

The following are the sequence of operations request is initiated:

- A client event occurs.
- An XMLHttpRequest object is created.
- The XMLHttpRequest object is configured.
- The XMLHttpRequest object makes an asynchronous request to the Webserver.
- The Webserver returns the result containing XML document.
- The XMLHttpRequest object calls the callback() function and processes the result.
- The HTML DOM is updated.

- **A client event occurs:**

- A JavaScript function is called as the result of an event.
  - **Example:** validateUserId() JavaScript function is mapped as an event handler to an onkeyup event on input form field whose id is set to "userid".

<input type="text" size="20" id="userid" name="id"onkeyup="validateUserId();">.

- **XMLHttpRequest object is created**

```
varajaxRequest; // AJAX variable
functionajaxFunction()
{ try
{ // This code is for browsers Opera 8.0+, Firefox, Safari
ajaxRequest =new XMLHttpRequest();          }
catch (e)
    {    // Internet Explorer Browsers
Try          {
ajaxRequest  = new  ActiveXObject("Msxml2.XMLHTTP");          }
catch (e)       {
try{
ajaxRequest = new ActiveXObject("Microsoft.XMLHTTP");                    }
catch (e){
                    // Something went wrong
alert("Your browser broke");
return false;                        }            }       } }
```

- **The XMLHttpRequest object is configured**

```
functionvalidateUserId()
{ ajaxFunction(); // Here processRequest() is the callback function.
ajaxRequest.onreadystatechange = processRequest;
if (!target) target = document.getElementById("userid");
varurl = "validate?id=" + escape(target.value);
ajaxRequest.open("GET", url, true);
ajaxRequest.send(null);       }
```

- **Making an asynchronous request to the Webserver**

  This is done using the XMLHttpRequest object ajaxRequest. Assume that the user enters Sona in the userid box, then in the above request, the URL is set to "validate?id=Sona".

- **Webserver Returns the Result Containing XML Document**

  Server-side script is implemented as follows:

  - Get a request from the client.
  - Parse the input from the client.
  - Do required processing.
  - Send the output to the client.

If we assume that the user is going to write a servlet, then:

```
public void doGet(HttpServletRequest request, HttpServletResponse response) throws
IOException, ServletException
{    String targetId = request.getParameter("id");
if ((targetId != null) && !accounts.containsKey(targetId.trim()))
  { response.setContentType("text/xml");
response.setHeader("Cache-Control", "no-cache");
response.getWriter().write("true"); }
else    {
response.setContentType("text/xml");
response.setHeader("Cache-Control", "no-cache");
response.getWriter().write("false"); } }
```

- **Callback Function processRequest() is Called**

The callback function is responsible for checking the progress of requests, identifying the result of the request and handling data returned from the server. Callback functions also serve as delegators, handing off to other areas of the application code. The XMLHttpRequest object was configured to call the processRequest() function when there is a state change to the readyState of the XMLHttpRequest object. Now this function will receive the result from the server and will do the required processing. As in the following example, it sets a variable message on true or false based on the returned value from the Webserver.

```
functionprocessRequest()
{ if (req.readyState == 4)
        { if (req.status == 200)
                { var  message = ...;
                                ...
}
```

- **The HTML DOM is updated.**

This is the final step and in this step, the HTML page will be updated. It happens in the following way:

- JavaScript gets a reference to any element in a page using DOM API.
- The recommended way to gain a reference to an element is to call.

  document.getElementById("userIdMessage"), // userIdMessage is ID attribute

  // of an element appearing in the HTML document
- JavaScript may now be used to modify the element's attributes; modify the element's style properties; or add, remove, or modify the child elements.

```
<script type="text/javascript">
<!— functionsetMessageUsingDOM(message)
{ varuserMessageElement = document.getElementById("userIdMessage");
varmessageText;
if (message == "false")
{ userMessageElement.style.color = "red";
messageText = "Invalid User Id";     }
else    {
```

```
userMessageElement.style.color = "green";
messageText  = "Valid User Id";     }
varmessageBody = document.createTextNode(messageText);
if (userMessageElement.childNodes[0])
{   userMessageElement.replaceChild(messageBody,    userMessageElement.childNodes[0]);
}
 Else    { userMessageElement.appendChild(messageBody);      } }
</script> <body> <div id="userIdMessage"><div> </body>
```

XMLHttpRequest Methods

Method	Description
abort()	Cancels the current request.
getAllResponseHeaders()	Returns the complete set of HTTP headers as a string.
getResponseHeader(headerName)	Returns the value of the specified HTTP header
open(method, URL) open(method, URL, async) open(method, URL, async, userName) open(method, URL, async, userName, password)	Specifies the method, URL, and other optional attributes of a request. The method parameter can have a value of "GET", "POST", or "HEAD". Other HTTP methods, such as "PUT" and "DELETE" may be possible. The "async" parameter specifies whether the request should be handled asynchronously or not. "true" means that the script processing carries on after the send() method without waiting for a response, and "false" means that the script waits for a response before continuing script processing.
send(content)	Sends the request.
setRequestHeader(label, value)	Adds a label/value pair to the HTTP header to be sent.

XMLHttpRequest Properties

- **onreadystatechange:** An event handler for an event that fires at every state change.
- **readyState:** The readyState property defines the current state of the XMLHttpRequest object.

State	Description
0	The request is not initialized.
1	The request has been set up.
2	The request has been sent.
3	The request is in process.
4	The request is completed.

- readyState = 0 After the user have created the XMLHttpRequest object, but before the call of the open() method.
- readyState = 1 After the user have called the open() method, but before the call of send().
- readyState = 2 After the user have called send().
- readyState = 3 After the browser has established a communication with the server, but before the server has completed the response.
- readyState = 4 After the request has been completed, and the response data has been completely received from the server.

  - responseText:Returns the response as a string.
  - responseXML:Returns the response as XML. This property returns an XML document object, which can be examined and parsed using the W3C DOM node tree methods and properties.
  - Status:Returns the status as a number (e.g., 404 for "Not Found" and 200 for "OK").
  - statusText:Returns the status as a string (e.g., "Not Found" or "OK").