प्राविधिक शिक्षा परिषद, उत्तर प्रदेश द्वारा स्वीकृत, नवीन "NSQF" आधारित संशोधित पाठ्यक्रमानुसार

एडवांस्ड जावा
ADVANCED JAVA

(For the Students of VIth Semester (Third Year) of Computer Science Engineering & Allied Branches

VIBHA

प्रियम तायल
असिस्टेंट प्रोफेसर
गेटवे इन्स्टीटयूट ऑफ इंजीनियरिंग एण्ड टेकनोलॉजी
सोनीपत (हरियाणा)

पवन कुमार गुप्ता
विभागाध्यक्ष, सूचना प्रौद्योगिकी (आई०टी०)
चौ० मुख्तार सिंह राजकीय बालिका पॉलीटेक्निक
दौराला, मेरठ (उत्तर प्रदेश)

प्रद्युमन कुमार
भूतपूर्व आई०टी० विश्लेषक, टी०सी०एस० लि०
प्रवक्ता, एम०एम०आई०टी०
संत कबीर नगर (उत्तर प्रदेश)

2021

प्रकाशक:
रशियन पब्लिशर्स, मुज़फ़्फ़रनगर
46/20, कम्बल वाला बाग, नई मण्डी, मुज़फ़्फ़रनगर–251 001 (उ०प्र०)

एडवांस्ड जावा

प्रकाशकः
एशियन पब्लिशर्स, मुज़फ़्फ़रनगर
46/20, कम्बल वाला बाग, नई मण्डी,
मुज़फ़्फ़रनगर – 251 001 (उ०प्र०)
फोन: 0131-2660989
Visit us at : www.asianpublishers.co.in
email : spmittal@asianpublishers.co.in
sales@asianpublishers.co.in

इस पुस्तक का कोई भी अंश लेखक एवं प्रकाशक की लिखित पूर्वानुमति के बिना किसी भी रूप में तथा किसी भी माध्यम से, उद्धृत, अनुवादित या प्रकाशित नहीं किया जा सकता।

© सर्वाधिकार लेखक के अधीन है।

प्रथम संस्करण : 2021-2022

मूल्य : ₹ 250.00

मुद्रकः
विमल प्रिन्टर्स
मेरठ
फोन: 0-9412203584

यद्यपि इस पुस्तक को प्रकाशित करने में तथा इसमें दिये गये तथ्यों की यथार्थता को सुनिश्चित करने हेतु अत्यंत सावधानी बरती गयी है किन्तु फिर भी किसी त्रुटि या मिस्प्रिंट के लिये तथा उससे होने वाले किसी भी प्रकार के नुकसान के लिये लेखक या प्रकाशक किसी रुप में जिम्मेदार नहीं होंगे। पाठकों को सलाह दी जाती है कि किसी भी प्रकार के संशय की स्थिति में इस पुस्तक में दी गई सामग्री का मिलान मानक पुस्तकों से कर लें।

प्रस्तावना

लेखकों का इस पुस्तक को लिखने का इरादा विशेषरूप से डिप्लोमा के विद्यार्थियों के लिए **एडवांस्ड जावा** की बुनयादी पाठ्यपुस्तक को सरल भाषा में बनाने का है। इस पुस्तक को लिखते समय बहुत ही सरल भाषा का उपयोग किया गया है तथा टेक्निकल शब्दों को यथासंभव अंग्रेजी भाषा में भी देने का प्रयास किया है। इस पुस्तक की विषय वस्तु को उत्तर प्रदेश बोर्ड ऑफ़ टेक्निकल एजुकेशन के द्वारा प्रतिपादित नवीन "NSQF" आधारित संशोधित पाठ्यक्रमानुसार इस प्रकार प्रस्तुत किया गया है कि एक सामान्य विद्यार्थी भी बिना किस तनाव के समझ कर याद कर सकता है। इस पुस्तक के प्रत्येक पाठ के अंत में अभ्यास प्रश्नों को एलंगित किया गया है जिसकी मदद से विद्यार्थी पाठ का संशोधन और अभ्यास कर सकते हैं।

लेखक अपने सभी सहयोगियों, शुभचिन्तकों, दोस्तों और साथी अध्यापकों का हार्दिक धन्यवाद प्रस्तुत करते हैं जिन्होंने इस पुस्तक को लिखने के दौरान सतत मदद और सहयोग प्रदान किया। लेखक अपने प्रकाशक, मेसर्स एशियन पब्लिशर्स, मुज़्ज़फ़रनगर (उत्तर प्रदेश) का भी दिल की गहराई से धन्यवाद प्रस्तुत करते हैं जिन्होंने लेखकों के ऊपर विश्वास किया है।

अन्त में, लेखक उन विद्यार्थियों या अध्यापकों की सराहना करते हैं जो भविष्य में इस पुस्तक में सुधार के लिए अपने विचार, आलोचनाएं, टिप्पणी और समीक्षा भेजेंगे।

–लेखक

Syllabus

ADVANCED JAVA

LTP
6 – 6

RATIONALE

The diploma holders in Computer Science and Engineering need to understand how server side programming can be done using Java/J2EE Technology They should be able to connect the middle layer to backend and frontend by server side programming Hence this subject is introduced in the curriculum.

SUGGESTED DISTRIBUTION OF MARKS

Topic No.	Time Allotted (Periods)	Marks Allotted (%)
1	10	10
2	14	16
3	16	22
4	14	16
5	16	22
6	14	14
Total 8	4	100

LEARNING OUTCOMES

After undergoing the subject, the students will be able to :

- Understand Server Side Architecture of Web Applications
- Connect to Database and do the CRUD Database operations using JDBC
- Develop Web Application by using Servlets and JSP
- Manage Session in the web application
- Understand Ajax Concept and uses

DETAILED CONTENTS

1. **Introduction Server Side Platform** **(10 Periods)**

Introduction to Web Applications, Dynamic websites, Three Layer Architecture of Web Application , Client Server Architecture, IP Address, Port, URL. Web Server, Introduction to Tomcat Web Server (Installation and its Services), Introduction to J2EE

2. **Database Programming using JDBC** **(14 Periods)**

Introduction to JDBC, JDBC Drivers & Architecture, JDBC API CURD operation Using JDBCAPI : Database Connection, JDBC Statement, Prepared Statements (Advantages and Disadvantages), Using Result Sets

(17 Periods)

3. **Java Servlets**
   Servlet introduction, working of servlet, advantage of servlet, servlet terminology, Servlet Container, Life cycle of a servlet, introduction to servlet API, Servlet interface, Generics Servlet class, Http servlet class, RequestDispatcher (include() and forward).

(14 Periods)

4. **Handling Sessions in Servlets**
   Introductionto Session, Session Tracking mechanism : URL rewriting, Hidden form fields, Cookiesand Http Session (Working, Advantages and Disadvantages of all session tracking mechanism).

(17 Periods)

5. **JSP**
   Introduction to JSP-Architecture, JSP-Life cycle, JSP-syntax, JSP-Directive, JSPActions, JSP-Implicit objects, JSP-Client request, JSP-Server response, JSP integration with database, JSP Session

(12 Periods)

6. **AJAX**
   AJAX Introduction, XMLHttp, Request object, server response, AJAX events, Validation, Interaction with API

LIST OF PRACTICALS

1. Exercises related to make JDBC connections and CRUD operations on database by using JDBC APIs
2. Installation and configuration of Web Server Tomcat
3. Exercises related to Java Servlets
4. Exercises related to JSP
5. Exercises related to AJAX.
6. Exercises related to Session and Cookies.

INSTRUCTIONAL STRATEGY

Since this subject is practice oriented, the teacher should demonstrate the capabilities of server-side programming to students while doing practical exercises. The students should be made familiar with web server and dynamic web site development tools and techniques along with three tier architecture concept.

MEANS OF ASSESSMENT

- Assignments and quiz/class tests, mid-term and end-term written tests
- Actual laboratory and practical work, exercises and viva-voce
- Software installation, operation, development and viva-voce

विषय-सूची

1. **सर्वर साइड प्रोग्रामिंग (SERVER SIDE PROGRAMMING)** 1–31

सर्वर साइड प्रोग्रामिंग (Server Side Programming)

क्लाइंट साइड प्रोग्रामिंग (Client Side Programming)

वेब अनुप्रयोग (Web applications)

वेब सर्वर और क्लाइंट (Web Server and Client)

डायनामिक वेबसाइट (Dynamic Website)

वेब अनुप्रयोग के तीन परत वास्तुकला (Three Layer Architecture of web application)

क्लाइंट सर्वर आर्किटेक्चर (Client Server Architecture)

आईपी अड्रेस (IP Address)

पोर्ट (Port)

URL

वेब सर्वर (Web Server)

टॉमकैट वेब सर्व

J2EE

2. **डाटाबेस प्रोग्रामिंग यूजिंग जेडीबीसी (DATABASE PROGRAMMING USING JDBC)** 32–82

JDBC

JDBC ड्राइवर

JDBC आर्किटेक्चर

JDBC API

JDBC कनेक्शन स्थापित करना (Establishing JDBC Connection)

JDBC API का उपयोग करके CURD ऑपरेशन (CURD operation Using JDBC API)

JDBC स्टेटमेंट (JDBC Statement)

RESULTSET

JDBC डेटा प्रकार

JDBC-एक्सेप्शन हैंडलिंग (JDBC-Exceptions Handling)

JDBC एप्लिकेशन बनाना

JDBC में ट्रेन्ज़ेक्शन Management

जेडीबीसी में बैच प्रोसेसिंग (Batch Processing in JDBC)
JDBC रोसेट (JDBC RowSet) 83–131

3. **जावा सर्वलेटस (JAVA SERVLETS)**
परिचय (Introduction)
सर्वलेट शब्दावली (Servlet Terminology)
एक सर्वलेट का जीवन चक्र (Life cycle of a servlet)
सर्वलेट एपीआई (Servlet API)
सर्वलेट इंटरफ़ेस (Servlet interface)
जेनरिक सर्वलेट क्लास (Generics Servlet class)
Http सर्वलेट क्लास
RequestDispatcher
सर्वलेट बनाना (Creating Servlet)
WAR फ़ाइल
web.xml ceW welcome-file-list
सर्वलेट फिल्टर (Servlet Filter) 132–167

4. **हैंडलीग सेशन (HANDLING SESSIONS)**
परिचय (Introduction)
सेशन ट्रैकिंग मैकेनिज्म (Session Tracking Mechanism) 168–238

5. **जेएसपी (JSP)**
परिचय (Introduction)
JSP एपीआई (JSP API)
Tomcat सर्वर के साथ Eclipse IDE में JSP बनाना
सिंटैक्स (Syntax)
JSP स्क्रिप्टिंग तत्व
JSP-डायरेक्टिव (JSP-Directive)
JSP-Implicit ऑब्जेक्ट्स
JSP -सर्वर response (JSP-Server response)
JSP integration डेटाबेस के साथ

JSP सेशन
JSP में exception हैंडलिंग
JSTL (JSP स्टैंडर्ड टैग लाइब्रेरी)

6. **AJAX** 239–291
परिचय (Introduction)
XMLHttp रिक्वेस्ट ऑब्जेक्ट (XMLHttp Request object)
XMLHttp रिक्वेस्ट ऑब्जेक्ट (XMLHttp Request object)
AJAX कैसे काम करता है?
प्राइमफेस AJAX
AJAX events
Ajax Listener
AJAX S क्शनलिस्ट
Ajax Validation
jQuerry Deewj AJAX(jQuerry and AJAX)

7. **प्रैक्टिकल (PRACTICALS)** 292–317

- **प्रश्न-पत्र** 318–320

1

सर्वर साइड प्रोग्रामिंग
SERVER SIDE PROGRAMMING

LEARNING OBJECTIVES :

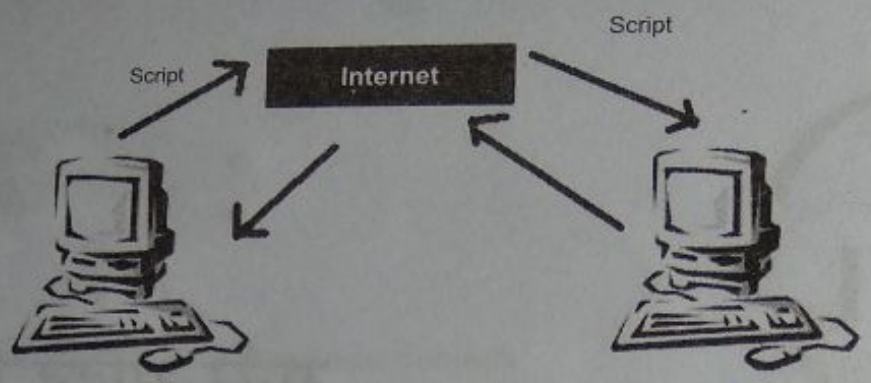
After reading this chapter, you would be able to understand :

- सर्वर साइड प्रोग्रामिंग (Server Side Programming)
- क्लाइंट साइड प्रोग्रामिंग (Client Side Programming)
- वेब अनुप्रयोग (Web applications)
- वेब सर्वर और क्लाइंट (Web Server and Client)
- डायनामिक वेबसाइट (Dynamic Website)
- वेब अनुप्रयोग के तीन परत वास्तुकला (Three Layer Architecture of web application)
- क्लाइंट सर्वर आर्किटेक्चर (Client Server Architecture)
- आईपी अड्रेस (IP Address)
- पोर्ट (Port)
- URL
- वेब सर्वर (Web Server)
- टॉमकैट वेब सर्वर
- J2EE

1.1 सर्वर साइड प्रोग्रामिंग (Server Side Programming)

सभी प्रकार के PROGRAMS सर्वर साइड प्रोग्रामिंग है जो वेब सर्वर पर चलते हैं। वे user इनपुट पर प्रक्रिया करते हैं, डेटाबेस के साथ इंटरैक्ट करते हैं और अपने request के जवाब के रूप में क्लाइंट को क्या कंटेन्ट वापस दिया जाता है, इसे नियंत्रित करते हैं। यह PHP, NodeJS, पायथन, आदि सहित कई प्रोग्रामिंग भाषाओं में लिखा गया है और इसकी पूरी पहुंच सर्वर के OS तक है और प्रोग्रामर उस भाषा को चुन सकता है जिस में कोड करना चाहता है।

सर्वर-साइड प्रोग्रामिंग बेहद उपयोगी है क्योंकि यह user-oriented कंटेन्ट को कुशलतापूर्वक वितरित करने में मदद करता है, और इस प्रकार user अनुभव को बढ़ाता है। इसका उपयोग user के डेटा (डेटा विश्लेषण) के आधार पर प्रतिक्रियाओं को परिष्कृत करने के लिए भी किया जा सकता है।



Server-Saide Programming

सर्वर साइड प्रोग्रामिंग की विशिष्टता:

यह प्रोग्राम है जो वेब पेज की कंटेन्ट के GENRATION के साथ काम करने वाले सर्वर पर चलता है।

(1) डेटाबेस को क्वेरी करना।

(2) डेटाबेस पर संचालन।

(3) सर्वर पर किसी फ़ाइल को एक्सेस करना ।

(4) अन्य सर्वर के साथ बातचीत।

(5) संरचना वेब एप्लीकेशन

(6) प्रोसेस यूजर इनपुट। उदाहरण के लिए यदि user इनपुट खोज बॉक्स में एक पाठ है, तो सर्वर पर संग्रहीत डेटा पर एक खोज एल्गोरिथम चलों और परिणाम भेजें।

सर्वर-साइड प्रोग्रामिंग के लिए प्रोग्रामिंग भाषों हैं—

(1) PHP

(2) C ++

(3) जावा और जेएसपी

(4) PYTHON

(5) रूबी

1.2 क्लाइंट साइड प्रोग्रामिंग (Client Side Programming)

क्लाइंट साइड प्रोग्रामिंग क्लाइंट के डिवाइस (ब्राउज़र) पर चलने वाले कोड को दिया जाने वाला सामान्य नाम है। यह आपके वेबपृष्ठ के व्यवहार और उपस्थिति को नियंत्रित करता है और इसका उपयोग इंटरैक्टिव वेब पेज बनाने, सर्वर को request भेजने और इससे डेटा प्राप्त करने के साथ-साथ वेब-पेज पर डायनामिक रूप से सामान (stors) बनाने के लिए किया जाता है। यह HTML, CSS और जावास्क्रिप्ट का उपयोग करके लिखा गया है।

दूसरे शब्दों में, यह वह प्रोग्राम है जो क्लाइंट मशीन (ब्राउज़र) पर चलता है और यूजर इंटरफेस / डिस्प्ले और किसी भी अन्य प्रोसेसिंग से संबंधित है जो क्लाइंट मशीन पर हो सकता है जैसे पढ़ना / लिखना।

क्लाइंट साइड प्रोग्रामिंग की विशिष्टता—

(1) अस्थायी संग्रहण (temperory storage) के साथ बातचीत।

(2) इंटरैक्टिव वेब पेज बनों।

(3) स्थानीय भंडारण (local storage) के साथ बातचीत करें।

(4) सर्वर को डेटा के लिए request भेजना।

(5) सर्वर के लिए request भेजें।

(6) सर्वर और user के बीच इंटरफेस के रूप में काम करें।

क्लाइंट-साइड प्रोग्रामिंग के लिए प्रोग्रामिंग भाषों हैं—

(1) जावास्क्रिप्ट

(2) VBScript

(3) HTML

(4) CSS

(5) AJAX

1.3 वेब Application (Web applications)

एक वेब एप्लीकेशन, वेब या एप्लीकेशन सर्वर का एक डायनामिक विस्तार है। वेब application प्रकृति द्वारा वितरित application हैं। इसका मतलब है कि कोई भी प्रोग्राम जो एक से अधिक कंप्यूटर पर चलता है और नेटवर्क और सर्वर का उपयोग करके संचार करता है। वेब एप्लीकेशन वेब ब्राउज़र का उपयोग करके एक्सेस किए जाते हैं, इसलिए वे user क्लाइंट के रूप में ब्राउज़र का उपयोग करने में आसानी के लिए बहुत लोकप्रिय हैं। हजारों क्लाइंट कंप्यूटरों पर किसी भी सॉफ्टवेयर को इंस्टॉल किए बिना वेब एप्लीकेशन को अपडेट करने और बनाए रखने की क्षमता मांग का एक महत्वपूर्ण कारण बन जाती है।

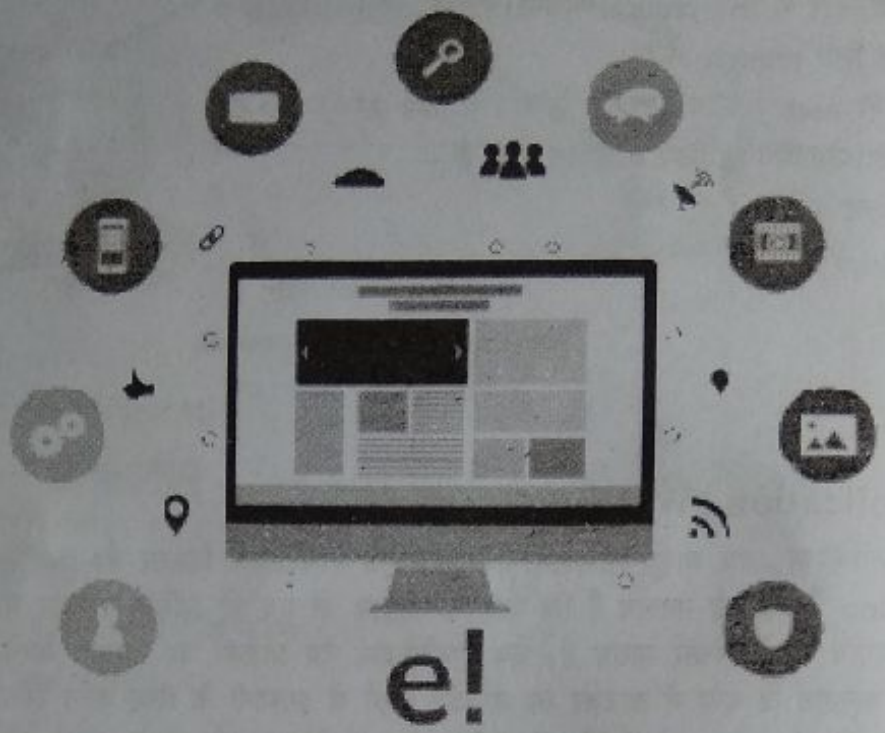
कई घटकों का उपयोग करके वेब एप्लीकेशन बनाए जाते हैं, जिनमें से कुछ में यूजर इंटरफेस होता है और जिनमें से कुछ में Graphical User Interface (GUI) की आवश्यकता नहीं होती है। इसके अलावा, वेब एप्लीकेशन को अक्सर एक अतिरिक्त मार्कअप या स्क्रिप्टिंग भाषा की आवश्यकता होती है। (HTML, CSS, या जावास्क्रिप्ट प्रोग्रामिंग भाषा)। कई एप्लीकेशन केवल जावा प्रोग्रामिंग भाषा का उपयोग करते हैं, जो इसकी बहुमुखी प्रतिभा के कारण आदर्श है।

वेब एप्लीकेशन दो प्रकार के होते हैं—

- **Presentation-oriented :** requests के जवाब में एक presentation-oriented web application विभिन्न प्रकार के मार्कअप भाषा (HTML, XML) और डायनामिक कंटेन्ट वाले इंटरैक्टिव वेब पेज बनाता है।
- **Service-oriented :** एक service oriented web application एक वेब सेवा के समापन बिंदु को लागू करता है। Presentuation-Oreintee application अक्सर सेवा-उन्मुख वेब applications के क्लाइंट होते हैं।

वेब एप्लीकेशन एक साधारण पृष्ठ हो सकता है जो वर्तमान दिनांक और समय या उन पृष्ठों का एक जटिल सेट दिखाता है जिस पर आप अपनी अगली छुट्टी के लिए सबसे सुविधाजनक उड़ान, होटल और कार किराए पर बुक कर सकते हैं।

वेब एप्लीकेशन बनाने के लिए उपयोग की जाने वाली जावा प्रौद्योगिकियाँ Jawa EE प्लेटफॉर्म का हिस्सा हैं। इन तकनीकों के लिए एक सर्वर पर काम करने के लिए, सर्वर में एक कंटेनर या वेब सर्वर होना चाहिए, जो आपके द्वारा बनाई गई classes को पहचानता है और चलाता है।



1.4 वेब सर्वर और क्लाइंट (Web Server and Client)

वेब सर्वर एक सॉफ्टवेयर है जो क्लाइंट request को संसाधित कर सकता है और क्लाइंट को प्रतिक्रिया भेज सकता है। उदाहरण के लिए, apache सबसे अधिक उपयोग किए जाने वाले वेब सर्वरों में से एक है। वेब सर्वर कुछ भौतिक मशीन पर चलता है और एक विशिष्ट पोर्ट पर क्लाइंट के request को सुनता है।

एक वेब क्लाइंट एक सॉफ्टवेयर है जो सर्वर के साथ संचार करने में मदद करता है। सबसे व्यापक रूप से उपयोग किए जाने वाले वेब क्लाइंट में से कुछ फ़ायरफ़ॉक्स, उददुत क्रोम, सफारी, आदि हैं जब हम सर्वर से कुछ request करते हैं (URL के माध्यम से), वेब क्लाइंट एक request बनाने और इसे सर्वर पर भेजने और फिर सर्वर response को पार्स करने का ख्याल रखता है।

1.5 डायनामिक वेबसाइट (Dynamic Website)

एक डायनामिक वेबसाइट या डायनामिक वेब पेज में वह जानकारी होती है जो दर्शक, दिन के समय, समय field, दर्शक की मूल भाषा और अन्य कारकों के आधार पर बदलती है।

एक डायनामिक वेबसाइट में क्लाइंट-साइड स्क्रिप्टिंग या सर्वर-साइड स्क्रिप्टिंग कंटेन्ट बदलने या स्क्रिप्टिंग दोनों प्रकारों के संयोजन को उत्पन्न करने के लिए प्रयोग हो सकता है । इन साइटों में मूल संरचना के लिए HTML प्रोग्रामिंग भी शामिल है। क्लाइंट-साइड या सर्वर-साइड स्क्रिप्टिंग साइट की हिम्मत का ख्याल रखती है। क्लाइंट-साइड HTML स्क्रिप्टिंग के साथ, पेज का उपयोग कर सकते हैं ,जावास्क्रिप्ट पृष्ठ के डेटा को बदलने के लिए एक अन्य स्क्रिप्टिंग भाषा, है।

सर्वर-साइड स्क्रिप्टिंग में स्क्रिप्ट सर्वर पर चलाए जाते हैं, जो उस पृष्ठ को होस्ट करता है। पृष्ठ कैसे बनाया जाता है इसके लिए प्रक्रिया सर्वर-साइड स्क्रिप्टिंम में परिभाषित parameters द्वारा निर्धारित की जाती है।

आप स्टेटिक पृष्ठ पर एक डायनामिक पृष्ठ क्यों चुनेंगे?

निम्नलिखित कई वजहों में से एक कंपनी को एक स्टेटिक पृष्ठ पर एक डायनामिक पृष्ठ की आवश्यकता हो सकती है।

1. एक डेटाबेस या बाहरी फ़ाइल का उपयोग करने की आवश्यकता है

ऐसा पृष्ठ जिसे डेटाबेस एक्सेस या जानकारी प्राप्त करने के लिए बाहरी फ़ाइल को डायनामिक होना चाहिए। उदाहरण के लिए, जब आप यात्रा करते हैं और उददुत पर एक खोज करें, उददुत आपकी खोज क्वेरी को सैकड़ों कंप्यूटरों पर भेजता है और उन कंप्यूटरों से सभी जानकारी को खोज परिणाम पृष्ठ पर जोड़ता है।

2. जानकारी अक्सर update की जाती है

वे वेबसाइटें जिनकी जानकारी अक्सर अपडेट होती है, वे ऑनलाइन जानकारी प्राप्त करना आसान और तेज़ बनाने के लिए डायनामिक होती हैं। उदाहरण के लिए, एक समाचार साइट में कई अलग-अलग पत्रकार हो सकते हैं जो कहानियां प्रस्तुत कर सकते हैं। स्क्रिप्ट के साथ, कंपनी अपने आप को होम पेज को अपडेट करने के लिए उन कहानियों को शामिल करने के लिए किसी पर निर्भर हो सकती है जो किसी स्टेटिक HTML पेज को संपादित करने के लिए हर बार एक नई कहानी और पेज जोड़ते हैं।

3. डायनामिक साइटें सभी के लिए योगदान देना आसान बनाती हैं

वर्डप्रेस साइटों और अन्य CMS समाधान HTML या प्रोग्रामिंग के बारे में बहुत कुछ जाने बिना किसी के लिए भी वेबसाइट बनाना आसान है। एक बार बन जाने के बाद, user ऑनलाइन संपादक का उपयोग करके पेज बना सकते हैं और उन्हें CMS का उपयोग करके ऑनलाइन पोस्ट कर सकते हैं। इसके विपरीत, एक स्टेटिक वेबसाइट के लिए आवश्यक होगा कि user पृष्ठ को HTML में बनाए।

स्टेटिक वेब पेज और डायनामिक वेब पेज के बीच महत्वपूर्ण अंतर हैं।

अनु क्रमांक	प्रमुख Field	स्टेटिक वेब पेज	डायनामिक वेब पेज
1	परिभाषा	स्टेटिक वेब पेज आम तौर पर सरल प्ऊश्थ लिखित पृष्ठ होते हैं जो ब्राउज़र से सर्वर तक प्रतिक्रिया के रूप में कार्य करते हैं जिसमें सभी जानकारी और डेटा प्रकृति में स्टेटिक होते हैं और यह तब तक परिवर्तित नहीं होता है जब तक कि कोई इसे मैन्युअल रूप से नहीं बदलता है।	दूसरी ओर डायनेमिक वेबपेज कुछ अधिक जटिल भाषा में लिखे गए पेज हैं जैसे कि ASP.NET जिसमें विभिन्न कॉल के लिए विशिष्ट कंटेन्ट का उत्पादन करने के लिए कुछ व्याख्या और क्षमता के बाद डेटा प्रदान किया जाता है।
2	जटिलता	जैसा कि उपर्युक्त बिंदु में बताया गया है कि स्टेटिक बेब पेजों में डेटा स्टेटिक है और स्टेटिक वेब पेज जटिलता में सरल हैं, रेंडर करने से पहले किसी भी व्याख्या की आवश्यकता नहीं है।	दूसरी ओर डायनामिक वेब पेज व्याख्या की प्रक्रिया करते हैं जो डेटा को डायनामिक बनाते हैं और जिसके कारण डायनेमिक वेब पेज स्टैटिक वेब पेज की तुलना में जटिलता में जटिल हो जाते हैं।

3	भाषा का प्रयोग किया	स्टेटिक वेब पेज आमतौर पर HTML, जावास्क्रिप्ट, सीएसएस, आदि जैसी सरल भाषाओं में लिखे जाते हैं।	अन्य डायनामिक वेब पेजों पर और अधिक जटिल भाषाओं जैसे कि CGI, AJAX, ASP, ASP.NET आदि लिखे जाते हैं।
4	रेंडर किया गया	डेटा स्टेटिक वेब पेजों के लिए डेटा तब तक नहीं बदलता है जब तक कि कोई इसे मैन्युअल रूप से नहीं बदलता है और इसलिए डेटा स्टेटिक है।	दूसरी ओर डायनामिक वेब पेज डेटा के लिए पहले सर्वर की ओर से इंटरप्रेट किया जाता है और इसके कारण यह हर कॉल पर एक जैसा नहीं रहता है और इससे डेटा डायनामिक हो जाता है।
5	समय	स्टेटिक डेटा के कारण स्टेटिक वेब पेज लोड होने में कम समय लेते हैं। जबकि डायनामिक डेटा के कारण	डायनामिक वेब पेज स्टेटिक वेब पेजों की तुलना में अपेक्षाकृत अधिक समय लेते हैं।
6	डेटाबेस	स्टेटिक वेब पेजों में आम तौर पर डेटा पुनर्विकास के लिए डेटाबेस की कोई भागीदारी नहीं होती है।	दूसरी ओर डायनामिक वेब पेज डेटाबेस के मामले में डेटा पुनर्विकास के लिए उपयोग किया जाता है।

1.6 वेब application के तीन परत आर्किटेक्चर (Three Layer Architecture of web application)

3-स्तरीय आर्किटेक्चर एक आर्किटेक्चर है जो तार्किक कंप्यूटिंग के तीन 'स्तरों' या 'परतों' से बना है। उन्हें अक्सर एक विशिष्ट प्रकार के क्लाइंट-सर्वर सिस्टम के रूप में अनुप्रयोगों में उपयोग किया जाता है। 3-स्तरीय आर्किटेक्चर user इंटरफ़ेस, व्यावसायिक तर्क और डेटा संग्रहण परतों को संशोधित करके उत्पादन और विकास के वातावरण के लिए कई लाभ प्रदान करते हैं। ऐसा करने से विकास टीमों को अधिक लचीलेपन की सुविधा मिलती है, जिससे वे स्वतंत्र रूप से अन्य भागों के एक विशेष हिस्से को अपडेट कर सकते हैं। यह लचीलेपन समग्र समय-समय पर बाजार में सुधार कर सकता है और विकास टीमों को सिस्टम के अन्य भागों को प्रभावित किए बिना स्वतंत्र स्तरों को बदलने या अपग्रेड करने की क्षमता देकर विकास चक्र समय को कम कर सकता है।

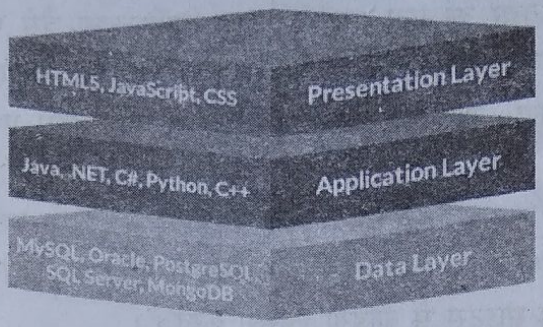
उदाहरण के लिए, एक वेब एप्लीकेशन का यूजर इंटरफेस अंतर्निहित कार्यात्मक व्यापार और डेटा एक्सेस लॉजिक को प्रभावित किए बिना पुनर्विकास या आधुनिकीकरण किया जा सकता है। यह आर्किटेक्चर सिस्टम अक्सर किसी मौजूदा एप्लीकेशन में 3rd पार्टी सॉफ्टवेयर को एम्बेड और एकीकृत करने के लिए आदर्श है।

3 स्तरों से क्या मतलब है?

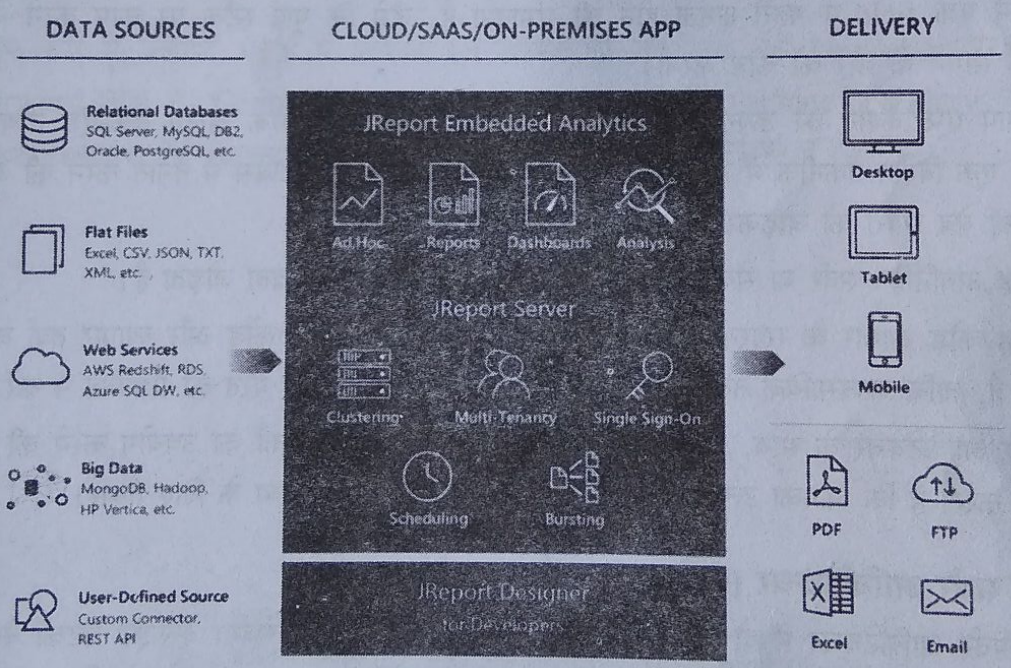
इसमें 3 परतें शामिल हैं—

- **प्रस्तुति टियर (Presentation Tier) :** 3-टियर सिस्टम में प्रेजेंटेशन टियर फ्रंट एंड लेयर है और इसमें यूजर इंटरफेस होता है। यह यूजर इंटरफेस अक्सर एक ग्राफिकल होता है जो वेब ब्राउज़र या वेब-आधारित एप्लीकेशन के माध्यम से सुलभ होता है और जो अंत user के लिए उपयोगी कंटेन्ट और सूचना प्रदर्शित करता है। यह स्तरीय अक्सर HTML5, जावास्क्रिप्ट, सीएसएस, या अन्य लोकप्रिय वेब विकास रूपरेखाओं

के माध्यम से वेब प्रौद्योगिकियों पर बनाया गया है, और एपीआई कॉल के माध्यम से अन्य परतों के साथ संचार करता है।



- **एप्लीकेशन टियर (Application Tier) :** एप्लीकेशन टियर में कार्यात्मक व्यावसायिक तर्क होता है जो एप्लीकेशन की मुख्य क्षमताओं को चलाता है। यह अक्सर जावा, .NET, C #, पायथन, C ++ आदि में लिखा जाता है।
- **डेटा टियर (Data Tier) :** डेटा टियर में डेटाबेस / डेटा स्टोरेज सिस्टम और डेटा एक्सेस लेयर शामिल हैं। ऐसे सिस्टम के उदाहरण MySQL, Oracle, PostgreSQL, Microsoft SQL Server, MongoDB, आदि हैं। डेटा को API कॉल के माध्यम से एप्लीकेशन लेयर द्वारा एक्सेस किया जाता है।



3-स्तरीय आर्किटेक्चर का उदाहरणः लोगी रिपोर्ट।

3-स्तरीय आर्किटेक्चर परिनियोजन के लिए विशिष्ट संरचना में डेस्कटॉप, लैपटॉप, टैबलेट या मोबाइल डिवाइस के लिए प्रस्तुत किया गया टियर एक वेब ब्राउज़र या वेब सर्वर का उपयोग करने वाला वेब-आधारित एप्लीकेशन होगा। अंतर्निहित एप्लीकेशन टियर को आमतौर पर एक या अधिक एप्लीकेशन सर्वर पर होस्ट किया जाता है, लेकिन इसे क्लाउड में भी होस्ट किया जा सकता है, या एप्लीकेशन द्वारा आवश्यक जटिलता और प्रसंस्करण शक्ति के आधार पर

एक समर्पित कार्य केंद्र पर। और डेटा लेयर में आमतौर पर एक या अधिक रिलेशनल डेटाबेस, बड़े डेटा स्रोत या अन्य प्रकार के डेटाबेस सिस्टम शामिल होते हैं जो या तो ऑन-प्रिमाइसेस या क्लाउड में होस्ट किए जाते हैं।

एक्शन में 3-स्तरीय आर्किटेक्चर का एक सरल उदाहरण नेटफिलक्स जैसे मीडिया अकाउंट में लॉग इन करना और वीडियो देखना होगा। आप वेब के माध्यम से या मोबाइल एप्लीकेशन के माध्यम से लॉग इन करके शुरू करते हैं। आपके द्वारा लॉग इन करने के बाद, आप नेटफिलक्स इंटरफ़ेस के माध्यम से एक विशिष्ट वीडियो का उपयोग कर सकते हैं, जो आपके द्वारा अंतिम user के रूप में उपयोग की जाने वाली प्रस्तुति स्तरीय है। एक बार जब आप एक वीडियो का चयन कर लेते हैं, तो जानकारी को एप्लीकेशन टियर पर भेज दिया जाता है, जो सूचना को कॉल करने के लिए डेटा टियर को क्वेरी कर देगा या इस स्थिति में एक वीडियो प्रेजेंटेशन टियर तक वापस आ जाएगा। ऐसा तब होता है जब आप ज्यादातर मीडिया साइट्स से वीडियो एक्सेस करते हैं।

3-लेयर आर्किटेक्चर का उपयोग करने के क्या लाभ हैं?

यहाँ एक एप्लीकेशन को स्तरों में अलग करने के 5 लाभ दिए गए हैं:

(1) यह आपको एप्लीकेशन के अन्य क्षेत्रों को प्रभावित किए बिना, एक स्तरीय तकनीक को अपडेट करने की क्षमता देता है।

(2) यह अलग-अलग विकास टीमों को अपनी विशेषज्ञता के क्षेत्रों में प्रत्येक कार्य के लिए अनुमति देता है। आज के डेवलपर्स में एक field में गहरी क्षमता होने की संभावना है, जैसे कि पूर्ण स्टैक पर काम करने के बजाय किसी एप्लीकेशन के सामने के छोर को कोड करना।

(3) आप एप्लीकेशन को ऊपर और बाहर स्केल करने में सक्षम हैं। एक अलग बैक-एंड टीयर, उदाहरण के लिए, आपको एक विशेष तकनीक में लॉक होने के बजाय विभिन्न प्रकार के डेटाबेस में तैनात करने की अनुमति देता है। यह आपको कई वेब सर्वरों को जोड़कर स्केल करने की अनुमति देता है।

(4) यह अंतर्निहित सर्वर या सेवाओं की विश्वसनीयता और अधिक स्वतंत्रता जोड़ता है।

(5) यह कोड आधार के रखरखाव में आसानी प्रदान करता है, प्रस्तुति कोड और व्यापार तर्क को अलग-अलग प्रबंधित करता है, ताकि व्यावसायिक तर्क में बदलाव, उदाहरण के लिए, प्रस्तुति परत को प्रभावित न करे।

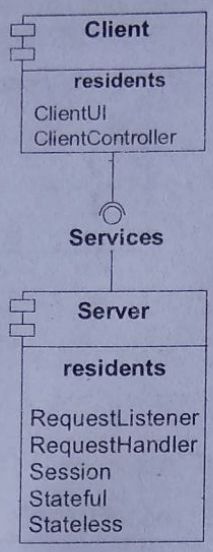
3-स्तरीय आर्किटेक्चर के साथ, आपके पास उपलब्ध होते ही नई तकनीकों का उपयोग करने की क्षमता होती है। यह सुनिश्चित करता है कि आपका उत्पाद अनुकूल होने के लिए तैयार है; भविष्य के लिए तैयार।

1.7 क्लाइंट सर्वर आर्किटेक्चर (Client Server Architecture)

क्लाइंट-सर्वर आर्किटेक्चर में दो प्रकार के घटक होते हैं: क्लाइंट और सर्वर। क्लाइंट घटकों के requests को सर्वर घटक हमेशा सुनता है। जब कोई request प्राप्त होता है, तो सर्वर request को संसाधित करता है, और फिर क्लाइंट को एक प्रतिक्रिया भेजता है। सर्वर को आगे स्टेटलेस या स्टेटफुल के रूप में वर्गीकृत किया जा सकता है। एक स्टेटफुल सर्वर के क्लाइंट composite request कर सकते हैं जिसमें कई atomic request शामिल हैं। यह क्लाइंट और सर्वर के बीच अधिक संवादात्मक या लेन-देन को सक्षम करता है। इसे पूरा करने के लिए, एक स्टेटफुल सर्वर प्रत्येक वर्तमान क्लाइंट के requests का रिकॉर्ड रखता है। इस रिकॉर्ड को एक सेशन कहा जाता है।

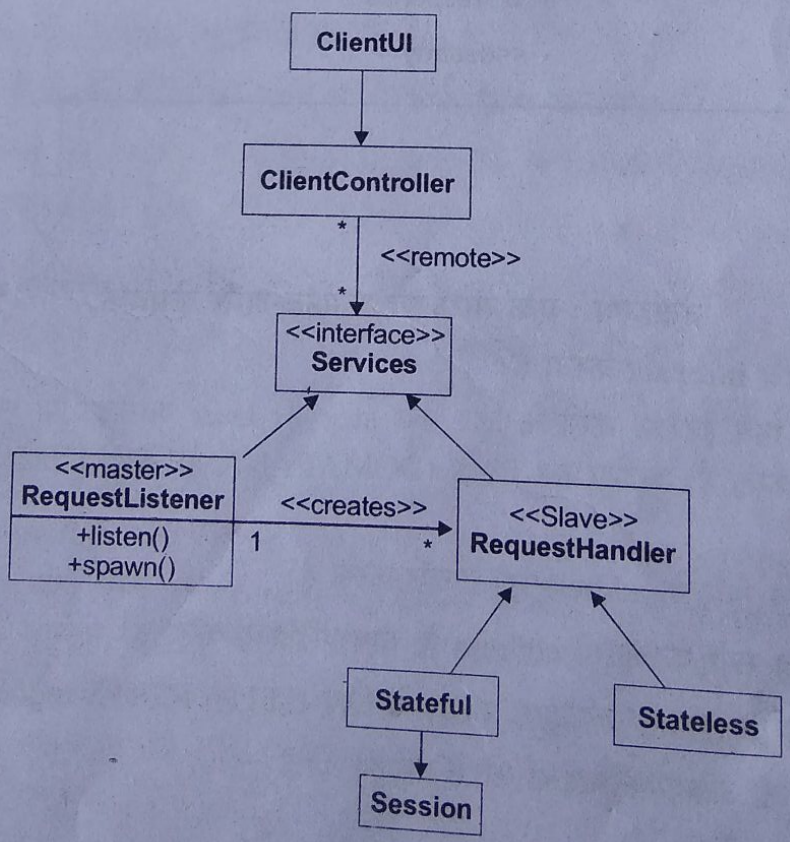
एक साथ कई क्लाइंट से रिक्वेस्ट प्रोसेस करने के लिए, एक सर्वर अक्सर मास्टर-स्लेव पैटर्न का उपयोग करता है। इस मामले में मास्टर क्लाइंट requests के लिए नियमित रूप से सुनता है। जब एक request प्राप्त होता है, तो

मास्टर request को संसाधित करने के लिए एक स्लेक बनाता है, और फिर सुनना शुरू करता है। इस बीच, स्लेक क्लाइंट के साथ सभी बाद के संचार करता है।

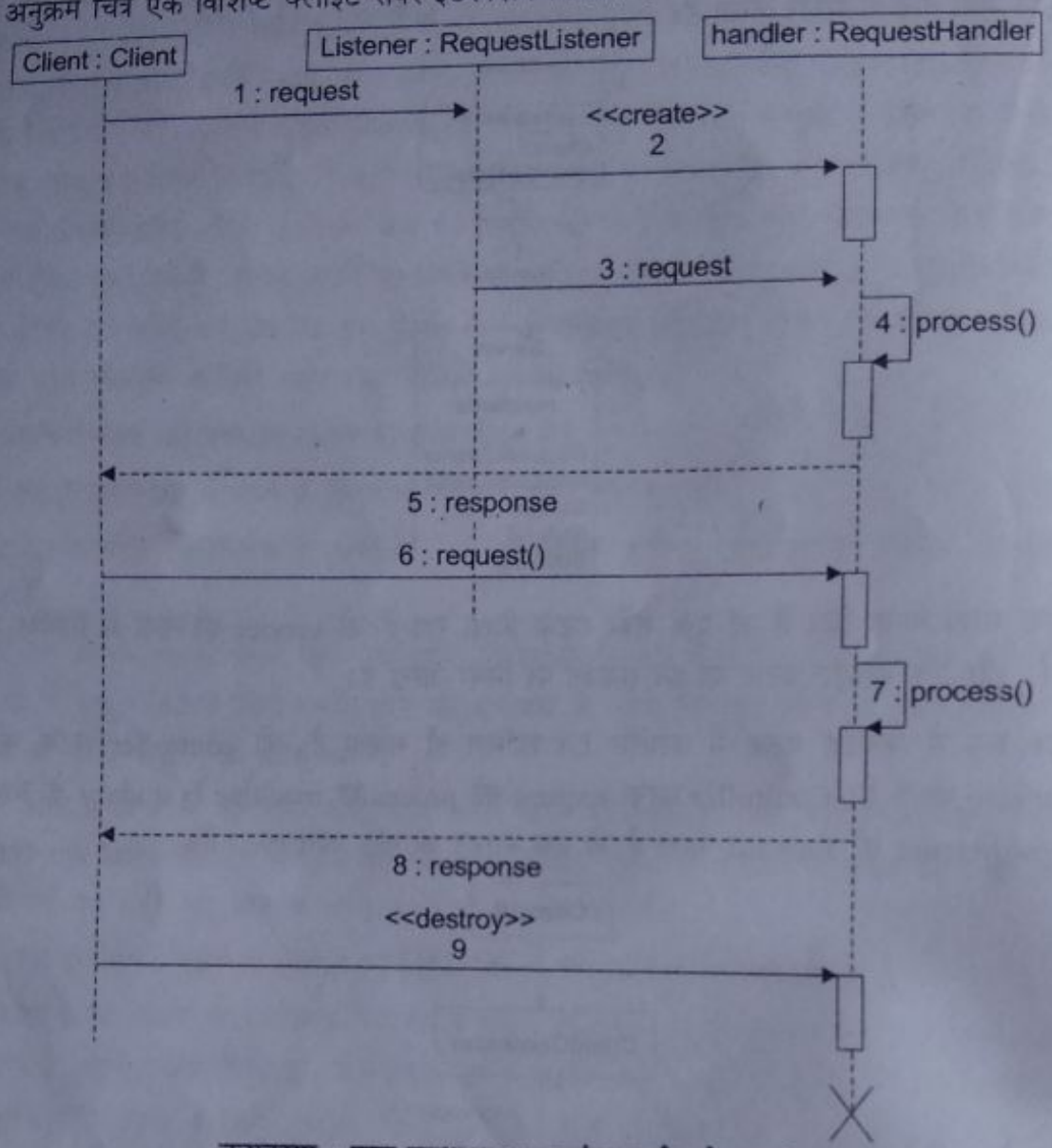


यहां एक सरल घटक चित्र है जो एक सर्वर घटक दिखा रहा है जो service इंटरफ़ेस में निर्दिष्ट संचालन को लागू करता है, और एक क्लाइंट घटक जो इन सेवाओं पर निर्भर करता है।

आंतरिक रूप से क्लाइंट घटक में क्लाइंट UI शामिल हो सकता है, जो controller घटक को user कि request forward करता है। Controller घटक request को proces या machine boundary के बहार, सर्वर के अन्दर request listener को forward करता है जो एक मास्टर की तरह काम करता है।



निम्न अनुक्रम चित्र एक विशिष्ट क्लाइंट-सर्वर इंटरैक्शन दिखाता है—



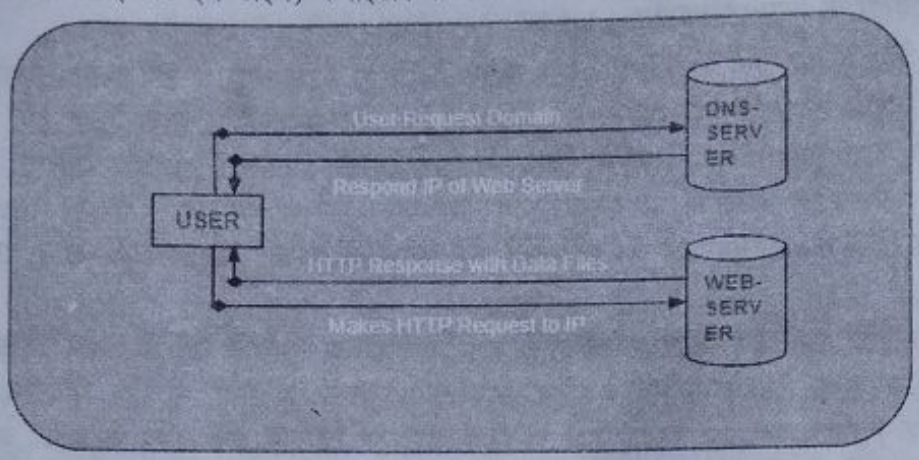
उदाहरण : एक साधारण क्लाइंट-सर्वर फ्रेमवर्क

कैसे ब्राउज़र सर्वर के साथ interact करता है?

सर्वर के क्लाइंट के साथ इंटरैक्ट करने के लिए कुछ step हैं। User वेबसाइट या फ़ाइल के URL (यूनिफ़ॉर्म रिसोर्स लोकेटर) में प्रवेश करता है। ब्राउज़र तब DNS (DOMAIN NAME SYSTEM) सर्वर का request करता है।

- **DNS सर्वर** वेब सर्वर के address की तलाश करता है।
- **DNS सर्वर** वेब सर्वर के आईपी address के साथ प्रतिक्रिया करता है।
- WEB सर्वर के IP (DNS सर्वर द्वारा प्रदान) के लिए HTTP / HTTPS request पर ब्राउज़र भेजता है।
- सर्वर वेबसाइट की आवश्यक फाइलों को भेजता है।

- ब्राउज़र तब फाइलों को प्रस्तुत करता है और वेबसाइट प्रदर्शित की जाती है। यह रेंडरिंग DOM (डॉक्यूमेंट ऑब्जेक्ट मॉडल) इंटरप्रेटर, CSS इंटरप्रेटर और JS इंजन की सहायता से किया जाता है जिसे सामूहिक रूप से JIT या (जस्ट इन टाइम) कंपाइलर के रूप में जाना जाता है।



लाभः

- एक ही स्थान पर सभी डेटा के साथ centralized system है।
- Less mainteince और डेटा रिकवरी संभव है।
- क्लाइंट और सर्वर की क्षमता को अलग-अलग बदला जा सकता है।

नुकसानः

- क्लाइंट वायरस के लिए Prone है। (अगर ट्रोजन और वर्म सर्वर में मौजूद है या सर्वर पर उप्लोडेड है)।
- Denial of Service (DOS) हमलों के लिए Prone हैं।
- ट्रांसमिशन के दौरान डेटा पैकेट को खराब या संशोधित किया जा सकता है।
- लॉगिन क्रेडेंशियल की फिशिंग या कैप्चरिंग या user की अन्य उपयोगी जानकारी आम है और MITM (Man in the Middle) हमले आम हैं।

1.8 आईपी अड्रेस (IP Address)

आईपी (इंटरनेट प्रोटोकॉल) (IP (Internet Protocol)) अड्रेस आपके नेटवर्क हार्डवेयर का एक अड्रेस है। यह आपके कंप्यूटर को आपके नेटवर्क और दुनिया भर के अन्य उपकरणों से जोड़ने में मदद करता है। एक IP अड्रेस संख्याओं या वर्णों से बना होता है। IP address का एक उदाहरण होगा— 506.457.14.512।

नेटवर्क की प्रत्येक मशीन में एक विशिष्ट पहचानकर्ता होता है। जैसे आप मेल में भेजने के लिए एक पत्र को संबोधित करेंगे, वैसे ही कंप्यूटर नेटवर्क पर विशिष्ट कंप्यूटरों को डेटा भेजने के लिए अद्वितीय पहचानकर्ता का उपयोग करते हैं। अधिकांश नेटवर्क सभी कंप्यूटरों Internet पर शामिल टीसीपी/आईपी प्रोटोकॉल का उपयोग करते हैं। टीसीपी/आईपी प्रोटोकॉल में, कंप्यूटर के लिए विशिष्ट पहचानकर्ता को उसका आईपी अड्रेस कहा जाता है।

IP Addresses के लिए दो मानक हैं: IP संस्करण 4 (IPv4) और IP address 6 (IPv6)। IP address वाले सभी कंप्यूटरों में IPv4 अड्रेस होता है, और अधिकांश नए IPv6 एड्रेस सिस्टम का भी उपयोग करते हैं। यहाँ दो अड्रेस प्रकारों के बीच अंतर हैं—

- IPv4 नेटवर्क पर एक एकल अद्वितीय अड्रेस बनाने के लिए 32 बाइनरी बिट्स का उपयोग करता है। IPv4 address को डॉट्स द्वारा अलग किए गए चार नंबरों द्वारा व्यक्त किया जाता है। प्रत्येक संख्या आठ-अंकीय बाइनरी (बेस-2) संख्या के लिए दशमलव (बेस-10) प्रतिनिधित्व है, जिसे ऑक्टेट भी कहा जाता है। उदाहरण के लिए: 216.27.61.137
- IPv6 नेटवर्क पर एक अद्वितीय अड्रेस बनाने के लिए 128 बाइनरी बिट्स का उपयोग करता है। एक IPv6 अड्रेस हेक्साडेसिमल (बेस-16) संख्याओं के आठ समूहों द्वारा व्यक्त किया गया है, जो 2001 में कॉलोनों द्वारा अलग किए गए थे, जैसा कि 2001 में : cdba: 0000: 0000: 0000: 0000: 3257: 9652। संख्याओं के समूह जिनमें सभी शून्य होते हैं, वे अक्सर space (memory) को बचाने के लिए हैं, जो अंतराल को चिह्नित करने के लिए एक कॉलन विभाजक को छोड़ देता है (2001 में: cdba :: 3257: 9652)।

IPv4 को संबोधित करते हुए, इंटरनेट आज की बड़ी व्यावसायिक सनसनी नहीं थी, और अधिकांश नेटवर्क निजी थे और दुनिया भर के अन्य नेटवर्क से बंद थे। जब इंटरनेट में विस्फोट हो गया, तो एक अद्वितीय इंटरनेट address की पहचान करने के लिए केवल 32 बिट्स होने से चिंता पैदा हुई कि हम लंबे समय से पहले आईपी address से बाहर निकल जोंगे। IPv4 के तहत, 232 संभावित संयोजन हैं, जो केवल 4.3 बिलियन अद्वितीय पतों के तहत प्रदान करता है। IPv6 ने तनाव बढ़ाने वाले 2,128 संभावित addresses को उठाया। बाद में, हम आपके कंप्यूटर के IPv4 या IPv6 addresses को समझने के तरीके पर ध्यान देंगे।

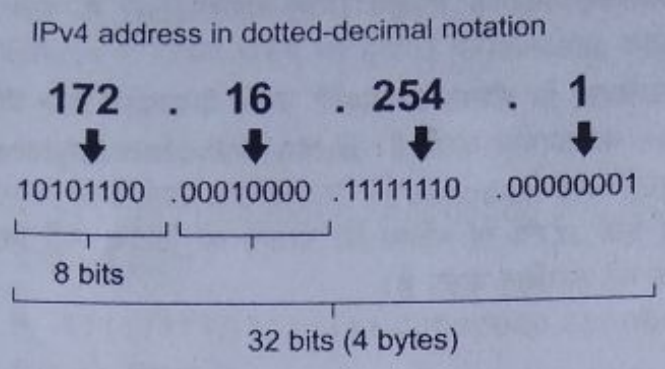
आपके कंप्यूटर को उसका IP अड्रेस कैसे मिलता है? एक आईपी अड्रेस डायनामिक या static हो सकता है। स्टेटिक अड्रेस एक स्थायी रूप से दिया गया अड्रेस है। इंटरनेट सेवा प्रदाताओं द्वारा निर्दिष्ट स्टेटिक आईपी address दुर्लभ हैं। आप अपने स्थानीय नेटवर्क पर उपकरणों के लिए स्टेटिक आईपी असाइन कर सकते हैं, लेकिन यदि आप टीसीपी/आईपी की अच्छी समझ के बिना इसका उपयोग करते हैं तो यह नेटवर्क समस्यों पैदा कर सकता है। डायनेमिक address सबसे आम हैं। वे डायनामिक होस्ट कॉन्फिगरेशन प्रोटोकॉल (DHCP) द्वारा असाइन किए गए हैं, जो नेटवर्क पर चल रही service है। DHCP आमतौर पर नेटवर्क हार्डवेयर पर चलता है जैसे routers या DHCP सर्वर।

डायनामिक आईपी एड्रेस एक leasing सिस्टम का उपयोग करके जारी किया जाता है, जिसका अर्थ है कि आईपी एड्रेस सीमित समय के लिए ही सक्रिय है। यदि lease समाप्त हो जाता है, तो कंप्यूटर स्वचालित रूप से एक नए lease का request करेगा। कभी-कभी, इसका मतलब है कि कंप्यूटर को एक नया आईपी अड्रेस मिलेगा, भी, खासकर अगर कंप्यूटर leases के बीच नेटवर्क से अनप्लग किया गया था। यह प्रक्रिया आमतौर पर user के लिए पारदर्शी होती है जब तक कि कंप्यूटर नेटवर्क पर आईपी address के संघर्ष के बारे में चेतावनी नहीं देता (दो कंप्यूटर एक ही आईपी address के साथ)। एक अड्रेस संघर्ष दुर्लभ है, और आज की तकनीक आमतौर पर समस्या को स्वचालित रूप से ठीक करती है।

अगला, आइए एक आईपी address के महत्वपूर्ण भागों और कुछ addresses की विशेष भूमिकाओं पर करीब से नज़र डालें।

आईपी क्लासेस (IP Classes)

इससे पहले, आप पढ़ चूके हैं कि IPv4 address चार, आठ अंकों के binary संख्या का प्रतिनिधित्व करते हैं। इसका मतलब है कि प्रत्येक संख्या बाइनरी में 00000000 से 11111111 या दशमलव (बेस-10) में 0 से 255 हो सकती है। दूसरे शब्दों में, 0.0.0.0 से 255.255.255.255। हालाँकि, उस श्रेणी के कुछ नंबर टीसीपी/आईपी नेटवर्क पर विशिष्ट उद्देश्यों के लिए आरक्षित हैं।



IP address का एक चित्र (IPv4)

ये आरक्षण टीसीपी/आईपी address पर authority द्वारा मान्यता प्राप्त हैं, Internet Assigned Numers Authority (IANA)। चार विशिष्ट आरक्षणों में निम्नलिखित शामिल हैं:

- **0.0.0.0:** यह डिफ़ॉल्ट नेटवर्क का प्रतिनिधित्व करता है, जो कि TCP/IP नेटवर्क से जुड़े होने की abstract अवधारणा है।
- **255.255.255.255 :** यह अड्रेस नेटवर्क प्रसारण, या संदेश जो नेटवर्क पर सभी कंप्यूटरों के लिए जाना चाहिए के लिए आरक्षित है।
- **127.0.0.1:** इसे लूपबैक एड्रेस कहा जाता है, जिसका अर्थ है कि आपके कंप्यूटर का खुद को पहचानने का तरीका, चाहे उसमें एक निर्धारित IP एड्रेस हो या न हो।
- **169.254.0.1 से 169.254.255.254:** यह Automatic Private IP Addressing (APIPA) श्रेणी का स्वचालित रूप से असाइन किए गए address की सीमा है जब एक कंप्यूटर का असफल डीएचसीपी सर्वर से अड्रेस प्राप्त होता है।

अन्य आईपी अड्रेस आरक्षण सबनेट class के लिए हैं। एक सबनेटवर्क एक रूटर के माध्यम से बड़े नेटवर्क से जुड़े कंप्यूटर का एक छोटा नेटवर्क है। सबनेट का अपना अड्रेस सिस्टम हो सकता है ताकि एक ही सबनेट पर कंप्यूटर बड़े नेटवर्क पर डेटा भेजे बिना जल्दी से संवाद कर सकें। इंटरनेट सहित टीसीपी/आईपी नेटवर्क पर एक राउटर को एक या अधिक सबनेट और रूट नेटवर्क ट्रैफिक को उचित रूप से पहचानने के लिए कॉन्फिगर किया गया है। निम्नलिखित आईपी address सबनेट के लिए आरक्षित हैं :

- **10.0.0.0 से 10.255.255.255 :** यह 1.0.0.0 से 127.0.0.0 के Class A address सीमा के भीतर आता है, जिसमें पहला बिट 0 है।
- **172.16.0.0 से 172.31.255.255:** यह class B अड्रेस सीमा 128.0.0.0 से 191.255.0.0 तक है, जिसमें पहले दो बिट 10 हैं।
- **192.168.0.0 से 192.168.255.255:** यह 222.255.255.0 से 192.0.0.0 के class C के भीतर आता है, जिसमें पहले तीन बिट्स 110 हैं।
- **मल्टीकास्ट (जिसे पहले** class D **कहा जाता था):** address में पहले चार बिट्स 1110 हैं, जिसमें 224.0.0.0 से 239.255.255.255 तक के address हैं।

- भविष्य के लिए आरक्षित/प्रायोगिक उपयोग (जिसे पहले class E कहा जाता है): 240.0.0.0 से 254.255.255.254 address।

पहले तीन (कक्षा ए, बी और सी के भीतर) उन सबनेट बनाने में उपयोग किए जाते हैं। बाद में, हम देखेंगे कि कैसे एक सबनेट इन addresses का उपयोग करता है। IANA ने Internet Engineering Task Force (IETF) दस्तावेज़ RF(577) के भीतर मल्टीकास्ट addresses के लिए विशिष्ट उपयोगों की रूपरेखा तैयार की। हालाँकि, इसने class E के address के लिए एक उद्देश्य या भविष्य की योजना को निर्दिष्ट नहीं किया है क्योंकि यह 1989 के दस्तावेज़ RFC 1112 में ब्लॉक को आरक्षित करता है।

इंटरनेट अड्रेस और सबनेट

- आईपी **अड्रेस**: 192.168.1.102
- सबनेट **मास्क**: 255.255.255.0
- चौबीस बिट्स (तीन **ओकटेट्स**) नेटवर्क पहचान के लिए आरक्षित हैं
- आठ बिट्स (एक **ओकटेट**) नोड्स के लिए आरक्षित है
- सबनेट मास्क (पहला **अड्रेस**) के आधार पर सबनेट पहचान: 192.168.1.0
- सबनेट (अंतिम **अड्रेस**) के लिए आरक्षित प्रसारण पता: 192.168.1.255
- Same नेटवर्क पर उदाहरण address : 192.168.1.1, 192.168.1.103
- उदाहरण address same नेटवर्क पर नहीं: 192.168.2.1, 192.168.2.103

IP addresses को संग्रहीत करने के अलावा, IANA IP addresses के ब्लॉक को कुछ संस्थाओं, आमतौर पर वाणिज्यिक या सरकारी संगठनों को सौंपने के लिए भी जिम्मेदार है। आपका Internet Service Provider (ISP) इन संस्थाओं में से एक हो सकता है, या यह उन संस्थाओं में से एक के नियंत्रण में बड़े ब्लॉक का हिस्सा हो सकता है। आप IPv4 addresses के लिए IANA असाइनमेंट और आरक्षण की पूरी सूची IANA कि Website पर देख सकते हैं।

यदि आप केवल एक कंप्यूटर को इंटरनेट से जोड़ते हैं, तो वह कंप्यूटर आपके ISP के address का उपयोग कर सकता है। कई घरों में, हालांकि, कई कंप्यूटरों के बीच एकल इंटरनेट कनेक्शन साझा करने के लिए राउटर का उपयोग किया जाता है।

यदि आप इंटरनेट कनेक्शन साझा करने के लिए राउटर का उपयोग करते हैं, तो राउटर को ISP से सीधे जारी आईपी एड्रेस मिलता है। फिर, यह उस राउटर से जुड़े सभी कंप्यूटरों के लिए एक सबनेट बनाता है जिसे उसका प्रबंधन करता है। यदि आपके कंप्यूटर का अड्रेस पहले से सूचीबद्ध आरक्षित सबनेट श्रेणियों में से एक में आता है, तो आप सीधे इंटरनेट से कनेक्ट होने के बजाय एक राउटर से गुजर रहे हैं।

एक सबनेट पर आईपी अड्रेस के दो भाग होते हैं: नेटवर्क और नोड। नेटवर्क भाग सबनेट को पहचानता है। नोड, जिसे होस्ट भी कहा जाता है, नेटवर्क से जुड़ा कंप्यूटर उपकरण का एक व्यक्तिगत टुकड़ा है जिसे एक अद्वितीय address की आवश्यकता होती है। प्रत्येक कंप्यूटर जानता है कि सबनेट मास्क का उपयोग करके आईपी address के दो हिस्सों को कैसे अलग किया जाए। एक सबनेट मास्क कुछ हद तक आईपी address की तरह दिखता है, लेकिन यह वास्तव में सिर्फ एक फिल्टर है जिसका उपयोग यह निर्धारित करने के लिए किया जाता है कि आईपी address का कौन सा हिस्सा नेटवर्क और नोड को निर्दिष्ट करता है।

एक सबनेट मास्क में 0 बिट्स की श्रृंखला के बाद 1 बिट्स की एक श्रृंखला होती है। 1 बिट्स उन लोगों को इंगित करते हैं जिन्हें आईपी address में नेटवर्क बिट्स को मुखौटा करना चाहिए, केवल उन लोगों को प्रकट करना जो उस नेटवर्क पर एक अद्वितीय नोड की पहचान करते हैं। IPv4 मानक में, सबसे अधिक इस्तेमाल किए जाने वाले सबनेट मास्क में 1s और 0s के पूर्ण ऑक्टेट निम्नानुसार हैं:

- 255.0.0.0.0 = 11111111.00000000.00000000.00000000 = नेटवर्क के लिए आठ बिट्स, नोड्स के लिए 24 बिट्स
- 255.255.0.0 = 11111111.11111111.00000000.00000000 = नेटवर्क के लिए 16 बिट्स, नोड्स के लिए 16 बिट्स
- 255.255.255.0 = 11111111. 11111111.11111111.00000000 = नेटवर्क के लिए 8 बिट्स, नोड्स के आठ बिट्स

जो लोग बड़े नेटवर्क सेट करते हैं वे निर्धारित करते हैं कि वांछित सबनेट या नोड्स की संख्या के आधार पर सबनेट मास्क सबसे अच्छा काम करता है। अधिक सबनेट के लिए, नेटवर्क के लिए अधिक बिट्स का उपयोग करें; सबनेट प्रति अधिक नोड्स के लिए, अधिक बिट्स का उपयोग करें। इसका मतलब non-standard mask मूल्यों का उपयोग करना हो सकता है। उदाहरण के लिए, यदि आप नेटवर्क के लिए 10 बिट्स और 22 नोड्स के लिए उपयोग करना चाहते हैं, तो आपके सबनेट मास्क मूल्य को दूसरे ऑक्टेट में 11000000 का उपयोग करना होगा, जिसके परिणामस्वरूप 255.192.0.0 का सबनेट मास्क मूल्य होगा।

एक सबनेट में आईपी address के बारे में ध्यान देने योग्य एक और महत्वपूर्ण बात यह है कि पहले और अंतिम address आरक्षित हैं। पहला अड्रेस सबनेट को स्वयं पहचानता है, और अंतिम अड्रेस उस सबनेट पर सिस्टम के लिए प्रसारण address की पहचान करता है।

1.9 पोर्ट

एक पोर्ट एक भौतिक डॉकिंग बिंदु है जिसके उपयोग से एक बाहरी डिवाइस को कंप्यूटर से जोड़ा जा सकता है। यह प्रोग्रामेटिक डॉकिंग पॉइंट भी हो सकता है, जिसके माध्यम से जानकारी किसी प्रोग्राम से कंप्यूटर या इंटरनेट पर प्रवाहित होती है।

एक नेटवर्क पोर्ट जो इंटरनेट प्रोटोकॉल सूट के ट्रांसपोर्ट लेयर प्रोटोकॉल द्वारा प्रदान किया जाता है, जैसे Transmission Control Protocol (TCP) और User Diagram Protocol (UDP) एक संख्या है जो दो कंप्यूटरों के बीच एंडपॉइंट संचार परोसता है।

यह निर्धारित करने के लिए कि आने वाले ट्रैफिक को किस दिशा में निर्देशित किया जाना चाहिए, विभिन्न पोर्ट नंबर का उपयोग किया जाता है। वे नेटवर्क सेवाओं को चलाने के लिए एकल आईपी address के साथ एकल होस्ट की अनुमति देते हैं। प्रत्येक पोर्ट नंबर की एक अलग सेवा होती है, और प्रत्येक होस्ट के लिए प्रति आईपी address पर 65535 पोर्ट हो सकते हैं। इन Ports के उपयोग के प्रबंधन के लिए इंटरनेट असाइनेड नंबर अथॉरिटी (IANA) जिम्मेदार है। आईएएनए द्वारा Ports के लिए तीन श्रेणियां हैं —

- **0 से 1023** - प्रसिद्ध पोर्ट या सिस्टम पोर्ट।

कुछ प्रसिद्ध PORT हैं—

पोर्ट नंबर	ट्रांसपोर्ट प्रोटोकॉल	सेवा का नाम
20, 21	TCP	File Transfer Protocol
23	TCP	Telnet
25	TCP	Simple Mail Transfer Protocol (SMTP)
53	TCP and UDP	Domain Name System (DNS)
110	TCP	Post Office Protocol (POP3)
123	UDP	Network Time Protocol (NTP)

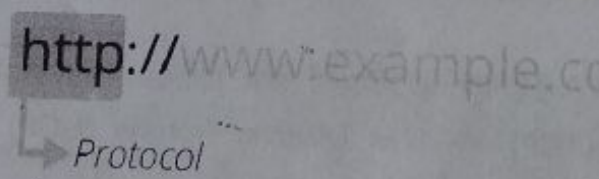
- **1024 से 49151** registered ports -अनुरोधित संस्था द्वारा आवेदन पर एक विशिष्ट सेवा के लिए आईएएनए द्वारा निर्दिष्ट requesting entity।
- **49152 से 65 535**-डायनामिक (प्राइवेट, हाई) पोर्ट 49,152 से लेकर 65,535 तक के हैं। निजी या क्लाइंट सेवा या अस्थायी प्रयोजनों के द्वारा उपयोग किया जा सकता है।

1.10 URL

एक URL (Uniform Resource Locator) वेब पर दिए गए किसी विशिष्ट resource के address से अधिक कुछ नहीं है। सिद्धांत रूप में, प्रत्येक मान्य URL एक अद्वितीय resource की ओर इशारा करता है। इस तरह के resource एक HTML पृष्ठ, एक सीएसएस दस्तावेज़, एक छवि, आदि हो सकते हैं। Practical में, कुछ एक्सेप्शन हैं, सबसे सामान्य एक resource की ओर इशारा करते हुए एक URL है जो अब मौजूद नहीं है या जो स्थानांतरित हो गया है। जैसा कि URL और URL द्वारा दर्शाए गए resource को वेब सर्वर द्वारा नियंत्रित किया जाता है, यह वेब सर्वर के मालिक पर निर्भर है कि वह उस resource और उससे जुड़े URL को सावधानीपूर्वक प्रबंधित करे।

एक URL विभिन्न भागों से बना होता है, कुछ अनिवार्य और अन्य वैकल्पिक। आइए निम्नलिखित URL का उपयोग करते हुए सबसे महत्वपूर्ण भाग देखें:

http://www.example.com:80/path/to/myfile.html?key1=value1&key2=value2#SomewhereInTheDocument



HTTP प्रोटोकॉल है। URL का पहला भाग इंगित करता है कि ब्राउज़र को किस प्रोटोकॉल का उपयोग करना चाहिए। एक प्रोटोकॉल एक कंप्यूटर नेटवर्क के आसपास डेटा का आदान-प्रदान या स्थानांतरित करने के लिए set methods है। आमतौर पर वेबसाइटों के लिए यह HTTP प्रोटोकॉल या इसका सुरक्षित संस्करण, HTTPS है। वेब को इन दोनों में से एक की आवश्यकता होती है, लेकिन ब्राउज़र यह भी जानते हैं कि अन्य प्रोटोकॉल जैसे कि कैसे संभालना nwmailto: (मेल क्लाइंट open करने के लिए) या एफ़टीपी: फ़ाइल स्थानांतरण को संभालने के लिए, यदि आप ऐसे प्रोटोकॉल देखते हैं तो आश्चर्यचकित न हों।

www.example.com

Domain Name

www.example.com डोमेन नाम है। यह इंगित करता है कि किस वेब सर्वर से request किया जा रहा है। वैकल्पिक रूप से, सीधे उपयोग करना संभव है लेकिन क्योंकि यह कम सुविधाजनक है, इसलिए इसका उपयोग अक्सर वेब पर नहीं किया जाता है।

:80

Port

: 80 PORT है। यह वेब सर्वर पर resources तक पहुंचने के लिए उपयोग किए जाने वाले तकनीकी 'गेट' को इंगित करता है। यह आमतौर पर omit किया जाता है यदि वेब सर्वर अपने resources तक पहुंच प्रदान करने के लिए HTTP प्रोटोकॉल (HTTP के लिए 80 और HTTPS के लिए 443) के मानक पोर्ट का उपयोग करता है। अन्यथा यह अनिवार्य है।

/path/to/myfile.html

Path to the file

/path/to/myfile.htmlkesye सर्वर पर resource के लिए path है। वेब के शुरुआती दिनों में, इस तरह से एक path वेब सर्वर पर एक भौतिक फ़ाइल स्थान का प्रतिनिधित्व करता था। आजकल, यह ज्यादातर बिना किसी भौतिक वास्तविकता के वेब सर्वर द्वारा संचालित एक abstraction है।

?key1=value1&key2=value2

Parameters

? KEY 1 = value 1 और KEY 2 = value 2 वेब सर्वर को प्रदान किए गए अतिरिक्त पैरामीटर हैं। वे पैरामीटर KEY/ value की एक सूची है जिन्हें अलग किया गया है। वेब सर्वर resource को वापस करने से पहले अतिरिक्त stuff को add करने के लिए उन parameter का उपयोग कर सकते हैं। प्रत्येक वेब सर्वर के parameters के संबंध में अपने नियम हैं, और यह जानने का एकमात्र विश्वसनीय तरीका है कि वेब सर्वर के मालिक से पूछकर कोई विशिष्ट वेब सर्वर पैरामीटर संभाल रहा है या नहीं।

#SomewhereInTheDocument

Anchor

#SomewhereInTheDocument resource के दूसरे भाग के लिए एक anchor है। एक एंकर resource के अंदर 'बुकमार्क' का एक प्रकार का प्रतिनिधित्व करता है, जिससे ब्राउज़र उस 'बुकमार्क' स्थान पर स्थित कंटेन्ट को दिखाने के लिए निर्देश देता है। एक HTML दस्तावेज़ पर, उदाहरण के लिए, ब्राउज़र उस बिंदु तक स्क्रॉल करेगा जहां anchor परिभाषित किया गया है; वीडियो या ऑडियो दस्तावेज़ पर, ब्राउज़र उस समय तक जाने की कोशिश करेगा

जब तक anchor प्रतिनिधित्व करता है। यह ध्यान देने योग्य है कि # के बाद का हिस्सा, जिसे Fragement identifier के रूप में भी जाना जाता है, ये request के साथ सर्वर पर कभी नहीं भेजा जाता है।

आप एक नियमित डाक address की तरह एक URL के बारे में सोच सकते हैं: प्रोटोकॉल उस डाक सेवा का प्रतिनिधित्व करता है जिसे आप उपयोग करना चाहते हैं, डोमेन नाम शहर है, और PORT जिप कोड की तरह है; Path उस इमारत का प्रतिनिधित्व करता है जहाँ आपका मेल पहुँचाया जाना चाहिए; पैरामीटर भवन में अपार्टमेंट की संख्या जैसे अतिरिक्त जानकारी का प्रतिनिधित्व करते हैं; और अंत में, एंकर वास्तविक व्यक्ति का प्रतिनिधित्व करता है, जिसे आपने अपना मेल संबोधित किया है।

ABSOLUTE URL बनाम RELATIVE URL

URL का आवश्यक भाग उस संदर्भ पर काफी हद तक निर्भर करता है जिसमें URL का उपयोग किया जाता है। आपके ब्राउज़र के अड्रेस बार में, किसी URL का कोई संदर्भ नहीं है, इसलिए आपको एक ABSOLUTE URL प्रदान करना होगा, आपको प्रोटोकॉल (ब्राउज़र डिफ़ॉल्ट रूप से HTTP का उपयोग करता है) या पोर्ट (जो केवल तब आवश्यक है जब लक्षित वेब सर्वर कुछ असामान्य पोर्ट का उपयोग कर रहा है) को शामिल करने की आवश्यकता नहीं है, लेकिन URL के अन्य सभी हिस्से आवश्यक हैं।

जब किसी डॉक्यूमेंट के भीतर URL का उपयोग किया जाता है, जैसे कि HTML पृष्ठ में, चीजें थोड़ी भिन्न होती हैं। क्योंकि ब्राउज़र में पहले से ही दस्तावेज़ का स्वयं का URL है, इसलिए यह इस जानकारी का उपयोग उस दस्तावेज़ के अंदर मौजूद किसी भी URL के गुम हिस्सों को भरने के लिए कर सकता है। हम केवल URL के Path भाग को देखकर किसी ABSOLUTE URL और किसी RELATIVE URL के बीच अंतर कर सकते हैं। यदि URL का Path भाग '/' वर्ण से शुरू होता है, तो ब्राउज़र उस resource को सर्वर के शीर्ष रूट से, वर्तमान दस्तावेज़ द्वारा दिए गए संदर्भ के संदर्भ के बिना लाएगा।

आइए इसे स्पष्ट करने के लिए कुछ उदाहरण देखें।

ABSOLUTE URL के उदाहरण

```
http://asianpublishers.co.in/index.php
```

IMPLICIT प्रोटोकॉल

```
asianpublishers.co.in/index.php
```

इस स्थिति में, ब्राउज़र उस URL को उसी प्रोटोकॉल के साथ कॉल करेगा, जो उस URL को होस्ट करने वाले दस्तावेज़ को लोड करने के लिए उपयोग किया जाता है।

IMPLICIT डोमेन नाम

```
/en-US/docs/Learn
```

यह एक HTML दस्तावेज़ के भीतर ABSOLUTE URL के लिए सबसे आम उपयोग मामला है। ब्राउज़र उसी प्रोटोकॉल और उसी डोमेन नाम का उपयोग करेगा, जिसका उपयोग उस URL को होस्ट करने वाले दस्तावेज़ को लोड करने के लिए किया गया था।

RELATIVE URL के उदाहरण

निम्नलिखित उदाहरणों को बेहतर ढंग से समझने के लिए, मान लें कि URL को निम्नलिखित URL पर स्थित दस्तावेज़ के भीतर से बुलाया गया है: http://asianpublishers.co.in/index.php

Sub-Resources

Contact

क्योंकि वह URL से शुरू नहीं होता है/, ब्राउज़र दस्तावेज़ को वर्तमान resource वाले उप-निर्देशिका में खोजने का प्रयास करेगा। इसलिए इस उदाहरण में, हम वास्तव में इस URL तक पहुँचना चाहते हैं : http://asianpublishers.co.in/Contact.php

डायरेक्टरी Tree में वापस जा रहे हैं

```
../CSS/display
```

इस मामले में, हम .. writing convention का उपयोग करते हैं-UNIX फ़ाइल सिस्टम world से Inherit है। उस ब्राउज़र को बताने के लिए जिसमें हम Directory से ऊपर जाना चाहते हैं। यहाँ हम इस URL तक पहुँचना चाहते हैं : http://asianpublishers.co.in/../CSS/display, जिसे सरल बनाया जा सकता है :: http://asianpublishers.co.in/CSS/display

SEMANTIC URL

URL वेब साइट के लिए एक मानव-पठनीय प्रविष्टि बिंदु का प्रतिनिधित्व करते हैं। उन्हें याद किया जा सकता है, और कोई भी उन्हें ब्राउज़र के एड्रेस बार में दर्ज कर सकता है। लोग वेब के core में हैं, और इसलिए इसे बनाने के लिए सबसे अच्छा अभ्यास माना जाता nwSEMANTIC URL। सिमेंटिक URL उन शब्दों का उपयोग करते हैं जिनमें IMPLICIT अर्थ होते हैं जो किसी को भी समझ में आ सकते हैं, भले ही उनका तकनीकी ज्ञान कैसा भी हो।

Linguistic SEMANTIC कंप्यूटर के लिए irrelevant हैं। आपने अक्सर ऐसे URL देखे होंगे जो random वर्णों के मैशअप की तरह दिखते हैं। लेकिन मानव-पठनीय URL बनाने के कई फायदे हैं :

- आपके लिए उन्हें manipulate करना आसान है।
- यह users के लिए चीजों को स्पष्ट करता है कि वे कहां हैं, वे क्या कर रहे हैं, वे क्या पढ़ रहे हैं या वेब पर किसके साथ बातचीत कर रहे हैं।
- कुछ खोज इंजन संबंधित पृष्ठों के वर्गीकरण को बेहतर बनाने के लिए उन शब्दार्थों का उपयोग कर सकते हैं।

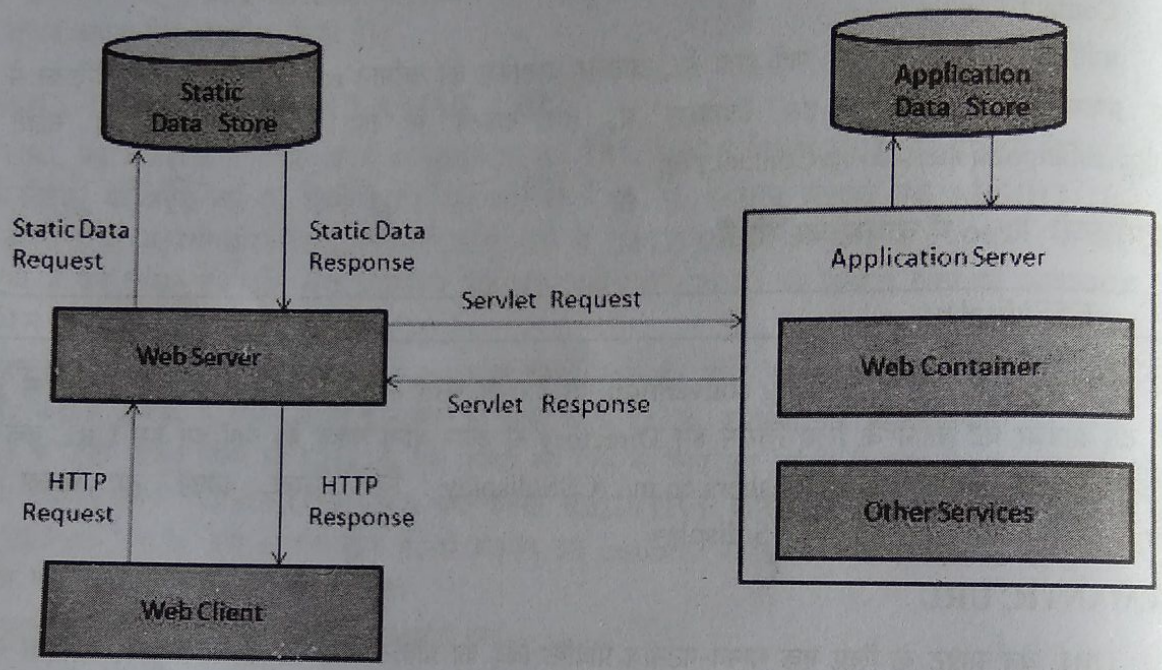
1.11 वेब सर्वर (Web Server)

वेब सर्वरएक कंप्यूटर है जहाँ वेब कंटेन्ट संग्रहीत की जाती है। मूल रूप से वेब सर्वर का उपयोग वेब साइटों को होस्ट करने के लिए किया जाता है, लेकिन अन्य वेब सर्वर जैसे गेमिंग, स्टोरेज, FTP, ईमेल आदि भी मौजूद हैं।

वेब सर्वर वर्किंग

वेब सर्वर क्लाइंट request के लिए निम्नलिखित दो तरीकों से जवाब देता है:

- अनुरोधित URL से जुड़े क्लाइंट को फ़ाइल भेजना।
- एक स्क्रिप्ट को आमंत्रित करके और डेटाबेस के साथ संवाद करके प्रतिक्रिया उत्पन्न करना



प्रमुख बिंदु

- जब क्लाइंट एक वेब पेज के लिए request भेजता है, तो वेब सर्वर अनुरोधित पृष्ठ के लिए खोज करता है यदि अनुरोधित पृष्ठ पाया जाता है तो यह क्लाइंट को HTTP response के साथ भेज देगा।
- यदि अनुरोधित वेब पेज नहीं मिला है, तो वेब सर्वर HTTP response भेजेगा: Error 404 Not found।
- यदि क्लाइंट ने कुछ अन्य resources के लिए request किया है, तो वेब सर्वर HTTP प्रतिक्रिया का निर्माण करने के लिए एप्लीकेशन सर्वर और डेटा स्टोर से संपर्क करेगा।

आर्किटेक्चर

वेब सर्वर आर्किटेक्चर निम्नलिखित दो दृष्टिकोणों का अनुसरण करता है :

1. Concurrent Approach
2. Single-Process-Event-Driven Approach.

समवर्ती दृष्टिकोण (Concurrent Approach)

Concurrent approach वेब सर्वर को एक ही समय में कई क्लाइंट requests को संभालने की अनुमति देता है। इसे निम्नलिखित विधियों द्वारा प्राप्त किया जा सकता है :

- Multi-process
- Multi-threaded
- Hybrid method.

मल्टी प्रोसेसिंग

इसमें एक एकल प्रक्रिया (पैरेंट प्रोसेस) कई एकल-थ्रेडेड (चाइल्ड प्रोसेस) शुरू करती है और इन चाइल्ड प्रोसेस में आने वाले requests को वितरित करती है। प्रत्येक child process एकल request को संभालने के लिए जिम्मेदार है।

यह पैरेंट प्रोसेस की जिम्मेदारी है कि वे लोड की निगरानी करें और निणर्य लें कि क्या प्रक्रियाओं को मारना चाहिए या कांटा जाना चाहिए।

मल्टी-थ्रेडेड

मल्टी-प्रोसेस के विपरीत, यह कई सिंगल-थ्रेडेड प्रक्रिया बनाता है।

हाइब्रिड

यह ऊपर दिए गए दो दृष्टिकोणों का संयोजन है। इस दृष्टिकोण में कई प्रक्रियों बनाई जाती हैं और प्रत्येक प्रक्रिया कई process शुरू करते हैं। प्रत्येक थ्रेड एक कनेक्शन संभालता है। एकल प्रक्रिया में कई थ्रेड का उपयोग करने से सिस्टम resources पर कम भार पड़ता है।

उदाहरण

निम्न तालिका आज उपलब्ध सबसे प्रमुख वेब सर्वरों का वर्णन करती है—

S.No.	वेब सर्वर Description
1	**Apache HTTP Server** यह Apache Software Foundation द्वारा विकसित दुनिया का सबसे लोकप्रिय वेब सर्वर है। अपाचे वेब सर्वर एक ओपन सोर्स सॉफ्टवेयर है और इसे लिनक्स, यूनिक्स, विंडोज, FreeBSD, Mac OS X और अधिक सहित लगभग सभी ऑपरेटिंग सिस्टम पर स्थापित किया जा सकता है। लगभग 60% वेब सर्वर मशीनें Apache Web Server चलाती हैं।
2	**Internet Information Services (IIS)** इंटरनेट सूचना सर्वर (IIS) Microsoft से एक उच्च प्रदर्शन वेब सर्वर है। यह वेब सर्वर विंडोज NT / 2000 और 2003 प्लेटफार्मों पर चलता है (और आगामी नए विंडोज संस्करण पर भी हो सकता है)। IIS Windows NT / 2000 और 2003 के साथ बंडल में आता है; क्योंकि घ्ए को ऑपरेटिंग सिस्टम के साथ कसकर एकीकृत किया गया है, इसलिए इसे प्रबंधित करना आसान है।
3	**lighttpd** Lighttpd, उच्चारण lighty भी एक फ्री वेब सर्वर है जो FreeBSD ऑपरेटिंग सिस्टम के साथ

	वितरित किया जाता है। यह ओपन सोर्स वेब सर्वर तेज, सुरक्षित है और सीपीयू की कम खपत करता है। लाइटटैप विंडोज, मैक ओएस एक्स, लिनक्स और सोलारिस ऑपरेटिंग सिस्टम पर भी चल सकता है।
4	**Sun Java System Web Server** Sun microsystem का यह वेब सर्वर मध्यम और बड़ी वेब साइटों के लिए अनुकूल है। सर्वर मुक्त होने के बावजूद open source नहीं है। हालाँकि, यह विंडोज, लिनक्स और यूनिक्स प्लेटफार्मों पर चलता है। सन जावा सिस्टम वेब सर्वर वेब 2.0 के लिए आवश्यक विभिन्न भाषाओं, लिपियों और तकनीकों का समर्थन करता है जैसे जेएसपी, जावा सर्वलेट्स, पीएचपी, पर्ल, पायथन, और रूबी ऑन रेल्स, एएसपी और कोल्डफ्यूज़न आदि।
5	**JIGSAW सर्वर** JIGSAW (W3C का सर्वर) वर्ल्ड वाइड वेब कंसोटिन्यम से आता है। यह open source और free है और लिनक्स, यूनिक्स, विंडोज और मैक ओएस एक्स फ्री बीएसडी आदि जैसे विभिन्न प्लेटफार्मों पर चल सकता है। JIGSAW जावा में लिखा गया है और सीजीआई स्क्रिप्ट और पीएचपी program चला सकते हैं।

1.12 टॉमकैट वेब सर्वर

Apache Tomcat सॉफ्टवेयर जावा सर्वलेट, JavaServer पेज, जावा एक्सप्रेशन लैंग्वेज और जावा websorket का एक ओपन सोर्स कार्यान्वयन है।

Apache Tomcat का उपयोग आमतौर पर सर्वलेट कंटेनर के रूप में किया जाता है भले ही Tomcat में स्टेटिक कंटेन्ट परोसने के लिए पूरी तरह कार्यात्मक HTTP सर्वर हो। अधिकांश उत्पादन में, Tomcat का उपयोग Apache HTTP सर्वर के साथ संयोजन में किया जाता है, जहाँ Apache HTTP सर्वर HTML, चित्र आदि जैसी स्टेटिक कंटेन्ट को सम्मिलित करता है, और Tomcat को डायनामिक कंटेन्ट के लिए requests को फॉरवर्ड करता है। ऐसा इसलिए है क्योंकि Apache HTTP सर्वर टॉमकैट की तुलना में अधिक उन्नत विकल्पों का समर्थन करता है।

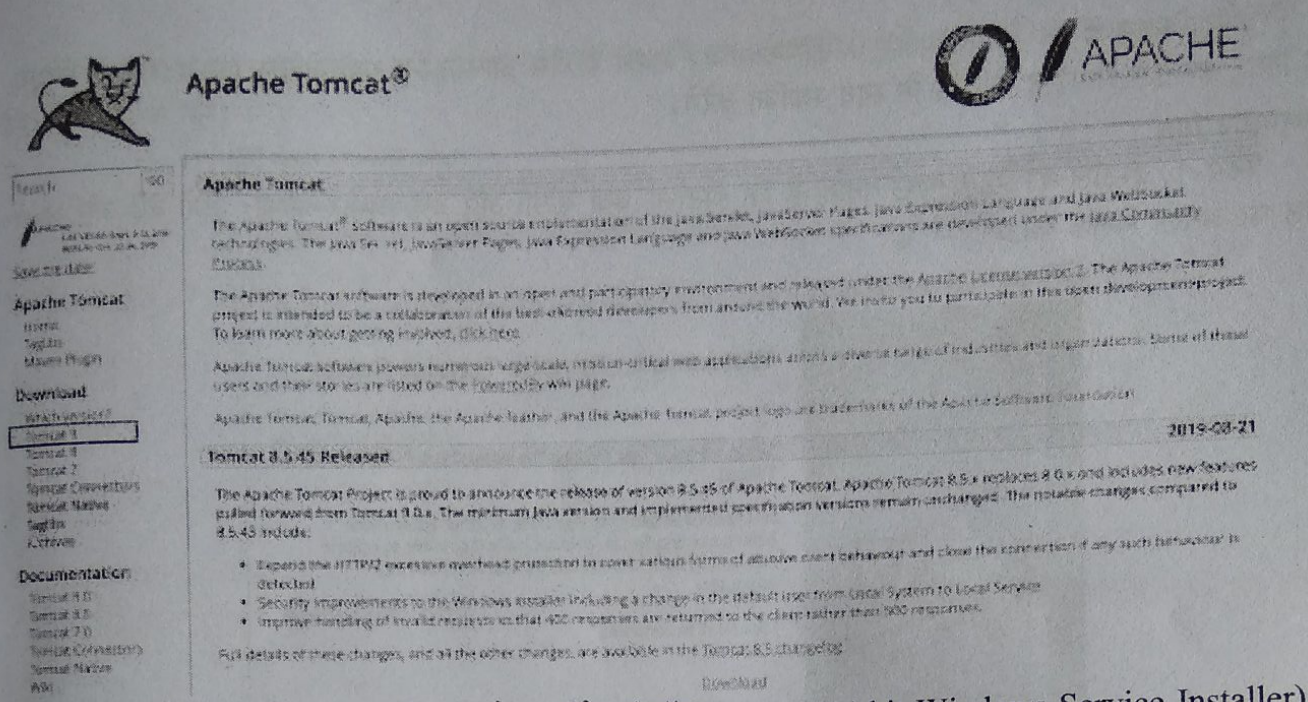
नवीनतम Apache Tomcat संस्करण 8.5 HTTP / 2, JSSE के लिए OpenSSL, TLS वर्चुअल होस्टिंग और जेपीएससी 1.1 के लिए समर्थन जोड़ता है

आवश्यक शर्तें

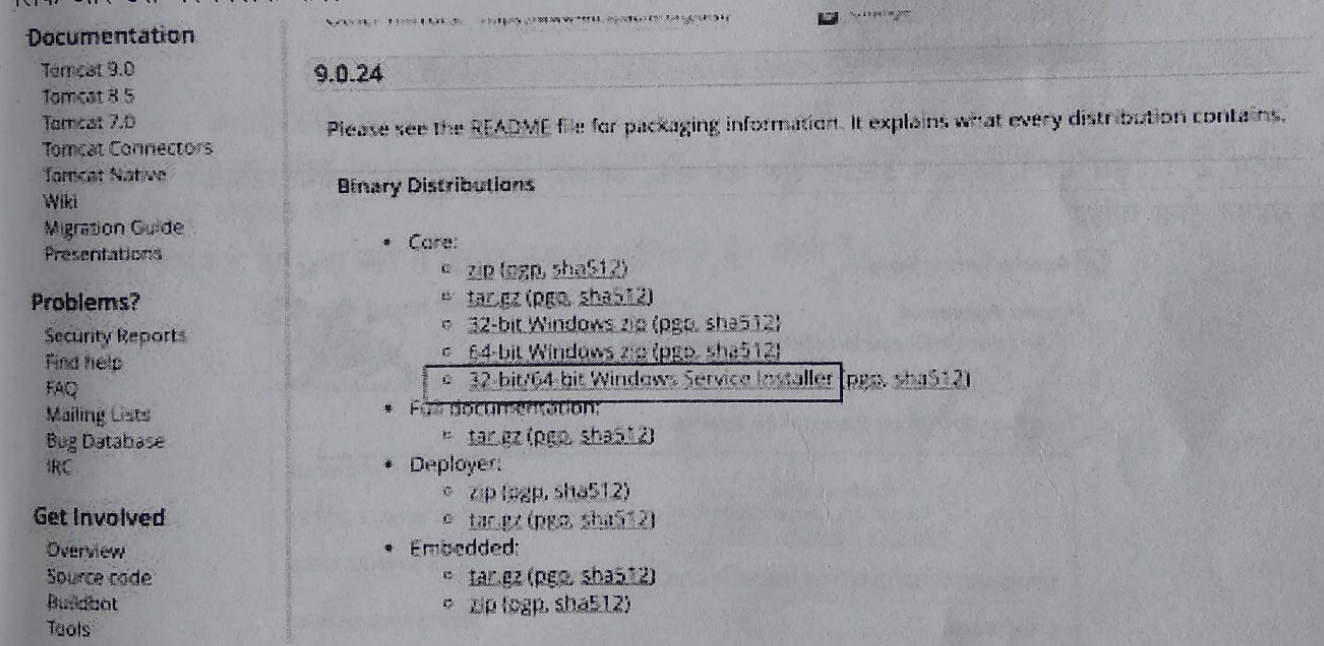
JDK या JRE को सर्वर पर Tomcat 9 को कॉन्फिगर करने से पहले विंडोज सर्वर पर स्थापित करने की आवश्यकता होगी। OpenJDK और Amazon Corretto ओपन-सोर्स जावा डेवलपमेंट किट प्रदाताओं के दो उदाहरण हैं।

तोमकाट 9 स्थापित करना

अपना ब्राउज़र खोलें और https: //tomcat.apache.org पर जों। Tomcat 9 के लिकं पर क्लिक करें



अगला, खोजें 32-बिट / 64-बिट विंडोज सर्विस इंस्टॉलर (32-bit/64-bit Windows Service Installer) लिंक और उस पर क्लिक करें।



यह लिंक स्वचालित रूप से विंडोज सर्विस इंस्टॉलर को खोलेगा। इंस्टॉलर विंडो पॉप अप हो जाने के बाद, आप स्टार्ट डाउनलोड बटन पर क्लिक करके इंस्टॉल शुरू कर देंगे। डाउनलोड पूरा होने के बाद, इंस्टॉलेशन शुरू करने के लिए .exe इंस्टॉलर पर डबल क्लिक करें।

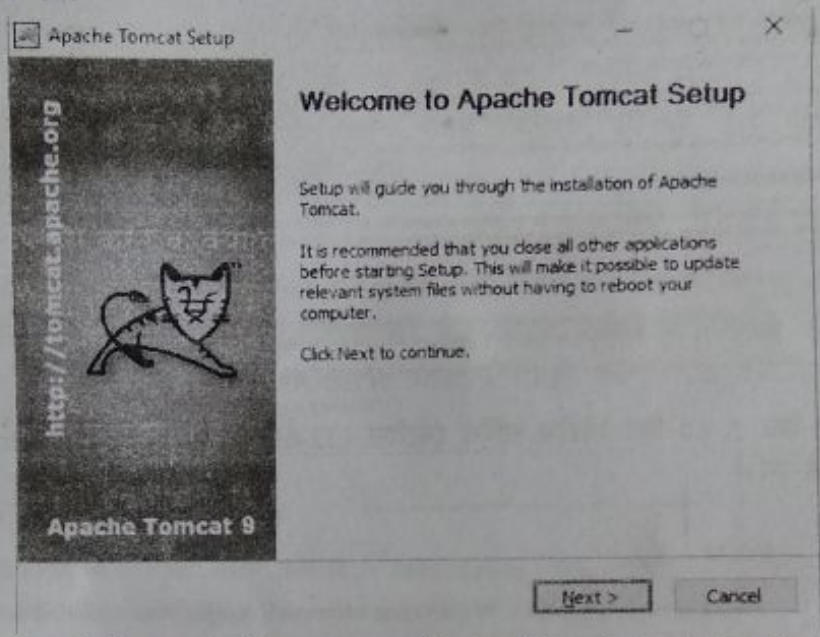
स्थापित विज़ार्ड को पूरा होने में केवल कुछ क्षण लगने चाहिए। स्थापित विज़ार्ड समाप्त होने के बाद, यह आपको कुछ अलग निणर्य लेने के लिए कहेगा।

नीचे निम्नलिखित विकल्प दिए गए हैं जो आपको टॉमकैट 9 को स्थापित करने के लिए कहा जाएगा।

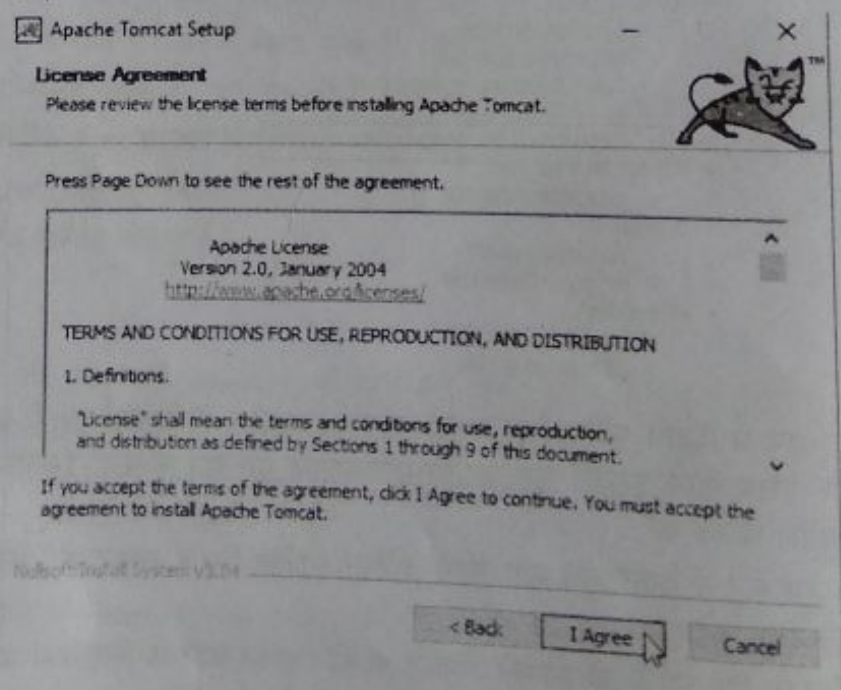
कृपया ध्यान दें कि आप टॉमकैट 9 कॉन्फिगर कर सकते हैं जैसे आप करना चाहे लेकिन इस प्रदर्शन के उद्देश्य के लिए, हम इसे डिफ़ॉल्ट सेटिंग्स के साथ स्थापित करेंगे।

Installation

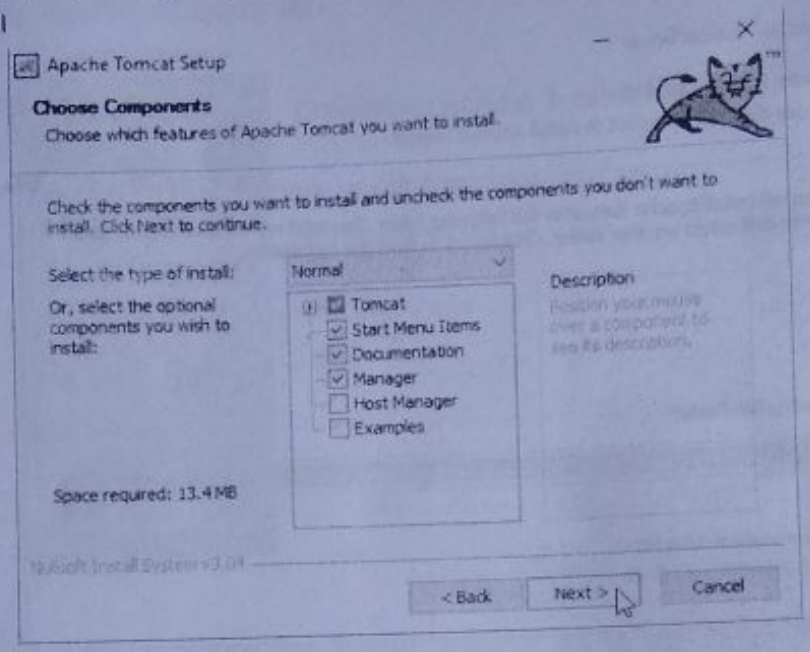
स्टेप 1. इंस्टॉल का पहला पृष्ठ बताता है कि इंस्टॉलर क्या करेगा और क्या उम्मीद करेगा। आगे बढ़ें और Next पृष्ठ पर अगला क्लिक करें।



स्टेप 2 : इससे पहले कि आप इंस्टॉल शुरू कर सकें, आपको अपाचे लाइसेंस एग्रीमेंट टॉमकैट 9 सर्विस के लिए सहमत होना चाहिए

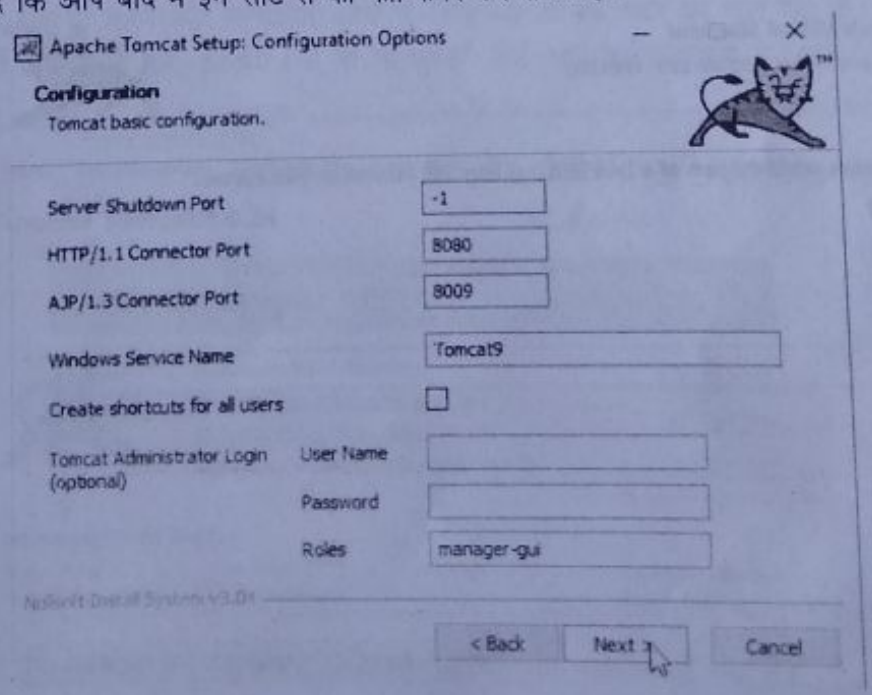


स्टेप 3 : 'इंस्टॉल का प्रकार चुनें' ड्रॉपडाउन सूची पर क्लिक करें और 'चुनें' Full 'विकल्प install करें और फिर next क्लिक करें।

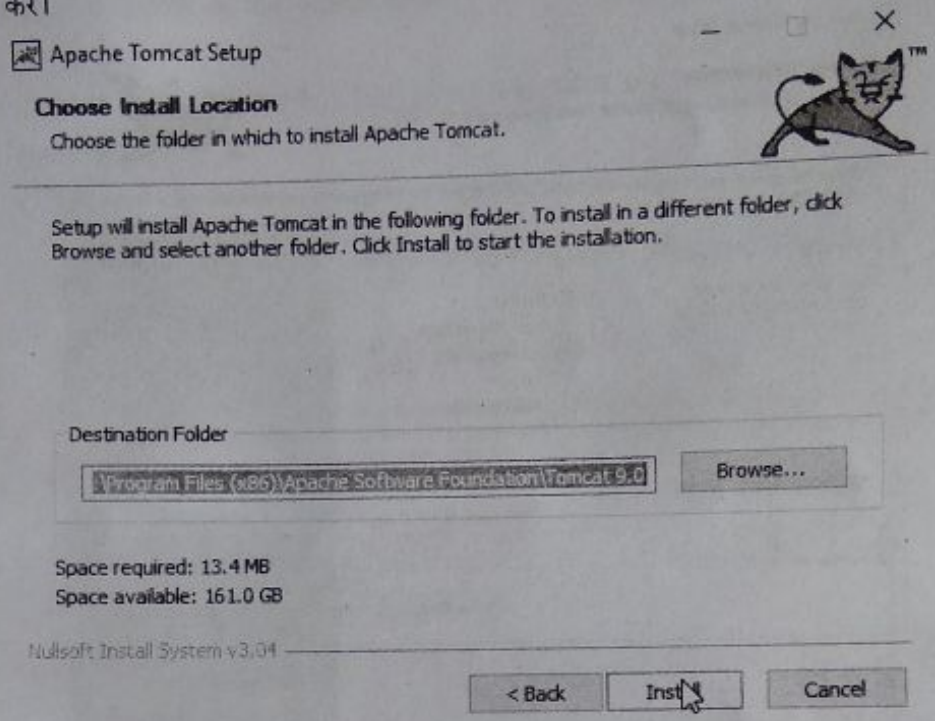


स्टेप 4 : अगली स्क्रीन कॉन्फिगरेशन स्क्रीन है। यह स्क्रीन आपको किसी भी डिफ़ॉल्ट पोर्ट को सेट करने की अनुमति देगा जिसे आप सेवा के माध्यम से कनेक्ट करना चाहते हैं, और आपको व्यवस्थापक user नाम और पासवर्ड सेट करने की भी अनुमति देगा।

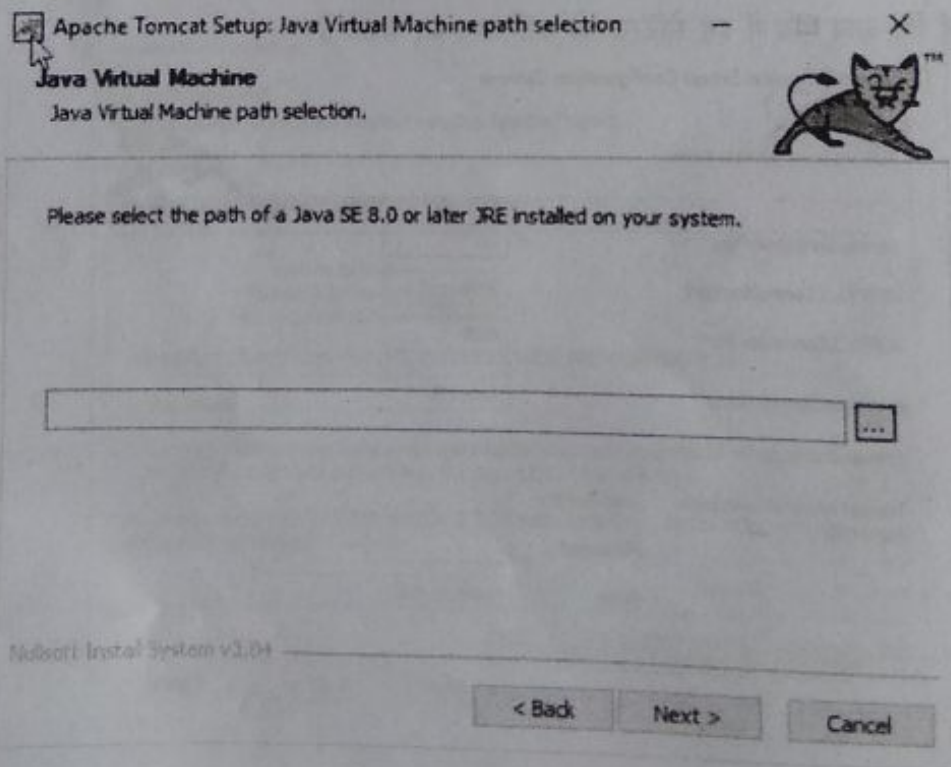
कृपया ध्यान दें कि आप बाद में इन सेटिंग्स को कॉन्फिगर कर सकते हैं।



स्टेप 5 : इसके बाद, आप वह स्थान चुनेंगे जहाँ आप एग्रीमेंट टॉककैट 9 सर्विस install करना चाहते हैं। next पर क्लिक करें।

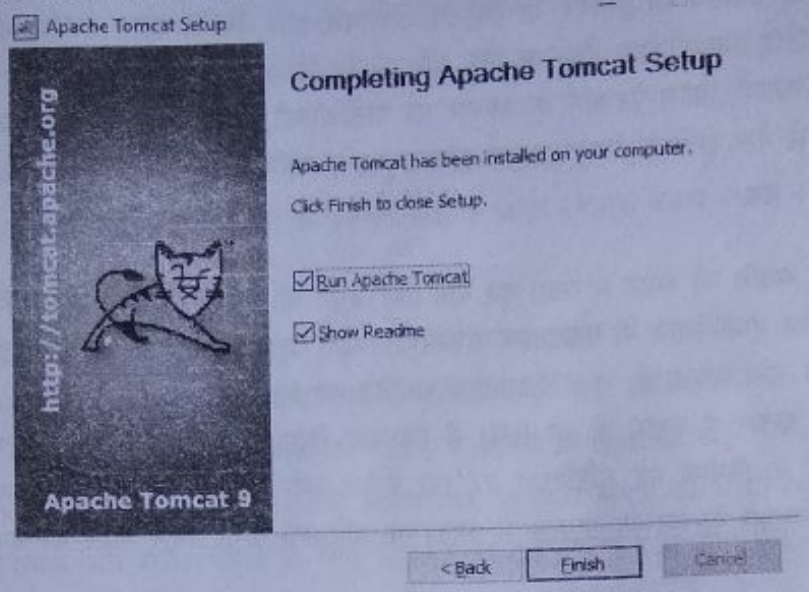


स्टेप 6 : अपने सर्वर पर टॉमकैट 9 को सही ढंग से स्थापित करने के लिए, विज़ार्ड आपको अपने सर्वर पर किसी अन्य जावा से संबंधित सॉफ़्टवेयर का स्थान चुनना चाहेगा।



स्टेप 7 : इंस्टॉल पूरा होने के बाद, next क्लिक करें।

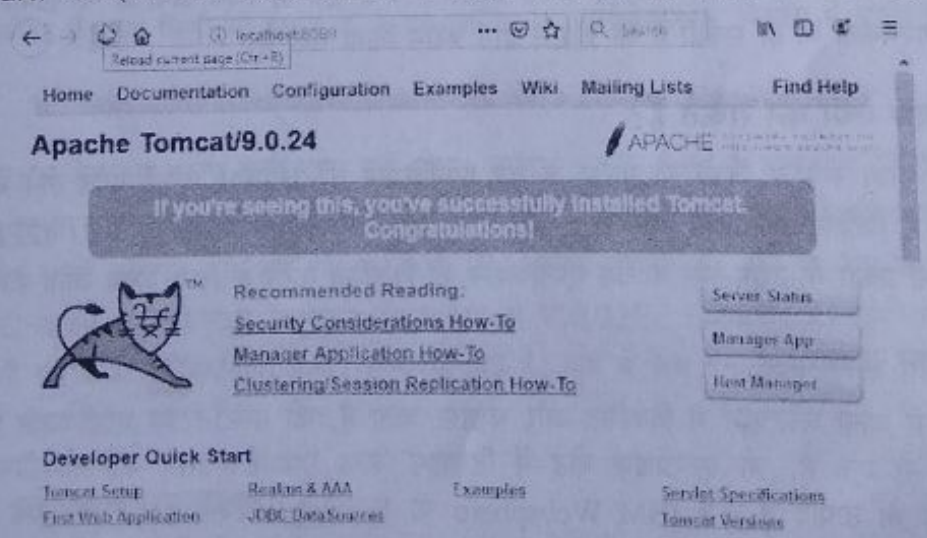
अंत में, Readme चेकबॉक्स को अनचेक करें।



अगर आप Apache शुरू करना पसंद करते हैंटॉम्केट 9 इंस्टॉल होने के बाद, चिह्नित किए गए चेकबॉक्स को छोड़ दें। यदि नहीं, तो आप उस चेकबॉक्स को अनचेक करें।

यह सुनिश्चित करने के लिए कि सेवा चल रही है, विंडोज स्टार्टअप मेनू पर जों और services.cmd टाइप करें। उपलब्ध सेवाओं की सूची से, Apache Tomcat 9 service खोजें, सेवा नाम पर राइट-क्लिक करें, start का चयन करें, और सुनिश्चित करें कि सेवा सफलतापूर्वक शुरू होती है। आपको सेवा नाम के बगल में 'रनिंग' स्थिति दिखनी चाहिए।

टॉमकैट 9 को स्थापित करने और सत्यापित करने के लिए यह आपके सर्वर पर चल रहा है, अपने ब्राउज़र को लोकलहोस्ट पर खोलें और इंगित करें: 8080 (या जो भी कस्टम पोर्ट आप कॉन्फिगरेशन में रखते हैं)।



लोड होने वाला पृष्ठ आपको बताएगा कि टॉमकैट 9 सफलतापूर्वक स्थापित हो गया है।

1.13 J2EE

J2EE को Java to Enterprise Edition के रूप में विस्तारित किया जा सकता है, जो एक स्टैंडअलोन जावा environment है जिसका उपयोग सॉफ्टवेयर या सिस्टम डेवलपर्स द्वारा वेब आधारित एप्लीकेशन या वेबसाइट के निर्माण और तैनाती के लिए किया जाता है। यह एक प्रसिद्ध मंच है जो जावा प्रोग्रामिंग भाषा का उपयोग इसकी विश्वसनीयता, सुरक्षा सुविधाओं, क्रॉस-प्लेटफॉर्म के आधार पर पोर्टेबिलिटी, आसान कार्यान्वयन, नवीनतम और ट्रेंडिंग तकनीकी आवश्यकताओं से मेल खाने के लिए लगातार अपग्रेड करने, आदि के लाभ के कारण करता है।

J2EE अंडरस्टैंडिंग

जावा की एक लंबी अवधि को ध्यान में रखते हुए एक छोटे एप्लेट को एक वेब एप्लीकेशन पर चलाने से विकसित किया गया है, अब यह वेब एप्लीकेशन के एंटरप्राइज़ संस्करण में आ गए है। इस तरह का एंटरप्राइज़ वेब एप्लीकेशन कई सर्वरों पर आसानी से चल सकता है, कुछ एंटरप्राइज़ संबंधित सहायता प्रदान करता है, इसका मतलब यह है कि यह एक एप्लीकेशन को चलाने में सक्षम हो जो EJB में विकसित किया गया है। यह प्लेटफ़ॉर्म मुख्य रूप से कुछ प्रकार के वेब एप्लीकेशन के निर्माण को सुनिश्चित कर रहा है जो सर्वर-साइड एप्लीकेशन के रूप में परिभाषित हो सकते हैं। यह विकसित सेवाओं के सेट के माध्यम से आवश्यक बुनियादी ढाँचा प्रदान करने की एक आर्किटेक्चर का अनुसरण कर रहा है।

J2EE काम करना इतना आसान कैसे करता है?

यह आम तौर पर कुछ महत्वपूर्ण व्यावसायिक उद्यम वेब एप्लीकेशन को बहुत आसानी से प्रदान करता है। यह डेवलपर को अपने वेब एप्लीकेशन को विकसित करने और एक विशिष्ट प्लेटफॉर्म में समान तैनाती के लिए महान प्लेटफॉर्मों में से एक के साथ आता है। J2EE आम तौर पर एपीआई की varieties रखता है जो एक कठिन उद्यम प्रणाली के निर्माण के लिए हमेशा एक सहायक वातावरण बनाते हैं। जबकि इसे पूरे J2EE फीचर्स की आधी तस्वीर माना जा सकता है। J2EE एपीआई के कुछ आवश्यक विनिर्देशों द्वारा एक साथ काम करने वाले किसी भी प्रकार के उद्यम के विकास और निर्माण या निर्माण के लिए मानक संस्करण में से एक है, जिसे ठीक से परिभाषित किया गया है और सेवाओं की आवश्यकता भी हो सकती है जो J2EE द्वारा प्रदान किया गया है।

आप J2EE के साथ क्या कर सकते हैं?

इसका उपयोग मुख्य रूप से किसी भी प्रकार के वेब एप्लीकेशन को HTML पृष्ठों मदद लेने के लिए किया जाता है से या जावा में विशिष्ट एप्लेट से और जावा आधारित एप्लीकेशन को परिभाषित करते हैं। J2EE का उपयोग मुख्य रूप से किसी भी प्रकार के उद्यम-स्तर के वेब एप्लीकेशन को विकसित करने के लिए किया जाता है।

J2EE के साथ काम करना

यह मुख्य रूप से जावा प्लेटफ़ॉर्म में विकसित और चलाया जाता है, जो मल्टी-टियर एप्लीकेशन को चलाने के लिए मुख्य एजेंडे में से एक है, जो एंटरप्राइज़ मोड में डिज़ाइन किया गया है। इसमें कई लोकप्रिय मल्टी-टियर एप्लीकेशन सर्वर तरह के उत्पाद हैं जैसे IBM Websphere या Weblogic जिसे J2EE तकनीक पर आधारित डिजाइन किया गया है। सामान्य रूप से J2EE प्रौद्योगिकी ने व्यावसायिक तर्क के बारे में जानकारी प्राप्त करने के लिए एंटरप्राइज़ बीन classes तक पहुंचने के एक सामान्य दृष्टिकोण का पालन किया।

लाभ

कुछ प्रमुख फायदे हैं:

- यह क्रॉस-प्लेटफॉर्म पोर्टेबिलिटी की एक महत्वपूर्ण कार्यक्षमता प्रदान करता है।
- किसी भी अन्य जावा-आधारित तकनीक की तरह, ओपन-सोर्स लाइब्रेरी आम डेवलपर के लिए आसानी से उपलब्ध हैं।
- सर्वर-साइड में आसानी से तैनाती के लिए कुछ बड़े एंटरप्राइज़-स्तर एप्लीकेशन को संभालना है।
- हमेशा J2EE प्रौद्योगिकियों द्वारा अधिकतम वैश्विक W3C मानक बनाए रखना और कवर करना।

इसकी आवश्यकता क्यों है?

J2EE को तेजी से समझने के लिए कुछ प्रमुख ज्ञान की आवश्यकता हो सकती है:

1. विशेष रूप से सर्वलेट और जेएसपी दृश्य में कुछ महत्वपूर्ण ज्ञान रखें।
2. किसी भी वेब ढांचे पर आरामदायक विशेष रूप से स्ट्रट्स या Spring पर।
3. Service-Oriented architecture में आरामदायक या स्पष्ट समझ की आवश्यकता होती है।
4. आरामदायक और स्पष्ट समझ के लिए SOAP या REST दोनों में वेब सेवा दृष्टिकोण की आवश्यकता होती है।
5. सीएसएस, जावास्क्रिप्ट, HTML या JQuery जैसी आमतौर पर उपलब्ध तकनीकों पर सहज है।
6. लोगों को मार्कअप भाषाओं (JSON या XML) की तरह आरामदायक होने पर इसे अतिरिक्त लाभ जोड़ा जाएगा।

हमें J2EE का उपयोग क्यों करना चाहिए?

J2EE को मुख्य रूप से सामान्य HTML पृष्ठों, अन्य अनुप्रयोगों, या वेब-आधारित एप्लीकेशन को विकसित करने में मदद करता है। J2EE द्वारा मुख्य रूप से संबोधित की जाने वाली दो प्रमुख प्रौद्योगिकियां जेएसपी (जावा सर्वर पेज) और सर्वलेट हैं। यह किसी भी प्रकार के उद्यम-स्तरीय वेब एप्लीकेशन के विकास में गंभीर रूप से उपयोग किया जाता है।

Scope

जावा/j2EE डेवलपर स्कोप कभी खत्म नहीं होगा। जावा / J2EE field में आवश्यक विशिष्ट विशेषज्ञता स्तर के लिए वर्तमान बाजार में बहुत सारे अवसर उपलब्ध हैं। किसी भी प्रकार के उपलब्ध लोकप्रिय टूल के लिए, java / j2EE विशेषज्ञ लोगों को उन टूल को पेश करने के लिए हमेशा स्वागत है। अब के रूप में अधिकतम संगठन या वर्तमान आईटी बाजार में अधिकतम उद्घाटन विशेष रूप से जावा/J2EE field पर हैं। इसलिए इस विशिष्ट क्षेत्र के लोग किसी भी प्रकार की परिस्थिति में हमेशा मूल्यवान होते हैं। कम से कम एक काम हमेशा बिना किसी एक्सेप्शन के उनके हाथ में होना चाहिए।

हमें J2EE की आवश्यकता क्यों है?

यह आर्किटेक्चर वर्तमान उद्योग में उपलब्ध लोकप्रिय आर्किटेक्चर में से एक है। यह एल्ह Java System एप्लीकेशन सर्वर समूह द्वारा स्थापित किया गया है, इसे हमेशा Sun Java Enterprise System के घटकों में से एक

माना जाता है। डेवलपर्स जो वास्तव में आरामदायक हैं और जावा/J2EE प्रौद्योगिकियों के विशेषज्ञ हैं, उनके पास वर्तमान उद्योग में अपने कौशल को लागू करने के लिए बहुत सारे अवसर हैं। कोई भी J2EE डेवलपर किसी भी तरह की वेब या वेब सेवा से संबंधित एप्लीकेशन को विकसित करने या बनाने में आसानी से सक्षम हो सकता है

Java EE architecture goals

जावा ईई आर्किटेक्चर कई मामलों में, आधुनिक अनुप्रयोगों के निर्माण के दौरान डेवलपर्स के सामने सबसे आम चुनौतियों को सरल बनाने वाली सेवों प्रदान करता है।

उदाहरण के लिए, एक आम चुनौती उद्यम डेवलपर्स का है कि वेब-आधारित ग्राहकों से आने वाले requests को कैसे संभालना है। इस चुनौती को आसान बनाने के लिए, जावा ईई प्रदान करता है सर्वलेट JavaServer Pages (JSP) एपीआई, जो गतिविधियों के लिए तरीके प्रदान करता है जैसे कि किसी user ने ऑनलाइन फॉर्म में टेक्स्ट फ़ील्ड में क्या टाइप किया है या ब्राउज़र पर कुकीज स्टोर करना।

एक अन्य सामान्य कार्य यह है कि किसी डेटाबेस में जानकारी को कैसे संग्रहीत और पुनर्प्राप्त किया जाए। इस लक्ष्य को पूरा करने के लिए, जावा ईई Java Persistence API (JPA) प्रदान करता है, जो किसी प्रोग्राम के भीतर इस्तेमाल होने वाले डेटा को डेटाबेस के table और rows में संग्रहीत जानकारी के लिए मैप करना आसान बनाता है। इन सभी एपीआई का अच्छी तरह से परीक्षण किया गया है, जावा डेवलपर्स के लिए सीखने में अपेक्षाकृत आसान है और उद्यम विकास के कुछ सबसे कठिन हिस्सों को सरल बना सकता है।

जावा ईई कोर प्रौद्योगिकियां

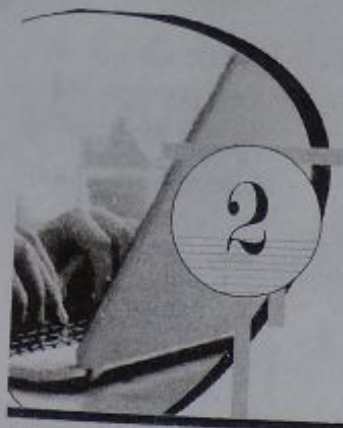
चार उपर्युक्त एपीआई के साथ, जावा ईई कोर प्रौद्योगिकियों के रूप में शामिल 30 से अधिक जावा एपीआई हैं, ये जावा ईई कोर प्रौद्योगिकियां मोटे तौर पर निम्नलिखित श्रेणियों में आती हैं —

- **HTTP क्लाइंट टेक्नोलॉजी :** जावा ईई में वेबसॉकेट प्रोग्रामिंग के लिए जावा एपीआई, के लिए एक एपीआई शामिल है JSON प्रसंस्करण, जेएसएफ और सर्वलेट एपीआई और जेएसपी स्टैंडर्ड टैग लाइब्रेरी (जेएसटीएल) शामिल है।
- **डेटाबेस और resource access technologies :** बाहरी और बैक-एंड सिस्टम के साथ बातचीत के लिए, जावा ईई में जावामेल, एक मानक कनेक्टर आर्किटेक्चर, एक Java Message Service (JMS) शामिल है।
- **REST और वेब सेवा प्रौद्योगिकियाँ :** REST के विकास और Deployment में मदद करने के लिए Soap,एक्सएमएल और JSON आधारित वेब सेवाएँ, RESTful वेब सेवा (JAX-RS) के लिए जावा एपीआई और XML-आधारित वेब सेवों (JAX-WS)XML मैसेजिंग और XML रजिस्ट्रियों (JAXR) के लिए एपीआई के साथ शामिल हैं।
- **Java EE security और container management :** कस्टम जावा ईई सुरक्षा और प्रबंधन को लागू करने के लिएजावा ईई कंटेनर, सॉफ्टवेयर डेवलपर्स के पास कंटेनरों के लिए जावा authorization contact और कंटेनरों के लिए जावा प्रमाणीकरण सेवा प्रदाता इंटरफ़ेस है।

अभ्यास

1. वेब अनुप्रयोगों को परिभाषित करें।
2. डायनामिक वेबसाइट क्या है?
3. डायनेमिक वेबसाइट स्टेटिक वेबसाइट से कितनी अलग है?
4. सर्वर साइड प्रोग्रामिंग की विशेषतों दें ।
5. वेब अनुप्रयोगों के प्रकारों की व्याख्या करें।
6. आप स्टेटिक पृष्ठ पर एक डायनामिक पृष्ठ क्यों चुनेंगे?
7. विस्तार से 3 परत आर्किटेक्चर की व्याख्या करें।
8. क्लाइंट सर्वर आर्किटेक्चर के बारे में विस्तार से बतों।
9. आईपी एड्रेस क्या है?
10. आईपी address के हिस्सों की व्याख्या करें।
11. पोर्ट परिभाषित करें।
12. URL क्या है?
13. वेब सर्वर को परिभाषित करें।
14. J2EE की प्रमुख विशेषताओं पर प्रकाश डालें।
15. J2EE की मुख्य प्रौद्योगिकियां क्या हैं?

❑❑❑



2

डाटाबेस प्रोग्रामिंग यूजिंग जेडीबीसी
DATABASE PROGRAMMING USING JDBC

LEARNING OBJECTIVES :

After reading this chapter, you would be able to understand :

- सर्वर साइड प्रोग्रामिंग (Server Side Programming)
- क्लाइंट साइड प्रोग्रामिंग (Client Side Programming)
- वेब अनुप्रयोग (Web applications)
- वेब सर्वर और क्लाइंट (Web Server and Client)
- डायनामिक वेबसाइट (Dynamic Website)
- वेब अनुप्रयोग के तीन परत वास्तुकला (Three Layer Architecture of web application)
- क्लाइंट सर्वर आर्किटेक्चर (Client Server Architecture)
- आईपी अड्रेस (IP Address)
- पोर्ट (Port)
- URL
- वेब सर्वर (Web Server)
- टॉमकैट वेब सर्वर
- J2EE

2.1 JDBC

JDBC (Java Database Connectivity) डेटाबेस के साथ क्वेरी को जोड़ने और Execute करने के लिए एक जावा एपीआई है। यह JavaSE (Java Standard Edition) का एक हिस्सा है। JDBC API डेटाबेस से जुड़ने के लिए JDBC ड्राइवरों का उपयोग करता है। व्ऊँ ड्राइवर चार टाइप के होते हैं—

- JDBC-ODBC ब्रिज ड्राइवर,
- NATIVE ड्राइवर,
- नेटवर्क प्रोटोकॉल ड्राइवर, और
- THIN ड्राइवर

हम किसी भी relational डेटाबेस में संग्रहीत सारणीबद्ध (tabular) डेटा तक पहुँचने के लिए JDBC API का उपयोग कर सकते हैं। JDBC API की मदद से, हम डेटाबेस से डेटा को सेव, अपडेट, डिलीट और fetch कर सकते हैं। यह Microsoft द्वारा प्रदान की गई Open Database Connectivity (ODBC) की तरह है।

'JDBC का अर्थ जावा डेटाबेस कनेक्टिविटी है, जो जावा प्रोग्रामिंग भाषा और डेटाबेस की एक विस्तृत श्रृंखला के बीच database independent कनेक्टिविटी के लिए एक मानक जावा एपीआई है।'

JDBC लाइब्रेरी में नीचे उल्लिखित प्रत्येक कार्य के लिए एपीआई शामिल है जो आमतौर पर डेटाबेस के उपयोग से जुड़े होते हैं।

- किसी डेटाबेस से संबंध बनाना।
- SQL या MySQL स्टेटमेंट बनाना।
- डेटाबेस में SQL या MySQL क्वेरी का निष्पादन।
- परिणामी रिकॉर्ड को देखना और संशोधित करना।

जेडीबीसी की आवेदन

मौलिक रूप से, JDBC एक स्पेसिफिकेशन है जो एक अंतर्निहित डेटाबेस के लिए पोर्टेबल एक्सेस की अनुमति देने वाले इंटरफेस का एक पूरा सेट प्रदान करता है। जावा का उपयोग विभिन्न प्रकार के एक्सेक्यूटबलेस लिखने के लिए किया जा सकता है, जैसे—

- Java Applications
- Java Applets
- Java Servlets
- Java ServerPages (JSPs)
- Enterprise JavaBeans (EJBs)

ये सभी अलग-अलग एक्सेक्यूटबलेस एक डेटाबेस तक पहुँचने के लिए, और संग्रहीत डेटा का लाभ उठाने के लिए JDBC ड्राइवर का उपयोग करने में सक्षम हैं।

JDBC ODBC जैसी ही क्षमताएं प्रदान करता है, जिससे जावा प्रोग्राम database-independent कोड को शामिल कर सकते हैं।

2.2 JDBC ड्राइवर

JDBC ड्राइवर एक सॉफ्टवेयर घटक है जो जावा एप्लिकेशन को डेटाबेस के साथ इंटरैक्ट करने में सक्षम बनाता है। JDBC ड्राइवर आपके डेटाबेस सर्वर के साथ बातचीत करने के लिए JDBC API में परिभाषित इंटरफेस को लागू करते हैं।

उदाहरण के लिए, JDBC ड्राइवर का उपयोग करने से आप डेटाबेस कनेक्शन खोलने में सक्षम हो जाते हैं और SQL या डेटाबेस कमांड भेजकर इसके साथ बातचीत कर सकते हैं और फिर जावा के साथ परिणाम प्राप्त कर सकते हैं।

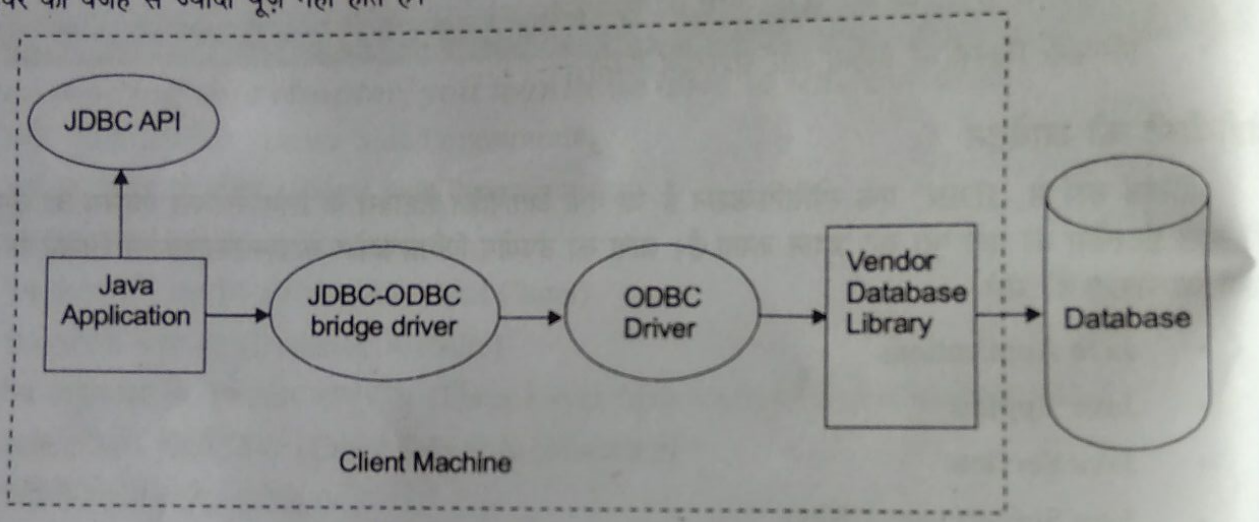
Java.sql पैकेज JDK के साथ शिप होते हैं, और उनके व्यवहार को परिभाषित करने के लिए विभिन्न काल्स होते हैं और उनका वास्तविक कार्यान्वयन तृतीय-पक्ष ड्राइवरों में किया जाता है। थर्ड पार्टी वेंडर अपने डेटाबेस ड्राइवर में java.sql.Driver इंटरफ़ेस लागू करते हैं।

JDBC ड्राइवर 4 प्रकार के होते हैं—

1. JDBC-ODBC ब्रिज ड्राइवर
2. NATIVE API ड्राइवर (आंशिक रूप से जावा ड्राइवर)
3. नेटवर्क प्रोटोकॉल ड्राइवर (पूरी तरह से जावा ड्राइवर)
4. THIN ड्राइवर (पूरी तरह से जावा ड्राइवर)

(1) JDBC-ODBC ब्रिज ड्राइवर

JDBC-ODBC ब्रिज ड्राइवर डेटाबेस से कनेक्ट करने के लिए ODBC ड्राइवर का उपयोग करता है। JDBC-ODBC ब्रिज ड्राइवर JDBC Method कॉल को ODBC फ़ंक्शन कॉल में परिवर्तित करता है। यह अब Thin ड्राइवर की वजह से ज्यादा यूज़ नहीं होते है।



JDBC-ODBC Bridge Driver

Oracle 8 से JDBC-ODBC ब्रिज का समर्थन नहीं करता है। Oracle अनुशंसा करता है कि आप अपने डेटाबेस के विक्रेता द्वारा दिए गए JDBC-ODBC ब्रिज के बजाय JDBC ड्राइवरों का उपयोग करें।

लाभः

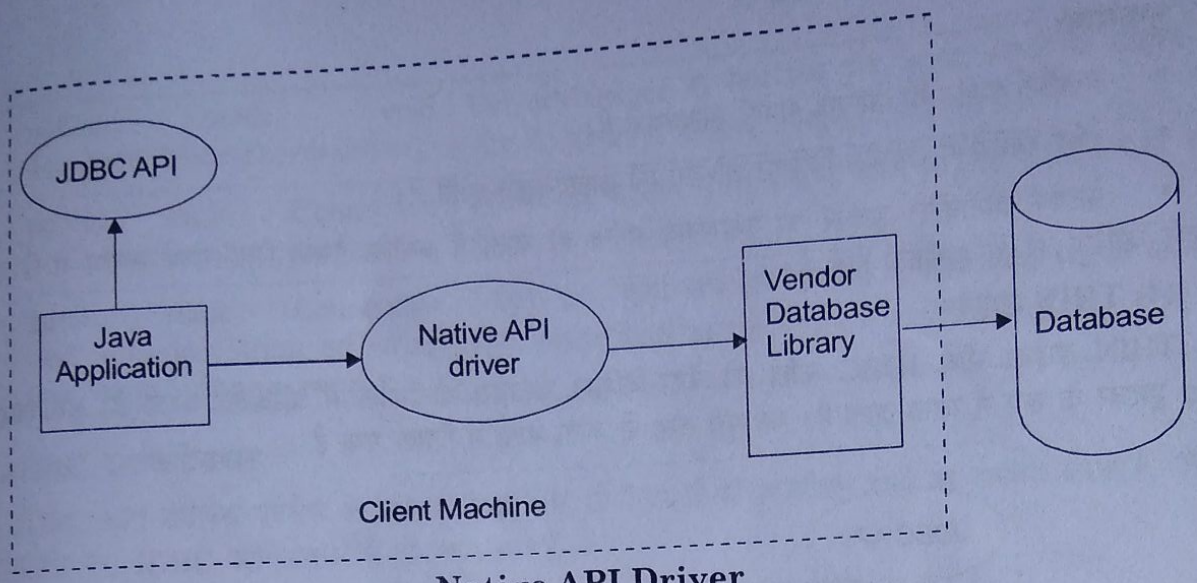
- प्रयोग करने में आसान।
- आसानी से किसी भी डेटाबेस से जुड़ा जा सकता है।

नुकसानः

- प्रदर्शन कम हो गया क्योंकि JDBC Method कॉल ODBC फ़ंक्शन कॉल में परिवर्तित हो गई है।
- ODBC ड्राइवर को क्लाइंट मशीन पर स्थापित करने की आवश्यकता है।

(2) NATIVE-API ड्राइवर

Native API ड्राइवर डेटाबेस के क्लाइंट-साइड लाइब्रेरीज़ का उपयोग करता है। ड्राइवर JDBC Method कॉल को डेटाबेस API के NATIVE कॉल में परिवर्तित करता है। यह पूरी तरह से जावा में नहीं लिखा जाता है।



Native API Driver

लाभः

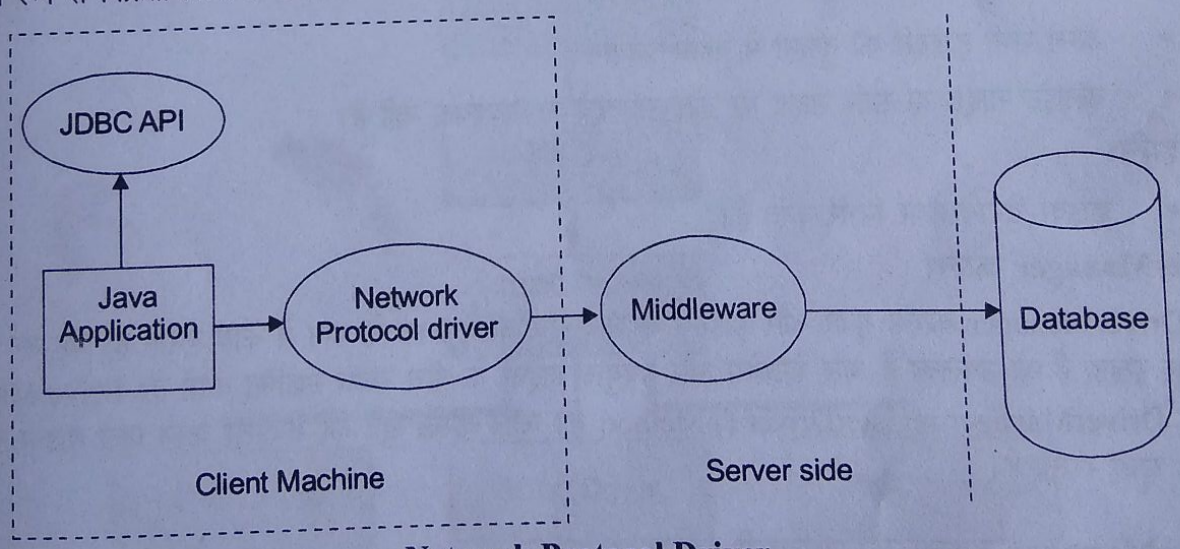
- प्रदर्शन JDBC-ODBC ब्रिज ड्राइवर की तुलना में उन्नत है।

हानिः

- प्रत्येक क्लाइंट मशीन पर NATIVE ड्राइवर को स्थापित करने की आवश्यकता है।
- क्लाइंट मशीन पर विक्रेता क्लाइंट लाइब्रेरी स्थापित करने की आवश्यकता है।

(3) नेटवर्क प्रोटोकॉल ड्राइवर

नेटवर्क प्रोटोकॉल ड्राइवर मिडिलवेयर (एप्लिकेशन सर्वर) का उपयोग करता है जो JDBC कॉल को प्रत्यक्ष या अप्रत्यक्ष रूप से विक्रेता-विशिष्ट डेटाबेस प्रोटोकॉल में परिवर्तित करता है। यह पूरी तरह से जावा में लिखा जाता है।



Network Protocol Driver

लाभः

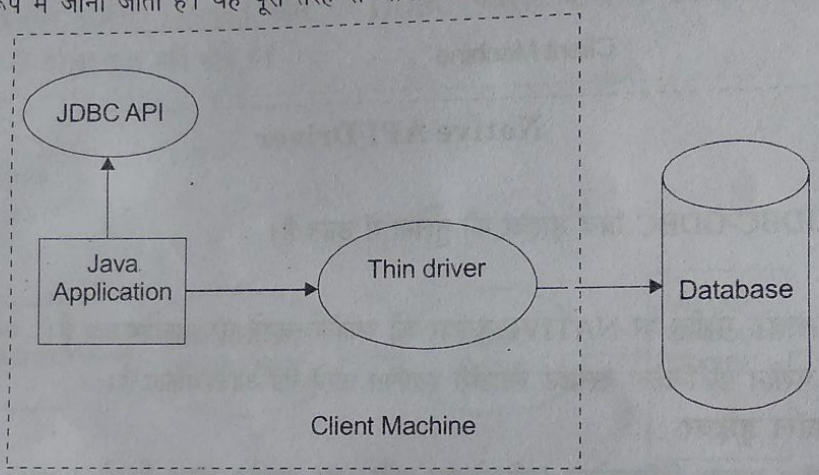
- एप्लिकेशन क्लाइंट के कारण किसी क्लाइंट साइड लाइब्रेरी की आवश्यकता नहीं होती है जो ऑडिटिंग, लोड बैलेंसिंग, लॉगिंग आदि जैसे कई कार्य कर सकती है।

नुकसानः

- क्लाइंट मशीन पर नेटवर्क सपोर्ट आवश्यक है।
- मध्य स्तरीय में डेटाबेस-विशिष्ट कोडिंग की आवश्यकता होती है।
- नेटवर्क प्रोटोकॉल ड्राइवर का रखरखाव महंगा हो जाता है क्योंकि इसके लिए मध्य स्तरीय में डेटाबेस-विशिष्ट कोडिंग की आवश्यकता होती है।

(4) THIN ड्राइवर

THIN ड्राइवर सीधे JDBC कॉल को वेंडर-विशिष्ट डेटाबेस प्रोटोकॉल में परिवर्तित करता है। इसीलिए इसे Thin ड्राइवर के रूप में जाना जाता है। यह पूरी तरह से जावा भाषा में लिखा गया है



Thin Driver

लाभः

- अन्य सभी ड्राइवरों की तुलना में बेहतर प्रदर्शन।
- क्लाइंट साइड या सर्वर साइड पर कोई सॉफ्टवेयर आवश्यक नहीं है।

हानिः

- ड्राइवर डेटाबेस पर निर्भर करते हैं।

DriverManager काल्स

DriverManager काल्स यूजर और ड्राइवरों के बीच एक अंतरफलक के रूप में कार्य करता है। यह उन ड्राइवरों पर नज़र रखता है जो उपलब्ध हैं और डेटाबेस और उपयुक्त ड्राइवर के बीच संबंध स्थापित करते हैं। DriverManager काल्स, DriverManager.registerDriver () Method को कॉल करके खुद को रजिस्टर करने वाले ड्राइवर काल्सेज की एक सूची रखता है।

DriverManager काल्स की उपयोगी METHODS

	METHOD	विवरण
1.	public static void registerDriver(Driver driver):	DriverManager के साथ दिए गए ड्राइवर को रजिस्टर करने के लिए उपयोग किया जाता है।

2.	public static void deregisterDriver(Driver driver):	DriverManager के साथ दिए गए ड्राइवर (सूची से ड्राइवर को छोड़ दें) को हटाने के लिए उपयोग किया जाता है।
3.	public static Connection getConnection(String url):	निर्दिष्ट url के साथ संबंध स्थापित करने के लिए उपयोग किया जाता है।
4.	public static Connection getConnection(String url,String userName,String password):	निर्दिष्ट url, यूजर नाम और पासवर्ड के साथ संबंध स्थापित करने के लिए उपयोग किया जाता है।

2.3 JDBC आर्किटेक्चर

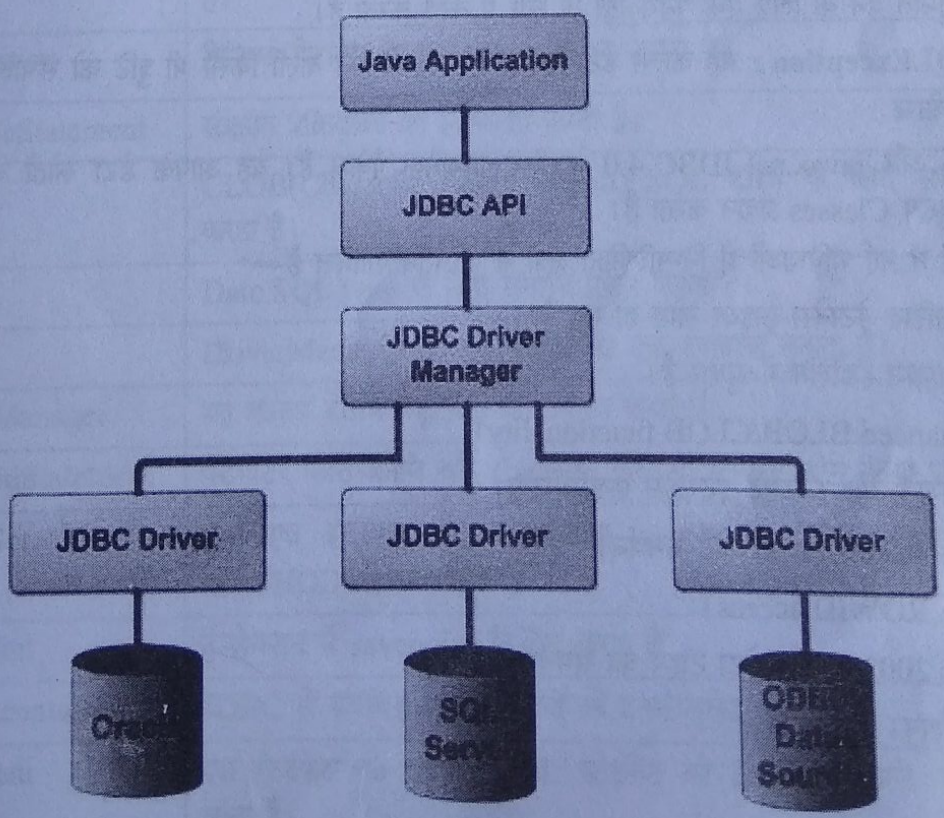
JDBC API डेटाबेस एक्सेस के लिए टू-टियर और थ्री-टियर प्रोसेसिंग मॉडल दोनों का समर्थन करता है, लेकिन सामान्य तौर पर, JDBC आर्किटेक्चर में दो लेयर होते हैं—

- **JDBC API :** यह application-to-JDBC Manager कनेक्शन प्रदान करता है।
- **JDBC ड्राइवर API :** यह JDBC Manager-to-Driver कनेक्शन का समर्थन करता है।

JDBC एपीआई विषम (hetrogenous) डेटाबेस के लिए पारदर्शी कनेक्टिविटी प्रदान करने के लिए एक ड्राइवर मैनेजर और डेटाबेस-विशिष्ट ड्राइवरों का उपयोग करता है।

JDBC ड्राइवर मैनेजर सुनिश्चित करता है कि प्रत्येक डेटा स्रोत तक पहुँचने के लिए सही ड्राइवर का उपयोग किया जाए। ड्राइवर मैनेजर कई विषम डेटाबेस से जुड़े कई समवर्ती ड्राइवरों का समर्थन करने में सक्षम है।

निम्नलिखित चित्र है, जो JDBC ड्राइवरों और जावा एप्लिकेशन के संबंध में ड्राइवर मैनेजर के स्थान को दर्शाता है—



सामान्य JDBC घटक

JDBC API निम्नलिखित इंटरफेस और Classes प्रदान करता है—

- **DriverManager :** यह काल्स डेटाबेस ड्राइवरों की सूची का प्रबंधन करता है। कम्युनिकेशन सब प्रोटोकॉल का उपयोग करके उचित डेटाबेस ड्राइवर के साथ जावा एप्लिकेशन से कनेक्शन अनुरोधों का मिलान करता है। पहले ड्राइवर जो जेडीबीसी के तहत एक निश्चित सबप्रोटोकॉल को पहचानता है, उसका उपयोग डेटाबेस कनेक्शन स्थापित करने के लिए किया जाएगा।
- **ड्राइवर :** यह इंटरफ़ेस डेटाबेस सर्वर के साथ संचार को संभालता है। आप ड्राइवर ऑब्जेक्ट्स के साथ सीधे बहुत कम बातचीत करेंगे। इसके बजाय, आप DriverManager ऑब्जेक्ट्स का उपयोग करते हैं, जो इस टाइप की ध्ँयिंऊए का प्रबंधन करता है। यह ड्राइवर ऑब्जेक्ट्स के साथ काम करने से जुड़े विवरणों को abstract भी करता है।
- **कनेक्शन :** एक डेटाबेस से संपर्क करने के लिए सभी Method के साथ यह इंटरफ़ेस है। कनेक्शन ऑब्जेक्ट संचार संदर्भ का प्रतिनिधित्व करता है, अर्थात, डेटाबेस के साथ सभी संचार केवल कनेक्शन ऑब्जेक्ट के माध्यम से होता है।
- **स्टेटमेंट (Statement) :** आप डेटाबेस से SQL स्टेटमेंट सबमिट करने के लिए इस इंटरफ़ेस से निर्मित OBJECTS का उपयोग करते हैं। कुछ व्युत्पन्न (derived) इंटरफेस संग्रहीत प्रोसेसेज को Execute करने के अलावा parameters को स्वीकार करते हैं।
- **RESULTSET :** स्टेटमेंट ऑब्जेक्ट्स का उपयोग करके SQL क्वेरी Execute करने के बाद ये ऑब्जेक्ट डेटाबेस से प्राप्त डेटा को धारण करते हैं। यह आपके डेटा के माध्यम से स्थानांतरित करने की अनुमति देने के लिए एक पुनरावृत्ति के रूप में कार्य करता है।
- **SQLException :** यह काल्स डेटाबेस एप्लीकेशन में होने वाली किसी भी त्रुटि को संभालता है।

JDBC 4.0 पैकेज

Java.sql और javax.sql JDBC 4.0 के लिए प्राथमिक पैकेज हैं। यह आपके डेटा स्रोतों के साथ बातचीत करने के लिए मुख्य Classes प्रदान करता है।

इन पैकेजों में नई सुविधाओं में निम्नलिखित क्षेत्रों में परिवर्तन शामिल हैं—

- स्वचालित डेटाबेस ड्राइवर लोड हो रहा है।
- एक्सेप्शन हैंडलिंग में सुधार है।
- Enhanced BLOB/CLOB functionality।
- कनेक्शन और स्टेटमेंट इंटरफ़ेस एन्हांसमेंट्स।
- National character सेट समर्थन।
- SQL ROWID access।
- SQL 2003 XML डेटा टाइप का समर्थन।
- एनोटेशन।

2.4 JDBC API

जावा डेटाबेस कनेक्टिविटी (JDBC) एपीआई जावा प्रोग्रामिंग भाषा से यूनिवर्सल डेटा एक्सेस प्रदान करता है। JDBC API का उपयोग करके, आप रिलेशनल डेटाबेस से स्प्रेडशीट तक virtually access कर सकते हैं। JDBC तकनीक एक सामान्य आधार भी प्रदान करती है जिसके आधार पर उपकरण और वैकल्पिक इंटरफेस बनाए जा सकते हैं।

JDBC API में दो पैकेज शामिल हैं—

- java.sql
- javax.sql

एक विशेष डेटाबेस प्रबंधन प्रणाली के साथ JDBC एपीआई का उपयोग करने के लिए, आपको JDBC तकनीक और डेटाबेस के बीच मध्यस्थता करने के लिए JDBC प्रौद्योगिकी-आधारित ड्राइवर की आवश्यकता होती है। विभिन्न कारकों के आधार पर, एक ड्राइवर को शुद्ध रूप से जावा प्रोग्रामिंग भाषा में या जावा प्रोग्रामिंग भाषा और जावा नेटिव इंटरफेस (जेएनआई) Native Method के मिश्रण में लिखा जा सकता है।

java.sql पैकेज

इस पैकेज में SQL क्वेरी बनाने और Execute करने जैसे लगभग सभी JDBC ऑपरेशन करने के लिए Classes और इंटरफ़ेस शामिल हैं।

java.sql पैकेज की महत्वपूर्ण classes और इंटरफेस

काल्सेज / इंटरफ़ेस	विवरण
java.sql.BLOB	BLOB(Binary Large Object) SQL टाइप के लिए समर्थन प्रदान करता है।
java.sql.Connection	विशिष्ट डेटाबेस के साथ एक कनेक्शन बनाता है।
java.sql.CallableStatement	संग्रहीत प्रक्रियाओं को निष्पादित करता है।
java.sql.CLOB	CLOB(Character Large Object) SQL टाइप के लिए समर्थन प्रदान करता है।
java.sql.Date	Date SQL type के लिए समर्थन प्रदान करता है।
java.sql.Driver	DriverManager के साथ ड्राइवर का एक उदाहरण बनाता है।
java.sql.DriverManager	यह काल्स डेटाबेस ड्राइवरों का प्रबंधन करता है।
java.sql.PreparedStatement	पैरामीटर क्वेरी बनाने और Execute करने के लिए उपयोग किया जाता है।
java.sql.ResultSet	यह एक इंटरफ़ेस है जो परिणाम row-by-row तक पहुंचने के लिए METHODS प्रदान करता है।
java.sql.Savepoint	ट्रेन्ज़ेक्शन में savepoint निर्दिष्ट करता है।
java.sql.SQLException	JDBC से संबंधित सभी अपवादों को इनकैप्सुलेट करता है।
java.sql.Statement	इस इंटरफ़ेस का उपयोग SQL स्टेटमेंट को Execute करने के लिए किया जाता है।

DatabaseMetaData	समग्र रूप से डेटाबेस के बारे में व्यापक जानकारी देता है।
DriverAction	जब कोई ड्राइवर DriverManager द्वारा अधिसूचित किया जाना चाहता है तो उसे लागू किया जाता है।
ResultSetMetaData	एक ऑब्जेक्ट जिसका उपयोग ResultSet ऑब्जेक्ट में कॉलम के टाइप और गुणों के बारे में जानकारी प्राप्त करने के लिए किया जाता है।
RowId	SQL ROWID मान के जावा प्रोग्रामिंग भाषा में प्रतिनिधित्व (मैपिंग) करना।
Savepoint	एक सेवपॉइंट का प्रतिनिधित्व, जो कि वर्तमान ट्रेन्ज़ेक्शन के भीतर का एक बिंदु है जिसे Connection.rollback METHOD से संदर्भित किया जा सकता है।
SQLData	जावा प्रोग्रामिंग भाषा में एक काल्स के लिए SQL User Defined Type (UDT) के कस्टम मैपिंग के लिए उपयोग किया जाने वाला इंटरफ़ेस।
SQLInput	एक इनपुट स्ट्रीम जिसमें SQL structred type या SQL diferent type की आवृत्ति का प्रतिनिधित्व करने वाले मानों की एक stream होती है।
SQLOutput	डेटाबेस में उपयोगकर्ता-परिभाषित प्रकार की विशेषताओं को लिखने के लिए आउटपुट स्ट्रीम होती है।
SQLType	एक ऐसी OBJECT जो JDBC टाइप या एक विक्रेता विशिष्ट डेटा टाइप नामक जेनेरिक SQL प्रकार की पहचान करने के लिए उपयोग की जाती है।
SQLXML	SQL XML type के लिए JavaTM प्रोग्रामिंग भाषा में मैपिंग।
Statement	स्टेटिक SQL स्टेटमेंट को Execute करने और इसके परिणाम उत्पन्न करने के लिए उपयोग की गई OBJECT।
Struct	SQL Structred type के लिए जावा प्रोग्रामिंग भाषा में Standard Mapping.
Wrapper	JDBC काल्सेज के लिए इंटरफ़ेस जो प्रतिनिधि उदाहरण को पुनः प्राप्त करने की क्षमता प्रदान करता है (जब प्रश्न में Instance एक प्रॉक्सी काल्स है)।

Javax.sql पैकेज

इस पैकेज को JDBC एक्सटेंशन API के रूप में भी जाना जाता है। यह सर्वर-साइड डेटा तक पहुंचने के लिए Classes और इंटरफ़ेस प्रदान करता है।

javax.sql पैकेज की महत्वपूर्ण Classes और इंटरफ़ेस

काल्सेज / इंटरफ़ेस	विवरण
javax.sql.ConnectionEvent	event के बारे में जानकारी प्रदान करता है।
javax.sql.ConnectionEventListener PooledConnection	ऑब्जेक्ट द्वारा उत्पन्न ईवेंट को पंजीकृत करने के लिए उपयोग किया जाता है।
javax.sql.DataSource	किसी एप्लीकेशन में उपयोग किए जाने वाले डेटा स्रोत इंटरफ़ेस का प्रतिनिधित्व करता है।

javax.sql.PooledConnection	कनेक्शन पूल manage करने के लिए ऑब्जेक्ट प्रदान करता है।
CommonDataSource	इंटरफ़ेस जो उन METHOD को परिभाषित करता है जो DataSource, XADataSource और ConnectionPoolDataSource के बीच आम हैं।
rowset	इंटरफ़ेस जो JavaBeans™ घटक मॉडल के लिए JDBC API को समर्थन करता है।
RowSetInternal	एक RowSet ऑब्जेक्ट जो इंटरफ़ेस RowSetReader या RowSetWriter ऑब्जेक्ट के लिए स्वयं को प्रस्तुत करने के लिए लागू करता है।
RowSetListener	एक इंटरफ़ेस जिसे एक घटक द्वारा कार्यान्वित किया जाना चाहिए जो कि सूचित करना चाहता है जब कोई महत्वपूर्ण event रोसेट ऑब्जेक्ट के जीवन में होती है।
RowSetMetaData	एक ऑब्जेक्ट जिसमें रॉसेट ऑब्जेक्ट में कॉलम के बारे में जानकारी होती है।
RowSetReader	Rowset OBJECT को कॉल करने वाली सुविधा डेटा की rows के साथ खुद को पॉप्युलेट करने के लिए कहती है।
RowSetWriter	एक OBJECT जो रोसेटसेट इंटरफेस को लागू करती है, इसलिए writer कहलाती है।
StatementEventListener	एक OBJECT जो स्टेटमेंट पूल में PREPARED की गई घटनाओं के बारे में सूचित करने के लिए पंजीकृत करती है।
XAConnection	एक OBJECT जो वितरित ट्रेन्ज़ेक्शन के लिए सहायता प्रदान करती है।
XADataSource	XAConnection OBJECTS के लिए एक factory जो आंतरिक रूप से उपयोग किया जाता है।

2.5 JDBC कनेक्शन स्थापित करना (Establishing JDBC Connection)

JDBC ODBC का एडवांसमेंट है, ODBC प्लेटफॉर्म पर निर्भर होने के कारण बहुत सारी कमियां थीं। ODBC API को C, C ++, Python, Core Java में लिखा गया था और प्लेटफार्म डिपेंडेंट हैं। इसलिए निर्भरता को दूर करने के लिए, JDBC को डेटाबेस विक्रेता द्वारा विकसित किया गया था जिसमें जावा में लिखी गई Classes और इंटरफेस शामिल थे।

जावा प्रोग्राम और डेटाबेस के बीच कनेक्टिविटी के लिए कदम

1. ड्राइवर लोड करना—आपको पहले प्रोग्राम में उपयोग करने से पहले ड्राइवर को लोड करना होगा या इसे रजिस्टर करना होगा। रजिस्ट्रेशन आपके प्रोग्राम में एक बार किया जाना है। आप नीचे बताए गए दो Method में से एक में ड्राइवर को रजिस्टर कर सकते हैं—

- **Class.forName () :** यहां हम रनटाइम पर ड्राइवर की काल्स फाइल को मेमोरी में लोड करते हैं। नए या ऑब्जेक्ट के निर्माण की आवश्यकता नहीं है। निम्न उदाहरण Oracle ड्राइवर को लोड करने के लिए Class.forName () का उपयोग करता है—

Class.forName ('oracle.jdbc.driver.OracleDriver');

- **DriverManager.registerDriver ():** DriverManager एक स्टेटिक सदस्य रजिस्टर के साथ एक जावा इनबिल्ट काल्स है। यहां हम ड्राइवर काल्स के कंस्ट्रक्टर को कंपाइल टाइम पर कॉल करते हैं। Oracle ड्राइवर को रजिस्टर करने के लिए निम्न उदाहरण DriverManager.registerDriver () का उपयोग करता है —

DriverManager.registerDriver (new oracle.jdbc.driver.OracleDriver ())

2. कनेक्शन बनाएँ

ड्राइवर को लोड करने के बाद, कनेक्शन का उपयोग करके स्थापित करें—

Connection con = DriverManager.getConnection(url,user,password)

Connection con = DriverManager.getConnection(url,user,password)

user – यूजर नाम जिससे आपका sql कमांड प्रॉम्प्ट एक्सेस किया जा सकता है।

password – वह पासवर्ड जिससे आपका sql कमांड प्रॉम्प्ट एक्सेस किया जा सकता है।

con: कनेक्शन इंटरफ़ेस का संदर्भ है।

url : Uniform Resource Locator। इसे निम्नानुसार बनाया जा सकता है—

Connection con = DriverManager.getConnection(url,user,password)

String url = ' jdbc:oracle:thin:@localhost:1521:xe'

जहाँ oracle डेटाबेस का उपयोग किया जाता है, Thin ड्राइवर का उपयोग किया जाता है, @localhost IP एड्रेस है जहाँ डेटाबेस संग्रहीत है, 1521 पोर्ट नंबर है और xe service प्रदाता है। उपरोक्त सभी 3 पैरामीटर स्ट्रिंग टाइप के हैं और फ़ंक्शन को कॉल करने से पहले प्रोग्रामर द्वारा घोषित किए जाने हैं। इसका उपयोग अंतिम कोड से संदर्भित किया जा सकता है।

3. एक स्टेटमेंट (STATEMENT) बनाएँ

एक बार कनेक्शन स्थापित हो जाने के बाद आप डेटाबेस के साथ बातचीत कर सकते हैं। JDBCStatement, CallableStatement और PreparedStatement इंटरफेस उन शब्द को परिभाषित करते हैं जो आपको SQL कमांड भेजने और अपने डेटाबेस से डेटा प्राप्त करने में सक्षम करते हैं। JDBC स्टेटमेंट का उपयोग इस प्रकार है—

Statement st = con.createStatement();

यहां, con पिछले स्टेप में उपयोग किए गए कनेक्शन इंटरफ़ेस का संदर्भ है।

4. क्वेरी Execute करना (Execute the query)

अब आता है सबसे महत्वपूर्ण हिस्सा यानी क्वेरी को Execute करना। यहाँ क्वेरी एक SQL क्वेरी है। अब हम जानते हैं कि हम कई प्रकार के querry हैं। उनमें से कुछ इस प्रकार हैं—

- डेटाबेस में तालिका को अपडेट/insert करने की क्वेरी।
- डेटा पुनर्प्राप्त करने की क्वेरी।

Statement इंटरफ़ेस का executeQuery() Method का उपयोग डेटाबेस से मान प्राप्त करने के प्रश्नों को Execute करने के लिए किया जाता है। executeUpdate(sql query) को Execute करने के querries को Execute करने के लिए निष्पादक इंटरफ़ेस का उपयोग किया जाता है।

उदाहरणः

```
int m = st.executeUpdate(sql);
if (m==1)
   System.out.println('inserted successfully : '+sql);
else
   System.out.println('insertion failed');
```

यहाँ sql/string के टाइप की sql केरी है।

5. कनेक्शन बंद करना (Close the connections)

अंत में हमने डेटा को निर्दिष्ट स्थान पर भेज दिया है और अब हम अपने काम के पूरा होने के कगार पर हैं। बंद होने वाले कनेक्शन, स्टेटमेंट और रिजल्टसेट के ऑब्जेक्ट अपने आप बंद हो जाएंगे। कनेक्शन को बंद करने के लिए कनेक्शन इंटरफ़ेस की close () method का उपयोग किया जाता है।

उदाहरण :

```
con.close ();
```

JDBC कनेक्शन जावा में स्थापित करना

कोडः

```
importjava.sql.*;
importjava.util.*;
class Main
{
    public static void main(String a[])
    {
        //Creating the connection
        String url = "jdbc:oracle:thin:@localhost:1521:xe";
        String user = "system";
        String pass = "12345";

        //Entering the data
        Scanner k = new Scanner(System.in);
        System.out.println("enter name");
        String name = k.next();
        System.out.println("enter roll no");
        int roll = k.nextInt();
        System.out.println("enter class");
        String cls = k.next();
```

```
//Inserting data using SQL query
String sql = "insert into student1 values('"+name+"','"+roll+"','"+cls+"')";
Connection con=null;
try
{
        DriverManager.registerDriver(new oracle.jdbc.OracleDriver());

        //Reference to connection interface
        con = DriverManager.getConnection(url,user,pass);

        Statement st = con.createStatement();
        int m = st.executeUpdate(sql);
        if (m == 1)
                System.out.println("inserted successfully : "+sql);
        else
                System.out.println("insertion failed");
        con.close();
}
catch(Exception ex)
{
        System.err.println(ex);
}
    }
}
```

आउटपुट:

```
C:\Windows\System32\cmd.exe
E:\>javac Main.java

E:\>java Main
enter name
Abc
enter roll no
14
enter class
6a
inserted successfully :  insert into student1 values('Abc','14','6a')

E:\>
```

2.6 JDBC API का उपयोग करके CURD ऑपरेशन (CURD operation Using JDBC API)

जब हम एपीआई का निर्माण कर रहे होते हैं, तो हम चाहते हैं कि हमारे मॉडल चार बुनियादी टाइप की कार्यक्षमता प्रदान करें। मॉडल को रिसोर्सेज Create, Read, Update और Delete करने के लिए सक्षम होना चाहिए। कम्प्यूटर वैज्ञानिक अक्सर इन कार्यों को संक्षिप्त CRUD द्वारा संदर्भित करते हैं। एक मॉडल में इन चार कार्यों को पूरा करने के लिए प्रदर्शन करने की क्षमता होनी चाहिए। यदि इन चार कार्यों में से किसी एक कार्रवाई का वर्णन नहीं किया जा सकता है, तो यह संभावित रूप से स्वयं का एक मॉडल होना चाहिए।

ये Basic संचालन INSERT, SELECT, UPDATE और DELETE statements SQL में हैं। यद्यपि लक्ष्य डेटाबेस सिस्टम ओरेकल डेटाबेस है, लेकिन समान तकनीकों को अन्य डेटाबेस सिस्टमों पर भी लागू किया जा सकता है क्योंकि क्वेरी सिंटैक्स का उपयोग मानक ऐंथ आमतौर पर सभी रिलेशनल डेटाबेस सिस्टम द्वारा समर्थित है।

Oracle डेटाबेस में एक यूजर बनाना और आवश्यक अनुमति देना:

1. Cmd का उपयोग करके ओरेकल खोलें। इसके लिए cmd में sqlplus टाइप करें और एंटर दबाएं।
2. पासवर्ड द्वारा सुरक्षित यूजर-आईडी बनाएं। इस यूजर-आईडी को चाइल्ड यूजर कहा जाता है।

create user identified by ;

3. चाइल्ड यूजर को आवश्यक अनुमति देना।

```
conn / as sysdba;
grant dba to ;
```

रिक्त फ़ील्ड के साथ एक नमूना तालिका बनों:

```
CREATE TABLE userid(
  id varchar2(30) NOT NULL PRIMARY KEY,
  pwd varchar2(30) NOT NULL,
  fullname varchar2(50),
  email varchar2(50)
);
```

प्रिंसिपल JDBC इंटरफेस और काल्सेस

आइए JDBC के मुख्य इंटरफेस और CLASSES पर एक नज़र डालें जो हम CURD को लागू करने के लिए उपयोग करेंगे। वे सभी java.sql पैकेज के तहत उपलब्ध हैं—

- **Class.forName () :** यहां हम रनटाइम पर ड्राइवर की काल्स फाइल को मेमोरी में लोड करते हैं। OBJECT के नए या निर्माण की आवश्यकता नहीं है।

Class.forName ('oracle.jdbc.driver.OracleDriver');

- **DriverManager** : इस class का उपयोग एक विशिष्ट डेटाबेस प्रकार के लिए ड्राइवर को रजिस्टर करने के लिए किया जाता है (जैसे कि इस ट्यूटोरियल में ओरेकल डेटाबेस) और इसके माध्यम से सर्वर के साथ डेटाबेस कनेक्शन स्थापित करने के लिए getConnection ()METHOD का यूज करता है।

- **कनेक्शन :** यह इंटरफ़ेस एक स्थापित डेटाबेस कनेक्शन (सेशन) का प्रतिनिधित्व करता है जिसमें से हम प्रश्नों को execute करने और परिणाम प्राप्त करने के लिए स्टेटमेंट बना सकते हैं, डेटाबेस के बारे में मेटाडेटा प्राप्त कर सकते हैं, कनेक्शन क्लोज कर सकते हैं, आदि।

```
Connection con = DriverManager.getConnection
('jdbc:oracle:thin:@localhost:1521:orcl', 'login1', 'pwd1');
```

- **स्टेटमेंट (STATEMENT) और Prepared Statement :** इन इंटरफेस का उपयोग स्टेटिक SQL क्वेरी और पैरामीटर किए गए SQL क्वेरी को Execute करने के लिए किया जाता है। Prepared स्टेटमेंट इंटरफ़ेस का सुपर इंटरफ़ेस Statement है। उनकी आमतौर पर इस्तेमाल की जाने वाली Methods हैं—

1. **boolean execute(String sql) :** एक सामान्य SQL स्टेटमेंट Execute करता है। यह True है या कुछ भी नहीं लौटाती है। इस Method का उपयोग केवल स्टेटमेंट के साथ किया जा सकता है।
2. **int executeUpdate(String sql) :** एक INSERT, UPDATE या DELETE स्टेटमेंट Execute करता है और एक अपडेट खाता लौटाता है जो प्रभावित Rows की संख्या को दर्शाता है (जैसे 1 row सम्मिलित, या 2 rows अपडेट, या 0 rows प्रभावित)।

```
Statement stmt = con.createStatement();
    String q1 = 'insert into userid values
    (" +id+ ", " +pwd+ ", " +fullname+ ", " +email+ ")';
    int x = stmt.executeUpdate(q1);
```

3. **ResultSet executeQuery (String sql) :** Select स्टेटमेंट Execute करता है और एक resultset object देता है जिसमें क्वेरी द्वारा लौटाए गए परिणाम होते हैं।

```
Statement stmt = con.createStatement();
    String q1 = 'select * from userid WHERE id = " + id + "
    AND pwd = " + pwd + "';
    ResultSet rs = stmt.executeQuery(q1);
```

- **RESULTSET :** Table डेटा Select क्वेरी द्वारा लौटाया गया है। next() Method का उपयोग करके RESULTSET में Rows पर पुनरावृति करने के लिए इस ऑब्जेक्ट का उपयोग करें।
- **SQLException :** यह जाँच किया गया एक्सेप्शन उपरोक्त सभी Methods द्वारा throw किया गया है, इसलिए हमें उपरोक्त CLASSES के Method को कॉल करते समय इस एक्सेप्शन को स्पष्ट रूप से catch होगा।

डेटाबेस से कनेक्ट करना

Oracle डाटाबेस सर्वर लोकलहोस्ट पर डिफ़ॉल्ट पोर्ट 1521 पर सुनता है। निम्न कोड यूजर login1 और पासवर्ड pwd1 द्वारा डेटाबेस नाम userid से जोड़ता है।

उदाहरण:

```
// Java program to illustrate
// Connecting to the Database
import java.sql.*;
```

```
public class connect
{
   public static void main(String args[])
   {
      try
      {
         Class.forName("oracle.jdbc.driver.OracleDriver");

         // Establishing Connection
         Connection con = DriverManager.getConnection(
          "jdbc:oracle:thin:@localhost:1521:orcl", "login1", "pwd1");

         if (con != null)
            System.out.println("Connected");
         else
            System.out.println("Not Connected");

         con.close();
      }
      catch(Exception e)
      {
         System.out.println(e);
      }
   }
```

आउटपुटः

Connected

इंसर्ट स्टेटमेंट लागू करना

```
// Java program to illustrate
// inserting to the Database
import java.sql.*;

public class insert1
{
   public static void main(String args[])
   {
      String id = "id1";
      String pwd = "pwd1";
      String fullname = "Asian Publishers";
```

```
        String email = "xyz@abc.org";

        try
        {
          Class.forName("oracle.jdbc.driver.OracleDriver");
          Connection con = DriverManager.getConnection("
           jdbc:oracle:thin:@localhost:1521:orcl", "login1", "pwd1");
          Statement stmt = con.createStatement();

          // Inserting data in database
          String q1 = "insert into userid values('" +id+ "', '" +pwd+
                      "', '" +fullname+ "', '" +email+ "')";
          int x = stmt.executeUpdate(q1);
          if (x > 0)
            System.out.println("Successfully Inserted");
          else
            System.out.println("Insert Failed");

          con.close();
        }
        catch(Exception e)
        {
          System.out.println(e);
        }
      }
    }
```

आउटपुट:

Successfully Registered

अपडेट स्टेटमेंट को लागू करना

```
// Java program to illustrate
// updating the Database
import java.sql.*;

public class update1
{
  public static void main(String args[])
  {
    String id = "id1";
```

```
        String pwd = "pwd1";
        String newPwd = "newpwd";
        try
        {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection con = DriverManager.getConnection("
             jdbc:oracle:thin:@localhost:1521:orcl", "login1", "pwd1");
            Statement stmt = con.createStatement();

            // Updating database
            String q1 = "UPDATE userid set pwd = '" + newPwd +
                  "' WHERE id = '" +id+ "' AND pwd = '" + pwd + "'";
            int x = stmt.executeUpdate(q1);

            if (x > 0)
                System.out.println("Password Successfully Updated");
            else
                System.out.println("ERROR OCCURED :(");

            con.close();
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
}
```

आउटपुटः

Password Successfully Updated

डिलीट स्टेटमेंट को लागू करना

```
// Java program to illustrate
// deleting from Database
import java.sql.*;

public class delete
{
    public static void main(String args[])
    {
        String id = "id2";
```

```
            String pwd = "pwd2";
            try
            {
                    Class.forName("oracle.jdbc.driver.OracleDriver");
                    Connection con = DriverManager.getConnection("
                    jdbc:oracle:thin:@localhost:1521:orcl", "login1", "pwd1");
                    Statement stmt = con.createStatement();

                    // Deleting from database
                    String q1 = "DELETE from userid WHERE id = '" + id +
                                                    "' AND pwd = '" + pwd + "'";

                    int x = stmt.executeUpdate(q1);

                    if (x > 0)
                            System.out.println("One User Successfully Deleted");
                    else
                            System.out.println("ERROR OCCURED :(");

                    con.close();
            }
            catch(Exception e)
            {
                    System.out.println(e);
            }
        }
    }
```

Output:

One User Successfully Deleted

Select Statement को लागू करना

```
// Java program to illustrate
// selecting from Database
import java.sql.*;

public class select
{
    public static void main(String args[])
    {
        String id = "id1";
        String pwd = "pwd1";
```

```
        try
        {
                Class.forName("oracle.jdbc.driver.OracleDriver");
                Connection con = DriverManager.getConnection("
                        jdbc:oracle:thin:@localhost:1521:orcl", "login1", "pwd1");
                Statement stmt = con.createStatement();

                // SELECT query
                String q1 = "select * from userid WHERE id = '" + id +

                ResultSet rs = stmt.executeQuery(q1);
                if (rs.next())
                {
                        System.out.println("User-Id : " + rs.getString(1));
                        System.out.println("Full Name :" + rs.getString(3));
                        System.out.println("E-mail :" + rs.getString(4));
                }
                else
                {
                        System.out.println("No such user id is already registered");
                }
                con.close();
        }
        catch(Exception e)
        {
                System.out.println(e);
        }
    }
}
```

आउटपुट:

User-Id: id1
Full Name: Asian Publishers
E-mail:xyz@abc.org

2.7 JDBC स्टेटमेंट (JDBC Statement)

JDBC स्टेटमेंट, CALLABLE स्टेटमेंट, और PREPARED स्टेटमेंट इंटरफेस उन Method और गुणों को परिभाषित करते हैं जो आपको SQL या PL/SQL कमांड भेजने और अपने डेटाबेस से डेटा प्राप्त करने में सक्षम करते हैं। वे उन Method को भी परिभाषित करते हैं जो डेटाबेस में उपयोग किए जाने वाले जावा और SQL डेटा प्रकारों के बीच डेटा टाइप के अंतर को मदद करते हैं।

एक बार कनेक्शन प्राप्त होने के बाद हम डेटाबेस के साथ बातचीत कर सकते हैं। व्ॅॅ स्टेटमेंट, CALLABL स्टेटमेंट, और PREPARED स्टेटमेंट इंटरफेस उन Method और गुणों को परिभाषित करते हैं जो आपको SQL PL/SQL कमांड भेजने और अपने डेटाबेस से डेटा प्राप्त करने में सक्षम करते हैं।

वे उन Method को भी परिभाषित करते हैं जो डेटाबेस में उपयोग किए जाने वाले जावा और SQL डेटा प्र के बीच डेटा टाइप के अंतर को मदद करते हैं।

निम्न तालिका इंटरफ़ेस का उपयोग करने के लिए तय करने के लिए प्रत्येक इंटरफ़ेस के उद्देश्य का सारांश प्र करती है।

इंटरफेस	अनुशंसित उपयोग
Statement	अपने डेटाबेस तक सामान्य-उद्देश्य पहुंच के लिए इसका उपयोग करें। उपयोगी है जब आप स्टेटि SQL स्टेटमेंट का उपयोग रनटाइम पर कर रहे हैं। स्टेटमेंट इंटरफ़ेस मापदंडों को स्वीकार नहीं क सकता है।
Prepared Statement	जब आप कई बार SQL स्टेटमेंट का उपयोग करने की योजना बनाते हैं तो इसका उपयोग क PREPARED स्टेटमेंट इंटरफ़ेस रनटाइम पर इनपुट parameters को स्वीकार करता है।
Callable Statement	जब आप डेटाबेस संग्रहीत Procedures का उपयोग करना चाहते हैं तो इसका उपयोग क CALLABLE स्टेटमेंट इंटरफ़ेस रनटाइम इनपुट मापदंडों को भी स्वीकार कर सकता है।

स्टेटमेंट OBJECTS

स्टेटमेंट ऑब्जेक्ट बनाना

किसी SQL स्टेटमेंट को Execute करने के लिए स्टेटमेंट ऑब्जेक्ट का उपयोग करने से पहले, आ कनेक्शन ऑब्जेक्ट के createStatement () Method का उपयोग करके एक बनाने की आवश्यकता है, जैसा निम्नलिखित उदाहरण में है—

```
Statement stmt = null;
try {
  stmt = conn.createStatement( );
  . . .
}
catch (SQLException e) {
  . . .
}
finally {
  . . .
}
```

एक बार जब आप स्टेटमेंट ऑब्जेक्ट बना लेते हैं, तो आप इसका उपयोग SQL स्टेटमेंट को उसके निष्पादन Methods में से एक के साथ Execute करने के लिए कर सकते हैं।

- **boolean execute (String SQL) :** यदि परिणामी OBJECT को पुनः प्राप्त किया जा सकता है True का बूलियन मान लौटाता है; अन्यथा, यह False है। SQL DDL स्टेटमेंट्स को Execute

के लिए या जब आपको वास्तव में डायनेमिक SQL का उपयोग करने की आवश्यकता हो, तो इस Method का उपयोग करें।

- **int executeUpdate (String SQL) :** SQL स्टेटमेंट के निष्पादन से प्रभावित Rows की संख्या लौटाता है। SQL कथनों को Execute करने के लिए इस मेथड का उपयोग करें, जिसके लिए आप कई Rows को प्रभावित होने की उम्मीद करते हैं—उदाहरण के लिए, INSERT, UPDATE, या DELETE स्टेटमेंट।
- **ResultSet executeQuery (String SQL) :** एक ResultSet object लौटाता है। जब आप result set का चयन करने की अपेक्षा करते हैं, तो इस मेथड का उपयोग करें, जैसा कि आप एक select स्टेटमेंट के साथ करेंगे।

Closing स्टेटमेंट OBJECT

जैसे आप डेटाबेस रिसोर्सेज को बचाने के लिए एक कनेक्शन ऑब्जेक्ट को बंद करते हैं, उसी कारण से आपको स्टेटमेंट ऑब्जेक्ट को भी बंद करना चाहिए।

close() Method के लिए एक सरल कॉल काम करेगा। यदि आप पहले कनेक्शन ऑब्जेक्ट को बंद करते हैं, तो यह स्टेटमेंट ऑब्जेक्ट को भी बंद कर देगा। हालाँकि, आपको हमेशा उचित क्लीन अप सुनिश्चित करने के लिए स्टेटमेंट ऑब्जेक्ट को स्पष्ट रूप से बंद करना चाहिए।

```
Statement stmt = null;
try {
  stmt = conn.createStatement( );
  . . .
}
catch (SQLException e) {
  . . .
}
finally {
  stmt.close();
}
```

Prepared स्टेटमेंट OBJECTS

Prepared स्टेटमेंट इंटरफ़ेस स्टेटमेंट इंटरफ़ेस का विस्तार करता है, जो आपको जेनेरिक स्टेटमेंट ऑब्जेक्ट पर कुछ लाभों के साथ कार्यक्षमता जोड़ता है।

Prepared स्टेटमेंट ऑब्जेक्ट बनाना

```
PreparedStatement pstmt = null;
try {
  String SQL = "Update Employees SET age = ? WHERE id = ?";
  pstmt = conn.prepareStatement(SQL);
  . . .
}
catch (SQLException e) {
  . . .
}
```

```
finally {
  . . .
}
```

JDBC में सभी parameters का प्रतिनिधित्व '?' द्वारा किया जाता है, जिसे पैरामीटर मार्कर के रूप में जाना जाता है। SQL स्टेटमेंट Execute करने से पहले आपको हर पैरामीटर के लिए मानों की आपूर्ति करनी चाहिए।

setXX() Methods parameters के मानों को बांधती हैं, जहां XXX इनपुट टाइप को बांधने की इच्छा वाले जावा डेटा टाइप का प्रतिनिधित्व करता है। यदि आप मूल्यों की आपूर्ति करना भूल जाते हैं, तो आपको एक SQLException प्राप्त होगी।

प्रत्येक पैरामीटर मार्कर को इसकी ordinal स्थिति से संदर्भित किया जाता है। पहला मार्कर स्थिति 1, अगली स्थिति 2 और इसके बाद का प्रतिनिधित्व करता है। यह Method जावा array indices से भिन्न होती है, जो 0 से शुरू होती है।

डेटाबेस के साथ बातचीत करने के लिए स्टेटमेंट ऑब्जेक्ट के सभी Methods (a) execute(), (b) executeQuery(), और (c) executeUpdate() भी PREPARED STATEMENT ऑब्जेक्ट के साथ काम करते हैं। हालाँकि, SQL स्टेटमेंट का उपयोग करने के लिए Methods को संशोधित किया जाता है जो parameters को इनपुट कर सकते हैं।

PREPARED STATEMENT OBJECT को बंद करना (Closing Prepared Statement Object)

जिस तरह आप एक स्टेटमेंट ऑब्जेक्ट को बंद करते हैं, उसी तरह से आपको PREPARED स्टेटमेंट ऑब्जेक्ट को भी बंद करना चाहिए।

close() Method के लिए एक सरल कॉल काम करेगा। यदि आप पहले कनेक्शन ऑब्जेक्ट को बंद करते हैं, तो यह PREPARED स्टेटमेंट ऑब्जेक्ट को भी बंद कर देगा। हालांकि, आपको हमेशा उचित क्लीन अप सुनिश्चित करने के लिए PREPARED स्टेटमेंट ऑब्जेक्ट को स्पष्ट रूप से बंद करना चाहिए।

```
PreparedStatement pstmt = null;
try {
  String SQL = "Update Employees SET age = ? WHERE id = ?";
  pstmt = conn.prepareStatement(SQL);
  . . .
}
catch (SQLException e) {
  . . .
}
finally {
  pstmt.close();
}
```

PREPARED Statement के फायदे निम्नलिखित हैं—

- स्टेटमेंट, PREPARED स्टेटमेंट तेजी से प्रदर्शन करते हैं।
- PREPARED स्टेटमेंट का उपयोग करके, हम उन्नत डेटाटिप्स जैसे कि BLOB, CLOB, OBJECT के लिए मान आसानी से PREPARED कर सकते हैं।

- PREPARED स्टेटमेंट मानों को सेट करने के लिए सेटर Method प्रदान करके उद्धरण में उद्धरण और अन्य विशेष वर्णों के उपयोग से बचा जाता है, और इस तरह यह SQL इंजेक्शन के हमलों से बच जाता है

PREPARED स्टेटमेंट की सीमाएँ निम्नलिखित हैं—

- चूंकि एक PREPARED स्टेटमेंट ऑब्जेक्ट एक बार में केवल एक SQL स्टेटमेंट का प्रतिनिधित्व करता है, हम केवल एक स्टेटमेंट को एक PREPARED स्टेटमेंट ऑब्जेक्ट द्वारा Execute कर सकते हैं।
- इंजेक्शन हमलों को रोकने के लिए यह एक स्थान धारक के लिए एक से अधिक मूल्य की अनुमति नहीं देता है।

CALLABLE स्टेटमेंट ऑब्जेक्ट

जिस तरह एक कनेक्शन ऑब्जेक्ट स्टेटमेंट और PREPARED स्टेटमेंट ऑब्जेक्ट बनाता है, वह CALLABLE स्टेटमेंट ऑब्जेक्ट भी बनाता है, जिसका उपयोग किसी डेटाबेस संग्रहीत Procedures को कॉल Execute करने के लिए किया जाएगा।

CALLABLE स्टेटमेंट ऑब्जेक्ट बनाना

मान लीजिए, आपको निम्नलिखित ओरेकल संग्रहीत प्रोसेस को Execute करने की आवश्यकता है—

```
CREATE OR REPLACE PROCEDURE getEmpName
  (EMP_ID IN NUMBER, EMP_FIRST OUT VARCHAR) AS
BEGIN
  SELECT first INTO EMP_FIRST
  FROM Employees
  WHERE ID = EMP_ID;

END;
```

ध्यान दें : ऊपर संग्रहीत प्रोसेस ओरेकल के लिए लिखी गई है, लेकिन हम MySQL डेटाबेस के साथ काम कर रहे हैं, तो आइए, हम MySQL के लिए एक ही संग्रहीत प्रोसेस लिखें, इसे EMP डेटाबेस में बनाने के लिए निम्नानुसार है—

```
DELIMITER $$

DROP PROCEDURE IF EXISTS `EMP`.`getEmpName` $$
CREATE PROCEDURE `EMP`.`getEmpName`
  (IN EMP_ID INT, OUT EMP_FIRST VARCHAR(255))
BEGIN
  SELECT first INTO EMP_FIRST
  FROM Employees
  WHERE ID = EMP_ID;
END $$

DELIMITER ;
```

तीन टाइप के पैरामीटर मौजूद हैं—IN, OUT, और INOUT~ PREPARED स्टेटमेंट ऑब्जेक्ट केवल IN पैरामीटर का उपयोग करता है। CALLABLE स्टेटमेंट ऑब्जेक्ट तीनों का उपयोग कर सकता है;

यहां प्रत्येक की परिभाषा दी गई है—

पैरामीटर	विवरण
IN	एक पैरामीटर जिसका मान अज्ञात है जब SQL स्टेटमेंट बनाया जाता है। आप मानों को setXXX() METHODS के साथ IN पैरामीटर में bind करते हैं।
OUT	एक पैरामीटर जिसका मान एसक्यूएल स्टेटमेंट द्वारा आपूर्ति किया जाता है। आप getXXX() METHODS के साथ पैरामीटर को मानों से पुनर्प्राप्त करते हैं।
INOUT	एक पैरामीटर जो इनपुट और आउटपुट दोनों मान प्रदान करता है। setXXX() METHODS के साथ चर को बांधते हैं और getXXX() METHODS के साथ मूल्यों को पुनः प्राप्त करते हैं।

निम्न कोड स्निपेट दिखाता है कि कैसे नियोजित करना है। पहले से संग्रहीत प्रक्रिया के आधार पर CallableStatement ऑब्जेक्ट को तुरंत कनेक्ट करने के लिए Connection.prepareCall () METHOD है—

```
CallableStatement cstmt = null;
try {
  String SQL = "{call getEmpName (?, ?)}";
  cstmt = conn.prepareCall (SQL);
  . . .
}
catch (SQLException e) {
  . . .
}
finally {
  . . .

}
```

CALLABLE स्टेटमेंट ऑब्जेक्ट का उपयोग करना बहुत हद तक PREPARED स्टेटमेंट ऑब्जेक्ट का उपयोग करने जैसा है। आपको स्टेटमेंट execute करने से पहले सभी parameters पर मानों को बांधना चाहिए, या आपको एक SQL एक्सेप्शन प्राप्त होगा।

यदि आपके पास IN पैरामीटर हैं, तो बस उन्हीं नियमों और तकनीकों का पालन करें जो एक PREPARED स्टेटमेंट ऑब्जेक्ट पर लागू होती हैं; setXXX() METHOD का उपयोग करें जो जावा डेटा टाइप से मेल खाती है जिन्हें आप bind करते हैं।

जब आप OUT और INOUT पैरामीटर का उपयोग करते हैं, तो आपको एक अतिरिक्त CALLABLE स्टेटमेंट METHOD, registerOutParameter () को नियुक्त करना होगा। RegisterOutParameter ()

METHOD JDBC डेटा टाइप को उस डेटा टाइप से bind करती है, जिसमें संग्रहीत Procedure वापस आने की उम्मीद है।

एक बार जब आप अपनी संग्रहीत Procedure को कॉल करते हैं, तो आप OUT पैरामीटर से उचित getXXX () METHOD से मान प्राप्त करते हैं। यह METHOD SQL प्रकार के पुनः प्राप्त मान को जावा डेटा टाइप में डालती है।

कॉलेबल स्टेटमेंट OBJECT को बंद करना (Closing Callable Statement Object)

जिस तरह आप अन्य स्टेटमेंट ऑब्जेक्ट को बंद करते हैं, उसी तरह आपको CALLABLE स्टेटमेंट ऑब्जेक्ट को भी बंद करना चाहिए।

close() METHOD के लिए एक सरल कॉल काम करेगा। यदि आप पहले कनेक्शन ऑब्जेक्ट को बंद करते हैं, तो यह कॉलेबल स्टेटमेंट ऑब्जेक्ट को भी बंद कर देगा। हालाँकि, आपको उचित clean up सुनिश्चित करने के लिए हमेशा CALLABLE स्टेटमेंट को स्पष्ट रूप से बंद करना चाहिए।

```
CallableStatement cstmt = null;
try {
  String SQL = "{call getEmpName (?, ?)}";
  cstmt = conn.prepareCall (SQL);
  . . .
}
catch (SQLException e) {
  . . .
}
finally {
  cstmt.close();
}
```

2.8 RESULTSET

SQL स्टेटमेंट जो डेटाबेस क्वेरी से डेटा पढ़ते हैं, वो डेटा RESULTSET में वापस करते हैं। SELECT स्टेटमेंट एक डेटाबेस से Rows को चुनने और उन्हें RESULTSET में देखने का मानक शूप्द् है। Java.sql.ResultSet इंटरफ़ेस डेटाबेस क्वेरी के ऍळथ्ऊएऊं का प्रतिनिधित्व करता है।

ResultSet ऑब्जेक्ट एक कर्सर रखता है जो RESULTSET में वर्तमान Row को इंगित करता है। 'RESULTSET' शब्द एक परिणामी OBJECT में निहित Row और स्तंभ डेटा को संदर्भित करता है।

ResultSet इंटरफ़ेस के शूप्द् को तीन श्रेणियों में तोड़ा जा सकता है—

- **Navigational methods :** कर्सर को इधर-उधर करने के लिए उपयोग किया जाता है।
- **Get methods :** कर्सर द्वारा इंगित की जा रही वर्तमान Row के कॉलम में डेटा देखने के लिए उपयोग किया जाता है।
- **Update methods :** वर्तमान Row के कॉलम में डेटा को अपडेट करने के लिए उपयोग किया जाता है। फिर अपडेट अंतर्निहित डेटाबेस में भी अपडेट किया जा सकता है।

ResultSet के गुणों के आधार पर कर्सर मूवेबल है। ये गुण तब निर्दिष्ट किए जाते हैं जब संबंधित विवरण जो ResultSet उत्पन्न करता है ।

JDBC desired रिजल्टसेट के साथ स्टेटमेंट बनाने के लिए निम्नलिखित कनेक्शन Method प्रदान करता है—

- **createStatement(int RSType, int RSConcurrency);**
- **prepareStatement(String SQL, int RSType, int RSConcurrency);**
- **prepareCall(String sql, int RSType, int RSConcurrency);**

पहला तर्क रिजल्टसेट ऑब्जेक्ट के टाइप को इंगित करता है और दूसरा तर्क दो रिजल्टसेट constants में से एक है जो ये निर्दिष्ट करता है की क्या RESULTSET केवल पढ़ने योग्य है या अपडेट करने योग्य है।

Result set का प्रकार

संभावित RSType नीचे दिए गए हैं। यदि आप किसी भी परिणाम प्रकार को निर्दिष्ट नहीं करते हैं, तो आपको स्वचालित रूप से वह मिलेगा जो TYPE_FORWARD_ONLY है।

प्रकार	विवरण
ResultSet.TYPE_FORWARD_ONLY	RESULTSET में कर्सर केवल आगे बढ़ सकता है।
ResultSet.TYPE_SCROLL_INSENSITIVE	कर्सर आगे और पीछे स्क्रॉल कर सकता है, और RESULTSET दूसरों के द्वारा डेटाबेस में किए गए परिवर्तनों के प्रति संवेदनशील नहीं है, जो RESULTSET के बाद उत्पन्न होता है।
ResultSet.TYPE_SCROLL_SENSITIVE	कर्सर आगे और पीछे की ओर स्क्रॉल कर सकता है, और RESULTSET दूसरों के द्वारा डेटाबेस में किए गए परिवर्तनों के प्रति संवेदनशील है जो RESULTSET के बाद उत्पन्न होता है।

ResultSet की संगामिति (Concurrency of ResultSet)

संभव RSConcurrency नीचे दिया गया है। यदि आप किसी भी संगामिति प्रकार को निर्दिष्ट नहीं करते हैं, तो आप स्वचालित रूप से एक प्राप्त करेंगे जो CONCUR_READ_ONLY है।

संगामिति	विवरण
ResultSet.CONCUR_READ_ONLY	केवल-पढ़ने के लिए सेट परिणाम बनाता है। यह डिफ़ॉल्ट है
ResultSet.CONCUR_UPDATABLE	Updateकरने योग्य RESULTSET बनाता है।

उदाहरण:

```
try {
    Statement stmt = conn.createStatement(
                    ResultSet.TYPE_FORWARD_ONLY,
                    ResultSet.CONCUR_READ_ONLY);
}
```

```
catch(Exception ex) {
    ....
}
finally {
    ....
}
```

रिजल्ट सेट को नेविगेट करना

ResultSet इंटरफ़ेस में कई ईऊिंष्ध्अए हैं जिसमें कर्सर को शामिल करना शामिल है, जिसमें शामिल हैं—

एस.एन.	METHODS और विवरण
1.	**public void beforeFirst() throws SQLException** पहली row के ठीक पहले कर्सर ले जाता है।
2.	**public void afterLast() throws SQLException** अंतिम row के ठीक बाद कर्सर ले जाता है।
3.	**public boolean first() throws SQLException** कर्सर को पहली row में ले जाता है।
4.	**public void last() throws SQLException** कर्सर को अंतिम row में ले जाता है।
5.	**public boolean absolute(int row) throws SQLException** कर्सर को निर्दिष्ट row में ले जाता है।
6.	**public boolean relative(int row) throws SQLException** कर्सर को आगे या पीछे की rows की संख्या को ले जाता है, जहाँ से वह वर्तमान में इंगित कर रहा है।
7.	**public boolean previous() throws SQLException** कर्सर को पिछली row में ले जाता है। यदि पिछली row resultset बंद है तो यह method गलत है।
8.	**public boolean next() throws SQLException** कर्सर को अगली row में ले जाता है। यदि resultset में कोई और rows नहीं हैं, तो यह method गलत है।
9.	**public int getRow() throws SQLException** कर्सर को इंगित करने वाली row संख्या लौटाता है।
10.	**public void moveToInsertRow() throws SQLException** RESULTSET में कर्सर को एक विशेष row में ले जाता है जिसका उपयोग डेटाबेस में एक नई row सम्मिलित करने के लिए किया जा सकता है। वर्तमान कर्सर स्थान को याद करता है।
11.	**public void moveToCurrentRow() throws SQLException** यदि कर्सर वर्तमान में सम्मिलित row में है तो कर्सर को वर्तमान row में वापस ले जाता है; अन्यथा, यह method कुछ भी नहीं करता है।

RESULTSET देखना (Viewing a Result Set)

ResultSet इंटरफ़ेस में वर्तमान row का डेटा प्राप्त करने के दर्जनों METHODS हैं।

प्रत्येक संभव डेटा प्रकारों के लिए एक GET METHOD है, और प्रत्येक प्राप्त GET METHOD के दो संस्करण हैं—

- एक जो कॉलम नाम में लेता है।
- एक जो एक कॉलम इंडेक्स में लेता है।

उदाहरण के लिए, यदि आप जिस कॉलम को देखने में रुचि रखते हैं, उसमें int शामिल है, तो आपको रिजल्टसेट के getInt () METHOD का उपयोग करने की आवश्यकता है—

S.No.	METHODS और विवरण
1.	**public int getInt(String columnName) throws SQLException** कॉलम नाम वाले कॉलम में वर्तमान row में int लौटाता है।
2.	**public int getInt(int columnIndex) throws SQLException** निर्दिष्ट कॉलम इंडेक्स में वर्तमान row में int को लौटाता है। कॉलम इंडेक्स 1 से शुरू होता है, जिसका अर्थ है कि एक row का पहला कॉलम 1 है, एक row का दूसरा कॉलम 2 है।

इसी तरह, आठ जावा primitive data type में से प्रत्येक के लिए रिजल्टसेट इंटरफ़ेस में METHODS प्राप्त होती हैं जैसे कि java.lang.String, java.lang.Object और java.net.URL।

SQL डेटा टाइप java.sql.Date, java.sql.Time, java.sql.TimeStamp, java.sql.Clob, और java.sql.Blob प्राप्त करने की METHODS भी हैं। इन SQL डेटा प्रकारों का उपयोग करने के बारे में अधिक जानकारी के लिए दस्तावेज़ की जाँच करें।

रिजल्ट सेट को अपडेट करना

RESULTSET के डेटा को अपडेट करने के लिए ResultSet इंटरफ़ेस में अपडेट METHODS का एक संग्रह है।

GET METHOD के साथ, प्रत्येक डेटा प्रकार के लिए दो अपडेट METHODS हैं—

- एक जो कॉलम नाम में लेता है।
- एक जो एक कॉलम इंडेक्स में लेता है।

उदाहरण के लिए, किसी RESULTSET की वर्तमान row के स्ट्रिंग कॉलम को अपडेट करने के लिए, आप निम्न अपडेट में से एक का उपयोग करेंगे।

S.No.	METHODS और विवरण
1.	**public void updateString(int columnIndex, String s) throws SQLException** स्ट्रिंग को एस के मान में निर्दिष्ट कॉलम में बदलता है।
2.	**public void updateString(String columnName, String s) throws SQLException** पिछले method के समान, सिवाय इसके कि colum को इसके Index के बजाय इसके नाम से निर्दिष्ट किया गया है।

Java.sql पैकेज में आठ primitive data type के साथ-साथ स्ट्रिंग, ऑब्जेक्ट, URL और SQL डेटा प्रकारों के लिए अपडेट METHODS हैं।

RESULTSET में एक row को अपडेट करने से ResultSet ऑब्जेक्ट में वर्तमान row के कॉलम बदल जाते हैं, लेकिन अंतर्निहित डेटाबेस में नहीं। डेटाबेस में row में अपने परिवर्तनों को अपडेट करने के लिए, आपको निम्न METHODS में से एक को लागू करना होगा।

S.No.	METHODS और विवरण
1.	**public void updateRow()** डेटाबेस में संबंधित row को अपडेट करके वर्तमान पंक्ति को अपडेट करता है।
2.	**public void deleteRow()** डेटाबेस से वर्तमान row हटाता है।
3.	**public void refreshRow()** डेटाबेस में किसी भी हाल के परिवर्तनों को प्रतिबिंबित करने के लिए सेट किए गए परिणाम में डेटा को Refresh करता है।
4.	**public void cancelRowUpdates()** वर्तमान row पर किए गए किसी भी update को रद्द कर देता है।
5.	**public void insertRow()** डेटाबेस में एक row सम्मिलित करता है। यह METHOD केवल तब लागू की जा सकती है जब कर्सर सम्मिलित row की ओर इशारा कर रहा हो।

2.9 JDBC डेटा प्रकार

JDBC ड्राइवर डेटाबेस में भेजने से पहले जावा डेटा प्रकार को उपयुक्त JDBC टाइप में परिवर्तित करता है। यह अधिकांश डेटा प्रकारों के लिए एक डिफ़ॉल्ट मैपिंग का उपयोग करता है। उदाहरण के लिए, जावा इंट को SQL INTEGER में बदल दिया जाता है। ड्राइवरों के बीच consistency प्रदान करने के लिए डिफ़ॉल्ट मैपिंग बनाई गई थी।

निम्न तालिका डिफ़ॉल्ट JDBC डेटा प्रकार को बताती है जिसे जावा डेटा टाइप में कनवर्ट किया जाता है, जब आप SetXXX () METHOD की तैयारी या callable ऑब्जेक्ट या ResultSet.updatexX () METHOD को कॉल करते हैं।

SQL	JDBC/Java	setXXX	updateXXX
VARCHAR	java.lang.String	setString	updateString
CHAR	java.lang.String	setString	updateString
LONGVARCHAR	java.lang.String	setString	updateString
BIT	boolean	setBoolean	updateBoolean
NUMERIC	java.math.BigDecimal	setBigDecimal	updateBigDecimal
TINYINT	byte	setByte	updateByte

SMALLINT	short	setShort	updateShort
INTEGER	int	setInt	updateInt
BIGINT	long	setLong	updateLong
REAL	float	setFloat	updateFloat
FLOAT	float	setFloat	updateFloat
DOUBLE	double	setDouble	updateDouble
VARBINARY	byte[ ]	setBytes	updateBytes
BINARY	byte[ ]	setBytes	updateBytes
DATE	java.sql.Date	setDate	updateDate
TIME	java.sql.Time	setTime	updateTime
TIMESTAMP	java.sql.Timestamp	setTimestamp	updateTimestamp
CLOB	java.sql.Clob	setClob	updateClob
BLOB	java.sql.Blob	setBlob	updateBlob
ARRAY	java.sql.Array	setARRAY	updateARRAY
REF	java.sql.Ref	SetRef	updateRef
STRUCT	java.sql.Struct	SetStruct	updateStruct

JDBC 3.0 ने BLOB, CLOB, ARRAY और REF डेटा प्रकारों के लिए समर्थन बढ़ाया है। ResultSet ऑब्जेक्ट में अब updateBLOB (), updateCLOB (), updateArray (), Deewj updateRef () METHODS हैं जो आपको सर्वर पर संबंधित डेटा को main pulato करने में सक्षम बनाते हैं।

SetXXX() और UpdateXXX() METHODS आपको विशिष्ट जावा प्रकार को विशिष्ट जेडीबीसी डेटा टाइप में बदलने में सक्षम बनाती हैं। METHOD, setObject () और updateObject (), आपको JDBC डेटा प्रकार के लगभग किसी भी जावा टाइप को मैप करने में सक्षम करते हैं।

ResultSet ऑब्जेक्ट column मान को प्राप्त करने के लिए प्रत्येक डेटा प्रकार के लिए इसी getXXX () METHOD प्रदान करता है। प्रत्येक METHOD का उपयोग column नाम के साथ या इसकी क्रमिक स्थिति के द्वारा किया जा सकता है।

दिनांक और समय डेटा प्रकार

Java.sql.Date class SQL DATE type को map करता है और java.sql.Time और java.sql.Timetamp Classes SQL TIME and SQL Timestamp data type को Map करता है।

निम्न उदाहरण दिखाता है कि दिनांक और समय Classes SQL डेटा टाइप की आवश्यकताओं से मेल खाने के लिए मानक जावा तिथि और समय मानों को कैसे प्रारूपित करती हैं।

```
import java.sql.Date;
import java.sql.Time;
import java.sql.Timestamp;
import java.util.*;
```

```
public class SqlDateTime {
  public static void main(String[] args) {
    //Get standard date and time
    java.util.Date javaDate = new java.util.Date();
    long javaTime = javaDate.getTime();
    System.out.println("The Java Date is:" +
        javaDate.toString());

    //Get and display SQL DATE
    java.sql.Date sqlDate = new java.sql.Date(javaTime);
    System.out.println("The SQL DATE is: " +
        sqlDate.toString());

    //Get and display SQL TIME
    java.sql.Time sqlTime = new java.sql.Time(javaTime);
    System.out.println("The SQL TIME is: " +
        sqlTime.toString());
    //Get and display SQL TIMESTAMP
    java.sql.Timestamp sqlTimestamp =
    new java.sql.Timestamp(javaTime);
    System.out.println("The SQL TIMESTAMP is: " +
        sqlTimestamp.toString());
  }//end main
}//end SqlDateTime
```

अब हम उपरोक्त उदाहरण को संकलित करते हैं -

```
C:\>javac SqlDateTime.java
C:\>
```

जब आप JDBCExample चलाते हैं, तो यह निम्नलिखित परिणाम उत्पन्न करता है -

```
C:\>java SqlDateTime
The Java Date is:Tue Aug 18 13:46:02 GMT+04:00 2009
The SQL DATE is: 2009-08-18
The SQL TIME is: 13:46:02
The SQL TIMESTAMP is: 2009-08-18 13:46:02.828
C:\>
```

NULL मूल्यों को संभालना (Handling NULL Values)

SQL का NULL मानों का उपयोग और जावा का null का उपयोग विभिन्न अवधारणों हैं। तो, जावा में SQL NULL मान को संभालने के लिए, तीन रणनीति हैं जिनका आप उपयोग कर सकते हैं—

- getXXX( ) METHODS का उपयोग करने से बचें जो primitive डेटा प्रकारों को return करते हैं।

- Primitive डेटा प्रकारों के लिए wrapper काल्स का उपयोग करें, और यह जांचने के लिए ResultSet ऑब्जेक्ट का wasNull () METHOD का उपयोग करें कि क्या getXXX () METHOD द्वारा लौटाया गया मान प्राप्त करने वाले wrapper काल्स चर को Null पर सेट किया जाना चाहिए।
- Primitive डेटा प्रकार और रिजल्टसेट ऑब्जेक्ट की wasNull () METHOD का परीक्षण करने के कि क्या getXXX( ) METHOD द्वारा लौटाए गए मूल्य प्राप्त करने वाले Primitive चर को सेट मूल्य पर सेट किया जाना चाहिए जिसे आपने NULL का प्रतिनिधित्व करने के लिए चुना है।

एक NULL मान को संभालने के लिए यहां एक उदाहरण है—

```
Statement stmt = conn.createStatement( );
String sql = "SELECT id, first, last, age FROM Employees";
ResultSet rs = stmt.executeQuery(sql);

int id = rs.getInt(1);
if( rs.wasNull( ) ) {
  id = 0;
}
```

2.10 JDBC-एक्सेप्शन हैंडलिंग (JDBC-Exceptions Handling)

एक्सेप्शन हैंडलिंग आपको नियंत्रित स्थिति में प्रोग्राम-परिभाषित त्रुटियों जैसी असाधारण स्थितियों को संभाल अनुमति देता है।

जब एक एक्सेप्शन स्थिति होती है, तो एक एक्सेप्शन फेंक दिया जाता है। Thrown का मतलब है कि प्रोग्राम का निष्पादन बंद हो जाता है, और नियंत्रण को निकटतम लागू catch क्लॉज पर पुनर्निर्देशित किया जा यदि कोई लागू catch क्लॉज मौजूद नहीं है, तो प्रोग्राम का निष्पादन समाप्त हो जाता है।

JDBC एक्सेप्शन हैंडलिंग जावा एक्सेप्शन हैंडलिंग के समान है, लेकिन JDBC के लिए, आपके स सबसे सामान्य एक्सेप्शन है, वह java.sql.SQLException है।

SQLException METHODS

ड्राइवर और डेटाबेस दोनों में एक SQLException हो सकती है। जब ऐसा एक्सेप्शन होता है SQLException टाइप की एक OBJECT को पकड़ने के लिए पारित किया जाएगा।

पारित SQLException ऑब्जेक्ट में एक्सेप्शन के बारे में अतिरिक्त जानकारी प्राप्त करने के लिए निम्नलि Methods उपलब्ध हैं -

METHOD	विवरण
getErrorCode ()	एक्सेप्शन के साथ संबद्ध त्रुटि संख्या (associated error number) होती है।
getMessage ()	एक त्रुटि के लिए JDBC ड्राइवर के त्रुटि संदेश को प्राप्त करता है, द्वारा नियंत्रित किया जाता है या डेटाबेस त्रुटि के लिए Oracle त्रुटि और संदेश प्राप्त करता है।

getSQLState ()	XOPEN SQLstate स्ट्रिंग प्राप्त करता है। JDBC ड्राइवर त्रुटि के लिए, इस METHOD से कोई उपयोगी जानकारी नहीं दी जाती है। डेटाबेस त्रुटि के लिए, पांच अंकों वाला XOPEN SQLstate कोड वापस आ जाता है। यह METHOD Null return कर सकता है।
getNextException ()	एक्सेप्शन Chain में अगले एक्सेप्शन OBJECT प्राप्त करता है।
printStackTrace( )	वर्तमान एक्सेप्शन या Throwable प्रिंट करता है, और यह मानक त्रुटि स्ट्रीम के लिए बैकट्रेस है।
printStackTrace(PrintStream s)	आपके द्वारा निर्दिष्ट प्रिंट स्ट्रीम के लिए Throwable और इसके बैकट्रेस को प्रिंट करता है।
printStackTrace(PrintWriter w)	यह Throwable प्रिंट करता है और यह आपके द्वारा निर्दिष्ट प्रिंट लेखक के लिए बैकट्रेस है।

एक्सेप्शन ऑब्जेक्ट से उपलब्ध जानकारी का उपयोग करके, आप एक एक्सेप्शन को पकड़ सकते हैं और अपने प्रोग्राम को उचित रूप से जारी रख सकते हैं। यहाँ एक TRY ब्लॉक का सामान्य रूप है—

```
try {
  // Your risky code goes between these curly braces!!!
}
catch(Exception ex) {
  // Your exception handling code goes between these
  // curly braces, similar to the exception clause
  // in a PL/SQL block.
}
finally {
  // Your must-always-be-executed code goes between these
  // curly braces. Like closing database connection.
}
```

उदाहरण

try....catch...finally के उपयोग को समझने के लिए निम्न उदाहरण कोड का अध्ययन करें ।

```
//STEP 1. Import required packages
import java.sql.*;

public class JDBCExample {
  // JDBC driver name and database URL
  static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
  static final String DB_URL = "jdbc:mysql://localhost/EMP";

  // Database credentials
  static final String USER = "username";
  static final String PASS = "password";
```

```
public static void main(String[] args) {
Connection conn = null;
try{
  //STEP 2: Register JDBC driver
  Class.forName("com.mysql.jdbc.Driver");

  //STEP 3: Open a connection
  System.out.println("Connecting to database...");
  conn = DriverManager.getConnection(DB_URL,USER,PASS);

  //STEP 4: Execute a query
  System.out.println("Creating statement...");
  Statement stmt = conn.createStatement();
  String sql;
  sql = "SELECT id, first, last, age FROM Employees";
  ResultSet rs = stmt.executeQuery(sql);

  //STEP 5: Extract data from result set
  while(rs.next(){
    //Retrieve by column name
    int id  = rs.getInt("id");
    int age = rs.getInt("age");
    String first = rs.getString("first");
    String last = rs.getString("last");

    //Display values
    System.out.print("ID: " + id);
    System.out.print(", Age: " + age);
    System.out.print(", First: " + first);
    System.out.println(", Last: " + last);
  }
  //STEP 6: Clean-up environment
  rs.close();
  stmt.close();
  conn.close();
}catch(SQLException se){
  //Handle errors for JDBC
  se.printStackTrace();
}catch(Exception e){
  //Handle errors for Class.forName
```

```
      e.printStackTrace();
    }finally{
      //finally block used to close resources
      try{
        if(conn!=null)
          conn.close();
      }catch(SQLException se){
        se.printStackTrace();
      }//end finally try
    }//end try
    System.out.println("Goodbye!");
  }//end main
}//end JDBCExample
```

अब, उपरोक्त उदाहरण को संकलित करते हैं—

```
C: \> javac JDBCExample.java
C: \>
```

जब आप JDBCExample चलाते हैं, तो कोई समस्या नहीं होने पर यह निम्न परिणाम उत्पन्न करता है, अन्यथा इसी त्रुटि को पकड़ा जाएगा और त्रुटि संदेश प्रदर्शित किया जाएगा —

```
C:\>java JDBCExample
Connecting to database...
Creating statement...
ID: 100, Age: 18, First: Prashant, Last: Singh
ID: 101, Age: 25, First: Harshit, Last: Sharma
ID: 102, Age: 30, First: Aman, Last: Jain
ID: 103, Age: 28, First: Sumit, Last: Mittal
```

2.11 JDBC एप्लिकेशन बनाना

JDBC एप्लिकेशन के निर्माण में शामिल छह चरण निम्नलिखित हैं—

- **पैकेज import करें :** आवश्यकता है कि आप डेटाबेस प्रोग्रामिंग के लिए आवश्यक JDBC CLASSES वाले पैकेजों को शामिल करें। सबसे अधिक बार, import java.sql का उपयोग करना।
- **JDBC ड्राइवर रजिस्टर करें :** इसके लिए आवश्यक है कि आप एक ड्राइवर को इनिशियलाइज़ करें ताकि आप डेटाबेस के साथ एक संचार चैनल खोल सकें।
- **एक कनेक्शन खोलें :** एक कनेक्शन ऑब्जेक्ट बनाने के लिए DriverManager.getConnection () Method का उपयोग करने की आवश्यकता होती है, जो डेटाबेस के साथ एक भौतिक कनेक्शन का प्रतिनिधित्व करता है।
- **एक क्वेरी Execute करें :** डेटाबेस के लिए SQL स्टेटमेंट बनाने और सबमिट करने के लिए स्टेटमेंट टाइप के ऑब्जेक्ट का उपयोग करने की आवश्यकता होती है।

- **RESULTSET से डेटा निकालें :** RESULTSET से डेटा को पुनः प्राप्त करने के लिए आपको उपयुक्त ResultSet.getXXX () Method का उपयोग करना होगा।
- **Environment को साफ करें :** स्पष्ट रूप से सभी डेटाबेस रिसोर्सेज को बंद करने की आवश्यकता है जो जेवीएम के Garbage collection पर निर्भर है।

नमूना कोड

यह नमूना उदाहरण एक टेम्पलेट के रूप में काम कर सकता है जब आपको भविष्य में अपना जेडीबीसी आवेदन बनाने की आवश्यकता होती है।

यह नमूना कोड पिछले अध्याय में किए गए environment और डेटाबेस सेटअप के आधार पर लिखा गया है।

निम्न उदाहरण को FirstExample.java में कॉपी और पेस्ट करें, निम्नानुसार संकलित करें और चलाएं—

```
//STEP 1. Import required packages
import java.sql.*;

public class FirstExample {
  // JDBC driver name and database URL
  static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
  static final String DB_URL = "jdbc:mysql://localhost/EMP";

  //  Database credentials
  static final String USER = "username";
  static final String PASS = "password";

  public static void main(String[] args) {
  Connection conn = null;
  Statement stmt = null;
  try{
    //STEP 2: Register JDBC driver
    Class.forName("com.mysql.jdbc.Driver");

    //STEP 3: Open a connection
    System.out.println("Connecting to database...");
    conn = DriverManager.getConnection(DB_URL,USER,PASS);

    //STEP 4: Execute a query
    System.out.println("Creating statement...");
```

```
    stmt = conn.createStatement();
    String sql;
    sql = "SELECT id, first, last, age FROM Employees";
    ResultSet rs = stmt.executeQuery(sql);

    //STEP 5: Extract data from result set
    while(rs.next()){
      //Retrieve by column name
      int id  = rs.getInt("id");
      int age = rs.getInt("age");
      String first = rs.getString("first");
      String last = rs.getString("last");

      //Display values
      System.out.print("ID: " + id);
      System.out.print(", Age: " + age);
      System.out.print(", First: " + first);
      System.out.println(", Last: " + last);
    }
    //STEP 6: Clean-up environment
    rs.close();
    stmt.close();
    conn.close();
  }catch(SQLException se){
    //Handle errors for JDBC
    se.printStackTrace();
  }catch(Exception e){
    //Handle errors for Class.forName
    e.printStackTrace();
  }finally{
    //finally block used to close resources
    try{
      if(stmt!=null)
        stmt.close();
```

```
    }catch(SQLException se2){
    }// nothing we can do
    try{
      if(conn!=null)
        conn.close();
    }catch(SQLException se){
      se.printStackTrace();
    }//end finally try
  }//end try
  System.out.println("Goodbye!");
}//end main
}//end FirstExample
```

अब हम उपरोक्त उदाहरण को संकलित करते हैं—

```
C:\>javac FirstExample.java
C:\>
```

जब आप FirstExample चलाते हैं, तो यह निम्न परिणाम उत्पन्न करता है—

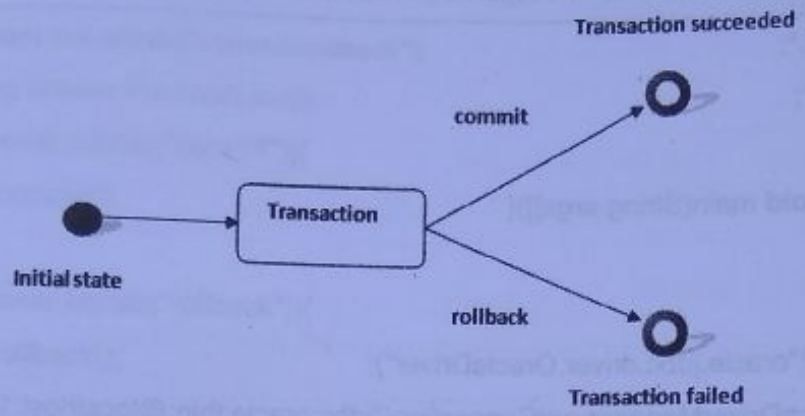
```
C:\>java FirstExample
Connecting to database...
Creating statement...
ID: 100, Age: 18, First: Prashant, Last: Singh
ID: 101, Age: 25, First: Harshit, Last: Sharma
ID: 102, Age: 30, Aman, Last: Jain
ID: 103, Age: 28, First: Sumit, Last: Mittal
```

2.12 JDBC में ट्रेन्ज़ेक्शन Management

ट्रेन्ज़ेक्शन कार्य की एक इकाई का प्रतिनिधित्व करता है।

ACID गुण ट्रेन्ज़ेक्शन प्रबंधन का अच्छी तरह से वर्णन करता है। ACID का मतलब एटोमिसिटी, कंसिस्टेंसी, आइसोलेशन और ड्यूरेबिलिटी है।

- **Atomicity :** सभी सफल या कोई भी नहीं है।
- **Consistency :** डेटाबेस को एक consistant स्थिति से दूसरे consistant स्थिति में लाना सुनिश्चित करता है।
- **Isolation :** यह सुनिश्चित करता है कि ट्रेन्ज़ेक्शन अन्य ट्रेन्ज़ेक्शन से अलग है।
- **Durability :** इसका मतलब है कि एक बार ट्रेन्ज़ेक्शन हो जाने के बाद, यह त्रुटियों की स्थिति में भी बना रहेगा।



ट्रेन्ज़ेक्शन Management का लाभ

तेजी से प्रदर्शन : यह प्रदर्शन को तेज़ बनाता है क्योंकि कमिट के समय डेटाबेस हिट हो जाता है।

JDBC में, कनेक्शन इंटरफ़ेस ट्रेन्ज़ेक्शन को प्रबंधित करने के Methods प्रदान करता है।

METHOD	विवरण
void setAutoCommit(boolean status)	यह bydefault true है मतलब कि प्रत्येक ट्रेन्ज़ेक्शन bydefault committed है।
void commit()	ट्रेन्ज़ेक्शन करता है।
void rollback()	ट्रेन्ज़ेक्शन रद्द करता है।

JDBC स्टेटमेंट में ट्रेन्ज़ेक्शन प्रबंधन का सरल उदाहरण

आइए स्टेटमेंट का उपयोग करके ट्रेन्ज़ेक्शन प्रबंधन का सरल उदाहरण देवें।

```
import java.sql.*;
class FetchRecords{
public static void main(String args[])throws Exception{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","syste
m","oracle");
con.setAutoCommit(false);

Statement stmt=con.createStatement();
stmt.executeUpdate("insert into user420 values(190,'abhi',40000)");
stmt.executeUpdate("insert into user420 values(191,'umesh',50000)");
con.commit();
con.close();
}}
```

JDBC में ट्रेन्ज़ेक्शन प्रबंधन का उदाहरण Prepared statement का उपयोग करके।

```
import java.sql.*;
import java.io.*;
class TM{
public static void main(String args[]){
try{

Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
con.setAutoCommit(false);

PreparedStatement ps=con.prepareStatement("insert into user420 values(?,?,?)");

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
while(true){

System.out.println("enter id");
String s1=br.readLine();
int id=Integer.parseInt(s1);

System.out.println("enter name");
String name=br.readLine();

System.out.println("enter salary");
String s3=br.readLine();
int salary=Integer.parseInt(s3);

ps.setInt(1,id);
ps.setString(2,name);
ps.setInt(3,salary);
ps.executeUpdate();
```

```
System.out.println("commit/rollback");
String answer =br.readLine();
if(answer.equals("commit")){
con.commit();
}
if(answer.equals("rollback")){
con.rollback();
}

System.out.println("Want to add more records y/n");
String ans=br.readLine();
if(ans.equals("n")){
break;
}
}
con.commit();
System.out.println("record successfully saved");

con.close();//before closing connection commit() is called
}catch(Exception e){System.out.println(e);}

}}
```

2.13 जेडीबीसी में बैच प्रोसेसिंग (Batch Processing in JDBC)

एकल क्वेरी Execute करने के बजाय, हम प्रश्नों के एक बैच (समूह) को Execute कर सकते हैं। यह प्रदर्शन को तेज करता है।

Java.sql.Statement और java.sql.PreparedStatement इंटरफेस बैच प्रोसेसिंग के लिए Methods प्रदान करते हैं।

बैच Processing का लाभ

तेज प्रदर्शन

स्टेटमेंट इंटरफेस के Methods

बैच Processing के लिए आवश्यक Methods नीचे दिए गए हैं—

METHOD	विवरण
void addBatch(String query)	यह बैच में क्वेरी जोड़ता है।
int[] executeBatch()	यह प्रश्नों के बैच को निष्पादित करता है।

जेडीबीसी में बैच प्रोसेसिंग का उदाहरण

आइए जदबीसी में बैच प्रोसेसिंग का सरल उदाहरण देवें। यह निम्न चरणों का पालन करता है:

- ड्राइवर काल्स लोड करें
- कनेक्शन बनों
- स्टेटमेंट बनों
- बैच में क्वेरी जोड़ें
- बैच Execute करें
- connection क्लोज़ करें

```
import java.sql.*;
class FetchRecords{
public static void main(String args[])throws Exception{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:152
1:xe","system","oracle");
con.setAutoCommit(false);

Statement stmt=con.createStatement();
stmt.addBatch("insert into user420 values(190,'abhi',40000)");
stmt.addBatch("insert into user420 values(191,'umesh',50000)");

stmt.executeBatch();//executing the batch

con.commit();
con.close();


}}
```

Prepared Statement से बैच प्रोसेसिंग का उदाहरण

```
import java.sql.*;
import java.io.*;
class BP{
public static void main(String args[]){
try{

Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe
","system","oracle");

PreparedStatement ps=con.prepareStatement("insert into user420 values(?,?,?)");

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
while(true){

System.out.println("enter id");
String s1=br.readLine();
int id=Integer.parseInt(s1);

System.out.println("enter name");
String name=br.readLine();

System.out.println("enter salary");
String s3=br.readLine();
int salary=Integer.parseInt(s3);

ps.setInt(1,id);
ps.setString(2,name);
ps.setInt(3,salary);

ps.addBatch();
System.out.println("Want to add more records y/n");
String ans=br.readLine();
if(ans.equals("n")){
break;
}

}
```

```
    ps.executeBatch();

    System.out.println("record successfully saved");

    con.close();
    }catch(Exception e){System.out.println(e);}

}}
```

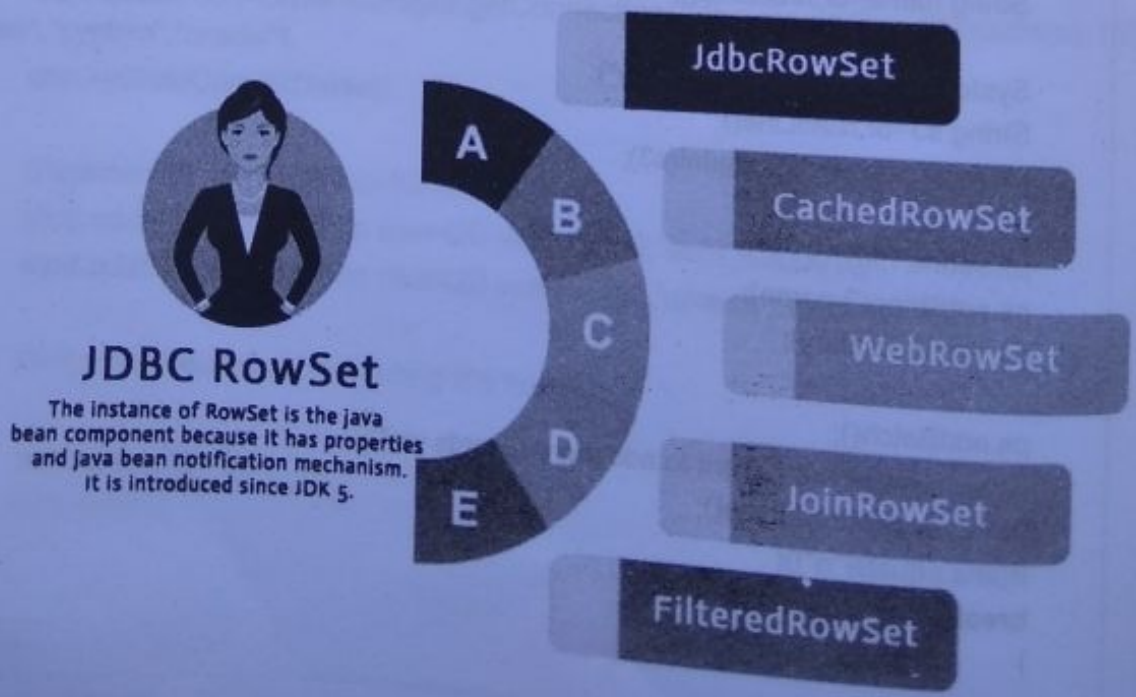
2.14. JDBC रोसेट (JDBC RowSet)

रोसेट का उदाहरण जावा बीन घटक है क्योंकि इसमें गुण और जावा बीन अधिसूचना मैकेनिज्म है। इसे JDK 5 के बाद से पेश किया गया है।

यह ResultSet का Wrapper है। यह ResultSet की तरह Tabular डेटा रखता है लेकिन इसका उपयोग करना आसान और लचीला है।

रोसेट इंटरफ़ेस के कार्यान्वयन काल्स निम्नानुसार हैं—

- JdbcRowSet
- CachedRowSet
- WebRowSet
- JoinRowSet
- FilteredRowSet



आइए देखें कि रोसेट को कैसे बनाएं और execute करें।

```
JdbcRowSet rowSet = RowSetProvider.newFactory().createJdbcRowSet();
rowSet.setUrl("jdbc:oracle:thin:@localhost:1521:xe");
rowSet.setUsername("system");
rowSet.setPassword("oracle");

rowSet.setCommand("select * from emp400");
rowSet.execute();
```

रोसेट का लाभ

रोसेट के उपयोग के फायदे नीचे दिए गए हैं—

1. यह प्रयोग करने में आसान और लचीला है।
2. यह डिफ़ॉल्ट रूप से स्क्रॉल करने योग्य और अपडेट करने योग्य है।

JdbcRowSet का सरल उदाहरण

आइए बिना इवेंट हैंडलिंग कोड के JdbcRowSet का सरल उदाहरण देवें।

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import javax.sql.RowSetEvent;
import javax.sql.RowSetListener;
import javax.sql.rowset.JdbcRowSet;
import javax.sql.rowset.RowSetProvider;

public class RowSetExample {
    public static void main(String[] args) throws Exception {
        Class.forName("oracle.jdbc.driver.OracleDriver");

  //Creating and Executing RowSet
    JdbcRowSet rowSet = RowSetProvider.newFactory().createJdbcRowSet();
    rowSet.setUrl("jdbc:oracle:thin:@localhost:1521:xe");
    rowSet.setUsername("system");
    rowSet.setPassword("oracle");
```

```
        rowSet.setCommand("select * from emp400");
        rowSet.execute();

    while (rowSet.next()) {
            // Generating cursor Moved event
            System.out.println("Id: " + rowSet.getString(1));
            System.out.println("Name: " + rowSet.getString(2));
            System.out.println("Salary: " + rowSet.getString(3));
        }

    }
}
```

इवेंट हैंडलिंग के साथ Jdbc रोसेट का पूर्ण उदाहरण

JdbcRowSet के साथ ईवेंट हैंडलिंग करने के लिए, आपको JDbcRowSet की addRowSetListener method में RowSetListener का उदाहरण जोड़ना होगा।

RowSetListener इंटरफ़ेस 3 METHOD प्रदान करता है जिसे लागू किया जाना चाहिए। वे इस प्रकार हैं—

(1) public void cursorMoved(RowSetEvent event);

(2) public void rowChanged(RowSetEvent event);

(3) public void rowSetChanged(RowSetEvent event);

चलो डेटा को पुनः प्राप्त करने के लिए कोड लिखते हैं और कर्सर ले जाने के दौरान कुछ अतिरिक्त कार्य करते हैं, कर्सर को बदल दिया जाता है या rows बदल दी जाती हैं। इवेंट हैंडलिंग ऑपरेशन को ResultSet का उपयोग करके नहीं किया जा सकता है, इसलिए इसे अभी पसंद किया गया है।

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import javax.sql.RowSetEvent;
import javax.sql.RowSetListener;
import javax.sql.rowset.JdbcRowSet;
import javax.sql.rowset.RowSetProvider;

public class RowSetExample {
    public static void main(String[] args) throws Exception {
        Class.forName("oracle.jdbc.driver.OracleDriver");
```

```
    //Creating and Executing RowSet
    JdbcRowSet rowSet = RowSetProvider.newFactory().createJdbcRowSet();
    rowSet.setUrl("jdbc:oracle:thin:@localhost:1521:xe");
    rowSet.setUsername("system");
    rowSet.setPassword("oracle");

      rowSet.setCommand("select * from emp400");
      rowSet.execute();

    //Adding Listener and moving RowSet
    rowSet.addRowSetListener(new MyListener());

            while (rowSet.next()) {
                // Generating cursor Moved event
                System.out.println("Id: " + rowSet.getString(1));
                System.out.println("Name: " + rowSet.getString(2));
                System.out.println("Salary: " + rowSet.getString(3));
            }

        }
    }

class MyListener implements RowSetListener {
    public void cursorMoved(RowSetEvent event) {
            System.out.println("Cursor Moved...");
    }
    public void rowChanged(RowSetEvent event) {
            System.out.println("Cursor Changed...");
    }
    public void rowSetChanged(RowSetEvent event) {
            System.out.println("RowSet changed...");
    }
}

}
```

अभ्यास

1. JDBC क्या है?
2. सभी JDBC ड्राइवर को समझों।
3. हम DriveManager काल्स का उपयोग कैसे कर सकते हैं?
4. JDBC आर्किटेक्चर के बारे में विस्तार से बतों।
5. CURD से आपका क्या अभिप्राय है?
6. हम JDBC में ऑपरेशन कैसे बना सकते हैं?
7. हम JDBC में रीड ऑपरेशन कैसे लागू कर सकते हैं?
8. हम जेडीबीसी में अपडेट ऑपरेशन कैसे लागू कर सकते हैं?
9. हम JDBC में डिलीट ऑपरेशन को कैसे लागू कर सकते हैं?
10. हम JDBC में डेटाबेस कनेक्शन कैसे स्थापित कर सकते हैं?
11. JDBC स्टेटमेंट को परिभाषित करें।
12. PREPARED स्टेटमेंट की व्याख्या करें।
13. PREPARED स्टेटमेंट के फायदे बतों।
14. PREPARED Statement के नुकसान दे।
15. हम JDBC में RESULTSET का उपयोग कैसे कर सकते हैं?
16. JDBC डेटा प्रकारों की व्याख्या करें।
17. JDBC एक्सेप्शन हैंडलिंग के बारे में बतों।

बहुविकल्पी प्रश्नः

1. निम्नलिखित में से किसमें तारीख और समय दोनों हैं?
   (a) java.io.date
   (b) java.sql.date
   (c) java.util.date
   (d) java.util.dateTime
   उत्तर : (d)
2. JDBC कनेक्शन पूल का उपयोग करने में निम्नलिखित में से कौन सा लाभ है?
   (a) धीमी performance
   (b) अधिक मेमरी का उपयोग करना
   (c) कम मेमोरी का उपयोग करना
   (d) बेहतर प्रदर्शन
   उत्तर : (d)
3. जावा में PREPAREDSTATEMENT का उपयोग करने में निम्नलिखित में से कौन सा लाभ है?
   (a) धीमा प्रदर्शन
   (b) SQL इंजेक्शन को प्रोत्साहित करता है
   (c) SQl इंजेक्शन को रोकता है
   (d) अधिक मेमोरी
   उत्तर : (c)

4. निम्नलिखित में से किस में तारीख की जानकारी है?
   (a) java.sql.TimeStamp
   (b) java.sql.Time
   (c) java.io.Time
   (d) java.io.TimeStamp

   उत्तर : (a)

5. setAutoCommit(false) क्या करता है?
   (a) प्रत्येक क्वेरी के बाद ट्रेन्ज़ेक्शन शुरू करता है
   (b) स्पष्ट रूप से ट्रेन्ज़ेक्शन को स्वीकार करता है
   (c) प्रत्येक क्वेरी के बाद स्वचालित रूप से ट्रेन्ज़ेक्शन नहीं करता है
   (d) ट्रेन्ज़ेक्शन कभी नहीं करता है।

   उत्तर : (c)

6. Stored Procedure को कॉल करने के लिए निम्न में से किसका उपयोग किया जाता है?
   (a) स्टेटमेंट
   (b) Prepared स्टेमेंटमेंट
   (c) कॉलेबल स्टैमेंट
   (d) Called स्टैटमेंट

   उत्तर : (c)

7. एक डेटाबेस और जावा प्रोग्रामिंग भाषा में लिखे गए एप्लिकेशन के बीच डेटा ट्रांसफर करने के लिए, JDBC API इनमें से कौन सी METHOD प्रदान करता है?
   (a) SQL टाइप के परिणाम के लिए जावा प्रकार के रूप में पुनर्प्राप्त करने के लिए परिणाम काल्स पर METHODS।
   (b) SQL स्टेटमेंट पैरामीटर के रूप में जावा प्रकार भेजने के लिए PREPAREDSTATEMENT काल्स पर METHODS।
   (c) जावा टाइप के रूप में SQL OUT parameters को पुनः प्राप्त करने के लिए CallableStatement काल्स पर METHODS।
   (d) उपरोक्त सभी कथन ।

   उत्तर : (d)

8. JDBC API ने हमेशा जावा प्रोग्रामिंग लैंग्वेज में परिभाषित OBJECTS के लगातार स्टोरेज को getObject और setOject के माध्यम से सपोर्ट किया है।
   (a) सच
   (b) असत्य

   उत्तर : (a)

9. कौन सा JDBC प्रकार Single Precision फ्लोटिंग पॉइंट संख्या का प्रतिनिधित्व करता है जो सात अंकों के मंटिसा का समर्थन करता है?
   (a) रिअल
   (b) डबल
   (c) फ्लोट
   (d) इन्टिजर

   उत्तर : (a)

10. JDBC 2.0 कोर एपीआई के साथ कितने रिजल्ट सेट उपलब्ध हैं?
   (a) 2
   (b) 3
   (c) 4
   (d) 5

   उत्तर : (b)

प्रोग्रामिंग अभ्यास

1. जावा का उपयोग करके डेटाबेस तालिका बनाने के लिए प्रोग्राम लिखें।
2. रिकॉर्ड डालने, अपडेट करने, हटाने और चयन करने के लिए प्रोग्राम लिखें ।
3. डेटाबेस से रिकॉर्ड हटाने के लिए प्रोग्राम लिवें।
4. ओरेकल डेटाबेस में तारीख डालने के लिए प्रोग्राम लिवें।
5. SQL SELECT क्वेरी Execute करने के लिए प्रोग्राम लिवें।
6. जावा के साथ MySQL या Oracle कनेक्ट करने के लिए प्रोग्राम लिखें।
7. JDBC में CURD को लागू करने के लिए प्रोग्राम लिदें।
8. JDBC में EXCEPTION हैंडलिंग को लागू करने के लिए प्रोग्राम लिखें।

❑

3

जावा सर्वलेटस
JAVA SERVLETS

LEARNING OBJECTIVES :

After reading this chapter, you would be able to understand :

- परिचय (Introduction)
- सर्वलेट शब्दावली (Servlet Terminology)
- एक सर्वलेट का जीवन चक्र (Life cycle of a servlet)
- सर्वलेट एपीआई (Servlet API)
- सर्वलेट इंटरफ़ेस (Servlet interface)
- जेनरिक सर्वलेट क्लास (Generics Servlet class)
- Http सर्वलेट क्लास
- RequestDispatcher
- सर्वलेट बनाना (Creating Servlet)
- WAR फ़ाइल
- web.xml में welcome-file-list
- सर्वलेट फिल्टर (Servlet Filter)

3.1 परिचय (Introduction)

आज हम सभी को डायनामिक वेब पेज बनाने की आवश्यकता के बारे में पता है, जो कि समय के अनुसार साइट कंटेंट को बदलने की क्षमता रखते हैं या क्लाइंट द्वारा प्राप्त रिक्वेस्ट के अनुसार कंटेंट उत्पन्न करने में सक्षम हैं। अगर आपको जावा में कोडिंग पसंद है, तो आपको यह जानकर खुशी होगी कि जावा का उपयोग करने से डायनामिक वेब पेज बनाने का एक मेथड भी मौजूद है और वह मेथड है जावा सर्वलेट। लेकिन इससे पहले कि हम अपने विषय के साथ आगे बढ़ें, चलो पहले सर्वर-साइड एक्सटेंशन की आवश्यकता को समझते हैं। सर्वलेट्स जावा प्रोग्राम हैं जो जावा-सक्षम वेब सर्वर या एप्लीकेशन सर्वर पर चलते हैं। वे वेब सर्वर से प्राप्त रिक्वेस्ट को संभालने के लिए उपयोग किए जाते हैं, रिक्वेस्ट को संसाधित करते हैं, प्रतिक्रिया उत्पन्न करते हैं, फिर वेब सर्वर पर प्रतिक्रिया भेजते हैं।

सर्वलेट्स के गुण:

- सर्वलेट्स सर्वर-साइड पर काम करते हैं।
- सर्वलेट्स वेब सर्वर से प्राप्त जटिल अनुरोधों को संभालने में सक्षम हैं।

सर्वलेट्स का निष्पादनः

सर्वलेट्स के निष्पादन में छह बुनियादी स्टेप शामिल हैं:

1. क्लाइंट वेब सर्वर को रिक्वेस्ट भेजते हैं।
2. वेब सर्वर रिक्वेस्ट प्राप्त करता है।
3. वेब सर्वर संबंधित सर्वलेट के लिए रिक्वेस्ट पारित करता है।
4. सर्वलेट रिक्वेस्ट को संसाधित करता है और आउटपुट के रूप में प्रतिक्रिया उत्पन्न करता है।
5. सर्वलेट प्रतिक्रिया को वेब सर्वर पर वापस भेजता है।
6. वेब सर्वर क्लाइंट को प्रतिक्रिया भेजता है और क्लाइंट ब्राउज़र स्क्रीन पर प्रदर्शित करता है।

सर्वलेट क्यों जानें?

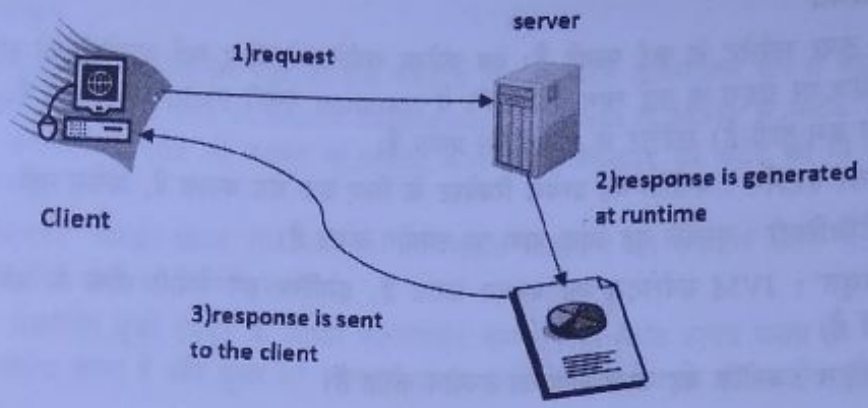
सर्वलेट्स का उपयोग करके, आप वेब पेज रूपों के माध्यम से यूजरस से इनपुट एकत्र कर सकते हैं, डेटाबेस या किसी अन्य स्रोत से रिकॉर्ड प्रस्तुत कर सकते हैं और डायनामिक रूप से वेब पेज बना सकते हैं।

जावा सर्वलेट्स अक्सर उसी उद्देश्य को पूरा करते हैं जैसे कि Common Gateway Interface (CGI) का उपयोग करके कार्यान्वित प्रोग्राम। लेकिन सर्वलेट्स CGI की तुलना में कई फायदे प्रदान करते हैं।

- प्रदर्शन काफी बेहतर है।
- सर्वलेट्स वेब सर्वर के एड्रेस स्पेस के भीतर Execute होते हैं। प्रत्येक क्लाइंट रिक्वेस्ट को संभालने के लिए एक अलग प्रोसेस बनाने के लिए आवश्यक नहीं है।
- सर्वलेट्स Platform indepdent होते हैं क्योंकि वे जावा में लिखे जाते हैं।
- सर्वर पर जावा सुरक्षा मैनेजर सर्वर मशीन पर रिसोर्सेज की सुरक्षा के लिए प्रतिबंधों का एक सेट लागू करता है, इसलिए सर्वलेट्स पर भरोसा किया जाता है।
- जावा क्लास के लाइब्रेरी की पूर्ण कार्यक्षमता एक सर्वलेट के लिए उपलब्ध है। यह सॉकेट्स और RMI मैकेनिज्म के माध्यम से एप्लेट्स, डेटाबेस या अन्य सॉफ्टवेयर के साथ संवाद कर सकता है जिसे आपने पहले ही देखा है।

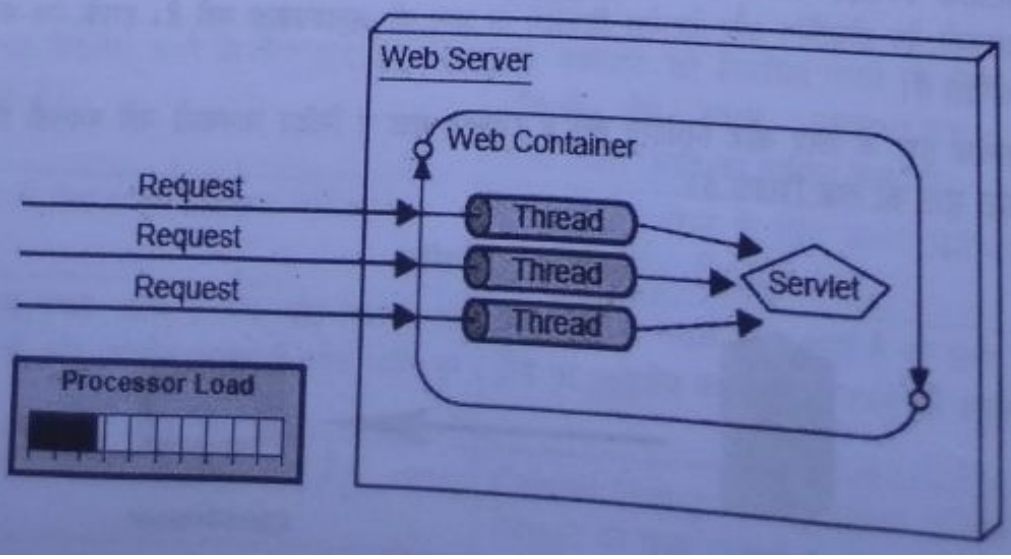
सर्वलेट का संदर्भ के आधार पर कई तरह से वर्णन किया जा सकता है।

- सर्वलेट एक तकनीक है जिसका उपयोग वेब एप्लीकेशन बनाने के लिए किया जाता है।
- सर्वलेट एक API है जो प्रलेखन सहित कई इंटरफेस और क्लास प्रदान करता है।
- सर्वलेट एक ऐसा इंटरफ़ेस है जिसे किसी भी सर्वलेट को बनाने के लिए लागू किया जाना चाहिए।
- सर्वलेट एक ऐसा क्लास है जो सर्वर की क्षमताओं का विस्तार करता है और आने वाले अनुरोधों का जवाब देता है। यह किसी भी रिक्वेस्ट का जवाब दे सकता है।
- सर्वलेट एक वेब घटक है जो एक डायनामिक वेब पेज बनाने के लिए सर्वर पर तैनात किया जाता है।



सर्वलेट के एप्लीकेशन

- क्लाइंट (ब्राउज़र) द्वारा भेजे गए स्पष्ट डेटा को पढ़ें। इसमें वेब पेज पर HTML फॉर्म शामिल है या यह एप्लेट या कस्टम HTTP क्लाइंट प्रोग्राम से भी आ सकता है।
- क्लाइंट (ब्राउज़र) द्वारा भेजे गए अंतर्निहित HTTP रिक्वेस्ट डेटा पढ़ें। इसमें कुकीज़, मीडिया टाइप और संपीड़न योजनों शामिल हैं जिन्हें ब्राउज़र समझता है ।
- डेटा संसाधित करें और परिणाम उत्पन्न करें। इस प्रोसेस में डेटाबेस से बात करना, RMI या CORBA कॉल को Execute करना, वेब service को लागू करना या सीधे प्रतिक्रिया की गणना करना पड़ सकता है।
- ग्राहकों (ब्राउज़रों) को स्पष्ट डेटा (यानी, दस्तावेज़) भेजें। यह दस्तावेज़ विभिन्न स्वरूपों में भेजा जा सकता है, जिसमें पाठ (HTML या XML), बाइनरी (GIF Image), Excel आदि शामिल हैं।
- क्लाइंट (ब्राउज़र) को Implicit HTTP प्रतिक्रिया भेजें। इसमें ब्राउज़र या अन्य क्लाइंट को यह बताना शामिल है कि किस टाइप का दस्तावेज़ लौटाया जा रहा है (उदाहरण के लिए, HTML), कुकीज और caching पैरामीटर सेट करना, और ऐसे अन्य कार्य।



सर्वलेट के फायदे

CGI के ऊपर सर्वलेट के कई फायदे हैं। वेब कंटेनर सर्वलेट के लिए कई अनुरोधों को संभालने के लिए थ्रेड बनाता है। प्रोसेसेज पर थ्रेड्स के कई लाभ हैं जैसे कि वे common मेमोरी Field साझा करते हैं, हल्के, थ्रेड के बीच संचार की लागत कम होती है। सर्वलेट के फायदे इस प्रकार हैं:

1. **बेहतर प्रदर्शन** : क्योंकि यह प्रत्येक रिक्वेस्ट के लिए एक थ्रेड बनाता है, प्रोसेस नहीं।
2. **पोर्टेबिलिटी** : क्योंकि यह जावा भाषा का उपयोग करता है।
3. **मजबूत** : JVM सर्वलेट्स का प्रबंधन करता है, इसलिए हमें मेमोरी लीक के बारे में चिंता करने की आवश्यकता नहीं है।
4. **सुरक्षित** : क्योंकि यह जावा भाषा का उपयोग करता है।

3.2 सर्वलेट शब्दावली (Servlet Terminology)

वेबसाइट

वेबसाइट संबंधित वेब पृष्ठों का एक संग्रह है जिसमें पाठ, चित्र, ऑडियो और वीडियो हो सकते हैं। वेबसाइट के पहले पेज को होम पेज कहा जाता है। प्रत्येक वेबसाइट का specific internet address (URL) होता है जिसे आपको वेबसाइट तक पहुंचने के लिए अपने ब्राउज़र में दर्ज करना होता है।

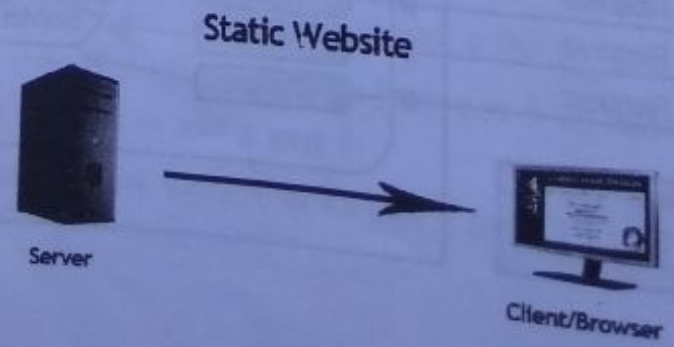
वेबसाइट को एक या अधिक सर्वरों पर होस्ट किया जाता है और इसे कम्प्यूटर नेटवर्क का उपयोग करके इसके होमपेज पर जाकर देखा जा सकता है। एक वेबसाइट को उसके मालिक द्वारा प्रबंधित किया जाता है जो एक व्यक्ति, कंपनी या एक संगठन हो सकता है।

एक वेबसाइट दो प्रकार की हो सकती है—

- स्टेटिक वेबसाइट
- डायनामिक वेबसाइट

स्टेटिक वेबसाइट

- स्टेटिक वेबसाइट बेसिक टाइप की वेबसाइट है जो बनाने में आसान है। स्टेटिक वेबसाइट बनाने के लिए आपको वेब प्रोग्रामिंग और डेटाबेस डिज़ाइन के ज्ञान की आवश्यकता नहीं है। इसके वेब पेज HTML में कोडित हैं।
- प्रत्येक पृष्ठ के लिए कोड निर्धारित होते हैं इसलिए पृष्ठ में निहित जानकारी नहीं बदलती है और यह एक प्रिंट पृष्ठ की तरह दिखता है।



डायनामिक वेबसाइट

डायनामिक वेबसाइट डायनामिक वेब पेजों का एक संग्रह है जिसकी कंटेंट डायनामिक रूप से बदल जाती है। यह एक डेटाबेस या कंटेंट प्रबंधन प्रणाली (Content Management System (CMS)) से कंटेंट तक पहुँचता है। इसलिए, जब आप डेटाबेस की कंटेंट को बदलते या अपडेट करते हैं, तो वेबसाइट की कंटेंट को भी बदल दिया जाता है या अपडेट किया जाता है।

डायनामिक वेबसाइट क्लाइंट-साइड स्क्रिप्टिंग या सर्वर-साइड स्क्रिप्टिंग का उपयोग करती है, या डायनामिक कंटेंट उत्पन्न करने के लिए दोनों का उपयोग करती है।

क्लाइंट साइड स्क्रिप्टिंग यूजर इनपुट के आधार पर क्लाइंट कम्प्यूटर पर कंटेंट उत्पन्न करता है। वेब ब्राउज़र सर्वर से वेब पेज को डाउनलोड करता है और यूजर को जानकारी प्रदान करने के लिए पेज के भीतर कोड को संसाधित करता है।

सर्वर साइड स्क्रिप्टिंग में, सॉफ्टवेयर सर्वर पर चलता है और सर्वर में प्रोसेसिंग पूरी हो जाती है फिर यूजर को सादे पेज भेजे जाते हैं।

Dynamic Website

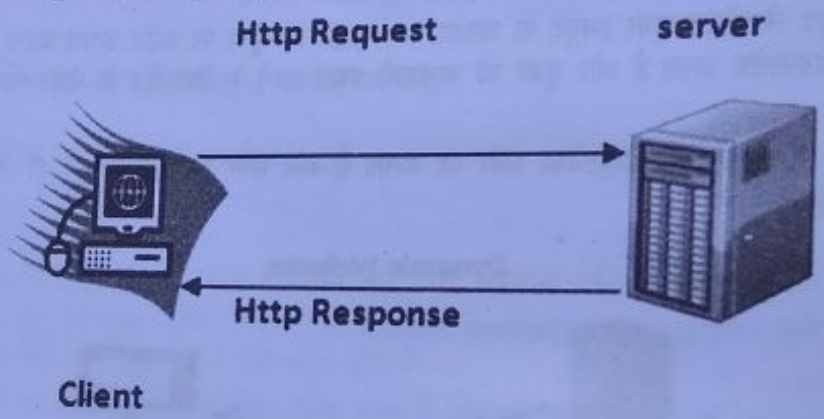
स्टेटिक बनाम डायनामिक वेबसाइट

स्टेटिक वेबसाइट	डायनामिक वेबसाइट
पृष्ठ लोड होने के बाद हर बार प्रीबिल्ट कंटेंट समान होती है।	कंटेंट जल्दी से उत्पन्न होती है और नियमित रूप से बदलती है।
यह एक वेबसाइट विकसित करने के लिए HTML कोड का उपयोग करता है।	यह वेबसाइट को विकसित करने के लिए सर्वर साइड भाषाओं जैसे PHP, SERVLET, JSP, और ASP.NET आदि का उपयोग करता है।
यह हर रिक्वेस्ट के लिए समान प्रतिक्रिया भेजता है।	यह प्रत्येक रिक्वेस्ट के लिए अलग HTML उत्पन्न कर सकता है।
कंटेंट केवल तब बदल जाती है जब कोई फ़ाइल को प्रकाशित करता है और अपडेट करता है (वेब सर्वर पर भेजता है)।	पृष्ठ में 'सर्वर-साइड' कोड होता है जो सर्वर को पेज लोड होने पर अद्वितीय कंटेंट उत्पन्न करने की अनुमति देता है।
लचीलापन स्टेटिक वेबसाइट का मुख्य लाभ है।	Content Management System (CMS) डायनामिक वेबसाइट का मुख्य लाभ है।

HTTP (हाइपर टेक्स्ट ट्रांसफर प्रोटोकॉल)

हाइपरटेक्स्ट ट्रांसफर प्रोटोकॉल (HTTP) सहयोगी, वितरित, हाइपरमीडिया सूचना प्रणाली के लिए एप्लीकेशन स्तरीय प्रोटोकॉल है। यह डेटा संचार प्रोटोकॉल है जिसका उपयोग क्लाइंट और सर्वर के बीच संचार स्थापित करने के लिए किया जाता है।

HTTP टीसीपी / आईपी आधारित संचार प्रोटोकॉल है, जिसका उपयोग डिफ़ॉल्ट पोर्ट के साथ World Wide Web (WWW) पर छवि फ़ाइलों, क्वेरी परिणामों, एचटीएमएल फाइलों आदि जैसे डेटा को वितरित करने के लिए किया जाता है। यह कंप्यूटरों को एक दूसरे के साथ संवाद के लिए मानकीकृत मेथड प्रदान करता है।



HTTP के characterstics :

- यह प्रोटोकॉल है जो वेब सर्वर और ब्राउज़रों को वेब पर डेटा का आदान-प्रदान करने की अनुमति देता है।
- यह एक रिक्वेस्ट प्रतिक्रिया प्रोटोकॉल है।
- यह ऊण्इ पोर्ट 80 पर डिफ़ॉल्ट रूप से विश्वसनीय ऊण्इ कनेक्शन का उपयोग करता है।
- यह स्टेटलेस है अर्थात प्रत्येक रिक्वेस्ट को नया रिक्वेस्ट माना जाता है। दूसरे शब्दों में, सर्वर डिफ़ॉल्ट रूप से यूज़र को नहीं पहचानता है।

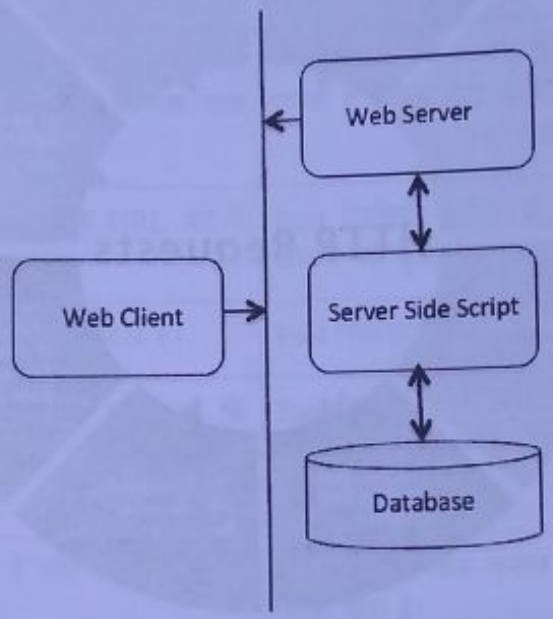
HTTP के features :

तीन मौलिक विशेषतों हैं जो HTTP को संचार के लिए उपयोग किया जाने वाला एक सरल और शक्तिशाली प्रोटोकॉल बनाती हैं—

- **HTTP मीडिया independent है :** यह निर्दिष्ट करता है कि किसी भी टाइप की मीडिया कंटेंट को HTTP द्वारा भेजा जा सकता है जब तक कि सर्वर और क्लाइंट दोनों डेटा कंटेंट को संभाल सकते हैं।
- **HTTP कनेक्शन रहित है :** यह एक कनेक्शन रहित दृष्टिकोण है जिसमें HTTP क्लाइंट अर्थात, एक ब्राउज़र HTTP रिक्वेस्ट शुरू करता है और रिक्वेस्ट के भेजे जाने के बाद सर्वर से क्लाइंट डिस्कनेक्ट हो जाता है और प्रतिक्रिया का इंतजार करता है।
- **HTTP स्टेटलेस है :** क्लाइंट और सर्वर केवल एक वर्तमान रिक्वेस्ट के दौरान एक दूसरे के बारे में जानते हैं। बाद में दोनों एक-दूसरे को भूल जाते हैं। प्रोटोकॉल की निष्क्रिय प्रकृति के कारण, न तो क्लाइंट और न ही सर्वर वेब पृष्ठों पर विभिन्न अनुरोधों के बारे में जानकारी को बरकरार रख सकता है।

HTTP का मूल आर्किटेक्चर (हाइपर टेक्स्ट ट्रांसफर प्रोटोकॉल) :

नीचे दिए गए चित्र वेब एप्लीकेशन की मूल आर्किटेक्चर का प्रतिनिधित्व करते हैं और दर्शाते हैं कि HTTP कैसे काम करता है—



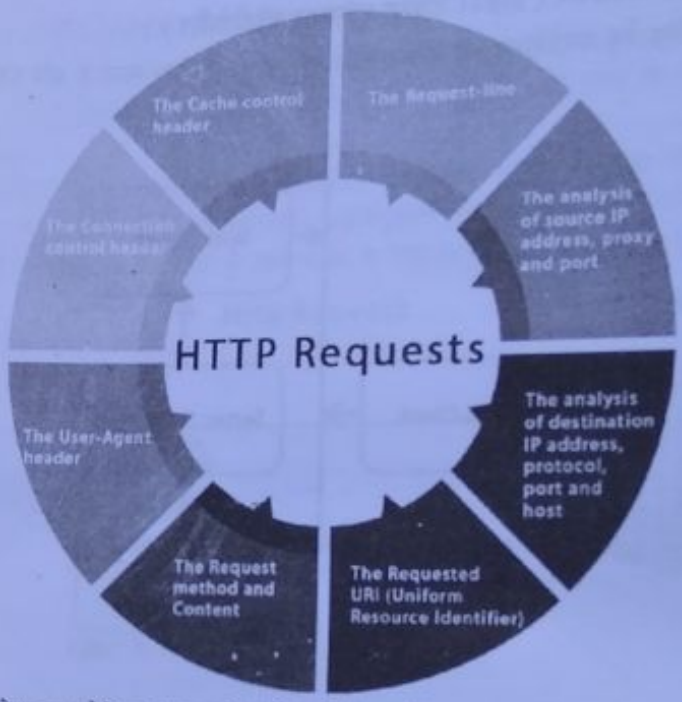
HTTP रिक्वेस्ट/रिस्पॉन्स प्रोटोकॉल है जो क्लाइंट/सर्वर आधारित आर्किटेक्चर पर आधारित है। इस प्रोटोकॉल में, वेब ब्राउज़र, सर्च इंजन आदि HTTP क्लाइंट के रूप में व्यवहार करते हैं और वेब सर्वर जैसे सर्वलेट एक सर्वर के रूप में व्यवहार करते हैं—

HTTP रिक्वेस्ट (HTTP Requests)

कम्प्यूटर द्वारा वेब सर्वर को भेजे गए रिक्वेस्ट में सभी टाइप की संभावित रोचक जानकारी होती है; इसे HTTP अनुरोधों के रूप में जाना जाता है।

HTTP क्लाइंट रिक्वेस्ट संदेश के रूप में सर्वर को रिक्वेस्ट भेजता है जिसमें निम्नलिखित जानकारी शामिल है—

- The Request-line
- The analysis of source IP address, proxy and port
- The analysis of destination IP address, protocol, port and host
- The Requested URI (Uniform Resource Identifier)
- The Request method and Content
- The User-Agent header
- The Connection control header
- The Cache control header



HTTP रिक्वेस्ट मेथड अनुरोधित URI (यूनिफ़ॉर्म रिसोर्स आइडेंटिफ़ायर) द्वारा पहचाने गए रिसोर्सेज पर की जाने वाला मेथड को इंगित करता है। यह मेथड केस-संवेदी है और इसे अपरकेस में उपयोग किया जाना चाहिए।

HTTP रिक्वेस्ट मेथड हैं—

HTTP अनुरोध	विवरण
GET	अनुरोधित URL पर रिसोर्सेज प्राप्त करने का प्रयास करता है।
POST	सर्वर से जानकारी को स्वीकार करने के लिए पूछता है।
HEAD	जो भी GET लौटेगा उसके केवल हेडर भाग के लिए पूछता है।
TRACE	परीक्षण या समस्या निवारण के लिए रिक्वेस्ट संदेश के लूपबैक के लिए पूछता है।
PUT	अनुरोधित URL पर संलग्न जानकारी (निकाय) डालने के लिए कहते हैं।
DELETE	अनुरोधित URL पर रिसोर्सेज को हटाने के लिए कहता है।
OPTIONS	HTTP methods की सूची के लिए पूछता है जिसमें रिक्वेस्ट URL पर मौजूद चीज़ प्रतिक्रिया दे सकते हैं।

सर्वर और क्लाइंट के बीच रिक्वेस्ट-प्रतिक्रिया के दो सामान्य मेथड्स हैं:

- GET- यह एक निर्दिष्ट रिसोर्सेज से डेटा का रिक्वेस्ट करता है
- POST - यह एक निर्दिष्ट रिसोर्सेज में संसाधित डेटा को प्रस्तुत करता है

GET बनाम POST

GET और पोस्ट रिक्वेस्ट के बीच कई अंतर हैं। आइए देखते हैं ये अंतर—

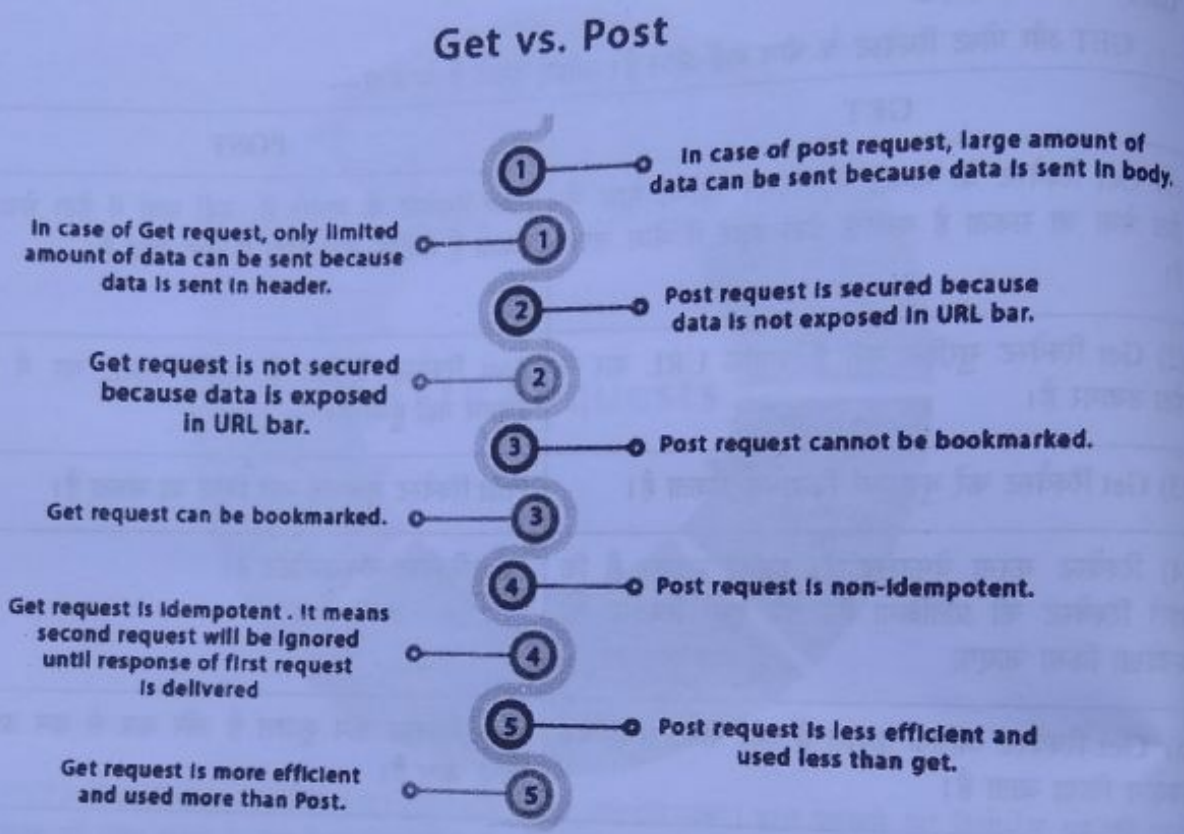
GET	POST
(1) Get रिक्वेस्ट के मामले में, केवल सीमित मात्रा में डेटा भेजा जा सकता है क्योंकि डेटा हेडर में भेजा जाता है।	Post रिक्वेस्ट के मामले में, बड़ी मात्रा में डेटा भेजा जा सकता है क्योंकि डेटा बॉडी में भेजा जाता है।
(2) Get रिक्वेस्ट सुरक्षित नहीं है क्योंकि URL बार में डेटा उजागर है।	Post रिक्वेस्ट सुरक्षित है क्योंकि URL बार में डेटा उजागर नहीं हुआ है।
(3) Get रिक्वेस्ट करें बुकमार्क किया जा सकता है।	Post रिक्वेस्ट बुकमार्क नहीं किया जा सकता है।
(4) रिक्वेस्ट करना बेमतलब है। इसका मतलब है कि पहले रिक्वेस्ट की प्रतिक्रिया देने तक दूसरे रिक्वेस्ट को अनदेखा किया जाएगा	Post रिक्वेस्ट नॉन-इम्पोर्टेंट है।
(5) Get रिक्वेस्ट अधिक कुशल है और पोस्ट से अधिक उपयोग किया जाता है।	Post रिक्वेस्ट कम कुशल है और कम से कम उपयोग किया जाता है।

GET रिक्वेस्ट की विशेषतों

- यह ब्राउज़र हिस्टरी में रहता है।
- इसे बुकमार्क किया जा सकता है।
- इसे cache किया जा सकता है।
- इसकी लंबाई प्रतिबंध है।
- संवेदनशील डेटा से निपटने के दौरान इसका उपयोग कभी नहीं किया जाना चाहिए।
- इसका उपयोग केवल डेटा को पुनर्प्राप्त करने के लिए किया जाना चाहिए।

POST अनुरोधों की विशेषतों हैं–

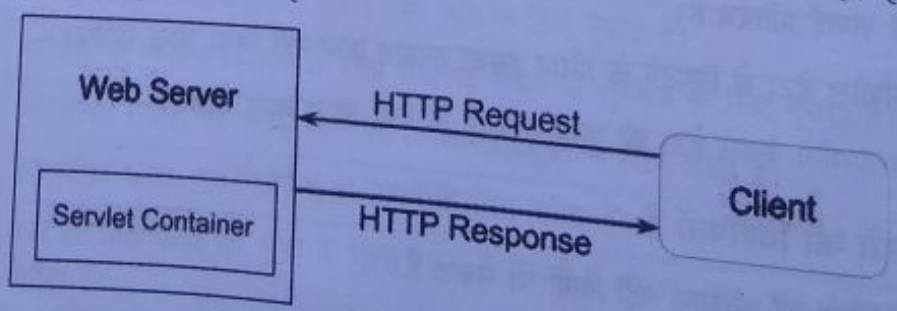
- इस रिक्वेस्ट को बुकमार्क नहीं किया जा सकता है।
- इस रिक्वेस्ट में डेटा की लंबाई पर कोई प्रतिबंध नहीं है।
- यह रिक्वेस्ट कभी cache नहीं किया जाता है।
- यह रिक्वेस्ट ब्राउज़र हिस्टरी में बरकरार नहीं है।



सर्वलेट कंटेनर

यह JavaEE (j2ee) अनुप्रयोगों के लिए रनटाइम वातावरण प्रदान करता है। क्लाइंट/यूजर सर्वर से केवल एक स्टेटिक वेबपेज का रिक्वेस्ट कर सकते हैं। यदि यूजर वेब पेजों को इनपुट के अनुसार पढ़ना चाहता है तो जावा में सर्वलेट कंटेनर का उपयोग किया जाता है।

सर्वलेट कंटेनर वेब सर्वर का हिस्सा है जिसे एक अलग प्रोसेस में चलाया जा सकता है। हम सर्वलेट कंटेनर स्टेट्स को तीन प्रकारों में वर्गीकृत कर सकते हैं—



सर्वलेट कंटेनर स्टेट्स

सर्वलेट कंटेनर वेब सर्वर का हिस्सा है जिसे एक अलग प्रोसेस में चलाया जा सकता है। हम सर्वलेट कंटेनर स्टेट्स को तीन प्रकारों में वर्गीकृत कर सकते हैं—

- **स्टैंडअलोन :** यह विशिष्ट जावा-आधारित सर्वर है जिसमें सर्वलेट कंटेनर और वेब सर्वर एक एकल प्रोग्राम का अभिन्न अंग हैं। उदाहरण के लिए—टॉमकैट खुद से चल रहा है
- **प्रोसेस में (In process) :** इसे वेब सर्वर से अलग किया जाता है, क्योंकि प्लग-इन के रूप में एक अलग प्रोग्राम मुख्य सर्वर के एड्रेस स्पेस में चलता है। उदाहरण के लिए—JBoss के अंदर चलने वाला टॉम्कट।
- **बाहर कि प्रोसेस (Out-of-process) :** वेब सर्वर और सर्वलेट कंटेनर एक अलग प्रोसेस में चलाए जाने वाले विभिन्न प्रोग्राम हैं। उनके बीच संचार करने के लिए, वेब सर्वर सर्वलेट कंटेनर द्वारा प्रदान किए गए प्लग-इन का उपयोग करता है।

सर्वलेट कंटेनर नीचे दिए गए कई ऑपरेशन करता है—

- लाइफ़ साइकल् का मैनेजमेंट
- ऑब्जेक्ट पूलिंग
- सुरक्षा आदि।

सर्वर

सर्वर एक डिवाइस या कम्प्यूटर प्रोग्राम है जो क्लाइंट के रूप में जाना जाने वाले अन्य प्रोग्राम द्वारा किए गए रिक्वेस्ट को स्वीकार और प्रतिक्रिया देता है। इसका उपयोग नेटवर्क रिसोर्सेज के प्रबंधन और सेवाओं को प्रदान करने वाले प्रोग्राम या सॉफ़्टवेयर को चलाने के लिए किया जाता है।

सर्वर दो टाइप के होते हैं:

1. वेब सर्वर
2. एप्लीकेशन सर्वर

वेब सर्वर

वेब सर्वर में केवल वेब या सर्वलेट कंटेनर होता है। इसका उपयोग सर्वलेट, JSP, JSF आदि के लिए किया जा सकता है। इसका उपयोग EJB के लिए नहीं किया जा सकता है।

यह एक ऐसा कम्प्यूटर है जहाँ वेब कंटेंट को संग्रहीत किया जा सकता है। सामान्य वेब सर्वर में वेब साइटों को होस्ट करने के लिए इस्तेमाल किया जा सकता है, लेकिन कुछ अन्य वेब सर्वरों जैसे कि FTP, ईमेल, स्टोरेज, गेमिंग आदि का भी उपयोग किया जाता है।

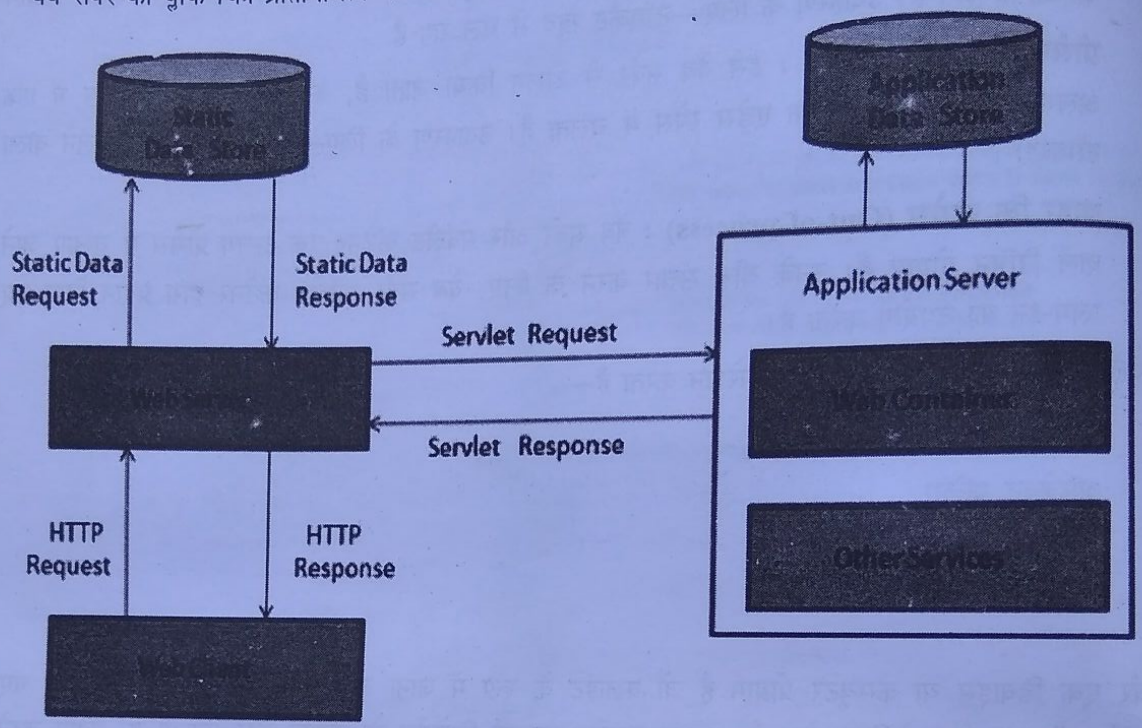
वेब सर्वर के उदाहरण हैं: अपाचे टोमैट और resin।

वेब सर्वर की वर्किंग

यह निम्नलिखित दो संभावित मेथड्स में से किसी भी क्लाइंट रिक्वेस्ट का जवाब दे सकता है—

- स्क्रिप्ट का उपयोग करके और डेटाबेस के साथ संवाद करके प्रतिक्रिया उत्पन्न करना।
- अनुरोधित URL से जुड़े क्लाइंट को फ़ाइल भेजना।

वेब सर्वर का ब्लॉक चित्र प्रतिनिधित्व नीचे दिखाया गया है—



महत्वपूर्ण बिंदु

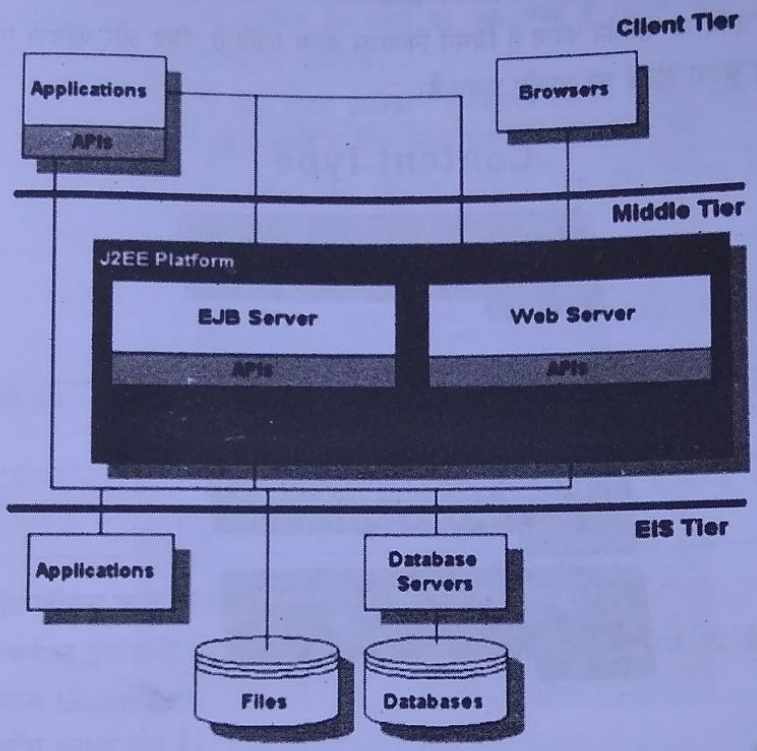
- यदि क्लाइंट पक्ष पर अनुरोधित वेब पेज नहीं मिला है, तो वेब सर्वर HTTP प्रतिक्रिया भेजेगा : error 404 NOT FOUND।
- जब वेब सर्वर अनुरोधित पृष्ठ खोज रहा है यदि अनुरोधित पृष्ठ पाया जाता है तो यह क्लाइंट को HTTP Response के साथ भेजेगा।
- यदि क्लाइंट कुछ अन्य रिसोर्सेज का रिक्वेस्ट करता है तो वेब सर्वर एप्लीकेशन सर्वर से संपर्क करेगा और HTTP प्रतिक्रिया के निर्माण के लिए डेटा स्टोर होगा।

एप्लीकेशन सर्वर

एप्लीकेशन सर्वर में वेब और EJB कंटेनर होते हैं। यह सर्वलेट, JSP, Struts, JSF, EJB आदि के लिए इस्तेमाल किया जा सकता है। यह एक घटक आधारित उत्पाद है जो सर्वर केंद्रित आर्किटेक्चर के मध्य-स्तरीय में निहित है।

यह दृढ़ता और डेटा एक्सेस के साथ स्टेट के रखरखाव और सुरक्षा के लिए मिडलवेयर सेवों प्रदान करता है। यह एक टाइप का सर्वर है जो आईटी सेवाओं, अंतिम यूजरस और संगठनों के लिए संबद्ध सेवाओं और अनुप्रयोगों को स्थापित, संचालित और होस्ट करने के लिए डिज़ाइन किया गया है।

एप्लीकेशन सर्वर का ब्लॉक चित्र प्रतिनिधित्व नीचे दिखाया गया है—



एप्लीकेशन सर्वर के उदाहरण हैं—

1. **JBoss :** JBoss ओपन-सोर्स सर्वर है।
2. **Glassfish :** सन माइक्रोसिस्टम द्वारा प्रदान किया गया। अब ओरेकल द्वारा अधिग्रहित।
3. **Weblogic :** ओरेकल द्वारा प्रदान किया गया। यह अधिक सुरक्षित है।
4. **Websphere :** IBM द्वारा प्रदान किया गया।

कंटेंट टाइप

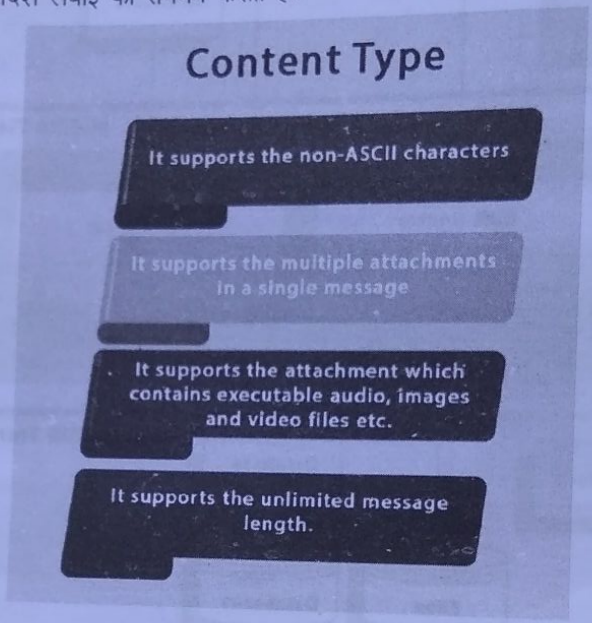
कंटेंट टाइप को MIME (Multipurpose internet Mail Extension) टाइप के रूप में भी जाना जाता है। यह एक HTTP हेडर है जो ब्राउज़र को आपके द्वारा भेजे जा रहे विवरण के बारे में बताता है।

MIME एक इंटरनेट मानक है जो एक संदेश में ध्वनियों, छवियों और पाठ के सम्मिलन की अनुमति देकर ईमेल की सीमित क्षमताओं को विस्तारित करने के लिए उपयोग किया जाता है।

ईमेल सेवाओं के लिए MIME द्वारा प्रदान की गई विशेषतों नीचे दी गई हैं—

- यह गैर-ASCII वर्णों का समर्थन करता है।

- यह एक संदेश में कई अटैचमेंट्स का समर्थन करता है।
- यह उस अटैचमेंट का समर्थन करता है जिसमें निष्पादन योग्य ऑडियो, चित्र और वीडियो फाइलें आदि हैं।
- यह असीमित संदेश लंबाई का समर्थन करता है।



कंटेंट प्रकारों की सूची

कई कंटेंट टाइप हैं। आमतौर पर प्रयुक्त कंटेंट टाइप नीचे दिए गए हैं—

- text/html
- text/plain
- application/msword
- application/vnd.ms-excel
- application/jar
- application/pdf
- application/octet-stream
- application/x-zip
- images/jpeg
- images/png
- images/gif
- audio/mp3
- video/mp4

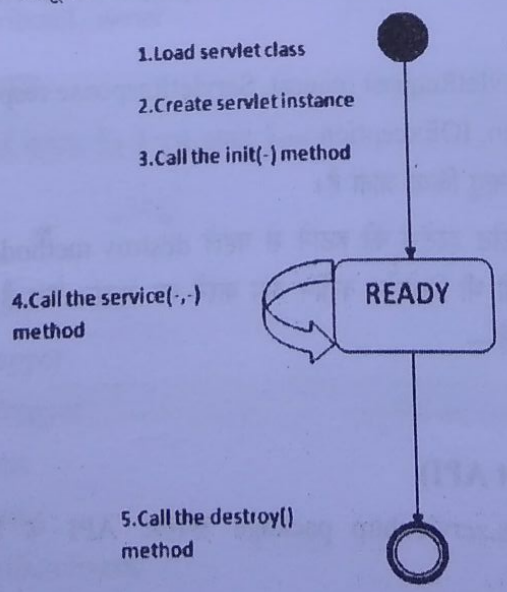
video/quicktime आदि।

सर्वलेट शब्दावली	विवरण
वेबसाइटः स्टेटिक बनाम डायनामिक	यह संबंधित वेब पृष्ठों का एक संग्रह है जिसमें पाठ, चित्र, ऑडियो और वीडियो हो सकते हैं।
एचटीटीपी	यह डेटा संचार प्रोटोकॉल है जिसका उपयोग क्लाइंट और सर्वर के बीच संचार स्थापित करने के लिए किया जाता है।
HTTP रिक्वेस्ट	यह कंप्यूटर द्वारा एक वेब सर्वर को भेजा गया रिक्वेस्ट है जिसमें सभी प्रकार की संभावित रोचक जानकारी होती है।
GET बनाम POST	यह GET और POST अनुरोध के बीच अंतर देता है।
Container	इसका उपयोग जावा में सर्वर साइड पर डायनामिक रूप से वेब पेज बनाने के लिए किया जाता है।
सर्वरः वेब बनाम एप्लीकेशन	इसका उपयोग नेटवर्क रिसोर्सेज के प्रबंधन और सेवाओं को प्रदान करने वाले प्रोग्राम या सॉफ़्टवेयर को चलाने के लिए किया जाता है।
कंटेंट टाइप	यह HTTP हैडर है जो आपको ब्राउज़र में भेजने के बारे में विवरण प्रदान करता है।

3.3 एक सर्वलेट का जीवन चक्र (Life cycle of a servlet)

वेब कंटेनर एक सर्वलेट इंस्टेंस के जीवन चक्र को बनाए रखता है। आइए देखें सर्वलेट का जीवन चक्र—

1. सर्वलेट क्लास loaded है।
2. सर्वलेट इंस्टेंस बनाया गया है।
3. init method को लागू किया जाता है।
4. service method लागू किया जाता है।
5. destroy method को लागू किया जाता है।



जैसा कि ऊपर चित्र में प्रदर्शित किया गया है, एक सर्वलेट के तीन स्टेट हैं—new, ready और end। यदि सर्वलेट इंस्टेंस बनाया गया है तो सर्वलेट नए स्टेट में है। Init () मेथड को लागू करने के बाद, सर्वलेट ready अवस्था में आता है। ready अवस्था में सर्वलेट सभी कार्य करता है। जब वेब कंटेनर destroy मेथड को लागू करता है, तो यह end स्टेट में शिफ्ट हो जाता है।

(1) सर्वलेट क्लास loaded है।

क्लास लोडर सर्वलेट क्लास को लोड करने के लिए जिम्मेदार है। वेब कंटेनर द्वारा सर्वलेट के लिए पहला रिक्वेस्ट प्राप्त होने पर सर्वलेट क्लास को लोड किया जाता है।

(2) सर्वलेट इंस्टेंस बनाया गया है।

वेब कंटेनर सर्वलेट क्लास को लोड करने के बाद एक सर्वलेट का उदाहरण बनाता है। सर्वलेट का उदाहरण सर्वलेट जीवन चक्र में केवल एक बार बनाया जाता है।

(3) init method को लागू किया जाता है।

वेब कंटेनर सर्वलेट आवृत्ति बनाने के बाद केवल एक बार init मेथड को कॉल करता है। इनलेट मेथड का उपयोग सर्वलेट को इनिशियलाइज़ करने के लिए किया जाता है। यह javax.servlet.Servlet इंटरफ़ेस का जीवन चक्र method है। Init मेथड का सिंटैक्स नीचे दिया गया है:

```
public void init(ServletConfig config) throws ServletException
```

(4) service method लागू किया जाता है।

जब सर्वलेट के लिए रिक्वेस्ट प्राप्त होता है तो वेब कंटेनर हर बार service मेथड को कॉल करता है। यदि सर्वलेट को प्रारंभ नहीं किया गया है, तो यह पहले तीन चरणों का पालन करता है जैसा कि पहले वर्णित है, फिर service मेथड को कॉल करता है। यदि सर्वलेट इनिशियलाइज़ किया जाता है, तो यह सर्विस मेथड call होता है। ध्यान दें कि सर्वलेट केवल एक बार आरंभिक होता है। सर्वलेट इंटरफ़ेस की service मेथड का सिंटैक्स नीचे दिया गया है—

```
public void service(ServletRequest request, ServletResponse response)
throws ServletException, IOException
```

(5) destroy method को लागू किया जाता है।

वेब कंटेनर service से सर्वलेट इंस्टेंस को हटाने से पहले destroy method को कॉल करता है। यह सर्वलेट को स्मृति, थ्रेड आदि के लिए किसी भी रिसोर्सेज क्लीन अप करने का अवसर देता है। सर्वलेट इंटरफेस की destroy मेथड का सिंटेक्स नीचे दिया गया है—

```
public void destroy()
```

3.4 सर्वलेट एपीआई (Servlet API)

Javax.servlet और javax.servlet.http package सर्वलेट API के लिए इंटरफेस और क्लास का प्रतिनिधित्व करते हैं।

Javax.servlet पैकेज में कई इंटरफेस और क्लास हैं जो सर्वलेट या वेब कंटेनर द्वारा उपयोग की जाती हैं। ये किसी भी प्रोटोकॉल के लिए विशिष्ट नहीं हैं।

Javax.servlet.http पैकेज में इंटरफेस और क्लास शामिल हैं जो केवल http अनुरोधों के लिए जिम्मेदार हैं।

आइए देखें कि javax.servlet पैकेज के इंटरफेस क्या हैं।

Javax.servlet पैकेज में इंटरफेस

Javax.servlet पैकेज में कई इंटरफेस हैं। वे इस प्रकार हैं—

1. Servlet
2. ServletRequest
3. ServletResponse
4. RequestDispatcher
5. ServletConfig
6. ServletContext
7. SingleThreadModel
8. Filter
9. FilterConfig
10. FilterChain
11. ServletRequestListener
12. ServletRequestAttributeListener
13. ServletContextListener
14. ServletContextAttributeListener

Javax.servlet पैकेज में क्लास

Javax.servlet पैकेज में कई क्लास हैं। वे इस प्रकार हैं—

1. GenericServlet
2. ServletInputStream
3. ServletOutputStream
4. ServletRequestWrapper
5. ServletResponseWrapper
6. ServletRequestEvent
7. ServletContextEvent
8. ServletRequestAttributeEvent
9. ServletContextAttributeEvent

10. ServletException
11. UnavailableException

Javax.servlet.http पैकेज में इंटरफेस

Javax.servlet.http पैकेज में कई इंटरफेस हैं। वे इस प्रकार हैं—

1. HttpServletRequest
2. HttpServletResponse
3. HttpSession
4. HttpSessionListener
5. HttpSessionAttributeListener
6. HttpSessionBindingListener
7. HttpSessionActivationListener
8. HttpSessionContext (deprecated now)
9. HttpSessionContext (Deye heoekevele)

Javax.servlet.http पैकेज में क्लास

Javax.servlet.http पैकेज में कई क्लास हैं। वे इस प्रकार हैं—

1. HttpServlet
2. Cookie
3. HttpServletRequestWrapper
4. HttpServletResponseWrapper
5. HttpSessionEvent
6. HttpSessionBindingEvent
7. HttpUtils (deprecated now)

3.5 सर्वलेट इंटरफ़ेस (Servlet interface)

सर्वलेट इंटरफ़ेस सभी सर्वलेट को सामान्य व्यवहार प्रदान करता है। सर्वलेट इंटरफ़ेस उन मेथड्स को परिभाषित करता है जिन्हें सभी सर्वलेट को लागू करना चाहिए।

सर्वलेट इंटरफ़ेस को किसी भी सर्वलेट (या तो प्रत्यक्ष या अप्रत्यक्ष रूप से) बनाने के लिए लागू करने की आवश्यकता है। यह 3 जीवन चक्र मेथड्स प्रदान करता है जिनका उपयोग सर्वलेट को शुरू करने, अनुरोधों को पूरा करने और सर्वलेट और 2 गैर-जीवन चक्र मेथड्स को नष्ट करने के लिए किया जाता है।

सर्वलेट इंटरफ़ेस के मेथड्स

सर्वलेट इंटरफ़ेस में 5 मेथड्स हैं। Init, service और destroy सर्वलेट के जीवन चक्र के मेथड्स हैं। ये वेब कंटेनर द्वारा मंगवाए गए हैं।

मेथड	विवरण
public void init(ServletConfig config)	सर्वलेट को प्रारंभ करता है। यह सर्वलेट का जीवन चक्र मेथड है और इसे केवल एक बार वेब कंटेनर द्वारा लागू किया जाता है।
public void service(ServletRequest request,ServletResponse response)	आने वाले रिक्वेस्ट के लिए प्रतिक्रिया प्रदान करता है। यह वेब कंटेनर द्वारा प्रत्येक रिक्वेस्ट पर लगाया जाता है।
public void destroy()	केवल एक बार invoke किया गया है और दर्शाता है कि सर्वलेट नष्ट हो रहा है।
public ServletConfig getServletConfig()	ServletConfig का object लौटाता है।
public String getServletInfo()	सर्वलेट के बारे में जानकारी देता है जैसे कि लेखक, कॉपीराइट, संस्करण आदि।

आइए सर्वलेट के सरल उदाहरण को सर्वलेट इंटरफेस को लागू करके देखें।

```
import java.io.*;
import javax.servlet.*;

public class First implements Servlet{
ServletConfig config=null;

public void init(ServletConfig config){
this.config=config;
System.out.println("servlet is initialized");
}

public void service(ServletRequest req,ServletResponse res)
throws IOException,ServletException{

res.setContentType("text/html");

PrintWriter out=res.getWriter();
out.print("<html><body>");
out.print("<b>hello simple servlet</b>");
out.print("</body></html>");
```

```
}
public void destroy(){System.out.println("servlet is destroyed");}
public ServletConfig getServletConfig(){return config;}
public String getServletInfo(){return "copyright 2007-1010";}

}
```

3.6 जेनरिक सर्वलेट क्लास (Generics Servlet class)

GenericServlet क्लास Servlet, ServletConfig और Serializable इंटरफेस को इम्प्लीमेंट करता है। यह service मेथड को छोड़कर इन इंटरफेस के सभी मेथड्स को लागू करता है।

GenericServlet क्लास किसी भी प्रकार के रिक्वेस्ट को संभाल सकता है इसलिए यह प्रोटोकॉल-indepedent है।

आप GenericServlet क्लास inherit करके और service मेथड को लागू करके एक सामान्य सर्वलेट बना सकते हैं।

GenericServlet क्लास के मेथड्स

GenericServlet क्लास में कई मेथड्स हैं। वे इस प्रकार हैं—

1. **public void init(ServletConfig config)**—सर्वलेट इनिशियलाइज़ करने के लिए उपयोग किया जाता है।

2. **public abstract void service (Servlet Request request, ServletResponse response)**—आने वाले रिक्वेस्ट के लिए service प्रदान करता है। यूजर द्वारा किसी सर्वलेट के लिए रिक्वेस्ट करने पर इसे हर बार लागू किया जाता है।

3. **public void destroy()**—पूरे जीवन चक्र में केवल एक बार invoke किया गया है और यह दर्शाता है कि सर्वलेट नष्ट हो रहा है।

4. **public ServletConfig getServletConfig()**—ServletConfig का object लौटाता है।

5. **public String getServletInfo()**—सर्वलेट के बारे में जानकारी देता है जैसे कि लेखक, कॉपीराइट, संस्करण आदि।

6. **public void init()**—यह सर्वलेट प्रोग्रामर के लिए एक सुविधाजनक मेथड है, अब super.init(config) को कॉल करने की आवश्यकता नहीं है।

7. **public ServletContext getServletContext()**—ServletContext का object लौटाता है।

8. **public String getInitParameter(String name)**—दिए गए पैरामीटर नाम के लिए पैरामीटर मान लौटाता है।

9. **public Enumeration getInitParameterNames()**—web.xml फ़ाइल में परिभाषित सभी पैरामीटर देता है।

10. **public String getServletName()**—सर्वलेट ऑब्जेक्ट का नाम देता है।

11. **public void log(String msg)**—दिए गए संदेश को सर्वलेट लॉग फ़ाइल में लिखते हैं।

12. **public void log(String msg,Throwable t)**—सर्वलेट लॉग फ़ाइल और स्टैक ट्रेस में व्याख्यात्मक संदेश लिखता है।

आइए GenericServlet क्लास को inherit करके सर्वलेट का सरल उदाहरण देखें।

```
import java.io.*;
import javax.servlet.*;

public class First extends GenericServlet{
public void service(ServletRequest req,ServletResponse res)
throws IOException,ServletException{

res.setContentType("text/html");

PrintWriter out=res.getWriter();
out.print("<html><body>");
out.print("<b>hello generic servlet</b>");
out.print("</body></html>");

}
}
```

3.7 Http सर्वलेट क्लास

HttpServlet क्लास जेनेरिकसर्वलेट क्लास का विस्तार करता है और सीरियल इंटरफ़ेस लागू करता है। यह http विशिष्ट मेथड्स प्रदान करता है जैसे कि doGet, doPost, doHead, doTrace आदि।

HttpServlet क्लास के मेथड्स

HttpServlet क्लास में कई मेथड्स हैं। वे इस प्रकार हैं—

1. public void service(ServletRequest req,ServletResponse res)
2. protected void service(HttpServletRequest req, HttpServletResponse res)
3. protected void doGet(HttpServletRequest req, HttpServletResponse res)
4. protected void doPost(HttpServletRequest req, HttpServletResponse res)
5. protected void doHead(HttpServletRequest req, HttpServletResponse res)

6. protected void doOptions(HttpServletRequest req, HttpServletResponse res)
7. protected void doPut(HttpServletRequest req, HttpServletResponse res)
8. protected void doTrace(HttpServletRequest req, HttpServletResponse res)
9. protected void doDelete(HttpServletRequest req, HttpServletResponse res)
10. protected long getLastModified(HttpServletRequest req)

3.8 RequestDispatcher

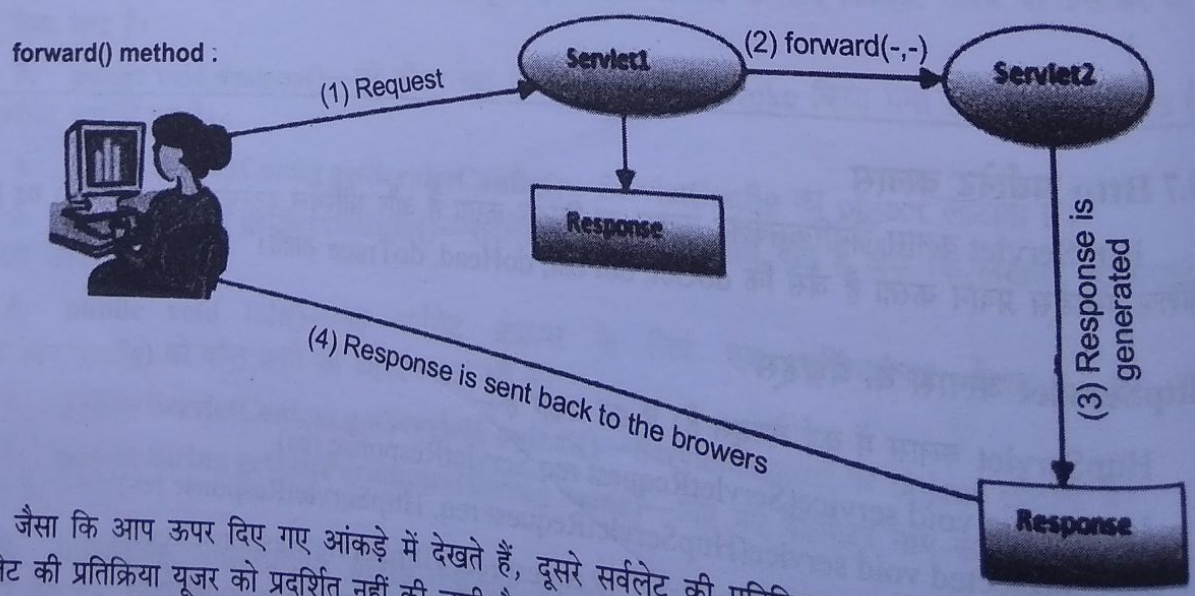
RequestDispatcher इंटरफ़ेस रिक्वेस्ट को किसी अन्य रिसोर्सेज पर भेजने की सुविधा प्रदान करता है जो कि html, सर्वलेट या jsp हो सकता है। इस इंटरफ़ेस का उपयोग किसी अन्य रिसोर्सेज की कंटेंट को भी शामिल करने के लिए किया जा सकता है। यह सर्वलेट सहयोग का एक मेथड है।

RequestDispatcher इंटरफ़ेस में दो मेथड्स परिभाषित हैं।

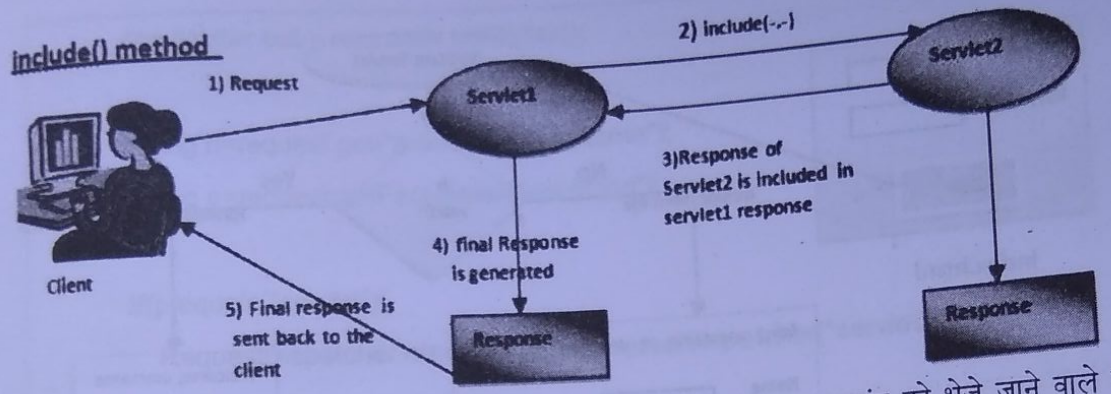
RequestDispatcher इंटरफ़ेस के मेथड्स

RequestDispatcher इंटरफ़ेस दो मेथड्स प्रदान करता है। वो हैं—

1. **public void forward(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException :** सर्वर पर एक सर्वलेट से दूसरे रिसोर्सेज (सर्वलेट, JSP फ़ाइल, या HTML फ़ाइल) के लिए रिक्वेस्ट भेजता है।
2. **public void include(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException :** प्रतिक्रिया में एक रिसोर्सेज (सर्वलेट, JSP पृष्ठ, या HTML फ़ाइल) की कंटेंट शामिल है।



जैसा कि आप ऊपर दिए गए आंकड़े में देखते हैं, दूसरे सर्वलेट की प्रतिक्रिया क्लाइंट को भेजी जाती है। पहले सर्वलेट की प्रतिक्रिया यूजर को प्रदर्शित नहीं की जाती है।



जैसा कि आप ऊपर दिए गए चित्र में देख सकते हैं, दूसरे सर्वलेट की प्रतिक्रिया क्लाइंट को भेजे जाने वाले पहले सर्वलेट की प्रतिक्रिया में शामिल है।

RequestDispatcher का OBJECT कैसे प्राप्त करें?

GetRequestDispatcher () ServletRequest इंटरफ़ेस का मेथड RequestDispatcher का उद्देश्य देता है।

सिंटैक्सः

GetRequestDispatcher मेथड का सिंटैक्स

```
public RequestDispatcher getRequestDispatcher(String resource);
```

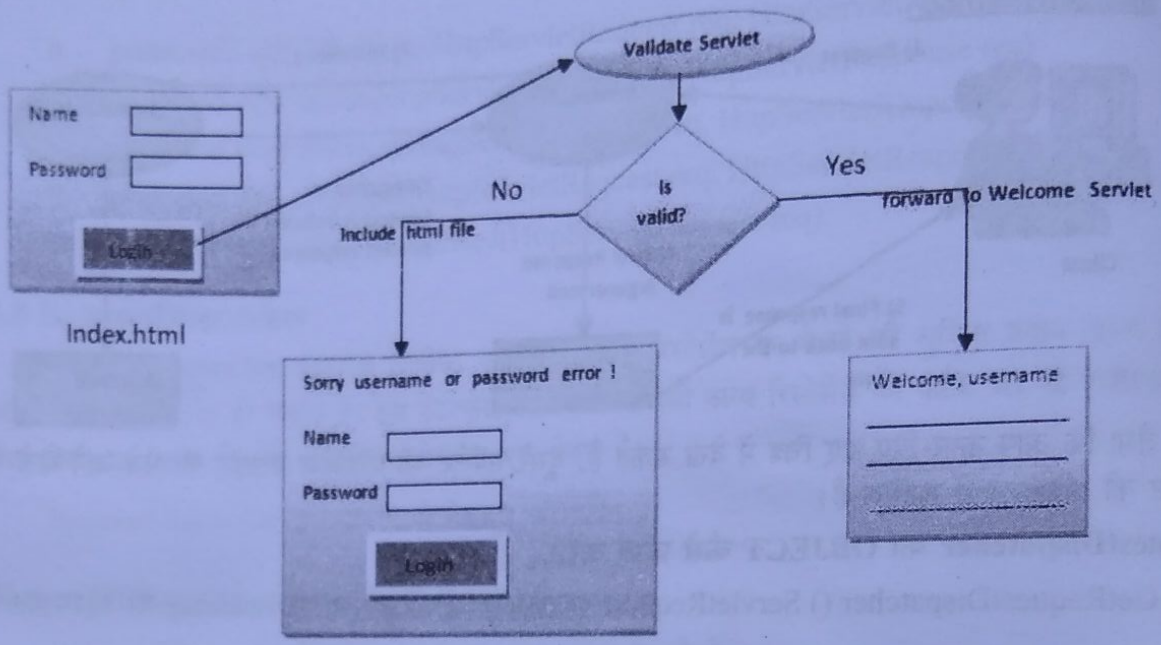
GetRequestDispatcher मेथड का उपयोग करने का उदाहरण

```
RequestDispatcher rd=request.getRequestDispatcher('servlet2');
//servlet2 is the url-pattern of the second servlet

rd.forward(request, response);//method may be include or forward
```

RequestDispatcher इंटरफ़ेस का उदाहरण

इस उदाहरण में, हम यूजर द्वारा दर्ज किए गए पासवर्ड को मान्य कर रहे हैं। यदि पासवर्ड सर्वलेट है, तो यह WelcomeServlet के रिक्वेस्ट को फॉरवर्ड करेगा, अन्यथा त्रुटि संदेश दिखाई देगा sorry username or password error! देगा। इस प्रोग्राम में, हम हार्डकोड जानकारी के लिए चेक कर रहे हैं। इस उदाहरण में, हमने निम्नलिखित फाइलें बनाई हैं—

- index.html file : यूजर से इनपुट प्राप्त करने के लिए।
- Login.java file : प्रतिक्रिया को संसाधित करने के लिए एक सर्वलेट क्लास। यदि पासवर्ड सर्वलेट है, तो यह स्वागत सर्वलेट के लिए रिक्वेस्ट फॉरवर्ड करेगा।
- WelcomeServlet.java file : स्वागत संदेश प्रदर्शित करने के लिए सर्वलेट क्लास।
- web.xml file : एक तैनाती डिस्क्रिप्टर फ़ाइल जिसमें सर्वलेट के बारे में जानकारी है।



index.html

```
<form action="servlet1" method="post">
Name:<input type="text" name="userName"/><br/>
Password:<input type="password" name="userPass"/><br/>
<input type="submit" value="login"/>
</form>
```

Login.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;


public class Login extends HttpServlet {

public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

  response.setContentType("text/html");
```

```
        PrintWriter out = response.getWriter();

        String n=request.getParameter("userName");
        String p=request.getParameter("userPass");

        if(p.equals("servlet"){
          RequestDispatcher rd=request.getRequestDispatcher("servlet2");
          rd.forward(request, response);
        }
        else{
          out.print("Sorry UserName or Password Error!");
          RequestDispatcher rd=request.getRequestDispatcher("/index.html");
          rd.include(request, response);

        }
      }

    }
```

WelcomeServlet.java

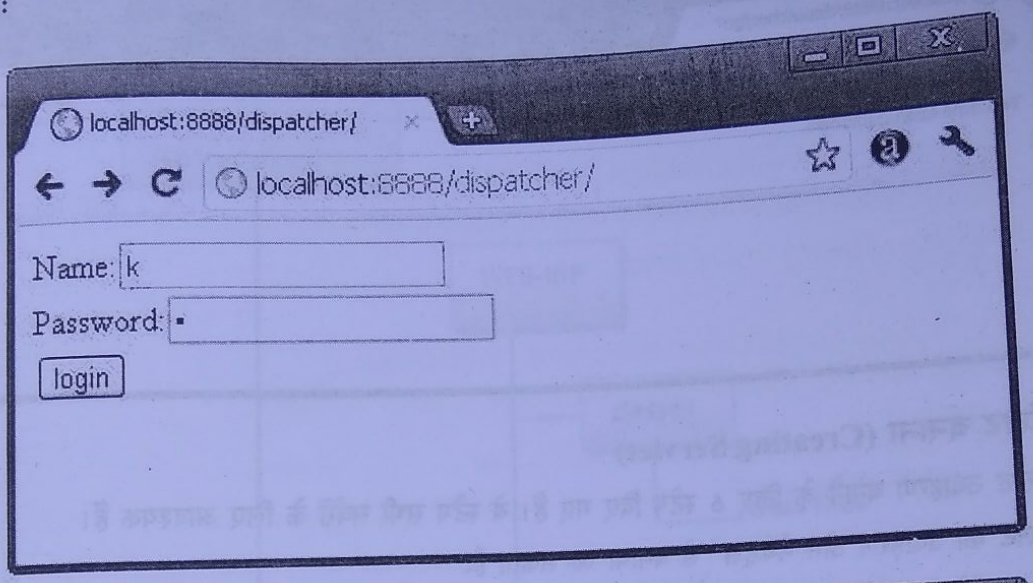
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class WelcomeServlet extends HttpServlet {

  public void doPost(HttpServletRequest request, HttpServletResponse response)

    throws ServletException, IOException {
```

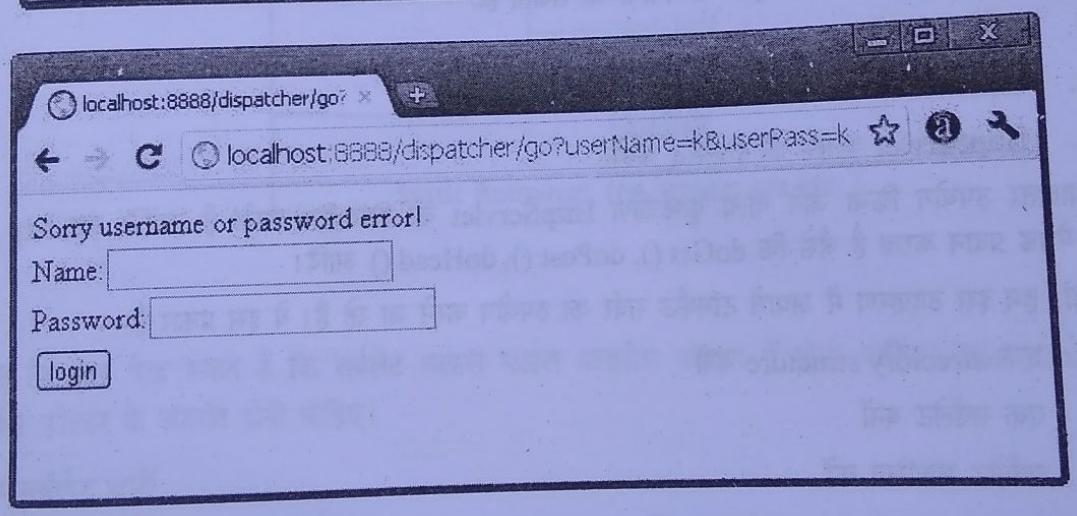
```
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        String n=request.getParameter("userName");
        out.print("Welcome "+n);
        }
    }
```

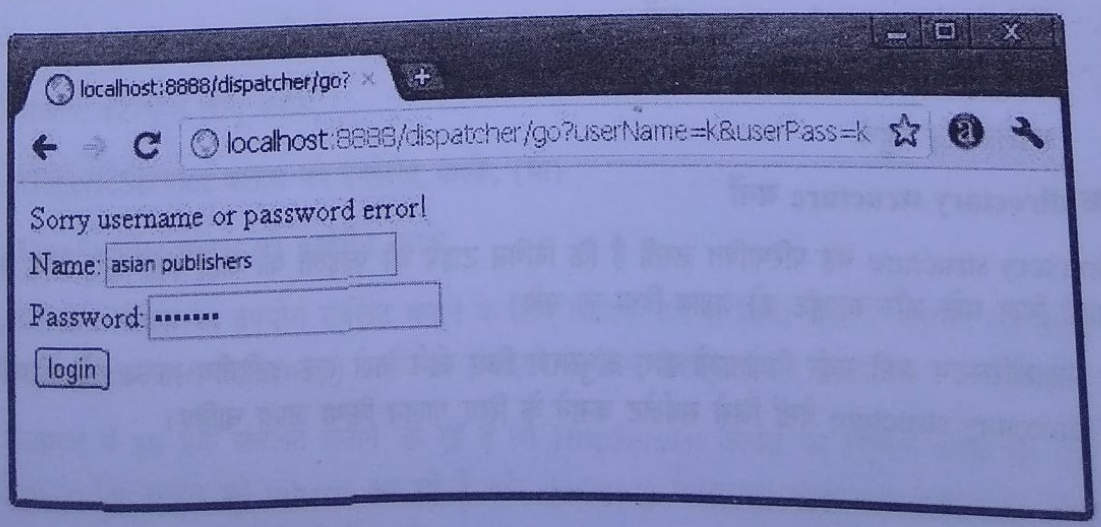
web.xml

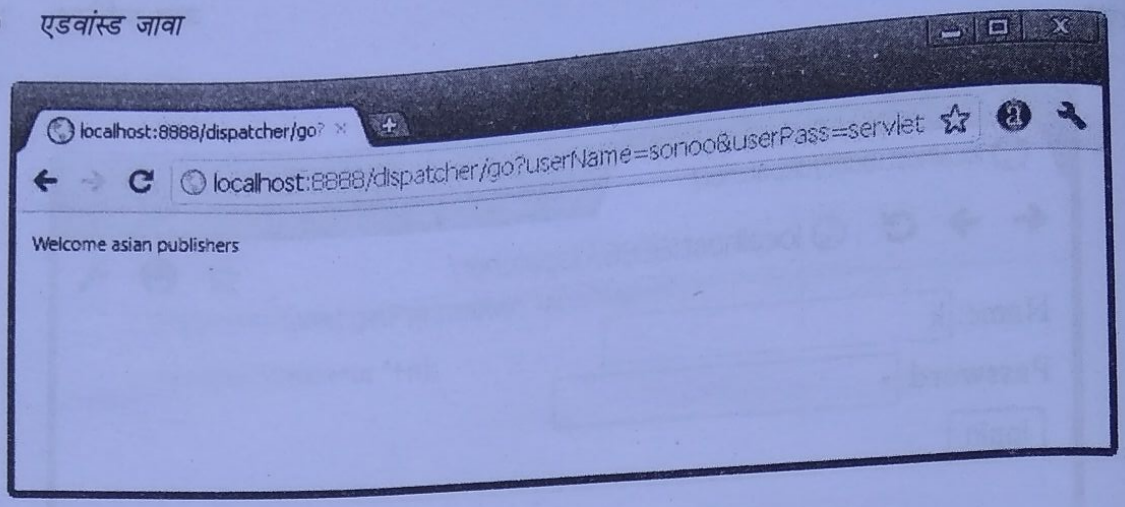
```
<web-app>
 <servlet>
   <servlet-name>Login</servlet-name>
   <servlet-class>Login</servlet-class>
 </servlet>
 <servlet>
   <servlet-name>WelcomeServlet</servlet-name>
   <servlet-class>WelcomeServlet</servlet-class>
 </servlet>

 <servlet-mapping>
   <servlet-name>Login</servlet-name>
   <url-pattern>/servlet1</url-pattern>
 </servlet-mapping>
 <servlet-mapping>
   <servlet-name>WelcomeServlet</servlet-name>
   <url-pattern>/servlet2</url-pattern>
 </servlet-mapping>

 <welcome-file-list>
   <welcome-file>index.html</welcome-file>
 </welcome-file-list>
</web-app>
```

आउटपुटः


localhost:8888/dispatcher/
localhost:8888/dispatcher/
Name: k
Password:
login


localhost:8888/dispatcher/go?
localhost:8888/dispatcher/go?userName=k&userPass=k
Sorry username or password error!
Name:
Password:
login


localhost:8888/dispatcher/go?
localhost:8888/dispatcher/go?userName=k&userPass=k
Sorry username or password error!
Name: asian publishers
Password: •••••••
login



3.9 सर्वलेट बनाना (Creating Servlet)

सर्वलेट उदाहरण बनाने के लिए 6 स्टेप दिए गए हैं। ये स्टेप सभी सर्वरों के लिए आवश्यक हैं।

सर्वलेट का उदाहरण तीन मेथड्स से बनाया जा सकता है:

1. सर्वलेट इंटरफ़ेस लागू करके,
2. GenericServlet क्लास को इनहेरिट् करके, (या)
3. HttpServlet क्लास को इनहेरिट् करके

ज्यादातर उपयोग किया जाने वाला दृष्टिकोण HttpServlet को विस्तारित करके है क्योंकि यह http रिक्वेस्ट विशिष्ट मेथड प्रदान करता है जैसे कि doGet (), doPost (), doHead () आदि।

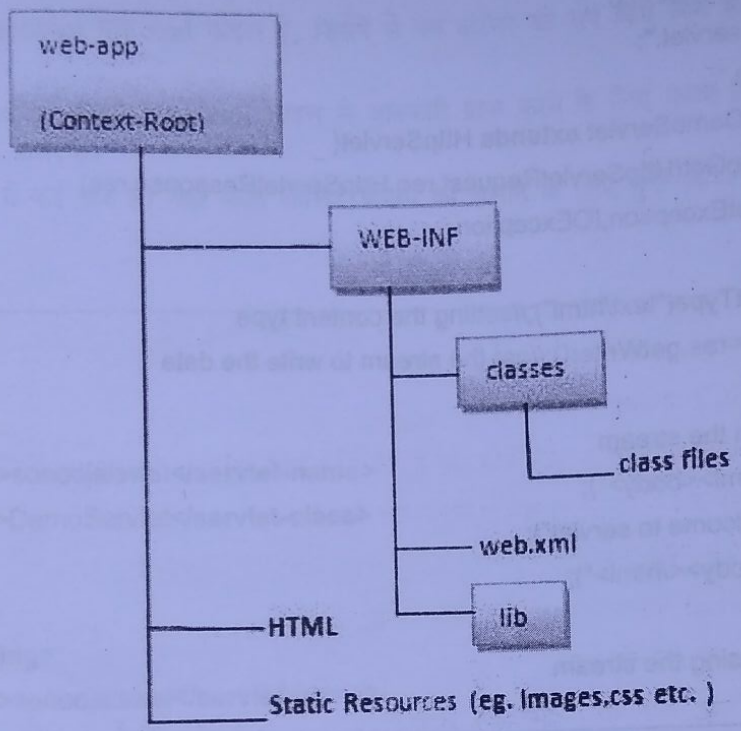
यहां, हम इस उदाहरण में अपाचे टॉमकैट सर्वर का उपयोग करने जा रहे हैं। वे इस प्रकार हैं—

1. एक directory structure बनों
2. एक सर्वलेट बनों
3. सर्वलेट संकलित करें
4. एक deployment descriptor बनों
5. सर्वर शुरू करें और प्रोजेक्ट को deploy करें
6. सर्वलेट तक पहुंचें

(1) एक directory structure बनों

Directory structure यह परिभाषित करती है कि विभिन्न टाइप की फ़ाइलों को कहाँ रखा जाए ताकि वेब कंटेनर को जानकारी मिल सके और क्लाइंट को जवाब दिया जा सके।

सन माइक्रोसिस्टम सभी सर्वर विक्रेताओं द्वारा अनुसरण किए जाने वाले एक अद्वितीय मानक को परिभाषित करता है। आइए directory structure देखें जिसे सर्वलेट बनाने के लिए पालन किया जाना चाहिए।



जैसा कि आप देख सकते हैं कि सर्वलेट क्लास फाइल क्लासेस फोल्डर में होनी चाहिए। Web.xml फ़ाइल WEB-INF फ़ोल्डर के अंतर्गत होनी चाहिए।

(2) एक सर्वलेट बनों

सर्वलेट बनाने के तीन मेथड्स हैं।

1. सर्वलेट इंटरफ़ेस लागू करके,
2. GenericServlet क्लास को इनहेरिट् करके, (या)
3. HttpServlet क्लास को इनहेरिट् करके

HttpServlet क्लास का उपयोग सर्वलेट बनाने के लिए व्यापक रूप से किया जाता है क्योंकि यह http अनुरोधों जैसे कि doGet (), doPost, doHead () आदि को संभालने के लिए मेथड्स प्रदान करता है।

इस उदाहरण में हम एक सर्वलेट बनाने जा रहे हैं जो HttpServlet क्लास का विस्तार करता है। इस उदाहरण में, हम HttpServlet क्लास को inherit कर रहे हैं और doGet () मेथड का कार्यान्वयन प्रदान कर रहे हैं। सूचना जो रिक्वेस्ट प्राप्त होती है वह डिफ़ॉल्ट रिक्वेस्ट है।

DemoServlet.java

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class DemoServlet extends HttpServlet{
public void doGet(HttpServletRequest req,HttpServletResponse res)
throws ServletException,IOException
{
res.setContentType("text/html");//setting the content type
PrintWriter pw=res.getWriter();//get the stream to write the data

//writing html in the stream
pw.println("<html><body>");
pw.println("Welcome to servlet");
pw.println("</body></html>");

pw.close();//closing the stream
}}
```

(3) सर्वलेट संकलित करें

सर्वलेट को संकलित करने के लिए, jar फ़ाइल को लोड करना आवश्यक है। विभिन्न सर्वर अलग jar फाइलें प्रदान करते हैं—

jar फ़ाइल	सर्वर
(1) servlet-api.jar	Apache Tomcat
(2) weblogic.jar	Weblogic
(3) javaee.jar	Glassfish
(4) javaee.jar	JBoss

jar फ़ाइल लोड करने के दो मेथड्स

1. सेट क्लासपाथ
2. JRE / lib / ext फ़ोल्डर में jar फ़ाइल पेस्ट करें

जावा फाइल को किसी भी फोल्डर में रखें। जावा फ़ाइल को संकलित करने के बाद, वेबलेट की क्लास फ़ाइल को WEB-INF/classes डायरेक्टरी में पेस्ट करें।

(4) Deployment descriptor (web.xml फ़ाइल) बनों

Deployment descriptor एक xml फ़ाइल है, जिसमें से वेब कंटेनर को सर्व किए जाने के बारे में जानकारी मिलती है।

वेब कंटेनर, Parser का उपयोग web.xml फ़ाइल से जानकारी प्राप्त करने के लिए करता है। SAX, DOM और Pull जैसे कई xml पार्सर हैं।

Web.xml फ़ाइल में कई तत्व हैं। यहाँ सरल सर्वलेट प्रोग्राम को चलाने के लिए कुछ आवश्यक तत्व दिए गए हैं।

web.xml फ़ाइल

```
<web-app>

<servlet>
<servlet-name>sonoojaiswal</servlet-name>
<servlet-class>DemoServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>sonoojaiswal</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>
</web-app>
```

Web.xml फ़ाइल के तत्वों का विवरण

Web.xml फ़ाइल में बहुत सारे तत्व हैं। यहाँ कुछ तत्वों का चित्रण है जो उपरोक्त web.xml फ़ाइल में उपयोग किया जाता है। तत्व इस प्रकार हैं—

<web-app> पूरे आवेदन का प्रतिनिधित्व करता है।

<servlet> <web-app> का उप तत्व है और सर्वलेट का प्रतिनिधित्व करता है।

<servlet-name> <servlet> का उप तत्व है, सर्वलेट के नाम का प्रतिनिधित्व करता है।

<servlet-class> <servlet> का उप तत्व है, सर्वलेट के क्लास का प्रतिनिधित्व करता है।

<servlet-mapping> <web-app> का उप तत्व है। इसका उपयोग सर्वलेट को मैप करने के लिए किया जाता है।

<url-pattern> < servlet-mapping > का उप तत्व है। इस पैटर्न का उपयोग क्लाइंट साइड पर सर्वलेट को लागू करने के लिए किया जाता है।

(5) सर्वर शुरू करें और प्रोजेक्ट को depoly करें

Apache Tomcat सर्वर को शुरू करने के लिए, apache-tomcat/bin directory के तहत स्टार्टअप पर डबल क्लीक करें।

Apache Tomcat Server के लिए वन टाइम कॉन्फिगरेशन

आपको 2 कार्य करने होंगे—

1. Environment variable में JAVA_HOME या JRE_HOME सेट करें (यह सर्वर शुरू करने के लिए आवश्यक है)।

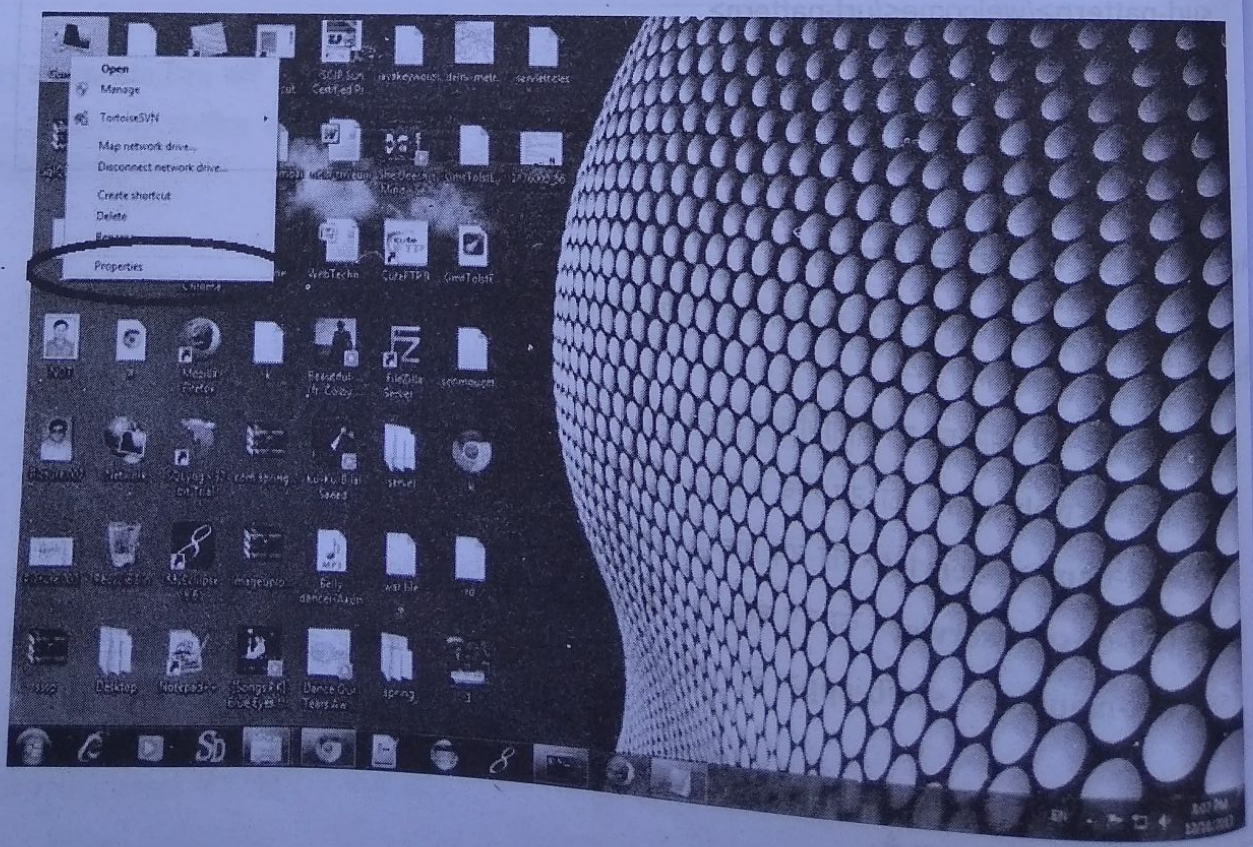
2. टॉमकैट (वैकल्पिक) के पोर्ट नंबर को बदलें। यदि किसी अन्य सर्वर समान पोर्ट (8080) पर चल रहा है, तो इसकी आवश्यकता है।

(1) Environment variable में JAVA_HOME कैसे सेट करें?

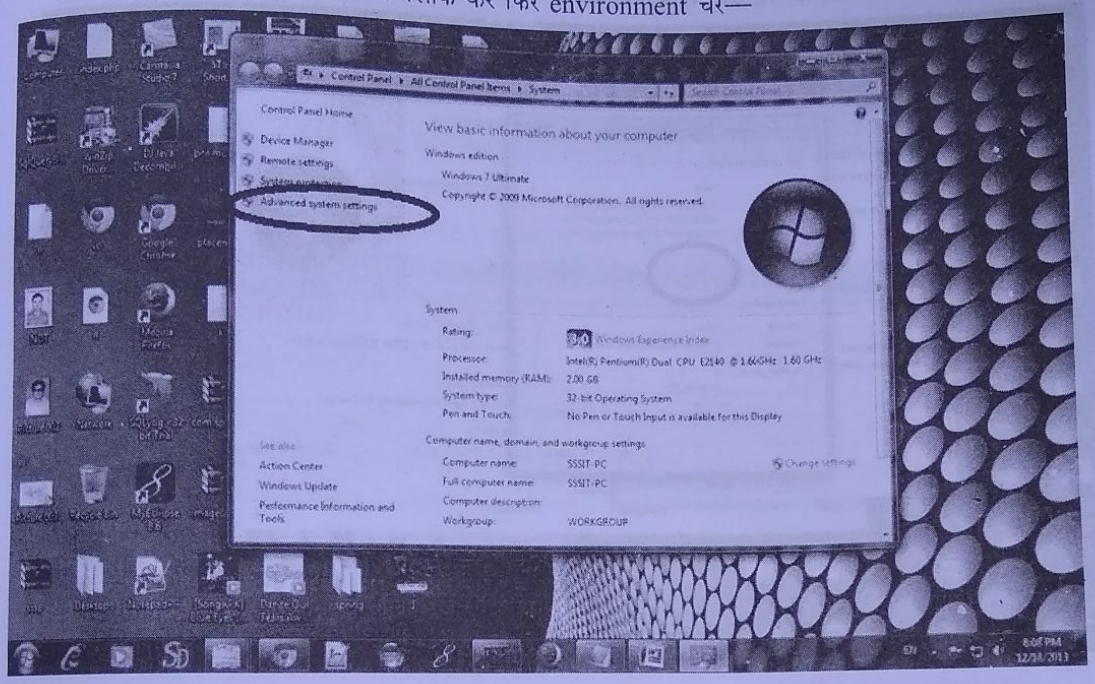
Apache Tomcat सर्वर शुरू करने के लिए JAVA_HOME और JRE_HOME को environment चर में सेट किया जाना चाहिए।

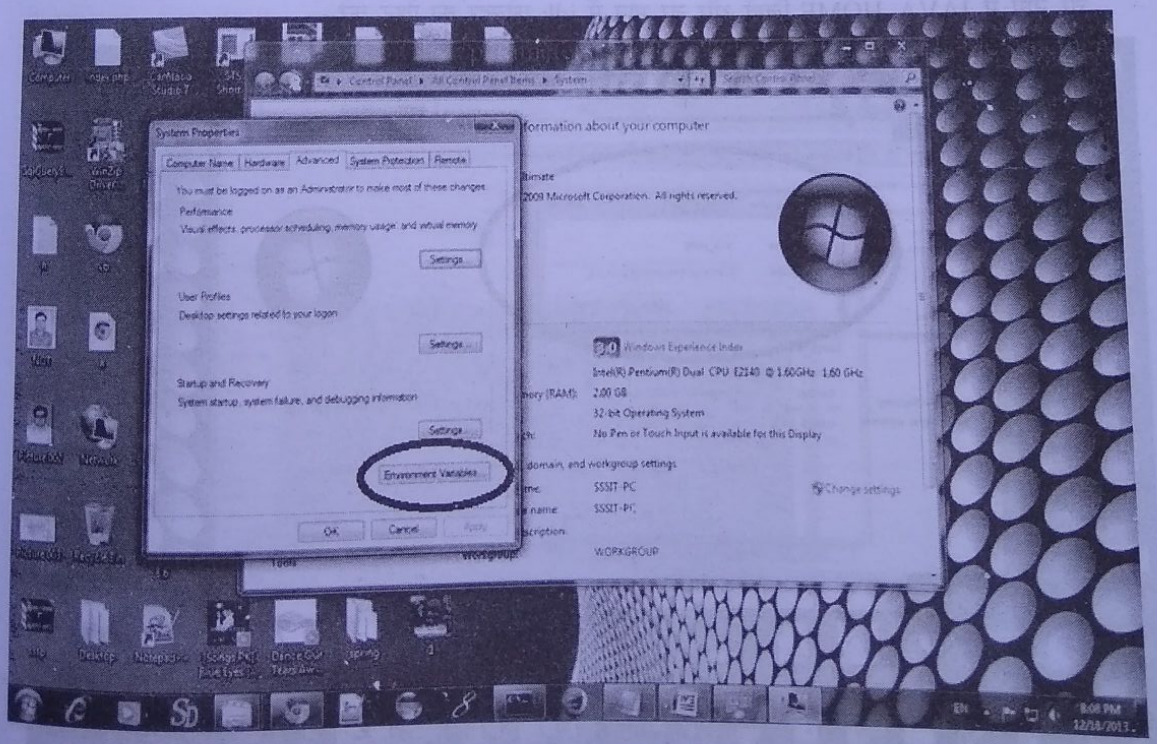
My Computer properties पर जाएं -> advanced tab पर क्लीक करें फिर environment variables -> user variable के नए टैब पर क्लीक करें -> चर नाम में JAVA_HOME लिखें और चर मान में jdk फ़ोल्डर का पथ पेस्ट करें -> ok -> ok - ok .

My Computer properties पर जों—

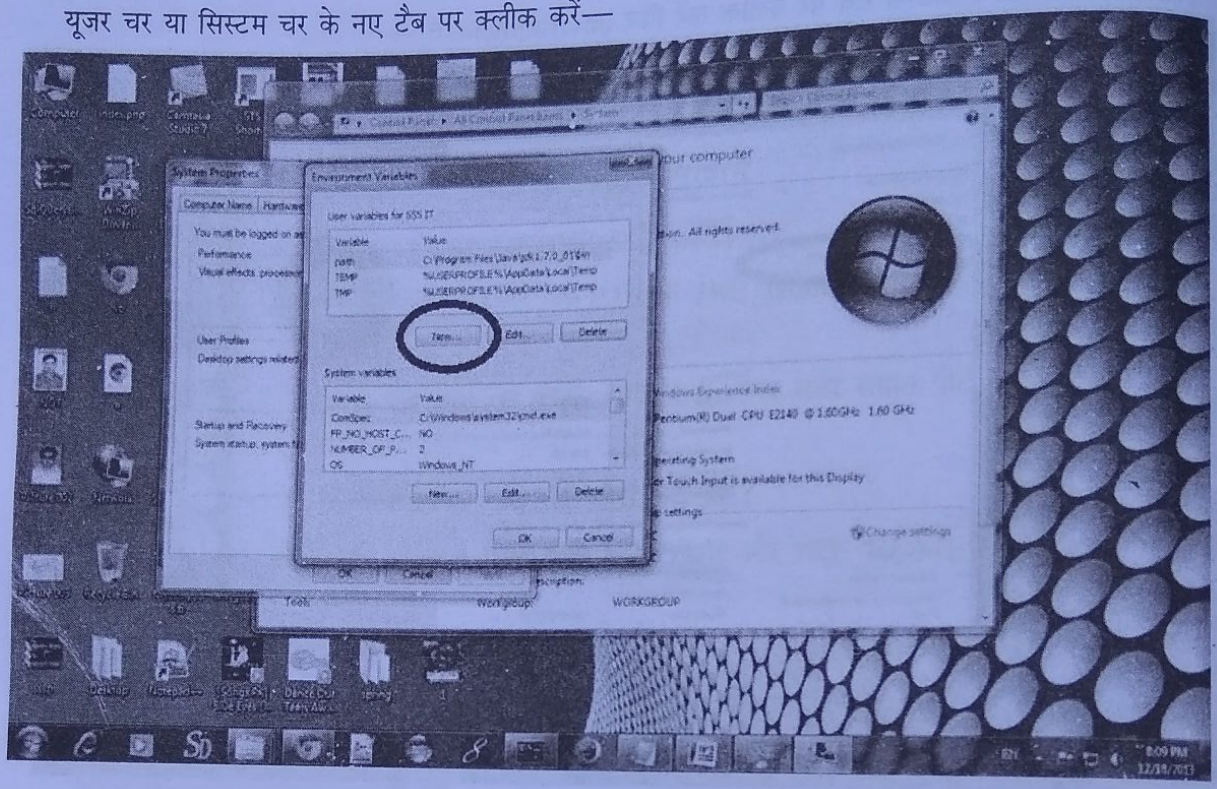


Advanced सिस्टम सेटिंग्स टैब पर क्लीक करें फिर environment चर—

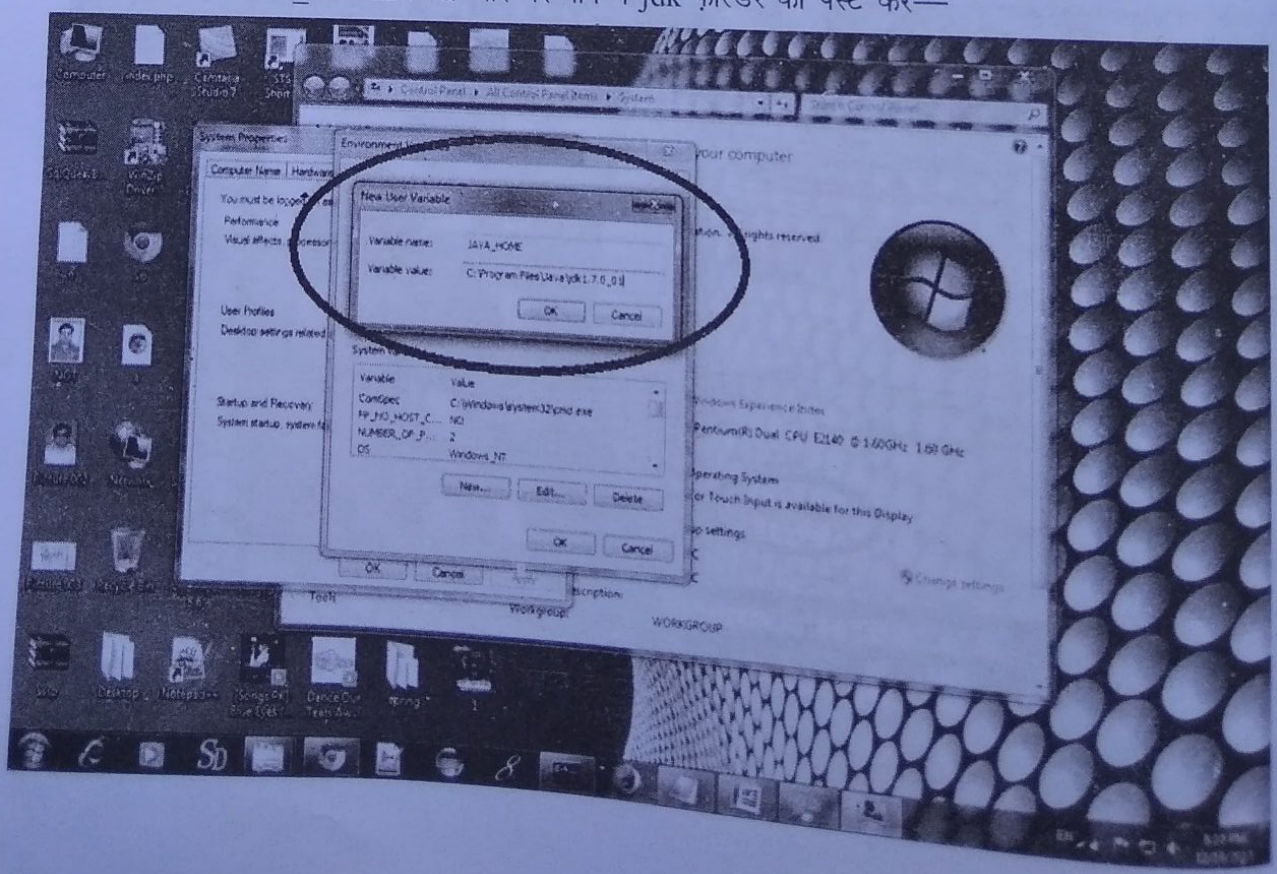




यूजर चर या सिस्टम चर के नए टैब पर क्लीक करें—



चर नाम में JAVA_HOME लिखें और चर मान में jdk फ़ोल्डर का पेस्ट करे—



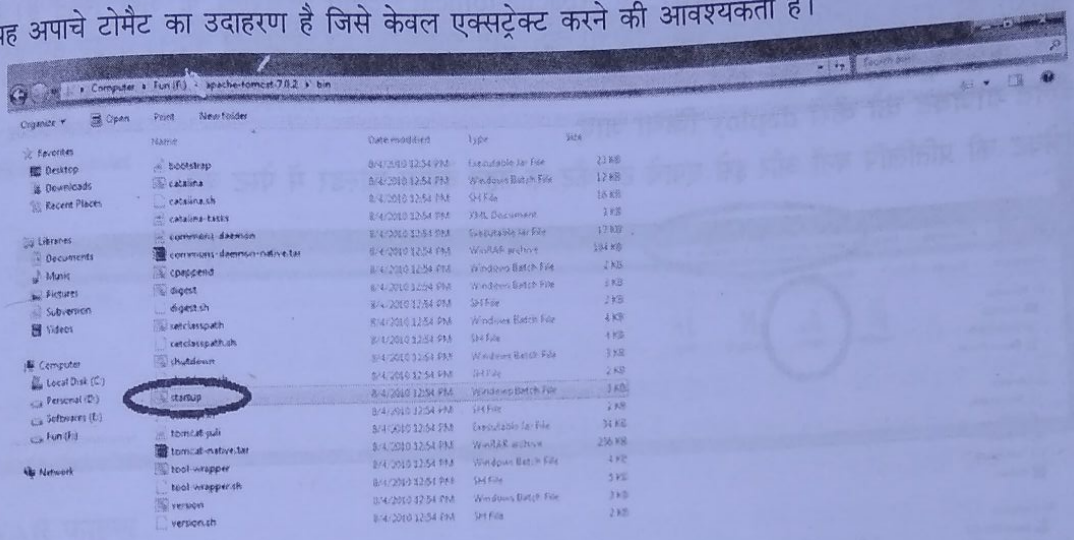
पथ के अंत में अर्धविराम (;) नहीं होना चाहिए।

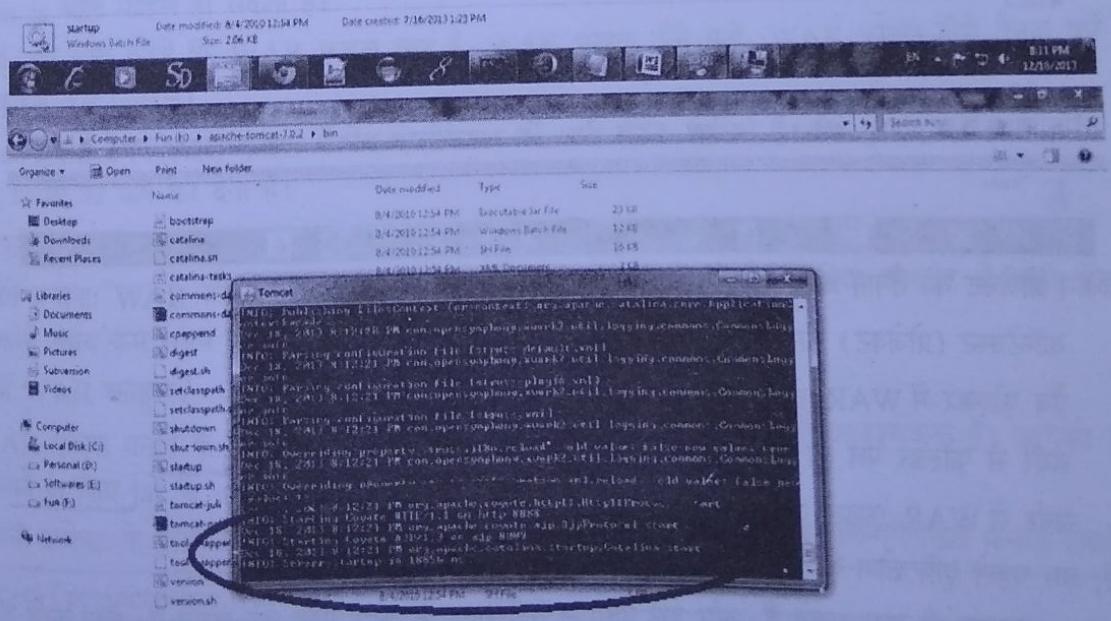
JAVA_HOME सेट करने के बाद apache tomcat / bin ceW startup.bat फाइल पर डबल क्लीक करें।

नोट—दो टाइप के टॉमकैट उपलब्ध हैं—

1. Apache tomcat जिसे केवल एक्सट्रेक्ट करने आवश्यकता है (इंस्टॉल करने की आवश्यकता नहीं है)
2. Apache tomcat जिसे इंस्टॉल करना है

यह अपाचे टोमैट का उदाहरण है जिसे केवल एक्सट्रेक्ट करने की आवश्यकता है।





अब सर्वर सफलतापूर्वक शुरू किया गया है।

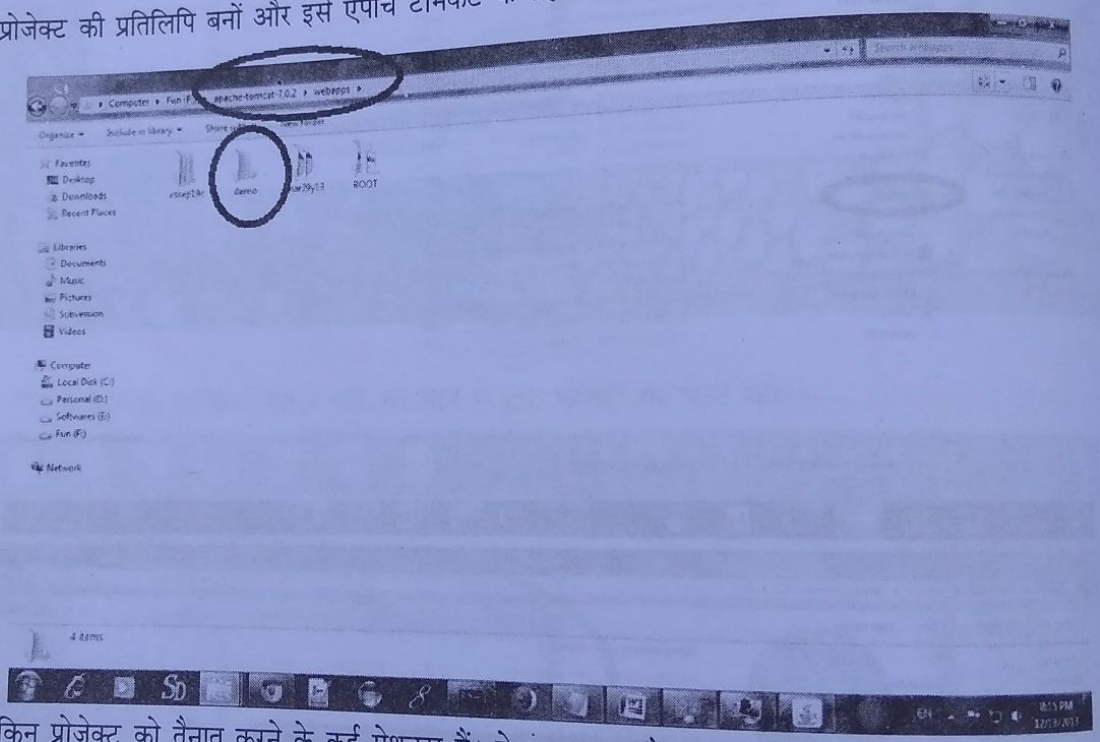
Apache tomcat की पोर्ट संख्या कैसे बदलें?

पोर्ट नंबर को बदलने की आवश्यकता होती है यदि उसी पोर्ट नंबर के साथ एक ही सिस्टम पर एक और सर्वर चल रहा हो। यदि आपने ऑर्कल स्थापित किया है, तो आपको अपाचे टॉमकट के पोर्ट नंबर को बदलने की आवश्यकता है क्योंकि दोनों में डिफ़ॉल्ट पोर्ट संख्या 8080 है।

नोटपैड में server.xml फ़ाइल खोलें। यह Apache-tomcat / conf निर्देशिका के अंदर स्थित है। कनेक्टर पोर्ट = 8080 को बदलें और 8080 के बजाय 8080 को किसी चार अंकों की संख्या से बदलें। आइए हम इसे 9999 से बदल दें और इस फाइल को सेव करें।

(5) सर्वलेट प्रोजेक्ट को कैसे deploy किया जाए

प्रोजेक्ट की प्रतिलिपि बनों और इसे एपाचे टॉमकैट के तहत वेबैप्स फ़ोल्डर में पेस्ट करें।



लेकिन प्रोजेक्ट को तैनात करने के कई मेथड्स हैं। वे इस प्रकार हैं—

- कॉन्टेक्स्ट (प्रोजेक्ट) फ़ोल्डर को वेब डायरेक्टरी में कॉपी करके
- वेब फ़ोल्डर में WAR फ़ोल्डर की प्रतिलिपि बनाकर
- सर्वर से फ़ोल्डर पथ का चयन करके
- सर्वर से WAR फ़ाइल का चयन करके

यहां, हम पहले दृष्टिकोण का उपयोग कर रहे हैं।

आप war फ़ाइल भी बना सकते हैं, और इसे वेबएप्स निर्देशिका के अंदर पेस्ट कर सकते हैं। ऐसा करने के लिए आपको WAR फ़ाइल बनाने के लिए jar टूल का उपयोग करना होगा। प्रोजेक्ट निर्देशिका (WEB-INF से पहले) के अंदर जों, फिर लिखें—

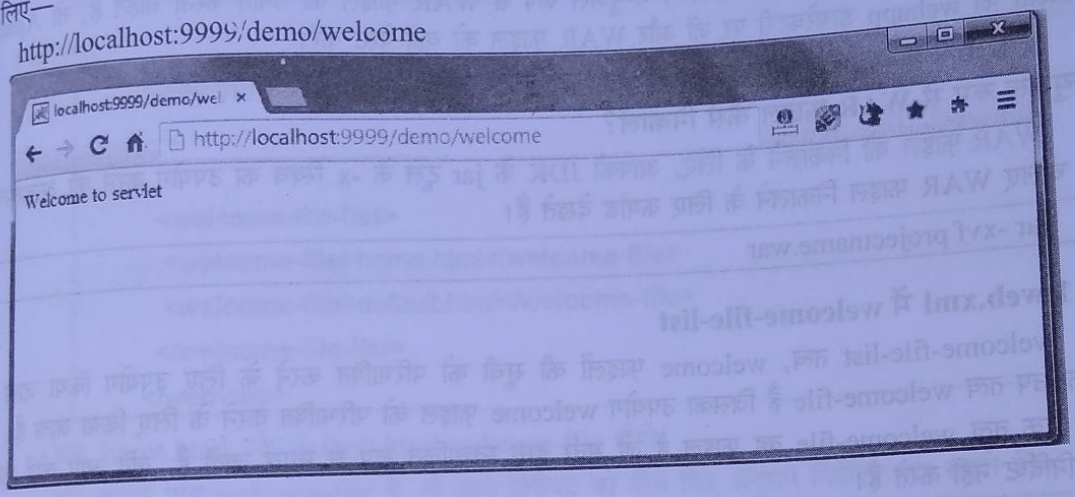
```
projectfolder> jar cvf myproject.war *
```

WAR फ़ाइल बनाने का एक फायदा है कि प्रोजेक्ट को एक स्थान से दूसरे स्थान पर ले जाने में कम समय लगता है।

(6) सर्वलेट का उपयोग कैसे करें

Browser खोलें और http: // hostname: portno / referenceroot / urlpatternofservlet लिखें। उदाहरण के लिए—

http://localhost:9999/demo/welcome



3.10 WAR फ़ाइल

एक WAR (वेब संग्रह) फ़ाइल में एक वेब प्रोजेक्ट की फाइलें होती हैं। इसमें सर्वलेट, xml, jsp, image, html, css, js आदि फाइलें हो सकती हैं।

यहां, हम चर्चा करेंगे कि WAR फ़ाइल क्या है, WAR फ़ाइल कैसे बनों, WAR फ़ाइल कैसे तैनात करें और WAR फ़ाइल कैसे निकालें।

Web archive (WAR) फ़ाइल में वेब एप्लीकेशन की सभी कंटेंट शामिल है। यह फ़ाइल को स्थानांतरित करने की समय अवधि को कम कर देता है।

WAR फ़ाइल का लाभ

समय बचाना: WAR की फाइल सभी फाइलों को एक इकाई में जोड़ती है। इसलिए क्लाइंट से सर्वर में फाइल ट्रांसफर करते समय कम समय लगता है।

कैसे बनों WAR फाइल?

WAR फ़ाइल बनाने के लिए, आपको JDK के jar टूल का उपयोग करने की आवश्यकता है। आपको WAR फ़ाइल बनाने के लिए jar के -c स्विच का उपयोग करने की आवश्यकता है।

अपनी प्रोजेक्ट के प्रोजेक्ट डायरेक्टरी (WEB-INF के बाहर) के अंदर जों, फिर निम्नलिखित कमांड लिखें—

```
jar -cvf projectname.war *
```

यहाँ, -c का उपयोग फ़ाइल बनाने के लिए किया जाता है, और वह वर्बोज आउटपुट फाइल को उत्पन्न करने के लिए और -f archive फाइल नाम specify करता है।

* (तारांकन) प्रतीक दर्शाता है कि इस निर्देशिका की सभी फाइलें (उप निर्देशिका सहित)।

WAR फाइल कैसे तैनात करें?

WAR फाइल को तैनात करने के दो मेथड्स हैं।

1. सर्वर कंसोल पैनल द्वारा
2. मैन्युअल रूप से सर्वर के विशिष्ट फ़ोल्डर में WAR फ़ाइल होने से।

यदि आप Apache tomcat सर्वर में मैन्युअल रूप से WAR फ़ाइल को तैनात करना चाहते हैं, तो Apache tomcat की webapp डायरेक्टरी पर जों और WAR फ़ाइल को यहाँ पेस्ट करें।

अब, आप ब्राउज़र के माध्यम से वेब प्रोजेक्ट को एक्सेस करने में सक्षम हैं।

मैन्युअल रूप से WAR फ़ाइल कैसे निकालें?

WAR फ़ाइल को निकालने के लिए, आपको JDK के jar टूल के -x स्विच का उपयोग करने की आवश्यकता है। चलिए WAR फाइल निकालने के लिए कमांड देखते हैं।

```
jar -xvf projectname.war
```

3.11 web.xml में welcome-file-list

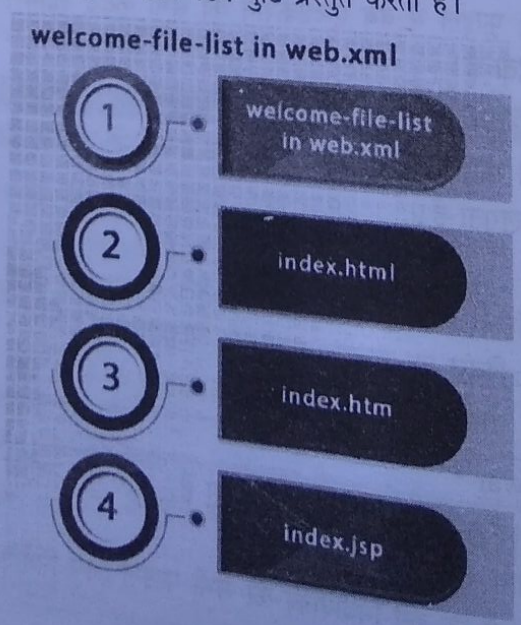
welcome-file-list तत्व, welcome फ़ाइलों की सूची को परिभाषित करने के लिए उपयोग किया जाता है। इसका उप तत्व welcome-file है जिसका उपयोग welcome फ़ाइल को परिभाषित करने के लिए किया जाता है।

एक तत्व welcome-file वह फ़ाइल है जो सर्वर द्वारा स्वचालित रूप से मंगाई जाती है, यदि आप कोई फ़ाइल नाम निर्दिष्ट नहीं करते हैं।

डिफ़ॉल्ट सर्वर निम्नलिखित क्रम में Welcome फ़ाइल को देखता है—

1. web.xml ceW welcome-file-list
2. index.html
3. index.htm
4. index.jsp

यदि इनमें से कोई भी फ़ाइल नहीं मिली, तो सर्वर 404 त्रुटि प्रस्तुत करता है।



यदि आपने web.xml में welcome-file निर्दिष्ट की है, और सभी फ़ाइलें index.html, index.htm और index.jsp मौजूद हैं, तो प्राथमिकता welcome-file में जाती है।

अगर welcome-file-list प्रविष्टि web.xml फ़ाइल में मौजूद नहीं है, तो प्राथमिकता index.html फ़ाइल और फिर index.html और अंत में index.jsp फ़ाइल में जाती है।

आइए देखें वेब.xml फ़ाइल जो welcome-file को परिभाषित करती है।

web.xml

```
<web-app>
....

<welcome-file-list>
<welcome-file>home.html</welcome-file>
<welcome-file>default.html</welcome-file>
</welcome-file-list>
</web-app>
```

अब, home.html और default.html welcome-file होंगी।

यदि आपके पास welcome-file है, तो आप प्रोजेक्ट को नीचे दिए अनुसार निर्देशित कर सकते हैं—

```
http: // localhost: 8888 / MyProject
```

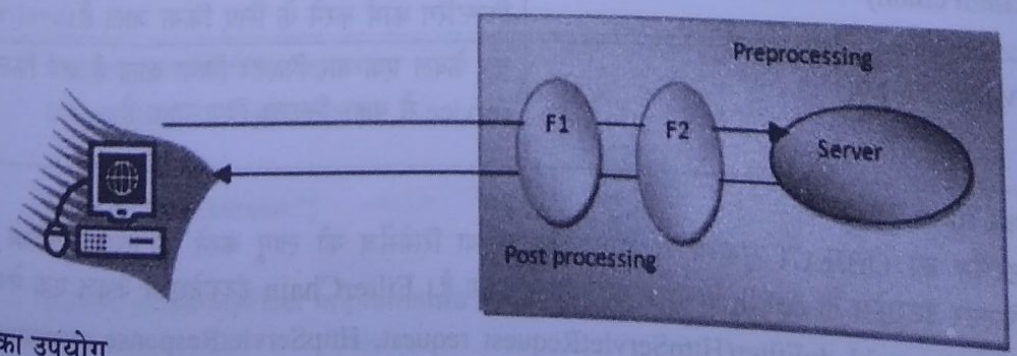
3.12 सर्वलेट फिल्टर (Servlet Filter)

एक फिल्टर एक वस्तु है जिसे रिक्वेस्ट के प्रीप्रोसेसिंग और पोस्टप्रोसेसिंग पर लगाया जाता है।

यह मुख्य रूप से रूपांतरण, लॉगिंग, compression, एन्क्रिप्शन और डिक्रिप्शन, इनपुट सत्यापन आदि जैसे फ़िल्टरिंग कार्य करने के लिए उपयोग किया जाता है।

सर्वलेट फिल्टर प्लग करने योग्य है, अर्थात इसका प्रवेश web.xml फ़ाइल में परिभाषित किया गया है, अगर हम web.xml फ़ाइल से फिल्टर की प्रविष्टि को हटा देते हैं, तो फिल्टर स्वचालित रूप से हटा दिया जाएगा और हमें सर्वलेट को बदलने की आवश्यकता नहीं है।

इसलिए रखरखाव लागत कम होगी।



फिल्टर का उपयोग

- सभी आने वाले अनुरोधों को रिकॉर्ड करना

- उन कंप्यूटरों के IP एड्रेस लॉग करता है जिनसे रिक्वेस्ट उत्पन्न होते हैं
- Conversion
- Data compression
- एन्क्रिप्शन और डिक्रिप्शन
- इनपुट सत्यापन आदि

फिल्टर का लाभ

1. फिल्टर प्लग करने योग्य है।
2. एक फिल्टर दूसरे रिसोर्सेज पर निर्भरता नहीं है।
3. कम रखरखाव

फिल्टर एपीआई

जैसे सर्वलेट फिल्टर का अपना API होता है। Javax.servlet पैकेज में फिल्टर API के तीन इंटरफ़ेस शामिल हैं।

1. Filter
2. FilterChain
3. FilterConfig

(1) फिल्टर इंटरफ़ेस

किसी भी फिल्टर को बनाने के लिए, आपको फिल्टर इंटरफ़ेस लागू करना होगा। फिल्टर इंटरफ़ेस एक फिल्टर के लिए जीवन चक्र के मेथड्स प्रदान करता है।

मेथड	विवरण
public void init(FilterConfig config)	init () मेथड केवल एक बार मंगाया जाता है। इसका उपयोग फिल्टर को इनिशियलाइज़ करने के लिए किया जाता है।
public void doFilter(HttpServletRequest request,HttpServletResponse response, FilterChain chain)	doFilter () मेथड को हर बार तब प्रयोग किया जाता है जब यूजर किसी भी रिसोर्सेज के लिए अनुरोध करता है, जिसके लिए फिल्टर मैप किया जाता है। इसका उपयोग फिल्टरिंग कार्य करने के लिए किया जाता है।
public void destroy()	यह केवल एक बार फिल्टर किया जाता है जब फिल्टर को service से बाहर निकाल दिया जाता है।

(2) FilterChain इंटरफ़ेस

फिल्टरचैन का OBJECT श्रृंखला में अगले फिल्टर या रिसोर्सेज को लागू करने के लिए जिम्मेदार है। यह ऑब्जेक्ट फिल्टर इंटरफ़ेस के doFilter मेथड में पारित हो गया है। FilterChain इंटरफ़ेस में केवल एक मेथड है—

1. public void doFilter(HttpServletRequest request, HttpServletResponse response) : यह नियंत्रण को अगले फिल्टर या रिसोर्सेज में भेजता है।

फिल्टर को कैसे परिभाषित करें

हम फिल्टर को सर्वलेट के रूप में परिभाषित कर सकते हैं। आइए फिल्टर और फिल्टर-मैपिंग के तत्वों को देखें।

```
<<web-app>

<filter>
<filter-name>...</filter-name>
<filter-class>...</filter-class>
</filter>

<filter-mapping>
<filter-name>...</filter-name>
<url-pattern>...</url-pattern>
</filter-mapping>

</web-app>
```

मैपिंग फिल्टर के लिए हम url- पैटर्न या सर्वलेट-नाम का उपयोग कर सकते हैं। Url-pattern तत्वों का सर्वलेट-नाम तत्व पर एक लाभ है अर्थात इसे सर्वलेट, JSP या HTML पर लागू किया जा सकता है।

फिल्टर का सरल उदाहरण

इस उदाहरण में, हम केवल जानकारी प्रदर्शित कर रहे हैं कि फिल्टर रिक्वेस्ट के पोस्ट प्रोसेसिंग के बाद स्वचालित रूप से लागू होता है।

index.html

```
<a href='servlet1'> यहां क्लीक करें </a>
```

MyFilter.java

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.*;

public class MyFilter implements Filter{

public void init(FilterConfig arg0) throws ServletException {}
```

```
public void doFilter(ServletRequest req, ServletResponse resp,
    FilterChain chain) throws IOException, ServletException {

    PrintWriter out=resp.getWriter();
    out.print("filter is invoked before");

    chain.doFilter(req, resp);//sends request to next resource

    out.print("filter is invoked after");
    }
    public void destroy() {}
}
```

HelloServlet.java

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.*;

public class HelloServlet extends HttpServlet {
  public void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    out.print("<br>welcome to servlet<br>");

  }

}
```

web.xml

फिल्टर को परिभाषित करने के लिए, वेब-ऐप के फिल्टर तत्व को सर्वलेट की तरह परिभाषित किया जाना चाहिए।

```
<web-app>

<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>HelloServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

<filter>
<filter-name>f1</filter-name>
<filter-class>MyFilter</filter-class>
</filter>

<filter-mapping>
<filter-name>f1</filter-name>
<url-pattern>/servlet1</url-pattern>
</filter-mapping>

</web-app>
```

Authentication फिल्टर

हम फिल्टर में Authentication कर सकते हैं। यहां, हम फिल्टर क्लास में यूजर द्वारा दिए गए पासवर्ड की जांच करने जा रहे हैं, यदि दिया गया पासवर्ड admin है, तो यह WelcomeAdmin सर्वलेट के रिक्वेस्ट को फॉरवर्ड करेगा अन्यथा यह त्रुटि संदेश प्रदर्शित करेगा।

फिल्टर का उपयोग करके यूजर को प्रमाणित करने का उदाहरण

आइए फिल्टर का उपयोग करके यूजर को प्रमाणित करने का सरल उदाहरण देखें।

यहां, हमने 4 फाइलें बनाई हैं—

- index.html

- MyFilter.java
- AdminServlet.java
- web.xml

index.html

```
<form action="servlet1">
Name:<input type="text" name="name"/><br/>
Password:<input type="password" name="password"/><br/>

<input type="submit" value="login">

</form>
```

MyFilter.java

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.*;

public class MyFilter implements Filter{

public void init(FilterConfig arg0) throws ServletException {}

public void doFilter(ServletRequest req, ServletResponse resp,
    FilterChain chain) throws IOException, ServletException {

  PrintWriter out=resp.getWriter();

  String password=req.getParameter("password");
  if(password.equals("admin")){
  chain.doFilter(req, resp);//sends request to next resource
  }
   else{
  out.print("username or password error!");
  RequestDispatcher rd=req.getRequestDispatcher("index.html");
  rd.include(req, resp);
  }
```

```
    }
    public void destroy() {}

}
```

AdminServlet.java

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.*;

public class AdminServlet extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    out.print("welcome ADMIN");
    out.close();
  }
}
```

web.xml

```
<web-app>
<servlet>
  <servlet-name>AdminServlet</servlet-name>
  <servlet-class>AdminServlet</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>AdminServlet</servlet-name>
  <url-pattern>/servlet1</url-pattern>
</servlet-mapping>
```

5. डेटाबेस से रिकॉर्ड का चयन करने के लिए सर्वलेट एप्लीकेशन लिखें।
6. लॉगिन पेज के लिए आवेदन के लिए सर्वलेट एप्लीकेशन लिखें।
7. डेटाबेस में रिकॉर्ड इन्सर्ट के लिए सर्वलेट एप्लीकेशन लिखें।
8. वेब पेज पर आने वाली यात्राओं की गणना के लिए सर्वलेट एप्लीकेशन लिखें।
9. डेटाबेस में शिक्षक रिकॉर्ड सम्मिलित करने के लिए सर्वलेट एप्लीकेशन लिखें।
10. मतदाता पात्रता की जांच के लिए सर्वलेट एप्लीकेशन लिखें।
11. मोबाइल डिटेल्स को नोट करने और प्रदर्शित करने के लिए सर्वलेट एप्लीकेशन लिखें।
12. कुकी को चयनित मूल्य में जोड़ने के लिए सर्वलेट एप्लीकेशन लिखें।

❑

4

हैंडलीग सेशन
HANDLING SESSIONS

LEARNING OBJECTIVES :

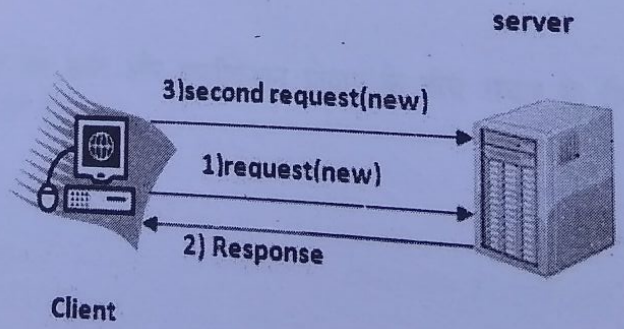
After reading this chapter, you would be able to understand :

- परिचय (Introduction)
- सेशन ट्रैकिंग मैकेनिज्म (Session Tracking Mechanism)

4.1 परिचय (Introduction)

सेशन का अर्थ है समय का एक विशेष अंतराल। सेशन ट्रैकिंग यूजर के स्टेट (डेटा) को बनाए रखने का एक मेथड है। इसे सर्वलेट में सेशन प्रबंधन के रूप में भी जाना जाता है।

Http एक स्टेटलेस प्रोटोकॉल है, इसलिए हमें सेशन ट्रैकिंग तकनीकों का उपयोग करके स्टेट बनाए रखने की आवश्यकता है। हर बार यूजर सर्वर से रिक्वेस्ट करता है, सर्वर रिक्वेस्ट को नए रिक्वेस्ट के रूप में मानता है। इसलिए हमें विशेष यूजर को पहचानने के लिए यूजर की स्थिति बनाए रखने की आवश्यकता है।



सेशन ट्रैकिंग का उपयोग क्यों करें?

यूजर को पहचानने के लिए : इसका उपयोग विशेष यूजर को पहचानने के लिए किया जाता है।

4.2 सेशन ट्रैकिंग मैकेनिज्म (Session Tracking Mechanism)

सेशन ट्रैकिंग में चार तकनीकों का उपयोग किया जाता है—

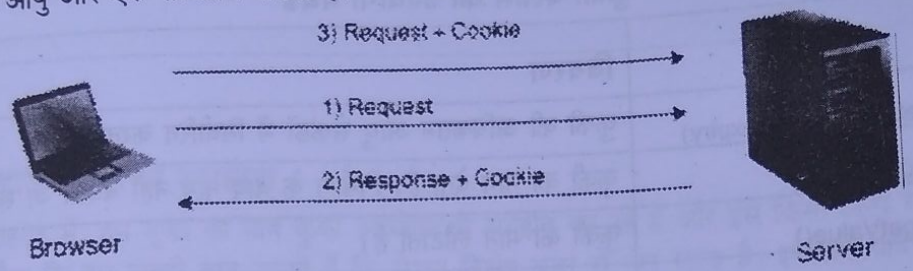
1. Cookies
2. Hidden Form Field

3. URL Rewriting
4. Http Session

सर्वलेट में कुकीज़

कुकी एक छोटी जानकारी होती है जो कई क्लाइंट अनुरोधों के बीच बनी रहती है।

कुकी एक नाम, एक एकल मान और वैकल्पिक विशेषतों जैसे कि एक टिप्पणी, पथ और डोमेन क्वालिफायर, एक अधिकतम आयु और एक संस्करण संख्या है।



कुकी के टाइप

सर्वलेट्स में 2 तरह की कुकीज होती हैं।

1. नॉन-पर्सिस्टेन्ट कुकी
2. पर्सिस्टेन्ट कुकी

नॉन-पर्सिस्टेन्ट कुकी

यह केवल एकल सेशन के लिए मान्य है। यूजर द्वारा ब्राउज़र बंद करने पर इसे हर बार हटा दिया जाता है।

पर्सिस्टेन्ट कुकी

यह कई सत्रों के लिए मान्य है। यूजर द्वारा ब्राउज़र बंद करने पर इसे हर बार नहीं हटाया जाता है। यूजर लॉगआउट या साइनआउट होने पर ही इसे हटाया जाता है।

कुकीज़ का लाभ

1. स्टेट को बनाए रखने की सबसे सरल तकनीक।
2. क्लाइंट पक्ष में कुकीज़ बनाए रखी जाती हैं।

कुकीज़ का नुकसान

1. यह काम नहीं करेगा यदि कुकी ब्राउज़र से डिसएबल है।
2. कुकी ऑब्जेक्ट में केवल पाठ्य सूचना निर्धारित की जा सकती है।

कुकी क्लास

javax.servlet.http.Cookie क्लास कुकीज़ का उपयोग करने की कार्यक्षमता प्रदान करता है। यह कुकीज़ के लिए बहुत से उपयोगी तरीके प्रदान करता है।

कुकी क्लास का कंस्ट्रक्टर

कंस्ट्रक्टर	विवरण
Cookie()	एक कुकी का निर्माण करता है।
Cookie(String name, String value)	एक निर्दिष्ट नाम और मूल्य के साथ एक कुकी का निर्माण करता है।

कुकी क्लास की उपयोगी मेथड

मेथड	विवरण
public void setMaxAge(int expiry)	कुकी की अधिकतम आयु सेकंडों में निर्धारित करता है।
public String getName()	कुकी का नाम देता है। निर्माण के बाद नाम नहीं बदला जा सकता है।
public String getValue()	कुकी का मान लौटाता है।
public void setName(String name)	कुकी का नाम बदल देता है।
public void setValue(String value)	कुकी का मूल्य बदलता है।

कुकी क्लास के कुछ सामान्य रूप से उपयोग किए गए तरीके हैं।

कुकीज़ का उपयोग करने के लिए आवश्यक अन्य मेथड

कुकी को जोड़ने या कुकी से मूल्य प्राप्त करने के लिए, हमें अन्य इंटरफेस द्वारा प्रदान किए गए कुछ तरीकों की आवश्यकता है। वो हैं—

1. **public void addCookie(Cookie ck) :** HttpServletResponse इंटरफ़ेस का मेथड का उपयोग प्रतिक्रिया ऑब्जेक्ट में कुकी जोड़ने के लिए किया जाता है।
2. **public Cookie[] getCookies() :** HttpServletRequest इंटरफ़ेस का मेथड का उपयोग ब्राउज़र से सभी कुकीज़ को वापस करने के लिए किया जाता है।

कुकी कैसे बनाएं?

आइए देखें कुकी बनाने के लिए सरल कोड।

```
Cookie ck=new Cookie('user','asian publishers');//creating cookie object
response.addCookie(ck);//adding cookie in the response
```

कुकी को कैसे हटों?

आइए कुकी को हटाने के लिए सरल कोड देखें। यह मुख्य रूप से यूजर को लॉगआउट या साइनआउट करने के लिए उपयोग किया जाता है।

```
Cookie ck=new Cookie('user','');//deleting value of cookie
ck.setMaxAge(0);//changing the maximum age to 0 seconds
response.addCookie(ck);//adding cookie in the response
```

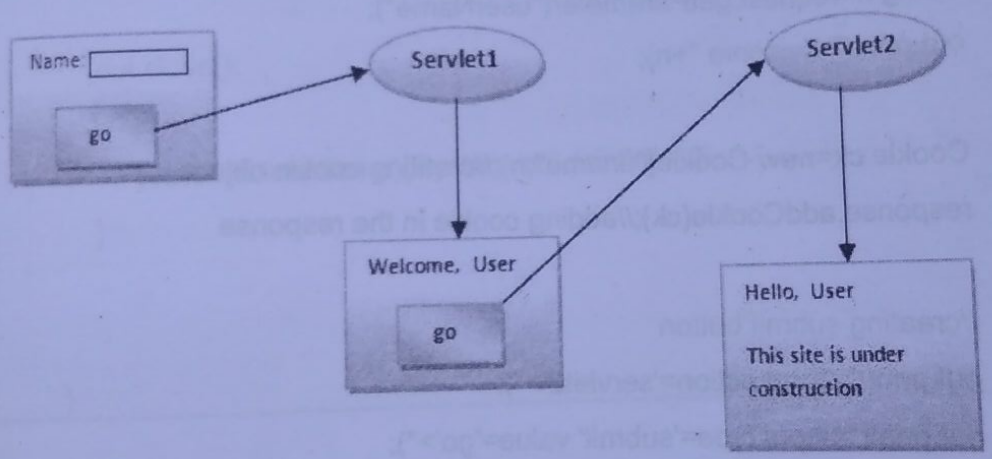
कुकीज़ कैसे प्राप्त करें?

चलो सभी कुकीज़ प्राप्त करने के लिए सरल कोड देखें।

```
Cookie ck[]=request.getCookies();
for(int i=0;i<ck.length;i++){
out.print("<br>"+ck[i].getName()+" "+ck[i].getValue());//printing name and value of cookie
}
```

सर्वलेट कुकीज़ का सरल उदाहरण

इस उदाहरण में, हम यूजर का नाम कुकी ऑब्जेक्ट में संग्रहीत कर रहे हैं और इसे किसी अन्य सर्वलेट में एक्सेस कर रहे हैं। जैसा कि हम अच्छी तरह जानते हैं कि सेशन विशेष यूजर से मेल खाता है। इसलिए यदि आप इसे विभिन्न मूल्यों वाले बहुत सारे ब्राउज़रों से एक्सेस करते हैं, तो आपको अलग-अलग मूल्य मिलेंगे।



index.html

```
<form action="servlet1" method="post">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
```

FirstServlet.java

```
import java.io.*;
import javax.servlet.*;
```

```
import javax.servlet.http.*;

public class FirstServlet extends HttpServlet {

 public void doPost(HttpServletRequest request, HttpServletResponse response){

  try{

  response.setContentType("text/html");
  PrintWriter out = response.getWriter();

  String n=request.getParameter("userName");
  out.print("Welcome "+n);

  Cookie ck=new Cookie("uname",n);//creating cookie object
  response.addCookie(ck);//adding cookie in the response

  //creating submit'button
  out.print("<form action='servlet2'>");
  out.print("<input type='submit' value='go'>");
  out.print("</form>");

  out.close();

   }catch(Exception e){System.out.println(e);}
 }
}
```

SecondServlet.java

```
import java.io.*;
import javax.servlet.*;
```

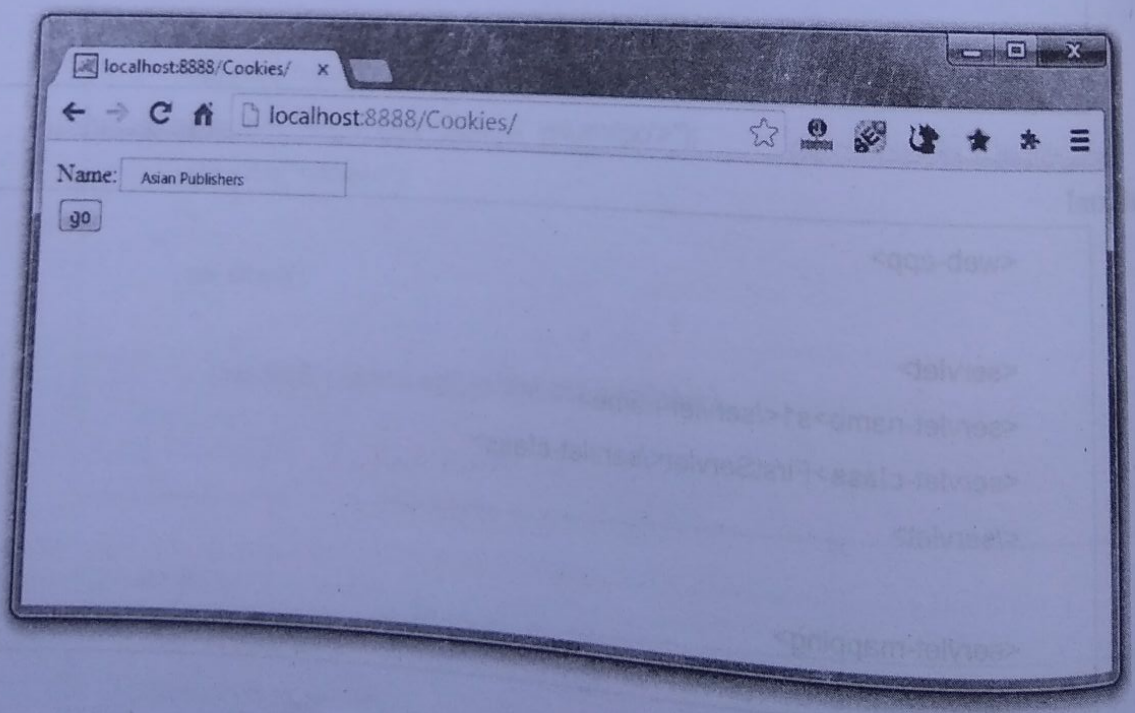
```
import javax.servlet.http.*;

public class SecondServlet extends HttpServlet {

public void doPost(HttpServletRequest request, HttpServletResponse response){
  try{

  response.setContentType("text/html");
  PrintWriter out = response.getWriter();

  Cookie ck[]=request.getCookies();
  out.print("Hello "+ck[0].getValue());

  out.close();

     }catch(Exception e){System.out.println(e);}
  }


}
```

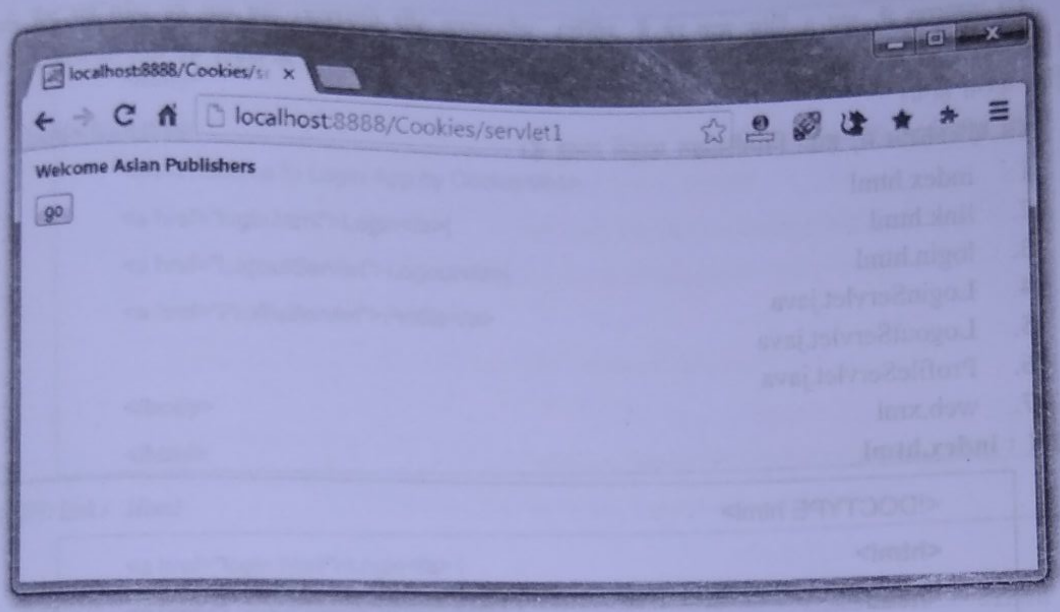
web.xml

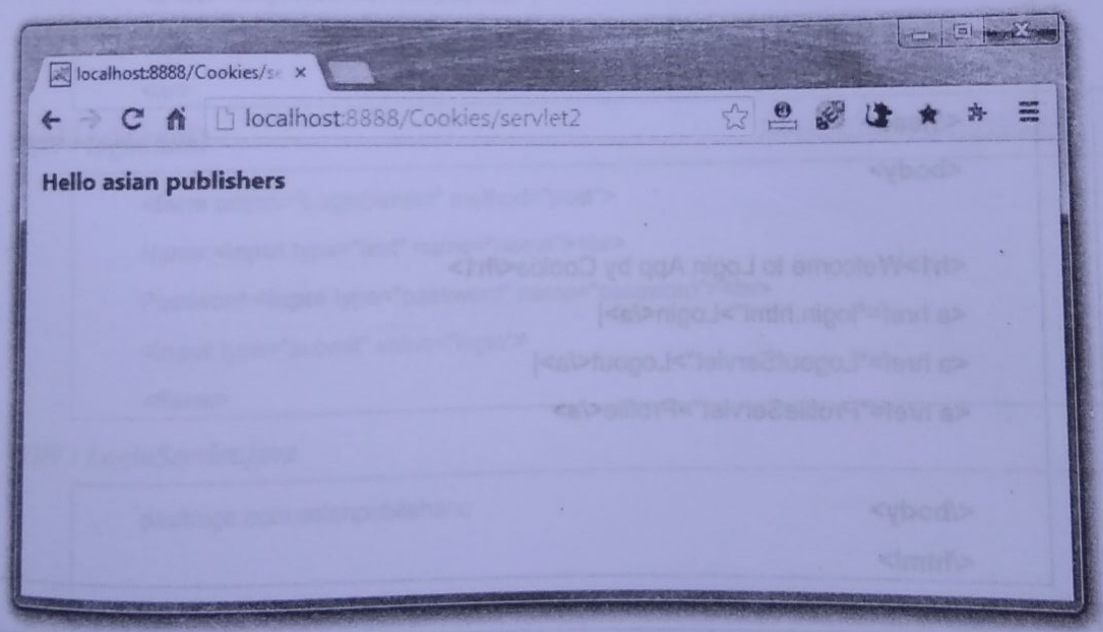
```
<web-app>

<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s1</servlet-name>
```

```
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

<servlet>
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
</servlet-mapping>

</web-app>
```

आउटपुट







सर्वलेट लॉगिन और लॉगआउट उदाहरण कुकीज़ का उपयोग करके

एक कुकी एक टाइप की जानकारी है जो क्लाइंट साइड पर संग्रहीत होती है।

पिछले पृष्ठ में, हमने कुकी के बारे में बहुत कुछ सीखा जैसे कि कुकी कैसे बनों, कुकी को कैसे हटों, कुकी कैसे प्राप्त करें आदि।

यहां, हम सर्वलेट कुकीज़ का उपयोग करके एक लॉगिन और लॉगआउट उदाहरण बनाने जा रहे हैं।

इस उदाहरण में, हम 3 लिंक बना रहे हैं: लॉगिन, लॉगआउट और प्रोफाइल। जब तक वह लॉग इन नहीं होता तब तक यूजर प्रोफ़ाइल पृष्ठ पर नहीं जा सकता। यदि यूजर लॉग आउट है, तो उसे प्रोफ़ाइल पर जाने के लिए फिर से लॉगिन करना होगा।

इस एप्लिकेशन में, हमने निम्नलिखित फाइलें बनाई हैं।

1. index.html
2. link.html
3. login.html
4. LoginServlet.java
5. LogoutServlet.java
6. ProfileServlet.java
7. web.xml

फाइल : index.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Servlet Login Example</title>
</head>
<body>

<h1>Welcome to Login App by Cookie</h1>
<a href="login.html">Login</a>|
<a href="LogoutServlet">Logout</a>|
<a href="ProfileServlet">Profile</a>

</body>
</html>
```

फाइल : link~ Html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Servlet Login Example</title>
```

```
</head>
<body>

<h1>Welcome to Login App by Cookie</h1>
<a href="login.html">Login</a>|
<a href="LogoutServlet">Logout</a>|
<a href="ProfileServlet">Profile</a>

</body>
</html>
```

फाइल: *link/ Html*

```
<a href="login.html">Login</a> |
<a href="LogoutServlet">Logout</a> |
<a href="ProfileServlet">Profile</a>
<hr>
```

फाइल : ***login.html***

```
<form action="LoginServlet" method="post">
Name:<input type="text" name="name"><br>
Password:<input type="password" name="password"><br>
<input type="submit" value="login">
</form>
```

फाइल : ***LoginServlet.java***

```
package com.asianpublishers;


import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
```

```
import javax.servlet.http.HttpServletResponse;
public class LoginServlet extends HttpServlet {
  protected void doPost(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out=response.getWriter();

    request.getRequestDispatcher("link.html").include(request, response);

    String name=request.getParameter("name");
    String password=request.getParameter("password");

    if(password.equals("admin123")){
      out.print("You are successfully logged in!");
      out.print("<br>Welcome, "+name);

      Cookie ck=new Cookie("name",name);
      response.addCookie(ck);
    }else{
      out.print("sorry, username or password error!");
      request.getRequestDispatcher("login.html").include(request, response);
    }

    out.close();
  }

}
```

फाइल: *LogoutServlet.java*

```
package com.asianpublishers;

import java.io.IOException;
```

```
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class LogoutServlet extends HttpServlet {
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out=response.getWriter();


    request.getRequestDispatcher("link.html").include(request, response);


    Cookie ck=new Cookie("name","");
    ck.setMaxAge(0);
    response.addCookie(ck);


    out.print("you are successfully logged out!");
  }
}
```

फाइल: *ProfileServlet.java*

```
package com.asianpublishers;


import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
```

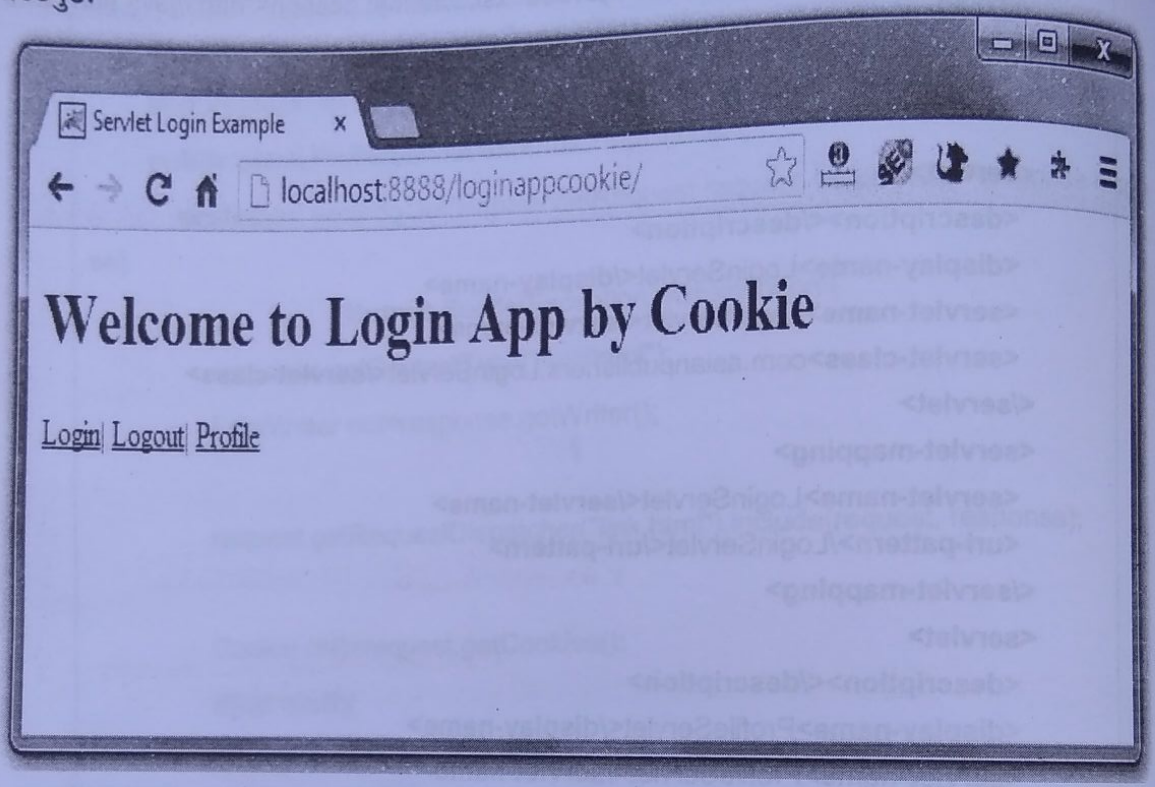
```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class ProfileServlet extends HttpServlet {
  protected void doGet(HttpServletRequest request, HttpServletResponse respon
se)
                throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out=response.getWriter();

    request.getRequestDispatcher("link.html").include(request, response);

    Cookie ck[]=request.getCookies();
    if(ck!=null){
     String name=ck[0].getValue();
    if(!name.equals("")||name!=null){
      out.print("<b>Welcome to Profile</b>");
      out.print("<br>Welcome, "+name);
    }
    }else{
      out.print("Please login first");
      request.getRequestDispatcher("login.html").include(request, response);
    }
    out.close();
  }
}
```

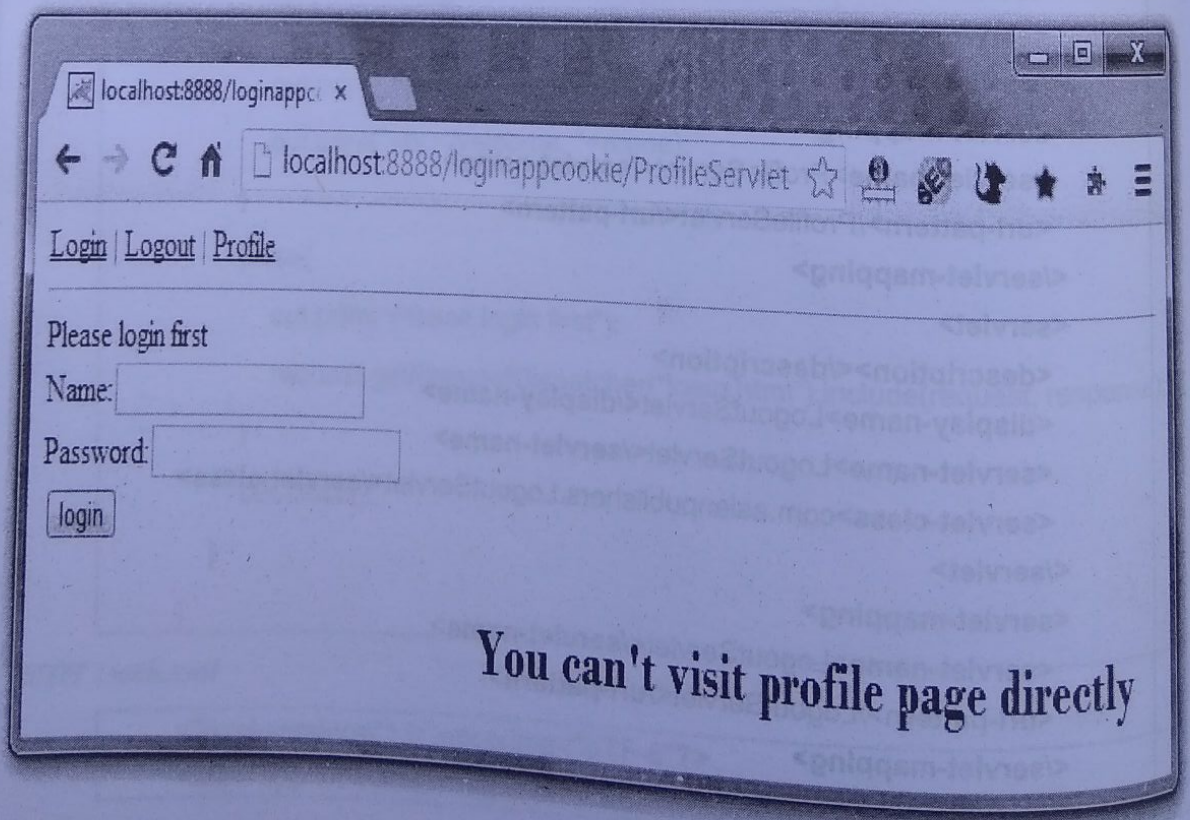
फाइल : *web.xml*

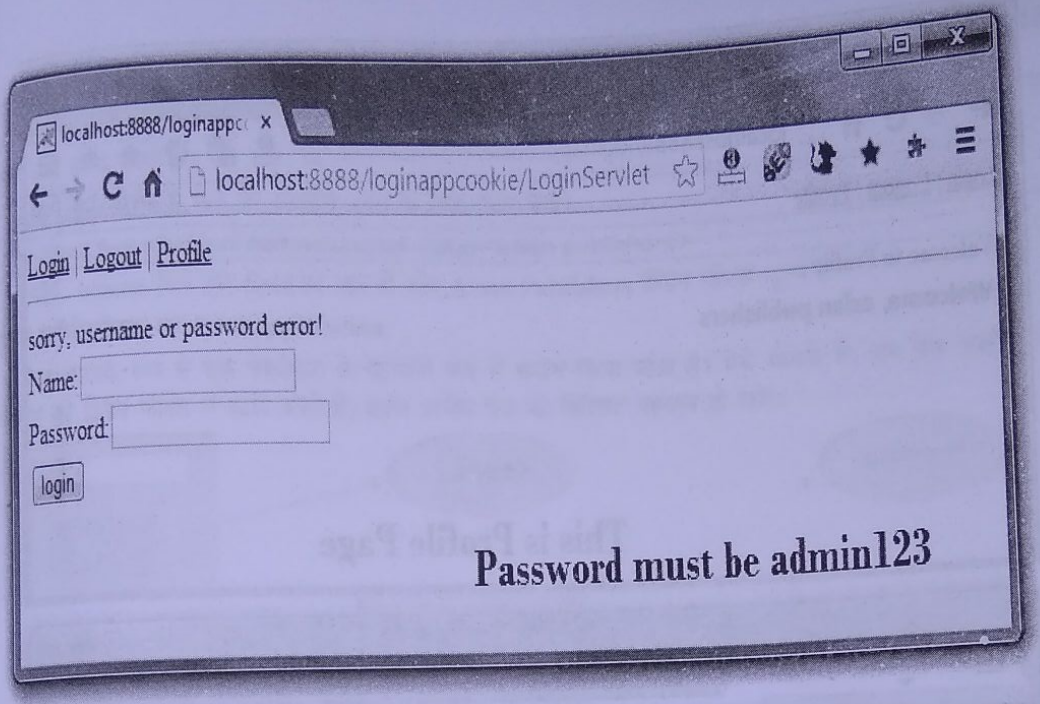
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

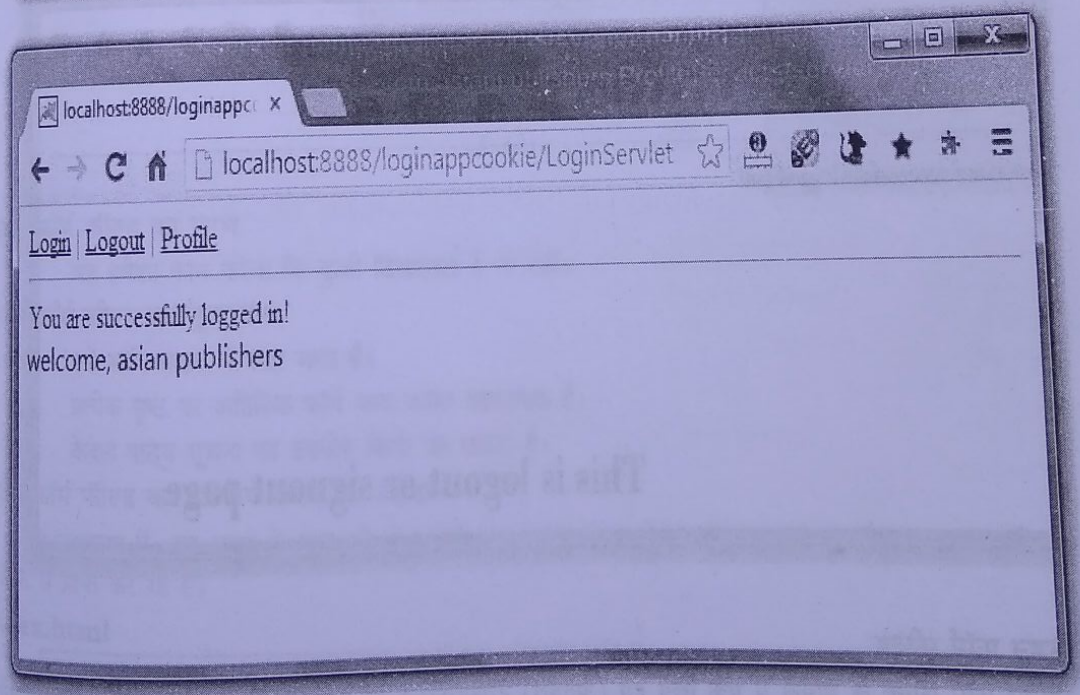
```
    xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/
xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-
app_2_5.xsd" id="WebApp_ID" version="2.5">

  <servlet>
   <description></description>
   <display-name>LoginServlet</display-name>
   <servlet-name>LoginServlet</servlet-name>
   <servlet-class>com.asianpublishers.LoginServlet</servlet-class>
  </servlet>
  <servlet-mapping>
   <servlet-name>LoginServlet</servlet-name>
   <url-pattern>/LoginServlet</url-pattern>
  </servlet-mapping>
  <servlet>
   <description></description>
   <display-name>ProfileServlet</display-name>
   <servlet-name>ProfileServlet</servlet-name>
   <servlet-class>com.asianpublishers.ProfileServlet</servlet-class>
  </servlet>
  <servlet-mapping>
   <servlet-name>ProfileServlet</servlet-name>
   <url-pattern>/ProfileServlet</url-pattern>
  </servlet-mapping>
  <servlet>
   <description></description>
   <display-name>LogoutServlet</display-name>
   <servlet-name>LogoutServlet</servlet-name>
   <servlet-class>com.asianpublishers.LogoutServlet</servlet-class>
  </servlet>
  <servlet-mapping>
   <servlet-name>LogoutServlet</servlet-name>
   <url-pattern>/LogoutServlet</url-pattern>
  </servlet-mapping>
  </web-app>
```

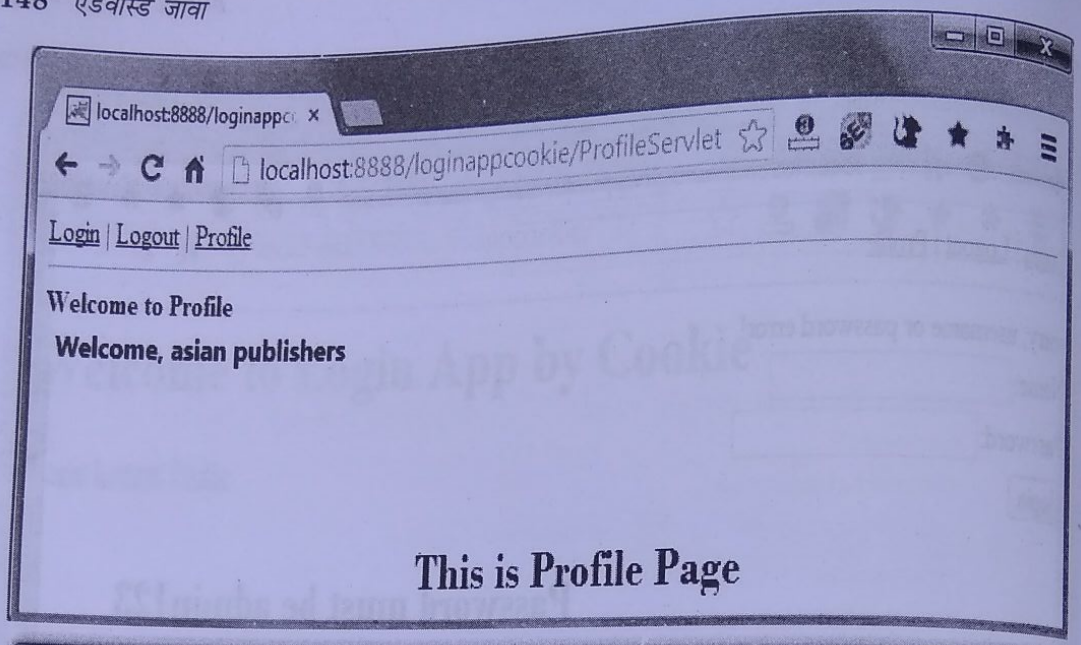
आउटपुट:

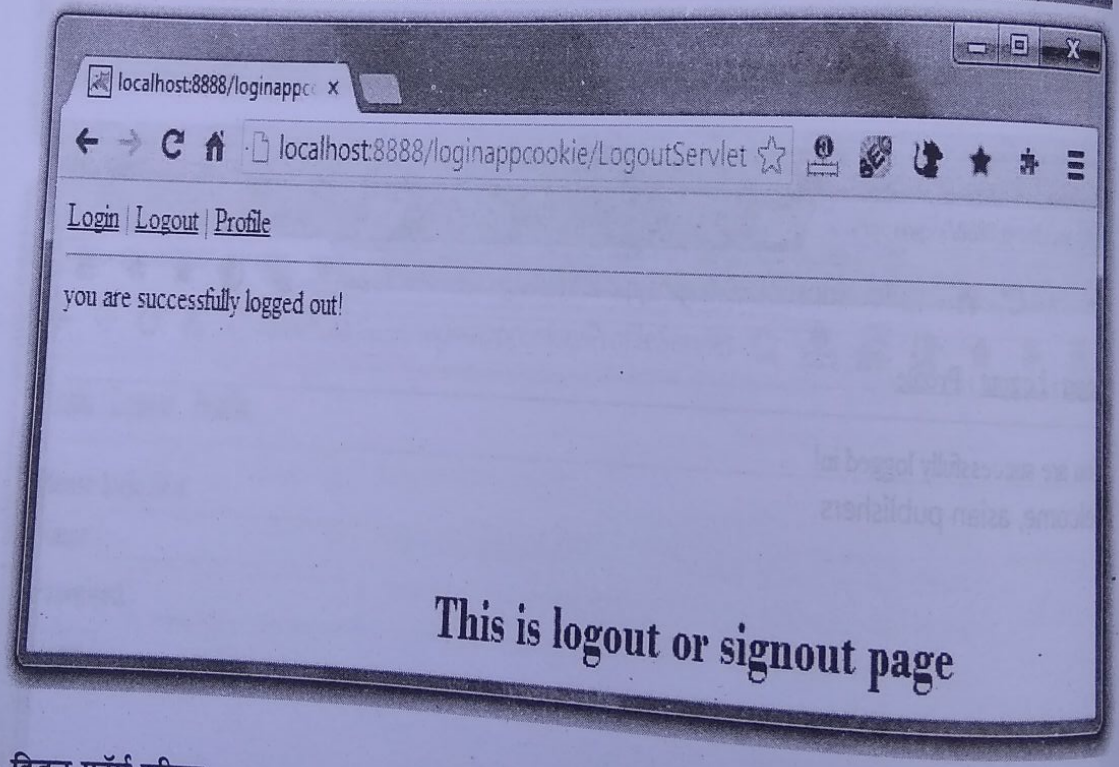

Servlet Login Example
localhost:8888/loginappcookie/
Welcome to Login App by Cookie
Login| Logout| Profile


localhost:8888/loginappc
localhost:8888/loginappcookie/ProfileServlet
Login | Logout | Profile
Please login first
Name:
Password:
login
You can't visit profile page directly


localhost:8888/loginappc
localhost:8888/loginappcookie/LoginServlet
Login | Logout | Profile
sorry, username or password error!
Name:
Password:
login
Password must be admin123


localhost:8888/loginappc
localhost:8888/loginappcookie/LoginServlet
Login | Logout | Profile
You are successfully logged in!
welcome, asian publishers





(2) हिडन फॉर्म फील्ड

हिडन फॉर्म फील्ड के मामले में एक छिपे हुए (अदृश्य) टेक्स्टफील्ड का उपयोग यूजर की स्थिति को बनाए रखने के लिए किया जाता है। ऐसे मामले में, हम जानकारी को छिपे हुए Field में संग्रहीत करते हैं और इसे किसी अन्य

सर्वलेट से प्राप्त करते हैं। यह दृष्टिकोण बेहतर है अगर हमें सभी पृष्ठों में फ़ॉर्म जमा करना है और हम ब्राउज़र पर निर्भर नहीं होना चाहते हैं।

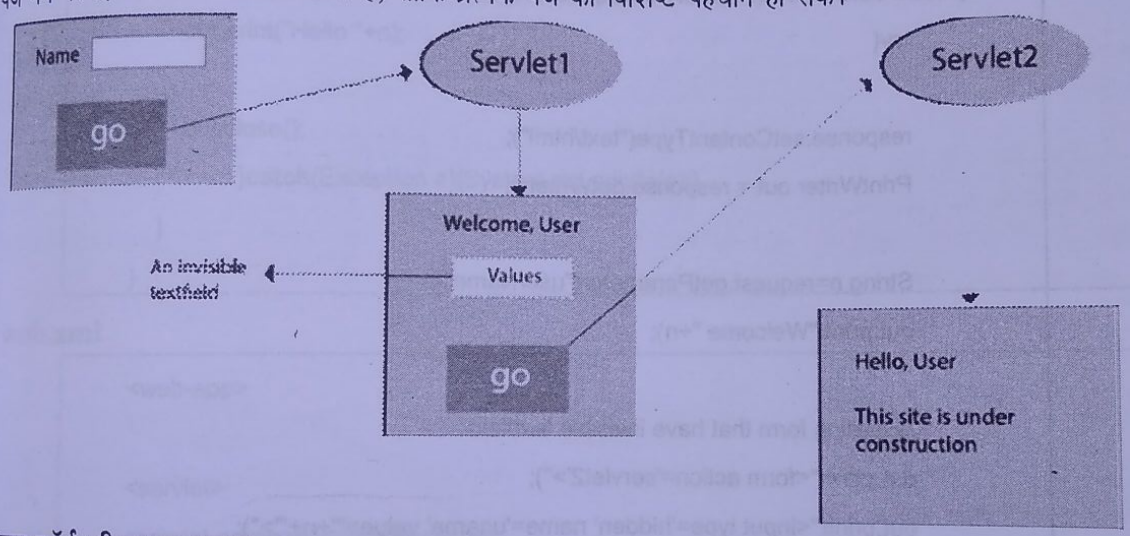
छिपे हुए फ़ील्ड में मान को संग्रहीत करने के लिए कोड देखें।

```
<input type='hidden' name='uname' value='asian publishers'>
```

यहाँ, uname छिपे हुए field का नाम है और Asian Publishers हिडन फील्ड मूल्य है।

हिडन फॉर्म फील्ड का Real application

यह व्यापक रूप से एक वेबसाइट के टिप्पणी रूप में प्रयोग किया जाता है। ऐसे मामले में, हम पेज आईडी या पेज नेम को हिडन फील्ड में स्टोर करते हैं, ताकि प्रत्येक पेज की विशिष्ट पहचान हो सके।



हिडन फॉर्म फील्ड का लाभ

1. यह हमेशा काम करेगा कि कुकी डिसएबल है या नहीं।

हिडन फॉर्म फील्ड का नुकसानः

1. इसे सर्वर साइड में रखा जाता है।
2. प्रत्येक पृष्ठ पर अतिरिक्त फॉर्म जमा करना आवश्यक है।
3. केवल पाठ्य सूचना का उपयोग किया जा सकता है।

हिडन फॉर्म फील्ड का उपयोग करने का उदाहरण

इस उदाहरण में, हम यूजर के नाम को एक छिपे हुए टेक्स्टफील्ड में संग्रहीत कर रहे हैं और उस मूल्य को दूसरे सर्वलेट से प्राप्त कर रहे हैं।

index.html

```
<form action="servlet1">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
```

FirstServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class FirstServlet extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response){
    try{

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    String n=request.getParameter("userName");
    out.print("Welcome "+n);

    //creating form that have invisible textfield
    out.print("<form action='servlet2'>");
    out.print("<input type='hidden' name='uname' value='"+n+"'>");
    out.print("<input type='submit' value='go'>");
    out.print("</form>");
    out.close();

        }catch(Exception e){System.out.println(e);}
    }

}
```

SecondServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SecondServlet extends HttpServlet {
```

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
    try{
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    //Getting the value from the hidden field
    String n=request.getParameter("uname");
    out.print("Hello "+n);

    out.close();
        }catch(Exception e){System.out.println(e);}
  }
}
```

web.xml

```
<web-app>

<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

<servlet>
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>
```

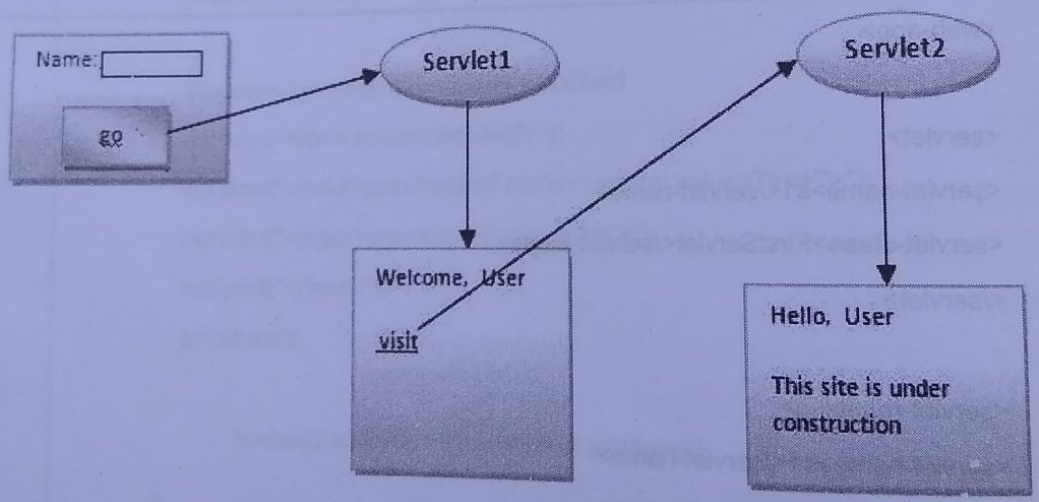
```
<servlet-mapping>
<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
</servlet-mapping>

</web-app>
```

(3) URL रीराइटिंग

URL Rewriting में, हम अगले सर्वलेट या अगले रिसोर्सेज के URL में एक टोकन या पहचानकर्ता जोड़ते हैं। हम निम्न प्रारूप का उपयोग करके पैरामीटर (नाम/मूल्य) भेज सकते हैं—

```
url?name1=value1&name2=value2&??
```

एक नाम और एक मान को बराबर = चिह्न का उपयोग करके अलग किया जाता है, एक पैरामीटर नाम/मूल्य जोड़ी को एम्परसेंड (&) का उपयोग करके दूसरे पैरामीटर से अलग किया जाता है। जब यूजर हाइपरलिंक पर क्लिक करता है, तो पैरामीटर नाम/मान सर्वर पर पास होता है। एक सर्वलेट से, हम पैरामीटर मान प्राप्त करने के लिए getParameter () मेथड का उपयोग कर सकते हैं।



यूआरएल रिवाइटरिंग का लाभ

1. यह हमेशा काम करेगा कि कुकी डिसएबल है या नहीं (ब्राउज़र independent)।
2. प्रत्येक पेज पर अतिरिक्त फॉर्म जमा करने की आवश्यकता नहीं है।

यूआरएल रिवाइटरिंग का नुकसान

1. यह केवल लिंक्स के साथ काम करेगा।
2. यह केवल पाठ्य सूचना भेज सकता है।

URL Rewriting का उपयोग करने का उदाहरण

इस उदाहरण में, हम लिंक का उपयोग करके यूजर की स्थिति का अनुमान लगा रहे हैं। इस प्रयोजन के लिए, हम क्वेरी स्ट्रिंग में यूजर का नाम जोड़ रहे हैं और क्वेरी स्ट्रिंग से दूसरे पृष्ठ में मान प्राप्त कर रहे हैं।

index.html

```
<form action="servlet1">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
```

FirstServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class FirstServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response){
    try{

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    String n=request.getParameter("userName");
    out.print("Welcome "+n);

    //appending the username in the query string
    out.print("<a href='servlet2?uname="+n+"'>visit</a>");
```

```
            out.close();

                }catch(Exception e){System.out.println(e);}

        }


    }
```

SecondServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SecondServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response)
    try{

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    //getting value from the query string
    String n=request.getParameter("uname");
    out.print("Hello "+n);

    out.close();

        }catch(Exception e){System.out.println(e);}
  }


}
```

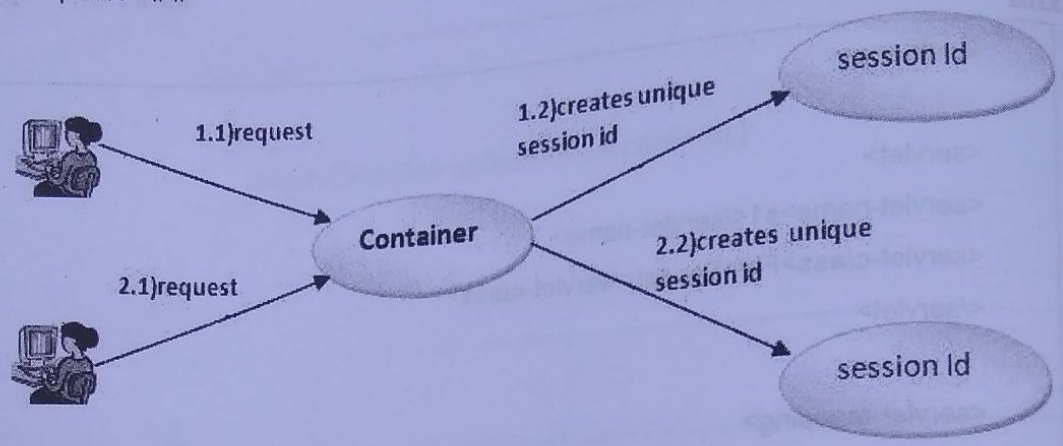
web.xml

```
<web-app>

<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

<servlet>
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
</servlet-mapping>

</web-app>
```

(4) HttpSession इंटरफ़ेस

ऐसे मामले में, कंटेनर प्रत्येक यूजर के लिए एक सेशन आईडी बनाता है। कंटेनर विशेष यूजर की पहचान करने के लिए इस आईडी का उपयोग करता है। HttpSession की एक ऑब्जेक्ट का उपयोग दो कार्यों को करने के लिए किया जा सकता है:

1. ऑब्जेक्टओं को बाँधना (bind objects)
2. किसी सेशन के बारे में जानकारी देखना और उसमें हेरफेर करना, जैसे सेशन पहचानकर्ता, क्रिएशन टाइम और अंतिम अक्सेस्सेड (accessed) टाइम।



HttpSession ऑब्जेक्ट कैसे प्राप्त करें?

HttpServletRequest इंटरफ़ेस HttpSession की ऑब्जेक्ट प्राप्त करने के लिए दो तरीके प्रदान करता है—

1. **public HttpSession getSession() :** इस रिक्वेस्ट से संबंधित वर्तमान सेशन लौटाता है, या यदि रिक्वेस्ट में सेशन नहीं है, तो एक बनाता है।

2. **public HttpSession getSession(boolean create) :** इस रिक्वेस्ट से जुड़े वर्तमान HttpSession को लौटाता है, या यदि कोई वर्तमान सेशन नहीं है और True है, तो नया सेशन लौटाता है।

HttpSession इंटरफ़ेस के सामान्य रूप से उपयोग किए जाने वाले तरीके

1. **public String getId() :** विशिष्ट identifier मान युक्त स्ट्रिंग लौटाता है।

2. **public long getCreationTime() :** यह सेशन जब बनाया गया था, उस समय को लौटाता है।

3. **public long getLastAccessedTime() :** लास्ट टाइम जब क्लाइंट ने इस सेशन के साथ एसोसिएट रिक्वेस्ट को भेजा था, उस समय को return करता है।

4. **public void invalidate() :** इस सेशन को Invalidate कर देता है और फिर इससे जुड़ी किसी भी ऑब्जेक्ट को हटा देता है।

HttpSession का उपयोग करने का उदाहरण

इस उदाहरण में, हम एक सर्वलेट में सेशन के दायरे में एट्रिब्यूट सेट कर रहे हैं और सेशन मान से किसी अन्य सर्वलेट में उस मान को प्राप्त कर रहे हैं। सेशन स्कोप में एट्रिब्यूट को सेट करने के लिए, हमने HttpSession इंटरफ़ेस के setAttribute() मेथड का उपयोग किया है और एट्रिब्यूट प्राप्त करने के लिए, हमने getAttribute मेथड का उपयोग किया है।

index.html

```
<form action="servlet1">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
```

FirstServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class FirstServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response){
    try{

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    String n=request.getParameter("userName");
    out.print("Welcome "+n);

    HttpSession session=request.getSession();
    session.setAttribute("uname",n);

    out.print("<a href='servlet2'>visit</a>");

    out.close();

        }catch(Exception e){System.out.println(e);}
  }

}
```

SecondServlet.java

```
import java.io.*;
import javax.servlet.*;
```

```
import javax.servlet.http.*;

public class SecondServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response)
    try{

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    HttpSession session=request.getSession(false);
    String n=(String)session.getAttribute("uname");
    out.print("Hello "+n);

    out.close();

        }catch(Exception e){System.out.println(e);}
  }

}
```

web.xml

```
<web-app>

<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s1</servlet-name>
```

```
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

<servlet>
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
</servlet-mapping>

</web-app>
```

सर्वलेट HttpSession लॉगिन और लॉगआउट उदाहरण

हम HttpSession instance पर ऑब्जेक्ट्स को बाइंड कर सकते हैं और setAttribute और getAttribute मेथड का उपयोग करके ऑब्जेक्ट प्राप्त कर सकते हैं।

पिछले पृष्ठ में, हमने सीखा है कि HttpSession क्या है, सेशन् ऑब्जेक्ट आदि से डेटा कैसे स्टोर करें और प्राप्त करें।

यहां, हम डेटाबेस कोड का उपयोग किए बिना एक real world लॉगिन और लॉगआउट एप्लिकेशन बनाने जा रहे हैं। हम मान रहे हैं कि पासवर्ड admin123 है।

इस उदाहरण में, हम 3 लिंक बना रहे हैं: लॉगिन, लॉगआउट और प्रोफाइल। जब तक वह लॉग इन नहीं होता तब तक यूजर प्रोफ़ाइल पृष्ठ पर नहीं जा सकता। यदि यूजर लॉग आउट करता है, तो उसे प्रोफ़ाइल पर जाने के लिए फिर से लॉगिन करना होगा।

इस एप्लिकेशन में, हमने निम्नलिखित फाइलें बनाई हैं।

1. index.html
2. link.html
3. login.html
4. LoginServlet.java
5. LogoutServlet.java
6. ProfileServlet.java
7. web.xml

फाइल : *index.html*

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Servlet Login Example</title>
</head>
<body>

<h1>Login App using HttpSession</h1>
<a href="login.html">Login</a>|
<a href="LogoutServlet">Logout</a>|
<a href="ProfileServlet">Profile</a>

</body>
</html>
```

फाइल: *login.html*

```
<form action="LoginServlet" method="post">
Name:<input type="text" name="name"><br>
Password:<input type="password" name="password"><br>
<input type="submit" value="login">
</form>
```

फाइल: *LoginServlet.java*

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
```

```
public class LoginServlet extends HttpServlet {
  protected void doPost(HttpServletRequest request, HttpServletResponse response)
          throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out=response.getWriter();
    request.getRequestDispatcher("link.html").include(request, response);

    String name=request.getParameter("name");
    String password=request.getParameter("password");

    if(password.equals("admin123")){
    out.print("Welcome, "+name);
    HttpSession session=request.getSession();
    session.setAttribute("name",name);
    }
    else{
      out.print("Sorry, username or password error!");
      request.getRequestDispatcher("login.html").include(request, response);
    }
    out.close();
  }
}
```

फाइल: *LogoutServlet.java*

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
public class LogoutServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
                  throws ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter out=response.getWriter();

      request.getRequestDispatcher("link.html").include(request, response);

      HttpSession session=request.getSession();
      session.invalidate();

      out.print("You are successfully logged out!");
```

```
        out.close();
    }
}
```

फाइल: *ProfileServlet.java*

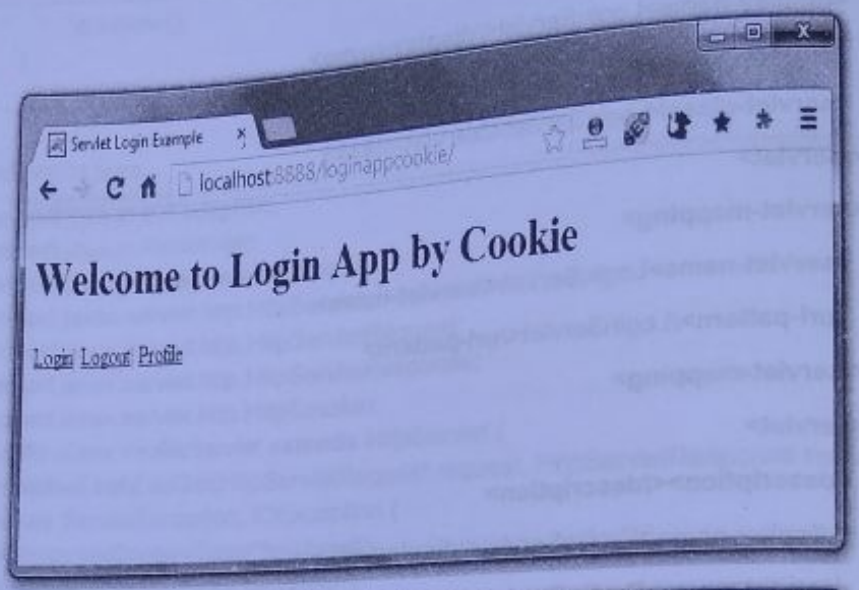
```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
public class ProfileServlet extends HttpServlet {
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
response.setContentType("text/html");
PrintWriter out=response.getWriter();
request.getRequestDispatcher("link.html").include(request, response);
HttpSession session=request.getSession(false);
if(session!=null){
String name=(String)session.getAttribute("name");
out.print("Hello, "+name+" Welcome to Profile");

}
else{
out.print("Please login first");
request.getRequestDispatcher("login.html").include(request, response);
}
out.close();
}
}
```

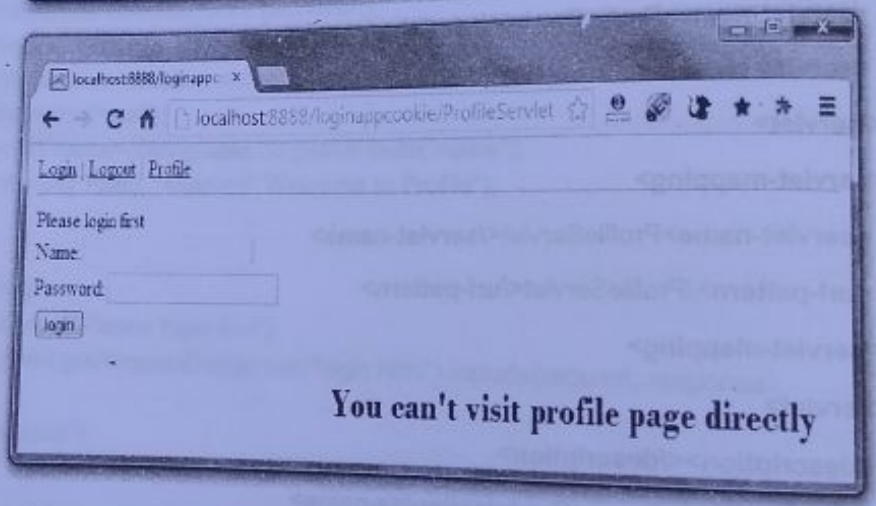
फाइल: **web.xml**

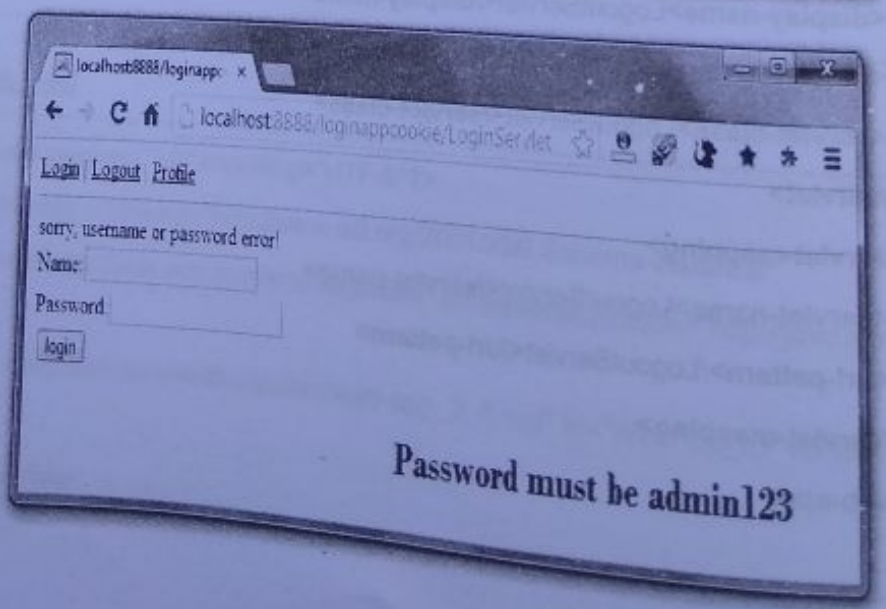
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/ja
vaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">

<servlet>
```

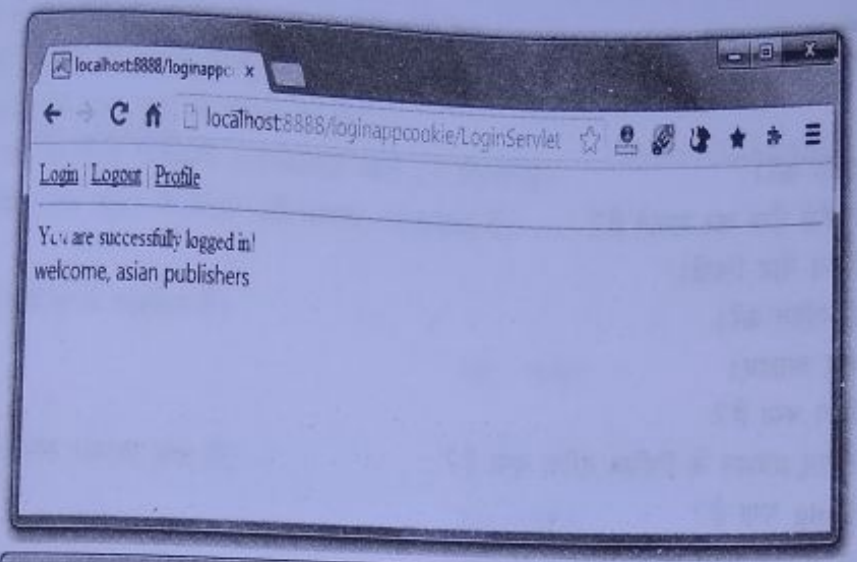
```
    <description></description>
    <display-name>LoginServlet</display-name>
    <servlet-name>LoginServlet</servlet-name>
    <servlet-class>LoginServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>LoginServlet</servlet-name>
    <url-pattern>/LoginServlet</url-pattern>
  </servlet-mapping>
  <servlet>
    <description></description>
    <display-name>ProfileServlet</display-name>
    <servlet-name>ProfileServlet</servlet-name>
    <servlet-class>ProfileServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>ProfileServlet</servlet-name>
    <url-pattern>/ProfileServlet</url-pattern>
  </servlet-mapping>
  <servlet>
    <description></description>
    <display-name>LogoutServlet</display-name>
    <servlet-name>LogoutServlet</servlet-name>
    <servlet-class>LogoutServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>LogoutServlet</servlet-name>
    <url-pattern>/LogoutServlet</url-pattern>
  </servlet-mapping>
</web-app>
```

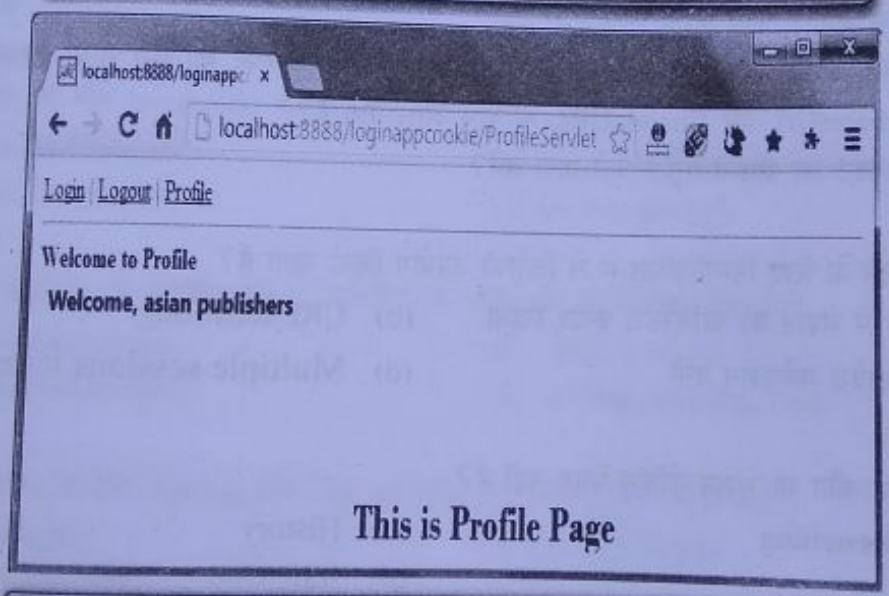
आउटपुटः


Servlet Login Example
localhost:8888/loginappcookie/
Welcome to Login App by Cookie
Login Logout Profile

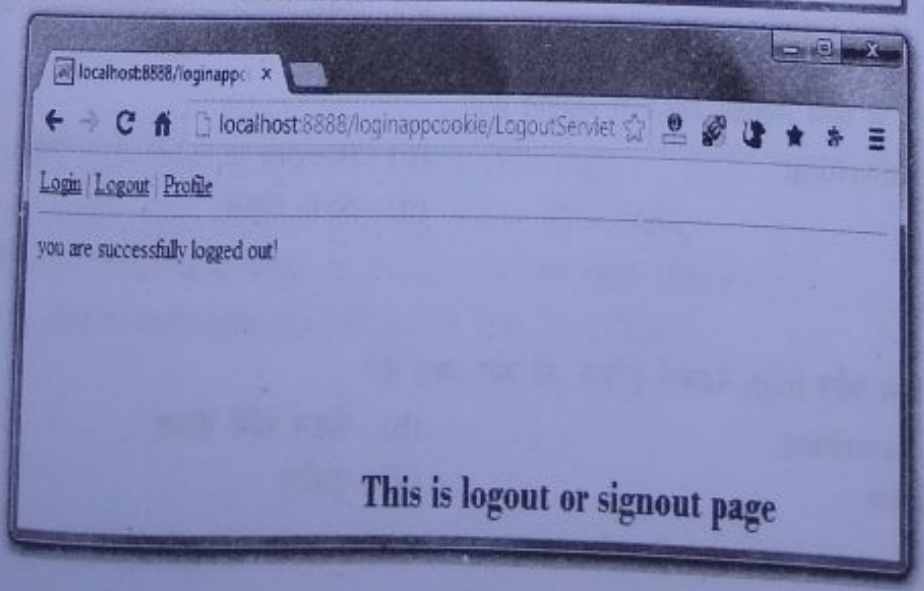

localhost:8888/loginappcookie/ProfileServlet
Login | Logout | Profile
Please login first
Name:
Password:
login
You can't visit profile page directly


localhost:8888/loginappcookie/LoginServlet
Login | Logout | Profile
sorry, username or password error!
Name:
Password:
login
Password must be admin123





This is Profile Page



This is logout or signout page

5

जेएसपी
JSP

LEARNING OBJECTIVES :

After reading this chapter, you would be able to understand :

- परिचय (Introduction)
- JSP एपीआई (JSP API)
- Tomcat सर्वर के साथ Eclipse IDE में JSP बनाना
- सिंटैक्स (Syntax)
- JSP स्क्रिप्टिंग तत्व
- JSP-डायरेक्टिव (JSP-Directive)
- JSP-Implicit ऑब्जेक्ट्स
- JSP -सर्वर response (JSP-Server response)
- JSP integration डेटाबेस के साथ
- JSP सेशन
- JSP में exception हैंडलिंग
- JSTL (JSP स्टैंडर्ड टैग लाइब्रेरी)

5.1 परिचय (Introduction)

JSP तकनीक का उपयोग सर्वलेट तकनीक की तरह वेद एप्लीकेशन बनाने के लिए किया जाता है। इसे सर्वलेट के विस्तार के रूप में माना जा सकता है क्योंकि यह सर्वलेट की तुलना में अधिक कार्यक्षमता प्रदान करता है जैसे कि एक्सप्रेशन भाषा, JSTL, आदि।

एक JSP पेज में HTML टैग और JSP टैग होते हैं। JSP पेज सर्वलेट से बनाए रखना आसान है क्योंकि हम डिजाइनिंग और विकास को अलग कर सकते हैं। यह कुछ अतिरिक्त सुविधों प्रदान करता है जैसे कि एक्सप्रेशन भाषा, कस्टम टैग आदि।

सर्वलेट पर JSP के फायदे

सर्वलेट पर JSP के कई फायदे हैं। वे इस प्रकार हैं—

(1) सर्वलेट का विस्तार

JSP तकनीक सर्वलेट तकनीक को एक्सटेंड है। हम JSP में सर्वलेट की सभी एट्रिब्यूट का उपयोग कर सकते हैं। इसके अलावा, हम JSP में Implicit ऑब्जेक्ट, predefined टैग, एक्सप्रेशन भाषा और कस्टम टैग का उपयोग कर सकते हैं, जो JSP के विकास को आसान बनाता है।

(2) मेन्टेन करना आसान है

JSP को आसानी से प्रबंधित किया जा सकता है क्योंकि हम प्रेजेंटेशन लॉजिक के साथ अपने बिजनेस लॉजिक को आसानी से अलग कर सकते हैं। सर्वलेट टेक्नोलॉजी में, हम अपने बिजनेस लॉजिक को प्रेजेंटेशन लॉजिक के साथ मिलाते हैं।

(3) तेजी से विकास: recompile और redeploy की जरूरत नहीं है

यदि JSP पेज संशोधित किया गया है, तो हमें प्रोजेक्ट को redeploy और recompile की आवश्यकता नहीं है। अगर हमें एप्लीकेशन के रूप और स्वरूप को बदलना है तो सर्वलेट कोड को अपडेट और फिर से तैयार करना होगा।

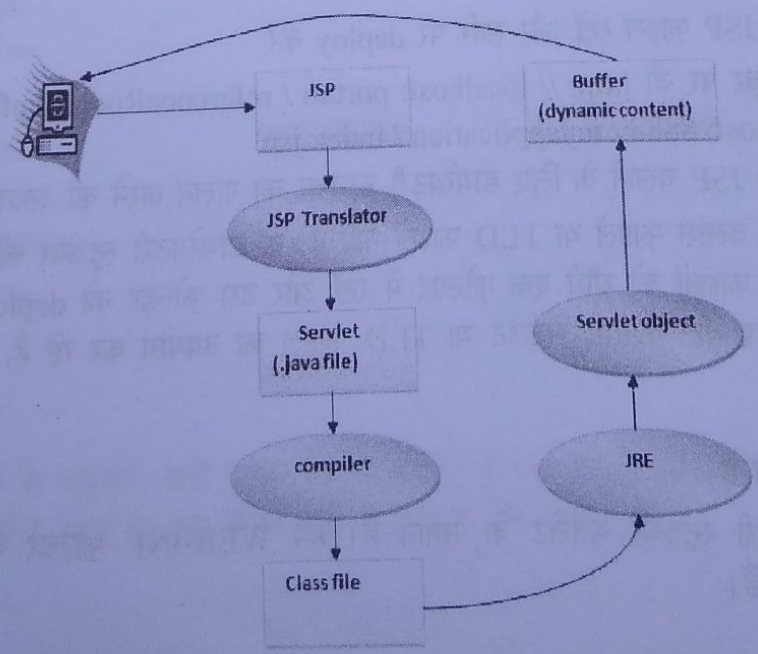
(4) सर्वलेट की तुलना में कम कोडिंग

JSP में, हम कई टैग जैसे कि एक्शन टैग, JSTL, कस्टम टैग आदि का उपयोग कर सकते हैं जो कोड को कम करता है। इसके अलावा, हम ईएल, Implicit ऑब्जेक्ट आदि का उपयोग कर सकते हैं।

एक JSP पेज का जीवनचक्र

JSP पेज इन स्टेप का पालन करते हैं—

- JSP पेज का ट्रांसलेशन
- JSP पेज का कंपाइलेशन
- क्लास लोडिंग (क्लास लोडर क्लास फाइल लोड करता है)
- Instantiation (जेनरेट सर्वलेट का object बनाया जाता है)।
- Intialization (कंटेनर jspInit () मेथड को invoke करता है)।
- रिक्वेस्ट प्रोसेस करता है (कंटेनर _jspService () मेथड को invoke करता है)।
- डिस्ट्रॉय करता है (कंटेनर jspDestroy () मेथड को invoke करता है)।



जैसा कि चित्र में दर्शाया गया है, JSP पेज को ब्राइ ट्रांसलेटर की सहायता से सर्वलेट में ट्रांसलेट किया गया है। JSP ट्रांसलेटर वेब सर्वर का एक हिस्सा है जो JSP पेज को सर्वलेट में ट्रांसलेट करने के लिए जिम्मेदार है। उसके बाद, सर्वलेट पेज को कंपाइलर द्वारा संकलित किया जाता है और क्लास फाइल में परिवर्तित कर दिया जाता है। इसके अलावा, सर्वलेट में होने वाली सभी प्रोसेसेज जैसे initialization,committing response और destroy JSP पर हो जाते हैं।

एक साधारण JSP पेज बनाना

पहला JSP पेज बनाने के लिए, नीचे दिए गए अनुसार कुछ HTML कोड लिखें, और इसे .jsp एक्सटेंशन द्वारा सहेजें। हमने इस फ़ाइल को index.jsp के रूप में सेव किया है। इसे एक फ़ोल्डर में रखें और JSP पेज को चलाने के लिए Apache tomcat में वेब-ऐप्स डायरेक्टरी में फ़ोल्डर को पेस्ट करें।

index.jsp

आइए देखें JSP का सरल उदाहरण जहां हम JSP पेज में जावा कोड डालने के लिए स्क्रिप्टलेट टैग का उपयोग कर रहे हैं। हम बाद में स्क्रिप्ट टैग सीखेंगे।

```
<html>
<body>
<% out.print(2*5); %>
</body>
</html>
```

यह ब्राउजर पर 10 प्रिंट करेगा।

एक साधारण JSP पेज कैसे चलों?

इस JSP पेज को Execute करने के लिए निम्नलिखित स्टेप का पालन करें—

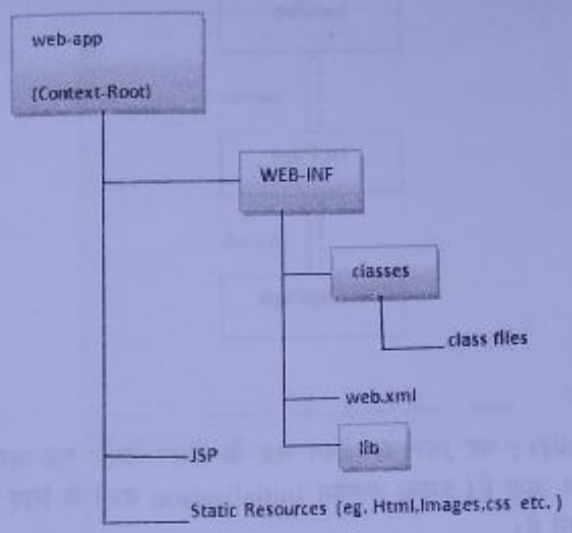
- सर्वर शुरू करें
- एक फ़ोल्डर में JSP फ़ाइल रखें और सर्वर पर deploy करें
- URL पर ब्राउज़र पर जों http: // localhost: portno / referenceRoot / jspfile, उदाहरण के लिए, http: // localhost: 8888 / myapplication / index.jsp

क्या मुझे एक साधारण JSP चलाने के लिए डायरेक्टरी स्ट्रक्चर का पालन करने की आवश्यकता है?

नहीं, यदि आपके पास क्लास फ़ाइलें या TLD फाइलें नहीं हैं, तो डायरेक्टरी स्ट्रक्चर की कोई आवश्यकता नहीं है। उदाहरण के लिए, JSP फ़ाइलों को सीधे एक फ़ोल्डर में रखें और उस फ़ोल्डर को deploy करें। यह ठीक चल रहा होगा। हालाँकि, यदि आप बीन क्लास, सर्वलेट या TLD फ़ाइल का उपयोग कर रहे हैं, तो डायरेक्टरी स्ट्रक्चर की आवश्यकता है।

JSP की डायरेक्टरी संरचना

JSP पेज की डायरेक्टरी स्ट्रक्चर सर्वलेट के समान है। हम WEB-INF फ़ोल्डर के बाहर या किसी भी डायरेक्टरी में JSP पेज रखते हैं।



5.2 JSP एपीआई (JSP API)

JSP API में दो पैकेज होते हैं—

1. javax.servlet.jsp
2. javax.servlet.jsp.tagext

javax.servlet.jsp पैकेज

Javax.servlet.jsp पैकेज में दो इंटरफेस और क्लास हैं। दो इंटरफेस इस प्रकार हैं—

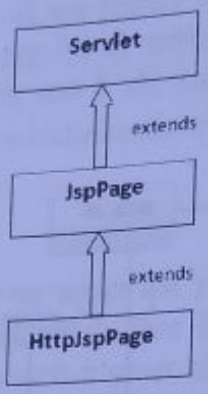
1. JspPage
2. HttpJspPage

क्लास इस प्रकार हैं—

- JspWriter
- PageContext
- JspFactory
- JspEngineInfo
- JspException
- JspErro

Jsp Page इंटरफ़ेस

JSP specificaion के अनुसार, सभी उत्पन्न सर्वलेट क्लास को Jsp Page इंटरफ़ेस को लागू करना चाहिए। यह सर्वलेट इंटरफ़ेस को एक्सटेंड करता है। यह जीवन चक्र दो मेथड्स प्रदान करता है।



JspPage इंटरफ़ेस के तरीके

1. **public void jspInit() :** यह JSP के जीवन चक्र के दौरान केवल एक बार लगाया जाता है जब JSP पेज को सबसे पहले रिक्वेस्ट किया जाता है। इसका उपयोग Initialization करने के लिए किया जाता है। यह सर्वलेट इंटरफ़ेस का init () मेथड के समान है।

2. **public void jspDestroy() :** यह JSP पेज के नष्ट होने से पहले JSP के जीवन चक्र के दौरान केवल एक बार ही किया जाता है। इसका उपयोग कुछ सफाई ऑपरेशन करने के लिए किया जा सकता है।

HttpJspPage इंटरफ़ेस

HttpJspPage इंटरफ़ेस JSP जीवन चक्र का एक मेथड प्रदान करता है। यह JspPage इंटरफ़ेस को एक्सटेंड करता है।

HttpJspPage इंटरफ़ेस का मेथड:

1. **public void _jspService() :** यह हर बार जब व्एइ पेज के लिए कंटेनर के लिए रिक्वेस्ट आता है। इसका उपयोग रिक्वेस्ट को संसाधित करने के लिए किया जाता है। अंडरस्कोर _ दर्शाता है कि आप इस मेथड को ओवरराइड नहीं कर सकते।

हम इस अध्याय में बाद में सभी अन्य क्लास और इंटरफेस को सीखेंगे।

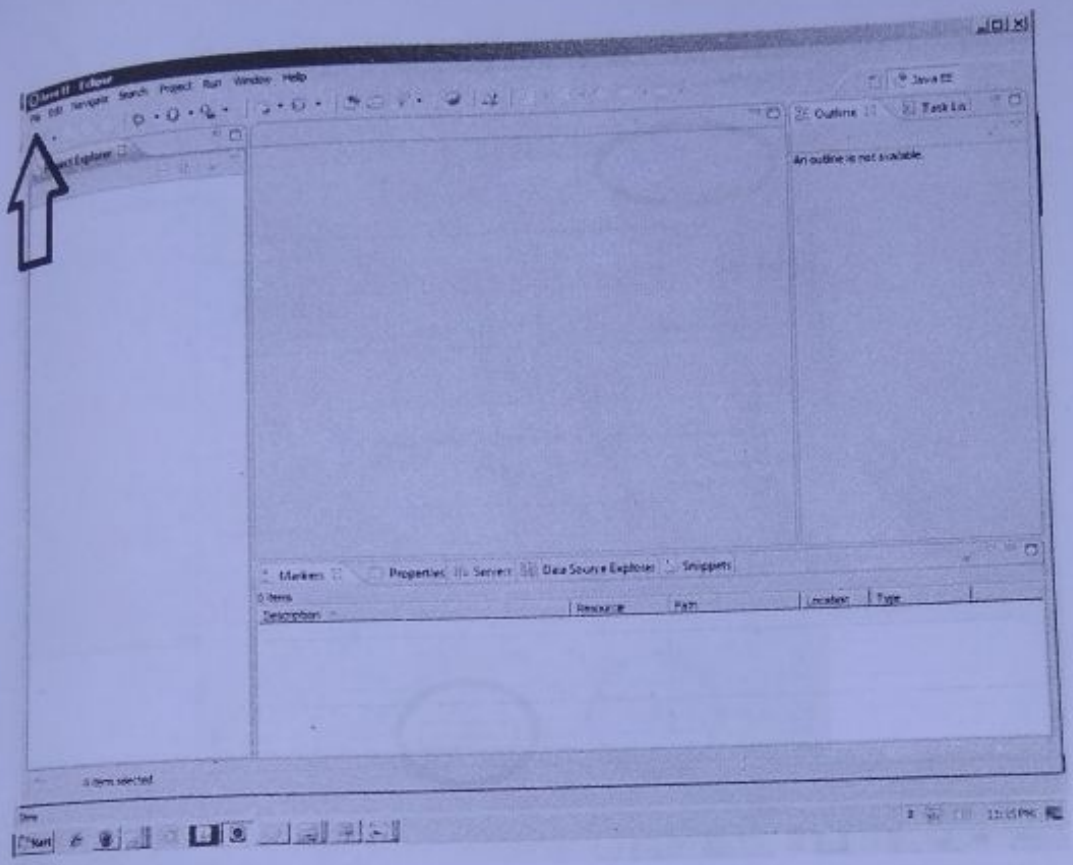
5.3 Tomcat सर्वर के साथ Eclipse IDE में JSP बनाना (Creating JSP in Eclipse IDE with Tomcat server)

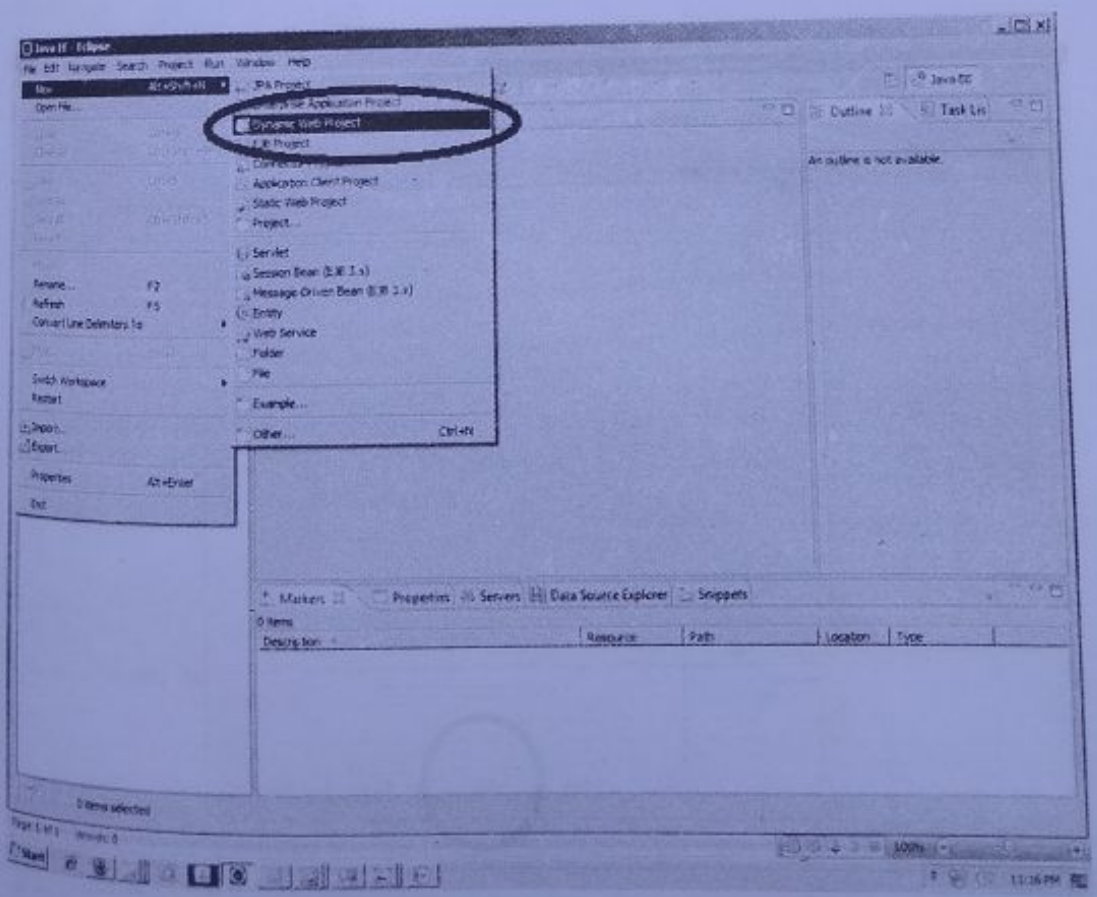
Eclipse IDE में JSP बनाने के लिए 3 स्टेप हैं—

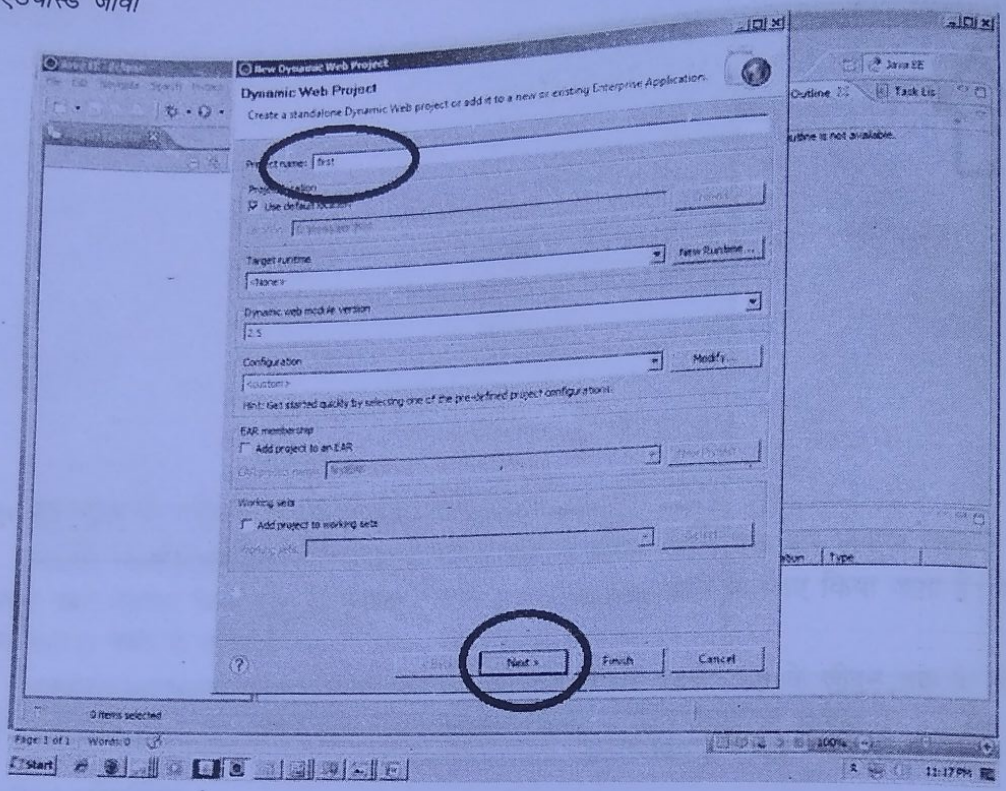
- एक डायनेमिक वेब प्रोजेक्ट बनों
- एक jsp बनों
- टॉमकैट सर्वर शुरू करें और प्रोजेक्ट को deploy करें

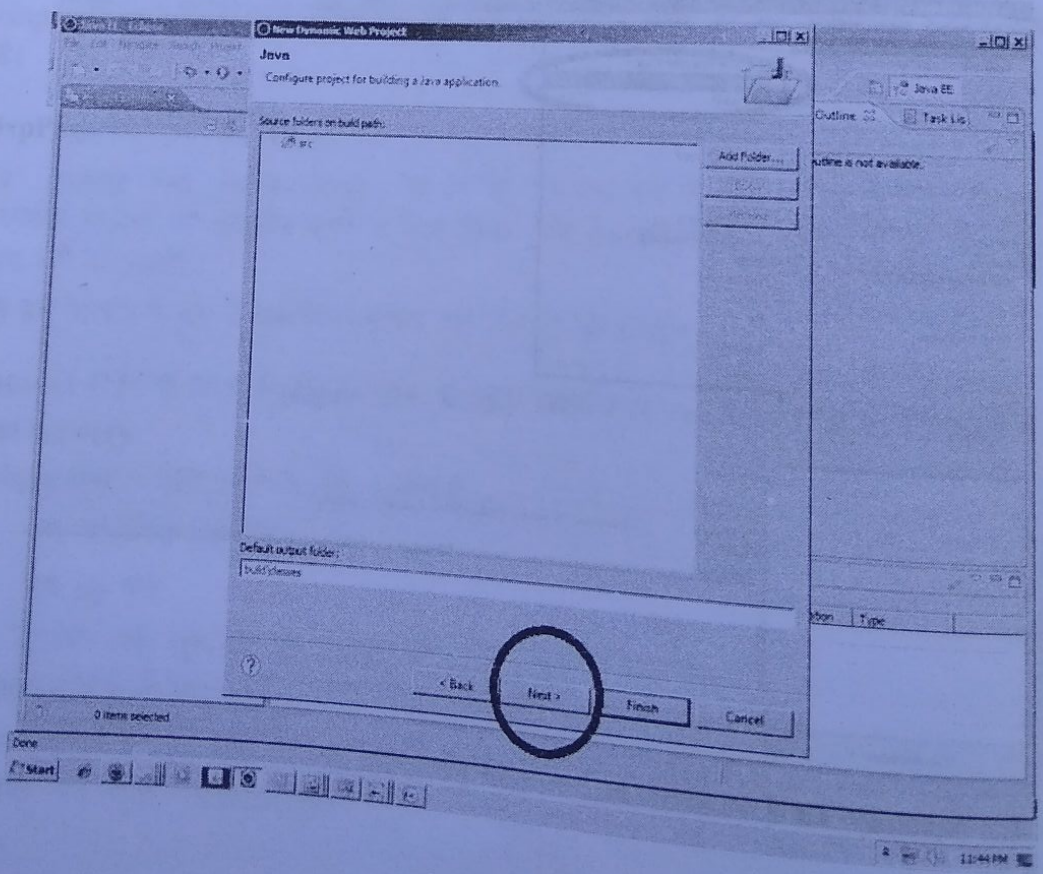
(1) डायनामिक वेब प्रोजेक्ट बनों

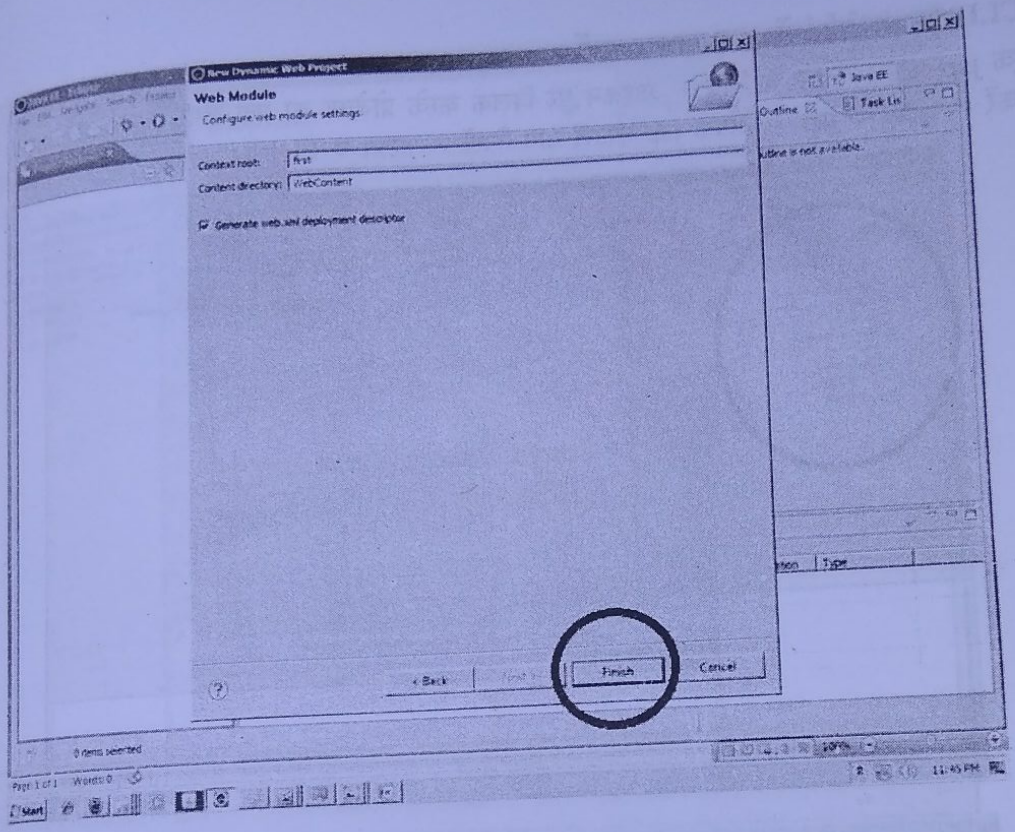
डायनामिक वेब प्रोजेक्ट बनाने के लिए फ़ाइल मेनू पर क्लिक करें -> -> New -> dynamic web project -> अपना प्रोजेक्ट नाम लिखें -> Finish करें।

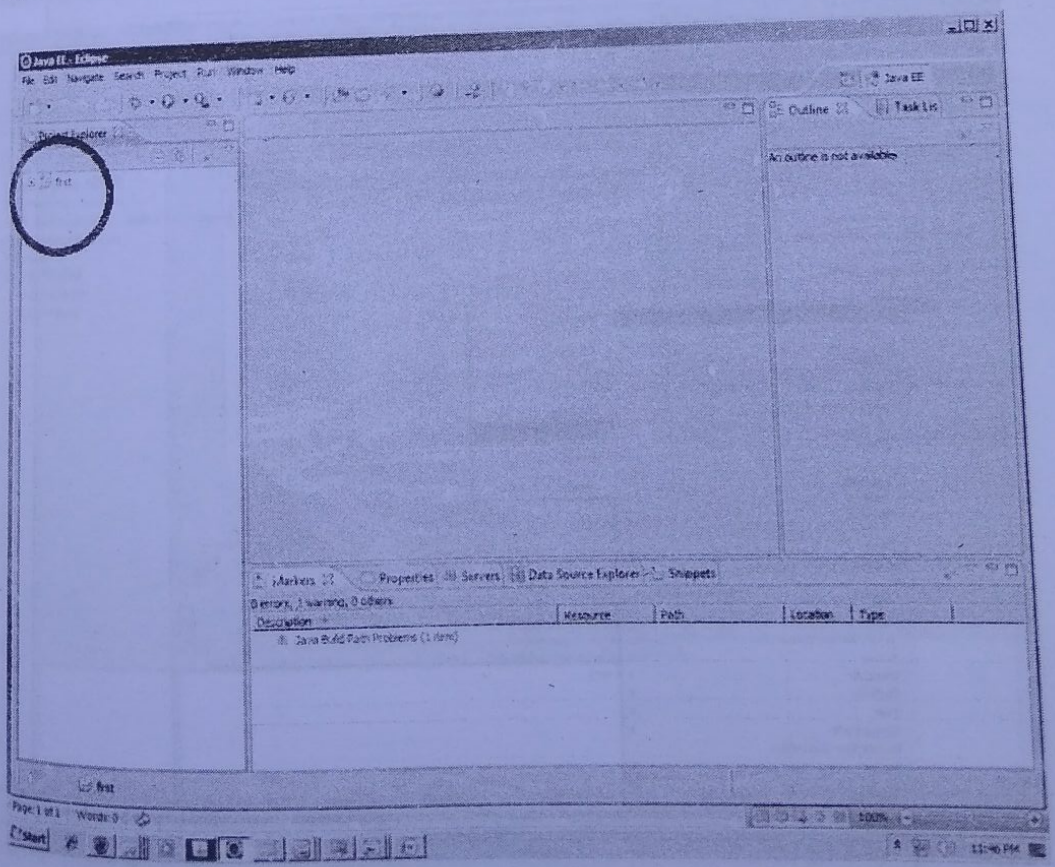

Outline
Task List
An outline is not available.
Markers
Properties
Servers
Data Source Explorer
Snippets
Description
Resource
Path
Location
Type


Dynamic Web Project
Application Client Project
Static Web Project
Project...
Servlet
Session Bean (EJB 3.x)
Message-Driven Bean (EJB 3.x)
Entity
Web Service
Folder
File
Example...
Other...
Ctrl+N
Rename...
Refresh
Convert Line Delimiters To
Switch Workspace
Restart
Properties
Alt+Enter
Exit
An outline is not available.
Markers
Properties
Servers
Data Source Explorer
Snippets
Description
Resource
Path
Location
Type


New Dynamic Web Project
Dynamic Web Project
Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.
Project name: first
Target runtime
New Runtime...
Dynamic web module version
2.5
Configuration
Modify...
EAR membership
Add project to an EAR
Working sets
Add project to working sets
Next >
Finish
Cancel

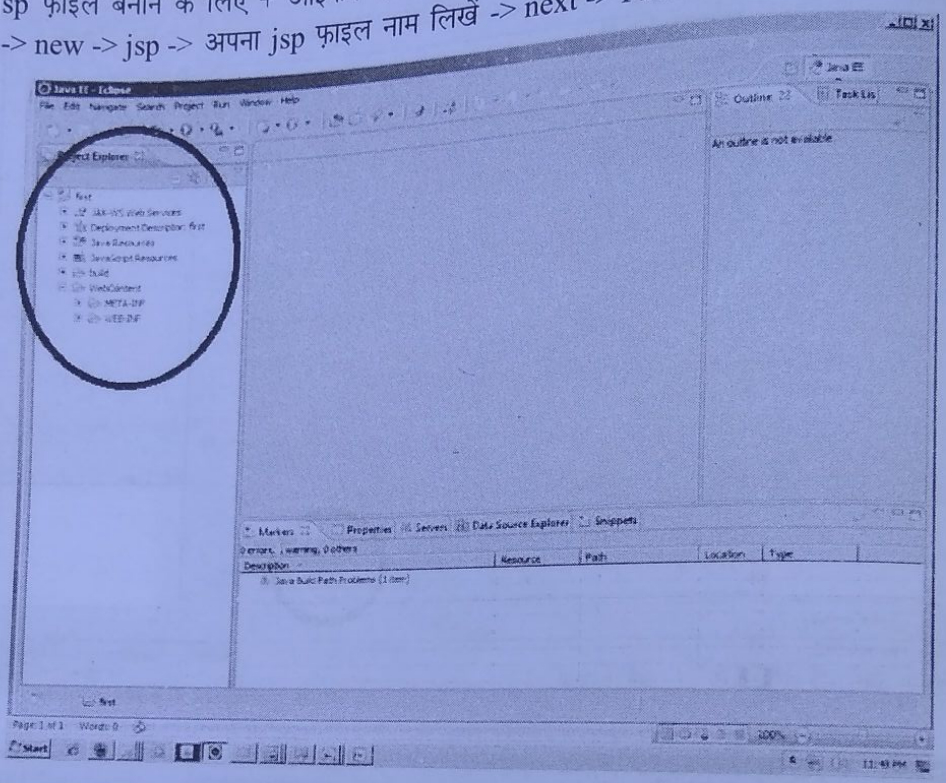

New Dynamic Web Project
Java
Configure project for building a Java application.
Source folders on build path:
src
Add Folder...
Default output folder:
build/classes
< Back
Next >
Finish
Cancel

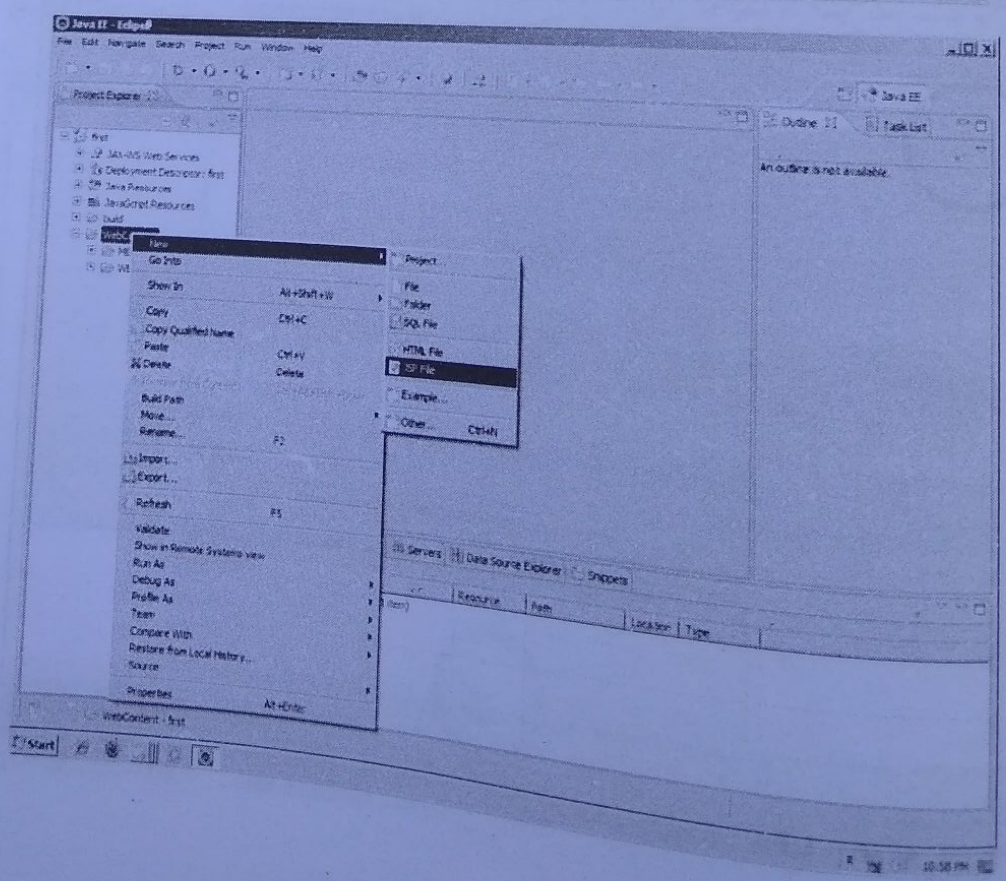

New Dynamic Web Project
Web Module
Configure web module settings
Context root:
first
Content directory:
WebContent
Generate web.xml deployment descriptor
< Back
Finish
Cancel

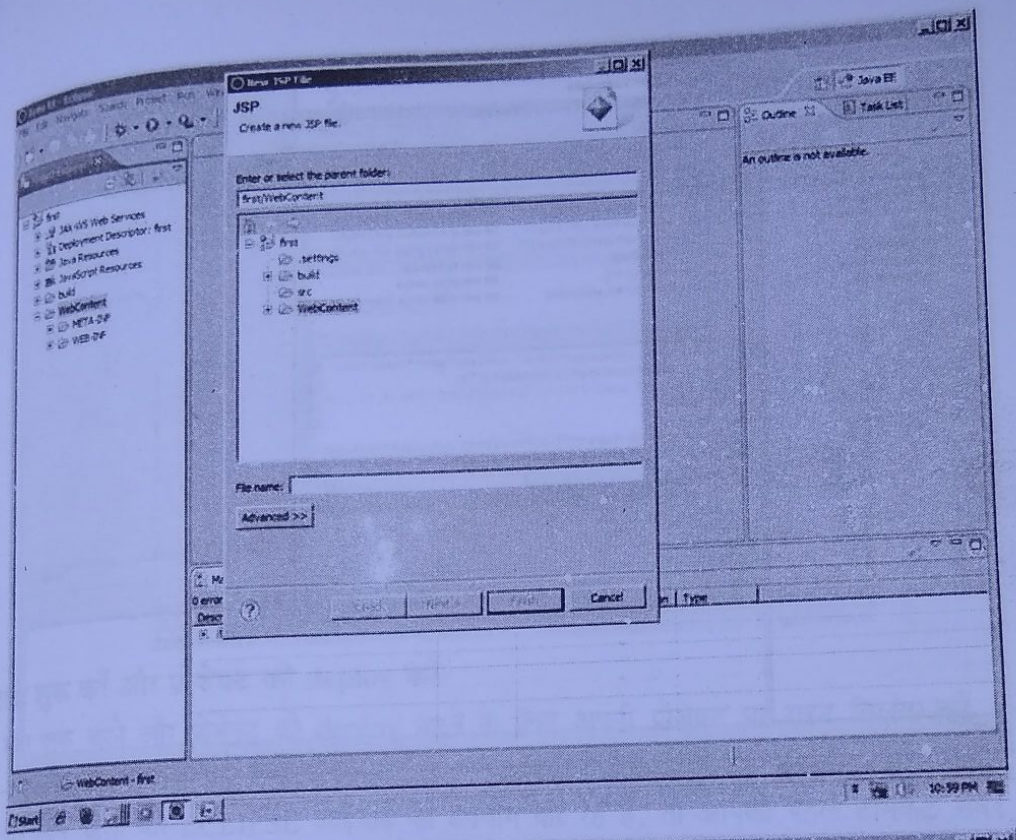

Java EE - Eclipse
File Edit Navigate Search Project Run Window Help
Project Explorer
first
Outline
Task List
An outline is not available.
Markers
Properties
Servers
Data Source Explorer
Snippets
0 errors, 1 warning, 0 others
Description
Resource
Path
Location
Type
Java Build Path Problems (1 item)

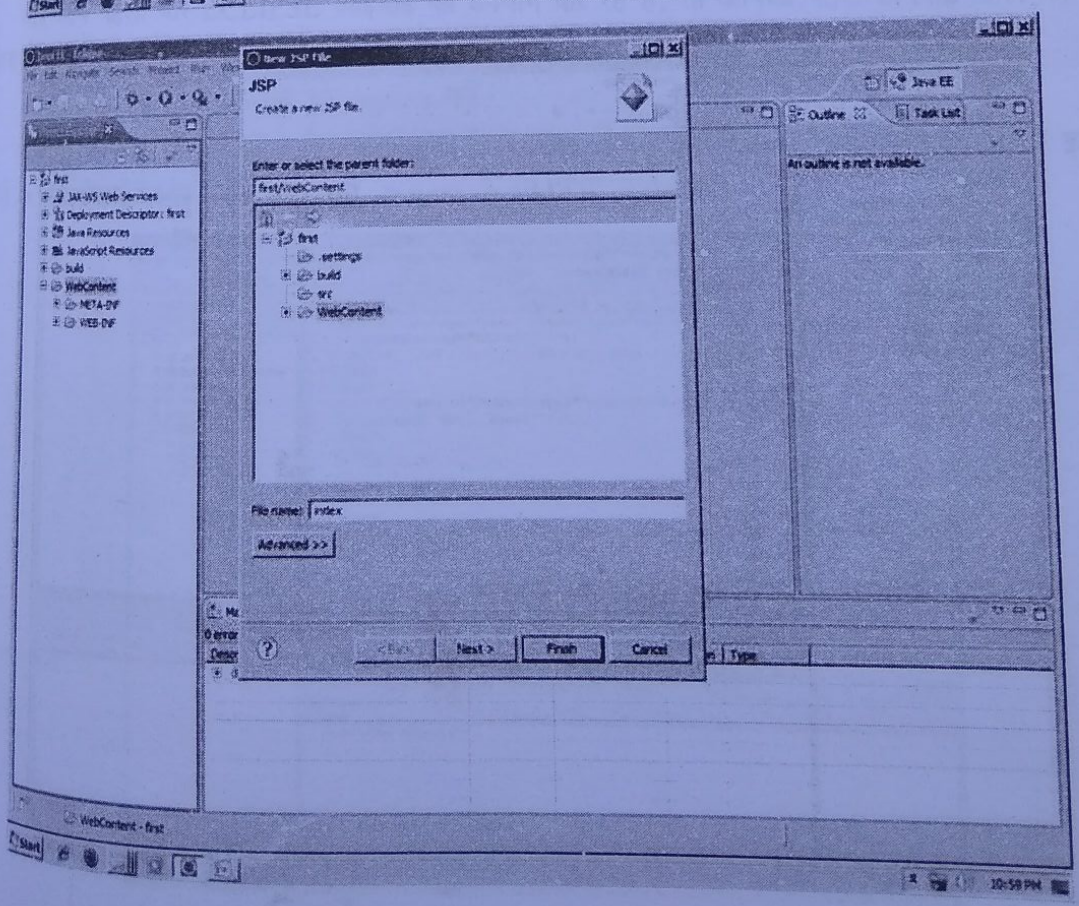
(2) ECLIPSE आईडीई में JSP फ़ाइल बनों

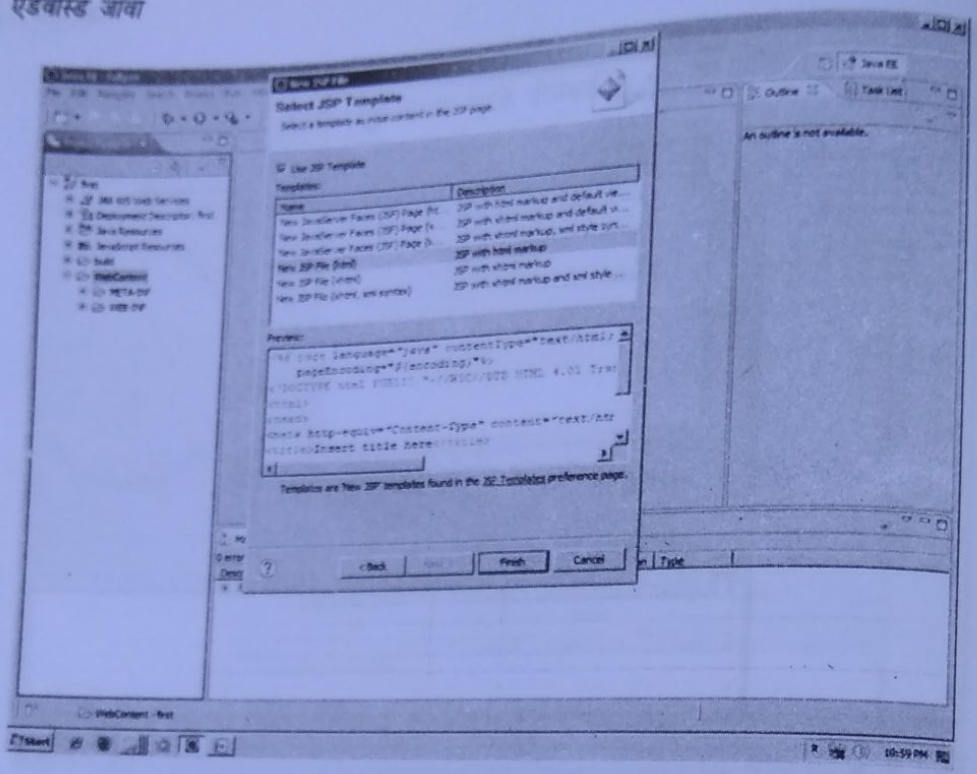
एक jsp फ़ाइल बनाने के लिए + आइकन पर क्लिक करके प्रोजेक्ट का पता लगों -> WebContent पर राइट क्लिक करें -> new -> jsp -> अपना jsp फ़ाइल नाम लिखें -> next -> Finish।

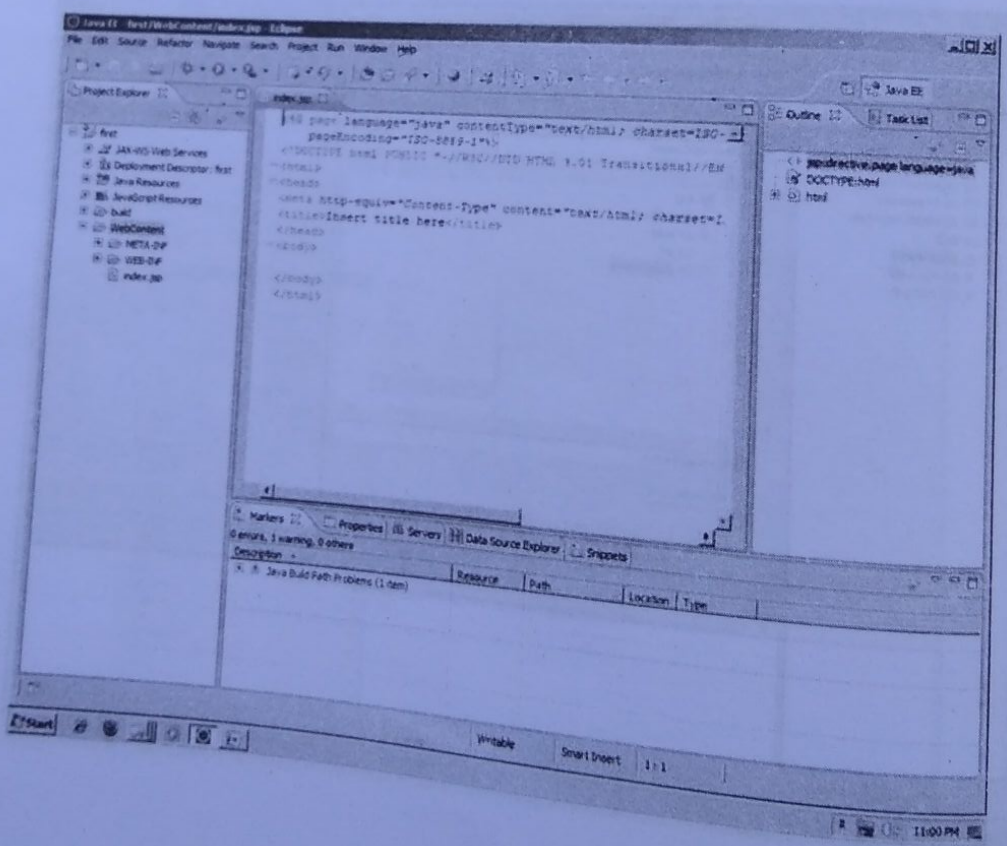




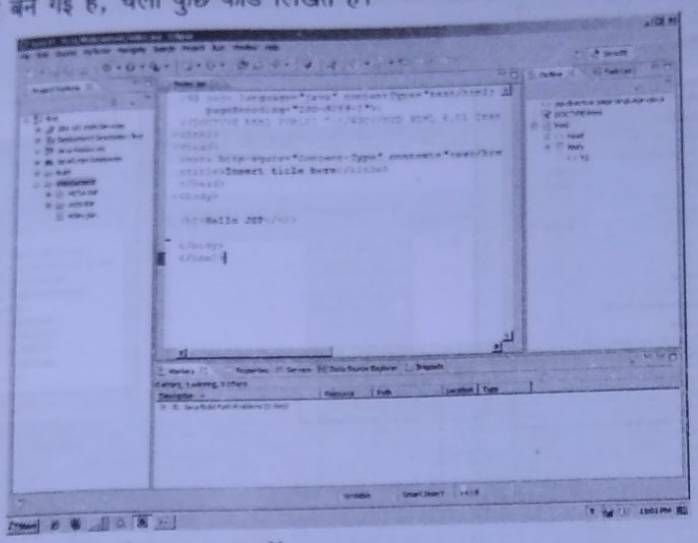

New JSP File
JSP
Create a new JSP file.
Enter or select the parent folder:
first/WebContent
first
.settings
build
src
WebContent
File name:
Advanced >>
< Back
Next >
Finish
Cancel
Java EE
Outline
Task List
An outline is not available.
Type
first
JAX-WS Web Services
Deployment Descriptor: first
Java Resources
JavaScript Resources
build
WebContent
META-INF
WEB-INF
WebContent - first
Start
10:59 PM


New JSP File
JSP
Create a new JSP file.
Enter or select the parent folder:
first/WebContent
first
.settings
build
src
WebContent
File name: index
Advanced >>
< Back
Next >
Finish
Cancel
Java EE
Outline
Task List
An outline is not available.
Type
first
JAX-WS Web Services
Deployment Descriptor: first
Java Resources
JavaScript Resources
build
WebContent
META-INF
WEB-INF
WebContent - first
Start
10:58 PM


Select JSP Template
Templates:
Name
Description
Preview:
Templates are New JSP templates found in the JSP Templates preference page.
< Back
Next >
Finish
Cancel


File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
Outline
Task List
Markers
Properties
Servers
Data Source Explorer
Snippets
Writable
Smart Insert

अब JSP फाइल बन गई है, चलो कुछ कोड लिखते हैं।



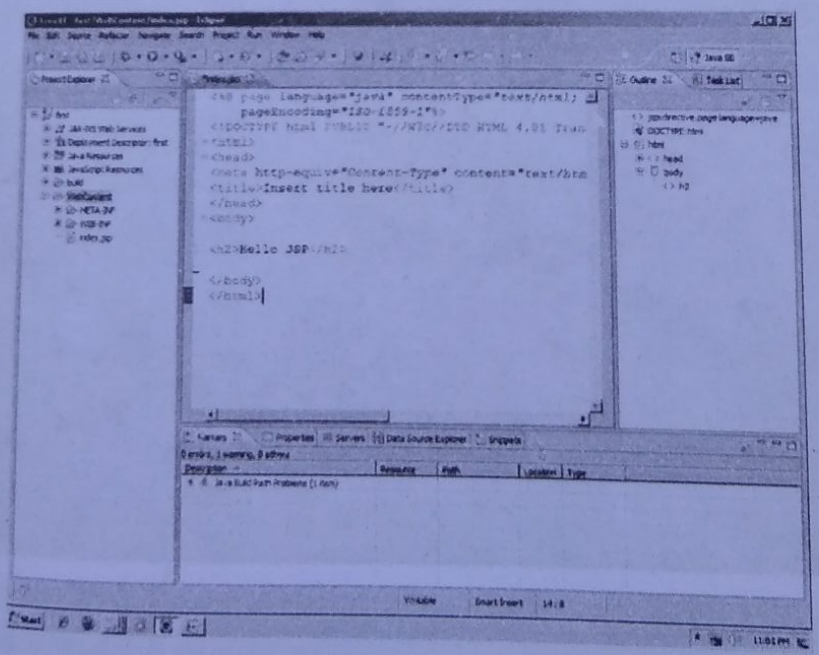
(3) सर्वर शुरू करें और प्रोजेक्ट को deploy करें

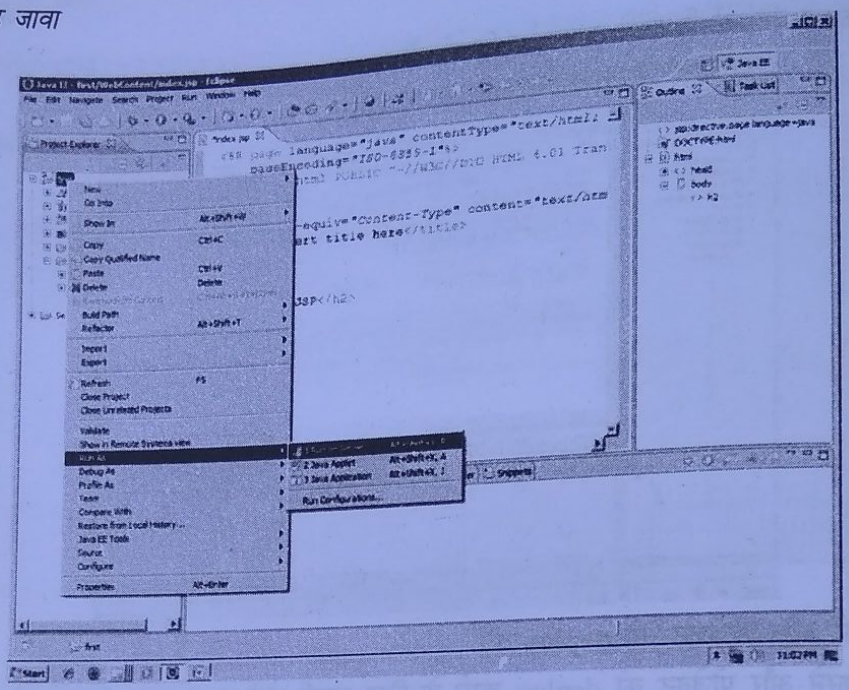
सर्वर शुरू करने और प्रोजेक्ट को deploy करने के लिए अपनी प्रोजेक्ट पर राइट क्लिक करें -> इस रूप में चलों -> सर्वर चलों -> टॉमकैट सर्वर चुनें -> next -> addAll -> finish।

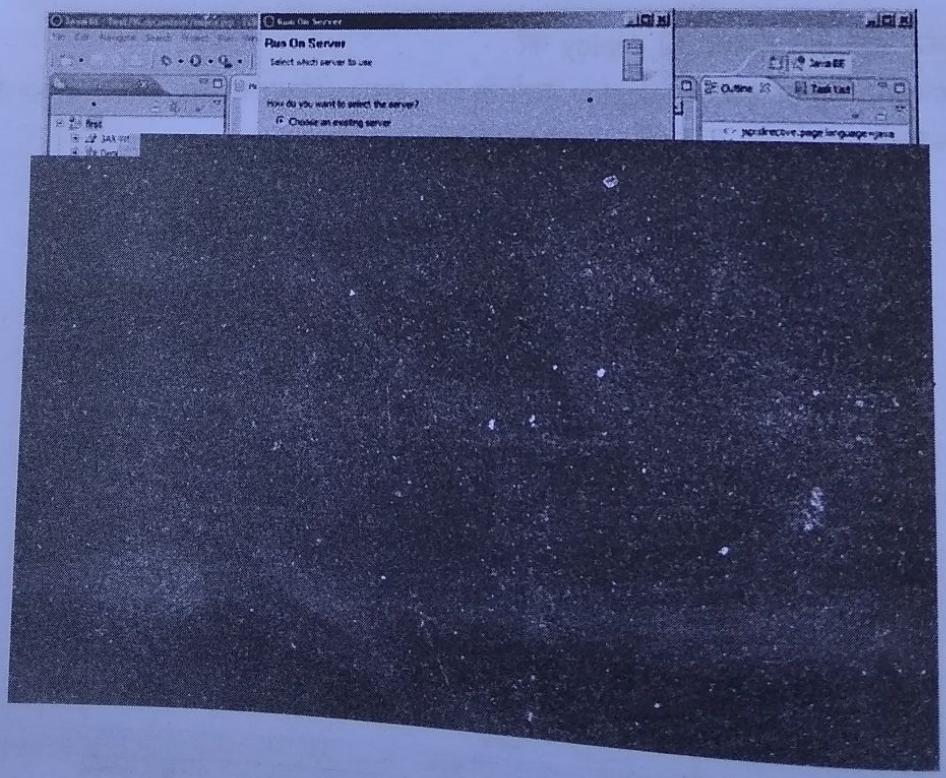
यदि आप पहली बार ECLIPSE आईडीई का उपयोग कर रहे हैं, तो आपको पहले टॉमकैट सर्वर को कॉन्फिगर करना होगा।

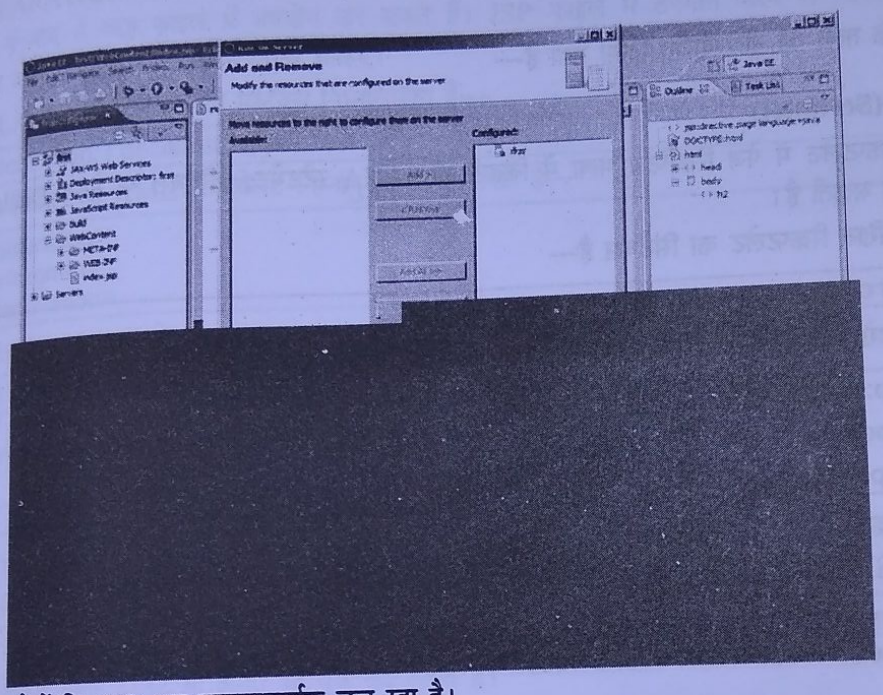
अब टॉमकैट सर्वर और प्रोजेक्ट को deploy करें

सर्वर शुरू करने और प्रोजेक्ट को deploy करने के लिए अपनी प्रोजेक्ट पर राइट क्लिक करें -> इस रूप में चलों -> सर्वर चलों -> टॉमकैट सर्वर चुनें -> next -> addAll -> finish।

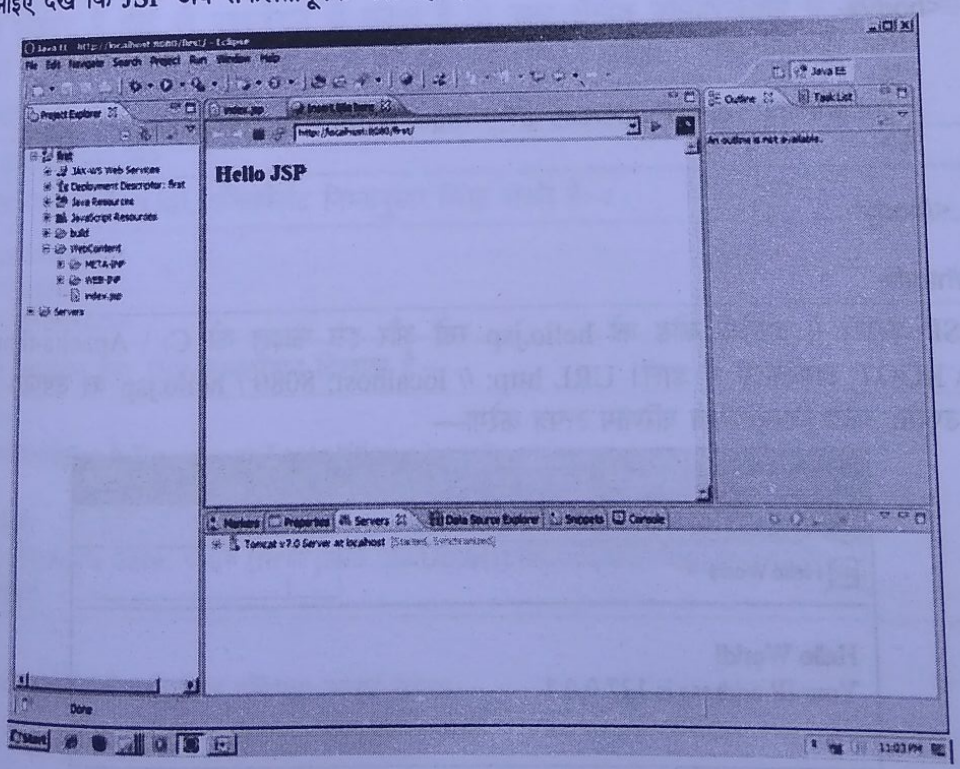





Run On Server
Select which server to use
How do you want to select the server?
Choose an existing server



हां, आइए देखें कि JSP अब सफलतापूर्वक चल रहा है।



5.4 सिंटैक्स (Syntax)

हम JSP विकास के साथ शामिल सरल सिंटैक्स (यानी, तत्वों) के मूल उपयोग को समझेंगे।

JSP के तत्व

JSP के तत्वों को नीचे वर्णित किया गया है—

स्क्रिप्टलेट (Scriptlet)

एक स्क्रिप्टलेट में पेज स्क्रिप्टिंग भाषा में कितनी भी JAVA स्टेटमेंट, वेरिएबल या मेथड Declarations, या एक्सप्रेशन हो सकती है।

निम्नलिखित स्क्रिप्टलेट का सिंटैक्स है—

```
<% code fragment %>
```

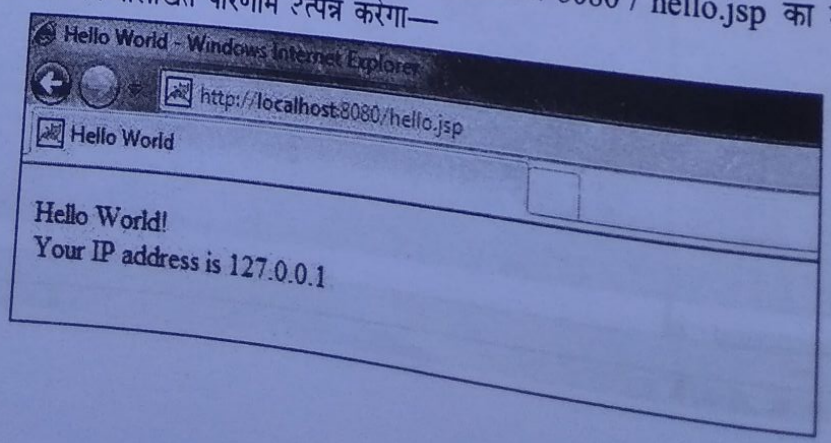
आप उपरोक्त सिंटैक्स के XML इक्विवैलेंट निम्नानुसार लिख सकते हैं—

```
<jsp:scriptlet>
  code fragment
</jsp:scriptlet>
```

आपके द्वारा लिखा गया कोई भी टेक्स्ट, HTML टैग या व्एइ तत्व स्क्रिप्टलेट के बाहर होना चाहिए। निम्नलिखित JSP के लिए सरल और पहला उदाहरण है—

```
<html>
  <head><title>Hello World</title></head>

  <body>
    Hello World!<br/>
    <%
      out.println("Your IP address is " + request.getRemoteAddr());
    %>

  </body>

</html>
```

हमें JSP फ़ाइल में उपरोक्त कोड को hello.jsp रखें और इस फाइल को C: \ Apache-tomcat7.0.2 \ webapps \ ROOT डायरेक्टरी में डालें। URL http: // localhost: 8080 / hello.jsp का उपयोग करके समान ब्राउज़ करें। उपरोक्त कोड निम्नलिखित परिणाम उत्पन्न करेगा—



JSP Declarations

एक DECLARATION एक या एक से अधिक वेरिएबल या मेथड का DECLARATION करती है, जो आप जावा कोड में बाद में व्ह्ह फ़ाइल में उपयोग कर सकते हैं। JSP फ़ाइल में उपयोग करने से पहले आपको वेरिएबल या मेथड का DECLARATION करनी चाहिए।

JSP DECLARATIONS के लिए सिंटैक्स निम्नलिखित है—

```
<%! declaration; [ declaration; ]+ ... %>
```

आप उपरोक्त सिंटैक्स के XML इक्विवैलेंट निम्नानुसार लिख सकते हैं—

```
<jsp:declaration>
  code fragment
</jsp:declaration>
```

JSP DECLARATIONS के लिए एक उदाहरण निम्नलिखित है—

```
<%! int i = 0; %>
<%! int a, b, c; %>
<%! Circle a = new Circle(2.0); %>
```

JSP एक्सप्रेशन

JSP एक्सप्रेशन एलिमेंट में एक स्क्रिप्टिंग लैंग्वेज एक्सप्रेशन होता है जिसका मूल्यांकन किया जाता है, एक स्ट्रिंग में परिवर्तित किया जाता है, और जहाँ JSP फाइल में एक्सप्रेशन दिखाई देता है, वहाँ डाला जाता है।

JSP फाइल में एक्सप्रेशन HTLM के साथ टैग की जाती है, एक एक्सप्रेशन की वैल्यू स्ट्रिंग में कन्वर्ट होती है। हम एक्सप्रेशन को लाइन ऑफ़ टेक्स्ट के अन्दर भी यूज़ कर सकते है।

एक्सप्रेशन एलिमेंट में कोई भी एक्सप्रेशन हो सकता है जो जावा लैंग्वेज स्पेसिफिकेशन के अनुसार मान्य हो लेकिन एक्सप्रेशन खत्म करने के लिए आप सेमीकॉलन का इस्तेमाल नहीं कर सकते।

JSP एक्सप्रेशन का सिंटैक्स निम्नलिखित है—

```
<%= expression %>
```

आप उपरोक्त सिंटैक्स के XML इक्विवैलेंट निम्नानुसार लिख सकते हैं—

```
<jsp:expression>
  expression
</jsp:expression>
```

निम्नलिखित उदाहरण एक JSP एक्सप्रेशन दिखाता है—

```
<html>
  <head><title>A Comment Test</title></head>

  <body>
    <p>Today's date: <%= (new java.util.Date()).toLocaleString()%></p>
  </body>
```

</html उपरोक्त कोड निम्नलिखित परिणाम उत्पन्न करेगा—

आज की तारीखः 11-सितंबर-2010 21:24:25

JSP कमेंट

JSP कमेंट पाठ या स्टेटमेंट को चिह्नित करती है जिसे JSP कंटेनर को अनदेखा करना चाहिए। एक JSP कमेंट तब उपयोगी होती है जब आप अपने JSP पेज का एक हिस्सा छुपाना या 'कमेंट आउट' करना चाहते हैं।

निम्नलिखित JSP कमेंट का सिंटैक्स है—

```
<%-- This is JSP comment --%>
```

निम्नलिखित उदाहरण JSP कमेंट दिखाता है—

```
<html>
  <head><title>A Comment Test</title></head>

  <body>
    <h2>A Test of Comments</h2>
    <%-- This comment will not be visible in the page source --%>
  </body>
</html>
```

उपरोक्त कोड निम्नलिखित परिणाम उत्पन्न करेगा—

A Test of Comments

कम संख्या में ऐसे विशेष निर्माण हैं जिनका उपयोग आप विभिन्न मामलों में कमेंट या करैक्टर को सम्मिलित करने के लिए कर सकते हैं जिन्हें अन्यथा विशेष रूप से माना जाएगा। यहाँ एक सारांश है—

क्र.सं.	सिंटेक्स और प्रयोजन
1	**<%-- comment --%>** एक JSP कमेंट। JSP इंजन द्वारा अनदेखा किया गया।
2	**<!-- comment -->** एक HTML कमेंट। ब्राउज़र द्वारा अनदेखा किया गया।
3	**<\%** स्टेटिक <% शाब्दिक का प्रतिनिधित्व करता है।
4	**% \>** स्टेटिक%> शाब्दिक का प्रतिनिधित्व करता है।
5	\' एक एट्रिब्यूट में एक एकल कोट्स जो एकल कोट्स का उपयोग करता है।
6.	\' एक एट्रिब्यूट में एक डबल कोट्स जो दोहरे कोट्स चिह्नों का उपयोग करता है।

Control-Flow Statements

आप अपने JSP प्रोग्रामिंग में जावा के सभी एपीआई और बिल्डिंग ब्लॉक्स का उपयोग कर सकते हैं, जिसमें निणर्य लेने वाले स्टेटमेंट, लूप आदि शामिल हैं।

Decision-Making Statements

if...else ब्लॉक एक सामान्य स्क्रिप्टलेट की तरह शुरू होता है, लेकिन स्क्रिप्टलेट HTML टैग्स के बीच शामिल HTML पाठ के साथ प्रत्येक Row में बंद है।

```
<%! int day = 3; %>
<html>
  <head><title>IF...ELSE Example</title></head>

  <body>
    <% if (day == 1 || day == 7) { %>
      <p> Today is weekend</p>
    <% } else { %>
      <p> Today is not weekend</p>
    <% } %>
  </body>
</html>
```

उपरोक्त कोड निम्नलिखित परिणाम उत्पन्न करेगा—

Today is not weekend

अब नीचे दिए गए स्विच ... केस ब्लॉक को देखें जिसे out.println () का उपयोग करके थोड़ा अलग लिखा गया है—

```
<%! int day = 3; %>
<html>
  <head><title>SWITCH...CASE Example</title></head>

  <body>
    <%
      switch(day) {
        case 0:
          out.println("It\'s Sunday.");
          break;
        case 1:
          out.println("It\'s Monday.");
          break;
        case 2:
          out.println("It\'s Tuesday.");
          break;
        case 3:
          out.println("It\'s Wednesday.");
          break;
        case 4:
          out.println("It\'s Thursday.");
          break;
        case 5:
          out.println("It\'s Friday.");
          break;
```

```
        default:
          out.println("It's Saturday.");
      }
    %>
  </body>
</html>
```

उपरोक्त कोड निम्नलिखित परिणाम उत्पन्न करेगा—

It's Wednesday.

लूप स्टेटमेंट

आप जावा में तीन बुनियादी प्रकार के लूपिंग ब्लॉकों का उपयोग कर सकते हैं—for, while, और ... do...while.

आइए हम for लूप का उदाहरण देखें—

```
<%! int fontSize; %>
<html>
  <head><title>FOR LOOP Example</title></head>

  <body>
    <%for ( fontSize = 1; fontSize <= 3; fontSize++){ %>
      <font color = "green" size = "<%= fontSize %>">
        JSP Tutorial
    </font><br />
    <%}%>
  </body>
</html>
```

उपरोक्त कोड निम्नलिखित परिणाम उत्पन्न करेगा—

```
JSP Tutorial

JSP Tutorial

JSP Tutorial
```

उपरोक्त उदाहरण को while लूप का उपयोग करते हुए लिखा जा सकता है—

```
<%! int fontSize; %>
<html>
  <head><title>WHILE LOOP Example</title></head>

  <body>
    <%while ( fontSize <= 3){ %>
      <font color = "green" size = "<%= fontSize %>">
        JSP Tutorial
      </font><br />
      <%fontSize++;%>
    <%}%>
  </body>
</html>
```

उपरोक्त कोड निम्नलिखित परिणाम उत्पन्न करेगा—

```
JSP Tutorial

JSP Tutorial

JSP Tutorial
```

JSP ऑपरेटर्स

JSP जावा द्वारा समर्थित सभी तार्किक और अंकगणितीय ऑपरेटरों का समर्थन करता है। एक एक्सप्रेशन के भीतर, High Precedence ऑपरेटरों का मूल्यांकन पहले किया जाएगा।

क्लास	ऑपरेटर	संबद्धता
Postfix	() [] . (dot operator)	बाएं से दाएं
Unary	++ - - ! ~	दाएं से बाएं
Multiplicative	* / %	बाएं से दाएं
Additive	+ -	बाएं से दाएं
Shift	>> >>> <<	बाएं से दाएं
Relational	> >= < <=	बाएं से दाएं
Equality	== !=	बाएं से दाएं

Bitwise AND	&	बाएं से दाएं
Bitwise XOR	^	बाएं से दाएं
Bitwise OR	|	बाएं से दाएं
Logical AND	&&	दाएं से बाएं
Logical OR	||	बाएं से दाएं
Conditional	?:	दाएं से बाएं
Assignment	= += -= *= /= %= >>= <<= &= ^= |=	दाएं से बाएं
Comma	,	बाएं से दाएं

JSP Literals

JSP एक्सप्रेशन भाषा निम्नलिखित Literals को परिभाषित करती है—

- **Boolean** - सच और झूठ
- **Integer** जैसा कि जावा में है
- **Floating point** - जैसा कि जावा में है
- **String**- सिंगल और डबल कोट्स के साथ
- **Null** - अशक्त

5.5 JSP स्क्रिप्टिंग तत्व

स्क्रिप्टिंग तत्व jsp के अंदर java code डालने की क्षमता प्रदान करता है। स्क्रिप्टिंग तत्व तीन प्रकार के होते हैं—

- scriptlet टैग
- एक्सप्रेशन टैग
- DECLARATION टैग

JSP scriptlet टैग

JSP सोर्स कोड को JSP में Execute करने के लिए एक स्क्रिप्टलेट टैग का उपयोग किया जाता है। सिंटेक्स इस प्रकार है—

```
<% java source code %>
```

JSP scriptlet टैग का उदाहरण

इस उदाहरण में, हम एक स्वागत योग्य संदेश प्रदर्शित कर रहे हैं।

```
<html>
<body>
<% out.print("welcome to jsp"); %>
```

```
</body>
</html>
```

JSP scriptlet टैग का उदाहरण जो यूजर के नाम को प्रिंट करता है

इस उदाहरण में, हमने दो फाइलें बनाई हैं index.html और wel.jsp। Index.html फ़ाइल को यूजर से यूजर नाम मिलता है और wel.jsp फ़ाइल यूजर को स्वागत संदेश के साथ प्रिंट करती है।

फाइल: index.html

```
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
</body>
</html>
```

फाइल: welcome.jsp

```
<html>
<body>
<%
String name=request.getParameter("uname");
out.print("welcome "+name);
%>
</form>
</body>
</html>
```

JSP एक्सप्रेशन टैग

JSP एक्सप्रेशन टैग के भीतर रखा कोड, प्रतिक्रिया की आउटपुट स्ट्रीम के लिए लिखा है। इसलिए आपको डेटा लिखने के लिए out.print () नहीं लिखना होगा। यह मुख्य रूप से वेरिएबल या मेथड के मूल्यों को प्रिंट करने के लिए उपयोग किया जाता है।

JSP एक्सप्रेशन टैग का सिंटैक्स

```
<%= statement %>
```

JSP एक्सप्रेशन टैग का उदाहरण

JSP एक्सप्रेशन टैग के इस उदाहरण में, हम बस एक स्वागत संदेश प्रदर्शित कर रहे हैं।

```
<html>
<body>
<%= "welcome to jsp" %>
</body>
</html>
```

JSP एक्सप्रेशन टैग का उदाहरण जो वर्तमान समय को प्रिंट करता है

वर्तमान समय को प्रदर्शित करने के लिए, हमने कैलेंडर क्लास की getTime () मेथड का उपयोग किया है। getTime () कैलेंडर क्लास का एक इनस्टेंस मेथड है, इसलिए हमने इसे getInstance() () मेथड द्वारा कैलेंडर क्लास का उदाहरण प्राप्त करने के बाद कहा है।

index.jsp

```
<html>
<body>
Current Time: <%= java.util.Calendar.getInstance().getTime() %>
</body>
</html>
```

JSP एक्सप्रेशन टैग का उदाहरण जो यूजर के नाम को प्रिंट करता है

इस उदाहरण में, हम एक्सप्रेशन टैग का उपयोग करके यूजर नाम प्रिंट कर रहे हैं। Index.html फ़ाइल को यूजर नाम मिलता है और वह wel.jsp फ़ाइल को रिक्वेस्ट भेजता है, जो यूजर नाम प्रदर्शित करता है।

फाइल: index.jsp

```
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname"><br/>
<input type="submit" value="go">
</form>
```

```
</body>
</html>
```

फाइल: *welcome.jsp*

```
<html>
<body>
<%= "Welcome "+request.getParameter("uname") %>
</body>
</html>
```

JSP DECLARATION टैग

JSP DECLARATION टैग का उपयोग फील्ड और मेथड को घोषित करने के लिए किया जाता है।

Jsp DECLARATION टैग के अंदर लिखा कोड ऑटो जेनरेट सर्वलेट के service () मेथड के बाहर रखा गया है।

इसलिए यह प्रत्येक रिक्वेस्ट पर स्मृति प्राप्त नहीं करता है।

JSP DECLARATION टैग का सिंटैक्स

DECLARATION टैग का सिंटैक्स निम्नानुसार है—

```
<%! field or method declaration %>
```

JSP स्क्रिप्टलेट टैग और DECLARATION टैग के बीच अंतर

Jsp स्क्रिप्टलेट टैग	Jsp DECLARATION टैग
Jsp स्क्रिप्टलेट टैग केवल वेरिएबल को घोषित कर सकता है, मेथड को नहीं।	स्क्रिप्ट टैग की DECLARATION को _jspService () मेथड के अंदर रखा गया है।
Jsp DECLARATION टैग वैरिएबल के साथ-साथ मेथड का भी DECLARATION कर सकता है।	Jsp DECLARATION टैग की DECLARATION _jspService () मेथड के बाहर रखा गया है।

फील्ड की DECLARATION करने वाले JSP DECLARATION टैग का उदाहरण

JSP DECLARATION टैग के इस उदाहरण में, हम फील्ड की DECLARATION कर रहे हैं और JSP एक्सप्रेशन टैग का उपयोग करके घोषित फील्ड के मूल्य को प्रिंट कर रहे हैं।

index.jsp

```
<html>
<body>
<%! int data=50; %>
<%= "Value of the variable is:"+data %>
```

```
</body>
</html>
```

JSP DECLARATION टैग का उदाहरण जो मेथड की DECLARATION करता है

JSP DECLARATION टैग के इस उदाहरण में, हम उस मेथड को परिभाषित कर रहे हैं जो दिए गए संख्या के घन को लौटाता है और इस मेथड को jsp एक्सप्रेशन टैग से कॉल करता है। लेकिन हम घोषित मेथड को कॉल करने के लिए jsp स्क्रिप्टलेट टैग का भी उपयोग कर सकते हैं।

index.jsp

```
<html>
<body>
<%!
int cube(int n){
return n*n*n*;
}
%>
<%= "Cube of 3 is:"+cube(3) %>
</body>
</html>
```

5.6 JSP-डायरेक्टिव (JSP-Directive)

JSP डायरेक्टिव वो संदेश हैं जो वेब कंटेनर को बताते हैं कि JSP पेज को संबंधित सर्वलेट में कैसे ट्रांसलेट किया जाए।

तीन प्रकार के डायरेक्टिव हैं—

- page directive
- include directive
- taglib directive

JSP डायरेक्टिव का सिंटैक्स

```
<%@ directive attribute="value" %>
```

JSP पेज डायरेक्टिव

पेज डायरेक्टिव संपूर्ण JSP पेज पर लागू होने वाली एट्रिब्यूट को परिभाषित करता है।

JSP पेज डायरेक्टिव का सिंटैक्स

```
<%@ page attribute="value" %>
```

करता है

रहे हैं जो दिए गए संख्या
म घोषित मेथड को कॉल

सर्वलेट में कैसे ट्रांसलेट

JSP पेज के डायरेक्टिव के एट्रिब्यूट

- import
- contentType
- extends
- info
- buffer
- language
- isELIgnored
- isThreadSafe
- autoFlush
- session
- pageEncoding
- errorPage
- isErrorPage

(1) IMPORT

IMPORT एट्रिब्यूट का उपयोग क्लास, इंटरफ़ेस या पैकेज के सभी सदस्यों को IMPORT करने के लि किया जाता है।

यह जावा क्लास या इंटरफ़ेस में कीवर्ड IMPORT करने के समान है।

IMPORT एट्रिब्यूट का उदाहरण

```
<html>
<body>

<%@ page import="java.util.Date" %>
Today is: <%= new Date() %>

</body>
</html>
```

(2) contentType

ContentType एट्रिब्यूट HTTP प्रतिक्रिया के MIME टाइप को परिभाषित करती है। डिफ़ॉल्ट 'text/html;charset=ISO-8859-1' है।

ContentType एट्रिब्यूट का उदाहरण

```
<html>
<body>

<%@ page contentType=application/msword %>
```

```
Today is: <%= new java.util.Date() %>

</body>
</html>
```

(3) extends

extends एट्रिब्यूट parent क्लास को परिभाषित करती है जो उत्पन्न सर्वलेट द्वारा inherit है। यह शायद ही कभी उपयोग किया जाता है।

(4) info

यह एट्रिब्यूट केवल JSP पेज की जानकारी सेट करती है जिसे बाद में प्राप्त किया जाता है, getServletInfo () सर्वलेट इंटरफ़ेस का मेथड का उपयोग करके।

infoएट्रिब्यूट का उदाहरण

```
<html>
<body>

<%@ page info="composed by ASIAN PUBLISHERS" %>
Today is: <%= new java.util.Date() %>

</body>
</html>
```

वेब कंटेनर परिणामी सर्वलेट में एक मेथड getServletInfo() बनाएगा। उदाहरण के लिए—

```
public String getServletInfo() {
 return "composed by ASIAN PUBLISHERS";
}
```

(5) बफर

बफर एट्रिब्यूट JSP पेज द्वारा उत्पन्न आउटपुट को संभालने के लिए किलोबाइट में बफर आकार सेट करता है। बफर का डिफ़ॉल्ट आकार 8Kb है।

बफर एट्रिब्यूट का उदाहरण

```
<html>
<body>

<%@ page buffer="16kb" %>
Today is: <%= new java.util.Date() %>

</body>
</html>
```

(6) language

language एट्रिब्यूट JSP पेज में प्रयुक्त स्क्रिप्टिंग भाषा को निर्दिष्ट करती है। डिफ़ॉल्ट मान 'जावा' है।

(7) isELIgnored

हम isELIgnored एट्रिब्यूट द्वारा jsp में एक्सप्रेशन भाषा को अनदेखा कर सकते हैं। डिफ़ॉल्ट रूप से इसका मान गलत (false) है अर्थात एक्सप्रेशन भाषा डिफ़ॉल्ट रूप से सक्षम है।

```
<%@ page isELIgnored="true" %>//Now EL will be ignored
```

(8) isThreadSafe

सर्वलेट और JSP दोनों ही मल्टीथ्रेडेड हैं। यदि आप JSP पेज के इस व्यवहार को नियंत्रित करना चाहते हैं, तो आप पेज डायरेक्टिव का isThreadSafe एट्रिब्यूट का उपयोग कर सकते हैं। isThreadread श्रेणी के मूल्य का मूल्य सत्य (true) है। यदि आप इसे गलत (false) बनाते हैं, तो वेब कंटेनर कई अनुरोधों को क्रमबद्ध करेगा, यानी यह तब तक इंतजार करेगा जब तक कि JSP किसी अन्य रिक्वेस्ट को पारित करने से पहले रिक्वेस्ट का जवाब नहीं दे देता। isThreadSafe एट्रिब्यूट का मान है—

```
<%@ page isThreadSafe="false" %>
```

ऐसे मामले में वेब कंटेनर, सर्वलेट उत्पन्न करेगा:

```
public class SimplePage_jsp extends HttpJspBase
 implements SingleThreadModel{
.......
}
```

(9) errorPage

त्रुटि पेज को परिभाषित करने के लिए errorPage एट्रिब्यूट का उपयोग किया जाता है, यदि वर्तमान पेज में एक्सेप्शन होता है, तो इसे त्रुटि पेज पर पुनर्निर्देशित किया जाएगा।

ErrorPage एट्रिब्यूट का उदाहरण

```
//index.jsp
<html>
<body>

<%@ page errorPage="myerrorpage.jsp" %>

<%= 100/0 %>

</body>
</html>
```

(10) IsErrorPage है

IsErrorPage एट्रिब्यूट का उपयोग यह घोषित करने के लिए किया जाता है कि वर्तमान पेज त्रुटि पेज है।

IsErrorPage एट्रिब्यूट का उदाहरण

```
//myerrorpage.jsp
<html>
<body>

<%@ page isErrorPage="true" %>

 Sorry an exception occured!<br/>
The exception is: <%= exception %>

</body>
</html>
```

Jsp Include Directive

Jsp Include डायरेक्टिव का उपयोग किसी भी रिसोर्सेज की कंटेंट को शामिल करने के लिए किया जाता है, जो jsp फ़ाइल, html फ़ाइल या text फ़ाइल हो सकती है। शामिल डायरेक्टिव में पेज ट्रांसलेशन टाइम में शामिल रिसोर्सेज की मूल कंटेंट शामिल है (JSP पेज का केवल एक बार ट्रांसलेट किया गया है, इसलिए स्टेटिक रिसोर्सेज को शामिल करना बेहतर होगा)।

Jsp Include डायरेक्टिव के लाभः

कोड पुनः प्रयोज्य

Jsp Include डायरेक्टिव का सिंटैक्स

```
<%@ include file="resourceName" %>
```

Jsp Include डायरेक्टिव का उदाहरण

इस उदाहरण में, हम header.html file की कंटेंट को शामिल कर रहे हैं। इस उदाहरण को चलाने के लिए आपको एक header.html फ़ाइल बनाना होगा।

```
<html>
<body>

<%@ include file="header.html" %>

Today is: <%= java.util.Calendar.getInstance().getTime() %>

</body>
</html>
```

JSP Taglib डायरेक्टिव

JSP **Taglib** डायरेक्टिव का उपयोग टैग लाइब्रेरी को परिभाषित करने के लिए किया जाता है जो कई टैग को परिभाषित करता है। हम टैग को परिभाषित करने के लिए TLD (Tag Library Descriptor) फ़ाइल का उपयोग करते हैं। कस्टम टैग सेक्शन में हम इस टैग का उपयोग करेंगे ताकि कस्टम टैग में इसे सीखना बेहतर होगा।

JSP Taglib डायरेक्टिव का सिंटेक्स

```
<%@ taglib uri="uriofthetaglibrary" prefix="prefixoftaglibrary" %>
```

JSP Taglib डायरेक्टिव का उदाहरण

इस उदाहरण में, हम currentDate नाम के अपने टैग का उपयोग कर रहे हैं। इस टैग का उपयोग करने के लिए हमें Taglib डायरेक्टिव को निर्दिष्ट करना होगा ताकि कंटेनर को टैग के बारे में जानकारी मिल सके।

```
<html>
<body>

<%@ taglib uri="http://www.asianpublishers.com/tags" prefix="mytag" %>

<mytag:currentDate/>

</body>
</html>
```

5.7 JSP एक्शन

कई JSP एक्शन टैग या तत्व हैं। प्रत्येक JSP एक्शन टैग का उपयोग कुछ विशिष्ट कार्यों को करने के लिए किया जाता है।

एक्शन टैग का उपयोग पृष्ठों के बीच प्रवाह को नियंत्रित करने और जावा बीन का उपयोग करने के लिए किया जाता है। Jsp एक्शन टैग नीचे दिए गए हैं।

JSP एक्शन टैग	स्टेटमेंट
jsp:forward	किसी अन्य रिसोर्सेज के लिए रिक्वेस्ट और प्रतिक्रिया फॉरवर्ड करता है।
jsp:include	एक और रिसोर्सेज शामिल हैं।
jsp:useBean	बीन ऑब्जेक्ट को बनाता है या उसका पता लगाता है।
jsp:setProperty	बीन ऑब्जेक्ट में प्रॉपर्टी का मूल्य निर्धारित करता है।
jsp:getProperty	बीन की प्रॉपर्टी का मूल्य छापता है।
jsp:plugin	एप्लेट जैसे अन्य घटकों को एम्बेड करता है।
jsp:param	पैरामीटर मान सेट करता है। इसका उपयोग फॉरवर्ड और इन्क्लूड में होता है ।

jsp:fallback	यदि संदेश काम कर रहा है तो प्लग इन का उपयोग किया जा सकता है। इसका उपयोग jsp: plugin में किया जाता है।

Jsp: useBean, jsp: setProperty and jsp: getProperty टैग का उपयोग बीन के विकास के लिए किया जाता है।

jsp:forward action tag

Jsp: फॉरवर्ड एक्शन टैग का उपयोग किसी अन्य रिसोर्सेज के रिक्वेस्ट को फॉरवर्ड करने के लिए किया जाता है, यह jsp, html या कोई अन्य रिसोर्सेज हो सकता है।

Jsp का सिंटैक्स : पैरामीटर के बिना आगे की एक्शन टैग

```
<jsp:forward page="relativeURL | <%= expression %>" />
```

Jsp का सिंटैक्स : पैरामीटर के साथ फॉरवर्ड एक्शन टैग

```
<jsp:forward page="relativeURL | <%= expression %>">
<jsp:param name="parametername" value="parametervalue | <%=expression%>" />
</jsp:forward>
```

Jsp का उदाहरण : बिना पैरामीटर के फॉरवर्ड एक्शन टैग

इस उदाहरण में, हम केवल printdate.jsp फ़ाइल के रिक्वेस्ट को फॉरवर्ड कर रहे हैं।

index.jsp

```
<html>
<body>
<h2>this is index page</h2>

<jsp:forward page="printdate.jsp" />
</body>
</html>
```

printdate.jsp

```
<html>
<body>
<% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
</body>
</html>
```

Jsp का उदाहरण : पैरामीटर के साथ आगे की एक्शन टैग

इस उदाहरण में, हम printdate.jsp फ़ाइल के लिए रिक्वेस्ट को पैरामीटर और printdate.jsp फ़ाइल के साथ फॉरवर्ड कर रहे हैं। दिनांक और समय के साथ पैरामीटर मान प्रिंट करता है।

index.jsp

```
<html>
<body>
<h2>this is index page</h2>

<jsp:forward page="printdate.jsp" >
<jsp:param name="name" value="ASIAN PUBLISHERS.com" />
</jsp:forward>


</body>
</html>
```

printdate.jsp

```
<html>
<body>

<% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
<%= request.getParameter("name") %>

</body>
</html>
```

jsp:include action tag

jsp:include action tag का उपयोग किसी अन्य रिसोर्सेज की कंटेंट को इन्कूड करने के लिए किया जाता है यह jsp, html या सर्वलेट हो सकता है।

Jsp में एक्शन टैग शामिल है रिक्वेस्ट समय पर रिसोर्सेज शामिल है इसलिए यह डायनेमिक पृष्ठों के लिए बेहतर है क्योंकि भविष्य में परिवर्तन हो सकते हैं।

Jsp: include टैग का उपयोग static और डायनेमिक पृष्ठों को शामिल करने के लिए किया जा सकता है।

jsp:include action tag पैरामीटर के बिना का सिंटैक्स

```
<jsp:include page="relativeURL | <%= expression %>" />
```

jsp:include action tag पैरामीटर के साथ का सिंटैक्स

```
<jsp:include page="relativeURL | <%= expression %>">
```

```
<jsp:param name="parametername" value="parametervalue | <%=expression%>" />
</jsp:include>
```

jsp:include action tag पैरामीटर के बिना का उदाहरण

इस उदाहरण में, index.jsp फ़ाइल में printdate.jsp फ़ाइल की कंटेंट शामिल है।

File: index.jsp

```
<h2>this is index page</h2>

<jsp:include page="printdate.jsp" />

<h2>end section of index page</h2>
```

File: printdate.jsp

```
<% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
```

JavaBean

एक जावाबीन एक जावा क्लास है जिसे निम्नलिखित conventions का पालन करना चाहिए—

- इसके पास एक no-args कंस्ट्रक्टर होना चाहिए।
- यह Serializable होना चाहिए।
- इसे गुणों के मूल्यों को प्राप्त करने और प्राप्त करने के तरीके प्रदान करने चाहिए, जिन्हें getter और setter मेथड के रूप में जाना जाता है।

JavaBean का उपयोग क्यों करें?

एक बीन कई ऑब्जेक्ट्स को एक ऑब्जेक्ट में एनकैप्सुलेट करता है ताकि हम इस ऑब्जेक्ट को कई जगहों से एक्सेस कर सकें। इसके अलावा, यह आसान मेन्टेन्स प्रदान करता है।

जावाबीन क्लास का सरल उदाहरण

```
//Employee.java

package mypack;
public class Employee implements java.io.Serializable{
private int id;
private String name;
public Employee(){}
public void setId(int id){this.id=id;}
public int getId(){return id;}
```

```
public void setName(String name){this.name=name;}
public String getName(){return name;}
}
```

जावाबीन क्लास का उपयोग कैसे करें?

जावाबीन क्लास तक पहुंचने के लिए, हमें getter और setter मेथड का उपयोग करना चाहिए।

```
package mypack;
public class Test{
public static void main(String args[]){
Employee e=new Employee();//object is created
e.setName("Arjun");//setting value to the object
System.out.println(e.getName());

}}
```

JavaBean प्रॉपर्टी

JavaBean प्रॉपर्टी एक नामित एट्रिब्यूट है जिसे ऑब्जेक्ट के यूजर द्वारा एक्सेस किया जा सकता है। यह सुविधा किसी भी जावा डेटा टाइप की हो सकती है, जिसमें आपके द्वारा परिभाषित क्लास शामिल हैं।

JavaBean के कार्यान्वयन क्लास में JavaBean सुविधाओं को दो तरीकों से एक्सेस किया जाता है:

1. getPropertyName ()

उदाहरण के लिए, यदि प्रॉपर्टी का नाम पहले नाम है, तो उस प्रॉपर्टी को पढ़ने के लिए मेथड का नाम getFirstName () होगा। इस मेथड को एक्सेसर कहा जाता है।

2. setPropertyName ()

उदाहरण के लिए, यदि प्रॉपर्टी का नाम पहले नाम है, तो उस प्रॉपर्टी को लिखने के लिए मेथड का नाम setFirstName () होगा। इस मेथड को mutator कहा जाता है।

जावाबीन के फायदे

JavaBean के निम्नलिखित फायदे हैं—

- JavaBean प्रॉपर्टी और मेथड को किसी अन्य एप्लीकेशन के लिए उजागर किया जा सकता है।
- यह सॉफ्टवेयर घटकों का पुनः उपयोग करने के लिए एक सहजता प्रदान करता है।

जावाबीन के नुकसान

जावाबीन के नुकसान निम्नलिखित हैं—

- JavaBeans परस्पर हैं। इसलिए, यह अपरिवर्तनीय ऑब्जेक्ट का लाभ नहीं उठा सकता है।
- प्रत्येक प्रॉपर्टी के लिए अलग से setter और getter मेथड बनाने से बॉयलरप्लेट कोड हो सकता है।

jsp:useBean action tag

Jsp: useBean एक्शन टैग का उपयोग सेम क्लास का पता लगाने या उसे instantiate करने के लिए किया जाता है। यदि बीन क्लास की बीन ऑब्जेक्ट पहले से ही बनाई गई है, तो यह स्कोप के आधार पर बीन नहीं बनाता है। लेकिन अगर बीन की ऑब्जेक्ट नहीं बनाई जाती है, तो यह बीन को तुरंत instantiate करती है।

jsp:useBean action tag का सिंटैक्सः

```
<jsp:useBean id= "instanceName" scope= "page | request | session | application"
class= "packageName.className" type= "packageName.className"
beanName="packageName.className | <%= expression >" >
</jsp:useBean>
```

प्रॉपर्टी और jsp:useBean action tag का उपयोग

1. **Id : निर्दिष्ट स्कोप में बीन की पहचान करने के लिए उपयोग किया जाता है।**
2. **स्कोप** : बीन के दायरे का प्रतिनिधित्व करता है। यह पेज, रिक्वेस्ट, सेशन या एपलीकेशन हो सकता है। डिफ़ॉल्ट स्कोप पेज है।
   - **पेज** : निर्दिष्ट करता है कि आप इस बीन का उपयोग JSP पेज के भीतर कर सकते हैं। डिफ़ॉल्ट स्कोप पेज है।
   - **रिक्वेस्ट** निर्दिष्ट करता है कि आप इस बीन का उपयोग किसी भी JSP पेज से कर सकते हैं जो समान रिक्वेस्ट को संसाधित करता है। इसमें पेज की तुलना में व्यापक गुंजाइश (wider scope) है।
   - **सेशन** : निर्दिष्ट करता है कि आप इस बीन का उपयोग किसी भी JSP पेज से एक ही सेशन में कर सकते हैं चाहे वह समान रिक्वेस्ट की प्रोसेस करता हो या नहीं। इसमें रिक्वेस्ट की तुलना में व्यापक गुंजाइश (wider scope) है।
   - **एप्लीकेशन** : निर्दिष्ट करता है कि आप इस बीन का उपयोग किसी भी JSP पेज से उसी एप्लीकेशन में कर सकते हैं। इसमें सेशन की तुलना में व्यापक गुंजाइश (wider scope) है।
3. **क्लास** : निर्दिष्ट बीन क्लास को तात्कालिक बनाता है (यानी बीन क्लास का एक ऑब्जेक्ट बनाता है) लेकिन इसमें कोई भी argument या कोई कंस्ट्रक्टर नहीं होना चाहिए ।
4. **टाइप** : यदि बीन पहले से ही दायरे में मौजूद है, तो बीन को एक डेटा टाइप प्रदान करता है। यह मुख्य रूप से क्लास या बीननाम एट्रिब्यूट के साथ प्रयोग किया जाता है।
5. **beanName** : java.beans.Beans.instantiate () मेथड का उपयोग करके बीन को तुरंत तैयार करता है।

उदाहरण—

इस उदाहरण में, हम केवल बीन क्लास की मेथड को लागू कर रहे हैं।

Calculator.java (एक साधारण बीन क्लास)

```
package com.ASIAN PUBLISHERS;
public class Calculator{
```

```
public int cube(int n){return n*n*n;}



}
```

index.jsp फ़ाइल

```
<jsp:useBean id="obj" class="com.ASIAN PUBLISHERS.Calculator"/>

<%
int m=obj.cube(5);
out.print("cube of 5 is "+m);

%>
```

jsp: setProperty और jsp: getProperty एक्शन टैग

जावा बीन के साथ वेब एप्लीकेशन विकसित करने के लिए सेटप्रॉपर्टी और गेटप्रॉपर्टी एक्शन टैग का उपयोग किया जाता है। वेब डेवेलोपमेंट में, बीन क्लास का उपयोग ज्यादातर किया जाता है क्योंकि यह एक पुनः प्रयोज्य सॉफ़्टवेयर घटक है जो डेटा का प्रतिनिधित्व करता है।

Jsp: setProperty एक्शन टैग सेटर मेथड का उपयोग करके बीन में एक प्रॉपर्टी मूल्य या मान सेट करता है।

jsp:setProperty action tag का सिंटैक्स :

```
<jsp:setProperty name="instanceOfBean" property= "*"  |
property="propertyName" param="parameterName" |
property="propertyName" value="{ string | <%= expression %>}"
/>
```

jsp:setProperty action tag का उदाहरणः यदि आप बीन में आने वाले रिक्वेस्ट के सभी मूल्यों को सेट करना चाहते हैं—

```
<jsp:setProperty name="bean" property="*" />
```

jsp:setProperty action tag का उदाहरणः यदि आपको आने वाली विशिष्ट प्रॉपर्टी का मूल्य निर्धारित करना है—

```
<jsp:setProperty name="bean" property="username" />
```

jsp:setProperty action tag का उदाहरणः एक्शन टैग यदि आपको प्रॉपर्टी में एक विशिष्ट मूल्य सेट करना है—

```
<jsp:setProperty name="bean" property="username" value="Asianpublishers" />
```

jsp: getProperty एक्शन टैग

Jsp: getProperty एक्शन टैग प्रॉपर्टी का मूल्य लौटाता है।

jsp:getProperty action tag का सिंटैक्स

```
<jsp:getProperty name="instanceOfBean" property="propertyName" />
```

jsp:getProperty action tag का सरल उदाहरण :

```
<jsp:getProperty name='obj' property='name' />
```

JSP में bean का उदाहरण

इस उदाहरण में 3 पेज हैं—

- index.html मानों के इनपुट के लिए।
- welocme.jsp फ़ाइल जो ऑब्जेक्ट के लिए आने वाले मान सेट करती है और एक मान प्रिंट करती है।
- User.java बीन क्लास जिसमें सेटर और गेट्टर मेथड हैं।

index.html

```
<form action="process.jsp" method="post">
Name:<input type="text" name="name"><br>
Password:<input type="password" name="password"><br>
Email:<input type="text" name="email"><br>
<input type="submit" value="register">
</form>
```

process.jsp

```
<jsp:useBean id="u" class="org.sssit.User"></jsp:useBean>
<jsp:setProperty property="*" name="u"/>

Record:<br>
<jsp:getProperty property="name" name="u"/><br>
<jsp:getProperty property="password" name="u"/><br>
<jsp:getProperty property="email" name="u" /><br>
```

User.java

```
package org.sssit;

public class User {
private String name,password,email;
//setters and getters
}
```

कई Jsp पृष्ठों में बीन का पुनः उपयोग

आइए सरल उदाहरण देखें, जो दो jsp पृष्ठों में बीन ऑब्जेक्ट के डेटा को प्रिंट करता है।

index.jsp

```
<jsp:useBean id="obj" class="com.ASIAN PUBLISHERS.Calculator"/>

<%
int m=obj.cube(5);
out.print("cube of 5 is "+m);
%>
```

User.java

```
package org.sssit;

public class User {
private String name,password,email;
//setters and getters
}
```

process.jsp

```
<jsp:useBean id="u" class="org.sssit.User" scope="session"></jsp:useBean>
<jsp:setProperty property="*" name="u"/>

Record:<br>
<jsp:getProperty property="name" name="u"/><br>
<jsp:getProperty property="password" name="u"/><br>
<jsp:getProperty property="email" name="u" /><br>

<a href="second.jsp">Visit Page</a>
```

second.jsp

```
<jsp:useBean id="u" class="org.sssit.User" scope="session"></jsp:useBean>
Record:<br>
<jsp:getProperty property="name" name="u"/><br>
<jsp:getProperty property="password" name="u"/><br>
<jsp:getProperty property="email" name="u" /><br>
```

SetProperty टैग में वैरिएबल मान का उपयोग करना

कुछ मामलों में, आपको डेटाबेस से कुछ मूल्य मिल सकता है, जो कि बीन ऑब्जेक्ट में सेट किया जाना है, ऐसे मामले में, आपको एक्सप्रेशन टैग का उपयोग करने की आवश्यकता है। उदाहरण के लिए:

process.jsp

```
<jsp:useBean id="u" class="org.sssit.User"></jsp:useBean>
<%
String name="arjun";
%>
<jsp:setProperty property="name" name="u" value="<%=name %>"/>

Record:<br>
<jsp:getProperty property="name" name="u"/><br>
```

JSP में एप्लेट प्रदर्शित करना (jsp:plugin action tag)

Jsp: प्लगइन एक्शन टैग का उपयोग रेज फाइल में एप्लेट को एम्बेड करने के लिए किया जाता है। Jsp: एप्लेट या बीन को Execute करने के लिए क्लाइंट साइड पर प्लागइन एक्शन टैग डाउनलोड प्लागइन रहता है ।

jsp:plugin action tag का सिंटैक्स

```
<jsp:plugin type= "applet | bean" code= "nameOfClassFile"
codebase= "directoryNameOfClassFile"
</jsp:plugin>
```

JSP में एप्लेट प्रदर्शित करने का उदाहरण

इस उदाहरण में, हम बस jsp: plugin टैग का उपयोग करके jsp में एप्लेट प्रदर्शित कर रहे हैं। आपके पास वर्तमान फ़ोल्डर में jsp फ़ाइल रहता है, जहां MouseDrag.class फ़ाइल (एक एप्लेट क्लास फ़ाइल) होनी चाहिए। आप बस इस प्रोग्राम को डाउनलोड कर सकते हैं जिसमें इस एप्लीकेशन को चलाने के लिए index.jsp, MouseDrag.java और MouseDrag.class फाइलें हैं।

index.jsp

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Mouse Drag</title>
  </head>
  <body bgcolor="khaki">
<h1>Mouse Drag Example</h1>
```

```
<jsp:plugin align="middle" height="500" width="500"
  type="applet"  code="MouseDrag.class" name="clock" codebase="."/>

  </body>
```

5.8 JSP-Implicit ऑब्जेक्ट्स

9 jsp implicit ऑब्जेक्ट्स हैं। ये ऑब्जेक्ट वेब कंटेनर द्वारा बनाए गए हैं जो सभी jsp पृष्ठों पर उपलब्ध हैं। उपलब्ध implicit ऑब्जेक्ट्स, रिक्वेस्ट, कॉन्फिगरेशन, सेशन, आवेदन आदि बाहर हैं।

9 implicit ऑब्जेक्ट की एक सूची नीचे दी गई है—

ऑब्जेक्ट	टाइप
Out	JspWriter
Request	HttpServletRequest
Response	HttpServletResponse
Config	ServletConfig
Application	ServletContext
Session	HttpSession
pageContext	PageContext
Page	Object
Exception	Throwable

(1) JSP out Implicit ऑब्जेक्ट

किसी भी डेटा को बफर में लिखने के लिए, JSP एक Implicit ऑब्जेक्ट प्रदान करता है जिसे नाम दिया गया है। यह JspWriter का **ऑब्जेक्ट** है। सर्वलेट के मामले में आपको लिखना होगा—

```
PrintWriter out=response.getWriter();
```

लेकिन JSP में, आपको यह कोड लिखने की आवश्यकता नहीं है।

out Implicit ऑब्जेक्ट का उदाहरण

इस उदाहरण में हम केवल दिनांक और समय प्रदर्शित कर रहे हैं।

index.jsp

```
<html>
<body>
<% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
</body>
</html>
```

(2) JSP request Implicit ऑब्जेक्ट

JSP request वेब कंटेनर द्वारा प्रत्येक jsp request के लिए बनाए गए, यह HttpServletRequest टाइप की एक Implicit ऑब्जेक्ट है। इसका उपयोग request प्राप्त करने के लिए किया जा सकता है जैसे पैरामीटर, हेडर जानकारी, रिमोट एड्रेस, सर्वर नाम, सर्वर पोर्ट, आदि।

यह jsp request फील्ड से एट्रिब्यूट को सेट करने, प्राप्त करने और हटाने के लिए भी उपयोग किया जा सकता है। आइए हम request ऑब्जेक्ट का सरल उदाहरण देखें जहां हम स्वागत संदेश के साथ यूजर का नाम प्रिंट कर रहे हैं।

JSP request implicit ऑब्जेक्ट का उदाहरण

index.html

```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

welcome.jsp

```
<%
String name=request.getParameter("uname");
out.print("welcome "+name);
%>
```

(3) JSP RESPONSE Implicit ऑब्जेक्ट

JSP में, RESPONSE HttpServletResponse टाइप की एक implicit ऑब्जेक्ट है। HttpServletResponse का उदाहरण प्रत्येक jsp request के लिए वेब कंटेनर द्वारा बनाया गया है।

इसका उपयोग RESPONSE को जोड़ने या manipualte करने के लिए किया जा सकता है जैसे कि किसी अन्य रिसोर्सेज पर पुनर्निर्देशित RESPONSE, त्रुटि भेजें आदि।

आइए, RESPONSE Implicit ऑब्जेक्ट का उदाहरण देखें जहां हम उददुत की RESPONSE को पुनर्निर्देशित कर रहे हैं।

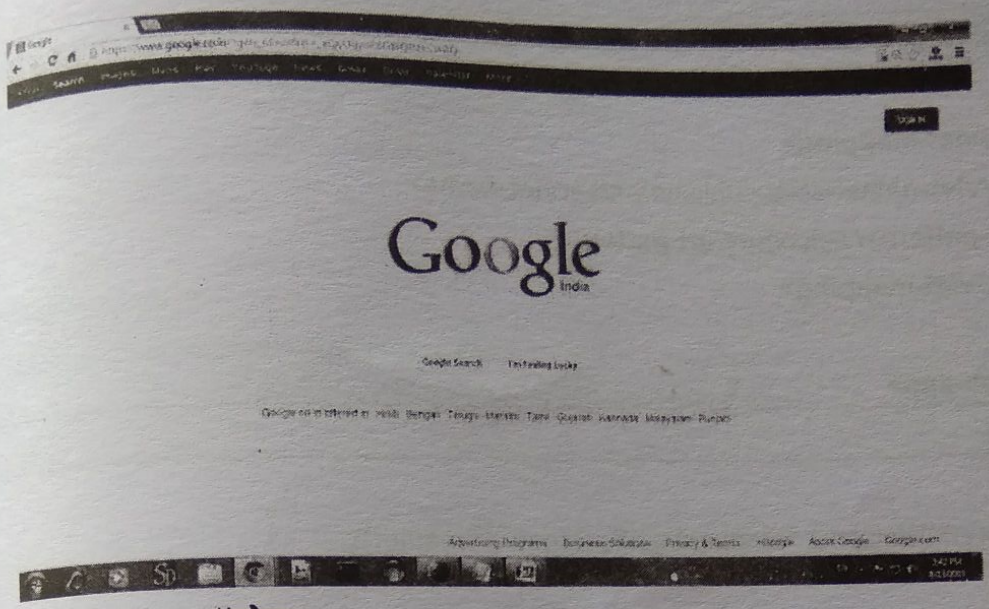
response implicit ऑब्जेक्ट का उदाहरण

index.html

```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

welcome.jsp

```
<%
response.sendRedirect("http://www.google.com");
%>
```



(4) JSP config implicit ऑब्जेक्ट

JSP में, config एक प्रकार की ServletConfig का implicit ऑब्जेक्ट है। इस ऑब्जेक्ट का उपयोग किसी विशेष JSP पेज के इनिशियलाइज़ेशन पैरामीटर प्राप्त करने के लिए किया जा सकता है। प्रत्येक jsp पेज के लिए वेब कंटेनर द्वारा कॉन्फिग ऑब्जेक्ट बनाया जाता है।

आमतौर पर, इसका उपयोग web.xml फ़ाइल से intialization पैरामीटर प्राप्त करने के लिए किया जाता है।

Config implicit ऑब्जेक्ट का उदाहरण:

index.html

```
<form action="welcome">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

web.xml file

```
<web-app>

<servlet>
<servlet-name>asianpublishers</servlet-name>
<jsp-file>/welcome.jsp</jsp-file>

<init-param>
<param-name>dname</param-name>
<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
</init-param>
```

```
</servlet>

<servlet-mapping>
<servlet-name>asianpublishers</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>

</web-app>
```

welcome.jsp

```
<%
out.print("Welcome "+request.getParameter("uname"));

String driver=config.getInitParameter("dname");
out.print("driver name is="+driver);
%>
```

(5) JSP एप्लीकेशन Implicit ऑब्जेक्ट

JSP में, एप्लीकेशन टाइप ServletContext की एक Implicit ऑब्जेक्ट है।

ServletContext का उदाहरण वेब कंटेनर द्वारा केवल एक बार बनाया जाता है जब एप्लीकेशन या प्रोजेक्ट सर्वर पर deploy किया जाता है।

इस इनिशियलाइज़ेशन पैरामीटर का उपयोग सभी jsp पृष्ठों द्वारा किया जा सकता है।

Application implicit ऑब्जेक्ट का उदाहरण:

index.html

```
<form action="welcome">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

web.xml file

```
<web-app>

<servlet>
<servlet-name>asianpublishers</servlet-name>
<jsp-file>/welcome.jsp</jsp-file>
</servlet>
```

```
<servlet-mapping>
<servlet-name>asianpublishers</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>

<context-param>
<param-name>dname</param-name>
<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
</context-param>

</web-app>
```

welcome.jsp

```
<%

out.print("Welcome "+request.getParameter("uname"));

String driver=application.getInitParameter("dname");
out.print("driver name is="+driver);

%>
```

(6) सेशन Implicit ऑब्जेक्ट

JSP में, सेशन HttpSession टाइप की एक Implicit ऑब्जेक्ट है। जावा डेवलपर इस ऑब्जेक्ट का उपयोग कर सेशन की जानकारी प्राप्त कर सकता है।

सेशन Implicit ऑब्जेक्ट का उदाहरण

index.html

```
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
</body>
```

```
</html>
```

welcome.jsp

```
<html>
<body>
<%

String name=request.getParameter("uname");
out.print("Welcome "+name);

session.setAttribute("user",name);

<a href="second.jsp">second jsp page</a>

%>
</body>
</html>
```

second.jsp

```
<html>
<body>
<%

String name=(String)session.getAttribute("user");
out.print("Hello "+name);

%>
</body>
</html>
```

(7) पेज कॉन्टेक्स्ट Implicit ऑब्जेक्ट

JSP में, पेज कॉन्टेक्स्ट, पेज कॉन्टेक्स्ट क्लास का एक Implicit ऑब्जेक्ट है।

JSP में, पेज स्कोप डिफ़ॉल्ट स्कोप है।

पेज कॉन्टेक्स्ट Implicit ऑब्जेक्ट का उदाहरण

index.html

```
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
</body>
</html>
```

welcome.jsp

```
<html>
<body>
<%

String name=request.getParameter("uname");
out.print("Welcome "+name);

pageContext.setAttribute("user",name,PageContext.SESSION_SCOPE);

<a href="second.jsp">second jsp page</a>

%>
</body>
</html>
```

second.jsp

```
<html>
<body>
<%

String name=(String)pageContext.getAttribute("user",PageContext.SESSION_SCOPE);
out.print("Hello "+name);
```

	यह हेडर If-Modified-Since का रिवर्स है; यह निर्दिष्ट करता है कि ऑपरेशन केवल तभी सफल होना चाहिए जब दस्तावेज़ निर्दिष्ट तिथि से पुराना हो।
12	**Referer** यह हेडर संदर्भित वेबपृष्ठों के URL को दर्शाता है। उदाहरण के लिए, यदि आप वेबपेज 1 पर हैं और वेबपेज 2 के एक लिंक पर क्लिक करते हैं, तो वेबपेज 1 का URL ब्राउजर के वेबपेज 2 का रिक्वेस्ट करने पर रेफर हेडर में शामिल किया जाता है।
13	**User-Agent** यह हेडर रिक्वेस्ट करने वाले ब्राउज़र या अन्य क्लाइंट की पहचान करता है और इसका उपयोग विभिन्न सामग्रियों को विभिन्न प्रकार के ब्राउज़रों में वापस करने के लिए किया जा सकता है।

HttpServletRequest ऑब्जेक्ट

रिक्वेस्ट ऑब्जेक्ट एक javax.servlet.http.HttpServletRequest ऑब्जेक्ट का एक उदाहरण है। जब भी कोई क्लाइंट किसी पेज का रिक्वेस्ट करता है, तो JSP इंजन उस रिक्वेस्ट का प्रतिनिधित्व करने के लिए एक नई ऑब्जेक्ट बनाता है।

रिक्वेस्ट ऑब्जेक्ट HTTP हेडर जानकारी प्राप्त करने के लिए तरीके प्रदान करता है जिसमें प्रपत्र डेटा, कुकीज़, HTTP तरीके आदि शामिल हैं।

निम्न तालिका उन महत्वपूर्ण मेथड को सूचीबद्ध करती है जिनका उपयोग आपके JSP प्रोग्राम में HTTP हेडर पढ़ने के लिए किया जा सकता है। ये तरीके HttpServletRequest ऑब्जेक्ट के साथ उपलब्ध हैं जो वेबसर्वर के लिए क्लाइंट रिक्वेस्ट का प्रतिनिधित्व करता है।

क्र.सं.	मेथड
1	**Cookie[] getCookies()**
2	**Enumeration getAttributeNames()**
3	**Enumeration getAttributeNames()**
4	**Enumeration getParameterNames()**
5	**HttpSession getSession ()**
6	**HttpSession getSession(boolean create)**
7	**Locale getLocale()**
8	**Object getAttribute(String name)**
9	**ServletInputStream getInputStream ()**
10	**String getAuthType()**
11	**String getCharacterEncoding()**
12	**String getContentType()**
13	**String getContextPath()**
14	**String getHeader(String name)**

15	**String getMethod()**
16	**String getParameter(String name)**
17	**String getPathInfo()**
18	**String getProtocol()**
19	**String getQueryString()**
20	**String getRemoteAddr().**
21	**String getRemoteHost()**
22	**String getRemoteUser()**
23	**String getRequestURI()**
24	**String getRequestedSessionId()**
25	**String getServletPath()**
26	**String[] getParameterValues(String name)**
27	**boolean isSecure()**
28	**int getContentLength()**
29	**int getIntHeader(String name)**
30	**int getServerPort()**

HTTP Header Request उदाहरण

निम्नलिखित उदाहरण है जो HTTP हेडर जानकारी को पढ़ने के लिए HttpServletRequest का getHeaderNames () मेथड का उपयोग करता है। यह मेथड एक एन्यूमरेशन देता है जिसमें वर्तमान HTTP रिक्वेस्ट से संबंधित हेडर जानकारी होती है।

एक बार हमारे पास एक एन्यूमरेशन है, हम स्टैंडर्ड तरीके से एन्यूमरेशन को कम कर सकते हैं। हम रोकने के लिए hasMoreElements () मेथड का उपयोग करेंगे और प्रत्येक पैरामीटर नाम का नाम प्राप्त करने के लिए nextElement () मेथड का उपयोग करेंगे।

```
<%@ page import = "java.io.*,java.util.*" %>

<html>
  <head>
    <title>HTTP Header Request Example</title>
  </head>

  <body>
    <center>
      <h2>HTTP Header Request Example</h2>

      <table width = "100%" border = "1" align = "center">
        <tr bgcolor = "#949494">
```

```
            <th>Header Name</th>
            <th>Header Value(s)</th>
          </tr>
          <%
            Enumeration headerNames = request.getHeaderNames();
            while(headerNames.hasMoreElements()) {
              String paramName = (String)headerNames.nextElement();
              out.print("<tr><td>" + paramName + "</td>\n");
              String paramValue = request.getHeader(paramName);
              out.println("<td> " + paramValue + "</td></tr>\n");
            }
          %>
        </table>
      </center>

    </body>
  </html>
```

आउटपुट:

HTTP Header Request Example

Header Name	Header Value(s)
user-agent	curl/7.35.0
host	127.0.0.1:8888
accept	*/*

5.10 JSP -सर्वर response (JSP-Server response)

जब कोई वेब सर्वर HTTP रिक्वेस्ट का जवाब देता है, तो प्रतिक्रिया में आमतौर पर एक स्टेटस लाइन, कुछ रिस्पॉन्स हेडर, एक खाली लाइन और डॉक्यूमेंट होते हैं। एक विशिष्ट response इस तरह दिखती है—

```
HTTP/1.1 200 OK
Content-Type: text/html
Header2: ...
...
HeaderN: ...
  (Blank Line)
<!doctype ...>

<html>
  <head>...</head>
  <body>
```

```
...
</body>
</html>
```

स्टेटस लाइन में HTTP वर्जन (उदाहरण में HTTP / 1.1), एक स्टेटस कोड (उदाहरण में 200), और स्टेटस कोड (उदाहरण में ओके) के अनुरूप एक बहुत छोटा संदेश होता है।

निम्नलिखित सबसे उपयोगी HTTP 1.1 प्रतिक्रिया हेडर का सारांश है जो वेब सर्वर से ब्राउज़र पर वापस जाता है। ये हेडर अक्सर वेब प्रोग्रामिंग में उपयोग किए जाते हैं—

क्र.सं.	हेडर और स्टेटमेंट
1	**Allow** यह हेडर request मेथड (GET, POST, आदि) को निर्दिष्ट करता है जो सर्वर समर्थन करता है।
2	**Cache-Control** यह हेडर उन परिस्थितियों को निर्दिष्ट करता है जिसमें response document सुरक्षित रूप से cache किया जा सकता है। इसमें **public, private** या नो-**no-cache** मूल्य हो सकते हैं।
3	**Connection** यह हेडर ब्राउज़र को डायरेक्टिव देता है कि वह लगातार HTTP कनेक्शन का उपयोग करे या नहीं।
4	**Content-Disposition** यह हेडर आपको रिक्वेस्ट करता है कि ब्राउज़र यूजर को दिए गए नाम की फ़ाइल में डिस्क की प्रतिक्रिया को सेव करे।
5	**Content-Encoding** यह हेडर उस तरीके को निर्दिष्ट करता है जिसमें ट्रांसमिशन के दौरान पेज एन्कोड किया गया था।
6	**Content-Language** यह हेडर उस भाषा को दर्शाता है जिसमें दस्तावेज़ लिखा गया है। उदाहरण के लिए, **en, en-us, ru,** आदि।
7	**Content-Length** यह हेडर प्रतिक्रिया में बाइट्स की संख्या को इंगित करता है। यह जानकारी केवल तभी आवश्यक है जब ब्राउज़र एक निरंतर (keep-alive) HTTP कनेक्शन का उपयोग कर रहा है।
8	**Content-Type** यह हेडर प्रतिक्रिया दस्तावेज़ के MIME **(Multipurpose Internet Mail Extension)** टाइप देता है।
9	**Expires** यह हेडर उस समय को निर्दिष्ट करता है जिस पर कंटेंट को पुराना माना जाना चाहिए।
10	**Last-Modified** यह हेडर इंगित करता है कि दस्तावेज़ अंतिम बार कब बदला गया था।
11	**Location** इस हैडर को उन सभी प्रतिक्रियाओं के साथ शामिल किया जाना चाहिए जिनके पास 300 में एक status कोड है।

12	**Refresh** यह हेडर निर्दिष्ट करता है कि ब्राउज़र को अपडेट पेज के लिए कितनी जल्दी पूछना चाहिए। आप कुछ सेकंड में समय निर्दिष्ट कर सकते हैं जिसके बाद पेज refresh किया जाएगा।
13	**Retry-After** इस हेडर का उपयोग क्लाइंट को बताने के लिए 503 (सेवा अनुपलब्ध) प्रतिक्रिया के साथ संयोजन के रूप में किया जा सकता है कि वह कितनी जल्दी अपने रिक्वेस्ट को दोहरा सकता है।
14	**Set-Cookie** यह header पेज से संबद्ध कुकी निर्दिष्ट करता है।

HttpServletResponse ऑब्जेक्ट

Response ऑब्जेक्ट एक javax.servlet.http.HttpServletResponse ऑब्जेक्ट का एक उदाहरण है। जिस तरह सर्वर रिक्वेस्ट ऑब्जेक्ट बनाता है, उसी तरह यह क्लाइंट के रिस्पॉन्स को दर्शाने के लिए ऑब्जेक्ट भी बनाता है।

प्रतिक्रिया ऑब्जेक्ट नए HTTP हेडर बनाने के साथ सौदा करने वाले इंटरफेस को भी परिभाषित करता है। इस ऑब्जेक्ट के माध्यम से, JSP प्रोग्रामर नए कुकीज़ या दिनांक टिकट, HTTP स्थिति कोड आदि जोड़ सकता है।

आपके सर्वलेट प्रोग्राम में HTTP प्रतिक्रिया हेडर सेट करने के लिए निम्न मेथड का उपयोग किया जा सकता है। ये तरीके HttpServletResponse ऑब्जेक्ट के साथ उपलब्ध हैं। यह ऑब्जेक्ट सर्वर प्रतिक्रिया का प्रतिनिधित्व करता है।

क्र.सं.	मेथड और स्टेटमेंट
1	**String encodeRedirectURL(String url)** SendRedirect मेथड में उपयोग के लिए निर्दिष्ट URL को एन्कोड करता है या, यदि एन्कोडिंग की आवश्यकता नहीं है, तो URL को अपरिवर्तित लौटाता है।
2	**String encodeURL(String url)** इसमें सेशन आईडी शामिल करके निर्दिष्ट URL को एनकोड करता है, या, यदि एन्कोडिंग की आवश्यकता नहीं है, तो URL अपरिवर्तित लौटाता है।
3	**boolean containsHeader(String name)** एक बूलियन को इंगित करता है कि नामांकित हेडर पहले ही सेट किया गया है।
4	**boolean isCommitted()** यदि प्रतिक्रिया हुई है तो एक बुलियन लौटता है।
5	**void addCookie(Cookie cookie)** निर्दिष्ट कुकी को प्रतिक्रिया में जोड़ता है।
6	**void addDateHeader(String name, long date)** दिए गए नाम और दिनांक-मान के साथ एक प्रतिक्रिया हेडर जोड़ता है।

7	**void addHeader(String name, String value)** दिए गए नाम और मान के साथ एक प्रतिक्रिया हेडर जोड़ता है।
8	**void addIntHeader(String name, int value)** दिए गए नाम और पूर्णांक मान के साथ एक प्रतिक्रिया हेडर जोड़ता है।
9	**void flushBuffer()** क्लाइंट को लिखी जाने वाली बफर की कंटेंट को मजबूर करता है।
10	**void reset()** बफर में मौजूद किसी भी डेटा को क्लियर करता है।
11	**void resetBuffer()** हेडर या स्थिति कोड को साफ़ किए बिना प्रतिक्रिया में Implicit बफर की कंटेंट को साफ करता है।
12	**void sendError(int sc)** निर्दिष्ट स्थिति कोड का उपयोग करके क्लाइंट को एक त्रुटि प्रतिक्रिया भेजता है और बफर को साफ़ करता है।
13	**void sendError(int sc, String msg)** निर्दिष्ट स्थिति का उपयोग करके क्लाइंट को एक त्रुटि प्रतिक्रिया भेजता है।
14	**void sendRedirect(String location** URL का उपयोग करके क्लाइंट को एक अस्थायी पुनर्निर्देशित प्रतिक्रिया भेजता है।
15	**void setBufferSize(int size)** प्रतिक्रिया के body के लिए पसंदीदा बफर आकार सेट करता है।
16	**void setCharacterEncoding(String charset)** क्लाइंट को भेजे जा रहे रिस्पॉन्स के कैरेक्टर एन्कोडिंग (MIME) को सेट करता है, उदाहरण के लिए, UTF-8 को।
17	**void setContentLength(int len)** प्रतिक्रिया में कंटेंट body की लंबाई सेट करता है HTTP सर्वलेट्स में; यह तरीका प्ऊऊइ कंटेंट-लेंथ हेडर भी सेट करता है।
18	**void setContentType(String type)** क्लाइंट को भेजी जा रही प्रतिक्रिया का कंटेंट टाइप सेट करता है, अगर प्रतिक्रिया अभी तक प्रतिबद्ध नहीं हुई है।

19	**void setDateHeader(String name, long date)** दिए गए नाम और दिनांक-मान के साथ एक प्रतिक्रिया हेडर सेट करता है।
20	**void setHeader(String name, String value)** दिए गए नाम और मान के साथ एक प्रतिक्रिया हेडर सेट करता है।
21	**void setIntHeader(String name, int value)** दिए गए नाम और पूर्णांक मान के साथ एक प्रतिक्रिया हेडर सेट करता है।
22	**void setLocale(Locale loc)** प्रतिक्रिया का स्थान निर्धारित करता है, अगर प्रतिक्रिया अभी तक प्रतिबद्ध नहीं हुई है।
23	**void setStatus(int sc)** इस प्रतिक्रिया के लिए स्थिति कोड सेट करता है।

HTTP हैडर रिस्पांस उदाहरण

एक डिजिटल घड़ी का अनुकरण करने के लिए रीफ्रेश हेडर सेट करने के लिए निम्न उदाहरण setIntHeader () मेथड का उपयोग करेंगे—

```
<%@ page import = "java.io.*,java.util.*" %>

<html>

  <head>
    <title>Auto Refresh Header Example</title>
  </head>

  <body>
    <center>
      <h2>Auto Refresh Header Example</h2>
      <%
        // Set refresh, autoload time as 5 seconds
        response.setIntHeader("Refresh", 5);

        // Get current time
        Calendar calendar = new GregorianCalendar();

        String am_pm;
        int hour = calendar.get(Calendar.HOUR);
        int minute = calendar.get(Calendar.MINUTE);
        int second = calendar.get(Calendar.SECOND);

        if(calendar.get(Calendar.AM_PM) == 0)
          am_pm = "AM";
        else
```

```
        am_pm = "PM";
        String CT = hour+":"+ minute +":"+ second +" "+ am_pm;
        out.println("Current Time is: " + CT + "\n");
    %>
  </center>

 </body>
</html>
```

अब उपरोक्त कोड main.jsp में डालें और इसे एक्सेस करने का प्रयास करें। यह वर्तमान सिस्टम समय को हर 5 सेकंड के बाद निम्नानुसार प्रदर्शित करेगा। JSP चलों। आपको निम्न आउटपुट प्राप्त होंगे—

Auto Refresh Header Example

Current Time is: 2:10:44 PM

5.11 JSP integration डेटाबेस के साथ (5.11 JSP integration with database)

डेटाबेस का उपयोग विभिन्न प्रकार के डेटा को संग्रहीत करने के लिए किया जाता है जो विशाल हैं और गीगाबाइट में भंडारण क्षमता है। JSP रिकॉर्ड बनाने और प्रबंधित करने के लिए ऐसे डेटाबेस से जुड़ सकता है।

बुनियादी अवधारणा से शुरू करने के लिए, आइए हम एक तालिका बनाते हैं और उस तालिका में कुछ रिकॉर्ड बनाते हैं—

तालिका बनों

EMP डेटाबेस में **Employees** तालिका बनाने के लिए, निम्नलिखित स्टेप का उपयोग करें—

स्टेप 1

एक कमांड प्रॉम्प्ट खोलें और निम्नानुसार इंस्टॉलेशन डायरेक्टरी में बदलें—

```
C:\>
C:\>cd Program Files\MySQL\bin
C:\Program Files\MySQL\bin>
```

स्टेप 2

डेटाबेस में निम्नानुसार लॉगिन करें—

```
C:\Program Files\MySQL\bin>mysql -u root -p
Enter password: ********
mysql>
>
```

स्टेप 3

TEST डेटाबेस में **Employee** तालिका इस प्रकार बनों—

```
mysql> use TEST;
mysql> create table Employees
  (
```

```
    id int not null,
    age int not null,
    first varchar (255),
    last varchar (255)
  );
Query OK, 0 rows affected (0.08 sec)
mysql>
```

डेटा रिकॉर्ड बनों

आइए अब हम Employee तालिका में कुछ रिकॉर्ड बनाते हैं—

```
mysql> INSERT INTO Employees VALUES (100, 18, 'Ram', 'Singh');
Query OK, 1 row affected (0.05 sec)

mysql> INSERT INTO Employees VALUES (101, 25, 'Mohan', 'Fatma');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Employees VALUES (102, 30, 'Vidit', 'Khan');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Employees VALUES (103, 28, 'Sumit', 'Mittal');
Query OK, 1 row affected (0.00 sec)

mysql>
```

SELECT ऑपरेशन

निम्नलिखित उदाहरण से पता चलता है कि हम JSP प्रोग्रामिंग में JTSL का उपयोग करके SQL SELECT स्टेटमेंट को कैसे Execute कर सकते हैं—

```
<%@ page import = "java.io.*,java.util.*,java.sql.*"%>
<%@ page import = "javax.servlet.http.*,javax.servlet.*" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix = "c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix = "sql"%>

<html>
  <head>
    <title>SELECT Operation</title>
  </head>

  <body>
    <sql:setDataSource var = "snapshot" driver = "com.mysql.jdbc.Driver"
      url = "jdbc:mysql://localhost/TEST"
      user = "root"  password = "pass123"/>

    <sql:query dataSource = "${snapshot}" var = "result">
      SELECT * from Employees;
    </sql:query>

    <table border = "1" width = "100%">
```

```
    <tr>
      <th>Emp ID</th>
      <th>First Name</th>
      <th>Last Name</th>
      <th>Age</th>
    </tr>

    <c:forEach var = "row" items = "${result.rows}">
      <tr>
        <td><c:out value = "${row.id}"/></td>
        <td><c:out value = "${row.first}"/></td>
        <td><c:out value = "${row.last}"/></td>
        <td><c:out value = "${row.age}"/></td>
      </tr>
    </c:forEach>
  </table>

 </body>
</html>
```

उपरोक्त JSP पर पहुँचें, निम्न परिणाम प्रदर्शित किया जाएगा—

Emp ID	First Name	Last Name	Age
100	Ram	Singh	18
101	Mohan	Fatma	25
102	Vidit	Khan	30
103	Sumit	Mittal	28

INSERT ऑपरेशन

निम्नलिखित उदाहरण से पता चलता है कि हम JSP प्रोग्रामिंग में JTSL का उपयोग करते हुए SQL INSERT स्टेटमेंट को कैसे Execute कर सकते हैं—

```
<%@ page import = "java.io.*,java.util.*,java.sql.*"%>
<%@ page import = "javax.servlet.http.*,javax.servlet.*" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix = "c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix = "sql"%>

<html>
 <head>
  <title>JINSERT Operation</title>
 </head>

 <body>
  <sql:setDataSource var = "snapshot" driver = "com.mysql.jdbc.Driver"
    url = "jdbc:mysql://localhost/TEST"
    user = "root"  password = "pass123"/>
    <sql:update dataSource = "${snapshot}" var = "result">
    INSERT INTO Employees VALUES (104, 2, 'Nuha', 'Singh');
```

```
    </sql:update>

    <sql:query dataSource = "${snapshot}" var = "result">
      SELECT * from Employees;
    </sql:query>


    <table border = "1" width = "100%">
      <tr>
        <th>Emp ID</th>
        <th>First Name</th>
        <th>Last Name</th>
        <th>Age</th>
      </tr>

      <c:forEach var = "row" items = "${result.rows}">
        <tr>
          <td><c:out value = "${row.id}"/></td>
          <td><c:out value = "${row.first}"/></td>
          <td><c:out value = "${row.last}"/></td>
          <td><c:out value = "${row.age}"/></td>
        </tr>
      </c:forEach>
    </table>


  </body>
</html>
```

उपरोक्त JSP पर पहुँचें, निम्न परिणाम प्रदर्शित किया जाएगा—

Emp ID	First Name	Last Name	Age
100	Ram	Singh	18
101	Mohan	Fatma	25
102	Vidit	Khan	30
103	Sumit	Mittal	28
104	Nuha	Singh	2

DELETE ऑपरेशन

निम्नलिखित उदाहरण से पता चलता है कि हम JSP प्रोग्रामिंग में JTSL का उपयोग करके SQL DELETE स्टेटमेंट को कैसे Execute कर सकते हैं—

```
<%@ page import = "java.io.*,java.utjl.*,java.sql.*"%>
<%@ page import = "javax.servlet.http.*,javax.servlet.*" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix = "c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix = "sql"%>
```

```
<html>
 <head>
  <title>DELETE Operation</title>
 </head>

 <body>
  <sql:setDataSource var = "snapshot" driver = "com.mysql.jdbc.Driver"
    url = "jdbc:mysql://localhost/TEST"
    user = "root" password = "pass123"/>


  <c:set var = "empId" value = "103"/>


  <sql:update dataSource = "${snapshot}" var = "count">
    DELETE FROM Employees WHERE Id = ?
    <sql:param value = "${empId}" />
  </sql:update>

  <sql:query dataSource = "${snapshot}" var = "result">
    SELECT * from Employees;
  </sql:query>

  <table border = "1" width = "100%">
    <tr>
      <th>Emp ID</th>
      <th>First Name</th>
      <th>Last Name</th>
      <th>Age</th>
    </tr>

    <c:forEach var = "row" items = "${result.rows}">
      <tr>
        <td><c:out value = "${row.id}"/></td>
        <td><c:out value = "${row.first}"/></td>
        <td><c:out value = "${row.last}"/></td>
        <td><c:out value = "${row.age}"/></td>
      </tr>
    </c:forEach>
  </table>


 </body>
</html>
```

उपरोक्त JSP पर पहुँचें, निम्न परिणाम प्रदर्शित किया जाएगा—

Emp ID	First Name	Last Name	Age
100	Ram	Singh	18
101	Mohan	Fatma	25
102	Vidit	Khan	30

UPDATE ऑपरेशन

निम्नलिखित उदाहरण से पता चलता है कि हम JSP प्रोग्रामिंग में JTSL का उपयोग करके SQL UPDATE स्टेटमेंट को कैसे Execute कर सकते हैं—

```
<%@ page import = "java.io.*,java.util.*,java.sql.*"%>
<%@ page import = "javax.servlet.http.*,javax.servlet.*" %>
<%@ taglib uri = "http://java.sun.com/jsp/jstl/core" prefix = "c"%>
<%@ taglib uri = "http://java.sun.com/jsp/jstl/sql" prefix = "sql"%>

<html>
  <head>
    <title>DELETE Operation</title>
  </head>

  <body>
    <sql:setDataSource var = "snapshot" driver = "com.mysql.jdbc.Driver"
      url = "jdbc:mysql://localhost/TEST"
      user = "root" password = "pass123"/>

    <c:set var = "empId" value = "102"/>

    <sql:update dataSource = "${snapshot}" var = "count">
      UPDATE Employees SET WHERE last = 'Singh'
      <sql:param value = "${empId}" />
    </sql:update>

    <sql:query dataSource = "${snapshot}" var = "result">
      SELECT * from Employees;
    </sql:query>

    <table border = "1" width = "100%">
      <tr>
        <th>Emp ID</th>
        <th>First Name</th>
        <th>Last Name</th>
        <th>Age</th>
      </tr>

      <c:forEach var = "row" items = "${result.rows}">
        <tr>
          <td><c:out value = "${row.id}"/></td>
          <td><c:out value = "${row.first}"/></td>
```

```
        <td><c:out value = "${row.last}"/></td>
        <td><c:out value = "${row.age}"/></td>
      </tr>
    </c:forEach>
  </table>

  </body>
</html>
```

उपरोक्त JSP पर पहुँचें, निम्न परिणाम प्रदर्शित किया जाएगा—

EmpID	First Name	Last Name	Age
100	Ram	Singh	18
101	Mohan	Fatma	25
102	Vidit	Singh	30

5.12 JSP सेशन

JSP में सेशन को एक Implicit ऑब्जेक्ट के रूप में परिभाषित किया गया है जो अपने सर्वलेट के मूल में HttpSession इंटरफ़ेस को लागू करता है। सेशन एकल इंटरैक्टिव यूजर निर्दिष्ट करता है। एक Implicit ऑब्जेक्ट सेशन, क्लाइंट विशिष्ट बातचीत को निर्दिष्ट करता है। Implicit ऑब्जेक्ट सेशन का उपयोग किसी एकल यूजर के लिए एक सेशन बनाने के लिए किया जाता है जो निर्दिष्ट करता है कि यूजर निर्मित सेशन के भीतर सक्रिय या निष्क्रिय है या नहीं। सेशन ऑब्जेक्ट का उपयोग करके आप इंटरफ़ेस HttpSession में घोषित इसके विभिन्न तरीकों को लागू कर सकते हैं।

आम तौर पर उपयोग किए जाने वाले कुछ तरीके जो एक सेशन ऑब्जेक्ट को लागू कर सकते हैं वे इस प्रकार हैं—

- **getAttribute(String name) :** इस मेथड का उपयोग उस एट्रिब्यूट मान को प्राप्त करने के लिए किया जाता है, जो इस सेशन के भीतर setAttribute() मेथड का उपयोग करके सेट किया गया है।
- **getAttributeNames () :** इस मेथड का उपयोग उन सभी एट्रिब्यूट के नाम प्राप्त करने के लिए किया जाता है, जो setAttribute() मेथड का उपयोग करके निर्धारित किए जाते हैं।
- **getCreationTime ():** इस मेथड का उपयोग सेशन का समय प्राप्त करने के लिए किया जाता है जब इसे बनाया गया था।
- **getId () :** इस मेथड का उपयोग सेशन की अनूठी आईडी प्राप्त करने के लिए किया जाता है।
- **getLastAccessedTime () :** इस मेथड का उपयोग इस सेशन के साथ यूजर के अंतिम समय तक पहुंचने के लिए किया जाता है।
- **getMaxInactiveInterval () :** इस मेथड का उपयोग सेकंड में उस समय को प्राप्त करने के लिए किया जाता है जो इस सेशन के साथ यूजर के अधिकतम समय अंतराल को निर्दिष्ट करता है जिसे एक्सेस करने के लिए एक सर्वलेट कंटेनर ओपन करेगा ।

- **setMaxInactiveInterval(int interval) :** इस मेथड का उपयोग सेकंड में समय निर्धारित करने के लिए किया जाता है जो इस सेशन के साथ यूजर के अधिकतम समय अंतराल को निर्दिष्ट करता है जो सर्वलेट कंटेनर एक्सेस करने के लिए सेट होगा।
- **setAttribute(String name, Object value) :** इस सेशन के भीतर यूजर के लिए एट्रिब्यूट मान सेट करने के लिए इस मेथड का उपयोग किया जाता है।
- **isNew() :** इस मेथड का उपयोग यह जांचने के लिए किया जाता है कि निर्दिष्ट सेशन नया है या नहीं।
- **invalidate() :** इस सेशन को समाप्त करने के लिए इस मैथड का उपयोग किया जाता है।
- **removeAttribute(String name) :** इस मेथड का उपयोग इस सेशन के लिए निर्दिष्ट एट्रिब्यूट के नाम के साथ जुड़ी हुई ऑब्जेक्ट को हटाने के लिए किया जाता है।

उदाहरण

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Session In JSP</title>
</head>
<body>
<%
request.getSession(true);
out.println("Is Session New : "+session.isNew());
out.println("Session Creation Time : "+session.getCreationTime());
Object name = "asian";
Object title = "publishers";
session.setAttribute("name", name);
session.setAttribute("title", title);
out.println("Name = "+ session.getAttribute("name"));
out.println("Title = "+ session.getAttribute("title"));
java.util.Enumeration e = session.getAttributeNames();
out.println("Attributes Are : ");
while(e.hasMoreElements())
```

```
out.println(e.nextElement());
}
session.invalidate();
%>
<body>
<html>
```

आउटपुट:

```
Is Session New : true Session Creation Time : 1603377385447 Name = asian Title = publishers
Attributes Are : title name
```

5.13 Jsp में exception हैंडलिंग

जब JSP के rules या syntax के खिलाफ अगर program को लिखा जाता है तब program पर exception आने की संभावना होती है।

अगर पूरे JSP Page में कहा पर भी exception हो तो इसकी वजह से बिना exception वाला म्द्ा भी execute नहीं हो पाता है।

For Example

```
<%
out.print("Hello World");
out.print(4/0);
%>
```

ऊपर दिए गए example पर पहला statement तो ठीक है लेकिन दूसरे statement में कुछ exception होने की वजह से पहला statement भी execute नहीं हो पाता है। यह example को try-Catch block पर पढ़ेंगे।

Exception : Exceptions keâes runtime पर handle किया जा सकता है। exception object को runtime hej throw किया जाता है। अगर user द्वारा कुछ गलत information को provide किया जाता है या network या database से connect नहीं होता है तो exception occur होता है। लेकिन इन exceptions को handle किया जा सकता है।

Error : Errors को handle नहीं किया जा सकता है। Errors बहुत ही critical होता है। Program अगर out of memory जाता है। अगर कहा पर syntax error होता है या system में या web server पर कुछ खराबी होती है तो Error आ जाता है। इन Errors को ठीक करना काफी कठिन काम होता है।

JSP Page को दो प्रकार से Handle किया जा सकता है।

1. Page-Level Exception Handling
2. Try-Catch Block Exception Handling

1. Page-Level Exception Handling in JSP

Page-Level Exception में एक single error page को create किया जाता है। अगर किसी JSP page में कुछ exception होता है तब बिना url change ngS create किये गए errorpage को execute किया जाता है।

उदाहरण

index.html

```
<form action="print.jsp">
Enter Name : <input type="text" name="myname"><br />
Enter Age : <input type="text" name="myage"><br />
<input type="submit" value="Submit">
```

सबसे पहले अगर Page-Level पर exception को handle करना हो तो handle किये जानेवाले code में page directive का errorPage Ues attribute के साथ value के रूप में exception handle करनेवाला code दिया जाता है।

```
<%@ page errorPage="exception.jsp" %>
<%
out.print(request.getParameter("myname"));
out.print(Integer.parseInt(request.getParameter("myage")));
%>
```

exception.jsp

जब exception handle करनेवाला code लिखा जाता है तब उस code में page directive का isErrorPage ये attribute और उसकी value को true देना जरुरी होता है।

```
<%@ page isErrorPage="true" %>
<% out.print("Exception : " + exception); %>
```

2. Try-Catch Block Exception Handling in JSP

Try-Catch Block में single page पर exception को handle किया जाता है।

Syntax :

```
try{
    try_block_code;
}catch{
    exception_handling_code;
}
```

आउटपुटः

```
<%
out.println("Hello World");

try{
out.println(4/0);
}
```

```
catch (Exception e){
    out.println("Exception : " + e);

}
%>
```

आउटपुट:

```
Hello World

Exception : java.lang.ArithmeticException: / by zero
```

5.14 JSTL (JSP स्टैंडर्ड टैग लाइब्रेरी)

JSP Standard Tag Library (JSTL) JSP विकास को आसान बनाने के लिए टैग्स के एक सेट का प्रतिनिधित्व करता है।

JSTL का लाभ

1. **तेजी से विकास :** JSTL कई टैग प्रदान करता है जो JSP को सरल बनाते हैं।
2. **कोड पुनः प्रयोज्य :** हम विभिन्न पृष्ठों पर JSTL टैग का उपयोग कर सकते हैं।
3. स्क्रिप्टलेट टैग का उपयोग करने की आवश्यकता नहीं है।

JSTL टैग

वहाँ JSTL मुख्य रूप से पाँच प्रकार के टैग प्रदान करता है—

टैग नाम	स्टेटमेंट
कोर टैग	JSTL कोर टैग वैरिएबल सपोर्ट, URL मैनेजमेंट, फ्लो कंट्रोल आदि प्रदान करता है। कोर टैग का URL http://java.sun.com/jsp/jstl/core है। कोर टैग का उपसर्ग c है।
फंक्शन टैग	फ़ंक्शन टैग स्ट्रिंग manipualte और स्ट्रिंग लंबाई के लिए समर्थन प्रदान करते हैं। फ़ंक्शन टैग के लिए URL http://java.sun.com/jsp/jstl/functions और उपसर्ग fn है।
Formatting टैग	Formatting टैग संदेश Formatting, संख्या और दिनांक Formatting आदि के लिए समर्थन प्रदान करते हैं। स्वरूपण टैग के लिए URL http://java.sun.com/jsp/jstl/fmt और उपसर्ग fmt है।
XML टैग	एक्सएमएल टैग प्रवाह नियंत्रण, परिवर्तन, आदि प्रदान करते हैं। एक्सएमएल टैग के लिए URL http://java.sun.com/jsp/jstl/xml है और उपसर्ग x है।
एसक्यूएल टैग	JSTL SQL टैग SQL सपोर्ट प्रदान करते हैं। SQL टैग का URL http://java.sun.com/jsp/jstl/sql और उपसर्ग sql है।

6

AJAX

LEARNING OBJECTIVES :

After reading this chapter, you would be able to understand :

- परिचय (Introduction)
- XMLHttp रिक्वेस्ट ऑब्जेक्ट (XMLHttp Request object)
- XMLHttp रिक्वेस्ट ऑब्जेक्ट (XMLHttp Request object)
- AJAX कैसे काम करता है?
- प्राइमफेस AJAX
- AJAX events
- Ajax Listener
- AJAX एक्शनलिस्ट
- Ajax Validation
- jQuerry और AJAX(jQuerry and AJAX)

6.1 परिचय (Introduction)

AJAX Asynchronous JavaScript and XML के लिए एक संक्षिप्त है। यह इंटर-रिलेटेड तकनीकों का एक समूह है जैसे जावास्क्रिप्ट, डोम,एक्सएमएल,एचटीएमएल/एक्सएचटीएमएल,सीएसएस,XMLHttpRequestआदि।

AJAX आपको वेब पेज पुनः लोड किए बिना डेटा भेजने और प्राप्त करने की अनुमति देता है। इसलिए यह तेज है।

AJAX
Asynchronous Javascript And XML

AJAX आपको केवल महत्वपूर्ण जानकारी सर्वर पर भेजने की अनुमति देता है न कि पूरे पृष्ठ को। इसलिए क्लाइंट की ओर से केवल मूल्यवान डेटा को सर्वर साइड में रूट किया जाता है। यह आपके एप्लीकेशन को इंटरैक्टिव और तेज़ बनाता है।

इसका उपयोग कहां किया जाता है?

वेब पर बहुत सारे वेब एप्लीकेशन चल रहे हैं जो ajax तकनीक का उपयोग कर रहे हैं जैसे gmail, facebook, twitter, google map, youtube आदि।

6.2 XMLHttp रिक्वेस्ट ऑब्जेक्ट (XMLHttp Request object)

सभी आधुनिक ब्राउज़र XMLHttpRequest ऑब्जेक्ट का समर्थन करते हैं।

XMLHttpRequest ऑब्जेक्ट का उपयोग पर्दे के पीछे सर्वर के साथ डेटा का आदान-प्रदान करने के लिए किया जा सकता है। इसका मतलब है कि पूरे पृष्ठ को फिर से लोड किए बिना, वेब पेज के कुछ हिस्सों को अपडेट करना संभव है

XMLHttpRequest का एक ऑब्जेक्ट क्लाइंट और सर्वर के बीच अतुल्यकालिक संचार के लिए उपयोग किया जाता है।

यह निम्नलिखित कार्य करता है—

1. क्लाइंट से बैकग्राउंड में डेटा भेजता है।
2. सर्वर से डेटा प्राप्त करता है।
3. वेब पेज को बिना पुनः लोड किए अपडेट करता है।

XMLHttpRequest ऑब्जेक्ट के प्रॉपर्टी

XMLHttpRequest ऑब्जेक्ट के सामान्य प्रॉपर्टी निम्नानुसार हैं—

प्रॉपर्टी	विवरण
onReadyStateChange	जब भी रीडिस्टेट एट्रिब्यूट बलदता है इसे कहा जाता है। इसे सिंक्रोनस रिक्वेस्ट के साथ उपयोग नहीं किया जाना चाहिए।
readyState	रिक्वेस्ट की स्थिति का प्रतिनिधित्व करता है। यह 0 से 4 तक होता है। 0 UNOPENED: open ()नहीं कॉल किया जाता है। 1 OPENED: open() कॉल किया जाता है। send ()नहीं कॉल किया जाता है । 2 HEADERS_RECEIVED: send ()कॉल किया जाता है, और हैडर और स्थिति उपलब्ध हैं। 3 LOADING: responseText डेटा रखता है। 4 DONE : ऑपरेशन कम्पलीट हो गया है।
reponseText	पाठ के रूप में प्रतिक्रिया देता है।
responseXML	XML के रूप में प्रतिक्रिया देता है।

XMLHttpRequest ऑब्जेक्ट के मेथड

XMLHttpRequest ऑब्जेक्ट के महत्वपूर्ण मेथड इस प्रकार हैं—

मेथड	विवरण
void open(method, URL)	GET या पोस्ट मेथड और यूआरएल को निर्दिष्ट करने का रिक्वेस्ट खोलता है।
void open(method, URL, async)	ऊपर जैसा है लेकिन निर्दिष्ट करता है की एसिंक्रोनस या नहीं।
void open(method, URL, async, username, password)	ऊपर जैसा है लेकिन यूजर नाम और पासवर्ड निर्दिष्ट करता है।
void send()	Get रिक्वेस्ट भेजता है।
void send(string)	पोस्ट रिक्वेस्ट भेजता है।
setRequestHeader(header,value)	यह रिक्वेस्टहैडर जोड़ता है।

XMLHttpRequest ऑब्जेक्ट बनाएँ

सभी आधुनिक ब्राउज़रों (क्रोम, फ़ायरफ़ॉक्स, IE7 +, Spe, सफारी ओपेरा) में एक अंतर्निहित XMLHttpRequest ऑब्जेक्ट है।

XMLHttpRequest ऑब्जेक्ट बनाने के लिए सिंटैक्स—

```
variable = new XMLHttpRequest();
```

उदाहरण—

```
var xhttp = new XMLHttpRequest();
```

AJAX-एक सर्वर को एक रिक्वेस्ट भेजें

XMLHttpRequest ऑब्जेक्ट का उपयोग सर्वर के साथ डेटा का आदान-प्रदान करने के लिए किया जाता है।

एक सर्वर के लिए एक रिक्वेस्ट भेजें

एक सर्वर को एक रिक्वेस्ट भेजने के लिए, हम और XMLHttpRequest ऑब्जेक्ट के open()और send() मेथड यूज़ करते हैं—

```
xhttp.open("GET", "ajax_info.txt", true);
xhttp.send();
```

मेथड	विवरण
open(method, url, async)	रिक्वेस्ट के टाइप को निर्दिष्ट करता है मेथडः रिक्वेस्ट का टाइपः GET या POST यूआरएलः सर्वर (फ़ाइल) location async: true (asynchronous) या false (synchronous)

send()	सर्वर के लिए रिक्वेस्ट भेजता है (GET के लिए उपयोग किया जाता है)
send(*string*)	सर्वर के लिए रिक्वेस्ट भेजता है (POST के लिए उपयोग किया जाता है)

GST या POST?

GET, POST की तुलना में सरल और तेज़ है, और इसका उपयोग ज्यादातर मामलों में किया जा सकता है। हालाँकि, हमेशा POST रिक्वेस्ट का उपयोग करें—

- Cache फ़ाइल options नहीं है (सर्वर पर फ़ाइल या डेटाबेस को अपडेट करें)।
- सर्वर को बड़ी मात्रा में डेटा भेजना (POST की कोई आकार सीमों नहीं हैं)।
- यूजर इनपुट भेजना (जिसमें अज्ञात वर्ण हो सकते हैं), POST GET की तुलना में अधिक मजबूत और सुरक्षित है।

GET रिक्वेस्ट

एक सरल GET रिक्वेस्टः

```
xhttp.open("GET", "demo_get.asp", true);
xhttp.send();
```

उपरोक्त उदाहरण में, आपको Cache परिणाम मिल सकता है। इससे बचने के लिए, URL में एक अद्वितीय आईडी जोड़ें—

```
xhttp.open ("GET", "Demo_get.asp? t =" + Math.random (), true);
xhttp:send ();
```

यदि आप GET मेथड के साथ जानकारी भेजना चाहते हैं, तो URL में जानकारी जोड़ें—

उदाहरण

```
xhttp.open ('GET', 'Demo_get2.asp? fname = Henry & lname = Ford', true);
xhttp.send ();
```

पोस्ट रिक्वेस्ट

एक साधारण पोस्ट रिक्वेस्टः

उदाहरण

```
xhttp.open ('POST', 'Demo_post.asp', true);
xhttp.send ();
```

HTML फॉर्म की तरह POST डेटा के लिए, setRequestHeader () के साथ HTTP हैडर जोड़ें। वह डेटा निर्दिष्ट करें जिसे आप send() मेथड भेजना चाहते हैं।

उदाहरण

```
xhttp.open('POST', 'demo_post2.asp', true);
xhttp.setRequestHeader('Content-type', 'application/x-www-form-urlencoded');
xhttp.send('fname=Henry&lname=Ford');
```

मेथड	विवरण
setRequestHeader(*header, value*)	रिक्वेस्ट करने के लिए HTTP हैडर जोड़ता है। हैडर:हैडर नाम निर्दिष्ट करता है। मूल्य: हैडर मान निर्दिष्ट करता है।

यूआरएल-एक सर्वर पर एक फ़ाइल

open () मेथड का url पैरामीटर, सर्वर पर एक फ़ाइल का एड्रेस है—

```
xhttp.open ('GET', 'ajax_test.asp', true);
```

फ़ाइल किसी भी तरह की फ़ाइल हो सकती है, जैसे .txt और .xml, या .asp और .php (जो प्रतिक्रिया वापस भेजने से पहले सर्वर पर एक्शन कर सकते हैं) जैसी सर्वर स्क्रिप्टिंग फ़ाइल।

Asynchronous - True या False?

सर्वर रिक्वेस्ट को एसिंक्रोनस रूप से भेजा जाना चाहिए।

open() मेथड के async पैरामीटर को true पर सेट किया जाना चाहिए—

```
xhttp.open ('GET', 'ajax_test.asp', true);
```

एसिंक्रोनस रूप से भेजने से, जावास्क्रिप्ट को सर्वर की प्रतिक्रिया का इंतजार नहीं करना पड़ता, बल्कि इसके बजाय—

- सर्वर प्रतिक्रिया की प्रतीक्षा करते समय अन्य स्क्रिप्ट Execute करें
- प्रतिक्रिया तैयार होने के बाद प्रतिक्रिया से निपटें

Onreadystatechange Property

XMLHttpRequest ऑब्जेक्ट के साथ आप रिक्वेस्ट को जवाब मिलने पर Execute होने वाली फ़ंक्शन को परिभाषित कर सकते हैं।

फ़ंक्शन को XMLHttpResponse ऑब्जेक्ट के onreadystatechange प्रॉपर्टी में परिभाषित किया गया है—

उदाहरण

```
xhttp.onreadystatechange = function() {
 if (this.readyState == 4 && this.status == 200) {
  document.getElementById("demo").innerHTML = this.responseText;
 }
};
xhttp.open("GET", "ajax_info.txt", true);
xhttp.send();
```

Synchronous रिक्वेस्ट

एक Synchronous रिक्वेस्ट को Execute करने के लिए, open() मेथड में तीसरे पैरामीटर को false में बदलें—

```
xhttp.open ('GET', 'ajax_info.txt', false);
```

कभी-कभी async = false का उपयोग त्वरित परीक्षण के लिए किया जाता है। आपको पुराने जावास्क्रिप्ट कोड में Synchronous रिक्वेस्ट भी मिलेंगे।

चूंकि कोड सर्वर के पूरा होने की प्रतीक्षा करेगा, इसलिए onreadystatechange फ़ंक्शन की कोई आवश्यकता नहीं है—

उदाहरण

```
xhttp.open("GET", "ajax_info.txt", false);
xhttp.send();
document.getElementById("demo").innerHTML = xhttp.responseText;
```

Onreadystatechange Property

रेडीस्टेट प्रॉपर्टी XMLHttpRequest के रिक्वेस्ट स्टेटस को रखता है।

Onreadystatechange प्रॉपर्टी एक फ़ंक्शन को तब परिभाषित करती है जब रेडीस्टेट में बदलाव होता है।

status प्रॉपर्टी और **statusText** प्रॉपर्टी XMLHttpRequest ऑब्जेक्ट की स्थिति रखती है।

प्रॉपर्टी	विवरण
onreadystatechange	एक फ़ंक्शन को परिभाषित करने के लिए कहा जाता है जब रेडीस्टेट प्रॉपर्टी बदल जाती है।
readyState	XMLHttpRequest की स्थिति प्राप्त करता है— 0: रिक्वेस्ट intialize नहीं हुई 1: सर्वर कनेक्शन एस्टाब्लिशड 2: रिक्वेस्ट received 3: प्रोसेसिंग रिक्वेस्ट 4: रिक्वेस्ट फिनिश्ड और रिस्पांस रेडी है
status	200: 'OK' 403: 'Forbidden' 404: 'Page not found'
statusText	स्थिति लौटाता है।

Onreadystatechange फ़ंक्शन को हर बार तैयार किए गए परिवर्तनों को कहा जाता है।

जब रेडीस्टेट 4 है और स्थिति 200 है, तो प्रतिक्रिया तैयार है:

उदाहरण

```
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("demo").innerHTML =
```

```
    this.responseText;
  }
};
xhttp.open("GET", "ajax_info.txt", true);
xhttp.send();
}
```

कॉलबैक फ़ंक्शन का उपयोग करना

कॉलबैक फ़ंक्शन एक फ़ंक्शन है जो किसी अन्य फ़ंक्शन के पैरामीटर के रूप में पारित किया जाता है।

यदि आपके पास एक वेबसाइट में एक से अधिक AJAX का कार्य है, तो आपको XMLHttpRequest ऑब्जेक्ट को Execute करने के लिए एक फ़ंक्शन और प्रत्येक AJAX कार्य के लिए एक कॉलबैक फ़ंक्शन बनाना चाहिए।

फ़ंक्शन कॉल में URL और प्रतिक्रिया तैयार होने पर किस फ़ंक्शन को कॉल करना होना चाहिए।

उदाहरण

```
loadDoc("url-1", myFunction1);

loadDoc("url-2", myFunction2);

function loadDoc(url, cFunction) {
 var xhttp;
 xhttp=new XMLHttpRequest();
 xhttp.onreadystatechange = function() {
  if (this.readyState == 4 && this.status == 200) {
   cFunction(this);
  }
 };
 xhttp.open("GET", url, true);
 xhttp.send();
}

function myFunction1(xhttp) {
 // action goes here
}
function myFunction2(xhttp) {
 // action goes here
}
```

Server Response Properties

प्रॉपर्टी	विवरण
responseText	एक स्ट्रिंग के रूप में प्रतिक्रिया डेटा प्राप्त करें।
responseXML	XML डेटा के रूप में प्रतिक्रिया डेटा प्राप्त करें।

सर्वर रिस्पांस मेथड

मेथड	विवरण
getResponseHeader ()	सर्वर रिसोर्सेज से विशिष्ट हैडर की जानकारी लौटाता है।
getAllResponseHeaders ()	सर्वर रिसोर्सेज से सभी हैडर की जानकारी लौटाता है।

ResponseText प्रॉपर्टी

ResponseText प्रॉपर्टी जावास्क्रिप्ट स्ट्रिंग के रूप में सर्वर प्रतिक्रिया देता है, और आप इसे तदनुसार उपयोग कर सकते हैं—

उदाहरण

```
document.getElementById("demo").innerHTML = xhttp.responseText;
```

ResponseXML प्रॉपर्टी

XML HttpRequest ऑब्जेक्ट में एक अंतर्निहित XML पार्सर है।

responseXML प्रॉपर्टी XML प्रतिक्रिया ऑब्जेक्ट के रूप में सर्वर प्रतिक्रिया देता है।

इस प्रॉपर्टी का उपयोग करके आप XML डोम ऑब्जेक्ट के रूप में प्रतिक्रिया पार्स कर सकते हैं—

उदाहरण

```
xmlDoc = xhttp.responseXML;txt = "";
x = xmlDoc.getElementsByTagName("ARTIST");
for (i = 0; i < x.length; i++) {
 txt += x[i].childNodes[0].nodeValue + "<br>";
 }
document.getElementById("demo").innerHTML = txt;
xhttp.open("GET", "cd_catalog.xml", true);
xhttp.send();
```

GetAllResponseHeaders () मेथड

GetAllResponseHeaders () मेथड सर्वर प्रतिक्रिया से सभी हैडर की जानकारी लौटाती है।

उदाहरण

```
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
 if (this.readyState == 4 && this.status == 200) {
  document.getElementById("demo").innerHTML =
  this.getAllResponseHeaders();
 }
};
```

GetResponseHeader () मेथड

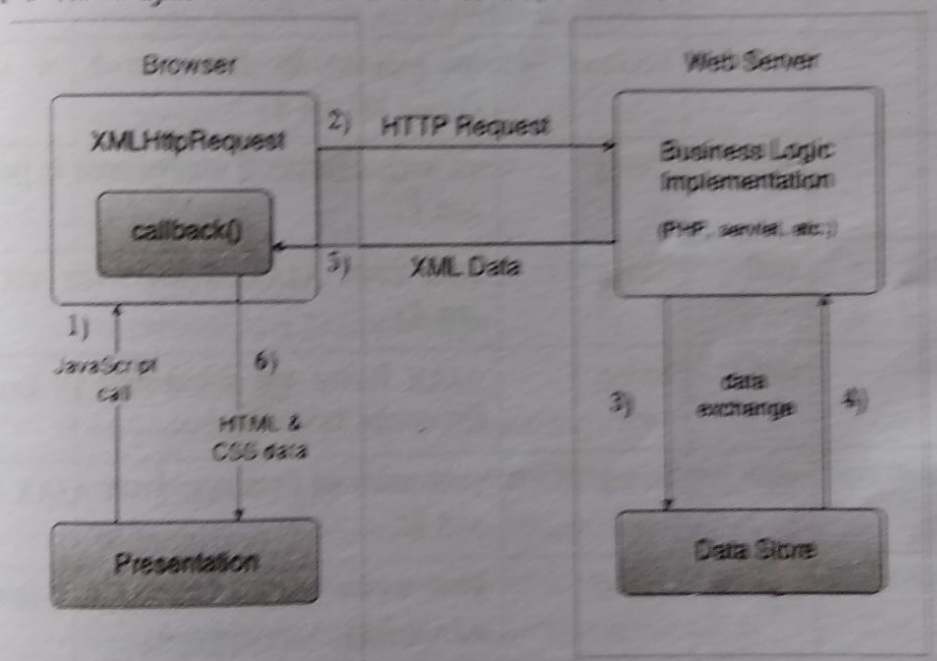
GetResponseHeader () मेथड विशिष्ट प्रतिक्रिया सर्वर जानकारी से हैडर की जानकारी देता है।

उदाहरण

```
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
 if (this.readyState == 4 && this.status == 200) {
  document.getElementById("demo").innerHTML =
  this.getResponseHeader("Last-Modified");
 }
};
xhttp.open("GET", "ajax_info.txt", true);
xhttp.send();
```

6.3 AJAX कैसे काम करता है?

AJAX XMLHttpRequest ऑब्जेक्ट का उपयोग कर सर्वर के साथ संचार करता है। आइए नीचे दिए गए चित्र द्वारा ajax के फंक्शन या ajax के कार्य करने के मेथड को समझने की कोशिश करें।



जैसा कि आप उपरोक्त उदाहरण में देख सकते हैं, XMLHttpRequest ऑब्जेक्ट एक महत्वपूर्ण भूमिका निभाता है।

1. यूजर UI से एक रिक्वेस्ट भेजता है और एक जावास्क्रिप्ट कॉल XMLHttpRequest ऑब्जेक्ट पर जाता है।
2. HTTP रिक्वेस्ट को XMLHttpRequest ऑब्जेक्ट द्वारा सर्वर पर भेजा जाता है।
3. सर्वर JSP, PHP, सर्वलेट, ASP.net आदि का उपयोग कर डेटाबेस के साथ इंटरैक्ट करता है।
4. डेटा पुनर्प्राप्त किया जाता है।
5. सर्वर XML डेटा या रॉ डेटा को XMLHttpRequest कॉलबैक फंक्शन पर भेजता है।
6. HTML और प्राप्त डेटा ब्राउज़र पर प्रदर्शित होता है।

6.4 प्राइमफेस AJAX

यह अपडेट, ईवेंट, इत्यादि जैसी विभिन्न ऐट्रिब्यूट्स प्रदान करता है। यहाँ, हम एक उदाहरण तैयार कर रहे हैं जो ajax ऐट्रिब्यूट्स बताता है।

AJAX ऐट्रिब्यूट्स

निम्न तालिका में AJAX ऐट्रिब्यूट्स हैं।

प्रॉपर्टी	डिफ़ॉल्ट मान	Return type	विवरण
listener	null	MethodExpr	इसे आंशिक रिक्वेस्ट में संसाधित करने के लिए उपयोग किया जाता है।
Immediate	false	boolean	यह एक बूलियन मान लौटाता है जो फेज को निर्धारित करता है।
async	false	boolean	जब true पर सेट किया जाता है, तो AJAX रिक्वेस्ट Queue नहीं होते हैं।
process	null	String	इसे आंशिक रिक्वेस्ट में संसाधित करने के लिए उपयोग किया जाता है।
update	null	String	इसका उपयोग AJAX के साथ अपडेट करने के लिए किया जाता है।
onstart	null	String	AJAX रिक्वेस्ट शुरू होने से पहले इसे Execute करने के लिए उपयोग किया जाता है।
oncomplete	null	String	इसका प्रयोग तब किया जाता है जब AJAX रिक्वेस्ट पूरा हो जाता है।
onsuccess	null	String	इसका उपयोग Execute करने के लिए किया जाता है जब AJAX रिक्वेस्ट सफल होता है।
delay	null	String	इसका उपयोग देरी (delay) करने के लिए, समय निर्धारित करने के लिए किया जाता है।
partialSubmit	false	boolean	आंशिक रूप से संसाधित घटकों से संबंधित मूल्यों के क्रमांकन को सक्षम करता है।
partialSubmitFilter	null	String	आंशिक submit दह होने पर चयन करने के लिए selector है। आंशिक रूप से संसाधित घटकों के सभी वंशज इनपुटों का चयन करने के लिए default value ': input' है।
event	null	String	क्लाइंट पक्ष event ajax रिक्वेस्ट को ट्रिगर करने के लिए।

Ajax Basic

यह उदाहरण एप्लीकेशन में एक अंतर्निहित Ajax उपयोग के बारे में बताता है। यह बैक-एंड मूल्य को अपडेट करता है और AJAX का उपयोग करके आउटपुट प्रदर्शित करता है।

इस उदाहरण में एक JSP पेज और एक ManagedBean शामिल है।

अनुप्रयुक्त फाइल

// ajax-basic.xhtml

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-
//W3C//DTD XHTML 1.0 Transitional//EN""http://www.w3.org/TR/xhtml1/DTD/xht
ml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:h="http://xmlns.jcp.org/jsf/html"
xmlns:p="http://primefaces.org/ui">
<h:head>
<title>Ajax Basic Example</title>
</h:head>
<h:body>
<h2>Basic Ajax Example</h2>
<h:form>
<h:panelGrid columns="4" cellpadding="5">
<h:outputLabel for="name" value="Name:" style="font-weight:bold" />
<p:inputText id="name" value="#{user.name}" />
<p:commandButton value="Submit" update="display" icon="ui-icon-check" />
<h:outputText id="display" value="#{user.name}" />
</h:panelGrid>
</h:form>
</h:body>
</html>
```

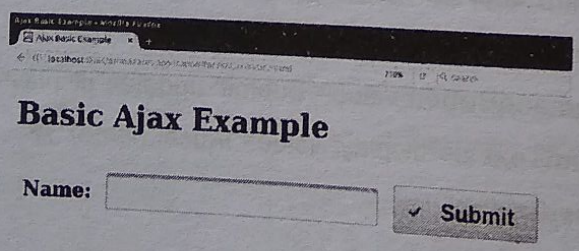
ManagedBean

// User.java

```
package com.javatpoint;
import java.io.Serializable;
```

```
import javax.faces.bean.ManagedBean;
@ManagedBean
public class User implements Serializable{
String name;
String lastName;
public String getName() {
return name;
}
public void setName(String name) {
this.name = name;
}

public String getLastName() {
return lastName;
}
public void setLastName(String lastName) {
this.lastName = lastName;
}
}
```

आउटपुटः



6.5 AJAX events

इस विशेषता का उपयोग निर्दिष्ट मेथड पर ईवेंट को ट्रिगर करने के लिए किया जाता है। हम इस विशेषता में ऑनक्लिक आदि इवेंटओं को पारित कर सकते हैं।

इनपुट घटक के लिए डिफॉल्ट क्लाइंट साइड इवेंट ऑनचेंज है। हम ईवेंट विशेषता का उपयोग करके इसे ओवरराइड कर सकते हैं। निम्नलिखित उदाहरण में, Ajax रिक्वेस्ट ट्रिगर किया जाता है जब इनपुट feild पर key होती है।

जेएसएफ फाइल

// ajax-event.xhtml

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-
//W3C//DTD XHTML 1.0 Transitional//EN""http://www.w3.org/TR/xhtml1/DTD/xht
ml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:h="http://xmlns.jcp.org/jsf/html"
xmlns:p="http://primefaces.org/ui">
<h:head>
<title>Ajax Event Example</title>
</h:head>
<h:body>
<h2>Ajax Event Example</h2>
<h:form>
<h:panelGrid columns="3" cellpadding="5">
<h:outputText value="KeyUp Event: " />
<p:inputText id="firstname" value="#{user.name}" placeholder="Enter Text">
<p:ajax event="keyup" update="out" />
</p:inputText>
<h:outputText id="out" value="#{user.name}" />
</h:panelGrid>
</h:form>
</h:body>
</html>
```

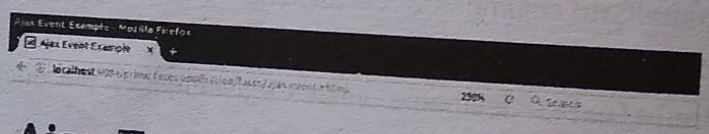
Managed Bean

// User.java

```
package com.asianpublishers;
import java.io.Serializable;
import javax.faces.bean.ManagedBean;
@ManagedBean
public class User implements Serializable{
String name;
String lastName;
```

```
public String getName() {
return name;
}
public void setName(String name) {
this.name = name;
}
public String getLastName() {
return lastName;
}
public void setLastName(String lastName) {
this.lastName = lastName;
}

}
}
```

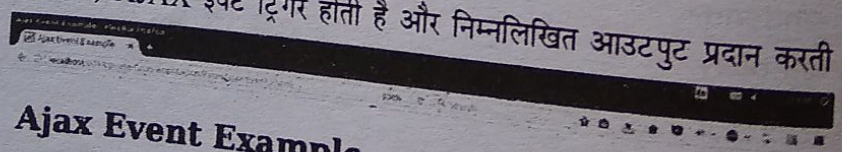
आउटपुट:



Ajax Event Example

KeyUp Event: Enter Text

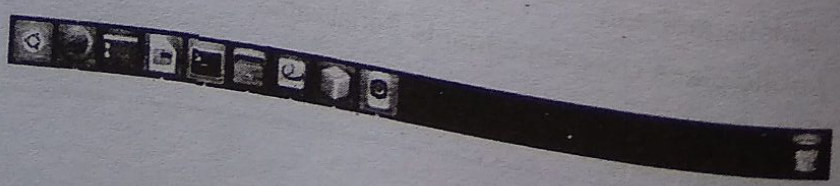
मान दर्ज करने के बाद, AJAX इवेंट ट्रिगर होती है और निम्नलिखित आउटपुट प्रदान करती है—



Ajax Event Example

KeyUp Event: asian publishers

asian publishers



6.6 Ajax Listener

इस विशेषता का उपयोग AJAX रिक्वेस्ट का उपयोग करके जावा मेथड को कॉल करने के लिए किया जाता है। यह <a: ajax> घटक का एक प्रॉपर्टी है।

यहां, हम एक उदाहरण बना रहे हैं जो ManagedBean से एक मेथड कॉल करता है। इसमें निम्न फाइलें और कोड शामिल हैं।

जेएसएफ फाइल

// ajax-listener.xhtml

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-
//W3C//DTD XHTML 1.0 Transitional//EN""http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:h="http://xmlns.jcp.org/jsf/html"
xmlns:p="http://primefaces.org/ui">
<h:head>
<title>Ajax Listener</title>
</h:head>
<h:body>
<h2>Ajax Listener Example</h2>
<h:form>
<h:panelGrid columns="3">
<h:outputText value="Enter: " />
<p:inputText id="counter" value="#{listenerBean.text}" placeholder="Enter Text">
<p:ajax event="keyup" update="out" listener="#{listenerBean.ajaxEvent()}" />
</p:inputText>
</h:panelGrid>
<h:panelGrid id="out">
<h2>Output</h2>
<h:outputText value="#{listenerBean.text}" />
</h:panelGrid>
</h:form>
</h:body>
</html>
```

Managed Bean

// ListenerBean.java

```
package com.asianpublishers;
import javax.faces.bean.ManagedBean;
@ManagedBean
public class ListenerBean {
String text;
public String getText() {
return text;
}
public void setText(String text) {
this.text = text;
}
public void ajaxEvent() {
text = text.toUpperCase();
}
}
```

आउटपुटः

Ajax Listener Example

Enter: Enter Text

Output

Ajax Listener Example

Enter: asian publishers

Output

ASIAN PUBLISHERS

6.7 AJAX एक्शनलिस्ट

इसका प्रयोग जावा मेथड को ट्रिगरिंग एक्शन के द्वारा किया जाता है। यह क्रिया या तो कमांड बटन या कमांड लिंक का उपयोग करके की जा सकती है। यहां, हम ManagedBean की एक मेथड को कॉल करते है और हर बार बटन क्लिक करने पर वर्तमान मान को AJAX के साथ अपडेट किया जाता है।

इस उदाहरण में निम्न फाइलें शामिल हैं।

जेएसएफ फाइल

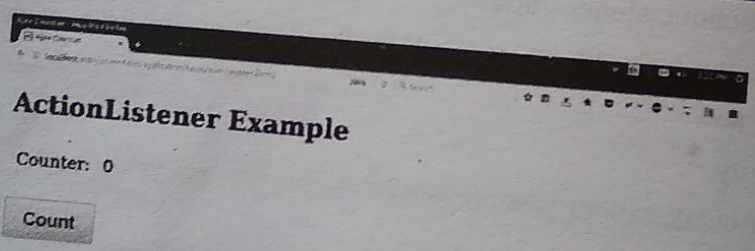
```
// actionListener.xhtml

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:h="http://xmlns.jcp.org/jsf/html"
xmlns:p="http://primefaces.org/ui">
<h:head>
<title>actionListener</title>
<meta name="viewport" content="width=device-width, initial-scale=1.0"/>
</h:head>
<h:body>
<h2>ActionListener Example</h2>
<h:form>
<h:panelGrid columns="2" cellpadding="5">
<h:outputText value="Counter: " />
<h:outputText id="output" value="#{ajaxCounter.counter}" />
</h:panelGrid><br/>
<p:commandButton value="Count" actionListener="#{ajaxCounter.increment()}"
 update="output" />
</h:form>
</h:body>
</html>
```

Managed Bean

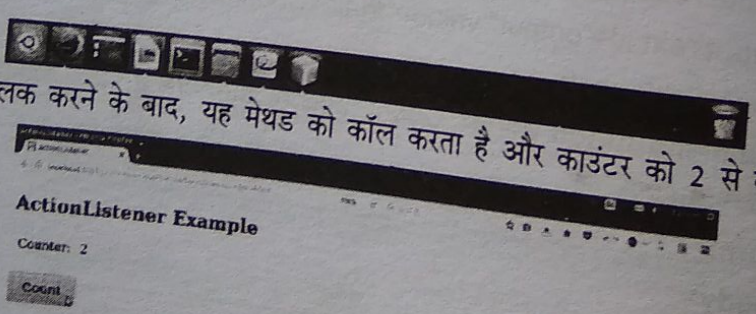
```
// AjaxCounter.java

package com.javatpoint;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.ViewScoped;
```

```
@ManagedBean
@ViewScoped
public class AjaxCounter {
int counter;
public int getCounter() {
return counter;
}
public void setCounter(int counter) {
this.counter = counter;
}
public void increment(){
counter+=2;
}
}
```

आउटपुट:



कमांडबटन पर क्लिक करने के बाद, यह मेथड को कॉल करता है और काउंटर को 2 से बढ़ाता है।



6.8 Ajax Validation

सर्वर साइड पर हमारे डेटा को मान्य करने के लिए Validation टैग है । AJAX VALIDATION करने के लिए <p: ajax> का उपयोग किया जाता है। यहां, हम एक उदाहरण बना रहे हैं जो दो इनपुट लेता है और सर्वर साइड पर मान्य होता है।

इस उदाहरण में निम्न फाइलें शामिल हैं।

JSF फाइल

// ajax-validation.xhtml

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-
//W3C//DTD XHTML 1.0 Transitional//EN""http://www.w3.org/TR/xhtml1/DTD/xht
ml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:h="http://xmlns.jcp.org/jsf/html"
xmlns:p="http://primefaces.org/ui"
xmlns:f="http://xmlns.jcp.org/jsf/core">
<h:head>
<title>ajax-validation</title>
</h:head>
<h:body>
<h:form>
<p:panel id="panel-id" header="Add a User">
<p:messages id="msgs" />
<h:panelGrid columns="3" cellpadding="5">
<p:outputLabel for="firstname" value="Firstname:" />
<p:inputText id="firstname" value="#{ajaxValidation.firstname}" required="true" l
abel="Firstname">
<f:validateLength minimum="2" />
</p:inputText>
<p:message for="firstname" display="icon" />
<p:outputLabel for="lastname" value="Lastname:" />
<p:inputText id="lastname" value="#{ajaxValidation.lastname}" label="Lastname
" required="true">
```

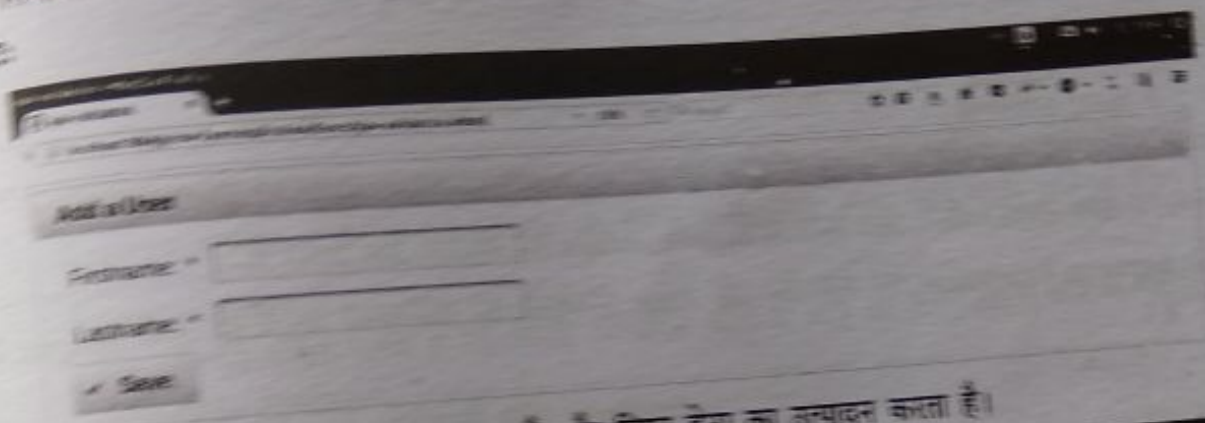
```
<f:validateLength minimum="2" />
<p:ajax update="msgLastname" event="keyup" />
</p:inputText>
<p:message for="lastname" id="msgLastname" display="icon" />
</h:panelGrid>
<p:commandButton value="Save" update="panel-
id" actionListener="#{ajaxValidation.save}" icon="ui-icon-check" />
</p:panel>
</h:form>
</h:body>
</html>
```

ManagedBean
// AjaxValidation.java

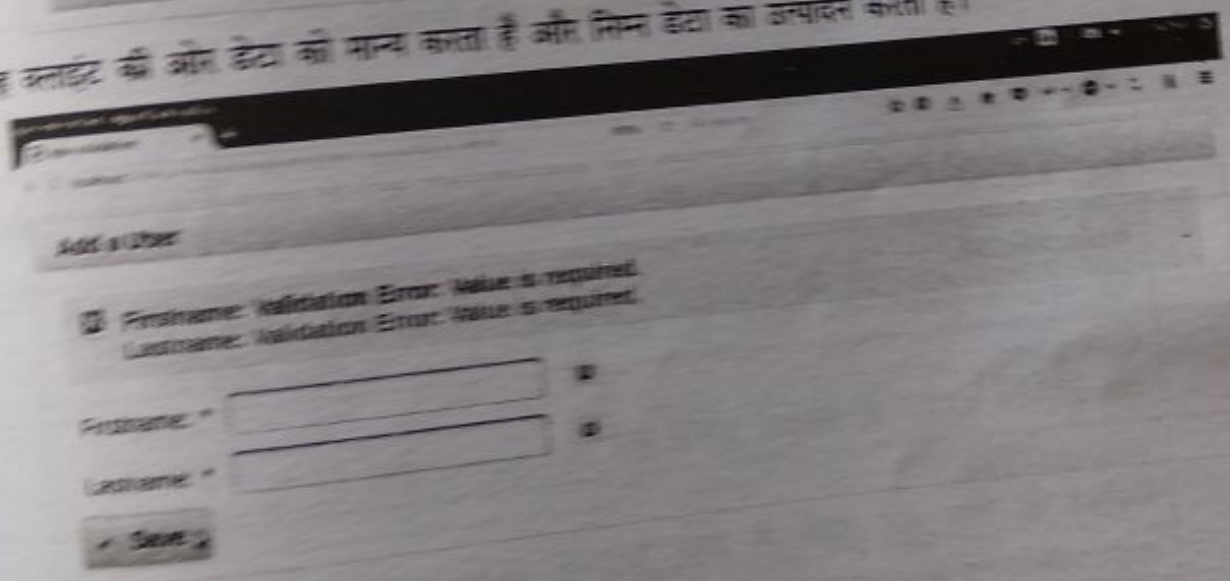
```
package com.javatpoint;
import javax.faces.application.FacesMessage;
import javax.faces.bean.ManagedBean;
import javax.faces.context.FacesContext;
@ManagedBean
public class AjaxValidation {
private String firstname;
private String lastname;
public String getFirstname() {
return firstname;
}
public void setFirstname(String firstname) {
this.firstname = firstname;
}
public String getLastname() {
return lastname;
}
public void setLastname(String lastname) {
this.lastname = lastname;
```

```
}
public void save() {
FacesContext.getCurrentInstance().addMessage(null,
new FacesMessage("Welcome " + firstname + " " + lastname));
}
}
```

यह निष्पादन के बाद निम्नलिखित आउटपुट का उत्पादन करता है।



यह क्लाइंट की ओर डेटा को मान्य करता है और निम्न डेटा का उत्पादन करता है।



6.9 jQuerry और AJAX(jQuerry and AJAX)

jQuery AJAX कार्यक्षमता के लिए कई मेथड प्रदान करता है। जैसे लाइब्रेरी में Ajax रिक्वेस्ट भेजने के लिए विभिन्न मेथड शामिल हैं। ये मेथड्स आंतरिक रूप से जावास्क्रिप्ट के XMLHttpRequest ऑब्जेक्ट का उपयोग करती है।

JQuery AJAX के मेथड

कई jQuery मेथड हैं जो AJAX प्रोग्रामिंग को आसान बनाते हैं। नीचे दिए गए तरीकों का वर्णन किया गया है—

1. jQuery.ajax (options)

JQuery.ajax (options) मेथड HTTP रिक्वेस्ट का उपयोग करके एक remote पृष्ठ लोड करती है।

$ .ajax,() XMLHttpRequest देता है जो इसे बनाता है। ज्यादातर मामलों में आपको उस ऑब्जेक्ट की सीधे तौर पर हेरफेर करने की आवश्यकता नहीं होगी, लेकिन यदि आपको मैन्युअल रूप से रिक्वेस्ट को रद्द करने की आवश्यकता है तो यह उपलब्ध है।

सिंटेक्स

यहाँ इस मेथड का उपयोग करने के लिए सरल सिंटेक्स है—

```
$.ajax( options )
```

पैरामीटर

इस मेथड द्वारा उपयोग किए जाने वाले सभी parameters का विवरण इस प्रकार है—

- **options**-key/value का एक सेट जो Ajax रिक्वेस्ट को कॉन्फिगर करता है। सभी options वैकल्पिक हैं।

अनु क्रमांक	Options और विवरण
1	**Async** एक बूलियन इंगित करता है कि रिक्वेस्ट को अतुल्यकालिक रूप से करना है या नहीं। डिफ़ॉल्ट मान true है।
2	**beforeSend** एक कॉलबैक फ़ंक्शन जिसे रिक्वेस्ट भेजे जाने से पहले Execute किया जाता है।
3	**Complete** एक कॉलबैक फ़ंक्शन जो रिक्वेस्ट समाप्त होने पर Execute होता है।
4	**contentType** रिक्वेस्ट के लिए सेट करने के लिए MIME contentType वाली स्ट्रिंग। डिफ़ॉल्ट मान application / x-www-form-urlencoded है।
5	**Data** एक map या स्ट्रिंग जो रिक्वेस्ट के साथ सर्वर को भेजी जाती है।
6	**dataFilter** XMLHttpRequest के raw responsed data को संभालने के लिए उपयोग किया जाने वाला फ़ंक्शन। यह एक पूर्व-फिल्टरिंग फ़ंक्शन है जो response को सैनिटाइज़ करने के लिए है।
7	**dataType** सर्वर (xml, html, json, या स्क्रिप्ट) से अपेक्षित डेटा के टाइप को परिभाषित करने वाली एक स्ट्रिंग।

8	**Error** एक कॉलबैक फ़ंक्शन जिसे रिक्वेस्ट विफल होने पर Execute किया जाता है।
9	**ग्लोबल** एक बूलियन संकेत करता है कि क्या इस रिक्वेस्ट से global AJAX event handlers को ट्रिगर किया जाएगा। डिफ़ॉल्ट मान true है।
10	**ifModified** एक बूलियन यह दर्शाता है कि क्या सर्वर को यह जांचना चाहिए कि रिक्वेस्ट के जवाब से पहले पृष्ठ को संशोधित किया गया है या नहीं।
11	**Jsonp** एक jsonp रिक्वेस्ट में कॉलबैक फ़ंक्शन नाम को ओवरराइड करता है।
12	**Password** HTTP एक्सेस ऑथेंटिकेशन रिक्वेस्ट के जवाब में इस्तेमाल किया जाने वाला पासवर्ड है।
13	**processData** एक बूलियन इंगित करता है कि किसी ऑब्जेक्ट फॉर्म से डेटा को क्वेरी-स्ट्रिंग रूप में परिवर्तित करना है या नहीं। डिफ़ॉल्ट मान true है।
14	**Success** एक कॉलबैक फ़ंक्शन जिसे रिक्वेस्ट सफल होने पर Execute किया जाता है।
15	**timeout** मिलीसेकंड की संख्या जिसके बाद रिक्वेस्ट विफलता में समय समाप्त हो जाएगा।
16	**Timeout** रिक्वेस्ट के लिए एक स्थानीय टाइमआउट (मिलीसेकंड में) सेट करें।
17	**Type** रिक्वेस्ट (GET या POST) के लिए उपयोग करने के लिए HTTP मेथड को परिभाषित करने वाली एक स्ट्रिंग। डिफ़ॉल्ट मान GET है।
18	**url** एक स्ट्रिंग जिसमें URL भेजा जाता है जिसमें रिक्वेस्ट भेजा जाता है।
19	**username** HTTP एक्सेस प्रमाणीकरण रिक्वेस्ट के जवाब में उपयोग किया जाने वाला यूजर नाम।
20	**XHR** XMLHttpRequest ऑब्जेक्ट बनाने के लिए कॉलबैक।

उदाहरण

मान लें कि हमारे पास result.html file में HTML कॉन्टेंट है—

```
<h1>THIS IS RESULT...</h1>
```

निम्नलिखित एक सरल उदाहरण है जो इस मेथड के उपयोग को दर्शाता है। यहाँ हम HTML को वापस लाने के लिए सफल हैंडलर का उपयोग करते हैं—

```
<html>
  <head>
    <title>The jQuery Example</title>
    <script type = "text/javascript"
      src = "https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js">
    </script>

    <script type = "text/javascript" language = "javascript">
      $(document).ready(function() {
        $("#driver").click(function(event){
          $.ajax( {
            url:'result.html',
            success:function(data) {
              $('#stage').html(data);
            }
          });
        });
      });
    </script>
  </head>

  <body>
    <p>Click on the button to load result.html file:</p>

    <div id = "stage" style = "background-color:blue;">
      STAGE
    </div>

    <input type = "button" id = "driver" value = "Load Data" />
  </body>
</html>
```

2. jQuery.ajaxSetup(options)

JQuery.ajaxSetup (options) मेथड भविष्य के AJAX रिक्वेस्ट के लिए global सेटिंग्स सेट करता है।

सिंटेक्स

यहाँ इस मेथड का उपयोग करने के लिए सरल सिंटेक्स है—

```
$.ajaxSetup (options)
```

पैरामीटर

इस मेथड द्वारा उपयोग किए जाने वाले सभी parameters का विवरण इस प्रकार है—

- options-key/value का एक सेट जो Ajax रिक्वेस्ट को कॉन्फिगर करता है। सभी options वैकल्पिक हैं।

अनु क्रमांक	Options और विवरण
1	**Async** एक बूलियन इंगित करता है कि रिक्वेस्ट को अतुल्यकालिक रूप से करना है या नहीं। डिफ़ॉल्ट मान true है।
2	**beforeSend** एक कॉलबैक फ़ंक्शन जिसे रिक्वेस्ट भेजे जाने से पहले Execute किया जाता है।
3	**Complete** एक कॉलबैक फ़ंक्शन जो रिक्वेस्ट समाप्त होने पर Execute होता है।
4	**contentType** रिक्वेस्ट के लिए सेट करने के लिए MIME contentType वाली स्ट्रिंग। डिफ़ॉल्ट मान application / x-www-form-urlencoded है।
5	**Data** एक map या स्ट्रिंग जो रिक्वेस्ट के साथ सर्वर को भेजी जाती है।
6	**dataFilter** XMLHttpRequest के raw responsed data को संभालने के लिए उपयोग किया जाने वाला फ़ंक्शन। यह एक पूर्व-फिल्टरिंग फ़ंक्शन है जो डेटा को सैनिटाइज़ करने के लिए है।
7	**dataType** सर्वर (xml, html, json, या स्क्रिप्ट) से अपेक्षित डेटा के टाइप को परिभाषित करने वाली एक स्ट्रिंग।
8	**Error** एक कॉलबैक फ़ंक्शन जिसे रिक्वेस्ट विफल होने पर Execute किया जाता है।
9	**ग्लोबल** एक बूलियन संकेत करता है कि क्या इस रिक्वेस्ट से global AJAX event handlers को ट्रिगर किया जाएगा। डिफ़ॉल्ट मान true है।
10	**ifModified** एक बूलियन यह दर्शाता है कि क्या सर्वर को यह जांचना चाहिए कि रिक्वेस्ट के जवाब से पहले पृष्ठ को संशोधित किया गया है या नहीं।
11	**Jsonp** एक रेदहज रिक्वेस्ट में कॉलबैक फ़ंक्शन नाम को ओवरराइड करता है।
12	**Password** HTTP एक्सेस ऑथेंटिकेशन रिक्वेस्ट के जवाब में इस्तेमाल किया जाने वाला पासवर्ड है।

13	**processData** एक बूलियन इंगित करता है कि किसी ऑब्जेक्ट फॉर्म से डेटा को क्वेरी-स्ट्रिंग रूप में परिवर्तित करना है या नहीं। डिफ़ॉल्ट मान true है।
14	**Success** एक कॉलबैक फ़ंक्शन जिसे रिक्वेस्ट सफल होने पर Execute किया जाता है।
15	**timeout** मिलीसेकंड की संख्या जिसके बाद रिक्वेस्ट विफलता में समय समाप्त हो जाएगा।
16	**Timeout** रिक्वेस्ट के लिए एक स्थानीय टाइमआउट (मिलीसेकंड में) सेट करें।
17	**Type** रिक्वेस्ट (GET या POST) के लिए उपयोग करने के लिए HTTP मेथड को परिभाषित करने वाली एक स्ट्रिंग। डिफ़ॉल्ट मान GET है।
18	**url** एक स्ट्रिंग जिसमें URL भेजा जाता है जिसमें रिक्वेस्ट भेजा जाता है।
19	**username** HTTP एक्सेस प्रमाणीकरण रिक्वेस्ट के जवाब में उपयोग किया जाने वाला यूजर नाम।
20	**XHR** XMLHttpRequest ऑब्जेक्ट बनाने के लिए कॉलबैक।

दाहरण

मान लें कि हमारे पास result.html file में HTML कॉन्टेंट है—

```
<h1>THIS IS RESULT...</h1>
```

निम्नलिखित एक सरल उदाहरण है जो इस मेथड के उपयोग को दर्शाता है। यहाँ हम HTML को वापस लाने के ए सफल हैंडलर का उपयोग करते हैं—

```
<html>
  <head>
    <title>The jQuery Example</title>
    <script type = "text/javascript"
      src = "https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js">
    </script>

    <script type = "text/javascript" language = "javascript">
      $(document).ready(function() {

        $("#driver").click(function(event){
          // Do global setting.
          $.ajaxSetup({
            url: "result.html"
```

```
<
</h
<bo
<
<
<
<
<i
</bo
</htm
```

3. **jQuery.get**

JQuery.get

लोड करती है।

मेथड XMLH

सिंटेक्स

यहाँ इस मेथड

$.get(

पैरामीटर

इस मेथड द्वारा

- **url** - U
- **data** -
- **callbac**
  Execut
- **type** -
  है— 'xr

उदाहरण

मान लें कि हम

```
        });

        $.ajax( {
          success:function(data) {
            $('#stage').html(data);
          }
        });
      });

    });
  </script>
</head>

<body>
  <p>Click on the button to load result.html file:</p>

  <div id = "stage" style = "background-color:#cc0;">
    STAGE
  </div>

  <input type = "button" id = "driver" value = "Load Data" />
</body>
</html>
```

3. **jQuery.get (url, [data], [callback], [type])**

JQuery.get (url, [data], [callback], [type]) मेथड GET HTTP रिक्वेस्ट का उपयोग करके सर्वर से डेटा लोड करती है।

मेथड XMLHttpRequest ऑब्जेक्ट देता है।

सिंटेक्स

यहाँ इस मेथड का उपयोग करने के लिए सरल सिंटेक्स है—

```
$.get( url, [data], [callback], [type] )
```

पैरामीटर

इस मेथड द्वारा उपयोग किए जाने वाले सभी parameters का विवरण इस प्रकार है—

- **url** - एक स्ट्रिंग जिसमें URL है जिससे रिक्वेस्ट भेजा जाती है।
- **data** - यह वैकल्पिक पैरामीटर key/value जोड़े का प्रतिनिधित्व करता है जो सर्वर को भेजा जाएगा।
- **callback** – जब भी डेटा सफलतापूर्वक लोड किया जाता है, तो यह वैकल्पिक पैरामीटर एक फ़ंक्शन को Execute करता है।
- **type** - यह वैकल्पिक पैरामीटर कॉलबैक फ़ंक्शन में दिए जाने वाले डेटा के टाइप का प्रतिनिधित्व करता है— 'xml', 'html', 'efm›eâhš', 'json', 'jsonp', या 'टेक्स्ट'।

उदाहरण

मान लें कि हम result.php फ़ाइल में PHP कॉन्टेंट का अनुसरण कर रहे हैं—

```
<?php
if( $_REQUEST["name"] ) {

  $name = $_REQUEST['name'];
  echo "Welcome ". $name;
}

?>
```

निम्नलिखित एक सरल उदाहरण है जो इस मेथड के उपयोग को दर्शाता है—

```
<html>
  <head>
    <title>The jQuery Example</title>
    <script type = "text/javascript"
      src = "https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js">
    </script>

    <script type = "text/javascript" language = "javascript">
      $(document).ready(function() {

        $("#driver").click(function(event){
          $.get(
            "result.php",
            { name: "Zara" },
            function(data) {
              $('#stage').html(data);
            }
          );
        });

      });
    </script>
  </head>

  <body>
    <p>Click on the button to load result.html file -</p>

    <span id = "stage" style = "background-color:#cc0;">
      STAGE
    </span>

    <div><input type = "button" id = "driver"
      value = "Load Data" /></div>


  </body>

</html>
```

jQuery.getJSON (url, [data], [callback])

jQuery.getJSON (url, [data], [callback]) मेथड GSON HTTP रिक्वेस्ट का उपयोग करके सर्वर से JSON डेटा लोड करती है।

मेथड XMLHttpRequest ऑब्जेक्ट देता है।

सिंटेक्स

यहाँ इस मेथड का उपयोग करने के लिए सरल सिंटेक्स है—

```
$.get( url, [data], [callback], [type] )
```

पैरामीटर

इस मेथड द्वारा उपयोग किए जाने वाले सभी parameters का विवरण इस प्रकार है—

- **url** - एक स्ट्रिंग जिसमें URL है जिससे रिक्वेस्ट भेजा जाती है।
- **data** - यह वैकल्पिक पैरामीटर key/value जोड़े का प्रतिनिधित्व करता है जो सर्वर को भेजा जाएगा।
- **callback** - जब भी डेटा सफलतापूर्वक लोड किया जाता है, तो यह वैकल्पिक पैरामीटर एक फ़ंक्शन को Execute करता है।

उदाहरण

मान लें कि हम result.json फ़ाइल में JSON कॉन्टेंट का अनुसरण कर रहे हैं—

```
<?php
if( $_REQUEST["name"] ) {

  $name = $_REQUEST['name'];
  echo "Welcome ". $name;
}

?>
```

निम्नलिखित एक सरल उदाहरण है जो इस मेथड के उपयोग को दर्शाता है—

```
<html>
  <head>
    <title>The jQuery Example</title>
    <script type = "text/javascript"
      src = "https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js">
    </script>

    <script type = "text/javascript" language = "javascript">
      $(document).ready(function() {

        $("#driver").click(function(event){
          $.get(
            "result.php",
            { name: "Zara" },
            function(data) {
              $('#stage').html(data);
            }
```

```
            );
          });

        });
      </script>
    </head>

    <body>
      <p>Click on the button to load result.html file -</p>

      <span id = "stage" style = "background-color:#cc0;">
        STAGE
      </span>

      <div><input type = "button" id = "driver"
        value = "Load Data" /></div>

    </body>
  </html>
```

5. jQuery.getScript(url, [callback])

JQuery.getScript (url, [callback]) मेथड HTTP GET रिक्वेस्ट का उपयोग करके जावास्क्रिप्ट फ़ाइल को लोड और Execute करता है।

मेथड XMLHttpRequest ऑब्जेक्ट देता है।

सिंटेक्स

यहाँ इस मेथड का उपयोग करने के लिए सरल सिंटेक्स है—

```
$.getScript (url, [कॉलबैक])
```

पैरामीटर

इस मेथड द्वारा उपयोग किए जाने वाले सभी parameters का विवरण इस प्रकार है—

- **url** - एक स्ट्रिंग जिसमें URL है जिससे रिक्वेस्ट भेजा जाती है।
- **callback** - जब भी डेटा सफलतापूर्वक लोड किया जाता है, तो यह वैकल्पिक पैरामीटर एक फ़ंक्शन को Execute करता है।

उदाहरण

मान लें कि हम result.js फ़ाइल में जावास्क्रिप्ट कॉन्टेंट का अनुसरण कर रहे हैं—

```
function CheckJS(){
  alert("This is JavaScript");
}
```

निम्नलिखित एक सरल उदाहरण है जो इस मेथड के उपयोग को दर्शाता है—

```
<html>
  <head>
    <title>The jQuery Example</title>
    <script type = "text/javascript"
```

```
  src = "https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js">
 </script>

 <script type = "text/javascript" language = "javascript">
  $(document).ready(function() {

    $("#driver").click(function(event){
      $.getScript('result.js', function(jd) {
        // Call custom function defined in script
        CheckJS();
      });
    });

  });
 </script>
</head>


<body>
 <p>Click on the button to load result.js file –</p>

 <div id = "stage" style = "background-color:cc0;">
  STAGE
 </div>


 <input type = "button" id = "driver" value = "Load Data" />
</body>
</html>
```

6. **jQuery.post (url, [data], [callback], [type])**

JQuery.post (url, [data], [callback], [type]) मेथड POST HTTP रिक्वेस्ट का उपयोग करके सर्वर से एक पेज लोड करता है।

मेथड XMLHttpRequest ऑब्जेक्ट देता है।

सिंटेक्स

यहाँ इस मेथड का उपयोग करने के लिए सरल सिंटेक्स है—

```
$.post (url, [data], [callback], [type])
```

पैरामीटर

इस मेथड द्वारा उपयोग किए जाने वाले सभी parameters का विवरण इस प्रकार है—

- **url** – एक स्ट्रिंग जिसमें URL भेजा जाता है जिसमें रिक्वेस्ट भेजा जाती है।
- **data** – यह वैकल्पिक पैरामीटर key/value जोड़े या .serialize () फ़ंक्शन का रिटर्न मान दर्शाता है जो सर्वर को भेजा जाएगा।

- **callback** – जब भी डेटा सफलतापूर्वक लोड किया जाता है, तो यह वैकल्पिक पैरामीटर एक फ़ंक्शन को Execute करता है।
- type – यह वैकल्पिक पैरामीटर कॉलबैक फ़ंक्शन में लौटाए जाने वाले एक टाइप के डेटा का प्रतिनिधित्व करता है—'xml', 'html', 'efm›eâhš', 'json', 'jsonp', या 'टेक्स्ट'।

उदाहरण

मान लें कि हम result.php फ़ाइल में इस्ड कॉन्टेंट का अनुसरण कर रहे हैं—

```
<?php
if( $_REQUEST["name"] ) {

  $name = $_REQUEST['name'];
  echo "Welcome ". $name;
}

?>
```

निम्नलिखित एक सरल उदाहरण है जो इस मेथड के उपयोग को दर्शाता है—

```
<html>
  <head>
    <title>The jQuery Example</title>
    <script type = "text/javascript"
      src = "https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js">
    </script>

    <script type = "text/javascript" language = "javascript">
      $(document).ready(function() {

        $("#driver").click(function(event){

          $.post(
            "result.php",
            { name: "Zara" },
            function(data) {
              $('#stage').html(data);
            }
          );

        });

      });
    </script>
  </head>

  <body>
    <p>Click on the button to load result.html file –</p>
```

```
    <div id = "stage" style = "background-color:cc0;">
      STAGE
    </div>

    <input type = "button" id = "driver" value = "Load Data" />
  </body>
</html>
```

7. **load(url, [data], [callback])**

load(url, data, callback) मेथड सर्वर से डेटा लोड करता है और रिटर्न किए गए HTML को मिलान किए गए तत्व में रखता है।

सिंटेक्स

यहाँ इस मेथड का उपयोग करने के लिए सरल सिंटेक्स है—

```
[selector].load( url, [data], [callback] )
```

पैरामीटर

इस मेथड द्वारा उपयोग किए जाने वाले सभी parameters का विवरण इस प्रकार है—

- **url** – एक स्ट्रिंग जिसमें URL भेजा जाता है जिसमें रिक्वेस्ट भेजा जाती है।
- **data** – यह वैकल्पिक पैरामीटर रिक्वेस्ट के साथ भेजे जाने वाले डेटा के मानचित्र का प्रतिनिधित्व करता है।
- **callback** – यह वैकल्पिक पैरामीटर एक फ़ंक्शन का प्रतिनिधित्व करता है जिसे रिक्वेस्ट सफल होने पर Execute किया जाता है

उदाहरण

मान लें कि हमारे पास result.html फ़ाइल में HTML कॉन्टेंट है—

```
<h1>THIS IS RESULT...</h1>
```

निम्नलिखित एक सरल उदाहरण है जो इस मेथड के उपयोग को दर्शाता है।

```
<html>
  <head>
    <title>The jQuery Example</title>
    <script type = "text/javascript"
      src = "https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js">
    </script>

    <script type = "text/javascript" language = "javascript">
      $(document).ready(function() {
        $("#driver").click(function(event){
          $('#stage').load('result.html');
        });
      });
    </script>
  </head>

  <body>
    <p>Click on the button to load result.html file:</p>
```

```
    <div id = "stage" style = "background-color:cc0;">
      STAGE
    </div>

    <input type = "button" id = "driver" value = "Load Data" />
  </body>
</html>
```

8. serialize( )

serialize( ) मेथड डेटा के एक स्ट्रिंग में इनपुट तत्वों के एक सेट को serialize करता है।

सिंटेक्स

यहाँ इस मेथड का उपयोग करने के लिए सरल सिंटेक्स है—

```
$.serialize( )
```

उदाहरण

मान लें कि हमारे पास serialize.php फ़ाइल में इस्ड कॉन्टेंट है—

```
<?php
if( $_REQUEST["name"] ) {

  $name = $_REQUEST['name'];
  echo "Welcome ". $name;
  $age = $_REQUEST['age'];
  echo "<br />Your age : ". $age;
  $sex = $_REQUEST['sex'];
  echo "<br />Your gender : ". $sex;
}
?>
```

निम्नलिखित एक सरल उदाहरण है जो इस मेथड के उपयोग को दर्शाता है—

```
<html>
  <head>
    <title>The jQuery Example</title>
    <script type = "text/javascript"
      src = "https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js">
    </script>

    <script type = "text/javascript" language = "javascript">
      $(document).ready(function() {

        $("#driver").click(function(event){

          $.post(
```

```
        "/jquery/serialize.php",
        $("#testform").serialize(),

        function(data) {
           $('#stage1').html(data);
        }

       );

       var str = $("#testform").serialize();
       $("#stage2").text(str);
      });

    });
  </script>
</head>

<body>
  <p>Click on the button to load result.html file:</p>

  <div id = "stage1" style = "background-color:blue;">
    STAGE - 1
  </div>
  <br />

  <div id = "stage2" style = "background-color:blue;">
    STAGE - 2
  </div>

  <form id = "testform">
    <table>
      <tr>
        <td><p>Name:</p></td>
        <td><input type = "text" name = "name" size = "40" /></td>
      </tr>

      <tr>
        <td><p>Age:</p></td>
        <td><input type = "text" name = "age" size = "40" /></td>
      </tr>

      <tr>
```

```
            <td><p>Sex:</p></td>
            <td> <select name = "sex">
              <option value = "Male" selected>Male</option>
              <option value = "Female" selected>Female</option>
            </select></td>
          </tr>

          <tr>
            <td colspan = "2">
              <input type = "button" id = "driver" value = "Load Data" />
            </td>
          </tr>
        </table>
      </form>
    </body>
  </html>
```

9. serializeArray ()

SerializeArray () मेथड .serialize () मेथड की तरह सभी रूपों और फार्म तत्वों को क्रमबद्ध करती है, लेकिन आपके साथ काम करने के लिए एक JSON डेटा संरचना लौटाता है।

सिंटेक्स

यहाँ इस मेथड का उपयोग करने के लिए सरल सिंटेक्स है—

```
$.serializeArray ()
```

उदाहरण

मान लें कि हमारे पास serialize.php फ़ाइल में PHP कॉन्टेंट है—

```
<?php
if( $_REQUEST["name"] ) {

  $name = $_REQUEST['name'];
  echo "Welcome ". $name;
  $age = $_REQUEST['age'];
  echo "<br />Your age : ". $age;
  $sex = $_REQUEST['sex'];
  echo "<br />Your gender : ". $sex;
}
?>
```

निम्नलिखित एक सरल उदाहरण है जो इस मेथड के उपयोग को दर्शाता है—

```
<html>
  <head>
    <title>The jQuery Example</title>
    <script type = "text/javascript"
      src = "https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js">
```

```
</script>

<script type = "text/javascript" language = "javascript">
  $(document).ready(function() {

    $("#driver").click(function(event){

      $.post(
        "/jquery/serialize.php",
        $("#testform").serializeArray(),
        function(data) {
          $('#stage1').html(data);
        }
      );

      var fields = $("#testform").serializeArray();
      $("#stage2").empty();

      jQuery.each(fields, function(i, field){
        $("#stage2").append(field.value + " ");
      });

    });

  });
  </script>
</head>

<body>
  <p>Click on the button to load result.html file:</p>

  <div id = "stage1" style = "background-color:blue;">
    STAGE - 1
  </div>

  <br />

  <div id = "stage2" style = "background-color:blue;">
    STAGE - 2
  </div>

  <form id = "testform">
    <table>
      <tr>
        <td><p>Name:</p></td>
        <td><input type = "text" name = "name" size = "40" /></td>
      </tr>

      <tr>
```

```
    });
  </script>
</head>

<body>
  <p>Click on the button to load result.html file:</p>

  <div id = "stage1" style = "background-color:blue;">
    STAGE - 1
  </div>

  <div id = "stage2" style = "background-color:blue;">
    STAGE - 2
  </div>

  <input type = "button" id = "driver" value = "Load Data" />
</body>

</html>
```

2. ajaxStart(callback) मेथड एक फ़ंक्शन mebueie>(attach) करती है जब भी AJAX रिक्वेस्ट शुरू होता है और पहले से ही सक्रिय नहीं होता है। यह एक Ajax इवेंट है।

सिंटेक्स

यहाँ इस मेथड का उपयोग करने के लिए सरल सिंटेक्स है—

```
$(document).ajaxStart( callback )
```

पैरामीटर

इस मेथड द्वारा उपयोग किए जाने वाले सभी parameters का विवरण इस प्रकार है—

- **callback** – कार्य िम्लू करने के लिए।

उदाहरण

मान लें कि हमारे पास परिणाम HTML फ़ाइल में HTML कॉन्टेंट है—

```
<h1>THIS IS RESULT...</h1>
```

निम्नलिखित एक सरल उदाहरण है जो इस मेथड के उपयोग को दर्शाता है।

```
<html>
  <head>
    <title>The jQuery Example</title>
    <script type = "text/javascript"
      src = "https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js">
    </script>

    <script type = "text/javascript" language = "javascript">
      $(document).ready(function() {
        /* Global variable */
        var count = 0;
```

```
      $("#driver").click(function(event){
        $('#stage1').load('result.html');
      });

      /* Gets called when request starts */

      $(document).ajaxStart(function(){
        count++;
        $("#stage2").html("<h1>Starts, Count :" + count + "</h1>");
      });

      /* Gets called when request complete */
      $(document).ajaxComplete(function(event,request,set){
        count++;
        $("#stage3").html("<h1>Completes,Count:" + count + "</h1>");
      });

    });
  </script>
</head>

<body>
  <p>Click on the button to load result.html file:</p>

  <div id = "stage1" style = "background-color:blue;">
    STAGE - 1
  </div>

  <div id = "stage2" style = "background-color:blue;">
    STAGE - 2
  </div>

  <div id = "stage3" style = "background-color:blue;">
    STAGE - 3
  </div>

  <input type = "button" id = "driver" value = "Load Data" />
</body>

</html>
```

3. AJAX रिक्वेस्ट विफल होने पर ajaxError(callback) मेथड एक कार्य Execute करता है। यह एक Ajax इवेंट है।

सिंटेक्स

यहाँ इस मेथड का उपयोग करने के लिए सरल सिंटेक्स है—

```
$(document).ajaxError( callback )
```

पैरामीटर

इस मेथड द्वारा उपयोग किए जाने वाले सभी parameters का विवरण इस प्रकार है—

- **callback**—कार्य Execute करने के लिए। XMLHttpRequest और उस रिक्वेस्ट के लिए उपयोग की जाने वाली सेटिंग्स इस फ़ंक्शन के तर्क के रूप में पारित की जाती हैं। एक तीसरा तर्क, एक एक्सेप्शन ऑब्जेक्ट, पारित किया जाता है यदि रिक्वेस्ट को संसाधित करते समय एक एक्सेप्शन उत्पन्न होता है।

उदाहरण

निम्नलिखित एक सरल उदाहरण है जो इस मेथड के उपयोग को दर्शाता है।

```
<html>
  <head>
    <title>The jQuery Example</title>
    <script type = "text/javascript"
      src = "https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js">
    </script>

    <script type = "text/javascript" language = "javascript">
      $(document).ready(function() {

        $("#driver").click(function(event){
          /* Assume result.text does not exist. */
          $('#stage1').load('/jquery/result.text');
        });

        $(document).ajaxError(function(event, request, settings ){
          $("#stage2").html("<h1>Error in loading page.</h1>");
        });

      });
    </script>
  </head>

  <body>
    <p>Click on the button to load result.text file:</p>

    <div id = "stage1" style = "background-color:blue;">
      STAGE - 1
    </div>

    <div id = "stage2" style = "background-color:blue;">
      STAGE - 2
    </div>

    <input type = "button" id = "driver" value = "Load Data" />
  </body>

</html>
```

4. ajaxSend(callback) मेथड जब AJAX रिक्वेस्ट भेजा जाता है तो एक कार्य Execute किया जाता है। यह एक Ajax इवेंट है।

सिंटेक्स

यहाँ इस मेथड का उपयोग करने के लिए सरल सिंटेक्स है—

```
$(document).ajaxSend( callback )
```

पैरामीटर

इस मेथड द्वारा उपयोग किए जाने वाले सभी parameters का विवरण इस प्रकार है—

- **callback**–कार्य Execute करने के लिए। XMLHttpRequest और उस रिक्वेस्ट के लिए उपयोग की जाने वाली सेटिंग्स को कॉलबैक के तर्क के रूप में पारित किया जाता है।

उदाहरण

मान लें कि हमारे पास परिणाम HTML फ़ाइल में HTML कॉन्टेंट है—

```
<h1>THIS IS RESULT...</h1>
```

निम्नलिखित एक सरल उदाहरण है जो इस मेथड के उपयोग को दर्शाता है।

```
<html>
  <head>
    <title>The jQuery Example</title>
    <script type = "text/javascript"
      src = "https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js">
    </script>

    <script type = "text/javascript" language = "javascript">
      $(document).ready(function() {

        /* Global variable */
        var count = 0;

        $("#driver").click(function(event){
          $('#stage0').load('result.html');
        });

        /* Gets called when request starts */
        $(document).ajaxStart(function(){
          count++;
          $("#stage1").html("<h1>Starts, Count :" + count + "</h1>");
        });

        /* Gets called when request is sent */
        $(document).ajaxSend(function(evt, req, set){
          count++;
          $("#stage2").html("<h1>Sends, Count :" + count + "</h1>");
          $("#stage2").append("<h1>URL :" + set.url  + "</h1>");
        });
```

```
        /* Gets called when request co:nplete */
        $(document).ajaxComplete(function(event,request,settings){
          count++;
          $("#stage3").html("<h1>Completes, Count :" + count + "</h1>");
        });

      });
    </script>
  </head>

  <body>
    <p>Click on the button to load result.html file:</p>

    <div id = "stage0" style = "background-color:blue;">
      STAGE - 0
    </div>

    <div id = "stage1" style = "background-color:blue;">
      STAGE - 1
    </div>

    <div id = "stage2" style = "background-color:blue;">
      STAGE - 2
    </div>

    <div id = "stage3" style = "background-color:blue;">
      STAGE - 3
    </div>

    <input type = "button" id = "driver" value = "Load Data" />
  </body>

</html>
```

5. ajaxStop(callback) मेथड जब भी AJAX के सभी रिक्वेस्ट समाप्त हो जाते हैं, तो एक कार्य Execute किया जाता है। यह एक Ajax इवेंट है।

सिंटेक्स

यहाँ इस मेथड का उपयोग करने के लिए सरल सिंटेक्स है—

```
$(document).ajaxStop( callback )
```

पैरामीटर

इस मेथड द्वारा उपयोग किए जाने वाले सभी parameters का विवरण इस प्रकार है—

- **callback** – कार्य Execute करने के लिए।

उदाहरण

मान लें कि हमारे पास परिणाम HTML फ़ाइल में HTML कॉन्टेंट है—

```
<h1>THIS IS RESULT...</h1>
```

निम्नलिखित एक सरल उदाहरण है जो इस मेथड के उपयोग को दर्शाता है।

```
<html>
  <head>
    <title>The jQuery Example</title>
    <script type = "text/javascript"
      src = "https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js">
    </script>

    <script type = "text/javascript" language = "javascript">
      $(document).ready(function() {

        /* Global variable */
        var count = 0;

        $("#driver").click(function(event){
          $('#stage0').load('result.html');
        });

        /* Gets called when request starts */
        $(document).ajaxStart(function(){
          count++;
          $("#stage1").html("<h1>Starts, Count :" + count + "</h1>");
        });

        /* Gets called when request is sent */
        $(document).ajaxSend(function(evt, req, set){
          count++;
          $("#stage2").html("<h1>Sends, Count :" + count + "</h1>");
          $("#stage2").append("<h1>URL :" + set.url  + "</h1>");
        });

        /* Gets called when request complete */
        $(document).ajaxComplete(function(event,request,settings){
          count++;
          $("#stage3").html("<h1>Completes, Count :" + count + "</h1>");
        });

        /* Gets called when all requests are ended */
        $(document).ajaxStop(function(event,request,settings){
          count++;
          $("#stage4").html("<h1>Stops, Count :" + count + "</h1>");
        });

      });
    </script>
  </head>

  <body>
    <p>Click on the button to load result.html file:</p>
```

```
<div id = "stage0" style = "background-color:blue;">
  STAGE - 0
</div>

<div id = "stage1" style = "background-color:blue;">
  STAGE - 1
</div>

<div id = "stage2" style="background-color:blue;">
  STAGE - 2
</div>

<div id ="stage3" style = "background-color:blue;">
  STAGE - 3
</div>

<div id = "stage4" style = "background-color:blue;">
  STAGE - 4
</div>

<input type = "button" id = "driver" value = "Load Data" />

</body>
```

6. AJAX रिक्वेस्ट सफलतापूर्वक पूरा होने पर ajaxSuccess(callback) मेथड एक फ़ंक्शन को Execute करने के लिए संलग्न (attach) करती है। यह एक Ajax इवेंट है।

सिंटेक्स

यहाँ इस मेथड का उपयोग करने के लिए सरल सिंटेक्स है—

```
$(document).ajaxSuccess( callback )
```

पैरामीटर

इस मेथड द्वारा उपयोग किए जाने वाले सभी parameters का विवरण इस प्रकार है—

- **callback** – कार्य Execute करने के लिए। इवेंट ऑब्जेक्ट, XMLHttpRequest, और उस रिक्वेस्ट के लिए उपयोग की जाने वाली सेटिंग्स कॉलबैक के तर्क के रूप में पारित की जाती हैं।

उदाहरण

मान लें कि हमारे पास परिणाम HTML फ़ाइल में HTML कॉन्टेंट है—

```
<h1>THIS IS RESULT...</h1>
```

निम्नलिखित एक सरल उदाहरण है जो इस मेथड के उपयोग को दर्शाता है।

```
<html>
  <head>
    <title>The jQuery Example</title>
    <script type = "text/javascript"
      src = "https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js">
    </script>

    <script type = "text/javascript" language = "javascript">
      $(document).ready(function() {
```

```
    /* Global variable */
    var count = 0;

    $("#driver").click(function(event){
      $('#stage0').load('result.html');
    });

    /* Gets called when request starts */
    $(document).ajaxStart(function(){
      count++;
      $("#stage1").html("<h1>Starts, Count :" + count + "</h1>");
    });

    /* Gets called when request is sent */
    $(document).ajaxSend(function(evt, req, set){
      count++;
      $("#stage2").html("<h1>Sends, Count :" + count + "</h1>");
      $("#stage2").append("<h1>URL :" + set.url  + "</h1>");
    });

    /* Gets called when request completes */
    $(document).ajaxComplete(function(event,request,settings){
      count++;
      $("#stage3").html("<h1>Completes,Count:" + count + "</h1>");
    });

    /* Gets called when request is stopped */
    $(document).ajaxStop(function(event,request,settings){
      count++;
      $("#stage4").html("<h1>Stops, Count :" + count + "</h1>");
    });

    /* Gets called when all request completes successfully */
    $(document).ajaxSuccess(function(event,request,settings){
      count++;
      $("#stage5").html("<h1>Success,Count :" + count + "</h1>");
    });

  });
 </script>
</head>

<body>
 <p>Click on the button to load result.html file:</p>

 <div id = "stage0" style = "background-color:blue;">
   STAGE - 0
 </div>
```

```
    <div id = "stage1" style = "background-color:blue;">
      STAGE - 1
    </div>

    <div id = "stage2" style = "background-color:blue;">
      STAGE - 2
    </div>

    <div id = "stage3" style = "background-color:blue;">
      STAGE - 3
    </div>

    <div id = "stage4" style = "background-color:blue;">
      STAGE - 4
    </div>

    <div id = "stage5" style = "background-color:blue;">
      STAGE - 5
    </div>

    <input type = "button" id = "driver" value="Load Data" />
  </body>
</html>
```

अनु क्रमांक	मेथड और विवरण
1	**ajaxComplete(callback)** जब भी AJAX रिक्वेस्ट पूरा होता है, तो किसी फ़ंक्शन को Execute करता है।
2	**ajaxStart(callback)** जब भी AJAX रिक्वेस्ट शुरू होता है और कोई भी सक्रिय नहीं होता है, तब किसी कार्य को Execute किया जाता है।
3	**ajaxError(callback)** जब भी AJAX रिक्वेस्ट विफल होता है, तो किसी फ़ंक्शन को Execute करता है।
4	**ajaxSend(callback)** AJAX रिक्वेस्ट भेजे जाने से पहले Execute किए जाने वाले फ़ंक्शन को संलग्न (attach) करता है।
5	**ajaxStop(callback)** जब भी सभी AJAX रिक्वेस्ट समाप्त हो गए हों, तब किसी कार्य को Execute करता है।
6	**ajaxSuccess(callback)** जब भी AJAX रिक्वेस्ट सफलतापूर्वक पूरा होता है, तो किसी फ़ंक्शन को Execute करता है।