xhr.open(“POST”, “products.php”, true);

xhr.setRequestHeader(“Content

-

Type”, “application/x

-

www

-

form

-

urlencoded”);

xhr.onreadystatechange = function(){

if(this.readyState == 4 && this.status == 200){

var hdrDate = this.getResponseHeader(“Date”);

var

xml = this.responseXML;

}

}

xhr.send(“productID=1234&catID=5678”);

•

Cross

-

Browser and Old Browser Support

var xhr;

if(typeof XMLHttpRequest != “undefined”){

xhr = new XMLHttpRequest();

}else if (window.ActiveXObject){

var activeXIDs

= [“MSXML2.XmlHttp.5.0”,

“MSXML2.XmlHttp.4.0”,

“MSXML2.XmlHttp.3.0”,

“MSXML2.XmlHttp”,

“Microsoft.XmlHttp”];

for(var i=0; i<activeXIDs.length;i++){

try{

xhr = new ActiveXObject(activeXIDs[i]);

}catch(e){}

}

}

•

Three important p

roperties of the

XMLHttpRequest

Object

o

onreadystatechange

♣

Stores a function to be called automatically each time

readyState

changes

o

readyState

♣

0 :

request not initialized

♣

1 : server connection established

♣

2 : request received

♣

3 : processing request

♣

4 :

request finished and response is ready

o

status

♣

200 :

OK

♣

404 :

Not Found

Document Object Model

•

Document Object Collections

o

anchors[]

o

forms[]

o

images[]

o

links[]

•

Document Object Properties

o

c

ookie

o

documentMode

o

domain

o

lastModified

o

readyState

o

referrer

o

title

o

URL

•

Document Object Methods

o

close()

o

getElementById()

o

getElementsByName()

o

getElementsByTagName()

o

open()

o

write()

o

writeln()

Java EE

•

Java EE is an open, standards

-

based development and deployment platform for creating

distributed, transactional, reliable, secure, multi

-

tiered, web

-

based, server

-

centric, component

-

based enterprise applications

•

Java EE Application Model

o

Java programming language, Java Virtual Machine (JVM)

o

Java EE Components

♣

Java EE Clients

•

Application Cl

ients, Applets (embedded in web clients)

♣

Web Components

•

Servlets, JavaServer Pages (JSP), JavaServer Faces (JSF)

♣

Enterprise JavaBeans (EJBM)

o

Java EE Containers

♣

Client containers, web container, EJB container

o

Java EE Server

•

Java EE Web Application

o

Collection of resources installed under a specific subset of the URL namespace of a web application server compliant with the

Java

EE Specification (e.g. Apache‟s Tomcat, Apache‟s Geronimo, Sun Microsystems‟ Glassfish, IBM‟s WebSphere, etc.)

o

Resources

♣

Stat

ic resources: web pages, images, stylesheets, etc.

(serves as is)

♣

Dynamic resources: servlets, JSPs

♣

Miscellaneous resources: business object classes (e.g. Java Beans, EJB), support libraries, etc.

o

XML

-

formatted descriptor and configuration files

♣

web.xml, a

pplication.xml, context.xml, etc.

o

Organized into a standard hierarchical structure and typically packaged and deployed as WAR or EAR files

•

Java EE APIs

o

Enterprise JavaBeans Technology

o

Java Servlet Technology

o

Java Server Pages

o

Java Server Pages Standard Tag

Library

o

Java Server Faces

o

Java Msg Service API

o

Java Transaction API

o

JavaMail API

o

JavaBeans Activation Framework

o

Java API for XML Processing

Servlets

•

Java Object based on the Servlet API

•

Runs in a server application to answer client requests; technically,

servlets are not tied to a specific client

-

server protocol, but they are most

commonly used with HTTP and the term „servlet‟ is often used in the context of an “HTTP Servlet”

•

Web

-

tier components in the Java EE architecture.

•

Runs in, and is managed by, a w

eb

-

tier container called the „Servlet Container‟

•

Mapped to URLs to which clients send requests

•

Typically asked with (among other things)

o

Processing and/or storing data submitted vial HTML forms

o

Generating dynamic content

•

javax.servlet

o

Servlet, GenericServl

et

o

ServletRequest, ServletResponse

o

ServletConfig, ServletContext

o

RequestDispatcher

•

javax.servlet.http

o

HttpServlet

o

HttpServletRequest

o

HttpServletResponse

o

HttpSession

o

Cookie

•

Servlet Processing

o

Client sends a request to a web server URL that is mapped to a

servlet. Web server passes on the request to the servlet container

o

Servlet container checks if servlet is already loaded

o

If it is not yet loaded, servlet container loads the servlet class and instantiates the servlet, and calls its

init

method.

o

Servlet con

tainer invokes the servlet‟s

service

method, passing request and response objects as arguments

o

Servlet processes the request using the response object to create the response, which is returned by the servlet container to

the

web server, which in turn sends

the response to the client

o

Subsequent request to the servlet will not require servlet re

-

instantiation, unless the servlet has been unloaded; before a servlet is

unloaded, the servlet container invokes its

destroy

method.

•

init(config)

o

Invoked once on the

servlet by the servlet container when the servlet is instantiated; can be used by the servlet for one

-

time startup

initialization

•

service(request, response)

o

Invoked each time the servlet is called upon to process a request (typically on a separate thread f

or each call)

o

In

HttpServlet

, the default Service implementation maps the call to a specific doXXX() method (e.g.

doGet

,

doPost

) which is

typically overridden to affect the servlet‟s functionality

•

Destroy()

o

Invoked on the servlet by the servlet container w

hen the servlet is to be unloaded (e.g. when the application is stopped or

undeployed); can be used by the servlet for clean

-

up processing (e.g. resource deallocation)

•

Servlet Request Processing (

HttpServletRequest

)

o

Retrieving user

-

supplied request paramet

ers

o

Retrieving request header values

•

Servlet Response Processing (

HttpServletResponse

)

o

Setting response status code

o

Setting response headers

o

Obtaining output object for sending the response

•

Servlet Request Dispatching (

RequestDispatcher

)

o

Obtain a

RequestDi

spatcher

to a resource (static or dynamic) from the request object

RequestDispatcher rqstDsp;

rqstDsp = request.getRequestDispatcher(res);

o

Include the dispatcher resource (or its output) in the current response; one or more resources can be included

(e.g. use for banners,

footers, etc.)

rqstDsp.include(request, response);

o

Forwards the processing of the current request to the dispatcher resource; the servlet processing the current request must no

t

generate a response ( e.g. use in MVC “controller”

servlets)

rqstDsp.forward(request, response);

•

Session Tracking

(HttpSession)

o

Session tracking support is implemented either cookies or URL

-

rewriting

o

Obtaining session object from the current request

HttpSession session;

s

ession = request.getSession(createNe

w);

o

Obtaining session information

(HttpSession)

♣

getCreationTime(), getLastAccessedTime(), getMaxInactiveInternal(), getId(), isNew(),

setMaxInactiveInterval(

int

val)

o

destroying sessions

♣

invalidate()

o

URL

-

rewriting

(HttpServletResponse)

♣

encodeURL(

String

url

), encodeRedirectURL(

String

url)

Web Context (

ServletContext

)

•

a web application is associated with a context, which is an object that provides methods that servlets use to communicate wit

h the servlet

container

•

obtaining the servlet context (

HttpServlet

)

ServletContext context;

context = this.getServletContext();

•

obtaining context information (

ServletContext

)

o

getServerInfo

()

, getContextPath

()

, getRealPath

()

, getResource

()

, getResourceAsStream

()

,

getMimeType, getInitParameter

()

, getInitParameterNames

()

, ge

tRequestDispatcher

()

, getContext

()

Servlet Configuration (

ServletConfig

)

•

getServletName(), getServletContext(), getInitParameter(), getInitParameterNames()

Information sharing using scope objects

•

A request may be processed by several web application compon

ents (e.g. through calls to

RequestDispatcher

forwad

/

include

) and

there may be a need for one component to communicate information to the other components in the request processing chain.

•

A client session typically consists of multiple requests, which due

to the stateless nature of HTTP, will appear to the application as being

“unrelated” to one another; the

HttpSession

object can be used to “relate” these requests together, but there may still be a need to share

information created in one request with a su

bsequent request

•

Different web application components may require access to common resources or information (e.g. page counters, shared databa

se

connection).

•

Information sharing is accomplished by creating attribute objects and exposing these objects in th

e appropriate scope.

•

Scopes:

o

Request scope (

HttpServletRequest

)

o

Session

s

cope (

HttpSession

)

o

Web Application or Web Context scope (

ServletContext

)

o

Page

s

cope (local objects in a servlet)

•

Creating

,

accessing, and removing attribute objects

o

setAttribute

(String attrName,

Object

attrValue)

o

getAttribute (String attrName)

o

getAttributeNames()

o

removeAttribute (String attrName)

Advanced Servlet Topics

Listeners

•

java objects used to “subscribe” to application “events” in order to be “notified” when these event

s occur

o

context

-

related events

♣

context initialized, context destroyed, context attribute changes

o

session

-

related events

♣

session created, session destroyed, session attribute changes

o

request

-

related events

♣

request initialized, request destroyed, request att

ribute changes

javax.servlet

•

ServletContextListener, ServletContextAttributelistener

•

ServletRequestListener, ServletRequestAttributeListener

javax.servlet.http

•

HttpSessionListener, HttpSessionAttributeListener

Filters

•

Java objects used to intercept

incoming requests and outgoing responses in order to perform various tasks such as:

o

Authentication and access control

o

Logging, auditing

o

Caching, data compression

o

Content Transformation

•

Filter objects are mapped to the URL patterns they are intended to inte

rcept

•

Filter objects can be “chained” together

javax.servlet

•

Filter, FilterChain, FilterConfig

Java Server Pages

•

Simply an HTML web page that contains additional bits of code that execute application logic to generate dynamic content.

•

Java Server Pages Ac

tions (JSP tags) perform a variety of functions and extend the capabilities of JSP.

•

Java Server Pages Actions use XML

-

like syntax, and are used to manage JavaBeans component.

•

Directives are instructions that are processed by the JSP engine when the page is

compiled to a servlet.

•

Directives are used to set page

-

level instructions, insert data from external files, and specify custom tag libraries

o

<%@ %>

•

Motivation

o

It is typically a good idea to separate business logic from presentation concern

♣

Allows

modern web development teams to be divided up into programmers and web page authors / designers

♣

Fosters component reuse (e.g. the same data object can be consumed by user agents of varying capabilities and needs)

o

Servlets can be very powerful for programm

ing business logic, but are very awkward to use when generating static (i.e. template)

content.

o

(X)HTML marked

-

up documents are very convenient for static content generation but cannot be used to program business logic (or

generate dynamic content arising

from data produced by the business logic).

•

Features

o

Text

-

based document capable of generating both static and dynamic content (typically intermixed)

o

Mark

-

up based document syntax (JSP

-

style or XML

-

style), combining (X) HTML elements as well as standard and

custom JSP

elements; thus, web page authors can feel right “at home” with the mark

-

up syntax.

o

Embedded Java Coding support via “scriptlets”

♣

<%

%>

o

Template text are converted into JSPWriter

•

Components

o

Template (i.e. static) text

o

JSP elements

♣

Directives

•

<

%@ page ContentType=”text/html” pageEncoding=”UTF

-

8” %>

•

<%@ page import=”java.util.Random” %>

o

autoFlush

o

buffer

o

contentType

o

errorPage

o

extends

o

import

o

info

o

isELIgnored

o

isErrorPage

o

isThreadSafe

o

language

o

pageEncoding

o

session

•

<%@ taglib uri=”http://java.sun.com/

jsp/jstl/core”

prefix=”c” %>

o

prefix

o

taglib

o

uri

♣

Scripting Elements

•

Declarations

o

<%! int a = 100; %>

o

<%! int square(int n) { return n\*n ; } %>

•

Expressions

o

<% String s = new java.util.Date().toString(); %>

•

Scriptlets

o

<% for(int i =

0; i < 10 ; i++) {

out.println(i); } %>

♣

Actions

•

Standard actions

o

<jsp:

directive.include>, <jsp:directive.page>

o

<jsp:

declarations>

o

<jsp:scriptlet>

o

<jsp:expression>

o

<jsp:

include>, <jsp:

forward>

o

<jsp:

useBean>, <jsp:setProperty>, <jsp:

getProperty>

o

<jsp:

plugin>, <jsp:param>, <

jsp:params>, <jsp:

fallback>

o

<jsp:

element>, <jsp:

attribute>, <jsp

:

body>

o

<jsp:text>

•

Custom Actions (JSTL)

o

JSTL, user

-

written custom tag libraries

•

Expression Language (EL)

o

${

}

•

Implicit Scripting Objects

o

request, response, out, pageContext

o

session,

pageContext

, application

o

config, page, exception

•

Implicit EL Objects

o

pageContext

o

pageScope

o

requestScope

o

sessionScope

o

applicationScope

o

param, paramValues

o

header, headervalues

o

cookie

o

initparam

•

Comments

o

<%

--

this is a JSP comment

--

%>

Servlets or JSPs?

•

The

common practice is to leverage both technologies to

implement the MVC design pattern

Model

-

View

-

Controller (MVC) Design Pattern

•

Model

o

Represents business objects (logic and state)

•

View

o

Presentation of the model in some appropriate

way

•

Controller

o

Mediates a

pplication flow

A sample web MVC framework can use

•

JavaBeans for the model

•

JSPs for the View

•

Servlets for the controller

JSP Standard Tag Library (JSTL)

•

Set of custom JSP elements that provide various

programmatic functionality via markup syntax

o

Core Tag L

ibrary

♣

variable support, flow control, URL

management

o

SQL Tag Library

♣

Database connections, queries, updates

o

Internationalization Tag Library

♣

Locate setting, message bundling,

number formatting, date formatting

o

XML

♣

Core XML processing, flow control,

transf

ormation

o

JSTL Function

♣

String functions, collection lengths

•

In addition to the JSTL, developers can also create their own

tag libraries for commonly occurring tasks

ACRONYMS:

Adobe AIR (Adobe Integrated

Runtime)

AJAX (Asynchronous JavaScript and XML)

ANSI (American National Standards Institute)

ASP (Active Server Pages)

BOM (Browser Object Model)

CFML (ColdFusion Markup Language)

CGI (Common Gateway Interface)

DHTML (Dynamic HTML)

DOM

(

Document Object

Model)

DTD (Document Type Definition)

EAR (Enterprise Archive)

EIS (Enterprise Information Systems)

EJB (Enterprise JavaBeans)

EL (Expression Language)

IIS (Internet Information Services)

JAR (Java ARchive)

JSF (Java Server Faces)

JSON (JavaScript Object

Notation)

JSP (Java Server Pages)

JSTL (JSP Standard Tag Libraries)

PERL (Practical Extraction and Reporting Language)

PHP (Hypertext Preprocessor)

RDF (Resource Description Framework)

RIA (Rich Internet Application)

RSS (Really Simple Syndication)

SMIL (

Synchronized Multimedia Integration Language)

SMX (Server Macro Expansion)

SOAP (Simple Object Access Protocol)

SVG (Scalable Vector Graphics)

WAP (Wireless Application Protocol)

WAR (Web Application Archive)

WSDL (Web Services Description Language)

XForms

(XML Forms)

XLink (XML Linking Language)

XPointer (XML Pointer Language)

XQuery (XML Query Language)

XSD (XML Schema)

XSL (Extensible Style Sheet Language)

XSL

-

FO (Extensible Style Sheet Language Formatting Objects)

XSLT (XSL Transformations)

XSRF

(Cross

-

site request forgery)

XSS (Cross

-

site scripting