Jspiders
Training & Development Center

Author:-Shishira Bhat
Technical Architect

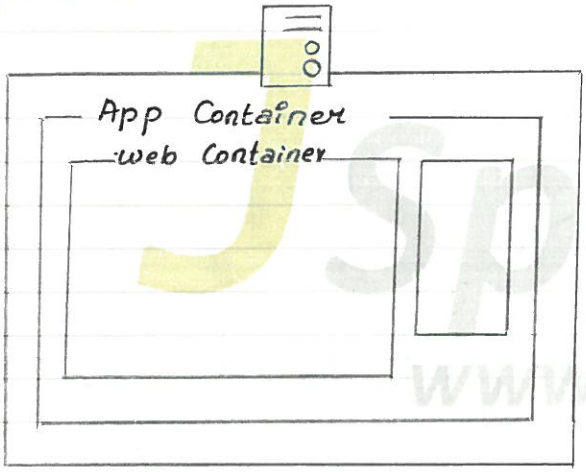
Server Side Programming

Servers :-

→ A server is the one which serves client requests.
→ For a server to server a client request, resource must be available.
→ Server does not creat or generat any resources on its own
→ 3 types of servers

1. Web Servers :
   Ex:- Apache tomcat, Sun / oracle / glassFish , Jetty
2. DB Servers :
   Ex: Oracle , MySQL, Sybase , IBM DB2, Derby etc
3. Application Servers:
   Ex:- JBoss, BEA / Oracle Weblogic , IBM webshere , Oracle app server

App Server and Web Server :-



App Server

Web Server

* To run a J2EE application the server must have J2EE container. ie we cannot simply run J2EE app in all available servers.

Resource Type or MIME Type [Multipur Internet Mail Extensions] ;

→ Every resource that we want to deploy has its own type which is called as resource type.

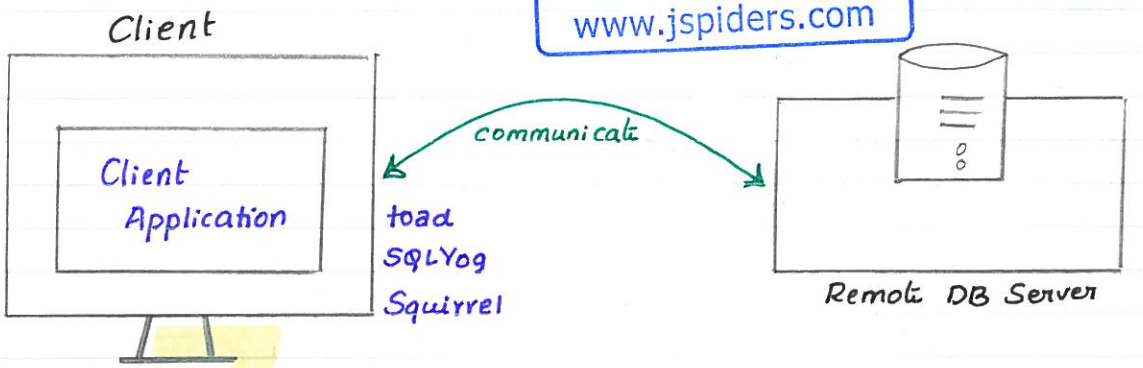
Ex: text / html
text / xml
image / jpeg
application / jar
application / pdf
video / quicktime

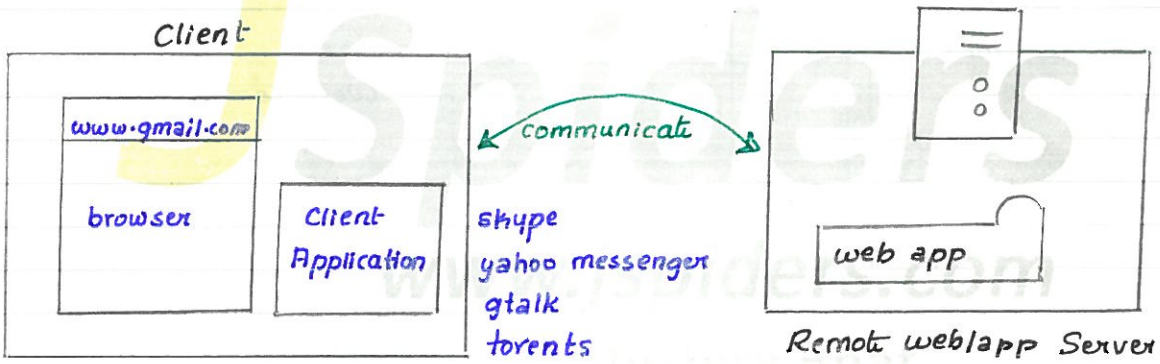
→ we can deploy multiple applications into a single server ie a server is capable of running many applications at a same time.
→ When we deploy multiple apps into a single server, each app must have unique name or context or root.

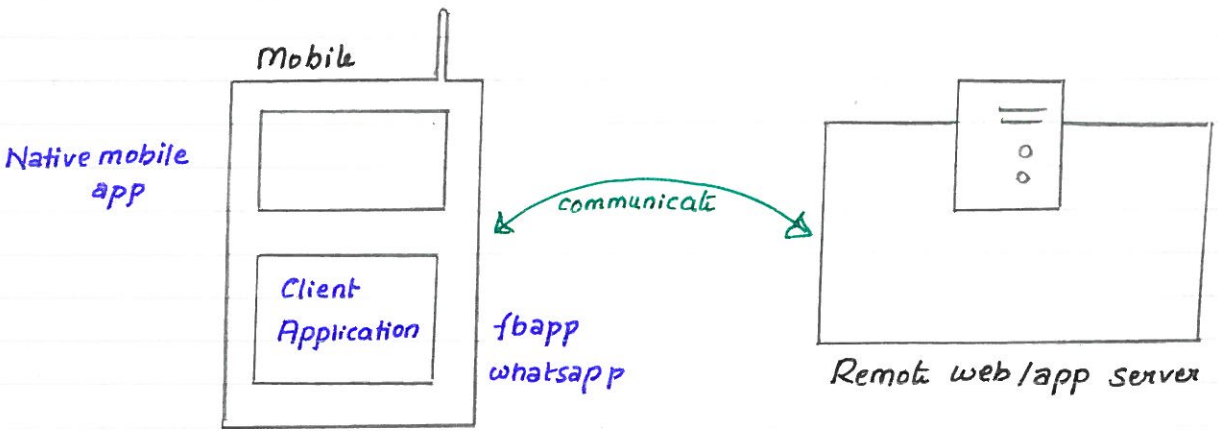
→ The process of making the resource available to server is called as deployment.

Client Application :

→ A client application is a small lightweight application which must be installed onto client system (client computer) using which we can communicate the remote or the local server.







URL (Uniform Resource Locator):

→ URL is the one using which we access the remote application.
→ If it is web based application then we use http protocol.
→ URL is used to identify or locate the resource over the web.
→ URL contains

Author:-Shishira Bhat
Technical Architect

a. Resource Path
b. Data [optional]

Resource Path ? Data k1 = v1 & k2 = v2 & k3 = v3

Left hand side ← → Right hand side.

Data :

→ In serverside application data is submitted to client to server in the form of key and value pair which is also called as associated values.
→ Data is optional because the possibility is that the html page may not contain any data fields.
→ The data which we enter in the form or page is called as formdata or UI Data.
→ Every html field in a page must have a unique identity which can be given by using name or id attribute

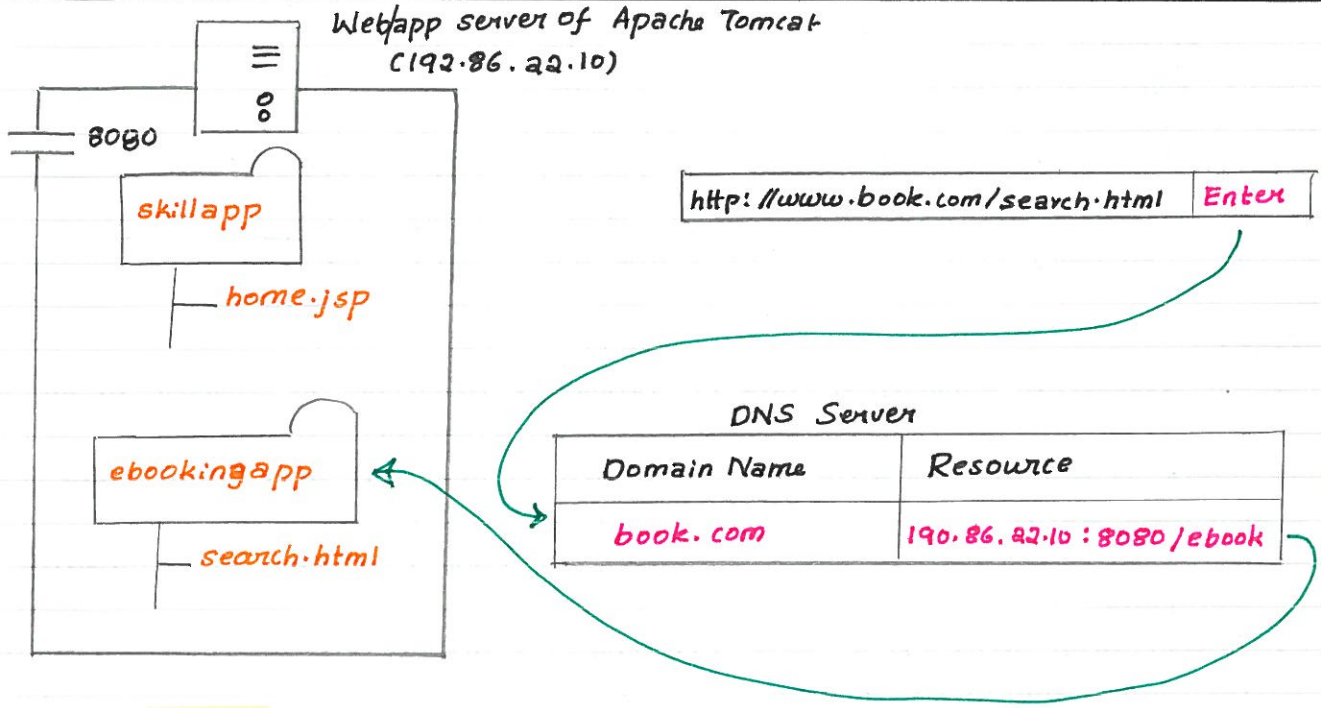
Ex:- 1. `<input type="text" name="nm">`
2. Address : `<textarea name="adr" rows="14" cols="15"></textarea>`
3. Gender
Male : `<input type="radio" name="gdr" value="male">` (unique identifier)
Female : `<input type="radio" name="gdr" value="Female">`
(name = key, value = value)

nm ← Name : Shishira
pn ← Contact : 9980517008
em ← email : imshishiva@gmail.com
Register

→ Multiple data is separated by & and resource path & data is separated by ?
Ex:- nm = shishira & pn = 9980517008 & email = imshishira@gmail.com
k1 = v1 & k2 = v2 & k3 = V3

Resource Path :

→ It locates or identifies resource uniquely over the web.
→ Resource path contains
a. type of control
b. domain name or (IP address + app name + port)
c. Resource name



http://190.86.22.10:8080/skillapp/home.jsp
http://190.86.22.10:8080/ebookingapp/search.html
Domain

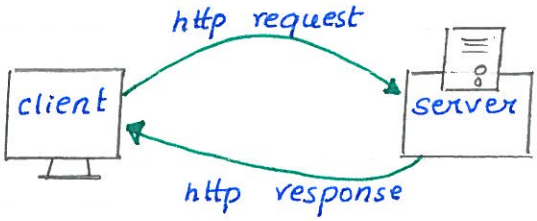
http://www.skill.com/home.jsp
http://www.book.com/search.html

https://www.facebook.com/confirm.php?id=sabhat&pn=9908517008&nm=abc
Resource Path ⟵ ⟶ Data [K=V]

HTTP (Hyper Text Transfer Protocol) :

→ Http is the protocol which is used to communicate with web application ie it defines the set of rules to communicate with web app which may be hosted in a local or remote server.

→ To complete one cycle we need http request and http response.



HTTP Request :-

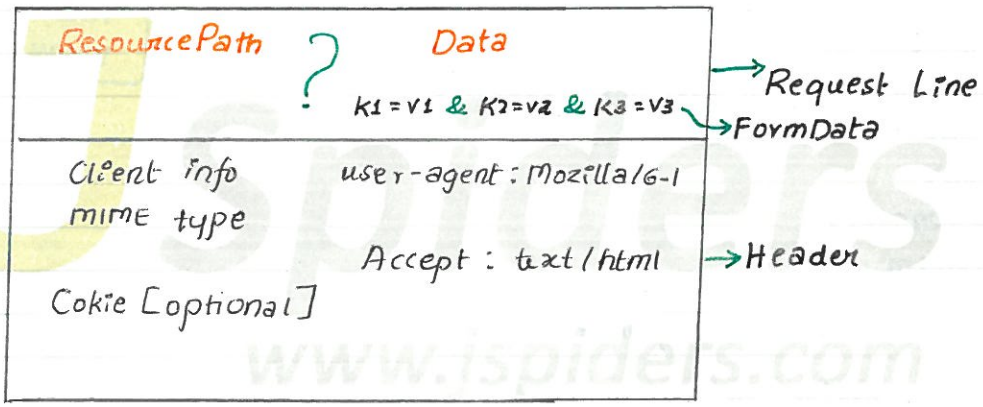
→ Below are the content type of the http request-

1. URL

2. Client info
3. Cookie (Optional)
4. Form / UI Data

→ There are 8 different types of http requests
a. get
b. post
c. delete
d. put
e. trace
f. options
g. head
h. connect

HTTP Get Request:

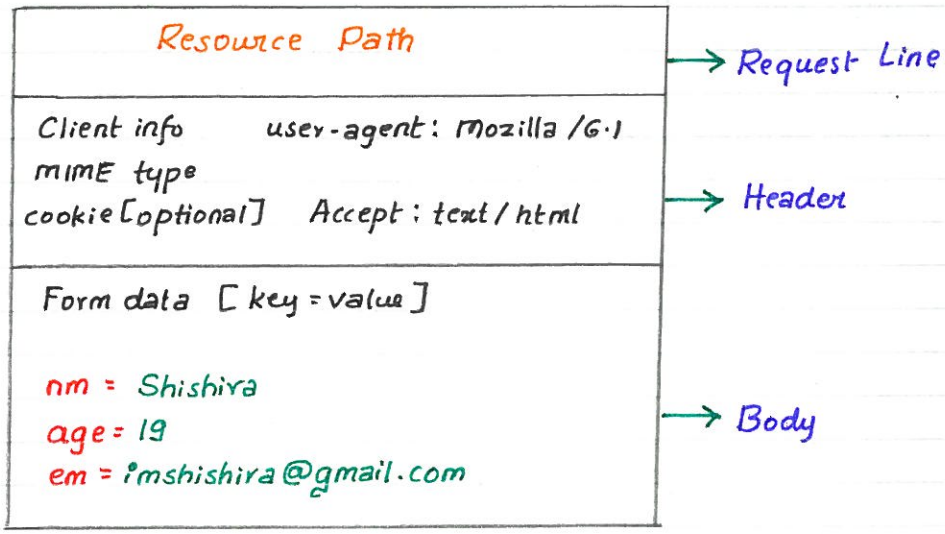


Get Request

→ As the name indicates GET request is used to get the response from the server.
Ex:- When we click on the hyperlink we get response from the server
→ GET is not generally used to deal with the data but still we can pass 256 characters of data ie limited data can be passed by using GET request.
→ GET carries all the limited form data in the URL. Hence it is not secured. [Data is exposed in URL]
→ GET request can be bookmarked.

HTTP POST Request:
→ Http post request is used to post the data to the server.
→ Here any no.of data can be posted or sent to the server.
→ All the form or UI data travel in the body part which is not exposed in the URL
→ Hence it is secured.
→ Post request cannot be bookmarked.

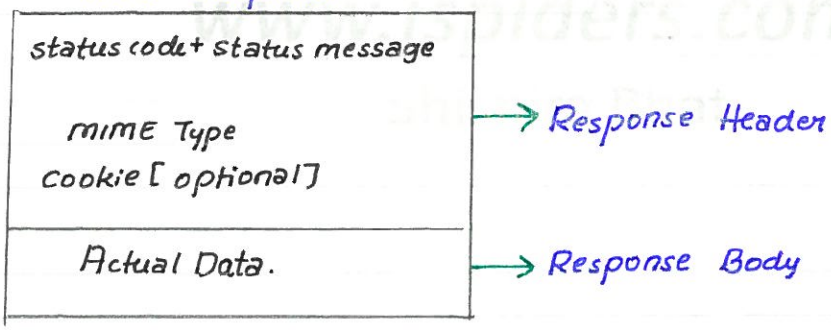
POST Request



HTTP Response :

→ Below are the contents of the http response
1. status code + status message
2. MIME Type
3. cookie [optional]
4. actual content.

HTTP Response



→ The status code, status message and cookie are automatically generated by server.

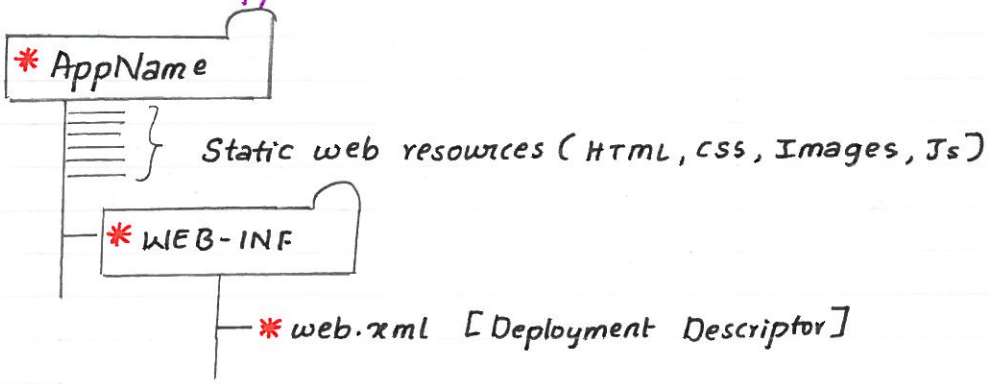
→ Examples of status code

Ex:- 2xx - 200 Successfully handled the request
4xx - 404 Resource not available
5xx - 500 Internal Server Error [may be wrong logic]
- 405 get-post or post-get mismatches.

→ The response body contains the actual content that the client has requested for. ie client is requested for an image, the response body must contain the actual image.

Author:-Shishira Bhat
Technical Architect

Structure of Static J2EE Applications :-



* Mandatory Fields

Configurable Resources :

→ There are some types of resources which has to be configured mandatorily
Ex: Servlets, Filters etc.

→ There are few types of resources which are not mandatory to configure but optionally we can configure.
Ex: Html page, JSP page etc.

→ The configurable resources has to be configured in web.xml

→ web.xml is also called as Deployment Descriptor.

→ web.xml at the net shell is the simple configuration file in the form of xml which generally contains information like

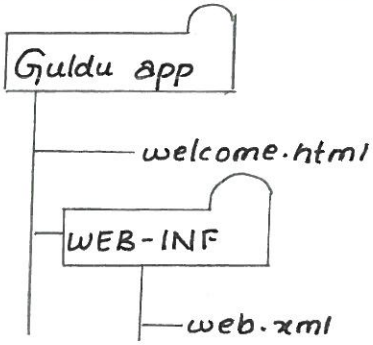
1. welcome file
2. Session timeout
3. Servlets.
4. JSP
5. Listeners
6. Filters
7. Security info and declarations etc.

→ One application must have one web.xml

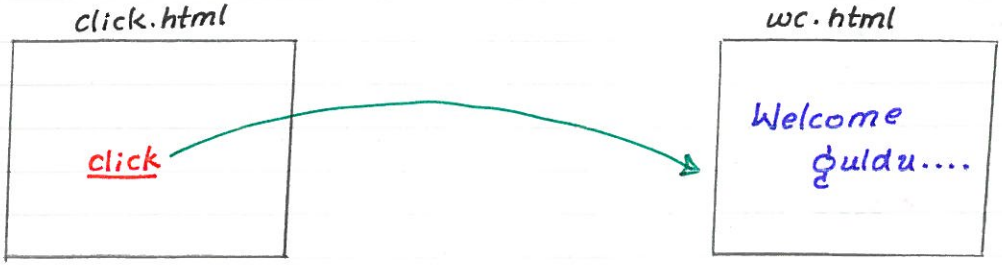
→ The first activity of server on startup is to read the web.xml file and incase if it has errors then server throws parse exception.

Note :

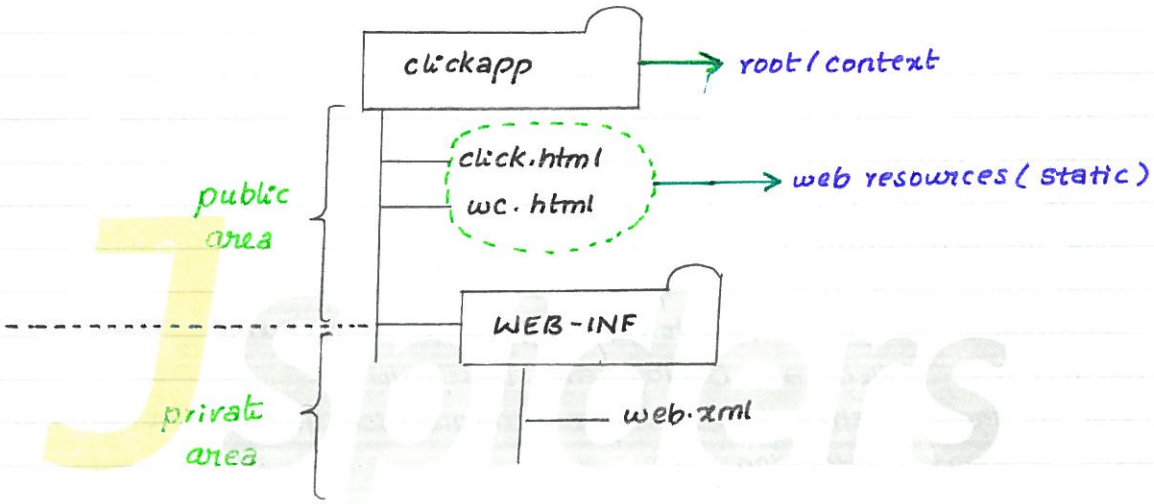
The latest versions of J2EE, it is not mandatory to use web.xml. Because it is Declarative [Annotation based]



Sample Example Application :-




```
<a href = "wc.html "> click </a>
```



→ It is not mandatory to keep all the web resources in public area
→ Optionally we can keep the web resources even in privat area. But in this case we need to configure the web resources (pages) in web.xml mandatorily.

Setting up of Apache Tomcat :

Prerequisites :

→ JRE must be present in the system.
→ Environment variables must be set.
  a. JAVA_HOME
  b. PATH
→ Apache Tomcat has 2 versions.
  1. exe file
  2. Compressed file (Zip file or tar file)

Path Names :

* JAVA_HOME :
  C: / Program Files / Java / jdk1.7.0-60

PATH :
C: \ program Files \ Java \ jdk1.7.0_60 \ bin

Author:-Shishira Bhat
Technical Architect

* CATALINA_HOME
  E:\SHISHIRA-BHAT\Apache-tomcat-7.0.37

  PATH:
  E:\SHISHIRA-BHAT\Apache-tomcat-7.0.37\bin

Folder Structure of Tomcat:

- bin
- conf
- lib
- logs
- temp
- webapps
- work

→ conf folder contains all the configurations related to tomcat in the form of xml
→ we can change the port number of tomcat inside the server.xml which is present in conf folder.
ie. CATALINA_HOME\conf\server.xml
<Connector port="8080">
→ The lib folder contains all the libraries in the form of jar file
Ex: servletapi.jar.
→ The log message generated by server is present (is stored) in the log files.
→ It is very important for a developer to analyse the log message in log file for application debugging.
→ The webapps folder stores all the deployed applications use.
→ We have to deploy the web apps into the webapps folder
→ webapps folder can contain one or more application
→ work is a folder where the actual processing of the webapp happens.

Types of Deployment:

1. Manual Deployment
2. Automated Deployment
   a. ant
   b. Maven

War Files:

→ war stands for web Archive ie it is a compressed format of web application
→ When we deploy the war file into the server, on startup of the server the war file will be extracted.
→ Command to creat a war file:

```
jar -cvf clickapp.war *.* WEB-INF
```

Author:-Shishira Bhat
Technical Architect

Servlets :-

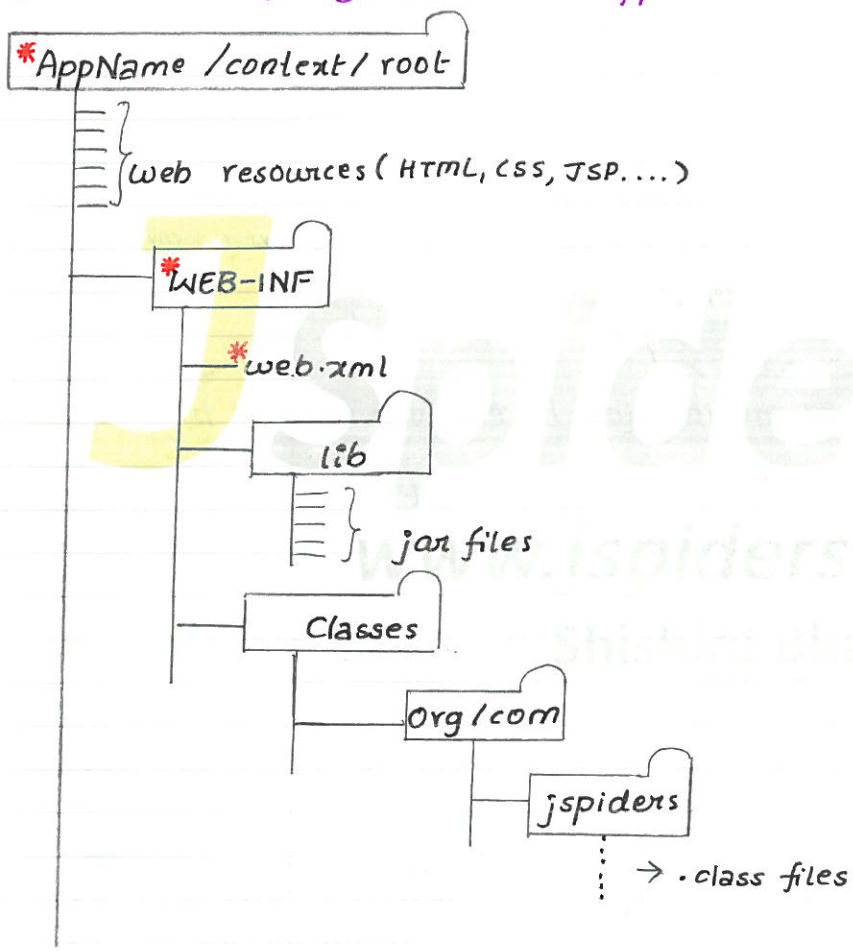
→ Servlet is a serverside java program which can communicate to http & performs different types of logic (business, controller, persistance and presentation) and responds to the client request.

→ The libraries which are required to develop the servlet program is present in ServletApi.jar file.

→ The jar file is present in CATALINA_HOME \ lib \ Servlet-api.jar.

→ We need to deploy the servlet.class to the server because server represent the runtime enviroment.

Directory Structure of Dynamic Web Application :-



Directory Structure of dynamic web app.

Welcome pages :-

→ When an end user access the application, then automatically a page must appear which is called as welcome page.

→ By default server looks for index.html as welcome file.

→ If we want the separate welcome file then we need to configure that in web.xml

Author:-Shishira Bhat
Technical Architect

Note :

When an web.xml is modified we need to restart the server in order to read web.xml again

Configuring welcomefile in web.xml :


```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
    <!-- configuring welcome file -->
      <welcome-file-list>
          <welcome file>send.html</welcome-file>
      </welcome-file-list>
</web-app>
```

→ we can configure multiple pages as welcome file in web.xml and in this case server looks for sequential priority.

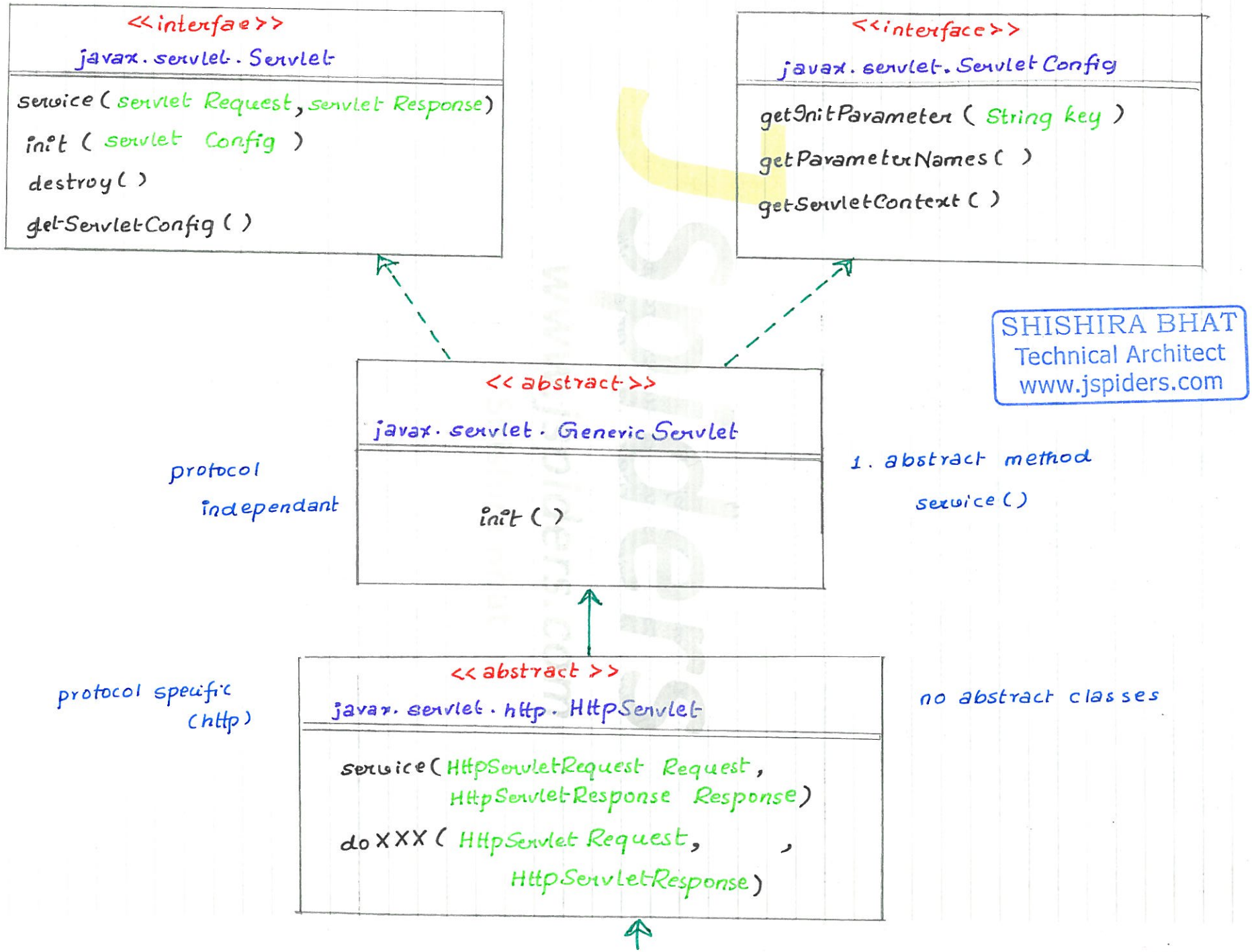
Setting up a development Environment :-

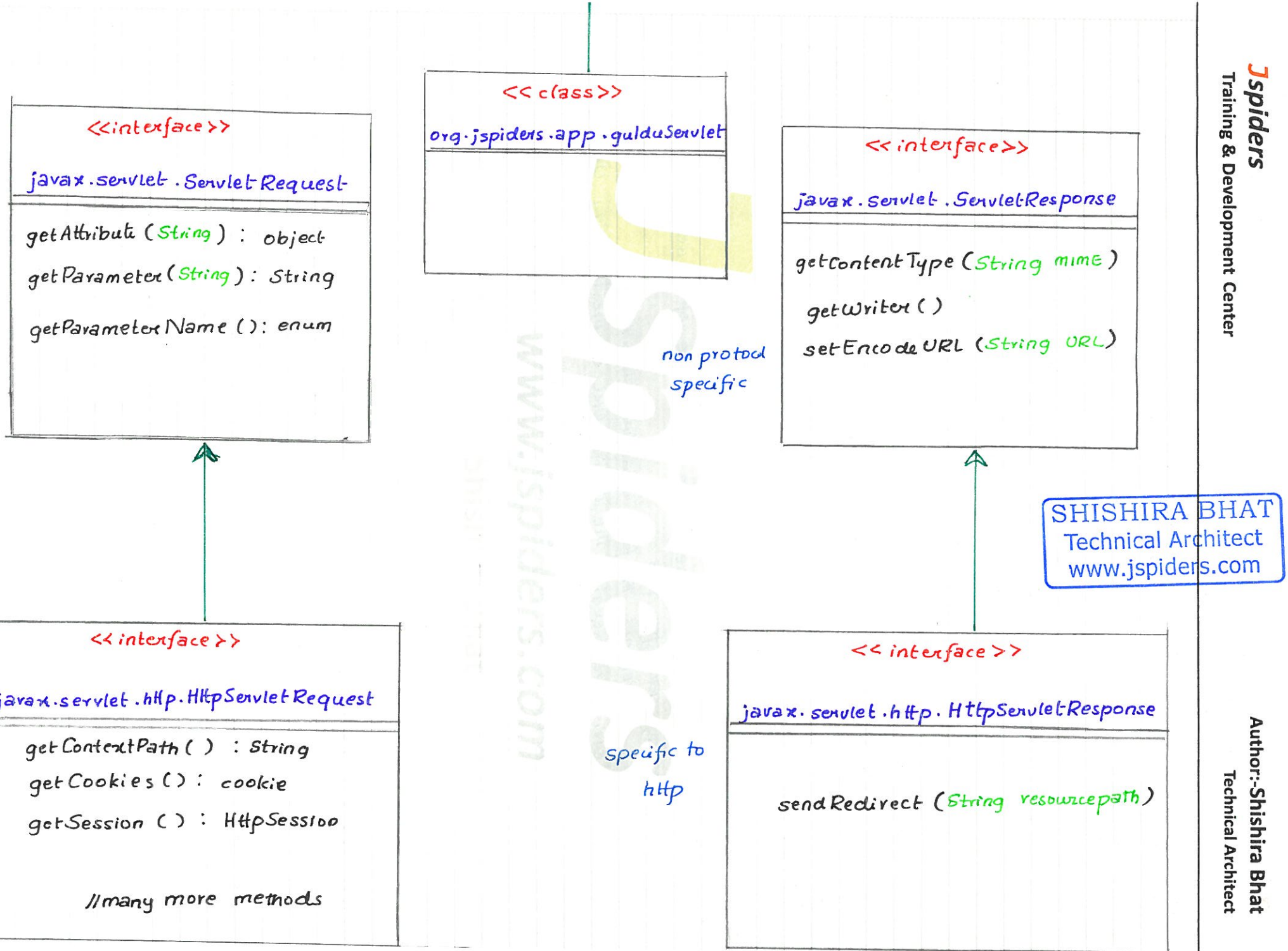
* Open eclipse
* Creat a new dynamic web project
* Creat appropriat package structure under src folder.
* copy all required libraries (JAR's) into project lib folder.
  Ex :- Driver jar file, servletapi.jar file
* Generat Deployment Descriptor (web.xml)
  Right click on project → java EE tools → generat Deployment Descriptor stub

Steps for creating a class :

1. Write a class which extends javax.servlet.GenericServlet which is an abstract class (Service() method is abstract)
2. Override service method.
3. Inside the service() write the logic or implementation.

Servlet Hierarchy





Author:-Shishira Bhat
Technical Architect

Configuring Servlet in web.xml :-

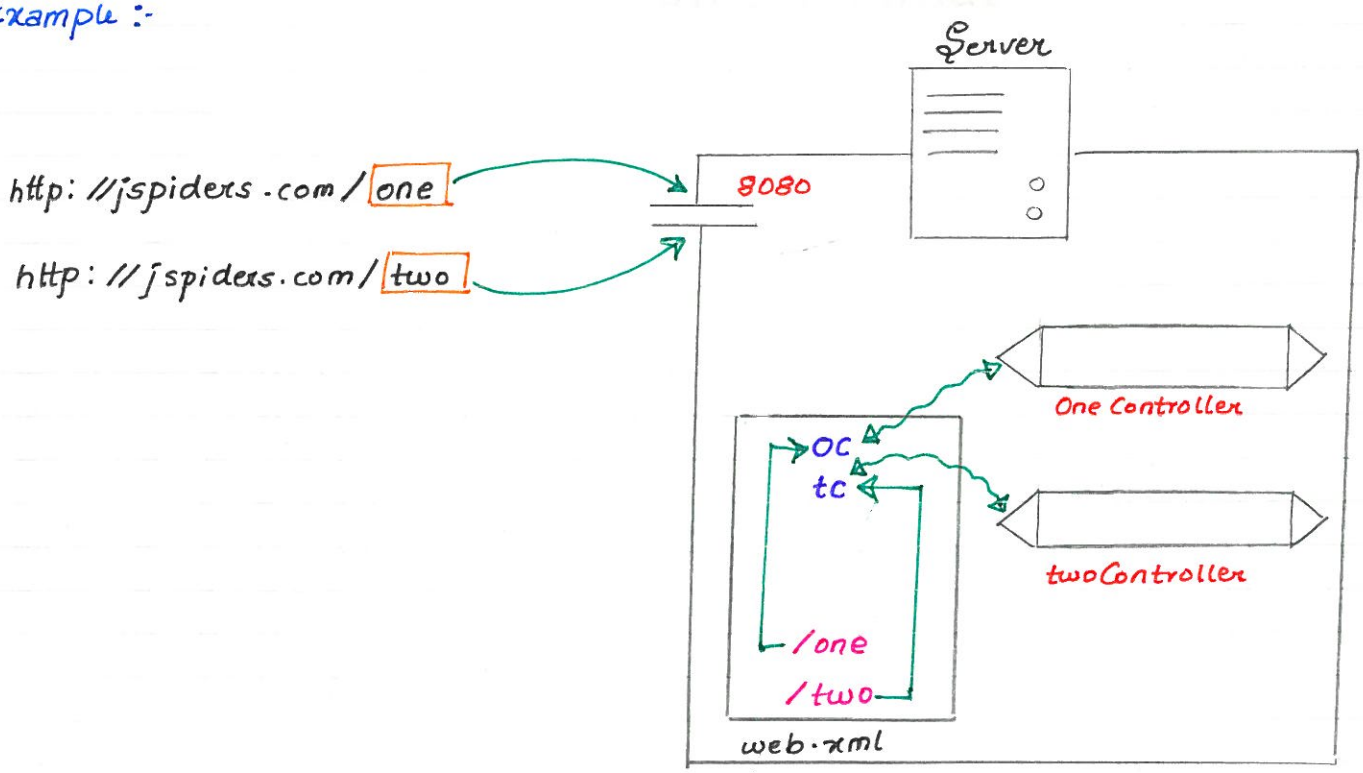
→ Whenever we write any servlet, we need to configure the servlet in web.xml

Ex :-

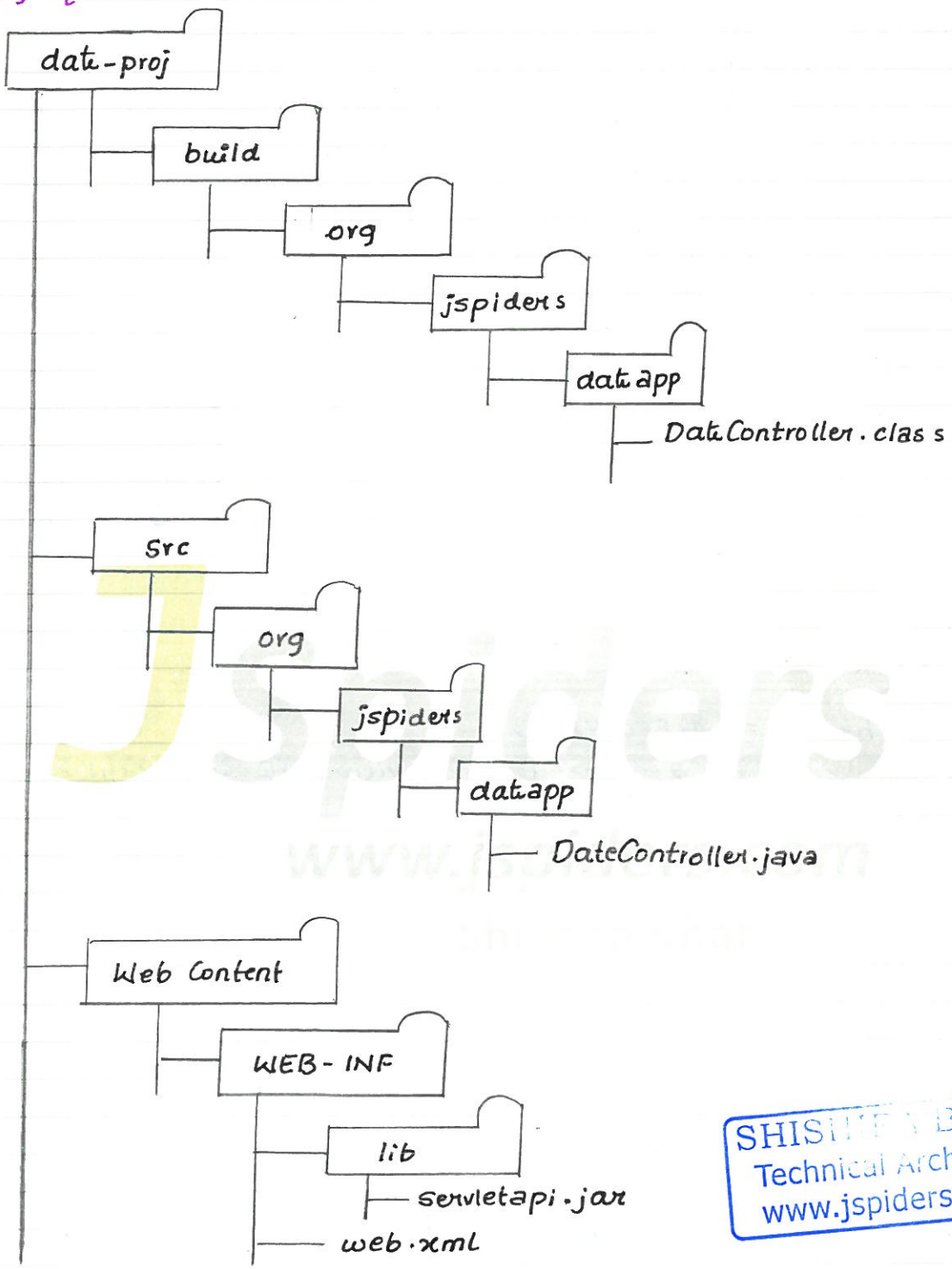
```
<?xml version="1.0" encoding="UTF-8" ?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <servlet>
        <servlet-name> dc </servlet-name>
        <servlet-class> org.jspiders.dateapp.DateController </servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name> dc </servlet-name>
        <url-pattern> /date </url-pattern>
    </servlet-mapping>

</web-app>
```


→ The servlet-name tag represent the alias name of a servlet
→ We can give any name for this.
→ Servlet class represents the fully qualified classname
→ The url-pattern represents the calling URL or the invoking URL of the servlet.
→ A servlet can be called only by using the URL pattern.
→ In web.xml, servlet name and URL-pattern must be unique.

Example :-



Directory Structure :-




```
// DateController.java
package org.jspiders.dataapp;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Date;
import javax.servlet.*;
public class DateController extends GenericServlet
{
```

```
@override
public void service (ServletRequest req, ServletResponse resp)
  throws ServletException, IOException
  {
    /* Impl / logic */
    Date d = new Date();
    String outputContent = "<html><body bgcolor="cyan">"+ " " +d+
          "</body></html>";
    PrintWriter pw = resp.getWriter();
    pw.println(outputContent);
    pw.flush();
    pw.close();
  }
}
```



```
// click.html
<html>
   <head>
      <title> Test </title>
   </head>
   <body bgcolor="#FFCCFF">
     <center><h1> <a href="Date"> view date </a> </h1> </center>
   </body>
</html>
```

```
// web.xml
<welcome-file-list>
   <welcome-file> click.html </welcome-file>
</welcome-file-list>
```

→ To call any servlet on performing some action or event in the html page, we need to mention the url pattern of servlet.

Ex:- `<h1> <a href="Date"> view date </a> </h1>`

`href="Date"` → url-pattern of servlet

```
<form method="POST" action="Date">
<input type="submit" value="click">
</form>
```

HttpServlet :-

→ HttpServlet is an abstract class which does not have any abstract method.
→ All the methods in HttpServlet class are concrete methods.
→ For different type of HttpServlet request, we have a corresponding doXXX() method in HttpServlet class
→ All the doXXX() methods are protected and takes two parameters

which are again specific to http, ie HttpServletRequest and HttpServletResponse.

→ Syntax:

protected void doXXX(HttpServletRequest req, HttpServletResponse resp)

→ Ex:- protected void doPost(✓, ✓)
protected void doGet(✓, ✓)

→ HttpServlet has doXXX() method for all 7 types of http request except "connect" ie HttpServlet does not have doConnect() method.

→ Its not a good practice to override the service method

→ Syntax of service() method is

* HttpServlet service() method:

protected void service(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException

* Servlet Interface service() method:

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

* Servlet Interface init() method:

public void init(ServletConfig conf)

* GenericServlet init() method:

public void init()

→ In the servlet hierarchy service() and init() are overloaded method.

→ If the method attribute is not defined for the form tag then type of request will be "GET".

Fetching the Form/UI Data :-

→ The form/UI data which are associated with the request object in the form of key-value pair can be fetched by using getParameter() method

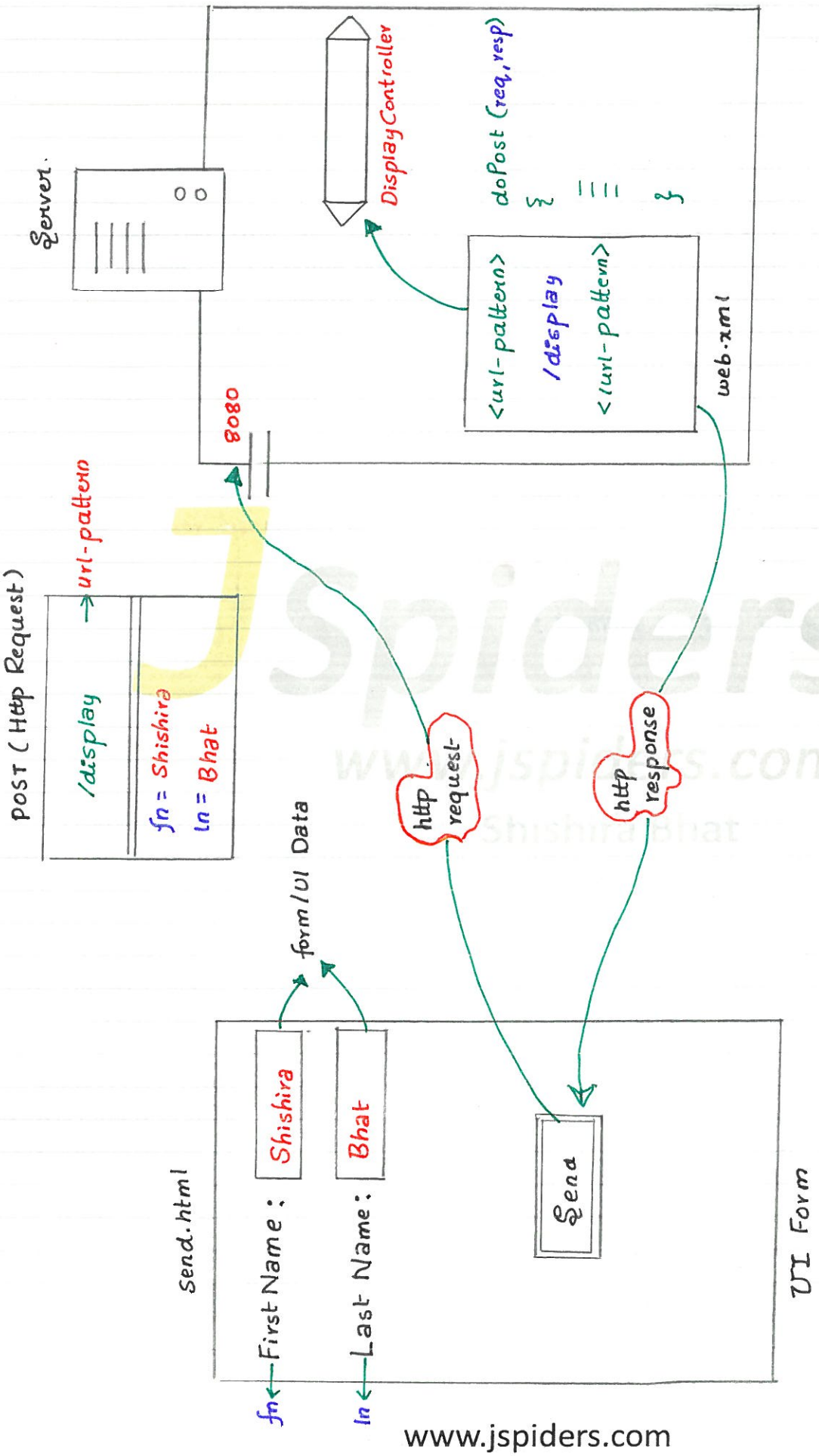
→ The syntax of method is

public string getParameter(String key)

→ Ex:- String firstName = req.getParameter("fn");
String lastName = req.getParameter("ln");

→ If the given key is not present then we get null value. But we dont get any exception or error.

→ Ex: String email = req.getParameter("em"); //null



Jspiders
Training & Development Center

Author:-Shishira Bhat
Technical Architect

Development Proceduce :

- Write the html page
- Configure the html page as welcome page in web.xml
- Creat Servlet
  - * Creat a class which extends HttpServlet
  - * Override doPost() method
  - * Write the logic inside doPost() method
- Configure the servlet in web.xml

Deployment Procedure :

→ Creat the root or the context.
→ Creat WEB-INF folder under root
→ Copy web.xml from eclipse to project Deployment folder (WEB-INF)
→ Copy classes folder (under build) to WEB-INF
→ Deploy the application to server.
→ start the server and access the application.
URL - http://localhost:8080/displayapp

```
// DisplayController.java
package org.shishira.displayapp;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.*;
public class DisplayController extends HttpServlet
{
  @override
  protected void doPost(HttpServletRequest req, HttpServletResponse resp)
      throws ServletException, IOException
    {
      /* fetch form/UI data */
      String firstName = req.getParameter("fn");
      String lastName = req.getParameter("ln");
      String fullName = firstName + " " + lastName;
      resp.setContentType("text/html");
      PrintWriter out = resp.getWriter();
      out.println("<html> <body bgcolor=\"red\" > ");
      out.println(fullName);
      out.println("</body> </html> ");
      out.flush();
      out.close();
    }
}
```

```
//web.xml
<?xml version="1.0" encoding="UTF-8" ?>
<web-app>
   <welcome-file-list>
      <welcome-file> send.html </welcome-file>
   </welcome-file-list>

   <servlet>
      <servlet-name> dc </servlet-name>
      <servlet-class> org.shishira.displayapp.DisplayController.java </servlet-class>
   </servlet>

   <servlet-mapping>
      <servlet-name> dc </servlet-name>
      <url-pattern> /display </url-pattern>
   </servlet-mapping>
</web-app>
```



```
//send.html
<html>
    <body bgcolor="#FFCC99">
     <form method="POST" action="display">
      First Name: <input type="text" name="fn"> <br>
      Last Name : <input type="text" name="ln"> <br>
      <input type="submit" value="send">
     </form>
    </body>
</html>
```

getParameterNames():

→ The getParameter() method returns value (UI Data) for the given key. But the getParameterNames() method returns the enumeration of keys or IDs or names

→ The getParameterNames() method does not return any UI Data. But it return the set of keys.

```
// MasterController.java
package org.shishira.enumapp;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Enumeration;
import javax.servlet.ServletException;
import javax.servlet.http.*;
public class MasterController extends HttpServlet {
```

```
@override
protected void doPOST(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException
    {
     /* fetch all the keys in the enumeration form */
    Enumeration <String> enumkeys = req.getParameterNames();
     while(enumkeys.hasMoreElement())
     {
      String key = enumKeys.nextElement();
      String value = req.getParameter(key);
      S.o.p(key + "      " + value);
     }
     PrintWriter out = resp.getWriter();
     out.println("<html> <body>");
     out.println("<h1> welcome... </h1> <p> </p>");
     out.println("<a href=\"index.html\"> click </a> ");
     out.println("</body> </html>");
     }
}
```


Note:
Instead of \"index.html\" we can write 'index.html'

```
//web.xml
<web-app>
  <servlet>
      <servlet-name> ms </servlet-name>
      <servlet-class> org.shishira.enumapp.MasterController </servlet-class>
  </servlet>

  <servlet-mapping>
      <servlet-name> ms </servlet-name>
      <url-pattern> /mast </url-pattern>
  </servlet-mapping>
</web-app>
```

Servlet Life Cycle:-

* Service Phase:

→ To serve the client request container calls the service() method ie conlainer is the caller of service() method.
→ Container creates implementation object of type HttpServletRequest Interface and implementation object of type HttpServletResponse Interface type.
→ And these two objects are passed as an argument to service() method.

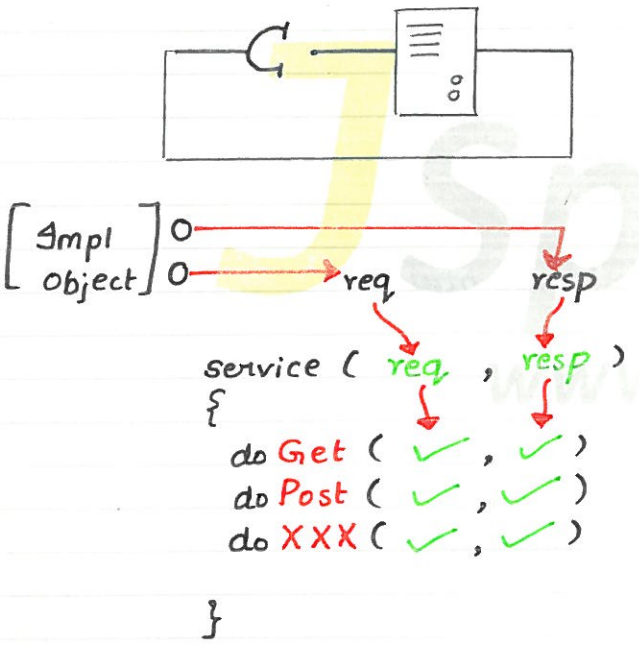
→ Container is the creator of request and response object.

Senario 1 : Service() method is not ready to serve :

> If the service() method is not ready to serve the client request then it throws ServletException to the container and will not call any doXXX() method.
> Server now internally creats appropriate error message and code and handles the further process based on the the project configuration.

Senario 2 : service() method is ready to serve the client request :

> If the service() method is ready to serve the client request then it identifies the type of Http Request and calls the appropriate doXXX() method by passing the same request and response object ie service() method is the caller of doXXX() method.




```
service ( HttpServletRequest req ,
          HttpServletResponse resp )
throws ServletException
{
 if (GET)
 {
   doGet ( );
 }

}
```


Instantiation & Initialization of Servlet :

* Instantiation :-

- Container creates an object of servlet by calling or invoking the zero parameterized or default constructor of the servlet.
- If serves does not find the zero param constructor in a servlet, then it throws InstantiationException.
- It is not a good practice to provide or write a parameterized constructor in a servlet.
- Constructor of servlet is used only for object creation.

* Initialisation :

→ Container calls the init() method which is parameterized and initializes the resources

Author:-Shishira Bhat
Technical Architect

the resources.

→ If we want to initialise the resources in a servlet, then we need to override the init() method.

→ The parameterized init() internally calls the zero parameterized init() method.

Note [Important] :

If we override the parameterized init() method, then we have to call the zero param init() method

```
@override
init ( ServletConfig conf )
{
 super.init();
 // initialisation
}
```

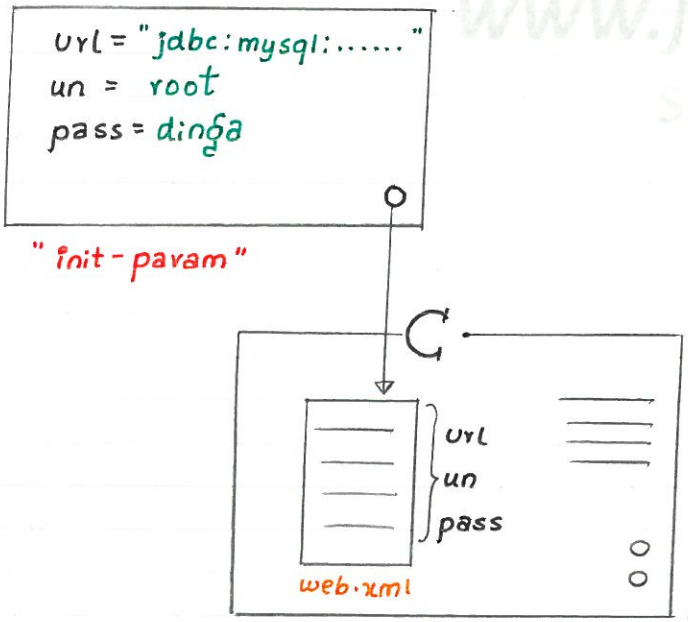

→ The data which are required to initialise the resources in the init() method is called as "init-parameter" [initialisation parameters]

→ Any no.of init parameter can be configured in web.xml

→ Container parses (reader) init parameter from web.xml and adds them to config object.

→ We can set any no.of init-parameter to the config object

→ Container calls the init() method by passing config object as an argument.




```
class GuluduServ ↑ HttpServlet
{
 @override
  init (ServletConfig conf)
  {
   super.init();
   // initialization
  }
}
```

① Object of servlet ⟶ zero/default constructor

```
GuluduServ serv = new GuluduServ();
ServletConfig conf = sub/Impl object
serv.init(config)
```

Author:-Shishira Bhat
Technical Architect

Servlet Life Cycle Notes :-

→ There are 4 stages in the lifecycle of servlet.
1. Instantiation Phase (object creation phase)
2. Initialization phase
3. Service phase
4. Destruction phase.

→ Based on the LoadOnStartup configuration in the web.xml or when a first request comes for a servlet; container loads the servlet.

Instantiation Phase :

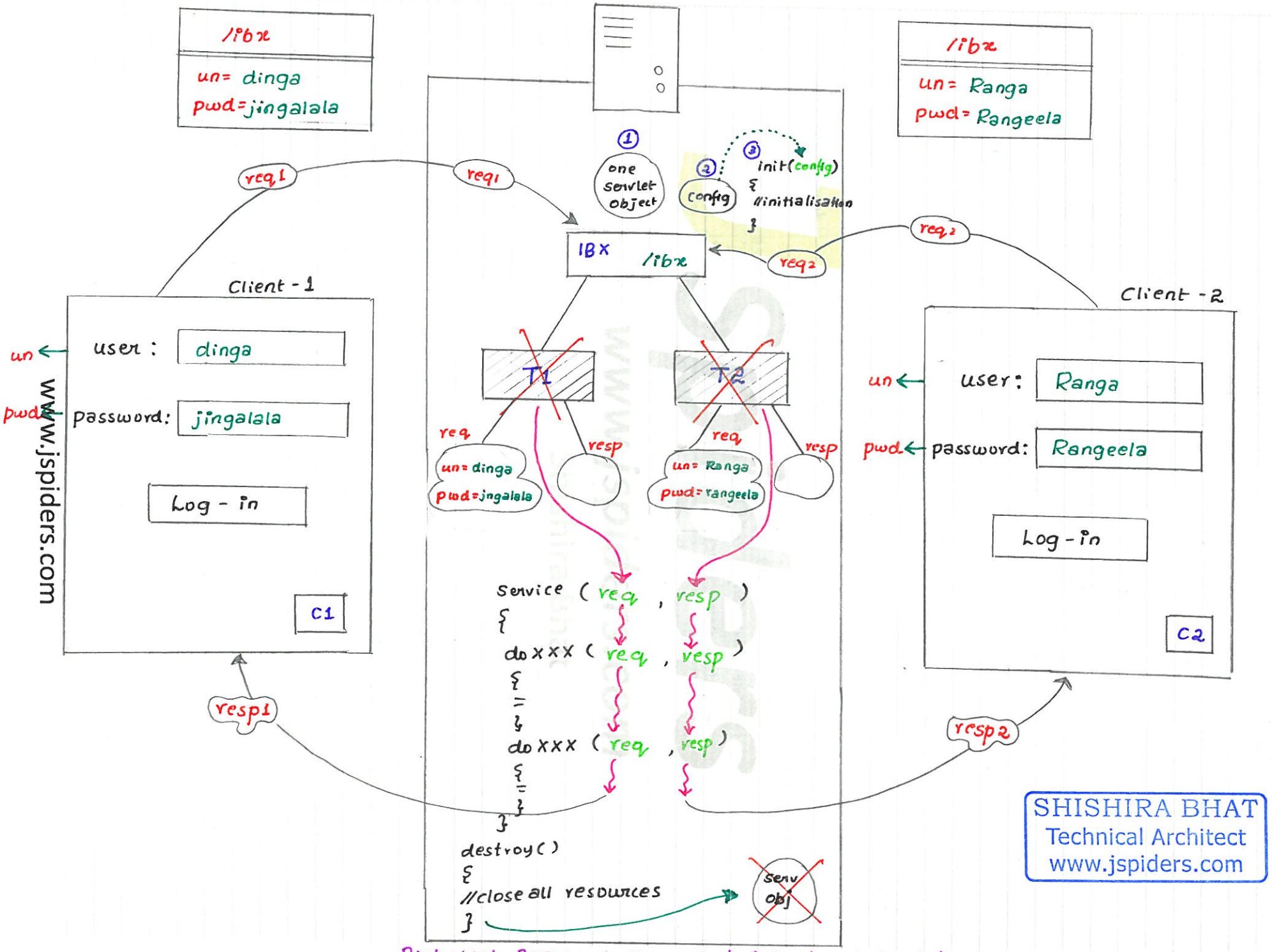
* Container calls the zero parameterized constructor of the servlet and creates one object of the servlet.
* If server does not find zero parameterized or default constructor, it throws instantiation exception wrapped into servlet exception.
* Container creats only one implementation object of type ServletConfig and adds the init-parameter into config object.
* Only one config object is creatd for one servlet.
* If the instantiation phase is successfull, then container process initialisation phase.

Initialization Phase :

→ Resource are initialized in this phase.
→ Container calls the parameterized init() method by passing the config object as an argument
→ The parameterized init() method internally calls the zero parameterized init() method.
→ The init() method is called only once by container
→ If initialisation phase fails container throws ServletException then container unloads the servlet.
→ If initialisation phase is success, then container process the service phase.

Service Phase :

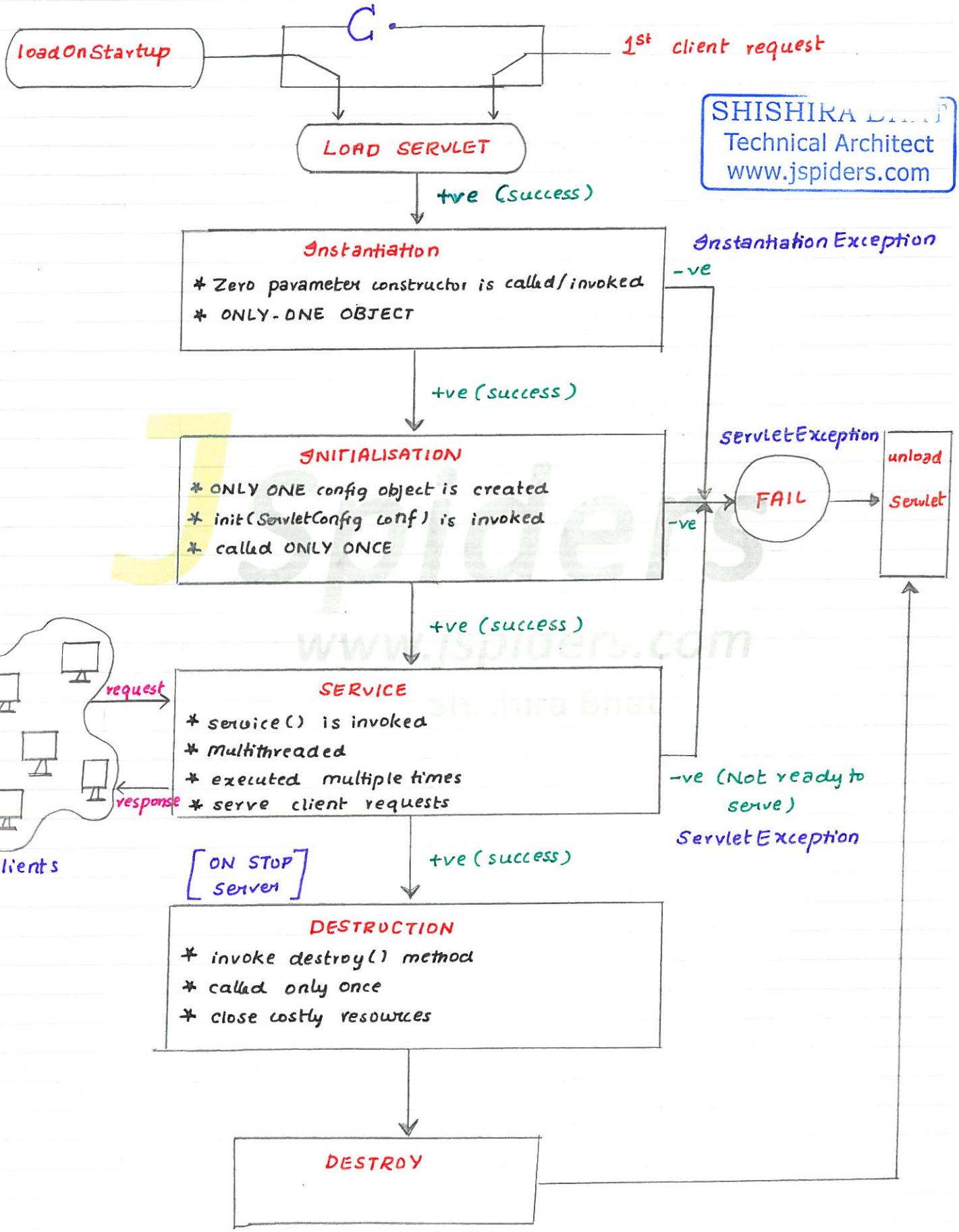
→ Container process the client request through service method.
→ Container executes the service method based on the multithreaded model. ie for every client request the container creates one thread and once the response is rendered or dispatched to the client then container destroys the thread.
→ For every client request container creat separate new thread along with implementation object of type ServletRequest and ServletResponse, UI Data or formdata are added to the request object
→ And then request and response objects are passed as an argument to service() method.



Pictorical Representation of LifeCycle of Servlet

Author:-Shishira Bhat
Technical Architect

FLOW DIAGRAM OF LIFECYCLE OF SERVLET



J spiders
Training & Development Center

Author:-Shishira Bhat
Technical Architect

Senario 1 :

→ If the service() method is not ready to serve, then it throws ServletException ie we can consider that the service phase has failed.
→ When service() methods fails container creates appropriate error code and message and process or render it into client (based on the web.xml configuration)
→ service() method is executed multiple times based on the client request
→ When service() executes successfully.

Scenario 2 :

→ If service() method executes successfully it renders or dispatches the response to the client.
→ Once the response is dispatched or rendered, Container destroys the thread along with associated request and response object.

* Service phase is executed multiple times based on the client request.
* When the server is stopped, before actually stopping the server container calls the destroy() method and then servlet object will be destroyed.
* When the servlet object is killed or destroyed we can say the lifeyle of servlet is ended.
* Now, server unloads the servlet
* We write the logic while overriding the destroy() method which are related to closing the opened costly resource.
* This method is called only once.

Note :-

We can override all these methods. But overriding service() method is not a good practice. Instead we have to override the doXXX() method.

Servlet Config :

→ ServletConfig is an interface which is present in javax.servlet package
→ The implementation for ServletConfig interface is provided by Container
→ The object of ServletConfig (Impl obj) is created by container immediately after the servlet object creation.
→ Only one config object is created for a single servlet
→ Config object stores all the configuration data which are required for the servlet

Configuration Data or init-parameters :

* Init-parameters are associated values in the form of key and value pair which are required to initialize the resource in the init() method.
* Ex: Database URL, DB username, DB Drivername etc.
* Any no of init-parameters can be configured inside the servlet tag (<servlet>)
* Container parses all the init-parameters and adds them into config object
* key must be unique.

→ init-parameters must be the subtags of servlet tag.
→ Ex:

```
<servlet>
    <servlet-name> dc </servlet-name>
    <servlet-class> org.shishira.lifeapp.DisplayController </servlet-class>
    <!-- init parameters START -->
        <init-param> dbdriver </init-param>
        <param-value> com.mysql.jdbc.Driver </param-value>
        <init-param>
            <param-name> dbdriver </param-name>
            <param-value> com.mysql.jdbc.Driver </param-name>
        </init-param>
        <init-param>
            <param-name> em </param-name>
            <param-value> imshishira@gmail.com </param-value>
        </init-param>
    <!-- init-parameters END -->
</servlet>
```

Accessing the Config Object :-

→ We can get the reference to a config object in two different ways.
  a. config reference is passed as an argument to init() method by the Container.
  b. By using a method called getServletConfig()
     Syntax: public ServletConfig getServletConfig()

→ getServletConfig() is an abstract method which is defined in Servlet-9nterface.
→ This method is overridden in GenericServlet abstract class. Hence this method is inherited to our servlet.
→ Ex:- ServletConfig myconfig = getServletConfig();
→ Syntax

```
public void init(ServletConfig config) throws ServletException
{
    // Initialisation
}
```


Accessing Declarative Data or 9nit Parameters :

→ The data which are configured in web.xml are called as Declarative data
→ The declarative data can be accessed only through the config reference by using the below 2 methods.

1. public String getInitParameter (String key);
2. public Enumeration <String> getInitParameterNames ();

→ The getInitParameter() returns the configured value if the key is present, if not returns null.
→ The getInitParameterNames() returns all the keys in the form of Enumeration and the keys are not configured (No init parameter are configured) then it returns empty Enumeration.

Ex:
```
@Override
public void init (ServletConfig config) throws ServletException
{
  super.init(config);   //important
  String email = config.getInitParameter("em");
  String driver = config.getInitParameter("driver");
  Enumeration <String> enumKeys = config.getInitParameterNames();
  while (enumKeys.hasMoreElements())
  {
   String key = enumKeys.nextElement();
   String value = config.getInitParameter(key);
   // use value and initialise the resources
  }
}
```

```
import java.util.Scanner;
public class Test
{
 public static void main (String[] args)
 {
  Scanner scan = new Scanner(System.in);
  S.o.p("Enter name");
  String name = scan.next();
  S.o.p("Enter ID");
  String id = scan.next();
  S.o.p("Enter percentage");
  double perc = scan.nextDouble();
  String qry = "insert into studentdb.student values " +
               ("'"+id+"'", '"+name+"', "+perc+")";

  S.o.p(qry);
 }
}
```

Approach :-

```
String qry = "insert into hybriddb.student values ('s1', 'kiran', 74.6 )";
Connection con;

① init ( config)
  {
   Class.forName ( config . getInitParameter ("dbdriver") );
      con = DriverManager. getConnection (url, un, pwd);
  }

② doPost ( ✓ , ✓ )
  {
   //insert
  }

③ destroy( )
  {
  //close all costly resources
  }
```


Load-on-Startup Concept :

→ <load-on-startup> is a subtag of <servlet> tag.
→ If we configure <load-on-startup> for any servlet in web.xml then Container creates an object of that configured servlet and initialises the servlet (calls the init()) method) on startup of the server without waiting for the 1st client request.
→ Instantiation and Initialisation phase executes on startup of the server.

Benefits of Load-on-Startup :

→ The delaytime or the waiting time for the first client request can be avoided.
→ We can make sure that the time taken to serve all the client request is same.
→ The value for <load-on-startup> must be a positive integer value.
→ Lowest positive integer values is given higher priority.
→ If the value of <load-on-startup> for multiple servlet is same , then priority depends on the parser user by the container

Servlet Context :

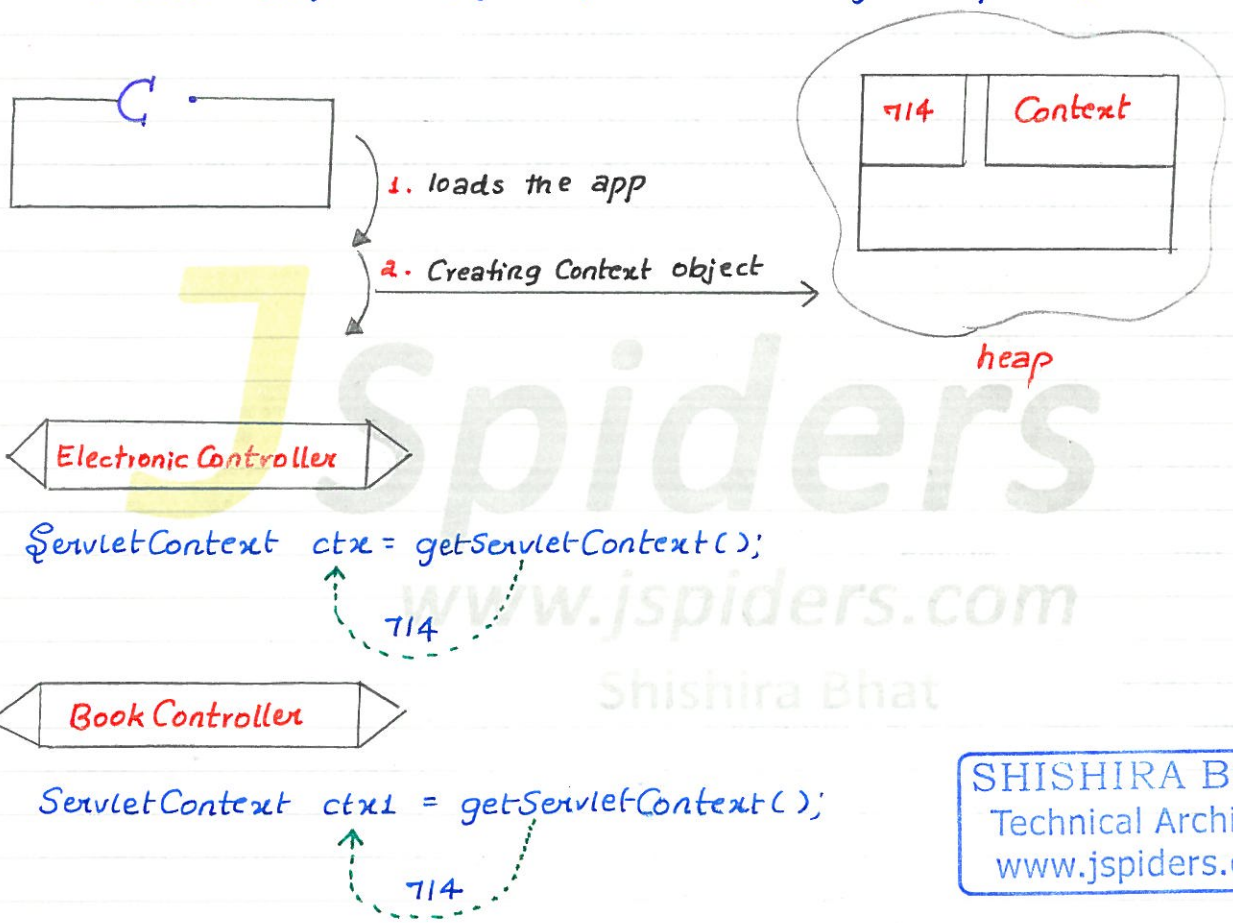
1. Servlet context is an interface which is present in javax.servlet package
2. When an application is loaded , container creates an implementation object of type ServletContext.
3. Only one object of context is created for one application
4. Container creates the context object much before the servlet object creation
5. Context is an implementation of singleton design pattern.

Author:-Shishira Bhat
Technical Architect

→ Context is referred as the environment which is created for an application
→ The scope of context is throughout the application ie the context object is available in the entire application.
→ We can get the reference to the context object by using an inherited method called getServletContext()
→ The getServletContext() is defined in ServletConfig interface, and this method is overridden in GenericServlet class.
→ Syntax:

```
public ServletContext getServletContext()
```

→ We can access only one object of the context by multiple references.



→ Both declarative as well as programatic data is applicable for context.
→ Declarative data → Context parameter
→ Programatic data → Attributes.

Declarative Data :

→ Declarative data is the data which we configure in web.xml
→ Declarative data is applicable for config and context
→ Declarative data for config is called as initParameters and declarative data for context is called as context-parameters.
→ Declarative data must be configured in the form of key-value pair where key must be unique.
→ init-parameters must be configured inside the <servlet> tag and

context-parameters must be configured outside the <servlet> tag.

→ Any number of declarative data can be configured in web.xml

→ context parameter syntax:

```
<context-param>
    <param-name> key </param-name>
    <param-value> value </param-value>
</context-param>
```

Programatic Data (Attributes):

* Programatic data means the data which we add programatically.
* We define the data in the form of attributes.
* Attributes are key-value pairs or associated values.
* Key must be unique and must be of String type.
* Value can be any object ie datatype of value can be any non-primitives.

Ex: String, Student object, User object, Mobile object and arrayList objects etc.

key	value
String	Object

Ex:

* Attribute 1

"nm"	"Shishira"

* Attribute 2

```
Mobile m = new Mobile("Galaxy 5", "red");
```

"mob"	m

* Attribute 3:

```
ArrayList<String> lst = new ArrayList<String>();
lst.add("niru");
lst.add("anu");
```

"names"	lst

→ The attributes can be added using below method

→ Syntax is

```
public void setAttribute(String key, Object value)
```

→ Programatic data is applicable Request, Context, Session object.

Jspiders
Training & Development Center

Author:-Shishira Bhat
Technical Architect

Note :

Programatic data is not applicable for config object.

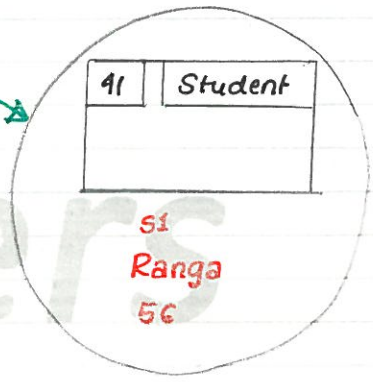
→ Programatic data can be fetched using below method

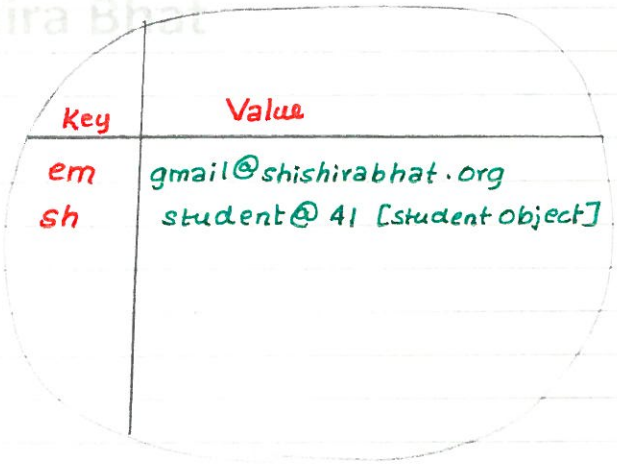
public Object getAttribute (String key)

→ If the key is not present, then we get the null value but not the exception nor the error.

setAttribute ( ) method :

```
req.setAttribute("em","gmail@shishirabhat.org");   //attribute 1

Student s = new Student("S1","Ranga",56);
student@41
req.setAttribute("sh",s);   //attribute 2
student@41
```





Note :-

While fetching the programatic data using getAttribute( ) we need to downcast to the appropriate datatype.

Ex: String email = (String) req.getAttribute("em");
String st = (Student) req.getAttribute("sh");

Servlet Chaining :

→ When a request comes to the servlet, the servlet can perform few logic and it can chain to another resources. So that the another chained resource can perform remaining logic

→ This way of communication b/w a servlet and a resource is called as Servlet chaining.

→ The another resource can be a static HTML page or a JSP page or another servlet.

→ Chaining can be done by using
1. Request Dispatcher
2. SendRedirect.

1. Request Dispatcher :

- RequestDispatcher is an interface which is present in javax.servlet package
- The implementation's object for RequestDispatcher is provided by Container.
- There are 2 methods in RequestDispatcher
  1. forward()
  2. include()
- Syntax of these methods,

```
public void forward(req, resp)

public void include(req, resp)
```

- RequestDispatcher works at serverside ie the chaining happens in the server.

Getting an object of RequestDispatcher:

→ There are 2 ways of getting the reference to RequestDispatcher

→ * RequestDispatcher = req.getRequestDispatcher("Path");

* RequestDispatcher = context.getRequestDispatcher("Path");

→ getRequestDispatcher() is a helper method which is defined in ServletRequest and ServletContext.

→ Syntax :

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

→ Ex:

```
req/context.getRequestDispatcher("serv");
req/context.getRequestDispatcher("another.html");
req/context.getRequestDispatcher("another.jsp");
```

Note :

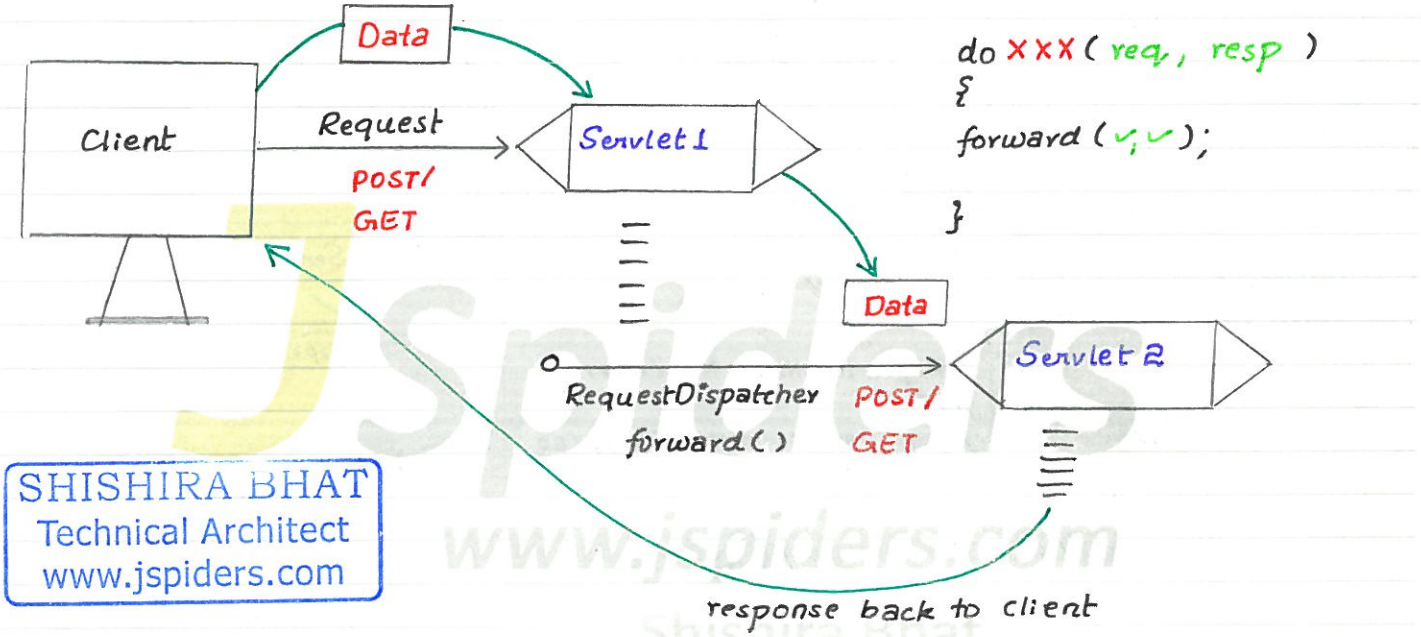
Since the RequestDispatcher works at serverside, the resourcename will not be displayed in browser ie the URL pattern of the second chained servlet, html page or jsp name will not be displayed.

→ Incase of RequestDispatcher the data associated with Request Object is consistently ~~men~~ maintained. Because only one request is forwarded to all chained resources.

→ The type of httpRequest is also consistently maintained across all chained resources.

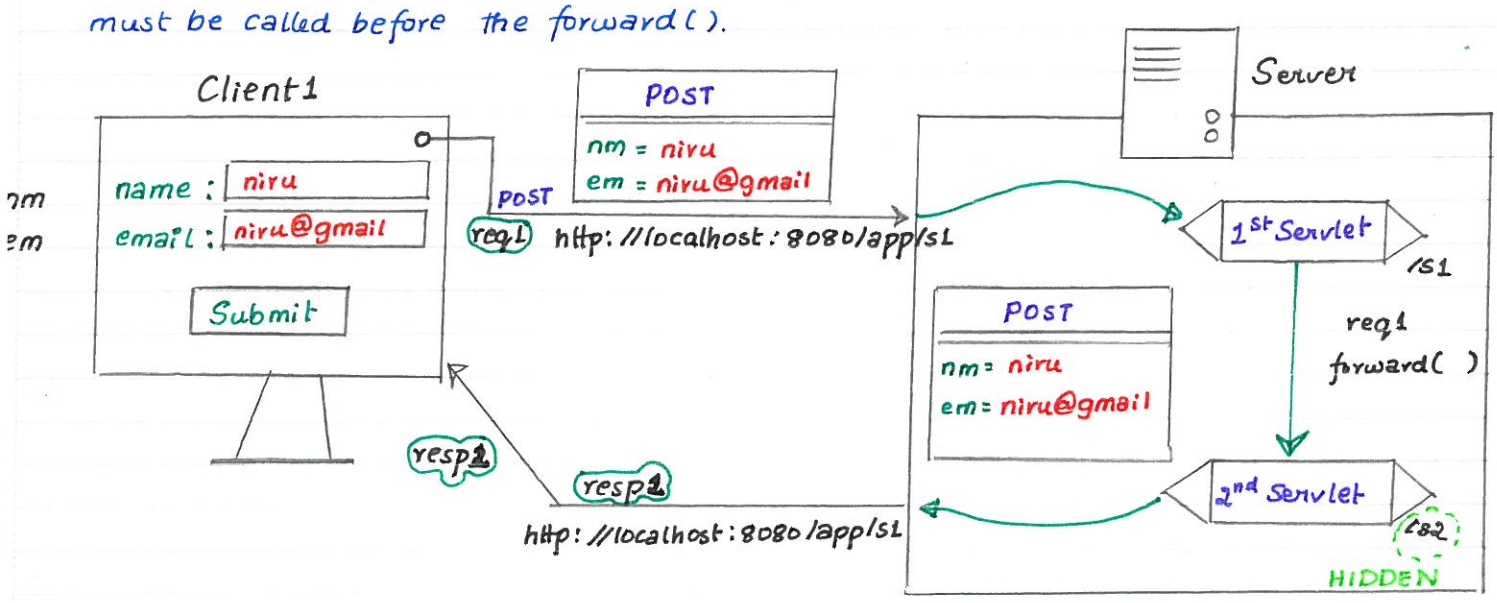
Note:

From one servlet, any no of inclusions can be done ie include() method can be called any no of times. But forward() called only once. Forward() must happen before committing the response to the client.



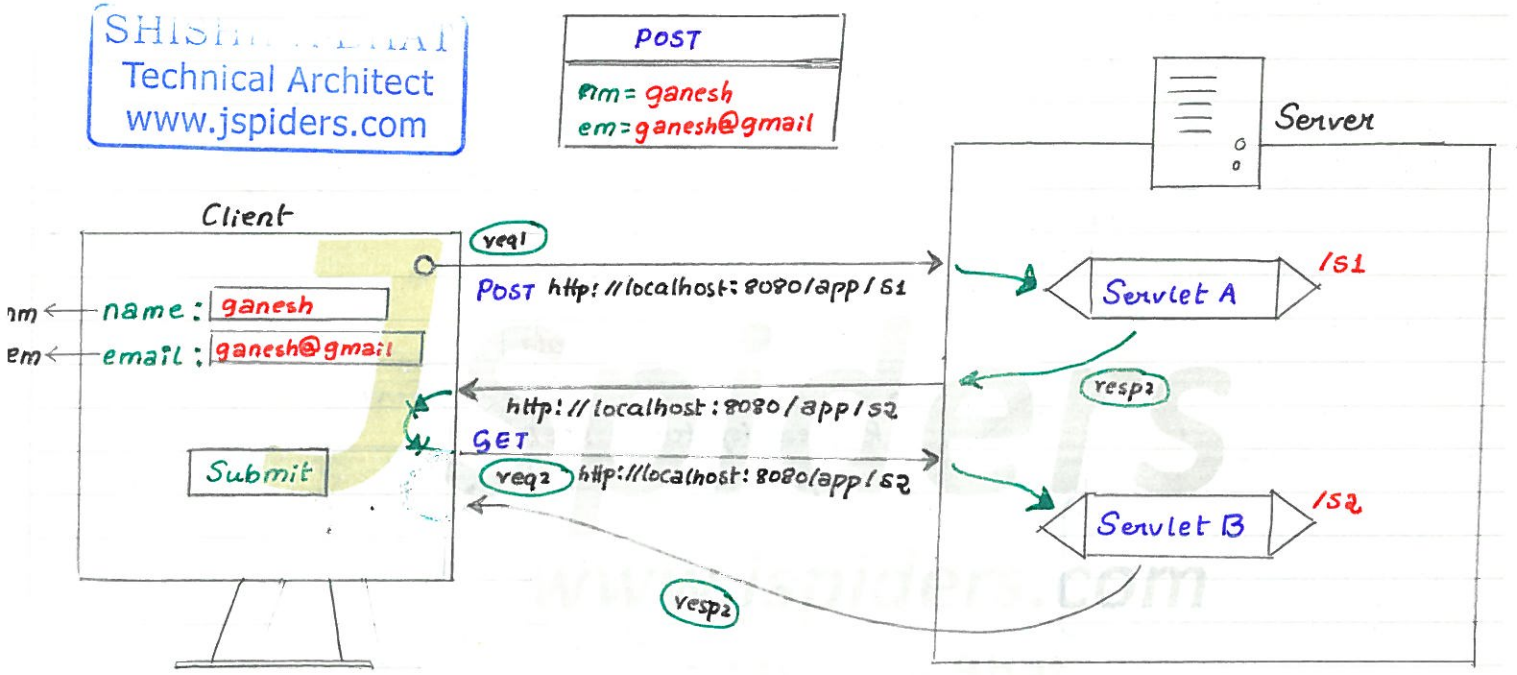
→ Once the request is forwarded from one resource to another, then it will not comeback to the first resource again.

→ include() and forward() can be used from the same servlet. But include() must be called before the forward().



SendRedirect :

* sendRedirect() is a method which is present in HttpServletResponse
* The change in the URL pattern is displayed in the browser.
* Using sendRedirect() we can chain to external resources.
* Type of request will be changed from POST to GET [But not from GET to POST]
* Data inconsistency ie data available for the first servlet is not available for the second servlet.
* The sendRedirect() works at the clientside or browserside ie the redirection happens in the browser.



Note :-

Don't use sendRedirect() to chain to other resources where data is involved. We can use SendRedirect() to communicate with extenal resources or to an internal resource where data is not involved.

URL - Rewriting :

: Cookies :

* A cookie is a file containing the information that is sent by a web server to a client.

* cookie store the data in the form of key-value/name-value string pair.

* Cookies are transmitted to clients through the HTTP response headers from the server. They are saved at the client side for the given domain & path.

* The cookie file persist on the client machine & the client browser returns the cookie to the original server.

* whenever the browser requests a resource, the cookie matching the domain & path of the request URL is sent to the server as a request header.

* Cookies are transmitted to the server through Http headers, when the request in sent to the server.

* with every consecutive request browser sends all corresponding cookies to the server. By accessing the cookies server able to remember client information across multiple request.

* A cookie consists of various attributes, such as name, value, maxage, domain, path, comment etc.

* Cookie must be used only to store non-sensitive infom.

* Although cookies were originally designed to help support session state, you can use custom cookies for other things.

* By default, a cookie lives only as long as a session; once the client quits browser, the cookie disappears. But you can tell a cookie to stay alive even AFTER the browser shuts down.

Author:-Shishira Bhat
Technical Architect

* The servlet API provides a class named Cookie under the javax.servlet.http package.

* The javax.servlet.http.Cookie object is designed to represent a HTTP cookie, which provides a convenient way to exchange data of a cookie between the container & the servlet.

* To send cookies to a client, a servlet can use the addCookie(Cookie ck) method of HttpServletResponse object.

This method adds fields to HTTP Response headers to send cookies on the client browser.

* To retrieve the cookies in a request returned by the browser, the servlet can use getCookies() method of the HttpServletResponse interface.

* The javax.servlet.http.Cookie class consists of one constructor with 2 String arguments.

```
public class Cookie
{
   public Cookie(String name, String value)
   {
     // initialize
   }
}
```


constructor, Key, Value

Ex; Cookie c1 = new Cookie("nm", "ShishiraABhat");

Cookie c2 = new Cookie("inst", "JSpiders");

<<interface>> HttpServletRequest
getCookies()

<<interface>> HttpServletResponse
addCookie(Cookie ck)

Important methods of Cookie class.

- getName() → returns the name of the cookie
- getValue() → returns the value of the cookie.
- public int getMaxAge();
  returns the maximum age of cookie in seconds.
- public void setMaxAge(int seconds)
  → setting maxage as -1, cookie will disable automatically when ever browser window closes.

ex:
```
Cookie c1 = new Cookie("nm", "Shishira Bhat");
c1.setMaxAge(20*60);
```
↳ 20 min * 60 Seconds

- public void setValue(String)
  → sets the value of a cookie. The value must not contain white space, brackets, parentheses, equals to sign, commas, double quotes, slashes, ?, @, ;, :.

- setDomain(String) and getDomain().

Advantages of cookies.

* cookies reduce network traffic, compared to URL rewriting.
* cookies maintain client data.
* cookies help reduce the complexity of application logic.
* All web servers provide support for cookie.
* persist across system even after system shut down.

Limitation of Cookie

* There may be a chance of disabling coockies at client side to meet security constraint. If coockies are disabled then session management by coockie is not possible,
* coockies are HTTP specific.
* coockie size allocated for a client is limited and varies from one client to another (Browser). Most browsers support 20 cookie for each domain, a total of 300 cookies, & 4 kB as the max size of each cookie.
* Due to limited size, cookies can not store large data.

Perpistent cookie & Non-perpistent cookie

If we are not setting maxAge() explicitly then such type of cookies are called non-persistent or temporary cookie. These will store in browser cache and non visible in local file system. once we close browser, the corresponding cookie disables automatically.

If we are setting maxAge() explicitly then such type of cookies are called persistent cookies. These will be stored in local file system & we can see. these cookies persists even after system shut down.

* Code Snippet *

```
/* Adding cookie */
package org.shishira.cookieapp;
import javax.servlet.http.*;
public class AddCookieDemoServlet extends HttpServlet
{
  @Override
  public void doXxx(request, response)
  {
    Cookie cookie = new Cookie("nm", "Shishira");
    cookie.setMaxAge(30*60);
    response.addCookie(cookie);
    // remaining code
  }
}
```

key
value
30 min
adding cookie to response header

```
/* Fetching cookies from Request */

package org.shishira.cookieapp;
import javax.servlet.http.*;
```



```
public class GetCookieDemo extends HttpServlet
{
    @Override
    public void doxxx(request, response)
    {
        Cookie[] cookies = request.getCookies();
        for(Cookie ck : cookies)
        {
            String name = ck.getName();
            String value = ck.getValue();
            // other code
        }
    }
```

For complete code & programs on cookie;-

REFER — CLASS NOTES.

"Service without Attitude,
Love without Reason,
Knowledge without Intellect, Life beyond Events
is WHAT you ARE."
— Gurudev

Author:-Shishira Bhat
Technical Architect