Module 2

❖ Web Application Basics

- **Web Server:**

  A web server takes a client request and gives something back to the client. The server uses HTTP to send HTML to the client.

- **Web client:**

  A web client lets the user request something on the server, and shows the user the result of the request.

  Client and servers know HTML and HTTP.

- **HTML & HTTP**

  HTML tells the browser how to display the content to the user.

  HTTP is the protocol clients and servers use on the web to communicate.

  HTTP method-

  The method name tells the server , the kind of request that is being made.Http protocal has several methods. The most often are GET and POST.

  GET:-

  - It is a simplest http method.
  - It ask the server to get a resource and send it back to the client.
  - The total number of characters in a get is limited(depending upon the server).
  - The data we send with get is appended to the URL up in the browser bar. So what ever data is send is exposed.
  - It is insecure.

POST:-

- Post is more powerful request.
- Using post we can request something and at the same time send form data to the server.
- There is no limitation of sending Form data.
- Data can be securely send to server is called payload.

MIME Type:-

- The HTTP response has both header and body. Header information tells the browser about the protocal being used ,weather the request is successful and kind of content included in the body etc.
- And the body contain the content for the browser to display.
- A MIME type tells the browser what kind of data the browser is going to receive so that the browser will know what to do with it.

❖ Web Servers and Servlet

Web server loves serving static page just like there in a directory. Server finds it and hand it back to the client as is.

Disadvantages of web server:-

Two things web server alone won’t do.

- Dynamic web pages because that doesn’t exist before the request.
- Ability to save some data(write) on the data base server.

❖ Introduction to Servlet

- Java Servlets are an efficient and powerful solution for creating dynamic content for the Web. Over the past few years Servlets have become the fundamental building block of mainstream server-side Java.
- The power behind Servlets comes from the use of Java as a platform and from interaction with a Servlet container. The Java platform provides a Servlet developer with a robust API, object-orientated programming, platform neutrality, strict types, garbage collection, and all the security features of the JVM.
- Servlets are always part of a larger project called a Web Application. A Web Application is a complete collection of resources for a Web site. A Web Application from consisting of zero, one, or multiple Servlets, but a Servlet container manages Servlets on a per Web Application basis.
- Servlets already use the security features provided by the Java Virtual Machine, but the Servlet specification also defines a mechanism for controlling access to resources in a Web Application.
- One of the best features of a Servlet is the ability to develop content for just about any language. A large part of this functionality comes directly from the Java platform’s support for internationalization and localization. The Servlet API keeps this functionality and can be easily used to create content in most of the existing languages.

❖ Servlets and HTTP Servlets

- The primary purpose of the Servlet specification is to define a robust mechanism for sending content to a client as defined by the Client/Server model. Servlets are most popularly used for generating dynamic content on the Web and have native support for HTTP.

❖ Life Cycle of a Servlet

- This life cycle governs the multi-threaded environment that Servlets run in and provides an insight to some of the mechanisms available to a developer for sharing server-side resources. Servlets follow a three-phase life: initialization, service, and destruction, with initialization and destruction typically performed once, and service performed many times.

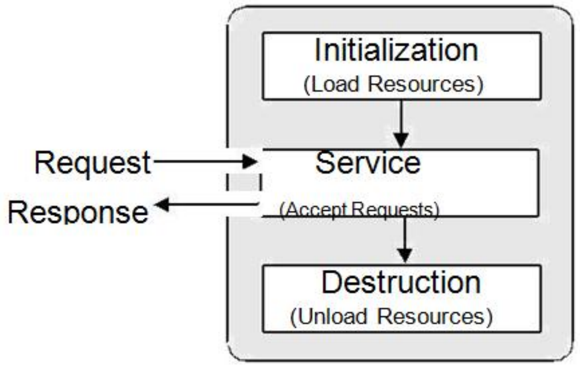


Fig: Diagram of the Servlet Life Cycle

- Initialization is the first phase of the Servlet life cycle and represents the creation and initialization of resources the Servlet may need to service requests. All Servlets must implement the javax.servlet.Servlet interface. This interface defines the init() method to match the initialization phase of a Servlet life cycle. When a container loads a Servlet, it invokes the init() method before servicing any requests.
- The service phase of the Servlet life cycle represents all interactions with requests until the Servlet is destroyed. The Servlet interface matches the service phase of the Servlet life cycle to the service() method. The service() method of a Servlet is invoked once per a request and is responsible for generating the response to that request. The Servlet specification defines the service() method to take two

parameters: a javax.servlet.ServletRequest and a javax. servlet.ServletResponse object. These two objects represent a client’s request for the dynamic resource and the Servlet’s response to the client.

- The destruction phase of the Servlet life cycle represents when a Servlet is being removed from use by a container. The Servlet interface defines the destroy() method to correspond to the destruction life cycle phase. Each time a Servlet is about to be removed from use, a container calls the destroy() method, allowing the Servlet to gracefully terminate and tidy up any resources it might have created.

Servlet Example:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloWorld extends HttpServlet {
public void doGet(HttpServletRequest request,HttpServletResponse response)throws
        IOException, ServletException
{
response.setContentType("text/html");
PrintWriter out = response.getWriter();
out.println("<html>");
out.println("<head>");
out.println("<title>Hello World!</title>");
out.println("</head>");
out.println("<body>");
out.println("<h1>Hello World!</h1>");
out.println("</body>");
out.println("</html>");
}}
```

Deploying a Servlet:

Servlet class file goes in the /WEB-INF/classes directory of the application with all the other Java classes. For a client to access a Servlet, a unique URL, or set of URLs, needs to be declared in the Web Application Deployment Descriptor. The web.xml deployment description relies on new elements14: servlet and servlet-mapping need to be introduced for use in web.xml. The servlet element is used to define a Servlet that should be loaded by a Web Application. The servlet-mapping element is used to map a Servlet to a given URL or set of URLs. Multiple tags using either of these elements can appear to define as many Servlets and Servlet mappings as needed.

Deploying HelloWorld Servlet:

```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee" version="2.4">
<servlet>
<servlet-name>HelloWorld</servlet-name>
<servlet-class>com.jspbook.HelloWorld</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>HelloWorld</servlet-name>
<url-pattern>/HelloWorld</url-pattern>
</servlet-mapping>
<welcome-file-list>
<welcome-file>welcome.html</welcome-file>
</welcome-file-list>
</web-app>
```

❖ Web Container:

- Servlets doesn’t have main() they are under the control of another java application called container.
- The web container has to instantiate the servet when the request comes.
- It calls the doGet() or doPost().
- It get the request and response to the servlet ,manages life death and resource of the servlet.
- Tomcat is an example of container.
- When the web server gets request for a servlet other than plain old HTML static page that web browser handles the request to the container in which the servlet is deployed.

❖ **Functions of a Container:-**

- Communication support:-Container provides you a way for your servlet to talk with web server. Container knows the protocal between the web browser and itself.
- Life cycle management:- The container controls the life and death of the servlet. It take care of loading classes ,instanciating the servle, invoking the servlet methods and making servlet instance elligible for garbage collection.
- Multi threading support : -Container autometically creates a new java thread for every servelt request it receives.
- Declarative Security:- With configuring deployment descriptor we can manage and we can change our security without touching or recompiling java source code.
- Jsp support:- compile jsp to java servlet class.

Examples of web container

Non commercial -Apache tomcat , Jetty

Commercial- Oracle application Server ,Web Sphere ,BEA Web Logic Server

Open source – JBoSS.

❖ The Servlet API

Javax.servlet Package

- Every servlet must implement javax.servlet.Servlet interface
- Most servlets implement the interface by extending one of these classes
  - javax.servlet.GenericServlet
  - javax.servlet.http.HttpServlet

❖ Generic Servlet class

- javax.servlet.GenericServlet is a abstract class that it implements most of the basic servlet methods.

  init(ServletConfig) – initialize the servlet

  service(ServletRequest , ServletResponse) – carries out single request from client.

  destroy() – cleanup what ever resource are being held.

  getServletConfig() – returns a Servlet config object which contains any initialization parameter and start up configuration for the servlet.

  getServletInfo() – returns a string containing inforamation about servlet such as author,version, copyright.

  getInitParameter(String) – get the paramter value that is configured in web.xml

  getServletContext() – returns a servlet context object

❖ Http Servlet class

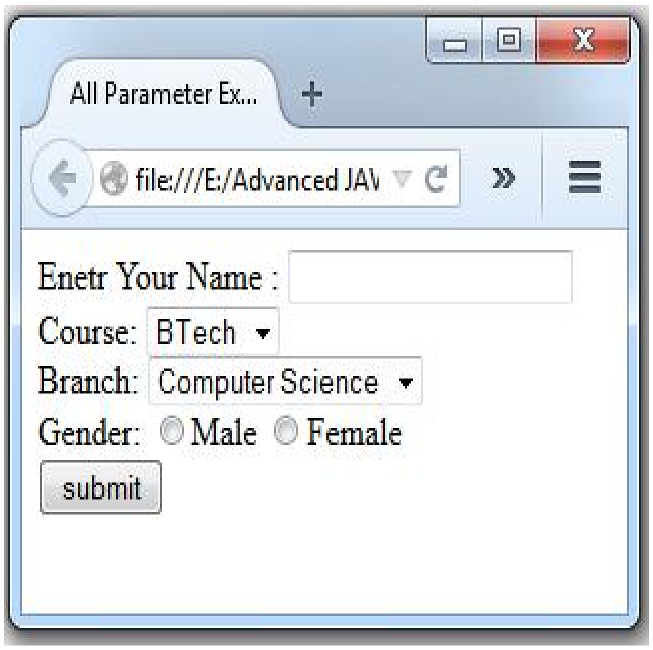
- It is present in javax.servlet.http package.
- service(HttpServletRequest , HttpServletResponse) – the service() contains http specific servlet response object.

- doGet(HttpServletRequest , HttpServletResponse) – handles get request
- doPost(HttpServletRequest , HttpServletResponse) – handles psot request
- doPut(HttpServletRequest , HttpServletResponse) – handles put request
- doDelete(HttpServletRequest , HttpServletResponse) – handles delete request
- doTrace(HttpServletRequest , HttpServletResponse) – handles tracing request

❖ Reading Servlet parameters

Example:-

To read all the Form parameters in a Servlet program.



Enetr Your Name :

Course: BTech

Branch: Computer Science

Gender: ○ Male ○ Female

submit

StudentForm.html

```
<html>
    <head>
            <title>All Parameter Example</title>
```

```
</head>
<body>
        <form method="get" action="YourName.do">
        Enetr Your Name :
        <input type="text" name="t1" value="" />
        </br>
                Course:
                <select name="course">
                        <option value="BTech">BTech</option>
                        <option value="MCA">MCA</option>
                        <option value="MBA">MBA</option>
                </select>
                </br>
                Branch:
                <select name="branch">
                        <option value="Computer Science">Computer
                        Science</option>
                        <option value="Electrical ">Electrical</option>
                        <option value="Mechanical">Mechanical</option>
                        <option value="Electronics">Electronics</option>
                </select>
                </br>
                Gender:
<input type="radio" name="c1" value="male"></input>Male
<input type="radio" name="c1" value="female"></input>Female
</br>
        <input type="submit" value="submit"/>
</form>
```

```
</body>
<html>
```

ServletExample.java

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

public class ServletExample extends HttpServlet
{
    public void doPost(HttpServletRequest req,HttpServletResponse res)
throws IOException,ServletException
    {
            Enumeration paramname=req.getParameterNames();
            PrintWriter out=res.getWriter();
            while(paramname.hasMoreElements())
            {

                    String s=(String)paramname.nextElement();
                    out.println(s+":"+req.getParameter(s)+"</br>");
            }
    }
}
```

- **ServletRequest Object**
  - It provides client request information to a servlet.
  - The servlet container creates a servlet request object and pass it as an argument to servlet’s service().
  - The ServletRequest interface define function to retrive the data send as client request like parameter name and value , attribues ,input streams etc.
  - HttpServletRequest extends ServletRequest interface to provide request information for HttpServlet.

Methods:-

<u>ServletRequest</u>

- getParameter(String)
- getParameterNames()
- getParameterValues(String)
- getAttribute(String)

<u>HttpServletRequest</u>

- getContextPath()
- getCookies()
- getSession()

- **ServletResponse Object**
  - It define an object to assist a servlet in sending a response to the client.
  - Servlet container creates a ServletResponse object and pass it as an argument to servlet’s service().

Methods:-

ServletResponse interface

- getBufferSize()
- setContentType()
- getOutputStream()
- getWriter()
- setContentLength()
- // MANY more methods...

HttpServletResponse

- addCookie()
- addHeader()
- encodeURL()
- sendError()
- setStatus()
- sendRedirect()

❖ **Redirecting Request and Request Dispatch:-**

Redirect Request:-

- sendRedirect() of a response object send the url to the browser that includes the parameter of sendRedirect().
- Browser treats it as a new request from the client.
- Redirect is similar to opening a new browser and type the url there itself.

response.SendRedirect("URL");

Redirect Dispatch:-

- forword() of RequestDispatcher is handled on the serer.
- Therefore the request and its associated session available to the forworded resource and we can pass the data in between them using request.setAttribute().
- The method generally sends a request and response object to the resourse in the same servletContext.
- RequestDispatcher view=req.getRequestDispatcher("jsp page");

  view.forword(request,response);

❖ **Reading initialization parameters**

ServletConfig:-

- Servlet can have initialized parameter.
- Its about deploy time value we have to configure for servlet.
- We don't want to hard code into servlet like database name.
- It is used to access ServletContext.
- Parameters are configured in the Deployment Descriptor.
- When container initialize the servlet it makes a unique ServletConfig for the Servlet.
- One servlet config for one servlet.
- Example:-

In Deplouemnt Descriptor :-

```
<servlet>
     <servlet-name>ServletExample</servlet-name>
     <servlet-class>ServletEx</servlet-class>
     <init-param>
               <param-name>email</param-name>
               <param-value>biswajitsamal@cutm.ac.in</param-value>
     </init-param>
  </servlet>
```

In Servlet

```
getServletConfig().getInitParam(“email”);
```

- Our servlet inherits getServletconfig() . so we can call this method to get ServletConfig object and we can call getInitParam().
- Container reads the Servlet init parameter from deployment descriptor and give them to ServletConfig and pass ServletConfig to Servlet init().

ServletContext:-

- ServletContext should have been named Application Context.
- One ServletContext per web application
- It is used to access web-application parameter also configured in deployment descriptor.
- It is used to get server info, including name and version of container and also the version of API that support.
- Context parameter are available to the entair application.

- Servlet and jsp in the application has autometically has access to the context-init parameter.

- In Deployemnt Descriptor:-

```
<context-param>
          <param-name>admin</param-name>
          <param-value>admin@123</param-value>
  </context-param>
```

- In Servelt or jsp

```
getServletContext().getInitParameter(“admin”);
```

❖ Developing and Deploying Servlets

Steps to run Servlet:-

1. Create a directory structure
2. Write the servlet code
3. Compile
4. Deploy servlet
5. Run tomcat
6. Call servlet from web browser

❖ Deployment Descriptor

- Servlet specification defines a configuration file called Deployment Descriptor.
- It contains meta data for web-application such as
    - Default page to show
    - Servlet to load

        - Security restriction etc.

- Servlet init parameter and context init parameter are also configured in web.xml
- The container that have a listener for this application is also configured in web.xml.
- The file is always located in WEB-INF folder and named as web.xml
- When the container loads the web application it check this file.
- Servlet sepcification defines that the entair WEB-INF directory of any web-application must be kept hidden from the user.

❖ **Session Tracking & Management**

Http is stateless and we need to keep track of transaction between request.We need to keep conversational state with the client across multiple requests. Session Tracking is done via 1. Cookeis 2. Hidden Fileds 3. URL rewriting 4. Session API.

❖ **Cookies**

- A cookie is a bit of information sent by web Server to a browser that can latter be read back from the browser.
- A server can take that bit of information and use it as a key to recover information about previous visit.
- This information may be database or shared object.
- Cookies are read from request object by calling getCookies() on the request object.
- Cookies are placed in the browser by calling addCookie() on the response object.

❖ Steps to create New Cookie:

1. Create a new cookie object.

   Cookie cookie=new Cookie(“name”,value);

Ex:- Cookie c=new Cookie(“userid”,”biswajit”);

2. Setting the Maximum age

   c.setMaxAge(60);

3. Add cookie to your response object

   HttpServletResponse response;

   Response.addCookie(c);

- ❖ Disadvantages:

  Cookies can be added or disabled by client

- ❖ Reading Cookies from Client

  Request.getCookie() – it returns always a Cookie object .

  Example-

```
Cookie [] cookies = request.getCookies();
        for(int i=0;i<cookies.length;i++)
        {
                Cookie c=cookies[i];
                if(c.getName().equals("username")){
                        String userName=c.getValue();
                        out.println("Hello"+userName);
                        break;
                }
        }
```

❖ **Hidden Form Fields:-**

Hidden Form field is an invisible text field is used for maintaining the state of the user. In such case we can store the information in the hidden field and get it from another servlet. This approach is better if we have to submit the form in all pages and we don’t want in the browser.

Properties:-

1. It always work weather cookies is diasable or not.
2. Mentain at server side
3. Extra form submision required on each page.
4. Only textual information is used.

❖ **URL Rewriting:-**

1. In url rewriting we append a token or identifier to the url of the next servlet or next resource.
2. We can send parameter name/value pair using following format.
   url ? name1=value1 & name2=value
3. A name/value is separated using = sign
4. A parameter name/value pair is separated from other parameter & symbol.
5. When a user clicks the hyperlink ,the parameter name/value pair will be passed to the server.
6. We can use getParameter() method to obtain the value.

❖ **HttpSession**

Session Management:-A servlet or jsp page which manages a request which needs session support must do following.

1. Session creation and identification

Identifying an already associated session which can be associated to the request.

HttpSession session=request.getSession();

Create a session if the request doesn’t contain any session identifier ,otherwise it returns the session identifier attached to that request and use it to build an object of HttpSession.

HttpSession session =request.getSession(true/false);

If false it retrives an already existing session but it deosn’t create new one.if true same as request.getSession().

2. Attribute Management
3. Session Tracking
4. Session Destruction

Methods of HttpSession Interface:-

- public String getId() –it gives a unique identier value.
- public long getCreatedTime():- it gives information about when the session is created.
- public void invalited():- it invalidate the session.

- **Events and Listeners in Servlet**

1. Events are basically occurrence of something.
2. Challenging the state of an object is known as event.
3. The servlet specification includes the capabilities to track key events in the web applicationthrough event Listeners.
4. Application event provides notifications of a change in state of the Servlet Context or of the HttpSession object.
5. We can write Event Listeners to respond to the change in the state and we configure application event and listener classes in the web application
6. Listener are classified on the basis of event types.
7. Event can be of request , sesison, application level scope.
8. Listener interface are present in javax.servlet package.

   Some

ContextListeners and Event Type

ServletContextListener	-	ServletContextEvent
ServletContextAttributeListener	-	ServletContextAttributeEvent

RequestListener

ServletRequestListener	-	ServletRequestEvent
ServletRequestAttributeListener	-	ServletRequestAttribute

SessionListener

HttpSessionListener	-	HttpSessionEvent
HttpSessionActivationListener	-	HttpSessionBindingEvent
HttpSessionBindingListener	-	HttpSessionBindingEvent
HttpSessionAttributeListenr	-	HttpSessionEvent

ServletContextListener and ServletContextEvent

ServletContextEvent is notified when web application is deployed on the server.For ServletContext events , the event listener classes can receive notification when the web application is deployed or being undeployed and when attributes are added , removed or repalced.

Method for ServletContext Listener interface:-

There are two methods declared in the ServletContextListeners interface which must be implemented by the servlet program to perform some action such as creating databse connection.

public void contextInitialized(ServletContextEvent servletContextEvent)

– this method is invoked when the application is deployed on the server.

public void contextDestroyed(ServletContextEvent servletContextEvent)

- this method is invoked when the application is undeployed on the server.

ServletContextAttributeListener and ServletContextAttributeEvent

Method for ServletContextAttributeListener interface:-

public void attributeAdded(ServletContextAttributeEvent e)

public void attributeRemoved(ServletContextAttributeEvent e)

public void attributeReplaced(ServletRequestAttributeEvent e)

ServletRequestListener and ServletRequestEvent

This listener listens lifecycle events of request object.

Methods:

public void requestInitialized(ServletRequestEvent e) :

This method will executed automatically at the time of request object creation i.e. before starting service method-

public void requestDestroyed(ServletRequestEvent rre)

This method will executed automatically at the time of request object destroyed.

ServletRequestAttributeListener:

This listener listens events related to request scoped attributes. It has 3 methods.

public void attributeAdded(ServletRequestAttributeEvent srae) public void attributeRemoved(ServletRequestAttributeEvent srae)

public void attributeReplaced(ServletRequestAttributeEvent srae)

HttpSession Listener and HttpSessionEvent

The HttpSession event is notified when the session object changed. The corresponding listener interface for this event is HttpSessionListener.

Methods of HttpSessionListener interface are

public void sessionCreated(HttpSessionEvent se)

public void sessionDestroyed(HttpSessionEvent se)

HttpSessionAttribute Listener

Methods are-

public void attributeAdded(HttpSessionBindingEvent e)

public void attributeRemoved(HttpSessionBindingEvent e)

public void attributeReplaced(HttpSessionBindingEvent e)

HttpSessionBinding Listener

public void valueBound(HttpSessionBindingEvent e)

public void unBound(HttpSessionBindingEvent e)

HttpSessionActivation Listener

public void sessionwillPassivate(HttpSessionBindingEvent e);

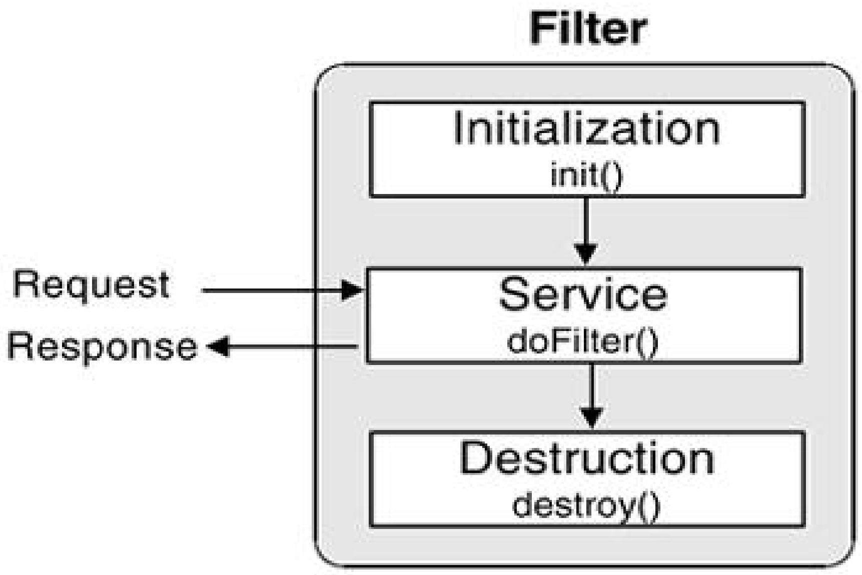
public void session DiActivate(HttpSessionBindingEvent e);

- **Filters**
  1. Filters concept has introduced in Servlet 2.3 version.
  2. A filter is a program that runs on the server before the Servlet or JSP page with which it is associated. A filter can be attached to one or more Servlet or JSP pages
  3. Filters can examine the request information going into these resources.
  4. Filters are used to preprocess the Request and post process the Response.
  5. Preprocessing involves
     - Authentication,
     - Logging,
     - Authorization,
     - Change request information , etc..
  6. Post processing involves
     - Compression of the response.
     - Encryption of the response.
     - To alter response information.
     - Other Tasks
     - Security verification
     - Session validation
     - Internationalization
     - Data compression
     - MIME type changing

Advantages of Filters

- Encapsulate common behavior.
    - Have 30 different Servlet or JSP pages that need to compress their content to decrease download time? Make 1 compression filter and apply it to all 30 resources.
    - The filter class encapsulates the logic that has to be executed before or after the actual request processing, which is done by the requested resources.
    - The filter class is declared in the deployment descriptor.
    - It provides the ability to encapsulate recurring tasks in reusable units., Which allows us to modularize the code, which makes the code more manageable, documentable, easy to debug, and can be reused in other settings.
- Separate high-level access decisions from presentation code.
    - Want to block access from certain sites without modifying the individual pages to which these access restrictions apply?
    - Create an access restriction filter and apply it to as many pages as you like.
- Apply wholesale changes to many different resources.
    - Have a bunch of existing resources that should remain unchanged except that the company name should be changed?
    - Make a string replacement filter and apply it wherever appropriate.

❖ The Filter Life Cycle



Filter API

This API comprises of three interfaces

- javax.servlet.Filter
- javax.servlet.FilterConfig
- javax.servlet.FilterChain
- javax.servlet.Filter

Every Filter class must implement Filter interface either directly or indirectly.

- void init(FilterConfig filterConfig)

  - Initializes a filter and makes it ready for providing service.
  - The Servlet container calls the init method exactly once after instantiating the filter.

    - The web container cannot place the filter into service if the init method either

  – Throws a ServletException
  – Does not return within a time period defined by the web container

- void doFilter (ServletRequest request, ServletResponse response, FilterChain chain)

  - Encapsulates service logic to be implemented on ServletRequest to generate the ServletResponse.
  - The FilterChain refernce passed as an argument to forward request/response pair to the filter or target resource of chain.
- void destroy( )

  Called by the web container to indicate to a filter that it is being taken out of service.
- javax.servlet.FilterConfig
    - Used during initialization of filter.
    - This object is used to fetch configuration information specified in web.xml
- String getFilterName()

  Returns the filter-name of this filter as defined in the deployment descriptor.
- String getInitParameter(String name)

  Returns a String containing the value of the named initialization parameter, or null if the parameter does not exist.
- Enumeration getInitParameterNames()

Returns the names of the filter's initialization parameters as an Enumeration of String objects, or an empty Enumeration if the filter has no initialization parameters.

- ServletContext getServletContext()

  Returns a reference to the ServletContext in which the caller is executing.
- javax.servlet.FilterChain

  The object of this interface stores information about a chain of filters.
- void doFilter(ServletRequest request, ServletResponse response)

  Causes the next filter in the chain to be invoked, or if the calling filter is the last filter in the chain, causes the resource at the end of the chain to be invoked.
  - Container doesn’t call this method.
  - We have to call it explicitly.
  - chain.doFilter(request, response);