**1.What is the Servlet?**

A servlet is a Java programming language class that is used to extend the capabilities of servers that host applications accessed by means of a request- response programming model.

**2.What are the new features added to Servlet 2.5?**

Following are the changes introduced in Servlet 2.5:

* A new dependency on J2SE 5.0
* Support for annotations
* Loading the class
* Several web.xml conveniences
* A handful of removed restrictions
* Some edge case clarifications

|  |
| --- |
|  |

**3.What are the uses of Servlet?**

Typical uses for HTTP Servlets include:

* Processing and/or storing data submitted by an HTML form.
* Providing dynamic content, e.g. returning the results of a database query to the client.
* A Servlet can handle multiple request concurrently and be used to develop high performance system
* Managing state information on top of the stateless HTTP, e.g. for an online shopping cart system which manages shopping carts for many concurrent customers and maps every request to the right customer.

**4.What are the advantages of Servlet over CGI?**

Servlets have several advantages over CGI:

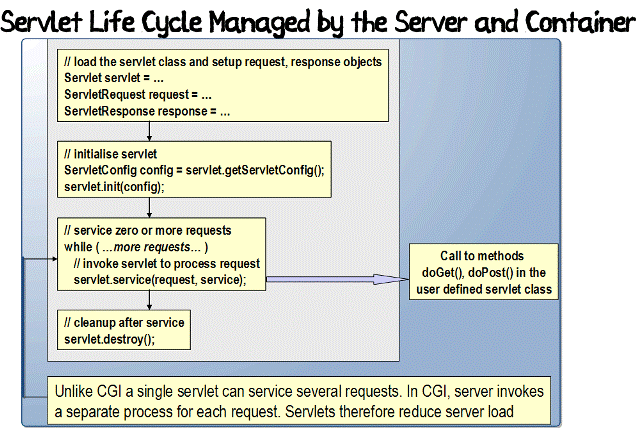
* A Servlet does not run in a separate process. This removes the overhead of creating a new process for each request.
* A Servlet stays in memory between requests. A CGI program (and probably also an extensive runtime system or interpreter) needs to be loaded and started for each CGI request.
* There is only a single instance which answers all requests concurrently. This saves memory and allows a Servlet to easily manage persistent data.
* Several web.xml conveniences
* A handful of removed restrictions
* Some edge case clarifications

**5.What are the phases of the servlet life cycle?**

The life cycle of a servlet consists of the following phases:

* **Servlet class loading** : For each servlet defined in the deployment descriptor of the Web application, the servlet container locates and loads a class of the type of the servlet. This can happen when the servlet engine itself is started, or later when a client request is actually delegated to the servlet.
* **Servlet instantiation** : After loading, it instantiates one or more object instances of the servlet class to service the client requests.
* **Initialization (call the init method)** : After instantiation, the container initializes a servlet before it is ready to handle client requests. The container initializes the servlet by invoking its init() method, passing an object implementing the ServletConfig interface. In the init() method, the servlet can read configuration parameters from the deployment descriptor or perform any other one-time activities, so the init() method is invoked once and only once by the servlet container.
* **Request handling (call the service method)** : After the servlet is initialized, the container may keep it ready for handling client requests. When client requests arrive, they are delegated to the servlet through the service() method, passing the request and response objects as parameters. In the case of HTTP requests, the request and response objects are implementations of HttpServletRequest and HttpServletResponse respectively. In the HttpServlet class, the service() method invokes a different handler method for each type of HTTP request, doGet() method for GET requests, doPost() method for POST requests, and so on.
* **Removal from service (call the destroy method)** : A servlet container may decide to remove a servlet from service for various reasons, such as to conserve memory resources. To do this, the servlet container calls the destroy() method on the servlet. Once the destroy() method has been called, the servlet may not service any more client requests. Now the servlet instance is eligible for garbage collection

The life cycle of a servlet is controlled by the container in which the servlet has been deployed.



**6.Why do we need a constructor in a servlet if we use the init method?**

Even though there is an init method in a servlet which gets called to initialize it, a constructor is still required to instantiate the servlet. Even though you as the developer would never need to explicitly call the servlet's constructor, it is still being used by the container (the container still uses the constructor to create an instance of the servlet). Just like a normal POJO (plain old java object) that might have an init method, it is no use calling the init method if you haven't constructed an object to call it on yet.

**7.How the servlet is loaded?**

A servlet can be loaded when:

* First request is made.
* Server starts up (auto-load).
* There is only a single instance which answers all requests concurrently. This saves memory and allows a Servlet to easily manage persistent data.
* Administrator manually loads.

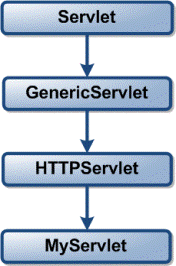
**8.How a Servlet is unloaded?**

A servlet is unloaded when:

* Server shuts down.
* Administrator manually unloads.

**9.What is Servlet interface?**

The central abstraction in the Servlet API is the Servlet interface. All servlets implement this interface, either directly or , more commonly by extending a class that implements it.



*Note: Most Servlets, however, extend one of the standard implementations of that interface, namely* javax.servlet.GenericServlet andjavax.servlet.http.HttpServlet.

|  |
| --- |
|  |

**10.What is the GenericServlet class?**

GenericServlet is an abstract class that implements the Servlet interface and the ServletConfig interface. In addition to the methods declared in these two interfaces, this class also provides simple versions of the lifecycle methods init and destroy, and implements the log method declared in the ServletContext interface.   
*Note: This class is known as generic servlet, since it is not specific to any protocol.*

**11.What's the difference between GenericServlet and HttpServlet?**

|  |  |
| --- | --- |
| **GenericServlet** | **HttpServlet** |
| The GenericServlet is an abstract class that is extended by HttpServlet to provide HTTP protocol-specific methods. | An abstract class that simplifies writing HTTP servlets. It extends the GenericServlet base class and provides an framework for handling the HTTP protocol. |
| The GenericServlet does not include protocol-specific methods for handling request parameters, cookies, sessions and setting response headers. | The HttpServlet subclass passes generic service method requests to the relevant doGet() or doPost() method. |
| GenericServlet is not specific to any protocol. | HttpServlet only supports HTTP and HTTPS protocol. |

|  |
| --- |
|  |

**12.Why is HttpServlet declared abstract?**

The HttpServlet class is declared abstract because the default implementations of the main service methods do nothing and must be overridden. This is a convenience implementation of the Servlet interface, which means that developers do not need to implement all service methods. If your servlet is required to handle doGet() requests for example, there is no need to write a doPost() method too.

**13.Can servlet have a constructor ?**

One can definitely have constructor in servlet.Even you can use the constrctor in servlet for initialization purpose,but this type of approch is not so common. You can perform common operations with the constructor as you normally do.The only thing is that you cannot call that constructor explicitly by the new keyword as we normally do.In the case of servlet, servlet container is responsible for instantiating the servlet, so the constructor is also called by servlet container only.

**14.What are the types of protocols supported by HttpServlet ?**

It extends the GenericServlet base class and provides a framework for handling the HTTP protocol. So, HttpServlet only supports HTTP and HTTPS protocol.

**15.What is the difference between doGet() and doPost()?**

|  |  |  |
| --- | --- | --- |
| **#** | **doGet()** | **doPost()** |
| 1 | In doGet() the parameters are appended to the URL and sent along with header information. | In doPost(), on the other hand will (typically) send the information through a socket back to the webserver and it won't show up in the URL bar. |
| 2 | The amount of information you can send back using a GET is restricted as URLs can only be 1024 characters. | You can send much more information to the server this way - and it's not restricted to textual data either. It is possible to send files and even binary data such as serialized Java objects! |
| 3 | doGet() is a request for information; it does not (or should not) change anything on the server. (doGet() should be idempotent) | doPost() provides information (such as placing an order for merchandise) that the server is expected to remember |
| 4 | Parameters are not encrypted | Parameters are encrypted |
| 5 | doGet() is faster if we set the response content length since the same connection is used. Thus increasing the performance | doPost() is generally used to update or post some information to the server.doPost is slower compared to doGet since doPost does not write the content length |
| 6 | doGet() should be idempotent. i.e. doget should be able to be repeated safely many times | This method does not need to be idempotent. Operations requested through POST can have side effects for which the user can be held accountable. |
| 7 | doGet() should be safe without any side effects for which user is held responsible | This method does not need to be either safe |
| 8 | It allows bookmarks. | It disallows bookmarks. |

**16.When to use doGet() and when doPost()?**

Always prefer to use GET (As because GET is faster than POST), except mentioned in the following reason:

* If data is sensitive
* Data is greater than 1024 characters
* If your application don't need bookmarks.

**17.How do I support both GET and POST from the same Servlet?**

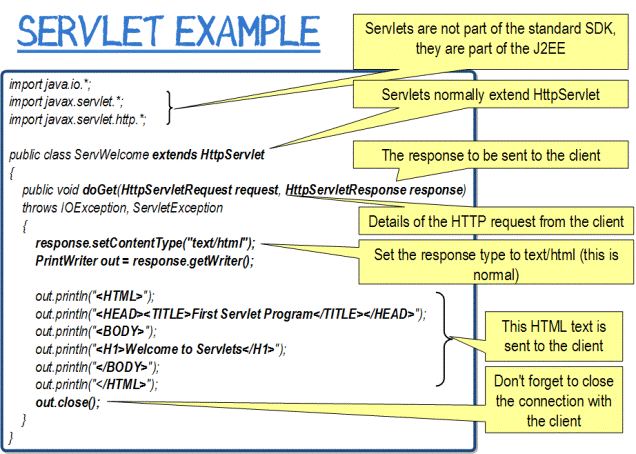
The easy way is, just support POST, then have your doGet method call your doPost method:

**public void doGet(HttpServletRequest request, HttpServletResponse response)  
                        throws ServletException, IOException  
{  
    doPost(request, response);     
}**

**18.Should I override the service() method?**

We never override the service method, since the HTTP Servlets have already taken care of it . The default service function invokes the doXXX() method corresponding to the method of the HTTP request.For example, if the HTTP request method is GET, doGet() method is called by default. A servlet should override the doXXX() method for the HTTP methods that servlet supports. Because HTTP service method check the request method and calls the appropriate handler method, it is not necessary to override the service method itself. Only override the appropriate doXXX() method.

**19.How the typical servlet code look like ?**



**20.What is a servlet context object?**

A servlet context object contains the information about the Web application of which the servlet is a part. It also provides access to the resources common to all the servlets in the application. Each Web application in a container has a single servlet context associated with it.

**21.What are the differences between the ServletConfig interface and the ServletContext interface?**

|  |  |
| --- | --- |
| **ServletConfig** | **ServletContext** |
| The ServletConfig interface is implemented by the servlet container in order to pass configuration information to a servlet. The server passes an object that implements the ServletConfig interface to the servlet's init() method. | A ServletContext defines a set of methods that a servlet uses to communicate with its servlet container. |
| There is one ServletConfig parameter per servlet. | There is one ServletContext for the entire webapp and all the servlets in a webapp share it. |
| The param-value pairs for ServletConfig object are specified in the <init-param> within the <servlet> tags in the web.xml file | The param-value pairs for ServletContext object are specified in the <context-param> tags in the web.xml file. |

**22.What's the difference between forward() and sendRedirect() methods?**

|  |  |
| --- | --- |
| **forward()** | **sendRedirect()** |
| A forward is performed internally by the servlet. | A redirect is a two step process, where the web application instructs the browser to fetch a second URL, which differs from the original. |
| The  browser is completely unaware that it has taken place, so its original URL remains intact. | The browser, in this case, is doing the work and knows that it's making a new request. |
| Any browser reload of the resulting page will simple repeat the original request, with the original URL | A browser reloads of the second URL ,will not repeat the original request, but will rather fetch the second URL. |
| Both resources must be part of the same context (Some containers make provisions for cross-context communication but this tends not to be very portable) | This method can be used to redirect users to resources that are not part of the current context, or even in the same domain. |
| Since both resources are part of same context, the original request context is retained | Because this involves a new request, the previous request scope objects, with all of its parameters and attributes are no longer available after a redirect. (Variables will need to be passed by via the session object). |
| Forward is marginally faster than redirect. | redirect is marginally slower than a forward, since it requires two browser requests, not one. |

**23.What is the difference between the include() and forward() methods?**

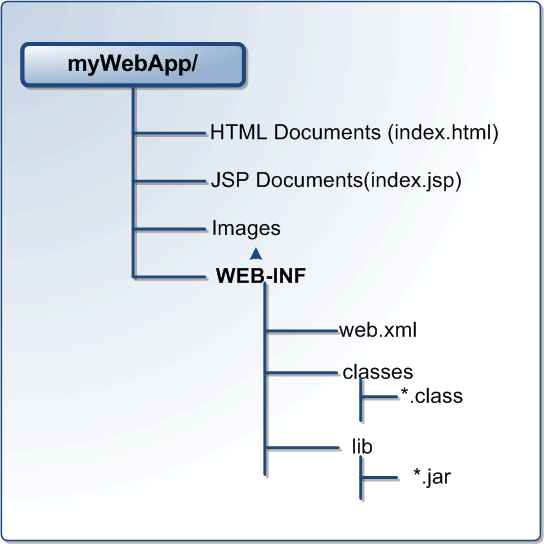
|  |  |
| --- | --- |
| **include()** | **forward()** |
| The RequestDispatcher include() method inserts the the contents of the specified resource directly in the flow of the servlet response, as if it were part of the calling servlet. | The RequestDispatcher forward() method is used to show a different resource in place of the servlet that was originally called. |
| If you include a servlet or JSP document, the included resource must not attempt to change the response status code or HTTP headers, any such request will be ignored. | The forwarded resource may be another servlet, JSP or static HTML document, but the response is issued under the same URL that was originally requested. In other words, it is not the same as a redirection. |
| The include() method is often used to include common "boilerplate" text or template markup that may be included by many servlets. | The forward() method is often used where a servlet is taking a controller role; processing some input and deciding the outcome by returning a particular response page. |

|  |
| --- |
|  |

**24.What's the use of the servlet wrapper classes??**

The HttpServletRequestWrapper and HttpServletResponseWrapper classes are designed to make it easy for developers to create custom implementations of the servlet request and response types. The classes are constructed with the standardHttpServletRequest and HttpServletResponse instances respectively and their default behaviour is to pass all method calls directly to the underlying objects.

**25.What is the directory structure of a WAR file?**



**26.What is a deployment descriptor?**

A deployment descriptor is an XML document with an .xml extension. It defines a component's deployment settings. It declares transaction attributes and security authorization for an enterprise bean. The information provided by a deployment descriptor is declarative and therefore it can be modified without changing the source code of a bean.  
The JavaEE server reads the deployment descriptor at run time and acts upon the component accordingly.

**27.What is the difference between the getRequestDispatcher(String path) method of javax.servlet.ServletRequest interface and javax.servlet.ServletContext interface?**

|  |  |
| --- | --- |
| **ServletRequest.getRequestDispatcher(String path)** | **ServletContext.getRequestDispatcher(String path)** |
| The getRequestDispatcher(String path) method ofjavax.servlet.ServletRequest interface accepts parameter the path to the resource to be included or forwarded to, which can be relative to the request of the calling servlet. If the path begins with a “/” it is interpreted as relative to the current context root. | The getRequestDispatcher(String path) method ofjavax.servlet.ServletContext interface cannot accept relative paths. All path must start with a “/” and are   interpreted as relative to current context root. |

|  |
| --- |
|  |

**28.What is preinitialization of a servlet?**

A container does not initialize the servlets as soon as it starts up, it initializes a servlet when it receives a request for that servlet first time. This is called lazy loading. The servlet specification defines the element, which can be specified in the deployment descriptor to make the servlet container load and initialize the servlet as soon as it starts up. The process of loading a servlet before any request comes in is called preloading or preinitializing a servlet.

**29.What is the <load-on-startup> element?**

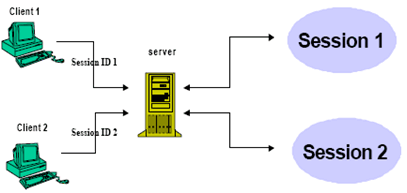
The <load-on-startup> element of a deployment descriptor is used to load a servlet file when the server starts instead of waiting for the first request. It is also used to specify the order in which the files are to be loaded. The <load-on-startup> element is written in the deployment descriptor as follows:

<servlet>  
   <servlet-name>ServletName</servlet-name>  
   <servlet-class>ClassName</servlet-class>  
   <load-on-startup>1</load-on-startup>  
</servlet>

*Note: The container loads the servlets in the order specified in the <load-on-startup> element.*

**30.What is session?**

A session refers to all the requests that a single client might make to a server in the course of viewing any pages associated with a given application. Sessions are specific to both the individual user and the application. As a result, every user of an application has a separate session and has access to a separate set of session variables.



|  |
| --- |
|  |

**31.What is Session Tracking?**

Session tracking is a mechanism that servlets use to maintain state about a series of requests from the same user (that is, requests originating from the same browser) across some period of time.

**32.What is the need of Session Tracking in web application?**

HTTP is a stateless protocol i.e., every request is treated as new request. For web applications to be more realistic they have to retain information across multiple requests. Such information which is part of the application is reffered as "state". To keep track of this state we need session tracking.   
  
*Typical example:* Putting things one at a time into a shopping cart, then checking out--each page request must somehow be associated with previous requests.

**33.What are the types of Session Tracking ?**

Sessions need to work with all web browsers and take into account the users security preferences. Therefore there are a variety of ways to send and receive the identifier:

* **URL rewriting :**URL rewriting is a method of session tracking in which some extra data (session ID) is appended at the end of each URL. This extra data identifies the session. The server can associate this session identifier with the data it has stored about that session. This method is used with browsers that do not support cookies or where the user has disabled the cookies.
* **Hidden Form Fields :** Similar to URL rewriting. The server embeds new hidden fields in every dynamically generated form page for the client. When the client submits the form to the server the hidden fields identify the client.
* **Cookies :**Cookie is a small amount of information sent by a servlet to a Web browser. Saved by the browser, and later sent back to the server in subsequent requests. A cookie has a name, a single value, and optional attributes. A cookie's value can uniquely identify a client.
* **Secure Socket Layer (SSL) Sessions :** Web browsers that support Secure Socket Layer communication can use SSL's support via HTTPS for generating a unique session key as part of the encrypted conversation.

|  |
| --- |
|  |

|  |
| --- |
|  |

**34.How do I use cookies to store session state on the client?**

In a servlet, the HttpServletResponse and HttpServletRequest objects passed to method HttpServlet.service() can be used to create cookies on the client and use cookie information transmitted during client requests. JSPs can also use cookies, in scriptlet code or, preferably, from within custom tag code.

* To set a cookie on the client, use the addCookie() method in class HttpServletResponse. Multiple cookies may be set for the same request, and a single cookie name may have multiple values.
* To get all of the cookies associated with a single HTTP request, use the getCookies() method of class HttpServletRequest

**35.What are some advantages of storing session state in cookies?**

* Cookies are usually persistent, so for low-security sites, user data that needs to be stored long-term (such as a user ID, historical information, etc.) can be maintained easily with no server interaction.
* For small- and medium-sized session data, the entire session data (instead of just the session ID) can be kept in the cookie.

|  |
| --- |
|  |

**36.What are some disadvantages of storing session state in cookies?**

* Cookies are controlled by programming a low-level API, which is more difficult to implement than some other approaches.
* All data for a session are kept on the client. Corruption, expiration or purging of cookie files can all result in incomplete, inconsistent, or missing information.
* Cookies may not be available for many reasons: the user may have disabled them, the browser version may not support them, the browser may be behind a firewall that filters cookies, and so on. Servlets and JSP pages that rely exclusively on cookies for client-side session state will not operate properly for all clients. Using cookies, and then switching to an alternate client-side session state strategy in cases where cookies aren't available, complicates development and maintenance.
* Browser instances share cookies, so users cannot have multiple simultaneous sessions.
* Cookie-based solutions work only for HTTP clients. This is because cookies are a feature of the HTTP protocol. Notice that the while package javax.servlet.http supports session management (via class HttpSession), packagejavax.servlet has no such support.

**37.What is URL rewriting?**

URL rewriting is a method of session tracking in which some extra data is appended at the end of each URL. This extra data identifies the session. The server can associate this session identifier with the data it has stored about that session.

Every URL on the page must be encoded using method HttpServletResponse.encodeURL(). Each time a URL is output, the servlet passes the URL to encodeURL(), which encodes session ID in the URL if the browser isn't accepting cookies, or if the session tracking is turned off.  
E.g., http://abc/path/index.jsp;jsessionid=123465hfhs

**Advantages**

* URL rewriting works just about everywhere, especially when cookies are turned off.
* Multiple simultaneous sessions are possible for a single user. Session information is local to each browser instance, since it's stored in URLs in each page being displayed. This scheme isn't foolproof, though, since users can start a new browser instance using a URL for an active session, and confuse the server by interacting with the same session through two instances.
* Entirely static pages cannot be used with URL rewriting, since every link must be dynamically written with the session state. It is possible to combine static and dynamic content, using (for example) templating or server-side includes. This limitation is also a barrier to integrating legacy web pages with newer, servlet-based pages.

**DisAdvantages**

* Every URL on a page which needs the session information must be rewritten each time a page is served. Not only is this expensive computationally, but it can greatly increase communication overhead.
* URL rewriting limits the client's interaction with the server to HTTP GETs, which can result in awkward restrictions on the page.
* URL rewriting does not work well with JSP technology.
* If a client workstation crashes, all of the URLs (and therefore all of the data for that session) are lost.

|  |
| --- |
|  |

**38.How can an existing session be invalidated?**

An existing session can be invalidated in the following two ways:

* Setting timeout in the deployment descriptor: This can be done by specifying timeout between the<session-timeout>tags as follows:

<session-config>  
       <**session-timeout>10</session-timeout>**  
</session-config>

This will set the time for session timeout to be ten minutes.

* Setting timeout programmatically: This will set the timeout for a specific session. The syntax for setting the timeout programmatically is as follows:

public void setMaxInactiveInterval(int interval)

The setMaxInactiveInterval() method sets the maximum time in seconds before a session becomes invalid.   
Note :Setting the inactive period a*s negative(-1), makes the container stop tracking session, i.e, session never expires.*

**39.How can the session in Servlet can be destroyed?**

An existing session can be destroyed in the following two ways:

* Programatically : Using session.invalidate() method, which makes the container abonden the session on which the method is called.
* When the server itself is shutdown.

**40.A client sends requests to two different web components. Both of the components access the session. Will they end up using the same session object or different session ?**

Creates only one session i.e., they end up with using same session .

Sessions is specific to the client but not the web components. And there is a 1-1 mapping between client and a session.

**41.What is servlet lazy loading?**

* A container doesnot initialize the servlets ass soon as it starts up, it initializes a servlet when it receives a request for that servlet first time. This is called lazy loading.
* The servlet specification defines the <load-on-startup> element, which can be specified in the deployment descriptor to make the servlet container load and initialize the servlet as soon as it starts up.
* The process of loading a servlet before any request comes in is called preloading or preinitializing a servlet.

**42.What is Servlet Chaining?**

Servlet Chaining is a method where the output of one servlet is piped into a second servlet. The output of the second servlet could be piped into a third servlet, and so on. The last servlet in the chain returns the output to the Web browser.

**43.How are filters?**

Filters are Java components that are used to intercept an incoming request to a Web resource and a response sent back from the resource. It is used to abstract any useful information contained in the request or response. Some of the important functions performed by filters are as follows:

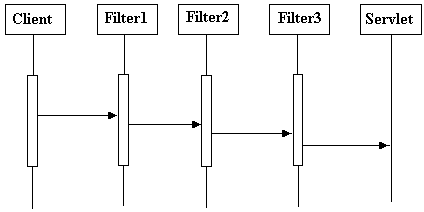
* Security checks
* Modifying the request or response
* Data compression
* Logging and auditing
* Response compression

Filters are configured in the deployment descriptor of a Web application. Hence, a user is not required to recompile anything to change the input or output of the Web application.

**44.What are the functions of an intercepting filter?**

The functions of an intercepting filter are as follows:

* It intercepts the request from a client before it reaches the servlet and modifies the request if required.
* It intercepts the response from the servlet back to the client and modifies the request if required.
* There can be many filters forming a chain, in which case the output of one filter becomes an input to the next filter. Hence, various modifications can be performed on a single request and response.



**45.What are the functions of the Servlet container?**

The functions of the Servlet container are as follows:

* **Lifecycle management** : It manages the life and death of a servlet, such as class loading, instantiation, initialization, service, and making servlet instances eligible for garbage collection.
* **Communication support** : It handles the communication between the servlet and the Web server.
* **Multithreading support** : It automatically creates a new thread for every servlet request received. When the Servlet service() method completes, the thread dies.
* **Declarative security** : It manages the security inside the XML deployment descriptor file.
* **JSP support** : The container is responsible for converting JSPs to servlets and for maintaining them.