03: Servlet interview Q&A

Posted on August 24, 2014 by Arulkumaran Kumaraswamipillai

Q1. What is a Servlet? Is a Servlet inherently multi-threaded?
A1. A Servlet is a Java class that runs within a web container in an application server, servicing multiple client requests concurrently forwarded through the server and the web container. The web browser establishes a socket connection to the host server in the URL, and sends the HTTP request. Servlets can forward requests to other servers and Servlets and can also be used to balance load among several servers. Servlet 3.0 allowed **asynchronous request processing** but only traditional I/O was permitted. In 3.1, asynchronous request processing is possible with NIO (i.e. non blocking IO).

HttpServlet spawns a lightweight Java thread to handle each HTTP request. Single copy of a type of HttpServlet but N number of threads (thread sizes can be configured in an application server). So, you need to code in a thread-safe manner.

Q. What are the different types of Servlets?
A.

GenericServlet	HttpServlet
A GenericServlet has a service( ) method to handle requests.	The HttpServlet extends GenericServlet and adds support for HTTP protocol based methods like doGet(), doPost(), doHead() etc. All client requests are handled through the service() method. The service method dispatches the request to an appropriate method like doGet(), doPost() etc to handle that request. HttpServlet also has methods like doHead(), doPut(), doOptions(), doDelete(), and doTrace().
Protocol independent. GenericServlet is for Servlets that might not use HTTP (for example FTP service).	Protocol dependent (i.e. HTTP).

Q. Which protocol is used to communicate between a browser and an HttpServlet?
A. A browser and a HttpServlet communicate using the HTTP protocol, which is a stateless request/response based protocol.

Q. What are the two objects an HttpServlet receives when it accepts a call from its client?
A. A "*ServletRequest*", which encapsulates client request from the client and the "*ServletResponse*", which encapsulates the communication from the Servlet back to the client.

SQL, XML, UML, JSO
Hadoop & BigData Int
Java Architecture Inte
Scala Interview Q&As
Spring, Hibernate, & M
Testing & Profiling/Sar
Other Interview Q&A f
▶Free Java Interview

16 Technical Key Areas

open all | close all
Best Practice (6)
Coding (26)
Concurrency (6)
Design Concepts (7)
Design Patterns (11)
Exception Handling (3
Java Debugging (21)
Judging Experience In
Low Latency (7)
Memory Management
Performance (13)
QoS (8)
Scalability (4)
SDLC (6)
Security (13)
Transaction Managem

80+ step by step Java Tutorials

open all | close all
Setting up Tutorial (6)
Tutorial - Diagnosis (2
Akka Tutorial (9)
Core Java Tutorials (2

In addition to both HTTP request and response, HTTP headers are informational additions that convey both essential and non-essential information. For example: HTTP headers are used to convey MIME (Multipurpose Internet Mail Extension) type of an HTTP request and also to set and retrieve cookies etc.

```
1 Content-Type: text/html
2 Set-Cookie:AV+USERKEY=AVSe5678f6c1tgfd;expires=Mon
3
4 response.setContentType(“text/html”);
5 response.addCookie(myCookie);
6
```

Q. What is a MIME type?
A. The “Content-Type” request or response header is known as MIME type. Server sends the MIME type to client to inform the type of data (e.g. XML, HTML, JSON, CSV, etc) it is sending, and client notifies the server its content type.

Q2. What is new in Servlet 3.x compared to Servlet 2.5?
A2.

1. Servlets 3.0 have come up with a set of **new Annotations** for the declarations of Servlet Mappings, Init-Params, Listeners, and Filters to make the code more readable by making the use of Deployment Descriptor (web.xml) absolutely optional.

```
1  package com.myapp;
2
3  @WebServlet(
4    asyncSupported = false,
5    name = "AccountingServlet",
6    urlPatterns = { "/acount" },
7    initParams = {
8      @WebInitParam(name = "param1", value = "value
9      @WebInitParam(name = "param2", value = "value
10   }
11 )
12 public class AccountingServlet extends HttpServle
13       //...
14 }
15
```

```
1 package com.myapp;
2
3 @WebFilter(filterName="countFilter", urlPattern={"
4
```

Hadoop & Spark Tutor
JEE Tutorials (19)
Scala Tutorials (1)
Spring & HIbernate Tu
Tools Tutorials (19)
Other Tutorials (45)

100+ Java pre-interview coding tests

open all | close all
Can you write code? (
♦ Complete the given
Converting from A to E
Designing your classe
Java Data Structures
Passing the unit tests
What is wrong with thi
Writing Code Home As
Written Test Core Java
Written Test JEE (1)

How good are your .....?

open all | close all
Career Making Know-
Job Hunting & Resum

```
5 public class CountFilter extends Filter {
6     ...
7 }
8
```

```
1 package com.myapp;
2
3 @WebListener
4 public class PaymentsListener extends ServletConte
5     //....
6 }
7
```

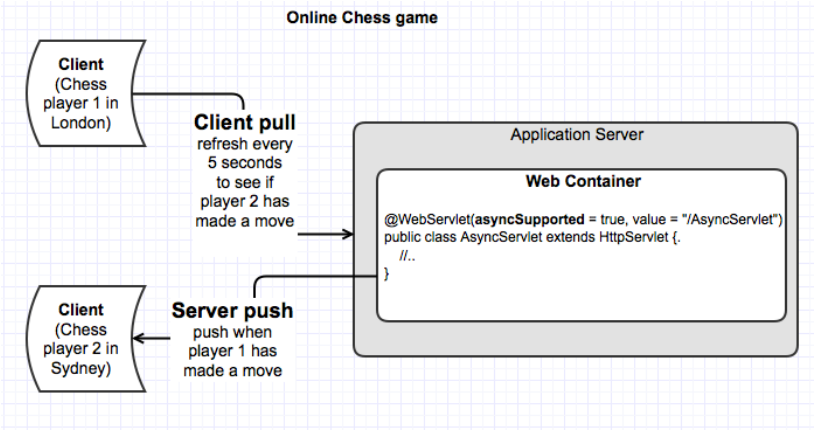
is better than declaring it verbosely in the web.xml as

```
 1  <web-app xmlns="http://java.sun.com/xml/ns/javaee
 2      <servlet>
 3          <servlet-name>accountingServlet</servlet-
 4          <servlet-class>com.myapp.AccountingServle
 5          <init-param>
 6              <param-name>param1</param-name>
 7              <param-value>value1</param-value>
 8          </init-param>
 9          <init-param>
10              <param-name>param2</param-name>
11              <param-value>value2</param-value>
12          </init-param>
13      </servlet>
14
15      <servlet-mapping>
16          <servlet-name> accountingServlet </servle
17          <url-pattern>/account/*</url-pattern>
18      </servlet-mapping>
19
20      <filter>
21          <filter-name>countFilter</filter-name>
22          <filter-class>com.myapp.CountFilter </fil
23      </filter>
24
25
26      <!-- Log for all URLs that use the "accountin
27      <filter-mapping>
28          <filter-name> countFilter </filter-name>
29          <servlet-name> accountingServlet </servle
30      </filter-mapping>
31
32      <listener>
33          <listener-class>com.myapp.PaymentsListene
34      </listener>
35
36  </web-app>
37
```

2. Built-in support for **File Upload with the @MultipartConfig** annotation. Having this annotation on the top of a servlet indicates that the Request that the Servlet is expecting will have **multipart/form-data** MIME type. In 2.x, you need to use

a third-party framework like Struts or Spring MVC to upload files.

3. Server push and **client pull** are techniques used to initiating delivery of content from a server to the client. In 2.x, you need to perform a client pull, which get the browser to request for an updated page in 10 seconds from the server.



Client pull Vs Server push

```
1 response.setHeader(“Refresh”, 10);
```

or

```
1 respose.setHeader(“Refresh”, “10;URL=http://localh
```

or in the section of the HTML

```
1 <META HTTP-EQUIV=”Refresh” CONTENT=”5; URL=http://
2
```

An example for the client pull would be playing an online game of chess. You need to know if the opponent on another machine, in an another country has made his or her move by periodically refreshing your page.

Server push technique can make use of the Asynchronous support from Servlet 3.x.

```
@WebServlet(asyncSupported = true, value = "/Asyn
public class AsyncServlet extends HttpServlet {
  private static final long serialVersionUID = 1L

  @EJB
  private AsyncBean asyncBean;

  protected void doGet(HttpServletRequest request
    AsyncContext asyncContext = request.startAsyn
    asyncBean.doAsyncTask(asyncContext);
  }
}

```

```
@Stateless
public class AsyncBean {

    @Asynchronous
    public void doAsyncTask(AsyncContext asyncCont
        asyncContext.getResponse().getWriter().wri
    }
}

```

4. Web Fragments in Servlet 3.0 to modularize the web application. Applications can be broken down into various modules by packaging them into separate JARs and include these archives into the main application war inside WEB-INF/lib folder. Each jar can use its own web application framework like Struts 2.0, Spring MVC, JSF, etc. These frameworks come along with their own default Request Processors and/or a Controllers to be configured, and these can be configured in each jar file's **web-fragment.xml**, under the META-INF folder. When the application is deployed, the server also scans JARs within WEB-INF/lib of the war under the META-INF directory and looks for web.xml fragments. If found, it will load the configurations. The web fragments has a number of benefits such as

a) web.xml will be smaller in size and easier to maintain.
b) application structures can be modularized and the configurations can be encapsulated (e.g. for different frameworks like Struts2, Spring MVC, etc).
c) Loosely couples the modules from the main application, and web frameworks can be easily plugged in and out.

Define the module1-web.jar with **META-INF/web-fragment.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<web-fragment xmlns:xsi="http://www.w3.org/2001/X
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:web="http://java.sun.com/xml/ns/javaee/
    xmlns:webfragment="http://java.sun.com/xml/ns
    xsi:schemaLocation="http://java.sun.com/xml/n
    http://java.sun.com/xml/ns/javaee/web-fragmen
    id="WebFragment_ID" version="3.0">

    <display-name>accounting</display-name>
    <name>mye</name>
    <servlet>
        <servlet-name>accountingServlet</servlet-
        <servlet-class>com.myapp.AccountingServle
    </servlet>
</web-fragment>

```

Define the main web application (i.e. war) in the WEB-INF/web.xml by defining the Servlet mapping

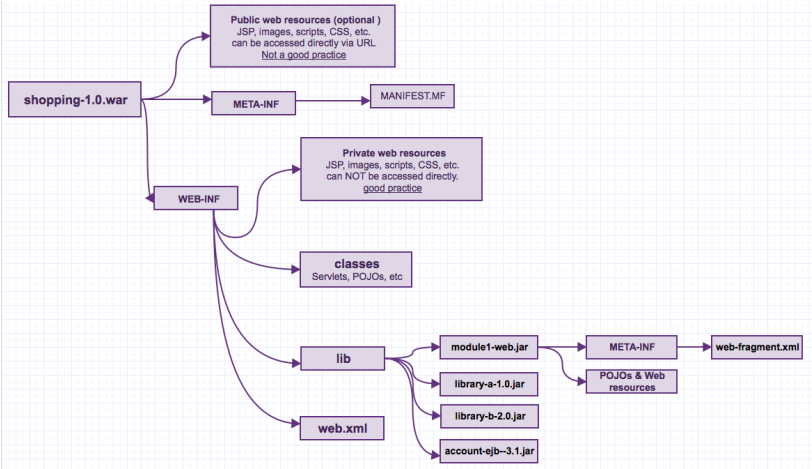
```
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSch
         xmlns="http://java.sun.com/xml/ns/javaee
         xmlns:web="http://java.sun.com/xml/ns/ja
         xsi:schemaLocation="http://java.sun.com/
         http://java.sun.com/xml/ns/javaee/web-ap
         id="WebApp_ID" version="3.0">

     <display-name>module1</display-name>
     <servlet-mapping>
         <servlet-name>accountingServlet</servlet
         <url-pattern>/account/*</url-pattern>
     </servlet-mapping>
</web-app>

```

5. Servlet 3.0 has programmatic support for adding web components. So, Servlets, filters, and Servlet/Filter mappings can be added during the runtime.

Q3. What is the structure of a war (i.e. Web ARchive) file?
A3. The resources directly under WAR (aka the “document root”) are know as the public resources. Resources under WEB-INF are known as the private resources. Resources that reside directly or under sub directories of the document root are directly accessible to the user through the URL. If you want to protect your Web resources then hiding the files like JSPs behind the WEB-INF directory can protect the files from direct access.



Web ARchive (i.e. WAR) structure

Q4. HTTP is a stateless protocol, hence how will you maintain state? What are the pros and cons of each approach?

A4.

State mechanism	Description and Pros/Cons
HttpSession	— There is no limit on the size of the session data kept. — The performance is good. — This is the preferred way of maintaining state. If we use the HTTP session with the application server's persistence mechanism (server converts the session object into BLOB type and stores it in the Database) then the performance will be moderate to poor. When using HttpSession mechanism you need to take care of the following points: — Remove session explicitly when you no longer require it. — Set the session timeout value. — Your application server may serialize session objects after crossing a certain memory limit. This is expensive and

	affects performance. So decide carefully what you want to store in a session.
Hidden fields	— There is no limit on size of the session data. — May expose sensitive or private information to others (So not good for sensitive information). — The performance is moderate.
URL rewriting	— There is a limit on the size of the session data. — Should not be used for sensitive or private information. — The performance is moderate.
Cookies	— There is a limit for cookie size. — The browser may turn off cookies. — The performance is moderate. The benefit of the cookies is that state information can be stored regardless of which server the client talks to and even if all servers go down. Also, if required, state information can be retained across sessions.

Q5. Explain the life cycle methods of a Servlet?
A5. The Web container is responsible for managing the Servlet's life cycle. The Web container creates an instance of the Servlet and then the container calls the init() method. At the completion of the init() method the Servlet is in ready state to service requests from clients. The container calls the Servlet's service() method for handling each request by spawning a new thread for each request from the Web container's thread pool. Before destroying the instance the container will call the destroy() method. After destroy() the Servlet becomes the potential candidate for garbage collection.

Q. What would be an effective use of the Servlet init() method?
A. One effective use of the Servlet init() method is the creation and caching of thread-safe resource acquisition mechanisms such as JDBC DataSources, EJB Homes, and Web Services SOAP Mapping Registry.

Q6. What is the difference between doGet () and doPost () or GET and POST?
A6. Prefer using doPost() because it is secured and it can send much more information to the server.

GET or doGet()	**POST or doPost()**

The request parameters are transmitted as a query string appended to the request. All the parameters get appended to the URL in the address bar. Allows browser bookmarks but not appropriate for transmitting private or sensitive information.
 	
`1 http://MyServer:8080/MyServlet?name=Paul`
 	
This is a security risk. In an HTML you can specify as follows:
 	
`1 <form name="SSS" method="GET" >`
 	
GET was originally intended for static resource retrieval.
	The request parameters are passed with the body of the request.
	
	More secured. In HTML you can specify as follows:
	
	`1 <form name="SSS" method="POST" >`
	
	POST was intended for form submits where the state of the model and database are expected to change.

GET is not appropriate when large amounts of input data are being transferred. Limited to 1024 characters.	Since it sends information through a socket back to the server and it won't show up in the URL address bar, it can send much more information to the server. Unlike doGet(), it is not restricted to sending only textual data. It can also send binary data such as serialized Java objects.

Q7. If you want a servlet to take the same action for both GET and POST request, what would you do?

A7. You should have doGet call doPost, or vice versa.

```
protected void doPost(HttpServletRequest req, Htt
            throws ServletException, IOException

      ServletOutputStream out =  resp.getOutputStr
      out.setContentType(“text/html”);
      out.println("<html><h1>Output to Browser</h1
      out.println("<body>Written as html from a Se
  }

  protected void doGet(HttpServletRequest req, Htt
            throws ServletException, IOException
      doPost(req, resp); //call doPost() for flow
  }
```

Q. How do you create a deadlock situation in a Servlet?

A. Create a cyclic call by getting doGet(..) to invoke doPost(..), and doPost(..) to invoke doGet(..).

Q8. What are the ServletContext and ServletConfig objects?

A8. The Servlet Engine uses both interfaces. The servlet engine implements the ServletConfig interface in order to pass configuration details from the deployment descriptor (web.xml) to a servlet via its init() method.

```
public class CRMServlet extends HttpServlet {
    //initializes the servlet
    public void init(ServletConfig config)throws S
        super.init(config);
    }
}
```

ServletConfig	ServletContext
The ServletConfig parameters are for a particular Servlet. The parameters are specified in the web.xml (i.e. deployment descriptor) or via annotation. It is created after a Servlet is	The ServletContext parameters are specified for the entire Web application. The parameters are specified in the web.xml (i.e. deployment descriptor). Servlet context is common

instantiated and it is used to pass initialization information to the Servlet.	to all Servlets. So all Servlets share information through ServletContext with setAttribute(…) and getAttribute(….) methods.

A ServletContext object is initialized once while a web application is being deployed, and this object can be accessed from anywhere within the same web application.

```
1 String strCfgPath = getServletConfig().getInitPara
2 String strServletName = getServletConfig().getServ
3
4 String strClassName = getServletContext().getAttri
5
```

Q.How can you invoke a JSP error page from a controller Servlet?
A. The following code demonstrates how an exception from a Servlet can be passed to an error JSP page.

```
1  protected void doGet(HttpServletRequest req, Http
2
3     try {
4        //doSomething
5     }
6     catch(Exception ex) {
7        req.setAttribute("javax.servlet.ex",ex);//s
8        ServletConfig sConfig = getServletConfig();
9        ServletContext sContext = sConfig.getServle
10       sContext.getRequestDispatcher("/jsp/ErrorPa
11       //request with the exception stored as an a
12       ex.printStackTrace();
13    }
14 }
15
```

Q9. What are the Servlet life cycle events?
A9. Servlet lifecycle events work like the Swing events. Any listener interested in observing the ServletContext lifecycle can implement the **ServletContextListener** interface and in the ServletContext attribute lifecycle can implement the ServletContextAttributesListener interface. The session listener model is similar to the ServletContext listener model (Refer Servlet spec 2.3 or later). Servlet contexts' and and sessions' listener objects are notified when servlet contexts

and sessions are initialized and destroyed, as well as when attributes are added or removed from a context or session.

Real life examples of using a ServletContextListener

Example 1: A typical example where a ServletContextListener is used is in bootstarpping a spring applicationContet.xml file into a web application that is using RESTEasy for RESTful web services.

Example 2: Another scenario that I came across recently where I had to provide my own implementation of the ServletContextListener when using yammer metrics to provide web based monitoring.

You can also define your own custom listeners. The server creates an instance of the listener class to receive events and uses introspection to determine what listener interface (or interfaces) the class implements.

```
1  @WebListener
2  public class MyJDBCConnectionManager implements S
3
4    public void contextInitialized(ServletContextEv
5         Connection con  = // create a connection
6         event.getServletContext().setAttribute("co
7    }
8
9    public void contextDestroyed(ServletContextEven
10        Connection con = (Connection) e.getServlet
11        try { con.close(); } catch (SQLException i
12   }
13 }
14
```

Q10. What is the difference between request parameters and request attributes?

A10. **Request parametersRequest attributes**

Parameters are form data that are sent in the request from the HTML page. These parameters are generally form fields in an HTML form like:

```
1 <input type=”text” name=”param1” />
2 <input type=”text” name=”param2” />
```

Form data can be attached to the end of the URL as shown below for GET requests

```
1 http://MyServer:8080/MyServlet? param1=Peter&param
```

or sent to the sever in the request body for POST requests. Sensitive form data should be sent as a POST request. Once a servlet gets a request, it can add additional attributes, then forward the request off to other servlets or JSPs for processing. Servlets and JSPs can communicate with each other by setting and getting attributes.

```
1 request.setAttribute(“calc-value”, new Float(7.0))
2 request.getAttribute(“calc-value”);
```

You can get them but cannot set them.

```
1 request.getParameter("param1");
2 request.getParameterNames();
3
```

You can both set the attribute and get the attribute. You can also get and set the attributes in session and application scopes.

Q. What are the different scopes or places where a servlet can save data for its processing?
A. Data saved in a request-scope goes out of scope once a response has been sent back to the client (i.e. when the request is completed).

```
1 //save and get request-scoped value
2 request.setAttribute(“calc-value”, new Float(7.0))
3 request.getAttribute(“calc-value”);
4
```

Data saved in a session-scope is available across multiple requests. Data saved in the session is destroyed when the session is destroyed (not when a request completes but spans several requests).

```
1 //save and get session-scoped value
2 HttpSession  session = request.getSession(false);
3 If(session != null) {
4     session.setAttribute("id", "DX12345");
5     value = session.getAttribute("id");
6 }
7
```

Data saved in a ServletContext scope is shared by all servlets and JSPs in the context. The data stored in the servlet context is destroyed when the servlet context is destroyed.

```
1 //save and get an application-scoped value
2 getServletContext().setAttribute("application-valu
3 value = getServletContext().getAttribute("applicat
4
```

Q11. What is the difference between using getSession(true) and getSession(false) methods?
A11.

getSession(true): This method will check whether there is already a session exists for the user. If a session exists, it returns that session object. If a session does not already exist then it creates a new session for the user.

getSession(false): This method will check whether there is already a session exists for the user. If a session exists, it returns that session object. If a session does not already exist then it returns null.

Q12. How can you set a cookie and delete a cookie from within a Servlet?
A12.

```
1  //to add a cookie
2  Cookie myCookie = new Cookie("aName", "aValue");
3  response.addCookie(myCookie);
4
5  //to delete a cookie
6  myCookie.setValue("aName", null);
7  myCookie.setMax(0);
8  myCookie.setPath("/");
9  response.addCookie(myCookie);
10
```

Q13. What is a RequestDispatcher? What object do you use to forward a request?

A13. A Servlet can obtain its RequestDispatcher object from its ServletContext.

```
1  //inside the doGet() method
2  ServletContext sc = getServletContext();
3  RequestDispatcher rd = sc.getRequestDispatcher("/
4
5  //forwards the control to another servlet or JSP
6
7  rd.forward(request,response);
8
9  // or includes the content of the resource such a
10 // calling Servlet's response.
11
12 rd.include(request, response);
13
```

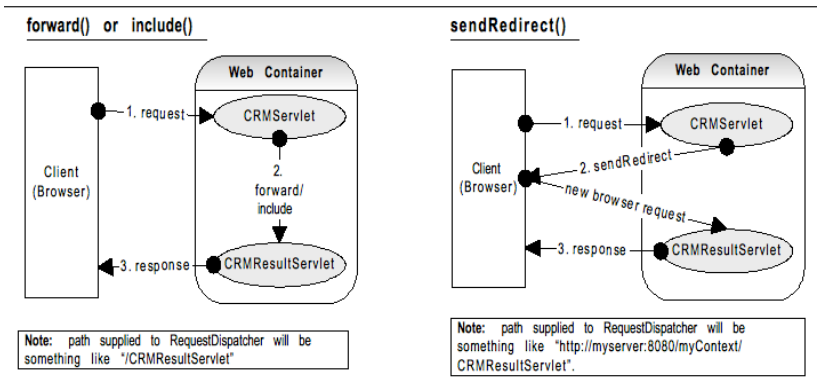
Q. What is the difference between the getRequestDispatcher(String path) method of "ServletRequest" interface and ServletContext interface?

A. javax.servlet.**ServletRequest**.*getRequestDispatcher(String path)* accepts path parameter of the servlet or JSP to be included or forwarded relative to the request of the calling servlet. If the path begins with a "/" then it is interpreted as relative to current context root.

javax.servlet.**ServletContext**.*getRequestDispatcher(String path)* does not accept relative paths and all path must start with a "/" and are interpreted as relative to current context root.

Q14. What is the difference between forwarding a request and redirecting a request?

A14. Both methods send you to a new resource like Servlet, JSP etc, but there is a key difference.



forward Vs sendRedirect

Forward action takes place within the server without the knowledge of the browser. Accepts relative path to the servlet or context root. So, there is no extra network trip.

sendRedirect sends a header back to the browser, which contains the name of the resource to be redirected to. The browser will make a fresh request from this header information. Need to provide absolute URL path. This has an overhead of extra remote trip but has the advantage of being able to refer to any resource on the same or different domain and also allows book marking of the page.

Q15. What are the considerations for Servlet clustering?
A15. The clustering promotes high availability and scalability. The considerations for servlet clustering are:

1) Objects stored in a session should be **serializable** to support in-memory replication of sessions. Also consider the overhead of serializing very large objects. Test the performance to make sure it is acceptable.
2) Design for **idempotence**. Failure of a request or impatient users clicking again can result in duplicate requests being submitted. So the Servlets should be able to tolerate duplicate requests.
3) Avoid using instance and static variables in read and write mode because different instances may exist on different JVMs. Any state should be held in an external resource such as a database.
4) Avoid storing values in a ServletContext. A ServletContext is not serializable and also the different instances may exist in

different JVMs.
5) Avoid using java.io.* because the files may not exist on all backend machines. Instead use ***getResourceAsStream()***.

Q16. What is pre-initialization of a Servlet?
A16. By default the container does not initialize the Servlets as soon as it starts up. It initializes a Servlet when it receives a request for the first time for that Servlet. This is called **lazy loading**. The servlet deployment descriptor (web.xml) defines the element, which can be configured 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 pre-loading or pre-initializing a servlet. We can also specify the order in which the servlets are initialized.

```
1 <load-on-startup>2</load-on-startup>
```

Q17. How to perform I/O operations in a Servlet/JSP?
A17. **Problem:** Since web applications are deployed as WAR files on the application server's web container, the full path and relative paths to these files vary for each server.

Solution -1: You can configure the file paths in web.xml using tags or via annotation @WebInitParam to retrieve file paths in your Servlets/JSPs.

Solution -2: You can overcome these configuration issues by using the features of *java.lang.ClassLoader* and *javax.servlet.ServletContext* classes. There are various ways of reading a file using the ServletContext API methods such as *getResource(String resource)*, *getResourceAsStream(String resource)*, *getResourcePaths(String path)* and *getRealPath(String path)*. The getRealPath(String path) method translates virtual URL into real path.

```
1 //Get the file “products.xml” under the WEB-INF fo
2 InputStream is = config.getServletContext().getRes
3
```

Alternatively you can use the APIs from ClassLoader as follows. The file “products.xml” should be placed under WEB-

INF/classes directory where all web application classes reside.

```
1 //Get the URL for the file and create a stream exp
2 URL url = config.getServletContext().getResource("
3 BufferedReader br = new BufferedReader(new InputSt
4
```

OR

```
1 //use the context class loader
2 URL url  = Thread.currentThread().getContextClassL
3 BufferedWriter bw = new BufferedWriter(new FileWri
4
```

Q18. How do you send a file to a browser from your Servlet? i.e. Download a file

A18. Files can be downloaded from a web application by using the right combination of headers.

```
1 response.setContentType("application/x-download");
```

use Content-Disposition "**attachment**" to invoke "Save As" dialog and "**inline**" for displaying the file content on the browser without invoking the "Save As" dialog.

```
1 response.setHeader("Content-disposition", "attachm
```

Q19. How do you send a file from a browser to your web application? i.e. Upload a file

A19. There are better and more secured ways to upload your files instead of using the Web. For example FTP, secure FTP etc. But if you need to do it via your web application then your default encoding and GET methods are not suitable for file upload and a form containing file input fields must specify the encoding type "**multipart/form-data**" and the **POST** method in the

tag as shown below:

```
1 <form enctype="multipart/form-data" method="POST"
2       <input type="file" name="products" />
```

```
3       <input type="submit" name="Upload" value="upl
4  </form>
```

When the user clicks the “Upload” button, the client browser locates the local file and sends it to the server using HTTP POST. When it reaches your server, your implementing servlet should process the POST data in order to extract the encoded file. In Servlet 3.0, you can use **@MultipartConfig** annotation.

```
1  @WebServlet(name = "StudentRegistrationUsn", urlPa
2  @MultipartConfig(maxFileSize = 10*1024*1024, maxRe
3  public class ActionRegistrationServlet extends Htt
4
5      protected void doPost(HttpServletRequest reque
6          //handle file upload
7      }
8  }
9
```

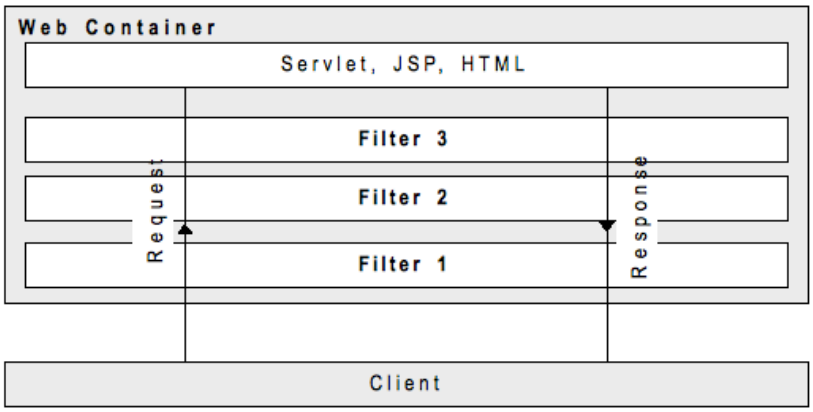
If using Servlet 2.5, there are a number of libraries available such as Apache Commons File Upload, which is a small Java package that lets you obtain the content of the uploaded file from the encoded form data. The API of this package is flexible enough to keep small files in memory while large files are stored on disk in a “temp” directory. You can specify a size threshold to determine when to keep in memory and when to write to disk.

Q20. If an object is stored in a session and subsequently you change the state of the object, will this state change replicated to all the other distributed sessions in the cluster?
A20. No. Session replication is the term that is used when your current service state is being replicated across multiple application instances. Session replication occurs when we replicate the information (i.e. session attributes) that are stored in your HttpSession. The container propagates the changes only when you call the **setAttribute(……) method**. So mutating the objects in a session and then by-passing the *setAttribute(……)* will not replicate the state change.

Example: If you have an ArrayList in the session representing shopping cart objects and if you just call getAttribute(…) to retrieve the ArrayList and then add or change something without calling the setAttribute(…) then the container may not

know that you have added or changed something in the ArrayList. So, the session will not be replicated.

Q21. What is a Servlet filter, and how does it work?
A21. A filter dynamically intercepts requests and responses to transform or use the information contained in the requests or responses but typically do not themselves create responses. Filters can also be used to transform the response from the Servlet or JSP before sending it back to client. Filters improve re-usability by placing recurring tasks in the filter as a reusable unit.



Servlet Filter

A good way to think of Servlet filters is as a chain of steps that a request and response must go through before reaching a Servlet, JSP, or static resource such as an HTML page in a Web application.The filters can be used for caching and compressing content, logging and auditing, image conversions (scaling up or down etc), authenticating incoming requests, XSL transformation of XML content, localization of the request and the response, site hit count etc.

```
<web-app>
   <filter>
      <filter-name>HitCounterFilter</filter-name>
      <filter-class>myPkg.HitCounterFilter</filte
   </filter>

   <filter-mapping>
      <filter-name>HitCounterFilter</filter-name>
      <url-pattern>/usersection/*</url-pattern>
   </filter-mapping>
   ...
</web-app>
```

```
13
```

Design Pattern: Servlet filters use the slightly modified version of the chain of responsibility design pattern. Unlike the classic (only one object in the chain handle the request) chain of responsibility where filters allow multiple objects (filters) in a chain to handle the request. In Servlet 3.0, you can use the @WebFilter annotation instead of declaring it in the web.xml file.

Here is an example that logs the time request/response times.

```
1  import javax.servlet.*;
2
3  @WebFilter(filterName="timerFilter", urlPattern={
4  public class TimerFilter implements javax.servlet
5  {
6      private FilterConfig filterConfig;
7
8      public void doFilter(ServletRequest request,
9             FilterChain chain)
10            throws java.io.IOException, javax.serv
11     {
12         //on the way up the request chain
13         long start = System.currentTimeMillis();
14         System.out.println("Milliseconds in: " +
15         chain.doFilter(request, response);
16         //on the way down the response chain
17         long end = System.currentTimeMillis();
18         System.out.println("Milliseconds out: " +
19
20     }
21
22     public void init(final FilterConfig filterCon
23     {
24         this.filterConfig = filterConfig;
25     }
26
27     public void destroy()
28     {
29         filterConfig = null;
30     }
31 }
32
```

Popular Posts

♦ 11 Spring boot interview questions & answers

825 views

♦ Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers

766 views

18 Java scenarios based interview Questions and Answers

400 views

001A: ♦ 7+ Java integration styles & patterns interview questions & answers

388 views

01b: ♦ 13 Spring basics Q8 – Q13 interview questions & answers

295 views

♦ 7 Java debugging interview questions & answers

293 views

01: ♦ 15 Ice breaker questions asked 90% of the time in Java job interviews with hints

285 views

♦ 10 ERD (Entity-Relationship Diagrams) Interview Questions and Answers

279 views

♦ Q24-Q36: Top 50+ Core on Java classes, interfaces and generics interview questions & answers

239 views

001B: ♦ Java architecture & design concepts interview questions & answers

201 views

Bio | Latest Posts

Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

About Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

‹ 01: ♦ 12 Web basics every Java web developer must know

05: Web patterns interview Q&A ›

Posted in member-paid, Web basics

Tags: Java/JEE FAQs, JEE FAQs, Novice FAQs

Empowers you to open more doors, and fast-track

Technical Know Hows

☀ Java generics in no time ☀ Top 6 tips to transforming your thinking from OOP to FP ☀ How does a HashMap internally work? What is a hashing function?
☀ 10+ Java String class interview Q&As ☀ Java auto un/boxing benefits & caveats ☀ Top 11 slacknesses that can come back and bite you as an experienced Java developer or architect

Non-Technical Know Hows

☀ 6 Aspects that can motivate you to fast-track your career & go places ☀ Are you reinventing yourself as a Java developer? ☀ 8 tips to safeguard your Java career against offshoring ☀ My top 5 career mistakes

Prepare to succeed

☀ Turn readers of your Java CV go from “Blah blah” to “Wow”? ☀ How to prepare for Java job interviews? ☀ 16 Technical Key Areas ☀ How to choose from multiple Java job offers?

Select Category ▼

© Disclaimer

The contents in this Java-Success are copy righted. The author has the right to correct or enhance the current content without any prior notice.

These are general advice only, and one needs to take his/her own circumstances into consideration. The author will not be held liable for any damages caused or alleged to be caused either directly or indirectly by these materials and resources. Any trademarked names or labels used in this blog remain the property of their respective trademark owners. No guarantees are made regarding the accuracy or usefulness of content, though I do make an effort to be accurate. Links to external sites do not imply endorsement of the linked-to sites.

↑