<http://javarevisited.blogspot.sg/2011/09/servlet-interview-questions-answers.html>

Top 10 Servlet Interview Question Answers - J2EE

This time its servlet interview questions, I was thinking what to pick for my interview series and than I thought about J2EE and Servlet is my favorite on that space. Servlet is an important part of any J2EE development and serves as Controller on many web MVC frameworks and that’s why it’s quite popular on J2EE interviews. These Servlet questions are based on my experience as well as collected by friends and colleague and they are not only good for interview practice but also shows a new direction of learning for any one who is not very familiar with servlet technology.

As said earlier this interview question article is part of my earlier series [java interview questions](http://javarevisited.blogspot.com/2011/04/top-20-core-java-interview-questions.html) , [UNIX command interview questions](http://javarevisited.blogspot.com/2011/05/unix-command-interview-questions.html) and [Java threading interview questions](http://javarevisited.blogspot.com/2011/07/java-multi-threading-interview.html).

You can find answers of all these questions on google but I have also listed my answers fo r quick reference.

### Servlet Questions Asked in Interview

**Question 1: In web.xml file   <load-on-startup>1</load-on-startup> is defined between <servlet></servlet> tag what does it means.**

Ans**:** whenever we request for any servlet the servlet container will initialize the servlet and load it which is defined in our config file called web.xml by default it will not initialize when our context is loaded .defining like this <load-on-startup>1</load-on-startup> is also known as pre initialization of servlet means now the servlet for which we have define this tag has been initialized in starting when context is loaded before getting any request.When this servlet question was asked to me in an interview few years back , I was not even aware of this element but this questions pointed me to look DTD of web.xml and understand other elements as well.**.**

**Question 2: How can we create deadlock condition on our servlet?**

Ans: one simple way to call doPost() method inside doGet() and doGet()method inside doPost() it will create deadlock situation for a servlet. This is rather simple servlet interview questions but yet tricky if you don’t think of it 

**Question 3: For initializing a servlet can we use constructor in place of init ().**

Ans: No, we can not use constructor for initializing a servlet because for initialization we need an object of servletConfig using this object we get all the parameter which are defined in deployment descriptor for initializing a servlet and in servlet class we have only default constructor according to older version of java so if we want to pass a

Config object we don’t have parametrized constructor and apart from this servlet is loaded and initialized by container so ots a job of container to call the method according to servlet specification they have lifecycle method so init() method is called firstly.

More important Java doesn't allow interfaces to declare constructors. These kinds of *servlet interview questions* are quite popular on service based companies who just want to dig one level more.

**Question 4: Why super.init (config) wiil be the first statement inside init(config) method**.

Ans: This will be the first statement if we are overriding the init(config ) method by this way we will store the config object for future reference and we can use by getServletConfig ()  to get information about config object if will not do this config object will be lost and we have only one way to get config object because servlet pass config object only in init method . Without doing this if we call the servletConfig method will get **NullPointerException.**

**Question5: Can we call destroy() method inside the init() method is yes what will happen?**

Ans:Yes we can call like this but  if we have not override this method container will call the default method and nothing will happen.after calling this if any we have override the method then the code written inside is executed.

**Question 6: How can we refresh servlet on client and server side automatically?**

Ans: On client side we can use Meta http refresh and server side we can use server push.

**Question 7: How can you get the information about one servlet context in another servlet**?

Ans: In context object we can set the attribute which we want on another servlet and we can get that attribute using their name on another servlet.

Context.setAttribute (“name”,” value”)

Context.getAttribute (“name”)

**Question 8: Why we need to implement Single Thread model in case of Servlet.**

Ans: In J2EE we can implement our servlet on two different ways either by using:

1. Single Thread Model

2. Multithread Model

Depending upon our scenario, if we have implemented single thread means only one instance is going handle one request at a time no two thread will concurrently execute service method of servlet.

**Example** in banking account where sensitive data is handle mostly this scenario was used this interface is deprecated in Servlet API version 2.4.

As the name signifies multi thread means a servlet is capable to handle multiple requests at same time. This servlet interview question was quite popular few years back on entry level but now its loosing its shine.

**Question 9: what is servlet collaboration?**

Ans communication between two servlet is called servlet collaboration which is achieved by 3 ways.

**1. RequestDispatchers include () and forward() method .**

**2. Using** [**sendRedirect()**](http://javarevisited.blogspot.com/2011/09/sendredirect-forward-jsp-servlet.html)**method of Response object.**

**3. Using servlet Context methods**

**Question 10: What is the difference between ServletConfig and ServletContext?**

Ans: **ServletConfig** as the name implies provide the information about configuration of a servlet which is defined inside the web.xml file or we can say deployment descriptor.its a specific object for each servlet.

**ServletContext** is application specific object which is shared by all the servlet belongs to one application in one JVM .this is single object which represent our application and all the servlet access application specific data using this object.servlet also use their method to communicate with container.

These Servlet interview questions are good for quick recap of important concept before appearing on any J2EE interview. Please share if you have come across any other interesting interview question on Servlets.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<http://www.codestyle.org/java/servlets/faq-Threads.shtml>

<https://docs.oracle.com/cd/E19798-01/821-1841/bnafd/index.html>

[**http://www.sql.ru/forum/943659/kolichestvo-instansov-servleta-v-tomcat**](http://www.sql.ru/forum/943659/kolichestvo-instansov-servleta-v-tomcat)

<http://www.sql.ru/forum/969093/kak-sozdaetsya-servlet-v-tomkate>

|  |
| --- |
| **Servlet-3\_0-final-spec.pdf** |
| 2.2. Number of instances. ... For a servlet **not hosted in a distributed** environment (the default), the servlet container **must use only one instance** per servlet declaration. However, for a servlet implementing the SingleThreadModel interface, the servlet container may instantiate multiple instances to handle a heavy request load and serialize requests to a particular instance. |

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<http://stackoverflow.com/questions/2183974/difference-between-each-instance-of-servlet-and-each-thread-of-servlet-in-servle>

Although there are already a few good answers, none of them spoke about a Java web application deployed in a distributed environment. This is a practical scenario where actually multiple instances of a single servlet are created. In a distributed environment you have a cluster of machines to handle the request and the request can go to any of these machines. Each of these machines should be capable to handle the request and hence every machine should have an instance of your MyAwesomeServlet in it's JVM.

So, the correct statement would be there is only one instance per JVM for every servlet, unless it implements SingleThreadModel.

SingleThreadModel in simple words says that you have to have only one thread per instance of Servlet, so basically you need to create one instance per coming request to handle it, which basically kills the whole concept of handling requests in a parallel fashion and isn't considered a good practice as the servlet object creation and initialization takes up time before it's ready to process the request.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<https://docs.oracle.com/javaee/5/api/javax/servlet/SingleThreadModel.html>

Ensures that servlets handle only one request at a time. This interface has no methods.

If a servlet implements this interface, you are *guaranteed* that no two threads will execute concurrently in the servlet's service method. The servlet container can make this guarantee by synchronizing access to a single instance of the servlet, or by maintaining a pool of servlet instances and dispatching each new request to a free servlet.

Note that SingleThreadModel does not solve all thread safety issues. For example, session attributes and static variables can still be accessed by multiple requests on multiple threads at the same time, even when SingleThreadModel servlets are used. It is recommended that a developer take other means to resolve those issues instead of implementing this interface, such as avoiding the usage of an instance variable or synchronizing the block of the code accessing those resources. This interface is deprecated in Servlet API version 2.4.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<http://docstore.mik.ua/orelly/xml/jxslt/ch06_05.htm>

This fixes our problem because each thread gets its own copy of local variables in Java. By simply removing the object field and replacing it with a local variable, this particular threading problem is resolved.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<http://stackoverflow.com/questions/6851111/how-to-configure-the-maximum-pool-size-of-servlets-implementing-singlethreadmode>

There's somewhere a major misunderstanding going on. There's usually only **one** instance of a mapped servlet during the entire application's lifetime. Perhaps your servlet implements the (since 2003 deprecated) SingleThreadModel interface? (which on Tomcat indeed has a default limit of 20 instances). If this is true, just get rid of that deprecated interface and rewrite your servlet so that it's threadsafe. Related:

<http://stackoverflow.com/questions/3106452/how-do-servlets-work-instantiation-shared-variables-and-multithreading/3106909#3106909>

Q: Suppose, I have a webserver which holds numerous Servlets. For information passing among those Servlets I am getting the Servlets context and setting session variables.

Now, if 2 or more users send request to this server then what happens to the session variables? Will they all be common for all the users or they will be different for each user. If they are different, then how was the server able to differentiate between different users?

One more similar question, if there are \*n\* users accessing a particular Servlets, then this Servlets gets instantiated only the first time the first user accessed it or does it get instantiated for all the users separately?

Answer:

## ServletContext

When the servletcontainer (like [Apache Tomcat](http://tomcat.apache.org)) starts up, it will deploy and load all webapplications. When a webapplication get loaded, the servletcontainer will create the [ServletContext](http://docs.oracle.com/javaee/7/api/javax/servlet/ServletContext.html) once and keep in server's memory. The webapp's web.xml will be parsed and every <servlet>, <filter> and <listener> found in web.xml, or annotated with respectively @WebServlet, @WebFilter and @WebListener, will be created once and kept in server's memory as well. For all filters, the init() method will also be invoked immediately. When the servletcontainer shuts down, it will unload all webapplications, invoke the destroy() of all initialized servlets and filters, and finally the ServletContext and all Servlet, Filter and Listener instances will be trashed.

When the Servlet in question has a <servlet><load-on-startup> or @WebServlet(loadOnStartup) value greater than 0, then its init() method will also immediately be invoked during startup. Those servlets are initialized in the same order as "load-on-startup" value represents, or if they are the same, then the order in the web.xml or @WebServlet classloading. Or, if the "load-on-startup" value is absent, then the init() method will only be invoked on very first HTTP request hitting the servlet in question.

## HttpServletRequest and HttpServletResponse

The servletcontainer is attached to a webserver which listens on HTTP requests on a certain port number, which is usually 8080 in development and 80 in production. When a client (user with a webbrowser) sends a HTTP request, the servletcontainer will create new [HttpServletRequest](http://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServletRequest.html) and [HttpServletResponse](http://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServletResponse.html) objects and pass it through the methods of the already-created Filter and Servlet instances whose url-pattern matches the request URL, all in the same thread.

The request object provides access to all information of the HTTP request, such as the request headers and the request body. The response object provides facility to control and send the HTTP response the way you want, such as setting headers and the body (usually with HTML content from a JSP file). When the HTTP response is committed and finished, then both the request and response objects will be trashed.

## HttpSession

When a client visits the webapp for the first time and/or the [HttpSession](http://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpSession.html) is to be obtained for the first time by request.getSession(), then the servletcontainer will create it, generate a long and unique ID (which you can get by session.getId()) and store it in server's memory. The servletcontainer will also set a [Cookie](http://docs.oracle.com/javaee/7/api/javax/servlet/http/Cookie.html) in the Set-Cookie header of the HTTP response with JSESSIONID as cookie name and the unique session ID as cookie value.

As per the [HTTP cookie specification](http://www.faqs.org/rfcs/rfc2965.html) (a contract a decent webbrowser and webserver has to adhere), the client (the webbrowser) is required to send this cookie back in the subsequent requests in the Cookie header as long as the cookie is valid. Using browser builtin HTTP traffic monitor you can check them (press F12 in Chrome / Firefox23+ / IE9+ and check Net/Network tab). The servletcontainer will determine the Cookie header of every incoming HTTP request for the presence of the cookie with the name JSESSIONID and use its value (the session ID) to get the associated HttpSession from server's memory.

The HttpSession lives until it has not been used for more than the <session-timeout> time, a setting you can specify in web.xml, which defaults to 30 minutes. So when the client doesn't visit the webapp anymore for over 30 minutes, then the servletcontainer will trash the session. Every subsequent request, even though with the cookie specified, will not have access to the same session anymore. The servletcontainer will create a new one.

On the other hand, the session cookie on the client side has a default lifetime which is as long as the browser instance is running. So when the client closes the browser instance (all tabs/windows), then the session will be trashed at the client side. In a new browser instance the cookie associated with the session won't be sent anymore. A new request.getSession() would return a brand new HttpSession and set a cookie with a brand new session ID.

## In a nutshell

* The ServletContext lives as long as the webapp lives. It's been shared among all requests in all sessions.
* The HttpSession lives as long as the client is interacting with the webapp with the same browser instance and the session hasn't timed out at the server side yet. It's been shared among all requests in the same session.
* The HttpServletRequest and HttpServletResponse lives as long as the client has sent it until the complete response (the webpage) is arrived. It is not being shared elsewhere.
* Any Servlet, Filter and Listener lives as long as the webapp lives. They are being shared among all requests in all sessions.
* Any attribute which you set in ServletContext, HttpServletRequest and HttpSession will live as long as the object in question lives.

## Threadsafety

That said, your major concern is possibly threadsafety. You should now have learnt that Servlets and filters are shared among all requests. That's the nice thing of Java, it's multithreaded and different threads (read: HTTP requests) can make use of the same instance. It would otherwise have been too expensive to recreate it on every request.

But you should also realize that you should **never** assign any request or session scoped data as an instance variable of a servlet or filter. It will be shared among all other requests in other sessions. That's threadunsafe! The below example illustrates that:

public class ExampleServlet extends HttpServlet {

private Object thisIsNOTThreadSafe;

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

Object thisIsThreadSafe;

thisIsNOTThreadSafe = request.getParameter("foo"); // BAD!! Shared among all requests!

thisIsThreadSafe = request.getParameter("foo"); // OK, this is thread safe.

}

}

### See also:

* [What is the difference between JSF, Servlet and JSP?](http://stackoverflow.com/questions/2095397/what-is-the-difference-between-jsf-servlet-and-jsp)
* [Best option for Session management in Java](http://stackoverflow.com/questions/1700390/best-option-for-session-management-in-java)
* [doGet and doPost in Servlets](http://stackoverflow.com/questions/2349633/doget-and-dopost-in-servlets)
* [Servlet seems to handle multiple concurrent requests synchronously](http://stackoverflow.com/questions/8011138/servlet-seems-to-handle-multiple-concurrent-requests-synchronously/)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

public interface ServletContext

Defines a set of methods that a servlet uses to communicate with its servlet container, for example, to get the MIME type of a file, dispatch requests, or write to a log file.

There is one context per "web application" per Java Virtual Machine. (A "web application" is a collection of servlets and content installed under a specific subset of the server's URL namespace such as /catalog and possibly installed via a .war file.)

In the case of a web application marked "distributed" in its deployment descriptor, there will be one context instance for each virtual machine. In this situation, the context cannot be used as a location to share global information (because the information won't be truly global). Use an external resource like a database instead.

The ServletContext object is contained within the [ServletConfig](http://docs.oracle.com/javaee/7/api/javax/servlet/ServletConfig.html) object, which the Web server provides the servlet when the servlet is initialized.

\*\*\*\*\*\*\*\*

public interface HttpSession

Provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user.

The servlet container uses this interface to create a session between an HTTP client and an HTTP server. The session persists for a specified time period, across more than one connection or page request from the user. A session usually corresponds to one user, who may visit a site many times. The server can maintain a session in many ways such as using cookies or rewriting URLs.

This interface allows servlets to

* View and manipulate information about a session, such as the session identifier, creation time, and last accessed time
* Bind objects to sessions, allowing user information to persist across multiple user connections

When an application stores an object in or removes an object from a session, the session checks whether the object implements [HttpSessionBindingListener](http://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpSessionBindingListener.html). If it does, the servlet notifies the object that it has been bound to or unbound from the session. Notifications are sent after the binding methods complete. For session that are invalidated or expire, notifications are sent after the session has been invalidated or expired.

When container migrates a session between VMs in a distributed container setting, all session attributes implementing the [HttpSessionActivationListener](http://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpSessionActivationListener.html) interface are notified.

A servlet should be able to handle cases in which the client does not choose to join a session, such as when cookies are intentionally turned off. Until the client joins the session, isNew returns true. If the client chooses not to join the session, getSession will return a different session on each request, and isNew will always return true.

Session information is scoped only to the current web application (ServletContext), so information stored in one context will not be directly visible in another.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*