JSF interview Q&A

Posted on September 8, 2014 by Arulkumaran Kumaraswamipillai — No Comments ↓

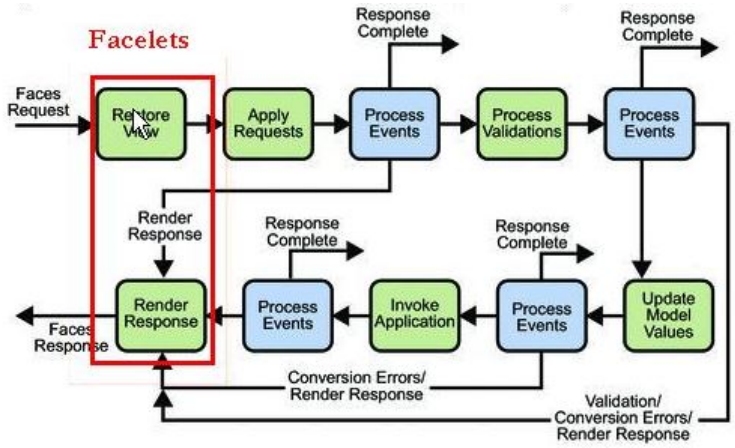
Q1. What are the 6 key phases of JSF? What other frameworks would you use with JSF, and in which phases do they fit in?

A1. JavaServer Faces (JSF) brings a new paradigm to developing Web-based applications in Java. However, it has a number of gaps in its out-of-the-box state. The frameworks such as **Facelets** and **JBoss seam** fill in these gaps.

Facelets are an alternative view technology based on pure XML templates (no scriptlets) which was introduced with Version 2 of the JSF standard. They can only be used in a JSF application. JSPs can be used with JSF for view, but unlike JSPs, Facelets is a templating language built from the ground up with the JSF component life cycle in mind.

With Facelets, you produce templates that build a component tree, not a servlet. This allows for greater reuse than JSPs

because you can compose components out of a composition of other components. Facelets tap into the “Restore View” and “Render Response” phases of the JSF life cycle as shown below. The 6 phases are in green.



JSF Phases

Seam integrates technologies such as Asynchronous JavaScript and XML (AJAX), JavaServer Faces (JSF), Java Persistence (JPA), Enterprise Java Beans (EJB 3.0) and Business Process Management (BPM) into a unified full-stack solution. Seam extends JSF with handful of annotations, extra functionality for multi-window operation and workspace management, conversational scopes to minimize HTTP session abuse, model-based validation, jBPM-based pageflow, internationalization, page fragment caching.and bookmarkable RESTful style web pages.

There are other useful optional frameworks like Tucky URL rewrite filter for RESTful URLs, JSFUnit for testing, JQuery for JavaScript, and JSF component libraries like RichFaces, ICEfaces, Trinidad, PrimeFaces, and Tomahawk for rich JSF components. But use these rich component libraries wisely as they could cause your generated HTML pages to be bloated. Bloated pages can take longer to render.

Q2. What is a viewstate in JSF?
A2. In JSF, there is a **viewstate associated with each page**, which is passed back and forth with each submits. The reason for the viewtate is that the **HTTP is a stateless**

- Hibernate Job Inter
- Spring Job Intervie
- Java Key Area Ess
- OOP & FP Essenti
- Code Quality Job I
- SQL, XML, UML, JSC
- Hadoop & BigData Int
- Java Architecture Inte
- Scala Interview Q&As
- Spring, Hibernate, & M
- Testing & Profiling/Sa
- Other Interview Q&A f
- Free Java Intervie

16 Technical Key Areas

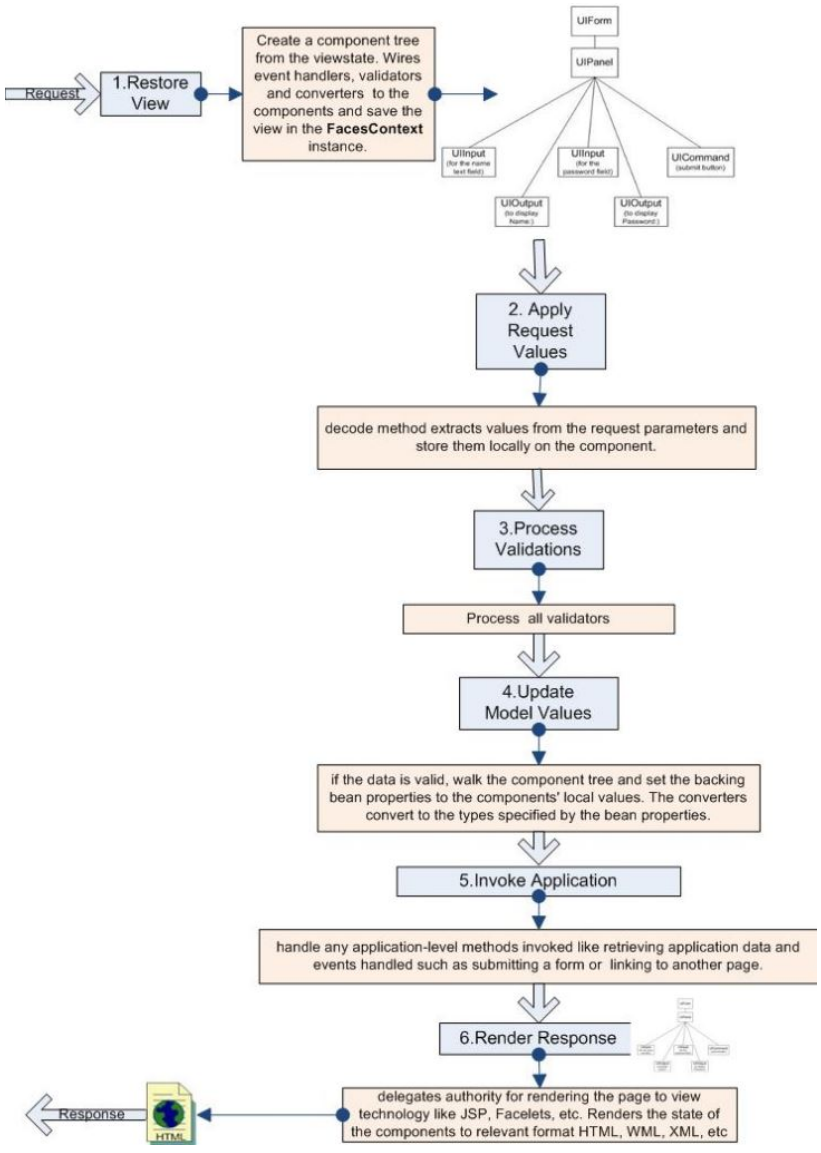
open all | close all
- Best Practice (6)
- Coding (26)
- Concurrency (6)
- Design Concepts (7)
- Design Patterns (11)
- Exception Handling (3
- Java Debugging (21)
- Judging Experience I
- Low Latency (7)
- Memory Managemen
- Performance (13)
- QoS (8)
- Scalability (4)
- SDLC (6)
- Security (13)
- Transaction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

80+ step by step Java Tutorials

protocol. The state of the components across requests need to be maintained. The viewstate can change in between requests as new controls like UIInput can be added or modified. The view state is divided into two parts.

Part 1: Structure of the components
Part 2: Defines the state of the components. For example. enabled/disabled, input values, checked/unchecked, selected item, etc.

Understanding the JSF lifecycle helps understand the viewstate.



JSF ViewState

open all | close all
Setting up Tutorial (6)
Tutorial - Diagnosis (2
Akka Tutorial (9)
Core Java Tutorials (2
Hadoop & Spark Tuto
JEE Tutorials (19)
Scala Tutorials (1)
Spring & HIbernate Tu
Tools Tutorials (19)
Other Tutorials (45)

It can be achieved by either storing the viewstate on the server side in a session and then passing the viewstate id to the client via a hidden field as shown below:

Server side: In web.xml file, set the state saving strategy:

```
1 <context-param>
2       <param-name>javax.faces.STATE_SAVING_METHOD<
3       <param-value>server</param-value>
4 </context-param>
```

The hidden field passed to the client

```
1 <input type="hidden" name="javax.faces.ViewState"
2    id="javax.faces.ViewState" value="cdx43wedsfdg
3
```

Client side: serializing the view state on the client. Do the following in web.xml

```
1 <context-param>
2       <param-name>javax.faces.STATE_SAVING_METHOD<
3       <param-value>client</param-value>
4 </context-param>
```

Q3. In your experience, what are some of the JSF pitfalls to watch out for?

A3. **#1. Putting business logic or data access logic in the getter methods**. Getters are solely there to access bean properties, not to do some business logic or retrieve data from a database. A getter will normally be called twice or three times per JSF request-response cycle, but it can get called more times, especially when used in UIData components. So, don't use a getter for other purposes than just returning the data.

Conditionally rendering, enabling/disabling, or making it editable or not, etc can get called many times in a data-table. For example, if the rendered attribute occurs in a single row of a data-table, then number of invocations of the backing method will be invoked by the number of rows in the table. In a table with 50 rows, it will be invoked 50 times. If it is used in

4 of the columns in each row, it will be invoked 50 * 4 = 200 times. So, it is imperative that the logic is efficient or else it will have a huge impact on performance.

#2. Using too many rich components from third-party libraries like Richfaces, ICEFaces, etc could adversely impact performance due to bloated HTML pages. JSF saves two things between requests:

– the view (all the HTML controls on the page like text fields, buttons, etc)
– the view state (the state of the controls)

So, having too many rich controls on a page could lead to HTML bloating, and adversely impact performance. This could be minimized by carefully designing the pages and using AJAX calls to render only a fragment of the HTML DOM (Document Object Model) tree. Also, care must be taken not to have multiple h:form elements on a page as this may result in a copy of the entire view state being included with every form.

#3. Having too many backing beans in session scope could lead to memory and performance issues. If you want to have book-markable URLs, you will need to perform redirects, and redirects will blow away the request parameters as new requests will be made. So, you must use the session scope. To prevent any potential issues due to large sessions, either provide your own implementation to periodically clear the session data or use frameworks like seam to take advantage of the conversational scope. If you use the “conversational” scope in Seam, the framework will manage the session data for you with the temporary conversation scope or you could manage it declaratively with the long-running conversational scope.

Put the components in a SEAM conversation scope, which is a slice of the HTTP session managed by the SEAM, and associated with a sequence of pages through the special token CID (i.e. Conversation ID). This conversation scope provides the developer the convenience of using the HTTP

session without the memory leakage concerns, since the conversation has a much shorter lifetime and is kept isolated from other parallel conversations in the same session. The conversation scope also has other benefits of

– ensuring that the records remain managed by the persistence context while being modified.
– reducing the number of calls to the database by making previously viewed result pages to load faster since the records are already in Hibernate persistent context (i.e. first-level cache).
– maintaining the pagination details like search criteria, current page number, page offset, etc.

#4. Not coding in a thread-safe manner. Calls to FacesContext.getCurrentInstance( ) return a thread local data structure, hence thread-safe. Request and unscoped managed beans are of course safe as well. But, Session and application scoped managed beans are obviously accessed by more than one thread. The PhaseListeners and Renderers are not thread safe. Each PhaseListener is global to the application and it subscribes to at least one phase event for every request. Components of the same type share the same Renderer instance for the entire application. When it comes to the converters and validators, the thread-safety depends on how they are used.

Following is thread-safe because a new instance will be created for every input element in view.

```
1 <h:inputText ...>
2     <f:converter converterId="converterId" />
3 </h:inputText>
4
```

Following is not threadsafe because the same instance will be shared across all views of the entire application

```
1 <h:inputText  converter="#{applicationBean.conver
```

#5. Not carefully designing the GUI and the relevant interactions. The JSF data centric applications involve presenting the data in a tabular format, filtering the data, paginating the data, selecting the data via check boxes, and modifying the data. It is imperative to properly think through the various interactions and properly performance test them. For example,

– Building a composite AJAX enabled component that allows partial page rendering can improve performance. This is provided by the Ajax4jsf component. Ajax4jsf provides a set of tag libraries that ties JSF generated event to the rendering of one or more regions of the user-interface identified by their JSF client IDs. When the AJAX event is sent to the server, instead of the JSF servlet returning an entire page response, it returns only fragments of the HTML to render a particular region.

– Thinking carefully about splashing rendered, editable, and disable method calls within each column and row in a data table versus having them on page level. For, example editable table versus non-editable, etc.

– Not utilizing the appropriate strategy to save the viewstate. Performance measurements have shown that plain server side state saving without serialization and compressing state comes with the best values. State saving at client side is also susceptible to security concerns as well as overhead of serialization of entire JSF tree every time. Having said that, the server side strategy will consume more memory. The numberOfViewsInSession parameter can be used to limit number of calls with back buttons inside a faces form to reduce memory consumption.

```
1  <context-param>
2      <param-name>com.sun.faces.numberOfViewsInSes
3      <param-value>3</param-value>
4  </context-param>
5
6  <context-param>
7      <param-name>com.sun.faces.numberOfLogicalVie
8      <param-value>10</param-value>
9  </context-param>
10
```

Q4. How would you go about debugging a JSF application?

A4. **#1. You can use a PhaseListener to trace the 6 phases of the JSF lifecycle** shown in the diagram above. You can use a “dummy” PhaseListener to debug the phases to see what is happening in each phase. Here is a basic example of such a LifeCycleListener you can use to trace the phases of the JSF lifecycle. For example, to better understand how the attribute immediate=true works.

```
1  package com.mypackage;
2
3  import javax.faces.event.PhaseEvent;
4  import javax.faces.event.PhaseId;
5  import javax.faces.event.PhaseListener;
6  import java.util.logging.Logger;
7
8  public class LifeCycleListener implements PhaseL
9
10     protected final Logger logger = Logger.getIn
11
12     public PhaseId getPhaseId() {
13         return PhaseId.ANY_PHASE;
14     }
15
16     public void beforePhase(PhaseEvent event) {
17         logger.info("START PHASE " + event.getPh
18     }
19
20     public void afterPhase(PhaseEvent event) {
21         logger.info("END PHASE " + event.getPhas
22     }
23 }
24
```

#2. Using the FacesTrace library. FacesTrace is an open-source library aiming to enhance the tracebility of applications based on JavaServer Faces. Performance metrics and general trace information of a JSF application is collected and presented in a user-friendly format. The simplest way to use the “runtime debugging” JSF tool FacesTrace is to place the tag at the end of a page. You will have to declare the tag first.

```
1 <%@ taglib uri="http://facestrace.sourceforge.net
```

#3. Configure your web.xml to run in development or debug modes. For example, if you are using Facelets with JSF, set the development mode to true.

```
<context-param>
  <param-name>facelets.DEVELOPMENT</param-name>
  <param-value>true</param-value>
</context-param>
```

and add the following to the beginning of your facelet template file.

```
<ui:debug hotkey="k" />
```

If you Ctrl+Shift+k is pressed, debug window will be opened and displays the component tree and scoped variables.

If you are using Seam, set the debug mode to true.

```
<context-param>
  <param-name>org.jboss.seam.core.init.debug</par
   <param-value>true</param-value>
</context-param>

```

#4. Use browsers' debugging capabilities like FireBug plugin for FireFox or Google chrome's out of the box inspect element capability to view the generated HTML source code to get some hints. FireBug for FireFox and Fiddler2 for Internet Explorer can debug AJAX calls.

Popular Posts

♦ 11 Spring boot interview questions & answers

825 views

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

767 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

240 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**

‹ 7 More Maven interview Questions & Answers

More JSF interview Q&A ›

Posted in FAQ JEE Job Interview Q&A Essentials, JSF, member-paid

Tags: Java/JEE FAQs

Leave a Reply

Logged in as geethika. Log out?

Comment

Post Comment

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

 ↑