05: Web patterns interview Q&A

Posted on August 24, 2014 by Arulkumaran Kumaraswamipillai

Q1. What do you know about model 0, model 1 and model 2 MVC design patterns?

A1. In the **model 0** pattern, which is also known as the **model-less pattern**, business logic is embedded in the JSP pages. The model 0 pattern is fine for a very basic JSP page, but real web applications would have business logic, data access logic etc, which would make the JSP code hard to read, difficult to maintain, difficult to refactor, and untestable. It is also not recommended to embed business logic and data access logic in a JSP page since it is protocol dependent (i.e. HTTP protocol) and makes it unable to be reused elsewhere like a wireless application using a WAP protocol, a standalone XML based messaging application, etc.

You can refactor the processing code containing business logic and data access logic into Java classes, which adhered to certain standards. This approach provides better testability, reuse and reduced the size of the JSP pages. This is known as the "**model 1**" pattern where JSPs retain the responsibility

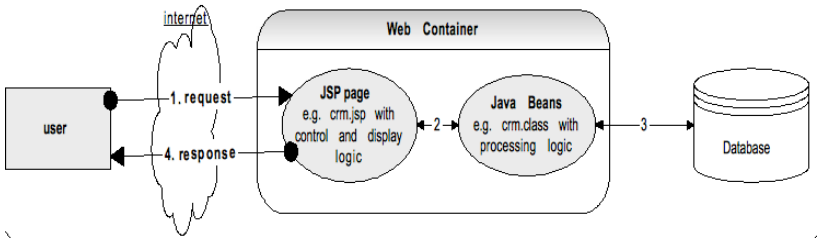
600+ Full Stack Java/JEE Interview Q&As ♥Free ♦FAQs

open all | close all

- Ice Breaker Interview
- Core Java Interview C
- JEE Interview Q&A (3
  - JEE Overview (2)
  - Web basics (8)
    - 01: ♦ 12 Web ba
    - 02: HTTP basics
    - 03: Servlet interv
    - 04: JSP overviev
    - 05: Web patterns
    - 06: ♦ MVC0, MV
    - 07: When to use
    - 08: Web.xml inte
  - WebService (11)
  - JPA (2)
  - JTA (1)
  - JDBC (4)
  - JMS (5)
  - JMX (3)
  - JNDI and LDAP (1)
- Pressed for time? Jav

of a controller, and view renderer with display logic but delegates the business processing to java classes known as Java Beans. The Java Beans are Java classes, which adhere to following items:

1) Implement java.io.Serializable or java.io.Externalizable interface.
2) Provide a no-arguments constructor.
3) Private properties must have corresponding getXXX/setXXX methods.



Model-1 MVC

The above model provides a great improvement from the model 0 or model-less pattern, but there are still some problems and limitations.

Problem with model-1: In the model 1 architecture the JSP page is alone responsible for processing the incoming request and replying back to the user. This architecture may be suitable for simple applications, but complex applications will end up with significant amount of Java code embedded within your JSP page, especially when there is a significant amount of data processing to be performed. This is a problem not only for Java developers due to design ugliness but also a problem for web designers when you have large amount of Java code in your JSP pages. In many cases, the page receiving the request is not the page, which renders the response as an HTML output because decisions need to be made based on the submitted data to determine the most appropriate page to be displayed. This would require your pages to be redirected (i.e. sendRedirect (…)) or forwarded to each other resulting in a messy flow of control and design ugliness for the application. So, why should you use a JSP

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 Interviev

16 Technical Key Areas

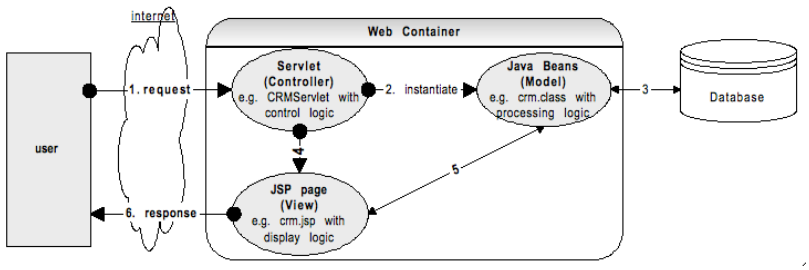
open all | close all
Best Practice (6)
Coding (26)
Concurrency (6)
Design Concepts (7)
Design Patterns (11)
Exception Handling (3
Java Debugging (21)
Judging Experience In
Low Latency (7)
Memory Managemen
Performance (13)
QoS (8)
Scalability (4)
SDLC (6)
Security (13)
Transaction Managen

80+ step by step Java Tutorials

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

page as a controller, which is mainly designed to be used as a template for the view?

Solution: You can use the **Model 2 architecture** (MVC – Model, View, Controller architecture), which is a hybrid approach for serving dynamic content, since it combines the use of both Servlets and JSPs. It takes advantage of the predominant strengths of both technologies where a Servlet is the target for submitting a request and performing flow-control tasks and using JSPs to generate the presentation layer. As shown in the diagram below, the **Servlet acts as the controller** and is responsible for request processing and the creation of any beans or objects used by the JSP as well as deciding, which JSP page to forward or redirect the request to (i.e. flow control) depending on the data submitted by the user. The **JSP page is responsible for** retrieving any objects or beans that may have been previously created by the Servlet, and as a **template for rendering the view** as a response to be sent to the user as an HTML.



MVC 2 design pattern

Q2. How do you prevent multiple submits due to repeated "refresh button" clicks?

A2. **Problem:** Very often a user is completely unaware that a browser re-sends information to the server when a "refresh button" in Microsoft Internet Explorer or a "reload button" in Netscape/Mozilla is clicked. Even if a browser warns user, a user cannot often understand the technical meaning of the warning. This action can cause form data to be resubmitted, possibly with unexpected results such as duplicate/multiple purchases of a same item, attempting to delete the previously deleted item from the database resulting in a *SQLException* being thrown. **Non-idempotent** methods are methods that

Hadoop & Spark Tuto
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 I
Designing your classe
Java Data Structures
Passing the unit tests
What is wrong with th
Writing Code Home A
Written Test Core Jav
Written Test JEE (1)

How good are your .....?

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

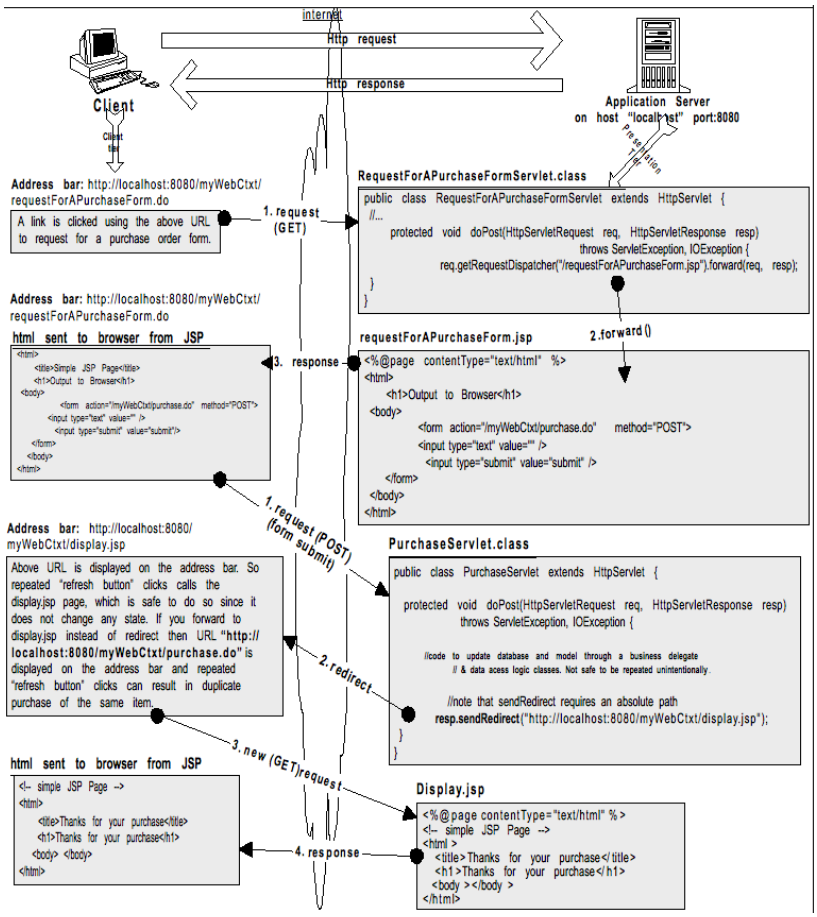
cause the state to change. But some operations like reading a list of products or customer details, etc are safe because they do not alter the state of the model and the database. These methods are known as **idempotent methods**.

Solution-1: You can use a **Post/Redirect/Get (aka PRG) pattern**. This pattern involves the following steps:

Step-1: First a user filled form is submitted to the server (i.e. a Servlet) using a “POST”. Servlet performs a business operation by updating the state in the database and the business model.

Step-2: Servlet replies with redirect response (i.e. sendRedirect() operation as opposed to the forward() operation) for a view page.

Step-3: Browser loads a view using a “GET” where no user data is sent. This is usually a separate JSP page, which is safe from “multiple submits”. For e.g. reading data back from a database to display, a confirmation page, etc.



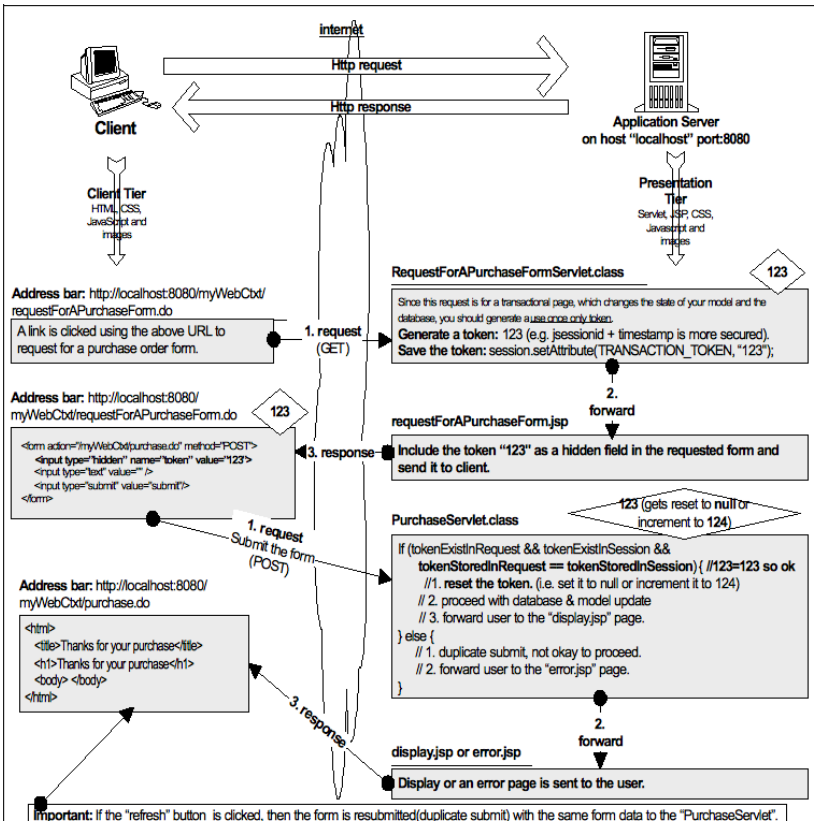
PRG (Post-Redirect-Get) pattern to prevent duplicate page submits

Advantages: Separates the view from model updates and URLs can be bookmarked.
Disadvantage: Extra network round trip.

Solution-2: The solution-1 has to make an extra network round trip. The **synchronizer token pattern** can be applied in conjunction with request forward (i.e. instead of redirect) to prevent multiple form submits with unexpected side effects without the extra round trip.

The basic idea of this pattern is to set a use **once only token** in a "session", when a form is requested and the token is stored in the form as a hidden field. When you submit the form the token in the request (i.e. due to hidden field) is compared with the token in the session. If tokens match, then reset the token in the session to null or increment it to a different value and proceed with the model & database update. If you inadvertently resubmit the form by clicking the

refresh button, the request processing Servlet (i.e. PurchaseServlet) first tests for the presence of a valid token in the request parameter by comparing it with the one stored in the session. Since the token was reset in the first submit, the token in the request (i.e 123) would not match with the token in the session (i.e. null or 124). Since the tokens do not match, an alternate course of action is taken like forwarding to an error.jsp page with an error message like “duplicate page submission detected”.

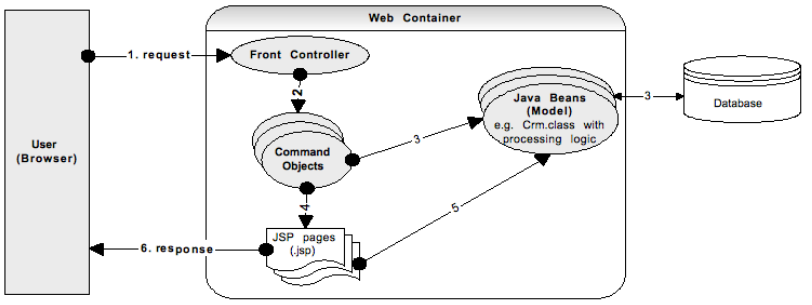


Synchronizer token pattern to prevent duplicate form submits

Q3. What is a Front Controller pattern with command objects?

A3. The **model-2 MVC pattern** can be further improved and simplified by using the **Front Controller pattern with command objects**. In a complex Web site there are many similar input control operations like security, internationalization, controlling and logging user’s progress through the site, etc you need to perform while handling a request. If these input control operations are scattered across multiple objects, much of these behaviors can end up

duplicated resulting in maintenance issues. The Front Controller pattern uses a single Servlet, which acts as initial point of contact for handling all the requests, including invoking services such as security (authentication and authorization), logging, gathering user input data from the request, gathering data required by the view, etc by delegating to the helper classes, and managing the choice of an appropriate view with the dispatcher classes. These helper and dispatcher classes are generally instances of a command design pattern and therefore usually termed as command objects.

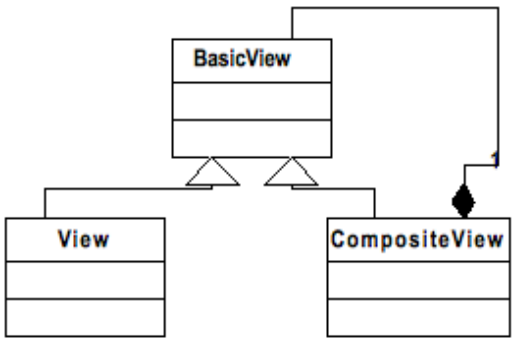


MVC-2 with Front controller and command objects

The popular MVC 2 frameworks like Struts, Spring MVC, etc use the **Front Controller pattern**, where a centralized single Servlet is used for channeling all requests and creating instances of command classes for processing user requests.

Q4. Briefly discuss the following patterns — Composite view, View helper, Dispatcher view and Service to worker?

A4. **Composite View:** Creates an aggregate view from atomic sub-views. The Composite view entirely focuses on the view. The view is typically a JSP page, which has the HTML and JSP Tags. The JSP display pages mostly have a side bar, header, footer and main content area. These are the sub-views of the view. The sub-views can be either static or dynamic. The best practice is to have these sub-views as separate JSP pages and include them in the whole view. This will enable reuse of JSP sub-views and improves maintainability by having to change them at one place only.



Composite Views

In JSF, you can create reusable**composite components**.

View Helper: When processing logic is embedded inside the controller or view it causes code duplication in all the pages. This causes maintenance problems, as any change to piece of logic has to be done in all the views. In the view helper pattern the view delegates its processing responsibilities to its helper classes. The helper classes JavaBeans, Tags, etc.

Service to Worker and **Dispatcher View**: These two patterns are a combination of Front Controller and View Helper patterns with a dispatcher component. One of the responsibilities of a Front Controller is choosing a view and dispatching the request to an appropriate view. This behavior can be partitioned into a separate component known as a dispatcher. But these two patterns differ in the way they suggest different division of responsibility among the components.

Service to Worker combines the front controller and dispatcher, with views and view helpersto handle client requests and dynamically prepares the response. Controllers delegate the content retrieval to the view helpers, which populates the intermediate model content for the view. Dispatcher is responsible for the view management and view navigation. This pattern promotes more up-front work by the **front controller and dispatcher** for the authentication, authorization, content retrieval, validation, view management and navigation.

Dispatcher View pattern is structurally similar to the service to worker, but the emphasis is on a different usage pattern. This combines the Front controller and the dispatcher with the view helpers but the **controller does not delegate content retrieval to view helpers because this activity is deferred to view processing**. Dispatcher is responsible for the view management and view navigation. This pattern promotes lightweight front controller and dispatcher with minimum functionality and most of the work is done by the view.

Q5. Can you describe optimistic concurrency control (i.e. OCC) with respect to web applications
A5.

Step 1: User A opens a Product catalog page, and makes some changes.
Step 2: User B opens the same Product catalog page, and makes some changes.
Step 3: Both users submit their changes the same time without being unaware of each others changes.
Step 4: Only one of the users change should go through (i.e. commit), and the other user should get an error message indicating “Concurrent modification detected to Product catalog with id 123, please refresh the page to view the update content, and retry.”

This can be accomplished on the back-end by having a **verison flag** or a **timestamp** column to the database table (e.g. Product) to detect concurrent updates. ORM tools like Hibernate supports “optimistic locking” out of the box with the help of a version or timestamp column to the underlying database tables.

In theory, the timestamp is a little less safe than a **version number**. This is because a timestamp can, in theory, be updated by two transactions at the exact same time (depends on the precision of the database) and allow both to update violating the optimistic lock. It can also be argued that timestamps also take up more space in the database.

Q6. How do you get your servlet to stop timing out on a really long database query?

A6. There are situations despite how much database tuning effort you put into a project, there might be complex queries or a batch process initiated via a Servlet, which might take several minutes to execute. The issue is that if you call a long query from a Servlet or JSP, the browser may time out before the call completes. When this happens, the user will not see the results of their request.

The solution is to use an asynchronous Servlet.

```
1  @WebServlet(asyncSupported = true, value = "/Unb
2  public class UnblockingServlet extends HttpServl
3
4    protected void doGet(HttpServletRequest reques
5      //doGet returns without blocking after deleg
6      //The context of the request is stored using
7      AsyncWork.add(request.startAsync());
8    }
9  }
10
```

```
1  public class AsyncWork implements ServletContext
2
3  private static final BlockingQueue queue = new L
4
5    private volatile Thread thread;
6
7    public static void add(AsyncContext c) {
8      queue.add(c);
9    }
10
11   @Override
12   public void contextInitialized(ServletContextE
13     thread = new Thread(new Runnable() {//run in
14       @Override
15       public void run() {
16         while (true) {
17           try {
18             Thread.sleep(5000);//emulate long ru
19             AsyncContext context;
20             while ((context = queue.poll()) != n
21               try {
22                 //all the blocked tasks in the q
23                 //by constructing and sending th
24                 ServletResponse response = conte
25                 response.setContentType("text/pl
26                 PrintWriter out = response.getWr
27                 out.printf("Thread %s completed
28                 out.flush();
29               } catch (Exception e) {
30                 throw new RuntimeException(e.getI
31               } finally {
32                 context.complete();
```

```
33                }
34              }
35            } catch (InterruptedException e) {
36              return;
37            }
38          }
39        }
40      });
41      thread.start();
42    }
43
44    @Override
45    public void contextDestroyed(ServletContextEve
46      thread.interrupt();
47    }
48  }
49
```

Instead of blocking hundreds of web container threads to wait for the long running queries to finish, the blocked tasks are queued by batching the similar groups together and processing the requests in a single thread.

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**

‹ 03: Servlet interview Q&A

04: JSP overview interview questions and answers ›

Posted in member-paid, Web basics

Tags: Architect 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.