4 JEE Design Patterns Interview Q&As

Posted on September 11, 2014 by Arulkumaran Kumaraswamipillai — No Comments ↓

Unlike J2EE, Java EE 6 consists of annotated classes without any dependencies on the platform. This approach eliminates the need to separate business logic from the infrastructure and makes the majority of J2EE patterns and best practices superfluous. The following J2EE design patterns can be obsolete:

— **Service Locator** (Use **Dependency Injection** (DI) instead): To address the complexity of JNDI look-up in the EJB container, the EJB3 group decided to use annotations for dependency injection

— **Business Delegate** (Use Java's interfaces instead of EJB's remote interfaces):

— **Data Transfer Objects** (DTOs) (as you can use POJOs for entities). DTO is not an anti pattern. DTO is meant to be used only when you need to pass data from one subsystem to another

Java EE 6 applications are written according to the **YAGNI** (You Ain't Gonna Need It), **DRY** (Don't Repeat Yourself), and **KISS** (Keep It Simple, Stupid) principles.

Q1. Can you briefly explain some of the JEE design patterns?
A1.

Front Controller Design Pattern

Problem: A JEE system requires a centralized access point for HTTP request handling to support the integration of system services like security, data validation etc, content retrieval, view management, and dispatching. When the user accesses the view directly without going through a centralized mechanism, two problems may occur:

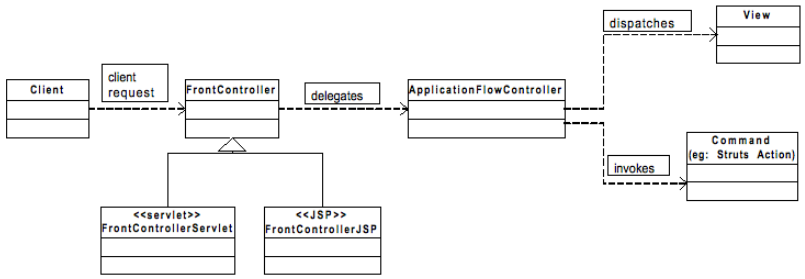
— Each view is required to provide its own system services often resulting in duplicate code.

— View navigation is left to the views. This may result in shared code for view content and view navigation.

— Distributed control is more difficult to maintain, since changes will often need to be made in numerous places.

Solution: Generally you write specific Servlets for specific request handling. These Servlets are responsible for data validation, error handling, invoking business services and finally forwarding the request to a specific JSP view to display the results to the user. The **Front Controller** suggests that

we only have one Servlet (instead of having specific Servlet for each specific request) centralizing the handling of all the requests and delegating the functions like validation, invoking business services etc to a command or a helper component. For example Struts framework uses the command design pattern to delegate the business services to an action class.



Front Controller Design Pattern

Benefits

— Avoid duplicating the control logic like security check, flow control etc.
— Apply the common logic, which is shared by multiple requests in the Front controller.
— Separate the system processing logic from the view processing logic.
— Provides a controlled and centralized access point for your system.

MVC Pattern

stands for **Model-View-Controller Pattern**. This pattern is used to separate application's concerns. It divides the functionality of displaying and maintaining of the data to minimize the degree of coupling (i.e. promotes loose coupling) between components. It is often used by applications that need the ability to maintain multiple views like html, wml, JFC/Swing, XML based Web service etc of the same data.

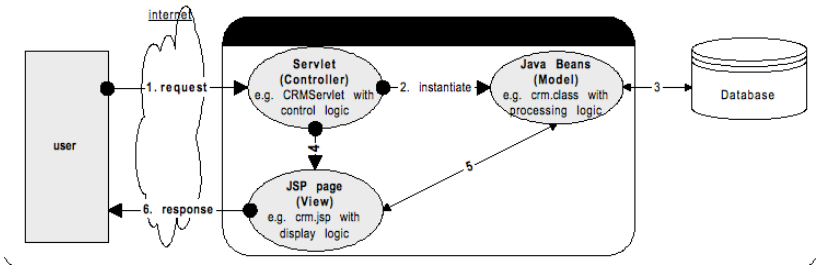
Git & SVN (6)
JMeter (2)
JSF (2)
Maven (3)
Testing & Profiling/Sa
Other Interview Q&A t
Free Java Interviev

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 I
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
Hadoop & Spark Tuto

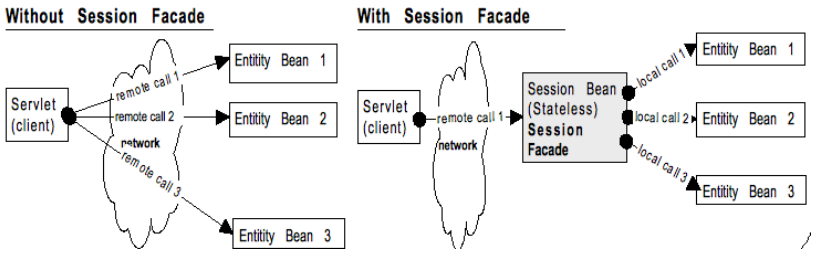


MVC Pattern

Session Facade Pattern

Problem: Too many method invocations between the client and the server will lead to network overhead, tight coupling due to dependencies between the client and the server, misuse of server business methods due to fine grained access etc.

Solution: Use a session bean as a façade to encapsulate the complexities between the client and the server interactions. The Session Facade manages the business objects, and provides a uniform coarse-grained service access layer to clients.



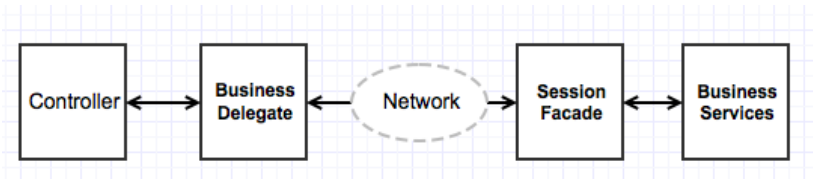
Sesssion Facade Pattern

Business Delegate Pattern

is used to decouple presentation tier and business tier. It is basically use to reduce communication or remote lookup functionality to business tier code in presentation tier code.

100+ Java pre-interview coding tests

How good are your .....?



Business Delegate Pattern

Problem: When presentation tier components interact directly with the business services components like EJB, the presentation components are vulnerable to changes in the implementation of business services components.

Solution: Use a **Business Delegate** to reduce the coupling between the presentation tier components and the business services tier components. Business Delegate hides the underlying implementation details of the business service, such as look-up and access details of the EJB architecture.

Business delegate is responsible for:

— Invoking session beans in Session Facade.
— Handling exceptions from the server side. (Unchecked exceptions get wrapped into the remote exception, checked exceptions can be thrown as an application exception or wrapped in the remote exception. unchecked exceptions do not have to be caught but can be caught and should not be used in the method signature.)
— Re-trying services for the client (For example when using optimistic locking business delegate will retry the method call when there is a concurrent access.).

Data Access Object Pattern or DAO pattern

is used to separate low level data accessing API or operations from high level business services.

Intercepting filter design pattern

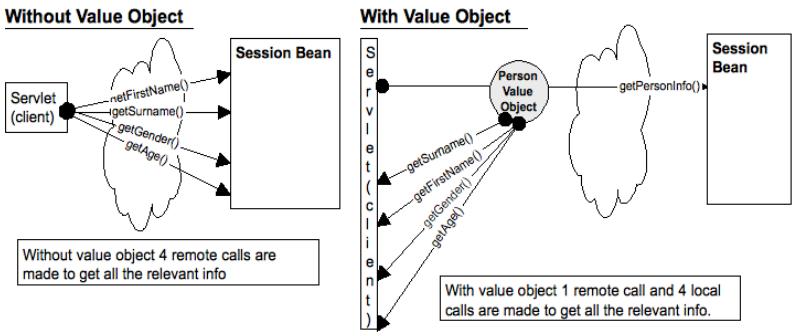
is used when we want to do some pre-processing/post-processing with request or response of the application. Filters are defined and applied on the request before passing the request to actual target application. Filters can do the authentication/ authorization/ logging or tracking of request and then pass the requests to corresponding handlers. For example, Servlet filters, Spring AOP based interceptors, etc.

Service Locator Design Pattern

is used when you want to locate various services using JNDI lookup. Considering high cost of looking up JNDI for a service, Service Locator pattern makes use of caching technique. For example, Data sources, JMS queues, etc can be looked up using this pattern.

Data Transfer Object (DTO) pattern

is used when you want to pass data with multiple attributes in one shot from client to server, hence reducing network round trips required. A DTO is also known as Value Object (i.e VO). A DTO is a simple POJO class having getter/setter methods and is serializable so that it can be transferred over the network. It does not have any behavior. Domain Objects representing a persistence store can be copied on to DTOs to be represented in the view layer.



DTO Pattern

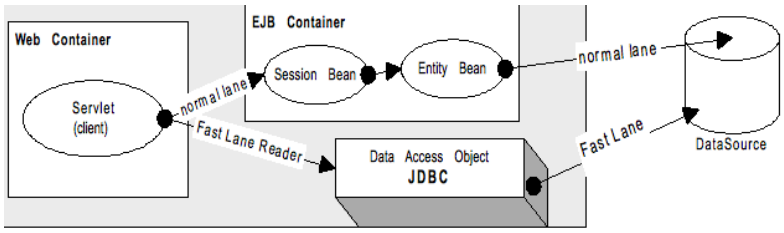
Composite Entity pattern

is used in EJB persistence mechanism. A Composite entity is an EJB entity bean which represents a graph of objects. When a composite entity is updated, internally dependent objects beans get updated automatically as being managed by EJB entity bean.

Fast lane reader pattern

Problem: Using Entity beans/Hibernate to represent persistent, read only tabular data incurs performance cost at no benefit (especially when large amount of data to be read).

Solution: Access the persistent data directly from the database using the DAO (Data Access Object) pattern instead of using Entity beans. The Fast lane readers commonly use JDBC, Connectors, etc to access the read-only data from the data source. The main benefit of this pattern is the faster data retrieval.



Fast lane reader pattern

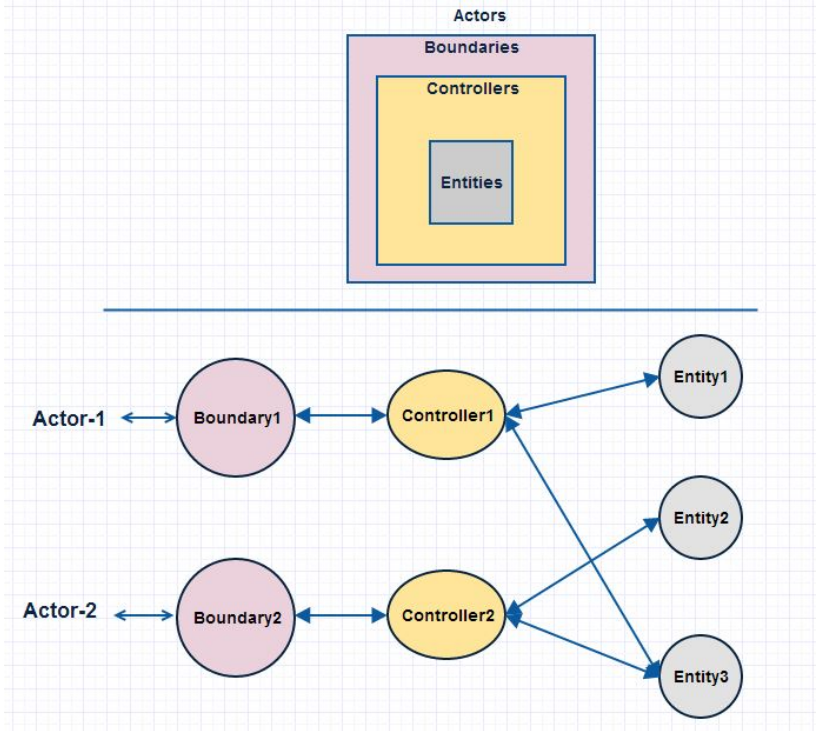
Q2. Do we really need a DAO pattern when there is JPA?
A2. JPA stands for Java Persistence API, and is part of the Java EE 5 specification and has been implemented by Hibernate, TopLink, EclipseLink, OpenJPA, and a number of other object-relational mapping (ORM) frameworks. When JPA provides the abstraction, why do we need a DAO class?

DAO pattern has the following advantages:

— No direct dependency on the JPA api from client code.
— Type-safety through the use of generics. Any casts that still need to be done are handled in the DAO implementation.
— One logical place to group all entity-specific JPA code.
— One location to add transaction markers, debugging, profiling, etc.
— One class to test when testing the database access code.

Having said this, some could argue that this is over engineering. You could eliminate the DAO layer, and instead hit the JPA entities directly within your business tier, but it is good to maintain a separate persistence (DAO) and business tier, particularly in those cases where you want to mix up some JPA with plain JDBC (i.e. Fast lane reader).

Q3. What is an Entity Control Boundary (**ECB**) pattern?
Q3. This pattern is similar to the **Model View Controller** (i.e MVC) pattern, the **Entity Control Boundary** (ECB) pattern is not solely appropriate for dealing with user interfaces, and it gives the controller a slightly different role to play. The main purpose of the boundary in the ECB pattern is to provide a clear separation between business and presentation logic.



ECB Pattern

— An **entity** class manages persistent data, for example a database. Entities are objects representing system data: Customer, Transaction, Cart, etc.

— A **boundary** class provides an interface to a user or an external system. For example, it might display a dialog requesting information from a user. Boundaries are objects that interface with system actors: user interfaces, gateways, proxies, etc.

— A **control** class encapsulates the main logic for a use case or part of a use case. Controllers are objects that mediate between boundaries and entities. They orchestrate the execution of commands coming from the boundary.

A boundary object could instantiate a control object, say to process information that a user has just entered, a control object could instantiate a boundary object, say to request information, but to ease reuse an entity object should not instantiate boundary or control objects.

Q4. What is CDI, and how does it support the ECB pattern?
Q4. CDI (Contexts and Depency Injection) is the Java standard for dependency injection (DI) and interception (AOP). CDI is a foundational aspect of Java EE 6.

Example:

— A stateless Session **EJB** annotated with JAX-RS annotations to implement a RESTful web service as a **Boundary**. Boundry will have the transaction demarcation with @TransactionAttribute(REQUIRED).

— **CDI managed beans** (or EJB) for business logic as a **Controller**. The scope will be @Dependent to ensure that each thread will have its own instance of bean.

— **JPA entity beans** managed via DAOs as **Entities**. DAOs with @Dependent scope to have separate instance for each thread.

The boundary layer is the facade, the control layer is responsible for the implementation of process and entity-independent logic, and the entity layer contains rich domain objects. In Java EE 6, the boundary is realized with EJBs. The control layer can contain either CDIs or EJBs, and the entity layer can contain either JPA entities or transient, unmanaged entities. For the controller, you can start with a CDI and convert it into an EJB down the track by using the @Stateless annotation.

Popular Posts

♦ 11 Spring boot interview questions & answers

827 views

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

768 views

18 Java scenarios based interview Questions and Answers

400 views

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

389 views

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

296 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

286 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

202 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: ♦10+ Know your industry Q&As for the Java developers

♥ Choosing from multiple Java job offers – analytical approach ›

Posted in Java Key Area Essentials, JEE Patterns, member-paid

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

↑