Advanced Java Programming

Design issues

Chris Wong

cw@it.uts.edu.au

Based on Sun J2EE design patterns, & Marinescu "EJB design patterns" (Wiley 2002)

Topics

- Design patterns
- Sun Design patterns
- Popular design patterns

Design Patterns

- Marinescu: “think of a pattern as a best practice solution to a common recurring problem.”
- Based on years of experience in developing applications
- The pattern bible:
  Gamma, E., R. Helm, R. Johnson, and J. Vlissides. 1995. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.
- See also:

Design Patterns

- We have already seen 2 design patterns in this course
  - Model-View-Controller
  - Session Facade
Sun Design

- Sun have categorised several design patterns.

http://java.sun.com/blueprints/patterns/catalog.html

Sun Design patterns-2

- Business Delegate
  - Reduce coupling between Web and EJB tiers
- Composite Entity
  - Model a network of related business entities
- Data Access Object (DAO)
  - Abstract and encapsulate data access mechanisms
- Fast Lane Reader
  - Improve read performance of tabular data
- Front Controller
  - Centralize application request processing
- Intercepting Filter
  - Pre- and post-process application requests

Sun Design patterns-3

- Model-View-Controller
  - Decouple data representation, application behaviour, and presentation
- Service Locator
  - Simplify client access to enterprise business services
- Session Facade
  - Coordinate operations between multiple business objects in a workflow
- Transfer Object
  - Transfer business data between tiers
- Value List Handler
  - Efficiently iterate a virtual list
- View Helper
  - Simplify access to model state and data access logic

Popular Design patterns

- Most popular design patterns are combination of:
  - MVC + Data Transfer Object + Session Façade
  - Useful toolset to run is Apache Struts
    http://jakarta.apache.org/struts/
- Another pattern that is common with asynchronous update operations is the:
  - Message façade pattern
Model View Controller


Session Facade

Data Transfer Object

- Also called the **Value Object**
- Design transfers related information/objects as one object
- This improves performance and provides logical grouping of objects
- Provides a façade that encapsulates other objects.
- Usually implemented as a JavaBean ie: serializable

Variations:

- **Data Transfer hash map** – instead of getter/setter methods, use attributes keywords get/put data. Passes hash map to backend object (eg: Session EJB)

- **Data Transfer rowset** – like above, but pass a java.sql.Rowset object. Use only with BMP or JDBC backends.
Message Façade-1

- Problem:
  sometimes you need to access multiple objects/EJB’s without blocking eg: airline reservation system and need high availability.
- Solution:
  Use asynchronous programming
- Message façade pattern
  - Uses Message Driven Beans to access Message Oriented Middleware like JMS
  - Servlet sends JMS message to destination.
  - MDB then calls appropriate session/entity EJB

Message Façade-2

- Issue:
  How to communicate back to client success/failure?
- You need to devise alternative methods eg: confirmation/receipt number displayed to client
- Alternative is to ensure all business logic on session façade bean, and message driven bean invokes session façade bean.
- Then if client uses the session façade, they will see updates automatically!

Other Design patterns

EJB Home Factory pattern
- Often need to have multiple JNDI lookups of EJB homes.
- This pattern suggests creating a EJBHomeFactory class which caches bean homes.
- Good separation of business logic from management logic
- Performance can be 10x better.
- Can also be used in servlet
Other Design patterns

Service Locator Pattern
- Often presentation tier needs to call business tier
- However, this means client code needs to know business tier details eg: EJB names. Also complex, can create network overhead, inefficient design
- **ServiceLocator** object can abstract JNDI, cache resources eg: InitialContext, EJB Homes
- Centralises all references to implementation, so you only need to change at one place.

Business Delegate Pattern
- Often presentation tier needs to call business tier
  - However, this means client code needs to know business tier details eg: EJB names.
- This pattern abstracts business tier via client side BusinessDelegate helper object.
- Business delegate does all JNDI, EJB calls. Can cache (see EJBHomePattern). Also can wrap EJB/remoteExceptions.
  - Good seperation between client development and server development
  - Typically uses the Service Locator pattern for system calls.
  - Eg: Shopping Cart Bean

Design patterns conclusion
- We have only lightly touched on common design patterns.
- Should apply these common patterns at first. Try use Struts if you can.
- Don't blindly follow patterns – use what is appropriate for you, even if this means not using EJB’s.
- (and some patterns provide alternatives to EJB’s)

Recommended reading: