J2EE Best Practices and Design Considerations

J2EE Patterns

information@middleware-company.com • +1 (877) 866-JAVA

Copyright © 2002
Transactions guarantee determinism

Transactions give you four virtues or ACID properties:
- Atomicity
- Consistency
- Isolation
- Durability
Balance is everything

Narrower scopes
- Increase transaction traffic
- Decrease transaction latency

Broader scopes
- Decrease transaction traffic
- Increase transaction latency
Likely Discoveries

(Provided transactional scope does not extend to the client layer)

- Transactional traffic is much more costly than latency
  - Transactional storms cause:
    - Increased chance of rollbacks
    - Increased chance of transaction interaction: Unexpected state

- Longer-lived transactions cause
  - Isolation conflicts: Blocking and waiting
  - More manageable and consistent operations

- Examples of longer-lived transactions include:
  - Performing multiple updates within a single database transaction
  - Using a session façade to demarcate transactions for entity beans
Take-away Points

Abstract and cache your data to:
- Decrease client complexity
- Decrease network latency and traffic

Be aware of how "up to date" you really need your data
- If an operation does not require pure data, give them latest data
- If an operation does not require latest data, give them stale data
- etc...

Increase transactional scope to:
- Ensure data consistency
- Reduce traffic
Roadmap

Where are we?

- **Transactions**
- **J2EE Design Concerns**
  - *EJB and Data Concerns*
  - General Design Choices and Strategies
- What We Can Control
A Graphical Look

Servlet, Applet, or Command-Line Client

EJB Container

Session Bean

Entity Bean

RMI

JDBC

JDBC

Database
Decision Points

▼ Use session beans with Java objects when
- Large volumes of data are retrieved in single SQL calls
- You are working with very small bits of data
- You have exclusive data (due to caching)
- You have programmers who are not familiar with OO data objects
- You have existing, (working) SQL code

▼ Use Entity Beans when
- You have shared data (due to caching)
- CMP is not prevented due to DB Optimizations
- You can take advantage of ejb20 relationship-management
- Data objects can be generated from UML diagrams through tools
When to Use Container-Managed Persistence

▼ Advantages of container-managed persistence
- Less code
- Faster to develop

▼ Advantages of bean-managed persistence
- More control (complex mappings)
- Can take advantage of existing tuned SQL (triggers)
- Less new learning (reuse and extend old JDBC code/components)

▼ CMP is a more realistic choice now because of:
- New EJB 2.0 relationship management
- New EJB 2.0 local interfaces
When to Use Stateful v. Stateless?

- Say we have a stateful business process (e.g. shopping cart)
- We can keep our state in one of three places
  - In the client's memory
  - In the server's memory
  - On the database disk

Thick Client (Applet/Application)
Thin Client (Browser)
Server (JSP, Servlets, EJB)
Database

Copyright © 2002 by The Middleware Company
Pros and Cons

Database Pros:
- Database is a transactional/recoverable device

Database Cons:
- Disk I/Os and network roundtrips are required
- Need to write code to marshal state in and out of component

Client Pros:
- If client crashes we expect to lose state anyway.
- No disk I/Os

Client Cons:
- Requires network overhead marshalling state back and forth
- The client API becomes more complex due to additional state parameters
- Need to write code to marshal state in and out of component

Server Pros:
- No code to marshal state in and out of component
- No network roundtrips nor disk I/Os to marshal state
- No single point of failure if clustering/replication is used (judiciously)
XML is useful when communicating between disparate systems

XML transformation and parsing is slow

**Do not** use XML for data transfer **within** a J2EE system

- The existing APIs work just fine
- Remember OO?
- J2EE applications are all Java

**Do** transfer data between heterogeneous systems as XML when the benefits it offers can be realized
Roadmap

Where are we?

- Transactions
- **J2EE Design Concerns**
  - EJB and Data Concerns
  - *General Design Choices and Strategies*
- What We Can Control
Dividing up your team

3 basic approaches

- Horizontal partitioning
  - Developers specialize in APIs
  - Examples: JSP team, an entity bean team, etc

- Vertical partitioning
  - Developers focus on use cases and are generalists (use all APIs)

- Hybrid partitioning
  - One subject matter expert per use case
  - Developers specialize in APIs and move from use case to use case

If goal is to get project success quickly and consistently, go for horizontal or hybrid

If goal is to educate developers, go for vertical
Reality and Reuse

Rocky history of reusing components within an organization
- Projects are too different, too much politics for reuse

Rather than reuse, we recommend striving for
- Common vocabulary between projects
  - Empowers communication
- Common design patterns
  - Enables developers to easily transition between projects
- Common frameworks
- Pseudo-reuse via copy-pasting of code between projects

Can achieve this via a "best practices task force"
Choosing an Application Server

If you have no time, you could just go with market leader

- **Warning:** vendor may not handle *your* needs in optimal way

Recommended process for selection of server

- List all features you want
- Weight and prioritize the feature list
- Eliminate vendors that don’t meet majority of criteria
- Download application servers from 2-3 remaining vendors
- In a lab environment, measure features most important to you
  - Scalability
  - Usability
  - Performance
Deploy vertical slice into those application servers

- You know how well the server handles your needs
- Entire process takes 2-4 weeks
- Can occur in parallel with development if you code portably
- Consultants can help you through this process
A Vertical Slice

What is a vertical slice?

- One or two use-cases in your system
- Example: search engine, or product catalog – but not complete site
- You build this slice, using all the relevant J2EE APIs:
  - JSPs, servlets, EJBs, etc

Benefits of implementing a vertical slice

- Gain experience with J2EE
- Reduce risk of 'unsound architecture'
- Prove project works to stakeholders – show them working use-case
- Answers question: "Will it scale?" – can performance tune slice early
- Helps you refine design patterns
- Not throwaway code: Slice is real piece of working system
When to Use 3-Tier v. 4-Tier J2EE Server

3 tier
- Thick Client (Applet/ Application)
- Thin Client (Browser)
- J2EE Server (JSP, Servlet, EJB)
- Database

4 tier
- Thick Client (Application)
- Thin Client (Applet or Browser)
- Web Server (JSP, Servlet)
- EJB Server (EJB)
- Database
When to Use 3-Tier v. 4-Tier

...continued

- **Load-balancing**
  - Required and only possible in 3 tier

- **Efficiency**
  - No RPCs in 3-tier –possibly no IPCs (inter-process calls)

- **Adaptability**
  - 3-tier is more self-adapting than 4-tier
  - Resources are used for whatever purpose is needed currently
    - Either web server or application server

- **Security**
  - Firewall may be deployed in-between boxes in 4-tier

- **Fail-over**
  - 4-tier has no fail-over capability in this limited scenario
Increase Responsiveness

- When Programming EJBs
  - Use Local Interfaces
  - Use Session or Message-driven façade

- Cache Data
  - Read-only Entity Beans
  - Data Transfer Objects

- Cache JNDI Resources
  - EJB Home Objects
  - Data Sources
  - JMS Connection Factories
Increase Responsiveness
...continued

▼ Serialization
- Use the transient keyword for fields that you don’t want serialized
- Use smaller transport objects to send the subset of info you need

▼ Reduce network traffic
- Make coarse-grained network calls (“chunk” your remote requests)
- Always be aware of your remote boundaries

▼ Key to designing correctly: perceived performance
- Be smart about how you optimize
- E.g. don’t optimize a Swing GUI if the app sits idle most of the time
- Don’t put long-running tasks on the UI thread
- Put them on a new thread to give your GUIs snappier performance
Increase Responsiveness...continued

- Garbage collection
  - Null out references that are no longer needed
  - They are more likely to be GC’ed

- Use lazy initialization, this is a powerful technique

```java
MySingleton singleton = null;
...
if (singleton == null)
    singleton = new MySingleton();
return singleton;
```
Increase Productivity

- Use/build frameworks

- What is a framework?
  - Set of related classes which you specialize and/or instantiate to implement an application or subsystem
  - Why are they useful?
    - The framework is not just a collection of classes – it also defines a generic design
    - Bi-directional flow of control means the framework can contain much more functionality than a traditional library
  - Obstacle: learning curve
Roadmap

Where are we?

- The Good of the Many: Data Tiers and Transaction Concepts
- J2EE Design Concerns
- **What We Can Control**
  - Handling Java Threads
  - Memory Management
  - Accurate Benchmarks
Handling Java Threads

- With EJB thread management is handled for us
- Vendors allow us to tune the # of threads in the App Server
  - Too many threads cause increased competition for resources
  - Too few threads cause under-utilization of CPU
- We can choose to deploy to multiple instances
  - Each VM has its own thread pool limit
  - Each tier has its own thread behavior
    - Web-tier uses greater number of threads of a shorter duration
    - EJB-tier uses fewer threads of a greater duration
  - Deploying to multiple CPUs decreases CPU exhaustion
Memory Management

▼ More is Better

▼ Do not starve your OS
  ■ If you have 2GB ram available save 256mb for the OS

▼ Use –Xms<x-Amount>m -Xmx<x-Amount>m arguments to VM
  ■ Ensure the minimum and maximum are identical
    ● You are certain to get what you need at startup
    ● You can easily partition your ram among multiple instances

▼ Remember: Actual RAM use can be twice that declared
Obtaining Accurate Benchmarks

System-centric
- System.currentTimeMillis()
  - Invasive, inaccurate by as much as tens of milliseconds
  - May be all you have
- CPU utilization
- Memory use

Test... Test... Test...
- Every application is different
- Fix what you can – it’s the least you can do
  - Use StringBuffers in place of Strings if doing much concatenation
  - Reuse objects where possible since object creation is costly
- Development-time performance measurement
  - Sitraka PerformaSure
  - Borland OptimizeIt
Obtaining Accurate Benchmarks
...continued

▼ User-centric
- What really counts
- Difficult to determine
  - Network issues
  - User error/unexpected behavior
  - Real-world load

▼ Requires the right tools
- Deployed/Live performance measurement and advice
  - Precise Indepth for J2EE
Summary

In this presentation we discussed:

- The Good of the Many: Transactions
- J2EE Design Concerns
  - When to Use Messaging
  - EJB and Data Concerns
  - General Design Choices and Strategies
- What We Can Control
  - Handling Java Threads
  - Memory Management
  - Accurate Benchmarks