WHITESTEIN
Technologies

Resource Management for J2SE Applications

JUGS

Zurich, May 26th 2005

About the Speaker

Name & Contact

- Martin Kernland
- mek@whitestein.com

Current Job

- Works at Whitestein Technologies AG – Software Agent Technology Experts
  http://www.whitestein.com
- Senior Software Engineer – Project Manager, Software Architect, Technology Consultant

Previous Job

- Worked at Softwired AG – A Java Message Service (JMS) Provider
  http://www.softwired-inc.com
- Software Engineer developing a JMS server called iBus//MessageServer

Objectives & Agenda

Objectives

- ***Understand*** what Resource Management is all about
- Raise the ***awareness*** for Resource Management
- Get some ***new ideas*** for your current project

Agenda

- What Is Resource Management?
- Resource Management Patterns
- What Java Already Provides
- Real-World Challenges
- Usage & Configuration
- Outlook
- Wrap-up

What Is Resource Management

The Problem

- Often Java developers don't think about resource usage
- Result:
  - → Late in development cycle (testing) – limits on resource usage are found (e.g. OutOfMemory Error occurs)
  - → Successful applications are used beyond the defined non-functional requirements

The Solution

- Control these resources (memory, CPU, connections, and components, plug-ins)
- Make sure these scenarios do not happen
- Resource Management != memory leak prevention
  - → Memory leak through resource acquisition, but no release
  - → Expect clean acquisition and release of resources are done

Resource Management Patterns

Three Categories

- Resource Acquisition Patterns
- Resource Lifecycle Patterns
- Resource Release Patterns

Roles

- Resource User
- Resource Provider

Additional Remark

- High Level Design Patterns – for some it might seem too boring, too general
- Good Categorization of Solutions

Resource Acquisition Patterns

Lookup Pattern

- Description: How to find and access resources (local or distributed) using a lookup service as a mediating instance.
- Examples:
  - JNDI / UDDI
  - Eclipse Plug-in registry
- Pros:
  - Location independence
  - Configuration simplicity (including property-based selection)
- Cons:
  - Single point of failure
  - Dangling references

Resource Acquisition Patterns (cont.)

Lazy Acquisition Pattern

- Description: Deferring resource acquisition to the latest possible time during execution
- Examples
  - Java class loading
  - Eclipse Plug-in
  - Singleton pattern (classic)
- Pros
  - Stability
  - Faster startup-time
- Cons
  - Less predictable
  - Time delay on usage of the resource

Resource Acquisition Patterns (cont.)

Eager Acquisition Pattern

- Description: acquiring resources before their use to have provide them when needed.
- Examples
  - Pooling
  - Eclipse Plug-in Declarations
  - Hamster (yes, the animal)
- Pros
  - Predictability
  - No time-delay on usage of the resource
- Cons
  - Less scalable through over-acquisition
  - Slower start-up time

Resource Acquisition Patterns (cont.)

Partial Acquisition Pattern

- Description: breaking up acquisition of a resource into multiple stages, each stage acquires a part of the resource
- Examples:
  - Socket input: blocks of data are read
  - Web browser: incremental image loading
  - Network management application: continuously updating list of resources
- Pros:
  - Scalability
  - Configurability
- Cons
  - Complexity

Resource Lifecycle Patterns

Caching Pattern

- Description: avoid expensive re-acquisition of resource by not releasing them immediately after their use. The resources ***retain their identity*** and are kept in a (fast) storage.
- Examples:
  - Hardware cache / Operating Systems (file system cache)
  - Databases
  - Web browsers
- Pros
  - Faster acquisition of resource
- Cons:
  - Synchronization complexity
  - More memory is used

Resource Lifecycle Patterns (cont.)

Pooling Pattern

- Description: avoid expensive acquisition and release of resources by recycling these resources. Recycled and pooled resources have no identity (and no state)
- Examples:
  - Thread Pool
  - State-Less Session Beans (SLSB) on J2EE application servers
- Pro:
  - Usually faster acquisition of resources
  - Stability/scalability
- Cons:
  - Synchronization
  - Depending on pool size, system might be slowed down

Resource Lifecycle Patterns (cont.)

Resource Lifecycle Manager Pattern

- Description: decouples the management of the resource from the their use by introducing a “Manager” who manages and maintains the resources of an application.
- Examples
  - Component Container (EJB Container, CCM Container)
  - JCA Container of J2EE
- Pros
  - Control (e.g. over interdependent resource)
  - Transparency
  - Performance enhancement
- Cons
  - Single-point of failure

Resource Release Patterns

Leasing Pattern

- Description: simplifies resource release by associating time-based leases with resources when they are acquired. The resource is automatically released when the lease expires.
- Examples:
  - Web sessions (web email, shopping)
  - DHCP (leases IP addresses)
  - Software licenses
- Pros:
  - Simplicity
  - Versioning
- Cons:
  - Additional overhead (checks, application logic, timer, etc.)

Resource Release Patterns (cont.)

Evictor Pattern

- Description: how and when to release resources to optimize resource management by applying different eviction strategies.
- Example:
  - Paging of memory (operating system)
  - Caches
  - JMS message queue (non-persistent topic messages may be deleted)
- Pros:
  - Scalability
  - Stability
- Cons:
  - Overhead (depending on the strategy used)

What Java Already Provides

Thread Pools

- java.util.concurrent.ThreadPoolExecutor
- Idea: have threads exeute a queue of Runnables
- Constructor:

```
public ThreadPoolExecutor(int corePoolSize,
                          int maximumPoolSize,
                          long keepAliveTime,
                          TimeUnit unit,
                          BlockingQueue<Runnable> workQueue,
                          ThreadFactory threadFactory)
```

What Java Already Provides (cont.)

JDBC Connection Pool

- javax.sql.ConnectionPoolDataSource (JDBC 3.0 specification)
- returns javax.sql.PooledConnection (represents ONE physical connection)

  → PooledConnection is reusable

```
javax.naming.Context ctx = new InitialContext();
javax.sql.DataSource ds = (DataSource)ctx.lookup("jdbc/myConnectionpool");
try {
    Connection con = ds.getConnection("username", "password");
} catch (SQLException ex) {
    // do something
} finally {
    if (con != null) con.close();
}
```

- Problem: not very feature rich (by specification)

  → definition of pool sizes, eviction strategy, etc. missing (features depend on driver)

What Java Already Provides (cont.)

NIO Socket Handling

- "Classic" socket handling: one thread per socket/client connection
  - → Does not scale
  - → What happens if 3000 clients connect?
- java.nio.* classes (nio = New I/O, since JDK 1.4)
- Support for multiplexed I/O (multiplexed = multiple signals/streams over single carrier)
- java.nio.channels.Selector handles many open sockets at the same time with one thread

→ Resource Lifecycle Manager Pattern

What Java Already Provides (cont.)

Soft References to Build Cache

- Soft, Weak, and Phantom References introduced in JDK 1.2 (but problematic implementation!) – fixed since JDK 1.4/5.0
- java.lang.ref.SoftReference
- An object with a SoftReference is cleaned up if the heap memory is low
  - → great for a cache: HashMap with SoftReferences to the values
    - → if memory is low: values are garbage collected
    - → if key is accessed, check of value is null
    - → if yes, then reload value, if no, then get strong reference and return value
- Pros: automatic memory management
- Cons: limited control; all values could be collected (better: keep limited number of strong references in HashMap)

Real-World Challenges

Controlling Amount of Incoming Data

- JMS Messaging server: clients as producers/consumers of messages
- Danger: several clients send large messages at once
- Solution: Flow-Control (start/stop of incoming traffic/client)

  → TCP does the same
- Centralized Resource Lifecycle Manager keeps counter of number of byte blocks allowed

  → If counter is low, clients (producers) are stopped

  → If counter is high, clients (producers) are started
- Solution works best with sockets, but also RMI is possible (custom ServerSocketFactory)
- Problems:
  - If byte blocks are not correctly freed or counter incremented, everything stops
  - Tricky to configure

Real-World Challenges (cont.)

When Producers Are Also Consumers: Software Agents

- ❒ No Flow-Control is possible (Agents are usually consumers and producers)
- ❒ “Ugly” trick: “paging” of new messages to disk
- ❒ Resource Lifecycle Manager checks high/low water marks
  - → If high watermark reached: incoming messages are stored on disk
  - → If low watermark reached: messages are loaded from disk
- ❒ Solution slows down application, but prevents a crash

→ Solution strongly influenced by ***operating system*** features!

Usage & Configuration

Usage

- Application deployed on different machines: different amount of resources available!
- Pool/Cache sizes should be ***configurable***
- Testing is easy with configurable pools: small pool size for testing

Configuration

- Configuration tricky: dependencies among resources!
- Dynamic configuration? – Cool but dangerous!

  → Possible solution: define different ***sets of configuration***

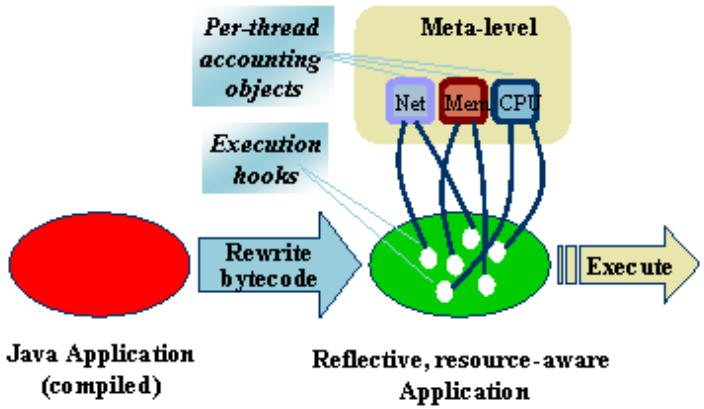
Always A Trade-off

- No free lunch: trade-off between Scalability, Performance, Stability, Predictability, Flexibility, Consistency, ...
- Resource Management often good choice – Example: thread pool (Stability, Performance)

Outlook

Advanced Resource Accounting

- Research project: JRAF2 - Java Resource Accounting Framework
  http://www.jraf2.org/
  - → Analyze bytecode of application
  - → Create wrappers of objects (AOP)
  - → Keep accounting for each thread (ThreadLocal?)
  - → CPU usage accounting is based on number of executed byte code instructions



Outlook (cont.)

More Features In JVM?

- Research group at Sun: Barcelona project (Lead: Dr. Grzegorz Czajkowski)
- In combination with JSR-121 (Isolation API)
- Idea:
  - Isolates: define groups of resources (memory area, threads, etc.)
  - Control these Isolates (security, limits of resource consumption)

Not Covered In This Talk

- java.lang.management (new in JDK 5.0)
- Several MXBeans with notifications for special events (low memory, etc.)

→ Great language features to build effective Resource Lifecycle Managers

Wrap-Up

Summary

- ❒ Overview through Patterns
- ❒ Discussed Java language features
- ❒ Presented Challenges
- → Something for everyone(?)

To Take Home With You

- ❒ ***THINK*** about resources and their usage – Resource Management might be appropriate
- ❒ Resource Management for ***stability*** and ... ***performance***
- ❒ Java provides more and more language features – why not use them
- ❒ You ***sleep better*** when your product is robust

References & Links

Patterns

- Pattern-Oriented Software Architecture, Vol.3 : Patterns for Resource Management
  Michael Kirchner, Prashant Jain – John Wiley & Sons, 2004

  → This book explains the presented Patterns. It doesn't offer much more and this was for me a bit disappointing. A good but not excellent book.

Java Features

- http://www.onjava.com/pub/a/onjava/2004/09/01/nio.html
  Article on using NIO for socket connections
- http://www.datadirect.com/developer/jdbc/docs/connpooling.pdf
  Short description of JDBC connection pooling: nice to get an overview
- http://java.sun.com/developer/JDCTechTips/2004/tt1116.html#2
  JDC Tech Tips on thread pooling
- http://www.devx.com/Java/Article/27439/
  Short article about threading and thread pooling with Java 5.0

References & Links (cont.)

- http://www.javaspecialists.co.za/archive/Issue098.html
  http://www.javaspecialists.co.za/archive/Issue015.html
  Two nice articles with a caching example based on soft references
- http://cpatutorials.skillspride.com/read/category/82/id/216/p/3
  Good article on GC and soft, weak, and phantom references

Outlook

- http://www.jraf2.org
  JRAF2 Project about Resource Accounting and Control: not too much information
- http://research.sun.com/projects/barcelona/
  Barcelona Project of SUN working on improving the JVM
- http://www.bitser.net/isolate-interest/papers/bryce-05.04.pdf
  Article on Isolation (JSR-121)