Performance Antipatterns

JUGS – Performance Abend

Mirko Novakovic

© 2007 codecentric GmbH

Agenda

- Patterns und Antipatterns
- Performance Antipatterns
- Fragen

Pattern (Entwurfsmuster)

- Ein Design Pattern ist eine bewährte Lösung für ein Entwurfsproblem.
  - „Gang of Four (GOF)“

- Elemente eines Pattern
  - Pattern Name
  - Problembeschreibung
  - Lösung
  - Konsequenzen

Antipattern

- Negativ Beispiel für bereits durchgeführte Lösungsmuster, die Hinweise zur Verbesserung geben.

- Elemente eines Antipattern
  - Antipattern Name
  - Symptome/Problembeschreibung
  - Konsequenzen
  - Lösung

Performance Antipattern

- Negativ Beispiel für eine Lösung die in schlechter Performance, d.h. schlechter Antwortzeit oder Durchsatz, resultiert.
  – Die Lösung kann auch im Widerspruch zu Design Patterns stehen (siehe auch J2EE Blueprints und JPetstore)

Performance Antipatterns

- Kategorien von Performance Antipatterns

Organisation/Kommunikation/Vorgehen
Architektur/Design
Implementierung/Infrastruktur

Performance Probleme

- Aus der Erfahrung von vielen Troubleshooting Einsätzen, sind die Top Performance Killer in Java Applikationen
  - Memory (Leaks, GC und Datenmenge)
  - DB Anbindung
  - Architektur/Design

Paralleles Schrauben

- Name: **Paralleles Schrauben**

- Symptome
  - Entwickler arbeiten parallel an Performance Tuning
  - Viele Ideen und Maßnahmen werden entwickelt und gleichzeitig umgesetzt
  - Die Performance Probleme werden nicht gelöst

- Konsequenzen
  - Tuning Maßnahmen beeinflussen sich gegenseitig
  - Viel Aufwand mit wenig Effizienz

Paralleles Schrauben

- Lösung
  - Implementierung eines Performance Teams
  - Zentrale Erfassung, Bewertung und Priorisierung der Tuning Maßnahmen
  - Umsetzung der Maßnahme mit der höchsten Priorität und Verifikation (Messung) der Wirkung bis Performance Ziele erreicht sind

Schuss ins Dunkle

- Name: **Schuss ins Dunkle**

- Symptome
  - Keine Performance Analyse Tools Vorhanden
  - Es werden eigene „Microbenchmarks“ geschrieben
  - Performance Experten fehlen
  - Performance Probleme werden mit dem Debugger und durch „Anschauen“ des Codes gesucht
  - Fingerpointing zwischen Entwicklung und Betrieb

Schuss ins Dunkle

- Konsequenzen
  - Entwickler verlieren Vertrauen beim Management
  - Stress Situationen im Projektteam
  - Es treten immer neue (Performance) Probleme auf
  - Probleme können nicht nachgestellt werden

Schuss ins Dunkle

- Lösung
  - Einführung von Tools (Profiler, Diagnose, Monitoring, Lasttest) für alle Layer der Applikation bis zur Hardware
  - Ausbildung der Mitarbeiter
  - Implementierung von Performance Management Prozessen
  - „Measure, don‘t guess!“
  - Feste Performance Ziele

Tools Überblick

- Profiler
  - Quest JProbe
  - dynaTrace diagnostics
  - JProfiler
  - YourKit
- Diagnose
  - Quest PerformaSure
  - dynaTrace diagnostics
  - JXInsight
- Monitoring
  - Quest Foglight
  - dynaTrace diagnostics
  - CA/Wily Introscope
  - IBM Tivoli Composite Application Manager (ITCAM)

Kombinationen

- Viele Antipatterns treten in Kombinationen auf
  - z.B. auch „Paralleles Schrauben“ und „Schuss ins Dunkle“
  - Die Wirkung wird meistens extrem verstärkt

Testdaten Falle

- Name: **Testdaten Falle**

- Symptome
  - Anwendungsfälle für Lasttests werden von Entwicklern erstellt
  - Lasttest-Ergebnisse sind gut und die Performance im Betrieb ist trotzdem schlecht
  - Es gibt keine Skripte zum Befüllen der Test-Datenbanken

Testdaten Falle

- Konsequenzen
  - Durch das Lasttesten von Use-Cases mit gleichen Daten werden unter Umständen nur gecachte Daten verwendet
  - Probleme durch zu kleine Cache/Pool Größen in DB und Anwendung werden nicht gefunden
  - Kritische Anwendungsfälle werden nicht getestet
  - Fehlende Indizes etc. werden nicht gefunden, weil nur mit den „Schlüsseln“ gearbeitet wird – die Realität sieht manchmal anders aus.
  - Durch geringe Datenmengen fallen bestimmte Probleme nicht auf – z.B. Full-Table-Scans

Testdaten Falle

- Lösung
  - Verwendung eines Coverage Tools
  - Caches/Pool/DB Monitoren und Analysieren
  - Testdaten realistisch wählen
  - DB Statistiken aus Produktion verwenden
  - JMX to the max!

Falsches Timing

- Name: **Falsches Timing**

- Symptome
  - Performance Messungen werden kurz vor dem Projektende geplant
  - Zu Beginn eines Projekts werden viele (Architektur) Entscheidungen auf Basis von Performance Annahmen getroffen

Falsches Timing

- Konsequenzen
  - Performance Maßnahmen werden aufgrund falscher Annahmen umgesetzt
  - Performance Engpässe können am Ende des Projekt zu großen Refactoring Maßnahmen führen
- Lösung
  - Kontinuierliches Messen von Performance als Teil des Continuous Integration Prozesses (mit den richtigen Tools)
  - Iteratives Vorgehen

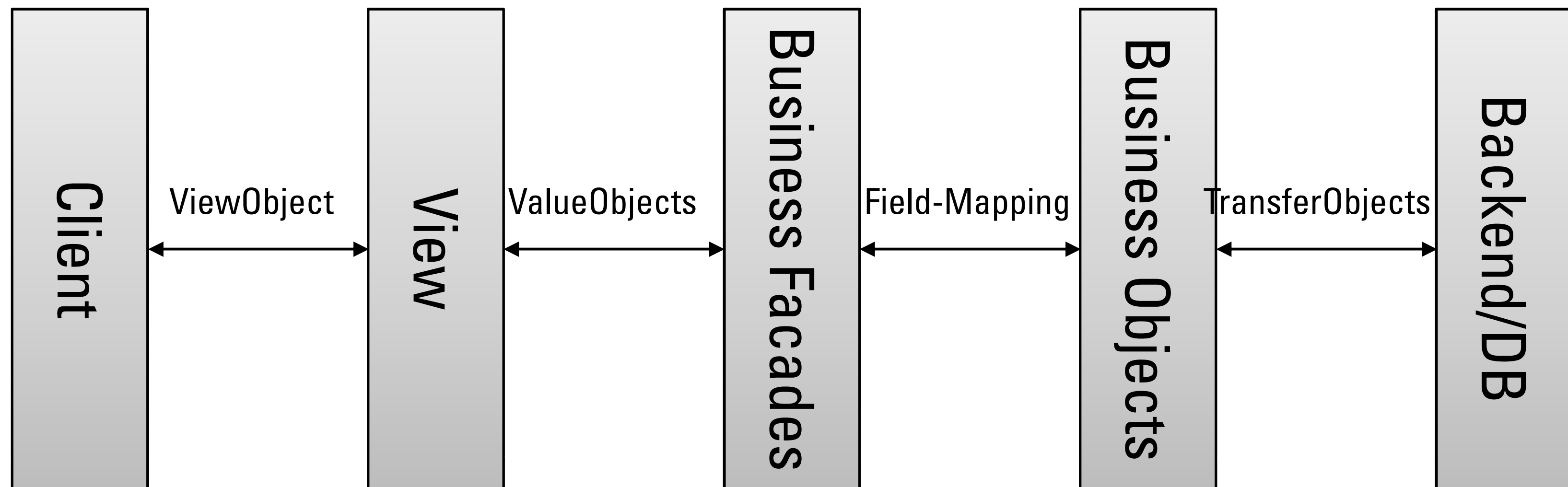
Multi Layering

- Name: **Multi Layering**

- Symptome
  - Viele logische Schichten in der Architektur
  - Entwickler verbringen viel Zeit beim „Mapping“ von Daten
  - Es kostet viel Aufwand einen „Durchstich“ vom Frontend bis ins Backend/DB umzusetzen
  - Anwendungen werden für unterschiedliche Clients und Backends designed

Multi Layering

- Multi Layer Architektur



Multi Layering

- Konsequenzen
  - Zeitverlust beim Mapping und Konvertieren von Daten
  - Cycling Object Problematik und hohe GC Belastung

- Lösung
  - Angemessene Anzahl Schichten verwenden (Anforderungen!)
  - Neue Framework Ansätze wie JBoss Seam

Manuelles Reporting

- Name: **Manuelles Reporting**

- Symptome
  - Es werden viele Auswertungen und Statistiken benötigt (gerade im Retail Bereich)
  - SQL Statements sind mehrere Seiten lang und beinhalten viele Joins und Subselects
  - Benutzer beschweren sich über minutenlange Antwortzeiten

Manuelles Reporting

- Konsequenzen
  - Der Aufwand für die Optimierung und Pflege der SQL Statements ist sehr groß
  - Eine gute Antwortzeit wird trotzdem nicht erreicht

- Lösung
  - Nicht jeder Anforderung kann in wenigen Sekunden online umgesetzt werden!
  - Business Intelligence und Datawarehouse Technologien verwenden
  - Eigene Tabellen für Report Daten anlegen und im „Batch“ Betrieb füllen

Session Cache

- Name: **Session Cache**

- Symptome
  - Hohe Auslastung des Heaps
  - Nur geringe Anzahl paralleler Benutzer möglich
  - Performance Probleme treten verstärkt bei Cluster-Anwendungen auf

Session Cache

- Konsequenzen
  - Die Session wird als Daten-Cache genutzt – gerade auch für Objekte, die aus relationalen Daten befüllt werden
  - Zusätzliche Hardware wird benötigt
  - Clustering für Stabilität nicht möglich
  - Session Größen sind meistens nicht bekannt und können auch nach Angabe der Entwickler nicht verkleinert werden

Session Cache

- Lösung
  - Memory Analyse durchführen
  - In die Session gehören nur „sitzungsbehaftete“ Daten, die nicht wiederhergestellt werden können
  - Andere Daten sind Cache Kandidaten – das Caching sollte in der Schicht erfolgen, die am Besten über die Datenverwaltung entscheiden kann (z.B. DB oder O/R Mapper)
  - Caches aufteilen in Lokal, JVM übergreifen, Cluster übergreifend
  - Timeouts zur Vermeidung von Memory Leaks verwenden
  - Möglichst keine eigenen Caches verwenden bzw. nur „teure“ Daten cachen

Unterschätztes Frontend

- Name: **Unterschätztes Frontend**

- Symptome
  - Es gibt einen Thin-Client auf Basis eines HTML Frontends
  - Die Anwendung verfügt über Rich-Client Funktionen
  - Die HTML Seiten sind sehr groß und es wird viel mit Bilder gearbeitet
  - Die Anbindung der Benutzer hat teilweise eine schmale Bandbreite (Mobilfunk, ISDN, DSL mit vielen Anwendern)

Unterschätztes Frontend

- Konsequenzen
  - Übertragungszeit der HTML Seiten sehr groß – dadurch lange Wartezeiten
  - Viele „Roundtrips“ zum Webserver – kritisch vor allem bei schlechter Latenzzeit
  - Seiten werden häufig komplett neu übertragen – z.B. durch Auswahl in einer Listbox
  - Auch statischer Content wird immer neu gerendert

Unterschätztes Frontend

- Lösung
  - Verwendung von CSS
  - Auslagerung von Javascript und CSS Daten in eigene Dateien (Browser Caching)
  - Setzen von time-to-live für statische Daten durch Proxy
  - Einsatz von AJAX z.B. für Listboxen
  - Nur 80% benötigte Daten/Felder anzeigen und andere Felder auf eigene Seiten auslagern
  - Caching für statischen Content verwenden (OSCache, CMS Funktionalität)
  - GZIP Funktion auf Webserver einschalten (seit HTTP 1.1)
  - YSlow Firefox Plugin von Yahoo mit sehr nützlicher Analysefunktion

Phantom Logging

- Name: **Phantom Logging**

- Symptome:
  - Keine isDebugEnabled() Abfragen

- Konsequenzen
  - Zu loggender String wird immer erzeugt – inkl. String Konkatinationen und toString() Methoden

- Lösung
  - isDebugEnabled() einbauen ☺

Kontakt

Mirko Novakovic
novakovic@codecentric.de
+49-(0)163-6681500

codecentric GmbH
Grünewalder Str. 29-31
42699 Solingen
http://www.codecentric.de