14.–17. 09. 2009
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Die Laier

Java Performance

Stefan Hildebrandt

consulting.hildebrandt.tk

Inhalt

- Einleitung
- Motivation
- Vergleich Messmethoden
  - Profiling Lösungen
  - Tracing Lösungen
  - AOP Eigenentwicklung
- Praxis
- Messpunkt Bibliothek

Inhalt

- Einleitung
- Motivation
- Vergleich Messmethoden
  - Profiling Lösungen
  - Tracing Lösungen
  - AOP Eigenentwicklung
- Praxis
- Messpunkt Bibliothek

Premisse

"Wer mißt,
mißt Mist"

Einleitung

- Java hat nach wie vor den Ruf, langsam zu sein
  - Ursprung liegt in den 1. Swing-Anwendungen und dem Startverhalten der VM
  - Für die meisten Anwendungsfälle ist Java schnell genug
    - Insbesondere auf dem Server
  - Performanceprobleme entstehen in der Regel durch
    - Architekturprobleme
    - An Schnittstellen zu anderen Systemen
    - Durch Fehler in der Entwicklung
    - Unkenntnis der Funktionsweise von Frameworks und Application Servern
- Ziel: *Erkennen* und beseitigen dieser Probleme

Inhalt

- Einleitung
- Motivation
- Vergleich Messmethoden
  - Profiling Lösungen
  - Tracing Lösungen
  - AOP Eigenentwicklung
- Praxis
- Messpunkt Bibliothek

Anforderungen?

- Es gibt immer irgendwelche Anforderungen
  - Durchsatzanforderungen
  - Anwenderempfinden
    - Nicht direkt messbar
    - Reaktionszeiten bis 0,5 Sekunden tolerabel
      - Bei Webanwendungen ist das Rendern mit einzubeziehen
- Alle Anforderungen sind zu
  - Erfassen
  - Bekannt zu machen

Konzept?

- Anforderung sind schon im Lastenheft zu erfassen
- Performance-Betrachtung sollte zusammen mit den Entwicklungsarbeiten starten
  - Später können hohe Refactoring-Kosten entstehen
  - Zeitpläne gefährdet werden
- „Optimierungen“ sollten nur nach Erfolgsnachweis übernommen werden
  - In der Regel schlechter Code und schlechtere Wartbarkeit
  - Testaufwand für nicht notwendige Änderung

Probleme? Noch nicht?!?

- Ein Kunde liefert plötzlich wesentlich mehr Daten
- Nach mehreren Jahren wird ein System plötzlich unbenutzbar?
  - Der Betrieb kann nicht erkennen welches System langsamer geworden ist
  - Es stehen keine Entwickler mehr für die Fehlersuche zur Verfügung
- Fazit
  - Auskunftsfähigkeit jederzeit sicherstellen
    - Betriebsverantwortliche sollten einbezogen werden

Wer wars?

- Probleme häufig in oder an der Schnittstelle zu Nebensystemen
  - Ineffiziente Aufrufe
  - Fehlerhafte Datenbank-Konfiguration
    - Indexes, Optimizer, …
  - Externe Services schlecht erreichbar
    - Netzwerk, Durchsatzbeschränkung, ...
  - IO Probleme
    - Falsche SAN-Konfiguration, …

Wieso?

- Entwicklung mit komplexen Frameworks
  - Fehleranfällig, da Auswirkungen nicht immer bekannt
  - Fehlersuche schwierig, da tiefgreifendes Verständnis der internen Funktion der Frameworks notwendig
- Einsatz bestimmter Features
  - Datenbank-Design
  - Berichte zur Laufzeit
  - UI-Widgets
- “Kosten“ sollten erkennbar sein
- Ausmessen über Logging zu aufwendig
  - Eventuell zu ungenau: Timer-Auflösung unter Windows

TODO

- Anforderungen sammeln
- Gesamtkonzept erstellen
- Messmethoden etablieren
- Umgang schulen
- Entwicklungsprozess anpassen
- Entwicklung überwachen

Entwicklungsprozess (Entwurf)

- „Optimierungen“ während der Entwicklung nur dort, wo Probleme bestehen
  - nie auf Verdacht und präventiv!
  - Ausnahme: Best Practices
- Mirco-Performance-Optimierungen nur wo sie den Code nicht verschlechtern
  - StringBuilder, if(Logger.isDebug()){...}
    - Aufwendige Loggenerierung → ja
  - Bit-Operationen → NEIN
- Refactoring nur mit messbarem Erfolg einbauen
- Code-Qualität sollte nur in begründeten Fällen leiden!

Inhalt

- Einleitung
- Motivation
- Vergleich Messmethoden
  - Profiling Lösungen
  - Tracing Lösungen
  - AOP Eigenentwicklung
- Praxis
- Messpunkt Bibliothek

Anforderungen an Messwerkzeuge

- Unterschiedliche Zielsetzungen
  - Entwicklung
    - Relative Genauigkeit
      - Führt zu Verschiebungen mit Nebensystemen
    - Einfache Identifizierurng von Hotspots
    - Blättern in Messwerten
    - Tracing
  - Betrieb
    - Wenig Overhead
    - Tracing ?!?
    - Keine Beeinträchtigung der Stabilität
      - Speicherverbrauch
      - CPU-Last

Fachhändler

- Fertige Lösungen (Jprobe, Dynatrace, ...)
- Für bestimmte Umgebungen vorbereitet (z.B. für komerzielle Application-Server)
- „Automatische Messpunkt Auswahl“
- Für Produktivsysteme geeignet
- Tracing über Knotengrenzen hinweg
- Automatisches Setup
  - Java-VM wird modifiziert
    - Nicht für jedes System lieferbar
    - Eventuell Supportverlust
- Serversysteme mit vielen CPUs → Sehr hohe Kosten
  - Stagingstuffen sollten auch einbezogen werden

Baumarkt

- Profiler für den Desktop
- Einbindung in die IDE
- Hotspot-Erkennung
- Sehr unterschiedliche Möglichkeiten zur Datenauswertung
- Nicht für Produktivumgebungen geeignet
  - In der Regel auch nicht auf Stagingstuffen

Eigenbau

- OpenSource-Lösungen
  - InFraRed
  - Java Execution Time Measurement Library (Jetm)
  - Kieker
  - …
- Komplette Eigenentwicklung
- Erweiterung der Messpunkte über AOP notwendig

AOP basierte Ansätze

- Einsatz eines beliebigen AOP Frameworks
  - Java EE 5 Interceptoren, Spring AOP, AspectJ, …
- Bestimmung der Messpunkte
- Einbauen in die Anwendung
  - Container
  - Load-Time-Weaving
    - Einfache Anwendung auch in Bibliotheken
    - Schneller Roundtrip in der Entwicklung von Messpunkten
    - Langsamerer Start der Anwendung
  - Compile-Time-Weaving
    - Für Produktivumgebungen besser geeignet
    - Komplizierteres Setup für Bibliotheken
    - Systembibliotheken (fast) nicht erreichbar

Bestimmung von Messpunkten

- Messung an Schnittstellen
  - Triggerevent (GUI-Event, Http-/Webservice-Request, ...)
  - Ausgang (Datenbank, Webservice-Aufrufe, Dateisystem-Zugriffe)
- Eigene Services, Actions, …
- Mapping !!!
- Zur Gliederung
- Betrachtung eines Triggerevents
  - Bestimmung aller nennenswerten Zeitverbraucher
  - Ziel 80-90% der Zeit erklären
    - Nur hier ist Potential für messbare Verbesserungen
- Die ersten Aufrufe sind in der Regel mit Initialisierungsaufwand belegt

InfraRED

- Pro
  - Clusterfähig
  - Aggregierte Ergebnisse
  - Tracing für letzten Durchlauf
  - Http-Console
  - Einfacher Einsatz durch einige vordefinierte Messpunkte
  - Produktivtauglich (zukünftig?)
- Kontra
  - Mäßige Genauigkeit
  - Projekt seit 3 Jahren „tot“
- http://infrared.sourceforge.net

Jetm

- Pro
  - Hohe Genauigkeit
  - Aggregierte Ergebnisse mit Nested Zusammenfassung
  - Http-Console
    - Auch außerhalb eines Web-Containers
  - JMX-Client
  - Erweiterbar über Plugins
  - Aus der Praxis und produktionstauglich
- Konta
  - Kein Trace-Support
  - Messpunkte müssen selbst bestimmt werden
- jetm.void.fm

Kieker

- Pro
  - Tracing
  - HttpConsole
  - JMX Einbindung
  - Interne Funktionsweise gut dokumentiert
- Kontra
  - Mäßige Genauigkeit
  - Wenig Praxiserfahrungen
  - Messpunkte müssen selbst bestimmt werden
- http://kieker.sourceforge.net/

Alternativen

- Dtrace
  - Inbesondere bei IO-Problemen
- Externe Messung mit Lastgeneratoren
  - Einhaltung von SLAs
  - Bestimmung des Erfolges von Optimierungen

Inhalt

- Einleitung
- Motivation
- Vergleich Messmethoden
  - Profiling Lösungen
  - Tracing Lösungen
  - AOP Eigenentwicklung
- Praxis
- Messpunkt Bibliothek

Weben von Bibliotheken mit Maven

- Anlegen einer neuen pom.xml
  - Versionsnummer um AOP-Lib-Version erweitert
  - Abhängigkeiten bestimmen (runtime, provided)
  - Zusätzlich:Ursprüngliche Library, Aspect und Aspectj
  - Wichtig: Dummy Klasse anlegen
  - Maven-aspectj-plugin nutzen
  - Einfache Umschaltung zwischen der gewebten und der ursprünglichen Lib durch Apendix an der Versionsnummer.
  - Automatischer Einbau neuer Aspekt-Versionen
  - Einfache Aktualisierung und Verteilung der Bibliothek im Releaseprozess

Beispiele

Web-Pitfals

- Ständiger Durchgriff auf die Datenbank
  → Caching in Page-Scope Variablen
- Entkoppeltes Backend
  - Extensives Mapping zwischen Frontend- und Backend-Model
- Unausgereifte Komponenten
- Ajax-Zugriffe
  - Client-Site-State
- Memory-Leaks in der HttpSession

Hibernate

- Viele Einzelzugriffe vs. allumfassender Join
  - Je nach Join-Anzahl, Kreuzproduktbildung, 2. Level Cache Nutzung , … kaum vorhersagbare Ergebnisse
  - Muss im Einzelfall betrachtet werden
  - Kann sich im Betrieb durch unterschiedliches Lastprofil ändern

Inhalt

- Einleitung
- Motivation
- Vergleich Messmethoden
  - Profiling Lösungen
  - Tracing Lösungen
  - AOP Eigenentwicklung
- Praxis
- Messpunkt Bibliothek

Messpunkt Bibliothek

- Bestimmung aussagekräftiger Messpunkte
  - Aufwendig
  - Kenntnisse über die Funktionsweise von Bibliothen notwendig
- Bibliotheken haben in allen Projekten potentiell die selben Messpunkte.
- Dieses Wissen kann in einer Gemeinschaft gepflegt werden

Vielen Dank!

Stefan Hildebrandt

consulting.hildebrandt.tk