

Troubleshooter oder Troublemaker

Der Entwickler und seine Feuerwehr

Rainer Jung
Geschäftsführer
kippdata informationstechnologie GmbH

Gliederung

- Aufwärmrunde
- Typische Problemsituationen
- Beobachtbarkeit und Beobachtung
- Zusammenfassung

- Aufwärmrunde
- Typische Problemsituationen
- Beobachtbarkeit und Beobachtung
- Zusammenfassung

Aufwärmrunde I

- Wer bin ich?
 - Geschäftsführer kippdata
 - Erfahrungshistorie: Schwerpunkt Systemintegration
 - Hinzufügen von Produktionsqualitäten (Performance, Ausfallsicherung) zu (leider) meist schon fertigen Anwendungen
 - Troubleshooting (über den ganzen Stack hindurch) und auf dieser Basis dann ...
 - Open Source-Leser (Problemanalyse) und -Contributor
 - Apache Tomcat-Committer
 - Vor allem mod_jk (Loadbalancing), Cluster (Ausfallsicherung)

Aufwärmrunde II

- Wer sind Sie?
 - Wer würde sich als (Java-)Entwickler bezeichnen?
 - Wer schreibt Code für verteilte Systeme?
 - Wessen Code ist kein Individualcode, d.h. er läuft in der gleichen Version in mehr als 10 Installationen?
 - Wer trägt Verantwortung für Produktionsstabilität bzw. Fehleranalyse?
 - Wer musste schon Probleme debuggen, die erst in der Produktion beobachtet wurden?
 - Wer hat mit 24x7x365-Systemen zu tun?

Aufwärmrunde III

- **Wie arbeiten Sie?**
 - Wer hat schon einmal ein Java-MBean geschrieben?
 - Wird es in Produktion genutzt?
 - Wer hat schon einmal einen Java-Thread-Dump analysiert?
 - Erfolgreich?
 - Bei wem gibt es eine schriftliche Richtlinie fürs Logging?
 - Passt diese in der Realität?
 - Bei wem basiert HW-Kapazitätsplanung auf gemessenen Werten?
 - Gilt dies auch für das Sizing von SW-Komponenten (Pools etc.)?

Aufwärmrunde IV

- Dieser Vortrag
 - Betrifft im wesentlichen Server-Anwendungen
 - Enthält keinen Code – und kein Marketing
 - Stellt keine **neuen** Frameworks oder APIs vor
 - Versucht meine persönlichen Erfahrungen aus den letzten acht Jahren **Produktions**-Troubleshooting von Java-Anwendungen wiederzugeben
 - Verursacht zusätzliche Kosten in Ihrer Software-Entwicklung (Troublemaker) und Infrastruktur
 - Verringert laufende Kosten im Anwendungsbetrieb!?

- Aufwärmrunde
- **Typische Problemsituationen**
- Beobachtbarkeit und Beobachtung
- Zusammenfassung

Typische Problemsituationen – Abstrakte Sicht

- **Abstrakte Problembeschreibung**
 - **Performanceprobleme**
 - Anwendung ist zu langsam (einzelne Vorfälle)
 - Es bilden sich Staus aus
(in Folge ist die Anwendung nicht mehr erreichbar)
 - **Stabilitätsprobleme**
 - Anwendung reagiert nicht mehr
- **Wir betrachten nicht**
 - Probleme der fachlichen Korrektheit

Typische Problemsituationen – Mögliche Rahmenbedingungen

- **Mögliche Rahmenbedingungen**
 - Probleme treten sporadisch auf, kein Muster **bekannt**
 - Dann zunächst meist nicht reproduzierbar
 - Probleme treten zu Hochlastzeiten auf
 - Reproduktionschancen besser (Ja ?)
 - Probleme treten nach einiger Zeit auf
 - Reproduktionschancen unklar

Typische Problemsituationen – Ausgangslage

- Ausgangslage
 - A) Schlechte Beobachtbarkeit
 - Umkehrschluss: Probleme mit guter Beobachtbarkeit lösen die Kunden alleine
 - B) Häufig Überlagerung mehrerer Probleme
 - Problembeobachtungen meist nicht exakt in der Zeit aufgelöst, protokolliert etc.
 - A)+B) => **Es ist extrem wichtig echte Beobachtungen von vermuteten Ursachen zu unterscheiden**

Typische Problemsituationen – Typ 1: Locking I

- Locking als Problemursache
 - Locking/Synchronisierung ist nicht böse, sondern absolut notwendig!
 - Aber
 - Die Lockbelegung darf nicht zu lange dauern
 - Die Akquisitionsrate darf nicht zu hoch sein
 - Genauer: Das Produkt aus beiden muss deutlich kleiner als 1 sein
 - Beispiele
 - Mittlere Belegung 10 Mikrosekunden, 2.000 Locks/Sekunde OK
 - Mittlere Belegung 5 Millisekunden, 4 Locks/Sekunde OK

Typische Problemsituationen – Typ 1: Locking II

- **Mögliche Auswirkungen von Lock-Problemen I**
 - **Sonderfall Deadlock**
 - Konsequenz Stillstand der betroffenen Threads, häufig der ganzen Anwendung
 - **Lock-Contention**
 - Quasi-Serialisierung eines Teils der Anwendung
 - Performance-Engpass auf Multi-CPU/-Cores Hardware
 - Das Phänomen tritt nicht-linear auf:
 - Bei 50% der möglichen Rate in der Regel nicht beobachtbar
 - Bei 80% gravierende Einschränkungen bis hin zum Stau

Typische Problemsituationen – Typ 1: Locking III

- **Mögliche Auswirkungen von Lock-Problemen II**
 - **Thread-Starvation**
 - Vor allem bei stark genutzten Locks
 - Einzelne Threads bekommen den Lock nicht oder zu selten
 - Locks haben keine Fairness
 - Es gibt Locks mit Fairness, aber
 - Fairness bedeutet komplexer Code (langsam)
 - Fairness kann konträr zum OS-Scheduler sein (langsam)
 - Fairness erhöht also in der Regel die Log-Contention

Typische Problemsituationen – Typ 1: Locking IV

- Locking-Regeln I
 - Vermeidung Lock-Contention
 - Keine komplexen Operationen während ein Lock gehalten wird (komplex = lange dauernd)
 - Insbesondere keine remote Aufrufe!
 - Weniger Information sharen, mehr echte Unabhängigkeit
 - Beschäftigen Sie Sich mit den Möglichkeiten von JSR-166 (Concurrency Utilities)
 - Häufig Probleme auch in Hilfsbibliotheken
 - Pools, Caches

Typische Problemsituationen – Typ 1: Locking V

- Locking-Regeln II
 - Vermeidung Deadlocks
 - Total Lock Ordering

„I believe the deadlock could only be triggered by a contrived test that creates an excessive number of threads that would never been used in a normal application.“

- dict.leo.org:
contrived = arrangiert, erfunden, gekünstelt, gestellt
- In der Regel aber kein allgemeines Rezept, sondern fallbezogene Lösung
- Und damit kommen wir zu ...

Typische Problemsituationen – Typ 1: Locking VI

- Wie finden wir Lock-Probleme?
 - Thread-Dumps !
 - Thread-Dumps !!
 - Thread-Dumps !!!
- Dazu gleich mehr ...

Typische Problemsituationen – Typ 2: Sizing I

- Viele Anwendungen haben Komponenten, die konfigurierbare Größen haben
 - Oder verwenden solche direkt
 - Oder verwenden Container, die solche haben
- Beispiele
 - Größe Thread-Pools, Größe Verbindungs-Pools (Datenbanken, Middleware, Web-Services)
 - Größe Caches
 - Timeouts

Typische Problemsituationen – Typ 2: Sizing II

- Bei Produktionsstart häufigstes Sizing:
 - Default
- Manchmal auch
 - Pi mal Daumen
 - In der Regel aber nicht einmal konsistent
 - Beispiel: Container-Threads, DB-Pool, DB-Prozesse

Typische Problemsituationen – Typ 2: Sizing III

- Verständlich ist
 - Schlechtes Sizing wegen mangelnder Vorerfahrung
 - Inkonsistente Einstellungen wegen fehlender Dokumentation
- Nicht verständlich ist
 - Fehlende Beobachtung der sich unter Produktionsbedingungen ergebenden Größen ...
 - ... und deren Entwicklung!

Typische Problemsituationen – Typ 2: Sizing IV

- Zusatznutzen der Produktionsbeobachtung
 - Entstehende Erfahrung zu sinnvollem Sizing
 - Entstehende Erfahrung zum Zusammenspiel der Größen
- ~~Verständlich~~ ist
 - Schlechtes Sizing wegen mangelnder Vorerfahrung
 - Inkonsistente Einstellungen wegen fehlender Dokumentation

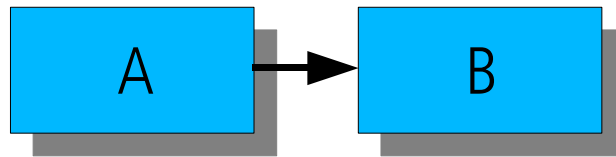
Typische Problemsituationen – Typ 2: Sizing V

- Hier ist ein Zusammenspiel von Entwicklung und Betrieb nötig
 - Monitoring und Kapazitätsmanagement ist Betriebsache ...
 - ... der Betrieb benötigt jedoch Unterstützung seitens der Entwicklung, solange die Grunderfahrung nicht vorliegt
- Zum Monitoring gleich mehr ...

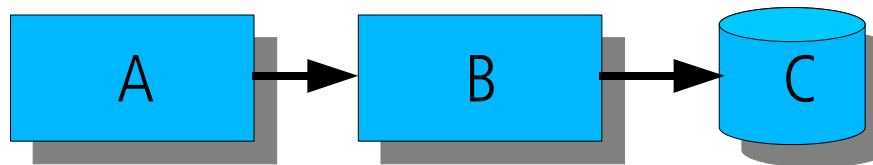
Typische Problemsituationen – Typ 3: Remote Probleme I

- Performance-Probleme bei verteilten Systemen

- Die Ursache ist nicht meine Komponente A, es ist wohl B



- Die Ursache ist nicht meine Komponente B, es ist wohl C



- Die Datenbank C langweilt sich
- Dann ist es wohl das Netz!
- Die Erfahrung zeigt: meistens nicht.

Typische Problemsituationen – Typ 3: Remote Probleme II

- Erneut ist die Beobachtbarkeit von zentraler Bedeutung
- Hier reicht jedoch meist ein Thread-Dump nicht aus
- Bei Problemanalyse zwischen grob-granularen Komponenten ist Logging sehr hilfreich
 - Insbesondere bei remote Calls
 - Insbesondere bei Wechsel von Technologie und/oder Zuständigkeit (Legacy, Datenbank, SAP, ...)
- Zum Logging gleich mehr ...

Typische Problemsituationen – Typ 4: Ressourcen-Engpass I

- Anwendungen nutzen Hardware-Ressourcen
 - CPU
 - Memory
 - I/O (Platte, Netz)
- Ein Engpaß an Ressourcen führt zu Performance-Problemen, seltener auch zu Instabilitäten

Typische Problemsituationen – Typ 4: Ressourcen-Engpass II

- Die Beobachtung und Analyse der meisten Engpässe bei HW-Ressourcen ist recht gut verstanden
 - CPU: leicht zu beobachten, Analyse durch
 - Zuordnung zu Prozessen und dann Threads
 - Thread-Dumps, nach Eingrenzung Profiling
 - Netz: Beobachtung durch Netzwerksniff, Analyse durch
 - Zuordnung zu TCP-Connections
 - Protokollanalyse
 - Platte: Beobachtung durch OS-Tools, Analyse durch
 - Zuordnung zu Dateien und Inhalt

Typische Problemsituationen – Typ 4: Ressourcen-Engpass III

- Die Beobachtung und Analyse von Engpässen im Bereich Memory ist weniger gut verstanden
 - Memory wird komplex als virtuelles Memory verwaltet
 - Das ideale Betriebssystem hat kein freies Memory
 - Weil es freies Memory automatisch für Caches verwendet
 - Oberhalb des virtuellen Memories im OS sitzt die Java-Memory-Verwaltung und Ihre GC-Algorithmen

Typische Problemsituationen – Typ 4: Ressourcen-Engpass IV

- Java-Memory und GC I
 - Zu konfigurieren sind
 - Die Größen von vier Memory-Bereichen
 - Eden, Survivor, Tenured, Perm
 - Die Algorithmen für zwei Garbage Collections
 - Minor GC, Major GC
 - Es gibt wenig korrekte und umfassende Dokumente dazu
 - Sie auch mein Vortrag JAX 2007
„Memories are made of this“
 - Oder kippdata-Workshop
"Java Garbage Collection - Theorie und Praxis"

Typische Problemsituationen – Typ 4: Ressourcen-Engpass V

- Java-Memory und GC II
 - Beobachtbarkeit
 - Die vermeintlich einfachen Verfahren sind Poll-Verfahren
 - Zu beobachten sind aber einzelne diskrete Events
 - Wie pollt man Events, von denen man nicht weiss, wann sie passieren?
 - Notausgang: (sehr) verboses GC-Logging

Typische Problemsituationen – Typ 4: Ressourcen-Engpass VI

- Java-Memory und GC III
 - GC-Logging
 - Keine absoluten Zeitstempel
(doch, neu seit dem letzten JVM 6-Update)
 - Keine Rotation
 - Kein einheitliches Format
(nur für GC-Entwickler gedacht gewesen)
 - Low-level-Information, schlecht zu parsen

Typische Problemsituationen – Typ 4: Ressourcen-Engpass VII

- Ressourcen-Virtualisierung
 - Unsere Schwierigkeiten bei der Memory-Analyse lassen befürchten ...
 - ... dass wir in Zukunft auch bei den verstandenen Ressourcen wie etwa CPU wieder Beobachtungs- und Analyseprobleme bekommen

Typische Problemsituationen – Lösungsansätze

- Lösungsansätze zur Ursachenfindung
 - Beobachtbarkeit herstellen
 - Thread-Dumps
 - Logging
 - Monitoring
- Darauf wollen wir jetzt näher eingehen

- Aufwärmrunde
- Typische Problemsituationen
- **Beobachtbarkeit und Beobachtung**
 - Thread-Dumps
 - Logging
 - JMX
 - Monitoring
- Zusammenfassung

Beobachtbarkeit und Beobachtung – Thread-Dumps I

- Inhalt eines Thread-Dumps
 - Ein Thread-Dump ist eine Momentaufnahme der Code-Ausführung in der JVM
 - Er enthält eine Liste aller Threads in der JVM
 - Mit Name und ID, sowie Zustand (etwa runnable)
 - ID meist abbildbar auf die Thread-Nummern des OS
 - Mit komplettem Funktionsstack der Java-Methoden
 - Mit Informationen bzgl. des Wartens auf Locks
 - Mit Ausgabe, ob ein Deadlock vorliegt

Beobachtbarkeit und Beobachtung – Thread-Dumps II

- **Momentaufnahme**
 - Meist muss mehr als ein Thread-Dump gemacht werden, um Zufallsbeobachtungen auszuschliessen
 - Z.B. 3 Dumps im Abstand von jeweils 3 Sekunden
- **Thread-Dumps sind ein JVM-Feature**
 - Klappt also für alle Java-Prozesse!
- **Thread-Dumps gehen sehr schnell**
- **Thread-Dumps sind OK in Produktion ! (ca. ab 1.4.2_10)**
- **Auch regelmässig (Abstand Stunden, nicht Sekunden)**

Beobachtbarkeit und Beobachtung – Thread-Dumps III

- Wohin geht der Dump?
 - Nach STDOUT
 - Im Startskript auffangen
 - Zeitstempel hinzufügen
 - Rotieren
 - Oder Service-Wrapper verwenden
 - Oder über Platform MBeans holen

Beobachtbarkeit und Beobachtung – Thread-Dumps IV

- Wie wird ein Dump erzeugt?
 - Unix/Linux: sende QUIT-Signal an Prozess (`kill -QUIT PID`)
 - Das darf der root-User und der Owner des Prozesses
 - Es beendet **nicht** den Prozess! Der Name ist irreführend.
 - Windows: send Break-Signal an den Prozess
 - Das darf nur ein Prozess, der in der gleichen Console-Gruppe wie die JVM ist
 - Meist nur auf Entwickler-PC oder in speziellen Fällen in Produktion so machbar
 - Start über DOS-Box

Beobachtbarkeit und Beobachtung – Thread-Dumps V

- Was machen wir bei einem Windows-Service?
 - Hat kein Terminal, deshalb klappt Break-Signal nicht
- Ab Java 5 gibt es einen programmatischen Weg, aus der Java-Anwendung heraus Thread-Dumps aufzurufen
 - **java.lang.management.ThreadMXBean**
- Vorsicht: der Dump ist etwas weniger aussagekräftig
 - Weniger Lock-Information (besser ab Java 6)
 - Weniger IDs zum Thread
 - Keine JVM-internen Threads

Beobachtbarkeit und Beobachtung – Thread-Dumps VI

- Aufrufmöglichkeiten MBean-basierter Thread-Dump (vor allem für Windows Service relevant)
 - Aufrufbar über JMX-Schnittstelle
 - Ab Java 6 ohne vorherige Aktivierung der Schnittstelle (Attach on Demand)
 - jstack (kann aber keine Authentisierung)
 - `<JDK_HOME>/demo/management/FullThreadDump/FullThreadDump.jar`
 - Eigener Client
 - Oder kapseln in HTTP-Servlet oder ähnlichem Aufrufweg
 - Beispiel: kpdexplorer

Beobachtbarkeit und Beobachtung – Thread-Dumps VII

- Spezialfall Windows und Tanukisoft Service Wrapper
 - Verwende Klasse
`org.tanukisoftwrapper WrapperActionServer`
 - Beispiel für Tomcat:
<http://people.apache.org/~fhanik/wrapper.html>
 - Hat aber keine Produktionsqualität (fehlende Authentisierung)

Beobachtbarkeit und Beobachtung – Thread-Dumps VIII

■ Beispiel Thread-Dump

Full thread dump Java HotSpot(TM) Server VM (1.6.0-beta2-b72 mixed mode):

```
"main" prio=10 tid=0x00030c00 nid=0x2 runnable [0xfe67e000..0xfe67fd80]
  java.lang.Thread.State: RUNNABLE
    at java.net.PlainSocketImpl.socketAccept(Native Method)
    at java.net.PlainSocketImpl.accept(PlainSocketImpl.java:384)
    - locked <0xf3cafbb0> (a java.net.SocksSocketImpl)
    at java.net.ServerSocket.implAccept(ServerSocket.java:450)
    at java.net.ServerSocket.accept(ServerSocket.java:421)
  ...
    at org.apache.catalina.startup.Bootstrap.start(Bootstrap.java:295)
    at org.apache.catalina.startup.Bootstrap.main(Bootstrap.java:433)
  ...
```

Beobachtbarkeit und Beobachtung – Thread-Dumps IX

■ Beispiel Thread-Dump - Fortsetzung

```
"http-28380-Processor2" daemon prio=10 tid=0x00968800 nid=0x1a runnable ...
  java.lang.Thread.State: RUNNABLE
    at java.net.PlainSocketImpl.socketAccept(Native Method)
    at java.net.PlainSocketImpl.accept(PlainSocketImpl.java:384)
    - locked <0xf4490718> (a java.net.SocksSocketImpl)
    at java.net.ServerSocket.implAccept(ServerSocket.java:450)
    at java.net.ServerSocket.accept(ServerSocket.java:421)
    at java.lang.Thread.run(Thread.java:626)
```

```
"http-28380-Processor1" daemon prio=10 tid=0x00969800 nid=0x19 in Object.wait()
  java.lang.Thread.State: WAITING (on object monitor)
    at java.lang.Object.wait(Native Method)
    - waiting on <0xf4202130>
    (a org.apache.tomcat.util.threads.ThreadPool$ControlRunnable)
    at java.lang.Object.wait(Object.java:484)
    at org.apache.tomcat.util.threads.ThreadPool$ControlRunnable.run
    (ThreadPool.java:687)
```

...

Beobachtbarkeit und Beobachtung – Thread-Dumps X

■ Ausgabe Deadlock

Deadlock found :-

```
"ajp-127.0.0.1-8009-42" Id=486 in BLOCKED on lock=java.lang.Object@1d461d7
  owned by ajp-127.0.0.1-8009-38 Id=460
  at jcifs.smb.SmbTree.treeConnect(SmbTree.java:128)
  at jcifs.smb.SmbTree.send(SmbTree.java:64)
  at jcifs.smb.SmbTree.treeDisconnect(SmbTree.java:168)
  at jcifs.smb.SmbSession.logoff(SmbSession.java:301)
  at jcifs.smb.SmbTransport.getSmbSession(SmbTransport.java:138)
  at jcifs.smb.SmbSession.logon(SmbSession.java:167)
  at jcifs.smb.SmbSession.logon(SmbSession.java:162)
```

...

```
"ajp-127.0.0.1-8009-38" Id=460 in BLOCKED on lock=jcifs.smb.SmbTransport@14cfb2c
  owned by ajp-127.0.0.1-8009-42 Id=486
  at jcifs.smb.SmbTree.treeConnect(SmbTree.java:130)
  at jcifs.smb.SmbSession.logon(SmbSession.java:169)
  at jcifs.smb.SmbSession.logon(SmbSession.java:162)
```

...

Beobachtbarkeit und Beobachtung – Thread-Dumps XI

- Konsequenz Deadlock

- Über 400 Threads im Stack

```
"ajp-127.0.0.1-8009-750" Id=2382 in BLOCKED on  
lock=jcifs.smb.SmbTransport@14cfb2c
```

```
owned by ajp-127.0.0.1-8009-42 Id=486
```

```
at jcifs.util.transport.Transport.connect(Transport.java:151)
```

```
at jcifs.smb.SmbTransport.connect(SmbTransport.java:287)
```

```
at jcifs.smb.SmbSession.getChallenge(SmbSession.java:146)
```

```
at jcifs.smb.SmbSession.getChallenge(SmbSession.java:140)
```

...

- Entstanden innerhalb weniger Minuten
- Keine Neuansmeldung mehr möglich
- Obwohl nur zwei Threads im Deadlock sind, bleiben sehr viele Threads dahinter hängen

Beobachtbarkeit und Beobachtung – Thread-Dumps XII

■ Lock-Contention

■ Viele Threads im Stack

```
"ajp-127.0.0.1-8009-XXX" Id=YYY in TIMED_WAITING on  
lock=com.mybiz.myapp.webapp.handler.FormHandler@be635d  
  at java.lang.Object.wait(Native Method)  
  at com.mybiz.myapp.webapp.handler.FormHandler.lock(Unknown Source)  
  at com.mybiz.myapp.webapp.UpdateAction.execute(Unknown Source)  
  at com.mybiz.myapp.webapp.ActionBase.execute(Unknown Source)  
  at  
org.apache.struts.action.RequestProcessor.processActionPerform(RequestProcessor.java:484)  
  at  
org.apache.struts.action.RequestProcessor.process(RequestProcessor.java:274)  
  at org.apache.struts.action.ActionServlet.process(ActionServlet.java:1482)  
  at org.apache.struts.action.ActionServlet.doPost(ActionServlet.java:525)  
...
```

■ Die Thread-IDs XXX und YYY wechseln langsam im Laufe der Zeit

Beobachtbarkeit und Beobachtung – Thread-Dumps XIII

- Dump-Auswertung
 - Meist sehen wir uns zunächst nur die obersten 5-10 Methoden des Stacks an
 - Wir wollen alle Threads gruppieren, deren Top-N Methoden-Stack gleich ist
 - Was machen die meisten Threads gerade?
 - Hier bietet sich ein Auswerteskript an
 - Beispiel: kpdtextplorer
 - Bei Ideen, was merkwürdig sein könnte, immer die Idee im vollen Dump überprüfen

Beobachtbarkeit und Beobachtung – Thread-Dumps XIV

- Phänomene, die sich meist gut durch Thread-Dumps verstehen lassen
 - Die Anwendung ist insgesamt langsam, obwohl die CPU-Auslastung gering ist
 - Dann wird meistens auf etwas gewartet
 - Remote Calls (Middleware, DB, Webservices)
 - Locks
 - Lieblingsformel
Durchsatz * Antwortzeit = Parallelität
 - Normale Anfragerate, erhöhte Antwortzeit => erhöhte Threadzahl

Beobachtbarkeit und Beobachtung – Thread-Dumps XV

- Phänomene, die sich meist gut durch Thread-Dumps verstehen lassen – Fortsetzung
 - Die CPU-Auslastung ist zu hoch
 - Was ist auf der CPU (Thread-Nummer)
 - Was macht der Thread?
 - Die Anwendung reagiert nicht mehr
 - Deadlock?
 - Alle Threads warten auf remote?

- Thread-Dumps – Vorschlag
 - Trauen Sie Sich Thread-Dumps in der Produktion zu machen
 - Natürlich erst im Test bzw. Staging
 - Versuchen Sie die Dumps zu verstehen
 - Insbesondere auch im Gut-Fall
 - Lassen Sie Sich nicht von der Größe abschrecken
 - Skript zum Zusammenfassen gleicher Top-N-Stacks
 - Erklären Sie Ihrem Betrieb, wie wichtig die Dumps sind
 - Und wie man sie herstellt

- Aufwärmrunde
- Typische Problemsituationen
- **Beobachtbarkeit und Beobachtung**
 - Thread-Dumps
 - Logging
 - JMX
 - Monitoring
- Zusammenfassung

Beobachtbarkeit und Beobachtung – Logging I

- Logging dient sehr verschiedenen Zwecken
 - Entwickler-Debugging
 - Anzeige von produktionsrelevanten Problemen und Informationen
 - Protokollierung von Vorgängen, Auditing
 - Performance-Messung
 - Erhebung SLA-relevanter Informationen
 - ...

Beobachtbarkeit und Beobachtung – Logging II

- Ein Log-Zweck ist nicht mit einem Log-Level gleichzusetzen!
 - Produktionsrelevante Meldung: INFO Rekonfiguration
 - Sicherheitsrelevante Meldung: WARN Drei aufeinanderfolgende Login-Fehlversuche
 - Performance-Protokollierung: INFO SID=12345 Account=0246 Method=Transfer Duration=110 Status=OK
 - Entwickler-Debugging: ERROR NullPointerException (Stack)

Beobachtbarkeit und Beobachtung – Logging III

- Log-Frameworks unterstützen Logger-Hierarchien (hier: Log4j)
 - **Plural!**
- Meist im Code nur eine Hierarchie
 - Ein Logger pro Klasse, der so heisst wie die Klasse
 - Das ist der Logger fürs Debuggen des Developers
 - Für andere Aspekte getrennte Logger verwenden!
 - Separate Log-Level
 - Separate Ausgabekanäle und Formate

Beobachtbarkeit und Beobachtung – Logging IV

- Diese anderen Logger ebenfalls hierarchisch benennen
 - Damit wird die Konfiguration einfacher
 - Falls der Code sinnvoll aufgebaut ist, evtl.
 - PERFORMANCE.Klassenname
 - AUDIT.Klassenname
 - ...
 - Überlegen Sie, welche Bedeutung Sie den Log-Levels für die anderen Logger geben wollen
 - Z.B. Detailgehalt von Audit-Logging

Beobachtbarkeit und Beobachtung – Logging V

- Grundregeln für die meisten Log-Zwecke
 - Logging ist nicht für Menschen (pro Meldung), sondern für Automaten
 - Zeilenorientiertes Logging
 - Strikte Formate, Konsistenz
 - Dokumentation
 - Viele Anwendungen, die neu in Produktion gehen laufen schon nach kurzer Zeit auf Debug-Log-Level
 - Solange die Applikation noch nicht gereift und stabil ist, verschiedene Log-Zwecke in verschiedene Kanäle ausgeben
 - Konsolidierung, erst wenn das Logging verstanden ist

Beobachtbarkeit und Beobachtung – Logging VI

- Insbesondere bei verteilten Anwendungen
 - Performance- und Erfolgsprotokollierung an den Schnittstellen
 - Ausgehend **und** eingehend
 - Methode, Verarbeitungsdauer, Erfolgscode, Vorgangs-ID
 - Zum Einstieg: Access-Log mit Log der Laufzeiten
 - NTP verwenden (Synchronisation der Uhren)
 - Vergeben Sie eindeutige Vorgangs-IDs
 - Reichen Sie diese durch Ihre Schnittstellen durch
 - Loggen Sie diese in den Meldungen

Beobachtbarkeit und Beobachtung – Logging VII

- Wie loggt man IDs auf gute Weise?
 - Log4J: Mapped Diagnostic Context
<http://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/MDC.html>
 - HashMap als ThreadLocal
 - `put(java.lang.String key, java.lang.Object o)`
 - `remove(java.lang.String key)`
 - Der Entwickler legt Information dann ab, wenn Sie entsteht und entfernt Sie am Ende der Verarbeitung
 - Die Info steht nicht im Log-Code
 - Welche Keys geloggt werden kann **zur Laufzeit** konfiguratativ entschieden werden

Beobachtbarkeit und Beobachtung – Logging VIII

- Das große Problem mit `java.util.logging`

- SimpleLogFormat

```
Nov 6, 2007 2:47:42 AM org.apache.catalina.startup.Catalina load  
INFO: Initialization processed in 4031 ms
```

- Erste Zeile

- Zeitstempel, Klasse, Methode

- Zweite Zeile

- Log-Level, Logmeldung

- Das ist völlig unbrauchbar

- Es gibt keinen wirklich gute LogFormatter für `j.u.l` !?

Beobachtbarkeit und Beobachtung – Logging IX

- **Ausgabekanäle**
 - Bitte nicht nach STDOUT/Console loggen
 - OK für Entwicklung
 - Ganz schlecht für Produktion
 - Es gibt Ausgabemeldungen, die wir nicht so einfach konfigurieren können, und evtl. nach STDOUT schicken müssen (GC-Log)
 - Bitte keine Remote-Logging-Experimente
 - Zumindest nicht für hohe Meldungsvolumina
 - Logging muss rock-solid sein
 - Insbesondere auch in Problemsituationen

Beobachtbarkeit und Beobachtung – Logging X

- Und was machen wir mit den Logfiles
 - In der ersten Produktionsphase ERROR-Meldungen auswerten, verstehen und reduzieren.
 - Ziel
 - Keine ERROR-Meldung ohne wichtigen Fehler
 - Keine FATAL-Meldung ohne unmittelbaren Reaktionsbedarf
 - Und umgekehrt
 - Performance-Logs auswerten
 - Langläufer: welche Methoden, wie häufig, zu welcher Zeit/Last
 - Gesamtperformance (Durchsatz, Antwortzeit), Fehlerrate

- Aufwärmrunde
- Typische Problemsituationen
- **Beobachtbarkeit und Beobachtung**
 - Thread-Dumps
 - Logging
 - **JMX**
 - Monitoring
- Zusammenfassung

Beobachtbarkeit und Beobachtung – JMX I

- Java Management Extensions (JMX)
 - JSR-3, Standardbestandteil ab Java 5, vorher etwa MX4J
- Sehr gute Möglichkeit interne Applikationszustände von außen pollbar zu machen
 - Größen
 - Pools etc.
 - Zähler
 - Anfragen, Dauern, Fehlerraten
 - Aber auch Konfigurationseinstellungen

Beobachtbarkeit und Beobachtung – JMX II

- Bereitstellung der Informationen geschieht über MBeans
 - Es ist nicht schwierig MBeans selbst bereitzustellen
 - MBeans können über eine Standard-Netzchnittstelle gepolt werden
 - Beispiel JConsole (rein interaktives Tool)
 - Auch andere Konnektoren denkbar, etwa JMXProxy in Tomcat (HTTP-Servlet) oder HTTP-Konnektor in MX4J
 - Standardisierung von JMX über HTTP work in progress

Beobachtbarkeit und Beobachtung – JMX III

- Bitte beachten
 - Das richtige Maß halten
 - Was wollen wir warum sehen?
 - Was fangen wir mit den Werten an?
 - Gibt es Schwellwerte für Gut/Schlecht?
 - Nach Möglichkeit skalare Werte verwenden
 - Die meisten externen Tools können noch keine zusammengesetzten Objekte pollen
 - Delta-Bildung bei Zählern ist Sache des pollenden Tools
 - Auch Operationen machbar: Reset, Aktivierung/Deaktivierung

Beobachtbarkeit und Beobachtung – JMX IV

- **Auswertung**
 - Mit dem Betrieb zusammen geeignete
 - Werkzeuge zum pollen und protokollieren
 - Auswerteverfahren
 - Schwellwerte
 - auswählen => Monitoring kommt gleich
 - Es reicht nicht, interaktive Beobachtung durchzuführen
 - Keine Historie, Single-User
 - Sehen Sie Sich auch einmal die MBeans der Container an
 - Tomcat: `http://localhost:8080/manager/jmxproxy?qry=*:*`

- Aufwärmrunde
- Typische Problemsituationen
- **Beobachtbarkeit und Beobachtung**
 - Thread-Dumps
 - Logging
 - JMX
 - **Monitoring**
- Zusammenfassung

Beobachtbarkeit und Beobachtung – Monitoring I

- **Mögliche Ziele von Monitoring**
 - **Störungen feststellen und melden**
 - Das ist das, was die meisten darunter verstehen
 - End-to-End-Monitoring (User-Simulation)
 - Überschreitung von Schwellwerten
 - Möglichst geringe Anzahl von Falschmeldungen
 - **Kontinuierliche Messung von Betriebszuständen um**
 - Komplexere Probleme analysieren zu können
 - Kapazitätsplanung zu ermöglichen

Beobachtbarkeit und Beobachtung – Monitoring II

- Die Betriebssicht
 - Den Betrieb interessiert meist nur die Betrachtung von Störungen
 - Der Betrieb greift auf die Entwicklung zurück, wenn es um die Analyse komplexerer Probleme geht
- Die Entwicklungssicht
 - Zur erfolgreichen Analyse von Betriebsproblemen benötigen Sie Auswertungen über das Zeitverhalten der Systemkomponenten

Beobachtbarkeit und Beobachtung – Monitoring III

- Symbiose
 - Symbiose bezeichnet das Zusammenleben von Organismen zweier Arten zum wechselseitigem Nutzen. [Wikipedia, verkürzt]
- Der Betrieb
 - Bekommt MBeans, Schwellwerte und Unterstützung bei Analyse schwieriger Probleme
- Die Entwicklung
 - Bekommt die Infrastruktur zur regelmäßigen Erfassung und Aufbereitung von Laufzeitzuständen

Beobachtbarkeit und Beobachtung – Monitoring IV

- Erklären Sie als Entwickler Ihrem Betrieb, warum Schwellwerte nicht ausreichen
 - Aber stellen Sie welche zur Verfügung
 - Beispiel: Stichwort Kapazitätsmanagement
- Helfen Sie dem Betrieb ggf. bei der Realisierung von Adaptern, um die MBeans abzufragen
 - Gängige Monitoring-Produkte sind leicht erweiterbar, was die Polling-Schnittstelle angeht

Beobachtbarkeit und Beobachtung – Monitoring V

- Versuchen Sie eine kontinuierliche Erfassung von interessanten Zuständen hinzubekommen, und zwar
 - Als Datenreihen
 - Aber auch visualisiert
 - Mit direktem Lese-Zugriff für die Entwicklung
- Beobachten Sie die Verläufe zu Beginn intensiver, um die Schwellwerte bzw. das Sizing nachzujustieren
- Sie werden einige Überraschungen erleben!

- Aufwärmrunde
- Typische Problemsituationen
- Beobachtbarkeit und Beobachtung
- Zusammenfassung

Zusammenfassung I

- Die häufigsten Produktionsprobleme sind
 - Performance-Probleme
 - Stabilitätsprobleme
- Für die Analyse benötigen wir
 - Beobachtungen
 - Und zwar auch aus der Zeit vor dem Problem
 - Was ist der Normalzustand?

Zusammenfassung II

- Beobachten können wir mittels
 - Thread-Dumps, pro-aktiv, aber auch post-mortem
 - Zweimal am Tag, alle Stunde, ...
 - Macht nur Sinn mit Auswerteskript
 - Logging
 - Aspektbezogen
 - Performance und Probleme
 - Monitoring
 - Auch von eigenen Laufzeitzuständen (etwa via JMX)
 - Macht mehr Sinn mit Visualisierung

Zusammenfassung – Lasttests

- Vor der Produktionsaufnahme sinnvoll
 - Lasttests inkl. dieser Beobachtungsmechanismen
 - Zuerst nur Komponentenlasttest, keine Integrationslasttests
 - Gut zu verstehende einfache Situation
 - Keine simulierten Echt-User mit Denkpausen
 - Unkontrollierbare Parallelitätsschwankungen
 - Uninteressante Dinge ggf. weglassen
 - Statischer Content oder SSL bei Webapps
 - Tools muss schlank sein um effizient Last zu erzeugen
 - Etwa JMeter
 - Validierung ob Test gültig war

Zusammenfassung – Troubleshooter und Troublemaker

- Troubleshooter und Troublemaker
 - Die meisten Probleme sind schon vor gravierenden Auswirkungen zu sehen
 - Wenn man den Normalzustand kennt
 - Schauen Sie hin und verkünden Sie, was Sie sehen
 - Proaktives Handeln ist häufig schwer durchzusetzen (Kosten/Nutzen)
 - Bauen Sie eine Brücke zur Betriebsabteilung
 - Wenigstens solange, bis Java in der Produktion gut verstanden ist

Abspann



- Bei Fragen

Rainer Jung
kippdata informationstechnologie GmbH
Bornheimer Str. 33a
53111 Bonn

rainer.jung@kippdata.de
www.kippdata.de, www.kippdata.de/tomcat
blogs.kippdata.de

Abspann

- kippdata informationstechnologie GmbH
 - Professioneller Open Source Support
 - Schwerpunkt Apache Tomcat und Apache httpd
 - Betriebskonzepte zu Hochlast und Ausfallsicherung
 - Healthchecks von produktiven JEE-Anwendungen
 - Lasttests und Sizing, Analyse von Produktionsproblemen
 - Workshops
- Besuchen Sie uns auf www.kippdata.de,
www.kippdata.de/tomcat und blogs.kippdata.de