Nachdenkzettel Logging

….....................................................................................................

Vorname, Name, Matrikelnummer

1. Kennzeichnen Sie in der Config die Stellen wo über das

- was geloggt wird🡪Klassen und Packages 🡪beliebig viele Logger🡪individuelle Levels können pro Package/Klasse festgelegt werden🡪spezielle Klassen, wo fehleranfällig🡪kann sich auch debug-Statements ausgeben lassen

- wieviel geloggt wird 🡪root ist default-Einstellung: geht von unten nach oben🡪wenn auf trace stellt, erhält alles

- wo geloggt wird🡪root,Konsole,file🡪hier File & Console; append=true🡪file vom vorherigen Lauf wird nicht gelöscht 🡪kann beim zweiten Mal unterscheiden 🡪kann Klasse, die geloggt werden soll, auch festlegen,in welchem Medium Geloggtes erscheint durch AppenderRef

- wie geloggt wird 🡪kann viel automatisch loggen

entschieden wird

<Configuration>

<Appenders>

<File name="A1" fileName="A1.log" append="false">

<PatternLayout pattern="%t %-5p %c{2} - %m%n"/>

</File>

<Console name="STDOUT" target="SYSTEM\_OUT">

<PatternLayout pattern="%d %-5p [%t] %C{2} (%F:%L) - %m%n"/>

</Console>

</Appenders>

<Loggers>

<!-- You may want to define class or package level per-logger rules -->

<Logger name="se2examples.core.businessLogic.VehicleManager" level="debug">

<AppenderRef ref="A1"/>

</Logger>

<Root level="debug">

<AppenderRef ref="STDOUT"/>

</Root>

</Loggers>

</Configuration>

Norm. Level info/warning🡪logging debug/trace beim Kunden im Betrieb nicht angeschaltet🡪bei Bug schaltet in „root“ um auf debug/trace

2. Geben Sie je ein Beispiel wann Sie den loglevel

- error 🡪 bei einem fatalen Fehler von besonderer Wichtigkeit: Datenbank-Verbindug z.B. fehlgeschlagen

- info 🡪um z.B. während der Runtime das Funktionieren bestimmter Anwendungen zu überprüfen (dass z.B. die Applikation gestartet ist)

- debug (🡪um z.B. während der Runtime das Funktionieren bestimmte Prozesse wie z.B. Anwendungen zu dokumentieren, wann was geschieht (dass z.B. dass package X aufgerufen wird etc.)(etwas Feingradiges🡪Fehlerbugs)

verwenden

3. Sie verwenden einen FileAppender für das Logging. Jetzt

soll Ihre Application im Datacenter laufen. Was machen Sie mit dem FileAppender?

Sicherheitslevel erhöhen:V erschlüsseln; alles was mit Security + Privacy zu tun hat 🡪log4j: gibt bestimmt Appender, die entsprechend verschlüsseln

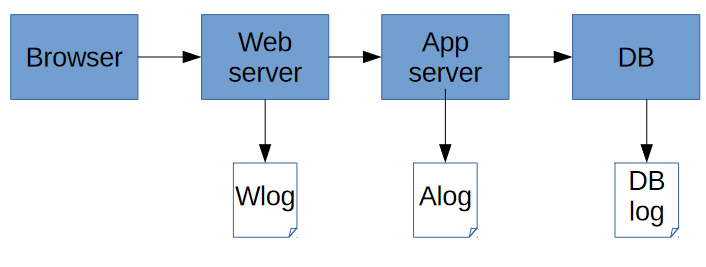
4. Macht logging Ihre Application langsamer? Was passiert wenn Sie log.debug(„foobar“); aufrufen? Wie sollte sich das Logging Subsystem verhalten?

Ja, für die normale Busines-Applikation:irrelevant/insignifikant; Real-Time App/Gaming: protokollieren Logger in RAM in norm. Files nichts, weil schnellste Mögichkeit 🡪höchste Geschwindigkeit

Wenn Prozess reinschreiben kann, dann können auch threads dort reinschreiben aber es wird unübersichtlich & kann nicht mehr filtern 🡪kann zum Fileappender sagen, dass wenn fertig, dann erst die anderen 🡪Zeit/Performance geringerer

5. Ein Request an Ihre Application durchläuft einen Proxy Server, dann einen Web Server, dann einen Application Server und dann die Datenbank. Auf jedem Server loggen Sie die Requests. Welches Problem tritt auf?

(Non-blocking-calls kann readsystemcall🡪Betriebssystem: lässt mit Fehler zurückkommen 🡪wenn ernst meint, würde schlafen legen aber mit Flags gesagt 🡪gibt Fehler sofort aus = non-Blocking)



Probleme

Gesplittete Log-files:3 verschiedene Logfiles

Muss anhand der Timestamps sortieren, ABER hat keine gemeinsame Uhrzeit🡪zusammenfügen passt nicht

Wo ist der Request: erst Webserver, dann Appserver, dann DB 🡪Welche Teile des Logs gehören zu welchem Request 🡪extrem schwierig nachzuvollziehen

Unterschiedliche Logs:

Problematik des Serversystems lässt sich lösen durch 🡪vers. Schnipsel zu einem Request

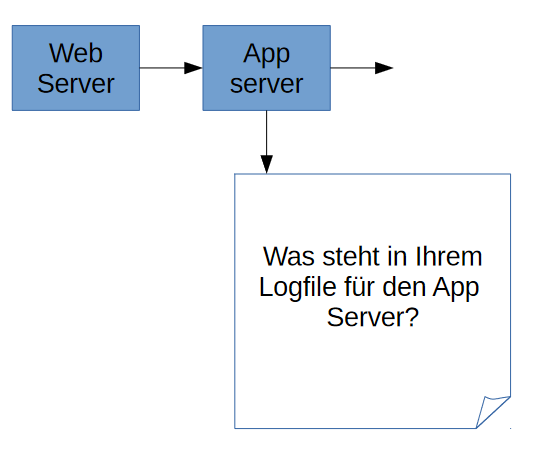
Trick in asynchronen Systemen: Starbucks=Name auf dem Becher 🡪weiß, welcher Becher zu welchem Kunden gehört

Corrolation ID (einzigartigen Schlüssel, Token) der durchläuft alle Server, welche ihn aufnehmen und weiterreichen

Lösung

Beim Ankommen einer Request 🡪verleiht Corrolation ID🡪nimmt jeder auf und wird weitergereicht, so dass es in alle Log-Statements eingeht

6. Was sollten Sie pro Komponente/Tier loggen?



App-Server:

Zeitstempel (Start und Ende von Aufrufen)

🡪schaut nach, ob alle Request gleich lang laufen 🡪wichtig in verteilten Systemen (Performance)

Status: Fehleranzeigen (ob Fehler auftreten), wie viele Requests, wie ist die CPU Auslastung

7. Aus Geschwindigkeitsgründen halten Sie teure DB-Connections auf Vorrat in einem Pool. Jeder Request vom Client braucht dann eine Connection. Der Pool hat die Methoden:DB Connection con = ConnectionPool.getConnection();ConnectionPool.freeConnection( DBConnection dbCon);Was loggen Sie in Ihrem App Server? Oder anders gefragt: Was wollen Sie beim Umgang mit dem Pool als Software-Architektin wissen?

Auslastung (ob Connections gespart werden können; wie viele Connections noch frei sind); prüft nach jeder Request, ob noch genügend Connections bereit 🡪 könnte reduzieren oder erhöhen 🡪Ressourcen sparen/verwalten bzw. Anwendung optimieren (kann überprüfen, welche Services z.B. wie viel Zeit halten (wie lange werden Ressourcen gehalten) und daher evtl. Performanceprobleme auftreten 🡪diese sieht man nur durch logging)

🡪LOGGING existentiell für die Wartbarkeit des Programms 🡪überall wo DB-Queries sofort logging: Vorgehen: Timestamp🡪dann Query mit Timestamp und bei Antwort Timestamp🡪kann überprüfen, ob zu lange dauert, wenn Differenz aus Timestamps bildet)