

LU03b - Logging

Übersicht

Standardmässig wird bei Java `System.out` für Logeinträge und `System.err` für Fehlermeldungen genutzt.

Dies ist natürlich störend, wenn die Applikation über die Konsole bedient wird.

Aber auch unabhängig davon, wird bei Applikation in der Regel ein Logging-Framework genutzt. Meldungen in einer Log-Datei können auch nach dem Beenden der Applikation oder nach einem Serverneustart gelesen werden.

- Java Util Logging <https://docs.oracle.com/javase/8/docs/api/java/util/logging/package-summary.html>
- Log4j <https://logging.apache.org/log4j/2.x/index.html>
- SLF4J (Simple Logging Facade for Java) <https://www.slf4j.org/>

SLF4J + Logback

Um SLF4J nutzen zu können, muss erstmal die Abhängigkeit in `build.gradle` ergänzt werden.

Als nächstes kann unter `src/main/resources/` die Logback-Konfigurationsdatei `logback.xml` erstellt werden. In dieser wird unter anderem definiert, welcher Log-Level genutzt wird. Ebenfalls kann dort ausgewählt werden, ob die Konsole (`ch.qos.logback.core.ConsoleAppender`) oder ein Log-File (`ch.qos.logback.core.rolling.RollingFileAppender`) benutzt werden soll.

Die Änderungen sind im Commit TODO enthalten.

LogLevel

Anschliessend können sämtliche Klassen, in denen Logging eingesetzt werden soll mit einer `Logger`-Objekt ausgestattet werden. Heutzutage wird oft `log` als Name gewählt, obwohl dieser von der Namenskonvention für Konstanten abweicht. Hauptsache die Benennung ist innerhalb des Projekts einheitlich.

| Vorher | Nachher |
|---|--|
| <pre>try { ... } catch (Exception e) { e.printStackTrace(); }</pre> | <pre>import org.slf4j.Logger; import org.slf4j.LoggerFactory; ... private static final Logger log = LoggerFactory.getLogger(<CLASS_NAME>.class); ... try { ... } catch (Exception e) { log.error("Error during ...", e); }</pre> |

| Level | Beschreibung | Beispiel |
|-------|--|--|
| DEBUG | Für Entwickler interessant, nicht für den Betrieb (interne Details, Diagnose, Ablaufverfolgung). | SQL-Statement, Aufruf-Parameter, Zwischenergebnisse |
| INFO | Normale, erwartete Ereignisse, die den regulären Ablauf dokumentieren. | Anwendung gestartet, Benutzer hat sich angemeldet, erfolgreicher DB-Aufruf |
| WARN | Unerwartete, aber tolerierbare Situation; Anwendung läuft weiter, sollte jedoch geprüft werden. | Ungültige Benutzereingabe, deprecated API verwendet, langsame Antwortzeit |
| ERROR | Schwerwiegender Fehler, der Funktionalität einschränkt oder nicht kompensiert werden kann. | Datenbank nicht erreichbar, IOException beim Schreiben einer Datei, Transaktion fehlgeschlagen |

Die Wahl des richtigen Loglevels ist essentiell, um bei grösseren Applikationen den Überblick zu behalten.

Grundsätzlich gilt: Eine Exception ist mindestens Stufe WARN, denn Exceptions sollten im Normalfall nicht auftauchen. Bei `.error(...)` und `'.warn(...)`' kann daher auch ein Stacktrace (Throwable) mitgegeben werden. Wenn man eine Exception „wrapped“ und weiterwirft (z.B. `NumberFormatException` in eine `IllegalArgumentException`), dann kann das Ereignis zwar geloggt werden, aber der StackTrace sollte nur einmal (beim endgültigen Fangen) ausgegeben werden.

Jedes catch-Statement sollte den Fehler loggen oder wrapped weiterwerfen. Überprüfen Sie all Ihre catch-Statements und bauen Sie entsprechende Log-Aufrufe ein.

From:
<https://wiki.bzz.ch/> - **BZZ - Modulwiki**

Permanent link:
<https://wiki.bzz.ch/de/modul/ffit/3-jahr/java/learningunits/lu03/b?rev=1756731879>

Last update: **2025/09/01 15:04**

