Altes SQL Logging (Bis Nuclos 4.5)

Um alle SQL Statements in alten Log4J-Konfiguration zu loggen, müssen folgende Zeilen in der log4j.properties hinzugefügt werden:

log4j

```
#SQL logging
log4j.logger.SQLLogger=DEBUG
```

Die Datei log4j.propierties ist im "conf" Verzeichnis des Servers zu finden, zusammen mit den Dateien jdbc.properties und server.properties. Dabei werden mit folgenden Einstellungen des log4j-Loggers alle relevanten Informationen ausgegeben:

```
log4j.appender.stdout.layout.ConversionPattern=%d %-5p %-9t [%c{3}] - %m%n
log4j.rootLogger=INFO, stdout, logfile
```

Die Ausgabe des SQL Statements **vor** der Ausführung erscheint dann in der Datei server.log im Verzeichnis logs und sieht z.B. folgendermaßen aus:

```
2013-06-27 14:44:09,567 DEBUG pool-2-thread-1 [SQLLogger] - SELECT t.INTID, t.DATCREATED, t.STRCREATED, t.
DATCHANGED, t.STRCHANGED, t.INTVERSION, t.INTNUCLETVERSION, t.STRPACKAGE, t.STRNAMESPACE, t.DESCRIPTION, t.NAME
FROM T_AD_APPLICATION t WHERE (1=1 AND t.INTID IN (?))
=[40013466]
```

SQL Performance Logging

Zusätzlich kann noch die Zeit der einzelnen SQL-Statmens gemessen werden. Dazu muss der SQLTimer aktiviert werden:

```
#SQL logging
log4j.logger.SQLTimer=DEBUG
```

Damit wird das SQL Statement **nach** der Ausführung zusammen mit der Dauer ausgegeben:

```
2013-06-27 14:44:09,599 DEBUG pool-2-thread-1 [SQLTimer] - SELECT t.INTID, t.DATCREATED, t.STRCREATED, t.
DATCHANGED, t.STRCHANGED, t.INTVERSION, t.INTNUCLETVERSION, t.STRPACKAGE, t.STRNAMESPACE, t.DESCRIPTION, t.NAME
FROM T_AD_APPLICATION t WHERE (1=1 AND t.INTID IN (?))
=[40013466]=(32 ms)
```

Hinweis: Es kann hilfreich sein, sowohl das Statement vor, als auch nach der Ausführung zu loggen. Insbesondere wenn Statements besonders lange dauern. Dafür werden einfach die beiden Zeilen kombiniert:

```
#SQL logging
log4j.logger.SQLLogger=DEBUG
log4j.logger.SQLTimer=DEBUG
```

SQL StackTrace Logging

Dies funktioniert ab Nuclos 3.14.9.

Es kann zusätzlich noch der StackTrace, sowohl vom Server, als auch vom Client, bis zur Ausführung des SQL-Statements geloggt werden. Das ist möglich für bestimmte SQL-Statements oder für alle SQL-Statements. Für bestimmte Statements kann in der ersten Methode der Klasse "DataSourceExecuter" ein String-Suchbegriff eingegeben werden (auch zur Laufzeit, wenn der Server im Debug-Modus gestertet wurde):

```
 private static boolean testForFullDebug(final ConnectionRunner<?> runner) {
     final String debugSQL = "FROM T_MD_STATE";
         ....
}
```

Außerdem lässt sich das StackTrace-Logging ganz allgemein für alle SQL-Statments in der log4j.propertier steuern:

```
#SQL logging
log4j.logger.SQLLogger=DEBUG
log4j.logger.ClientStackTrace=DEBUG
log4j.logger.ServerStackTrace=DEBUG
```

Mit der Option "DEBUG" werden nur StackTrace Elemente, welche mit "org.nuclos" beginnen geloggt. Wenn alle Element ausgegeben werden sollen, dann sollte die Option "TRACE" verwenden werden:

```
#SQL logging
log4j.logger.SQLLogger=DEBUG
log4j.logger.ClientStackTrace=TRACE
log4j.logger.ServerStackTrace=TRACE
```

Allgemeines zum Logging

Nuclos verwendete intern log4j Version 1.2.x für das Logging. Wenn Sie in Ihrer Installation die Datei `<nuclos_home>/conf/log4j.properties` anpa ssen, können Sie das Log Verhalten des Servers in weiten Bereichen konfigurieren. Auf diese Seite finden Sie verschiedene Beispielkonfigurationen, die

- bestimmte Log Events in eine separate Datei schreiben,
- bestimmte Log Events statt in einer Datei in der Datenbank loggen und
- die Ausführlichkeit des Loggings für verschiedene Log Bereiche (z.B. Java Klassen) anpassen.

Weitergehende Informationen zu log4j finden Sie z.B. unter

- http://www.benmccann.com/dev-blog/sample-log4j-properties-file/
- https://logging.apache.org/log4j/1.2/manual.html
- http://www.torsten-horn.de/techdocs/java-log4j.htm

Die hier vorgestellte Methode, unterschiedliche Log Dateien zu erzeugen und/oder das Logging in eine Datenbank umzuleiten, funktionieren auch in weiteren Fällen. Beispielsweise ist es möglich, das client lifecycle logging auf dem Server entsprechend zu konfigurieren. Das zu konfigurierende Log Handle ist hierbei `org.nuclos.server.web.ClientLifecycle`.

Separate Log Datei für Client Lifecycle Events

```
# standard log stuff
log4j.appender.logfile=org.apache.log4j.RollingFileAppender
log4j.appender.logfile.File=/home/tpasch2/nuclos4/logs/server.log
log4j.appender.logfile.MaxBackupIndex=1000
log4j.appender.logfile.MaxFileSize=1GB
log4j.appender.logfile.layout=org.apache.log4j.PatternLayout
log4j.appender.logfile.layout.ConversionPattern=%d %p [%c] - %m%n
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d %p [%c] - %m%n
log4j.rootLogger=INFO, stdout, logfile

# create another log file
log4j.appender.clientlifecycle=org.apache.log4j.RollingFileAppender
log4j.appender.clientlifecycle.File=/home/tpasch2/nuclos4/logs/clientlifecycle.log
log4j.appender.clientlifecycle.MaxBackupIndex=1000
log4j.appender.clientlifecycle.MaxFileSize=1GB
log4j.appender.clientlifecycle.layout=org.apache.log4j.PatternLayout
log4j.appender.clientlifecycle.layout.ConversionPattern=%d %p [%c] - %m%n

# use the file for a certain category
log4j.category.org.nuclos.server.web.ClientLifecycle=INFO, clientlifecycle
log4j.additivty.org.nuclos.server.web.ClientLifecycle=false
log4j.logger.org.nuclos.server.web.ClientLifecycle=INFO
```

JDBCAppender SQL Logging

Zusätzlich (oder alternativ) lassen sich die SELECT SQL Statements direkt in eine Datenbank schreiben. Dies erledigt das Apache log4j-Framework. Dazu braucht man eine Datenbank, die nicht notwendigerweise identisch mit der Nuclos-DB sein muss. Dort muss eine Tabelle angelegt werden, z.B. für PostgreSQL (bei Oracle sollte die Spalte "MESSAGE" ein CLOB sein):

POSTGRESQL

```
CREATE TABLE <SCHEMA>.LOGS(
        DATED TIMESTAMP NOT NULL PRIMARY KEY,
        LOGGER VARCHAR(127) NOT NULL,
        LEVEL VARCHAR(15) NOT NULL,
        MESSAGE VARCHAR(10485760) NOT NULL,
        THREAD VARCHAR(4000) NOT NULL,
        TIME INTEGER,
        PARAMS VARCHAR(4000)
);
```

ORACLE

```
CREATE TABLE <SCHEMA>.LOGS(
        "DATED" TIMESTAMP NOT NULL PRIMARY KEY,
        "LOGGER" VARCHAR(127) NOT NULL,
        "LEVEL" VARCHAR(15) NOT NULL,
        "MESSAGE" CLOB NOT NULL,
        "THREAD" VARCHAR(4000) NOT NULL,
        "TIME" NUMBER(11),
        "PARAMS" VARCHAR(4000)
);
```

Dazu sind folgende Parameter in die o.g. log4j.propierties hinzuzufügen und entsprechend anzupassen.

POSTGRESQL

```
#SQL logging
log4j.logger.SQLLogger=DEBUG, DB
log4j.logger.SQLTimer=DEBUG, DB

#JDBCAppender
log4j.appender.DB=org.apache.log4j.jdbc.JDBCAppender
log4j.appender.DB.driver=org.postgresql.Driver
log4j.appender.DB.URL=jdbc\:postgresql\://datenbankip\:5432/postgres
log4j.appender.DB.user=nuclos
log4j.appender.DB.password=nuclos
log4j.appender.DB.sql=INSERT INTO <SCHEMA>.LOGS VALUES('%d{yyyy-MM-dd HH:mm:ss.SSS}','%c','%p','%m','%t')
log4j.appender.DB.layout=org.apache.log4j.PatternLayout
```

ORACLE

```
#SQL logging
log4j.logger.SQLLogger=DEBUG, DB
log4j.logger.SQLTimer=DEBUG, DB

#JDBCAppender
log4j.appender.DB=org.apache.log4j.jdbc.JDBCAppender
log4j.appender.DB.driver=oracle.jdbc.driver.OracleDriver
log4j.appender.DB.URL=jdbc:oracle:thin:@datenbankip:1521:oracle
log4j.appender.DB.user=nuclos
log4j.appender.DB.password=nuclos
log4j.appender.DB.sql=INSERT INTO <SCHEMA>.LOGS ("DATED", "LOGGER", "LEVEL", "MESSAGE", "THREAD", "TIME",
"PARAMS") VALUES (to_timestamp('%d{yyyy-MM-dd HH:mm:ss.SSS}','YYYY-MM-DD HH24:MI:SS.FF3'),'%c','%p','%m','%t',
0,'')
log4j.appender.DB.layout=org.apache.log4j.PatternLayout
log4j.appender.DB.commit=true
```

Es ist zu empfehlen, den Log-Level möglichst hoch zu setzten (wie hier "WARN") damit die DB beim Loggin nicht überlastet wird. Durch entfernen von "stdout" oder "logfile" kann das Logging auf die DB selber beschränkt werden.

Auswertung der SQLTimer Werte aus der Datenbank

Mit "SQLTimer" werden die Angaben zur verbrauchten Zeit an die Message gehängt. Um diese einfacher auszuwerten, kann folgendes DB-Statements durchgeführt werden:

POSTGRESQL

```
UPDATE <SCHEMA>.LOGS SET TIME = CAST(SUBSTRING(
MESSAGE FROM POSITION('=(' IN MESSAGE)+2 FOR POSITION(' ms)' IN MESSAGE)-POSITION('=(' IN MESSAGE)-2) AS
INTEGER),
PARAMS = SUBSTRING(MESSAGE FROM POSITION('=[' IN MESSAGE)+2 FOR POSITION(']=' IN MESSAGE)-POSITION('=[' IN
MESSAGE)-2),
MESSAGE = RTRIM(SUBSTRING(MESSAGE FROM 0 FOR POSITION('=[' IN MESSAGE)-1))
WHERE LOGGER = 'SQLTimer'
```

Hinweis: Die Syntax der Statements ist PostgreSQL-Spezifisch. Für andere Datenbanken müssen die Statements angepasst werden.

Beispiel einer möglichen Auswertung:

```
DELETE FROM <SCHEMA>.LOGS WHERE LOGGER != 'SQLTimer'
SELECT SUM(TIME) AS SUM, COUNT(TIME) AS COUNT, MESSAGE FROM <SCHEMA>.LOGS GROUP BY MESSAGE ORDER BY SUM DESC
```

Ganz altes Standard SQL Logging (3.13.8 oder früher)

Um alle SQL Statements zu loggen, müssen folgende Zeilen in der log4j.properties hinzugefügt werden:

```
#SQL logging
log4j.logger.org.nuclos.server.dblayer=DEBUG
log4j.logger.org.nuclos.server.dal.processor.jdbc.impl.standard.StandardSqlDBAccess=DEBUG
```

Damit die neuen Parameter wirksam werden, muss der Server neu gestartet werden. Die Datei log4j.propierties ist im "conf" Verzeichnis des Servers zu finden, zusammen mit den Dateien jdbc.properties und server.properties.

Die Ausgabe erscheint dann in der Datei server.log im Verzeichnis logs und sieht z.B. folgendermaßen aus:

```
2013-02-11 10:02:28,208 DEBUG localhost-startStop-1 [impl.standard.StandardSqlDBAccess] - Query to execute:
 SELECT * FROM (SELECT entity.INTID  FROM T_MD_entity entity WHERE entity.STRentity = ?) WHERE ROWNUM <= 2
 [PSDH_Typ]
```

Wenn die Ausführung des SQL Statements länger als 0.1 Sekunden braucht, wird anschließend noch eine Zeitangabe (in Millisekunden) ausgegeben:

```
2013-02-11 10:05:49,069 DEBUG Thread-6  [impl.standard.StandardSqlDBAccess] - Query to execute:
 SELECT t.INTID, t.ID_ADDR, t.ID_MODULE, t.ID_PROJECT, t.ID_VARIANT FROM (SELECT t.INTID, t.ID_ADDR, t.
ID_MODULE, t.ID_PROJECT, t.ID_VARIANT , ROW_NUMBER() OVER ( ORDER BY t.INTID ASC) AS RN  FROM SV_CONS t WHERE
1=1) t WHERE RN > 6000 AND RN <= 7000
 []
2013-02-11 10:05:49,681 DEBUG Thread-6  [impl.standard.StandardSqlDBAccess] - Time consumed: 610 ms
```