Anyframe Logging Sql Plugin

Version 1.1.0

I.Introduction

런타임 시에 실제 실행되는 SQL을 로그로 남기기 위해서는 log4jdbc(http://code.google.com/p/log4jdbc/) [http://code.google.com/p/log4jdbc/]라는 오픈소스를 활용할 수 있다. logging-sql plugin은 log4jdbc를 사용하는데 필요한 라이브러리들을 포함하고 있다.

Installation

Command 창에서 다음과 같이 명령어를 입력하여 logging-sql plugin을 설치한다.

```
mvn anyframe:install -Dname=logging-sql
```

installed(mvn anyframe:installed) 혹은 jetty:run(mvn clean jetty:run) command를 이용하여 설치 결과를 확인해볼 수 있다.

Plugin Name	Version Range
core [http://dev.anyframejava.org/docs/anyframe/plugin/essential/core/1.6.0/reference/htmlsingle/core.html]	2.0.0 > * > 1.4.0

II.Logging SQL

Query 서비스를 통해 수행되는 SQL을 로그로 남기기 위해서는 log4jdbc라는 오픈소스를 활용할 수 있다. log4jdbc는 JDBC 호출이나 SQL문에 대해 로그를 남길 수 있는 JDBC Driver를 제공하고 있다. log4jdbc에 대한 보다 자세한 내용은 http://code.google.com/p/log4jdbc/ 를 참조한다. SQL문을 로그로 남기기 위한 library는 해당 플러그인을 설치하면 자동 설치된다. library를 수동으로 설치할 경우에는 Library 수동 설치 방법을 참조한다.

다음에서는 log4jdbc를 사용하여 SQL을 로그로 남기기 위한 절차를 3개의 STEP으로 나누어 설명하고자 한다.

- Step 1. DataSource 속성 정의
- Step 2. Query 서비스 속성 정의
- Step 3. Logger 정의

1.DataSource 속성 정의

1.1.JDBCDataSource를 사용할 경우

- 기본적으로 지원되는 JDBC Driver일 경우 DataSource 속성 정의시 driverClassName은 net.sf.log4jdbc.DriverSpy로 정의하고 url은 사용하고 있는 url 앞에 'jdbc:log4'를 추가한다. 다음은 일반적인 유형의 DataSource 속성 정의 파일인 context-datasource.xml 이다.

```
<bean id="commonDataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-
method="close">
    <property name="driverClassName" value="net.sf.log4jdbc.DriverSpy"/>
    <property name="url" value="jdbc:log4jdbc:hsqldb:file:/./DATABASE/db/sampledb"/>
    <property name="username" value="sa"/>
</bean>
```

[참고] DriverSpy에서 지원하는 기본 JDBC Driver 목록은 다음과 같다.

```
. oracle.jdbc.driver.OracleDriver
. com.sybase.jdbc2.jdbc.SybDriver
. net.sourceforge.jtds.jdbc.Driver
. com.microsoft.jdbc.sqlserver.SQLServerDriver
. weblogic.jdbc.sqlserver.SQLServerDriver
. com.informix.jdbc.IfxDriver
. org.apache.derby.jdbc.ClientDriver
. org.apache.derby.jdbc.EmbeddedDriver
. com.mysql.jdbc.Driver
. org.postgresql.Driver
. org.hsqldb.jdbcDriver
. org.h2.Driver
```

- **기본적으로 지원되지 않는 JDBC Driver일 경우**

net.sf.log4jdbc.DriverSpy에서 기본적으로 지원하는 JDBC Driver가 아닌 경우에는 앞서 언급한 기본 정의 방식과 동일하게 정의하되, System Property에 대한 추가 셋팅이 필요하다. Eclipse를 통해 작업하는 경우 Open Run Diaglog > Arguments 탭 > VM arguments 에 **log4jdbc.drivers** 를 속성키로, 실제 DB의 Driver 클래스명을 속성값으로 정의해주면 된다.

```
-Dlog4jdbc.drivers=com.ibm.db2.jcc.DB2Driver
```

1.2.JNDIDataSource를 사용할 경우

JNDIDataSource를 사용하는 경우 해당하는 WAS에 사용하고자 하는 DataSource에 대해 정의되어 있어야 한다.

- **WebLogic인 경우**

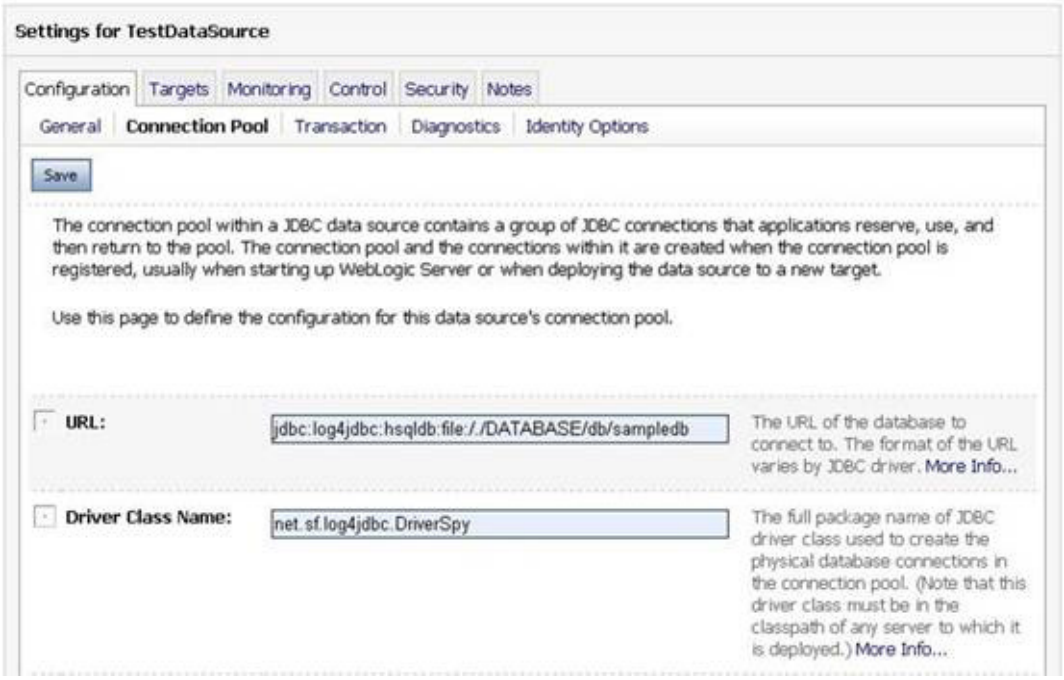
1. **DataSource 추가**

   WebLogic에 사용하고자 하는 DataSource를 정의한다. 이 때, JDBCDataSource 정의시와 마찬가지로 **driverClassName은 net.sf.log4jdbc.DriverSpy로 정의하고 url은 사용하고 있는 url 앞에 'jdbc:log4'를 추가** 하도록 한다. url 정의시 Step 3의 **JDBCDataSource를 사용할 경우** 를 참고하도록 한다.

참고

WebLogic Server에 net.sf.log4jdbc.DriverSpy를 이용한 DataSource를 추가하기 위해서는 WebLogic과 log4jdbc 사이에서 정상적인 호출이 발생할 수 있도록 해야 한다. 따라서, WebLogic Server 실행 전에 [Domain Server Home/lib]에 log4jdbcX-*.jar, slf4j-api-*.jar, slf4j-log4j12-*.jar를 복사해두어야 함에 유의하도록 한다.

다음은 WebLogic 9.2 Server에 추가한 TestDataSource에 대한 설정 내용이다.



2. **DataSource 속성 정의**

JNDIDataSource를 정의한 후에 해당 JNDIDataSource를 사용하기 위해서는 다음을 참조하여 context-datasource-jndi.xml 파일을 정의할 수 있다.

```
<bean id="commonDataSource" class="org.springframework.jndi.JndiObjectFactoryBean">
    <property name="jndiName" value="TestDataSource" />
    <property name="jndiTemplate" ref="jnditemplate" />
</bean>
<bean id="jnditemplate"
        class="org.springframework.jndi.JndiTemplate">
    <property name="environment">
        <props>
            <prop key="java.naming.factory.initial">
                weblogic.jndi.WLInitialContextFactory
            </prop>
            <prop key="java.naming.provider.url">
                t3://localhost:7001
            </prop>
        </props>
    </property>
</bean>
```

- **JEUS**인 경우

1. **DataSource 추가**

JEUS에 사용하고자 하는 JNDIDataSource를 추가한다. 다음은 사용하고자 하는 JNDIDataSource가 추가된 JEUS Server의 JEUSMain.xml 파일 내용의 일부이다.

```
1. XA 모드일 경우
<resource>
    <data-source>
        <database>
            <vendor>oracle</vendor>
            <export-name>OracleDS</export-name>
            <data-source-class-name>
                 oracle.jdbc.xa.client.OracleXADataSource
            </data-source-class-name>
            <data-source-type>XADataSource</data-source-type>
            <database-name>test2</database-name>
            <data-source-name>
                 oracle.jdbc.xa.client.OracleXADataSource
            </data-source-name>
            <port-number>1521</port-number>
            <server-name>server.ip</server-name>
            <user>anyframe</user>
            <password>anyframe</password>
            <driver-type>thin</driver-type>
            <connection-pool>
                <pooling>
                    <min>2</min>
                    <max>30</max>
                    <step>4</step>
                    <period>3600000</period>
                </pooling>
            </connection-pool>
        </database>
    </data-source>
</resource>
2. ConnectionPool 모드일 경우
<resource>
    <data-source>
        <database>
            <vendor>oracle</vendor>
            <export-name>OraclePoolDS</export-name>
            <data-source-class-name>
                 oracle.jdbc.pool.OracleConnectionPoolDataSource
            </data-source-class-name>
            <data-source-type>ConnectionPoolDataSource</data-source-type>
            <database-name>test2</database-name>
            <data-source-name>
                 oracle.jdbc.pool.OracleConnectionPoolDataSource
            </data-source-name>
            <port-number>1521</port-number>
            <server-name>server.ip</server-name>
            <user>anyframe</user>
            <password>anyframe</password>
            <driver-type>thin</driver-type>
            <connection-pool>
                <pooling>
                    <min>2</min>
                    <max>30</max>
                    <step>4</step>
                    <period>3600000</period>
                </pooling>
                <check-query>select sysdate from dual</check-query>
                <check-query-period>10000</check-query-period>
```

```
            </connection-pool>
        </database>
    </data-source>
</resource>
```

2. **DataSource Wrapper 정의 및 컴파일**

WAS가 JEUS인 경우 JNDIDataSource 추가시 DB URL을 별도로 정의할 수 없으므로 DriverSpy를 통한 SQL Logging을 수행할 수 없다. 이 경우에는 다음의 AnyframeDataSourceSpy와 같이 별도의 DataSource Wrapper 클래스를 정의하여 사용할 수 있다.

```
package net.sf.log4jdbc;

import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.SQLException;

import javax.sql.DataSource;

import net.sf.log4jdbc.ConnectionSpy;
import net.sf.log4jdbc.SpyLogDelegator;
import net.sf.log4jdbc.SpyLogFactory;
import net.sf.log4jdbc.RdbmsSpecifics;

public class AnyframeDataSourceSpy implements DataSource {

    private DataSource dataSource = null;

    static final SpyLogDelegator log = SpyLogFactory.getSpyLogDelegator();

    static RdbmsSpecifics defaultRdbmsSpecifics = new RdbmsSpecifics();

    public AnyframeDataSourceSpy() {
    }
    public Connection getConnection() throws SQLException {
        return getWrappedConnection(dataSource.getConnection());
    }
    public Connection getConnection(String username, String password)
            throws SQLException {
        return getWrappedConnection(dataSource
                .getConnection(username, password));
    }
    public PrintWriter getLogWriter() throws SQLException {
        return dataSource.getLogWriter();
    }
    public int getLoginTimeout() throws SQLException {
        return dataSource.getLoginTimeout();
    }
    public void setLogWriter(PrintWriter out) throws SQLException {
        dataSource.setLogWriter(out);
    }
    public void setLoginTimeout(int seconds) throws SQLException {
        dataSource.setLoginTimeout(seconds);
    }
    private Connection getWrappedConnection(Connection con) {
        if (log.isJdbcLoggingEnabled())
        {
            ConnectionSpy cspy = new ConnectionSpy(con);
            cspy.setRdbmsSpecifics(defaultRdbmsSpecifics);
            return cspy;
        }
```

```
        else
        {
            return con;
        }
    }
    public void setDataSource(DataSource dataSource) {

        this.dataSource = dataSource;
    }
}
```

3. **DataSource 속성 정의**

   앞서 정의한 DataSource Wrapper를 통해 해당 어플리케이션의 DataSource가 실행될 수 있도록 다음을 참조하여 context-datasource-wrapper.xml 파일을 정의한다.

```
<bean id="realDataSource"
        class="org.springframework.jndi.JndiObjectFactoryBean">
    <property name="jndiName" value="OraclePoolDS" />
    <property name="jndiTemplate" ref="jnditemplate" />
</bean>
<bean id="jnditemplate"
        class="org.springframework.jndi.JndiTemplate">
    <property name="environment">
        <props>
            <prop key="java.naming.factory.initial">
                jeus.jndi.JNSContextFactory
            </prop>
            <prop key="java.naming.provider.url">server.ip:9736</prop>
        </props>
    </property>
</bean>

<bean id="commonDataSource" class="net.sf.log4jdbc.AnyframeDataSourceSpy">
    <property name="dataSource" ref="realDataSource"/>
</bean>
```

2.Query 서비스 속성 정의

Query 서비스에서 참조하는 DataSource를 앞서 정의한 dataSource의 Bean Id인 'commonDataSource'로 정의 한다. 다음은 Query 서비스 속성을 정의한 샘플 context-query.xml 파일의 일부 내용이다.

```
<query:auto-config dbType="oracle" datasource-ref="commonDataSource" />
```

3.Logger 정의

lo4jdbc를 사용하여 로그를 남기기 위해서는 log4j.xml 파일 내에 다음을 참고하여, 필요한 Logger를 정의하도록 한다.

- **jdbc.sqlonly** : SQL문만을 로그로 남기며, PreparedStatement일 경우 관련된 argument 값으로 대체된 SQL문이 보여진다.
- **jdbc.sqltiming** : SQL문과 해당 SQL을 실행시키는데 수행된 시간 정보(milliseconds)를 포함한다.
- **jdbc.audit** : ResultSet을 제외한 모든 JDBC 호출 정보를 로그로 남긴다. 많은 양의 로그가 생성되므로 특별히 JDBC 문제를 추적해야 할 필요가 있는 경우를 제외하고는 사용을 권장하지 않는다.
- **jdbc.resultset** : ResultSet을 포함한 모든 JDBC 호출 정보를 로그로 남기므로 매우 방대한 양의 로그가 생성된다.

또한, 각 Logger에 대한 로그 레벨은 DEBUG, INFO, ERROR 중 하나를 선택할 수 있다.

- DEBUG - SQL이 실행된 클래스명과 Line 번호를 로그로 남긴다.
- INFO - SQL문을 로그로 남긴다.
- ERROR - SQL 실행 에러가 발생한 경우 stack trace 정보를 로그로 남긴다.

단, WebLogic Server에 정의된 JNDIDataSource를 사용할 경우 **WAS에 추가한 JNDIDataSource에서 log4j.xml 파일을 읽어낼 수 있어야 하므로, log4j-*.jar와 log4j.xml은 해당 WAS의 클래스패스 상([Domain Server Home/lib])에 놓여 있어야 함에 유의해야 한다.** WebLogic 9.2 기반에서 log4j.xml 파일의 경우 jar 파일 형태로 압축하여 [Domain Server Home/lib]에 위치시키거나, WebLogic Server 실행을 위한 자바 옵션에 다음과 같이 추가해 줄 수 있다.

```
-Dlog4j.configuration=file:///path../log4j.xml
```

Anyframe Monitoring Tool을 통해 모니터링하는 경우

해당 웹어플리케이션에 대해 Anyframe Monitoring Tool을 통해 모니터링하는 경우, Monitoring Tool의 Logging 처리 로직으로 인해 WAS 로드시 ClassCastException이 발생하므로 -Dlog4j.defaultInitOverride=true 옵션을 추가해 주어야 한다. 이러한 경우 Monitoring Agent에 대한 로그는 확인할 수 없게 된다.

부록 A. Library 수동 설치 방법

A.1.Log4jdbc 라이브러리 다운로드

다음을 참고하여, 필요한 log4jdbcX-*.jar 파일을 다운로드한 후, **[Anyframe 설치 폴더] 또는 [Web Root/WEB-INF/lib 폴더] 내에 복사** 한다. WebLogic JNDIDataSource를 사용할 경우에는 **WAS 시작시 로드될 수 있도록 해당 WAS의 클래스패스 상에 복사한다.**

파일명	설명	다운로드
log4jdbc3-1.2alpha2.jar	If you are using JDK 1.4 or 1.5, you should use the JDBC 3 version of log4jdbc.	Download [http://log4jdbc.googlecode.com/files/log4jdbc3-1.2alpha2.jar]

A.2.Simple Logging Facade for Java 라이브러리 다운로드

log4jdbc는 Simple Logging Facade for Java (SLF4J) [http://slf4j.org/] 를 이용하여 어플리케이션에서 사용하는 Logging 서비스와 유연하게 연동할 수 있도록 하고 있다. 따라서 다음 파일들을 http://www.slf4j.org/dist [http://slf4j.org/dist]로부터 다운로드한 후, **[Anyframe 설치 폴더] 또는 [Web Root/WEB-INF/lib 폴더] 내에 복사** 한다. WebLogic JNDIDataSource를 사용할 경우에는 **WAS 시작시 로드될 수 있도록 해당 WAS의 클래스패스 상에 복사한다.**

파일명	설명	
slf4j-api-*.jar	log4jdbc와 logging 서비스 연동을 위한 API 제공	
slf4j-log4j12-*.jar	log4jdbc와 Log4j 기반의 Logging 서비스 연동을 위한 구현 라이브러리 제공	