Anyframe Logback Plugin

Version 1.0.0

I.Introduction

Logback plugin은 Logback framework와 Anyframe간의 연계방법을 가이드 하기 위한 샘플코드와 참조 라이브러리와 설정파일로 구성되어 있다.

Installation

Command 창에서 다음과 같이 명령어를 입력하여 Logback plugin을 설치한다.

```
mvn anyframe:install -Dname=logback
```

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

Dependent Plugins

Plugin Name	Version Range
core [http://dev.anyframejava.org/docs/anyframe/plugin/essential/core/1.0.4/reference/htmlsingle/core.html]	2.0.0 > *

주의 사항

- Logback 플러그인을 설치한 후 다른 플러그인을 설치하는 경우 slf4j-log4j 라이브러리와 logback 라이브러리가 충돌하여 SLF4J: Class path contains multiple SLF4J bindings 라는 경고 메시지가 나타날 수 있으나 어플리케이션 실행에는 영향을 미치지 않는다. 해당 경고 메시지를 제거하기 위해서는 slf4j-log4j 라이브러리를 수동으로 제거하면 문제를 해결할 수 있다. **하지만, Logback 플러그인 설치 시 slf4j-log4j 라이브러리 참조를 자동으로 제거하므로, multiple binding 문제를 피하기 위해서는 Logback 플러그인을 가장 마지막에 설치하는 것을 권장한다.**
- Logback을 사용하는 경우 core 플러그인을 제외한 다른 플러그인 설치시 log4j.xml에 자동으로 추가되는 logger를 직접 logback.xml에 등록해주어야한다.
- Logback 플러그인에서는 LoggingAspect를 이용하여 로그정보를 출력한다. LoggingAspect의 경우 로그정보를 debug 레벨로 출력하는데 기본적으로 Logback에서 사용하는 logger의 레벨은 INFO로 설정되어 있으므로 로그정보를 확인하기 위해서는 해당 Logger의 level을 debug로 설정하여야 한다.
- Logback 플러그인에서는 해당 로그파일이 생성되는 경로를 상대경로로 설정하였다. 상대경로로 설정한 경우 해당 로그는 생성되는 경로는 어플리케이션을 실행한 곳을 기준으로 상대경로로 생성이 된다. 예를 들면 eclipse 와 eclipse가 내장된 tomcat을 이용하여 플러그인을 실행한 경우에는 eclipse가 설치된 경로이고 특정 WAS에 배포하여 사용하는 경우에는 해당 WAS에 실행되는 상대경로를 확인하여야 한다.

II.Logback

Logback은 널리 알려진 Java Logging Framework인 Log4j의 후속 프로젝트이다. Logback은 log4j에 비해서 속도나 메모리 사용면에서 개선되었고, slf4j 인터페이스를 구현함으로써 다른 Logging framework 와의 변경을 손쉽게 할 수 있다. 또한 Logback은 Filter, SiftAppender 등 추가적인 기능을 지원한다. 이 장에서는 Logback에서 새롭게 제공하는 기능들에 대해서 알아보도록 한다.

1.Features

Logback에서 새롭게 제공하는 기능은 다음과 같다. logback.xml과 같은 설정파일만을 수정함으로써 간단하게 기능을 적용할 수 있다.

- 설정파일의 Dynamic Reloading 지원
- 설정파일의 조건부 처리 기능
- 로그파일에 대한 자동압축, 자동 삭제 기능 제공
- 런타임에 설정한 값에 따라 로그를 분리하여 처리할 수 있는 SiftingAppender 제공
- groovy 언어로 설정파일 작성 기능(logback.groovy)
- FileAppender 사용 시 다수의 JVM이 동시에 하나의 파일에 로그를 남길 수 있는 prudent mode를 지원
- 다양한 조건에 따른 로깅처리 여부를 결정할 수 있는 Filter 제공

위 기능들 중 중요 기능에 대해서 좀 더 알아보도록 한다.

1.1.Dynamic Reloading

Logback은 자체적으로 설정 파일을 reloading 할 수 있는 기능을 제공한다. 설정파일의 <configuration>의 scan 속성과 scanPeriod 속성을 설정함으로써 Dynamic Reoloading을 설정할 수 있다. scanPeriod 속성을 명시하지 않는 경우 디폴트 값은 1분이 적용된다. scanPeriod에 적용할 수 있는 단위는 milliseconds, seconds, minutes, hours가 있고 단위를 생략하는 경우 경우 디폴트 단위는 milliseconds가 적용이 된다.

```
<configuration scan="true" scanPeriod="30 seconds">
.....
</configuration>
```

주의할 점은 Logback은 로그 수행요청이 있을 경우에 설정파일에 명시한 시간간격을 체크 하여 설정파일을 reload 하므로, 로그 요청이 수행되어야 하고 로그가 호출되는 약간의 시간을 추가로 필요로 한다는 점이다.

1.2.Conditional Processing of Configuration Files

Logback은 하나의 설정파일을 다양한 조건에 따라 처리할 수 있는 <if>, <then>, <else> 태그를 제공한다. 해당 조건 태그들은 중첩해서 사용가능하고 설정파일의 어느위치에도 올 수 있다. 해당 기능을 사용하기 위해서 <if> 태그의 condition 속성에 expression을 설정해야하는데 expression을 작성하는 방식은 Java언어를 이용하는 방법과, groovy언어를 이용하는 방법이 존재한다. Java언어를 사용하기 위해서는 Janino 라이브러리(2.6 버전이상의 경우 Janino, commons-compiler), groovy언어를 사용하기 위해서는 groovy 런타임이 추가 라이브러리로 필요하다.

```
<if condition='java or groovy conditional expression'>
    <then>
        ...
    </then>
    <else>
     ...
```

```
    </else>
</if>
```

1.3.Archiving

Logback에서는 RollingFileAppender를 사용하는 경우 로깅이 완료된 파일에 대해서 자동으로 압축하는 기능을 지원한다. 압축기능을 설정하기 위해서는 <fileNamePattern>에 확장자를 zip, gz로 설정하면 압축기능을 설정할 수 있다. 또한 TimeBasedRollingPolicy를 사용하는 경우 maxhistory 설정을 통해서 보관할 로그파일의 개수를 설정할 수 있고 오래된 파일을 자동으로 삭제해주는 기능을 지원한다.

```
<appender name="file" class="ch.qos.logback.core.rolling.RollingFileAppender">
    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

        <fileNamePattern> someFileName.log.zip (or gz) </fileNamePattern>

          <maxHistory>90</maxHistory>
    </rollingPolicy>
    ....
</appender>
```

fileNamePattern을 지정할 때 유의해야 하는 점은 날짜 형식을 파일명 혹은 폴더명으로 사용할 때 해당 OS지원하지 않는 특수문자가 포함되어서는 안된다는 점이다.

1.4.SiftingAppender

Logback에서는 특정 런타임 기준에 따라 logger를 분리하여 로깅을 수행할 수 있는 SiftingAppender를 제공한다. 디폴트로 MDC(Mapped Diagnostic Context)에 존재하는 키 값을 이용하여 로그조건을 분리하는 기능을 제공한다. 로그를 분리하여 남기기 위해서는 <sift> 태그에 실제 출력을 할 appender를 지정한다. 로그를 분리하기 위해서는 appender 객체가 구분되어서 생성되어야 하고 해당 appender의 출력위치도 구분이 되어야 한다. 해당 appender 객체와 파일이 구분되게 하기 위해 appender의 경우 name 속성에 서로 다른 식별자를 사용하여야 하기 때문에 해당 name속성은 <discriminator> 태그에 설정한 키 값을 이용하여 이름을 설정하고 출력할 파일 경로도 해당 키값을 이용하여 분리해준다. 다음은 SiftingAppender를 의 key를 이용하여 설정하는 예제이다. MDC에 관한 추가적인 정보는 MDC를 참조한다.

```
<appender name="sift" class="ch.qos.logback.classic.sift.SiftingAppender">
    <discriminator>
        <key>mdcKey</key>
        <defaultValue>defaultValue</defaultValue>
    </discriminator>
    <sift>
        <appender name="${mdcKey}_appender"  class="ch.qos.logback.core.FileAppender">
            <file>${mdcKey}.log</file>
            ...
        </appender>
        ...
    </sift>
</appender>
```

III.Logback Integration

본 매뉴얼에서는 property와 MDC(mapped diagnostic context)를 이용하여 사용자가 설정한 조건에 따라 로그를 남기를 샘플 소스 코드를 통하여 Logback과 Anyframe의 연계 방법을 설명한다.

2.Dependencies

Logback을 사용하기 위해서는 logback-core와 logback-classic 라이브러리가 필수적으로 필요하다. 또한 Logback은 sf4j를 구현하였기 때문에 slf4j-api 라이브러리를 필요로 한다. 다음은 필수라이브러리에 대한 추가 설명이다.

- slf4j-api : slf4j 표준 인터페이스 구현체로, core 플러그인 설치시 디폴트로 추가된다.
- logback-core : logback-classic과 의존 모듈로 appender, layout 등이 구현되어 있는 모듈이다.
- logback-classic : Logback의 logger와 SiftAppender등이 구현되어 있는 모듈이다.

Logback 플러그인 에서는 <if> 태그를 제공하여 하나의 설정파일을 다양한 조건에 의해 변경할 수 있는 방법을 제공하는데 Java를 이용한 태그를 사용한 방식과 groovy언어를 사용한 방식 두 가지를 지원한다. Logback 플러그인에서는 Java를 이용한 방식을 사용하였는데 해당 기능을 이용하기 위해서는 janino와 commons-complier 라이브러리가 추가된다.

core 모듈이 설치되면 sl4j-log4j*.jar가 참조 라이브러리로 지정이 된다. sl4j-log4j*.jar와 logback-classic이 존재하게 되면, slf4j-api 구현체가 중복으로 존재하게 되므로, Logback 플러그인 설치시 slf4j-log4j*.jar에 대한 dependency를 제거 한다.

3.logback.xml

다음은 60초 간격으로 해당 설정 파일의 변경여부를 체크하고 변경된 설정파일을 적용하도록 한 설정이다.

```
<configuration scan="true" scanPeriod="60 seconds">
...
</configuration>
```

다음은 loggingCondition이라는 property를 이용하여 설정된 값에 따라 appender 설정을 변경하는 설정이다. 해당 appender에서는 로그 메시지와 함께 MDC에 있는 사용자 아이디, IP, 요청 URL 정보를 출력한다.

```
<property name="loggingCondition" value="ID"/>
    <if condition='property("loggingCondition").equals("ID")'>
        <then>
            <!-- separate logging using user ID-->
            <appender name="file" class="ch.qos.logback.classic.sift.SiftingAppender">
                <discriminator>
                    <key>userId</key>
                    <defaultValue>guest</defaultValue>
                </discriminator>
                <sift>
                    <appender name="id-${userId}"
class="ch.qos.logback.core.FileAppender">
                        <file>userId/${userId}.log</file>
                        <encoder>
                            <pattern>[%-5level] %d{yyyy-MM-dd HH:mm:ss} %logger %n%msg%n
userId:%X{userId},clientIP:%X{remoteHost},requestURL:%X{requestURL}%n</pattern>
                        </encoder>
                    </appender>
                </sift>
            </appender>
        </then>
        <else>
            <if condition='property("loggingCondition").equals("IP")'>
                <then>
                <!-- separate logging using client IP-->
                <appender name="file"
class="ch.qos.logback.classic.sift.SiftingAppender">
                    <discriminator>
                        <key>remoteHost</key>
                        <defaultValue>unknownHost</defaultValue>
                    </discriminator>
                    <sift>
                        <appender name="ip-${remoteHost}"
class="ch.qos.logback.core.FileAppender">
                            <file>ip/${remoteHost}.log</file>
                            <encoder>
                                <pattern>[%-5level] %d{yyyy-MM-dd HH:mm:ss} %logger %n
%msg%n userId:%X{userId},clientIP:%X{remoteHost},requestURL:%X{requestURL}%n</pattern>
                            </encoder>
                        </appender>
                    </sift>
                </appender>
            </then>
```

```
                <else>
                    <!-- default daily rolling -->
                    <appender name="file"
 class="ch.qos.logback.core.rolling.RollingFileAppender">
                        <file>currentLog.log</file>
                        <rollingPolicy
 class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
                            <fileNamePattern>logs/logFile.%d{yyyy-MM-dd HH.mm.ss}.log.zip</
fileNamePattern>
                            <maxHistory>30</maxHistory>
                        </rollingPolicy>
                        <encoder>
                            <pattern>[%-5level] %d{yyyy-MM-dd HH:mm:ss} %logger %n%msg%n
 userId:%X{userId},clientIP:%X{remoteHost},requestURL:%X{requestURL}%n</pattern>
                        </encoder>
                    </appender>
                </else>
            </if>
        </else>
    </if>
```

4.MDC(Mapped Diagnostic Context)

Request 또는 사용자별로 특정한 정보를 로그로 남기기 위해서 사용할 수 있는것이 MDC이다. MDC는 Mapped Diagnostic Context의 약자로 쓰레드 별로 존재하는 Map과 유사한 객체이다. 하나의 request를 처리하기 위해서 thread를 사용하므로 이 MDC를 이용함으로써 사용자정보를 남길 수 있고 Logback은 MDC를 이용한 다양한 기능을 제공한다. 일반적인 web 환경의 경우 ServeltFilter를 이용하여 해당 MDC의 정보를 관리하면 좀더 손쉽게 해당정보를 관리할 수 있다.

다음은 ServletFilter 를 이용하여 MDC의 정보를 기록하는 MDCServletFilter 클래스이다. 요청처리전 requst와 session 정보를 이용하여 필요한 정보를 MDC에 넣고 요청이 처리가 완료되면 MDC의 값을 clear 한다.

```
public class MDCServletFilter implements Filter{

    public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain) throws IOException, ServletException {
        insertIntoMDC(request);
        try{
            chain.doFilter(request, response);
        }finally{
            clearMDC();
        }
    }

    private void insertIntoMDC(ServletRequest request){
        MDC.put(REMOTE_HOST_MDC_KEY, request.getRemoteHost());

        if (request instanceof HttpServletRequest){
            HttpServletRequest httpServletRequest = (HttpServletRequest) request;

            StringBuffer requestURL = httpServletRequest.getRequestURL();
            if (requestURL != null) {
                MDC.put(REQUEST_URL_MDC_KEY, requestURL.toString());
   }
            HttpSession session = httpServletRequest.getSession();

            if (session != null && session.getAttribute(USER_ID_MDC_KEY) != null){
                MDC.put(USER_ID_MDC_KEY, (String)session.getAttribute(USER_ID_MDC_KEY));
            }

        }
    }
...
```

주의할 점은 필터를 이용하여 MDC에 기록한 값은 처리가 request 처리가 끝난 후 반드시 clear를 해줘야 한다는 점이다.

MDCServletFilter는 플러그인 설치시 web.xml에 등록이 된다.

```
<filter>
    <filter-name>mdcServletFilter</filter-name>
    <filter-class>....logback.filter.MDCServletFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>mdcServletFilter</filter-name>
    <url-pattern>*.do</url-pattern>
</filter-mapping>
```

IV.References

- 참고자료
  - Logback [http://logback.qos.ch/]
  - MDC [http://logback.qos.ch/manual/mdc.html]
  - Groovy Configuration [http://logback.qos.ch/manual/groovy.html]