<?xml version="1.0" encoding="UTF-8"?>  
<configuration>  
 <property name="LOGS\_ABSOLUTE\_PATH" value="./logs" />  
 <property name="LOG\_DAYS" value="20" />  
 <property name="LOG\_FILESIZE\_LOG" value="100M" />  
  
 <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">  
 <filter class="ch.qos.logback.classic.filter.ThresholdFilter"> <!--지정한 level 이상만 보여줌-->  
 <level>INFO</level>  
 </filter>  
 <!--<filter class="ch.qos.logback.classic.filter.LevelFilter"> 지정한 level만 보여줌  
 <level>INFO</level>  
 <onMatch>ACCEPT</onMatch>  
 <onMismatch>DENY</onMismatch>  
 </filter>-->  
 <file>${LOGS\_ABSOLUTE\_PATH}/communicator.log</file>  
 <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">  
 <pattern>[%d{yyyy/MM/dd HH:mm:ss.SSS}] [%5p] [%3t] %C{1} %M => %m%n</pattern>  
 </encoder>  
 <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">  
 <timeBasedFileNamingAndTriggeringPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">  
 <!-- 파일당 최고 용량 10MB -->  
 <maxFileSize>10MB</maxFileSize>  
 </timeBasedFileNamingAndTriggeringPolicy>  
 <!-- 일자별 로그파일 최대 보관주기(일단위) 만약 해당 설정일 이상된 파일은 자동으로 제거-->  
 <maxHistory>20</maxHistory>  
 <fileNamePattern>${LOGS\_ABSOLUTE\_PATH}/communicator.%d{yyyy-MM-dd}\_%i.log</fileNamePattern>  
 </rollingPolicy>  
 </appender>  
  
 <appender name="FILE\_RS\_MSG" class="ch.qos.logback.core.rolling.RollingFileAppender">  
 <file>${LOGS\_ABSOLUTE\_PATH}/rs/msg/rs\_msg.log</file>  
 <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">  
 <pattern>[%d{yyyy/MM/dd HH:mm:ss.SSS}] [%5p] [%3t] %C{1} %M => %m%n</pattern>  
 </encoder>  
 <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">  
 <timeBasedFileNamingAndTriggeringPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">  
 <!-- 파일당 최고 용량 10MB -->  
 <maxFileSize>10MB</maxFileSize>  
 </timeBasedFileNamingAndTriggeringPolicy>  
 <!-- 일자별 로그파일 최대 보관주기(일단위) 만약 해당 설정일 이상된 파일은 자동으로 제거-->  
 <maxHistory>20</maxHistory>  
 <fileNamePattern>${LOGS\_ABSOLUTE\_PATH}/rs/msg/rs\_msg.%d{yyyy-MM-dd}\_%i.log</fileNamePattern>  
 </rollingPolicy>  
 </appender>  
  
 <appender name="FILE\_RS\_REPORT" class="ch.qos.logback.core.rolling.RollingFileAppender">  
 <file>${LOGS\_ABSOLUTE\_PATH}/rs/report/rs\_report.log</file>  
 <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">  
 <pattern>[%d{yyyy/MM/dd HH:mm:ss.SSS}] [%5p] [%3t] %C{1} %M => %m%n</pattern>  
 </encoder>  
 <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">  
 <timeBasedFileNamingAndTriggeringPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">  
 <!-- 파일당 최고 용량 10MB -->  
 <maxFileSize>10MB</maxFileSize>  
 </timeBasedFileNamingAndTriggeringPolicy>  
 <!-- 일자별 로그파일 최대 보관주기(일단위) 만약 해당 설정일 이상된 파일은 자동으로 제거-->  
 <maxHistory>20</maxHistory>  
 <fileNamePattern>${LOGS\_ABSOLUTE\_PATH}/rs/report/rs\_report.%d{yyyy-MM-dd}\_%i.log</fileNamePattern>  
 </rollingPolicy>  
 </appender>  
  
 <appender name="FILE\_IBK\_RECEIVER" class="ch.qos.logback.core.rolling.RollingFileAppender">  
 <file>${LOGS\_ABSOLUTE\_PATH}/ibk/receiver/receiver.log</file>  
 <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">  
 <pattern>[%d{yyyy/MM/dd HH:mm:ss.SSS}] [%5p] [%3t] %C{1} %M => %m%n</pattern>  
 </encoder>  
 <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">  
 <timeBasedFileNamingAndTriggeringPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">  
 <!-- 파일당 최고 용량 10MB -->  
 <maxFileSize>10MB</maxFileSize>  
 </timeBasedFileNamingAndTriggeringPolicy>  
 <!-- 일자별 로그파일 최대 보관주기(일단위) 만약 해당 설정일 이상된 파일은 자동으로 제거-->  
 <maxHistory>20</maxHistory>  
 <fileNamePattern>${LOGS\_ABSOLUTE\_PATH}/ibk/receiver/receiver.%d{yyyy-MM-dd}\_%i.log</fileNamePattern>  
 </rollingPolicy>  
 </appender>  
  
 <appender name="FILE\_IBK\_SENDER" class="ch.qos.logback.core.rolling.RollingFileAppender">  
 <file>${LOGS\_ABSOLUTE\_PATH}/ibk/sender/sender.log</file>  
 <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">  
 <pattern>[%d{yyyy/MM/dd HH:mm:ss.SSS}] [%5p] [%3t] %C{1} %M => %m%n</pattern>  
 </encoder>  
 <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">  
 <timeBasedFileNamingAndTriggeringPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">  
 <!-- 파일당 최고 용량 10MB -->  
 <maxFileSize>10MB</maxFileSize>  
 </timeBasedFileNamingAndTriggeringPolicy>  
 <!-- 일자별 로그파일 최대 보관주기(일단위) 만약 해당 설정일 이상된 파일은 자동으로 제거-->  
 <maxHistory>20</maxHistory>  
 <fileNamePattern>${LOGS\_ABSOLUTE\_PATH}/ibk/sender/sender.%d{yyyy-MM-dd}\_%i.log</fileNamePattern>  
 </rollingPolicy>  
 </appender>  
  
 <appender name="FILE\_IBK\_DECODER" class="ch.qos.logback.core.rolling.RollingFileAppender">  
 <file>${LOGS\_ABSOLUTE\_PATH}/ibk/decoder/decoder.log</file>  
 <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">  
 <pattern>[%d{yyyy/MM/dd HH:mm:ss.SSS}] [%5p] [%3t] %C{1} %M => %m%n</pattern>  
 </encoder>  
 <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">  
 <timeBasedFileNamingAndTriggeringPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">  
 <!-- 파일당 최고 용량 10MB -->  
 <maxFileSize>10MB</maxFileSize>  
 </timeBasedFileNamingAndTriggeringPolicy>  
 <!-- 일자별 로그파일 최대 보관주기(일단위) 만약 해당 설정일 이상된 파일은 자동으로 제거-->  
 <maxHistory>20</maxHistory>  
 <fileNamePattern>${LOGS\_ABSOLUTE\_PATH}/ibk/decoder/decoder.%d{yyyy-MM-dd}\_%i.log</fileNamePattern>  
 </rollingPolicy>  
 </appender>  
  
 <appender name="FILE\_RS\_DECODER" class="ch.qos.logback.core.rolling.RollingFileAppender">  
 <file>${LOGS\_ABSOLUTE\_PATH}/rs/decoder/decoder.log</file>  
 <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">  
 <pattern>[%d{yyyy/MM/dd HH:mm:ss.SSS}] [%5p] [%3t] %C{1} %M => %m%n</pattern>  
 </encoder>  
 <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">  
 <timeBasedFileNamingAndTriggeringPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">  
 <!-- 파일당 최고 용량 10MB -->  
 <maxFileSize>10MB</maxFileSize>  
 </timeBasedFileNamingAndTriggeringPolicy>  
 <!-- 일자별 로그파일 최대 보관주기(일단위) 만약 해당 설정일 이상된 파일은 자동으로 제거-->  
 <maxHistory>20</maxHistory>  
 <fileNamePattern>${LOGS\_ABSOLUTE\_PATH}/rs/decoder/decoder.%d{yyyy-MM-dd}\_%i.log</fileNamePattern>  
 </rollingPolicy>  
 </appender>  
  
 <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">  
 <layout class="ch.qos.logback.classic.PatternLayout">  
 <Pattern>[%d{yyyy/MM/dd HH:mm:ss.SSS}] [%5p] [%3t] %C{1} %M => %m%n</Pattern>  
 </layout>  
 </appender>  
 <!--<logger name="com.biz.communicator" level="info"/>-->  
  
 <logger name="com.biz.communicator.service.rs.active.msgreq" level="info">  
 <appender-ref ref="FILE\_RS\_MSG"/>  
 </logger>  
 <logger name="com.biz.communicator.service.rs.active.report" level="info">  
 <appender-ref ref="FILE\_RS\_REPORT"/>  
 </logger>  
 <logger name="com.biz.communicator.service.ibk.active" level="info">  
 <appender-ref ref="FILE\_IBK\_SENDER"/>  
 </logger>  
 <logger name="com.biz.communicator.service.ibk.passive" level="info">  
 <appender-ref ref="FILE\_IBK\_RECEIVER"/>  
 </logger>  
  
 <logger name="com.biz.communicator.service.ibk.codec" level="debug">  
 <appender-ref ref="FILE\_IBK\_DECODER"/>  
 </logger>  
  
 <logger name="com.biz.communicator.service.rs.codec" level="debug">  
 <appender-ref ref="FILE\_RS\_DECODER"/>  
 </logger>  
  
  
 <root level="info">  
 <appender-ref ref="STDOUT"/>  
 <appender-ref ref="FILE" />  
 </root>  
  
 <!--<root level="info">  
 <appender-ref ref="STDOUT" />  
 <appender-ref ref="FILE" />  
 </root>-->  
  
</configuration>

- 로그back을사용하였을 경우 이점 (<https://beyondj2ee.wordpress.com/2012/11/09/logback-%EC%82%AC%EC%9A%A9%ED%95%B4%EC%95%BC-%ED%95%98%EB%8A%94-%EC%9D%B4%EC%9C%A0-reasons-to-prefer-logback-over-log4j/>)

(1) Faster implementation  
“LOGBack“은 새롭게 작성 한것이 아니라, 오랫동안 검증된 “LOG4J”의 아키텍쳐 기반으로  
 재작성 되었습니다.  
특히 성능은 약 10배 정도 빠르게 개선을 했고, 또한 메모리 점유도 보다 적게 사용을 합니다.  
역시 성능에 관련된 얘기 입니다.

(2) Extensive battery of tests  
위에서 말씀 드렸듯 “새로운 오픈소스” 이지만 , 이미  “Log4j” 시절 부터 수년간 수 많은 광범위한 테스트  
 를 진행했고, 더욱더 많은 테스트의 반복 과정을 수행 했습니다.  
“Log4j보다 훨씬 뛰어난 가장 큰 이유중 하나“라고 합니다.  
그 만큼 “높은 쿼리티“에 대한 자신감을 엿볼수 있습니다.

(3) logback-classic speaks SLF4J natively  
“LOGBack“은 크게 3가지 컴포넌트로 구성이 되어 있습니다.  
“logback-core“는 말 그대로 핵심 코어 컴포넌트 입니다.  
“logback-classic“은 “slf4j“에서 사용이 가능하도록 만든 플러그인 컴포넌트 입니다.  
“logback-access“는 사용하는 어플리케이션이 “웹 어플리케이션“일 경우 빛을 바라는  
 컴포넌트 입니다.” HTTP 요청에 대한 강력한 디버깅 기능을 제공 합니다.”

(4) Extensive documentation  
“오픈소스 선택“에 있어서 레퍼런스 메뉴얼은 상당히 중요한 포인트 입니다.  
“LOGBack“은 지속적이면서, 상세한 메뉴얼을 제공 합니다.

(5) Configuration files in XML or Groovy  
logging 설정에 대한 syntax는 기본적으로 “XML” 형태로 되어 있습니다.  
또한 Groovy의 “syntax” 를 지원 합니다.

(6) Automatic reloading of configuration files  
개인적으로 호감이 가는 기능중 하나 입니다. 일하는 도메인 마다 틀리겠지만  
 기본적으로 “운영 서버” 모드에서는 로그레벨을 “WARN” 또는 “ERROR” 입니다.

하지만 만약 운영중에 좀더 상세한 로그를 보기 원할 경우가 있습니다.  
예를 들어서 “INFO” 레벨로 변경하는 경우 입니다.

“log4j” 같은 경우는 다음과 같이  “서버를 셧다운 -> 재설정 -> 서버 기동” 의  
 절차로 진행을 합니다. 즉, 핵심 포인트는 “서버를 재기동” 해야 한다는 것입니다.

이러한 매커니즘은 “내부 스캐닝하는 별도의 쓰레드“가 감지를 하기 때문 입니다.  
하지만 이런 “스캐닝 쓰레드”는 메모리에 대한 점유율을 최소화 하며,  
심지어 “100개의 쓰레드가 초당 백만 invocation“을 발생해도 시스템에  
 크게 무리를 주지 않는 다고 합니다.

주로 이런 기능은 “WAS에서 동장하는 웹 어플리케이션“에서 많이 사용이 될듯 합니다.

(7) Graceful recovery from I/O failures  
“Log4j” 같은 경우  “JDBC” , “Socket“등 다양한 “Appender“를 지원 합니다.  
특히. 대다수의 환경에서는 “File”에 제일 많이 로그를 저장 할 것입니다.

하지만 만약 “파일 서버가 일시적으로 장애가 발생 할경우” , “파일 서버가 복구 될때까지”  
 “어플리케이션을 중지 시켰다가” 다시 파일 서버가 복구되면, 그때 서버를 재기동 할것 입니다.  
하지만 “LOGBack“은 “서버 중지 없이, 이전 시점부터 복구를 graceful “하게 지원을 합니다.

(8) Automatic removal of old log archives  
대부분 환경에서는 “하나의 파일”에 기록하는 것이 아니고, “특정 시간” 또는 “특정 파일 사이즈”  
로 “Rolling” 해서 “Archiving“을 할 것입니다.

하지만 “Archiving된 로그 파일“을 계속 유지하지 않고, 일정 기간 지나면  
“서비스에  부담을 주지 않기 위해서” 파일을 삭제 할것 입니다.

이럴경우 “Crontab” 또는 다른 삭제 스케줄러를 설정 또는 개발을 할것입니다.  
“LOGBack“은 “maxHistory“라는 설정 정보를 통해서 주기적으로 “Archive”  
파일을 자동 삭제 합니다.  
“maxHistory“의 값이 “12“일 경우, “12개월 이후 삭제 하라는” 뜻입니다.

(9) Automatic compression of archived log files  
“Archived Log File“를 만들때 비동기적으로 롤링 할 동안  
 자동 압축을 실행 합니다. 심지어 파일 용량이 커도, 또한  
 이러한 압축 작업은 어플리케이션을 block 시키지 않습니다.

(10) Prudent mode  
만약 하나의 서버에서 “다수의 JVM“이 기동중일 경우 “하나의 파일“에  
 로그를 수집하려고 할때 사용하는 기능 입니다.

즉, “Log Aggregation” 기능 입니다. 조금만 아이디어를 내면, 매우 유용하게  
 사용이 가능 합니다.  
다만 서로 쓰레드간 writing 할때 경합이 생기기 때문에 대량의 데이터를 사용할때  
 는 다소 피해야 합니다.

그렇기 때문에 반드시 해당 옵션을 적용시 “LOGBack” 메뉴얼을 참고 하시기 바랍니다.

(11) Lilith  
“Lilith“은 “현재 발생하는 로그 이벤트에 대한 상태 정보를 볼수 있도록 지원 하는  
뷰어” 입니다.  
“log4j“를 지원하는 “chainsaw” 와 비슷한 기능을 합니다. 다만 차이점은 “Lilith”  
 “Large data“를 지원 합니다.

(12) Conditional processing of configuration files  
빌드를 해본 사람이라면 아실듯 합니다. 빌드시 제일 골치가 아픈 것이 “config” 정보 와  
“log file path” 입니다.  
이유는 어플리케이션이 구동하는 환경이 “로컬, staging, cbt” 마다 틀리기 때문 입니다.  
이런점을 해결 하기 위해서 “Maven Filter” 기능을 사용 하거나,  “JVM -D환경변수“를  
 통해서 각 환경 마다 선택적으로 “log 설정파일“을 선택하도록 구성을 할 것입니다.

“LOGBack“은 이런 이슈를 좀더 “Graceful“하게 지원을 합니다.  
“JSTL” 같이 분기 스크립트 (<if>, <then> and <else>)를 제공해서 하나의 파일  
 에서 다양한 빌드 환경을 제공 하도록 지원을 합니다.

(13) Filters  
“Filter” 기능은 “Log4j“에서 지원하는 기능입니다. 주 기능은 “로깅을 남길지 , 말지를”  
핸드링 할수 있는 기능 입니다.  
좀더 이해를 돕기 위해서 “UseCase“를 설명 드리겠습니다.

만약 “A“라는 사용자에게 비즈니스적으로 문제점이 발견이 되었습니다.  
이럴경우는 “logical exception“이기 때문에 원인을 잡기가 쉽지가 않습니다.  
또한 현재 “운영 서버“의 “Log LEVEL“이 “ERROR“일 경우 로그를 확인 할수가 없을  
 것입니다.  
더우기 “운영 서버 logical exception“은 “staging” 환경과 데이터가 틀리기 때문에  
 실제 운영 로그를 확인하지 않고서는 재현을 할수가 없습니다.

그래서 “운영 서버의 로그 레벨“을   “DEBUG“로 수정합니다. 하지만 로그는 보이지만 전체 사용자가  
 해당이 되기 때문에 로그 데이터를 분석하기가 어렵습니다.  
이럴 경우 “Filter“를 사용해서 “다른 사용자“는 “ERROR” 레벨을 유지하고,  
“A” 사용자만 “DEBUG“로 설정을 하면, 분석하는데 많은 도움을 받을 수 있습니다.

(14) SiftingAppender  
“SiftingAppender“는 “Filter“의 기능과 유사하면서 다른 기능을 제공 합니다.  
“로그 파일을 특정 주제별로 분류“를 할 수 있도록 합니다.  
예를 들어서 “HTTP Session“별로 로그 파일을 저장한다거나, “사용자별“로 별도의 로그파일을  
 저장 할 수 있습니다.

(15) Stack traces with packaging data  
자바 어플리케이션에서 “Exception“이 발생을 하면 “Stack Trace“를 출력을 합니다.  
자바는 “Exception” 정의가 잘되어 있어서 쉽게 “디버깅“을 할 수 있습니다.

하지만 제일 디버깅이 힘든 것중 하나가 “라이브러리에 의한 Exception” 입니다.  
이럴 경우 “LOGBack“은 Exception 발생시 참조 했던 외부 라이브러리 버전을  
 출력하게 해줍니다.

(16) Logback-access, i.e. HTTP-access logging with brains, is an integral part of logback  
위에서 언급했듯이 “logback-access“는 “웹 어플리케이션” 사용시 유용한 툴로써 제공 됩니다.  
특히나 요새는 전통적인 “HTML” 출력이 아닌 “REST 서버“로써의 역할을 많이 합니다.  
이럴 경우 “HTTP 디버깅“을 제공 합니다.  
Conclusion

지금 까지 간략하게 “LOGBack” 기능을 살펴 봤습니다.  
이외에 유용하고, 멋진 기능들이 많이 있습니다.  
“logging“의 중요함은 지나치게 말씀을 드려도 지나치지 않습니다.  
특히나 “클라우드 빅데이터” 시대에서 “logger“는 바로 최초 시작점 입니다.

제일 궁금했던 것이 “log4j를 만든 사람이 왜 ? logback를 만들었을까?” 입니다.

개인적 생각은 “log4j“를 “하위 호환성을 유지하면서 큰 리팩토링을 하기에는 다소 부담감”  
과 “상용 버전을 위한 비즈니스적 관점“이 아닐까 조심 스럽게 말씀 드리고 싶습니다.

“LOGBack“의 큰 매력은 “Log4J 와 상당히 친숙함” 일것입니다. 이말은 그 만큼 사용하는데  
 있어서 낯설지 않다는 얘기 입니다.  
심지어 “log4j.properties“를 “logback.xml” 설정 파일로 변환하는 “웹 변환기“도 제공 합니다.

또한 “SLF4J“를 지원하기 때문에 마음에 들지 않으면 “언제든지 다른 로거로 swiching“이 가능 합니다.

출처: <https://goddaehee.tistory.com/45> [갓대희의 작은공간]