1. Log4j 란?
   -ref : https://kyome.tistory.com/57
   -ref : https://to-dy.tistory.com/20
   1)Program을 작성하는 도중에 log를 남기기 위해 사용되는 Java-based Logging Utility.
   2)Apache에서 만든 Open Source Library
   3)Debugging용 도구로 주로 사용.
   4)log4j의 최근 Version에 의하면 높은 등급에서 낮은 등급으로의 6개 log level을 가지고 있다.
   5)설정 File(xml, properties)에 대상별(Java에서는 Package)로 level을 지정이 가능하고 그 등급 이상의 log만 저장하는 방식이다.
   6)Log for Java의 줄임말

2. 목적
   1)System.out.println() 을 사용하여 log를 확인할 경우 사용하지 않게 되면 일일이 주석처리를 해야 한다.
   2)Log의 level이나 log문의 level에 따라서 log를 유연하게 출력하여 불필요한 업무를 줄이고 성능을 최적화 할 수 있다(그대로 둔다면 Program 성능에 영향을 미칠 수 있다).

3. 구성 요소
   1)Logger
      -Log 기록하고, Appender에 전달
      -Log의 주체 : Log File을 작성하는 Class
      -Log level을 가짐 (Log문의 level과 Logger level를 비교하여 log의 출력여부를 결정)
      -출력할 Message를 Appender에 전달

   2)Appender
      -Log를 File, Console, DB 등의 위치에 출력
      -전달된 log를 어디에 출력할지 결정 (Console / File / DB 등)

   3)Layout
      -Log 출력 형식 결정

4. Log Level종류
   -6개의 Level로 구성됨
   -상태 및 목적에 따라
   1)FATAL
      -아주 심각한 Error가 발생한 상태를 나타낸다.
      -Application 작동이 불가능한 상태

   2)ERROR
      -어떠한 요청을 처리하는 중 문제가 발생한 상태를 나타낸다.
      -요청을 처리할 수 없는 상태

   3)WARN
      -Program의 실행에는 문제가 없지만, 향후 System Error의 원인이 될 수 있는 경고성 Message를 나타낸다.

   4)INFO
      -어떠한 상태변경과 같은 정보성 Message를 나타낸다.

   5)DEBUG
      -개발 시 Debug 용도로 사용하는 Message를 나타낸다.

   6)TRACE
      -Debug level이 너무 광범위한 것을 해결하기 위해서 좀 더 상세한 Event를 나타낸다.

   7)Level은 FATAL > ERROR > WARN > INFO > DEBUG > TRACE의 순서대로 정렬된다.
   8)즉 DEBUG level로 설정하면 INFO ~ FATAL까지 모두 logging된다.

5. Appender 종류
   1)WriterAppender
      -Writer 객체에 Log를 남기는 Appender

   2)ConsoleAppender
      -System.out, System.err에 log를 남기는 Appender
      -Option

--Target : System.out / System.err
--Follow : true -> SystemOutStream 에 저장
--activeOption : appender를 활성화

3)FileAppender
-File에 Log를 남기는 Appender
-Option
--File : File명 Append : 추가 모드 여부 (true/false)
--BufferedIO : Buffer 사용 여부 (true/false)
--BufferSize
--Threshold : (AppenderSkelton으로부터 계승)
--ImmediateFlush : (WriteAppender로부터 계승)
--Encoding : (WriteAppender로부터 계승)

4)RollingFileAppender
-크기에 따라 File명을 변환하며 Log를 남기는 Appender
-자동 backup 및 기록
-Option
--MaxFileSize
--MaxBackupIndex : Log 최대 개수
--File, Append, BufferdIO, BufferSize, Threshold, ImmediateFlush, Encoding

5)DailyRollingFileAppender
-날짜에 따라 File명을 변환하며 Log를 남기는 Appender
-일정 단위로 log 기록
-Option
--DatePattern : 날짜 형식(yyyy-MM, yyyy-ww,yyyy-MM-dd, yyyy-MM-dd-a, yyyy-MM-dd-HH 등등)

6)RollingFileAppender
-크기에 따라 File명을 변환하며 Log를 남기는 Appender

7)AsyncAppender
-Logging Event 발생시 Thread를 생성하여 Log를 남기는 Appender
-Option
--triggeringPolicy
--rollingPolicy
--org.apache.log4j.rolling.TimeBasedRollingPolicy : Time base
--org.apache.log4j.rolling.SizeBasedTriggeringPolicy : Size base
--org.apache.log4j.rolling.FilterBasedTriggeringPolicy : Filter base
org.apache.log4j.rolling.FixedWindowRollingPolicy : Index base Backup File

8)SMTPAppender
-Log를 Email로 전달하는 Appender

9)JDBCAppender
-DB에 log를 기록

10)NTEventAppender
-Windows System Event Log로 Message 전송

6. Layout
1)어떤 형식으로 출력할지 결정
2)Layout 종류
-DateLayout
-HTMLLayout
-PatternLayout (일반적으로 사용)
-SimpleLayout
-XMLLayout

7. PatternLayout
1)PatternLayout (org.apache.log4j.PatternLayout) 상세 설명
2)%C
-Class명을 출력
-설정을 추가하여 Class 이름 또는 특정 Package 이상만 출력하도록 설정 가능
-만일 Class의 구조가 org.apache.xyz.SomeClass라면 %C{2} -> xyz.SomeClass 로 출력.

3)%m : Log로 전달된 Message를 출력.
4)%M : Log를 수행한 Method명을 출력.
5)%n : 줄 바꿈
6)%p : Log Event명 (debug, info, warn, error, fatal등) priority 출력.
7)%r : Log 처리시간 (milliseconds), Application 시작 이후부터 logging이 발생한 시점의 시간 출력.
8)%d : Log 시간을 출력.
-java.text.SimpleDateFormat에서 적절한 출력 format을 지정가능.
-%d{HH:mm:ss}
9)%F : File 이름을 출력.
-Log 시 수행한 Method, Line번호가 함께 출력.
-Logging이 발생한 Program File 이름 출력.
10)%l (location) : Logging Event가 발생한 Class의 Full Name.
-Method명(File명:Line번호) 출력.
11)%L : Logging이 발생한 caller의 Line수 출력.
12)%c : Category 출력.
-Category 구조가 a.b.c처럼 되어 있다면 %c{2}는 b.c 출력.
13)T : Logging Event가 발생한 Thread명
14)% : % 표시 출력.

8. 작업순서
1)Maven 환경일 때 아래의 dependency 추가

```
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.17</version>
</dependency>
```

2)log4j.xml을 생성(수정)
-Spring 으로 세팅하면 기본적으로 log4j를 포함하고 있기 때문에 별도 생성할 필요는 없지만 혹시 없다면 log4j.xml를 생성해야 한다.
-Path : src/main/resources/log4j.xml

3)log4j.xml 구성
-Appender, logger, root 로 구성.
-Appender는 Log 찍을 대상이나 어떤 방식으로 찍을 지를 결정.
-기본 설정값은 console에 찍는 방식으로 되어있고 PatternLayout Class를 사용해서 Layout을 잡는다.
-logger는 Application Loggers라고 주석이 달린 것처럼 package와 같이 영역을 지정하고 해당 영역에서 사용할 logger를 정의하는 태그.
-하위 Parameter로 level을 받아서 출력할 log level을 정한다.
-appender-ref tag를 사용하여 ref 값에 참조할 appender를 입력하여 출력방식을 정할 수 있다.
-root는 default라고 생각.
-설정하지 않은 logger에 대해서만 root logger를 출력하게 한다.
-구성요소는 logger와 유사하다.

**같은 log가 두번 찍힌다면 Additivity 속성에 대해 확인

4)설정 File 기본값

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration PUBLIC "-//APACHE//DTD LOG4J 1.2//EN" "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">

    <!-- Appenders -->
    <appender name="console" class="org.apache.log4j.ConsoleAppender">
        <param name="Target" value="System.out" />
        <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern" value="%-5p: %c - %m%n" />
        </layout>
    </appender>

    <!-- Application Loggers -->
    <logger name="com.example.biz">
        <level value="info" />
    </logger>

    <!-- 3rdparty Loggers -->
    <logger name="org.springframework.core">
```

```
        <level value="info" />
    </logger>

    <logger name="org.springframework.beans">
        <level value="info" />
    </logger>

    <logger name="org.springframework.context">
        <level value="info" />
    </logger>

    <logger name="org.springframework.web">
        <level value="info" />
    </logger>

    <!-- Root Logger -->
    <root>
        <priority value="info" />
        <appender-ref ref="console" />
    </root>

</log4j:configuration>
```

9. 사용방법
   1)사용할 Class에 private static final logger 변수 선언
   2)이렇게 선언을 하면 xml에서 해당 Application(Package)의 Logger 생성
   3)선언
   -Logger의 name을 Package로 잡았을 경우, Package내의 Class이름.class를 parameter로 선언

```
import org.apache.log4j.Logger;
...
private static final Logger logger = Logger.getLogger(선언한 Class명.class);
```

   -Logger의 name을 변수명으로 잡았을 경우, logger의 이름(문자열)을 parameter로 선언

```
private static final Logger logger = Logger.getLogger("test");
<!-- Application Loggers --> <logger name="test"> <level value="info" /> </logger>
```

10. log4j.xml이 아닌 log4j.properties로 설정하기
   -ref : https://shxrecord.tistory.com/126

```
# Root logger option

# Log4j Setting file
log4j.rootLogger=DEBUG, console, R

# Daily file log
log4j.appender.R=org.apache.log4j.DailyRollingFileAppender
log4j.appender.R.File=D:\\ws_study/shXorld/logs/web.log
log4j.appender.R.DatePattern='.'yyyy-MM-dd
log4j.appender.R.layout=org.apache.log4j.PatternLayout
log4j.appender.R.layout.ConversionPattern=[%d{HH:mm:ss}][%-5p](%F:%L)-%m%n

# Console log
log4j.appender.console=org.apache.log4j.ConsoleAppender
log4j.appender.console.layout=org.apache.log4j.PatternLayout
log4j.appender.console.layout.ConversionPattern=[%d{HH:mm:ss}][%-5p](%F:%L)-%m%n
log4j.appender.console.ImmediateFlush=true
```

11. 위 code 설명
   log4j.rootLogger=DEBUG, console, R
   - DEBUG
   : Log level을 의미하며 DEBUG로 설정한 message를 출력시키겠다는 의미.
   *Log level은 [TRACE < DEBUG < INFO < WARN < ERROR < FATAL] 로 오른쪽으로 갈 수록 높은 level이다.

(예를 들어, WARN으로 설정한 경우 WARN level 이상의 경우(WARN, ERROR, FATAL)만 출력된다)

- console, R
: appender의 이름을 뜻하며, console이라는 이름과 R이라는 이름을 사용하겠다는 의미이다.
: appender
: Logging message를 File에 기록할 것인지 Console창에 출력할 것인지 구분하기 위한 매개체

#Log File 기록하는 부분
log4j.appender.R=org.apache.log4j.DailyRollingFileAppender
: R이라는 appender는 File에 기록하겠다는 의미이다.

log4j.appender.R.File=D:\\ws_study/shXorld/logs/web.log
: Log를 남길 File명과 경로를 지정한다.

log4j.appender.R.DatePattern='.'yyyy-MM-dd
: Log File명의 format으로 하루가 지나면 Log File의 Backup File을 생성하는데 이 때 File명 뒤에 날짜가 붙는다.
ex) web.log.2019-01-09

log4j.appender.R.layout=org.apache.log4j.PatternLayout :
: layout에는 HTMLLayout, PatternLayout, XMLLayout, SimpleLayout 등이 있는데 PatternLayout은 다양한 Log message 조합을 만들 수 있다.

log4j.appender.R.layout.ConversionPattern=[%d{HH:mm:ss}][%-5p](%F:%L)-%m%n
: Log의 출력 layout을 설정한다.
- %d{HH:mm:ss} : %d는 Logging Event가 발생한 시간을 뜻하며 {HH:mm:ss}가 뒤에 붙으면 시:분:초 형태로 기록이 된다.
- %-5p : 좌측 정렬로 Log level을 기록한다.
- %F : Log가 발생한 File명
- %L : :Log가 발생한 Line번호
- %m : Log의 내용
- %n : Enter 처리

#Console창에 보여질 부분
log4j.appender.console=org.apache.log4j.ConsoleAppender
: console이라는 appender는 Console창에 출력하겠다는 의미이다.

log4j.appender.console.layout=org.apache.log4j.PatternLayout
: 위와 동일

log4j.appender.console.layout.ConversionPattern=[%d{HH:mm:ss}][%-5p](%F:%L)-%m%n
: 위와 동일

log4j.appender.console.ImmediateFlush=true
: Log message의 buffer 유무로 기본값은 true이다.

12. Log Pattern 사례
-ref : https://cofs.tistory.com/314?category=627167
[%5p] %d{hh\:mm s} (%F\:%L) %c{1}.%M \: %m%n
-Log level 발생한시간 File명 Line수 Category Method 내용 순.
위 pattern으로 console에 출력되는 log 모양은 다음과 같다.

```
[ INFO] 01:34 56 (AbstractHandlerMethodMapping.java:220) RequestMappingHandlerMapping.registerHandlerMethod : Mapped ...
[ INFO] 01:34 56 (AbstractHandlerMethodMapping.java:220) RequestMappingHandlerMapping.registerHandlerMethod : Mapped ...
[ INFO] 01:34 56 (AbstractHandlerMethodMapping.java:220) RequestMappingHandlerMapping.registerHandlerMethod : Mapped ...
[ INFO] 01:34 56 (AbstractHandlerMethodMapping.java:220) RequestMappingHandlerMapping.registerHandlerMethod : Mapped ...
[ INFO] 01:34 56 (AbstractHandlerMethodMapping.java:220) RequestMappingHandlerMapping.registerHandlerMethod : Mapped ...
[ INFO] 01:34 56 (AbstractHandlerMethodMapping.java:220) RequestMappingHandlerMapping.registerHandlerMethod : Mapped ...
[ INFO] 01:34 56 (AbstractHandlerMethodMapping.java:220) RequestMappingHandlerMapping.registerHandlerMethod : Mapped ...
[ INFO] 01:34 56 (AbstractHandlerMethodMapping.java:220) RequestMappingHandlerMapping.registerHandlerMethod : Mapped ...
```

[INFO] 01:34 56 (AbstractHandlerMethodMapping.java:220) RequestMappingHandlerMapping.registerHandlerMethod : Mapped ...

-Level : INFO
-Log발생 File : AbstractHandlerMethodMapping.java
-Category : RequestMappingHandlerMapping
-Method : registerHandlerMethod