JAVA LOGGERY

Przemysław Grzesiowski
6 października 2018

1. Logowanie

Śledzimy działanie aplikacji

Logowanie
Dlaczego to takie ważne?

- w przypadku awarii lub błędnego zachowania aplikacji ułatwiają namierzenie bezpośredniej lub pośredniej przyczyny incydentu

Logowanie
Dlaczego to takie ważne?

- w przypadku awarii lub błędnego zachowania aplikacji ułatwiają namierzenie bezpośredniej lub pośredniej przyczyny incydentu
- pozwalają na śledzenie działania aplikacji

Logowanie
Dlaczego to takie ważne?

- w przypadku awarii lub błędnego zachowania aplikacji ułatwiają namierzenie bezpośredniej lub pośredniej przyczyny incydentu
- pozwalają na śledzenie działania aplikacji
- nie wpływają bezpośrednio na funkcjonalność aplikacji

Logowanie
Dlaczego to takie ważne?

- w przypadku awarii lub błędnego zachowania aplikacji ułatwiają namierzenie bezpośredniej lub pośredniej przyczyny incydentu
- pozwalają na śledzenie działania aplikacji
- nie wpływają bezpośrednio na funkcjonalność aplikacji

- logowanie odbywa się za pośrednictwem Loggerów.

Co logujemy?

- obsłużone wyjątki
- zdarzenia nieudanego tworzenia nowego konta
- próby pobrania danych, zalogowania się do zasobu

Co logujemy?

- obsłużone wyjątki
- zdarzenia nieudanego tworzenia nowego konta
- próby pobrania danych, zalogowania się do zasobu
- informacje o kluczowych krokach procesów
- informacje o użytych konfiguracjach

Obsługa wyjątków

```
try {
    OutputStream outputStream = new BufferedOutputStream(new FileOutputStream(new File("tmp.txt")));
} catch (FileNotFoundException e) {
    e.printStackTrace();
}
```

Obsługa wyjątków

```
try {
    OutputStream outputStream = new BufferedOutputStream(new FileOutputStream(new File("tmp.txt")));
} catch (FileNotFoundException e) {
    e.printStackTrace();
}
```

Throwable	initCause(Throwable cause) Initializes the *cause* of this throwable to the specified value.
void	printStackTrace() Prints this throwable and its backtrace to the standard error stream.
void	printStackTrace(PrintStream s) Prints this throwable and its backtrace to the specified print stream.
void	printStackTrace(PrintWriter s) Prints this throwable and its backtrace to the specified print writer.
void	setStackTrace(StackTraceElement[] stackTrace) Sets the stack trace elements that will be returned by getStackTrace() and printed by printStackTrace() and related methods.
String	toString() Returns a short description of this throwable.

Obsługa wyjątków

```
try {
    OutputStream outputStream = new BufferedOutputStream(new FileOutputStream(new File("tmp.txt")));
} catch (FileNotFoundException e) {
    e.printStackTrace();
}
```

```
try {
    OutputStream outputStream = new BufferedOutputStream(new FileOutputStream(new File("tmp.txt")));
} catch (FileNotFoundException e) {

}
```

Obsługa wyjątków

```
try {
    OutputStream outputStream = new BufferedOutputStream(new FileOutputStream(new File("tmp.txt")));
} catch (FileNotFoundException e) {
    e.printStackTrace();
}
```

```
try {
    OutputStream outputStream = new BufferedOutputStream(new FileOutputStream(new File("tmp.txt")));
} catch (FileNotFoundException e) {

}
```

```
try {
    OutputStream outputStream = new BufferedOutputStream(new FileOutputStream(new File("tmp.txt")));
} catch (FileNotFoundException e) {
    logger.log(Level.FATAL, e.getMessage(), e);
}
```

2. Bliższe spojrzenie

Jak wygląda wpis?

- Każdy komunikat opatrzony jest poziomem, czasem wystąpienia zdarzenia oraz dodatkowymi informacjami opisującymi zdarzenie

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.0.3.RELEASE)

2018-06-17 16:55:55.601  INFO 6082 --- [           main] c.b.s.SpringBootLoggingApplication       : Starting SpringBootLoggingApplication v0.0.1-SNAPSHOT on Phoenix2 with PID 6082 (/home/andrea/git/tutorials/spring-boot-logging/target/spring-boot-logging-0.0.1-SNAPSHOT.jar started by andrea in /home/andrea/git/tutorials/spring-boot-logging)
2018-06-17 16:55:55.609  INFO 6082 --- [           main] c.b.s.SpringBootLoggingApplication       : No active profile set, falling back to default profiles: default
2018-06-17 16:55:55.749  INFO 6082 --- [           main] ConfigServletWebServerApplicationContext : Refreshing org.springframework.boot.web.servlet.context.AnnotationConfigServletWebServerApplicationContext@1d9b7cce: startup date [Sun Jun 17 16:55:55 CEST 2018]; root of context hierarchy
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.springframework.cglib.core.ReflectUtils$1 (jar:file:/home/andrea/git/tutorials/spring-boot-logging/target/spring-boot-logging-0.0.1-SNAPSHOT.jar!/BOOT-INF/lib/spring-core-5.0.7.RELEASE.jar!/) to method java.lang.ClassLoader.defineClass(java.lang.String,byte[],int,int,java.security.ProtectionDomain)
WARNING: Please consider reporting this to the maintainers of org.springframework.cglib.core.ReflectUtils$1
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
2018-06-17 16:55:59.231  INFO 6082 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
2018-06-17 16:55:59.312  INFO 6082 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2018-06-17 16:55:59.313  INFO 6082 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet Engine: Apache Tomcat/8.5.31
2018-06-17 16:55:59.331  INFO 6082 --- [ost-startStop-1] o.a.catalina.core.AprLifecycleListener   : The APR based Apache Tomcat Native library which allows optimal performance in production environments was not found on the java.library.path: [/usr/java/packages/lib:/usr/lib/x86_64-linux-gnu/jni:/lib/x86_64-linux-gnu:/usr/lib/x86_64-linux-gnu:/usr/lib/jni:/lib:/usr/lib]
2018-06-17 16:55:59.471  INFO 6082 --- [ost-startStop-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2018-06-17 16:55:59.472  INFO 6082 --- [ost-startStop-1] o.s.web.context.ContextLoader            : Root WebApplicationContext: initialization completed in 3737 ms
2018-06-17 16:55:59.926  INFO 6082 --- [ost-startStop-1] o.s.b.w.servlet.ServletRegistrationBean  : Servlet dispatcherServlet mapped to [/]
2018-06-17 16:55:59.933  INFO 6082 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'characterEncodingFilter' to: [/*]
2018-06-17 16:55:59.933  INFO 6082 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'hiddenHttpMethodFilter' to: [/*]
2018-06-17 16:55:59.933  INFO 6082 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'httpPutFormContentFilter' to: [/*]
2018-06-17 16:55:59.934  INFO 6082 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'requestContextFilter' to: [/*]
2018-06-17 16:56:00.228  INFO 6082 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/**/favicon.ico] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
2018-06-17 16:56:00.810  INFO 6082 --- [           main] s.w.s.m.m.a.RequestMappingHandlerAdapter : Looking for @ControllerAdvice: org.springframework.boot.web.servlet.context.AnnotationConfigServletWebServerApplicationContext@1d9b7cce: startup date [Sun Jun 17 16:55:55 CEST 2018]; root of context hierarchy
2018-06-17 16:56:01.023  INFO 6082 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/]}" onto public java.lang.String com.baeldung.springbootlogging.LoggingController.index()
2018-06-17 16:56:01.044  INFO 6082 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/error],produces=[text/html]}" onto public org.springframework.web.servlet.ModelAndView org.springframework.boot.autoconfigure.web.servlet.error.BasicErrorController.errorHtml(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletResponse)
2018-06-17 16:56:01.047  INFO 6082 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/error]}" onto public org.springframework.http.ResponseEntity<java.util.Map<java.lang.String, java.lang.Object>> org.springframework.boot.autoconfigure.web.servlet.error.BasicErrorController.error(javax.servlet.http.HttpServletRequest)
2018-06-17 16:56:01.119  INFO 6082 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/webjars/**] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
2018-06-17 16:56:01.120  INFO 6082 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/**] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestH
```

- Logger przekazuje komunikat do wybranego miejsca docelowego (standardowe wyjście, plik, zdalny serwer logów)

Może tak samemu napisać mechanizm logowania?

- Najczęstszym przypadkiem jest logowanie do pliku co jest jedną z najdroższych operacji (zapis/odczyt)
- Własna próba tworzenia loggera może skończyć się spowolnieniem działania aplikacji lub utratą kontroli nad kolejnością logowanych zdarzeń (async)

Może tak samemu napisać mechanizm logowania?

- Najczęstszym przypadkiem jest logowanie do pliku co jest jedną z najdroższych operacji (zapis/odczyt)
- Własna próba tworzenia loggera może skończyć się spowolnieniem działania aplikacji lub utratą kontroli nad kolejnością logowanych zdarzeń (async)
- Biblioteki 3rd party radzą sobie z problemem kosztu obsługi pliku jak również **asynchroniczności** nieblokującej działania aplikacji

Wizualizacja logów

Logować możemy najróżniejsze zdarzenia: koniecznie błędy ale również każde logowanie użytkownika, informację o pełnym załadowaniu się serwisu, założonych kontach itp.

Wizualizacja logów

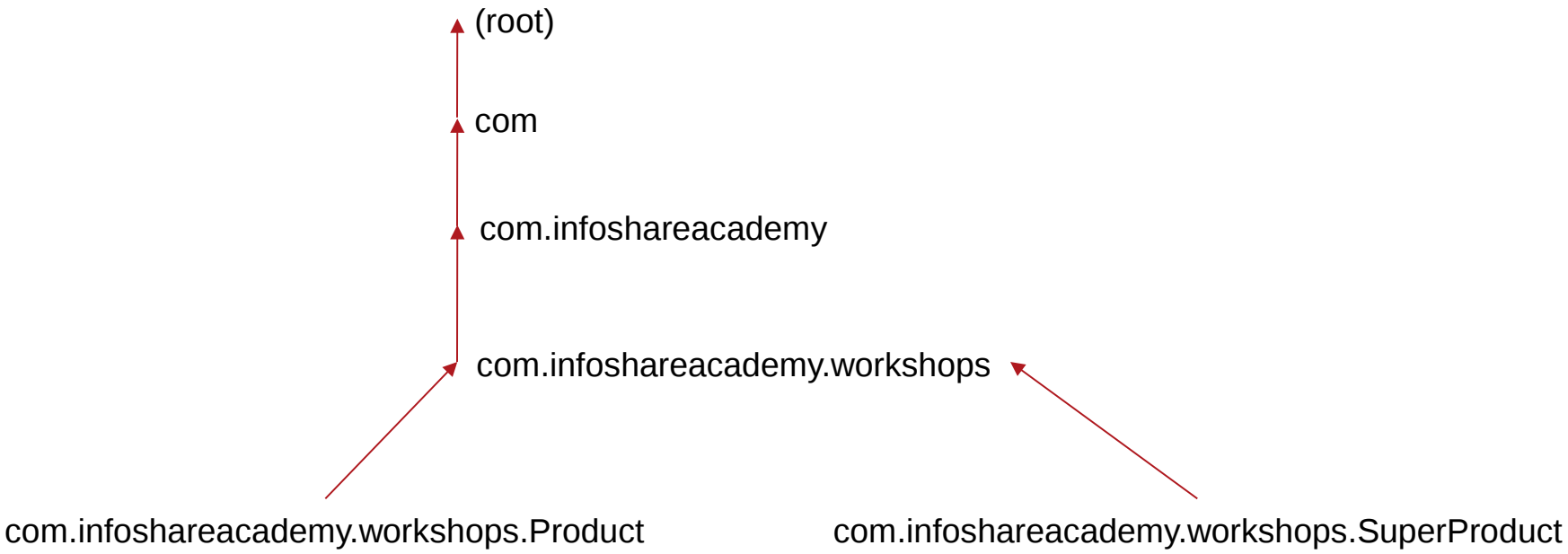
Logować możemy najróżniejsze zdarzenia: koniecznie błędy ale również każde logowanie użytkownika, informację o pełnym załadowaniu się serwisu, założonych kontach itp.

Takie informacje mają też wartość biznesową (pozwalają określić z jakich funkcjonalności korzystają użytkownicy, śledzić ich zachowania, przygotowywać statystyki rozkładu operacji w ciągu doby itp.)

Grafana
Przykład wizualizacji logów

http://play.grafana.org

Logger
Nazewnictwo i hierarchia



Logger
Nazewnictwo i hierarchia

- Każdy logger ma swoją nazwę (name) – najczęściej jest to nazwa klasy (pełna z nazwą pakietu - *fully-qualified-class-name (FQCN)*), w której jest on użyty

- Pełna nazwa klasy określa hierarchię loggerów
  Logger o nazwie: *com.infoshareacademy.domain.Product* należy do hierarchii *com*, oraz *com.infoshareacademy* ale już nie należy np. do *org.springframework*

Logger
Nazewnictwo i hierarchia

- Każdy logger ma swoją nazwę (name) – najczęściej jest to nazwa klasy (pełna z nazwą pakietu - *fully-qualified-class-name (FQCN)*), w której jest on użyty

- Pełna nazwa klasy określa hierarchię loggerów
  Logger o nazwie: *com.infoshareacademy.domain.Product* należy do hierarchii *com*, oraz *com.infoshareacademy* ale już nie należy np. do *org.springframework*

- Dla każdej hierarchii możemy stosować osobną konfigurację co uelastycznia podejście do tematu logów (na przykład w pliku mogą lądować tylko logi z danej hierarchii)

Poziomy logowania

- Poziom logowania to nic innego jak „ważność” danej wiadomości zapisywanej w logach

- Każda biblioteka ma różne zestawy poziomów:

„FATAL, ERROR, WARNING, INFO, DEBUG, TRACE”
„SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST”

Poziomy
logowania - log4j

Level	Description
OFF	The highest possible rank and is intended to turn off logging.
FATAL	Severe errors that cause premature termination. Expect these to be immediately visible on a status console.
ERROR	Other runtime errors or unexpected conditions. Expect these to be immediately visible on a status console.
WARN	Use of deprecated APIs, poor use of API, 'almost' errors, other runtime situations that are undesirable or unexpected, but not necessarily "wrong". Expect these to be immediately visible on a status console.
INFO	Interesting runtime events (startup/shutdown). Expect these to be immediately visible on a console, so be conservative and keep to a minimum.
DEBUG	Detailed information on the flow through the system. Expect these to be written to logs only. Generally speaking, most lines logged by your application should be written as DEBUG.
TRACE	Most detailed information. Expect these to be written to logs only. Since version 1.2.12

https://en.wikipedia.org/wiki/Log4j

Poziomy logowania

[http://javarevisited.blogspot.com/2011/05/top-10-tips-on-logging-in-java.html]

- DEBUG
  - the lowest restricted java logging level
  - everything we need to debug an application; should only be used on Development and Testing environment and must not be used in production environment.

- INFO
  - is more restricted than DEBUG level
  - messages which are informative purpose (e.g. *server has been started, Incoming messages, outgoing messages etc.*)

Poziomy logowania

[http://javarevisited.blogspot.com/2011/05/top-10-tips-on-logging-in-java.html]

- WARN
  - is more restricted than INFO
  - warning sort of messages (e.g. *Connection lost between client and server, Database connection lost, Socket reaching to its limit*)
  - you can setup alert (e.g. email) on these logging messages and let your support team monitor health of your java application and react
- ERROR
  - serious errors and exceptions
  - you can setup alert on these logging messages and let your support team monitor health of your java application and react
  - ERROR is serious for logging and you should always print it.

Poziomy logowania

[http://javarevisited.blogspot.com/2011/05/top-10-tips-on-logging-in-java.html]

- FATAL
  - designates very severe error events that will presumably lead the application to abort. After this mostly your application crashes and stopped.

W oparciu o poziomy logowania

- Poziomy mogą się przydać w celu np.:
  - Wysłania maila/zdarzenia jeśli zalogowany został wpis z poziomem FATAL czy ERROR
  - Zapisywania w dzienniku logów o poziomach wyższych niż wybrany – **czym grozi brak filtrowania poziomów w dzienniku produkcyjnym?**

library	levels	Standard appenders	License
Java Logging API	SEVERE WARNING INFO CONFIG FINE FINER FINEST	Sun's default Java Virtual Machine (JVM) has the following: ConsoleHandler, FileHandler, SocketHandler, MemoryHandler	default Java Virtual Machine (JVM), Comes with the JRE
Log4J	FATAL ERROR WARN INFO DEBUG TRACE	AsyncAppender, JDBCAppender, JMSAppender, LF5Appender, NTEventLogAppender, NullAppender, SMTPAppender, SocketAppender, SocketHubAppender, SyslogAppender, TelnetAppender, WriterAppender	Apache License, Version 2.0
Logging	FATAL ERROR WARN INFO DEBUG TRACE	Depends on the underlying framework	Apache License, Version 2.0
SLF4J	ERROR WARN INFO DEBUG TRACE	Depends on the underlying framework, which is pluggable	MIT License
tinylog	ERROR WARNING INFO DEBUG TRACE	ConsoleWriter, FileWriter, LogcatWriter, JdbcWriter, RollingFileWriter, SharedFileWriter and null (discards all log entries) [1]	Apache License, Version 2.0
Logback	ERROR WARN INFO DEBUG TRACE	AbstractServerSocketAppender, AbstractSocketAppender, AbstractSSLSocketAppender, AppenderBase, AsyncAppender, AsyncAppenderBase, ConsoleAppender, CountingConsoleAppender, CyclicBufferAppender, DBAppender, DBAppender, DBAppenderBase, FileAppender, ListAppender, NOPAppender, OutputStreamAppender, RollingFileAppender, ServerSocketAppender, ServerSocketAppender, SiftingAppender, SiftingAppender, SiftingAppenderBase, SMTPAppender, SMTPAppender, SMTPAppenderBase, SocketAppender, SocketAppender, SSLServerSocketAppender, SSLServerSocketAppender, SSLServerSocketAppenderBase, SSLSocketAppender, SSLSocketAppender, SyslogAppender, SyslogAppenderBase, TrivialLogbackAppender, UnsynchronizedAppenderBase	LGPL, Version 2.1

Repozytorium
GitHub

```
git clone
https://github.com/infoshareacademy
/jdqz2-loggery
```

3.
log4j v2

Maven
Zacznijmy od zależności

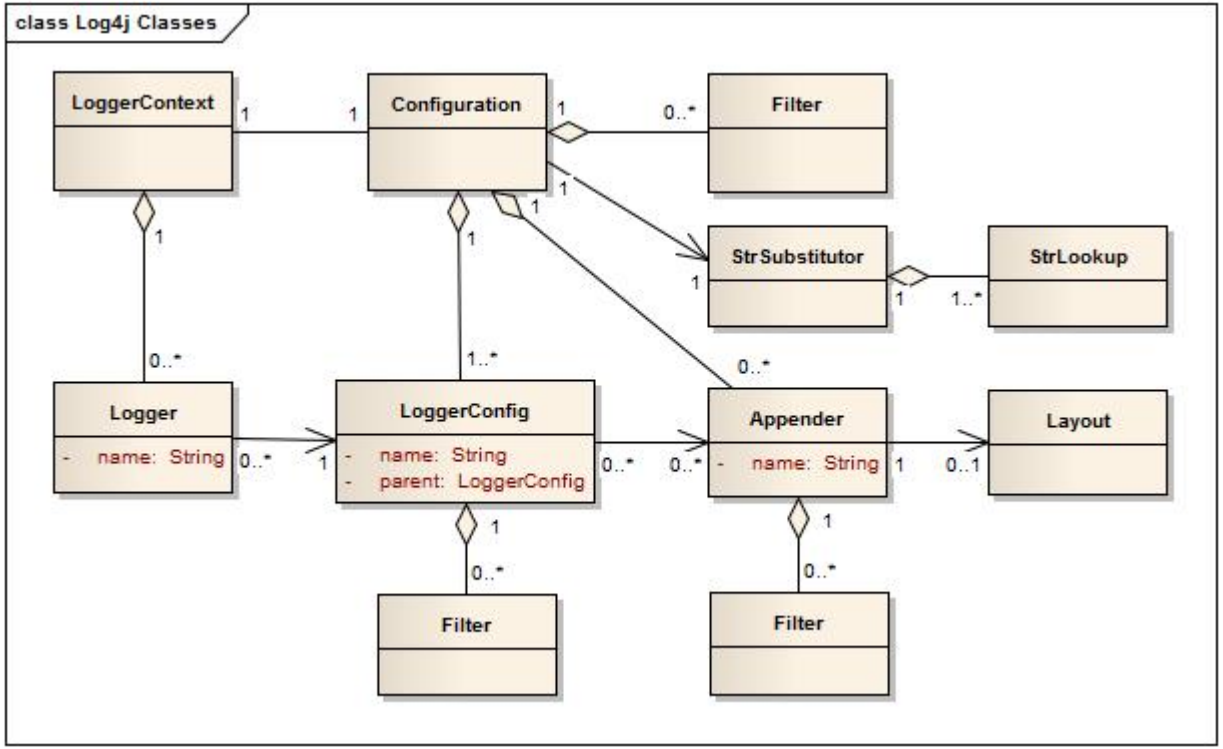
```
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
    <version>2.11.1</version>
</dependency>

<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-api</artifactId>
    <version>2.11.1</version>
</dependency>
```

https://logging.apache.org/log4j/2.x/maven-artifacts.html#Using_Log4j_in_your_Apache_Maven_build

log4j2
opis działania

- Aplikacja odwołuje się do LogManager’a po instancję Logger’a z konkretną nazwą
- LogManager zlokalizuje odpowiedni LoggerContext, a następnie przydzieli i zwróci instancję Logger’a (jeśli istnieje)
- Jeśli Logger jeszcze nie istnieje, zostanie utworzony i powiązany z LoggerConfig
- LoggerConfig tworzony jest z instancji Logger’a w pliku konfiguracyjnym. Służy również do obsługi LogEvents’ów i delegowania ich do Appenders’ów

Architektura
log4j2



Logger

- jest obiektem Javy
- ma swoją nazwę (name)
- do jednego loggera może istnieć wiele referencji (np. gdy istnieje wiele obiektów tej samej klasy)
- ma swoją konfigurację
  (jego konfiguracja jest zależna od konfiguracji innych logerów – o tym później)

log4j2
konfiguracja

Istnieje kilka sposobów konfiguracji loggera:

- Użycie konfiguracji opisanej jako XML, JSON, YAML
- Zaprogramowany, na przykład tworząc fabrykę konfiguracji i implementując konfigurację

log4j2
konfiguracja

Domyślnie, log4j2 sprawdza wartość właściwości systemowej: **log4j.configurationFile**. To właśnie w tym miejscu spodziewa się ścieżki do pliku konfiguracyjnego:

```
System.setProperty("log4j.configurationFile",
"log4j.xml");
```

lub jako VM Options:

```
-Dlog4j.configurationFile=log4j.xml
```

log4j2
konfiguracja domyślna

Jeśli log4j2 nie znajdzie pliku konfiguracyjnego rozpocznie poszukiwania na classpath:

- log4j2-test.properties
- log4j2-test.yaml lub log4j2-test.yml
- log4j2-test.jsn lub log4j2-test.json
- log4j2-test.xml
- log4j2.properties
- log4j2.yml lub log4j2.yaml
- log4j2.jsn lub log4j2.json
- log4j2.xml

log4j2
konfiguracja domyślna

ERROR StatusLogger No log4j2 configuration file found. Using default configuration: logging only errors to the console. Set system property 'log4j2.debug' to show Log4j2 internal initialization logging.

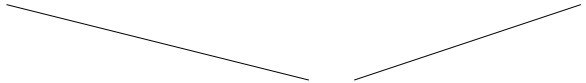
log4j2
konfiguracja domyślna

Jeśli nie zostanie znaleziony plik konfiguracji, zostaje załadowana domyślna konfiguracja (DefaultConfiguration) z domyślnym zachowaniem i tylko jednym loggerem (root)

- Logger: ROOT
- Level: ERROR
- Target: Console
- Format: %d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n

Deklaracja
log4j

```
Logger logger =
LogManager.getLogger(MyClass.class.getName());
```



="com.my.package.MyClass"

Zadanie 3.1:

1. Otwórz “emptyProj”, odpal program (App->main)
2. Dodaj zależności log4j2 do pliku pom.xml
3. W klasie App utwórz nowy logger, wykonaj logowanie (poziom info) np. “Hello world!” lub dzisiejsza data. Odpal. Czy widzisz działanie loggera w konsoli?

Konfiguracja

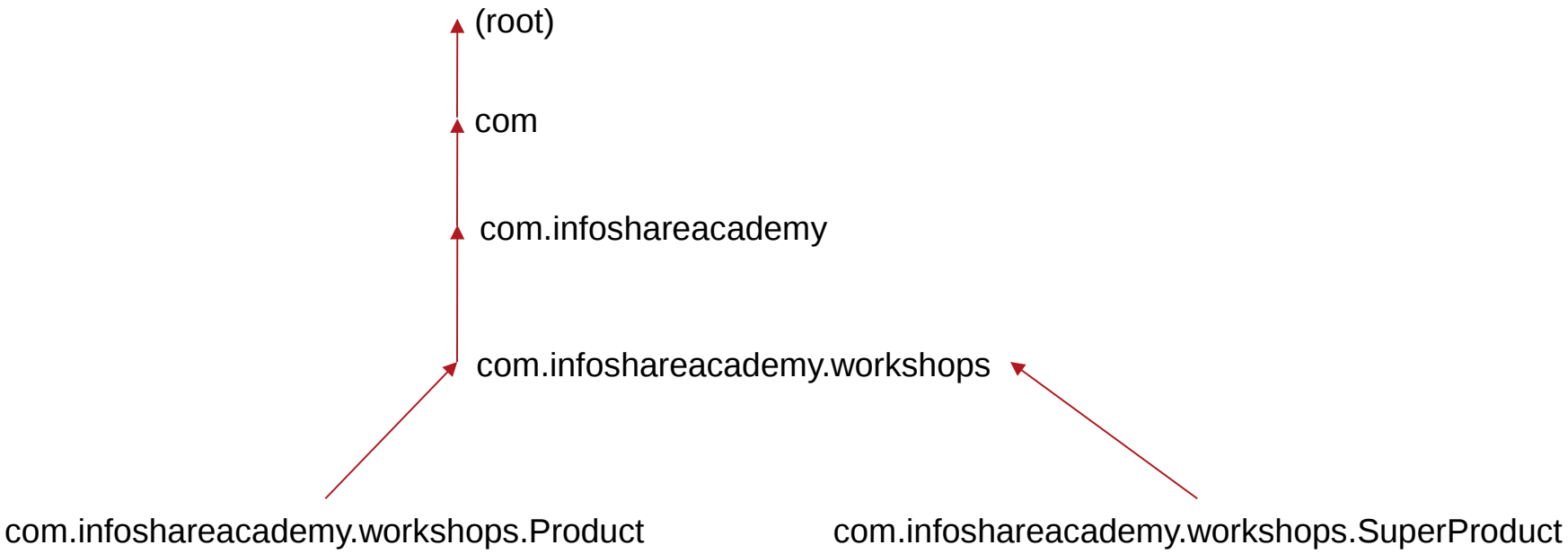
Message (log entry) level

setting

	FATAL	ERROR	WARN	INFO	DEBUG	TRACE
OFF						
FATAL	x					
ERROR	x	x				
WARN	x	x	x			
INFO	x	x	x	x		
DEBUG	x	x	x	x	x	
TRACE	x	x	x	x	x	x
ALL	x	x	x	x	x	x

x: Visible

Logger
Nazewnictwo i hierarchia



Root logger – logger “Adam”

- Logger „root” jest przypadkiem wyjątkowym, zawsze istnieje i jest na samej górze hierarchii loggerów
- Przykład pobrania referencji do loggera root:

```
Logger logger =
LogManager.getLogger(LogManager.ROOT_LOGGER_NAME);

Logger logger = LogManager.getRootLogger();
```

Użycie
log4j

```
logger.info(„Processing {} elements",
      list.size()
);
```

```
logger.info("This is simple log from class {} in package {}",
new Object[]{
      ConsoleLoggerExample.class.getName(),
      ConsoleLoggerExample.class.getPackage().getName()
});
```

log4j2
Logowanie do konsoli

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Configuration>
    <Appenders>
        <Console name="Console">
            <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
        </Console>
    </Appenders>
    <Loggers>
        <Logger name="root" level="info">
            <AppenderRef ref="Console"/>
        </Logger>
    </Loggers>
</Configuration>
```

Zadanie 3.1 – c.d.

1. Otwórz “emptyProj”, odpal program (App->main)
2. Dodaj zależności log4j2 do pliku pom.xml
3. W klasie App utwórz nowy logger, wykonaj logowanie(poziom info) np. “Hello world!” lub dzisiejsza data. Odpal.
4. Utwórz plik **main/resources/log4j2.xml** Zamieść w pliku XML konfigurację loggera wg przykładu z poprzedniego slajdu. Odpal.
5. Dodaj więcej wpisów do logu – np. “witaj Przemku!”, “fatalny błąd aplikacji” itp. Niech każdy wpis będzie innego poziomu (info, debug, error).
6. Modyfikuj plik xml (zmieniaj poziom logowania loggera root) - po każdej zmianie odapalaj aplikację i obserwuj konsolę.

Levels - Wartości numeryczne

LEVEL	WEIGHT
OFF	0
FATAL	100
ERROR	200
WARN	300
INFO	400
DEBUG	500
TRACE	600
ALL	Integer.MAX_VALUE

Levels
Wartości niestandardowe

Istnieje możliwość stworzenia własnego, niestandardowego poziomu logów:

```
Level VERBOSE = Level.forName("VERBOSE", 550);

logger.log(VERBOSE, "This is VERBOSE log.");
```

Zadanie 3.2:

1. Włącz debug log4j żeby zobaczyć co dzieje się “wewnątrz” mechanizmu logowania (log4j2.debug)

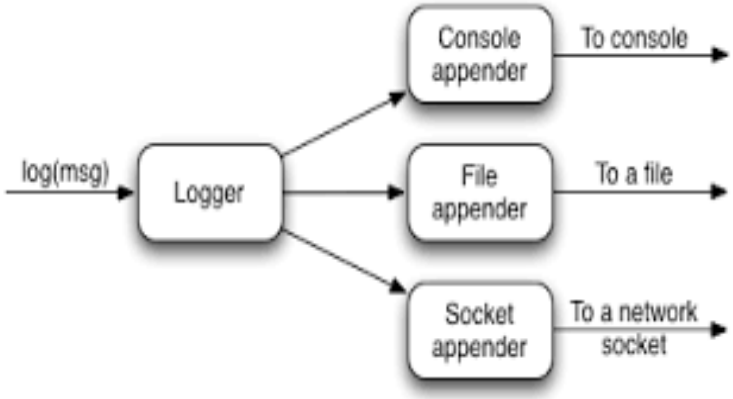
log4j2
Appenders

Do tej pory używaliśmy tylko konsoli jako miejsca do którego wysyłane są wszystkie logi. Log4j2 posiada wiele innych możliwości, nazywanych **appenderami**:

ConsoleAppender, AsyncAppender, FailoverAppender, FileAppender, FlumeAppender, JDBCAppender, JMSAppender, JPAAppender, MemoryMappedFileAppender, NoSQLAppender, OutputStreamAppender, RandomAccessFileAppender, RewriteAppender, RollingFileAppender, RollingRandomAccessFileAppender, RoutingAppender, SMTPAppender, SocketAppender, SyslogAppender

log4j2

Appenders Przychodzi wiadomość do logera i co dalej?



Zadanie 3.3:

1. Zmodyfikuj konfiguracje log4j2 i dodaj appander plikowy zamiast istniejącego appendera konsolowego. Zweryfikuj działanie. Wykorzystaj dokumentację log4j2 (https://logging.apache.org/log4j/2.x/manual/appenders.html)
2. Skonfiguruj dwa appendery – plikowy i konsolowy. Zweryfikuj działanie.

log4j2
Filters

Nawet w przypadku istnienia kandydata do obsłużenia danego zdarzenia logowania, istnieje możliwość dokonfigurowania odrzucenia zdarzenia. Realizuje się takie zachowanie z pomocą filtrów.

log4j2
Filters

Nawet w przypadku istnienia kandydata do obsłużenia danego zdarzenia logowania, istnieje możliwość dokonfigurowania odrzucenia zdarzenia. Realizuje się takie zachowanie z pomocą filtrów.

Przykładowo: do konsoli chcemy logować każde zdarzenie jakie wystąpi w czasie działania aplikacji jednak w bazie danych/pliku chcemy zacząć “widzieć” dane zdarzenie tylko gdy wystąpiło więcej niż 10 razy w ciągu minuty (BurstFilter)

Przykłady Filtrów

ThresholdFilter – filtr który akceptuje logi o zadanym poziomie lub bardziej szczegółowym. (Np. <ThresholdFilter level="INFO"/> zignoruje wpisy z poziomu DEBUG, ale “przepuści” wpisy INFO i wyższe (np. ERROR))

TimeFilter – filtr, który akceptuje logi z określonej pory dnia

Script – wykonuje skrypt, akceptacja logu jest w zależności od zwróconego true/false

RegexFilter – konfrontuje wiadomość z regexem, dopasowuje te wiadomości, które pasują do wzoru

lokalizacja Filtrów

- Context-wide Filters
- Logger Filters
- Appender Filters
- Appender Reference Filters

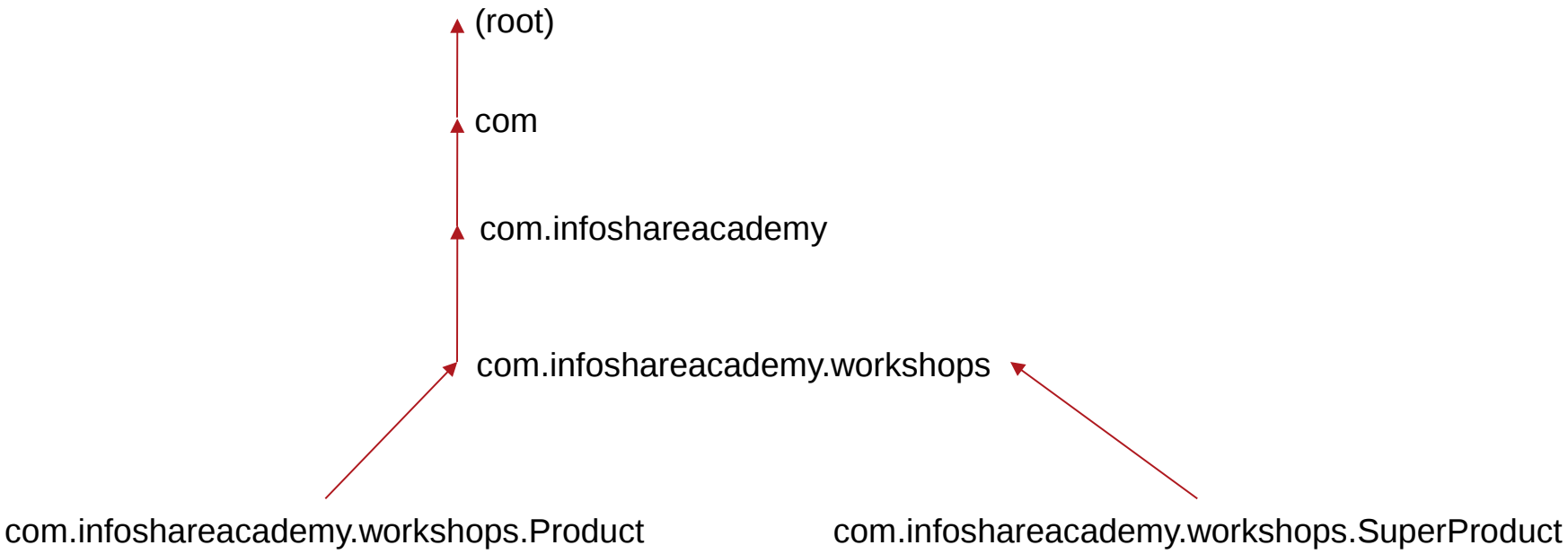
Zadanie 3.4:

1. Zmodyfikuj konfiguracje log4j2 i dodaj filtry do obu apenderów:
   Apender Konsolowy – wyświetla wszystko
   Apender Plikowy – zapisuje do pliku tylko level WARN lub wyższy.
2. Zweryfikuj działanie.

Logger

- jest obiektem Javy
- ma swoją nazwę (name)
- do jednego loggera może istnieć wiele referencji (np. gdy istnieje wiele obiektów tej samej klasy)
- ma swoją konfigurację
- **konfiguracja loggera X jest zależna od konfiguracji innych logerów których nazwa stanowi prefix nazwy logera X (wyższych w hierarchi)**

Logger
Nazewnictwo i hierarchia



level inheritance

```
<Loggers>
  <logger name="root" level="error"/>
</Loggers>
```

logger	Effective level
root	error
com	error
com.isa	error
com.isa.MyClass	error

https://logging.apache.org/log4j/2.x/manual/architecture.html

level inheritance

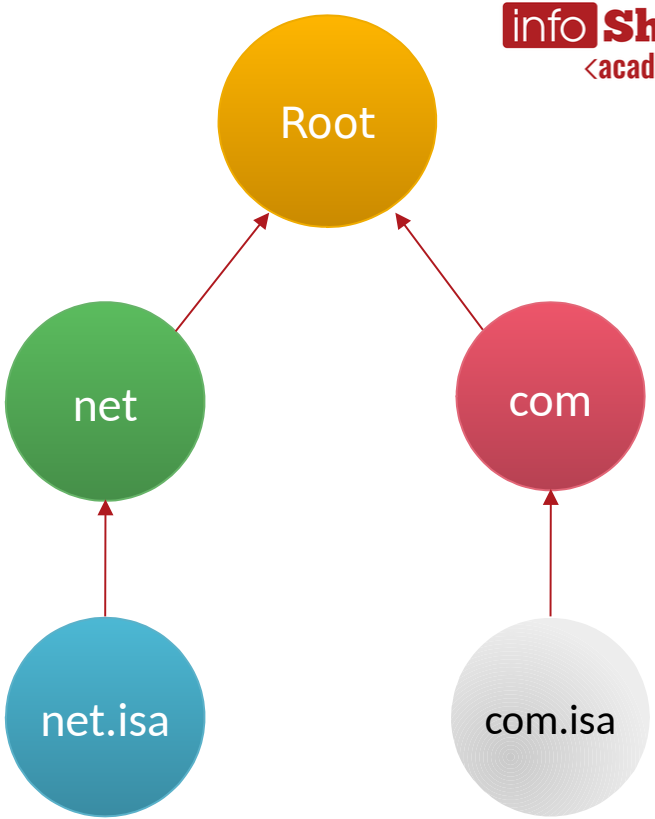
```
<Loggers>
  <logger name="root" level="error"/>
  <logger name="com.isa" level="warn"/>
</Loggers>
```

logger	Effective level
root	error
com	error
com.isa	warn
com.isa.MyClass	warn

Appender Additive w hierarchii logerów

Domyślnie, log4j2 jest addytywne. Oznacza to, że apendery skonfigurowane dla nadrzędnych loggerów również zostaną użyte.

additivity=false – żeby wyłączyć



Apender additivity

Each enabled logging request for a given logger will be forwarded to all the appenders in that Logger's LoggerConfig as well as the Appenders of the LoggerConfig's parents.

= Appenders are inherited additively from the LoggerConfig hierarchy

Apender additivity

Each enabled logging request for a given logger will be forwarded to all the appenders in that Logger's LoggerConfig as well as the Appenders of the LoggerConfig's parents.

= Appenders are inherited additively from the LoggerConfig hierarchy

DEMO:
demo/com/isa/workshops/log4j/additivity/AdditivityLoggerExample.java

log4j2
Layouts

Appender używa Layout’ów do sformatowania wiadomości loga w taki sposób aby pasował on do wyjściowego mechanizmu składania logów.

Layouts
Pattern

Realizowane za pomocą: PatternLayout
Elastyczny layout konfigurowany za pomocą kombinacji znaków przypominających wyrażenia regularne.

Wyrażenie zapisujemy w parametrze „pattern”.

Opis wszystkich dozwolonych opcji:
https://logging.apache.org/log4j/2.0/manual/layouts.html

Pattern
Przykład

Konfig.:

```
<PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
```

Użycie:

```
logger.debug("one={}, two={}, three={}", 1, 2, 3);
```

Doprowadzi do wyjścia:

```
20:10:54.488 [main] DEBUG pattern-layout - one=1, two=2, three=3
```

Layouts
CSV

Realizowane za pomocą: CsvParameterLayout
Tworzy log w formacie CSV.

DEMO:
demo/src/main/java/com/isa/workshops/log4j/layouts/CSVLayouts.java

Layouts
JSON

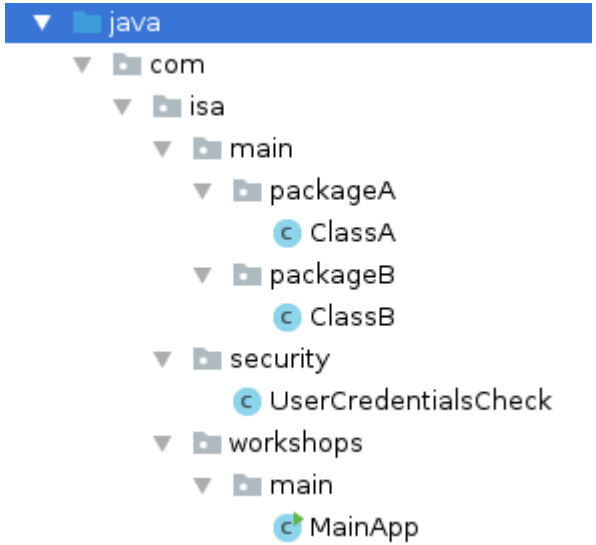
Realizowane za pomocą: JsonLayout
Wyjście formatowane do struktury JSON.

DEMO:
demo/src/main/java/com/isa/workshops/log4j/layouts/JsonLayouts.java

Wybrane parametry:
compact = jeśli true, to appender nie użyje żadnych znaków nowej linii jak i wcięć

includeStacktrace – jeśli true, przy każdym logowaniu Throwable będzie zalogowany stack trace (def true)

Demo 2


java
com
isa
main
packageA
ClassA
packageB
ClassB
security
UserCredentialsCheck
workshops
main
MainApp

Pytania?