Notatki - Logowanie - Intro

Spis treści

Czym jest logowanie?

Logowanie (**Logging**) jest procesem zapisywania przebiegu wykonania aplikacji w jakimś miejscu. Przebieg działania aplikacji może być drukowany na ekranie w trakcie działania aplikacji lub zapisywany do pliku - mówimy wtedy o logach zapisanych w pliku. Zapisywanie przebiegu na ekranie nie jest w żaden sposób trwałe, bo jak aplikacja przestanie działać i wyłączymy komputer, to tracimy te zapisy. Ewentualnie jeżeli uruchomimy aplikację ponownie to również stracimy zapis z poprzedniego wykonania, który został wydrukowany na ekranie. Jeżeli chodzi o zapis do plików - jest on trwały, przynajmniej do momentu usunięcia takiego pliku 😁.

Logowanie w rozumieniu opisanym powyżej jest czym innym niż logowanie na Facebooka, czyli jest to co innego niż podanie użytkownika i hasła. Niefortunna zbieżność nazw...

Po co się to robi?

Wyobraź sobie, że uruchamiasz program, który liczy coś przez bardzo długi czas. Zostawiasz taki program na noc. W jaki sposób można później prześledzić jego wykonanie w celu ewentualnej diagnozy jakiś błędów? Najlepiej jest zapisać przebieg takiego programu w jakiś trwały sposób, np. do pliku tekstowego.

W praktyce sytuacja staje się o wiele bardziej skomplikowana, gdyż takie pliki z logami muszą być przetrzymywane w jakimś centralnym miejscu (aby móc łatwo znaleźć zapis przebiegu aplikacji z danego dnia i godziny), do tego również powinniśmy mieć możliwość znalezienia w łatwy sposób interesującej nas akcji. To prowadzi do wielu zagadnień, których nie będziemy tutaj poruszać. Skupimy się na tym, co trzeba zrobić, aby nasza aplikacja mogła logować, czyli zapisywać przebieg swojego

działania.

Dostępne rozwiązania

W praktyce spotkasz się z bibliotekami, które zapewniają nam obiekty i metody, pozwalające w prosty sposób określić konfigurację logowania. Biblioteki takie będą również realizowały samą kwestię logowania, czyli zapisywania przebiegu działania aplikacji. Z racji dostępności wielu bibliotek, powstały również narzędzia będące warstwą abstrakcji nad tymi bibliotekami, aby umożliwić prostą wymianę takiej biblioteki pod spodem, jeżeli okaże się, że wynikła w naszym projekcie taka potrzeba.

W tym miejscu należy dodać, że teoretycznie możliwe jest aby takie logowanie realizować w sposób "manualny", "ręcznie", "po swojemu". Wrócimy do tego tematu na jak już poznamy dostępne możliwości.

Framework

Zacznijmy od frameworków i ich nazw, które możesz spotkać na swojej drodze. Zanim jednak to nastąpi wyjaśnijmy różnicę.

Czym różni się library od framework? Klik

- **library** - zestaw funkcji/metod, które można wywołać, które są zorganizowane w postaci klas. Po każdym wywołaniu funkcji z biblioteki to my przejmujemy kontrolę nad dalszym wywołaniem kodu.
- **framework** - jest abstrakcyjnym projektem, który posiada wbudowane zachowania. Aby go używać, wstawiamy swoje własne zachowania/metody w różne miejsca we frameworku np. przez tworzenie własnych klas i wstawianie ich we framework. Następnie kod frameworka wykonuje nasz kod w miejscach, które uzupełniliśmy.

Takie są definicje, natomiast w praktyce jest to używane wymiennie, albo framework mówi się na biblioteki, które są "duże" ☺.

Tutaj znajdziesz link do Wikipedii, gdzie zostały wymienione frameworki do logowania w Java. Służą one do ułatwiania i standaryzacji procesu logowania. Przejdźmy zatem do najważniejszych i najbardziej popularnych z nich.

java.util.logging

Paczka `java.util.logging` zapewnia podstawowe klasy i interfejsy dające możliwość logowania w aplikacji. Są dostępne razem z JDK od Javy w wersji 1.4.

Log4j

Log4j jest bardzo znanym frameworkiem, który można często spotkać w przeróżnych projektach. Zostało jednak ogłoszone, że 5 Sierpnia 2015 `log4j` doszedł do końca swojej przygody i zaleca się update do jego następcy **Log4j 2**. Link

Logback

Początkowo Logback miał być zastępcą `Log4j`. Jest szybszy niż `Log4j`. Zapewnia również wsparcie dla

kontenerów (servlet containers) takich jak `Tomcat` lub `Jetty` (niżej uwaga do tego). Daje również możliwość kompresowania logów (jeżeli mamy pliki z logami, możemy je potem spakować do plików, np. `.zip`). Zapewnia również wbudowaną integrację z **SLF4J**, o którym powiemy zaraz. Składa się z 3 modułów:

- **logback-core** – moduł zawiera podstawowe funkcje logowania,
- **logback-classic** – moduł zawiera m.in. wsparcie `slf4j`,
- **logback-access** – moduł zapewnia integrację z servlet containers (na razie się tym nie martw).

Wiem, że jeszcze nie wiadomo co to Tomcat albo Jetty. Wiem, że brzmi to jak abstrakcja. Spokojnie, będzie w swoim czasie. W skrócie powiem, że będzie nam to potrzebne jak przejdziemy do pisania aplikacji, które dają możliwość komunikacji przez internet.

Pod tym linkiem znajdziesz stronę projektu i dokumentację.

Log4j 2

Log4j 2 jest nowszą wersją swojego poprzednika `Log4j`. Zapewnia ulepszenia, które również zapewnia `logback`. Naprawia również niektóre problemy, które występują w `logback`. Strona projektu

SLF4J

To jest bardzo ważny skrót od słów **Simple Logging Facade**. `SLF4J` jest warstwą abstrakcji, która może zostać "nałożona" na rozwiązania wspomniane wcześniej. Czyli inaczej mówiąc `SLF4J` może być naszym pośrednikiem w użyciu frameworków do logowania. Pozwala to nam jako użytkownikom określić w konfiguracji, którego z frameworków do logowania chcemy używać, natomiast w kodzie stosować warstwę abstrakcji, którą daje nam `slf4j`. Jeżeli w pewnym momencie zdecydujemy, że chcemy zmienić framework do logowania, który jest "pod spodem", nie musimy nic zmieniać w kodzie. Wystarczy, że zmienimy konfigurację projektu i załatwione. Pod tym linkiem znajdziesz stronę projektu i dokumentację.

Jak żyć?

Jak zdecydować co wybrać? To zależy. Od tego, czy trafiamy na nowy projekt, czy może na taki, gdzie ta decyzja została już podjęta.

Jeżeli trafimy do projektu, w którym stosowane są starsze rozwiązania do logowania, warto zastanowić się nad wprowadzeniem warstwy abstrakcji, którą daje nam `slf4j`, a następnie po ujednoliceniu tego w kodzie, nad przepięciem frameworka do logowania pod spodem.

Jeżeli pracujemy nad nowym projektem i musimy wybrać rozwiązanie do logowania, możemy wybrać `logback` lub `log4j 2` i nałożyć na to warstwę abstrakcji w postaci `slf4j`.

Logowanie w Javie

Przejdźmy do omówienia kluczowych klas, które są używane w procesie logowania.

Logger

`Logger` jest obiektem używanym w kodzie do wywoływania metod logujących. Obiekt ten jest z reguły definiowany dla każdej z klas, w których chcemy logować. Dzięki temu mamy zachowany kontekst, która klasa w projekcie dokonała danego zapisu, czyli wyprodukowała daną linijkę w logach. Konfiguracja logowania w aplikacji polega na dodaniu odpowiedniego frameworka do projektu i zdefiniowaniu loggera w klasach, z których chcemy zapisać informację w logach. Definiujemy taki `Logger` jak poniżej, w każdej klasie, z której chcemy logować. Później poznamy sposób jak to skrócić.

```
package pl.zajavka;

import java.util.logging.Logger;

public class LoggingExample {

    private static final Logger logger = Logger.getLogger(LoggingExample.class.getName());
}
```

Handler

`Handler` jest używany do eksportowania logów do interesującego nas miejsca docelowego. Takim miejscem docelowym może być konsola aplikacji, plik na dysku lub jakaś lokalizacja w internecie, z którą musimy się połączyć zdalnie. Możemy wyróżnić dwa standardowe handlery:

- **ConsoleHandler** - drukuje logi na konsolę przy wykorzystaniu `System.err`. Inaczej mówi się o tym, że logi są wyrzucane na wyjście standardowe,
- **FileHandler** - umożliwia zapis logów do pliku.

Handlers możemy włączać lub wyłączać za pomocą konfiguracji. Czyli możemy logować tylko w konsoli, tylko w pliku, albo w obu itp. Oczywiście robimy to przy wykorzystaniu konfiguracji danego frameworka do logowania. Domyślną implementację `ConsoleHandler` umieszczam poniżej. Możemy również zdefiniować własny `Handler` jeżeli będzie nam to potrzebne. Najczęściej stosowane są te standardowe.

Kod źródłowy klasy ConsoleHandler

```
public class ConsoleHandler extends StreamHandler {

    public ConsoleHandler() {
        // configure with specific defaults for ConsoleHandler
        super(Level.INFO, new SimpleFormatter(), null); ①

        setOutputStreamPrivileged(System.err); ②
    }

    @Override
    public void publish(LogRecord record) {
        super.publish(record);
        flush();
    }

    @Override
```

```
    public void close() {
        flush();
    }
}
```

① Ustawiany jest jakiś poziom `Level.INFO`, o tym będziemy niedługo rozmawiać.

② Zwróć uwagę na wykorzystanie `System.err`.

A dlaczego nie zostać przy System.out.println()?

Możesz zadać sobie teraz pytanie, po co to wszystko. Przecież stosowaliśmy wcześniej `System.out.println()`, informacje drukowały się w konsoli i wszystko było w porządku. Dowiedzieliśmy się też jak zapisywać informacje do plików, to po co teraz wymyślamy jakieś frameworki do logowania. Dlaczego nie robimy tego samodzielnie? Powodów jest kilka:

- Musielibyśmy pisać kod do realizacji procesu logowania sami, a po co to robić jeżeli mamy już gotowe biblioteki do tego,
- Jeżeli mamy gotową bibliotekę do robienia czegoś, to warto jest ją zastosować zamiast wymyślać koło na nowo, chociażby z tego powodu, że biblioteka taka rozwiązała już problemy, na które i tak byśmy się natknęli sami. Po zo zatem tracić na to czas,
- Logowanie potrafi znacząco spowolnić aplikacje, bo przecież zapis do pliku zajmuje czas. Jest to dodatkowe wywołanie kodu i wykonanie jakiejś czynności. Dlatego kolejne rozwiązania starają się być szybsze niż poprzednie,
- W pewnym momencie pojawi się taki problem, że nie wszystkie wiadomości chcemy mieć zapisane w pliku, wiadomości w pewnym momencie zaczęłyby mieć priorytet, np. błędy chcemy logować bardzo często (w praktyce zawsze), ale informacje dotyczące normalnego przebiegu programu mogą nas interesować tylko wybiórczo. Pojawi nam się zatem temat priorytetów logowanych wiadomości. W tym właśnie celu stosowane są poziomy logowania, o których niżej.
- Jeżeli będziemy chcieli po prostu wyłączyć zapisywanie informacji na ekranie podczas działania aplikacji, to stosując `System.out.println()` trzeba się trochę pobawić żeby zrobić to w prosty sposób.