Logging

18 Apr 2023

Что такое Logging?

- Процесс сбора и хранения данных, сгенерированных приложением
- Пример данных: от сообщений об ошибках до метрик приложения
- Цель – получить информацию, которую можно использовать для диагностики и устранения проблем в ПО, анализа его работы

Типы логов

- Логи приложения
  - Поведение приложения
  - Ошибки/исключения
  - Действия пользователей
- Системные логи
  - Информация про ОС, оборудование, использование памяти и т.д.
- Секьюрити логи
  - Логи аутентификации/авторизации

Уровни логгирования

- В зависимости от степени «серьёзности», логи делят на несколько уровней: TRACE, DEBUG, INFO, WARN, ERROR, FATAL/CRITICAL.
  - **TRACE** – слишком большая детализация информации, вплоть до вызова каждого метода. Помогают в диагностике сложных проблем, которые сложно воспроизвести
  - **DEBUG** – отладочная информация
  - **INFO** – обычное поведение приложения, «дежурная» информация
  - **WARN** – потенциальные проблемы, неожиданное поведение
  - **ERROR** – ошибка, влияет на поведение приложения, нужно срочно обратить внимание, сбой работы приложения

Как это выглядит?

```
import java.util.logging.Logger;

Logger logger = Logger.getLogger(MyClass.class.getName());
logger.trace("This is a trace message");
logger.debug("This is a trace message");
logger.info("This is an information message");
…
try {
…
} catch (Exception ex) {
   logger.error("Some error occurred!");
   throw new RuntimeException(ex);
}
```

Как это выглядит в консоли?

```
2022-04-08 08:42:57,458 ERROR [credit-client-pool-68]
{da7b76be-7a98-4a2f-9758-4b3df818d903}
Could not calculate score for PremiumBankingAccount[id=2001000, ...]
ru.hse.java.client.ClientException: java.util.concurrent.ExecutionException:
io.grpc.StatusRuntimeException: {FAILED_PRECONDITION} Blahblahblah...
        at ClientBase.a(ClientBase.java:13)
        at CreditClient.score(CreditClient.java:438)
Caused by: java.util.concurrent.ExecutionException: io.grpc.StatusRuntimeException:
{FAILED_PRECONDITION} Blahblahblah...
        at ClientBase.c(ClientBase.java:23)
        at ClientBase.b(ClientBase.java:17)
        at ClientBase.a(ClientBase.java:11)
        ... 1 more
Caused by: io.grpc.StatusRuntimeException: {FAILED_PRECONDITION} Blahblahblah...
        at ClientBase.e(ClientBase.java:30)
        at ClientBase.d(ClientBase.java:27)
        at ClientBase.c(ClientBase.java:21)
        ... 3 more
```

Почему не подойдет *println()*?

```
var principal = ...;
try (var creditClient = ...) {
    return creditClient.score(principal);
} catch (ClientException e) {
    System.err.println(java.time.Instant.now() + " "
        + "ERROR "
        + "[" + Thread.currentThread().getName() + "] "
        + "{" + MyMagicContext.getRequestId() + "} "
        + "Could not calculate score for " + principal);
    e.printStackTrace(); // Defaults to System.err output
    return DEFAULT_CREDIT_SCORE;
}
```

Copy-paste, дублирование кода, риск допустить ошибку, заново изобретаем «колесо»

Лучшие практики

- Чёткие, краткие, полезные, осмысленные сообщения в логах
- Избегайте ненужных логов
  - Слишком много логов – большой расход памяти, медленная работа файлов с логами
- Разделение логов по источнику/приложению
  - В большом проекте с множеством модулей это необходимо, поможет в решении проблем
- Периодическое архивирование логов
  - Со временем логи занимают много места на диске, хорошая практика – их архивировать (не удалять хотя бы в течение времени X)
- Используйте фреймворки. Log4j, Logback, SLF4J. Встроенный механизм в Java может не покрывать все use cases
  - Более гибкие и производительные, могут настраивать логгирование по модулям и т.д.

Популярные фреймворки

- **Log4j 2.x**
  - Можно отправлять логи в разные destinations (файлы, БД, email)
  - Конфигурируемые уровни логгирования
  - Кастомизированное форматирование логов – сами указываете нужный формат логов
  - Производительность – легко работает с большим объёмом логов
  - **Log4j 1.x: Legacy Version, Do Not Use in New Code!**
- **Logback**
  - Разработан тем же разработчиком, что и Log4J
  - Более современный и гибкий, чем предшественник
  - Больше продвинутых настроек для логгирования
- **SLF4J** (Simple Logging Facade for Java)
  - logging API – facade над фреймворками, включая Log4j и Logback
  - Позволяет разработчикам легко переключаться между фреймворками
  - Возможности те же, что и у других фрейморков