Logger

Java Collections
4 уровень, 9 лекция

ОТКРЫТА

— А, вот ты где! Ты не забыл, что у нас сегодня еще одна лекция?

— Нет, я как раз тебя искал. Почти…

— Отлично, тогда начнем. Сегодня я хочу рассказать тебе про логирование.

Лог – это список произошедших событий. Почти как «морской журнал» или «дневник». Ну, или Твиттер – кому что ближе. А, соответственно, логгер — это объект, с помощью которого можно вести логирование.

В программировании принято логировать практически все. А в Java – так вообще все и даже немного больше.

Дело в том, что Java-программы – это очень часто большие серверные приложения без UI, консоли и т.д. Они обрабатывают одновременно запросы тысяч пользователей, и нередко при этом возникают различные ошибки. Особенно, когда разные нити начинают друг другу мешать.

И, фактически, единственным способом поиска редко воспроизводимых ошибок и сбоев в такой ситуации есть запись в лог/файл всего, что происходит в каждой нити.

Чаще всего в лог пишется информация о параметрах метода, с которыми он был вызван, все перехваченные ошибки, и еще много промежуточной информации.

Чем полнее лог, тем легче восстановить последовательность событий и отследить причины возникновения сбоя или ошибки.

Иногда логи достигают нескольких гигабайт в сутки. Это нормально.

— Нескольких гигабайт? O_o

— Ага. Чаще всего при этом, лог-файлы автоматически архивируются, с указанием дня – за какой день это архив с логом.

— Ничего себе.

— Ага. Изначально в Java не было своего логгера, что привело к написанию нескольких независимых логгеров. Самым распространенным из них стал log4j.

Спустя несколько лет, в Java все же был добавлен свой логгер, но его функциональность была гораздо беднее и большого распространения он не получил.

Факт, как говорится, на лицо – в Java есть официальный логгер, но все сообщество Java-программистов предпочитает пользоваться другим.

На основе log4j потом было написано еще несколько логгеров

НАЧАТЬ ОБУЧЕНИЕ

А затем для них всех был написан специальный универсальный логгер slf4j, который сейчас повсеместно используют. Он очень похож на log4j, поэтому я расскажу тебе логирование на его примере.

Весь процесс логирования состоит из трех частей.

Первая часть – это сбор информации.

Вторая часть – это фильтрование собранной информации.

Третья часть – это запись отобранной информации.

Начнем со сбора. Вот типичный пример класса, который ведет лог:

Класс с логированием

```
1  class Manager
2  {
3   private static final Logger logger = LoggerFactory.getLogger(Manager.class);
4
5   public boolean processTask(Task task)
6   {
7    logger.debug("processTask id = " + task.getId());
8    try
9    {
10    task.start();
11    task.progress();
12    task.complete();
13    return true;
14   }
15   catch(Exception e)
16   {
17    logger.error("Unknown error", e);
18    return false;
19   }
20  }
21 }
```

Обрати внимание на слова, выделенные красным.

Строка 3 – создание объекта logger. Такой статический объект создают практически в каждом классе! Ну, разве что кроме классов, которые ничего не делают, а только хранят данные.

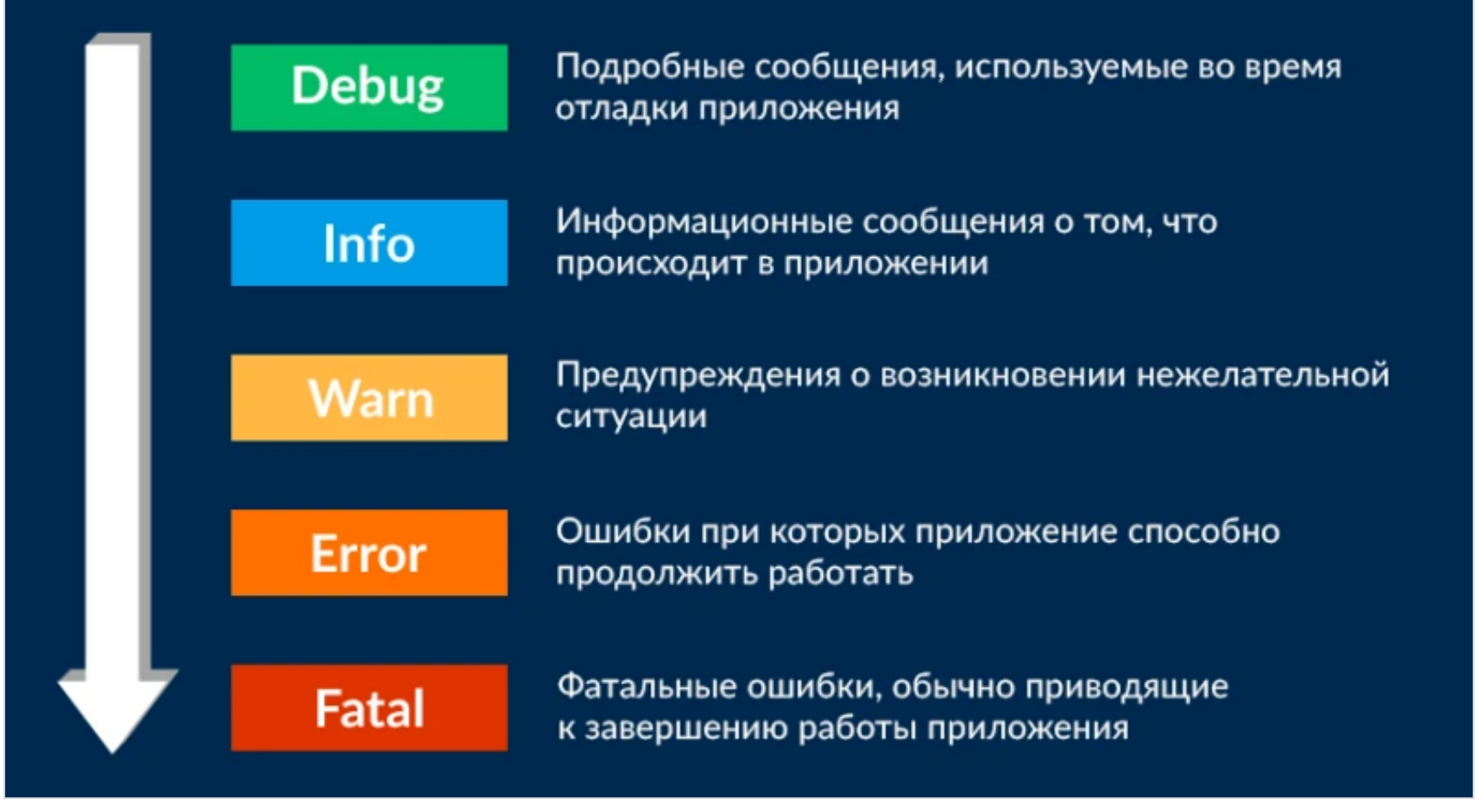
LoggerFactory – это специальный класс для создания логгеров, а getLogger – это его статический метод. В него обычно передают текущий класс, хотя возможны различные варианты.

Строка 7 – в логгер пишется информация о вызове метода. Обрати внимание – это первая строчка метода. Только метод вызвался – сразу пишем информацию в лог.

Мы вызываем метод debug, это значит, что важность информации «уровня DEBUG». Этот факт используется на уровне фильтрации. Об этом я расскажу через пару минут.

Строка 17 – мы перехватили исключение и… сразу же записали его в лог! Именно так и нужно делать.

На этот раз мы вызываем метод error, что сразу придает информации статус «ERROR»



— Пока вроде все ясно. Ну, насколько это может быть ясно в середине разговора.

— Отлично, тогда перейдем к записи фильтрации.

Обычно, у каждого лог-сообщения есть своя степень важности, и, используя ее можно часть этих сообщений отбрасывать. Вот эти степени важности:

Степень важности	**Описание**
ALL	Все сообщения
TRACE	Мелкое сообщение при отладке
DEBUG	**Сообщения важные при отладке**
INFO	**Просто сообщение**
WARN	**Предупреждение**
ERROR	**Ошибка**
FATAL	**Фатальная ошибка**
OFF	Нет сообщения

Эти уровни используются еще и при отсеве сообщений.

Скажем, если выставить уровень логирования в WARN, то все сообщения, менее важные, чем WARN будут отброшены: TRACE, DEBUG, INFO.

Если выставить уровень фильтрации в FATAL, то будут отброшены даже ERROR’ы.

Есть еще два уровня важности, которые используются при фильтрации – это OFF – отбросить все сообщения и ALL – показать все сообщения (не отбрасывать ничего).

— А как настраивать фильтрацию и где?

— Сейчас расскажу.

Обычно настройки логгера log4j задаются в файле log4j.properties.

представить в виде воды.

Вот тебе несколько примеров:

Запись лога в консоль

```
1  # Root logger option
2  log4j.rootLogger=INFO, stdout
3
4  # Direct log messages to stdout
5  log4j.appender.stdout=org.apache.log4j.ConsoleAppender
6  log4j.appender.stdout.Target=System.out
7  log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
8  log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss}
```

Строки 1 и 4 – это комментарии

Строка 2 – мы указываем уровень сообщений, которые оставляем. Все менее важные уровни будут отброшены (DEBUG,TRACE)

Там же, через запятую, мы указываем имя объекта (сами придумываем), куда будет писаться лог. В строках 5-8 идут его настройки.

Строка 5 – указываем тип апендера – консоль (ConsoleAppender).

Строка 6 – указываем, куда именно будем писать – System.out.

Строка 7 – задаем класс, который будет управлять шаблонами записей – PatternLayout.

Строка 8 – задаем шаблон для записи, который будет использоваться. В примере выше это дата и время.

А вот как выгладит запись в файл:

Запись лога в файл

```
1  # Root logger option
2  log4j.rootLogger=INFO, file
3
4  # Direct log messages to a log file
5  log4j.appender.file=org.apache.log4j.RollingFileAppender
6  log4j.appender.file.File=C:\\loging.log
7  log4j.appender.file.MaxFileSize=1MB
8  log4j.appender.file.MaxBackupIndex=1
9  log4j.appender.file.layout=org.apache.log4j.PatternLayout
10 log4j.appender.file.layout.ConversionPattern= %-5p %c{1}:%L - %m%n
```

Строка 2 задает уровень фильтрации сообщений и имя объекта-апендера (стока).

Строка 5 – указываем тип апендера – файл (RollingFileAppender).

Строка 6 – указываем имя файла – куда писать лог.

Строка 7 – указываем максимальный размер лога. При превышении размера, начнет писаться новый файл.

Строка 8 – указываем количество старых файлов логов, которые надо хранить.

Строки 9-10 – задание шаблона сообщений.

— Я не знаю, что тут происходит, но догадываюсь. Что не может не радовать.

Запись лога на консоль и в файл

```
# Root logger option
log4j.rootLogger=INFO, file, stdout

# Direct log messages to a log file
log4j.appender.file=org.apache.log4j.RollingFileAppender
log4j.appender.file.File=C:\\loging.log
log4j.appender.file.MaxFileSize=1MB
log4j.appender.file.MaxBackupIndex=1
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern= %-5p %c{1}:%L - %m%n

# Direct log messages to stdout
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss}
```

— Ага, оказывается и так можно? Это же отлично!

— Ага. Ты можешь объявить сколько угодно апендеров и настроить каждый из них по-своему.

Более того, каждому апендеру можно очень гибко настроить фильтр его сообщений. Мы можем не только задать каждому апендеру свой уровень фильтрации сообщений, но и отфильтровать их по пакетам! Вот для чего надо указывать класс при создании логгера (я про **LoggerFactory.getLogger**).

Пример:

Запись лога на консоль и в файл

```
# Root logger option
log4j.rootLogger=INFO, file, stdout

# Direct log messages to a log file
log4j.appender.file=org.apache.log4j.RollingFileAppender
log4j.appender.file.threshold=DEBUG
log4j.appender.file.File=C:\\loging.log
log4j.appender.file.MaxFileSize=1MB
log4j.appender.file.MaxBackupIndex=1
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern= %-5p %c{1}:%L - %m%n

# Direct log messages to stdout
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.threshold=ERROR
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss}

log4j.logger.org.springframework=ERROR
log4j.logger.org.hibernate=ERROR
log4j.logger.com.javarush=DEBUG
log4j.logger.org.apache.cxf=ERROR
```

Строки 20-23 мы указываем имя пакета и тип фильтрации его сообщений. «log4j.logger» — это префикс, имя пакета выделено оранжевым.

— Ничего себе? Даже так можно. Ну, круто!

— Кстати, ни log4j, ни slf4j не входят в JDK, скачивать их надо отдельно. Это можно сделать вот тут. Но есть и второй способ:

Шаг 1. Добавляешь в класс импорты:

```
1  import org.slf4j.Logger;
2  import org.slf4j.LoggerFactory;
```

Шаг 2. Становишься курсором на эти строчки и нажимаешь Alt+Enter в Intellij IDEA

Шаг 3. Выбираешь пункт File jar on web.

Шаг 4. Выбираешь – slf4j-log4j13.jar

Шаг 5. Указываешь, куда скачать библиотеку (jar)

Шаг 6. Пользуешься нужными тебе классами.

‹ Предыдущая лекция

Следующая лекция ›

— Ничего себе! Да что же сегодня за день-то такой. Столько нового и столько классного!

+78

Комментарии (50) + 1 — популярные — новые — старые

JavaCoder

Введите текст комментария

Станислав Future Уровень 38, Москва, Russian Federation — позавчера, 17:43

Ладно, все. Иди отдыхай, программист.

Ответить 0

Бостон Уровень 26, Russian Federation — 8 июня, 22:30

В последних 2 примерах если прописать log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} - на консоль ничего не будет выводится кроме даты сообщения. Для нормального вывода везде нужно указывать паттерн: **%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n**

Ответить +3

Андрей Гузанов Уровень 37, Новосибирск, Россия — 15 ноября 2021, 14:07

Полезная статья - как подключить Maven зависимость Slf4j и log4j .

Так же рекомендую посмотреть видео:
1) Loggers
2) Часть 1 - Головач log4j Slf4j
3) Часть 2 - Головач log4j Slf4j
4) И все остальные лекции Головача на эту тему

Ответить +4

Pineapple Уровень 45, Абакан, Россия — 8 августа 2021, 15:55

куча комментариев, но так и не понятно как подтянуть все это через мавен
и не просто обертку, а что бы все работало..

Ответить 0

Не совсем понятно, что вам не понятно. Что значит подтянуть(Добавить зависимость в pom.xml/build.gradle/build.xml etc.?)?

Ответить

0

ilya Уровень 40 19 июля 2021, 23:30

String log4jConfPath
="/...../JavaRushTasks/4.JavaCollections/src/com/javarush/task/task34/task3412/log4j.properties";
PropertyConfigurator.configure(log4jConfPath);

Ответить

0

Матвей Уровень 28 15 июля 2021, 16:17

"Я не знаю, что тут происходит, но догадываюсь." - слоган многих лекций курса коллекций

Ответить

+9

Денис Кайдунов Уровень 38, Гомель, Беларусь 5 мая 2021, 00:55

https://mkyong.com/logging/log4j-hello-world-example/ просто и понятно по шагам

Ответить

+3

Dmitry Gidlevsky Уровень 35, Киев, Украина 25 марта 2021, 01:21

Сэкономлю вам много времени на ошибках в IDEA)
Что нужно качать для разных целей

Ответить

+3

Anonymous #2491313 Уровень 35 19 февраля 2021, 09:42

>> был написан специальный универсальный логгер slf4j, который сейчас повсеместно используют

Именно поэтому мы его не будем разбирать, а разберём log4j
БРАВО!

Ответить

+12

Valua Sinicyn Уровень 41, Харьков, Украина 10 февраля 2021, 14:49

Амиго, по факту, задрот, все ему понятно...

Ответить

+1

Показать еще комментарии

Русский

СКАЧИВАЙТЕ НАШИ ПРИЛОЖЕНИЯ

ДОСТУПНО В Google Play

Загрузите в App Store