Программирование. Вывод в лог

Software development. Logging

Ключевые слова: логирование – application logging; журналирование – ; программирование – computer programming.

Статья освещает вопросы программирования вывода в лог. В работе описываются возможности библиотек логирования, приводятся правила и рекомендации по написанию программного кода, осуществляющего вывод в лог.

The article highlights aspects of proper application logging. It describes the capabilities of logging frameworks, provides guidelines for writing software code with clean and helpful logs.

БАШКИНА / BASHKINA I.
Ирина Александровна
(irb2910@gmail.com)
преподаватель кафедры Информатики и математики,
СПбГУКИ,
Санкт-Петербург

ВВЕДЕНИЕ

Программирование на сегодняшний день – это огромная индустрия; для успешной работы в ней необходимо множество составляющих, начиная от знаний языков программирования, заканчивая налаженными механизмами поставок готового программного продукта. Рассмотрим небольшую часть, зачастую несправедливо воспринимающейся большинством как второстепенную, – логирование.

ПРЕДНАЗНАЧЕНИЕ ЛОГИРОВАНИЯ

Логирование – это вывод сообщений о ходе выполнения программы в лог файл. Пример типичного сообщения в логе:

```
2013-05-05 21:26:44 DEBUG [my.examples.Sort] (main) - Populating
an array of 256 elements in descending order
```

Анализ данных сообщений позволяет определить состояние системы, определить цепь событий, предшествующих этому состоянию. Логирование широко применяется в следующих ситуациях:

– мониторинг возникновения ошибок;

– анализ причин падения программы, аварийное завершение или зависание;

– анализ проблем безопасности, например, неавторизованного доступа;

– мониторинг характера потребления ресурсов и нагрузки на подсистемы.

Современные системы логирования также предоставляют дополнительные возможности, например выступать посредником в сборе статистики использования той или иной функциональности продукта пользователями. В данном случае система логирования служит не для анализа выполнения программы, а больше как средство оценки поведения пользователя. Например, это может быть статистика посещения страницы некоторого товара в интернет магазине.

ИСТОРИЯ

История развития средств логирования не такая простая как могло бы показаться. Не так давно были времена, когда результат работы программ печатался на бумаге, а ход выполнения программы буквально можно было наблюдать по миганию ламп, из которых состояла вычислительная техника. Сейчас средства логирования получили широкое развитие, даже стали частью программных платформ, например JSR47[1] (Java Logging API). Вывод в лог теперь доступен разработчику всего лишь с помощью пары строк программного кода, однако на деле есть много тонких моментов.

Интересно упомянуть, что развитие систем логирования для Java платформы не обошлось без конфронтации. Приблизительно в 1999 появляется библиотека, дефакто ставшая стандартом вывода в лог – log4j[2]. В 2001 log4j уже получил широкую известность, у него была удачная архитектура и высокая стабильность в работе. Данная библиотека была хорошим кандидатом, чтобы войти в стандарт Java платформы. На деле оказалось, что Graham Hamilton, человек ответственный за разработку стандарта JSR47 выбрал за основу не log4j, а утилиту Logging Toolkit for Java, разработанную в IBM, автор которой, Chris Barlock, на тот момент входил в экспертную группу по JSR47. Принятый JSR47 нельзя назвать плохим, но по ряду характеристик он сильно уступал log4j, был недостаточно гибким и несовместимым с предыдущими версиями Java платформы. Данные обстоятельства привели к открытой критике стан-

дарта JSR47. Разработчики log4j обратились в Sun с письмом[3], под которым подписались более 200 человек с предложением принять за основу нового стандарта хорошо зарекомендовавшую себя библиотеку log4j. На данное письмо был дан ответ[4], суть которого сводилась к утверждению, что уже слишком поздно для внесения изменений в готовый к выходу грядущий релиз Java платформы версии 1.4.

На сегодняшний день для Java платформы существует несколько конкурирующих библиотек:

– log4j – известная библиотека, широко используется;

– JUL[5] (реализация JSR47) – является частью Java платформы, мало популярный;

– commons-logging[6] – обычно используется в legacy разработках, мало удобный в использовании;

– SLF4J[6] – очень популярная обертка для логирования;

– Logback[7] – современная утилита для логирования, используется как альтернатива log4j и SLF4J.

Одновременное существование нескольких утилит для логирования в рамках одной платформы дает некоторую свободу выбора, но есть и отрицательная сторона такой гибкости. Разработка систем, использующих несколько компонент, основанных на разных системах логирования усложняется, так как требуется корректно настроить вывод в общий лог файл несколько библиотек логирования.

ВОЗМОЖНОСТИ

Все библиотеки для логирования предоставляют широкий набор возможностей. Базовый набор общий для всех и может быть перечислен.

Первая, наиболее очевидная возможность, – это уровни логирования (log levels). Обычно доступны 6ть уровней, которые определяют критичность информации:

– Fatal – обозначает серьезные проблемы в программе, которые приводят к аварийному завершению;

– Error – неустранимая ошибка в программе;

– Warn – нештатная работа программы;

– Info – информационное сообщение;

– Debug – сообщение с отладочной информацией;

– Trace – максимально подробная отладочной информация.

Уровни логирования отсортированы и неразрывно связаны с возможностью настройки фильтрации того, какие сообщения попадут в лог файл. Например, при настройке вывода не ниже уровня Info, в лог файл будут записываться сообщения уровня Fatal, Error, Warn и Info. Debug и Trace проигнорируются.

Ротация лог файлов. Это механизм препятствующий неконтролируемому увеличению размера лог файла. В зависимости от настройки, при достижении лог файла определенного размера или по прошествию определенного периода времени, активный лог файл подменяется на новый.

Форматированный вывод (layout). Обеспечивает настройку что и в какой последовательности пишется в лог для каждого сообщения (время, имя метода, имя потока, сообщение, уровень логирования и другие параметры)

Поддержка клиентских потребителей сообщений (appenders или handlers). Запись в файл не всегда приемлема, в таком случае можно перенаправить вывод в базу данных, очередь или сеть, написать любую логику обработки сообщений – например, рассылать смс при превышении порога частоты возникновения Error сообщений.

Отдельно отметим требования на высокую стабильность и гибкое конфигурирование библиотек логирования. К важным свойствам также относятся потокобезопасность, поддержка контекста (NDC, MDC) и буферизация сообщений.

Легко заметить, что библиотеки логирования являются сложными программными продуктами и требуют понимания принципов своей работы.

ПРАВИЛА И РЕКОМЕНДАЦИИ

Можно с уверенностью утверждать, что большинство разработчиков программного обеспечения в своей повседневной работе применяют логирование. При разработке больших систем, которые требуют администрирования, поддержки и эксплуатации, наличие логирования становится очевидной необходимостью.

Трудности программирования вывода в лог можно разделить на два типа. Первый – недостаточное уделение внимания написанию логирующего кода. Это небрежное обращение с уровнями вывода в лог, недостаточный вывод в лог, излишнее логирование, непонимания принципов логирования исключений, а так же плохо составленные и малоинформативные сообщения. Вторая трудность как следствие первой состоит в том, что объектно ориентированный подход в разработке ПО недостаточно хорошо подходит для нужд логирования.

Логирование служит вполне определенным целям, но это не должно означать, что код вывода в лог живет отдельно от остального продукта. Он должен быть понятен, не вызывать сложности в сопровождении, выглядеть лаконично. Исполь-

зовать следует библиотеку для логирования, которая поддерживает шаблоны при формировании сообщений:

```
LOGGER.debug("transaction completed, id = {}", txId);
```

Не следует без явной необходимости использовать конструкции с обертками:

```
if (LOGGER.isDebugEnabled())
   {LOGGER.debug("transaction completed, id = " + txId);}
```

Логировать сообщения нужно всегда на латинице. Придерживаться этого принципа важнее, чем написание комментариев в коде на английском, так как в отличии от компиляции в полностью контроллируемом окружение, просмотр логов и их обработка происходит зачастую на различных платформах и в разных консолях.

В зависимости от типа программы и требований на нее, условия что логировать и с какими уровнями могут различаться. Исходить следует из следующих правил.

– Trace – сообщения этого уровня содержат максимально подробную информацию и генерируют большой объем логов. Сообщения этого уровня могут быть полезны только для программиста, в случаях когда требуется анализ выполнения операции шаг за шагом.

– Debug – основной уровень логирования, который используется в коде. Сообщения этого уровня позволяют установить причину многих ошибок, а так же часто используются на начальном этапе эксплуатирования новой функциональности, позволяя убедиться, что все работает штатно.

– Info – используется для журналирования жизненного цикла приложения (старт, загрузка модулей).

– Warn – обычно используется в случает возникновения ошибок, подлежащих восстановлению. Список ситуаций применения Warn:

1. ошибки, для который предусмотрен резервный механизм, обеспечивающий безотказное продолжение;

2. выявление непредусмотренных действий со стороны клиента (неожиданный запрос, нарушение протокола, неверный параметр вызова);

3. с оговорками возможно использование в случае, если основная операция завершилась успешно, а сопутствующая второстепенная – с ошибкой.

– Error – применяется для ошибок, которые не попадают под категорию Warn. Это ошибки не поддающиеся восстановлению и указывающие на неработоспособность подсистемы целиком или некоторой ее части.

– Fatal – как и Info используется для журналирования возникновения ошибок на уровне жизненного цикла приложения, приводящих к невозможности функционирования приложения в целом.

Данные правила и простые и в тоже время сложные. На практике часто возникают сомнения – какой уровень использовать Warn или Error, Debug или Info. Небольшое пояснение может пригодится.

В нормальной ситуации приложение должно эксплуатироваться с настройкой уровня логирования не ниже Info. При такой настройке в логе должно быть относительно мало сообщений, приложение будет порождать некоторое количество Info на старте и возможно Warn, если клиент не достаточно стабилен или перекладывает часть контроля за соблюдением протоколов и обработку подобных ситуаций на сервисную сторону. Следовательно, наличие Warn сообщений – это допустимая ситуация, которая не требует немедленного реагирования и может быть проанализирована в удобное время. Error сообщения в логе, напротив, требуют немедленного вмешательства, так как указывают на ошибки в окружении, в конфигурации или в коде программы.

Выбор между Info и Debug обычно неоднозначен в случае логирования действий в ответственной подсистеме, например, обработка денежных транзакций. Велик соблазн использования Info вместо Debug. Это возможно, но только с оговоркой, что Info используется крайне консервативно, только для логирования инициирующего вызова операции; далее должен быть применен Debug.

Логирование исключений (exceptions) имеет свои правила. Исключение следует логировать на уровне инициирующего метода самого внешнего компонента, в противном случае, если логировать при каждом пробрасывании или трансляции исключения, то лог будет заполнен по большей части идентичными и бесполезными сообщениями. Не каждое перехваченное исключение должно логироваться с уровнем Error, на практике обычно подавляющее большинство исключений – это Warn, как результат тех или иных нарушений контракта клиентом. Вывод в лог stack trace в основном должен применятся только в случае уровней Fatal и Error. Решение о выводе stack trace в лог не освобождает от сопроводительного сообщения.

[1] Встроенные (embedded) приложения зачастую могут работать с уровнем не ниже Error, логируя только сообщения типа Error и Fatal.

```
LOGGER.error("Billing unavailable", ex);
```

Перечисленных приемов достаточно для написания адекватного кода логирования и получения полезных сообщений в лог файле. Дальнейшее улучшение возможно за счет преодоления второй трудности, упомянутой выше, за счет использования принципов аспектно-ориентированного программирования[9] (АОП). АОП применяется для решения задач сквозной функциональности. Примеры такой функциональности: обработка ошибок, обеспечение транзакционности, проверка прав доступа. Перечисленные задачи с успехом решаются средствами АОП. Например, для языка программирования Java для всех перечисленных задач в платформах JEE и Spring[10] доступны и предлагаются аспектно-ориентированные подходы в качестве основных решений. Логирование несомненно является сквозной функциональностью, но ни JEE, ни Spring не содержат готового решения для логирования с помощью АОП. Разработка целостного АОП решение для логирования задача довольно объемная. Примером такого решения является библиотека aop-logging[11] для декларативного логирования в Spring приложениях.

```
@LogInfo
@LogException(
  value = {@Exc(value = Exception.class, stacktrace = true)},
  warn = {@Exc({IllegalArgumentException.class,
               NotEnoughMoneyException.class})})
public PaymentContractResponse processContract(
  PaymentContract request) {
    return shop.checkPayment(request);
}
```

Логирование с помощью АОР помогает разработчику освободиться от скорпулезного расставления вызовов вывода в лог сообщений о запуске и завершении методов. АОП избавляет от написания перехватчиков исключений с целью логирования. Инфраструктура АОП и единообразный подход позволяет обеспечить эфективный вывод в лог объектов, для которых не реализованы методы преобразования в строку. Как результат и объем написания кода и время на разработку системы уменьшаются.

ЗАКЛЮЧЕНИЕ

Все области программирования требуют внимательного отношения и логирование не исключение. Вывод в лог не должен воспринимается как второстепенная функциональность. Практики программирования логирования выработаны и описаны, их следует знать и использовать. Это позволит добиться информативного и гибкого логирования, не засоряя при этом исходный код громоздкими конструкциями. На примере аспектно-ориентированного подхода к логированию видно, что библиотекам для логирования есть поле для развития в том числе в программных платформах уровня предприятия.

Литература

[1] JSR 47: Logging API Specification: http://jcp.org/en/jsr/detail?id=47
[2] Apache log4j: http://logging.apache.org/log4j/1.2/
[3] JSR47 vs. log4j by Ceki Gülcü: http://web.archive.org/web/20011020060948/http://jakarta.apache.org/log4j/docs/critique.html http://web.archive.org/web/20020214222741/jakarta.apache.org/log4j/docs/srtw.html
[4] Subject: Log4J Vote: http://web.archive.org/web/20011109032926/http://jakarta.apache.org/log4j/docs/pub-support/GrahamHamilton.html
[5] JUL: http://docs.oracle.com/javase/7/docs/technotes/guides/logging/index.html
[6] commons-logging: http://commons.apache.org/proper/commons-logging/
[7] SLF4J: http://www.slf4j.org/
[8] Logback: http://logback.qos.ch/
[9] Aspect-oriented programming. http://en.wikipedia.org/wiki/Aspect-oriented_programming
[10] Spring Projects. http://www.springsource.org/spring-framework
[11] aop-logging utility. https://github.com/nickvl/aop-logging