

nekoval 4 февраля 2011 в 12:58

Java Logging: история кошмара

Java*

Вступление

Тернист и извилист путь Java-платформы к правильному способу записи строчек в лог-файлы. История logging в Java довольно познавательна в плане изучения особенностей Open Source, в том числе его взаимодействия с корпорациями и единичными программистами. Я собираюсь рассказать столько, сколько возможно, об истории развития Java logging, а также о том, к чему все пришло и как жить дальше. Мой анализ ситуации будет довольно субъективен про причине того, что logging — это всегда дело вкуса, а вкусы у меня сформировались свои, взрослые. Я думаю, что это будет познавательно не сколько в плане каких-то технических особенностей всего зоопарка logging frameworks, а в плане политики и психологии разработчиков в модели Open Source.

Начало

Понятно, что любая logging-библиотека должна позволять как минимум печатать строку на консоль/в лог-файл.

В начале был, конечно `System.err.println`. Кроме того, первая версия Servlet API имела в составе функцию `log` (впрочем, довольно примитивную).

Одним из вариантов более продвинутых решений в 1999 году был проект Avalon (и подпроекты, которые назывались Excalibur и Fortress), который помимо сервисов DI предлагал интерфейс LogEnabled. В компонент, который объявлял себя LogEnabled, инжектировался (я применяю это слово вместо «инжектировался», чтобы подчеркнуть его связь с DI) объект типа Logger, куда можно было писать: а) строки б) exceptions. Подход этот по тем временам казался свежим и новаторским, однако с современной точки зрения это чистой воды идиотизм и over-engineering. Использовать DI для логгирования никакого смысла нет, и статический экземпляр этого самого Logger вполне бы всех устроил. В Avalon же приходилось думать, куда этот проклятый Logger сохранить и что же делать, если класс не использует DI (т.е. не управляется контейнером), а логгировать в нем очень хочется.

Приблизительно в 1999 появляется библиотека нового поколения — log4j. Прототип библиотеки был разработан IBM (еще в эпоху, когда голубой гигант пытался втиснуть Java в OS/2), затем эстафету подхватил ASF. Продукт был уже гораздо более продуманный и обкатанный на реальных нуждах. Вообще надо сказать, что серверным приложениям на Java к тому моменту исполнилось всего где-то годик, а логгирование всегда было востребовано именно на сервере. За это время Java-сообщество начало постепенно понимать, что и как им нужно.

log4j разделил понятие логгера или *категории* (т.е. область приложения, которая хочет записать в лог), собственно записи в лог, которую осуществляют так называемые *appenders*, и форматирования записей (*layout*). Конфигурация log4j определяет, какие appenders к каким категориям прикрепляются и сообщения какого уровня (*log level*) попадают в каждый appender.

Краеугольный камень log4j — это иерархичность категорий. Например, можно логгировать все сообщения из `org.hibernate` и заглушить всё из `org.hibernate.type`. Через некоторое время де-факто установилась практика соответствия иерахии категорий и иерархии пакетов в приложении.

Иерархия категорий позволяет довольно эффективно отсекать лишние сообщения, поэтому log4j работал чрезвычайно шустро. Кстати, принципиальной для логгеров является не столько скорость записи, сколько скорость фильтрации ненужного (а ненужного обычно более 90%) и форматирование.

Принципы, заложенные в log4j, были довольно удачно портированы на другие языки: log4cxx,

Итак, резюмируем. Удачная архитектура, понятная схема конфигурирования, принцип fail-safe — почему бы не включить такую замечательную библиотеку в состав платформы?

Java Logging API

На деле все получилось странно. IBM, в недрах которой возник log4j, оказалась довольно шустрой в вопросах формирования нового JSR47 (Java Logging API). В частности, ответственный за JSR47 товарищ Graham Hamilton решил взять за основу *не log4j*, а *оригинальный IBM logging toolkit*. Причем logging toolkit был использован на полную катушку: совпадали не только имена всех основных классов, но и их реализации; код старались допиливать как можно меньше, видимо, чтобы успеть к очередному релизу платформы. Впрочем, концептуально это было очень похоже на log4j, только вместо appenders это называлось handlers, а вместо layout был formatter.

Поскольку основное назначение JSR47 — определять **API, а не реализацию**, доступных (по умолчанию в платформе) средств вывода было всего 4 (в log4j более 10), а средства форматирования были настолько бедны, что практически сразу приходилось делать свои formatter-ы, поскольку готовых не хватало. JSR47 предлагал использовать конфигурацию в виде `.properties`, причем в скобках отмечалось, что в файле можно описать не все. Таким образом, при усложнении конфигурации программист неожиданно обнаруживал, что опять требуется писать код, т.к. в виде `.properties` его конфигурация нереализуема.

Нельзя сказать, чтобы JSR47 проигрывал в производительности. Местами он обгонял log4j за счет поддержания в памяти специального представления своей конфигурации (что, кстати, одновременно усложняло эту самую конфигурацию). Однако, как выяснилось, JSR47 в обязательном порядке собирал так называемую Caller Information, то бишь «откуда логируется данное сообщение». Получение Caller Information — операция довольно дорогостоящая, протекает она с использованием Native-кода. Опытные дяди из log4j это знали, поэтому предоставляли эту возможность с оговоркой «лучше не включайте».

Разработчики log4j выступили с открытой [петицией](#), где потребовали «снять JSR47 с конвейера», пока он еще не попал в состав платформы. Петицию подписали более 100 человек... Однако было уже поздно. Следующий релиз JDK был утвержден и платформа понеслась в будущее с рудиментарным `java.util.logging`, или сокращенно JUL. Новый логгинг был настолько неразвит и неудобен, что использовать его решились только в нескольких серверах приложений (среди них Resin и Jetty). Sun, впрочем, отреагировала на петицию и большинство крупных проблем оригинального JSR47 постепенно были устранены. Тем не менее, эти манипуляции походили скорее на установку подпорок к деревянному мосту, которые ну никак не сделают этот мост железобетонным. Разработчики log4j сделали [реверанс](#) в сторону Sun, заметив, однако, что степень кривизны JUL все еще довольно высока. Помимо всего прочего, лицензия JDK 1.4 **не позволяла** использовать log4j в качестве реализации JUL. Последний поезд для log4j ушел.

Не будучи способным поддержать большое число лог-писателей (т.е. handlers), JUL выпендрился, определив невероятное число уровней логгирования. Например, для отладочных сообщений существовало аж 3 уровня — FINE, FINER и FINEST. Видя всё это, разработчики зачастую совершенно не понимали, какой же из трех уровней, чёрт возьми, надо использовать.

Java-сообщество было совершенно дезориентировано появлением «стандартного» логгинга параллельно с популярным, стабильным и развивающимся log4j. Никто не понимал, кто из них двоих жилец. Нередки были ситуации, когда в проекте было собрано несколько библиотек, каждая из которых использовала свой логгинг и свои настройки, записывая совершенно вразнобой свои лог-файлы.

Разумеется, сообщество попыталось исправить эту проблему. Началась **оберточная эпидемия**. Или, я бы даже сказал, пандемия.

Wrapper Hell

Когда вы подключаете несколько библиотек и пытаетесь соединить их логи в одно целое (а код модифицировать нельзя), это будет называться Adapter. Были написаны переходники из JUL в log4j и наоборот. К сожалению, переходники по функционалу являются «наименьшим общим кратным». Даже когда в log4j появилась поддержка контекста (NDC и MDC), при переливании в JUL она терялась. Хуже того, JUL работал только начиная с JDK 1.4, в то время как невероятное количество enterprise-приложений все еще сидело на 1.3. В итоге, сообщество стало одержимо идеей создания «общего стандарта де-факто», который бы все стали дружно употреблять и

который работал всегда и везде.

Приблизительно в 2002 из группы Jakarta выделился проект под названием commons-logging (JCL = Jakarta Commons Logging). Фактически это была обертка всех существующих на тот момент средств логгинга. Предлагалось писать приложения так, что они обращались к обертке (интерфейсу под названием `Log`), которая выбирала «подходящую» систему логгинга и сама к ней подключалась. Обертка была бедновата функционально и никаких дополнений к существующим средствам логгинга не вносила.

Как же *автоматически* выбиралась подходящая система логгирования? А вот это самое интересное. Во-первых, можно было задать ее явным образом размещением специального `commons-logging.properties` -файла где-нибудь в CLASSPATH. Во-вторых, через системное свойство (что, очевидно, никто делать не будет). В-третьих, если где-то в CLASSPATH обнаруживался log4j, то он автоматически задействовался. Таким же методом разыскивались реализации всех остальных библиотек, всегда подключалась первая найденная.

Красиво! Ну то есть *было бы* красиво, если бы весь софт в мире использовал бы commons-logging. Тогда можно было спокойно собрать JARы, положить в сервер приложений, а там уж JCL подхватит логгинг данного сервера приложений и вуаля!

На самом деле, как выяснилось, куча софта использует обычно «любимый логгинг своего разработчика». Это означает, что совершенно произвольная библиотека может в виде зависимости подтянуть, например, log4j, который таким образом попадет в CLASSPATH и неожиданно переключит JCL на использование log4j. Еще хуже с `commons-logging.properties`. Если какой-нибудь деятель додумывался запихнуть его в свой JAR, то при подключении этого JAR-а — сами понимаете — пиши пропало. Особую пикантность ситуации придавало то, что совершенно непонятно было, из какого именно JAR-а приехала инфекция. Иногда помогал перебор всех JAR-ов в алфавитном порядке. Иногда бубен.

Полная непредсказуемость выбора логгинга оказалась главной и очень веселой особенностью JCL. Группа log4j разразилась гневной статьей [Think again before adopting the commons-logging API](#), где предлагала остановить эпидемию и сосредоточить внимание на доработке существующего решения — log4j.

К сожалению, было уже поздно. С подачи Jakarta на commons-logging были переведены сотни, а затем тысячи библиотек. В их числе были Hibernate, Spring, Tomcat. После чего многочисленных пользователей этих библиотек захлестнула волна проблем, в целом описываемых как **ClassLoader hell**. В серверах приложений используется довольно сложная иерархия ClassLoader-ов, причем зачастую с серьезными отклонениями от стандарта J2EE. В этих условиях *иногда* JCL инициализируется дважды, причем неправильно, приводя к совершенно мистическим stack traces, не позволяющим даже заподозрить, что проблема в лог-обертке.

Почему, собственно говоря, Open Source сработал таким странным образом, породив на свет данное извращение? Почему разработчики не решились просто так взять и использовать другой зрелый и популярный Open Source продукт — log4j? Дело здесь, возможно, в некоторой инертности сообщества, привыкшего идти на поводу либо у ASF (а группа Jakarta, породившая данный кошмар, есть часть ASF), либо у Sun. Как только образуется критическая масса проектов, использующих JCL, все остальные (и не самые глупые люди, так ведь, Gavin King?) начинают использовать JCL (ибо Apache — это круто!). Это в целом напоминает броуновское движение, где такие бренды как Apache или Sun способны создавать области низкого давления, куда устремляются миллионы разработчиков. В случае JCL «история успеха» описана в блоге Rod Waldhoff (один из разработчиков так называемых Jakarta Commons) в 2003 году.

Новый виток прогресса

Итак, где-то на 2004 год имеем в комплекте:

1. Стабильный и функционально развитый log4j
2. Унылый java.util.logging
3. Проблемный commons-logging
4. Несколько мелких логгеров, недостойных упоминания

Отметим, что в проекте log4j в это время преобладали консервативные настроения. Особое внимание уделялось вопросу совместимости со старыми JDK. Вроде бы начинается разработка

новой ветки log4j — 1.3.x. Эта версия — своего рода компромиссное решение: да, хочется новый функционал, да, хочется поддерживать обратную совместимость, да, попробуем угодить и нашим и вашим. А тем временем на подходе JDK 1.5 с varargs, JMX extensions и кучей других подарков. В команде log4j началось брожение умов. Отпочковывается ветка 2.x — несовместимая с основной веткой 1.2.x и созданная специально для JDK 1.5. Java-сообщество изнывает в нетерпении. Происходит вроде бы как *что-то*. Но что именно, не понять — log4j 2.0 по-прежнему остается недостижимой альфой, log4j 1.3 дико глюкав и не обеспечивает обещанной drop-in совместимости. И только ветка 1.2 по-прежнему стабильна и жива-здорова, прыгнув *за несколько лет* — внимание! — с версии 1.2.6 до 1.2.12.

Где-то в 2006 году один из отцов-основателей log4j — Ceki Gülcü — решает выйти из стремительно тухнущей команды. Так появляется на свет очередная «обертка всего» под названием SLF4J (Simple Logging Facade for Java). Теперь это обертка вокруг: log4j, JUL, commons-logging и нового логгера под названием logback. Как видно, прогресс быстро дошел до стадии «обертка вокруг обертки». *Нетрудно спрогнозировать, что по той же схеме число обертываемых библиотек будет расти как факториал.* Однако SLF4J предлагает и другие прочие выверты. Это специальные binary-переходники: из log4j в SLF4J, из commons-logging в SLF4J и тому подобное. Делаются такие переходники для кода, исходники которого недоступны; при этом они должны подменить оригинальные JAR-ы лог-библиотек. Не берусь представить себе, какая каша при этом образуется, но если очень хочется, то можно и так.

При всей моей ненависти к оберткам, положи руку на сердце, SLF4J — хорошо сделанный продукт. Были учтены все недостатки предшественников. Например, вместо шаманских плясок с поиском классов в CLASSPATH придумана более надежная схема. Теперь вся обертка делится на две части — API (который используется приложениями) и Реализация, которая представлена отдельными JAR-файлами для каждого вида логгирования (например, `slf4j-log4j12.jar` , `slf4j-jdk14.jar` и т.д.). Теперь достаточно только подключить к проекту нужный файл Реализации, после чего — опа! весь код проекта и все используемые библиотеки (при условии, что они обращаются к SLF4J API) будут логгировать в нужном направлении.

Функционально SLF4J поддерживал все современные навороты типа NDC и MDC. Помимо собственно обертывания вызовов, SLF4J предлагал небольшой, но полезный бонус при форматировании строк. Бонус тут в следующем. В коде часто приходится печатать конструкции вида:

```
log.debug("User " + user + " connected from " + request.getRemoteAddr());
```

Помимо собственно печати строки, тут неявно произойдет преобразование `user.toString()` с последующей конкатенацией строк. Все бы ничего. В отладочном режиме скорость выполнения нас не очень волнует. Однако даже если мы выставим уровень, скажем, в INFO, окажется, что конструирование строки все равно будет происходить! Никаких чудес: строка конструируется *перед* вызовом `log.debug` , поэтому log4j не имеет возможности как-то это контролировать. Если представить, что этот `log.debug` размещен в каком-то критическом внутреннем цикле... в общем, так жить нельзя. Разработчики log4j предложили обрамлять отладочный код так:

```
if (log.isDebugEnabled()) {
    log.debug("User " + user + " connected from " + request.getRemoteAddr());
}
```

Нехорошо получается. По идее все эти проблемы должна брать на себя сама logging-библиотека. Эта проблема стала просто ахиллесовой пятой log4j. Разработчики вяло реагировали на пинки, рассказывая, что в logging-вызовы теперь можно еще добавить объект (ровно один!), да еще описать, как этот объект будет записан в лог с помощью интерфейса `ObjectRenderer` . По большому счету, все это были отмазки и полумеры.

SLF4J не был стиснут рамками совместимости со старыми версиями JDK и API, поэтому с ходу предложил более изящное решение:

```
log.debug("User {} connected from {}", user, request.getRemoteAddr());
```

В общем-то, все просто. В данной строке `{}` — это ссылки на параметры, которые передаются отдельно. Преобразование параметров в строку и окончательное форматирование лог-записи

происходит *только* при установленном уровне DEBUG. Параметров можно передавать много. Работает! Не надо писать обрмляющий `if` и прочую тупость!

В скобках надо отметить, что данную возможность также совершенно неожиданно реализовал язык Groovy, где есть понятие GString, т.е. строка вида

```
"User ${user} connected from ${request.getRemoteAddr()}"
```

, которая неявно связана с несколькими контекстными переменными (здесь это `user` , `request`), причем вычисление строки происходит *отложенным образом*. Это очень удобно для таких лог-библиотек как log4j — можно получить на вход GString, а затем или выбросить его без вычисления, или все-таки преобразовать в нормальную (статическую) строку — String.

Короче говоря, SLF4J был сделан грамотно, с заделом на будущее. Это вызвало серьезный рост его популярности среди сообщества: сейчас SLF4J используют такие значимые проекты, как Jetty, Hibernate, Mina, Geronimo, Mule, Wicket, Nexus... в общем, практически все неудачники, зависшие в свое время на commons-logging, перешли на SLF4J. Интересно, что мешало усовершенствовать commons-logging до нужного состояния много лет назад? Но таковы реалии Open Source — развитие софта в нем происходит скорее революционно, чем эволюционно.

Одновременно с SLF4J был подан к столу совершенно новый логгер — Logback. Он был сделан человеком, который на логгировании собаку съел, и на поверку действительно оказался хорошим продуктом. Logback был изначально заточен под JDK 1.5+, одним махом избавившись от всех старческих болезней обратной совместимости, свойственных проекту log4j. А это значит — `varargs`, `java.util.concurrent` и прочие прелести. Например, за счет встроенной системы runtime-фильтрации можно менять уровень логгирования в зависимости от пользовательской сессии, разбрасывать пользователей по разным лог-файлам и прочее, прочее.

Я подкину горчички в идиллию, нарисованную автором. Большинство этих возможностей можно реализовать в виде дополнительных appender-ов к log4j. Придется искривить и подпилить конфигурацию, это сложнее, но — факт, что переходить для этого на новый логгер не_обязательно. Таким образом, все рекламируемые Logback фишки — удобные, но не уникальные.

Что касается сообщества, то оно к Logback относится с осторожностью. Во-первых, за несколько лет он добрался до версии 0.9.x, а это пугает некоторых программеров. Во-вторых, Logback не находится ни под зонтиком Apache, ни в области действия Sun. Это смущает людей щепетильных. В-третьих, автору надо кушать, поэтому за некоторые довески к Logback и поддержку он требует денег. Это иногда отпугивает студентов. Помимо всего прочего, Logback имеет довольно сложную двойную лицензию (LGPL/EPL), в то время как log4j — универсальную лицензию Apache. Для библиотек и вообще redistributable софта лицензирование является очень тонким моментом.

По большому счету, Logback на сегодняшний день — вершина эволюции. Помимо Logback появилось уже с десяток новых logging-библиотек, но с большой вероятностью ни одна из них не выживет. Подводя итоги, ситуация на данный момент следующая:

- **log4j** — используют подсевшие на него изначально и не видящие необходимости перехода.
- **JUL** — тихо умирающий стандарт. Все, кто изначально пытался его использовать, переезжают на Logback.
- **commons-logging** — обычно задействован в легасу-библиотеках, которые очень боятся причинить неудобства пользователем, переехав на что-нибудь получше.
- **SLF4J** — очень популярен в библиотеках. Многие переехали на него, не выдержав ужасов commons-logging
- **Logback** — обычно современные high-performance серверы, которых не устраивает log4j.

Я уже говорил, что Open Source сообщество имеет тенденцию стекаться к «центрам тяжести». Сейчас таким центром тяжести выступает скорее SLF4J в силу «универсальности». Относительная популярность SLF4J в какой-то степени гарантирует от появления новых оберток. Число проектов, использующих SLF4J, уже является достаточным для накопления «критической массы». У Logback (того же автора, заметьте) такой критической массы нет. (Кстати, log4j по прежнему обещает нам золотые горы и версию 2.0, однако воз и ныне там.) Думаю, если Logback умирит свою гордыню и двинется в Apache, его позиции сильно улучшатся.

Заключение

Интересно посмотреть на историю вопроса под углом психологии программистов. Ведь в принципе всё это спиральное (и вроде как *прогрессирующее!*) движение — бесконечный «reinvent the wheel». То есть из двух вариантов «доработать существующее» и «сделать свое» всегда выбирался второй. Поэтому ни один из упомянутых проектов не выбился в безусловные лидеры (в те самые стандарты «де-факто»). Вместо этого разработчики были в разное время «нашинкованы» на разные проекты и действовали раздельно, вместо того, чтобы действовать сообща. Хотя не факт, что все авторы смогли бы работать в одной упряжке. Тут действовали и политические моменты (вспомним, как Graham Hamilton любил IBM), и просто банальные ссоры в команде. Стремление же участников Jakarta Commons обеспечить сообществу «свободу выбора» вообще обернулось для сообщества длительной «эпидемией оберток».

В общем-то, все эти пороки типичны для открытого сообщества. Эта более чем 10-летняя история также показывает, насколько ошибочно распространенное сейчас мнение, что Sun как будто бы что-то решало в Java-сообществе. Мы видим, что многие вещи происходили вопреки Sun и независимо от Sun. Одним словом, интересно, как оно пойдет дальше. В одном я уверен — проекты приходят и уходят, люди не меняются :)


Теги: java, java.util.logging, JUL, commons-logging, JCL, SLF4J, logback, log4j

Хабы: Java

Редакторский дайджест

Присылаем лучшие статьи раз в месяц

Электронпочта



76

Карма


0

Рейтинг

Сергей @nekoval


Пользователь

Комментарии 89




SSiarhei 04.02.2011 в 13:25


Спасибо, качественный разбор.




+3

Ответить









Antelle 04.02.2011 в 13:26


В log4net (порте log4j) нет проблемы в вызовами IsDebugEnabled. Странно, что они до сих пор не реализовали DebugFormat и в log4j



0

Ответить








aymeshkov 04.02.2011 в 14:04


Обратная совместимость же, не могут (в 1.2.x во всяком случае)




0

Ответить









Antelle 04.02.2011 в 14:16


как это связано с обратной совместимостью? добавили новый метод — ничего же от этого не пострадает




0

Ответить







johndow 04.02.2011 в 15:01

Обратная совместимость здесь с Java 1.4 в которой нет varargs.

НЛО прилетело и опубликовало эту надпись здесь

johndow

04.02.2011 в 15:31

А если мне надо 20 параметров передать? 20 методов делать? Нафиг нафиг. Лучше уж что-нибудь типа `info(String format, Object[] args)` — но всё равно неудобно.

НЛО прилетело и опубликовало эту надпись здесь

johndow

04.02.2011 в 15:54

Да тут пол статьи о том, что вышло в результате принятия неудачного API, которое вошло в стандарт и породило кошмар. Пять методов(а на самом деле это ещё надо помножить на кол-во уровней логирования) делающих одно и то же только с разным кол-вом параметров — это костыль, а не решение. И в публичной библиотеке такое недопустимо.

НЛО прилетело и опубликовало эту надпись здесь

johndow

04.02.2011 в 18:50

Спорить совсем не хочется(хотя хотелось, но литр жигулевского сделал своё дело :)). ИМНО, самое главное в библиотеке — это её интерфейс, а для библиотек которыми пользуются тысячи разработчиков по всему миру тем более. Хороший API — библиотека имеет шансы. Плохой API — библиотека плоха, независимо от её реализации. А добавление кучи методов делающих по сути одно и тоже никак нельзя назвать хорошим интерфейсом.

tonsky

05.02.2011 в 16:57

Вполне можно, по-моему, если таким образом использовать ее становится удобно. Не вижу проблемы. Двадцать аргументы передать — во-первых, тут уж будьте добры, воспользуйтесь массивом, а во-вторых, вам правдо случалось двадцать аргументов в лог-строку передавать? Я имею в виду, не «а что если», а вот на самом деле, по-настоящему?

nekoval

04.02.2011 в 16:40

кстати slf4j так и делает

SergeyGrigorev

04.02.2011 в 13:33

Жалко, что `log4j` не развивается. По мне он достаточно удобный и простой. Да и как уже было сказано [@antelle](#), его портированная версия для .NET уже достаточно хорошо развилась (а еще и NLog, как его аналог). Поэтому мне хотелось бы дальнейшего развития `log4j`, вместо того, чтобы переходить на что-то другое.

nekoval

04.02.2011 в 13:50

как ни странно, он хорошо прижился на Питоне

ophiuhus

04.02.2011 в 13:42

Логгирование — «ахилесова пята» более половины проектов на Java. Зачастую некоторые IDE — типа Netbeans при генерации блоков (`try/catch` и т.п.) автоматически пуляют туда логгирование с помощью JUL и так оно и остается в коде, при не внимательности.

○  **AlexanderYastrebov** 04.02.2011 в 15:10 ^

Да бросьте вы про невнимательность, вы что код совсем не читаете который среда генерит?

 **+2** Ответить  

○  **ophiuhus** 04.02.2011 в 15:32 ^

А Вы в команде не работали никогда? Когда потом подчищать код по всем классам.
Я не против авто генерации разных блоков, конструкторов и т.д., но не люблю когда среда навязывает какое то поведение.
И да, я это отключаю в в шаблонах генерации.

 **+1** Ответить  

○  **AlexanderYastrebov** 04.02.2011 в 15:37 ^

Работаю. самого раздражает.
Предлагаю читать предыдущий комментарий в ключе: «Руки отрывать надо тому, кто такое оставляет в своем коде»

 **0** Ответить  

○  **alexey_lahtadir** 04.02.2011 в 14:36

Спасибо. Очень интересно было прочитать историю. В целом, любая компания, где разработчики не являются цельной командой эта история в миниатюре повторяется. В нашей компании в результате несколько лет назад все-таки признали log4j стандартом de-facto, и споры хотя бы относительно этой библиотеки прекратились. Да, есть масса нюансов использования. Шишки набивались долго. Но в целом удобно и юзабельно

 **0** Ответить  

○  **nekoval** 04.02.2011 в 14:49 ^

А я помню жуткие времена, когда все писали свои обертки. У каждой компании была своя обертка и свой log viewer. По-моему, это был где-то 2001 год.

 **+1** Ответить  

○  **khorost** 04.02.2011 в 15:00 ^

Практически каждый разработчик участвовал в «изобретении велосипеда» :). Наиболее частые примеры — это своя реализация списка, вектора и системы логирования. На прежнем месте работы я попробовал «заменить» систему логирования разработанную и развиваемую одним из ведущих разработчиков на log4sxx.

Предложенный вариант снисходительно игнорировался и то, что в нем уже давно было реализовано и легко настраивалось через log4sxx.properties, изобреталось в собственной разработке снова и снова. Поэтому я, чтобы не создавать зоопарк разных реализаций библиотек логирования, отказался там от использования log4sxx. Но в остальных проектах где я принимаю архитектурные решения — использую log4sxx, log4j, log4net. Схожая реализация упрощает понимание применения библиотек логирования с разными языками программирования.

 **0** Ответить  

○  **alexey_lahtadir** 04.02.2011 в 15:11 ^

Ну у нас один программист попался, который писал с нуля отправку mail по спецификациям, свой http клиент по спецификациям... о системе логирования уже молчу. Есть и человек, который не признавал ничего готового а все писал сам, потому что запоминать свое проще. У него есть либа метров на 10, которая ходит по его проектам.
А вообще, любой программист, при поверхностном изучении почти любого фреймворка, выбирает ряд типовых задач, которыми он будет пользоваться и старается написать оберточку вокруг этих задач, чтобы все делалось желательно одним методом и без ошибок вылетающих дальше по стеку.

 **+3** Ответить  

○  **AlexanderYastrebov** 04.02.2011 в 15:15 ^



Каждый программист за свою жизнь должен написать тетрис, систему логирования и XML парсер :)

 **+4** Ответить  

○  **burdakovd** 04.02.2011 в 17:23 ^

А ещё систему веб-шаблонов

 0 [Ответить](#)  

 **Colwin** 09.02.2011 в 15:05 

0 из 3 :-)
видимо, жить мне еще предстоит до-олго)

 0 [Ответить](#)  

 **alienff** 04.02.2011 в 15:52

торт!



 +4 [Ответить](#)  

 **StShadow** 04.02.2011 в 15:52

Тут бы еще упомянуть довольно корявую реализацию логгирования в Eclipse (при разработке plugins или RCP Application). С одной стороны — вроде как есть уже что-то готовое и тянуть еще-одну-либу log4j нет смысла, с другой — ILog уж больно страшен.

 0 [Ответить](#)  



НЛО прилетело и опубликовало эту надпись здесь

 **nekoval** 04.02.2011 в 16:39 

Думаю, что зря. Что делать, если класс не управляется Guice?

 0 [Ответить](#)  



НЛО прилетело и опубликовало эту надпись здесь

 **nekoval** 04.02.2011 в 16:52 

я просто хочу использовать new MyObject(), в таком случае зачем мне сдался весь этот DI?

 0 [Ответить](#)  



НЛО прилетело и опубликовало эту надпись здесь

 **nekoval** 04.02.2011 в 17:35 

OMG!

 0 [Ответить](#)  

НЛО прилетело и опубликовало эту надпись здесь

 **nekoval** 04.02.2011 в 17:54 

Скорее это Factory. Описывается в документации log4j:



```
public class MyApp {

    // Define a static logger variable so that it references the
    // Logger instance named "MyApp".
    static Logger logger = Logger.getLogger(MyApp.class);
```

и, кстати, это работает ВЕЗДЕ.

 0 [Ответить](#)  

НЛО прилетело и опубликовало эту надпись здесь

 **nekoval** 04.02.2011 в 23:07 

По времени log4j был создан когда уже почти загибался проект Avalon с его дурацкими Loggable/LogEnabled.

И не делайте из еды культа :) С таким подходом мне как минимум придется писать следующее:

```
myObj1 = new MyClass(Logger.getLogger("MyClass"));
myObj2 = new MyClass2(Logger.getLogger("MyClass2"));
```

Это может быть и гибко, но делать это ВО ВСЕМ приложении весьма утомительно.

Кстати, мне интересно, что нам предложит DI, если мы захотим logger в статическом методе.

0 Ответить

НЛО прилетело и опубликовало эту надпись здесь

nekoval 07.02.2011 в 10:49

>что бы было гибко и не утомительно
Правильно так: Чтобы было гибко И утомительно

0 Ответить

НЛО прилетело и опубликовало эту надпись здесь

nekoval 07.02.2011 в 12:49

я уже написал, что не все классы используют DI.
если для того, что написать строчку в лог, я должен включать мой объект в контейнер DI и перестать использовать конструктор — это называется «утомительно».

0 Ответить

НЛО прилетело и опубликовало эту надпись здесь

nekoval 07.02.2011 в 13:27

Мда. Хабр — это, оказывается, так весело)))))))
Слушайте, ну возьмите вы исходники какой-нибудь реальной библиотеки, например, Spring или HttpClient и посмотрите, использует ли там кто-нибудь DI. А также обратите внимание на статические методы.

0 Ответить

НЛО прилетело и опубликовало эту надпись здесь

nekoval 07.02.2011 в 16:09

а зачем мне их сравнивать? у меня вот приложение содержит и низкоуровневые компоненты, и бизнес-логику, и статические методы.

вот вы бы сами попробовали написать реальную DI-конфигурацию для логинга ну хотя бы для 5-10 классов, сами все сразу поймете

0 Ответить

НЛО прилетело и опубликовало эту надпись здесь

Colwin 09.02.2011 в 15:12

Да нет, не аномалия)


Иногда появляются утилитные бизнес-методы этак на 2-7 строчек, и их не привяжешь к какому-либо объекту, т.к. это будет противоречить принципам ООП.

Вот и выделяются в static-методы утилитных классов. Их, конечно, можно пихать в DI, но делать из утилитного класса singleton только для пропихивания logger'a — ИМХО, некрасиво.

А насчет AOP... Ну, не все используют AOP. Из разных соображений, но иногда полномочий не хватает, чтобы внедрить технологию, приходится использовать то, что есть, с полной отдачей.

0 Ответить

НЛО прилетело и опубликовало эту надпись здесь



Semenych

04.02.2011 в 18:18

↑


Logger.getLogger(MyApp.class);

Имеет неприятный эффект его использование под некоторыми версиями tomcat может вести к тому что у вас не будут выгружаться из памяти классы старого war при горячем редеплое. т.к. логгер будет завязан на этот класс. Лучше писать


Logger.getLogger(MyApp.class.getName());

◆ 0

Ответить



...




Stocker

04.02.2011 в 16:21


Спасибо, побольше бы таких статей.

◆ +2

Ответить



...




ostapbender

04.02.2011 в 16:23

Отлично написано!


◆ +1

Ответить



...

НЛО прилетело и опубликовало эту надпись здесь




Semenych

04.02.2011 в 17:34


Статья интересная, но вот что меня интересует — а почему JUL то умирает. Успешно использовал в куче проектов ни разу проблем не было. Чем он плох? Я совершенно серьезно спрашиваю.

◆ 0

Ответить



...



nekoval

04.02.2011 в 17:38


↑

там целый комплекс проблем.


одна из них навскиду — очень сложно апгрейдить, ведь JUL является частью платформы. В то время как log4j апгрейдится заменой JAR-файла.

◆ 0

Ответить



...



Semenych

04.02.2011 в 17:43

↑

м-м-м а зачем апгрейдить? 99% он вполне покрывает.

Мне в голову приходит пожалуй только невозможность сделать

if (log.isEnabled(DEBUG))

{


String txt = composeTimeConsumingLogInfo();

log.debug(txt);


}

◆ 0

Ответить



...



nekoval

04.02.2011 в 17:48

↑


может быть миллион причин апгрейдить... наличие бага или уязвимости или какая-то новая фича.

это может быть и пустяк... но в целом слабую функциональность JUL мало что вылечит.


Например, попробуйте там найти Handler для записи в syslog. Его **до сих пор** нет. А в log4j он присутствует где-то с 2000 года.

◆ 0

Ответить



...



Semenych

04.02.2011 в 17:53

↑

Ну так получилось что в syslog мне писать не надо. А если надо то я пожалуй сам могу с log4J содрать.


Я тут наоборот всех своих практикантов ругаю, говорю берите JUL

1. Он уже есть, не увеличивайте сложность сверх необходимого

2. Он простой, у вас будет меньше шансов наколбасить что-то сложное.

◆

Ответить



...

0 Ответить

nekoval 04.02.2011 в 17:58

ну для практикантов может быть JUL действительно в чем-то лучше.
вообще то, что он «уже есть» наверное, единственный аргумент для его использования

0 Ответить

Semenych 04.02.2011 в 18:03

Ну основной point (см пост ниже) в 99% случаев сложное не надо а JUL для простых случаев более чем подходит.
Самое прикольное для serverside быстрота тоже особенно не важна. т.к. на 100% загрузить процессор все равно не реально.
Из продвинутых вещей мне бы было нужен logbag но тащить его если честно не хочется ибо игра не будет стоять свеч
(http://www.theserverside.com/news/thread.tss?thread_id=42243 logBag thread)

0 Ответить

vrupkin 05.02.2011 в 13:21

Самое прикольное для serverside быстрота тоже особенно не важна. т.к. на 100% загрузить процессор все равно не реально.

```
logger.info(«x = {}», new Object[]{String.valueOf(x)});
```

0 Ответить

vrupkin 05.02.2011 в 13:30

... чё сказать то хотел....:

— Самое прикольное для serverside быстрота тоже особенно не важна. т.к. на 100% загрузить процессор все равно не реально.

использование логинга из примера

— `logger.info(«x = {}», new Object[]{String.valueOf(x)});`

может не только процессор на 100% но и память ВСЮ отожрать за несколько минут работы сервера, рождая миллионы объектов.

ЗЫ+ _____
*а что у меня форматирование/html-теги не работают совсем !?
* FireFox + Ubuntu >8-E
*ASCII-artom как то не круто получается
.....+ _____

0 Ответить

bachin 04.02.2011 в 17:40

```
public abstract class Debug
{
    public static final void _trace ( final String text )
    {
        System.out.println ( "Trace: " + text );
    }
}
```

На этапе сборки релизной версии препроцессор (писанный на коленке) попросту выкидывает из кода куски от «Debug._trace (» и до соответствующей закрывающей скобки. Я не претендую на то, чтобы это было эталонным подходом к логированию, но мне этого (чуть понавороченнее, я тут только суть указал) пока хватает за глаза.

-1 Ответить

Semenych 04.02.2011 в 17:44

Увы при разработки логи мне например нужны в занчительно меньшей степени чем в production.
да и возможность раскидывать по нескольким файлам очень важна

 **+2** [Ответить](#)  

 **Semenych** 04.02.2011 в 17:47

Коллеги а никто не помнит как назывался логгер который умел logBags. Было это примерно так



```
Message msg = receiveMessage
LogBag bag = log.getBag(msg.phoneNumber);
дальше я препарирую message логируя с уровнем debug и соответственно в лог это не пишется
log.debug(bag,«message body\n»+msg.dumpBytes());
и так далее
НО если возникла ошибка
log.error(bag,«фигня случилась»);
то уже логируется все что относилось к этому bag включая debug
```

 **0** [Ответить](#)  

 **fls_welvet** 04.02.2011 в 18:23

на codegeeks тоже была интересная [статья](#) про логирование

 **+2** [Ответить](#)  

 **r0zh0k** 04.02.2011 в 21:29 

А еще есть хороший обзор с примерами у skipy.

 **+2** [Ответить](#)  

 **Taro** 05.02.2011 в 16:59

Большое спасибо. Замечательная статья. Жаль, нет кармы, плюсанул бы.

 **-1** [Ответить](#)  

 **sskorykh** 05.02.2011 в 19:50

С SLF4J тоже не все сладко. Одно время я практиковал встраивание Jetty в собственное приложение. Приложение могло работать под управлением внешнего сервера Jetty/Resin/Tomcat, а могло и само обслуживать HTTP-запросы. Это очень удобно в некоторых случаях.

Но тут выяснилось, что если моё приложение использует SLF4J, а при этом еще из приложения запускается Jetty, который тоже использует SLF4J, то я лишаюсь журналирования. Даже если я не использую SLF4J напрямую, он может быть задействован в одной из библиотек, например Hibernate, и я таки опять лишаюсь журналирования.

Причину, помнится, я более-менее понял, но так и не понял, кого именно винить: Jetty за его ClassLoader-ы или SLF4J, который оказался недостаточно гибким. Обе точки зрения имеют право на жизнь, а я на всякий случай откатился в стан почитателей Log4J.

 **0** [Ответить](#)  

 **nekoval** 07.02.2011 в 11:38 

да, есть такое — если кто-то подложил свой implementation jar, то он перекрывает все остальное видимо в jetty не очень разобрались с зависимостями

 **0** [Ответить](#)  

 **sskorykh** 07.02.2011 в 15:11 

Там проблема в том, что Jetty строит для запускаемого контекста classpath, в который адаптер SLF4J-Log4J включён дважды. А архитектура SLF4J не позволяет такие трюки, в результате «Class path contains multiple SLF4J bindings». С одной стороны, косяк на стороне Jetty, с другой — SLF4J мог бы быть и подружественнее к такого рода проблемам.

Сегодня с удивлением наткнулся еще на одну проблему. Приложение использует Hibernate, а тот использует SLF4J. При попытке запуска приложения под Resin 4.0.14 сервер не смог подхватить SLF4J и Log4J из каталога WEB-INF. При переносе этих библиотек в [Resin]/lib все запустилось. В Resin 3.1.10 все запускается без проблем. Детально разбираться не стал. По всей видимости, косяк в Resin, в котором SLF4J то добавляют, то убирают.

 **0** [Ответить](#)  

- nekoval

07.02.2011 в 15:36

jetty разве использует log4j? мне казалось, что там logback

что касается Resin, то видимо где-то в рутовых classloader-ах лежит slf4j. что вообще необычно, т.к. Resin очень любит JUL.

0

Ответить
- sskorykh

07.02.2011 в 18:04

Что касается Jetty, я наверное не очень ясно выразился. Подробно проблема описана здесь:

www.rsdn.ru/forum/java/3934217.flat.aspx

А по поводу Resin в одном из issue, касающемся Resin 4.0.12 я нашел такой коммент:

slf4j removed from distribution, since Resin does not depend on it.

Из чего и сделал вывод, что Resin пытались скрестить с SLF4J.

Надо ли говорить, что всё это очень сильно действует на нервы. Платформа Java имеет некоторое количество проблем, из которых Logging — самая серьёзная. Так что правильная тема в статье поднята.

0

Ответить
- malkolm

05.02.2011 в 21:44

непонятно причем тут опенсорс сообщества, упоминаемое на каждом шагу

0

Ответить
- nekoval

07.02.2011 в 10:53

Упоминания относятся к Java-open source-сообществу.

0

Ответить
- malkolm

07.02.2011 в 11:04

статья ведётся, словно опенсорсное сообщество прям пользует и определяет что хорошо, а что плохо — хотя это шокапец не верно

0

Ответить
- nekoval

07.02.2011 в 11:35

Это откуда такое? Там другие механизмы действуют. Но вообще это оффтопик.

0

Ответить
- 23derevo

17.02.2011 в 21:40

отлично, всё здорово разобрано.

0

Ответить
- javenue

07.03.2011 в 21:45

Огромное спасибо за статью. Уже очень долгое время:

— использую Log4j,

— матерю java-util-logging,

— тихо ненавижу commons-logging

— побаиваюсь SLF4J.

После прочтения статьи появилось желание попробовать logback. Что и сделаю в ближайшее время.

0

Ответить
- WFrags

06.04.2011 в 08:54

Про logback не указана, на мой взгляд, важная особенность. Он нативно реализует SLF4J API (команда/спонсор судя по всему одна и та же — QOS.ch). А это значит, что используя «православную» связку SLF4J + logback, обёрток как бы и нет.

0

Ответить

1ex 23.08.2011 в 19:21

Согласен с WFrog, кроме того SLF4J — просто кучка интерфейсов, Logback — просто реализация (нативная), поэтому ставить их в один ряд немного неправильно. Вообще мне больше всего нравится через slf4j и его адаптеры перенаправлять все в Logback, и в своем проекте использовать slf4j. Самый народный вариант и все зависимости логируют свой злам единообразно.

0 Ответить

archislav 12.05.2012 в 22:09

Спасибо за отличную статью!

0 Ответить

Flammar 25.03.2013 в 16:01

бесконечный «reinvent the wheel». То есть из двух вариантов «доработать существующее» и «сделать свое» всегда выбирался второй.

«Существующее» нередко имеется в виде, уже распухшем почти до уровня мини-ОС, так что в таких случаях «сделать своё» — это ещё и тупо быстрее. К тому же «дорабатывать» — это не брать готовое, по трудозатратам вполне сопоставимо со «сделать своё», и при этом зависеть от выпуска новых версий (и изменения политики лицензирования) чужого продукта.

0 Ответить

nekoval 25.03.2013 в 16:46

Ответ на вопрос «подружиться с разработчиками продукта с миллионом пользователей или писать свой, с 0 пользователями» не так однозначен и не только трудозатраты решают. Хотя ссоры в оперсorcовых командах не редкость — тот же ffmpeg распилили вдоль и поперек.

0 Ответить

cdkrot 01.07.2013 в 11:09

Нельзя сказать, чтобы JSR47 проигрывал в производительности. Местами он обгонял log4j за счет поддержания в памяти специального представления своей конфигурации (что, кстати, одновременно усложняло эту самую конфигурацию). Однако, как выяснилось, JSR47 в обязательном порядке собирал так называемую Caller Information, то бишь «откуда логируется данное сообщение». **Получение Caller Information — операция довольно дорогостоящая, протекает она с использованием Native-кода.** Опытные дяди из log4j это знали, поэтому предоставляли эту возможность с оговоркой «лучше не включайте».

Может я не понимаю, но разве это нельзя получить через new Exception().getStackTrace()?

0 Ответить

nekoval 02.07.2013 в 10:16

Может я не понимаю, но разве это нельзя получить через new Exception().getStackTrace()?

Можно. Только этот метод появился в Java 1.4 :) Раньше наиболее быстрым был вызов Reflection.getCallerClass(...) , но он все равно слишком медленный для логгирования.

0 Ответить

cdkrot 02.07.2013 в 17:27

Спасибо.

0 Ответить

gkislin 11.10.2013 в 21:34

1. На мой взгляд наиболее важное преимущество любой сторонней библиотеки логгирования перед JUL- возможность для каждого задеплоенного модуля в контейнере иметь свой собственный лог-файл, а не валить все в одну кучу.

2. Пользую slf4j с адаптерами. При этом если модуль подтягивает зашаренную библиотеку контейнера,


то эти (общие) классы или не имеют реализации slf4j- тогда весь логгинг у них теряется, либо, если подложить реализацию в общие библиотеки, будет двойной биндинг. В любом случае логгинг зашаренных классов по модулям уже корректно не растащить. Может кто подскажет решение? (не считая решением тащить все библиотеки с собой)

3. Пользую таки свою обертку с такими вспомогательными вещами как:

```
public IllegalStateException getIllegalStateException(String msg, @Nullable Throwable
    logger.error(msg, e);
    return new IllegalStateException(msg, e);
}

public IllegalArgumentException getIllegalArgumentException(...)
...
public UserSecurityException getSecurityException(String msg, String user) {
```

0 Ответить

 **AlexSerbul** 10.01.2015 в 20:35
Прекрасная статья, спасибо!

0 Ответить

Только полноправные пользователи могут оставлять комментарии. Войдите, пожалуйста.

ПОХОЖИЕ ПУБЛИКАЦИИ

28 сентября 2016 в 11:17

The Game of Java: Java-конференция в Киеве 14-15 октября 2016 года

9

3.5K

3 +3

1 сентября 2016 в 15:09

The Game of Java: Java-конференция в Киеве 14-15 октября 2016 года

9

4.7K

6 +6

25 февраля 2016 в 22:38

Справочник по синхронизаторам java.util.concurrent.*

817

2

14 +14

МИНУТОЧКУ ВНИМАНИЯ

Разместить



Коллекция

Как приготовить лучший кейс: пробуем статьи на вкус



Мегаквест

Квантовый сисадмин чинит облачное фэнтези [квест]



Опрос

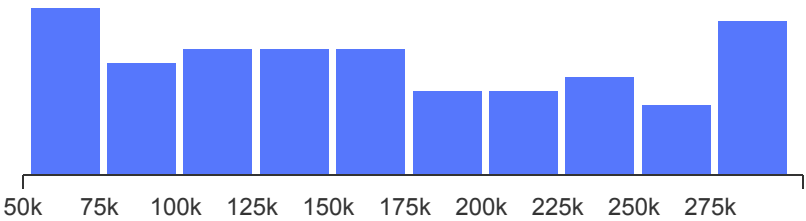
Третье хабраисследование ru-IT-брендов

СРЕДНЯЯ ЗАРПЛАТА В IT

157 142

137 142 /мес.

— средняя зарплата во всех IT-специализациях по данным из 4 003 анкет, за 2-ое пол. 2022 года. Проверьте «в рынке» ли ваша зарплата или нет!



Проверить свою зарплату

ЛУЧШИЕ ПУБЛИКАЦИИ ЗА СУТКИ

вчера в 14:36

Как я хакнул свой автомобиль: завершение истории

+79 7.1K 45 3 +3

вчера в 10:01

Ещё наши эпические грабли работы с маркетплейсами

+56 8.5K 17 24 +24

вчера в 12:21

Kubernetes 1.25: обзор нововведений

+43 3K 8 2 +2

вчера в 12:32

Yandex и VK разыграли многоходовую комбинацию с Delivery Club, Дзен, Новостями и доменом yandex.ru

+33 14K 10 74 +74

вчера в 10:46

Подбираем скины в Counter-Strike: Global Offensive в цвет сумочки

+33 2.5K 11 8 +8

Хотим узнать, что вы думаете о своём месте работы (анонимно)

Опрос

ЧИТАЮТ СЕЙЧАС

Российский аналог «Википедии» перестал работать через несколько часов после запуска

9K 49 +49

21 год Windows XP. Вспоминаем, как это было

4.6K 20 +20

Просверлить или ударить молотком: поддержка Samsung попросила клиента сломать неисправный SSD перед отправкой на замену

9.1K 44 +44

Перекатываемся в Райффайзен из Тинькофф...

12K 32 +32

Инженер создал портативный метатель ножей с прицеливанием и расчётом броска с помощью лидара

4K 7 +7

Текст как блюдо: определяем ингредиенты и составляем рецепт

РАБОТА

Java разработчик
469 вакансий

Все вакансии

Ваш аккаунт

Войти
Регистрация

Разделы

Публикации
Новости
Хабы
Компании
Авторы
Песочница

Информация

Устройство сайта
Для авторов
Для компаний
Документы
Соглашение
Конфиденциальность

Услуги

Корпоративный блог
Медийная реклама
Нативные проекты
Образовательные
программы
Стартапам
Мегапроекты



Настройка языка

Техническая поддержка

Вернуться на старую версию