03-2014 | Herbst | www.ijug.eu

Java aktuell

Praxis. Wissen. Networking. Das Magazin für Entwickler

Aus der Community – für die Community

Java aktuell

Java im Mittelpunkt

Aktuell
NoSQL für Java

Performance
Java-Tuning

Pimp my Jenkins

Logging
Apache Log4j 2.0

Sonderdruck

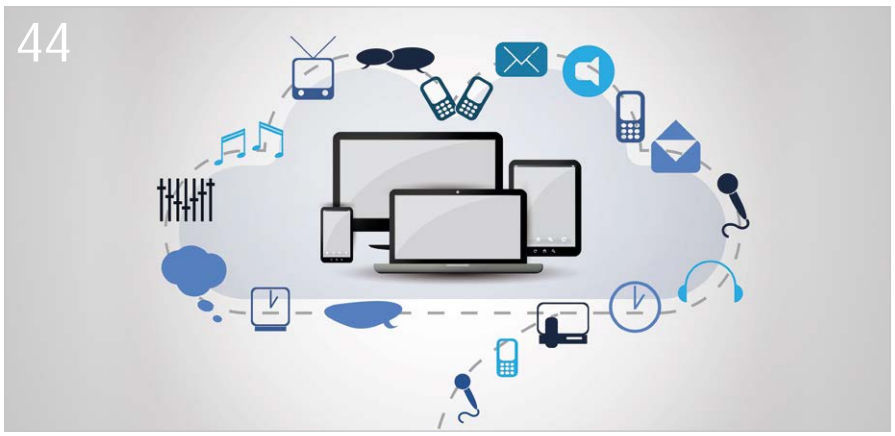
D: 4,90 EUR A: 5,60 EUR CH: 9,80 CHF Benelux: 5,80 EUR ISSN 2191-6977

4 191978 304903 02

iJUG Verbund

JavaLandConf 2014 Twitterwall

JavaFX– das neue Gesicht für Java-Anwendungen

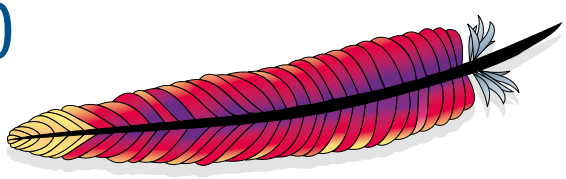


Connectivity-as-a-Service für Cloud-basierte Datenquellen

Unbekannte Kostbarkeiten des SDK

Logging und Apache Log4j 2.0

Christian Grobmeier, grobmeier.de



Wir schreiben das Jahr 2014. Warum wird eigentlich immer noch über Logging diskutiert? Weil es wichtig ist. Und tatsächlich, es gibt einiges zu bereden. Warum braucht es ein Log4j 2.0? Und warum werden wir auch damit noch weiterdiskutieren müssen? In diesem Artikel geht es um die Geschichte des Java-Logging, um das neue Log4j und um eine Idee, wohin die Reise gehen könnte.

Im Dezember 2003 gründeten sechs Entwickler das Projekt „Apache Logging Services" (LS) unter dem Deckmantel der Apache Software Foundation (ASF) [1]. Darin enthalten ist das allseits bekannte Apache Log4j, das damit seinen 10. Geburtstag feiert. Das erste Log4j-Release war eigentlich im Jahr 1999, allerdings im Rahmen des Apache-Jakarta-Projekts.

Log4j hat das Loggen einfacher gemacht. Überhaupt, Log4j ist es zu verdanken, dass der Begriff „Logging" in das Bewusstsein vieler Entwickler gedrungen ist. Durch seinen flexiblen Aufbau wurde es zur Standard-Lösung in zahllosen Projekten. Einer (nicht-repräsentativen) Umfrage von „Zero Turnaround" zufolge verwenden heute etwa 52 Prozent aller Entwickler Log4j [2]. Vor einigen Jahren dürften es noch wesentlich mehr gewesen sein und Log4j hätte der König der Logging-Frameworks werden können. Doch es gab Probleme im Projekt, deren Nachwirkungen wir heute noch spüren.

Die Logging-Kriege

Diese Probleme fanden im Jahr 2005 ihren Höhepunkt. Jeder kann sich selbst ein Bild davon machen, wenn er die öffentlichen Mail-Archive des Projekts einsieht. Letztlich hat einer der Gründer von Apache Log4j, Ceki Gülcü, das Projekt gegabelt, überarbeitet und unter dem Namen „QOS Logback" veröffentlicht. Gülcü hat Apache LS nie verlassen und ist nach wie vor Mitglied des Project Management Committee (PMC). In seinem Blog führt er seine Kritik an der ASF aus [3].

Durch Wahl wird man in einem ASF-Projekt zunächst zu einem „Committer", dann später möglicherweise sogar Mitglied im PMC. Im Prinzip hat jedes gewählte Projekt-Mitglied eine Stimme bei Entscheidungen und jede Stimme zählt gleich. Nur in wenigen Situationen zählt die Stimme eines PMC-Mitglieds mehr. Es ist möglich, dass eine Person eine technische Änderung mit einem Veto blockiert, wenn sie nicht damit einverstanden ist.

Ein Veto führt normalerweise zu Diskussionen, die in vielen Fällen mit einem Konsens oder zumindest mit einem Kompromiss abschließen. Wenn die Meinungen allerdings zu unterschiedlich sind, kann dies auch zu einer Art „Deadlock" führen. Gülcü bevorzugt ein anderes Modell [4], in dem eher Stimmrechte nach Anzahl der Code-Modifikationen vergeben werden. Damit gäbe es immer eine starke Person, die das Sagen hätte. Dieser Ansatz ist aber mit der ASF nicht vereinbar. Außerdem gibt es auch gute Gründe, den ASF-Ansatz zu wählen, doch mehr dazu später.

Damit haben technische Unstimmigkeiten und die Unfähigkeit, einen Konsens zu finden, zum Beinahe-Ende von Log4j und der Gründung von Logback geführt. Logback konnte sich in relativ kurzer Zeit etablieren und hat einige interessante Änderungen eingeführt, die es in Log4j Version 1 nicht gab. Zudem wurden die ersten Versionen von „slf4j" veröffentlicht. Es handelt sich dabei um einen Wrapper für Logging-Frameworks, ebenfalls aus dem Hause QOS. Wer diesen verwendet, kann sein Logging-Framework auswechseln (zumindest theoretisch).

Die Idee ist nicht neu: Apache-Commons pflegt eine Komponente, die genau das Gleiche anbietet. Allerdings spielt bei Commons-Logging ein modernes API keine wesentliche Rolle. Der Fokus wird auf Stabilität gesetzt. Dieser eigentlich lobenswerte Ansatz rächt sich jedoch in diesem Fall: „slf4j" ist ebenfalls sehr stabil und bietet einfach mehr. Dementsprechend gibt es nur wenige Projekte wie zum Beispiel Apache Tomcat, die auf Commons-Logging setzen.

Nicht dass zwei Logging-Engines und zwei Logging-Facades genug wären – auch das JDK liefert bekanntermaßen seine eigene Logging-Engine mit: „java.util.logging" (JUL). Der Autor kenne keine Fans von JUL, jedoch gibt es tatsächlich einige Entwickler, die es benutzen. „Zero Turnaround" nennt in seiner Befragung etwa sieben Prozent Marktanteil.

Damit ist das Chaos perfekt. Als Entwickler will man eigentlich nur loggen. Aber womit? In großen Projekten kann es passieren, dass jedes Framework, das man verwendet, seine eigenen Logging-Abhängigkeiten mitnimmt. Im schlimmsten Fall hat man nicht nur die oben genannten, sondern möglicherweise auch noch ein paar Exoten und vielleicht sogar verschiedene Versionen im Einsatz. Letzten Endes ist mühseliges Konfigurieren angesagt, was einem schon mal den Geduldsfaden reißen lassen kann.

Einfach nur loggen

Logging ist ein mühseliges Thema, mit dem sich niemand gern beschäftigt. Beim Begriff „Log4j 2.0" schlägt einem oft eine Mischung aus Misstrauen und Stirnrunzeln entgegen. „Warum kann man nicht an einem Strang ziehen?", lautet der Vorwurf. Während man sich sein Web-Framework sehr genau ansieht und prüft, ob Leistung und Konzept zur eigenen Vision passen, interessiert sich kaum jemand für seine Logging-Engine. Obwohl Logging enorm wichtig ist, dürften die wenigsten wissen, was ein Receiver oder der Watchdog ist. Oft weiß man nicht einmal, welche Arten

von Logs es zu schreiben gibt. In vielen Projekten reicht eine Datei, die man zum Entwickeln beschreibt – und für den Live-Betrieb leer lässt.

Ein Logging-Framework ist eines der wenigen Projekte, die sich eigentlich nicht weiterentwickeln dürfen. Von log4j Version 1 wird erwartet, das man Upgrades nicht spürt und alles abwärtskompatibel bleibt. Erst seit dem Jahr 2012 bietet log4j 1.x keine Unterstützung mehr für Java 1.3. Wer einmal auf Logback gewechselt hat, erwartet dasselbe auch hier. Seit einiger Zeit arbeitet der Autor an Log4j 2.0. Wer sich mit dem Projekt outet, erntet oft eine hochgezogene Augenbraue: „Noch ein Framework?"

Aber haben wir tatsächlich noch ein neues Framework? Irgendwie „ja" und irgendwie „nein". Es ist eine neue Version, die fundamentale Probleme von Log4j Version 1 und Logback behebt. Log4j 2.0 hat allerdings nicht mehr viel mit Log4j Version 1 oder Logback zu tun, wenn man sich den Code ansieht. Wie gesagt: Die Architektur wurde stark überarbeitet. Nur Konzepte und Ideen wurden verwendet und daraus eine verbesserte, stabilere Engine gebaut.

Wer auf Log4j Version 2 wechselt, muss eine neue Abhängigkeit einbinden. Die Neuigkeiten wollen erlernt, die neuen Features erforscht sein. Eigentlich ist das spannend und macht auch Spaß. Aber die Begeisterung hält sich in Grenzen, wenn es sich um ein Logging-Framework handelt.

Auch Logging-Frameworks müssen sich entwickeln und aus ihren Fehlern lernen. Wir Entwickler müssen akzeptieren, dass wir uns auch ums Logging kümmern und mit den dortigen Entwicklungen Schritt halten müssen. Wären die bekannten Logging-Frameworks Web-Frameworks, würde man schon längst über deren Bewegungslosigkeit jammern.

Die Bedingungen für unsere Anwendungen verändern sich. Logging passiert nicht mehr nur auf dem Desktop (wenn überhaupt noch). Seit ein paar Jahren ist Logging in der Cloud wichtig. Logging auf mobilen Geräten. Mit den heutigen, teils fast unglaublichen Cloud-Applikationen haben wir eine neue Dimension von Logging erreicht – eine Art „Big Data"-Logging. Vergessen wir nicht: Egal in welcher Umgebung wir uns befinden, unsere Engines sind die Basis aller Logger.

Warum Log4j wiederbelebt wurde

Warum sich eigentlich die Mühe machen und Log4j 2.0 schreiben? Das ist unglaublich viel Aufwand, und immerhin gab es ja schon Logback. Ein nicht unwesentlicher Grund ist natürlich, wie bereits erwähnt, die technische Sicht. Einige Dinge, die in Log4 1.x, aber auch in Logback schlecht liefen, wurden aus Sicht der Entwickler verbessert. Beispielsweise ist es möglich, Log4j Version 2 als Audit-Framework einzusetzen. Logback verliert möglicherweise Log-Events, wenn es neu konfiguriert wird – Log4j Version 2 nicht.

Die neue Architektur basiert auf dem Concurrency-API von Java 5. Damit synchronisiert sich Log4j auf einem möglichst atomaren Level. Log4j Version 1 kämpft mit zahlreichen Deadlocks. Logback ist hier schon etwas besser, arbeitet an einigen Stellen trotzdem nicht atomar genug. So schwerwiegend diese Argumente auch bereits sein mögen: Technik allein war nicht der ausschlaggebende Punkt.

Anfangs wurde schon angesprochen, dass es ein Problem sein kann, wenn jeder eine Stimme hat. In einem harmonischen Team kann das jedoch auch ein unschlagbarer Vorteil sein. Die Motivation, jede Änderung zu sichten und auf die Waagschale zu legen, ist enorm hoch. Gerade große Umbauten müssen gut begründet werden, damit das Team sie akzeptiert. Dadurch steigert sich die Qualität enorm.

Das Projekt bleibt außerdem ständig in Bewegung. Nachdem es keinen Leader gibt, der alle Änderungen absegnen muss, gibt es auch kein Problem, wenn dieser einmal im Urlaub oder krank ist. Gerade wenn es ein kritischer Bug ist, kann schnell reagiert werden. Mit gleichen Rechten kann sich eine starke Community bilden. Genau dies ist geschehen.

Aber auch das gesamte Umfeld ist von Bedeutung. Apache Log4j wird im Rahmen der ASF entwickelt. Damit bleibt der Code immer frei verfügbar. Die Lizenz wird auch immer die Apache-Lizenz bleiben. Als Benutzer kann man davon ausgehen, dass die Software frei von den Ansprüchen Dritter ist, und wer möchte, kann auch mitwirken. Es kann sich zwar niemand einkaufen, aber engagierte Personen werden eingeladen und erhalten eine Stimme im Projekt. Es ist wichtig, diesen Punkt zu erwähnen. Nicht alle Projekte, die Open Source vermarkten, sind auch wirklich bereit, Änderungen von außen anzunehmen oder gar einen Entwickler mit Stimmrecht zu versehen. Dies kann man oft in Zusammenhang mit großen Projekten sehen, die von Firmen betrieben werden.

Die ASF ist also mehr als nur ein großes Repository. Sie bietet eine Plattform für die Community, freie Software-Entwicklung und Schutz für Benutzer und Entwickler.

Das größte Risiko besteht darin, dass ein Projekt verwaist und stillsteht. Allerdings können auch von Firmen getriebene Projekte verwaisen. Wer Software der ASF benutzt, macht sich nicht von einem Anbieter, der möglicherweise sehr restriktiv mit seinem Projekt umgeht, abhängig.

```
@Plugin(
   name = "Sandbox",
   type = "Core", elementType = "appender")
public class SandboxAppender extends AppenderBase {
    private SandboxAppender(String name, Filter filter) {
        super(name, filter, null);
    }

    public void append(LogEvent event) {
        // event.getMessage().getFormattedMessage());
    }

    @PluginFactory
    public static SandboxAppender createAppender(
         @PluginAttr("name") String name,
         @PluginElement("filters") Filter filter) {
         return new SandboxAppender(name, filter);
    }
}
```

Listing 1

Neben der verbesserten Technik, dem schnellen Entwicklungszyklus mit hoher Code-Qualität und dem Schutz der ASF hat jeder der Entwickler eigene Motive aufzuweisen. In meinem Fall: der enorme Spaß- und Lernfaktor, den die Zusammenarbeit mit so talentierten, hochqualifizierten Kollegen mitbringt.

```
<Configuration>
   <Appenders>
      <Sandbox name="myname" />
   </Appenders>
</Configuration>
```

Listing 2

Was neu an Log4j 2.0 ist

Ein paar Neuerungen wurden ja schon genannt. Glücklicherweise spickt das Projekt nicht nur bei den anderen und macht es dann besser, sondern versucht auch selbst Innovationen zu implementieren. Als Beispiel seien hier die neuen asynchronen Logger genannt, die eine sehr gute Performance aufweisen. Diese neuartigen Logger setzen auf den LMAX-Disrupter [5] auf, was ihnen zu einer Super-Power verhilft. Die Performance der Log4j-2-Logger ist um den Faktor zehn besser, verglichen mit den Altbekannten aus Log4j Version 1 und Logback [6]. Ohne diese Logger ist die Performance vergleichbar zu Logback.

Als weitere Neuerung stellt Log4j 2.0 einen Plug-in-Mechanismus bereit. Im Laufe der Zeit wurde klar, dass die Logging-Bedürfnisse manchmal nicht mit dem mitgelieferten Standard-Set an Funktionalität abgedeckt werden können. Zu vielfältig sind die Datenbanken und Use Cases von heute. Log4j 2.0 kann schnell und sauber erweitert werden. Mit folgender Anweisung in der Konfigurationsdatei werden Plug-ins im angegebenen Package gesucht: „<configuration … packages="de.grobmeier.log4j2.plugins">".

Das Plug-in selbst (in diesem Fall ein Appender) ist eine Klasse, die von „AppenderBase" ableitet und mittels Annotations konfiguriert wird. Damit legt man fest, wie das Plug-in später zu verwenden ist. Zu guter Letzt muss noch eine statische Factory-Methode angelegt werden, die das Plug-in erzeugt (siehe Listing 1).

Die Konfiguration hat sich außerdem verändert. Zunächst kann man nun die Namen der Elemente direkt angeben, wie im Beispiel von Listing 2. Neben XML ist nun auch JSON möglich. Das Plug-in könnte etwa so konfiguriert sein.

Ein besonders nettes Detail: Konfigurationen können automatisch neu geladen werden, ähnlich wie in Logback, nur mit dem Unterschied, dass beim Reload der Konfiguration, wie bereits erwähnt, keine Events verloren gehen. Weil wir gerade bei Logback sind: In Log4j 2.0 kann man quasi alle modernen Features erwarten, die auch in Logback gang und gäbe sind. Dazu gehören das Filtern von Log-Events (wie Marker) oder auch die folgende Schreibweise: „logger.debug("Logging with {} is fun!", log4j.getName());". Übrigens kann Log4j 2.0 auch perfekt mit slf4j verbunden werden. Log4j bietet zwar auch ein eigenes API, aber der Bedarf an slf4j-Unterstützung ist einfach nicht zu leugnen.

Houston, wir haben noch immer ein Problem

Wir haben ein freies Logging-Framework der ASF. Es gibt trotzdem noch ein Problem. Immer mehr und mehr Entwickler setzen auf slf4j, um die Logging-Engine darunter austauschbar zu machen. slf4j wird von QOS entwickelt. Jeder Benutzer macht sich damit von der Firma abhängig. Eine Alternative gibt es nicht. Obwohl der Autor selbst im Apache-Commons-Team aktiv ist, kann er Commons-Logging nicht ruhigen Gewissens empfehlen. Logj4 2.0 bietet auch ein API. Aber die ASF kann schwerlich genau das anbieten, was man für so ein wichtiges API benötigt: einen Standard.

Das JDK sollte die Interfaces für das Logging bereitstellen. Apache Log4j und alle anderen Anbieter wären damit austauschbar, weil die Interfaces vom JDK mitgeliefert würden; eine Abhängigkeit fiele weg. Die Unterstützung für diese Interfaces wäre vermutlich enorm.

Wenn es zu so einem Aufwand kommt, sollte auch geklärt werden, ob die Konfiguration standardisiert wird. Und wenn wir gerade dabei sind: Es kann dann auch gleich geklärt werden, wie Logging im Java-EE-Umfeld aussieht.

Tatsächlich gibt es bereits eine Initiative für dieses Szenario, und zwar im Rahmen eines Java.net-Projekts, das hoffentlich irgendwann einmal in einen JSR mündet [7].

Wenn wir erreichen, dass es zu einer Änderung im JDK kommt, sind vermutlich 95 Prozent aller Entwickler ein für alle Mal vom Logging-Ballast befreit. Und die restlichen fünf Prozent könnten sich in aller Ruhe nach dem für sie am besten passenden/geeigneten Logging-Framework umsehen.

Weiterführende Links

[1] http://logging.apache.org/charter.html
[2] http://zeroturnaround.com/rebellabs/the-state-of-logging-in-java-2013/
[3] Ceki Gülcü: Confusing intent and outcome. http://ceki.blogspot.de/2011/11/confusing-intent-and-outcome.html
[4] Ceki Gülcü: Commitocracy as an alternative for conflict resolution in OSS projects. http://ceki.blogspot.de/2010/05/committocracy-as-alternative-for.html
[5] http://lmax-exchange.github.io/disruptor
[6] http://logging.apache.org/log4j/2.x/performance.html
[7] https://java.net/projects/newlogging

Christian Grobmeier
cg@grobmeier.de

Christian Grobmeier ist Chair des Apache-Logging-Services-Projekts und auch in vielen anderen Apache-Projekten aktiv. Ab und zu befüllt er seinen Blog (www.grobmeier.de) oder schreibt an irgendeinem Buch. Manchmal findet er dann auch noch Zeit, seinem eigentlichen Job als freier Software-Entwickler nachzugehen. In seiner wenig vorhandenen Freizeit beschäftigt er sich mit Psychologie und der japanischen Bambuslängsflöte Shakuhachi.