Lieber mit Batch-Framework

Batch-Computing in Java

Phillip Ghadir, Frank Hinkel

Auch heute noch gibt es Bedarf für die wohl älteste Form der elektronischen Datenverarbeitung. Dass Batch immer noch relevant ist, zeigt der junge JSR 352 für Batch-Verarbeitung in Java EE 7. In diesem Beitrag diskutieren wir verschiedene Fragestellungen, die im Zusammenhang mit der Modernisierung bestehender und Umsetzung neuer Batch-Prozesse auftreten, und gehen auf Strategien und Technologieauswahl ein, die bei der Implementierung von Batch in der JVM auftreten können.

Stapellauf

Typische Anwendungsfälle für Batch-Verarbeitung sind Lohnabrechnungsläufe, Zahlungsverkehr, Verarbeitung von LKW-Ladungen an Eingangspost und sonstige terminierte Programmläufe. Diese Anwendungsfälle zeichnen sich dadurch aus, dass Massen von Daten innerhalb einer Interaktion verarbeitet werden müssen.

Batch-Verarbeitung läuft in der Regel vollautomatisch ab. Die Benutzerinteraktion beschränkt sich auf das Starten sowie in Ausnahmefällen auf das Unterbrechen und das erneute Starten. Ansonsten werden Batch-Prozesse zeitgesteuert für Massen von Daten gestartet. Feedback über den Erfolg oder Misserfolg des Programmes erfolgt ausschließlich durch Protokollierung.

Batch-Verarbeitung

Auch heute noch gibt es stichtagsbasierte Vorgänge innerhalb von Unternehmen – wie zum Beispiel das Versenden von Gehaltsabrechnungen. Selbst wenn wir Systeme und Prozesse dank Event-getriebener Architekturen zeitnah asynchron koppeln würden, gäbe es immer noch Geschäftsprozesse, die erst stichtagsbezogen über den Gesamtdatenbestand hinweg abgeschlossen werden können.

In größeren Unternehmen werden jede Nacht Tausende Batch-Programme gestartet, die zum Teil essenzielle Geschäftsprozesse abwickeln.

Viele Unternehmen beziehungsweise Rechenzentren verfügen über umfangreiche Batch-Pläne, die ganze Ketten und Netze von Batch-Programmen definieren. In solchen Unternehmen wäre das Ablösen von Batch-Jobs durch reaktionsfreudige, ereignisbasierte Integrationen der beteiligten IT-Systeme eine zum Teil ausgiebige Re-Engineering-Aufgabe.

Viele Unternehmen berechnen Kennzahlen oder Statistiken auf Basis des Gesamtdatenbestands. Für solche Szenarien ist die Batch-Verarbeitung immer noch üblich. In solchen Fällen ist zumindest einmalig ein Bootstrapping mit den Gesamtdaten erforderlich, selbst wenn danach vielleicht Datenänderungen nicht mehr Batch-orientiert, sondern Event-getrieben verarbeitet werden.

Lokalität der Batch-Verarbeitung

Die Effizienz in der Massendatenverarbeitung hängt stark davon ab, wie nah die Verarbeitung an den Daten selbst erfolgt. Müssen die Daten erst einmal aus einer Datenbank gelesen und in einen Applikationsserver übertragen werden, ist dies tendenziell aufwendiger, als wenn die gleiche Verarbeitung bereits in der Datenbank erledigt werden könnte. Daher vertreten einige die Ansicht, dass alle Massendatenverarbeitungen in der Datenbank oder zumindest im Mainframe ausgeführt werden sollten.

Ohne auf die unzähligen Abwägungen einzugehen, die für oder wider eine Batch-Implementierung auf dem Mainframe oder in der Datenbank sprechen, gibt es in vielen Fällen einen guten Grund für die Implementierung eines Batches in Java: die Nutzung bereits in Java realisierter Verarbeitungslogik.

Auch wenn wir in der Regel nicht die gesamte Fachlogik für die Implementierung eines Batch-Jobs wiederverwenden können, erleichtern bestehende Funktionen die fachliche Identifikation oder partielle Verarbeitung von Daten erheblich. Wir möchten Fachlogik gern – aber eben nicht unreflektiert! – wiederverwenden.

Muss ein Batch-Job mit der Außenwelt interagieren (und das schließt den Mainframe ein), ist dies in der Java VM einfacher als aus dem Datenbanksystem heraus. Integrationsthemen wie Massen-E-Mail-Versand, Dokumenten-Druck oder auch Interaktion mit Um-Systemen über Standardprotokolle sind in Java praktikabel.

Braucht man Batch-Frameworks?

Bei der Diskussion über die Entwicklung eines Batch-Jobs kommt häufig die These auf, dass der Einarbeitungsaufwand in ein weiteres Framework nicht gerechtfertigt sei, da die nötige Funktionalität recht einfach und effektiv selbst implementiert werden könne.

Unserer Erfahrung nach sollte man – wenn überhaupt – nur für triviale Batches auf die Verwendung von Batch-Frameworks verzichten. Bei IT-Systemen mit umfangreicheren beziehungsweise nicht trivialen Batch-Jobs sollte man sich auf die Fachlogik konzentrieren können und nicht zu viel Aufmerksamkeit auf die betriebstechnischen Anforderungen des Batch-Processing legen müssen. Und genau darauf sind Batch-Frameworks ausgelegt!

Vielen Softwareentwicklern ist gar nicht bewusst, wie viele organisatorische Regelungen für die Batch-Verarbeitung in ihrer Organisation existieren: Batches werden standardisiert und überwacht in Rechenzentren ausgeführt. Dem zu genügen,

würde ohne Frameworks eine Menge von Boiler-Plate-Code erfordern.

Frameworks wie Spring Batch bringen eine eigene Infrastruktur mit, die es ermöglicht, „auf Knopfdruck“ ein Modell für die zu verarbeitenden Schritte zu erstellen und zur Laufzeit zu verwalten. Die typischen Anwendungsfälle Restart, Auswertung der ausgesteuerten Einzelsätze usw. werden durch so ein Framework direkt unterstützt.

Komplexere Disziplinen der Batch-Verarbeitung – wie Abbruch beim Auftreten von zu vielen Fehlern in der Verarbeitung, Parallelisierung der Verarbeitung einzelner Schritte, Verkettung einzelner Schritte zu Jobs und Jobs zu Ketten von Batches – werden durch solche Frameworks auch Entwicklern zugänglich, die sich nicht tagtäglich mit den Details der Batch-Verarbeitung beschäftigen.

Gehen wir auf einige Details der Frameworks für die Entwicklung von Batch-Jobs näher ein.

Frameworks

Spring Batch [SpringBatch] ist der Platzhirsch unter den freien Batch-Frameworks. Zudem gibt es seit gut einem Jahr mit JBatch eine finale Spezifikation des JSR 352 [JSR352], deren Grundkonzepte Christoph Schmidt-Casdorff im JavaSPEKTRUM schön beschrieben hat [Sch14].

Die Integration in die Systemlandschaft erfolgt typischerweise über Mechanismen, die nur zum Teil durch das Batch-Framework implementiert sind. Hier bieten beispielsweise Spring Integration [SpringIntegration] und auch Apache Camel [Camel] Implementierungen der Enterprise Integration Patterns [Hohpe03].

Zudem gibt es weitere Batch-Frameworks im Rahmen proprietärer Produkte oder Lösungen, die hier nicht weiter betrachtet werden.

Domänenmodell der Batch-Verarbeitung

Die beiden auf Batch-Verarbeitung spezialisierten Frameworks Spring Batch und JBatch orientieren sich an einem Domänenmodell, das die Begriffe Batch-Job, Batch-Instance, Step und Item unterscheidet. Items – die kleinsten Dateneinheiten in der Batch-Verarbeitung – werden durch ItemReader gelesen, durch ItemProcessors verarbeitet und durch ItemWriter geschrieben.

Ein Batch-Job besteht aus einer Menge von Einzelschritten (Steps), in denen Chunks von Items transaktional verarbeitet werden. Schritte, für die nicht der klassische Ablauf von Lesen → Verarbeiten → Schreiben genügt, können speziell implementiert werden.

JBatch definiert dafür sogenannte BatchLets, deren Pendant in Spring Batch TaskLets heißen. Ein BatchLet/TaskLet unterliegt keinen besonderen Regeln und muss sich selbst um Datenbeschaffung, Transaktionssteuerung und Statusmeldung für die umgebende Laufzeitumgebung kümmern. Damit lassen sich zum Beispiel Transaktionen auf dem Mainframe realisieren.

Batch-Resilience

Da meist sehr große Datenmengen verarbeitet werden, ist eine manuelle Nachbearbeitung auf Einzelsatzebene unerwünscht – wenn nicht gar unmöglich.

Dennoch müssen produktive Batch-Läufe häufig mit fehlerhaften Daten umgehen. Manchmal ist dabei sogar eine manuelle Korrektur mit anschließendem Wiederanlauf erforderlich.

Da Batches viele Daten verarbeiten, deren Alter (und „Lebensweg“) sich völlig unterscheiden können, ist die Wahrscheinlichkeit hoch, dass sie irgendwann inkorrekte Datensätze als Eingabe erhalten. Darin unterscheiden sich Produktionsdaten von Testdatenbeständen erheblich: Ob manuelle Datenkorrekturen durch einen wohlmeinenden 2nd-Level-Supporter vorgenommen wurden oder ob Inkonsistenzen, die nur in einigen unerkannten Grenzfällen auftreten können, im großen Topf der Produktionsdaten gespeichert wurden, der Batch soll dennoch alle Daten bestmöglich – und stets deterministisch – verarbeiten.

Wer einen komplexeren Batch-Job ohne spezielles Batch-Framework schreiben will, muss sich um die unterschiedlichsten Anforderungen aus Betriebssicht selbst kümmern und die gewünschten Fehlerbehandlungsstrategien individuell selbst implementieren. In den Frameworks kann man für einen Batch-Job und einen Batch-Step entsprechende Strategien einfach konfigurieren.

Vollständige Verarbeitung oder Rollback

Im einfachsten Fall existiert für solche Fehlerfälle ein Rollback, der zum letzten konsistenten Stand zurückrollt. Ein solcher Abbruch könnte – nach Korrektur der Datenbasis beziehungsweise der Batch-Implementierung – bei einem Neustart wieder dort aufsetzen, wo der alte Batch-Lauf abgebrochen ist.

Dies ist zwar gängige Praxis, nutzt aber das für den Batch-Lauf zur Verfügung stehende Zeitfenster nicht effektiv aus. Darüber hinaus können abhängige Batch-Programme nicht laufen, sodass potenziell das gesamte Batch-Netz blockiert wird.

Aussteuern und Fortsetzen

Viel effektiver ist das automatische Aussortieren fehlerhafter Datensätze. Die Entwicklung einer solchen Logik ist im Verhältnis zu mancher Fachlogik komplex und muss höchst zuverlässig sein. Verwendet man ein Batch-Framework, muss hier nichts selbst programmiert werden. Es genügt eine einfache Konfiguration – hier am Beispiel JBatch

```
<skippable-exception-classes>
  <include class="com.example.InvalidDataException"/>
</skippable-exception-classes>
```

Die dort eingetragenen Exception-Klassen führen nun nicht mehr zum Abbruch des Programmes, sondern erlauben der Ablaufsteuerung weiterzuarbeiten. Es empfiehlt sich, zusätzlich einen Listener zu konfigurieren, welcher die fehlerhaften Datensätze in einen separaten Output schreibt.

Sollte bei zu großer Fehleranzahl ein Batch-Lauf abgebrochen werden, lässt sich in der Produktionssteuerung ein sogenanntes Skip-Limit definieren, das zum fehlerhaften Abbruch des gesamten Batch-Jobs führt, sobald in einem Durchlauf mehr als die mit dem Skip-Limit definierten Fehler auftreten.

Resilience in verteilten Systemen

In verteilten Umgebungen ist nicht nur die Wahrscheinlichkeit groß, dass Datensätze inkorrekt, sondern auch dass Randsys-

teme temporär nicht verfügbar sind. Im Fall einer kurzen Störung können deshalb automatische Retries sinnvoll sein.

Dabei sollte ein Limit für die Anzahl der Retries gesetzt sowie der Rollback einer Transaktionsklammer verhindert werden. Das lässt sich mit den Batch-Frameworks analog zu den Skippable Exceptions konfigurieren, indem für definierte Exception-Klassen Retries und No-Rollbacks in die JBatch-Konfiguration eingetragen werden

```
<retryable-exception-classes>
  <include class="com.example.MyTimeoutException"/>
</retryable-exception-classes>

<no-rollback-exception-classes>
  <include class="com.example.MyTimeoutException"/>
</no-rollback-exception-classes>
```

Ein Chunk definiert einen für JBatch unteilbaren Datenblock. Dieser wird am Stück innerhalb einer Transaktion verarbeitet. Die folgende Konfiguration definiert, wie groß ein Chunk sein soll oder nach welchem Zeitintervall ein Commit ausgeführt wird

```
<chunk checkpoint-policy="item"
  item-count="1000"
  time-limit="120"
  skip-limit="50"
  retry-limit="20"
/>
```

Die Verarbeitung von Chunks lässt sich einfach konfigurieren. In einem sogenannten Batch-Step lesen ItemReader den Input, den optionale ItemProcessor verarbeiten. Gemäß der Chunk-Konfiguration wird der ItemWriter aufgerufen, der dann den aktuellen Datenblock wegschreibt.

Verwendbare Basis-Komponenten

Eine große Stärke der Batch-Frameworks sind die standardisierten Abstraktionen für die Ready-to-use ItemReader und ItemWriter. Für Standardfälle sollten diese nicht selbst geschrieben werden müssen. Dies trifft für Spring Batch weitgehend zu. Für JBatch muss man – Stand Oktober 2014 – noch einiges selbst implementieren (oder von Spring Batch leihen).

Dennoch sorgen die Standardschnittstellen dafür, dass Entwickler Implementierungen durch einfache Konfiguration wiederverwenden können, die für andere Batches bereits geschrieben wurden.

Als Paradebeispiel lässt sich Spring Batch nennen, welches rund 30 ausgereifte ItemReader bereitstellt. Besonders nennenswert sind hier `FlatFileItemReader`, `HibernateCursorItemReader`, `JdbcCursorItemReader`, `JmsItemReader`, `JpaPagingItemReader`, `MongoItemReader`, `MultiResourceItemReader`, `Neo4jItemReader`, `StaxEventItemReader` und `StoredProcedureItemReader`. Den passenden Satz von ItemWritern liefert Spring Batch ebenfalls aus und erspart Entwicklern somit einen nicht zu unterschätzenden Entwicklungs- und Testaufwand.

In Abbildung 1 ist der Entwurf eines Batch-Jobs dargestellt, welcher unter Verwendung der Ready-to-use Komponenten sequenziell eine Datei einliest und anhand einer Klassifizierungsregel die Daten in unterschiedliche Kanäle aufteilt, um diese im Anschluss in völlig unterschiedliche Zielsysteme zu schreiben: MongoDB, Neo4J, JMS usw.

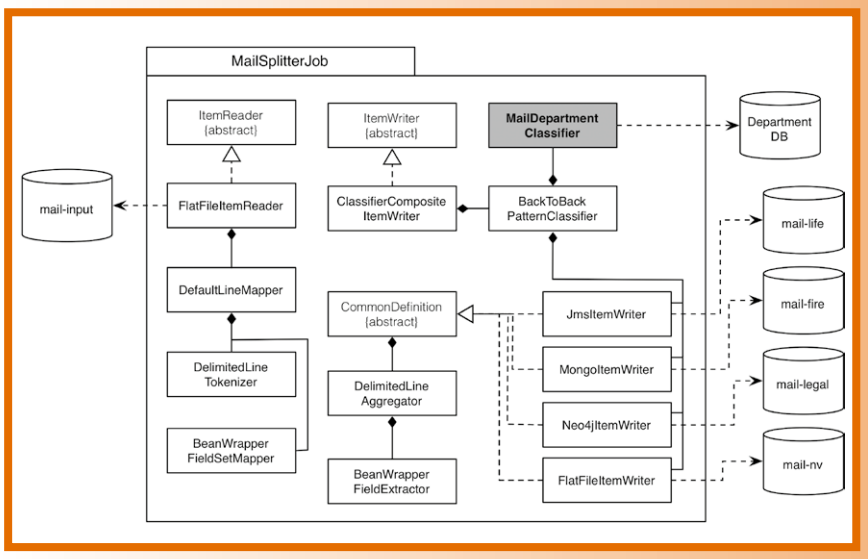

Abb. 1: Mail-Splitter

Hervorzuheben ist einerseits die flexible Konfiguration, welche es ermöglicht, jeden beliebigen Writer oder Reader auszutauschen. Viel erstaunlicher ist jedoch, wie wenig Quellcode für diesen Job nötig ist. Ausschließlich der grau hinterlegte `MailDepartmentClassifier` beinhaltet Programmlogik zur Klassifizierung der Datensätze. Alles Andere sind konfigurierte vordefinierte Spring Batch-Komponenten.

Paralleles Verarbeiten

Sowohl Spring Batch als auch JBatch bieten Möglichkeiten der horizontalen Skalierung. Hier unterscheiden sich die beiden Frameworks jedoch. Bei Spring Batch stehen dem Entwickler vier Skalierungsstrategien zur Verfügung:

- Multithreaded Step
- Parallel Step
- Remote Chunking
- Partitioned Step

Der Multithreaded Step tut genau das, was man von ihm erwartet: Er verarbeitet Chunks parallel in unterschiedlichen Threads. Für das Threading muss in Spring ein Executor konfiguriert werden, wofür sich meist der `ThreadPoolTaskExecutor` anbietet. Dieser erhält eine Pool- und Queue-Größe und übernimmt somit die Verteilung

```
<bean id="taskExecutor"
    class=
    "org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor">
  <property name="corePoolSize" value="5"/>
  <property name="maxPoolSize" value="10"/>
  <property name="queueCapacity" value="25"/>
</bean>
```

Der Parallel Step hingegen führt unterschiedliche Steps eines Jobs parallel aus. Dazu muss zunächst ein Split-Element konfiguriert werden, welches die einzelnen Batch-Steps enthält. Die Split- und Flow-Struktur der Job-Konfiguration hat maßgeblich zu der Spezifikation von JBatch beigetragen. JBatch greift diesen Ansatz auf und führt ihn beispielsweise mit Decision-Elementen und Transitions weiter.

Beim Remote Chunking konfiguriert man zunächst einen Master, welcher für das Lesen und Verteilen von Chunks zuständig ist. Diesem Master dienen 1 bis n Slaves, welche über

einen ItemProcessor und ItemWriter verfügen. Für die Kommunikation zwischen Master und Slave empfiehlt es sich, MessageChannels aus Spring Integration einzusetzen, da diese mit wenig Aufwand für die Chunk-Verteilung konfiguriert werden können.

Die vierte Skalierungsstrategie Partitioned Steps ermöglicht eine feingranulare Skalierung. Um dies umzusetzen, muss ein Partitionierer den Job in kleine ausführbare Einheiten aufteilen und einem PartitionHandler übergeben. Dieser kann entweder lokal oder analog zum Remote Chunking auf anderen Maschinen zur Verfügung stehen.

Betrieb & Scheduling

Positiv erwähnenswert ist, dass sowohl JBatch als auch Spring Batch keinen Applikationsserver benötigen. Batch-Programme, welche eines der beiden Frameworks verwenden, können weiterhin als JAR über die Kommandozeile ausgeführt werden.

Will man Batches einfach bedienen und überwachen können, scheinen auch hier die bereits in vorigen Kolumnen erwähnten Eigenschaften der 12-Factor-Apps [Ghad14] gut zu passen. Im Idealfall ist jeder Batch-Job ein kleines JAR, das über ein simples Skript (bzw. eine standardisierte Schnittstelle) gestartet werden kann. Und wer mag, integriert diese dann in eine Web-Schnittstelle.

So kann der Betrieb die Batch-Jobs einfach in seine Batch-Netze integrieren. Dazu verwendet er eine proprietäre oder standardisierte Job-Steuerung.

Sowohl Spring Batch als auch JBatch bieten ein Job-Repository an, in dem die Definitionen der Batch-Jobs sowie die Historie der Batch-Läufe abgelegt werden. Auf dieses Repository kann nicht nur das Framework selbst programmatisch zugreifen, sondern auch die eigene Implementierung. Dies kann sinnvoll sein, um eine eigene Job-Steuerung anzubinden.

Fazit

Beim Implementieren und Renovieren von Batch-Jobs scheint die Frage zeitlos, ob man wirklich so ein Batch-Framework einsetzen sollte. Der mit solchen Frameworks verbundene Lernaufwand sollte nicht unterschätzt werden. Unserer Erfahrung nach lohnt sich die Einarbeitung, wenn mehrere Batch-Jobs anstehen. Wir sind sogar der Ansicht, dass bereits ein einzelner komplexer Batch-Job den Aufwand rechtfertigt.

Ob man sich für die junge Java EE 7-Variante namens JBatch oder die reife und verbreitete Spring Batch-Variante entscheidet, macht – zumindest Stand heute – einen Unterschied in der Anzahl der vorhandenen ItemReader und ItemWriter. Konzeptionell ist der Unterschied tendenziell gering. Allerdings ist Spring Batch selbst schon mehrere Jahre produktionserprobt.

Besonders wichtig bei der Einführung oder Modernisierung von Batch-Verfahren ist die Zusammenarbeit mit dem operativen Betrieb. Dieser muss in den kritischen Zeitfenstern die Programmläufe begleiten, interpretieren und bedienen können und hat sicherlich bereits Vorstellungen darüber, wie sie einen Batch steuern wollen, die nicht zwingend zu der Java-zentrierten Sicht der Batch-Frameworks passen.

Literatur und Links

[Camel] `http://camel.apache.org/`

[Ghad14] Ph. Ghadir, Micro-Services in Java realisieren – Teil 1, in: JavaSPEKTRUM, 4/2014, `http://www.sigs-datacom.de/fileadmin/user_upload/zeitschriften/js/2014/04/ghadir_JS_04_14_uVvU.pdf`

[Hohpe03] G. Hohpe, Enterprise Integration Patterns, Addison-Wesley, 2003

[JSR352] Java Specification Request 352: Batch Applications for the Java Platform, `https://jcp.org/en/jsr/detail?id=352`

[Sch14] Ch. Schmidt-Casdorff, Batchverarbeitung in JEE7, in: JavaSPEKTRUM, 2/2014, s. a.: `http://www.sigs-datacom.de/fileadmin/user_upload/zeitschriften/js/2014/02/schmidt_casdorff_JS_02_14_8fYh.pdf`

[SpringBatch] `http://projects.spring.io/spring-batch/`

[SpringIntegration] `http://projects.spring.io/spring-integration/`

Phillip Ghadir baut am liebsten tragfähige, langlebige Softwaresysteme. Er ist Mitglied der Geschäftsleitung bei innoQ und hat sich früh auf Architekturen für verteilte, unternehmenskritische Systeme spezialisiert. Darüber hinaus ist er Mitbegründer und aktives Mitglied des iSAQB, des International Software Architecture Qualification Board.
E-Mail: phillip.ghadir@innoq.com

Frank Hinkel ist Senior Consultant bei der innoQ Deutschland GmbH mit über 10 Jahren Erfahrung in der professionellen Softwareentwicklung. Als Berater und Entwickler beschäftigt er sich hauptsächlich mit Themen im Java-Enterprise-Bereich. Seine Schwerpunkte liegen dabei in der Versicherungs- und Telekommunikationsbranche.
E-Mail: frank.hinkel@innoq.com.