Abseits des Hypes

Batchverarbeitung in JEE7

Christoph Schmidt-Casdorff

Batchanwendungen fristen im Java-Biotop ein stiefmütterliches Dasein. Daher gibt es in der Java-Community erst jetzt das Bemühen, auch für diesen Bereich Standards zu schaffen. Batchanwendungen sind aber in vielen Unternehmen das Rückgrat der Informationsverarbeitung und werden für Dateiverarbeitung, Massenverarbeitung oder zeitgesteuerte Hintergrundverarbeitung eingesetzt. Mit JSR 352 existiert jetzt ein Standard, welcher Batchverarbeitung im Umfeld von Java definiert. Der vorliegende Artikel stellt Ihnen die grundlegenden Konzepte vor.

Konzepte der Batchverarbeitung

Unter Batchverarbeitung versteht man im Allgemeinen eine Verarbeitung ohne Nutzerinteraktion. Auf diesem Gebiet tummeln sich bereits so etablierte Mitspieler wie *Apache Camel* [Cam] oder *Spring Integration* [SprI]. Warum ist dann eine neue Spezifikation notwendig? JSR 352 liefert eine andere Sicht auf Batchverarbeitung.

Stellen Sie sich das Szenario einer satzorientierten Verarbeitung einer Datei vor. In dieser Datei finden sich gleichartige Sätze, welche auf gleiche oder zumindest vergleichbare Art behandelt werden sollen. Es wird immer nach gleichem Muster „lesen, verarbeiten, schreiben" vorgegangen. Diese Idee kommt der klassischen Stapelverarbeitung nahe.

Diese spezielle Sicht auf Batchverarbeitung als Stapelverarbeitung ist Grundlage für den JSR 352 und macht es möglich, einheitliche Lösungen für immer wiederkehrende Funktionalität (Querschnittsfunktionalität wie Transaktionsbehandlung, Wiederanlauf oder Nebenläufigkeit) zu realisieren.

Eine Verarbeitungskette (*Job*) kann sich in mehrere Einzelschritte aufteilen, im Sprachgebrauch der Spezifikation *Step* genannt. In einem Step wird eine Menge an *Items* verarbeitet. Items können je nach Anwendungsfall Sätze einer Datei, Zeilen eines Tabellenblatts in Excel, Sätze einer Datenbank oder andere gleichartige Größen ihrer Programmlogik sein. Items werden nach dem beschriebenen Muster „lesen, verarbeiten, schreiben" behandelt.

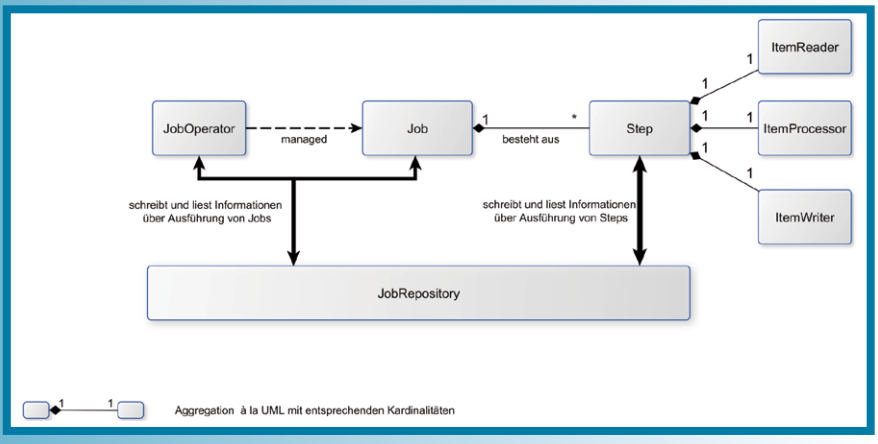

Abb. 1: Grundlegende Architektur eines Batchjobs (Quelle [JSR352])

Architektur

Ein Job setzt sich aus einer Kette von Einzelschritten zusammen, den Steps. Ein Job beschreibt die Steps, legt ihre Verarbeitungsreihenfolge fest und definiert Eigenschaften wie Wiederanlauffähigkeit. Ein Job wird mittels einer XML-Beschreibungssprache *JSL* (*Job Specification Language*) definiert. Diese wird weiter unten genauer betrachtet.

Laufzeitmodell

Ein Job kann mit einem Satz an benannten Parametern gestartet werden. Diese *JobParameter* können zu jedem Start des Jobs neu definiert werden. Ein Job mit einem festgelegten Parametersatz ist eine *JobInstance*. Im Falle eines Abbruchs kann eine solche JobInstance durchaus nochmals gestartet werden. Daher muss zwischen den einzelnen Ausführungen einer JobInstance (*JobExecution*) unterschieden werden.

Die Zentrale zur Verwaltung von Jobs ist der *JobOperator*. Dort werden Jobs gestartet und gegebenenfalls abgebrochen und es können alle Informationen über aktuell verfügbare Jobs, JobInstances und JobExecutions abgerufen werden.

Alle Informationen über Jobs müssen gesichert werden. In dieser Sicherung finden sich die Informationen über die tatsächlichen Jobausführungen, aber auch Informationen zum Wiederanlauf. Diese Aufgabe fällt dem *JobRepository* zu. Über die Struktur und genaue Eigenschaften macht die Spezifikation keine Aussagen. Es wird nur die Existenz eines solchen Repositorys vorgegeben.

Die Ausführung und Koordination der Jobs wird durch die Batchlaufzeitumgebung (*batch runtime*) übernommen. Einen Überblick über die Architektur liefert Abbildung 1. Dort werden insbesondere die logischen Beziehungen der einzelnen Komponenten zum JobRepository beschrieben.

Verarbeitungsschritte – Steps

Abbildung 2 illustriert das oben genannte Verarbeitungsmuster. Items werden einzeln eingelesen (*ItemReader*) und verarbeitet (*ItemProcessor*). Anschließend werden sie zu Gruppen (*Chunks*) zusammengefasst. Diese Gruppen werden schlussendlich durch den *ItemWriter* herausgeschrieben. Eine solche Verarbeitungssequenz ist ein *Step*.

Neben standardmäßigen Kriterien zur Bildung eines Chunks wie Anzahl an Items oder Zeitspanne bis zum Abschluss der Verarbeitung können auch eigene Algorithmen definiert werden.

Der Zusammenhang zwischen Chunks und Transaktionen ist sehr griffig: Transaktionen richten sich nach Chunks. Wird ein neuer Chunk eröffnet, so wird auch eine Transaktion gestartet. Mit der abschließenden Verarbeitung des Chunks im Schreibvorgang wird die Transaktion geschlossen. Diese Form der Verarbeitung heißt *Chunk-orientierte* Verarbeitung und ist die reguläre Verarbeitungslogik von Steps.

Um *Task-orientierte* Schritte wie Initialisierungen oder abschließende Aufräumarbeiten auszuführen, welche nicht in das Korsett einer Chunk-Orientierung passen, bietet die Spezifikation das *Batchlet* an. Ein Batchlet repräsentiert eine Task und kann gestartet und gegebenenfalls gestoppt werden. Es werden keinerlei Annahmen über innere Struktur und Ablauf gemacht. Für Batchlets gibt es keine transaktionale Unterstützung der Batchlaufzeitumgebung.

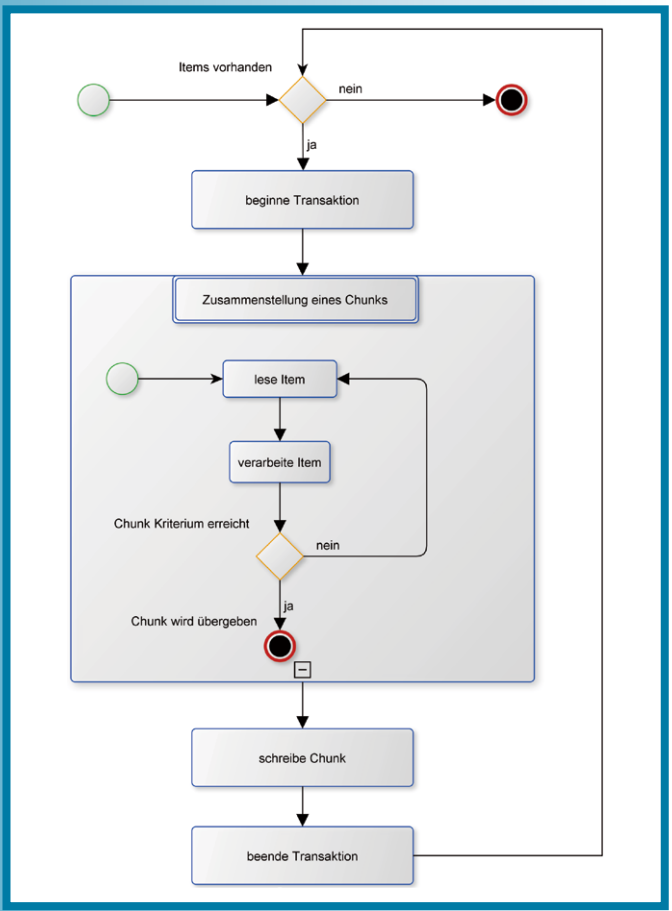

Abb. 2: Verarbeitung innerhalb eines Chunk-orientierten Steps

Ein Step beinhaltet entweder ein Batchlet oder setzt eine Chunk-orientierte Verarbeitung um.

JobContext und StepContext

Wenn ein Step auf seine Metadaten zugreifen oder Daten für den Wiederanlauf sichern möchte, benötigt er Zugriff auf seinen Ausführungskontext. Dieser hält einen speziellen Datenbereich für die aktuelle Ausführung dieses Steps bereit. Insbesondere haben unterschiedliche Steps unterschiedliche Ausführungskontexte. Technologisch wird dieser Ausführungskontext mit dem aus anderen Technologien bekannten Konzept des *Thread-affinen Kontexts* umgesetzt.

Die Batchlaufzeitumgebung stellt sicher, dass für den Step, der im aktuellen Thread ausgeführt wird, der richtige *StepContext* (Ausführungskontext eines Steps) an diesen Thread gebunden wird. Somit wird jedem Step sein eigener Ausführungskontext während der gesamten Verarbeitung zur Verfügung gestellt.

Neben Informationen über Name und Propertys des Steps stellt der StepContext zusätzlich zwei unterschiedliche Datenbereiche zur Verfügung:

- Der Step kann dem StepContext zu *persistierende* Daten übergeben. Diese werden über die Ausführung des Steps hinaus gesichert und beim Wiederanlauf dem Step übergeben.
- *Transiente* (d. h. nicht persistente) Daten gelten nur während der Ausführung eines Steps. Diese Daten werden genutzt, um zwischen unterschiedlichen Komponenten eines Steps Informationen auszutauschen.

Es existiert auch ein Ausführungskontext für den gesamten Job, der sogenannte *JobContext*. Dieser steht allen Steps des Jobs während ihrer Verarbeitung zur Verfügung. Neben den Informationen über den Job und dessen aktuelle Ausführung liefert der JobContext die aktuellen *JobPropertys* (vgl. BatchPropertys weiter unten). Zusätzlich kann er transiente Daten übernehmen. Mit deren Hilfe können Daten zwischen unterschiedlichen Steps ausgetauscht werden.

Der Zugriff auf Step- und JobContext wird durch die Batchlaufzeitumgebung via *Dependency Injection* (DI) bereitgestellt (s. Listing 1). Mit Hilfe des Kontexts können

- spezifische Propertys an den Job bzw. Step übergeben werden,
- Daten zur Laufzeit gesichert und beim Wiederanlauf übergeben werden und
- Daten zwischen und innerhalb von Steps ausgetauscht werden.

```
@Inject JobContext jctxt;
@Inject StepContext sctxt;
```

Listing 1: Zugriff auf Step- und JobContext

Scope und Lebenszyklus

Alle Artefakte, die innerhalb einer Batchverarbeitung instanziiert werden, haben eine definierte Sichtbarkeit (*Scope*). So ist eine Instanz eines ItemReaders nur für seinen aktuellen Step sichtbar. Wird der Step beendet, so ist auch die Instanz des ItemReaders nicht mehr gültig. Sie ist an den Lebenszyklus des Steps gebunden (genauer: an den Lebenszyklus der Instanz des Steps).

Es gibt die drei unterschiedlichen Lebenszyklen des Jobs, des Steps und der Partition (siehe unten). Alle Instanzen von Job-Artefakten sind genau einem dieser Lebenszyklen zugeordnet.

Beschreibungssprache für Jobs – JSL

Zur Konfiguration eines Jobs wurde eine eigene domänenspezifische Sprache entwickelt, die JSL (*Job Specification Language*). Diese basiert auf XML.

Mittels der JSL werden die einzelnen Artefakte des Jobs wie Steps, Chunks oder Batchlets konfiguriert, sowie die Verarbeitungskette des gesamten Jobs festgelegt. Dazu besitzt die JSL Sprachmittel, um den Übergang zwischen Steps zu definieren (*Transitionen*). Dieses Thema wird im Einzelnen noch behandelt.

```
 1 <job id="payroll"
       xmlns="http://xmlns.jcp.org/xml/ns/javaee" version="1.0">
 2   <step id="process">
 3     <properties>
 4       <property name="filestem" value="postings"/>
 5     </properties>
 6     <chunk item-count="5">
 7       <properties>
 8         <property name="xyz" value="#{jobParameters['abc']}"/>
 9         <property name="infile.name"
                    value="#{jobProperties['filestem']}.txt"/>
10       </properties>
11       <reader ref="SimpleItemReader"></reader>
12       <processor ref="SimpleItemProcessor"></processor>
13       <writer ref="SimpleItemWriter"></writer>
14     </chunk>
15   </step>
16 </job>
```

Listing 2: Beispiel für eine Job-Konfiguration mit der JSL

In Listing 2 sehen Sie eine Jobkonfiguration mit genau einem Step. Dessen Chunk hat eine Größe von 5. ItemReader, ItemProcessor und ItemWriter sind explizit zugewiesen.

Jeder Step besitzt eine jobweit eindeutige ID. Diese wird genutzt, um den Verarbeitungsablauf innerhalb eines Jobs zu definieren.

BatchPropertys

BatchPropertys dienen der Konfiguration eines Jobs. Artefakten eines Jobs wie Chunks, Steps oder ItemReader können eigene Propertys zugewiesen werden. Die Werte dieser Propertys können einerseits fest vereinbart oder aber anderseits durch Verweis auf andere Konfigurationsgrößen gewonnen werden.

In Listing 2 wird in Zeile 4 die Property *filesystem* im Kontext des Steps mit der ID *process* definiert und der Wert *postings* gesetzt. In Zeile 9 wird auf diese Property Bezug genommen und deren Wert als neuer Wert der Property *infile.name* genutzt.

Neben Propertys können auch *JobParameter* (s. Zeile 8) oder *SystemPropertys* referenziert werden. Der Verweis auf JobParameter bei der Wahl des Wertes einer Property ist Best Practice, um diese in die Konfiguration einfließen zu lassen.

In Listing 2 wird die Hierarchie innerhalb der Definition eines Jobs deutlich. Ein Step gehört zu genau einem Job, ein ItemReader zu genau einem Step. Die Gültigkeit von Propertys wird durch die Hierarchie in der Jobdefinition bestimmt.

Auf Propertys eines Steps oder Jobs kann mit Hilfe der entsprechenden Ausführungskontexte zugegriffen werden. Andere Artefakte wie ItemReader oder -Processor, welche keinen eigenen Ausführungskontext besitzen, greifen auf ihre Propertys mittels DI zu (s. Listing 3).

```
@Inject @BatchProperty ("filename") String fname;
```

Listing 3: Zugriff auf Propertys mittels DI

Ablaufsteuerung durch Exceptions

Falls in der Verarbeitung von Items gemäß Abbildung 2 eine Ausnahme in Form einer *Exception* auftritt, so werden die aktuelle Transaktion (*rollback*) und die Verarbeitung des gesamten Jobs abgebrochen. Dies ist die standardmäßige Reaktion auf eine Exception.

Soll als Reaktion auf eine Ausnahme allerdings eine nochmalige Verarbeitung der betroffenen Items ohne Abbruch der Verarbeitung ausgelöst werden (*retry*), so ist in der Jobkonfiguration die entsprechende Exception als eine solche zu qualifizieren, welche ein *retry* auslöst. Die Batchlaufzeitumgebung fängt die Ausnahme dann ab und stößt eine entsprechende Wiederholung an. Die JSL liefert eigene Sprachelemente, um Exceptions so zu qualifizieren, dass ihr Auftreten eine Wiederholung der Verarbeitung des Chunks auslöst.

Ebenso können Exceptions so qualifiziert werden, dass ihr Auftreten bewirkt, dass die Verarbeitung des Chunks ignoriert (*skip*) oder ein Rollback verhindert (*no-rollback*) wird.

Das Verhalten im Fehlerfall wird pro Step über Qualifizierung der Exceptions konfiguriert.

Beobachtung des Jobs – Monitoring

Mit Hilfe von *Listenern* kann in einer Vielzahl von Erweiterungspunkten der Fortschritt der Verarbeitung beobachtet werden. So können Listener definiert werden, die entweder über Start bzw. Ende der Ausführung eines Jobs oder Steps oder über die einzelnen Schritte innerhalb einer Chunk-Verarbeitung informiert werden.

Metriken sind ein weiteres Instrument zur Beobachtung eines Steps. Am Ende eines jeden Steps kann über die Metriken die Anzahl der gelesenen/verarbeiteten/geschriebenen Items oder die Anzahl der durchgeführten Commit- oder Rollback-Operationen abgefragt werden.

Ablaufsteuerung eines Jobs

Nebenläufige Verarbeitung – Split

Im Standardfall werden Steps sequenziell und innerhalb eines Threads verarbeitet. Aber die JSL bietet Sprachmittel, um auch andere Verarbeitungsabläufe zu beschreiben.

Der Ablauf von Steps innerhalb eines Jobs heißt *Flow*. Flows können Steps, aber auch weitere Flows enthalten. Insbesondere ist ein Step die einfachste Variante eines Flows. Im Folgenden werden wir anstatt von Steps von Flows sprechen.

Flows können so konfiguriert werden, dass sie nebenläufig ausgeführt werden (*Split*). Jeder Flow wird in einem eigenen Thread ausgeführt. Ein Split ist also eine Gruppe von Flows, die parallel ausgeführt werden. Die Batchlaufzeitumgebung startet die einzelnen Flows. Sind alle Flows beendet, so verzweigt die Batchlaufzeitumgebung zum nachfolgenden Step/Flow. Auf diese Weise wird das Ende der einzelnen Flows synchronisiert und zu einem gemeinsamen Ende zusammengeführt.

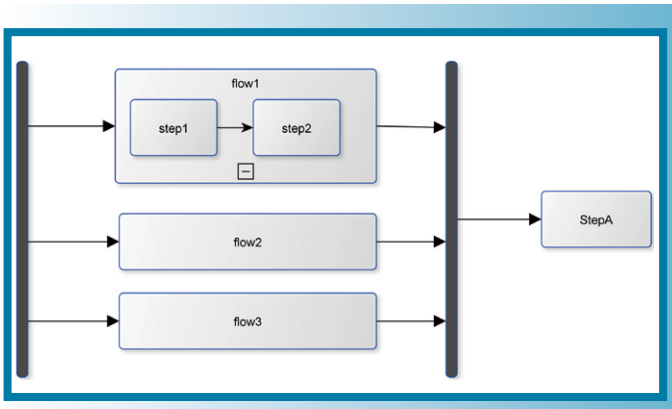

Abb. 3: Verarbeitungsablauf eines Splits (Steps in flow2 und flow3 sind nicht dargestellt)

Abbildung 3 zeigt den Verarbeitungsablauf eines Splits, der aus drei Flows besteht. Innerhalb von *flow1* sind zwei Steps konfiguriert, welche sequenziell verarbeitet werden. Sind alle Flows beendet, so wird zu *StepA* verzweigt.

Dieser Mechanismus erlaubt eine parallele Verarbeitung. Dazu müssen die Arbeitsabläufe, die parallelisiert werden sollen, in jeweils einem Flow beschrieben und diese Flows in einem Split zusammengefasst werden.

Die Batchlaufzeitumgebung ist für das Management der Threads zuständig und entscheidet, welche Flows nebenläufig auszuführen sind. Sie ist ebenso für die Synchronisierung der parallel ablaufenden Splits verantwortlich.

Von Schritt zu Schritt – Transitionen

Transitionen beschreiben, wie die Verarbeitung nach Ende eines Steps weitergeht. Es muss entschieden werden, ob der gesamte Job beendet oder zu einem nachfolgenden Step verzweigt wird.

Ist der Step erfolgreich beendet, so kann mittels des *next*-Attributs der nachfolgende Step vereinbart werden. Fehlt dieser, so wird der Job beendet. Neben dieser Transition für den Positivfall bietet die JSL weitere Sprachmittel, um komplexere Abläufe zu steuern. Diese werten den Status aus, welchen der Step nach seinem Ende besitzt (*step exist status*).

```
<step id="Step1">
  <next on="COMPLETE" to="Step2"/>
  <fail on="FAILED" exit-status="EARLY COMPLETION"/>
</step>
```

Listing 4: Verzweigung des Ablaufs mittels next-Element

Folgende Sprachelemente erweitern die Möglichkeit der Ablaufsteuerung:

- *next*-Element verzweigt zum nächsten Step/Split/Flow (s. Listing 4),
- *fail*-Element beendet den Job und markiert diesen als fehlerhaft beendet (s. Listing 4),
- *stop*-Element beendet den Job und markiert diesen als gestoppt,
- *end*-Element beendet den Job und markiert diesen als ordnungsgemäß beendet.

Darüber hinaus ermöglicht eine *Decision* eine programmatische Verzweigung. Eine Decision kann als Ziel einer Transition angesteuert werden und liefert die ID des nächsten Steps/Flows. Die Ablaufsteuerung wird auf der Ebene der Steps/Flows konfiguriert.

Wiederanlauf

Falls ein Job zu einem beliebigen Zeitpunkt seiner Verarbeitung gewollt oder ungewollt abbricht, muss es möglich sein, diesen Job zu wiederholen und dort aufzusetzen, wo er abgebrochen ist. Daten, die erfolgreich verarbeitet wurden, sollen nicht nochmals untersucht werden.

Die Batchlaufzeitumgebung weiß nichts über die Geschäftslogik und die verarbeiteten Daten. Daher müssen ihr die Wiederanlaufinformationen bereitgestellt werden. Die Aufgabe der Batchlaufzeitumgebung ist:

- diese Informationen im richtigen Moment abzufragen,
- sie zu persistieren, um das Ende des Jobs zu überstehen,
- sie beim Wiederanlauf zu übergeben.

Wiederanlaufinformationen beschreiben den aktuellen Zustand der Verarbeitung, sodass mit ihrer Hilfe deutlich wird, welche Daten bereits verarbeitet wurden und welche nicht. Um den aktuellen Zustand der Verarbeitung eines Steps vollständig zu beschreiben, werden sowohl Informationen über den Stand des Einlesens (*ItemReader*) als auch des Schreibens (*ItemWriter*) benötigt.

Wiederanlaufinformationen sind naturgemäß an Transaktionen gebunden. So überrascht es nicht, dass die Batchlaufzeitumgebung sie unmittelbar vor dem Ende der Transaktion abfragt (s. Abb. 2).

Wird eine JobInstance erneut gestartet und liegen für einen nicht abgeschlossenen Step Wiederanlaufinformationen vor, so erhalten ItemReader und -Writer diese bei der Instanziierung und können sich entsprechend ausrichten.

Partitionierung

Falls die Eingabedaten disjunkt zerlegt (partitioniert) werden können, bietet sich die Möglichkeit, alle Partitionen parallel zu verarbeiten. Beispielsweise können in einer Dateiverarbeitung mehrere Eingangsdateien parallel verarbeitet werden.

Im ersten Schritt sind die Kriterien zur Partitionierung anzugeben. Ein Kriterium kann beispielsweise eine von mehreren Eingangsdateien sein, aber auch einen Postleitzahlrahmen umfassen, für den eine Partition innerhalb einer Adressverarbeitung zuständig ist.

Parallel versus nebenläufig

Sollen Partitionen parallel verarbeitet werden, müssen die Verarbeitungen der einzelnen Partitionen unabhängig voneinander sein. Allerdings bleibt im Gegensatz zur nebenläufigen Verarbeitung mittels Split ein Zusammenhang zwischen den Partitionen erhalten. Eine nebenläufige Ausführung von Steps (Split) lässt solche Steps unabhängig voneinander ausführen, die aus Sicht des Laufzeitsystems nichts miteinander zu tun haben (bis auf die Tatsache, dass sie synchronisiert beendet werden).

Bei der Partitionierung bleibt trotz der parallelen Verarbeitung der einzelnen Partitionen das Gesamtergebnis des Steps im Blick. Stellen Sie sich vor, Sie wollen ein Gesamtprotokoll über die Verarbeitung aller Partitionen erstellen. Die partitionierte Verarbeitung bietet Mechanismen, um die Ergebnisse der einzelnen Partitionen zusammenzuführen.

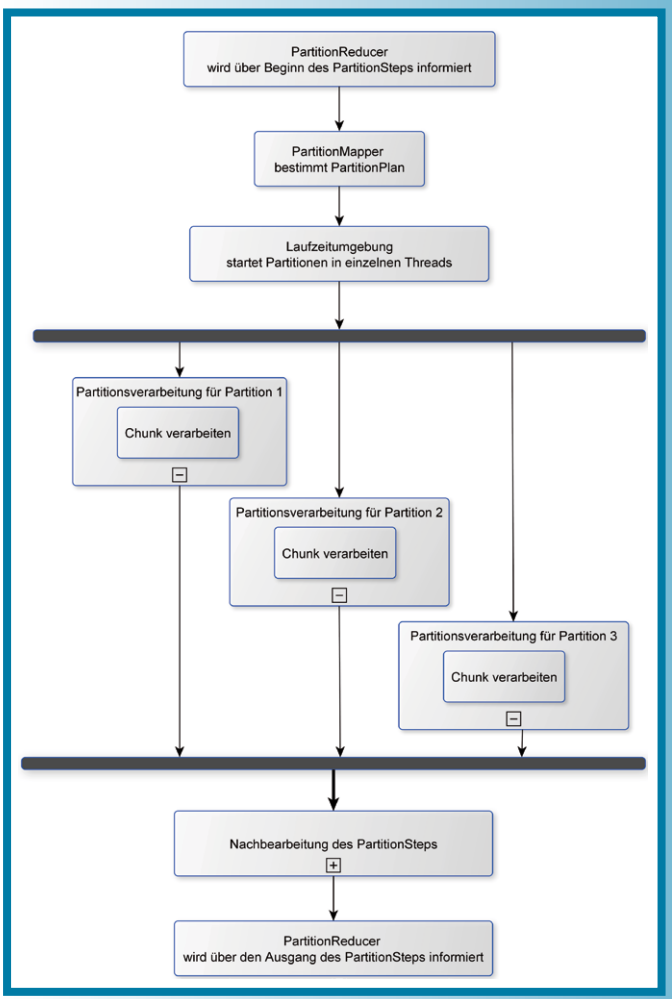

Abb. 4: Verarbeitung eines partitionierten Steps (vereinfachte Darstellung)

Partitionierung – Blick ins Innere

Die Partitionierungskriterien werden durch den *partition plan* bereitgestellt. Dieser kann entweder in der JSL konfiguriert oder aber programmatisch bereitgestellt werden. In beiden Fällen werden Propertys an die Partition übergeben, welche die Kriterien für die Partitionierung zusammenstellen.

Dieses Verfahren setzt voraus, dass jede Partition ihren eigenen separaten Datenbereich besitzt, in dem die für sie spezifischen Propertys bereitstehen. Tatsächlich existiert analog zum StepContext ein *PartitionContext*. Auch benötigt jede Partitionsverarbeitung ihre eigenen Instanzen, beispielsweise des Item-

Reader, ItemProcessor und ItemWriter. Daher existiert auch ein eigener Scope einer Partition, der den Lebenszyklus der Artefakte einer Partition bestimmt.

Aber wie wird der Zusammenhang zwischen den einzelnen Partitionen aufrechterhalten? Wie kann beispielsweise ein Gesamtprotokoll über alle Partitionen erstellt werden? Dazu wird nach jeder Verarbeitung eines Chunks der aktuelle Stand der Verarbeitung für die aktuelle Partition einem sogenannten *PartitionCollector* bereitgestellt. Dieser sammelt die Informationen während der Verarbeitung der Partition und übergibt diese an den für alle Partitionen gemeinsamen *PartitionAnalyzer*. Dort werden diese Informationen zusammengeführt. Mittels dieses Mechanismus kann beispielsweise auch aus den einzelnen Protokollinformationen der Partitionen ein Gesamtprotokoll erstellt werden.

Nachdem die Partition beendet wurde, wird ihr Status an den PartitionAnalyzer übergeben (s. Abb. 4 – Nachbearbeitung). Dieser kann auf den Status reagieren und gegebenenfalls die gesamte Verarbeitung abbrechen.

Implementierungen des Standards

Die Spezifikation JSR 352 beinhaltet wenige Abhängigkeiten zur Infrastruktur, auf der sie umgesetzt wird. Insbesondere finden sich dort keine Abhängigkeiten zu JEE. Es werden keine Anforderungen an Persistenzverfahren, Transaktionsverarbeitung oder Threadmanagement gestellt. Obwohl die Spezifikation Teil des JEE-Standards ist, besitzt sie keinerlei Abhängigkeit zu weiteren JEE-Spezifikationen.

Der Standard verlangt nicht einmal eine allgemeingültige Unterstützung von DI und gibt auch kein bestimmtes Konzept wie *CDI* oder *Spring DI* vor. Es werden lediglich einige wenige Anforderungen an die Unterstützung von DI gestellt, wie die Möglichkeit, Kontexte zu injizieren oder `@BatchProperty` zu unterstützen.

Die Referenzimplementierung [JBatch] nutzt CDI (JEE-Standard für DI) mit der Implementierung [Weld] als Umsetzung von DI. Sie ist grundsätzlich unter Java SE lauffähig, ihre Integration in *Glassfish 4* [GF4] zeigt aber ihre Kompatibilität zu einer JEE-Umgebung.

Mit *Spring Batch* existiert seit Jahren ein Projekt, welches sich mit Batchverarbeitung beschäftigt. Dieses Projekt wird mit der kommenden Version 3 die Spezifikation umsetzen. Spring Batch bringt aufgrund seiner jahrelangen Historie eine Menge an Unterstützung mit. ItemReader und ItemWriter in Spring Batch stimmen zwar nicht in der Signatur der Interfaces mit JSR 352 überein, sind allerdings semantisch eng verwandt. Daher ist es möglich, Adapter für ItemReader und ItemWriter aus Spring Batch zu implementieren, falls diese in JSR 352 genutzt werden sollen (s. [JvS]), um sich so aus dem großen Fundus von Spring Batch zu bedienen.

Der Standard macht keinerlei Aussagen über asynchrone Verarbeitung. Die Ablaufsteuerung ist synchron und gibt keinerlei Aussagen über die Kopplung von Steps. Erst die Integration mit anderen Projekten wie *Apache Camel* oder *Spring Integration* erlaubt die asynchrone Verarbeitung in Spring Batch [B&I].

Ausblicke

Es handelt sich um die Version 1.0 der Spezifikation, sodass – auch nach eigenem Bekunden [JvS] – noch wichtige Aspekte unberücksichtigt sind. So gibt es keinerlei Aussagen über die Verwaltung von Jobs. Es gibt weder normierte externe Schnittstellen wie *JMX* oder *(restful) Webservices* noch spezifizierte Konsolen/Webanwendungen zur Administration. Die Referenzimplementierung in Glassfish 4 bietet eine eigene in Glassfish integrierte Konsole zur Administration von Jobs an. Spring Batch bietet mit dem Projekt *Spring Batch Admin* eine webbasierte Lösung zur Verwaltung von Jobs an. In diesem Punkt scheint es bei anbieterspezifischen Lösungen zu bleiben.

Auch fehlen in JSR 352 jegliche Aussagen zur verteilten Ausführung von Steps, sodass in diesem wichtigen Punkt auf kommende Versionen der Spezifikation zu hoffen ist. Spring Batch adressiert diesen Aspekt durch die Zusammenarbeit von *Spring Batch* und *Spring Integration* und ist damit ein proprietärer Ansatz [SprBI].

Fazit

Die Spezifikation JSR 352 ist möglichst nahe an den Konzepten von Spring Batch geblieben. Wo Spring Batch von JEE abweicht (z. B. Transaktionen, Nebenläufigkeit), sind die Anforderungen so formuliert, dass die Spezifikation nicht von anderen JEE-Spezifikationen abhängt und sowohl in Spring als auch in JEE umgesetzt werden kann.

Mit *JBatch* [JBatch] und in absehbarer Zeit auch mit *Spring Batch 3* [SprB] stehen stabile und bewährte Implementierungen bereit. Während Batchverarbeitung im Umfeld der JEE ein neues Mitglied ist, stellt der Umstieg auf JSR 352 für erfahrene Nutzer von Spring Batch kein großes Hindernis dar.

Links

[B&I] A. Elmore, Building For Performance with Spring Integration & Spring Batch, `http://springone2gx.com/conference/washington/2012/10/video_view?presentationId=27691`

[Cam] Apache Camel™, `http://camel.apache.org/`

[GF4] GlassFish 4, `https://glassfish.java.net/`

[JBatch] Referenzimplementierung, `https://java.net/projects/jbatch`

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

[JvS] J. Fullam, W. Lund, Batch Standardisation JSR-352, `http://springone2gx.com/conference/washington/2012/10/video_view?presentationId=27676`

[SprB] Spring Batch, `http://projects.spring.io/spring-batch/`

[SprBI] Spring Batch Integration, `https://github.com/spring-projects/spring-batch-admin/tree/master/spring-batch-integration`

[SprI] Spring Integration, `http://projects.spring.io/spring-integration/`

[Weld] `http://weld.cdi-spec.org/`

Christoph Schmidt-Casdorff ist als IT-Berater bei der iks Gesellschaft für Informations- und Kommunikationssysteme tätig. Er beschäftigt sich seit mehreren Jahren in großen Kundenprojekten mit dem Thema der modellgetriebenen Softwareentwicklung und mit flexiblen Softwarearchitekturen mit JEE, Spring und OSGi.
E-Mail: c.schmidt-casdorff@iks-gmbh.com