Java Batch

Vorstellungsrunde

- Name
- Rolle im Unternehmen
- Themenbezogene Vorkenntnisse
- Aktuelle Problemsituation
- Individuelle Zielsetzung

Remote Rechner

- Nur Benutzen falls keine eigene Umgebung genutzt werden soll
  - Java 8+
  - IDE (Eclipse, IntelliJ)
  - Zugriff auf Maven über Internet
  -

Zuordnung

tn11.raum01@integrata-cegos.de	3338_tn11	administrator	
tn12.raum01@integrata-cegos.de	3338_tn12	administrator	
tn13.raum01@integrata-cegos.de	3338_tn13	administrator	
tn14.raum01@integrata-cegos.de	3338_tn14	administrator	
tn15.raum01@integrata-cegos.de	3338_tn15	administrator	
tn16.raum01@integrata-cegos.de	3338_tn16	administrator	
Deskmate-Link:	https://integrata-cegos.deskmate.me/		

Support:
Zentrale IT
Telefon:
+49 711 62010 355
ZentraleIT@integrata-cegos.de

Ausgangssituation

Die Programmiersprache Java

integrata cegos

- Variablen eines Programms werden im Hauptspeicher des Java-Prozesses abgelegt
  - Persistierung von Daten muss selbst programmiert sein
    - Serialisierung, Ablage in einer Datenbank
- Ausführung des Programms erfolgt synchron und sequenziell
  - Parallelisierung von Vorgängen ist nicht Bestandteil der elementaren Sprach-Syntax, sondern wird durch Hilfsklassen realisiert
    - executorService.run(() -> {//sequenz1});
    - executorService.run(() -> {//sequenz2});
  - Klassischen Ressourcen-Zugriffe sind **blockierend**
    - var result = entityManager.find(Book.class, “Isbn42”)**;**
      - Reactive Programmierstyle ist auch state of the art
        - entityManager.find(Book.class, “Isbn42”).subscribe((result) -> {//…})
- Fehlerbehandlung mit try-catch-Blöcken

Batch-Programme

- Restart nach Abbrüchen ist die Regel
  - Batch-Variablen sind immer als persistent aufzufassen
- Der gesamte Batch-Lauf muss ausführlich protokolliert werden
  - Eine ausgeführte Batch-Sequenz
    - seq1
    - seq2
    - seq3
  - Damit können Retry-Mechanismen realisiert werden
- Batch-Programm wird immer als ein Workflow aufgefasst

Java und Batch: Wie gut passt das?

integrata cegos

- Das passt direkt gar nicht zusammen
  - genauer: Es erfordert einiges an zusätzlichem Coding, (boilerplate-code)
    - dafür sollten sinnvollerweise Frameworks genutzt werden
  - Batch-Programme in Java
    - Einarbeitung in das Programmier-API des Frameworks notwendig
      - trotzdem irgendwie stilistisch “komisch”
    - Das Framework muss intern einiges an Infrastruktur bereitstellen
      - Daten-Senke
        - Definition der einzelnen Batch-Programme
        - Protokollierung
        - Batch-Variablen

Batch-Programmierung in Java

integrata cegos

- War ursprünglich überhaupt nicht im Fokus der Java Community
- Das erste relevante Batch-Framework in Java wurde als Spring Community-Projekt realisiert
  - Spring Batch
- Java Enterprise Community “fand Spring Batch ganz gut”
  - Java Batch als Fork von Spring Batch
    - Refactoring primär auf Paket- und Typ-Namen
    - Integration in das JEE-Ökosystem
      - insbesondere ist der Applikationsserver eine laufzeitumgebung für Batch-Programme
  - Die Communities haben Batch faktisch nicht sonderlich weiterentwickelt
    - Problemstellung “Batch” hat sich kaum verändert

Spring Batch

integrata cegos

- Core Batch Framework
  - Batch-Runtime
  - Daten-Senke
- Integration in Spring Core
  - Context & Dependency Injection
    - Insbesondere der Zugriff von Batch-Jobs auf beliebige Fachklassen
  - Zusätzliche Scopes
    - Spring Core:
      - singleton, prototype
    - Spring MVC für Web-Anwendungen
      - session, request, page, view, flow
    - Spring Batch
      - job, step
- Bereitstellung von Utilities

es gibt eigentlich keine Gründe, Java Batch einzusetzen

Spring Batch ist “unvollständig”

integrata cegos

- Insbesondere die Anzahl der Utilities ist ziemlich beschränkt
  - Dateien, Datenbanken, Messaging-Systeme
- Es fehlt
  - Scheduling
  - Job-Ausführungen direkt über http-Requests
  - Zugriffe auf z.B. NoSql-Datenbanken
- Dafür ist das Projekt “Spring Integration” gedacht

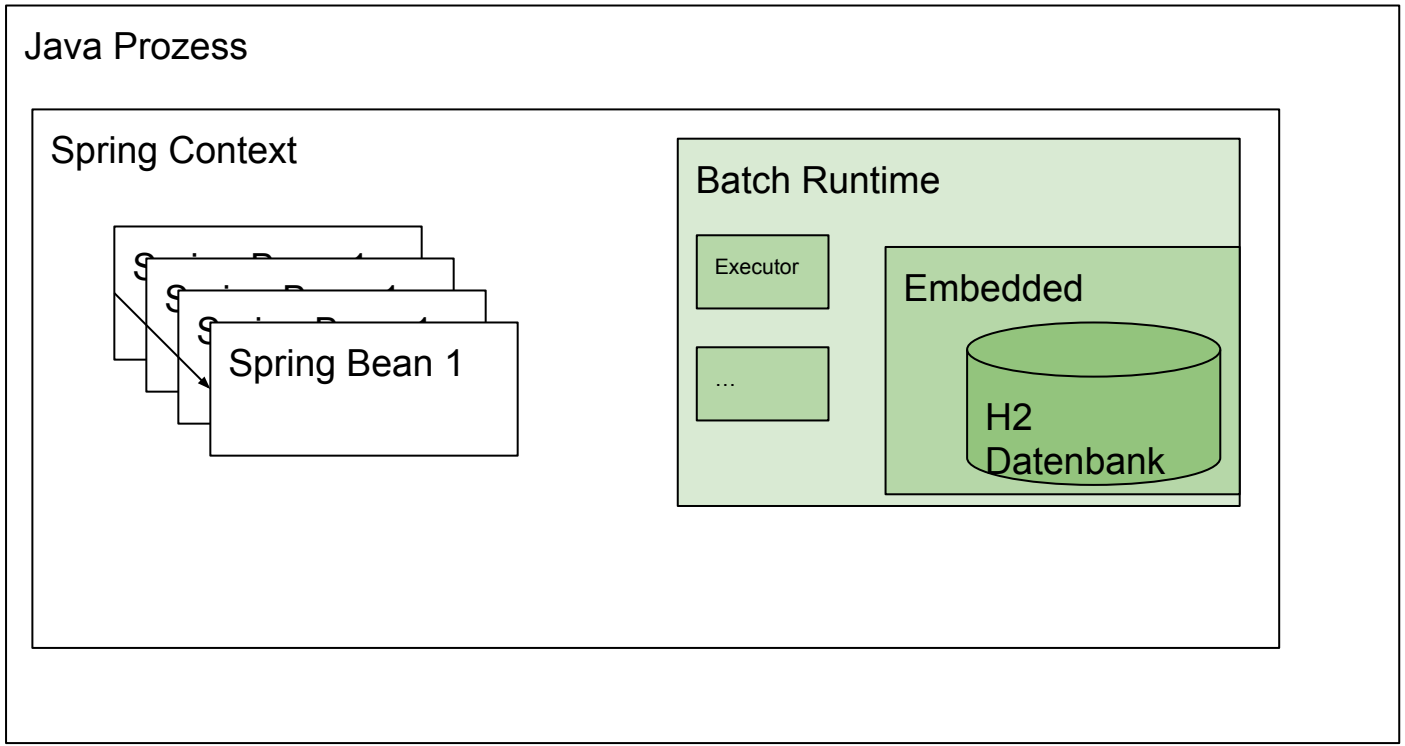
Spring Batch: First Contact

Unser Projekt auf Basis von Spring Boot

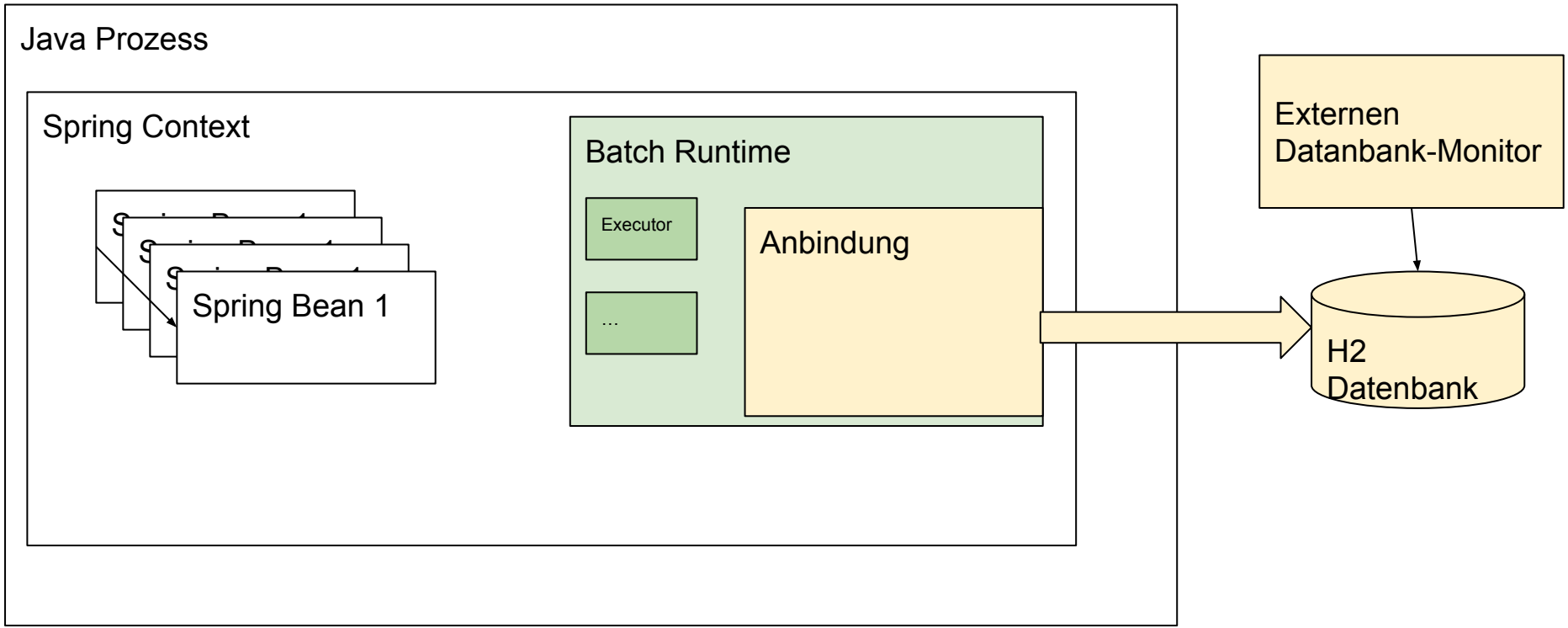
integrata cegos

- Dependency Management mit Maven Parent und Startern
- SpringBootApplication
- Testen über SpringBootTest
- AutoConfiguration mit der Möglichkeit der Umkonfiguration
  - application.properties, -.yaml
- Bereitstellung einer vollständig funktionsfähigen Infrastruktur

Die Infrastruktur von Spring Batch



Die angepasste Infrastruktur von Spring Batch



Start der external H2

```
java -cp ~/.m2/repository/com/h2database/h2/2.1.212/h2-2.1.212.jar
org.h2.tools.Server -web -webAllowOthers -tcp -tcpAllowOthers
-ifNotExists
```

Ein erster Job

- Unsere Sequenz
  - System.out.println(“Hello World”);
- Programmiermodell
  - Ein Tasklet ist eine Hüllklasse um eine beliebige Java-Sequenz

Batch im Detail

Job

- Job
  - Abstrakt
    - Vergleichbar eine Java-Klasse
- Job Execution
  - Eine konkrete “Instanz” eines Jobs
  - Zu Laufzeit erreichbar über eine Referenz
  - Identifikation in der Datenbank über eine eindeutige Id
- Die Job Execution Id
  - Fachlich bestimmt aus
    - Job-Name
    - Untermenge sogenannter Job-Parameter

Step

- Step
  - Abstrakt
    - Vergleichbar eine Java-Klasse
- Step Execution
  - Eine konkrete “Instanz” eines Step
  - Zu Laufzeit erreichbar über eine Referenz
  - Identifikation in der Datenbank über eine eindeutige Id
- Ein Step wird einem Job zugeordnet
  - Job ist ein Container für Steps

Spring Boot Application

- Bisher:
  - besteht aus einer einzigen eindeutigen Job-Definition
  - diese wird automatisch gestartet, sobald die Anwendung gestartet wird
- Nun
  - spring.batch.job.enabled=false
  - Programmatisches Starten über eine Job-Launcher
    - launcher.run(helloWorldJob, jobParameters);
  - JobLauncher-Instanz ist Bestandteil des Spring-Context und kann deshalb autowired werden
- Trainings-Umgebung
  - Aufruf eines Jobs erfolgt über einen REST-Aufruf angesteuert über eine Web-Anwendung

Schritt für Schritt

integrata cegos

- Neue Dependencies im pom.xml
  - spring-starter-web
  - openapi-ui
  - CHECK
    - Nach Starten der Anwendung läuft ein WebServer
    - application.properties: server.port=9090
    - http://localhost:9090/swagger-ui.html
- Umstellung auf die embedded Datenbank
  - application.properties
    - mem-Jdbc-Url ist aktiv
    - spring.h2.console.enabled=true
    - Check
      - http://localhost:9090/h2-console
      - Anmeldung mit den credentials und der Datenbank-URL der application.properties

Schritt für Schritt

- Hinzufügen einer Daten-Klasse
  - JobRequest
    - Der Name des auszuführenden Jobs
    - Map mit key-Value-Paaren, die als Job-Parameter genutzt werden sollen
- Hinzufügen eines RestControllers
  - Dieser dispatched programmatisch den Job Request auf eine vorhandene Job-Instanz

ToDo

- Schreiben Sie eine neue Job-Definition in einer eigenen @Configuration-Klasse!
- Nehmen Sie diesen in das Dispatching im RestController auf!
- Testen Sie den Zugriff über Web und das Auftreten der erwarteten Ausgaben in den Datenbank-Tabellen

Sequenz innerhalb eines Jobs

- Tasklet mit mehr als einer Java-Anweisung
- Chunk-Verarbeitung (-> Morgen)
  - ItemReader
    - liefert Daten
  - ItemProcessor
    - verarbeitet Daten
  - ItemWriter
    - schreibt Daten
- Definition eines Jobst mit mehr als einem Step

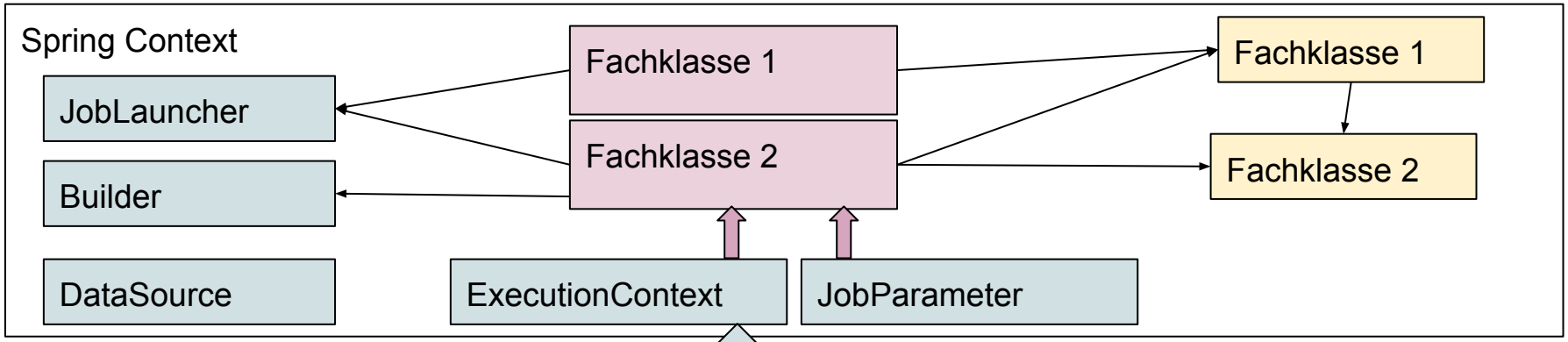
Hinweis

- Arbeiten mit dem ChunkContext wird sich in der Praxis einer Spring Batch-Anwendung auf wenige Ausnahmen reduzieren
  - StepContext und JobContext werden als entsprechende “gescopete” Spring Beans injected werden

Todo

- Nehmen Sie als Grundlage die SequenceConfiguration auf GitHub
- Integrieren Sie diese in den RestController
- Step1 schreibt etwas in den StepContext -> CHECK Tabelle STEP-EXECUTION_CONTEXT
- Step1 liest aus den JobParametern -> CHECK: Das Auslesen funktioniert
- Experimentieren Sie mit z.B.
  - step3 liest einen Job Parameter aus
  - …

Stand



Infrastruktur

Read-Only Injection über @Value

Beliebige Fachklassen
+ singleton

Batch-relevant
+ JobScope
+ StepScope

Relationale Datenbank
+ Schema der Batch-Tabellen

Spring Batch

- Stellt ein Programmiermodell bereit
  - Zentrale Komponente: “Tasklet”

Nächste Schritte

- Definition eines Workflows
  - start -> next -> next -> end
  - Umgang mit Fehlern
    - retry
    - skip-Logik
- Einführung eines speziellen Tasklets: “Chunk”
  - Abbildung einer klassischen Datenverarbeitung
    - read -> process - > write
- Job-Listener
- Umstellen bzw. Ergänzen mit XML statt @Configuration

Chunk-Verarbeitung

Programmiermodell

- Interfaces
  - ItemReader
  - ItemProcessor
  - ItemWriter
- Reader und Processor arbeiten jeweils pro Datensatz
- Writer arbeitet auf einer Ergebnis-Liste
  - Größe der Ergebnis-Liste: commit
  - Im Endeffekt ist das ein Parameter zur Optimierung

Chunk aus Sicht von Spring Batch

- Nichts anderes als ein Tasklet
  - Verwechseln bitte nicht Reader, Processor, Writer mit Steps!
-

Chunk aus fachlicher Sicht

- Ein Verarbeitungsschritt
- Damit ist es durchaus möglich, dass die einzelnen Implementierungen Daten austauschen sollen / müssen
  - Ausschließlich “in memory”
  - persistieren über den ExecutionContext

Lösung mit CDI und Spring Batch

- Datenaustausch-Klasse als Spring-Komponente z.B. im StepScope
  - Interne Struktur beinhaltet die auszutauschenden Daten

Todo

- Frage:
  - Könnte / sollte das SimpleData auch JobScope haben?
    - Sinn in diesem Beispiel?
    - Sinn allgemein?
- Frage:
  - Sind die Daten von SimpleData Bestandteil des ExecutionContext?
    - Hinweis: Schauen Sie mal in den Tabellen nach!
- Recherche
  - Was wäre zu tun, um das SimpleData zu persistieren?
    - Hinweis: Es gibt z.B. einen StepExecutionListener

Fehlerbehandlung im Job

integrata cegos

- Ausgerichtet auf das Werfen von Exceptions
- Verhalten des Jobs?
  - Ein fehlgeschlagener Step führt zu einem Abbruch des Jobs
    - Wird protokolliert
  - Ein fehlgeschlagener Job kann wiederholt werden
    - VORSICHT
      - JobLauncher kann das, aber nur wenn sich die Parameter nicht geändert haben!
      - Alternative: JobExplorer, eine Utility, einen Job über seine Id identifiziert
  - Step-Konfiguration
    - retry-limit
      - Maximal-Anzahl der retry-Versuche
    - skip-Limit
      - Anzahl der Lesevorgänge, die eine Exception werfen dürfen

ToDo

- Skip-Beispiel
- Zusätzlich
  - Idee
    - Der Writer schreibt seine erfolgreich geschriebenen Daten weg
    - Reader überliest die bereits verarbeiteten Daten
    - Hinweis:
      - Die fertigen Reader/Writer-Implementierungen von Spring Batch machen dies bereits

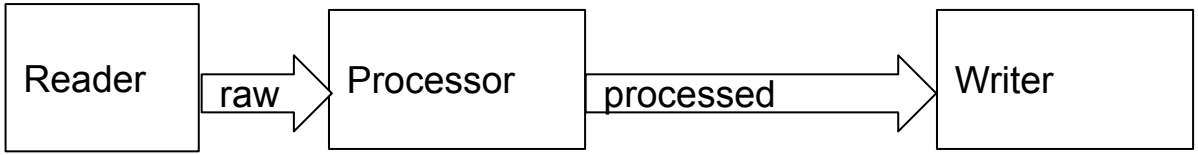
Reader / Writer-Implementierungen von Spring Batch

Übersicht

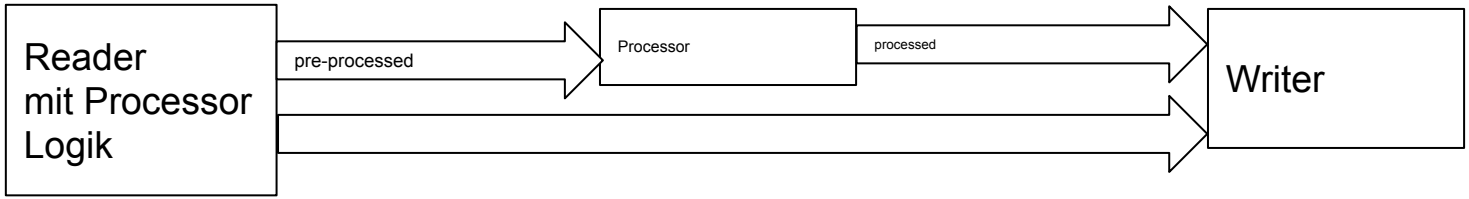
org.springframework.batch.item
org.springframework.batch.item.adapter
org.springframework.batch.item.amqp
org.springframework.batch.item.amqp.builder
org.springframework.batch.item.avro
org.springframework.batch.item.avro.builder
org.springframework.batch.item.data
org.springframework.batch.item.data.builder
org.springframework.batch.item.database
org.springframework.batch.item.database.builder
org.springframework.batch.item.database.orm
org.springframework.batch.item.database.support
org.springframework.batch.item.file
org.springframework.batch.item.file.builder
org.springframework.batch.item.file.mapping
org.springframework.batch.item.file.separator
org.springframework.batch.item.file.transform
org.springframework.batch.item.function
org.springframework.batch.item.jms
org.springframework.batch.item.jms.builder
org.springframework.batch.item.json
org.springframework.batch.item.json.builder
org.springframework.batch.item.kafka
org.springframework.batch.item.kafka.builder
org.springframework.batch.item.ldif
org.springframework.batch.item.ldif.builder
org.springframework.batch.item.mail
org.springframework.batch.item.mail.builder
org.springframework.batch.item.mail.javamail
org.springframework.batch.item.support
org.springframework.batch.item.support.builder
org.springframework.batch.item.util
org.springframework.batch.item.validator
org.springframework.batch.item.xml
org.springframework.batch.item.xml.builder
org.springframework.batch.item.xml.stax

Hinweis

- Implementierungen der Spring Batch Community trennen nicht konsequent den Reader vom Processor

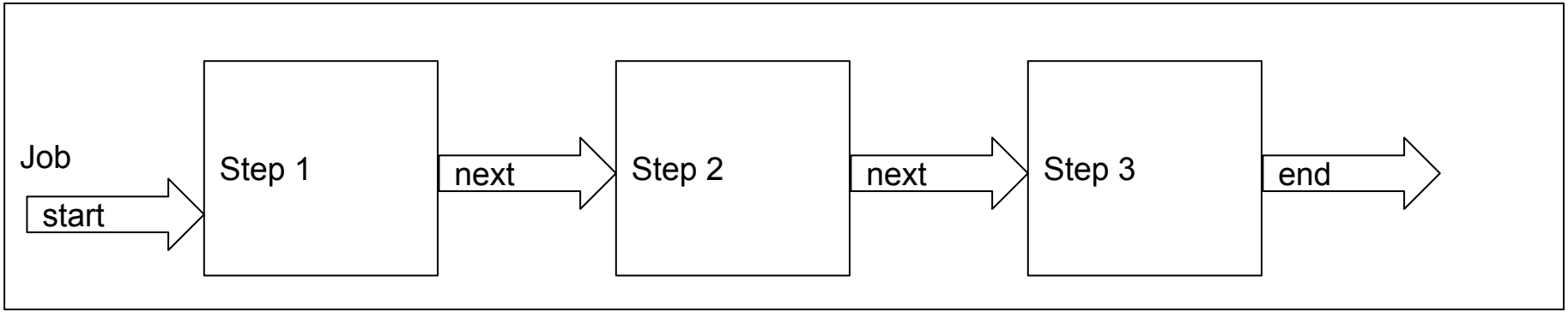


  - Reader enthalten bereits Processor-Logik

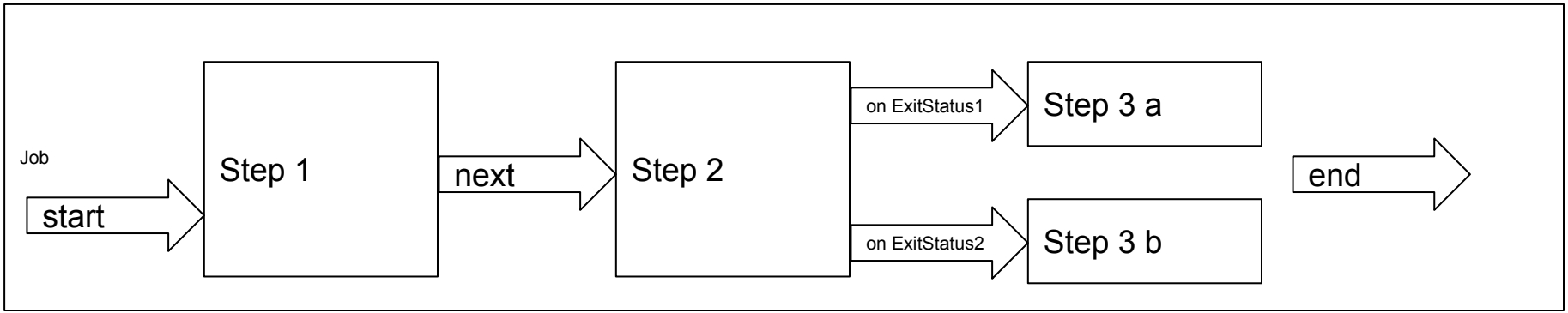


Batch-Workflows

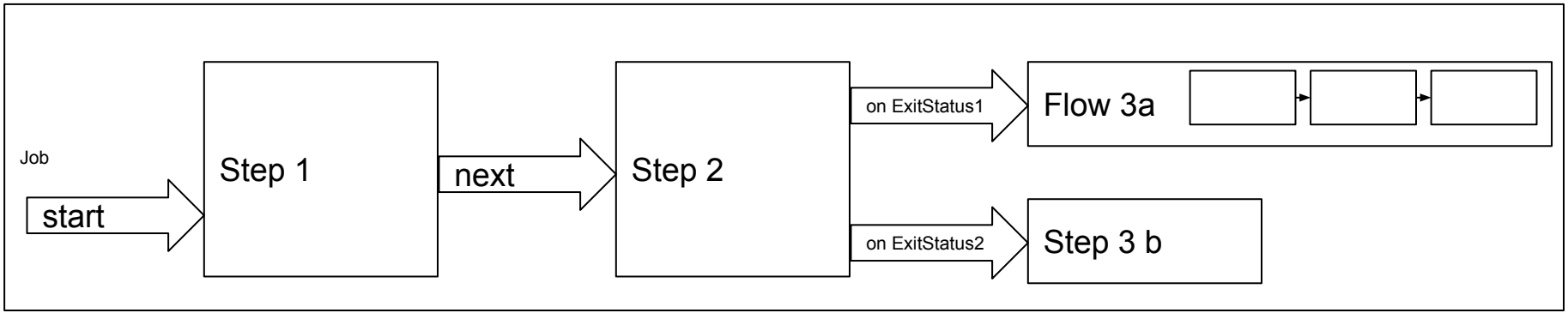
Simpler Sequence Workflow



Conditional Sequence Workflow



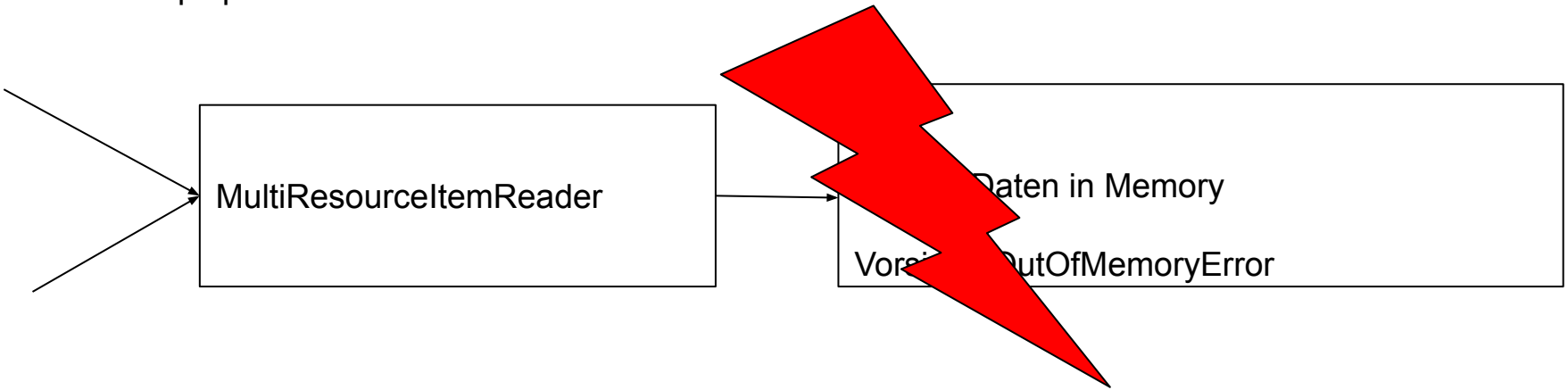
Conditional Sequence Workflow mit Flows



Beispiel: Aggregation von zwei Eingangsdateien

integrata cegos

Ein Job ist State-behaftet, aber beschränkt auf den Hauptspeicher



Beispiel: Aggregation von zwei Eingangsdateien

integrata cegos

Ein Job ist State-behaftet, aber beschränkt auf den Hauptspeicher