JSR 352 Expert Group

Working Session
2 March 2012
Agenda

- Review: Readers/Writers
- Discussion: Listeners
- First Look: Concurrency – the Models
- List for Next Meeting
Annotations

@ItemReader (or @ItemWriter)
public class MyReader {
    @Parameter String fileName = null;
    @Open void open(CheckpointInfo chkpt) {…}
    @Close void close() {…}
    @ReadItem {Type} read() {…}
    @GetCheckpointInfo CheckpointInfo getCheckpt() {…}
}

@WriteItem void write({Type} record) {…}
Usage

@Step(name="Postings")
public class PostingsStep {

    @ItemReader FileReader reader;
    @ItemWriter FileWriter writer;
    @ItemProcessor {Type1} process({Type2} rec) {…}

}

Defining reader/writer as field allows natural use of DI, e.g. CDI example with qualifier:

@Inject @TestMode
    @ItemReader FileReader reader;
Discussion: Listeners

- **Spring Batch Listeners**
  - JobExecutionListener
  - StepExecutionListener
  - ChunkListener
  - ItemReadListener\(<T>\)
  - ItemProcessListener\(<T, S>\)
  - ItemWriteListener\(<S>\)
  - SkipListener\(<T,S>\)
  - RepeatListener
  - RetryListener

- **WebSphere Batch Listeners**
  - JobListener (combines Job/StepExecutionListeners)
  - SkipListener
  - RetryListener
  - CheckpointListener (i.e. ChunkListener)
**Discussion: Listeners**

- **JobExecutionListener**
  ```java
  @Job(name="Job1")
  public class MyJob {
    @BeforeJob void before();
    @AfterJob void after();
  }
  ```

- **StepExecutionListener**
  ```java
  @Step(name="Step1")
  public class MyStep {
    @BeforeStep void before();
    @AfterStep void after();
  }
  ```

How do listeners know identity? I.e. job name and step name.

Job name and current step name could be exposed in runtime “batch context” object.
Discussion: Listeners

- **ChunkListener**
  ```java
  @Step
  public class MyStep {
      @BeforeCheckpoint void before();
      @AfterCheckpoint void after();
  }
  ```

- **ItemReadListener<T>**
  ```java
  @Step
  public class MyStep {
      @BeforeRead void beforeRead();
      @AfterRead void afterRead({Type} item);
      @OnReadError void onReadError(Exception ex);
  }
  ```
ItemProcessListener<T, S>

@Step
public class MyStep {
    @BeforeProcess void beforeProcess({Type} item);
    @AfterProcess void afterProcess({Type} item, {Type2} result);
    @OnProcessError void onProcessError(Exception ex, {Type} item);
}

ItemWriteListener<S>

@Step
public class MyStep {
    @BeforeWrite void beforeWrite(List<{Type}> items);
    @AfterWrite void afterWrite(List<{Type}> items);
    @OnWriteError void onWriteError(Exception ex, List<{Type}> items);
}
Discussion: Listeners

- **SkipListener<T,S>**
  ```java
  @Step
  public class MyStep {
      @OnSkipInRead void onSkipInRead(Throwable t);
      @OnSkipInProcess void onSkipInProcess({Type} item, Throwable t);
      @OnSkipInWrite void onSkipInWrite({Type2} result, Throwable t);
  }
  ```

- **RepeatListener**
  ```java
  @Step
  public class MyStep {
      @BeforeRepeat void beforeRepeat(RepeatContext context);
      @AfterRepeat void afterRepeat(RepeatContext context, RepeatStatus result);
      @OpenRepeat void openRepeat(RepeatContext context);
      @OnErrorRepeat void onErrorRepeat(RepeatContext context, Throwable e);
      @CloseRepeat closeRepeat(RepeatContext context);
  }
  ```
@Step
public class MyStep {
    @OpenRetry void openRetry(RetryContext context,
                               RetryCallback<Type> callback);
    @OnErrorRetry void onErrorRetry(RetryContext context,
                                     RetryCallback<Type> callback, Throwable);
    @CloseRetry void closeRetry(RetryContext context,
                                 RetryCallback<Type> callback, Throwable e);
}
First Look: Concurrency

Parallel Step

Thread1 → @ReadItem

Thread2 → @ProcessItem

... → @ProcessItem

ThreadN → @ProcessItem

@WriteItem

Slave threads could be remoted.

N threads

M items per write
Concurrent Steps, Single JVM

Thread1

Step1

Thread2

Step2

Step1 and Step2 started at same time on separate threads
First Look: Concurrency

Concurrent and Parallel can be combined

Step 1
- Thread1
  - @ReadItem
  - Thread2
    - @ProcessItem
  - ...
  - ThreadN
    - @ProcessItem
  - @WriteItem

Step 2
- Thread1
  - @ReadItem
  - Thread2
    - @ProcessItem
  - ...
  - ThreadN
    - @ProcessItem
  - @WriteItem
First Look: Concurrency

Concurrent Steps, Multiple JVMs

Step1 and Step2 started at the same time on separate threads, in different JVMs.
Partitioned Step, Single JVM

Multiple instances of Step1 started at same time on separate threads
First Look: Concurrency

Partitioned and parallel can be combined.

Multiple instances of Step1 started at same time on separate threads
First Look: Concurrency

Partitioned Step, Multiple JVMs

Multiple instances of Step1 started together on separate threads in different JVMs
First Look: Concurrency

Concurrent Partitioned Steps, Single JVM

Each partitioned step run on its own set of threads. The number of threads per step may be different.

Partitioned Step1
- Thread 1
- Step 1
- ... 
- Thread N
- Step 1

Partitioned Step2
- Thread N+1
- Step 2
- ... 
- Thread N+M
- Step 2

Partitioned Step1 and Partitioned Step2 are started together.
Each partitioned step run on its own set of threads. The number of threads per step may be different. Threads are in different JVMs.

Partitioned Step1

Partitioned Step2

Partitioned Step1 and Partitioned Step2 are started together.
Each pair of concurrent steps is run in partitioned in the same JVM. The number of threads per step may be different.

Partitioned Concurrent Steps, Single JVM

Concurrent steps Step1, Step2

Partitioned Concurrent Steps, Single JVM

Steps Step1 and Step2 are started together as pairs.
Each pair of concurrent steps is run in partitioned across multiple JVMs. The key requirement is same-JVM proximity for the concurrent steps, and then partitioning of the concurrent pairs.
Each threads run the same sequence of steps.
Partitioned Sequential Steps, Multiple JVMs

Each thread runs the same sequence of steps, each sequence runs in separate JVMs.

Partitioned Sequences Step1 and Step2 are started together.
List for Next Meeting

- Repeat
- Retry
- More on Concurrency
- Future
  - Exit codes
  - Step conditions
  - Execution Context
  - Metrics
  - Java EE