JSR 352 Expert Group

Working Session

7 March 2012
Agenda

- Review: Lifecycle Events vs Listeners
- Review: Concurrency Parallel Models
- Discussion: Repeat, Retry, Skip
- List for Next Meeting
From public mailing list, Proposed lifecycle events per batch artifact:

@Job
  @BeginJob
  @EndJob
@Step
  @BeginStep
  @Process or @ProcessItem
  @EndStep
@ItemReader
  @Open
  @ReadItem
  @Close
  @GetCheckpointInfo
@ItemWriter
  @Open
  @WriteItem
  @Close
  @GetCheckpointInfo
Proposed listeners:

- JobListener
- StepListener
- CheckpointListener
- SkipListener
- RetryListener

Jury still out on these listeners;

- ItemReadListener
- ItemProcessListener
- ItemWriteListener
- RepeatListener
Listeners and their subordinate annotations:

@JobListener
   @BeforeJob
   @AfterJob
@StepListener
   @BeforeJob
   @AfterJob
@CheckpointListener
   @BeforeCheckpoint
   @AfterCheckpoint
@SkipListener
   @OnSkipInRead
   @OnSkipInProcess
   @OnSkipInWrite
@RetryListener
   @BeforeRetry
   @AfterRetry
Example JobListener:

```java
@JobListener
// Count and record jobs
public class CountJobs {
    @BeforeJob public void before() {...}
    @AfterJob public void after() {...}
}
```

Attach @JobListener to job:

```java
@Job(name="Job1")
public class TestJob {
    @JobListener CountJobs jobCounter;
    ....
}
```

The same construction pattern would apply to each listener. Any listener can be attached to any artifact - i.e. job, step, reader, writer.
Review: Parallel Models

Pipeline Step, Single JVM

Thread1 → @ReadItem

Thread2 → @ProcessItem

ThreadN → @ProcessItem

pipeline

pipeline

@WriteItem

N threads

M items per write
Review: Parallel Models

Pipeline Step, Multiple JVMs

Thread1 @ReadItem
pipeline

Thread2 @ProcessItem

pipeline

Thread3 @ProcessItem
JVM

ThreadN @ProcessItem
JVM

@WriteItem
Review: Parallel Models

Concurrent Steps, Single JVM

Thread1

Step1

Thread2

Step2

Step1 and Step2 started at same time on separate threads
Concurrent and Pipeline can be combined
Review: Parallel Models

Concurrent Steps, Multiple JVMs

Step 1 and Step 2 started at the same time on separate threads, in different JVMs.
Multiple instances of Step 1 started at the same time on separate threads
Review: Parallel Models

Partitioned and pipeline can be combined.

Multiple instances of Step1 started at same time on separate threads
Partitioned Step, Multiple JVMs

Multiple instances of Step1 started together on separate threads in different JVMs
Review: Parallel Models

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 runs 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.

Concurrent steps Step1, Step2  Concurrent steps Step1, Step2

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.

Steps Step1 and Step2 are started together as pairs across multiple JVMs.
Review: Parallel Models

Partitioned Sequential Steps, Single JVM

Each thread runs the same sequence of steps.

Thread1

1

Step1

2

Step2

Thread2

1

Step1

2

Step2
Each thread runs the same sequence of steps, each sequence runs in separate JVMs.
What is repeatable?

- Item processing iteration

  e.g. @ItemProcessor <S> process(<T> item) until no more items or terminating exception

- User-defined iteration?

  e.g. @Process RepeatStatus process()
@ItemProcessor is transactional

e.g. each checkpoint (chunk) is a transaction

Should @Process be

  (optionally) transactional?

e.g. @Process(transactional=true) process()

allow checkpoints?

  e.g.

  @Process RepeatStatus process() {
      return RepeatStatus.CHECKPOINT_AND_CONTINUE;
  }
Do we need a Repeat Listener?

```java
@RepeatListener
public class MyRepeatListener {
    @BeforeRepeat void before(RepeatContext ctx) {}
    @AfterRepeat void after(RepeatContext ctx, RepeatStatus s) {}
    @OpenRepeat void open(RepeatContext ctx) {}
    @CloseRepeat void close(RepeatContext ctx) {}
    @OnError void onError(RepeatContext ctx, Throwable t) {}
}

@Job(name="Job1")
public class MyJob {
    @Step SomeStep step1;
    @RepeatListener MyRepeatListener listener;
    ...
}
```
Discussion: Retry

- Needs
  - Step-level callback
  - Configurable retryable exceptions
  - Optional retry limit
  - Ability to backout (rollback)

- Step-level Callback – e.g.

```java
@Step public MyStep {
    @BeforeRetry RetryStatus OnRetry(RetryContext ctx) {} 
    @ItemProcessor <S> process(<T> item) {} 
    @AfterRetry RetryStatus AfterRetry(RetryContext ctx) {} 
}
```
Configurable Retryable Exceptions – e.g.

```java
@Step
@Retry(@Exceptions(include={},exclude={}))
public MyStep {
    @BeforeRetry RetryStatus OnRetry(RetryContext ctx) {}
    @ItemProcessor <S> process(<T> item) {}
    @AfterRetry RetryStatus AfterRetry(RetryContext ctx) {}
}
```
Optional retry limit – e.g.

```java
@Step
@Retry(limit=3)
public MyStep {
    @BeforeRetry RetryStatus OnRetry(RetryContext ctx) {}
    @ItemProcessor <S> process(<T> item) {}
    @AfterRetry RetryStatus AfterRetry(RetryContext ctx) {}
}
```
Discussion: Retry

- Ability to Backout (rollback)

```java
@Step
@Retry(limit=3, backout=true)
public MyStep {
    @BeforeRetry RetryStatus OnRetry(RetryContext ctx) {}
    @ItemProcessor <S> process(<T> item) {}
    @AfterRetry RetryStatus AfterRetry(RetryContext ctx) {}
}
```
Do we need a Retry Listener?

```java
@RetryListener
class MyRetryListener {
    @OpenRetry void open(RetryContext ctx) {}
    @CloseRetry void close(RetryContext ctx) {}
    @OnError void onError(RetryContext ctx, Throwable t) {}
}

@Job(name="Job1")
class MyJob {
    @Step SomeStep step1;
    @RetryListener MyRetryListener listener;
    ...
}
```
Needs

- Ability to define skippable conditions (just reads and writes?)
- Step-level callback

```java
@Step
@Skip(@Exceptions(include={}, exclude={}, limit=n))
public MyStep {
    @OnSkipInRead SkipStatus onSkip(Throwable t)
    @OnSkipInWrite SkipStatus onSkip(Throwable t, <T> Item)
}
```
Do we need a Skip Listener?

```java
@SkipListener
public class MySkipListener {
    @OnSkipInRead void skipRead(Throwable t) {}
    @OnSkipInProcess void skipProcess(Throwable t, <T> item) {}
    @OnSkipInWrite void skipWrite(Throwable t, <S> item) {}
}

@Job(name="Job1")
public class MyJob {
    @Step SomeStep step1;
    @SkipListener MySkipListener listener;
    ...
}
```
List for Next Meeting

- Parallel Annotations
- Future
  - Exit codes
  - Step conditions
  - Execution Context
  - Metrics
  - Java EE