JSR 352 Expert Group

Working Session

9 March 2012

Agenda

- (Time-boxed) Review: Listeners Redux
- Discussion: Repeat, Retry, Skip
- List for Next Meeting

Time-boxed Review: Listeners Redux

- **Out of scope for today**: global listeners, callback signatures
- **In scope today**: placement of listener annotation
  - DI: to be or not to be?
  - Spring Harmony?

```
@StepListener
public class MyStepListener {
   @BeforeStep public void before() {...}
   @AfterStep public void after() {...}
}
```

Attach @StepListener to job:

```
@Job(name="Job1")
public class TestJob {
   @StepListener MyStepListener listener;
   ....
}
```

Or JPA style attachment:

```
@Job(name="Job1")
@StepListener({MyListener.class})
public class TestJob {
....
}
```

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

Discussion: Repeat

- 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()

Discussion: Repeat

- @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;
}
```

Discussion: Repeat

- Do we need a Repeat Listener?

```
@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.

```
@Step public MyStep {
    @BeforeRetry RetryStatus OnRetry(RetryContext ctx) {}
    @ItemProcessor <S> process(<T> item) {}
    @AfterRetry RetryStatus AfterRetry(RetryContext ctx) {}
}
```

Discussion: Retry

- Configurable Retryable Exceptions – e.g.

```
@Step
@Retry(@Exceptions(include={},exclude={}))
public MyStep {
    @BeforeRetry RetryStatus OnRetry(RetryContext ctx) {}
    @ItemProcessor <S> process(<T> item) {}
    @AfterRetry RetryStatus AfterRetry(RetryContext ctx) {}
}
```

Discussion: Retry

- Optional retry limit – e.g.

```
@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)

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

Discussion: Retry

- Do we need a Retry Listener?

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

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

Discussion: Skip

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

```
@Step
@Skip(@Exceptions(include={},exclude={},limit=n)
public MyStep {
    @OnSkipInRead SkipStatus onSkip(Throwable t)
    @OnSkipInWrite SkipStatus onSkip(Throwable t, <T> Item)
}
```

Discussion: Skip

- Do we need a Skip Listener?

```
@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

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