Quick look at Batch Applications for the Java Platform

Anissa Lam (梁鈺儀)
Principal Member of Technical Staff

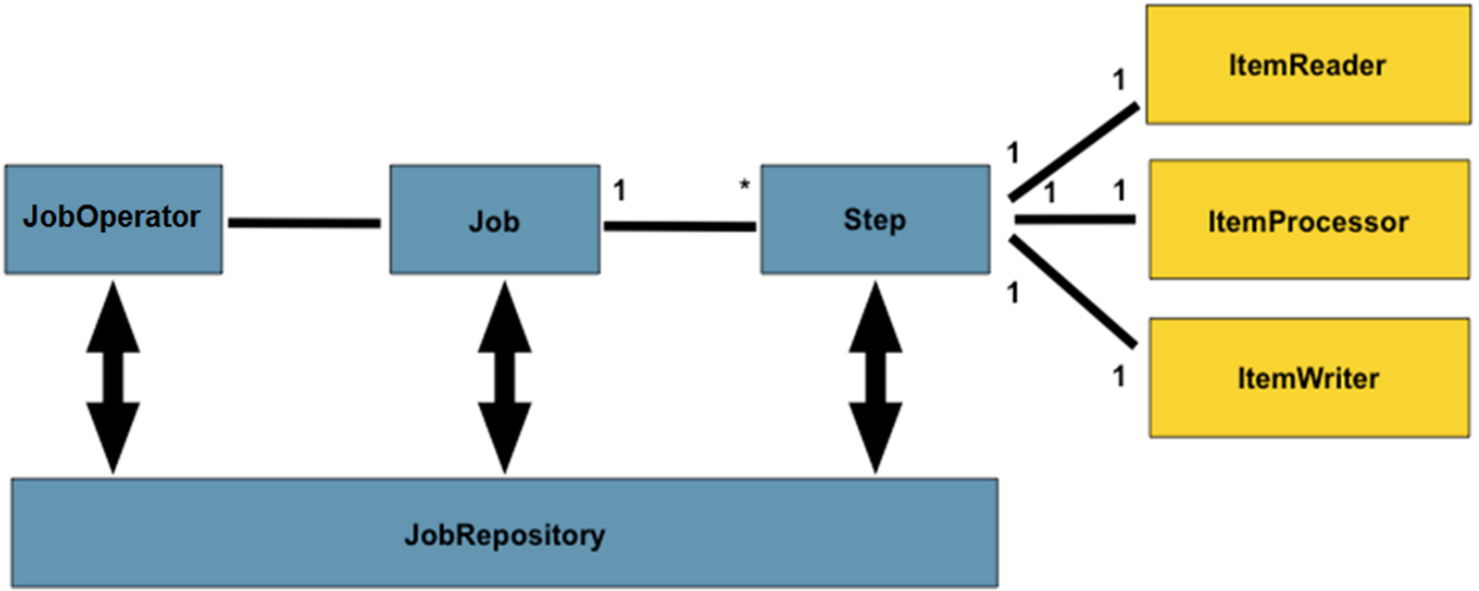
The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Batch Applications for the Java Platform

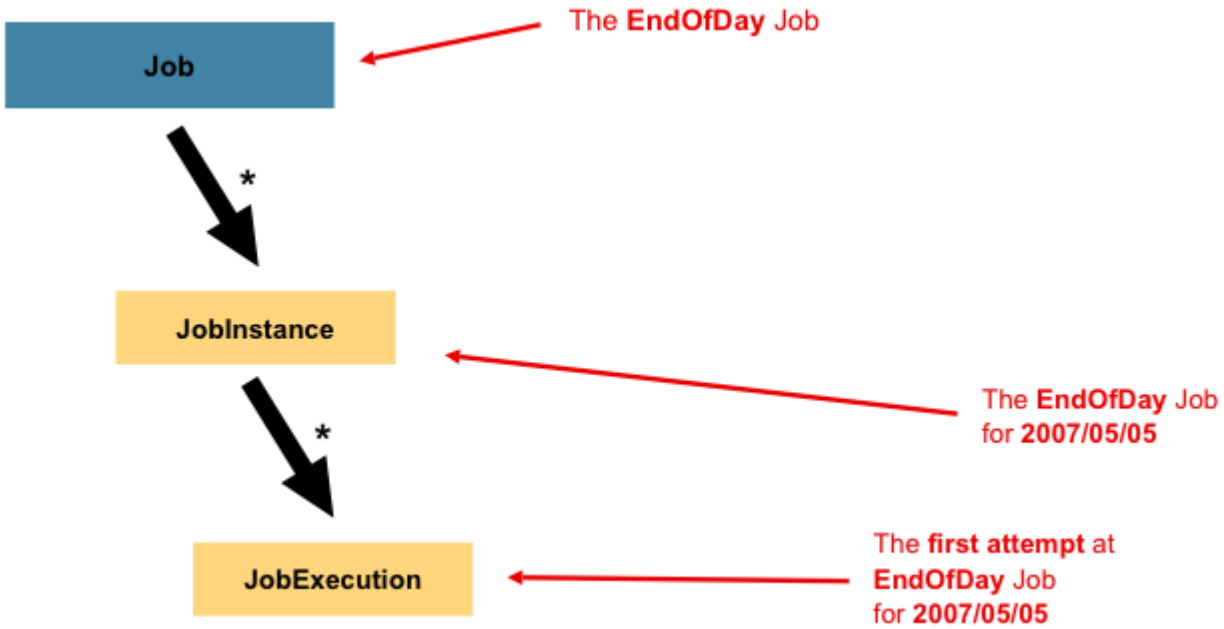
Overview

- New for Java EE 7, can be used in Java SE
- Standardizes batch processing for Java
  - Non-interactive, bulk-oriented, long-running
  - Data or computationally intensive
  - Sequentially or in parallel
  - Ad-hoc, scheduled or on-demand execution
- Led by IBM

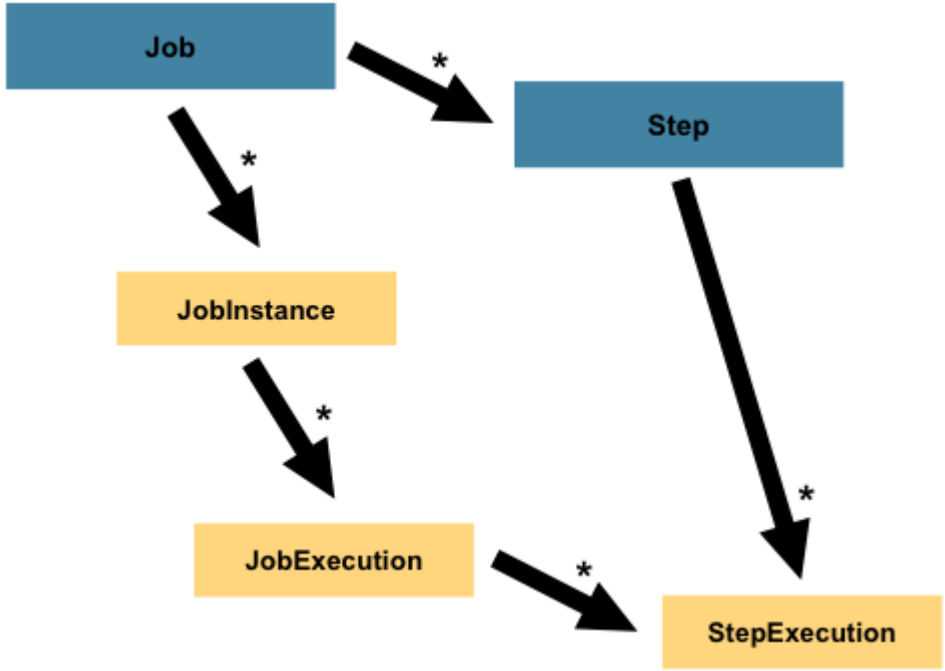
Basic Components of Batch Processing



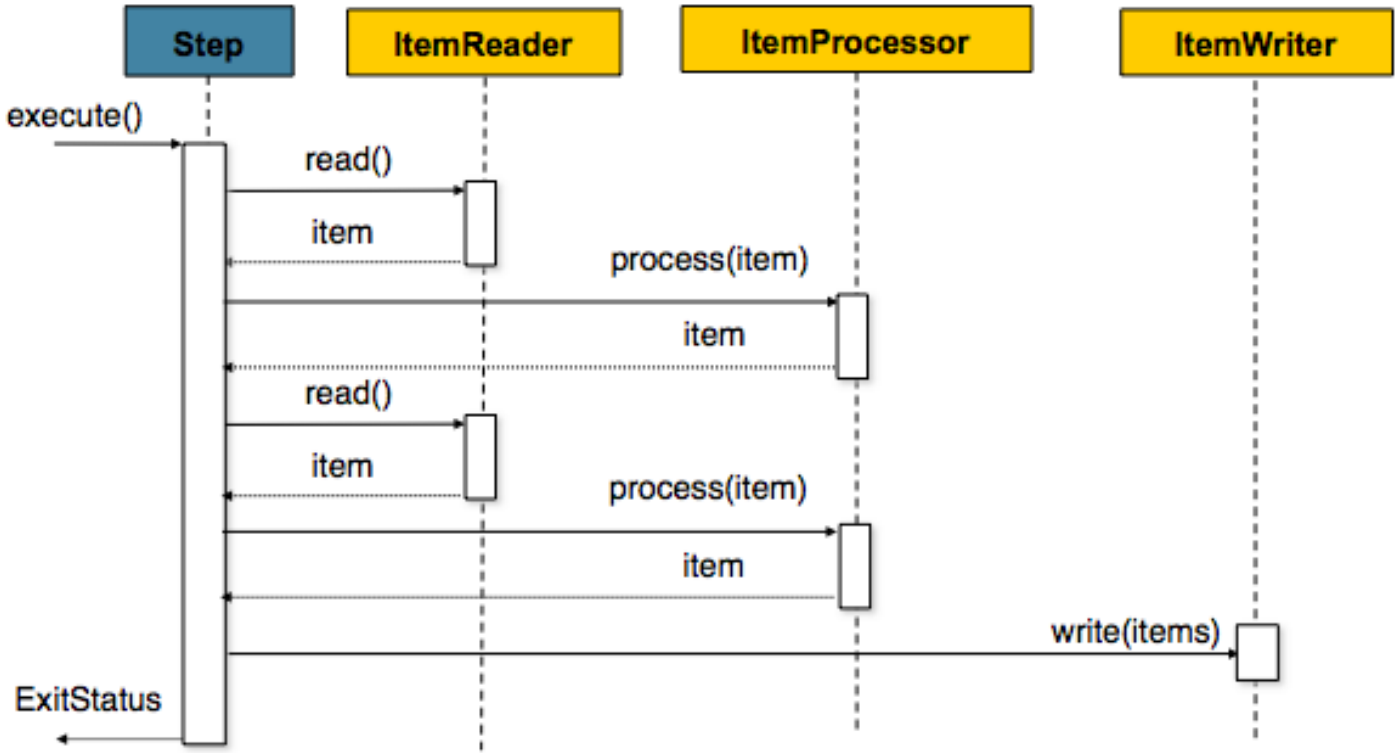
Batch Domain Language



Batch Domain Language



Chunked Processing



Job Specification Language (JSL)

- Specifies the Steps in the order for execution to accomplish the Job.
- Powerful
  - Conditional execution of steps
  - Each step can have different properties, listeners etc.
  - Exception Handling
- Each application can have more than one JSL (job XMLfile)
- Job XML filename must be unique
- Placed in META-INF/batch-jobs directory

Batch Applications for the Java Platform

Step Example using Job Specification Language (JSL)

```
<step id="process">
  <chunk item-count="2"
        <reader    ref="simpleItemReader" />
        <processor ref="simpleItemProcessor"/>
        <writer    ref="simpleItemWriter" />
  <chunk/>
</step>
```

```
@Named
Public class simpleItemReader extends AbstractItemReader {
   @Inject
   private JobContext jobcontext;
   …
}
```

Chunked Processing: Reader, Processor, Writer

```
public interface ItemReader<T> {
  public void open(Externalizable checkpoint);
  public T readItem();
  public Externalizable checkpointInfo();
  public void close();
}

public interface ItemProcessor<T, R> {
  public R processItem(T item);
}

public interface ItemWriter<T> {
  public void open(Externalizable checkpoint);
  public void writeItems(List<T> items);
  public Externalizable checkpointInfo();
  public void close();
}
```

Checkpointing

- For data intensive tasks, long periods of time
  - Checkpoint/restart is a common design requirement
- Basically saves Reader, Writer positions
  - Naturally fits into Chunk oriented steps
  - `reader.checkpointInfo()` and `writer.checkpointInfo()` are called
  - The resulting `Externalizable` data is persisted
  - When the Chunk restarts, the reader and writer are initialized with the respective `Externalizable` data

Chunked Processing: Checkpoint

```
public interface ItemReader<T> {
  public void open(Externalizable checkpoint);
  public T readItem();
  public Externalizable checkpointInfo();
  public void close();
}

public interface ItemProcessor<T, R> {
  public R processItem(T item);
}

public interface ItemWriter<T> {
  public void open(Externalizable checkpoint);
  public void writeItems(List<T> items);
  public Externalizable checkpointInfo();
  public void close();
}
```

Handling Exceptions

```
<job id=...>
...
  <chunk skip-limit="5" retry-limit="5">
    <skippable-exception-classes>
      <include class="java.lang.Exception"/>
      <exclude class="java.io.FileNotFoundException"/>
    </skippable-exception-classes>
    <retryable-exception-classes>
      ...
    </retryable-exception-classes>
    <no-rollback-exception-classes>
      ...
    </no-rollback-exception-classes>
  </chunk>
...
</job>
```

Advanced Topics

- Partitioned Steps
- Advanced Partitioning Scenarios
  - Partition Mapper, Reducer, Collector, Analyzer
- Flow & Split
- Batchlet
- Batch Runtime Specification

Example: Simple Payroll Processing Application

- Simple payroll processing application.
- Demonstrates some of the key features of JSR 352
- SimplePayrollJob
  - Reading input data for payroll processing from CVS file
  - Each line contains employee ID and base salary
  - Calculates tax, bonus, and net salary
  - Finally write out the processed records into Database.

Resources

- Final Specification at : http://java.net/projects/jbatch/
- Questions and comments : public@jbatch.java.net