|  |  |  |
| --- | --- | --- |
| **Parameter** | **Tasklet** | **Chunk** |
| **When to use** | Suppose the job to be run a single granular task then Tasklet processing is used. | Suppose the job to be run is complex and involves executing of tasks involving reads, processing and writes the we use chunk oriented processing |
| **How it works** | No aggregation, just the task gets executed. | It involves reading an input, processing it based on the business logic and then aggregating it till the commit-interval is reached and finally writing out the chunk of data output to a file or database table. |
| **Usage** | Its not used commonly. | Its the most common way of executing a Step. |
| **Use Case** | Usually Used in scenarios invloving a single task like deleting a resource or executing a query . | Usually used in scenarios where multiple aggregated steps need to be run like copying, processing and transferring of data . |
| **Example** | <job id="taskletJob"> <step id="callingStoredProc"> <tasklet ref="callProc"/> </step> </job> [Spring Batch Tasklet Example](http://www.javainuse.com/spring/springbatchtasklet) | <job id="sampleJob" job-repository="jobRepository"> <step id="step1"> <tasklet transaction-manager="transactionManager"> <chunk reader="itemReader" writer="itemWriter" commit-interval="10"/> </tasklet> </step> </job> [Spring Batch Chunk Processing Example](http://www.javainuse.com/spring/bootbatch) |

**ItemReader -**Strategy interface for providing the data. Implementations are expected to be stateful and will be called multiple times for each batch, with each call to read() returning a different value and finally returning null when all input data is exhausted. Implementations need not be thread-safe and clients of a ItemReader need to be aware that this is the case. A richer interface (e.g. with a look ahead or peek) is not feasible because we need to support transactions in an asynchronous batch.

**ItemProcessor -**Interface for item transformation. Given an item as input, this interface provides an extension point which allows for the application of business logic in an item oriented processing scenario. It should be noted that while it's possible to return a different type than the one provided, it's not strictly necessary. Furthermore, returning null indicates that the item should not be continued to be processed.

**ItemStreamWriter -**Basic interface for generic output operations. Class implementing this interface will be responsible for serializing objects as necessary. Generally, it is responsibility of implementing class to decide which technology to use for mapping and how it should be configured. The write method is responsible for making sure that any internal buffers are flushed. If a transaction is active it will also usually be necessary to discard the output on a subsequent rollback. The resource to which the writer is sending data should normally be able to handle this itself.