**SBATCH RETRY NOTE:**

**SCENARIO:**

**Transient Exception:**

Una eccezione transient e’ un ’eccezione legata ad un ostacolo transitorio che potrebbe essere risolto con una nuova esecuzione.

**Esempi:**- Lock su tabella  
- Risorsa di rete non disponibile  
- Time out su una chiamata remota

**Deterministic Exception:**

Eccezione che non puo’ essere rimossa in una nuova esecuzione del job.  
Esempi:  
Dati formalmente incorretti o inconsistenti che impediscono l’elaborazione (da DB o da file)

Tra le Deterministic Exception si possono identificare quelle **Skippable**.  
La singola operazione (il singolo record) viene saltato e si continua col prossimo.

**CHUNK PROCESSING:**

**ItemReader:**

Le eccezioni **retryable** devono essere anche Skippable se vengono lanciate su ItemReader.  
Anche perche’ se la lettura fallisce si puo’ solo pensare di saltarla e riprovare.

**ItemProcessor, ItemWriter:**

Le eccezioni **retryable** comportano la ripetizione della singola operazione (non dello step).  
Per cui se il reader ha prodotto dei dati, la ripetizione del processor o del writer si basera’ sugli stessi dati.  
La transitorieta’ dell’ eccezione deve essere legata alla operazione di **<<process>>** o **<<write>>**(Lock record che si cerca di aggiornare, servizio per update non disponibile)

**TASKLET:**In base a:   
http://stackoverflow.com/questions/27509585/spring-batch-skippable-exception-in-a-taskletstep

Within a Tasklet, the responsibility for exception handling is on the implementation of the Tasklet. The skip logic available in chunk oriented processing is due to the exception handling provided by the ChunkOrientedTasklet. If you want to skip exceptions in your own Tasklet implementation, you need to write the code to do so in within your own implementation.

**Lasciare come esercizio: (impl completa)**1) Ripetere esecuzione del ChunkOriented (puo’ anche essere uno starter) (Tag Sbatch300Ex-S1)  
2) Impl gestione eccezioni. (Retry e Skip) (Tag Sbatch300Ex-S2)

**Implementazione nel metodo tasklet:**

1. RetryEx
2. SkipEx

**TUTORIAL**Classe abstract da cui ereditano i Chunk Item, gestisce conf per eccezioni.  
Commentare/Decommentare retriable e Skippable exceptions in onguna delle classi**.**

**DEMO:  
Flusso:**

1. **ChunckReader:**Riceve numero casuale da NumberService. (100 volte, poi conclude)
2. **ChunckProcessor:**Incrementa di 1 il numbero
3. **ChunckWriter:**Stampa i numeri ricevuti in blocchi di <commit-interval>

**Eccezioni:**Ognuno di loro esegue un metodo con alcune probabilita di generare eccezioni (Retry o Skip)  
- metodo evaluate (usa un generatore di numeri)

**CONFIGURAZIONE DEMO:**

**AbstractChunkProcessing: (contiene configurazioni per gestire probabilita di eccezioni):**Configurazione demo per generare eccezioni (Skippable o Retriable)  
1) RetryFactor (se divisibile per) => Ecc Retriable  
2) SkippableThreshold= (se minore di) =>Ecc Skippable

**TUTORIAL:**

1. Check disabilitati e probabilita alta
2. Abilitare ovunque con possibilita impossibile per R e S
3. Aumentare probabilita R retry=1 (100% fallisce)
4. Abilitare su Writer
5. Aumentare S s=1 (100% fallisce)

**TEST : (vedi tabella sotto)**

1. **Reader:**
   1. Skippable => Quel record viene saltata e prosegue con gli altri.
   2. Retriable =>
      1. R non inclusa in Skippable => fallisce lo Step immediatamente
      2. R inclusa nelle S da config => retry
2. **Processor**:
   1. Retriable: =>
      1. R non inclusa in Skippable => fallisce lo Step immediatamente
      2. Ripete metodo fino a retry-limit, poi applica (se esiste) la skip logic
   2. Skippable:
      1. Salta ogni record in eccezione fino a skiplimit
3. **Writer**:
   1. Skippable:
      1. Eccezione su un item nel write.
      2. Annulla transazione
      3. Riesegue il processor dell’item andato in ex
      4. Se ancora va in ex lo salta e va al successivo
      5. Quando accumula complessivamente un numero di writeSkipCount>skiplimit lo Step fallisce
   2. Retriable:
      1. Ripete da processing su intero chunk fino a retry-limit.
      2. Alla prima eccezione su write l’intero chunk viene riprocessato.
      3. Superato il retry limit
      4. Esegue su quel chunk la skip logic (se presente).

**TEST:**

|  |  |  |
| --- | --- | --- |
| **TipoStep:** | **Skippable Exception triggered** | **Retriable Exception triggered** |
| **Tasket** | Non supportata (da impl in Tasket ) | Non supportata (da impl in Tasket ) |
| **Reader** | Esegue N read fino a skip-limit e poi fallisce lo Step.  Vedi da summary readSkipCount | Deve essere anche skippable per non bloccare lo step.  Viene eseguito il retry entro gli skip limit. |
| **Processor** | Esegue Skip del record che fallisce e va avanti fino a skipLimit.  Verifica **processSkipCount** | Deve essere anche skippable per non bloccare lo step.  Viene eseguito il retry fino al retry-limit, poi lo salta.  Quindi se r-limit=5 e s-limit=10 alla 51 esecuzione fallisce lo Step.  Seguire **processSkipCount** Quando ==skipLimit > Step fallisce |
| **Writer** | Ripete da Processor uno alla volta.  Salta quelli che vanno in eccezione fino allo SkipLimit.  Verifica **writeSkipCount** Quando ==Skip-Limit Step fallisce. | Riprova con questa logica:   1. Ripete Process sulla collection di reader 2. Riprova il Write e torna ad 1) al primo item che fallisce   Per retry-limit volte.  Dopo applica skip-logic (vedi accanto) |

**Code review: Classi**

**LISTENERS:**

Implementare interfaccia  
extendere Classi Support (adapter con metodi vuoti)  
Annotations su classe Item<<Operation>>

**TASKLET:**

**3) RetryTemplate  
 4) SkipTemplate**