**SBATCH RETRY AFU:**

**Obiettivi:**

**SKIP: (sia per CO e Tasklet)**

1. Scenario
2. Da batch config skippable exceptions
3. SkipListener
4. Skip Template
5. Skip Policy Bean
6. Skip con aop e interceptors

**RETRY: (sia per CO e Tasklet)**

1. Transient exceptions (scenario)
2. Retry dichiarativa xml
3. Retry Template
4. Retry con aop e interceptors (meglio per utests)
5. RetryPolicy Bean
6. Retry listener

**ROLLBACK: (NOTE, SPIEGATO SU DEMO SPECIFICA)**

1. Ogni eccezione che si verifica in ItemWriter di default comporta il rollback su transazione
   1. Marcare eccezioni che non comportano rollback
2. TransactionalReader:  
   Evitare Rollback su Reader (esempio read from JMS, se fallisce ogni read successivo preleva sempre lo stesso. Logica skip)

**TEORIA:**

1. Eccezioni transitorie o deterministiche
2. Scenari

**DEMO:**1) Demo generatore sequenziale con ack ed eccezioni

**LABS:**1) QuizLab  
2) Impl stessa logica in Tasklet  
3) Es riepilogativo (nazioni e UnreliableService):  
3.1) Cerca Lab vecchio ed aggiungi allo starter l Unreliable service con un metodo che randomicamente fallisce)

**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:  
      Eccezione su un item nel write.  
      Essendo un’operazione sulla collection, il writer non puo’ determinare quale elemento ha determinato l’eccezione (e che si deve saltare).  
      Per cui esegue un process>write su ogni singolo elemento della collection.  
        
      Nel dettaglio:
      1. Annulla transazione corrente.
      2. Per ogni record nella collection creata dal reader
         1. Esegue il process sul singolo record
         2. Esegue il write del singolo in una singola transazione (requires\_new)
         3. Se va in eccezione, salta lo specifico record.)
      3. Se accumula complessivamente un numero di writeSkipCount>skiplimit lo Step fallisce
   2. Retriable:
      1. Ripete processing su intero chunk
      2. Ripete write dell intero chunk (page)
      3. Ripete i e ii fino a retry-limit.
      4. Alla prima eccezione su write l’intero chunk viene riprocessato.
      5. Superato il retry limit esegue su quel chunk la skip logic (se anche skippable).

**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) |

**QUIZ-LABS:**

**Cosa succede se?**

**Code review: Classi**

**DEMO:**

**SBATCH-300-EXCEPTIONS:**

OBIETTIVO:  
Gestione delle eccezioni in modalita dichiarativa

DESCRIZIONE:  
- Generatore di numeri sequenziali con ack (conferma precedente ricevuto).  
- Sul metodo <dangerousCall> eccezioni transient sulla base di generatore probabilita.  
  
TEST  
- Eccezioni retriable e skippable da testare:  
- valori dei campi skipThreshold e retryFactor. (cambiarli per verifica tests)

ECCEZIONI FLUSSO: dangerousCall:  
1) numeroGenerato < skipThreshold -> lancia SkippableException  
2) numeroGenerato % retryFactor -> Lancia RetryException  
  
**QUESTIONARIO (Da testare e scrivere)**

**Tutorial** (punti di intervento)

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

**LAB:**Implementazione in Tasklet logica descritta.

1. Isolare metodo ripetibile (read, processSingle, writeSingle)
2. Retry fino a limite per RetryEx
3. Per gestire eventuale skippable da Retry catch:
   1. Lanciare skippable all superamento retry limit
4. catch successivo SkippableEx (RetryEx e’ anche Skippable):
   1. Per gestire eventuale skippable da Retry catch
   2. Se non e’ nemmeno skippable allora va in Ex ordinaria e Step fallisce.

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

**LISTENERS:  
  
1) Abilitarli da config  
2)**

**UC:**1) Salvare su file i record andati in eccezione  
2) Salvare performance (time) di ogni chunk (read, process, write (list))  
3) ItemProcessor come filtro (salvare quanti processati, quanti scartati)  
4) Salvare status (condiviso a livello step) che cambia tra steps?

**SBATCH-310-EXCEPTIONS-POLICY:**

OBIETTIVO:  
Gestione delle eccezioni con policy

DESCRIZIONE: