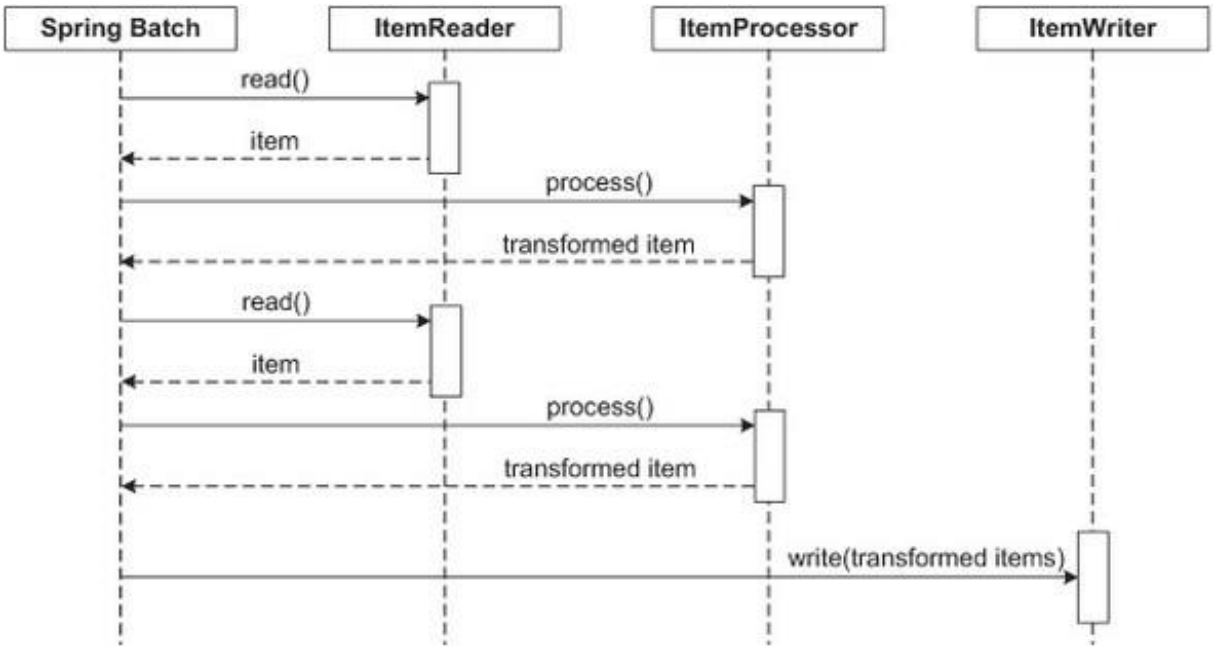
Spring Batch for sequencing DNA

A computational model for DNA damage repair and could explain how CRISPR Cas9 works

Author : Wadï Mami

E-mail : wmami@steg.com.tn / didipostman77@gmail.com

Date : 09/10/2023



Spring Batch reads and process DNA ***sequentially*** until reaching commit-interval value then it writes transformed items (DNA) ***simultaneously***.

Below is a code representation of the same concepts shown above:

```
List items = new Arraylist();
for(int i = 0; i < commitInterval; i++){
    Object item = itemReader.read()
    Object processedItem = itemProcessor.process(item);
    items.add(processedItem);
}
itemWriter.write(items);
```

In my template I used a DNASequence_Processor class that implements itemProcessor and use Karp Rabin Algorithm. (You can use any other algorithm of pattern recognition instead)

Here my repository : https://github.com/didipostman/CrisprCas9/

And here path to DNA_SequenceProcessor.java

src/main/java/com/juxtapose/example/ch02/DNA_SequenceProcessor.java

```
/**
 *
 */
package com.juxtapose.example.ch02;

import org.springframework.batch.item.ItemProcessor;
import com.juxtapose.example.ch02.RabinKarp;

/**

 * @author wadï mami (mailto:didipostman77@gmail.com)
 * 2013-1-6下午09:55:38
 */
public class DNA_SequenceProcessor implements
                ItemProcessor<DNA_Sequence, DNA_Sequence> {


        private String[] snippetsVirusDna = {"TGCTGCT", "AATTCC", "
GGAATAA"};  // the array can be more larger it is just an explanation

        public DNA_Sequence process(DNA_Sequence dnas) throws Exception {

        String CRISPR_ARRAYS ="";

        for (int i=0; i<snippetsVirusDna.length; i++)
           {


        String pat = snippetsVirusDna[i] ;

         String txt = dnas.getDna() ;

         RabinKarp searcher = new RabinKarp.getInstance(pat);

                int[] offset = searcher.search(txt);

                for (int j=0; j< offset.length; j++)

                CRISPR_ARRAYS = CRISPR_ARRAYS +"("
+String.valueOf(offset[j]) + "," + pat.length() + ")";



        }
                dnas.setCrissprArrays(CRISPR_ARRAYS);
     }
```

```
}
```

Spring Batch is the bacteria.

The bacteria capture snippets of DNA from invading viruses and use them to create DNA segments known as CRISPR arrays

```
private String[] snippetsVirusDna = {"TGCTGCT", "AATTCC", " GGAATAA"};  //
the array can be more larger it is just an explanation
```

github.com/didipostman/CrisprCas9/blob/main..

<=> Spring Batch read DNA file or DNA database , The DNA file or the DNA database are Viruses DNA.

SpringBatch read() --->ItemReader and ItemReader return item. and Spring Batch process() ----> ItemProcessor and

return transformed item = DNA segments known as CRISPR arrays Here I used DNA_sequenceProcessor class that

implements ItemProcessor and uses Karp Rabin (you can use other DNA pattern recognition algorithm)

github.com/didipostman/CrisprCas9/blob/main..

The CRISPR arrays allow the bacteria to "remember" the viruses (or closely related ones). If the viruses attack again,

the bacteria produce RNA segments from the CRISPR arrays to target the viruses' DNA example private String

```
private String[] snippetsVirusDna = {"TGCTGCT", "AATTCC", " GGAATAA"};  //
the array can be more larger it is just an explanation
```

github.com/didipostman/CrisprCas9/blob/main..

The bacteria then use Cas9 or a similar enzyme to cut the DNA apart, which disables the virus.

<=> Spring batch write(transformed items) ----> ItemWriter (cut Virus DNA).

A *step* is an object that encapsulates a sequential phase of a job and holds all the necessary information to define and control processing. It delegates all the information to a Job (job.xml) to carry out its task.

```
<job
id="dnaSeq">
                        <step id="dnaSeqStep">
                            <tasklet transaction-manager="transactionManager">
                                <chunk reader="csvItemReader" writer="csvItemWriter"
                                    processor="DNA_SequenceProcessor" commit-interval="2">
                                </chunk>
                            </tasklet>
                        </step>
                </job>
```

Configuring ItemReader

We will now define *ItemReader* for our model which will be used for reading data from CSV file.

```
<bean:bean id="csvItemReader"
                class="org.springframework.batch.item.file.FlatFileItemReader"
                scope="step">
                <bean:property name="resource"
                   value="classpath:ch02/data/DNA.csv"/>
                <bean:property name="lineMapper">
                   <bean:bean
                      class="org.springframework.batch.item.file.mapping.DefaultLineMapper">
                      <bean:property name="lineTokenizer" ref="lineTokenizer"/>
                      <bean:property name="fieldSetMapper">
                         <bean:bean
class="org.springframework.batch.item.file.mapping.BeanWrapperFieldSetMapper">
                            <bean:property name="prototypeBeanName" value="DNA_Sequence">
```

```
                    </bean:property>
                </bean:bean>
            </bean:property>
        </bean:bean>
    </bean:property>
</bean:bean>
<!-- lineTokenizer -->
<bean:bean id="lineTokenizer"
    class="org.springframework.batch.item.file.transform.DelimitedLineTokenizer">
    <bean:property name="delimiter" value=","/>
    <bean:property name="names">
        <bean:list>
            <bean:value>dna</bean:value>
                    <bean:value>crissprArrays</bean:value>

        </bean:list>
    </bean:property>
</bean:bean>
```

Configuring ItemProcessor

```
<bean:bean id="DNA_SequenceProcessor" scope="step"
        class="com.juxtapose.example.ch02.DNA_SequenceProcessor">
    </bean:bean>
```

As you can see I used a DNASequence_Processor class that implements itemProcessor and used Karp Rabin Algorithm.

ItemWriter

Once the data is processed, the data needs to be stored in a file as per our requirement.

```
<bean:bean id="csvItemWriter"
                class="org.springframework.batch.item.file.FlatFileItemWriter"
                scope="step">
                <bean:property name="resource" value="file:target/ch02/outputFile.csv"/>
                <bean:property name="lineAggregator">
                  <bean:bean
                    class="org.springframework.batch.item.file.transform.DelimitedLineAggregator">
                    <bean:property name="delimiter" value="|"></bean:property>
                    <bean:property name="fieldExtractor">
                      <bean:bean
                        class="org.springframework.batch.item.file.transform.BeanWrapperFieldExtractor">
                        <bean:property name="names"
                          value="dna, seqDNA_Arrays">
                        </bean:property>
                      </bean:bean>
                    </bean:property>
                  </bean:bean>
                </bean:property>
              </bean:bean>
```

Conclusion

This article just scratched the surface of Spring Batch in general. The example used in this article is not production-ready code. You can define job configuration depending on your project requirements. Here The Github repository for the project https://github.com/didipostman/CrisprCas9/

Processing large volume of data has always been a major problem due to the increasing volume of the data. Batch processing can be applied in many use cases. Among them why not Pattern Matching for DNA Sequencing Data. Spring Batch provides functions for processing large volumes of data in batch jobs. In our case reading DNA file or database table and seeking for patterns I mean all the locations of the specified pattern inside a DNA sequence.

Spring batch to process huge data : Spring Batch is a lightweight, comprehensive batch framework designed to enable the development of robust batch applications vital for the daily operations of enterprise systems.

DNA is a sequence of letters such as A, C, G, T. Searching for specific sequences is often difficult due to measurement errors, mutations or evolutionary alterations. Thus, similarity of two sequences using Levenshtein Distance is more useful than exact matches.

So instead of Karp Rabin we will use Levenshtein Distance or Jaro_Winkler_Similarity by using

Package org.apache.commons.text.similarity commons.apache.org/proper/commons-text/apid..

So

Spring Batch + Levenshtein Distance or Jaro_Winkler Similarity = How Crispr cas9 Works due to (https://www.tudelft.nl/en/2018/tu-delft/mathematics-explains-why-crispr-cas9-sometimes-cuts-the-wrong-dna)