# BL-07 ファイルアクセス機能

## ■ 概要

### ◆ 機能概要

- TERASOLUNA Batch Framework for Java ver 3.x で使用するファイルアクセス機能は、TERASOLUNA Batch Framework for Java ver 2.x で使用していたファイルアクセス機能と同一のものを利用して、ファイルアクセスを行う。
- 本項目では、TERASOLUNA Batch Framework for Java ver 3.x でファイルアクセス機能を使用する場合の TERASOLUNA Batch Framework for Java ver 2.x との違いのみを説明するものとし、ファイルアクセス機能の詳細な説明は別資料の「BC-01 ファイルアクセス機能」の機能説明書を参照すること。

### ◆ 概念図の違いについて

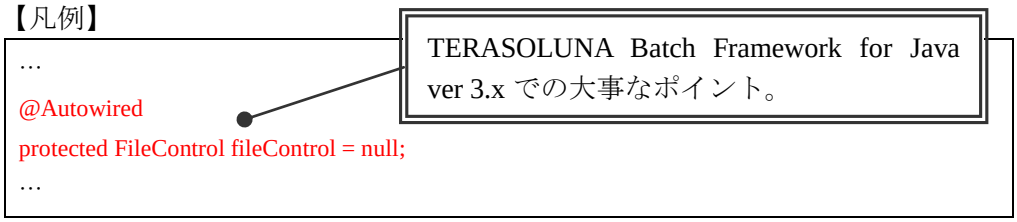
- 別資料の「BC-01 ファイルアクセス機能」では概念図「ファイル入力処理の概念図(Collector から利用される場合)」において Collector から利用される場合の概念図を掲載しているが、TERASOLUNA Batch Framework for Java ver 3.x では、このような使い方はしないため、読み飛ばすこと。
- TERASOLUNA Batch Framework for Java ver 3.x でファイルアクセス機能を利用するときはビジネスロジックから利用するので、「ファイル入力処理の概念図(ビジネスロジックから利用される場合)」の概念図を参考にすること。

## ◆ コーディングポイント

- 本説明書でのコーディングポイントは、別資料の「BC-01 ファイルアクセス機能」のコーディングポイントと異なる以下の項目についてのみ説明を行う。
  - ・ ファイル入力チェックについて
  - ・ 例外処理
  - ・ ファイル入力の実装例
  - ・ ファイル出力の実装例

  その他の項目については、別資料の「BC-01 ファイルアクセス機能」を参照すること。

- 次ページからのコーディングポイントの中で、TERASOLUNA Batch Framework for Java ver 3.x で本機能を使用する際に大事なポイントについては、二重線の吹き出しを使用して強調している。




- ファイル入力チェックについて

  TERASOLUNA Batch Framework for Java ver 3.x ではファイル、DB から取得したデータの入力チェックを行うための『AL-043 入力チェック機能』を提供している。
  使用方法についての詳細は『AL-043 入力チェック機能』の機能説明書を参照すること。

- 例外処理について

  TERASOLUNA Batch Framework for Java ver 3.x では例外処理のための「BL-04 例外ハンドリング機能」を提供している。
  使用方法についての詳細は「BL-04 例外ハンドリング機能」の機能説明書を参照すること。

- ファイル入力の実装例
  - ファイル入力処理の実装

(1) ファイル行オブジェクトを実装する。

(2) ファイル入力処理を行うクラスの Bean 定義を行う。
【ジョブ Bean 定義ファイルの設定例】

```
...
    <!-- コンポーネントスキャン設定 -->
    <context:component-scan base-package="jp.terasoluna.batch.sample.sample00001" />
...
```

Batch 3.x では@Component アノテーションを使用して、Bean を自動的に検出し、DI コンテナで管理する。

上記の例ではパッケージ[jp.terasoluna.batch.sample.sample00001]内に存在する@Component アノテーションが付与されたクラスを自動的に検出する。

(3) ファイル入力処理を行うクラスでは、FileQueryDAO の execute()メソッドでファイル入力用イテレータを取得する。ファイル入力用イテレータ取得時に、ファイルオープンが行われる。
ファイル入力用イテレータの next メソッドで、ファイル行オブジェクトを取得する。

➢ ビジネスロジック実装例

```
@Component
public class B001001BLogic implements BLogic {

    @Autowired
    @Qualifier("csvFileQueryDAO")
    protected FileQueryDAO fileQueryDAO = null;

    public int execute(BLogicParam arg0) {

        ...
        // ファイル入力用イテレータの取得
        FileLineIterator<SampleFileLineObject> fileLineIterator
                                = fileQueryDAO.execute(basePath +
                                        "/some_file_path/uriage.csv",FileColumnSample.class);

        try {

      // ヘッダ部の読み込み
          List<String> headerData = fileLineIterator.getHeader();
          ... // 読み込んだヘッダ部に対する処理

          while(fileLineIterator.hasNext()){
              // データ部の読み込み
            SampleFileLineObject sampleFileLine
                                        = fileLineIterator.next();

          ... // 読み込んだ行に対する処理
          }
          // トレイラ部の読み込み
          List<String> trailerData = fileLineIterator.getTrailer();
          ... // 読み込んだトレイラ部に対する処理

        } finally {
            // ファイルのクローズ
            fileLineIterator.closeFile();
        }
... (以下略)
```

Batch 3.x では@Autowired アノテーションを使用して、ビジネスロジックへの DAO の DI を行う。(FileDAO の場合は FileQueryDAO インターフェースの実装クラスが多数存在するので、@Qualifier を使用して Bean 名を指定する必要がある。)

ファイルパスとファイル行オブジェクトクラスを引数にして、ファイル入力用イテレータを取得する

アノテーション FileFormt の headerLineCount で設定した行数分のヘッダ部を取得する。

ファイル形式に関わらず、next()メソッドを使用する

アノテーション FileFormt の trailerLineCount で設定した行数分のトレイラ部を取得する

closeFile()メソッドでファイルを閉じること

➢ ファイルの入力順序
トレイラ部の入力は、データ部の入力が全て終わった後に行う必要がある点に留意すること。

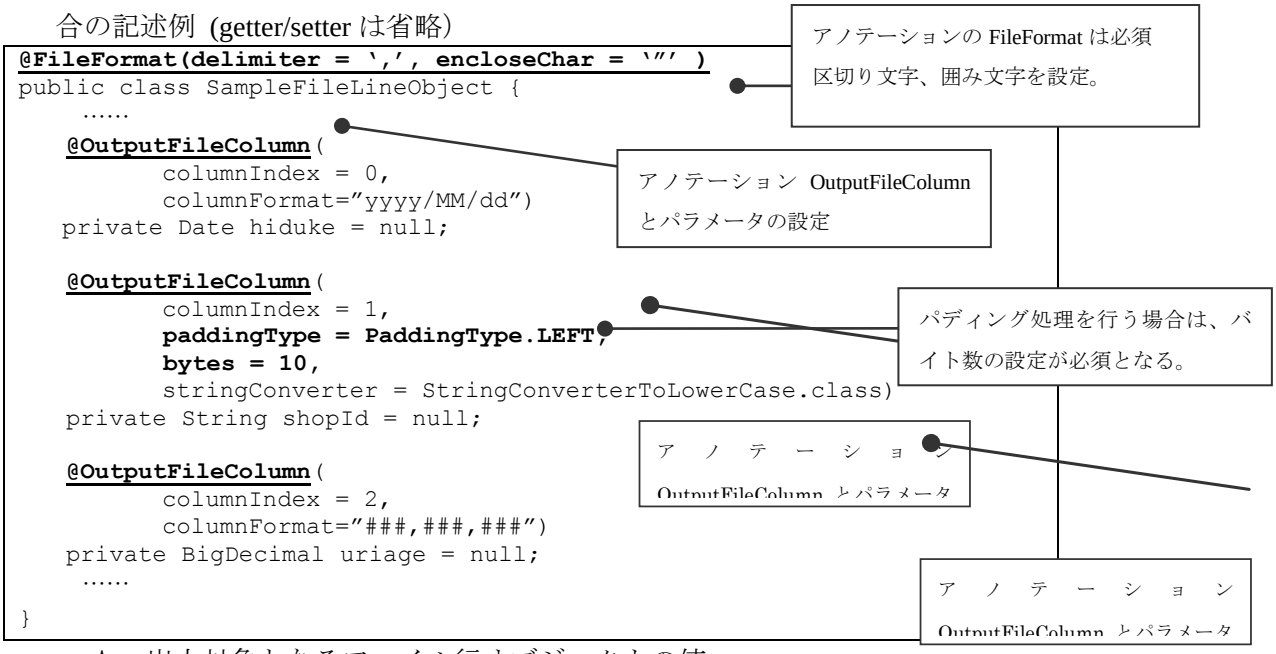
➢ スキップ処理
ファイル入力機能では入力を開始する行を指定する事ができる。
スキップ処理は中断していたファイルの読み込みを再開するような場合に使用する事が出来る。

➢ ビジネスロジック実装例

```
……
// スキップ処理
fileLineIterator.skip(1000);
……
```

fileLineIterator のカレント行から 1000 行分のデータ行を読み飛ばす処理を行う

- ファイル出力の実装例
  - ファイルの設定として、囲み文字と区切り文字を設定し、データの一部をデフォルトのパディング文字でパディング処理したデータをファイルに出力する場合の記述例 (getter/setter は省略)




    - 出力対象となるファイル行オブジェクトの値

```
hiduke = Sat Jul 01 00:00:00 JST 2006
shopId = SHOP01
uriage = 1000000
```

    - 上記のファイル行オブジェクトを出力すると以下の値となる。

```
"2006/07/01","    shop01","1,000,000"
```

➢ データの一部を個別のパディング文字でパディング処理したデータをファイルに出力する場合の記述例 (getter/setter は省略)

```
@FileFormat(delimiter = ',', encloseChar = '"' )
public class SampleFileLineObject {
     ……
    @OutputFileColumn(
           columnIndex = 0,
           columnFormat="yyyy/MM/dd")
    private Date hiduke = null;

    @OutputFileColumn(
           columnIndex = 1,
           paddingType = PaddingType.RIGHT,
           paddingChar = '0',
           bytes = 10,
           stringConverter = StringConverterToLowerCase.class)
    private String shopId = null;

    @OutputFileColumn(
           columnIndex = 2,
           columnFormat="###,###,###")
    private BigDecimal uriage = null;
     ……
}
```

右側のパディング処理を行い、パディング文字として'0'を設定する。

アノテーション OutputFileColumn とパラメータ

✧ 出力対象となるファイル行オブジェクトの値

```
 hiduke = Sat Jul 01 00:00:00 JST 2006
shopId = SHOP01
uriage = 1000000
```

✧ 上記のファイル行オブジェクトを出力すると以下の値となる。

```
"2006/07/01","shop010000","1,000,000"
```

- ファイル出力処理の実装（１メソッドでファイルをオープン・クローズする場合）
-

(1) ファイル行オブジェクトを実装する。

(2) ファイル出力処理を行うクラスの Bean 定義を行う。

【ジョブ Bean 定義ファイルの設定例】

```
...
    <!-- コンポーネントスキャン設定 -->
    <context:component-scan base-package="jp.terasoluna.batch.sample.Sample00001" />
...
```

Batch 3.x では@Component アノテーションを使用して、Bean を自動的に検出し、DI コンテナで管理する。

上記の例ではパッケージ[jp.terasoluna.batch.sample.Sample00001]内に存在する@Component アノテーションが付与されたクラスを自動的に検出する。

※ファイル入力処理の Bean 定義の際に上記の記述を行っていた場合は、改めて記述する必要はない。

(3) ファイル出力処理を行うクラスでは、FileUpdateDAO の execute メソッドでファイル出力用行ライタを取得する。ファイル出力用行ライタの取得時に、ファイルがオープンされる。

➢ ビジネスロジック実装例

```
@Component
public class B001001BLogic implements BLogic {

    @Autowired
    @Qualifier("csvFileUpdateDAO")
    protected FileUpdateDAO csvFileUpdateDAO = null;

    public int execute(BLogicParam arg0) {

        // ファイル出力用行ライタの取得
        FileLineWriter< SampleFileLineObject > fileLineWriter
                = fileUpdateDAO.execute(basePath + "/some_file_path/uriage.csv",
                                        SampleFileLineObject.class);

        try {
            // ヘッダ部の出力
            fileLineWriter.printHeaderLine(headerString);
            ...
        while ( ... ) {
            ...
            // データ部の出力（1行）
            fileLineWriter.printDataLine(sampleFileLineObject);
            ...
        }
        ...
        // トレイラ部の出力
        fileLineWriter.printTrailerLine(trailerString);
        ...
    } finally {
        // ファイルのクローズ
        fileLineWriter.closeFile();
    }
...(以下略)
```

ファイル入力の際と同様に @Autowired アノテーションを使用して、ビジネスロジックへの DAO の DI を行う。

ファイル名とパラメータクラスを引数に、ファイル出力用行ライタを取得する。

ヘッダ部を出力する、String 型の変数を引数とする。

ファイル形式に関わらず、printDataLine メソッドで出力する。
出力される項目には、項目定義用のアノテーションを付加しておく。

トレイラ部を出力する。String 型の変数を引数とする。

出力が終了したら、ファイルをクローズする。

### ◆ 拡張ポイント

- なし

## ■ 関連機能

- 『AL-043 入力チェック機能』
- 『BL-04 例外ハンドリング機能』

## ■ 使用例

- 機能網羅サンプル(terasoluna-batch-functionsample)
- チュートリアル(terasoluna-batch-tutorial)

## ■ 備考

- なし