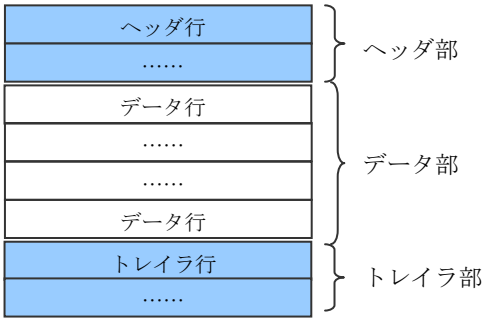
BC-01 ファイルアクセス機能

■ 概要

◆ 機能概要

- CSV 形式、固定長形式、可変長形式ファイルの入出力機能を提供する。
  - ファイル入力機能は順次読込のみ提供する。

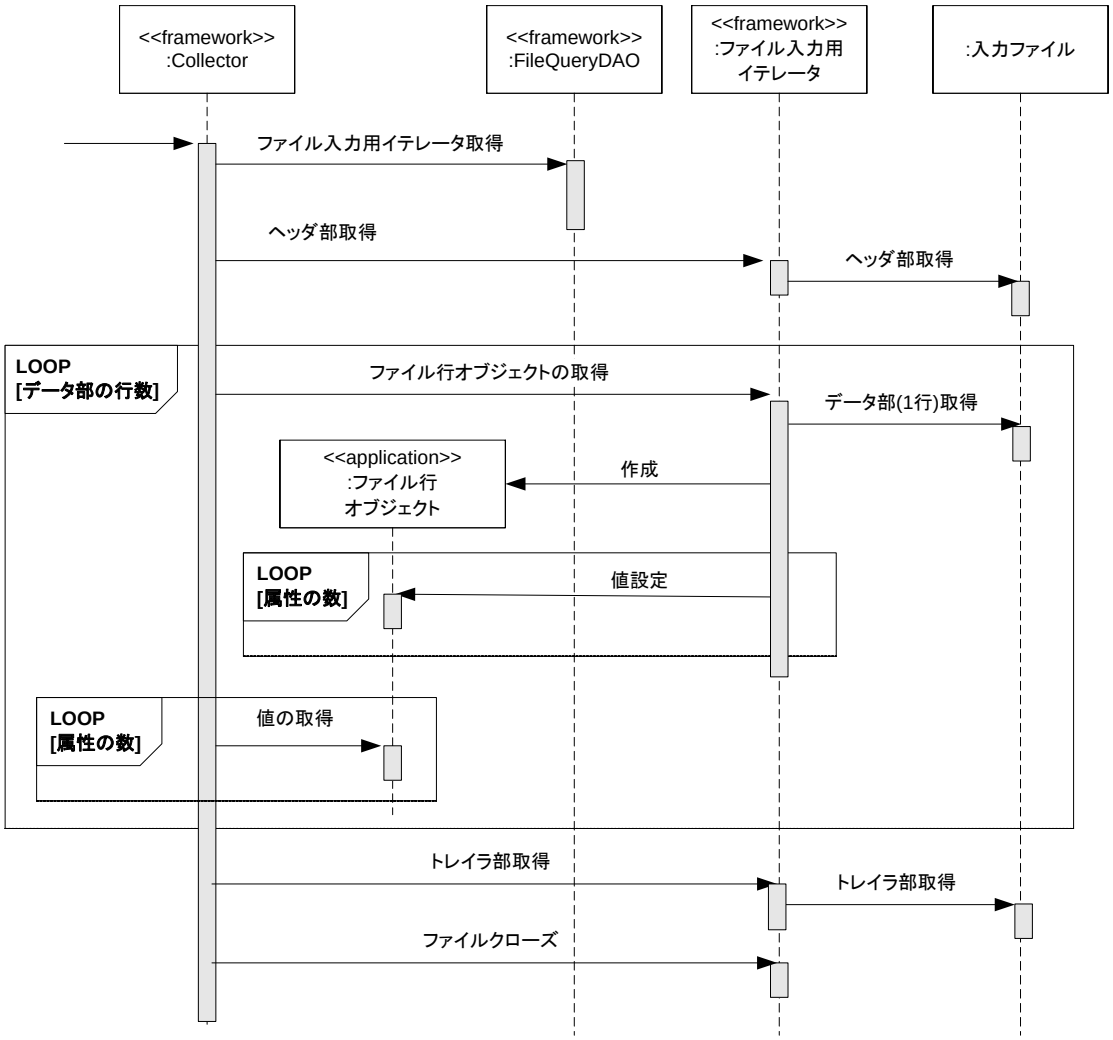
- ファイルアクセス機能で対象とするファイル構造は下図のとおりである。
  - ヘッダ部、トレイラ部の無いファイルについては、ヘッダ部、トレイラ部を 0 行とする。
  - データ部のデータ構造はファイル行オブジェクト（POJO）にアノテーションを使用して定義する。



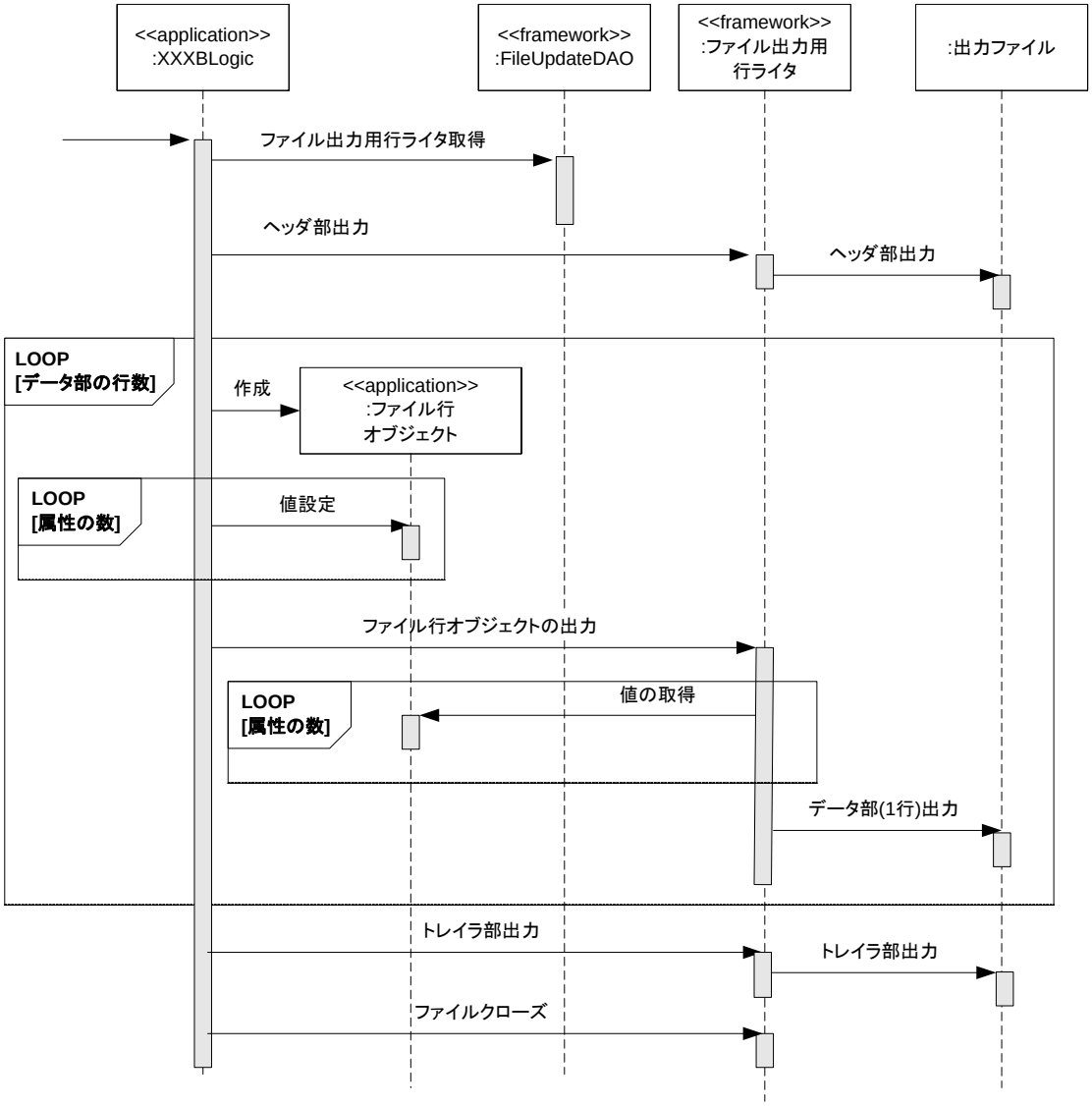
- 入出力データに対するフォーマット処理を行う。
  - ファイル行オブジェクト（POJO）の定義により、項目に対するパディング(Padding)、トリム（Trim）、文字変換（StringConverter）等が指定できる。

◆ 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）



- ファイル出力処理の概念図（ビジネスロジックから利用される場合）



◆ 解説

- ファイルアクセス機能で取り扱うファイル
  TERASOLUNA-Batch のファイルアクセス機能は CSV 形式、固定長形式、可変長形式、文字列データのファイル入出力機能を提供する。ファイル内の各行の項目数及び項目の並び順は同一である必要がある。

  - CSV 形式
    CSV 形式とは、データを「,(カンマ)」で区切ったものである。データを区切る際に使用しているカンマを特に"区切り文字”と呼ぶ。CSV 形式は可変長ファイルの区切り文字を「,(カンマ)」に固定したものになる。

  - 固定長形式
    固定長形式とは、対象データの 1 行の各項目の長さ（バイト数）が全ての行で同じであるもの。対象データを区切る方法は、各項目に割り当てられているバイト数をもとに行う。

  - 可変長形式
    可変長形式とは、対象データの 1 行の各項目の長さ（バイト数）が可変である（異なる）もの。対象データは"区切り文字” を使って区切る。

  - 文字列データ
    文字列データとは、区切り文字やバイト数でデータを区切る必要が無いもの。１行分のデータを String 型として扱う。

- ファイル行オブジェクト
  ファイル入出力を使用する場合には、ファイルのデータ部の 1 行分のデータに対応するように Java Bean のクラスを作成する。作成する Java Bean のクラスを、本フレームワークではファイル行オブジェクトと呼ぶ。
  ファイル行オブジェクトのクラスには、ファイル全体に関わる定義情報（改行文字等）を Java アノテーションにより設定する。
  ファイル行オブジェクトのクラスが持つ属性には、ファイルの個々の項目の定義情報（バイト長等）を Java アノテーションにより設定する。

- ファイル全体に関わる定義情報（FileFormat アノテーション）
  ファイル全体に関わる定義情報は、ファイル行オブジェクトのクラスに対してアノテーション FileFormat により設定する。FileFormat アノテーションは入力ファイル、および出力ファイルのどちらの場合にも同じアノテーションを設定する。
  - アノテーション FileFormat 設定項目

項番	論理項目名 物理項目名	説明	デフォルト値	CSV		固定長		可変長		文字列データ	
				入	出	入	出	入	出	入	出
1	行区切り文字 lineFeedChar	行区切り文字(改行文字)を設定する。	システムデフォルト/なし(固定長)	○	○	○	○	○	○	○	○
2	区切り文字 delimiter	「,（カンマ）」等の区切り文字を設定する。	「,(カンマ)」	×	×	×	×	○	○		
3	囲み文字 encloseChar	「“（ダブルクォーテーション）」等のカラムの囲み文字を設定する。	なし	○	○	×	×	○	○		
4	ファイルエンコーディング fileEncoding	入出力を行うファイルのエンコーディングを設定する。	システムデフォルト	○	○	○	○	○	○	○	○
5	ヘッダ行数 headerLineCount	入力ファイルのヘッダ部に相当する行数を設定する。	0	○		○		○		○	
6	トレイラ行数 trailerLineCount	入力ファイルのトレイラ部に相当する行数を設定する。	0	○		○		○		○	
7	ファイル上書きフラグ overWriteFlg	出力ファイルと同じ名前のファイルが存在する場合に上書きするかどうかを設定する。 [true/false]（上書きする/上書きしない）	FALSE		○		○		○		○

※○の項目は必要に応じて設定可。×の項目は設定できないことを表している（×の項目を設定した場合、実行時にエラーとなる）。無印は設定を無視することを表している。
※「行区切り文字」の"システムデフォルト"とは、
System.getProperty(”line.separator”);で取得できる実行環境に依存した値である。
固定長形式の場合のみ、デフォルト値はなし。
※固定長形式で行区切り文字を無し（改行無し）とした場合、ヘッダ行数とトレイラ行数を設定することはできない。
※「ファイルエンコーディング」の"システムデフォルト"とは、
System.getProperty(”file.encoding”);で取得できる実行環境に依存した値である。

※「行区切り文字」、「区切り文字」でタブ、改行文字を使用する場合、Java 言語仕様で定められているエスケープシーケンス（¥t、¥r 等）で記述すること。
※可変長の「区切り文字」として「￥u0000」を設定することはできない。
※「区切り文字」と「囲み文字」は同一の値を設定することができない。
※「行区切り文字」と「区切り文字」は同一の値を設定することができない。

- ファイル項目の定義情報
  （InputFileColumn アノテーション、OutputFileColumn アノテーション）
  ファイル項目の定義情報は、ファイル行オブジェクトのクラスが持つ属性に対してアノテーション InputFileColumn、OutputFileColumn により設定する。アノテーション InputFileColumn、OutputFileColumn は入力ファイル、および出力ファイルのどちらの場合にも同じアノテーションを設定する。
  ひとつのファイル行オブジェクトが入力ファイル、および出力ファイルの両方で使用される場合には、ひとつの属性に対して二つのアノテーション
  （InputFileColumn、OutputFileColum）を設定する。

  - InputFileColumn,OutputFileColumn の設定項目

項番	論理項目名 物理項目名	説明	デフォルト値	CSV		固定長		可変長	
				入	出	入	出	入	出
1	カラムインデックス columnIndex	データ部の 1 行のカラムの内、何番目のデータをファイル行オブジェクトの属性に格納するのかを設定する。インデックスは「0(ゼロ)」から始まる整数。	なし	◎	◎	◎	◎	◎	◎
2	フォーマット columnFormat	BigDecimal 型、Date 型に対するフォーマットを設定する。	なし	○	○	○	○	○	○
3	バイト長 bytes	各カラムに対するバイト長を設定する。	なし	○	○	×	×	○	○
4	パディング種別 paddingType	パディングの種別を設定する。列挙型 PaddingType から値を選択する。[RIGHT/LEFT/NONE] (右寄せ/左寄せ/パディングなし)	NONE		○		○		○
5	パディング文字 paddingChar	パディングする文字を設定する(半角文字のみ設定可能)。	なし		○		○		○
6	トリム種別 trimType	トリムの種別を設定する。列挙型 TrimType から値を選択する。[RIGHT/LEFT/BOTH /NONE] (右寄せ/左寄せ/両側/トリムなし)	NONE	○	○	○	○	○	○
7	トリム文字 trimChar	トリムする文字を設定する(半角文字のみ設定可能)。	なし	○	○	○	○	○	○
8	文字変換種別 stringConverter	String 型のカラムについて、大文字変換等を設定する。StringConverter インタフェースの実装クラスを指定する。 StringConverterToUpperCase.class (大文字に変換) / StringConverterToLowerCase.class(小文字に変換) / NullStringConverter.class(変換しない)	NullStringConverter.class	○	○	○	○	○	○
9	囲み文字 columnEncloseChar	カラム単位で「“(ダブルクォーテーション)」等のカラムの囲み文字を設定する。	なし	○	○	×	×	○	○

※ ◎の項目はアノテーションを設定する際の必須項目(必須項目を設定しなかった場合、コンパイルエラーとなる)。○の項目は必要に応じて設定可。×の項目は設定できないことを表している（×の項目を設定した場合、実行時にエラーとなる)。無印

は設定を行っても有効にならないことを表している。

※ バイト長とは、入力時はファイルから取得する時点の長さであり、各種変換処理後の長さとは異なる。出力時はファイルへ出力する時点の長さであり、各種変換処理後の長さである。

※ パディング種別、トリム種別を指定したときには、それぞれパディング文字、トリム文字を必ず設定すること。

※ パディング種別で NONE 以外を指定したときはバイト長を必ず設定すること。ここでのバイト長は、パディング処理を行った後のバイト長を設定すること。

※ 入力処理時にパディングを設定した場合、取得データがバイト長で設定した長さ以外の場合はバイト長チェックでエラーが発生し、バイト長で設定した長さと一致する場合はパディングすべきデータ数が 0 となるため、パディング処理を行っても取得データと同じになる。つまり、入力時にパディングの設定を行っても有効にならないことに留意すること。

※ 変換処理の順番は、入力時と出力時で異なることに留意すること。

- 入力時はバイト数チェック、トリム処理、パディング処理、文字変換処理である。
- 出力時はトリム処理、パディング処理、文字変換処理、バイト数チェックである。

※ FileFormat の encloseChar と InputFileColumn または OutputFileColumn の columnEncloseChar の両方が設定されている場合、columnEncloseChar の設定が優先される

※ CSV 形式、可変長形式の場合、ファイル行オブジェクトに InputFileColumn、OutputFileColum が一つも設定されていない場合、実行時にエラーとなる。

- データ項目定義と異なるデータを入出力した場合の例外処理
  対象データの入力の際、アノテーションの記述と異なるデータがあった場合、フレームワークは例外を発生させる。例外が発生する例としては、日付型のフォーマットを設定しているところに、数値型のデータを格納しようとした場合などが挙げられる。

- ファイル行オブジェクトの属性について
  - ファイル行オブジェクトで使用できる属性の型は、java.lang.String、int 、java.math.BigDecimal、java.util.Date の 4 種類とする。
  - 各属性には値を操作するために可視性が public である setter/getter を用意すること。

- フォーマットについての補足
  数値型、日付型の入出力は、フォーマットとして入力した文字列に沿ってデータの入出力を行う。詳細については、Java Platform Standard Edition (Java SE) API 仕様 を参照のこと。
  "java.text.DecimalFormat"、"java.text.SimpleDateFormat"

- ファイルの 1 行あたりのカラム数についての補足
  ファイル入力の際、アノテーションを設定したカラム数とファイルのカラム数が異なる場合、フレームワークは例外を発生させる。また、固定長ファイル入力の際、アノテーションの bytes で設定したカラムのバイト数の合計と読み取った 1 行のバイト数が異なる場合、フレームワークは例外を発生させる。

- 囲み文字についての補足
  囲み文字が出力項目に含まれている場合、同じ囲み文字を追加してエスケープ編集を行う。また、ファイル入力の際、カラムに囲み文字と同一の文字が含まれておりエスケープ編集されていないカラムの場合、フレームワークは例外を発生させる。

- ファイル上書きフラグについての補足
  FileUpdateDAO を DI したビジネスロジック分割ジョブで起動した場合、ファイル上書きフラグを True に設定するとデータが破損する可能性があることに留意すること。また、分割ジョブにてファイル上書きファイルフラグを false にした場合、複数スレッドから一つのファイルにアクセスするため、データの出力順番がランダムになることに留意すること。

- ファイル入力用 DAO
  フレームワークではファイル入力用 DAO インタフェース、およびファイル入力用イテレータインタフェースを規定し、ファイル形式に対応したそれぞれのデフォルト実装を提供する。
  ファイル入力用 DAO の execute()メソッドを実行し、ファイル入力用イテレータを取得する。ファイルの各行は、ファイル入力用イテレータの next()メソッドで取得する。
  - ファイル入力用インタフェース

項番	インタフェース名	概要
1	jp.terasoluna.fw.file.dao.FileQueryDAO	ファイル入力用 DAO インタフェース
2	jp.terasoluna.fw.file.dao.FileLineIterator	ファイル入力用イテレータインタフェース

  - ファイル入力用 DAO 実装クラス

項番	クラス名	概要
1	jp.terasoluna.fw.file.dao.standard.CSVFileQueryDAO	CSV 形式のファイル入力を行う場合に利用する
2	jp.terasoluna.fw.file.dao.standard.FixedFileQueryDAO	固定長形式のファイル入力を行う場合に利用する
3	jp.terasoluna.fw.file.dao.standard.VariableFileQueryDAO	可変長形式のファイル入力を行う場合に利用する
4	jp.terasoluna.fw.file.dao.standard.PlainFileQueryDAO	文字列データをファイルから入力する場合に利用する

    - ファイル入力用 DAO は、ファイル入力用イテレータを生成する。

  - ファイル入力用イテレータ実装クラス

項番	クラス名	概要
1	jp.terasoluna.fw.file.dao.standard.CSVFileLineIterator	CSV 形式のファイル入力を行う場合に利用する
2	jp.terasoluna.fw.file.dao.standard.FixedFileLineIterator	固定長形式のファイル入力を行う場合に利用する
3	jp.terasoluna.fw.file.dao.standard.VariableFileLineIterator	可変長形式のファイル入力を行う場合に利用する
4	jp.terasoluna.fw.file.dao.standard.PlainFileLineIterator	文字列データをファイルから入力する場合に利用する

    - ファイル入力でのデータ部の入力は、データ部の 1 行分のデータを入出力オブジェクトに格納し、呼び出し元に返却する処理を提供する。
    - ヘッダ部、トレイラ部からの入力用メソッドを提供する。
    - 囲み・区切り文字として設定された文字が入力データの文字列にあると正しく動作しない。

- ファイル出力用 DAO
  フレームワークではファイル出力用 DAO、およびファイル出力用行ライタのインタフェースを規定し、ファイル形式に対応したそれぞれのデフォルト実装を提供する。
  ファイル出力用 DAO の execute()メソッドを実行し、ファイル出力用行ライタを取得する。ファイルの各行は、ファイル出力用行ライタの printDataLine()メソッドで出力する。

  ➢ ファイル出力用インタフェース

項番	インタフェース名	概要
1	jp.terasoluna.fw.file.dao.FileUpdateDAO	ファイル出力用 DAO インタフェース
2	jp.terasoluna.fw.file.dao.FileLineWriter	ファイル出力用行ライタインタフェース

  ✧ ファイル出力用 DAO 実装クラス

項番	クラス名	概要
1	jp.terasoluna.fw.file.dao.standard.CSVFileUpdateDAO	CSV 形式のファイル出力を行う場合に利用する
2	jp.terasoluna.fw.file.dao.standard.FixedFileUpdateDAO	固定長形式のファイル出力を行う場合に利用する
3	jp.terasoluna.fw.file.dao.standard.VariableFileUpdateDAO	可変長形式のファイル出力を行う場合に利用する
4	jp.terasoluna.fw.file.dao.standard.PlainFileUpdateDAO	文字列データをファイルへ出力する場合に利用する

  ・ ファイル出力用 DAO は、ファイル出力用イテレータを生成する。

  ✧ ファイル出力用行ライタ実装クラス

項番	クラス名	概要
1	jp.terasoluna.fw.file.dao.standard.CSVFileLineWriter	CSV 形式のファイル出力を行う場合に利用する
2	jp.terasoluna.fw.file.dao.standard.FixedFileLineWriter	固定長形式のファイル出力を行う場合に利用する
3	jp.terasoluna.fw.file.dao.standard.VariableFileLineWriter	可変長形式のファイル出力を行う場合に利用する
4	jp.terasoluna.fw.file.dao.standard.PlainFileLineWriter	文字列データをファイルへ出力する場合に利用する

  ・ ファイル出力でのデータ部の出力は、ファイル行オブジェクトに格納された一行分のデータをファイルに書込む処理を提供する。
  ・ ヘッダ部、トレイラ部への出力メソッドを提供する
  ・ ファイル生成時、フォルダ名は存在するフォルダを設定する必要がある。 存在しないとファイルは生成されない。

- ファイル入力チェックについて
  Collector での対象データ取得時に、入力ファイルに対して入力チェックを行うことができる。
  入力チェックについての詳細は、『BD-02 対象データ取得機能』を参照のこと。

- 例外処理
  ファイルアクセス時に例外が発生した場合、ファイルアクセス用の DAO からスローする例外クラスに、エラーが発生したファイルの情報を格納する。例外が発生した処理に対する後処理(処理過程で生成されたファイルの削除処理など) は例外ハンドラで実装すること。例外ハンドラの詳細については『BH-01 例外ハンドリング機能』を参照のこと。
  ファイルアクセス時の例外クラスには、以下の２つのクラスがある。

項番	例外クラス名	概要
1	jp.terasoluna.fw.file.dao. FileException	ファイル全体に関わるエラーに対応する例外クラス。 以下の情報を保持する。 ・ファイル名
2	jp.terasoluna.fw.file.dao. FileLineException （FileException のサブクラス）	ファイルの行に関わるエラーに対応する例外クラス。 以下の情報を保持する。 ・ファイル名 ・エラーが発生した箇所の行番号 ・エラーが発生したカラムのカラムインデックス（0 から開始） ・エラーが発生したカラムのカラム名（ファイル行オブジェクトのプロパティ名）

  例外クラスが保持する情報を用いて、ログ出力などを行うことができる。

■ 使用方法

◆ コーディングポイント

- ファイル行オブジェクトの実装例
  - CSV 形式のデータをファイル行オブジェクトに格納する場合の記述例 (getter/setter は省略)

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

    @InputFileColumn(
           columnIndex = 1,
           stringConverter = StringConverterToUpperCase.class)
    private String shopId = null;

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

アノテーションの FileFormat は必須

アノテーション InputFileColumn とパラメータの設定

    - ✧ 上記のファイル行オブジェクトに下記の CSV 形式のデータを格納すると、各属性の値は以下の通りとなる。

```
"2006/07/01","shop01","1,000,000"  ←  CSV形式のデータ
```

    - ✧ ファイル行オブジェクトに設定される値

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

文字列データ(PlainFileQueryDAO, PlainFileUpdateDAO)を利用する場合は、@FileFormat のみを記述したファイル行オブジェクトを使用すること。

➢ ファイル全体に関わる定義情報を設定する場合の記述例 (getter/setter は省略)

```
@FileFormat(encloseChar = '"', lineFeedChar="¥r¥n",
  headerLineCount = 1, trailerLineCount= 1)
public class SampleFileLineObject {
     ……
    @InputFileColumn(
           columnIndex = 0,
           columnFormat="yyyy/MM/dd")
   private Date hiduke = null;

    @InputFileColumn(
           columnIndex = 1,
           stringConverter = StringConverterToUpperCase.class)
    private String shopId = null;

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

FileFormat で、ヘッダ部行数とトレイラ部行数を指定する。

✧ 上記のファイル行オブジェクトに下記のデータを格納すると、各属性の値は以下の通りとなる。

```
支店名：千葉支店                          ← ヘッダ部
"2006/07/01","shop01","1,000,000"        ← データ部
合計金額：1,000,000                       ← トレイラ部
```

ヘッダ部とトレイラ部を含んだファイル

✧ ファイル行オブジェクトに設定される値

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

※ ヘッダ部とトレイラ部はファイル行オブジェクトに格納されない。

- ファイル項目でトリム種別を設定し、デフォルトのトリム文字を使用した定義情報を設定する場合の記述例 (getter/setter は省略)

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

    @InputFileColumn(
           columnIndex = 1,
           trimType = TrimType.RIGHT,
           stringConverter = StringConverterToUpperCase.class)
    private String shopId = null;

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

右側の空白をトリム(削除)するように設定する。

  - 上記のファイル行オブジェクトに下記の CSV 形式のデータを格納すると、各属性の値は以下の通りとなる。

```
"2006/07/01","shop01  ","1,000,000"        ←  データ部
```

  - ファイル行オブジェクトに設定される値

```
hiduke = Sat Jul 01 00:00:00 JST 2006
shopId = SHOP01      ←右側にあった空白文字を削除している。
uriage = 1000000
```

➢ ファイル項目でトリム種別を設定し、個別のトリム文字を使用した定義情報を設定する場合の記述例 (getter/setter は省略)

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

    @InputFileColumn(
           columnIndex = 1,
           trimType = TrimType.LEFT,
           trimChar = '0',
           stringConverter = StringConverterToUpperCase.class)
    private String shopId = null;

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

左側の'0'の文字ををトリム(削除)するように設定する。

✧ 上記のファイル行オブジェクトに下記の CSV 形式のデータを格納すると、各属性の値は以下の通りとなる。

```
"2006/07/01","000shop01","1,000,000"        ← データ部
```

✧ ファイル行オブジェクトに設定される値

```
hiduke = Sat Jul 01 00:00:00 JST 2006
shopId = SHOP01      ←対象文字列の左側にある'0'が削除される。
uriage = 1000000
```

● ジョブ Bean 定義ファイルの設定例

```
<bean id="CSVFile01"
   class="jp.terasoluna.XXX.・・・">
      <!-- 入力ファイルの設定 -->
      <property name="fileDao">
         <ref bean="csvFileQueryDAO" />
      </property>
```

ファイル入出力用 DAO を使うクラス

ファイル入出力用 DAO 実装クラス。
参照する Bean は「FileAccessBean.xml」を参照のこと

- ファイル全体に囲み文字を設定し、更にファイル項目で個別の囲み文字を使用した定義情報を設定する場合の記述例 (getter/setter は省略)

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

    @InputFileColumn(
           columnIndex = 1,
           columnEncloseChar = '¥'',
           stringConverter = StringConverterToUpperCase.class)
    private String shopId = null;

    @InputFileColumn(
           columnIndex = 2,
           columnEncloseChar = '|',
           columnFormat="###,###,###")
    private BigDecimal uriage = null;
     ……
}
```

ファイル全体に囲み文字を設定する。

カラム単位で囲み文字を設定する。
(全体の設定より優先される)

  - 上記のファイル行オブジェクトに下記の CSV 形式のデータを格納すると、各属性の値は以下の通りとなる。

```
"2006/07/01",'shop01',|1,000,000|    ← データ部
```

  - ファイル行オブジェクトに設定される値

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

- 固定長形式で行区切り無しのデータをファイル行オブジェクトに格納する場合の記述例(getter/setter は省略)

```
@FileFormat(lineFeedChar = "")
public class SampleFileLineObject {
     ……
    @InputFileColumn(
           columnIndex = 0,
           bytes = 10,
           columnFormat="yyyy/MM/dd")
    private Date hiduke = null;

    @InputFileColumn(
           columnIndex = 1,
           bytes = 6,
           stringConverter = StringConverterToUpperCase.class)
    private String shopId = null;

    @InputFileColumn(
           columnIndex = 2,
           bytes = 9,
           columnFormat="###,###,###")
    private BigDecimal uriage = null;
     ……
}
```

行区切り文字を無しに設定する。

各カラムにバイト数を設定する。

✧ 上記のファイル行オブジェクトに下記の固定長形式のデータを格納すると、各属性の値は以下の通りとなる。

```
2006/07/01shop011,000,000  ←  データ部
```

✧ ファイル行オブジェクトに設定される値

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

- 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プロパティに、FileQueryDAO 実装クラスを設定する。

    【ジョブ Bean 定義ファイルの設定例】

```
<bean id="blogic"
  class="jp.terasoluna.batch.sample.SampleLogic">
  <property name="fileQueryDAO" ref="csvFileQueryDao" />
</bean>
```

参照する Bean は「FileAccessBean.xml」を参照のこと

    (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。

    【実装例】

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

```
...
   // 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",FileColumnSa
                                        mple.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();
   }
...
```

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できる。これは主に『BE-04 リスタート機能』で中断したジョブを再開する際、リスタートポイントからファイルの読み込みを再開するために利用する。

【ビジネスロジックの実装例】

```
// 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

- ファイル入力処理の実装（ファイル入力を行うクラスでファイルをオープンしたまま、読み込みを行う場合）
  ファイル入力処理を行うクラスに、（File 更新用 Dao ではなく）ファイル入力用行イテレータを直接設定する。ファイル入力用行イテレータのコンストラクタで、ファイルのパス、ファイル行オブジェクトのクラス等を指定する。

【ジョブ Bean 定義ファイルの設定例】

```
<bean id="blogic" class="testBlogic">
  <property name="iterator">
    <bean class="jp.terasoluna.fw.file.dao.standard.
      CSVFileLineIterator"
      destroy-method="closeFile">
        <constructor-arg index="0" value="some_file_path/uriage.csv" />
        <constructor-arg index="1"
                value="jp.terasoluna.batch.sample.FileColumnSample" />
            <constructor-arg index="2" ref="columnParserMap" />
    </bean>
  </property>
</bean>
```

bean のコンストラクタにファイル名(1 番目の引数。文字型) 、パラメータクラス(2 番目の引数、クラス型)、テキスト変換処理(3 番目の引数。FileAccessBean.xml で定義されている Bean "columnParserMap"を固定で指定) を設定する。

- ファイル入力処理の実装についての補足
  ファイル入力処理を行う場合は、FileQueryDAO を利用してもファイル入力用行イテレータを直接設定する利用する方法のどちらかを利用することで実装出来る。FieQueryDAO を利用する場合は DAO が呼ばれるたびにファイルのオープン/クローズ処理が行われる。そのため、ファイル入力用行イテレータを直接設定した場合と比べて処理が遅くなる。また、ファイル入力用行イテレータを直接設定した場合に、ビジネスロジックで入力処理を行い後処理でファイルの移動などを行うと後処理ではまだファイルストリームが存在するため、エラーが発生することに留意すること。

- 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)

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

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

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

アノテーションの FileFormat は必須
区切り文字、囲み文字を設定。

アノテーション OutputFileColumn
とパラメータの設定

パディング処理を行う場合は、バイト数の設定が必須となる。

アノテーション OutputFileColumn とパラメータ

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","    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プロパティに、FileUpdateDAO 実装クラスを設定する。

【ジョブ Bean 定義ファイルの設定例】

```
<bean id="blogic"
  bean class="jp.terasoluna.batch.sample.SampleLogic">
   <property name="fileUpdateDAO" ref="csvFileUpdateDao" />
</bean>
```

(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。

【実装例】

```
...
    // 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();
}
...
```

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。

- ファイル出力処理の実装（ファイル出力を行うクラスでファイルをオープンしたまま、追記する場合）

【ジョブ Bean 定義ファイルの設定例】

```
<bean id="blogic" class="testBlogic">
  <property name="writer">
    <bean class="jp.terasoluna.fw.file.dao.standard.
      CSVFileLineWriter"
      destroy-method="closeFile">
        <constructor-arg index="0" value="some_file_path/uriage.csv" />
        <constructor-arg index="1"
             value="jp.terasoluna.batch.sample.SampleFileLineObject" />
             <constructor-arg index="2" ref="columnFormatterMap" />
    </bean>
  </property>
</bean>
```

bean のコンストラクタにファイル名(1 番目の引数。文字型) 、パラメータクラス(2 番目の引数、クラス型)、テキスト変換処理(3 番目の引数。FileAccessBean.xml で定義されている Bean "columnFormatterMap"を固定で指定) を設定する。

- ファイル出力処理の実装についての補足
  ファイル出力処理を行う場合は、FileUpdateDAO を利用してもファイル出力用行ライタを直接設定する利用する方法のどちらかを利用することで実装出来る。FieUpdateDAO を利用する場合は DAO が呼ばれるたびにファイルのオープン/クローズ処理が行われる。そのため、ファイル出力用行ライタを直接設定した場合と比べて処理が遅くなる。また、ファイル出力用行ライタを直接設定した場合に、ビジネスロジックで出力処理を行い後処理でファイルの移動などを行うと後処理ではまだファイルストリームが存在するため、エラーが発生することに留意すること。

- ファイルの出力順序についての補足
  ヘッダ部の出力は、データ部の出力の前に行う必要がある点に留意すること。同様にトレイラ部の出力は、データ部の出力がすべて終わった後に行う必要がある点に留意すること。但し、FileUpdateDAO を前処理、主処理、後処理で利用する場合は順序性が担保出来ているか判断することが出来ないため「トレイラ(前処理)⇒データ(主処理)⇒ヘッダ(後処理)」という出力が不可能となる。前処理、主処理、後処理の各々で FileUpdateDAO を利用する場合はヘッダ部、データ部、トレイラ部の出力処理順序の注意が必要である。

◆ 拡張ポイント

- なし

■ 関連機能

- 『BD-02 対象データ取得機能』
- 『BE-04 リスタート機能』
- 『BH-01 例外ハンドリング機能』

■ 使用例

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

■ 備考

- encloseChar で囲い文字を指定しており、囲み文字の前後にデータが存在する場合のフィルアクセス機能の挙動について。
  - ➢ 以下の１行を読み込む場合を想定する。

```
"001",A"toyosu"B
```

  - ➢ ファイル行オブジェクトの実装例(getter/setter は省略)

```
@FileFormat(encloseChar = '"')
public class SampleFileLineObject {
    ……
    @InputFileColumn(columnIndex = 0)
     private String id = null;

    @InputFileColumn(columnIndex = 1)
    private String name = null;
}
```

  - ➢ このように、囲い文字の前後にデータが存在するような場合、デフォルトではファイル読み込み時に例外「FileLineException」をスローする。

  - ➢ 次ページでこの設定を変更するための手順を以下の２つの例に分けて紹介する。
    1. 囲い文字の中のデータのみ取得し、例外をスローさせない方法。
    2. カラム内のデータ全てを取得し、例外をスローさせない方法。

1. 囲い文字の中のデータを自動的に取得し、例外をスローさせない方法。
(以下のようにファイル行オブジェクトを取得し、処理を継続するパターン)
✧ 取得できるファイル行オブジェクトの内容
id = 001
name = toyosu

➢ この場合はファイルアクセス用 DAO の Bean 定義において、プロパティ「throwExceptionAtFormatViolation」を false に設定する。

➢ 以下に、Bean 定義ファイルの設定例を掲載する。

```
（略）
    <!-- CSV ファイルアクセス用（入力）DAO -->
    <bean id="csvFileQueryDAO"
        class="jp.terasoluna.fw.file.dao.standard.CSVFileQueryDAO"
        parent="fileQueryDAO">
        <property name="throwExceptionAtFormatViolation" value="false"/>
    </bean>

    <!-- 可変長ファイルアクセス用（入力）DAO -->
    <bean id="variableFileQueryDAO"
        class="jp.terasoluna.fw.file.dao.standard.VariableFileQueryDAO"
        parent="fileQueryDAO">
        <property name="throwExceptionAtFormatViolation" value="false"/>
    </bean>
（略）
```

✧

2. カラム内のデータ全てを取得し、例外をスローさせない方法。
(以下のようにファイル行オブジェクトを取得し、処理を継続するパターン)
id = 001
name = A”toyosu”B

➢ この場合はファイル行オブジェクトを修正し、カラム毎に個別に囲い文字を設定するする事で実現できる。

➢ 以下に、ファイル行オブジェクトの実装例を掲載する。

```
@FileFormat()
public class SampleFileLineObject {
    ……
    @InputFileColumn(columnIndex = 0 , columnEncloseChar = '"')
     private String id = null;

    @InputFileColumn(columnIndex = 1 ,
        columnEncloseChar = Character.MIN_VALUE)
    private String name = null;
（略）
}
```

対象のカラム全体を取得できるように、
個別に囲い文字を設定する。