AL-042　コントロールブレイク機能

■ 概要

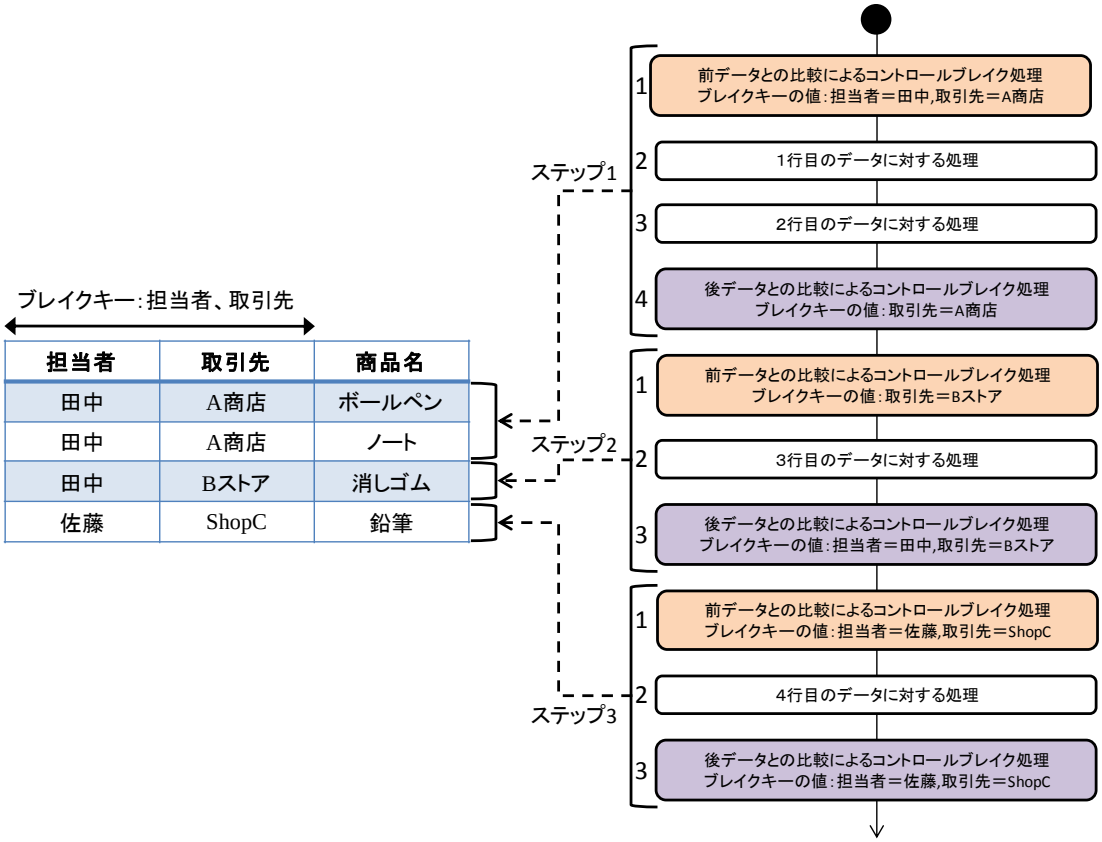
◆ 機能概要

- コントロールブレイク処理とは、ある項目をキーとして、キーが変わるまで集計したり見出しを追加したりする処理である。キーブレイク処理とも呼ばれる。
- 本機能では、コントロールブレイク処理を行うためのユーティリティを提供する。
- コントロールブレイクの判断には「前データとの比較」「後データとの比較」の 2 種類を用意している。
  - 取得したデータの処理前にコントロールブレイク処理を行いたい場合は「前データとの比較」を使用する。(例：見出しの作成など)
  - 取得したデータの処理後にコントロールブレイク処理を行いたい場合は「後データとの比較」を使用する。(例：データの集計など)

◆ 注意点

- 本機能を使用する際には、『AL-041 入力データ取得機能』の使用が前提条件となる。

◆ 概念図



◆ 解説

ステップ		処理内容
1	1	前データとの比較によるコントロールブレイク処理が実行される。
	2	1 行目に対する処理が実行される。
	3	2 行目に対する処理が実行される。
	4	後データとの比較によるコントロールブレイク処理が実行される
2	1	前データとの比較によるコントロールブレイク処理が実行される。
	2	3 行目に対する処理が実行される。
	3	後データとの比較によるコントロールブレイク処理が実行される
3	1	前データとの比較によるコントロールブレイク処理が実行される。
	2	4 行目に対する処理が実行される。
	3	後データとの比較によるコントロールブレイク処理が実行される

- 設定したブレイクキーの値が切り替わったタイミングでコントロールブレイク処理が実行される。
- ビジネスロジック中でブレイクキーを取得することにより、ビジネスロジック中で切り替わった値を取得することが可能である。

■ 使用例

◆ コーディングポイント

【コーディングポイントの構成】

- ファイル行オブジェクトクラスの実装例
- コントロールブレイク処理の実装例(単一のコントロールブレイクキー)
  - ビジネスロジックの処理イメージ
  - ビジネスロジックの実装例
- コントロールブレイク処理の実装例(複数のコントロールブレイクキー)
  - ビジネスロジックの処理イメージ
  - ビジネスロジックの実装例
- コントロールブレイク機能を使用する場合の注意点
- コントロールブレイク機能が持つメソッドの一覧、及び解説
  - ControlBreakChecker クラスのメソッド一覧
  - ControlBreakChecker クラスのメソッドで使用する引数一覧

- ファイル行オブジェクトクラスの実装例

以降のビジネスロジックの実装例で使用するファイル行オブジェクトの実装例を掲載する。

ファイル行オブジェクトクラスは以下の 2 つの役割を持っている。

- DB から取得した 1 行分のデータをマッピングさせるオブジェクト
- 1 行分のデータをファイルに出力するためのファイル行オブジェクト

```
@FileFormat(encloseChar = '"', overWriteFlg = true)
public class CBData {

    @OutputFileColumn(columnIndex = 0)
    private String tantousya = null;

    @OutputFileColumn(columnIndex = 1)
    private String shopName = null;

    @OutputFileColumn(columnIndex = 2)
    private String itemName = null;

      (setter/getter は特筆する点がないので省略する)
```

囲み文字、上書きフラグの設定

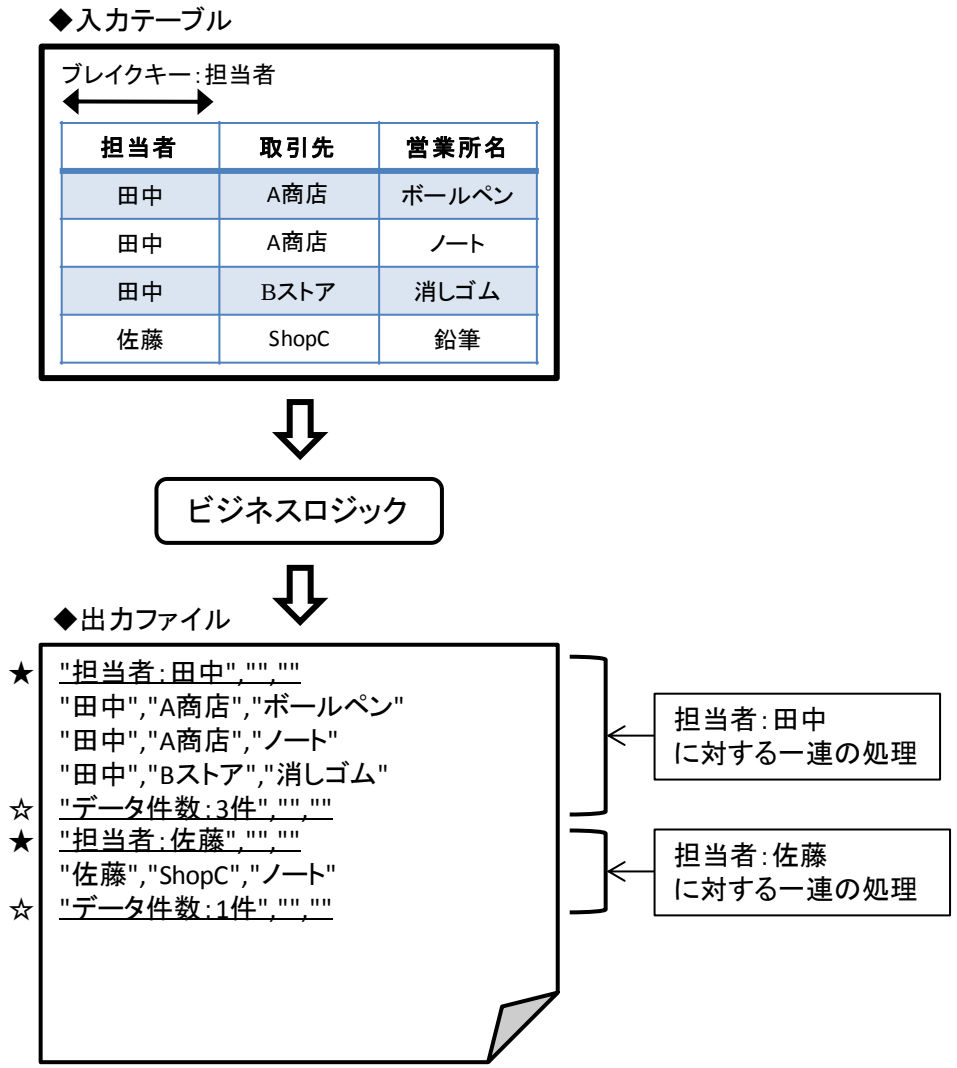
フィールドとカラム番号の紐付け

その他、ファイル出力に関するアノテーションについての詳細は『BL-07 ファイルアクセス機能』の機能説明書を参照すること。

- コントロールブレイク処理の実装例(単一のコントロールブレイクキー)

概念図の表に対して、コントロールブレイクキーを「担当者」のみに絞った場合のビジネスロジックの実装例を以下に掲載する

➢ ビジネスロジックの処理イメージ



図中でアンダーラインを引いている行は、コントロールブレイク処理によって出力された行である。

[実装イメージ中の記号の意味]

★…前データとの比較によるコントロールブレイク処理による出力

☆…後データとの比較によるコントロールブレイク処理による出力

次ページで上記イメージ図のビジネスロジックの実装例を解説と一緒に掲載する。

➢ ビジネスロジックの実装例

```
@Component
public class B101BLogic extends AbstractTransactionBLogic {

    @Inject
    B101Dao b101Dao;

    @Inject
    @Named("csvFileUpdateDAO")
    FileUpdateDAO csvFileUpdateDAO;

    @Override
    public int doMain(BLogicParam param) {
        // コレクタの生成
        Collector<CBData> collector = new DaoCollector<CBData>(
                this.b101Dao, "collectData01", null);

        // ファイル出力用行ライタの取得
        FileLineWriter<CBData> fileLineWriter = csvFileUpdateDAO.execute(
                "outputFile/SampleOutput01.csv", CBData.class);

        // データ件数カウント用
        private int dataCount = 0;
        // コントロールブレイクキーの設定
        String cbKey = "tantousya";

        try {
            while (collector.hasNext()) {
                // データの取得
                CBData inputData = collector.next();
                // コントロールブレイク判断（前データとの比較）
                if (ControlBreakChecker.isPreBreak(collector, cbKey)) {
                    // コントロールブレイクキー取得
                    Map<String, Object> keyMap = ControlBreakChecker
                            .getPreBreakKey(collector, cbKey);
                    // コントロールブレイク処理
                    fileLineWriter.printDataLine(new CBData("担当者："
                            + keyMap.get(cbKey).toString(), "",    ""));
                }
```

(次ページに続く)

String 型でコントロール
ブレイクキーの設定を行う。

前データとの比較は
取得データに対する
処理の手前で行う。

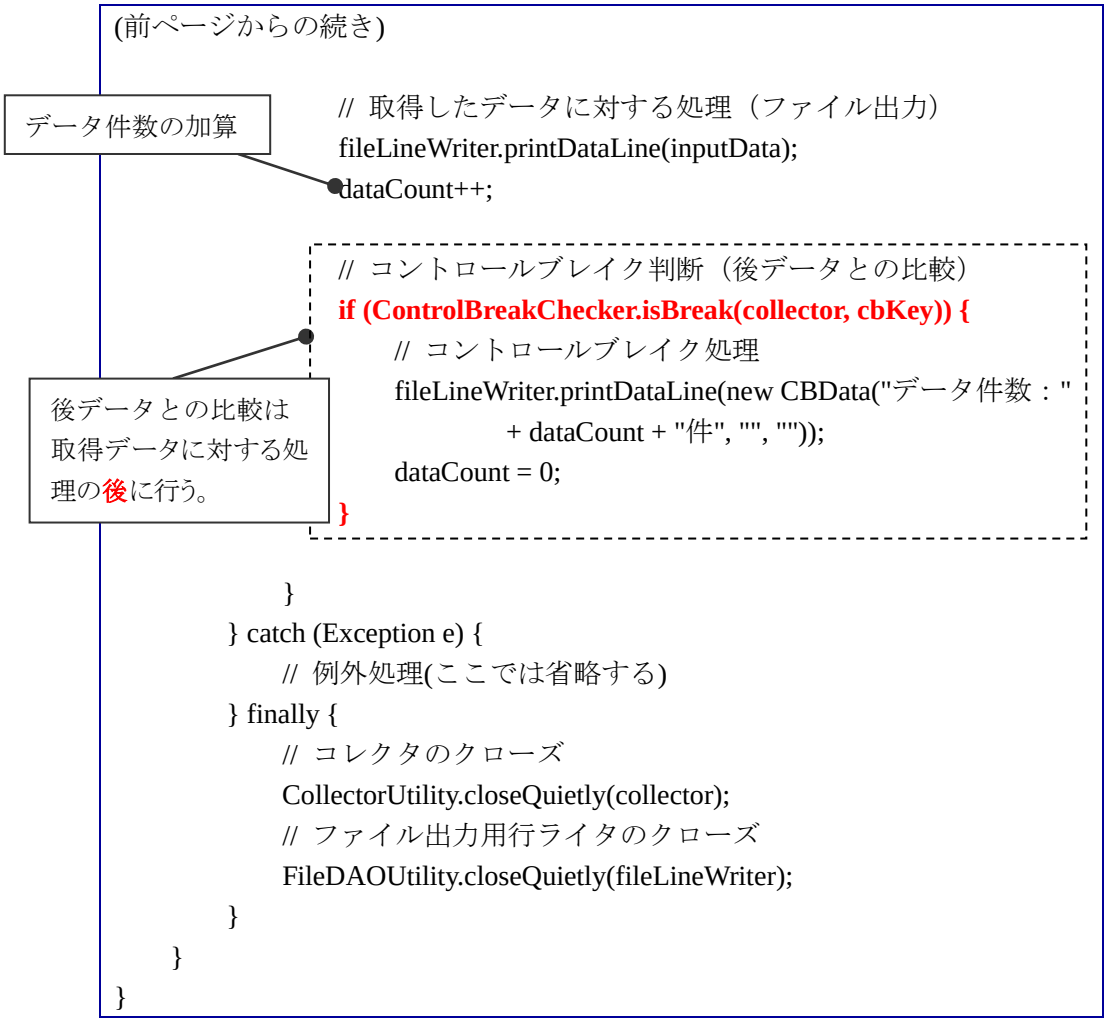
前データとの比較の際は
isPreBreak メソッドを使用する。

getPreBreakKey メソッドで切り
替わった値を Map 型で取得で
きる。
具体例(4行目のデータ処理前)

keyMap

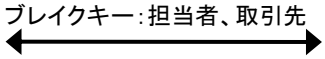
key	value
tantousya	佐藤

今回は取得した Map から
担当者名を取得し、
ヘッダに出力している。



- コントロールブレイク処理の実装例(複数のコントロールブレイクキー)

➢ コントロールブレイクキーを複数設定した場合は、上位のキーは下位のキーを包含する関係となる。
➢ すなわち、上位のキーが切り替わった際、例え下位のキーが切り替わっていなくとも、getPreBreakKey(もしくは getBreakKey)メソッドを呼び出すと、上位のコントロールブレイクキーの値と同時に、下位のコントロールブレイクキーの値を取得できる。

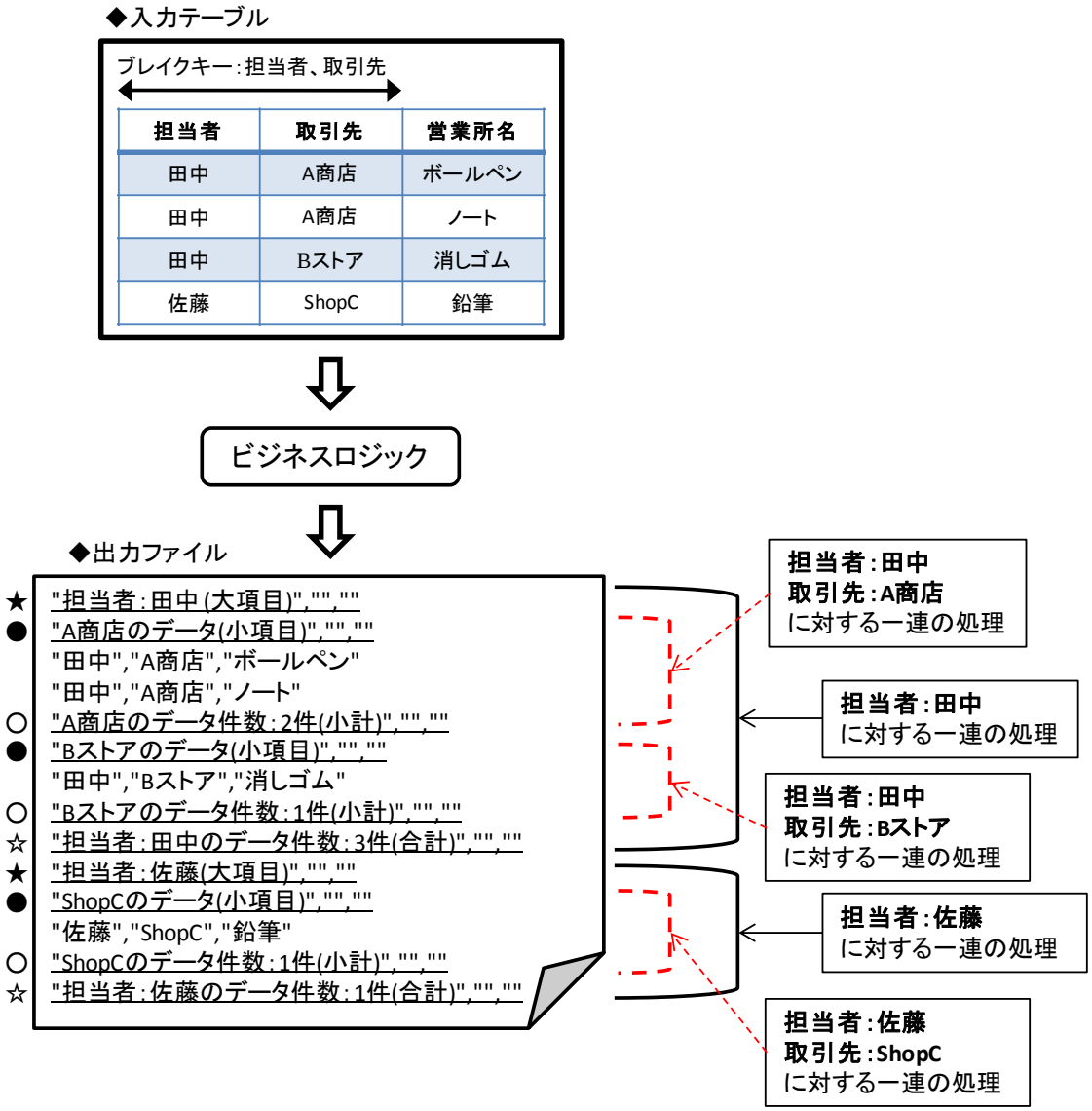


担当者	取引先	営業所名
田中	A商店	ボールペン
佐藤	Bストア	ノート
佐藤	ShopC	消しゴム
鈴木	ShopC	鉛筆

➢ 上記の表に対して具体的な例を挙げる。
  ✧ 上記の表では 3 行目と 4 行目の間で上位キーである担当者の値が「佐藤」から「鈴木」へと切り替わるが、下位キーである取引先の値は、3 行目も 4 行目も「ShopC」のままである。
  ✧ しかしながら、上位キーは下位キーと包含関係にあるため、担当者の切り替わる時に getPreBreakKey メソッド、もしくは getBreakKey メソッドを呼び出した場合は「担当者」の値だけでなく、実際には値の変更のない「取引先」の値も取得できる。
  ✧ その結果、担当者・取引先の 2 つのコントロールブレイクキーを取得し、取得したそれぞれのキーに対してコントロールブレイク処理を実施することが可能である。

➢ ブレイクキーを「担当者」「取引先」の 2 つを設定した場合のビジネスロジックの処理イメージ・実装例を次ページから掲載する。

➢ ビジネスロジックの処理イメージ



[実装イメージ中の記号の意味]

★…担当者、取引先が切り替わった時に動作する、「**前データ**との比較」によるコントロールブレイク処理による出力行

●…取引先が切り替わった時に動作する、「**前データ**との比較」によるコントロールブレイク処理による出力行

☆…担当者、取引先が切り替わった時に動作する、「**後データ**との比較」によるコントロールブレイク処理による出力行

○…取引先が切り替わった時に動作する、「**後データ**との比較」によるコントロールブレイク処理による出力行

➢ ビジネスロジックの実装例

```
@Component
public class B102BLogic extends AbstractTransactionBLogic {

    @Inject
    B101Dao b101Dao;

    @Inject
    @Named("csvFileUpdateDAO")
    FileUpdateDAO csvFileUpdateDAO;

    @Override
    public int doMain(BLogicParam param) {

        // コレクタの生成
        Collector<CBData> collector = new DaoCollector<CBData>(
                this.b101Dao, "collectData02", null);

        // ファイル出力用行ライタの取得
        FileLineWriter<CBData> fileLineWriter = csvFileUpdateDAO.execute(
                "outputFile/SampleOutput02.csv", CBData.class);

        // 担当者データ件数カウント用(合計用)
        private int dataCount1 = 0;

        // 取引先データ件数カウント用(小計用)
        private int dataCount2 = 0;

        // コントロールブレイクキーの設定
        String[] cbKey = new String[] { "tantousya", "shopName" };
```

(次ページに続く)

複数のコントロールブレイクキーを設定する場合は、String 型の配列で生成する

getPreBreakKey メソッドで切り替わった値を Map 型で取得できる。
具体例(4行目のデータ処理前)

keyMap

key	value
tantousya	佐藤
shopName	ShopC

(前ページからの続き)

```
try {
    while (collector.hasNext()) {

        // データの取得
        CBData inputData = collector.next();

        // コントロールブレイク判断（前データとの比較）
        if (ControlBreakChecker.isPreBreak(collector, cbKey)) {
            // コントロールブレイクキー取得
            Map<String, Object> keyMap = ControlBreakChecker
                    .getPreBreakKey(collector, cbKey);

            // コントロールブレイク処理(大項目の出力)
            if (keyMap.containsKey(cbKey[0])) {
                fileLineWriter.printDataLine(new CBData("担当者："
                    + keyMap.get(cbKey[0]).toString() + "(大項目)",
                        "", ""));
            }

            // コントロールブレイク処理(小項目の出力)
            if (keyMap.containsKey(cbKey[1])) {
                fileLineWriter.printDataLine(new CBData(
                    keyMap.get(cbKey[1]).toString() +
                      "のデータ(小項目)", "", ""));
            }
        }

        // 取得したデータに対する処理（ファイル出力）
        fileLineWriter.printDataLine(inputData);
        dataCount1++;
        dataCount2++;
```

if 文を使用して、
取得できた keyMap に応じて
コントロールブレイク処理を行う。

データ件数の加算

(次ページに続く)

後データとの比較の場合、getBreakKey メソッドで切り替わった値を Map 型で取得できる。
具体例(3 行目のデータ処理後)

keyMap

key	value
tantousya	田中
shopName	Bストア

```
(前ページからの続き)
            // コントロールブレイク判断（後データとの比較）
            if (ControlBreakChecker.isBreak(collector, cbKey)) {
                // コントロールブレイクキー取得
                Map<String, Object> keyMap = ControlBreakChecker
                        .getBreakKey(collector, cbKey);

                // コントロールブレイク処理(小計)
                if (keyMap.containsKey(cbKey[1])) {
                    fileLineWriter.printDataLine(new CBData(
                            + keyMap.get(cbKey[1])
                            + "のデータ件数：" + dataCount2
                            + "件(小計)", "", ""));
                    dataCount2 = 0;
                }

                // コントロールブレイク処理(合計)
                if (keyMap.containsKey(cbKey[0])) {
                    fileLineWriter.printDataLine(new CBData("担当者："
                            + keyMap.get(cbKey[0])
                            + "のデータ件数：" + dataCount1
                            + "件(合計)", "", ""));
                    dataCount1 = 0;
                }
            }
        } catch (Exception e) {
            // 例外処理(ここでは省略する)
        } finally {
            // コレクタのクローズ
            CollectorUtility.closeQuietly(collector);
            // ファイル出力用行ライタのクローズ
            CollectorUtility.closeQuietly(fileLineWriter);
        }
    }
}
```

前データとの比較の際と同様に取得したコントロールブレイクキーに合わせてフッタを出力する。

取引先のデータ件数のリセット

担当者のデータ件数のリセット

- コントロールブレイク機能を使用する場合の注意点
  - 入力データは、コントロールブレイクキーでソートしておくこと。
  - DB から入力する場合は、入力時に ORDER BY 句を利用してソートすること。
  - ファイルから入力する場合は、ソートされたデータが格納されたファイルから入力すること。
- コントロールブレイク機能が持つメソッドの一覧、及び解説
  - ControlBreakChecker クラスのメソッド一覧

メソッド名	引数	戻り値	解説
isPreBreak	(Collector<?>, String...)	boolean	前データとの比較により、コントロールブレイクを判断するメソッド
isPreBreak	(Collector<?>, CompareStrategy<?>[], String[])	boolean	前データとの比較により、コントロールブレイクを判断するメソッド
isBreak	(Collector<?>, String...)	boolean	後データとの比較により、コントロールブレイクを判断するメソッド
isBreak	(Collector<?>, CompareStrategy<?>[], String[])	boolean	後データとの比較により、コントロールブレイクを判断するメソッド
getPreBreakKey	(Collector<?>, String...)	Map<String, Object>	前データとの比較の際に、コントロールブレイクキーを取得するメソッド
getPreBreakKey	(Collector<?>, CompareStrategy<?>[], String[])	Map<String, Object>	前データとの比較の際に、コントロールブレイクキーを取得するメソッド
getBreakKey	(Collector<?>, String...)	Map<String, Object>	後データとの比較の際に、コントロールブレイクキーを取得するメソッド
getBreakKey	(Collector<?>, CompareStrategy<?>[], String[])	Map<String, Object>	後データとの比較の際に、コントロールブレイクキーを取得するメソッド

- ControlBreakChecker クラスのメソッドで使用する引数一覧

引数	解説	省略
Collector<?>	DB コレクタや、ファイルコレクタなどのコレクタ実装クラス。	不可
String…, String[]	コントロールブレイクキー。複数のブレイクキーを設定する際は、String の配列型で渡す。	不可
CompareStrategy<?>[]	コントロールブレイクキーの値の比較方法を実装したクラス。複数のブレイクキーを設定する際は、ブレイクキーごとに設定可能。たとえば、Date 型のブレイクキー項目があり、月の切り替わりでコントロールブレイク処理を行いたい場合は、年と月のみを比較する CompareStrategy 実装クラスを作成し、引数に与える。	可

■ リファレンス

◆ 構成クラス

	クラス名	概要
1	jp.terasoluna.fw.collector.util.ControlBreakChecker	コントロールブレイク判定クラス。 前データとの比較と後データとの比較の 2 種類の方法により、コントロールブレイクを判断する。
2	jp.terasoluna.fw.collector.util.strategy.CompareStrategy	2 つのオブジェクトが等しいか等しくないかを判断する方法を実装/提供するためのインタフェース。
3	jp.terasoluna.fw.collector.util.strategy.ComparatorCompareStrategy	CompareStrategy 実装クラス。 外部 Comparator の compare メソッドで比較するストラテジ。
4	jp.terasoluna.fw.collector.util.strategy.EqualsCompareStrategy	CompareStrategy 実装クラス。 比較対象オブジェクトの equals メソッドで比較するストラテジ。

◆ 拡張ポイント

なし

■ 関連機能

- 『AL-041 入力データ取得機能』
- 『AL-043 入力チェック機能』

■ 使用例

- **機能網羅サンプル**(terasoluna-batch-functionsample)

■ 備考

◆ 例外発生データ検出時の振る舞い

- データ入力に成功後、入力チェックエラーにより、Collector#next 実行時に入力チェック例外が発生するケースでは、入力されたデータを参照してコントロールブレイク判定を行う。
- データ入力が正常にできておらず、Collector#next 実行時に例外(データ入力時に発生した例外。FileLineException 等)が発生するケースでは、コントロールブレイク判定時にも例外(データ入力時に発生した例外)をスローする。