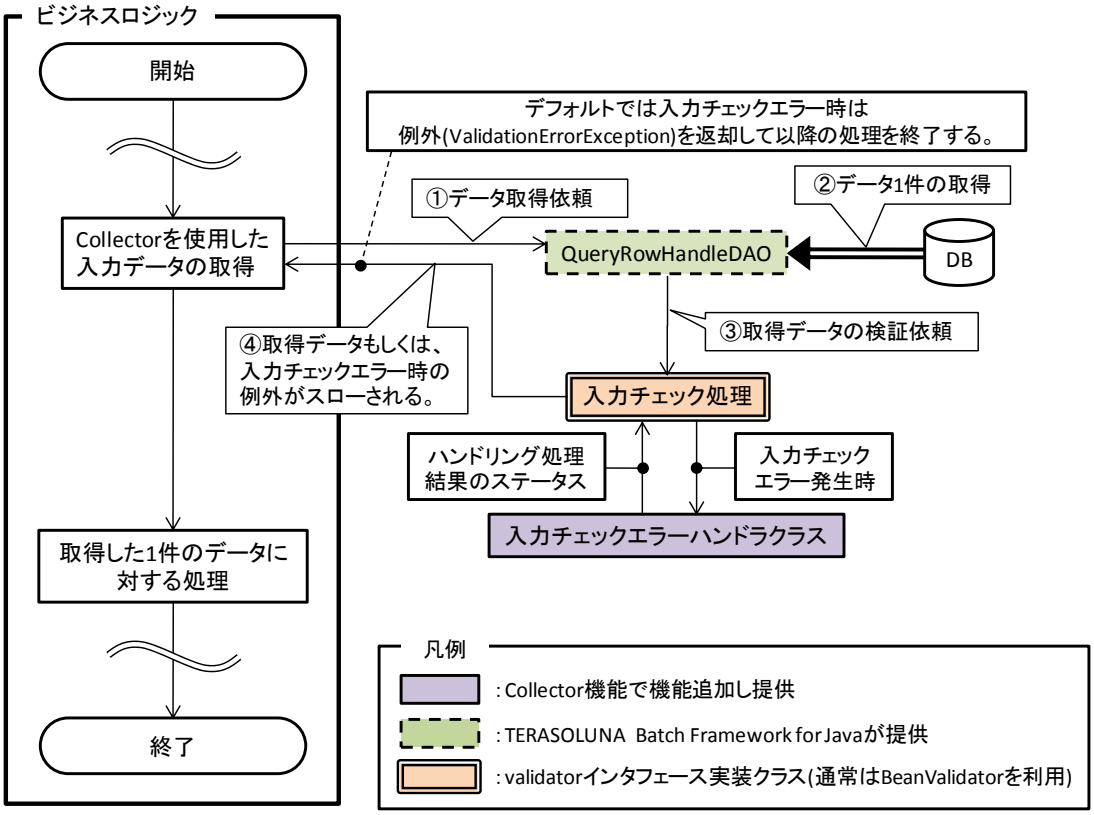
AL-043　入力チェック機能

■ 概要

◆ 機能概要

- 「AL-041 入力データ取得機能」を使用した際に、DB やファイルから取得したデータ 1 件毎に入力チェックを行う機能を提供する。
- 入力チェックは、DB やファイルからデータを取得するタイミングで行われる。
- 本機能を使用することで、コントロールブレイク判断に空レコードが渡されることの回避が可能となる。
- 本機能では、TERASOLUNA Server Framework for Java (Rich 版)の機能である「RF-02 入力チェック機能」と同様の入力チェックを実施することができる。

◆ 概念図



◆ 解説

① Collector はデータの取得を QueryRowHandleDAO に依頼する。
② QueryRowHandleDAO は DB からデータを 1 件取得する。
③ QueryRowHandleDAO は取得したデータを返却する前に、validator インタフェース実装クラスに入力チェック処理を依頼する。
④ validator インタフェース実装クラスは入力チェックの結果に応じて、処理を振り分ける。
- 正常系の場合…取得データをビジネスロジックに返却する。
- 異常系の場合…入力チェックエラーハンドラクラスによって入力チェックエラー時の例外「ValidationErrorException」がビジネスロジックに返却される。

- この時、独自に作成した拡張入力チェックエラーハンドラクラスを使用することによって、例外「ValidationErrorException」をスローすることなく以降の処理を継続させることも可能である。
- 拡張入力チェックエラーハンドラクラスを作成する場合は、拡張ポイントの項目を参照すること。

◆ コーディングポイント

【コーディングポイントの構成】

- 入力チェックを行う場合のビジネスロジックの実装例
  - ビジネスロジックの実装例(DB からのデータ取得)
  - ビジネスロジックの実装例(ファイルからのデータ取得)
- validation.xml(入力チェックルール)の設定例
- 本機能が提供する、入力チェックエラーハンドラクラスについて
- 入力チェック対応 Collector クラスのコンストラクタについて
  - コンストラクタで設定できる内容について
  - 入力チェック対応 Collector クラスのコンストラクタ一覧
  - コンストラクタ引数一覧

- 入力チェックを行う場合のビジネスロジックの実装例
  以下に DB からデータを取得する際に入力チェックを行う際の実装例を掲載する。
  使用する Collector クラスが、入力チェックを行わない場合と異なる点に注意する。
  - ビジネスロジックの実装例(DB からのデータ取得)
    (TERASOLUNA Batch Framework for Java ver 3.x の場合)

```
@Component
public class Sample01BLogic extends AbstractTransactionBLogic {

    @Autowired
    protected QueryRowHandleDAO queryRowHandleDAO;

    @Autowired
    protected Validator validator;

    @Override
    public int doMain(BLogicParam param) {

        // Collector の生成
        Collector<Sample01Bean> collector = new DBValidateCollector<Sample01Bean>(
                this.queryRowHandleDAO, "Sample.selectData01", null, validator);
        try {
            Sample01Bean inputData = null;
            while (collector.hasNext()) {
                // データの取得
                inputData = collector.next();

                // 取得データに対する処理
                // (ここでは省略する)
            }
        } catch (Exception e) {
            // 例外処理
        } finally {
            // Collector のクローズ
            CollectorUtility.closeQuietly(collector);
        }
        return 0;
    }
}
```

BeanValidator の DI を行う

DBValidateCollector を生成する。
DBCollector と異なり、第四引数に入力チェックを行う BeanValidator クラスを渡している点に注意すること。

入力チェックの実行や、入力チェックエラーハンドラの実行は、非同期で事前に対象データが取得されたタイミングで行われ、next メソッドを実行した際に、次に取得する対象データの入力チェック結果がフィードバックされる。Collector クラスのコンストラクタにて、(拡張)入力チェックエラーハンドラを渡していない場合、入力チェックエラー発生時に「ValidationErrorException」がスローされる。

必ず処理の最後にコレクタをクローズすること

- ビジネスロジックの実装例(ファイルからのデータ取得)
  (TERASOLUNA Batch Framework for Java ver 3.x の場合)

```
@Component
public class Sample02BLogic extends AbstractTransactionBLogic {

    @Autowired
    @Qualifier(value = "csvFileQueryDAO")
    protected FileQueryDAO csvFileQueryDAO;

    @Autowired
    protected Validator validator;

    @Override
    public int doMain(BLogicParam param) {
        // Collector の生成
        Collector<Sample02Bean> collector = new FileValidateCollector<Sample02Bean>(
                this.csvFileQueryDAO, "inputFile/SampleFile.csv", Sample02Bean.class, validator);

        try {
            Sample02Bean inputData = null;
            while (collector.hasNext()) {

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

                // DB の更新など、取得データに対する処理を記述する(実装は省略)
            }
        } catch (Exception e) {
            // 例外処理
        } finally {
            // Collector のクローズ
            CollectorUtility.closeQuietly(collector);
        }
        return 0;
    }
}
```

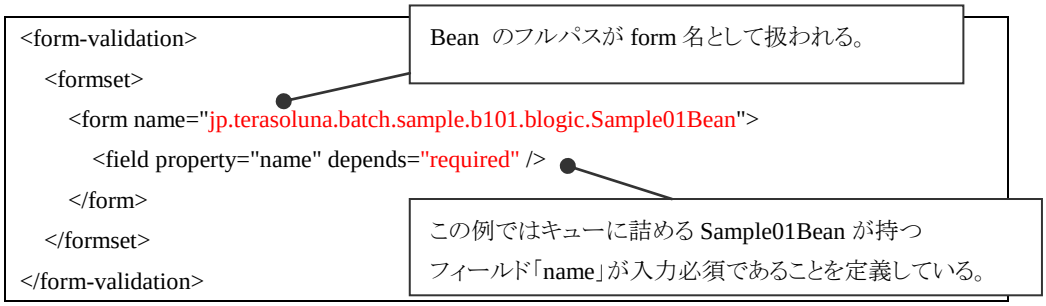
BeanValidator の DI を行う

FileValidateCollector を生成する。
FileCollector と異なり、第四引数に入力チェックを行う BeanValidator クラスを渡している点に注意すること。

入力チェックの実行や、入力チェックエラーハンドラの実行は、非同期で事前に対象データが取得されたタイミングで行われ、next メソッドを実行した際に、次に取得する対象データの入力チェック結果がフィードバックされる。
Collector クラスのコンストラクタにて、(拡張)入力チェックエラーハンドラを渡していない場合、入力チェックエラー発生時に「ValidationErrorException」がスローされる。

必ず処理の最後に Collector をクローズすること

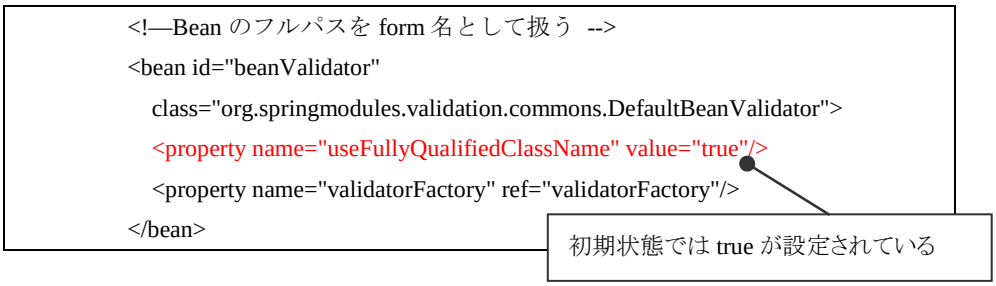
- validation.xml(入力チェックルール)の設定例
  form 要素の name に入力チェック対象の Bean のフルパスを設定し、field 要素にフィールドに対する入力チェックルールを設定する。
  先ほど実装例を掲載した「ビジネスロジックの実装例(DB からのデータ取得)」に合わせた、入力チェックルールの設定例を以下に掲載する。



TERASOLUNA Batch Framework for Java が提供する入力チェック機能は、TERASOLUNA Server Framework for Java (Rich 版)で提供しているものと同一であるため、validation.xml は以下の機能説明書を参照して設定すること(※)。

- ✧ TERASOLUNA Server Framework for Java (Rich 版)
  RF-02 入力チェック機能
  - ◆コーディングポイント
  - ◆入力チェックルール解説

(※)ただし、TERASOLUNA Server Framework for Java (Rich 版)では BeanValidator の Bean 定義の際に、プロパティ「useFullyQualifiedClassName」の設定を行っていないため、form 要素の name にクラス名を設定していることに注意する。
TERASOLUNA Batch Framework for Java でもこの設定変更は有効だが、初期状態では以下の通り true に設定されているため、Bean のフルパスを設定する必要がある。



本機能が提供する、入力チェックエラーハンドラクラスについて

入力チェックエラーハンドラクラス	仕様
ExceptionValidationErrorHandler	デフォルトで使用される入力チェックエラーハンドラクラス。入力チェックエラーが発生した時点で例外をスローする。この入力チェックエラーハンドラは以下の場合に使用する。 ✓ 入力チェックエラー検出時に処理を異常終了させる場合 ✓ 入力チェックエラー例外を呼び出し元でハンドリングして、処理を継続(次のデータを処理)する場合

- 入力チェック対応 Collector クラスのコンストラクタについて
  DBValidateCollector と FileValidateCollector が用意するコンストラクタと、コンストラクタに使用される引数の一覧を掲載する。
  - コンストラクタで設定できる内容について
    実装例で使用した基本的なコンストラクタの他に、引数を与えることにより、以下の項目を設定することが可能である。
    ① iBATIS の groupBy 属性使用の有無(DB のみ) (※１)
    ② キューサイズ
    ③ 拡張例外ハンドラクラス(※２)
    ④ 使用する入力チェックエラーハンドラクラス(※３)

    ※1. iBATIS の groupBy 属性を使用することによって、1:N 関係にあるテーブルの内容を、１つのクエリーで取得することができる。
    詳細は iBATIS の機能説明書 P42 の「N+1 Selects を回避する」の項目を参照の事。(http://ibatis.apache.org/docs/java/pdf/iBATIS-SqlMaps-2_ja.pdf)
    ※2. 拡張例外ハンドラクラスに関しては、「AL-041 入力データ取得機能」の機能説明書の拡張ポイントの項目を参照すること。
    ※3. デフォルトでは先に紹介した「ExceptionValidationErrorHandler」が使用される、独自にハンドラクラスを作成することも可能。
    ハンドラクラスを独自実装する場合は後述の拡張ポイントの項目を参照の事。

- 入力チェック対応 Collector クラスのコンストラクター一覧
  先ほどの番号と合わせて以下にコンストラクタを列挙し、概要を掲載する。
  引数についての詳細は、次ページのコンストラクタ引数一覧を参照すること。
  - DBValidateCollector のコンストラクター一覧

コンストラクタ	概要
DBValidateCollector<P>(QueryRowHandleDAO, String, Object, Validator)	実装例で掲載した基本となるコンストラクタ これら４つの引数は必須である。
DBValidateCollector<P>(QueryRowHandleDAO, String, Object, Validator, ValidationErrorHandler)	基本となるコンストラクタ及び、 使用する入力チェックエラー ハンドラクラスを設定する。
DBValidateCollector<P>(QueryRowHandleDAO, String, Object, boolean, Validator)	基本となるコンストラクタ及び、 1:N マッピング使用の有無を設定する。
DBValidateCollector<P>(QueryRowHandleDAO, String, Object, boolean, Validator, ValidationErrorHandler)	基本となるコンストラクタ及び、 1:N マッピング使用の有無、 使用する入力チェックエラーハンドラクラスを設定する。
DBValidateCollector<P>(QueryRowHandleDAO, String, Object, int, Validator)	基本となるコンストラクタ及び、 キューサイズを設定する。
DBValidateCollector<P>(QueryRowHandleDAO, String, Object, int, Validator, ValidationErrorHandler)	基本となるコンストラクタ及び、 キューサイズ、 使用する入力チェックエラーハンドラクラスを使用する。
DBValidateCollector<P>(QueryRowHandleDAO, String, Object, int, CollectorExceptionHandler, Validator)	基本となるコンストラクタ及び、 キューサイズ、 拡張例外ハンドラクラスを設定する。
DBValidateCollector<P>(QueryRowHandleDAO, String, Object, int, CollectorExceptionHandler, Validator, ValidationErrorHandler)	基本となるコンストラクタ及び、 キューサイズ、 拡張例外ハンドラクラス、 使用する入力チェックエラーハンドラクラスを設定する。
DBValidateCollector<P>(QueryRowHandleDAO, String, Object, int, boolean, CollectorExceptionHandler, Validator)	基本となるコンストラクタ及び、 1:N マッピング使用の有無、 キューサイズ、 拡張例外ハンドラクラスを設定する。
DBValidateCollector<P>(QueryRowHandleDAO, String, Object, int, boolean, CollectorExceptionHandler, Validator, ValidationErrorHandler)	基本となるコンストラクタ及び、 1:N マッピング使用の有無、 キューサイズ、 拡張例外ハンドラクラス、 使用する入力チェックエラーハンドラクラスを設定する。

✧ FileValidateCollector のコンストラクタ一覧

コンストラクタ	概要
FileValidateCollector<P>(FileQueryDAO, String, Class<P>, Validator)	実装例で掲載した基本となるコンストラクタ これら４つの引数は必須である。
FileValidateCollector<P>(FileQueryDAO, String, Class<P>, Validator, ValidationErrorHandler)	基本となるコンストラクタ及び、 使用する入力チェックエラーハンドラクラスを設定する。
FileValidateCollector<P>(FileQueryDAO, String, Class<P>, CollectorExceptionHandler, Validator)	基本となるコンストラクタ及び、 拡張例外ハンドラクラスを設定する。
FileValidateCollector<P>(FileQueryDAO, String, Class<P>, CollectorExceptionHandler, Validator, ValidationErrorHandler)	基本となるコンストラクタ及び、 拡張例外ハンドラクラス、 使用する入力チェックエラーハンドラクラスを設定する。
FileValidateCollector<P>(FileQueryDAO, String, Class<P>, int, CollectorExceptionHandler, Validator)	基本となるコンストラクタ及び、 キューサイズ、 拡張例外ハンドラクラスを設定する。
FileValidateCollector<P>(FileQueryDAO, String, Class<P>, int, CollectorExceptionHandler, Validator, ValidationErrorHandler)	基本となるコンストラクタ及び、 キューサイズ、 拡張例外ハンドラクラス、 使用する入力チェックエラーハンドラクラスを設定する。

➢ コンストラクタ引数一覧

前ページで列挙したコンストラクタで使用される引数を以下に列挙する。
Collector 機能からの新規要素については**太字**で掲載する

✧ DBValidateCollector のコンストラクタで渡される引数

引数	解説	デフォルト値	省略
QueryRowHandleDAO	DB にアクセスするための DAO	―	不可
String	SqlMap で定義した SQLID	―	不可
Object	SQL にバインドされる値を格納したオブジェクト、バインドする値が存在しない場合は省略せず、null を渡すこと。	―	不可
int	キューサイズ、0 以下の値は無視される。基本的に変更不要。	20	可
CollectorExceptionHandler	例外ハンドラクラス、	Null	可
boolean	iBATIS の 1:N マッピング使用時は true を渡す。 true にすることにより、メモリの肥大化を最小限に抑えることができる。	false	可
Validator	**入力チェックを行う Validator。 通常は Spring が提供する BeanValidator(Validator インタフェース実装クラス)を使用する。**	―	不可
ValidationErrorHandler	**入力チェックエラーハンドラクラス。**	**ExceptionValidationErrorHandler**	**可**

✧ FileValidateCollector のコンストラクタで渡される引数

引数	解説	デフォルト値	省略
FileQueryDAO	ファイルにアクセスするための DAO	―	不可
String	読み込むファイル名	―	不可
Class<P>	ファイル行オブジェクトクラス	―	不可
int	キューサイズ、0 以下の値は無視される。基本的に変更不要。	20	可
CollectorExceptionHandler	例外ハンドラクラス、	Null	可
Validator	**入力チェックを行う Validator。 通常は Spring が提供する BeanValidator(Validator インタフェース実装クラス)を使用する。**	―	不可
ValidationErrorHandler	**入力チェックエラーハンドラクラス。**	**ExceptionValidationErrorHandler**	**可**

拡張ポイント

- 拡張入力チェックエラーハンドラクラスを独自実装する方法
  - ValidationErrorHandler インタフェースの実装クラスを作成することにより、拡張入力チェックエラーハンドラクラスを作成することが可能である。
  - 拡張入力チェックエラーハンドラクラスは、ExceptionValidationErrorHandler のように例外をスローする他、以降の処理を制御するステータス ValidateErrorStatus を返却することができる。
  - ValidateErrorStatus の一覧表
    (入力チェックエラーハンドリングクラスが返却するステータス)

ValidateErrorStatus	Collector の next メソッドの取得対象が入力チェックエラーデータの場合の挙動	Collector の getNext (getPrevious) メソッドの取得対象が入力チェックエラーデータの場合の挙動
SKIP	**エラーデータは取得せずに**、その後の正常なデータを取得する。その後の処理は継続する。	**エラーデータは取得せずに**、その後の正常なデータを取得する。
CONTINUE	**エラーデータを取得する。**その後の処理は継続する。	**エラーデータを取得する。**
END	エラーデータは取得せず、以降のデータも取得しない。 ※事前の hasNext による問合せに false を返す。	getNext は **null** を返却する(次のデータが存在しない、終端を意味する)。 getPrevious では参照できない。
なし (例外がスローされた場合)	例外がスローされる。ビジネスロジックで処理を停止しない限り、処理は継続する。	**エラーデータを取得する。**

※コントロールブレイク機能では、コントロールブレイク判定時に使用されるデータは後ブレイク判定の場合は getNext メソッド、前ブレイク判定の場合は getPrevious メソッドで前後のデータを取得し、ブレイク判定を行っている。コントロールブレイク判定時の比較対象のデータ getNext、getPrevious メソッドの返却値を意識すること。

- 以下に拡張入力チェックエラーハンドラクラスの実装例を掲載する。
  実装例では拡張入力チェックエラーハンドラクラスは以下の仕様で作成する。
  【仕様】
  ① 入力チェックエラー発生時にログレベル info でエラー発生を通知する。
  ② 入力チェックエラーが発生したデータは無視し、以降の処理を継続する。
- 拡張入力チェックエラーハンドラクラス実装例

```
public class CustomValidationErrorHandler implements ValidationErrorHandler {

    private static Log logger =
        LogFactory.getLog(CustomValidationErrorHandler.class);

    @Override
    public ValidateErrorStatus handleValidationError(
            DataValueObject dataValueObject, Errors errors) {

        // ログ出力
        if(logger.isInfoEnabled()){
            logger.info("入力チェックエラー発生");
        }

        // ValidateErrorStatus の設定
        return ValidateErrorStatus.SKIP;
    }
}
```

ValidationErrorHandler インタフェースを実装する。

handleValidationError メソッドの実装を行う

仕様①に従い、info レベルで
エラーの発生を通知する

仕様②に従い、SKIP を返却することにより、
エラーが発生したデータを無視して、
その後の処理を継続する。

➢ ビジネスロジックの実装例(DB)
(TERASOLUNA Batch Framework for Java ver 3.x の場合)

```
@Component
public class Sample03BLogic extends AbstractTransactionBLogic {

    @Autowired
    protected QueryRowHandleDAO queryRowHandleDAO;

    @Autowired
    private Validator validator;

    CustomValidationErrorHandler handler = new CustomValidationErrorHandler()

    @Override
    public int doMain(BLogicParam param) {

        // Collector の生成
        Collector<Sample03Bean> collector = new DBValidateCollector<Sample03Bean>(
                this.queryRowHandleDAO, "Sample.selectData06", null,
                validator, handler);

        try {
            Sample03Bean inputData = null;
            while (collector.hasNext()) {

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

                // ここにファイルの出力など、
                // 取得データに対する処理を
                // 記述する

            }
        } catch (Exception e) {
            // 例外処理
        } finally {
            // Collector の破棄
            CollectorUtility.closeQuietly(collector);
        }
    return 0;
    }
}
```

BeanValidator の DI を行う

独自実装した拡張入力チェックエラーハンドラクラスのインスタンスを生成する。

コレクタ生成時に上で生成した拡張入力チェックエラーハンドラクラスを渡しておく。

入力チェックの実行や、拡張入力チェックエラーハンドラの実行は、非同期で事前に対象データが取得されたタイミングで行われ、next メソッドを実行した際に、次に取得する対象データの入力チェック結果がフィードバックされる。拡張入力チェックエラーハンドラが SKIP や END を返す場合、next メソッドで取得できる件数自体が変わるため、入力チェック結果は、hasNext メソッドにも影響を及ぼす。例えば、SKIP や END の結果、next メソッドが参照できるデータが 1 つもなくなるケースにおいては、直前の hasNext メソッド呼び出し時に false を返す。

このように Collector インスタンス生成時にあらかじめ拡張入力チェックエラーハンドラクラスを渡すことにより、入力チェックエラー発生時にはこのハンドラクラスが使用されることになる。

■ リファレンス

◆ 構成クラス

	クラス名	概要
1	jp.terasoluna.fw.collector.db.DBValidateCollector	DBCollector 拡張クラス DBCollector を入力チェックに対応させている。
2	jp.terasoluna.fw.collector.file.FileValidateCollector	FileCollector 拡張クラス FileCollector を入力チェックに対応させている。
3	jp.terasoluna.fw.collector.validate.ValidationErrorHandler	入力チェックエラーハンドラインタフェース 入力チェックエラーが発生した際の処理を宣言している。
4	jp.terasoluna.fw.collector.validate.AbstractValidationErrorHandler	ValidationErrorHandler クラスを実装した抽象クラス コンストラクタによるログレベルの変更や、ログ出力用のメソッドなどの処理を定義している。
5	jp.terasoluna.fw.collector.validate.ExceptionValidationErrorHandler	AbstractValidationErrorHandler クラスの拡張クラス 入力チェックエラーが発生した場合は TRACE ログにエラーコードを出力し、例外をスローする(処理が途中で中断する)
6	jp.terasoluna.fw.collector.validate.ValidateErrorStatus	列挙型クラス 入力チェックエラーハンドラクラスはこの値によって、入力チェックエラー発生後の挙動を決定する。
7	jp.terasoluna.fw.collector.validate.ValidationErrorException	RuntimeException を拡張した入力チェックエラークラス 入力チェックエラー発生時にスローされる。

◆ 関連機能

- 『AL041 入力データ取得機能』

◆ 使用例

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

◆ 注意事項

- 数値範囲（intRange）のチェックをする場合
  入力値が数値（int に変換可能）の場合は正常に数値範囲の検証が行われるが、入力値が数値以外（int に変換不可）の場合はフレームワークによって挙動が異なる。TERASOLUNA Server Framework for Java (Rich 版)、TERASOLUNA Batch Framework for Java では、チェックエラーとなる。一方、TERASOLUNA Server Framework for Java (Web 版)では、NumberFormatException が発生し不正終了する。そのため、数値範囲チェックに加えて数値チェックも実施するなどの対応が必要となる。

◆ 備考

- なし