入力チェック

方針

- 単項目の検証
  - Bean Validationで検証する
  - 次のBean Validation用アノテーションを使用する
    - javax.validation.constraints
    - org.hibernate.validator.constraints
    - jp.co.nssol.dukenavire.validator
- 複数項目の検証、条件付きの単項目の検証など
  - SpringのValidatorインターフェースを実装したFormValidatorクラスで検証する
  - 単純な複数項目の検証であれば、Bean Validation用アノテーションがあることに留意 (e.g. DNValidCompareField)
- 業務ロジックを伴う検証
  - 検証ロジックはサービス層で実装すること
  - サービス層で実装した検査用ロジックを使ったBean Validationアノテーションを作りたい場合はAP基盤に連絡すること
    - 参考: 4.1.3.2.3. 業務ロジックチェック — 4.1. 入力チェック — TERASOLUNA Server Framework for Java (5.x) Development Guideline 5.3.0.RELEASE documentation

Bean Validationによる検証

次の様にFormクラスのフィールドにBean Validation用のアノテーションを付与する。

```
public class DBAS0010P01PForm implements Serializable {

    private static final long serialVersionUID = 1L;

    @NotNull //  <1>
    private String foo;

    @LengthMax(5)
    private String bar//  <2>

    @NotNull
    @Min(1)
    private Integer baz;

    @NotNull
    @Valid //  <3>
    private DBAS0010P01PQuxForm sub;

    @Valid //  <3>
    private List<DBAS0010P01PQuxForm> list;

    //  getter/setter 省略
}

public class DBAS0010P01PQuxForm implements Serializable {

    private static final long serialVersionUID = 1L;

    @NotNull
    private String hoge;

    @NotNull
    private Integer fuga;

    //  getter/setter 省略
}
```

- <1> Formクラスのフィールドに Bean Validation用アノテーションを付ける

- <2>
  NotNullアノテーション以外は未入力(null)の場合に検証OKになる。従って、この例ではbarフィールドが未入力の場合は検証OKで、barフィールドに6文字以上入力された場合は検証NGになる。
- <3> ネストしたFormオブジェクトやコレクション内のFormオブジェクトに対してBean Validationを実行したい場合はjavax.validation.Validアノテーションを付ける

そして、次の様にControllerのリクエストマッピングメソッドの引数でValidatedアノテーションを付与することでバリデーションを実行する。

```
@Controller
@RequestMapping("foo/DBAP0010")
public class DBAP0010PController   {

    @PostMapping("process")
    public String   process(@Validated   DBAS0010P01PForm   form,   BindingResult   bindingRes//   <1>
        if   (bindingResult.hasErrors())   {
            return "DBAS0010P01P";
                    }

        return "redirect:DBAP0010/complete";
        }

}
```

- <1> バリデーションを実行するForm引数にValidatedアノテーションを付与し、その引数の右隣にBindingResult引数を設ける
  - BindingResult引数は必ずバリデーション対象引数の1つ右である必要がある

Bean Validationアノテーション一覧

利用可能なBean Validationアノテーションは以下の通り。
ここに掲載してないもので汎用的なアノテーションが必要であれば、AP基盤に申請すること。
アプリ固有のBean Validationは必要に応じてアプリ側でアノテーションをつくること。

検証	アノテーション	デフォルトメッセージキー	備考
必須	NotNull	javax.validation.constraints.NotNull.message	(*1)
必須	NotBlank	org.hibernate.validator.constraints.NotBlank.message	(*1)
半角	DNValidHalfWidthCharacters	dukenavire.validation.constraints.DNValidHalfWidthCharacters.message	
全角	DNValidFullWidthCharacters	dukenavire.validation.constraints.DNValidFullWidthCharacters.message	
半角大文字	Uppercase	jp.co.anas.atc.fw.core.validator.constraints.Uppercase.message	
半角小文字	Lowercase	jp.co.anas.atc.fw.core.validator.constraints.Lowercase.message	
半角数字	DNValidHalfWidthNumber	dukenavire.validation.constraints.DNValidHalfWidthNumber.message	
半角英字	DNValidHalfWidthAlphabet	dukenavire.validation.constraints.DNValidHalfWidthAlphabet.message	
半角英数字	DNValidHalfWidthAlphabetNumber	dukenavire.validation.constraints.DNValidHalfWidthAlphabetNumber.message	
半角記号	DNValidHalfWidthSymbol	dukenavire.validation.constrai	

		nts.DNValidHalfWidthSymbol.message	
半角数字記号	DNValidHalfWidthNumberSymbol	dukenavire.validation.constraints.DNValidHalfWidthNumberSymbol.message	
半角英字記号	DNValidHalfWidthAlphabetSymbol	dukenavire.validation.constraints.DNValidHalfWidthAlphabetSymbol.message	
半角英数字記号	DNValidHalfWidthCharacters	dukenavire.validation.constraints.DNValidHalfWidthCharacters.message	
全角カナ	AtcFullWidthKatakana	jp.co.anas.atc.fw.core.validator.constraints.AtcFullWidthKatakana.message	DNValidFullWidthKatakanaはチェック仕様がASWツアーに適してないので使用しない
半角カナ	DNValidHalfWidthKatakana	dukenavire.validation.constraints.DNValidHalfWidthKatakana.message	
日付	DNValidDateTimeFormat	dukenavire.validation.constraints.DNValidDateTimeFormat.message	(*2)
時刻	DNValidDateTimeFormat	dukenavire.validation.constraints.DNValidDateTimeFormat.message	(*2)
最小文字数	LengthMin	jp.co.anas.atc.fw.core.validator.constraints.LengthMin.message	
最大文字数	LengthMax	jp.co.anas.atc.fw.core.validator.constraints.LengthMax.message	
文字数範囲	Length	org.hibernate.validator.constraints.Length.message	min, max片方だけ指定する場合はLengthMin, LengthMaxを使うこと
クレジットカード番号	DNValidCreditCardFormat	dukenavire.validation.constraints.DNValidCreditCardFormat.message	
メールアドレス	Email	org.hibernate.validator.constraints.Email.message	(*2)
最大値	Max	javax.validation.constraints.Max.message	
最小値	Min	javax.validation.constraints.Min.message	
数値範囲	Range	org.hibernate.validator.constraints.Range.message	min, max片方だけ指定する場合はMin, Maxを使うこと
正規表現	Pattern	javax.validation.constraints.Pattern.message	(*2)
URL	URL	org.hibernate.validator.constraints.URL.message	(*2)
電話番号	DNValidTelephoneNumber	dukenavire.validation.constraints.DNValidTelephoneNumber.message	(*2)
郵便番号			必要があればAP基盤に申請してください。作成に時間はかかりません。
日付比較1	DNValidCompareDateFuture	dukenavire.validation.constraints.DNValidCompareDateFutur	

		e.message	
日付比較2	DNValidCompareDatePast	dukenavire.validation.constraints.DNValidCompareDatePast.message	
値比較	DNValidCompareFields	dukenavire.validation.constraints.DNValidCompareFields.message	
Windows31Jの文字集合	Windows31jCharacters	jp.co.anas.atc.fw.core.validator.constraints.Windows31jCharacters.message	

- (*1) NotEmptyやNotBlankではなく統一のためNotNullで必須チェックをすること
  - ATCアーキテクチャでは全てのフォームフィールド、リクエストパラメータをトリムして空文字列ならnullにしているため (参考: StringTrimmerEditor を共通的に適用している)
  - 但し、JSONに対してはStringTrimmerEditorは動作しないので、必要に応じてNotBlankで必須チェックすること
- (*2)
  メッセージの出しわけで困ったり、適用するフィールドの数が多くてバリデーション設定値が管理しづらい場合はアプリ側で独自のBean Validationアノテーションを作ること

※ /04_実装標準/05_実装ガイド/入力値検証ガイド.xlsx の 「2.(2)ア．単体検証仕様と設計書」
から変更しています。差分は次の通り。

- 「大文字」検証用のアノテーションを追加 (Uppercase)
- 「小文字」検証用のアノテーションを追加 (Lowercase)
- 「半角カナ」検証について明記
  - 中途半端に記述されていたため
- 「時刻」検証用のアノテーションにはPatternではなくDNValidDateTimeFormatを使う
- 「最小文字数」検証用のアノテーションを追加 (LengthMin)
- 「最大文字数」検証用のアノテーションを追加 (LengthMax)
- 「文字数範囲」検証用のアノテーションについて記述追加
  - 「最小文字数」、「最大文字数」のアノテーションを追加したため
- 「数値範囲」検証用のアノテーションについて記述追加
  - 元々記述が無かったが、文字数範囲チェックに合わせて追加
- 「最小バイト数」「最大バイト数」の検証について記載を削除
  - 元々「利用しない」としていたため
- 「Windows31Jの文字集合」検証用のアノテーションを追加 (Windows31jCharacters)
- 「全角カナ」検証の仕様を変更 (#6944)

FormValidatorによる検証

FormValidatorの実装方法

```
@Component //  <1>
public class DBAS0010P01PFormValidator implements Validator  //  <2>

    @Override
    public boolean  supportsClass<?>  clazz)  {
        return  DBAS0010P01PForm.class.isAssignableFrom(clazz)//  <3>
          }

    @Override
    public void  validateObject  target,  Errors  errors)  {
        if  (errors.hasFieldErrors"foo")  ||  errors.hasFieldErro"bar"))  //  <4>
            return;
                    }
                    DBAS0010P01PForm  form  =  (DBAS0010P01PForm)  target;
        String  foo  =  form.getFoo();
        String  bar  =  form.getBar();

        if  (foo.length  ==  bar.length)  {
                            ValidatorUtils.rejfoo", "{エラーコード}", "エラーメッセージパラメータ1", "
エラーメッセージパラメータ2"); //  <5>
                    }
```

```
        }
}
```

- <1> @Componentを付与し、Validatorをコンポーネントスキャン対象にする
- <2> org.springframework.validation.Validatorを実装する
- <3> バリデーション対象のクラスを指定する
- <4> 対象フィールドで単項目チェックエラーが発生しているかどうかを判定したい場合は、このように判定する
  - 相関チェックを必ず行う場合は、この判定処理は不要
- <5> jp.co.anas.atc.fw.core.validator.util.ValidatorUtilsでバリデーションエラーを登録する
  - ValidatorUtilsでエラー登録をすると、メッセージ内の{0}にBean Validationと同様、フォームプロパティのラベルが埋め込まれる
  - エラーメッセージパラメータが無い場合は単に省略すれば良い (ValidatorUtils.rejectValue(errors, "foo", "{エラーコード}"); という様に)
  - 詳細はValidatorUtils.rejectValueのJavaDocを参照のこと

なお、Bean Validationによる検証と同じ検証を行いたい場合は、jp.co.anas.atc.fw.core.validator.util.ValidatorUtilsのisValid～メソッドを使用すること。

FormValidatorの利用方法

基本的には次の様に利用する。

```
@Controller
@RequestMapping("foo/DBAP0010")
public class DBAP0010PController   {

    @Autowired
    private   DBAS0010P01PFormValidator   validato//   <1>

    @InitBinder
    public void   initBinder(WebDataBinder   webDataBinder)   {
                    webDataBinder.addValidators(//   <1>r);
        }

    @PostMapping("process")
    public String   process@Validated   DBAS0010P01PForm   form,   BindingResult   bindingRes//   <2>
        if   (bindingResult.hasErrors())   {
            return "DBAS0010P01P";
                    }

                //   ...

        return "redirect:DBAP0010/complete";
        }
}
```

- <1>
  FormValidatorを利用するにはControllerにインジェクションして、InitBinderアノテーションを付与したメソッドで、WebDataBinderに追加する
- <2> Bean Validationと同様に、Form引数にValidatedアノテーションを付与することでバリデーションを実行する

一つのControllerで複数のフォームを扱う場合は、Validatorの対象を限定するために、次の様に@InitBinder("xxx")でModelAttribute名を指定する必要がある。

```
@Controller
@RequestMapping("foo/DBAP0010")
public class DBAP0010PController   {

    @Autowired
    private   DBAS0010P01PProcess1FormValidator   validator1;

    @Autowired
    private   DBAS0010P01PProcess2FormValidator   validator2;
```

```
    @InitBinder("DBAS0010P01PProcess1Form ") //   <1>
    public void   initBinderForDBAS0010P01PProcess1Form   (WebDataBinder   webDataBinder)   {
                    webDataBinder.addValidators(validator1);
        }

    @InitBinder("DBAS0010P02PProcess2Form ") //   <1>
    public void   initBinderForDBAS0010P02PProcess2Form   (WebDataBinder   webDataBinder)   {
                    webDataBinder.addValidators(validator);
        }

    @PostMapping("process1")
    public String   process1(@Validated   DBAS0010P01PProcess1Form   form,   BindingResult   bindingResult)   {
        if   (bindingResult.hasErrors())   {
            return "DBAS0010P01P";
                    }

        return "redirect:DBAP0010/complete";
        }

    @PostMapping("process2")
    public String   process2(@Validated   DBAS0010P01PProcess2Form   form,   BindingResult   bindingRes//   <3>
        if   (bindingResult.hasErrors())   {
            return "DBAS0010P01P";
                    }

        return "redirect:DBAP0010/complete";
        }
}
```

- <1> @InitBinder("xxx")でModelAttribute名を指定する
  - 一つのControllerで複数の(バリデーションする)フォームを扱う場合のみ指定する

Bean Validationアノテーションの実装方法

Eclipseプロジェクト構成に記載の通り、jp.co.anas.atp.xyz.web.{division}.validation パッケージに
http://terasolunaorg.github.io/guideline/5.3.0.RELEASE/ja/ArchitectureInDetail/WebApplicationDetail/Validation.html#how-to-extend
の様に作成します。

次のクラスのソースコードを参考に作成してください。

- jp.co.anas.atc.fw.core.validator.constraints.Uppercase
- jp.co.anas.atc.fw.core.validator.constraints.LengthMax
- jp.co.anas.atc.fw.core.validator.constraints.Windows31jCharacters

バリデーションエラーメッセージの画面表示方法 (画面表示リクエストの場合)

バリデーションエラーメッセージを画面上部などに次の様なDOM構造で出力する場合は、

```
  <div class="asw-notice-massage">
    <ul>
        <li>メッセージ１</li>
        <li>メッセージ２</li>
    </ul>
  </div>
```

次の様にJSPで実装する。

```
<spring:hasBindErrors name="DBAS0010P01PForm"> <!--1-->
  <atc:sortErrors var="sortedErrorList" errors="${errors}"> <!--2-->
    <atc:path value="foo" />
    <atc:path value="bar" />
```

```
    <atc:path value="sub.hoge" />
    <atc:path value="sub.huga" />
  </atc:sortErrors>
  <div class="asw-notice-massage">
    <ul>
      <c:forEach var="error" items="${sortedErrorList}">
        <li><spring:message message="${error}" htmlEscape="true" /></li>
      </c:forEach>
    </ul>
  </div>
</spring:hasBindErrors>
```

- <1> name属性にはバリデーション対象フォームのModelAttribute名を指定する
  (form:formタグのmodelAttribute属性に指定している値と同じ)
    - spring:hasBindErrorsタグは指定のフォームでバリデーションエラーが発生している場合にのみ、タグのボディを評価し、タグのボディの中でのみ有効なerrors変数にErrorsオブジェクトを設定する
- <2> atc:sortErrorsタグでエラーメッセージの表示順序を指定する
    - 子要素のatc:pathタグでフォームプロパティパスを指定することで、表示順序を指定する
    - 指定されてないフォームプロパティに対するバリデーションエラーメッセージは、末尾に順不同で表示されることになる
    - この例では、次の順序でバリデーションエラーメッセージが表示されることになる
        1. DBAS0010P01PFormフォームに対するバリデーションエラーメッセージ
            - Errors#reject で登録できるグローバルエラー(オブジェクトエラー)のメッセージ
            - 今のところ、グローバルエラーを登録するケースは想定していないので気にしなくて良い
        2. DBAS0010P01PFormフォームのfooプロパティに対するバリデーションエラーメッセージ
        3. DBAS0010P01PFormフォームのbarプロパティに対するバリデーションエラーメッセージ
        4. DBAS0010P01PFormフォームのsubプロパティに設定されているオブジェクトのhogeプロパティに対するバリデーションエラーメッセージ
        5. DBAS0010P01PFormフォームのsubプロパティに設定されているオブジェクトのhugaプロパティに対するバリデーションエラーメッセージ
        6. DBAS0010P01PFormフォームの他のプロパティに対するバリデーションエラーメッセージ (順不同)
    - なお、指定フォームプロパティに対して複数のバリデーションエラーがある場合、そのエラーメッセージの表示順は順不同

メッセージ本体でHTMLタグを使用したい場合のJSP実装例

次の様にメッセージパラメータのみHTMLエスケープすることで、
メッセージプロパティでHTMLタグを記述しておき画面表示に利用できる。

```
<spring:hasBindErrors name="DBAS0010P01PForm">
  <atc:sortErrors var="sortedErrorList" errors="${errors}">
    <%--  省略  -->%
  </atc:sortErrors>
  <div class="asw-notice-massage">
    <ul>
      <c:forEach var="error" items="${sortedErrorList}">
        <%--  メッセージにHTMLタグを含められるようにメッセージパラメータのみHTMLエスケープする  -->%
        <li><spring:message message="${atcf:hMsgSrcResolvable(error)}" /></li> <!--1-->
      </c:forEach>
    </ul>
  </div>
</spring:hasBindErrors>
```

- <1> EL関数 atcf:hMsgSrcResolvable を使うことでメッセージパラメータのみHTMLエスケープできる

バリデーションエラーメッセージの画面表示方法 (Ajaxリクエストの場合)

Ajaxエラーハンドリング を参照のこと

バリデーションエラーメッセージ定義

方針

- 「Bean Validationアノテーションのデフォルトメッセージキーを用いたメッセージ定義」を基本とし、

- フォームプロパティ個別にメッセージを定義する必要がある場合は「フォームプロパティ個別のメッセージキーを用いたメッセージ定義」を使用する
- FormValidatorで登録するエラーについては「プロジェクト標準のエラーメッセージキーを用いたメッセージ定義(FormValidator用)」を使用する
- 型変換エラーについては「型変換エラーのメッセージキーを用いたメッセージ定義」を使用する
- なお、メッセージに含めるフォームプロパティのラベルに関しては「フォームプロパティのラベル定義」を使用して、各メッセージ内の {0} に埋め込む形を基本とする

この方針に従ったメッセージ定義は次の様な形になる。

ValidationMessages.propertiesの例

```
#　汎用型変換エラーメッセージ
typeMismatch=入力形式が不正です。

#　型変換エラーメッセージ
typeMismatch.java.lang.Integer={0}は整数で入力してください。
typeMismatch.java.lang.Long={0}は整数で入力してください。

#　Bean　Validation　(JSR-303)
javax.validation.constraints.Max.message={0}は{value}以下の値を入力してください。
javax.validation.constraints.Min.message={0}は{value}以上の値を入力してください。
javax.validation.constraints.NotNull.message={0}の値が未入力です。

#　Hibernate　Validator
org.hibernate.validator.constraints.Range.message={0}は{min}から{max}の間の値を入力してください。

#　DukeNavire　自製の入力チェック用アノテーションのエラーメッセージ
dukenavire.validation.constraints.DNValidFullWidthHiragana.message={0}はひらがなで入力してください。

#　フォームプロパティ個別のメッセージ定義
Length.loginForm.userId　=　{0}は{2}文字以上、{1}文字以下で入力してください！！

#　フォームプロパティのラベル定義
userId　=　ユーザID
loginForm.userId　=　ログインユーザID
```

Bean Validationアノテーションのデフォルトメッセージキーを用いたメッセージ定義

各アノテーションのデフォルトメッセージキーは上記の「Bean Validationアノテーション一覧」を参照のこと。

メッセージ定義例

```
org.hibernate.validator.constraints.Range.message={0}は{min}から{max}の間の値を入力してください。
```

- {0}にはバリデーションエラーとなったフォームプロパティのラベルが埋め込まれる
- {min}, {max} にはBean Validationアノテーションの該当の属性値が埋め込まれる

フォームプロパティ個別のメッセージキーを用いたメッセージ定義

メッセージキーの形式は次の通り。

アノテーション名.フォーム属性名.プロパティ名

メッセージの解決ではBean Validationアノテーションのデフォルトメッセージキーを用いたメッセージ定義より、こちらの定義が優先される。

メッセージ定義例

```
Length.loginForm.userId　=　{0}は{2}文字以上、{1}文字以下で入力してください！！
```

- {0}にはバリデーションエラーとなったフォームプロパティのラベルが埋め込まれる
- {1}～{N}にはアノテーションの属性値が埋め込まれる
    - インデックスはアノテーションの属性名のアルファベット順(昇順)におけるインデックス
    - 例のLengthアノテーションの場合に埋め込まれる値は次の通り
        - {0} ・・・ フォームプロパティのラベル
        - {1} ・・・ Lengthアノテーションのmax属性値
        - {2} ・・・ Lengthアノテーションのmin属性値

メッセージキーの詳細仕様

次の形式のメッセージキーを導出し、上から順にメッセージ解決を試行する。
また、プロパティがリストの場合は添え字無しのメッセージキーも導出する。

1. 「アノテーション名.フォーム属性名.プロパティ名」
2. 「アノテーション名.プロパティ名」
3. 「アノテーション名.末端のプロパティ名」(プロパティが構造体の場合)

例えば、 sampleForm.f1.list2[0].f3 というフォームプロパティがNotNull制約に違反した場合のメッセージキーは次の通り。
NotNull.list2[0].f3 というメッセージキーは導出されないことに注意。

優先順位	メッセージキー	メッセージキー形式
1	NotNull.sampleForm.f1.list2[0].f3	「アノテーション名.フォーム属性名.プロパティ名」
2	NotNull.sampleForm.f1.list2.f3	「アノテーション名.フォーム属性名.プロパティ名」(添え字無し)
3	NotNull.f1.list2[0].f3	「アノテーション名.プロパティ名」
4	NotNull.f1.list2.f3	「アノテーション名.プロパティ名」(添え字無し)
5	NotNull.f3	「アノテーション名.末端のプロパティ名」

以下、上記のコード例におけるDBAS0010P01PFormを例に、メッセージキーの例を示す。
なお、DBAS0010P01PFormのmodelAttribute名はDBAS0010P01PFormとする。

- DBAS0010P01PForm.sub.hoge が制約違反した場合のメッセージキー
    1. NotNull.DBAS0010P01PForm.sub.hoge
    2. NotNull.sub.hoge
    3. NotNull.hoge
- DBAS0010P01PForm.list[0].hoge が制約違反した場合のメッセージキー
    1. NotNull.DBAS0010P01PForm.list[0].hoge
    2. NotNull.DBAS0010P01PForm.list.hoge
    3. NotNull.list[0].hoge
    4. NotNull.list.hoge
    5. NotNull.hoge

プロジェクト標準のエラーメッセージキーを用いたメッセージ定義 (FormValidator用)

- メッセージキーの形式
    - プロジェクト標準のエラーメッセージキーと同じ (e.g. SERR0001)
- メッセージ定義の形式
    - 「フォームフィールド個別のメッセージキーを用いたメッセージ定義」と同じ

メッセージ定義例

```
SERR0001  =  {0}は{2}文字以上、{1}文字以下で入力してください！！
```

- {0}にはバリデーションエラーとなったフォームプロパティのラベルが埋め込まれる
- {1}～{N}にはValidatorUtils.rejectValue(...)の引数に渡されるエラーメッセージパラメータが埋め込まれる

型変換エラーのメッセージキーを用いたメッセージ定義

型変換エラーメッセージはSpringがリクエストパラメータをFormクラスのフィールドにマッピング出来なかった際のエラーメッセージ。
型変換エラーは例えば、FormクラスのIntegerフィールドに対して、"a"というようなリクエストパラメータをマッピングしようとした場合に発生する。

メッセージキーの形式は次の通り。

- 「typeMismatch」
  - 型ミスマッチエラーのデフォルトメッセージ用
  - 必ず定義しておく
  - 不正アクセスしない限り、画面には表示されないメッセージである想定
  - java.lang.Booleanなどへの変換エラーで使用
- 「typeMismatch.対象のFQCN」
  - 特定の型ミスマッチエラーのデフォルトメッセージ
- 「typeMismatch.フォーム属性名.プロパティ名」
  - 特定のフォームのフィールドに対する型ミスマッチエラーのメッセージ
  - 個別にメッセージ定義したい場合のみ使用する

メッセージ定義例

```
#　汎用型変換エラーメッセージ
#　java.lang.Booleanなどへの型変換エラーなどはこのメッセージを使う
typeMismatch=入力形式が不正です。

#　型変換エラーメッセージ
typeMismatch.java.lang.Integer={0}は整数で入力してください。
typeMismatch.java.lang.Long={0}は整数で入力してください。
```

- {0}にはバリデーションエラーとなったフォームプロパティのラベルが埋め込まれる

フォームプロパティのラベル定義

メッセージキーの形式は次の通り。

- 「プロパティ名」
  - 基本的にはこの形式を使用する
- 「フォーム属性名.プロパティ名」
  - フォームによってラベルを出しわけたい場合はこの形式を使用する
  - 同じプロパティ名に対して両方の形式で定義されている場合は、こちらの定義が優先される

フォームプロパティのラベル定義例

```
userId　=　ユーザID
loginForm.userId　=　ログインユーザID
```

フォーム属性名とはModelAttribute名を指す。
ModelAttribute名は以下の通り。

```
@Controller
@RequestMapping("foo/DBAP0010")
public class DBAP0010PController　{

    @PostMapping("process1")
    public String　process1(@Validated　DBAS0010P01PForm　form,　BindingResult　bindingResu//　<1>
        if　(bindingResult.hasErrors())　{
            return "DBAS0010P01P";
                    }

        return "redirect:DBAP0010/complete";
```

```
        }

    @PostMapping("process2")
    public String process2(@Validated @ModelAttribute("sampleForm"
) DBAS0010P01PForm form, BindingResult bindingRes// <2>
        if (bindingResult.hasErrors()) {
            return "DBAS0010P01P";
                    }

        return "redirect:DBAP0010/complete";
        }

}
```

- <1> ModelAttributeアノテーションを付与しない場合はフォームクラス名からModelAttribute名が導出される
  - この例の場合はModelAttribute名はDBAS0010P01PFormになる
- <2> ModelAttributeアノテーションを付与し、ModelAttribute名を指定することも可能
  - この例の場合はModelAttribute名はsampleFormになる

ラベルメッセージキーの詳細仕様

次の形式のメッセージキーを導出し、上から順にメッセージ解決を試行する。
また、プロパティがリストの場合は添え字無しのメッセージキーも導出する。
なお、このメッセージキーの導出仕様は「フォームプロパティ個別のメッセージキーを用いたメッセージ定義」と同様の仕様になっている。

1. 「フォーム属性名.プロパティ名」
2. 「プロパティ名」
3. 「末端のプロパティ名」(プロパティが構造体の場合)

例えば、 sampleForm.f1.list2[0].f3 というフォームプロパティが制約違反した場合のラベルメッセージキーは次の通り。
list2[0].f3 というメッセージキーは導出されないことに注意。

優先順位	メッセージキー	メッセージキー形式
1	sampleForm.f1.list2[0].f3	「フォーム属性名.プロパティ名」
2	sampleForm.f1.list2.f3	「フォーム属性名.プロパティ名」(添え字無し)
3	f1.list2[0].f3	「プロパティ名」
4	f1.list2.f3	「プロパティ名」(添え字無し)
5	f3	「末端のプロパティ名」

以下、上記のコード例におけるDBAS0010P01PFormを例に、プロパティのラベルメッセージキーの例を示す。
なお、DBAS0010P01PFormのmodelAttribute名はDBAS0010P01PFormとする。

- DBAS0010P01PForm.sub.hoge が制約違反した場合のラベルメッセージキー
  1. DBAS0010P01PForm.sub.hoge
  2. sub.hoge
  3. hoge
- DBAS0010P01PForm.list[0].hoge が制約違反した場合のラベルメッセージキー
  1. DBAS0010P01PForm.list[0].hoge
  2. DBAS0010P01PForm.list.hoge
  3. list[0].hoge
  4. list.hoge
  5. hoge

参考

- 4.1. 入力チェック — TERASOLUNA Server Framework for Java (5.x) Development Guideline 5.3.0.RELEASE documentation