BeanBinder リファレンス
version 0.2.3

Author:Tomohiro Umeda

目次

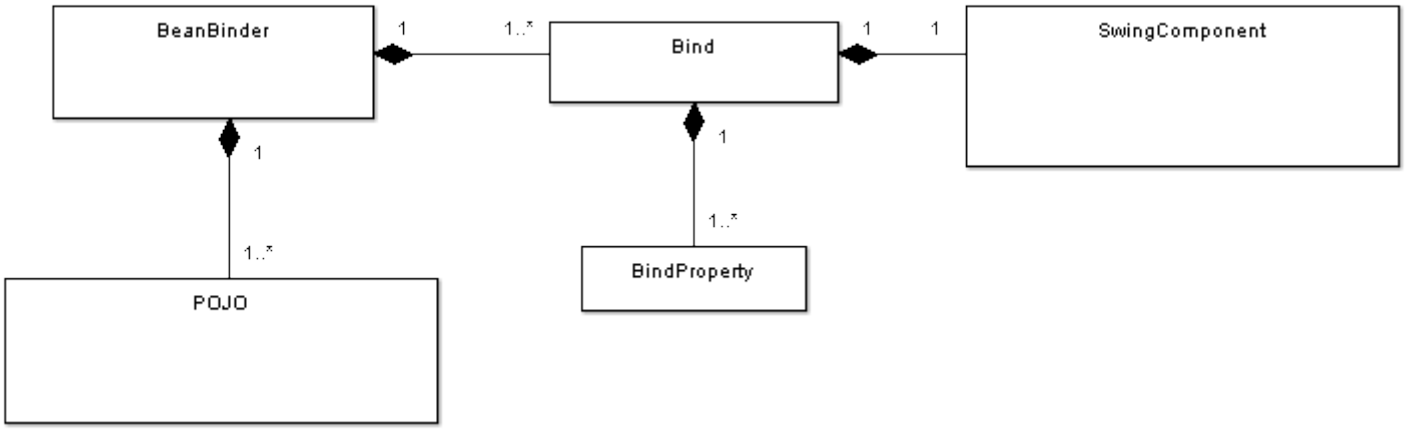
1. BeanBinder の概要
BeanBinder はオブジェクトと Swing コンポーネントとの関連付けを責務とし、Swing 開発における頻出する問題を解決する事が目的となります。 プレゼンテーション層とビジネスモデル、ロジックの疎結合を促し保守性と開発効率を向上させる事が目標となります。

オブジェクトと Swing コンポーネント間との協調（コンポーネントの再描画、オブジェクトへの反映など）、Validation（値の検証）処理の枠組みの提供やコンポーネント毎のIMEの切替といった日本固有の問題も解決します。

システム要件と依存関係
JDK1.4.2～以降
commons-logging.jar
commons-beanutils.jar
cglib-nodep.jar
※1.5.0 以降では別途 beanbinder-annotations.jar を利用する事で、Annotation によるマッピングも対応

サンプルアプリケーション SimplePetStore

主要なオブジェクト



■BeanBinder

オブジェクトとSwingコンポーネントとの協調を責務とします。
全ての操作はBeanBinderを経由して行いますので最も主要なオブジェクトとなります。

■BeanBinderFactory

BeanBinderを生成する責務を持ちます。

■Bind

オブジェクトとSwingコンポーネントとのマッピング情報の表現を責務とします。

■BindProperty

どのプロパティとコンポーネントが関係を持つのかを定義します。
また、IMEの設定や左寄・右寄・中央などの位置あわせ等の視覚的な情報も持ちます。
通常BindPropertyは1コンポーネント毎に定義する必要があり、JTableのような表形式のコンポーネントの場合は1カラム毎に1つのBindPropertyを必要とします。

■BindFactory

コンポーネントの種類に応じたBindオブジェクトを作成します。
マッピング情報作成の多くはこのオブジェクトを利用する事で解決します。

■XMLParser

BindFactoryに代わってオブジェクト/コンポーネントをXMLを用いてマッピングする責務を持つ。

■AnnotationParser

BindFactoryに代わってオブジェクト/コンポーネントをアノテーションを用いてマッピングする責務を持つ。
別途beanbinder-annotations.jarをクラスパスに追加する事で利用できます。

■オブジェクト(POJO)

オブジェクトはJavaBeansである必要は必ずしも無く、不要であればアクセッサ(Getters/Setters)なしにフィールドでのアクセスが可能となっています。

2. 基本編

Example　オブジェクトの準備

顧客区分オブジェクト

```
public class Category{
     private Long id;
     private String name;

     public Long getId() { return this.id }
     public void setId(Long id) { this.id = id; }
     public String getName() { return this.name }
     public void setName(String name) { this.name = name; }
}
```

顧客オブジェクト

```
public class Customer {
     private Long id;
     private String name;
     private int age;
     private Category category;

     public Long getId() { return this.id }
     public void setId(Long id) { this.id = id; }
     public String getName() { return this.name }
     public void setName(String name) { this.name = name; }
     public int getAge() { return this.age; }
     public void setAge(int age) { this.age = age; }
     public Category getCategory() { return this.category; }
     public void setCategory(Category category) {
          this.category = category;
     }
}
```

Example.1 テキストボックス

BeanBinder のインスタンスを作成、JFrame や JPanel 等の主要な画面一ずつに用意する事を推奨します。

```
BeanBinder binder = BeanBinderFactory.createSwingBinder();
```

オブジェクトをキー名″cst″で BeanBinder に登録

```
Customer c = new Customer();
c.setName("Yamada Tarou");
binder.setObject("cst", c);
```

コンポーネント(JTextField)とオブジェクトの関連付け

```
JTextField nameTxt = new JTextField();
Bind bind = BindFactory.createBind( "cst", nameTxt, "name" );
binder.bind( "nameBind", bind );

JTextField ageTxt = new JTextField();
bind = BindFactory.createBind( "cst", ageTxt, "age" );
binder.bind( "ageBind", bind );
```

″cst″(顧客オブジェクト)と nameTxt(JTextField)との関連付け
第 1 引数:バインド対象のオブジェクトを示すキー名
第 2 引数:バインド対象のコンポーネントを示すキー名
第 3 引数:表示内容と編集内容を示すプロパティ名

```
Bind bind = BindFactory.createBind( "cst", nameTxt, "name" );
```

BeanBinder にバインド情報を登録する
第一引数:バインド情報を登録する為のキー名、第二引数はバインド情報

```
binder.bind( "nameBind", bind );
```

コンポーネントからオブジェクトを更新する

```
binder.updateObject( "cust" );
```

オブジェクトからコンポーネントを更新するには

```
binder.updateComponent( "nameBind" );//コンポーネントを明示しても良い
binder.updateComponent( ".*" );//全コンポーネントの更新、このように正規表現も可能。
```

Example.2 コンボボックス

顧客区分リストをデータソースとして登録

```
List<Category> list = new ArrayList<Category>();
Category category = new Category();
category.setId(new Long(1));
category.setName("一般顧客");
list.add( category );

category = new Category();
category.setId(new Long(2));
category.setName("会員顧客");
list.add( category );

binder.setObject( "categoryList", list );
```

編集対象となる顧客を登録

```
Customer c = new Customer();
binder.setObject( "target", customer );
```

顧客選択用のコンボボックスとデータソース、変更後のターゲットとの関連付け

```
JComboBox combo = new JComboBox();

Bind bind = BindFactory.createBindForCombo(
     "target", combo, "category", "category.id",
     "categoryList", "name", null, "id"
);
binder.bind( "customerBind", bind );
```

第1引数:バインド対象のオブジェクトを示すキー名
第2引数:バインド対象のコンポーネント
第3引数:選択したオブジェクトの反映先を示すプロパティ名(この場合"target"のプロパティを指す)
第4引数:反映先オブジェクトのアイデンティティを示すプロパティ名
第5引数:データソースとなるオブジェクトを示すキー名
第6引数:表示内容を示すプロパティ名(この場合データソースのプロパティ名を指す)
第7引数:選択した際に取得する値を示すプロパティ名(この場合データソースのプロパティ名を指す)
第8引数:データーソースのアイデンティティを示すキー名

尚、データソースとなりえるのはコレクションオブジェクトのみである。

```
Bind bind = BindFactory.createBindForCombo(
     "target", combo, "category", "category.id",
     "categoryList", "name", null, "id"
);
```

コンポーネントからオブジェクトを更新するには

```
binder.updateObject( "target" );
```

変更後のオブジェクトを取得するには

```
Customer c = (Customer)binder.getObject( "target" );
```

コンボボックスから選択したオブジェクトを取得するには

```
Category cat = (Category)binder.getSelectedValue( "customerBind" );
```

Example.3 テーブルコンポーネント

顧客の氏名・年齢を編集する2列の表を作成する。

```
List<Customer> list = new ArrayList<Customer>();
Customer c = new Customer();
c.setName("Yamada");
c.setAge(30);
list.add(c);
binder.setObject( "cstList", list );

BindProperty[] columns = new BindProperty[2];
columns[0] = BindFactory.createColumn( "氏名", "name", false );
columns[1] = BindFactory.createColumn( "年齢", "age", false );

JTable tableBox = new JTable();
Bind bind = BindFactory.createBindForTable("cstList", tableBox, columns);
binder.bind("helloBind", bind);
```

第1引数:列の見出しを表すテキスト
第2引数:表示や編集内容の反映先などを示すプロパティ名
第3引数:この列が読込み専用か否か

```
columns[0] = BindFactory.createColumn("年齢", "age", false );
```

3. セルエディタ & セルレンダラ

ここでは JTable のセルエディタ・セルレンダラのカスタマイズ方法を示す。
通常はテキスト入力のみであるが、コンボボックスやチェックボックス等の特殊な入力を行う場合に
必要なものである。
また、カスタムコンポーネントの章をマスターすればオリジナルのコンポーネントをセルエディタに適用する
事も可能である。

3. 1 テーブルコンポーネント – セルエディタ

顧客編集表の顧客区分列をコンボボックスを用いて選択する。

```
List<Customer> list = new ArrayList<Customer>();
Customer c = new Customer();
c.setName("Yamada"); c.setAge(30);
list.add(c);
binder.setObject( "cstList", list );

List<Category> categories = new ArrayList<Category>();
Category cat = new Category();
cat.setName( "一般顧客" );
categories.add( cat );

cat = new Category();
cat.setName( "会員顧客" );
categories.add( cat );
binder.setObject( "catList", categories );

JComboBox combo = new JComboBox();
Bind bind = BindFactory.createBindForComboCellEditor( combo,
                                          "catList","name", null );
binder.bind( "categoryCombo", bind );

CellEditor cellEditor = new GeneralCellEditor(binder, combo) {
     public Object getCellEditorValue(BeanBinder binder) {
          return binder.getSelectedValue("categoryCombo");
     }
     public void updateCellEditor( BeanBinder binder, Object value,
                                          int row, int col) {
     }
};
```

（続く～）

（～続き）

```
BindProperty[] columns = new BindProperty[3];
columns[0] = BindFactory.createColumn( "氏名", "name", false );
columns[1] = BindFactory.createColumn( "年齢", "age", false );
columns[2] = BindFactory.createColumn(
                    "区分", "category.name", "category", cellEditor );

JTable tableBox = new JTable();
Bind bind = BindFactory.createBindForTable("cstList", tableBox, columns);
binder.bind("helloBind", bind);
```

セルエディタとなるコンボボックスのバインドを作成
第 1 引数はコンポーネント
第 2 引数はデータソースを示すキー名
第 3 引数は表示内容を示すプロパティ名
第 4 引数は選択した時に取得する値を示すプロパティ名

```
JComboBox combo = new JComboBox();
Bind bind = BindFactory.createBindForComboCellEditor(
                                        combo,"catList","name",null );
binder.bind( "categoryCombo", bind );
```

セルエディタの作成をするには GeneralCellEditor を実装する。
getCellEditorValue メソッドは、セルエディタによって変更した値を戻す必要がある。
updateCellEditor メソッドは、セル内のオブジェクトから
セルエディタの内容を変更する必要がある場合のみ実装する。（以下の例では何もしていない）

```
CellEditor cellEditor = new GeneralCellEditor(binder, combo) {
     public Object getCellEditorValue(BeanBinder binder) {
          return binder.getSelectedValue("categoryCombo");
     }
     public void updateCellEditor( BeanBinder binder, Object value,
                                             int row, int col) {
     }
};
```

次にカラムのマッピングを行うには BindFactory の createColumn メソッドを利用する。
第 1 引数:列の見出し
第 2 引数:セルの表示内容を示すプロパティ名
第 3 引数:編集内容の反映先を示すプロパティ名
第 4 引数:セルエディタを指定

```
columns[2] = BindFactory.createColumn(
                    "区分", "category.name", "category", cellEditor );
```

3. 2 テーブルコンポーネント - セルレンダラ

Person オブジェクトの性別を boolean で表現する事とした。(boolean sex)
この時、性別の値が true ならば男性、false なら女性と表示する場合。

```
JLabel lbl = new JLabel();
Bind lblBind = BindFactory.createBind( "sexValue", lbl, "" );
binder.bind( "lbl", lblBind );

CellRenderer cellRenderer = new CellRenderer(binder, lbl){
    public void update(BeanBinder binder, BindProperty property,
          Object displayValue, Object value, int row, int column)
     {
          binder.setObject("sexValue", (value.equals(true)?"男性":"女性"));
          binder.updateComponent("lbl");
     }
};

BindProperty[] columns = new BindProperty[2];
columns[0] = BindFactory.createColumn( "氏名", "name", false );
columns[1] = BindFactory.createColumn( "性別", "sex", false );
columns[1].getColumnProperty().setCellRenderer(cellRenderer);

JTable tableBox = new JTable();
Bind bind = BindFactory.createBindForTable("persons", tableBox, columns);
binder.bind("helloBind", bind);
```

4. Validation

入力が 0～999999 の間が許容範囲である場合の Validation 処理

```
JLabel label = new JLabel();
ValidateDisplay display = new SwingValidateDisplay( label );
Validator validator = new RangeValidator( 0, 999999,
                                    "入力は 0～999999 の範囲です。", display );
JTextField txt = new JTextField();
Bind bind = BindFactory.createBind(
                    "product", txt, "unitPrice", validator );
binder.bind( "helloBind", bind );
```

ValidateDisplay は検証結果の表示先を表現します。
SwingValidateDisplay のコンストラクタに渡せるコンポーネントとしては
JLabel、JTextComponent、Window 等を指定出来ます。

```
ValidateDisplay display = new SwingValidateDisplay( label );
```

5. IME

```
JTextField txt = new JTextField();
Bind bind = BindFactory.createBindForText(
                    "product", txt, "name", IMEMode.JP, null );
binder.bind( "helloBind", bind );
```

他にも IMEMode.DEFAULT、IMEMode.HALFKANA などが指定出来る。

6. XML

プログラムコードからマッピングする方法を示したが、ここからは XML ファイルを用いてマッピングする方法を解説する。 必要なものは DOM パーサーであるが、JDK1.4.2 以降であれば付属の実装で問題ない。
基本的に、一画面（Window や JPanel など）辺り、一つの XML によってマッピングされるべきである。
なお、後述するアノテーションと XML のマッピング方法はそれぞれ互換性がある。

マッピング対象例

```
public class HelloWindow extends JFrame {
     private BeanBinder binder = BeanBinderFactory.createSwingBinder();
     private JTextField nameTextBox = new JTextField();

     public HelloWindow() {
          Customer c = new Customer();
          c.setName("Yamada Tarou");
          this.binder.setObject( "customer", c );
          new XMLParser(this.binder, this).parse();
     }
}
```

マッピング例

```
<?xml version="1.0" encoding="utf-8" ?>

<bind>
     <basic field="nameTextBox" objectKey="customer" prop="name" />

</bind>
```

6.1. <bind>～</bind>

必ずこのタグ内にマッピング情報を記述する。

6.2. <basic>～</basic>

コンポーネントとオブジェクトの基本的なマッピングを定義する。 指定可能な要素は次の通りである。
field：バインド対象のコンポーネントを示すフィールド名。
objectKey：バインド対象のオブジェクトを示すキー名。
prop：表示内容または編集内容の反映先を示すプロパティ名。
group：コンポーネントをグループ化する際に必要。
imeMode：IME の状態を設定。IMEMode.DEFAULT、IMEMode.JP、IMEMode.HALFKANA のいずれか。
formatField：java.text.Format の実装を示すフィールド名。
horizon：水平方向のレイアウト。AlignType.LEFT、AlignType.CENTER、AlignType.RIGHT のいずれか。
validatorField：Validator の実装を示すフィールド名。

尚、次の2つの XML は同じ意味になる。

```
<?xml version="1.0" encoding="utf-8" ?>

<bind>
     <basic field="nameTextBox" objectKey="customer" prop="name" />
</bind>
```

2通りの記述が出来る。

```
<?xml version="1.0" encoding="utf-8" ?>

<bind>
     <basic>
          <field>nameTextBox</field>
          <objectKey>customer</objectKey>
          <prop>name</prop>
     </basic>
</bind>
```

6.3. <select>～</select>

コンボボックスやリストボックス等の選択を伴うコンポーネントに対して使用する。
次の例は従業員一覧を表すコンボボックスの例である。

```
<?xml version="1.0" encoding="utf-8" ?>

<bind>
     <select>
          <field>empComboBox</field>

          <destKey>target</destKey>
          <destValueProp>employee</destValueProp>
          <destIdProp>employee.id</destIdProp>

          <srcKey>empList</srcKey>
          <srcDisplayProp>name</srcDisplayProp>
          <srcIdProp>id</srcIdProp>
     </select>
</bind>
```

この例の場合、受注の担当者を決定するコンボボックス等に使える。
dest は編集結果の反映先を示し、src はコンポーネントに表示するデータソースを示す。

<select>タグに指定可能な要素は以下の通りである。
field：バインド対象のコンポーネントを示すフィールド名
destKey：選択した値の反映先オブジェクトを示すキー名
destValueProp：選択した値の反映先を示すプロパティ名
destIdProp：編集先のアイデンティティを示すプロパティ名
srcKey：データーソースを示すキー名
srcDisplayProp：表示する内容を示すプロパティ名（この場合データソースのプロパティを示す）
srcValueProp：選択した時に取得する値を示すプロパティ名
srcIdProp：アイデンティティを示すプロパティ名（この場合データソースのプロパティを示す）

6.4. <table>～</table> & <column>～</column>

表形式のコンポーネントに対して適用する。

```
<?xml version="1.0" encoding="utf-8" ?>

<bind>
     <table field="customerTable" objectKey="customerList">
          <column prop="name" text="氏名" imeMode="IMEMode.JP" />
          <column prop="age" text="年齢" />
     </table>
</bind>
```

上記の例では、顧客一覧表を定義している。 氏名列、年齢列の計2列の表である。

<table>タグの指定可能な要素は以下の通りである。
field：JTable 等の表形式のコンポーネントを示すフィールド名
objectKey：表示内容、または編集先のオブジェクトを示すキー名
group：コンポーネントをグループ化する際に必要
height：行の高さを指定。
sortable：ソートをするかしないか、true または false で指定

<column>タグの指定可能な要素は以下の通りである。
text：列のタイトル
prop：表示内容、または編集先を示すプロパティ名
readOnly：読み取り専用か否か、true または false で指定
width：列の横幅を指定
displayProp：表示内容を示すプロパティ名
valueProp：編集先、または取得する値を示すプロパティ名
idProp：アイデンティティを示すプロパティ名

imeMode：IME モードを指定、IMEMode.DEFAULT、IMEMode.JP、IMEMode.HALFKANA のいずれか
formatField：java.text.Format の実装を示すフィールド名
horizon：水平位置のレイアウト、AlignType.LEFT、AlignType.CENTER、AlignType.RIGHT のいずれか
validatorField：Validator の実装を示すフィールド名
cellEditorField：CellEditor の実装を示すフィールド名
cellRendererField：CellRenderer の実装を示すフィールド名

7. アノテーション

JDK1.5.0 以降のアノテーションを使ったオブジェクト/コンポーネントのマッピングは
beanbinder-annotations.jar を用いることにより可能となる。

7.1. @Basic

最もシンプルなマッピングを提供する。

```
public class HelloWindow extends JFrame {
     private BeanBinder binder = BeanBinderFactory.createSwingBinder();

     @Basic( objectKey="customer", prop="name" )
     private JTextField nameText = new JTextField();

     public HelloWindow() {
          Customer c = new Customer();
          c.setName("Yamada Tarou");
          this.binder.setObject( "customer", c );
          new AnnotationParser(this.binder, this).parse();
     }
}
```

objectKey はバインド対象となるオブジェクトを示すキー名を、prop には表示内容や編集先を示すプロパティ名をそれぞれ指定する。 またアノテーションはフィールドのみに付加させる事が出来る。

```
@Basic( objectKey="customer", prop="name" )
private JTextField nameText = new JTextField();
```

AnnotationParser は指定されたオブジェクトからアノテーションを取得し、BeanBinder に適切な設定を行う責務を持つ。 第 1 引数は設定先の BeanBinder を指定、第 2 引数はアノテーションが記述されているオブジェクトを指定する。

```
new AnnotationParser(this.binder, this).parse();
```

また、次のようにIMEの設定を定義する事も可能である。

```
@Basic( objectKey="customer", prop="name", imeMode=IMEMode.JP )
private JTextField nameText = new JTextField();
```

次のマッピングで右寄せで表示される。

```
@Basic( objectKey="customer", prop="name", horizon=AlignType.RIGHT )
private JTextField nameText = new JTextField();
```

7.2. @Select

コンボボックスやリストボックスなどのマッピングを表す。
destKey：選択した値の反映先オブジェクトを示すキー名
destValueProp：選択した値の反映先を示すプロパティ名
destIdProp：編集先のアイデンティティを示すプロパティ名
srcKey：データーソースを示すキー名
srcDisplayProp：表示する内容を示すプロパティ名（この場合データソースのプロパティを示す）
srcValueProp：選択した時に取得する値を示すプロパティ名
srcIdProp：アイデンティティを示すプロパティ名（この場合データソースのプロパティを示す）

```
@Select(  destKey="target",
          destValueProp="employee", destIdProp="employee.id",
          srcKey="empList", srcDisplayProp="name",
          srcValueProp="", srcIdProp="id"
)
private JComboBox cmb = new JComboBox();
```

7.3. @Table、@Column

@Table・・・表形式のコンポーネントとのマッピングを表現します。
objectKey：表示内容または編集先を示すキー名を指定します。
columns：@Column の配列。 列を定義します。

@Column・・・列単位でのマッピングを表現します。
prop：表示内容または編集先を示すプロパティ名。
text：列の見出しを定義。

```
@Table( objectKey="customerList",
     columns={
          @Column( prop="name", text="氏名" ),
          @Column( prop="age", text="年齢" )
     }
)
private JTable tbl = new JTable();
```

以下の記述は上記のサンプルと全く同じ動作をします。
displayProp は表示内容を示すプロパティ名を、valueProp は編集先を示すプロパティ名を指定します。

```
@Table( objectKey="customerList",
     columns={
          @Column( displayProp="name", valueProp="name", text="氏名" ),
          @Column( displayProp="age", valueProp="age", text="年齢" )
     }
)
private JTable tbl = new JTable();
```

7.4. @Table セルエディタの使用

```
@Table( objectKey="customerList",
     columns={
          @Column( prop="name", text="氏名" ),
          @Column( prop="age", text="年齢" ),
          @Column( displayProp="category.name",
                    valueProp="category", text="顧客区分",
                    cellEditorField="editor1"
          )
     }
)
private JTable tbl = new JTable();

@Select( srcKey="categoryList", srcDisplayProp="name" )
private JComboBox combo = new JComboBox();

private CellEditor cellEditor1 = new GeneralCellEditor(binder, combo) {
     public Object getCellEditorValue(BeanBinder binder) {
          return binder.getSelectedValue("combo");
     }
     public void updateCellEditor( BeanBinder binder, Object value,
                                             int row, int col) {}
};
```

顧客区分を表からコンボボックスを用いて編集出来るようにする。
表示内容は category.name、編集先は category プロパティとする。
cellEditorField は CellEditor のフィールド名を指定します。

```
@Column( displayProp="category.name",
     valueProp="category", text="顧客区分",
     cellEditorField="editor1"
)
```

7.5. アノテーションによる Validation

顧客年齢入力欄を 0～100 である事を検証するプログラム。

```
@Basic( objectKey="customer", prop="age", validatorField="v1" )
private JTextField nameText = new JTextField();

private JLabel errMsgLabel = new JLabel();
private ValidateDisplay disp = new SwingValidateDisplay(errMsgLabel);
private Validator v1 = new RangeValidator(0,100,"0～100 までです", disp);
```

ValidateDisplay は検証処理結果の表示先。
SwingValidateDisplay 実装のコンストラクタには表示先コンポーネントを渡します。

```
private ValidateDisplay disp = new SwingValidateDisplay(errMsgLabel);
```

数値入力であるかを検証する場合は NumberValidator を利用します。
様々な Validator が用意されていますが自作の Validator を用いる事も出来ます。

```
private Validator v1 = new NumberValidator("数値を入力して下さい。", disp);
```

複数の Validator を組み合わせるには ValidateContainer オブジェクトを利用します。

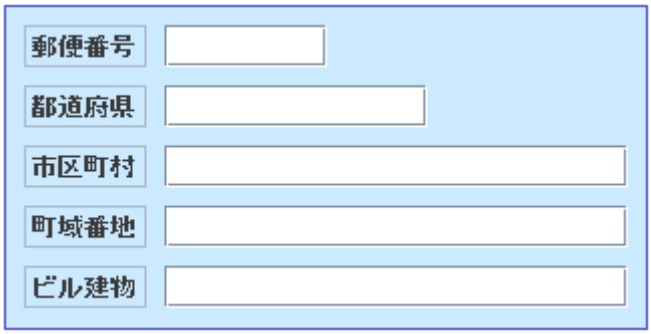
```
private Validator v1 = new ValidateContainer() {{
     this.addValidator(
          new NumberValidator("数値入力をして下さい。",display)
     );
     this.addValidator(
          new RangeValidator(0, 100, "0-100 の範囲で入力します。",display)
     );
}};
```

8. カスタムコンポーネント

これまではプリミティブなコンポーネントに対するマッピングを解説してきた。
この章ではオリジナルのコンポーネント・その他既存の Swing コンポーネントを利用する方法を解説する。

住所入力を行うカスタムコンポーネントを利用する方法を解説する。

このコンポーネントは JPanel を継承して作られており、郵便番号(postcode)、都道府県(province)、市区町村(city)、町域番地(street)、ビル建物(bldg)の JTextField を持つ。



このカスタムコンポーネント自身も BeanBinder によってバインド済みである。
具体的なコードを次に示す。

```
public class AddressBox extends JPanel {
  public BeanBinder binder = BeanBinderFactory.createSwingBinder();

  @Basic( objectKey="bean", prop="postcode" )
  private JTextField postcodeTxt = null;

  @Basic( objectKey="bean", prop="province", imeMode=IMEMode.JP )
  private JTextField provinceTxt = null;

  @Basic( objectKey="bean", prop="city", imeMode=IMEMode.JP )
  private JTextField cityTxt = null;

  @Basic( objectKey="bean", prop="street", imeMode=IMEMode.JP )
  private JTextField streetTxt = null;

  @Basic( objectKey="bean", prop="bldg", imeMode=IMEMode.JP )
  private JTextField bldgTxt = null;

  public AddressBox() {
     //コンポーネントのインスタンスの作成や配置など

     //アノテーションからマッピング
    new AnnotationParser(this.binder, this).parse();
  }
}
```

次に、このコンポーネントとバインドするオブジェクトである Address クラスを示す。

```
public class Address {
  private String postcode;     //郵便番号
  private String province;     //都道府県
  private String city;         //市区町村
  private String street;       //町域番地
  private String bldg;         //ビル・建物
  //以下 Getters/Setters
}
```

以上でコンポーネントとオブジェクトの準備が出来たが、このままでは BeanBinder はカスタムコンポーネントを認識しない。 次に、BeanBinder に対してカスタムコンポーネントを認識する方法を示す。

BindStrategyDispatcher はコンポーネント毎に適切なバインド戦略を提供する責務を持つ。
既定では JTextComponent、JComboBox、JList、JCheckBox、JRadioButton、
JTable 等の基本的なコンポーネントに対応している。

```
binder = BeanBinderFactory.createSwingBinder();
BindStrategyDispatcher bindingImpl = binder.getBindStrategyDispatcher();
```

BindStrategyDispatcher に対してオリジナルのバインド戦略を追加するには ComponentStrategyPlug を追加すれば良い。 ComponentStrategyPlug のコンストラクタは次の通りである。
第 1 引数:カスタムコンポーネントのクラス
第 2 引数:カスタムコンポーネント毎の、バインド戦略を実装したオブジェクトのインスタンス。

```
ComponentStrategyPlug plug =
   new ComponentStrategyPlug(AddressBox.class, new AddressBoxStrategy());

bindingImpl.addComponentStrategyPlug(plug);
```

コンポーネント毎のバインド戦略の実装方法を次に示す。

```
public class AddressBoxStrategy extends AbsComponentStrategy {
  public Object getValue(Bind bind) {
    AddressBox addressBox = (AddressBox)bind.getComponent();
    return addressBox.binder.getObject("bean");
  }
  public void updateDisplay(Bind bind, Object value) {
    AddressBox addressBox = (AddressBox)bind.getComponent();

    addressBox.binder.setObject("bean", value);
    addressBox.binder.updateAllComponents();
  }
}
```

AbsComponentStrategy は getValue、updateDisplay の2つのメソッドを実装する事を要求する。
getValue メソッドはコンポーネントの値を戻す必要があります。

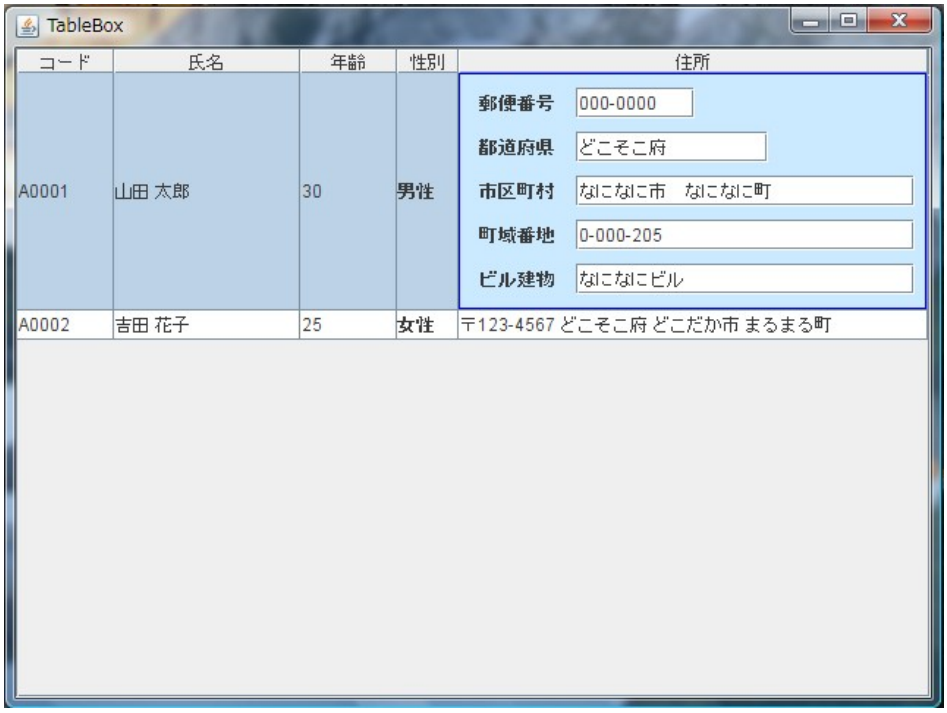
```
public Object getValue(Bind bind, Object component) {
     AddressBox addressBox = (AddressBox)component;
     return addressBox.binder.getObject("bean");
}
```

updateDisplay メソッドはコンポーネントの再描画を実装します。

```
public void updateDisplay(Bind bind, Object component, Object value) {
     AddressBox addressBox = (AddressBox)component;

     addressBox.binder.setObject("bean", value);
     addressBox.binder.updateAllComponents();
}
```

```
binder = new BeanBinder(bindingImpl);
BindStrategyDispatcher bindingImpl = binder.getBindStrategyDispatcher();

ComponentStrategyPlug plug =
   new ComponentStrategyPlug(AddressBox.class, new AddressBoxStrategy());

bindingImpl.addComponentStrategyPlug(plug);
```

以上で BeanBinder はカスタムコンポーネントを認識するようになります。
応用をするとカスタムコンポーネントをセルエディタに使用する事も可能です。



■最後に

BeanBinder は既存のプロジェクトに組み込んでも不要であればいつでも取り外せるほどに軽量である事を心掛けています。(取り外されないように最善を尽くします)

また、LGPL ライセンスを採用している為、いつでも改変して頂く事が可能です。 現在 BeanBinder は SourceForge.jp にホスティングしており、いつでも Subversion のリポジトリからソースコードを取得する事が可能です。

[beanbinder]
svn checkout http://svn.sourceforge.jp/svnroot/beanbinder/*trunk*

[beanbinder-annotations]
svn checkout http://svn.sourceforge.jp/svnroot/beanbinder/*annotations*

また現バージョンのドキュメントはまだ完全ではありません、残りの *Validation* やコレクションオブジェクトのマッピング方法など詳しくは「*http://sourceforge.jp/projects/beanbinder/*」にてデモやサンプルコードをリリースしていますのでこちらを参照して下さい。