JavaBinding

Version 0.0.1

Developer Guide

retro della copertina

Table of Contents

1 Preface

1.1 Introduction

JavaBinding is a library with scope to help a developer to implement a java mapping system.

There are function in mapping systems more similar how:

- mapping a property
- create a new instance of a java class
- using default or custom constructor
- Convert value
- extends existent mapping
- ecc...

More function can be reused in all system mapping delegating to the developer only the function of setting and getting values to the serialized sysstem (xml, json, hashtable, ecc...)

1.2 Overview

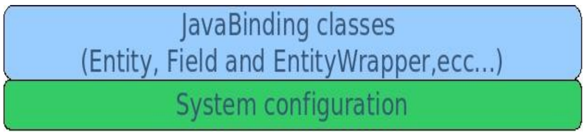
JavaBinding supply support classes to resolve mapping problems. It includes this principal packages:

- beans ==> Supply utility class to resolve reflection
- converters ==> Supply interfaces to manage the custom conversion of values
- factory ==> Supply interfaces to manage the correct reference to other mappings in order to resolve dinamically ilmapping id in base to an object
- mappings ==> Core package that contains mapping information and execute marshal and unmarshal

All classes present in it.dangelo.javabinding supply a way to wrap the mappings package.

This library don't supply any method to map the mappings class (Entity, Field, ecc...) to the configuration system. The configuration is developer's responsability that use this system.

Below a simple schema:

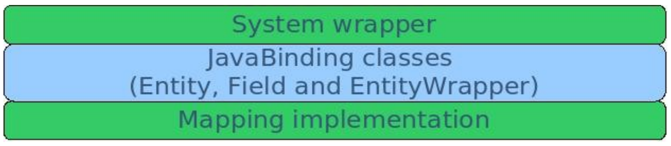


The developer must implement a system configuration to configure the Entity, EntityWrapper classes, BindFactory and Converters. This implementation can be executed in xml, property files or other.

Now JavaBinding has the information about mapping to resolve marshal and unmarshal.

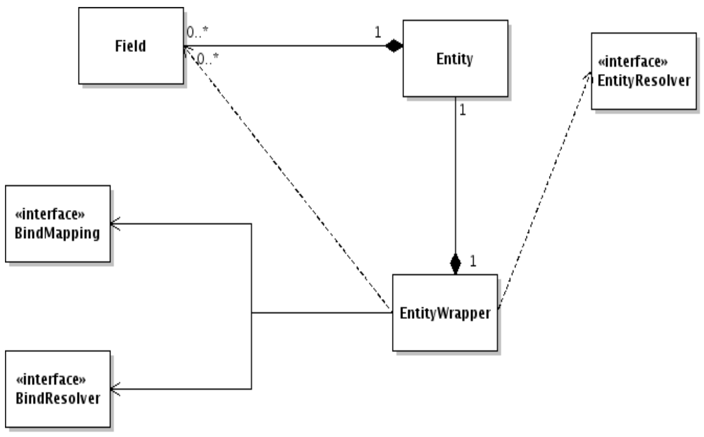
The developer must implements two interfaces to manage elements. Elements are the serialized form of the java classes, this is, what you want map (xml, Hashtable, json, ecc...).

Now a system wrapper to hide JavaBinding is required to create a correct interface for the client.



2 Mapping guide

The mappings package contains the core system for mapping java classes to other serialized forms (xml, json, ecc...).



Field represents a mapping of a single property or attribute. Contains informations of;

- attribute name ==> name used in the serialized form
- property name ==> name of the java bean property

- class type
- collection type ==> if not null is the type used to generate the list or array. To set the array you can use Object[].class
- converters
- factory
- ref id ==> reference to other id (dependencies)
- setter and getter method ==> if name are different from set[Propertyname] and is/get[Propertyname]
- ecc...

Entity represents a mapping to a java class, is composed of Field. Contains information about:

- class type
- extension of other mappings
- logical name used for mapping (example to map a java instance to a xml tag)

EntityWrapper is the executor of the marshal and unmarshal. It uses an implementation of BindMapping and BindResolver to interact with the implementation of the serialized form.

2.1 BindMapping

BindMapping supply a contract between javabinding core and the developer to interact with elements in order to create the serialized form.

It's used during the marshalling process to construct the serialized form.

Example:

```java
protected BindMapping createBindMapping() {
  return new BindMapping() {
    public Object createElement(String name) throws Exception {
      return new HashMap<String, Object>();
    }
    public Object createList(String name, Class<?> listType, int length)
                                                     throws Exception {
      return new ArrayList<HashMap<String, Object>>(length);
    }
    @SuppressWarnings("unchecked")
    public void setValue(Object element, String name, Object value)
                                                                  throws Exception {
      HashMap<String, Object> map = (HashMap<String, Object>) element;
      map.put(name, value);
    }
   @SuppressWarnings("unchecked")
   public void setValueList(Object elementList, Object value, int index)
                                                                  throws Exception {
     ArrayList<HashMap<String, Object>> list =
               (ArrayList<HashMap<String, Object>>) elementList;
     HashMap<String, Object> map = (HashMap<String, Object>) value;
     list.add(index, map);
   }
};
```

2.2 BindResolver

BindResolver supply a contract between javabinding core and the developer to interact with elements in order to retrieve data by the serialized form.

It's used during the unmarshalling process to construct the java instance.

Example:

```java
return new BindResolver() {
  @SuppressWarnings("unchecked")
  public int getLength(Object element) throws Exception {
    ArrayList<HashMap<String, Object>> list = (ArrayList<HashMap<String, Object>>) element;
    if(list == null)
      return 0;
    else
      return list.size();
  }
  @SuppressWarnings("unchecked")
  public Object getValue(Object element, String name)  throws Exception {
    if(element == null) return null;
    HashMap<String, Object> map = (HashMap<String, Object>) element;
    return map.get(name);
  }
  @SuppressWarnings("unchecked")
  public Object getValue(Object element, int index) throws Exception {
    if(element == null) return null;
    ArrayList<HashMap<String, Object>> list = (ArrayList<HashMap<String, Object>>) element;
    return list.get(index);
  }
  public boolean isArray(Object element) throws Exception {
    return (element instanceof List);
 }
};
```

3 Using JavaBinding-step by step