

Module 8 sample

Enhancing Java core libraries with BeanUtils and Lang

8.1 BeanUtils: JavaBeans made easy.....	1
8.2 Mind your Lang(uage).....	22
8.3 Summary.....	36
Index.....	37

This module continues where the previous one ended, by continuing to discuss the Commons components that enhance the Java core libraries. In the previous module, we discussed the Collections component. In this module, we'll look at the BeanUtils component, which is used to supplement the Java Reflection and Introspection API; and Lang, which provides helper classes for the Java `Lang` package.

Let's start with BeanUtils. We'll take a quick tour of its API and packaging and then delve into some examples.

8.1 BeanUtils: JavaBeans made easy

Although BeanUtils is really a supplement to the Java Reflection and Introspection API, its main functionality is directed toward the ability to read and write JavaBeans without needing to know the name and signature of their properties or methods.

BeanUtils (short for Bean Introspection Utilities) relies on the JavaBeans that it works on to follow the standard JavaBean specification. The specification puts some restrictions on the structure of JavaBeans. However, BeanUtils is lenient in terms of these restrictions and only requires the following two to be enforced:

- A JavaBean must have a no-argument constructor, and its class must be public.
- The properties of a JavaBean must follow a standard pattern. This pattern ensures that a property can be accessed by using the method `getXXX` (or `isXXX` for boolean properties), where `XXX` is the property name. Similarly, properties can be modified by using the method `setXXX`. These methods are typically called the accessor and mutator methods, respectively. Note that there must be only one accessor or mutator for each property; overloaded methods will fail to work with BeanUtils.

The current version of BeanUtils is 1.7.0. With this release, BeanUtils has been separated into two libraries. The first library, `commons-beanutils-core.jar`, contains the core BeanUtils classes. The second library, `commons-beanutils-bean-collections.jar`, is an add-on that allows you to use BeanUtils with the Collections component. If you don't plan to use BeanUtils with the Collections API, it's recommended that you use the core library, because it removes the dependency on the Collections component. In this module we'll use both libraries, using the combined jar file `commons-beanutils.jar`. This means that the Collections components jar file must also be in your `CLASSPATH` in order for any of the Collection-BeanUtils examples to work.

Before we look at any examples, you must first understand the terminology associated with JavaBeans and BeanUtils.

8.1.1 Understanding the terminology

Simple. Scalar. Indexed. Nested. Mapped. DynaBean. *Lazy* DynaBean. If these terms leave you confused in relation to JavaBeans and BeanUtils, then this section will give you a quick overview of what they all mean. It's important that you have a clear understanding of these terms, because we'll use them a lot later in this module. Therefore, with the help of an example, we'll explain them here. If you already understand these terms, then it's safe to skip this section, keeping in mind that this section also introduces the JavaBeans we'll work with while explaining the BeanUtils examples.

Property types

JavaBean properties can be divided into four types, based on what they represent. These four types are scalar (or simple), indexed, mapped, and nested. Before we describe these types, look at listings 8.1, 8.2, and 8.3, which show three example JavaBeans.

Listing 8.1 Going to the movies with the `Movie` JavaBean

```
package com.manning.common.chapter08;

import java.util.Map;
import java.util.List;
import java.util.Date;

public class Movie {
    public Movie() {
    }

    public Date getDateOfRelease() { return this.dateOfRelease; }
    public void setDateOfRelease(Date dateOfRelease) {
        this.dateOfRelease = dateOfRelease;
    }

    public String getTitle() { return this.title; }
    public void setTitle(String title) {this.title = title; }

    public Person getDirector() { return this.director; }
    public void setDirector(Person director) { this.director = director; }

    public List getActors() { return this.actors; }
    public void setActors(List actors) { this.actors= actors; }

    public String[] getKeywords() { return this.keywords; }
    public void setKeyWords(String[] keywords) { this.keywords = keywords; }

    public Map getGenre() { return this.genre; }
    public void setGenre(Map genre) { this.genre = genre; }

    private Date dateOfRelease;
    private String title;
    private Person director;
}
```