DaoJones 2.0

Accessing non-relational databases the easy way

Ralf Zahn

ARS Training & Consulting

Table of Contents

List of Examples

Acknowledgements

This book is produced by the Wikbook tool. Wikbook is an open source project for converting wiki files into a set of docbook files.

Preface

The preface is any content before a wiki section.

1

Introduction

DaoJones is designed for accessing databases by mapping Java objects to database entities. Those databases must not be relational.

1.1. History

The project was initially founded in 2007 as part of the ARS Website project. The main purpose at that time was to encapsulate accesses to the website database - which is a Notes database - in a separate layer to make the business entities independent from the Notes Java API. This allowed to implement thread safety at a central position and to make this available for the whole application.

By now, DaoJones is used by multiple ARS-internal projects (e.g. the *Unterlagengenerierung*) and by currently a single customer project too.

The name *DaoJones* is derived from the abbreviation *DAO*, which is used for *Data Access Objects*, a design pattern used for persistence implementations. The term is not used anymore, it was replaced by a simple POJO, briefly called *bean*.

> Although it allows to use different database drivers, currently there is only one driver implementation to access Notes databases. Further drivers must yet be implemented.

1.2. Intention

The framework is intented to be used by Java applications to map business entities to databases, no matter what kind of database (relational, document-based, object-based, file-based) is used at runtime. The database can be replaced without any changes within the application code. The developer can handle the concepts and terms of DaoJones and does not have to get familiar with database-specific concepts and terms.

2

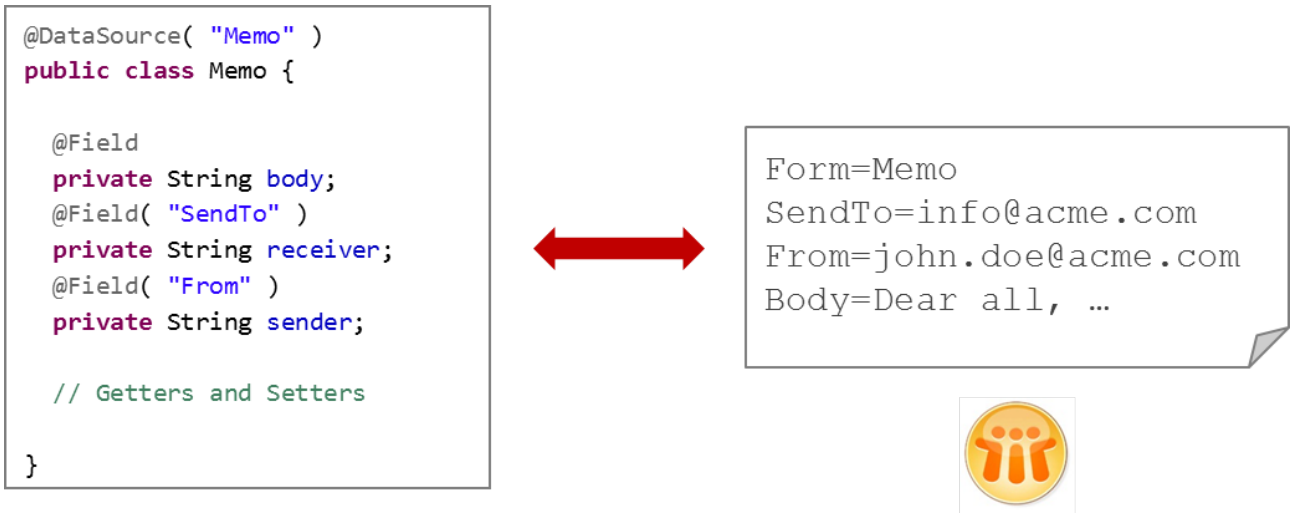
Concepts

This chapter names and describes basic concepts and terms that should be known to understand the functionality of the framework.

2.1. Bean Mapping

The most important concept is the mapping of beans to an artifact within the database. The following graphic shows the mapping between a Java bean and a Notes document. The Java bean has a class that is mapping to the *Form* field of the Notes document. The fields of the class are mapped to further fields of the Notes document. This mapping can be customized by annotations, an external XML file or programmatically at runtime using the *Bean Model API*. See chapter Bean Model for detailed information.

Figure 2.1. The bean mapping to a Notes document



It is also possible to map a bean to a view, i.e. a selection of entities based on a query and a provision of columns based on single fields or on calculated values. Views can provide sorting and categorizing entries.

Figure 2.2. The bean mapping to a Notes view

```
@DataSource( value="Persons",
    type=DataSourceType.VIEW )
public class Person {

  @Field
  private String firstName;
  @Field
  private String lastName;
  @Field
  private Date birthDate;

  // Getters and Setters

}
```

firstName	lastName	birthDate
John	Doe	01.01.1970
Max	Miller	01.02.1934

Please be aware that view columns with calculated values can only be mapped to read-only fields and are therefore cannot be used for updates.

2.2. Datatype Mapping

The mapping of Java and database fields or columns is not limited to strings. The following types can be used as Java field type:

- Primitive types and their wrapper classes
- `java.lang.String`
- `java.util.Date`
- One-dimensional Arrays of these types
- `de.ars.daojones.runtime.beans.fields.Resource` (for BLOBs, CLOBs, Attachments...) - see Section Attachments

- "Bean", Mapping (Field and DataSource/TABLE/VIEW, Supported Types (inkl. Arrays, Resource, ...)) - Models (Connection-, Bean-, ConnectionFactory-, CacheFactory-, Application-) - DaoJonesContext, -Factory und -Config

2.3. Bean Model

3

Database Access (Basics)

TODO

3.1. A simple example

Example 3.1. A first example code

```
public void demo1() throws DataAccessException,
ConfigurationException {
  final DaoJonesContextFactory dcf = new DaoJonesContextFactory();
  dcf.setBeanConfigurations( new ApplicationBeanConfigurationProvider(
          DaoJonesDemo.APP,
          new AnnotationBeanConfigurationProvider() ) );
  dcf.setConnectionConfigurations( new ApplicationConnectionConfigurationProv
          DaoJonesDemo.APP,
          new XmlConnectionConfigurationProvider(
                  DaoJonesDemo.class
                  .getResource( "daojones-connections.xml" ) ) ) );

  final DaoJonesContext ctx = dcf.createContext();
  try {
    final ConnectionProvider cp = ctx
            .getApplication( DaoJonesDemo.APP );
    final Connection<Memo> con = cp.getConnection( Memo.class );
```

```
  try {
    // SearchCriteria etwas anders?
    final SearchResult<Memo> memos = con.findAll( Query.create()
            .only( 2 ) );
    for ( final Memo memo : memos ) {
      // Iterate through the search results
      // (Streamed results)

    }
  } finally {
    con.close();
  }
} finally {
  ctx.close();
}

Query.create().only(
        property( "receiver" ).isEmpty().or().property( "sender" )
        .isEmpty() );
Query.create().only( not( TRUE() ).and().TRUE().or().FALSE() );
Query.create()
.only( property( "title" ).isEmpty().and().field( "text" )
        .asString().startsWith( "Hallo" ) ).only( 10 );
Query.create().only(
        property( "title" ).isEmpty().or().asString()
        .contains( "Tagebuch" ).and().startsWith( "Mein" )
        .or().endsWith( "Dein" ) );
Query.create().only(
        property( "authors" )
```

```
            .asCollectionOf( String.class )
            .containsAllOf( "Ralf Zahn", "Michael Mueller" )
            .or()
            .containsOneOf( "Stefan Schaeffer",
                    "Domink Ebert" ) );
```

```
}
```

1. Compatible with try-with-resources (since Java 7)

4

Database Access (Advanced)

TODO

4.1. Attachments

5

JUnit Test Support

TODO

6

Integration into Java Environments

TODO

7

Perspectives

TODO