Alles nur geklaut?

Jakarta Data

Sascha Groß
sascha.gross@mathema.de

Agenda

- Was ist Jakarta Data
- Architektur
- API
- Beispiele Demo
- Ausblick

Jakarta Data 1.0

"Jakarta Data standardizes a programming model for relational and non-relational data access in which the user composes repository interfaces that define operations on entities (simple Java objects that represent data), and the container/runtime provides the implementation."

- Release 30. September 2024
- Bestandteil von Jakarta EE 11
- mindestens Java 17
- Jakarta NoSQL 1.0 (nicht Bestandteil von JEE 11)
- Jakarta Persitence 3.2

Features

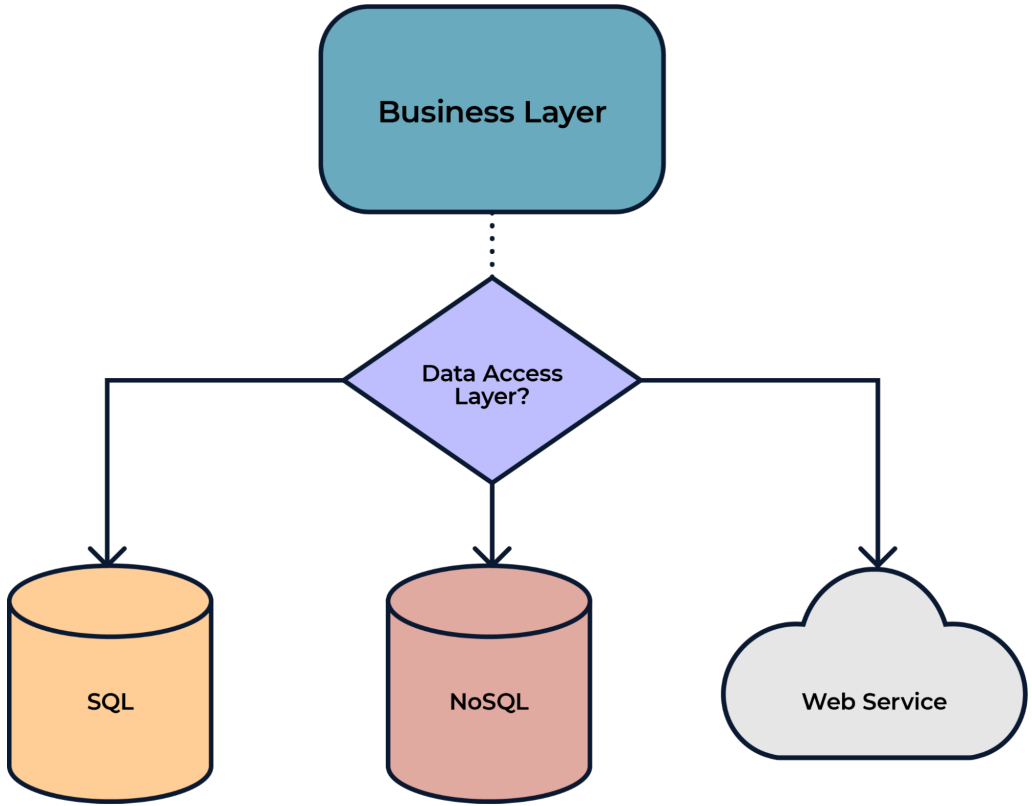
- Built-in Repository interfaces
- Benötige Entity models von Jakarta Persistence und Jakarta NoSQL
- Injection of repositories (@Inject)
- @Insert, @Delete, @Save, @Update,
- @Find, @Query
  - Limit, Sortierung, Pagination
- Jakarta Data Query Language (JDQL) (Untermenge von Jakarta Persistence Query Language)
- Query by Method Name

Implementierungen

- Hibernate 6.6 https://hibernate.org/orm/releases/6.6/
- Eclipse JNoSQL 1.1.4 https://github.com/eclipse-jnosql/jnosql/releases/tag/1.1.4
- Open Liberty 24.0.0.6

  https://public.dhe.ibm.com/ibmdl/export/pub/software/openliberty/runtime/tck/2024-05-06_1951/openliberty-24.0.0.6-beta.zip

Architektur

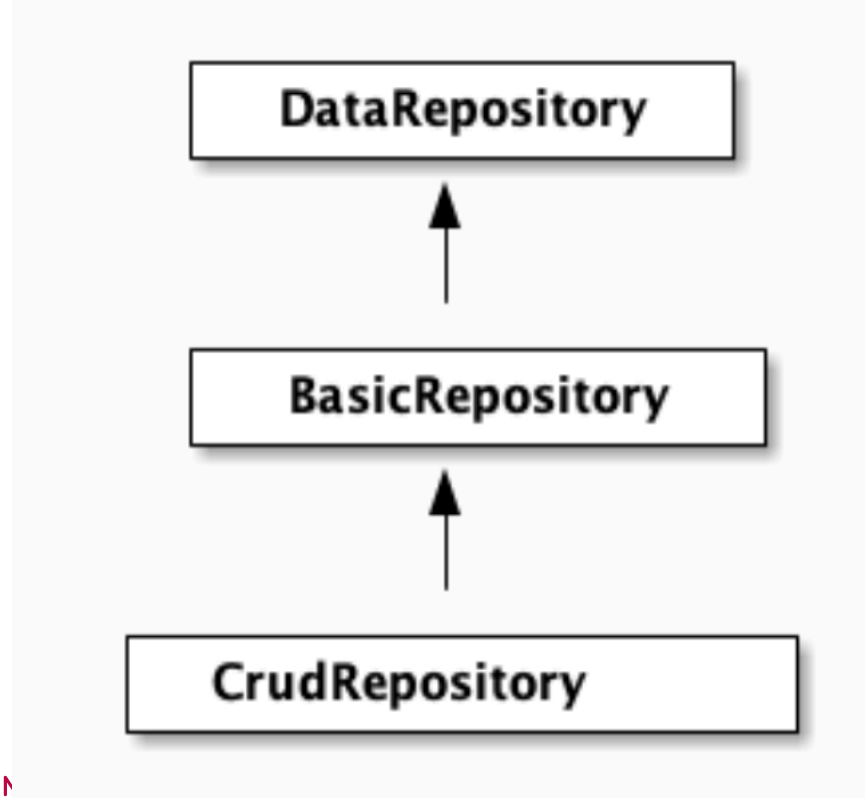


Architektur - Repository Pattern

“ The Repository pattern is a fundamental concept within Jakarta Data that plays a central role in data access and management. Essentially, a repository is a mediator between an application's domain logic and the underlying data storage, be it a relational database, NoSQL database, or any other data source.”

Repositories

```
1   @Repository
```



"**Stateless Repositories**: Repositories are stateless. This specification does not address the definition of repositories that externalize Jakarta Persistence-style stateful persistence contexts."

Repository - DataRepository

```
1   public interface DataRepository<T, K> {
2
3   }
```

Repository - BasicRepository

```java
public interface BasicRepository<T, K> extends DataRepository<T, K> {
    @Delete
    void delete(T entity);
    @Delete
    void deleteById(@By(ID) K id);
    @Delete
    void deleteAll(List<? extends T> entities);

    @Find
    Optional<T> findById(@By(ID) K id);
    @Find
    Stream<T> findAll();
    @Find
    Page<T> findAll(PageRequest pageRequest, Order<T> sortBy);

    @Save
    <S extends T> S save(S entity);
    @Save
    <S extends T> List<S> saveAll(List<S> entities);
}
```

Repository - CrudRepository

```java
public interface CrudRepository<T, K> extends BasicRepository<T, K> {

    @Insert
    <S extends T> S insert(S entity);

    @Insert
    <S extends T> List<S> insertAll(List<S> entities);

    @Update
    <S extends T> S update(S entity);

    @Update
    <S extends T> List<S> updateAll(List<S> entities);
}
```

Repository ohne Built-in Supertypes

```
1  @Repository
2  public interface Garage {
3      @Insert
4      Car park(Car car);
5
6      @Delete
7      void unpark(Car car);
8  }
```

Repository mit Built-in Supertypes

```
1  @Repository
2  public interface Garage extends BasicRepository<Car, String> {
3
4  }
```

Annotated Query methods

```
@Query("where title like :title order by title asc, id asc")
Page<Book> booksByTitle(String title, PageRequest pageRequest);

@Query("where p.name = :prodname")
Optional<Product> findByName(@Param("prodname") String name);
```

Parameter-based automatic query methods

```
@Find
Book bookByIsbn(String isbn);

@Find
List<Book> booksByYear(Year year, Sort<Book> order, Limit limit);

@Find
Page<Book> find(
    @By("year") Year publishedIn,
    @By("genre") Category type,
    Order<Book> sortBy,
    PageRequest pageRequest);
```

Resource accessor method

```
1   java.sql.Connection connection();
2
3   default void cleanup() {
4       try (Statement s = connection().createStatement()) {
5           s.executeUpdate("truncate table books");
6       }
7   }
```

Unterstütze Ressourcen sind abhängig vom Jakarta Data provider

- java.sql.Connection (JDBC)
- java.sql.DataSource (JDBC)
- jakarta.persistence.EntityManager (JPA)

Sortieren, Pagination, Limit

```
1   @Query("WHERE u.age > ?1")
2   @OrderBy(_User.AGE)
3   Page<User> findByNamePrefix(
4       String namePrefix,
5       PageRequest pagination,
6       Order<User> sorts);
7
8   @Query("WHERE u.age > ?1")
9   @OrderBy(_User.AGE)
10  List<User> findByNamePrefix(
11      String namePrefix,
12      Sort<?>... sorts);
13
14  @Query("where name like :pattern")
15  List<Product> findByNameLike(String pattern, Limit max, Sort<?>... sorts);
```

Query by Method Name

Query by Method Name is a query language suitable for embedding in the names of methods written in Java. As such, its syntax is limited to the use of legal identifier characters, so the text of a query must contain neither whitespace, nor punctuation characters, nor numeric operators, nor comparison operators.

Jakarta Data 1.0 offers a Query by Method Name facility as an extension to the specification, providing a migration path for existing applications written for repository frameworks which offer similar functionality.

```
1   @Repository
2   public interface ProductRepository extends BasicRepository<Product, Long> {
3
4     List<Product> findByName(String name);
5
6     @OrderBy("price")
7     List<Product> findByNameLike(String namePattern);
8
9     List<Product> findByNameLikeAndPriceLessThanOrderByPriceDesc(String namePattern, float priceBelow);
10
11  }
```

Query by Method Name

"A Jakarta Data provider is required to support the Query by Method Name extension in Jakarta Data 1.0."

"NOTE: A Jakarta Data provider backed by a key-value or wide-column datastore is not required to support Query by Method Name."

" WARNING: This functionality is considered deprecated, and the requirement that a provider support the Query by Method Name extension will be removed in a future version of Jakarta Data."

Query by Method Name Extension - BNF

```
query : find | action
find : "find" limit? ignoredText? restriction? order?
action : ("delete" | "count" | "exists") ignoredText? restriction?
restriction : "By" predicate
limit : "First" max?
predicate : condition (("And" | "Or") condition)*
condition : attribute "IgnoreCase"? "Not"? operator?
operator
    : "Contains" | "EndsWith" | "StartsWith" | "LessThan"
    | "LessThanEqual" | "GreaterThan" | "GreaterThanEqual"
    | "Between" | "Like" | "In"
    | "Null" | "True" | "False"
attribute : identifier ("_" identifier)*
identifier : word
max : digit+
order : "OrderBy" (attribute | orderItem+)
orderItem : attribute ("Asc" | "Desc")
```

BNF elements (1/3)

Rule name	Explanation
`query`	May be a `find` query, or a `delete`, `count`, or `exists` operation.
`find`	A `find` query has an optional limit and optional restriction on records to be retrieved, and optional sorting.
`action`	Any other kind of operation has only a restriction to a subset of records.
`restriction`	Restricts the records returned to those which satisfy a predicate
`limit`	Limits the records retrieved by a `find` query to a hardcoded maximum, such as `First10`.

BNF elements (2/3)

Rule name	Explanation
`ignoredText`	Optional text that does not contain `By`, `All`, or `First`.
`predicate`	A filtering criteria, which may include multiple conditions separated by `And` or `Or`.
`condition`	An attribute of the queried entity and an operator.
`operator`	An operator belonging to a condition, for example, `Between` or `LessThan`. When absent, equality is implied.
`attribute`	An entity attribute name, which can include underscores for nested attributes.

BNF elements (3/3)

Rule name	Explanation
`identifier`	A legal Java identifier, not containing an underscore.
`max`	A positive whole number.
`order`	Specifies that results of a `find` query should be sorted lexicographically, with respect to one or more order items.
`orderItem`	An entity attribute used to sort results, where `Asc` or `Desc` specifies the sorting direction.

Return Types

Operation	Return type	Notes
`count`	`long`	
`delete`	`void` , `long` , `int`	
`exists`	`boolean`	

Return Types

Operation	Return type	Notes
`find`	`E` or `Optional<E>`	For queries returning a single item (or none)
`find`	`E[]` or `List<E>`	For queries where it is possible to return more than one item
`find`	`Stream<E>`	The caller must call `java.util.stream.BaseStream.close()` for every stream returned by the repository method
`find` accepting a `PageRequest`	`Page<E>` or `CursoredPage<E>`	For use with pagination

Persistent attribute names in Query by Method Name

```
class Person {
  private Long id;
  private MailingAddress address;
}

class MailingAddress {
  private int zipcode;
}
```

```
List<Person> findByAddressZipCode(int zipCode);
List<Person> findByAddress_zipcode(int zipCode);
```

Persistent attribute names in Query by Method Name

```
class Customer {
  private Long id;
  private String addressZipCode;
  private MailingAddress address;
}

class MailingAddress {
  private int zipcode;
}
```

```
List<Customer> findByAddress_zipcode(int zipCode);
```

Demo

Ausblic
Jakarta Data 1.1 (under development)

Erweiterungen for Jakarta EE 12

- Reactive patterns and integration with Jakarta Concurrency Asynchronous
- Move Jakarta Data Query Language to Jakarta Query specification (if accepted and included in Jakarta EE 12) where it will remain fully compatible.
- Configuration via Jakarta Config (if Jakarta Config is released and makes it into Jakarta EE 12 in time)

MATHEMA CAMPUS

Alles nur geklaut? - Jakarta Data

Dankeschön / Fragen?

https://github.com/saschagr/jakarta-data

Sascha Groß
sascha.gross@mathema.de

Quellen

https://jakarta.ee/specifications/data/

https://jakarta.ee/specifications/data/1.0/jakarta-data-1.0.pdf

https://jakarta.ee/specifications/data/1.0/jakarta-data-addendum-1.0.pdf

https://hibernate.org/orm/

https://github.com/hibernate/hibernate-orm

https://docs.jboss.org/hibernate/orm/6.6/repositories/html_single/Hibernate_Data_Repositories.html

https://www.mastertheboss.com/java-ee/jakarta-ee/getting-started-with-jakarta-data-api/

https://www.mastertheboss.com/java-ee/jakarta-ee/jakarta-data-in-action/

https://github.com/OpenLiberty/sample-jakarta-data

https://github.com/JNOSQL/demos-se/