JakartaOne 2025 Japan

Jakarta Data/Persistence のアップデート

2025/7/30 14:00-14:20

NEC　景井教天

自己紹介

NEC
テクノロジーサービスソフトウェア統括部
景井 教天

2021年 NEC 入社

Javaアプリケーションサーバ(WebOTX Application Server) 開発・保守
Springサポート 企画・保守

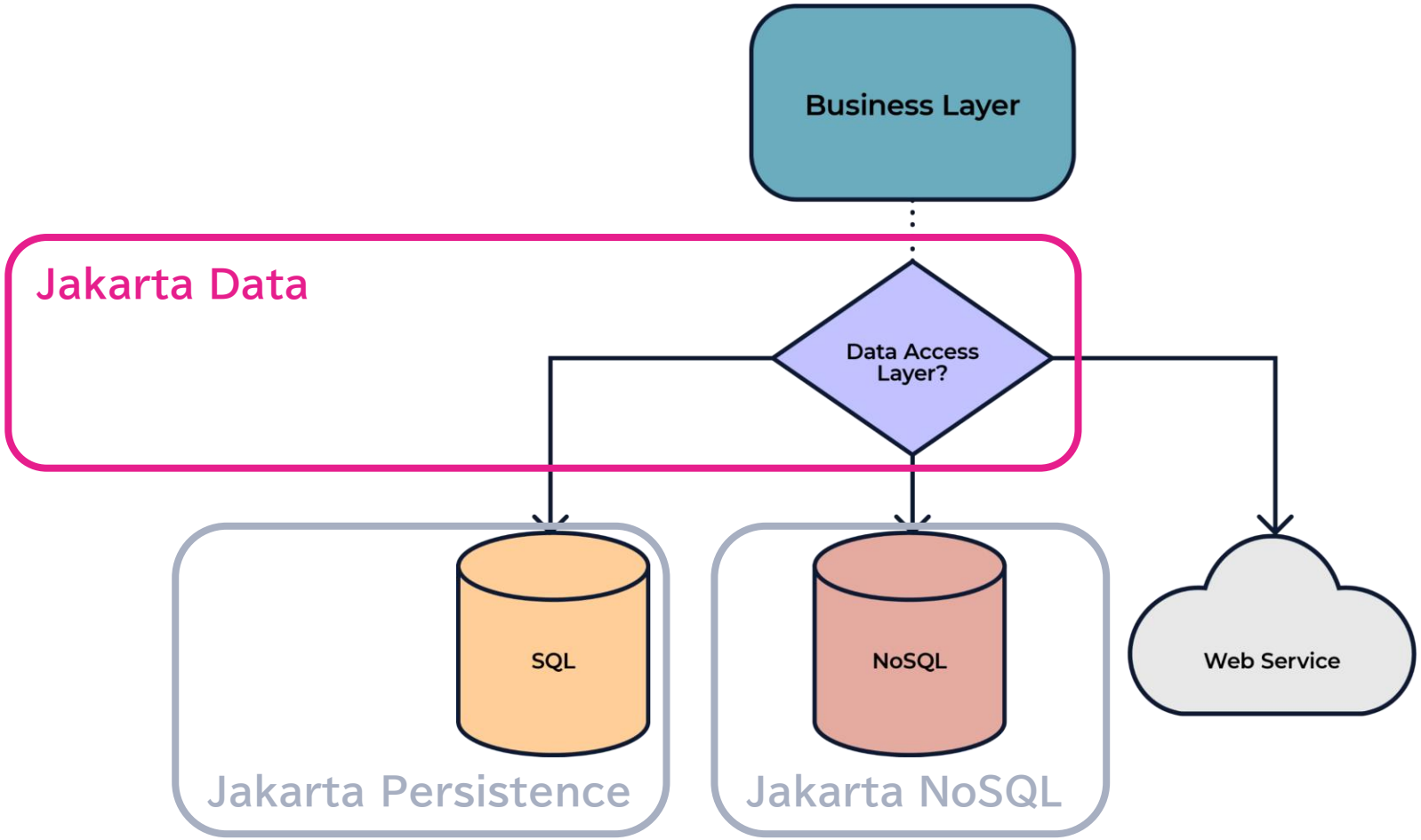
2025年 技術書出版

改訂3版　パーフェクトJava
https://gihyo.jp/book/2025/978-4-297-14680-1

Jakarta EE 11

Jakarta Data とは

現代開発において、扱うデータの多様化により管理が複雑化した
データアクセスのための定型的なコードが排除され、開発者がビジネスロジックに集中しやすい仕様が求められた



従来比較

本セッションは下記情報を前提に説明

■ Bookエンティティ

```
@Entity
public class Book {
    @Id
    private Long id;
    private String title;
    private String author;
    private int price;

    // getter, setter
}
```

データアクセス処理の呼び出し

■ DAOの場合

```
@Inject
private BookDao bookDao;
...
bookDao.findById(id);
```

■ リポジトリの場合

```
@Inject
private BookRepository repository;
...
repository.findById(id);
```

従来比較

従来手法におけるリポジトリパターンの定型的なコードがシンプルに！

従来(Jakarta Persistenceのみ)

```java
@RequestScoped
public class BookDao {

    @PersistenceContext
    private EntityManager em;

    public Book findById(Long id) {
        return em.find(Book.class, id);
    }

    public void save(Book book) {
        em.persist(book);
    }

    public List<Book> findByAuthor(String author) {
        return em.createQuery(
                    "SELECT b FROM Book b " +
                    "WHERE b.author = :author", Book.class)
                 .setParameter("author", author)
                 .getResultList();
    }
}
```

Jakarta Data

```java
@Repository
public interface BookRepository extends CrudRepository<Book, Long> {
    @Find
    List<Book> findByAuthor(String author);
}
```

←スライドの都合上、その他メソッドを割愛

データアクセスレイヤーの構成

リポジトリとカスタムクエリを組み合わせてデータアクセスレイヤーを構築

Repositories in Jakarta Data

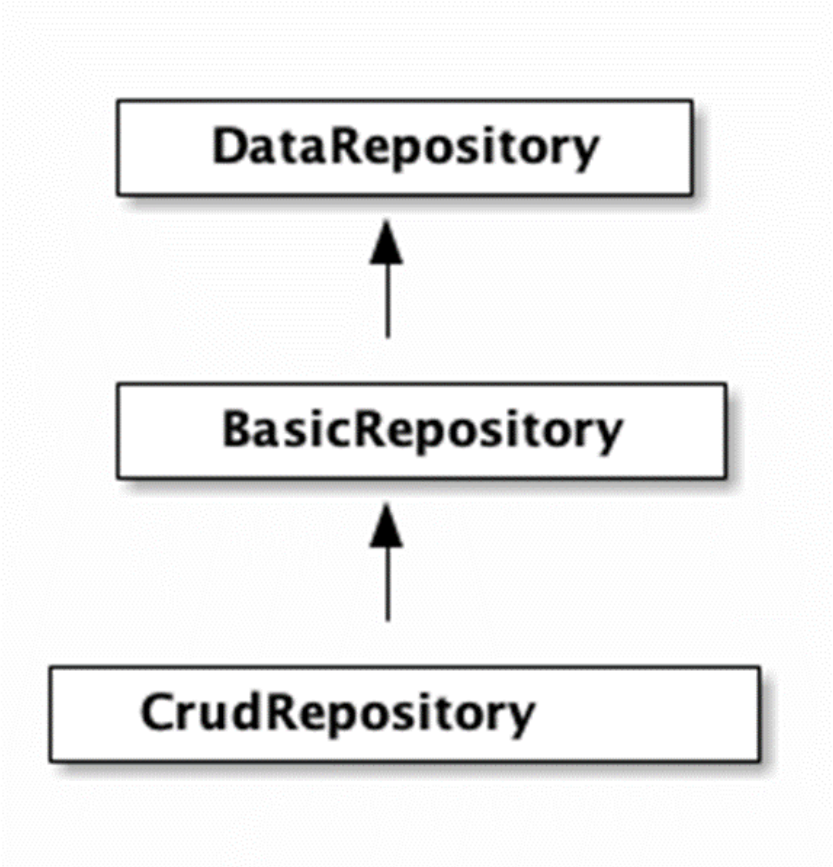
リポジトリがデータ操作
(保存/取得/更新/削除)を担当

Querying in Jakarta Data

カスタムクエリで複雑なデータ操作を実現

Jakarta Data 提供のリポジトリ

一般的なデータアクセス処理が定義されたリポジトリインタフェースを提供



DataRepository

メソッドなし

BasicRepository

操作	メソッド
取得	findById(), findAll()
削除	delete(), deleteById(), deleteAll()
保存	save(), saveAll()
	データが存在する　：エンティティの更新 データが存在しない: エンティティの挿入

CrudRepository

操作	メソッド
挿入	insert(), insertAll()
更新	update(), updateAll()

リポジトリの自動実装

リポジトリインタフェースを継承することでデータアクセス処理をJakarta Data側で自動実装

@Repository アノテーション

- コンテナ/ランタイムがリポジトリと認識
- CDIが有効な場合は、CDI Bean定義のアノテーション

サンプルコード

- Bookエンティティに対するCRUD操作が自動実装される

```
@Repository
public interface BookRepository extends CrudRepository<Book, Long> {
}
```

リポジトリメソッドの独自実装

リポジトリインタフェースで提供される処理で不十分な場合は、リポジトリメソッドを独自に実装

推奨

1. Lifecycle Methods

Jakarta Data 独自のライフサイクルアノテーションを使用してリポジトリメソッドを実装

推奨

2. Parameter-based Automatic Query Methods

メソッドの引数情報を使用して動的にクエリを生成

推奨

3. JDQL Query Methods

複雑なクエリをJPQLに似た構文で記述

拡張機能

4. Query by Method Name

リポジトリメソッド名から自動的にクエリを生成

1. Lifecycle Methods

■ Jakarta Data 独自のライフサイクルアノテーション

アノテーション	説明
@Insert	エンティティの新規作成
@Update	既存エンティティの更新
@Save	データが存在する　: エンティティの更新 データが存在しない: エンティティの新規作成
@Delete	エンティティの削除

記述例
@アノテーション
*型[*1]　メソッド名(型　引数);*

*1: 指定可能な型一覧: E, List<E>, E[], void(@Deleteでは使用不可)

■ サンプルコード

```
@Insert
Book create(Book book);

@Update
List<Book> modifyAll(List<Book> books);

@Save
void upsertBooks(List<Book> books);

@Delete
void remove(Book book);
```

2. Parameter-based Automatic Query Methods

■ @Find アノテーション

コンテナ/ランタイムが
リポジトリメソッドを検索クエリメソッドと認識

クエリ条件を引数で指定

記述例
@Find
*型[*1]　メソッド名(型　引数);*

*1: 指定可能な型一覧: E, Optional<E>, E[], List<E>, Page<E>

■ サンプルコード

```
@Find
Book bookById(String id);

@Find
Optional<Book> bookByIsbn(
        @By("id") Long isbn );

@Find
List<Book> booksCreatedBy(
        @By("author") String creator );
```

■ @By アノテーション

永続化フィールドと引数を対応付け

記述例
@By(“永続化フィールド名”)　型　引数

2. Parameter-based Automatic Query Methods（ソート機能）

■ ソート

取得結果を指定した条件で並び替え

リポジトリ側 / 呼び出し側で条件指定

- リポジトリ側: @OrderBy
- 呼び出し側:　Order/Sort オブジェクト

■ サンプルコード

リポジトリメソッド

```
@Find
@OrderBy("title")
List<Book> findBooksByAuthor(String author);


@Find
List<Book> findBooksByAuthor(
        String author, Sort sort );
```

呼び出し例

```
// 1000円未満 かつ titleの昇順で結果取得
repository.findBooks(1000);

repository.findBooks(1000, Sort.asc("title"));
```

2. Parameter-based Automatic Query Methods (取得数指定)

取得件数の指定

大量データを一度で取得することを防げる

■ **制限付き**

Limit オブジェクトで件数を指定
先頭からN件という取得方法

■ **ページネーション**

大量データを一度で取得することを防げる
指定した「ページ」単位で結果を取得

■ **サンプルコード**

リポジトリメソッド

```
@Find
List<Book> findByAuthor(
        String author, Limit limit );

@Find
Page<Book> findByAuthor(
        @By(“author”) String author,
        PageRequest pageRequest );
```

呼び出し例

```
repository.findByAuthor(
        "noritaka kagei", Limit.of(3) );

PageRequest request = PageRequest.ofPage(
        1, 10, true );
Page<Book> page = repository.findByAuthor(
        "noritaka kagei", request );
List<Book> books = page.content();
```

3. JDQL Query Methods

■ @Query アノテーション

結合/集計/サブクエリなどの高度なクエリ生成に有用

JDQL(Jakarta Data Query Language)記述[*1]を使用
SELECT/FROM句を省略可能[*2]

いずれかのパラメータでフィールド値を指定

- 名前付きパラメータ *(:paramName)*
- 位置パラメータ *(?1)*

■ サンプルコード

```
// JDQL with a named parameters
@Query("where title like :title")
List<Book> booksByType(String title);

// JDQL with positional parameters
@Query("update book set price = ?1 where title
= ?2")
Optional<Book> changePriceByTitle(int price,
String title);

// JPQL using a named parameter
@Query("delete Book where title = :title")
void removeBook(String title);
```

*1 プロバイダーが対応していればJPQL(Jakarta Persistence Query Language)記述も可能
*2 SELECT対象がEntity全体の場合、FROM対象がリポジトリの型から推測可能な場合

4. Query by Method Name

メソッドの命名規則に従い、自然言語のようにクエリを生成

■ 命名規則

記述例

*<**prefix**>By<expression>(型　引数);*

*expressionはEntityのフィールド、**keyword**(条件、順序)で構成される*

prefix	keyword(条件)		keyword(順序)
count	And	LessThan	Asc
delete	Between	LessThanEqual	Desc
exists	Contains	Like	OrderBy
find	EndsWith	Not	
	False	Null	
	GreaterThan	Or	
	GreaterThanEqual	StartsWith	
	IgnoreCase	True	
	In	First*1	

*1: メソッド名は findFirst〇By… という規則に従う。先頭から〇件の情報取得

■ サンプルコード

```
long countByTitleContains(String keyword);
void deleteByPriceGreaterThan(int price);
List<Book> findByAuthor(String author);
```

■ 注意事項

同様の機能を持つ既存アプリケーションからの
移行容易性のために提供される機能

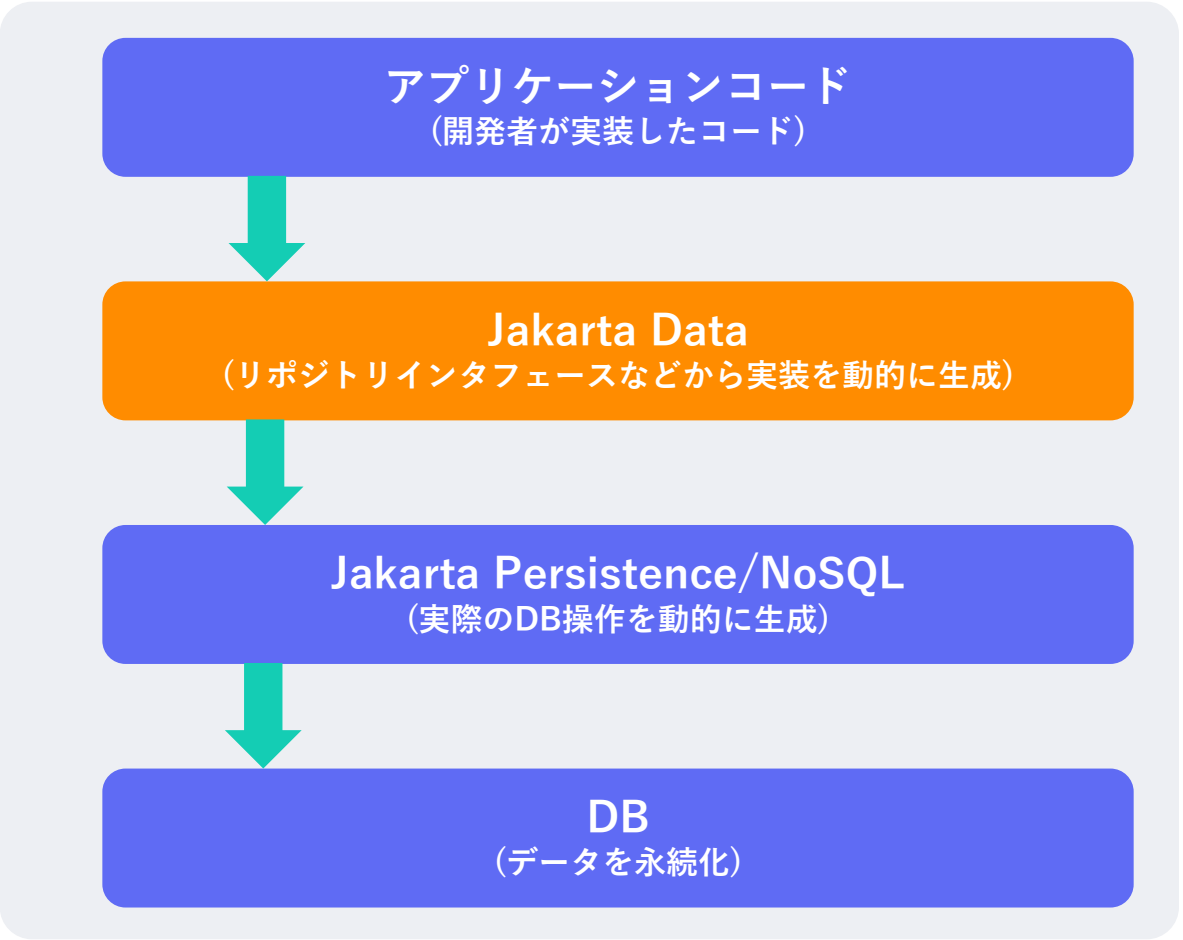
将来的にサポートされない可能性が示唆されている

(Jakarta Data 1.0 仕様より抜粋)
As an extension to the core specification, Jakarta Data 1.0 offers a Query by Method Name facility to provide a migration path for existing applications written for repository frameworks which offer similar functionality. Query by Method name is described in a companion document to this specification.
A Jakarta Data provider is required to support the Query by Method Name extension in Jakarta Data 1.0. This requirement will be removed in a future version of Jakarta Data.

まとめ

Jakarta Data は Jakarta Persistence/NoSQL などの仕様の上で動作
Jakarta Data で実装できないアクセス処理は1つ下のレイヤーで実装

- 定型的なコードの削減による開発生産性の向上
- 最新仕様(機能)への追従が容易



最後に

Jakarta Data を活用することで
Javaエンタープライズアプリケーションの標準仕様(Jakarta EE)に準拠し
開発効率・堅牢性・信頼性などのエンタープライズ要件を満たすことができる

\Orchestrating a brighter world