フレームワーク応用実習

02. MyBatis 入門

株式会社ジードライブ

© ZDRIVE, K.K. All rights reserved.

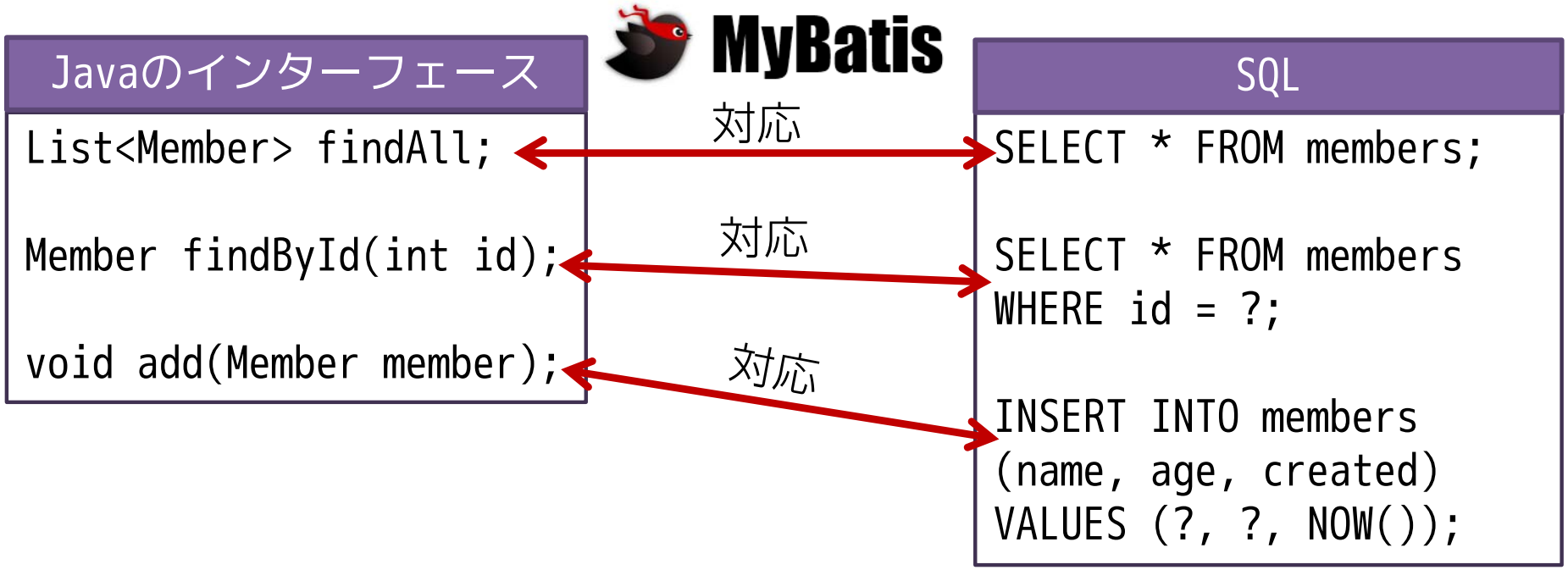
今回学ぶこと

- MyBatisの利用方法

© ZDRIVE, K.K. All rights reserved.

MyBatisとは

- Java用のデータベース・アクセス・フレームワーク
  - JavaのメソッドとSQLの対応付け(マッピング)を行う
    ⇒ SQL Mapperと呼ばれる



© ZDRIVE, K.K. All rights reserved.

マッピングの指定

- マッピングを指定する方法は２通りある

① XMLファイルを使用する方法
  - <select> や <insert> などのタグを記述する
  - 複雑なマッピングに対応しやすい

② アノテーションを使用する方法
  - @Select や @Insert など を記述する
  - シンプルなマッピングに向いている

XMLを利用する方法を学習する

© ZDRIVE, K.K. All rights reserved.

XMLによるマッピング指定

インターフェース

```
@Mapper
public interface MemberMapper {
    List<Member> findAll();
    void add(Member member);
}
```

XML

```
<mapper namespace="com.example.mapper.MemberMapper">
  <select id="findAll" resultType="com.example.domain.Member">
    SELECT * FROM members
  </select>
  <insert id="add" parameterType="com.example.domain.Member">
    INSERT INTO members (name, age, created)
    VALUES (#{name}, #{age}, NOW())
  </insert>
</mapper>
```

© ZDRIVE, K.K. All rights reserved.

アノテーションによるマッピング指定

インターフェース

```
@Mapper
public interface MemberMapper {
    @Select("SELECT * FROM members")
    List<Member> findAll();

    @Insert("INSERT INTO members (name, age, created)"
            + " VALUES (#{name}, #{age}, NOW())")
    void add(Member member);

    @Update("UPDATE members SET"
            + " name = #{name}, age = #{age}"
            + " WHERE id = #{id}")
    void edit(Member member);

    @Delete("DELETE FROM members WHERE id = #{id}")
    void deleteById(int id);
}
```

© ZDRIVE, K.K. All rights reserved.

MyBatisの使用手順

1. pom.xmlに依存関係を追記する
2. application.propertiesにデータベース接続設定やMyBatisの設定を記述する
3. SQLと紐づけるメソッドを定義したインターフェース(マッパー)を作成する
4. SQL文を記すためのXMLファイルを作成する
5. 3番で作成したマッパーを使用し、データベースに対する操作を行う

© ZDRIVE, K.K. All rights reserved.

1. 依存関係の追記

- 新規プロジェクト作成時に、依存関係に以下を追加する

カテゴリ	依存関係	説明
SQL	MySQL Driver	データベース接続に必要
SQL	MyBatis Framework	データベースとの連携を行うためのO/Rマッパー

pom.xmlを直接編集して依存関係を追加する場合の例

```
<dependency>
      <groupId>com.mysql</groupId>
      <artifactId>mysql-connector-j</artifactId>
      <scope>runtime</scope>
 </dependency>
<dependency>
    <groupId>org.mybatis.spring.boot</groupId>
    <artifactId>mybatis-spring-boot-starter</artifactId>
    <version>3.0.2</version>
</dependency>
```

© ZDRIVE, K.K. All rights reserved.

2. データベース/MyBatisの設定

- application.propertiesにデータベース接続やMyBatisについての設定を記述する

例：mydbを利用する際のapplication.propertiesの記述

```
# データベース接続設定
spring.datasource.url=jdbc:mysql://127.0.0.1:3306/mydb?useUni
code=true&characterEncoding=utf8&serverTimezone=Asia/Tokyo
spring.datasource.username=root
spring.datasource.password=

# MyBatisの設定
mybatis.mapper-locations=classpath*:/mybatis/**/*.xml
mybatis.configuration.map-underscore-to-camel-case=true
```

改行せずに１行で記述する

© ZDRIVE, K.K. All rights reserved.

2. データベース/MyBatisの設定

- データベース接続に関するプロパティの例
  - MyBatis以外のフレームワークを使う場合にも使用する

プロパティ名	説明
spring.datasource.url	接続するデータベースのURL
spring.datasource.username	データベース接続時のユーザー名
spring.datasource.password	データベース接続時のパスワード

© ZDRIVE, K.K. All rights reserved.

2. データベース/MyBatisの設定

- MyBatisに関するプロパティの例

プロパティ	説明
mapper-locations	SQL等を記したXMLファイルの配置場所
configuration.map-underscore-to-camel-case	テーブルのカラム名とドメインクラスのフィールド名を自動マッピングする際に、スネークケース⇔キャメルケースの自動変換を行うための設定
type-aliases-package	マッピング用XMLでクラス名を記述する際に、FQCNではなく、パッケージ名を省略した形式で記述できるようにするための設定

参考: https://qiita.com/kazuki43zoo/items/ea79e206d7c2e990e478

© ZDRIVE, K.K. All rights reserved.

3. インターフェースの作成

- データベース操作を行うためのメソッドを、インターフェースで定義する
  - @Mapperアノテーションを付与する
  - このインターフェースを実装するクラスは作成不要

例：com.example.mapper.MemberMapper.java

```
@Mapper
public interface MemberMapper {
  List<Member> selectAll();
  Member selectById(Integer id);
  void insert(Member member);
  void update(Member member);
  void deleteById(Integer id);
}
```

© ZDRIVE, K.K. All rights reserved.

4. マッピング用XMLの作成

- 先に作成したインターフェース内のメソッドに対応するSQLを記したXMLファイルを作成する
- 全体をmapperタグで囲む
  ⇒ namespace属性として、対応するインターフェースを指定する

例：MemberMapper.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.example.mapper.MemberMapper">

  <select>SELECT * FROM members</select>
  <insert>INSERT INTO members VALUES …</insert>
  <update>UPDATE members SET name = #{name}…</update>
  <delete>DELETE FROM members WHERE id = #{id}</delete>

</mapper>
```

namespace属性は、対応するインターフェースをFQCNで指定する

© ZDRIVE, K.K. All rights reserved.

4. マッピング用XMLの作成

- mapper内に、select, insert, update, deleteタグを記述し、その要素としてSQL文を記述する
  - 各タグのid属性を使い、インターフェースで定義したメソッドとの対応付けを行う

例：MemberMapper.xml

```
<mapper namespace="com.example.mapper.MemberMapper">
  <select id="selectAll" resultType="com.example.domain.Member">
    SELECT * FROM members
  </select>

  <insert id="insert" parameterType="com.example.domain.Member">
     INSERT INTO members (name, age, created)
     VALUES (#{name}, #{age}, NOW())
  </insert>
</mapper>
```

id属性は、インターフェースに定義したメソッド名を指定する

© ZDRIVE, K.K. All rights reserved.

5. マッパーの利用

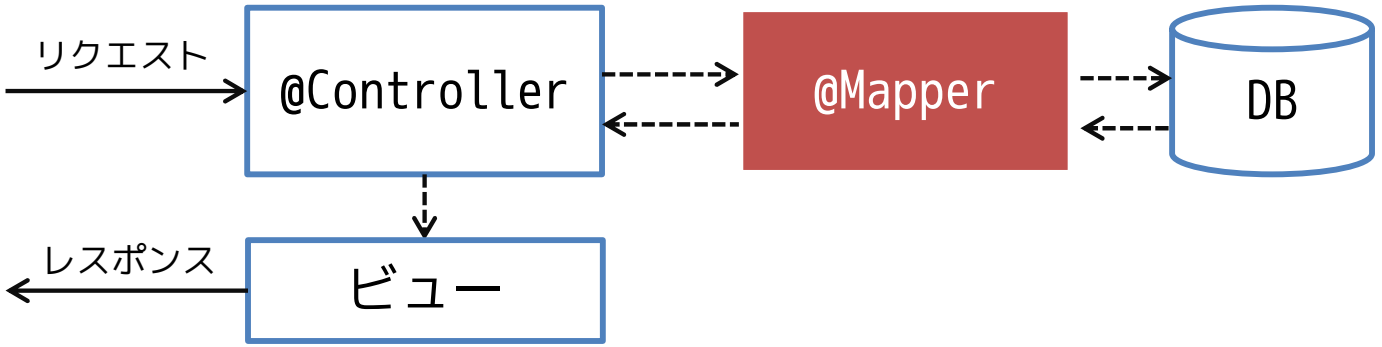
- 先に作成したマッパー(インターフェース)を利用する
  - インターフェースを実装するクラスの作成は不要

```
@Controller
@RequiredArgsConstructor
public class MemberController {

    private final MemberMapper mapper;

    @GetMapping("/members")
    public String showMemberList(Model model) {
        List<Member> memberList = mapper.selectAll();
        model.addAttribute("members", memberList);
        return "memberListPage";
    }

    ...
```

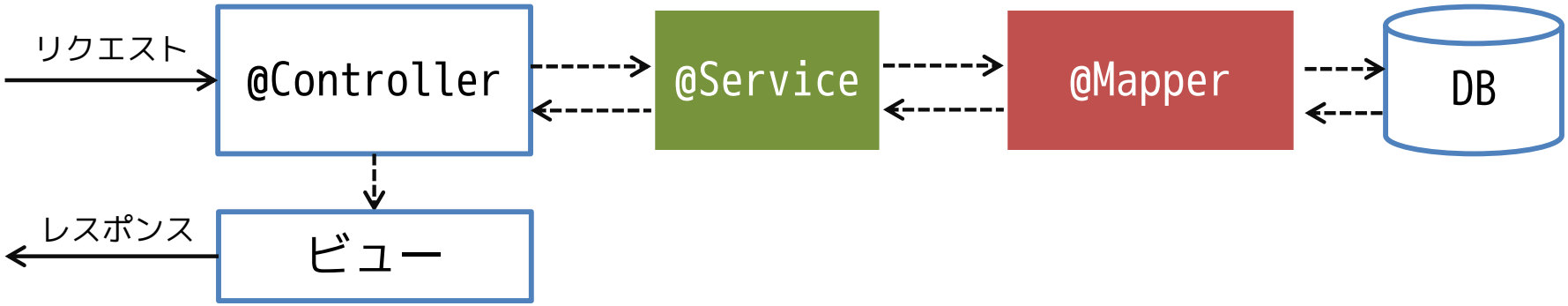
© ZDRIVE, K.K. All rights reserved.

5. マッパーの利用

- マッパーはコントローラーから利用できる



- コントローラーとマッパーの間にサービスクラスを介在させることもある



© ZDRIVE, K.K. All rights reserved.

練習

- 練習02

© ZDRIVE, K.K. All rights reserved.