フレームワーク応用実習

03. MyBatis 基礎

株式会社ジードライブ

今回学ぶこと

- マッパーXMLの記述
  - resultType属性等の型の指定
  - resultMap属性によるマッピング
  - MyBatisの設定
- トランザクション

select要素

- SELECT文を内包する

```
<select id="findAll" resultType="com.example.domain.Member">
  SELECT *, type_id AS typeId FROM members
</select>

<select id="findById" parameterType="int" resultMap="memberResult">
  SELECT * FROM members
  WHERE members.id = #{id}
</select>
```

select要素の属性

属性	値／説明
id	Javaから呼び出す際の識別子（インターフェースのメソッド名になる）
resultMap	取得されるデータを紐づけるresultMapのid属性値を指定
resultType	取得されるデータの型を指定（自動マッピング）
parameterType	Javaのメソッドを通じて利用される際の引数の型を指定 引数が複数になる場合は指定しない

どちらか一方を設定

resultType属性

- resultType属性を使い、取得したデータをドメインオブジェクトに変換する場合、取得データのカラム名とドメインクラスのフィールド名を合わせる必要がある

マッピングファイルの例：ASを使い、ドメインクラスのフィールド名に合わせる

```
<select id="findAll" resultType="com.example.domain.Member">
  SELECT id, name, type_id AS typeId FROM members
</select>
```

ドメインクラス(**com.example.domain.Member**)

```
public class Member {
      private Integer id;
      private String name;
      private Integer typeId
}
```

設定ファイルで、スネークケースとキャメルケースの自動変換を有効にすることも可能

resultMap要素

- SELECT文で取得されるデータとドメインクラスの関連を記述する
  - JOIN等を含む複雑なSELECT文と組み合わせて使用する
  - シンプルなSQLの場合、resultMap要素は必須ではない

resultMap要素の属性

属性	値／説明
id	selectタグのresultMap属性と紐づく識別子
type	対応するドメインクラスをパッケージ名を含む完全修飾クラス名(FQCN)で指定する

```
<resultMap id="memberResult" type="com.example.domain.Member">
  <id property="id" column="id" />
  <result property="name" column="name" />
  <result property="age" column="age" />
  <result property="typeId" column="type_id" />
  <result property="created" column="created" />
</resultMap>
```

resultMap要素

- テーブルの主キーは、id要素で関連を指定する
- その他のカラムは、 result要素で関連を指定する

id要素とresult要素の属性

属性	値／説明
property	ドメインクラスのフィールド名を指定する
column	propertyに対応するテーブルのカラム名を指定する SELECT文でASを使う場合は、変更後のカラム名を指定する

```
<resultMap id="memberResult" type="com.example.domain.Member">
  <id property="id" column="id" />
  <result property="name" column="name" />
  <result property="age" column="age" />
  <result property="typeId" column="type_id" />
  <result property="created" column="created" />
</resultMap>
```

ドメインのフィールド名

テーブルのカラム名

型の指定

- select要素のresultType属性やparameterType属性、resultMapのtype属性などでは、型をFQCNで指定する必要がある
- ただし、よく使われる型については、エイリアス(別名)が用意されており、短く記述することができる
  - エイリアスは大文字、小文字の区別がない
  - https://mybatis.org/mybatis-3/ja/configuration.html#typealiases

よく使われる型とエイリアス

型	エイリアス
java.lang.String	string
java.lang.Integer	int または integer
java.lang.Long	long
java.lang.Double	double
java.util.Date	date

練習

- 練習03-1

SQLの記述とバインド変数

- SQL文中ではバインド変数 #{変数名} を使用することができ、インターフェースで定義されたメソッドの引数と連携することができる
  - WHERE句、INSERT文のVALUE句、UPDATE文のSET句などで利用することができ、PreparedStatememtとして実行される
  - テーブル名やカラム名については、置換変数 ${変数名} が使えるが、SQLインジェクションに注意する必要がある

XML

```
<select id="findById" parameterType="int"
            resultType="com.example.domain.Member">
    SELECT * FROM members WHERE id = #{x}
</select>
```

バインド変数

インターフェース

```
Member findById(Integer id);
```

引数の型はparameterTypeと一致

連携する引数（引数名とバインド変数名は一致しなくてよい）

複数パラメーターの指定

- 複数のバインド変数がある場合、インターフェース側で**@Param("バインド変数名")**と記述することでバインド変数との紐づけを行うことができる

XML

```
<select id="getLimitedMembers"  resultType="com.example.domain.Member">
  SELECT * FROM members
  LIMIT #{offset}, #{count}
</select>
```

複数の引数を想定している場合、parameterType属性は設定しない

インターフェース

```
@Mapper
public interface MemberMapper {
  List<User> getLimitedMembers(@Param("offset") int offset,
                               @Param("count") int count);
}
```

不等号を含むSQLの記述

- XMLにおいて不等号はタグを構成する特別な記号のため、比較などの用途で、不等号をそのまま記述することはできない
- SQL文に不等号が含まれる場合、<![CDATA[…]]> で囲む必要がある
  - > や < を使用してエスケープする方法もある

```
<select id="getYoungerMembers" parameterType="int"
            resultType="com.example.domain.Member">
  <![CDATA[
  SELECT *, type_id AS "typeId" FROM members
  WHERE age <= #{age}
  ]]>
</select>
```

SQL文が不等号を含んでいるので<![CDATA[…]]> が必要

SQLの記述例

- COUNT()関数を含むSELECT文の記述例

```
<select id="countMembers" resultType="long">
    SELECT COUNT(*) FROM members
</select>
```

- WHERE句にLIKEを伴う場合の記述例
  - CONCAT()関数 を利用し、バインド変数の文字列結合を行う

```
<select id="search" parameterType="string"
        resultType="com.example.domain.Member">
    SELECT * FROM members
    WHERE name LIKE CONCAT('%', #{keyword}, '%')
</select>
```

複数件データの取得

- 複数件データの取得が想定されるSELECT文にマッピングされたメソッドは、戻り値の型をListにする
  - 戻り値の型をListにしないと、TooManyResultsException発生の可能性が生じる

XML

```
<select id="search" parameterType="string"
        resultType="com.example.domain.Member">
  SELECT * FROM members
  WHERE name LIKE CONCAT('%', #{keyword}, '%')
</select>
```

複数件データの取得が想定されるSELECT文

インターフェース

```
List<Member> search(String keyword);
```

resultTypeの型と一致させる

複数件のデータ取得が想定されるのでListにする

HashMapでのデータ取得

- @MapKeyアノテーションを利用することで、HashMapでデータを取得することができる

XML

```
<select id="getAverageAgeByAddress"
        resultType="com.example.domain.Member">
  SELECT address, AVG(age) AS age FROM members GROUP BY address;
</select>
```

インターフェース

```
@MapKey("address")
HashMap<String, Member> getAverageAgeByAddress();
```

Mapのキーとなるカラムを指定する

resultTypeの型と一致させる

addressの型と一致させる

練習

- 練習03-2

insert要素

- INSERT文を内包する

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

insert要素の属性

属性	値／説明
id	Javaから呼び出す際の識別子
parameterType	Javaのメソッドとして利用される際の引数の型を指定 ⇒ insertやupdateの場合、通常はドメインクラスを指定

update, delete要素

- UPDATE文／DELETE文を内包する

```
<update id="updateMember"
              parameterType="com.example.domain.Member">
  UPDATE members
  SET name = #{name}, age = #{age},
         type_id = #{typeId}, address = #{address}
  WHERE id = #{id}
</update>

<delete id="deleteMember" parameterType="int">
  DELETE FROM members WHERE id = #{id}
</delete>
```

parameterType属性

- insertやupdate要素で、parameterTypeにドメインクラスを指定する場合、バインド変数とドメインクラスのフィールド名が一致する必要がある

マッピングファイル

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

ドメインクラス（com.example.domain.Member）

```
public class Member {
  private Integer id;
  private String name;
  private Integer age;
  private Integer typeId;
  …
```

バインド変数のネスト

- バインド変数では、ドットを使い、階層関係を示すことができる

マッピングファイル

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

ドメインクラス（Member）

```
public class Member {
  private Integer id;
  private String name;
  private Integer age;
  private MemberType memberType;
  …
```

ドメインクラス（MemberType）

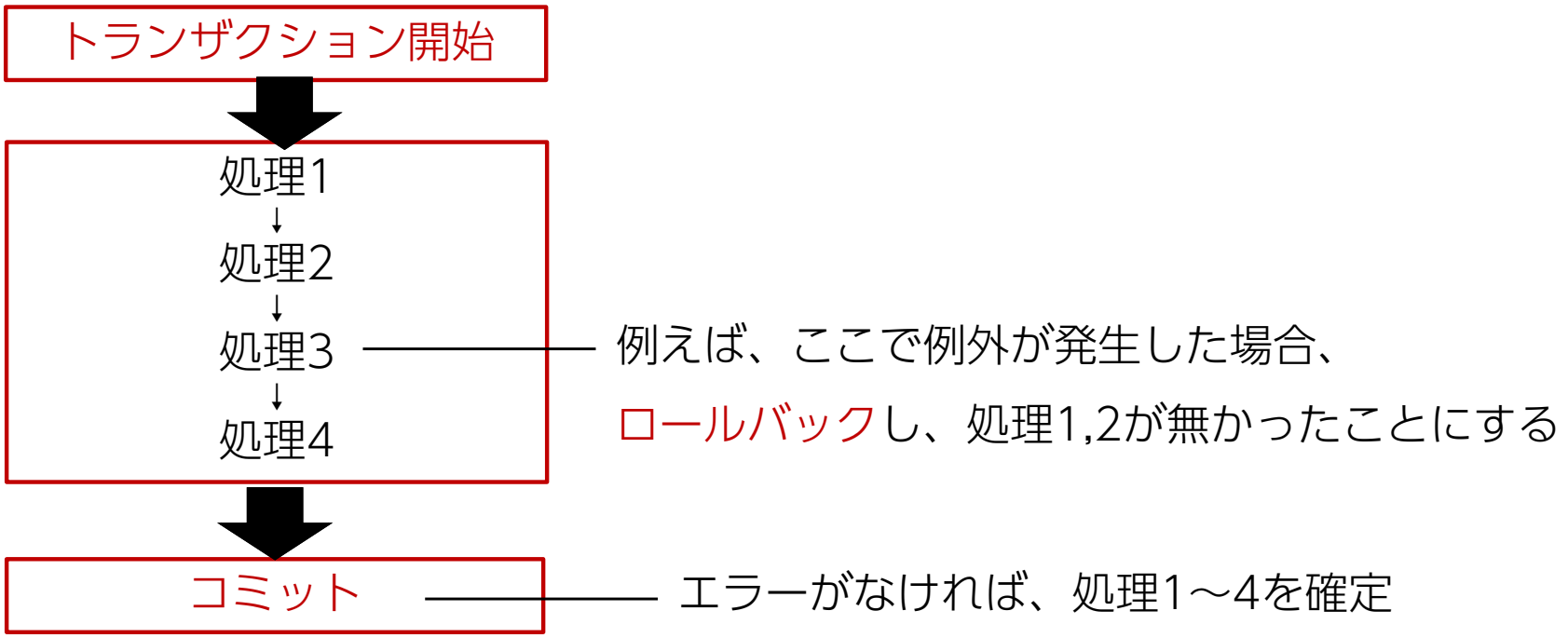
```
public class MemberType {
  private Integer id;
  private String name;
…
```

練習

- 練習03-3

トランザクション管理

- トランザクション：複数の処理をひとまとめにしたもの
  - トランザクション処理とは、この一連の処理の途中で例外が発生した場合、**ロールバック**することで、それまでの処理を無かったことにすること
  - Springでは、@Transactionalアノテーションで実装できる



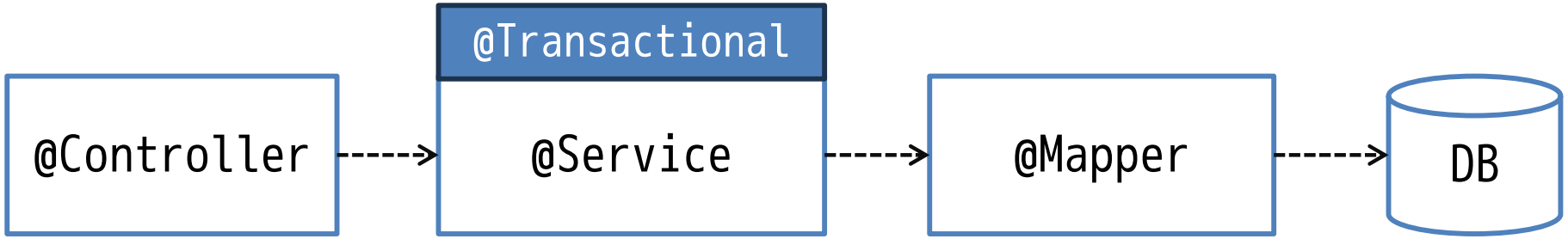
@Transactional

- @Transactional アノテーションは、クラスまたはメソッドに付与する
  - クラスに付与した場合、全メソッドに対して有効になる
  - ロールバック対応をする例外を指定することができる(指定がない場合、RuntimeExceptionに対応)

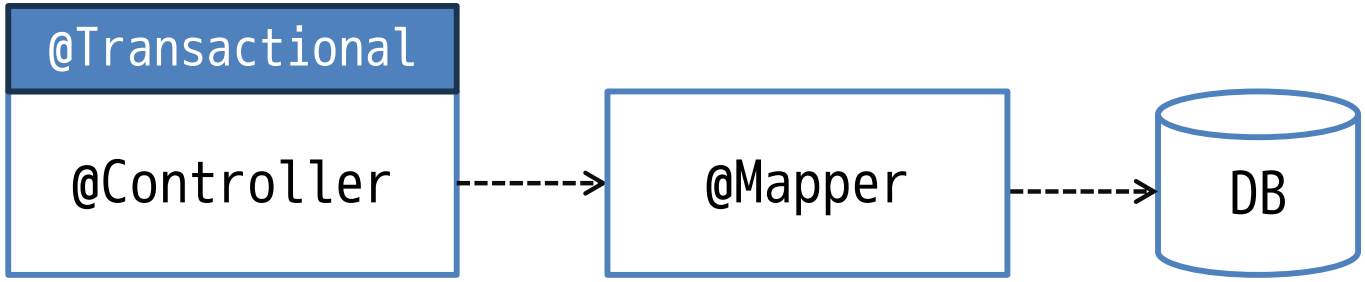
```
@Transactional(rollbackFor = Exception.class)
public void add(Member member, MemberDetail detail) {
      // 会員を追加⇒AutoIncrementで生成されたIDがセットされる
      memberMapper.insert(member);
      // 会員の詳細情報を追加
      detail.setId(member.getId());
      memberDetailMapper.insert(detail);
}
```

@Transactional

- @Transactionalは、基本的にサービスに付与する



  - サービスクラスがない場合、コントローラー内で使用する



簡略化のための設定

- @SpringBootApplicationが付与されたクラスに、@MapperScanアノテーションを追記することで、Mapperインターフェースに付ける@Mapperアノテーションを省略できる

```
@MapperScan("com.example.demo.mapper")
@SpringBootApplication
public class MyBatisSampleApplication {
    ...
```

- application.propertiesに、type-aliases-packageの設定を追記することで、resultTypeやparameterTypeの属性値からパッケージ名を省略できるようになる

```
mybatis.type-aliases-package=com.example.demo.domain
```

SQLの確認

- 実行されたSQLを確認するには、application.propertiesに以下を追記する

```
logging.level.com.example.demo.mapper.*=DEBUG
```

Mapperインターフェースを
配置しているパッケージ

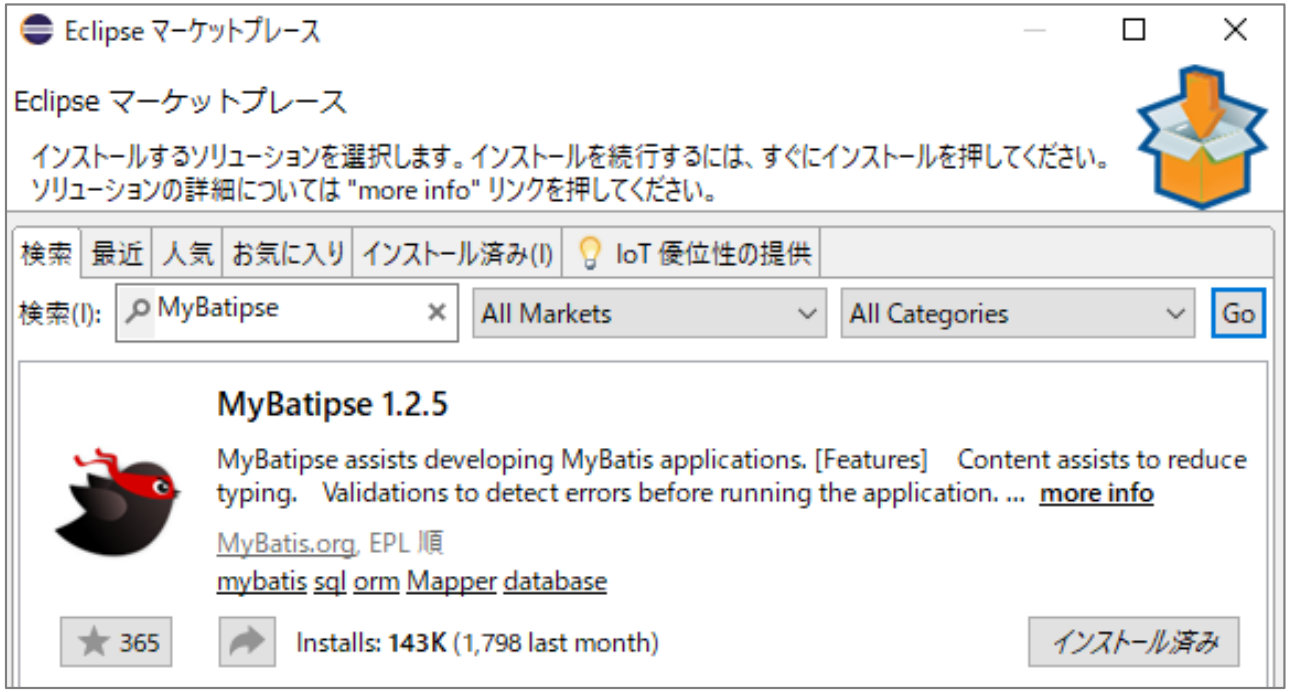
出力されるログの一例

```
MemberMapper.findByAge    : ==>  Preparing: SELECT * FROM members WHERE age < ?
MemberMapper.findByAge    : ==> Parameters: 27(Integer)
MemberMapper.findByAge    : <==      Total: 3
```

MyBatipse

- Eclipseマーケットプレースで入手可能なプラグイン
  - XMLとインターフェースの連携が容易になる
  - XMLの入力補助が強化される

    参考：https://qiita.com/YAKINIKU/items/9d7886a456e533555f80



練習

- 練習03-4
- 練習03-5
- 練習03-6
- 練習03-7