BL-06 データベースアクセス機能

■ 概要

◆ 機能概要

- TERASOLUNA Batch Framework for Java ver 3.5.0(以下、TERASOLUNA Batch)では、データベースへアクセスする機能として、MyBatis-Spring を使用して Spring Framework と連携した MyBatis3 を提供する。

- MyBatis3 の詳細は、ガイドラインの「5.3.1.1. MyBatis3 について」(http://terasolunaorg.github.io/guideline/5.0.x/ja/ArchitectureInDetail/DataAccessMyBatis3.html#dataaccessmybatis3overviewaboutmybatis3)を、MyBatis-Spring の詳細は、ガイドラインの「5.3.1.2. Mybatis3 と Spring の連携について」(http://terasolunaorg.github.io/guideline/5.0.x/ja/ArchitectureInDetail/DataAccessMyBatis3.html#mybatis3spring)を参照すること。

- 本書では、MyBatis3 から追加された Mapper インタフェースの仕組みを使用し、DAO インタフェースを経由してデータベースアクセスを行う方法について説明する。Mapper インタフェースの仕組みの詳細は、は、ガイドラインの「5.3.4.1. Mapper インタフェースの仕組みについて」(http://terasolunaorg.github.io/guideline/5.0.x/ja/ArchitectureInDetail/DataAccessMyBatis3.html#dataaccessmybatis3appendixaboutmappermechanism)を参照すること。

- ガイドラインを参照する際の注意事項

ガイドラインは TERASOLUNA Server Framework for Java 向けの記述となっているため、記述内容の読み替えが必要になる箇所がある。

- 業務処理を提供するクラスの読み替え

  TERASOLUNA Server Framework for Java では Service という名称を利用しているが、TERASOLUNA Batch では BLogic という名称を利用している。そのため、ガイドラインの Service は BLogic に読み替えること。

- データベースアクセスに関係するクラスの読み替え

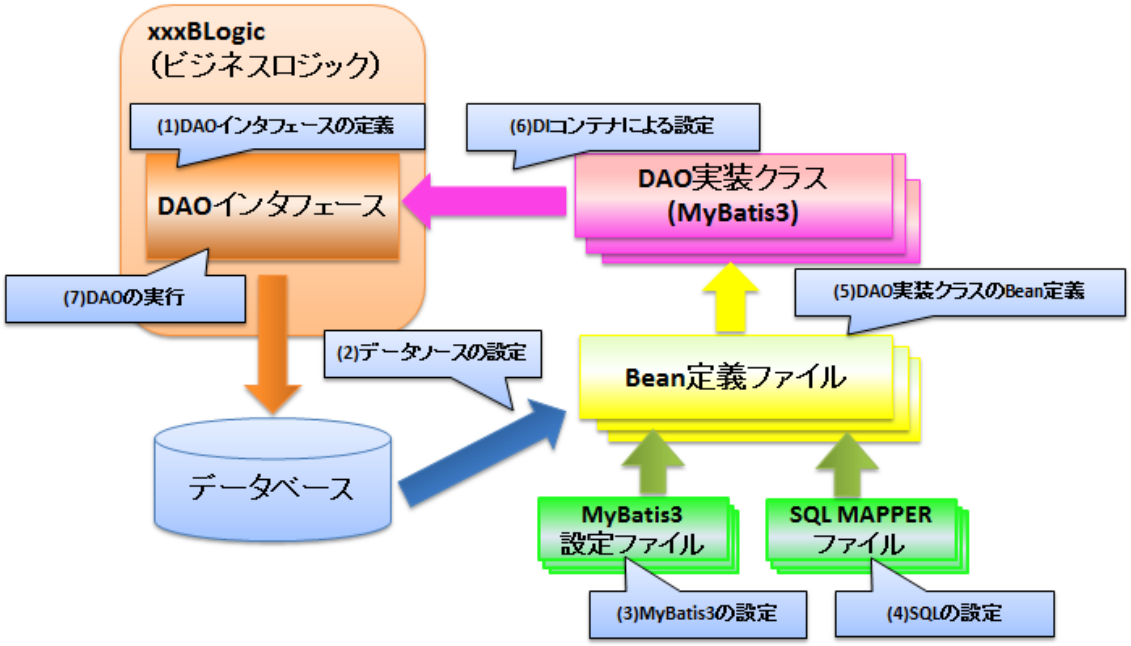
  TERASOLUNA Server Framework for Java では Entity と Repository を規定しているが、TERASOLUNA Batch では考え方の違い(※1)から、DTO と DAO を規定している。そのため、ガイドラインの Entity は DTO、Repository は DAO に読み替えること。

(※1)Entity/Repository と DTO/DAO の違い

Entity はデータベースのあるテーブルの 1 レコードを表現するクラスであり、Repository は Entity の問い合わせや、作成、更新、削除のような CRUD 処理を担うクラスである。一方、DTO はデータベースのあるテーブルの 1 レコードを表現するものに限らず、処理に必要なデータをまとめて表現するクラスであり、DAO はデータベースアクセスを担うクラスである。

バッチ処理では複数のテーブルを結合し、処理に必要な項目のみを抜き出すことが多いため、Entity/Repository の考え方にはそぐわないことが多い。バッチ処理で無理に Entity/Repository の考え方を採用すると、処理に必要なデータを取得するための SQL 発行回数が多くなってしまいがちなので、利用しないこと。

◆ 概念図



◆ 解説

- 概念図を構成する要素

	構成要素名	説明
1	DAO インタフェース	ビジネスロジックからデータベースアクセスを行うインタフェース。
2	DAO 実装クラス (MyBatis3)	DAO インタフェースに注入される、実際のデータベースアクセスを行うクラス。
3	Bean 定義ファイル	DAO インタフェースに DAO 実装クラス (MyBatis3)を注入する設定を定義するファイル。
4	SQL MAPPER ファイル	データベースアクセスを行う SQL を定義するファイル。
5	MyBatis3 設定ファイル	MyBatis3 の設定ファイル。
6	データベース	アクセス対象のデータベース。

- 必要な作業

(1) DAO インタフェースの定義

データベースアクセスを実行する DAO のインタフェースを定義する。データベースアクセスのパターンごとにメソッドを追加する。

(2) データソースの設定

アクセス対象のデータベースを Bean 定義ファイルにデータソースとして設定する。

(3) MyBatis3 の設定

DAO 実装クラスや、SQL の設定に影響する、MyBatis3 全体の設定を MyBatis3 設定ファイルに設定する。

(4) SQL の設定

(1)で定義した DAO インタフェースの定義をもとに、データベースアクセスに使用する SQL を SQL MAPPER ファイルに設定する。

(5) DAO 実装クラスの Bean 定義

(1)で定義した DAO インタフェースの実装となる DAO 実装クラスを DI コンテナで管理できるよう、Bean 定義ファイルに設定を追加する。

(6) DI コンテナによる設定

ビジネスロジックで宣言した DAO インタフェースのフィールドに DI コンテナで管理された DAO 実装クラスを注入する設定をビジネスロジックに追加する。

(7) DAO の実行

ビジネスロジックに設定された DAO インタフェースのメソッドを呼びだすことで、DAO 実装クラスが実際にデータベースへアクセスし、結果をオブジェクトにマッピングする。

■ 使用方法

◆ コーディングポイント

データベースアクセス機能を利用するにあたり、必要な各作業の項目単位にコーディングのポイントを説明する。

- (1) DAO インタフェースの定義

  ビジネスロジックからデータベースアクセスを実行するための DAO インタフェースを作成する。データベースへのアクセスパターンごとにメソッドを定義する。

➢ データを取得する例

データを取得する場合、メソッドは次のように宣言する。

✓ 戻り値

データベースアクセスの結果をマッピングする DTO クラスを指定する。結果が複数件になる場合は、DTO クラスの配列またはコレクションを指定する。

✓ 引数

SQL のパラメータ引数を格納する DTO クラスを指定する。

大量データを取得する場合は、引数に ResultHandler を追加することで、ResultHandler を使用しない場合と比べ、ヒープメモリの消費量を抑制することができる。

TERASOLUNA Batch では、入力データを取得する際は「AL-041 入力データ取得機能(コレクタ)」を使用することを推奨しているため、詳細は「AL-041 入力データ取得機能(コレクタ)」を参照のこと。

✓ DAO インタフェースの定義例(データの取得)

```
public interface SampleDao {

    /**
     * 全社員情報を取得する。
     * @param param SQL パラメータ引数オブジェクト
     */
    public List<EmployeeDataDto> findAllEmployee();

}
```

➢ データを挿入する例

データを挿入する場合、メソッドは次のように宣言する。

✓ 戻り値

int を指定することで、挿入件数を取得することができる。

✓ 引数

SQL のパラメータ引数を格納する DTO クラスを指定する。

✓ DAO インタフェースの定義例(データの挿入)

```
public interface SampleDao {

    /**
     * 社員情報を挿入する。
     * @param param SQL パラメータ引数オブジェクト
     */
    public int insertEmployee(InsertEmployeeDataInputDto param);

}
```

➢ データを更新する例

データを更新する場合、メソッドは次のように宣言する。

✓ 戻り値

int を指定することで、更新件数を取得することができる。

✓ 引数

SQL のパラメータ引数を格納する DTO クラスを指定する。

✓ DAO インタフェースの定義例(データの更新)

```
public interface SampleDao {

    /**
     * 社員情報を更新する。
     * @param param SQL パラメータ引数オブジェクト
     */
    public int updateEmployee(UpdateEmployeeDataInputDto param);

}
```

- データを削除する例

データを削除する場合、メソッドは次のように宣言する。

- ✓ 戻り値

  int を指定することで、削除件数を取得することができる。

- ✓ 引数

  SQL のパラメータ引数を格納する DTO クラスを指定する。

- ✓ DAO インタフェースの定義例(データの削除)

```
public interface SampleDao {

    /**
     * 社員情報を削除する。
     * @param param SQL パラメータ引数オブジェクト
     */
    public int deleteEmployee(DeleteEmployeeDataInputDto param);

}
```

- (2) データソースの設定

➢ データソースの Bean 定義

データベースアクセスに使用するデータソースを Bean 定義ファイルに定義する。フレームワークが利用する beansAdminDef/dataSource.xml と、各ジョブが利用する beansDef/dataSource.xml の両方に設定が必要となる。

✓ Bean 定義例(beansAdminDef/dataSource.xml)

```
<!-- DBCP のデータソースを設定する。 -->
<context:property-placeholder location="mybatisAdmin/jdbc.properties" />
<bean id="dataSource" destroy-method="close"
    class="org.apache.commons.dbcp.BasicDataSource">
    <property name="driverClassName" value="${jdbc.driver}" />
    <property name="url" value="${jdbc.url}" />
    <property name="username" value="${jdbc.username}" />
    <property name="password" value="${jdbc.password}" />
    <property name="maxActive" value="10" />
    <property name="maxIdle" value="1" />
    <property name="maxWait" value="5000" />
</bean>
```

設定値はプロパティファイルに切り離し、プレースホルダを利用して設定する。

✓ プロパティファイル例(myBatisAdmin/jdbc.properties)

```
#
# ジョブ管理テーブル DB 接続先
#
jdbc.driver=org.postgresql.Driver
jdbc.url=jdbc:postgresql://127.0.0.1:5432/postgres
jdbc.username=postgres
jdbc.password=postgres
```

- (3) MyBatis3 の設定

  - 実行モードの設定

    MyBatis3 では、DAO が SQL を実行する際の挙動を「実行モード」として、SIMPLE、REUSE、BATCH の 3 つから選択する。各実行モードの挙動の詳細は、ガイドラインの「5.3.2.3.1. SQL 実行モードの設定」(http://terasolunaorg.github.io/guideline/5.0.x/ja/ArchitectureInDetail/DataAccessMyBatis3.html#dataaccessmybatis3howtousesettingsexecutortype)を参照のこと。

    MyBatis3 標準は「SIMPLE」だが、設定ファイルの「defaultExecutorType」項目を明示的に指定することで、実行モードを指定することができる。[1]
    「defaultExecutorType」項目の指定についての詳細は、ガイドラインの「5.3.3.4.1. PreparedStatement 再利用モードの利用」(http://terasolunaorg.github.io/guideline/5.0.x/ja/ArchitectureInDetail/DataAccessMyBatis3.html#preparedstatement)を参照のこと。

  - TypeHandler の設定

    MyBatis3 標準でサポートされていない Java クラスとのマッピングが必要な場合や、MyBatis3 標準の振舞いを変更する必要がある場合は、独自の TypeHandler を作成してマッピングを行う必要がある。
    詳細は、ガイドラインの「5.3.2.3.4. TypeHandler の設定」(http://terasolunaorg.github.io/guideline/5.0.x/ja/ArchitectureInDetail/DataAccessMyBatis3.html#typehandler)を参照のこと。

  - TypeAlias の設定

    TypeAlias を設定すると、後述する SQL の設定で指定する SQL パラメータ引数オブジェクトのクラスや結果をマッピングするクラスに対して、エイリアス名(短縮名)を割り当てることができる。
    詳細は、ガイドラインの「5.3.2.3.2. TypeAlias の設定」(http://terasolunaorg.github.io/guideline/5.0.x/ja/ArchitectureInDetail/DataAccessMyBatis3.html#typealias)を参照のこと。

---

[1] 「defaultExecutorType」項目はすべての DAO インタフェースの実行モードに影響する。DAO インタフェースごとの指定方法は「(5)DAO 実装クラスの Bean 定義」を参照のこと。

➢ 実行ログの出力

DAO インタフェースのパッケージやフルパスに対してログ出力を有効にすると、実行されるステートメントのログが出力される。

✓ slf4j+Logback での設定例(logback.xml)

```
<configuration>
…
    <logger name="com.example.sample" level="INFO" />
    <logger name="com.example.sample.SampleDAO" level="INFO" />
    <logger name="com.example.sample.SampleDAO.findAllEmployee" level="INFO" />
…
</configuration>
```

パッケージやフルパスを指定してログを出力する。

フルパスに続けてメソッド名を指定することで、特定のステートメントのみのログを出力することができる。

- (4) SQL の設定

  - MappedStatement の定義

    定義した DAO インタフェースのパッケージとクラスパスのルートから見て同じ階層になるように、SQL MAPPER ファイルを作成する。SQL MAPPER のファイル名は、DAO インタフェースのクラス名とする。例では、「com/example/sample」ディレクトリを作成し、その中に「SampleDao.xml」を作成する。

    ✓ SQL MAPPER ファイルの定義例(com/example/sample/SampleDao.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.sample.SampleDao">
… SQL の設定 (省略) …
</mapper>
```

DAO インタフェースのフルパスを設定する。

SQL MAPPER ファイル内には、<mapper>タグの namespace 属性に設定、および、DAO インタフェースに定義したメソッドが実行する SQL の設定を行う。SQL の設定方法の詳細は、MyBatis3 のドキュメント「Mapper XML ファイル」(http://mybatis.github.io/mybatis-3/ja/sqlmap-xml.html)、「動的 SQL」(http://mybatis.github.io/mybatis-3/ja/dynamic-sql.html)を参照のこと。

- データを取得する例

select タグを使用して SQL を定義する。

✓ SQL_ID の設定

select タグの id 属性に、(1)DAO インタフェースの定義で定義したメソッド名と同じ文字列を指定する。

✓ データベースアクセス結果のマッピングの設定

resultType 属性に、データベースアクセスの結果 1 件を格納する DTO クラスを指定するとともに、取得したカラムの別名を DTO クラスのフィールド名とするか、resultMap 属性を使用し、マッピングを手動で定義するかのどちらかを選択する。

✓ SQL 引数パラメータの設定

parameterType 属性に、SQL 引数パラメータを格納する DTO クラスを指定する。この属性は省略できる。

✓ SQL MAPPER ファイルの定義例(データの取得(resultType 属性を使用))

```
<select id="findAllEmployee"
    resultType="com.example.sample.dto.EmployeeDataDto">
    SELECT
        EMPLOYEE_ID AS id
        ,EMPLOYEE_FAMILY_NAME AS familyName
        ,EMPLOYEE_FIRST_NAME AS firstName
        ,EMPLOYEE_AGE AS age
    FROM
        EMPLOYEE
</select>
```

EmployeeDataDto クラスのフィールド名と合わせる。

✓ SQL MAPPER ファイルの定義例(データの取得(resultMap 属性を使用))

```
<resultMap id="sampleResultMap"
    resultType="com.example.sample.dto.EmployeeDataDt
    <result property="id"
        column="EMPLOYEE_ID">
    <result property="familyName"
            column="EMPLOYEE_FAMILY_NAME">
        <result property="firstName"
            column="EMPLOYEE_FIRST_NAME">
        <result property="age"
            column="EMPLOYEE_AGE">
    </resultMap>
<select id="findAllEmployee" resultMap="sampleResultMap">
    SELECT
        EMPLOYEE_ID
        ,EMPLOYEE_FAMILY_NAME
        ,EMPLOYEE_FIRST_NAME
        ,EMPLOYEE_AGE
    FROM
        EMPLOYEE
</select>
```

ResultMap に ID を付与する。

resultMap タグを使用してマッピング方法を指定する。
result タグを使用し、property 属性に EmployeeDataDto クラスのフィールド名を指定し、column 属性にマッピングするカラム名を指定する。

resultMap 属性に、作成した ResultMap の ID を指定する。

- fetchSize 属性の指定について

大量のデータを返すようなクエリを記述する場合には、fetchSize 属性に適切な値を設定すること。 fetchSize 属性は、JDBC ドライバとデータベース間の通信において、一度の通信で取得するデータの件数を設定するパラメータである。fetchSize 属性を省略した場合は、JDBC ドライバのデフォルト値が利用されるため、デフォルト値が全件取得する JDBC ドライバの場合、メモリの枯渇の原因になる可能性があるので、 注意が必要となる。

- N+1 問題への対応

N+1 問題とは、一覧テーブルと明細テーブルからデータを取得する際に、一覧テーブルからデータを取得した後に、取得したデータ 1 件ごとに明細テーブルにアクセスするなど、レコード数に比例して実行する SQL の数が増えてしまうことにより、データベースへの負荷およびレスポンスタイムの劣化を引き起こす問題のことである。N+1 問題を回避する手段としては、関連するテーブルを結合することで 1 回の SQL で必要なデータを取得する方法がある。

N+1 問題の詳細は、ガイドラインの「5.1.4.1. N+1 問題の対策方法」(http://terasolunaorg.github.io/guideline/5.0.x/ja/ArchitectureInDetail/DataAccessCommon.html#n-1)を参照のこと。

MyBatis3 では、関連するテーブルを結合し、必要なデータを取得した際に、ResultMap による手動マッピングで collection タグを使用すると、1:N のマッピングを行うことができる。1:N のマッピング方法については、MyBatis3 のドキュメント「Mapper XML ファイル」(http://mybatis.github.io/mybatis-3/ja/sqlmap-xml.html)の「collection」節を参照のこと。

- データを挿入する例

insert タグを使用して SQL を定義する。

✓ SQL_ID の設定

insert タグの id 属性に、(1)DAO インタフェースの定義で定義したメソッド名と同じ文字列を指定する。

✓ 挿入件数の取得

挿入件数の取得を SQL MAPPER ファイルで考慮する必要はないため、resultType 属性は指定しない。

✓ SQL 引数パラメータの設定

parameterType 属性に、SQL 引数パラメータを格納する DTO クラスを指定する。この属性は省略できる。

✓ SQL MAPPER ファイルの定義例(データの挿入)

```
<insert id="insertEmployee"
    parameterType="com.example.sample.dto.InsertEmployeeDataInputDto">
    INSERT INTO
        EMPLOYEE (
            EMPLOYEE_ID
            ,EMPLOYEE_FAMILY_NAME
            ,EMPLOYEE_FIRST_NAME
            ,EMPLOYEE_AGE
        ) VALUES (
            #{id}
            ,#{familyName}
            ,#{firstName}
            ,#{age}
        )
</insert>
```

parameterType 属性に SQL パラメータ引数オブジェクトの型を指定する。

#{変数名}で SQL パラメータ引数オブジェクトのデータをバインドする。エスケープは自動的に実行される。

- データを更新する例

update タグを使用して SQL を定義する。

✓ SQL_ID の設定

update タグの id 属性に、(1)DAO インタフェースの定義で定義したメソッド名と同じ文字列を指定する。

✓ 更新件数の取得

更新件数の取得を SQL MAPPER ファイルで考慮する必要はないため、resultType 属性は指定しない。

✓ SQL 引数パラメータの設定

parameterType 属性に、SQL 引数パラメータを格納する DTO クラスを指定する。この属性は省略できる。

✓ SQL MAPPER ファイルの定義例

```
<update id="updateEmployee"
    parameterType="com.example.sample.dto.UpdateEmployeeDataInputDto">
    UPDATE
        EMPLOYEE
    SET
        EMPLOYEE_FAMILY_NAME = #{familyName}
    WHERE
        EMPLOYEE_ID = #{id}
</update>
```

parameterType 属性に SQL パラメータ引数オブジェクトの型を指定する。

#{変数名}で SQL パラメータ引数オブジェクトのデータをバインドする。エスケープは自動的に実行される。

- データを削除する例

delete タグを使用して SQL を定義する。

✓ SQL_ID の設定

delete タグの id 属性に、(1)DAO インタフェースの定義で定義したメソッド名と同じ文字列を指定する。

✓ 削除件数の取得

削除件数の取得を SQL MAPPER ファイルで考慮する必要はないため、resultType 属性は指定しない。

✓ SQL 引数パラメータの設定

parameterType 属性に、SQL 引数パラメータを格納する DTO クラスを指定する。この属性は省略できる。

✓ SQL MAPPER ファイルの定義例

```
<delete id="deleteEmployee"
    parameterType="com.example.sample.dto.DeleteEmployeeDataInputDto">
    DELETE FROM
        EMPLOYEE
    WHERE
        EMPLOYEE_ID = #{id}
</update>
```

parameterType 属性に SQL パラメータ引数オブジェクトの型を指定する。

#{変数名}で SQL パラメータ引数オブジェクトのデータをバインドする。エスケープは自動的に実行される。

- (5) DAO 実装クラスの Bean 定義

DAO インタフェースの実装となる DAO 実装クラス(MyBatis3)を DI コンテナで管理するには、MapperFactoryBean を使用する必要がある。MapperFactoryBean には、SqlSessionFactory の Bean 定義が必要になる。加えて、MyBatis3 の実行モードを DAO ごとに変更する必要がある場合は SqlSessionTemplate の Bean 定義が必要になる。

➢ SqlSessionFactory の Bean 定義

SqlSessionFactoryBean を使用して SqlSessionFactory を生成する Bean 定義を設定する。DAO が使用するデータソースと MyBatis3 設定ファイルの格納先を設定する必要がある。

✓ MyBatis3 設定ファイル

"configLocation"プロパティに、(3)MyBatis3 の設定で設定した MyBatis3 設定ファイルのパスを指定する。

✓ データソース

"dataSource"プロパティに、(2)データソースの設定で設定したデータソースの Bean 定義 ID を指定する。複数のデータソースを使用する場合は、データソースごとに SqlSessionFactory を分ける必要がある。

✓ Bean 定義例(beansDef / dataSource.xml)

```
<!-- SqlSessionFactory 定義 -->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="configLocation" value="mybatis/mybatis-config.xml" />
    <property name="dataSource" ref="dataSource" />
</bean>
```

➢ SqlSessionTemplate の Bean 定義

SqlSessionTemplate 自身のコンストラクタを使用して SqlSessionTemplate を生成する。生成には、SqlSessionFactory と実行モードを指定する文字列を設定する必要がある。

✓ SqlSessionFactory

SqlSession に SqlSessionTemplate の生成に使用する SqlSessionFactory の Bean 定義 ID を指定する。そのため、複数のデータソースを使用する場合は、SqlSessionFactory ごとに SqlSessionTemplate を分ける必要がある。

✓ 実行モードの設定

実行モードを「SIMPLE」「REUSE」「BATCH」から選択する。

✓ SqlSessionTemplate の定義例(beansDef/dataSource.xml)

```
<!-- SqlSessionTemplate 定義 -->
<bean id="sqlSessionTemplate" class="org.mybatis.spring.SqlSessionTemplate">
    <constructor-arg index="0" ref="sqlSessionFactory" />
    <constructor-arg index="1" value="SIMPLE" />
</bean>
```

➢ DAO 実装クラスの Bean 定義

MapperFactoryBean を SqlSessionFactoryBean を使用して DAO 実装クラス(MyBatis3)を生成する Bean 定義を設定する。DAO インタフェースと使用する SqlSessionFactory または SqlSessionTemplate を設定する必要がある。

✓ DAO インタフェース

(1)で定義した DAO インタフェースのフルパスを設定する。

✓ 使用する SqlSessionFactory

SqlSessionFactory の Bean 定義 ID を設定する。

✓ 使用する SqlSessionTemplate

SqlSessionTemplate の Bean 定義 ID を設定する。

✓ ジョブ個別 Bean 定義ファイルの定義例(SqlSessionTemplate を利用)

```
<!-- SMP000Dao 設定 -->
<bean class="org.mybatis.spring.mapper.MapperFactoryBean">
    <property name="mapperInterface" value="com.example.sample.SampleDao" />
    <property name="sqlSessionFactory" ref="sqlSessionFactory" />
</bean>
```

DAO インタフェースのフルパスを設定する。

✓ ジョブ個別 Bean 定義ファイルの定義例(SqlSessionTemplate を利用)

```
<!-- SMP000Dao 設定 -->
<bean class="org.mybatis.spring.mapper.MapperFactoryBean">
    <property name="mapperInterface"
value="jp.terasoluna.batch.tutorial.sample000.SMP000Dao" />
    <property name="sqlSessionTemplate" ref="sqlSessionTemplate" />
</bean>
```

選択した実行モードで生成された SqlSessionTemplate を設定する。

✓ Bean 定義ファイルの設定箇所について

DAO インタフェースの Bean 定義ファイルは、beansDef/dataSource.xml に設定せず、ジョブ個別 Bean 定義ファイル(beansDef/(ジョブ ID).xml)に設定することを推奨する。dataSource.xml に設定すると、ジョブが使用しない DAO を DI コンテナで管理することになるため、ジョブの起動に時間がかかる、消費するメモリ量が増えるといったデメリットがある。

- (6) DI コンテナによる設定

  - @Inject アノテーションを使用した依存性の注入
    ビジネスロジックに DAO インタフェース型のフィールドを宣言し、@Inject アノテーションを使用して依存性を注入する。

    - ビジネスロジック実装例

```
@Inject
SampleDao sampleDao = null;

    public int execute(BLogicParam arg0) {
        .....
        List<EmployeeDataDto> employeeList
            = sampleDao.findAllEmoloyee();
        .....
    }
```

sampleDao に DAO 実装クラスのインスタンスが設定される。データベースアクセスには sampleDao の各メソッドを使用する。

- (7)DAO の実行

  - データを取得する例
    (1)DAO インタフェースの定義で DAO インタフェースに定義した findAllEmployee メソッドを実行すると、MyBatis3 は(4)SQL の設定で SQL MAPPER ファイルに定義した SQL_ID「findAllEmployee」を利用してデータベースアクセスを実行し、戻り値を「EmployeeDataDto」のコレクションとして返却する。
    ビジネスロジックでは、DAO インタフェースの戻り値である「EmployeeDataDto」の List として受け取り、employeeList の各データに対する処理を実行する。

    - ビジネスロジック実装例

```
@Inject
SampleDAO sampleDao = null;

    public int execute(BLogicParam arg0) {
        .....
        List<EmployeeDataDto> employeeList
            = sampleDao.findAllEmoloyee();
        .....
    }
```

DAO インタフェースのメソッドを使用する。SQL パラメータ引数オブジェクトは不要なので、メソッドの引数はない。

➢ データを挿入する例

(1)DAO インタフェースの定義で DAO インタフェースに定義した insertEmployee メソッドを実行すると、MyBatis3 は(4)SQL の設定で SQL MAPPER ファイルに定義した SQL_ID「insertEmployee」を利用してデータベースアクセスを実行する。SQL 中の#{id}、#{firstName}、#{familyName}、#{age}には、InsertEmployeeDataInputDto 型オブジェクトに格納された変数 id、firstName、familyName、age の値がバインドされる。戻り値は挿入に成功したデータ件数となる。

✓ ビジネスロジック実装例

```
@Inject
SampleDAO sampleDao = null;

    public int execute(BLogicParam arg0) {
        .....
        InsertEmployeeDataInputDto dto
          = new InsertEmployeeDataInputDto();
        // 挿入する社員情報を設定
        dto.setId("xxx");
        dto.setFistName("yyy");
        .....
        int insertCount
              = sampleDao.insertEmoloyee(dto);
        .....
    }
```

SQL 引数パラメータオブジェクトを生成し、DAO のメソッドの引数に渡す。

➢ データを更新する例

(1)DAO インタフェースの定義で DAO インタフェースに定義した updateEmployee メソッドを実行すると、MyBatis3 は(4)SQL の設定で SQL MAPPER ファイルに定義した SQL_ID「updateEmployee」を利用してデータベースアクセスを実行する。SQL 中の #{id}、#{familyName} には、UpdateEmployeeDataInputDto 型オブジェクトに格納された変数 id、familyName の値がバインドされる。戻り値は更新に成功したデータ件数となる。

✓ ビジネスロジック実装例

```
@Inject
SampleDAO sampleDao = null;

    public int execute(BLogicParam arg0) {
        .....
        UpdateEmployeeDataInputDto dto
          = new UpdateEmployeeDataInputDto();
        // 更新対象の社員情報のIDを設定
        dto.setId("xxx");
        // 更新する社員情報を設定
        dto.setFamilyName("yyy");
        .....
        int updateCount
              = sampleDao.updateEmolovee(dto);
        .....
    }
```

SQL 引数パラメータオブジェクトを生成し、DAO のメソッドの引数に渡す。

➢ データを削除する例

(1)DAO インタフェースの定義で DAO インタフェースに定義した deleteEmployee メソッドを実行すると、MyBatis3 は(4)SQL の設定で SQL MAPPER ファイルに定義した SQL_ID「deleteEmployee」を利用してデータベースアクセスを実行する。SQL 中の#{id}には、DeleteEmployeeDataInputDto 型オブジェクトに格納された変数 id の値がバインドされる。戻り値は削除に成功したデータ件数となる。

✓ ビジネスロジック実装例

```
@Inject
SampleDAO sampleDao = null;

    public int execute(BLogicParam arg0) {
        .....
        DeleteEmployeeDataInputDto dto
          = new DeleteEmployeeDataInputDto();
        // 削除する社員情報のIDを設定
        dto.setId("xxx");
        .....
        int deleteCount
              = sampleDao.deleteEmoloyee(dto);
        .....
    }
```

SQL 引数パラメータオブジェクトを生成し、DAO のメソッドの引数に渡す。

➢ 実行モードに「BATCH」を選択した場合の注意点

(3) MyBatis3 の設定において、実行モードに「BATCH」を選択し、バッチ更新を行う場合、戻り値の取得について注意点が存在する。
詳細は、ガイドライン 5.3.3.4.3 .バッチモードの Repository 利用時の注意点(http://terasolunaorg.github.io/guideline/5.0.x/ja/ArchitectureInDetail/DataAccessMyBatis3.html#dataaccessmybatis3howtoextendexecutortypebatchnotes)を参照のこと。

◆ 拡張ポイント

なし

■ 関連機能

なし

■ 使用例

- 機能網羅サンプル(terasoluna-batch-functionsample)

- チュートリアル(terasoluna-batch-tutorial)

■ 備考

なし