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ための独自の機能を提供する代わりに、MyBatis3 の Mapper インタフェースを使用するデータベースアクセス機能を提供する。また、MyBatis3 と Spring Framework を連携するために、MyBatis-Spring を使用する。MyBatis3、MyBatis-Spring の汎用的な説明は「TERASOLUNA Server Framework (5.x) for Java Development Guideline」(以下、ガイドライン)を参照することとし、本書は TERASOLUNA Batch 向けの機能説明に特化して説明することとする。

- MyBatis3、Mapper インタフェース、MyBatis-Spring の詳細な説明は、ガイドラインの以下の項目を参照すること。

  - MyBatis3

「5.3.1.1. MyBatis3 について」
(http://terasolunaorg.github.io/guideline/5.0.0.RELEASE/ja/ArchitectureInDetail/DataAccessMyBatis3.html#dataaccessmybatis3overviewaboutmybatis3)

  - Mapper インタフェース

「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/5.0.0.RELEASE/ja/ArchitectureInDetail/DataAccessMyBatis3.html#dataaccessmybatis3appendixaboutmappermechanism)

  - MyBatis-Spring

「5.3.1.2. Mybatis3 と Spring の連携について」
(http://terasolunaorg.github.io/guideline/5.0.0.RELEASE/ja/ArchitectureInDetail/DataAccessMyBatis3.html#mybatis3spring)

- 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、データベースアクセスを伴う業務処理を Service クラスに実装しているが、TERASOLUNA Batch ではアーキテクチャの違いから、BLogic クラスに実装する。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、使用する際は注意すること。

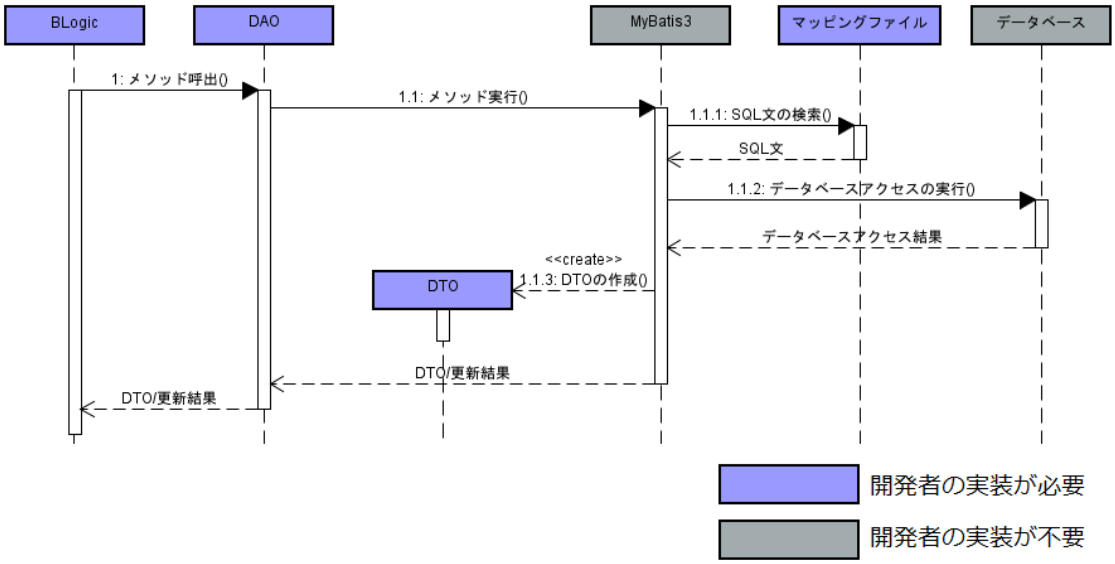
- MyBatis3 のドキュメントについて

  本書が参照する URL から"/ja"を取り除くと、英語のドキュメントを参照することができる。日本語訳ドキュメントに誤植がないとは言い切れないため、英語・日本語の両方を参照することを推奨する。

  日本語のドキュメント：http://mybatis.github.io/mybatis-3/ja/index.html

  英語のドキュメント：http://mybatis.github.io/mybatis-3/index.html

◆ 概念図



◆ 解説

- 概念図を構成する要素

	構成要素名	説明
i	BLogic	業務処理を行うクラス。データベースを行う際は DAO のメソッドを呼び出す。
ii	DAO	データベースアクセスを行うクラス。インタフェースを定義するだけで良く、実装クラスを作成する必要がない。
iii	DTO	データベースアクセスの結果を保持するクラス。String 型などを直接指定することも可能である。
iv	MyBatis3	DAO のメソッドが実行されると、対応する SQL 文を実行し、結果を DTO にマッピングし返却する。
v	マッピングファイル	データベースアクセスを行う SQL を定義する MyBatis3 の設定ファイル。
vi	MyBatis3 設定ファイル	MyBatis3 の動作をカスタマイズする設定ファイル。
vii	Bean 定義ファイル	データベースアクセスに関する定義を行う Spring Framework の Bean 定義ファイル。
viii	データベース	アクセス対象のデータベース。

- 処理の流れ

データベースアクセスの流れを以下に示す。

(1) メソッドの呼出
BLogic が DAO のメソッドを呼び出す。

(2) メソッドの実行
DAO は、MyBatis3 にデータベースアクセスの実行を委譲する。

(3) SQL 文の検索
MyBatis3 は、呼び出した DAO のフルパス、メソッド名に対応づけられている SQL を検索する。

(4) データベースアクセスの実行
MyBatis3 は、検索した SQL を使用してデータベースアクセスを実行する。

(5) DTO の作成
MyBatis3 は、データベースアクセスの結果をもとに DTO を作成する。

■ 使用方法

◆ コーディングポイント

- データソースの Bean 定義
- SqlSessionFactory の Bean 定義
- DAO の作成
  - 参照系
  - 更新系
- マッピングファイルの作成
  - マッピングファイルの配置
  - Namespace の設定
  - SQL 定義の追加
    - 参照系
    - 更新系
    - 動的 SQL について
- DAO の Bean 定義
- ビジネスロジックの実装
  - @Inject アノテーションを使用した依存性の注入
  - データベースアクセスの実行

- データソースの Bean 定義

  DAO がアクセスする対象となるデータソースの Bean 定義を追加する。TERASOLUNA Batch がアクセスするデータソースを設定する Bean 定義ファイル(ブランクプロジェクトでは、beansAdminDef ディレクトリ配下の AdminDataSource.xml)と、各ジョブがアクセスするデータソースを設定する Bean 定義ファイル(ブランクプロジェクトでは beansDef ディレクトリ配下の dataSource.xml)の両方に設定が必要となる。

  ✓ データソースの Bean 定義例(beansAdminDef/AdminDataSource.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.dbcp2.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="maxTotal" value="10" />
      <property name="maxIdle" value="1" />
      <property name="maxWaitMillis" value="5000" />
</bean>
```

設定値はプロパティファイルに切り離し、プレースホルダを使用して設定する。

  ✓ プロパティファイルの作成例(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
```

- SqlSessionFactory の Bean 定義

DAO の Bean 定義に必要な SqlSessionFactory の Bean 定義を追加する。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。

✓ データソース

"dataSource"プロパティに、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。

✓ MyBatis3 設定ファイル

"configLocation"プロパティに、MyBatis3 設定ファイルの格納先を指定する。MyBatis3 の設定をカスタマイズしない場合は、指定しなくても良い。

✓ SqlSessionFactory の 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>

<bean id="sqlSessionFactory2" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="configLocation" value="mybatis/mybatis-config.xml" />
    <property name="dataSource" ref="dataSource2" />
</bean>
```

- DAO の作成
  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 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宣言する。

  - 参照系

  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 を指定する。結果が複数件になる場合は、DTO の配列またはコレクションで指定する。

  ✓ 引数
  SQL のパラメータ引数を格納する DTO を指定する。全件取得の場合など、パラメータ引数がない場合は、引数を指定しなくて良い。

  ✓ DAO のメソッドの宣言例(参照系)

```
public interface SampleDao {

    String findPersonNameByReservationNo(FindReservationInputDto dto)

    ReservationDto findOneReservationDataByReserveDate(FindReservationInputDto dto);

    List<ReservationDto> findAllReservationData();

}
```

- 更新系

データを挿入・更新・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、挿入・更新・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 を指定する。パラメータ引数がない場合は、引数を指定しなくて良い。

✓ DAO のメソッドの宣言例(更新系)

```
public interface SampleDao {

    int insertReservationData(InsertReservationInputDto dto);

    int insertInitialData();

    int updateReservationData(UpdateReservationInputDto dto);

    int updateBussinessDate();

    int deleteReservationData(DeleteReservationInputDto dto);

    int deleteAllOldReservationData();

}
```

- マッピングファイルの作成

  - マッピングファイルの配置
    マッピングファイルは以下の構成で作成する。

    - ✓ 配置先
      DAO インタフェースが格納されているパッケージ階層と同一階層

    - ✓ ファイル名
      DAO インタフェース名 + ".xml"

  - namespace の設定
    マッピングファイル内のルート要素 mapper タグの namespace 属性には、DAO のフルパスを設定すること。

    ✓ マッピングファイルの作成例(com/example/sample/SampleDao.xml)

```
<mapper namespace="com.example.sample.SampleDao">
…(SQL の設定)…
</mapper>
```

  - SQL 定義の追加

    mapper タグの子要素として、DAO に宣言したメソッドごとに select タグ等を用いて SQL 定義を追加する。SQL 定義方法の詳細は、MyBatis3 のドキュメント「Mapper XML ファイル」(http://mybatis.github.io/mybatis-3/ja/sqlmap-xml.html)を参照のこと。

    - 参照系
      select タグを使用して SQL を定義する。

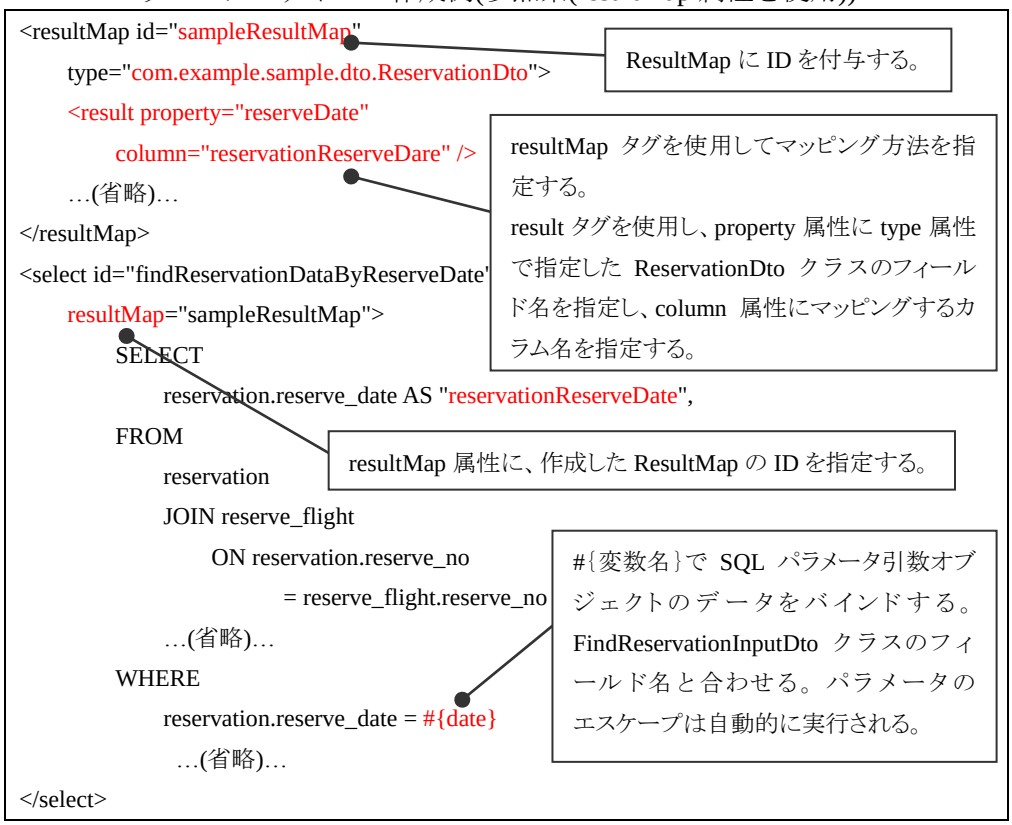
      ✓ SQL_ID の設定
      select タグの id 属性に、DAO に宣言したメソッド名と同じ文字列を指定する。

      ✓ ResultMap の作成と設定
      MyBatis3 がデータベースアクセス結果のマッピングに使用する ResultMap を、resultMap タグを使用して作成する。select タグの resultMap 属性に、作成した ResultMap の ID を指定する。
      取得したカラム名と DTO のプロパティ名が一致する場合は、resultType 属性に DTO を指定することで、ResultMap の作成と設定を省略できる。(SQL の AS 句を使用してカラム名に別名を付与し、DTO のプロパティ名と一致させてもよい)

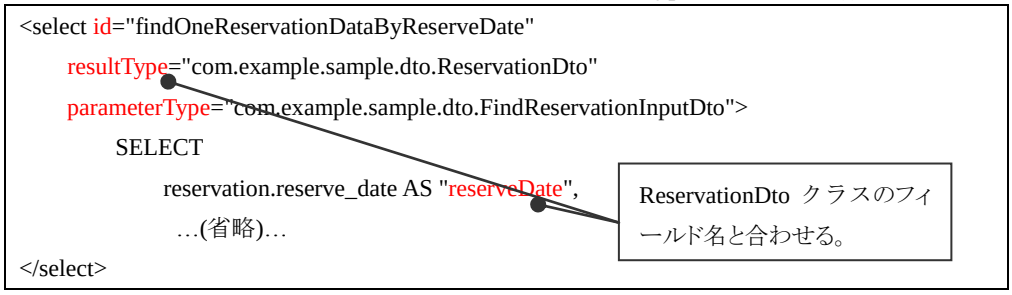
✓ SQL 引数パラメータの設定

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

✓ マッピングファイルの作成例(参照系(resultMap 属性を使用))



✓ マッピングファイルの作成例(参照系(resultType 属性を使用))



- 更新系

insert、update、delete タグを使用して SQL を定義する。

✓ SQL_ID の設定

insert、update、delete タグの id 属性に、DAO に宣言したメソッド名と同じ文字列を指定する。

✓ 更新結果の取得

件数の取得をマッピング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 を指定する。この属性は省略できる。

✓ マッピングファイルの作成例(更新系)

```
<insert id="insertReservationData"
    parameterType="com.example.sample.dto.InsertReservationInputDto">
    INSERT INTO RESERVATION (
        RESERVE_ID
        …(省略)…
    ) VALUES (
        #{id}
        …(省略)…
    )
</insert>
<update id="updateReservationData"
    parameterType="com.example.sample.dto.UpdateReservationInputDto">
    UPDATE RESERVATION SET
        RESERVE_DATE = #{newReserveDate}
        …(省略)…
    WHERE
        RESERVATION_ID = #{id}
</update>
<delete id="deleteReservationData"
    parameterType="com.example.sample.dto.DeleteReservationInputDto">
    DELETE FROM RESERVATION
    WHERE
        RESERVATION_ID = #{id}
</delete>
```

- 動的 SQL について

MyBatis3 では、SQL を動的に組み立てるための XML 要素と、OGNL ベースの式(Expression 言語)を使用することで、動的 SQL を組み立てる仕組みを提供している。詳細は、「動的 SQL」 (http://mybatis.github.io/mybatis-3/ja/dynamic-sql.html)を参照のこと。

- DAO の Bean 定義

DAO をビジネスロジックで使用するために Bean 定義を追加する。DAO のフルパスと、使用する SqlSessionFactory を設定する。

✓ DAO のフルパス

"mapperInterface"プロパティに DAO のフルパスを設定する。

✓ 使用する SqlSessionFactory の設定

"sqlSessionFactory"プロパティに SqlSessionFactory の Bean 定義 ID を設定する。

✓ DAO の Bean 定義例(ジョブ Bean 定義ファイル)

```
<!-- 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 のフルパスを設定する。

✓ Bean 定義ファイルの設定箇所について

DAO の Bean 定義は、各ジョブがアクセスするデータソースを設定する Bean 定義ファイル(ブランクプロジェクトでは 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ためである。

- ビジネスロジックの実装

  - @Inject アノテーションを使用した依存性の注入
    ビジネスロジックのフィールドに DAO 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。

  - データベースアクセスの実行
    DAO のフィールドに定義されたメソッドを実行する。SQL パラメータ引数がある場合は、メソッドを実行する前にパラメータ引数オブジェクトを作成し、メソッド実行時にメソッドの引数に与える。

    - 参照系
      データベースアクセスの結果をメソッドの戻り値として取得できるため、メソッドの戻り値を取得してデータベースアクセスの結果を処理する。

      ✓ ビジネスロジック実装例(参照系)

```
@Inject
SampleDao sampleDao;

    public int execute(BLogicParam arg0) {
        ...(省略)...
        FindReservationInputDto dto
          = new FindReservationInputDto();
        // 検索条件の設定
        dto.setId("xxx");
        ...(省略)...
        ReservationDto resavationDto =
            = sampleDao.findOneReservationDataByReserveDate(dto);
        ...(省略)...
    }
```

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。

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。

- 更新系

更新件数をメソッドの戻り値として取得できるため、メソッドの戻り値を取得して更新の妥当性チェックを行うことができる。

✓ ビジネスロジック実装例(更新系)

```
@Inject
SampleDao sampleDao;

    public int execute(BLogicParam arg0) {
        ...(省略)...
        UpdateReservationInputDto dto
          = new UpdateReservationInputDto();
        // 更新情報を設定
        dto.setId("xxx");
        ... (省略)...
        int updateCount
              = sampleDao.updateReservationData(dto);
        if(updateCount == 0) {
              // エラー処理
        }
        ...(省略)...
    }
```

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。

戻り値を使用した更新結果の判定も可能。

◆ 拡張ポイント

- MyBatis3 の設定

  「SqlSessionFactory の Bean 定義」で指定する MyBatis3 設定ファイル(ブランクプロジェクトでは mybatis/mybatis-config.xml、または、mybatisAdmin/mybatis-config.xml)内で設定可能な設定項目のうち、よく使う設定について、以下に示す。その他の設定項目については、MyBatis3 のドキュメント「設定」(http://mybatis.github.io/mybatis-3/ja/configuration.html)を参照のこと。

  - 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/5.0.0.RELEASE/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/5.0.0.RELEASE/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/5.0.0.RELEASE/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/5.0.0.RELEASE/ja/ArchitectureInDetail/DataAccessMyBatis3.html#typealias)を参照のこと。

---

[1] 「defaultExecutorType」項目はすべての DAO の実行モードに影響する。DAO ごとの指定方法は「DAO ごとの実行モードの変更」を参照のこと。

- DAO ごとの実行モードの変更

バッチ処理ではオンライン処理と異なり、大量のデータを挿入・更新・削除するケースが多々ある。そのような場合、使用する DAO の実行モードとして「BATCH」を選択すると、性能の向上が期待できる。個々の DAO ごとに実行モードを切り替える場合は、業務開発者が実施する DAO の Bean 定義の際に、使用する SqlSessionFactory を指定する代わりに、SqlSessionTemplate を指定する。

➢ 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。

✓ SqlSessionFactory

コンストラクタの第 1 引数(constructor-arg index="0")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

コンストラクタの第 2 引数(construnctor-arg index="1")に、「SIMPLE」「REUSE」「BATCH」から選択した実行モードを指定する。

✓ SqlSessionTemplate の Bean 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>
<bean id="batchSqlSessionTemplate" class="org.mybatis.spring.SqlSessionTemplate">
    <constructor-arg index="0" ref="sqlSessionFactory" />
    <constructor-arg index="1" value="BATCH" />
</bean>

<bean id="sqlSessionTemplate2" class="org.mybatis.spring.SqlSessionTemplate">
    <constructor-arg index="0" ref="sqlSessionFactory2" />
    <constructor-arg index="1" value="SIMPLE" />
</bean>
<bean id="batchSqlSessionTemplate2" class="org.mybatis.spring.SqlSessionTemplate">
    <constructor-arg index="0" ref="sqlSessionFactory2" />
    <constructor-arg index="1" value="BATCH" />
</bean>
```

➢ DAO の Bean 定義

"sqlSessionFactory"プロパティの代わりに、"sqlSessionTemplate"プロパティを使用する。

✓ DAO ごとの実行モードの変更例(beansDef/B000001.xml)

```
<!-- 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。

■ 関連機能

なし

■ 注意事項

◆ 大量データの取得について

- ResultHandler の使用

  大量のデータを返すようなクエリを記述する場合には、引数に ResultHandler を追加することで、ResultHandler を使用しない場合と比べ、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。

- fetchSize 属性の指定

  大量のデータを返すようなクエリを記述する場合には、select タグの 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使用されるため、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 問題への対応について

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し、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/5.0.0.RELEASE/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。

◆ 大量データの更新について

- UNDO 領域の増加による性能劣化の考慮

  1 トランザクション内で大量のデータを更新する場合、DB サーバの UNDO 領域が増加することにより性能が劣化する場合があるので、適切な単位でトランザクションを分割できないか検討すること。

◆ DAO の実行モードに「BATCH」を選択した場合について

- 戻り値の妥当性チェックについて

  バッチ更新を行う場合、DAO のメソッドからの戻り値は常に 0 件となる。戻り値の妥当性をチェックする必要がある場合は、実行モードに「BATCH」を選択してはならない。

  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/5.0.0.RELEASE/ja/ArchitectureInDetail/DataAccessMyBatis3.html#dataaccessmybatis3howtoextendexecutortypebatchnotes)を参照のこと。

- DAO のメソッドの呼出回数の考慮

  1 トランザクション内で DAO のメソッドが呼び出される回数に注意する必要がある。呼出回数に応じてメモリの消費量が増えていくため、メモリが枯渇する原因となる。この問題を回避する手段としては、適切な単位でトランザクションを分割する(バッチ更新を実行し、コミットする)方法や、適度な間隔で

SqlSession クラスの flushStatement メソッドを呼び出す(バッチ更新を実行するが、コミットはしない)方法がある。

トランザクションの分割については、BLogic の分割、または、BLogic 内でのプログラマティックなトランザクション制御で実現する。BLogic 内でのプログラマティックなトランザクション制御についての詳細は、トランザクション管理機能の機能説明書を参照のこと。SqlSession クラスの flushStatement メソッドを呼び出す例を、以下に示す。

✓ SqlSession クラスの flushStatements メソッドの使用例(ビジネスロジック)

```
@Inject
SampleDao sampleDao;

@Inject
@Named("batchSqlSessionTemplate")
SqlSession sqlSession;

    public int doMain(BLogicParam arg0) {
        ... (省略)...
        int updateCount = 0;
        for (UpdateReservationInputDto dto : updateReservationInputDtoList){
            sampleDao.updateReservationData(dto);
            updateCount++;
            if(updateCount % 1000 == 0) {
                // 1000 件単位にバッチ更新を実行する。
                sqlSession.flushStatements();
            }
        }
        ...(省略)...
    }
```

SampleDao の Bean 定義で使用した SqlSessionTemplate がインジェクションされるように@Named アノテーションに設定する。

例では、DAO のメソッドを呼び出した回数をカウントしておき、1000 件呼び出すごとにバッチ更新を実行している。コミットは行わない。

◆ 複数データソースの使用時について

複数のデータソースを使用する場合、データ不整合が発生しないように注意する必要がある。TERASOLUNA Batch では複数データソース全体の原子性を保証できないため、ビジネスロジック内で保証すること。

ビジネスロジック内で保証する手段としては、BLogic 内でのプログラマティックなトランザクション制御を行う方法と、BLogic の処理終了前に SqlSession クラスの flushStatements メソッドを使用してバッチ更新を実行し、バッチ更新の結果により、ロールバックが必要な場合に例外を発生させる方法の 2 つがある。BLogic 内でのプログラマティックなトランザクション制御についての詳細は、トランザクション管理機能の機能説明書を参照のこと。

■ 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

◆ ログの出力

MyBatis3 は Logback にログの出力を委譲する。そのため、Logback のロガー名、ログレベルを設定することでデータベースアクセスの実行ログを出力することができる。

- ロガー名の設定
  ロガー名と設定の有効範囲は以下の通り。

	ロガー名	設定の有効範囲
1	パッケージ	指定したパッケージに含まれるすべてのクラス ※DAO 以外のログ(例えば、BLogic)に対しても有効になる。
2	DAO のフルパス	指定した DAO
3	DAO のフルパス + "." + メソッド名	指定した DAO のメソッド

- ログレベルの設定
  ログレベルとログの出力内容は以下の通り。

	ログレベル	ログに出力される内容
1	ERROR	MyBatis3 はログを出力しない
2	WARN	
3	INFO	
4	DEBUG	実行されるステートメント、パラメータのログ
5	TRACE	実行した結果のログ

✓ slf4j+Logback での設定例(logback.xml)

```
<configuration>
…(省略)…
    <logger name="com.example.sample" level="INFO " />
    <logger name="com.example.sample.SampleDao" level="DEBUG" />
    <logger
        name="com.example.sample.SampleDao.findOneReservationDataByReserveDate"
        level="DEBUG" />
…(省略)…
</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もできる。

- MyBatis3 の logPrefix 属性の使用について

  データベースアクセスの実行ログを出力するためには、ログレベルを DEBUG または TRACE に設定する必要がある。データベースアクセスの実行ログを出力したいが、DAO と同一パッケージに存在する BLogic では DEBUG や TRACE ログを出力させたくない場合は、個々の DAO ごとにロガー名、ログレベルの設定をするよりも、次のように MyBatis3 の logPrefix 属性を使用するほうが、簡潔に定義できる。

  例として、com.example.sample パッケージに存在する BLogic では INFO レベルのログを出力し、com.example.sample パッケージに存在する DAO では TRACE レベルのログを出力する例を以下に示す。

  ✓ slf4j+Logback での設定例(logback.xml)

```
<configuration>
…(省略)…
    <logger name="MyBatis." level="TRACE" />
    <logger name="com.example.sample" level="INFO " />
…(省略)…
</configuration>
```

DAO のロガー名に、MyBatis3 設定ファイルで設定した接頭辞を付与する。

  ✓ logPrefix 属性の設定例(mybatis-config.xml)

```
<configuration>
    <settings>
        <setting name="logPrefix" value="MyBatis." />
    </settings>
</configuration>
```

◆ SQL 引数パラメータの与え方について

MyBatis3 では、SQL 引数パラメータを可変長引数で渡すことができる。その際の SQL パラメータ引数は、DAO のメソッドの宣言時に引数に付与する@Param アノテーションで指定した名称で取得する。

✓ SQL 引数パラメータに複数パラメータを与える例(ビジネスロジック)

```
@Inject
SampleDao sampleDao;

    public int execute(BLogicParam arg0) {
        ...(省略)...
        // 更新情報を設定
        int id = 123;
        String reserveDate = new Date();
        ... (省略)...
        int updateCount
            = sampleDao.updateReservationData(id, reserveDate, …(省略)… );
        if(updateCount == 0) {
            // エラー処理
        }
        ...(省略)...
    }
```

✓ SQL 引数パラメータに複数パラメータを与える例(DAO)

```
public interface SampleDao {

    ReservationDto findOneReservationDataByReserveDate(
        @Param("id") int id, @Param("reserveDate") Date reserveDate, …(省略)…);

}
```

✓ SQL 引数パラメータに複数パラメータを与える例(マッピングファイル)

```
<update id="updateReservationData"
    parameterType="com.example.sample.dto.UpdateReservationInputDto">
    UPDATE RESERVATION SET
        RESERVE_DATE = #{reserveDate}
        …(省略)…
    WHERE
        RESERVATION_ID = #{id}
</update>
```