AxiZ
Academy eXtend IT skill Zero

実践 Spring Framework + MyBatis

MyBatis-Spring Framework連携 編

ver. 1.2.1

TC. Techno-core Corporation 株式会社テクノコア
http://www.techno-core.jp

はじめに

MyBatis-Spring Framework連携

Spring Frameworkは、柔軟に他のフレームワークを組み合わせる事が可能です。

今回は、データアクセス部にMyBatisを使用し、Spring FrameworkのDIコンテナと連携するためのアプリケーション構築/実装方法を紹介します。

Spring FrameworkとMyBatisを連携させることで、Spring FrameworkはMyBatisのセッション(SqlSession)を読み込むようになり、また DIコンテナが MyBatisのデータMapperを管理下のオブジェクトにインジェクション(注入)することができるようになります。

そして、Spring Frameworkのトランザクション管理をMyBatisへ適用することもできます。

さらに、MyBatisの例外をSpring FrameworkのDataAccessExceptionへ委譲することも可能となります。

MyBatis-Spring連携ライブラリを使用して両フレームワークの結合を実現します(このライブラリの使用にはJava5以降が必須)。

☆今回MyBatis-Spring連携ライブラリは、以下を使用します。

mybatis-spring-1.2.0

開発環境構築

MyBatis-Spring連携ライブラリ ダウンロード

http://goo.gl/pwtO4 から mybatis-spring-1.2.0-bundle.zip をダウンロードします。

ダウンロードした.zipファイルを解凍すると、以下の構成で展開されます。
また、Spring Frameworkの2つのライブラリを新たに使用します。

【mybatis-spring-1.2.0-bundle.zip】

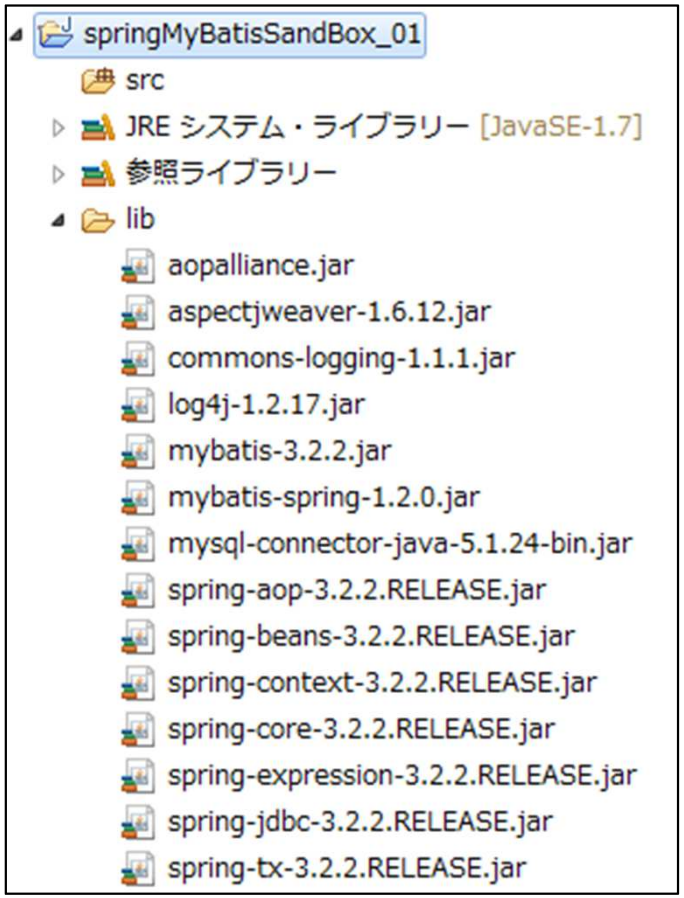
```
mybatis-spring-1.2.0
  |-- mybatis-spring-1.2.0.jar
  |-- mybatis-spring-1.2.0.tar.gz
  |-- mybatis-spring-1.2.0-sources.jar
  |-- mybatis-spring-1.2.0-sources.jar
  |-- LICENSE
  |-- NOTICE
```

【 spring-framework-3.2.2.RELEASE-dist.zip 】

```
spring-framework-3.2.2.RELEASE
  |-- libs
        |-- spring-jdbc-3.2.2.RELEASE.jar
        |-- spring-tx-3.2.2.RELEASE.jar
        |-- 略...
  |-- docs
  |-- schema
  |-- license.txt
  |-- notice.txt
  |-- readme.txt
```

開発環境構築

開発用プロジェクト作成



左図のパッケージ構成でプロジェクトを作成します。

・libフォルダには、以下の.jarファイルを格納してビルド･パスに追加します

☆3ページのライブラリの中から朱書きの.jarファイル

☆「MyBatis – Object/Relational Mapping 編」で使用した.jarファイル

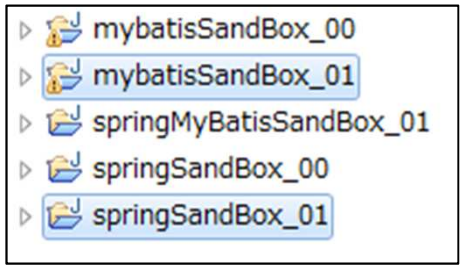
log4j-1.2.17.jar
mybatis-3.2.2.jar
mysql-connector-java-5.1.24-bin.jar

☆「Spring Framework – DIコンテナ/AOP 編」で使用した.jarファイル

aopalliance.jar
aspectjweaver-1.6.12.jar
commons-logging-1.1.1.jar
spring-aop-3.2.2.RELEASE.jar
spring-beans-3.2.2.RELEASE.jar
spring-context-3.2.2.RELEASE.jar
spring-core-3.2.2.RELEASE.jar
spring-expression-3.2.2.RELEASE.jar

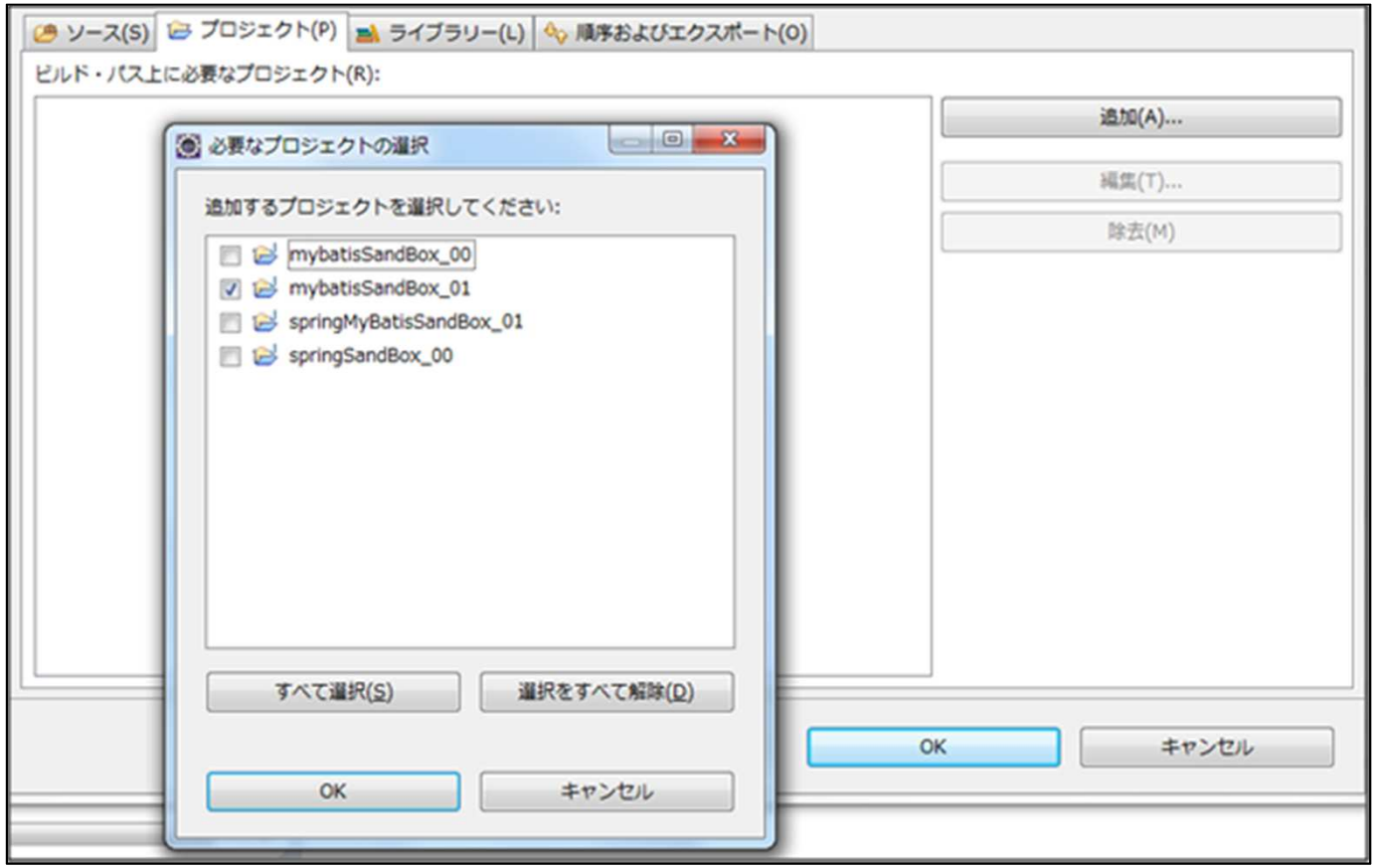
開発環境構築

参照用プロジェクト作成



MyBatis及びSpringで作成したプロジェクトを、連携用にコピーします（左図参照）。

また、springSandBox_01 から mybatisSandBox_01 のクラスを利用できるようにプロジェクトの参照設定を行います。



MyBatis-Spring Framework連携

ファイルの配置

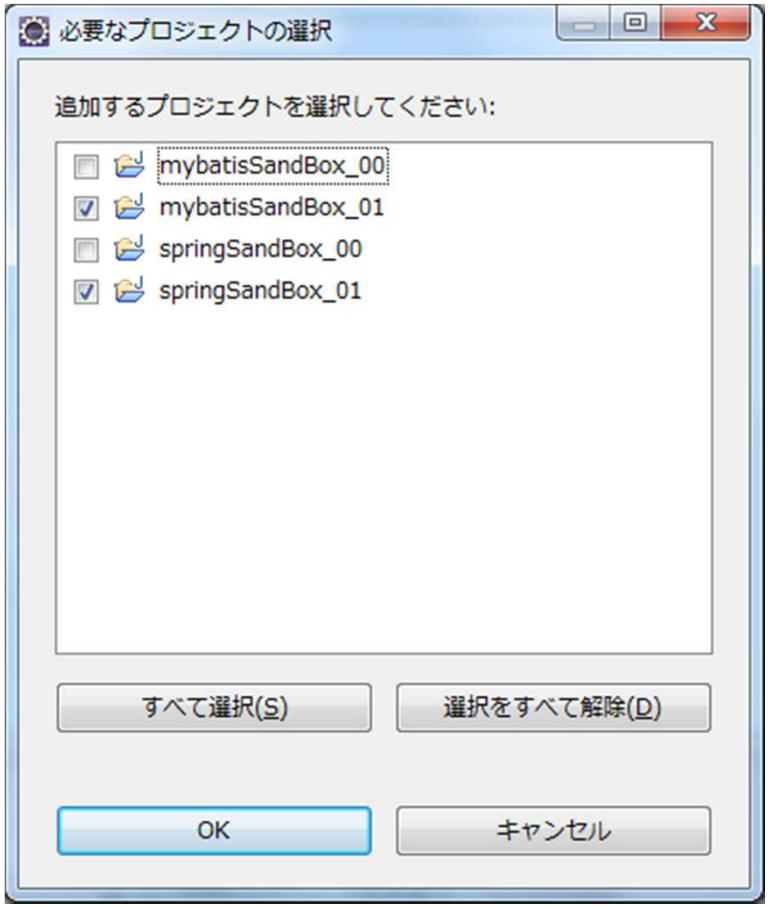
「MyBatis – Object/Relational Mapping 編」で実装したproperties/dataSource.properties

「Spring Framework – DIコンテナ/AOP 編」で実装した

beans.xml

log4j.xml

をspringMyBatisSandBox_01 にコピーし、springSandBox_01 と mybatisSandBox_01 のクラスを利用できるように、プロジェクトの参照設定をします。



MyBatis-Spring Framework連携

ファイルの配置

springSandBox_01 のDefaultUsrMstDaoは、DIコンテナが UsrMstMapper をインジェクション(注入)できるよう、右朱書きのように修正します。

【DefaultUsrMstDao.java】

```
package jp.co.axiz.app.dao.impl;

import jp.co.axiz.app.dao.UsrMstDao;
import jp.co.axiz.app.data.UsrMstMapper;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;

/**
 * {@link UsrMstDao}のデフォルト実装クラス。
 *
 * @author {@code AxiZ} t.matsumoto
 * @see UsrMstDao
 */
@Repository("usrMstDao")
public class DefaultUsrMstDao implements UsrMstDao {
/** {@link UsrMstMapper}オブジェクトを保持します。 */
    @Autowired
    private UsrMstMapper usrMstMapper;

    /**
     * 指定した条件に合致するレコード数を取得します。
     *
     * @param userId ユーザ{@code ID}
     * @param pwd ログインパスワード
     * @return レコード数
     * @see UsrMstDao#getCountByAccount(String, String)
     */
    @Override
    public int getCountByAccount(String userId, String pwd) {
        return usrMstMapper.getCountById(userId);
    }

}
```

MyBatis-Spring Framework連携

(Spring Framework)DIコンテナ設定ファイル

次にspringMyBatisSandBox_01の(Spring Framework)DIコンテナ設定ファイル(beans.xml)も編集します。

朱書きの定義が連携用に追記するものです。

【beans.xml】

```
<?xml version="1.0" encoding="UTF-8" ?>

<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
                           http://www.springframework.org/schema/context
                           http://www.springframework.org/schema/context/spring-context-3.2.xsd
                           http://www.springframework.org/schema/aop
                           http://www.springframework.org/schema/aop/spring-aop-3.2.xsd">

  <context:component-scan base-package="jp.co.axiz.app"/>
  <context:property-placeholder location="classpath:properties/dataSource.properties" local-override="true"/>

  <!-- ========================================================= DataSource -->

  <!-- MySQLデータソース -->
  <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName" value="${driver}" />
    <property name="url"             value="${url}" />
    <property name="username"        value="${username}" />
    <property name="password"        value="${password}" />
  </bean>
```

MyBatis-Spring Framework連携

(Spring Framework)DIコンテナ設定ファイル

```
 <!-- ========================================================= SessionFactory -->

  <!-- mybatis用SqlSeessionFactory -->
  <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
  </bean>

  <!-- ============================================================== AOP -->

  <!-- トレースログ用Advice -->
  <bean id="traceAdvice" class="jp.co.axiz.aop.TraceAdvice" />

  <!-- AOP Setting -->
  <aop:config>
    <aop:aspect id="traceAspect" ref="traceAdvice">
      <aop:pointcut id="tracePointcut" expression="execution(* isAllowLogin(..))" />
      <aop:around method="traceAround" pointcut-ref="tracePointcut" />
    </aop:aspect>
  </aop:config>

  <!-- ========================================================= MapperFactory -->

  <!-- ユーザマスタ(usr_mst)テーブルMapper -->
  <bean id="usrMstMapper" class="org.mybatis.spring.mapper.MapperFactoryBean">
    <property name="mapperInterface"   value="jp.co.axiz.app.data.UsrMstMapper" />
    <property name="sqlSessionFactory" ref="sqlSessionFactory" />
  </bean>

</beans>
```

*このbeans.xmlは、ログインサービス用の最小構成となっています

MyBatis-Spring Framework連携

動作確認

では、動作確認してみましょう。

「Spring Framework – DIコンテナ/AOP 編」で実装した LoginTest を使用します。

LoginTest.javaをspringMyBatisSandBox_01プロジェクトにコピーし、ログインサービスの動作確認してみてください。

ユーザIDによって正しいログイン判定が行われていれば成功です。

＊ UsrMstMapper#getCountById(String)は、現状では引数としてユーザIDしか受け取っていません

動作確認がとれたら、MyBatis設定ファイル(mybatis-config.xml)を削除して再度実行してみてください。

(1) mybatis-config.xml に定義していたプロパティファイルやデータソースは、Spring Framework設定ファイル(beans.xml)に移しました。

(2) データMapperオブジェクトの生成は MyBatis-Spring連携ライブラリの MapperFactoryBean を使用し、DIコンテナの管理下におくよう、(Spring Framework)DIコンテナ設定ファイル(beans.xml)に定義しました。

以上のことから、mybatis-config.xml は不要となります。

＊ ただし、今回のように Mapper I/F とマッピングファイルの名前を一致させ、かつ格納場所を同一ディレクトリとしておく必要があります

では、これまでの内容の確認の意味も含めて *UsrMstMapper#getCountById(String)* および関連するソースを修正し、ユーザ*ID*とパスワードでログイン可否を判定できるようにバグフィックスしましょう。

また、メソッド名も機能にふさわしい名前を考え、変更してください。

MyBatis-Spring Framework連携

Memo

今回のハンズオンの実装例では、MyBatis の Mapperインタフェースとマッピングファイルは一組しか作りませんでした。

しかし、実際の開発ではもっと多くのインターフェース・マッピングファイルが必要になります。

それぞれをbeanタグを使用してコンポーネント登録することももちろんできますが、自動検出させることで手間を軽減することができます。

いくつか方法がありますが、一番手軽なものを紹介しておきます。

beansタグにmybatis名前空間を追加し、mybatis:scanタグを使用します。

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:mybatis="http://mybatis.org/schema/mybatis-spring"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
                           http://mybatis.org/schema/mybatis-spring
                           http://mybatis.org/schema/mybatis-spring.xsd">

 <mybatis:scan base-package="jp.co.axiz.app.data" />
```

MyBatis-Spring Framework連携

トランザクション管理

DBを利用したアプリケーションでは、トランザクション管理が必要となる場合があります。
正規化されたテーブルに対して処理を行う際は、ほぼ必ずといっていいでしょう。

AOP的な観点で見れば、「トランザクション管理は横断的関心事である」と言えます。
そしてSpring Frameworkには、このトランザクション管理を統一的に実現する仕組みが用意されており、「宣言的トランザクション管理」と「プログラマティック･トランザクション管理」の2つに大別されます。

- 宣言的トランザクション管理 - 設定に基づきコンテナがトランザクション制御する
- プログラマティック･トランザクション管理 - APIを使用し、プログラムによってトランザクション制御する

Spring Frameworkでは、特別な場合を除き宣言的トランザクション管理の使用が推奨されており、この宣言モデルを使用することによって、トランザクション制御に関するコードを排除することができます。

今回は、Spring Frameworkの宣言的トランザクション管理を使用した方法で、トランザクション制御を実現する方法を紹介します。

MyBatis-Spring Framework連携

トランザクション管理(ユーザ情報新規登録処理)

では、「ユーザ情報新規登録処理(ユーザマスタ(usr_mst)テーブルとユーザ情報(usr_inf)テーブルへのINSERT)」を例にして Spring Frameworkの宣言的トランザクション管理の実装をしていきます。

まず、「ユーザ情報新規登録処理」に関する以下のモジュールが存在するとします。

- ユーザ情報サービス#ユーザ情報登録
- ユーザマスタ(usr_mst)DAO#ユーザマスタINSERT
- ユーザ情報(usr_inf)DAO#ユーザ情報INSERT

ユーザ情報サービスは、ユーザマスタ、ユーザ情報それぞれのDAOの insertメソッドを呼び出してユーザ情報を登録し、これを1トランザクションとして扱います。

【UserInfoService.java】

```
package jp.co.axiz.app.service;

import jp.co.axiz.app.entity.UsrDetail;

/**
 * ユーザ情報サービス･インタフェース。
 *
 * @author {@codeAxiZ} t.matsumoto
 */
public interface UserInfoService {

    /**
     * 指定した条件のユーザ情報を新規登録({code 1}件)し、登録処理結果を返却します。
     * 登録成功の場合は{@code true}を返却します。
     *
     * @param param ユーザ詳細情報格納{@code Entity}
     * @return 登録処理結果
     */
    boolean registerUserAccount(UsrDetail param);
}
```

MyBatis-Spring Framework連携

トランザクション管理(ユーザ情報新規登録処理)

【UsrInfDao.java】

```
import jp.co.axiz.app.entity.UsrDetail;

/**
 * ユーザ情報({@code usr_inf})テーブルアクセスのための{@code DAO}インタフェース。
 * @author {@code AxiZ} t.matsumoto
 */
public interface UsrInfDao {

    /**
     * 指定した条件のレコードを登録し、登録レコード数を返却します。
     * @param param ユーザ詳細情報{@code Entity}
     * @return 登録レコード数
     */
    int insert(UsrDetail param);
}
```

springSandBox_01 のUsrMstDao.javaにinsertメソッドを追加します。

【UsrMstDao.java】

```
    /**
     * 指定した条件のレコードを登録し、登録レコード数を返却します。
     *
     * @param param ユーザ詳細情報{@code Entity}
     * @return 登録レコード数
     */
    int insert(UsrDetail param);
```

MyBatis-Spring Framework連携

トランザクション管理の設定

次に、Spring Frameworkの宣言的トランザクション管理を利用するため、(Spring Framework)DIコンテナ設定ファイル(beans.xml)に定義を追加します。

なお、今回はアノテーションによる宣言的トランザクション制御を行います。

【beans.xml】

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
                           http://www.springframework.org/schema/context
                           http://www.springframework.org/schema/context/spring-context-3.2.xsd
                           http://www.springframework.org/schema/aop
                           http://www.springframework.org/schema/aop/spring-aop-3.2.xsd
                           http://www.springframework.org/schema/tx
                           http://www.springframework.org/schema/tx/spring-tx-3.2.xsd">

  <!-- ========================================== TransactionManager -->

  <!-- アノテーション･トランザクション管理有効 -->
  <tx:annotation-driven transaction-manager="transactionManager" />

  <bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource" />
  </bean>
```

MyBatis-Spring Framework連携

トランザクション制御対象メソッド(@Transactional)

最後に、Spring Frameworkにトランザクション制御を委譲するメソッドまたはクラスに対し、アノテーション@Transactional を付与します。

【DefaultUserInfoService.java】

```
package jp.co.axiz.app.service.impl;

/**
 * {@link UserInfoService}のデフォルト実装クラス。
 *
 * @author {@code Axiz} t.matsumoto
 * @see UserInfoService
 */
@Service("userInfoService")
@Transactional
public class DefaultUserInfoService implements UserInfoService {

    // 略.

    /**
     * 指定した条件のユーザ情報を新規登録({code 1}件)し、登録処理結果を返却します。<p />
     *
     * 登録成功の場合は{@code true}を返却します。
     *
     * @param param ユーザ詳細情報格納{@code Entity}
     * @return 登録処理結果
     * @see UserInfoService#registerUserAccount(UsrDetail)
     */
    @Override
    public boolean registerUserAccount(UsrDetail param) {
        // TODO: ユーザマスタテーブルとユーザ情報テーブルにデータを登録する.
        return false;
    }
}
```

MyBatis-Spring Framework連携

トランザクション制御対象メソッド(@Transactional)

今回のケースでは、対象メソッドは「ユーザ情報サービスのユーザ情報新規登録」メソッドになります。

なお、MyBatisからスローされた例外はorg.springframework.dao.DataAccessExceptionでキャッチできますが、@Transactionalを付与したメソッドの外側にスローするように実装してください。

トランザクション管理されているモジュール内で例外を握り潰すと、正しくトランザクション制御がされません。

例外発生時にログ出力する場合等、注意が必要です。

MyBatis-Spring Framework連携

Spring Frameworkによる宣言的トランザクション管理

DBを使用するアプリケーションにおいて、トランザクション制御は非常に重要で、かつ同じようなコードを冗長に実装することになりがちです。

しかし、Spring Frameworkの宣言的トランザクション管理機能を使用することで、(煩わしい)トランザクション制御に関するコードを記述する必要がなくなりました。

例として挙げたように、宣言的トランザクション管理を利用することを設定ファイルに定義し、あとはトランザクション制御対象メソッドにアノテーション @Transactional を付与するだけです。

＊ ただし、P.17 にも記述したとおり、例外のハンドリングにはくれぐれも注意が必要です。

では、実際に P. 13～16 の「ユーザ情報新規登録」モジュールと関連ソースを実装してみましょう。

また、テーブルのデータを直接編集し、意図的に(一意制約違反など)データ操作時の例外を発生させて、このトランザクションが正常に制御されていることを確認してみてください。