MyBatis Spring 1.0.0-RC2
参考文档

MyBatis 社区 (MyBatis.org)

本文档由南磊（nanlei1987@gmail.com）翻译

目录

第一章 介绍

1.1 整合动机

正如第二版，Spring 仅支持 iBatis2。那么我们就想将 MyBatis3 的支持加入到 Spring3.0（参考 Spring 的 Jira 的问题）中。不幸的是，Spring 3.0 的开发在 MyBatis 3.0 官方发布前就结束了。因为 Spring 开发团队不想发布一个基于非发行版的 MyBatis 的整合支持，那么 Spring 官方的支持就不得不等到至少 3.1 版本了。要在 Spring 中支持 MyBatis，MyBatis 社区认为现在应该是自己团结贡献者和有兴趣的人一起来开始进行 Spring 和 MyBatis 整合的时候了。

这个小类库就来创建丢失的粘贴 Spring 和 MyBtatis 这两个流行框架的胶水。减少用户不得不来配置 MyBatis 和 Spring 3.X 上下文环境的样板和冗余代码。

1.2 要求

在开始阅读本手册之前，很重要的一点是你要熟悉 Spring 和 MyBatis 这两个框架还有和它们有关的术语，否则可能会很难理解手册中的表述内容。

和 MyBatis 一样，MyBatis-Spring 也需要 Java 5 或更高版本。

1.3 感谢

非常感谢那些使得本项目成为现实的人们。Hunter Presnall和Putthibong Boonbong编写了所有的硬编码，Eduardo Macarron完成了MapperFactoryBean和文档，Andrius Juozapaitis，Giovanni Cuccu和Raj Nagappan的贡献和支持，而Simone Tripodi发现了这些人并把他们带入项目之中。没有他们的努力，这个项目是不可能存在的。

第二章 入门

MyBatis-Spring 帮助了你的 MyBatis 代码和 Spring 进行无缝整合。使用这个类库中的类，Spring 将会为你加载必要的 MyBatis 工厂和 session 类。这个小类库也会提供一个简便的方式向你的 service 层 bean 中注入 MyBatis 的数据映射器。最终，MyBatis-Spring 将会控制事务，翻译 MyBatis 异常到 Spring 的 DataAccessException（数据访问异常，译者注）。

2.1 安装

要使用 MyBatis-Spring 模块，你只需要包含 mybatis-spring-1.0.0-RC2.jar 文件，并在类路径中加入依赖关系。

如果你使用 Maven，那么在 pom.xml 中加入下面的代码即可：

```
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis-spring</artifactId>
    <version>1.0.0-RC2</version>
</dependency>
```

2.2 快速创建

要和 Spring 一起使用 MyBatis，你需要在 Spring 应用上下文中定义至少两样东西：一个 SqlSessionFactory 和至少一个数据映射器类。

在 MyBatis-Spring 中，SqlSessionFactoryBean 是用于创建 SqlSessionFactory.的。要配置这个工厂 bean，放置下面的代码在 Spring 的 XML 配置文件中：

```
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
</bean>
```

要注意 SqlSessionFactory 需要一个 DataSource.（数据源，译者注）。这可以是任意的 DataSource.，配置它就和配置其它 Spring 数据库连接一样。

假设你有一个如下编写的数据映射器类：

```
public interface UserMapper {
    @Select("SELECT * FROM user WHERE id = #{userId}")
    User getUser(@Param("userId") String userId);
}
```

那么可以使用，像下面这样来把接口加入到 Spring 中：

```
<bean id="userMapper" class="org.mybatis.spring.MapperFactoryBean">
  <property name="mapperInterface" value="org.mybatis.spring.sample.mapper.UserMapper" />
  <property name="sqlSessionFactory" ref="sqlSessionFactory" />
</bean>
```

要注意指定的映射器类必须是一个接口，而不是具体的实现类。在这个示例中，注解被用来指定 SQL 语句，但是 MyBatis 的映射器 XML 文件也可以用。

一旦配置好，你可以以注入其它任意 Spring 的 bean 相同的方式直接注入映射器到你的 business/service 对象中。`MapperFactoryBean` 控制 `SqlSession` 创建和关闭它。如果使用了 Spring 的事务，那么当事务完成时，session 将会提交或回滚。最终，任何异常都会被翻译成 Spring 的 `DataAccessException` 异常。

调用 MyBatis 数据方法下载只需一行代码：

```
User user = userMapper.getUser(userId);
```

第三章 SqlSessionFactoryBean

在基本的 MyBatis 中，session 工厂可以使用 SqlSessionFactoryBuilder.来创建。在 MyBatis-Spring 中，使用了 SqlSessionFactoryBean 来替代。

3.1 创建

要创建工厂 bean，放置下面的代码在 Spring 的 XML 配置文件中：

```
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
  <property name="dataSource" ref="dataSource" />
</bean>
```

要注意 SqlSessionFactoryBean 实现了 Spring 的 FactoryBean 接口（请参考 Spring 文档的 3.8 章节部分）。这就说明 bean 最终的创建不是 SqlSessionFactoryBean 本身完成的，但是工厂类 getObject()返回的方法的结果是基于那个类的。这种情况下，Spring 将会在应用启动时为你创建 SqlSessionFactory 对象，然后将它以 SqlSessionFactory 为名来存储。在 Java 中，相同的代码是：

```
SqlSessionFactoryBean factoryBean = new SqlSessionFactoryBean();
SqlSessionFactory sessionFactory = factoryBean.getObject();
```

在普通的 MyBatis-Spring 使用中，你不需要使用 SqlSessionFactoryBean 或直接和其对应的 SqlSessionFactory。而 session 工厂将会被注入到 MapperFactoryBean 中或其它扩展了 SqlSessionDaoSupport 的 DAO（Data Access Object，数据访问对象，译者注）中。

3.2 属性

SqlSessionFactory 有一个必须的属性，就是 JDBC 的 DataSource。这可以是任意的 DataSource，配置和其它 Spring 数据库连接是一样的。

一个通用的属性是 configLocation，它是用来指定 MyBatis 的 XML 配置文件路径的。如果基本的 MyBatis 配置需要改变，那么这就是一个需要它的地方。通常这会是<settings>或<typeAliases>部分。

要注意这个配置文件不需要是一个完整的 MyBatis 配置。确定地来讲，任意环境，数据源和 MyBatis 的事务管理器都会被忽略。SqlSessionFactoryBean 会创建它自己的，使用这些值定制 MyBatis 的 Environment 时是需要的。

如果 MyBatis 映射器 XML 文件在和映射器类相同的路径下不存在，那么另外一个需要配置文件的原因就是它了。使用这个配置，有两种选择。第一个是手动在 MyBatis 的 XML 配置文件中使用<mappers>部分来指定类路径。第二个是使用工厂 bean 的 mapperLocations 属性。

mapperLocations 属性一个资源位置的 list。这个属性可以用来指定 MyBatis 的 XML 映射器文件的位置。它的值可以包含 Ant 样式来加载一个目录中所有文件，或者从基路径下递归搜索所有路径。比如：

```
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
  <property name="dataSource" ref="dataSource" />
  <property name="mapperLocations" value="classpath*:sample/config/mappers/**/*.xml" />
</bean>
```

这会从类路径下加载在 sample.config.mappers 包和它的子包中所有的 MyBatis 映射器 XML 文件。

在容器环境管理事务中，一个可能需要的属性是 `transactionFactoryClass`。我们可以在第四章（事务）中来查看有关部分。

第四章 事务

一个使用 MyBatis-Spring 的主要原因是它允许 MyBatis 参与到 Spring 的事务中。而不是给 MyBatis 创建一个新的特定的事务管理器，MyBatis-Spring 利用了 Spring 中的 `DataSourceTransactionManager`。

一旦 Spring 的 PlatformTransactionManager 配置好了，你可以在 Spring 中以你通常的做法来配置事务。`@Transactional` 注解和 AOP（Aspect-Oriented Program，面向切面编程，译者注）样式的配置都是支持的。在事务期间，一个单独的 `SqlSession` 对象将会被创建和使用。当事务完成时，这个 session 会以合适的方式提交或回滚。

一旦事务创建之后，MyBatis-Spring 将会透明的管理事务。在你的 DAO 类中就不需要额外的代码了。

4.1 标准配置

要开启 Spring 的事务处理，在你的 Spring 的 XML 配置文件中简单创建一个 `DataSourceTransactionManager` 对象：

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

指定的 `DataSource` 可以是你通常使用 Spring 的任意 JDBC `DataSource`。这包含了连接池和通过 JNDI 查找获得的 `DataSource`。

要注意，为事务管理器指定的 `DataSource` 必须和用来创建 `SqlSessionFactoryBean` 的是同一个数据源，否则事务管理器就无法工作了。

4.2 容器管理事务

如果你正使用一个 JEE 容器而且想让 Spring 参与到容器管理事务（Container managed transactions，CMT，译者注）中，那么 Spring 应该使用 `JtaTransactionManager` 或它的容器指定的子类来配置。做这件事情的最方便的方式是用 Spring 的事务命名空间：

```
<tx:jta-transaction-manager />
```

在这种配置中，MyBatis 将会和其它由 CMT 配置的 Spring 事务资源一样。Spring 会自动使用任意存在的容器事务，在上面附加一个 `SqlSession`。如果没有开始事务，或者需要基于事务配置，Spring 会开启一个新的容器管理事务。

注意，如果你想使用 CMT，而不想使用 Spring 的事务管理，你就必须配置 `SqlSessionFactoryBean` 来使用基本的 MyBatis 的 `ManagedTransactionFactory`：

```
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="transactionFactoryClass"
        value="org.apache.ibatis.transaction.managed.ManagedTransactionFactory" />
</bean>
```

第五章 MapperFactoryBean

为了代替手工编写数据访问对象（DAO）的代码，MyBatis-Spring 提供了一个动态代理的实现：MapperFactoryBean.。这个类可以让你直接注入数据映射器接口到你的 service 层 bean 中。当使用映射器时，你仅仅如调用你的 DAO 一样调用它们就可以了，但是你不需要编写任何 DAO 实现的代码，因为 MyBatis-Spring 将会为你创建代理。

使用注入的映射器代码，在 MyBatis，Spring 或 MyBatis-Spring 上面不会有直接的依赖。MapperFactoryBean 创建的代理控制开放和关闭 session，翻译任意的 DataAccessException 异常到 Spring 的异常中。此外，如果需要或参与到一个已经存在对象中，代理将会开启一个新的 Spring 事务。

5.1 创建

数据映射器接口可以按照如下做法加入到 Spring 中：

```
<bean id="userMapper" class="org.mybatis.spring.MapperFactoryBean">
  <property name="mapperInterface" value="org.mybatis.spring.sample.mapper.UserMapper" />
  <property name="sqlSessionFactory" ref="sqlSessionFactory" />
</bean>
```

MapperFactoryBean 创建的代理类实现了 UserMapper 接口，并且注入到应用程序中。因为代理创建在运行时环境中（Runtime，译者注），那么指定的映射器必须是一个接口，而不是一个实现类。

如果 UserMapper 有一个对应的 MyBatis 的 XML 映射器文件，如果 XML 文件在类路径的位置和映射器类相同时，它会被 MapperFactoryBean 自动解析。没有必要在 MyBatis 配置文件中去指定映射器，除非映射器的 XML 文件在不同的类路径下。可以参考 SqlSessionFactoryBean 的 configLocation 属性来获取更多信息。

注意，当 MapperFactoryBean 需要 SqlSessionFactory 或 SqlSessionTemplate 时。这些可以通过各自的 SqlSessionFactory 或 SqlSessionTemplate 属性来设置，或者可以由 Spring 来自动装配。如果来年哥哥属性都设置了，那么 SqlSessionFactory 就会被忽略，因为 SqlSessionTemplate 是需要有一个 session 工厂的设置；那个工厂会由 MapperFactoryBean.来使用。

5.2 注入映射器

你可以在 business/service 对象中直接注入映射器，和注入 Spring 的 bean 是一样的：

```
<bean id="fooService" class="org.mybatis.spring.sample.mapper.FooServiceImpl">
   <property name="userMapper" ref="userMapper" />
</bean>
```

这个 bean 可以直接用在应用程序的逻辑中：

```
public class FooService {
    private UserMapper userMapper;
    ...
    public User doSomeBusinessStuff(String userId) {
        return this.userMapper.getUser(userId);
    }
}
```

注意，在这段代码中没有 SqlSession 或 MyBatis 的引用。也不需要去创建，打开或关闭 session。如果使用了 Spring 的事务，那么当事务完成时，session 将会自动被提交或者回滚。

5.3 自动配置

没有必要在 Spring 的 XML 配置文件中注册所有的映射器。相反，你可以使用一个 MapperScannerPostProcessor，它将会查找类路径下的映射器并自动创建它们。

要创建 MapperScannerPostProcessor，可以在 Spring 的配置中添加如下代码：

```
<bean class="org.mybatis.spring.annotation.MapperScannerPostProcessor">
   <property name="basePackage" value="org.mybatis.spring.sample.mapper" />
</bean>
```

basePackage 属性是让你为映射器接口文件设置基本的包路径。你可以使用分号或逗号作为分隔符设置多于一个的包路径。每个映射器将会在指定的包路径中递归地被搜索到。

注意，没有必要去指定 SqlSessionFactory 或 SqlSessionTemplate，因为 MapperScannerPostProcessor 将会创建 MapperFactoryBean，之后自动装配。但是，如果你使用了一个以上的 DataSource，那么自动装配可能会失效。这种情况下，你可以使用 sqlSessionFactoryBeanName 或 sqlSessionTemplateBeanName 属性来设置正确的 bean 名称来使用。这就是它是怎么配置的，注意 bean 的名称是必须的，而不是 bean 的引用，因此 value 属性要代替通常的 ref:属性：

```
<property name="sqlSessionFactoryBeanName" value="sqlSessionFactory" />
```

要开启 MapperScannerPostProcessor 来指定你的映射器类，它们必须由@Mapper 来注解。

```
@Mapper("userMapper")
public interface UserMapper {
    User getUser(String userId);
}
```

注解也让你指定了 bean 的名称。如果你不提供名字，那么它就以类名来注册。

第六章 SqlSessionTemplate 和 SqlSessionDaoSupport

6.1 SqlSessionTemplate

SqlSessionTemplate 是 MyBatis-Spring 的核心。这个类负责管理 MyBatis 的 SqlSession，调用 MyBatis 的 SQL 方法，翻译异常。它的作用就是无需替换 SqlSession。SqlSessionTemplate 通常应该用来代替 SqlSession，因为基本的 MyBatis 的 session 不能参与到 Spring 的事务中。同一个应用中两个类之间的转换可以引起数据完整性的问题。

当调用 SQL 方法时，包含从映射器 getMapper()方法返回的方法，SqlSessionTemplate 将会保证的 SqlSession 使用是和当前 Spring 的事务相关的。此外，它管理 session 的生命周期，包含必要的关闭，提交或回滚操作。

并不需要直接创建或使用 SqlSessionTemplate。很多情况下，MapperFactoryBean 内部使用一个模板，那就是所有需要的。当需要访问一个 SqlSessionTemplate 时，它可以使用 SqlSessionFactory 作为构造方法的参数来创建。

```
SqlSessionTemplate sessionTemplate = new SqlSessionTemplate(sqlSessionFactory);
```

相似地，模板可以在 Spring 的 XML 文件中配置。

此外，SqlSession 中和 SQL 相关的所有方法，SqlSessionTemplate 会提供一个通用的 execute 方法。这个方法需要定制的 SqlSessionCallback 作为参数，所以你可以在一个 SqlSession 中执行多个 SQL 方法：

```
public void insertUser(final User user) {
    getSqlSessionTemplate().execute(new SqlSessionCallback<Object>() {
        public Object doInSqlSession(SqlSession sqlSession) {
            sqlSession.insert("org.mybatis.spring.sample.mapper.UserMapper.insert
        User", user);
            sqlSession.insert("org.mybatis.spring.sample.mapper.UserMapper.insert
        Account", user.getAccount());
            return null;
        }
    });
}
```

注意 execute 方法可以接收 ExecutorType 参数。如果想要的执行方法和 SqlSessionFactory 的默认设置不同，那么这个可以用了。使用 ExecutorType.BATCH 对批量 SQL 查询的性能非常有用。对于这种方法的形式仅有一个警告，那就是当这个方法被调用时，不能有一个存在的事务和不同的 ExecutorType 一起运行。要么在独立的事务中使用 PROPAGATION_REQUIRES_NEW 或完全不再一个事务中，确保调用 execute(SqlSessionCallback, ExecutorType)来运行，

6.2 SqlSessionDaoSupport

SqlSessionDaoSupport 是一个抽象的支持类，用来为你构建 SqlSessionTemplate。调用 getSqlSessionTemplate()方法会给你模板的访问，之后可以用于执行 SQL 方法，就像下面这样：

```
public class UserMapperDaoImpl extends SqlSessionDaoSupport implements UserMapper {
    public User getUser(String userId) {
        return (User) getSqlSessionTemplate().selectOne("sample.UserMapper.getUser",
    userId);
    }
}
```

通常 MapperFactoryBean 是这个类的首选，因为它不需要额外的代码。但是，如果你需要在 DAO 中做其它非 MyBatis 的工作或需要具体的类，那么这个类就很有用了。

SqlSessionDaoSupport 的配置和 MapperFactoryBean 很相似。它需要设置 sqlSessionFactory 或 sqlSessionTemplate 属性。这些被明确地设置或由 Spring 来自动装配。如果两者都被设置了，那么 SqlSessionFactory 是被忽略的。

假设 UserMapperImpl 是 SqlSessionDaoSupport 的子类，它可以在 Spring 中进行如下的配置：

```
<bean id="userMapper" class="org.mybatis.spring.sample.mapper.UserMapperImpl">
    <property name="sqlSessionFactory" ref="sqlSessionFactory" />
</bean>
```

第七章 使用 MyBatis API

使用 MyBatis-Spring，你可以继续直接使用 MyBatis 的 API。仅仅在 Spring 中使用 SqlSessionFactoryBean 来创建一个 SqlSessionFactory，之后在你的代码中使用这个工厂。这种情况下你就不需要在代码中编写任何 MyBatis-Spring 的依赖了；仅在 DAO 中使用注入的 SqlSessionFactory 就行了。

```
public class UserMapperSqlSessionImpl implements UserMapper {
    // SqlSessionFactory会通常由SqlSessionDaoSupport来设置
    private SqlSessionFactory sqlSessionFactory;
    public void setSqlSessionFactory(SqlSessionFactory sqlSessionFactory) {
        this.sqlSessionFactory = sqlSessionFactory;
    }
    public User getUser(String userId) {
        // 注意标准的MyBatis API用法 - 手动打开和关闭session
        SqlSession session = sqlSessionFactory.openSession();
        try {
            return (User)
        session.selectOne("org.mybatis.spring.sample.mapper.UserMapper.getUser",
        userId);
        } finally {
            session.close();
        }
    }
}
```

注意这种用法不会参与到任何 Spring 的事务中。更进一步来说，如果 SqlSession 使用了 DataSource，它也被 Spring 的事务管理器使用了，还有当前过程中的事务，那么这段代码就会抛出异常。出于这个原因，如果你必须使用这种样式的代码，你就不应该使用 Spring 的事务数据库连接。

第八章 示例代码

你可以从 Google Code 的 MyBatis 资源库中检出示例代码。

- Java 代码
- 配置文件

要运行示例，仅在 JUnit 4 中执行 MyBatisSampleTest.java 即可。

这个示例代码有一个 service 接口，FooService，它作为一个被事务管理的 service。当它的任意方法被调用时就开启和结束一个事务。

```
@Transactional
public interface FooService {
    User doSomeBusinessStuff(String userId);
}
```

要注意，事务特性是由@Transactional 属性配置的。这不是必须的；任何其它由 Spring 提供的方法都可以用来标定你的事务。

FooServiceImpl.java 是 FooService 接口的实现，它仅仅使用了 Spring 启动时注入的 DAO/mapper。要注意代码不需要调用任何 Spring 或 MyBatis 的方法。

```
public class FooServiceImpl implements FooService {
    private UserMapper userMapper;
    public void setUserMapper(UserMapper userMapper) {
        this.userMapper = userMapper;
    }
    public User doSomeBusinessStuff(String userId) {
        return this.userMapper.getUser(userId);
    }
}
```

在这个手册中，数据库访问层已经实现了使用四种不同技术来解释：使用 MapperFactoryBean（可以直接或通过 MapperScannerPostProcessor），和使用了基本 MyBatis API 的 DAO 实现一起使用 SqlSessionDaoSupport。可以参考 applicationContext.xml 来获取完整的 Spring 配置信息。