

# MyBatis Spring 1.0.0

## 参考文档

MyBatis 社区 ([MyBatis.org](http://MyBatis.org))

Copyright © 2010

本文档的拷贝仅允许您个人使用或分发给其他用户，但是不能收取任何费用，后期的发布无论是印刷版或电子版，也会进行版权声明。

本文档由南磊 ([nanlei1987@gmail.com](mailto:nanlei1987@gmail.com)) 翻译

## 目录

第一章 介绍 .....	3
1.1 什么是 MyBatis-Spring? .....	3
1.2 整合动机 .....	3
1.3 要求 .....	3
1.4 感谢 .....	3
第二章 入门 .....	4
2.1 安装 .....	4
2.2 快速创建 .....	4
第三章 SqlSessionFactoryBean .....	6
3.1 创建 .....	6
3.2 属性 .....	6
第四章 事务 .....	8
4.1 标准配置 .....	8
4.2 容器管理事务 .....	8
4.3 编程式事务管理 .....	9
第五章 使用 SqlSession .....	10
5.1 SqlSessionTemplate .....	10
5.2 SqlSessionSupport .....	11
第六章 注入映射器 .....	13
6.1 MapperFactoryBean .....	13
6.2 MapperScannerConfigurer .....	14
第七章 使用 MyBatis API .....	15
第八章 示例代码 .....	16

# 第一章 介绍

## 1.1 什么是 MyBatis-Spring?

MyBatis-Spring 帮助你无缝地整合 MyBatis 代码到 Spring 中。使用这个类库中的类, Spring 将会加载必要的 MyBatis 工厂类和 session 类。这个类库也提供一个简单的方式来注入 MyBatis 数据映射器和 SqlSession 到业务层的 bean 中。而且它也会处理事务, 翻译 MyBatis 异常到 Spring 的 `DataAccessException` 异常 (数据访问异常, 译者注)。最终, 它不依赖于 MyBatis, Spring 或 MyBatis-Spring 来构建应用程序代码。

## 1.2 整合动机

正如第二版, Spring 3.0 仅支持 iBatis2。那么, 我们就想将 MyBatis3 的支持添加到 Spring 3.0 (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继续等待了。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将 Spring 的整合作为 MyBatis 社区的子项目的时候了。

## 1.3 要求

在开始使用 MyBatis-Spring 整合之前, 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, 本手册不会提供二者的背景内容, 基本安装和配置教程。

像 MyBatis 和 Spring 3.0 一样, MyBatis-Spring 也需要 Java 5 或更高版本。

## 1.4 感谢

特别感谢那些使得本项目成为现实的人们 (按字母顺序排序)。Eduardo Macarron, Hunter Presnall和Putthibong Boonbong的编码, 测试和文档修改工作; Andrius Juozapaitis, Giovanni Cuccu, Raj Nagappan和Tomas Pinos的贡献; 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

本章将会以简略的步骤告诉你如何安装和创建 MyBatis-Spring 并构建一个简单的数据访问事务性的应用程序。

### 2.1 安装

要使用 MyBatis-Spring 模块，你只需要包含 mybatis-spring-1.0.0.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</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 MapperFactoryBean，像下面这样来把接口加入到 Spring 中：

```
<bean id="userMapper" class="org.mybatis.spring.mapper.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，你可以用注入其它任意 Spring 的 bean 相同的方式直接注入映射器到你的 business/service 对象中。MapperFactoryBean 处理 SqlSession 的创建和关闭它。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 数据方法现在只需一行代码：

```
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);
    }
}
```

## 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）。这就说明由 Spring 最终创建的 bean 不是 `SqlSessionFactoryBean` 本身，而是工厂类的 `getObject()` 返回的方法的结果。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();
```

在一般的 MyBatis-Spring 用法中，你不需要直接使用 `SqlSessionFactoryBean` 或和它对应的 `SqlSessionFactory`。相反，session 工厂将会被注入到 `MapperFactoryBean` 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` 有一个单独的必须属性，就是 JDBC 的 `DataSource`。这可以是任意的 `DataSource`，其配置应该和其它 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>` 的部分。

要注意这个配置文件不需要是一个完整的 MyBatis 配置。确定地说，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。第二是使用工厂 bean 的 `mapperLocations` 属性。

`mapperLocations` 属性使用一个资源位置的 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`。请参考第四章（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 的事务管理中。而不是给 MyBatis 创建一个新的特定的事务管理器，MyBatis-Spring 利用了存在于 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。在事务处理期间，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 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 一般可以是你使用 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 而不是其它任意的 Spring 事务管理器：

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

## 4.3 程式事务管理

MyBatis 的 `SqlSession` 提供指定的方法来处理程式的事务。但是当使用 MyBatis-Spring 时, bean 将会使用 Spring 管理的 `SqlSession` 或映射器来注入。那就是说 Spring 通常是处理事务的。

你不能在 Spring 管理的 `SqlSession` 上调用 `SqlSession.commit()`, `SqlSession.rollback()` 或 `SqlSession.close()` 方法。如果这样做了, 就会抛出 `UnsupportedOperationException` 异常。注意在使用注入的映射器时不能访问那些方法。

无论连接是否设置为自动提交, `SqlSession` 数据方法的执行或在 Spring 事务之外任意调用映射器方法都将会自动被提交。

如果你想程式地控制事务, 请参考 Spring 手册的 10.6 章节。这段代码展示了如何手动使用在 10.6.2 章节描述的 `PlatformTransactionManager` 来处理事务。

```
DefaultTransactionDefinition def = new DefaultTransactionDefinition();
def.setPropagationBehavior(TransactionDefinition.PROPROPAGATION_REQUIRED);
TransactionStatus status = txManager.getTransaction(def);
try {
    userMapper.insertUser(user);
} catch (MyException ex) {
    txManager.rollback(status);
    throw ex;
}
txManager.commit(status);
```

注意这段代码展示了一个映射器, 但它也能和 `SqlSession` 一起使用。

# 第五章 使用 SqlSession

在 MyBatis 中，你可以使用 `SqlSessionFactory` 来创建 `SqlSession`。一旦你获得一个 `session` 之后，你可以使用它来执行映射语句，提交或回滚连接，最后，当不再需要它的时候，你可以关闭 `session`。使用 MyBatis-Spring 之后，你不再需要直接使用 `SqlSessionFactory` 了，因为你的 `bean` 可以通过一个线程安全的 `SqlSession` 来注入，基于 Spring 的事务配置来自动提交，回滚，关闭 `session`。

注意通常不必直接使用 `SqlSession`。在大多数情况下 `MapperFactoryBean`，将会在 `bean` 中注入所需要的映射器。下一章节中的 `MapperFactoryBean` 会解释这个细节。

## 5.1 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。`SqlSessionTemplate` 是线程安全的，可以被多个 DAO 所共享使用。

当调用 SQL 方法时，包含从映射器 `getMapper()` 方法返回的方法，`SqlSessionTemplate` 将会保证使用的 `SqlSession` 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。

`SqlSessionTemplate` 实现了 `SqlSession`，这就是说要对 MyBatis 的 `SqlSession` 进行简易替换。

`SqlSessionTemplate` 通常是被用来替代默认的 MyBatis 实现的 `DefaultSqlSession`，因为它不能参与到 Spring 的事务中也不能被注入，因为它是线程不安全的。相同应用程序中两个类之间的转换可能会引起数据一致性的问题。

`SqlSessionTemplate` 对象可以使用 `SqlSessionFactory` 作为构造方法的参数来创建。

```
<bean id="sqlSession" class="org.mybatis.spring.SqlSessionTemplate">
    <constructor-arg index="0" ref="sqlSessionFactory" />
</bean>
```

这个 `bean` 现在可以直接注入到 DAO `bean` 中。你需要在 `bean` 中添加一个 `SqlSession` 属性，就像下面的代码：

```
public class UserDaoImpl implements UserDao {
    private SqlSession sqlSession;

    public void setSqlSession(SqlSession sqlSession) {
        this.sqlSession = sqlSession;
    }

    public User getUser(String userId) {
        return (User) sqlSession.selectOne(
            ("org.mybatis.spring.sample.mapper.UserMapper.getUser", userId);
        );
    }
}
```

如下注入 `SqlSessionTemplate`：

```
<bean id="userDao" class="org.mybatis.spring.sample.dao.UserDaoImpl">
    <property name="sqlSession" ref="sqlSession" />
</bean>
```

SqlSessionTemplate 有一个使用 ExecutorType 作为参数的构造方法。这允许你用来创建对象，比如，一个批量 sqlSession，但是使用了下列 Spring 配置的 XML 文件：

```
<bean id="sqlSession" class="org.mybatis.spring.SqlSessionTemplate">
    <constructor-arg index="0" ref="sqlSessionFactory" />
    <constructor-arg index="1" value="BATCH" />
</bean>
```

现在你所有的语句可以批量操作了，下面的语句就可以在 DAO 中使用了。

```
public void insertUsers(User[] users) {
    sqlSession.insert("org.mybatis.spring.sample.mapper.UserMapper.insertUser", user);
}
}
```

注意，如果所需的执行方法和默认的 sqlSessionSessionFactory 设置不同，这种配置风格才能使用。

对这种形式需要说明的是当这个方法被调用时，不能有一个存在使用不同 ExecutorType 运行的事务。也要保证在不同的事务中，使用不同执行器来调用 sqlSessionTemplate 时，比如 PROPAGATION\_REQUIRES\_NEW 或完全在一个事务外面。

## 5.2 sqlSessionSupport

sqlSessionDaoSupport 是一个抽象的支持类，用来为你提供 sqlSession。调用 getSession() 方法你会得到一个 sqlSessionTemplate，之后可以用于执行 SQL 方法，就像下面这样：

```
public class UserDaoImpl extends sqlSessionDaoSupport implements UserDao {
    public User getUser(String userId) {
        return (User) getSession()
            .selectOne("org.mybatis.spring.sample.mapper.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 需要一个 sqlSessionFactory 或 sqlSessionTemplate 属性来设置。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。

假设类 UserDaoImpl 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>
```

## 第六章 注入映射器

为了代替手工使用 `SqlSessionDaoSupport` 或 `SqlSessionTemplate` 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`, 翻译任意的异常到 **Spring** 的 `DataAccessException` 异常中。此外, 如果需要或参与到一个已经存在活动事务中, 代理将会开启一个新的 **Spring** 事务。

### 6.1 MapperFactoryBean

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.mapper.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, 而不是一个具体的实现类。

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 (第三章) 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。如果两个属性都设置了, 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。

你可以直接在 `business/service` 对象中以和注入任意 **Spring bean** 的相同方式直接注入映射器:

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

这个 `bean` 可以直接在应用程序逻辑中使用:

```

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);
    }
}

```

注意在这段代码中没有 `SqlSession` 或 `MyBatis` 的引用。也没有任何需要创建，打开或关闭 `session` 的代码，`MyBatis-Spring` 会来关心它的。

## 6.2 MapperScannerConfigurer

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 `MapperScannerConfigurer`，它将会查找类路径下的映射器并自动将它们创建成 `MapperFactoryBeans`。

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

```

<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <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 `MapperScannerConfigurer` 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`（因此，也是多个的 `SqlSessionFactory`），那么自动装配可能会失效。这种情况下，你可以使用 `sqlSessionFactory` 或 `sqlSessionTemplate` 属性来设置正确的工厂/模板。

`MapperScannerConfigurer` 支持过滤由指定的创建接口或注解创建映射器。`annotationClass` 属性指定了要寻找的注解名称。`markerInterface` 属性指定了要寻找的父接口。如果两者都被指定了，加入到接口中的映射器会匹配两种标准。默认情况下，这两个属性都是 `null`，所以在基包中给定的所有接口可以作为映射器加载。

被发现的映射器将会使用 `Spring` 对自动侦测组件（参考 `Spring` 手册的 3.14.4）默认的命名策略来命名。也就是说，如果没有发现注解，它就会使用映射器的非大写的非完全限定类名。但是如果发现了 `@Component` 或 JSR-330 `@Named` 注解，它会获取名称。注意你可以配置 `annotationClass` 到 `org.springframework.stereotype.Component`，`javax.inject.Named`（如果你使用 JSE 6）或你自己的注解（肯定是自我注解）中，这样注解将会用作生成器和名称提供者。

## 第七章 使用 MyBatis API

使用 MyBatis-Spring, 你可以继续直接使用 MyBatis 的 API。仅仅在代码中使用 Spring 中的 `SqlSessionFactoryBean` 来创建一个 `SqlSessionFactory`。

```
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();
        }
    }
}
```

小心使用此选项, 因为错误的使用会产生运行时错误, 或者更糟糕的数据一致性的问题。这些是告诫:

- 它不会参与到 Spring 的事务之中。
- 如果 `SqlSession` 使用 `DataSource`, 它也会被 Spring 事务管理器使用, 而且当前有事务在进行时, 这段代码会抛出异常。
- MyBatis 默认的 `SqlSession` 是线程不安全的。如果在 `bean` 中注入了它, 就会发生错误。
- 你必须保证 `SqlSession` 用完就关闭了。

# 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](#)

任何示例都可以使用 JUnit 4 来运行。

这个示例代码展示了一个典型的事务服务层从数据访问层获取领域对象的设计。

这个 service 由 FooService.java 和 FooServiceImpl.java 实现类组成。这个 Service 是事务性的，所以当任何方法被调用和提交并没有以抛出非检查的异常而结束时，事务都会启动。

```
@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。

这个 service 使用 MyBatis 调用了一个数据访问层。这层由 MyBatis 的映射器接口 UserMapper.java 和由接口 UserDao.java 组成的 DAO，还有它对应的实现类 MapperImpl.java 组成。

在所有的示例代码中，映射器都被注入到 service bean 中，所以 service 代码可以如下编写：

```
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);
    }
}
```

这个手册中的数据库访问层使用一些不同的技术来实现。

表格 8.1 示例测试类

示例测试	描述
SampleBasicTest	展示了建议的，和简单的基于 MapperFactoryBean 的配置
SampleAutoTest	展示了基于组件扫描和自动装配的最小 xml 配置
SampleSqlSessionTest	展示了如何使用 Spring 管理的 SqlSession 来编写 DAO 代码
SampleBatchTest	展示了如何使用批量 SqlSession

请参看不同的 applicationContext.xml 文件来参考 MyBatis-Spring 不同的实战应用。