

# MyBatis Spring 1.0.0-SNAPSHOT

## 参考文档

MyBatis 社区 (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 为什么整合 MyBatis 和 Spring .....                          | 3 |
| 1.2 要求 .....  | 3 |
| 1.3 感谢 .....  | 3 |
| 第二章 入门 .....  | 4 |
| 2.1 介绍 .....  | 4 |
| 2.2 安装 .....  | 4 |
| 2.3 创建 SqlSessionFactory .....                            | 4 |
| 第三章 注入映射器 .....   | 5 |
| 3.1 注入映射器 .....   | 5 |
| 第四章 使用 SqlSessionDaoTemplate 和 SqlSessionDaoSupport ..... | 6 |
| 4.1 SqlSessionDaoTemplate .....                           | 6 |
| 4.2 SqlSessionDaoSupport .....                            | 6 |
| 第五章 使用 MyBatis API .....                                  | 7 |
| 5.1 使用 MyBatis API .....                                  | 7 |
| 第六章 示例代码 .....  | 8 |
| 6.1 示例代码 .....  | 8 |

# 第一章 介绍

## 1.1 为什么整合 MyBatis 和 Spring

每个 Spring 的用户都曾不耐烦地等待 3.X 版本的发布，但不幸的是，这让 MyBatis 用户相当失望：发布时没有对 MyBatis 这一优秀的 SQL 映射框架提供支持。在 Spring 的 JIRA 问题列表中，很多要求整合和发布补丁的请求，虽然被接受但是也被延迟处理了。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。

所以这个小类库就来创建丢失的粘贴 Spring 和 MyBatis 这两个流行框架的胶水。减少用户不得不开配置 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

### 2.1 介绍

MyBatis-Spring 的整合帮助你使代码和 Spring 无缝对接。Spring 就会为你加载创建必须的 MyBatis 类, 将会控制事务, 翻译表达式, 而且还可以向 Service Bean 中注入映射器 Mapper。

### 2.2 安装

要运行 MyBatis-Spring 模块, 你只需要包含 mybatis-spring-1.0.0-SNAPSHOT.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-SNAPSHOT</version>
</dependency>
```

### 2.3 创建 SqlSessionFactory

正如你已经知道的, 要使用 MyBatis 你需要创建一个 SqlSessionFactory 实例。MyBatis-Spring 会在 Spring 启动时为你创建。下面的 XML 片段就展示了需要创建 SqlSessionFactoryBean 的配置:

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

已经看到了, 要设置 SqlSessionFactory 就需要一个数据源。你可以提供 mybatis-config.xml 来设置 configLocation 属性:

```
<property name="configLocation" value="classpath:sample/mybatis-config.xml" />
```

当使用 MyBatis-Spring 时, 你应该在 mybatis-config.xml 中包含 transactionManager 和 dataSource 部分。当注入映射器时, 映射器列表也是需要的。

## 第三章 注入映射器

### 3.1 注入映射器

MyBatis-Spring 允许你在 Service Bean 中注入映射器。当使用映射器时，就像调用 DAO 那样来调用映射器就可以了，但是此时你就不需要进行任何 DAO 实现的编码，因为 MyBatis 会为你进行。

使用注入的映射器，你的代码就不会出现任何 MyBatis-Spring 依赖和 MyBatis 依赖。

在我们的应用中有这样一个简单的映射器。你也应该知道映射器仅仅是一个接口：

```
public interface UserMapper {  
    User getUser(String userId);  
}
```

这是你使用 MyBatis-Spring 来创建映射器的方式：

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

当使用映射器时你不需要在 mybatis-config.xml 文件中创建映射器列表，因为映射器可以在 MyBatis 启动时自行完成注册。

现在你的映射器已经准备在 Service 对象中注入了：

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

# 第四章 使用 SqlSessionDaoTemplate 和 SqlSessionDaoSupport

## 4.1 SqlSessionDaoTemplate

如果你需要使用 MyBatis 的 `SqlSession`，那么你应该使用 `SqlSessionDaoTemplate`。这个对象能够创建一个新的 `SqlSession` 或者从当前事务中获取活动的 `SqlSession`。它也可以给 Spring 的通用 `DataAccessException` 层次结构翻译表达式。

`SqlSessionDaoTemplate` 可以使用 `SqlSessionFactory` 作为构造方法参数来创建。

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

如下面这个片段展示的，你可以使用 `SqlSessionDaoTemplate` 来代替 `SqlSession` 执行 MyBatis 方法 (`selectOne`, `selectList`...):

```
public User getUser(String userId) {  
    return (User) sessionTemplate.selectOne("sample.UserMapper.getUser", userId);  
}
```

`SqlSessionDaoTemplate` 也提供一个通用的方法，使用一个自定义的 `SqlSessionCallback` 作为参数，这样你可以在一个 `SqlSession` 上执行多个方法：

```
public void insertUser(final User user) {  
    getSqlSessionTemplate().execute(new SqlSessionCallback<Object>() {  
        public Object doInSqlSession(SqlSession sqlSession) throws SQLException {  
            sqlSession.insert("sample.UserMapper.insertUser", user);  
            sqlSession.insert("sample.UserMapper.insertAccount", user.getId());  
            return null;  
        }  
    });  
}
```

## 4.2 SqlSessionDaoSupport

`SqlSessionDaoSupport` 是一个为你创建 `SqlSessionDaoTemplate` 的支持类，这样你可以通过调用 `getSqlSessionTemplate()` 方法来使用，比如：

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

## 第五章 使用 MyBatis API

### 5.1 使用 MyBatis API

你也可以直接使用 MyBatis API。这种情况下你不需要 MyBatis-Spring 依赖，仅仅在 DAO 中使用一个注入 SqlSessionFactory 的即可：

```
public class UserMapperSqlSessionImpl implements UserMapper {  
    private SqlSessionFactory sqlSessionFactory;  
    public void setSqlSessionFactory(SqlSessionFactory sqlSessionFactory) {  
        this.sqlSessionFactory = sqlSessionFactory;  
    }  
    public User getUser(String userId) {  
        SqlSession session = sqlSessionFactory.openSession();  
        try {  
            User user = (User) session.selectOne("sample.UserMapper.getUser",  
userId);  
            return user;  
        } finally {  
            session.close();  
        }  
    }  
}
```

在这种情况下 SqlSession 将不会在一个事务中被重用，也不会有异常翻译。Spring 的事务管理将会在内部的数据库连接中一直起作用。

# 第六章 示例代码

## 6.1 示例代码

你可以查看或从 MyBatis 的 Google Code 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。

`FooService` 作为事务服务。当它的任意一个方法被调用时开始和结束一个事务。看一看 `FooService.java` 文件来看看事务是怎么通过 `@Transactional` 属性配置的。这仅仅是一个示例，你可以使用任意一种 Spring 提供的方式来划定事务。

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

`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);
    }
}
```

在本文档中，数据库访问层已经使用 3 中不同的技术来实现了，看一看 `context.xml`。