

参考文档

简介

入门

SqlSessionFactoryBean

事务

使用 SqlSession

SqlSessionTemplate

SqlSessionDaoSupport

注入映射器

使用 Spring Boot

使用 MyBatis API


使用 Spring Batch

示例代码

返回代码

项目信息

项目链接



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了的语言, 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 可以被注入一个线程安全的 `SqlSession`, 它能基于 Spring 的事务配置来自动提交、回滚、关闭 `session`。

SqlSessionTemplate

`SqlSessionTemplate` 是 MyBatis-Spring 的核心, 作为 `SqlSession` 的一个实现, 这意味着你可以使用它来替代你代码中已经使用的 `SqlSession`。`SqlSessionTemplate` 是线程安全的, 可以被多个 DAO 或映射器所共享使用。

当调用 SQL 方法时 (包括由 `getMapper()` 方法返回的映射器中的方法), `SqlSessionTemplate` 将会保证使用的 `SqlSession` 与当前 Spring 的事务相关。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。另外, 它也负责将 MyBatis 的异常翻译成 Spring 中的 `DataAccessExceptions`。

由于模板可以参与到 Spring 的事务管理中, 并且由于它是线程安全的, 它可以供多个映射器类使用, 你试试总是用 `SqlSessionTemplate` 来替换 MyBatis 默认的 `DefaultSqlSession` 实现。在同一应用程序中的不同类之间混用使用可能会引起数据一致性的问题。

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

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

```
@Configuration
public class MyBatisConfig {
    @Bean
    public SqlSessionTemplate sqlSession() throws Exception {
        return new SqlSessionTemplate(sqlSessionFactory());
    }
}
```

现在, 这个 bean 就可以直接注入到你的 DAO bean 中了。你需要在你的 bean 中添加一个 `SqlSession` 属性, 就像下面这样:

```
public class UserDaoImpl implements UserDao {

    private SqlSession sqlSession;

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

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

按下面这样, 注入 `SqlSessionTemplate`:

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

`SqlSessionTemplate` 还有一个接收 `ExecutorType` 参数的构造方法。这允许你使用如下 Spring 配置来批量创建对象, 例如批量创建一些 `SqlSession`:

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

```
@Configuration
public class MyBatisConfig {

    @Bean
    public SqlSessionTemplate sqlSession() throws Exception {
        return new SqlSessionTemplate(sqlSessionFactory(), ExecutorType.BATCH);
    }
}
```

现在所有的映射语句包可以进行批量操作了, 可以在 DAO 中编写如下的代码

```
public class UserService {

    private final SqlSession sqlSession;

    public UserService(SqlSession sqlSession) {
        this.sqlSession = sqlSession;
    }

    public void insertUsers(List<User> users) {
        for (User user : users) {
            sqlSession.insert("org.mybatis.spring.sample.mapper.UserMapper.insertUser", user);
        }
    }
}
```

注意, 只需要在希望语句执行的方法与 `SqlSessionTemplate` 中的默认设置不同时使用这种配置。

这种配置的弊端在于, 当调用这个方法时, 不能存在使用不同 `ExecutorType` 的进行中的事务, 那么确保对不同 `ExecutorType` 的 `SqlSessionTemplate` 的调用处在不同的事务中, 要么完全不使用事务。

SqlSessionDaoSupport

`SqlSessionDaoSupport` 是一个抽象的支持类, 用来为你提供 `SqlSession`。调用 `getSqlSession()` 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 getSqlSession().selectOne("org.mybatis.spring.sample.mapper.UserMapper.getUser", userId);
    }
}
```

在这个类里, 通常更倾向于使用 `MapperFactoryBean`, 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 的工作或需要一个非抽象的实现类, 那么这个类就很有用了。

`SqlSessionDaoSupport` 需要通过属性设置一个 `sqlSessionFactory` 或 `SqlSessionTemplate`。如果两个属性都被设置了, 那么 `SqlSessionFactory` 将被忽略。

假设类 `UserMapperImpl` 是 `SqlSessionDaoSupport` 的子类, 可以编写如下的 Spring 配置来执行设置:

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