MyBatis与Spring整合

王新阳
wxyyuppie@bjfu.edu.cn

主要内容

1 导入相关JAR包

2 在Spring中配置MyBatis工厂

3 使用Spring管理MyBatis的数据操作接口

4 框架整合示例

MyBatis与Spring的整合——导入相关JAR包

1．MyBatis框架所需的JAR包

MyBatis框架所需的JAR包，包括它的核心包和依赖包。

2．Spring框架所需的JAR包

Spring框架所需的JAR包，包括它的核心模块JAR、AOP开发使用的JAR、JDBC和事务的JAR包（其中依赖包不需要再导入，因为MyBatis已提供），具体如下：

```
aopalliance-1.0.jar
aspectjweaver-1.8.13.jar
spring-aop-5.0.2.RELEASE.jar
spring-aspects-5.0.2.RELEASE.jar
spring-beans-5.0.2.RELEASE.jar
spring-context-5.0.2.RELEASE.jar
spring-core-5.0.2.RELEASE.jar
spring-expression-5.0.2.RELEASE.jar
spring-jdbc-5.0.2.RELEASE.jar
spring-tx-5.0.2.RELEASE.jar
```

MyBatis与Spring的整合——导入相关JAR包

3．MyBatis与Spring整合的中间JAR包

本课程采用的版本为mybatis-spring-1.3.1.jar。可从地址“http://mvnrepository.com/artifact/org. mybatis/mybatis-spring/1.3.1”下载。

4．数据库驱动JAR包

本书所使用的MySQL数据库驱动包为mysql-connector-java-5.1.45-bin.jar。

5．数据源所需的JAR包

整合时使用的是DBCP数据源，需要准备DBCP和连接池的JAR包。最新版本的DBCP的JAR包为commons-dbcp2-2.2.0.jar，可从地址“http://commons.apache.org/proper/commons-dbcp/ download_ dbcp.cgi”下载；最新版本的连接池的JAR包为commons-pool2-2.5.0.jar，可从地址“http://commons.apache.org/ proper/commons-pool/download_pool.cgi”下载。

MyBatis与Spring的整合——在Spring中配置MyBatis工厂

通过与Spring的整合，MyBatis的SessionFactory交由Spring来构建。构建时需要在Spring的配置文件中添加如下代码：

```
<!-- 配置数据源 -->
<bean id="dataSource" class="org.apache.commons.dbcp2.BasicDataSource">
    <property name="driverClassName" value="com.mysql.jdbc.Driver" />
    <property name="url" value="jdbc:mysql://localhost:3306/
                springtest?characterEncoding=utf8" />
    <property name="username" value="root" />
    <property name="password" value="root" />
       ……
</bean>
<!-- 配置MyBatis工厂，同时指定数据源，并与MyBatis完美整合 -->
 <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
        <!-- configLocation的属性值为MyBatis的核心配置文件 -->
    <property name="configLocation" value="classpath:com/mybatis/mybatis-config.xml"/>
 </bean>
```

使用Spring管理MyBatis的数据操作接口

使用Spring管理MyBatis的数据操作接口的方式有多种。其中，最常用最简洁的一种是基于MapperScannerConfigurer的整合。该方式需要在Spring的配置文件中加入以下内容：

```
<!--Mapper代理开发，使用Spring自动扫描MyBatis的接口并装配（Spring将指定包中所有被@Mapper注解标注的接口自动装配为MyBatis的映射接口） -->

<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
<!-- mybatis-spring组件的扫描器。com.dao只需要接口（接口方法 与SQL映射文件中相同-->
    <property name="basePackage" value="com.dao"/>
    <property name="sqlSessionFactoryBeanName"     value="sqlSessionFactory"/>
 </bean>
```

框架整合示例

1．创建应用并导入相关JAR包

见工程ch6SS

2．创建持久化类

3．创建SQL映射器文件和MyBatis核心配置文件

4．创建数据访问接口

5．创建日志文件

6．创建控制层

7．创建Spring的配置文件

8．创建测试类

框架整合示例

主要整合处

自动扫描指定包

```
<bean id="dataSource" class="org.apache.commons.dbcp2.BasicDataSource">
        <property name="driverClassName" value="com.mysql.jdbc.Driver" />
        <property name="url" value="jdbc:mysql://localhost:3306/java web?characterEncoding=utf8" />
        <property name="username" value="root" />
        <property name="password" value="root" />
        <!-- 最大连接数 -->
        <property name="maxTotal" value="30"/>
        <!-- 最大空闲连接数 -->
        <property name="maxIdle" value="10"/>
        <!-- 初始化连接数 -->
        <property name="initialSize" value="5"/>
</bean>
<!-- 添加事务支持 -->
<bean id="txManager"
    class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource" />
</bean>
<!-- 开启事务注解-->
<tx:annotation-driven transaction-manager="txManager" />
<!-- 配置MyBatis工厂，同时指定数据源，并与MyBatis完美整合 -->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <!-- configLocation的属性值为MyBatis的核心配置文件 -->
    <property name="configLocation" value="classpath:com/mybatis/mybatis-config.xml"/>
</bean>
<!-Mapper代理开发，使用Spring自动扫描MyBatis的接口并装配
（Spring将指定包中所有被@Mapper注解标注的接口自动装配为MyBatis的映射接口） -->
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <!-- mybatis-spring组件的扫描器 -->
    <property name="basePackage" value="com.dao"/>
    <property name="sqlSessionFactoryBeanName" value="sqlSessionFactory"/>
</bean>
```

Spring注解装配

```java
@Repository("userDao")
@Mapper
/*使用Spring自动扫描MyBatis的接口并装配
（Spring将指定包中所有被@Mapper注解标注的接口自动装配为MyBatis的映射接口*/
public interface UserDao {
    /**
     * 接口方法对应SQL映射文件UserMapper.xml中的id
     */
    public MyUser selectUserById(Integer uid);
    public List<MyUser> selectAllUser();
    public int addUser(MyUser user);
    public int updateUser(MyUser user);
    public int deleteUser(Integer uid);
}
```

自动匹配注解，会自动将具有与当前接口全路径名相同的Mapper的namespace对应，并将mapper中的相应sql查询对应到该接口同名方法上，同名方法的参数即为mapper的参数，mapper的返回类型即为同名方法的返回类型。

```xml
<mapper namespace="com.dao.UserDao">
    <!-- 根据uid查询一个用户信息 -->
    <select id="selectUserById" parameterType="Integer" resultType="com.po.MyUser">
        select * from user where uid = #{uid}
    </select>
    <!-- 查询所有用户信息 -->
    <select id="selectAllUser"  resultType="com.po.MyUser">
        select * from user
    </select>
    <!-- 添加一个用户，#{uname}为com.po.MyUser的属性值-->
    <insert id="addUser" parameterType="com.po.MyUser">
        insert into user (uname,usex) values(#{uname},#{usex})
    </insert>
    <!-- 修改一个用户 -->
    <update id="updateUser" parameterType="com.po.MyUser">
        update user set uname = #{uname},usex = #{usex} where uid = #{uid}
    </update>
    <!-- 删除一个用户 -->
    <delete id="deleteUser" parameterType="Integer">
        delete from user where uid = #{uid}
    </delete>
</mapper>
```

框架整合示例——与未整合对比

```java
public interface UserDao {

    /**
     * 根据id查询用户信息
     *
     * @param id
     * @return
     */
    public User queryUserById(String id);

    /**
     * 查询所有用户信息
     *
     * @return
     */
    public List<User> queryUserAll();

    /**
     * 新增用户
     *
     * @param user
     */
    public void insertUser(User user);
}
```

必须通过sqlSession显式调用相关方法实现对应的接口

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

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

    @Override
    public User queryUserById(String id) {
        return this.sqlSession.selectOne("UserDao.queryUserById", id);
    }

    @Override
    public List<User> queryUserAll() {
        return this.sqlSession.selectList("UserDao.queryUserAll");
    }

    @Override
    public void insertUser(User user) {
        this.sqlSession.insert("UserDao.insertUser", user);
    }
```

```java
@Repository("userDao")
@Mapper
/*使用Spring自动扫描MyBatis的接口并装配
（Spring将指定包中所有被@Mapper注解标注的接口自动装配为MyBatis的映射接口*/
public interface UserDao {
    /**
     * 接口方法对应SQL映射文件UserMapper.xml中的id
     */
    public MyUser selectUserById(Integer uid);
    public List<MyUser> selectAllUser();
    public int addUser(MyUser user);
    public int updateUser(MyUser user);
    public int deleteUser(Integer uid);
}
```

```java
public class UserController {
    @Autowired
    private UserDao userDao;
    public void test() {
        //查询一个用户
        MyUser auser = userDao.selectUserById(1);
        System.out.println(auser);
        System.out.println("================");
        //添加一个用户
        MyUser addmu = new MyUser();
        addmu.setUname("chen");
        addmu.setUsex("male");
        int add = userDao.addUser(addmu);
        System.out.println("添加了" + add + "条记录");
        System.out.println("================");
        //修改一个用户
        MyUser updatemu = new MyUser();
        updatemu.setUid(1);
        updatemu.setUname("张三");
        updatemu.setUsex("女");
        int up = userDao.updateUser(updatemu);
        System.out.println("修改了" + up + "条记录");
```

数据库操作变得像使用普通方法一样简单直接

感谢聆听

Thanks For Your Listening!