整合MyBatis

步骤：

- 导入相关jar包（添加依赖）
  - junit
  - mybatis
  - mysql
  - spring相关的
  - aop织入
  - mybatis-spring【专门整合spring和mybatis的】

```
<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.12</version>
    </dependency>

    <!--Mysql驱动-->
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.46</version>
    </dependency>

    <dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis</artifactId>
        <version>3.4.6</version>
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>5.2.0.RELEASE</version>
    </dependency>

    <!--Spring操作数据库的话，还需要一个Spring-jdbc
    这里对应 后续配置DataSource
    -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-jdbc</artifactId>
        <version>5.2.0.RELEASE</version>
    </dependency>

    <dependency>
        <groupId>org.aspectj</groupId>
        <artifactId>aspectjweaver</artifactId>
        <version>1.8.13</version>
    </dependency>

```

```
    <dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis-spring</artifactId>
        <version>2.0.2</version>
    </dependency>

    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <version>1.18.8</version>
    </dependency>
</dependencies>
```

- 编写配置文件
- 测试运行

1 回顾MyBatis，测试环境

- 实体类

```
@Data
@AllArgsConstructor
@NoArgsConstructor
public class User implements Serializable {
    private int id;
    private String name;
    private String pwd;
}
```

这里使用了Lombok，所以没有get、set、toString方法。

- mapper接口

```
public interface UserMapper {
    /**
     * 查询所有的用户
     *
     * @return
     */
    public List<User> getUsers();
}
```

- mapper.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
        PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="com.kuang.mapper.UserMapper">
    <select id="getUsers" resultType="user">
        select * from mybatis.user
    </select>

</mapper>
```

- 核心配置文件

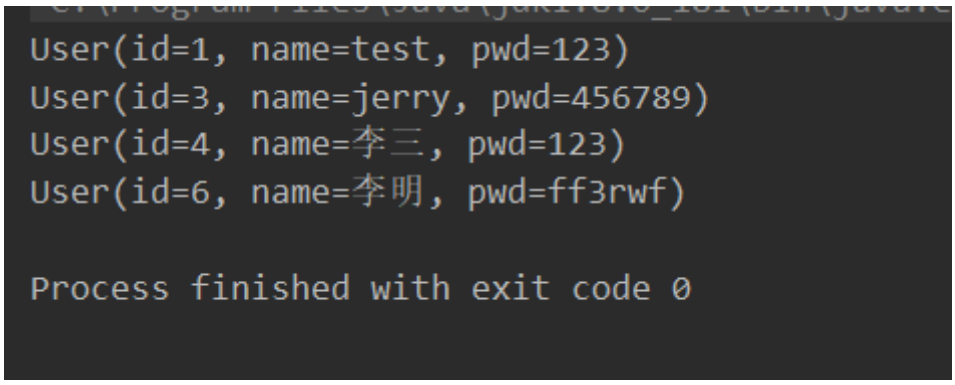
```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
        PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>
    <typeAliases>
        <package name="com.kuang.pojo"/>
    </typeAliases>

    <environments default="development">
        <environment id="development">
            <transactionManager type="JDBC"/>
            <dataSource type="POOLED">
                <property name="driver" value="com.mysql.jdbc.Driver"/>
                <property name="url"
                          value="jdbc:mysql://localhost:3306/mybatis?useSSL=true&amp;useUnicode=true&amp;characterEncoding=UTF-8"/>
                <property name="username" value="root"/>
                <property name="password" value="mynewroot"/>
            </dataSource>
        </environment>
    </environments>

    <mappers>
        <mapper class="com.kuang.mapper.UserMapper"/>
    </mappers>

</configuration>
```

- 工具类

```java
public class MybatisUtils {
    private static SqlSessionFactory sqlSessionFactory;

    static {
        //使用Mybatis的第一步：获取sqlSessionFactory对象
        try {
            String resource = "mybatis-config.xml";
            InputStream inputStream = Resources.getResourceAsStream(resource);
            sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }


    //    使用SqlSessionFactory获取SqlSession
    public static SqlSession getSqlSession() {
        return sqlSessionFactory.openSession();
    }
}
```

- 测试运行

```
@Test
public void getUsers(){
    SqlSession sqlSession = MybatisUtils.getSqlSession();

    UserMapper mapper = sqlSession.getMapper(UserMapper.class);
    List<User> users = mapper.getUsers();

    for (User user : users) {
        System.out.println(user);
    }
}
```



2 mybatis—spring（方式一）

官网：http://mybatis.org/spring/zh/index.html

介绍：

1. 版本注意

MyBatis-Spring	MyBatis	Spring 框架	Spring Batch	Java
2.0	3.5+	5.0+	4.0+	Java 8+
1.3	3.4+	3.2.2+	2.1+	Java 6+

2. MyBatis-Spring 会帮助你将 MyBatis 代码无缝地整合到 Spring 中。它将允许 MyBatis 参与到 Spring 的事务管理之中，创建映射器 mapper 和 `SqlSession` 并注入到 bean 中。
3. 所指定的映射器类必须是一个**接口**，而不是具体的实现类。

2.1 编写核心配置文件

- 这里我们说明一点：我们会写一个spring-dao.xml，里面专注于一些固定的配置（数据库、整合mybatis等等）
- 而applicationContext.xml，是总的配置文件，我们在这个文件中只做两个任务：
  - 引入spring-dao.xml（在这个配置文件中，整合了Mybatis的配置，所以其实我们也是可以省略myabtis的核心配置文件的）
  - 注册管理bean
- applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
        https://www.springframework.org/schema/beans/spring-beans.xsd
```

```
        http://www.springframework.org/schema/aop
        https://www.springframework.org/schema/aop/spring-aop.xsd">

    <import resource="spring-dao.xml"/>

    <bean id="userMapper" class="com.kuang.mapperImpl.UserMapperImpl">
        <!--set方法注入sqlSessionTemplate-->
        <property name="sqlSessionTemplate" ref="sqlSessionTemplate"/>
    </bean>
</beans>
```

- spring-dao.xml

  一共分为三部分：

  - DataSource（数据源）
  - SqlSessionFactory （里面配置数据源、还有myabtis的配置）
  - SqlSessionTemplate（也就是SqlSession），以后不再使用工具类获得SqlSession

  重点在SqlSessionFactory和SqlSessionTemplate这两个对象。

  DataSource要注入到SqlSessionFactory中。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
        https://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/aop
        https://www.springframework.org/schema/aop/spring-aop.xsd">

    <!--
    DataSource:使用Spring的数据源替换MyBatis的配置
    c3p0、druid、dbcp
    我们这里使用Spring提供的JDBC：依赖引入了spring-jdbc
    -->
    <bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName"
value="com.mysql.jdbc.Driver"/>
        <property name="url"
                  value="jdbc:mysql://localhost:3306/mybatis?
useSSL=true&amp;useUnicode=true&amp;characterEncoding=UTF-8"/>
        <property name="username" value="root"/>
        <property name="password" value="mynewroot"/>
    </bean>

    <!--sqlSessionFactory
    记得配置数据源
    -->
    <bean id="sqlSessionFactory"
class="org.mybatis.spring.SqlSessionFactoryBean">
        <!--配置数据源-->
        <property name="dataSource" ref="dataSource"/>

        <!--可以绑定myabtis的配置文件
            这样子就可以引用mybatis中的一些配置（别名。。。）
        -->
```

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

        <!--当然也可以在这里配置myabtis里面的配置
        例如：mapper
        一般我们的myabtis的配置文件只留两个：别名管理 和 setting设置
        -->
        <property name="mapperLocations" value="classpath:com/kuang/mapper/*.xml"/>

    </bean>

    <!--sqlSession
    sqlSessionTemplate 相当于我们的SqlSession
    -->
    <bean id="sqlSessionTemplate" class="org.mybatis.spring.SqlSessionTemplate">
        <!--看源码可知，这里缺了个构造参数的注入
        我们只能使用构造器注入，因为没有set方法（源码有）
        -->
        <constructor-arg index="0" ref="sqlSessionFactory"/>
    </bean>


</beans>
```

- 因为在spring-dao中有所配置了一些mybatis的配置，所以我们的myabtis的核心配置文件以后一般只写两种配置：
  - 别名
  - setting

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
        PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>
    <typeAliases>
        <package name="com.kuang.pojo"/>
    </typeAliases>


</configuration>
```

2.2 需要给接口加一个实现类

```
public class UserMapperImpl implements UserMapper {
    //在原来，我们所有的操作都是用SqlSession，而现在使用SqlSessionTemplate
    private SqlSessionTemplate sqlSessionTemplate;

    //注入SqlSessionTemplate
    public void setSqlSessionTemplate(SqlSessionTemplate sqlSessionTemplate) {
        this.sqlSessionTemplate = sqlSessionTemplate;
    }
```

```
9
10
11      @Override
12      public List<User> getUsers() {
13
14          UserMapper mapper = sqlSessionTemplate.getMapper(UserMapper.class);
15          return mapper.getUsers();
16      }
17  }
```

2.2.1 这是和myabtis不同的地方：接口有了一个实现类。

2.2.2 这个实现类中我们关注两个地方：

- SqlSessionTemplate：原先我们使用SqlSession，现在我们使用这个进行操作（增删改查），比如getMapper。
- 注入SqlSessionTemplate：记得在applicationContext管理这个接口实现类的时候，记得使用set注入

2.2.3 这个接口实现类有什么意义?

- 完成一些业务操作
- 我们后续只要调用这个接口实现类的方法即可
- 接口实现类记得注册到Spring容器中，让Spring管理

2.3 测试运行

```
1   @Test
2   public void getUsers2(){
3       ApplicationContext context = new
    ClassPathXmlApplicationContext("applicationContext.xml");
4
5       UserMapper userMapper = context.getBean("userMapper",
    UserMapper.class);
6       List<User> users = userMapper.getUsers();
7
8       for (User user : users) {
9           System.out.println(user);
10      }
11  }
```

```
User(id=1, name=test, pwd=123)
User(id=3, name=jerry, pwd=456789)
User(id=4, name=李三, pwd=123)
User(id=6, name=李明, pwd=ff3rwf)

Process finished with exit code 0
```

3 SqlSessionDaoSupport（方式二）

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。

- 接口实现类

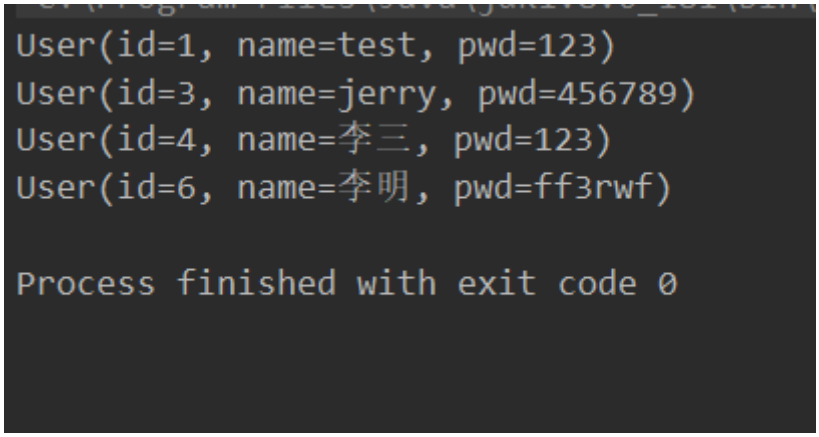
```
public class UserMapperImpl2 extends SqlSessionDaoSupport implements UserMapper {
    @Override
    public List<User> getUsers() {
        return getSqlSession().getMapper(UserMapper.class).getUsers();
    }
}
```

- 注册bean，在bean中配置sqlSessionFactory即可

```
<bean id="userMapper2" class="com.kuang.mapperImpl.UserMapperImpl2">
    <property name="sqlSessionFactory" ref="sqlSessionFactory"/>
</bean>
```

- 测试运行

```
@Test
public void getUsersbydaosupport(){
    ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

    UserMapper userMapper = context.getBean("userMapper2", UserMapper.class);
    List<User> users = userMapper.getUsers();

    for (User user : users) {
        System.out.println(user);
    }
}
```



注意点：

1. bean中需要配置SqlSessionFactory（核心）
2. 可以省略之前配置的sqlSessionTemplate，因为通过getSession()方法会得到一个SqlSessionTemplate