

Spring笔记

目录

第一章、HelloSpring

什么是spring

spring核心

spring优点

spring体系结构

控制反转(ioc: Inversion of Control) / 依赖注入 (DI: Dependency Injection)

spring框架搭建步骤

eclipse schema xml提示

核心API

面向切面编程 (AOP)

什么是AOP?

AOP实现原理

AOP术语【掌握】

案例-使用AOP打印Log日志:

第二章、Ioc和AOP

案例-使用构造注入

设值注入 (set注入) 和构造注入的区别:

案例-使用P命名空间注入属性值

案例-注入不同数据类型

案例-定义异常抛出增强

案例-定义最终增强

案例-定义环绕增强

Spring 常用注解

1.声明bean的注解

2.注入bean的注解

3.java配置类相关注解

4.切面（AOP）相关注解

5.@Bean的属性支持

6.@Value注解

7.环境切换

8.异步相关

9.定时任务相关

10.@Enable*注解说明

11.测试相关注解

装配Bean基于注解

案例-使用注解实现IoC

案例-使用注解实现切面

案例-使用注解实现异常抛出增强

案例-使用注解实现最终增强

案例-使用注解实现环绕增强

第三章、Spring整合MyBatis

整体思路：

整合环境：

Spring整合MyBatis-基础版

Spring整合MyBatis-使用SqlSessionDaoSupport简化代码

Spring整合MyBatis-MapperCannerConfig简化配置工作量

Spring整合MyBatis-使用MapperCannerConfig简化配置工作量

Spring整合MyBatis-为业务层添加声明式事务

Spring整合MyBatis-使用注解为业务层添加声明式事务

Spring整合MyBatis-使用properties文件配置数据源

Spring整合MyBatis-使用JNDI

第一章、HelloSpring

什么是spring

Spring是一个开源框架，Spring是于2003 年兴起的一个轻量级的Java 开发框架，由Rod Johnson 在其著作Expert One-On-One J2EE Development and Design中阐述的部分理念和原型衍生而来。它是为了解决企业应用开发的复杂性而创建的。框架的主要优势之一就是其分层架构，分层架构允许使用者选择使用哪一个组件，同时为 J2EE 应用程序开发提供集成的框架。Spring使用基

本的JavaBean来完成以前只可能由EJB完成的事情。然而，Spring的用途不仅限于服务器端的开发。从简单性、可测试性和松耦合的角度而言，任何Java应用都可以从Spring中受益。**Spring的核心是控制反转 (IoC) 和面向切面 (AOP)**。简单来说，Spring是一个分层的JavaSE/EE **full-stack(一站式)** 轻量级开源框架。

- 轻量级：与EJB对比，依赖资源少，销毁的资源少。
- 分层：一站式，每一个层都提供的解决方案
web层：struts, spring-MVC
service层：spring
dao层：hibernate, mybatis , jdbcTemplate --> spring-data
- AOP实现
- 数据访问支持
 - 简化JDBC/ORM（对象关系映射）框架
 - 声明式事务
- WEB集成

spring核心

Spring的核心是**控制反转 (IoC)** 和 **面向切面 (AOP)**

spring优点

- 方便解耦，简化开发（高内聚低耦合）
- **Spring就是一个大工厂（容器）**，可以将所有对象创建和依赖关系维护，交给Spring管理
- **spring工厂是用于生成bean**
- AOP编程的支持
Spring提供面向切面编程，可以方便的实现对程序进行权限拦截、运行监控等功能
- 声明式事务的支持
只需要通过配置就可以完成对事务的管理，而无需手动编程

- 方便程序的测试

Spring对Junit4支持，可以通过注解方便的测试Spring程序

- 方便集成各种优秀框架

Spring不排斥各种优秀的开源框架，其内部提供了对各种优秀框架（如：Struts、Hibernate、MyBatis、Quartz等）的直接支持

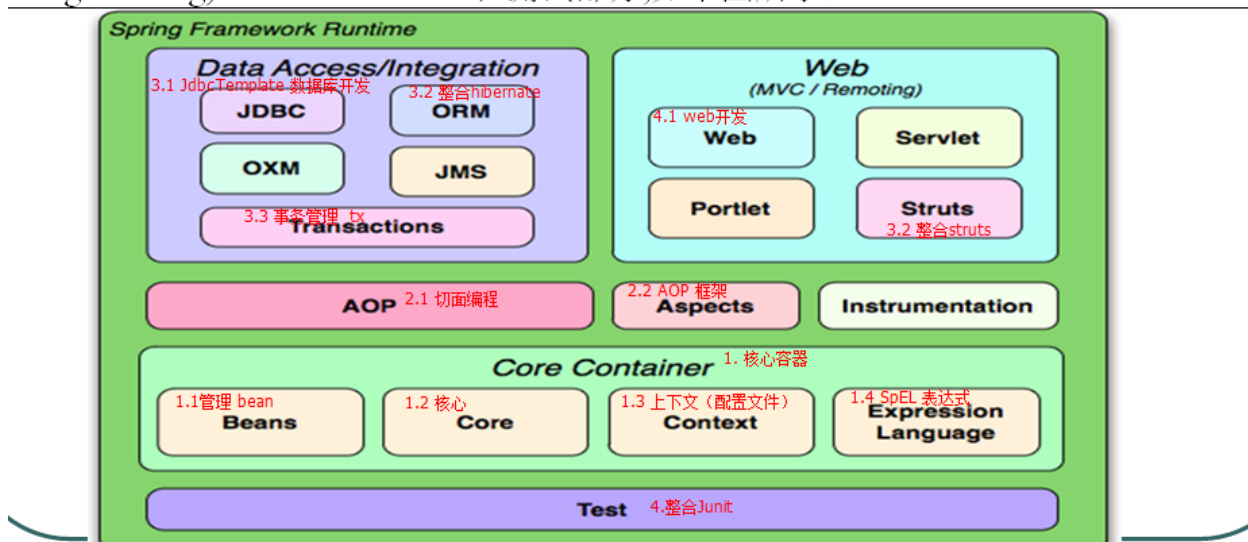
- 降低JavaEE API的使用难度

Spring 对JavaEE开发中非常难用的一些API（JDBC、JavaMail、远程调用等），都提供了封装，使这些API应用难度大大降低

- 低侵入式设计
- 独立于各种应用服务器
- 面向切面编程特性允许将通用任务进行集中式处理

spring体系结构

Spring 框架是一个分层架构,它包含一系列的功能要素并被分为大约20个模块。这些模块分为Core Container、Data Access/Integration、Web、AOP（Aspect Oriented Programming）、Instrumentation和测试部分,如下图所示:



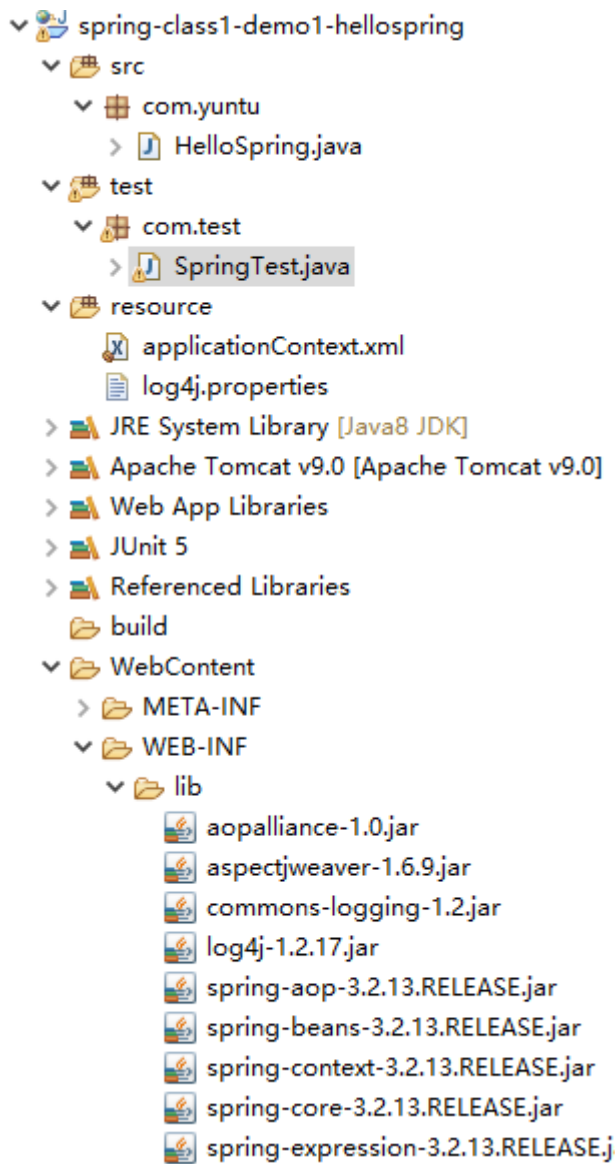
核心容器: beans、core、context、expression

控制反转(ioc: Inversion of Control) / 依赖注入 (DI: Dependency Injection)

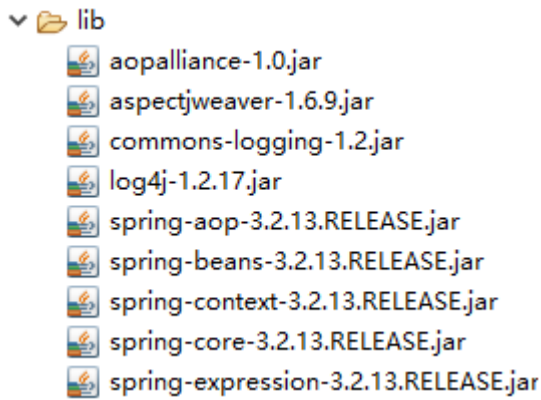
- 组件化的思想：分离关注点，使用接口，不再关注实现
- 依赖注入：将组件的构建和使用分开

spring框架搭建步骤

项目结构：



1.创建JavaWeb项目，导入jar包到WebRoot的lib文件夹（4个核心（beans、core、context、expression） + 1个依赖（commons-loggins...jar））



2.目标类

```
1 package com.yuntu;
2 /**
3  * 依赖注入(DI)
4  * @author 阿华
5  *
6  */
7 public class HelloSpring {
8     //定义who属性，通过属性值控制在spring框架中进行设置
9     private String who=null;
10
11     /**
12     * get set方法
13     * @return
14     */
15     public String getWho() {
16         return who;
17     }
18
19     public void setWho(String who) {
20         this.who = who;
21     }
22
23     /**
24     * 定义打印方法，输出内容
25     */
26     public void print() {
```

```
27 System.out.println("Hello,"+this.who);
28 }
29 }
```

3.Spring配置文件

- 位置：任意，开发中一般在classpath下（src）
- 名称：任意，开发中常用applicationContext.xml
- 内容：添加schema约束

约束文件位置：spring-framework-3.2.0.RELEASE\docs\spring-framework-reference\html\xsd-config.html

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!-- Spring配置文件的根标签 -->
3 <!--
4 xmlns: 引入的文件路径
5 xsi:schemaLocation: 对应约束文件的路径
6 -->
7 <beans xmlns="http://www.springframework.org/schema/beans"
8 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
9 xsi:schemaLocation="
10 http://www.springframework.org/schema/beans
11 http://www.springframework.org/schema/beans/spring-beans-4.0.x
12 sd ">
13 <!-- 注入 -->
14 <!-- 通过bean元素声明所需要的spring创建的实例，通过调用setWho()方法
15  进行传值 -->
16 <!-- id:实例化的对象名 class: 类文件完整路径，需要实例化的类文件 -->
17 <!-- HelloSpring helloSpring = new HelloSpring(); -->
18 <bean id="helloSpring" class="com.yuntu.HelloSpring">
19 <!-- 向对象中注入属性，通过调用setWho()方法进行传值 -->
20 <!-- name: name值必须和类文件中的属性名一致 -->
21 <property name="who">
22 <!-- 此处是将字符串 "Spring!!!" 赋值给who属性 -->
23 <value>Spring!!!</value>
```

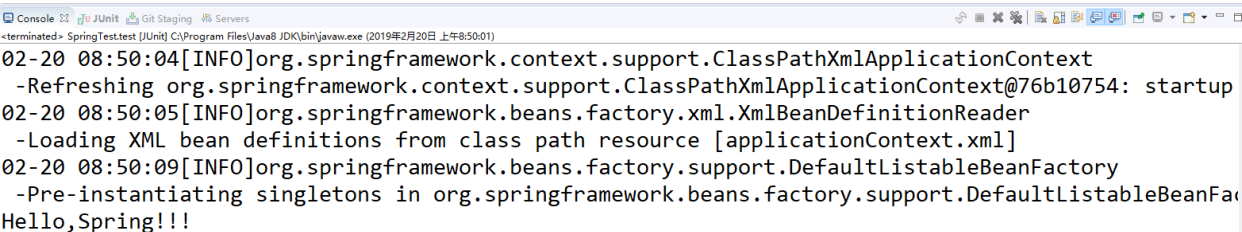


```
23 </property>
24 </bean>
25 </beans>
```

4.测试

```
1 package com.test;
2
3 import org.junit.jupiter.api.Test;
4 import org.springframework.context.ApplicationContext;
5 import org.springframework.context.support.ClassPathXmlApplication
  onContext;
6 import com.yuntu.HelloSpring;
7
8 public class SpringTest {
9     @Test
10    public void test() {
11        //1.通过ClassPathXmlApplicationContextSpring实例化上下文对象
12        ApplicationContext ctx = new ClassPathXmlApplicationContext("a
  pplicationContext.xml");
13        //2.通过ApplicationContext提供的getBean()方法，根据xml中的id获取b
  ean的实例化对象
14        HelloSpring helloSpring =(HelloSpring) ctx.getBean("helloSprin
  g");
15        //3.调用输出方法
16        helloSpring.print();
17    }
18 }
```

运行结果：

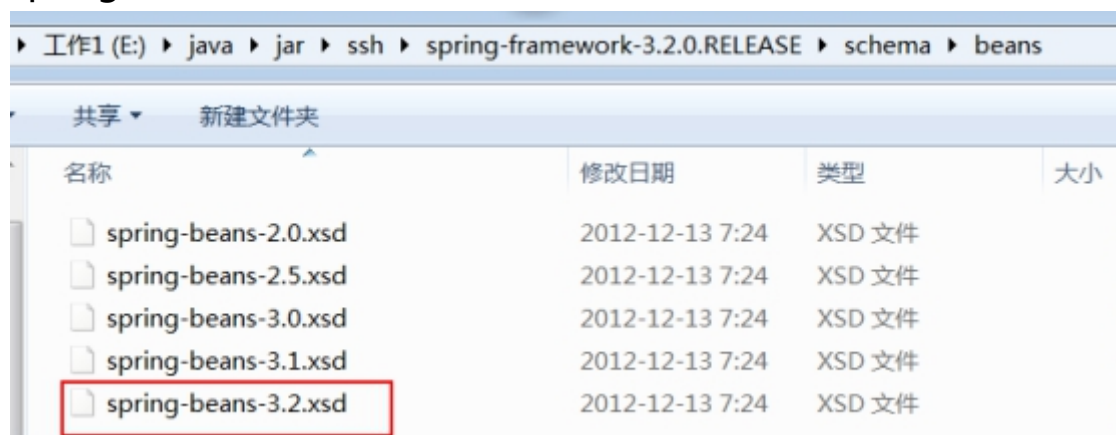


```
<terminated> SpringTest.test [JUnit] C:\Program Files\Java\jdk-11.0.2\bin\javaw.exe (2019年2月20日 上午8:50:01)
02-20 08:50:04[INFO]org.springframework.context.support.ClassPathXmlApplicationContext
-Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@76b10754: startup
02-20 08:50:05[INFO]org.springframework.beans.factory.xml.XmlBeanDefinitionReader
-Loading XML bean definitions from class path resource [applicationContext.xml]
02-20 08:50:09[INFO]org.springframework.beans.factory.support.DefaultListableBeanFactory
-Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBeanFac
HelloSpring!!!
```

eclipse schema xml提示

步骤一：确定xsd文件位置

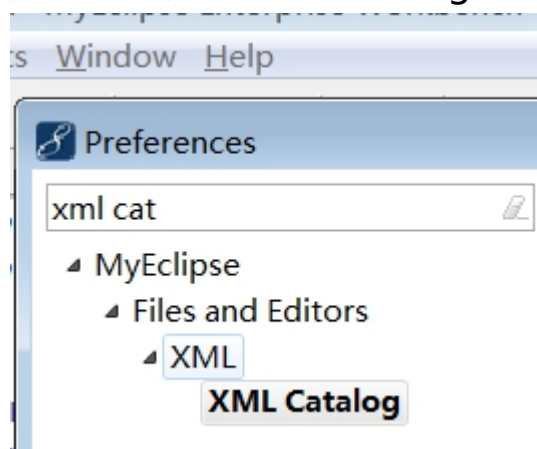
spring-framework-3.2.0.RELEASE\schema\beans



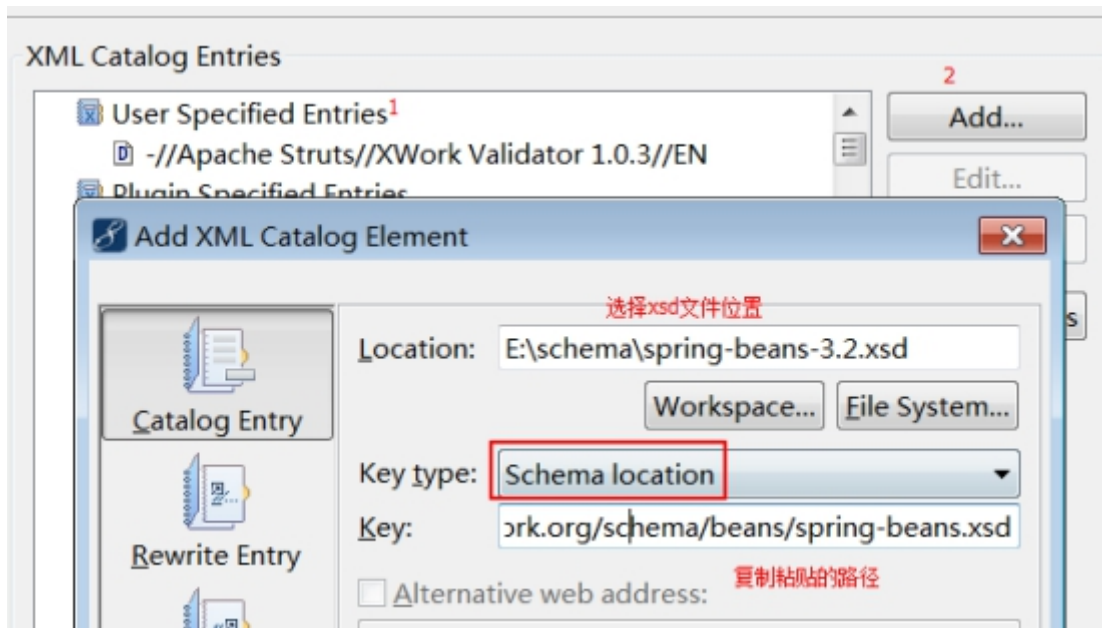
步骤二：复制路径

```
2 <beans xmlns="http://www.springframework.org/schema/beans"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://www.springframework.org/schema/beans
5         http://www.springframework.org/schema/beans/spring-beans.xsd">
6 <!--
```

步骤三：搜索 “xml catalog”

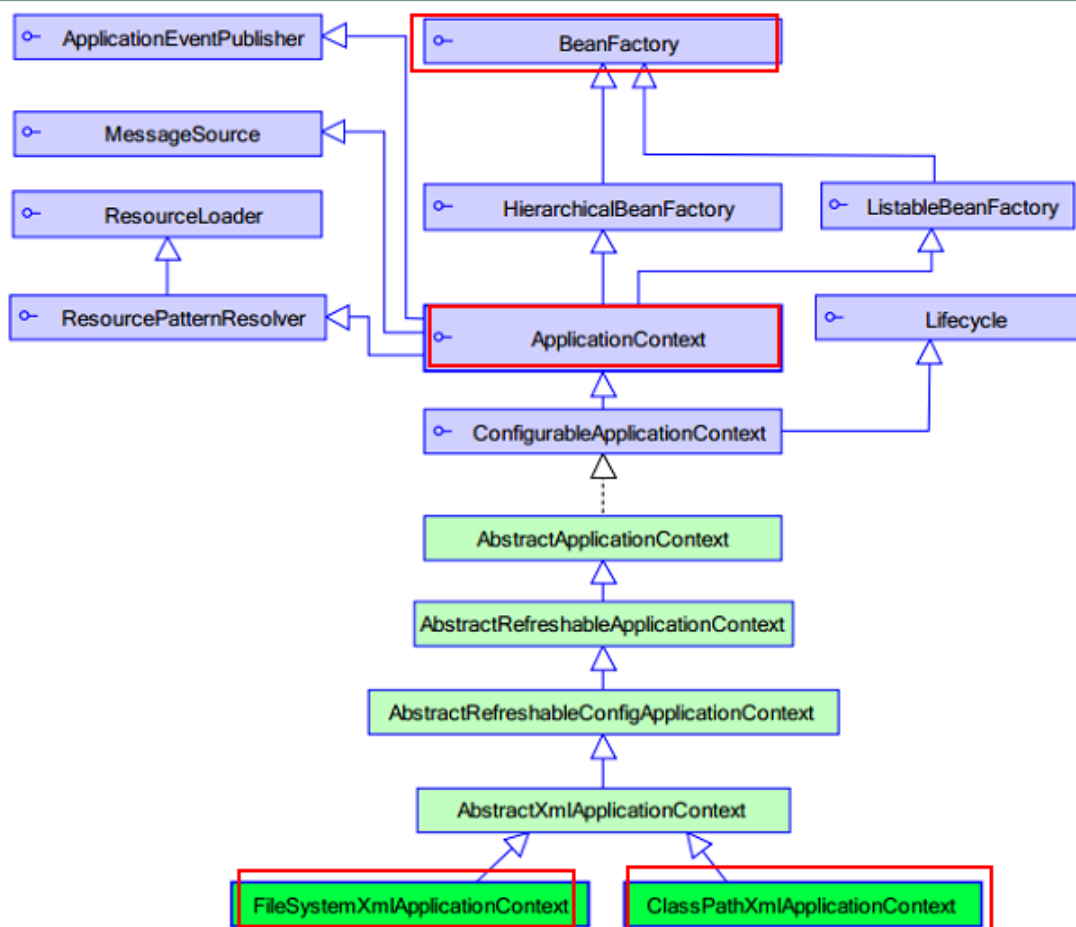


步骤四：添加约束提示



核心API

api整体了解，之后不使用，在学习过程需要。



- BeanFactory：这是一个工厂，用于生成任意bean。

采取延迟加载，第一次getBean时才会初始化Bean

- ApplicationContext：是BeanFactory的子接口，功能更强大。（国际化处理、事件传递、Bean自动装配、各种不同应用层的Context实现）。当配置文件被加载，就进行对象实例化。

ClassPathXmlApplicationContext 用于加载classpath（类路径、src）下的xml

加载xml运行时位置 --> /WEB-INF/classes/...xml

FileSystemXmlApplicationContext 用于加载指定盘符下的xml

加载xml运行时位置 --> /WEB-INF/...xml

通过java web ServletContext.getRealPath() 获得具体盘符

面向切面编程（AOP）

什么是AOP？

- 在软件业，AOP为Aspect Oriented Programming的缩写，意为：面向切面编程，通过预编译方式和运行期动态代理实现程序功能的统一维护的一种技术。AOP是OOP（面向对象编程）的延续，是软件开发中的一个热点，也是Spring框架中的一个重要内容，是函数式编程的一种衍生范型。利用AOP可以对业务逻辑的各个部分进行隔离，从而使得业务逻辑各部分之间的耦合度降低，提高程序的可重用性，同时提高了开发的效率。
- AOP采取横向抽取机制，取代了传统纵向继承体系重复性代码
- 经典应用：事务管理、性能监视、安全检查、缓存、日志等
- Spring AOP使用纯Java实现，不需要专门的编译过程和类加载器，在运行期通过代理方式向目标类织入增强代码
- AspectJ是一个基于Java语言的AOP框架，Spring2.0开始，Spring AOP引入对Aspect的支持，AspectJ扩展了Java语言，提供了一个专门的编译器，在编译时提供横向代码的织入

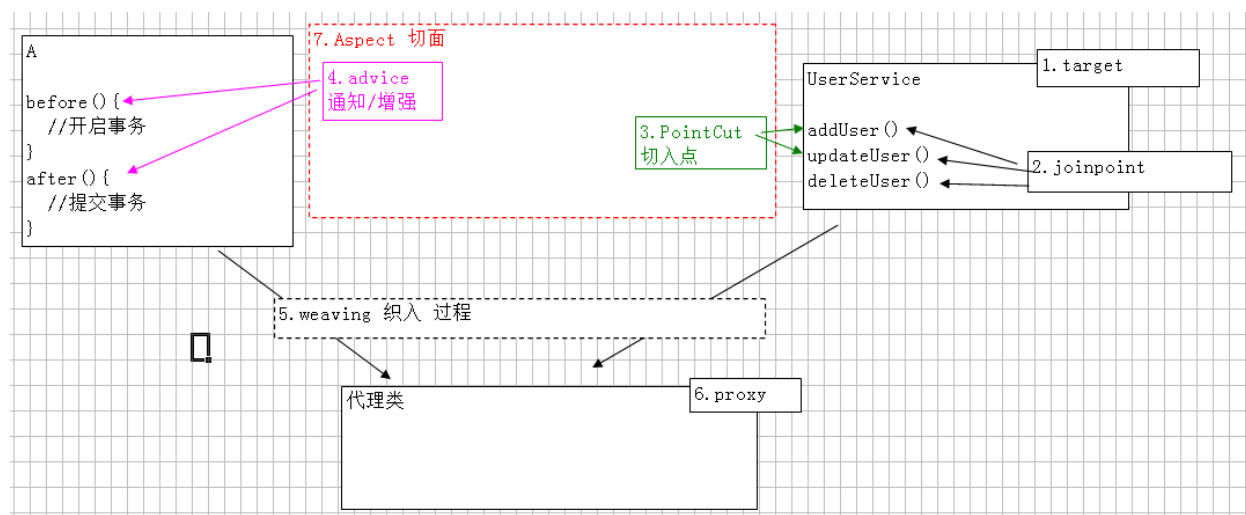
AOP实现原理

- aop底层将采用代理机制进行实现。
- 接口 + 实现类：spring采用jdk的**动态代理**Proxy。

- 实现类：spring 采用 **cglib字节码增强**。

AOP术语【掌握】

- 1.target: 目标类, 需要被代理的类。例如: UserService
- 2.Joinpoint(连接点):所谓连接点是指那些可能被拦截到的方法。例如: 所有的方法
- 3.PointCut 切入点: 已经被增强的连接点。例如: addUser()
- 4.advice 通知/增强处理, 增强代码。例如: after、before
 - 前置增强
 - 后置增强
 - 环绕增强、异常抛出增强、最终增强等类型
5. Weaving(织入):是指把增强advice应用到目标对象target来创建新的代理对象proxy的过程.
- 6.proxy 代理类
7. Aspect(切面): 是切入点pointcut和通知advice的结合
 - 一个线是一个特殊的面。
 - 一个切入点和一个通知, 组成成一个特殊的面。



案例-使用AOP打印Log日志:

实体类:

```
1 package entity;
2 /**
3  * 实体类
4  * @author 阿华
```

```
5  *
6  */
7  public class User {
8      private Integer id;
9      private String name;
10     private String pwd;
11
12     public User() {
13         super();
14     }
15     public User(Integer id, String name, String pwd) {
16         super();
17         this.id = id;
18         this.name = name;
19         this.pwd = pwd;
20     }
21
22     public Integer getId() {
23         return id;
24     }
25     public void setId(Integer id) {
26         this.id = id;
27     }
28     public String getName() {
29         return name;
30     }
31     public void setName(String name) {
32         this.name = name;
33     }
34     public String getPwd() {
35         return pwd;
36     }
37     public void setPwd(String pwd) {
38         this.pwd = pwd;
39     }
```

```
40 }
```

dao接口:

```
1 package dao;
2
3 import entity.User;
4
5 /**
6  * 用户数据操作接口
7  * @author 阿华
8  *
9  */
10 public interface UserDao {
11     /**
12     * 保存
13     */
14     public void save(User user);
15 }
16
```

dao实现类:

```
1 package dao.impl;
2
3 import dao.UserDao;
4 import entity.User;
5 /**
6  * 实现类
7  * @author 阿华
8  *
9  */
10 public class UserDaoImpl implements UserDao {
11     //模拟去数据库中添加数据
12     @Override
13     public void save(User user) {
14         System.out.println("添加到数据库中成功");
15     }
16 }
```

```
15 }  
16 }
```

service接口:

```
1 package service;  
2  
3 import entity.User;  
4  
5 public interface UserService {  
6     /**  
7      * 添加用户  
8      */  
9     public void addUser(User user);  
10 }  
11
```

service实现类:

```
1 package service.impl;  
2  
3 import dao.UserDao;  
4 import entity.User;  
5 import service.UserService;  
6  
7 public class UserServiceImpl implements UserService{  
8     //创建dao层对象，用于Spring进行注入  
9     private UserDao dao;  
10  
11     public void setDao(UserDao dao) {  
12         this.dao = dao;  
13     }  
14  
15     @Override  
16     public void addUser(User user) {  
17         dao.save(user);  
18     }  
19 }
```


spring配置文件:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xmlns:aop="http://www.springframework.org/schema/aop"
5   xsi:schemaLocation="
6     http://www.springframework.org/schema/beans
7     http://www.springframework.org/schema/beans/spring-beans.xsd
8     http://www.springframework.org/schema/aop
9     http://www.springframework.org/schema/aop/spring-aop-3.2.xsd">
10
11   <!-- 实例化dao -->
12   <bean id="udao" class="dao.impl.UserDaoImpl"></bean>
13
14   <!-- 注入Service -->
15   <bean id="service" class="service.impl.UserServiceImpl">
16     <property name="dao" ref="udao"></property>
17   </bean>
18
19   <!-- 声明增强方法所在的类 -->
20   <bean id="userLoggerAop" class="aop.UserLoggerAop"></bean>
21
22   <!-- 配置切面 -->
23   <aop:config>
24     <!-- 定义切入点 -->
25     <aop:pointcut expression="execution(public void addUser(entity
26       y.User))" id="pointcut"/>
27     <!-- 引入包含增强方法的bean -->
28     <aop:aspect ref="userLoggerAop">
29       <!-- 将before()方法定义为前置增强, 并引用pointcut切入点 -->
30       <aop:before method="before" pointcut-ref="pointcut"/>
31       <!-- 将after()方法定义为后置增强, 并引用pointcut切入点 -->
32       <aop:after method="after" pointcut-ref="pointcut"/>
33       <!-- 通过returning属性指定明为result的参数注入返回值 -->
34       <aop:after-returning method="afterReturning" pointcut-ref="pointcut"/>
35     </aop:aspect>
36   </aop:config>
37 </beans>
```

```
32 <aop:after-returning method="after" pointcut-ref="pointcut" re
    turning="result"/>
33 </aop:aspect>
34 </aop:config>
35 </beans>
```

测试:

```
1 package test;
2
3 import org.junit.jupiter.api.Test;
4 import org.springframework.context.ApplicationContext;
5 import org.springframework.context.support.ClassPathXmlApplicati
    onContext;
6 import entity.User;
7 import service.UserService;
8
9 public class AopLoggerTest {
10     @Test
11     public void test() {
12         ApplicationContext ctx = new ClassPathXmlApplicationContext("a
            pplicationContext.xml");
13         UserService service = (UserService) ctx.getBean("service");
14         User user = new User();
15         user.setId(1);
16         user.setName("test");
17         user.setPwd("123456");
18         service.addUser(user);
19     }
20 }
```

运行结果:

```
Console  [JUnit]  Git Staging  Servers
<terminated> AopLoggerTesttest (1) [JUnit] C:\Program Files\Java8\jdk\bin\javaw.exe (2019年2月21日 上午9:11:57)
02-21 09:11:59[INFO]org.springframework.beans.factory.support.DefaultListableBeanFactory
-Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBean
02-21 09:12:00[INFO]aop.UserLoggerAop
-调用service.impl.UserServiceImpl@73e9cf30的addUser方法，方法的参数是：{entity.User@6221a451}
添加到数据库中成功
02-21 09:12:00[INFO]aop.UserLoggerAop
-调用service.impl.UserServiceImpl@73e9cf30的addUser方法，方法的返回结果：null
```

第二章、Ioc和AOP

案例-使用构造注入

设值注入（set注入）和构造注入的区别：

设值注入（set注入）	构造注入
通过setter访问实现	通过构造方法实现
灵活性好，但是setter方法太多	灵活性太差，仅依靠重载，限制太多
时效性差	时效性好
通过无参构造实例化	通过匹配的构造方法实例化（建议保留无参构造）

实体类：

```
1 package entity;
2 /**
3  * 实体类
4  * @author 阿华
5  *
6  */
7 public class User {
8     private Integer id;
9     private String name;
10    private String pwd;
11
12    public User() {
13        super();
14    }
15    public User(Integer id, String name, String pwd) {
16        super();
```

```
17  this.id = id;
18  this.name = name;
19  this.pwd = pwd;
20  }
21
22  public Integer getId() {
23  return id;
24  }
25  public void setId(Integer id) {
26  this.id = id;
27  }
28  public String getName() {
29  return name;
30  }
31  public void setName(String name) {
32  this.name = name;
33  }
34  public String getPwd() {
35  return pwd;
36  }
37  public void setPwd(String pwd) {
38  this.pwd = pwd;
39  }
40  }
```

dao接口:

```
1  package dao;
2
3  import entity.User;
4
5  /**
6   * 用户数据操作接口
7   * @author 阿华
8   *
9   */
```

```
10 public interface UserDao {
11     /**
12      * 保存
13      */
14     public void save(User user);
15 }
16
```

dao实现类:

```
1 package dao.impl;
2
3 import dao.UserDao;
4 import entity.User;
5 /**
6  * 实现类
7  * @author 阿华
8  *
9  */
10 public class UserDaoImpl implements UserDao {
11     //模拟去数据库中添加数据
12     @Override
13     public void save(User user) {
14         System.out.println("添加到数据库中成功");
15     }
16 }
```

service接口:

```
1 package service;
2
3 import entity.User;
4
5 public interface UserService {
6     /**
7      * 添加用户
8      */
9 }
```

```
9  public void addUser(User user);
10 }
11
```

service实现类:

```
1  package service.impl;
2
3  import dao.UserDao;
4  import entity.User;
5  import service.UserService;
6
7  public class UserServiceImpl implements UserService{
8      //创建dao层对象，用于Spring进行注入
9      private UserDao dao;
10
11     //无参构造
12     public UserServiceImpl() {
13
14     }
15     //有参构造
16     public UserServiceImpl(UserDao dao) {
17         super();
18         this.dao = dao;
19     }
20
21     @Override
22     public void addUser(User user) {
23         dao.save(user);
24     }
25 }
```

spring配置文件:

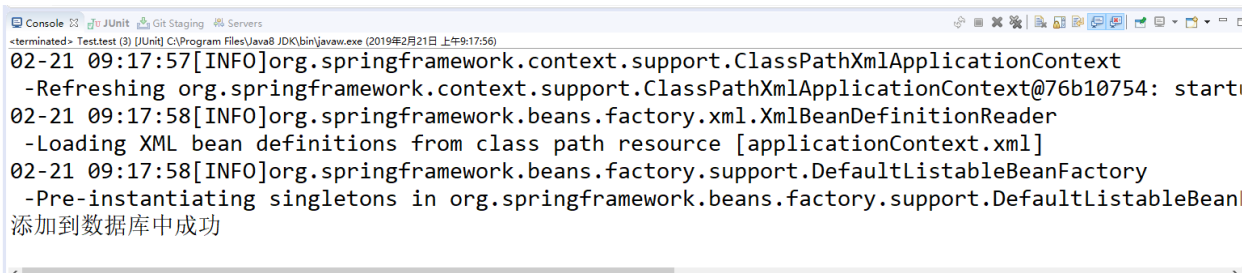
```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!-- Spring配置文件的根标签 -->
3  <!--
```

```
4  xmlns: 引入的文件路径
5  xsi:schemaLocation: 对应约束文件的路径
6  -->
7  <beans xmlns="http://www.springframework.org/schema/beans"
8  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
9  xsi:schemaLocation="
10 http://www.springframework.org/schema/beans
11 http://www.springframework.org/schema/beans/spring-beans-3.2.x
sd ">
12
13  <!--
14  注入的对比:
15  设值注入 (set注入) 构造注入
16  通过setter访问实现 通过构造方法实现
17  灵活性好, 但是setter方法太多 灵活性太差, 仅依靠重载限制太多
18  时效性差 时效性好
19  通过无参构造实例化 通过匹配的构造方法实例化 (强烈建议保留无参构造)
20  -->
21  <!-- 实例化dao对象 -->
22  <bean id="userDao" class="dao.impl.UserDaoImpl"></bean>
23
24  <!-- 实例化service对象 -->
25  <bean id="userService" class="service.impl.UserServiceImpl">
26  <!--
27  通过定义带参构造为userService的dao属性赋值
28  constructor-arg: 构造注入
29  -->
30  <constructor-arg>
31  <!-- 引入id为用户Dao的对象, 为用户Service的dao对象赋值
32  bean: 对象
33  -->
34  <ref bean="userDao"/>
35  </constructor-arg>
36  </bean>
37  </beans>
```

测试代码：

```
1 package test;
2
3 import org.springframework.context.ApplicationContext;
4 import org.springframework.context.support.ClassPathXmlApplication
  onContext;
5
6 import entity.User;
7 import service.UserService;
8
9 public class Test {
10
11     @org.junit.jupiter.api.Test
12     public void test() {
13         ApplicationContext ctx = new ClassPathXmlApplicationContext("a
  pplicationContext.xml");
14         UserService service = (UserService)
  ctx.getBean("userService");
15         User user = new User();
16         user.setId(1);
17         user.setName("test");
18         user.setPwd("123456");
19         service.addUser(user);
20
21     }
22 }
23
```

运行结果：



```
Console  JUnit  Git Staging  Servers
<terminated> Test.test (3) [JUnit] C:\Program Files\Java\jdk-11\bin\javaw.exe (2019年2月21日 上午9:17:56)
02-21 09:17:57[INFO]org.springframework.context.support.ClassPathXmlApplicationContext
  -Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@76b10754: start
02-21 09:17:58[INFO]org.springframework.beans.factory.xml.XmlBeanDefinitionReader
  -Loading XML bean definitions from class path resource [applicationContext.xml]
02-21 09:17:58[INFO]org.springframework.beans.factory.support.DefaultListableBeanFactory
  -Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBean
  添加到数据库中成功
```


案例-使用P命名空间注入属性值

特点：使用属性，而不是bean的子元素的形式配置bean的属性，从而简化了配置代码

对“setter方法注入”进行简化，替换<property name="属性名">，而是在<bean p:属性名="普通值" p:属性名-ref="引用值">

p命名空间使用前提，必须添加命名空间

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xmlns:p="http://www.springframework.org/schema/p"
5       xsi:schemaLocation="http://www.springframework.org/sche
```

实体类：

```
1 package entity;
2 /**
3  * 实体类
4  * @author 阿华
5  *
6  */
7 public class User {
8     private Integer id;
9     private String name;
10    private String pwd;
11
12    public User() {
13        super();
14    }
15    public User(Integer id, String name, String pwd) {
16        super();
17        this.id = id;
18        this.name = name;
19        this.pwd = pwd;
20    }
21
22    public Integer getId() {
23        return id;
```

```
24  }
25  public void setId(Integer id) {
26      this.id = id;
27  }
28  public String getName() {
29      return name;
30  }
31  public void setName(String name) {
32      this.name = name;
33  }
34  public String getPwd() {
35      return pwd;
36  }
37  public void setPwd(String pwd) {
38      this.pwd = pwd;
39  }
40  }
41
```

dao接口:

```
1  package dao;
2
3  import entity.User;
4
5  /**
6   * 用户数据操作接口
7   * @author 阿华
8   *
9   */
10 public interface UserDao {
11     /**
12      * 保存
13      */
14     public void save(User user);
15 }
```

dao实现类:

```
1 package dao.impl;
2
3 import dao.UserDao;
4 import entity.User;
5 /**
6  * 实现类
7  * @author 阿华
8  *
9  */
10 public class UserDaoImpl implements UserDao {
11     //模拟去数据库中添加数据
12     @Override
13     public void save(User user) {
14         System.out.println("添加到数据库中成功");
15         System.out.println("存储数据
16 为: "+user.getId()+" "+user.getName()+" "+user.getPwd());
17     }
18 }
```

service接口:

```
1 package service;
2
3 import entity.User;
4
5 public interface UserService {
6     /**
7     * 添加用户
8     */
9     public void addUser(User user);
10 }
11
```

service实现类:

```
1 package service.impl;
2
3 import dao.UserDao;
4 import entity.User;
5 import service.UserService;
6
7 public class UserServiceImpl implements UserService{
8     //创建dao层对象，用于Spring进行注入
9     private UserDao dao;
10
11     //无参构造
12     public UserServiceImpl() {
13
14     }
15     //有参构造
16     public UserServiceImpl(UserDao dao) {
17         super();
18         this.dao = dao;
19     }
20
21     //spring容器注入(bean文件注入)
22     public void setDao(UserDao dao) {
23         this.dao = dao;
24     }
25
26     @Override
27     public void addUser(User user) {
28         dao.save(user);
29     }
30 }
31
```

spring配置文件

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!-- Spring配置文件的根标签 -->
3 <!--
4   xmlns: 引入的文件路径
5   xsi:schemaLocation: 对应约束文件的路径
6   -->
7 <beans xmlns="http://www.springframework.org/schema/beans"
8   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
9   xmlns:p="http://www.springframework.org/schema/p"
10  xsi:schemaLocation="http://www.springframework.org/schema/beans
11  http://www.springframework.org/schema/beans/spring-beans-3.2.xsd">
12
13  <!-- 实例化User用户 -->
14  <!-- p命名空间注入属性值
15  语法: p:属性值(属性名) ="注入的内容"
16  特点: 使用属性, 而不是bean的子元素的形式配置bean的属性, 从而简化了配置代码
17  -->
18  <bean id="user" class="entity.User"
19  p:id="1" p:name="张三" p:pwd="123456"></bean>
20
21  <!-- 实例化dao -->
22  <bean id="userDao" class="dao.impl.UserDaoImpl"></bean>
23
24  <!-- 实例化service -->
25  <!-- p:dao-ref: 采用p命名空间注入dao对象 -->
26  <bean id="userService" class="service.impl.UserServiceImpl"
27  p:dao-ref="userDao"></bean>
28 </beans>

```

测试:

```

1 package test;
2
3 import org.springframework.context.ApplicationContext;

```

```

4 import org.springframework.context.support.ClassPathXmlApplication
onContext;

5

6 import entity.User;
7 import service.UserService;
8

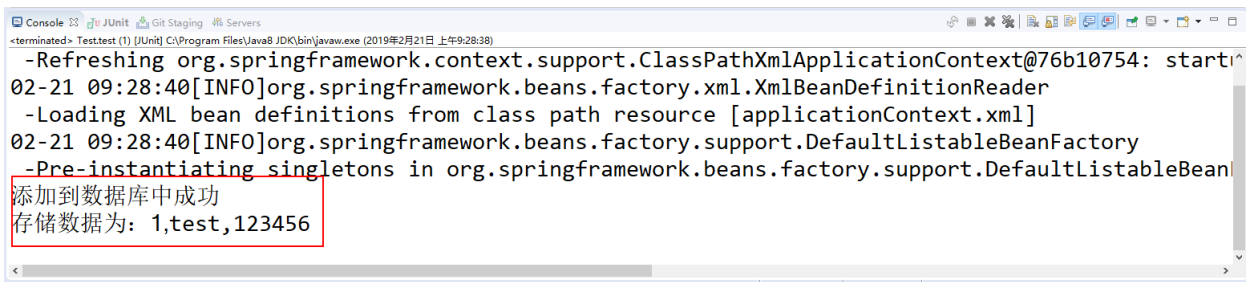
9 public class Test {
10

11     @org.junit.jupiter.api.Test
12     public void test() {
13         ApplicationContext ctx = new ClassPathXmlApplicationContext("a
pplicationContext.xml");
14         UserService service = (UserService)
ctx.getBean("userService");
15         User user = (User)ctx.getBean("user");
16         user.setId(1);
17         user.setName("test");
18         user.setPwd("123456");
19         service.addUser(user);
20

21     }
22 }
23

```

运行结果：



```

Console  JUnit  Git Staging  Servers
<terminated> Test.test (1) [JUnit] C:\Program Files\Java\jdk-8.0.602\bin\javaw.exe (2019年2月21日 上午9:28:38)
-Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@76b10754: startup...
02-21 09:28:40[INFO]org.springframework.beans.factory.xml.XmlBeanDefinitionReader
-Loading XML bean definitions from class path resource [applicationContext.xml]
02-21 09:28:40[INFO]org.springframework.beans.factory.support.DefaultListableBeanFactory
-Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBean
添加到数据库中成功
存储数据为: 1,test,123456

```

案例-注入不同数据类型

实体类：

```

1 package entity;
2

```

```

3 public class User {
4     private String userName;
5
6     public User() {
7         super();
8     }
9
10    public User(String userName) {
11        super();
12        this.userName = userName;
13    }
14
15    public String getUserName() {
16        return userName;
17    }
18
19    public void setUserName(String userName) {
20        this.userName = userName;
21    }
22 }
23

```

测试实体类：

```

1 package entity;
2
3 import java.util.List;
4 import java.util.Map;
5 import java.util.Properties;
6 import java.util.Set;
7
8 /**
9  * 测试注入不同数据
10  * @author 阿华
11  *
12  */

```

```
13 public class TestEntity {
14     private String speStringCharString1;//特殊字符1
15     private String speStringCharString2;//特殊字符2
16     private User innerBean;//JavaBean类型
17     private List<String> list;//List类型
18     private String[] array;//数组
19     private Set<String> set;//Set类型
20     private Map<String, String> map;//Map类型
21     private Properties props;//Properties类型
22     private String emptyValue;//空的字符串
23     private String nullValue;//null值
24
25     //set方法
26     public void setSpeStringCharString1(String speStringCharString
27 1) {
28         this.speStringCharString1 = speStringCharString1;
29     }
30     public void setSpeStringCharString2(String speStringCharString
31 2) {
32         this.speStringCharString2 = speStringCharString2;
33     }
34     public void setInnerBean(User innerBean) {
35         this.innerBean = innerBean;
36     }
37     public void setList(List<String> list) {
38         this.list = list;
39     }
40     public void setArray(String[] array) {
41         this.array = array;
42     }
43     public void setSet(Set<String> set) {
44         this.set = set;
45     }
46     public void setMap(Map<String, String> map) {
47         this.map = map;
48     }
49 }
```



```

47 public void setProps(Properties props) {
48     this.props = props;
49 }
50 public void setEmptyValue(String emptyValue) {
51     this.emptyValue = emptyValue;
52 }
53 public void setNullValue(String nullValue) {
54     this.nullValue = nullValue;
55 }
56
57 public void showValue() {
58     System.out.println(this.speStringCharString1);
59     System.out.println(this.speStringCharString2);
60     System.out.println(this.innerBean);
61     System.out.println(this.list.get(0));
62     System.out.println(this.array[0]);
63     System.out.println(this.set);
64     System.out.println(this.map);
65     System.out.println(this.props);
66     System.out.println "["+this.emptyValue+"]";
67     System.out.println "["+this.nullValue+"]";
68 }
69 }

```

spring配置文件

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!-- Spring配置文件的根标签 -->
3 <!--
4     xmlns: 引入的文件路径
5     xsi:schemaLocation: 对应约束文件的路径
6     -->
7 <beans xmlns="http://www.springframework.org/schema/beans"
8     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
9     xmlns:p="http://www.springframework.org/schema/p"

```

```
10  xsi:schemaLocation="http://www.springframework.org/schema/bean
   S
11  http://www.springframework.org/schema/beans/spring-beans-3.2.x
   sd">
12
13  <!-- 实例化TestEntity对象 -->
14  <bean id="testEntity" class="entity.TestEntity">
15  <!-- 注入特殊字符1 使用转义标记<![CDATA[]]>形式处理xml特殊字符 -->
16  <property name="speStringCharString1">
17  <value><![CDATA[A&HUA]]></value>
18  </property>
19
20  <!-- 注入特殊字符2 把xml特殊字符替换为实体引用-->
21  <property name="speStringCharString2">
22  <value>A&HUA</value>
23  </property>
24
25  <!-- 注入内部对象 -->
26  <property name="innerBean">
27  <bean class="entity.User">
28  <property name="userName">
29  <value>AHUA</value>
30  </property>
31  </bean>
32  </property>
33
34  <!-- 注入List类型 -->
35  <property name="list">
36  <list>
37  <value>哈哈</value>
38  <value>呵呵</value>
39  <value>罗拉</value>
40  <value>嘿嘿</value>
41  </list>
42  </property>
43
```

```
44 <!-- 注入数组类型 -->
45 <property name="array">
46 <list>
47 <value>哈哈array</value>
48 <value>罗拉array</value>
49 <value>嘿嘿array</value>
50 </list>
51 </property>
52
53 <!-- 注入Set类型 -->
54 <property name="set">
55 <set>
56 <value>哈哈set</value>
57 <value>罗拉set</value>
58 <value>嘿嘿set</value>
59 </set>
60 </property>
61
62 <!-- 注入Map类型 -->
63 <property name="map">
64 <map>
65 <!-- 定义map中的键值对 -->
66 <entry>
67 <!-- 键 -->
68 <key>
69 <value>hh</value>
70 </key>
71 <!-- 值 -->
72 <value>
73 哈哈
74 </value>
75 </entry>
76 <entry>
77 <key>
78 <value>ll</value>
```

```

79 </key>
80 <value>
81 拉拉
82 </value>
83 </entry>
84 </map>
85 </property>
86
87 <!-- 注入Properties类型 -->
88 <property name="props">
89 <props>
90 <!-- 定义Properties的键值对 -->
91 <prop key="dd">弟弟</prop>
92 <prop key="gg">哥哥</prop>
93 </props>
94 </property>
95
96 <!-- 注入空字符串 -->
97 <property name="emptyValue">
98 <value></value>
99 </property>
100
101 <!-- 注入null空值 -->
102 <property name="nullValue">
103 <null/>
104 </property>
105 </bean>
106 </beans>

```

测试:

```

1 package test;
2
3 import org.springframework.context.ApplicationContext;
4 import org.springframework.context.support.ClassPathXmlApplicati
onContext;
5

```

```

6 import entity.TestEntity;
7
8 public class test {
9
10     @org.junit.jupiter.api.Test
11     public void test() {
12         ApplicationContext ctx = new ClassPathXmlApplicationContext("a
13         pplicationContext.xml");
14         TestEntity entity =(TestEntity)ctx.getBean("testEntity");
15         entity.showValue();
16     }
17 }
18

```

运行结果:

```

Console  JUnit  Git Staging  Servers
<terminated> Test.test [JUnit] C:\Program Files\Java8\JDK\bin\javaw.exe (2019年2月21日 上午9:31:30)
-Refreshing org.springframework.context.support.ClassPathXmlApplicationContext
02-21 09:31:31[INFO]org.springframework.beans.factory.xml.XmlBeanDefinitionRe
-Loading XML bean definitions from class path resource [applicationContext.xml]
02-21 09:31:31[INFO]org.springframework.beans.factory.support.DefaultListable
-Pre-instantiating singletons in org.springframework.beans.factory.support.L
A&HUA
A&HUA
entity.User@9597028
哈哈
哈哈array
[哈哈set, 罗拉set, 嘿嘿set]
{hh=
, ll=
}
{dd=弟弟, gg=哥哥}
[]
[null]
哈哈
拉拉

```

案例-定义异常抛出增强

实体类:

```

1 package entity;
2 /**
3  * 实体类

```

```
4  * @author 阿华
5  *
6  */
7  public class User {
8      private Integer id;
9      private String name;
10     private String pwd;
11
12     public User() {
13         super();
14     }
15     public User(Integer id, String name, String pwd) {
16         super();
17         this.id = id;
18         this.name = name;
19         this.pwd = pwd;
20     }
21
22     public Integer getId() {
23         return id;
24     }
25     public void setId(Integer id) {
26         this.id = id;
27     }
28     public String getName() {
29         return name;
30     }
31     public void setName(String name) {
32         this.name = name;
33     }
34     public String getPwd() {
35         return pwd;
36     }
37     public void setPwd(String pwd) {
38         this.pwd = pwd;
```

```
39     }  
40 }  
41
```

dao接口:

```
1 package dao;  
2  
3 import entity.User;  
4  
5 /**  
6  * 用户数据操作接口  
7  * @author 阿华  
8  *  
9  */  
10 public interface UserDao {  
11     /**  
12     * 保存  
13     */  
14     public void save(User user);  
15 }  
16
```

dao实现类:

```
1 package dao.impl;  
2  
3 import dao.UserDao;  
4 import entity.User;  
5 /**  
6  * 实现类  
7  * @author 阿华  
8  *  
9  */  
10 public class UserDaoImpl implements UserDao {  
11     //模拟去数据库中添加数据  
12     @Override
```

```

13  public void save(User user) {
14      System.out.println("UserDaoImpl类处理增强方法开始执行");
15      System.out.println("添加到数据库中成功");
16      throw new RuntimeException("未测试程序运行效果，模拟产生异常-User
    DaoImpl类");
17  }
18  }
19

```

service接口:

```

1  package service;
2
3  import entity.User;
4
5  public interface UserService {
6      /**
7       * 添加用户
8       */
9      public void addUser(User user);
10 }
11

```

service实现类:

```

1  package service.impl;
2
3  import dao.UserDao;
4  import entity.User;
5  import service.UserService;
6
7  public class UserServiceImpl implements UserService{
8      //创建dao层对象，用于Spring进行注入
9      private UserDao dao;
10
11     public void setDao(UserDao dao) {
12         this.dao = dao;
13     }
14 }

```



```

13     }
14
15     @Override
16     public void addUser(User user) {
17         dao.save(user);
18     }
19 }
20

```

aop切面类:

```

1  package aop;
2  /**
3   * 切面增强处理的方法所在的类
4   * @author 阿华
5   */
6
7  import org.apache.log4j.Logger;
8  import org.aspectj.lang.JoinPoint;
9
10 public class ErrorLoggerAop {
11     private static final Logger log=Logger.getLogger(ErrorLoggerAop.class);
12     /**
13      * 定义后置异常处理增强方法
14      */
15     public void afterThrowing(JoinPoint jp, RuntimeException e) {
16         System.out.println("ErrorLoggerAop类开始执行");
17         log.error("-----"+jp.getSignature().getName()+"方法发生异常: "+e+"-ErrorLoggerAop-----");
18         System.out.println("ErrorLoggerAop类执行完毕");
19     }
20 }
21

```

spring配置文件

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!-- Spring配置文件的根标签 -->
3 <!--
4   xmlns: 引入的文件路径
5   xsi:schemaLocation: 对应约束文件的路径
6   -->
7 <beans xmlns="http://www.springframework.org/schema/beans"
8   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
9   xmlns:aop="http://www.springframework.org/schema/aop"
10   xsi:schemaLocation="
11   http://www.springframework.org/schema/beans
12   http://www.springframework.org/schema/beans/spring-beans-3.2.x
13   sd
14   http://www.springframework.org/schema/aop
15   http://www.springframework.org/schema/aop/spring-aop-3.2.xsd">
16 <!--
17   aop:after-throwing: 用于定义后置异常处理增强
18   -->
19
20 <!-- 实例化dao -->
21 <bean id="udao" class="dao.impl.UserDaoImpl"></bean>
22
23 <!-- 注入(实例化)service -->
24 <bean id="service" class="service.impl.UserServiceImpl">
25   <property name="dao" ref="udao"></property>
26 </bean>
27
28 <!-- 声明增强方法所在的bean -->
29 <bean id="errorLoggerAop" class="aop.ErrorLoggerAop"></bean>
30
31 <!-- 配置切面 -->
32 <aop:config>
33   <!-- 定义切入点 -->
34   <aop:pointcut id="pointcut" expression="execution(* service.UserService.*(..))"></aop:pointcut>
35
```

```

36 <!-- 引入包含增强方法bean
37 pointcut-ref: 切入点
38 -->
39 <aop:aspect ref="errorLoggerAop">
40 <!-- 将afterThrowing 定义为异常抛出增强，并引入pointcut切入点 --
41 <aop:after-throwing method="afterThrowing" pointcut-ref="pointcut" throwing="e"/>
42 </aop:aspect>
43 </aop:config>
44 </beans>

```

测试：

```

1 package test;
2
3 import org.junit.jupiter.api.Test;
4 import org.springframework.context.ApplicationContext;
5 import org.springframework.context.support.ClassPathXmlApplicationContext;
6
7 import entity.User;
8 import service.UserService;
9
10 public class ErrorLoggerTest {
11     @Test
12     public void test() {
13         ApplicationContext ctx = new ClassPathXmlApplicationContext("applicationContext.xml");
14         UserService service = (UserService) ctx.getBean("service");
15         User user = new User();
16         user.setId(1);
17         user.setName("test");
18         user.setPwd("123456");
19         service.addUser(user);
20
21     }
22 }

```

运行结果：

```

Console  JUnit  Git Staging  Servers
<terminated> ErrorLoggerTest (1) [JUnit] C:\Program Files\Java8\jdk\bin\java.exe (2019年2月21日 上午9:34:57)
-Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@76b10754: start
02-21 09:34:59[INFO]org.springframework.beans.factory.xml.XmlBeanDefinitionReader
-Loading XML bean definitions from class path resource [applicationContext.xml]
02-21 09:34:59[INFO]org.springframework.beans.factory.support.DefaultListableBeanFactory
-Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBean
 UserDaoImpl类处理增强方法开始执行
添加到数据库中成功
ErrorLoggerAop类开始执行
02-21 09:34:59[ERROR]aop.ErrorLoggerAop
-----addUser方法发生异常: java.lang.RuntimeException: 未测试程序运行效果，模拟产生异常-U
ErrorLoggerAop类执行完毕

```

```

Console  JUnit  Git Staging  Servers
Finished after 0.723 seconds
Runs: 1/1  Errors: 1  Failures: 0
ErrorLoggerTest (Run: JUnit 5) (0.614 s)
test() (0.614 s)
Failure Trace
java.lang.RuntimeException: 未测试程序运行效果，模拟产生异常-UserDaoImpl类
at dao.impl.UserDaoImpl.save(UserDaoImpl.java:16)
at service.impl.UserServiceImpl.addUser(UserServiceImpl.java:17)
at org.springframework.aop.support.AopUtils.invokeJoinpointUsingReflection(AopUtils.java:317)
at org.springframework.aop.framework.ReflectiveMethodInvocation.invokeJoinpoint(ReflectiveMethodInvocation.java:183)
at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:150)
at org.springframework.aop.aspectj.AspectJAfterThrowingAdvice.invoke(AspectJAfterThrowingAdvice.java:55)
at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:172)
at org.springframework.aop.interceptor.ExposeInvocationInterceptor.invoke(ExposeInvocationInterceptor.java:91)
at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:172)
at org.springframework.aop.framework.JdkDynamicAopProxy.invoke(JdkDynamicAopProxy.java:204)
at com.sun.proxy.$Proxy8.addUser(Unknown Source)
at test.ErrorLoggerTest.test(ErrorLoggerTest.java:19)
at java.util.ArrayList.forEach(Unknown Source)
at java.util.ArrayList.forEach(Unknown Source)

```

案例-定义最终增强

实体类：

```

1 package entity;
2
3 /**
4  * 实体类
5  * @author 阿华
6  *
7  */
8 public class User {
9     private Integer id;
10    private String name;
11    private String pwd;
12
13    public User() {
14        super();
15    }
16 }

```

```
15  public User(Integer id, String name, String pwd) {
16      super();
17      this.id = id;
18      this.name = name;
19      this.pwd = pwd;
20  }
21
22  public Integer getId() {
23      return id;
24  }
25  public void setId(Integer id) {
26      this.id = id;
27  }
28  public String getName() {
29      return name;
30  }
31  public void setName(String name) {
32      this.name = name;
33  }
34  public String getPwd() {
35      return pwd;
36  }
37  public void setPwd(String pwd) {
38      this.pwd = pwd;
39  }
40  }
41
```

dao接口:

```
1  package dao;
2
3  import entity.User;
4
5  /**
6   * 用户数据操作接口
```

```

7  * @author 阿华
8  *
9  */
10 public interface UserDao {
11     /**
12      * 保存
13      */
14     public void save(User user);
15 }
16

```

dao实现类:

```

1  package dao.impl;
2
3  import dao.UserDao;
4  import entity.User;
5  /**
6   * 实现类
7   * @author 阿华
8   *
9   */
10 public class UserDaoImpl implements UserDao {
11     //模拟去数据库中添加数据
12     @Override
13     public void save(User user) {
14         System.out.println("UserDaoImpl类处理增强方法开始执行");
15         System.out.println("添加到数据库中成功");
16         throw new RuntimeException("未测试程序运行效果，模拟产生异常");
17     }
18 }
19

```

service接口:

```

1  package service;
2

```

```
3 import entity.User;
4
5 public interface UserService {
6     /**
7      * 添加用户
8      */
9     public void addUser(User user);
10 }
11
```

service实现类:

```
1 package service.impl;
2
3 import dao.UserDao;
4 import entity.User;
5 import service.UserService;
6
7 public class UserServiceImpl implements UserService{
8     //创建dao层对象，用于Spring进行注入
9     private UserDao dao;
10
11     public void setDao(UserDao dao) {
12         this.dao = dao;
13     }
14
15     @Override
16     public void addUser(User user) {
17         dao.save(user);
18     }
19 }
20
```

aop切面类:

```
1 package aop;
2 /**
```

```

3  * 切面增强处理的方法所在的类
4  * @author 阿华
5  */
6
7  import org.apache.log4j.Logger;
8  import org.aspectj.lang.JoinPoint;
9
10 public class AfterLoggerAop {
11     private static final Logger log=Logger.getLogger(AfterLoggerAop.class);
12     /**
13      * 定义后置异常处理增强方法
14      */
15     public void afterLogger(JoinPoint jp) {
16         System.out.println("ErrorLoggerAop类开始执行");
17         log.error("-----"+jp.getSignature().getName()+"方法发生异常: "+"-----");
18         System.out.println("ErrorLoggerAop类执行完毕");
19     }
20 }
21

```

spring配置文件

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <!-- Spring配置文件的根标签 -->
3  <!--
4      xmlns: 引入的文件路径
5      xsi:schemaLocation: 对应约束文件的路径
6      -->
7  <beans xmlns="http://www.springframework.org/schema/beans"
8      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
9      xmlns:aop="http://www.springframework.org/schema/aop"
10     xsi:schemaLocation="
11     http://www.springframework.org/schema/beans

```



```
12 http://www.springframework.org/schema/beans/spring-beans-3.2.x
sd
13 http://www.springframework.org/schema/aop
14 http://www.springframework.org/schema/aop/spring-aop-3.2.xsd">
15
16 <!--
17 aop:after: 最终增强，无论方法是否抛出异常都会在目标方法后面织入增强
处理
18 （无论怎样都会执行的类似于try catch finally 中的finally作用）
19 作用：用来释放资源
20 -->
21
22 <!-- 实例化dao -->
23 <bean id="uDao" class="dao.impl.UserDaoImpl"></bean>
24
25 <!-- 注入(实例化)service -->
26 <bean id="uService" class="service.impl.UserServiceImpl">
27 <property name="dao" ref="uDao"></property>
28 </bean>
29
30 <!-- 声明增强方法所在的bean -->
31 <bean id="afterLoggerAop" class="aop.AfterLoggerAop"></bean>
32
33 <!-- 配置切面 -->
34 <aop:config>
35 <!-- 定义切入点 -->
36 <aop:pointcut id="pointcut" expression="execution(* service.User
erService.*(..))"></aop:pointcut>
37
38 <!-- 引入包含增强方法bean
39 pointcut-ref: 切入点
40 -->
41 <aop:aspect ref="afterLoggerAop">
42 <aop:after method="afterLogger" pointcut-ref="pointcut"/>
43 </aop:aspect>
44 </aop:config>
45 </beans>
```

测试:

```
1 package test;
2
3 import org.junit.jupiter.api.Test;
4 import org.springframework.context.ApplicationContext;
5 import org.springframework.context.support.ClassPathXmlApplication
  onContext;
6
7 import entity.User;
8 import service.UserService;
9
10 public class AfterLoggerTest {
11     @Test
12     public void test() {
13         ApplicationContext ctx = new ClassPathXmlApplicationContext("a
  pplicationContext.xml");
14         UserService service =(UserService)ctx.getBean("uService");
15         User user = new User();
16         user.setId(1);
17         user.setName("test");
18         user.setPwd("123456");
19         service.addUser(user);
20
21     }
22 }
23
```

运行结果:

```
Console JUnit Git Staging Servers
<terminated> AfterLoggerTest.test [JUnit] C:\Program Files\Java\jdk-8\bin\javaw.exe (2019年2月21日 上午9:44:27)

02-21 09:44:29[INFO]org.springframework.beans.factory.xml.XmlBeanDefinitionReader
-Loading XML bean definitions from class path resource [applicationContext.xml]
02-21 09:44:30[INFO]org.springframework.beans.factory.support.DefaultListableBeanFactory
-Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBean
 UserDaoImpl类处理增强方法开始执行
添加到数据库中成功
ErrorLoggerAop类开始执行
02-21 09:44:30[ERROR]aop.AfterLoggerAop
-----addUser方法发生异常: -----
ErrorLoggerAop类执行完毕
```

```
Console JUnit Git Staging Servers
Finished after 1.481 seconds

Runs: 1/1 Errors: 1 Failures: 0

AfterLoggerTest [Runner: JUnit 5] (1.340 s)
test() (1.340 s)

Failure Trace
java.lang.RuntimeException: 未测试程序运行效果，模拟产生异常
at dao.impl.UserDaoImpl.save(UserDaoImpl.java:16)
at service.impl.UserServiceImpl.addUser(UserServiceImpl.java:17)
at org.springframework.aop.support.AopUtils.invokeJoinpointUsingReflection(AopUtils.java:317)
at org.springframework.aop.framework.ReflectiveMethodInvocation.invokeJoinpoint(ReflectiveMethodInvocation.java:183)
at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:150)
at org.springframework.aop.aspectj.AspectJAfterAdvice.invoke(AspectJAfterAdvice.java:42)
at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:172)
at org.springframework.aop.interceptor.ExposeInvocationInterceptor.invoke(ExposeInvocationInterceptor.java:91)
at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:172)
at org.springframework.aop.framework.JdkDynamicAopProxy.invoke(JdkDynamicAopProxy.java:204)
at com.sun.proxy.$Proxy8.addUser(Unknown Source)
at test.AfterLoggerTest.test(AfterLoggerTest.java:19)
at java.util.ArrayList.forEach(Unknown Source)
at java.util.ArrayList.forEach(Unknown Source)
```

案例-定义环绕增强

实体类：

```
1 package entity;
2 /**
3  * 实体类
4  * @author 阿华
5  *
6  */
7 public class User {
8     private Integer id;
9     private String name;
10    private String pwd;
11
12    public User() {
13        super();
14    }
15
16    public User(Integer id, String name, String pwd) {
17        super();
18        this.id = id;
```

```
18  this.name = name;
19  this.pwd = pwd;
20  }
21
22  public Integer getId() {
23  return id;
24  }
25  public void setId(Integer id) {
26  this.id = id;
27  }
28  public String getName() {
29  return name;
30  }
31  public void setName(String name) {
32  this.name = name;
33  }
34  public String getPwd() {
35  return pwd;
36  }
37  public void setPwd(String pwd) {
38  this.pwd = pwd;
39  }
40  }
41
```

dao接口:

```
1  package dao;
2
3  import entity.User;
4
5  /**
6   * 用户数据操作接口
7   * @author 阿华
8   *
9   */
```

```
10 public interface UserDao {
11     /**
12      * 保存
13      */
14     public void save(User user);
15 }
16
```

dao实现类:

```
1 package dao.impl;
2
3 import dao.UserDao;
4 import entity.User;
5 /**
6  * 实现类
7  * @author 阿华
8  *
9  */
10 public class UserDaoImpl implements UserDao {
11     //模拟去数据库中添加数据
12     @Override
13     public void save(User user) {
14         System.out.println("添加到数据库中成功");
15     }
16 }
17
```

service接口:

```
1 package service;
2
3 import entity.User;
4
5 public interface UserService {
6     /**
7      * 添加用户
```

```

8  */
9  public void addUser(User user);
10 }
11

```

service实现类:

```

1  package service.impl;
2
3  import dao.UserDao;
4  import entity.User;
5  import service.UserService;
6
7  public class UserServiceImpl implements UserService{
8      //创建dao层对象，用于Spring进行注入
9      private UserDao dao;
10
11     public void setDao(UserDao dao) {
12         this.dao = dao;
13     }
14
15     @Override
16     public void addUser(User user) {
17         dao.save(user);
18     }
19 }
20

```

spring配置文件:

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <!-- Spring配置文件的根标签 -->
3  <!--
4      xmlns: 引入的文件路径
5      xsi:schemaLocation: 对应约束文件的路径
6      -->
7  <beans xmlns="http://www.springframework.org/schema/beans"

```

```
8  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
9  xmlns:aop="http://www.springframework.org/schema/aop"
10  xsi:schemaLocation="
11  http://www.springframework.org/schema/beans
12  http://www.springframework.org/schema/beans/spring-beans-3.2.x
sd
13  http://www.springframework.org/schema/aop
14  http://www.springframework.org/schema/aop/spring-aop-3.2.xsd">
15
16  <!--
17  环绕增强 aop:around
18  特点:
19  1.同样可以进行前后增强
20  2.功能相对更强大
21  3.可以控制目标方法是否执行，同时可以直接进行异常处理
22  -->
23  <!-- 实例化dao -->
24  <bean id="udao" class="dao.impl.UserDaoImpl"></bean>
25
26  <!-- 注入service -->
27  <bean id="service" class="service.impl.UserServiceImpl">
28  <property name="dao" ref="udao"></property>
29  </bean>
30
31  <!-- 声明增强方法所在的bean -->
32  <bean id="aroundLoggerAop" class="aop.AroundLoggerAop"></bean>
33
34  <!-- 配置切面 -->
35  <aop:config>
36  <!-- 定义切入点
37  expression: 切入的方法
38  -->
39  <aop:pointcut id="pointcut" expression="execution(* service.UserService.*(..))"></aop:pointcut>
40
41  <!-- 引入包含增强方法bean -->
```

```

42 <aop:aspect ref="aroundLoggerAop">
43 <!-- 使用环绕增强处理目标方法 -->
44 <aop:around method="aroundLogger" pointcut-ref="pointcut"/>
45 </aop:aspect>
46 </aop:config>
47 </beans>

```

aop切面类:

```

1 package aop;
2 /**
3  * 切面增强处理的方法所在的类
4  * @author 阿华
5  *
6  */
7
8 import org.apache.catalina.tribes.util.Arrays;
9 import org.apache.log4j.Logger;
10 import org.aspectj.lang.JoinPoint;
11 import org.aspectj.lang.ProceedingJoinPoint;
12
13 public class AroundLoggerAop {
14     private static final Logger log=Logger.getLogger(AroundLoggerAop.class);
15
16     public Object aroundLogger(ProceedingJoinPoint jp) throws Throwable {
17         /**
18          * 环绕前增强
19          */
20         log.info("调用"+jp.getTarget()+"的"+jp.getSignature().getName()+"
21             "方法，方法的参数是："+Arrays.toString(jp.getArgs()));
22
23         try {
24             //执行目标方法
25             Object result=jp.proceed();

```



```

26
27  /**
28   * 环绕后增强
29   */
30   log.info("调
用"+jp.getTarget()+"的"+jp.getSignature().getName()+
31   "方法，方法的返回值是："+result);
32   return result;
33   } catch (Exception e) {
34   e.printStackTrace();
35   log.error(jp.getSignature().getName()+"方法发生异常："+e);
36   throw e;
37   } finally {
38   System.out.println(jp.getSignature().getName()+"方法结束");
39   }
40
41   }
42   }
43

```

测试:

```

1  package test;
2
3  import org.junit.jupiter.api.Test;
4  import org.springframework.context.ApplicationContext;
5  import org.springframework.context.support.ClassPathXmlApplicati
onContext;
6
7  import entity.User;
8  import service.UserService;
9
10 public class AroundLoggerTest {
11     @Test
12     public void test() {
13         ApplicationContext ctx = new ClassPathXmlApplicationContext("a
pplicationContext.xml");

```

```

14  UserService service = (UserService) ctx.getBean("service");
15  User user = new User();
16  user.setId(1);
17  user.setName("test");
18  user.setPwd("123456");
19  service.addUser(user);
20
21  }
22  }
23

```

运行结果：

```

<terminated> AroundLoggerTest.test(2) [JUnit] C:\Program Files\Java\jdk-8.0.60\bin\javaw.exe (2019年2月21日 上午9:48:27)
-Loading XML bean definitions from class path resource [applicationContext.xml]
02-21 09:48:28[INFO]org.springframework.beans.factory.support.DefaultListableBeanFactory
-Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBeanFactory
02-21 09:48:29[INFO]aop.AroundLoggerAop
-调用service.impl.UserServiceImpl@55634720的addUser方法，方法的参数是：{entity.User@771a660}
添加到数据库中成功
02-21 09:48:29[INFO]aop.AroundLoggerAop
-调用service.impl.UserServiceImpl@55634720的addUser方法，方法的返回值是：null
addUser方法结束

```

Spring 常用注解

注解本身没有功能的，就和 xml 一样。注解和 xml 都是一种元数据，元数据即解释数据的数据，这就是所谓配置。

1.声明bean的注解

@Component 组件，没有明确的角色

@Service 在业务逻辑层使用（service层）

@Repository 在数据访问层使用（dao层）

@Controller 在展现层使用，控制器的声明（C）

2.注入bean的注解

@Autowired：由Spring提供

@Inject：由JSR-330提供

@Resource：由JSR-250提供

都可以注解在set方法和属性上，推荐注解在属性上（一目了然，少写代码）。

3.java配置类相关注解

@Configuration 声明当前类为配置类，相当于xml形式的Spring配置（类上）

@Bean 注解在方法上，声明当前方法的返回值为一个bean，替代xml中的方式（方法上）

@Configuration 声明当前类为配置类，其中内部组合了@Component注解，表明这个类是一个bean（类上）

@ComponentScan 用于对Component进行扫描，相当于xml中的（类上）

@WishlyConfiguration 为@Configuration与@ComponentScan的组合注解，可以替代这两个注解

4.切面（AOP）相关注解

Spring支持AspectJ的注解式切面编程。

@Aspect 声明一个切面（类上）

使用@After、@Before、@Around定义建言（advice），可直接将拦截规则（切点）作为参数。

@After 在方法执行之后执行（方法上）

@Before 在方法执行之前执行（方法上）

@Around 在方法执行之前与之后执行（方法上）

@PointCut 声明切点

在java配置类中使用@EnableAspectJAutoProxy注解开启Spring对AspectJ代理的支持（类上）

5.@Bean的属性支持

@Scope 设置Spring容器如何新建Bean实例（方法上，得有@Bean）

其设置类型包括：

Singleton （单例,一个Spring容器中只有一个bean实例，默认模式），

Protetype （每次调用新建一个bean），

Request （web项目中，给每个http request新建一个bean），

Session （web项目中，给每个http session新建一个bean），

GlobalSession （给每一个 global http session新建一个Bean实例）

@StepScope 在Spring Batch中还有涉及

@PostConstruct 由JSR-250提供，在构造函数执行完之后执行，等价于xml配置文件中bean的initMethod

@PreDestory 由JSR-250提供，在Bean销毁之前执行，等价于xml配置文件中bean的destroyMethod

6.@Value注解

@Value 为属性注入值（属性上）

支持如下方式的注入：

》注入普通字符

```
1 @Value("Michael Jackson")
2 String name;
```

》注入操作系统属性

```
1 @Value("#{systemProperties['os.name']}")
2 String osName;
```

》注入表达式结果

```
1 @Value("#{ T(java.lang.Math).random() * 100 }")
2 String randomNumber;
```

》注入其它bean属性

```
1 @Value("#{domeClass.name}")
2 String name;
```

》注入文件资源

```
1 @Value("classpath:com/hgs/hello/test.txt")
2 String Resource file;
3
```

》注入网站资源

```
1 @Value("http://www.cznovel.com")
2 Resource url;12
```

》注入配置文件

```
1
2 @Value("${book.name}")
3 String bookName;
```

注入配置使用方法：

```
1 ① 编写配置文件（test.properties）
2 book.name=《三体》
3
4 ② @PropertySource 加载配置文件(类上)
5 @PropertySource("classpath:com/hgs/hello/test/test.propertie")
6 ③ 还需配置一个PropertySourcesPlaceholderConfigurer的bean。
```

7.环境切换

@Profile 通过设定Environment的ActiveProfiles来设定当前context需要使用的配置环境。（类或方法上）

@Conditional Spring4中可以使用此注解定义条件话的bean，通过实现Condition接口，并重写matches方法，从而决定该bean是否被实例化。（方法上）

8.异步相关

@EnableAsync 配置类中，通过此注解开启对异步任务的支持，叙事性AsyncConfigurer接口（类上）

@Async 在实际执行的bean方法使用该注解来申明其是一个异步任务（方法上或类上所有的方法都将异步，需要@EnableAsync开启异步任务）

9.定时任务相关

@EnableScheduling 在配置类上使用，开启计划任务的支持（类上）

@Scheduled 来申明这是一个任务，包括cron,fixDelay,fixRate等类型（方法上，需先开启计划任务的支持）

10.@Enable*注解说明

这些注解主要用来开启对xxx的支持。

@EnableAspectJAutoProxy 开启对AspectJ自动代理的支持

@EnableAsync 开启异步方法的支持

@EnableScheduling 开启计划任务的支持

@EnableWebMvc 开启Web MVC的配置支持

@EnableConfigurationProperties 开启对@ConfigurationProperties注解配置Bean的支持

@EnableJpaRepositories 开启对SpringData JPA Repository的支持

@EnableTransactionManagement 开启注解式事务的支持

@EnableTransactionManagement 开启注解式事务的支持

@EnableCaching 开启注解式的缓存支持

11.测试相关注解

@RunWith 运行器，Spring中通常用于对JUnit的支持

```
1 @RunWith(SpringJUnit4ClassRunner.class)1
```

@ContextConfiguration 用来加载配置ApplicationContext，其中classes属性用来加载配置类

```
1 @ContextConfiguration(classes={TestConfig.class})1
```

装配Bean基于注解

- 注解：就是一个类，使用@注解名称
- 开发中：使用注解 取代 xml配置文件。

1. @Component取代<bean class="">

@Component("id") 取代 <bean id="" class="">

2.web开发, 提供3个@Component注解衍生注解 (功能一样) 取代<bean class="">

@Repository : dao层

@Service: service层

@Controller: web层

3.依赖注入, 给私有字段设置, 也可以给setter方法设置

普通值: @Value("")

引用值:

方式1: 按照【类型】注入

@Autowired

方式2: 按照【名称】注入1

@Autowired

@Qualifier("名称")

方式3: 按照【名称】注入2

@Resource("名称")

4.生命周期

初始化: @PostConstruct

销毁: @PreDestroy

5.作用域

@Scope("prototype") 多例

注解使用前提, 添加命名空间, 让spring扫描含有注解类

```
2<beans xmlns="http://www.springframework.org/schema/beans"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xmlns:context="http://www.springframework.org/schema/context"
5      xsi:schemaLocation="http://www.springframework.org/schema/beans
6                          http://www.springframework.org/schema/beans/spring-beans.xsd
7                          http://www.springframework.org/schema/context
8                          http://www.springframework.org/schema/context/spring-context.xsd">
9
10  <!-- 1.命名空间声明
11      默认: xmlns=""
12      显示: xmlns:别名=""
13      标注: <标签名> -> <bean>
14      标注: <别名:标签名> -> <context:....>
15
16  2.确定schema xsd 文件位置
17      xsi:schemaLocation="名称 位置 名称2 位置2 ...."
18      内容都是成对了【名称 位置】
19  -->
20  <bean class="com.itheima.springservice.UserService" />
21</beans>
```

```
1 <beans xmlns="http://www.springframework.org/schema/beans"
2       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3       xmlns:context="http://www.springframework.org/schema/context"
4       xsi:schemaLocation="http://www.springframework.org/schema/beans
5                           http://www.springframework.org/schema/beans/spring-beans.xsd
6                           http://www.springframework.org/schema/context
7                           http://www.springframework.org/schema/context/spring-context.xsd">
```

```

4      xsi:schemaLocation="http://www.springframework.org/schema
beans
5          http://www.springframework.org/schema/beans/spring-be
beans.xsd
6          http://www.springframework.org/schema/context
7          http://www.springframework.org/schema/context/spring-
context.xsd">
8 <!-- 组件扫描，扫描含有注解的类 -->
9 <context:component-scan base-package="com.itheima.g_annotation.a
_ioc"></context:component-scan>
10 </beans>

```

案例-使用注解实现IoC

实体类：

```

1 package entity;
2 /**
3  * 实体类
4  * @author 阿华
5  *
6  */
7 public class User {
8     private Integer id;
9     private String name;
10    private String pwd;
11
12    public User() {
13        super();
14    }
15    public User(Integer id, String name, String pwd) {
16        super();
17        this.id = id;
18        this.name = name;
19        this.pwd = pwd;
20    }
21
22    public Integer getId() {

```



```

23     return id;
24 }
25 public void setId(Integer id) {
26     this.id = id;
27 }
28 public String getName() {
29     return name;
30 }
31 public void setName(String name) {
32     this.name = name;
33 }
34 public String getPwd() {
35     return pwd;
36 }
37 public void setPwd(String pwd) {
38     this.pwd = pwd;
39 }
40 }
41

```

dao接口:

```

1 package dao;
2
3 import entity.User;
4
5 /**
6  * 用户数据操作接口
7  * @author 阿华
8  *
9  */
10 public interface UserDao {
11     /**
12     * 保存
13     */
14     public void save(User user);
15

```

```
15 }  
16
```

dao实现类:

```
1 package dao.impl;  
2  
3 import org.springframework.stereotype.Repository;  
4  
5 import dao.UserDao;  
6 import entity.User;  
7 /**  
8  * 实现类  
9  * @author 阿华  
10  *  
11  */  
12  
13 @Repository  
14 public class UserDaoImpl implements UserDao {  
15     //模拟去数据库中添加数据  
16     @Override  
17     public void save(User user) {  
18         System.out.println("添加到数据库中成功");  
19     }  
20 }  
21
```

service接口:

```
1 package service;  
2  
3 import entity.User;  
4  
5 public interface UserService {  
6     /**  
7     * 添加用户  
8     */  
9 }
```

```
9  public void addUser(User user);
10 }
11
```

service实现类:

```
1  package service.impl;
2
3  import org.springframework.beans.factory.annotation.Autowired;
4  import org.springframework.beans.factory.annotation.Qualifier;
5  import org.springframework.stereotype.Service;
6
7  import dao.UserDao;
8  import entity.User;
9  import service.UserService;
10 /**
11  * 注解
12  * @Component 实现bean组件的定义
13  * @Repository 用于实现dao类组件
14  * @Service 用于实现业务层组件
15  * @Controller 实现控制器（表示层）组件
16  *
17  * @author 阿华
18  *
19  */
20 @Service("service")
21 public class UserServiceImpl implements UserService{
22     //第三种方式：直接在属性上自动装配（注解dao层时不需指明实例化出来的dao的名字）
23     @Autowired
24     private UserDao dao;
25
26     //第一种方式：装配到setter方法（注解dao层时需指明实例化出来的dao的名字）
27     // @Autowired
28     // public void setDao(@Qualifier("dao") UserDao dao) {
```

```

29 // this.dao = dao;
30 // }
31
32 public UserServiceImpl() {
33     super();
34 }
35
36 //第二种方式：装配到有参构造方法（注解dao层时需指明实例化出来的dao的
    名字）
37 // @Autowired
38 // public UserServiceImpl(@Qualifier("dao") UserDao dao) {
39 //     super();
40 //     this.dao = dao;
41 // }
42
43
44 @Override
45 public void addUser(User user) {
46     dao.save(user);
47 }
48 }
49

```

spring配置文件：

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!-- Spring配置文件的根标签 -->
3 <!--
4     xmlns: 引入的文件路径
5     xsi:schemaLocation: 对应约束文件的路径
6     -->
7 <beans xmlns="http://www.springframework.org/schema/beans"
8     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
9     xmlns:aop="http://www.springframework.org/schema/aop"
10     xmlns:context="http://www.springframework.org/schema/context"
11     xsi:schemaLocation="

```

```

12 http://www.springframework.org/schema/beans
13 http://www.springframework.org/schema/beans/spring-beans-3.2.x
   sd
14 http://www.springframework.org/schema/context
15 http://www.springframework.org/schema/context/spring-context-
   3.2.xsd
16 http://www.springframework.org/schema/aop
17 http://www.springframework.org/schema/aop/spring-aop-3.2.xsd">
18
19 <!-- 扫描包中注解标注的类
20 base-package: 包名
21 -->
22 <context:component-scan base-package="service,dao"></context:c
   omponent-scan>
23 </beans>

```

测试:

```

1 package test;
2
3 import org.springframework.context.ApplicationContext;
4 import org.springframework.context.support.ClassPathXmlApplicati
   onContext;
5
6 import entity.User;
7 import service.UserService;
8
9 public class Test {
10     @org.junit.jupiter.api.Test
11     public void test() {
12         ApplicationContext ctx = new ClassPathXmlApplicationContext("a
   pplicationContext.xml");
13         UserService service = (UserService) ctx.getBean("service");
14         User user = new User();
15         user.setId(1);
16         user.setName("test");
17         user.setPwd("123456");

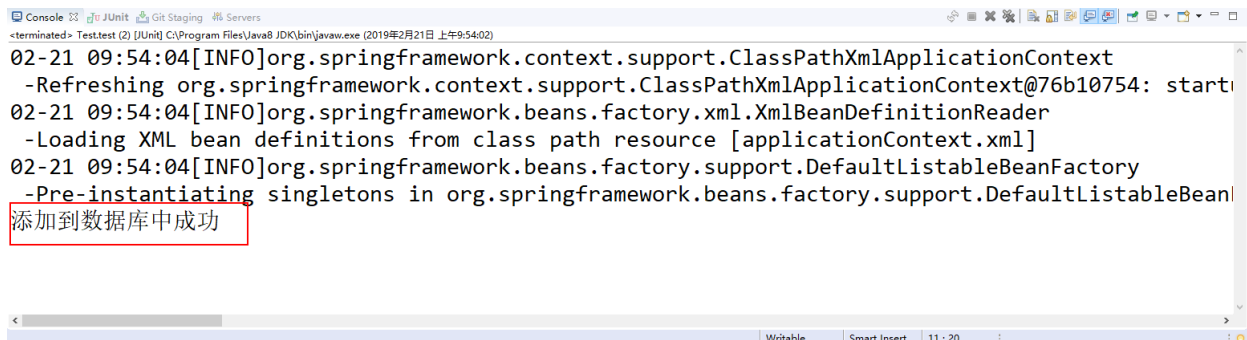
```

```

18  service.addUser(user);
19  }
20  }
21

```

运行结果：



```

02-21 09:54:04[INFO]org.springframework.context.support.ClassPathXmlApplicationContext
-Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@76b10754: start
02-21 09:54:04[INFO]org.springframework.beans.factory.xml.XmlBeanDefinitionReader
-Loading XML bean definitions from class path resource [applicationContext.xml]
02-21 09:54:04[INFO]org.springframework.beans.factory.support.DefaultListableBeanFactory
-Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBean
添加到数据库中成功

```

案例-使用注解实现切面

实体类：

```

1  package entity;
2  /**
3   * 实体类
4   * @author 阿华
5   *
6   */
7  public class User {
8      private Integer id;
9      private String name;
10     private String pwd;
11
12     public User() {
13         super();
14     }
15     public User(Integer id, String name, String pwd) {
16         super();
17         this.id = id;
18         this.name = name;
19         this.pwd = pwd;
20     }

```

```
21
22  public Integer getId() {
23      return id;
24  }
25  public void setId(Integer id) {
26      this.id = id;
27  }
28  public String getName() {
29      return name;
30  }
31  public void setName(String name) {
32      this.name = name;
33  }
34  public String getPwd() {
35      return pwd;
36  }
37  public void setPwd(String pwd) {
38      this.pwd = pwd;
39  }
40  }
41
```

dao接口:

```
1  package dao;
2
3  import entity.User;
4
5  /**
6   * 用户数据操作接口
7   * @author 阿华
8   *
9   */
10 public interface UserDao {
11     /**
12     * 保存
```

```
13  */
14  public void save(User user);
15  }
16
```

dao实现类:

```
1  package dao.impl;
2
3  import org.springframework.stereotype.Repository;
4  import org.springframework.stereotype.Service;
5
6  import dao.UserDao;
7  import entity.User;
8  /**
9   * 实现类
10   * @author 阿华
11   *
12   */
13  @Repository("dao")
14  public class UserDaoImpl implements UserDao {
15      //模拟去数据库中添加数据
16      @Override
17      public void save(User user) {
18          System.out.println("添加到数据库中成功");
19      }
20  }
21
```

service接口:

```
1  package service;
2
3  import entity.User;
4
5  public interface UserService {
6      /**
```



```
7  * 添加用户
8  */
9  public void addUser(User user);
10 }
11
```

service实现类:

```
1  package service.impl;
2
3  import org.springframework.beans.factory.annotation.Autowired;
4  import org.springframework.stereotype.Repository;
5  import org.springframework.stereotype.Service;
6
7  import dao.UserDao;
8  import entity.User;
9  import service.UserService;
10
11 @Service("service")
12 public class UserServiceImpl implements UserService{
13     //创建dao层对象，用于Spring进行注入
14     @Autowired
15     private UserDao dao;
16
17     public void setDao(UserDao dao) {
18         this.dao = dao;
19     }
20
21     @Override
22     public void addUser(User user) {
23         dao.save(user);
24     }
25 }
26
```

spring配置文件:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!-- Spring配置文件的根标签 -->
3 <!--
4   xmlns: 引入的文件路径
5   xsi:schemaLocation: 对应约束文件的路径
6   -->
7 <beans xmlns="http://www.springframework.org/schema/beans"
8   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
9   xmlns:aop="http://www.springframework.org/schema/aop"
10  xmlns:context="http://www.springframework.org/schema/context"
11  xsi:schemaLocation="http://www.springframework.org/schema/beans
12  http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
13  http://www.springframework.org/schema/context
14  http://www.springframework.org/schema/context/spring-context-3.2.xsd
15  http://www.springframework.org/schema/aop
16  http://www.springframework.org/schema/aop/spring-aop-3.2.xsd">
17
18 <!-- 扫描注解文件所在的包 -->
19 <context:component-scan base-package="service,dao"></context:component-scan>
20
21 <!-- 实例化切面 -->
22 <bean class="aop.UserLoggerAop"></bean>
23 <!-- 织入切面(自动为spring容器中配置@AspectJ切面bean实例化创建代理) -->
24 <aop:aspectj-autoproxy></aop:aspectj-autoproxy>
25 </beans>
```

aop切面类:

```
1 package aop;
2 /**
3  * 切面增强处理的方法所在的类
4  * @author 阿华
```

```
5  *
6  */
7
8  import org.apache.catalina.tribes.util.Arrays;
9  import org.apache.log4j.Logger;
10 import org.aspectj.lang.JoinPoint;
11 import org.aspectj.lang.annotation.After;
12 import org.aspectj.lang.annotation.AfterReturning;
13 import org.aspectj.lang.annotation.Aspect;
14 import org.aspectj.lang.annotation.Before;
15 import org.aspectj.lang.annotation.Pointcut;
16
17 @Aspect
18 public class UserLoggerAop {
19     private static final Logger
20     log=Logger.getLogger(UserLoggerAop.class);
21
22     //创建切入点方法
23     @Pointcut("execution(* service.UserService.*(..))")
24     public void pointcut() { }
25
26     @Before("pointcut()")
27     public void before(JoinPoint jp) {
28         log.info("调
29         用"+jp.getTarget()+"的"+jp.getSignature().getName()+
30         "方法，方法的参数是: "+Arrays.toString(jp.getArgs()));
31     }
32
33     @AfterReturning(pointcut="pointcut()", returning="result")
34     public void after(JoinPoint jp,Object result) {
35         log.info("调
36         用"+jp.getTarget()+"的"+jp.getSignature().getName()+
37         "方法，方法的返回结果: "+result);
38     }
39 }
```

测试:

```
1 package test;
2
3 import org.junit.jupiter.api.Test;
4 import org.springframework.context.ApplicationContext;
5 import org.springframework.context.support.ClassPathXmlApplication
  onContext;
6
7 import entity.User;
8 import service.UserService;
9
10 public class AopLoggerTest {
11     @Test
12     public void test() {
13         ApplicationContext ctx = new ClassPathXmlApplicationContext("a
  pplicationContext.xml");
14         UserService service = (UserService) ctx.getBean("service");
15         User user = new User();
16         user.setId(1);
17         user.setName("test");
18         user.setPwd("123456");
19         service.addUser(user);
20
21     }
22 }
23
```

运行结果:



```
<terminated> AopLoggerTest (2) [JUnit] C:\Program Files\Java\jdk-11.0.2\bin\javaw.exe (2019年2月21日 上午9:56:11)
-Loading XML bean definitions from class path resource [applicationContext.xml]
02-21 09:56:13[INFO]org.springframework.beans.factory.support.DefaultListableBeanFactory
-Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBeanFactory
02-21 09:56:13[INFO]aop.UserLoggerAop
-调用service.impl.UserServiceImpl@6c4906d3的addUser方法, 方法的参数是: {entity.User@7bd7d6d6}
添加到数据库中成功
02-21 09:56:13[INFO]aop.UserLoggerAop
-调用service.impl.UserServiceImpl@6c4906d3的addUser方法, 方法的返回结果: null
```

案例-使用注解实现异常抛出增强

实体类:

```
1 package entity;
2 /**
3  * 实体类
4  * @author 阿华
5  *
6  */
7 public class User {
8     private Integer id;
9     private String name;
10    private String pwd;
11
12    public User() {
13        super();
14    }
15    public User(Integer id, String name, String pwd) {
16        super();
17        this.id = id;
18        this.name = name;
19        this.pwd = pwd;
20    }
21
22    public Integer getId() {
23        return id;
24    }
25    public void setId(Integer id) {
26        this.id = id;
27    }
28    public String getName() {
29        return name;
30    }
31    public void setName(String name) {
32        this.name = name;
```

```

33     }
34     public String getPwd() {
35         return pwd;
36     }
37     public void setPwd(String pwd) {
38         this.pwd = pwd;
39     }
40 }
41

```

dao接口:

```

1  package dao;
2
3  import entity.User;
4
5  /**
6   * 用户数据操作接口
7   * @author 阿华
8   *
9   */
10 public interface UserDao {
11     /**
12     * 保存
13     */
14     public void save(User user);
15 }
16

```

dao实现类:

```

1  package dao.impl;
2
3  import org.springframework.stereotype.Repository;
4
5  import dao.UserDao;
6  import entity.User;

```

```

7  /**
8   * 实现类
9   * @author 阿华
10  *
11  */
12  @Repository("dao")
13  public class UserDaoImpl implements UserDao {
14      //模拟去数据库中添加数据
15      @Override
16      public void save(User user) {
17          System.out.println("UserDaoImpl类处理增强方法开始执行");
18          System.out.println("添加到数据库中成功");
19          throw new RuntimeException("未测试程序运行效果，模拟产生异常-UserDaoImpl类");
20      }
21  }
22

```

service接口:

```

1  package service;
2
3  import entity.User;
4
5  public interface UserService {
6      /**
7       * 添加用户
8       */
9      public void addUser(User user);
10  }
11

```

service实现类:

```

1  package service.impl;
2
3  import org.springframework.beans.factory.annotation.Autowired;

```

```

4 import org.springframework.stereotype.Repository;
5 import org.springframework.stereotype.Service;
6
7 import dao.UserDao;
8 import entity.User;
9 import service.UserService;
10 @Service("service")
11 public class UserServiceImpl implements UserService{
12     //创建dao层对象，用于Spring进行注入
13     @Autowired
14     private UserDao dao;
15
16     public void setDao(UserDao dao) {
17         this.dao = dao;
18     }
19
20     @Override
21     public void addUser(User user) {
22         dao.save(user);
23     }
24 }
25

```

spring配置文件:

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!-- Spring配置文件的根标签 -->
3 <!--
4     xmlns: 引入的文件路径
5     xsi:schemaLocation: 对应约束文件的路径
6     -->
7 <beans xmlns="http://www.springframework.org/schema/beans"
8     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
9     xmlns:aop="http://www.springframework.org/schema/aop"
10     xmlns:context="http://www.springframework.org/schema/context"

```



```

11  xsi:schemaLocation="http://www.springframework.org/schema/beans
12  http://www.springframework.org/schema/beans/spring-beans-3.2.x
13  http://www.springframework.org/schema/context
14  http://www.springframework.org/schema/context/spring-context-
15  3.2.xsd
16  http://www.springframework.org/schema/aop
17  http://www.springframework.org/schema/aop/spring-aop-3.2.xsd">
18  <context:component-scan base-package="service,dao"></context:component-scan>
19  <bean class="aop.ErrorLoggerAop"></bean>
20  <aop:aspectj-autoproxy></aop:aspectj-autoproxy>
21 </beans>

```

aop切面类:

```

1  package aop;
2  /**
3   * 切面增强处理的方法所在的类
4   * @author 阿华
5   */
6
7  import org.apache.log4j.Logger;
8  import org.aspectj.lang.JoinPoint;
9  import org.aspectj.lang.annotation.AfterThrowing;
10 import org.aspectj.lang.annotation.Aspect;
11
12 @Aspect
13 public class ErrorLoggerAop {
14     private static final Logger log=Logger.getLogger(ErrorLoggerAop.class);
15     /**
16      * 定义后置异常处理增强方法
17      */
18     @AfterThrowing(pointcut="execution(* service.UserService.*(..))",throwing="e")

```

```

19 public void afterThrowing(JoinPoint jp, RuntimeException e) {
20     System.out.println("ErrorLoggerAop类开始执行");
21     log.error("-----"+jp.getSignature().getName()+"方法发
    生异常: "+e+"-ErrorLoggerAop-----");
22     System.out.println("ErrorLoggerAop类执行完毕");
23 }
24 }
25

```

测试:

```

1 package test;
2
3 import org.junit.jupiter.api.Test;
4 import org.springframework.context.ApplicationContext;
5 import org.springframework.context.support.ClassPathXmlApplicati
    onContext;
6
7 import entity.User;
8 import service.UserService;
9
10 public class ErrorLoggerTest {
11     @Test
12     public void test() {
13         ApplicationContext ctx = new ClassPathXmlApplicationContext("a
    pplicationContext.xml");
14         UserService service = (UserService) ctx.getBean("service");
15         User user = new User();
16         user.setId(1);
17         user.setName("test");
18         user.setPwd("123456");
19         service.addUser(user);
20
21     }
22 }
23

```

运行结果：

```
Console | JUnit | Git Staging | Servers
<terminated> ErrorLoggerTest (2) [JUnit] C:\Program Files\Java8\jdk\bin\javaw.exe (2019年2月21日 上午9:59:12)
02-21 09:59:13[INFO]org.springframework.beans.factory.xml.XmlBeanDefinitionReader
-Loading XML bean definitions from class path resource [applicationContext.xml]
02-21 09:59:14[INFO]org.springframework.beans.factory.support.DefaultListableBeanFactory
-Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBean
 UserDaoImpl类处理增强方法开始执行
添加到数据库中成功
ErrorLoggerAop类开始执行
02-21 09:59:14[ERROR]aop.ErrorLoggerAop
-----addUser方法发生异常: java.lang.RuntimeException: 未测试程序运行效果，模拟产生异常-U
ErrorLoggerAop类执行完毕
```

```
Console | JUnit | Git Staging | Servers
Finished after 1.028 seconds
Runs: 1/1 | Errors: 1 | Failures: 0
ErrorLoggerTest [Runner: JUnit 5] (0.903 s)
test() (0.903 s)
Failure Trace
java.lang.RuntimeException: 未测试程序运行效果，模拟产生异常-UserDaoImpl类
at dao.impl.UserDaoImpl.save(UserDaoImpl.java:19)
at service.impl.UserServiceImpl.addUser(UserServiceImpl.java:22)
at org.springframework.aop.support.AopUtils.invokeJoinpointUsingReflection(AopUtils.java:317)
at org.springframework.aop.framework.ReflectiveMethodInvocation.invokeJoinpoint(ReflectiveMethodInvocation.java:183)
at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:150)
at org.springframework.aop.aspectj.AspectJAfterThrowingAdvice.invoke(AspectJAfterThrowingAdvice.java:55)
at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:172)
at org.springframework.aop.interceptor.ExposeInvocationInterceptor.invoke(ExposeInvocationInterceptor.java:91)
at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:172)
at org.springframework.aop.framework.JdkDynamicAopProxy.invoke(JdkDynamicAopProxy.java:204)
at com.sun.proxy.$Proxy14.addUser(Unknown Source)
at test.ErrorLoggerTest.test(ErrorLoggerTest.java:19)
at java.util.ArrayList.forEach(Unknown Source)
at java.util.ArrayList.forEach(Unknown Source)
```

案例-使用注解实现最终增强

实体类：

```
1 package entity;
2 /**
3  * 实体类
4  * @author 阿华
5  *
6  */
7 public class User {
8     private Integer id;
9     private String name;
10    private String pwd;
11
12    public User() {
13        super();
14    }
15    public User(Integer id, String name, String pwd) {
16        super();
```

```
17  this.id = id;
18  this.name = name;
19  this.pwd = pwd;
20  }
21
22  public Integer getId() {
23  return id;
24  }
25  public void setId(Integer id) {
26  this.id = id;
27  }
28  public String getName() {
29  return name;
30  }
31  public void setName(String name) {
32  this.name = name;
33  }
34  public String getPwd() {
35  return pwd;
36  }
37  public void setPwd(String pwd) {
38  this.pwd = pwd;
39  }
40  }
41
```

dao接口:

```
1  package dao;
2
3  import entity.User;
4
5  /**
6   * 用户数据操作接口
7   * @author 阿华
8   *
```

```

9  */
10 public interface UserDao {
11     /**
12      * 保存
13      */
14     public void save(User user);
15 }
16

```

dao实现类:

```

1  package dao.impl;
2
3  import org.springframework.beans.factory.annotation.Autowired;
4  import org.springframework.stereotype.Repository;
5
6  import dao.UserDao;
7  import entity.User;
8  /**
9   * 实现类
10   * @author 阿华
11   *
12   */
13  @Repository("dao")
14  public class UserDaoImpl implements UserDao {
15      //模拟去数据库中添加数据
16      @Override
17      public void save(User user) {
18          System.out.println("UserDaoImpl类处理增强方法开始执行");
19          System.out.println("添加到数据库中成功");
20          throw new RuntimeException("未测试程序运行效果，模拟产生异常");
21      }
22  }
23

```

service接口:

```
1 package service;
2
3 import entity.User;
4
5 public interface UserService {
6     /**
7      * 添加用户
8      */
9     public void addUser(User user);
10 }
11
```

service实现类:

```
1 package service.impl;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.stereotype.Service;
5
6 import dao.UserDao;
7 import entity.User;
8 import service.UserService;
9 @Service("service")
10 public class UserServiceImpl implements UserService{
11     //创建dao层对象，用于Spring进行注入
12     @Autowired
13     private UserDao dao;
14
15     public void setDao(UserDao dao) {
16         this.dao = dao;
17     }
18
19     @Override
20     public void addUser(User user) {
21         dao.save(user);
22     }
23 }
```

```
22 }
23 }
24
```

aop切面类:

```
1 package aop;
2 /**
3  * 切面增强处理的方法所在的类
4  * @author 阿华
5  */
6
7 import org.apache.log4j.Logger;
8 import org.aspectj.lang.JoinPoint;
9 import org.aspectj.lang.annotation.After;
10 import org.aspectj.lang.annotation.Aspect;
11
12 @Aspect
13 public class AfterLoggerAop {
14     private static final Logger log=Logger.getLogger(AfterLoggerAop.class);
15     /**
16      * 定义后置异常处理增强方法
17      */
18     @After("execution(* service.UserService.*(..))")
19     public void afterLogger(JoinPoint jp) {
20         log.error(jp.getSignature().getName()+"方法执行结束");
21     }
22 }
23
```

配置文件

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!-- Spring配置文件的根标签 -->
3 <!--
4     xmlns: 引入的文件路径
```

```

5  xsi:schemaLocation: 对应约束文件的路径
6  -->
7  <beans xmlns="http://www.springframework.org/schema/beans"
8  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
9  xmlns:aop="http://www.springframework.org/schema/aop"
10  xmlns:context="http://www.springframework.org/schema/context"
11  xsi:schemaLocation="http://www.springframework.org/schema/beans
12  http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
13  http://www.springframework.org/schema/context
14  http://www.springframework.org/schema/context/spring-context-
15  3.2.xsd
16  http://www.springframework.org/schema/aop
17  http://www.springframework.org/schema/aop/spring-aop-3.2.xsd">
18
19  <context:component-scan base-package="service,dao"></context:component-scan>
20  <bean class="aop.AfterLoggerAop"></bean>
21  <aop:aspectj-autoproxy></aop:aspectj-autoproxy>
22 </beans>

```

测试:

```

1  package test;
2
3  import org.junit.jupiter.api.Test;
4  import org.springframework.context.ApplicationContext;
5  import org.springframework.context.support.ClassPathXmlApplicationContext;
6
7  import entity.User;
8  import service.UserService;
9
10 public class AfterLoggerTest {
11     @Test
12     public void test() {

```



```

13  ApplicationContext ctx = new ClassPathXmlApplicationContext("a
    pplicationContext.xml");
14  UserService service =(UserService)ctx.getBean("service");
15  User user = new User();
16  user.setId(1);
17  user.setName("test");
18  user.setPwd("123456");
19  service.addUser(user);
20  }
21  }
22

```

运行结果：

```

Console 33  JUnit  Git Staging  Servers
<terminated> AfterLoggerTest.test (1) [JUnit] C:\Program Files\Java8\jdk\bin\javaw.exe (2019年2月21日 上午9:22:59)
-Loading XML bean definitions from class path resource [applicationContext.xml]
02-21 09:23:00[INFO]org.springframework.beans.factory.support.DefaultListableBeanFactory
-Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBean
UserDaoImpl类处理增强方法开始执行
添加到数据库中成功
02-21 09:23:01[ERROR]aop.AfterLoggerAop
-addUser方法执行结束

```

案例-使用注解实现环绕增强

实体类：

```

1  package entity;
2  /**
3   *  实体类
4   *  @author 阿华
5   *
6   */
7  public class User {
8      private Integer id;
9      private String name;
10     private String pwd;
11
12     public User() {
13         super();

```

```
14 }
15 public User(Integer id, String name, String pwd) {
16     super();
17     this.id = id;
18     this.name = name;
19     this.pwd = pwd;
20 }
21
22 public Integer getId() {
23     return id;
24 }
25 public void setId(Integer id) {
26     this.id = id;
27 }
28 public String getName() {
29     return name;
30 }
31 public void setName(String name) {
32     this.name = name;
33 }
34 public String getPwd() {
35     return pwd;
36 }
37 public void setPwd(String pwd) {
38     this.pwd = pwd;
39 }
40 }
41
```

dao接口:

```
1 package dao;
2
3 import entity.User;
4
5 /**
```

```
6  * 用户数据操作接口
7  * @author 阿华
8  *
9  */
10 public interface UserDao {
11     /**
12     * 保存
13     */
14     public void save(User user);
15 }
16
```

dao实现类:

```
1  package dao.impl;
2
3  import org.springframework.stereotype.Repository;
4
5  import dao.UserDao;
6  import entity.User;
7  /**
8   * 实现类
9   * @author 阿华
10  *
11  */
12  @Repository("dao")
13  public class UserDaoImpl implements UserDao {
14      //模拟去数据库中添加数据
15      @Override
16      public void save(User user) {
17          System.out.println("添加到数据库中成功");
18      }
19  }
20
```

service接口:

```
1 package service;
2
3 import entity.User;
4
5 public interface UserService {
6     /**
7      * 添加用户
8      */
9     public void addUser(User user);
10 }
11
```

service实现类:

```
1 package service.impl;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.stereotype.Service;
5
6 import dao.UserDao;
7 import entity.User;
8 import service.UserService;
9
10 @Service("service")
11 public class UserServiceImpl implements UserService{
12     //创建dao层对象，用于Spring进行注入
13     @Autowired
14     private UserDao dao;
15
16     public void setDao(UserDao dao) {
17         this.dao = dao;
18     }
19
20     @Override
21     public void addUser(User user) {
```

```

22 dao.save(user);
23 }
24 }
25

```

spring配置文件:

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!-- Spring配置文件的根标签 -->
3 <!--
4   xmlns: 引入的文件路径
5   xsi:schemaLocation: 对应约束文件的路径
6   -->
7 <beans xmlns="http://www.springframework.org/schema/beans"
8   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
9   xmlns:aop="http://www.springframework.org/schema/aop"
10  xmlns:context="http://www.springframework.org/schema/context"
11  xsi:schemaLocation="http://www.springframework.org/schema/beans
12  http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
13  http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.2.xsd
14  http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-3.2.xsd">
15   <context:component-scan base-package="service,dao"></context:component-scan>
16   <bean class="aop.AroundLoggerAop"></bean>
17   <aop:aspectj-autoproxy></aop:aspectj-autoproxy>
18 </beans>

```

aop切面类:

```

1 package aop;
2 /**
3  * 切面增强处理的方法所在的类
4  * @author 阿华
5  *
6  */

```

```
7
8 import org.apache.catalina.tribes.util.Arrays;
9 import org.apache.log4j.Logger;
10 import org.aspectj.lang.JoinPoint;
11 import org.aspectj.lang.ProceedingJoinPoint;
12 import org.aspectj.lang.annotation.Around;
13 import org.aspectj.lang.annotation.Aspect;
14 @Aspect
15 public class AroundLoggerAop {
16     private static final Logger log=Logger.getLogger(AroundLoggerAop.class);
17
18     @Around("execution(* service.UserService.*(..))")
19     public Object aroundLogger(ProceedingJoinPoint jp) throws Throwable {
20         /**
21          * 环绕前增强
22          */
23         log.info("调用"+jp.getTarget()+"的"+jp.getSignature().getName()+"方法，方法的参数是："+Arrays.toString(jp.getArgs()));
24
25         try {
26             //执行目标方法
27             Object result=jp.proceed();
28
29             /**
30              * 环绕后增强
31              */
32             log.info("调用"+jp.getTarget()+"的"+jp.getSignature().getName()+"方法，方法的返回值是："+result);
33             return result;
34         } catch (Exception e) {
35             e.printStackTrace();
36             log.error(jp.getSignature().getName()+"方法发生异常："+e);
37             throw e;
38         }
39     }
40 }
```

```
40 } finally {
41     System.out.println(jp.getSignature().getName()+"方法结束");
42 }
43
44 }
45 }
46
```

测试:

```
1 package test;
2
3 import org.junit.jupiter.api.Test;
4 import org.springframework.context.ApplicationContext;
5 import org.springframework.context.support.ClassPathXmlApplication
onContext;
6
7 import entity.User;
8 import service.UserService;
9
10 public class AroundLoggerTest {
11     @Test
12     public void test() {
13         ApplicationContext ctx = new ClassPathXmlApplicationContext("a
pplicationContext.xml");
14         UserService service = (UserService) ctx.getBean("service");
15         User user = new User();
16         user.setId(1);
17         user.setName("test");
18         user.setPwd("123456");
19         service.addUser(user);
20
21     }
22 }
23
```

运行结果：



```
<terminated> AroundLoggerTesttest (1) [JUnit] C:\Program Files\Java\jdk-8.0.60\bin\javaw.exe (2019年2月21日 上午10:01:23)
02-21 10:01:25[INFO]org.springframework.beans.factory.xml.XmlBeanDefinitionReader
-Loading XML bean definitions from class path resource [applicationContext.xml]
02-21 10:01:25[INFO]org.springframework.beans.factory.support.DefaultListableBeanFactory
-Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBeanFactory
02-21 10:01:26[INFO]aop.AroundLoggerAop
-调用service.impl.UserServiceImpl@8462f31的addUser方法，方法的参数是：{entity.User@73eb439a}
添加到数据库中成功
02-21 10:01:26[INFO]aop.AroundLoggerAop
-调用service.impl.UserServiceImpl@8462f31的addUser方法，方法的返回值是：null
addUser方法结束
```

第三章、Spring整合MyBatis

整体思路：

需要Spring通过单例方式管理SqlSessionFactory。

Spring和MyBatis整合生成代理对象，使用SqlSessionFactory创建SqlSession。（Spring和MyBatis整合自动完成）

持久层的mapper都需要由Spring进行管理。

整合环境：

jar包：

















mybatis3.2.7的jar包

spring3.2.0的jar包

dbcp连接池

数据库驱动

mybatis和spring的整合包：早期ibatis和spring整合是由spring官方提供，mybatis和spring整合由mybatis提供。

 aopalliance-1.0.jar
 aspectjweaver-1.6.9.jar
 commons-dbcp-1.4.jar
 commons-logging-1.2.jar
 commons-pool-1.6.jar
 log4j-1.2.17.jar
 mybatis-3.2.2.jar
 mybatis-spring-1.2.0.jar
 mysql-connector-java-5.1.0-bin.jar
 spring-aop-3.2.13.RELEASE.jar
 spring-beans-3.2.13.RELEASE.jar
 spring-context-3.2.13.RELEASE.jar
 spring-core-3.2.13.RELEASE.jar
 spring-expression-3.2.13.RELEASE.jar
 spring-jdbc-3.2.13.RELEASE.jar
 spring-tx-3.2.13.RELEASE.jar

Spring整合MyBatis-基础版

项目结构：

- ▼ resource
 - applicationContext.xml
 - log4j.properties
 - mybatis-config.xml
- ▼ test
 - ▼ test
 - StuTest.java
- ▼ src
 - ▼ dao
 - StuMapper.java
 - StuMapper.xml
 - ▼ dao.impl
 - StuMapperImpl.java
 - ▼ entity
 - Stu.java
 - ▼ service
 - StuService.java
 - ▼ service.impl
 - StuServiceImpl.java
- > JRE System Library [Java8 JDK]
- > Apache Tomcat v9.0 [Apache Tomcat v9.0]
- > Web App Libraries
- > JUnit 5
- build
- ▼ WebContent
 - > META-INF
 - ▼ WEB-INF
 - > lib
- log.log

Stu.java

```

1 package entity;
2 /**
3  * 与表进行ORM映射的实体类
4  * @author 阿华
5  *
6  */
7 public class Stu {
8     //属性
9     private Integer id;
10    private String sName;
11    private Integer sAge;
12

```

```

13 //对外方法
14 public Integer getId() {
15     return id;
16 }
17 public void setId(Integer id) {
18     this.id = id;
19 }
20 public String getName() {
21     return sName;
22 }
23 public void setName(String sName) {
24     this.sName = sName;
25 }
26 public Integer getAge() {
27     return sAge;
28 }
29 public void setAge(Integer sAge) {
30     this.sAge = sAge;
31 }
32 }

```

StuMapper.java

```

1 package dao;
2 /**
3  * stu表对应的接口操作方法
4  * @author 阿华
5  *
6  */
7 import java.util.List;
8 import entity.Stu;
9 public interface StuMapper {
10     /**
11     * 查询所有
12     */
13     public List<Stu> getStuList(Stu stu);

```

StuMapper.xml

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE mapper
3   PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4   "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5
6 <mapper namespace="dao.StuMapper">
7   <!-- 对应resultMap属性的配置信息，并起名id，用来被调用type配置信息的
      类型 -->
8   <resultMap type="Stu" id="stuList"></resultMap>
9   <!-- 查询sql语句
10    parameterType:传递的参数类型（必须指明完全限定类名，如果在配置文件
11    中指定的全局实体类位置可以省略包）
12    resultMap: 返回结果需要自己规定的时候，进行结果配置（stuList: 配置
      信息的id）
13    -->
14   <select id="getStuList" parameterType="Stu" resultMap="stuList">
15     select * from stu
16   </select>
17 </mapper>

```

StuMapperImpl.java

```

1 package dao.impl;
2
3 import java.util.List;
4 import org.apache.ibatis.session.SqlSession;
5 import org.mybatis.spring.SqlSessionTemplate;
6 import dao.StuMapper;
7 import entity.Stu;
8
9 public class StuMapperImpl implements StuMapper{
10   //创建用于mybatis进行crud操作的sqlSession对象

```

```

11  private SqlSessionTemplate sqlSession;//（需要注入）
12
13  public void setSqlSession(SqlSessionTemplate sqlSession) {
14      this.sqlSession = sqlSession;
15  }
16
17  @Override
18  public List<Stu> getStuList(Stu stu) {
19      /**
20       * sqlSession.selectList: 查询多个结果
21       * 第一个参数: 指明调用的mapper.xml文件中的哪一个语句
22       * 第二个参数: 需要传递的参数
23       */
24      return sqlSession.selectList("dao.StuMapper.getStuList",stu);
25  }
26  }

```

StuService.java

```

1  package service;
2  /**
3   * 业务层接口
4   * @author 阿华
5   *
6   */
7
8  import java.util.List;
9  import entity.Stu;
10
11  public interface StuService {
12      public List<Stu> findStuWithCounditions(Stu stu);
13  }
14

```

StuServiceImpl.java

```

1  package service.impl;

```

```

2
3 import java.util.List;
4
5 import dao.StuMapper;
6 import entity.Stu;
7 import service.StuService;
8
9 public class StuServiceImpl implements StuService{
10     //dao层对象
11     private StuMapper stuMapper;
12
13     public StuMapper getStuMapper() {
14         return stuMapper;
15     }
16
17     public void setStuMapper(StuMapper stuMapper) {
18         this.stuMapper = stuMapper;
19     }
20
21     @Override
22     public List<Stu> findStuWithCounditions(Stu stu) {
23         try {
24             return stuMapper.getStuList(stu);
25         } catch (RuntimeException e) {
26             e.printStackTrace();
27             throw e;
28         }
29     }
30 }

```

mybatis-config.xml

```

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

```

```

4
5 <!-- mybatis配置文件 -->
6 <configuration>
7   <!-- 类型别名 -->
8   <typeAliases>
9     <package name="entity"/>
10  </typeAliases>
11
12  <!-- 将下面的配置全部交由spring进行整合
13  数据源
14  sqlSession相关的实例化
15  mapper.xml文件的引入
16  -->
17 </configuration>

```

applicationContext.xml

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xmlns:p="http://www.springframework.org/schema/p"
5   xmlns:context="http://www.springframework.org/schema/context"
6   xmlns:mvc="http://www.springframework.org/schema/mvc"
7   xmlns:aop="http://www.springframework.org/schema/aop"
8   xmlns:tx="http://www.springframework.org/schema/tx"
9   xmlns:task="http://www.springframework.org/schema/task"
10  xsi:schemaLocation="
11    http://www.springframework.org/schema/beans
12    http://www.springframework.org/schema/beans/spring-beans-3.2.x
13    sd
14    http://www.springframework.org/schema/context
15    http://www.springframework.org/schema/context/spring-context-
16    3.2.xsd
17    http://www.springframework.org/schema/mvc
18    http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd
19    http://www.springframework.org/schema/aop

```

```
18 http://www.springframework.org/schema/aop/spring-aop-3.2.xsd
19 http://www.springframework.org/schema/tx
20 http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
21 http://www.springframework.org/schema/task
22 http://www.springframework.org/schema/task/spring-task-3.2.xsd">
23
24 <!-- 创建数据源，并进行实例化
25 destroy-method="close": 当我们数据库连接不使用的時候，就把该链接重新放在数据库中，
26 方便下次使用时候调用
27 Spring容器中，当容器关闭数据源能正常关闭；并不是真的把资源销毁，而是进行了回收
28 -->
29 <bean id="dataSource"
30 class="org.apache.commons.dbcp.BasicDataSource"
31 destroy-method="close">
32 <!-- 驱动类
33 name="driverClassName" : 注入的属性名
34 -->
35 <property name="driverClassName"
36 value="com.mysql.jdbc.Driver"></property>
37 <!-- URL地址 -->
38 <property name="url" value="jdbc:mysql://127.0.0.1/people?
39 useUnicode=true&characterEncoding=utf-8"></property>
40 <!-- 用户名 -->
41 <property name="username" value="root"></property>
42 <!-- 密码 -->
43 <property name="password" value="123456"></property>
44 </bean>
45
46 <!-- 配置SqlSessionFactoryBean对象 -->
47 <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSession
48 ionFactoryBean">
49 <!-- 引入数据源的组件 -->
50 <property name="dataSource" ref="dataSource"></property>
51 <!-- 导入mybatis配置文件中的配置信息 -->
```



```

50 <property name="configLocation" value="classpath:mybatis-config.xml"></property>
51 <!-- 配置（引入）sql映射文件 -->
52 <property name="mapperLocations">
53 <list>
54 <value>classpath:dao/*.xml</value>
55 </list>
56 </property>
57 </bean>
58
59 <!-- 配置SqlSessionFactory，将上面的sqlSessionFactory注入 -->
60 <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionTemplate">
61 <!-- 构造注入 -->
62 <constructor-arg name="sqlSessionFactory" ref="sqlSessionFactory"></constructor-arg>
63 </bean>
64
65 <!-- 实例化dao -->
66 <bean id="stuMapper" class="dao.impl.StuMapperImpl">
67 <property name="sqlSession" ref="sqlSessionTemplate"></property>
68 </bean>
69
70 <!-- 实例化业务 -->
71 <bean id="stuService" class="service.impl.StuServiceImpl">
72 <property name="stuMapper" ref="stuMapper"></property>
73 </bean>
74 </beans>

```

StuTest.java

```

1 package test;
2
3 import java.util.List;
4
5 import org.junit.jupiter.api.Test;

```

```

6 import org.springframework.context.ApplicationContext;
7 import org.springframework.context.support.ClassPathXmlApplication
  onContext;
8
9 import entity.Stu;
10 import service.StuService;
11 import service.impl.StuServiceImpl;
12
13 public class StuTest {
14
15     @Test
16     public void test() {
17         ApplicationContext ctx = new ClassPathXmlApplicationContext("a
  pplicationContext.xml");
18         StuServiceImpl stuService =(StuServiceImpl) ctx.getBean("stuSe
  rvice");
19         Stu stu = new Stu();
20         List<Stu> list = stuService.findStuWithCounditions(stu);
21         for (Stu st : list) {
22             System.out.println(st.getId()+" "+st.getName()+"
  "+st.getsAge());
23         }
24     }
25 }
26

```

程序运行结果:

```

[-----]
1   张三  18
2   李四  20
3   王五  22
4   哈哈  21
5   呵呵  30
6   吉吉国王  25
7   asd   18

```

Spring整合MyBatis-使用SqlSessionDaoSupport简化代码

StuTest.java

代码见上一实例，此处省略...

StuMapper.java

代码见上一实例，此处省略...

StuMapper.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE mapper
3   PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4   "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5
6 <mapper namespace="dao.StuMapper">
7   <!-- 对应resultMap属性的配置信息，并起名id，用来被调用type配置信息的
      类型 -->
8   <resultMap type="Stu" id="stuList"></resultMap>
9   <!-- 查询sql语句
10   parameterType:传递的参数类型（必须指明完全限定类名，如果在配置文件
11   中指定的全局实体类位置可以省略包）
12   resultMap: 返回结果需要自己规定的时候，进行结果配置（stuList: 配置
      信息的id）
13   -->
14   <select id="getStuList" parameterType="Stu" resultMap="stuList">
15     select * from stu
16   </select>
17 </mapper>
```

StuMapperImpl.java

```
1 package dao.impl;
2
3 import java.util.List;
4 import org.apache.ibatis.session.SqlSession;
```

```

5 import org.mybatis.spring.SqlSessionTemplate;
6 import org.mybatis.spring.support.SqlSessionDaoSupport;
7 import dao.StuMapper;
8 import entity.Stu;
9 /*
10  * SqlSessionDaoSupport: 对dao层操作的支持类（自动加载SqlSessionTemplate, 方便直接
11  * 得到sqlSession对象也用来进行crud）
12  */
13 public class StuMapperImpl extends SqlSessionDaoSupport implements StuMapper{
14
15     @Override
16     public List<Stu> getStuList(Stu stu) {
17         /**
18          * sqlSession.selectList: 查询多个结果
19          * 第一个参数: 指明调用的mapper.xml文件中的哪一个语句
20          * 第二个参数: 需要传递的参数
21          */
22         return this.getSqlSession().selectList("dao.StuMapper.getStuList",stu);
23     }
24 }
25

```

StuService.java

```

1 package service;
2 /**
3  * 业务层接口
4  * @author 阿华
5  *
6  */
7
8 import java.util.List;
9
10 import entity.Stu;

```

```
11
12 public interface StuService {
13     public List<Stu> findStuWithCounditions(Stu stu);
14 }
15
```

StuServiceImpl.java

```
1 package service.impl;
2
3 import java.util.List;
4 import dao.StuMapper;
5 import entity.Stu;
6 import service.StuService;
7
8 public class StuServiceImpl implements StuService{
9     //dao层对象
10     private StuMapper stuMapper;
11
12     public StuMapper getStuMapper() {
13         return stuMapper;
14     }
15
16     public void setStuMapper(StuMapper stuMapper) {
17         this.stuMapper = stuMapper;
18     }
19
20     @Override
21     public List<Stu> findStuWithCounditions(Stu stu) {
22         try {
23             return stuMapper.getStuList(stu);
24         } catch (RuntimeException e) {
25             e.printStackTrace();
26             throw e;
27         }
28     }
29 }
```

mybatis-config.xml

```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//
   EN"
3   "http://mybatis.org/dtd/mybatis-3-config.dtd">
4
5 <!-- mybatis配置文件 -->
6 <configuration>
7   <!-- 类型别名 -->
8   <typeAliases>
9     <package name="entity"/>
10  </typeAliases>
11
12  <!-- 将下面的配置全部交由spring进行整合
13  数据源
14  sqlSession相关的实例化
15  mapper.xml文件的引入
16  -->
17 </configuration>

```

applicationContext.xml

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xmlns:p="http://www.springframework.org/schema/p"
5   xmlns:context="http://www.springframework.org/schema/context"
6   xmlns:mvc="http://www.springframework.org/schema/mvc"
7   xmlns:aop="http://www.springframework.org/schema/aop"
8   xmlns:tx="http://www.springframework.org/schema/tx"
9   xmlns:task="http://www.springframework.org/schema/task"
10  xsi:schemaLocation="
11  http://www.springframework.org/schema/beans

```

```
12 http://www.springframework.org/schema/beans/spring-beans-3.2.x
sd
13 http://www.springframework.org/schema/context
14 http://www.springframework.org/schema/context/spring-context-
3.2.xsd
15 http://www.springframework.org/schema/mvc
16 http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd
17 http://www.springframework.org/schema/aop
18 http://www.springframework.org/schema/aop/spring-aop-3.2.xsd
19 http://www.springframework.org/schema/tx
20 http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
21 http://www.springframework.org/schema/task
22 http://www.springframework.org/schema/task/spring-task-3.2.xs
d">
23
24 <!-- 创建数据源，并进行实例化
25 destroy-method="close": 当我们数据库连接不使用的時候，就把该链接重
新放在数据库中，
26 方便下次使用时候调用
27 Spring容器中，当容器关闭数据源能正常关闭；并不是真的把资源销毁，而是
进行了回收
28 -->
29 <bean id="dataSource"
30 class="org.apache.commons.dbcp.BasicDataSource"
31 destroy-method="close">
32 <!-- 驱动类
33 name="driverClassName" : 注入的属性名
34 -->
35 <property name="driverClassName"
value="com.mysql.jdbc.Driver"></property>
36 <!-- URL地址 -->
37 <property name="url" value="jdbc:mysql://127.0.0.1/people?
useUnicode=true&characterEncoding=utf-8"></property>
38 <!-- 用户名 -->
39 <property name="username" value="root"></property>
40 <!-- 密码 -->
41 <property name="password" value="123456"></property>
```

```

43 </bean>
44
45 <!-- 配置SqlSessionFactoryBean对象 -->
46 <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionSessionFactoryBean">
47 <!-- 引入数据源的组件 -->
48 <property name="dataSource" ref="dataSource"></property>
49 <!-- 导入mybatis配置文件中的配置信息 -->
50 <property name="configLocation" value="classpath:mybatis-config.xml"></property>
51 <!-- 配置（引入）sql映射文件 -->
52 <property name="mapperLocations">
53 <list>
54 <value>classpath:dao/*.xml</value>
55 </list>
56 </property>
57 </bean>
58
59 <!-- 实例化dao -->
60 <bean id="stuMapper" class="dao.impl.StuMapperImpl">
61 <!-- name="sqlSessionFactory" : 继承的父类中体现的 -->
62 <property name="sqlSessionFactory" ref="sqlSessionFactory"></property>
63 </bean>
64
65 <!-- 实例化业务 -->
66 <bean id="stuService" class="service.impl.StuServiceImpl">
67 <property name="stuMapper" ref="stuMapper"></property>
68 </bean>
69 </beans>

```

StuTest.java

```

1 package test;
2
3 import java.util.List;
4 import org.junit.jupiter.api.Test;

```



```

5 import org.springframework.context.ApplicationContext;
6 import org.springframework.context.support.ClassPathXmlApplication
  onContext;
7 import entity.Stu;
8 import service.StuService;
9 import service.impl.StuServiceImpl;
10
11 public class StuTest {
12
13     @Test
14     public void test() {
15         ApplicationContext ctx = new ClassPathXmlApplicationContext("a
  pplicationContext.xml");
16         StuServiceImpl stuService =(StuServiceImpl) ctx.getBean("stuSe
  rvice");
17         Stu stu = new Stu();
18         List<Stu> list = stuService.findStuWithCounditions(stu);
19         for (Stu st : list) {
20             System.out.println(st.getId()+" "+st.getsName()+"
  "+st.getsAge());
21         }
22     }
23 }
24

```

程序运行结果:

```

1  张三  18
2  李四  20
3  王五  22
4  哈哈  21
5  呵呵  30
6  吉吉国王  25
7  asd   18

```

Spring整合MyBatis-MapperCannerConfig简化配置工作量

Stu.java

```
1 package entity;
2 /**
3  * 与表进行ORM映射的实体类
4  * @author 阿华
5  *
6  */
7 public class Stu {
8     //属性
9     private Integer id;
10    private String sName;
11    private Integer sAge;
12
13    //对外方法
14    public Integer getId() {
15        return id;
16    }
17    public void setId(Integer id) {
18        this.id = id;
19    }
20    public String getName() {
21        return sName;
22    }
23    public void setName(String sName) {
24        this.sName = sName;
25    }
26    public Integer getAge() {
27        return sAge;
28    }
29    public void setAge(Integer sAge) {
30        this.sAge = sAge;
31    }
32
33 }
```

```
34 }  
35
```

StuMapper.java

```
1 package dao;  
2 /**  
3  * stu表对应的接口操作方法  
4  * @author 阿华  
5  *  
6  */  
7  
8 import java.util.List;  
9  
10 import entity.Stu;  
11  
12 public interface StuMapper {  
13     /**  
14     * 查询所有  
15     */  
16     public List<Stu> getStuList(Stu stu);  
17  
18 }  
19
```

StuMapper.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>  
2 <!DOCTYPE mapper  
3     PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  
4     "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  
5  
6 <mapper namespace="dao.StuMapper">  
7     <!-- 对应resultMap属性的配置信息，并起名id，用来被调用type配置信息的  
8         类型 -->  
9     <resultMap type="Stu" id="stuList"></resultMap>  
10  
11     <!-- 查询sql语句
```

```
10 parameterType:传递的参数类型（必须指明完全限定类名，如果在配置文件中指明的全局实体类位置可以省略包）
11
12 resultMap: 返回结果需要自己规定的时候，进行结果配置（stuList: 配置信息的id）
13 -->
14 <select id="getStuList" parameterType="Stu" resultMap="stuList">
15     select * from stu
16 </select>
17 </mapper>
```

StuService.java

```
1 package service;
2 /**
3  * 业务层接口
4  * @author 阿华
5  *
6  */
7
8 import java.util.List;
9
10 import entity.Stu;
11
12 public interface StuService {
13     public List<Stu> findStuWithCounditions(Stu stu);
14 }
15
```

StuServiceImpl.java

```
1 package service.impl;
2
3 import java.util.List;
4
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.stereotype.Service;
```

```

7
8 import dao.StuMapper;
9 import entity.Stu;
10 import service.StuService;
11
12 @Service("stuService")
13 public class StuServiceImpl implements StuService{
14     //dao层对象
15     @Autowired
16     private StuMapper stuMapper;
17
18     @Override
19     public List<Stu> findStuWithCounditions(Stu stu) {
20         try {
21             return stuMapper.getStuList(stu);
22         } catch (RuntimeException e) {
23             e.printStackTrace();
24             throw e;
25         }
26     }
27
28 }
29

```

mybatis-config.xml

```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//
3 EN"
4 "http://mybatis.org/dtd/mybatis-3-config.dtd">
5
6 <!-- mybatis配置文件 -->
7 <configuration>
8     <!-- 类型别名 -->
9     <typeAliases>
10         <package name="entity"/>
11     </typeAliases>
12
13 
```

```

10 </typeAliases>
11
12 <!-- 将下面的配置全部交由spring进行整合
13 数据源
14 sqlSession相关的实例化
15 mapper.xml文件的引入
16 -->
17 </configuration>

```

applicationContext.xml

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xmlns:p="http://www.springframework.org/schema/p"
5   xmlns:context="http://www.springframework.org/schema/context"
6   xmlns:mvc="http://www.springframework.org/schema/mvc"
7   xmlns:aop="http://www.springframework.org/schema/aop"
8   xmlns:tx="http://www.springframework.org/schema/tx"
9   xmlns:task="http://www.springframework.org/schema/task"
10  xsi:schemaLocation="
11    http://www.springframework.org/schema/beans
12    http://www.springframework.org/schema/beans/spring-beans-3.2.x
13    sd
14    http://www.springframework.org/schema/context
15    http://www.springframework.org/schema/context/spring-context-
16    3.2.xsd
17    http://www.springframework.org/schema/mvc
18    http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd
19    http://www.springframework.org/schema/aop
20    http://www.springframework.org/schema/aop/spring-aop-3.2.xsd
21    http://www.springframework.org/schema/tx
22    http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
23    http://www.springframework.org/schema/task
24    http://www.springframework.org/schema/task/spring-task-3.2.xs
25    d">

```

```
24  <!-- 创建数据源，并进行实例化
25  destroy-method="close": 当我们数据库连接不使用的時候，就把该链接重新放在数据库中，
26  方便下次使用时候调用
27  Spring容器中，当容器关闭数据源能正常关闭；并不是真的把资源销毁，而是进行了回收
28  -->
29  <bean id="dataSource"
30  class="org.apache.commons.dbcp.BasicDataSource"
31  destroy-method="close">
32  <!-- 驱动类
33  name="driverClassName" : 注入的属性名
34  -->
35  <property name="driverClassName"
36  value="com.mysql.jdbc.Driver"></property>
37  <!-- URL地址 -->
38  <property name="url" value="jdbc:mysql://127.0.0.1/people?
39  useUnicode=true&characterEncoding=utf-8"></property>
40  <!-- 用户名 -->
41  <property name="username" value="root"></property>
42  <!-- 密码 -->
43  <property name="password" value="123456"></property>
44  </bean>
45  <!-- 配置SqlSessionFactoryBean对象 -->
46  <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSession
47  ionFactoryBean">
48  <!-- 引入数据源的组件 -->
49  <property name="dataSource" ref="dataSource"></property>
50  <!-- 导入mybatis配置文件中的配置信息 -->
51  <property name="configLocation" value="classpath:mybatis-config.xml"></property>
52  </bean>
53  <!--
54  MapperScannerConfigurer作用：
55  配置自动扫描指定包下的mapper接口，扫描基准包下所有的接口
```

```

56  根据sql相关的文件的动态注册为MapperFactoryBean，如此即可批量产生映射器实现类
57  （根据接口和sql文件，自动创建Mapper接口的实现类，可以省略Mapper实现类
58  和SqlSessionFactory中的sql映射文件路径）
59  -->
60  <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer"
61  <property name="basePackage" value="dao"></property>
62  </bean>
63
64  <!-- 扫描service注解 -->
65  <context:component-scan base-package="service"></context:component-scan>
66  </beans>

```

StuTest.java

```

1  package test;
2
3  import java.util.List;
4
5  import org.junit.jupiter.api.Test;
6  import org.springframework.context.ApplicationContext;
7  import org.springframework.context.support.ClassPathXmlApplicationContext;
8
9  import entity.Stu;
10 import service.StuService;
11 import service.impl.StuServiceImpl;
12
13 public class StuTest {
14
15     @Test
16     public void test() {
17         ApplicationContext ctx = new ClassPathXmlApplicationContext("applicationContext.xml");

```



```

18  StuServiceImpl stuService =(StuServiceImpl) ctx.getBean("stuSe
rvvice");
19  Stu stu = new Stu();
20  List<Stu> list = stuService.findStuWithCounditions(stu);
21  for (Stu st : list) {
22  System.out.println(st.getId()+" "+st.getName()+"
st.getAge());
23  }
24  }
25  }
26

```

程序运行结果:

```

1  张三  18
2  李四  20
3  王五  22
4  哈哈  21
5  呵呵  30
6  吉吉国王  25
7  asd   18

```

Spring整合MyBatis-使用MapperCannerConfig简化配置工作量

Stu.java

```

1  package entity;
2
3  import java.io.Serializable;
4
5  /**
6   * 与表进行ORM映射的实体类
7   * @author 阿华
8   *
9   */
10 public class Stu implements Serializable{
11     /**

```

```
12  *
13  */
14  private static final long serialVersionUID = -5565239645524914
    518L;
15
16  //属性
17  private Integer id;
18  private String sName;
19  private Integer sAge;
20
21  //对外方法
22  public Integer getId() {
23      return id;
24  }
25  public void setId(Integer id) {
26      this.id = id;
27  }
28  public String getName() {
29      return sName;
30  }
31  public void setName(String sName) {
32      this.sName = sName;
33  }
34  public Integer getAge() {
35      return sAge;
36  }
37  public void setAge(Integer sAge) {
38      this.sAge = sAge;
39  }
40
41
42  }
43
```

StuMapper.java

```
1  package dao;
```

```

2  /**
3   * stu表对应的接口操作方法
4   * @author 阿华
5   *
6   */
7
8  import java.util.List;
9  import entity.Stu;
10 public interface StuMapper {
11     /**
12      * 查询所有
13      */
14     public List<Stu> getStuList(Stu stu);
15
16     /**
17      * 添加
18      * @param stu
19      * @return
20      */
21     public int add(Stu stu);
22 }
23

```

StuMapper.xml

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE mapper
3      PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4      "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5
6  <mapper namespace="dao.StuMapper">
7      <!-- 对应resultMap属性的配置信息，并起名id，用来被调用type配置信息的
8           类型 -->
9      <resultMap type="Stu" id="stuList"></resultMap>
10     <!-- 查询sql语句
11     parameterType:传递的参数类型（必须指明完全限定类名，如果在配置文件

```

```

11  中指定的全局实体类位置可以省略包)
12  resultMap: 返回结果需要自己规定的时候, 进行结果配置 (stuList: 配置
    信息的id)
13  -->
14  <select id="getStuList" parameterType="Stu" resultMap="stuList">
15      select * from stu
16  </select>
17
18  <!-- 添加用户 -->
19  <insert id="add" parameterType="Stu">
20      insert into stu (sname, sage)
21      values (#{sName},#{sAge})
22  </insert>
23
24  </mapper>

```

StuService.java

```

1  package service;
2  /**
3   * 业务层接口
4   * @author 阿华
5   *
6   */
7
8  import java.util.List;
9  import entity.Stu;
10
11  public interface StuService {
12      public List<Stu> findStuWithCounditions(Stu stu);
13
14      public boolean addNewStu(Stu stu);
15  }
16

```

StuServiceImpl.java

```
1 package service.impl;
2
3 import java.util.List;
4 import org.springframework.beans.factory.annotation.Autowired;
5 import org.springframework.stereotype.Service;
6 import dao.StuMapper;
7 import entity.Stu;
8 import service.StuService;
9
10 @Service("stuService")
11 public class StuServiceImpl implements StuService{
12     //dao层对象
13     @Autowired
14     private StuMapper stuMapper;
15
16     @Override
17     public List<Stu> findStuWithCounditions(Stu stu) {
18         try {
19             return stuMapper.getStuList(stu);
20         } catch (RuntimeException e) {
21             e.printStackTrace();
22             throw e;
23         }
24     }
25
26     @Override
27     public boolean addNewStu(Stu stu) {
28         boolean result=false;
29         try {
30             if (stuMapper.add(stu) == 1) {
31                 result=true;
32             }
33         } catch (RuntimeException e) {
34             e.printStackTrace();
35             throw e;
```

```
36 }
37 return result;
38 }
39 }
```

mybatis-config.xml

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//
  EN"
3   "http://mybatis.org/dtd/mybatis-3-config.dtd">
4
5 <!-- mybatis配置文件 -->
6 <configuration>
7   <!-- 类型别名 -->
8   <typeAliases>
9     <package name="entity"/>
10   </typeAliases>
11 </configuration>
```

applicationContext.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xmlns:p="http://www.springframework.org/schema/p"
5   xmlns:context="http://www.springframework.org/schema/context"
6   xmlns:mvc="http://www.springframework.org/schema/mvc"
7   xmlns:aop="http://www.springframework.org/schema/aop"
8   xmlns:tx="http://www.springframework.org/schema/tx"
9   xmlns:task="http://www.springframework.org/schema/task"
10   xsi:schemaLocation="
11     http://www.springframework.org/schema/beans
12     http://www.springframework.org/schema/beans/spring-beans-3.2.x
13     sd
14     http://www.springframework.org/schema/context
```

```
14 http://www.springframework.org/schema/context/spring-context-
3.2.xsd
15 http://www.springframework.org/schema/mvc
16 http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd
17 http://www.springframework.org/schema/aop
18 http://www.springframework.org/schema/aop/spring-aop-3.2.xsd
19 http://www.springframework.org/schema/tx
20 http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
21 http://www.springframework.org/schema/task
22 http://www.springframework.org/schema/task/spring-task-3.2.xs
d">
23
24 <!-- 创建数据源，并进行实例化
25 destroy-method="close": 当我们数据库连接不使用的時候，就把该链接重
新放在数据库中，
26 方便下次使用时候调用
27 Spring容器中，当容器关闭数据源能正常关闭；并不是真的把资源销毁，而是
进行了回收
28 -->
29 <bean id="dataSource"
30 class="org.apache.commons.dbcp.BasicDataSource"
31 destroy-method="close">
32 <!-- 驱动类
33 name="driverClassName" : 注入的属性名
34 -->
35 <property name="driverClassName"
value="com.mysql.jdbc.Driver"></property>
36 <!-- URL地址 -->
37 <property name="url" value="jdbc:mysql://127.0.0.1/people?
useUnicode=true&characterEncoding=utf-8"></property>
38 <!-- 用户名 -->
39 <property name="username" value="root"></property>
40 <!-- 密码 -->
41 <property name="password" value="123456"></property>
42 </bean>
43
44
45 <!-- 配置SqlSessionFactoryBean对象 -->
```

```
46 <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSession
    sessionFactoryBean">
47 <!-- 引入数据源的组件 -->
48 <property name="dataSource" ref="dataSource"></property>
49 <!-- 导入mybatis配置文件中的配置信息 -->
50 <property name="configLocation" value="classpath:mybatis-config.
    xml"></property>
51 </bean>
52
53 <!--
54 MapperScannerConfigurer作用：
55 配置自动扫描指定包下的mapper接口,扫描基准包下所有的接口
56 根据sql相关的文件的动态注册为MapperFactoryBean，如此即可批量产生映射
    器实现类
57 （根据接口和sql文件，自动创建Mapper接口的实现类，可以省略Mapper实现
    类
58 和SqlSessionFactory中的sql映射文件路径）
59 -->
60 <bean class="org.mybatis.spring.mapper.MapperScannerConfigure
    r">
61 <property name="basePackage" value="dao"></property>
62 </bean>
63
64 <!-- 扫描service注解 -->
65 <context:component-scan base-package="service"></context:comp
    onent-scan>
66
67 <!-- =====配置事务处理配置开始===== -->
68 <!--
69 事务的特点(ACID):
70 1.原子性：事务是数据库的逻辑单元单位，事务中的很多操作要么全都做，要
    么全不做
71 2.一致性：事务执行结果必须是数据库中的从一个一致性状态变成另一个一致
    性状态
72 3.隔离性：一个数据的执行时不会被其他事务影响干扰
73 4.持续性：一旦提交一个事务，它对数据库影响改变是永久的
74 -->
75 <!-- 配置事务管理器bean，并向事务中注入数据源 -->
```



```
76 <!-- id="transactionManager": 建议此命名, 目的是方便配置事务增强时的默认调用 -->
77 <bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
78 <property name="dataSource" ref="dataSource"></property>
79 </bean>
80
81 <!-- 配置事务增强
82 transaction-manager="transactionManager": 调用默认的事务管理器, 当事务管理器
83 的名字为transactionManager的时候, 可以省略不写
84 -->
85 <tx:advice id="txAdvice">
86 <!-- 事务配置 -->
87 <tx:attributes>
88 <!--
89 propagation: 事务传播(处理)机制
90 REQUIRED(默认, 最常用): 支持当前事务, 如果执行时没有事务, 就新建一个事务
91 SUPPORTS: 支持当前事务, 如果当前没有事务, 就以非事务方式运行
92 MANDATORY: 支持当前事务, 如果没有事务就抛出异常
93 REQUIRES_NEW: 新建事务, 如果存在事务就把当前的事务挂起(暂停)
94 NOT_SUPPORTED: 以非事务方式执行, 如果已经存在事务, 就把当前的事务挂起(暂停)
95 NESTED: 以非事务方式执行, 如果已经存在事务就抛出异常
96 NEVER: 支持当前事务, 如果当前已经存在一个事务, 则执行一个嵌套事务, 如果没有就新建一个
97 -->
98 <!--
99 isolation事务的隔离机制(等级):
100 READ_COMMITTED: 提交读(当事务a更新数据的时候, 不允许其他事务进行任何操作包含读取;
101 当事务a读取, 其他事务可以读取、更新)
102 READ_UNCOMMITTED: 未提交读(当事务a更新数据的时候, 不允许其他事务进行更新数据,
103 但是可以读取)
104 REPEATABLE_READ: 重复读(当事务a更新数据的时候, 不允许其他事务进行任何操作,
```

```

105  当事务a进行读取时候，其他事务只能读取不能更新）
106  SERIALIZABLE：序列化（最严格的隔离级别，事务必须依次进行，并发性也是最差的）
107  -->
108
109  <!--
110  tx:method属性：
111  timeout="-1"：事务超时时间，允许事务运行的最长时间，以秒为单位，-1表示不超时
112  read-only="false"：事务为只读，默认false
113  rollback-for：设置能够触发回滚的异常类型（默认Spring只有RuntimeException是回滚）
114  no-rollback-for：设置不触发回滚的异常类型（默认Spring只有在checkedException不会触发事务回滚）
115  -->
116  <tx:method name="find*" propagation="SUPPORTS"/>
117  <tx:method name="add*" propagation="REQUIRED"/>
118  <tx:method name="del*" propagation="REQUIRED"/>
119  <tx:method name="update*" propagation="REQUIRED"/>
120  <tx:method name="*" propagation="REQUIRED"/>
121  </tx:attributes>
122  </tx:advice>
123
124  <!-- 定义切面 -->
125  <aop:config>
126  <aop:pointcut expression="execution(* service.*(..))" id="serviceMethod"/>
127  <!-- 对切点引用事务 -->
128  <aop:advisor advice-ref="txAdvice" pointcut-ref="serviceMethod"/>
129  </aop:config>
130  <!-- =====配置事务处理配置结束===== -->
131  </beans>

```

StuTest.java

```

1  package test;
2

```

```
3 import java.util.List;
4
5 import org.junit.jupiter.api.Test;
6 import org.springframework.context.ApplicationContext;
7 import org.springframework.context.support.ClassPathXmlApplication
onContext;
8
9 import entity.Stu;
10 import service.StuService;
11 import service.impl.StuServiceImpl;
12
13 public class StuTest {
14
15     @Test
16     public void test() {
17         ApplicationContext ctx = new ClassPathXmlApplicationContext("a
pplicationContext.xml");
18         StuService stuService =(StuService) ctx.getBean("stuService");
19         Stu stu = new Stu();
20         // List<Stu> list = stuService.findStuWithCounditions(stu);
21         // for (Stu st : list) {
22         // System.out.println(st.getId()+" "+st.getName()+" "+st.getAge());
23         // }
24         stu.setName("asd");
25         stu.setAge(18);
26         System.out.println(stuService.addNewStu(stu));
27     }
28 }
29
```

程序运行结果：

```
Console [3] JUnit Git Staging Servers
<terminated> StuTest.test (3) [JUnit] C:\Program Files\Java8\jdk\bin\javaw.exe (2019年2月22日 下午1:48:28)
[DEBUG] 2019-02-22 13:48:30,072 org.mybatis.spring.SqlSessionUtils - releasing transaction
[DEBUG] 2019-02-22 13:48:30,672 org.springframework.jdbc.datasource.DataSourceTransaction
[DEBUG] 2019-02-22 13:48:30,672 org.springframework.jdbc.datasource.DataSourceTransaction
[DEBUG] 2019-02-22 13:48:30,718 org.mybatis.spring.SqlSessionUtils - Transaction synchr
[DEBUG] 2019-02-22 13:48:30,719 org.mybatis.spring.SqlSessionUtils - Transaction synchr
[DEBUG] 2019-02-22 13:48:30,719 org.springframework.jdbc.datasource.DataSourceTransaction
[DEBUG] 2019-02-22 13:48:30,719 org.springframework.jdbc.datasource.DataSourceUtils - Ret
true
```

Spring整合MyBatis-为业务层添加声明式事务

Stu.java

```
1 package entity;
2
3 import java.io.Serializable;
4
5 /**
6  * 与表进行ORM映射的实体类
7  * @author 阿华
8  *
9  */
10 public class Stu implements Serializable{
11     /**
12     *
13     */
14     private static final long serialVersionUID = -5565239645524914
15     518L;
16     //属性
17     private Integer id;
18     private String sName;
19     private Integer sAge;
20
21     //对外方法
22     public Integer getId() {
23         return id;
24     }
25     public void setId(Integer id) {
26         this.id = id;
```

```

27     }
28     public String getName() {
29         return sName;
30     }
31     public void setName(String sName) {
32         this.sName = sName;
33     }
34     public Integer getAge() {
35         return sAge;
36     }
37     public void setAge(Integer sAge) {
38         this.sAge = sAge;
39     }
40 }

```

StuMapper.java

```

1  package dao;
2  /**
3   * stu表对应的接口操作方法
4   * @author 阿华
5   *
6   */
7
8  import java.util.List;
9
10 import entity.Stu;
11
12 public interface StuMapper {
13     /**
14     * 查询所有
15     */
16     public List<Stu> getStuList(Stu stu);
17
18     /**
19     * 添加

```

```

20  * @param stu
21  * @return
22  */
23  public int add(Stu stu);
24  }
25

```

StuMapper.xml

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE mapper
3    PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5
6  <mapper namespace="dao.StuMapper">
7    <!-- 对应resultMap属性的配置信息，并起名id，用来被调用type配置信息的
      类型 -->
8    <resultMap type="Stu" id="stuList"></resultMap>
9    <!-- 查询sql语句
10     parameterType:传递的参数类型（必须指明完全限定类名，如果在配置文件
11     中指定的全局实体类位置可以省略包）
12     resultMap: 返回结果需要自己规定的时候，进行结果配置（stuList: 配置
      信息的id）
13     -->
14    <select id="getStuList" parameterType="Stu" resultMap="stuList">
15      select * from stu
16    </select>
17
18    <!-- 添加用户 -->
19    <insert id="add" parameterType="Stu">
20      insert into stu (sname, sage)
21      values (#{sName},#{sAge})
22    </insert>
23
24  </mapper>

```

StuService.java

```
1 package service;
2 /**
3  * 业务层接口
4  * @author 阿华
5  *
6  */
7
8 import java.util.List;
9
10 import entity.Stu;
11
12 public interface StuService {
13     public List<Stu> findStuWithCounditions(Stu stu);
14
15     public boolean addNewStu(Stu stu);
16 }
17
```

StuServiceImpl.java

```
1 package service.impl;
2
3 import java.util.List;
4
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.stereotype.Service;
7
8 import dao.StuMapper;
9 import entity.Stu;
10 import service.StuService;
11
12 @Service("stuService")
13 public class StuServiceImpl implements StuService{
14     //dao层对象
15     @Autowired
```

```

16  private StuMapper stuMapper;
17
18  @Override
19  public List<Stu> findStuWithCounditions(Stu stu) {
20  try {
21  return stuMapper.getStuList(stu);
22  } catch (RuntimeException e) {
23  e.printStackTrace();
24  throw e;
25  }
26  }
27
28  @Override
29  public boolean addNewStu(Stu stu) {
30  boolean result=false;
31  try {
32  if (stuMapper.add(stu) == 1) {
33  result=true;
34  }
35  } catch (RuntimeException e) {
36  e.printStackTrace();
37  throw e;
38  }
39  return result;
40  }
41
42  }
43

```

mybatis-config.xml

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//
   EN"
3  "http://mybatis.org/dtd/mybatis-3-config.dtd">
4

```



```

5 <!-- mybatis配置文件 -->
6 <configuration>
7   <!-- 类型别名 -->
8   <typeAliases>
9     <package name="entity"/>
10  </typeAliases>
11
12  <!-- 将下面的配置全部交由spring进行整合
13  数据源
14  sqlSession相关的实例化
15  mapper.xml文件的引入
16  -->
17 </configuration>

```

applicationContext.xml

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xmlns:p="http://www.springframework.org/schema/p"
5   xmlns:context="http://www.springframework.org/schema/context"
6   xmlns:mvc="http://www.springframework.org/schema/mvc"
7   xmlns:aop="http://www.springframework.org/schema/aop"
8   xmlns:tx="http://www.springframework.org/schema/tx"
9   xmlns:task="http://www.springframework.org/schema/task"
10  xsi:schemaLocation="
11    http://www.springframework.org/schema/beans
12    http://www.springframework.org/schema/beans/spring-beans-3.2.x
13    sd
14    http://www.springframework.org/schema/context
15    http://www.springframework.org/schema/context/spring-context-
16    3.2.xsd
17    http://www.springframework.org/schema/mvc
18    http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd
19    http://www.springframework.org/schema/aop
20    http://www.springframework.org/schema/aop/spring-aop-3.2.xsd

```

```
19 http://www.springframework.org/schema/tx
20 http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
21 http://www.springframework.org/schema/task
22 http://www.springframework.org/schema/task/spring-task-3.2.xsd">
23
24 <!-- 创建数据源，并进行实例化
25 destroy-method="close": 当我们数据库连接不使用的時候，就把该链接重新放在数据库中，
26 方便下次使用时候调用
27 Spring容器中，当容器关闭数据源能正常关闭；并不是真的把资源销毁，而是进行了回收
28 -->
29 <bean id="dataSource"
30 class="org.apache.commons.dbcp.BasicDataSource"
31 destroy-method="close">
32 <!-- 驱动类
33 name="driverClassName" : 注入的属性名
34 -->
35 <property name="driverClassName"
36 value="com.mysql.jdbc.Driver"></property>
37 <!-- URL地址 -->
38 <property name="url" value="jdbc:mysql://127.0.0.1/people?
39 useUnicode=true&characterEncoding=utf-8"></property>
40 <!-- 用户名 -->
41 <property name="username" value="root"></property>
42 <!-- 密码 -->
43 <property name="password" value="123456"></property>
44 </bean>
45
46 <!-- 配置SqlSessionFactoryBean对象 -->
47 <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
48 <!-- 引入数据源的组件 -->
49 <property name="dataSource" ref="dataSource"></property>
50 <!-- 导入mybatis配置文件中的配置信息 -->
```

```

50 <property name="configLocation" value="classpath:mybatis-config.xml"></property>
51 </bean>
52
53 <!--
54 MapperScannerConfigurer作用：
55 配置自动扫描指定包下的mapper接口,扫描基准包下所有的接口
56 根据sql相关的文件的动态注册为MapperFactoryBean，如此即可批量产生映射器实现类
57 （根据接口和sql文件，自动创建Mapper接口的实现类，可以省略Mapper实现类
58 和SqlSessionFactory中的sql映射文件路径）
59 -->
60 <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
61 <property name="basePackage" value="dao"></property>
62 </bean>
63
64 <!-- 扫描service注解 -->
65 <context:component-scan base-package="service"></context:component-scan>
66
67 <!-- =====配置事务处理配置开始===== -->
68 <!--
69 事务的特点(ACID):
70 1.原子性：事务是数据库的逻辑单元单位，事务中的很多操作要么全都做，要么全不做
71 2.一致性：事务执行结果必须是数据库中的从一个一致性状态变成另一个一致性状态
72 3.隔离性：一个数据的执行时不会被其他事务影响干扰
73 4.持续性：一旦提交一个事务，它对数据库影响改变是永久的
74 -->
75 <!-- 配置事务管理器bean，并向事务中注入数据源 -->
76 <!-- id="transactionManager": 建议此命名，目的是方便配置事务增强时的默认调用 -->
77 <bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
78 <property name="dataSource" ref="dataSource"></property>

```

```
79 </bean>
80
81 <!-- 配置事务增强
82 transaction-manager="transactionManager": 调用默认的事务管理器，
   当事务管理器
83 的名字为transactionManager的时候，可以省略不写
84 -->
85 <tx:advice id="txAdvice">
86 <!-- 事务配置 -->
87 <tx:attributes>
88 <!--
89 propagation: 事务传播（处理）机制
90 REQUIRED（默认，最常用）：支持当前事务，如果执行时没有事务，就新建一个事务
91 SUPPORTS: 支持当前事务，如果当前没有事务，就以非事务方式运行
92 MANDATORY: 支持当前事务，如果没有事务就抛出异常
93 REQUIRES_NEW: 新建事务，如果存在事务就把当前的事务挂起（暂停）
94 NOT_SUPPORTED: 以非事务方式执行，如果已经存在事务，就把当前的事务挂起（暂停）
95 NESTED: 以非事务方式执行，如果已经存在事务就抛出异常
96 NEVER: 支持当前事务，如果当前已经存在一个事务，则执行一个嵌套事务，如果没有就新建一个
97 -->
98 <!--
99 isolation事务的隔离机制（等级）：
100 READ_COMMITTED: 提交读（当事务a更新数据的时候，不允许其他事务进行任何操作包含读取；
101 当事务a读取，其他事务可以读取、更新）
102 READ_UNCOMMITTED: 未提交读（当事务a更新数据的时候，不允许其他事务进行更新数据，
103 但是可以读取）
104 REPEATABLE_READ: 重复读（当事务a更新数据的时候，不允许其他事务进行任何操作，
105 当事务a进行读取时候，其他事务只能读取不能更新）
106 SERIALIZABLE: 序列化（最严格的隔离级别，事务必须依次进行，并发性也是最差的）
107 -->
108
```

```

109 <!--
110 tx:method属性:
111 timeout="-1": 事务超时时间, 允许事务运行的最长时间, 以秒为单位, -1
    表示不超时
112 read-only="false": 事务为只读, 默认false
113 rollback-for: 设置能够触发回滚的异常类型 (默认Spring只有RunTimeEx
    ception是回滚)
114 no-rollback-for: 设置不触发回滚的异常类型 (默认Spring只有在checke
    dException不会触发事务回滚)
115 -->
116 <tx:method name="find*" propagation="SUPPORTS"/>
117 <tx:method name="add*" propagation="REQUIRED"/>
118 <tx:method name="del*" propagation="REQUIRED"/>
119 <tx:method name="update*" propagation="REQUIRED"/>
120 <tx:method name="*" propagation="REQUIRED"/>
121 </tx:attributes>
122 </tx:advice>
123
124 <!-- 定义切面 -->
125 <aop:config>
126 <aop:pointcut expression="execution(* service.*(..))" id="se
    rviceMethod"/>
127 <!-- 对切点引用事务 -->
128 <aop:advisor advice-ref="txAdvice" pointcut-ref="serviceMetho
    d"/>
129 </aop:config>
130 <!-- =====配置事务处理配置结束===== -->
131 </beans>

```

StuTest.java

```

1 package test;
2
3 import java.util.List;
4
5 import org.junit.jupiter.api.Test;
6 import org.springframework.context.ApplicationContext;

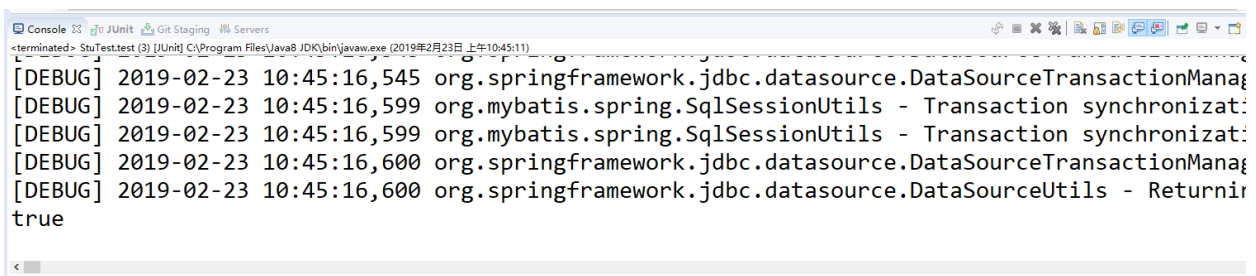
```

```

7 import org.springframework.context.support.ClassPathXmlApplication
onContext;
8
9 import entity.Stu;
10 import service.StuService;
11 import service.impl.StuServiceImpl;
12
13 public class StuTest {
14
15     @Test
16     public void test() {
17         ApplicationContext ctx = new ClassPathXmlApplicationContext("a
pplicationContext.xml");
18         StuService stuService =(StuService) ctx.getBean("stuService");
19         Stu stu = new Stu();
20         // List<Stu> list = stuService.findStuWithCounditions(stu);
21         // for (Stu st : list) {
22         // System.out.println(st.getId()+" "+st.getsName()+" "+st.getsA
ge());
23         // }
24         stu.setsName("asd");
25         stu.setsAge(18);
26         System.out.println(stuService.addNewStu(stu));
27     }
28 }
29

```

程序运行结果:



```

<terminated> StuTest.test(3) [JUnit] C:\Program Files\Java\jdk-8.0.60\bin\javaw.exe (2019年2月23日 上午10:45:11)
[DEBUG] 2019-02-23 10:45:16,545 org.springframework.jdbc.datasource.DataSourceTransactionManag
[DEBUG] 2019-02-23 10:45:16,599 org.mybatis.spring.SqlSessionUtils - Transaction synchronizati
[DEBUG] 2019-02-23 10:45:16,599 org.mybatis.spring.SqlSessionUtils - Transaction synchronizati
[DEBUG] 2019-02-23 10:45:16,600 org.springframework.jdbc.datasource.DataSourceTransactionManag
[DEBUG] 2019-02-23 10:45:16,600 org.springframework.jdbc.datasource.DataSourceUtils - Returnir
true

```

Spring整合MyBatis-使用注解为业务层添加声明式事务

Stu.java

```
1 package entity;
2
3 import java.io.Serializable;
4
5 /**
6  * 与表进行ORM映射的实体类
7  * @author 阿华
8  *
9  */
10 public class Stu implements Serializable{
11     /**
12      *
13      */
14     private static final long serialVersionUID = -5565239645524914518L;
15
16     //属性
17     private Integer id;
18     private String sName;
19     private Integer sAge;
20
21     //对外方法
22     public Integer getId() {
23         return id;
24     }
25     public void setId(Integer id) {
26         this.id = id;
27     }
28     public String getName() {
29         return sName;
30     }
31     public void setName(String sName) {
32         this.sName = sName;
33     }
```

```
34 public Integer getsAge() {
35     return sAge;
36 }
37 public void setsAge(Integer sAge) {
38     this.sAge = sAge;
39 }
40
41
42 }
43
```

StuMapper.java

```
1 package dao;
2 /**
3  * stu表对应的接口操作方法
4  * @author 阿华
5  *
6  */
7
8 import java.util.List;
9
10 import entity.Stu;
11
12 public interface StuMapper {
13     /**
14      * 查询所有
15      */
16     public List<Stu> getStuList(Stu stu);
17
18     /**
19      * 添加
20      * @param stu
21      * @return
22      */
23     public int add(Stu stu);
24 }
```



```
24 }  
25
```

StuMapper.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>  
2 <!DOCTYPE mapper  
3 PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  
4 "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  
5  
6 <mapper namespace="dao.StuMapper">  
7 <!-- 对应resultMap属性的配置信息，并起名id，用来被调用type配置信息的  
8 类型 -->  
9 <resultMap type="Stu" id="stuList"></resultMap>  
10 <!-- 查询sql语句  
11 parameterType:传递的参数类型（必须指明完全限定类名，如果在配置文件  
12 中指定的全局实体类位置可以省略包）  
13 resultMap: 返回结果需要自己规定的时候，进行结果配置（stuList: 配置  
14 信息的id）  
15 -->  
16 <select id="getStuList" parameterType="Stu" resultMap="stuList">  
17     select * from stu  
18 </select>  
19 <!-- 添加用户 -->  
20 <insert id="add" parameterType="Stu">  
21     insert into stu (sname, sage)  
22     values (#{sName},#{sAge})  
23 </insert>  
24 </mapper>
```

StuService.java

```
1 package service;  
2 /**  
3  * 业务层接口
```

```
4  * @author 阿华
5  *
6  */
7
8  import java.util.List;
9
10 import entity.Stu;
11
12 public interface StuService {
13     public List<Stu> findStuWithCounditions(Stu stu);
14
15     public boolean addNewStu(Stu stu);
16 }
17
```

StuServiceImpl.java

```
1  package service.impl;
2
3  import java.util.List;
4
5  import org.springframework.beans.factory.annotation.Autowired;
6  import org.springframework.stereotype.Service;
7  import org.springframework.transaction.annotation.Propagation;
8  import org.springframework.transaction.annotation.Transactional;
9
10 import dao.StuMapper;
11 import entity.Stu;
12 import service.StuService;
13 @Transactional
14 @Service("stuService")
15 public class StuServiceImpl implements StuService{
16     //dao层对象
17     @Autowired
18     private StuMapper stuMapper;
19
```

```

20  @Override
21  @Transactional(propagation=Propagation.SUPPORTS)
22  public List<Stu> findStuWithCounditions(Stu stu) {
23  try {
24  return stuMapper.getStuList(stu);
25  } catch (RuntimeException e) {
26  e.printStackTrace();
27  throw e;
28  }
29  }
30
31  @Override
32  @Transactional(propagation=Propagation.REQUIRED)
33  public boolean addNewStu(Stu stu) {
34  boolean result=false;
35  try {
36  if (stuMapper.add(stu) == 1) {
37  result=true;
38  }
39  } catch (RuntimeException e) {
40  e.printStackTrace();
41  throw e;
42  }
43  return result;
44  }
45
46  }
47

```

applicationContext.xml

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4  xmlns:p="http://www.springframework.org/schema/p"
5  xmlns:context="http://www.springframework.org/schema/context"

```

```
6   xmlns:mvc="http://www.springframework.org/schema/mvc"
7   xmlns:aop="http://www.springframework.org/schema/aop"
8   xmlns:tx="http://www.springframework.org/schema/tx"
9   xmlns:task="http://www.springframework.org/schema/task"
10  xsi:schemaLocation="
11    http://www.springframework.org/schema/beans
12    http://www.springframework.org/schema/beans/spring-beans-3.2.x
sd
13    http://www.springframework.org/schema/context
14    http://www.springframework.org/schema/context/spring-context-
3.2.xsd
15    http://www.springframework.org/schema/mvc
16    http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd
17    http://www.springframework.org/schema/aop
18    http://www.springframework.org/schema/aop/spring-aop-3.2.xsd
19    http://www.springframework.org/schema/tx
20    http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
21    http://www.springframework.org/schema/task
22    http://www.springframework.org/schema/task/spring-task-3.2.xs
d">
23
24  <!-- 创建数据源，并进行实例化
25   destroy-method="close": 当我们数据库连接不使用的時候，就把该链接重
新放在数据库中，
26   方便下次使用时候调用
27   Spring容器中，当容器关闭数据源能正常关闭；并不是真的把资源销毁，而是
进行了回收
28   -->
29  <bean id="dataSource"
30    class="org.apache.commons.dbcp.BasicDataSource"
31    destroy-method="close">
32    <!-- 驱动类
33    name="driverClassName" : 注入的属性名
34    -->
35    <property name="driverClassName"
value="com.mysql.jdbc.Driver"></property>
36    <!-- URL地址 -->
```

```

37 <property name="url" value="jdbc:mysql://127.0.0.1/people?
38 useUnicode=true&characterEncoding=utf-8"></property>
39 <!-- 用户名 -->
40 <property name="username" value="root"></property>
41 <!-- 密码 -->
42 <property name="password" value="123456"></property>
43 </bean>
44
45 <!-- 配置SqlSessionFactoryBean对象 -->
46 <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSess
47 ionFactoryBean">
48 <!-- 引入数据源的组件 -->
49 <property name="dataSource" ref="dataSource"></property>
50 <!-- 导入mybatis配置文件中的配置信息 -->
51 <property name="configLocation" value="classpath:mybatis-confi
52 g.xml"></property>
53 </bean>
54
55 <!--
56 MapperScannerConfigurer作用：
57 配置自动扫描指定包下的mapper接口,扫描基准包下所有的接口
58 根据sql相关的文件的动态注册为MapperFactoryBean，如此即可批量产生映
59 射器实现类
60 （根据接口和sql文件，自动创建Mapper接口的实现类，可以省略Mapper实现
61 类和SqlSessionFactory中的sql映射文件路径）
62 -->
63
64 <bean class="org.mybatis.spring.mapper.MapperScannerConfigure
65 r">
66 <property name="basePackage" value="dao"></property>
67 </bean>
68
69 <!-- 扫描service注解 -->
70 <context:component-scan base-package="service"></context:compo
71 nent-scan>
72
73 <!-- =====配置事务处理配置开始===== -->

```

```

68 <!-- 配置事务管理器bean，并向事务中注入数据源 -->
69 <bean id="transactionManager" class="org.springframework.jdbc.
datasource.DataSourceTransactionManager">
70 <property name="dataSource" ref="dataSource"></property>
71 </bean>
72 <!-- 事务的扫描 -->
73 <tx:annotation-driven></tx:annotation-driven>
74 <!-- =====配置事务处理配置结束===== -->
75 </beans>

```

mybatis-config.xml

```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//
EN"
3 "http://mybatis.org/dtd/mybatis-3-config.dtd">
4
5 <!-- mybatis配置文件 -->
6 <configuration>
7 <!-- 类型别名 -->
8 <typeAliases>
9 <package name="entity"/>
10 </typeAliases>
11
12 <!-- 将下面的配置全部交由spring进行整合
13 数据源
14 sqlSession相关的实例化
15 mapper.xml文件的引入
16 -->
17 </configuration>

```

StuTest.java

```

1 package test;
2
3 import java.util.List;
4

```

```

5 import org.junit.jupiter.api.Test;
6 import org.springframework.context.ApplicationContext;
7 import org.springframework.context.support.ClassPathXmlApplication
  onContext;
8
9 import entity.Stu;
10 import service.StuService;
11 import service.impl.StuServiceImpl;
12
13 public class StuTest {
14
15     @Test
16     public void test() {
17         ApplicationContext ctx = new ClassPathXmlApplicationContext("a
  pplicationContext.xml");
18         StuService stuService =(StuService) ctx.getBean("stuService");
19         Stu stu = new Stu();
20         // List<Stu> list = stuService.findStuWithCounditions(stu);
21         // for (Stu st : list) {
22         // System.out.println(st.getId()+" "+st.getName()+" "+st.getSA
  ge());
23         // }
24         stu.setName("asd");
25         stu.setAge(18);
26         System.out.println(stuService.addNewStu(stu));
27     }
28 }
29

```

程序运行结果：

同上一案例

Spring整合MyBatis-使用properties文件配置数据源

Stu.java

```

1 package entity;

```

```
2
3 import java.io.Serializable;
4
5 /**
6  * 与表进行ORM映射的实体类
7  * @author 阿华
8  *
9  */
10 public class Stu implements Serializable{
11     /**
12      *
13      */
14     private static final long serialVersionUID = -5565239645524914
15     518L;
16     //属性
17     private Integer id;
18     private String sName;
19     private Integer sAge;
20
21     //对外方法
22     public Integer getId() {
23         return id;
24     }
25     public void setId(Integer id) {
26         this.id = id;
27     }
28     public String getName() {
29         return sName;
30     }
31     public void setName(String sName) {
32         this.sName = sName;
33     }
34     public Integer getAge() {
35         return sAge;
36     }
37 }
```



```
37 public void setsAge(Integer sAge) {
38     this.sAge = sAge;
39 }
40
41
42 }
43
```

StuMapper.java

```
1 package dao;
2 /**
3  * stu表对应的接口操作方法
4  * @author 阿华
5  *
6  */
7
8 import java.util.List;
9
10 import entity.Stu;
11
12 public interface StuMapper {
13     /**
14      * 查询所有
15      */
16     public List<Stu> getStuList(Stu stu);
17
18     /**
19      * 添加
20      * @param stu
21      * @return
22      */
23     public int add(Stu stu);
24 }
25
```

StuMapper.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE mapper
3   PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4   "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5
6 <mapper namespace="dao.StuMapper">
7   <!-- 对应resultMap属性的配置信息，并起名id，用来被调用type配置信息的
   类型 -->
8   <resultMap type="Stu" id="stuList"></resultMap>
9   <!-- 查询sql语句
10   parameterType:传递的参数类型（必须指明完全限定类名，如果在配置文件
11   中指明全局实体类位置可以省略包）
12   resultMap: 返回结果需要自己规定的时候，进行结果配置（stuList: 配置
   信息的id）
13   -->
14   <select id="getStuList" parameterType="Stu" resultMap="stuList">
15     select * from stu
16   </select>
17
18   <!-- 添加用户 -->
19   <insert id="add" parameterType="Stu">
20     insert into stu (sname, sage)
21     values (#{sName},#{sAge})
22   </insert>
23
24 </mapper>
```

StuService.java

```
1 package service;
2 /**
3  * 业务层接口
4  * @author 阿华
5  *
6  */
```

```

7
8 import java.util.List;
9
10 import entity.Stu;
11
12 public interface StuService {
13     public List<Stu> findStuWithCounditions(Stu stu);
14
15     public boolean addNewStu(Stu stu);
16 }
17

```

StuServiceImpl.java

```

1 package service.impl;
2
3 import java.util.List;
4
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.stereotype.Service;
7 import org.springframework.transaction.annotation.Propagation;
8 import org.springframework.transaction.annotation.Transactional;
9
10 import dao.StuMapper;
11 import entity.Stu;
12 import service.StuService;
13 @Transactional
14 @Service("stuService")
15 public class StuServiceImpl implements StuService{
16     //dao层对象
17     @Autowired
18     private StuMapper stuMapper;
19
20     @Override
21     @Transactional(propagation=Propagation.SUPPORTS)
22     public List<Stu> findStuWithCounditions(Stu stu) {

```

```

23     try {
24         return stuMapper.getStuList(stu);
25     } catch (RuntimeException e) {
26         e.printStackTrace();
27         throw e;
28     }
29 }
30
31 @Override
32 @Transactional(propagation=Propagation.REQUIRED)
33 public boolean addNewStu(Stu stu) {
34     boolean result=false;
35     try {
36         if (stuMapper.add(stu) == 1) {
37             result=true;
38         }
39     } catch (RuntimeException e) {
40         e.printStackTrace();
41         throw e;
42     }
43     return result;
44 }
45
46 }
47

```

applicationContext.xml

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xmlns:p="http://www.springframework.org/schema/p"
5     xmlns:context="http://www.springframework.org/schema/context"
6     xmlns:mvc="http://www.springframework.org/schema/mvc"
7     xmlns:aop="http://www.springframework.org/schema/aop"
8     xmlns:tx="http://www.springframework.org/schema/tx"

```

```
9  xmlns:task="http://www.springframework.org/schema/task"
10  xsi:schemaLocation="
11  http://www.springframework.org/schema/beans
12  http://www.springframework.org/schema/beans/spring-beans-3.2.x
sd
13  http://www.springframework.org/schema/context
14  http://www.springframework.org/schema/context/spring-context-
3.2.xsd
15  http://www.springframework.org/schema/mvc
16  http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd
17  http://www.springframework.org/schema/aop
18  http://www.springframework.org/schema/aop/spring-aop-3.2.xsd
19  http://www.springframework.org/schema/tx
20  http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
21  http://www.springframework.org/schema/task
22  http://www.springframework.org/schema/task/spring-task-3.2.xs
d">
23  <!-- 引入properties文件 -->
24  <bean class="org.springframework.beans.factory.config.Property
PlaceholderConfigurer">
25  <property name="location">
26  <value>classpath:database.properties</value>
27  </property>
28  </bean>
29
30  <!-- 创建数据源，并进行实例化 -->
31  <bean id="dataSource"
32  class="org.apache.commons.dbcp.BasicDataSource"
33  destroy-method="close">
34  <!-- 驱动类
35  name="driverClassName" : 注入的属性名
36  -->
37  <property name="driverClassName" value="${jdbc.driver}"></prop
erty>
38  <!-- URL地址 -->
39  <property name="url" value="${jdbc.url}"></property>
40  <!-- 用户名 -->
```

```
41 <property name="username" value="${jdbc.username}"></property>
42 <!-- 密码 -->
43 <property name="password" value="${jdbc.password}"></property>
44 </bean>
45
46 <!-- 配置SqlSessionFactoryBean对象 -->
47 <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSession
  ionFactoryBean">
48 <!-- 引入数据源的组件 -->
49 <property name="dataSource" ref="dataSource"></property>
50 <!-- 导入mybatis配置文件中的配置信息 -->
51 <property name="configLocation" value="classpath:mybatis-confi
  g.xml"></property>
52 </bean>
53
54 <!--
55 MapperScannerConfigurer作用：
56 配置自动扫描指定包下的mapper接口,扫描基准包下所有的接口
57 根据sql相关的文件的动态注册为MapperFactoryBean，如此即可批量产生映
  射器实现类
58 （根据接口和sql文件，自动创建Mapper接口的实现类，可以省略Mapper实现
  类
59 和SqlSessionFactory中的sql映射文件路径）
60 -->
61 <bean class="org.mybatis.spring.mapper.MapperScannerConfigure
  r">
62 <property name="basePackage" value="dao"></property>
63 </bean>
64
65 <!-- 扫描service注解 -->
66 <context:component-scan base-package="service"></context:compo
  nent-scan>
67
68 <!-- =====配置事务处理配置开始===== -->
69 <!-- 配置事务管理器bean，并向事务中注入数据源 -->
70 <bean id="transactionManager" class="org.springframework.jdbc.
  datasource.DataSourceTransactionManager">
71 <property name="dataSource" ref="dataSource"></property>
```

```

72 </bean>
73 <!-- 事务的扫描 -->
74 <tx:annotation-driven></tx:annotation-driven>
75 <!-- =====配置事务处理配置结束===== -->
76 </beans>

```

mybatis-config.xml

```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//
  EN"
3   "http://mybatis.org/dtd/mybatis-3-config.dtd">
4
5 <!-- mybatis配置文件 -->
6 <configuration>
7   <!-- 类型别名 -->
8   <typeAliases>
9     <package name="entity"/>
10  </typeAliases>
11
12 <!-- 将下面的配置全部交由spring进行整合
13 数据源
14 sqlSession相关的实例化
15 mapper.xml文件的引入
16 -->
17 </configuration>

```

database.properties

```

1 jdbc.driver=com.mysql.jdbc.Driver
2 jdbc.url=jdbc:mysql://127.0.0.1/people?useUnicode=true&character
  Encoding=utf-8
3 jdbc.username=root
4 jdbc.password=123456

```

StuTest.java

```

1 package test;

```

```

2
3 import java.util.List;
4
5 import org.junit.jupiter.api.Test;
6 import org.springframework.context.ApplicationContext;
7 import org.springframework.context.support.ClassPathXmlApplication
  onContext;
8
9 import entity.Stu;
10 import service.StuService;
11 import service.impl.StuServiceImpl;
12
13 public class StuTest {
14
15     @Test
16     public void test() {
17         ApplicationContext ctx = new ClassPathXmlApplicationContext("a
  pplicationContext.xml");
18         StuService stuService =(StuService) ctx.getBean("stuService");
19         Stu stu = new Stu();
20         // List<Stu> list = stuService.findStuWithCounditions(stu);
21         // for (Stu st : list) {
22         // System.out.println(st.getId()+" "+st.getName()+" "+st.getSA
  ge());
23         // }
24         stu.setName("asd");
25         stu.setAge(18);
26         System.out.println(stuService.addNewStu(stu));
27     }
28 }
29

```

程序运行结果：

见上一实例

Spring整合MyBatis-使用JNDI

Stu.java

```
1 package entity;
2
3 import java.io.Serializable;
4
5 /**
6  * 与表进行ORM映射的实体类
7  * @author 阿华
8  *
9  */
10 public class Stu implements Serializable{
11     /**
12     *
13     */
14     private static final long serialVersionUID = -5565239645524914518L;
15
16     //属性
17     private Integer id;
18     private String sName;
19     private Integer sAge;
20
21     //对外方法
22     public Integer getId() {
23         return id;
24     }
25     public void setId(Integer id) {
26         this.id = id;
27     }
28     public String getName() {
29         return sName;
30     }
31     public void setName(String sName) {
32         this.sName = sName;
```

```
33     }
34     public Integer getsAge() {
35         return sAge;
36     }
37     public void setsAge(Integer sAge) {
38         this.sAge = sAge;
39     }
40
41
42 }
43
```

StuMapper.java

```
1  package dao;
2  /**
3   * stu表对应的接口操作方法
4   * @author 阿华
5   *
6   */
7
8  import java.util.List;
9
10 import entity.Stu;
11
12 public interface StuMapper {
13     /**
14      * 查询所有
15      */
16     public List<Stu> getStuList(Stu stu);
17
18     /**
19      * 添加
20      * @param stu
21      * @return
22      */
23 }
```

```
23 public int add(Stu stu);
24 }
25
```

StuMapper.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE mapper
3 PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4 "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5
6 <mapper namespace="dao.StuMapper">
7   <!-- 对应resultMap属性的配置信息，并起名id，用来被调用type配置信息的
      类型 -->
8   <resultMap type="Stu" id="stuList"></resultMap>
9   <!-- 查询sql语句
10   parameterType:传递的参数类型（必须指明完全限定类名，如果在配置文件
11   中指明全局实体类位置可以省略包）
12   resultMap: 返回结果需要自己规定的时候，进行结果配置（stuList: 配置
      信息的id）
13   -->
14   <select id="getStuList" parameterType="Stu" resultMap="stuList">
15     select * from stu
16   </select>
17
18   <!-- 添加用户 -->
19   <insert id="add" parameterType="Stu">
20     insert into stu (sname, sage)
21     values (#{sName},#{sAge})
22   </insert>
23
24 </mapper>
```

StuService.java

```
1 package service;
2 /**
```

```

3  * 业务层接口
4  * @author 阿华
5  *
6  */
7
8  import java.util.List;
9
10 import entity.Stu;
11
12 public interface StuService {
13     public List<Stu> findStuWithCounditions(Stu stu);
14
15     public boolean addNewStu(Stu stu);
16 }
17

```

StuServiceImpl.java

```

1  package service.impl;
2
3  import java.util.List;
4
5  import org.springframework.beans.factory.annotation.Autowired;
6  import org.springframework.stereotype.Service;
7  import org.springframework.transaction.annotation.Propagation;
8  import org.springframework.transaction.annotation.Transactional;
9
10 import dao.StuMapper;
11 import entity.Stu;
12 import service.StuService;
13 @Transactional
14 @Service("stuService")
15 public class StuServiceImpl implements StuService{
16     //dao层对象
17     @Autowired
18     private StuMapper stuMapper;
19
20 }
21

```

```

19
20 @Override
21 @Transactional(propagation=Propagation.SUPPORTS)
22 public List<Stu> findStuWithCounditions(Stu stu) {
23     try {
24         return stuMapper.getStuList(stu);
25     } catch (RuntimeException e) {
26         e.printStackTrace();
27         throw e;
28     }
29 }
30
31 @Override
32 @Transactional(propagation=Propagation.REQUIRED)
33 public boolean addNewStu(Stu stu) {
34     boolean result=false;
35     try {
36         if (stuMapper.add(stu) == 1) {
37             result=true;
38         }
39     } catch (RuntimeException e) {
40         e.printStackTrace();
41         throw e;
42     }
43     return result;
44 }
45
46 }
47

```

applicationContext.xml

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xmlns:p="http://www.springframework.org/schema/p"

```

```
5  xmlns:context="http://www.springframework.org/schema/context"
6  xmlns:mvc="http://www.springframework.org/schema/mvc"
7  xmlns:aop="http://www.springframework.org/schema/aop"
8  xmlns:tx="http://www.springframework.org/schema/tx"
9  xmlns:task="http://www.springframework.org/schema/task"
10  xsi:schemaLocation="
11  http://www.springframework.org/schema/beans
12  http://www.springframework.org/schema/beans/spring-beans-3.2.x
sd
13  http://www.springframework.org/schema/context
14  http://www.springframework.org/schema/context/spring-context-
3.2.xsd
15  http://www.springframework.org/schema/mvc
16  http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd
17  http://www.springframework.org/schema/aop
18  http://www.springframework.org/schema/aop/spring-aop-3.2.xsd
19  http://www.springframework.org/schema/tx
20  http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
21  http://www.springframework.org/schema/task
22  http://www.springframework.org/schema/task/spring-task-3.2.xs
d">
23
24  <!-- 创建JNDI数据源 -->
25  <bean id="dataSource" class="org.springframework.jndi.JndiObje
ctFactoryBean">
26  <property name="jndiName">
27  <value>java:comp/env/jdbc/people</value>
28  </property>
29  </bean>
30
31
32  <!-- 配置SqlSessionFactoryBean对象 -->
33  <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSess
ionFactoryBean">
34  <!-- 引入数据源的组件 -->
35  <property name="dataSource" ref="dataSource"></property>
36  <!-- 导入mybatis配置文件中的配置信息 -->
```

```

37 <property name="configLocation" value="classpath:mybatis-config.xml"></property>
38 </bean>
39
40 <!--
41 MapperScannerConfigurer作用：
42 配置自动扫描指定包下的mapper接口,扫描基准包下所有的接口
43 根据sql相关的文件的动态注册为MapperFactoryBean，如此即可批量产生映射器实现类
44 （根据接口和sql文件，自动创建Mapper接口的实现类，可以省略Mapper实现类
45 和SqlSessionFactory中的sql映射文件路径）
46 -->
47 <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
48 <property name="basePackage" value="dao"></property>
49 </bean>
50
51 <!-- 扫描service注解 -->
52 <context:component-scan base-package="service"></context:component-scan>
53
54 <!-- =====配置事务处理配置开始===== -->
55 <!-- 配置事务管理器bean，并向事务中注入数据源 -->
56 <bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
57 <property name="dataSource" ref="dataSource"></property>
58 </bean>
59 <!-- 事务的扫描 -->
60 <tx:annotation-driven></tx:annotation-driven>
61 <!-- =====配置事务处理配置结束===== -->
62 </beans>

```

mybatis-config.xml

```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"

```

```

3  "http://mybatis.org/dtd/mybatis-3-config.dtd">
4
5  <!-- mybatis配置文件 -->
6  <configuration>
7    <!-- 类型别名 -->
8    <typeAliases>
9      <package name="entity"/>
10     </typeAliases>
11
12    <!-- 将下面的配置全部交由spring进行整合
13    数据源
14    sqlSession相关的实例化
15    mapper.xml文件的引入
16    -->
17  </configuration>

```

StuServlet.java

```

1  package test;
2
3  import java.io.IOException;
4  import javax.servlet.ServletException;
5  import javax.servlet.annotation.WebServlet;
6  import javax.servlet.http.HttpServlet;
7  import javax.servlet.http.HttpServletRequest;
8  import javax.servlet.http.HttpServletResponse;
9
10 import org.springframework.context.ApplicationContext;
11 import org.springframework.context.support.ClassPathXmlApplicationContext;
12
13 import entity.Stu;
14 import service.StuService;
15
16 @WebServlet("/StuServlet")
17 public class StuServlet extends HttpServlet {

```



```

18  private static final long serialVersionUID = 1L;
19
20  public StuServlet() {
21
22  }
23
24  protected void doGet(HttpServletRequest request, HttpServletResponse
    sponse response) throws ServletException, IOException {
25      ApplicationContext ctx = new ClassPathXmlApplicationContext("a
    pplicationContext.xml");
26      StuService stuService =(StuService) ctx.getBean("stuService");
27      Stu stu = new Stu();
28      stu.setName("asd");
29      stu.setAge(18);
30      System.out.println(stuService.addNewStu(stu));
31  }
32
33  protected void doPost(HttpServletRequest request, HttpServletResponse
    sponse response) throws ServletException, IOException {
34      request.setCharacterEncoding("UTF-8");
35      doGet(request, response);
36  }
37  }
38

```

WebContent/META-INF/context.xml

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <Context>
3      <Resource
4          name="jdbc/people"
5          auth="Container"
6          type="javax.sql.DataSource"
7          maxActive="100"
8          maxIdle="30"
9          maxWait="10000"
10         username="root"

```

```
11 password="123456"
12 driverClassName="com.mysql.jdbc.Driver"
13 url="jdbc:mysql://127.0.0.1:3306/people?useUnicode=true&characterEncoding=utf-8"
14 />
15 </Context>
```

StuTest.java

```
1 package test;
2
3 import java.util.List;
4
5 import org.junit.jupiter.api.Test;
6 import org.springframework.context.ApplicationContext;
7 import org.springframework.context.support.ClassPathXmlApplicationContext;
8
9 import entity.Stu;
10 import service.StuService;
11 import service.impl.StuServiceImpl;
12
13 public class StuTest {
14
15     @Test
16     public void test() {
17         ApplicationContext ctx = new ClassPathXmlApplicationContext("applicationContext.xml");
18         StuService stuService = (StuService) ctx.getBean("stuService");
19         Stu stu = new Stu();
20         // List<Stu> list = stuService.findStuWithCounditions(stu);
21         // for (Stu st : list) {
22         //     System.out.println(st.getId()+" "+st.getName()+" "+st.getAge());
23         // }
24         stu.setName("asd");
25         stu.setAge(18);
```

```
26 System.out.println(stuService.addNewStu(stu));  
27 }  
28 }  
29
```

程序运行结果：

见上一实例

Copyright © 曹帅华 All Rights Reserved