Spring笔记

一、xml配置bean方式使用（一般不这么用）

1.bean.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans.xsd"* >    <bean id=*"bean01"* class=*"com.zhangfd.ioc.TestIoc01"* scope=*"singleton"* ></bean>    </beans> |

2.通过加载配置bean.xml通过spring生成bean

①加载文件时就把bean生成：

|  |
| --- |
| @Test  **public** **void** testDi() {  ApplicationContext ac = **new** ClassPathXmlApplicationContext("classpath\*:spring-ioc2.xml");  UserInfo userInfo = (UserInfo) ac.getBean("userInfo");  System.***out***.println(userInfo);  } |

②延迟加载方式生成bean（有用到才生成，一般c/s模式才用）

|  |
| --- |
| @Test  **public** **void** testBeanFactory() {  Resource resource = **new** ClassPathResource("/spring-ioc2.xml");  BeanFactory bf = **new** ~~XmlBeanFactory~~(resource);  Class01 c01 = (Class01) bf.getBean("class01");  c01.show();  } |

3.DI

没啥好说的，就是配置bean.xml的时候配置让spring生成bean的时候给里面的属性赋值。

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans.xsd"* >    <bean id=*"userInfo"* class=*"com.zhangfd.di.UserInfo"*>  <property name=*"id"* value=*"1"*></property>  <property name=*"name"* value=*"zfd"*></property>  </bean>    <bean id=*"uc"* class=*"com.zhangfd.DIMVC.controller.UserInfoController"*>  <property name=*"userInfoService"* ref=*"us"*></property>  </bean>    <bean id=*"us"* class=*"com.zhangfd.DIMVC.service.impl.UserInfoService"*>  <property name=*"userInfoDao"* ref=*"ud"*></property>  </bean>    <bean id=*"ud"* class=*"com.zhangfd.DIMVC.dao.impl.UserInfoDao"*></bean>    </beans> |

二、注解方式使用spring

1.开启注解支持

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns:p=*"http://www.springframework.org/schema/p"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context.xsd"*>  <context:annotation-config></context:annotation-config>  <context:component-scan base-package=*"annotation01"*></context:component-scan>    </beans> |

2.加注解

①不指定beanid，默认beanid为类名（首字母小写）

|  |
| --- |
| @Component  **public** **class** MyClass {  **public** **void** show() {  System.***out***.println("这是采用注解的");  }    } |

②指定名称（beanId）

|  |
| --- |
| @Controller("uc")//或 @Controller(value = "uc")  **public** **class** UserController {  **public** **void** show() {  System.***out***.println("这里是uc");  }    } |

3.测试

|  |
| --- |
| @Test  **public** **void** test01() {  ApplicationContext ac = **new** ClassPathXmlApplicationContext("annotation01/spring-annotation.xml");  MyClass m = (MyClass) ac.getBean("myClass");  m.show();  UserController uc = (UserController) ac.getBean("uc");  uc.show();  } |

4.注解的方式依赖注入：

①@Autowired

1）@Autowired默认按类型装配（这个注解是属业spring的）

默认情况下必须要求依赖对象必须存在，如果要允许null值，可以设置它的required属性为false，如：@Autowired(required=false) ，如果我们想使用名称装配可以结合@Qualifier注解进行使用，如下：

|  |
| --- |
| @Autowired()@Qualifier("baseDao")  private BaseDao baseDao; |

2）注意一个接口多个实现类问题

由于@Autowired是根据类型匹配，而一般是用接口实现类来注入的（例：private IxxxDao xxxDao;），如果这个接口有多个实现类就会报错。这时就需要配合@Qualifier注解指定名称注入。

②@Resource

1）@Resource默认按照名称进行装配（这个注解属于J2EE的）

名称可以通过name属性进行指定，如果没有指定name属性，当注解写在字段上时，默认取字段名进行安装名称查找，如果注解写在setter方法上默认取属性名进行装配。当找不到与名称匹配的bean时才按照类型进行装配。但是需要注意的是，如果name属性一旦指定，就只会按照名称进行装配。

|  |
| --- |
| @Resource(name="baseDao")  private BaseDao baseDao; |

三、AOP

1.概念

不同的功能模块，可以提取出相同的功能这个共同的功能是与每一个具体的类所真要实现的业务功能是分离，剥离出来不影响原来模块的功能，这样共同的功能才适合作为aop。

2.应用场景

①日志②性能分析③权限管理④我的足迹

3.范例1

①需要添加的jar包



②spring配置文件里开启aop注解支持

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xmlns:aop=*"http://www.springframework.org/schema/aop"*  xmlns:tx=*"http://www.springframework.org/schema/tx"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd*  *http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd*  *http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop.xsd*  *http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx.xsd*  *"*>  <!-- 使用aop注解 -->  <aop:aspectj-autoproxy></aop:aspectj-autoproxy>  <bean id=*"dao"* class=*"AOP.aop01.BankAccountDao"*></bean>  <bean id=*"lDao"* class=*"AOP.aop01.LogInfoDao"*></bean>  </beans> |

③植入功能类

|  |
| --- |
| @Aspect //这个注解指明了当前这个类是公共功能类  **public** **class** LogInfoDao {    @Pointcut("excution(\* AOP.aop01.\*Dao.\*(..))")  **private** **void** any(){};    @After("any()")  **public** **void** addLog(){  System.***out***.println("记录下来各种操作"+**new** Date());  }  } |

④要被植入的类

|  |
| --- |
| **public** **class** BankAccountDao **implements** IBankAccountDao {  @Override  **public** **void** transfer() {  // **TODO** Auto-generated method stub  System.***out***.println("这里进行转账");  }  } |

⑤测试类

|  |
| --- |
| **public** **static** **void** main(String[] args) {  // **TODO** Auto-generated method stub  ApplicationContext ac =**new** ClassPathXmlApplicationContext("AOP/aop01/beans.xml");  IBankAccountDao dao = (IBankAccountDao) ac.getBean("dao");  dao.transfer();  } |

4.AspectJ的Execution表达式：

就是类似3. ③里的execution里的语法，详情见csdn：https://blog.csdn.net/peng658890/article/details/7223046

5.aop名词概念

[点击跳转](SPRING%20aop%20名词.doc)

6.范例2、3

不具体写了，和范例1差不多，[点击跳转源码](AOP%20code)，看aop2,aop3

四.spring整合mybatis

1.配置文件

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xmlns:aop=*"http://www.springframework.org/schema/aop"*  xmlns:tx=*"http://www.springframework.org/schema/tx"*  xsi:schemaLocation=*"*  *http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd*  *http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd*  *http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop.xsd*  *http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx.xsd*  *"*>  <!-- 引入外部的属性文件 -->  <context:property-placeholder location=*"classpath:db.properties"*/>  <!-- 数据库配置信息 -->  <!-- oracle -->  <bean id=*"dataSource"* class=*"org.springframework.jdbc.datasource.DriverManagerDataSource"*>  <property name=*"driverClassName"* value=*"oracle.jdbc.driver.OracleDriver"*></property>  <property name=*"url"* value=*"jdbc:oracle:thin:@127.0.0.1:1521:oradb"*></property>  <property name=*"username"* value=*"scott"*></property>  <property name=*"password"* value=*"scott"*></property>  </bean>  <!-- mysql -->  <bean id=*"dataSourceMysql"* class=*"org.springframework.jdbc.datasource.DriverManagerDataSource"*>  <property name=*"driverClassName"* value=*"com.mysql.jdbc.Driver"*></property>  <property name=*"url"* value=*"jdbc:mysql://127.0.0.1:3306/mydb?useUnicode=true&amp;characterEncoding=UTF-8"*></property>  <property name=*"username"* value=*"root"*></property>  <property name=*"password"* value=*""*></property>  </bean>  <!-- 配置生成SQLSessionFactory对象的bean -->  <bean id=*"sqlSessionFactory"* class=*"org.mybatis.spring.SqlSessionFactoryBean"*>  <property name=*"dataSource"* ref=*"dataSource"*></property>  <property name=*"configLocation"* value=*"classpath:settings.xml"*></property>  </bean>  <!-- 配置生成处理实体dao的bean(sqlSession) -->  <!-- 这里要注入前面生成的sqlSessionFactory -->  <bean class=*"org.mybatis.spring.mapper.MapperScannerConfigurer"*>  <property name=*"sqlSessionFactory"* ref=*"sqlSessionFactory"*></property>  <property name=*"basePackage"* value=*"mapper"*></property>  </bean>  <!-- 配置事物管理器 -->  <bean id=*"tx"* class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>  <property name=*"dataSource"* ref=*"dataSource"*></property>  </bean>  <!-- 配置注解事物驱动 -->  <tx:annotation-driven transaction-manager=*"tx"*/>  <!-- 告诉spring采用注解的方式进行ioc(开启注解支持) -->  <context:annotation-config></context:annotation-config>  <!-- 要管理的类所在的包 -->  <context:component-scan base-package=*"dao"*></context:component-scan>  </beans> |

2.spring事物

①命名空间和jar包

命名空间里带tx的那个，spring的tx包，aopalliance-.jar包

②配置事物管理器

③使用方法：在DAO加注解@Transactional(propagation = Propagation.*REQUIRED*, isolation = Isolation.*DEFAULT*)

1）加在类上

所有的方法都使用相同的事务管理策略

2）方法上面

不同的方法使用不同的事务策略

④事务的传播特性（propagation）

1）required(默认）

如果已经存在事务，后面产生的新的sql的请求会加入到当前的事务里面，变成一个事务，如果没有存在现成的事务，那么就自己新建一个

2）required\_new

各自拥有自己的事务，不管原来是否有事务，都产生一个新的事务，原来的挂起，新的执行完毕之后，再执行老的。注意，这种传播特性有要求不同的语句放在不同的类里，否则视为同一个事物（既事物要设置在业务层里）。

3）supports

支持当前事务，如果没有就以非事务的方式支持

4）mandatory

支持当前事务，如果当前没有事务，就会报错

5）never

坚决拒绝事务，如果有报错

6）not\_supported

以非事务的方式运行，如果当前有事务则挂起

7）nested

特别适合嵌套场景，没有嵌套场景，则类似required的特点