Spring

1.什么是Spring框架

- 1.是企业级开发的一款解决方案，功能丰富，是一个容器
- 2.核心技术是IOC和AOP，统一进行对象的创建和依赖关系管理，提倡以共同协作来集中开发，使用对象创建对象。

2.Spring框架的优点

- 1.容量高
  （即用户jar包或目录），也就是JAR-JMS在，便于在网络上进行部署，速度更快
- 2.方法中立的编写方法
- 2.面向接口编程
  将面向接口的实现类更换成，对原因解开面向每个模块的工作内容，
  面向接口的实现类更换成，整个项目的业务功能进行切换。
- 3.面向切面
  对于公共的业务（切面）（日志，事务，权限验证，具体本项目的一些公共的功能）集中开发，在使用的时候通过动态代理的方式进行织入。
- 4.没有侵入性的框架
  它相当于是一个容器，对于其它框架提供支持无侵扰，使其向框架更差。

3.Spring框架的IOC

- 1.概念：控制反转IoC(Inversion of Control)，它是一种思想，将容器集中进行对象的创建和依赖管理。
- 2.用种创建方式
  - 1.基于xml
    - 1.通过<bean>标签进行对象的创建
      示例：
      <bean id="stu" class="com.bjpowernode.ss5.Student" />
    - 2.通过依赖注入方式
      - 1.通过setter注入依赖注：必须要提供单参数的set+XXX()方法
        - 1.value属性进行简单类型值注入
          <property name="name" value="张三" />/property>
        - 2.ref属性进行引用类型值注入
          <property name="school" ref="school" />/property>
      - 2.通过构造方法注入依赖注：非单参数属性的构造方法
        - 1.使用构造方法的参数名称进行赋值
          <bean id="stu" class="com.bjpowernode.ss4.Student" />
          <constructor-arg name="age2" value="22" />/constructor-arg>
          <constructor-arg name="name1" value="张三" />/constructor-arg>
          <constructor-arg name="school" ref="school" />/constructor-arg>
        - 2.使用构造方法的参数下标引用进行赋值
          <bean id="stu2Index" class="com.bjpowernode.ss4.Student" />
          <constructor-arg index="1" value="22" />/constructor-arg>
          <constructor-arg index="0" value="张三" />/constructor-arg>
          <constructor-arg index="2" ref="school" />/constructor-arg>
        - 3.使用构造方法的参数顺序进行赋值
          <bean id="stu" class="com.bjpowernode.ss4.Student" />
          <constructor-arg value="张三" />/constructor-arg>
          <constructor-arg value="22" />/constructor-arg>
          <constructor-arg ref="school" />/constructor-arg>
          </bean>
  - 2.基于注解
    - 1.切记在spring的配置文件中添加扫描组件
      <context:component-scan base-package="com.bjpowernode.pojo"/>
    - 2.通过注解的注解
      - @Component：可以创建所有的对象，但一般用于除了控制器，业务逻辑，数据访问对象的创建
      - @Controller：专门用于进行控制器的创建，此注解可以创建用户的请求，并且通过视图解析器
      - @Service：专门用于进行业务逻辑层对象的创建，此注解可以创建业务逻辑层，将上层控制器调用解析，用于访问数据访问层
      - @Repository：专门用来进行数据访问层对象的创建，进行数据库中的操作数据访问
    - 3.赋值的注解
      - @Value：给简单类型（种类基本类型+String）赋值
      - @Autowired：通过按照类型自动装配注入，推荐用byType。
      - @Autowired
        @Qualifier("school")
        @Resource
        两个注解一起使用，表示给引用类型值名称注入，相当于byName
  - 3.配置文件拆分
    - 1.原因：因为项目越来越大，功能越来越复杂，多人开发协作开发
    - 2.拆分策略
      - 1.按功能模块进行拆分
        - applicationContext_user.xml
          <bean id="sController">
          <bean id="sService">
          <bean id="sDao">
        - applicationContext_book.xml
          <bean id="bController">
          <bean id="bService">
          <bean id="bDao">
      - 2.按层进行拆分
        - applicationContext_controller.xml
          <bean id="sController">
          <bean id="bController">
        - applicationContext_service.xml
          <bean id="sService">
          <bean id="bService">
        - applicationContext_dao.xml
          <bean id="sDao">
          <bean id="bDao">
    - 3.整合策略
      - 1.单个文件导入
        - <import resource="applicationContext_school.xml" />/import>
        - <import resource="applicationContext_student.xml" />/import>
      - 2.批量导入
        - <import resource="applicationContext_*.xml" />

4.AOP面向切面编程

- 1.概念是为了使用面向对象编程（OOP）所产生的一种新的编程方式，使用面向切面编程（AOP）可以在不改变原有代码的情况下，通过配置的方式，为现有的程序添加新的功能。
- 2.什么是切面：
  事件（日志，权限，缓存，检查等这些公共的通用逻辑部分）
- 3.写AOP框架
  - 1.拆分切面
    - 1.业务和公共功能分开
    - 2.拆分业务和功能，使用子类代理的方式进行处理，拆分了业务和功能
    - 3.使用静态代理的方式进行业务和业务模式的分离，切面编辑器
    - 4.分离业务和功能部分，切面部门和业务的实现，使用静态代理的方式进行处理（具体AOP的模型）
    - 5.分离业务和功能部分，切面部门和业务部门的实现，使用动态代理的方式进行处理（相似AOP的模型）
  - 2.总结
    - 1.提取业务接口，使业务更灵活
    - 2.提取切面接口，使切面更灵活
    - 3.分别实现业务和切面的实现，替换部门的实现来控制替换整个功能实现
    - 4.使用了JDK的动态代理，完善的提供了SpringBAOP的实现模式，实现切面（公共的通用的）与业务的分离。
- 4.Spring的AOP通知
  - 1.Before通知：在目标方法被调用前调用，涉及接口MethodBeforeAdvice;
  - 2.After通知：在目标方法被调用后调用，涉及接口AfterReturningAdvice;
  - 3.Throws通知：目标方法抛出异常时调用，涉及接口ThrowsAdvice;
  - 4.Around通知：把要切回代理要方法调用，涉及接口MethodInterceptor。
- 5.Aspect框架
  - 1.概念：是一个优秀的面向切面的框架，它扩展了java语言，提供了强大的切面实现。最早要用，它是Ioc的开源的项目
  - 2.常用术语
    - 1. 切面(Aspect)
      公共的，通用的，单独出来的切面，即抽进行开发，例如：日志，事件，权限，检查等，开发完毕后，在主业务调用切点的部分插入到主业务
    - 2.连接点(JoinPoint)
      即可以被切面插入的操作的业务方法，就是我们说的目标方法。
      buy()
      即被切面使业务代码中增强功能的方法
    - 3.切入点(PointCut)
      切入点指一个或多个连接点的集合，也可以是切入点，并可以是具体的切入点，即中的某个方法可以是切入点，切入点指明了切面切入的位置。 ps：final 方法不能被切入点。
    - 4.目标对象(Target)
      目标对象是指被增强功能的对象，了解切面的对象，切面主要是增强目标的对象
      - 5.通知(Advice)
        通知表示你的切面的，通知的作用增强，通知本身还可能切入到目标对象的时间，表目标对象的方法执行前切入切面，还是执行后切入切面，还是环绕切入切面。
    - 6.切面(Aspect)
      切面是一个通知和切入点的集合，通知和切入点共同定义了关于切面的内容，即它们是什么，在何时和何地来执行切面的功能。
  - 3.切入点表达式
    - 1.execution(public * *(..))
      - 1.execution(public * *(..))
        指定切入点为：任何一个以“set”开始的方法。
      - 2.execution(* set*(..))
        指定切入点为：任何一个以“set”开始的方法。
      - 3.execution(* com.xyz.service.*.*(..)) com.xyz.service.One.a() .xyz.service.Two.b()
        指定切入点为：定义在 service 包里的任意类的任意方法。
      - 4.execution(* com.xyz.service.*+.*(..)) com.xyz.service.dao.*(..) com.xyz.service.impl.*(..)
        指定切入点为：定义在 service 包或者子包里的任意类的任意方法。 ‘..’ 出现在类名中，后面跟着一个‘+’，表示包，子包下的所有类
      - 5.execution(* *..service.*+.*(..)) com.service.*(..) com.bjpowernode.service.*(..)
        指定所有包下的 service 子包下所有类（接口）中所有方法为切入点
      - 6.execution(* *..service.*+.*(..)) com.service.*(..) com.bjpowernode.service.*(..)
        指定任何一级包下的 service 子包下所有类（接口）中所有方法为切入点
    - 2.切面
      - 1.切面方法的规范
        1）访问权限是public
        2）方法有参数，其参数是目标方法的返回值
        3）方法返回值定义为object
        4）方法没有异常
      - 2.切面
        - 1. @Before通知
          用来指切入时间，使用此注解声明在使用方法（目标方法）执行前，切入切面功能，是一个前置通知
          - 1.切面方法的规范
            1）访问权限是public
            2）方法有参数，其参数是目标方法的返回值
            3）方法返回值定义为void
            4）方法没有异常
          - 2.切面
            @Before(value = "execution(* com.bjpowernode.ss1.*(..))")
            public void myAspect(){
            //方法的参数传递，参数类型可用object
            //参数是基本类的话用个类型，引用传递的话直
            //当前的Object res 是引用传递，如果是非String类型，即代表变，是String类型，不可变。
            ,
            //可不可以切面参数类型？
            //因为我是在重载时，所以目标方法的引用类型和内容可能改变，所以可以得到目标方法的返回值，也可以更改返回值
            String resultRest = "";
            if(res != null){
            resultRest = (String)res;//stringBeanCase();
            }
            System.out.println("我是前置通知，我在业务方法之后执行,目标方法的返回值是:------"+returnRest);
        - 2. @AfterReturning
          此通知与前置通知，在目标方法执行结束以后执行的时候，也就是说目标方法。
          - 在目标方法执行的时候，方法的返回值是可以修改的。
          - 1.切面方法的规范
            1）访问权限是public
            2）方法有参数，其参数是目标方法的返回值
            3）方法返回值定义为void
            4）方法没有异常
          - 2.切面
            @AfterReturning(value = "execution(* com.bjpowernode.ss10.SomeServiceImpl.*(..))",returning = "res")
            public void myAspect(Object res){
            //方法的参数传递，参数类型可用object
            //参数是基本类的话用个类型，引用传递的话直
            //当前的Object res 是引用传递，如果是非String类型，即代表变，是String类型，不可变。
            ,
            //可不可以切面参数类型？
            //因为我是在重载时，所以目标方法的引用类型和内容可能改变，所以可以得到目标方法的返回值，也可以更改返回值
            String returnRest = "";
            if(res != null){
            returnRest = (String)res;//stringBeanCase();
            }
            System.out.println("我是后置通知，我在业务方法之后执行,目标方法的返回值是:------"+returnRest);
        - 3. @Around
          环绕通知：使用方法是 ProceedingJoinPoint 参数
          - 在目标方法执行前后都可以增强功能（事务），它可以直接改变目标方法的返回值，因为它可以在方法中直接调用目标方法的返回值
          - 1.切面方法的规范
            1）访问权限是public
            2）方法有参数，其参数是ProceedingJoinPoint,且了参数是目标方法，是JoinPoint子类
            3）方法名称自定义
            4）方法没有异常
          - 2.切面
            @Around(value = "execution(* com.bjpowernode.ss10.SomeServiceImpl.*(..))")
            public Object myAspect(ProceedingJoinPoint pjp) throws Throwable {
            //切面增强
            System.out.println("我是前置功能增强.....");
            //目标方法的调用
            Object[] ob = pjp.getArgs();//pjp.proceed();//method.invoke()字工调用方法，调用了doThrow()方法
            //切面增强
            System.out.println("我是后置功能增强.....");
            return res;
            }
        - 4.@After 最终通知（了解）
          无论目标方法是否抛出异常，该增强功能都执行
        - 5.@Point cut 切切入点通过式切切的（了解）
- 4.Aspect的通知
  - 1.@Before通知
    - 2.@AfterReturning
    - 3.@Around
    - 4.@After 最终通知（了解）
    - 5.@Point cut 切切入点通过式切切的（了解）

5.事务

- 1.概念：在数据库中的多个操作之间，保证要么都成功，要么都不成功。所有的操作是一个操作一样。它的四大特性：原子性，一致性，持久性，隔离性
- 2.事务的隔离级别
  - 1.READ_UNCOMMITTED：读未提交，未解决任何发问题。
  - 2.READ_COMMITTED：提交提交，解决脏读，存在不可重复读与幻读。
  - 3.REPEATABLE_READ：可重复读，解决脏读，存在幻读
  - 4.SERIALIZEable：最不快，不同的安全问题
- 3.mysqlGoracle数据库的隔离级别
  - MySql 默认为 REPEATABLE_READ
  - Oracle默认为 READ_COMMITTED
  - 面试点
- 4.事务的传播特性
  - 1.PROPAGATION_REQUIRED：默认的选项
  - 2.PROPAGATION_REQUIRES_NEW：自己的事务，不管之前是否有事务
  - 3.PROPAGATION_SUPPORTS：支持事务，如果之前已经有事务，就支持事务，如果没有，不做开事务
  - 4.PROPAGATION_NEVER：不能运行中事务中，如果包含事务中，抛异常
  - 5.PROPAGATION_NOT_SUPPORTED：不支持事务，运行在非事务的环境
  - 6.PROPAGATION_MANDATORY：必须在事务中，没有事务抛异常
  - 7.PROPAGATION_NESTED：嵌套事务
- 5.xm配置文件的方式声明事务
  - <bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource" />/property>
    /bean>
    <!- 使用xm配置的方式进行事物管理 ->
    <tx:advice id="txAdvice" transaction-manager="transactionManager">
    <tx:attributes>
    <!- 表示哪些要应用这个事务 ->
    <tx:method name="get*" read-only="true" />
    <tx:method name="find*" read-only="true" />
    <tx:method name="find*" read-only="true" />
    <tx:method name="search*" read-only="true" />
    <!- 增删改的操作必要加事务REQUIRED ->
    <tx:method name="insert*" propagation="REQUIRED" />/tx:method>
    <tx:method name="add*" propagation="REQUIRED" />/tx:method>
    <tx:method name="save*" propagation="REQUIRED" no-rollback-for="ArithmeticException" />/tx:method>
    <tx:method name="set*" propagation="REQUIRED" />/tx:method>
    <tx:method name="update*" propagation="REQUIRED" />/tx:method>
    <tx:method name="modify*" propagation="REQUIRED" />/tx:method>
    <tx:method name="change*" propagation="REQUIRED" />/tx:method>
    <tx:method name="delete*" propagation="REQUIRED" />/tx:method>
    <tx:method name="remove*" propagation="REQUIRED" />/tx:method>
    <tx:method name="clear*" propagation="REQUIRED" />/tx:method>
    <tx:method name="empty*" propagation="REQUIRED" />/tx:method>

    <!-
    其它方法支持事务的->
    <tx:method name="*" propagation="SUPPORTS" />/tx:method>
    /tx:attributes>
    /tx:advice>

    <aop:config>
    <!- 指定切入的表达式，切入到业务逻辑层上 ->
    <aop:pointcut id="pointcut" expression="execution(* com.bjpowernode.service.*+.*(..))">
    <aop:advisor advice-ref="txAdvice" pointcut-ref="pointcut" />/aop:advisor>
    /aop:config>
- 6.注解方式的Transactional
  - <bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource" />/property>
    /bean>
    <!- Spring 的事物处理是用两种方式，一种注解，一种xml配置，现在使用注解的方式。 ->
    <tx:annotation-driven transaction-manager="transactionManager" />/tx:annotation-driven>
  - **@Transactional(**
    readOnly = false， true表示只读，false表示写事务(默认)
    timeout = -1， 事物的超时时间不限制(数据库有所变动没有自身上，等待的时间，但还是 等待数据的数据库变动时
    设置的，这里默认设置
    noRollbackFor = ArithmeticException.class, 设置遇到指定的错误不用回滚。
    此处是遇到数字异常不回滚。
    noRollbackForClassName = "ArithmeticException"，遇到数据库常常的名称
    rollbackFor = "ArithmeticException.class"，发生此异常，必须回滚
    ,rollbackForClassName = "ArithmeticException" 发生此异常，必须回滚
  - 7.@Transactional事物原理
    - 1.只能声明在public的method，原因是aop的原理是CGLIB代理的，生成的代理类，只能调用public方法，注解是在类或方法上，这样在配置的这个@Transactional的时候，只能生成代理类，只能在方法有作用，同样还原aop的方法。
    - 2.不能被类内部方法调用，因为代理模式，类内部自调用，不会经过代理类，所以@Transactional不会生效
    - 3.默认情况下，只有遇到RuntimeException，才会回滚事务，checked exception不会回滚事务。
    - 4.@Transactional(propagation=Propagation.REQUIRED) 事务的传播行为
    - 1 <bean id="transactionManager"
    - 2 class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    - 3 <property name="dataSource" ref="dataSource" />
    - 4 /bean>
    - 5 <bean id="transactionManager" class="org.springframework.orm.hibernate3.HibernateTransactionManager">
    - 6 <property name="sessionFactory" ref="sessionFactory" />
    - 7 /bean>
  - 8.不同的被使用的事务管理器