Spring 中 AOP 的应用场景、 Aop 原理、好处?

答:Aop 是 Spring 两大核心之一,叫做面向切面编程,它是 oop 的延续

1. 它主要用于解决方法之间的依赖,如声明式事务管理,它的实现原理是动态代理, 分为 jdk 的动态代理和 cglib 动态代理,

Jdk 动态代理只针对于接口操作,目标类必须实现一个接口,他两是兄弟关系

Cglib 既可以代理有接口的类，也可以代理无接口的类。目标类不能用 final 修饰,代理是目标类的子类

2.作用：在程序运行期间，不修改源码对已有方法进行增强。

可以减少重复代码,提高开发效率以及维护方便!

AOP 相关术语:

Joinpoint(连接点):目标对象的所有方法

Pointcut(切入点):连接点中需要增强的方法

Advice(通知/ 增强):增强的代码,类型:前置通知,后置通知,异常通知,最终通知,环绕通知。

Weaving(织入):是指把增强应用到目标对象来创建新的代理对象的过程。

Aspect(切面):是切入点和通知（引介）的结合。

注意:spring 框架默认情况下，会对有接口的类使用 proxy 代理。没有接口的类使用 cglib 代理(面试)

Spring 中 IOC 的作用与原理？对象创建的过程。

答:

作用与原理:IOC 是 Spring 两大核心之一,叫做控制反转,它的作用是解决类与类之间的依赖,它还有个功能叫作依赖注入,将对象的创建交给 Spring 的容器管理,它的原理是利用反射创建对象

创建的过程:Spring 的容器创建对象有三种方式,分别是默认无参构造函数创建,静态工厂的静态方法创建,实例工厂的方法创建.

解析配置文件中配置的 Bean 信息,根据 class 属性中指定的全限定类名创建该类的对象存入 spring 容器中,该容器的本质是一个 map 集合,bean 标签中的 id 作为 key,class 作为 value,将来可以使用 beanFatory 等对象从 map 中根据 key 取出对象!

介绍 spring 框架。

答:spring 是一款轻量级的 javaEE 开源框架,控制反转和面向切面编程是它的两大内核,主要作用是降低程序之间的耦合,它提供了展示层的框架 springMVC 以及持久层的框架 springMybatis 等,它还可以整合其它框架进行开发

小结:

1.注解方式的通知有顺序要求,使用注解配置代理对象建议使用环绕通知(该通知是手动整合上面四种通知)
2.aop 是一种切面编程思想,当我们需要目标类的对象时,spring 实际返回给我们的是代理对象,代理对象执行 involve 方法,实际上仍然是使用目标类的对象执行方法!
3.可以使用 proxy-target-class 属性指定为 true 表示使用 cglib 动态代理!
4.aop 原理:
以前的事务都是在 dao 层自动提交,多次对数据库的操作就会生成多个连接对象,也就是

多个独立的事务对象,导致转账功能的异常.

解决思路:

需要将转账的功能使用同一个事务进行管理(同一个连接对象),将事务的提交转移到业务层进行手动提交(有个属性),保证一个线程对应一个连接对象,当使用完毕后归还连接对象,并将连接对象和线程进行解绑!

保证一个线程对应一个连接对象的做法:

使用 ThreadLocal(一个线程对应一个该对象)获取连接,如果获取不到则从数据源中获取一个连接对象,并存入 ThreadLocal 对象,如果已经有的直接取出该连接对象(连接对象与线程绑定),从而保证,一个线程只有一个连接对象.先拿出连接对象再操作事务!

11.单元测试中的方法不能有返回值,也不能有参数!否则初始化错误!

12.别人写的类使用 xml 配置,自己写的类使用注解配置,建议以后两种结合使用

1、 Spring 常见创建对象的注解?

创建 bean 对象(前提是提供包扫描):

@Component :用于非三层架构中的类的对象创建

@Controller:用于 web 层对象的创建

@Service:用于业务层对象的创建

@Repository:用于 dao 层对象的创建

@Transactional:注解方式配置事务

依赖注入(本地):

@Autowired:按照类型注入对象

@Qualifier:一般与 Autowired 一起使用,指定匹配的 id 注入对象

@Resource:直接按照给定的 name 值去容器中匹配 ID

@Value:用于注入基本类型和 String 类型的数据

1.SpEL:#{表达式}

2.EL 表达式:${表达式}

1. 写在 Mybatis 的配置文件中,那么就是字符串拼接
2. 写在 JSP 中,从四大域中取值(pageContext,Request,Session,Application)

3.Jquery$(选择器):$()

4.Mybatis 中 OGNL 表达式:#{}

2、 Spring 中用到的设计模式。

答:Spring 容器创建 bean 对象时用到工厂模式

指定 scope 属性时用到单例设置模式

Spring 的优点、缺点

优点:

1. **提供了 IOC 和 DI,降低了类与类之间的耦合,将对象交由 spring 容器管理,使用 autowrite 注入**
2. **aop 采用动态代理实现,降低了方法之间的耦合**
3. **可以整合其它框架**

缺点(springBoot 解决):

1. **配置文件繁琐**
2. **坐标配置繁琐**

Spring Bean 的作用域之间有什么区别?

答:scope 属性,用于指定 bean 的作用范围

singleton：单例的（默认值）

prototype：多例的

request：作用于 web 应用的请求范围

session：作用于 web 应用的会话范围

global-session：作用于集群环境的会话范围（全局会话范围），当不是集群环境时，它就是 session

Bean 对象的生命周期:

单例对象

出生：当容器创建时对象出生
活着：只要容器还在，对象一直活着
死亡：容器销毁，对象消亡
总结：单例对象的生命周期和容器相同

多例对象

出生：当我们使用对象时 spring 框架为我们创建
活着：对象只要是在使用过程中就一直活着。
死亡：当对象长时间不用，且没有别的对象引用时，由 Java 的垃圾回收器回收

依赖注入的三种方式:

1.构造函数注入
2.Set 方法注入
3.注解注入(autowirte)

在 service 接口中定义了一个成员变量 dao 接口的引用,如何生成 dao 接口的对象?

可以使用依赖注入的 3 种方式解决:

常用的方法是在该类中为需要注入的对象提供 set 方法,在该类的 ioc 配置中使用 property 实现,而不能再使用那 3 种创建对象的方式注入对象,会出现死循环!

工厂模式的流程步骤:

定义一个类,在该类中提供获取配置文件中声明创建的对象的方法,在该方法中类加载器读取配置文件再利用反射创建该对象,最终在项目中用到配置文件对象的地方调用该方法即可但使用 newInstance 创建的对象是多个,效率低,应该将工厂类设置成单列的,可以使用一个 map 集合将第一次生产的对象保存,有从 map 中拿出该对象即可!

使用的类中,使用接口接收工厂创建的对象!

aop 原理:

以前的事务都是在 dao 层自动提交,多次对数据库的操作就会生成多个连接对象,也就是多个独立的事务对象,导致转账功能的异常.

解决思路:

需要将转账的功能使用同一个事务进行管理(同一个连接对象),将事务的提交转移到业务层进行手动提交(有个属性),保证一个线程对应一个连接对象,当使用完毕后归还连接对象,并将连接对象和线程进行解绑!

保证一个线程对应一个连接对象的做法:

使用 ThreadLocal 对象(一个线程对应一个该对象)获取连接,如果获取不到则从数据源中获取一个连接对象,并存入 ThreadLocal 对象,如果已经有的直接取出该连接对象(连接对象与线程绑定),从而保证,一个线程只有一个连接对象.先拿出连接对象再操作事务!

Spring 管理事务有几种方式?

答:两种,一种是编程实现事务管理,另一种是配置实现事务管理,其中配置方式实现的管理是通过三组 API 实现的,PlatformTransactionManager 是 spring 的事务管理器,提供了操作事务的方法,TransactionDefinition,设置事务的属性,.TransactionStatus 提供事务的运行状态

spring 中自动装配的方式有哪些?

什么是 IOC,什么又是 DI,他们有什么区别?

答:IOC 指控制反转,DI 指依赖注入,它是控制反转中的内容,都是将对象的创建权交由 spring 的容器管理,降低了程序间的耦合性,便于后期的维护

区别:使用配置方式在 bean 标签中指定 class 属性时都能创建该类的对象,当需要在一个类中使用另一个类的对象时,比如 service 层要调用 dao 层方法时需要 dao 层的对象,这里就可以在 service 层中使用 Aotuwired 从 spring 容器中注入该对象

spring 有哪两种代理方式？请介绍一下。

答:

Jdbc 的动态代理:Jdk 动态代理只针对于接口操作,目标对象必须实现一个接口,代理类与目标类是兄弟关系

Cglib 既可以代理有接口的类，也可以代理无接口的类。目标类不能用 final 修饰,代理类是目标类的子类

SpringMVC 的流程

1. 当启动 Tomcat 服务器的时候，因为在 web.xml 中配置了 load-on-startup 标签，所以会创建 DispatcherServlet 对象，就会加载 springmvc.xml 配置文件 WebApplicationContext
2. WebApplicationContext 开启了注解扫描，那么 HelloController 对象就会被创建,并存入 spring 容器中,然后流程如下

1. **客户端请求访问---->2.前端控制器 DispatcherServlet 接收用户请求,请求查找 handler--->3.处理器映射器根据方法上注解的路径,找到 Controller 注解的类中的方法返回一个执行链给 DispatcherServlet --->4.控制器指挥处理器适配器去匹配相应的方法,然后去执行该方法,返回一个 modleAndView 给前端控制器--->5.前端控制器将该结果交给视图解析器,视图解析器返回 view 对象给前端控制器--->6.前端控制器将 view 对象交由浏览器进行渲染显示**

springMVC 的常用注解:

1.RequestMapping:建立请求 URL 和处理方法之间的对应关系

2.RequestParam:如果表单传递的参数和方法的形参不一致时,封装不上数据,需在"形参位置上"指定 name 属性为表单传递的参数名,然后该注解将表单传递的参数赋

值给方法的形参封装

3.RequestBody:写在形参前面,用来将请求体中 json 格式的字符串转换为 java 对象

4.@ResponseBody:可以写在方法上,也可以写在返回值前,用来将 java 对象转换为 Json 格式字符串并且写出

springMVC 响应数据类型:

1.string 类型

1.Controller 方法返回字符串可以直接指定逻辑视图的名称(success.jsp),默认用的就是转发

如果想要方法返回的是字符串(视图的名称),并且要在 request 域中存值.

1. 在方法的形参上声明一个 Request 对象,并且使用 setAttribute 方法存值
2. 在方法的形参上声明一个 Model 对象.使用 Model 的 addAttribute,这时 SpringMVC 会将 Model 中的键值对都存到 Request 域中.

2.void 类型

默认查找 JSP 页面没有找到。默认会跳转到@RequestMapping(value="/initUpdate") initUpdate 的页面。不会走视图解析器

3.ModelAndView 类型

ModelAndView 对象是 Spring 提供的一个对象，可以用来调整具体的 JSP 视图

注意:

1.DispatcherServlet 会拦截到所有的资源:

导致一个问题就是静态资源（img、css、js）也会被拦截到，从而不能被使用。解决问题就是需要配置静态资源不进行拦截，在 springmvc.xml 配置文件添加不拦截配置:`<mvc:default-servlet-handler />`

2.传统文件上传需要注意事项：

1、设置 enctype

2、post 请求方式

3、文件上传表单项

Springmvc 的优点

1. 接收前端数据或返回数据给前端更加方便
2. 采用 mvc 架构,将控制层解耦

Mybatis 中一级缓存与二级缓存区别?

Mybatis 中的缓存

什么是缓存?

存在于内存中的临时数据。

为什么使用缓存?

减少和数据库的交互次数，提高执行效率。

什么样的数据能使用缓存，什么样的数据不能使用?

适用于缓存：

经常查询并且不经常改变的。

数据的正确与否对最终结果影响不大的。

不适用于缓存：

经常改变的数据

数据的正确与否对最终结果影响很大的。

例如：商品的库存，银行的汇率，股市的牌价。

Mybatis 中的一级缓存和二级缓存

一级缓存：

它指的是 Mybatis 中 SqlSession 对象的缓存。

当我们执行查询之后，查询的结果会同时存入到 SqlSession 为我们提供一块区域中。该区域的结构是一个 Map。当我们再次查询同样的数据，mybatis 会先去 sqlsession 中查询是否有，有的话直接拿出来用。

当 SqlSession 对象消失时，mybatis 的一级缓存也就消失了。

触发清空一级缓存的情况

一级缓存是 SqlSession 范围的缓存，当调用 SqlSession 的修改，添加，删除，commit()，close()等方法时，就会清空一级缓存。

二级缓存:

它指的是 Mybatis 中 SqlSessionFactory 对象的缓存。由同一个 SqlSessionFactory 对象创建的 SqlSession 共享其缓存。

自己的总结:

```
<!--namespace 的值是接口的全类名,-->
<mapper namespace="cn.itheima.dao.RoleDao">


    <!--
       resultMap 作用是告诉 mybatis 当调用方法执行完 sql 语句后返回的数据应该怎么封装,多用于数据库表的字段与实体类属性不对应时,
       resultMap 的 id 是唯一标识,后面用于 sql 语句的方法上作为引用该 resultMap 的配置,告诉当 sql 执行完毕后如何封装数据
       type 指封装到哪个类中,使用全类名,当主配置文件使用 page 属性指定到接口位置的名称时,这里可以直接写类名
       property 指实体类中的属性,column 指数据库中的字段,将来将 column 的值通过 setXXX 方法设置进实体类的属性中

    -->
    <resultMap id="roleMap" type="cn.itheima.domain.Role">
        <id property="roleId" column="rid"></id> <!--数据库是 id 这里 sql 取了别名说以是 rid-->
        <result property="roleName" column="role_name"></result>
        <result property="roleDesc" column="role_desc"></result>


        <!--
        上面完成了对 Role 实体类属性的封装,一对一关系中 Role 还声明了 User 类对象的引用,所以还需完成对该引用的封装
        同样 column 指 User 类属性,column 指数据库中表的字段,将查询到的 column 使用 set 方法封装到实体类属性中,最后
        将整个 User 对象赋值给 collection property="user"中的 user

        -->
        <collection property="user"   ofType="cn.itheima.domain.User">
            <id column="id" property="id"></id>
            <result column="username" property="username"></result>
            <result column="address" property="address"></result>
            <result column="sex" property="sex"></result>
            <result column="birthday" property="birthday"></result>
        </collection>
    </resultMap>



    <!-- 查询角色信息及角色对应的用户信息,id 是接口的方法名,resultMap 表示将 sql 查询到的结果集封装到 resultMap 定义的内容中 -->
        <select id="findAll" resultMap="roleMap">
            select
```

```
            u.*,r.id as rid,r.role_name,r.role_desc
        from role r
        left outer join user_role ur    on r.id = ur.rid
        left outer join user u on u.id = ur.uid

    </select>
```

重点知道执行流程的细节:

```
<mapper namespace="com.itheima.dao.IUserDao">

    <!-- 定义 User 的 resultMap-->
    <resultMap id="userAccountMap" type="user">
        <id property="id" column="id"></id>
        <result property="username" column="username"></result>
        <result property="address" column="address"></result>
        <result property="sex" column="sex"></result>
        <result property="birthday" column="birthday"></result>
        <!-- 配置 user 对象中 accounts 集合的映射 -->
        <collection property="accounts" ofType="account"
select="com.itheima.dao.IAccountDao.findAccountByUid" column="id"></collection>
    </resultMap>

    <!-- 查询所有 -->
    <select id="findAll" resultMap="userAccountMap">
        select * from user

</mapper>
```

一级缓存示例中:

sql 语句属于多表查询(左外连接查询),查询一个用户的同时显示该用户的所有账户信息!

多表查询的话直接将两张表的所有数据查询封装显示,不存在缓存!

缓存的意思是,查询用户时,不必将用户的所有账户都查询出来,所以需要将 sql 查询语句进行拆分

流程是:

通过 select 标签的 id 属性.可以得到”全限定名”即接口的全类名+方法名,使用它可以执行 sql 语句只查询用户的所有信息,结果返回给 resultMap 指定的配置中:
resultMap 标签的 id 属性是唯一标识,在 select 标签中可以通过该 id 的值引用 resultMap 中的相关配置:type 值表示 sql 查询结果封装的类型,这里查询到 user 表中所有信息封装为 user 对象,但下面还有一个 collection 标签,该标签配置的是 User 实体类中 account 类的对象引用,所以还需要将结果集封装到该属性中,如何封装?
`select` 标签又指定了一个全限定名,该全限定名的作用是查询所有的账户数据,要获得这些数据需要使用 user 类中的 id,所以将 column 的值(user 中的 id)作为参数传递进方法中执行查询账户数据,查询的结果封装到 ofType 指定的 Account 类中,然后将该对象赋值给 User 类中定义的 Acconut 对象的引用!

全限定名:接口全类名+方法,
之所以可以执行 sql,是因为在主配置文件中创建了一个 map 集合,key 是全类名.方法名,value 是 sql+返回值,所以可以根据全限定名作为 key 找到对应的 sql 执行,再将结果返回

类的属性与数据库中的字段不一致时:
xml 使用 rsultMap 解决
注解使用 Results 解决
注解中@one 替换 xml 中 assiontion(对象时,集合是 collection)的内容,它的 select 通过 nameSpace 属性找到 sql

代理对象调用方法实际上是调用 sqlSession 的方法,代理对象调用增删改,以及另外三个方法都会清楚一级缓存

ResaultMap 和 ResaultType 的区别

resuleMap:实体类属性与字段不一致时告诉 mybatis 查询的结果集如何封装

ResaultType :与方法的返回值类型相同

一对一关系:association

一对多/多对一:collection

3、 mybatis 如何处理批量插入

mybatis 有什么优点?

它是一个持久层框架，解决项目对数据库的 CRUD 操作。

Mapper 映射文件与 Dao 接口之间的对应关系

① mapper 中的 namespace 值与接口的全限定名相同
② Mapper 中的 id 与方法名相同
③ Mapper 中的 resultType 与方法的返回值类型相同
④ Mapper 中的 parameterType 与方法的参数类型相同

#{} 和 ${} 的区别是什么?

#{}:OGNL 表达式,如果 sql 语句需要参数,可以使用来获取参数的值

${} :el 表达式

mybatis 动态 SQL 是做什么的？都有哪些动态 SQL ？能简述一下动态 SQL 的执行原理吗?

If/ where /foreach

JDBC 编程有哪些不足之处，MyBatis 是如何解决这些问题的?

jdbc 问题总结如下：

1、 数据库链接创建、释放频繁造成系统资源浪费从而影响系统性能，如果使用数据库链接池可解决此问题。
2、 Sql 语句在代码中硬编码，造成代码不易维护，实际应用 sql 变化的可能较大，sql 变动需要改变 java 代码。
3、 使用 preparedStatement 向占有位符号传参数存在硬编码，因为 sql 语句的 where 条件不一定，可能多也可能少，修改 sql 还要修改代码，系统不易维护。
4、 对结果集解析存在硬编码（查询列名），sql 变化导致解析代码变化，系统不易维护，如果能将数据库记录封装成 pojo 对象解析比较方便。

解决(重点):

Mybatis 封装了 jdbc 操作的很多细节,开发者只需关注 sql 本身,mybatis 通过 xml 或注解方式将要执行的各种哪个 statement 配置起来,通过 ORM 思想将实体类与数据库映射,最后由 mybatis 执行 sql 并将结果映射成 java 对象返回!

ORM 对象关系映射:

⑤ 将 Java 类与数据库表对应
⑥ 将类中的属性与数据库中的字段对应
⑦ 查询出表中的一条数据,将会封装成类的一个对象

Spring 整合其它框架

整合的思路

1. 先搭建整合的环境
2. 先把 Spring 的配置搭建完成
3. 再使用 Spring 整合 SpringMVC 框架
4. 最后使用 Spring 整合 MyBatis 框架

Spring 整合 springmvc:

启动 tomcat 服务器的时候,需要加载 spring 的配置文件,使用依赖注入的方式,创建 service 层类的对象,这样在 springMVC 中才能调用 service 层的方法!

如何加载 spring 的配置文件?

答:服务器启动的时候 ServletContext 创建,服务器关闭的时候 servletContext 销毁,有一类监听器,可以监听 ServletContext 对象的创建和销毁,所以使用监听器去加载 spring 的配置文件,它会创建 web 版本的工厂,将配置文件的信息存储到 servletContext 对象中,使用这些信息时就可以从全局对象中获取!

<!--配置 spring 的监听器,默认之家在 WEB-INF 目录下的 applicationCintext.xml 配置文件,作用是 spring 用于整合其他框架-->
<listener>

```
<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class
>
        </listener>
        <!--设置配置文件的路径,让监听器去这个路径下加载资源-->
        <context-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath:applicationContext.xml</param-value>
        </context-param>
```

spring 整合 mybatis 原理:

只要 service 层中能注入 dao 层的代理对象,即将 dao 的代理对象交个 spring 容器管理,调用 dao 层的方法操作到数据库,那证明整合成功了.

如何将 dao 的代理对象交个 spring 容器管理?

答:

```
<!--在 spring.xml 中整合 Mybatis 框架,目的是创建出代理对象交由 Spring 容器管理,然后注入的方式能在 service 层中拿到 dao 层接口的代理对象-->
        <!--1.配置连接池-->
        <bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
            <property name="driverClass" value="com.mysql.jdbc.Driver"/>
            <property name="jdbcUrl" value="jdbc:mysql:///ssm"/>
            <property name="user" value="root"/>
            <property name="password" value="123"/>
        </bean>
```

```
        <!--2.配置 SqlSessionFactory 工厂,让框架创建动态代理对象,然后交给 ioc 管理-->
        <bean id="sqlSessionFactoryBean" class="org.mybatis.spring.SqlSessionFactoryBean">
            <property name="dataSource" ref="dataSource"/>
        </bean>

        <!--3.配置 AccountDao 接口所在的包,为框架指定为哪个接口生成动态代理对象-->
        <bean id="mapperScanner" class="org.mybatis.spring.mapper.MapperScannerConfigurer">
            <property name="basePackage" value="cn.itcast.dao"></property>
        </bean>
```