1、描述IOC和DI的区别，AOP是什么，以及在spring中的应用

解释一下（介绍一下）什么是 IOC？IOC 和 DI 的区别？

1) IOC 和 DI 是从不同的维度所起的名称。
IOC 表示控制反转，DI 表示依赖注入。
2) 控制反转指的是，在我的***项目中，控制层需要 new 出业务层（biz）对象，业务层需要 new 出数据访问层（DAO）对象，这样的话，属于主动获取（因为是我们主动 new 的），那么有了 IOC 之后，我可以把我（Biz）内部需要的 DAO 实例这样的需求告诉 spring（在配置文件中描述），然后 spring 其实是一个大的 BeanFactory，它负责创建菜单（ApplicationContext.xml）里面的所有的 bean 对象，然后根据配置文件中的需要进行注入。简单讲，就是之前是我主动 new，主动创造，现在变成我只需要在配置文件中填写我的需求，spring 就会创造好给我，spring 并负责这些对象的生命周期管理，这个称之为控制反转（创建和注入反转给了 spring）。
3) DI 指的是依赖注入，先解释依赖，依赖指的是 Biz 内部需要 DAO，没有 DAO，那么 Biz 就不能工作，Action 内部需要 Biz，没有 Biz 的话 Action 就不能工作，我们称之为依赖。如果是我们在代码里面 new 的话，那么 Action 就会和具体的 Biz 实现类耦合了，如果我们想平滑的切换不同的 Biz 实现，那么势必对代码造成改动，那这样的话不利于降低耦合，因此我们把这种依赖关系转移到配置文件中，让 spring 给我们注入这种依赖关系，称之为依赖注入。
4) 总结：IOC 和 DI 其实指的是一码事，只是从不同的维度所起的名称；控制反转指的是之前是我自己创建，现在是 spring 创建好给我，并且 spring 负责维护创建的这些 bean 的生命周期，控制权完全交给了 spring。DI 指的是依赖注入，之前依赖关系是我在代码里面自己维护的，现在我要转移到配置文件，一旦依赖关系有变化我只需要改动配置文件，而不需要重新修改类代码重新编译。

AOP:面向切面编程

分为：静态代理、动态代理（JDK代理，CGLIB代理）

举例：武大郎卖烧饼

武大郎：目标对象（Traget）

卖烧饼：目标方法（本质工作）

武大郎分店（代理商）：为了提高市场竞争力，卖烧饼时，可以送大麦茶

送大麦茶：额外操作（Advice）

分店卖烧饼也没有问题了，武大郎需要推出新产品：卖煎饼，卖手抓饼

为了推广新品，因此卖煎饼和卖手抓饼时才送大麦茶，卖烧饼不送大麦茶。

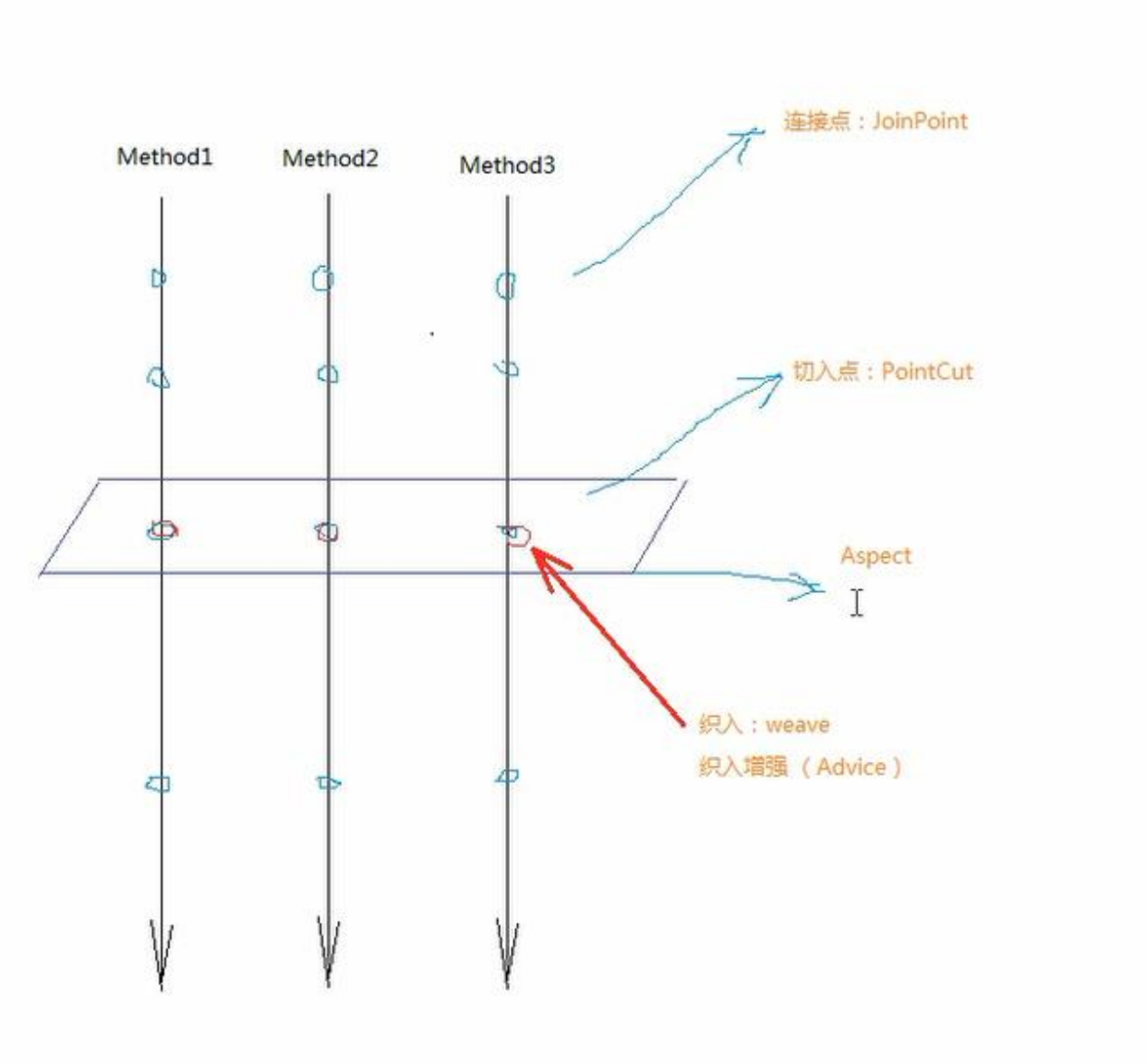
连接点：JoinPoint

具体哪个点攻击：切入点（PointCut）

点连成一个面：切面（Aspect）

往切入点注入一些事物，叫织入（weave），织入增强（Advice）

项目用到AOP的地方：对事务的管理（方法有 C U R D 增删改查方法需要添加可读写事务：当操作成功后，我们需要提交事务，当操作失败时，我们需要回滚事物操作。当两个操作重叠时，我们需要合并事物等等）




```
connection.beginTransaction() //1.开启事务
try{
        //增、删、改    --------只有这里的操作才是和业务密切相关的
        trans.commit()
} catch(Exception ex){
        //回滚事务操作
        trans.rollback();
}finllay{
        //关闭连接
}
```

```
<tx:pointcut id="tx_pc" expression="切点表达式">
<tx:advice id="tx_advice" ......>
        <tx:attribute method="get*" read-only>
        <tx:attribute method="*" />
</tx:advice>
```

作为

面向对象编程的一种补充，广泛应用于处理一些具有横切性质的服务，如事物管理、安全检查、缓存、对象池管理等。AOP实现的关键就在于AOP框架自动创建的AOP代理，AOP代理则可分为静态代理和动态代理两大类，其中静态代理是指使用AOP框架提供的命令进行编译，从而在编译阶段就可生成AOP代理类，因此也称为编译时增强；而动态代理则在运行时借助JDK动态代理，因此称为运行时增强

```
<aop:aspectj-autoproxy/>

<bean id="personServiceImpl" class="com.serviceImpl.PersonServiceImpl"></bean>
<bean id="personInterceptor" class="com.service.MyInterceptor2"></bean>

<aop:config>
    <aop:aspect id="asp" ref="personInterceptor">
        <aop:pointcut id="myCut" expression="execution (* com.serviceImpl.PersonServiceImpl.*(..))"/>
        <aop:before pointcut-ref="myCut" method="doAccessCheck"/>
    </aop:aspect>
</aop:config>
```

```
//IOC
ApplicationContext ac = new ClassPathXmlApplicationContext("beans.xml");
PersonDAO pd = (PersonDAO) ac.getBean("personDAO");
pd.save();
//DI
PersonService ps = (PersonService) ac.getBean("personService");
ps.save();
```

（1）AOP指的是面向切面编程

（2）在AOP中，可以切入的地方我们称之为切入点（PointCut）；而实际我们会在方法上进行切入，具体切入的点我们称之为连接点（JointPoint）；符合切入规则的方法有很多，那么从纵向上看，n多个点组成一个面，所以我们称之为切面（Aspect）.目标对象称之为Target;额外操作我们称之为Advice(或者增强)，将增强添加到目标对象的操作我们称之为Weave（织入），AOP底层可以使用动态代理。

（3）我在项目中使用到了AOP技术，在做事物管理的时候，我将Biz中的各个方法看成是目标对象的本质工作（Traget）,然后将事物策略看成是增强（Advice）,我们需要在Biz的各个业务方法上织入增强。这样可以统一的做事物管理。

```xml
<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
    <property name="driverClass" value="org.gjt.mm.mysql.Driver"/>
    <property name="jdbcUrl" value="jdbc:mysql://localhost:3306/fruit_db"/>
    <property name="user" value="root"/>
    <property name="password" value="ok"/>
    <property name="initialPoolSize" value="5"/>
    <property name="maxPoolSize" value="50"/>
</bean>

<bean id="sessionFactory" class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
    <property name="dataSource" ref="dataSource"/>
    <property name="configLocation" value="classpath:hibernate.cfg.xml"/>
</bean>

<bean id="transactionManager" class="org.springframework.orm.hibernate4.HibernateTransactionManager">
    <property name="sessionFactory" ref="sessionFactory"/>
</bean>

<tx:advice id="tx" transaction-manager="transactionManager">
    <tx:attributes>
        <tx:method name="get*" read-only="true"/>
        <tx:method name="*"/>
    </tx:attributes>
</tx:advice>

<aop:config>
    <aop:pointcut id="tx_pc" expression="execution(* com.gem.fruit.biz.impl.*.*(..))"/>
    <aop:advisor advice-ref="tx" pointcut-ref="tx_pc"/>
</aop:config>

<bean id="fruitDAO" class="com.gem.fruit.dao.impl.FruitDAOImpl">
    <property name="sessionFactory" ref="sessionFactory"/>
</bean>
```

2、Spring中做事物管理时事物策略有哪些?

（1）事物有四大特征：ACID；分别是原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation）、持久性（Durability）

（2）原子性指的是事物是一个整体，不可分割

（3）一致性指的是事物的前后状态要求保持一致，比如两个账号转账，要不转账成功（A.钱 - 100 ， B.钱 + 100），要不就转账不成功

（4）隔离性指的是两个事物互不干扰

（5）持久性指的是一旦事物提交，那么对数据库的影响是永久的

（6）事物的隔离性包括：脏读；不可重复读

假设有A、B两个事物

脏读：A事物对数据库中的记录做了更改，但是未提交，此时B事物读取到这些更新后的数据，然后A又回滚了事物，此时出现脏读。

51. spring 中做事务管理时事务策略有哪些？

1) 事务有四大特性：ACID；分别是原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation）、持久性（Durability）；
2) 原子性指的是事务是一个整体，不可分割。
3) 一致性指的是事务的前后状态要求保持一致，比如两个账号转账，要不转账成功（A 钱少了，B 钱多了），要不就转账不成功（AB 的钱不变），这叫一致性。
4) 隔离性指的是两个事务互相不干扰 (延伸面试题：隔离级别有哪些？)
5) 持久性，也称之为永久性，指的是一旦事务被提交，那么对数据库的影响是永久的。
6) 事务的隔离性包含：脏读；不可重复度；幻像读（虚读）；
假设有 A、B 两个事务
7) 脏读：A 事务对数据库中的记录做了更改，但是未提交，此时 B 事务读取到这些更新后的数据，然后 A 又回滚了事务，此时则出现了脏读。
8) 不可重复读：A 事务在第一个时间点读取记录，然后 B 事务修改了这些记录数据，然后 A 再次读取这些事务的时候，发现和之前读取的不一致，则出现了不可重复读。
9) 虚读(幻读)：A 事务获取到一批数据，将某一列的值（假设为 1）修改为 2。此时，B 事务执行了 insert 操作，插入的数据这一列的值为 1，然后提交了事务，这个时候 A 事务再次读取这一批数据的时候，发现有一条数据不是为 2，而是为 1，这样就发生了幻读。
10) 脏读和虚读都是针对已经存在的数据；幻读是新增了数据。
那么如何避免上面的情况呢？MySQL 数据库为我们提供了四种隔离级别机制：
11) Serializable（串行化）： 可避免脏读、不可重复读、幻读。但是性能最差
12) Repeatable read (可重复读)：可避免脏读、不可重复读的发生。
13) Read committed (读已提交)：可避免脏读的发生。
14) Read uncommitted (读未提交)：最低级别，任何情况都无法保证
以上四种隔离级别最高的是 Serializable 级别，最低的是 Read uncommitted 级别，当然级别越高，执行效率就越低。像 Serializable 这样的级别，就是以锁表的方式(类似于 Java 多线程中的锁)使得其他的线程只能在锁外等待，所以平时选用何种隔离级别应该根据实际情况。在 MySQL 数据库中默认的隔离级别为 Repeatable read (可重复读)

3、@Transactional注解是干什么的?

52. @transactional 注解是干什么的？

事物传播行为介绍:
@Transactional(propagation=Propagation.REQUIRED) ：如果有事务，那么加入事务，没有的话新建一个(默认情况下)
@Transactional(propagation=Propagation.NOT_SUPPORTED) ：容器不为这个方法开启事务

@Transactional(propagation=Propagation.REQUIRES_NEW) ：不管是否存在事务,都创建一个新的事务,原来的挂起,新的执行完毕,继续执行老的事务
@Transactional(propagation=Propagation.MANDATORY) ：必须在一个已有的事务中执行,否则抛出异常
@Transactional(propagation=Propagation.NEVER) ：必须在一个没有的事务中执行,否则抛出异常(与 Propagation.MANDATORY 相反)
@Transactional(propagation=Propagation.SUPPORTS) ：如果其他 bean 调用这个方法,在其他 bean 中声明事务,那就用事务.如果其他 bean 没有声明事务,那就不用事务.
事物超时设置:
@Transactional(timeout=30) //默认是 30 秒
事务隔离级别:
@Transactional(isolation = Isolation.READ_UNCOMMITTED)：读取未提交数据(会出现脏读，不可重复读) 基本不使用
@Transactional(isolation = Isolation.READ_COMMITTED)：读取已提交数据(会出现不可重复读和幻读)
@Transactional(isolation = Isolation.REPEATABLE_READ)：可重复读(会出现幻读)
@Transactional(isolation = Isolation.SERIALIZABLE)：串行化
MYSQL：默认为 REPEATABLE_READ 级别
SQLSERVER：默认为 READ_COMMITTED

2、spring如何对事物进行控制