上节课成回顾

- 面向切片编程 AOP
- 通过动态代理 + 反射 来实现，非入侵式的 功能增强
  - 第一步 我们定义一个 切片类
  - 我们声明一个 切入点
  - 我们定义 通知方法
  - 我们通过通知方法拿到 连接点
- 动态代理 （ 面试可能被问到 ）
- AOP 实现原理 需要背题
- 动态代理 实现原理
- 静态代理 实现原理

实现JDBC 的增删改查

我们学过 Servlet 知道有个 功能强大 JDBC 用来连接数据库

那么我们现在 已经升级， 我们现在使用 Spring框架了， JDBC 也升级了

Spring框架 他把 JDBC 给封装了一下 Spring-jdbc

1配置依赖

```
<!--spring jdbc  Spring 持久化层支持jar包-->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-jdbc</artifactId>
        <version>6.0.2</version>
    </dependency>
    <!-- MySQL驱动 -->
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>8.0.30</version>
    </dependency>
    <!-- 数据源 -->
    <dependency>
        <groupId>com.alibaba</groupId>
        <artifactId>druid</artifactId>
        <version>1.2.15</version>
    </dependency>
```

2 创建配置档

```
jdbc.properties
jdbc.user=root
jdbc.password=root
jdbc.url=jdbc:mysql://localhost:3306/test?characterEncoding=utf8&useSSL=false
jdbc.driver=com.mysql.cj.jdbc.Driver
```

3 配置beans对象

```
<?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"
       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">

    <!-- 导入外部属性文件 -->
    <context:property-placeholder location="classpath:jdbc.properties" />

    <!-- 配置数据源 -->
    <bean id="druidDataSource" class="com.alibaba.druid.pool.DruidDataSource">
        <property name="url" value="${jdbc.url}"/>
        <property name="driverClassName" value="${jdbc.driver}"/>
        <property name="username" value="${jdbc.user}"/>
        <property name="password" value="${jdbc.password}"/>
    </bean>

    <!-- 配置 JdbcTemplate -->
    <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
        <!-- 装配数据源 -->
        <property name="dataSource" ref="druidDataSource"/>
    </bean>

</beans>
```

配置Spring事务

1 配置TransactionManager事务对象

```
<bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="druidDataSource"></property>
</bean>
<-- 开启事务的注解驱动 -->
<tx:annotation-driven transaction-manager="transactionManager" />
```

我们都知道， 我们框架 其实就是 一堆 类对象， 在Spring框架内， 叫Bean对象

我们也知道 我们使用 Spring框架的 Ioc 功能， 是为了更加方便， 配置对象和对象之间的关系

节省代码量， 如果不是用 Ioc ， 我们要自己维护对象之间的关系， 特别是 要 new 很多对象， 并且

我们 还要 对对象进行初始化， 有了 IOC之后， 这些都简单多了

1. 事务注解的范围
    a. 方法
    b. 类
1. 事务的属性
    a. 超时
    b. 回滚策略
        i. 可以通过@Transactional中相关属性设置回滚策略
        ii. rollbackFor属性：需要设置一个Class类型的对象
        iii. rollbackForClassName属性：需要设置一个字符串类型的全类名
        iv. noRollbackFor属性：需要设置一个Class类型的对象
        v. rollbackFor属性：需要设置一个字符串类型的全类名
    c. 事务传播
        i. REQUIRED：支持当前事务，如果不存在就新建一个(默认)**【没有就新建，有就加入】**
        ii. SUPPORTS：支持当前事务，如果当前没有事务，就以非事务方式执行**【有就加入，没有就不管了】**
        iii. MANDATORY：必须运行在一个事务中，如果当前没有事务正在发生，将抛出一个异常**【有就加入，没有就抛异常】**
        iv. REQUIRES_NEW：开启一个新的事务，如果一个事务已经存在，则将这个存在的事务挂起**【不管有没有，直接开启一个新事务，开启的新事务和之前的事务不存在嵌套关系，之前事务被挂起】**
        v. NOT_SUPPORTED：以非事务方式运行，如果有事务存在，挂起当前事务**【不支持事务，存在就挂起】**
        vi. NEVER：以非事务方式运行，如果有事务存在，抛出异常**【不支持事务，存在就抛异常】**
        vii. NESTED：如果当前正有一个事务在进行中，则该方法应当运行在一个嵌套式事务中。被嵌套的事务

可以独立于外层事务进行提交或回滚。如果外层事务不存在，行为就像REQUIRED一样。

【有事务的话，就在这个事务里再嵌套一个完全独立的事务，嵌套的事务可以独立的提交和回滚。没有事务就和REQUIRED一样。】

d. 事务隔离级别

i. 隔离级别一共有四种：

ii. 读未提交：READ UNCOMMITTED, 允许Transaction01读取Transaction02未提交的修改。

iii. 读已提交：READ COMMITTED, 要求Transaction01只能读取Transaction02已提交的修改。

iv. 可重复读：REPEATABLE READ, 确保Transaction01可以多次从一个字段中读取到相同的值，即Transaction01执行期间禁止其它事务对这个字段进行更新。

v. 串行化：SERIALIZABLE, 在Transaction1执行期间，禁止其它事务对这个表进行添加、更新、删除操作。可以避免任何并发问题，但性能十分低下。

```
@Transactional(isolation = Isolation.DEFAULT)//使用数据库默认的隔离级别
@Transactional(isolation = Isolation.READ_UNCOMMITTED)//读未提交
@Transactional(isolation = Isolation.READ_COMMITTED)//读已提交
@Transactional(isolation = Isolation.REPEATABLE_READ)//可重复读
@Transactional(isolation = Isolation.SERIALIZABLE)//串行化
```

各个隔离级别会存在的问题

隔离级别	脏读	不可重复读	幻读
READ UNCOMMITTED	有	有	有
READ COMMITTED	无	有	有
REPEATABLE READ	无	无	有
SERIALIZABLE	无	无	无

全注解开发