/\*\*

\* 声明式事务：

\*

\* 环境搭建：

\* 1、导入相关依赖

\* 数据源、数据库驱动、Spring-jdbc模块

\* 2、配置数据源、JdbcTemplate（Spring提供的简化数据库操作的工具）操作数据

\* 3、给方法上标注 @Transactional 表示当前方法是一个事务方法；

\* 4、 @EnableTransactionManagement 开启基于注解的事务管理功能；

\* @EnableXXX

\* 5、配置事务管理器来控制事务;

\* @Bean

\* public PlatformTransactionManager transactionManager()

\*

\*

\* 原理：

\* 1）、@EnableTransactionManagement

\* 利用TransactionManagementConfigurationSelector给容器中会导入组件

\* 导入两个组件

\* AutoProxyRegistrar

\* ProxyTransactionManagementConfiguration

\* 2）、AutoProxyRegistrar：

\* 给容器中注册一个 InfrastructureAdvisorAutoProxyCreator 组件；

\* InfrastructureAdvisorAutoProxyCreator：？

\* 利用后置处理器机制在对象创建以后，包装对象，返回一个代理对象（增强器），代理对象执行方法利用拦截器链进行调用；

\*

\* 3）、ProxyTransactionManagementConfiguration 做了什么？

\* 1、给容器中注册事务增强器；

\* 1）、事务增强器要用事务注解的信息，AnnotationTransactionAttributeSource解析事务注解

\* 2）、事务拦截器：

\* TransactionInterceptor；保存了事务属性信息，事务管理器；

\* 他是一个 MethodInterceptor；

\* 在目标方法执行的时候；

\* 执行拦截器链；

\* 事务拦截器：

\* 1）、先获取事务相关的属性

\* 2）、再获取PlatformTransactionManager，如果事先没有添加指定任何transactionmanger

\* 最终会从容器中按照类型获取一个PlatformTransactionManager；

\* 3）、执行目标方法

\* 如果异常，获取到事务管理器，利用事务管理回滚操作；

\* 如果正常，利用事务管理器，提交事务

\*

\*/

@EnableTransactionManagement

@ComponentScan("com.atguigu.tx")

@Configuration

public class TxConfig {

//数据源

@Bean

public DataSource dataSource() throws Exception{

ComboPooledDataSource dataSource = new ComboPooledDataSource();

dataSource.setUser("root");

dataSource.setPassword("123456");

dataSource.setDriverClass("com.mysql.jdbc.Driver");

dataSource.setJdbcUrl("jdbc:mysql://localhost:3306/test");

return dataSource;

}

//

@Bean

public JdbcTemplate jdbcTemplate() throws Exception{

//Spring对@Configuration类会特殊处理；给容器中加组件的方法，多次调用都只是从容器中找组件

JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource());

return jdbcTemplate;

}

//注册事务管理器在容器中

@Bean

public PlatformTransactionManager transactionManager() throws Exception{

return new DataSourceTransactionManager(dataSource());

}

}