**一.概念**

**1.什么是事务**

（1）事务是数据库操作最基本单元，逻辑上一组操作，要么都成功，如果有一个失败所有操作都失败。

（2）典型场景：银行转账

A转账给B100元

A少100，B多100

**2.事务的四个特性（ACID）**

原子性

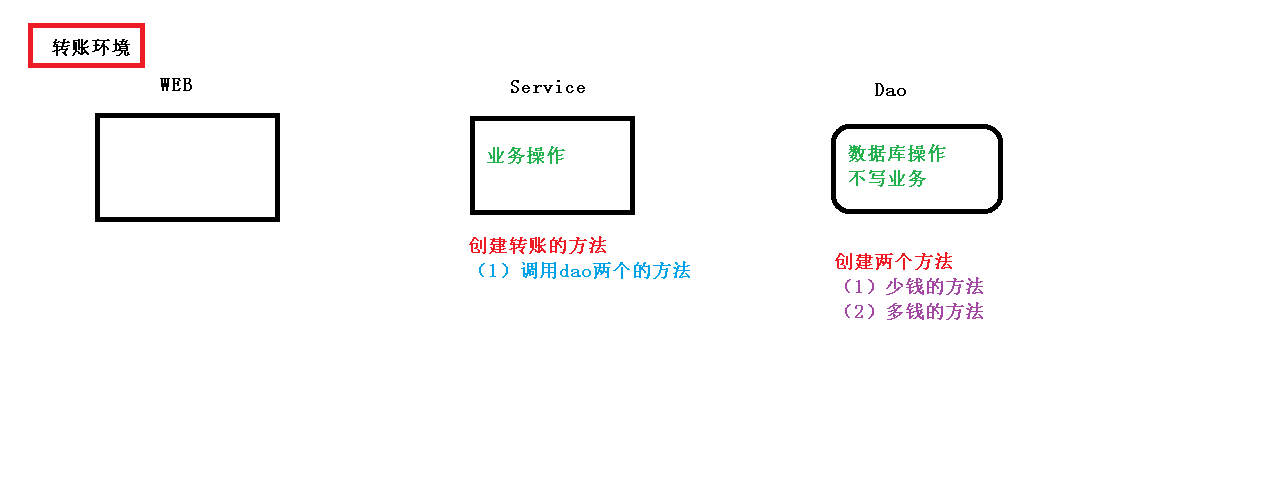
一致性

隔离性

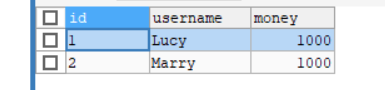
持久性

**二.搭建事务操作环境**

1.流程图



2.创建数据库，添加记录



3.创建service，搭建dao，完成对象创建和注入关系

（1）service注入dao，在dao注入jdbcTemplate，在jdbcTemplate注入DataSource

核心代码：

/\*\*

\* @author xh

\* @date 2022/10/17

\* @apiNote

\*/

@Component

public class AccountService {

//注入dao

@Autowired

private AccountDao accountDao;

}

@Component

public class AcountDaoImpl implements AccountDao{

@Autowired

private JdbcTemplate jdbcTemplate;

@Override

public void reduceMoney() {

}

@Override

public void addMoney() {

}

}

//jdbctemplate注入DataSource

<!-- 数据库连接池 -->

<bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource"

destroy-method="close">

<property name="url" value="jdbc:mysql:///user\_db" />

<property name="username" value="root" />

<property name="password" value="123456" />

<property name="driverClassName" value="com.mysql.cj.jdbc.Driver" />

</bean>

<!--JDBCTemplate对象-->

<bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">

<!--注入DataSource-->

<property name="dataSource" ref="dataSource"></property>

</bean>

4.在dao层创建多钱和少钱的方法，在service层创建转账的方法

@Override

public void reduceMoney() {

String sql = "update t\_account set money = money - ? where id = 1";

jdbcTemplate.update(sql,100);

}

@Override

public void addMoney() {

String sql = "update t\_account set money = money + ? where id = 2";

jdbcTemplate.update(sql,100);

}

//Service层

//转账

public void addAndReduce(){

accountDao.reduceMoney();

accountDao.addMoney();

}

//测试

@Test

public void test1(){

ApplicationContext context = new ClassPathXmlApplicationContext("bean1.xml");

AccountService accountService = context.getBean("accountService", AccountService.class);

accountService.addAndReduce();

}

5.模拟异常场景，引入事务操作

问题引入：

如下所示异常会导致A减少钱，而B没有增加钱



解决思路：利用事务的一致性来解决，使用try-catch捕获异常，一旦捕获到异常就回滚事务，中断操作

public void addAndReduce(){

try{

//1.开启事务

//2.业务操作

accountDao.reduceMoney();

//模拟异常

int b = 10/0;

accountDao.addMoney();

//3.没有问题,提交事务

}catch (Exception e){

//4.捕获到异常，事务回滚

}

}

**三.事务操作**

**3.1主要概念**

1.事务添加到JavaEE三层结构的service层（业务逻辑层）

2.在Spring中进行事务管理操作有编程式事务管理和声明式事务管理

3.声明式事务管理具体方式

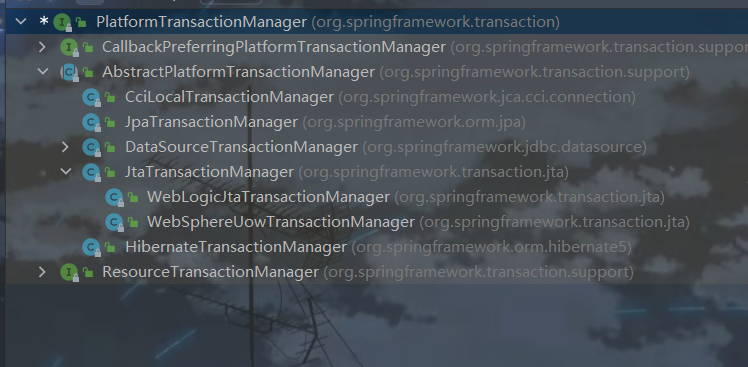
（1）使用注解

（2）使用配置文件

4.在spring中进行声明式事务管理，底层使用AOP原理

5.Spring事务管理API

（1）提供一个接口，代表事务管理器，它针对不同的框架有不同的实现类



**3.2具体操作**

1.在配置文件中配置文件管理器

<!--配置事务管理器-->

<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

<!--注入数据源-->

<property name="dataSource" ref="dataSource"></property>

</bean>

2.添加命名空间并开启事务注解

xmlns:tx="<http://www.springframework.org/schema/tx>"

<!--开启事务注解[-->](http://www.springframework.org/schema/tx)

<tx:annota[tion-driven transaction-manager="transac](http://www.springframework.org/schema/tx)tionManager"></tx:annotation-driven>

3.在service类上面添加事务注解

（1）@Transactional,此注解既可以添加到类上方，也可添加到方法上方

（2）添加到类上，该类所有方法都可开启事务

（3）添加到方法上，只有该方法可以开启事务

package Spring5.service;

import Spring5.dao.AccountDao;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Component;

import org.springframework.transaction.PlatformTransactionManager;

import org.springframework.transaction.annotation.Transactional;

/\*\*

\* @author xh

\* @date 2022/10/17

\* @apiNote

\*/

@Component

@Transactional

public class AccountService {

//注入dao

@Autowired

private AccountDao accountDao;

//转账

public void addAndReduce(){

// try{

//

// //1.开启事务

//

// //2.业务操作

// accountDao.reduceMoney();

//

// //模拟异常

// int b = 10/0;

//

// accountDao.addMoney();

//

// //3.没有问题,提交事务

// }catch (Exception e){

// //4.捕获到异常，事务回滚

// }

accountDao.reduceMoney();

//模拟异常

int b = 10/0;

accountDao.addMoney();

}

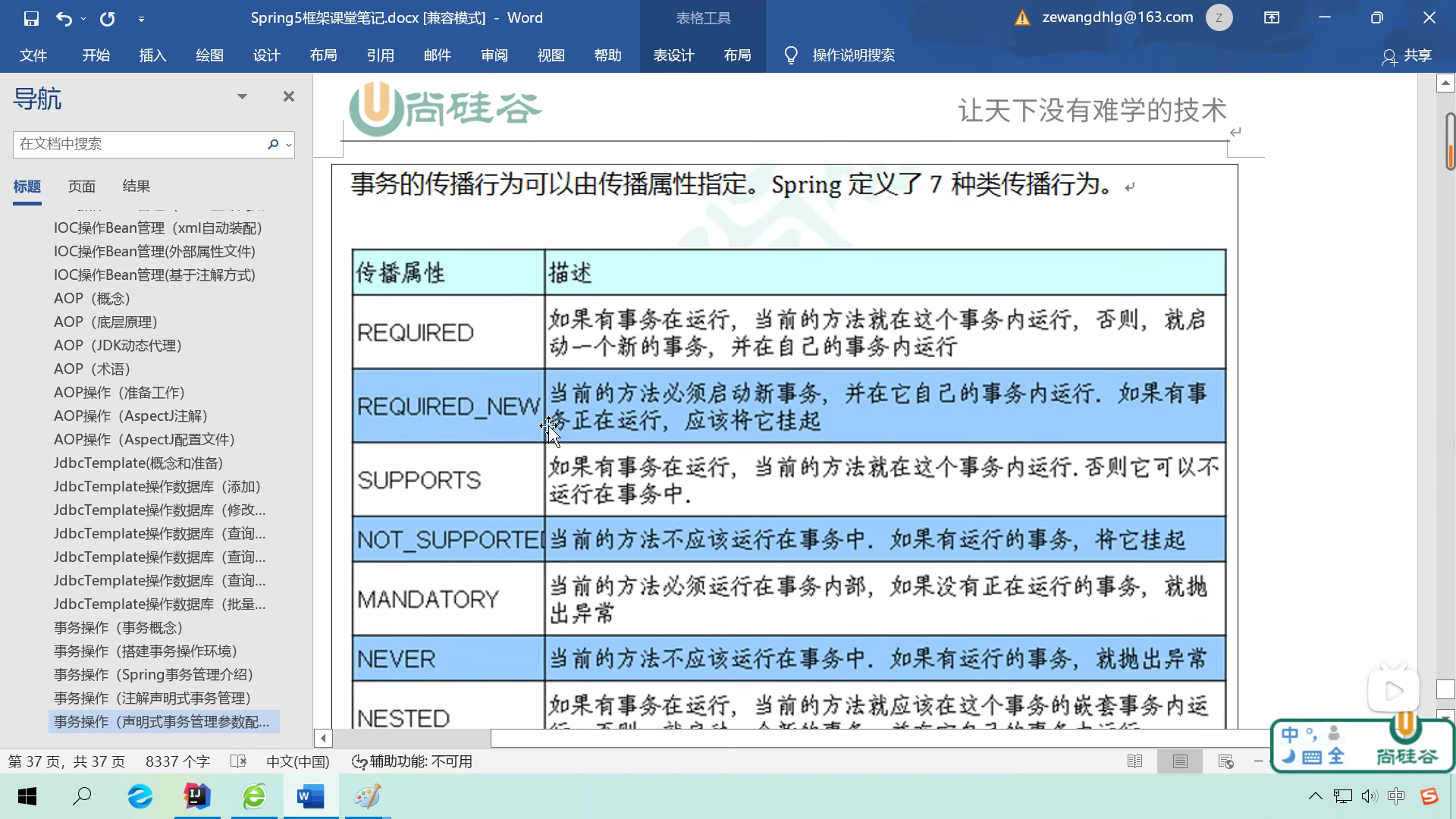
}

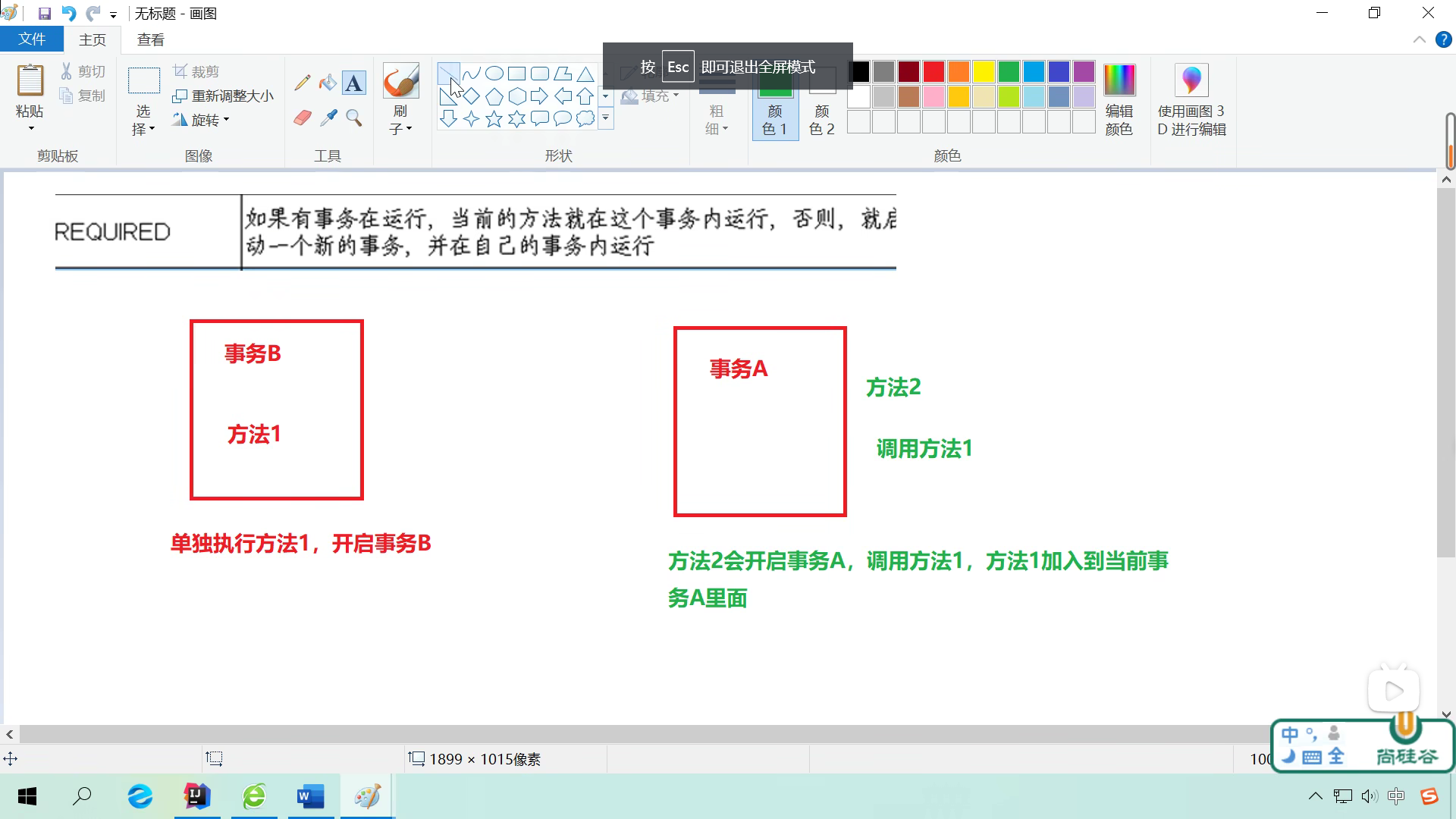
**3.3事务的主要参数**

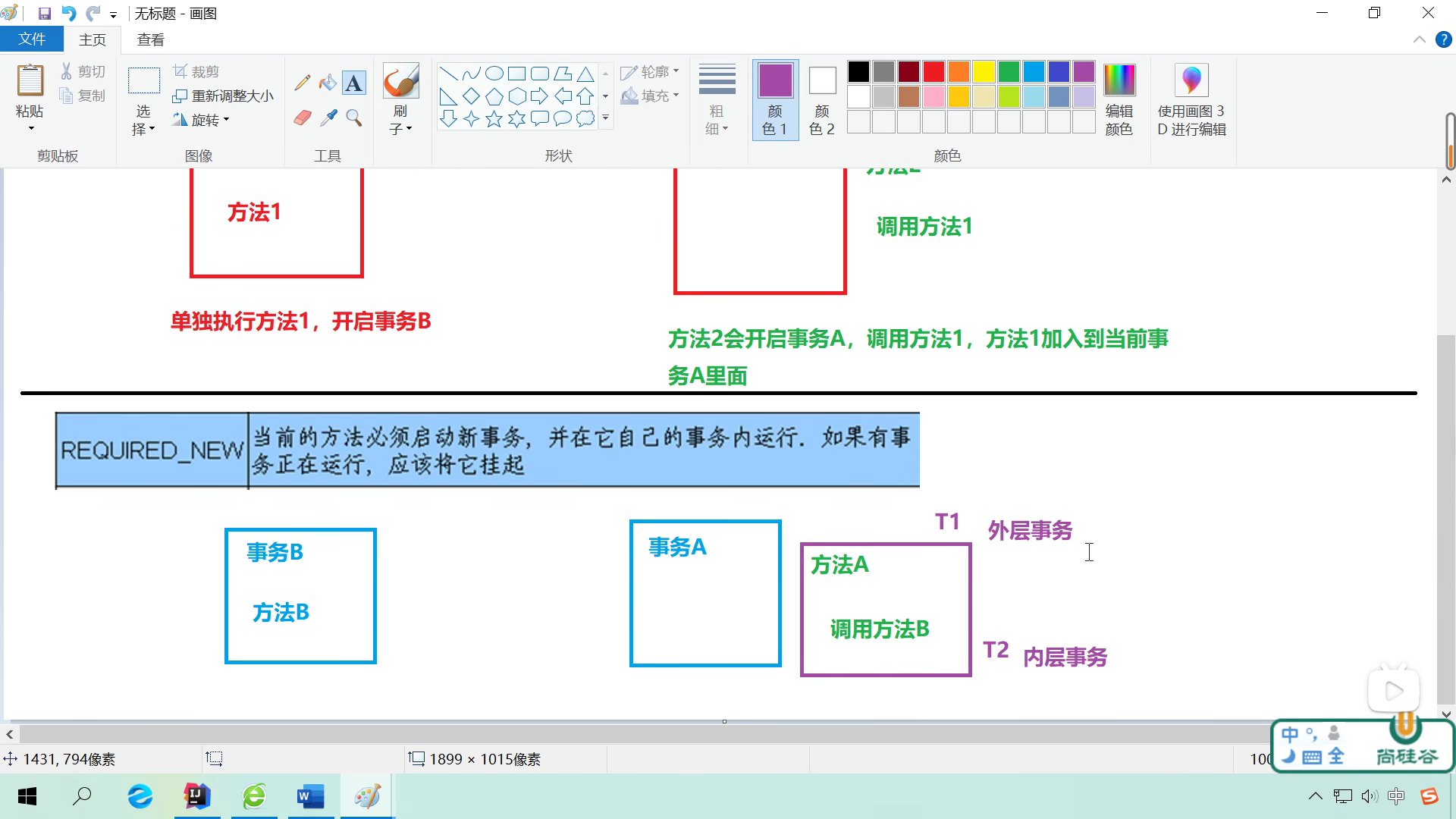
**1.propagation：事务的传播行为**

事务传播行为是指多个拥有事务的方法在嵌套调用时的事务控制方式

当一个事务方法被另一个事务方法调用时，该事务方法如何进行？







**2.ioslation：事务的隔离级别**

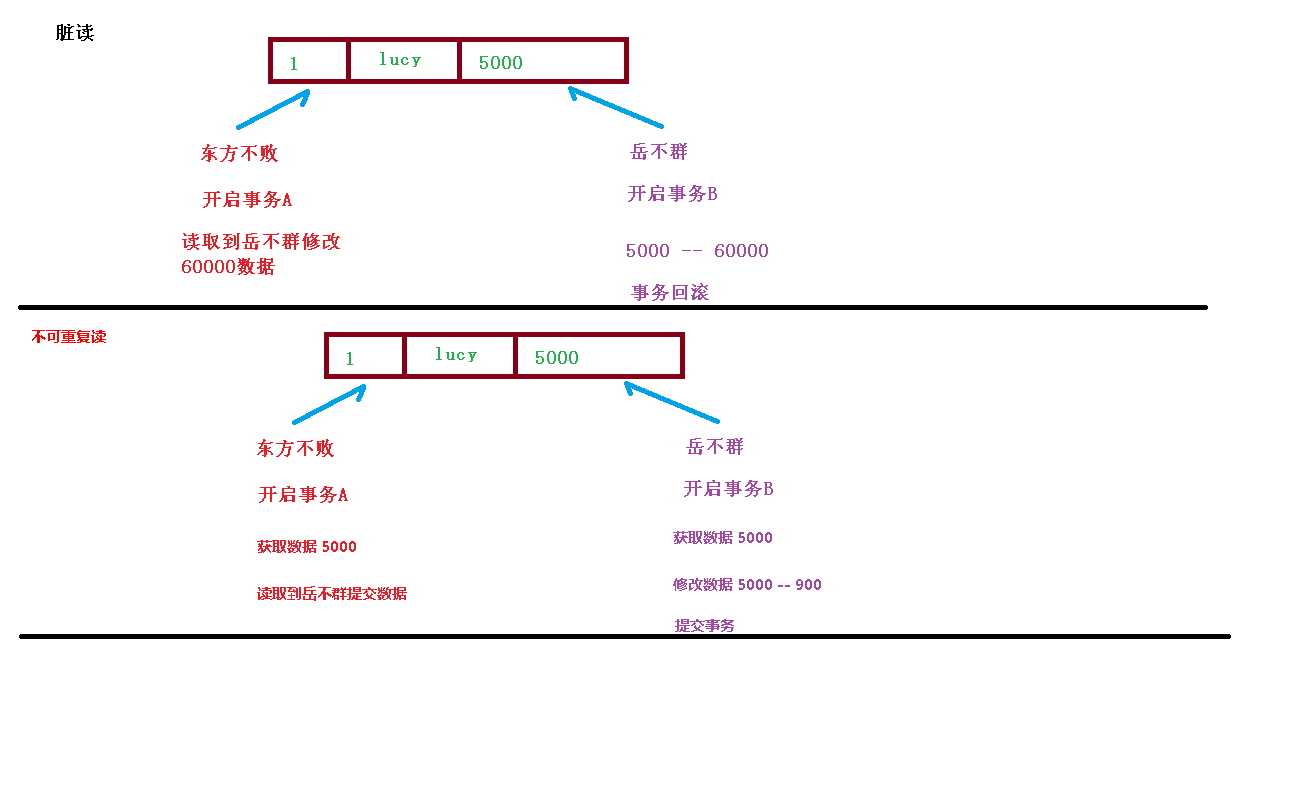
（1）事务有特性称为隔离性，多事务操作之间不会产生影响，不考虑隔离性会产生许多问题

（2）三个读问题：脏读、不可重复读、虚读（幻读）

（3）脏读：一个未提交的事务读取到另一个未提交事务的数据

（4）不可重复读：一个未提交事务读取到另一个提交事务修改的数据

（5）幻读：一个未提交事务读取到另一个提交事务添加的数据



（6）通过设置事务的隔离级别，解决读问题



**3.timeout：超时时间**

（1）事务需要在一定时间内进行提交，否则进行回滚

（2）默认值-1，设置时间以秒为单位进行计算

**4.readOnly：是否只读**

（1）读：查询操作 写：包含增删改操作

（2）readOnly默认值false，表示可以查询，可以添加修改删除操作

（3）设置readOnly为True，表示只能查询

**5.rollbackFor：回滚**

设置出现哪些异常进行事务回滚

**6.notRollbackFor：不回滚**

设置出现哪些异常不进行事务回滚

**四.Spring声明式事务管理（xml配置文件）**

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/b[eans"](http://www.springframework.org/schema/beans)

xmlns:xsi="http://www.w3.org/2001/XMLSchema-in[stance"](http://www.springframework.org/schema/beans)

xmlns:context="http://www.springframework.org/[schema/context"](http://www.springframework.org/schema/beans)

xmlns:aop="http://www.springframework.org/sche[ma/aop"](http://www.springframework.org/schema/beans)

xmlns:tx="http://www.springframework.org/schem[a/tx"](http://www.springframework.org/schema/beans)

xsi:schemaLocation="http://www.springframework[.org/schema/beans http://www.springframewor](http://www.springframework.org/schema/beans)k.org/schema/beans/s[pring-beans.xsd](http://www.w3.org/2001/XMLSchema-instance)

http://www.springframework[.org/schema/aop http://www.springframework.](http://www.springframework.org/schema/beans)org/schema/aop/sprin[g-aop.xsd](http://www.w3.org/2001/XMLSchema-instance)

http://www.springframework[.org/schema/tx http://www.springframework.o](http://www.springframework.org/schema/beans)rg/schema/tx/spring-[tx.xsd](http://www.w3.org/2001/XMLSchema-instance)

http://www.springframework[.org/schema/context http://www.springframew](http://www.springframework.org/schema/beans)ork.org/schema/conte[xt/spring-context.xsd">](http://www.w3.org/2001/XMLSchema-instance)

<!--开启组件扫描-->

<context:component-scan base-package="Spring5.dao[,Spring5.service"></context:component-scan>](http://www.springframework.org/schema/beans)

<!-- 数据库连接池 -->

<bean id="dataSource" class="com.alibaba.druid.po[ol.DruidDataSource"](http://www.springframework.org/schema/beans)

destroy-method="close">

<property name="url" value="jdbc:mysql:///use[r\_db" />](http://www.springframework.org/schema/beans)

<property name="username" value="root" />

<property name="password" value="123456" />

<property name="driverClassName" value="com.m[ysql.cj.jdbc.Driver" />](http://www.springframework.org/schema/beans)

</bean>

<!--JDBCTemplate对象-->

<bean id="jdbcTemplate" class="org.springframewor[k.jdbc.core.JdbcTemplate">](http://www.springframework.org/schema/beans)

<!--注入DataSource-->

<property name="dataSource" ref="dataSource">[</property>](http://www.springframework.org/schema/beans)

</bean>

<!--1.配置事务管理器-->

<bean id="transactionManager" class="org.springfr[amework.jdbc.datasource.DataSourceTransacti](http://www.springframework.org/schema/beans)onManager">

<!--注入数据源-->

<property name="dataSource" ref="dataSource">[</property>](http://www.springframework.org/schema/beans)

</bean>

<!--开启事务注解-->

<tx:annotation-driven transaction-manager="transa[ctionManager"></tx:annotation-driven>](http://www.springframework.org/schema/beans)

<!--2.配置通知-->

<tx:advice id="txadvice">

<tx:attributes>

<!--指定哪种规则的方法上开启事务-->

<tx:method name="addAndReduce" propagatio[n="REQUIRED"/>](http://www.springframework.org/schema/beans)

</tx:attributes>

</tx:advice>

<!--3.配置切入点和切面-->

<aop:config>

<!--配置切入点-->

<aop:pointcut id="pct" expression="execution([\* Spring5.service.AccountService.\*(..))"/>](http://www.springframework.org/schema/beans)

<!--配置切面-->

<aop:advisor advice-ref="txadvice" pointcut-r[ef="pct"></aop:advisor>](http://www.springframework.org/schema/beans)

</aop:config>

</beans>

**五.Spring声明式事务管理（纯注解）**

**1.编写配置类**

package Spring5.Config;

import com.alibaba.druid.pool.DruidDataSource;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

import org.springframework.jdbc.core.JdbcTemplate;

import org.springframework.jdbc.datasource.DataSourceTransactionManager;

import org.springframework.transaction.annotation.EnableTransactionManagement;

/\*\*

\* @author xh

\* @date 2022/10/18

\* @apiNote

\*/

@Configuration //配置类

@ComponentScan(basePackages = "Spring5") //组件扫描

@EnableTransactionManagement //开启事务

public class TxConfig {

//创建数据库连接池

@Bean

public DruidDataSource getDruidDataSource(){

String url = "jdbc:mysql:///user\_db";

String username = "root";

String password = "123456";

String driverClassName = "com.mysql.cj.jdbc.Driver";

DruidDataSource druidDataSource = new DruidDataSource();

druidDataSource.setUrl(url);

druidDataSource.setUsername(username);

druidDataSource.setPassword(password);

druidDataSource.setDriverClassName(driverClassName);

return druidDataSource;

}

//创建jdbcTemplate对象

@Bean

public JdbcTemplate getJdbcTemplate(DruidDataSource druidDataSource){

JdbcTemplate jdbcTemplate = new JdbcTemplate();

jdbcTemplate.setDataSource(druidDataSource);

return jdbcTemplate;

}

//配置事务管理器

@Bean

public DataSourceTransactionManager getDataSourceTransactionManager(DruidDataSource druidDataSource){

DataSourceTransactionManager dataSourceTransactionManager = new DataSourceTransactionManager();

dataSourceTransactionManager.setDataSource(druidDataSource);

return dataSourceTransactionManager;

}

}

**2.编写测试类**

@Test

public void test3(){

ApplicationContext context = new AnnotationConfigApplicationContext("Spring5.Config。");

AccountService accountService = context.getBean("accountService", AccountService.class);

accountService.addAndReduce();

}