

Spring5框架重点之事务管理

Spring官网: <https://spring.io/>

SpringFramework的核心

IoC[DI]

AOP 面向切面编程

1.JdbcTemplate

Jdbc模板, 基于Jdbc封装的组件

依赖坐标

```
<dependencies>
  <!-- https://mvnrepository.com/artifact/org.springframework/spring-context -->
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-orm</artifactId>
    <version>5.1.17.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.1.17.RELEASE</version>
  </dependency>
  <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.16</version>
  </dependency>

  <!-- https://mvnrepository.com/artifact/org.aspectj/aspectjweaver -->
  <dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
    <version>1.9.1</version>
  </dependency>
</dependencies>
```

创建对应的业务类

```
package com.gupaoedu.pojo;
```

```

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 */
public class User {

    private Integer id;

    private String name;

    private Integer age;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Integer getAge() {
        return age;
    }

    public void setAge(Integer age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return "User{" +
            "id=" + id +
            ", name='" + name + '\'' +
            ", age=" + age +
            '}';
    }
}

```

dao

```

package com.gupaoedu.dao;

import com.gupaoedu.pojo.User;

import java.util.List;

```



```

        user.setName(resultSet.getString("name"));
        user.setAge(resultSet.getInt("age"));
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return user;
}
});
return users;
}
}

```

service

```

package com.gupaoedu.service;

import com.gupaoedu.pojo.User;

import java.util.List;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 */
public interface IUserService {

    public Integer addUser(User user);

    public List<User> query();
}

```

```

package com.gupaoedu.service.impl;

import com.gupaoedu.dao.IUserDao;
import com.gupaoedu.pojo.User;
import com.gupaoedu.service.IUserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 */
@Service
public class UserServiceImpl implements IUserService {

    @Autowired

```

```

        private IUserDao dao;

        public Integer addUser(User user) {
            return dao.addUser(user);
        }

        public List<User> query() {
            return dao.query();
        }
    }
}

```

Java配置及测试

```

package com.gupaoedu;

import com.gupaoedu.pojo.User;
import com.gupaoedu.service.IUserService;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.*;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.datasource.DriverManagerDataSource;

import javax.sql.DataSource;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 */
@Configuration
@ComponentScan
public class JavaConfig {

    @Bean
    public DataSource dataSource(){
        DriverManagerDataSource dataSource = new DriverManagerDataSource();
        dataSource.setDriverClassName("com.mysql.cj.jdbc.Driver");
        dataSource.setUrl("jdbc:mysql://localhost:3306/gp?
serverTimezone=UTC&characterEncoding=utf8&useUnicode=true&useSSL=false");
        dataSource.setUsername("root");
        dataSource.setPassword("123456");
        return dataSource;
    }

    @Bean
    @DependsOn("dataSource")
    public JdbcTemplate template(DataSource dataSource){
        return new JdbcTemplate(dataSource);
    }

    public static void main(String[] args) {
        ApplicationContext ac = new
        AnnotationConfigApplicationContext(JavaConfig.class);
    }
}

```

```

        IUserService bean = ac.getBean(IUserService.class);
        /*User user = new User();
        user.setName("Mic");
        user.setAge(18);
        bean.addUser(user);*/
        System.out.println(bean.query());
    }
}

```

2.事务管理

事务仅与数据库有关

事务必须满足ACID原则：

- 原子性 (atomicity)
- 一致性 (consistency)
- 隔离性 (isolation)
- 持久性 (durability)

没有使用事务造成的数据安全问题案例

下订单：

订单主表： 订单总金额， 用户， 下单地址， 时间 ...

订单详情表： 购买的每样商品都有一条记录

出现的与我们期望不相符的情况~

```

public void dml(User user,String userName,String password){

    try {
        Class.forName(DRIVERNAME);
        conn = DriverManager.getConnection(URL, USERNAME, PASSWORD);
        // 默认是自动提交的，事务管理我们需要关闭掉自动提交
        conn.setAutoCommit(false);
        sql = "insert into users(name,age)values(?,?)";
        ps = conn.prepareStatement(sql);
        ps.setString(1,user.getName());
        ps.setInt(2,user.getAge());
        ps.executeUpdate();
        sql = "insert into t_user(username,password)values(?,?)";
        ps = conn.prepareStatement(sql);
        ps.setString(1,userName);
        ps.setString(2,password);
        ps.executeUpdate();
        conn.commit();
    } catch (Exception e) {

```

```

        try {
            conn.rollback();
        } catch (SQLException e1) {
            e1.printStackTrace();
        }
        e.printStackTrace();
    }finally {

        if(ps != null){
            try {
                ps.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if(conn != null){
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
}

```

基于Jdk动态代理实现事务管理

公共代码管理

```

package com.gupaoedu.utils;

import java.sql.Connection;
import java.sql.DriverManager;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 */
public class DbUtils {
    private static final String DRIVENAME = "com.mysql.cj.jdbc.Driver";
    private static final String URL="jdbc:mysql://localhost:3306/gp?serverTimezone=UTC&characterEncoding=utf8&useUnicode=true&useSSL=false";
    private static final String USERNAME = "root";
    private static final String PASSWORD = "123456";

    private static Connection conn = null;

    public static Connection getConnection(){
        if(conn == null){
            try {
                Class.forName(DRIVENAME);
                conn = DriverManager.getConnection(URL,USERNAME,PASSWORD);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}

```

```

        }
    }
    return conn;
}
}

```

Dao代码抽取

```

package com.gupaoedu.dao.impl;

import com.gupaoedu.dao.IUserDao;
import com.gupaoedu.pojo.User;
import com.gupaoedu.utils.DbUtils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.RowMapper;
import org.springframework.stereotype.Repository;

import java.sql.*;
import java.util.List;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 */
@Repository
public class UserDaoImpl implements IUserDao {

    @Autowired
    private JdbcTemplate template;
    private String sql;
    Connection conn = null;
    PreparedStatement ps = null;

    public Integer addUser(User user) throws Exception{
        /*sql = "insert into users(name,age)values(?,?)";
        return template.update(sql,user.getName(),user.getAge());*/
        int count = 0;

        sql = "insert into users(name,age)values(?,?)";
        ps = DbUtils.getConnection().prepareStatement(sql);
        ps.setString(1,user.getName());
        ps.setInt(2,user.getAge());
        count = ps.executeUpdate();

        return count ;
    }
}

```



```

public Integer addUser1(String userName, String password) throws Exception {
    /*sql = "insert into t_user(username,password)values(?,?)";
    return template.update(sql,userName,password);*/
    int count = 0;
    sql = "insert into t_user(username,password)values(?,?)";
    ps = DbUtils.getConnection().prepareStatement(sql);
    ps.setString(1,userName);
    ps.setString(2,password);
    count = ps.executeUpdate();
    return count;
}

public List<User> query() {
    sql = "select * from users";
    List<User> users = template.query(sql, new RowMapper<User>() {
        /**
         * 我们自己手动将查询出来的结果集和Java对象中的属性做映射
         * @param resultSet
         * @param i
         * @return
         */
        public User mapRow(ResultSet resultSet, int i) {
            User user = new User();
            try {
                user.setId(resultSet.getInt("id"));
                user.setName(resultSet.getString("name"));
                user.setAge(resultSet.getInt("age"));
            } catch (SQLException e) {
                e.printStackTrace();
            }
            return user;
        }
    });
    return users;
}
}

```

业务实现管理

```

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 */
@Service
public class UserServiceImpl implements IUserService {

    //@Autowired
    private IUserDao dao = new UserDaoImpl();

    public Integer addUser(User user) throws Exception {
        return dao.addUser(user);
    }
}

```

```

    public List<User> query() {
        return dao.query();
    }

    public Integer addUser1(String userName, String password) throws Exception {
        return dao.addUser1(userName,password);
    }

    public Integer business(User user, String userName, String password) throws
Exception{
        dao.addUser(user);
        dao.addUser1(userName,password);
        return null;
    }
}

```

代理模式实现

```

package com.gupaoedu;

import com.gupaoedu.pojo.User;
import com.gupaoedu.service.IUserService;
import com.gupaoedu.service.impl.UserServiceImpl;
import com.gupaoedu.utils.DbUtils;

import java.lang.reflect.InvocationHandler;
import java.lang.reflect.Method;
import java.lang.reflect.Proxy;
import java.sql.Connection;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 */
public class TestMain {
    public static void main(String[] args) {
        /*ApplicationContext ac = new
AnnotationConfigApplicationContext(JavaConfig.class);
        IUserService bean = ac.getBean(IUserService.class);
        User user = new User();
        user.setName("TOM");
        user.setAge(18);
        bean.addUser(user);
        //System.out.println(bean.query());
        bean.addUser1("admin111","123");*/

        // 目标对象
        final IUserService target = new UserServiceImpl();
        // 获取代理对象
        IUserService proxy = (IUserService)
Proxy.newProxyInstance(JavaConfig.class.getClassLoader(),
target.getClass().getInterfaces(), new InvocationHandler() {
            public Object invoke(Object proxy, Method method, Object[] args)
throws Throwable {

```

行

```
        Connection conn = null;
        try{
            conn = DbUtils.getConnection();
            conn.setAutoCommit(false);
            method.invoke(target,args[0],args[1],args[2]); // 目标对象的执

            System.out.println("提交成功...");
            conn.commit();
        }catch (Exception e){
            System.out.println("执行失败，数据回滚");
            conn.rollback();
        }finally {
            conn.close();
        }

        return null;
    }
});
User user = new User();
user.setName("波波2");
user.setAge(18);

    try {
        proxy.business(user,"root2","11111");
    } catch (Exception e) {
        e.printStackTrace();
    }
}
}
```

基于AspectJ来自定义实现事务的处理

定义自定义注解

```
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 */
@Retention(RetentionPolicy.RUNTIME)
public @interface GupaoTx {
}
```

在需要AOP支持的方法上添加该注解

```

@GupaoTx
public Integer business(User user, String userName, String password) throws
Exception{
    dao.addUser(user);
    dao.addUser1(userName,password);
    return null;
}

```

自定义切面类

```

package com.gupaoedu.aspect;

import com.gupaoedu.utils.DbUtils;
import org.aspectj.lang.annotation.*;
import org.springframework.context.annotation.Bean;
import org.springframework.stereotype.Component;

import java.sql.Connection;
import java.sql.SQLException;

/**
 * 让每一个人的职业生涯不留遗憾
 *
 * @author 波波老师【咕泡学院】
 */
@Aspect
@Component
public class GupaoAspect {

    /**
     * 事务处理的前置方法
     */
    @Before(value = "@annotation(com.gupaoedu.annotation.GupaoTx)")
    public void beforeTx() throws SQLException {
        Connection conn = DbUtils.getConnection();
        conn.setAutoCommit(false);
    }

    /**
     * 后置通知
     * @throws SQLException
     */
    @AfterReturning("@annotation(com.gupaoedu.annotation.GupaoTx)")
    public void afterReturningTx() throws SQLException {
        Connection conn = DbUtils.getConnection();
        System.out.println("提交成功...");
        conn.commit();
    }

    /**
     * 最终通知
     * @throws SQLException
     */
    @After("@annotation(com.gupaoedu.annotation.GupaoTx)")
    public void afterTx() throws SQLException {

```

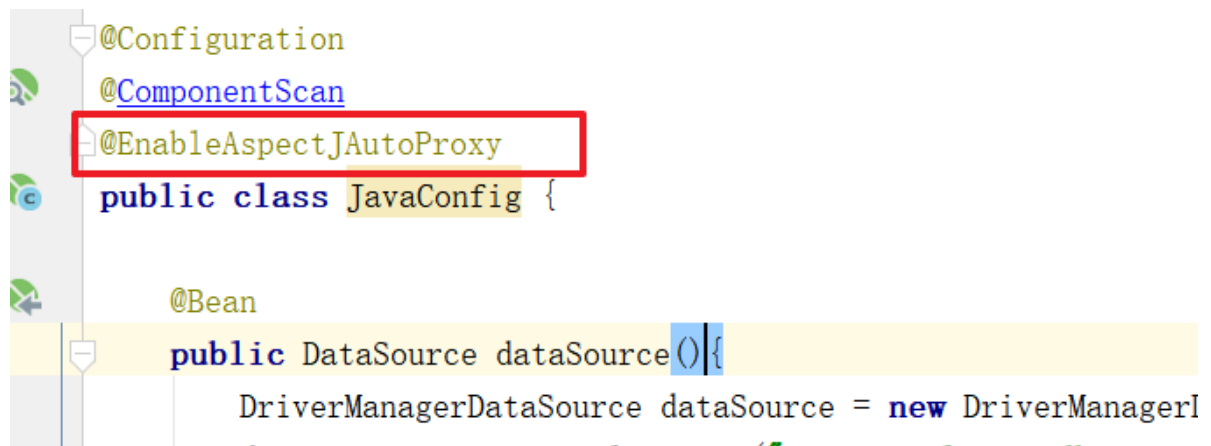
```

        Connection conn = DbUtils.getConnection();
        //conn.close();
    }

    @AfterThrowing("@annotation(com.gupaoedu.annotation.GupaoTx)")
    public void afterThrowing() throws SQLException {
        Connection conn = DbUtils.getConnection();
        System.out.println("失败回滚....");
        conn.rollback();
    }
}

```

放开AspectJ的支持



```

@Configuration
@ComponentScan
@EnableAspectJAutoProxy
public class JavaConfig {

    @Bean
    public DataSource dataSource() {
        DriverManagerDataSource dataSource = new DriverManagerI
    }
}

```

测试代码

```

public class JavaConfigMain {

    public static void main(String[] args) throws Exception {
        ApplicationContext ac = new
        AnnotationConfigApplicationContext(JavaConfig.class);
        IUserService bean = ac.getBean(IUserService.class);
        User user = new User();
        user.setName("James2");
        user.setAge(22);
        bean.business(user, "abc2", "123321");
    }
}

```

Spring中事务管理

1.基于xml文件的形式

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       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-4.3.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-4.3.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-4.1.xsd">

    <context:component-scan base-package="com.gupaoedu.*" />

    <context:property-placeholder location="db.properties"/>
    <!-- 配置数据源 -->
    <bean class="org.springframework.jdbc.datasource.DriverManagerDataSource"
id="dataSource">
        <property name="url" value="${jdbc_url}"/>
        <property name="driverClassName" value="${jdbc_driver}"/>
        <property name="username" value="${jdbc_username}"/>
        <property name="password" value="${jdbc_password}"/>
    </bean>

    <!-- 配置JdbcTemplate -->
    <bean class="org.springframework.jdbc.core.JdbcTemplate" >
        <constructor-arg name="dataSource" ref="dataSource"/>
    </bean>

    <bean id="txManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource"/>
    </bean>

    <tx:advice id="txAdvice" transaction-manager="txManager">
        <!-- the transactional semantics... -->
        <tx:attributes>
            <!-- all methods starting with 'get' are read-only -->
            <tx:method name="bus*" propagation="REQUIRED"/>
            <!-- other methods use the default transaction settings (see below) -->
            <tx:method name="*" />
        </tx:attributes>
    </tx:advice>

    <aop:config>
        <aop:pointcut id="tx" expression="execution(* com.gupaoedu.service...*(..))"/>
        <aop:advisor advice-ref="txAdvice" pointcut-ref="tx"/>
    </aop:config>
```

```
</beans>
```

2. 基于注解的方式

需要放开对注解使用的支持

```
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:p="http://www.springframework.org/schema/p"
        xmlns:context="http://www.springframework.org/schema/context"
        xmlns:aop="http://www.springframework.org/schema/aop"
        xmlns:tx="http://www.springframework.org/schema/tx"
        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-4.3.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-4.3.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-4.1.xsd">

    <context:component-scan base-package="com.gupaoedu.*" />

    <context:property-placeholder location="db.properties"/>
    <!-- 配置数据源 -->
    <bean class="org.springframework.jdbc.datasource.DriverManagerDataSource"
        id="dataSource">
        <property name="url" value="${jdbc_url}"/>
        <property name="driverClassName" value="${jdbc_driver}"/>
        <property name="username" value="${jdbc_username}"/>
        <property name="password" value="${jdbc_password}"/>
    </bean>

    <!-- 配置JdbcTemplate -->
    <bean class="org.springframework.jdbc.core.JdbcTemplate" >
        <constructor-arg name="dataSource" ref="dataSource"/>
    </bean>

    <bean id="txManager"
        class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource"/>
    </bean>

    <!-- 开启事务的注解支持-->
    <tx:annotation-driven transaction-manager="txManager"/>
</beans>
```

在需要使用注解的方法获取类的头部添加对应的注解 @Transactional

```

@Transactional
public void business() throws Exception{
    User user = new User("lisi",24);
    addUser(user);
    user.setId(13);
    user.setUserName("bbb");
    updateUser(user);
}

```

3.事务的传播行为

Spring中的7个事务传播行为:

事务行为	说明
PROPAGATION_REQUIRED	支持当前事务，假设当前没有事务。就新建一个事务
PROPAGATION_SUPPORTS	支持当前事务，假设当前没有事务，就以非事务方式运行
PROPAGATION_MANDATORY	支持当前事务，假设当前没有事务，就抛出异常
PROPAGATION_REQUIRES_NEW	新建事务，假设当前存在事务。把当前事务挂起
PROPAGATION_NOT_SUPPORTED	以非事务方式运行操作。假设当前存在事务，就把当前事务挂起
PROPAGATION_NEVER	以非事务方式运行，假设当前存在事务，则抛出异常
PROPAGATION_NESTED	如果当前存在事务，则在嵌套事务内执行。如果当前没有事务，则执行与PROPAGATION_REQUIRED类似的操作。

举例说明

ServiceA

```

ServiceA {
    void methodA() {
        ServiceB.methodB();
    }
}

```

ServiceB

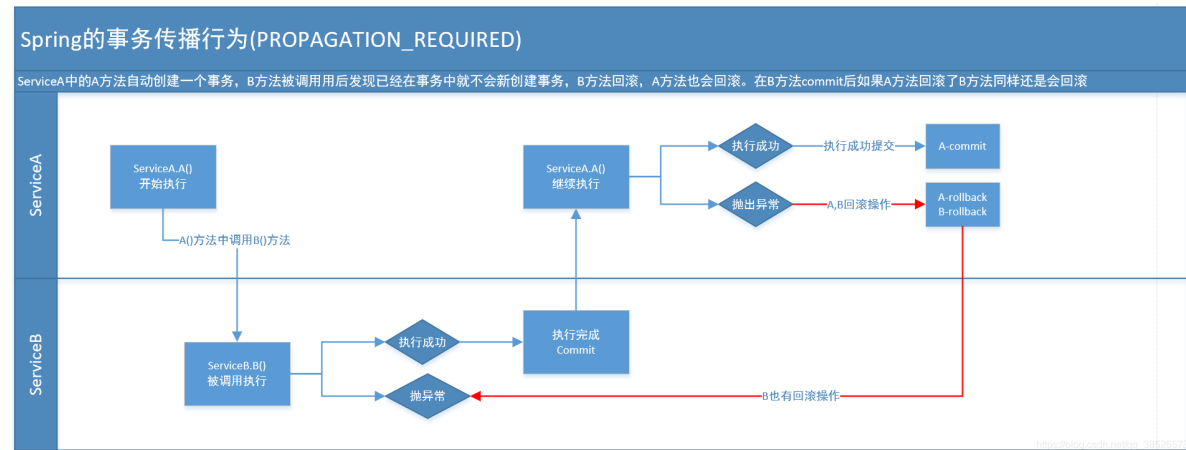
```

ServiceB {
    void methodB() {
    }
}

```


1.PROPGATION_REQUIRED

假如当前正要运行的事务不在另外一个事务里，那么就起一个新的事务 比方说，ServiceB.methodB的事务级别定义PROPAGATION_REQUIRED, 那么因为执行ServiceA.methodA的时候，ServiceA.methodA已经起了事务。这时调用ServiceB.methodB，ServiceB.methodB看到自己已经执行在ServiceA.methodA的事务内部。就不再起新的事务。而假如ServiceA.methodA执行的时候发现自己没有在事务中，他就会为自己分配一个事务。这样，在ServiceA.methodA或者在ServiceB.methodB内的不论什么地方出现异常。事务都会被回滚。即使ServiceB.methodB的事务已经被提交，可是ServiceA.methodA在接下来fail要回滚，ServiceB.methodB也要回滚



2.PROPGATION_SUPPORTS

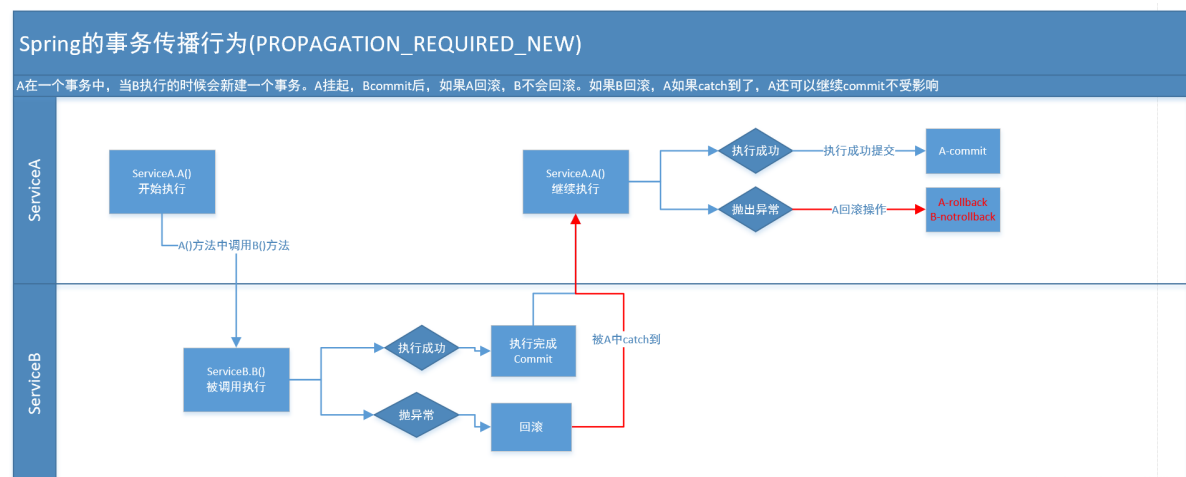
假设当前在事务中。即以事务的形式执行。假设当前不在一个事务中，那么就以非事务的形式执行

3PROPAGATION_MANDATORY

必须在一个事务中执行。也就是说，他仅仅能被一个父事务调用。否则，他就要抛出异常

4.PROPGATION_REQUIRES_NEW

这个就比较绕口了。比方我们设计ServiceA.methodA的事务级别为PROPAGATION_REQUIRED，ServiceB.methodB的事务级别为PROPAGATION_REQUIRES_NEW。那么当运行到ServiceB.methodB的时候，ServiceA.methodA所在的事务就会挂起。ServiceB.methodB会起一个新的事务。等待ServiceB.methodB的事务完毕以后，他才继续运行。他与PROPAGATION_REQUIRED 的事务差别在于事务的回滚程度了。由于ServiceB.methodB是新起一个事务，那么就是存在两个不同的事务。假设ServiceB.methodB已经提交，那么ServiceA.methodA失败回滚。ServiceB.methodB是不会回滚的。假设ServiceB.methodB失败回滚，假设他抛出的异常被ServiceA.methodA捕获，ServiceA.methodA事务仍然可能提交。



5.PROPAGATION_NOT_SUPPORTED

当前不支持事务。比方ServiceA.methodA的事务级别是PROPAGATION_REQUIRED。而ServiceB.methodB的事务级别是PROPAGATION_NOT_SUPPORTED，那么当执行到ServiceB.methodB时。ServiceA.methodA的事务挂起。而他以非事务的状态执行完，再继续ServiceA.methodA的事务。

6.PROPAGATION_NEVER

不能在事务中执行。

如果ServiceA.methodA的事务级别是PROPAGATION_REQUIRED。而ServiceB.methodB的事务级别是PROPAGATION_NEVER，那么ServiceB.methodB就要抛出异常了。

7.PROPAGATION_NESTED

如果当前存在事务，则在嵌套事务内执行。如果当前没有事务，则执行与PROPAGATION_REQUIRED类似的操作。

##

```
<tx:advice id="txAdvice" transaction-manager="txManager">
  <tx:attributes>
    <!--设置所有匹配的方法，然后设置传播级别和事务隔离-->
    <tx:method name="save*" propagation="REQUIRED" />
    <tx:method name="add*" propagation="REQUIRED" />
    <tx:method name="create*" propagation="REQUIRED" />
    <tx:method name="insert*" propagation="REQUIRED" />
    <tx:method name="update*" propagation="REQUIRED" />
    <tx:method name="merge*" propagation="REQUIRED" />
    <tx:method name="del*" propagation="REQUIRED" />
    <tx:method name="remove*" propagation="REQUIRED" />
    <tx:method name="put*" propagation="REQUIRED" />
    <tx:method name="get*" propagation="SUPPORTS" read-only="true" />
    <tx:method name="count*" propagation="SUPPORTS" read-only="true" />
    <tx:method name="find*" propagation="SUPPORTS" read-only="true" />
    <tx:method name="list*" propagation="SUPPORTS" read-only="true" />
    <tx:method name="*" propagation="SUPPORTS" read-only="true" />
  </tx:attributes>
</tx:advice>
```

注解的方式

```
<!--开启注解的方式-->
<tx:annotation-driven transaction-manager="transactionManager" />
```

@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)

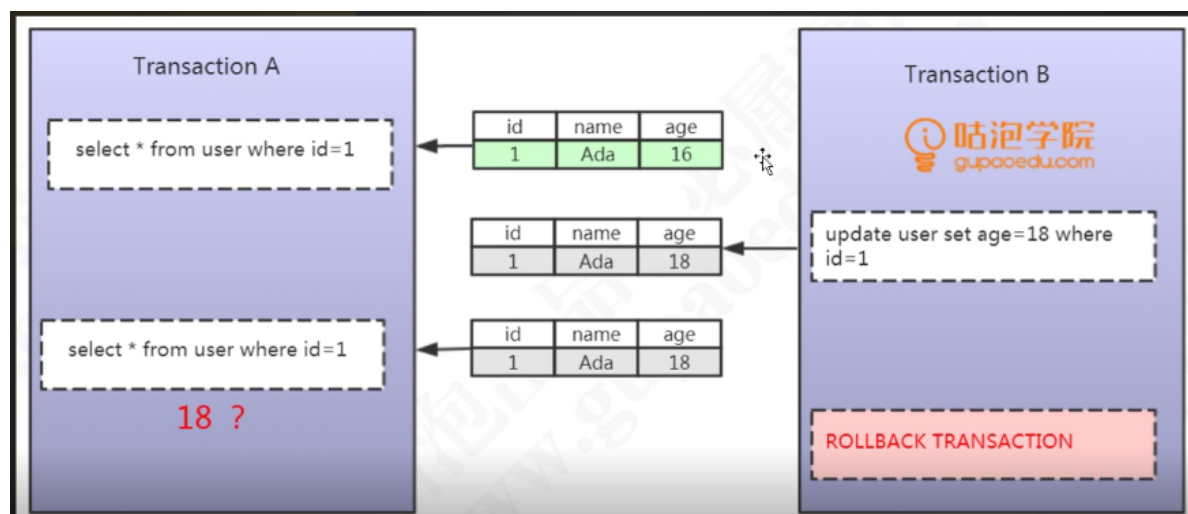
如果其他bean调用这个方法,在其他bean中声明事务,那就用事务.如果其他bean没有声明事务,那就不用事务.

4.Spring支持的隔离级别

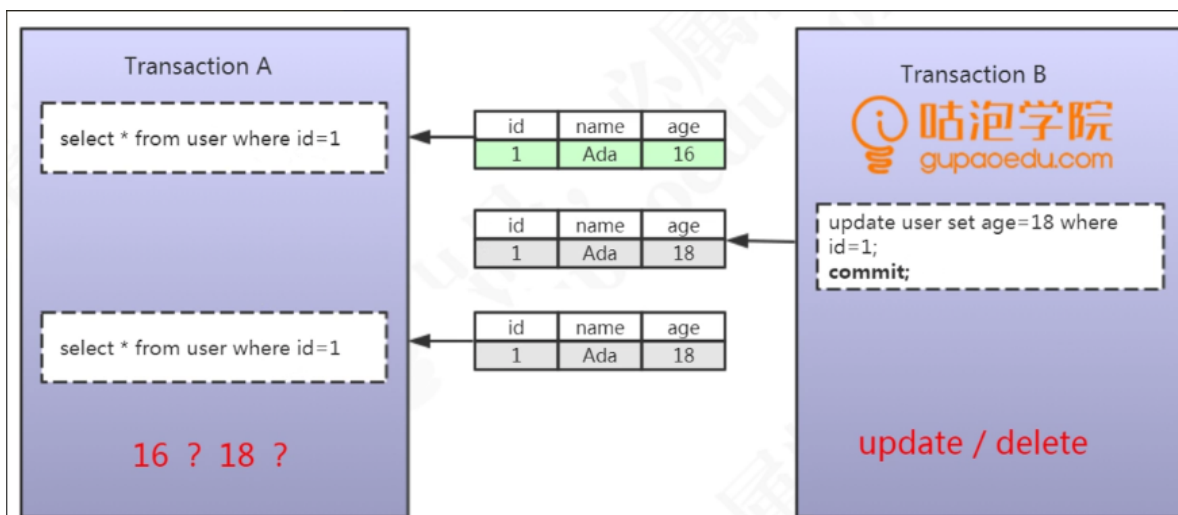
事务隔离级别指的是一个事务对数据的修改与另一个并行的事务的隔离程度,当多个事务同时访问相同数据时,如果没有采取必要的隔离机制,就可能发生以下问题:

问题	描述
脏读	一个事务读到另一个事务未提交的更新数据 , 所谓脏读, 就是指事务A读到了事务B还没有提交的数据, 比如银行取钱, 事务A开启事务, 此时切换到事务B, 事务B开启事务-->取走100元, 此时切换回事务A, 事务A读取的肯定是数据库里面的原始数据, 因为事务B取走了100块钱, 并没有提交, 数据库里面的账务余额肯定还是原始余额, 这就是脏读
幻读	是指当事务不是独立执行时发生的一种现象 , 例如第一个事务对一个表中的数据进行了修改, 这种修改涉及到表中的全部数据行。同时, 第二个事务也修改这个表中的数据, 这种修改是向表中插入一行新数据。那么, 以后就会发生操作第一个事务的用户发现表中还有没有修改的数据行, 就好象 发生了幻觉一样。
不可重复读	在一个事务里面的操作中发现了未被操作的数据 比方说在同一个事务中先后执行两条一模一样的select语句, 期间在此次事务中没有执行过任何DDL语句, 但先后得到的结果不一致, 这就是不可重复读

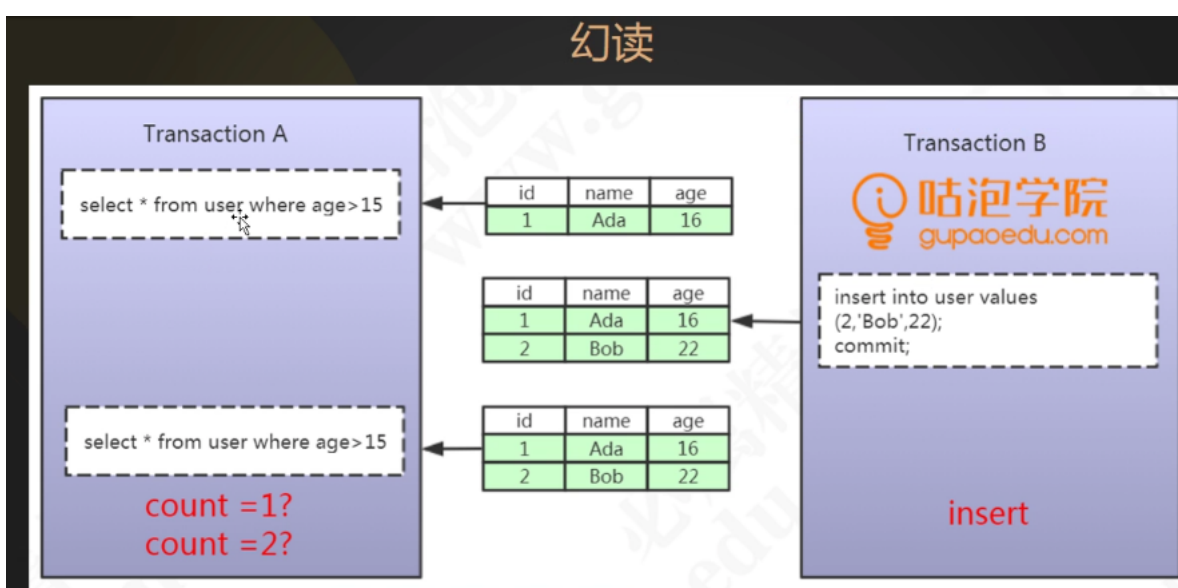
脏读



不可重复读【修改删除】



幻读【插入】



隔离级别	描述
DEFAULT	使用数据库本身使用的隔离级别 ORACLE (读已提交) MySQL (可重复读)
READ_UNCOMMITTED	读未提交 (脏读) 最低的隔离级别, 一切皆有可能。
READ_COMMITTED	读已提交, ORACLE默认隔离级别, 有幻读以及不可重复读风险。
REPEATABLE_READ	可重复读, 解决不可重复读的隔离级别, 但还是有幻读风险。
SERIALIZABLE	串行化, 最高的事务隔离级别, 不管多少事务, 挨个运行完一个事务的所有子事务之后才可以执行另外一个事务里面的所有子事务, 这样就解决了脏读、不可重复读和幻读的问题了

隔离级	脏读可能性	不可重复读可能性	幻读可能性	加锁读
READ UNCOMMITTED	是	是	是	否
READ COMMITTED	否	是	是	否
REPEATABLE READ	否	否	是	否
SERIALIZABLE	否	否	否	是

再必须强调一遍，不是事务隔离级别设置得越高越好，事务隔离级别设置得越高，意味着势必要花手段去加锁用以保证事务的正确性，那么效率就要降低，因此实际开发中往往要在效率和并发正确性之间做一个取舍，一般情况下会设置为READ_COMMITTED，此时避免了脏读，并发性也还不错，之后再通过一些别的手段去解决不可重复读和幻读的问题就好了。

Spring设置事务隔离级别

配置文件的方式

```
<tx:advice id="advice" transaction-manager="transactionManager">
  <tx:attributes>
    <tx:method name="fun*" propagation="REQUIRED" isolation="DEFAULT"/>
  </tx:attributes>
</tx:advice>
```

注解的方式

```
@Transactional(isolation=Isolation.DEFAULT)
public void fun(){
    dao.add();
    dao.udpate();
}
```

小结

Spring建议的是使用**DEFAULT**，就是数据库本身的隔离级别，配置好数据库本身的隔离级别，无论在哪个框架中读写数据都不用操心了。而且万一Spring没有把这几种隔离级别实现的很完善，出了问题就麻烦了。