Spring lesson 7

xieqiaoyun

知识要点

- 了解事物隔离级别
- 了解事务传播规则
- 了解事务回滚机制
- 掌握声明式事务或编程式事务使用

（一）验证跨线程的情况下，事务是怎样处理的

编写PersonService2组件和PersonService1组件，更改propagation和isolation调试

personService2

```
@Transactional(propagation = Propagation.REQUIRED, isolation = Isolation.SERIALIZABLE)
public void updatePerson(int id,String city,String country){
    System.out.println("-->> try to update city[ "+city+" ] ... ");
    ThreadLocalUtil.dumphreadLocals();
    System.out.println("  Δ before update: " + personService1.getPerson(id));
    int rows = jdbcTemplate.update( sql: "update person set city=?,country=? where id=?", new Object[]{city,cc
    if(rows == 0){
        throw new RuntimeException("  X update person failure, do rollback...");
    }
    System.out.println("  Δ after update: " + personService1.getPerson(id));
    System.out.println("  √ update person complete ... ");
}
```

personService1

```
@Transactional(propagation = Propagation.REQUIRED)
public Person getPerson(int id){
    System.out.println("-->> try to get a person ... ");
    ThreadLocalUtil.dumphreadLocals();
    Person person = jdbcTemplate.queryForObject( sql: "select * from person where id=?",
            new Object[]{id},
            (rs, i) -> {
                Person p = new Person(rs.getString( columnLabel: "first_name"),rs.getString( columnLabel: "last
                p.setId(rs.getInt( columnLabel: "id"));
                return p;
            });
    System.out.println("  √ get a person: " + person);
    return person;
}
```

（一）验证跨线程的情况下，事务是怎样处理的

编写测试类，开启多线程调用带有事务的方法

```
        ApplicationContext ctx = new AnnotationConfigApplicationContext(TransactionConfig.class);

        IPersonService personService1 = ctx.getBean( s: "personService1", IPersonService.class);
        ExecutorService executorService = Executors.newFixedThreadPool( nThreads: 2);
        List<Person> persons = (List<Person>) personService1.queryAll();
//...

        System.out.println();
//        List<Person> persons2 = (List<Person>) personService1.queryPerson("Java");
//        System.out.println();
        PersonService2 personService2 = (PersonService2) ctx.getBean( s: "personService2");
//        personService2.updatePerson(2,"SZ-"+(Math.round(Math.random()*100)),"China");
        for(int i=0;i<4;i++) {
            int finalI = i;
            executorService.execute(() -> {
                personService2.updatePerson( id: 2, city: "SZ-"+(Math.round(Math.random()*100)), country: "China");
            });
        }
        executorService.shutdown();
```

（一）验证跨线程的情况下，事务是怎样处理的

事务不同的隔离级别、传播规则，将产生不同的效果

（一）事务的隔离级别

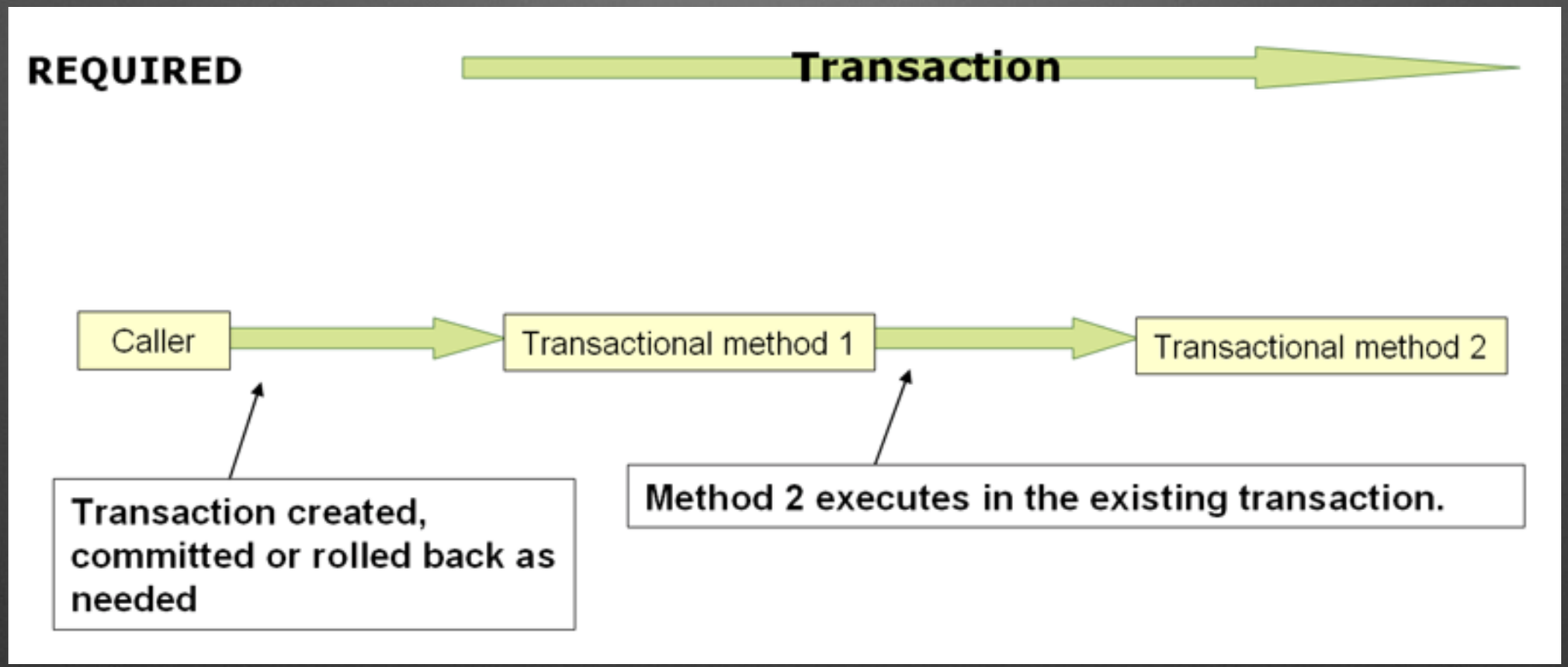
- Default — 使用底层数据库默认级别
- READ_COMMITTED — 可发生不可重复读或幻象读(读到的数据是不确定的)
- READ_UNCOMMITTED — 读到未提交的数据
- REPEATABLE_READ — 可重复读，也许读到的数据是不确定的
- SERIALIZABLE — 锁的最高级别，防止不可重复读和幻想读

（一）事务传播规则

- REQUIRED — 没有事务将创建事务（后续方法将共享同一个事务环境）
- REQUIRED_NEW — 调用该方法前开启一个新事务，原来事务将挂起
- MANDATORY — 强制要求事务，如果没有将抛异常
- SUPPORT — 不管有没事务，将正常执行，常用于拉取数据
- NOT_SUPPORT — 不需要传播事务上下文。常见于事务环境中执行内存的操作
- NEVER — 不能在事务环境中运行，否则将抛异常
- NESTED — 嵌套事务，使用同一个物理事务，根据设置回滚点回滚

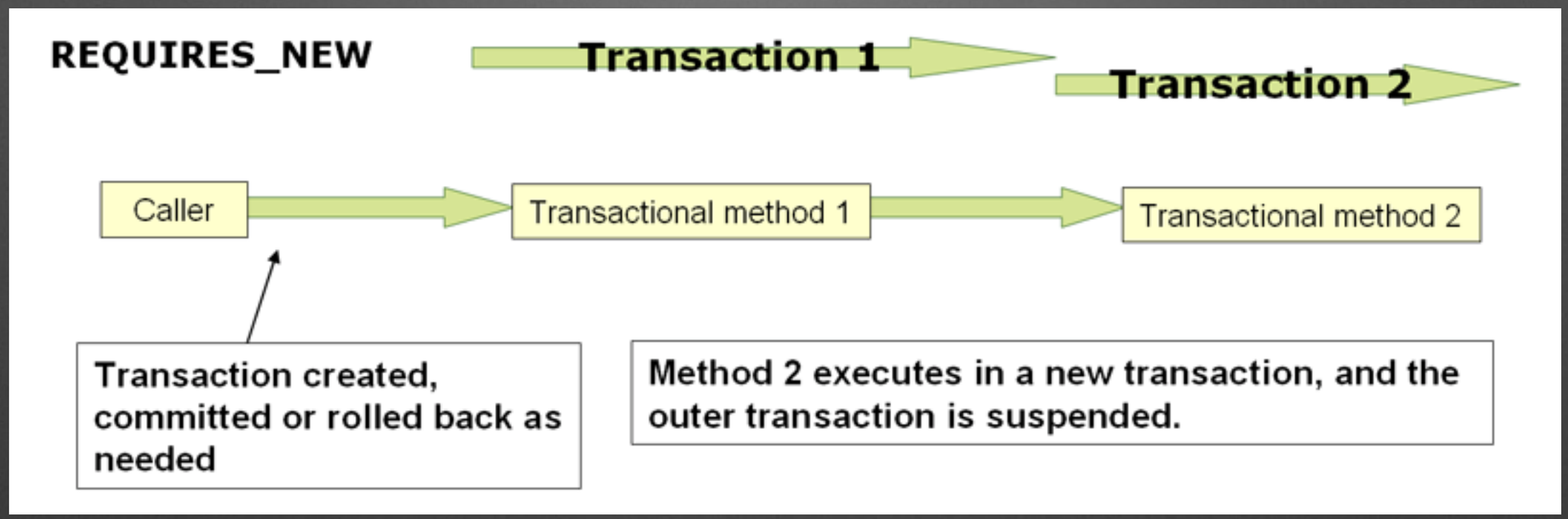
PROPAGATION_REQUIRED

- 没有事务将创建事务（后续方法将共享同一个事务环境）



PROPAGATION_REQUIRES_NEW

- 调用该方法前开启一个新事务，原来事务将挂起



（一）验证跨线程的情况下，事务是怎样处理的

Example:

- personService2(事务 + SERIALIZABLE)调用personService1的(REQUIRED/REQUIRED_NEW/MANDATORY/SUPPORT)事务 -> 资源竞争，容易造成死锁

```
19:05:35.773 [pool-1-thread-3] DEBUG org.springframework.jdbc.datasource.DataSourceTransactionManager - Initiating transaction rollbac
19:05:35.774 [pool-1-thread-1] DEBUG org.springframework.jdbc.datasource.DataSourceTransactionManager - Initiating transaction rollbac
19:05:35.775 [pool-1-thread-4] DEBUG org.springframework.jdbc.datasource.DataSourceTransactionManager - Rolling back JDBC transaction
19:05:35.775 [pool-1-thread-3] DEBUG org.springframework.jdbc.datasource.DataSourceTransactionManager - Rolling back JDBC transaction
19:05:35.776 [pool-1-thread-1] DEBUG org.springframework.jdbc.datasource.DataSourceTransactionManager - Rolling back JDBC transaction
19:05:35.776 [pool-1-thread-3] DEBUG org.springframework.jdbc.datasource.DataSourceUtils - Resetting isolation level of JDBC Connectio
19:05:35.776 [pool-1-thread-4] DEBUG org.springframework.jdbc.datasource.DataSourceUtils - Resetting isolation level of JDBC Connectio
Exception in thread "pool-1-thread-4" Exception in thread "pool-1-thread-3" org.springframework.dao.DeadlockLoserDataAccessException:
    at org.springframework.jdbc.support.SQLErrorCodeSQLExceptionTranslator.doTranslate(SQLErrorCodeSQLExceptionTranslator.java:263)
    at org.springframework.jdbc.support.AbstractFallbackSQLExceptionTranslator.translate(AbstractFallbackSQLExceptionTranslator.java:7
    at org.springframework.jdbc.core.JdbcTemplate.execute(JdbcTemplate.java:649)
```

（一）验证跨线程的情况下，事务是怎样处理的

- personService2(事务+SERIALIZABLE)调用personService1(NOT_SUPPORT) -> 先挂起当前事务，再执行

```
19:12:20.089 [pool-1-thread-4] DEBUG org.springframework.jdbc.core.JdbcTemplate - SQL update affected 1 rows
19:12:20.090 [pool-1-thread-4] DEBUG org.springframework.jdbc.datasource.DataSourceTransactionManager - Suspending current transaction
-->> try to get a person ...
_____________ [ dump thread start ] ____________
...Spring threadlocal var: Current transaction name=leader.tx.PersonService1.getPerson
...Spring threadlocal var: Current aspect-driven transaction=PROPAGATION_NOT_SUPPORTED,ISOLATION_DEFAULT; ''
...Spring threadlocal var: Current transaction read-only status=null
...Spring threadlocal var: Current transaction isolation level=null
...Spring threadlocal var: Actual transaction active=null
...Spring threadlocal var: Transaction synchronizations=[]
_____________ [ dump  thread  end ] ____________
19:12:20.090 [pool-1-thread-4] DEBUG org.springframework.jdbc.core.JdbcTemplate - Executing prepared SQL query
19:12:20.091 [pool-1-thread-4] DEBUG org.springframework.jdbc.core.JdbcTemplate - Executing prepared SQL statement [select * from person
19:12:20.091 [pool-1-thread-4] DEBUG org.springframework.jdbc.datasource.DataSourceUtils - Fetching JDBC Connection from DataSource
19:12:20.091 [pool-1-thread-4] DEBUG org.springframework.jdbc.datasource.DataSourceUtils - Registering transaction synchronization for JI
  √ get a person: Person{id=2 fist_name=Java last_name=Honk city=SZ-55 country=China}
19:12:20.093 [pool-1-thread-4] DEBUG org.springframework.jdbc.datasource.DataSourceUtils - Returning JDBC Connection to DataSource
19:12:20.093 [pool-1-thread-4] DEBUG org.springframework.jdbc.datasource.DataSourceTransactionManager - Resuming suspended transaction a
  Δ after update: Person{id=2 fist_name=Java last_name=Honk city=SZ-55 country=China}
```

（一）验证跨线程的情况下，事务是怎样处理的

- personService2(SUPPORT)调用personService1(NEVER) -> 事务和非事务环境都执行
- personService2(NOT_SUPPORT)调用personService1(NEVER) -> 非事务环境执行

```
19:22:48.628 [pool-1-thread-1] DEBUG org.springframework.jdbc.core.JdbcTemplate - SQL update affected 1 rows
-->> try to get a person ...
_____________ [ dump thread start ] ____________
...Spring threadlocal var: Current transaction name=leader.tx.PersonService2.updatePerson
...Spring threadlocal var: Current aspect-driven transaction=PROPAGATION_NEVER,ISOLATION_DEFAULT; ''
...Spring threadlocal var: Current transaction read-only status=null
...Spring threadlocal var: Current transaction isolation level=1
...Spring threadlocal var: Transactional resources={org.apache.commons.dbcp.BasicDataSource@59474f18=org.springframework.jdbc.dataso
...Spring threadlocal var: Actual transaction active=null
...Spring threadlocal var: Transaction synchronizations=[org.springframework.jdbc.datasource.DataSourceUtils$ConnectionSynchronizati
```

SUPPORT

```
_____________ [ dump  thread  end ] ____________
19:22:48.629 [pool-1-thread-1] DEBUG org.springframework.jdbc.core.JdbcTemplate - Executing prepared SQL query
19:22:48.629 [pool-1-thread-1] DEBUG org.springframework.jdbc.core.JdbcTemplate - Executing prepared SQL statement [select * from pe
  √ get a person: Person{id=2 fist_name=Java last_name=Honk city=SZ-34 country=China}
  Δ after update: Person{id=2 fist_name=Java last_name=Honk city=SZ-34 country=China}
  √ update person complete ...
19:22:48.631 [pool-1-thread-1] DEBUG org.springframework.jdbc.datasource.DataSourceUtils - Returning JDBC Connection to DataSource
```

```
_____________ [ dump  thread  end ] ____________
  √ get a person: Person{id=2 fist_name=Java last_name=Honk city=SZ-63 country=China}
19:27:26.219 [pool-1-thread-3] DEBUG org.springframework.jdbc.core.JdbcTemplate - Executing prepared SQL query
  Δ after update: Person{id=2 fist_name=Java last_name=Honk city=SZ-63 country=China}
  √ update person complete ...
19:27:26.219 [pool-1-thread-3] DEBUG org.springframework.jdbc.core.JdbcTemplate - Executing prepared SQL statement [select * from pe
  √ get a person: Person{id=2 fist_name=Java last_name=Honk city=SZ-14 country=China}
  Δ after update: Person{id=2 fist_name=Java last_name=Honk city=SZ-14 country=China}
  √ update person complete ...
19:27:26.220 [pool-1-thread-4] DEBUG org.springframework.jdbc.datasource.DataSourceUtils - Returning JDBC Connection to DataSource
19:27:26.219 [pool-1-thread-1] DEBUG org.springframework.jdbc.datasource.DataSourceUtils - Returning JDBC Connection to DataSource
  √ get a person: Person{id=2 fist_name=Java last_name=Honk city=SZ-14 country=China}
  Δ after update: Person{id=2 fist_name=Java last_name=Honk city=SZ-14 country=China}
  √ update person complete
```

NOT_SUPPORT

并没有启用事务，跟普通的方法调用一样

（一）验证跨线程的情况下，事务是怎样处理的

- personService2(REQUIRED/REQUIRED_NEW/MANDATORY)调用personService1(NEVER) -> 抛异常。不能在事务环境执行

```
19:31:04.428 [pool-1-thread-1] DEBUG org.springframework.jdbc.datasource.DataSourceTransactionManager - Releasing JDBC Connection [jdbc:
19:31:04.428 [pool-1-thread-3] DEBUG org.springframework.jdbc.datasource.DataSourceTransactionManager - Releasing JDBC Connection [jdbc:
19:31:04.428 [pool-1-thread-1] DEBUG org.springframework.jdbc.datasource.DataSourceUtils - Returning JDBC Connection to DataSource
19:31:04.428 [pool-1-thread-3] DEBUG org.springframework.jdbc.datasource.DataSourceUtils - Returning JDBC Connection to DataSource
Exception in thread "pool-1-thread-2" org.springframework.transaction.IllegalTransactionStateException: Existing transaction found for t
    at org.springframework.transaction.support.AbstractPlatformTransactionManager.handleExistingTransaction(AbstractPlatformTransactionM
    at org.springframework.transaction.support.AbstractPlatformTransactionManager.getTransaction(AbstractPlatformTransactionManager.java
    at org.springframework.transaction.interceptor.TransactionAspectSupport.createTransactionIfNecessary(TransactionAspectSupport.java:4
    at org.springframework.transaction.interceptor.TransactionAspectSupport.invokeWithinTransaction(TransactionAspectSupport.java:277)
    at org.springframework.transaction.interceptor.TransactionInterceptor.invoke(TransactionInterceptor.java:96)
    at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:179)
    at org.springframework.aop.framework.JdkDynamicAopProxy.invoke(JdkDynamicAopProxy.java:213) <1 internal calls>
    at leader.tx.PersonService2.updatePerson(PersonService2.java:40)
    at leader.tx.PersonService2$$FastClassBySpringCGLIB$$2a827db7.invoke(<generated>)
    at org.springframework.cglib.proxy.MethodProxy.invoke(MethodProxy.java:204)
    at org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.invokeJoinpoint(CglibAopProxy.java:738)
    at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:157)
```

```
r - Releasing JDBC Connection [jdbc:mysql://localhost:3306/leader, UserName=roo
r - Releasing JDBC Connection [jdbc:mysql://localhost:3306/leader, UserName=roo
 JDBC Connection to DataSource
 JDBC Connection to DataSource
on: Existing transaction found for transaction marked with propagation 'never'
saction(AbstractPlatformTransactionManager.java:405)
tractPlatformTransactionManager.java:349)
sary(TransactionAspectSupport.java:447)
ransactionAspectSupport.java:277)
tor.java:96)
on.java:179)
nternal calls>
```

（二）验证针对特定的应用异常，SPRING托管的事务回滚的问题

- Spring托管的事务，默认遇到uncheck异常和RuntimeException时执行回滚
- 如新开事务运行，先回滚内部事务，再恢复外部事务
- 如嵌套事务，回滚到savepoint

NESTED

```
k.jdbc.core.JdbcTemplate - Executing prepared SQL query
k.jdbc.core.JdbcTemplate - Executing prepared SQL statement [select * from person where id=?]
k.jdbc.datasource.DataSourceTransactionManager - Rolling back transaction to savepoint
k.jdbc.datasource.DataSourceTransactionManager - Initiating transaction rollback
k.jdbc.datasource.DataSourceTransactionManager - Rolling back JDBC transaction on Connection [jdbc:mysql
k.jdbc.datasource.DataSourceUtils - Resetting isolation level of JDBC Connection [jdbc:mysql://localhost
k.jdbc.datasource.DataSourceTransactionManager - Releasing JDBC Connection [jdbc:mysql://localhost:3306/
k.jdbc.datasource.DataSourceUtils - Returning JDBC Connection to DataSource
```

REQUIRE_NEW

```
mplate - Executing prepared SQL query
mplate - Executing prepared SQL statement [select * from person where id=?]
DataSourceTransactionManager - Initiating transaction rollback
DataSourceTransactionManager - Rolling back JDBC transaction on Connection [jdbc:mysql://localhost:3306/leader, UserName=root@localhost, MySQL Connecto
DataSourceTransactionManager - Releasing JDBC Connection [jdbc:mysql://localhost:3306/leader, UserName=root@localhost, MySQL Connector Java] after trans
DataSourceUtils - Returning JDBC Connection to DataSource
DataSourceTransactionManager - Resuming suspended transaction after completion of inner transaction
DataSourceTransactionManager - Initiating transaction rollback
DataSourceTransactionManager - Rolling back JDBC transaction on Connection [jdbc:mysql://localhost:3306/leader, UserName=root@localhost, MySQL Connecto
DataSourceUtils - Resetting isolation level of JDBC Connection [jdbc:mysql://localhost:3306/leader, UserName=root@localhost, MySQL Connector Java] to 4
```

（三）从日志角度分析，SPRING托管事务的开始和结束边界在哪里?

- jdk代理：可以在接口上标记事务，并且只作用到public方法上
- cglib代理：由于通过继承实现，只能在类上标记事务，可以是非public方法

（三）从日志角度分析，SPRING托管事务的开始和结束边界在哪里?

```
public interface IPersonService {
    @Transactional(propagation = Propagation.REQUIRED, is
    List<?> queryAll();
```

```
@Configuration
@EnableAspectJAutoProxy
//@EnableAspectJAutoProxy(proxyTargetClass = true)
@EnableTransactionManagement
@ComponentScan(basePackages = {"leader.tx"})
public class TransactionConfig {
```

```
@Component
public class PersonService1 implements IPersonService{
    public PersonService1() { System.out.println("------
    @Autowired
    JdbcTemplate jdbcTemplate;
    public List<?> queryAll(){
        System.out.println("-->> try to query all person
        ThreadLocalUtil.dumphreadLocals();
        return query( keyname: "");
```

切换jdk代理和cglib代理调试

```
ic static void main(String[] args) throws IOException {
ApplicationContext ctx = new AnnotationConfigApplicationContext(
System.out.println("......context is created...");
IPersonService personService1 = ctx.getBean( s: "personService1",
        IPersonService.class);
List<Person> persons = (List<Person>) personService1.queryAll();
```

```
@Configuration
@EnableAspectJAutoProxy(proxyTargetClass = true)
@EnableTransactionManagement
@ComponentScan(basePackages = {"leader.tx"})
public class TransactionConfig {
```

proxyTargetClass=false

```
[ dump thread start ] ------------
dlocal var: Current transaction name=leader.tx.PersonService1.c
dlocal var: Current aspect-driven transaction=PROPAGATION_REQUI
dlocal var: Prototype beans currently in creation=null
dlocal var: Current transaction read-only status=null
dlocal var: Current transaction isolation level=8
dlocal var: Transactional resources={org.apache.commons.dbcp.Ba
dlocal var: Actual transaction active=true
dlocal var: Transaction synchronizations=[]
[ dump  thread  end ] ------------
ain] DEBUG org.springframework.jdbc.core.JdbcTemplate - Executi
t name=Java last name=Honk city=SZ-14 country=China}
```

proxyTargetClass=true

```
743 [main] DEBUG org.springframework.beans.factory.support.Defa
818 [main] DEBUG org.springframework.context.annotation.Annotat
819 [main] DEBUG org.springframework.beans.factory.support.Defa
823 [main] DEBUG org.springframework.core.env.PropertySourcesPr
text is created...
825 [main] DEBUG org.springframework.beans.factory.support.Defa
to query all person  ...
_____ [ dump thread start ] ------------
 threadlocal var: Prototype beans currently in creation=null
_____ [ dump  thread  end ] ------------
203 [main] DEBUG org.springframework.jdbc.core.JdbcTemplate - E
211 [main] DEBUG org.springframework.jdbc.datasource.DataSource
```

（四）事务注解到接口上的时候，SPRING如何扫描发现接口实例是需要被生成代理的?

接口

```java
public interface IPersonService {

    @Transactional(propagation = Propagation.REQUIRED, isolation = Isolation.SERIALIZABLE)
    List<?> queryAll();

    List<?> queryPerson(String keyname);

    Person getPerson(int id);
```

接口实现类

```java
@Component
public class PersonService1 implements IPersonService{

    @Autowired
    JdbcTemplate jdbcTemplate;

    public List<?> queryAll(){
        System.out.println("-->> try to query all person  ...
        ThreadLocalUtil.dumphreadLocals();
        return query( keyname: "");
    }
```

bean配置类，开启事务管理注解

```java
@Configuration
@EnableAspectJAutoProxy
//@EnableAspectJAutoProxy(proxyTargetClass = true)
@EnableTransactionManagement
@ComponentScan(basePackages = {"leader.tx"})
public class TransactionConfig {

    @Bean
    public DataSource dataSource(){
        String connectionString = "jdbc:mysql://localhost:3306/leader";
        BasicDataSource dataSource = new BasicDataSource();
```

测试案例

```java
    IPersonService personService1 = ctx.getBean( s: "personService1", IPersonService.class);
    List<Person> persons = (List<Person>) personService1.queryAll();
```

（四）事务注解到接口上的时候，SPRING如何扫描发现接口实例是需要被生成代理的?

- 从EnableTransactionManager注解，注册Spring事务相关需要的(基础设施)组件

```
pringframework.transaction.annotation.ProxyTransactionManagementConfiguration'
ingframework.transaction.annotation.ProxyTransactionManagementConfiguration.org.springframework.transaction.config.internalTransactionAdvisor()
ingframework.transaction.annotation.ProxyTransactionManagementConfiguration.transactionAttributeSource()
ingframework.transaction.annotation.ProxyTransactionManagementConfiguration.transactionInterceptor()
ingframework.transaction.annotation.ProxyTransactionManagementConfiguration.org.springframework.transaction.config.internalTransactionalEventListenerFactory
x.TransactionConfig.dataSource()
```

事务拦截

如果开启@EnableAspectJAutoProxy(proxyTargetClass = true)使用cglib代理，注册到接口上的事务将不会被扫描到

- 根据bean定义，实例化目标对象，生成代理对象，并注入目标对象

```
354 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'personService1'
354 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating instance of bean 'personService1'
354 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Returning cached instance of singleton bean 'org.springfram
rsonService1 loaded...
rsonService1 created...
360 [main] DEBUG org.springframework.beans.factory.annotation.InjectionMetadata - Registered injected element on class [leader.tx.PersonService1]:
360 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Eagerly caching bean 'personService1' to allow for resolvin
362 [main] DEBUG org.springframework.beans.factory.annotation.InjectionMetadata - Processing injected element of bean 'personService1': AutowiredFi
363 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Returning cached instance of singleton bean 'jdbcTemplate'
363 [main] DEBUG org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor - Autowiring by type from bean name 'personServi
363 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Returning cached instance of singleton bean 'org.springfram
363 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Returning cached instance of singleton bean 'org.springfram
366 [main] DEBUG org.springframework.transaction.annotation.AnnotationTransactionAttributeSource - Adding transactional method 'leader.tx.PersonSer
375 [main] DEBUG org.springframework.aop.aspectj.annotation.AnnotationAwareAspectJAutoProxyCreator - Creating implicit proxy for bean 'personServic
376 [main] DEBUG org.springframework.aop.framework.JdkDynamicAopProxy - Creating JDK dynamic proxy: target source is SingletonTargetSource for targ
380 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Finished creating instance of bean 'personService1'
```

（五）模仿最后的AspectJ事务模式，来实现编程控制新事物的做法。日志验证

Spring提供两种方式来编程式事务：

继承PlatformTransactionManager的实现类

或使用TransactionTemplate)

继承PlatformTransactionManager实现类上一次课已使用，下面试试TransactionTemplate

（五）模仿最后的AspectJ事务模式，来实现编程控制新事物的做法。日志验证

```java
@Component
public class EmployeeService {
    @Autowired
    JdbcTemplate jdbcTemplate;

    @Autowired
    TransactionTemplate transactionTemplate;
```

Service注入TransactionTemplate

```java
public void updateEmployee(int empid, long salary){
    System.out.println("    try to update employee ... "+empid);
    transactionTemplate.execute(new TransactionCallbackWithoutResult() {
        @Override
        protected void doInTransactionWithoutResult(TransactionStatus transactionStatus) {
            try {
                int rows = jdbcTemplate.update( sql: "update employee set salary = ? where empid = ?", n
                if(rows==0){
                    throw new RuntimeException("    X update error ");
                }
                System.out.println("    √ update employee ... ");
            } catch (Exception e) {
                System.out.println("    X update failure, do rollback...");
                transactionStatus.setRollbackOnly();
            }
        }
```

没有返回值的

和

有返回值的

各写一个

```java
public int deleteEmployee(int empid){
    System.out.println("    try to delete employee ... "+empid);
    return transactionTemplate.execute(new TransactionCallback<Integer>() {
        @Override
        public Integer doInTransaction(TransactionStatus transactionStatus) {
            try {
                int rows = jdbcTemplate.update( sql: "delete from employee where empid=?", empid);
                if(rows==0){
                    throw new RuntimeException("    X delete error ");
                }
                System.out.println("    √ delete employee ... ");
                return rows;
            } catch (Exception e) {
                System.out.println("    X delete failure, do rollback...");
                transactionStatus.setRollbackOnly();
            }
            return 0;
```

（五）测试案例及运行测试

```
13:14:52.917 [main] DEBUG org.springframework.jdbc.datasource
    √ query 3 employees ...
        EMPLOYEE{empid=2 name=emp002 age=23 salary=2200}
        EMPLOYEE{empid=3 name=emp003 age=25 salary=3200}
        EMPLOYEE{empid=35 name=employee35 age=0 salary=1916}
    try to create employee ...
13:14:52.919 [main] DEBUG org.springframework.jdbc.datasource
```

```
    √ query 2 employees ...
        EMPLOYEE{empid=15 name=CPP-15 age=33 salary=15000}
        EMPLOYEE{empid=21 name=CPP-21 age=33 salary=21000}
    try to update employee ... 16
13:14:52.963 [main] DEBUG org.springframework.jdbc.datasource
13:14:52.964 [main] DEBUG org.springframework.jdbc.datasource
13:14:52.966 [main] DEBUG org.springframework.jdbc.datasource
13:14:52.966 [main] DEBUG org.springframework.jdbc.core.JdbcT
13:14:52.966 [main] DEBUG org.springframework.jdbc.core.JdbcT
13:14:52.967 [main] DEBUG org.springframework.jdbc.core.JdbcT
    X update failure, do rollback...
13:14:52.967 [main] DEBUG org.springframework.jdbc.datasource
13:14:52.968 [main] DEBUG org.springframework.jdbc.datasource
13:14:52.968 [main] DEBUG org.springframework.jdbc.datasource
13:14:52.971 [main] DEBUG org.springframework.jdbc.datasource
13:14:52.971 [main] DEBUG org.springframework.jdbc.datasource
13:14:52.971 [main] DEBUG org.springframework.jdbc.core.JdbcT
13:14:52.971 [main] DEBUG org.springframework.jdbc.datasource
13:14:52.973 [main] DEBUG org.springframework.jdbc.datasource
    √ query 2 employees ...
        EMPLOYEE{empid=15 name=CPP-15 age=33 salary=15000}
        EMPLOYEE{empid=21 name=CPP-21 age=33 salary=21000}
```

没有实现AOP，不合格。。。。

```
EmployeeService employeeService = ctx.getBean( s: "employeeSe
List<Employee> list = employeeService.queryEmployees( name: "
int empid = 0;
for(int i=0;i<2;i++){
    empid = Math.toIntExact(Math.round(Math.random() * 100))
    employeeService.createEmployee(new Employee(empid, name:
}
employeeService.queryEmployees( name: "CPP");
employeeService.updateEmployee( empid: 16, salary: 25000);
employeeService.queryEmployees( name: "CPP");
employeeService.deleteEmployee(empid);
employeeService.queryEmployees( name: "CPP");
employeeService.deleteEmployee( empid: 1);
```

```
    try to delete employee ... 21
3:14:52.974 [main] DEBUG org.springframework.jdbc.datasource.
3:14:52.975 [main] DEBUG org.springframework.jdbc.datasource.
3:14:52.976 [main] DEBUG org.springframework.jdbc.datasource.
3:14:52.976 [main] DEBUG org.springframework.jdbc.core.JdbcTe
3:14:52.976 [main] DEBUG org.springframework.jdbc.core.JdbcTe
3:14:52.977 [main] DEBUG org.springframework.jdbc.core.JdbcTe
    √ delete employee ...
3:14:52.979 [main] DEBUG org.springframework.jdbc.datasource.
3:14:52.980 [main] DEBUG org.springframework.jdbc.datasource.
3:14:52.984 [main] DEBUG org.springframework.jdbc.datasource.
3:14:52.984 [main] DEBUG org.springframework.jdbc.datasource.
3:14:52.984 [main] DEBUG org.springframework.jdbc.core.JdbcTe
3:14:52.984 [main] DEBUG org.springframework.jdbc.datasource.
3:14:52.986 [main] DEBUG org.springframework.jdbc.datasource.
    √ query 1 employees ...
        EMPLOYEE{empid=15 name=CPP-15 age=33 salary=15000}
    try to delete employee ... 1
3:14:52.986 [main] DEBUG org.springframework.jdbc.datasource.
3:14:52.987 [main] DEBUG org.springframework.jdbc.datasource.
3:14:52.988 [main] DEBUG org.springframework.jdbc.datasource.
3:14:52.989 [main] DEBUG org.springframework.jdbc.core.JdbcTe
3:14:52.990 [main] DEBUG org.springframework.jdbc.core.JdbcTe
3:14:52.991 [main] DEBUG org.springframework.jdbc.core.JdbcTe
    X delete failure, do rollback...
3:14:52.991 [main] DEBUG org.springframework.jdbc.datasource.
```

Thank you

–Xieqiaoyun