基于Spring JDBC的事务管理

事务（Transaction）：是一种数据库中能够保证一系列的写操作（增、删、改）要么全部成功，要么全部失败的机制。

假设存在某个转账行为：

```
update 账户表 set 余额=余额-1000 where 账户名='传奇';
```

```
update 账户表 set 余额=余额+1000 where 账户名='克晶';
```

以上2个`UPDATE`操作，无论它们执行先后顺序是哪种，只要第1条执行成功，第2条没有执行成功，就是不可接受的结果！

在基于Spring JDBC的数据库编程中，如果需要某个方法是"事务性"的，在此方法上添加`@Transactional`注解即可！

关于`@Transactional`注解，可以添加在：

- 方法上
  - 作用于当前方法，但此方法必须是重写的接口中声明过的方法
- 类上
  - 作用于当前类中所有重写的接口中声明过的方法，即自定义的方法不可能是事务性的
- 接口中的抽象方法上
  - 作用于实现了接口的类中重写的方法
- 接口上
  - 作用于当前接口的实现类中的所有重写的方法

> 提示：Spring JDBC框架在实现事务管理时，使用了基于接口的代理模式！

使用`@Transactional`注解时，应该在接口的抽象方法上使用此注解！但是，在学习过程中，建议在接口上添加此注解。

Spring JDBC在处理事务时，执行过程大致是：

```
try {
    开启事务：BEGIN
    执行业务
    提交事务：COMMIT
} catch (RuntimeException e) {
    回滚事务：ROLLBACK
}
```

默认情况下，Spring JDBC会在执行业务的过程中出现`RuntimeException`时执行回滚，如果需要根据其它异常执行回滚，可以配置`@Transactional`注解的`rollbackFor`属性，例如：

```
@Transactional(rollbackFor = {
    ServiceException.class,
    NullPointerException.class
})
```

或者，配置`rollbackForClassName`属性，例如：

```
@Transactional(rollbackForClassName = {
    "cn.tedu.csmall.product.ex.ServiceException",
    "java.lang.NullPointerException"
})
```

还可以配置`noRollbackFor`属性，指定哪些异常不会导致回滚，例如：

```
@Transactional(noRollbackFor = {
    ClassCastException.class,
    IndexOutOfBoundsException.class
})
```

也有对应的`noRollbackForClassName`属性。

注意：无论如何配置，导致回滚的异常类型都必须是`RuntimeException`的子孙类。

由于Spring JDBC会根据`RuntimeException`及其子孙类异常执行回滚，所以，在业务方法中执行任何增、删、改操作时，都应该及时获取受影响的行数，并判断此值是否符合预期，如果不符合，必须抛出`RuntimeException`或其子孙类异常，触发回滚机制！

另外，你应该了解事务的ACID特性，事务的传播、事务的隔离。