参考：

玩转Spring全家桶（丁雪丰）

前言

整理MySQL时就一直绕不开一个话题——事务，InnoDB引擎提供了事务支持，保证了数据的一致性。当然去使用事务肯定是客户端的事情，我们的系统对于MySQL就是客户端，当然在JDBC中，我们是有事务的处理方式的，我们称之为编程式事务，但是显示的去使用一来很繁杂，二来让事务侵入了业务代码多少有点冗余。

Spring两大干将——IoC和AOP，其中的AOP完美实现了面向切面编程，似乎是一个能不错解决事务代码侵入业务代码的有效方式。所以Spring官方也整合了事务的处理方式，因此诞生了声明式事务。本篇便就Spring事务相关展开去叙述。

Demo

我这里贴出一个声明式事务的使用的前奏，里面还涉及到声明式事务的一个陷阱，我们慢慢来看。

本事例运用了H2内嵌数据库，所以和平时使用MyBatis不太一样，我们把目光集中在事例的事务本身，具体代码实现，就不去详细介绍了。

启动类。

```
@SpringBootApplication
@EnableTransactionManagement(mode = AdviceMode.PROXY)
@Slf4j
public class DeclarativeTransactionDemoApplication implements CommandLineRunner {
    @Autowired
    private FooService fooService;
    @Autowired
    private JdbcTemplate jdbcTemplate;
```

```
    public static void main(String[] args) {
        SpringApplication.run(DeclarativeTransactionDemoApplication.class, args);
    }

    @Override
    public void run(String... args) throws Exception {
        fooService.insertRecord();
        log.info("AAA {}",
                jdbcTemplate
                        .queryForObject("SELECT COUNT(*) FROM FOO WHERE
BAR='AAA'", Long.class));
        try {
            fooService.insertThenRollback();
        } catch (Exception e) {
            log.info("BBB {}",
                    jdbcTemplate
                            .queryForObject("SELECT COUNT(*) FROM FOO WHERE
BAR='BBB'", Long.class));
        }

        try {
            fooService.invokeInsertThenRollback();
        } catch (Exception e) {
            log.info("BBB {}",
                    jdbcTemplate
                            .queryForObject("SELECT COUNT(*) FROM FOO WHERE
BAR='BBB'", Long.class));
        }
    }
}
```

@EnableTransactionManagement注解就是Spring Boot方式开启全局事务管理，为了代码篇幅的减少，我们是在Spring Boot启动类启动时直接执行的业务，粗略一看是三个操作，我们看看业务类。

```
@Component
public class FooServiceImpl implements FooService {
  @Autowired
  private JdbcTemplate jdbcTemplate;

  @Override
  @Transactional
  public void insertRecord() {
    jdbcTemplate.execute("INSERT INTO FOO (BAR) VALUES ('AAA')");
  }

  @Override
  @Transactional(rollbackFor = RollbackException.class)
  public void insertThenRollback() throws RollbackException {
    jdbcTemplate.execute("INSERT INTO FOO (BAR) VALUES ('BBB')");
    throw new RollbackException();
```

```
    }

    @Override
    public void invokeInsertThenRollback() throws RollbackException {
        insertThenRollback();
    }
}
```

一个标准的业务类，其中三个方法就是我这边声明式事务的演示了，第一个方法加了事务注解正常操作，第二个方法加了事务注解但进行了回滚操作，第三个方法调用了第二个方法。

想一想运行后，这三个方法结果都是什么呢？

```
2019-02-24 15:20:22.891  INFO 30088 --- [           main] .d.DeclarativeTransactionDemoApplication : AAA 1
2019-02-24 15:20:22.894  INFO 30088 --- [           main] .d.DeclarativeTransactionDemoApplication : BBB 0
2019-02-24 15:20:22.895  INFO 30088 --- [           main] .d.DeclarativeTransactionDemoApplication : BBB 1
```

是不是有一个有点出乎意料，就是第三个方法，调用的不就是第二个方法吗，为啥没回滚。这里我们就要搞清楚这个事务注解到底干了什么。

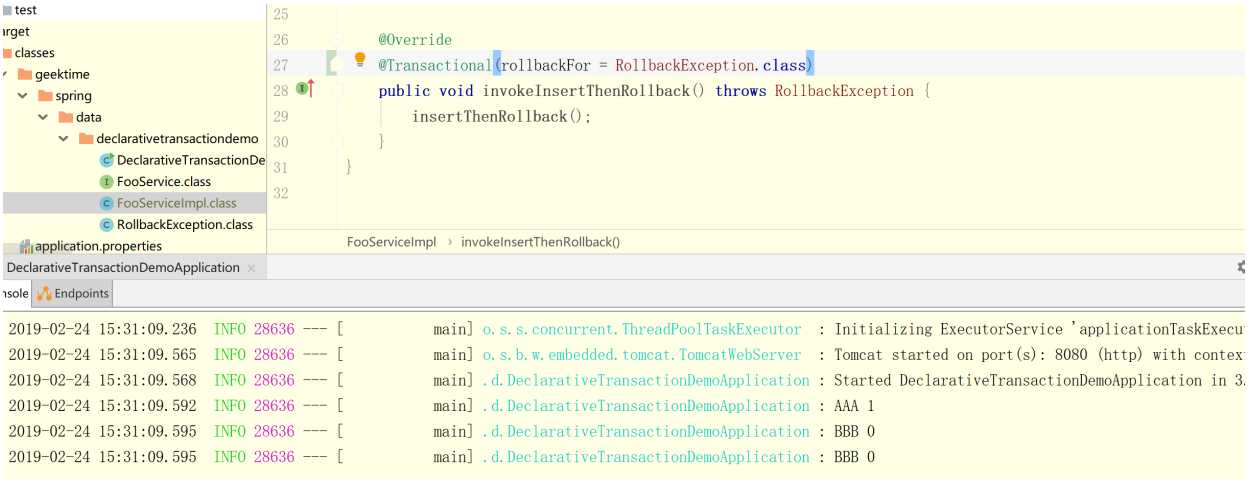
前言我也说了，声明式事务就是对Spring AOP的运用，这就要扯到代理模式了，我们也整理过代理模式，也知道Spring AOP整合了JDK动态代理（针对有接口的实现类）和CGLIb动态代理（针对光棍类）。

所以这里我们用了注解后，其实最终Spring管理的bean是代理生成的类而不是我们写的这个类本身，这样似乎就能理解了。

第三个方法调用的方法是我们自己写的类本身的方法，而不是代理类的方法，类本身的方法并没有事务处理，所以自然也不会有什么回滚的操作了。

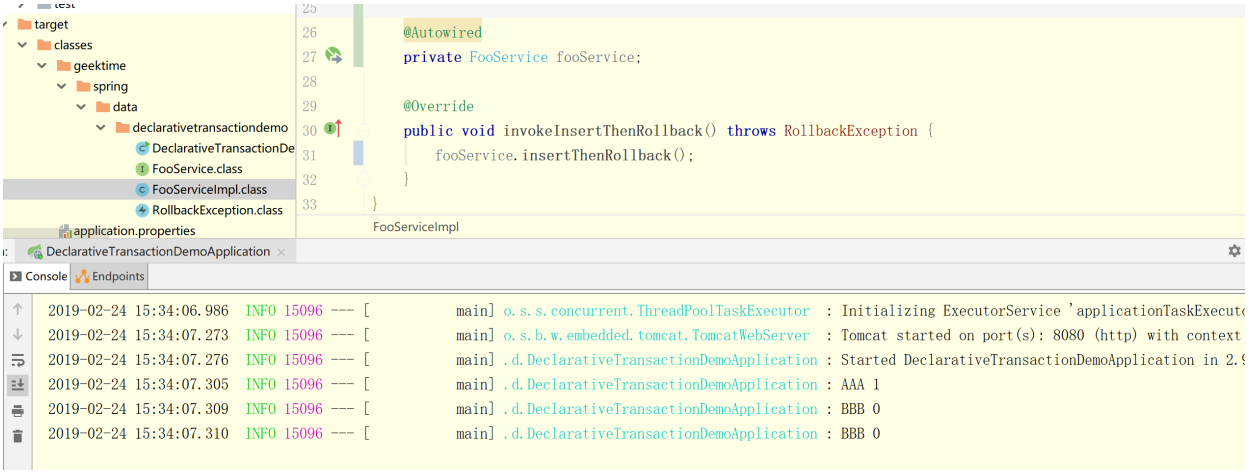
那么这里应该怎么让第三个方法能够使用事务呢？

1.笨方法，给这个方法也加事务注解，或者所有的方法都不加，类统一加。



那显然真是个笨方法，设计上都是不符合我们初衷的；

2.仔细想一想，我们不就是因为调用的不是代理类方法才有问题的吗，那我们访问代理类的方法不就是了，别忘了代理类的bean是Spring在管理的，我们完全是可以这样做的。



这才是智慧者的做法。

Spring事务隔离特性讲解

针对这个讲解可以先看一下MySQL的该笔记02-事务隔离的探究。

我们从源码中追寻下Spring隔离性的设置，@Transactional中。

```java
@Target({ElementType.METHOD, ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Inherited
@Documented
public @interface Transactional {
    @AliasFor("transactionManager")
    String value() default "";

    @AliasFor("value")
    String transactionManager() default "";

    Propagation propagation() default Propagation.REQUIRED;

    Isolation isolation() default Isolation.DEFAULT;
```

我们可以看到默认值是个DEFAULT，我们看看这个枚举类。

```java
public enum Isolation {
    DEFAULT( value: -1),
    READ_UNCOMMITTED( value: 1),
    READ_COMMITTED( value: 2),
    REPEATABLE_READ( value: 4),
    SERIALIZABLE( value: 8);
```

除了第一个DEFAULT下面四个都可以对应我们整理MySQL所讲：

1. READ_UNCOMMITTED　　读未提交
2. READ_COMMITTED　　读提交
3. REPEATABLE_READ　　可重复读
4. SERIALIZABLE　串行化

具体什么含义可以点击上文链接详细查看笔记，Spring这里说到底只是数据库的客户端，只负责设值，至于DEFAULT是什么含义，就是数据库本身设置的什么级别就什么级别，我Spring不管。

Spring事务传播特性讲解

概括一下就是如何开启一个事务，注解对于这个也是给了默认值的。

```
public @interface Transactional {
    @AliasFor("transactionManager")
    String value() default "";

    @AliasFor("value")
    String transactionManager() default "";

    Propagation propagation() default Propagation.REQUIRED;

    Isolation isolation() default Isolation.DEFAULT;
```

```
public enum Propagation {
    REQUIRED( value: 0),
    SUPPORTS( value: 1),
    MANDATORY( value: 2),
    REQUIRES_NEW( value: 3),
    NOT_SUPPORTED( value: 4),
    NEVER( value: 5),
    NESTED( value: 6);
```

具体这个七个特性的含义，我一一道来：

1. REQUIRED　　当前有事务就用当前的，没有就用新的
2. SUPPORTS　　事务可有可无，不是必须的
3. MANDATORY　　当前一定要有事务，不然就抛异常
4. REQUIRES_NEW　　无论是否有事务，都起个新的事务
5. NOT_SUPPORTED　　不支持事务，按非事务方式运行
6. NEVER　　不支持事务，如果有事务则抛异常
7. NESTED　　当前有事务就在当前事务里再起一个事务