Spring中事务嵌套这么用一定得注意了！！

Original JAVA旭阳 JAVA旭阳 2023-04-22 08:50 Posted on 福建

前言

最近项目上有一个使用事务相对复杂的业务场景报错了。在绝大多数情况下，都是风平浪静，没有问题。其实内在暗流涌动，在有些异常情况下就会报错，这种偶然性的问题很有可能就会在暴露到生产上造成事故，那究竟是怎么回事呢？

问题描述

我们用一个简单的例子模拟下，大家也可以看看下面这段代码输出的结果是什么。

1. 在类 `SecondTransactionService` 定义一个简单接口 `transaction2` ，插入一个用户，同时必然会抛出错误

```
@Override
@Transactional(rollbackFor = Exception.class)
public void transaction2() {
    System.out.println("do transaction2.....");
    User user = new User("tx2", "111", 18);
    // 插入一个用户
    userService.insertUser(user);
    // 跑错了
    throw new RuntimeException();
}
```

2. 在另外一个类 `FirstTransactionService` 定义一个接口 `transaction1` ，它调用 `transaction2` 方法，同时做了 `try catch` 处理

```
@Override
@Transactional(rollbackFor = Exception.class)
public void transaction1() {
    System.out.println("do transaction1 .......");
    try {
        // 调用另外一个事务，try catch住
        secondTransactionService.transaction2();
    } catch (Exception e) {
        e.printStackTrace();
    }

    // 插入当前用户tx1
    User user = new User("tx1", "111", 18);
    userService.insertUser(user);
}
```

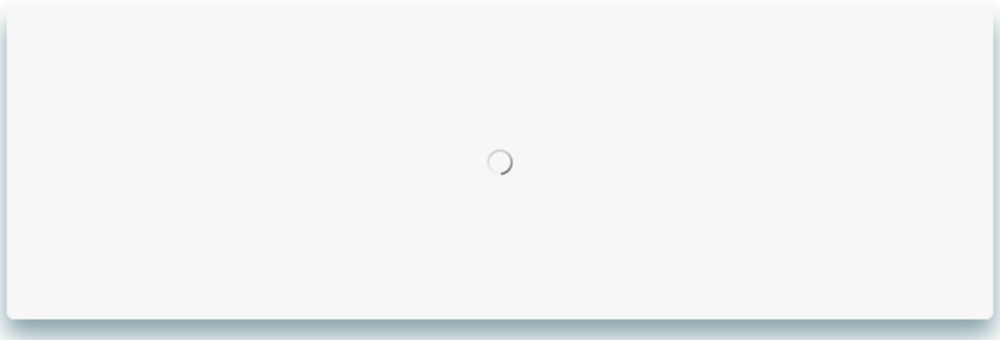
3. 定义一个 `controller` ，调用 `transaction1` 方法

```
@GetMapping("/testNestedTx")
public String testNestedTx() {
    firstTransactionService.transaction1();
    return "success";
}
```

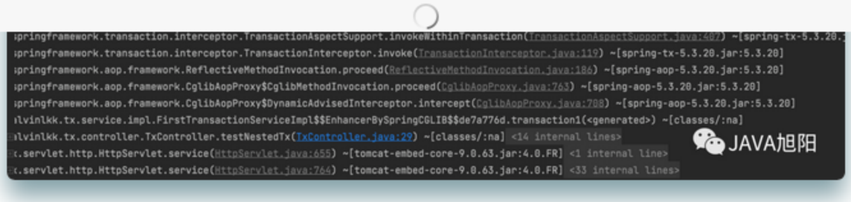
大家觉得调用这个 `http` 接口，最终数据库插入的是几条数据呢？

问题结果

正确答案是数据库插入了0条数据。



同时控制台也报错了，报错原因是：`org.springframework.transaction.UnexpectedRollbackException: Transaction rolled back because it has been marked as rollback-only`



是否和你预想的一样呢？你知道是为什么吗？

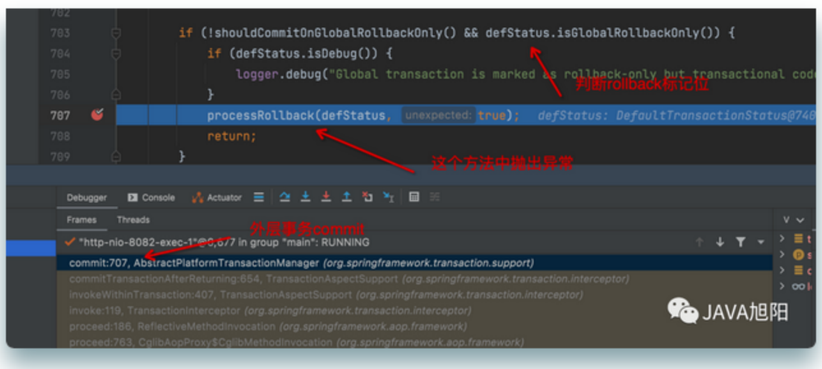
原因追溯

其实原因很简单，我们都知道，一个事务要么全成功提交事务，要么失败全部回滚。如果出现在一个事务中部分SQL要回滚，部分SQL要提交，这不就主打的一个”前后矛盾，精神分裂“吗？

```
controller.testNestedTx()
  ||
  /
FirstTransactionService.transaction1()   REQUIRED隔离级别
       ||
       ||
       || 捕获异常，提交事务，出错啦
       / ||
FirstTransactionService.transaction2()   REQUIRED隔离级别
       || ||
       || 抛出异常,标记事务为rollback only
       ======================
```

1. 事务的隔离级别为 `REQUIRED` ，那么发现没有事务开启一个事务操作，有的话，就合并到这个事务中，所以 `transaction1()` 、 `transaction2()` 是在同一个事务中。
2. `transaction2()` 抛出异常，那么事务会被标记为 `rollback only` ，源码如下所示：

3. `transaction1()` 由于 `try catch` 异常，正常运行，想必就要可以提交事务了，在提交事务的时



这下，是不是很清楚知道报错的原因了，那想想该怎么处理呢？

解决之道

知道了根本原因之后，是不是解决的方案就很明朗了，我们可以通过调整事务的传播方式分拆多个事务管理，或者让一个事务"前后一致"，做一个诚信的好事务。

- 将 `try catch` 放到内层事务中，也就是 `transaction2()` 方法中，这样内层事务会跟着外部事务进行提交或者回滚。

```
@Override
    @Transactional(rollbackFor = Exception.class)
    public void transaction2() {
        try {
            System.out.println("do transaction2.....");
            User user = new User("tx2", "111", 18);
            userService.insertUser2(user);
            throw new RuntimeException();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

- 如果希望内层事务抛出异常时中断程序执行，直接在外层事务的 `catch` 代码块中抛出 `e` ，这样同一个事务就都会回滚。
- 如果希望内层事务回滚，但不影响外层事务提交，需要将内层事务的传播方式指定为 `PROPAGATION_NESTED` 。 `PROPAGATION_NESTED` 基于数据库 `savepoint` 实现的嵌套事务，外层事务的提交和回滚能够控制嵌内层事务，而内层事务报错时，可以返回原始 `savepoint` ，外层事务可以继续提交。

```
public void transaction2() {
    System.out.println("do transaction2.....");
    User user = new User( userName: "tx2",  password: "111",  age: 18);
```


事务的传播机制

前面提到了事务的传播机制，我们再看都有哪几种。

- `PROPAGATION_REQUIRED`：加入到当前事务中，如果当前没有事务，就新建一个事务。这是最常见的选择，也是Spring中默认采用的方式。
- `PROPAGATION_SUPPORTS`：支持当前事务，如果当前没有事务，就以非事务方式执行。
- `PROPAGATION_MANDATORY`：支持当前事务，如果当前没有事务，就抛出异常。
- `PROPAGATION_REQUIRES_NEW`：新建一个事务，如果当前存在事务，把当前事务挂起。
- `PROPAGATION_NOT_SUPPORTED`：以非事务方式执行操作，如果当前存在事务，就把当前事务挂起。
- `PROPAGATION_NEVER`：以非事务方式执行，如果当前存在事务，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，则进行与`PROPAGATION_REQUIRED`类似的操作。

如何理解`PROPAGATION_NESTED`的传播机制呢，和`PROPAGATION_REQUIRES_NEW`又有什么区别呢？我们用一个例子说明白。

- 定义`serviceA.methodA()`以`PROPAGATION_REQUIRED`修饰;
- 定义`serviceB.methodB()以`表格中三种方式修饰;
- `methodA`中调用`methodB`;

异常状态	PROPAGATION_REQUIRES_NEW *(两个独立事务)*	PROPAGATION_NESTED *(B的事务嵌套在A的事务中)*	PROPAGATION_REQUIRED *(同一个事务)*
methodA抛异常 methodB正常	A回滚，B正常提交	A与B一起回滚	A与B一起回滚
methodA正常 methodB抛异常	1.如果A中捕获B的异常，并没有继续向上抛异常，则B先回滚，A再正常提交； 2.如果A未捕获B的异常，默认则会将B的异常向上抛，则B先回滚，A再回滚	B先回滚，A再正常提交	A与B一起回滚
methodA抛异常 methodB抛异常	B先回滚，A再回滚	A与B一起回滚	A与B一起回滚
methodA正常 methodB正常	B先提交，A再提交	A与B一起提交	A与B一起提交

总结

在我的项目中之所以会报“`rollback-only`”异常的根本原因是代码风格不一致的原因。外层事务对错误的处理方式是返回true或false来告诉上游执行结果，而内层事务是通过抛出异常来告诉上游（这里指外层事务）执行结果，这种差异就导致了“`rollback-only`”异常。大家也可以去review自己项目中的代码，是不是也偷偷犯下同样的错误了。

如果觉得这篇文章对你有所帮助，还请帮忙点赞、在看、转发给更多的人，非常感谢！

欢迎点击关注公众号，更多技术干货及时获得。

JAVA旭阳
旭阳，希望自己能像初升的太阳一样，对任何事情充满希望~~
142篇原创内容
公众号

收录于合集 #Spring 19

上一篇 · SpringBoot项目中使用缓存Cache的正确姿势！！！