18个示例详解 Spring 事务传播机制

发布于 2023-02-25 20:50:34 1.6K 0

文章被收录于专栏： 码出code

什么是事务传播机制

事务的传播机制，顾名思义就是多个事务方法之间调用，事务如何在这些方法之间传播。

> 举个例子，方法 A 是一个事务的方法，方法 A 执行的时候调用了方法 B,此时方法 B 有无事务以及是否需要事务都会对方法 A 和方法 B 产生不同的影响，而这个影响是由事务传播机制决定的。

传播属性 Propagation 枚举

Spring 对事务的传播机制在 Propagation 枚举中定义了7个分类：

REQUIRED 默认

SUPPORTS 支持

MANDATORY 强制

REQUIRES_NEW 新建

NOT_SUPPORTED 不支持

NEVER 从不

NESTED 嵌套

事务的传播机制，是 spring 规定的。因为在开发中，最简单的事务是，业务代码都处于同一个事务下，这也是默认的传播机制，如果出现的报错，所有的数据回滚。但是在逻辑时，方法之间的调用，有以下的需求：

调用的方法需要新增一个事务，新事务和原来的事务各自独立。

调用的方法不支持事务

调用的方法是一个嵌套的事务

7种传播机制详解

首先创建两个方法 A 和 B 实现数据的插入，插入数据A:

代码语言：javascript

```
public class AService {
    public void A(String name) {
        userService.insertName("A-" + name);
    }

}
```

插入数据B:

代码语言：javascript

```
public class BService {
    public void B(String name) {
        userService.insertName("B-" + name);
    }

}
```

使用伪代码创建 mainTest 方法和 childTest 方法

代码语言：javascript

```
    public void mainTest(String name)  {
         // 存入a1
         A(a1);
         childTest(name);
     }
```

main 调用 test 方法，其中

代码语言：javascript

```
    public void childTest(String name)  {
         // 存入b1
         B(b1);
        throw new RuntimeException();
        // 存入 b2
        B2(b2);
     }
```

以上伪代码，调用 mainTest 方法，如果mainTest 和childTest **都不使用事务**的话， 数据存储 的结果是如何呢?

因为都没使用事务，所以 a1 和 b1 都存到成功了，而之后抛出异常之后，b2是不会执行的。所以 a1 和 b1 都插入的数据，而 b2 没有插入数据。

REQUIRED（默认事务）

代码语言：javascript

```
      /**
       * Support a current transaction, create a new one if none exists.
       * Analogous to EJB transaction attribute of the same name.
       * <p>This is the default setting of a transaction annotation.
       */
      REQUIRED(TransactionDefinition.PROPAGATION_REQUIRED),
```

如果当前不存在事务，就新建一个事务。如果存在事务，就加入到当前事务。这是一个默认的事务。

示例1：根据场景举个例子，在 childTest 添加事务，设置传播属性为 REQUIRED，伪代码如下：

代码语言：javascript

```
    public void mainTest(String name)  {
         // 存入a1
         A(a1);
         childTest(name);
     }
    @Transactional(propagation = Propagation.REQUIRED)
    public void childTest(String name)  {
         // 存入b1
         B(b1);
        throw new RuntimeException();
     }
```

因为 mainTest 没有事务，而 childTest 又是新建一个事务，所以 a1 添加成功。在 childTest 因为抛出了异常，不会执行 b2 添加，而 b1 添加回滚。最终 a1 添加成功，b1没

示例2：在 mainTest 和 childTest 都添加事务，传播属性都为 REQUIRED，伪代码如下：

代码语言：javascript

```
    @Transactional(propagation = Propagation.REQUIRED)
    public void mainTest(String name) {
         // 存入a1
         A(a1);
         childTest(name);
     }
    @Transactional(propagation = Propagation.REQUIRED)
    public void childTest(String name) {
         // 存入b1
         B(b1);


```

```
        throw new RuntimeException();
    }
```

根据 REQUIRED 传播属性，如果存在事务，就加入到当前事务。两个方法都属于同一个事务，同一个事务的话，如果有发生异常，则全部都回滚。所以 a1 和 b1 都没添加成

SUPPORTS

代码语言：javascript

```
/**
     * Support a current transaction, execute non-transactionally if none exists.
     * Analogous to EJB transaction attribute of the same name.
     * <p>Note: For transaction managers with transaction synchronization,
     */
```

如果当前没有事务，则以非事务的方式运行。如果存在事务，就加入到当前事务。

示例3：childTest 添加事务，传播属性设置为 SUPPORTS，伪代码如下：

代码语言：javascript

```
    public void mainTest(String name) {
         A(a1);
         childTest(name);
     }
    @Transactional(propagation = Propagation.SUPPORTS)
    public void childTest(String name) {
         B(b1);
        throw new RuntimeException();
     }
```

传播属性为 SUPPORTS，如果没有事务，就以非事务的方式运行。表明两个方法都没有使用事务，没有事务的话，a1、b1 都添加成功。

示例4：mainTest 添加事务，设置传播属性为 REQUIRED。childTest 添加事务，设置传播属性为 SUPPORTS，伪代码如下：

代码语言：javascript

```
    @Transactional(propagation = Propagation.REQUIRED)
    public void mainTest(String name) {
         A(a1);
         childTest(name);
     }
    @Transactional(propagation = Propagation.SUPPORTS)
    public void childTest(String name) {
         B(b1);
        throw new RuntimeException();
        B2(b2);
     }
```

SUPPORTS 传播属性，如果存在事务，就加入到当前事务。mainTest 和 childTest 都属于同一个事务，而 childTest 抛出异常，a1 和b1 添加都回滚，最终 a1、b1 添加失败

MANDATORY

代码语言：javascript

```
/**
     * Support a current transaction, throw an exception if none exists.
     * Analogous to EJB transaction attribute of the same name.
     */
```

如果存在事务，就加入到当前事务。如果不存在事务，就报错。这就说明如果想调用 MANDATORY 传播属性的方法，一定要有事务，不然就会报错。

> MANDATORY 类似功能限制，必须要被有事务的方法的调用，不然就会报错。

示例5：首先在 childTest 添加事务，设置传播属性为 MANDATORY，伪代码如下：

代码语言：javascript

```
    public void mainTest(String name) {
         A(a1);
         childTest(name);
     }
    @Transactional(propagation = Propagation.MANDATORY)
    public void childTest(String name) {
         B(b1);
        throw new RuntimeException();
        B2(b2);
     }
```

在控制台直接报错：

代码语言：javascript

```
No existing transaction found for transaction marked with propagation 'mandatory'
```

说明被标记为 mandatory 传播属性没找到事务，直接报错。因为 mainTest 没有事务，a1 添加成功。而 childTest 由于报错，b1 添加失败。

示例6：mainTest 添加事务，设置传播属性为 REQUIRED。childTest 添加事务，设置传播属性为 MANDATOR，伪代码如下：

代码语言：javascript

```
    @Transactional(propagation = Propagation.REQUIRED)
    public void mainTest(String name) {
         A(a1);
         childTest(name);
     }
    @Transactional(propagation = Propagation.MANDATORY)
    public void childTest(String name) {
         B(b1);
        throw new RuntimeException();
     }
```

如果存在事务，就把事务加入到当前事务。同一个事务中 childTest 抛出异常，a1 和 b1 添加被回滚，所以a1 和 b1添加失败。

REQUIRES_NEW

代码语言：javascript

```
    /**
      * Create a new transaction, and suspend the current transaction if one exists.
      * Analogous to the EJB transaction attribute of the same name.
      * <p><b>NOTE:</b> Actual transaction suspension will not work out-of-the-box
      */
```

创建一个新的事务。如果存在事务，就将事务挂起。

> 无论是否有事务，都会创建一个新的事务。

示例7：childTest 添加事务，设置传播属性为 REQUIRES_NEW，伪代码如下：

代码语言：javascript

```
    public void mainTest(String name) {
         A(a1);
         childTest(name);
     }
    @Transactional(propagation = Propagation.REQUIRES_NEW)
    public void childTest(String name) {
         B(b1);
        throw new RuntimeException();
     }
```

mainTest 不存在事务，a1 添加成功，childTest 新建了一个事务，报错，回滚 b1。所以 a1 添加成功，b1 添加失败。

示例8：mainTest 添加事务，设置传播属性为 REQUIRED。childTest 添加事务，设置传播属性为 REQUIRES_NEW，伪代码如下：

代码语言：javascript

```
    @Transactional(propagation = Propagation.REQUIRED)
    public void mainTest(String name) {
         A(a1);
         childTest(name);
     }
    @Transactional(propagation = Propagation.REQUIRES_NEW)
    public void childTest(String name) {
         B(b1);
        throw new RuntimeException();
     }
```

mainTest 创建了一个事务，childTest 新建一个事务，在 childTest 事务中，抛出异常，b1 回滚，异常抛到 mainTest 方法，a1 也回滚，最终 a1 和 b1 都回滚。

示例9：在**示例8**中，如果不想让 REQUIRES_NEW 传播属性影响到被调用事务，将异常捕获就不会影响到被调用事务。

代码语言：javascript

```
    @Transactional(propagation = Propagation.REQUIRED)
    public void mainTest(String name) {
         A(a1);
         try {
                childTest(name);
         } catch (Exception e) {
             e.printStackTrace();
         }
     }
    @Transactional(propagation = Propagation.REQUIRES_NEW)
    public void childTest(String name) {
         B(b1);
        throw new RuntimeException();
     }
```

childTest 抛出了异常，在 mainTest 捕获了，对 mainTest 没有影响，所以 b1 被回滚，b1 添加失败，a1 添加成功。

示例10：mainTest 设置传播属性为 REQUIRED，并在 mainTest 抛出异常。childTest 同样设置 REQUIRES_NEW 传播属性，伪代码如下：

代码语言：javascript

```
    @Transactional(propagation = Propagation.REQUIRED)
    public void mainTest(String name) {
         A(a1);
         childTest(name);
         throw new RuntimeException();
     }
    @Transactional(propagation = Propagation.REQUIRES_NEW)
    public void childTest(String name) {
         B(b1);
         B2(b2);
     }
```

childTest 是一个新建的事务，只要不抛出异常是不会回滚，所以 b1 添加成功，而 mainTest 抛出了异常，a1 添加失败。

> REQUIRES_NEW 传播属性如果有异常，只会从被调用方影响调用方，而调用方不会影响调用方，即 childTest 抛出异常会影响 mainTest，而 mainTest 抛出异常不会到

NOT_SUPPORTED

代码语言：javascript

```
   /**
      * Execute non-transactionally, suspend the current transaction if one exists.
      * Analogous to EJB transaction attribute of the same name.
      * <p><b>NOTE:</b> Actual transaction suspension will not work out-of-the-box
      */
```

无论是否存在当前事务，都是以非事务的方式运行。

示例11：childTest 添加事务，设置传播属性为 NOT_SUPPORTED，伪代码如下：

代码语言：javascript

```
    public void mainTest(String name) {
         A(a1);
         childTest(name);

     }
    @Transactional(propagation = Propagation.NOT_SUPPORTED)
    public void childTest(String name) {
         B(b1);
        throw new RuntimeException();
     }
```

NOT_SUPPORTED 都是以非事务的方式执行，childTest 不存在事务，b1 添加成功。而 mainTest 也是没有事务，a1 也添加成功。

示例12：childTest 添加事务，设置传播属性为 NOT_SUPPORTED，mainTest 添加默认传播属性 REQUIRED，伪代码如下：

代码语言：javascript

```
    @Transactional(propagation = Propagation.REQUIRED)
    public void mainTest(String name) {
         A(a1);
         childTest(name);

     }
    @Transactional(propagation = Propagation.NOT_SUPPORTED)
    public void childTest(String name) {
         B(b1);
        throw new RuntimeException();
      }
```

其中 childTest 都是以非事务的方式执行，b1 添加成功。而 mainTest 存在事务，报错后回滚，a1 添加失败。

NEVER

代码语言：javascript

```
    /**
      * Execute non-transactionally, throw an exception if a transaction exists.
      * Analogous to EJB transaction attribute of the same name.
      */
```

不使用事务，如果存在事务，就抛出异常。

> NEVER 的方法不使用事务，调用 NEVER 方法如果有事务，就抛出异常。

示例13：childTest 添加 NEVER 传播属性，伪代码如下：

代码语言：javascript

```
    public void mainTest(String name) {
         A(a1);
         childTest(name);

     }
    @Transactional(propagation = Propagation.NEVER)
    public void childTest(String name) {
         B(b1);
        throw new RuntimeException();
        B2(b2);
     }
```

NEVER 不使用事务，mainTest 也不使用事务，所以 a1 和 b1 都添加成功，b2添加失败。

示例14：mainTest 添加 REQUIRED 传播属性，childTest 传播属性设置为 NEVER，伪代码如下：

代码语言：javascript

```
    @Transactional(propagation = Propagation.REQUIRED)
    public void mainTest(String name) {
         A(a1);
         childTest(name);

     }
    @Transactional(propagation = Propagation.NEVER)
    public void childTest(String name) {
         B(b1);
        throw new RuntimeException();
     }
```

mainTest 存在事务，导致 childTest 报错，b1添加失败，childTest 抛错到 mainTest，a1 添加失败。

NESTED

代码语言：javascript

```
/**
     * Execute within a nested transaction if a current transaction exists,
     * behave like PROPAGATION_REQUIRED else. There is no analogous feature in EJB.
     * <p>Note: Actual creation of a nested transaction will only work on specific
     */
```

如果当前事务存在，就运行一个嵌套事务。如果不存在事务，就和 REQUIRED 一样新建一个事务。

示例15: childTest 设置 NESTED 传播属性，伪代码如下：

代码语言：javascript

```
    public void mainTest(String name) {
         A(a1);
         childTest(name);

     }
    @Transactional(propagation = Propagation.NESTED)
    public void childTest(String name) {
         B(b1);
        throw new RuntimeException();
     }
```

在 childTest 设置 NESTED 传播属性，相当于新建一个事务，所以 b1 添加失败， mainTest 没有事务，a1 添加成功。

示例16：设置 mainTest 传播属性为 REQUIRED，新建一个事务，并在方法最后抛出异常。childTest 设置属性为 NESTED，伪代码如下：

代码语言：javascript

```
    @Transactional(propagation = Propagation.REQUIRED)
    public void mainTest(String name) {
         A(a1);
         childTest(name);
         throw new RuntimeException();
     }
    @Transactional(propagation = Propagation.NESTED)
    public void childTest(String name) {
         B(b1);
        B2(b2);
     }
```

childTest 是一个嵌套的事务，当主事务的抛出异常时，嵌套事务也受影响，即 a1、b1 和 b2 都添加失败。和**示例10**不同的是，**示例10**不会影响 childTest 事务。

NESTED 和 REQUIRED_NEW 的区别：

- REQUIRED_NEW 是开启一个新的事务，和调用的事务无关。调用方回滚，不会影响到 REQUIRED_NEW 事务。
- NESTED 是一个嵌套事务，是调用方的一个子事务，如果调用方事务回滚，NESTED 也会回滚。

示例17：和**示例16**一样，在 mainTest 设置传播属性为 REQUIRED，childTest 设置传播属性为 NESTED，不同的是，在 mainTest 捕获 childTest 抛出的异常，伪代码如下：

代码语言：javascript

```
    @Transactional(propagation = Propagation.REQUIRED)
    public void mainTest(String name) {
         A(a1);
        try {
             childTest(name);
         } catch (RuntimeException e) {
             e.printStackTrace();
         }
     }
    @Transactional(propagation = Propagation.NESTED)
    public void childTest(String name) {
        B(b1);
        B2(b2);
        throw new RuntimeException();
     }
```

childTest 是一个子事务，报错回滚，b1 和 b2 都添加失败。而 mainTest 捕获了异常，不受异常影响，a1 添加成功。

示例18：将**示例2**改造一下，mainTest 捕获 childTest 抛出的异常，伪代码如下：

代码语言：javascript

```
    @Transactional(propagation = Propagation.REQUIRED)
    public void mainTest(String name) {
         A(a1);
        try {
             childTest(name);
         } catch (RuntimeException e) {
             e.printStackTrace();
         }
     }
    @Transactional(propagation = Propagation.REQUIRED)
    public void childTest(String name) {
        B(b1);
        B2(b2);
        throw new RuntimeException();
     }
```

mainTest 和 childTest 两个方法都处于同一个事务，如果有一个方法报错回滚，并且被捕获。整个事务如果还有数据添加就会抛出 `Transaction rolled back because it has been marked as rollback-only` 异常，同一个事务，不能出现有的回滚了，有的不回滚，要么一起回滚，要不一起以全部数据都添加失败。

对比**示例17**和**示例18**，NESTED 和 REQUIRED 的区别：

- REQUIRED 传播属性表明调用方和被调用方都是使用同一个事务，被调用方出现异常，无论异常是否被捕获，因为属于同一个事务，只要发生异常，事务都会回滚。
- NESTED 被调用方出现异常，只要异常被捕获，只有被调用方事务回滚，调用方事务不受影响。

总结

传播属性	总结
REQUIRED	默认属性，所有的事务都处于同一个事务下，出现异常，不管是否捕获所有事务回滚
SUPPORTS	如果不存事务，就以非事务的方式运行，存在事务就加入该事务
MANDATORY	强制调用方添加事务，如果不存在事务就报错，存在事务就加入该事务
REQUIRES_NEW	无论调用方是否存在事务，都会创建新的事务，并且调用方异常不会影响 REQUIRES_NEW事务
NOT_SUPPORTED	无论是否调用方是否存在事务，都是以非事务的方式执行，出现异常也会回滚
NEVER	不用事务，存在事务就报错，和 MANDATORY 相反
NESTED	嵌套事务，新建一个子事务，事务执行相互独立，如果调用方出现异常，直接回滚

测试源码

传播属性源码

参考

带你读懂Spring 事务——事务的传播机制

spring es nested 事务 异常