**声明式事务**

在Spring中，声明式事务是用事务参数来定义的。一个事务参数就是对事务策略应该如何应用到某个方法的一段描述，如下图所示一个事务参数共有5个方面组成：

**传播行为（是spring独有的概念）**

事务的第一个方面是传播行为。传播行为定义关于客户端和被调用方法的事务边界。

此处需要注意一下，如

M1(){}

@trasactional

M2 (){}

若果m1调用了m2方法，m2 的事务传播行为由其本身的@trasactional决定，同时不同传播行为对不含有调用行为的时候也有一定影响

Spring定义了7中传播行为。

以下实例中均为m1调用m2

| **传播行为** | **意义** |
| --- | --- |
| PROPAGATION\_MANDATORY（非常用） | 表示该方法必须运行在一个事务中。如果当前没有事务正在发生，将抛出一个异常 |
| PROPAGATION\_NEVER（不常用） | 表示当前的方法不应该在一个事务中运行。如果一个事务正在进行，则会抛出一个异常。 |
| PROPAGATION\_NOT\_SUPPORTED | 表示该方法不应该在一个事务中运行。如果一个现有事务正在进行中，它将在该方法的运行期间事务失效，同时也不会向上抛出异常。  M2独立调用时不具有事务  m1有事务，m2使用该传播特性，m1内部调用容器中的m2,m2抛出出现异常时不会事务回滚， m1也不会回滚，m2的成功与否不会影响m1。  m1无事务时，m2事务失效  即：不支持事务，同时虽然会抛出异常，但是不影响影响外部事务  应用场景：应用在m2业务失败与否无关紧要，并且不影响m1,如记录操作日志 |
| PROPAGATION\_NESTED | 该方法应当运行在一个嵌套式事务中，才会创建事务。  M2独立调用时支持事务  M1有事务时，m2存在异常回滚，m1不会回滚，m1存在异常时m1\m2都回滚  M1 无事务时 m1存在异常时，m1、m2均不回滚，m2存在异常时，m2回滚  即支持事务特性，不影响外部事务，受外部事务影响 |
| PROPAGATION\_SUPPORTS | 表示当前方法不需要事务性上下文，但是如果有一个事务已经在运行的话，它也可以在这个事务里运行。m2使用该传播特性，m1内部调用容器中的m2：  M2独立调用时支持事务  m1有事务，m2存在异常时会事务回滚，即使m2的方法被try catch 时m1也会回滚，但是m1中的异常不会影响m2  m1无事务时，m2存在异常回滚  即支持事务，对外部事务有影响，不受外部事务影响 |
| PROPAGATION\_REQUIRES\_NEW | 无论什么时候都独立新开事务  对外部事务无影响，同时也不受其事务影响 |
| PROPAGATION\_REQUIRES | 表示当前方法可以在一个事务中运行。如果一个现有事务正在进行中，该方法将在那个事务中运行，否则就要开始一个新事务。（spring默认值）  既受外部事务影响，也影响外部事务 |

**隔离级别**

声明式事务的第二个方面是隔离级别。隔离级别定义一个事务可能受其他并发事务活动活动影响的程度。另一种考虑一个事务的隔离级别的方式，是把它想象为那个事务对于事物处理数据的自私程度。

在一个典型的应用程序中，多个事务同时运行，经常会为了完成他们的工作而操作同一个数据。并发虽然是必需的，但是会导致一下问题：

* 脏读（Dirty read）-- 脏读发生在一个事务读取了被另一个事务改写但尚未提交的数据时。如果这些改变在稍后被回滚了，那么第一个事务读取的数据就会是无效的。
* 不可重复读（Nonrepeatable read）-- 不可重复读发生在一个事务执行相同的查询两次或两次以上，但每次查询结果都不相同时。这通常是由于另一个并发事务在两次查询之间更新了数据。
* 幻影读（Phantom reads）-- 幻影读和不可重复读相似。当一个事务（T1）读取几行记录后，另一个并发事务（T2）插入了一些记录时，幻影读就发生了。在后来的查询中，第一个事务（T1）就会发现一些原来没有的额外记录。

在理想状态下，事务之间将完全隔离，从而可以防止这些问题发生。然而，完全隔离会影响性能，因为隔离经常牵扯到锁定在数据库中的记录（而且有时是锁定完整的数据表）。侵占性的锁定会阻碍并发，要求事务相互等待来完成工作。

表格 1spring隔离级别

| **隔离级别** | **含义** |
| --- | --- |
| ISOLATION\_DEFAULT | 使用后端数据库默认的隔离级别。 |
| ISOLATION\_READ\_UNCOMMITTED | 允许读取尚未提交的更改。可能导致脏读、幻影读或不可重复读。 |
| ISOLATION\_READ\_COMMITTED | 允许从已经提交的并发事务读取。可防止脏读，但幻影读和不可重复读仍可能会发生。 |
| ISOLATION\_REPEATABLE\_READ | 对相同字段的多次读取的结果是一致的，除非数据被当前事务本身改变。可防止脏读和不可重复读，但幻影读仍可能发生。 |
| ISOLATION\_SERIALIZABLE | 完全服从ACID的隔离级别，确保不发生脏读、不可重复读和幻影读。这在所有隔离级别中也是最慢的，因为它通常是通过完全锁定当前事务所涉及的数据表来完成的。 |

考虑到完全隔离会影响性能，而且并不是所有应用程序都要求完全隔离，所以有时可以在事务隔离方面灵活处理。因此，就会有好几个隔离级别。

**只读**

声明式事务的第三个特性是它是否是一个只读事务。如果一个事务只对后端数据库执行读操作，那么该数据库就可能利用那个事务的只读特性，采取某些优化 措施。通过把一个事务声明为只读，可以给后端数据库一个机会来应用那些它认为合适的优化措施。由于只读的优化措施是在一个事务启动时由后端数据库实施的， 因此，只有对于那些具有可能启动一个新事务的传播行为（PROPAGATION\_REQUIRES\_NEW、PROPAGATION\_REQUIRED、 ROPAGATION\_NESTED）的方法来说，将事务声明为只读才有意义。

此外，如果使用Hibernate作为持久化机制，那么把一个事务声明为只读，将使Hibernate的flush模式被设置为FLUSH\_NEVER。这就告诉Hibernate避免和数据库进行不必要的对象同步，从而把所有更新延迟到事务的结束。

**事务超时**

为了使一个应用程序很好地执行，它的事务不能运行太长时间。因此，声明式事务的下一个特性就是它的超时。

假设事务的运行时间变得格外的长，由于事务可能涉及对后端数据库的锁定，所以长时间运行的事务会不必要地占用数据库资源。这时就可以声明一个事务在特定秒数后自动回滚，不必等它自己结束。

由于超时时钟在一个事务启动的时候开始的，因此，只有对于那些具有可能启动一个新事务的传播行为（PROPAGATION\_REQUIRES\_NEW、PROPAGATION\_REQUIRED、ROPAGATION\_NESTED）的方法来说，声明事务超时才有意义。

**回滚规则**

事务五边形的对后一个边是一组规则，它们定义哪些异常引起回滚，哪些不引起。在默认设置下，事务只在出现运行时异常（runtime exception）时回滚，而在出现受检查异常（checked exception）时不回滚（这一行为和EJB中的回滚行为是一致的）。

不过，也可以声明在出现特定受检查异常时像运行时异常一样回滚。同样，也可以声明一个事务在出现特定的异常时不回滚，即使那些异常是运行时一场。