链滴

Spring 事务的传播行为

作者：zxd

原文链接：https://ld246.com/article/1496841029139

来源网站：链滴

<h4 id="事务的传播行为和隔离级别-transaction-behavior-and-isolated-level-">事务的传播行为隔离级别[transaction behavior and isolated level]</h4>
Spring 中事务的定义：
一、Propagation ：
　　key 属性确定代理应该给哪个方法增加事务行为。这样的属性最重要的部份是传播行为。有下选项可供使用：
PROPAGATION_REQUIRED--支持当前事务，如果当前没有事务，就新建一个事务。这是最常的选择。
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。
PROPAGATION_NEVER--以非事务方式执行，如果当前存在事务，则抛出异常。
很多人看到事务的传播行为属性都不甚了解，我昨晚看了 j2ee without ejb 的时候，看到这里不了解，甚至重新翻起数据库系统的教材书，但是也没有找到对这个的分析。今天搜索，找到一篇极的分析文章，虽然这篇文章是重点分析 PROPAGATION_REQUIRED 和 PROPAGATION_REQUIRED NESTED 的
解惑 spring 嵌套事务
TransactionDefinition 接口定义**********
<pre><code class="highlight-chroma">**
    int PROPAGATON_REQUIRED = 0;

   **

** **

    int PROPAGATON_SUPPORTS = 1;

    int PROPAGATON_MANDATORY = 2;

** **

    int PROPAGATON_REQUIRES_NEW = 3;

    int PROPAGATON_NOT_SUPPORTED = 4;

    int PROPAGATON_NEVER = 5;

</code></pre>

    int PROPAGATION_NESTED = 6;
</code></pre>
<hr>
在这篇文章里，他用两个嵌套的例子辅助分析，我这里直接引用了。
sample***
<pre><code class="highlight-chroma">_ServiceA {
    void methodA() {
        ServiceB.methodB();
    }

}

ServiceB {

    void methodB() {
    }

}   _
</code></pre>
<hr>
我们这里一个个分析吧
1： PROPAGATION_REQUIRED
加入当前正要执行的事务不在另外一个事务里，那么就起一个新的事务
比如说，ServiceB.methodB 的事务级别定义为 PROPAGATION_REQUIRED, 那么由于执行 ServiceA methodA 的时候，
ServiceA.methodA 已经起了事务，这时调用 ServiceB.methodB，ServiceB.methodB 看到自己已运行在 ServiceA.methodA
的事务内部，就不再起新的事务。而假如 ServiceA.methodA 运行的时候发现自己没有在事务中，就会为自己分配一个事务。
这样，在 ServiceA.methodA 或者在 ServiceB.methodB 内的任何地方出现异常，事务都会被回滚即使 ServiceB.methodB 的事务已经被
提交，但是 ServiceA.methodA 在接下来 fail 要回滚，ServiceB.methodB 也要回滚
2： PROPAGATION_SUPPORTS
如果当前在事务中，即以事务的形式运行，如果当前不再一个事务中，那么就以非事务的形式运行
这就跟平常用的普通非事务的代码只有一点点区别了。不理这个，因为我也没有觉得有什么区别
3： PROPAGATION_MANDATORY
必须在一个事务中运行。也就是说，他只能被一个父事务调用。否则，他就要抛出异常。
4： PROPAGATION_REQUIRES_NEW
这个就比较绕口了。 比如我们设计 ServiceA.methodA 的事务级别为 PROPAGATION_REQUIRED，erviceB.methodB 的事务级别为 PROPAGATION_REQUIRES_NEW，
那么当执行到 ServiceB.methodB 的时候，ServiceA.methodA 所在的事务就会挂起，ServiceB.methodB 会起一个新的事务，等待 ServiceB.methodB 的事务完成以后，
他才继续执行。他与 PROPAGATION_REQUIRED 的事务区别在于事务的回滚程度了。因为 ServiceB methodB 是新起一个事务，那么就是存在
两个不同的事务。如果 ServiceB.methodB 已经提交，那么 ServiceA.methodA 失败回滚，ServiceB

methodB 是不会回滚的。如果 ServiceB.methodB 失败回滚，
如果他抛出的异常被 ServiceA.methodA 捕获，ServiceA.methodA 事务仍然可能提交。
5： PROPAGATION_NOT_SUPPORTED
当前不支持事务。比如 ServiceA.methodA 的事务级别是 PROPAGATION_REQUIRED ，而 ServiceB methodB 的事务级别是 PROPAGATION_NOT_SUPPORTED ，
那么当执行到 ServiceB.methodB 时，ServiceA.methodA 的事务挂起，而他以非事务的状态运行完再继续 ServiceA.methodA 的事务。
6： PROPAGATION_NEVER
不能在事务中运行。假设 ServiceA.methodA 的事务级别是 PROPAGATION_REQUIRED， 而 ServieB.methodB 的事务级别是 PROPAGATION_NEVER ，
那么 ServiceB.methodB 就要抛出异常了。
7： PROPAGATION_NESTED
理解 Nested 的关键是 savepoint。他与 PROPAGATION_REQUIRES_NEW 的区别是，PROPAGATON_REQUIRES_NEW 另起一个事务，将会与他的父事务相互独立，
而 Nested 的事务和他的父事务是相依的，他的提交是要等和他的父事务一块提交的。也就是说，如父事务最后回滚，他也要回滚的。
而 Nested 事务的好处是他有一个 savepoint。
<hr>
<pre><code class="highlight-chroma">ServiceA {
 void methodA() {

     try {
  //savepoint
         ServiceB.methodB();   //PROPAGATION_NESTED 级别
     } catch (SomeException) {
         // 执行其他务, 如 ServiceC.methodC();
     }
 }

}
</code></pre>
<hr>
也就是说 ServiceB.methodB 失败回滚，那么 ServiceA.methodA 也会回滚到 savepoint 点上 ServiceA.methodA 可以选择另外一个分支，比如
ServiceC.methodC，继续执行，来尝试完成自己的事务。
但是这个事务并没有在 EJB 标准中定义。
二、Isolation Level(事务隔离等级):
1、Serializable：最严格的级别，事务串行执行，资源消耗最大；
2、REPEATABLE READ：保证了一个事务不会修改已经由另一个事务读取但未提交（回滚）的据。避免了“脏读取”和“不可重复读取”的情况，但是带来了更多的性能损失。
3、READ COMMITTED:大多数主流数据库的默认事务等级，保证了一个事务不会读到另一个并事务已修改但未提交的数据，避免了“脏读取”。该级别适用于大多数系统。
4、Read Uncommitted：保证了读取过程中不会读取到非法数据。
隔离级别在于处理多事务的并发问题。
我们知道并行可以提高数据库的吞吐量和效率，但是并不是所有的并发事务都可以并发运行，这需要看数据库教材的可串行化条件判断了。
这里就不阐述。
我们首先说并发中可能发生的 3 中不讨人喜欢的事情
1： Dirty reads--读脏数据。也就是说，比如事务 A 的未提交（还依然缓存）的数据被事务 B 读走如果事务 A 失败回滚，会导致事务 B 所读取的的数据是错误的。

2： non-repeatable reads--数据不可重复读。比如事务 A 中两处读取数据-total-的值。在第读的时候，total 是 100，然后事务 B 就把 total 的数据改成 200，事务 A 再读一次，结果就发现，total 竟然就变成 200 了，造成事务 A 数据混乱。
3： phantom reads--幻象读数据，这个和 non-repeatable reads 相似，也是同一个事务中次读不一致的问题。但是 non-repeatable reads 的不一致是因为他所要取的数据集被改变了（比如 total 的数据），但是 phantom reads 所要读的数据的不一致却不是他所要读的数据集改变，而是他的件数据集改变。比如 Select account.id where account.name="ppgogo*",第一次读去了 6 个符合件的 id，第二次读取的时候，由于事务 b 把一个帐号的名字由"dd"改成"ppgogo1"，结果取出来了 7 个数据。
<pre><code class="highlight-chroma"> Dirty reads non-repeatable reads phantom reads
Serializable 不会 不会 不会
REPEATABLE REA 不会 不会 会
READ COMMITTE 不会 会 会
Read Uncommitt d 会 会 会
</code></pre>
三、readOnly
事务属性中的 readOnly 标志表示对应的事务应该被最优化为只读事务。这是一个最优化提示。在一情况下，一些事务策略能够起到显著的最优化效果，例如在使用 Object/Relational 映射工具（如：Hbernate 或 TopLink）时避免 dirty checking（试图“刷新”）。
四、Timeout
在事务属性中还有定义“timeout”值的选项，指定事务超时为几秒。在 JTA 中，这将被简单地递到 J2EE 服务器的事务协调程序，并据此得到相应的解释。