个人资料

蓝精灵lx

访问： 461449次
积分： 10150
等级： BLOG 7
排名： 第1302名

Spring 管理事务(传播特性、隔离级别、readonly)

标签： spring 事务 管理

2016-07-03 23:50 508人阅读 评论(0) 收藏 举报

分类： java（466）

一、事务传播机制（Propagation ）

1、spring使用动态代理来为某个方法自动添加事务，而代理应该给哪个方法增加事务行为，是有传播机制决定的。这个传播机制有哪些属性：

1）REQUIRED

默认的，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看到自己已经运行在ServiceA.methodA的事务内部，就不再起新的事务。而假 ServiceA.methodA运行的时候发现自己没有在事务中，他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）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。

3）MANDATORY
必须在一个事务中运行，如果没有就抛出异常。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）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，ServiceB.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）NOT_SUPPORTED
当前不支持事务，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。比如ServiceA.methodA的PROPAGATION_REQUIRED，而ServiceB.methodB的事务级别是PROPAGATION_NOT_SUPPORTED，那么当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）NEVER

resin (4)
zookeeper (3)
分布式 (1)
mq (6)
jmeter (8)
thrift (6)
hbase (8)
hive (8)

文章存档

2016年12月 (24)
2016年11月 (33)
2016年10月 (27)
2016年09月 (17)
2016年08月 (33)

展开

阅读排行

echarts中横坐标值显示不 (9143)
tomcat7中cookie写入中文 (4919)
JAVA中Long与Integer比较 (3857)
jqgrid——treegrid动态异 (3770)
spring使用aop时需要设置 (3206)
Mybatis 大数据量的批量i (2713)
ldap学习 (2689)
mybatis 对于tinyint 类型 (2651)
virtualbox虚拟机设置双网 (2499)
Communications link fail (2485)

不能在事务中运行，如果当前存在事务，则抛出异常。假设ServiceA.methodA的事务级别是PROPAGATION_REQUIRED，而ServiceB.methodB的事务级别是PROPAGATION_NEVER ，那么ServiceA.methodA中调用ServiceB.methodB时就要抛出异常了。

7）NESTED

理解Nested的关键是savepoint。他与PROPAGATION_REQUIRES_NEW的区别是，PROPAGATION_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，如果父事务最后回滚，他也要回滚的。

而Nested事务的好处是他有一个savepoint。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。

2、配置：

```
[html]
01. <!-- 事务-->
02. <beanidbeanid="transactionManager"class="org.springframework.jdbc.datasource.DataSourceTransact
03. <propertynamepropertyname="dataSource" ref="dataSource" />
04. </bean>
05. <tx:adviceidtx:adviceid="txAdvice" transaction-manager="transactionManager">
06. <tx:attributes>
07. <tx:methodnametx:methodname="add*" propagation="REQUIRED"  read-only="false"/>
08. <tx:methodnametx:methodname="save*" propagation="REQUIRED"read-only="false" />
09. <tx:methodnametx:methodname="update*" propagation="REQUIRED"read-only="false" />
10. <tx:methodnametx:methodname="delete*" propagation="REQUIRED"read-only="false"/>
11. <tx:methodnametx:methodname="*" propagation="REQUIRED" /><!--read-only="true" -->
12. </tx:attributes>
13. </tx:advice>
14.
15. <aop:config>
16. <aop:pointcutidaop:pointcutid="transactionPointCut" expression="execution(*cn.edu.nuc.service..
    (..))" />
17. <aop:advisoradvice-refaop:advisoradvice-ref="txAdvice" pointcut-ref="transactionPointCut"/>
```

```
18.  </aop:config>
```

二、事务隔离级别（Isolation ）

1、这里的事务隔离级别和数据库的隔离级别是一个意思。数据库的隔离级别（MySQL）有4个：

读数据一致性及允许的并发副作用 隔离级别	读数据一致性	脏读	不可重复读	幻读
未提交读（Read uncommitted）	最低级别，只能保证不读取物理上损坏的数据，事务可以看到其他事务没有被提交的数据（脏数据）	是	是	是
已提交读（Read committed）	语句级，事务可以看到其他事务已经提交的数据	否	是	是
可重复读（Repeatable read）	事务级，事务中两次查询的结果相同	否	否	是
可序列化（Serializable）	最高级别，事务级。顺序执行	否	否	否

spring控制事务隔离级别，属性有：

1）DEFAULT （默认）

这是一个PlatfromTransactionManager默认的隔离级别，使用数据库默认的事务隔离级别.另外四个与JDBC的隔离级别相对应。

2）READ_UNCOMMITTED （读未提交）

这是事务最低的隔离级别，它允许另外一个事务可以看到这个事务未提交的数据。这种隔离级别会产生脏读，不可重复读和幻像读。

3）READ_COMMITTED （读已提交）

保证一个事务修改的数据提交后才能被另外一个事务读取。另外一个事务不能读取该事务未提交的数据。这种事务隔离级别可以避免脏读出现，但是可能会出现不可重复读和幻像读。

4）REPEATABLE_READ （可重复读）

这种事务隔离级别可以防止脏读，不可重复读。但是可能出现幻像读。它除了保证一个事务不能读取另一个事务未提交的数据外，还保证了不可重复读

5）SERIALIZABLE（串行化）

这是花费最高代价但是最可靠的事务隔离级别。事务被处理为顺序执行。除了防止脏读，不可重复读外，还避免了幻像读。

2、配置：

```[html]
<!-- 事务-->
<beanidbeanid="transactionManager"class="org.springframework.jdbc.datasource.DataSourceTransact
<propertynamepropertyname="dataSource" ref="dataSource" />
</bean>
<tx:adviceidtx:adviceid="txAdvice" transaction-manager="transactionManager">
<tx:attributes>
<tx:methodnametx:methodname="add*" propagation="REQUIRED"isolation="DEFAULT"/>
<tx:methodnametx:methodname="save*" propagation="REQUIRED" isolation="DEFAULT"/>
<tx:methodnametx:methodname="update*" propagation="REQUIRED" isolation="DEFAULT" />
<tx:method name="delete*"propagation="REQUIRED" isolation="DEFAULT"/>
<tx:methodnametx:methodname="*" propagation="REQUIRED" /><!--read-only="true" -->
</tx:attributes>
</tx:advice>

<aop:config>
<aop:pointcutidaop:pointcutid="transactionPointCut" expression="execution(*cn.edu.nu
(..))" />
<aop:advisoradvice-refaop:advisoradvice-ref="txAdvice" pointcut-ref="transactionPointCut"/>
</aop:config>
```

三、只读事务（readonly）

“只读事务”并不是一个强制选项，它只是一个“暗示”，提示数据库驱动程序和数据库系统，这个事务并不包含更改数据的操作，那么JDBC驱动程序和数据库就有可能根据这种情况对该事务进行一些特定的优化，比方说不安排相应的数据库锁，以减轻事务对数据库的压力，毕竟事务也是要消耗数据库的资源的。

```
[html]
<!-- 事务-->
<beanidbeanid="transactionManager"class="org.springframework.jdbc.datasource.DataSourceTransact
<propertynamepropertyname="dataSource" ref="dataSource" />
</bean>
<tx:adviceidtx:adviceid="txAdvice" transaction-manager="transactionManager">
<tx:attributes>
<tx:method name="add*" propagation="REQUIRED"read-only="false" rollback-
for="java.lang.Exception"/>
<tx:methodnametx:methodname="save*" propagation="REQUIRED"read-only="false" rollback-
for="java.lang.Exception"/>
<tx:methodnametx:methodname="update*" propagation="REQUIRED"read-only="false" rollback-
for="java.lang.Exception"/>
<tx:methodnametx:methodname="delete*" propagation="REQUIRED"read-only="false" rollback-
for="java.lang.Exception"/>
<tx:methodnametx:methodname="*" propagation="REQUIRED" /><!--read-only="true" -->
</tx:attributes>
</tx:advice>

<aop:config>
<aop:pointcutidaop:pointcutid="transactionPointCut" expression="execution(*cn.edu.nu
(..))" />
<aop:advisoradvice-refaop:advisoradvice-ref="txAdvice" pointcut-ref="transactionPoin
</aop:config>
```

顶 踩

0 0

上一篇 linux下压缩与解压(zip、unzip、tar)详解

下一篇 ExecutorService.invokeAny()和ExecutorService.invokeAll()的使用剖析

我的同类文章

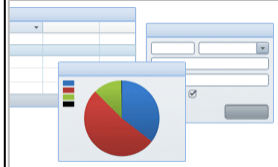
java（466）

- java实现定时任务的三种方法 2016-12-28 阅读 49
- 浅谈Java中CyclicBarrier的用法 2016-12-28 阅读 42
- FileChannel 2016-12-26 阅读 36
- java版“本佛祖保佑永无bug” 2016-12-21 阅读 103
- 通过Spring @PostConstruct … 2016-12-19 阅读 335
- 在web项目启动时，执行某个… 2016-12-15 阅读 79
- spring task 定时任务实现 2016-12-28 阅读 43
- 如何重置一个ArrayList--clear … 2016-12-28 阅读 55
- Java高效读取大文件 2016-12-26 阅读 38
- 图解JDK7的Comparison met… 2016-12-20 阅读 28
- java工厂模式 2016-12-17 阅读 78

更多文章

参考知识库

猜你在找



查看评论

暂无评论

发表评论

用 户 名： andilyliao

评论内容：

提交

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题 Hadoop AWS 移动游戏 Java Android iOS Swift 智能硬件 Docker OpenStack VPN
Spark ERP IE10 Eclipse CRM JavaScript 数据库 Ubuntu NFC WAP jQuery BI HTML5
Spring Apache .NET API HTML SDK IIS Fedora XML LBS Unity Splashtop UML
components Windows Mobile Rails QEMU KDE Cassandra CloudStack FTC coremail OPhone
CouchBase 云计算 iOS6 Rackspace Web App SpringSide Maemo Compuware 大数据 aptech
Perl Tornado Ruby Hibernate ThinkPHP HBase Pure Solr Angular Cloud Foundry Redis
Scala Django Bootstrap