链滴

Spring 简单介绍之 AOP

作者：goker
原文链接：https://ld246.com/article/1606733748443
来源网站：链滴

Spring简单介绍之AOP

概念： 在不修改核心代码的情况下，增加额外的功能。

原理： JDKproxy（需要接口）、cglib（不需要接口）

核心概念点

- Aspect（切面）一个切面对应一个方法
- pointcut（切点）执行事务类的位置
- JoinPoint（连接点）

被拦截的连接点，就是被拦截的方法

- Advice（通知）

执行方法的5个方位

- Weaving（织入）？
- Target（目标对象）

需要织入切面的对象

- Proxy（代理对象）

一. 额外功能

配置<aop:config></aop:config>

切点：expression织语表达式，到要生成动态代理的类。

<aop:pointcut id="" expression="execution(public * com.liuxx.service..*.*(..))"/>

编写额外功能的java类：<bean id="logInfo" class="xxx.xxx.LogInfo">

切面：

```
<aop:aspect  ref=  “logInfo”  >
        <aop:before method="before  “ pointcut-ref="pointcut"/>
        <aop:after method="after  “ pointcut-ref="pointcut"/>
        <aop:after-returning method="afterReturning" pointcut-ref="pointcut" returning="res"
>
        <aop:after-throwing method="afterThrowing" pointcut-ref="pointcut"  throwing="e"/

        <aop:around method="around" pointcut-ref="pointcut"/>
</aop:aspect>
```

二. 事务

1. 配置

```
<aop:config>
      <!-- 执行事务的文件位置 -->
      <aop:pointcut id="txPointcut" expression="execution(public * com.liuxx.service..*.*(..))"/
```

```
    <!-- 执行事务 -->
    <aop:advisor advice-ref="txAdvice" pointcut-ref="txPointcut"/>
</aop:config>

    <!--使用cglib动态代理，默认使用的是jdk-->
    <aop:aspectj-autoproxy proxy-target-class="true"/>

<tx:advice id="txAdvice" transaction-manager="transactionManager">
    <tx:attributes>
        <tx:method name="add*" propagation="REQUIRED"/>
        <tx:method name="append*" propagation="REQUIRED"/>
        <tx:method name="insert*" propagation="REQUIRED"/>
        <tx:method name="save*" propagation="REQUIRED"/>
        <tx:method name="update*" propagation="REQUIRED"/>
        <tx:method name="modify*" propagation="REQUIRED"/>
        <tx:method name="edit*" propagation="REQUIRED"/>
        <tx:method name="delete*" propagation="REQUIRED"/>
        <tx:method name="remove*" propagation="REQUIRED"/>
        <tx:method name="repair" propagation="REQUIRED"/>
        <tx:method name="delAndRepair" propagation="REQUIRED"/>

        <tx:method name="get*" propagation="SUPPORTS" read-only="true"/>
        <tx:method name="find*" propagation="SUPPORTS" read-only="true"/>
        <tx:method name="load*" propagation="SUPPORTS" read-only="true"/>
        <tx:method name="query*" propagation="SUPPORTS" read-only="true"/>
        <tx:method name="search*" propagation="SUPPORTS" read-only="true"/>
        <tx:method name="datagrid*" propagation="SUPPORTS" read-only="true"/>

        <tx:method name="*" propagation="SUPPORTS"/>
    </tx:attributes>
</tx:advice>

<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTra
sactionManager">
    <property name="dataSource" ref="dataSource"/>
</bean>
```

2. 执行事务的方法

```
<tx:method name="" propagation="" ...>
```

(1). 执行事务的方法

name=""; service层执行事务的方法，必须要有：如add*/update*以add开头，update开头

(2). 事务的传播行为（7种）

propagation=""

- required-new；需要新的事务，之前的被挂起
- required；有就使用，无则创建
- support；有也行，没有也行
- no-supports；不支持事务，有就被挂起
- nested；有事务开启内嵌事务，外部依然存在， 无则创建新的

- never；没有事务，有就抛异常
- mandatory； 强制有事务，无就抛异常

(3). 事务的隔离级别

isolation=""

- serializable；一个一个过（串型的）
- read committed； （oracle 默认）读提交，会有不可重复读
- read uncommit；读未提交，会有脏读
- repeatable read； （mysql 默认）可重复读

read-only="true":开启只读事务，用于查询语句

执行多条查询语句，保证一致性；当前条查询的数据被人修改了，会出现数据不一致。

rollback-for:设置哪些异常类回滚

rollback-for="java.lang.Exception" ；遇到异常必须进行回滚

no-rollback-for="java.lang.RuntimeException"；遇到异常不回滚

(4). 数据库并发的问题

1. 脏读

一个事务开始读取了某行数据，但是另外一个事务已经更新了，此数据但没有能够及时提交。可能所操作都被回滚。

解决：设置read commited

2. 幻读

事务在操作过程中进行两次查询，第二次查询结果包含了第一次查询中未出现的数据（这里并不要求次查询SQL语句相同）这是因为在两次查询过程中有另外一个事务插入数据造成的。

3. 不可重复读

一个事务对同一行数据重复读取两次但是却得到了不同结果。例如在两次读取中途有另外一个事务对行数据进行了修改并提交 解决：设置read repeatable

4. 更新丢失

两个事务都同时更新一行数据但是第二个事务却中途失败退出导致对数据两个修改都失效了，这是系没有执行任何锁操作，因此并发事务并没有被隔离开来。

5. 两次更新

无法重复读取特例，有两个并发事务同时读取同一行数据然后其中一个对它进行修改提交而另一个也行了修改提交这就会造成第一次写操作失效。