spring注解-@Transactional基本使用与几点注意

一、@Transactional的使用

步骤一、在xml配置文件中，添加事务管理器bean配置

```
<!-- 事务管理器配置，单数据源事务 -->
    <bean id="pkgouTransactionManager"
        class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="pkGouDataSource" />
    </bean>
<!-- 使用annotation定义事务  开启事务注解 -->
    <tx:annotation-driven transaction-manager="pkgouTransactionManager" />
```

步骤二、注解@Transactional

直接注解在需要的方法或类上。

注解位置说明

1. 不要在接口上声明@Transactional ，而要在具体类的方法上使用 @Transactional 注解，否则注解可能无效。（注解信息不能传递到具体类）
2. 不要图省事，将@Transactional放置在类级的声明中，放在类声明，会使得所有方法都有事务。故@Transactional应该放在方法级别，不需要使用事务的方法，就不要放置事务，比如查询方法。否则对性能是有影响的。

   如果，在类级别和方法级别上都配置了，Spring会按"就近原则"处理。
3. 使用了@Transactional的方法，只能是public，@Transactional注解的方法都是被外部其他类调用才有效，故只能是public。故在 protected、private 或者 package-visible 的方法上使用 @Transactional 注解，它也不会报错，但事务无效。(特别注意)

   ```
   private static class DynamicAdvisedInterceptor implements MethodInterceptor,
   Serializable {
       public Object intercept(Object proxy, Method method, Object[] args, MethodProxy
   methodProxy) throws Throwable {
           ...
            //判断方法是否为public，如果不是，则事务管理注解失效。
           if (chain.isEmpty() && Modifier.isPublic(method.getModifiers())) {
           }
           ...
       }
   }
   ```
4. 使用了@Transactional的方法，同一个类里面的方法调用， @Transactional无效。比如有一个类Test，它的一个方法A，A再调用Test本类的方法B（不管B是否public还是private），但A没有声明注解事务，而B有。则外部调用A之后，B的事务是不会起作用的。（经常在这里出错）

   总之，直接调用加了@Transactional方法，可以回滚；间接调用，不会回滚。

```
class Test(){
    public void A () {
        B();//此时调用B方法属于内部调用，事务失效
    }
    @Transactional
    public void B () {
    }
}
```

5. 在action中加上@Transactional，不会回滚。切记不要在action中加上事务。

设置合适的属性信息

属性名	说明
name	当在配置文件中有多个 TransactionManager , 可以用该属性指定选择哪个事务管理器。
propagation	事务的传播行为，默认值为 REQUIRED。
isolation	事务的隔离度，默认值采用 DEFAULT。
timeout	事务的超时时间，默认值为-1。如果超过该时间限制但事务还没有完成，则自动回滚事务。
read-only	指定事务是否为只读事务，默认值为 false；为了忽略那些不需要事务的方法，比如读取数据，可以设置 read-only 为 true。
rollback-for	用于指定能够触发事务回滚的异常类型，如果有多个异常类型需要指定，各类型之间可以通过逗号分隔。
no-rollback-for	抛出 no-rollback-for 指定的异常类型，不回滚事务。

到此，你发觉使用@Transactional注解管理事务很简单。但是如果对Spring中的@Transactional理解得不够透彻，就很容易出现问题，比如：事务该回滚（rollback）而没有回滚。于是，请往下看关于对Spring的注解方式的事务实现机制。

二、什么情况触发事务

首先，需要知道的是，数据库默认处于**自动提交模式**，即：每条语句都是一个单独的事务。

所以，如果需要使用事务管理（即：一组相关操作处于一个事务中），则必须关闭自动提交模式。而spring在org.springframework.jdbc.datasource.DataSourceTransactionManager.java中已设置自动提交为false。

```
@Override
    protected void doBegin(Object transaction, TransactionDefinition definition) {
        DataSourceTransactionObject txObject = (DataSourceTransactionObject)
transaction;
        Connection con = null;

        try {
            // Switch to manual commit if necessary. This is very expensive in some
JDBC drivers,
```

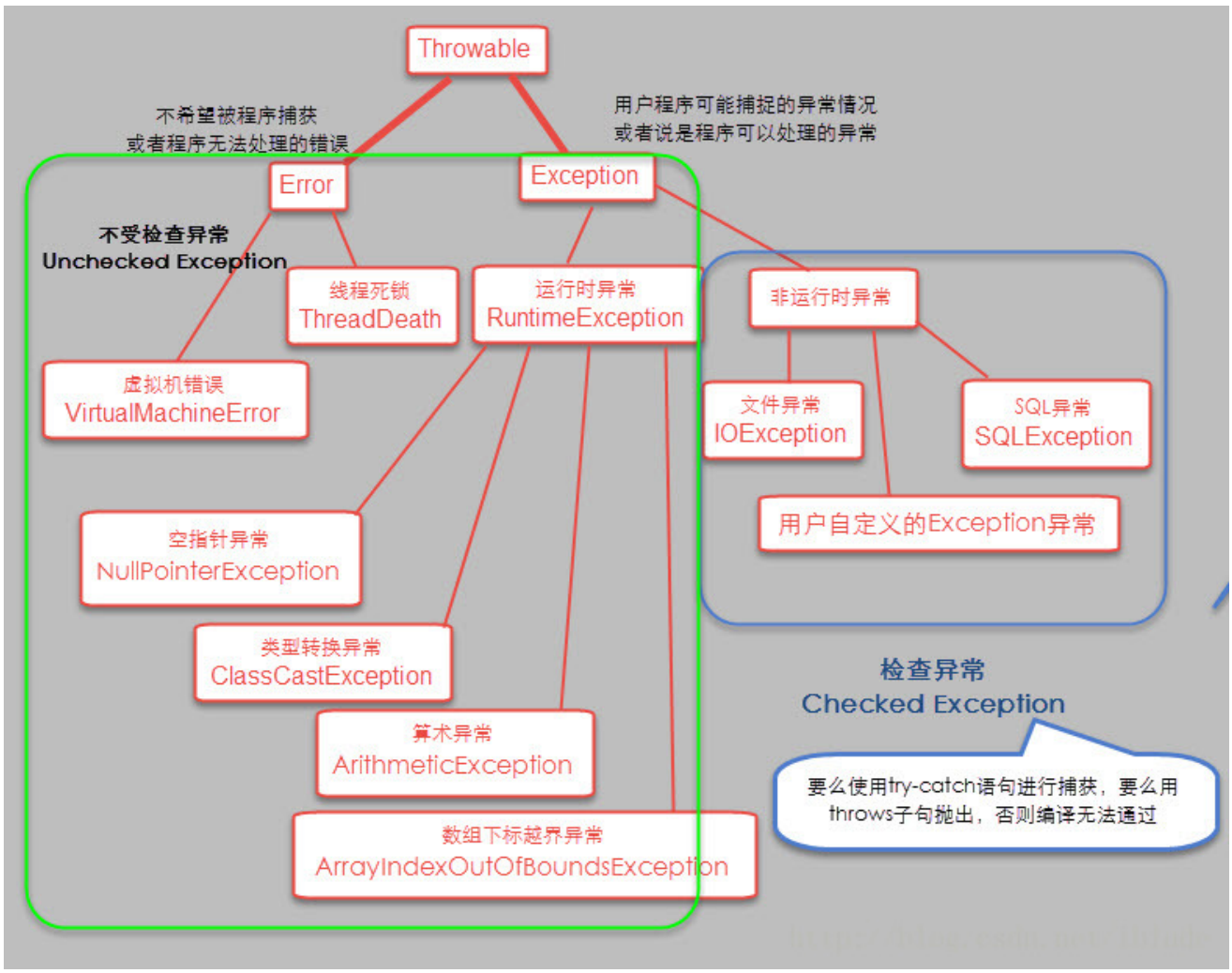
```
            // so we don't want to do it unnecessarily (for example if we've explicitly
            // configured the connection pool to set it already).
            if (con.getAutoCommit()) {
                txObject.setMustRestoreAutoCommit(true);
                if (logger.isDebugEnabled()) {
                    logger.debug("Switching JDBC Connection [" + con + "] to manual
commit");
                }
                  //关闭自动提交
                con.setAutoCommit(false);
            }
        ...
    }
```

但是，什么情况触发事务管理呢？因此，必须知道spring事务回滚规则,也就是说遇到什么异常进行回滚。

默认情况下，运行时unchecked异常时才回滚该事务，即抛出的异常为RuntimeException的子类或Error的子类。而checked异常不会回滚。

unchecked异常（非检查型异常）：RuntimeException的子类或Error的子类

checked异常（检查型异常）：其他，编译器要检查这类异常，编写代码时会有提示。



通常，我们一般都使用默认。

但是，可能有些特殊的情况，所以，Spring支持配置抛出哪些异常时回滚或不回滚。示例如下：

```
//指定回滚，遇到异常Exception时回滚;
@Transactional(rollbackFor=Exception.class)
public void methodName()
{
    throw new Exception("注释");//回滚
    throw new RuntimeException("注释");//回滚
}
//上面结果没有意外，遇到RuntimeException异常时也回滚。
```

此时，如果你以为RuntimeException是Exception的子类的话，可能会对下面的例子感到意外。

```
//指定不回滚
@Transactional(noRollbackFor=Exception.class)
public ItimDaoImpl getItemDaoImpl()
{
    throw new Exception("注释");//不回滚
    throw new RuntimeException("注释");//回滚
}
//意外的是，遇到RuntimeException异常时也回滚
```

Exception异常不回滚是肯定的，但是RuntimeException作为Exception的子类应当也不回滚，但结果却回滚了。

三、开发注意

参考：https://www.cnblogs.com/chenxiaohei/p/8491493.html

A. 一个功能是否要事务，必须纳入设计、编码考虑。

B. 如果加了事务，必须做好开发环境测试（测试环境也尽量触发异常、测试回滚），确保事务生效。

实战时错误：1、间接调用了注解事务的方法（内部调用）

2、方法使用了private修饰符