事务异常类型和回滚操作

使用spring难免要用到spring的事务管理，要用事务管理又会很自然的选择声明式的事务管理，在spring的文档中说道，spring声明式事务管理默认对非检查型异常和运行时异常进行事务回滚，而对检查型异常则不进行回滚操作。

那么什么是检查型异常什么又是非检查型异常呢？

最简单的判断点有两个：

1.继承自runtimeexception或error的是非检查型异常，而继承自exception的则是检查型异常（当然，runtimeexception本身也是exception的子类）。

2.对非检查型类异常可以不用捕获，而检查型异常则必须用try语句块进行处理或者把异常交给上级方法处理总之就是必须写代码处理它。所以必须在service捕获异常，然后再次抛出，这样事务方才起效。

结论：

在spring的事务管理环境下，使用unckecked exception可以极大地简化异常的处理，只需要在事务层声明可能抛出的异常（这里的异常可以是自定义的unckecked exception体系），在所有的中间层都只是需要简单throws即可，不需要捕捉和处理，直接到最高层，比如UI层再进行异常的捕捉和处理

在service类前加上@Transactional，声明这个service所有方法需要事务管理。每一个业务方法开始时都会打开一个事务。

Spring默认情况下会对运行期例外(RunTimeException)进行事务回滚。这个例外是unchecked

如果遇到checked意外就不回滚。

如何改变默认规则：

1 让 checked 例 外 也 回 滚 ： 在 整 个 方 法 前 加 上 @Transactional(rollbackFor=Exception.class)

2 让 unchecked 例 外 不 回 滚 ： @Transactional(notRollbackFor=RunTimeException.class)

3 不 需 要 事 务 管 理 的 （ 只 查 询 的 ） 方 法 ： @Transactional(propagation=Propagation.NOT_SUPPORTED)

注意： 如果异常被try { } catch { } 了，事务就不回滚了，如果想让事务回滚必须再往外抛try { } catch { throw new RuntimeException } 。

一个统一的异常层次结构对于提供服务抽象是必需的。 最重要的就是org.springframework.dao.DataAccessException以及其子类了。 需要强调的是Spring的异常机制重点在于应用编程模型。与SqlException和其他数据存取API不同的是: Spring的异常机制是为了让开发者使用最少,最清晰的代码。DataAccessException和其他底层异常都是非检查性异常(unchecked exception)。 spring的原则之一就是基层异常就应该是非检查性异常. 原因如下:

1. 基层异常通常来说是不可恢复的。

2. 检查性异常将会降低异常层次结构的价值.如果底层异常是检查性的,那么就需要在所有地方添加catch语句进行捕获。

3.try/catch代码块冗长混乱, 而且不增加多少价值。

使用检查异常理论上很好, 但是实际上好象并不如此。

Hibernate3也将从检查性异常转为非检查性异常。