之前看到网上说springBoot项目要用事务，需要在启动类加上@EnableTransactionManagement。

但是我发现我的项目都没加，一样可以用@Transactional注解来控制事务

SpringBoot启动类不加上@EnableTransactionManagement，也是可以用@Transactional注解的，因为：@SpringBootApplication的自动配置已经配置好了。

~/org/springframework/boot/spring-boot-autoconfigure/2.3.0.RELEASE/spring-boot-autoconfigure-2.3.0.RELEASE.jar!/META-INF/spring.factories

```
# Auto Configure
org.springframework.boot.autoconfigure.transaction.TransactionAutoConfiguration,\
```

```
/**
 * {@link org.springframework.boot.autoconfigure.EnableAutoConfiguration
 * Auto-configuration} for Spring transaction.
 *
 * @author Stephane Nicoll
 * @since 1.3.0
 */
@Configuration
@ConditionalOnClass(PlatformTransactionManager.class)
@AutoConfigureAfter({ JtaAutoConfiguration.class,
HibernateJpaAutoConfiguration.class,
        DataSourceTransactionManagerAutoConfiguration.class,
        Neo4jDataAutoConfiguration.class })
@EnableConfigurationProperties(TransactionProperties.class)
public class TransactionAutoConfiguration {

    ......

    @Configuration
    @ConditionalOnBean(PlatformTransactionManager.class)

@ConditionalOnMissingBean(AbstractTransactionManagementConfiguration.class)
    public static class EnableTransactionManagementConfiguration {

        @Configuration
        @EnableTransactionManagement(proxyTargetClass = false) //这里其实已经加了
        @ConditionalOnProperty(prefix = "spring.aop", name = "proxy-target-class",
                havingValue = "false", matchIfMissing = false)
        public static class JdkDynamicAutoProxyConfiguration {
```

```
        }

        @Configuration
        @EnableTransactionManagement(proxyTargetClass = true)  //这里其实已经加了
        @ConditionalOnProperty(prefix = "spring.aop", name = "proxy-target-class",
                havingValue = "true", matchIfMissing = true)
        public static class CglibAutoProxyConfiguration {

        }

    }

}
```

@Transactional参数说明：

参数	功 能
readOnly	该属性用于设置当前事务是否为只读事务，设置为true表示只读，false则表示可读写，默认值为false。例如：@Transactional(readOnly=true)
rollbackFor	该属性用于设置需要进行回滚的异常类数组，当方法中抛出指定异常数组中的异常时，则进行事务回滚。例如：指定单一异常类：@Transactional(rollbackFor=RuntimeException.class)指定多个异常类： 默认是什么:RuntimeExcption 和 Error
rollbackForClassName	该属性用于设置需要进行回滚的异常类名称数组，当方法中抛出指定异常名称数组中的异常时，则进行事务回滚。例如：指定单一异常类名称@Transactional(rollbackForClassName=”RuntimeException”)指定多个异常类名称：@Transactional(rollbackForClassName={“RuntimeException”,”Exception”})
noRollbackFor	该属性用于设置不需要进行回滚的异常类数组，当方法中抛出指定异常数组中的异常时，不进行事务回滚。例如： 指定单一异常类：@Transactional(noRollbackFor=RuntimeException.class) 指定多个异常类：@Transactional(noRollbackFor={RuntimeException.class, Exception.class})
noRollbackForClassName	该属性用于设置不需要进行回滚的异常类名称数组，当方法中抛出指定异常名称数组中的异常时，不进行事务回滚。例如： 指定单一异常类名称：@Transactional(noRollbackForClassName=”RuntimeException”) 指定多个异常类名称：@Transactional(noRollbackForClassName={“RuntimeException”,”Exception”})
propagation	该属性用于设置事务的传播行为。例如：@Transactional(propagation=Propagation.NOT_SUPPORTED)
timeout	该属性用于设置事务的超时秒数，默认值为-1表示永不超时
isolation	该属性用于设置底层数据库的事务隔离级别，事务隔离级别用于处理多事务并发的情况，通常使用数据库的默认隔离级别即可，基本不需要进行设置

事务只读属性

只读事务用于客户代码只读但不修改数据的情形，只读事务用于特定情景下的优化。

事务的回滚条件

可以设置当方法遇到某种异常类型回滚或者不回滚。