事务

1 回顾事务

- 原子性：要么都成功，要么都失败
- 在项目开发中十分重要，涉及到数据的一致性问题
- 确保完整性和一致性
- 事务的ACID原则：
  - 原子性
  - 一致性
  - 隔离性：多个业务可能操作同一个资源，防止数据损坏
  - 持久性：事务一旦提交，无论系统发生什么问题，结果都将不会再被影响，被持久化写到存储器中（如数据库）

2 环境搭建

（和之前的整合mybatis的环境一样）

- 导包（添加依赖）
- 实体类 User
- 接口 UserMapper
- mapper.xml
- 核心配置文件：
  - spring-dao（整合mybatis）
  - mybatis-config（myabtis核心配置文件）
  - applicationContext (Spring核心配置文件)
- 接口实现类 UserMapperImpl
- 测试运行

3 添加几个业务来展示未事务时的样子

- 接口（需求）

```java
/**
 * 增加一个用户
 *
 * @param user
 * @return
 */
public int addUser(User user);

/**
 * 删除一个用户
 *
 * @param id
 * @return
 */
public int deleteUser(int id);
```

- mapper.xml

```
<!--增加用户-->
<insert id="addUser" parameterType="user">
    insert into mybatis.user(id, name, pwd)
    values(#{id},#{name}, #{pwd})
</insert>

<!--删除用户-->
<delete id="deleteUser" parameterType="int">
    deletes from mybatis.user where id = #{id}
</delete>
```

这里我们故意写错delete语句，为的就是对比未添加事务的效果。

- 接口实现类

```
@Override
public List<User> getUsers() {
    UserMapper mapper = getSqlSession().getMapper(UserMapper.class);

    User user = new User(2, "小东", "4141d1e2");
    mapper.addUser(user);

    mapper.deleteUser(2);

    return mapper.getUsers();
}

@Override
public int addUser(User user) {
    return getSqlSession().getMapper(UserMapper.class).addUser(user);
}

@Override
public int deleteUser(int id) {
    return getSqlSession().getMapper(UserMapper.class).deleteUser(id);
}
```

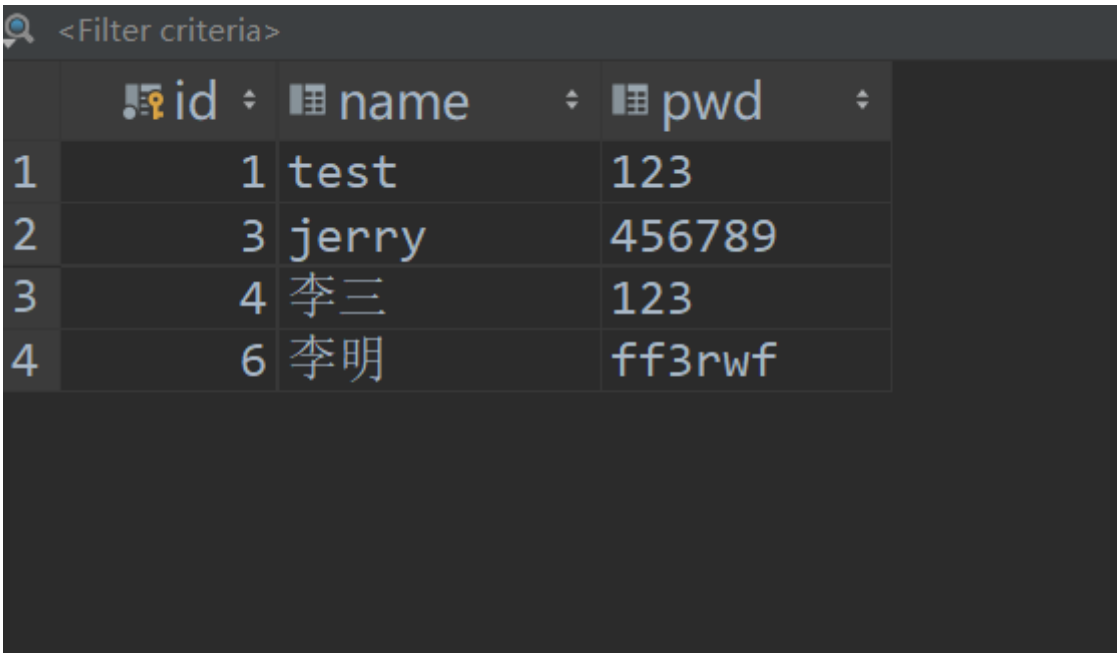
这里我们在获取用户这个需求里，过程中加上 添加用户 和 刚才写错的删除用户的需求。

- 测试运行

  执行获取用户的需求。

```java
@Test
public void getUsers(){
    ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

    UserMapper userMapper = context.getBean("userMapper", UserMapper.class);

    List<User> users = userMapper.getUsers();

    for (User user : users) {
        System.out.println(user);
    }
}
```

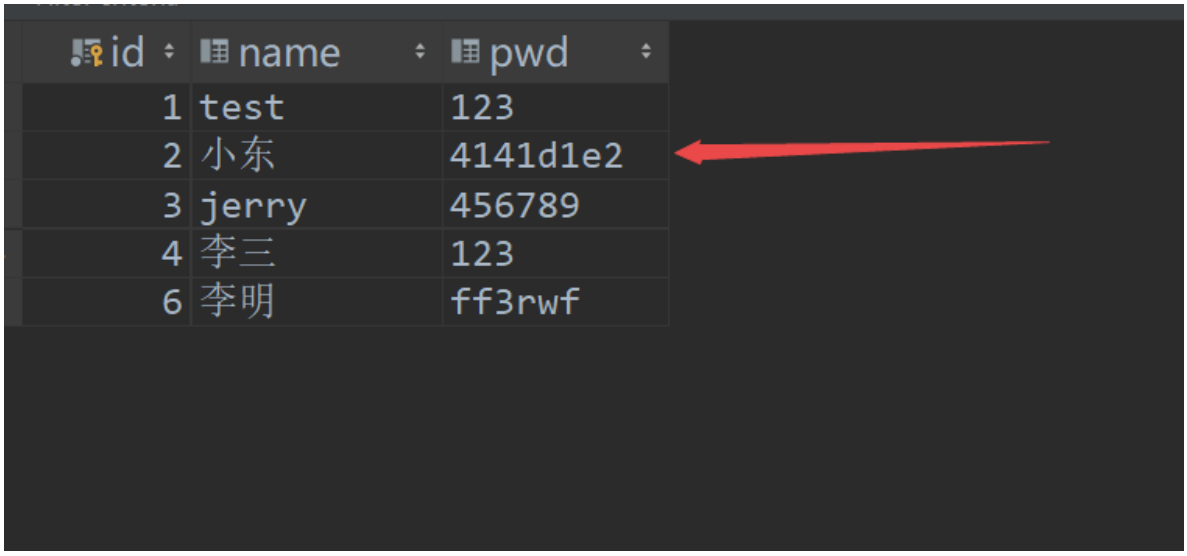
执行前的数据库：



运行报错了：

```
org.springframework.jdbc.BadSqlGrammarException:
### Error updating database.  Cause: com.mysql.jdbc.exceptions.jdbc4.MySQLSyntaxErrorException: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'deletes from mybatis.user where id = 2' at line 1
### The error may exist in file [D:\IDEASSMReview\Spring-Study\spring-11-transaction\target\classes\com\kuang\mapper\UserMapper.xml]
### The error may involve com.kuang.mapper.UserMapper.deleteUser-Inline
### The error occurred while setting parameters
### SQL: deletes from mybatis.user where id = ?
### Cause: com.mysql.jdbc.exceptions.jdbc4.MySQLSyntaxErrorException: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'deletes from mybatis.user where id = 2' at line 1
; bad SQL grammar []; nested exception is com.mysql.jdbc.exceptions.jdbc4.MySQLSyntaxErrorException: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'deletes from mybatis.user where id = 2' at line 1

    at org.springframework.jdbc.support.SQLErrorCodeSQLExceptionTranslator.doTranslate(SQLErrorCodeSQLExceptionTranslator.java:235)
    at org.springframework.jdbc.support.AbstractFallbackSQLExceptionTranslator.translate(AbstractFallbackSQLExceptionTranslator.java:72)
    at org.mybatis.spring.MyBatisExceptionTranslator.translateExceptionIfPossible(MyBatisExceptionTranslator.java:74)
    at org.mybatis.spring.SqlSessionTemplate$SqlSessionInterceptor.invoke(SqlSessionTemplate.java:440) <1 internal call>
    at org.mybatis.spring.SqlSessionTemplate.delete(SqlSessionTemplate.java:303)
    at org.apache.ibatis.binding.MapperMethod.execute(MapperMethod.java:69)
    at org.apache.ibatis.binding.MapperProxy.invoke(MapperProxy.java:58) <1 internal call>
    at com.kuang.mapperImpl.UserMapperImpl.getUsers(UserMapperImpl.java:19) <4 internal calls>
    at org.springframework.aop.support.AopUtils.invokeJoinpointUsingReflection(AopUtils.java:344)
    at org.springframework.aop.framework.ReflectiveMethodInvocation.invokeJoinpoint(ReflectiveMethodInvocation.java:198)
    at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:163)
    at org.springframework.transaction.interceptor.TransactionAspectSupport.invokeWithinTransaction(TransactionAspectSupport.java:353)
    at org.springframework.transaction.interceptor.TransactionInterceptor.invoke(TransactionInterceptor.java:99)
    at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:186)
    at org.springframework.aop.interceptor.ExposeInvocationInterceptor.invoke(ExposeInvocationInterceptor.java:93)
    at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:186)
    at org.springframework.aop.framework.JdkDynamicAopProxy.invoke(JdkDynamicAopProxy.java:212) <1 internal call>
    at com.kuang.test.TestUser.getUsers(TestUser.java:18) <22 internal calls>
```

再看一下数据库：



id	name	pwd
1	test	123
2	小东	4141d1e2
3	jerry	456789
4	李三	123
6	李明	ff3rwf

插入成功了，但是删除失败了。

可以看出，并没有实现事务的效果，尽管过程中出错了，但是前面的执行还是成功了，不符合开发规范!

4 Spring中的事务管理

- 声明式事务：AOP的应用，交由容器管理事务【推荐】
- 编程式事务：在代码中进行事务的管理（try……catch……）

这里我们使用声明式事务，也就是使用AOP，将事务切入到我们想要的方法中。

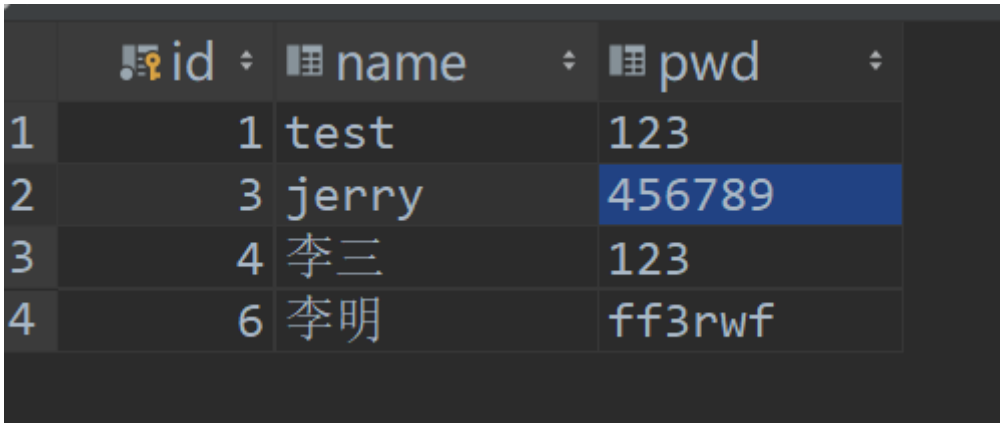
4.1 使用Spring中的AOP进行事务

在Spring核心配置文件中配置，这里我们写在spring-dao.xml中。因为后续不再进行更改。

```
<!--配置声明式事务-->
<bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <constructor-arg ref="dataSource"/>
</bean>

<!--结合AOP，实现事务的织入-->
<!--配置事务通知-->
<tx:advice id="transactionInterceptor" transaction-
manager="transactionManager">
    <!--给哪些方法配置事务    * ：  所有方法    name：要添加事务的方法名

    propagation：配置事务的传播特性 默认为：REQUIRED

    read-only="true"  用于查询时，代表不可以对数据库进行修改
    -->
    <tx:attributes>
        <tx:method name="*" propagation="REQUIRED"/>
    </tx:attributes>
</tx:advice>

<!--配置事务切入
    因为是使用AOP，所有一切都得遵循配置AOP的规则
-->
<aop:config>
    <!--切入点-->
```

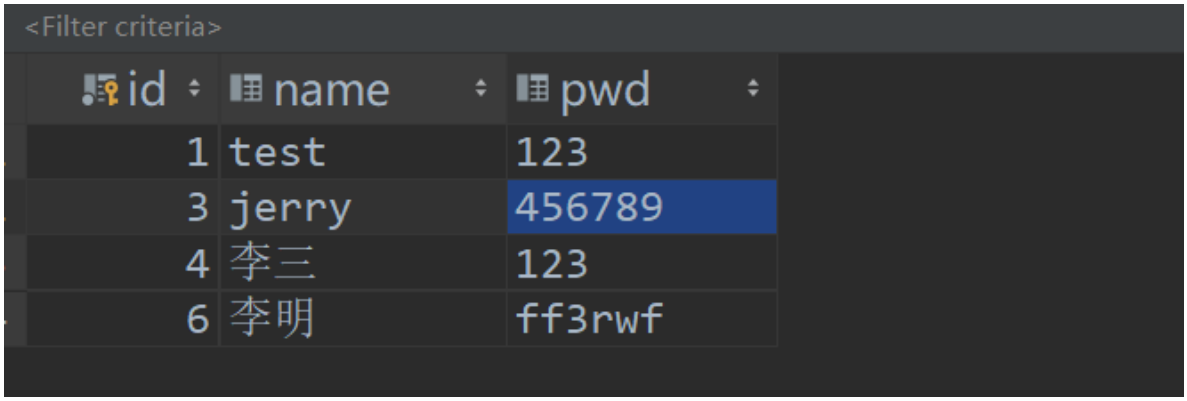
```
25      <aop:pointcut id="txPointCut" expression="execution(*
    com.kuang.mapperImpl.*.*(..))"/>
26
27      <aop:advisor advice-ref="transactionInterceptor" pointcut-
    ref="txPointCut"/>
28  </aop:config>
```

运行前的数据库：



	id	name	pwd
1	1	test	123
2	3	jerry	456789
3	4	李三	123
4	6	李明	ff3rwf

运行后的数据库：

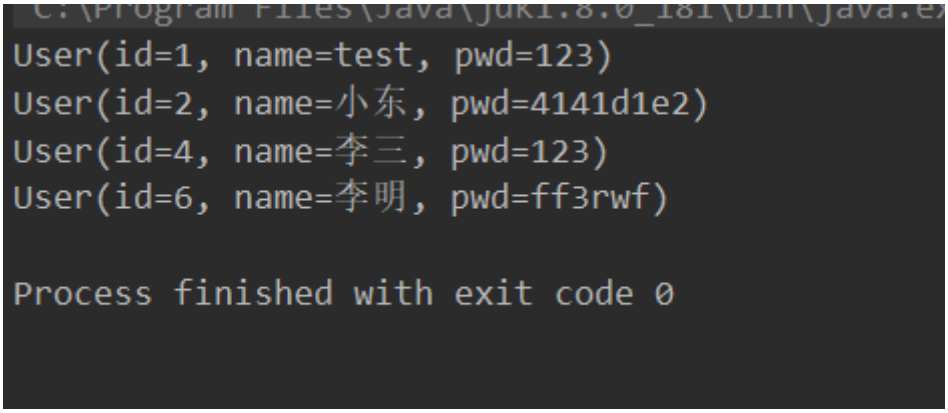


<Filter criteria>

id	name	pwd
1	test	123
3	jerry	456789
4	李三	123
6	李明	ff3rwf

很明显，开启事务之后，在代码中 出错之前的插入id=5的用户已经被事务回滚了。

- 我们将错误的sql语句改正，并将删除id=2改为删除id=3，这样子结果比较明显



```
C:\Program Files\Java\jdk1.8.0_181\bin\java.e
User(id=1, name=test, pwd=123)
User(id=2, name=小东, pwd=4141d1e2)
User(id=4, name=李三, pwd=123)
User(id=6, name=李明, pwd=ff3rwf)

Process finished with exit code 0
```

4.2 propagation

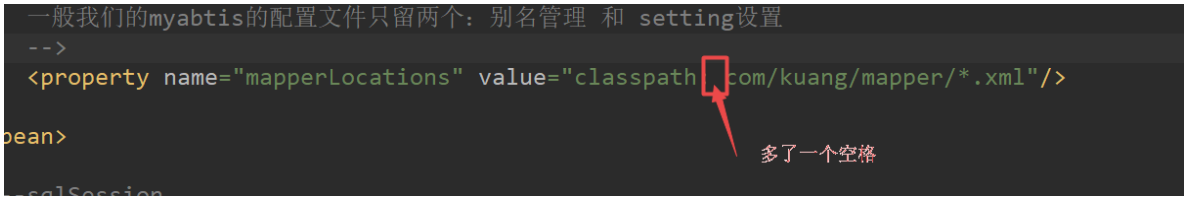
- REQUIRED【常用】：支持当前事务，如果当前没有事务，就新建一个事务。这是最常见的选择。
- SUPPORTS：支持当前事务，如果当前没有事务，就以非事务方式执行。
- MANDATORY：支持当前事务，如果当前没有事务，就抛出异常。

- REQUIRES_NEW：新建事务，如果当前存在事务，把当前事务挂起。
- NOT_SUPPORTED：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。
- NEVER：以非事务方式执行，如果当前存在事务，则抛出异常。
- NESTED【常用】：支持当前事务，如果当前事务存在，则执行一个嵌套事务，如果当前没有事务，就新建一个事务。

5 为什么使用事务?

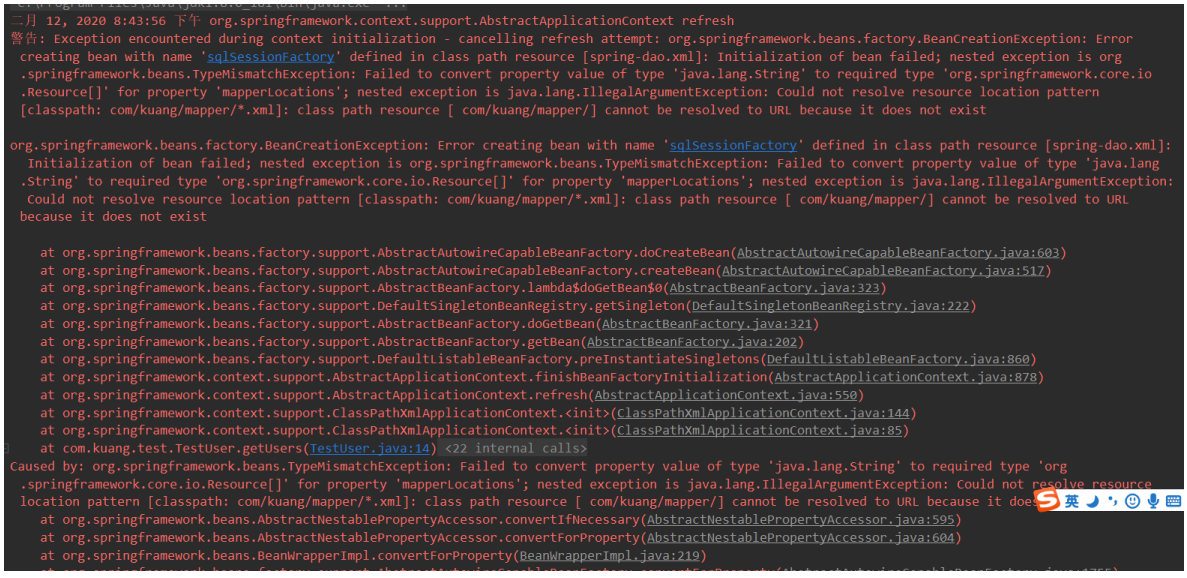
- 如果不配置事务，可能会存在数据不一致的情况
- 如果不在Spring中开启事务，那么则需要在代码中手动配置事务（try……catch……）
- 事务在项目的开发中十分重要，涉及到数据的一致性和完整性

最后，注意一个很难排查的小问题（细心问题）：



这种路径是绝对不允许多一个空格或者多一个什么字符，十分严格！

而且这个东西报的错还十分难找！是这样子的：



完全和空格不沾边。