创建一个项目，模拟电商购买商品的实例，先不使用事务机制。

需求：通过购买方法，购买一件商品，需要商品编号、名称和购买的数量，然后添加到商品表单中。

首先需要建立两张表：

t_goods，商品表，有id、name、amount、price四个字段。表示商品编号（主键）、名称、库存、单价

t_sales，商品表单，有id、gid、gname、amount四个字段。表示主键id、商品编号、商品名称、购买数量

```
drop if exist table t_goods;
create table t_goods(
    id int(11) not null auto_increament,
    name varchar(255) default null,
    amount int(255) default null,
    price decimal(10,2) default null,
    primary key(id)
)ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
drop table if exists t_sale;
create table t_sale(
    id int(11) not null auto_increment,
    gid int(11) default null,
    gname varchar(255) default null,
    amount int(255) default null,
    primary key(id)
)ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

在idea-maven-spring工程中创建模块项目f-spring-not-trans-06，模板使用quickstart，pom文件如下

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.studymyself</groupId>
  <artifactId>f-spring-not-trans-06</artifactId>
  <version>1.0-SNAPSHOT</version>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
```

```
  </properties>

  <dependencies>
    <!--    单元测试依赖-->
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>

    <!--    spring依赖，核心IOC-->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>5.2.5.RELEASE</version>
    </dependency>
    <!--    下面两个做Spring事务的jar包,因为要访问数据库，事务相关-->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-tx</artifactId>
      <version>5.2.5.RELEASE</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-jdbc</artifactId>
      <version>5.2.5.RELEASE</version>
    </dependency>

    <!--    mybatis依赖-->
    <dependency>
      <groupId>org.mybatis</groupId>
      <artifactId>mybatis</artifactId>
      <version>3.5.3</version>
    </dependency>

    <!--    mybatis和Spring集成的依赖,由mybatis官方提供-->
    <dependency>
      <groupId>org.mybatis</groupId>
      <artifactId>mybatis-spring</artifactId>
      <version>2.0.3</version>
    </dependency>

    <!--    mysql驱动依赖-->
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>5.1.13</version>
    </dependency>

    <!--       阿里数据源依赖-->
    <dependency>
      <groupId>com.alibaba</groupId>
      <artifactId>druid</artifactId>
      <version>1.1.20</version>
    </dependency>
  </dependencies>
```

```
  <build>
    <!--    因为mybatis的dao层存放映射文件，不放在resources目录下，所以要加resource插件指定扫描-->
    <resources>
      <resource>
        <!--表示编译时在src/main/java目录下的-->
        <directory>src/main/java</directory>
        <!--所有properties、xml类型的文件可以被扫描到,然后拷贝到编译生成文件夹的对应目录中-->
        <includes>
          <include>**/*.properties</include>
          <include>**/*.xml</include>
        </includes>
        <!--filtering的值false表示不启用过滤器，上面include已经是过滤操作了-->
        <filtering>false</filtering>
      </resource>
    </resources>
  </build>
</project>
```

1、创建实体类商品类Goods和表单类Sale

```
package com.studymyself.entity;

/**
 * 商品类
 */
public class Goods {

    private Integer id;
    private String name;
    private Integer amount;
    private double price;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Integer getAmount() {
        return amount;
    }

    public void setAmount(Integer amount) {
```

```
        this.amount = amount;
    }

    public double getPrice() {
        return price;
    }

    public void setPrice(double price) {
        this.price = price;
    }

    @Override
    public String toString() {
        return "Goods{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", amount=" + amount +
                ", price=" + price +
                '}';
    }
}
```

```
package com.studymyself.entity;

/**
 * 购买的商品表单类
 */
public class Sale {

    private Integer id;
    private Integer gid;
    private String gname;
    private Integer amount;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public Integer getGid() {
        return gid;
    }

    public void setGid(Integer gid) {
        this.gid = gid;
    }

    public String getGname() {
        return gname;
    }

    public void setGname(String gname) {
        this.gname = gname;
    }
```

```
    public Integer getAmount() {
        return amount;
    }

    public void setAmount(Integer amount) {
        this.amount = amount;
    }

    @Override
    public String toString() {
        return "Sale{" +
                "id=" + id +
                ", gid=" + gid +
                ", gname='" + gname + '\'' +
                ", amount=" + amount +
                '}';
    }
}
```

2、创建商品表和销售表单的dao接口以及mapper文件，GoodsDao.java、Sale.java和GoodsDao.xml、Sale.xml

```
package com.studymyself.dao;

import com.studymyself.entity.Goods;
import com.studymyself.entity.Sale;

public interface GoodsDao {

    //修改表中amount字段的值
    public int updateAmount(Sale sale);

    //根据商品id和名称查询数据
    public Goods selectByIdAndName(Sale sale);

}
```

```
package com.studymyself.dao;

import com.studymyself.entity.Goods;
import com.studymyself.entity.Sale;

public interface SaleDao {

    //向表中插入数据
    public int insertSale(Sale sale);

}
```

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
        PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="com.studymyself.dao.GoodsDao">

    <select id="selectByIdAndName" resultType="com.studymyself.entity.Goods">
        select id,name,amount,price from t_goods where id=#{gid} and name=#
{gname}
    </select>

    <update id="updateAmount">
        update t_goods set amount = amount - #{nums} where id = #{gid} and
name=#{gname}
    </update>

</mapper
```

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
        PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="com.studymyself.dao.SaleDao">

    <insert id="insertSale">
        insert into t_sale(gid,gname,nums) values(#{gid},#{gname},#{nums})
    </insert>

</mapper>
```

3、创建service接口和实现类以及自定义的运行时异常

```java
package com.studymyself.exceptions;

/**
 * 自定义的运行时异常
 */
public class NotEnoughException extends RuntimeException{

    public NotEnoughException() {
        super();
    }

    public NotEnoughException(String message) {
        super(message);
    }
}
```

```java
package com.studymyself.service;

public interface BuyGoodsService {

    public void buy(Integer goodId,String goodName,Integer nums);

}
```

```java
package com.studymyself.service.impl;

import com.studymyself.dao.GoodsDao;
import com.studymyself.dao.SaleDao;
import com.studymyself.entity.Goods;
import com.studymyself.entity.Sale;
import com.studymyself.exceptions.NotEnoughException;
import com.studymyself.service.BuyGoodsService;

public class BuyGoodsServiceImpl implements BuyGoodsService {

    GoodsDao goodsDao;
    SaleDao saleDao;

    //购买商品的方法
    @Override
    public void buy(Integer goodId, String goodName, Integer nums) {

        System.out.println("==执行buy方法，开始购买商品==");
        //添加商品销售记录（向t_sale表中插入数据）
        //创建Sale类，并且向其中属性赋值
        Sale sale1 = new Sale();
        sale1.setGid(goodId);
        sale1.setGname(goodName);
        sale1.setNums(nums);
        int count = saleDao.insertSale(sale1);

        //根据提供的信息数据查询商品信息
        Goods good = goodsDao.selectByIdAndName(sale1);

        //根据查询结果判断是否要更新商品库存
        if (good == null){
            count -= 1;
            throw new NullPointerException("商品:"+sale1.getGname()+"编
号:"+sale1.getGid()+"不存在！！！");
        }else if (good.getAmount()<nums){
            count -= 1;
            throw new NotEnoughException("商品:"+sale1.getGname()+"编
号:"+sale1.getGid()+"库存不足！！！");

        }
        //更新商品表
        goodsDao.updateAmount(sale1);
        if (count>1){
            System.out.println("已购买商品");
        }
```

```
    }

    public void setGoodsDao(GoodsDao goodsDao) {
        this.goodsDao = goodsDao;
    }

    public void setSaleDao(SaleDao saleDao) {
        this.saleDao = saleDao;
    }
}
```

4、编写mybatis核心配置文件和配置文件Spring配置文件以及属性配置文件

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
        PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>

    <!--注意这些属性设置从上往下是有顺序的，写反就会报错-->

    <!--配置属性资源文件-->
    <properties resource="jdbc.properties"/>

    <!--该属性的作用：控制mybatis全局行为，
        顺序要在上面的properties属性下面-->
    <settings>
        <!--设置mybatis输出日志-->
        <setting name="logImpl" value="STDOUT_LOGGING"/>
    </settings>

    <!--给这个包下的所有类起别名-->
    <typeAliases>
        <package name="com.studymyself.entity"/>
    </typeAliases>

    <!--配置插件-->
<!--    <plugins>-->
<!--        &lt;!&ndash;分页插件&ndash;&gt;-->
<!--        <plugin interceptor="com.github.pagehelper.PageInterceptor">
</plugin>-->
<!--    </plugins>-->


    <mappers>
        <!--第一种方式-->
        <!-- <mapper resource="mapper文件的类根路径"/>-->

        <!--第二种方式：使用包名
            其中name属性的值：mapper文件所在的包名
            使用package的条件：
                1、mapper文件的名称要和接口的名称一致，包括大小写一致
                2、mapper文件要和接口在同一个目录之中-->
```

```
        <package name="com.studymyself.dao"/>
    </mappers>
</configuration>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:util="http://www.springframework.org/schema/util"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/util
https://www.springframework.org/schema/util/spring-util.xsd
http://www.springframework.org/schema/context
https://www.springframework.org/schema/context/spring-context.xsd">

    <!--让spring知道jdbc.properties文件位置-->
    <context:property-placeholder location="classpath:jdbc.properties"/>

    <!--声明数据源DataSource对象，作用是连接数据库-->
    <bean id="myDataSource" class="com.alibaba.druid.pool.DruidDataSource"
          init-method="init" destroy-method="close">
<!--          使用set注入赋值给DruidDataSource提供连接数据库的信息-->
        <property name="url" value="${jdbc.url}"/>
        <property name="username" value="${jdbc.username}"/>
        <property name="password" value="${jdbc.password}"/>

        <property name="maxActive" value="${jdbc.maxActive}"/>
     </bean>

<!--    声明mybatis提供的SqlSessionFactoryBean类对象，这个类内部有一个方法是创建
SqlSessionFactory-->
<!--    对象的，这样我们声明SqlSessionFactoryBean类对象就是声明SqlSessionFactory对象
了。就像之前-->
<!--    使用的SqlSessionFactoryBuilder类中的build方法一样-->
    <bean id="sqlSessionFactory"
class="org.mybatis.spring.SqlSessionFactoryBean">
        <!--我们知道之前创建SqlSessionFactory对象时需要使用到mybatis的核心配置文件，而
            配置文件中的核心配置中没有了。改为上面的声明对象了，所以SqlSessionFactoryBean
类中
            有一个属性存储数据库信息，这时我们就把上面的数据源对象放到该属性中-->

        <!--通过set注入把数据库连接池赋给了dataSource属性,引用类型属性注意用ref-->
        <property name="dataSource" ref="myDataSource"/>

        <!--下面就是把核心配置文件读取到这个类中的configLocation属性中，属性是Resource类
型
            的，就像之前用mybatis中获取核心配置文件时Resources.getResourceAsStream一
样，是
            读取配置文件的,注意注入值是文件时要在前面添加路径标识：classpath-->
        <property name="configLocation" value="classpath:mybatis.xml"/>

        <!--最后创建的SqlSessionFactory对象类名是DefaultSqlSessionFactory,这是
SqlSessionFactory接口的实现类-->
    </bean>
```

```
<!--    这里就是创建dao对象，或者dao的代理对象，使用SqlSession的getMapper(接口的Class对象)
        我们不能为每一个dao接口去一个一个调用getMapper方法生成代理对象，所以声明某个类对象
        一次性把所有的dao接口都生成代理对象，如下
        MapperScannerConfigurer这个类：可以一次性把符合条件的dao接口都生成其代理对象
        其中每个代理对象存储在Spring容器集合中的key是接口的首字母小写-->
    <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
        <!--之前我们要调用getMapper方法需要用到sqlSessionFactory对象和dao接口的Class对象
        所以MapperScannerConfigurer中有两个属性要获取这些数据-->

        <!--sqlSessionFactory对象使用上面的声明的bean，用set注入-,属性是String类型，用value-->
        <property name="sqlSessionFactoryBeanName" value="sqlSessionFactory"/>
        <!--这里直接把dao接口的包名赋值给获取dao接口Class对象的属性，一次性创建所有dao接口的代理对象
            MapperScannerConfigurer会扫描这个包装的所有接口，为每个接口都执行一次getMapper方法，得到
            的每个接口的代理对象都放到Spring容器中
             -->
        <property name="basePackage" value="com.studymyself.dao"/>
    </bean>

<!--    声明service类对象-->
    <bean id="buyGoodsService"
class="com.studymyself.service.impl.BuyGoodsServiceImpl">
        <property name="goodsDao" ref="goodsDao"/>
        <property name="saleDao" ref="saleDao"/>
    </bean>

</beans>
```

```
jdbc.url=jdbc:mysql://localhost:3306/mysqlstudy?useSSH=false
jdbc.username=root
jdbc.password=rong195302
jdbc.maxActive=30
```

5、创建测试程序测试代码

```
package com.studymyself;

import com.studymyself.service.BuyGoodsService;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MyTest {

    /**
     * 测试购买商品的方法
     */
    @Test
    public void testBuy(){

        //定义Spring配置文件路径
        String config = "applicationContext.xml";
```

```
        //获取Spring容器对象
        ApplicationContext ac = new ClassPathXmlApplicationContext(config);

        //获取service对象
        BuyGoodsService buyGoodsService = (BuyGoodsService)
ac.getBean("buyGoodsService");
        System.out.println(buyGoodsService.getClass().getName());

        buyGoodsService.buy(1003,"手机",2000);

    }

}
```