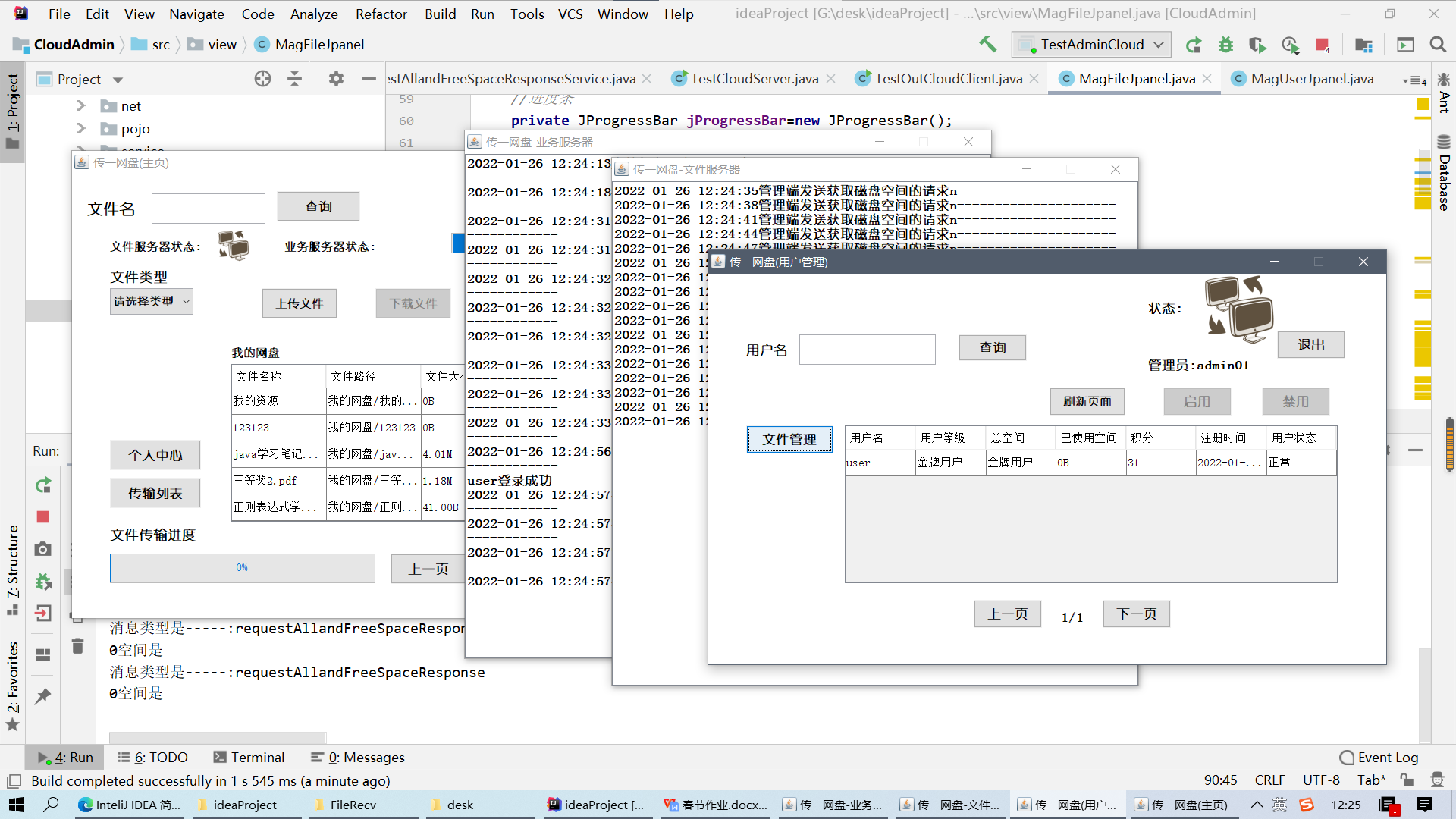
1. 安装IDEA，并将网盘项目移植到IDEA上运行，并截图。



1. 跟据以下提纲预习MySql相关内容

* 什么是事务、详述事务的四大特性？

**事务的定义**：事务用于保证数据的一致性，它由一组相关的DML语句组成，DML语句要么全部成功，要么全部失败。如：转账功能就要用到事务来处理，用以保证数据的一致性。

**事务的四大特性：**

1. 原子性：原子性是指事务时一个不可分割的工作单位，事务中的操作要么发生，要么不发生。
2. 一致性：事务必须使数据库从一个一致性状态变换到另外一个一致状态。
3. 隔离性：事务的隔离性是多个用户并发访问数据库时，数据库为每一个用户开启的事务，不能被其他事务的数据所干扰，多个并发事务之间要相互隔离。
4. 持久性：持久性指一事务一旦被提交，它对数据库中数据的改变就是永久性的，接下来即使数据库发生故障也不应该对其有任何影响。

**事务的重要操作：**

1. start transaction ——开启一个事务
2. Savepoint 保存点名 ——设置保存点
3. Rollback to 保存点名 ——回滚事务
4. Rollback ——回退全部事务
5. Commit ——提交事务，所有操作生效，不能回退

* 事务的隔离级别有哪些？

**定义**：多个连接开启各自事务操作数据库中数据的时候，数据库系统要负责隔离操作，以保证各个连接在获取数据的准确性。

如果不考虑隔离性，可能会引发如下问题：

**脏读**（当一个事务读取另一个事务尚未提交的改变（update、insert、delete）时）产生脏读）（个人理解：当一个事务在执行增删改的操作时，它还没提交，而另外一个数据库可以读取它这个事务增删改操作的结果）

**不可重复读**（同一查询在同一事务中多次进行，由于其他提交事务所做的修改和删除，每次返回不同的结果集，此时就发生了不可重复读）（个人理解：当一事务进行增删改操作的时候，它还未提交，另外一个事务查询不到它增删改的结果，就会发生一个不可重复读的结果，幻读也是一样幻读针对的是插入操作）。

幻读（同一查询在同一事务中多次进行，由于其他事务所做的修改和删除，每次返回不同的结果集，此时就发生了不可重复读）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Mysql隔离级别 | 脏读 | 不可重复读 | 幻读 | 加锁读 |
| 读未提交（read uncommitted） | √ | √ | √ | 不加锁 |
| 读已提交（read committed） | × | √ | √ | 不加锁 |
| 可重复读（repetable read） | × | × | × | 不加锁 |
| 可串行化  （serializable） | × | × | × | 加锁 |

* 数据库锁的作用？

**定义**：当数据库有并发事务的时候，可能会产生数据的不一致，只是偶需要一些机制来保证访问的次序，锁机制就是这样的一个机制。

**作用**：用户管理对共享资源的并发访问，保证数据库的完整性和一致性，跟我们之前加同步锁来控制数据的一致性很相似。

锁分为表级锁、行级锁、页面锁。

**表级锁**：表级锁是mysql中锁定粒度最大的一种锁，表示对当前操作的整张表加锁，它实现简单，资源消耗小，最常使用的myisam和innodb都支持表级锁定。表级锁分为共享锁和排他锁。特点：开销小、加锁快，不会出现死锁，锁定粒度大、发生锁冲突的概率最高、并发度最低。

**行级锁**：行级锁是mysql中锁定粒度最细的一种锁，表示只针对当前操作的行进行加锁，行级锁能大大减少数据库操作的冲突，其加锁粒度最小，但加锁的而开销也最大。特点：开销大，加锁慢；会出现死锁；锁定粒度最小，发生锁冲突的概率最低，并发度也最高。

**页面锁：**页面锁是mysql中锁定粒度介于行级锁和表级锁中间的一种锁。表级锁速度快，但冲突多，行级冲突少，但速度慢。所以取了折中的页级，一次锁定相邻的恶意阻记录。

特点：开销和加锁时间介于表锁和行锁之间；会出现死锁；锁定粒度介于表锁和行锁之间，并发度一般。

* 什么是存储引擎？MySql存储引擎有哪几种？默认用的是哪一种？

**存储引擎定义**：MySQL中的数据用各种不同的技术存储在文件（或者内存）中。这些技术中的每一种技术都使用不同的存储机制、索引技巧、锁定水平并且最终提供广泛的不同的功能和能力。通过选择不同的技术，你能够获得额外的速度或者功能，从而改善你的应用的整体功能。

存储引擎的分类:

**Innodb**:Innodb是事务性数据库的首选引擎，如果要我们的表要使用到事务的会，就要用到这个存储引擎，是目前最重要、使用最广泛的存储引擎。支持事务安全表，支持行锁定和外键。特性：提供了提交、回滚和崩溃恢复能力的事务安全存储引擎。是为处理巨大数据量的最大性能设计，它的cpu效率可能是任何其他基于磁盘的关系型数据库引擎锁不能匹敌的。Innodb支持外键完整性约束。

**MyISAM**:myisam基于isam存储疫情，并对其进行扩展，它时在web、数据仓储和其他应用环境下最常使用的存储引擎之一。Myisam拥有较高的插入、查询速度，但不支持事务和外键。特性：支持大文件。当把删除和更新及插入操作混合使用的时候，动态尺寸的行产生更少碎片。可以把数据库文件和索引文件放在不同的目录。每个字符列可以有不同的字符集，有varchar的表可以固定或动态记录长度。

**MERGE:**merge存储疫情一组myisam表的组合，这些myisam表结构必须完全相同，简单来说，merge就是几个相同myisam表的聚合器，merge表中并没有数据，对merger类型的表可以进行查询、更新、删除操作，这些操作实际上是对内部的myisam表进行操作。

**MEMORY:**memory存储引擎将表中的数据存储到内存中，为查询和引用其他表数据提供快速访问。特性：memory表的每个表可以有多达32个索引，每个索引16列，以及500字节的最大键长度，memory存储引擎执行hash和btree索引，可以在一个memory表中有非唯一键值，memory表使用一个固定的记录长度格式，memory不支持blob或者text列。Memory支持auto\_increment列和对可包含null值的列的索引。

* 分别从锁的粒度（表锁、页锁、行锁）和锁的类型两方面来描述InnoDB中提供的数据库锁

**表级锁**：表级锁是mysql中锁定粒度最大的一种锁，表示对当前操作的整张表加锁，它实现简单，资源消耗小，最常使用的myisam和innodb都支持表级锁定。表级锁分为共享锁和排他锁。特点：开销小、加锁快，不会出现死锁，锁定粒度大、发生锁冲突的概率最高、并发度最低。

**行级锁**：行级锁是mysql中锁定粒度最细的一种锁，表示只针对当前操作的行进行加锁，行级锁能大大减少数据库操作的冲突，其加锁粒度最小，但加锁的而开销也最大。特点：开销大，加锁慢；会出现死锁；锁定粒度最小，发生锁冲突的概率最低，并发度也最高。

**页面锁：**页面锁是mysql中锁定粒度介于行级锁和表级锁中间的一种锁。表级锁速度快，但冲突多，行级冲突少，但速度慢。所以取了折中的页级，一次锁定相邻的恶意阻记录。

特点：开销和加锁时间介于表锁和行锁之间；会出现死锁；锁定粒度介于表锁和行锁之间，并发度一般。

* 索引的作用？MySql如何创建索引？

索引可以提高数据库的性能，不用改程序，不用调sql，就可以使得查询速度变得很快。

创建索引有很多种方式，创建索引的类型也有很多。

索引的类型

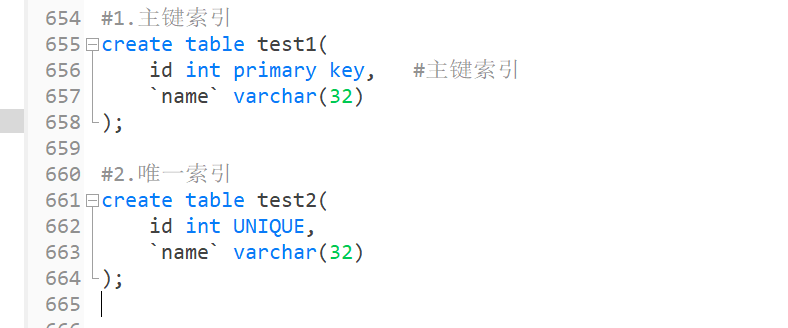
1.主键索引，主键自动的为主索引

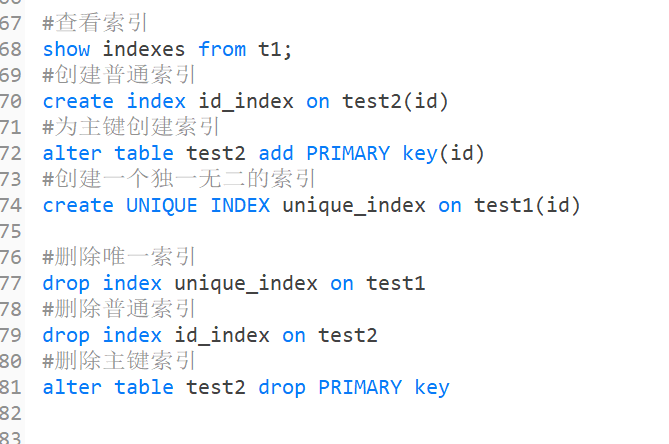
2.唯一索引（unique）

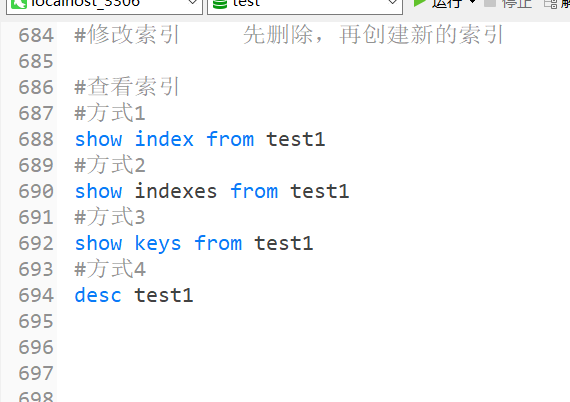
3.普通索引（index）

4.全文索引（fulltext）【使用与myisam存储引擎】

索引基本命令







1. 跟据以下提纲预习MyBatis相关内容（已完成）

* 什么是框架？

Java框架就是一些类和接口的集合，通过这些类和接口协调来完成一系列的程序实现。框架又叫做开发中的半成品，它不能提供整个WEB应用程序的所有东西，但是有了框架，我们就可以集中精力进行业务逻辑的开发而不用去关心它的技术实现以及一些辅助的业务逻辑。说白了Java框架就是封装好方便程序员操作的类，使项目的开发更简单，维护起来也更容易。

* 什么是Mybatis？

MyBatis源自Apache的iBatis开源项目, 从iBatis3.x开始正式更名为MyBatis。它是一个优秀的持久层框架。Mybatis是一个半自动的ORM持久层框架，内部封装了JDBC。作为开发者只需要关注sql语句本身。Mybatis是通过xml或注解的方式将需要执行的各种statement配置起来。通过Java对象和statement中的sql动态参数映射生成最终执行的sql语句，最终由Mabtais框架执行sql并将结果映射为Java对象并返回。MyBatis 支持定制化 SQL、存储过程以及高级映射。MyBatis 是可以双向映射的，可以将数据集映射为Java对象，也可以将Java对象映射为数据库中的记录。

* Mybatis的优缺点？

优点：

1. 简单易学，容易上学，基于SQL编程。
2. 与JDBC相比，减少了50%以上的代码量，消除了JDBC大量冗余的代码，不需要手动开关连接。
3. 很好的与各种数据库兼容，开发人员不需要考虑数据库的差异性。
4. 提供了很多第三方插件（分页插件/逆向工程）。
5. 能够与Spring很好的集成。
6. 很灵活，不会对应用程序或者数据库的现有设计强加任何影响，SQL写在XML里，从程序代码中彻底分离，解除sql与程序代码的耦合，便于同一管理和优化，并可重用。
7. 提供映射标签，支持对象与数据库的ORM字段关系映射。

缺点：

（1）SQL语句的编写工作量大，尤其是字段多、关联表多时，更是如此，对开发人员编写SQL语句的功底有一定要求。

（2）SQL语句依赖于数据库，导致数据库移植性差，不能随意更换数据库。

* XML文件格式解读

**mybatis-config.xml**

<?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>

<environments default="mysql">

<environment id="mysql">

<transactionManager type="JDBC"></transactionManager>

<dataSource type="POOLED">

<property name="driver" value="com.mysql.jdbc.Driver"></property>

<propertyname="url" value="jdbc:mysql://127.0.0.1:3306/mybbs"></property>

<property name="username" value="root"></property>

<property name="password" value="1234"></property>

</dataSource>

</environment>

</environments>

<mappers>

<mapper resource="User.xml"></mapper>

</mappers>

</configuration>

**User.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.test.springtest.User">

<select id="GetUserByID" parameterType="int" resultType="com.test.springtest.dao.MUser">

select \* from `student` where id = #{id}

</select>

<insert

id="saveUser" parameterType="com.test.springtest.User"

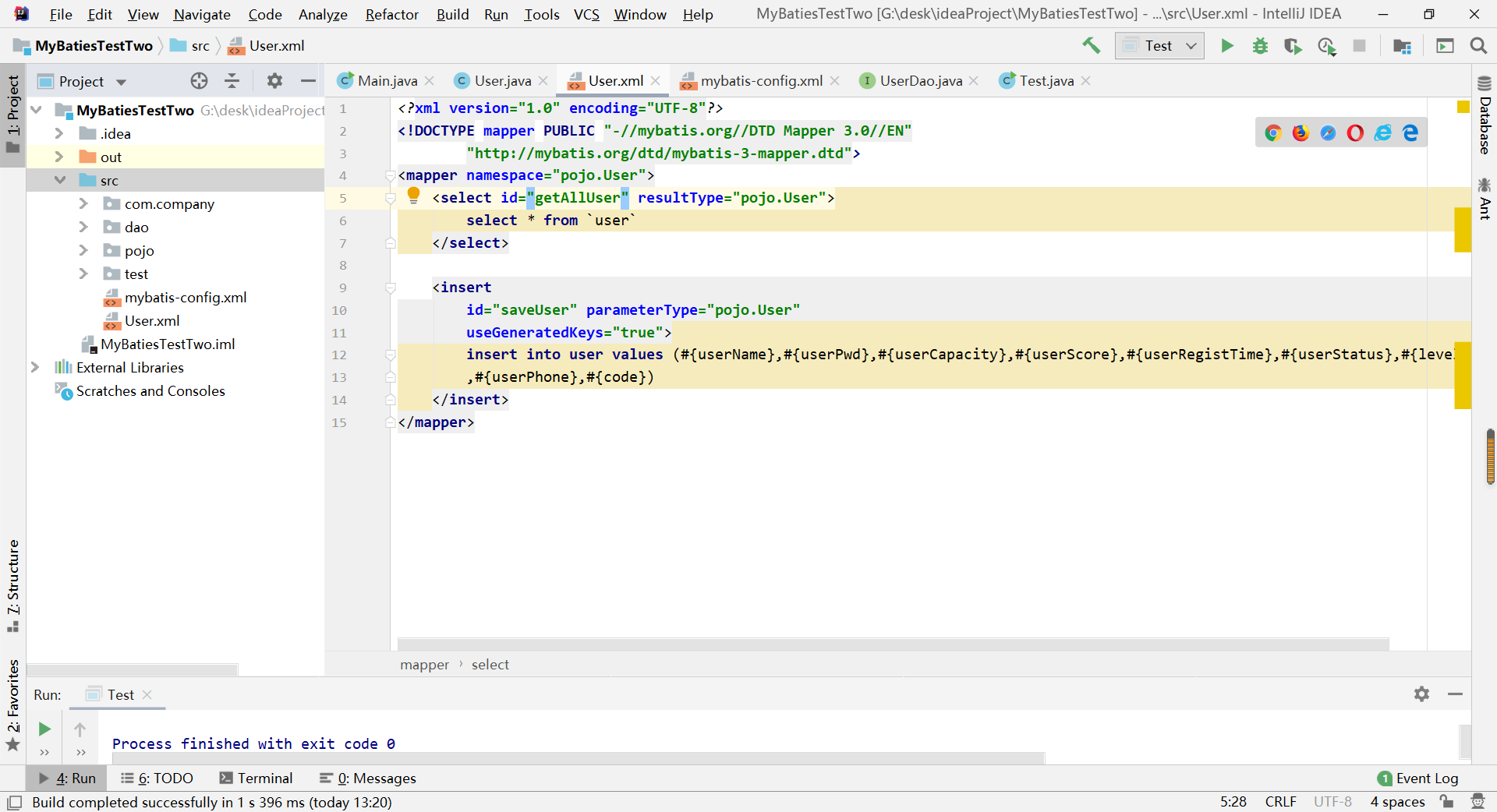
useGeneratedKeys="true">

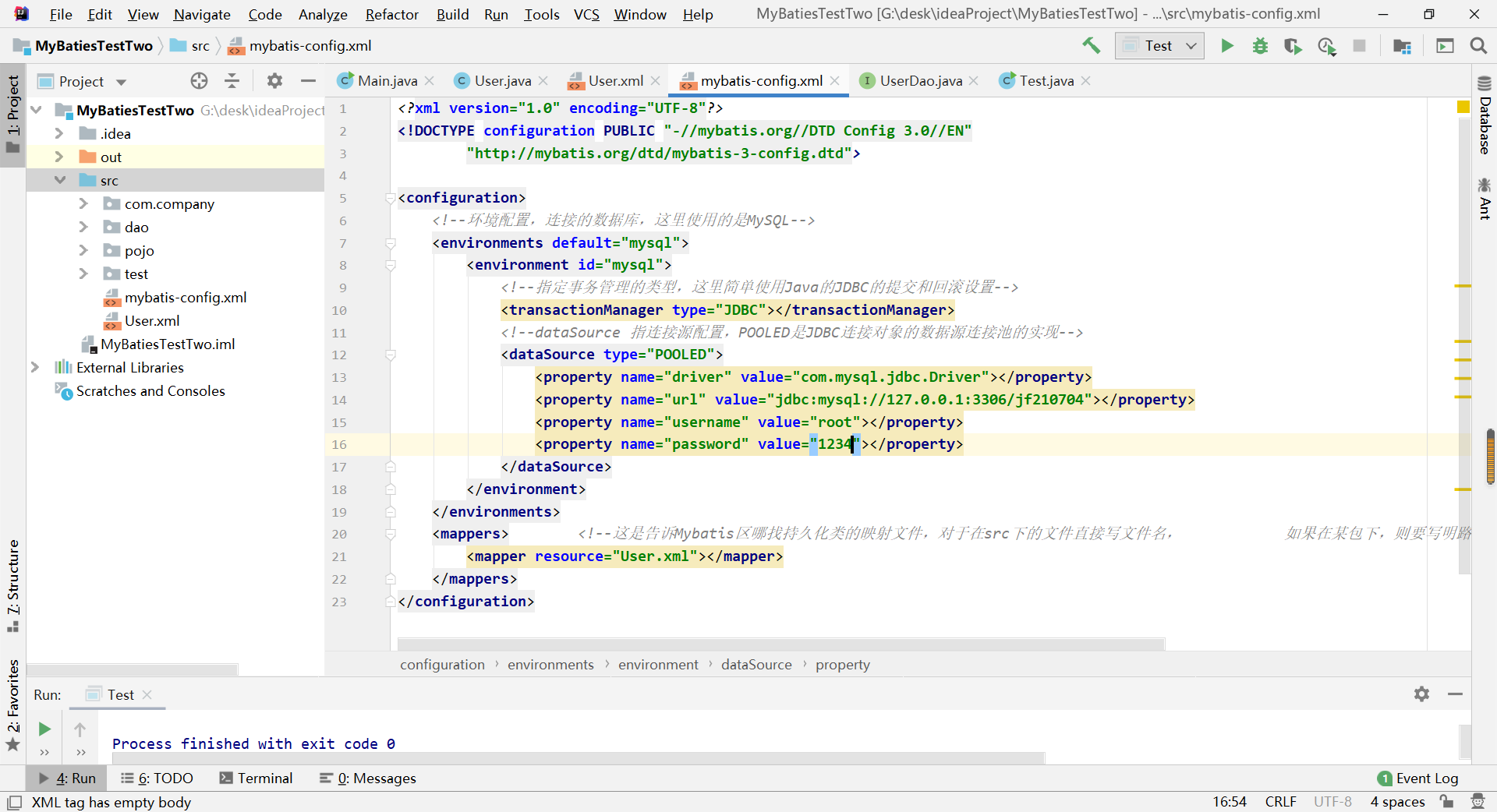
insert into student(NAME,AGE) values (#{name},#{age})

</insert>

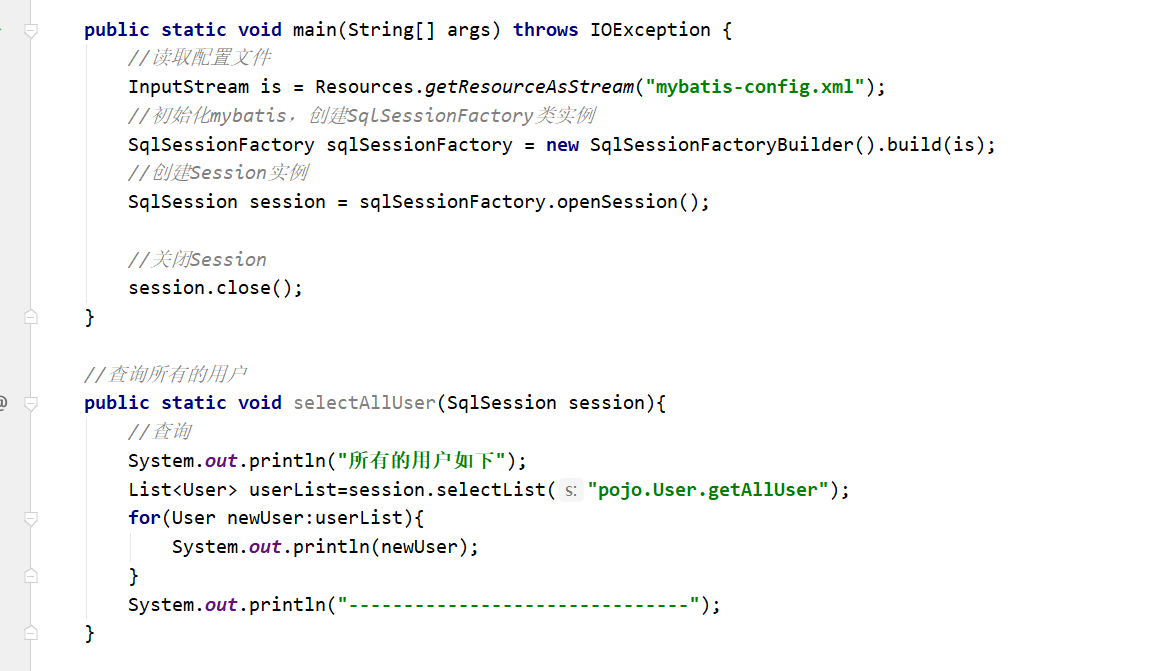
</mapper>

* 请分别使用配置文件方式和注解方式实现增、删、改查（编写4个Demo）（还未完全完善好）





第一种方式较为传统

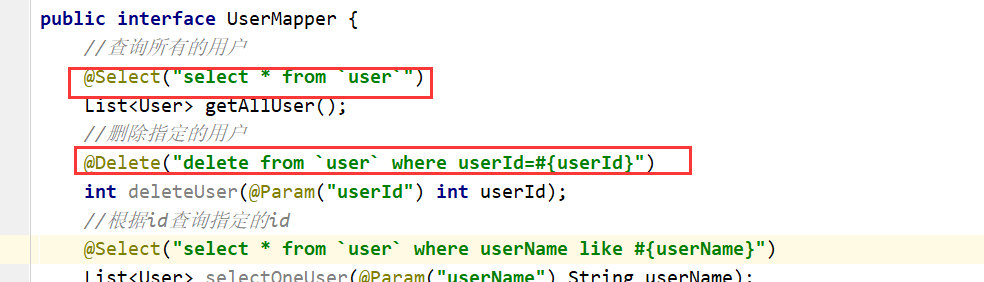




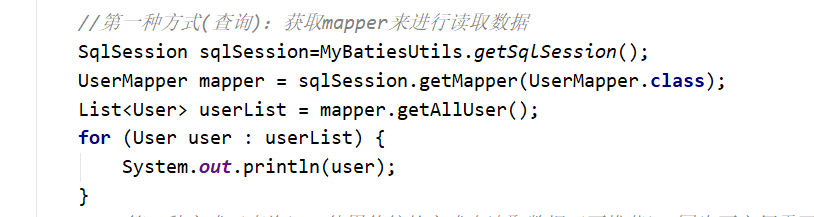
第二种方式（通过获取mapper接口来获取对应的mapper接口类中的方法（常用），但还是使用配置文件的形式 下面是使用注解进行的查询）



使用注解的增删改查







* Mapper映射器的作用？

首先来看一下Mapper的定义，它是符合映射文件要求的接口

接口要求

a. 方法名要与sql的id一致。

b. 方法的参数类型要与parameterType一致。

c. 方法的返回类型要与resultType一致。

映射文件要求:namespace必须等于接口名(包含包名)

作用：如果我们不使用Mapper映射器，那么我们就必须调用SqlSession的相应方法区执行相应的增、删、改、查操作；如果我们使用了Mapper映射器，那么我们执行增、删、改、查这些操作时使用的就是Mapper映射器中的相应方法；但是要想使用Mapper映射器中的方法必须要有一个Mapper映射器对象，这个对象可以通过SqlSession的getMapper方法得到，例如：AdminDao ad = sqlSession.getMapper(AdminDao.class);当然也有得到Mapper对象的简便方法，那就是使用spring去集成mybatis，通过在spring配置文件中配置MapperScannerConfigurer就可以帮相应的Mapper映射器配置一个bean，以后我们就可以通过spring容器来使用这个bean。

* Mapper中的resource、class和package属性的区别

1.resource加载方式

适用于类路径下的文件加载(注意:mybatis原始开发Dao.xml文件与接口文件不在同一路径下,仅能用resource加载映射文件)。

2.class加载方式

仅适用于类路径下,接口文件与映射文件在同一路径下,且接口名与映射文件名相同,并且映射文件命名为接口全类名的情况。

3.package 加载方式(class加载方式的简写方式)

适用于类路径下,接口文件与映射文件在同一路径下,且接口名与映射文件名相同,并且映射文件命名为接口全类名的情况。

* ${}和#{}有什么区别

#{}:表示一个占位符号，通过#{}可以实现preparedStatement向占位符中设置值，自动进行java类型和jdbc类型转换，#{}可以有效防止sql注入。

#{}可以接收简单类型值或pojo属性值。

如果parameterType传输单个简单类型值，#{}括号中可以是value或其它名称。

“%”#{name}”%”

${}:表示拼接sql串，通过${}可以将parameterType 传入的内容拼接在sql中且不进行jdbc类型转换， ${}可以接收简单类型值或pojo属性值，如果parameterType传输单个简单类型值，${}括号中只能是value。

* 使用MyBatis的mapper接口调用时有哪些要求

（1）Mapper 接口方法名和 mapper.xml 中定义的每个 sql 的 id 相同。

（2）Mapper 接口方法的输入参数类型和 mapper.xml 中定义的每个 sql 的

parameterType 的类型相同。

（3）Mapper 接口方法的输出参数类型和 mapper.xml 中定义的每个 sql 的

resultType 的类型相同。

（4）Mapper.xml 文件中的 namespace 即是 mapper 接口的类路径。

* Mybatis动态sql是做什么的？都有哪些动态sql？能简述一下动态sql的执行原理不？

作用：动态sql是指在进行sql操作的时候，传入的参数对象或者参数值，根据匹配的条件，有可能需要动态的去判断是否为空，循环，拼接等情况。

动态sql：

if标签中可以判断传入的值是否符合某种规则，比如是否不为空。

where标签可以用来做动态拼接查询条件，当和if标签配合的时候，不用显示的声明类似where 1=1这种无用的条件，来达到匹配的时候and会多余的情况。

include可以把大量重复的代码整理起来，当使用的时候直接include即可，减少重复代码的编写。

choose、when、otherwise 标签

类似于 Java 中的 switch、case、default。只有一个条件生效，也就是只执行满足的条件 when，没有满足的条件就执行 otherwise，表示默认条件。

foreach 标签

foreach标签可以把传入的集合对象进行遍历，然后把每一项的内容作为参数传到sql语句中，里面涉及到 item(具体的每一个对象), index(序号), open(开始符), close(结束符), separator(分隔符)

原理：

首先在解析xml配置文件的时候，会有一个SqlSource sqlSource = langDriver.createSqlSource(configuration, context, parameterTypeClass) 的操作

createSqlSource底层使用了XMLScriptBuilder来对xml中的标签进行解析

XMLScriptBuilder调用了parseScriptNode()的方法，

在parseScriptNode()的方法中有一个parseDynamicTags()方法，会对nodeHandlers里的标签根据不同的handler来处理不同的标签

然后把DynamicContext结果放回SqlSource中

DynamicSqlSource获取BoundSql

在Executor执行的时候，调用DynamicSqlSource的解析方法，并返回解析好的BoundSql，和已经排好序，需要替换的参数。

* 了解Mybatis高级映射

在开发需求中有一些一对一、一对多和多对多的需求开发，如在开发购物车的时候，订单和用户是一对一，用户和订单是一对多，用户和商品是多对多。这些在Hibernate开发中也是常见的，Hibernate中是通过数据映射来实现的，在MyBatis中也是通过配置文件的数据映射来实现。

* 什么是一对一、一对多的关联查询

**多对一的理解：**

例如网盘项目中：

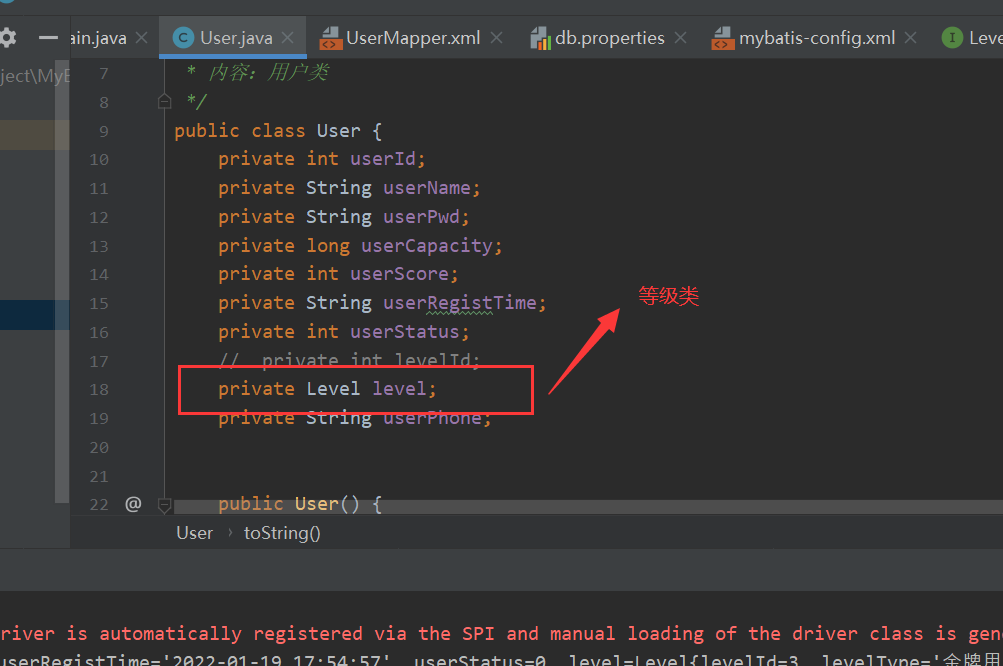
多个用户对应一个等级

如果对于用户这边，就是一个多对一的现象，即从用户这边关联一个等级！

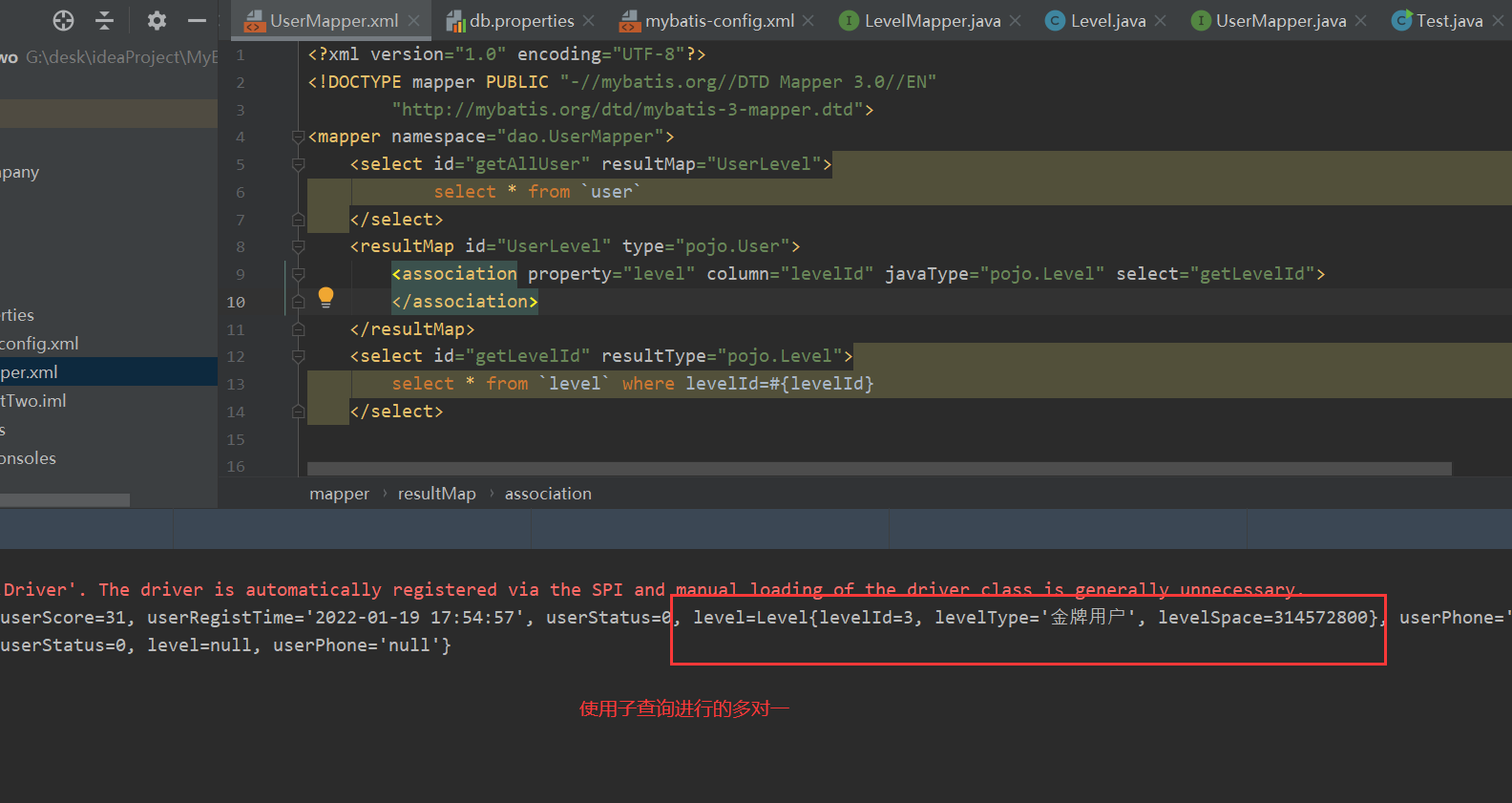
数据库对应关系

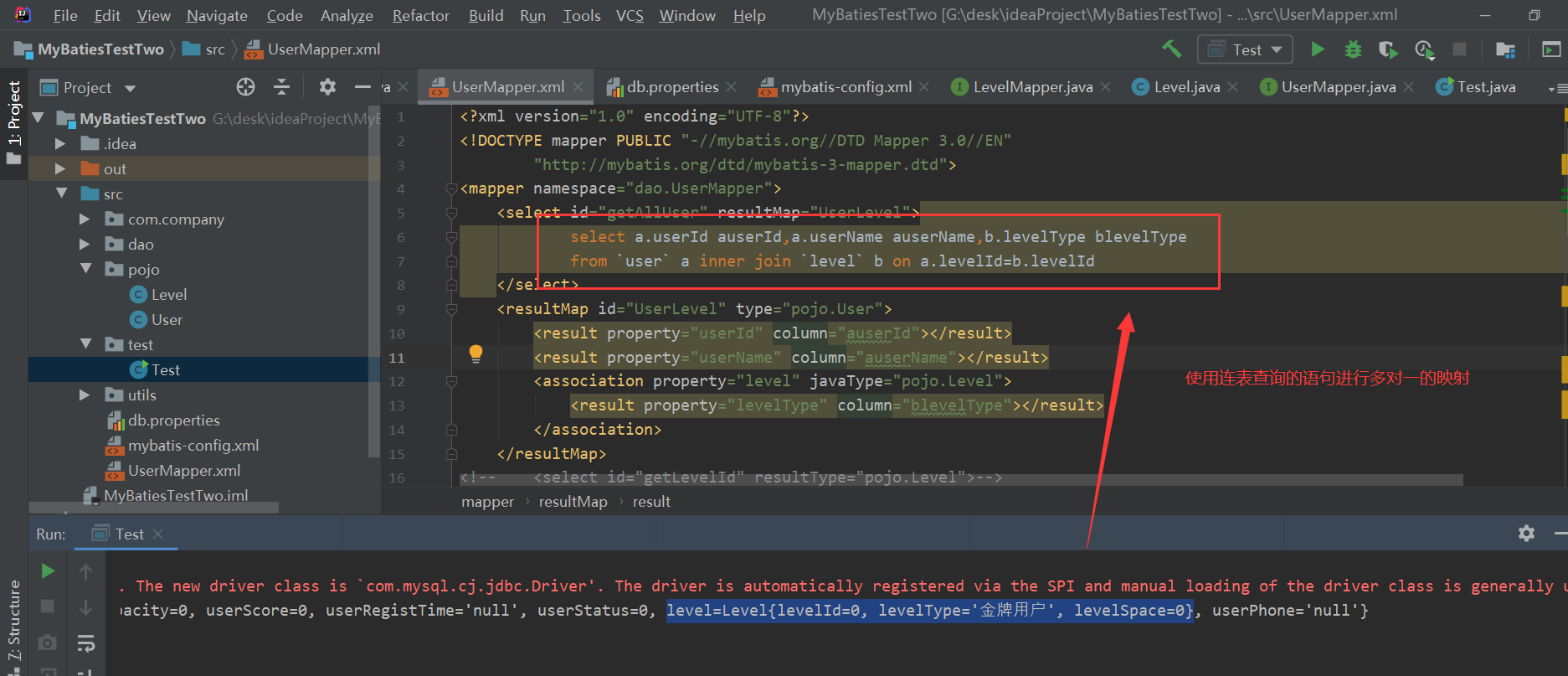


Pojo中对应关系



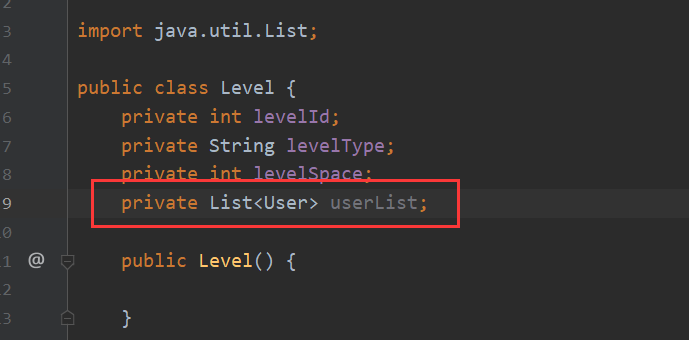
Mybaties中实现

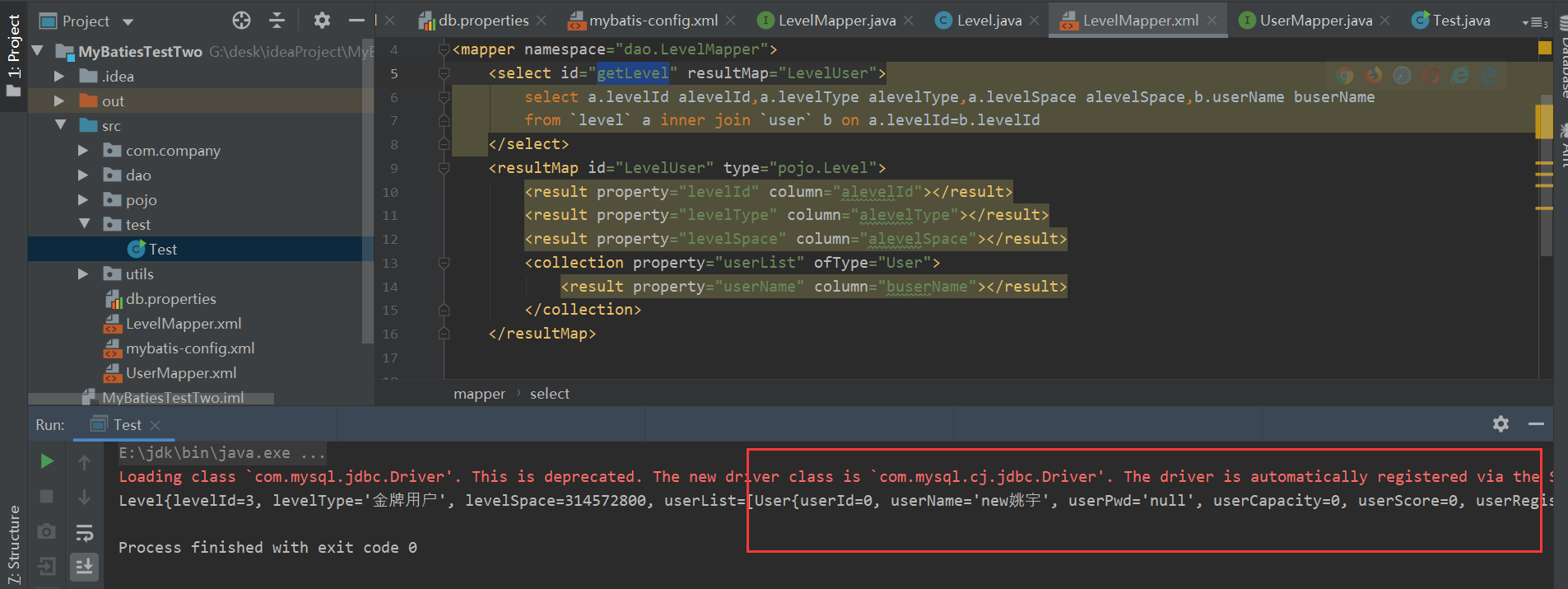


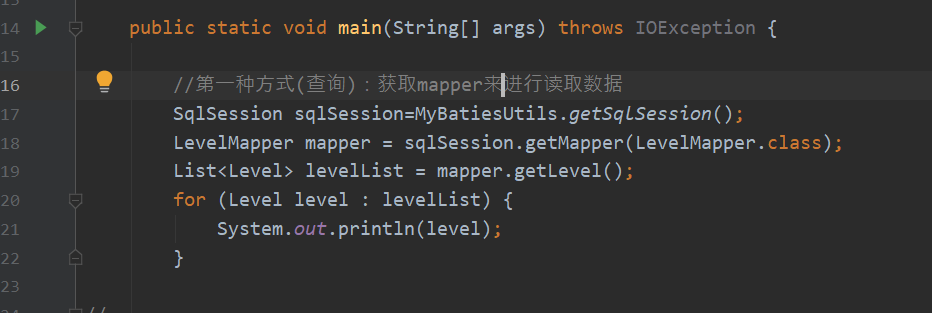


一对多的理解：

例如：某一个等级有许多个用户







* Mybatis都有哪些Executor执行器？它们之间的区别是什么？默认使用哪种执行器？

mybatis有三种基本的执行器：

第一种（SimpleExecutor）：每执行一次update或者select，就开启一个Statement对象，用完立刻关闭对象。

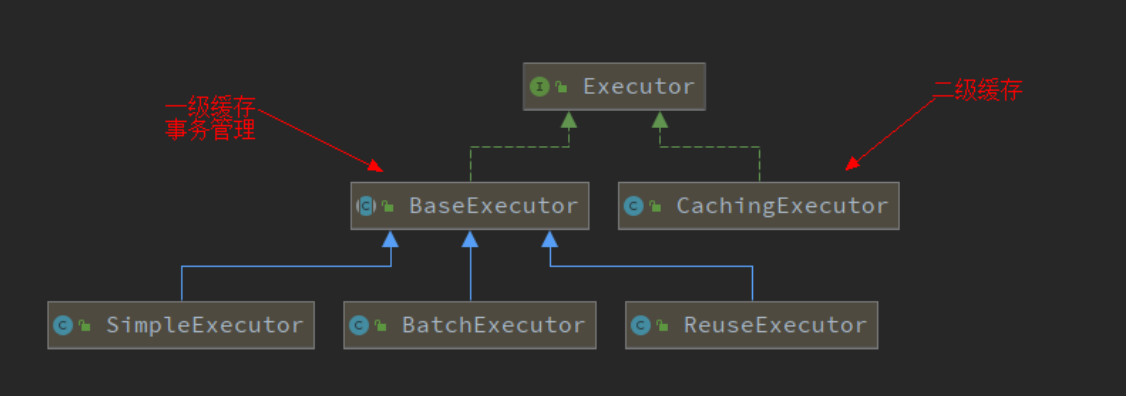
第二种（ReuseExecutor）：执行update或select，以sql作为key 查找statement对象，存在就是用，不存在就创建，不关闭statement对象，而是放置在map集合中，提供下一次是用，作到复用的效果。

第三种（BactchExectutor）：执行update，将所有sql都添加到批处理中,addBatch（）等待统一执行executeBatch（），它缓存了多个Statement对象，每个Statement对象都是addBactch（）完毕后，等待逐一执行executeBatch（）批处理，与jdbc批处理相同。

第四种（CachingExecutor）：CachingExecutor是一个Executor接口的装饰器，它为Executor对象增加了二级缓存的相关功能，委托的执行器对象可以是SimpleExecutor、ReuseExecutor、BatchExecutor中任一一个。执行 update 方法前判断是否清空二级缓存；执行 query 方法前先在二级缓存中查询，命中失败再通过被代理类查询。

使用场景及默认执行器：

默认执行器是SimpleExecutor，Executor的这些特点，都严格限制在SqlSession生命周期范围内。在创建SqlSession对象的时候指定执行器的类型即可。ReuseExecutor和BatchExecutor适用特定场景的sql执行使用，但是必须自己维护Statement对象，执行SqlSession的flushStatements来清除缓存不推荐使用。而CachingExecutor通常来说会配合redis等第三方储存来实现分布式二级缓存。



* MyBatis的缓存有什么作用？一级和二级缓存有何区别？

简单理解来说：目的就是提升查询的效率和减少数据库的压力 。

　 一级缓存：它指的是Mybatis中sqlSession对象的缓存，当我们执行查询以后，查询的结果会同时存入到SqlSession为我们提供的一块区域中，该区域的结构是一个Map，当我们再次查询同样的数据，mybatis会先去sqlsession中查询是否有，的话直接拿出来用，当SqlSession对象消失时，mybatis的一级缓存也就消失了，同时一级缓存是SqlSession范围的缓存，当调用SqlSession的修改、添加、删除、commit(),close等

方法时，就会清空一级缓存。

　　二级缓存：他值得是Mybatis中SqlSessionFactory对象的缓存，由同一个SqlSessionFactory对象创建的SqlSession共享其缓存，但是其中缓存的是数据而不是对象，所以从二级缓存再次查询出得结果的对象与第一次存入的对象是不一样的。

* Mybatis是否支持延迟加载？如果支持，它的实现原理是什么？

Mybatis仅支持association关联对象和collection关联集合对象的延迟加载，association指的就是多对一，collection指的就是一对多查询。

在Mybatis配置文件中，可以配置是否启用延迟加载lazyLoadingEnabled=true|false。

它的原理是，使用CGLIB创建目标对象的代理对象，当调用目标方法时，进入拦截器方法，比如调用a.getB().getName()，拦截器invoke()方法发现a.getB()是null值，那么就会单独发送事先保存好的查询关联B对象的sql，把B查询上来，然后调用a.setB(b)，于是a的对象b属性就有值了，接着完成a.getB().getName()方法的调用。

1. **跟据以下提纲预习Spring相关内容**

* Spring是什么？

Spring是一个开放源代码的设计层面框架，他解决的是业务逻辑层和其他各层的松耦合问题，因此它将面向接口的编程思想贯穿整个系统应用。“耦合”一般指软件组件之间的依赖程度。

* Spring的IOC为我们解决了什么问题？

　IoC只是一种思想，一个重要的面向对象编程的法则，它能指导我们如何设计出松耦合、更优良的程序。有了IoC容器后，把创建和查找依赖对象的控制权交给了容器，由容器进行注入组合对象，所以对象与对象之间是 松散耦合，这样也方便测试，利于功能复用，更重要的是使得程序的整个体系结构变得非常灵活。在IoC/DI思想中，应用程序变成被动的了，被动的等待IoC容器来创建并注入它所需要的资源。

* Spring IOC的注入方式有哪几种？

1、Setter方法注入

Setter方法注入是容器通过调用无参构造器或无参static工厂 方法实例化bean之后，调用该bean的setter方法，即实现了基于setter的依赖注入。

2、构造方法注入

构造器依赖注入通过容器触发一个类的构造器来实现的，该类有一系列参数，每个参数代表一个对其他类的依赖。

3、P命名空间注入

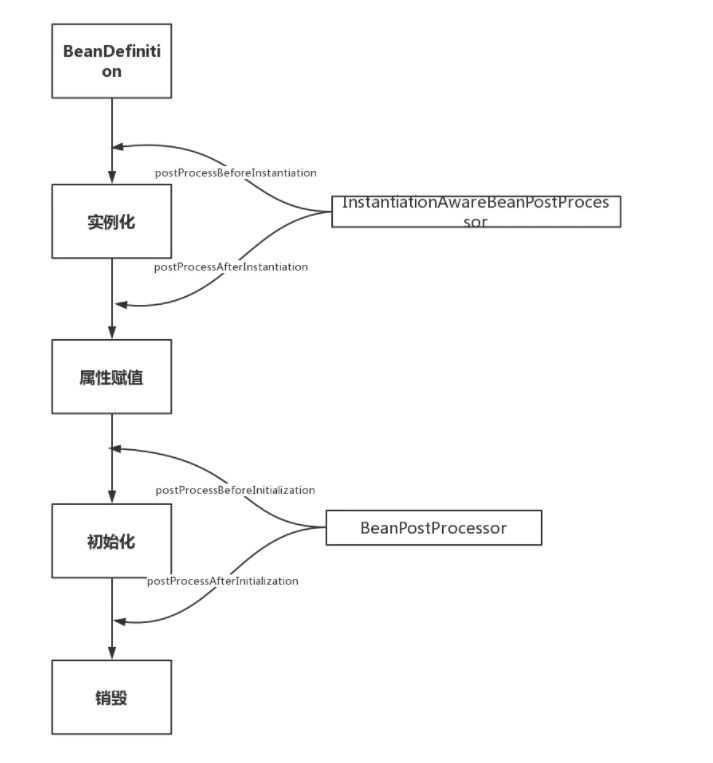
* Spring Bean的生命周期？

实例化 Instantiation

属性赋值 Populate

初始化 Initialization

销毁 Destruction



* Spring中bean的作用域？



1. 当一个bean的作用域为Singleton，那么Spring IoC容器中只会存在一个共享的bean实例，并且所有对bean的请求，只要id与该bean定义相匹配，则只会返回bean的同一实例。Singleton是单例类型，就是在创建起容器时就同时自动创建了一个bean的对象，不管你是否使用，他都存在了，每次获取到的对象都是同一个对象。注意，Singleton作用域是Spring中的缺省作用域。
2. 当一个bean的作用域为Prototype，表示一个bean定义对应多个对象实例。Prototype作用域的bean会导致在每次对该bean请求（将其注入到另一个bean中，或者以程序的方式调用容器的getBean()方法）时都会创建一个新的bean实例。Prototype是原型类型，它在我们创建容器的时候并没有实例化，而是当我们获取bean的时候才会去创建一个对象，而且我们每次获取到的对象都不是同一个对象。根据经验，对有状态的bean应该使用prototype作用域，而对无状态的bean则应该使用singleton作用域。
3. 当一个bean的作用域为Session，表示在一个HTTP Session中，一个bean定义对应一个实例。该作用域仅在基于web的Spring ApplicationContext情形下有效。
4. 当一个bean的作用域为Request，表示在一次HTTP请求中，一个bean定义对应一个实例；即每个HTTP请求都会有各自的bean实例，它们依据某个bean定义创建而成。该作用域仅在基于web的Spring ApplicationContext情形下有效。
5. 当一个bean的作用域为Global Session，表示在一个全局的HTTP Session中，一个bean定义对应一个实例。典型情况下，仅在使用portlet context的时候有效。该作用域仅在基于web的Spring ApplicationContext情形下有效。

* Spring框架中的Bean是线程安全的么？如果线程不安全，那么如何处理？

Spring容器本身没有提供Bean的线程安全策略，因此，也可以说Spring容器中的bean不是线程安全的。

解决方式：

但是如果要保证线程安全，可以将bean的作用域改为prototype，比如像Model View。另外还可以采用ThreadLocal来解决线程安全问题。ThreadLocal为每个线程保存一个副本变量，每个线程只操作自己的副本变量。

* Spring提供了哪几种自动装配方式？

no：默认的方式是不进行自动装配，通过显式设置 ref 属性来进行装配。

byName：通过参数名 自动装配，Spring 容器在配置文件中发现 bean的 autowire 属性被设置成 byname，之后容器试图匹配、装配和该 bean 的属性具有相同名字的 bean。

byType:：通过参数类型自动装配，Spring 容器在配置文件中发现 bean的 autowire 属性被设置成 byType，之后容器试图匹配、装配和该 bean 的属性具有相同类型的 bean。如果有多个 bean 符合条件，则抛出错误。

constructor：这个方式类似于 byType， 但是要提供给构造器参数，如果没有确定的带参数的构造器参数类型，将会抛出异常。

autodetect：首先尝试使用 constructor 来自动装配，如果无法工作，则使用 byType 方式。

1. **回顾预科学习的HTML、CSS和JS，完成12306登录和注册页面**

