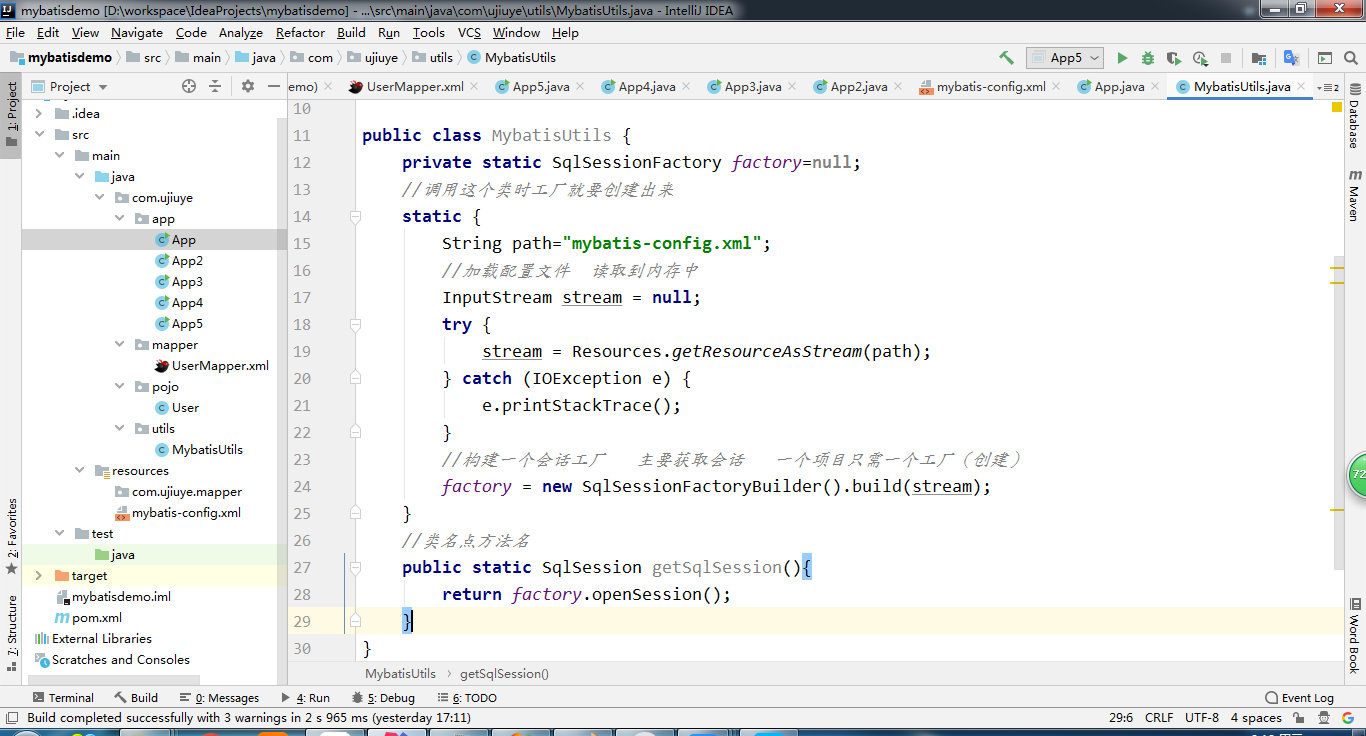
# 第三讲MyBatis（上）

### （二）使用抽取的工具类完成操作

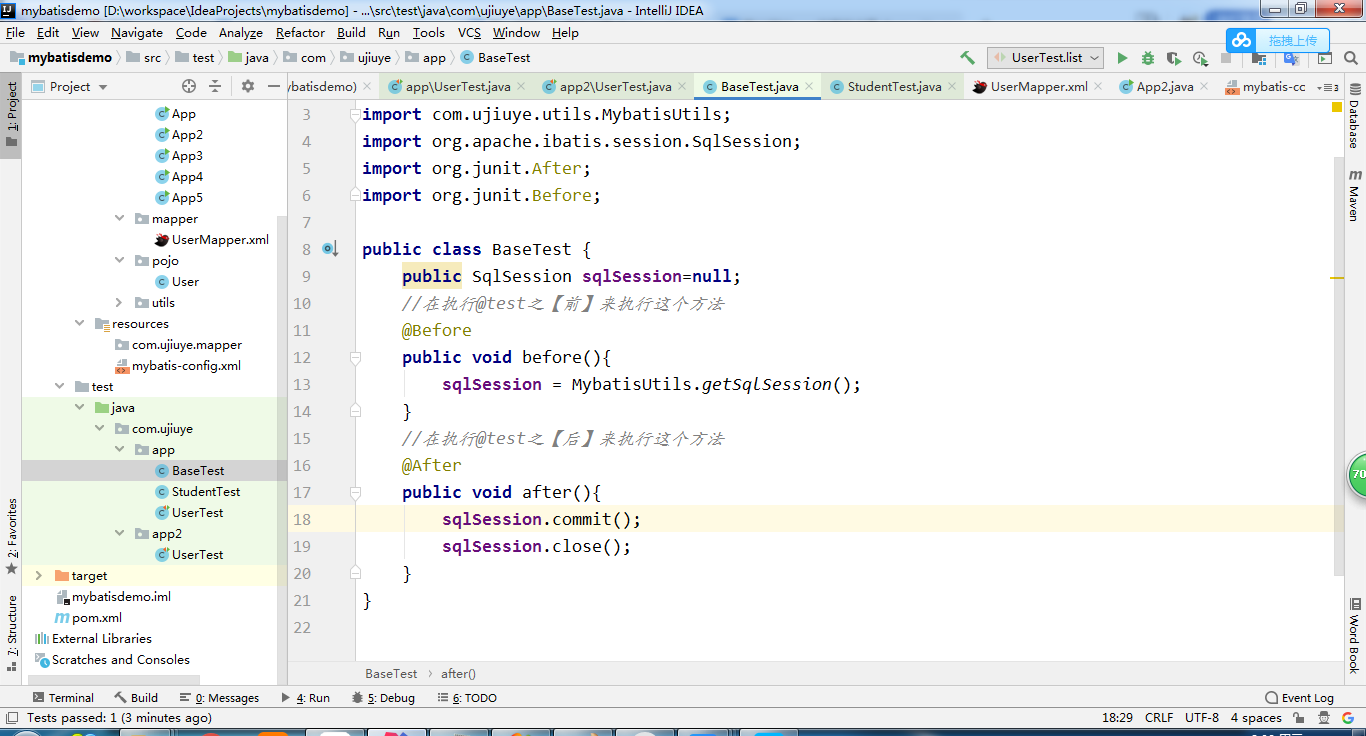
1、制作统一的工具类

新建包：utils,新建工具类，MybatisUtils



2、抽取测试类代码基类

新建BaseTest

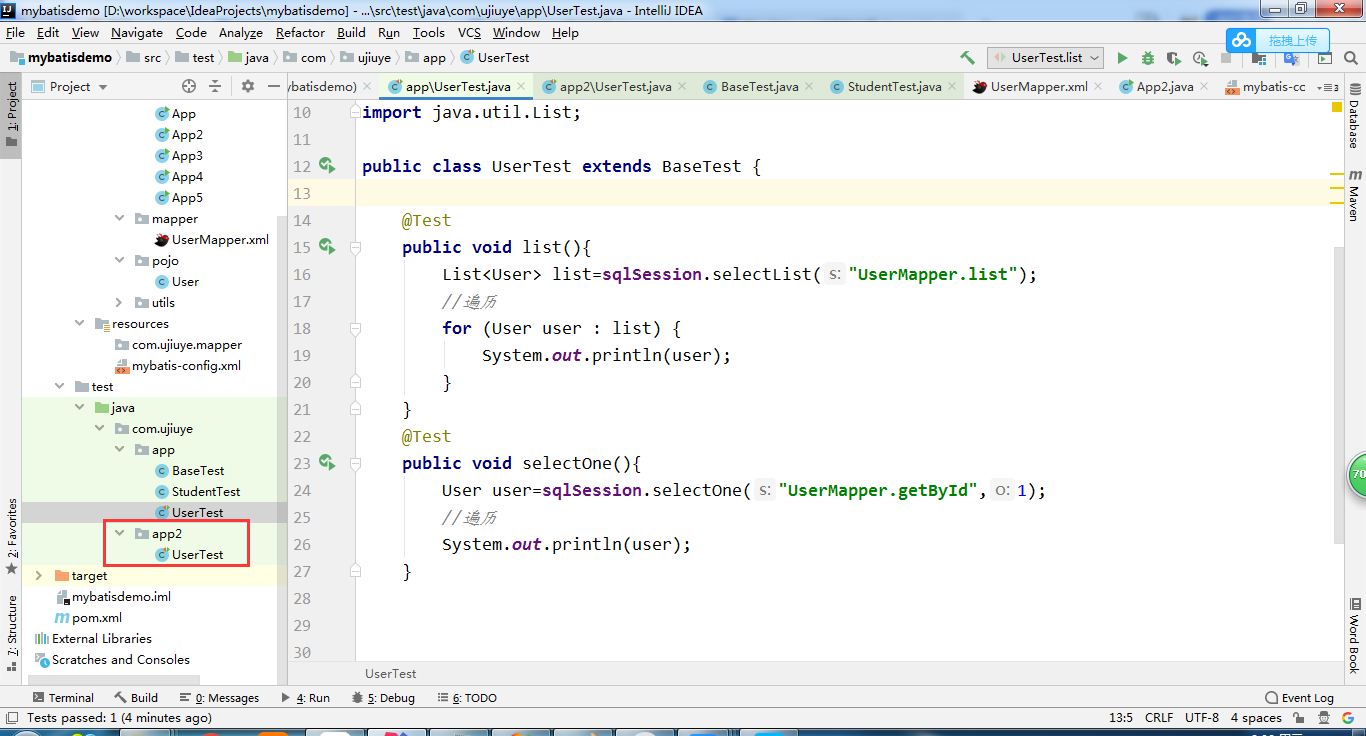


3、使用工具类完成操作

添加依赖

|  |
| --- |
| <**dependency**>  <**groupId**>junit</**groupId**>  <**artifactId**>junit</**artifactId**>  <**version**>4.13</**version**> </**dependency**> |

新建测试类



### （三） MyBatis核心配置文件详解

1、MyBatis核心配置文件层级关系



2、environments标签

MyBatis 可以配置成适应多种环境，这种机制有助于将 SQL 映射应用于多种数据库之中， 现实情况下有多种理由需要这么做。例如，开发、测试和生产环境需要有不同的配置；或者想在具有相同 Schema 的多个生产数据库中使用相同的 SQL 映射。还有许多类似的使用场景。

不过要记住：尽管可以配置多个环境，但每个 SqlSessionFactory 实例只能选择一种环境。

所以，如果你想连接两个数据库，就需要创建两个 SqlSessionFactory 实例，每个数据库对应一个。而如果是三个数据库，就需要三个实例，依此类推，记起来很简单：

每个数据库对应一个 SqlSessionFactory 实例为了指定创建哪种环境，只要将它作为可选的参数传递给 SqlSessionFactoryBuilder 即可。可以接受环境配置的两个方法签名是：

SqlSessionFactory factory = new SqlSessionFactoryBuilder().build(reader, environment);

SqlSessionFactory factory =

new SqlSessionFactoryBuilder().build(reader, environment,properties);

如果忽略了环境参数，那么将会加载默认环境，如下所示：

SqlSessionFactory factory = new SqlSessionFactoryBuilder().build(reader);

SqlSessionFactory factory = new SqlSessionFactoryBuilder().build(reader, properties);

**事务管理器（transactionManager）**

在 MyBatis 中有两种类型的事务管理器（也就是 type="[JDBC|MANAGED]"）：

JDBC – 这个配置直接使用了 JDBC 的提交和回滚设施，它依赖从数据源获得的连接来管理事务作用域。

MANAGED – 这个配置几乎没做什么。它从不提交或回滚一个连接，而是让容器来管理事务的整个生命周期（比如 JEE 应用服务器的上下文context）。

如果你正在使用 Spring + MyBatis，则没有必要配置事务管理器，因为 Spring 模块会使用自带的管理器来覆盖前面的配置。

**数据源（dataSource）**

**UNPOOLED**– (没有用连接池)这个数据源的实现会每次请求时打开和关闭连接。

**POOLED**– (用连接池)这种数据源的实现利用“池”的概念将 JDBC 连接对象组织起来，避免了创建新的连接实例时所必需的初始化和认证时间。 这种处理方式很流行，能使并发 Web 应用快速响应请求。

3、mappers标签

既然 MyBatis 的行为已经由上述元素配置完了，我们现在就要来定义 SQL 映射语句了。 但首先，我们需要告诉 MyBatis 到哪里去找到这些语句。 在自动查找资源方面，Java 并没有提供一个很好的解决方案，所以最好的办法是直接告诉 MyBatis 到哪里去找映射文件。

<!-- 使用相对于类路径的资源引用 -->

<mappers>

<mapper resource="org/mybatis/builder/AuthorMapper.xml"/>

<mapper resource="org/mybatis/builder/BlogMapper.xml"/>

<mapper resource="org/mybatis/builder/PostMapper.xml"/>

</mappers>

<!-- 使用完全限定资源定位符（URL） 了解 -->

<mappers>

<mapper url="file:///var/mappers/AuthorMapper.xml"/>

<mapper url="file:///var/mappers/BlogMapper.xml"/>

<mapper url="file:///var/mappers/PostMapper.xml"/>

</mappers>

<!-- 使用映射器【接口】实现类的完全限定类名 -->

<mappers>

<mapper class="org.mybatis.builder.AuthorMapper"/>

<mapper class="org.mybatis.builder.BlogMapper"/>

<mapper class="org.mybatis.builder.PostMapper"/>

</mappers>

<!-- 将包内的映射器【接口】实现全部注册为映射器 【推荐】-->

<mappers>

<package name="org.mybatis.builder"/>

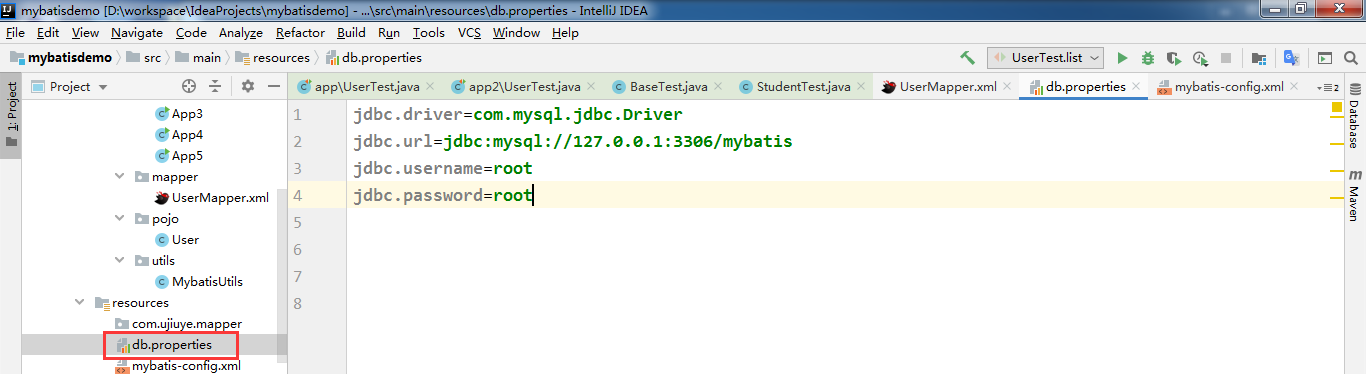
</mappers>

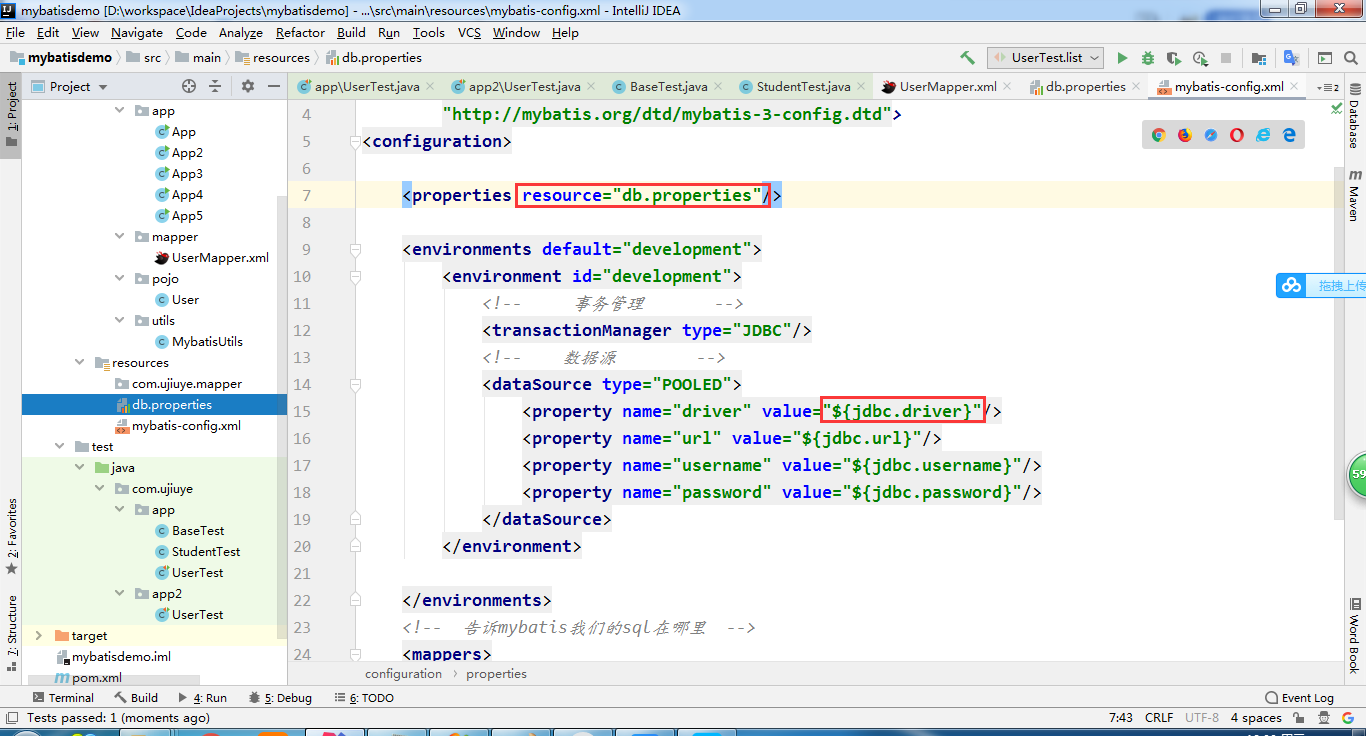
确保包中的所有映射文件都是正确（没有错），如果有错会导程序运行不了。

4、properties标签

一般我们开发时会把数据库信息提取出来。写到一个文件，方便维护。

db.properties





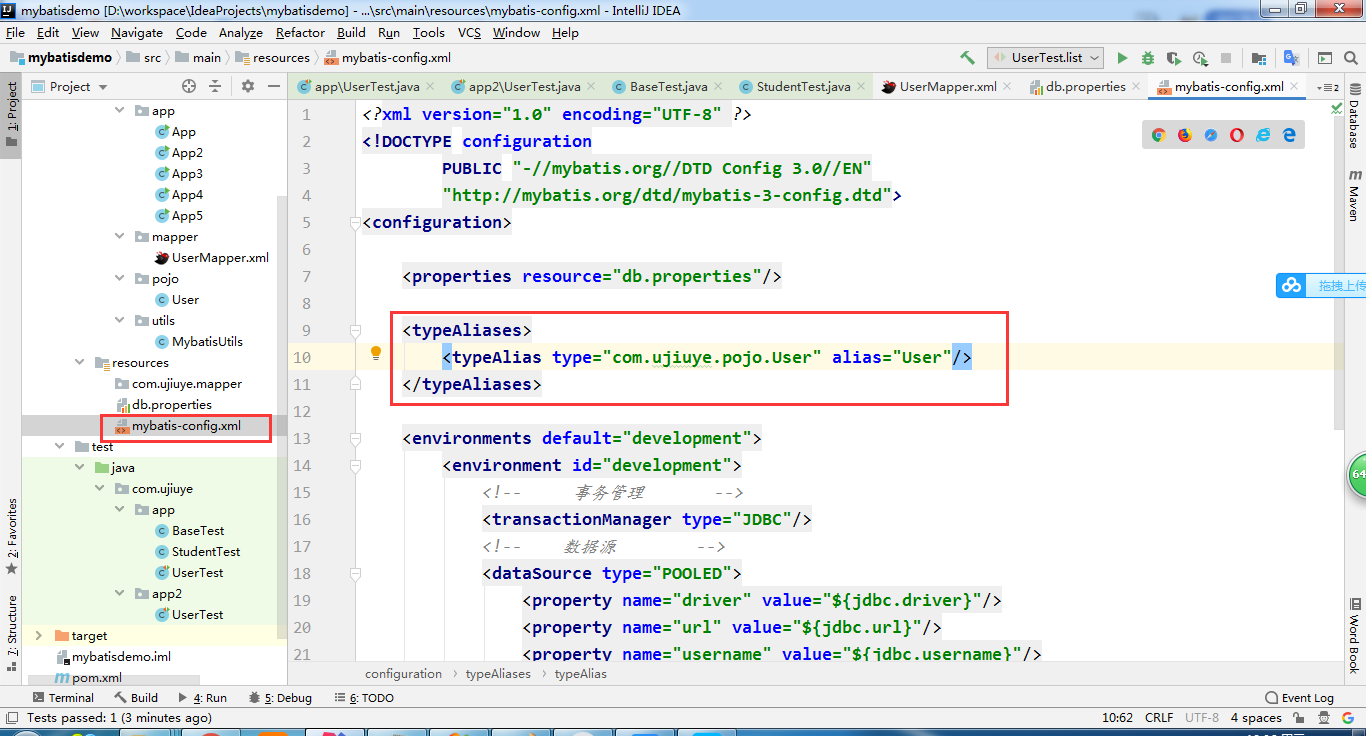
5、typeAliases标签

类型别名可为 Java 类型设置一个缩写名字。 它仅用于 XML 配置，意在降低冗余的全限定类名书写。

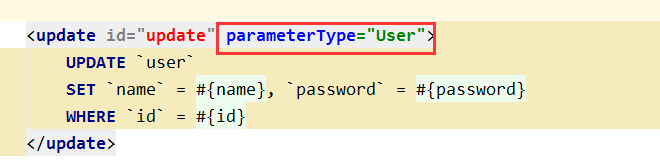
昨天我们写代码时，这个类是要写全限定类名



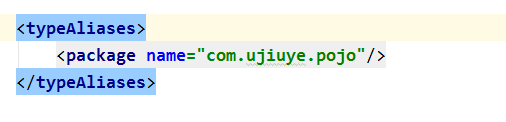
为User类起别名



在映射文件中使用别名

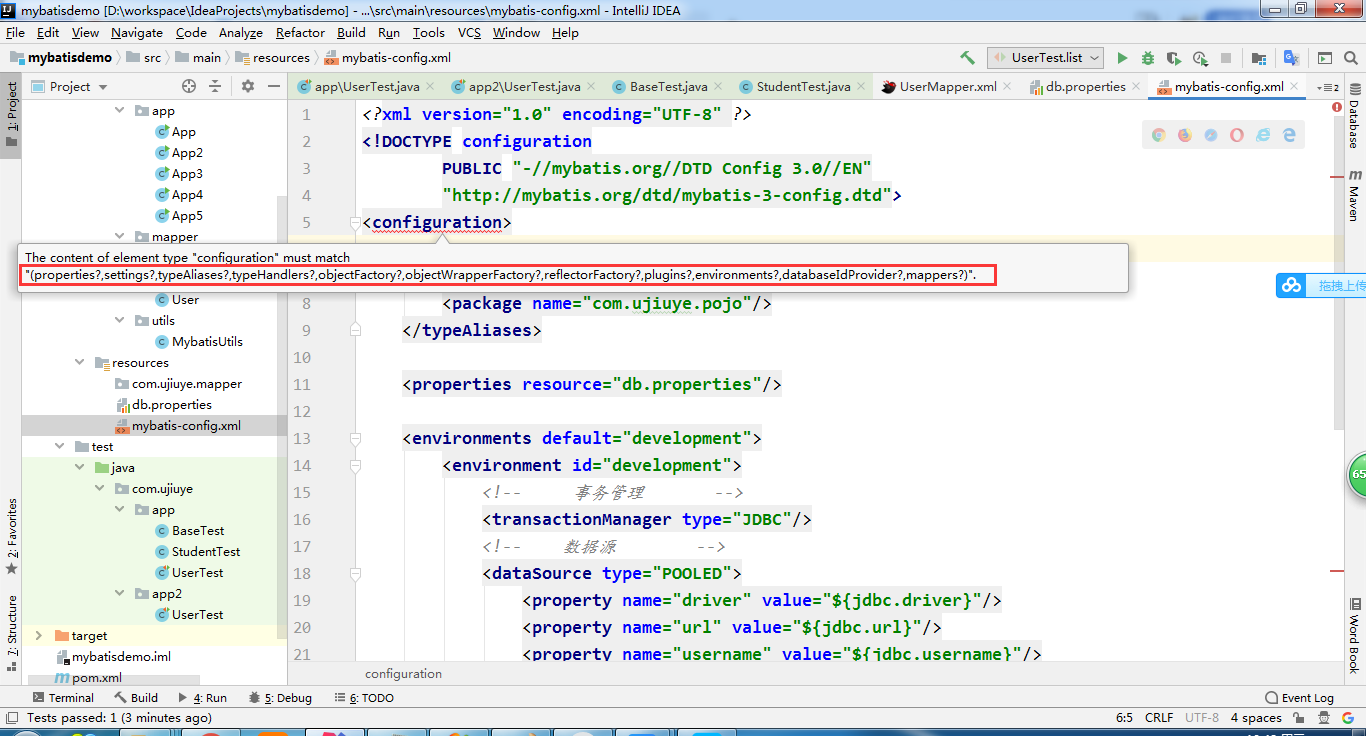


直接描包（推荐）



别名就是类名

注意：配置文件的书写是有顺序的

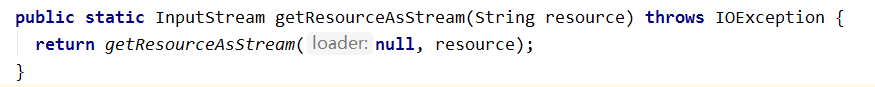


### （四） MyBatis核心API概述

1、Resources

Resources是用来从类路径下加载资源文件。

这里我们使用了 Resources 工具类，这个类在 org.apache.ibatis.io 包中。Resources 类正如其名，会帮助你从类路径下、文件系统或一个 web URL 中加载资源文件。在略读该类的源代码或用 IDE 查看该类信息后，你会发现一整套相当实用的方法。



2、SqlSessionFactoryBuilder

SqlSessionFactoryBuilder是用来创建 SqlSessionFactory。

SqlSessionFactory 本身是由 SqlSessionFactoryBuilder 创建的，它可以从 XML、注解或 Java 配置代码来创建 SqlSessionFactory。

SqlSessionFactoryBuilder 有五个 build() 方法，每一种都允许你从不同的资源中创建一个 SqlSessionFactory 实例。

SqlSessionFactory build(InputStream inputStream)

SqlSessionFactory build(InputStream inputStream, String environment)

3、SqlSessionFactory

SqlSessionFactory是用来创建SqlSession

SqlSession openSession()

SqlSession openSession(boolean autoCommit)

SqlSession openSession(Connection connection)

默认的 openSession() 方法没有参数，它会创建具备如下特性的 SqlSession：

事务作用域将会开启（也就是不自动提交）。

将由当前环境配置的 DataSource 实例中获取 Connection 对象。

事务隔离级别将会使用驱动或数据源的默认设置。

预处理语句不会被复用，也不会批量处理更新。

SqlSessionFactory是MyBatis的关键对象,它是个单个数据库映射关系经过编译后的内存镜像.SqlSessionFactory对象的实例可以通过SqlSessionFactoryBuilder对象类获得,而SqlSessionFactoryBuilder则可以从XML配置文件或一个预先定制的Configuration的实例构建出SqlSessionFactory的实例.每一个MyBatis的应用程序都以一个SqlSessionFactory对象的实例为核心.同时SqlSessionFactory也是线程安全的,SqlSessionFactory一旦被创建,应该在应用执行期间都存在.在应用运行期间不要重复创建多次,建议使用单例模式.

一个程序对应一个SqlSessionFactory，生命周期是跟随程序

4、 SqlSession

正如之前所提到的，SqlSession 在 MyBatis 中是非常强大的一个类。它包含了所有执行语句、提交或回滚事务以及获取映射器实例的方法。

<T> T selectOne(String statement, Object parameter)

<E> List<E> selectList(String statement, Object parameter)

int insert(String statement, Object parameter)

int update(String statement, Object parameter)

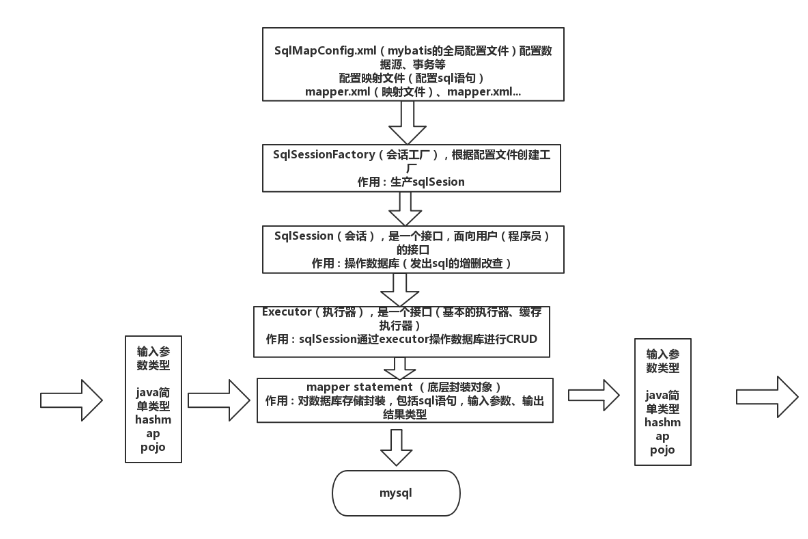
int delete(String statement, Object parameter)

selectOne 和 selectList 的不同仅仅是 selectOne 必须返回一个对象或 null 值。如果返回值多于一个，就会抛出异常。如果你不知道返回对象会有多少，请使用 selectList。

每个线程都应该有它自己的SqlSession实例.SqlSession的实例不能被共享,同时SqlSession也是线程不安全的,绝对不能讲SqlSeesion实例的引用放在一个类的静态字段甚至是实例字段中.也绝不能将SqlSession实例的引用放在任何类型的管理范围中,比如Servlet当中的HttpSession对象中.使用完SqlSeesion之后关闭Session很重要,应该确保使用finally块来关闭它.

### MyBatis的执行原理（面试题）

背



详细流程如下：

1、加载mybatis全局配置文件（数据源、mapper映射文件等），解析配置文件，MyBatis基于XML配置文件生成Configuration，和一个个MappedStatement（包括了参数映射配置、动态SQL语句、结果映射配置），其对应着<select | update | delete | insert>标签项。

2、SqlSessionFactoryBuilder通过Configuration对象生成SqlSessionFactory，用来开启SqlSession。

3、SqlSession对象完成和数据库的交互：

（1）用户程序调用mybatis接口层api（即Mapper接口中的方法）

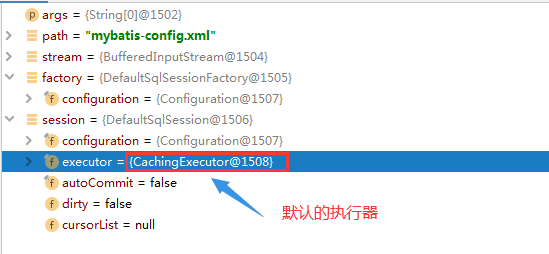
（2）SqlSession通过调用api的Statement ID找到对应的MappedStatement对象

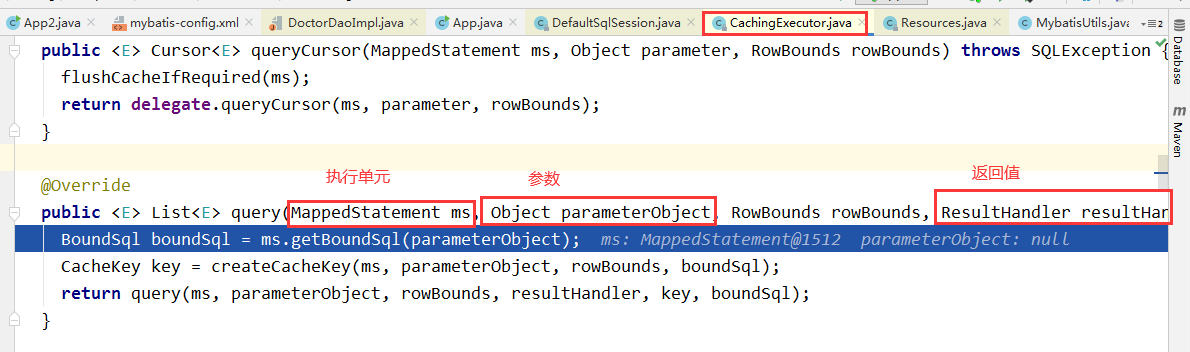
（3）通过Executor（负责动态SQL的生成和查询缓存的维护）将MappedStatement对象进行解析，sql参数转化、动态sql拼接，生成jdbc Statement对象

（4）JDBC执行sql。

（5）借助MappedStatement中的结果映射关系，将返回结果转化成HashMap、JavaBean等存储结构并返回。







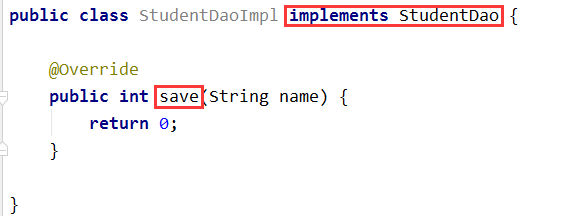
## MyBatis基于接口代理方式实现Dao层开发

### （一）基于接口代理方式实现Dao的开发(重点)

1、介绍

回顾：

接口》实现类

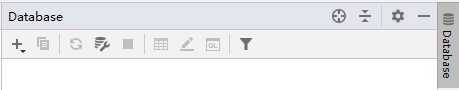


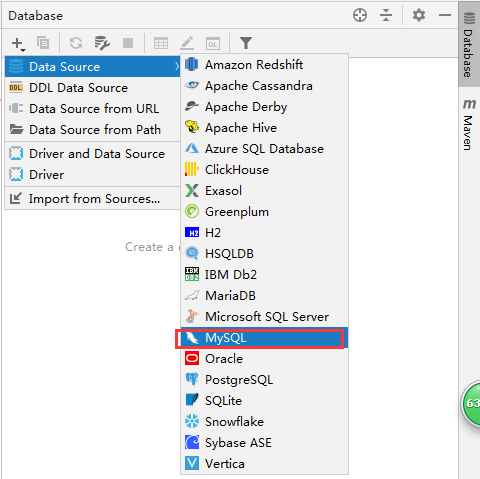
接口》映射文件

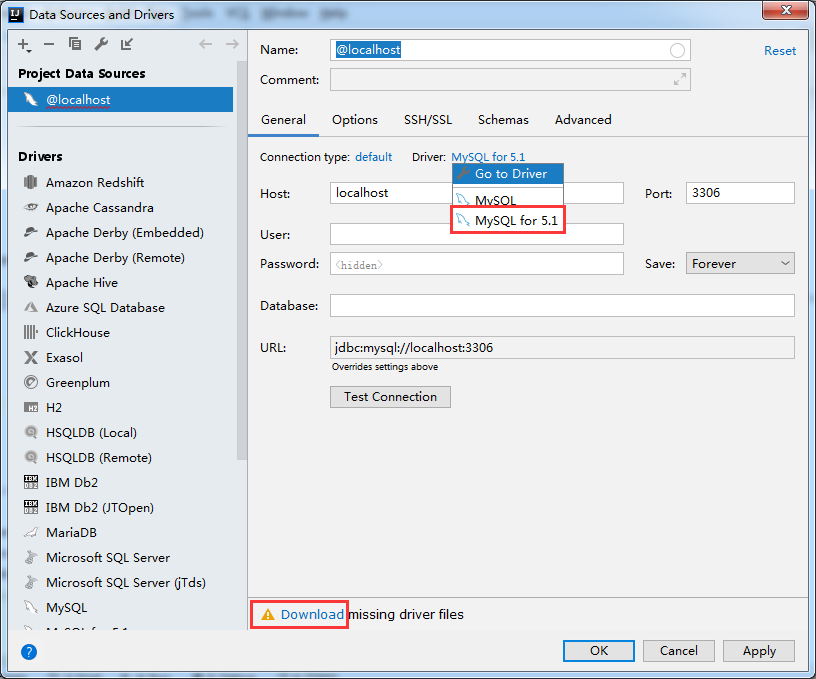
规则：

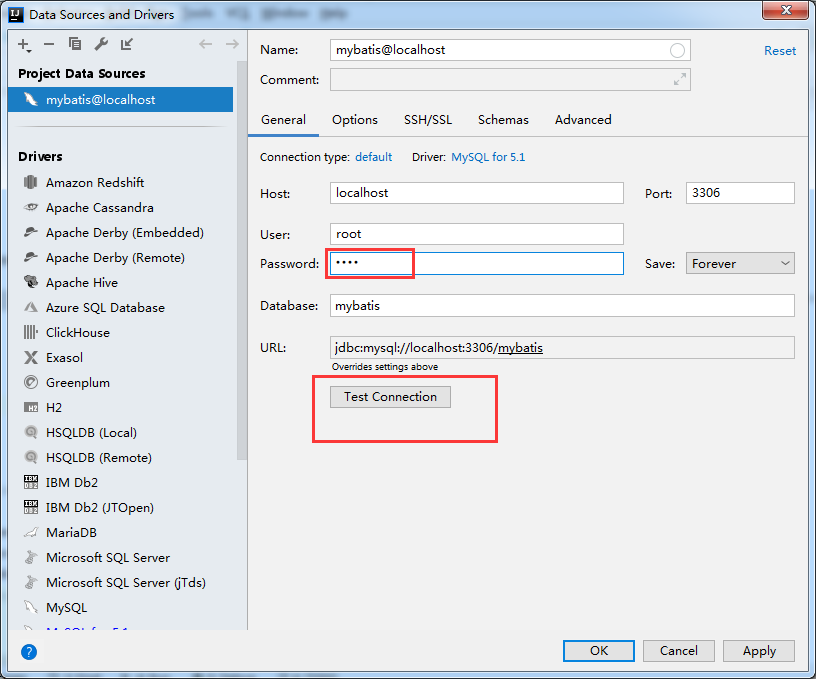
1. 接口文件名与映射文件名称要一致 （StudentDao）
2. 映射文件的namespace命名空间要写接口的全限定名 implements StudentDao
3. 执行单元的id要写接口中的方法名 （id的值，实现其对应方法）
4. 执行单元的参数类型与接口的参数类型一致
5. 执行单元的返回值类型与接口的返回值类型一致

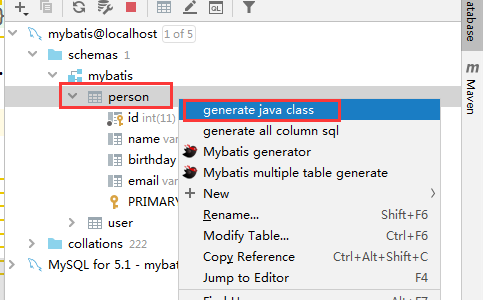
Idea database使用





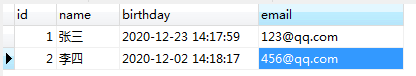






2、实现步骤

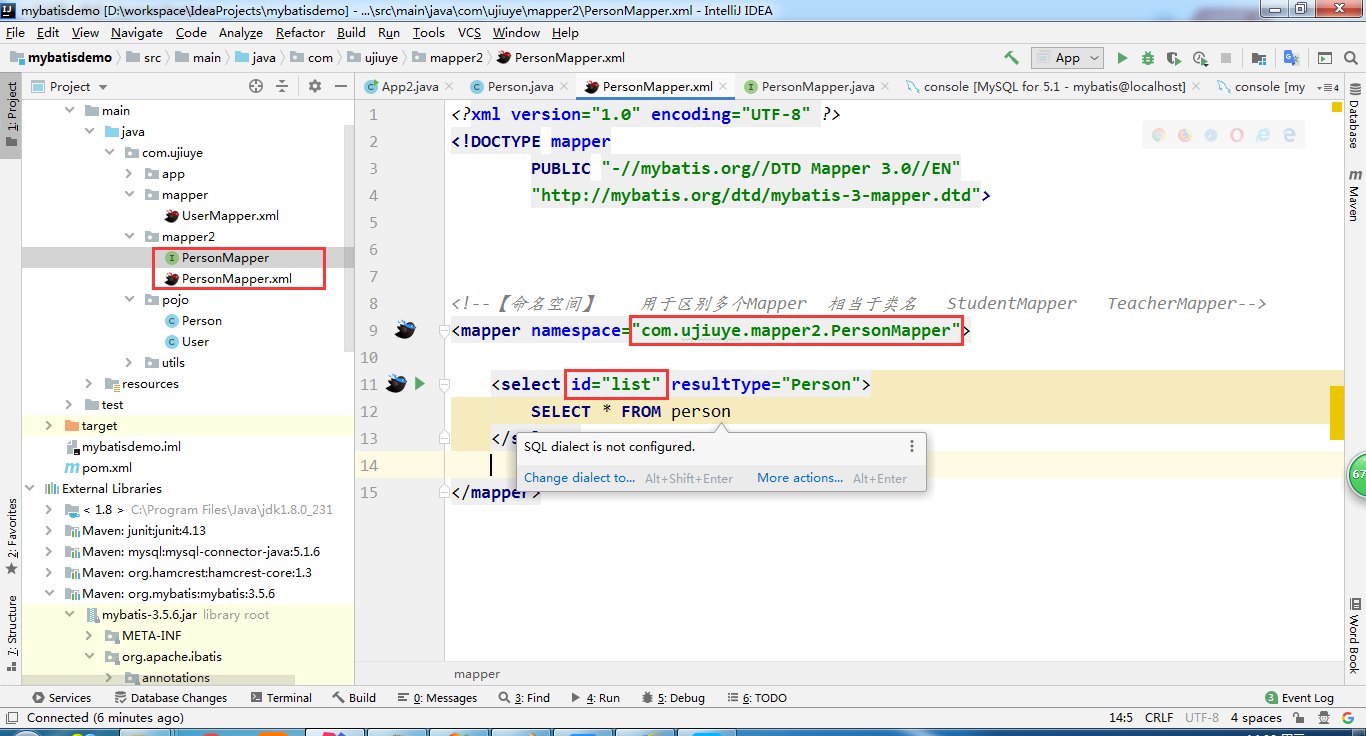
对person表进行CURD



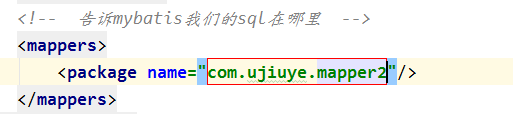
1. 新建实体类
2. 新建接口



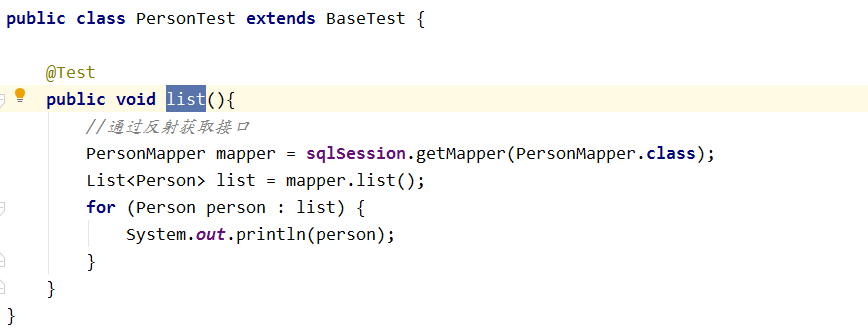
1. 新建映射文件 （实现类）



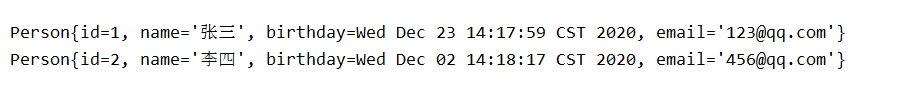
1. 修改配置文件



1. 测试

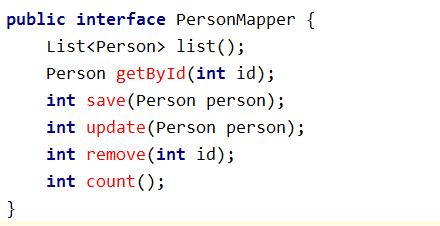


运行

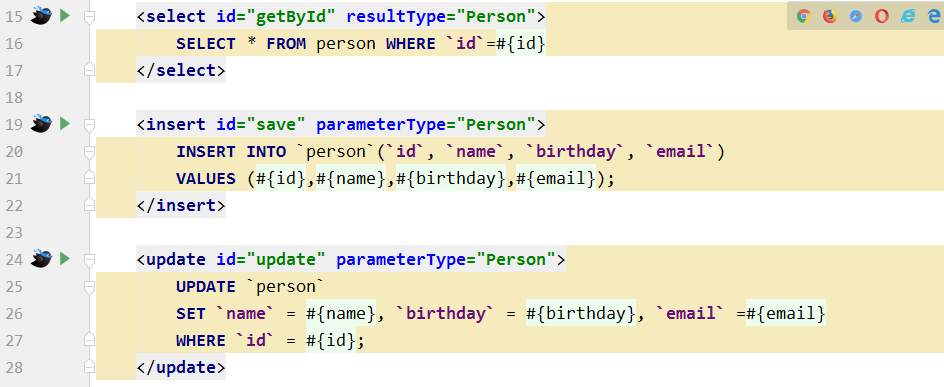


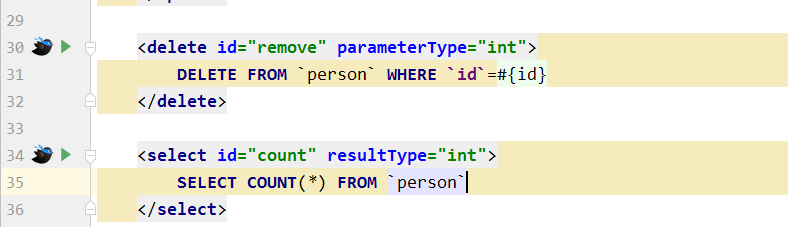
添加，删除，修改，查单个，统计

修改接口

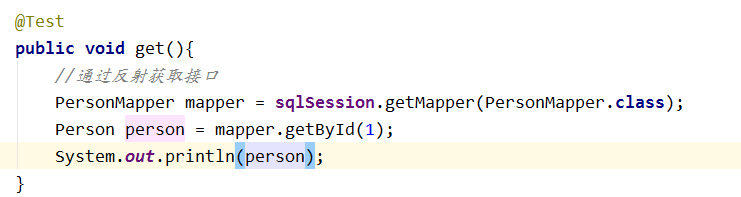


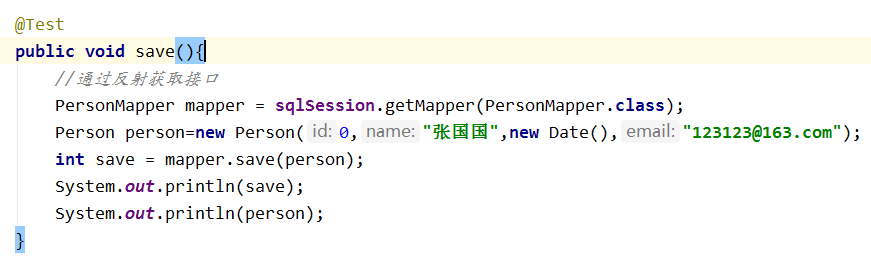
映射文件

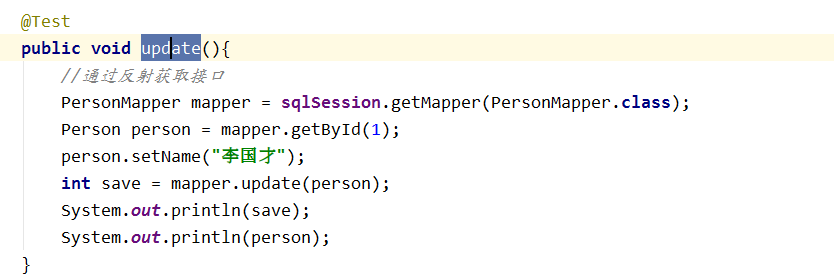


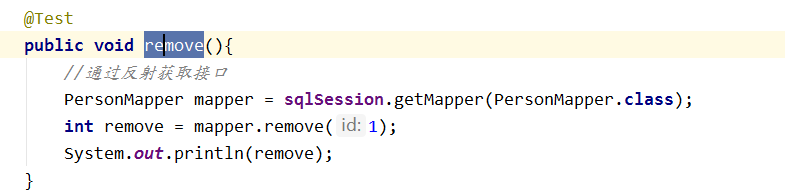


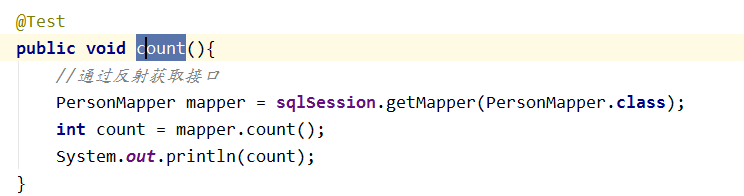
测试类







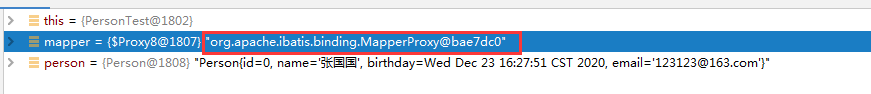




3、基于代理实现的内部执行原理

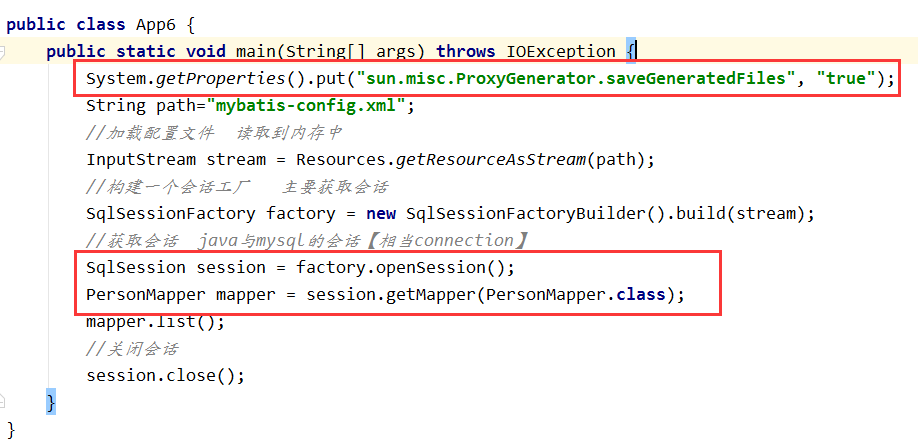
我们都接口是不工作的，但我们又没有写实现类，那么mybatis是怎么工作。

Mybatis自动为我们生成了一个实现类(代理类)----设计模式【动态代理模式】

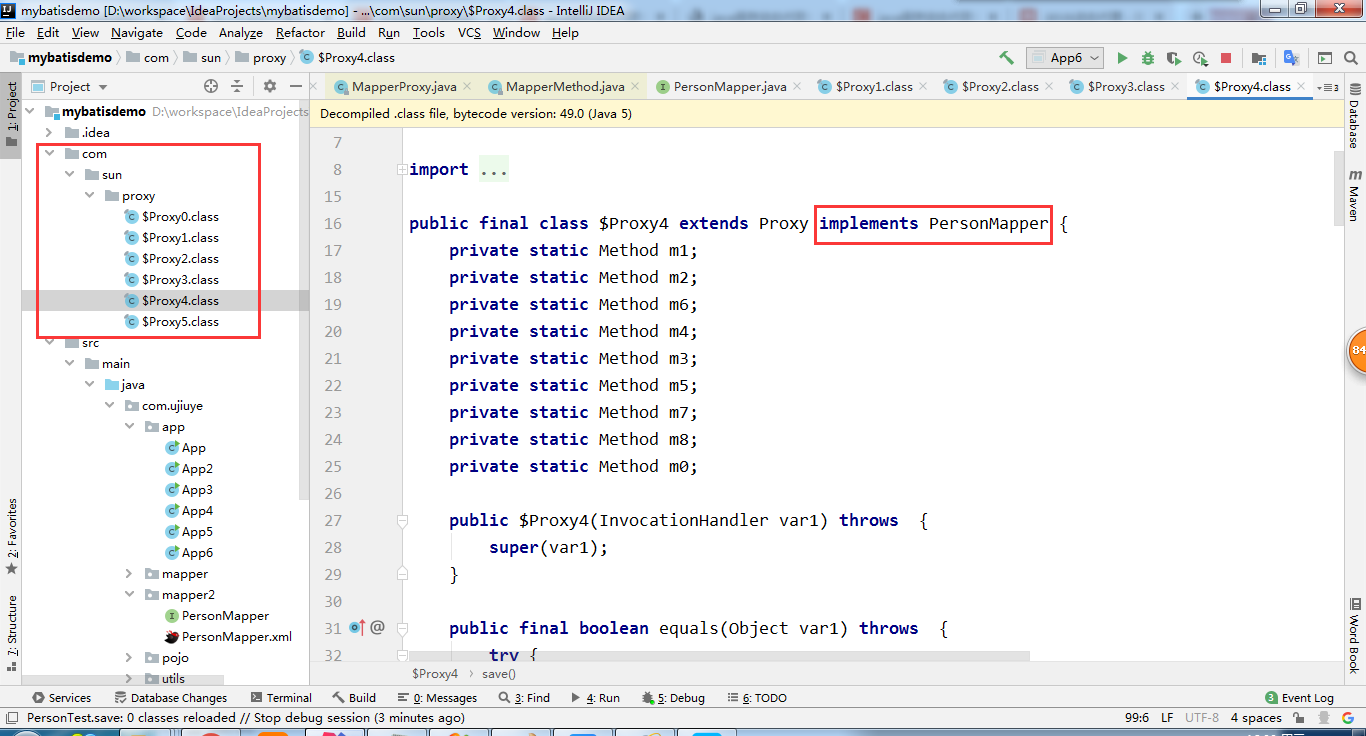


实现类（代理类）保存在内存中，运行完就释放

保存到本地：

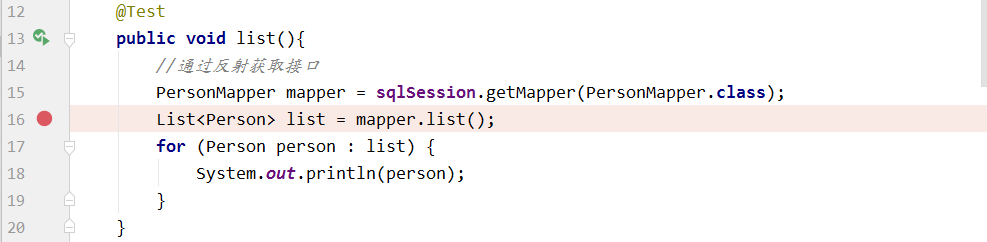


System.getProperties().put("sun.misc.ProxyGenerator.saveGeneratedFiles", "true");

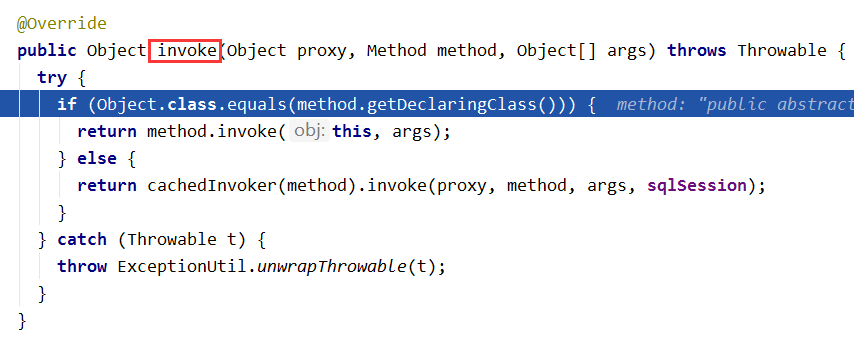


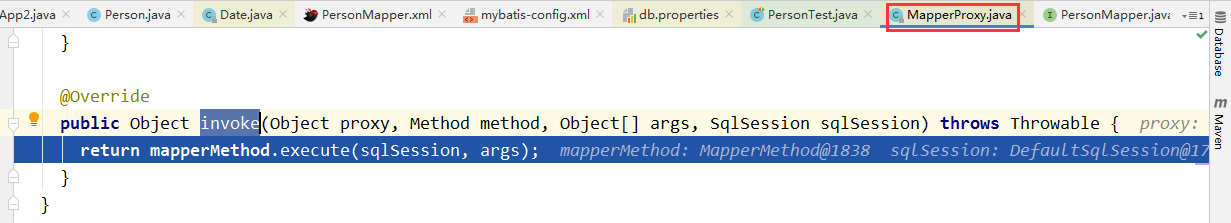
跟踪源码：

1、打断点

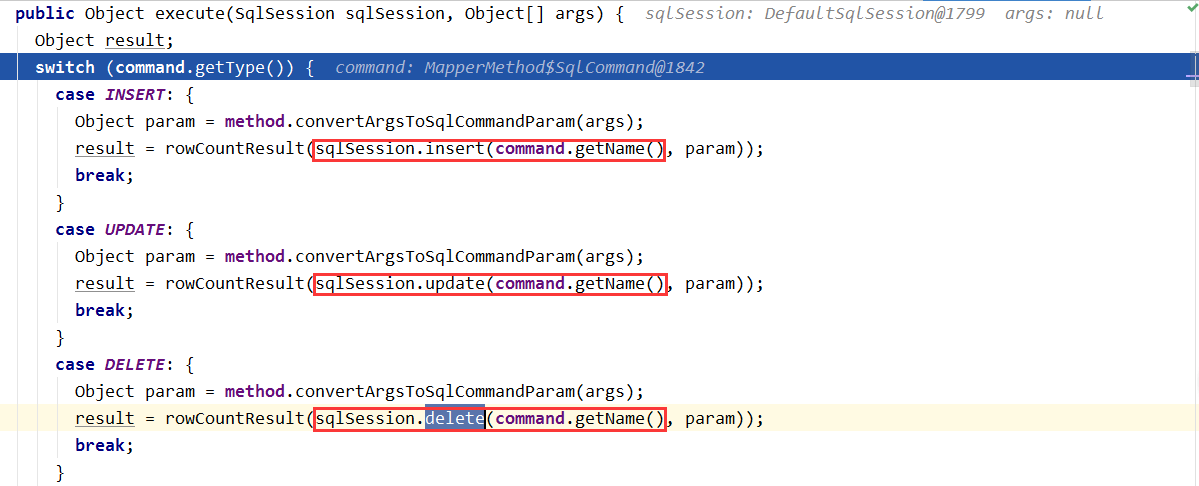


1. 可以发现工作是invoke（invoke）,这个方法是java动态代理的方法，说明了mybatis用了动态代理模式。





我们发现真正工作的还是sqlsession



### （二） MyBatis高级查询

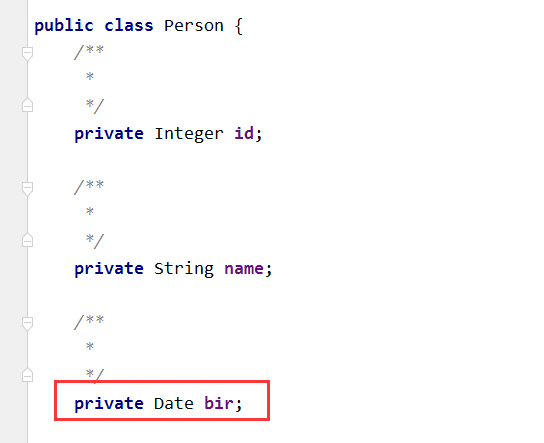
1、ResultMap标签

我们之前写的属性名都与表中的字段一一对应的，但是我们在开发中经常遇到不一致。

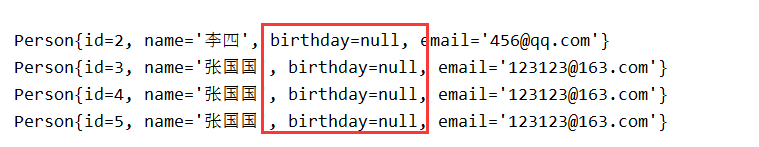
如：

表：birthday 属性bir

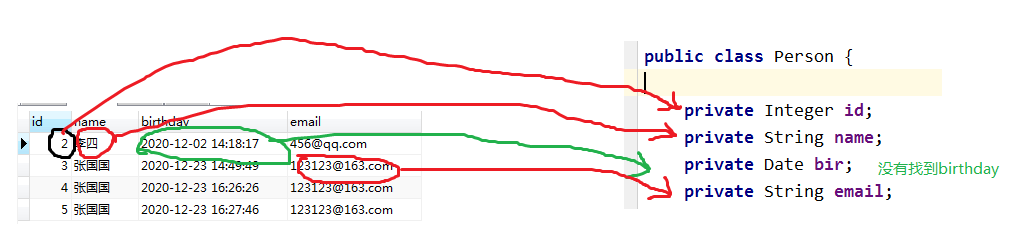
修改Person



重新运行

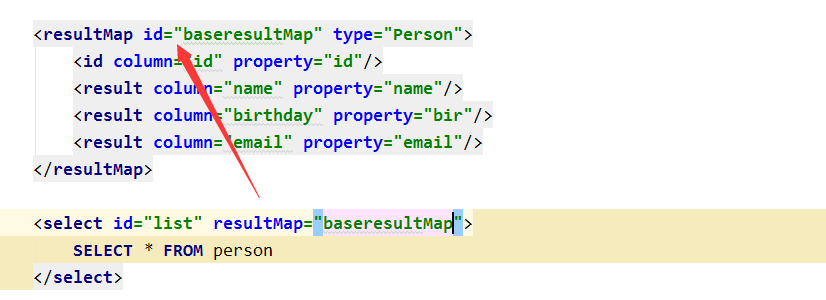


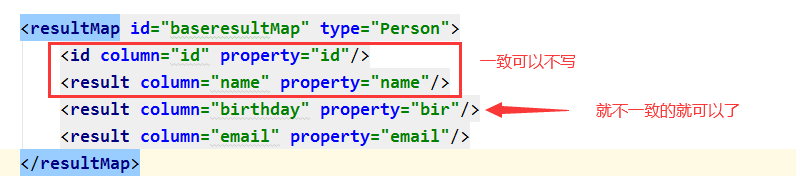
Bir与birthday没有对上。



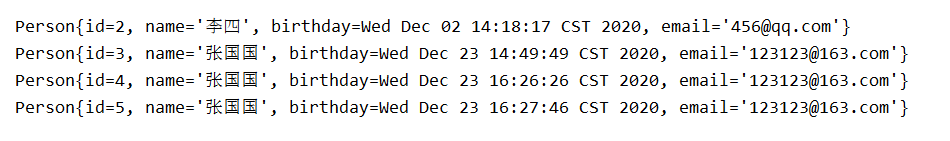
**resultType**以表中的字段名为条件去实体类中匹配。（字段名与属性名一致）

我们自定义一个映射规则：





重新运行



说明：

resultMap 自定义映射规则

Id 主键字段

Result 普通字段

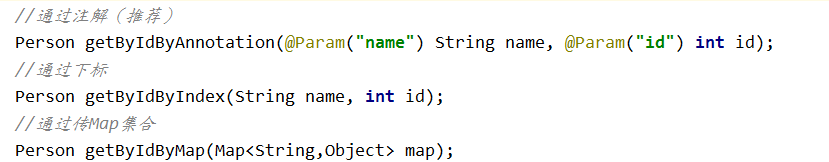
总结：

字段名与属性名一致时可以使用resultType和resultMap

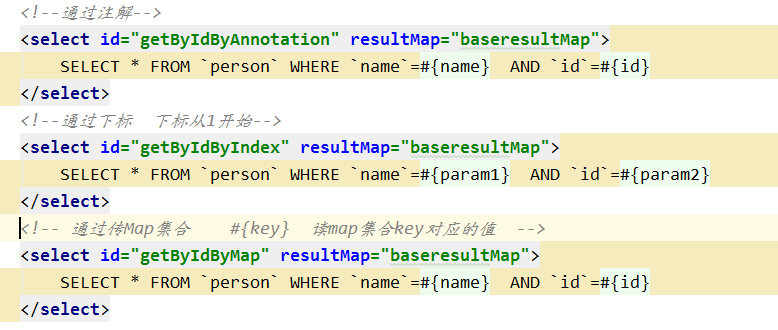
字段名与属性名【不】一致时只能使用resultMap

### （三） MyBatis多参数处理

1. 通过注解（推荐）
2. 通过下标
3. 通过传Map集合

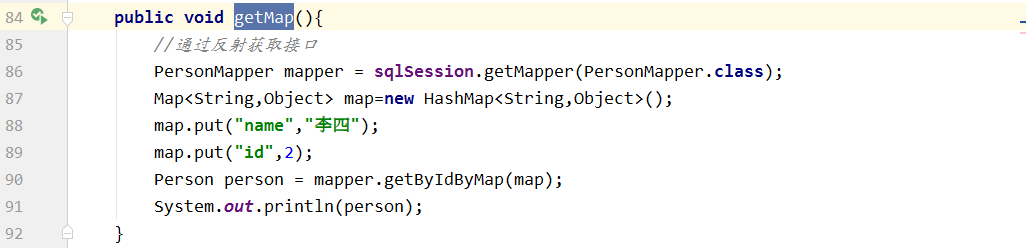


映射文件

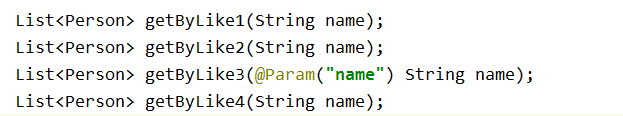


测试

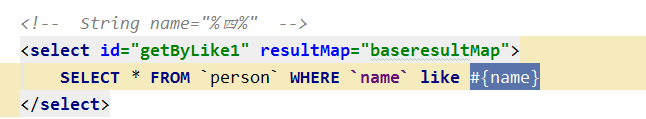




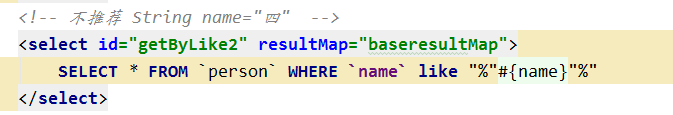
### 模糊查询



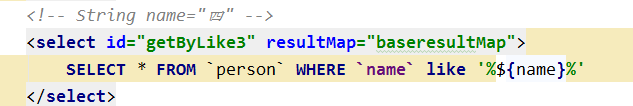
1、占位符方式#{name}



2、字符串拼接方式 "%"#{name}"%"

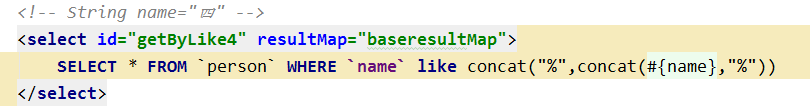


3、$的拼接方式'%${value}%'



如果方法没有写注解，那么这里只能写${value}，value是固定的

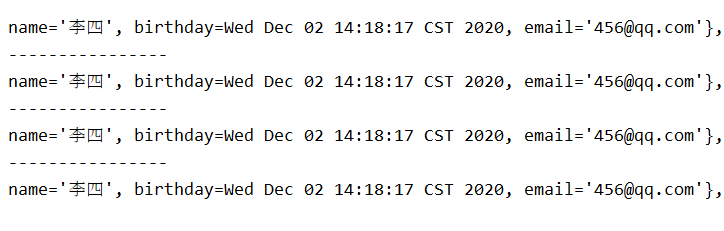
4、使用函数拼接数据 concat(concat('%',#{name}),'%');



测试



运行



总结：${} 与 #{} 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。

${} :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，会出现sql注入问题。

${} 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， ${} 括号中可以使任意名称。

Order by ${value}