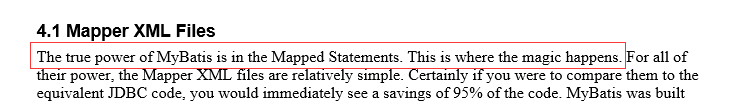
Sql语句的xml的映射文件

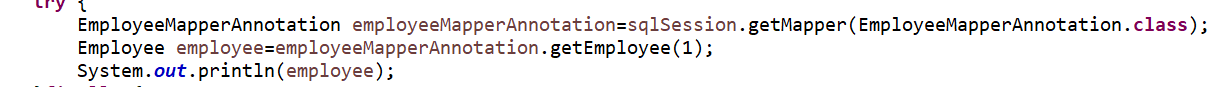


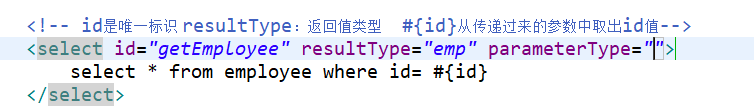
首先，看一下，官方文档的这句话，简直把这个xml文件描述的太牛掰了。所以这么牛掰的东西得好好看看唉。

标签

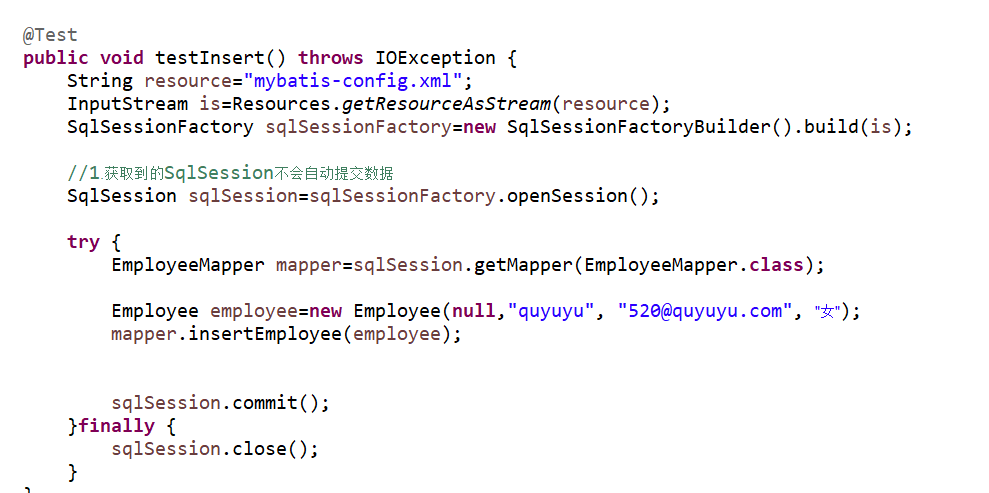
1.增、删、改、查

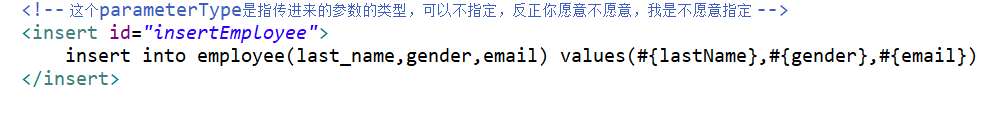
①查select



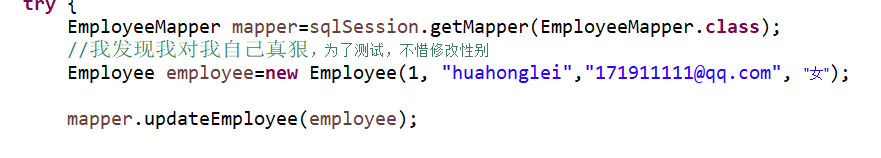


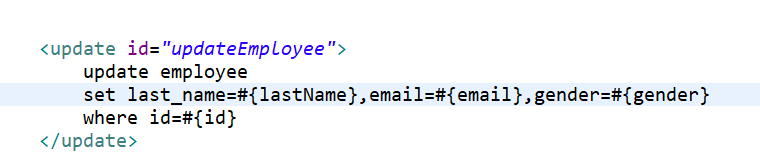
②增insert





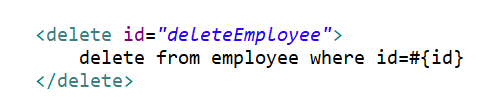
③改update





④删delete





2.Mybatis允许增删改直接定义以下类型返回值

Integer、Long、Boolean

另外，sqlSessionFactory.openSession()，这个版本是需要自己手动提交的，需要写commit

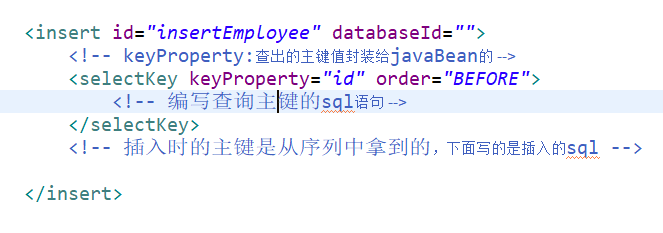
sqlSessionFactory.openSession(true)：这个重载的版本时可以自动提交的

3.、

Mysql支持自增，可以在insert那使用useGeneratedKeys=“true”，使用主键自增的策略，用keyProperty：指定对应的主键属性，也就是mybatis获取主键以后，将这个值封装给javaBean的哪个属性

Oracle不支持自增：Oracle使用序列来模拟自增

每次输入的数据的主键是从序列中拿到的值，如何获取到这个值



写到这的时候我的eclipse斜了，对的，斜了，整个的斜了，先不管了，第二天再解决这个问题吧，你还得问，，上面sql语句为毛线不写，，，好吧，，我能说我不会吗，，，这里先了解一下标签怎么使用吧

BEFORE：当前sql执行之前

AFTER：当前sql执行之后

Mybatis的参数处理

1.单个参数：mybatis不会做特殊处理

#{参数名}：取出参数值，，其实这个参数名随便你写，因为只有这一个参数，你不写参数名也无所谓

2.多个参数

你肯定是像这样搞（因为我也想这样搞）：

操纵：方法：public Employee getEmpByIdAndLastName（Integer id，String lastName）

取值：#{id},#{lastName}

但是会报如下错误



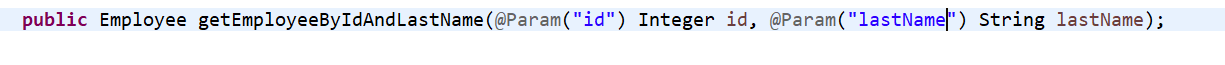
原因是，mybaits遇见多个参数会做特殊处理，多个参数会被封装成一个map

Key：param1，param2或者参数的索引页可以（1,2），所以可以这样写



但是如果是这样写的话，如果十来个参数，你瞅着写，，确实挺累的，所以可以使用命名参数的方式

命名参数：明确指定封装参数时map的key(“多个参数的底层实现是放在一个map里面”)，@param{“id”}，这样key就是@param指定的值，value是赋的值



所以多个参数推荐使用命名参数的形式

POJO

如果多个参数正好是我们业务逻辑的数据类型，我们就可以直接传入POJO

#{属性值}取出传入的pojo的属性值（这个以前的helloworld就是这么搞的）

Map

如果多个参数不是业务逻辑的数据类型，不经常使用的话，我们还可以采用传入一个map的形式

#{key}：map中对应的value,

注意：这里需要指定resultType



如果多个参数不是业务模型中的数据，但是经常使用，推荐编写一个TO（Transfer Object）数据传输对象

3.特殊，Public Employee getEmpById(List<Integer> id)

如果是Collection（list，set）类型或者是数组，也会特殊处理，也是把传入的list或者数据封装在map中，key为collection，如果是list还可以使用key为list，是数组还可以使用key为array

例子练习

Public Employee getEmp（@Param(“id”) Integer id,String lastName）

取值：#{id/param1} ,#{param2}

Public Employee getEmp(Integer id,Employee emp)

取值：id🡪#{param1} lastName🡪 #{param2.lastName}

特殊：

总结：参数多时会封装map，为了不混轮，我们可以使用@Param来指定封装时使用的key，#{key}就可以取出map中的值

4重要

.#{}：可以获取map中的值，或者pojo对象属性的值

${}:可以获取map中的值或者pojo对象属性的值

区别：

#{}：是以预编译的形式，将参数设置到sql语句中，PreparedStatement，防止sql注入

${}：取出的值直接拼接在sql语句中，会有安全问题

大多情况下，我们取参数的值都应该使用#{}

但特殊情况，比如分表：按照年份分表析分

原生jdbc不支持占位符的地方我们就可以使用${}进行取值

Select \* from ${year}\_salary where xxx;

Select \* from employee order by ${f\_name} ${order}

标签

1.select标签

其中属性

Id：对应接口中的方法名

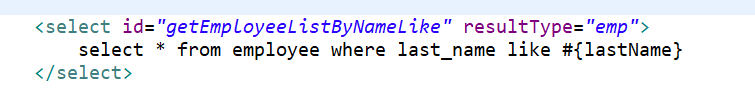
resultType：返回值类型

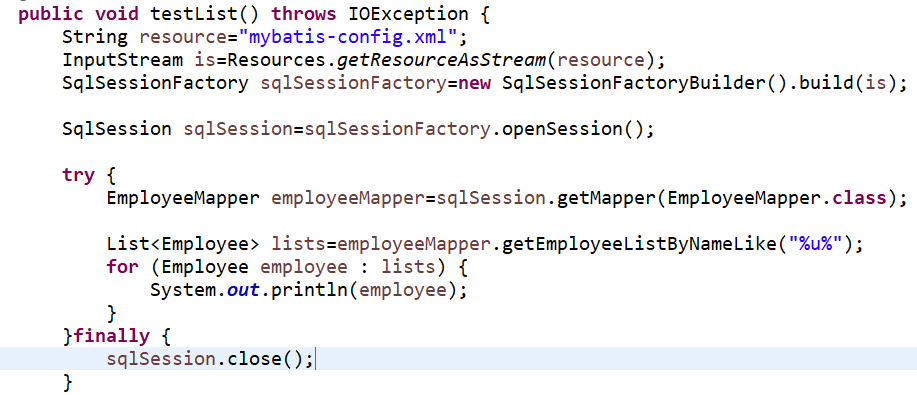
databaseId：数据库Id，就是这条sql语句具体支持哪个数据库，这个databaseId可以指定在全局配置文件databaseIdProvider里面指定的

parameterType：参数类型（一般不写，他会自动识别）



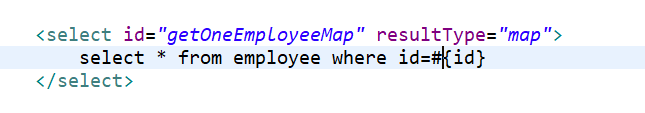
注意：1.如果返回值类型是List类型的，，resultType里指定的是List里面元素的类型





2.返回一条记录的map：key：列名，value是列名对应的值



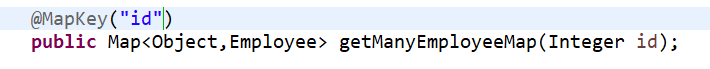


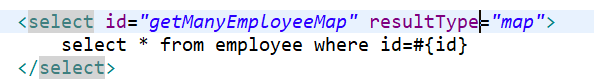
这里能直接写map，是因为mybatis里面齐了，别名，具体请参见以前的文件



3.返回多条记录的map，key：自己指定，value：Employee对象

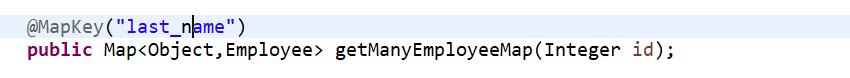
这里要使用@Mapkey注解，这个注解可以指定，你生成的那个map；里面谁来当key







下面是以last\_name来作为key



4.resultMap

resultType和resultMap只能选一个，resultType只要用来自动封装（自动封装靠的是全局属性里面的setting设置，autoMappingbehavior默认是PARTIAL，开启自动映射），resultMap可以自己定义封装的规则，所以更多更复杂的封装规则，纯靠resultMap，，所以这个很重要

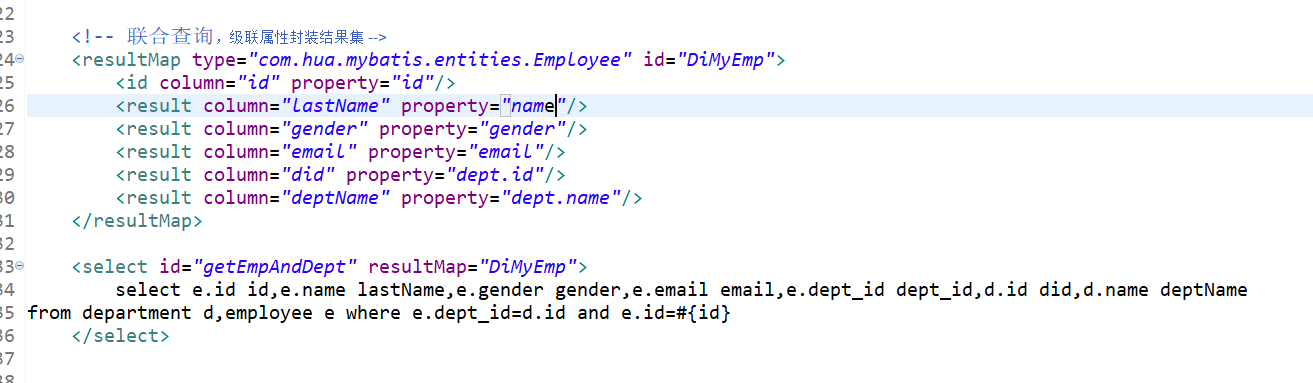
resultMap：自定义结果集的封装规则，比如以前我们last\_name转换为lastName是靠全局配置文件里设置的，而现在则使用resultMap来自定义Employee的结果集



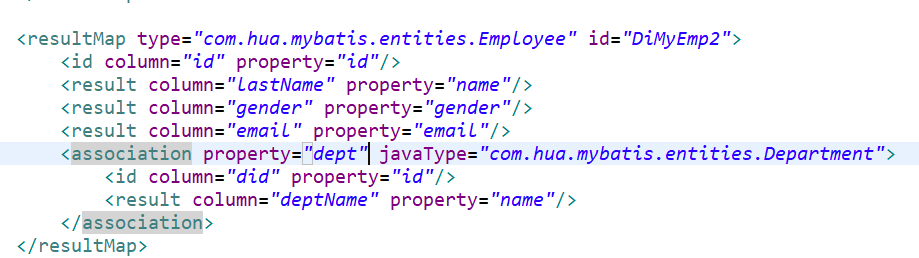
5.resultMap更强大的用法

要求，employee里面有个department对象，然后在查employee的时候，顺带把department里面的属性查询出来

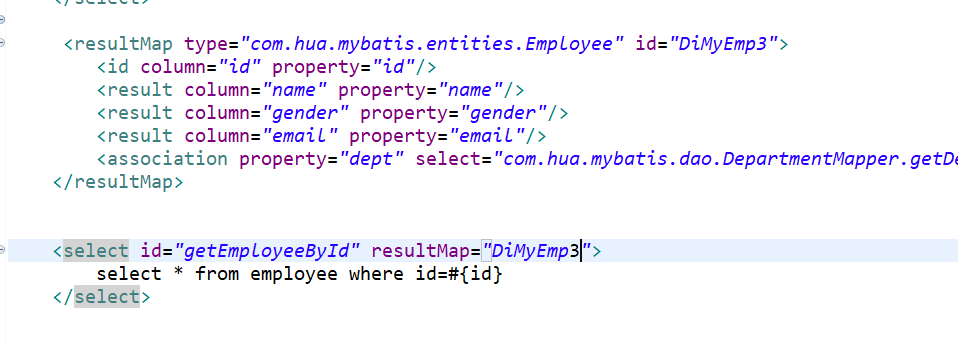
联合查询，可以使用 属性.属性，来引用别的对象里面的属性。



还可以使用association标签，association可以指定联合的javaBean对象，property=”dept”,指定哪个属性是联合的对象，javaType：指定这个属性的类型【不能省略】



还可以使用分布查询的方式来实现这一功能



其中property指定关联的属性，select：表明当前属性时调用select指定方法查出的结果

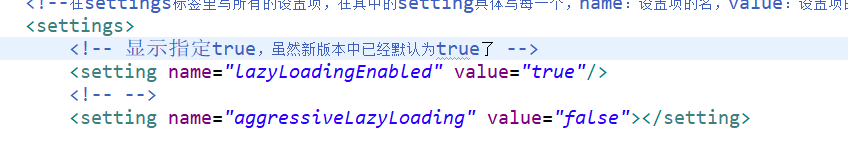
Column：指定将哪一列的值传给这个方法。

总流程的话：使用select指定的方法，传入column指定的这列参数的值，查出对象，并封装给property

分布查询的优点：可以组合已有的方法来完成查询

有延迟加载的功能（每次查询Employee对象的时候不先查出Department，使用Department的时候再去查Department）

要实现延迟加载的话需要在全局配置文件里面的settings里面设置两个setting

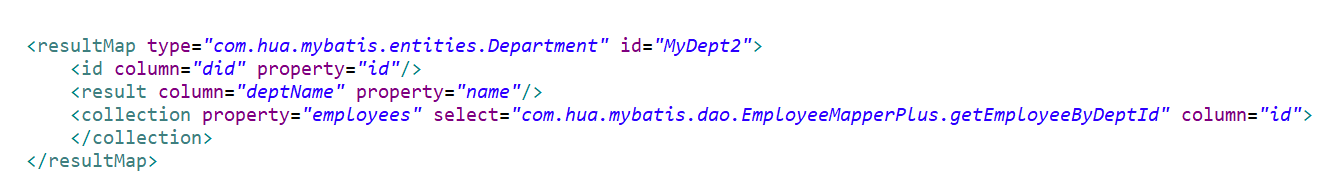


6.在部门类里加上所有员工的一个List，要求查询部分时将部门对应的所有的员工的信息查询出来（训练Collection的使用）

①．联合查询

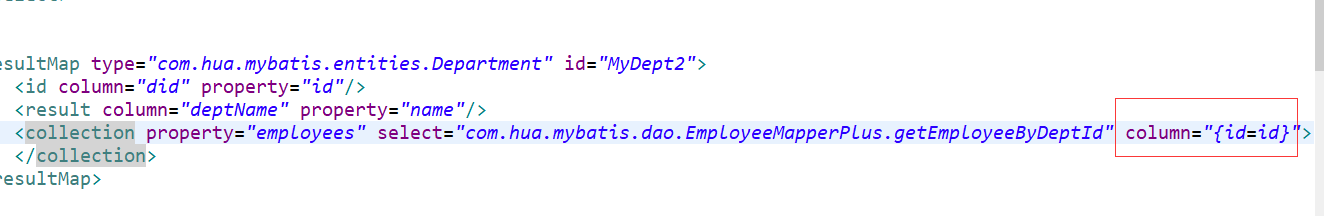


②分布查询



7.扩展，如果在分布查询，想将多列的值都传过去

解决方法：将column里面的封装成一个map，形式是column=”{key1=column1，key2=column2}



”

如果想局部开启或者关闭延迟加载还可以使用collection里面的fetchType属性，可以设置为lazy和eager

8.resultMap里面的鉴别器discriminator，了解一下即可

