**SqlMapConfig.xml(全局配置文件)**

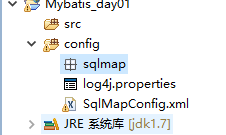
—SqlSessionFactory

--SqlSession

--exceutor

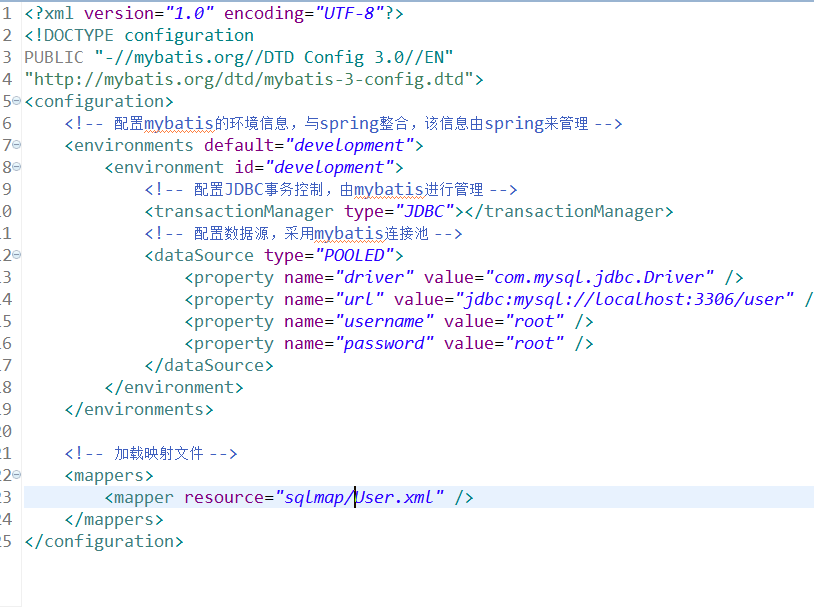
--statement

--数据库



映射文件存放sqlmap中 并在全局配置文件中加载

全局配置文件SqlMapConfig.xml：



映射文件配置sql语句（xxx.xml）



#{参数} 简单类型 参数任意名

${参数} 参数只能叫value

mapper代理开发规则：mapper.java mapper.xml

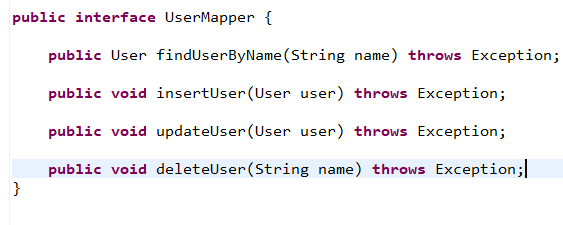
namespace=mapper接口地址

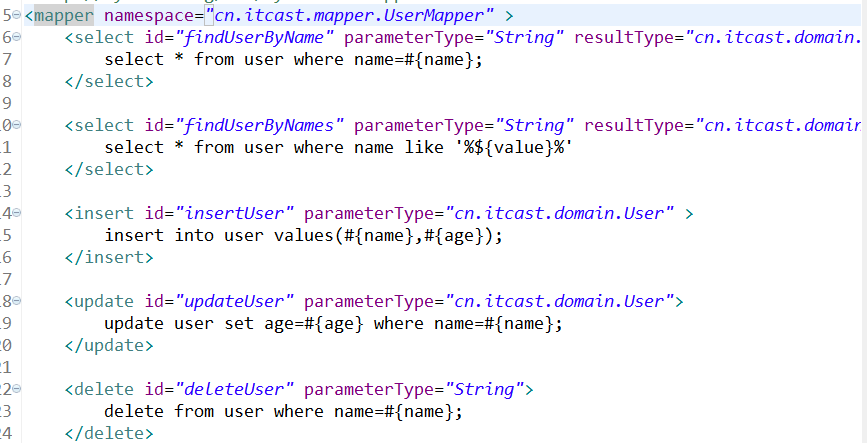


接口中的方法遵循规则

接口中方法名和xml中的statement的id相同

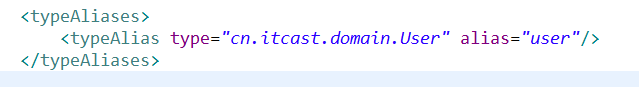
接口中方法参数类型 返回值类型遵循规则

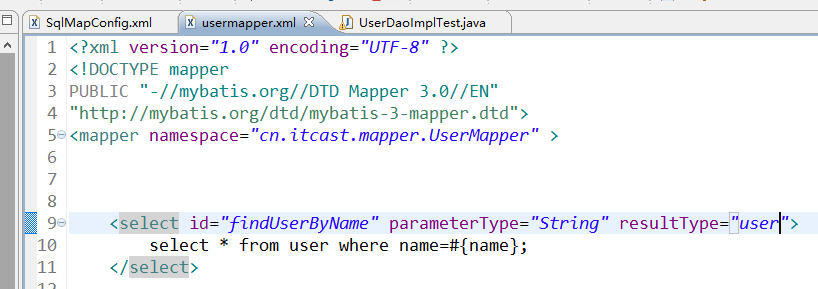




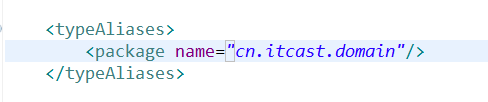
别名的定义

SqlMapConfig.xml

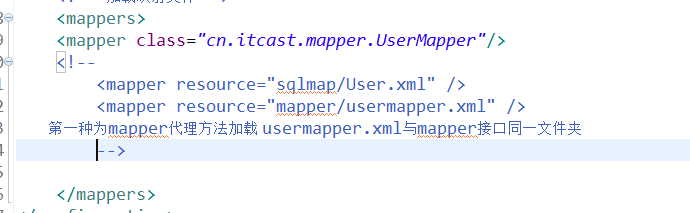




多个别名可以扫描配置 默认类名小写



映射文件的加载





批量加载参考别名的规则

包装类型pojo 为了应对复杂查询而存在

UserQueryVo包含UserCustom字段

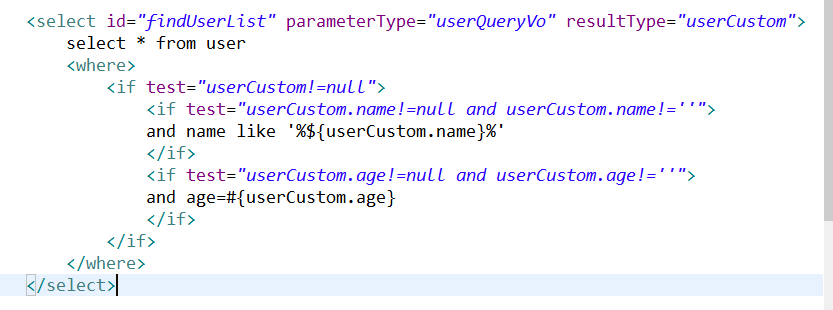
UserCustom继承User

Pojo做高级查询

首先查出记录数

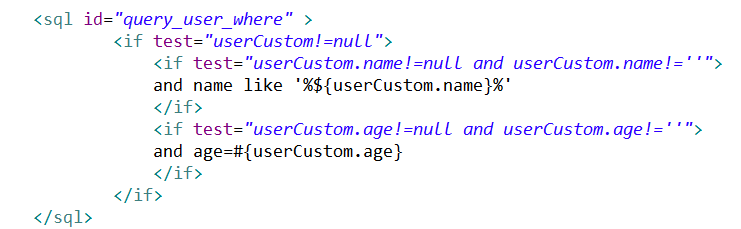
动态SQL

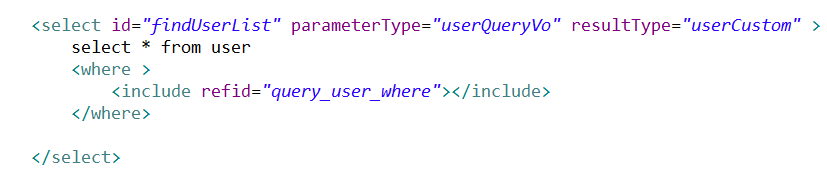
Sql拼接示例代码



Sql片段

定义sql片段在statement中引用他





## 涉及多表查询的N对N的映射查询

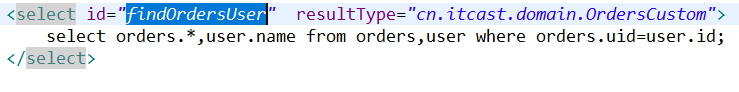
例:根据订单查询用户 一个订单对应多个用户

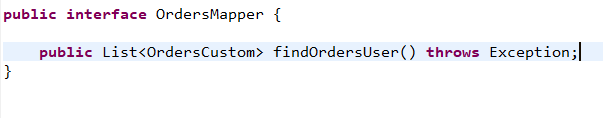
ResultType实现：

设计OrderCustom实体类 封装查询信息(实际操作为了方便 继承查询字段较多的类)



设计ordersmapper的xml和接口 并在主配置文件配置该xml

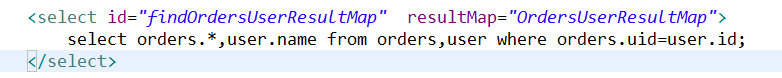


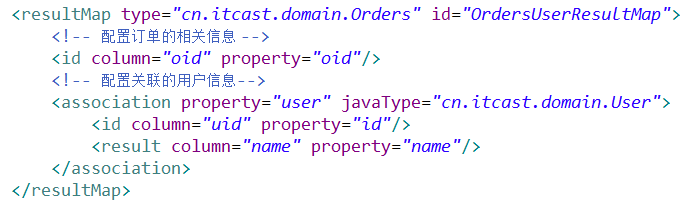


ResultMap实现：

查询出的订单信息保存到Orders对象中，Orders对象包含User对象

同理配置xml和mapper接口





配置信息中的id唯一标示信息的列

column为查询出的列名

pro为实体类中的属性名

association为关联查询的类 其id如上 其他信息用result封装

定义map的id在statement引用

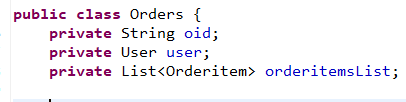
接口



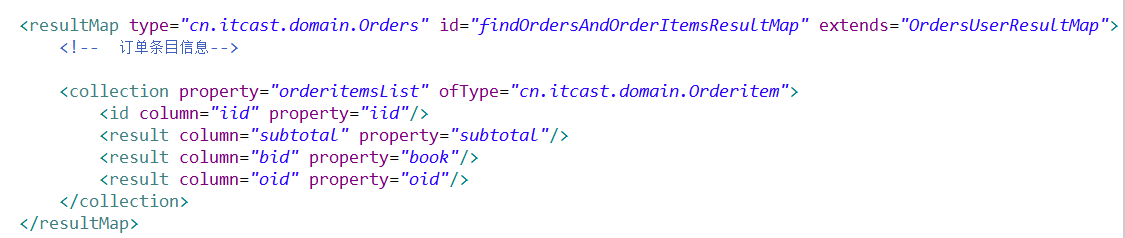
## 一对多查询

一个订单有多个订单条目信息

实体类设计

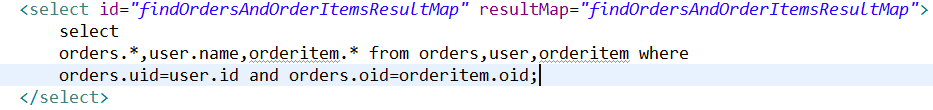


xml的配置 resultMap可以继承 省去重复代码



集合属性利用<collection>标签封装

Sql语句



多对多查询

例如查询用户拥有的订单的所有条目信息

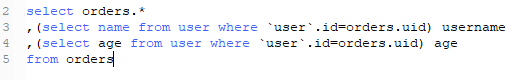
一个用户-多个订单-

一个订单-多个条目

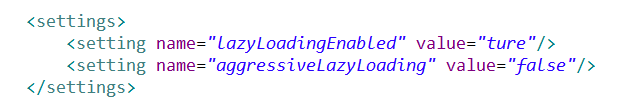
属性集合依旧<collection>封装



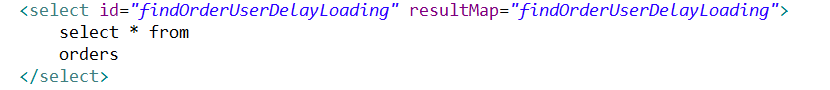
延迟加载:sql语句

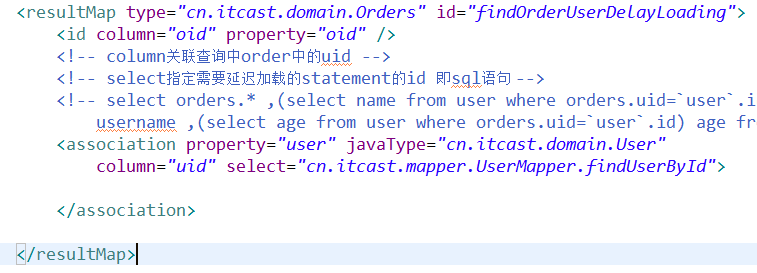


在主配置文件中开启延迟加载的选项



ResultMap中的select collection标签支持延迟加载





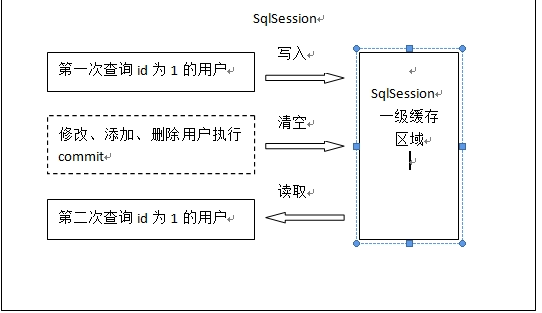
引用其他命名空间的statement注意namespace+id

一级缓存:sqlSession级别的缓存 SqlSession其中有HashMap数据结构用来存储缓存数据 多个sqlSession之间不影响

二级缓存:mapper级别的缓存 多个sqlSession执行同一个mapper的sql语句操作数据库得到的数据存储在二级缓存中

因此二级缓存是跨sqlSession的

一级缓存的工作原理:



一级缓存测试:

Mybatis默认支持一级缓存 无需配置

两个mapper如果namespace相同 则共用二级缓存区域

二级缓存按照namespace划分

清空缓存commit或者手动清空

二级缓存对应的pojo类应该实现序列化接口 因为缓存可能不在内存保留

sqlSession关闭缓存才会写入

mybatis和分布式缓存框架整合:my自身提供了cache接口

### 逆向工程

需要xml和java代码执行

**import** java.io.File;

**import** java.util.ArrayList;

**import** java.util.List;

**import** org.mybatis.generator.api.MyBatisGenerator;

**import** org.mybatis.generator.config.Configuration;

**import** org.mybatis.generator.config.xml.ConfigurationParser;

**import** org.mybatis.generator.internal.DefaultShellCallback;

**public** **class** Demo {

/\*\*

\* **@param** args

\*/

**public** **static** **void** main(String[] args) **throws** Exception {

List<String> warnings = **new** ArrayList<String>();

**boolean** overwrite = **true**;

//指向逆向工程配置文件

File configFile = **new** File("src/Demo.xml");

ConfigurationParser cp = **new** ConfigurationParser(warnings);

Configuration config = cp.parseConfiguration(configFile);

DefaultShellCallback callback = **new** DefaultShellCallback(overwrite);

MyBatisGenerator myBatisGenerator = **new** MyBatisGenerator(config,

callback, warnings);

myBatisGenerator.generate(**null**);

}

}

会自动生成mapper文件和pojo类

### Spring和mybatis整合

sqlSessionFactory由spring进行管理

**mapper代理开发方法整合**：

