Mybatis第二天

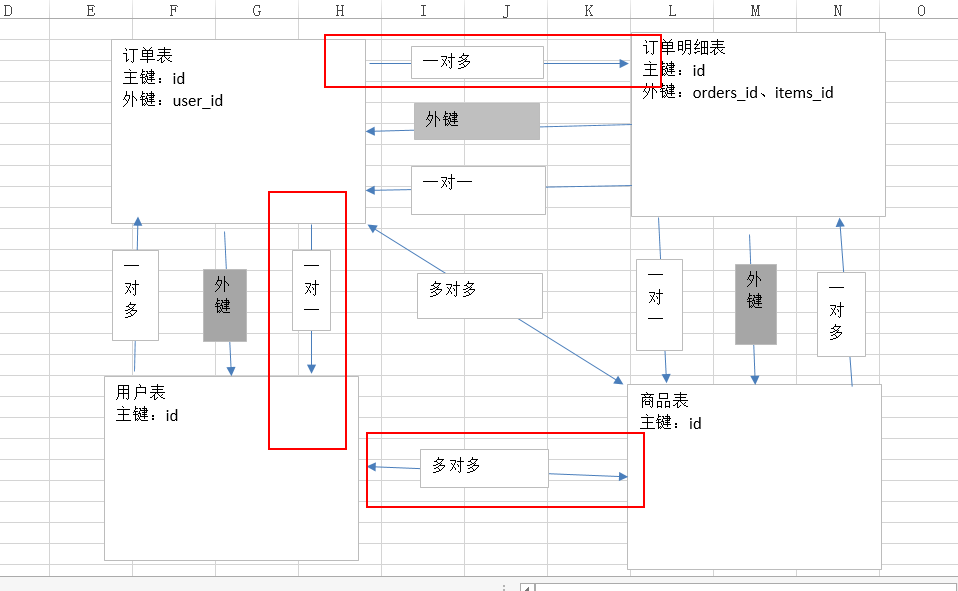
# 课程计划

1. 高级结果映射（一对一、一对多、多对多）（重点）
2. 延迟加载
3. 查询缓存
4. Spring和mybatis的整合（重点）
5. 逆向工程

# 高级结果映射

## 数据模型分析

1. 明确每张表存储的信息
2. 明确每张表中关键字段（主键、外键、非空）
3. 明确数据库中表与表之间的外键关系
4. 明确业务中表与表的关系（建立在具体的业务）



## 一对一映射

### 需求

查询订单信息，关联查询用户信息

### Sql

主信息：orders

从信息：user

SELECT

orders.`id`,

orders.`user\_id`,

orders.`number`,

user.`username`,

user.`sex`

FROM

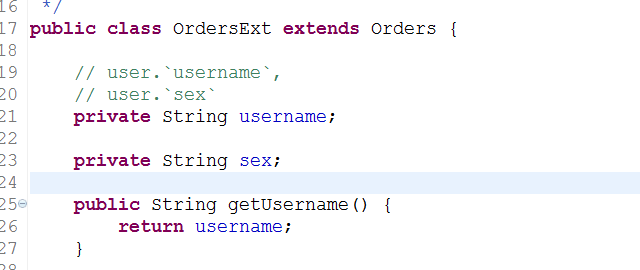
orders,

USER

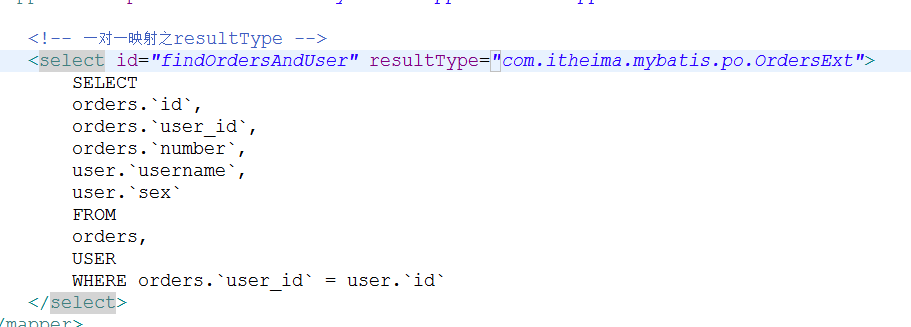
WHERE orders.`user\_id` = user.`id`

### resultType

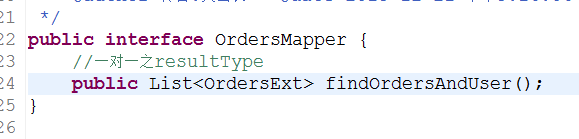
#### 创建扩展类



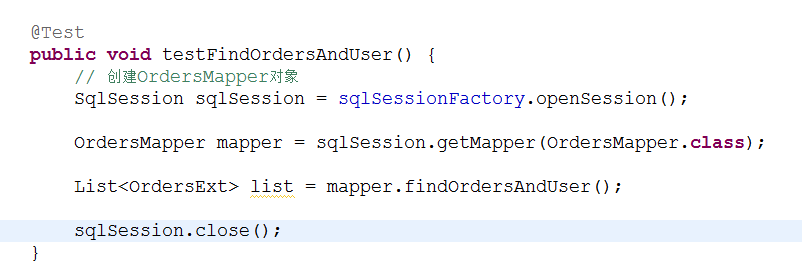
#### 映射文件



#### Mapper接口



#### 测试代码



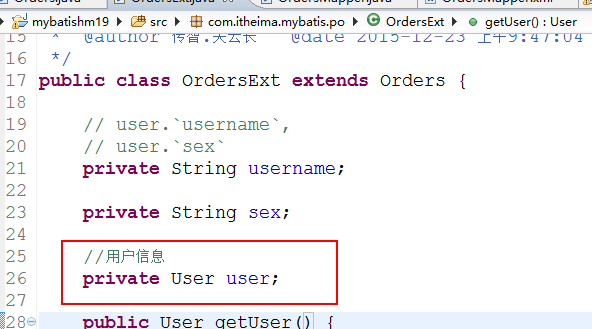
#### 小结

使用resultType来进行一对一结果映射，查询出的列的个数和映射的属性的个数要一致。而且映射的属性要存在与一个大的对象中，它是一种平铺式的映射，即数据库查询出多少条记录，则映射成多少个对象。

### resultMap

使用resultMap来进行一对一结果映射，它是将关联对象添加到主信息的对象中，具体说是对象嵌套对象的一种映射方式。

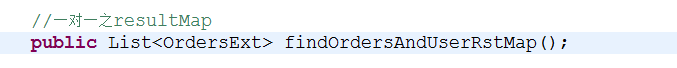
#### 修改扩展类



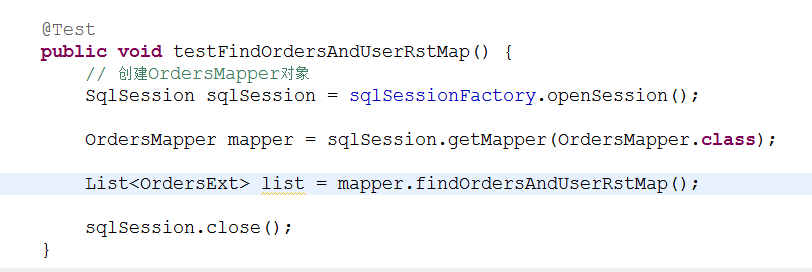
#### 映射文件



#### Mapper接口



#### 测试代码



### 小结

在一对一结果映射时，使用resultType更加简单方便，如果有特殊要求（对象嵌套对象）时，需要使用resultMap进行映射，比如：查询订单列表，然后在点击列表中的查看订单明细按钮，这个时候就需要使用resultMap进行结果映射。而resultType更适应于查询明细信息，比如，查询订单明细列表。

## 一对多映射

### 需求

查询订单信息，关联查询订单明细信息及用户信息

### Sql

主信息：orders

从信息：orderdetail、user

SELECT

orders.`id`,

orders.`user\_id`,

orders.`number`,

user.`username`,

user.`sex`,

orderdetail.`id` detailId,

orderdetail.`items\_id`,

orderdetail.`items\_num`

FROM

orders,

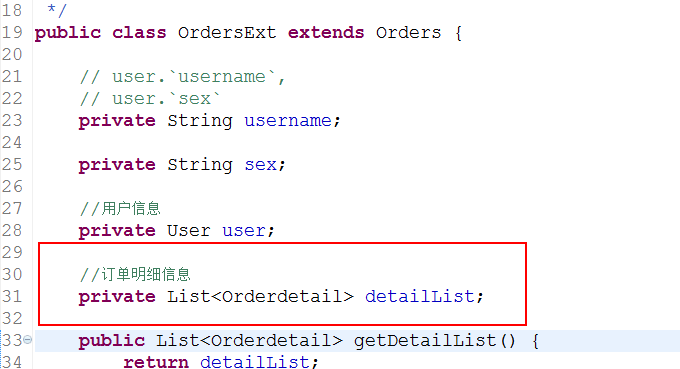
USER,

orderdetail

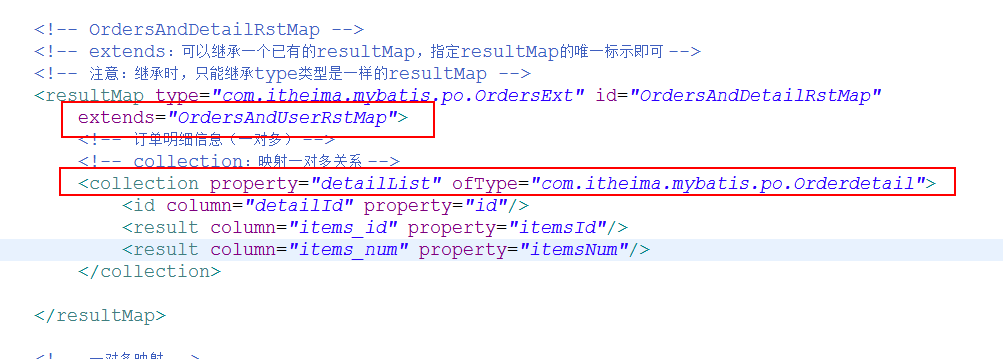
WHERE orders.`user\_id` = user.`id`

AND orders.`id` = orderdetail.`orders\_id`

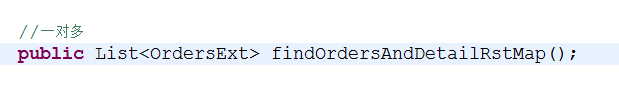
### 修改扩展类



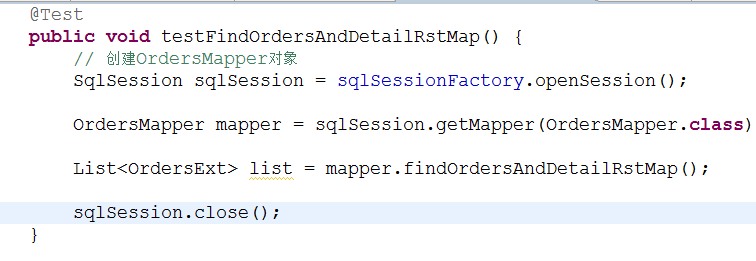
### 映射文件



### Mapper接口



### 测试代码



## 多对多映射

多对多映射是一对多映射的特例

### 需求

查询用户信息，关联查询该用户购买的商品信息

### Sql

主信息：user

从信息：items、orders、orderdetail

SELECT

orders.`id`,

orders.`user\_id`,

orders.`number`,

user.`username`,

user.`sex`,

orderdetail.`id` detailId,

orderdetail.`items\_id`,

orderdetail.`items\_num`,

items.`name`,

items.`price`

FROM

orders,

USER,

orderdetail,

items

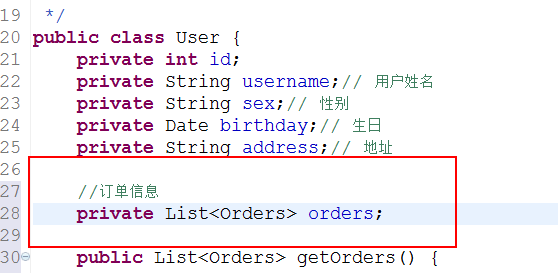
WHERE orders.`user\_id` = user.`id`

AND orders.`id` = orderdetail.`orders\_id`

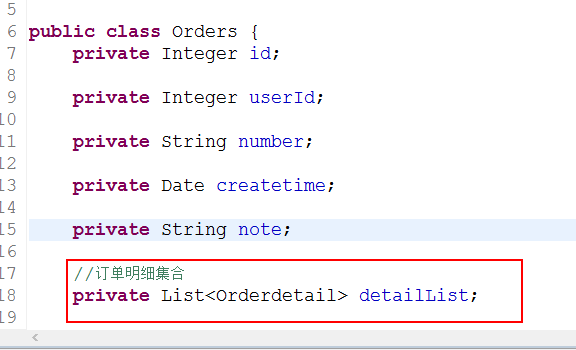
AND orderdetail.`items\_id` = items.`id`

### 修改po类

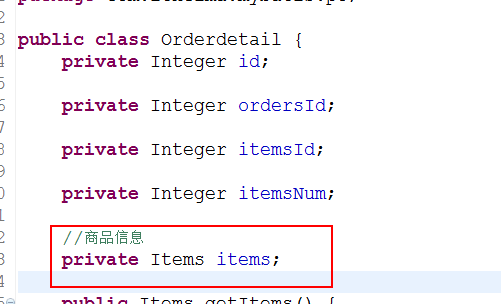
在User类中添加List<Orders> orders;



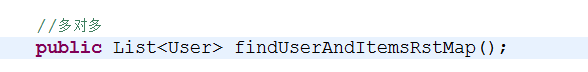
在Orders类中添加List<Orderdetail> detailList;



在Orderdetail中添加Items items；

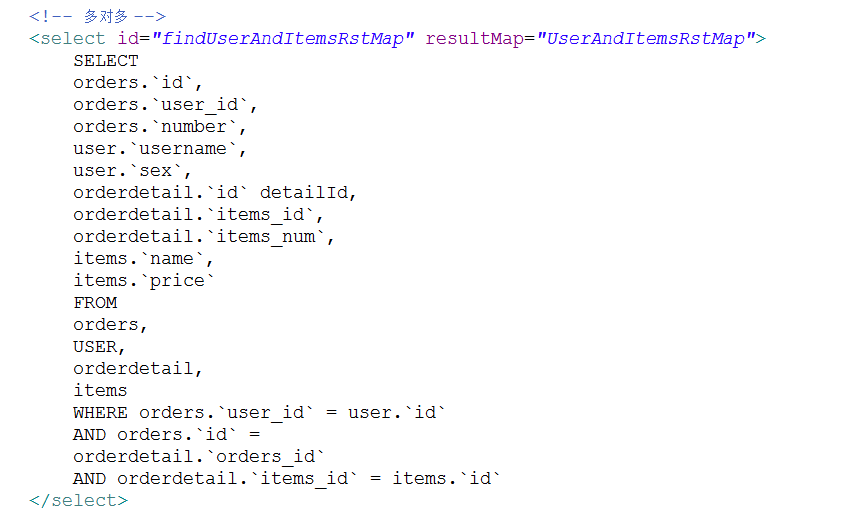


### Mapper接口

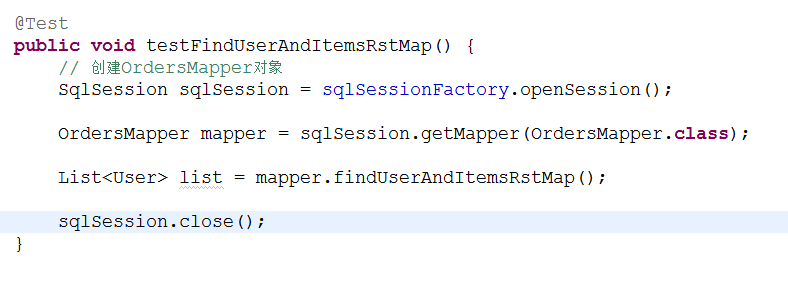


### 映射文件





### 测试代码



# 延迟加载

## 什么是延迟加载

延迟加载又叫懒加载，也叫按需加载。也就是说先加载主信息，在需要的时候，再去加载从信息。

在mybatis中，resultMap标签 的association标签和collection标签具有延迟加载的功能。

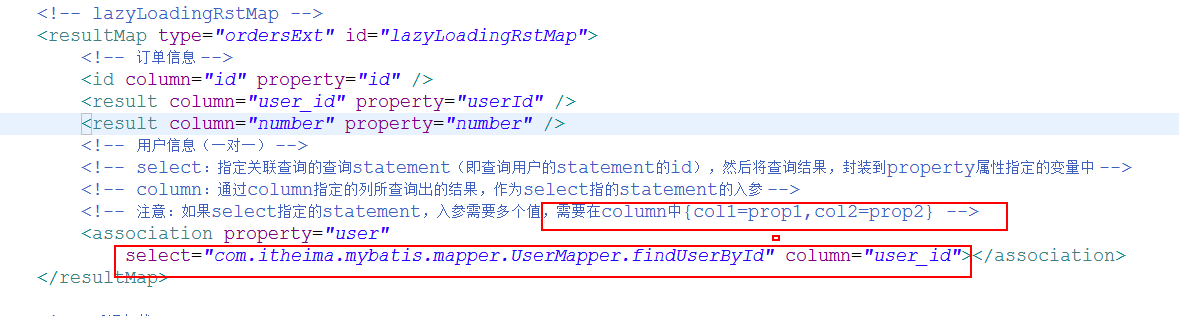
## 需求

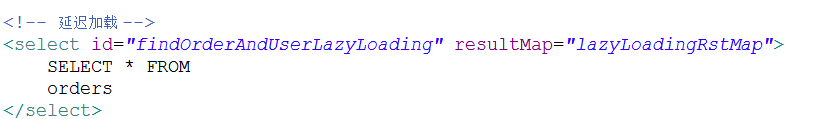
查询订单信息，关联查询用户信息

1. 创建一个statement来查询订单信息
2. 创建一个statement来查询用户信息

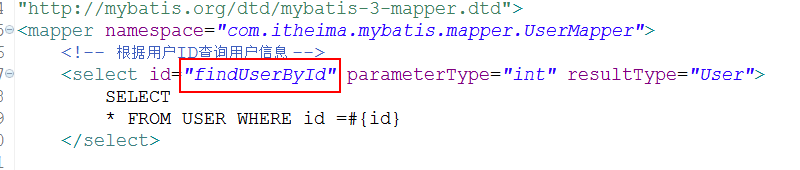
## 映射文件

1. 创建查询订单信息的映射文件

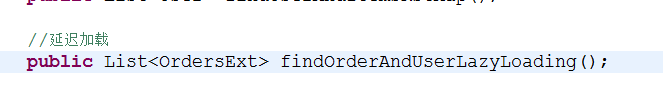




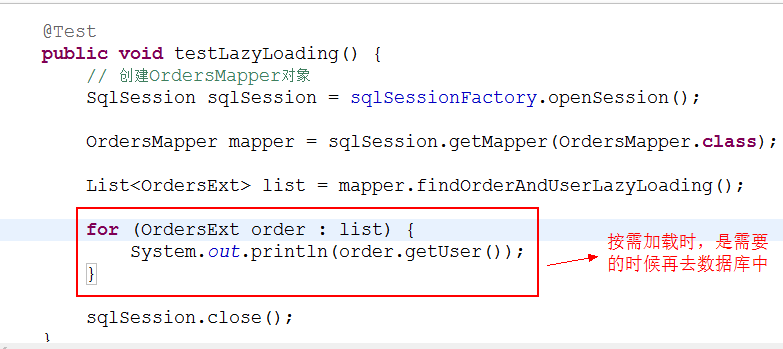
1. 创建查询用户信息的映射文件



## Mapper接口

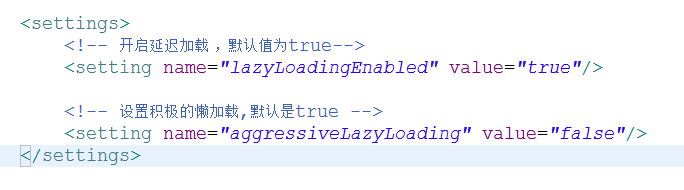


## 测试代码



## 设置延迟加载

在SqlMapConfig.xml中，配置settings标签



# 查询缓存

## Mybatis的缓存理解

sqlSession1

sqlSession 2

sqlSession3...

一级缓存

一级缓存

一级缓存

Mapper(namespace)二级缓存

Mybatis的缓存，包括一级缓存和二级缓存

一级缓存指的就是sqlsession，在sqlsession中有一个数据区域，是map结构，这个区域就是一级缓存区域。一级缓存中的key是由sql语句、条件、statement等信息组成一个唯一值。一级缓存中的value，就是查询出的结果对象。

二级缓存指的就是同一个namespace下的mapper，二级缓存中，也有一个map结构，这个区域就是一级缓存区域。一级缓存中的key是由sql语句、条件、statement等信息组成一个唯一值。一级缓存中的value，就是查询出的结果对象。

一级缓存是默认使用的。

二级缓存需要手动开启。

## 一级缓存

### 原理

第一次查询id为1的用户

SqlSession

一级缓存区域

Map

SqlSession

写入

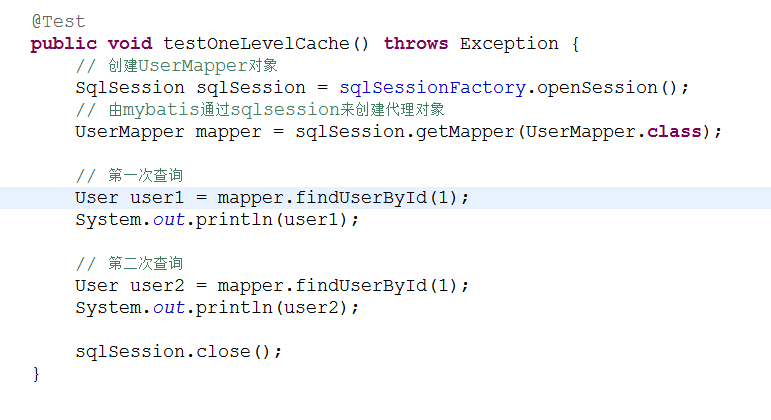
修改、添加、删除用户执行commit

清空

第二次查询id为1的用户

读取

### 测试1



### 测试2





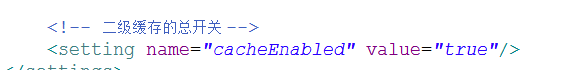
## 二级缓存

### 原理

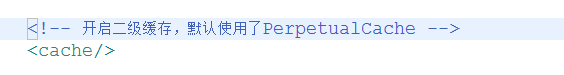


### 开启二级缓存

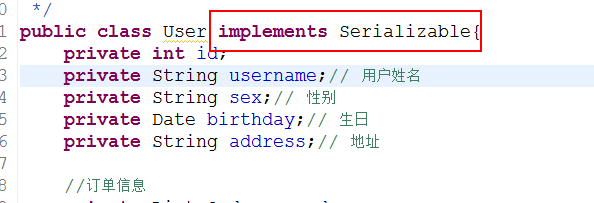
1. 开启二级缓存的总开关



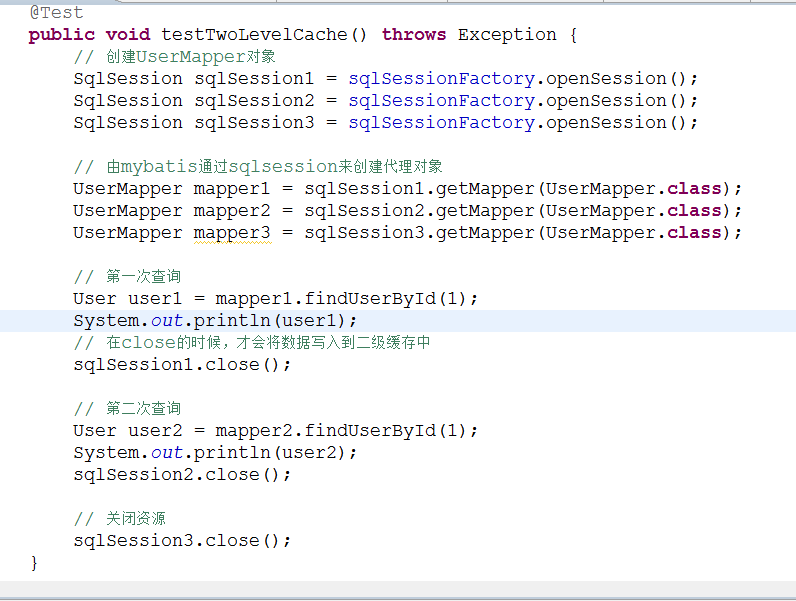
1. 在mapper映射文件中开启二级缓存

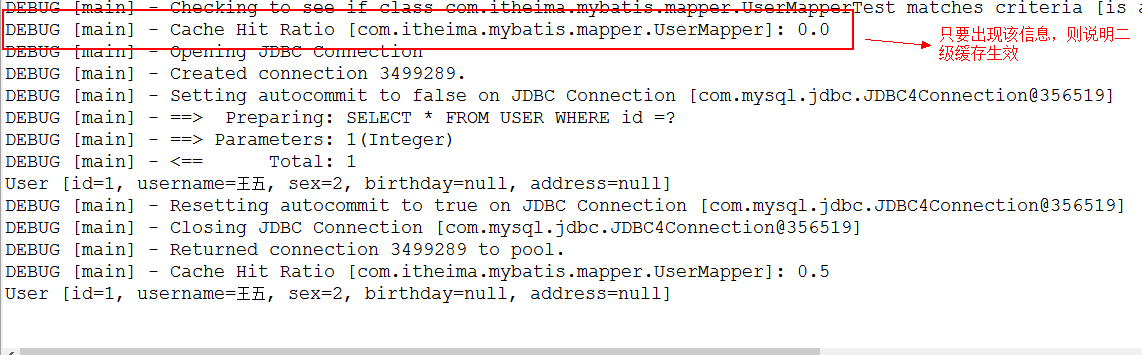


### 序列化



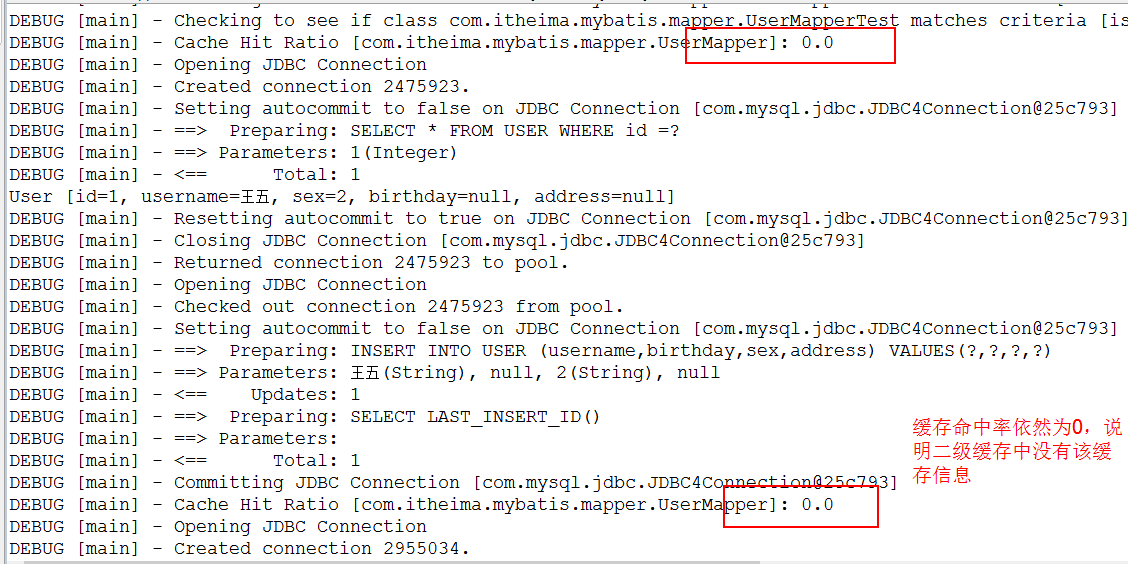
### 测试1





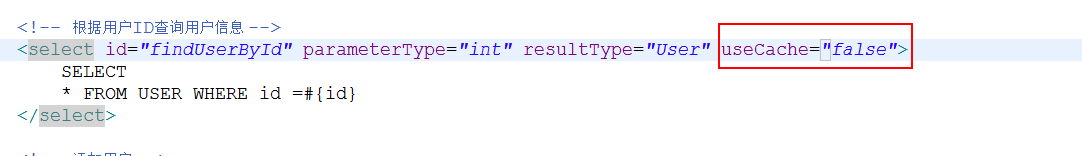
### 测试2



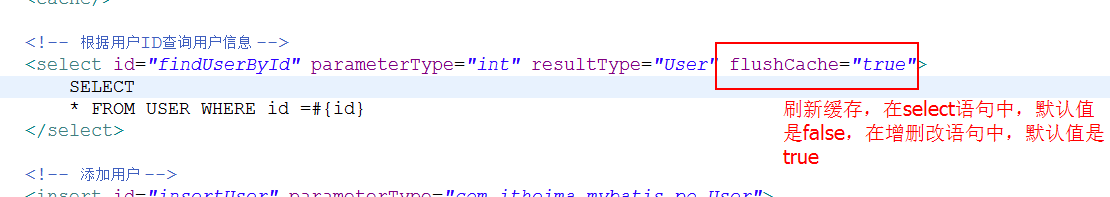


### 禁用缓存

默认值是true



### 刷新缓存



### 整合ehcache

Mybatis本身是一个持久层框架，它不是专门的缓存框架，所以它对缓存的实现不够好，不能支持分布式。

Ehcache是一个分布式的缓存框架。

#### 什么是分布式

系统为了提高性能，通常会对系统采用分布式部署（集群部署方式）

系统工程

springmvc

mybatis

服务器1

系统工程

springmvc

mybatis

服务器2

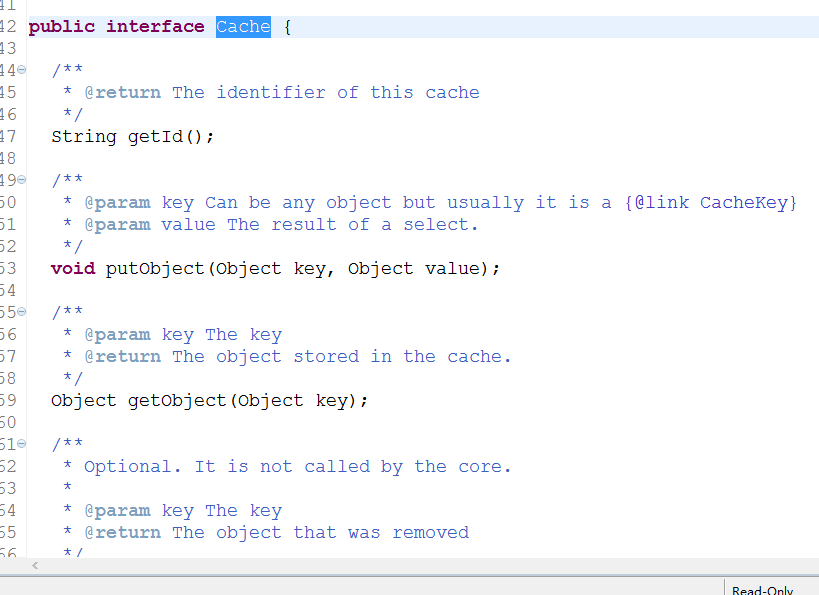
对缓存数据进行集中管理（redis集群）

使用分布式缓存框架

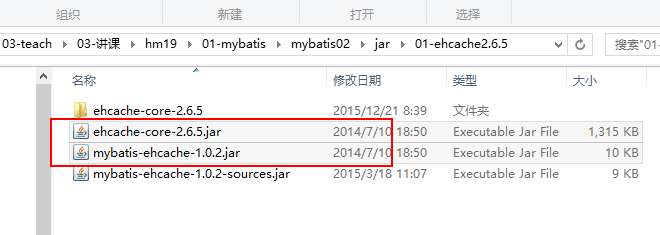
redis、memcached、ehcache。

#### 整合思路

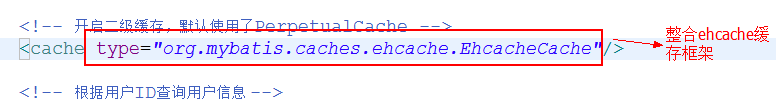
Cache是一个接口，它的默认实现是mybatis的PerpetualCache。如果想整合mybatis的二级缓存，那么实现Cache接口即可。



#### 添加jar包



#### 设置映射文件中cache标签的type值为ehcache的实现类

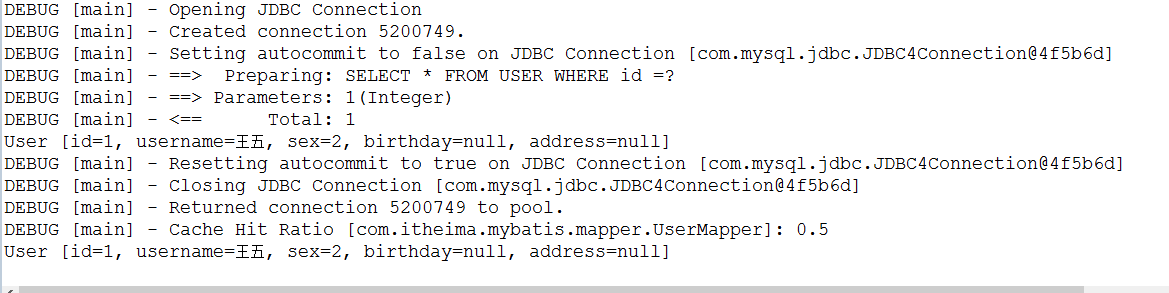


#### 添加ehcache的配置文件

在config下，创建ehcache.xml



#### 测试ehcache的二级缓存



### 应用场景

使用场景：**对于访问响应速度要求高，但是实时性不高的查询，可以采用二级缓存技术**。

注意：在使用二级缓存的时候，要设置一下刷新间隔（cache标签中有一个flashInterval属性）来定时刷新二级缓存，这个刷新间隔根据具体需求来设置，比如设置30分钟、60分钟等，单位为毫秒。

### 局限性

**Mybatis二级缓存对细粒度的数据，缓存实现不好。**

场景：对商品信息进行缓存，由于商品信息查询访问量大，但是要求用户每次查询都是最新的商品信息，此时如果使用二级缓存，就无法实现当一个商品发生变化只刷新该商品的缓存信息而不刷新其他商品缓存信息，因为二级缓存是mapper级别的，当一个商品的信息发送更新，所有的商品信息缓存数据都会清空。

解决此类问题，需要在业务层根据需要对数据有针对性的缓存。

比如可以对经常变化的 数据操作单独放到另一个namespace的mapper中。

# Mybatis整合spring

## 整合思路

1. 数据源信息交给spring管理
2. SqlSessionFactory交给spring进行单例管理
3. 由spring来管理原始dao的实现类或者mapper代理的代理类。

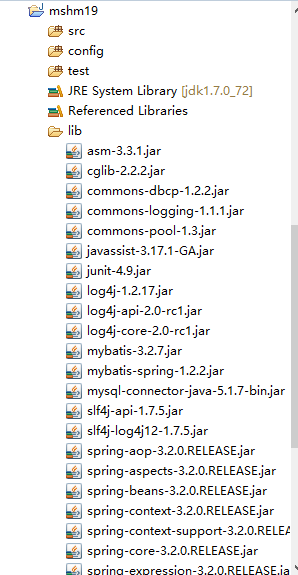
## 需求

使用原始dao方式和mapper代理方式实现以下功能：

根据用户ID查询商品信息

## 工程搭建

* Mysql的驱动包
* Mybatis的核心包和依赖包
* Mybatis和spring的整合包
* Spring的包
* dbcp数据库连接池包



## 具体整合

### 整合配置文件

#### Mybatis

在config下，创建mybatis目录，然后创建SqlMapConfig.xml



将db.properties和log4j.properties拷贝到config目录下。

#### Spring

在config下，创建spring目录，然后创建applicationContext.xml

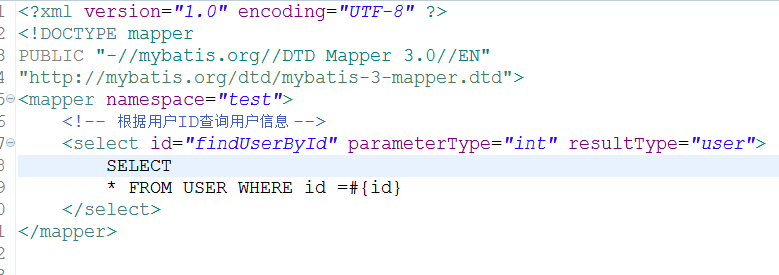


### 整合代码

#### 原始dao开发方式

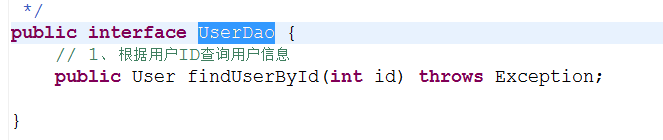
##### 映射文件

在config/mybatis下创建sqlmap，然后创建User.xml

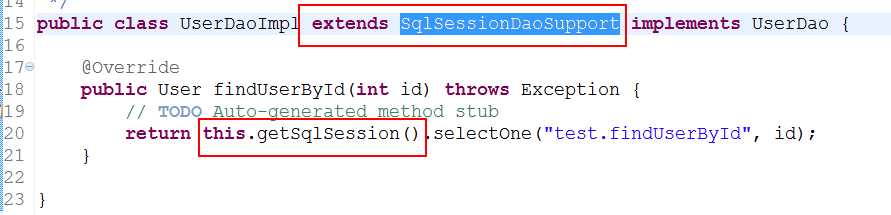


##### Dao代码

Dao接口

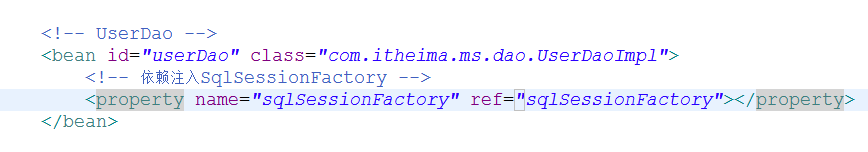


Dao实现类

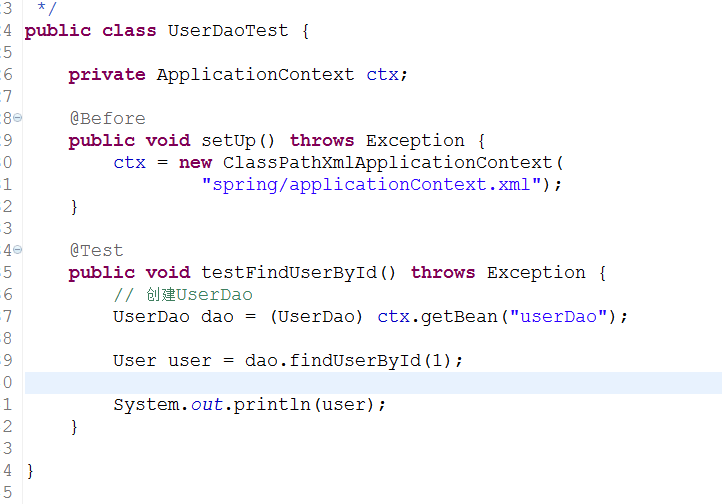


##### 配置UserDao实现类

在applicationContext.xml中配置UserDao实现类



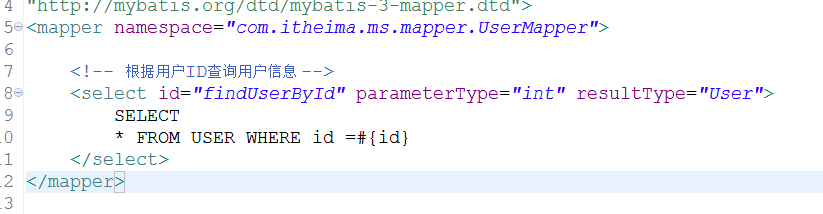
##### 测试代码



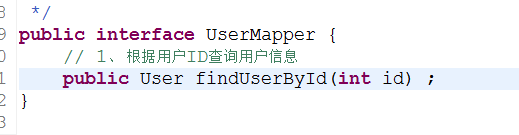
#### Mapper代理

##### 映射文件

将映射文件放到UserMapper接口的同包下

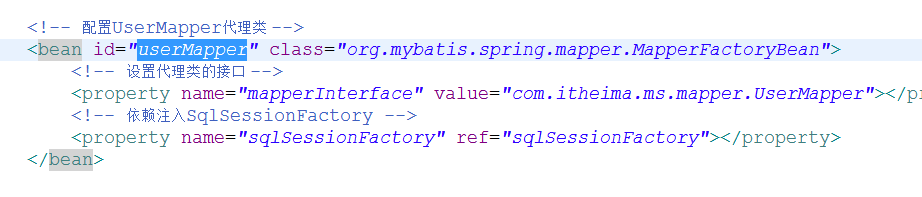


##### Mapper接口



##### 配置mapper代理类

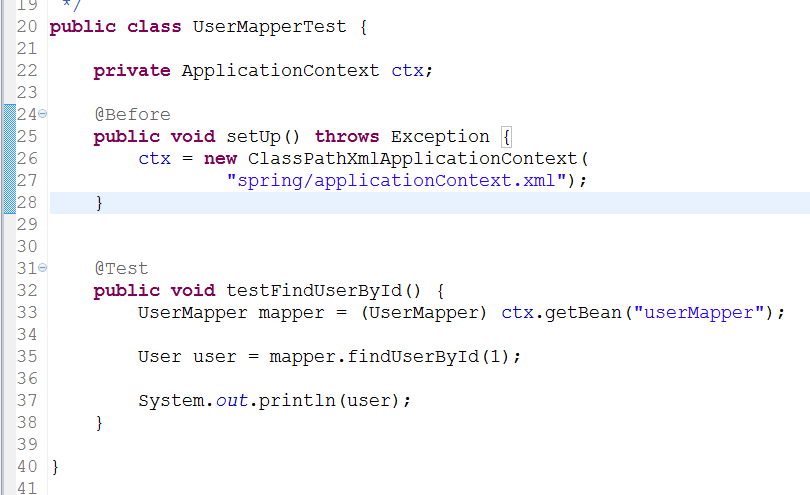
* 单个mapper代理类配置



* 批量设置mapper代理类



##### 测试代码



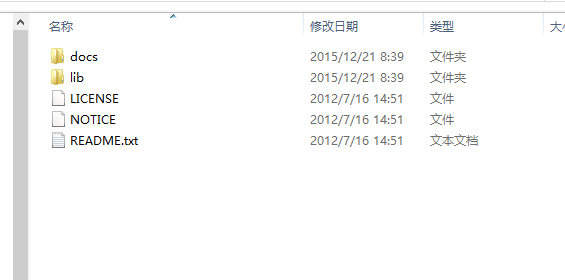
# 逆向工程（会用）

## 什么是逆向工程

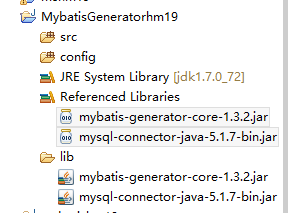
Mybatis提供来一个逆向工程工具，通过逆向工程，可以帮助程序员根据单表来生成po类、mapper映射文件、mapper接口。

## 下载逆向工程

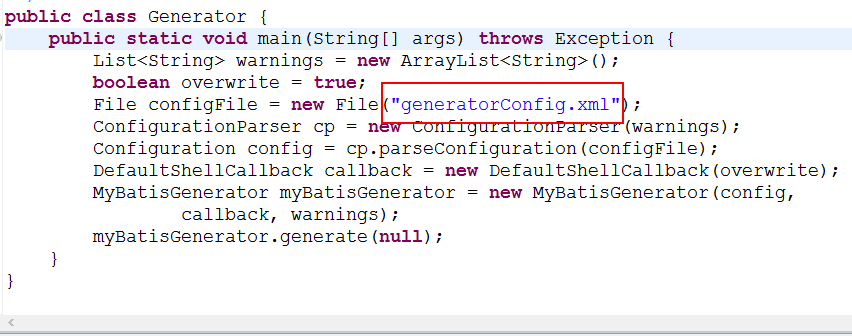
<https://github.com/mybatis/generator/releases/tag/mybatis-generator-1.3.2>



## 创建逆向工程



## 创建Generator.java

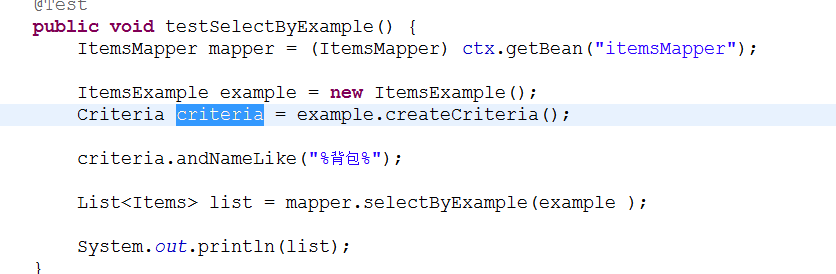


## 添加generatorConfig.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <!DOCTYPE generatorConfiguration  PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"  "http://mybatis.org/dtd/mybatis-generator-config\_1\_0.dtd">  <generatorConfiguration>  <context id=*"testTables"* targetRuntime=*"MyBatis3"*>  <commentGenerator>  <!-- 是否去除自动生成的注释 true：是 ： false:否 -->  <property name=*"suppressAllComments"* value=*"true"* />  </commentGenerator>  <!--数据库连接的信息：驱动类、连接地址、用户名、密码 -->  <jdbcConnection driverClass=*"com.mysql.jdbc.Driver"*  connectionURL=*"jdbc:mysql://localhost:3306/mybatis"* userId=*"root"*  password=*"root"*>  </jdbcConnection>  <!-- <jdbcConnection driverClass="oracle.jdbc.OracleDriver" connectionURL="jdbc:oracle:thin:@127.0.0.1:1521:yycg"  userId="yycg" password="yycg"> </jdbcConnection> -->  <!-- 默认false，把JDBC DECIMAL 和 NUMERIC 类型解析为 Integer，为 true时把JDBC DECIMAL  和 NUMERIC 类型解析为java.math.BigDecimal -->  <javaTypeResolver>  <property name=*"forceBigDecimals"* value=*"false"* />  </javaTypeResolver>  <!-- targetProject:生成PO类的位置 -->  <javaModelGenerator targetPackage=*"com.itheima.ms.po"*  targetProject=*".\src"*>  <!-- enableSubPackages:是否让schema作为包的后缀 -->  <property name=*"enableSubPackages"* value=*"false"* />  <!-- 从数据库返回的值被清理前后的空格 -->  <property name=*"trimStrings"* value=*"true"* />  </javaModelGenerator>  <!-- targetProject:mapper映射文件生成的位置 -->  <sqlMapGenerator targetPackage=*"com.itheima.ms.mapper"*  targetProject=*".\src"*>  <!-- enableSubPackages:是否让schema作为包的后缀 -->  <property name=*"enableSubPackages"* value=*"false"* />  </sqlMapGenerator>  <!-- targetPackage：mapper接口生成的位置 -->  <javaClientGenerator type=*"XMLMAPPER"*  targetPackage=*"com.itheima.ms.mapper"* targetProject=*".\src"*>  <!-- enableSubPackages:是否让schema作为包的后缀 -->  <property name=*"enableSubPackages"* value=*"false"* />  </javaClientGenerator>  <!-- 指定数据库表 -->  <table tableName=*"items"*></table>  <table tableName=*"orders"*></table>  <table tableName=*"orderdetail"*></table>  <table tableName=*"user"*></table>  </context>  </generatorConfiguration> |

## 将逆向工程生成的代码拷贝到指定项目中

## 使用逆向工程生成的代码



## 注意事项

Mapper.xml文件已经存在时，如果进行重新生成则mapper.xml文件时，内容不被覆盖而是进行内容追加，结果导致mybatis解析失败。

解决方法：**删除原来已经生成的mapper xml文件再进行生成。**

Mybatis自动生成的po及mapper.java文件不是内容而是直接覆盖没有此问题。