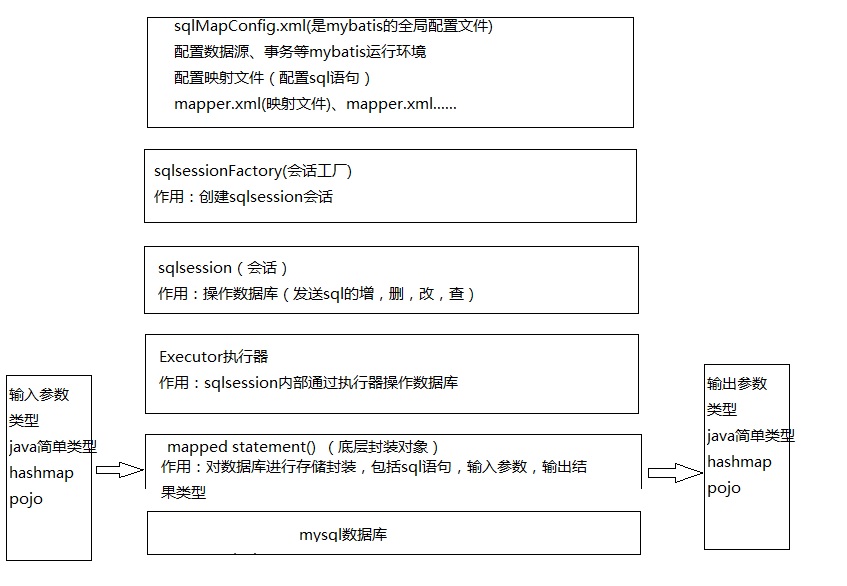
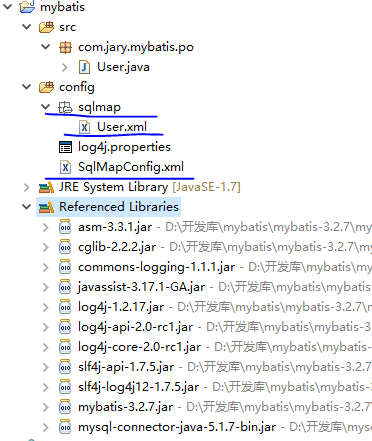
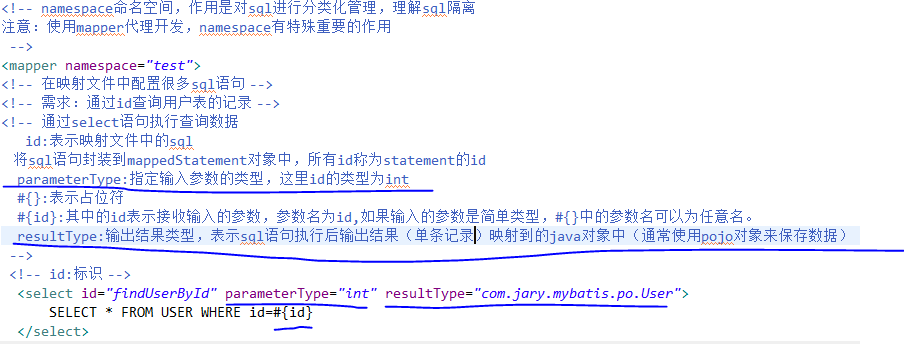
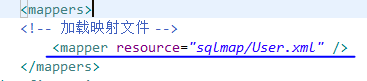
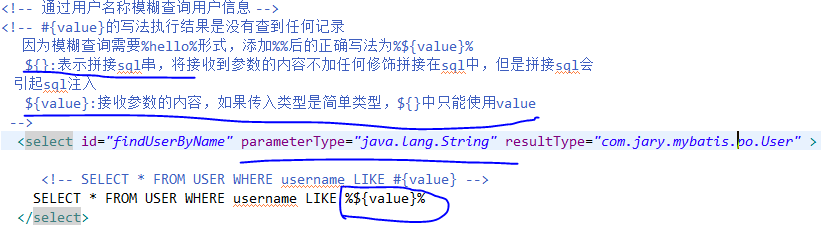
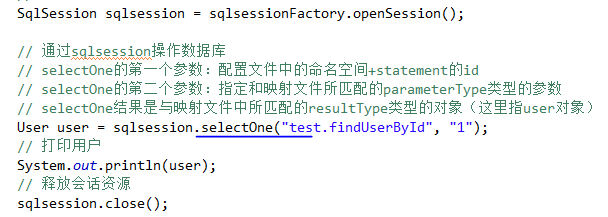
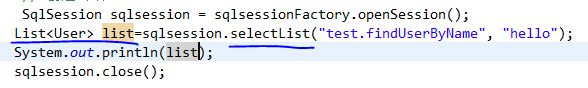
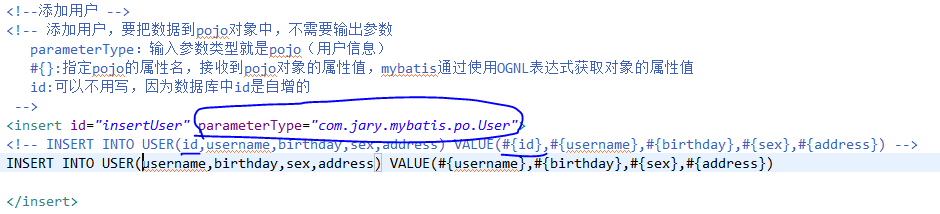
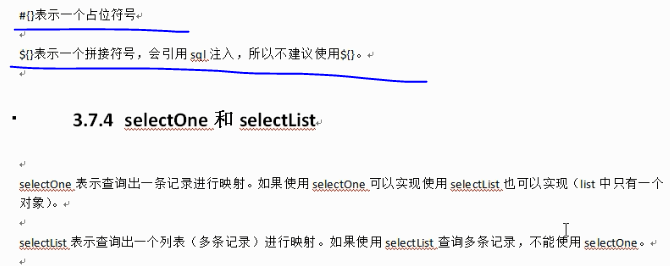
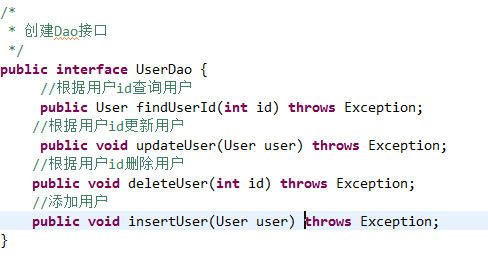
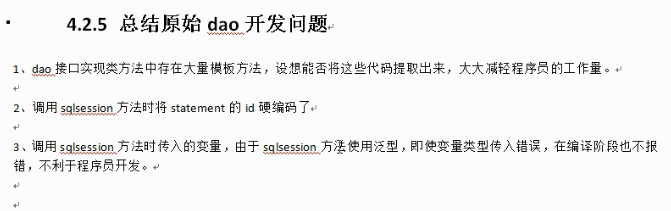
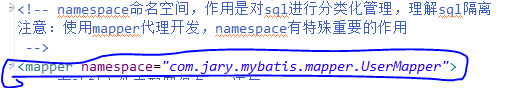
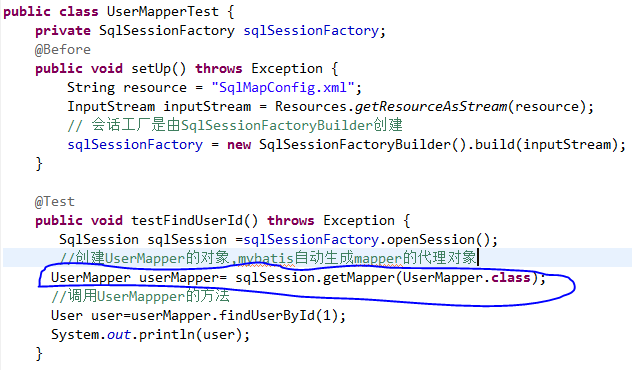
**Mybatis总结**

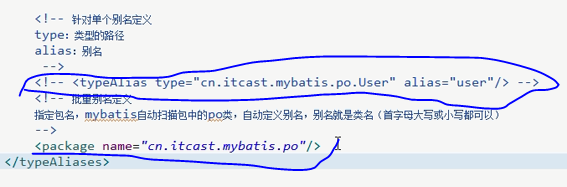
**使用jdbc开发时，和mybatis相比的不足**   
1，**[数据库](http://lib.csdn.net/base/mysql" \o "MySQL知识库" \t "_blank)**连接，使用时就创建，不使用就释放，对数据库进行频繁连接开关和关闭，造成数据库资源浪费，影响数据库的性能   
解决：使用数据库连接池管理数据库的连接   
2，sql语句使用硬编码在**[Java](http://lib.csdn.net/base/javase" \o "Java SE知识库" \t "_blank)**程序中，修改sql语句，就需要重新编译java代码，不利于系统维护   
解决：把sql语句放在xml配置文件中，修改sql语句也不需要重新编译java代码   
3，向预编译语句PreparedStatement中设置参数，对占位符位置和设置参数值，硬编码，修改sql语句也不需要重新编译java代码   
解决：把sql语句和占位符设置参数值放在xml配置文件中   
4，从result中遍历结果集数据时，存在硬编码，将获取表的字段进行硬编码   
解决：将查询的结果集，自动映射成 java对象

二 mybatis框架，是一个持久层框架，是apache下的顶级项目   
mybatis让程序员将主要精力放在sql上，通过mytabis提供的映射方式，自动生成满足需要的sql语句   
mybatis可以向PreparedStatement中输入参数自动进行**输入映射**，将查询结果集灵活的映射成Java对象（**输出映射**），输入映射和输出映射这是mybatis的核心   
mybatis框架执行流程图   
  
三 mybatis的工作环境搭建和**[架构](http://lib.csdn.net/base/architecture" \o "大型网站架构知识库" \t "_blank)**示意图   
   
四 mybatis的开发   
1.映射文件的开发如下图   
  
2,映射文件配置好了之后，还需要在全局配置文件sqlMapConfig.xml中添加映射文件   
   
3，sqlsession会话去执行操作查询数据库映射文件，下图中的错误纠正为’%${value}%’   
  
查询出的是单条记录使用selectOne，下图中的错误纠正为把“1”改为int类型的1   
sqlsession.selectOne(“test.findUserById”, 1);   
   
查询出的是多条记录使用selectList   
sqlsession.selectList(“test.findUserByName”, “hello”);   
   
4，添加用户映射文件配置如下：   
  
程序代码：   
  
5，总结：   
   
四 mybatis开发dao方法   
mybatis的配置文件不变   
1，先使用原型的开发dao方法   
开发接口   
   
2， 开发接口实现   
  
3， [**测试**](http://lib.csdn.net/base/softwaretest)代码   
  
4，总结   
   
五 mybatis利用mapper代理开发dao（**重点掌握**）   
mapper代理开发，就不需要接口的实现类，只需要接口UserMapper.java和映射文件UserMapper.xml就可以了，但是遵循一定的开发规范：   
**1，在UserMapper.xml文件中namespace等于UserMapper接口地址**   
   
**2，UserMapper.java接口中的方法名要和UserMapper.xml中的statement的id一致**   
**3，UserMapper.java接口中的方法输入参数要和UserMapper.xml中的statement的parameterType指定的类型一致**   
**4，UserMapper.java接口中的方法的返回值类型要和UserMapper.xml中的statement的resultType指定的类型一致**   
测试代码：   
   
**上图画线区域:这里没有实现接口的实现类，而是使用mybatis生成的代理对象来生成UserMappper接口的对象，从而能够调用其方法**

mapper代理开发dao出现的问题总结：   
1，代理对象内部调用selectOne或selectList   
如果mapper方法返回单个pojo对象（非集合对象），代理对象内部通过selectOne查询数据库，也可以使用selectList查询。   
如果mapper方法返回集合对象，代理对象内部通过selectList查询 数据库，不能使用selectOne查询，否则会出错。   
**问题是： 编译期间不会报错，二者容易写错使用。**   
2，mapper接口方法参数只有一个   
根据规范编写的代理对象的传入参数只能有一个（mapper.xml文件中的parameterType参数只有一个），不利于系统的扩展   
解决：即使mapper接口中只有一个参数，可以使用包装类型的pojo满足不同的业务方法需求

**mybatis的一些细节剖析：**

1，全局配置文件sqlMapConfig.xml中配置内容如下：

1. properties(属性)   
   注意：mybatis将按照下面的顺序来加载属性：   
   （1）在properties元素体内定义的属性首先被读取。（可以在此属性中加入jdbc的配置文件db.properties）,在sqlMapConfig.xml中就不需要对数据库连接参数进行硬编码了。
2. settings全局参数设置   
   mybatis框架运行时可以调整一些运行参数，会影响mybatis运行行为，所以建议不要随便修改   
   比如：二级缓存，开启延时加载。。。
3. **typeAliases(别名) 重点掌握**   
   

<typeAliases>

<!--针对单个别名定义

type：类型的路径

alias：别名 -->

<typeAlias type="com.jary.mybatis.po.User" alias="user" />

<!--还可以进行批量别名定义

指定包名，mybatis自动扫描包中的po类 -->

<package name="com.jary.mybatis.po" />

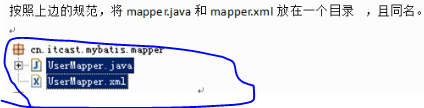
</typeAliases>

上面的别名定义后，在mapper.xml中就可以这样使用了   
  
**user代替了输出结果类型com.jary.mybatis.po.User。**

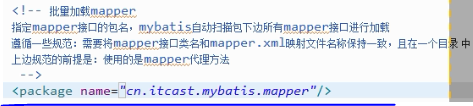
4.映射文件（mapper）

通过resource加载单个的映射文件

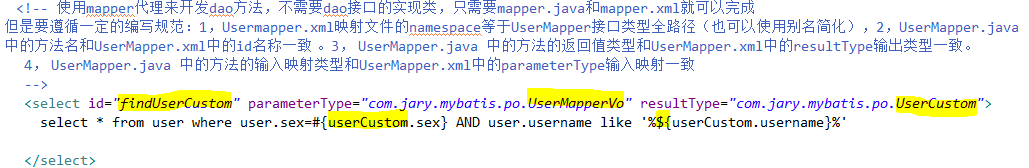
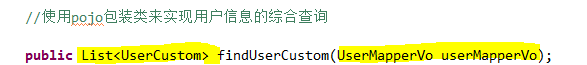
<mapper resource="mapper/UserMapper.xml" />

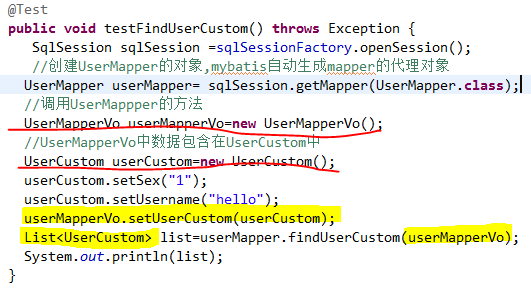
通过mapper接口加载单个mapper,要遵循一定的规范：   
（1）前提是使用mapper代理开发（已经有4个规范）   
（2）需要将mapper接口类名和mapper.xml映射文件名称保持一致，且在同一目录下   


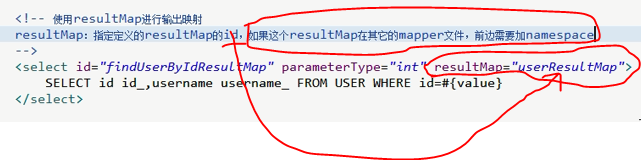
<mapper class="com.jary.mybatis.mapper.UserMapper" />

**通过批量加载mapper（推荐使用）**：实现条件是   
要满足mapper接口加载映射文件和使用mapper代理开发同时满足   
   
**mybatis的一些细节剖析结束**

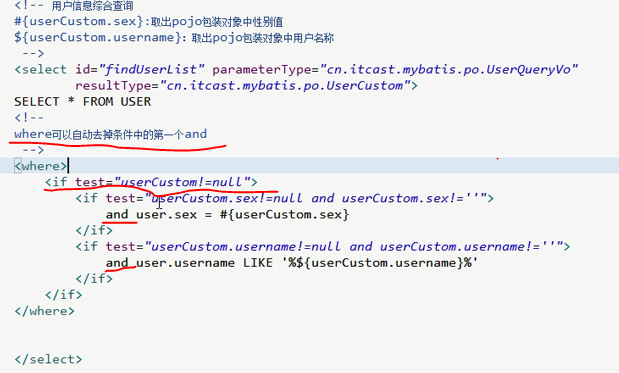
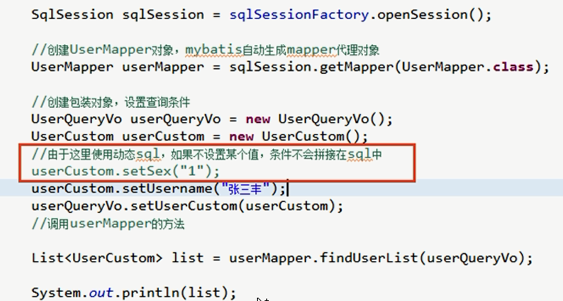
**mybatis的核心输入映射和输出映射开始：**

**输入映射**   
通过parameterType指定输入参数类型，类型可以是简单类型、hashmap、pojo的包装类型   
1，传递pojo的包装对象   
（1）需求   
完成用户信息的综合查询，需要传入查询条件复杂（可能包括用户信息，商品信息，商品订单等），这样靠一个parameterType只能传入一个输入参数，所有需要pojo的包装类型来实现   
（2）定义包装类型pojo   
针对上面的需求，在包装类型的pojo中把这些复杂的查询条件包装进去，定义包装类UserQueryVo，把需要查询的条件全部定义在里面   
   
上图中标注的用户查询条件使用的是User的扩展类（**因为User类一般是由逆向工程自动生成的，不要进行修改，所有使用的扩展类来实现**）   
映射文件UserMapper.xml配置   
  
UserMapper.java接口文件的配置   
   
上图中，把包装类作为参数传入，返回值是一个用户列表所以用list集合接收

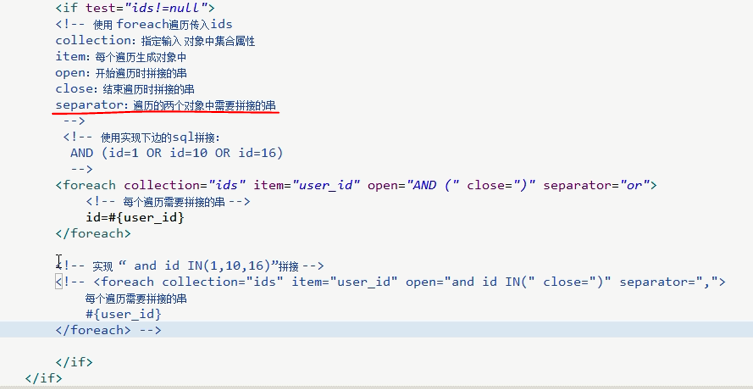
测试代码如下图：   


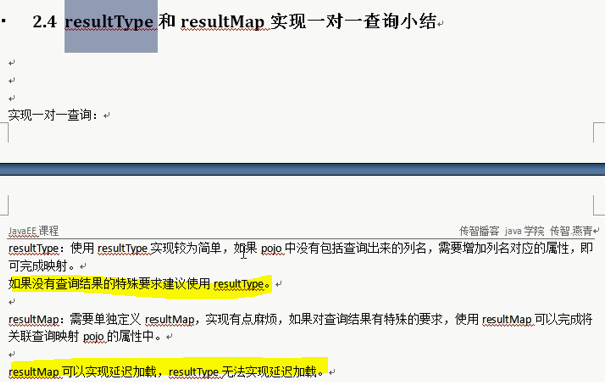
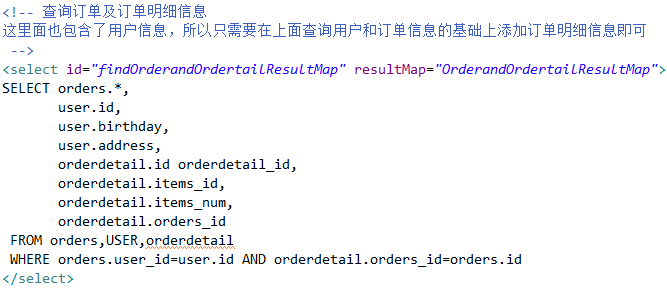
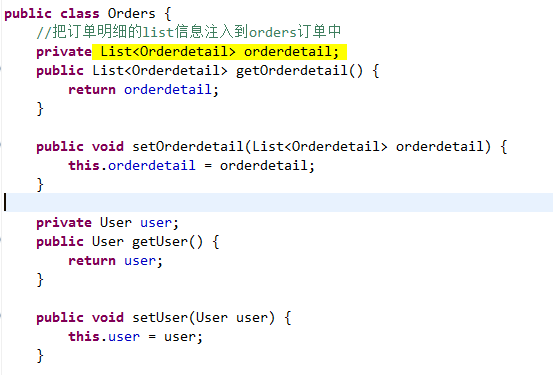
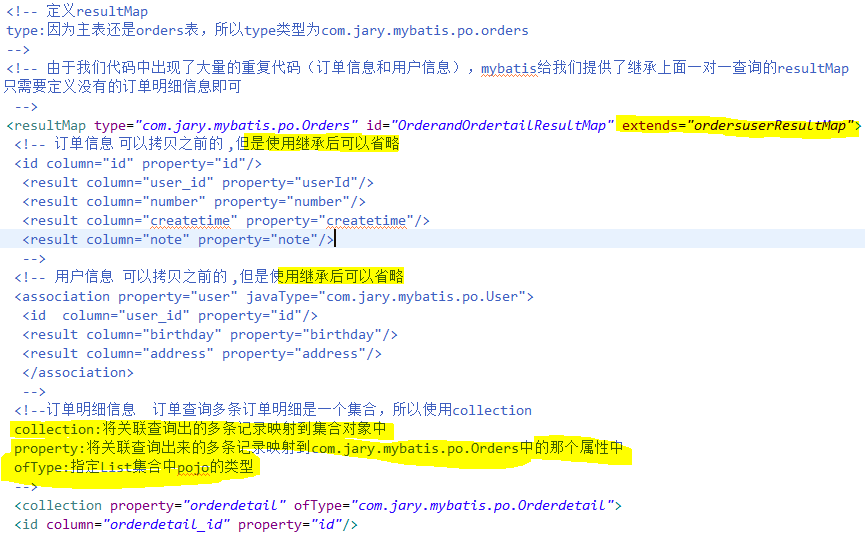
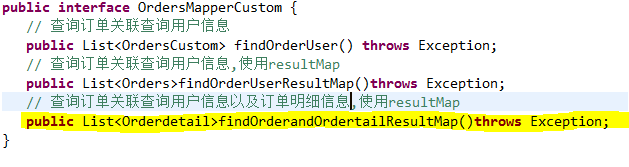
**输出映射**   
1，resultType   
**使用resultType进行输出映射时，只有查询输出结果列名和pojo中的属性名一致才可以，映射成功**   
如果查询出来的列名和pojo中的属性名没有一个一致的，就不会创建pojo对象   
如果查询出来的列名和pojo中的属性名有一个一致，就会创建pojo对象   
输出pojo对象和pojo列表   
不管是输出的pojo单个对象还是一个列表（list中包含pojo），在mapper.xml中resultType指定的类型是一样的   
在mapper.java指定的方法返回值类型不一样：   
（1）输出单个pojo对象，方法返回值是个单个对象类型   
（2）输出pojo对象list，方法返回值就是list对象类型   
在动态代理对象中，是根据mapper方法的**返回值类型**来确定是调用selectOne（返回单个对象)还是selectList（返回集合对象）   
2，resultMap   
使用resultMap进行映射时，查询结果列名和pojo的属性名不一致时，resultMap会对列名和pojo属性名进行映射，保证其成功映射   
使用resultMap需要这二步：   
（1）定义resultMap   
   
（2）使用resultMap作为statement的输出映射类型   
   
**mybatis的核心输入映射和输出映射结束：**

**mybatis的动态sql和sql片段开始：**

动态sql   
mybatis核心就是对sql语句进行灵活操作，通过**表达式**进行判断，对sql进行灵活拼接和组装。   
需求：用户信息查询的综合信息需要使用动态sql   
对查询条件进行判断，如果出入参数不为空，才进行拼接   
   
测试代码需要注意的是如下图：   
   
如果不设置某个值，条件将不拼接在sql中

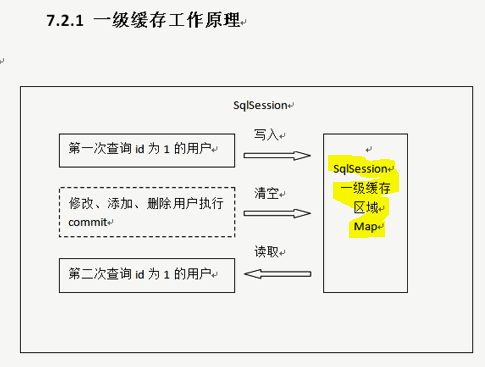
sql片段   
需求：将上面的动态sql（重复的sql语句）抽取出来做成一个片段，方便其他statement语句重复使用sql片段   
   
使用sql片段   


使用foreach标签遍历   
给sql传入数组或者集合时，mybatis使用foreach解析   
需求：在用户信息的综合查询中增加多个id传入   
sql语句如下：   
select \* from user where id=1 or id=3 or id=5   
也可以使用select \* from user where id in(1,3,5)   
  
**mybatis的动态sql和sql片段结束**

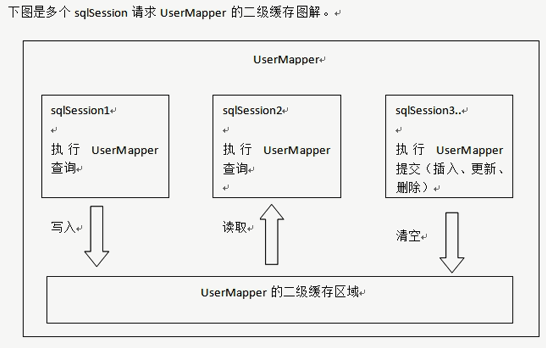
**mybatis高级映射开始：**   
一、 高级映射一对一查询（**使用到assocition标签实现关联对象的一对一查询映射**），分别使用resultType和resultMap，并且比较二者的区别   
   
还有一点就是，resultType查询关联列表结果列如果和pojo属性名不一致，需要自己创建扩展类（继承包括结果集列名多的pojo对象，这样可以少写一点属性名）。resultMap则不需要创建扩展类，而是把关联信息的对象注入，从而实现结果集列名和pojo属性名保持一致。   
二、高级映射一对多查询（**使用collection标签来实现关联对象的一对多查询映射**），一对多，就是关联的对象查询结果是一个List集合   
开发步骤：   
（1）首先写sql语句   
需求：查询订单及订单明细信息（一个订单包含多个订单明细，所以一对多）   
主表：订单表   
关联表：订单明细表   
经过之前一对一查询（用户的订单）的分析，我们只需要在此基础上关联订单明细表即可。   
   
（2）pojo类(resultType时用到扩展类，这里我们使用resultMap不需要包装类)   
   
（3）mapper.xml（**这里我们使用了resultMap的继承，不需要重新关联订单表和用户表，通过继承之前的一对一（用户查询订单）所写的二个表，可以减少大量的重复代码。同时使用了collection集合标签将关联查询出的多条记录映射到List集合中**）   
  
(4)mapper.java   


一对多查询总结：   
mybatis使用resultMap的collection（resultType没有此标签）对关联查询的多条记录映射到一个list集合中。   
使用resultType通过双重循环遍历可以实现，去除重复记录，将订单明细映射在orderdetail中（了解）   
多对多查询实例和上边类似。主要是搞清楚各个表之间的对应关系，订单的collection中嵌套订单明细的collection，而订单明细的collection中嵌套商品信息。

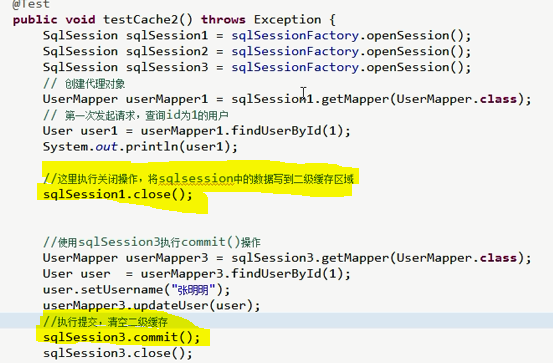
**mybatis高级映射结束**

**mybatis的延迟加载和缓存技术开始**   
mybatis一级缓存   
   
mybatis的二级缓存

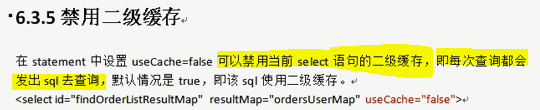
mybatis默认是没有开启二级缓存的。   
开启二级缓存需要在mybatis的全局配置文件sqlMapConfig.xml中加入   
  
**除了开启二级缓存开关外，还需要在各自的mapper.xml中开启二级缓存。**   
这里写图片描述

原理图：   
   
如上图：sqlsession1去查询id为1的用户信息，查询到用户信息就会查询数据存放在二级缓存区域（**hashmap**）中  
sqlsession2去查询id为1的用户信息，首先去缓存中查找是否存在数据，如果存在就直接从二级缓存中取出数据。   
二级缓存和一级缓存的区别：二级缓存的范围更大，多个sqlsession   
可以共享usermapper的二级缓存。   
二级缓存是根据mapper的namespace来划分的，相同namaspace下的mapper共享二级缓存，反之   
如果sqlsession3去执行相同mapper下sql，并执行commit（）操作，则清空该命名空间下的二级缓存

二级缓存的测试代码：

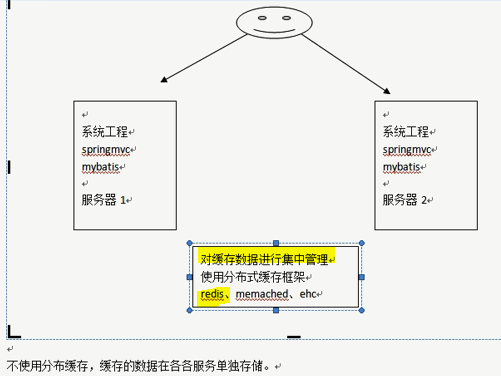


上面涂黄部分要特别注意，sqlsession关闭时才可以把数据写到二级缓存区域中，如果本namespace下的sqlsession执行了commit（）操作，二级缓存就会清空

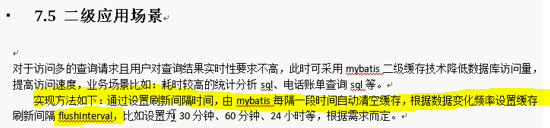
**禁用二级缓存**   
也可以禁用单个查询的二级缓存，这样要保证每次查询的都是最新数据。   


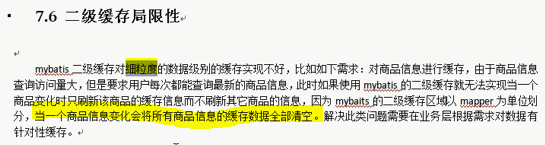
**刷新二级缓存（就是清空缓存，切记）**   
   
总结：一般情况下，执行commit()操作之后，都要刷新缓存，因此flushCache都设为true，来避免数据的脏读。

**mybatis cache的参数设置**   
flushInterval(刷新间隔)，可以为任意整数，单位为毫秒值，这个比较有用。

**mybatis和第三方分布式缓存框架整合（ehcache，[Redis](http://lib.csdn.net/base/redis" \o "Redis知识库" \t "_blank),memcache）**   
mybatis在缓存方面还是比较弱，特别是分布式缓存不支持   
   
我们的系统为了提高系统并发，性能，一般对系统进行分布式部署（集群部署方式）

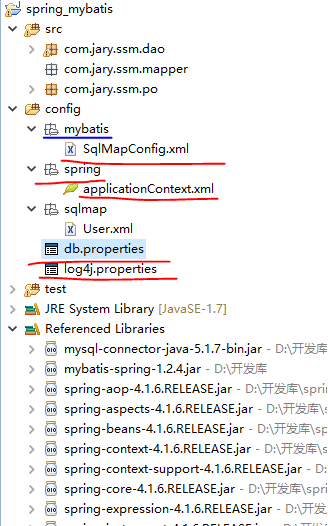
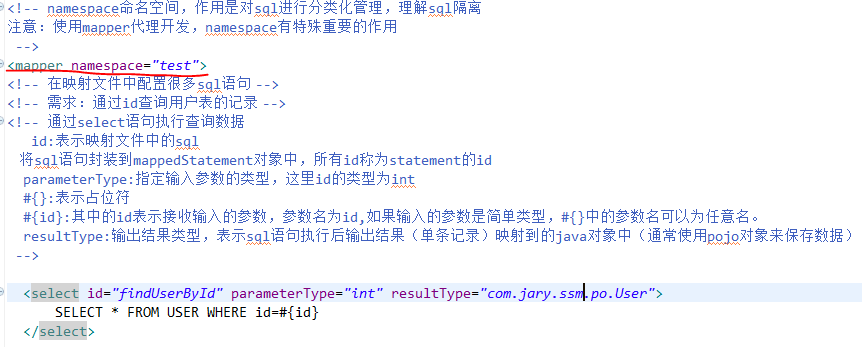
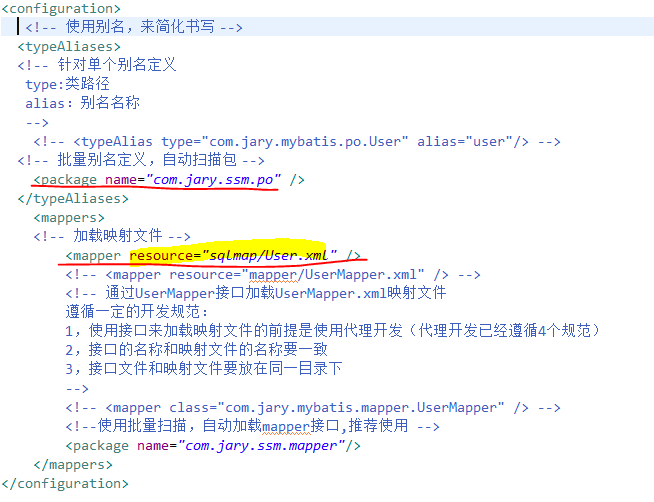
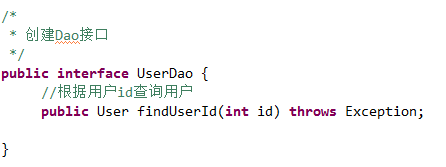
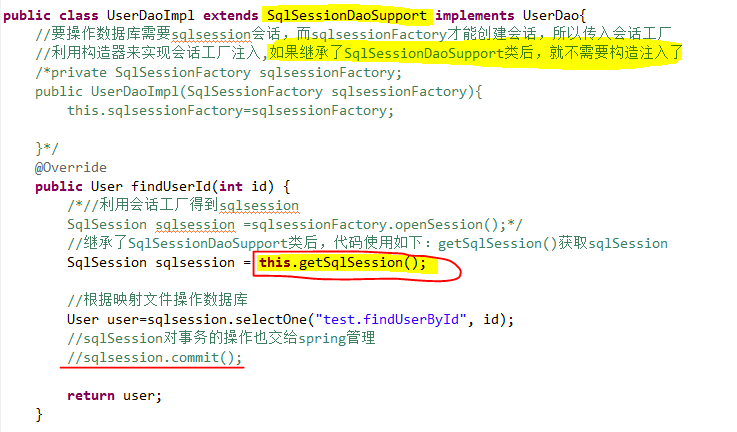
**整合方法**   
mybatis提供了一个cache接口，如果要实现自己的缓存逻辑，实现cache接口即可   
mybatis和ehcache整合，mybatis和ehcache整合包中提供了一个cache接口的实现类。   


二级缓存的应用场景（实际开发中用 刷新间隔）   


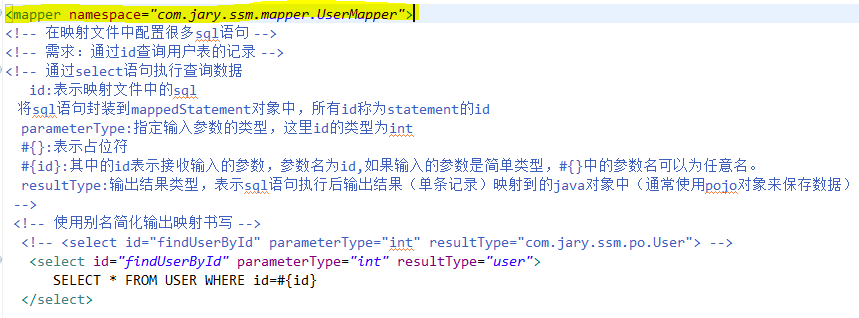
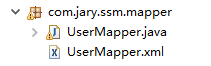
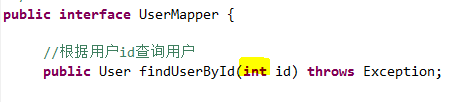
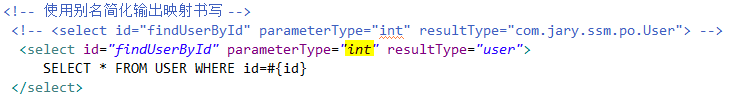
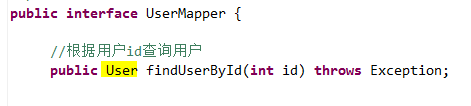
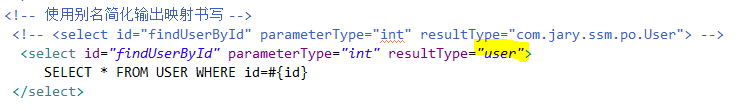
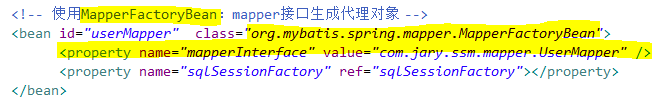
二级缓存额局限性   
   
**细粒度缓存,就是修改一个商品信息（即执行commit（）方法），只修改缓存中这一个商品的信息，其余的信息不清空。**

**mybatis的延迟加载和缓存技术结束**

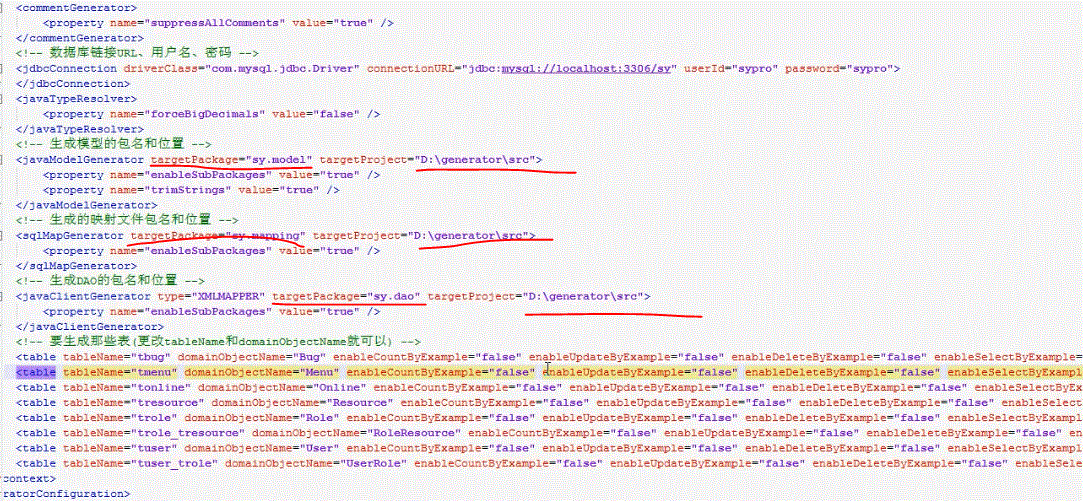
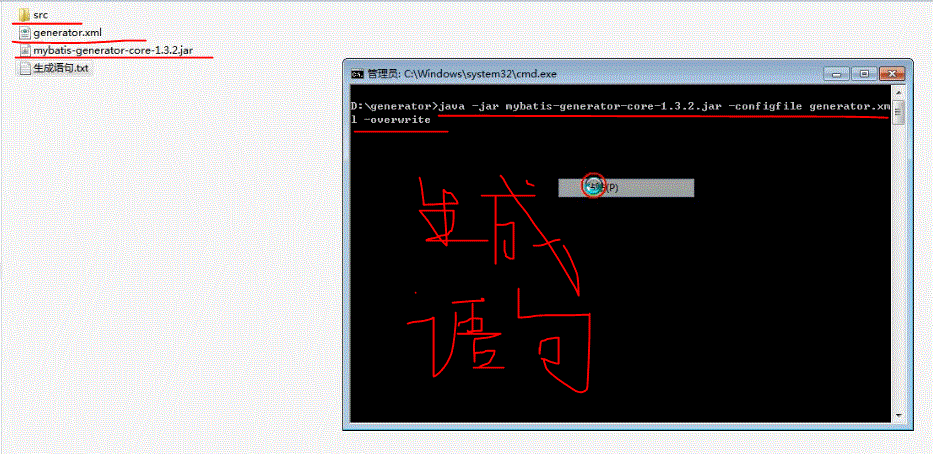
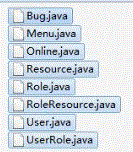
**mybatis和[spring](http://lib.csdn.net/base/javaee" \o "Java EE知识库" \t "_blank)的整合（重点掌握）开始**

1、整合的思路：   
（1）首先需要spring通过单例方式管理sqlSessionFactory   
（2）spring和mybatis整合生成代理对象，使用SqlSessionFactory创建sqlSession会话（此步是由spring和mybatis整合自动完成）   
（3）持久层的mapper，dao都需要由spring来管理   
2、环境的搭建   
   
上面的sqlmap目录下的User.xml是为了原始dao开发使用的。还要加载spring，mybatis，mybatis-srping等jar包。   
3、在spring的配置文件中配置sqlSessionFactory和数据源   
sqlSessionFactory在mybatis和spring的整合包下   
   
上图中：使用C3P0配置数据库连接池，属性参数名要按照规定写，不能自己定义，否则会报错，而使用dbcp就可以自定义参数名，这点注意。   
在加载配置文件时，都要加上类路径名classpath   
在使用原始dao开发时，属性name值要与UserDaoImpl类中变量名一致（特别是大小写）   
4、***\*原始Dao的开发***（和spring整合后）\*   
4.1 User.xml(也称mapper.xml更准确)   
  
和spring整合后，需要使用spring来管理mapper，spring配置文件为applicationContext.xml   
还有mybatis的配置文件来加载User.xml   
   
4.2 UserDAO   
   
基本上不用改变   
4.3 UserDaoImpl（**重点在Dao的实现类上**）   
  
上图中的代码最重要的就是继承了SqlSessionDaoSupport通过this.getSqlSession()来得到SqlSession会话   
这里也不需要写sqlSession的事务提交（更新操作不用写）和sqlSession关闭   
4.4 测试代码   


**5、使用mapper代理来开发 （mybatis和spring整合后**）   
和利用原始dao开发差不多，只是不需要dao接口的实现类   
而是根据一个规范来实现dao接口生成代理对象

5.1规范：   
（1）mapper.xml中的namespace命名空间等于mapper.java的全路径名   
  
（2）mapper.xml和mapper.java应在同一个目录下   
   
（3）mapper.xml中的statement语句的输入参数paramType类型应与mapper.java中方法中传递的参数类型一致   
   
  
（4）mapper.xml中的statement语句的输出参数resultType类型应与mapper.java中方法返回值类型一致   
   
  
5.2 让spring来管理mapper，在配置文件中   
   
重点在这里，使用mybatis和spring的整合包中MapperFactoryBean来实现mapper接口生成代理对象   
属性值有mapperInterface和sqlSessionFactory   
**总结：此方法存在一个大问题，需要针对每个mapper进行配置，太麻烦**   
**终极解决方案：** 通过mapper批量扫描，从mapper包中扫描出mapper接口，自动创建代理对象并且在spring容器中注册   


<property name="sqlSessionFactoryBeanName" value="sqlSessionFactory" />

上面代码，不能使用ref而是使用value,刚开始用错了。   
开发中推荐使用自动扫描，   
**mybatis和spring的整合（重点掌握）结束**   
**mybatis的逆向工程**（了解会用就行）   
  
generator.xml的配置，这里要记住，**上图中最下边table后面的一长串值等于false的属性，是为了不生成其他与我们无关的example等代码**   
下面需要mybatis-generator-core-1.3.2.jar和generator.xml文件在同于目录下，并且建立src目录接收生成的文件   
  
生成后的如下图   
   
没有了example的无用类了，比较干净，推荐使用