Mybatis第二天

课程安排：

以订单商品案例学习mybatis的关联查询：

学习订单商品数据模型（数据表、表之间关系）(掌握)

使用mybatis实现：

一对一

一对多

多对多

使用resultType和resultMap实现。

resultType(掌握)

resultMap（了解）

mybatis延迟加载（了解）

mybatis缓存：

一级缓存(了解)

二级缓存(掌握性能优化方法)

Mybatis和spring整合：(重点)

Mybatis+spring+springmvc整合

Mybatis逆向工程：（掌握，企业实际开发中使用较多）

由数据表生成mapper.xml、mapper.java、po类

工作流案例系统：企业采购系统

架构：Mybatis+spring+springmvc

系统部署

# 复习

## Mybatis是什么？

是apache的开源项目，是一个持久层的框架，对jdbc进行封装，是一个不完全ORM框架。需要程序员手动编写sql，自动去完成向sql中映射输入参数，sql查询结果映射成java对象。

Mybatis应用场合：对需求变更大的项目，关系数据模型不固定，比如互联网项目，建议使用mybatis，灵活去编写sql语句，对sql语句进行修改、优化方便。

## Mybatis开发过程

1. 配置SqlMapConfig.xml（全局配置文件，名称不固定）
2. 配置XXXMapper.xml映射文件
3. 根据配置创建SqlSessionFactory

SqlSessionFactory在实际使用时，建议使用单例模式

1. 根据SqlSessionFactory创建Sqlsession

Sqlsession是一个面向用户的接口，是线程不安全的，最佳应用场合是方法体内

SqlSession提供操作数据库的常用方法，内部使用Exceutor（基本执行器、缓存执行器）底层封装类操作数据库。

1. 通过Sqlsession操作数据库

对于插入、删除、更新进行事务提交

SqlSesssion使用完毕进行关闭。

## 动态Sql

#{}或${}：

#{}是占位符。(建议使用#{})

${}是拼接sql符号。

If判断：根据条件进行sql 拼接。

Sql片段：将公用的sql(查询条件、表的查询列)抽取出来组成 一个sql片段，在其它的mapper.xml中就可以引用了（如果跨namespace，引用时：namespace.sql片段的id）

## resultType和resultMap区别

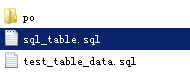
resultType：指定sql查询结果集中单条记录所映射的java对象，要求sql查询的列名和java对象(pojo)的属性名必须一致方可映射成功。

resultMap：如果 sql查询的列名和java对象(pojo)的属性名不一致，定义一个resultMap将sql查询的列名和java对象(pojo)的属性名作一个对应关系，最终将sql查询结果映射到java对象中。

# 订单商品数据模型

环境准备：

订单商品数据模型包括的数据库表：



在mysql数据库执行sql\_table.sql。



需求：**用户**购买（一次可以购买多个）商品，需要创建**订单**（可以创建多个），一个订单中包括了（购买的商品、购买数量、购买价格）

上边表之间的关系：

1. 搞清楚数据库级别表之间的关系（外键关系）
2. 搞清楚业务关系（根据具体业务需求分析）

用户表：user

记录了购买商品的用户信息

Id：唯一标识一个用户

订单表：orders

记录了用户创建的订单

创建订单的用户:user\_id(外键)

订单号

创建时间

订单状态

外键：

User\_id

订单明细表：orderdetail

记录了用户购买信息

所属订单：orders\_id（外键）

商品id:item\_id（外键）

商品数量

商品购买价格

外键：orders\_id

商品信息：items

记录了所有商品信息

商品id：id（主键）

商品名称：

商品介绍

商品价格

外键：

item\_id

一对多

用户可以创建多个订单

一对一：

一个订单只能由一个用户创建

一对多

一个订单包括多个购买明细

一对一

一个订单明细只能

属于一个订单

一对一

一个明细对应一个商品

一对多

一个商品对应多个订单明细

通过订单明细表订单表和商品表建立 关系

一个订单对应多个商品

一个商品对应多个订单

订单表和商品表是多对多关系

**注意：在分析表之间的业务关系时一定要在某个业务意义基础上来分析。**

# 订单商品一对一查询

## 需求

查询所有订单信息，关联查询订单所属用户信息。

## 方法1使用resultType

### Sql语句：

主查询表：根据需求条件显示主体信息，不能因为和关联表关联导致主体信息无法显示。

订单信息表orders

关联查询表（用户表：user）：显示关联信息，使用内链接。

使用内链接还是外链接：

内链接：如果主查询表和关联查询表存在外键关系就可以使用内链接。

外链接：主查询表中和关联表没有外键关系，查询的主表信息一部分在关联查询表中，此时要用外链接。

SELECT

orders.\*,

user.username,

user.address

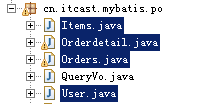
FROM

orders,

USER

WHERE orders.user\_id = user.id

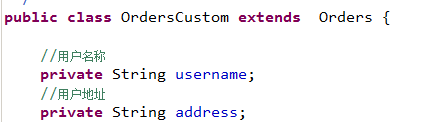
根据表结构创建po类：



### maper.xml：

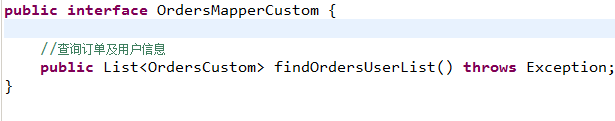
定义statement时，resultType指定pojo包括上边sql查询的所有字段（订单信息、用户信息），需要自定义一个pojo继承原始的po类，自定义的pojo中包括订单信息、用户信息。

开发经验：自定义的pojo建议继承sql查询列较多的po类。





### Mapper.java

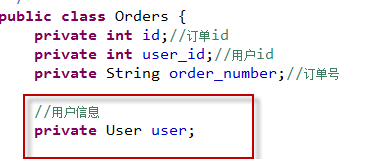


## 方法2使用resultMap

### 实现目标

通过订单查询用户，一个订单只能关联查询出一个用户，在订单的po类中创建一个user属性（记录用户信息），让resultMap将sql查询结果集中用户信息映射到订单po类的user属性中。

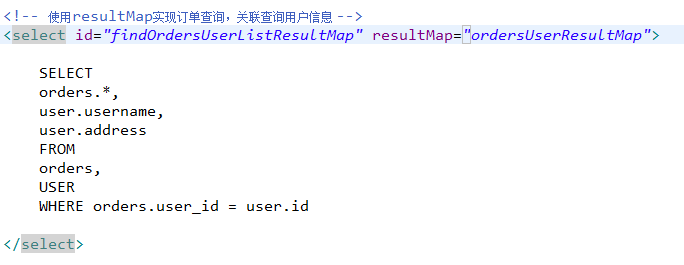
### 在Orders.java中添加User属性



### Mapper.xml

定义resultMap：





### Mapper.java



## resultType和resultMap对比

resultType：实现查询时，需要自定义pojo，pojo的属性名和sql查询列名一致。企业开发中resultType简单方便建议使用。

resultMap：可以将sql查询结果信息中部分属性映射到一个pojo中。需要程序员进行映射配置，比较麻烦。

针对一对一查询建议使用resultType。

如果一对一查询要使用mybatis提供的延迟加载 功能，必须使用resultMap。

# 订单商品一对多查询

## 需求

查询所有订单信息及订单下的订单明细信息

## Sql语句

主查询表：订单信息

关联查询表：订单明细信息，使用内链接

SELECT

orders.\*,

user.username,

user.address,

orderdetail.id orderdetail\_id,

orderdetail.item\_id,

orderdetail.item\_num,

orderdetail.item\_price

FROM

orders,

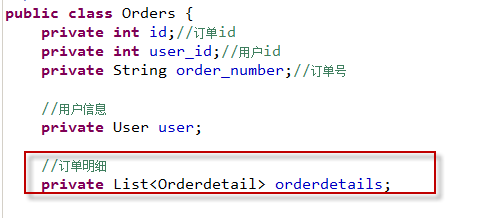
USER,

orderdetail

WHERE orders.user\_id = user.id AND orders.id = orderdetail.orders\_id

## 使用resultMap将订单明细信息映射到Orders.java的属性中

在Orders.java创建订单明细属性（是一个集合对象，因为一个订单对应多个明细）。

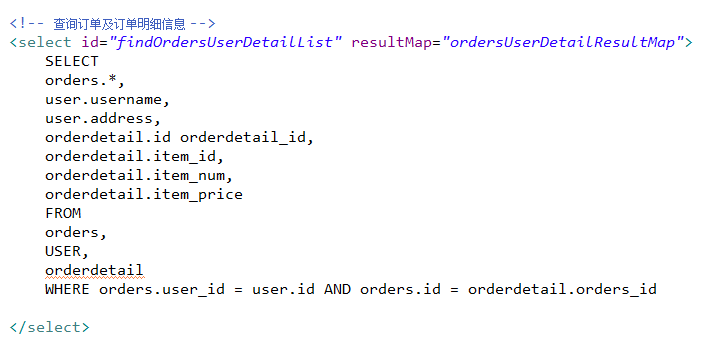


目标：将订单及订单明细映射到orders类中，将明细信息（多个）映射到List<Orderdetail> orderdetails。

## Mapper.xml

定义resultmap：





## Mapper.java



## 小结

使用resultType是无法将查询结果映射到一个list集合对象中，只能使用resultMap将查询结果映射到list集合对象中。

# 订单商品多对多查询

## 需求

查询所有订单信息及订单明细的商品信息

## Sql语句

主查询表：订单表

关联查询表：用户表、订单明细表、商品表

SELECT

orders.\*,

user.username,

user.address,

orderdetail.id orderdetail\_id,

orderdetail.item\_id,

orderdetail.item\_num,

orderdetail.item\_price ,

items.item\_detail,

items.item\_name,

items.item\_price item\_price\_price

FROM

orders,

USER,

orderdetail,

items

WHERE orders.user\_id = user.id

AND orders.id = orderdetail.orders\_id

AND orderdetail.item\_id = items.id

## 使用resultMap将订单明细及商品信息映射

因为商品信息和订单明细有外键关联，在订单明细po类添加商品信息属性：



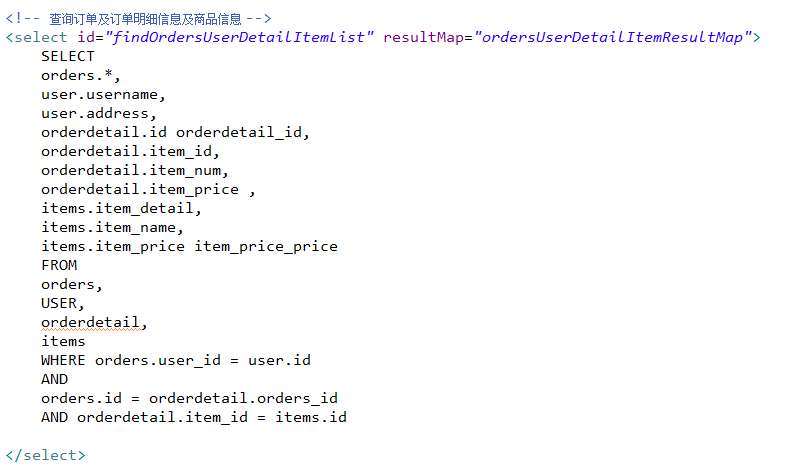
目标：resultMap将订单明细映射到orders.java中的List<Orderdetail> orderdetails，将商品信息映射到Orderdetail.java中商品属性中(Items items)。

## Mapper.xml

定义resultMap：

目标：resultMap将订单明细映射到orders.java中的List<Orderdetail> orderdetails，将商品信息映射到Orderdetail.java中商品属性中(Items items)。





## Mapper.java



## 小结

使用resultType是无法将查询结果映射到一个list集合对象中，只能使用resultMap将查询结果映射到list集合对象中。

# 延迟加载

延迟加载意义：在需求允许的情况下，先查询单表，当需要关联其它表查询时，进行延迟加载，去关联查询，达到目标：不需要关联信息时不查询，需要时再查询。好处：提高数据库的性能。

## 需求

查询订单信息，关联查询用户信息。

延迟加载需求：首次只查询订单信息，当需要关联查询用户信息时，再查询用户信息。

## 方法1不使用mybatis提供延迟加载特性

定义两个mapper接口。

1. 查询订单信息列表
2. 根据用户id查询用户信息

查询过程：

首次查询只调用第一个mapper接口（查询订单信息列表）

当需要查询某个订单的用户信息时，从订单信息中获取用户id，调用第二个mapper（根据用户id查询用户信息）。

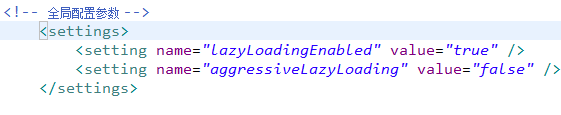
## 使用mybatis提供延迟加载 特性

### 打开延迟加载开关

在SqlMapConfig.xml中配置setting全局参数：

lazyLoadingEnabled：延迟加载的总开关，设置为true

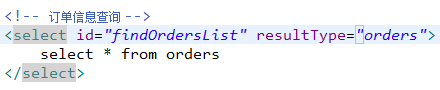
aggressiveLazyLoading：设置为false，实现按需加载（将积极变为消极）



### 定义mapper.xml

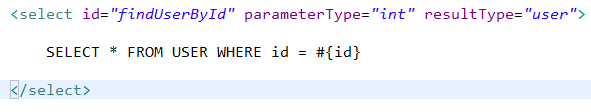
定义statement，只查询订单。

在OrdersMapperCustom.xml定义：



还需要定义一个根据用户id查询用户信息的statement。

在UserMapper.xml定义：



配置延迟加载 ：

要根据订单信息中的用户id(外键)查询用户信息，修改上边订单信息查询的statement。

将resultType改为resultMap，

定义resultMap，配置延迟加载：



### Mapper.java



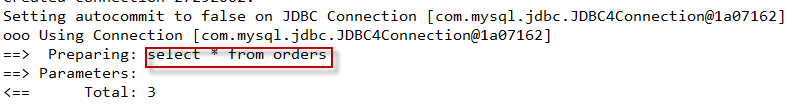
### 测试

调用上边的mapper接口，查询到仅仅是订单信息。

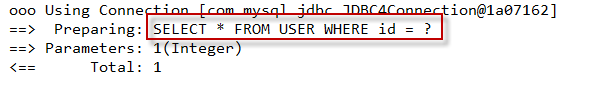
遍历订单信息，如果要获取用户信息（调用getUser()方法），进行延迟加载，调用查询用户信息的sql。

测试跟踪：

查询订单信息，日志发现只能有一条sql（查询订单信息）



当调用getUser()方法，日志中打出查询用户信息的sql



## 小结

使用association关联查询单个对象和使用collection关联查询集合对象都支持延迟加载。

# 缓存

## 为什么使用缓存

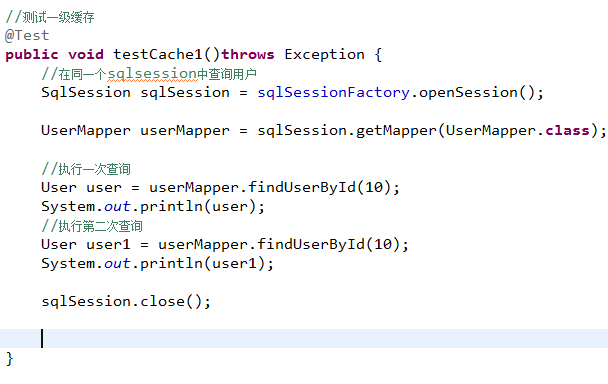
将从数据库中查询出来的数据缓存起来，缓存介质：内存、磁盘，从缓存中取数据，而不从数据库查询，减少了数据库的操作，提高了数据处理性能。

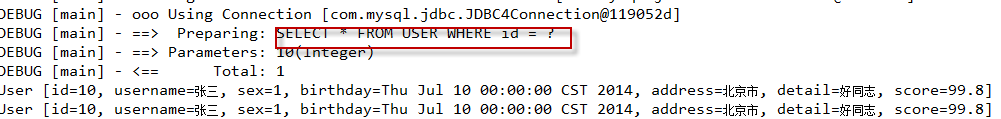
## 一级缓存

Mybatis默认提供一级缓存，缓存范围是一个sqlSession。

在同一个SqlSession中，两次执行相同的sql查询，第二次不再从数据库查询。

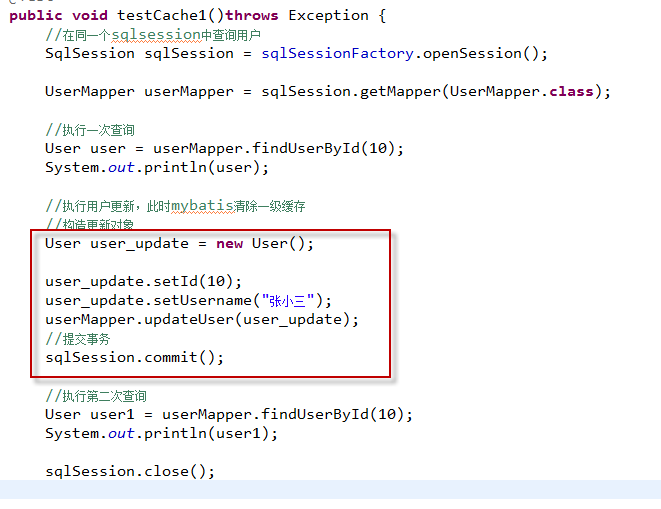
测试：

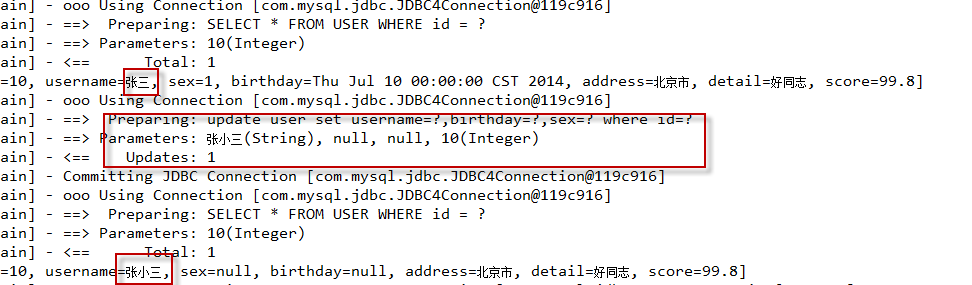




执行提交清除缓存测试：

如果第一次查询后，执行commit提交，mybatis会清除缓存，第二次查询从数据库查询。



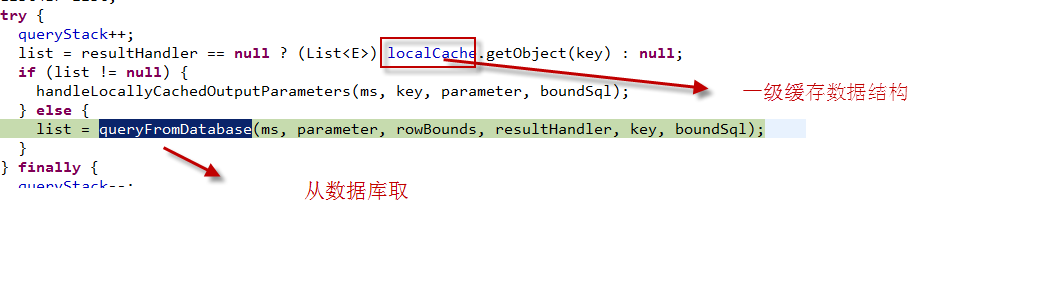


### 一级缓存原理：

一级缓存采用Hashmap存储，mybatis执行查询时，从缓存中查询，如果缓存中没有从数据库查询。

如果该SqlSession执行commit()提交，清除缓存。

Map的key：（code+。。statement的id+sql+输入参。。）



## 二级缓存

缓存范围是跨SqlSession的，范围是mapper的namespace，相同的namespace使用一个二级缓存结构。

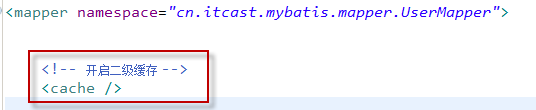
需要进行参数配置让mybatis支持二级缓存。

### 二级缓存配置

1. 在核心配置文件SqlMapConfig.xml中加入，表示打开二级缓存开关

<setting name=*"cacheEnabled"* value=*"true"*/>

1. 还需要在mapper.xml中配置是否打开该mapper的二级缓存。



### 二级缓存注意事项

* 实现序列化：

注意：将查询结果的pojo对象进行序列化实现 java.io.Serializable接口



* 如何清除缓存：

执行相同的statement，如果执行提交操作需要清除二级缓存。

如果想让statement执行后刷新缓存（清除缓存），在statement中设置flushCache="true" （默认值 是true）

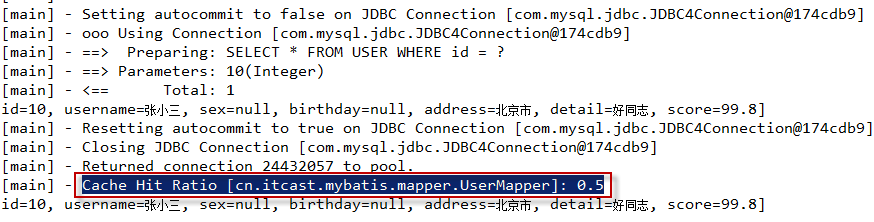
* 设置statement是否开启二级缓存

如果让某个statement启用二级缓存，设置useCache=true（默认值为true）

### 测试

在测试方法，创建多个Sqlsession。

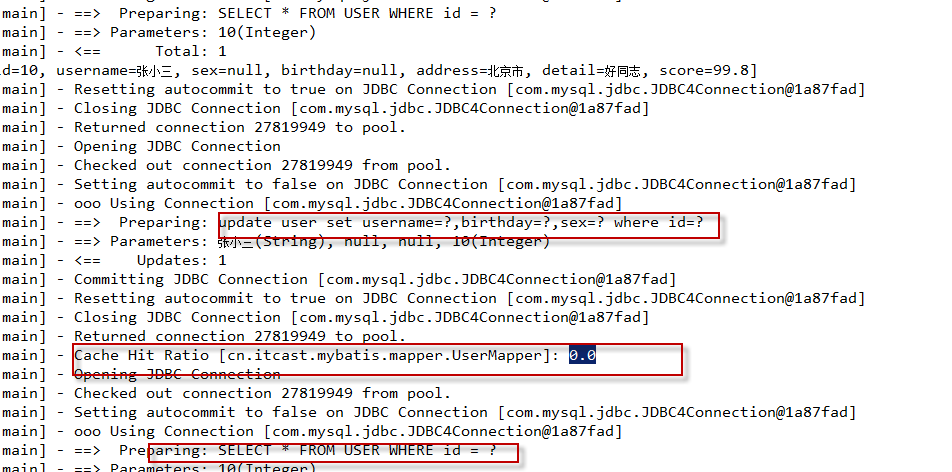




提交事务后，清除二级测试：

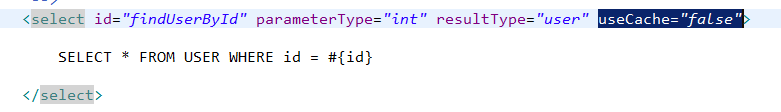
再创建一个SqlSession，执行用户更新，清除二级缓存。





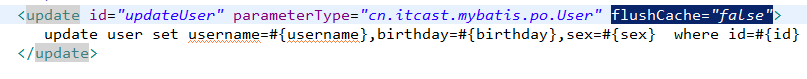
测试 useCache=”false”：

修改statement的useCache为false，表示该statement不再进行二级缓存



测试 flushCache="false"

修改更新用户的statement，



执行更新操作后，不刷新二级缓存。

### 二级缓存原理

如果二缓存开启，首先从二级缓存查询数据，如果二级缓存有则从二级缓存中获取数据，如果二级缓存没有，从一级缓存找是否有缓存数据，如果一级缓存没有，查询数据库。

### 二级缓存应用场景(重点)

1、针对复杂的查询或统计的功能，用户不要求每次都查询到最新信息，使用二级缓存，通过刷新间隔flushInterval设置刷新间隔时间，由mybatis自动刷新。

比如：实现用户分类统计sql，该查询非常耗费时间。

将用户分类统计sql查询结果使用二级缓存，同时设置刷新间隔时间：flushInterval（一般设置时间较长，比如30分钟，60分钟，24小时，根据需求而定）

2、针对信息变化频率高，需要显示最新的信息，使用二级缓存。

将信息查询的statement与信息的增、删、改定义在一个mapper.xml中，此mapper实现二级缓存，当执行增、删、修改时，由mybatis及时刷新缓存，满足用户从缓存查询到最新的数据。

比如：新闻列表显示前10条，该查询非常快，但并发大对数据也有压力。

将新闻列表查询前10条的sql进行二级缓存，这里不用刷新间隔时间，当执行新闻添加、

最佳的方案使用**页面缓存**。

### 二级缓存使用Ehcache

Mybatis控制二级缓存策略，二级缓存缓存介质使用Ehcache。

让Ehcache和mybatis进行整合。

配置过程参考笔记。

# Mybatis和spring整合

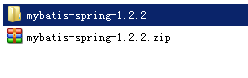
## 搭建环境

### Jar包

Mybatis3.2.3+spring3.1.4+springmvc3.1.4

Mybatis和spring整合：

从mybatis官方下载Mybatis和spring整合包：mybatis-spring-1.2.2.jar



Mybatis3.2.3+spring3.1.4+springmvc3.1.4的jar包括：

Mybatis核心和Mybatis依赖包

Mybatis和spring整合包

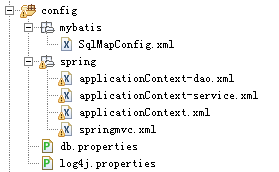
Spring的jar（包括springmvc的jar包）

数据库驱动包

第三方数据库连接池



### 配置文件



db.properties—数据库连接参数

log4j.properties---日志 配置文件

mybatis/SqlMapConfig.xml---mybatis全局配置文件

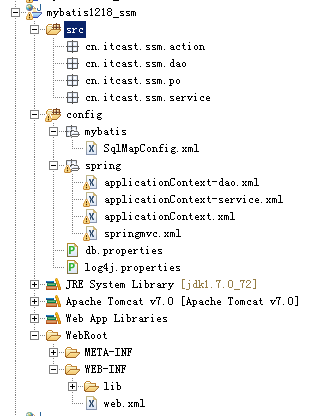
spring/springmvc.xml----------springmvc的全局配置文件

spring/applicationContext.xml---spring配置文件（配置公用内容：数据源、事务）

spring/ applicationContext- dao.xml---spring和mybatis整合的配置（SqlSessionFactory、mapper配置）

spring/ applicationContext-service.xml---配置业务接口

### 工程结构

****

## 整合思路

Spring容器

控制层：springmvc

业务层：service由spring管理理

持久层：mybatis

整合步骤：

1. 整合持久层

Mybatis和spring整合

整合目标：mapper创建由spring来管理

SqlSessionFactory由spring管理（设置单例 ）

1. 整合业务层

Spring管理service

整合目标：Service通过spring调用mapper

1. 整合控制层

因为springmvc是spring一个模块，只要加入jar包，配置springmvc.xml和web.xml即可。

整合目标：action中通过spring调用service

## 整合持久层

目标：mapper创建由spring来管理

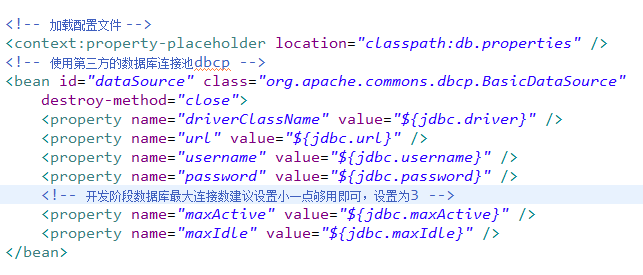
### 方法1通过spring开发原始dao

#### 准备环境：

applicationContext.xml

配置数据源：

开发阶段数据库最大连接数建议设置小一点够用即可。



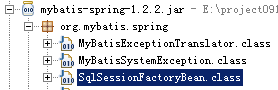
SqlMapConfig.xml

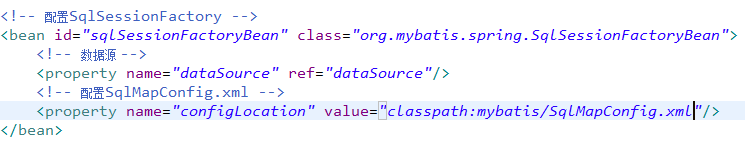


#### 让spring管理SqlSessionFactory

在applicationContext-dao.xml配置SqlSessionFactory

从mybatis和spring的整合包找SqlSessionFactory：

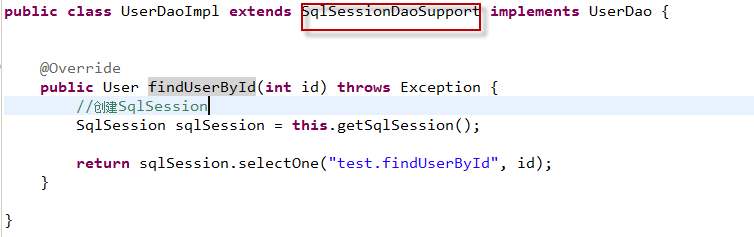




#### 开发dao

程序员需要写dao接口和dao实现类。

编写dao的实现类，继承SqlSessionDaoSupport

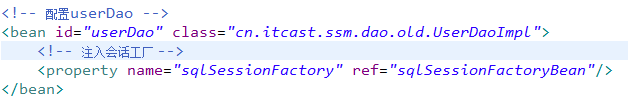


定义User.xml，且在SqlMapConfig.xml中配置。



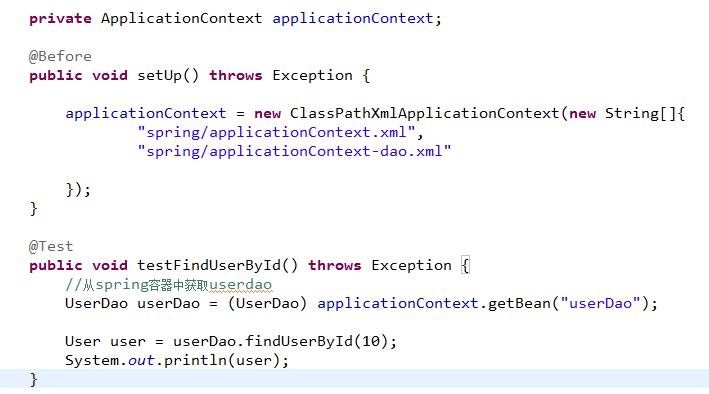


在applicationContext-dao.xml配置dao：



#### 测试

加载spring容器，从容器中取出UserDao的bean。



#### 小结

由于企业中原来使用的是ibatis，很多使用原始dao开发方法。

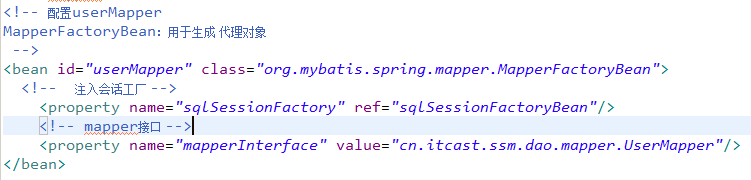
### 方法2通过spring生成mapper动态代理

Mybatis和spring整合使用mapper动态代理的开发方法

#### 开发mapper接口。



#### 在applicationContext-dao.xml配置mapper(让spring生成动态代理)



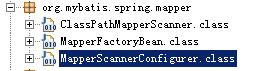
#### 问题

需要在spring容器中配置每个mapper接口，比较麻烦。

使用下边讲的扫描器来解决。

#### 使用mapper批量扫描器自动创建代理对象

从mybatis和spring整合包找扫描器。



在applicationContext-dao.xml中配置扫描器：



由于使用spring和mybatis整合的mapper扫描器，原来在SqlMapConifg.xml配置mapper项就可以省略了。

**使用扫描器简单方便，最终使用mapper扫描器方式。**

## 整合业务层

Spring管理service

整合目标：

Service通过spring调用mapper

由spring进行事务控制

### 开发service接口



### Service接口交给spring管理

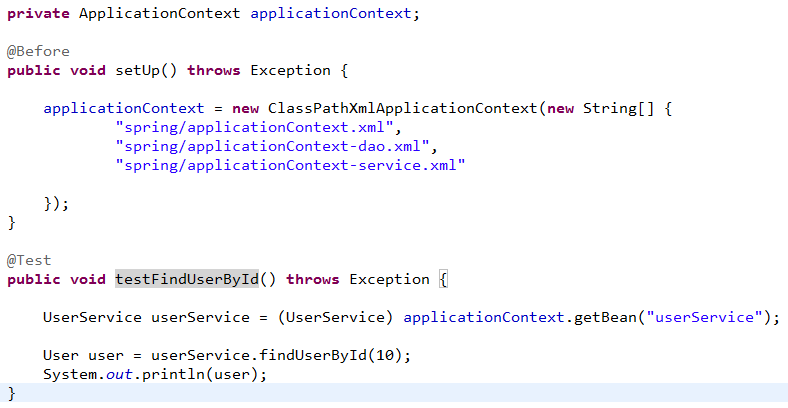
为了方便系统维护企业开发中建议使用配置方式，在applicationContext-service.xml配置service。



### 测试 service

加载spring容器，加载配置文件包括：





### 事务控制

为了规范程序员编码，采用声明式事务配置方式。

在applicationContext.xml配置事务管理：

<!-- 事务管理器

mybatis使用jdbc事务管理

-->

<bean id=*"transactionManager"* class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>

<!-- 数据源 -->

<property name=*"dataSource"* ref=*"dataSource"*/>

</bean>

<!-- 通知 -->

<tx:advice id=*"txAdvice"* transaction-manager=*"transactionManager"*>

<!-- 配置传播行为 -->

<tx:attributes>

<tx:method name=*"save\*"* propagation=*"REQUIRED"*/>

<tx:method name=*"insert\*"* propagation=*"REQUIRED"*/>

<tx:method name=*"update\*"* propagation=*"REQUIRED"*/>

<tx:method name=*"delete\*"* propagation=*"REQUIRED"*/>

<tx:method name=*"find\*"* propagation=*"SUPPORTS"* read-only=*"true"*/>

<tx:method name=*"get\*"* propagation=*"SUPPORTS"* read-only=*"true"*/>

<tx:method name=*"select\*"* propagation=*"SUPPORTS"* read-only=*"true"*/>

</tx:attributes>

</tx:advice>

<!-- aop配置 -->

<aop:config>

<aop:advisor advice-ref=*"txAdvice"*

pointcut=*"execution(\* cn.itcast.ssm.service.impl.\*.\*(..))"*/>

</aop:config>

### 测试事务

一个service方法是一个事务单元 。

创建一个service方法，方法中发起多次数据库操作，只要有一个次操作失败（抛出runtime异常）事务回滚，该方法中其它成功操作自动回滚。

Service方法：



## 整合控制层

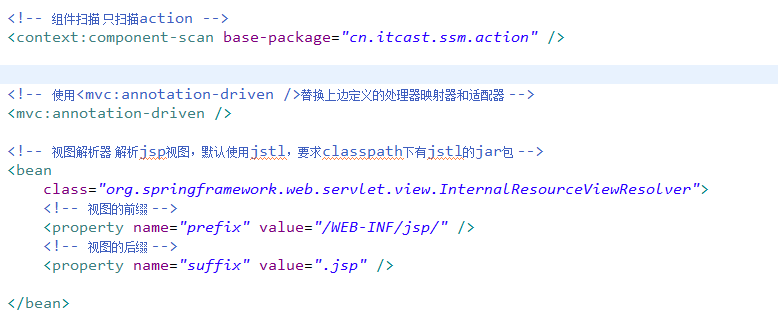
整合目标：action中通过spring调用service

### 准备环境

#### 配置Springmvc.xml：

Springmvc.xml,处理器映射器、处理器适配器(可以mvc注解驱动规则这两个处理器映射器、处理器适配器)，视图解析器。

配置action的组件扫描。



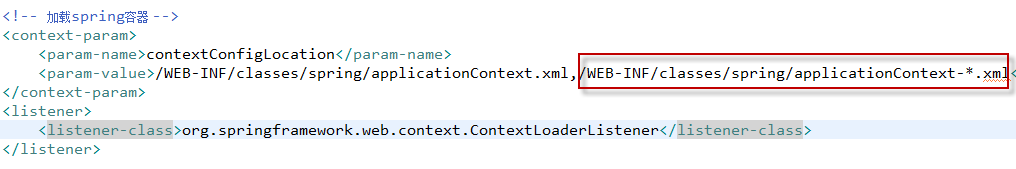
#### 配置前端控制器

在web.xml配置：

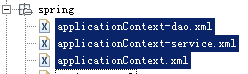


#### 在web.xml加载spring容器

通过统配符加载spring配置文件：

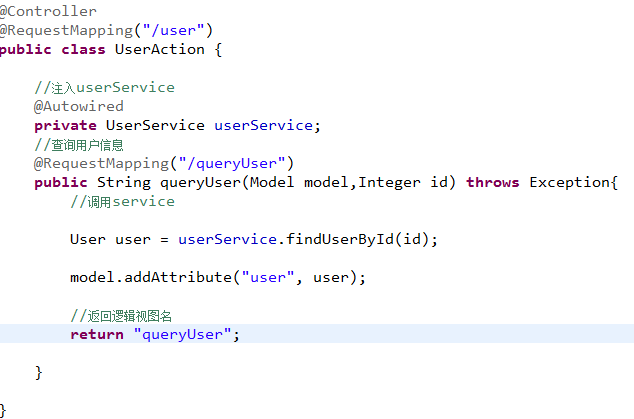


配置文件包括：



#### 编写action

需求：通过调用userService取出用户信息，在页面显示。



#### 编写页面



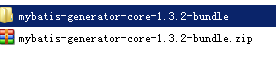
#### Tomcat部署测试

访问路径 ：http://localhost:8080/mybatis1218\_ssm/user/queryUser.action?id=10

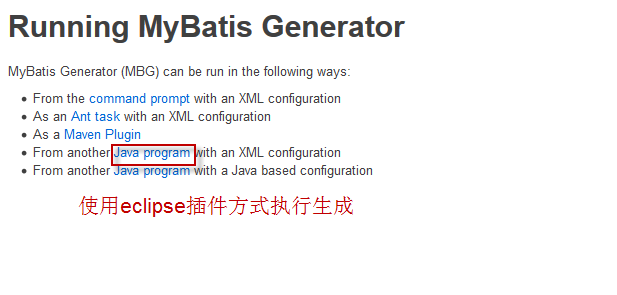
# Mybatis逆向工程

Mybatis官方提供逆向工程，实现由数据库表生成mapper.xml、mapper.java、po类 及相关类。

## 下载逆向工程



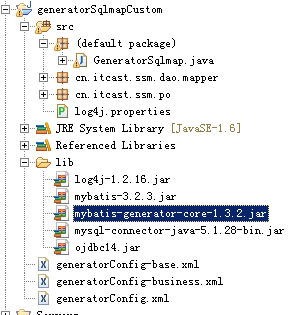
使用方式：



使用java程序方式执行逆向工程。

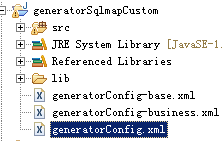
## 创建逆向工程

单独 创建java工程：generatorSqlmapCustom



## 逆向工程配置文件

添加配置文件，在配置文件逆向工程配置信息，根据配置信息生成表相关的mapper文件。



注意需要配置：

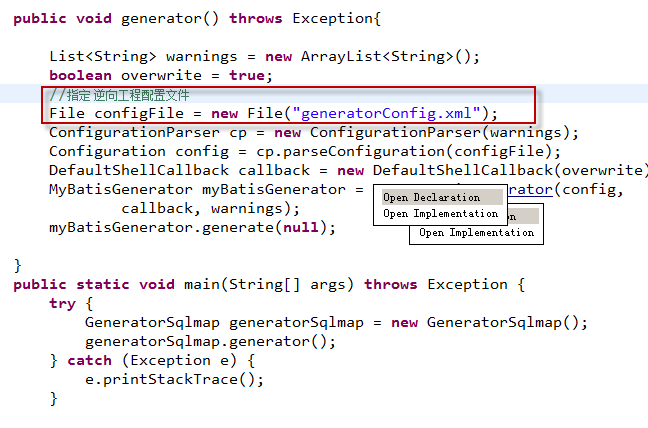
**数据库连接参数**

**Po类存放位置**

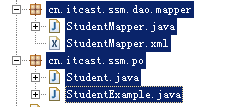
**Mapper映射文件和mapper接口的存放位置**

**配置表名**

## 写java执行逆向工程



## 拷贝 逆向工程中生成的代码拷贝到正式工程



注意：尽量不要修改自动生成的类。

## 学习生成mapper接口使用方法

//根据主键删除

@Test

**public** **void** testDeleteByPrimaryKey() {

studentMapper.deleteByPrimaryKey(4);

//自定义条件删除，符合条件的全部删除

//studentMapper.deleteByExample(example)

}

//添加信息

@Test

**public** **void** testInsert() {

//插入对象

Student student = **new** Student();

student.setName("赵六");

studentMapper.insert(student);

}

//根据查询条件查询信息

@Test

**public** **void** testSelectByExample() {

//自定义条件

StudentExample studentExample = **new** StudentExample();

StudentExample.Criteria criteria = studentExample.createCriteria();

//拼接查询条件

criteria.andSexEqualTo("2");//sex等于2

criteria.andNameEqualTo("王五");//姓名 等于王五

//根据查询条件查询返回集合对象

List<Student> list = studentMapper.selectByExample(studentExample);

System.*out*.println(list.size());

}

//根据主键查询信息

@Test

**public** **void** testSelectByPrimaryKey() {

Student student = studentMapper.selectByPrimaryKey(1);

System.*out*.println(student);

}

@Test

**public** **void** testUpdateByPrimaryKeySelective() {

//传入更新对象，必须要id，对象中不空的属性才更新

Student student = **new** Student();

student.setId(3);

student.setName("王五");

studentMapper.updateByPrimaryKeySelective(student);

}

//根据主键更新信息

@Test

**public** **void** testUpdateByPrimaryKey() {

//传入更新对象，必须要id，不管对象中属性是否为空全部更新

//必须先从数据库查询出原始记录

Student student = studentMapper.selectByPrimaryKey(3);

//设置更新内容

student.setName("王小五");

studentMapper.updateByPrimaryKey(student);

}

# 采购系统

需求：员工创建采购单、经理审核采购单。

员工和经理信息在pur\_sys\_user表中。

密码：全部111111。

导入mysql数据库：



在eclipse中导入 工程：

