mybatis第三部分 高级映射 查询缓存 和spring整合

# 课程计划

1. Ssm整合 收尾
2. 搞一个 maven的私服， svn server。 需要一个 maven私服。 10.0.13.200
3. 逆向工程 mybatis
4. Sql代码块
5. 事务 。
6. Ssm 项目

复习：

1. 容器 ssm使用了 2 种容器 ？（全局容器spring、局部容器springmvc）
2. 2种容器里面的javaBean的互相调用的情况、权限 ？
3. 问题： maven核心服务器、阿里云镜像、maven的私服 的区别和关系 ？
4. 什么是逆向工程 ？
5. 逆向工程的项目 同使用代码项目有什么要求 ？
6. 动态sql的概念 ？
7. 其他

复习：

1. 丢失更新 ？
2. 事务隔离性的传播级别 。

。

Maven的私服：

1. Linux unbutn
2. Jdk 安装好了

export JAVA\_HOME=/home/jdk8

export JRE\_HOME=${JAVA\_HOME}/jre

export CLASSPATH=.:${JAVA\_HOME}/lib:${JRE\_HOME}/lib

export PATH=${JAVA\_HOME}/bin:$PATH

1. Svn server 安装

apt-get update

apt-get install subversion

/home/svn/java2/conf/svnserve.conf

[general]

anon-access = none

auth-access = write

password-db = passwd

authz-db = authz

passwd 文件

authz 文件

在bash 下面执行 这个命令：

/usr/bin/svnserve -d -r /home/svn

1. 其他

# 查询缓存

## 缓存的意义

将用户经常查询的数据放在缓存（内存）中，用户去查询数据就不用从磁盘上(关系型数据库数据文件)查询，从缓存中查询，从而提高查询效率，解决了高并发系统的性能问题。（如果我们的系统是一个高并发系统）

User

表现层，展现层（mvc， struts

业务层（）

持久层（dao）， mybatis，hibernate

页面数据。可以缓存

查询业务，可以缓存

查询的结果可以缓存

## mybatis持久层缓存

mybatis提供一级缓存和二级缓存

## 一级缓存

### 原理

第一次发出一个查询sql，sql查询结果写入sqlsession的一级缓存中，缓存使用的数据结构是一个map<key,value>

key：hashcode+sql+sql输入参数+输出参数（sql的唯一标识）

value：用户信息

，

每次查询都先从缓存中查询：

sqlSession1

sqlSession2

sqlSesion3

二级缓存： 跨 sqlSession

如果缓存中查询到则将缓存数据直接返回。

如果缓存中查询不到就从数据库查询：

SqlSesson 的缓存，

Map（key， value）

Key：

Hashcode， statement（sql语句、输入参数、输出参数）

Select， 查询语句 。

Update， delete， modify

Select 2

### 一级缓存配置

mybatis默认支持一级缓存不需要配置。

**注意：mybatis和spring整合后进行mapper代理开发，不支持一级缓存，mybatis和spring整合，spring按照mapper的模板去生成mapper代理对象，模板中在最后统一关闭sqlsession。**

### 一级缓存同 spring 整合后的情况

一般在mybatis集成spring时，会把SqlSessionFactory设置为单例注入到IOC容器中，不把sqlsession也设置为单例的原因是sqlsession是线程不安全的，所以不能为单例。那也就意味着其实是有关闭sqlsession的过程的。其实，对于每一个service中的sqlsession是不同的，这是通过mybatis-spring中的org.mybatis.spring.mapper.MapperScannerConfigurer创建sqlsession自动注入到service中的。

而一级缓存的设计是每个sqlsession单独使用一个缓存空间，不同的sqlsession是不能互相访问数据的。当然，在sqlsession关闭后，其中数据自然被清空。

特此警告！！！！

当MyBatis与spring整合后，如果没有事务，一级缓存是失效的！一级缓存是失效的！一级缓存是失效的！

原因就是两者结合后，sqlsession如果发现当前没有事务，那么每执行一个mapper方法，sqlsession就被关闭了。

### 一级缓存测试

## 二级缓存

### 原理

二级缓存的范围是mapper级别（mapper同一个命名空间），mapper以命名空间为单位创建缓存数据结构，结构是map<key、value>。

每次查询先看是否开启二级缓存，如果开启从二级缓存的数据结构中取缓存数据，

如果从二级缓存没有取到，再从一级缓存中找，如果一级缓存也没有，从数据库查询。

### mybatis二级缓存配置

在核心配置文件SqlMapConfig.xml中加入

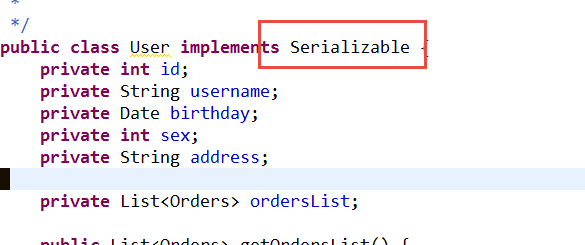
<setting name=*"cacheEnabled"* value=*"true"*/>

|  |  |  |  |
| --- | --- | --- | --- |
|  | 描述 | 允许值 | 默认值 |
| cacheEnabled | 对在此配置文件下的所有cache 进行全局性开/关设置。 | true false | true |

要在你的Mapper映射文件中添加一行：  <cache /> ，表示此mapper开启二级缓存， 位置在 namespace 之后

**<cache eviction="FIFO" flushInterval="60000" size="512" readOnly="true"/>**

### 查询结果映射的pojo序列化



### 二级缓存禁用

对于变化频率较高的sql，需要禁用二级缓存：

在statement中设置useCache=false可以禁用当前select语句的二级缓存，即每次查询都会发出sql去查询，默认情况是true，即该sql使用二级缓存。

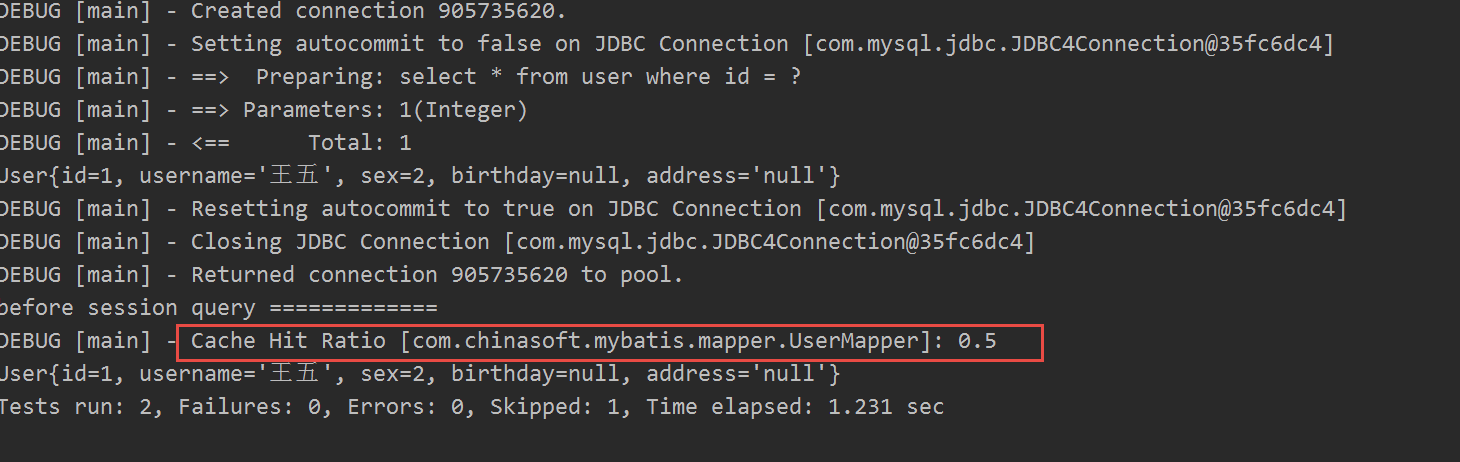
<select id="findOrderListResultMap" resultMap="ordersUserMap" useCache="false">

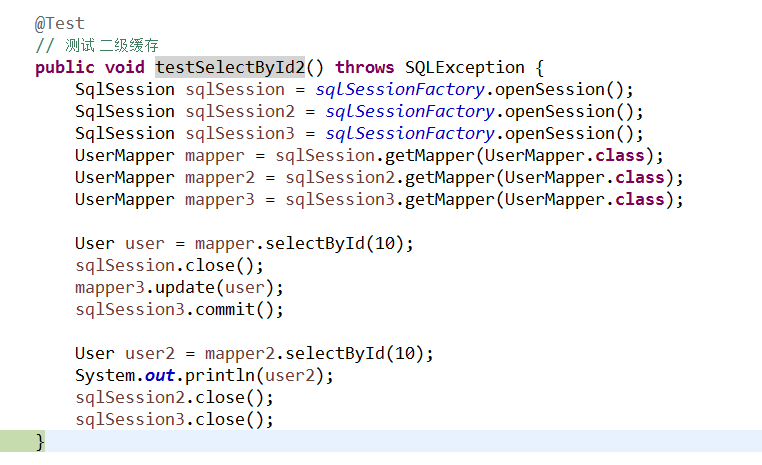
### 刷新缓存（update、delete、 insert）

如果sqlsession操作commit操作，对二级缓存进行刷新（全局清空）。

设置statement的flushCache是否刷新缓存，默认值是true。

### 测试代码





必须先关闭 sqlSession， 才能使用它的缓存 。

### mybatis的cache参数（了解）

**<cache eviction="FIFO" flushInterval="60000" size="512" readOnly="true"/>**

mybatis的cache参数只适用于mybatis维护缓存。

flushInterval（刷新间隔）可以被设置为任意的正整数，而且它们代表一个合理的毫秒形式的时间段。默认情况是不设置，也就是没有刷新间隔，缓存仅仅调用语句时刷新。

size（引用数目）可以被设置为任意正整数，要记住你缓存的对象数目和你运行环境的可用内存资源数目。默认值是1024。

readOnly（只读）属性可以被设置为true或false。只读的缓存会给所有调用者返回缓存对象的相同实例。因此这些对象不能被修改。这提供了很重要的性能优势。可读写的缓存会返回缓存对象的拷贝（通过序列化）。这会慢一些，但是安全，因此默认是false。

如下例子：

<cache  eviction="FIFO"  flushInterval="60000"  size="512"  readOnly="true"/>

这个更高级的配置创建了一个 FIFO 缓存,并每隔 60 秒刷新,存数结果对象或列表的 512 个引用,而且返回的对象被认为是只读的,因此在不同线程中的调用者之间修改它们会导致冲突。可用的收回策略有, 默认的是 LRU:

1. LRU – 最近最少使用的:移除最长时间不被使用的对象。
2. FIFO – 先进先出:按对象进入缓存的顺序来移除它们。
3. SOFT – 软引用:移除基于垃圾回收器状态和软引用规则的对象。
4. WEAK – 弱引用:更积极地移除基于垃圾收集器状态和弱引用规则的对象。

堆 Heap

Stack 堆栈

User:xxxx

Id:

Username:

sex

User2:xxxx

Id:

Username:

sex

Id:

Username:

sex

## mybatis和ehcache缓存框架整合

mybatis二级缓存通过ehcache维护缓存数据。

Redis： memcache，

### 分布缓存

将缓存数据数据进行分布式管理。

系统1

springmvc

mybatis

系统2

springmvc

mybatis

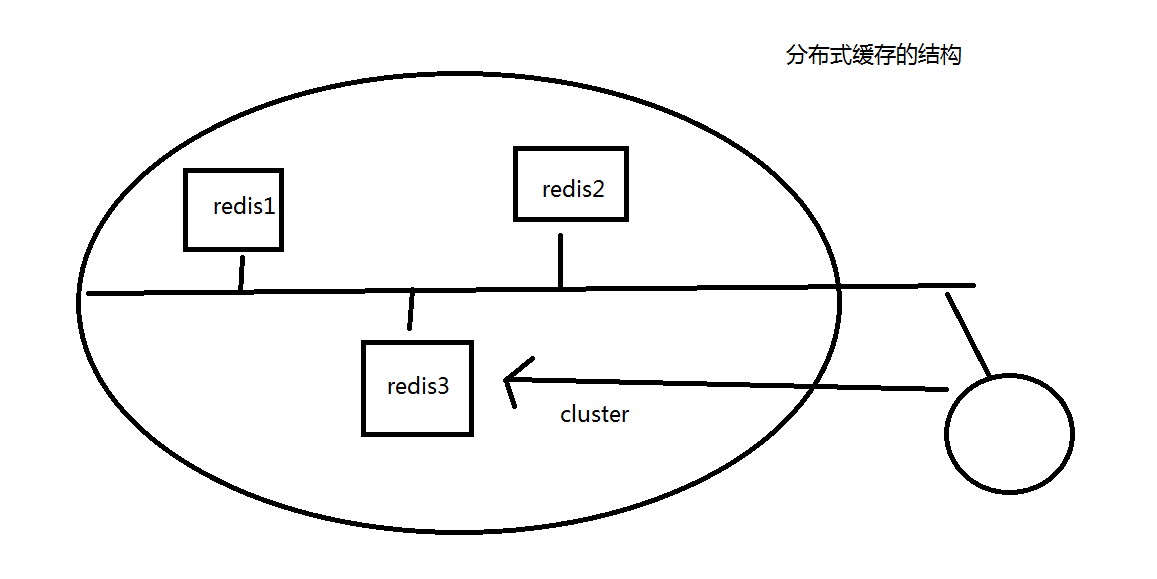
用户请求

缓存框架（远程服务器集群）

redis、memcached、ehcache

存取缓存数据

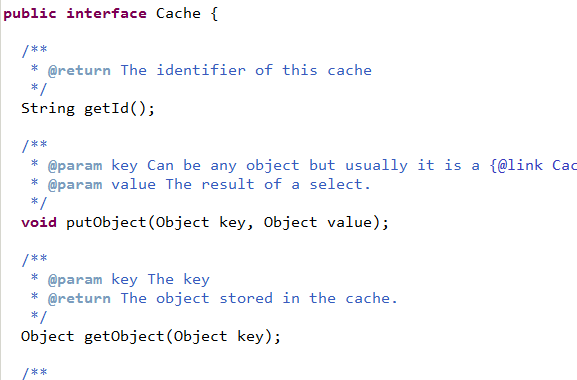
存取缓存数据



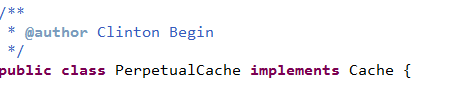
### mybatis和ehcache思路

通过mybatis和ehcache框架进行整合，就可以把缓存数据的管理托管给ehcache。(redis)

在mybatis中提供一个cache接口，只要实现cache接口就可以把缓存数据灵活的管理起来。



mybatis中默认实现：



### 下载和ehcache整合的jar包

在 项目 pom.xml 里面 修改：

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis-ehcache</artifactId>

<version>1.0.0</version>

</dependency>

### 配置ehcache.xml

文件内容， 放到 resources 目录 下面 就可以了。

<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="../config/ehcache.xsd">

<!--diskStore：缓存数据持久化的目录 地址 -->

<diskStore path="D:\develop\ehcache" />

<defaultCache

maxElementsInMemory="1000"

maxElementsOnDisk="10000000"

eternal="false"

overflowToDisk="false"

diskPersistent="true"

timeToIdleSeconds="120"

timeToLiveSeconds="120"

diskExpiryThreadIntervalSeconds="120"

memoryStoreEvictionPolicy="LRU">

</defaultCache>

</ehcache>

### 整合测试Wor27D3

在mapper.xml添加ehcache配置：（ UserMapper.xml ）

<!-- 开启二级缓存 -->

<cache type="org.mybatis.caches.ehcache.EhcacheCache">

<property name="timeToIdleSeconds" value="12000"/>

<property name="timeToLiveSeconds" value="3600"/>

<!-- 同ehcache参数maxElementsInMemory -->

<property name="maxEntriesLocalHeap" value="1000"/>

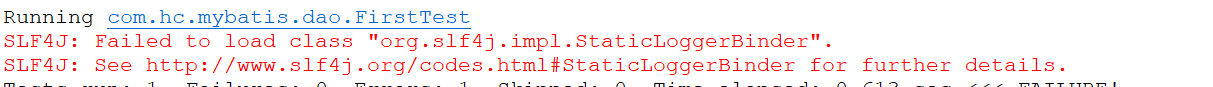
<!-- 同ehcache参数maxElementsOnDisk -->

<property name="maxEntriesLocalDisk" value="10000000"/>

<property name="memoryStoreEvictionPolicy" value="LRU"/>

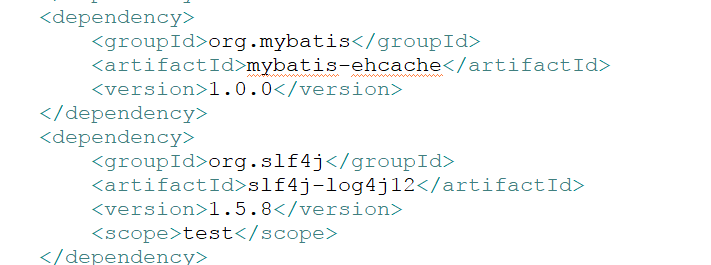
</cache>

### 错误及处理



遇见错误怎么 办，

解决方法， 修改 项目 的 pom.xml



#### Slf4j 是 什么 ？

slf4j 日志的统一接口（ common-loggin ） (jdbc) (interface)

log4j

logback

类

不管怎样，让我们回到这个话题，SLF4J不同于其他日志类库，与其它有很大的不同。SLF4J(Simple logging Facade for Java)不是一个真正的日志实现，而是一个抽象层（ abstraction layer），它允许你在后台使用任意一个日志类库。如果是在编写供内外部都可以使用的API或者通用类库，那么你真不会希望使用你类库的客户端必须使用你选择的日志类库。

总的来说，SLF4J使你的代码独立于任意一个特定的日志API，这是一个对于开发API的开发者很好的思想。虽然抽象日志类库的思想已经不是新鲜的事物而且Apache commons logging也已经在使用这种思想了，但现在SLF4J正迅速成为Java世界的日志标准。让我们再看看几个使用SLF4J而不是log4j、logback或者java.util.logging的理由。

## 二级缓存的应用场景

对查询频率高，变化频率低的数据建议使用二级缓存。

对于访问多的查询请求且用户对查询结果实时性要求不高，此时可采用mybatis二级缓存技术降低数据库访问量，提高访问速度，业务场景比如：耗时较高的统计分析sql、电话账单查询sql等。

实现方法如下：通过设置刷新间隔时间，由mybatis每隔一段时间自动清空缓存，根据数据变化频率设置缓存刷新间隔flushInterval，比如设置为30分钟、60分钟、24小时等，根据需求而定。

## mybatis局限性

mybatis二级缓存对细粒度的数据级别的缓存实现不好，比如如下需求：对商品信息进行缓存，由于商品信息查询访问量大，但是要求用户每次都能查询最新的商品信息，此时如果使用mybatis的二级缓存就无法实现当一个商品变化时只刷新该商品的缓存信息而不刷新其它商品的信息，因为mybaits的二级缓存区域以mapper为单位划分，当一个商品信息变化会将所有商品信息的缓存数据全部清空。解决此类问题需要在业务层根据需求对数据有针对性缓存。（用的很少）

# mybatis和spring整合

## mybaits和spring整合的思路

见 xmind 文件， 主要就是 夺权 ！

## 创建整合工程

## 加入jar包

修改 pom.xml

<dependencies>

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis</artifactId>

<version>${mybatis.version}</version>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>1.2.17</version>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.38</version>

</dependency>

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis-ehcache</artifactId>

<version>1.0.0</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>1.5.8</version>

</dependency>

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis-spring</artifactId>

<version>1.3.2</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>4.3.16.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-test</artifactId>

<version>4.3.16.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-orm</artifactId>

<version>4.3.16.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aspects</artifactId>

<version>4.3.16.RELEASE</version>

</dependency>

<!-- <dependency> <groupId>commons-dbcp</groupId> <artifactId>commons-dbcp</artifactId>

<version>1.4</version> </dependency> -->

<dependency>

<groupId>c3p0</groupId>

<artifactId>c3p0</artifactId>

<version>0.9.1.2</version>

</dependency>

Parent/pom.xml 的情况， 会 发给大家

## log4j.properties

log4j.appender.Console.layout.ConversionPattern=%d{mm:ss} [%t] %-5p [%.30c] - %m%n

上面的位置 进行了 修改：

log4j.rootLogger=DEBUG, Console

#Console

log4j.appender.Console=org.apache.log4j.ConsoleAppender

log4j.appender.Console.layout=org.apache.log4j.PatternLayout

#log4j.appender.Console.layout.ConversionPattern=%d [%t] %-5p [%c] - %m%n

log4j.appender.Console.layout.ConversionPattern=%d{mm:ss} [%t] %-5p [%.30c] - %m%n

log4j.logger.java.sql.ResultSet=INFO

log4j.logger.org.apache=INFO

log4j.logger.java.sql.Connection=DEBUG

log4j.logger.java.sql.Statement=DEBUG

log4j.logger.java.sql.PreparedStatement=DEBUG

## SqlMapconfig.xml

mybatis配置文件：别名、settings，数据源不在这里配置、

<!DOCTYPE configuration

PUBLIC "-//mybatis.org//DTD Config 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>

<!-- 读取配置文件 -->

<settings>

<!-- 延迟加载总开关 -->

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

<!-- 设置按需加载 -->

<setting name=*"aggressiveLazyLoading"* value=*"false"* />

<!-- 打开 二级缓存 -->

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

</settings>

<typeAliases>

<!-- 别名的配置 -->

<!-- <typeAlias type="com.hc.mybatis.po.User" alias="user"/> -->

<package name=*"com.hc.mybatis.po"*/>

</typeAliases>

<!-- 和spring整合后 environments配置将废除 -->

<mappers>

<!-- <mapper resource="sqlmap/User.xml"/> -->

<!-- <mapper resource="com/hc/mybatis/mapper/UserMapper.xml" /> -->

<!-- <mapper class="com.hc.mybatis.mapper.UserMapper"/> -->

</mappers>

</configuration>

## applicationContext.xml

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns="http://www.springframework.org/schema/beans" xmlns:aop="http://www.springframework.org/schema/aop"

xmlns:context="http://www.springframework.org/schema/context" xmlns:tx="http://www.springframework.org/schema/tx"

xmlns:cache="http://www.springframework.org/schema/cache" xmlns:p="http://www.springframework.org/schema/p"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/aop

http://www.springframework.org/schema/aop/spring-aop.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd

http://www.springframework.org/schema/tx

http://www.springframework.org/schema/tx/spring-tx.xsd

http://www.springframework.org/schema/cache http://www.springframework.org/schema/cache/spring-cache.xsd">

<!-- 加载配置文件 -->

<context:property-placeholder location="classpath:db.properties" />

<!-- 数据库连接池 -->

<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"

destroy-method="close">

<property name="driverClassName" value="${jdbc.driver}" />

<property name="url" value="${jdbc.url}" />

<property name="username" value="${jdbc.username}" />

<property name="password" value="${jdbc.password}" />

<property name="maxActive" value="10" />

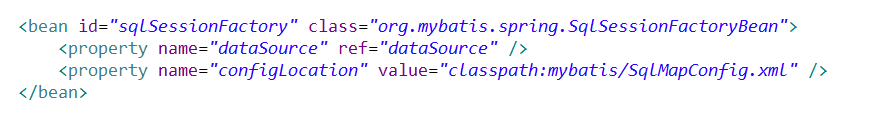
<property name="maxIdle" value="5" />

</bean>

</beans>

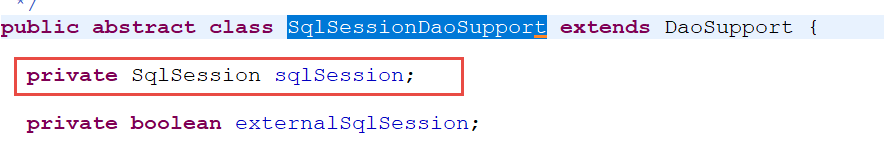
## 整合开发原始dao接口

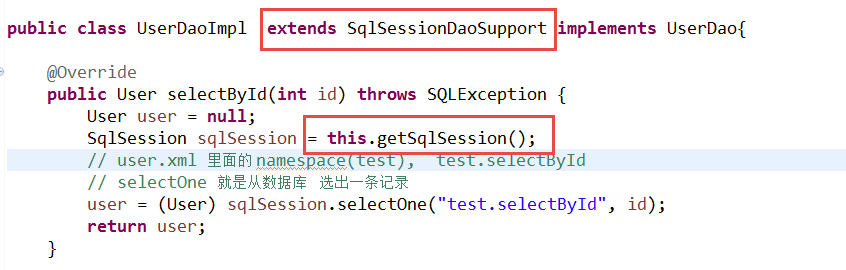
### 配置SqlSessionFactory



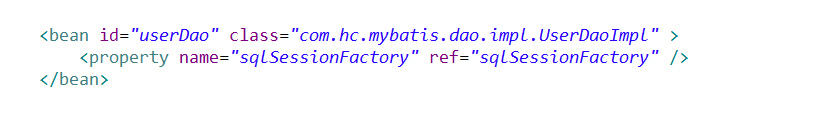
在 applicationContext.xml配置SqlSessionFactory

### 开发dao





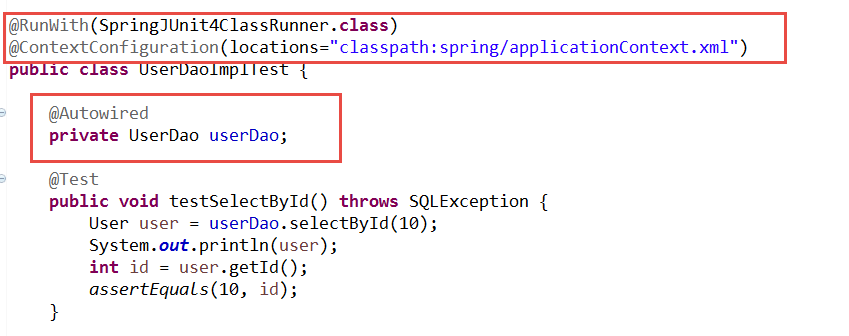
### 配置 dao



### 测试dao接口

通过注解的方式从 applicationContext.xml 到 applicationContext 的 生成，

userDao 通过注解的方式 注入

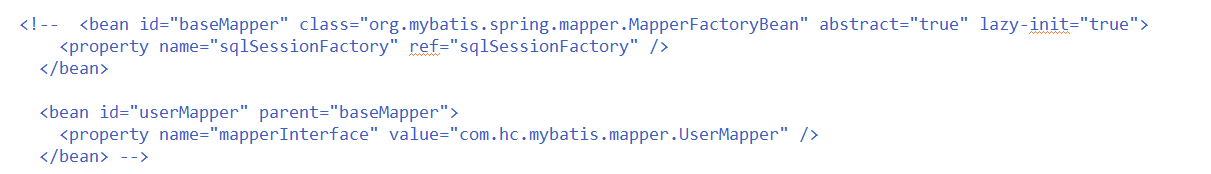


## 整合开发mapper代理方法

### 开发mapper.xml和mapper.java

复制以前的项目

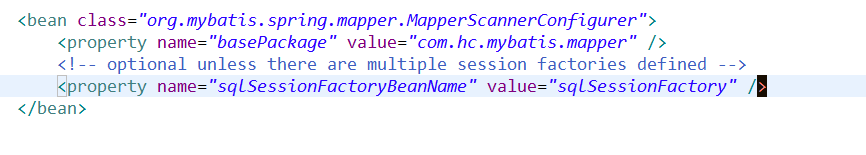
### 使用MapperFactoryBean

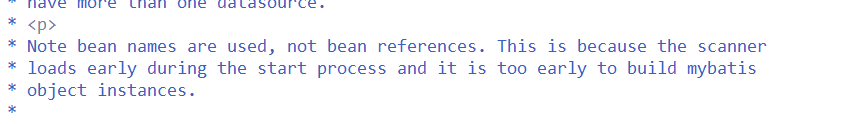


使用此方法对于每个mapper都需要配置，比较繁琐。

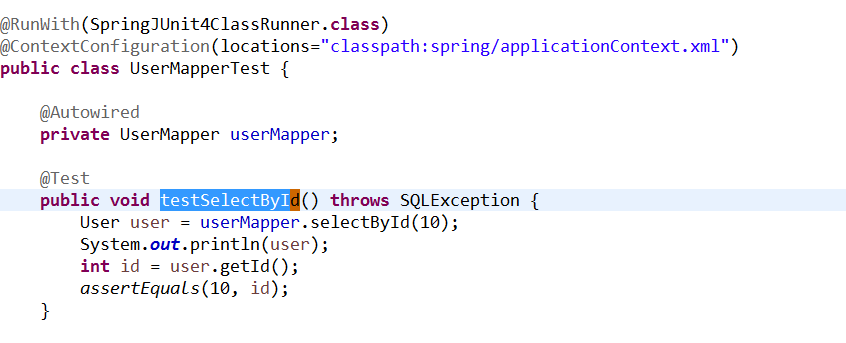
### 使用MapperScannerConfigurer（扫描mapper）

使用扫描器自动扫描mapper，生成代理对象，比较方便。





### 测试mapper接口



## 替换 数据库连接池

### 引包修改pom.xml

<dependency>

<groupId>com.mchange</groupId>

<artifactId>c3p0</artifactId>

<version>0.9.5.2</version>

</dependency>

### 修改 applicationContext.xml

<context:property-placeholder location="classpath:jdbc.properties"/>

<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">

<property name="driverClass" value="${jdbc.driver}"/>

<property name="jdbcUrl" value="${jdbc.url}"/>

<property name="user" value="${jdbc.username}"/>

<property name="password" value="${jdbc.password}"/>

</bean>

# mybatis逆向工程 （MyBatis Generator）

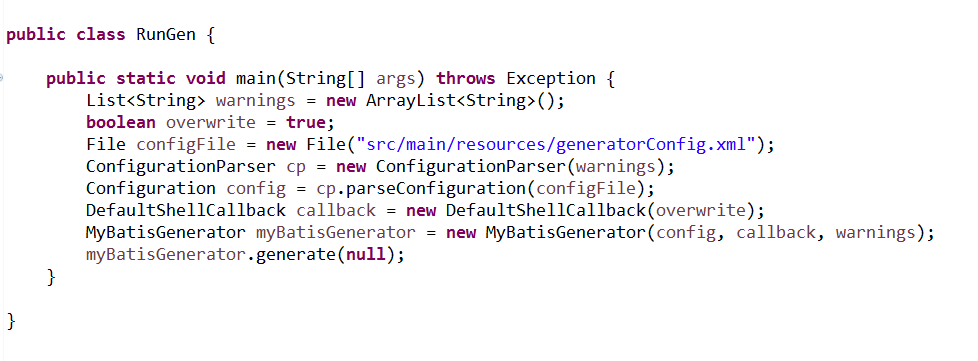
## 什么是mybatis的逆向工程

我们正常开发， 就是 根据 数据库的设计 来进行 dao 层的开发 。 就是根据 数据库的 表设计（表） 来自动生成 持久层的代码（mapper）， mapper 的 interface 和 mapper的 xml ， 还有 pojo 类（ 数据库表的pojo类）。搞定后， 持久层的代码 完成 。 但是， 它只能满足 一些简单需求 。如果是 复杂需求还是需要我们自己进行mapper的 sql语句的开发和编写。

## 逆向工程 使用配置

运行逆向工程 方法：

本教程使用通过java程序运行逆向工程 ， 使用 xml 进行工程的配置。



逆向工程 运行所需要的jar包：

<dependency>

<groupId>org.mybatis.generator</groupId>

<artifactId>mybatis-generator-core</artifactId>

<version>1.3.5</version>

</dependency>

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis</artifactId>

</dependency>

数据库驱动包

相应的 版本控制 见 parent/pom.xml

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<argLine>-Dfile.encoding=UTF-8</argLine>

<mybatis.version>3.4.5</mybatis.version>

</properties>

<dependencies>

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis</artifactId>

<version>${mybatis.version}</version>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>1.2.17</version>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.38</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>1.5.8</version>

</dependency>

<dependency>

<groupId>org.mybatis.generator</groupId>

<artifactId>mybatis-generator-core</artifactId>

<version>1.3.5</version>

</dependency>

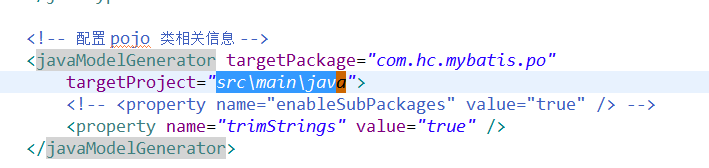
### xml配置

需要使用用配置的地方：GeneratorConfig.xml

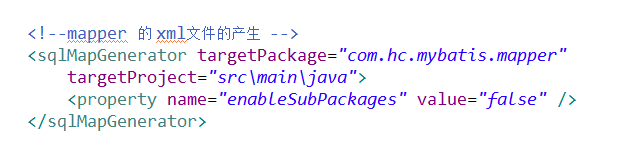
1、连接数据库的地址和驱动



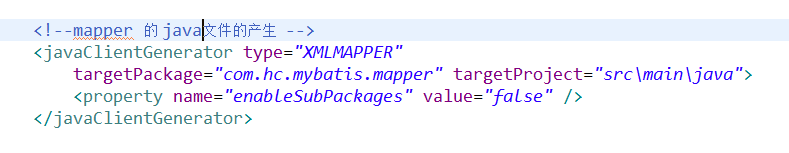
2、需要配置po类的包路径



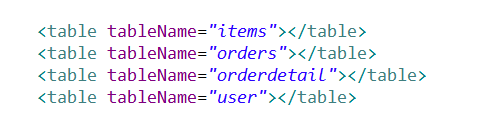
3、需要配置mapper包的路径  
 mapper 的 xml 文件



interface 文件：



4、指定数据表



文件的文本内容

<?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?useSSL=false" userId="peter"

password="peter">

</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.hc.ssm.po"

targetProject=".\src\main\java">

<!-- enableSubPackages:是否让schema作为包的后缀 -->

<property name="enableSubPackages" value="false" />

<!-- <property name="constructorBased" value="true" /> -->

<!-- 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.hc.ssm.mapper"

targetProject=".\src\main\java">

<!-- 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.hc.ssm.mapper"

targetProject=".\src\main\java">

<!-- 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>

<!-- 有些表的字段需要指定java类型

<table schema="" tableName="">

<columnOverride column="" javaType="" />

</table> -->

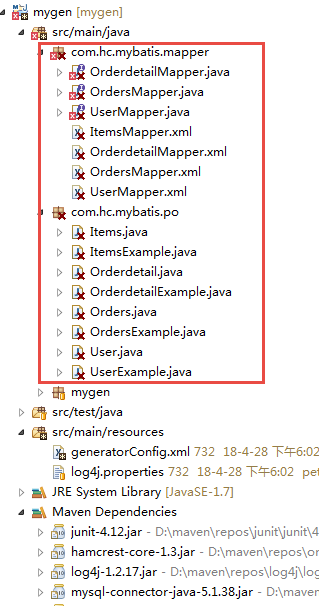
</context>

</generatorConfiguration>

### java程序

http://www.mybatis.org/generator/running/runningWithJava.html

通过java程序生成mapper类、po类。。



## 使用逆向工程生成的代码

### 第一步配置generatorConfig.xml

参考：见上面， 包的设置， 必须同 使用代码的工程 。

### 第二步配置执行java程序

每次执行java程序前：必须删除 上次产生的代码（包， mapper、po包），执行java程序后，所生成的代码已经生成到 当前的工程中。

### 第三步将生成的代码拷贝到使用代码的工程 中

## 测试生成的代码



参考：