# JavaEE 之 MyBatis 技术栈

**讲师:刘峰吉**

# 一、MyBatis 概述

## 1、MyBatis 学习目标

1）MyBatis简介

2）MyBatis-HelloWorld

3）MyBatis-全局配置文件

4）MyBatis-映射文件

5）MyBatis-动态SQL

6）MyBatis-缓存机制

7）MyBatis-Spring整合

8）MyBatis-逆向工程

## 2、MyBatis 简介

1）MyBatis 是支持定制化 SQL、存储过程以及高级映射的优秀的持久层框架。

2）MyBatis 避免了几乎所有的 JDBC 代码和手动设置参数以及获取结果集。

3）MyBatis可以使用简单的XML或注解用于配置和原始映射，将接口和Java的POJO（Plain Old Java Objects，普通的Java对象）映射成数据库中的记录.

## 3、MyBatis 历史

1）原是Apache的一个开源项目iBatis, 2010年6月这个项目由Apache Software Foundation 迁移到了Google Code，随着开发团队转投Google Code旗下， iBatis3.x正式更名为MyBatis ，代码于2013年11月迁移到Github（下载地址见后）。

2）iBatis一词来源于“internet”和“abatis”的组合，是一个基于Java的持久层框架。 iBatis提供的持久层框架包括SQL Maps和Data Access Objects（DAO）

## 4、为什么要使用MyBatis？

1）MyBatis是一个半自动化的持久化层框架。

2）JDBC

（1）SQL夹在Java代码块里，耦合度高导致硬编码内伤

（2）维护不易且实际开发需求中sql是有变化，频繁修改的情况多见

3）Hibernate和JPA

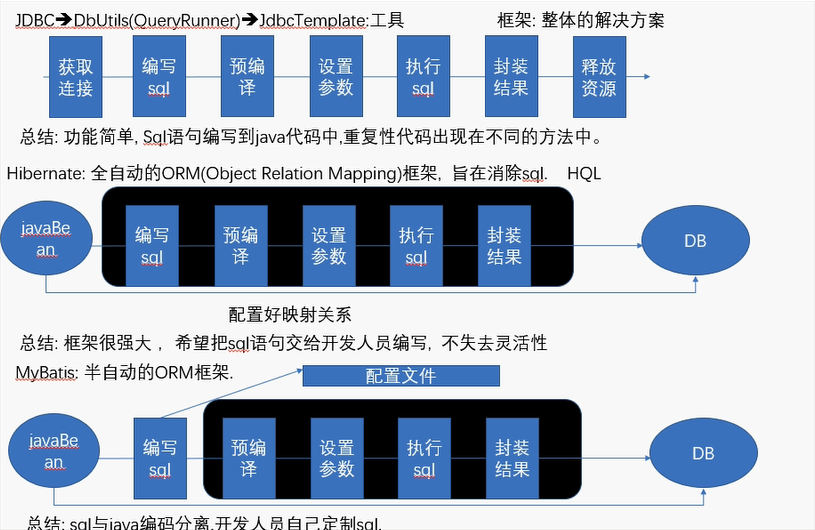
（1）长难复杂SQL，对于Hibernate而言处理也不容易

（2）内部自动生产的SQL，不容易做特殊优化。

（3）基于全映射的全自动框架，大量字段的POJO进行部分映射时比较困难。导致数据库性能下降。

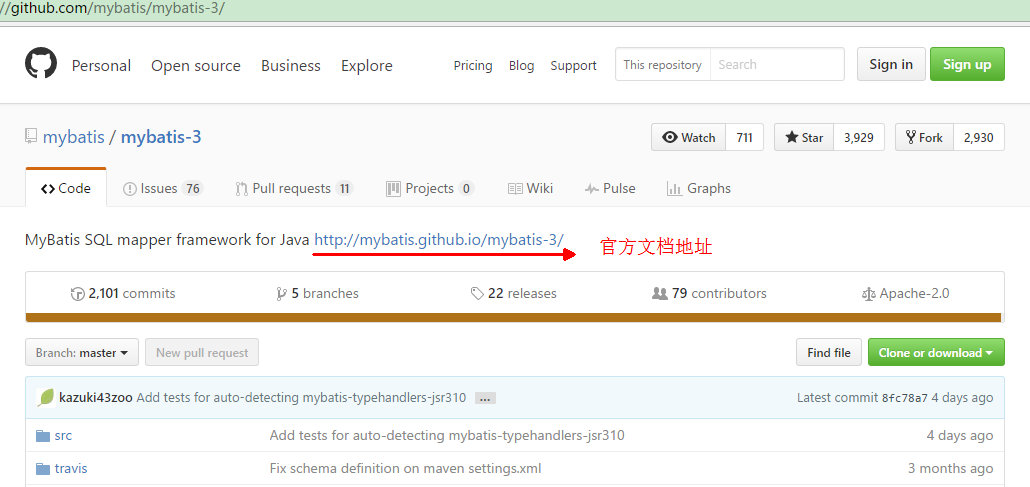
4）对开发人员而言，核心sql还是需要自己优化

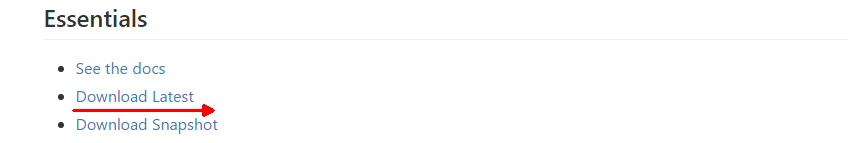
5）sql和java编码分开，功能边界清晰，一个专注业务、一个专注数据。

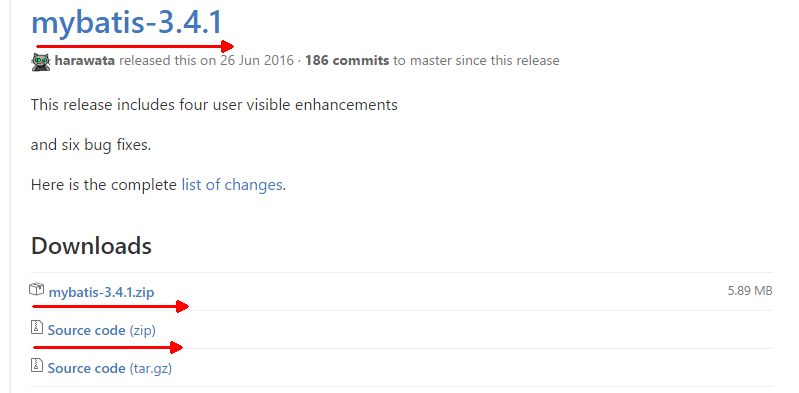


## 5、去哪里找MyBatis？

https://github.com/mybatis/mybatis-3/







# 二、MyBatis 之 HelloWorld（简单版）MyBatis01\_HelloWorld

## 1、MyBatis HelloWord 思想

1）创建一个javaProject

2）MyBatis框架的jar包导入

Mybatis框架的基本操作只需要mybatis-3.4.1.jar

加入log4j.jar的目的是在MyBatis操作的关键步骤输出一些日志信息。

3）准备测试表

tbl\_employee

4）编写java类

Employee.java

5）编写mybatis的全局配置文件，sql映射文件

mybatis的全局配置文件: mybatis-config.xml

sql映射文件: xxxxMapper.xml

6）编码测试

（1）通过mybatis-config.xml创建SqlSessionFactory对象.

（2）通过SqlSessionFactory创建SqlSession对象，SqlSession对象是与数据库的会话对象。

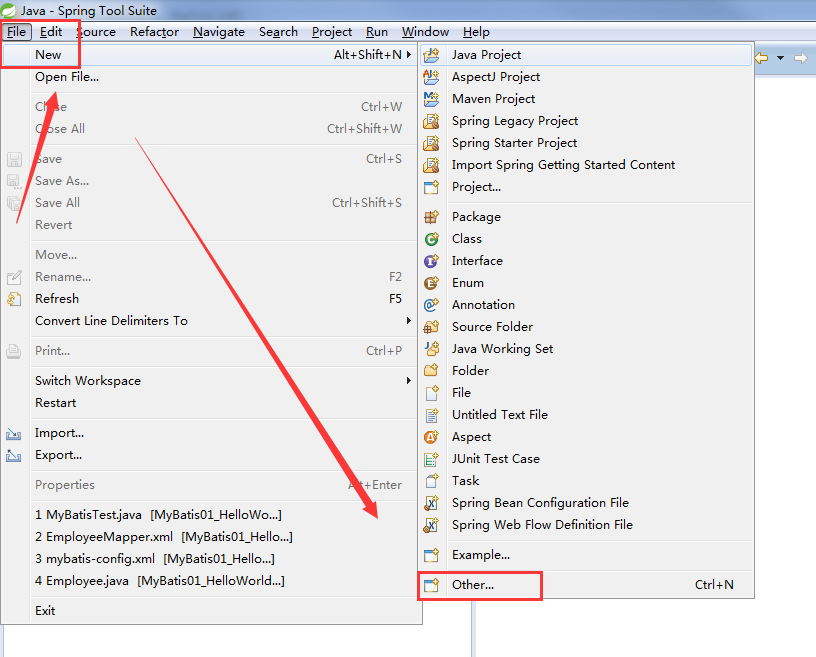
可以简单理解为Connection对象.通过SqlSession来完成CRUD的操作

（3）调用SqlSession的API方法， 指定上Sql语句的唯一标识以及传递给sql语句的参数.

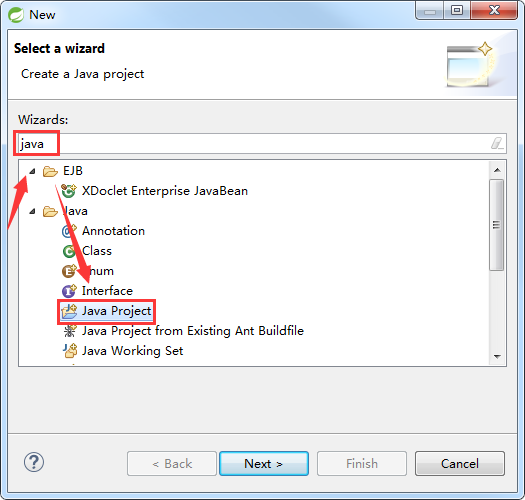
（4）关闭SqlSession对象，释放资源.

## 2、创建Java project

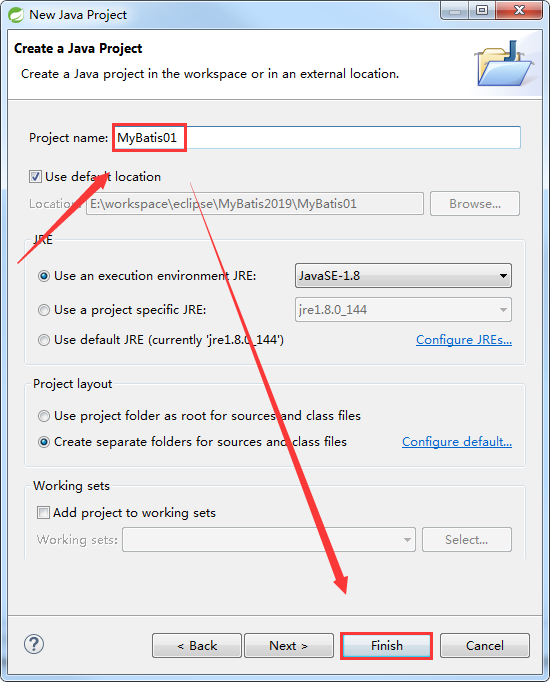
### 1）File -> New –> Other



### 2）java -> Java Project -> Next



### 3）Project name -> Finish



## 3、添加Jar包

1）创建lib文件夹

2）添加jar包

|  |
| --- |
| log4j.jar  mybatis-3.4.1.jar  mysql-connector-java-5.1.37-bin.jar |

3）build path

## 4、编写代码

### 1）Employee

|  |
| --- |
| package com.alex.mybatis.entity;  public class Employee {    private Integer id;  private String lastName;  private String email;  private String gender;    public Employee() {    }  public Employee(Integer id, String lastName, String email, String gender) {  this.id = id;  this.lastName = lastName;  this.email = email;  this.gender = gender;  }  public Integer getId() {  return id;  }  public void setId(Integer id) {  this.id = id;  }  public String getLastName() {  return lastName;  }  public void setLastName(String lastName) {  this.lastName = lastName;  }  public String getEmail() {  return email;  }  public void setEmail(String email) {  this.email = email;  }  public String getGender() {  return gender;  }  public void setGender(String gender) {  this.gender = gender;  }  @Override  public String toString() {  return "Employee [id=" + id + ", lastName=" + lastName + ", email=" + email + ", gender=" + gender + "]";  }  } |

### 2）MyBatisTest

|  |
| --- |
| package com.alex.mybatis.test;  import java.io.InputStream;  import org.apache.ibatis.io.Resources;  import org.apache.ibatis.session.SqlSession;  import org.apache.ibatis.session.SqlSessionFactory;  import org.apache.ibatis.session.SqlSessionFactoryBuilder;  import org.junit.Test;  import com.alex.mybatis.entity.Employee;  public class MyBatisTest {    @Test  public void TestGetSqlSessionFactory() throws Exception {  String resource = "mybatis-config.xml";  InputStream inputStream = Resources.getResourceAsStream(resource);  SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);  System.out.println(sqlSessionFactory);  }  public SqlSessionFactory getSqlSessionFactory() throws Exception {  String resource = "mybatis-config.xml";  InputStream inputStream = Resources.getResourceAsStream(resource);  SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);  System.out.println(sqlSessionFactory);  return sqlSessionFactory;  }  @Test  public void testHelloWorldSimple() throws Exception {  // 1.获取SqlSessionFactory  SqlSessionFactory ssf = getSqlSessionFactory();  // 2.获取SqlSession对象  SqlSession session = ssf.openSession();  try {  /\*\*  \* selectOne的两个参数:  \* 1. sql语句的唯一标识 （namespace + id）  \* 2. 执行sql的参数  \*/  Employee employee = session.selectOne("com.alex.mybatis.selectEmployee", 1001);  System.out.println(employee);  } finally {  session.close();  }  }  } |

## 5、添加配置文件

### 1）log4j.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">    <log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">    <appender name="STDOUT" class="org.apache.log4j.ConsoleAppender">  <param name="Encoding" value="UTF-8" />  <layout class="org.apache.log4j.PatternLayout">  <param name="ConversionPattern" value="%-5p %d{MM-dd HH:mm:ss,SSS} %m (%F:%L) \n" />  </layout>  </appender>  <logger name="java.sql">  <level value="debug" />  </logger>  <logger name="org.apache.ibatis">  <level value="info" />  </logger>  <root>  <level value="debug" />  <appender-ref ref="STDOUT" />  </root>  </log4j:configuration> |

### 2）mybatis-config.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE configuration  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-config.dtd">  <!-- MyBatis的全局配置文件 -->  <configuration>  <environments default="development">  <environment id="development">  <transactionManager type="JDBC" />  <dataSource type="POOLED">  <property name="driver" value="com.mysql.jdbc.Driver" />  <property name="url" value="jdbc:mysql://localhost:3306/mybatis" />  <property name="username" value="root" />  <property name="password" value="root" />  </dataSource>  </environment>  </environments>  <mappers>  <mapper resource="EmployeeMapper.xml" />  </mappers>  </configuration> |

### 3）EmployeeMapper.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <!-- Employee的sql映射文件 -->  <!--  namespace: 名称空间.  如果不使用Mapper接口开发，则namesapace可以随便写  如果使用Mapper接口开发，则nameSpace不能随便写.写成Mapper接口的全类名.  -->  <mapper namespace="com.alex.mybatis">  <!--  select: 定义查询的sql语句  id: sql语句的唯一标识  resultType: 定义封装的结果类型。  #{id}: 获取参数  -->  <select id="selectEmployee" resultType="com.alex.mybatis.entity.Employee">  <!-- select \* from tbl\_employee where id = #{id} -->  select id,last\_name lastName,email,gender from tbl\_employee where id = #{id}  </select>  </mapper> |

## 6、添加数据库数据

### 1）mybatis.sql

|  |
| --- |
| /\*  Navicat MySQL Data Transfer  Source Server : localhost  Source Server Version : 50515  Source Host : localhost:3306  Source Database : mybatis  Target Server Type : MYSQL  Target Server Version : 50515  File Encoding : 65001  Date: 2019-10-12 10:49:14  \*/  -- 创建一个数据库，并制定数据库的编码格式  CREATE DATABASE mybatis DEFAULT CHARACTER SET utf8;  SET FOREIGN\_KEY\_CHECKS=0;  -- 使用数据库  USE mybatis;  -- 创建表  -- ----------------------------  -- Table structure for tbl\_employee  -- ----------------------------  DROP TABLE IF EXISTS `tbl\_employee`;  CREATE TABLE `tbl\_employee` (  `id` int(11) NOT NULL AUTO\_INCREMENT,  `last\_name` varchar(50) DEFAULT NULL,  `email` varchar(50) DEFAULT NULL,  `gender` char(1) DEFAULT NULL,  PRIMARY KEY (`id`)  ) ENGINE=InnoDB AUTO\_INCREMENT=1002 DEFAULT CHARSET=utf8;  -- 插入测试数据  -- ----------------------------  -- Records of tbl\_employee  -- ----------------------------  INSERT INTO `tbl\_employee` VALUES ('1001', 'Tom', 'Tom@aliyun.com', '1'); |

# 三、MyBatis 之 HelloWorld（接口版）MyBatis02\_HelloWorld

## 1、MyBatis HelloWord 思想

1）Mapper接口: Dao层的接口.定义了具体的增删改查的方法.

在Mybatis中只需要写好Mapper接口,不需要做具体的实现.因为MyBatis会自动生成代理的实现类，使用代理实现类对象去完成具体的增删改查

2）Mapper接口开发的两个绑定

（1）Mapper接口与Sql映射文件的绑定: sql映射文件的namespace的值必须指定成接口的全类名

（2）Mapper接口方法与sql语句的绑定: sql语句的唯一标识(id)必须指定成方法名.

3）Mapper接口的好处

（1）方法的定义有明确的类型约束(方法的参数/方法的返回值)

（2） 接口: 接口本身就是抽象,抽出了规范. 不要求具体的实现细节. 将定义与实现分离。

EmployeeMapper == > MyBatis的代理实现类 JDBC Hibernate

4）其它基础步骤和简单版的HelloWorld 一样

## 2、核心代码

### 1）EmployeeMapper

|  |
| --- |
| package com.alex.mybatis.mapper;  import com.alex.mybatis.entity.Employee;  public interface EmployeeMapper {  // 根据id查询Employe对象  public Employee getEmpById(Integer id);  } |

### 2）MyBatisTest

|  |
| --- |
| package com.alex.mybatis.test;  import java.io.InputStream;  import org.apache.ibatis.io.Resources;  import org.apache.ibatis.session.SqlSession;  import org.apache.ibatis.session.SqlSessionFactory;  import org.apache.ibatis.session.SqlSessionFactoryBuilder;  import org.junit.Test;  import com.alex.mybatis.entity.Employee;  import com.alex.mybatis.mapper.EmployeeMapper;  public class MyBatisTest {  public SqlSessionFactory getSqlSessionFactory() throws Exception {  String resource = "mybatis-config.xml";  InputStream inputStream = Resources.getResourceAsStream(resource);  SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);  System.out.println(sqlSessionFactory);  return sqlSessionFactory;  }  @Test  public void testHelloWorldMapper() throws Exception {  //1.获取SqlSessionFactory对象  SqlSessionFactory ssf = getSqlSessionFactory();  //2.获取SqlSession对象  SqlSession session = ssf.openSession();  try {  //EmployeeMapper 接口中的 getEmpById(Integer id)  //获取Mapper接口的代理实现类对象  EmployeeMapper mapper = session.getMapper(EmployeeMapper.class);  System.out.println(mapper.getClass().getName());  Employee employee = mapper.getEmpById(1001);  System.out.println(employee);  }finally {  session.close(); // 关闭session  }  }  } |

### 3）EmployeeMapper.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <!-- Employee的sql映射文件 -->  <!--  namespace: 名称空间.  如果不使用Mapper接口开发，则namesapace可以随便写  如果使用Mapper接口开发，则nameSpace不能随便写.写成Mapper接口的全类名.  -->  <mapper namespace="com.alex.mybatis.mapper.EmployeeMapper">  <!--  select: 定义查询的sql语句  id: sql语句的唯一标识  resultType: 定义封装的结果类型。  #{id}: 获取参数  -->  <!-- public Employee getEmpById(Integer id ); -->  <select id="getEmpById" resultType="com.alex.mybatis.entity.Employee">  select id, last\_name lastName ,email ,gender from tbl\_employee where id = #{id}  </select>  </mapper> |

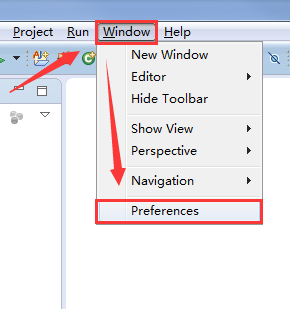
# 四、MyBatis DTD 文件的配置

## 1、DTD: 文档类型定义

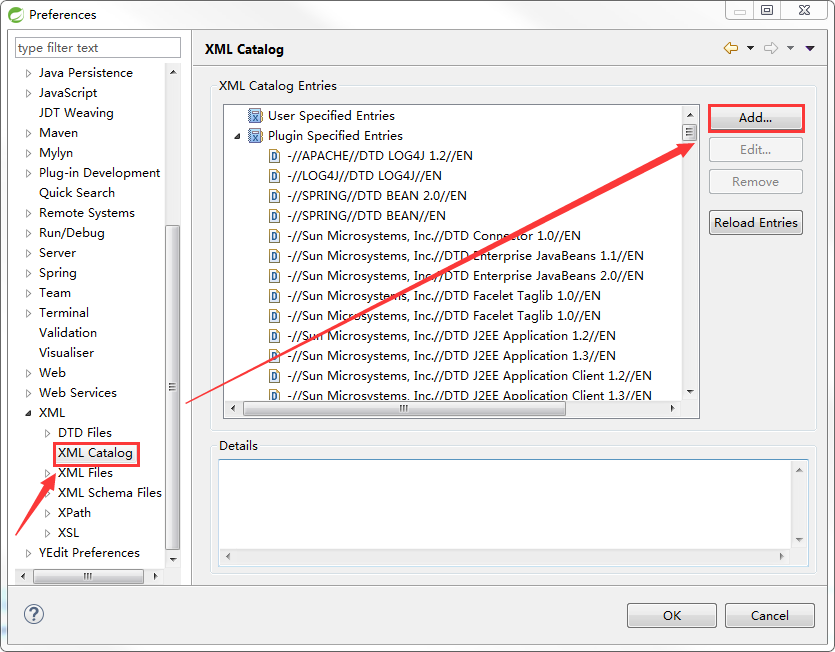
主要是用来约束xml中可以写哪些标签以及各个子标签，以及标签中属性. 以及标签与标签的前后顺序等.

## 2、Eclipse中配置DTD实操

### 1）window -> Preferences

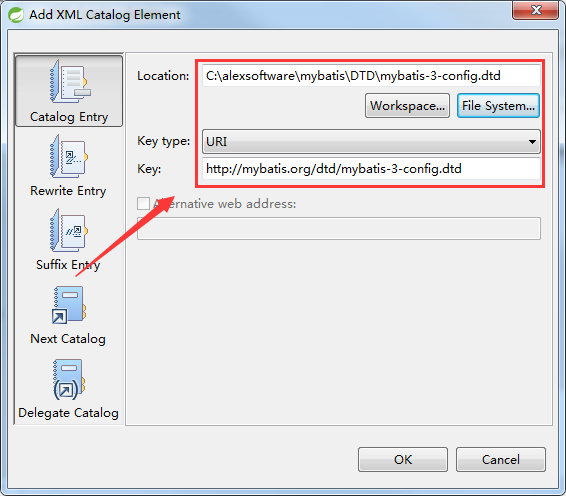


### 2）XML -> XML Catalog -> Add



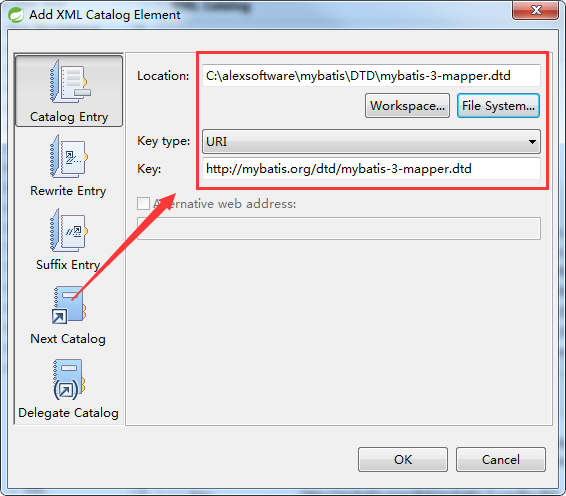
### 3）配置mybatis-3-config.dtd

Key: http://mybatis.org/dtd/mybatis-3-config.dtd



### 4）配置mybatis-3-mapper.dtd

Key：http://mybatis.org/dtd/mybatis-3-mapper.dtd



# 五、MyBatis 全局配置文件 mybatis-config.xml

## 1、properties（MyBatis03\_Properties）

引入外部化的配置文件

### 1）图一



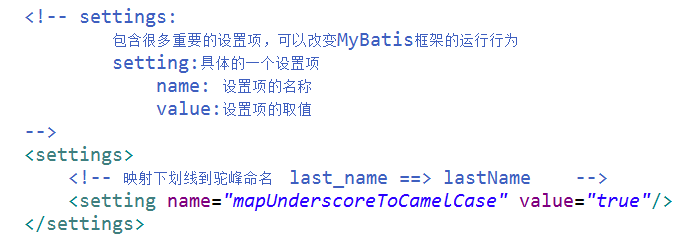
### 2）mybatis-config.xml 核心代码

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE configuration  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-config.dtd">  <!-- MyBatis的全局配置文件 -->  <configuration>  <!-- properties:  引入外部化的配置文件  resource: 加载类路径下的资源文件  url: 加载网络路径或者是磁盘路径下的资源文件  -->  <properties resource="db.properties" ></properties>    <environments default="development">  <environment id="development">  <transactionManager type="JDBC" />  <dataSource type="POOLED">  <property name="driver" value="${jdbc.driver}" />  <property name="url" value="${jdbc.url}" />  <property name="username" value="${jdbc.username}" />  <property name="password" value="${jdbc.password}" />  </dataSource>  </environment>  </environments>    <mappers>  <mapper resource="EmployeeMapper.xml" />  </mappers>    </configuration> |

## 2、settings（MyBatis04\_Settings）

包含很多重要的设置项，可以改变Mybatis的运行行为

### 1）图一



### 2）mybatis-config.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE configuration  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-config.dtd">  <!-- MyBatis的全局配置文件 -->  <configuration>  <!-- settings:  包含很多重要的设置项，可以改变MyBatis框架的运行行为  setting:具体的一个设置项  name: 设置项的名称  value:设置项的取值  -->  <settings>  <!-- 映射下划线到驼峰命名last\_name ==> lastName -->  <setting name="mapUnderscoreToCamelCase" value="true"/>  </settings>    <environments default="development">  <environment id="development">  <transactionManager type="JDBC" />  <dataSource type="POOLED">  <property name="driver" value="com.mysql.jdbc.Driver" />  <property name="url" value="jdbc:mysql://localhost:3306/mybatis" />  <property name="username" value="root" />  <property name="password" value="root" />  </dataSource>  </environment>  </environments>    <mappers>  <mapper resource="EmployeeMapper.xml" />  </mappers>    </configuration> |

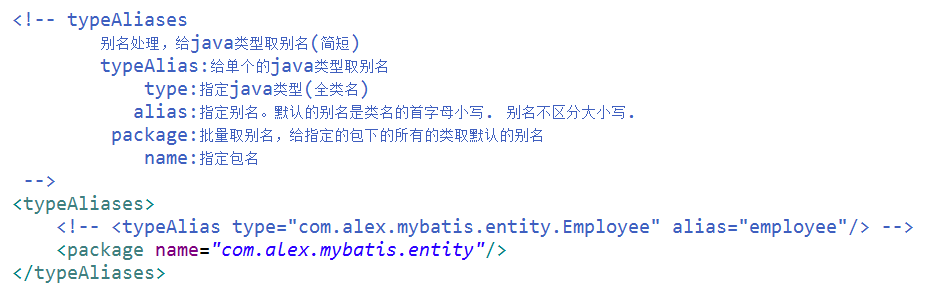
### 3）EmployeeMapper.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <!-- Employee的sql映射文件 -->  <!--  namespace: 名称空间.  如果不使用Mapper接口开发，则namesapace可以随便写  如果使用Mapper接口开发，则nameSpace不能随便写.写成Mapper接口的全类名.  -->  <mapper namespace="com.alex.mybatis.mapper.EmployeeMapper">  <!--  select: 定义查询的sql语句  id: sql语句的唯一标识  resultType: 定义封装的结果类型。  #{id}: 获取参数  -->  <!-- public Employee getEmpById(Integer id ); -->  <select id="getEmpById" resultType="com.alex.mybatis.entity.Employee">  <!--  开启运行时，驼峰命名，就可以不用起别名了  select id, last\_name lastName ,email ,gender from tbl\_employee where id = #{id}  -->  select \* from tbl\_employee where id = #{id}  </select>  </mapper> |

## 3、typeAliases（MyBatis05\_TypeAliases）

别名处理，给java类取别名

### 1）图一



### 2）mybatis-config.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE configuration  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-config.dtd">  <!-- MyBatis的全局配置文件 -->  <configuration>  <!-- typeAliases  别名处理，给java类型取别名(简短)  typeAlias:给单个的java类型取别名  type:指定java类型(全类名)  alias:指定别名。默认的别名是类名的首字母小写. 别名不区分大小写.  package:批量取别名，给指定的包下的所有的类取默认的别名  name:指定包名  -->  <typeAliases>  <!-- <typeAlias type="com.alex.mybatis.entity.Employee" alias="employee"/> -->  <package name="com.alex.mybatis.entity"/>  </typeAliases>    <environments default="development">  <environment id="development">  <transactionManager type="JDBC" />  <dataSource type="POOLED">  <property name="driver" value="com.mysql.jdbc.Driver" />  <property name="url" value="jdbc:mysql://localhost:3306/mybatis" />  <property name="username" value="root" />  <property name="password" value="root" />  </dataSource>  </environment>  </environments>    <mappers>  <mapper resource="EmployeeMapper.xml" />  </mappers>    </configuration> |

### 3）EmployeeMapper.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <!-- Employee的sql映射文件 -->  <!--  namespace: 名称空间.  如果不使用Mapper接口开发，则namesapace可以随便写  如果使用Mapper接口开发，则nameSpace不能随便写.写成Mapper接口的全类名.  -->  <mapper namespace="com.alex.mybatis.mapper.EmployeeMapper">  <!--  select: 定义查询的sql语句  id: sql语句的唯一标识  resultType: 定义封装的结果类型。  #{id}: 获取参数  -->  <!-- public Employee getEmpById(Integer id ); -->  <select id="getEmpById" resultType="employee">  select id, last\_name lastName ,email ,gender from tbl\_employee where id = #{id}  </select>  </mapper> |

## 4、environments（MyBatis06\_Environments）

environments环境的配置，Mybatis支持配置多个环境，通过default来指定具体使用的环境.

### 1）图一



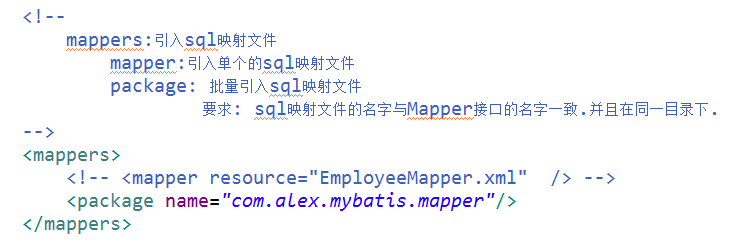
### 2）mybatis-config.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE configuration  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-config.dtd">  <!-- MyBatis的全局配置文件 -->  <configuration>    <!-- environments: 环境们。 支持配置多个环境. 通过default来指定具体使用的环境.  environment: 具体的一个环境配置，必须包含transactionManager，dataSource。  id: 当前环境的唯一标识.  transactionManager: 事务管理器.  JDBC: JdbcTransactionFactory  将来事务管理会交给Spring的声明式事务.@Trancation  dataSource:数据源.  POOLED: PooledDataSourceFactory  将来数据源交给Spring管理,使用c3p0或者dbcp等  -->  <environments default="development">  <environment id="development">  <transactionManager type="JDBC" />  <dataSource type="POOLED">  <property name="driver" value="com.mysql.jdbc.Driver" />  <property name="url" value="jdbc:mysql://localhost:3306/mybatis" />  <property name="username" value="root" />  <property name="password" value="root" />  </dataSource>  </environment>    <environment id="test">  <transactionManager type="JDBC" />  <dataSource type="POOLED">  <property name="driver" value="com.mysql.jdbc.Driver" />  <property name="url" value="jdbc:mysql://localhost:3306/mybatis" />  <property name="username" value="root" />  <property name="password" value="root" />  </dataSource>  </environment>  </environments>    <mappers>  <mapper resource="EmployeeMapper.xml" />  </mappers>    </configuration> |

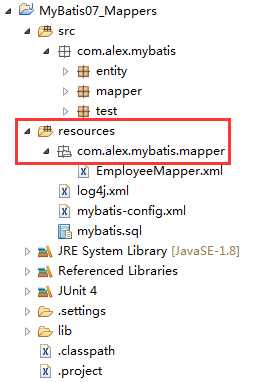
## 5、mappers（MyBatis07\_Mappers）

引入sql映射文件

### 1）图一



### 2）图二



### 3）mybatis-config.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE configuration  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-config.dtd">  <!-- MyBatis的全局配置文件 -->  <configuration>  <environments default="development">  <environment id="development">  <transactionManager type="JDBC" />  <dataSource type="POOLED">  <property name="driver" value="com.mysql.jdbc.Driver" />  <property name="url" value="jdbc:mysql://localhost:3306/mybatis" />  <property name="username" value="root" />  <property name="password" value="root" />  </dataSource>  </environment>  </environments>    <!--  mappers:引入sql映射文件  mapper:引入单个的sql映射文件  package: 批量引入sql映射文件  要求: sql映射文件的名字与Mapper接口的名字一致.并且在同一目录下.  -->  <mappers>  <!-- <mapper resource="EmployeeMapper.xml" /> -->  <package name="com.alex.mybatis.mapper"/>  </mappers>    </configuration> |

## 6、综合应用（MyBatis08\_Configuration）

MyBatis08\_Configuration

# 六、MyBatis 映射文件

## 1、概念

映射文件指导着MyBatis如何进行数据库增删改查，有着非常重要的意义

cache –命名空间的二级缓存配置

cache-ref – 其他命名空间缓存配置的引用。

resultMap – 自定义结果集映射

parameterMap – 已废弃！老式风格的参数映射

sql –抽取可重用语句块。

insert – 映射插入语句

update – 映射更新语句

delete – 映射删除语句

select – 映射查询语句

## 2、Insert、update、delete 元素



## 3、案例实操核心代码（MyBatis09\_CRUD）

### 1）EmployeeMapper

|  |
| --- |
| package com.alex.mybatis.mapper;  import com.alex.mybatis.entity.Employee;  public interface EmployeeMapper {  //添加Employee对象  public Boolean addEmp(Employee employee);    //删除Employee对象  public void deleteEmpById(Integer id );    //修改Employee对象  public void updateEmp(Employee employee);    //根据id查询Employe对象  public Employee getEmpById(Integer id);  } |

### 2）EmployeeMapper.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <!-- Employee的sql映射文件 -->  <!--  namespace: 名称空间.  如果不使用Mapper接口开发，则namesapace可以随便写  如果使用Mapper接口开发，则nameSpace不能随便写.写成Mapper接口的全类名.  -->  <mapper namespace="com.alex.mybatis.mapper.EmployeeMapper">    <!-- public Employee getEmpById(Integer id ); -->  <select id="getEmpById" resultType="employee" >  select \* from tbl\_employee where id = #{id}  </select>      <!-- public void addEmp(Employee employee);  parameterType:用于指定参数的类型. 该配置可以省略不配.Mybatis可以自动识别参数类型  -->  <insert id="addEmp" parameterType="com.alex.mybatis.entity.Employee" >  insert into tbl\_employee(last\_name,email,gender) values(#{lastName},#{email},#{gender})  </insert>    <!-- public void updateEmp(Employee employee); -->  <update id="updateEmp" >  update tbl\_employee  set last\_name = #{lastName},  email = #{email},  gender = #{gender}  where id = #{id}  </update>    <!-- public void deleteEmpById(Integer id ); -->  <delete id="deleteEmpById">  delete from tbl\_employee where id = #{id}  </delete>    </mapper> |

### 3）MyBatisTest

|  |
| --- |
| package com.alex.mybatis.test;  import java.io.InputStream;  import org.apache.ibatis.io.Resources;  import org.apache.ibatis.session.SqlSession;  import org.apache.ibatis.session.SqlSessionFactory;  import org.apache.ibatis.session.SqlSessionFactoryBuilder;  import org.junit.Test;  import com.alex.mybatis.entity.Employee;  import com.alex.mybatis.mapper.EmployeeMapper;  public class MyBatisTest {  public SqlSessionFactory getSqlSessionFactory() throws Exception {  String resource = "mybatis-config.xml";  InputStream inputStream = Resources.getResourceAsStream(resource);  SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);  System.out.println(sqlSessionFactory);  return sqlSessionFactory;  }  @Test  public void testCRUD\_insert() throws Exception {  SqlSessionFactory ssf = getSqlSessionFactory();  SqlSession session = ssf.openSession();  // ssf.openSession(true); 获取带自动提交的session  try {  EmployeeMapper mapper = session.getMapper(EmployeeMapper.class);  // 添加  Employee employee = new Employee(null, "Rose", "alex@aliyun.com", "0");  Boolean result = mapper.addEmp(employee);  System.out.println(result);  // 提交  session.commit();  } finally {  session.close();  }  }  @Test  public void testCRUD\_delete() throws Exception {  SqlSessionFactory ssf = getSqlSessionFactory();  SqlSession session = ssf.openSession();  // ssf.openSession(true); 获取带自动提交的session  try {  EmployeeMapper mapper = session.getMapper(EmployeeMapper.class);  // 删除  mapper.deleteEmpById(1003);  // 提交  session.commit();  } finally {  session.close();  }  }  @Test  public void testCRUD\_update() throws Exception {  SqlSessionFactory ssf = getSqlSessionFactory();  SqlSession session = ssf.openSession();  // ssf.openSession(true); 获取带自动提交的session  try {  EmployeeMapper mapper = session.getMapper(EmployeeMapper.class);  // 添加  Employee employee = new Employee(1004, "Rose", "rose@qq.com", "0");  // 修改  mapper.updateEmp(employee);  // 提交  session.commit();  } finally {  session.close();  }  }  @Test  public void testCRUD\_select() throws Exception {  SqlSessionFactory ssf = getSqlSessionFactory();  SqlSession session = ssf.openSession();  // ssf.openSession(true); 获取带自动提交的session  try {  EmployeeMapper mapper = session.getMapper(EmployeeMapper.class);  Employee employee = mapper.getEmpById(1001);  System.out.println(employee);  } finally {  session.close();  }  }  } |

### 4）mybatis-config.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE configuration  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-config.dtd">  <!-- MyBatis的全局配置文件 -->  <configuration>  <!-- properties:  引入外部化的配置文件  resource: 加载类路径下的资源文件  url: 加载网络路径或者是磁盘路径下的资源文件  -->  <properties resource="db.properties" ></properties>    <!-- settings:  包含很多重要的设置项，可以改变MyBatis框架的运行行为  setting:具体的一个设置项  name: 设置项的名称  value:设置项的取值  -->  <settings>  <!-- 映射下划线到驼峰命名 last\_name ==> lastName -->  <setting name="mapUnderscoreToCamelCase" value="true"/>  </settings>    <!-- typeAliases  别名处理，给java类型取别名(简短)  typeAlias:给单个的java类型取别名  type:指定java类型(全类名)  alias:指定别名。默认的别名是类名的首字母小写. 别名不区分大小写.  package:批量取别名，给指定的包下的所有的类取默认的别名  name:指定包名  -->  <typeAliases>  <!-- <typeAlias type="com.alex.mybatis.entity.Employee" alias="employee"/> -->  <package name="com.alex.mybatis.entity"/>  </typeAliases>    <!-- environments: 环境们。 支持配置多个环境. 通过default来指定具体使用的环境.  environment: 具体的一个环境配置，必须包含transactionManager，dataSource。  id: 当前环境的唯一标识.  transactionManager: 事务管理器.  JDBC: JdbcTransactionFactory  将来事务管理会交给Spring的声明式事务.@Trancation  dataSource:数据源.  POOLED: PooledDataSourceFactory  将来数据源交给Spring管理,使用c3p0或者dbcp等  -->  <environments default="development">  <environment id="development">  <transactionManager type="JDBC" />  <dataSource type="POOLED">  <property name="driver" value="${jdbc.driver}" />  <property name="url" value="${jdbc.url}" />  <property name="username" value="${jdbc.username}" />  <property name="password" value="${jdbc.password}" />  </dataSource>  </environment>    <environment id="test">  <transactionManager type="JDBC" />  <dataSource type="POOLED">  <property name="driver" value="${jdbc.driver}" />  <property name="url" value="${jdbc.url}" />  <property name="username" value="${jdbc.username}" />  <property name="password" value="${jdbc.password}" />  </dataSource>  </environment>  </environments>    <!--  mappers:引入sql映射文件  mapper:引入单个的sql映射文件  package: 批量引入sql映射文件  要求: sql映射文件的名字与Mapper接口的名字一致.并且在同一目录下.  -->  <mappers>  <!-- <mapper resource="EmployeeMapper.xml" /> -->  <package name="com.alex.mybatis.mapper"/>  </mappers>    </configuration> |

### 5）log4j.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">    <log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">    <appender name="STDOUT" class="org.apache.log4j.ConsoleAppender">  <param name="Encoding" value="UTF-8" />  <layout class="org.apache.log4j.PatternLayout">  <param name="ConversionPattern" value="%-5p %d{MM-dd HH:mm:ss,SSS} %m (%F:%L) \n" />  </layout>  </appender>  <logger name="java.sql">  <level value="debug" />  </logger>  <logger name="org.apache.ibatis">  <level value="info" />  </logger>  <root>  <level value="debug" />  <appender-ref ref="STDOUT" />  </root>  </log4j:configuration> |

### 6）db.properties

|  |
| --- |
| jdbc.driver=com.mysql.jdbc.Driver  jdbc.url=jdbc:mysql://localhost:3306/mybatis  jdbc.username=root  jdbc.password=root |

## 4、映射文件总结

1）select 完成查询操作

2）insert 完成添加操作

3）update 完成更新操作

4）delete 完成删除操作

5）select、insert、update、delete标签都必须使用id来指定唯一标识. 都可以选配parameterType.

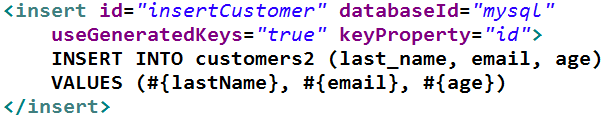
6）select标签中的resultType是用来指定 让MyBatis将一条数据封装成什么类型的对象.

7）对于insert update delete操作来说，如果想要获取到对数据库的影响条数，直接在接口的方法上声明返回值即可。支持: int Integer long Long boolean Boolean 等

# 七、MyBatis 主键生成方式以及主键值的获取

## 1、概念

若数据库支持自动生成主键的字段（比如 MySQL 和 SQL Server），则可以设置 useGeneratedKeys=”true”，然后再把 keyProperty 设置到目标属性上



## 2、案例实操核心代码（MyBatis10\_Key）

### 1）MyBatisTest.java

|  |
| --- |
| package com.alex.mybatis.test;  import java.io.InputStream;  import org.apache.ibatis.io.Resources;  import org.apache.ibatis.session.SqlSession;  import org.apache.ibatis.session.SqlSessionFactory;  import org.apache.ibatis.session.SqlSessionFactoryBuilder;  import org.junit.Test;  import com.alex.mybatis.entity.Employee;  import com.alex.mybatis.mapper.EmployeeMapper;  public class MyBatisTest {  public SqlSessionFactory getSqlSessionFactory() throws Exception {  String resource = "mybatis-config.xml";  InputStream inputStream = Resources.getResourceAsStream(resource);  SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);  System.out.println(sqlSessionFactory);  return sqlSessionFactory;  }  @Test  public void testCRUD\_insert() throws Exception {  SqlSessionFactory ssf = getSqlSessionFactory();  SqlSession session = ssf.openSession();  // ssf.openSession(true); 获取带自动提交的session  try {  EmployeeMapper mapper = session.getMapper(EmployeeMapper.class);  // 添加  Employee employee = new Employee(null, "Rose", "alex@aliyun.com", "0");  Boolean result = mapper.addEmp(employee);  System.out.println(result);  System.out.println("key:" + employee.getId());  // 提交  session.commit();  } finally {  session.close();  }  }  } |

### 2）EmployeeMapper.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <!-- Employee的sql映射文件 -->  <!--  namespace: 名称空间.  如果不使用Mapper接口开发，则namesapace可以随便写  如果使用Mapper接口开发，则nameSpace不能随便写.写成Mapper接口的全类名.  -->  <mapper namespace="com.alex.mybatis.mapper.EmployeeMapper">    <!-- public void addEmp(Employee employee);  parameterType:用于指定参数的类型. 该配置可以省略不配.Mybatis可以自动识别参数类型  userGeneratedKeys="true": 使用的是自增主键  keyProperty: 指定用来接收数据库返回的主键值 的 属性。  jdbc: Statement.getGeneratedKeys()  -->  <insert id="addEmp" parameterType="com.alex.mybatis.entity.Employee"  useGeneratedKeys="true" keyProperty="id">  insert into tbl\_employee(last\_name,email,gender) values(#{lastName},#{email},#{gender})  </insert>    </mapper> |

# 八、参数传递（MyBatis11\_Parameter）

## 1、单个参数

单个普通类型的参数(基本类型/包装类型+String)

MyBatis不会做特殊的处理。

取值:#{参数名(随便)}

## 2、多个参数

### 1）操作

|  |
| --- |
| public Employee getEmpByIdAndLastName(Integer id,String lastName);  <!-- public Employee getEmpByIdAndLastName(Integer id , String lastName); -->  <select id="getEmpByIdAndLastName" resultType="com.alex.mybatis.entity.Employee">  select \* from  tbl\_employee where id = #{id} and last\_name = #{lastName}  </select> |

### 2）异常

org.apache.ibatis.exceptions.PersistenceException:

Cause: org.apache.ibatis.binding.BindingException: Parameter 'id' not found.

Available parameters are [0, 1, param1, param2]

### 3）修改

多个参数的情况下，MyBatis会做特殊的处理，多个参数会被封装成一个map

封装规则:

key: param1 param2 .... paramN / 0 1 2 ... N-1

value: 具体传入的参数值

封装结果:{0=1001,1="Tom",param1=1001,param2="Tom"}

|  |
| --- |
| public Employee getEmpByIdAndLastName(Integer id,String lastName);  <!-- public Employee getEmpByIdAndLastName(Integer id , String lastName); -->  <select id="getEmpByIdAndLastName" resultType="com.alex.mybatis.entity.Employee">  select \* from  tbl\_employee where id = #{0} and last\_name = #{1}  </select> |

## 3、命名参数

1）明确的指定封装map时，使用key。

2）在方法的形参前面使用@Param("key值")来指定封装map所用的key.

3）操作

|  |
| --- |
| public Employee getEmpByIdAndLastName(@Param("id") Integer id, @Param("lastName") String lastName);  <!-- public Employee getEmpByIdAndLastName(Integer id , String lastName); -->  <select id="getEmpByIdAndLastName" resultType="com.alex.mybatis.entity.Employee">  select \* from  tbl\_employee where id = #{id} and last\_name = #{lastName}  </select> |

4）封装规则

key: 使用@Param注解指定的key / param1 param2 .... paramN

value: 具体传递的参数值

封装结果: {id=1001,lastName="Tom",param1=1001,param2="Tom"}

## 4、POJO

如果参数很多，正好是我们业务逻辑的数据模型中的属性,直接传入POJO

取值: #{对象的属性名}

## 5、Map

如果参数很多，不是业务逻辑的数据模型中的属性，不经常时候用。 为了方便，直接传入map

也就是说，我们也可以封装多个参数为map，直接传递

取值:#{map中的key}

## 6、TO： Transfer Object 数据传输对象

如果参数很多，不是业务逻辑的数据模型中的属性，经常使用， 推荐编写一个TO进行数据传输.

eg: 分页操作

频繁的传输: pageSize pageNo totalPages isFirstPage isLastPage isHasPrevPage isHasNextPage

Page{

int pageSize;

int pageNo;

.....

}

## 7、如果参数是集合或者是数组类型

如果参数是集合或者是数组类型，Mybatis也会做特殊的处理，封装Map

取值时用的key:

参数是collection(list、set)： collection

参数是list: collection/list

参数是Array: array

## 8、解析源码(了解)

args:[1001,"Tom"]

names={0=id,1=lastName}

|  |
| --- |
| public Object getNamedParams(Object[] args) {  final int paramCount = names.size();  if (args == null || paramCount == 0) {  return null;  } else if (!hasParamAnnotation && paramCount == 1) {  return args[names.firstKey()];  } else {  final Map<String, Object> param = new ParamMap<Object>();  int i = 0;  for (Map.Entry<Integer, String> entry : names.entrySet()) {  param.put(entry.getValue(), args[entry.getKey()]);  // add generic param names (param1, param2, ...)  final String genericParamName = GENERIC\_NAME\_PREFIX + String.valueOf(i + 1);  // ensure not to overwrite parameter named with @Param  if (!names.containsValue(genericParamName)) {  param.put(genericParamName, args[entry.getKey()]);  }  i++;  }  return param;  }  } |

# 九、Select 元素

1、Select元素来定义查询操作。

2、Id：唯一标识符。

用来引用这条语句，需要和接口的方法名一致

3、parameterType：参数类型。

可以不传，MyBatis会根据TypeHandler自动推断

4、resultType：返回值类型。

别名或者全类名，如果返回的是集合，定义集合中元素的类型。不能和resultMap同时使用



# 十、ResultMap 高级映射

ResultMap 实现自定义结果集高级映射

## 1、ResultType 自动映射

全局setting设置autoMappingBehavior默认是PARTIAL，开启自动映射的功能。唯一的要求是列名和javaBean属性名一致

如果autoMappingBehavior设置为null则会取消自动映射

数据库字段命名规范，POJO属性符合驼峰命名法，如A\_COLUMNaColumn，我们可以开启自动驼峰命名规则映射功能，mapUnderscoreToCamelCase=true。

autoMappingBehavior 没有设置，默认是开启的

## 2、ResultMap 自定义映射

实现高级结果集映射

1）id 映射主键列

2）result 映射普通列

列名和属性名一样的话，是可以省略的

3）association:映射关联属性

（1）property: 指定关联的属性

（2）javaType: 指定关联属性的类型

（3）select: 分步查询中指定调用的查询的sql语句的id值.

（4）column: 分步查询中调用sql查询时传递的参数

4）association分步查询支持延迟加载

编辑mybatis-config.xml文件，在全局settings设置中加入两个配置

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

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

5）collection ：映射集合类型的关联属性.

（1）property：指定关联属性

（2）ofType: 指定的是集合中的元素类型

（3）select: 分步查询中指定要调用的查询sql的id值

（4）column: 分步查询中传给调用的sql的参数

## 3、id + result 级联实操（MyBatis12\_ResultMap\_Cascade）

**需求: 查询员工对象 并且查询到员工所属部门信息**

### 1）EmployeeMapper.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <!-- Employee的sql映射文件 -->  <!--  namespace: 名称空间.  如果不使用Mapper接口开发，则namesapace可以随便写  如果使用Mapper接口开发，则nameSpace不能随便写.写成Mapper接口的全类名.  -->  <mapper namespace="com.alex.mybatis.mapper.EmployeeMapper">  <!-- 需求: 查询员工对象 并且查询到员工所属部门信息 -->  <!-- public Employee getEmpAndDept(Integer id ); -->  <select id="getEmpAndDept" resultMap="MyEmpDept">  select  e.id eid ,e.last\_name ,e.gender, e.email ,e.d\_id ,  d.id did ,d.department\_name  from tbl\_employee e,tbl\_dept d  where e.d\_id = d.id and e.id=#{id}  </select>  <!--  type:封装的类型(返回值类型)  eid last\_name gender email d\_id did department\_name  1001 Tom 1 tom@sina.com 1 1 开发部  -->  <resultMap type="employee" id="MyEmpDept">  <!-- 用于映射主键列  column:指定列名  property:指定属性名  列名和属性名一样的话，是可以省略的  -->  <id column="eid" property="id"/>  <!-- 用于映射普通列 -->  <result column="last\_name" property="lastName"/>  <result column="gender" property="gender"/>  <result column="email" property="email"/>  <!-- 使用级联的方式 -->  <result column="did" property="department.id"/>  <result column="department\_name" property="department.departmentName"/>  </resultMap>    </mapper> |

## 4、association 实操（MyBatis13\_ResultMap\_Association）

### 1）EmployeeMapper.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <mapper namespace="com.alex.mybatis.mapper.EmployeeMapper">  <!-- 需求: 查询员工对象 并且查询到员工所属部门信息 -->  <!-- public Employee getEmpAndDept(Integer id ); -->  <select id="getEmpAndDept" resultMap="MyEmpDept">  select  e.id eid ,e.last\_name ,e.gender, e.email ,e.d\_id ,  d.id did ,d.department\_name  from tbl\_employee e,tbl\_dept d  where e.d\_id = d.id and e.id=#{id}  </select>  <!--  type:封装的类型(返回值类型)  eid last\_name gender email d\_id did department\_name  1001 Tom 1 tom@sina.com 1 1 开发部  -->  <resultMap type="employee" id="MyEmpDept">  <!-- 用于映射主键列  column:指定列名  property:指定属性名  -->  <id column="eid" property="id"/>  <!-- 用于映射普通列 -->  <result column="last\_name" property="lastName"/>  <result column="gender" property="gender"/>  <result column="email" property="email"/>    <!-- 使用association -->  <association property="department" javaType="com.alex.mybatis.entity.Department">  <id column="did" property="id"/>  <result column="department\_name" property="departmentName"/>  </association>  </resultMap>    </mapper> |

## 5、association 分步实操（MyBatis14\_ResultMap\_Association\_Step）

### 1）EmployeeMapper.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <mapper namespace="com.alex.mybatis.mapper.EmployeeMapper">  <!-- association 支持分步查询  需求:查询员工对象 并且查询到员工所属部门信息  1.先根据id查询员工对象  2.根据查询到的d\_id 查询部门对象  -->  <!-- public Employee getEmpAndDeptStep(Integer id ); -->  <select id="getEmpAndDeptStep" resultMap="myEmpAndDeptStep">  select \* from tbl\_employee where id = #{id}  </select>    <resultMap type="com.alex.mybatis.entity.Employee" id="myEmpAndDeptStep">  <id column="id" property="id"/>  <result column="last\_name" property="lastName"/>  <result column="gender" property="gender"/>  <result column="email" property="email"/>  <association property="department"  select="com.alex.mybatis.mapper.DepartmentMapper.getDeptById"  column="d\_id">  </association>  </resultMap>    </mapper> |

### 2）DepartmentMapper.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <mapper namespace="com.alex.mybatis.mapper.DepartmentMapper">  <!-- public Department getDeptById(Integer id ); -->  <select id="getDeptById" resultType="com.alex.mybatis.entity.Department">  select \* from tbl\_dept where id = #{id}  </select>    </mapper> |

## 6、association Lazy 分步实操（MyBatis15\_ResultMap\_Association\_Step\_Lazy）

懒加载、按需加载

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE configuration  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-config.dtd">  <!-- MyBatis的全局配置文件 -->  <configuration>  <!-- properties:  引入外部化的配置文件  resource: 加载类路径下的资源文件  url: 加载网络路径或者是磁盘路径下的资源文件  -->  <properties resource="db.properties" ></properties>    <!-- settings:  包含很多重要的设置项，可以改变MyBatis框架的运行行为  setting:具体的一个设置项  name: 设置项的名称  value:设置项的取值  -->  <settings>  <!-- 映射下划线到驼峰命名 last\_name ==> lastName -->  <setting name="mapUnderscoreToCamelCase" value="true"/>  <!-- 开启延迟加载 -->  <setting name="lazyLoadingEnabled" value="true"/>  <!-- 指定加载的属性是按需加载. -->  <setting name="aggressiveLazyLoading" value="false"/>  </settings>    <!-- typeAliases  别名处理，给java类型取别名(简短)  typeAlias:给单个的java类型取别名  type:指定java类型(全类名)  alias:指定别名。默认的别名是类名的首字母小写. 别名不区分大小写.  package:批量取别名，给指定的包下的所有的类取默认的别名  name:指定包名  -->  <typeAliases>  <!-- <typeAlias type="com.alex.mybatis.entity.Employee" alias="employee"/> -->  <package name="com.alex.mybatis.entity"/>  </typeAliases>    <!-- environments: 环境们。 支持配置多个环境. 通过default来指定具体使用的环境.  environment: 具体的一个环境配置，必须包含transactionManager，dataSource。  id: 当前环境的唯一标识.  transactionManager: 事务管理器.  JDBC: JdbcTransactionFactory  将来事务管理会交给Spring的声明式事务.@Trancation  dataSource:数据源.  POOLED: PooledDataSourceFactory  将来数据源交给Spring管理,使用c3p0或者dbcp等  -->  <environments default="development">  <environment id="development">  <transactionManager type="JDBC" />  <dataSource type="POOLED">  <property name="driver" value="${jdbc.driver}" />  <property name="url" value="${jdbc.url}" />  <property name="username" value="${jdbc.username}" />  <property name="password" value="${jdbc.password}" />  </dataSource>  </environment>    <environment id="test">  <transactionManager type="JDBC" />  <dataSource type="POOLED">  <property name="driver" value="${jdbc.driver}" />  <property name="url" value="${jdbc.url}" />  <property name="username" value="${jdbc.username}" />  <property name="password" value="${jdbc.password}" />  </dataSource>  </environment>  </environments>    <!--  mappers:引入sql映射文件  mapper:引入单个的sql映射文件  package: 批量引入sql映射文件  要求: sql映射文件的名字与Mapper接口的名字一致.并且在同一目录下.  -->  <mappers>  <!-- <mapper resource="EmployeeMapper.xml" /> -->  <package name="com.alex.mybatis.mapper"/>  </mappers>    </configuration> |

## 7、Collection（MyBatis16\_ResultMap\_Collection）

一对多

### 1）Department

|  |
| --- |
| package com.alex.mybatis.entity;  import java.util.List;  public class Department {    private Integer id;  private String departmentName;  private List<Employee> employeeList;    public Department() {    }  public Department(Integer id, String departmentName) {  this.id = id;  this.departmentName = departmentName;  }  public Integer getId() {  return id;  }  public void setId(Integer id) {  this.id = id;  }  public String getDepartmentName() {  return departmentName;  }  public void setDepartmentName(String departmentName) {  this.departmentName = departmentName;  }  public List<Employee> getEmployeeList() {  return employeeList;  }  public void setEmployeeList(List<Employee> employeeList) {  this.employeeList = employeeList;  }  @Override  public String toString() {  return "Department [id=" + id + ", departmentName=" + departmentName + ", employeeList=" + employeeList + "]";  }    } |

### 2）DepartmentMapper.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <mapper namespace="com.alex.mybatis.mapper.DepartmentMapper">    <!-- public Department getDeptAndEmps(Integer id ); -->  <select id="getDeptAndEmps" resultMap="MyDeptAndEmps">  select  d.id did ,d.department\_name,  e.id eid ,e.last\_name ,e.email,e.gender ,e.d\_id  from tbl\_dept d left outer join tbl\_employee e  on d.id = e.d\_id  where d.id = #{id}  </select>    <resultMap id="MyDeptAndEmps" type="com.alex.mybatis.entity.Department" >  <id column="did" property="id"/>  <result column="department\_name" property="departmentName"/>    <collection property="employeeList" ofType="com.alex.mybatis.entity.Employee">  <id column="eid" property="id"/>  <result column="last\_name" property="lastName"/>  <result column="email" property="email"/>  <result column="gender" property="gender"/>  </collection>  </resultMap>    </mapper> |

## 8、Collection 分布实操（MyBatis17\_ResultMap\_Collection\_Step）

### 1）DepartmentMapper.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <mapper namespace="com.alex.mybatis.mapper.DepartmentMapper">    <!-- public Department getDeptAndEmpsStep(Integer id ); -->  <select id="getDeptAndEmpsStep" resultMap="myDeptAndEmpsStep">  select \* from tbl\_dept where id = #{id}  </select>    <resultMap id="myDeptAndEmpsStep" type="com.alex.mybatis.entity.Department" >  <id column="id" property="id"/>  <result column="department\_name" property="departmentName"/>  <collection property="employeeList"  select="com.alex.mybatis.mapper.EmployeeMapper.getEmployeeList"  column="id">  </collection>  </resultMap>    </mapper> |

### 2）EmployeeMapper.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <mapper namespace="com.alex.mybatis.mapper.EmployeeMapper">  <!-- 需求二: 查询部门的时候，将该部门中所有的员工信息也查询出来,在DepartmentMapper中讲解 -->  <!-- public List<Employee> getEmployeeList(Integer did ); -->  <select id="getEmployeeList" resultType="com.alex.mybatis.entity.Employee">  select \* from tbl\_employee where d\_id = #{did}  </select>  </mapper> |

## 9、Collection 懒加载实操（MyBatis18\_ResultMap\_Collection\_Step\_Lazy）

按需加载

### 1）mybatis-config.xml

|  |
| --- |
| <settings>  <!-- 映射下划线到驼峰命名 last\_name ==> lastName -->  <setting name="mapUnderscoreToCamelCase" value="true"/>  <!-- 开启延迟加载 -->  <setting name="lazyLoadingEnabled" value="true"/>  <!-- 指定加载的属性是按需加载. -->  <setting name="aggressiveLazyLoading" value="false"/>  </settings> |

# 十一、MyBatis 动态SQL

## 1、MyBatis 动态SQL简述

1）动态 SQL是MyBatis强大特性之一。极大的简化我们拼装SQL的操作。

2）动态 SQL 元素和使用 JSTL 或其他类似基于 XML 的文本处理器相似。

3）MyBatis 采用功能强大的基于 OGNL 的表达式来简化操作。

if

choose (when, otherwise)

trim (where, set)

foreach

## 2、if + where 实操（MyBatis19\_DynamicSQL\_if\_where）

<if>: 进行条件的判断

<Where>:解决拼装sql的时候where关键的问题, 以及where后面第一个出现的and/or问题.

### 1）MyBatisTestDynamicSQL.java

|  |
| --- |
| package com.alex.mybatis.test;  import java.io.InputStream;  import java.util.List;  import org.apache.ibatis.io.Resources;  import org.apache.ibatis.session.SqlSession;  import org.apache.ibatis.session.SqlSessionFactory;  import org.apache.ibatis.session.SqlSessionFactoryBuilder;  import org.junit.Test;  import com.alex.mybatis.entity.Employee;  import com.alex.mybatis.mapper.EmployeeMapperDynamicSQL;  public class MyBatisTestDynamicSQL {    public SqlSessionFactory getSqlSessionFactory() throws Exception {  String resource = "mybatis-config.xml";  InputStream inputStream = Resources.getResourceAsStream(resource);  SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);  System.out.println(sqlSessionFactory);  return sqlSessionFactory;  }  @Test  public void testIf() throws Exception{  SqlSessionFactory ssf = getSqlSessionFactory();  SqlSession session = ssf.openSession();  try {  EmployeeMapperDynamicSQL mapper = session.getMapper(EmployeeMapperDynamicSQL.class);  Employee employee = new Employee();  employee.setId(1001);  employee.setLastName("%T%");  employee.setEmail("Tom@aliyun.com");  employee.setGender("1");  List<Employee> emps =mapper.getEmpsByConditionIf(employee);  System.out.println(emps);  }finally {  session.close();  }  }    } |

### 2）EmployeeMapperDynamicSQL.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <mapper namespace="com.alex.mybatis.mapper.EmployeeMapperDynamicSQL">    <!-- public List<Employee> getEmpsByConditionIf(Employee employee); -->  <select id="getEmpsByConditionIf" resultType="com.alex.mybatis.entity.Employee">  select \* from tbl\_employee <!-- where 1=1 -->  <where>  <!--  test: 判断的表达式 OGNL表达式  -->  <if test="id!=null">  and id = #{id}  </if>  <if test="lastName!=null &amp;&amp; lastName!=&quot;&quot;">  and last\_name like #{lastName}  </if>  <if test="email!=null and email.trim()!=''">  and email =#{email}  </if>  <if test="gender==0 or gender ==1">  and gender = #{gender}  </if>  </where>  </select>  </mapper> |

## 3、trim 实操（MyBatis20\_DynamicSQL\_Trim）

<trim>: 解决拼装sql的时候，sql语句前后的添加/截取指定字符的问题.

prefix: 添加一个前缀

prefixOverrides: 去掉前缀

suffix: 添加一个后缀

suffixOverrides: 去掉后缀

### 1）MyBatisTestDynamicSQL.java

|  |
| --- |
| package com.alex.mybatis.test;  import java.io.InputStream;  import java.util.List;  import org.apache.ibatis.io.Resources;  import org.apache.ibatis.session.SqlSession;  import org.apache.ibatis.session.SqlSessionFactory;  import org.apache.ibatis.session.SqlSessionFactoryBuilder;  import org.junit.Test;  import com.alex.mybatis.entity.Employee;  import com.alex.mybatis.mapper.EmployeeMapperDynamicSQL;  public class MyBatisTestDynamicSQL {    public SqlSessionFactory getSqlSessionFactory() throws Exception {  String resource = "mybatis-config.xml";  InputStream inputStream = Resources.getResourceAsStream(resource);  SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);  System.out.println(sqlSessionFactory);  return sqlSessionFactory;  }  @Test  public void testTrim() throws Exception{  SqlSessionFactory ssf = getSqlSessionFactory();  SqlSession session = ssf.openSession();  try {  EmployeeMapperDynamicSQL mapper = session.getMapper(EmployeeMapperDynamicSQL.class);  Employee employee = new Employee();  employee.setId(1001);  employee.setLastName("%T%");  employee.setEmail("Tom@aliyun.com");  employee.setGender("1");  List<Employee> emps =mapper.getEmpsByConditionTrim(employee);  System.out.println(emps);  }finally {  session.close();  }  }    } |

### 2）EmployeeMapperDynamicSQL.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <mapper namespace="com.alex.mybatis.mapper.EmployeeMapperDynamicSQL">    <!-- public List<Employee> getEmpsByConditionTrim(Employee employee); -->  <select id="getEmpsByConditionTrim" resultType="com.alex.mybatis.entity.Employee">  select \* from tbl\_employee  <trim prefix="where" suffixOverrides="and">  <if test="id!=null">  id = #{id} and  </if>  <if test="lastName!=null &amp;&amp; lastName!=&quot;&quot;">  last\_name like #{lastName} and  </if>  <if test="email!=null and email.trim()!=''">  email =#{email} and  </if>  <if test="gender==0 or gender ==1">  gender = #{gender} and  </if>  </trim>  </select>  </mapper> |

## 4、Set 实操（MyBatis21\_DynamicSQL\_Set）

<set>: 解决拼装sql的时候， 逗号的问题.

### 1）MyBatisTestDynamicSQL.java

|  |
| --- |
| package com.alex.mybatis.test;  import java.io.InputStream;  import java.util.List;  import org.apache.ibatis.io.Resources;  import org.apache.ibatis.session.SqlSession;  import org.apache.ibatis.session.SqlSessionFactory;  import org.apache.ibatis.session.SqlSessionFactoryBuilder;  import org.junit.Test;  import com.alex.mybatis.entity.Employee;  import com.alex.mybatis.mapper.EmployeeMapperDynamicSQL;  public class MyBatisTestDynamicSQL {  public SqlSessionFactory getSqlSessionFactory() throws Exception {  String resource = "mybatis-config.xml";  InputStream inputStream = Resources.getResourceAsStream(resource);  SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);  System.out.println(sqlSessionFactory);  return sqlSessionFactory;  }  @Test  public void testUpdateSet() throws Exception {  SqlSessionFactory ssf = getSqlSessionFactory();  SqlSession session = ssf.openSession();  try {  EmployeeMapperDynamicSQL mapper = session.getMapper(EmployeeMapperDynamicSQL.class);  Employee employee = new Employee();  employee.setId(1001);  employee.setLastName("T");  employee.setEmail("Tom@aliyun.com");  employee.setGender("1");  int updatecount = mapper.updateEmpByConditionSet(employee);  System.out.println(updatecount);  } finally {  session.close();  }  }  } |

### 2）EmployeeMapperDynamicSQL.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <mapper namespace="com.alex.mybatis.mapper.EmployeeMapperDynamicSQL">    <update id="updateEmpByConditionSet">  update tbl\_employee <!-- set -->  <set>  <if test="lastName!=null &amp;&amp; lastName!=&quot;&quot;">  last\_name = #{lastName},  </if>  <if test="email!=null and email.trim()!=''">  email = #{email},  </if>  <if test="gender==0 or gender ==1">  gender = #{gender},  </if>  </set>  where id = #{id}  </update>  </mapper> |

## 5、Choose 实操（MyBatis22\_DynamicSQL\_Choose）

<choose> <when> <otherwise>: 类似于带了break的switch case分支语句，只会执行其中的一个分支，当什么条件都不满足的时候，会执行<otherwise>

### 1）MyBatisTestDynamicSQL.java

|  |
| --- |
| package com.alex.mybatis.test;  import java.io.InputStream;  import java.util.List;  import org.apache.ibatis.io.Resources;  import org.apache.ibatis.session.SqlSession;  import org.apache.ibatis.session.SqlSessionFactory;  import org.apache.ibatis.session.SqlSessionFactoryBuilder;  import org.junit.Test;  import com.alex.mybatis.entity.Employee;  import com.alex.mybatis.mapper.EmployeeMapperDynamicSQL;  public class MyBatisTestDynamicSQL {  public SqlSessionFactory getSqlSessionFactory() throws Exception {  String resource = "mybatis-config.xml";  InputStream inputStream = Resources.getResourceAsStream(resource);  SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);  System.out.println(sqlSessionFactory);  return sqlSessionFactory;  }  @Test  public void testSelectChoose() throws Exception {  SqlSessionFactory ssf = getSqlSessionFactory();  SqlSession session = ssf.openSession();  try {  EmployeeMapperDynamicSQL mapper = session.getMapper(EmployeeMapperDynamicSQL.class);  Employee employee = new Employee();  employee.setId(1001);  employee.setLastName("T");  employee.setEmail("Tom@aliyun.com");  employee.setGender("1");  List<Employee> emps = mapper.getEmpsByConditionChoose(employee);  System.out.println(emps);  } finally {  session.close();  }  }  } |

### 2）EmployeeMapperDynamicSQL.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <mapper namespace="com.alex.mybatis.mapper.EmployeeMapperDynamicSQL">  <select id="getEmpsByConditionChoose" resultType="com.alex.mybatis.entity.Employee">  select \* from tbl\_employee  <where>  <choose>  <when test="id != null">  id = #{id}  </when>  <when test="lastName!=null &amp;&amp; lastName!=&quot;&quot;">  last\_name like #{lastName}  </when>  <when test="email!=null and email.trim()!=''">  email = #{email}  </when>  <otherwise>  gender = 0  </otherwise>  </choose>  </where>  </select>  </mapper> |

## 6、Foreach 实操（MyBatis23\_DynamicSQL\_Foreach）

<foreach>: 用于迭代

collection: 指定要迭代的集合

item:用来表示当前从集合中迭代出的元素

open: 开始字符

close:结束字符

separator: 元素与元素之间的分隔符

index: 如果迭代的是集合，index表示集合元素的下标

如果迭代的是map,index表示的是map中当前元素的key.

### 1）EmployeeMapperDynamicSQL.java

|  |
| --- |
| package com.alex.mybatis.mapper;  import java.util.List;  import org.apache.ibatis.annotations.Param;  import com.alex.mybatis.entity.Employee;  public interface EmployeeMapperDynamicSQL {    public List<Employee> getEmpsByConditionForeach(List<Integer> ids);    public List<Employee> getEmpsByConditionForeachParam(@Param("ids")List<Integer> ids);    } |

### 2）MyBatisTestDynamicSQL.java

|  |
| --- |
| package com.alex.mybatis.test;  import java.io.InputStream;  import java.util.ArrayList;  import java.util.List;  import org.apache.ibatis.io.Resources;  import org.apache.ibatis.session.SqlSession;  import org.apache.ibatis.session.SqlSessionFactory;  import org.apache.ibatis.session.SqlSessionFactoryBuilder;  import org.junit.Test;  import com.alex.mybatis.entity.Employee;  import com.alex.mybatis.mapper.EmployeeMapperDynamicSQL;  public class MyBatisTestDynamicSQL {  public SqlSessionFactory getSqlSessionFactory() throws Exception {  String resource = "mybatis-config.xml";  InputStream inputStream = Resources.getResourceAsStream(resource);  SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);  System.out.println(sqlSessionFactory);  return sqlSessionFactory;  }  @Test  public void testSelectForeach() throws Exception {  SqlSessionFactory ssf = getSqlSessionFactory();  SqlSession session = ssf.openSession();  try {  EmployeeMapperDynamicSQL mapper = session.getMapper(EmployeeMapperDynamicSQL.class);  List<Integer> ids = new ArrayList<Integer>();  ids.add(1001);  ids.add(1002);  ids.add(1003);  List<Employee> emps = mapper.getEmpsByConditionForeach(ids);  List<Employee> emps2 = mapper.getEmpsByConditionForeachParam(ids);  System.out.println(emps);  System.out.println(emps2);  } finally {  session.close();  }  }  } |

### 3）EmployeeMapperDynamicSQL.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <mapper namespace="com.alex.mybatis.mapper.EmployeeMapperDynamicSQL">  <select id="getEmpsByConditionForeach" resultType="com.alex.mybatis.entity.Employee">  <!--  select \* from tbl\_employee where id = ? or id = ? or id = ?  select \* from tbl\_employee where id in(?,?,?)  -->  select \* from tbl\_employee where id in  <foreach collection="list" item="curr\_id" open="(" close=")" separator=",">  #{curr\_id}  </foreach>  </select>    <select id="getEmpsByConditionForeachParam" resultType="com.alex.mybatis.entity.Employee">  <!--  select \* from tbl\_employee where id = ? or id = ? or id = ?  select \* from tbl\_employee where id in(?,?,?)  -->  select \* from tbl\_employee where id in  <foreach collection="ids" item="curr\_id" open="(" close=")" separator=",">  #{curr\_id}  </foreach>  </select>  </mapper> |

## 7、Foreach Batch（MyBatis24\_DynamicSQL\_Batch）

1）批量操作: 与数据库的一次会话期间，执行多次操作.

2）批量操作的方式：

insert into xxx (column ,column2,column3...) values(?,?,?),(?,?,?)....

insert into xxx (column ,column2,column3...) values(?,?,?);

insert into xxx (column ,column2,column3...) values(?,?,?);

insert into xxx (column ,column2,column3...) values(?,?,?);

对于第二中操作，默认情况不支持的。 需要在链接的url上添加: allowMultiQueries=true.

### 1）EmployeeMapperDynamicSQL

|  |
| --- |
| package com.alex.mybatis.mapper;  import java.util.List;  import org.apache.ibatis.annotations.Param;  import com.alex.mybatis.entity.Employee;  public interface EmployeeMapperDynamicSQL {    public void addEmpsOne(@Param("emps")List<Employee> emps);    public void addEmpsTwo(@Param("emps")List<Employee> emps);    } |

### 2）MyBatisTestDynamicSQL

|  |
| --- |
| package com.alex.mybatis.test;  import java.io.InputStream;  import java.util.ArrayList;  import java.util.List;  import org.apache.ibatis.io.Resources;  import org.apache.ibatis.session.SqlSession;  import org.apache.ibatis.session.SqlSessionFactory;  import org.apache.ibatis.session.SqlSessionFactoryBuilder;  import org.junit.Test;  import com.alex.mybatis.entity.Employee;  import com.alex.mybatis.mapper.EmployeeMapperDynamicSQL;  public class MyBatisTestDynamicSQL {  public SqlSessionFactory getSqlSessionFactory() throws Exception {  String resource = "mybatis-config.xml";  InputStream inputStream = Resources.getResourceAsStream(resource);  SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);  System.out.println(sqlSessionFactory);  return sqlSessionFactory;  }  @Test  public void testBatch() throws Exception{  SqlSessionFactory ssf = getSqlSessionFactory();  SqlSession session = ssf.openSession();  try {  EmployeeMapperDynamicSQL mapper = session.getMapper(EmployeeMapperDynamicSQL.class);  List<Employee> emps = new ArrayList<Employee>();  emps.add(new Employee(null, "YangBuHui", "ybh@aliyun.com", "0"));  emps.add(new Employee(null,"YinSuSu","yss@aliyun.com","0"));  emps.add(new Employee(null,"小昭","xz@aliyun.com","0"));    mapper.addEmpsOne(emps);  mapper.addEmpsTwo(emps);  session.commit();  }finally {  session.close();  }  }  } |

### 3）EmployeeMapperDynamicSQL.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <mapper namespace="com.alex.mybatis.mapper.EmployeeMapperDynamicSQL">  <!-- public void addEmps(@Param("emps")List<Employee> emps); Mysql数据库的批量插入:  1. insert into xxx (column ,column2,column3...) values(?,?,?),(?,?,?)....  2. insert into xxx (column ,column2,column3...) values(?,?,?); insert into  xxx (column ,column2,column3...) values(?,?,?); insert into xxx (column ,column2,column3...)  values(?,?,?); -->  <insert id="addEmpsOne">  insert into tbl\_employee(last\_name,email,gender) values  <foreach collection="emps" item="emp" separator=",">  (#{emp.lastName},#{emp.email},#{emp.gender})  </foreach>  </insert>  <!-- 默认情况下，不支持将多条sql语句通过;隔开，拼成一条sql来使用 解决: 在链接的url上: allowMultiQueries=true -->  <insert id="addEmpsTwo">  <foreach collection="emps" item="emp">  insert into tbl\_employee(last\_name,email,gender) values  (#{emp.lastName},#{emp.email},#{emp.gender});  </foreach>  </insert>  </mapper> |

### 4）db.properties

|  |
| --- |
| jdbc.driver=com.mysql.jdbc.Driver  jdbc.url=jdbc:mysql://localhost:3306/mybatis?allowMultiQueries=true  jdbc.username=root  jdbc.password=root |

# 十二、MyBatis-缓存机制

## 1、概念

1）MyBatis 包含一个非常强大的查询缓存特性,它可以非常方便地配置和定制。缓存可以极大的提升查询效率

2）MyBatis系统中默认定义了两级缓存。

3）一级缓存和二级缓存。

（1）默认情况下，只有一级缓存（SqlSession级别的缓存，也称为本地缓存）开启。

（2）二级缓存需要手动开启和配置，他是基于namespace级别的缓存。

（3）为了提高扩展性。MyBatis定义了缓存接口Cache。我们可以通过实现Cache接口来自定义二级缓存

## 2、一级缓存（本地缓存）（MyBatis25\_One\_Cache）

1）一级缓存默认是开始的。

2）一级缓存是基于SqlSession级别的缓存. 每个sqlSession对象都有自己的一级缓存，且相互不共享.

3）与数据库的一次会话期间查询到的数据库会放在本地缓存, 以后如果需要获取相同的数据，则直接从缓存中获取，而没必要去数据库查询.

|  |
| --- |
| @Test  public void testFirstLevelCache() throws Exception {  SqlSessionFactory ssf = getSqlSessionFactory();  SqlSession session = ssf.openSession();  try {  EmployeeMapperCache mapper = session.getMapper(EmployeeMapperCache.class);  Employee employee1 = mapper.getEmpById(1001);  System.out.println(employee1);  System.out.println("--------------------------");  Employee employee2 = mapper.getEmpById(1001);  System.out.println(employee2);  } finally {  session.close();  }  } |

尖叫提示：第二次查询没有sql语句，直接从缓存里面取出来的，一级缓存是默认配置

## 3、一级缓存失效的情况（MyBatis26\_One\_FailCache）

1）sqlSession不同。

2）sqlSession相同，但是查询的条件不同.

3）sqlSession相同，两次查询期间执行了增删改操作. 认为这次增删改操作可能会改变当前缓存中数据对应着的数据库中的数据.

4）sqlSession相同，手动清除了一级缓存.

|  |
| --- |
| @Test  public void testFirstLevelCache() throws Exception {  SqlSessionFactory ssf = getSqlSessionFactory();  SqlSession session = ssf.openSession();    //一级缓存失效方式一  SqlSession session2 = ssf.openSession();    try {  EmployeeMapperCache mapper = session.getMapper(EmployeeMapperCache.class);  EmployeeMapperCache mapper2 = session2.getMapper(EmployeeMapperCache.class);    Employee employee1 = mapper.getEmpById(1001);  System.out.println(employee1);  System.out.println("--------------------------");  // 一级缓存失效方式二 -> 查询和查询之间有增删改操作  // mapper.deleteEmpById(1004);  //一级缓存失效方式三 -> 手动清除缓存  session.clearCache(); // 清除缓存  Employee employee2 = mapper2.getEmpById(1001);  System.out.println(employee2);  } finally {  session.close();  }  } |

## 4、二级缓存（全局缓存）概述

1）二级缓存默认不开启，需要通过手动的方式配置开启.

2）二级缓存是基于namespace级别的缓存.多个SqlSession可以共享二级缓存

3）工作机制:

一个会话(SqlSession)查询一条数据，这个数据会被放在当前会话的一级缓存中

当会话关闭或者提交,才会把一级缓存中的数据保存到二级缓存中

新的会话查询数据的时候，就可以从二级缓存中查找.

4）使用步骤

（1）开启全局二级缓存配置

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

（2）在想要使用二级缓存的sql映射文件中配置使用二级缓存

<cache eviction="LRU" flushInterval="60000" readOnly="false" size="1024" ></cache>

（3）java类实现序列化接口

public class Employee implements Serializable{

private static final long serialVersionUID = 1L;

## 5、二级缓存实操（MyBatis27\_Two\_Cache）

### 1）Employee.java

|  |
| --- |
| package com.alex.mybatis.entity;  import java.io.Serializable;  public class Employee implements Serializable{    private static final long serialVersionUID = 1L;    private Integer id;  private String lastName;  private String email;  private String gender;  private Department department;//员工所属的部门    public Employee() {    }  public Employee(Integer id, String lastName, String email, String gender) {  this.id = id;  this.lastName = lastName;  this.email = email;  this.gender = gender;  }  public Integer getId() {  return id;  }  public void setId(Integer id) {  this.id = id;  }  public String getLastName() {  return lastName;  }  public void setLastName(String lastName) {  this.lastName = lastName;  }  public String getEmail() {  return email;  }  public void setEmail(String email) {  this.email = email;  }  public String getGender() {  return gender;  }  public void setGender(String gender) {  this.gender = gender;  }  public Department getDepartment() {  return department;  }  public void setDepartment(Department department) {  this.department = department;  }  @Override  public String toString() {  return "Employee [id=" + id + ", lastName=" + lastName + ", email=" + email + ", gender=" + gender  + ", department=" + department + "]";  }    } |

### 2）mybatis-config.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE configuration  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-config.dtd">  <!-- MyBatis的全局配置文件 -->  <configuration>  <!-- properties:  引入外部化的配置文件  resource: 加载类路径下的资源文件  url: 加载网络路径或者是磁盘路径下的资源文件  -->  <properties resource="db.properties" ></properties>    <!-- settings:  包含很多重要的设置项，可以改变MyBatis框架的运行行为  setting:具体的一个设置项  name: 设置项的名称  value:设置项的取值  -->  <settings>  <!-- 映射下划线到驼峰命名 last\_name ==> lastName -->  <setting name="mapUnderscoreToCamelCase" value="true"/>  <!-- 开启延迟加载 -->  <setting name="lazyLoadingEnabled" value="true"/>  <!-- 指定加载的属性是按需加载. -->  <setting name="aggressiveLazyLoading" value="false"/>  <!-- 二级缓存 -->  <setting name="cacheEnabled" value="true"/>  </settings>    <!-- typeAliases  别名处理，给java类型取别名(简短)  typeAlias:给单个的java类型取别名  type:指定java类型(全类名)  alias:指定别名。默认的别名是类名的首字母小写. 别名不区分大小写.  package:批量取别名，给指定的包下的所有的类取默认的别名  name:指定包名  -->  <typeAliases>  <!-- <typeAlias type="com.alex.mybatis.entity.Employee" alias="employee"/> -->  <package name="com.alex.mybatis.entity"/>  </typeAliases>    <!-- environments: 环境们。 支持配置多个环境. 通过default来指定具体使用的环境.  environment: 具体的一个环境配置，必须包含transactionManager，dataSource。  id: 当前环境的唯一标识.  transactionManager: 事务管理器.  JDBC: JdbcTransactionFactory  将来事务管理会交给Spring的声明式事务.@Trancation  dataSource:数据源.  POOLED: PooledDataSourceFactory  将来数据源交给Spring管理,使用c3p0或者dbcp等  -->  <environments default="development">  <environment id="development">  <transactionManager type="JDBC" />  <dataSource type="POOLED">  <property name="driver" value="${jdbc.driver}" />  <property name="url" value="${jdbc.url}" />  <property name="username" value="${jdbc.username}" />  <property name="password" value="${jdbc.password}" />  </dataSource>  </environment>    <environment id="test">  <transactionManager type="JDBC" />  <dataSource type="POOLED">  <property name="driver" value="${jdbc.driver}" />  <property name="url" value="${jdbc.url}" />  <property name="username" value="${jdbc.username}" />  <property name="password" value="${jdbc.password}" />  </dataSource>  </environment>  </environments>    <!--  mappers:引入sql映射文件  mapper:引入单个的sql映射文件  package: 批量引入sql映射文件  要求: sql映射文件的名字与Mapper接口的名字一致.并且在同一目录下.  -->  <mappers>  <!-- <mapper resource="EmployeeMapper.xml" /> -->  <package name="com.alex.mybatis.mapper"/>  </mappers>    </configuration> |

### 3）EmployeeMapperCache.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <mapper namespace="com.alex.mybatis.mapper.EmployeeMapperCache">  <!-- 配置使用二级缓存  eviction: 缓存的回收策略. 默认值是LRU  LRU: 最近最少使用的.  FIFO: 先进先出  flushInternval: 缓存的刷新间隔  缓存多长时间清空一次. 默认是不清空.  可以通过一个毫秒值来设置清空的时间  readOnly: 设置缓存是否只读.  true: 只读 MyBatis认为所有从缓存中获取数据的操作都是只读的。不会进行修改。  MyBatis为了加快获取速度，直接将数据在缓存中的引用交给用户.  速度快，不安全  false: 非只读 MyBatis认为从缓存中获取的数据可能会被修改》  MyBatis就会利用序列化及反序列的技术克隆一份新的数据交给用户.  安全，速度稍慢.  size: 缓存中存放多少元素  type: 指定使用的缓存类. 主要用来指定自定义的缓存实现类或者是第三方的缓存实现类.  <cache eviction="LRU" flushInterval="60000" readOnly="false" size="1024" ></cache>  -->  <!-- 配置Ehcache -->  <cache eviction="LRU" flushInterval="60000" readOnly="false" size="1024" ></cache>    <!-- public Employee getEmpById(Integer id ); -->  <select id="getEmpById" resultType="Employee">  select \* from tbl\_employee where id = #{id}  </select>    </mapper> |

### 4）MyBatisTestCache.java

|  |
| --- |
| package com.alex.mybatis.test;  import java.io.InputStream;  import org.apache.ibatis.io.Resources;  import org.apache.ibatis.session.SqlSession;  import org.apache.ibatis.session.SqlSessionFactory;  import org.apache.ibatis.session.SqlSessionFactoryBuilder;  import org.junit.Test;  import com.alex.mybatis.entity.Employee;  import com.alex.mybatis.mapper.EmployeeMapperCache;  public class MyBatisTestCache {  public SqlSessionFactory getSqlSessionFactory() throws Exception {  String resource = "mybatis-config.xml";  InputStream inputStream = Resources.getResourceAsStream(resource);  SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);  System.out.println(sqlSessionFactory);  return sqlSessionFactory;  }  @Test  public void testSecondLevelCache() throws Exception{  SqlSessionFactory ssf = getSqlSessionFactory();  SqlSession session1 = ssf.openSession();  SqlSession session2 = ssf.openSession();    try {  EmployeeMapperCache mapper1 = session1.getMapper(EmployeeMapperCache.class);  EmployeeMapperCache mapper2 = session2.getMapper(EmployeeMapperCache.class);    Employee employee1 = mapper1.getEmpById(1001);  System.out.println(employee1);  session1.commit();    System.out.println("--------------------------");    Employee employee2 = mapper2.getEmpById(1001);  System.out.println(employee2);    }finally {  session1.close();  session2.close();  }  }  } |

### 5）Console

|  |
| --- |
| org.apache.ibatis.session.defaults.DefaultSqlSessionFactory@31dc339b  DEBUG 10-24 09:54:44,729 Cache Hit Ratio [com.alex.mybatis.mapper.EmployeeMapperCache]: 0.0 (LoggingCache.java:62)  DEBUG 10-24 09:54:44,965 ==> Preparing: select \* from tbl\_employee where id = ? (BaseJdbcLogger.java:145)  DEBUG 10-24 09:54:44,997 ==> Parameters: 1001(Integer) (BaseJdbcLogger.java:145)  DEBUG 10-24 09:54:45,020 <== Total: 1 (BaseJdbcLogger.java:145)  Employee [id=1001, lastName=Tom, email=Tom@aliyun.com, gender=1, department=null]  --------------------------  DEBUG 10-24 09:54:45,070 Cache Hit Ratio [com.alex.mybatis.mapper.EmployeeMapperCache]: 0.5 (LoggingCache.java:62)  Employee [id=1001, lastName=Tom, email=Tom@aliyun.com, gender=1, department=null] |

## 6、缓存常用设置

1）cacheEnabled=true/false: 改为false后，关闭的是二级缓存，一级缓存依旧可以使用。

2）每个select标签都有 useCache="true"的属性:

当改为false后,不使用二级缓存， 一级缓存依旧可以使用 .

3）每个增删改标签都有 flushCache="true":

缓存是可以使用的，但是会清空一二级缓存中的数据

每个select标签也有 flushCache 属性，默认值是false

4）sqlSession.clearCache(): 只清空当前session的一级缓存，二级缓存还有，不清空

5）localeCacheScope: 本地缓存作用域, 影响一级缓存的.

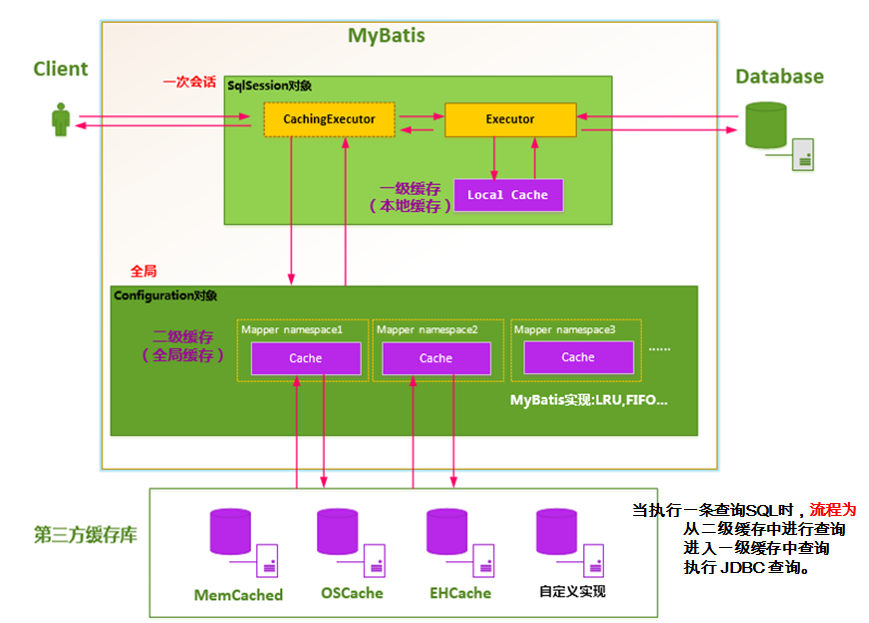
SESSION: 一次会话.

STATEMENT: 一次Sql的执行期间. 可以禁用一级缓存.

<setting name="localeCacheScope" value="session"/>

## 7、缓存的查找顺序:二一库

先二级缓存，然后再一级缓存，最后再数据库



## 8、第三方缓存

### 1）概念

EhCache 是一个纯Java的进程内缓存框架，具有快速、精干等特点，是Hibernate中默认的CacheProvider。

MyBatis定义了Cache接口方便我们进行自定义扩展。

步骤：

1、导入ehcache包，以及整合包，日志包

ehcache-core-2.6.8.jar、mybatis-ehcache-1.0.3.jar

slf4j-api-1.6.1.jar、slf4j-log4j12-1.6.2.jar

2、编写ehcache.xml配置文件

3、配置cache标签

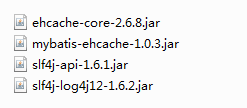
<cache type="org.mybatis.caches.ehcache.EhcacheCache"></cache>

参照缓存：若想在命名空间中共享相同的缓存配置和实例。可以使用 cache-ref 元素来引用另外一个缓存。



### 2）步骤

（1）导入相关jar包



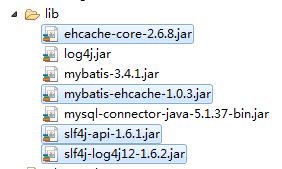
（2）添加ehcache的配置文件

（3）在sql映射文件中配置使用的二级缓存实现类.

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

### 3）实操（MyBatis28\_Three\_Cache）

#### （1） jar包 –> Build Path



#### （2）ehcache.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:noNamespaceSchemaLocation="../config/ehcache.xsd">  <!-- 磁盘保存路径 -->  <diskStore path="D:\alex\ehcache" />    <defaultCache  maxElementsInMemory="1"  maxElementsOnDisk="10000000"  eternal="false"  overflowToDisk="true"  timeToIdleSeconds="120"  timeToLiveSeconds="120"  diskExpiryThreadIntervalSeconds="120"  memoryStoreEvictionPolicy="LRU">  </defaultCache>  </ehcache>    <!--  属性说明：  l diskStore：指定数据在磁盘中的存储位置。  l defaultCache：当借助CacheManager.add("demoCache")创建Cache时，EhCache便会采用<defalutCache/>指定的的管理策略    以下属性是必须的：  l maxElementsInMemory - 在内存中缓存的element的最大数目  l maxElementsOnDisk - 在磁盘上缓存的element的最大数目，若是0表示无穷大  l eternal - 设定缓存的elements是否永远不过期。如果为true，则缓存的数据始终有效，如果为false那么还要根据timeToIdleSeconds，timeToLiveSeconds判断  l overflowToDisk - 设定当内存缓存溢出的时候是否将过期的element缓存到磁盘上    以下属性是可选的：  l timeToIdleSeconds - 当缓存在EhCache中的数据前后两次访问的时间超过timeToIdleSeconds的属性取值时，这些数据便会删除，默认值是0,也就是可闲置时间无穷大  l timeToLiveSeconds - 缓存element的有效生命期，默认是0.,也就是element存活时间无穷大  diskSpoolBufferSizeMB 这个参数设置DiskStore(磁盘缓存)的缓存区大小.默认是30MB.每个Cache都应该有自己的一个缓冲区.  l diskPersistent - 在VM重启的时候是否启用磁盘保存EhCache中的数据，默认是false。  l diskExpiryThreadIntervalSeconds - 磁盘缓存的清理线程运行间隔，默认是120秒。每个120s，相应的线程会进行一次EhCache中数据的清理工作  l memoryStoreEvictionPolicy - 当内存缓存达到最大，有新的element加入的时候， 移除缓存中element的策略。默认是LRU（最近最少使用），可选的有LFU（最不常使用）和FIFO（先进先出）  --> |

#### （3）EmployeeMapperCache.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <mapper namespace="com.alex.mybatis.mapper.EmployeeMapperCache">  <!-- 配置使用二级缓存  eviction: 缓存的回收策略. 默认值是LRU  LRU: 最近最少使用的.  FIFO: 先进先出  flushInternval: 缓存的刷新间隔  缓存多长时间清空一次. 默认是不清空.  可以通过一个毫秒值来设置清空的时间  readOnly: 设置缓存是否只读.  true: 只读 MyBatis认为所有从缓存中获取数据的操作都是只读的。不会进行修改。  MyBatis为了加快获取速度，直接将数据在缓存中的引用交给用户.  速度快，不安全  false: 非只读 MyBatis认为从缓存中获取的数据可能会被修改》  MyBatis就会利用序列化及反序列的技术克隆一份新的数据交给用户.  安全，速度稍慢.  size: 缓存中存放多少元素  type: 指定使用的缓存类. 主要用来指定自定义的缓存实现类或者是第三方的缓存实现类.  <cache eviction="LRU" flushInterval="60000" readOnly="false" size="1024" ></cache>  -->  <!-- 配置Ehcache -->  <cache type="org.mybatis.caches.ehcache.EhcacheCache"></cache>    <!-- public Employee getEmpById(Integer id ); -->  <select id="getEmpById" resultType="Employee" >  select \* from tbl\_employee where id = #{id}  </select>    </mapper> |

#### （4）MyBatisTestCache.java

|  |
| --- |
| package com.alex.mybatis.test;  import java.io.InputStream;  import org.apache.ibatis.io.Resources;  import org.apache.ibatis.session.SqlSession;  import org.apache.ibatis.session.SqlSessionFactory;  import org.apache.ibatis.session.SqlSessionFactoryBuilder;  import org.junit.Test;  import com.alex.mybatis.entity.Employee;  import com.alex.mybatis.mapper.EmployeeMapperCache;  public class MyBatisTestCache {  public SqlSessionFactory getSqlSessionFactory() throws Exception {  String resource = "mybatis-config.xml";  InputStream inputStream = Resources.getResourceAsStream(resource);  SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);  System.out.println(sqlSessionFactory);  return sqlSessionFactory;  }  @Test  public void testThreeLevelCache() throws Exception{  SqlSessionFactory ssf = getSqlSessionFactory();  SqlSession session1 = ssf.openSession();  SqlSession session2 = ssf.openSession();    try {  EmployeeMapperCache mapper1 = session1.getMapper(EmployeeMapperCache.class);  EmployeeMapperCache mapper2 = session2.getMapper(EmployeeMapperCache.class);    Employee employee1 = mapper1.getEmpById(1001);  System.out.println(employee1);  session1.commit();    System.out.println("--------------------------");    Employee employee2 = mapper2.getEmpById(1001);  System.out.println(employee2);    }finally {  session1.close();  session2.close();  }  }  } |

#### （5）Console

|  |
| --- |
| DEBUG 10-24 10:44:03,577 Configuring ehcache from ehcache.xml found in the classpath: file:/E:/workspace/eclipse/MyBatis2019/MyBatis28\_Three\_Cache/bin/ehcache.xml (ConfigurationFactory.java:132)  DEBUG 10-24 10:44:03,581 Configuring ehcache from URL: file:/E:/workspace/eclipse/MyBatis2019/MyBatis28\_Three\_Cache/bin/ehcache.xml (ConfigurationFactory.java:98)  DEBUG 10-24 10:44:03,582 Configuring ehcache from InputStream (ConfigurationFactory.java:150)  DEBUG 10-24 10:44:03,597 Ignoring ehcache attribute xmlns:xsi (BeanHandler.java:271)  DEBUG 10-24 10:44:03,598 Ignoring ehcache attribute xsi:noNamespaceSchemaLocation (BeanHandler.java:271)  DEBUG 10-24 10:44:03,600 Disk Store Path: D:\alex\ehcache (DiskStoreConfiguration.java:141)  DEBUG 10-24 10:44:03,610 Creating new CacheManager with default config (CacheManager.java:1036)  DEBUG 10-24 10:44:03,615 propertiesString is null. (PropertyUtil.java:88)  DEBUG 10-24 10:44:03,625 No CacheManagerEventListenerFactory class specified. Skipping... (ConfigurationHelper.java:185)  DEBUG 10-24 10:44:04,040 No BootstrapCacheLoaderFactory class specified. Skipping... (Cache.java:955)  DEBUG 10-24 10:44:04,041 CacheWriter factory not configured. Skipping... (Cache.java:929)  DEBUG 10-24 10:44:04,042 No CacheExceptionHandlerFactory class specified. Skipping... (ConfigurationHelper.java:96)  DEBUG 10-24 10:44:04,063 Initialized net.sf.ehcache.store.MemoryStore for com.alex.mybatis.mapper.EmployeeMapperCache (MemoryStore.java:152)  DEBUG 10-24 10:44:04,074 Using diskstore path D:\alex\ehcache (DiskStorePathManager.java:169)  DEBUG 10-24 10:44:04,075 Holding exclusive lock on D:\alex\ehcache\.ehcache-diskstore.lock (DiskStorePathManager.java:170)  DEBUG 10-24 10:44:04,078 Failed to delete file com%002ealex%002emybatis%002emapper%002e%0045mployee%004dapper%0043ache.index (DiskStorageFactory.java:860)  DEBUG 10-24 10:44:04,086 Matching data file missing (or empty) for index file. Deleting index file D:\alex\ehcache\com%002ealex%002emybatis%002emapper%002e%0045mployee%004dapper%0043ache.index (DiskStorageFactory.java:168)  DEBUG 10-24 10:44:04,087 Failed to delete file com%002ealex%002emybatis%002emapper%002e%0045mployee%004dapper%0043ache.index (DiskStorageFactory.java:860)  DEBUG 10-24 10:44:04,093 Initialised cache: com.alex.mybatis.mapper.EmployeeMapperCache (Cache.java:1165)  DEBUG 10-24 10:44:04,094 CacheDecoratorFactory not configured for defaultCache. Skipping for 'com.alex.mybatis.mapper.EmployeeMapperCache'. (ConfigurationHelper.java:354)  org.apache.ibatis.session.defaults.DefaultSqlSessionFactory@5e25a92e  DEBUG 10-24 10:44:04,140 Cache Hit Ratio [com.alex.mybatis.mapper.EmployeeMapperCache]: 0.0 (LoggingCache.java:62)  DEBUG 10-24 10:44:04,356 ==> Preparing: select \* from tbl\_employee where id = ? (BaseJdbcLogger.java:145)  DEBUG 10-24 10:44:04,380 ==> Parameters: 1001(Integer) (BaseJdbcLogger.java:145)  DEBUG 10-24 10:44:04,404 <== Total: 1 (BaseJdbcLogger.java:145)  Employee [id=1001, lastName=Tom, email=Tom@aliyun.com, gender=1, department=null]  DEBUG 10-24 10:44:04,405 put added 0 on heap (Segment.java:425)  --------------------------  DEBUG 10-24 10:44:04,409 Cache Hit Ratio [com.alex.mybatis.mapper.EmployeeMapperCache]: 0.5 (LoggingCache.java:62)  Employee [id=1001, lastName=Tom, email=Tom@aliyun.com, gender=1, department=null] |

# 十三、MyBatis-逆向工程

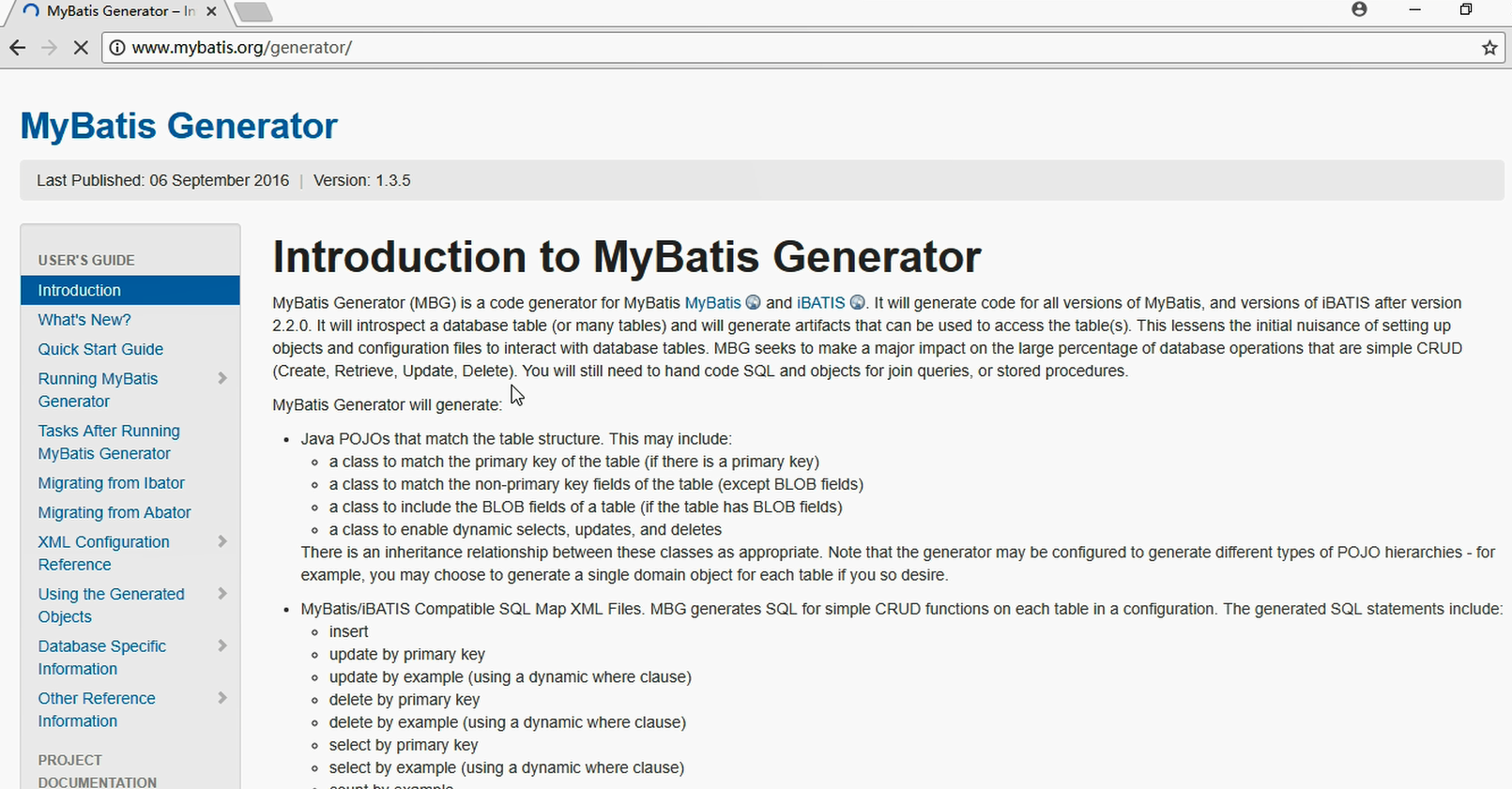
## 1、逆向工程概述

MyBatis Generator：

简称MBG，是一个专门为MyBatis框架使用者定制的代码生成器，可以快速的根据表生成对应的映射文件，接口，以及bean类。支持基本的增删改查，以及QBC风格的条件查询。但是表连接、存储过程等这些复杂sql的定义需要我们手工编写

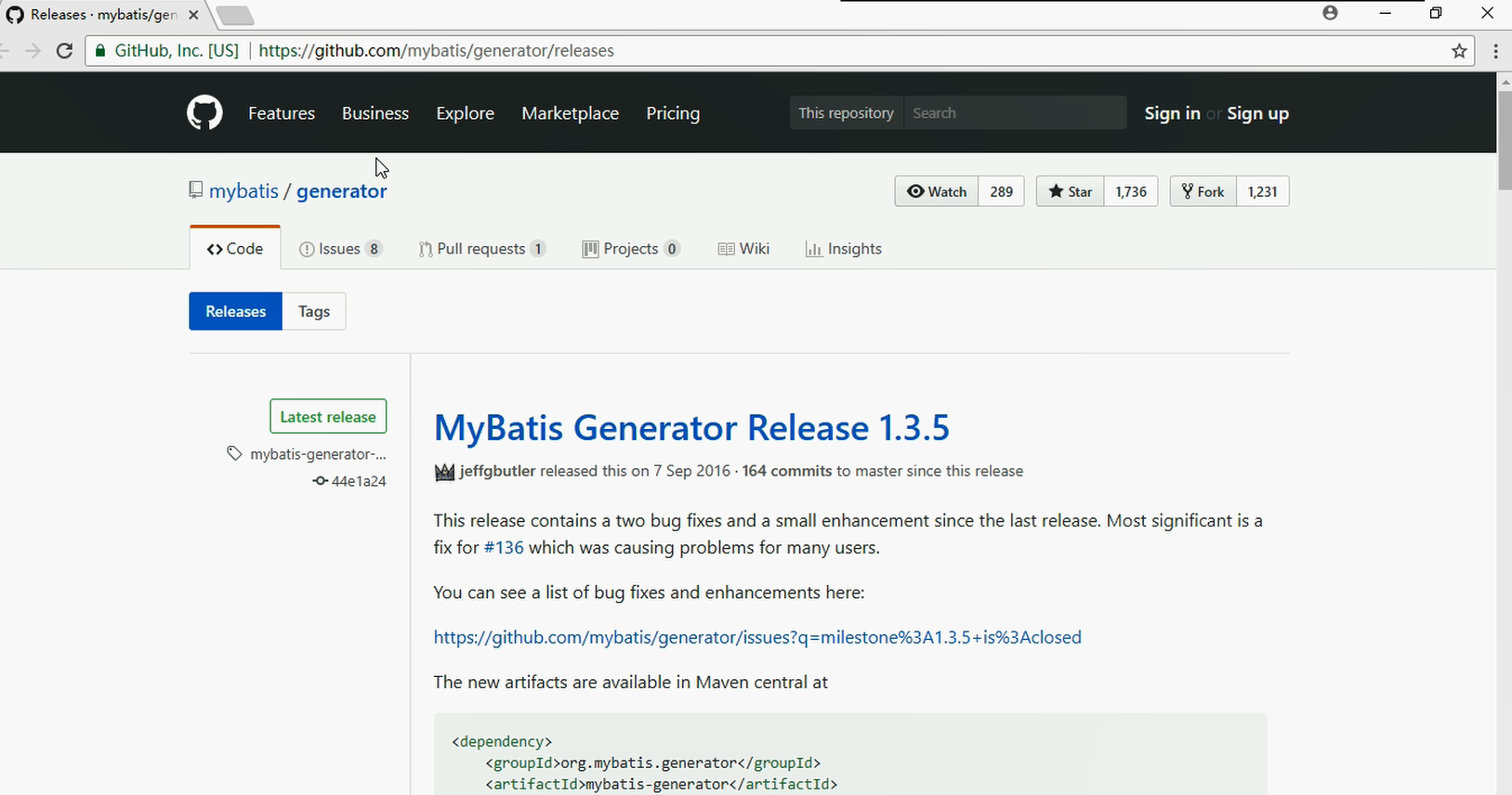
## 2、官方文档地址

http://www.mybatis.org/generator/



## 3、官方工程地址

https://github.com/mybatis/generator/releases



## 4、逆向工程实操（MyBatis29\_MBG）

使用步骤：

1）编写MBG的配置文件（重要几处配置）

（1）jdbcConnection配置数据库连接信息

（2）javaModelGenerator配置javaBean的生成策略

（3）sqlMapGenerator 配置sql映射文件生成策略

（4）javaClientGenerator配置Mapper接口的生成策略

（5）table 配置要逆向解析的数据表

tableName：表名

domainObjectName：对应的javaBean名

2）运行代码生成器生成代码

注意：

Context标签

targetRuntime=“MyBatis3“可以生成带条件的增删改查

targetRuntime=“MyBatis3Simple“可以生成基本的增删改查

如果再次生成，建议将之前生成的数据删除，避免xml向后追加内容出现的问题。

### 1）导入jar包

mybatis-generator-core-1.3.2.jar

### 2）编写mbg.xml配置文件

|  |
| --- |
| <?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>  <!-- targetRuntime: 指定生成的逆向工程的版本  MyBatis3: 生成带条件的增删改查.  MyBatis3Simple: 生成基本的增删改查.  -->  <context id="DB2Tables" targetRuntime="MyBatis3Simple">    <!-- 数据库的连接 -->  <jdbcConnection driverClass="com.mysql.jdbc.Driver"  connectionURL="jdbc:mysql://localhost:3306/mybatis"  userId="root"  password="root">  </jdbcConnection>  <!-- 指定javabean的生成策略  targetPackage: 指定包  targetProject: 指定工程  -->  <javaModelGenerator targetPackage="com.alex.mybatis.entity" targetProject=".\src">  <property name="enableSubPackages" value="true" />  <property name="trimStrings" value="true" />  </javaModelGenerator>    <!-- sql映射的生成策略 -->  <sqlMapGenerator targetPackage="com.alex.mybatis.mapper" targetProject=".\resources">  <property name="enableSubPackages" value="true" />  </sqlMapGenerator>    <!-- mapper接口的生成策略 -->  <javaClientGenerator type="XMLMAPPER" targetPackage="com.alex.mybatis.mapper" targetProject=".\src">  <property name="enableSubPackages" value="true" />  </javaClientGenerator>  <!-- 指定逆向分析的表 -->  <table tableName="tbl\_dept" domainObjectName="Department"></table>  <table tableName="tbl\_employee" domainObjectName="Employee"></table>    </context>  </generatorConfiguration> |

### 3）运行逆向工程的生成代码

MyBatisTestMBG.java

|  |
| --- |
| package com.alex.mybatis.test;  import java.io.File;  import java.io.InputStream;  import java.util.ArrayList;  import java.util.List;  import org.apache.ibatis.io.Resources;  import org.apache.ibatis.session.SqlSessionFactory;  import org.apache.ibatis.session.SqlSessionFactoryBuilder;  import org.junit.Test;  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 MyBatisTestMBG {  public SqlSessionFactory getSqlSessionFactory() throws Exception {  String resource = "mybatis-config.xml";  InputStream inputStream = Resources.getResourceAsStream(resource);  SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);  System.out.println(sqlSessionFactory);  return sqlSessionFactory;  }  @Test  public void createMBG() throws Exception {  List<String> warnings = new ArrayList<String>();  boolean overwrite = true;  File configFile = new File("mbg.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);  }    } |

### 4）生成的代码 - 查询实操（MyBatis30\_MBG\_Test）

|  |
| --- |
| package com.alex.mybatis.test;  import java.io.File;  import java.io.InputStream;  import java.util.ArrayList;  import java.util.List;  import org.apache.ibatis.io.Resources;  import org.apache.ibatis.session.SqlSession;  import org.apache.ibatis.session.SqlSessionFactory;  import org.apache.ibatis.session.SqlSessionFactoryBuilder;  import org.junit.Test;  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;  import com.alex.mybatis.entity.Employee;  import com.alex.mybatis.entity.EmployeeExample;  import com.alex.mybatis.entity.EmployeeExample.Criteria;  import com.alex.mybatis.mapper.EmployeeMapper;  public class MyBatisTestMBG {  public SqlSessionFactory getSqlSessionFactory() throws Exception {  String resource = "mybatis-config.xml";  InputStream inputStream = Resources.getResourceAsStream(resource);  SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);  System.out.println(sqlSessionFactory);  return sqlSessionFactory;  }  @Test  public void createMBG() throws Exception {  List<String> warnings = new ArrayList<String>();  boolean overwrite = true;  File configFile = new File("mbg.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);  }    //根据id查询  @Test  public void testMBG01() throws Exception {  SqlSessionFactory ssf = getSqlSessionFactory();  SqlSession session = ssf.openSession();  try {  EmployeeMapper mapper = session.getMapper(EmployeeMapper.class);  Employee employee = mapper.selectByPrimaryKey(1001);  System.out.println(employee);  } finally {  session.close();  }  }    //没有条件的情况，查询全部信息  @Test  public void testMBG02() throws Exception {  SqlSessionFactory ssf = getSqlSessionFactory();  SqlSession session = ssf.openSession();  try {  EmployeeMapper mapper = session.getMapper(EmployeeMapper.class);  List<Employee> emps = mapper.selectByExample(null);  for (Employee employee : emps) {  System.out.println(employee);  }  } finally {  session.close();  }  }    //按照条件查询  @Test  public void testMBG03() throws Exception {  SqlSessionFactory ssf = getSqlSessionFactory();  SqlSession session = ssf.openSession();  try {  EmployeeMapper mapper = session.getMapper(EmployeeMapper.class);    // 查询员工名字中带有e字母 和员工性别是2的 或者email中带有y字母的  EmployeeExample example = new EmployeeExample();    Criteria criteria = example.createCriteria();  criteria.andLastNameLike("%e%");  criteria.andGenderEqualTo("0");    //or的条件需要重新使用一个Criteria对象  Criteria criteria2 = example.createCriteria();  criteria2.andEmailLike("%y%");    //对于封装or条件的criteria，需要or到example  example.or(criteria2);  List<Employee> emps = mapper.selectByExample(example);  for (Employee employee : emps) {  System.out.println(employee);  }  } finally {  session.close();  }  }    } |

# 十四、PageHelper页面分页实现

## 1、概念

PageHelper是MyBatis中非常方便的第三方分页插件。

官方文档：

https://github.com/pagehelper/Mybatis-PageHelper/blob/master/README\_zh.md

我们可以对照官方文档的说明，快速的使用插件

## 2、使用步骤

1）导入相关包pagehelper-x.x.x.jar 和 jsqlparser-0.9.5.jar。

2）在MyBatis全局配置文件中配置分页插件。

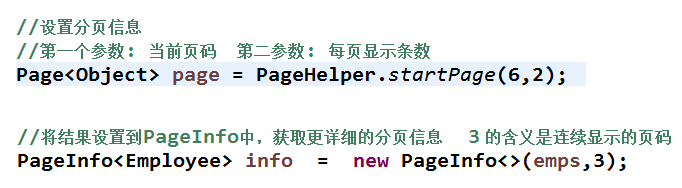


<plugins>

<plugin interceptor="com.github.pagehelper.PageInterceptor"></plugin>

</plugins>

3）使用PageHelper提供的方法进行分页



//设置分页信息

//第一个参数: 当前页码 第二参数: 每页显示条数

Page<Object> page = PageHelper.startPage(2,2);

List<Employee> emps = mapper.getEmps();

//将结果设置到PageInfo中，获取更详细的分页信息 3 的含义是连续显示的页码

PageInfo<Employee> info = new PageInfo<>(emps,3);

4）可以使用更强大的PageInfo封装返回结果



## 3、实操（MyBatis31\_PageHelper）

### 1）jar包放到lib文件夹

pagehelper-5.0.0-rc.jar

jsqlparser-0.9.5.jar

### 2）mybatis-config.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE configuration  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-config.dtd">  <!-- MyBatis的全局配置文件 -->  <configuration>  <!-- properties:  引入外部化的配置文件  resource: 加载类路径下的资源文件  url: 加载网络路径或者是磁盘路径下的资源文件  -->  <properties resource="db.properties" ></properties>    <!-- settings:  包含很多重要的设置项，可以改变MyBatis框架的运行行为  setting:具体的一个设置项  name: 设置项的名称  value:设置项的取值  -->  <settings>  <!-- 映射下划线到驼峰命名 last\_name ==> lastName -->  <setting name="mapUnderscoreToCamelCase" value="true"/>  <!-- 开启延迟加载 -->  <setting name="lazyLoadingEnabled" value="true"/>  <!-- 指定加载的属性是按需加载. -->  <setting name="aggressiveLazyLoading" value="false"/>  <!-- 二级缓存 -->  <setting name="cacheEnabled" value="true"/>  </settings>    <!-- typeAliases  别名处理，给java类型取别名(简短)  typeAlias:给单个的java类型取别名  type:指定java类型(全类名)  alias:指定别名。默认的别名是类名的首字母小写. 别名不区分大小写.  package:批量取别名，给指定的包下的所有的类取默认的别名  name:指定包名  -->  <typeAliases>  <!-- <typeAlias type="com.alex.mybatis.entity.Employee" alias="employee"/> -->  <package name="com.alex.mybatis.entity"/>  </typeAliases>      <!--  插件: 主要是用来拦截MyBatis的四大对象。完成动态的一些修改等.  Executor: 执行器 用来执行具体的增删改查  ParameterHandler: 参数处理器 给Sql语句的占位符赋值  ResultSetHandler: 结果集处理器 用于处理结果的封装  StatementHandler: sql语句编译 用于sql的编译或者预编译.  -->  <plugins>  <plugin interceptor="com.github.pagehelper.PageInterceptor"></plugin>  </plugins>    <!-- environments: 环境们。 支持配置多个环境. 通过default来指定具体使用的环境.  environment: 具体的一个环境配置，必须包含transactionManager，dataSource。  id: 当前环境的唯一标识.  transactionManager: 事务管理器.  JDBC: JdbcTransactionFactory  将来事务管理会交给Spring的声明式事务.@Trancation  dataSource:数据源.  POOLED: PooledDataSourceFactory  将来数据源交给Spring管理,使用c3p0或者dbcp等  -->  <environments default="development">  <environment id="development">  <transactionManager type="JDBC" />  <dataSource type="POOLED">  <property name="driver" value="${jdbc.driver}" />  <property name="url" value="${jdbc.url}" />  <property name="username" value="${jdbc.username}" />  <property name="password" value="${jdbc.password}" />  </dataSource>  </environment>    <environment id="test">  <transactionManager type="JDBC" />  <dataSource type="POOLED">  <property name="driver" value="${jdbc.driver}" />  <property name="url" value="${jdbc.url}" />  <property name="username" value="${jdbc.username}" />  <property name="password" value="${jdbc.password}" />  </dataSource>  </environment>  </environments>    <!--  mappers:引入sql映射文件  mapper:引入单个的sql映射文件  package: 批量引入sql映射文件  要求: sql映射文件的名字与Mapper接口的名字一致.并且在同一目录下.  -->  <mappers>  <!-- <mapper resource="EmployeeMapper.xml" /> -->  <package name="com.alex.mybatis.mapper"/>  </mappers>    </configuration> |

### 3）MyBatisTestPageHelper.java

|  |
| --- |
| package com.alex.mybatis.test;  import java.io.InputStream;  import java.util.List;  import org.apache.ibatis.io.Resources;  import org.apache.ibatis.session.SqlSession;  import org.apache.ibatis.session.SqlSessionFactory;  import org.apache.ibatis.session.SqlSessionFactoryBuilder;  import org.junit.Test;  import com.alex.mybatis.entity.Employee;  import com.alex.mybatis.mapper.EmployeeMapper;  import com.github.pagehelper.Page;  import com.github.pagehelper.PageHelper;  import com.github.pagehelper.PageInfo;  public class MyBatisTestPageHelper {  public SqlSessionFactory getSqlSessionFactory() throws Exception {  String resource = "mybatis-config.xml";  InputStream inputStream = Resources.getResourceAsStream(resource);  SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);  System.out.println(sqlSessionFactory);  return sqlSessionFactory;  }  @Test  public void testPageHelper() throws Exception {  SqlSessionFactory ssf = getSqlSessionFactory();  SqlSession session = ssf.openSession();    try {  EmployeeMapper mapper = session.getMapper(EmployeeMapper.class);    //设置分页信息  //第一个参数: 当前页码 第二参数: 每页显示条数  Page<Object> page = PageHelper.startPage(2,2);    List<Employee> emps = mapper.getEmps();    //将结果设置到PageInfo中，获取更详细的分页信息 3 的含义是连续显示的页码  PageInfo<Employee> info = new PageInfo<>(emps,3);    for (Employee employee : emps) {  System.out.println(employee);  }  System.out.println("----------------------------------");  //获取当前分页的详细信息  System.out.println("当前页码:" + page.getPageNum());  System.out.println("总记录数:" + page.getTotal());  System.out.println("总页码:" + page.getPages());  System.out.println("每页的记录数:" + page.getPageSize()) ;  System.out.println("----------------------------------");  System.out.println("当前页码:" + info.getPageNum());  System.out.println("总记录数:" + info.getTotal());  System.out.println("总页码:" + info.getPages());  System.out.println("每页的记录数:" + info.getPageSize()) ;  System.out.println("是否第一页:" + info.isIsFirstPage());  System.out.println("是否最后一页:" + info.isIsLastPage());  System.out.println("是否有上一页:" + info.isHasPreviousPage());  System.out.println("是否有下一页:" +info.isHasNextPage());  System.out.println("连续显示的页码:");  int [] nums = info.getNavigatepageNums();  for (int i : nums) {  System.out.print(i + " ");  }  //组件化分页  //在java代码中 将页面中的分页全部计算好，生成一个字符串. 将来直接在页面中展示.  //String str= "<a href=''>上一页</a>";    } finally {  session.close();  }  }  } |

# 十五、SSM整合

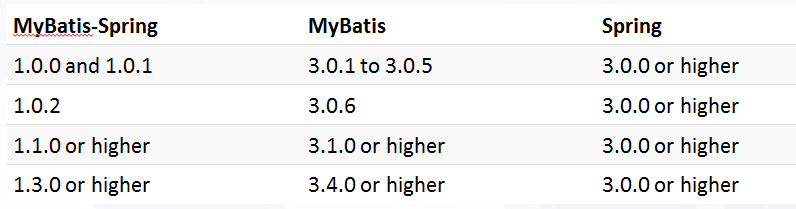
## 1、整合前需要注意的问题

**尖叫提示：框架版本是否兼容**

1） 查看不同MyBatis版本整合Spring时使用的适配包； http://www.mybatis.org/spring/

2）下载整合适配包

https://github.com/mybatis/spring/releases



3）官方整合示例，jpetstore

https://github.com/mybatis/jpetstore-6

## 2、SSM整合思想

1）导入整合需要的jar包

（1）Spring的jar包

（2）SpringMVC的jar包

（3）MyBatis的jar包

（4）整合的适配包

（5）数据库驱动包、日志包、连接池等

2）MyBatis环境的搭建

（1）MyBatis的全局配置文件

（2）编写javaBean，Mapper接口，sql映射文件

3）Spring SpringMVC环境的搭建

（1）web.xml中配置SpringIOC容器启动的监听器、SpringMVC的核心控制器、REST过滤器

（2）编写Spring的配置文件: applicationContext.xml

（3）编写SpringMVC的配置文件: springmvc.xml

4）Spring整合MyBatis

（1）配置创建SqlSession对象的SqlSessionFactoryBean

（2）配置扫描所有的Mapper接口，批量生成代理实现类，交给Spring管理的，能够完成自动注入.

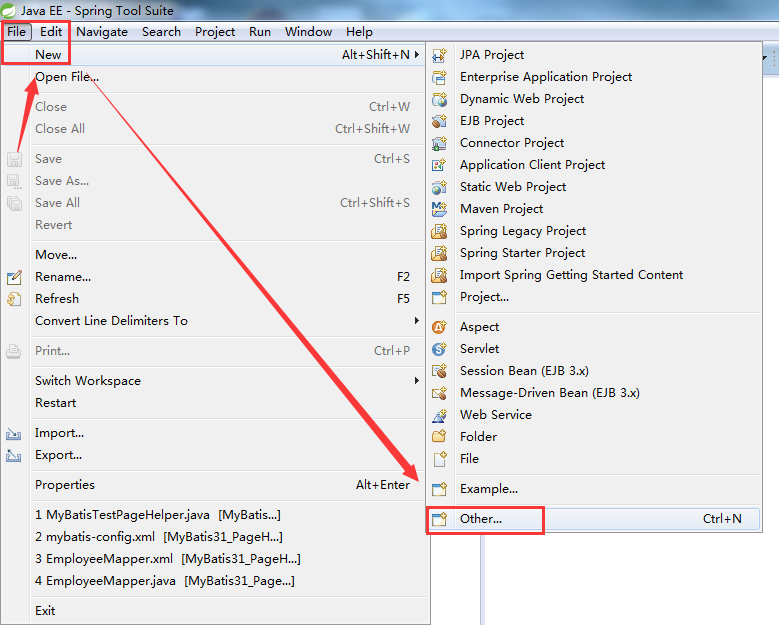
5）编码测试

完成员工的增删改查.

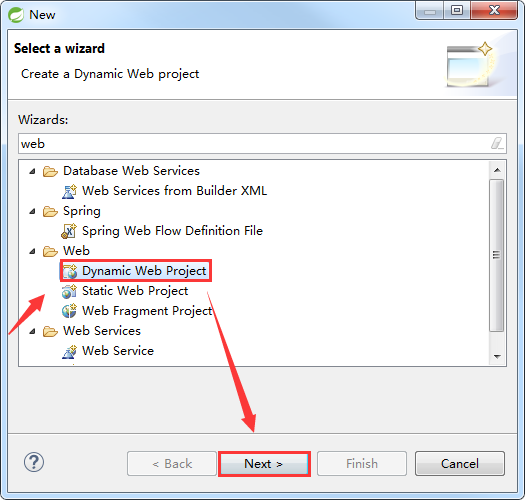
## 3、SSM 实操（SSM）

### 1）创建web 工程

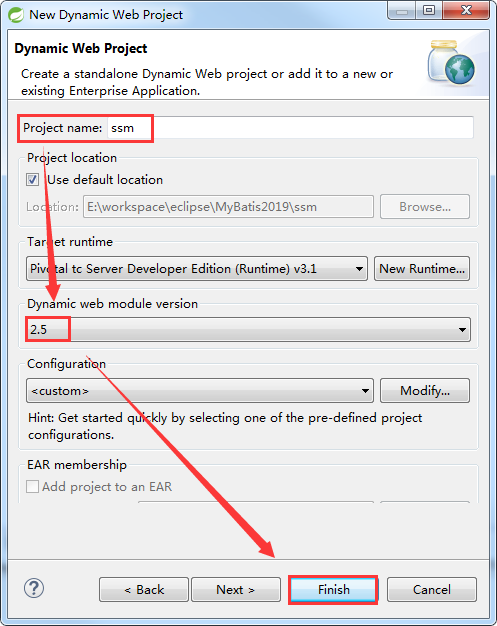
#### （1） File -> New -> Other



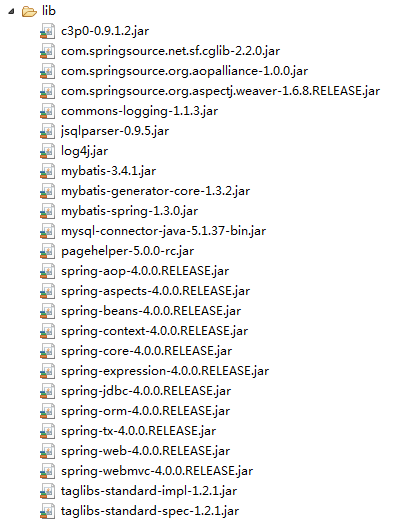
#### （2）Dynamic Web Project -> Next



#### （3）Project name -> Finish



### 2）导入jar包



#### （1）Spring jar 包

com.springsource.net.sf.cglib-2.2.0.jar

com.springsource.org.aopalliance-1.0.0.jar

com.springsource.org.aspectj.weaver-1.6.8.RELEASE.jar

spring-aop-4.0.0.RELEASE.jar

spring-aspects-4.0.0.RELEASE.jar

spring-beans-4.0.0.RELEASE.jar

spring-context-4.0.0.RELEASE.jar

spring-core-4.0.0.RELEASE.jar

spring-expression-4.0.0.RELEASE.jar

spring-jdbc-4.0.0.RELEASE.jar

spring-orm-4.0.0.RELEASE.jar

spring-tx-4.0.0.RELEASE.jar

commons-logging-1.1.3.jar

#### （2）Spring MVC jar 包

spring-web-4.0.0.RELEASE.jar

spring-webmvc-4.0.0.RELEASE.jar

#### （3）MyBatis jar包

mybatis-3.4.1.jar

#### （4）JSTL jar 包

taglibs-standard-impl-1.2.1.jar

taglibs-standard-spec-1.2.1.jar

#### （5）整合的适配包

mybatis-spring-1.3.0.jar

#### （6）驱动、日志、连接池包

c3p0-0.9.1.2.jar

log4j.jar

mysql-connector-java-5.1.37-bin.jar

#### （7）反向工程jar包

mybatis-generator-core-1.3.2.jar

#### （8）log4j.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">    <log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">    <appender name="STDOUT" class="org.apache.log4j.ConsoleAppender">  <param name="Encoding" value="UTF-8" />  <layout class="org.apache.log4j.PatternLayout">  <param name="ConversionPattern" value="%-5p %d{MM-dd HH:mm:ss,SSS} %m (%F:%L) \n" />  </layout>  </appender>  <logger name="java.sql">  <level value="debug" />  </logger>  <logger name="org.apache.ibatis">  <level value="info" />  </logger>  <root>  <level value="debug" />  <appender-ref ref="STDOUT" />  </root>  </log4j:configuration> |

#### （9）分页jar包

pagehelper-5.0.0-rc.jar

jsqlparser-0.9.5.jar

### 4）MyBatis 环境搭建

思想

MyBatis的全局配置文件

编写javaBean，Mapper接口，sql映射文件

方式

使用逆向工程方式实现

#### （1）添加逆向工程jar包

mybatis-generator-core-1.3.2.jar

#### （2）添加mbg.xml 配置文件

mbg.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>  <!-- targetRuntime: 指定生成的逆向工程的版本  MyBatis3: 生成带条件的增删改查.  MyBatis3Simple: 生成基本的增删改查.  -->  <context id="DB2Tables" targetRuntime="MyBatis3">    <!-- 数据库的连接 -->  <jdbcConnection driverClass="com.mysql.jdbc.Driver"  connectionURL="jdbc:mysql://localhost:3306/mybatis"  userId="root"  password="root">  </jdbcConnection>  <!-- 指定javabean的生成策略  targetPackage: 指定包  targetProject: 指定工程  -->  <javaModelGenerator targetPackage="com.alex.ssm.entity" targetProject=".\src">  <property name="enableSubPackages" value="true" />  <property name="trimStrings" value="true" />  </javaModelGenerator>    <!-- sql映射的生成策略 -->  <sqlMapGenerator targetPackage="com.alex.ssm.mapper" targetProject=".\resources">  <property name="enableSubPackages" value="true" />  </sqlMapGenerator>    <!-- mapper接口的生成策略 -->  <javaClientGenerator type="XMLMAPPER" targetPackage="com.alex.ssm.mapper" targetProject=".\src">  <property name="enableSubPackages" value="true" />  </javaClientGenerator>  <!-- 指定逆向分析的表 -->  <table tableName="tbl\_dept" domainObjectName="Department"></table>  <table tableName="tbl\_employee" domainObjectName="Employee"></table>    </context>  </generatorConfiguration> |

#### （3）创建包文件

com.alex.ssm.entity

com.alex.ssm.mapper

com.alex.ssm.test

#### （4）创建逆向工程方法

MyBatisTestMBG.java

|  |
| --- |
| package com.alex.mybatis.test;  import java.io.File;  import java.io.InputStream;  import java.util.ArrayList;  import java.util.List;  import org.apache.ibatis.io.Resources;  import org.apache.ibatis.session.SqlSessionFactory;  import org.apache.ibatis.session.SqlSessionFactoryBuilder;  import org.junit.Test;  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 MyBatisTestMBG {  @Test  public void createMBG() throws Exception {  List<String> warnings = new ArrayList<String>();  boolean overwrite = true;  File configFile = new File("mbg.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);  }  } |

#### （5）创建db.properties文件

|  |
| --- |
| jdbc.driver=com.mysql.jdbc.Driver  jdbc.url=jdbc:mysql://localhost:3306/mybatis?allowMultiQueries=true  jdbc.username=root  jdbc.password=root |

#### （6）mybatis-config.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE configuration  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-config.dtd">  <!-- MyBatis的全局配置文件 -->  <configuration>  <!-- properties:  引入外部化的配置文件  resource: 加载类路径下的资源文件  url: 加载网络路径或者是磁盘路径下的资源文件  -->  <properties resource="db.properties" ></properties>    <!-- settings:  包含很多重要的设置项，可以改变MyBatis框架的运行行为  setting:具体的一个设置项  name: 设置项的名称  value:设置项的取值  -->  <settings>  <!-- 映射下划线到驼峰命名 last\_name ==> lastName -->  <setting name="mapUnderscoreToCamelCase" value="true"/>  <!-- 开启延迟加载 -->  <setting name="lazyLoadingEnabled" value="true"/>  <!-- 指定加载的属性是按需加载. -->  <setting name="aggressiveLazyLoading" value="false"/>  <!-- 二级缓存 -->  <setting name="cacheEnabled" value="true"/>  </settings>    <!-- typeAliases  别名处理，给java类型取别名(简短)  typeAlias:给单个的java类型取别名  type:指定java类型(全类名)  alias:指定别名。默认的别名是类名的首字母小写. 别名不区分大小写.  package:批量取别名，给指定的包下的所有的类取默认的别名  name:指定包名  -->  <typeAliases>  <!-- <typeAlias type="com.alex.ssm.entity.Employee" alias="employee"/> -->  <package name="com.alex.ssm.entity"/>  </typeAliases>    <!-- environments: 环境们。 支持配置多个环境. 通过default来指定具体使用的环境.  environment: 具体的一个环境配置，必须包含transactionManager，dataSource。  id: 当前环境的唯一标识.  transactionManager: 事务管理器.  JDBC: JdbcTransactionFactory  将来事务管理会交给Spring的声明式事务.@Trancation  dataSource:数据源.  POOLED: PooledDataSourceFactory  将来数据源交给Spring管理,使用c3p0或者dbcp等  -->  <environments default="development">  <environment id="development">  <transactionManager type="JDBC" />  <dataSource type="POOLED">  <property name="driver" value="${jdbc.driver}" />  <property name="url" value="${jdbc.url}" />  <property name="username" value="${jdbc.username}" />  <property name="password" value="${jdbc.password}" />  </dataSource>  </environment>    <environment id="test">  <transactionManager type="JDBC" />  <dataSource type="POOLED">  <property name="driver" value="${jdbc.driver}" />  <property name="url" value="${jdbc.url}" />  <property name="username" value="${jdbc.username}" />  <property name="password" value="${jdbc.password}" />  </dataSource>  </environment>  </environments>    <!--  mappers:引入sql映射文件  mapper:引入单个的sql映射文件  package: 批量引入sql映射文件  要求: sql映射文件的名字与Mapper接口的名字一致.并且在同一目录下.  -->  <mappers>  <!-- <mapper resource="EmployeeMapper.xml" /> -->  <package name="com.alex.ssm.mapper"/>  </mappers>    </configuration> |

#### （7）mybatis.sql

|  |
| --- |
| /\*  Navicat MySQL Data Transfer  Source Server : localhost  Source Server Version : 50515  Source Host : localhost:3306  Source Database : mybatis  Target Server Type : MYSQL  Target Server Version : 50515  File Encoding : 65001  Date: 2019-10-16 14:38:06  \*/  SET FOREIGN\_KEY\_CHECKS=0;  -- ----------------------------  -- Table structure for tbl\_dept  -- ----------------------------  DROP TABLE IF EXISTS `tbl\_dept`;  CREATE TABLE `tbl\_dept` (  `id` int(11) NOT NULL AUTO\_INCREMENT,  `department\_name` varchar(50) DEFAULT NULL,  PRIMARY KEY (`id`)  ) ENGINE=InnoDB AUTO\_INCREMENT=5 DEFAULT CHARSET=utf8;  -- ----------------------------  -- Records of tbl\_dept  -- ----------------------------  INSERT INTO `tbl\_dept` VALUES ('1', '开发部');  INSERT INTO `tbl\_dept` VALUES ('2', '测试部');  INSERT INTO `tbl\_dept` VALUES ('3', '财务部');  INSERT INTO `tbl\_dept` VALUES ('4', '人事部');  -- ----------------------------  -- Table structure for tbl\_employee  -- ----------------------------  DROP TABLE IF EXISTS `tbl\_employee`;  CREATE TABLE `tbl\_employee` (  `id` int(11) NOT NULL AUTO\_INCREMENT,  `last\_name` varchar(50) DEFAULT NULL,  `email` varchar(50) DEFAULT NULL,  `gender` char(1) DEFAULT NULL,  `d\_id` int(11) DEFAULT NULL,  PRIMARY KEY (`id`),  KEY `fk\_emp\_dept` (`d\_id`),  CONSTRAINT `fk\_emp\_dept` FOREIGN KEY (`d\_id`) REFERENCES `tbl\_dept` (`id`)  ) ENGINE=InnoDB AUTO\_INCREMENT=1006 DEFAULT CHARSET=utf8;  -- ----------------------------  -- Records of tbl\_employee  -- ----------------------------  INSERT INTO `tbl\_employee` VALUES ('1001', 'Tom', 'Tom@aliyun.com', '1', '1');  INSERT INTO `tbl\_employee` VALUES ('1002', 'Rose', 'rose@aliyun.com', '0', '2');  INSERT INTO `tbl\_employee` VALUES ('1003', 'Alex', 'alex@qq.com', '1', '3');  INSERT INTO `tbl\_employee` VALUES ('1004', 'Thone', 'Thone@aliyun.com', '0', '4');  INSERT INTO `tbl\_employee` VALUES ('1005', 'Jerry', 'jerry@aliyun.com', '1', '2'); |

#### （8）创建Mybatis测试方法

|  |
| --- |
| package com.alex.mybatis.test;  import java.io.InputStream;  import java.util.List;  import org.apache.ibatis.io.Resources;  import org.apache.ibatis.session.SqlSession;  import org.apache.ibatis.session.SqlSessionFactory;  import org.apache.ibatis.session.SqlSessionFactoryBuilder;  import org.junit.Test;  import com.alex.ssm.entity.Employee;  import com.alex.ssm.entity.EmployeeExample;  import com.alex.ssm.entity.EmployeeExample.Criteria;  import com.alex.ssm.mapper.EmployeeMapper;  public class MyBatisTestMBG {  public SqlSessionFactory getSqlSessionFactory() throws Exception {  String resource = "mybatis-config.xml";  InputStream inputStream = Resources.getResourceAsStream(resource);  SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);  System.out.println(sqlSessionFactory);  return sqlSessionFactory;  }  // 根据id查询  @Test  public void testMBG01() throws Exception {  SqlSessionFactory ssf = getSqlSessionFactory();  SqlSession session = ssf.openSession();  try {  EmployeeMapper mapper = session.getMapper(EmployeeMapper.class);  Employee employee = mapper.selectByPrimaryKey(1001);  System.out.println(employee);  } finally {  session.close();  }  }  // 没有条件的情况，查询全部信息  @Test  public void testMBG02() throws Exception {  SqlSessionFactory ssf = getSqlSessionFactory();  SqlSession session = ssf.openSession();  try {  EmployeeMapper mapper = session.getMapper(EmployeeMapper.class);  List<Employee> emps = mapper.selectByExample(null);  for (Employee employee : emps) {  System.out.println(employee);  }  } finally {  session.close();  }  }  // 按照条件查询  @Test  public void testMBG03() throws Exception {  SqlSessionFactory ssf = getSqlSessionFactory();  SqlSession session = ssf.openSession();  try {  EmployeeMapper mapper = session.getMapper(EmployeeMapper.class);  // 查询员工名字中带有e字母 和员工性别是2的 或者email中带有y字母的  EmployeeExample example = new EmployeeExample();  Criteria criteria = example.createCriteria();  criteria.andLastNameLike("%e%");  criteria.andGenderEqualTo("0");  // or的条件需要重新使用一个Criteria对象  Criteria criteria2 = example.createCriteria();  criteria2.andEmailLike("%y%");  // 对于封装or条件的criteria，需要or到example  example.or(criteria2);  List<Employee> emps = mapper.selectByExample(example);  for (Employee employee : emps) {  System.out.println(employee);  }  } finally {  session.close();  }  }  } |

### 5）Spring SpringMVC 环境的搭建

#### （1）思想

web.xml中配置

SpringIOC容器启动的监听器

SpringMVC的核心控制器

REST过滤器

UTF-8编码

编写Spring的配置文件: applicationContext.xml

编写SpringMVC的配置文件: springmvc.xml

#### （2）web.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd" id="WebApp\_ID" version="2.5">  <display-name>ssm</display-name>    <!-- 实例化SpringIOC容器的监听器 -->  <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:spring-context.xml</param-value>  </context-param>  <listener>  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  </listener>    <!-- 配置utf-8编码 -->  <filter>  <filter-name>CharacterEncodingFilter</filter-name>  <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>  <init-param>  <param-name>encoding</param-name>  <param-value>UTF-8</param-value>  </init-param>  </filter>  <filter-mapping>  <filter-name>CharacterEncodingFilter</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping>    <!-- REST 过滤器 -->  <filter>  <filter-name>HiddenHttpMethodFilter</filter-name>  <filter-class>org.springframework.web.filter.HiddenHttpMethodFilter</filter-class>  </filter>  <filter-mapping>  <filter-name>HiddenHttpMethodFilter</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping>      <!-- Springmvc的核心控制器 -->  <servlet>  <servlet-name>springDispatcherServlet</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  <init-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:springmvc-context.xml</param-value>  </init-param>  <load-on-startup>1</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>springDispatcherServlet</servlet-name>  <url-pattern>/</url-pattern>  </servlet-mapping>    <welcome-file-list>  <welcome-file>index.jsp</welcome-file>  </welcome-file-list>    </web-app> |

#### （3）spring-context.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns:context="http://www.springframework.org/schema/context"  xmlns:tx="http://www.springframework.org/schema/tx"  xmlns:jdbc="http://www.springframework.org/schema/jdbc"  xmlns:aop="http://www.springframework.org/schema/aop"  xmlns:mvc="http://www.springframework.org/schema/mvc"  xsi:schemaLocation="http://www.springframework.org/schema/jdbc http://www.springframework.org/schema/jdbc/spring-jdbc-4.0.xsd  http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd  http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd  http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd  http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd  http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd">  <!-- 开启注解扫描 -->  <context:component-scan base-package="com.alex.ssm">  <!-- Springmvc管理的，Spring就不管理 -->  <context:exclude-filter type="annotation"  expression="org.springframework.stereotype.Controller" />  </context:component-scan>  <!-- 读取外部文件 -->  <context:property-placeholder location="classpath:/db.properties" />  <!-- C3P0 数据源 -->  <bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">  <property name="user" value="${jdbc.username}" />  <property name="password" value="${jdbc.password}" />  <property name="jdbcUrl" value="${jdbc.url}" />  <property name="driverClass" value="${jdbc.driver}" />  <property name="initialPoolSize" value="${jdbc.initialPoolSize}" />  <property name="minPoolSize" value="${jdbc.minPoolSize}" />  <property name="maxPoolSize" value="${jdbc.maxPoolSize}" />  <property name="acquireIncrement" value="${jdbc.acquireIncrement}" />  <property name="maxStatements" value="${jdbc.maxStatements}" />  <property name="maxStatementsPerConnection" value="${jdbc.maxStatementsPerConnection}" />  </bean>  <!-- 事务切面 -->  <bean id="transactionManager"  class="org.springframework.jdbc.datasource.DataSourceTransactionManager">  <property name="dataSource" ref="dataSource"></property>  </bean>  <!-- 开启基于注解的声明式事务 transaction-manager="transactionManager" 默认值,可以省略. -->  <tx:annotation-driven transaction-manager="transactionManager" />  </beans> |

#### （4）springmvc-context.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns:context="http://www.springframework.org/schema/context"  xmlns:mvc="http://www.springframework.org/schema/mvc"  xmlns:aop="http://www.springframework.org/schema/aop"  xmlns:jdbc="http://www.springframework.org/schema/jdbc"  xmlns:util="http://www.springframework.org/schema/util"  xmlns:tx="http://www.springframework.org/schema/tx"  xsi:schemaLocation="http://www.springframework.org/schema/jdbc http://www.springframework.org/schema/jdbc/spring-jdbc-4.0.xsd  http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd  http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd  http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd  http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd  http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd  http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-4.0.xsd">    <!-- 注解扫描 -->  <context:component-scan base-package="com.alex.ssm" use-default-filters="false">  <!-- 只扫描SpringMVC相关的组件 -->  <context:include-filter type="annotation"  expression="org.springframework.stereotype.Controller"/>  </context:component-scan>    <!-- 视图解析器 -->  <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">  <property name="prefix" value="/WEB-INF/views/"></property>  <property name="suffix" value=".jsp"></property>  </bean>    <mvc:default-servlet-handler/>  <mvc:annotation-driven/>    </beans> |

#### （5）db.properties

|  |
| --- |
| jdbc.driver=com.mysql.jdbc.Driver  jdbc.url=jdbc:mysql://localhost:3306/mybatis?allowMultiQueries=true  jdbc.username=root  jdbc.password=root  jdbc.initialPoolSize=30  jdbc.minPoolSize=10  jdbc.maxPoolSize=100  jdbc.acquireIncrement=5  jdbc.maxStatements=1000  jdbc.maxStatementsPerConnection=10 |

### 6）Spring整合MyBatis

配置创建SqlSession对象的SqlSessionFactoryBean

配置扫描所有的Mapper接口，批量生成代理实现类，交给Spring管理的，能够完成自动注入.

#### （1）spring-context.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns:context="http://www.springframework.org/schema/context"  xmlns:tx="http://www.springframework.org/schema/tx"  xmlns:jdbc="http://www.springframework.org/schema/jdbc"  xmlns:aop="http://www.springframework.org/schema/aop"  xmlns:mvc="http://www.springframework.org/schema/mvc"  xmlns:mybatis-spring="http://mybatis.org/schema/mybatis-spring"  xsi:schemaLocation="http://www.springframework.org/schema/jdbc http://www.springframework.org/schema/jdbc/spring-jdbc-4.0.xsd  http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd  http://mybatis.org/schema/mybatis-spring http://mybatis.org/schema/mybatis-spring-1.2.xsd  http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd  http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd  http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd  http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd">  <!-- 开启注解扫描 -->  <context:component-scan base-package="com.alex.ssm">  <!-- Springmvc管理的，Spring就不管理 -->  <context:exclude-filter type="annotation"  expression="org.springframework.stereotype.Controller" />  </context:component-scan>  <!-- 读取外部文件 -->  <context:property-placeholder location="classpath:/db.properties" />  <!-- C3P0 数据源 -->  <bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">  <property name="user" value="${jdbc.username}" />  <property name="password" value="${jdbc.password}" />  <property name="jdbcUrl" value="${jdbc.url}" />  <property name="driverClass" value="${jdbc.driver}" />  <property name="initialPoolSize" value="${jdbc.initialPoolSize}" />  <property name="minPoolSize" value="${jdbc.minPoolSize}" />  <property name="maxPoolSize" value="${jdbc.maxPoolSize}" />  <property name="acquireIncrement" value="${jdbc.acquireIncrement}" />  <property name="maxStatements" value="${jdbc.maxStatements}" />  <property name="maxStatementsPerConnection" value="${jdbc.maxStatementsPerConnection}" />  </bean>  <!-- 事务切面 -->  <bean id="transactionManager"  class="org.springframework.jdbc.datasource.DataSourceTransactionManager">  <property name="dataSource" ref="dataSource"></property>  </bean>  <!-- 开启基于注解的声明式事务 transaction-manager="transactionManager" 默认值,可以省略. -->  <tx:annotation-driven transaction-manager="transactionManager" />    <!-- 整合 MyBatis MyBatis的配置都可以在Spring里面配置，但是尽量把MyBatis独有的还是放在MyBatis配置文件里-->  <bean id="sqlSessionFactoryBean" class="org.mybatis.spring.SqlSessionFactoryBean">  <!-- 注入数据源 -->  <property name="dataSource" ref="dataSource" ></property>  <!-- 指定MyBatis的全局配置文件 -->  <property name="configLocation" value="classpath:mybatis-config.xml"></property>  <!-- 指定sql映射文件 -->  <property name="mapperLocations" value="classpath:com/alex/ssm/mapper/\*.xml"></property>  </bean>    <!-- 扫描所有mapper接口，批量生成代理实现类，交给Spring容器来管理 -->  <!-- 这是老版本的配置方式，没办法指定ID,ID默认是类的首字母小写 -->  <!--  <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">  <property name="basePackage" value="com.alex.ssm.mapper"></property>  </bean>  -->    <!-- 扫描所有mapper接口，批量生成代理实现类，交给Spring容器来管理 -->  <!-- 新方式 ， 两种方式，使用哪一种都可以-->  <mybatis-spring:scan base-package="com.alex.ssm.mapper"/>  </beans> |

#### （2）mybatis-config.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE configuration  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-config.dtd">  <!-- MyBatis的全局配置文件 -->  <configuration>    <settings>  <!-- 映射下划线到驼峰命名 last\_name ==> lastName -->  <setting name="mapUnderscoreToCamelCase" value="true"/>  <!-- 开启延迟加载 -->  <setting name="lazyLoadingEnabled" value="true"/>  <!-- 指定加载的属性是按需加载. -->  <setting name="aggressiveLazyLoading" value="false"/>  <!-- 二级缓存 -->  <setting name="cacheEnabled" value="true"/>  </settings>    <typeAliases>  <package name="com.alex.ssm.entity"/>  </typeAliases>    <plugins>  <plugin interceptor="com.github.pagehelper.PageInterceptor"></plugin>  </plugins>    </configuration> |

**尖叫提示：把交给Spring 管理的从MyBatis配置文件中删掉**

### 7）编码测试

完成员工的增删改查.

#### （1）EmployeeController.java

|  |
| --- |
| package com.alex.ssm.controller;  import java.util.List;  import java.util.Map;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Controller;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestMethod;  import com.alex.ssm.entity.Employee;  import com.alex.ssm.service.EmployeeService;  @Controller  public class EmployeeController {  @Autowired  private EmployeeService employeeService ;    @RequestMapping(value="/emps",method=RequestMethod.GET)  public String listEmps(Map<String,Object> map) {  List<Employee> emps = employeeService.getAllEmps();  map.put("emps",emps);  return "list";  }  } |

#### （2）EmployeeService.java

|  |
| --- |
| package com.alex.ssm.service;  import java.util.List;  import com.alex.ssm.entity.Employee;  public interface EmployeeService {  public List<Employee> getAllEmps();  } |

#### （3）EmployeeServiceImpl.Java

|  |
| --- |
| package com.alex.ssm.serviceimpl;  import java.util.List;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Service;  import com.alex.ssm.entity.Employee;  import com.alex.ssm.mapper.EmployeeMapper;  import com.alex.ssm.service.EmployeeService;  @Service  public class EmployeeServiceImpl implements EmployeeService{    @Autowired  private EmployeeMapper employeeMapper ;  @Override  public List<Employee> getAllEmps() {  List<Employee> example = employeeMapper.selectByExample(null);  return example;  }  } |

#### （4）index.jsp

|  |
| --- |
| <%@ page language="java" contentType="text/html; charset=UTF-8"  pageEncoding="UTF-8"%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>Insert title here</title>  </head>  <body>  <a href="${pageContext.request.contextPath}/emps">List ALL Emps</a>  </body>  </html> |

#### （5）list.jsp

|  |
| --- |
| <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>  <!-- 导入JSTL标签库 -->  <%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>Insert title here</title>  </head>  <body>  <h1 align="center" >员工信息列表</h1>  <table border="1px" width="70%" cellspacing="0px" align="center">  <tr>  <th>ID</th>  <th>LastName</th>  <th>Email</th>  <th>Gender</th>  <!-- <th>DeptName</th> -->  <th>Operation</th>  </tr>  <!-- items:指定要迭代的集合 var：代表当前迭代出的对象-->  <c:forEach items="${requestScope.emps }" var="emp">  <tr align="center">  <td>${emp.id }</td>  <td>${emp.lastName }</td>  <td>${emp.email }</td>  <td>${emp.gender==0?'女':'男' }</td>  <%-- <td>${emp.dept.departmentName}</td> --%>  <td>  <a href="#">Edit</a>  &nbsp;&nbsp;  <a href="#">Delete</a>  </td>  </tr>  </c:forEach>    </table>  </body>  </html> |