# MyBatis

尚硅谷Java研究院

版本：V1.0

## 第1章 MyBatis简介

### 1.1 MyBatis历史

1）MyBatis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。

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）。

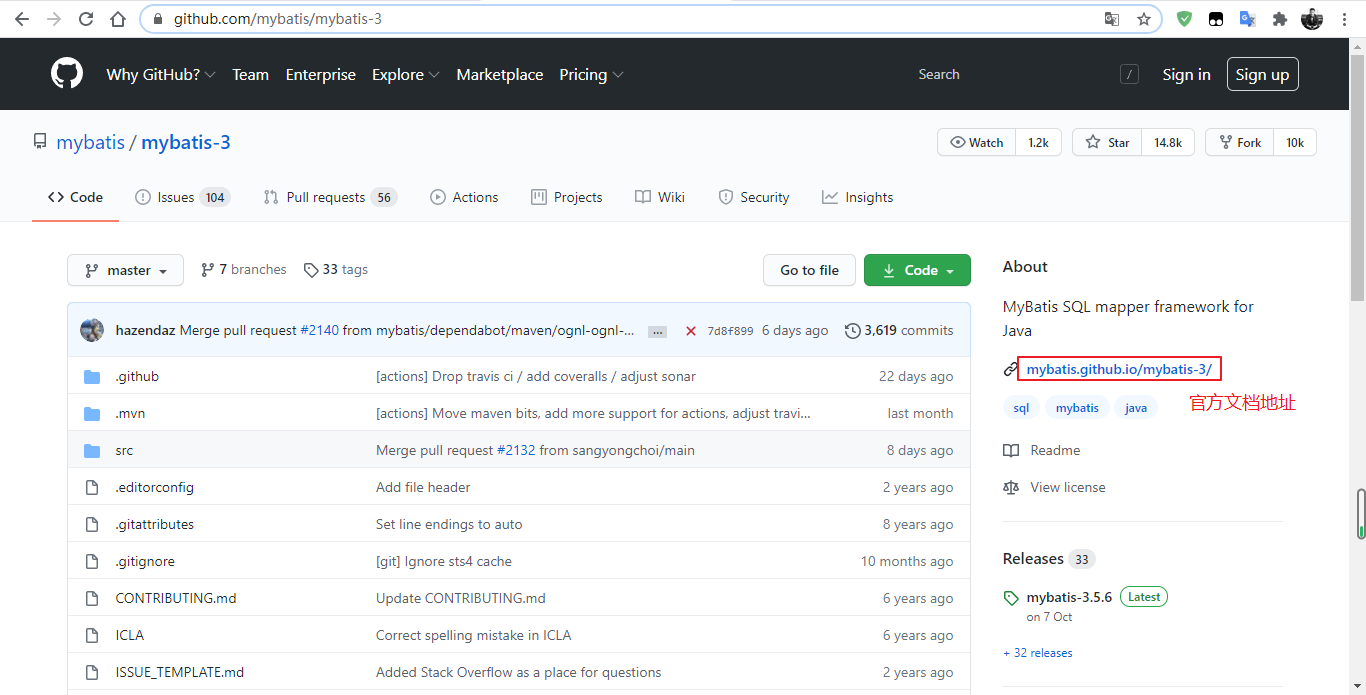
### 1.2 MyBatis简介

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 Ordinary Java Object，普通的Java对象）映射成数据库中的记录
4. Mybatis 是一个 半自动的ORM（Object Relation Mapping）框架

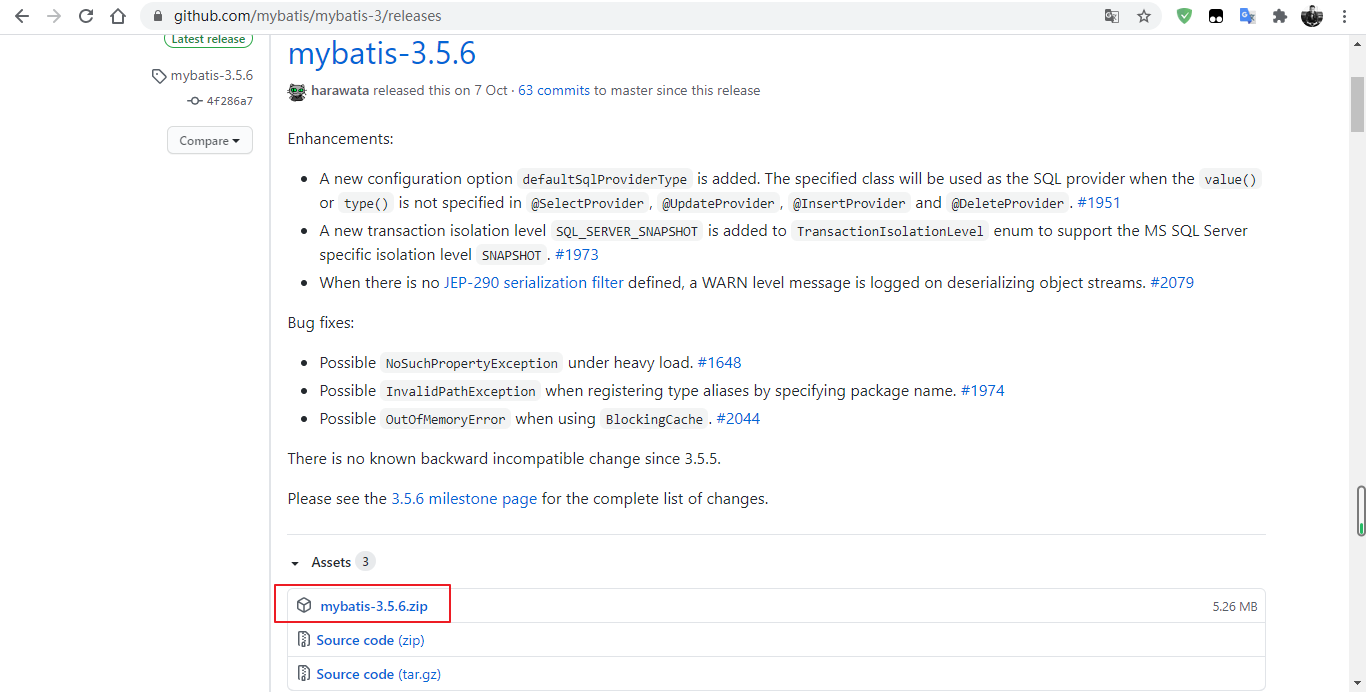
### 1.3 如何下载MyBatis

1. 下载网址

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







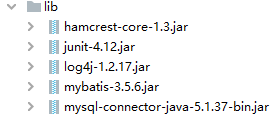
### 1.4 为什么要使用MyBatis – 现有持久化技术的对比

1. JDBC
2. SQL夹在Java代码块里，耦合度高导致硬编码内伤
3. 维护不易且实际开发需求中sql有变化，频繁修改的情况多见
4. Hibernate和JPA
5. 长难复杂SQL，对于Hibernate而言处理也不容易
6. 内部自动生产的SQL，不容易做特殊优化
7. 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
8. MyBatis
9. 对开发人员而言，核心sql还是需要自己优化
10. 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

## 第2章 HelloWorld

### 2.1 开发环境的准备

1. 导入以下jar包



1. 导入log4j 的配置文件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**> |

### 2.2 创建数据库和表

在MySQL中执行资料中的mybatis.sql文件

### 2.3 创建Employee类

|  |
| --- |
| **package** com.atguigu.mybatis.entities;  **public class** Employee {  **private** Integer **id**;  **private** String **lastName**;  **private** String **email**;  **private** Double **salary**;  **private** Integer **deptId**;   **public** Employee() {  }   **public** Employee(Integer id, String lastName, String email, Double salary, Integer deptId) {  **this**.**id** = id;  **this**.**lastName** = lastName;  **this**.**email** = email;  **this**.**salary** = salary;  **this**.**deptId** = deptId;  }   **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** Double getSalary() {  **return salary**;  }   **public void** setSalary(Double salary) {  **this**.**salary** = salary;  }   **public** Integer getDeptId() {  **return deptId**;  }   **public void** setDeptId(Integer deptId) {  **this**.**deptId** = deptId;  }   @Override  **public** String toString() {  **return "Employee{"** +  **"id="** + **id** +  **", lastName='"** + **lastName** + **'\''** +  **", email='"** + **email** + **'\''** +  **", salary="** + **salary** +  **", deptId="** + **deptId** +  **'}'**;  } } |

### 2.4 创建EmployeeMapper(Dao)接口

|  |
| --- |
| **package** com.atguigu.mybatis.dao;  **import** com.atguigu.mybatis.entities.Employee;  **public interface** EmployeeMapper {    Employee getEmployeeById(Integer id); } |

### 2.5 创建Mybatis的sql映射文件

1. 参考MyBatis的官方文档创建EmployeeMapper.xml映射文件
2. 完成两个绑定
   1. <mapper>标签中的namespace属性必须指定成Mapper接口的全类名
   2. Mapper映射文件中的增删改查标签的id必须指定成Mapper接口中的方法名

|  |
| --- |
| *<?***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"*>*** *<!--namespace属性：设置为接口的全类名-->* <**mapper namespace="com.atguigu.mybatis.dao.EmployeeMapper"**>   *<!--  id属性：设置为接口中的方法名  resultType属性：设置为方法的返回值的类型的全类名  -->* <**select id="getEmployeeById" resultType="com.atguigu.mybatis.entities.Employee"**>  select id,last\_name lastName,email,salary,dept\_id deptId  from employees  where id = #{id}  </**select**> </**mapper**> |

### 2.6 创建MyBatis的全局配置文件

参考MyBatis的官网文档创建mybatis-config.xml全局配置文件

|  |
| --- |
| *<?***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"*>*** <**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**>  *<!--设置Mapper映射文件（sql映射文件）-->* <**mappers**>  <**mapper resource="com/atguigu/mybatis/dao/EmployeeMapper.xml"**/>  </**mappers**> </**configuration**> |

### 2.7 测试

参考MyBatis的官方文件创建测试代码

|  |
| --- |
| **package** com.atguigu.mybatis.test;  **import** com.atguigu.mybatis.dao.EmployeeMapper; **import** com.atguigu.mybatis.entities.Employee; **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** java.io.IOException; **import** java.io.InputStream;  **public class** MyBatisTest {   */\*  测试HelloWorld  \*/* @Test  **public void** testHelloWorld() **throws** IOException {  *//设置MyBatis全局配置文件的路径* String resource = **"mybatis-config.xml"**;  *//读取类路径下的配置文件得到输入流* InputStream inputStream = Resources.*getResourceAsStream*(resource);  *//创建SqlSessionFactory对象* SqlSessionFactory sqlSessionFactory = **new** SqlSessionFactoryBuilder().build(inputStream);  *//获取SqlSession对象，相当于Connection对象* SqlSession sqlSession = sqlSessionFactory.openSession();  **try** {  *//获取Mapper代理对象* EmployeeMapper mapper = sqlSession.getMapper(EmployeeMapper.**class**);  *//调用EmployeeMapper中获取一个对象的方法* Employee employeeById = mapper.getEmployeeById(1);  System.***out***.println(employeeById);  } **finally** {  *//关闭sqlSession* sqlSession.close();  }  } } |

## 第3章 MyBatis全局配置文件

### 3.1 MyBatis全局配置文件简介

MyBatis 的配置文件包含了会深深影响 MyBatis 行为的设置和属性信息。 配置文档的顶层结构如下：



### 3.2 properties属性

1. 可外部配置且可动态替换的，既可以在典型的 Java 属性文件中配置，亦可通过 properties 元素的子元素来配置

|  |
| --- |
| <**properties**>  <**property name="username" value="hanzong"**/>  <**property name="password" value="123456"**/> </**properties**> |

1. 然而properties的作用并不单单是这样，你可以创建一个资源文件，名为jdbc.properties的文件,将四个连接字符串的数据在资源文件中通过键值 对(key=value)的方式放置，不要任何符号，一条占一行
   1. jdbc.properties

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

* 1. 通过properties标签引入jdbc.properties文件

|  |
| --- |
| *<!--  resource属性：引入类路径下的属性文件  url属性：引入网络或磁盘路径下的属性文件 -->* <**properties resource="jdbc.properties"**></**properties**> |

* 1. 在environment元素的dataSource元素中为其动态设置

|  |
| --- |
| <**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**> |

1. 如果一个属性在不只一个地方进行了配置，那么，MyBatis 将按照下面的顺序来加载：
   1. 首先读取在 properties 元素体内指定的属性。
   2. 然后根据 properties 元素中的 resource 属性读取类路径下属性文件，或根据 url 属性指定的路径读取属性文件，并覆盖之前读取过的同名属性。
   3. 最后读取作为方法参数传递的属性，并覆盖之前读取过的同名属性。

### settings设置

这是 MyBatis 中极为重要的调整设置，它们会改变 MyBatis 的运行时行为。 下表描述了设置中各项常见设置的含义、默认值等

|  |  |  |
| --- | --- | --- |
| 设置名 | 描述 | 默认值 |
| cacheEnabled | 缓存的全局开关 | true |
| lazyLoadingEnabled | 延迟加载的全局开关 | false |
| aggressiveLazyLoading | 开启时，任一方法的调用都会加载该对象的所有延迟加载属性。 否则，每个延迟加载属性会按需加载 | false （在 3.4.1 及之前的版本中默认为 true） |
| useGeneratedKeys | 允许 JDBC 支持自动生成主键，需要数据库驱动支持。如果设置为 true，将强制使用自动生成主键。尽管一些数据库驱动不支持此特性，但仍可正常工作（如 Derby）。 | false |
| autoMappingBehavior | 指定 MyBatis 应如何自动映射列到字段或属性。 NONE 表示关闭自动映射；PARTIAL 只会自动映射没有定义嵌套结果映射的字段。 FULL 会自动映射任何复杂的结果集（无论是否嵌套）。 | partial |
| mapUnderscoreToCamelCase | 是否开启驼峰命名自动映射，即从经典数据库列名 A\_COLUMN 映射到经典 Java 属性名 aColumn | false |

### 3.4 typeAliases 别名处理

1. 通过子标签typeAlias设置别名

|  |
| --- |
| <**typeAliases**>  *<!--  子标签typeAlias：用来给某些类指定别名  type属性：指定起别名的类的全类名  alias属性：指定别名，如果没有指定则是类命的首字母小写，但是别名大小写不敏感  -->* <**typeAlias type="com.atguigu.mybatis.entities.Employee" alias="employee"**></**typeAlias**> </**typeAliases**> |

1. 通过子标签package设置别名

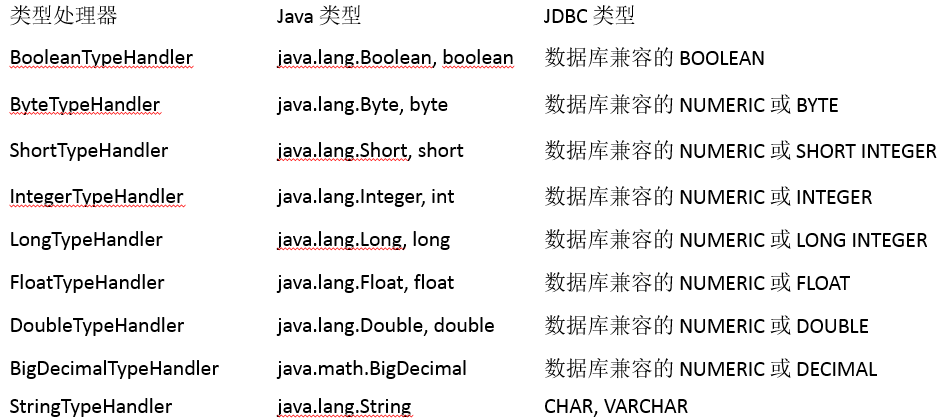
|  |
| --- |
| <**typeAliases**>  *<!--  子标签package：通过指定包名给包下所有的类起别名  name属性：指定包名  -->* <**package name="com.atguigu.mybatis.entities"**/> </**typeAliases**> |

1. MyBatis已经取好的别名



### 3.5 typeHandlers 类型处理器

1. 无论是 MyBatis 在预处理语句（PreparedStatement）中设置一个参数时，还是从结果集中取出一个值时， 都会用类型处理器将获取的值以合适的方式转换成 Java 类型
2. MyBatis中提供的类型处理器:



1. 日期和时间的处理，JDK1.8以前一直是个头疼的问题。我们通常使用JSR310规范领导者Stephen Colebourne创建的Joda-Time来操作。1.8已经实现全部的JSR310规范了
2. 日期时间处理上，我们可以使用MyBatis基于JSR310（Date and Time API）编写的各种日期时间类型处理器。
3. MyBatis3.4以前的版本需要我们手动注册这些处理器，以后的版本都是自动注册的,如需注册，需要下载mybatistypehandlers-jsr310,并通过如下方式注册



1. 自定义类型转换器
2. 我们可以重写类型处理器或创建自己的类型处理器来处理不支持的或非标准的类 型
3. 步骤

* 实现org.apache.ibatis.type.TypeHandler接口或者继承org.apache.ibatis.type.BaseTypeHandler
* 指定其映射某个JDBC类型（可选操作）
* 在mybatis全局配置文件中注册

### 3.6 plugins 插件机制

1. 插件是MyBatis提供的一个非常强大的机制，我们可以通过插件来修改MyBatis的一些核心行为。插件通过动态代理机制，可以介入四大对象的任何一个方法的执行
2. 四大对象:

Executor (update, query, flushStatements, commit, rollback, getTransaction, close, isClosed)

ParameterHandler (getParameterObject, setParameters)

ResultSetHandler (handleResultSets, handleOutputParameters)

StatementHandler (prepare, parameterize, batch, update, query)

### 3.7 environments 环境配置

1. MyBatis可以配置多种环境，比如开发、测试和生产环境需要有不同的配置
2. 每种环境使用一个environment标签进行配置并通过id属性指定唯一标识符
3. 可以通过environments标签中的default属性指定一个环境的标识符来快速的切换环境
4. environment-指定具体环境
   1. id：指定当前环境的唯一标识
   2. transactionManager和dataSource都必须有

|  |
| --- |
| <**environments default="development"**>  *<!--开发环境-->* <**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**>   *<!--生产环境-->* <**environment id="online"**>  <**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**> |

1. transactionManager

type： JDBC | MANAGED | 自定义

JDBC：使用了 JDBC 的提交和回滚设置，依赖于从数据源得到的连接来管理事务范 围。 JdbcTransactionFactory

MANAGED：不提交或回滚一个连接、让容器来管理事务的整个生命周期（比如 JEE 应用服务器的上下文）。 ManagedTransactionFactory

自定义：实现TransactionFactory接口，type=全类名/别名

1. dataSource

type： UNPOOLED | POOLED | JNDI | 自定义

UNPOOLED：不使用连接池， UnpooledDataSourceFactory

POOLED：使用连接池， PooledDataSourceFactory

JNDI： 在EJB 或应用服务器这类容器中查找指定的数据源

自定义：实现DataSourceFactory接口，定义数据源的获取方式。

1. 可以通过org.apache.ibatis.session. Configuration查询上述信息
2. 实际开发中我们使用Spring管理数据源，并进行事务控制的配置来覆盖上述配置

### 3.8 databaseIdProvider数据库厂商标识

1. MyBatis 可以根据不同的数据库厂商执行不同的语句

|  |
| --- |
| *<!--设置数据库厂商标识-->* <**databaseIdProvider type="DB\_VENDOR"**>  <**property name="MySQL" value="mysql"**/>  <**property name="Oracle" value="oracle"**/> </**databaseIdProvider**> |

1. type： DB\_VENDOR, 使用MyBatis提供的VendorDatabaseIdProvider解析数据库厂商标识。也可以实现DatabaseIdProvider接口来自定义。

VendorDatabaseIdProvider会通过 DatabaseMetaData#getDatabaseProductName() 返回的字符串进行设置。由于通常情况下这个字符串都非常长而且相同产品的不同版本会返回不同的值，所以最好通过设置属性别名来使其变短。

name：数据库厂商标识

value：为标识起一个别名，方便SQL语句使用databaseId属性引用

1. 配置了databaseIdProvider后，在SQL映射文件中的增删改查标签中使用databaseId

来指定数据库标识的别名

|  |
| --- |
| <**select id="getEmployeeById" resultType="employee" databaseId="mysql"**>  select id,last\_name,email,salary,dept\_id  from employees  where id = #{id} </**select**> |

1. MyBatis匹配规则如下:
2. 如果没有配置databaseIdProvider标签，那么databaseId=null
3. 如果配置了databaseIdProvider标签，使用标签配置的name去匹配数据库信息，匹配上则设置databaseId=配置指定的值，否则依旧为null
4. 如果databaseId不为null，他只会找到配置databaseId的sql语句
5. MyBatis 会加载不带 databaseId 属性和带有匹配当前数据库databaseId 属性的所有语句。如果同时找到带有 databaseId 和不带databaseId 的相同语句，则后者会被舍弃。

### 3.9 mappers 映射器

1. 用来在mybatis初始化的时候，告诉mybatis需要引入哪些Mapper映射文件
2. mapper标签：逐个注册SQL映射文件
   1. resource属性：引入类路径下的文件
   2. url 属性：引入网络路径或者是磁盘路径下的文件
   3. class属性：设置Mapper接口的全类名
      1. 如果有SQL映射文件，要求Mapper接口与 SQL映射文件同名同包。
      2. 如果没有SQL映射文件 ，使用注解在接口的方法上写SQL语句。

|  |
| --- |
| <**mappers**>  <**mapper resource="com/atguigu/mybatis/dao/EmployeeMapper.xml"**/> </**mappers**> |

1. package标签：批量注册SQL映射文件
   1. name 属性：设置Mapper接口的全类名
      1. 如果有SQL映射文件 ，要求Mapper接口与 SQL映射文件同名同包。
      2. 如果没有SQL映射文件，使用注解在接口的方法上写SQL语句。

|  |
| --- |
| <**mappers**>  <**package name="com.atguigu.mybatis.dao"**/> </**mappers**> |

## 第4章 MyBatis 映射文件

### 4.1 Mybatis映射文件简介

1. MyBatis 的真正强大在于它的映射语句，也是它的魔力所在。由于它的异常强大，映射器的 XML 文件就显得相对简单。如果拿它跟具有相同功能的 JDBC 代码进行对比，你会立即发现省掉了将近 95% 的代码。MyBatis 就是针对 SQL 构建的，并且比普通的方法做的更好。
2. SQL 映射文件只有很少的几个顶级元素（按照应被定义的顺序列出）：
   * cache – 该命名空间的缓存配置。
   * cache-ref – 引用其它命名空间的缓存配置。
   * resultMap – 描述如何从数据库结果集中加载对象，是最复杂也是最强大的元素。
   * ~~parameterMap – 老式风格的参数映射。此元素已被废弃，并可能在将来被移除！请使用行内参数映射。文档中不会介绍此元素。~~
   * sql – 可被其它语句引用的可重用语句块。
   * 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。

### 4.2 Mybatis映射CRUD

#### 4.2.1 select

1. Mapper接口方法

|  |
| --- |
| Employee getEmployeeById(Integer id); |

1. Mapper映射文件

|  |
| --- |
| <**select id="getEmployeeById" resultType="employee" databaseId="mysql"**>  select id,last\_name,email,salary,dept\_id  from employees  where id = #{id} </**select**> |

1. 属性说明
   1. id：指定接口中的方法名。
   2. parameterType：设置传入的参数的类型，也可以不指定，因为 MyBatis 可以通过类型处理器（TypeHandler）推断出具体传入语句的参数。
   3. resultType：设置方法的返回值的类型。注意，如果返回的是集合，那应该设置为集合包含的类型，而不是集合本身的类型。 resultType 和 resultMap 之间只能同时使用一个。
   4. resultMap：设置对外部 resultMap高级结果集标签 的引用。
   5. flushCache：将其设置为 true 后，只要语句被调用，都会导致本地缓存和二级缓存被清空，默认值：false。
   6. useCache：将其设置为 true 后，将会导致本条语句的结果被二级缓存缓存起来，默认值：对 select 元素为 true。
   7. databaseId：如果配置了数据库厂商标识（databaseIdProvider），MyBatis 会加载所有不带 databaseId 或匹配当前 databaseId 的语句；如果带和不带的语句都有，则不带的会被忽略。

#### 4.2.2 insert

1. Mapper接口方法

|  |
| --- |
| **void** addEmployee(Employee employee); |

1. Mapper映射文件

|  |
| --- |
| *<!--  parameterType属性：设置请求参数的类型的全类名，该属性也可以不指定，  MyBatis 可以通过类型处理器推断出具体传入语句的参数 -->* <**insert id="addEmployee" parameterType="com.atguigu.mybatis.entities.Employee"**>  insert into employees(last\_name,email,salary,dept\_id)  values(#{lastName},#{email},#{salary},#{deptId}) </**insert**> |

#### 4.2.3 update

1. Mapper接口方法

|  |
| --- |
| **void** updateEmployee(Employee employee); |

1. Mapper映射文件

|  |
| --- |
| <**update id="updateEmployee"**>  update employees set  last\_name=#{lastName},  email=#{email},  salary=#{salary},  dept\_id=#{deptId}  where id=#{id} </**update**> |

#### 4.2.4 delete

1. Mapper接口方法

|  |
| --- |
| **void** deleteEmployeeById(Integer id); |

1. Mapper映射文件

|  |
| --- |
| <**delete id="deleteEmployeeById"**>  delete from employees where id = #{id} </**delete**> |

1. insert、update、delete标签属性说明
   1. flushCache：将其设置为 true 后，只要语句被调用，都会导致本地缓存和二级缓存被清空，默认值：（对 insert、update 和 delete 语句）true。
   2. useGeneratedKeys：（仅适用于 insert 和 update）这会令 MyBatis 使用 JDBC 的 getGeneratedKeys 方法来取出由数据库内部生成的主键（比如：像 MySQL 和 SQL Server 这样的关系型数据库管理系统的自动递增字段），默认值：false。
   3. keyProperty：（仅适用于 insert 和 update）指定能够唯一识别对象的属性，MyBatis 会使用 getGeneratedKeys 的返回值或 insert 语句的 selectKey 子元素设置它的值，默认值：未设置（unset）。如果生成列不止一个，可以用逗号分隔多个属性名称。
   4. keyColumn：（仅适用于 insert 和 update）设置生成键值在表中的列名，在某些数据库（像 PostgreSQL）中，当主键列不是表中的第一列的时候，是必须设置的。如果生成列不止一个，可以用逗号分隔多个属性名称。

### 4.3 主键生成方式、获取主键值

#### 4.3.1 主键生成方式

1. 支持主键自增，例如MySQL数据库
2. 不支持主键自增，例如Oracle数据库

#### 4.3.2 获取主键值

1. 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。

|  |
| --- |
| <**insert id="addEmployee" useGeneratedKeys="true" keyProperty="id"**>  insert into employees(last\_name,email,salary,dept\_id)  values(#{lastName},#{email},#{salary},#{deptId}) </**insert**> |

1. 而对于不支持自增型主键的数据库（例如 Oracle），则可以使用 selectKey 子元素：selectKey 元素将会首先运行，id 会被设置，然后插入语句会被调用。

|  |
| --- |
| <**insert id="addEmployee" databaseId="oracle"**>  <**selectKey order="BEFORE" keyProperty="id" resultType="integer"**>  select employee\_seq.nextval from dual  </**selectKey**>  insert into oracle\_employees(id,last\_name,email,salary,dept\_id)  values(#{id},#{lastName},#{email},#{salary},#{deptId}) </**insert**> |

或者是

|  |
| --- |
| <**insert id="addEmployee" databaseId="oracle"**>  <**selectKey order="AFTER" keyProperty="id" resultType="integer"**>  select employee\_seq.currval from dual  </**selectKey**>  insert into oracle\_employees(id,last\_name,email,salary,dept\_id)  values(employee\_seq.nextval,#{lastName},#{email},#{salary},#{deptId}) </**insert**> |

### 4.4 参数传递

#### 4.4.1 参数传递的方式

1. 单个普通类型参数

可以接受基本类型，包装类型，字符串类型等。这种情况MyBatis可直接使用这个参数，不需要经过任何处理。

1. 多个参数

任意多个参数，都会被MyBatis重新包装成一个Map传入。Map的key是param1，param2，或者arg0，arg1…，值就是参数的值。

1. 命名参数

为参数使用@Param起一个名字，MyBatis就会将这些参数封装进map中，key就是我们自己指定的名字。

1. POJO

当这些参数属于我们业务POJO时，我们直接传递POJO。

1. Map

我们也可以封装多个参数为map，直接传递。

1. Collection/Array

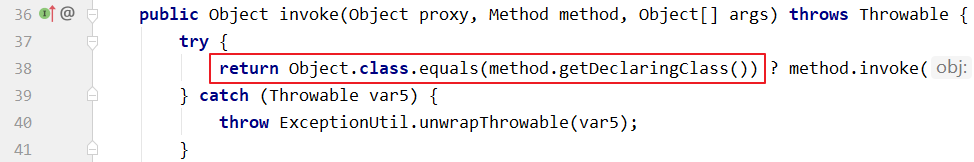
会被MyBatis封装成一个map传入, Collection对应的key是collection,Array对应的key是array. 如果确定是List集合，key还可以是list。

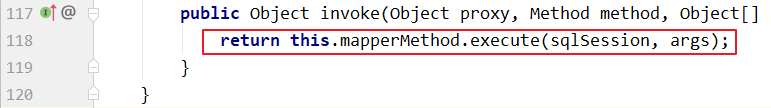
#### 4.4.2 参数传递源码分析

1. 以命名参数为例:

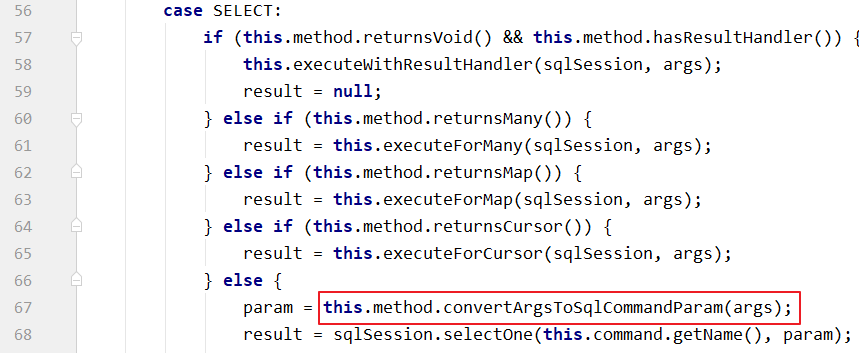
|  |
| --- |
| Employee getEmployeeByLastNameAndEmail(@Param(**"lastName"**) String lastName , @Param(**"email"**) String email); |

1. 底层源码:
   1. 在MapperProxy<T>类的38行和118行

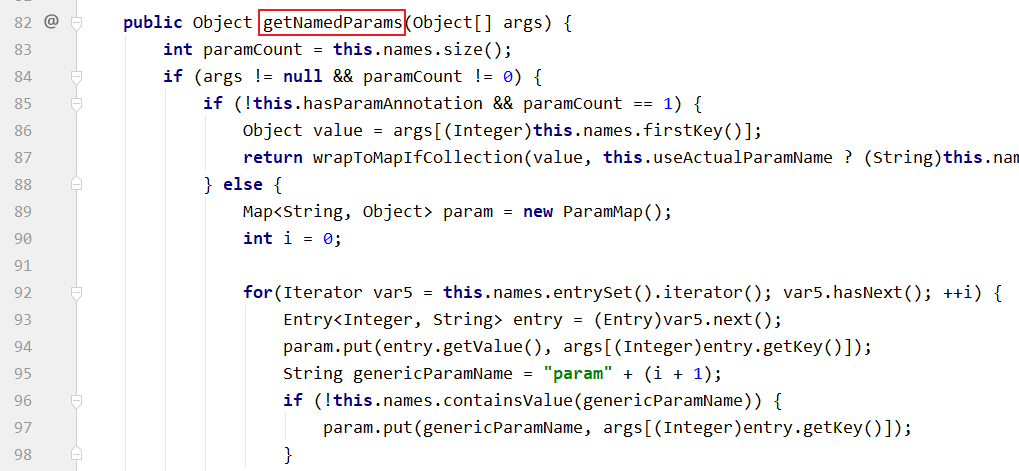




* 1. 在MapperMethod类的67行



* 1. 在ParamNameResolver类的82行



#### 4.4.3 参数处理

1. 参数位置支持的属性:

javaType、jdbcType、mode、numericScale、resultMap、typeHandler、jdbcTypeName、expression

1. 实际上通常被设置的是：可能为空的列名指定 jdbcType ,例如:

|  |
| --- |
| <**insert id="addEmployee"**>  insert into oracle\_employees(last\_name,email,salary,dept\_id)  values(#{lastName,jdbcType=null},#{email},#{salary},#{deptId}) </**insert**> |

#### 4.4.4 参数的获取方式

1. #{key}：获取参数的值，预编译到SQL中。安全。
2. ${key}：获取参数的值，拼接到SQL中。有SQL注入问题。

### 4.5 select查询的几种情况

1. 查询单行数据返回单个对象

|  |
| --- |
| Employee getEmployeeById(Integer id); |

1. 查询多行数据返回对象的集合

|  |
| --- |
| List<Employee> getEmployees(); |

1. 查询单行数据返回Map集合
   1. 数据库中的字段名为key
   2. 数据库中的字段值为value

|  |
| --- |
| Map<String,Employee> getEmployeeByIdReturnMap(Integer id ); |

1. 查询多行数据返回Map集合

|  |
| --- |
| @MapKey(**"id"**) *//指定使用数据库中的那个字段作为map的key* Map<Integer,Employee> getEmployeesReturnMap(); |

### 4.6 resultType自动映射

1. autoMappingBehavior默认是PARTIAL，开启自动映射的功能。唯一的要求是结果集列名和POJO属性名一致。
2. 如果autoMappingBehavior设置为null则会取消自动映射。
3. 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。

### 4.7 resultMap自定义映射

1. resultMap，实现高级结果集映射
2. id ：用于完成主键值的映射
3. result ：用于完成普通列的映射
4. association ：一个复杂的类型关联;许多结果将包成这种类型
5. collection ： 复杂类型的集

#### 4.7.1 id&result 中的属性

|  |  |
| --- | --- |
| 属性 | 描述 |
| property | 映射到列结果的字段或属性。比如，你可以这样映射一些简单的东西：“username”，或者映射到一些复杂的东西上：“address.street.number” |
| column | 数据库中的列名，或者是列的别名。一般情况下，这和传递给 resultSet.getString(columnName) 方法的参数一样 |
| javaType | 一个 Java 类的全限定名，或一个类型别名，如果你映射到一个 JavaBean，MyBatis 通常可以推断类型。 |
| jdbcType | JDBC 类型 |
| typeHandler | 使用这个属性，你可以覆盖默认的类型处理器。 这个属性值是一个类型处理器实现类的全限定名，或者是类型别名 |

|  |
| --- |
| *<!--  resultMap属性：引用高级结果集映射，设置为resultMap标签的id属性值 -->*  <**select id="getEmployeeById" resultMap="myResult"**>  select id,last\_name,email,salary,dept\_id  from employees  where id = #{id} </**select**>  <**resultMap id="myResult" type="com.atguigu.mybatis.entities.Employee"**>  <**id property="id" column="id"**></**id**>  <**result property="lastName" column="last\_name"**></**result**>  <**result property="email" column="email"**></**result**>  <**result property="salary" column="salary"**></**result**>  <**result property="deptId" column="dept\_id"**></**result**> </**resultMap**> |

#### 4.7.2 association

1. POJO中的属性可能会是一个对象,我们可以使用联合查询，并以级联属性的方式封装对象。使用association标签定义对象的封装规则。
2. 修改Employee类

|  |
| --- |
| **package** com.atguigu.mybatis.entities;  **public class** Employee {  **private** Integer **id**;  **private** String **lastName**;  **private** String **email**;  **private** Double **salary**;  **private** Department **dept**;   **public** Employee() {  }   **public** Employee(Integer id, String lastName, String email, Double salary, Department dept) {  **this**.**id** = id;  **this**.**lastName** = lastName;  **this**.**email** = email;  **this**.**salary** = salary;  **this**.**dept** = dept;  }   **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** Double getSalary() {  **return salary**;  }   **public void** setSalary(Double salary) {  **this**.**salary** = salary;  }   **public** Department getDept() {  **return dept**;  }   **public void** setDept(Department dept) {  **this**.**dept** = dept;  }   @Override  **public** String toString() {  **return "Employee{"** +  **"id="** + **id** +  **", lastName='"** + **lastName** + **'\''** +  **", email='"** + **email** + **'\''** +  **", salary="** + **salary** +  **", dept="** + **dept** +  **'}'**;  } } |

|  |
| --- |
| **package** com.atguigu.mybatis.entities;  **public class** Department {  **private** Integer **id**;  **private** String **name**;   **public** Department() {  }   **public** Department(Integer id, String name) {  **this**.**id** = id;  **this**.**name** = name;  }   **public** Integer getId() {  **return id**;  }   **public void** setId(Integer id) {  **this**.**id** = id;  }   **public** String getName() {  **return name**;  }   **public void** setName(String name) {  **this**.**name** = name;  }   @Override  **public** String toString() {  **return "Department{"** +  **"id="** + **id** +  **", name='"** + **name** + **'\''** +  **'}'**;  } } |

1. 使用级联的方式给部门属性赋值

|  |
| --- |
| <**select id="getEmployeeAndDept" resultMap="myEmp"**>  SELECT e.\*,d.id d\_id,d.name d\_name  FROM employees e  LEFT JOIN departments d  ON e.dept\_id = d.id  WHERE e.id = #{id}; </**select**>  <**resultMap id="myEmp" type="com.atguigu.mybatis.entities.Employee"**>  <**id property="id" column="id"**></**id**>  <**result property="lastName" column="last\_name"**></**result**>  <**result property="email" column="email"**></**result**>  <**result property="salary" column="salary"**></**result**>  *<!--通过级联方式给部门属性赋值-->* <**result property="dept.id" column="d\_id"**></**result**>  <**result property="dept.name" column="d\_name"**></**result**> </**resultMap**> |

1. 通过association标签给部门属性赋值

|  |
| --- |
| <**resultMap id="myEmp2" type="com.atguigu.mybatis.entities.Employee"**>  <**id property="id" column="id"**></**id**>  <**result property="lastName" column="last\_name"**></**result**>  <**result property="email" column="email"**></**result**>  <**result property="salary" column="salary"**></**result**>  *<!--通过association标签给部门属性赋值  property属性：指定Employee中部门的属性名  javaType属性：指定属性的类型  -->* <**association property="dept" javaType="com.atguigu.mybatis.entities.Department"**>  <**id property="id" column="d\_id"**></**id**>  <**result property="name" column="d\_name"**></**result**>  </**association**> </**resultMap**> |

#### 4.7.3 association 分步查询

1. 实际的开发中，对于每个实体类都应该有具体的增删改查方法，也就是DAO层， 因此，对于查询员工信息并且将对应的部门信息也查询出来的需求，就可以通过分步的方式完成查询。
2. 先通过员工的id查询员工信息
3. 再通过查询出来的员工信息中的外键(部门id)查询对应的部门信息
4. 将部门信息设置到员工中
5. 创建DeparmentMapper接口

|  |
| --- |
| **public interface** DepartmentMapper {   Department getDepartmentById(Integer id);   } |

1. 创建Mapper映射文件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"*>*** *<!--namespace属性：设置接口的全类名-->* <**mapper namespace="com.atguigu.mybatis.dao.DepartmentMapper"**>   <**select id="getDepartmentById" resultType="com.atguigu.mybatis.entities.Department"**>  select id,name   from departments  where id = #{id}  </**select**>   </**mapper**> |

1. 在EmployeeMapper中添加一个方法

|  |
| --- |
| Employee getEmployeeAndDeptByStep(Integer id); |

1. 在EmployeeMapper.xml的映射文件中自定义高级映射实现分步查询

|  |
| --- |
| <**select id="getEmployeeAndDeptByStep" resultMap="myEmp3"**>  select id,last\_name,email,salary,dept\_id  from employees  where id = #{id} </**select**>  *<!--  分步查询：  1.根据员工的id查询员工信息  2.根据员工的部门id查询部分信息  3.将部门信息设置到员工中 -->* <**resultMap id="myEmp3" type="com.atguigu.mybatis.entities.Employee"**>  <**id property="id" column="id"**></**id**>  <**result property="lastName" column="last\_name"**></**result**>  <**result property="email" column="email"**></**result**>  <**result property="salary" column="salary"**></**result**>  *<!--通过association标签分步查询给部门属性赋值  property属性：指定Employee中部门的属性名  select属性：指定调用那个接口的那个方法查询部门信息  column属性：指定将那个字段的值传入到select中调用的方法中  -->* <**association property="dept" select="com.atguigu.mybatis.dao.DepartmentMapper.getDepartmentById"  column="dept\_id"**></**association**> </**resultMap** |

#### 4.7.4 association 分步查询使用延迟加载

1. 在分步查询的基础上，可以使用延迟加载来提升查询的效率，只需要在全局的

settings中进行如下的配置:

|  |
| --- |
| <**settings**>  *<!-- 开启延迟加载 -->* <**setting name="lazyLoadingEnabled" value="true"**/>  *<!-- 设置加载的数据是按需加载-->* <**setting name="aggressiveLazyLoading" value="false"**/> </**settings**> |

1. 在association标签中也可以设置延迟加载，将覆盖全局配置

|  |
| --- |
| <**resultMap id="myEmp3" type="com.atguigu.mybatis.entities.Employee"**>  <**id property="id" column="id"**></**id**>  <**result property="lastName" column="last\_name"**></**result**>  <**result property="email" column="email"**></**result**>  <**result property="salary" column="salary"**></**result**>  *<!--通过association标签分步查询给部门属性赋值  property属性：指定Employee中部门的属性名  select属性：指定调用那个接口的那个方法查询部门信息  column属性：指定将那个字段的值传入到select中调用的方法中  fetchType属性：是否使用延迟加载（全局开关需要打开）  lazy： 使用延迟加载  eager：关闭延迟加载  -->* <**association property="dept" select="com.atguigu.mybatis.dao.DepartmentMapper.getDepartmentById"  column="dept\_id" fetchType="lazy"**></**association**> </**resultMap**> |

#### 4.7.5 collection

1. POJO中的属性可能会是一个集合对象，我们可以使用联合查询，并以级联属性的方式封装对象，使用collection标签定义对象的封装规则

|  |
| --- |
| **public class** Department {  **private** Integer **id**;  **private** String **name**;  **private** List<Employee> **emps**; |

1. collection

|  |
| --- |
| <**select id="getDepartmentAndEmps" resultMap="myDept"**>  SELECT d.id d\_id,d.name d\_name,e.\*  FROM departments d  LEFT JOIN employees e  ON d.id = e.dept\_id  WHERE d.id = #{id} </**select**>  <**resultMap id="myDept" type="com.atguigu.mybatis.entities.Department"**>  <**id property="id" column="d\_id"**></**id**>  <**result property="name" column="d\_name"**></**result**>  *<!--  property属性：集合属性的属性名  ofType属性：集合中元素的类型  -->* <**collection property="emps" ofType="com.atguigu.mybatis.entities.Employee"**>  <**id property="id" column="id"**></**id**>  <**result property="lastName" column="last\_name"**></**result**>  <**result property="email" column="email"**></**result**>  <**result property="salary" column="salary"**></**result**>  </**collection**> </**resultMap**> |

#### 4.7.6 collection 分步查询

1. 实际的开发中，对于每个实体类都应该有具体的增删改查方法，也就是DAO层， 因此，对于查询部门信息并且将对应的所有的员工信息也查询出来的需求，就可以通过分步的方式完成查询。
2. 先通过部门的id查询部门信息
3. 再通过部门id作为员工的外键查询对应的员工信息.。
4. 将所有员工设置到部门中
5. 在EmployeeMapper中添加一个方法

|  |
| --- |
| List<Employee> getEmployeesByDeptId(Integer deptId); |

1. 在EmployeeMapper.xml文件中添加映射信息

|  |
| --- |
| <**select id="getEmployeesByDeptId" resultType="com.atguigu.mybatis.entities.Employee"**>  select id,last\_name,email,salary,dept\_id  from employees  where dept\_id = #{deptId} </**select**> |

1. 在DepartmentMapper中添加一个方法

|  |
| --- |
| Department getDepartmentAndEmpsByStep(Integer id); |

1. 在DepartmentMapper.xml的映射文件中自定义高级映射实现分步查询

|  |
| --- |
| <**select id="getDepartmentAndEmpsByStep" resultMap="myDept2"**>  select id,name  from departments  where id = #{id} </**select**>  *<!--  分步查询：  1.根据部门的id查询部门信息  2.根据部门id查询出该部门下所有的员工  3.将员工设置到部门中 -->* <**resultMap id="myDept2" type="com.atguigu.mybatis.entities.Department"**>  <**id property="id" column="id"**></**id**>  <**result property="name" column="name"**></**result**>  *<!--  property属性：集合属性的属性名  select属性：调用那个接口的那个方法查询员工信息  column属性：将那个字段的值传入到select属性调用的方法中  -->* <**collection property="emps" select="com.atguigu.mybatis.dao.EmployeeMapper.getEmployeesByDeptId"  column="id"**></**collection**> </**resultMap**> |

#### 4.7.7 collection 分步查询使用延迟加载

|  |
| --- |
| <**collection property="emps" select="com.atguigu.mybatis.dao.EmployeeMapper.getEmployeesByDeptId"  column="id" fetchType="lazy"**></**collection**> |

#### 4.7.8 扩展: 分步查询多列值的传递

1. 如果分步查询时，需要传递给调用的查询中多个参数，则需要将多个参数封装成

Map来进行传递，语法如下: {k1=v1, k2=v2....}

1. 在所调用的查询方，取值时就要参考Map的取值方式，需要严格的按照封装map

时所用的key来取值.

## 第5章 MyBatis 动态SQL

### 5.1 MyBatis动态SQL简介

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

1. OGNL（ Object Graph Navigation Language ）对象图导航语言，这是一种强大的

表达式语言，通过它可以非常方便的来操作对象属性。 类似于我们的EL，SpEL等

访问对象属性： person.name

调用方法： person.getName()

调用静态属性/方法： @java.lang.Math@PI

@java.util.UUID@randomUUID()

调用构造方法： new com.atguigu.bean.Person(‘admin’).name

运算符： +,-\*,/,%

逻辑运算符： in,not in,>,>=,<,<=,==,!=

注意：xml中特殊符号如 ” , > , < 等这些都需要使用转义字符

### 5.2 if where

1. if用于完成简单的判断.
2. where用于解决SQL语句中where关键字以及条件中第一个and或者or的问题

|  |
| --- |
| <**select id="getEmployeeByConditionIf" resultType="com.atguigu.mybatis.entities.Employee"**>  select id,last\_name,email,salary  from employees  <**where**>  <**if test="id!=null"**>  id=#{id}  </**if**>  <**if test="lastName!=null&amp;&amp;lastName!=&quot;&quot;"**>  and last\_name=#{lastName}  </**if**>  <**if test="email!=null and email.trim()!=''"**>  and email=#{email}  </**if**>  <**if test="salary!=null"**>  and salary=#{salary}  </**if**>  </**where**> </**select**> |

### 5.3 trim

1. trim 可以在条件判断完的SQL语句前后 添加或者去掉指定的字符

prefix: 添加前缀

prefixOverrides: 去掉前缀

suffix: 添加后缀

suffixOverrides: 去掉后缀

|  |
| --- |
| <**select id="getEmployeeByConditionIf" resultType="com.atguigu.mybatis.entities.Employee"**>  select id,last\_name,email,salary  from employees  <**trim prefix="where" suffixOverrides="and"**>  <**if test="id!=null"**>  id=#{id} and  </**if**>  <**if test="lastName!=null&amp;&amp;lastName!=&quot;&quot;"**>  last\_name=#{lastName} and  </**if**>  <**if test="email!=null and email.trim()!=''"**>  email=#{email} and  </**if**>  <**if test="salary!=null"**>  salary=#{salary}  </**if**>  </**trim**> |

### 5.4 set

1. set 主要是用于解决修改操作中SQL语句中可能多出逗号的问题

|  |
| --- |
| <**update id="updateEmployeeByConditionSet"**>  update employees  <**set**>  <**if test="lastName!=null and lastName.trim()!=''"**>  last\_name = #{lastName},  </**if**>  <**if test="email!=null and email.trim()!=''"**>  email = #{email},  </**if**>  <**if test="salary!=null"**>  salary = #{salary},  </**if**>  </**set**>  where id = #{id} </**update**> |

### 5.5 choose(when、otherwise)

1. choose 主要是用于分支判断，类似于java中的switch case,只会满足所有分支中的一个

|  |
| --- |
| <**select id="getEmployeeByConditionChoose" resultType="com.atguigu.mybatis.entities.Employee"**>  select id,last\_name,email,salary  from employees  <**where**>  <**choose**>  <**when test="id!=null"**>  id = #{id}  </**when**>  <**when test="lastName!=null"**>  last\_name = #{lastName}  </**when**>  <**when test="email!=null"**>  email = #{email}  </**when**>  <**otherwise**>  salary = #{salary}  </**otherwise**>  </**choose**>  </**where**> </**select**> |

### 5.6 foreach

1. foreach 主要用于循环迭代

collection: 要迭代的集合

item: 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:

迭代的是List集合: index表示的当前元素的下标

迭代的Map集合: index表示的当前元素的key

1. EmployeeMapper中的方法

|  |
| --- |
| List<Employee> getEmployeesByConditionForeach(@Param(**"ids"**) List<Integer> ids); |

1. EmployeeMapper.xml文件中的映射

|  |
| --- |
| <**select id="getEmployeesByConditionForeach" resultType="com.atguigu.mybatis.entities.Employee"**>  select id,last\_name,email,salary  from employees  where id in  <**foreach collection="ids" open="(" close=")" separator="," item="id"**>  #{id}  </**foreach**> </**select**> |

### 5.7 sql

1. sql 标签是用于抽取可重用的sql片段，将相同的，使用频繁的SQL片段抽取出来，单独定义，方便多次引用.
2. 抽取SQL:

|  |
| --- |
| <**sql id="selectSql"**>  select id,last\_name,email,salary  from employees </**sql**> |

1. 引用SQL:

|  |
| --- |
| <**include refid="selectSql"**></**include**> |

## 第6章 MyBatis 缓存机制

### 6.1 缓存机制简介

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

一级缓存

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

### 6.2 一级缓存的使用

1. 一级缓存(local cache), 即本地缓存, 作用域默认为sqlSession。当 Session flush 或 close 后, 该 Session 中的所有 Cache 将被清空。
2. 本地缓存不能被关闭, 但可以调用 clearCache() 来清空本地缓存, 或者改变缓存的作用域.
3. 在mybatis3.1之后, 可以配置本地缓存的作用域. 在 mybatis的全局配置文件中配置。

|  |  |  |
| --- | --- | --- |
| 设置名 | 描述 | 默认值 |
| localCacheScope | MyBatis 利用本地缓存机制（Local Cache）防止循环引用和加速重复的嵌套查询。 默认值为 SESSION，会缓存一个会话中执行的所有查询。 若设置值为 STATEMENT，本地缓存将仅用于执行语句，对相同 SqlSession 的不同查询将不会进行缓存。 | SESSION |

1. 一级缓存的工作机制

同一次会话期间只要查询过的数据都会保存在当前SqlSession的一个Map中

key: hashCode+查询的SqlId+编写的sql查询语句+参数

### 6.3 一级缓存失效的几种情况

1. 不同的SqlSession对应不同的一级缓存
2. 同一个SqlSession但是查询条件不同
3. 同一个SqlSession两次查询期间执行了任何一次增删改操作
4. 同一个SqlSession两次查询期间手动清空了缓存

### 6.4 二级缓存的使用

1. 二级缓存(second level cache)，全局作用域缓存
2. 二级缓存默认不开启，需要手动配置
3. MyBatis提供二级缓存的接口以及实现，缓存实现要求POJO实现Serializable接口
4. 二级缓存在 SqlSession 关闭或提交之后才会生效
5. 二级缓存使用的步骤:
6. 全局配置文件中开启二级缓存<setting name="cacheEnabled" value="true"/>
7. 需要使用二级缓存的映射文件处使用cache配置缓存<cache />
8. 注意：POJO需要实现Serializable接口
9. 二级缓存相关的属性
10. eviction=“FIFO”：缓存回收策略：

LRU – 最近最少使用的：移除最长时间不被使用的对象。

FIFO – 先进先出：按对象进入缓存的顺序来移除它们。

SOFT – 软引用：移除基于垃圾回收器状态和软引用规则的对象。

软引用：软引用需要用SoftReference类来实现，对于只有软引用的对象来说，当系统内存足够时它不会被回收，当系统内存空间不足时它会被回收。软引用通常用在对内存敏感的程序中。

WEAK – 弱引用：更积极地移除基于垃圾收集器状态和弱引用规则的对象。

弱引用：弱引用需要用WeakReference类来实现，它比软引用的生存期更短，对于只有弱引用的对象来说，只要垃圾回收机制一运行，不管JVM的内存空间是否足够，总会回收该对象占用的内存。

默认的是 LRU。

1. flushInterval：刷新间隔，单位毫秒

默认情况是不设置，也就是没有刷新间隔，缓存仅仅调用语句时刷新

1. size：引用数目，正整数

代表缓存最多可以存储多少个对象，太大容易导致内存溢出

1. readOnly：只读，true/false

true：只读缓存；会给所有调用者返回缓存对象的相同实例。因此这些对象不能被修改。这提供了很重要的性能优势。

false：读写缓存；会返回缓存对象的拷贝（通过序列化）。这会慢一些，但是安全，因此默认是 false。

### 6.5 缓存的相关属性设置

1. 全局setting的cacheEnable：

配置二级缓存的开关，一级缓存一直是打开的。

1. select标签的useCache属性：

这个属性是用来配置这个select是否使用二级缓存，默认为true。

一级缓存一直是使用的。

1. sql标签的flushCache属性：

增删改默认flushCache=true。sql执行以后，会同时清空一级和二级缓存。

查询默认 flushCache=false。

1. sqlSession.clearCache()：只是用来清除一级缓存。

### 6.6 整合第三方缓存

1. 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. 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
3. 整合EhCache缓存的步骤:
4. 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

1. 编写ehcache.xml配置文件

|  |
| --- |
| *<?***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="E:\mybatis\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）  -->* |

1. 配置cache标签

|  |
| --- |
| *<!--使用第三方二级缓存-->* <**cache type="org.mybatis.caches.ehcache.EhcacheCache"**></**cache**> |

## 第7章 MyBatis 逆向工程

### 7.1 逆向工程简介

1. 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

官方文档地址

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

官方工程地址

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

### 7.2 逆向工程的配置

1. 导入逆向工程的jar包

mybatis-generator-core-1.3.2.jar

1. 编写MBG的配置文件mbg.xml（重要几处配置）,参考官方文档

|  |
| --- |
| **<!DOCTYPE generatorConfiguration PUBLIC  "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"  "http://mybatis.org/dtd/mybatis-generator-config\_1\_0.dtd"*>*** <**generatorConfiguration**>  *<!--  id属性：设置一个唯一标识  targetRuntime属性值说明：  MyBatis3Simple：基本的增删改查  MyBatis3：带条件查询的增删改查  -->* <**context id="simple" targetRuntime="MyBatis3Simple"**>  *<!--设置连接数据库的相关信息-->* <**jdbcConnection driverClass="com.mysql.jdbc.Driver"  connectionURL="jdbc:mysql://localhost:3306/mybatis"  userId="root"  password="root"**/>   *<!--设置JavaBean的生产策略-->* <**javaModelGenerator targetPackage="com.atguigu.mybatis.mbg.beans" targetProject="mybatis/src"**/>   *<!--设置SQL映射文件的生产策略-->* <**sqlMapGenerator targetPackage="com.atguigu.mybatis.mbg.mapper" targetProject="mybatis/src"**/>   *<!--设置Mapper接口的生产策略-->* <**javaClientGenerator type="XMLMAPPER" targetPackage="com.atguigu.mybatis.mbg.mapper" targetProject="mybatis/src"**/>   *<!--逆向分析的表-->* <**table tableName="employees" domainObjectName="Employee"**/>  <**table tableName="departments" domainObjectName="Department"**/>  </**context**> </**generatorConfiguration**> |

1. 参考官方文档创建代码生成器运行代码

|  |
| --- |
| @Test **public void** testMGB() **throws** IOException, XMLParserException, InvalidConfigurationException, SQLException, InterruptedException {  List<String> warnings = **new** ArrayList<String>();  **boolean** overwrite = **true**;  File configFile = **new** File(**"mybatis/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**); } |

### 7.3 逆向工程的使用

1. 基本查询的测试

|  |
| --- |
| */\*  测试获取一个对象 \*/* @Test **public void** testSelectByPrimaryKey() **throws** IOException {  *//设置MyBatis全局配置文件的路径* String resource = **"mybatis-config.xml"**;  *//读取类路径下的配置文件得到输入流* InputStream inputStream = Resources.*getResourceAsStream*(resource);  *//创建SqlSessionFactory对象* SqlSessionFactory sqlSessionFactory = **new** SqlSessionFactoryBuilder().build(inputStream);  *//获取SqlSession对象，相当于Connection对象* SqlSession sqlSession = sqlSessionFactory.openSession();  **try** {  *//获取Mapper代理对象* EmployeeMapper mapper = sqlSession.getMapper(EmployeeMapper.**class**);  Employee employee = mapper.selectByPrimaryKey(1);  System.***out***.println(employee);  } **finally** {  *//关闭sqlSession* sqlSession.close();  } } |

1. 带条件查询的测试
   1. 查询名字里包含a并且工资大于等于10000的员工

|  |
| --- |
| @Test **public void** testSelectByExample() **throws** IOException {  *//设置MyBatis全局配置文件的路径* String resource = **"mybatis-config.xml"**;  *//读取类路径下的配置文件得到输入流* InputStream inputStream = Resources.*getResourceAsStream*(resource);  *//创建SqlSessionFactory对象* SqlSessionFactory sqlSessionFactory = **new** SqlSessionFactoryBuilder().build(inputStream);  *//获取SqlSession对象，相当于Connection对象* SqlSession sqlSession = sqlSessionFactory.openSession();  **try** {  *//获取Mapper代理对象* EmployeeMapper mapper = sqlSession.getMapper(EmployeeMapper.**class**);  *//创建EmployeeExample对象* EmployeeExample employeeExample = **new** EmployeeExample();  *//创建Criteria对象* EmployeeExample.Criteria criteria = employeeExample.createCriteria();  *//**查询名字里包含a并且工资大于等于10000的员工* criteria.andLastNameLike(**"%a%"**).andSalaryGreaterThanOrEqualTo(10000.00);  List<Employee> employees = mapper.selectByExample(employeeExample);  **for** (Employee employee : employees) {  System.***out***.println(employee);  }  } **finally** {  *//关闭sqlSession* sqlSession.close();  } } |

* 1. 测试名字了包含a或者工资小于等于10000的员工

|  |
| --- |
| @Test **public void** testSelectByExample() **throws** IOException {  *//设置MyBatis全局配置文件的路径* String resource = **"mybatis-config.xml"**;  *//读取类路径下的配置文件得到输入流* InputStream inputStream = Resources.*getResourceAsStream*(resource);  *//创建SqlSessionFactory对象* SqlSessionFactory sqlSessionFactory = **new** SqlSessionFactoryBuilder().build(inputStream);  *//获取SqlSession对象，相当于Connection对象* SqlSession sqlSession = sqlSessionFactory.openSession();  **try** {  *//获取Mapper代理对象* EmployeeMapper mapper = sqlSession.getMapper(EmployeeMapper.**class**);  *//创建EmployeeExample对象* EmployeeExample employeeExample = **new** EmployeeExample();  *//创建Criteria对象* EmployeeExample.Criteria criteria = employeeExample.createCriteria();  *//查询名字里包含a的员工* criteria.andLastNameLike(**"%a%"**);  EmployeeExample.Criteria criteria1 = employeeExample.createCriteria();  *//查询工资小于等于10000的员工* criteria1.andSalaryLessThanOrEqualTo(10000.00);  *//使用or关联两个Criteria对象* employeeExample.or(criteria1);  List<Employee> employees = mapper.selectByExample(employeeExample);  **for** (Employee employee : employees) {  System.***out***.println(employee);  }  } **finally** {  *//关闭sqlSession* sqlSession.close();  } } |

## 第8章 扩展-PageHelper分页插件

### 8.1 PageHelper 分页插件简介

1. PageHelper是MyBatis中非常方便的第三方分页插件
2. 官方文档：

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

1. 我们可以对照官方文档的说明，快速的使用插件

### 8.2 PageHelper的使用步骤

1. 导入以下jar包

pagehelper-5.0.0.jar

jsqlparser-0.9.5.jar

1. 在MyBatis全局配置文件中配置分页插件

|  |
| --- |
| <**plugins**>  *<!--配置PageHelper分页插件-->* <**plugin interceptor="com.github.pagehelper.PageInterceptor"**></**plugin**> </**plugins**> |

1. 使用PageHelper提供的方法进行分页
2. 可以使用更强大的PageInfo封装返回结果

### 8.3 Page对象的使用

1. 在查询之前通过PageHelper.startPage(页码，条数)设置分页信息，该方法返回Page对象

|  |
| --- |
| @Test **public void** testPageHelper() **throws** IOException {  *//设置MyBatis全局配置文件的路径* String resource = **"mybatis-config.xml"**;  *//读取类路径下的配置文件得到输入流* InputStream inputStream = Resources.*getResourceAsStream*(resource);  *//创建SqlSessionFactory对象* SqlSessionFactory sqlSessionFactory = **new** SqlSessionFactoryBuilder().build(inputStream);  *//获取SqlSession对象，相当于Connection对象* SqlSession sqlSession = sqlSessionFactory.openSession();  **try** {  *//获取Mapper代理对象* EmployeeMapper mapper = sqlSession.getMapper(EmployeeMapper.**class**);  *//设置分页信息，每页显示3条记录，获取第1页* Page<Object> page = PageHelper.*startPage*(1, 3);  *//获取所有员工* List<Employee> employees = mapper.getEmployees();  System.***out***.println(**"当前页是："**+page.getPageNum());  System.***out***.println(**"每页显示的条数是："**+page.getPageSize());  System.***out***.println(**"总页数是："**+page.getPages());  System.***out***.println(**"总记录数是："**+page.getTotal());  System.***out***.println(**"当前页中的记录有："**);  **for** (Employee employee : employees) {  System.***out***.println(employee);  }  } **finally** {  *//关闭sqlSession* sqlSession.close();  } } |

### 8.4 PageInfo对象的使用

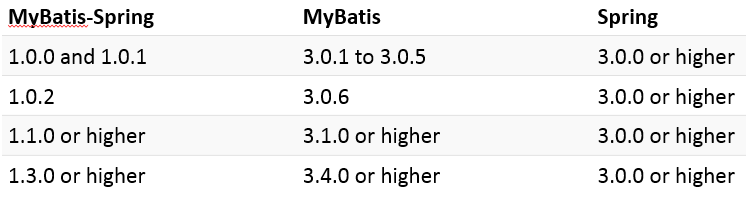
1. 在查询完数据后，使用PageInfo对象封装查询结果，可以获取更详细的分页信息以及可以完成分页逻辑

|  |
| --- |
| @Test **public void** testPageHelper() **throws** IOException {  *//设置MyBatis全局配置文件的路径* String resource = **"mybatis-config.xml"**;  *//读取类路径下的配置文件得到输入流* InputStream inputStream = Resources.*getResourceAsStream*(resource);  *//创建SqlSessionFactory对象* SqlSessionFactory sqlSessionFactory = **new** SqlSessionFactoryBuilder().build(inputStream);  *//获取SqlSession对象，相当于Connection对象* SqlSession sqlSession = sqlSessionFactory.openSession();  **try** {  *//获取Mapper代理对象* EmployeeMapper mapper = sqlSession.getMapper(EmployeeMapper.**class**);  *//设置分页信息* Page<Object> page = PageHelper.*startPage*(4, 2);  *//获取所有员工* List<Employee> employees = mapper.getEmployees();  *//创建PageInfo对象设置每页只显示5个页码* PageInfo<Employee> pageInfo = **new** PageInfo<>(employees, 5);  System.***out***.println(**"当前页是："**+pageInfo.getPageNum());  System.***out***.println(**"每页显示的条数是："**+pageInfo.getPageSize());  System.***out***.println(**"总页数是："**+pageInfo.getPages());  System.***out***.println(**"总记录数是："**+pageInfo.getTotal());  System.***out***.println(**"是否有上一页："**+pageInfo.isHasPreviousPage());  System.***out***.println(**"上一页是："**+pageInfo.getPrePage());  System.***out***.println(**"是否有下一页："**+pageInfo.isHasNextPage());  System.***out***.println(**"下一页是："**+pageInfo.getNextPage());  System.***out***.println(**"是否是第一页："**+pageInfo.isIsFirstPage());  System.***out***.println(**"是否是最后一页："**+pageInfo.isIsLastPage());  System.***out***.println(**"导航页的第一个页码是："**+pageInfo.getNavigateFirstPage());  System.***out***.println(**"导航页的最后一个页码是："**+pageInfo.getNavigateLastPage());  System.***out***.println(**"导航页的总页码是："**+pageInfo.getNavigatePages());  System.***out***.println(**"当前页中的记录有："**);  **for** (Employee employee : employees) {  System.***out***.println(employee);  }  System.***out***.println(**"页码信息："**);  **int**[] navigatepageNums = pageInfo.getNavigatepageNums();  **for** (**int** navigatepageNum : navigatepageNums) {  System.***out***.print(navigatepageNum+**" "**);  }  } **finally** {  *//关闭sqlSession* sqlSession.close();  } } |

## 第9章 SSM框架整合

### 9.1 整合注意事项

1. 查看不同MyBatis版本整合Spring时使用的适配包；



1. 下载整合适配包

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

1. 官方整合示例，jpetstore

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

### 9.2 整合步骤

1. 搭建环境

创建一个动态的WEB工程

导入SSM需要使用的jar包

导入整合适配包

导入其他技术的一些支持包 连接池 数据库驱动 日志....

1. Spring + Springmvc

在web.xml中配置: Springmvc的前端控制器 实例化Spring容器的监听器 字符编码过滤器 REST 过滤器

创建Spring的配置文件: applicationContext.xml:组件扫描、 连接池、 事务.....

创建Springmvc的配置文件: springmvc.xml : 组件扫描、 视图解析器 <mvc:...>

1. MyBatis

创建MyBatis的全局配置文件

编写实体类 Mapper接口 Mapper映射文件

1. Spring + MyBatis ：

MyBatis的 SqlSession的创建

MyBatis的 Mapper接口的代理实现类

1. 测试: REST CRUD

课堂: 查询所有的员工信息,列表显示

课下: 增删改

### 9.3 整合的配置

#### 9.3.1 web.xml

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_4\_0.xsd"  version="4.0"**>   *<!--配置CharacterEncodingFilter过滤器，目的是为了解决POST请求的请求乱码问题-->* <**filter**>  <**filter-name**>encodingFilter</**filter-name**>  <**filter-class**>org.springframework.web.filter.CharacterEncodingFilter</**filter-class**>  *<!--给encoding属性赋值-->* <**init-param**>  <**param-name**>encoding</**param-name**>  <**param-value**>UTF-8</**param-value**>  </**init-param**>  *<!--给forceRequestEncoding属性赋值-->* <**init-param**>  <**param-name**>forceRequestEncoding</**param-name**>  <**param-value**>true</**param-value**>  </**init-param**>  </**filter**>  <**filter-mapping**>  <**filter-name**>encodingFilter</**filter-name**>  <**url-pattern**>/\*</**url-pattern**>  </**filter-mapping**>  *<!--配置前端控制器DispatcherServlet-->* <**servlet**>  <**servlet-name**>dispatcherServlet</**servlet-name**>  <**servlet-class**>org.springframework.web.servlet.DispatcherServlet</**servlet-class**>  <**init-param**>  <**param-name**>contextConfigLocation</**param-name**>  <**param-value**>classpath:springmvc.xml</**param-value**>  </**init-param**>  <**load-on-startup**>1</**load-on-startup**>  </**servlet**>  <**servlet-mapping**>  <**servlet-name**>dispatcherServlet</**servlet-name**>  <**url-pattern**>/</**url-pattern**>  </**servlet-mapping**>   *<!--配置HiddenHttpMethodFilter过滤器，目的是为了将POST请求转换为Put或Delete请求-->* <**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**>    <**context-param**>  <**param-name**>contextConfigLocation</**param-name**>  <**param-value**>classpath:beans.xml</**param-value**>  </**context-param**>  *<!--配置ContextLoaderListener监听器，目的是为了加载Spring的配置文件，帮我们初始化IOC容器-->* <**listener**>  <**listener-class**>org.springframework.web.context.ContextLoaderListener</**listener-class**>  </**listener**>  </**web-app**> |

#### 9.3.2 Spring配置

|  |
| --- |
| *<?***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"  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd http://www.springframework.org/schema/context https://www.springframework.org/schema/context/spring-context.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx.xsd"**>  *<!--配置自动扫描的包-->* <**context:component-scan base-package="com.atguigu.ssm"**>  *<!--配置不扫描Controller-->* <**context:exclude-filter type="annotation" expression="org.springframework.stereotype.Controller"**/>  </**context:component-scan**>   *<!--引入外部属性文件-->* <**context:property-placeholder location="classpath:druid.properties"**></**context:property-placeholder**>   *<!--配置数据源-->* <**bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource"**>  <**property name="username" value="${jdbc.username}"**></**property**>  <**property name="password" value="${jdbc.password}"**></**property**>  <**property name="url" value="${jdbc.url}"**></**property**>  <**property name="driverClassName" value="${jdbc.driverClassName}"**></**property**>  <**property name="initialSize" value="${jdbc.initialSize}"**></**property**>  <**property name="maxActive" value="${jdbc.maxActive}"**></**property**>  </**bean**>   *<!--配置事务管理器-->* <**bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager"**>  <**property name="dataSource" ref="dataSource"**></**property**>  </**bean**>   *<!--开启事务注解支持-->* <**tx:annotation-driven**></**tx:annotation-driven**> </**beans**> |

#### 9.3.3 SpringMVC配置

|  |
| --- |
| *<?***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"  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd http://www.springframework.org/schema/context https://www.springframework.org/schema/context/spring-context.xsd http://www.springframework.org/schema/mvc https://www.springframework.org/schema/mvc/spring-mvc.xsd"**>   *<!--配置自动扫描的包-->* <**context:component-scan base-package="com.atguigu.ssm" use-default-filters="false"**>  *<!--配置只扫描Controller-->* <**context:include-filter type="annotation" expression="org.springframework.stereotype.Controller"**/>  </**context:component-scan**>   *<!--配置视图解析器-->* <**bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver"**>  *<!--配置前缀-->* <**property name="prefix" value="/WEB-INF/views/"**></**property**>   *<!--配置后缀-->* <**property name="suffix" value=".jsp"**></**property**>  </**bean**>    *<!--配置处理静态资源-->* <**mvc:default-servlet-handler**></**mvc:default-servlet-handler**>   *<!--配置了处理静态资源之后必须配置以下标签-->* <**mvc:annotation-driven**></**mvc:annotation-driven**> </**beans**> |

#### 9.3.4 MyBatis配置

1. 全局文件的配置

|  |
| --- |
| *<?***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"*>*** <**configuration**>   *<!-- Spring 整合 MyBatis 后， MyBatis中配置数据源，事务等一些配置都可以  迁移到Spring的整合配置中。MyBatis配置文件中只需要配置与MyBatis相关  的即可。  -->  <!-- settings: 包含很多重要的设置项 -->* <**settings**>  *<!-- 设置驼峰命名映射规则 -->* <**setting name="mapUnderscoreToCamelCase" value="true"**/>  *<!-- 设置Mybatis对null值的默认处理 -->* <**setting name="jdbcTypeForNull" value="NULL"**/>  *<!-- 开启延迟加载 -->* <**setting name="lazyLoadingEnabled" value="true"**/>  *<!-- 设置数据按需加载 -->* <**setting name="aggressiveLazyLoading" value="false"**/>  *<!-- 配置开启二级缓存 -->* <**setting name="cacheEnabled" value="true"**/>  </**settings**> </**configuration**> |

1. EmployeeMapper接口

|  |
| --- |
| List<Employee> getEmployees(); |

1. EmployeeMapper映射文件

|  |
| --- |
| *<?***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"*>*** *<!--namespace属性：设置接口的全类名-->* <**mapper namespace="com.atguigu.ssm.mapper.EmployeeMapper"**>   <**select id="getEmployees" resultMap="myEmp"**>  SELECT e.\*,d.id d\_id,d.name d\_name  FROM employees e  LEFT JOIN departments d  ON e.dept\_id = d.id  </**select**>   *<!--自定义结果集-->* <**resultMap id="myEmp" type="com.atguigu.ssm.entities.Employee"**>  <**id column="id" property="id"**></**id**>  <**result column="last\_name" property="lastName"**></**result**>  <**result column="email" property="email"**></**result**>  <**result column="gender" property="gender"**></**result**>  *<!--使用级联方式设置部门信息-->* <**association property="department" javaType="com.atguigu.ssm.entities.Department"**>  <**id column="d\_id" property="id"**></**id**>  <**result column="d\_name" property="departmentName"**></**result**>  </**association**>  </**resultMap**>   </**mapper**> |

#### 9.3.5 Spring 整合MyBatis 配置

|  |
| --- |
| *<!--整合MyBatis--> <!--1.配置SqlSessionFactoryBean-->* <**bean id="sqlSessionFactoryBean" class="org.mybatis.spring.SqlSessionFactoryBean"**>  *<!--配置数据源属性-->* <**property name="dataSource" ref="dataSource"**></**property**>  *<!--配置MyBatis全局配置文件的路径-->* <**property name="configLocation" value="classpath:mybatis-config.xml"**></**property**>  *<!--配置Mapper映射文件的路径-->* <**property name="mapperLocations" value="classpath:com/atguigu/ssm/mapper/\*.xml"**></**property**>  *<!--配置别名等原来在MyBatis全局配置文件中的其他配置-->* </**bean**>  *<!--2.扫描Mapper接口-->* <**mybatis-spring:scan base-package="com.atguigu.ssm.mapper"**></**mybatis-spring:scan**> |

### 9.4 整合测试

1. 首页中创建超链接

|  |
| --- |
| <**a href="${**pageContext.request.contextPath**}/getEmployees"**>获取所有员工</**a**> |

1. 编写处理器和处理方法

|  |
| --- |
| **package** com.atguigu.ssm.controller;  **import** com.atguigu.ssm.entities.Employee; **import** com.atguigu.ssm.mapper.EmployeeMapper; **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** java.util.List; **import** java.util.Map;  @Controller **public class** EmployeeController {   @Autowired  **private** EmployeeMapper **employeeMapper**;   *//获取所有员工* @RequestMapping(value = **"/getEmployees"**,method = RequestMethod.***GET***)  **public** String getEmployees(Map<String , Object> map){  *//调用EmployeeMapper中获取所有员工的方法* List<Employee> employees = **employeeMapper**.getEmployees();  *//将所有的员工放到map中* map.put(**"emps"**,employees);  **return "list"**;  } } |

1. 创建视图页面展示所有员工

|  |
| --- |
| <%@ **page contentType**="**text/html;charset=UTF-8**" **language**="**java**" %> <%@ **taglib prefix**="**c**" **uri**="**http://java.sun.com/jsp/jstl/core**" %> <**html**> <**head**>  <**title**>显示所有员工页面</**title**> </**head**> <**body**>  <**form action="" method="post" id="deleteForm"**>  <**input type="hidden" name="\_method" value="delete"**>  </**form**>  <**center**>  <**c:if test="${empty** requestScope.emps**}"**>  <**h1**>没有任何员工</**h1**>  </**c:if**>  <**c:if test="${not empty** requestScope.emps**}"**>  <**h1**>员工信息</**h1**>  <**table border="1" cellspacing="0" cellpadding="10" style="text-align**: **center"**>  <**tr**>  <**th**>Id</**th**>  <**th**>LastName</**th**>  <**th**>Email</**th**>  <**th**>Gender</**th**>  <**th**>Department</**th**>  <**th colspan="2"**>Operate</**th**>  </**tr**>  <**c:forEach items="${**requestScope.emps**}" var="emp"**>  <**tr**>  <**td**>**${**emp.id**}**</**td**>  <**td**>**${**emp.lastName**}**</**td**>  <**td**>**${**emp.email**}**</**td**>  <**td**>  <**c:if test="${**emp.gender==1**}"**>  男  </**c:if**>  <**c:if test="${**emp.gender==0**}"**>  女  </**c:if**>  </**td**>  <**td**>**${**emp.department.departmentName**}**</**td**>  <**td**><**a href="#"**>Edit</**a**></**td**>  <**td**><**a href="#"**>Delete</**a**></**td**>  </**tr**>  </**c:forEach**>  </**table**>  </**c:if**>  <**br**>  <**hr**>  <**br**>  </**center**> </**body**> </**html**> |