**MyBatis框架**

尚硅谷java研究院

版本：V 1.0

# 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 Old Java Objects，普通的Java对象）映射成数据库中的记录
4. 半自动ORM（Object Relation Mapping`）框架

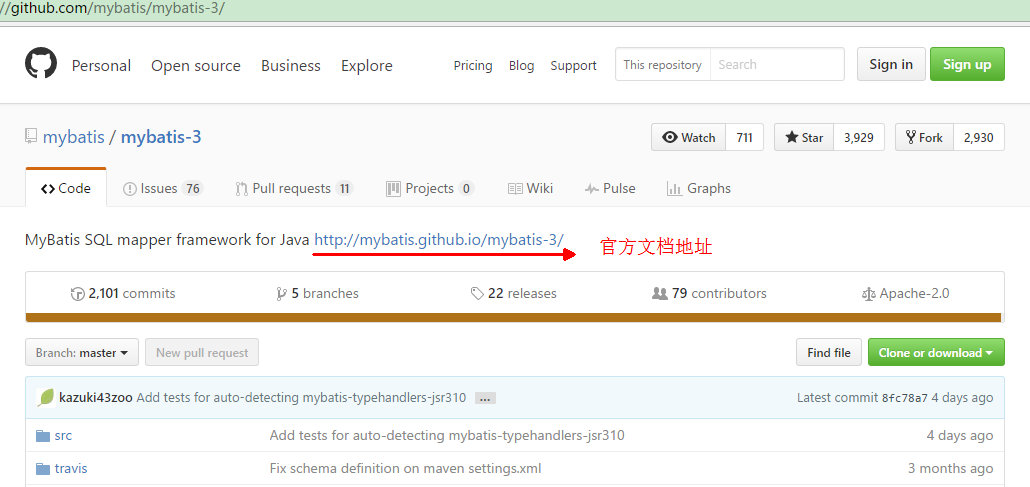
## 1.3 为什么要使用MyBatis – 现有持久化技术的对比

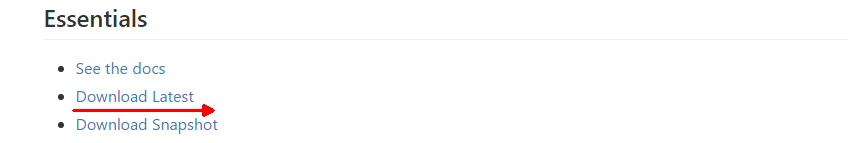
1. JDBC
2. SQL夹在Java代码块里，耦合度高导致硬编码内伤
3. 维护不易且实际开发需求中sql是有变化，频繁修改的情况多见
4. Hibernate和JPA
5. 长难复杂SQL，对于Hibernate而言处理也不容易
6. 内部自动生产的SQL，不容易做特殊优化
7. 基于全映射的全自动框架，大量字段的POJO进行部分映射时比较困难。导致数据库性能下降
8. MyBatis
9. 对开发人员而言，核心sql还是需要自己优化
10. sql和java编码分开，功能边界清晰，一个专注业务、一个专注数据

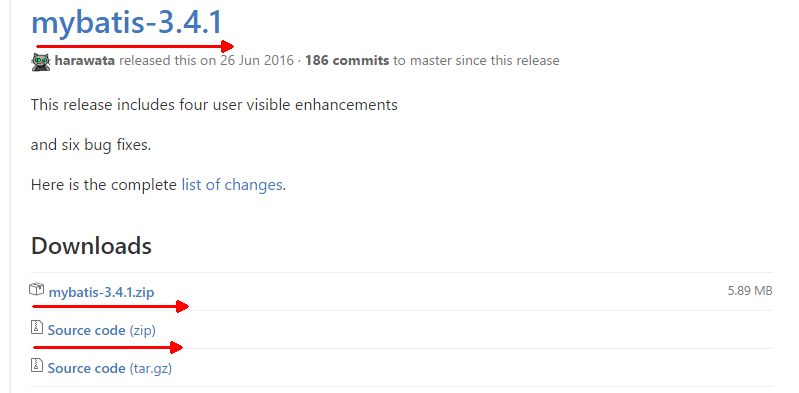
## 1.4 如何下载MyBatis

1. 下载网址

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







# 第2章 MyBatis HelloWorld

## 2.1 开发环境的准备

1. 导入MyBatis框架的jar包、Mysql驱动包、log4j的jar包

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

1. 导入log4j 的配置文件

|  |
| --- |
| <?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 创建测试表

|  |
| --- |
| -- 创建库  CREATE DATABASE test\_mybatis;  -- 使用库  USE test\_mybatis;  -- 创建表  CREATE TABLE tbl\_employee(  id INT(11) PRIMARY KEY AUTO\_INCREMENT,  last\_name VARCHAR(50),  email VARCHAR(50),  gender CHAR(1)  ); |

## 2.3 创建javaBean

|  |
| --- |
| **public** **class** Employee {  **private** Integer id ;  **private** String lastName;  **private** String email ;  **private** String 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 + "]";  } |

## 2.3 创建MyBatis的全局配置文件

1. 参考MyBatis的官网手册

|  |
| --- |
| <?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>  <!-- 数据库连接环境的配置 -->  <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\_1129"* />  <property name=*"username"* value=*"root"* />  <property name=*"password"* value=*"1234"* />  </dataSource>  </environment>  </environments>  <!-- 引入SQL映射文件,Mapper映射文件 -->  <mappers>  <mapper resource=*"EmployeeMapper.xml"* />  </mappers>  </configuration> |

## 2.4 创建Mybatis的sql映射文件

1. 参考MyBatis的官方手册

|  |
| --- |
| <?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=*"suibian"*>  <select id=*"selectEmployee"* resultType=*"com.atguigu.myabtis.helloWorld.Employee"*>  select id ,last\_name lastName ,email ,gender from tbl\_employee where id = #{id}  <!-- select \* from tbl\_employee where id = #{id} -->  </select>  </mapper> |

## 2.5 测试

1. 参考MyBatis的官方手册

|  |
| --- |
| @Test  **public** **void** test() **throws** Exception {  String resource = "mybatis-config.xml";  InputStream inputStream = Resources.*getResourceAsStream*(resource);  SqlSessionFactory sqlSessionFactory =  **new** SqlSessionFactoryBuilder().build(inputStream);  System.***out***.println(sqlSessionFactory);    SqlSession session = sqlSessionFactory.openSession();  **try** {  Employee employee =  session.selectOne("suibian.selectEmployee", 1001);  System.***out***.println(employee);  } **finally** {  session.close();  }  } |

## 2.6 Mapper接口开发MyBatis HelloWorld

1. 编写Mapper接口

|  |
| --- |
| **public** **interface** EmployeeMapper {    **public** Employee getEmployeeById(Integer id );    } |

1. 完成两个绑定
2. Mapper接口与Mapper映射文件的绑定

在Mppper映射文件中的<mapper>标签中的namespace中必须指定Mapper接口

的全类名

1. 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.
2. 获取Mapper接口的代理实现类对象

|  |
| --- |
| @Test  **public** **void** test() **throws** Exception{  String resource = "mybatis-config.xml";  InputStream inputStream =  Resources.*getResourceAsStream*(resource);  SqlSessionFactory sqlSessionFactory =  **new** SqlSessionFactoryBuilder()  .build(inputStream);  SqlSession session =  sqlSessionFactory.openSession();    **try** {  //Mapper接口:获取Mapper接口的 代理实现类对象  EmployeeMapper mapper =  session.getMapper(EmployeeMapper.**class**);  Employee employee =  mapper.getEmployeeById(1006);  System.***out***.println(employee);  } **finally** {  session.close();  }  } |

# 第3章 MyBatis全局配置文件

## 3.1 MyBatis全局配置文件简介

1. The MyBatis configuration contains settings and properties that have a dramatic effect on how MyBatis behaves.

MyBatis 的配置文件包含了影响 MyBatis 行为甚深的设置（settings）和属性（properties）信息。

1. 文件结构如下:

configuration 配置

[properties 属性](http://www.mybatis.org/mybatis-3/zh/configuration.html)

[settings 设置](http://www.mybatis.org/mybatis-3/zh/configuration.html)

[typeAliases 类型命名](http://www.mybatis.org/mybatis-3/zh/configuration.html)

[typeHandlers 类型处理器](http://www.mybatis.org/mybatis-3/zh/configuration.html)

[objectFactory 对象工厂](http://www.mybatis.org/mybatis-3/zh/configuration.html)

[plugins 插件](http://www.mybatis.org/mybatis-3/zh/configuration.html)

[environments 环境](http://www.mybatis.org/mybatis-3/zh/configuration.html)

environment 环境变量

transactionManager 事务管理器

dataSource 数据源

[databaseIdProvider 数据库厂商标识](http://www.mybatis.org/mybatis-3/zh/configuration.html)

[mappers 映射器](http://www.mybatis.org/mybatis-3/zh/configuration.html)

## 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="driver" value="com.mysql.jdbc.Driver" />  <property name="url"  value="jdbc:mysql://localhost:3306/test\_mybatis" />  <property name="username" value="root" />  <property name="password" value="1234" />  </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

|  |
| --- |
| jdbc.driver=com.mysql.jdbc.Driver  jdbc.url=jdbc:mysql://localhost:3306/mybatis\_1129  jdbc.username=root  jdbc.password=1234 |

|  |
| --- |
| <!--  properties: 引入外部的属性文件  resource: 从类路径下引入属性文件  url: 引入网络路径或者是磁盘路径下的属性文件  -->  <properties resource=*"db.properties"* ></properties> |

3)在environment元素的dataSource元素中为其动态设置

|  |
| --- |
| <environments default=*"oracle"*>  <environment id=*"mysql"*>  <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> |

## 3.3 settings设置

1. 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。
2. 包含如下的setting设置:

|  |
| --- |
| <settings> <setting name="cacheEnabled" value="true"/> <setting name="lazyLoadingEnabled" value="true"/> <setting name="multipleResultSetsEnabled" value="true"/> <setting name="useColumnLabel" value="true"/> <setting name="useGeneratedKeys" value="false"/> <setting name="autoMappingBehavior" value="PARTIAL"/> <setting name="autoMappingUnknownColumnBehavior" value="WARNING"/> <setting name="defaultExecutorType" value="SIMPLE"/> <setting name="defaultStatementTimeout" value="25"/> <setting name="defaultFetchSize" value="100"/> <setting name="safeRowBoundsEnabled" value="false"/> <setting name="mapUnderscoreToCamelCase" value="false"/> <setting name="localCacheScope" value="SESSION"/> <setting name="jdbcTypeForNull" value="OTHER"/> <setting name="lazyLoadTriggerMethods"  value="equals,clone,hashCode,toString"/> </settings> |

## 3.4 typeAliases 别名处理

1. 类型别名是为 Java 类型设置一个短的名字，可以方便我们引用某个类。

|  |
| --- |
| <typeAliases>  <typeAlias type=*"com.atguigu.mybatis.beans.Employee"*  alias=*"emp"*/>  </typeAliases> |

1. 类很多的情况下，可以批量设置别名这个包下的每一个类创建一个默认的别名，就是简单类名小写

|  |
| --- |
| <typeAliases>  <package name=*"com.atguigu.mybatis.beans"*/>  </typeAliases> |

1. MyBatis已经取好的别名



## 3.5 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

id：指定当前环境的唯一标识

transactionManager、和dataSource都必须有

|  |
| --- |
| <environments default=*"oracle"*>  <environment id=*"mysql"*>  <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=*"oracle"*>  <transactionManager type=*"JDBC"*/>  <dataSource type=*"POOLED"*>  <property name=*"driver"* value=*"${orcl.driver}"* />  <property name=*"url"* value=*"${orcl.url}"* />  <property name=*"username"* value=*"${orcl.username}"* />  <property name=*"password"* value=*"${orcl.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. 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

## 3.6 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逐个注册SQL映射文件

resource : 引入类路径下的文件

url : 引入网络路径或者是磁盘路径下的文件

class : 引入Mapper接口.

有SQL映射文件 , 要求Mapper接口与 SQL映射文件同名同位置.

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=*"EmployeeMapper.xml"* />  <mapper class=*"com.atguigu.mybatis.dao.EmployeeMapper"*/>  <package name=*"com.atguigu.mybatis.dao"*/>  </mappers> |

1. 使用批量注册，这种方式要求SQL映射文件名必须和接口名相同并且在同一目录下

|  |
| --- |
| <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 映射文件有很少的几个顶级元素（按照它们应该被定义的顺序）：

cache – 给定命名空间的缓存配置。

cache-ref – 其他命名空间缓存配置的引用。

resultMap – 是最复杂也是最强大的元素，用来描述如何从数据库结果集中来加 载对象。

parameterMap – ~~已废弃！老式风格的参数映射。内联参数是首选,这个元素可能~~  ~~在将来被移除，这里不会记录。~~

sql – 可被其他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

## 4.2 Mybatis使用insert|update|delete|select完成CRUD

### 4.2.1 select

1. Mapper接口方法

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

1. Mapper映射文件

|  |
| --- |
| <select id=*"getEmployeeById"*  resultType=*"com.atguigu.mybatis.beans.Employee"*  databaseId=*"mysql"*>  select \* from tbl\_employee where id = ${\_parameter}  </select> |

### 4.2.2 insert

1. Mapper接口方法

|  |
| --- |
| **public** Integer insertEmployee(Employee employee); |

1. Mapper映射文件

|  |
| --- |
| <insert id=*"insertEmployee"*  parameterType=*"com.atguigu.mybatis.beans.Employee"*  databaseId=*"mysql"*>  insert into tbl\_employee(last\_name,email,gender) values(#{lastName},#{email},#{gender})  </insert> |

### 4.2.3 update

1. Mapper接口方法

|  |
| --- |
| **public** Boolean updateEmployee(Employee employee); |

1. Mapper映射文件

|  |
| --- |
| <update id=*"updateEmployee"* >  update tbl\_employee set last\_name = #{lastName},  email = #{email},  gender = #{gender}  where id = #{id}  </update> |

### 4.2.4 delete

1. Mapper接口方法

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

1. Mapper映射文件

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

## 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=*"insertEmployee"* parameterType=*"com.atguigu.mybatis.beans.Employee"*  databaseId=*"mysql"*  useGeneratedKeys=*"true"*  keyProperty=*"id"*>  insert into tbl\_employee(last\_name,email,gender) values(#{lastName},#{email},#{gender})  </insert> |

## 4.4 参数传递

### 4.4.1 参数传递的方式

1. 单个普通(基本/包装+String)参数

这种情况MyBatis可直接使用这个参数，不需要经过任 何处理。

取值:#{随便写}

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0，1…，值就是参数的值

取值: #{0 1 2 …N / param1 param2 ….. paramN}

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

取值: #{自己指定的名字 / param1 param2 … paramN}

1. POJO

当这些参数属于我们业务POJO时，我们直接传递POJO

取值: #{POJO的属性名}

1. Map

我们也可以封装多个参数为map，直接传递

取值: #{使用封装Map时自己指定的key}

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.

取值:

Array: #{array}

Collection(List/Set): #{collection}

List : #{collection / list}

### 4.4.2 参数传递源码分析

1. 以命名参数为例:

|  |
| --- |
| **public** Employee getEmployeeByIdAndLastName  (@Param("id")Integer id, @Param("lastName")String lastName); |

1. 源码:  
   前提: args=[1024,苍老师] 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;  }  } |

### 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 into orcl\_employee(id,last\_name,email,gender) values(employee\_seq.nextval,#{lastName, ,jdbcType=NULL },#{email},#{gender}) --Oracle |

### 4.4.4 参数的获取方式

1. #{key}：可取单个普通类型、 POJO类型 、多个参数、 集合类型

获取参数的值，预编译到SQL中。安全。 PreparedStatement

1. ${key}：可取单个普通类型、POJO类型、多个参数、集合类型.

注意: 取单个普通类型的参数，${}中不能随便写，必须使用 \_parameter

\_parameter 是Mybatis的内置参数.

获取参数的值，拼接到SQL中。有SQL注入问题。 Statement

原则: 能用#{}取值就优先使用#{},#{}解决不了的可以使用${}.

例如: 原生的JDBC不支持占位符的地方，就可以使用${}

Select column1 ,column2… from 表 where 条件group by 组标识 having 条件 order by 排序字段 desc/asc limit x, x

## 4.5 select查询的几种情况

1. 查询单行数据返回单个对象

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

1. 查询多行数据返回对象的集合

|  |
| --- |
| **public** List<Employee> getAllEmps(); |

1. 查询单行数据返回Map集合

|  |
| --- |
| **public** Map<String,Object> getEmployeeByIdReturnMap(Integer id ); |

1. 查询多行数据返回Map集合

|  |
| --- |
| @MapKey("id") // 指定使用对象的哪个属性来充当map的key  **public** Map<Integer,Employee> getAllEmpsReturnMap(); |

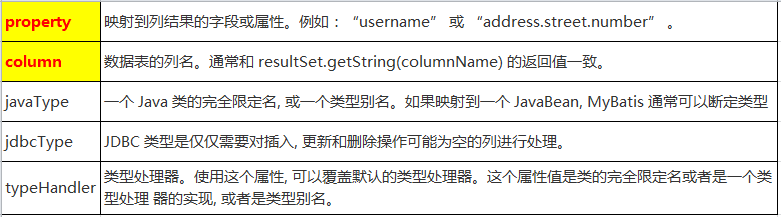
## 4.6 resultType自动映射

1. autoMappingBehavior默认是PARTIAL，开启自动映射的功能。唯一的要求是列名和javaBean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. 自定义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



|  |
| --- |
| <select id=*"getEmployeeById"* resultMap=*"myEmp"*>  select id, last\_name,email, gender from tbl\_employee where id =#{id}  </select>  <resultMap type=*"com.atguigu.mybatis.beans.Employee"* id=*"myEmp"*>  <id column=*"id"* property=*"id"* />  <result column=*"last\_name"* property=*"lastName"*/>  <result column=*"email"* property=*"email"*/>  <result column=*"gender"* property=*"gender"*/>  </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

|  |
| --- |
| **public** **class** Department {  **private** Integer id ;  **private** String departmentName ;  // 省略 get/set方法  } |

|  |
| --- |
| **public** **class** Employee {  **private** Integer id ;  **private** String lastName;  **private** String email ;  **private** String gender ;  **private** Department dept ;  // 省略 get/set方法  } |

1. 使用级联的方式:

|  |
| --- |
| <select id=*"getEmployeeAndDept"* resultMap=*"myEmpAndDept"* >  SELECT e.id eid, e.last\_name, e.email,e.gender ,d.id did, d.dept\_name FROM tbl\_employee e , tbl\_dept d WHERE e.d\_id = d.id AND e.id = #{id}  </select>    <resultMap type=*"com.atguigu.mybatis.beans.Employee"* id=*"myEmpAndDept"*>  <id column=*"eid"* property=*"id"*/>  <result column=*"last\_name"* property=*"lastName"*/>  <result column=*"email"* property=*"email"*/>  <result column=*"gender"* property=*"gender"*/>  <!-- 级联的方式 -->  <result column=*"did"* property=*"dept.id"*/>  <result column=*"dept\_name"* property=*"dept.departmentName"*/>  </resultMap> |

1. Association‘

|  |
| --- |
| <resultMap type=*"com.atguigu.mybatis.beans.Employee"* id=*"myEmpAndDept"*>  <id column=*"eid"* property=*"id"*/>  <result column=*"last\_name"* property=*"lastName"*/>  <result column=*"email"* property=*"email"*/>  <result column=*"gender"* property=*"gender"*/>  <association property=*"dept"* javaType=*"com.atguigu.mybatis.beans.Department"*>  <id column=*"did"* property=*"id"*/>  <result column=*"dept\_name"* property=*"departmentName"*/>  </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。

1. 先通过员工的id查询员工信息
2. 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.

|  |
| --- |
| <select id=*"getEmployeeAndDeptStep"* resultMap=*"myEmpAndDeptStep"*>  select id, last\_name, email,gender,d\_id from tbl\_employee where id =#{id}  </select>    <resultMap type=*"com.atguigu.mybatis.beans.Employee"* id=*"myEmpAndDeptStep"*>  <id column=*"id"* property=*"id"* />  <result column=*"last\_name"* property=*"lastName"*/>  <result column=*"email"* property=*"email"*/>  <result column=*"gender"* property=*"gender"*/>  <association property=*"dept"* select=*"com.atguigu.mybatis.dao.DepartmentMapper.getDeptById"*  column=*"d\_id"* fetchType=*"eager"*>  </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:

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

### 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 departmentName ;  **private** List<Employee> emps ;  } |

1. Collection

|  |
| --- |
| <select id=*"getDeptAndEmpsById"* resultMap=*"myDeptAndEmps"*>  SELECT d.id did, d.dept\_name ,e.id eid ,e.last\_name ,e.email,e.gender  FROM tbl\_dept d LEFT OUTER JOIN tbl\_employee e ON d.id = e.d\_id  WHERE d.id = #{id}  </select>  <resultMap type=*"com.atguigu.mybatis.beans.Department"* id=*"myDeptAndEmps"*>  <id column=*"did"* property=*"id"*/>  <result column=*"dept\_name"* property=*"departmentName"*/>  <!--  property: 关联的属性名  ofType: 集合中元素的类型  -->  <collection property=*"emps"* ofType=*"com.atguigu.mybatis.beans.Employee"*>  <id column=*"eid"* property=*"id"*/>  <result column=*"last\_name"* property=*"lastName"*/>  <result column=*"email"* property=*"email"*/>  <result column=*"gender"* property=*"gender"*/>  </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。

1. 先通过部门的id查询部门信息
2. 再通过部门id作为员工的外键查询对应的部门信息.

|  |
| --- |
| <select id=*"getDeptAndEmpsByIdStep"* resultMap=*"myDeptAndEmpsStep"*>  select id ,dept\_name from tbl\_dept where id = #{id}  </select>  <resultMap type=*"com.atguigu.mybatis.beans.Department"* id=*"myDeptAndEmpsStep"*>  <id column=*"id"* property=*"id"*/>  <result column=*"dept\_name"* property=*"departmentName"*/>  <collection property=*"emps"*  select=*"com.atguigu.mybatis.dao.EmployeeMapper.getEmpsByDid"*  column=*"id"*>  </collection>  </resultMap> |

### 4.7.7 collection 分步查询使用延迟加载

### 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.

### 4.7.9 扩展: association 或 collection的 fetchType属性

1. 在<association> 和<collection>标签中都可以设置fetchType，指定本次查询是否要使用延迟加载。默认为 fetchType=”lazy” ,如果本次的查询不想使用延迟加载，则可设置为

fetchType=”eager”.

1. fetchType可以灵活的设置查询是否需要使用延迟加载，而不需要因为某个查询不想使用延迟加载将全局的延迟加载设置关闭.

# 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=*"getEmpsByConditionIf"* resultType=*"com.atguigu.mybatis.beans.Employee"*>  select id , last\_name ,email , gender  from tbl\_employee  <where>  <if test=*"id!=null"*>  and 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=*"&quot;m&quot;.equals(gender) or &quot;f&quot;.equals(gender)"*>  and gender = #{gender}  </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=*"getEmpsByConditionTrim"* resultType=*"com.atguigu.mybatis.beans.Employee"*>  select id , last\_name ,email , gender  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 = #{lastName} and  </if>  <if test=*"email!=null and email.trim()!=''"*>  email = #{email} and  </if>  <if test=*"&quot;m&quot;.equals(gender) or &quot;f&quot;.equals(gender)"*>  gender = #{gender}  </if>  </trim>  </select> |

## 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=*"updateEmpByConditionSet"*>  update tbl\_employee  <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=*"&quot;m&quot;.equals(gender) or &quot;f&quot;.equals(gender)"*>  gender = #{gender}  </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=*"getEmpsByConditionChoose"* resultType=*"com.atguigu.mybatis.beans.Employee"*>  select id ,last\_name, email,gender from tbl\_employee  <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>  gender = 'm'  </otherwise>  </choose>  </where>  </select> |

## 5.6 foreach

1. foreach 主要用户循环迭代

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

|  |
| --- |
| <select id=*"getEmpsByConditionForeach"* resultType=*"com.atguigu.mybatis.beans.Employee"*>  select id , last\_name, email ,gender from tbl\_employee where id in  <foreach collection=*"ids"* item=*"curr\_id"* open=*"("* close=*")"* separator=*","* >  #{curr\_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 ,gender from tbl\_employee  </sql> |

1. 引用SQL:

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