# Mybatis介绍(了解)

MyBatis本是apache的一个开源项目iBatis, 2010年这个项目由apache software foundation 迁移到了google code，并且改名为MyBatis。2013年11月迁移到Github。

MyBatis是一个优秀的**持久层框架**，它**对jdbc的操作数据库的过程进行封装**，使开发者只需要**关注 SQL 本身**，而不需要花费精力去处理例如注册驱动、创建connection、创建statement、手动设置参数、结果集检索等jdbc繁杂的过程代码。

Mybatis通过xml或注解的方式将要执行的各种statement（statement、preparedStatemnt、CallableStatement）配置起来，并通过java对象和statement中的sql进行映射生成最终执行的sql语句，最后由mybatis框架执行sql并将结果映射成java对象并返回。

# Mybatis架构(重点)



1. **mybatis配置：**

**SqlMapConfig.xml**：**mybatis的全局配置文件**，配置了mybatis的运行环境等信息（文件名不固定）。

**mapper.xml**：**sql映射文件**，文件中配置了操作数据库的sql语句等等。

1. 加载mybatis环境等配置信息构造**SqlSessionFactory，即会话工厂**
2. 由会话工厂创建sqlSession即会话，操作数据库需要通过sqlSession进行。
3. mybatis底层自定义了Executor**执行器**接口操作数据库，Executor接口有两个实现，一个是基本执行器、一个是缓存执行器。
4. **Mapped Statemen**t也是mybatis一个**底层封装对象**(类似一个pojo对象)，它包装了mybatis配置信息及sql映射信息等。**mapper.xml文件中一个sql对应一个Mapped Statement对象，sql的id即是Mapped statement的id。**
5. Mapped Statement对sql执行输入参数进行定义，包括HashMap、基本类型、pojo，Executor通过Mapped Statement在执行sql前将输入的java对象映射至sql中，**输入参数映射就是jdbc编程中对preparedStatement设置参数**。
6. Mapped Statement对sql执行输出结果进行定义，包括HashMap、基本类型、pojo，Executor通过Mapped Statement在执行sql后将输出结果映射至java对象中，**输出结果映射过程相当于jdbc编程中对结果的解析处理过程**。

## mybatis与hibernate不同

|  |
| --- |
| Mybatis和hibernate不同，它不完全是一个ORM框架，因为MyBatis需要程序员自己编写Sql语句。mybatis可以通过XML或注解方式灵活配置要运行的sql语句，并将java对象和sql语句映射生成最终执行的sql，最后将sql执行的结果再映射生成java对象。  Mybatis学习门槛低，简单易学，程序员直接编写原生态sql，可严格控制sql执行性能，灵活度高，非常适合对关系数据模型要求不高的软件开发，例如互联网软件、企业运营类软件等，因为这类软件需求变化频繁，一但需求变化要求成果输出迅速。但是灵活的前提是**mybatis无法做到数据库无关性**，如果需要实现支持多种数据库的软件则需要自定义多套sql映射文件，工作量大。  **Hibernate对象/关系映射能力强，数据库无关性好**，对于关系模型要求高的软件（例如需求固定的定制化软件）如果用hibernate开发可以节省很多代码，提高效率。但是Hibernate的学习门槛高，要精通门槛更高，而且怎么设计O/R映射，在性能和对象模型之间如何权衡，以及怎样用好Hibernate需要具有很强的经验和能力才行。  总之，按照用户的需求在有限的资源环境下只要能做出维护性、扩展性良好的软件架构都是好架构，所以框架只有适合才是最好。 |

# SqlSession的使用范围(了解)

|  |
| --- |
| SqlSessionFactoryBuilder SqlSessionFactoryBuilder用于创建SqlSessionFacoty，SqlSessionFacoty一旦创建完成就不需要SqlSessionFactoryBuilder了，因为SqlSession是通过SqlSessionFactory创建的。所以可以将**SqlSessionFactoryBuilder当成一个工具类**使用，最佳使用范围是方法范围即方法体内局部变量。 SqlSessionFactory SqlSessionFactory是一个接口，接口中定义了openSession的不同重载方法，SqlSessionFactory的最佳使用范围是整个应用运行期间，一旦创建后可以重复使用，**通常以单例模式管理SqlSessionFactory。** SqlSession SqlSession是一个面向用户的接口，sqlSession中定义了数据库操作方法。  **每个线程都应该有它自己的SqlSession实例**。**SqlSession的实例不能共享使用，它也是线程不安全的**。**因此最佳的范围是请求或方法范围。绝对不能将SqlSession实例的引用放在一个类的静态字段或实例字段中.** |

# SqlMapConfig.xml配置文件

只需要关注一下加粗下划线的几个。

## 配置内容

SqlMapConfig.xml中配置的内容和顺序如下：

**(在sqlMapConfig.xml文件中也要按照以下的顺序要求)**

**properties**（属性）

settings（全局配置参数）

**typeAliases**（类型别名）

typeHandlers（类型处理器）

objectFactory（对象工厂）

plugins（插件）

environments（环境集合属性对象）

environment（环境子属性对象）

transactionManager（事务管理）

dataSource（数据源）

**mappers**（映射器）

## properties（属性）

作用：**引用外部配置文件**

<!-- 是用resource属性加载外部配置文件 -->

<propertiesresource=*"db.properties"*>

<propertyname=*"jdbc.username"*value=*"root123"*/>

<propertyname=*"jdbc.password"*value=*"root123"*/>

<!—也可以在properties内部用property定义属性 ，如果**外部配置文件有该属性，则内部定义属性被外部属性覆盖** -->注意：**是外部会覆盖内部！！！**

</properties>

</configuration>

注意：MyBatis将按照下面的顺序来加载属性：

* 在properties 元素体内定义的属性首先被读取。
* 然后会读取properties 元素中resource或url加载的属性，它会覆盖已读取的同名属性。

## typeAliases（类型别名）

### 自定义别名：

在SqlMapConfig.xml中配置后，在mapper.xml配置文件中，就可以使用设置的别名了。并且别名大小写不敏感。

如下：

**<typeAliases>**

<!--**单个别名定义** -->

**<typeAlias type="org.hut.pojo.User" alias="user"/>**

<!--**批量别名定义**，扫描整个包下的类，**别名为类名**（大小写不敏感） -->

**<package name="org.hut"/>**

</typeAliases>

## mappers（映射器）(重点)

Mapper配置的几种方法：

### <mapper resource="" />

**加载的是mapper.xml文件**

如：<mapper resource="sqlmap/User.xml" />（路径是相对src的相对路径）

### <mapper class="" />（不推荐）

**加载的是接口**

如：<mapper class="cn.itcast.mybatis.mapper.UserMapper"/>

注意：此种方法要求mapper接口名称和mapper映射文件名称相同，且放在同一个目录中。**（从它的要求可以看出，加载接口==加载接口对应的mapper.xml文件）**

### <package name=""/>

**加载的是包**

**（可以加载到该包下的所有Mapper.xml文件）**

如：<package name="cn.itcast.mybatis.mapper"/>

注意：只使用mybatis时，此种方法要求mapper接口名称和mapper映射文件名称相同，且放在同一个目录中。

# 输入映射和输出映射

## 输入类型--parameterType

### 传递简单类型

使用#{}占位符，或者${}进行sql拼接。

**#{}可以是任意字母，${}里面必须写value。**

### 传递pojo对象

Mybatis使用ognl表达式解析对象字段的值**，#{}或者${}括号中的值为pojo属性名称**。

### 传递map

Parameter写xxxMap(比如hashmap)，取出方式和pojo一样，写键值即可。

### 传递数组

Parameter可以不写，这时一般会用到foreach标签。

### 传递list

## 输出类型---resultType

## 输出类型---resultMap（重点）

**resultType**可以指定将查询结果映射为pojo，但**要求pojo的属性名和sql查询的列名一致**方可映射成功。

**如果sql查询字段名和pojo的属性名不一致，可以通过resultMap将字段名和属性名作一个对应关系**，resultMap实质上还需要将查询结果映射到pojo对象中。

**resultMap可以实现将查询结果映射为复杂类型的pojo，比如在查询结果映射对象中包括pojo和list实现对一查询和对多查询。**

# 动态sql（重点）

## If标签

|  |
| --- |
| <!-- 动态IF条件 -->  <selectid=*"selectListUseIf"*parameterType=*"Dept"*resultMap=*"deptResultMap"*>  select \* from dept where 1=1  **<if test=*"deptId!=null and deptId!=‘’”"*>**  and dept\_id=#{deptId}  </select> |

## Where标签

上面的sql还有where 1=1 这样的语句，很麻烦

**Where标签的作用：where标签可以自动添加where关键字，同时处理sql语句中and关键字。**

|  |
| --- |
| <!-- 动态Where条件 ,一般也需要与if结合使用，与纯if比较，省略了where 1=1-->  <selectid=*"selectListUseWhere"*parameterType=*"Dept"*resultMap=*"deptResultMap"*>  select \* from dept  <where>  <iftest=*"deptId!=null"*>  and dept\_id=#{deptId}  </if>  <iftest=*"deptName!=null"*>  and dept\_name=#{deptName}  </if>  <iftest=*"deptAddress!=null"*>  and dept\_address=#{deptAddress}  </if>  </where>  </select> |

## choose(when,otherwise)语句

**一般和where一起用。类似switch 语句。**

**如果when中有一个满足条件，则结束choose，**

**如果都不满足，则执行otherwise。**

**使用场景：避免上面使用where+if时，如果条件都不满足，则变成查询所有。**

## 2.4set语句

|  |
| --- |
| <update id="updateUser\_if\_set" parameterType="com.pojo.User">  UPDATE user  **<set>**  **<if** test="username!= null and username != '' ">  username = #{username},  </if>  <if test="sex!= null and sex!= '' ">  sex = #{sex},  </if>  <if test="birthday != null ">  birthday = #{birthday},  </if>  </set>  WHERE user\_id = #{userid};  </update>  **当在 update 语句中使用if标签时，如果前面的if没有执行，则或导致逗号多余错误。使用set标签可以将动态的配置 SET 关键字，并剔除追加到条件末尾的任何不相关的逗号。使用if+set标签修改后，如果某项为 null 则不进行更新** |

## 2.5 Include语句+sql片段

|  |
| --- |
| <insert id="insertDeptUseInclude" parameterType="dept">  insert into dept  <include refid="key"></include>  values  <include refid="value"></include>  </insert>  **<sql id="key">**  <!--suffixOverrides="," 可以忽略最后“，”号 -->  **<trim suffixOverrides="," prefix="(" suffix=")">**  <if test="deptName!=null">dept\_name,</if>  <if test="deptAddress!=null">dept\_address,</if>  </trim>  </sql>  **<sql id="value">**  **<trim suffixOverrides="," prefix="(" suffix=")">**  <if test="deptName!=null">#{deptName},</if>  <if test="deptAddress!=null">#{deptAddress},</if>  </trim>  </sql>  **trim：prefix前缀、suffix后缀、prefixOverrides可以去除头部的指定内容、suffixOverrides可以去除尾部的指定内容**  **如果要使用别的Mapper.xml配置的sql片段，可以在refid前面加上对应的Mapper.xml的namespace** |

## foreach标签

|  |
| --- |
| <!-- 根据ids查询用户 -->  <selectid="queryUserByIds"parameterType="queryVo"resultType="user">  SELECT \* FROM user  <where>  **<foreach collection="ids" item="item" open=" (" close=")"**  **separator=",">**  **id IN** #{item}  </foreach>  </where>  </select>  <!-- foreach标签，进行遍历 -->  **<!-- collection：遍历的集合，这里是QueryVo的ids属性 -->**  **collection中遍历的名称规定：**  **1. 数组：用array**  **2. 集合：用list**  **3. 属性：直接用属性**  <!-- item：表示集合（数组）中的项目，可以随便写什么，如果里面是pojo的话，下面可以用.属性的方式如果#{item.name} -->  <!-- open：在前面添加的sql片段 -->  <!-- close：在结尾处添加的sql片段 -->  <!-- separator：指定遍历的元素之间使用的分隔符 --> |

# 关联查询（重点）

## 对一查询——association

一对一或者多对一

|  |
| --- |
| *<resultMap type="order" id="orderUserResultMap">*  *<id property="id" column="id"/>****主键***  *<result property="userId" column="user\_id"/>****普通字段***  *。。。*  ***<!--association ：配置一对一属性 -->***  *<association property="user" javaType="user">*  ***Association的多种写法，见下面***  *</association>*  *</resultMap>* |

***Association写法：***

**第一种方式：直接写次表的映射关系**

|  |
| --- |
| <association property="属性" javaType="类型（**别名也可以**）">  ***以下为次表的映射关系：***  <id column="主表的字段"property="次表的类的字段"></id>  。。。。 |

**第二种方式：给association传入一个ResultMap：**

|  |
| --- |
| <association property="customer" javaType="Customer"  resultMap="org.hut.dao.CustomerMapper.BaseResultMap"></association>  **和第一种方式不一样的地方在于：由外部导入次表的映射关系。（但是要注意的是association里的ResultMap与自己所处的ResultMap里的column列名有没有重复。不过这种情况也比较少）**  **导入外部的resultMap：nameSpace+resultMap的id**  **这种方式在双向关联时会有一个弊端：进入死循环，StackOverFlowError堆栈内存溢出。**  **解决办法：拆分resultMap为：一个基础resultMap和一个扩展的resultMap（继承基础resultMap）。**  **或者使用其他两种方式。** |

**第三种方式：给association传入一个select**

|  |
| --- |
| <association property="customer" column="cust\_id" select="org.hut.dao.CustomerMapper.selectByPrimaryKey"></association>  **这里的column表示要传到select代表的sql语句的字段。如果需要多个字段可以使用column= “{prop1=col1,prop2=col2}”** |

## 对多查询（一对多或者多对多）

**一对一可以看做一对多的特例。**

|  |
| --- |
| 一对一的三种方式同样适用于一对多。  第一种方式上有区别：使用ofType指定类型。  **<collectionproperty=*"orders"*javaType=*"list"*ofType=*"order"*>** |

# 缓存

|  |
| --- |
| **一级缓存：**  mybatis的一级缓存是SqlSession级别的缓存，在操作数据库的时候需要先创建SqlSession会话对象，在对象中有一个HashMap用于存储缓存数据，此HashMap是当前会话对象私有的，别的SqlSession会话对象无法访问。  源码：  https://images2017.cnblogs.com/blog/663108/201709/663108-20170912191237032-1438769206.png  **具体流程：**  　　1.第一次执行select完毕会将查到的数据写入SqlSession内的HashMap中缓存起来  　　2.第二次执行select会从缓存中查数据，如果select相同切传参数一样，那么就能从缓存中返回数据，不用去数据库了，从而提高了效率  **注意事项：**  　　　1.如果SqlSession执行了DML操作（insert、update、delete），并commit了，那么mybatis就会清空当前SqlSession缓存中的所有缓存数据，这样可以保证缓存中的存的数据永远和数据库中一致，避免出现脏读  　　　2.当一个SqlSession结束后那么他里面的一级缓存也就不存在了，mybatis默认是开启一级缓存，不需要配置 |

|  |
| --- |
| **二级缓存：**  二级缓存是mapper级别的缓存。  **具体流程：**  　　　1.当一个sqlseesion执行了一次select后，在关闭此session的时候，会将查询结果缓存到二级缓存  　　　2.当另一个sqlsession执行select时，首先会在他自己的一级缓存中找，如果没找到，就回去二级缓存中找，找到了就返回，就不用去数据库了，从而减少了数据库压力提高了性能  **注意事项：**  　　　1.如果SqlSession执行了DML操作（insert、update、delete），并commit了，那么mybatis就会清空当前mapper缓存中的所有缓存数据，这样可以保证缓存中的存的数据永远和数据库中一致，避免出现脏读  **开启二级缓存：**  需要开启全局缓存，文件级缓存，语句级缓存，才能使用二级缓存。**默认情况下文件级缓存没有开启，全局缓存和语句级缓存都是开的**。（全局>文件级>语句级，全局是最大的，如果全局缓存关了，则文件级和语句级也不起作用了。）  全局缓存：  <settings>  <setting name="cacheEnabled" value="true"/>默认是true：打开缓存  <settings>  文件级缓存：  <cache eviction="LRU"flushInterval="60000" size="512"/>  这里配置了一个LRU缓存，并每隔60秒刷新，最大存储512个对象。不写这些配置也可以，只写cache标签也算是开的。  LRU –移除最近最少使用的缓存。  FIFO – 先进先出:按对象进入缓存的顺序来移除它们。  语句级缓存：  <select…省略其他….useCache="true"> |

# 批量处理

1. 传入参数会是集合或者数组，但是parameterType不用指定具体类型，因为批量处理绝对会用到foreach标签，里面的collection会指定传入的参数。
2. 批量删除、批量修改、批量增加。百度

# Mybatis整合spring

## 整合思路

1. SqlSessionFactory对象应该放到spring容器中作为单例存在。
2. Mapper代理形式中，应该从spring容器中直接获得mapper的代理对象。

|  |
| --- |
| 之前的操作：  先写好mapper.xml文件和sqlMapConfig核心配置文件。然后做好映射，接下来就是关键点，也是Spring整合Mybatis的核心内容，获取XXXMapper对象（即mapper.xml文件对应的接口）：  SqlSessionFactoryBuilderbuilder = **new**SqlSessionFactoryBuilder();  InputStreaminputStream = Resources.*getResourceAsStream*("sqlMapConfig.xml");  SqlSessionFactoryfactory = builder.build(inputStream);  SqlSessionsession = factory.openSession();  UserMappermapper = session.getMapper(UserMapper.**class**);  我们的最终目的：是得到xxxMapper的Dao接口对应的mapper对象，总结其需要的因素（也就是“属性Property”）：   1. 需要SqlSessionFactory，也就是Mybatis的工厂，所以首先需要配置这个工厂：   而配置这个工厂需要的Property有：  连接池、配置文件sqlMapCofig  配置文件我们在上面的代码就可以看得出来是需要的，而连接池在Mybatis的核心配置文件sqlMapCofig中的环境配置可以看出是是需要连接池。  2．有了SqlSessionFactory我们就可以得到session,此时还需要指定我们要得到的“类”才可以得到我们需要的mapper.  依赖关系:  Mapper  |-sqlSessionFactory  |-连接池  |-C3P0/DBCP/druid  |-JDBC的配置文件  |-Mybatis的配置文件  |-指定类  配置单个mapper：  <bean name="userDao" class="org.mybatis.spring.mapper.**MapperFactoryBean**">  <property name="**sqlSessionFactory**" ref="sqlSessionFactory"></property>  <property name="**mapperInterface**" value="org.hut.dao.UserDao"></property>  </bean>  --------------------------------------------------------------  对上述方法的增加----**扫描包形式**配置mapper  <beanclass="org.mybatis.spring.mapper.**MapperScannerConfigurer**">  <!-- 配置Mapper接口 -->  <propertyname="**basePackage**"value="cn.itcast.mybatis.dao"/>  <--**不需要配置sqlSessionFactory,Spring对于这种包形式的配置会自动查找SqlSessionFactory.如果有多个sqlSessionFactory，则需要指定。**-->  </bean>  解释：这种方式会扫描basepackage包下的所有**dao接口类。**   1. 如果mapper.xml文件名和dao接口类的类名一致且在一个包中：则会自动查找指定的mapper.xml配置文件。 2. 如果文件名不一致：则需要配置mapperLocations，告诉Spring，mapper.xml文件的所在位置。之后会根据mapper.xml文件中的namespace来查找对应的类。（注：**MapperScannerConfigurer扫描的是dao类，如果dao类和mapper.xml文件不同名或不在同一个包中，它是找不到对应的mapper.xml文件的**）   在sqlSessionFactory中配置如下：  <beanname=*"sqlSessionFactory"*class=*"org.mybatis.spring.SqlSessionFactoryBean"*>  <propertyname=*"configLocation"*value=*"classpath:sqlMapConfig.xml"*></property>  <propertyname=*"dataSource"*ref=*"dataSource"*></property>  **<propertyname=*"mapperLocations"*>**  **<list>**  **<value>classpath\*:org/hut/dao/\*.xml</value>**  **</list>**  **</property>**  </bean>  **注：**  **1. 每个mapper代理对象的id就是类名，首字母小写。也可以通过getMapper(Class)的方式获取**  **2.mapper.xml文件和dao类时通过namespace对应的。所以只要能保证：扫到了mapper.xml文件和扫到dao类和 namespace对应了类路径就可以了。** |

## 整合的步骤

### 导入jar包

1. spring的jar包
2. Mybatis的jar包
3. Spring+mybatis的整合包。
4. 数据库驱动jar包。
5. 数据库连接池的jar包。

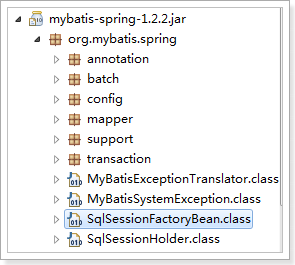
### 加入配置文件

#### SqlMapConfig.xml

|  |
| --- |
| <?xmlversion=*"1.0"*encoding=*"UTF-8"*?>  <!DOCTYPEconfiguration  PUBLIC"-//mybatis.org//DTD Config 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-config.dtd"> |

#### applicationContext.xml

整合Mybatis需要的是SqlSessionFactoryBean，位置如下图：



applicationContext.xml，配置内容如下

|  |
| --- |
| <context:property-placeholderlocation=*"classpath:db.properties"*/>  <!-- 连接池 -->  <beanid=*"dataSource"*class=*"org.apache.commons.dbcp.BasicDataSource"*>  <propertyname=*"driverClassName"*value=*"${jdbc.driver}"*/>  <propertyname=*"url"*value=*"${jdbc.url}"*/>  <propertyname=*"username"*value=*"${jdbc.username}"*/>  <propertyname=*"password"*value=*"${jdbc.password}"*/>  <propertyname=*"maxActive"*value=*"10"*/>  <propertyname=*"maxIdle"*value=*"5"*/>  </bean>  <!-- *sqlSessionFactory* -->  <beanname=*"sqlSessionFactoryBean"*class=*"org.mybatis.spring.SqlSessionFactoryBean"*>  2个属性：mybatis的核心配置文件+连接池  <propertyname=*"configLocation"*value=*"classPath:sqlMapConfig.xml"*></property>  <propertyname=*"dataSource"*ref=*"dataSource"*></property>  补充一个属性：mapperLocations，用于加载mapper.xml文件  <propertyname=*"mapperLocations"*>  <list> <value>classpath\*:org/hut/biz/admin/book/dao/\*.xml</value>  <value>classpath\*:org/hut/biz/admin/user/dao/\*.xml</value>  </list>  **（单个value，直接value=…，多个属性时用list+value标签）**  </property>  </bean>  </beans>  注意：Mybatis的SqlSessionFactory是需要两个property的，其中包括连接池！！！因为本身Mybatis的配置文件中也有配置连接池。 |

#### db.properties

#### log4j.properties

## Dao的开发

两种dao的实现方式：

1. 原始dao的开发方式
2. 使用Mapper代理形式开发方式
   1. 直接配置Mapper代理
   2. 使用扫描包配置Mapper代理

### Mapper代理形式开发dao

#### 方式一：单个的形式配置mapper代理

在applicationContext.xml添加配置

**MapperFactoryBean**也是属于mybatis-spring整合包

<!-- Mapper代理的方式开发方式一，配置单个Mapper代理对象 -->

<beanid=*"userMapper"*class=*"org.mybatis.spring.mapper.****MapperFactoryBean****"*>

<!-- 配置Mapper接口 -->

<propertyname=*"mapperInterface"*value=*"cn.itcast.mybatis.mapper.UserMapper"*/>

<!-- 配置sqlSessionFactory -->

<propertyname=*"sqlSessionFactory"*ref=*"sqlSessionFactory"*/>

</bean>

#### 方式二：扫描包形式配置mapper

<!-- Mapper代理的方式开发方式二，扫描包方式配置代理 -->

<beanclass=*"org.mybatis.spring.mapper.MapperScannerConfigurer"*>

<propertyname=*"basePackage"*value=*"cn.itcast.ssm.mapper"*/>

1. **如果需要加载多个包，直接写进来，中间用，分隔**
2. 会扫描子孙包

</bean>

**每个mapper代理对象的id就是类名，首字母小写**