本文档仅记录eclipse上基于maven项目的使用mybatis的接口编程方式的使用方法。pom.xml中加入：

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis</artifactId>

<version>3.4.6</version>

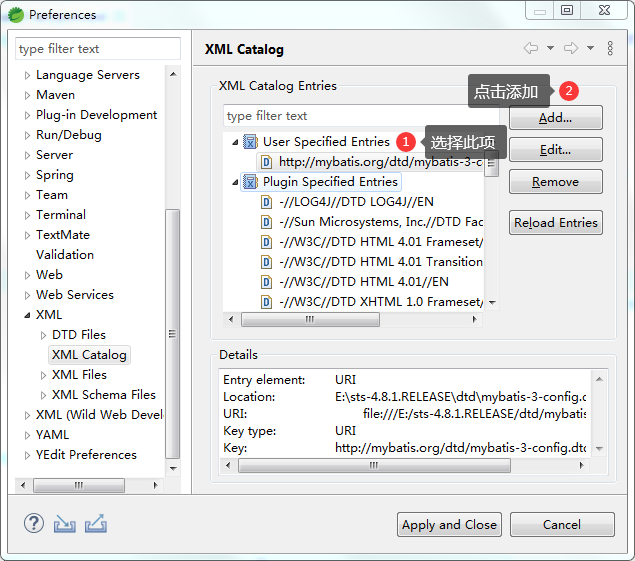
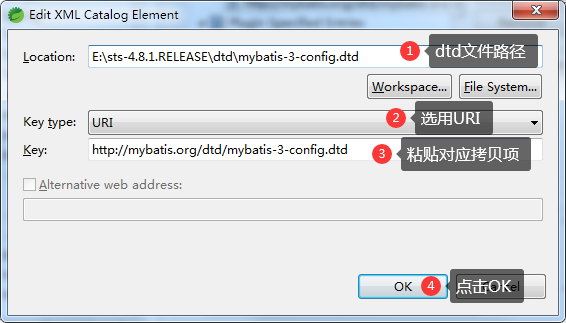
</dependency>

# mybatis

## eclipse配置xml自动提示

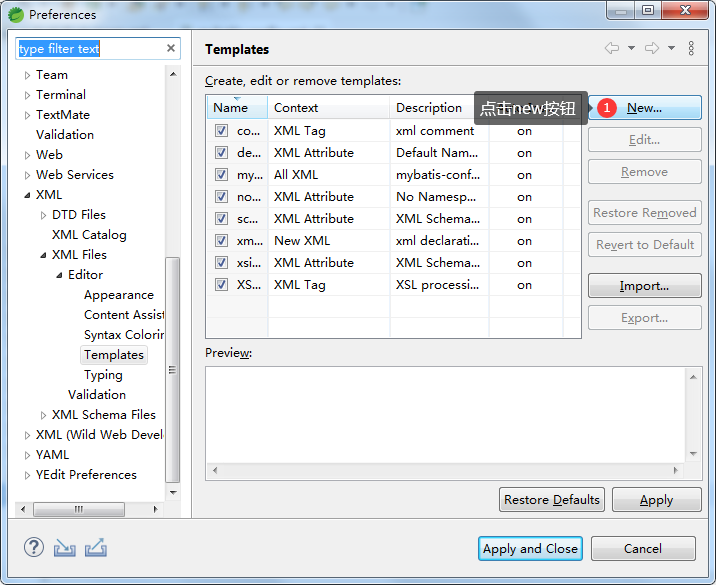
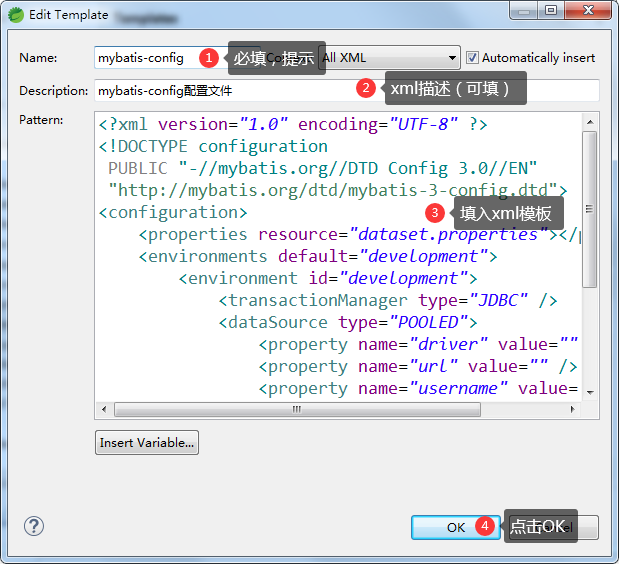
1. 打开mybatis.jar找org\apache\ibatis\builder\xml路径下的mybatis-3-config.dtd和mybatis-3-mapper.dtd。拷贝这两文件到eclipse安装目录中。
2. 拷贝：http://mybatis.org/dtd/mybatis-3-config.dtd和http://mybatis.org/dtd/mybatis-3-mapper.dtd（根据mybatis版本修改红色字）。

eclipse：Window → Perferences → XML → XML Catalog

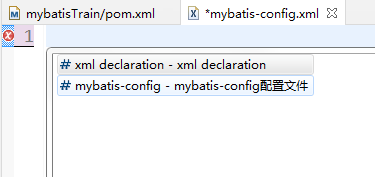
 

## eclipse配置xml自动生成模板

eclipse: Window → Preference → XML→ XML Files → Editor → Templates

使用模板：创建xml后，快捷键：alt + /调出自定义模板选择框（如下图）



## mybatis-config.xml（重要）

mybatis-config.xml是mybatis的重要且基础配置文件，基础格式如下。

<?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=*""* />

<property name=*"url"* value=*""* />

<property name=*"username"* value=*""* />

<property name=*"password"* value=*""* />

</dataSource>

</environment>

</environments>

<mappers>

<mapper resource=*""* />

<mapper class=*""* />

<mapper url=*""* />

</mappers>

</configuration>

### environments

environments下可配置多个环境，用default指定项目运行时使用哪个环境，default所带值和众多environment的id必须吻合一个。



<transactionManager type="JDBC" />：事务管理器，type可选值JDBC,MANAGED。

JDBC:

MANAGED:

<dataSource type="POOLED">：数据源，type可选值UNPOOLED，POOLED，JNDI。

UNPOOLED:为每一个SQL请求开一个新连接，使用完毕后关闭。

POOLED:mybatis创建一个连接池，SQL请求用完归还入池。

JNDI:

<mapper resource/class/url="" />：mapper中属性值可选

resource：

class：

url：

### 引入properties配置文件

添加配置<properties resource="dataset.properties"></properties>数据库配置就可以写在properties文件中了，应用格式如下（重点看红字）

dataset.properties配置文件

jdbc.driver=..

jdbc.url=...

jdbc.username=...

jdbc.password=...

mybatis-config.xml配置文件

**<properties resource=*"dataset.properties"*></properties>**

<environments default=*"development"*>

<environment id=*"development"*>

<transactionManager type=*"JDBC"* />

<dataSource type=*"POOLED"*>

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

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

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

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

</dataSource>

</environment>

</environments>

### settings配置mybatis

mybatis-config.xml配置

### typeAliases配置java类别名

#### 配置单个别名

alias就是别名名称。

<typeAliases>

<typeAlias type=*"com.pojo.Person"* alias=*"Per"*/>

</typeAliases>

除了xml中配置方式，在pojo类中使用@Alias("别名称")也可以达到同样目的。

#### 配置包下所有类的别名

Mybatis扫描该包下所有类纳入别名命名，规则是大写首字母

<typeAliases>

<package name=*"com.pojo"* />

</typeAliases>

### 利用plugins配置mybatis插件

## mapper.xml

xml中判断符号转义符号对照

|  |  |  |
| --- | --- | --- |
| < | &lt; | 小于 |
| > | &gt; | 大于 |
| & | &amp; | 与 |
| <= | &lt;= | 小于等于 |
| >= | &gt;= | 大于等于 |
| ‘ | &apos | 单引号 |
| “ | &quot; | 双引号 |

mapper.xml是SQL映射文件，注意事项如下：

1. mapper.java中接口方法必须和mapper.xml中的操作名一样且相同
2. Mapper.java中接口方法的参数必须和mapper.xml中的参数类型一样

基础格式如下:

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

<!DOCTYPE mapper

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

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

<mapper namespace=*"com.mybaties.EmployeeMapper"*>

<insert id=*""*>...</insert>

<delete id=*""*>...</delete>

<select id=*""* resultType=*""*>...</select>

<update id=*""*>...</update>

</mapper>

### insert

#### 简单添加

Mapper.java

public void insertAPerson1(@Param("id") Integer id,

@Param("name") String name,

@Param("age") Integer age,

@Param("email") String email);

Mapper.xml

<insert id="insertAPerson" parameterType="Person">

insert into person(name,age,email,id) values ( #{name}, #{age},#{email},#{id} )

</insert>

#### 单对象添加

单对象添加，映射接口中方法的参数可以写对象，但是在SQL语句中还需要手动对齐字段。

Mapper.java

public void insertAPerson(@Param("person") Person person);

Mapper.xml

<insert id="insertAPerson" parameterType="com.pojo.Person">

insert into person(name,age,email,id) values

(#{person.name}, #{person.age}, #{person.email}, #{person.id} )

</insert>

#### 对象集添加

Mapper.java

public void insertPersons(@Param("persons") List<Person> persons);

Mapper.xml

<insert id="insertPersons" parameterType="**java.util.List**" >

insert into person(name,age,email,id) values

<foreach collection="persons" item="item" index="index" separator="," >

( #{item.name},#{item.age},#{item.email},#{item.id} )

</foreach>

</insert>

#### 不对应添加

在数据表字段与实体对象属性不对应的前提下，手动对应即可。

Mapper.java

略

Mapper.xml

略

### delete

Mapper.java

public void deletePerson(@Param("id") Integer id);

Mapper.xml

<delete id="deletePerson">

delete from person where id = ${id}

</delete>

### select

#### 普通查询

Mapper.java

public Person selectAPerson(Integer id)

Mapper.xml

<select id="selectAPerson" resultType="com.pojo.Person">

select \* from person where id = #{id}

</select>

注意:sql中查询的字段若不包含所有pojo中所有属性也不会出错。

#### 查询对象集

条件是所查字段必须与对象属性对应

Mapper.java

public List<Person> selectPersons(@Param("id") Integer id);

Mapper.xml

<select id="selectPersons" resultType="com.pojo.Person">

select \* from person where id > #{id}

</select>

#### resultMap查询不对应的对象集

数据表字段和实体属性并不一一对应，使用resultMapper进行手动对齐。Column是数据表字段名，property是java类属性。

Mapper.java

public List<Person> selectPersons(@Param("id") Integer id);

Mapper.xml

<!--property是java类属性。column是数据库字段-->

<resultMap type="com.pojo.Person" id="resultMapPerson">

<result column="id" property="id" javaType="java.lang.String" jdbcType="VARCHAR"/>

<result column="name" property="name" javaType="java.lang.String" jdbcType="VARCHAR"/>

<result column="age" property="age" javaType="java.lang.Integer" jdbcType="INTEGER"/>

<result column="email" property="email" javaType="java.lang.String" jdbcType="VARCHAR"/>

</resultMap>

<select id="selectPersons1" resultMap="resultMapPerson">

select \* from person where id > #{id}

</select>

注意：javaType和jdbcType分别是java类型和数据表类型，相同的情况下可以忽略不写，但为了严谨，建议写上。此外，若java类中多余属性并没有对应的查询结果，也不会影响，mybatis会自动忽略掉Java类中没有对应的属性。如下是javaType和jdbcType对应关系参考:

|  |  |  |
| --- | --- | --- |
| Java | MySQL | Oracle |
| java.lang.String | char |  |
|  | varchar |  |
|  | longtext |  |
| java.lang.Integer | int |  |
|  | integer |  |
| java.lang.Double |  |  |
|  |  |  |
| Java.math.BigDecimal | decimal（） |  |
|  | numeric（） |  |
| java.lang.Boolean | bit(1:true,0:false) |  |
| java.sql.Date | date |  |
|  | datetime |  |

#### 分页

##### 用SQL分页

简单利用SQL提供的语句进行分页，但是在web应用中，这样的分页无法做到查询后一同返回对应页数值。

Mapper.java

/\*current：当前页，pagesize：每页数量\*/

public List<Person> getByPage(@Param("current") Integer current,@Param("pagesize") Integer pagesize);

Mapper.xml

<select id="selectPersonsPage" resultType="Person" >

select <include refid="personfield"></include> from person

order by id limit **#{current},#{pagesize}**

</select>

##### ipage分页（mybatis-plus提供）

略

### update

#### 普通更新

Mapper.java

public void modName(@Param("id") Integer id,@Param("name") String name)

Mapper.xml

<update id="modName">

update person set name = #{name} where id = #{id}

</update>

#### 更新单对象

Mapper.java

public void modPerson(@Param("person") Person person);

Mapper.xml

<update id="modPerson" parameterType="Person">

update person set name=#{person.name},email=#{person.email},age=#{person.age}

where id = #{person.id}

</update>

#### 更新对象集

更新对象集即可使用对象list也可使用Map<name,List>。

mybatis-config.xml

<dataSource type="POOLED" >

<property name="driver" value="com.mysql.cj.jdbc.Driver" />

<property name="url" value="jdbc:mysql://localhost:3306/train?characterEncoding=utf-8

&amp;useSSL=false&amp;serverTimezone=UTC&amp;**allowMultiQueries=true**" />

<property name="username" value="root" />

<property name="password" value="root" />

</dataSource>

mapper.java

public void modPersons(@Param("persons") List<Person> persons);

mapper.xml

<update id="modPersons" parameterType="java.util.List">

<foreach collection="persons" index="index" item="item" separator=";" close=";">

update person set name=#{item.name},email=#{item.email},age=#{item.age}

where id = #{item.id}

</foreach>

</update>

### 注释

Mapper.xml文件使用<!-- -->打注释，不能用SQL注释形式“- -”

### foreach

配置在sql语句中用于循环组织sql语句。

例如：

<foreach collection="persons" index="index" item="item" separator=";" open="" close=";">

update person set name=#{item.name} where id = #{item.id}

</foreach>

collection:遍历项，需要接口传递进来，通常是list，list的实现类，数组

index:下标（从0开始），可以在sql语句中使用

item:遍历项,在sql中的使用格式为：item.属性

separator:分隔符号

open:开始字符串

close:结束字符串

小记：参数为数组时，mapper.xml中根据下标也可以获取值。

mapper.java

List<Goods> selectGoodsByName(@Param("names") String[] names);

mapper.xml

<select id="selectGoodsByName" resultType="com.pojo.Goods">

select <include refid="allFields"></include> from goods where name in (#{names[0]},#{names[1]})

</select>

### if

配置在sql语句中,mybatis将根据条件动态选择sql执行。

<select id="selectGrade" resultType="Person">

select sum(score) from grade where id = #{id}

<if test="timer == day">group by substr(time,1,11)</if>

<if test="timer != day">group by name</if>

</select>

注意：test中不能写#{...}格式，数字就写成数字，不用加单引号；另外比较也可以用如下形式的代码替换比较符gt：大于，lt：小于，gte：大于等于，lte：小于等于，e：等于。

### sql

可被其他语句引用的可重用语句块;查询，插入都涉及到指定字段，重复写浪费时间。使用<include refid="">调用即可。

Mapper.xml

<sql id="personfield">

name,age,email,id

</sql>

<!--调用-->

<select id="selectAPerson" resultType="Person">

select

<include refid="personfield"></include>

from person where id = #{id}

</select>

<insert id="insertAPerson1">

insert into person(<include refid="personfield"></include> )

values ( #{name}, #{age},#{email},#{id} )

</insert>

### bind

条件格式化拼接，将传入的值自定义一个拼接模板，写SQL时加入模板即可。

Mapper.xml

<bind name="**bindID**" value="'%'+id+'%'"/>

<select id="selectById" resultType="com.heiketu.pojo.Users">

<bind name="abc" value="id"/>

select \* from usrs where id = #{**bindID**}

</select>

### where

条件域，可省略手写sql中where。where标签使用于条件if判断较多的SQL中，会自动去掉或保留if中包裹的SQL的头部and。

Mapper.xml

<select id="selectGoodsByName" resultType="com.pojo.Goods">

select \* from goods

<where>

name in (#{names[0]},#{names[1]})

</where>

</select>

### set

常用于Update的语句。去掉其中if包裹的sql的最后一个sql尾部的逗号。

Mapper.xml

<update id="updateGoods" parameterType="com.pojo.Goods">

update goods

<set>

<if test=" newgoods.price gte 10">

price = #{newgoods.price},

</if>

<if test=" newgoods.total >= 0">

total = #{newgoods.total},

</if>

</set>

<where>

name = #{newgoods.name}

</where>

</update>

### trim

SQL前后格式自定义，若多条判断语句，and 关键字都在sql之后，用trim解决，最后一个and无法去掉的问题。<trim prefix="" suffix="" prefixOverrides="" suffixOverrides="">

Mapper.xml

<select id="selectGoodsByName" resultType="com.pojo.Goods">

select <include refid="allFields"></include> from goods

<trim prefix="where" prefixOverrides="" suffixOverrides="or">

<if test="names[0] == '饼干'">

name = #{names[0]} or

</if>

<if test="names[1] == '火腿肠'">

name = #{names[1]} or

</if>

</trim>

</select>

说明：

prefix：加整体SQL前缀

suffix：加整体SQL后缀

prefixOverrides：去掉最开始一条if sql的前缀

suffixOverrides：去掉最末尾一条if sql的后缀

### choose

类似于程序的switch…case…

Mapper.xml

<select id="selectGoodsByName" resultType="com.pojo.Goods">

select <include refid="allFields"></include> from goods

<where>

<choose>

<when test="names[0] == '饼干'">name = #{names[0]}</when>

<when test="names[0] == '果汁'">name = #{names[1]}</when>

<otherwise></otherwise>

</choose>

</where>

</select>

### \_parameter

Mybatis内置参数;\_parameter:参数全集,\_databaseId:数据库产品id，如mysql，Oracle,使用时写在#{...}中即可。

### \_databaseId数据库厂商标识

内置参数，与配置<databaseIdProvider>结合使用；供Mybatis自动判断所用数据库类型，从而执行不同的SQL语句。

Mybatis-config.xml

<databaseIdProvider type="DB\_VENDOR">

<property name="MySQL" value="mysql"/>

<property name="Oracle" value="oracle"/>

</databaseIdProvider>

Mapper.xml

<select id="selectAll" resultType="com.pojo.Goods" databaseId="mysql">

select \* from goods where price > 4

</select>

<select id="selectAll" resultType="com.pojo.Goods" databaseId="oracle">

select \* from goods where price &lt; 4

</select>

2个select的id是相同的，databaseId不同，mybatis可自动判断在不同的数据库中执行哪一个SQL

### 一级缓存

mybatis一级缓存在sqlSession建立时建立，sqlSession close时关闭。数据被查询出来后并不会立即在内存上被抹掉，而是暂存在内存中；后面所执行内容与本次相同，则程序不再从数据库查询而是直接拿取缓存中的数据，这种制度叫做缓存。缓存失效的情况：

1. 所查内容与原内容不同。
2. 两次查询过程中有涉及增、删、改操作
3. Sqlsession不同

### 二级缓存

一个namespace具备一个二级缓存。sqlSession会话关闭时一级缓存的数据进入二级缓存中。通常情况下项目还会引入Redis用于管理缓存，这块了解即可。

二级缓存使用：

Mybatis-config.xml

<settings>

<setting name="cacheEnabled" value="true"/><!-- 加入二级缓存 -->

</settings>

Mapper.xml

<cache eviction="FIFO" flushInterval="60000" readOnly="true" size="10000"></cache>

|  |  |  |
| --- | --- | --- |
| eviction: | 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（默认）  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  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 |
| flushInterval | 缓存刷新间隔 | 毫秒级。多长时间清空一次，默认不清空 |
| readOnly |  | true：只读；mybatis认为所有从缓存中获取数据都是只读操作，不会修改数据。Mybatis为了加快获取速度，直接将数据在缓存中的引用交给用户。（不安全，速度快）  false：非只读（默认）。Mybatis觉得获取的数据可能会被修改，mybatis会利用序列化技术克隆一份新的数据。（安全，速度慢） |
| blocking |  |  |
| size | 缓存大小 | Byte |
| type | 自定义缓存全类名 | 继承Cache即可，可以不定义 |

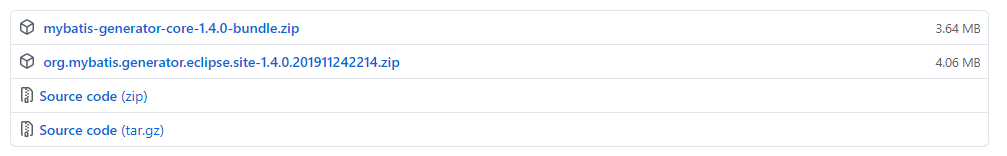
### 事务

### 存储过程

## mybatis逆向工程

从数据库中已存在的表为参照自动生成对应的Java bean，Mapper.xml，Mapper.java。

### eclipse配置dtd及xml文件

1. github下载**第二个**zip包（https://github.com/mybatis/generator/releases/tag/mybatis-generator-1.4.0）。
2. 解压zip包，找到target和plugins两个文件夹，对应着把这两文件夹中内容拷贝到eclipse安装目录下的target和plugins中。重启eclipse，在新建里可以见到mybatis generator。
3. 配置dtd的步骤和mybatis-config.dtd步骤一样，从jar包中找dtd文件配置进eclipse。

### mgc.xml

# mybatis-plus

在mybatis基础上只做增强，不做更改。基于springboot的Pom.xml中引入依赖：

<dependency>

<groupId>com.baomidou</groupId>

<artifactId>mybatis-plus-boot-starter</artifactId>

<version>3.4.1</version>

</dependency>

基于springMVC引入依赖：

<dependency>

<groupId>com.baomidou</groupId>

<artifactId>mybatis-plus</artifactId>

<version>3.4.1</version>

</dependency>