Mybatis

这里采用的是机器人的列子 必须要上慕课网看视频才知道讲的是什么

Servlet JQuery Mybais JSP **JSTL EL** JDBC 采用了这些技术

基本内容：接受发送指令
根据指令自动回复对应的内容

模块划分

回复功能划分
对话功能
回复内容列表：
回复内容删除

页面放在WEB-INFO下面是为了方便管理便于控制

代码不能少注释，每个类的上面都要有注释，string ，stringBuffer ，StringBuilder要分情况用

1 代码量繁多，
2一个方法堆积

DAO层对象负责
1 对象能与数据库交互
2 能执行SQL语句

Mybatis之SqlSession
@·Mybatis之sqlsession
作用
1.向SQL语句传入参数
2.执行SQL语句
3.获取执行SQL语句的结果
4.实务的控制

sqlsession的作用
1 向sql语句传入参数
2 执行SQL语句
3 获取执行SQL语句的结果
4 事务的控制

如何获得SQLsession

1 通过配置文件获取数据库连接相关信息
2 通过配置信息构建SqlsessionFactory
3通过SqlsessionFactory打开数据库会话

SqlSession与JDBC差不多

SQL配置

这是Configuration.xml（主要配置xml）

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

<!DOCTYPE configuration
  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>
<!--
  <settings>
    <setting name="useGeneratedKeys" value="false"/>
    <setting name="useColumnLabel" value="true"/>
```

```
  </settings>

  <typeAliases>
    <typeAlias alias="UserAlias" type="org.apache.ibatis.submitted.complex_property.User"/>
  </typeAliases>
  -->

  <environments default="development">
    <environment id="development">
      <transactionManager type="JDBC">
        <property name="" value=""/>
      </transactionManager>
      <dataSource type="UNPOOLED">
        <property name="driver" value="com.mysql.jdbc.Dirver"/>
        <property name="url" value="jdbc:mysql://localhost:3306/t2" />
        <property name="username" value="root"/>
        <property name="password" value="199666"/>
      </dataSource>
    </environment>
  </environments>

  <mappers>
    <mapper resource="com/imooc/config/sql/Message.xml"/>
  </mappers>

</configuration>
```

在user.xml中进行相关配置

这是一个例子的文档 Message.xml，用户配置的文档（**用户配置XML**）

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

<!DOCTYPE mapper
    PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="Message">

  <resultMap type="com.imooc.bean.Message" id="MessageResult">
    <id column="ID" jdbcType="INTEGER" property="id"/>
    <result column="command" jdbcType="VARCHAR" property="Command"/>
    <result column="description" jdbcType="VARCHAR" property="description"/>
    <result column="content" jdbcType="BOOLEAN" property="content"/>
  </resultMap>

  <select id="queryMessage" parameterType="com.imooc.bean.Message " resultMap="MessageResult">
  select id,command,descroption,content from message where 1=1
  <if test="(command!=null and &amp;amp;&amp;amp; &amp;quot; &amp;quot;.equals(command.trim()))">and Command=#{command}</if>
   <if test="(description!=null and &amp;amp;&amp;amp; &amp;quot; &amp;quot;.equals(description.trim()))">and descroption=like '%' #
{description} '%' </if>
  </select>

  <select id="version" parameterType="long" resultType="int">
    SELECT version FROM user WHERE id = #{id,jdbcType=INTEGER}
  </select>


<delete id="deleteOne" parameterType="int">
delete  form MESSAGE where ID=#{_parameter
}
</delete>
  <delete id="delete" parameterType="UserAlias">
    DELETE FROM user WHERE id = #{id:INTEGER}
  </delete>

  <insert id="insert" parameterType="UserAlias" useGeneratedKeys="false">
    INSERT INTO user
```

```
  ( id,
  username,
  password,
  administrator
  )
  VALUES
  ( #{id},
  #{username,jdbcType=VARCHAR},
  #{password.encrypted:VARCHAR},
  #{administrator,jdbcType=BOOLEAN}
  )
 </insert>

 <update id="update" parameterType="UserAlias">
  UPDATE user SET
  username = #{username,jdbcType=VARCHAR},
  password = #{password.encrypted,jdbcType=VARCHAR},
  administrator = #{administrator,jdbcType=BOOLEAN}
  WHERE
  id = #{id,jdbcType=INTEGER}
 </update>

<delete id="deleteBatch" parameterType="java.util.List">
delete from MESSAGE where ID in(<foreach collection="list" item="item" separator=",">
#{item},</foreach>)
</delete>

 <!--   Unique constraint check -->
 <select id="isUniqueUsername" parameterType="map" resultType="boolean">
  SELECT (count(*) = 0)
  FROM user
  WHERE ((#{userId,jdbcType=BIGINT} IS NOT NULL AND id != #{userId,jdbcType=BIGINT}) OR #{userId,jdbcType=BIGINT} IS
  NULL)  <!-- other than me -->
  AND (username = #{username,jdbcType=VARCHAR})
 </select>
</mapper>
```

主要就是这俩个XML进行配置，其中Configuration.xml是关键的文档，是我们主配置文件
而Message是我们的user.xml，用户.xml，我们具体配置的文档
Ps1：
<mapper namespace="">//namespace属性必须存在不可省略。
<select id="">
如果namespace名不同，则id可以一样，调用的时候，namespace名.id名；若namespace名相同，则id不能一样。
Ps2：
<mapper namespace="">
<resultMap id="">
//同上...
但是注意：resultMap——id可以与select——id一样（即使namespace相同），只要它们的id在各自领域（resultMap或select）范围内唯一即可。
Ps3：<resultMap>的子标签中，如果是主键栏，用<id>子标签，如果是普通栏，用<result>子标签。
Ps4：<select resultMap="resultMap中的id名">。
Ps5：<mappers>中子标签<mapper>的resource属性也是从src根目录开始算起，“.”改为“/”；如果有多个<mapper>可以继续写，个人理解相当于注册。

mybatis的sql语句通过xml文件进行配置
sql的配置文件中的<mapper>标签的namespace要唯一，调用sql语句，eg：sqlSession.selectList("Messages.list");//namespace的名字点上语句的ID

<resultMap type="" id="Message">//映射的是封装返回结果的bean，type是bean的全类名，id要唯一(resultMap中)
<id column="ID" jdbcType="VARCHAR" property="id"/> //主键使用，column对应的是数据的字段名，jdbcType对应的是数据字段的类型，property对应的是实体的属性名
<result /> //其他字段使用
</resultMap>

查询语句：
<select id="list" resultMap="Message">sql语句</select>

写好的sql配置文件，可在mybatis的连接配置文件中引入：
<mappers>
<mapper resource="sql配置文件 路径" />
</mappers>

OGNL可以直接调用Java类的方法；
OGNL中的特殊字符**需要转义**，如"转义为" &转义为&或者写为其特有操作符and。（遵循HTML转义规则）

MyBatis配置的时候，写SQL语句的时候，无需特意空格，**MyBatis自动会识别加空格。**
and COMMAND=?;(1,command)（注意：问号+问号赋值工程）等价于and COMMAND=#{command}。
关键代码：

```
<if test="command!=null and !&amp;quot;&amp;quot;.equals(command.trim())">
and COMMAND=#{command}
</if>
<if test="description!=null &amp;amp;amp; !&amp;quot;&amp;quot;.equals(description.trim())">
and DESCRIPTION like '%' #{description} '%'
</if>
```

OGNL是个功能强大的语言

Mybatis中的OGNL表达式		
取值范围	标签的属性中	
取值写法	String与基本数据类型	_parameter
	自定义类型(Message)	属性名(command)
	集合	数组：array
		List：list
		Map：_parameter
操作符	java常用操作符	+、-、*、/、==、!=、||、&&等
	自己特有的操作符	and、or、mod、in、not in

Mybatis中的OGNL表达式			
从集合中取出一条数据	数组	array[索引](String[ ])	
		array[索引].属性名(Message[ ])	
	List	list[索引](List<String>)	
		list[索引].属性名(List<Message>)	
	Map	_parameter.key(Map<String,String>)	
		key.属性名(Map<String,Message>)	
利用 foreach 标签 从集合中取出数据	<foreach collection="array" index="i" item="item">		
	数组	i：索引（下标）	item
	List		
	Map	i：key	item.属性名

log4j.properties说明：

```
log4j.rootLogger=DEBUG,Console
log4j.appender.Console=org.apache.log4j.ConsoleAppender
log4j.appender.Console.layout=org.apache.log4j.PatternLayout
log4j.appender.Console.layout.ConversionPattern=%d [%t] %-5p [%c] - %m%n
log4j.logger.org.apache=INFO
```

1.debug：输出的级别（模仿Android），Console输出端的名称（可以“自定义名称”，但是下面的appender.“自定义名称”一致）。
2.输出到控制台。
3.布局。
4.输出格式：%d产生时间，%t线程，%p日志级别（“-”号，空格在最右边（左对齐），去掉“-”号，空格在最左边（右对齐）；5：至少占位5个位置），%c日志打印的类，%m输出内容，%n换行。
5.org.apache:该包下的日志级别，第一行是针对所有的日志定义的级别。（相当于Android中Log中有个全部项目的日志信息，还有一个专门该项目的日志信息）

单独

Ps1：顺序（从高层到底层）：servlet负责接收页面的值和向页面传值。如果有业务逻辑需要处理则调用相应的service。service接收servlet传过来的值，并对其进行处理，做业务的操作，算法等等，如果有需要则调用相应的dao层。dao层完成与数据库的交互，执行相应的SQL语句。
Ps2：basePath后面默认有个“/”。

servlet负责接收页面的值和向页面传值。如果有业务逻辑需要处理则调用相应的service。service接收servlet传过来的值，并对其进行处理，做业务的操作，算法等等，如果有需要则调用相应的dao层。dao层完成与数据库的交互，执行相应的SQL语句。

Connection为何不需要commit();因为conn.setAutoCommit(true);设置为自动提交，而我们MyBatis把它封装后，setAutoCommit(false);所以需要手动提交事务。

Ps1：a href="#" 这个在html中有什么作用？跳转到本页面顶部，一般建议写成javascript:void(0);要好一点，点了一点反应都没有，写#点了会跳一下的。
Ps2：servlet拿到什么数据类型就什么类型，至于service不一致时，再service里面去修改，这也是service作用之一。
Ps3：MyEclipse对JS等文件报错处理：http://jingyan.baidu.com/article/ca41422fe094251eae99ede7.html
Ps4：

```
/**
* 调用后台批量删除方法
*/
function deleteBatch(basePath){
$("#mainForm").attr("action",basePath+"DeleteBatchServlet.action");
$("#mainForm").submit();
}
```

解析：将id为mainForm的表单的action提交路径改为basePath+"DeleteBatchServlet.action"这个并且执行submit提交表单!

s1：a href="#" 这个在html中有什么作用？跳转到本页面顶部，一般建议写成javascript:void(0);要好一点，点了一点反应都没有，写#点了会跳一下的。
Ps2：servlet拿到什么数据类型就什么类型，至于service不一致时，再service里面去修改，这也是service作用之一。
Ps3：MyEclipse对JS等文件报错处理：http://jingyan.baidu.com/article/ca41422fe094251eae99ede7.html
Ps4：

```
/**
* 调用后台批量删除方法
*/
function deleteBatch(basePath){
$("#mainForm").attr("action",basePath+"DeleteBatchServlet.action");
$("#mainForm").submit();
}
```

解析：将id为mainForm的表单的action提交路径改为basePath+"DeleteBatchServlet.action"这个并且执行submit提交表单!

配置一对多

cloumn是指<select>标签中sql查询的结果集的列名，有别名的就是别名为列名
多表连接查询时，通过别名来区别是哪个表中的键

只需要在1表中添加一个集合，包含于2表的信息 mybatis内部不关心如何实现，添加就好
在实体类中添加一个集合，例如：list
在配置文件中配置：
<collection property = "commandContentList" resultMap = "commandContent.content"/ >配置一对多

快捷键：
1、Ctrl+Shift+R：查看该高亮部分是哪个类。
2、Alt+Shift+R：选择高亮部分修改关联的名称（不要自己手动去改，否则所有都要自己手动去改）。

关联(映射)到子表<association ></association>

mybatis常用标签

功能	标签名称
定义SQL语句	insert
	delete
	update
	select
配置java对象属性与查询结果集中列名对应关系	resultMap
控制动态SQL拼接	foreach
	if
	choose
格式化输出	where
	set
	trim
配置关联关系	collection
	association
定义常量	sql
引用常量	include

http://www.bug315.com/article/359.htm**foreach详解**

resultMap	resultType
parameterMap	parameterType
#{}	${}
#{}	ognl

resultType是java类型，map需要配置
resultType与resultMap存在数据库使用时存在问题，resultType必须要写清楚的类型，有时候可能会导致出错，而resultMap则很少出现这种情况
里面有一个typeHandler属性 作类型转换，因为数据库中的类型与JAVA是不同的，有时会因为这个类型，值与我们预想的有些差异，会出错

paramterMap表明参数中的属性与数据库的列对应的关系（不推荐使用的）

#{}与${}符号的区别

```
select ID,COMMAND from MESSAGE where COMMAND=#{command}

select ID,COMMAND from MESSAGE where COMMAND=${command}
```

使用时被mybatis使用 $符号没有预编译效果的 但是这样是错的，因为没有预编译，执行不起来

```
select ID,COMMAND from MESSAGE where COMMAND= ?

select ID,COMMAND from MESSAGE where COMMAND= 段子
```

这样写才能有用

一般不会用$符号

```
select ID,COMMAND from MESSAGE where COMMAND='${command}'
```

parameterType	取值写法	
String或基本数据类型	#{_parameter}	√
	#{随便写点啥}	√
	ognl:_parameter	√
	ognl:随便写点啥	×
自定义类型(Message)	#{随便写点啥}	×

代码要有整体的风格，要被人好看懂，一定要这样

一、获取自增主键值：

这是在**mysql**数据库用法，设置主键自增的时候 **primary key auto_increment**

```
<insert id="insert" useGeneratedKeys="true" keyProperty="id" parameterType="com.imooc.bean.Command">

insert into command(name,description) values(#{name},#{description})

</insert>
```

//解析：添加数据（在主外键的关系中）可以使用 useGeneratedKeys="true"，可获取自增长的id，并配合keyProperty="id"（java中实体类的属性名）指定该对象的主键值。

二、找不到namespace.id的异常效果：

1、在Configuration.xml没有配置<mappers><mapper resource="*.xml"></mapper></mappers>；

2、在1的配置中或selectList("Message.queryMessageList",message);中，名字写错。

三、排查SQL语法错误：如果控制台出现SQL语句问题，复制到SQL软件上执行，若有参数，手动写上并执行。

四、不要过度使用${}。

这个就像java代码拼接字符串一样，感觉非常的方便，但是，这样导致里面的#{}这些东西就不会解析了，让sql语句回归到java代码的方式呢，叫注解sql，但是注解sql像动态sql这样实现动态是很麻烦的，实际项目中运用就感觉到了

五、乱码问题：

1. servlet传参时的编码：request.setCharacterEncoding("utf-8");或直接使用过滤器；

2. Java文件本身的编码；

3. 连接数据库的参数中，设定编码方式：jdbc:mysql://192.168.1.1:3306/cms?characterEncoding=utf-8

4. 数据库、表的编码；

5. 展示页面的编码：<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

6. 浏览器编码问题等。