第一天:mybatis(上)

备课内容

1. mybatis简介

2. mybatis框架执行原理

3. mybatis初次的入门案例

4. mybatis配置文件详解

5. mybatis 方法多参数的处理

6. mybatis返回主键值

7. sql代码段

8. 自定义结果类型ResultMap

---

mybatis简介

mybatis本身是一个轻量级的持久化层框架(1.这里讲解一下何为持久化|2.也可引出序列化操作),本身也是基于JDBC的封装(回顾一下JDBC的链接步骤).开发者本身更多的关注SQL语句的执行效率,除此之外mybatis也是一个半自动的ORM映射框架(支持一对一,一对多的实现,多对多采用双一对多实现)

注意:实际的开发过程中,因为大量的关系相互映射的存在,在查询数据这一块不便于后期项目本身的项目维护扩展。所以更多的方向是思考数据库中的设计和利用本身mybatis提供的数据自定义封装和其它的类似缓存机制的特点

> mybatis的优势

- 比起jdbc的操作,减少了一些代码量，也方便能够继承到后期所讲的spring中

- mybatis现在提供在XML中(有结构的标准中编写sql语句),不直接入侵在代码中（也就是经常说的降低耦合度和代码的复用）

- 分别提供的xml标签和映射标签(xml标签可实现动态SQL语句,也就是嵌入条件判断和循环,比较类似存储过程),映射标签支持正向的ORM映射action

mybatis框架执行原理

> sqlConfigXMl配置文件(一个全局的配置文件)(可配置映射文件和连接数据源和事务等)

> 通过配置文件构建出可构建操作数据会话的会话工厂，也就是我们常说的sqlSessionFactory(可以说下利用反射的工厂模式)

> 通过sqlSessionFactory生产出相互独立的sqlsession，为什么是独立的会话，既然是独立的会话，那也有全局的会话(简单提及缓存)进行数据库层面上面的操作

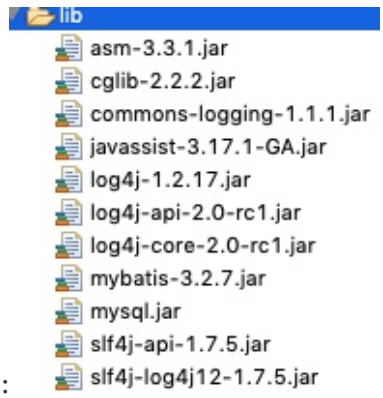
> sqlsession之所以能够操作，依赖一个叫Executor的执行器，通过该执行器进行数据库的CRUD操作

> Executor的执行器需要操作CRUD的动作由谁而来，就是由mapper statement(映射语句集读取我们的mapper映射文件)

我们可以也就可以看到在配置xml文件的时候，可以支持多种对象级数据参数

mybatis初次的入门案例 (开发工具:eclipse , mysql-version:8.0.12)

1. 构建一个普通的web项目,结构如下:



大致介绍一下**jar**包的结构

2. src目录下构建mybatis的全局配置文件mybatisCfg.xml,配置文件如下:

```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>
    <!-- 数据库连接环境配置 -->
    <environments default="development">
        <!-- 标明mybaitis环境 id唯一 -->
        <environment id="development">
            <!-- JDBC – 这个配置直接简单使用了 JDBC 的提交和回滚设置。 它依赖于从数据源得 到的连接来管理事务范围。JDBC默认是自动提交 -->
            <transactionManager type="JDBC" />
            <!-- 采用数据库连接池 -->
            <dataSource type="POOLED">
                <property name="driver" value="com.mysql.jdbc.Driver" />
                <!-- 避免环境的不统一，造成数据操作乱码 -->
                <property name="url"
                    value="jdbc:mysql://localhost:3306/mybatis?characterEncoding=utf-8" />
                <property name="username" value="root" />
                <property name="password" value="123456" />
            </dataSource>
        </environment>
    </environments>
    <!-- 添加需要被映射的文件 -->
    <mappers>
        <mapper resource="com/wwj/dao/PersonMapper.xml" />
    </mappers>
</configuration>
```

3. 构建模型类 com.wwj.model 和 数据库表

```java
import java.io.Serializable;
import java.util.Date;
/**
 * 基本的模型类
 * @author wwj
 *对象序列化是一个用于将对象状态转换为字节流的过程，可以将其保存到磁盘文件中或通过网络发送到任何其他程序；从字节流创建对象的相反的过程称为反序列化。而创建的字节流是与平台无关的，在一个平台上序列化的对象可以在不同的平台上反序列化。
 *无需序列化的变量使用transient
 */
public class Person implements Serializable {
    //Java的序列化机制是通过在运行时判断类的serialVersionUID来验证版本一致性的
    //这里是用来表明版本的一致性
    private static final long serialVersionUID  = 1L;

    private  Integer   id;
    private  String name;
    private  Date bir;
    private  String address;
    //自行get和set
    }
```

4. 创建数据层的操作也就是mapper操作接口

```
/**
 * person层的操作
 *
 * @author Yun
 *
 */
public interface PersonDao {
    /**
     * 新增用户
     *
     * @param p
     *            传入需要新增的对象
     * @return 0,1代表结果
     */
    int savePerson(Person p);

    /**
     * 更新用户对象
     * @param p 需要被更新的对象
     * @return 0,1代表结果
     */
    int updatePerson(Person p);

    /**
     * 根据用户id进行删除
     * @param id  唯一用户id
     * @return  0, 1代表结果
     */
    int  deletePersonById(int  id);

    /**
     * 获取所有的信息
     * @return  所有的人员信息
     */
    List<Person>  getPersonInfos();
}
```

5. 构建对应的mapper映射文件

```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="com.wwj.dao.PersonDao">
    <insert id="savePerson" parameterType="com.wwj.model.Person">
        insert into
        person(name,address,bir) values(#{name},#{address},#{bir});
    </insert>

    <update id="updatePerson" parameterType="com.wwj.model.Person">
        update person set
        name=#{p.name},address=#{address},bir=#{bir}
        where id=#{id}
    </update>

    <delete id="deletePersonById" parameterType="int">
        delete from person
        where id=#{id}
    </delete>


    <select id="getPersonInfos" resultType="com.wwj.model.Person">
        select * from person
    </select>

</mapper>
```

6. 编码测试

```java
/**
 * 测试mybatis的CRUD操作
 *
 * @author wwj
 *
 */
public class TestMybatis {
    public static void main(String[] args) throws IOException, ParseException {
        /*
         * 日期上面的处理
         */
        SimpleDateFormat sf = new SimpleDateFormat("yyyy-MM-dd");
        String format = sf.format(new Date());
        Date parse = sf.parse(format);

        InputStream is = Resources.getResourceAsStream("mybatisCfg.xml");
        SqlSessionFactory build = new SqlSessionFactoryBuilder().build(is);
        // 生成 session
        SqlSession session = build.openSession();
```

```
        Person per = new Person();
        per.setName(" 小王 ");
        per.setAddress(" 重庆 ");
        per.setBir(parse);
        // 操作数据
        int insert = session.insert("savePerson",per);
        // 提交事务
        session.commit();
        // 关闭 session
        session.close();

    }
}
```

注意下时间的处理

结果图示

id	name	bir	address
1	小王	2019-07-16	重庆

同理依次测试更新和删除,以及查询

更新

```
        Person per = new Person();
        per.setId(1);
        per.setName(" 小张 ");
        per.setAddress(" 重庆 ");
        per.setBir(parse);
        // 操作数据
        int update = session.update("updatePerson", per);
```

id	name	bir	address
1	小张	2019-07-16	重庆

查询

```
        List<Person> pers  = session.selectList("getPersonInfos");
```

删除

```
    int de  = session.delete("deletePersonById", 1);
```

mybatis配置文件详解

全局配置文件详解

- environments 环境配置，可以配置多种环境 default 指定使用某种环境
- transactionManager 事务管理器 有两种取值 JDBC ,managed.我们选择jdbc即可
- dataSource 配置数据源，采用默认的连接池选择项POOLED
- mappers标签里面填入需要映射的数据操作xml文件
- 映射的数据操作文件需要和接口保持同个路径(可以把mapper当成接口的实现类)

映射文件详解

- namespace 表明需要对应映射的空间即是接口所在的全路径名称
- id 与 接口中的方法保持一致
- parameterType 填写自定义对象的全路径名称
- 接收参数采用 #{objAttrName}

mybatis 方法多参数的处理 （代码示例）

1.索引接收 （了解即可）

```
    List<Person>  getPersonInfosByNameAndID(String name ,int id);

    <select id="getPersonInfosByNameAndID" >
        select * from person where name = #{0} and id = #{1}
    </select>
```

2.map接收 （重点）

```
    /**
     *  根据map进行查询
     * @param attrs
     * key1 id  key2 name
     * @return
     */
    List<Person>  getPersonInfosByMap(Map attrs);
    //------------------
```

```
  <select id="getPersonInfosByMap"  parameterType="java.util.Map"
resultType="com.wwj.model.Person">
        select * from person where id =
        #{id} and name = #{name}
    </select>
      //----------------
      Map<String,Object>  attrs = new HashMap<>();
        attrs.put("id", 2);
        attrs.put("name", "小王");
        session.selectList("getPersonInfosByMap", attrs);
```

3.注解@Param接收 （重点）

```
    /**
     * 根据用户唯一id查询信息
     * @param id
     * @return
     */
    Person  getPersonInfo(@Param("pid") int id);
    //---------------
    <select id="getPersonInfo"
        resultType="com.wwj.model.Person">
        select * from person where id =
        #{pid}
    </select>
    //---------
    session.selectOne("getPersonInfo", 2);
```

6.mybatis返回主键值

应用场景:当我们需要在一个事务插入数据后获取数据的主键id，做下一步额外操作，并且而不为并发高的情况下取错值而考虑

修改代码如下

```
  <insert id="savePerson" parameterType="com.wwj.model.Person">
        <selectKey keyProperty="id" resultType="int" order="AFTER">
            select last_insert_id()
        </selectKey>
        insert into
        person(name,address,bir) values(#{name},#{address},#{bir});
    </insert>
    //-------------------------------------------------------------
        // 操作数据
        Person p = new Person();
        p.setName("小小王");
        p.setBir(parse);
        int result = session.insert("savePerson", p);
```

```
14          System.out.println(result);
15          System.out.println(p.getId());
```

- keyProperty="返回主键的id的属性名”
- resultType="主键类型"
- order=""什么时候执行，在SQL执行前还是执行后执行，两个取值：BEFORE和AFTER
- select last_insert_id()取到最后生成的主键，只在当前事务中取

7. sql代码段

** 如果场景中有大量的重复的公共sql 那么可以考虑声明公共的部分 **

** 示例如下 **

```
1     /**
2      * sql片段
3      * @param id  根据用户的id查询姓名
4      * @return
5      */
6     String  getPersonName(@Param("pid") int id);
7
8   <sql id="nameCol"> name</sql>
9     <select id="getPersonName"   resultType="java.lang.String">
10        select
11        <include refid="nameCol"></include>
12        from person where
13        id =
14        #{pid}
15    </select>
16
17        // 操作数据
18        String name  = session.selectOne("getPersonName",2);
19        System.out.println(name);
```

自定义结果类型ResultMap(开发中长期使用)

** 应用场景:假设我们的实际开发过程中,数据表组合字段多，又不想关心配置映射关系，只想关心sql语句，并且关心sql语句的效率 **

1. 假设2张表 person和card 1:m关系
2. 连接查询需要person中的人名和card表中的卡号名字
3. 不想做映射关系的配置

> 操作步骤如下:

1.在任意自定对象上添加属性

```
public class Card {

    private String cname;
    }
//-----
public class Person implements Serializable {


    private  Integer   id;
    private  String name;
    private  Date bir;
    private  String address;

    private  List<Card>  cards;
    }
//----构建自定义的resultmap封装 注意   collection 代表多   association 代表一
    <resultMap type="com.wwj.model.Person" id="personRS">
        <!--column指向数据库列名 property指向pojo对象中字段名 -->
        <result column="name" property="name" />
        <!-- property指的是在bean中字段名 ofType类的全定向名 -->
        <collection property="cards" ofType="com.wwj.model.Card">
            <result column="cname" property="cname" />
        </collection>
    </resultMap>
//--------代码操作
        List<Person> persons = session.selectList("getPersonsOfCard");
        for (Person person : persons) {
            System.out.println(person.getCards().get(0).getCname());
            System.out.println(person.getCards().get(1).getCname());
        }
```

*** 注意1:开发中可以不需要添加多的vo进行操作,只需关注定义的类需要添加的属性和自定义的resultMap ***

*** 注意2:如果需要暴露一部分数据出去的,可能还是会单独做接口和设计VO ***