

# Day29

java课程-李彦伯

Day29

Mybatis

介绍

Mybatis的HelloWorld

HelloWorld讲解

将mybatis应用到dao层

mapper动态代理开发

## Mybatis

### 介绍

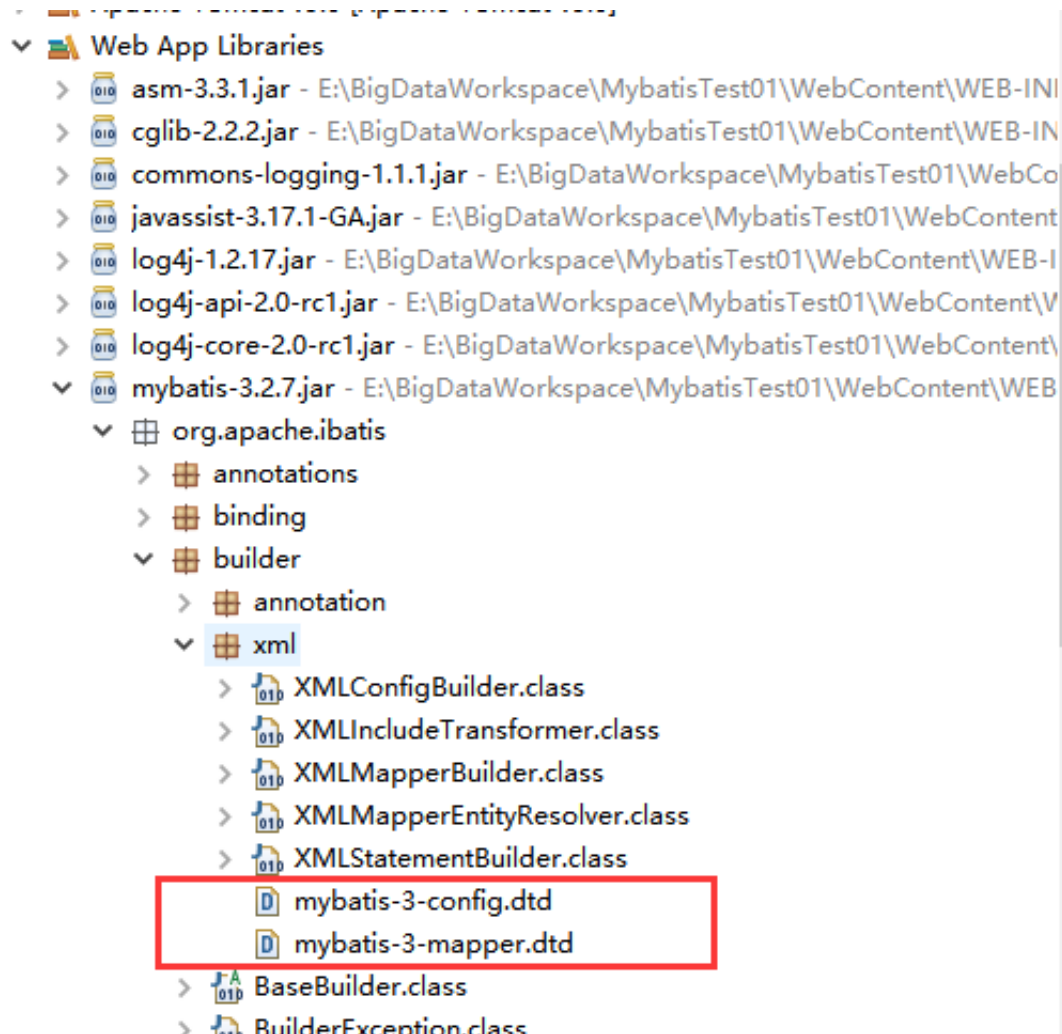
MyBatis 本是apache的一个开源项目iBatis, 2010年这个项目由apache software foundation 迁移到了google code , 并且改名为MyBatis 。2013年11月迁移到Github。







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。

- 对比JDBC开发,传统的JDBC存在如下问题
  - 数据库连接创建、释放频繁造成系统资源浪费，从而影响系统性能。如果使用数据库连接池可解决此问题。
  - Sql语句在代码中硬编码，造成代码不易维护，实际应用中sql变化的可能较大，sql变动需要改变java代码。
  - 使用preparedStatement向占有位符号传参数存在硬编码，因为sql语句的where条件不一定，可能多也可能少，修改sql还要修改代码，系统不易维护。
  - 对结果集解析存在硬编码（查询列名），sql变化导致解析代码变化，系统不易维护，如果能将数据库记录封装成pojo对象解析比较方便

## Mybatis的HelloWorld

- 下载Mybatis的jar包<https://github.com/mybatis/mybatis-3/releases>
- 添加jar包,mybatis核心包+mybatis依赖包+数据库驱动包
- 添加xml文件约束dtd,在核心包 `/org/apache/ibatis/builder/xml/mybatis-3-config.dtd`和 `/org/apache/ibatis/builder/xml/mybatis-3-mapper.dtd`对应的两个key值分别是 `-//mybatis.org//DTD Config 3.0//EN`和 `-//mybatis.org//DTD Mapper 3.0//EN` RootElements分别是 `configuration`和 `mapper`



- >  `CacheRefResolver.class`
- >  `IncompleteElementException.class`
- >  `MapperBuilderAssistant.class`
- >  `ParameterExpression.class`
- >  `ResultMapResolver.class`
- >  `SqlSourceBuilder.class`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN" "mybatis-3-config.dtd">
<configuration>
    <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://bigdata14" />
                <property name="username" value="root" />
                <property name="password" value="root" />
            </dataSource>
        </environment>
    </environments>
    <mappers>
        <mapper resource="sqlmap/User.xml" />
    </mappers>
</configuration>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "
mybatis-3-mapper.dtd">
<mapper>
</mapper>
```

- 配置日志文件,创建log4j.properties,将以下的内容赋值进去

```
# Global logging configuration
log4j.rootLogger=DEBUG, stdout
# Console output...
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%5p [%t] - %m%n
```

- 配置mapper文件

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"mybatis-3-mapper.dtd">
<!--
```

必须要配置namespace属性,使用namespace可以区分不同的mapper中有相同的sql的名称

```
-->
<mapper namespace="user">
<insert id="addUser" parameterType="com.zhiyou100.mybatis.model.User">
insert into user values (null,#{username},#{birthday},#{sex},#{address})
</insert>
<select id="findUserById" parameterType="Integer" resultType="com.zhiyou100.mybatis.model.User">
    select * from user where id = #{v}
</select>

<update id="updateUser" parameterType="com.zhiyou100.mybatis.model.User">

    update user set username=#{username},birthday=#{birthday},
        sex=#{sex},address=#{address} where id = #{id}
```

```

</update>

<delete id="deleteUser" parameterType="Integer">
delete from user where id = #{v}
</delete>

<select id="findAllUser" resultType="com.zhiyou100.mybatis.model.User">
    select * from user
</select>

<select id="findUserByName" resultType="com.zhiyou100.mybatis.model.User" parameterType="String">
    select * from user where username like '%' #{v} '%'
</select>
<!--
    #{v} 表示sql中的 ? 其中的内容可以随便写
    相当于在执行的时候会自动会加 ' ' 推荐这种方式
    ${value} 表示字符串拼接,不会加任何符号,需要我们自己拼接
-->
</mapper>

```

- 写一个测试类
  1. 创建sqlSessionFactory
  2. 通过文件创建sqlSession



3. 通过openSession建立连接
4. 通过sqlSession的增删改查

```
public class TestDemo {
    @Test
    public void test01() throws IOException{

        /*
         * 1.加载配置文件
         * 2.创建sqlsession工程
         * 3.创建sqlSession操作数据库
         */
        InputStream in = Resources.getResourceAsStream("sqlMapConfig.xml");

        SqlSessionFactory sf = new SqlSessionFactoryBuilder().build(in);

        SqlSession session = sf.openSession();

        /*      User u = session.selectOne("u.findUserById",1);
        System.out.println(u);*/
        /*      User n1 = new User();
        n1.setAddress("xxxxx");
        n1.setUsername("张无忌");
        n1.setBirthday(new Date(System.currentTimeMillis()));
        n1.setSex("男");
        session.insert("user.insertUser",
n1);*/
```

```

        //session.delete("user.deleteUser", 27);

        User n1 = new User();
        n1.setId(29);
        n1.setAddress("xxxxx");
        n1.setUsername("张三丰");
        n1.setBirthday(new Date(System.currentTimeMillis()));
        n1.setSex("女");
        session.update("user.updateUser", n1);

        List<User> list = session.selectList("user.findAllUser");
        System.out.println(list);
        List<User> list2 = session.selectList("user.findUserByName", "杨");
        System.out.println(list2);
        session.commit();
        session.close();
    }
}

```

## HelloWorld讲解

- namespace,必须要进行设置,如果不设置,文件无法加载可以表示当前mapper中的增删改查数据哪个命名空间,用以区别其他的sql中的内容
- `#{任意写}` 和 `${value}` 的区别, `#{任意写}` 用来表示sql

中的占位符？使用 `${value}` 用来表示字符串的拼接

- 核心配置文件中的 `<mappers>` 标签用来在核心配置文件中引入对应的对应的mapper文件
- mapper配置文件是用来书写sql语句的,一个配置文件中只能有一个mapper标签,mapper标签内部有增删改查标签,用来书写增删改查操作
- 增删改查标签中的parameterType属性为设置执行sql语句的时候传过来的值的类型
- 增删改查标签中的resultType属性为设置执行sql语句的时候返回值的类型
- 如果返回值是list,返回值的类型同样还是写成对应集合中存取对象的类型

## 将mybatis应用到dao层

```
public class UserDaoImp implements UserDao {
    //通过spring容器注入进来
    private SqlSessionFactory factory;

    public void setFactory(SqlSessionFactory factory) {
        this.factory = factory;
    }
    @Override
    public void addUser(User u) {
        factory.openSession().insert("user.addUser", u);
    }
    @Override
    public void deleteUser(int id) {

        factory.openSession().delete("user.deleteUser", id);
    }

    @Override
    public void updateUser(User u) {
        factory.openSession().update("user.updateUser", u);
    }

    @Override
    public List<User> findAllUser() {
```

```
        List<User> list = factory.openSession().selectList("user.findAllUser");  
        return list;  
    }  
}
```

## mapper动态代理开发

- 创建接口写接口方法确保4个一致
  - 方法名称和文件中的id一致
  - 方法的返回值和对应标签内的resultType一致
  - 方法的参数和parameterType一致
  - namespace和接口的全名称一致

```
public class TestMapperDemo {  
    private SqlSessionFactory factory;  
  
    @Before  
    public void before() throws IOException{  
        factory = new SqlSessionFactoryBuilder().build(Resources.getResourceAsStream("sqlMapConfig.xml"));  
    }  
  
    @Test  
    public void test01(){  
        SqlSession session = factory.openSession();  
        //mybatis会动态的创建接口的实现类,调用接口相应的方法就会执行相关的数据库操作  
        UserMapper user = session.getMapper(UserMapper.class); System.out.println(user.findAllUser());  
    }  
}
```