MyBatis

主流的 ORM 框架，之前叫做 iBatis，后来更名为 MyBatis，实现数据持久化的框架。

同时 Java，.NET，Ruby 三种语言，MyBatis 是一个对 JDBC 进行封装的框架。

ORM 框架 Hibernate，MyBatis 和 Hibernate 的区别?

Hibernate 是一个“全自动化” ORM 框架，MyBatis 是一个“半自动化的”ORM框架。

全自动化：开发者只需要调用相关接口就可以完成操作，整个流程框架都已经进行了封装。

Hibernate 实现了 POJO 和数据库表之间的映射，同时可以自动生成 SQL 语句并完成执行。

半自动化：框架只提供一部分功能，剩下的工作仍需要开发者手动完成，MyBatis 没有提供 POJO 与数据库表的映射，只实现了 POJO 与 SQL 之间的映射关系，需要开发者自定义 SQL 语句，以及数据与 POJO 之间的装配关系。

虽然功能没有 Hinbernate 更加方便，但是这种“半自动化”的方式**提高了框架的灵活性**，开发者可以根据具体的业务需求，完成定制化的持久层解决方案。

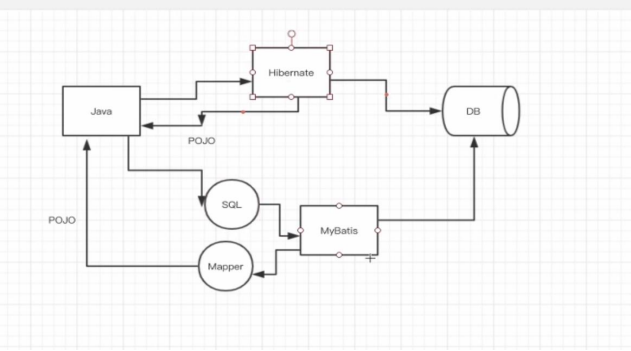
MyBatis 对所有的 JDBC 进行了封装，包括参数设置、SQL 执行、结果集解析等，通过 XML 配置/注解的方式完成 POJO 与数据的映射。

简单讲，使用 MyBatis 进行开发，主要完成两步操作：

- 自己编写 SQL
- 自己完成数据库数据与 POJO 的映射

MyBatis

- 极大简化了 JDBC 代码的开发
- 简单好用、容易上手、具有更好的灵活性
- 通过将 SQL 定义在 XML 中的方式降低程序的耦合性
- 支持动态 SQL，可以根据具体业务需求灵活实现功能



MyBatis

- 相比于 Hibernate，开发者需要完成更多工作，比如定义 SQL、设置 POJO 与数据的映射关系等。
- 要求开发人员具备一定的 SQL 编写能力，在一些特定场景下工作量比较大。
- 数据库移植性差，以为 SQL 依赖于底层数据库，如果要进行数据库迁移，部分 SQL 需要重写编写。

MyBatis 入门

1、创建 Maven 工程，pom.xml 引入相关依赖。

```
<dependencies>
    <!-- MyBatis -->
    <dependency>
```

```
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis</artifactId>
        <version>3.4.5</version>
    </dependency>

    <!-- MySQL驱动 -->
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>8.0.19</version>
    </dependency>
</dependencies>
```

2、创建实体类

```
package com.southwind.entity;

import lombok.Data;

@Data
public class People {
    private Integer id;
    private String name;
    private Double money;
}
```

实体类要和表中的字段对应起来

实体对象用包装类 以mybatis为例,如果是包装类型,只需要判断是否为null,便可以决定是否更新这个字段

3、配置 MyBatis 环境，在 resources 路径下创建 config.xml（文件名可以自定义）,配置数据源信息。

```
<?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>
    <!-- 配置 MyBatis 运行环境 -->
    <environments default="development">
        <environment id="development">
            <transactionManager type="JDBC"></transactionManager>
            <!-- 数据源 -->
            <dataSource type="POOLED">
                <property name="driver" value="com.mysql.cj.jdbc.Driver"/>
                <property name="url" value="jdbc:mysql://localhost:3306/test?
useUnicode=true&amp;characterEncoding=UTF-8"/>
                <property name="username" value="root"/>
                <property name="password" value="root"/>
            </dataSource>
        </environment>
    </environments>
</configuration>
```

可以配置多个environment标签比如dev与pro环境

数据库连接池

4、MyBatis 开发有两种方式

- 使用原生接口
- Mapper 代理实现自定义接口

使用原生接口

1、创建 Mapper 文件 PeopleMapper.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.southwind.mapper.PeopleMapper">

    <select id="findById" parameterType="java.lang.Integer"
resultType="com.southwind.entity.People">
        select * from people where id = #{id}
    </select>

</mapper>
```

命名空间表示文件所在位置

sql语句查处结果，resultType这里就完成了结果与POJO的映射

namespace 通常设置为文件所在包名 + 文件名，parameterType 是参数数据类型，resultType 是返回值数据类型。

2、在全局配置文件 config.xml 中注册 PeopleMapper.xml。

必须注册才能生效

```
<mappers>
    <mapper resource="com/southwind/mapper/PeopleMapper.xml"></mapper>
</mappers>
```

3、调用 API 完成操作。

```
package com.southwind.test;

import com.southwind.entity.People;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;

import java.io.InputStream;

public class Test {
    public static void main(String[] args) {
        //加载MyBatis配置文件
        InputStream inputStream =
Test.class.getClassLoader().getResourceAsStream("config.xml");
        SqlSessionFactoryBuilder sqlSessionFactoryBuilder = new
SqlSessionFactoryBuilder();
```

```
        SqlSessionFactory sqlSessionFactory =
sqlSessionFactoryBuilder.build(inputStream);
        //获取 SqlSession
        SqlSession sqlSession = sqlSessionFactory.openSession();
        //调用 MyBatis 原生接口执行 SQL 语句
        String statement = "com.southwind.mapper.PeopleMapper.findById";
        People people = sqlSession.selectOne(statement,1);
        System.out.println(people);
        sqlSession.close();
    }
}
```

namespace+id

4、IDEA 中无法直接读取 resources 路径外的 XML 文件，需要进行设置，pom.xml

idea里面的maven默认只能在resource里面读xml文件，现在的文件在java目录下

```
<build>

    <resources>
        <resource>
            <directory>src/main/java</directory>
            <includes>
                <include>**/*.xml</include>
            </includes>
        </resource>
    </resources>

</build>
```

Mapper 代理实现自定义接口

开发中一般使用这种

开发者只需要定义接口，并不需要实现接口，具体的实现工作由 Mapper 代理结合配置文件完成。

1、自定义接口

只要定义不要实现

```
package com.southwind.repository;

import com.southwind.entity.People;

import java.util.List;

public interface PeopleRepository {
    public int save(People people);
    public int deleteById(Integer id);
    public int update(People people);
    public People findById(Integer id);
    public List<People> findAll();
}
```

2、创建 PeopleMapper.xml，定义接口方法对应的 SQL 语句，statement 标签根据 SQL 执行的业务可以选择 select、insert、delete、update，MyBatis 会自动**根据规则**创建 PeopeleRepository 接口的实现类代理对象。

规则如下：

- PeopleMapper.xml 中的 namespace 为接口的全限定类名（带着包名的类名）
- PeopleMapper.xml 中的 statement 的 id 为接口中对应的方法名
- PeopleMapper.xml 中的 parameterType 和接口中对应方法的参数类型一致
- PeopleMapper.xml 中的 resultType 和接口中对应方法的返回值类型一致

```
<?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.southwind.repository.PeopleRepository">
    <insert id="save" parameterType="com.southwind.entity.People">
        insert into people(name,money) values(#{name},#{money})
    </insert>

    <delete id="deleteById" parameterType="java.lang.Integer">
        delete from people where id = #{id}
    </delete>

    <update id="update" parameterType="com.southwind.entity.People">
        update people set name = #{name},money = #{money} where id = #{id}
    </update>

    <select id="findById" parameterType="java.lang.Integer"
resultType="com.southwind.entity.People">
        select * from people where id = #{id}
    </select>

    <select id="findAll" resultType="com.southwind.entity.People">
        select * from people
    </select>
</mapper>
```

返回值类型确定的情况下不需要指定resultType

3、完成注册

```
<mappers>
    <mapper resource="com/southwind/repository/PeopleRepository.xml"></mapper>
</mappers>
```

4、调用API

```
package com.southwind.test;

import com.southwind.entity.People;
import com.southwind.repository.PeopleRepository;
```

```
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;

import java.io.InputStream;
import java.util.List;

public class Test2 {
    public static void main(String[] args) {
        InputStream inputStream =
Test2.class.getClassLoader().getResourceAsStream("config.xml");
        SqlSessionFactoryBuilder sqlSessionFactoryBuilder = new
SqlSessionFactoryBuilder();
        SqlSessionFactory sqlSessionFactory =
sqlSessionFactoryBuilder.build(inputStream);
        SqlSession sqlSession = sqlSessionFactory.openSession();
        //获取实现了自定义接口的代理对象
        PeopleRepository peopleRepository =
sqlSession.getMapper(PeopleRepository.class);
//        People people = new People();
//        people.setName("小明");
//        people.setMoney(Double.parseDouble("666"));
//        int row = peopleRepository.save(people);
//        People people = peopleRepository.findById(6);
//        System.out.println(people);
//        people.setName("小红");
//        people.setMoney(Double.parseDouble("800"));
//        peopleRepository.update(people);
//        System.out.println(row);
//        peopleRepository.deleteById(6);
//        sqlSession.commit();
        List<People> list = peopleRepository.findAll();
        for(People people:list){
            System.out.println(people);
        }
        sqlSession.close();
    }
}
```

Mapper.xml 常用配置

MyBatis 配置文件有两种：

- 全局环境配置文件（数据源、事务管理、Mapper 注册、打印 SQL、惰性加载、二级缓存。。。）
- Mapper 配置文件（定义自定义接口的具体实现方案：SQL、数据与 POJO 的映射）

多表关联查询包括一对一、一对多、多对多

单表查询

```
<select id="findById" parameterType="java.lang.Integer"
resultType="com.southwind.entity.People">
    select * from people where id = #{id}
</select>
```

业务：通过 id 查询 People 对象

目标表：test/people　test库中的people表

实体类：com.southwind.entity.People

Mapper.xml 设置相关配置逻辑，由 MyBatis 自动完成查询，生成 POJO。

statement 标签主要属性有 id、parameterType、resultType

id 对应接口的方法名，parameterType 定义参数的数据类型、resultType 定义查询结果的数据类型（实体类的成员变量列表必须与目标表的字段列表一致）　重要前提只有这样resultType才能映射

paramterType

支持基本数据类型、包装类、String、多参数、POJO 等。

1、基本数据类型，通过 id 查询 POJO。

```
public People findById(int id);
```

```
<select id="findById" parameterType="int"
resultType="com.southwind.entity.People">
    select * from people where id = #{num}
</select>
```

2、包装类

```
public People findById(Integer id);
```

Integer

```
<select id="findById" parameterType="int"
resultType="com.southwind.entity.People">
    select * from people where id = #{num}
</select>
```

3、String 类型

```
public People findByName(String name);
```

```
<select id="findByName" parameterType="java.lang.String"
resultType="com.southwind.entity.People">
  select * from people where name = #{name}
</select>
```

4、多参数

```
public People findByIdAndName(Integer id,String name);
```

```
<select id="findByIdAndName" resultType="com.southwind.entity.People">
    select * from people where id = #{id} and name = #{name}
</select>
```

5、POJO

```
public int update(People people);
```

```
<update id="update" parameterType="com.southwind.entity.People">
    update people set name = #{name},money = #{money} where id = #{id}
</update>
```