ORMapping： Object Relationship Mapping

对象指面向对象

关系指关系型数据库

Java到Mysql的映射，开发者可以以面向对象的思想来管理数据库。

如何使用

- 新建Maven工程， pom.xml文件中添加相关依赖

```
<dependencies>
    <dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis</artifactId>
        <version>3.4.5</version>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>8.0.11</version>
    </dependency>
</dependencies>
```

- 新建数据表

```
use fxc;
create table t_account{
    id int primary key auto_increment,
    username varchar(11),
    password varchar(11),
    age int
}
```

- 新建数据表对应的实体类Account

```
package com.fxc.entity;

import lombok.Data;

@Data
public class Account {
    private int id;
    private String username;
    private String password;
    private int age;
}
```

- 创建 MyBatis 的配置文件config.xml，文件名可自定义

```
<?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="mysql">
        <environment id="mysql">
            <!-- 配置JDBC事务管理 -->
            <transactionManager type="JDBC"></transactionManager>
            <!-- POOLED配置JDBC数据源连接池 -->
            <dataSource type="POOLED">
                <property name="driver" value="com.mysql.jdbc.Driver"/>
                <property name="url" value="jdbc:mysql://localhost:3306/fxc?
useUnicode=true&amp;characterEncoding=UTD-8"/>
                <property name="username" value="root"/>
                <property name="password" value="333"/>
            </dataSource>
        </environment>
    </environments>
</configuration>
```

> 使用原生接口

1、MyBatis 框架需要开发者自定义SQL语句，写在Mapper.xml 文件中，实际开发中，会为每个实体类创建对应的Mapper.xml， 定义管理该对象数据的 SQL。

```
    ```xml
```

insert into t_account(username, password, age) values(#{username}, #{password}, #{age}) ```

- namespace 通常设置为文件所在包+文件名的形式
- insert 标签表示执行添加操作
- select 标签表示执行查询操作
- update 标签表示执行更新操作
- delete 标签表示执行删除操作
- id 是实际调用Mybatis方法时需要用到的参数
- parameterType 是调用对应方法时参数的数据类型

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

```
<!--  注册AccountMapper.xml  -->
    <mappers>
    <mapper resource="com/fxc/mapper/AccountMapper.xml"></mapper>
</mappers>
```

3、调用MyBatis原生接口来执行添加操作

```
public class Test01 {
    public static void main(String[] args) {
        //  加载MyBatis配置文件
        InputStream inputStream =
Test01.class.getClassLoader().getResourceAsStream("config.xml");
        SqlSessionFactoryBuilder builder = new SqlSessionFactoryBuilder();
        SqlSessionFactory factory = builder.build(inputStream);
        SqlSession session = factory.openSession();
        String statement = "com.fxc.mapper.AccountMapper.save";
        Account account = new Account(1, "张三", "123123", 19);
        session.insert(statement, account);
```
```

```
        session.commit();
        session.close();
    }
}
```

> 通过Mapper 代理实现自定义接口

- 自定义接口，定义相关业务方法。
- 编写与方法相对应的Mapper.xml 。

1、自定义接口

```
public interface AccountRepository {
    public int save(Account account);
    public int update(Account account);
    public int delete(int id);
    public List<Account> findAll();
    public Account findById(int id);
}
```

2、创建接口对应的Mapper.xml文件，定义接口方法对应的SQL语句。

statement 标签可根据SQL执行的业务选择 insert、delete、update、select。

MyBatis 框架会根据规则自动创建接口实现类的代理对象。

规则：

- Mapper.xml 中 namespace 为接口的全类名。
- Mapper.xml 中 statement 的 id 为接口中对应的方法名。
- Mapper.xml 中 statement 的 parameterType 和接口中对应方法的参数类型一致。
- Mapper.xml 中 statement 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.fxc.repository.AccountRepository">
    <insert id="save" parameterType="com.fxc.repository.Account">
        insert into t_account (username, password, age) values (#{username}, #
{password}, #{age})
    </insert>
    <update id="update" parameterType="com.fxc.repository.Account">
        update t_account set username=#{username}, password=#{password}, age=#
{age} where id=#{id}
    </update>
    <delete id="delete" parameterType="java.lang.Integer">
        delete from t_account where id=#{id}
    </delete>
    <select id="findAll" resultType="com.fxc.repository.Account">
        select * from t_account
    </select>
    <select id="findById" parameterType="java.lang.Integer"
resultType="com.fxc.repository.Account">
        select * from t_account where id=#{id}
    </select>
</mapper>
```

3、在config.xml 文件中 注册AccountRepository.xml

```
<mapper resource="com/fxc/mapper/AccountRepositoryMapper.xml"></mapper>
```

Mapper.xml

- statement 标签：select、update、delete、insert 分别对应查询、修改、删除、插入操作
- parameterType：参数数据类型

  1、基本数据类型，通过 id 查询 Account

  ```
  <select id="findById" parameterType="int"
  resultType="com.fxc.entity.Account">
      select * from t_account where id=#{id}
  </select>
  ```

  2、String 类型， 通过 name 查询 Account

  ```
  <select id="findByName" parameterType="java.lang.String"
  resultType="com.fxc.entity.Account">
      select * from t_account where username=#{username}
  </select>
  ```

  3、包装类，通过 id 查询 Account

  ```
  <select id="findById" parameterType="java.lang.Integer"
  resultType="com.fxc.entity.Account">
      select * from t_account where id=#{id}
  </select>
  ```

  4、多个参数，通过name 和 age 查询 Account

  ```
  <select id="findByNameAndAge" resultType="com.fxc.entity.Account">
      select * from t_account where username=#{param1} and age=#{param2}
  </select>
  ```

  5、Java Bean

  ```
  <update id="update" parameterType="com.fxc.entity.Account">
      update t_account set username=#{username}, password=#{password}, age=#
  {age} where id=#{id}
  </update>
  ```

- resultType：结果类型

  1、基本数据类型，统计Account总数

  ```
  <select id="count" resultType="int">
      select count(id) from t_account
  </select>
  ```

  2、包装类型，统计Account总数

```
<select id="count" resultType="java.lang.Integer">
    select count(id) from t_account
</select>
```

3、String类型，通过id查询Account的username

```
<select id="findNameById" parameterType="int" resultType="java.lang.String">
    select username from t_account where id=#{id}
</select>
```

4、Java Bean 类型

```
<select id="findById" parameterType="java.lang.Integer"
resultType="com.fxc.entity.Account">
    select * from t_account where id=#{id}
</select>
```

级联查询

Student

```
package com.fxc.entity;

import lombok.Data;

@Data
public class Student {
    private int id;
    private String name;
    private Classes classes;
}
```

Classes

```
package com.fxc.entity;

import lombok.Data;

import java.util.List;

@Data
public class Classes {
    private int id;
    private String name;
    private List<Student> students;
}
```

StudentRepository

```java
package com.fxc.repository;

import com.fxc.entity.Student;

public interface StudentRepository {
    public Student findById(int id);
}
```

StudentRepository.xml

```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.fxc.repository.StudentRepository">
    <resultMap id="studentMap" type="com.fxc.entity.Student">
        <id column="id" property="id"></id>
        <result column="name" property="name"></result>
        <association property="classes" javaType="com.fxc.entity.Classes">
            <id column="cid" property="id"></id>
            <result column="cname" property="name"></result>
        </association>
    </resultMap>
    <select id="findById" parameterType="int" resultMap="studentMap">
        select s.id as id, s.name as name, c.id as cid, c.name as cname from
        student s, classes c where s.id=#{id} and s.cid=c.id;
    </select>
</mapper>
```

ClassesRepository

```java
package com.fxc.repository;

import com.fxc.entity.Classes;

public interface ClassesRepository {
    public Classes findById(int id);
}
```

ClassesRepository.xml

```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.fxc.repository.ClassesRepository">
    <resultMap id="classesMap" type="com.fxc.entity.Classes">
        <id column="cid" property="id"/>
        <result column="cname" property="name"/>
        <collection property="students" ofType="com.fxc.entity.Student">
            <id column="id" property="id"/>
            <result column="name" property="name"/>
        </collection>
    </resultMap>
    <select id="findById" parameterType="int" resultMap="classesMap">
```

```
        select s.id, s.name, c.id as cid, c.name as cname from student s, classes
c where c.id=#{id} and s.cid=c.id
    </select>
</mapper>
```

- 多对多

Customer

```
package com.fxc.entity;

import lombok.Data;

import java.util.List;

@Data
public class Customer {
    private int id;
    private String name;
    private List<Goods> goods;
}
```

Goods

```
package com.fxc.entity;

import lombok.Data;

import java.util.List;

@Data
public class Goods {
    private int id;
    private String name;
    private List<Customer> customers;
}
```

CustomerRepository

```
package com.fxc.repository;

import com.fxc.entity.Customer;

public interface CustomerRepository {
    public Customer findById(int id);
}
```

CustomerRepository.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.fxc.repository.CustomerRepository">
```

```
    <resultMap id="customerMap" type="com.fxc.entity.Customer">
        <id column="cid" property="id"/>
        <result column="cname" property="name"/>
        <collection property="goods" ofType="com.fxc.entity.Goods">
            <id column="gid" property="id"/>
            <result column="gname" property="name"/>
        </collection>
    </resultMap>
    <select id="findById" resultMap="customerMap">
        select c.id cid, c.name cname, g.id gid, g.name gname from customer c,
goods g, customer_goods cg where c.id=#{id} and cg.cid=c.id and cg.gid=g.id
    </select>
</mapper>
```

GoodsRepository

```
package com.fxc.repository;

import com.fxc.entity.Goods;

public interface GoodsRepository {
    public Goods findById(int id);
}
```

GoodsRepository.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.fxc.repository.GoodsRepository">
    <resultMap id="goodsMap" type="com.fxc.entity.Goods">
        <id column="gid" property="id"/>
        <result column="gname" property="name"/>
        <collection property="customers" ofType="com.fxc.entity.Customer">
            <id column="cid" property="id"/>
            <result column="cname" property="name"/>
        </collection>
    </resultMap>
    <select id="findById" resultMap="goodsMap">
        select c.id cid, c.name cname, g.id gid, g.name gname from customer c,
goods g, customer_goods cg where g.id=#{id} and cg.cid=c.id and cg.gid=g.id
    </select>
</mapper>
```

逆向工程

MyBatis 框架需要：实体类、自定义 Mapper 接口、 Mapper.xml

传统的开发中上述的三个组件需要开发者手动创建，逆向工程可以帮助开发者来自动创建三个组件，减轻开发者的工作量，提高工作效率。

如何使用

MyBatis Generator，简称 MBG，是一个专门为 MyBatis 框架开发者定制的代码生成器， 可自动生成 MyBatis 框架所需要的实体类、Mapper 接口、Mapper.xml，支持基本的 CRUD 操作，但是一些相对复杂的 SQL 需要开发者自己来完成。

- 新建 Maven 工程， pom.xml 引入相关依赖

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

    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.38</version>
    </dependency>

    <dependency>
        <groupId>org.mybatis.generator</groupId>
        <artifactId>mybatis-generator-core</artifactId>
        <version>1.3.2</version>
    </dependency>
</dependencies>
```

- 创建 MBG 配置文件 generatorConfig.xml 文件

  1、jdbcConnection 配置数据库连接信息

  2、javaModelGenerator 配置 JavaBean 的生成策略

  3、sqlMapGenerator 配置 SQL 映射文件生成策略

  4、javaClientGenerator 配置 Mapper 接口生成策略

  5、table 配置目标数据表（tableName：表明；domianObjectName：JavaBean 类名）

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE generatorConfiguration
        PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"
        "http://mybatis.org/dtd/mybatis-generator-config_1_0.dtd">
<generatorConfiguration>
    <context id="testTables" targetRuntime="MyBatis3">
        <jdbcConnection
                driverClass="com.mysql.jdbc.Driver"
                connectionURL="jdbc:mysql://localhost:3306/fxc?
useUnicode=true&amp;characterEncoding=UTF-8"
                userId="root"
                password="333"></jdbcConnection>
        <javaModelGenerator targetPackage="com.fxc.entity"
targetProject="./src/main/java"></javaModelGenerator>
        <sqlMapGenerator targetPackage="com.fxc.repository"
targetProject="./src/main/java"></sqlMapGenerator>
        <javaClientGenerator type="XMLMAPPER" targetPackage="com.fxc.repository"
targetProject="./src/main/java"></javaClientGenerator>
        <table tableName="t_user" domainObjectName="User"></table>
```

```
    </context>
</generatorConfiguration>
```

- 创建 Generator 执行类

```
package com.fxc.test;

import org.mybatis.generator.api.MyBatisGenerator;
import org.mybatis.generator.config.Configuration;
import org.mybatis.generator.config.xml.ConfigurationParser;
import org.mybatis.generator.exception.InvalidConfigurationException;
import org.mybatis.generator.exception.XMLParserException;
import org.mybatis.generator.internal.DefaultShellCallback;

import java.io.File;
import java.io.IOException;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        List<String> warnings = new ArrayList<String>();
        boolean overwrite = true;
        String genCig = "/generatorConfig.xml";
        File configFile = new File(Main.class.getResource(genCig).getFile());
        ConfigurationParser parser = new ConfigurationParser(warnings);
        Configuration configuration = null;
        try {
            configuration = parser.parseConfiguration(configFile);
        } catch (IOException e) {
            e.printStackTrace();
        } catch (XMLParserException e) {
            e.printStackTrace();
        }
        DefaultShellCallback callback = new DefaultShellCallback(true);
        MyBatisGenerator generator = null;
        try {
            generator = new MyBatisGenerator(configuration, callback, warnings);
        } catch (InvalidConfigurationException e) {
            e.printStackTrace();
        }
        try {
            generator.generate(null);
        } catch (SQLException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

MyBatis 延迟加载

- 什么是延迟加载

  延迟加载也叫做懒加载、惰性加载。使用延迟加载可以提高程序的运行效率，针对数据持久层的操作，在某些特定的情况下去访问特定的数据库，在其他情况下可以不访问某些表，从一定程度上减少了 Java 应用与数据库的交互次数。

  查询学生和班级时，学生和班级是两张不同的表，如果当前需求只需要获取学生的信息，那么查询学生单表即可，如果需要通过学生获取对应的班级信息，则必须查询两张表。

  不同的业务需求，需要查询不同的表，根据具体的业务需求来动态减少数据表查询的工作就是延迟加载。

  - 在 config.xml 文件中添加

```
<?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>
        <!-- 打印SQL -->
        <setting name="logImpl" value="STDOUT_LOGGING"/>
        <!-- 开启延迟加载 -->
        <setting name="lazyLoadingEnabled" value="true"/>
    </settings>

    <!--  配置MyBatis运行环境  -->
    <environments default="mysql">
        <environment id="mysql">
            <!-- 配置JDBC事务管理 -->
            <transactionManager type="JDBC"></transactionManager>
            <!-- POOLED配置JDBC数据源连接池 -->
            <dataSource type="POOLED">
                <property name="driver" value="com.mysql.jdbc.Driver"/>
                <property name="url"
value="jdbc:mysql://localhost:3306/fxc?
useUnicode=true&amp;characterEncoding=UTF-8"/>
                <property name="username" value="root"/>
                <property name="password" value="333"/>
            </dataSource>
        </environment>
    </environments>

    <!--  注册AccountMapper.xml  -->
    <mappers>
        <mapper resource="com/fxc/mapper/AccountMapper.xml"></mapper>
        <mapper resource="com/fxc/mapper/AccountRepositoryMapper.xml">
</mapper>
        <mapper resource="com/fxc/mapper/StudentRepository.xml">
</mapper>
        <mapper resource="com/fxc/mapper/ClassesRepository.xml">
</mapper>
        <mapper resource="com/fxc/mapper/CustomerRepository.xml">
</mapper>
        <mapper resource="com/fxc/mapper/GoodsRepository.xml"></mapper>
    </mappers>
</configuration>
```

- 将多表关联查询拆分成多个单表查询

  StudentRepository

```
public Student findByIdLazy(int id);
```

  StudentRepository.xml

```
<resultMap id="studentMap" type="com.fxc.entity.Student">
    <id column="id" property="id"></id>
    <result column="name" property="name"></result>
    <association property="classes" javaType="com.fxc.entity.Classes"

select="com.fxc.repository.ClassesRepository.findByIdLazy"
                column="cid">
    </association>
</resultMap>
<select id="findByIdLazy" parameterType="int" resultMap="studentMap">
    select * from student where id = #{id}
</select>
```

  ClassesRepository

```
public Classes findByIdLazy(int id)
```

  ClassesRepository.xml

```
<select id="findByIdLazy" parameterType="int"
resultType="com.fxc.entity.Classes">
    select * from classes where id=#{id}
</select>
```

MyBatis 缓存

- 什么是 MyBatis 缓存

  使用缓存可以减少 Java 应用与数据库的交互次数，从而提升程序的运行效率。比如查询出 id = 1 的对象，第一次查询出之后会自动将该对象保存到缓存中，当下一次查询时，直接从缓存中去取出对象即可，无需再次访问数据库。

- MyBatis 缓存分类

  1、一级缓存： SqlSession 级别，默认开启，且不能关闭

  操作数据库时需要创建 SqlSession 对象，在对象中有一个 HashMap 用于存储缓存数据，不同的 SqlSession 之间缓存数据区域是互不影响的。

  一级缓存的作用是 SqlSession 范围的，当在同一个 SqlSession 中执行两次相同的 SQL 语句时，第一次执行完毕会将结果保存到缓存中，第二次查询时直接从缓存中获取。

  需要注意的是，如果 SqlSession 执行了 DML 操作（insert、update、delete），MyBatis 必须将缓存清空以保证数据的准确性。

  2、二级缓存：Mapper 级别，默认关闭，可以开启

使用二级缓存时，多个 SqlSession 使用同一个 Mapper 的 SQL 语句操作数据库，得到的数据会在二级缓存区，同样使用 HashMap 进行数据存储，相比较于一级缓存，二级缓存的范围更大，多个 SqlSession 可以公用二级缓存，二级缓存是跨 SqlSession 的。

二级缓存是多个 SqlSession 共享的，其作用域是 Mapper 的同一个 namespace，不同的 SqlSession 两次执行相同的 namespace 下的 SQL 语句，参数也相等，则第一次执行成功之后会将数据保存到二级缓存中，第二次可直接从二级缓存中取出数据。

- 一级缓存

> 代码

```
package com.fxc.test;

import com.fxc.entity.Account;
import com.fxc.repository.AccountRepository;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;

import javax.swing.*;
import java.io.InputStream;

/**
 * 测试 MyBatis 缓存机制
 */
public class Test07 {
    public static void main(String[] args) {
        InputStream inputStream =
Test07.class.getClassLoader().getResourceAsStream("config.xml");
        SqlSessionFactoryBuilder builder = new SqlSessionFactoryBuilder();
        SqlSessionFactory factory = builder.build(inputStream);
        SqlSession session = factory.openSession();
        AccountRepository accountRepository =
session.getMapper(AccountRepository.class);
        Account account = accountRepository.findById(1);
        System.out.println(account);
        session.close();
        session = factory.openSession();
        AccountRepository accountRepository1 =
session.getMapper(AccountRepository.class);
        Account account1 = accountRepository1.findById(1);
        System.out.println(account1);
    }
}
```

> 运行结果

```
Fri Jan 15 15:16:23 CST 2021 WARN: Establishing SSL connection without server's
identity verification is not recommended. According to MySQL 5.5.45+, 5.6.26+ and
5.7.6+ requirements SSL connection must be established by default if explicit
option isn't set. For compliance with existing applications not using SSL the
verifyServerCertificate property is set to 'false'. You need either to
explicitly disable SSL by setting useSSL=false, or set useSSL=true and provide
truststore for server certificate verification.
Created connection 1225439493.
```

```
Setting autocommit to false on JDBC Connection
[com.mysql.jdbc.JDBC4Connection@490ab905]
==>  Preparing: select * from t_account where id=?
==> Parameters: 1(Integer)
<==    Columns: id, username, password, age
<==        Row: 1, 张三, 123123, 19
<==      Total: 1
Account(id=1, username=张三, password=123123, age=19)
Resetting autocommit to true on JDBC Connection
[com.mysql.jdbc.JDBC4Connection@490ab905]
Closing JDBC Connection [com.mysql.jdbc.JDBC4Connection@490ab905]
Returned connection 1225439493 to pool.
Opening JDBC Connection
Checked out connection 1225439493 from pool.
Setting autocommit to false on JDBC Connection
[com.mysql.jdbc.JDBC4Connection@490ab905]
==>  Preparing: select * from t_account where id=?
==> Parameters: 1(Integer)
<==    Columns: id, username, password, age
<==        Row: 1, 张三, 123123, 19
<==      Total: 1
Account(id=1, username=张三, password=123123, age=19)

Process finished with exit code 0
```

- 二级缓存

1、 MyBatis 自带的二级缓存

- config.xml 配置开启二级缓存

```
<settings>
    <!-- 打印SQL -->
    <setting name="logImpl" value="STDOUT_LOGGING"/>
    <!-- 开启延迟加载 -->
    <setting name="lazyLoadingEnabled" value="true"/>
    <!-- 开启二级缓存-->
    <setting name="cacheEnabled" value="true"/>
</settings>
```

- Mapper.xml 中添加二级缓存

```
<cache></cache>
```

- 实体类实现序列化接口

```java
package com.fxc.entity;
import lombok.AllArgsConstructor;
import lombok.Data;
import java.io.Serializable;
@Data
@AllArgsConstructor
public class Account implements Serializable {
    private int id;
    private String username;
    private String password;
    private int age;
}
```

2、ehcache 二级缓存（第三方缓存）

- pom.xml 添加相关依赖

```xml
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis-ehcache</artifactId>
    <version>1.0.0</version>
</dependency>
<dependency>
    <groupId>net.sf.ehcache</groupId>
    <artifactId>ehcache-core</artifactId>
    <version>2.4.3</version>
</dependency>
```

- 添加 ehcache.xml

```xml
<ehcahce xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="../config/ehcache.xsd">
    <diskStroe/>
    <defaultCache
        maxElementsInMemory="1000"
        maxElementsOnDisk="10000000"
        eternal="false"
        overflowToDisk="false"
        timeToIdleSeconds="120"
        timeToLiveSeconds="120"
        diskExpiryThreadIntervalSeconds="120"
        memoryStoreEvictionPolicy="LRU">
    </defaultCache>
</ehcahce>
```

- config.xml 配置开启二级缓存

```xml
<settings>
    <!-- 打印SQL -->
    <setting name="logImpl" value="STDOUT_LOGGING"/>
    <!-- 开启延迟加载 -->
    <setting name="lazyLoadingEnabled" value="true"/>
    <!-- 开启二级缓存-->
    <setting name="cacheEnabled" value="true"/>
</settings>
```

- Mapper.xml 中配置二级缓存

```
<cache type="org.mybatis.caches.ehcache.EhcacheCache">
    <!-- 缓存创建之后，最后一次访问缓存的时间至缓存失效的时间 -->
    <property name="timeToIdleSeconds" value="3600"/>
    <!-- 缓存自创建时间起至失效的时间间隔 -->
    <property name="timeToLiveSeconds" value="3600"/>
    <!-- 缓存回收策略，LRU表示移除近期使用最少的对象 -->
    <property name="memoryStoreEvictionPolicy" value="LRU"/>
</cache>
```

- 实体类不需要实现序列化接口

```
package com.fxc.entity;

import lombok.AllArgsConstructor;
import lombok.Data;

import java.io.Serializable;

@Data
@AllArgsConstructor
public class Account{
    private int id;
    private String username;
    private String password;
    private int age;
}
```

MyBatis 动态 SQL

使用动态 SQL 可简化代码的开发，减少开发者的工作量，程序可以自动根据业务参数来决定 SQL 的组成。

- if 标签

```
<select id="findByAccount" parameterType="com.fxc.entity.Account"
resultType="com.fxc.entity.Account">
    select * from t_account where
    <if test="username != null">
        username=#{username}
    </if>
    <if test="password != null">
        and password=#{password}
    </if>
    <if test="age>=1">
        and age=#{age}
    </if>
</select>
```

if 标签可以自动根据表达式的结果来决定是否将对应的语句添加到 SQL 中，如果条件不成立不添加，否则添加

- where 标签

```
<select id="findByAccount" parameterType="com.fxc.entity.Account"
resultType="com.fxc.entity.Account">
    select * from t_account
    <where>
        <if test="username != null">
            username=#{username}
        </if>
        <if test="password != null">
            and password=#{password}
        </if>
        <if test="age>=1">
            and age=#{age}
        </if>
    </where>
</select>
```

where 标签可以自动判断是否要删除语句块中的 and 关键字，如果检测到 where 直接跟 and 拼接，则自动删除 and，通常情况下 if 和 where 结合起来使用

- choose、when 标签

```
<select id="findByAccount" parameterType="com.fxc.entity.Account"
resultType="com.fxc.entity.Account">
    select * from t_account
    <where>
        <choose>
            <when test="id!=0">
                id=#{id}
            </when>
            <when test="username!=null">
                username=#{username}
            </when>
            <when test="password!=null">
                password=#{password}
            </when>
            <when test="age!=0">
                age=#{age}
            </when>
        </choose>
    </where>
</select>
```

- trim 标签

  trim 标签中的 prefix 和 suffix 属性会被用于生成实际的 SQL 语句，会和标签内部的语句进行拼接，如果语句前后出现了 prefixOverrides 或者 suffixOverrides 属性中指定的值， MyBatis 框架会自动将其删除。

```
<select id="findByAccount" parameterType="com.fxc.entity.Account"
resultType="com.fxc.entity.Account">
    select * from t_account
    <trim prefix="where" prefixOverrides="and">
        <if test="id!=0">
            id=#{id}
        </if>
        <if test="username!=null">
            and username=#{username}
```

```
        </if>
        <if test="password">
            and password=#{password}
        </if>
        <if test="age!=0">
            and age=#{age}
        </if>
    </trim>
</select>
```

- set 标签

  set 标签用于 update 操作，会自动根据参数选择生成 SQL 语句。

```
<update id="update" parameterType="com.fxc.entity.Account">
    update t_account
    <set>
        <if test="username!=null">
            username=#{username}
        </if>
        <if test="password!=null">
            password=#{password}
        </if>
        <if test="age!=0">
            age=#{age}
        </if>
    </set>
    where id=#{id}
</update>
```

- foreach 标签

  foreach 标签可以迭代生成一系列值，这个标签主要用于 SQL 的 in 语句。

```
<select id="findByIds" parameterType="com.fxc.entity.Account"
resultType="com.fxc.entity.Account">
    select * from t_account
    <where>
        <foreach collection="ids" open="id in (" close=")" item="id"
separator=",">
            #{id}
        </foreach>
    </where>
</select>
```