MyBatis 延迟加载

1、创建实体类

```
package com.southwind.entity;

import lombok.Data;

import java.util.List;

@Data
public class Customer {
    private Integer id;
    private String name;
    private List<Order> orders;
}
```

```
package com.southwind.entity;

import lombok.Data;

@Data
public class Order {
    private Integer id;
    private String name;
    private Customer customer;
}
```

2、config.xml 中开启打印 SQL

```
<settings>
    <setting name="logImpl" value="STDOUT_LOGGING" />
</settings>
```

3、创建 OrderRepository

```
package com.southwind.repository;

import com.southwind.entity.Order;

public interface OrderRepository {
    public Order findById(Integer id);
}
```

```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.repository.OrderRepository">

    <resultMap id="orderMap" type="com.southwind.entity.Order">
        <id column="id" property="id"></id>
        <result column="name" property="name"></result>
        <association
                property="customer"
                javaType="com.southwind.entity.Customer"
                select="com.southwind.repository.CustomerRepository.findById"
                column="cid"
        ></association>
    </resultMap>

    <select id="findById" parameterType="java.lang.Integer" resultMap="">
        select * from orders where id = #{id}
    </select>

</mapper>
```

id，name映射

id：16，name：订单1，cid：10

类型

从哪里来

我们要通过cid查找customer，通过select * from customer才能找到

4、创建 CustomerRepository

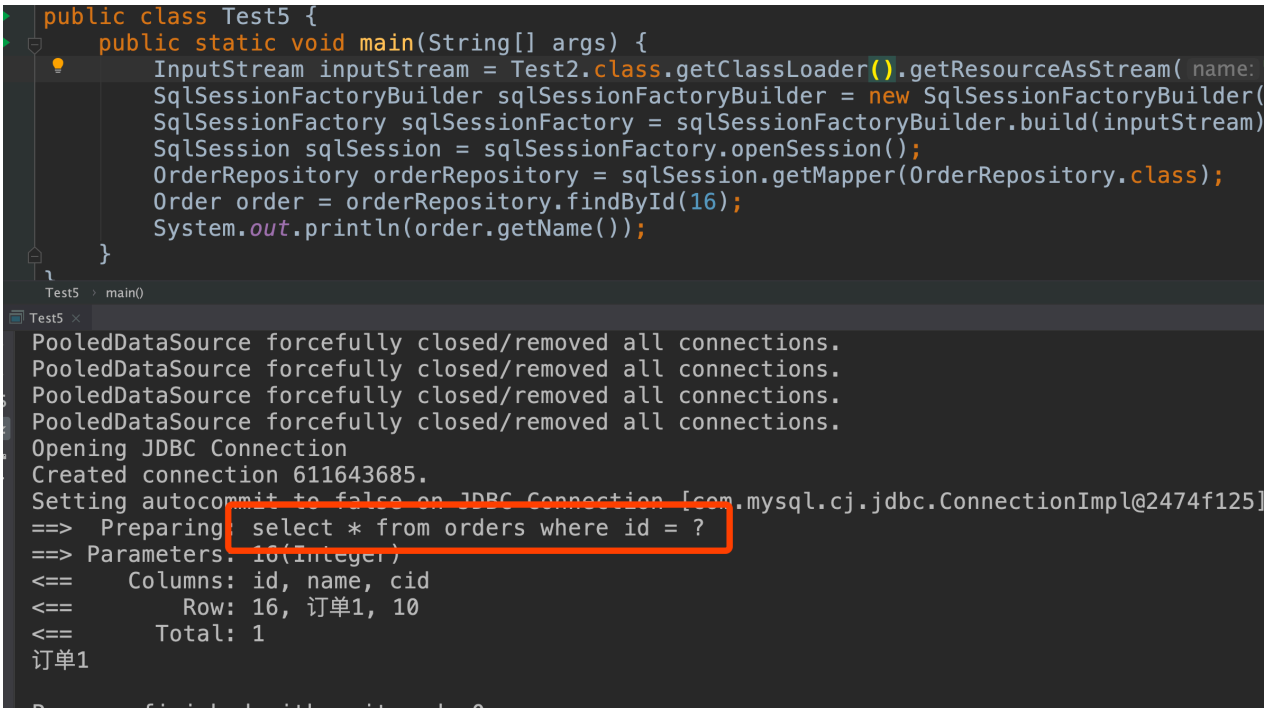
```java
package com.southwind.repository;

import com.southwind.entity.Customer;

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

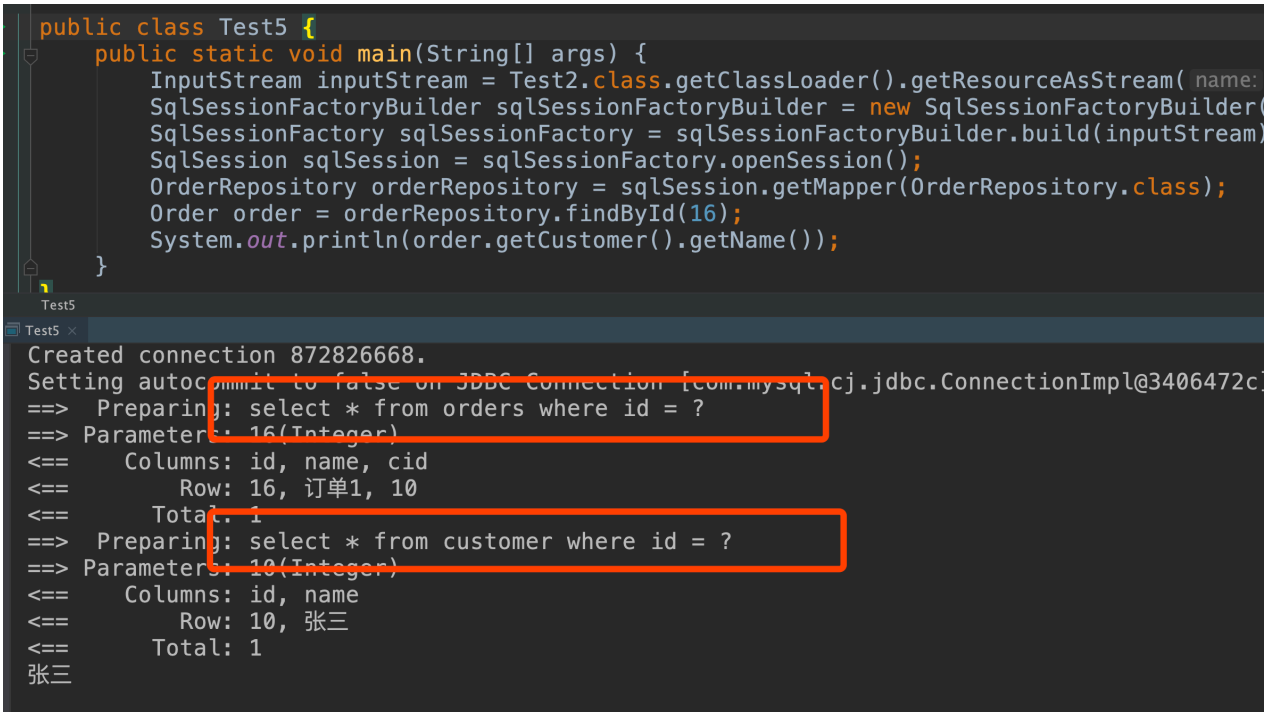
```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.southwind.repository.CustomerRepository">

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

</mapper>
```

5、config.xml 中开启延迟加载

```
<!-- 延迟加载 -->
<setting name="lazyLoadingEnabled" value="true"/>
```

这里把两个sql都结果都执行了



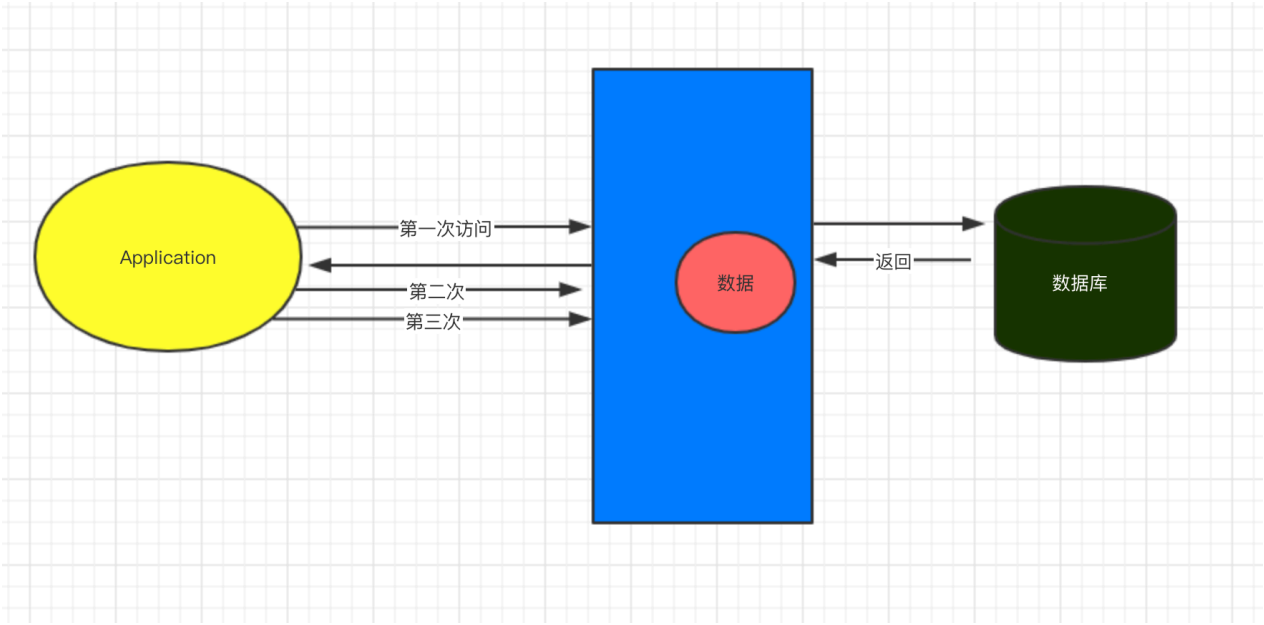


按需加载，如果我只需要订单信息就不要执行第二条sql了，比如入参为order.getname

MyBatis 延迟加载机制，是实际开发中使用频率较高的一个功能，正确地使用延迟加载，可以有效减少 Java Application 和数据库的交互次数，从而提高整个系统的运行效率，延迟加载是为了提高程序运行效率的一种手段，一般应用于多表关联查询的业务场景。

MyBatis 缓存

使用缓存的作用也是为了减少 Java Application 与数据库的交互次数，从而提升程序的运行效率。



MyBatis 有两种缓存：一级缓存和二级缓存。

一级缓存

MyBatis 自带一级缓存，并且是无法关闭的，一直存在，一级缓存的数据存储在 SqlSession 中。

即使用同一个 SqlSession 进行查询操作的时候，一级缓存存在，如果使用多个 SqlSession 进行查询操作，一级缓存不存在，缓存只针对于查询，但是如果 SqlSession 执行了增、删、改操作，MyBatis 会自动清空 SqlSession 缓存中的数据，以此来保证数据的一致性。

一级缓存不需要进行任何配置，直接使用即可。

1、实体类

```
package com.southwind.entity;

import lombok.Data;

@Data
public class MyClass {
    private Integer id;
    private String name;
}
```

2、Mapper.java

```java
package com.southwind.repository;

import com.southwind.entity.MyClass;

public interface MyClassRepository {
    public MyClass findById(Integer id);
}
```

3、Mapper.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.southwind.repository.MyClassRepository">

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

</mapper>
```

4、Test

```java
package com.southwind.test;

import com.southwind.entity.MyClass;
import com.southwind.repository.MyClassRepository;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;

import java.io.InputStream;

public class Test6 {
    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();
        MyClassRepository myClassRepository =
sqlSession.getMapper(MyClassRepository.class);
        MyClass myClass = myClassRepository.findById(1);
        System.out.println(myClass);
        MyClass myClass2 = myClassRepository.findById(1);
```

```
        System.out.println(myClass2);
    }
}
```

```
WARNING: All illegal access operations will be denied in a future re
Logging initialized using 'class org.apache.ibatis.logging.stdout. t
PooledData ource forcefully closed/removed all connections.
PooledData ource forcefully closed/removed all connections.
PooledData ource forcefully closed/removed all connections.
PooledData ource forcefully closed/removed all connections.
Opening JDBC Connection
Created connection 611643685.
 etting autocommit to false on JDBC Connection [com.mysql.cj.jdbc.Co
==>  Preparing: select * from t_classes where id = ?
==> Parameters: 1(Integer)
<==    Columns: id, name
<==        Row: 1, 一班
<==      Total: 1
MyClass(id=1, name=一班)
MyClass(id=1, name=一班)

Process finished with exit code 0
```

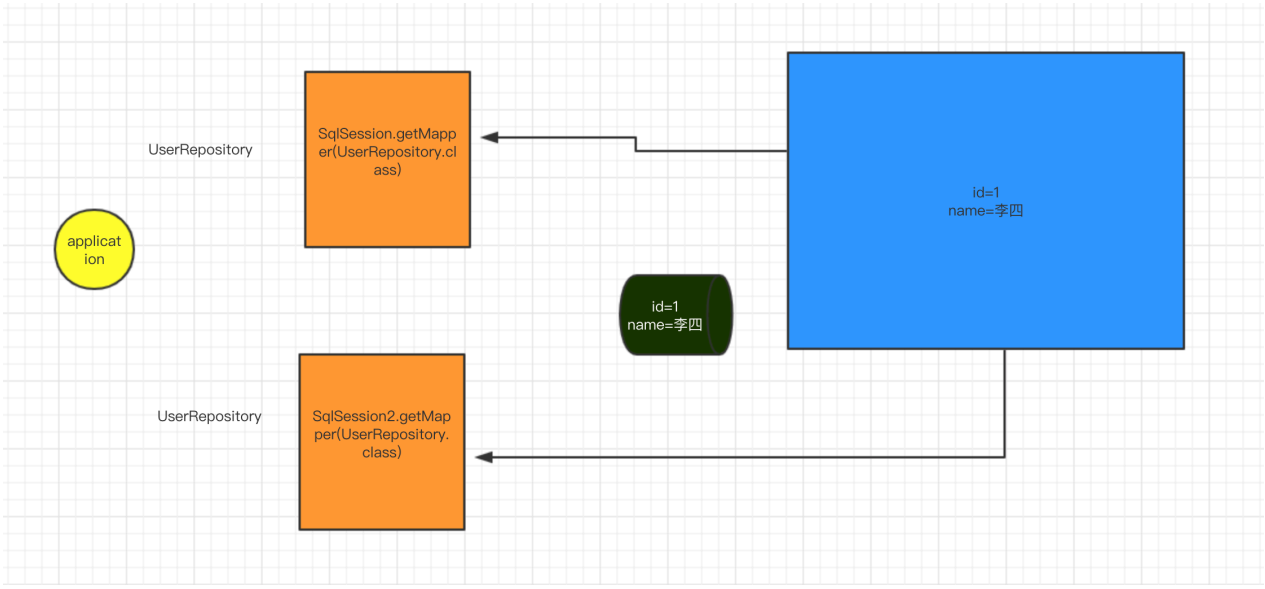
```
package com.southwind.test;

import com.southwind.entity.MyClass;
import com.southwind.repository.MyClassRepository;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;

import java.io.InputStream;

public class Test6 {
    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();
        MyClassRepository myClassRepository =
sqlSession.getMapper(MyClassRepository.class);
        MyClass myClass = myClassRepository.findById(1);
        System.out.println(myClass);
        //关闭sqlSession
        sqlSession.close();
        //开启新的sqlSession
        sqlSession = sqlSessionFactory.openSession();
        myClassRepository = sqlSession.getMapper(MyClassRepository.class);
```

```
        MyClass myClass2 = myClassRepository.findById(1);
        System.out.println(myClass2);
    }
}
```

```
Opening JDBC Connection
Created connection 611643685.
 etting autocommit to false on JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@2474f12
==>  Preparing: select * from t_classes where id = ?
==> Parameters: 1(Integer)
<==    Columns: id, name
<==        Row: 1, 一班
<==      Total: 1
MyClass(id=1, name=一班)
Resetting autocommit to true on JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@2474f1
Closing JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@2474f125]
Returned connection 611643685 to pool.
Opening JDBC Connection
Checked out connection 611643685 from pool.
 etting autocommit to false on JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@2474f12
==>  Preparing: select * from t_classes where id = ?
==> Parameters: 1(Integer)
<==    Columns: id, name
<==        Row: 1, 一班
<==      Total: 1
```

二级缓存

MyBatis 二级缓存是比一级缓存作用域更大的缓存机制，它是 Mapper 级别的，只要是同一个 Mapper，无论使用多少个 SqlSession，数据都是共享的。



MyBatis 二级缓存默认是关闭的，需要使用时可以通过配置手动开启。

MyBatis 可以使用自带的二级缓存，也可以使用第三方 ehcache 二级缓存。

自带二级缓存

1、config.xml 中配置开启二级缓存。

```
<settings>
    <!-- 开启二级缓存 -->
    <setting name="cacheEnabled" value="true"/>
</settings>
```

2、Mapper.xml 中配置二级缓存。

```
<?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.MyClassRepository">

    <cache></cache>

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

</mapper>
```

3、实体类实现 Serializable 接口。

实现序列化才能存到二级缓存中

映射的实体类需要序列化：
理由：由于二级缓存的数据不一定都是存储到内存中，它的存储介质多种多样，所以需要给缓存的对象执行序列化。

```
package com.southwind.entity;

import lombok.Data;

import java.io.Serializable;

@Data
public class MyClass implements Serializable {
    private Integer id;
    private String name;
}
```

1、序列化是干什么的？
简单说就是为了保存在内存中的各种对象的状态，并且可以把保存的对象状态再读出来。虽然你可以用你自己的各种各样的方法来保存Object States，但是Java给你提供一种应该比你自己好的保存对象状态的机制,那就是序列化。

2、什么情况下需要序列化

a）当你想把的内存中的对象保存到一个文件中或者数据库中时候；
b）当你想用套接字在网络上传送对象的时候；
c）当你想通过RMI传输对象的时候；

4、测试。

```
package com.southwind.test;

import com.southwind.entity.MyClass;
import com.southwind.repository.MyClassRepository;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;

import java.io.InputStream;
```

```
public class Test6 {
    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();
        MyClassRepository myClassRepository =
sqlSession.getMapper(MyClassRepository.class);
        MyClass myClass = myClassRepository.findById(1);
        System.out.println(myClass);
        sqlSession.close();
        sqlSession = sqlSessionFactory.openSession();
        myClassRepository = sqlSession.getMapper(MyClassRepository.class);
        MyClass myClass2 = myClassRepository.findById(1);
        System.out.println(myClass2);
    }
}
```

```
PooledDataSource forcefully closed/removed all connections.
PooledDataSource forcefully closed/removed all connections.
PooledDataSource forcefully closed/removed all connections.
Cache Hit Ratio [com.southwind.repository.MyClassRepository]: 0.0
Opening JDBC Connection
Created connection 417557780.
Setting autocommit to false on JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@18e36d14]
==>  Preparing: select * from t_classes where id = ?
==> Parameters: 1(Integer)
<==    Columns: id, name
<==        Row: 1, 一班
<==      Total: 1
MyClass(id=1, name=一班)
Resetting autocommit to true on JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@18e36d14]
Closing JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@18e36d14]
Returned connection 417557780 to pool.
Cache Hit Ratio [com.southwind.repository.MyClassRepository]: 0.5
MyClass(id=1, name=一班)

Process finished with exit code 0
```

第三方 ehcache 二级缓存

1、pom.xml

```
<!-- ehcache -->
<dependency>
    <groupId>net.sf.ehcache</groupId>
    <artifactId>ehcache-core</artifactId>
    <version>2.4.3</version>
</dependency>

<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis-ehcache</artifactId>
    <version>1.0.0</version>
</dependency>
```

2、resources 路径下创建 ehcache.xml

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

3、config.xml 中配置二级缓存。

```
<settings>
    <!-- 开启二级缓存 -->
    <setting name="cacheEnabled" value="true"/>
</settings>
```

4、Mapper.xml 配置二级缓存。

```
<?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.MyClassRepository">

    <!-- 开启二级缓存 -->
    <cache type="org.mybatis.caches.ehcache.EhcacheCache" >
        <!-- 缓存创建以后，最后一次访问缓存的时间至失效的时间间隔 -->
```

```
        <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移除近期最少使用的对象 -->
        <property name="memoryStoreEvictionPolicy" value="LRU"/>
    </cache>

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

</mapper>
```

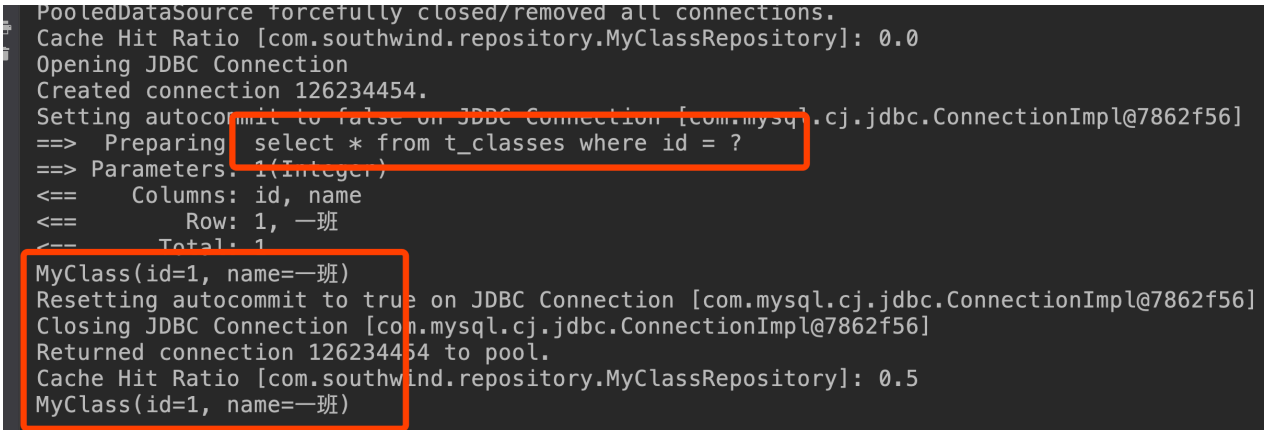
5、实体类不需要实现序列化接口。

```
package com.southwind.entity;

import lombok.Data;

@Data
public class MyClass {
    private Integer id;
    private String name;
}
```

6、测试

```
package com.southwind.test;

import com.southwind.entity.MyClass;
import com.southwind.repository.MyClassRepository;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;

import java.io.InputStream;

public class Test6 {
    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();
```

```
        MyClassRepository myClassRepository =
sqlSession.getMapper(MyClassRepository.class);
        MyClass myClass = myClassRepository.findById(1);
        System.out.println(myClass);
        sqlSession.close();
        sqlSession = sqlSessionFactory.openSession();
        myClassRepository = sqlSession.getMapper(MyClassRepository.class);
        MyClass myClass2 = myClassRepository.findById(1);
        System.out.println(myClass2);
    }
}
```



MyBatis 动态 SQL

```
public class User{
  private Integer id;
  private String username;
  private String password;
  private Integer age;
}
```

- 通过 id 和 username 查询 User。
- 通过 username 和 password 查询 User。
- 通过 password 和 age 查询 User。

在 UserRepository 中定义上述 3 个方法

```
public User findByUser1(User user);
public User findByUser2(User user);
public User findByUser3(User user);
```

UserRepository.xml

```
<select id="findByUser1" parameterType="User" resultType="User">
  select * from t_user where id = #{id} and username = #{username}
</select>
<select id="findByUser2" parameterType="User" resultType="User">
  select * from t_user where username = #{username} and password = #{password}
</select>
<select id="findByUser3" parameterType="User" resultType="User">
  select * from t_user where password = #{password} and age = #{age}
</select>
```

MyBatis 动态 SQL，SQL 不是固定的，可以根据不同的参数信息来动态拼接不同的 SQL，以适应不同的需求。

1、创建实体类

```
public class User{
  private Integer id;
  private String username;
  private String password;
  private Integer age;
}
```

2、Mapper.java

```
package com.southwind.repository;

import com.southwind.entity.User;

public interface UserRepository {
    public User findByUser(User user);
}
```

3、Mapper.xmll

```
<?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.UserRepository">

    <select id="findByUser" parameterType="com.southwind.entity.User"
resultType="com.southwind.entity.User">
        select * from t_user where
        id = #{id} and username = #{username}
        and password = #{password} and age = #{age}
    </select>

</mapper>
```

4、config.xml 注册 Mapper.xml

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

5、测试

```
package com.southwind.test;

import com.southwind.entity.User;
import com.southwind.repository.UserRepository;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;

import java.io.InputStream;

public class Test7 {
    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();
        UserRepository userRepository =
sqlSession.getMapper(UserRepository.class);
        User user = new User();
//        user.setId(2);
        user.setUsername("lisi");
        user.setPassword("000");
        user.setAge(5);
        User user2 = userRepository.findByUser(user);
        System.out.println(user2);
    }
}
```

动态 SQL

- if
- where

```
<?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.UserRepository">

    <select id="findByUser" parameterType="com.southwind.entity.User"
resultType="com.southwind.entity.User">
        select * from t_user
        <where>
            <if test="id!=null">
                id = #{id}
            </if>
            <if test="username!=null">
                and username = #{username}
            </if>
            <if test="password!=null">
                and password = #{password}
            </if>
            <if test="age!=null">
                and age = #{age}
            </if>
        </where>
    </select>

</mapper>
```

- choose、when

```
<select id="findByUser" parameterType="com.southwind.entity.User"
resultType="com.southwind.entity.User">
    select * from t_user
    <where>
        <choose>
            <when test="id!=null">
                id = #{id}
            </when>
            <when test="username!=null">
                and username = #{username}
            </when>
            <when test="password!=null">
                and password = #{password}
            </when>
            <when test="age!=null">
                and age = #{age}
            </when>
        </choose>
    </where>
</select>
```

- trim

通过设置 prefix 和 suffix 参数来完成使用的。

```
<select id="findByUser" parameterType="com.southwind.entity.User"
resultType="com.southwind.entity.User">
    select * from t_user
    <trim prefix="where" prefixOverrides="and">
        <if test="id!=null">
            id = #{id}
        </if>
        <if test="username!=null">
            and username = #{username}
        </if>
        <if test="password!=null">
            and password = #{password}
        </if>
        <if test="age!=null">
            and age = #{age}
        </if>
    </trim>
</select>
```

- set

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

Mapper.java

```
package com.southwind.repository;

import com.southwind.entity.User;

public interface UserRepository {
    public int update(User users);
}
```

Mapper.xml

```
<update id="update" parameterType="com.southwind.entity.User">
  update t_user
  <set>
    <if test="username!=null">
      username = #{username},
    </if>
    <if test="password!=null">
      password = #{password},
    </if>
    <if test="age!=null">
      age = #{age}
    </if>
```

```
  </set>
  where id = #{id}
</update>
```

```
package com.southwind.test;

import com.southwind.entity.User;
import com.southwind.repository.UserRepository;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;

import java.io.InputStream;

public class Test7 {
    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();
        UserRepository userRepository =
sqlSession.getMapper(UserRepository.class);
        User user = new User();
        user.setId(2);
        user.setUsername("tom");
        user.setAge(3);
        userRepository.update(user);
        sqlSession.commit();
    }
}
```

```
PooledDataSource forcefully closed/removed all connections.
PooledDataSource forcefully closed/removed all connections.
Opening JDBC Connection
Created connection 473053293.
Setting autocommit to false on JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@1c32386d]
==>  Preparing: update t_user SET username = ?, age = ? where id = ?
==> Parameters: tom(String), 3(Integer), 2(Integer)
<==    Updates: 1
Committing JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@1c32386d]

Process finished with exit code 0
```