

```
<?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的环境 default指定id=mysql的环境-->
  <environments default="mysql">
    <environment id="mysql">
      <!--配置事务管理策略-->
      <transactionManager type="JDBC"></transactionManager>
      <!--配置数据源-->
      <dataSource type="POOLED">
        <property name="driver" value="com.mysql.jdbc.Driver"/>
        <property name="url" value="jdbc:mysql://itcastmybatis"/>
        <property name="username" value="root"/>
        <property name="password" value="root"/>
      </dataSource>
    </environment>
  </environments>
  <!--配置映射文件的信息-->
  <mapppers>
    <mapper resource="com/itheima/dao/UserDao.xml"></mapper>
  </mapppers>
</configuration>
```

找到映射文件的位置

```
<?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.itheima.dao.UserDao">
  <!--查询所有-->
  <select id="findAll" resultType="com.itheima.domain.User">
    select * from user
  </select>
</mapper>
```

找到接口的位置

sqlMapConfig.xml, 创建在resources目录下

```
public class MyBatisTest {
    public static void main(String[] args) throws IOException {
        InputStream in =
            Resources.getResourceAsStream("sqlMapConfig.xml");
        SqlSessionFactoryBuilder builder = new SqlSessionFactoryBuilder();
        SqlSessionFactory sqlSessionFactory = builder.build(in);
        SqlSession sqlSession = sqlSessionFactory.openSession();
        UserDao userDao = sqlSession.getMapper(UserDao.class);
        List<User> list = userDao.findAll();
        for (User user : list) {
            System.out.println(user);
        }
        sqlSession.close();
        in.close();
    }
}
```

测试类

UserDao.xml映射文件, 创建在resources下, 创建包com.itheima.dao 要与接口路径对应

<!--保存用户-->

```
<insert id="saveUser" parameterType="com.itheima.domain.User">
    insert into user(username,address,sex,birthday)values(#{username},#{address},#{sex},#{birthday});
</insert>
```

这个sql语句中使用#{ }字符代表占位符，它们都是代表占位符，具体的值是由User类的username属性来决定的。使用OGNL表达式

注意

```
//提交事务
sqlSession.commit();
//释放资源
sqlSession.close();
//关闭读取资源的流
in.close();
```

问题

新增用户后，同时还要返回当前新增用户的id值，因为id是由数据库的自动增长来实现的，所以就相当于我们要在新增后将自动增长auto_increment的值返回。

解决方法一： UserDao.xml

<!--保存用户-->

```
<insert id="saveUser" parameterType="com.itheima.domain.User">
```

```
<!-- 配置插入操作后，获取插入数据的id -->
```

```
<selectKey keyProperty="id" keyColumn="id" resultType="int" order="AFTER">
```

```
    select last_insert_id();
```

```
</selectKey>
```

```
    insert into user(username,address,sex,birthday)values(#{username},#{address},#{sex},#{birthday});
</insert>
```

解决方法二: UserDao.xml

```
<insert id="saveUser" parameterType="com.itheima.domain.User" keyProperty="id"
useGeneratedKeys="true">
    INSERT INTO t_user(username,sex,birthday,address) VALUES
    (#{username},#{sex},#{birthday},#{address}) </insert>
```

解决方法三: SqlMapConfig.xml

```
<settings>
    <setting name="useGeneratedKeys" value="true"/>
</settings>
```

属性	描述
keyProperty	selectKey 语句结果应该被设置的目标属性。
keyColumn	查询哪个字段的结果, 进行封装。
resultType	结果的类型。MyBatis 通常可以算出来,但是写上也没有问题。MyBatis 允许任何简单类型用作主键的类型,包括字符串。
order	这可以被设置为 BEFORE 或 AFTER。如果设置为 BEFORE,那么它会首先选择主键,设置 keyProperty 然后执行插入语句。如果设置为 AFTER,那么先执行插入语句,然后是 selectKey 元素-这和如 Oracle 数据库相似,可以在插入语句中嵌入序列调用。

修改用户

```
<update id="updateUser" parameterType="com.itheima.domain.User">
    update user set username=#{username},address=#{address},sex=#{sex},birthday=#{birthday} where
    id=#{id}
</update>
```

<!-- 删除用户 -->

```
<delete id="deleteUser" parameterType="java.lang.Integer">
    delete from user where id = #{uid}
</delete>
```

删除的细节

其中的#{uid}是占位符，代表参数的值由方法的参数传入进来的。

注意：

1. 此处的#{uid}中的id其实只是一个形参，所以它的名称是自由定义的，比如定义成#{abc}也是可以的。
 2. parameterType取值，对于基本类型我们可以直接写成int,short,double.....也可以写成java.lang.Integer。
 3. 字符串可以写成string,也可以写成java.lang.String
- 也就是说：int是java.lang.Integer的别名
string是java.lang.String的别名
别名是不区分大小写

<!-- 根据id查询用户-->

```
<select id="findByld" parameterType="INT" resultType="com.itheima.domain.User">
    select * from user where id = #{uid}
</select>
```

<!-- 根据名称模糊查询 一-->

```
<select id="findByName" parameterType="string" resultType="com.itheima.domain.User">
    select * from user where username like concat('%',#{name},'%')
</select>
```

注意：此时的#{name}，因为这时候是普通的参数，所以它的起名是随意的，比如我们改成#{abc}也是可以的。

<!-- 根据名称模糊查询 二-->

```
<select id="findByName" parameterType="string" resultType="com.itheima.domain.User">
    select * from user where username like '%${value}%'
</select>
```

我们在上面将原来的#{ }占位符，改成了\${value}。
注意如果用模糊查询的这种写法，
那么\${value}的写法就是固定的，不能写成其它名字。

#{ } 表示一个占位符号

通过#{ }可以实现 preparedStatement 向占位符中设置值, 自动进行 java 类型和 数据库 类型转换

#{ }可以有效防止 sql 注入

#{ }可以接收简单类型值(基本类型,String)或 pojo 属性值

如果 parameterType 传输单个简单类型值(基本类型,String),
#{ } 括号中可以是任意名称。

\${ } 表示拼接 sql 串

通过\${ }可以将 parameterType 传入的内容拼接在 sql 中且不进行 jdbc 类型转换.

\${ }不能防止 sql 注入

\${ }可以接收简单类型值或 pojo 属性值

如果 parameterType 传输单个简单类型值.\${ }括号中只能是 value

<!-- 查询所有 -->

```
<select id="findAll" resultMap="userMap">
  select * from user;
</select>
```

<!-- 配置 查询结果的列名和实体类的属性名的对应关系 -->

```
<resultMap id="userMap" type="com.itheima.domain.User">
  <!-- 主键字段的对应 -->
  <id property="userId" column="id"></id>
  <!-- 非主键字段的对应 -->
  <result property="userName" column="username"></result>
  <result property="userAddress" column="address"></result>
  <result property="userSex" column="sex"></result>
  <result property="userBirthday" column="birthday"></result>
</resultMap>
```

实体类的字段

数据库的字段

其他方案一：别名

```
select id as userId,username as userName,address as userAddress,sex as userSex,birthday as userBirthday from user;
```

其他方案二：只适用于下划线转驼峰

```
<setting name="mapUnderscoreToCamelCase" value="true"/>
```

配置内容

SqlMapConfig.xml中配置的内容和顺序如下：

properties（属性）

settings（全局配置参数）

typeAliases（类型别名）

typeHandlers（类型处理器）

objectFactory（对象工厂）

plugins（插件）

environments（环境集合属性对象）

environment（环境子属性对象）

transactionManager（事务管理）

dataSource（数据源）

mappers（映射器）


```
<environments default="mysql">
  <environment id="mysql">
    <transactionManager type="JDBC"></transactionManager>
    <dataSource type="POOLED">
      <property name="driver" value="com.mysql.jdbc.Driver"></property>
      <property name="url" value="jdbc:mysql:///itheimamybatis"></property>
      <property name="username" value="root"></property>
      <property name="password" value="root"></property>
    </dataSource>
  </environment>
</environments>
```

第一步：在src\main\resources下定义jdbcConfig.properties文件

```
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/mybatis
jdbc.username=root
jdbc.password=root
```

第二步：在sqlMapConfig.xml中配置：

```
<properties resource="jdbcConfig.properties"></properties>
<environments default="mysql">
  <environment id="mysql">
    <transactionManager type="JDBC"></transactionManager>
    <dataSource type="POOLED">
      <property name="driver" value="${jdbc.driver}"></property>
      <property name="url" value="${jdbc.url}"></property>
      <property name="username" value="${jdbc.username}"></property>
      <property name="password" value="${jdbc.password}"></property>
    </dataSource>
  </environment>
</environments>
```

在SqlMapConfig.xml中配置（配置别名）

```
<typeAliases>
<typeAlias type="com.itheima.domain.User" alias="user"></typeAlias>
</typeAliases>
```

第一种：配置一个

typeAlias用于配置别名。
type属性指定的是实体类全限定类名。
alias属性指定别名，当指定了别名就不再区分大小写

```
<package name="com.itheima.domain"></package>
```

第二种：配置domain包下所有的类，
把该代码添加到箭头位置即可

在UserDao.xml中配置（使用别名）

```
<!--查询用户-->
<select id="findAll" resultType="UsEr">
    select * from user;
</select>
```

```
<!--保存用户-->
<insert id="saveUser" parameterType="UsEr">
    <selectKey keyProperty="userId" keyColumn="id" resultType="int" order="AFTER">
        select last_insert_id();
    </selectKey>
    insert into user(username,address,sex,birthday)values(#{userName},#{userAddress},#{userSex},#{userBirthday});
</insert>
```

1: `<mapper resource="" />`

使用相对于类路径的资源

如: `<mapper resource="com/itheima/dao/UserDao.xml" />`

应用场景: 指定XML映射配置

2: `<mapper class="" />`

使用mapper接口类路径

如: `<mapper class="com.itheima.dao.UserDao"/>`

注意: 此种方法要求mapper接口名称和mapper映射文件名称相同, 且放在同一个目录中。

应用场景: 指定注解配置

3: `<package name="" />`

注册指定包下的所有mapper接口

如: `<package name="cn.itcast.dao"/>`

注意: 此种方法要求mapper接口名称和mapper映射文件名称相同, 且放在同一个目录中。

应用场景: 即可以使用XML配置文件, 也可以使用注解。

`<!-- 配置映射文件的位置 -->`

`<mappers>`

`<mapper resource="com/itheima/dao/UserDao.xml"></mapper>`

`<mapper class="com.itheima.dao.UserDao"/>`

`<package name="com.itheima.dao"></package>`

`</mappers>`

略，具体查看mybatis第二天



// 手动提交, 默认是false

```
sqlSession = sessionFactory.openSession();  
sqlSession.commit()
```

// 自动提交

```
sqlSession = sessionFactory.openSession(true);
```

<!--根据条件查询-->

```
<select id="findByCondition" parameterType="user" resultMap="userMap">
  select * from user where 1=1
  <if test="userName != null and userName.length>0">
    and username = #{userName}
  </if>
  <if test="userSex != null and userSex.length>0">
    and sex = #{userSex}
  </if>
</select>
```

<!--where标签的使用-->

```
<select id="findByCondition" parameterType="user" resultMap="userMap">
  select * from user
  <where>
    <if test="userName != null">
      and username = #{userName}
    </if>
    <if test="userSex != null">
      and sex = #{userSex}
    </if>
  </where>
</select>
```

<!--foreach 标签的使用方案一-->

```
<select id="findInIds" parameterType="queryVo" resultMap="userMap">
  select * from user
  <where>
    <if test="ids != null and ids.size()>0">
      <foreach collection="ids" open=" and (id=" close=")" item="uid" separator=" or id=">
        #{uid}
      </foreach>
    </if>
  </where>
</select>
```

<!--foreach 标签的使用方案二-->

```
<select id="findInIds" parameterType="queryVo" resultMap="userMap">
  select * from user
  <where>
    <if test="ids != null and ids.size()>0">
      <foreach collection="ids" open=" and id in (" close=")" item="uid" separator=",">
        #{uid}
      </foreach>
    </if>
  </where>
</select>
```

使用<sql>标签，定义出公共部分

```
<sql id="defaultUser">
```

```
  select * from user
```

```
</sql>
```

<!--foreach 标签的使用-->

```
<select id="findInIds" parameterType="queryVo" resultMap="userMap">
```

```
  <include refid="defaultUser"></include>
```

```
    <where>
```

```
      <if test="ids != null and ids.size()>0">
```

```
        <foreach collection="ids" open=" and id in (" close=")" item="uid"
```

```
separator=",">
```

```
          #{uid}
```

```
        </foreach>
```

```
      </if>
```

```
    </where>
```

```
</select>
```

第一步

```
public class Account implements Serializable {  
    private Integer id;  
    private Integer uid;  
    private Double money;  
    private User user;  
    public User getUser() {  
        return user;  
    }  
    public void setUser(User user) {  
        this.user = user;  
    }  
}
```

第二步

```
<resultMap id="accountMap" type="account">
  <id property="id" column="aid"></id>
  <result property="uid" column="uid"></result>
  <result property="money" column="money"></result>
  <association property="user" javaType="user">
    <id property="id" column="id"></id>
    <result property="username" column="username"></result>
    <result property="birthday" column="birthday"></result>
    <result property="sex" column="sex"></result>
    <result property="address" column="address"></result>
  </association>
</resultMap>
```

```
<select id="findByAccountUser2" resultMap="accountMap">
  select u.*,a.id as aid,a.uid,a.money from account a,user u where u.id = a.uid
</select>
```

第一步

```
public class User implements Serializable {
```

```
    private Integer id;  
    private String username;  
    private String address;  
    private String sex;  
    private Date birthday;
```

```
    private List<Account> accounts;
```

```
    public List<Account> getAccounts() {  
        return accounts;  
    }
```

```
    public void setAccounts(List<Account> accounts) {  
        this.accounts = accounts;  
    }
```

```
}
```


第二步

<!--定义用户和账号的查询-->

```
<resultMap id="userMap" type="user">
  <id property="id" column="id"></id>
  <result property="username" column="username"></result>
  <result property="address" column="address"></result>
  <result property="sex" column="sex"></result>
  <result property="birthday" column="birthday"></result>
  <collection property="accounts" ofType="account">
    <id property="id" column="aid"></id>
    <result property="uid" column="uid"></result>
    <result property="money" column="money"></result>
  </collection>
</resultMap>
```

<!-- 根据用户和账号的信息 -->

```
<select id="findUserAccountList" resultMap="userMap">
```

```
  SELECT u.*, a.id as aid, a.uid, a.money FROM user u LEFT JOIN account a ON u.id = a.uid
</select>
```

```
<resultMap id="roleMap" type="com.itheima.domain.Role">
  <id property="roleId" column="rid"></id>
  <result property="roleName" column="role_name"></result>
  <result property="roleDesc" column="role_desc"></result>
  <collection property="users" ofType="user">
    <id property="id" column="id"></id>
    <result property="username" column="username"></result>
    <result property="address" column="address"></result>
    <result property="sex" column="sex"></result>
    <result property="birthday" column="birthday"></result>
  </collection>
</resultMap>
<!-- 查询所有角色，同时查询所有用户-->
<select id="findRoleUserList" resultMap="roleMap">
  SELECT u.*,r.id AS rid,r.role_name,r.role_desc FROM role r
  LEFT JOIN user_role ur ON r.ID = ur.rid
  LEFT JOIN USER u ON u.id = ur.uid
</select>
```

```
<resultMap id="userMap" type="user">
  <id property="id" column="id"></id>
  <result property="username" column="username"></result>
  <result property="address" column="address"></result>
  <result property="birthday" column="birthday"></result>
  <result property="sex" column="sex"></result>
  <collection property="roles" ofType="role">
    <id property="roleId" column="rid"></id>
    <result property="roleName" column="role_name"></result>
    <result property="roleDesc" column="role_desc"></result>
  </collection>
</resultMap>
<!-- 查询所有的用户，同时查询关联的角色信息-->
<select id="findUserRoleList" resultMap="userMap">
  SELECT u.*,r.id AS rid,r.role_name,r.role_desc FROM USER u
  LEFT JOIN user_role ur ON u.ID = ur.uid
  LEFT JOIN role r ON r.id = ur.rid
</select>
```

```

<resultMap id="accountMap" type="account">
  <id property="id" column="id"></id>
  <result property="uid" column="uid"></result>
  <result property="money" column="money"></result>
  <!--association用来关联对象，
  property代表加载对象，
  javaType代表加载对象的数据类型，可以写成com.itheima.domain.User
  select 属性指定的内容：查询用户的唯一标识，指延迟加载要执行的statement的id
  要使用UserDao.xml中的findById完成根据用户id查询用户信息
  column 属性指定的内容：用户根据id查询时，所需要的参数的值
  -->
  <association property="user" column="uid" javaType="user" select="com.itheima.dao.UserDao.findById">
  </association>
</resultMap>

```

```

<!-- 查询所有账号 -->
<select id="findByAccountUser" resultMap="accountMap">
  select * from account;
</select>

```

```

<!-- 根据id查询用户 -->
<select id="findById" parameterType="INT"
  resultType="com.itheima.domain.User">
  select * from user where id = #{uid}
</select>

```

注意：（配置SqlMapConfig.xml）

①

```

<settings>
  <!--打开延迟加载的开关-->
  <setting name="lazyLoadingEnabled" value="true"/>
  <!--将立即加载改为按需加载，即延迟加载-->
  <setting name="aggressiveLazyLoading" value="false"/>
</settings>

```

②我们也可以在AccountDao.xml中使用fetchType属性用来控制延迟检索和立即检索

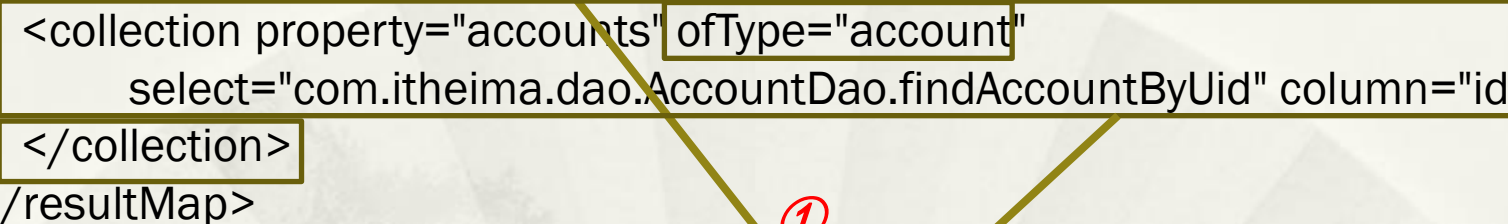
```

<association property="user" javaType="user"
  select="com.itheima.dao.UserDao.findById" column="uid"
  fetchType="eager">
</association>

```

<!--定义用户和账号的查询-->

```
<resultMap id="userMap" type="user">
  <id property="id" column="id"></id>
  <result property="username" column="username"></result>
  <result property="address" column="address"></result>
  <result property="sex" column="sex"></result>
  <result property="birthday" column="birthday"></result>
  <collection property="accounts" ofType="account"
    select="com.itheima.dao.AccountDao.findAccountByUid" column="id">
  </collection>
</resultMap>
```

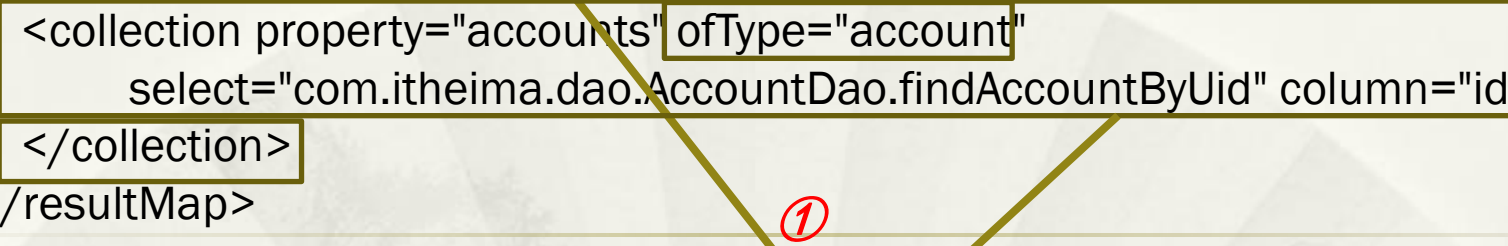


<!-- 根据用户和账号的信息 -->

```
<select id="findUserAccountList" resultMap="userMap">
  SELECT * FROM user
</select>
```

<!-- 更加用户id查询账号列表-->

```
<select id="findAccountByUid" resultType="account" parameterType="int">
  select * from account where uid = #{uid};
</select>
```



一级缓存:

它指的是Mybatis中SqlSession对象的缓存。

当我们执行查询之后，查询的结果会同时存入到SqlSession为我们提供一块区域中。该区域的结构是一个Map。

当我们再次查询同样的数据，mybatis会先去sqlsession中查询是否有，有的话直接拿出来用，如果没有再查询数据库。

当SqlSession对象消失时，mybatis的一级缓存也就消失了。

第一次发起查询用户id 为1 的用户信息，先去找缓存中是否有id 为1 的用户信息，如果没有，从数据库查询用户信息。得到用户信息，将用户信息存储到一级缓存中

如果sqlSession 去执行commit 操作（执行插入、更新、删除），清空SqlSession 中的一级缓存，

这样做的目的为了让缓存中存储的是最新的信息，避免脏读。

第二次发起查询用户id 为1 的用户信息，先去找缓存中是否有id 为1 的用户信息，缓存中有，直接从缓存中获取用户信息。

测试代码

```
public void testFirstLevelCacheCycle(){
    User user1 = userDao.findById(50);
    System.out.println(user1);
    sqlSession.close();
    sqlSession = factory.openSession();
    userDao = sqlSession.getMapper(UserDao.class);
    User user2 = userDao.findById(50);
    System.out.println(user2);
    System.out.println(user1==user2);
}
```


第一步：在SqlMapConfig.xml 文件开启二级缓存。

因为cacheEnabled 的取值默认就为true，
所以这一步可以省略不配置。
为true 代表开启二级缓存； 为false 代表不开启二级缓存。

```
<!--需要配置延迟加载策略-->
<settings>
  <!--开启二级缓存-->
  <setting name="cacheEnabled" value="true"/>
</settings>
```

第二步：配置UserDao.xml中相关的Mapper 映射文件

<cache>标签表示当前这个mapper 映射将使用二级缓存，
区分的标准就看mapper 的namespace 值。

```
<mapper namespace="com.itheima.dao.UserDao">
  <cache/>
</mapper>
```

第三步：配置UserDao.xml中statement 上面的useCache 属性，在select 标签中配置

```
<!-- 根据id查询用户 -->
<select id="findById" parameterType="INT" resultType="com.itheima.domain.User"
  useCache="true">
  select * from user where id = #{uid}
</select>
```

将UserDao.xml 映射文件中的
<select> 标签中设置useCache="true"
代表当前这个statement 要使用二级缓存，
如果不使用二级缓存可以设置为false。
注意：针对每次查询都需要最新的数据sql，
要设置成useCache=false，禁用二级缓存。

测试代码

```

public void testSecondLevelCache(){
    SqlSession sqlSession1 = factory.openSession();
    UserDao userDao = sqlSession1.getMapper(UserDao.class);
    User user1 = userDao.findById(50);
    System.out.println(user1);
    sqlSession1.close();
    SqlSession sqlSession2 = factory.openSession();
    UserDao userDao2 = sqlSession2.getMapper(UserDao.class);
    User user2 = userDao2.findById(50);
    System.out.println(user2);
    sqlSession2.close();
    System.out.println(user1==user2);
}

```

它指的是Mybatis 中SqlSessionFactory对象的级别缓存。
由同一个SqlSessionFactory对象创建的SqlSession共享其缓存。
把散装数据封装成对象实现的共享

```

2018-09-06 16:39:46,390 987 [main] DEBUG ansaction.jdbc.JdbcTransaction - Setting autocommit to false on JDBC Connection
2018-09-06 16:39:46,393 990 [main] DEBUG m.itheima.dao.UserDao.findById - ==> Preparing: select * from user where id = ?
2018-09-06 16:39:46,429 1026 [main] DEBUG m.itheima.dao.UserDao.findById - ==> Parameters: 41(Integer)
2018-09-06 16:39:46,459 1056 [main] DEBUG m.itheima.dao.UserDao.findById - <== Total: 1
com.itheima.domain.User@430e6c
2018-09-06 16:39:46,472 1069 [main] DEBUG ansaction.jdbc.JdbcTransaction - Resetting autocommit to true on JDBC Connection
2018-09-06 16:39:46,473 1070 [main] DEBUG ansaction.jdbc.JdbcTransaction - Closing JDBC Connection [com.mysql.jdbc.JDBC4Con
2018-09-06 16:39:46,473 1070 [main] DEBUG source.pooled.PooledDataSource - Returned connection 2562358 to pool.
2018-09-06 16:39:46,543 1140 [main] DEBUG com.itheima.dao.UserDao - Cache Hit Ratio [com.itheima.dao.UserDao]: 0.5
com.itheima.domain.User@3f91c3
false

```

```
<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
  <property name="dataSource" ref="dataSource"></property>
</bean>
```

<!--2: 配置通知（事务管理器的方法），同时对切入点的方法进行细化-->

```
<tx:advice id="myAdvice" transaction-manager="transactionManager">
  <tx:attributes>
    <tx:method name="save*" read-only="false" isolation="DEFAULT" propagation="REQUIRED"/>
    <tx:method name="insert*" read-only="false" isolation="DEFAULT" propagation="REQUIRED"/>
    <tx:method name="update*" read-only="false" isolation="DEFAULT" propagation="REQUIRED"/>
    <tx:method name="edit*" read-only="false" isolation="DEFAULT" propagation="REQUIRED"/>
    <tx:method name="delete*" read-only="false" isolation="DEFAULT" propagation="REQUIRED"/>
    <tx:method name="*" read-only="true"/>
  </tx:attributes>
</tx:advice>
```

<!--3: 配置aop，让通知（事务）关联切入点（Service）-->

```
<aop:config>
  <aop:pointcut id="myPointcut" expression="execution(* com.itheima.ssm.service..*.*(..))"></aop:pointcut>
  <aop:advisor advice-ref="myAdvice" pointcut-ref="myPointcut"></aop:advisor>
</aop:config>
```