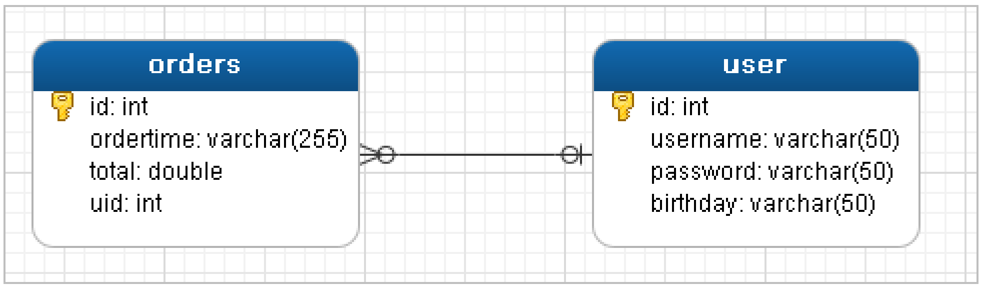
1.Mybatis多表查询

1.1 一对一查询

1.1.1 一对一查询的模型MapperScannerConfigurer

用户表和订单表的关系为，一个用户有多个订单，一个订单只从属于一个用户

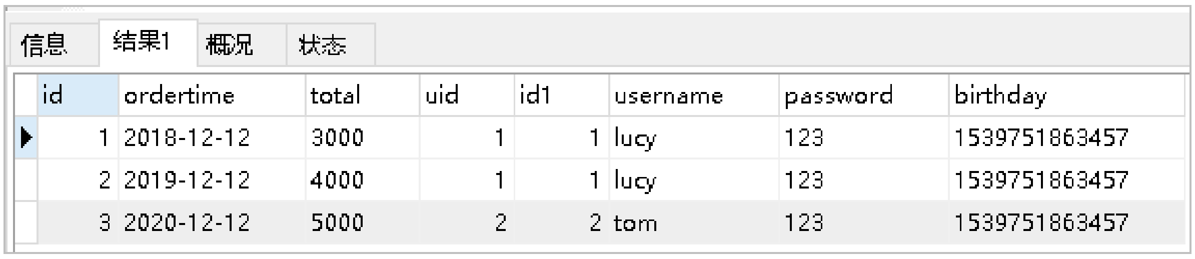
一对一查询的需求：查询一个订单，与此同时查询出该订单所属的用户



1.1.2一对一查询的语句

对应的sql语句：select * from orders o,user u where o.uid=u.id;

查询的结果如下：



id	ordertime	total	uid	id1	username	password	birthday
1	2018-12-12	3000	1	1	lucy	123	1539751863457
2	2019-12-12	4000	1	1	lucy	123	1539751863457
3	2020-12-12	5000	2	2	tom	123	1539751863457

1.1.3 创建Order和User实体

```java
public class Order {

    private int id;
    private Date ordertime;
    private double total;

    //代表当前订单从属于哪一个客户
    private User user;
}

public class User {

    private int id;
    private String username;
    private String password;
    private Date birthday;

}
```

1.1.4 创建OrderMapper接口

```java
public interface OrderMapper {
    List<Order> findAll();
}
```

1.1.5 配置OrderMapper.xml

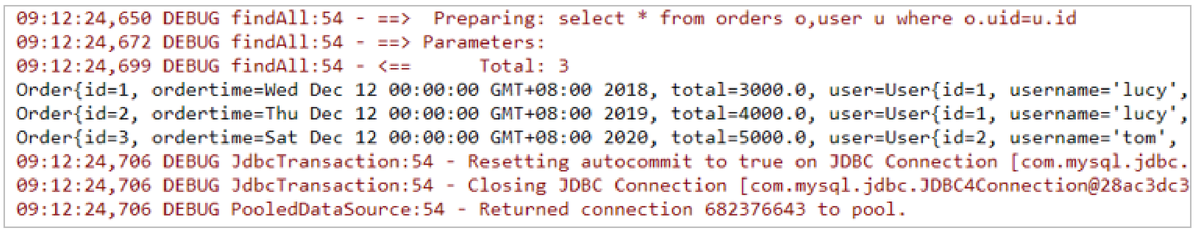
```xml
<mapper namespace="com.itheima.mapper.OrderMapper">
    <resultMap id="orderMap" type="com.itheima.domain.Order">
        <result column="uid" property="user.id"></result>
        <result column="username" property="user.username"></result>
        <result column="password" property="user.password"></result>
        <result column="birthday" property="user.birthday"></result>
    </resultMap>
    <select id="findAll" resultMap="orderMap">
        select * from orders o,user u where o.uid=u.id
    </select>
</mapper>
```

其中还可以配置如下：

```xml
<resultMap id="orderMap" type="com.itheima.domain.Order">
    <result property="id" column="id"></result>
    <result property="ordertime" column="ordertime"></result>
    <result property="total" column="total"></result>
    <association property="user" javaType="com.itheima.domain.User">
        <result column="uid" property="id"></result>
        <result column="username" property="username"></result>
        <result column="password" property="password"></result>
        <result column="birthday" property="birthday"></result>
    </association>
</resultMap>
```

1.1.6 测试结果

```java
OrderMapper mapper = sqlSession.getMapper(OrderMapper.class);
List<Order> all = mapper.findAll();
for(Order order : all){
    System.out.println(order);
}
```



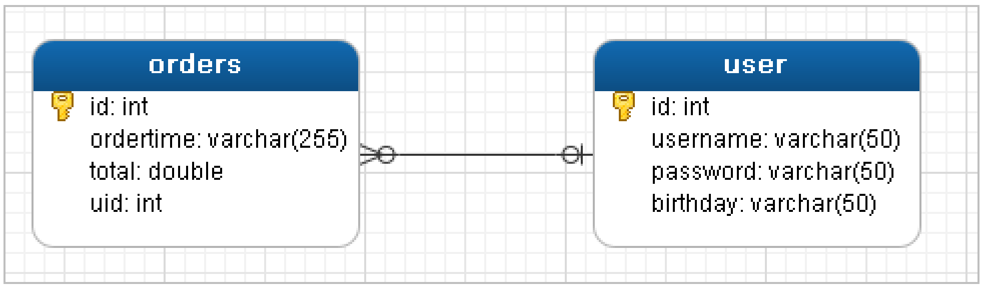
```
09:12:24,650 DEBUG findAll:54 - ==>  Preparing: select * from orders o,user u where o.uid=u.id
09:12:24,672 DEBUG findAll:54 - ==> Parameters: 
09:12:24,699 DEBUG findAll:54 - <==      Total: 3
Order{id=1, ordertime=Wed Dec 12 00:00:00 GMT+08:00 2018, total=3000.0, user=User{id=1, username='lucy',
Order{id=2, ordertime=Thu Dec 12 00:00:00 GMT+08:00 2019, total=4000.0, user=User{id=1, username='lucy',
Order{id=3, ordertime=Sat Dec 12 00:00:00 GMT+08:00 2020, total=5000.0, user=User{id=2, username='tom',
09:12:24,706 DEBUG JdbcTransaction:54 - Resetting autocommit to true on JDBC Connection [com.mysql.jdbc.
09:12:24,706 DEBUG JdbcTransaction:54 - Closing JDBC Connection [com.mysql.jdbc.JDBC4Connection@28ac3dc3
09:12:24,706 DEBUG PooledDataSource:54 - Returned connection 682376643 to pool.
```

1.2 一对多查询

1.2.1 一对多查询的模型

用户表和订单表的关系为，一个用户有多个订单，一个订单只从属于一个用户

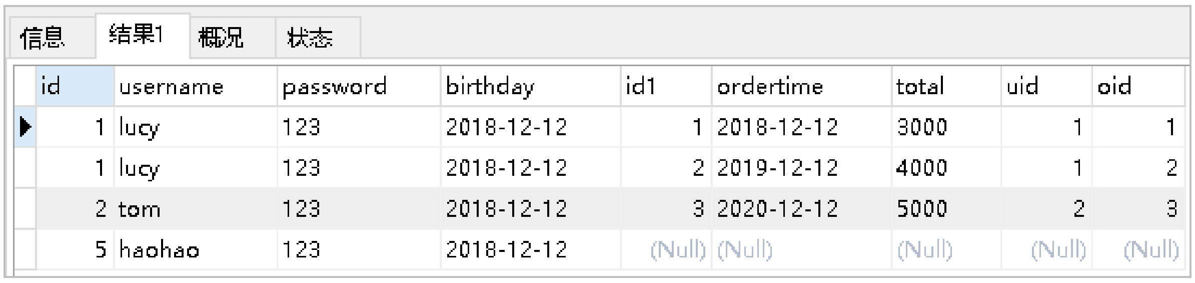
一对多查询的需求：查询一个用户，与此同时查询出该用户具有的订单



1.2.2 一对多查询的语句

对应的sql语句：select *,o.id oid from user u left join orders o on u.id=o.uid;

查询的结果如下：



id	username	password	birthday	id1	ordertime	total	uid	oid
1	lucy	123	2018-12-12	1	2018-12-12	3000	1	1
1	lucy	123	2018-12-12	2	2019-12-12	4000	1	2
2	tom	123	2018-12-12	3	2020-12-12	5000	2	3
5	haohao	123	2018-12-12	(Null)	(Null)	(Null)	(Null)	(Null)

1.2.3 修改User实体

```java
public class Order {

    private int id;
    private Date ordertime;
    private double total;

    //代表当前订单从属于哪一个客户
    private User user;
}

public class User {

    private int id;
    private String username;
    private String password;
    private Date birthday;
    //代表当前用户具备哪些订单
    private List<Order> orderList;
}
```

1.2.4 创建UserMapper接口

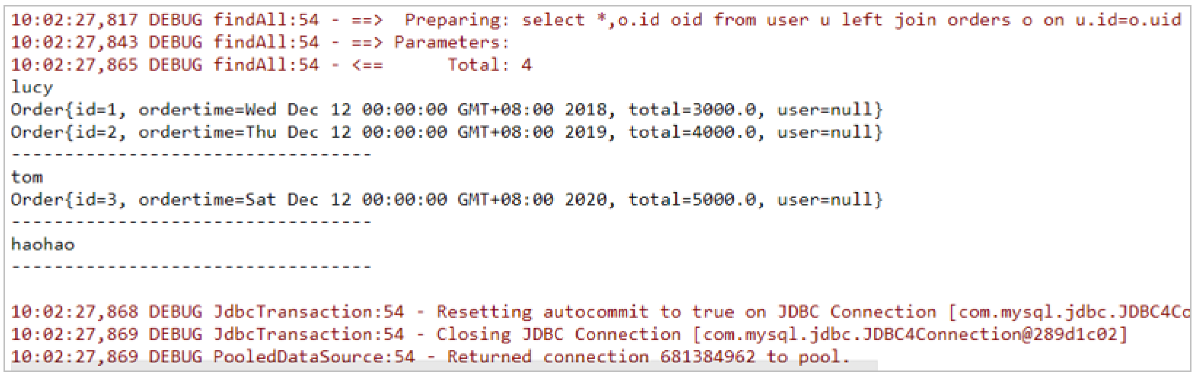
```java
public interface UserMapper {
    List<User> findAll();
}
```

1.2.5 配置UserMapper.xml

```xml
<mapper namespace="com.itheima.mapper.UserMapper">
    <resultMap id="userMap" type="com.itheima.domain.User">
        <result column="id" property="id"></result>
        <result column="username" property="username"></result>
        <result column="password" property="password"></result>
        <result column="birthday" property="birthday"></result>
        <collection property="orderList" ofType="com.itheima.domain.Order">
            <result column="oid" property="id"></result>
            <result column="ordertime" property="ordertime"></result>
            <result column="total" property="total"></result>
        </collection>
    </resultMap>
    <select id="findAll" resultMap="userMap">
        select *,o.id oid from user u left join orders o on u.id=o.uid
    </select>
</mapper>
```

1.2.6 测试结果

```java
UserMapper mapper = sqlSession.getMapper(UserMapper.class);
List<User> all = mapper.findAll();
for(User user : all){
    System.out.println(user.getUsername());
    List<Order> orderList = user.getOrderList();
    for(Order order : orderList){
        System.out.println(order);
    }
    System.out.println("----------------------------------");
}
```



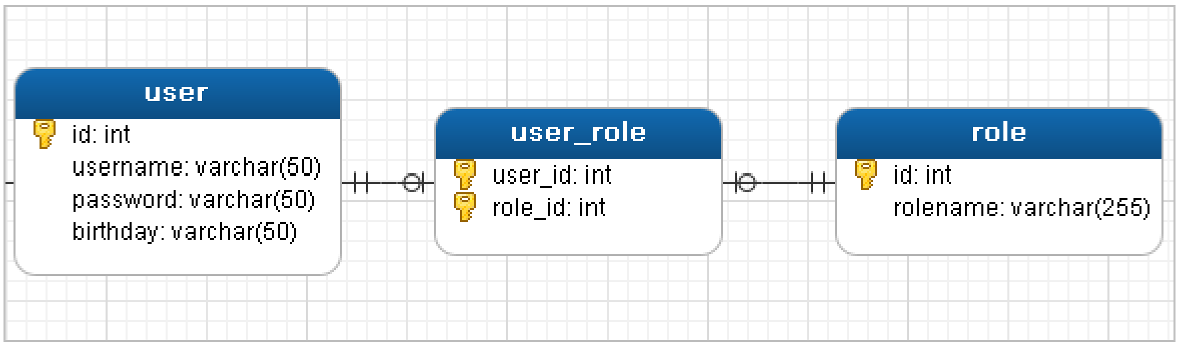
```
10:02:27,817 DEBUG findAll:54 - ==>  Preparing: select *,o.id oid from user u left join orders o on u.id=o.uid
10:02:27,843 DEBUG findAll:54 - ==> Parameters: 
10:02:27,865 DEBUG findAll:54 - <==      Total: 4
lucy
Order{id=1, ordertime=Wed Dec 12 00:00:00 GMT+08:00 2018, total=3000.0, user=null}
Order{id=2, ordertime=Thu Dec 12 00:00:00 GMT+08:00 2019, total=4000.0, user=null}
----------------------------------
tom
Order{id=3, ordertime=Sat Dec 12 00:00:00 GMT+08:00 2020, total=5000.0, user=null}
----------------------------------
haohao
----------------------------------

10:02:27,868 DEBUG JdbcTransaction:54 - Resetting autocommit to true on JDBC Connection [com.mysql.jdbc.JDBC4Co
10:02:27,869 DEBUG JdbcTransaction:54 - Closing JDBC Connection [com.mysql.jdbc.JDBC4Connection@289d1c02]
10:02:27,869 DEBUG PooledDataSource:54 - Returned connection 681384962 to pool.
```

1.3 多对多查询

1.3.1 多对多查询的模型

用户表和角色表的关系为，一个用户有多个角色，一个角色被多个用户使用

多对多查询的需求：查询用户同时查询出该用户的所有角色

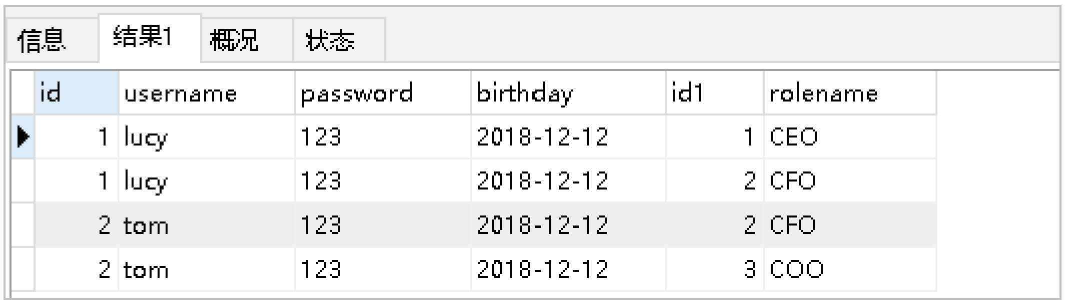


1.3.2 多对多查询的语句

对应的sql语句：select u.,r.,r.id rid from user u left join user_role ur on u.id=ur.user_id

inner join role r on ur.role_id=r.id;

查询的结果如下：



id	username	password	birthday	id1	rolename
1	lucy	123	2018-12-12	1	CEO
1	lucy	123	2018-12-12	2	CFO
2	tom	123	2018-12-12	2	CFO
2	tom	123	2018-12-12	3	COO

1.3.3 创建Role实体，修改User实体

```
public class User {
    private int id;
    private String username;
    private String password;
    private Date birthday;
    //代表当前用户具备哪些订单
    private List<Order> orderList;
    //代表当前用户具备哪些角色
    private List<Role> roleList;
}

public class Role {

    private int id;
    private String rolename;

}
```

1.3.4 添加UserMapper接口方法

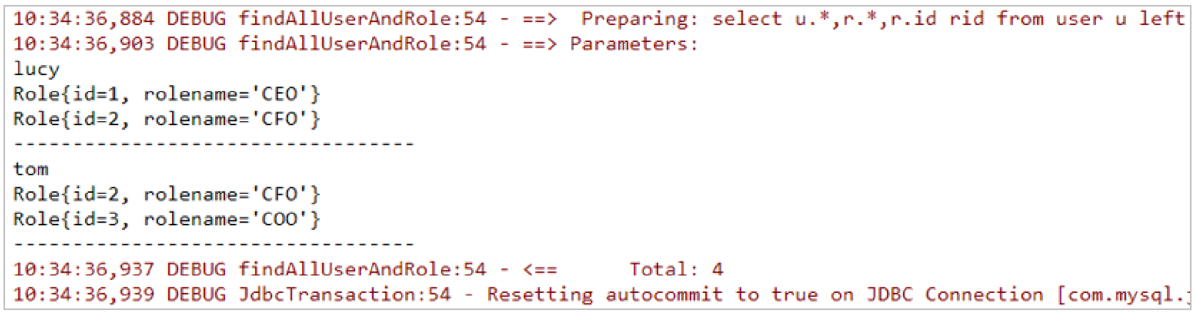
```java
List<User> findAllUserAndRole();
```

1.3.5 配置UserMapper.xml

```xml
<resultMap id="userRoleMap" type="com.itheima.domain.User">
    <result column="id" property="id"></result>
    <result column="username" property="username"></result>
    <result column="password" property="password"></result>
    <result column="birthday" property="birthday"></result>
    <collection property="roleList" ofType="com.itheima.domain.Role">
        <result column="rid" property="id"></result>
        <result column="rolename" property="rolename"></result>
    </collection>
</resultMap>
<select id="findAllUserAndRole" resultMap="userRoleMap">
    select u.*,r.*,r.id rid from user u left join user_role ur on
u.id=ur.user_id
    inner join role r on ur.role_id=r.id
</select>
```

1.3.6 测试结果

```java
UserMapper mapper = sqlSession.getMapper(UserMapper.class);
List<User> all = mapper.findAllUserAndRole();
for(User user : all){
    System.out.println(user.getUsername());
    List<Role> roleList = user.getRoleList();
    for(Role role : roleList){
        System.out.println(role);
    }
    System.out.println("----------------------------------");
}
```



```
10:34:36,884 DEBUG findAllUserAndRole:54 - ==>  Preparing: select u.*,r.*,r.id rid from user u left
10:34:36,903 DEBUG findAllUserAndRole:54 - ==> Parameters: 
lucy
Role{id=1, rolename='CEO'}
Role{id=2, rolename='CFO'}
----------------------------------
tom
Role{id=2, rolename='CFO'}
Role{id=3, rolename='COO'}
----------------------------------
10:34:36,937 DEBUG findAllUserAndRole:54 - <==      Total: 4
10:34:36,939 DEBUG JdbcTransaction:54 - Resetting autocommit to true on JDBC Connection [com.mysql.
```

1.4 知识小结

MyBatis多表配置方式：

一对一配置：使用做配置

一对多配置：使用+做配置

多对多配置：使用+做配置

2.Mybatis的注解开发

2.1 MyBatis的常用注解

这几年来注解开发越来越流行，Mybatis也可以使用注解开发方式，这样我们就可以减少编写Mapper映射文件了。我们先围绕一些基本的CRUD来学习，再学习复杂映射多表操作。

@Insert：实现新增

@Update：实现更新

@Delete：实现删除

@Select：实现查询

@Result：实现结果集封装

@Results：可以与@Result 一起使用，封装多个结果集

@One：实现一对一结果集封装

@Many：实现一对多结果集封装

2.2 MyBatis的增删改查

我们完成简单的user表的增删改查的操作

```java
private UserMapper userMapper;

@Before
public void before() throws IOException {
    InputStream resourceAsStream = 
Resources.getResourceAsStream("SqlMapConfig.xml");
    SqlSessionFactory sqlSessionFactory = new 
                 SqlSessionFactoryBuilder().build(resourceAsStream);
    SqlSession sqlSession = sqlSessionFactory.openSession(true);
    userMapper = sqlSession.getMapper(UserMapper.class);
}

@Test
public void testAdd() {
    User user = new User();
    user.setUsername("测试数据");
    user.setPassword("123");
    user.setBirthday(new Date());
    userMapper.add(user);
}
@Test
public void testUpdate() throws IOException {
    User user = new User();
    user.setId(16);
    user.setUsername("测试数据修改");
    user.setPassword("abc");
    user.setBirthday(new Date());
    userMapper.update(user);
}
```

```
@Test
public void testDelete() throws IOException {
    userMapper.delete(16);
}
@Test
public void testFindById() throws IOException {
    User user = userMapper.findById(1);
    System.out.println(user);
}
@Test
public void testFindAll() throws IOException {
    List<User> all = userMapper.findAll();
    for(User user : all){
        System.out.println(user);
    }
}
```

修改MyBatis的核心配置文件，我们使用了注解替代的映射文件，所以我们只需要加载使用了注解的Mapper接口即可

```
<mappers>
    <!--扫描使用注解的类-->
    <mapper class="com.itheima.mapper.UserMapper"></mapper>
</mappers>
```

或者指定扫描包含映射关系的接口所在的包也可以

```
<mappers>
    <!--扫描使用注解的类所在的包-->
    <package name="com.itheima.mapper"></package>
</mappers>
```

2.3 MyBatis的注解实现复杂映射开发

实现复杂关系映射之前我们可以在映射文件中通过配置来实现，使用注解开发后，我们可以使用@Results注解，@Result注解，@One注解，@Many注解组合完成复杂关系的配置

注解	说明
@Results	代替的是标签<resultMap>该注解中可以使用单个@Result注解，也可以使用@Result集合。使用格式：@Results（{@Result（），@Result（）}）或@Results（@Result（））
@Resut	代替了<id>标签和<result>标签 @Result中属性介绍： column：数据库的列名 property：需要装配的属性名 one：需要使用的@One 注解（@Result（one=@One）（））) many：需要使用的@Many 注解（@Result（many=@many）（））)

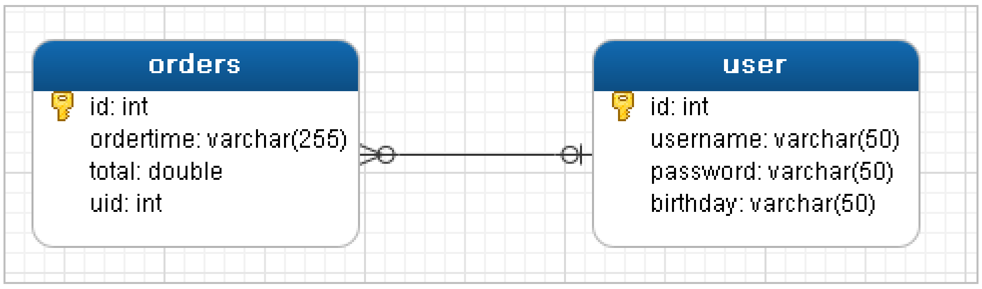
注解	说明
@One （一对一）	代替了<assocation> 标签，是多表查询的关键，在注解中用来指定子查询返回单一对象。 @One注解属性介绍： select: 指定用来多表查询的 sqlmapper 使用格式：@Result(column=" ",property="",one=@One(select=""))
@Many （多对一）	代替了<collection>标签, 是是多表查询的关键，在注解中用来指定子查询返回对象集合。 使用格式：@Result(property="",column="",many=@Many(select=""))

2.4 一对一查询

2.4.1 一对一查询的模型

用户表和订单表的关系为，一个用户有多个订单，一个订单只从属于一个用户

一对一查询的需求：查询一个订单，与此同时查询出该订单所属的用户

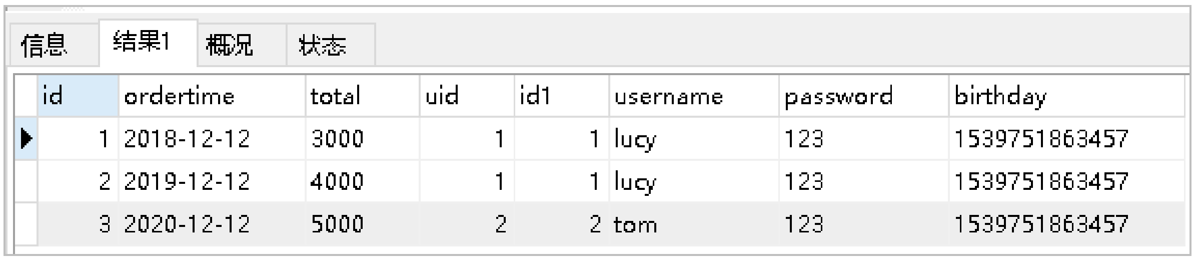


2.4.2 一对一查询的语句

对应的sql语句：

```
select * from orders;

select * from user where id=查询出订单的uid;
```

查询的结果如下：



id	ordertime	total	uid	id1	username	password	birthday
1	2018-12-12	3000	1	1	lucy	123	1539751863457
2	2019-12-12	4000	1	1	lucy	123	1539751863457
3	2020-12-12	5000	2	2	tom	123	1539751863457

2.4.3 创建Order和User实体

```
public class Order {

    private int id;
    private Date ordertime;
    private double total;

    //代表当前订单从属于哪一个客户
    private User user;
}

public class User {
```

```java
    private int id;
    private String username;
    private String password;
    private Date birthday;

}
```

2.4.4 创建OrderMapper接口

```java
public interface OrderMapper {
    List<Order> findAll();
}
```

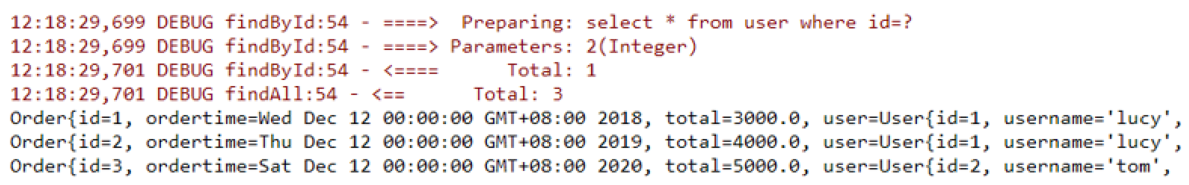
2.4.5 使用注解配置Mapper

```java
public interface OrderMapper {
    @Select("select * from orders")
    @Results({
            @Result(id=true,property = "id",column = "id"),
            @Result(property = "ordertime",column = "ordertime"),
            @Result(property = "total",column = "total"),
            @Result(property = "user",column = "uid",
                    javaType = User.class,
                    one = @One(select =
"com.itheima.mapper.UserMapper.findById"))
    })
    List<Order> findAll();
}
```

```java
public interface UserMapper {

    @Select("select * from user where id=#{id}")
    User findById(int id);

}
```

2.4.6 测试结果

```java
@Test
public void testSelectOrderAndUser() {
    List<Order> all = orderMapper.findAll();
    for(Order order : all){
        System.out.println(order);
    }
}
```



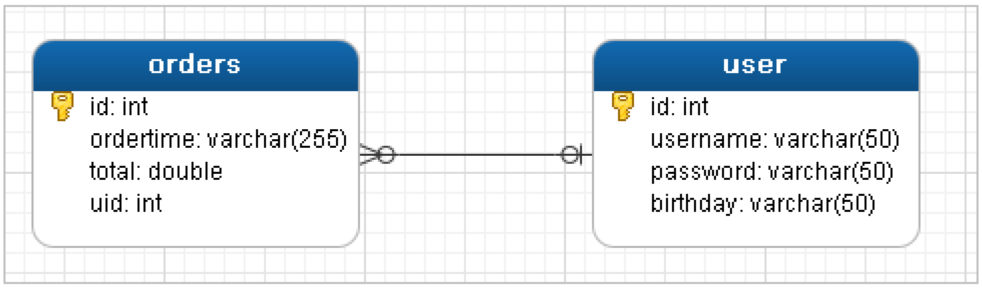
```
12:18:29,699 DEBUG findById:54 - ====>  Preparing: select * from user where id=?
12:18:29,699 DEBUG findById:54 - ====> Parameters: 2(Integer)
12:18:29,701 DEBUG findById:54 - <====      Total: 1
12:18:29,701 DEBUG findAll:54 - <==      Total: 3
Order{id=1, ordertime=Wed Dec 12 00:00:00 GMT+08:00 2018, total=3000.0, user=User{id=1, username='lucy',
Order{id=2, ordertime=Thu Dec 12 00:00:00 GMT+08:00 2019, total=4000.0, user=User{id=1, username='lucy',
Order{id=3, ordertime=Sat Dec 12 00:00:00 GMT+08:00 2020, total=5000.0, user=User{id=2, username='tom',
```

2.5 一对多查询

2.5.1 一对多查询的模型

用户表和订单表的关系为，一个用户有多个订单，一个订单只从属于一个用户

一对多查询的需求：查询一个用户，与此同时查询出该用户具有的订单

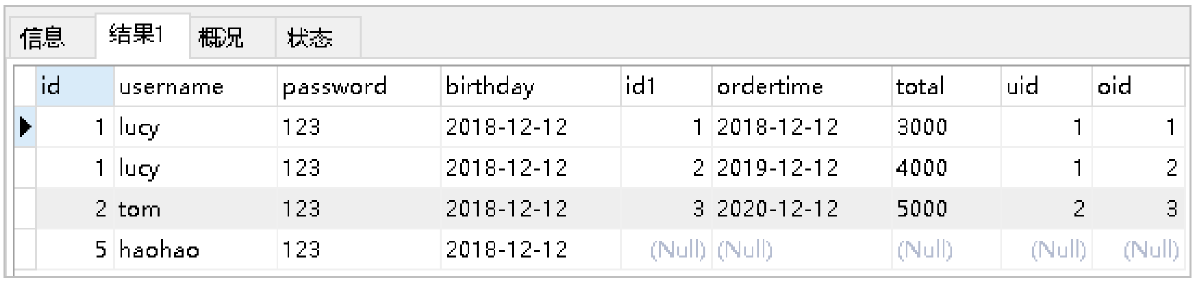


2.5.2 一对多查询的语句

对应的sql语句：

```
select * from user;

select * from orders where uid=查询出用户的id;
```

查询的结果如下：



id	username	password	birthday	id1	ordertime	total	uid	oid
1	lucy	123	2018-12-12	1	2018-12-12	3000	1	1
1	lucy	123	2018-12-12	2	2019-12-12	4000	1	2
2	tom	123	2018-12-12	3	2020-12-12	5000	2	3
5	haohao	123	2018-12-12	(Null)	(Null)	(Null)	(Null)	(Null)

2.5.3 修改User实体

```
public class Order {

    private int id;
    private Date ordertime;
    private double total;

    //代表当前订单从属于哪一个客户
    private User user;
}

public class User {

    private int id;
    private String username;
    private String password;
    private Date birthday;
    //代表当前用户具备哪些订单
    private List<Order> orderList;
}
```

2.5.4 创建UserMapper接口

```
List<User> findAllUserAndOrder();
```

2.5.5 使用注解配置Mapper

```
public interface UserMapper {
    @Select("select * from user")
    @Results({
            @Result(id = true,property = "id",column = "id"),
            @Result(property = "username",column = "username"),
            @Result(property = "password",column = "password"),
            @Result(property = "birthday",column = "birthday"),
            @Result(property = "orderList",column = "id",
                    javaType = List.class,
                    many = @Many(select =
"com.itheima.mapper.OrderMapper.findByUid"))
    })
    List<User> findAllUserAndOrder();
}

public interface OrderMapper {
    @Select("select * from orders where uid=#{uid}")
    List<Order> findByUid(int uid);

}
```

2.5.6 测试结果

```
List<User> all = userMapper.findAllUserAndOrder();
for(User user : all){
    System.out.println(user.getUsername());
    List<Order> orderList = user.getOrderList();
    for(Order order : orderList){
        System.out.println(order);
    }
    System.out.println("----------------------------");
}
```

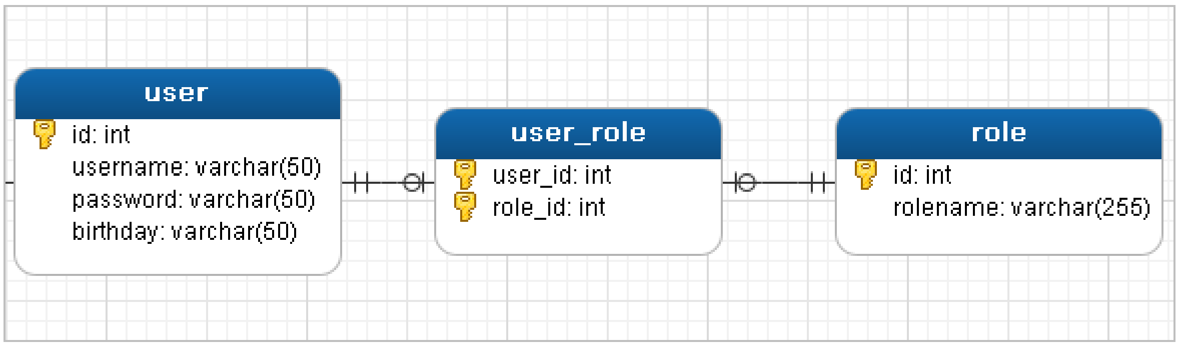
```
14:32:14,813 DEBUG findAllUserAndOrder:54 - ==>  Preparing: select * from user
14:32:14,844 DEBUG findAllUserAndOrder:54 - ==> Parameters:
14:32:14,860 DEBUG findByUid:54 - ====>  Preparing: select * from orders where uid=?
lucy
Order{id=1, ordertime=Wed Dec 12 00:00:00 GMT+08:00 2018, total=3000.0, user=null}
Order{id=2, ordertime=Thu Dec 12 00:00:00 GMT+08:00 2019, total=4000.0, user=null}
-----------------------------
tom
Order{id=3, ordertime=Sat Dec 12 00:00:00 GMT+08:00 2020, total=5000.0, user=null}
-----------------------------
haohao
-----------------------------
```

2.6 多对多查询

2.6.1 多对多查询的模型

用户表和角色表的关系为，一个用户有多个角色，一个角色被多个用户使用

多对多查询的需求：查询用户同时查询出该用户的所有角色

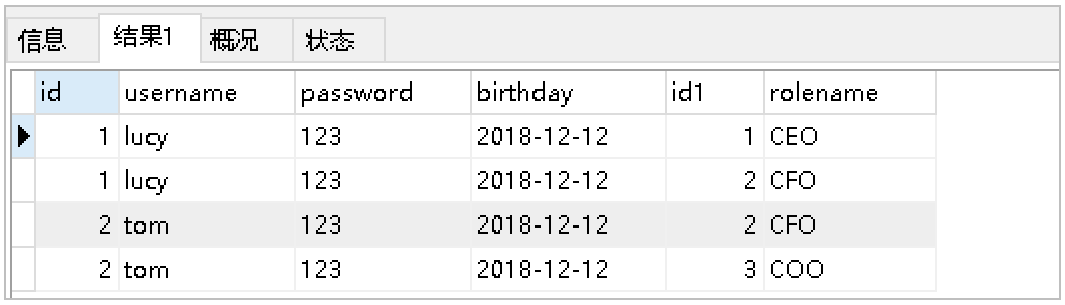


2.6.2 多对多查询的语句

对应的sql语句：

```
select * from user;

select * from role r,user_role ur where r.id=ur.role_id and ur.user_id=用户的id
```

查询的结果如下：



id	username	password	birthday	id1	rolename
1	lucy	123	2018-12-12	1	CEO
1	lucy	123	2018-12-12	2	CFO
2	tom	123	2018-12-12	2	CFO
2	tom	123	2018-12-12	3	COO

2.6.3 创建Role实体，修改User实体

```java
public class User {
    private int id;
    private String username;
    private String password;
    private Date birthday;
    //代表当前用户具备哪些订单
    private List<Order> orderList;
    //代表当前用户具备哪些角色
    private List<Role> roleList;
}

public class Role {

    private int id;
    private String rolename;

}
```

2.6.4 添加UserMapper接口方法

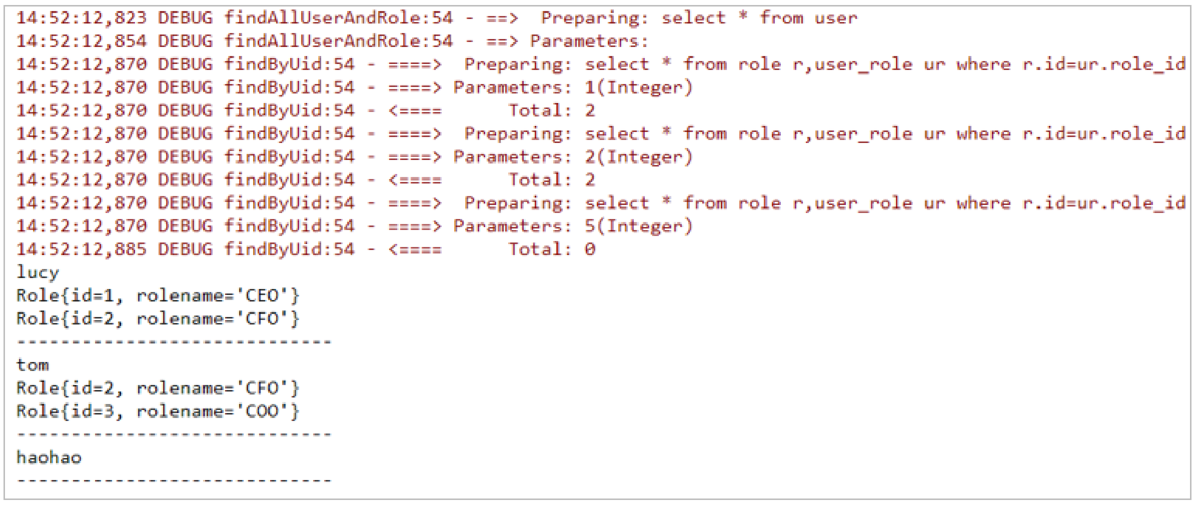
```
List<User> findAllUserAndRole();
```

2.6.5 使用注解配置Mapper

```
public interface UserMapper {
    @Select("select * from user")
    @Results({
        @Result(id = true,property = "id",column = "id"),
        @Result(property = "username",column = "username"),
        @Result(property = "password",column = "password"),
        @Result(property = "birthday",column = "birthday"),
        @Result(property = "roleList",column = "id",
                javaType = List.class,
                many = @Many(select =
"com.itheima.mapper.RoleMapper.findByUid"))
})
List<User> findAllUserAndRole();}



public interface RoleMapper {
    @Select("select * from role r,user_role ur where r.id=ur.role_id and
ur.user_id=#{uid}")
    List<Role> findByUid(int uid);
}
```

2.6.6 测试结果

```
UserMapper mapper = sqlSession.getMapper(UserMapper.class);
List<User> all = mapper.findAllUserAndRole();
for(User user : all){
    System.out.println(user.getUsername());
    List<Role> roleList = user.getRoleList();
    for(Role role : roleList){
        System.out.println(role);
    }
    System.out.println("----------------------------------");
}
```



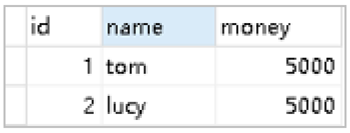
```
14:52:12,823 DEBUG findAllUserAndRole:54 - ==>  Preparing: select * from user
14:52:12,854 DEBUG findAllUserAndRole:54 - ==> Parameters: 
14:52:12,870 DEBUG findByUid:54 - ====>  Preparing: select * from role r,user_role ur where r.id=ur.role_id
14:52:12,870 DEBUG findByUid:54 - ====> Parameters: 1(Integer)
14:52:12,870 DEBUG findByUid:54 - <====      Total: 2
14:52:12,870 DEBUG findByUid:54 - ====>  Preparing: select * from role r,user_role ur where r.id=ur.role_id
14:52:12,870 DEBUG findByUid:54 - ====> Parameters: 2(Integer)
14:52:12,870 DEBUG findByUid:54 - <====      Total: 2
14:52:12,870 DEBUG findByUid:54 - ====>  Preparing: select * from role r,user_role ur where r.id=ur.role_id
14:52:12,870 DEBUG findByUid:54 - ====> Parameters: 5(Integer)
14:52:12,885 DEBUG findByUid:54 - <====      Total: 0
lucy
Role{id=1, rolename='CEO'}
Role{id=2, rolename='CFO'}
----------------------------
tom
Role{id=2, rolename='CFO'}
Role{id=3, rolename='COO'}
----------------------------
haohao
----------------------------
```

SSM框架整合

1.1 原始方式整合

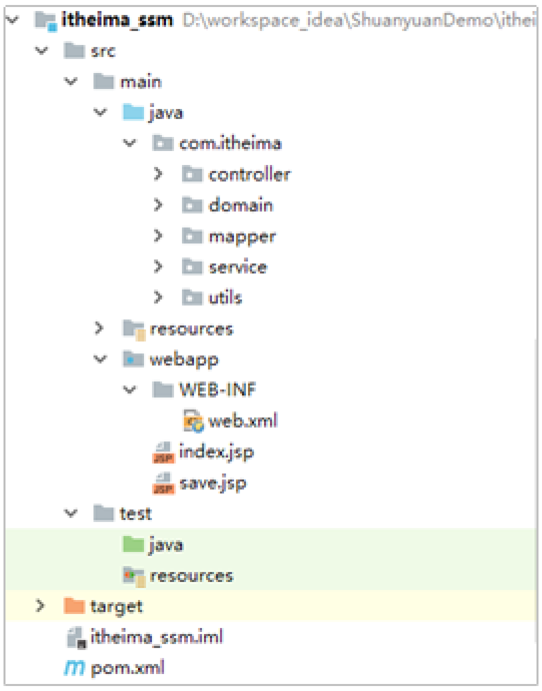
1.准备工作

```
create database ssm;
create table account(
    id int primary key auto_increment,
    name varchar(100),
    money double(7,2)
);
```



id	name	money
1	tom	5000
2	lucy	5000

2.创建Maven工程



```
itheima_ssm D:\workspace_idea\ShuanyuanDemo\ithei
  src
    main
      java
        com.itheima
          controller
          domain
          mapper
          service
          utils
      resources
      webapp
        WEB-INF
          web.xml
        index.jsp
        save.jsp
    test
      java
      resources
  target
  itheima_ssm.iml
  pom.xml
```

3.导入Maven坐标

参考：**素材/配置文件/pom.xml文件**

4.编写实体类

```java
public class Account {
    private int id;
    private String name;
    private double money;
    //省略getter和setter方法
}
```

5.编写Mapper接口

```java
public interface AccountMapper {
    //保存账户数据
    void save(Account account);
    //查询账户数据
    List<Account> findAll();
}
```

6.编写Service接口

```java
public interface AccountService {
    void save(Account account); //保存账户数据
    List<Account> findAll(); //查询账户数据
}
```

7.编写Service接口实现

```java
@Service("accountService")
public class AccountServiceImpl implements AccountService {
    public void save(Account account) {
        SqlSession sqlSession = MyBatisUtils.openSession();
        AccountMapper accountMapper = sqlSession.getMapper(AccountMapper.class);
        accountMapper.save(account);
        sqlSession.commit();
        sqlSession.close();
    }
    public List<Account> findAll() {
        SqlSession sqlSession = MyBatisUtils.openSession();
        AccountMapper accountMapper = sqlSession.getMapper(AccountMapper.class);
        return accountMapper.findAll();
    }
}
```

8.编写Controller

```
@Controller
public class AccountController {
    @Autowired
    private AccountService accountService;
    @RequestMapping("/save")
    @ResponseBody
    public String save(Account account){
        accountService.save(account);
        return "save success";
    }
    @RequestMapping("/findAll")
    public ModelAndView findAll(){
        ModelAndView modelAndView = new ModelAndView();
        modelAndView.setViewName("accountList");
        modelAndView.addObject("accountList",accountService.findAll());
        return modelAndView;
    }
}
```

9.编写添加页面

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
    <h1>保存账户信息表单</h1>
    <form action="${pageContext.request.contextPath}/save.action" method="post">
        用户名称<input type="text" name="name"><br/>
        账户金额<input type="text" name="money"><br/>
        <input type="submit" value="保存"><br/>
    </form>
</body>
</html>
```

10.编写列表页面

```
<table border="1">
    <tr>
        <th>账户id</th>
        <th>账户名称</th>
        <th>账户金额</th>
    </tr>
    <c:forEach items="${accountList}" var="account">
        <tr>
            <td>${account.id}</td>
            <td>${account.name}</td>
            <td>${account.money}</td>
        </tr>
    </c:forEach>
</table>
```

11.编写相应配置文件(文件参考目录：素材/配置文件)

•Spring配置文件：applicationContext.xml

•SprngMVC配置文件：spring-mvc.xml

•MyBatis映射文件：AccountMapper.xml

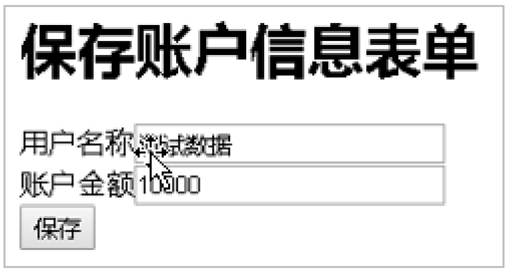
•MyBatis核心文件：sqlMapConfig.xml

•数据库连接信息文件：jdbc.properties

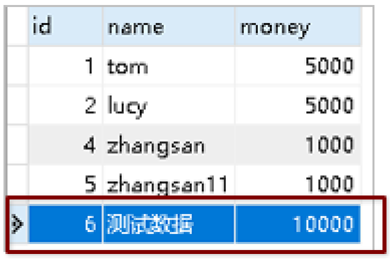
•Web.xml文件：web.xml

•日志文件：[log4j.xml](

12.测试添加账户

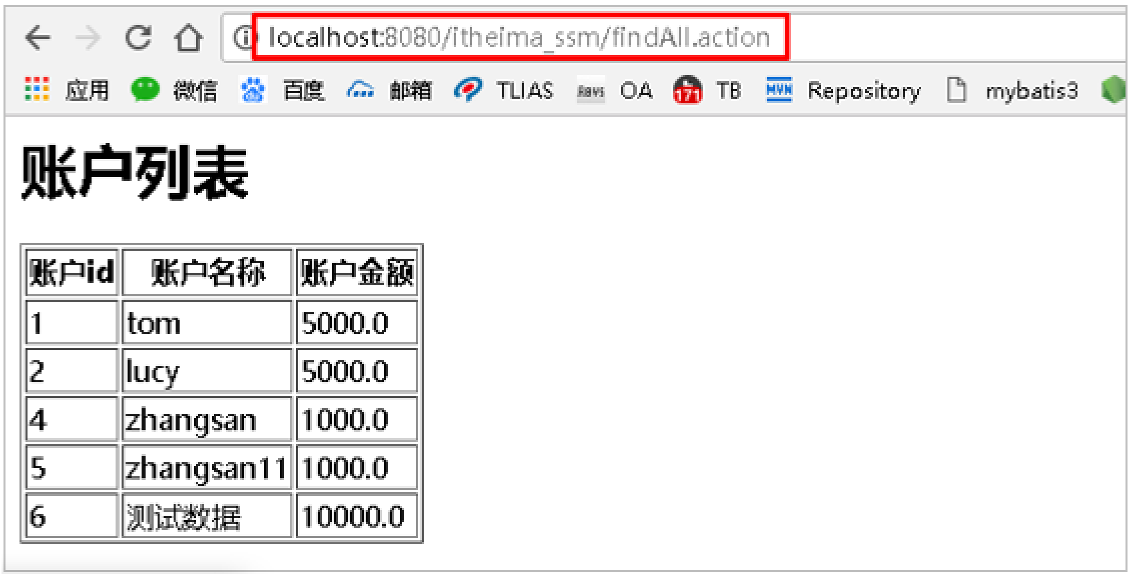

保存账户信息表单

用户名称	测试数据
账户金额	10000



id	name	money
1	tom	5000
2	lucy	5000
4	zhangsan	1000
5	zhangsan11	1000
6	测试数据	10000

13.测试账户列表

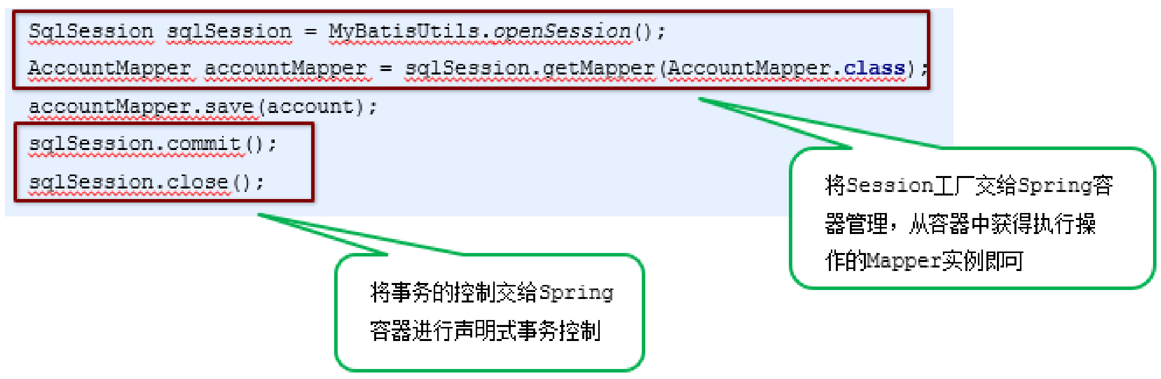

localhost:8080/itheima_ssm/findAll.action

账户列表

账户id	账户名称	账户金额
1	tom	5000.0
2	lucy	5000.0
4	zhangsan	1000.0
5	zhangsan11	1000.0
6	测试数据	10000.0

1.2 Spring整合MyBatis

1.整合思路



```java
SqlSession sqlSession = MyBatisUtils.openSession();
AccountMapper accountMapper = sqlSession.getMapper(AccountMapper.class);
accountMapper.save(account);
sqlSession.commit();
sqlSession.close();
```

将Session工厂交给Spring容器管理，从容器中获得执行操作的Mapper实例即可
将事务的控制交给Spring容器进行声明式事务控制

2.将SqlSessionFactory配置到Spring容器中

```xml
<!--加载jdbc.properties-->
<context:property-placeholder location="classpath:jdbc.properties"/>
<!--配置数据源-->
<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
    <property name="driverClass" value="${jdbc.driver}"/>
    <property name="jdbcUrl" value="${jdbc.url}"/>
    <property name="user" value="${jdbc.username}"/>
    <property name="password" value="${jdbc.password}"/>
</bean>
<!--配置MyBatis的SqlSessionFactory-->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource"/>
    <property name="configLocation" value="classpath:sqlMapConfig.xml"/>
</bean>
```

3.扫描Mapper，让Spring容器产生Mapper实现类

```xml
<!--配置Mapper扫描-->
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage" value="com.itheima.mapper"/>
</bean>
```

4.配置声明式事务控制

```xml
<!--配置声明式事务控制-->
<bean id="transacionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource"/>
</bean>
<tx:advice id="txAdvice" transaction-manager="transacionManager">
    <tx:attributes>
        <tx:method name="*"/>
    </tx:attributes>
</tx:advice>
<aop:config>
    <aop:pointcut id="txPointcut" expression="execution(*
com.itheima.service.impl.*.*(..))"/>
    <aop:advisor advice-ref="txAdvice" pointcut-ref="txPointcut"/>
</aop:config>
```

5.修改Service实现类代码

```java
@Service("accountService")
public class AccountServiceImpl implements AccountService {

    @Autowired
    private AccountMapper accountMapper;

    public void save(Account account) {
        accountMapper.save(account);
    }
    public List<Account> findAll() {
        return accountMapper.findAll();
    }
}
```