SSM订单操作

1.表结构分析

1.1.订单表信息描述 orders

序号	字段名称	字段类型	字段描述
1	id	varchar2(32)	无意义、主键uuid
2	orderNum	varchar2(50)	订单编号 不为空 唯一
3	orderTime	timestamp	下单时间
4	peopleCount	int	出行人数
5	orderDesc	varchar2(500)	订单描述(其它信息)
6	payType	int	支付方式(0 支付宝 1 微信 2其它)
7	orderStatus	int	订单状态(0 未支付 1 已支付)
8	productId	int	产品id 外键
9	memberid	int	会员(联系人）id 外键

productId描述了订单与产品之间的关系。

memberid描述了订单与会员之间的关系。

创建表sql

```
CREATE TABLE orders(
  id varchar2(32) default SYS_GUID() PRIMARY KEY,
  orderNum VARCHAR2(20) NOT NULL UNIQUE,
  orderTime timestamp,
  peopleCount INT,
  orderDesc VARCHAR2(500),
  payType INT,
  orderStatus INT,
  productId varchar2(32),
  memberId varchar2(32),
  FOREIGN KEY (productId) REFERENCES product(id),
  FOREIGN KEY (memberId) REFERENCES member(id)
)
insert into ORDERS (id, ordernum, ordertime, peoplecount, orderdesc, paytype, orderstatus,
productid, memberid)

values ('0E7231DC797C486290E8713CA3C6ECCC', '12345', to_timestamp('02-03-2018 12:00:00.000000',
```

```
'dd-mm-yyyy hh24:mi:ss.ff'), 2, '没什么', 0, 1, '676C5BD1D35E429A8C2E114939C5685A',
'E61D65F673D54F68B0861025C69773DB');
insert into ORDERS (id, ordernum, ordertime, peoplecount, orderdesc, paytype, orderstatus,
productid, memberid)
values ('5DC6A48DD4E94592AE904930EA866AFA', '54321', to_timestamp('02-03-2018 12:00:00.000000',
'dd-mm-yyyy hh24:mi:ss.ff'), 2, '没什么', 0, 1, '676C5BD1D35E429A8C2E114939C5685A',
'E61D65F673D54F68B0861025C69773DB');
insert into ORDERS (id, ordernum, ordertime, peoplecount, orderdesc, paytype, orderstatus,
productid, memberid)
values ('2FF351C4AC744E2092DCF08CFD314420', '67890', to_timestamp('02-03-2018 12:00:00.000000',
'dd-mm-yyyy hh24:mi:ss.ff'), 2, '没什么', 0, 1, '12B7ABF2A4C544568B0A7C69F36BF8B7',
'E61D65F673D54F68B0861025C69773DB');
insert into ORDERS (id, ordernum, ordertime, peoplecount, orderdesc, paytype, orderstatus,
productid, memberid)
values ('A0657832D93E4B10AE88A2D4B70B1A28', '98765', to_timestamp('02-03-2018 12:00:00.000000',
'dd-mm-yyyy hh24:mi:ss.ff'), 2, '没什么', 0, 1, '12B7ABF2A4C544568B0A7C69F36BF8B7',
'E61D65F673D54F68B0861025C69773DB');
insert into ORDERS (id, ordernum, ordertime, peoplecount, orderdesc, paytype, orderstatus,
productid, memberid)
values ('E4DD4C45EED84870ABA83574A801083E', '11111', to_timestamp('02-03-2018 12:00:00.000000',
'dd-mm-yyyy hh24:mi:ss.ff'), 2, '没什么', 0, 1, '12B7ABF2A4C544568B0A7C69F36BF8B7',
'E61D65F673D54F68B0861025C69773DB');
insert into ORDERS (id, ordernum, ordertime, peoplecount, orderdesc, paytype, orderstatus,
productid, memberid)
values ('96CC8BD43C734CC2ACBFF09501B4DD5D', '22222', to_timestamp('02-03-2018 12:00:00.000000',
'dd-mm-yyyy hh24:mi:ss.ff'), 2, '没什么', 0, 1, '12B7ABF2A4C544568B0A7C69F36BF8B7',
'E61D65F673D54F68B0861025C69773DB');
insert into ORDERS (id, ordernum, ordertime, peoplecount, orderdesc, paytype, orderstatus,
productid, memberid)
values ('55F9AF582D5A4DB28FB4EC3199385762', '33333', to_timestamp('02-03-2018 12:00:00.000000',
'dd-mm-yyyy hh24:mi:ss.ff'), 2, '没什么', 0, 1, '9F71F01CB448476DAFB309AA6DF9497F',
'E61D65F673D54F68B0861025C69773DB');
insert into ORDERS (id, ordernum, ordertime, peoplecount, orderdesc, paytype, orderstatus,
productid, memberid)
values ('CA005CF1BE3C4EF68F88ABC7DF30E976', '44444', to_timestamp('02-03-2018 12:00:00.000000',
'dd-mm-yyyy hh24:mi:ss.ff'), 2, '没什么', 0, 1, '9F71F01CB448476DAFB309AA6DF9497F',
'E61D65F673D54F68B0861025C69773DB');
insert into ORDERS (id, ordernum, ordertime, peoplecount, orderdesc, paytype, orderstatus,
productid, memberid)
values ('3081770BC3984EF092D9E99760FDABDE', '55555', to_timestamp('02-03-2018 12:00:00.000000',
'dd-mm-yyyy hh24:mi:ss.ff'), 2, '没什么', 0, 1, '9F71F01CB448476DAFB309AA6DF9497F',
'E61D65F673D54F68B0861025C69773DB');
```

实体类

```
public class Orders {

    private String id;
    private String orderNum;
    private Date orderTime;

    private String orderTimeStr;
```

```
    private int orderStatus;
    private int peopleCount;
    private Product product;
    private List<Traveller> travellers;
    private Member member;
    private Integer payType;
    private String payTypeStr;
    private String orderDesc;
    //省略getter/setter
}
```

1.2 会员表信息描述member

订单与会员之间是多对一关系，我们在订单表中创建一个外键来进行关联。

序号	字段名称	字段类型	字段描述
1	id	varchar2(32)	无意义、主键uuid
2	name	varchar2(20)	姓名
3	nickName	varchar2(20)	昵称
4	phoneNum	varchar2(20)	电话号码
5	email	varchar2(50)	邮箱

创建表sql

```
CREATE TABLE member(
       id varchar2(32) default SYS_GUID() PRIMARY KEY,
       NAME VARCHAR2(20),
       nickname VARCHAR2(20),
       phoneNum VARCHAR2(20),
       email VARCHAR2(20)
)
insert into MEMBER (id, name, nickname, phonenum, email)
values ('E61D65F673D54F68B0861025C69773DB', '张三', '小三', '18888888888', 'zs@163.com');
```

实体类

```
public class Member {

    private String id;
    private String name;
    private String nickname;
    private String phoneNum;
    private String email;
    //省略getter/setter
}
```

1.3.旅客表信息描述 traveller

序号	字段名称	字段类型	字段描述
1	id	varchar2(32)	无意义、主键uuid
2	name	varchar2(20)	姓名
3	sex	varchar2(20)	性别
4	phoneNum	varchar2(20)	电话号码
5	credentialsType	int	证件类型 0身份证 1护照 2军官证
6	credentialsNum	varchar2(50)	证件号码
7	travellerType	int	旅客类型(人群) 0 成人 1 儿童

创建表sql

```
CREATE TABLE traveller(
    id varchar2(32) default SYS_GUID() PRIMARY KEY,
    NAME VARCHAR2(20),
    sex VARCHAR2(20),
    phoneNum VARCHAR2(20),
    credentialsType INT,
    credentialsNum VARCHAR2(50),
    travellerType INT
)
insert into TRAVELLER (id, name, sex, phonenum, credentialstype, credentialsnum, travellertype)
values ('3FE27DF2A4E44A6DBC5D0FE4651D3D3E', '张龙', '男', '13333333333', 0,
'123456789009876543', 0);
insert into TRAVELLER (id, name, sex, phonenum, credentialstype, credentialsnum, travellertype)
values ('EE7A71FB6945483FBF91543DBE851960', '张小龙', '男', '15555555555', 0,
'987654321123456789', 1);
```

实体类

```
public class Traveller {
    private String id;
    private String name;
    private String sex;
    private String phoneNum;
    private Integer credentialsType;
    private String credentialsTypeStr;
    private String credentialsNum;
    private Integer travellerType;
    private String travellerTypeStr;
    //省略getter/setter
}
```

旅客与订单之间是多对多关系，所以我们需要一张中间表（order_traveller）来描述。

序号	字段名称	字段类型	字段描述
1	orderId	varchar2(32)	订单id
2	travellerId	varchar2(32)	旅客id

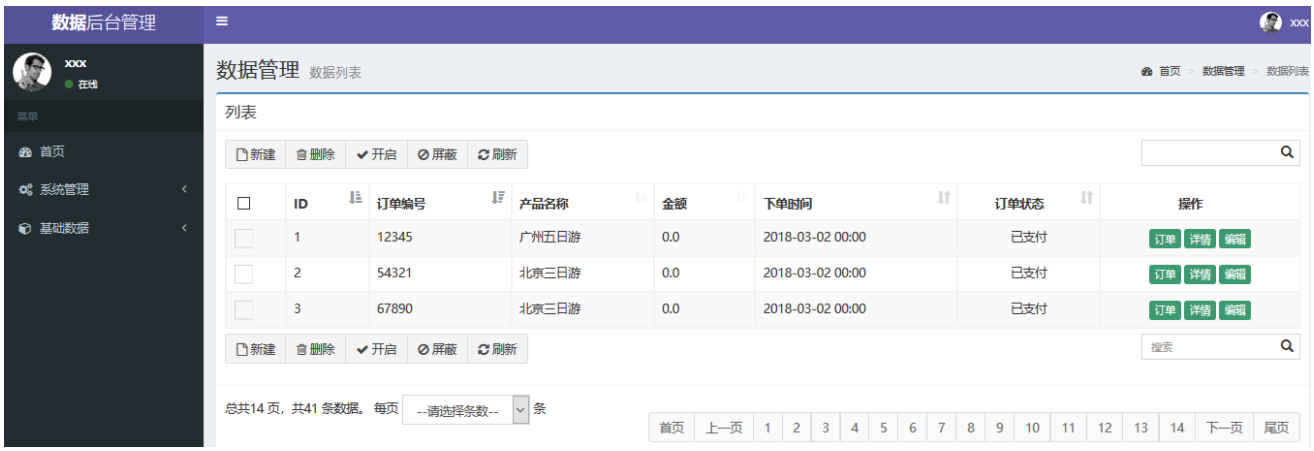
创建表sql

```
CREATE TABLE order_traveller(
    orderId varchar2(32),
    travellerId varchar2(32),
    PRIMARY KEY (orderId,travellerId),
    FOREIGN KEY (orderId) REFERENCES orders(id),
    FOREIGN KEY (travellerId) REFERENCES traveller(id)
)

insert into ORDER_TRAVELLER (orderid, travellerid)
values ('0E7231DC797C486290E8713CA3C6ECCC', '3FE27DF2A4E44A6DBC5D0FE4651D3D3E');
insert into ORDER_TRAVELLER (orderid, travellerid)
values ('2FF351C4AC744E2092DCF08CFD314420', '3FE27DF2A4E44A6DBC5D0FE4651D3D3E');
insert into ORDER_TRAVELLER (orderid, travellerid)
values ('3081770BC3984EF092D9E99760FDABDE', 'EE7A71FB6945483FBF91543DBE851960');
insert into ORDER_TRAVELLER (orderid, travellerid)
values ('55F9AF582D5A4DB28FB4EC3199385762', 'EE7A71FB6945483FBF91543DBE851960');
insert into ORDER_TRAVELLER (orderid, travellerid)
values ('5DC6A48DD4E94592AE904930EA866AFA', '3FE27DF2A4E44A6DBC5D0FE4651D3D3E');
insert into ORDER_TRAVELLER (orderid, travellerid)
values ('96CC8BD43C734CC2ACBFF09501B4DD5D', 'EE7A71FB6945483FBF91543DBE851960');
insert into ORDER_TRAVELLER (orderid, travellerid)
values ('A0657832D93E4B10AE88A2D4B70B1A28', '3FE27DF2A4E44A6DBC5D0FE4651D3D3E');
insert into ORDER_TRAVELLER (orderid, travellerid)
values ('CA005CF1BE3C4EF68F88ABC7DF30E976', 'EE7A71FB6945483FBF91543DBE851960');
insert into ORDER_TRAVELLER (orderid, travellerid)
values ('E4DD4C45EED84870ABA83574A801083E', 'EE7A71FB6945483FBF91543DBE851960');
```

2.订单查询

2.1 订单查询页面 order-list.jsp



详细代码请查看今天课程资料

2.2 Controller

```
@Controller
@RequestMapping("/orders")
public class OrdersController {

    @Autowired
    private IOrdersService ordersService;

    //未分页
    @RequestMapping("/findAll.do")
    public ModelAndView findAll(@RequestParam(name = "page", required = true, defaultValue =
"1") Integer page, @RequestParam(name = "pageSize", required = true, defaultValue = "10")
Integer pageSize) throws Exception {
        List<Orders> ordersList = ordersService.findAllByPage(page, pageSize);
        ModelAndView mv = new ModelAndView();
        mv.setViewName("order-list");
        mv.addObject("ordersList", ordersList);
        return mv;
    }
}
```

2.3 Dao

IOrdersDao

```
public interface IOrdersDao {
    @Select("select * from orders")
    @Results({
            @Result(id=true,column = "id",property = "id"),
```

```
            @Result(column = "orderNum",property = "orderNum"),
            @Result(column = "orderTime",property = "orderTime"),
            @Result(column = "orderStatus",property = "orderStatus"),
            @Result(column = "peopleCount",property = "peopleCount"),
            @Result(column = "payType",property = "payType"),
            @Result(column = "orderDesc",property = "orderDesc"),
            @Result(column = "productId",property = "product",one = @One(select =
"com.itheima.ssm.dao.IProductDao.findById"))
    })
    List<Orders> findAll() throws Exception;
}
```

IProductDao的findById

```
@Select("select * from product where id=#{id}")
Product findById(String id) throws Exception;
```

3.订单分页查询

3.1 PageHelper介绍

PageHelper是国内非常优秀的一款开源的mybatis分页插件，它支持基本主流与常用的数据库，例如mysql、oracle、mariaDB、DB2、SQLite、Hsqldb等。

本项目在 github 的项目地址：https://github.com/pagehelper/Mybatis-PageHelper

本项目在 gitosc 的项目地址：http://git.oschina.net/free/Mybatis_PageHelper

3.2 PageHelper使用

3.2.1.集成

引入分页插件有下面2种方式，推荐使用 Maven 方式。

3.2.1.1. 引入 Jar 包

你可以从下面的地址中下载最新版本的 jar 包

- https://oss.sonatype.org/content/repositories/releases/com/github/pagehelper/pagehelper/
- http://repo1.maven.org/maven2/com/github/pagehelper/pagehelper/

由于使用了sql 解析工具，你还需要下载 jsqlparser.jar：

- http://repo1.maven.org/maven2/com/github/jsqlparser/jsqlparser/0.9.5/

3.2.1.2. 使用 Maven

在 pom.xml 中添加如下依赖：

```
<dependency>
    <groupId>com.github.pagehelper</groupId>
    <artifactId>pagehelper</artifactId>
    <version>最新版本</version>
</dependency>
```

3.2. 2.配置

特别注意，新版拦截器是 `com.github.pagehelper.PageInterceptor`。 `com.github.pagehelper.PageHelper` 现在是一个特殊的 `dialect` 实现类，是分页插件的默认实现类，提供了和以前相同的用法。

3.2.2.1. 在 MyBatis 配置 xml 中配置拦截器插件

```
<!--
    plugins在配置文件中的位置必须符合要求，否则会报错，顺序如下:
    properties?, settings?,
    typeAliases?, typeHandlers?,
    objectFactory?,objectWrapperFactory?,
    plugins?,
    environments?, databaseIdProvider?, mappers?
-->
<plugins>
    <!-- com.github.pagehelper为PageHelper类所在包名 -->
    <plugin interceptor="com.github.pagehelper.PageInterceptor">
        <!-- 使用下面的方式配置参数，后面会有所有的参数介绍 -->
        <property name="param1" value="value1"/>
    </plugin>
</plugins>
```

3.2.2.2. 在 Spring 配置文件中配置拦截器插件

使用 spring 的属性配置方式，可以使用 `plugins` 属性像下面这样配置:

```
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
  <!-- 注意其他配置 -->
  <property name="plugins">
    <array>
      <bean class="com.github.pagehelper.PageInterceptor">
        <property name="properties">
          <!--使用下面的方式配置参数，一行配置一个 -->
          <value>
            params=value1
          </value>
        </property>
      </bean>
    </array>
  </property>
</bean>
```

3.2.3 分页插件参数介绍

1. `helperDialect`：分页插件会自动检测当前的数据库链接，自动选择合适的分页方式。 你可以配置`helperDialect`属性来指定分页插件使用哪种方言。配置时，可以使用下面的缩写值：
   `oracle`,`mysql`,`mariadb`,`sqlite`,`hsqldb`,`postgresql`,`db2`,`sqlserver`,`informix`,`h2`,`sqlserver2012`,`derby`
   特别注意：使用 SqlServer2012 数据库时，需要手动指定为`sqlserver2012`，否则会使用 SqlServer2005 的方式进行分页。
   你也可以实现`AbstractHelperDialect`，然后配置该属性为实现类的全限定名称即可使用自定义的实现方法。
2. `offsetAsPageNum`：默认值为`false`，该参数对使用`RowBounds`作为分页参数时有效。 当该参数设置为`true`时，会将`RowBounds`中的`offset`参数当成`pageNum`使用，可以用页码和页面大小两个参数进行分页。
3. `rowBoundsWithCount`：默认值为`false`，该参数对使用`RowBounds`作为分页参数时有效。 当该参数设置为`true`时，使用`RowBounds`分页会进行 count 查询。
4. `pageSizeZero`：默认值为`false`，当该参数设置为`true`时，如果`pageSize=0`或者`RowBounds.limit = 0`就会查询出全部的结果（相当于没有执行分页查询，但是返回结果仍然是`Page`类型）。
5. `reasonable`：分页合理化参数，默认值为`false`。当该参数设置为`true`时，`pageNum<=0`时会查询第一页，`pageNum>pages`（超过总数时），会查询最后一页。默认`false`时，直接根据参数进行查询。
6. `params`：为了支持`startPage(Object params)`方法，增加了该参数来配置参数映射，用于从对象中根据属性名取值， 可以配置`pageNum,pageSize,count,pageSizeZero,reasonable`，不配置映射的用默认值， 默认值为
   `pageNum=pageNum;pageSize=pageSize;count=countSql;reasonable=reasonable;pageSizeZero=pageSizeZero`。
7. `supportMethodsArguments`：支持通过 Mapper 接口参数来传递分页参数，默认值`false`，分页插件会从查询方法的参数值中，自动根据上面`params`配置的字段中取值，查找到合适的值时就会自动分页。 使用方法可以参考测试代码中的`com.github.pagehelper.test.basic`包下的`ArgumentsMapTest`和`ArgumentsObjTest`。
8. `autoRuntimeDialect`：默认值为`false`。设置为`true`时，允许在运行时根据多数据源自动识别对应方言的分页 （不支持自动选择`sqlserver2012`，只能使用`sqlserver`），用法和注意事项参考下面的**场景五**。
9. `closeConn`：默认值为`true`。当使用运行时动态数据源或没有设置`helperDialect`属性自动获取数据库类型时，会自动获取一个数据库连接， 通过该属性来设置是否关闭获取的这个连接，默认`true`关闭，设置为`false`后，不会关闭获取的连接，这个参数的设置要根据自己选择的数据源来决定。

3.2.4.基本使用

PageHelper的基本使用有6种，大家可以查看文档，最常用的有两种

3.2.4.1. RowBounds方式的调用（了解）

```
List<Country> list = sqlSession.selectList("x.y.selectIf", null, new RowBounds(1, 10));
```

使用这种调用方式时，你可以使用RowBounds参数进行分页，这种方式侵入性最小，我们可以看到，通过RowBounds方式调用只是使用了这个参数，并没有增加其他任何内容。

分页插件检测到使用了RowBounds参数时，就会对该查询进行**物理分页。**

关于这种方式的调用，有两个特殊的参数是针对`RowBounds`的，你可以参看上面的分页插件参数介绍

注：不只有命名空间方式可以用RowBounds，使用接口的时候也可以增加RowBounds参数，例如：

```
//这种情况下也会进行物理分页查询
List<Country> selectAll(RowBounds rowBounds);
```

注意： 由于默认情况下的 `RowBounds` 无法获取查询总数，分页插件提供了一个继承自 `RowBounds` 的 `PageRowBounds`，这个对象中增加了 `total` 属性，执行分页查询后，可以从该属性得到查询总数。

3.2.4.2. PageHelper.startPage 静态方法调用（重点）

这种方式是我们要掌握的 在你需要进行分页的 MyBatis 查询方法前调用PageHelper.startPage 静态方法即可，紧跟在这个方法后的第一个MyBatis 查询方法会被进行分页。

```
//获取第1页，10条内容，默认查询总数count
PageHelper.startPage(1, 10);
//紧跟着的第一个select方法会被分页
List<Country> list = countryMapper.selectIf(1);
```

3.3 订单分页查询

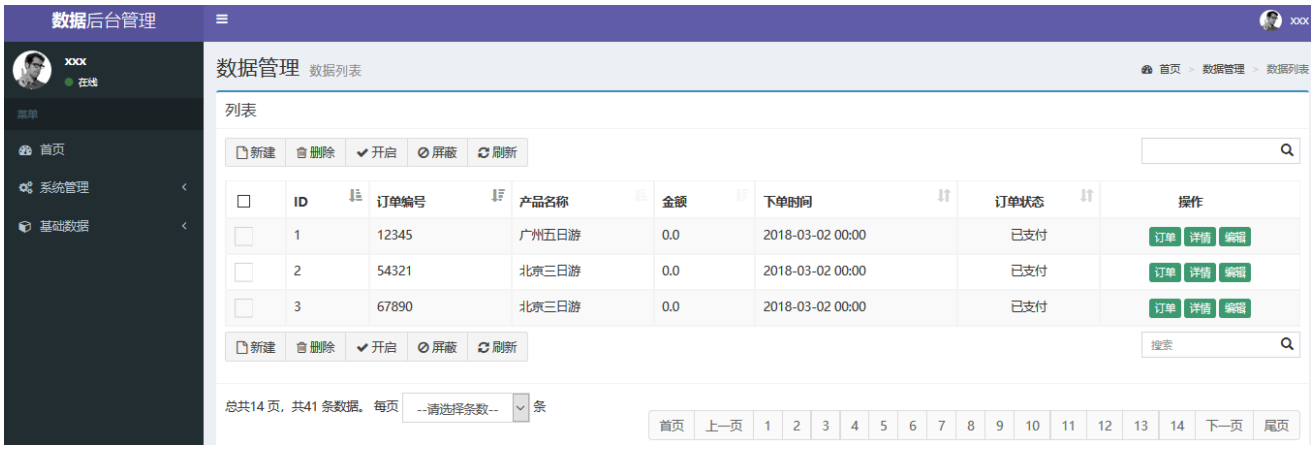
3.3.1 Service

```
@Override
public List<Orders> findAllByPage(int page, int pageSize) throws Exception {
        PageHelper.startPage(page, pageSize);
        return ordersDao.findAllByPage();
}
```

3.3.2 Controller

```
@RequestMapping("/findAll.do")
public ModelAndView findAll(@RequestParam(name = "page", required = true, defaultValue = "1")
Integer page, @RequestParam(name = "pageSize", required = true, defaultValue = "10") Integer
pageSize) throws Exception {
        List<Orders> ordersList = ordersService.findAllByPage(page, pageSize);
        PageInfo pageInfo = new PageInfo(ordersList);
        ModelAndView mv = new ModelAndView();
        mv.setViewName("order-list");
        mv.addObject("pageInfo", pageInfo);
        return mv;
    }
```

3.3.3 订单分页查询页面Order-list.jsp



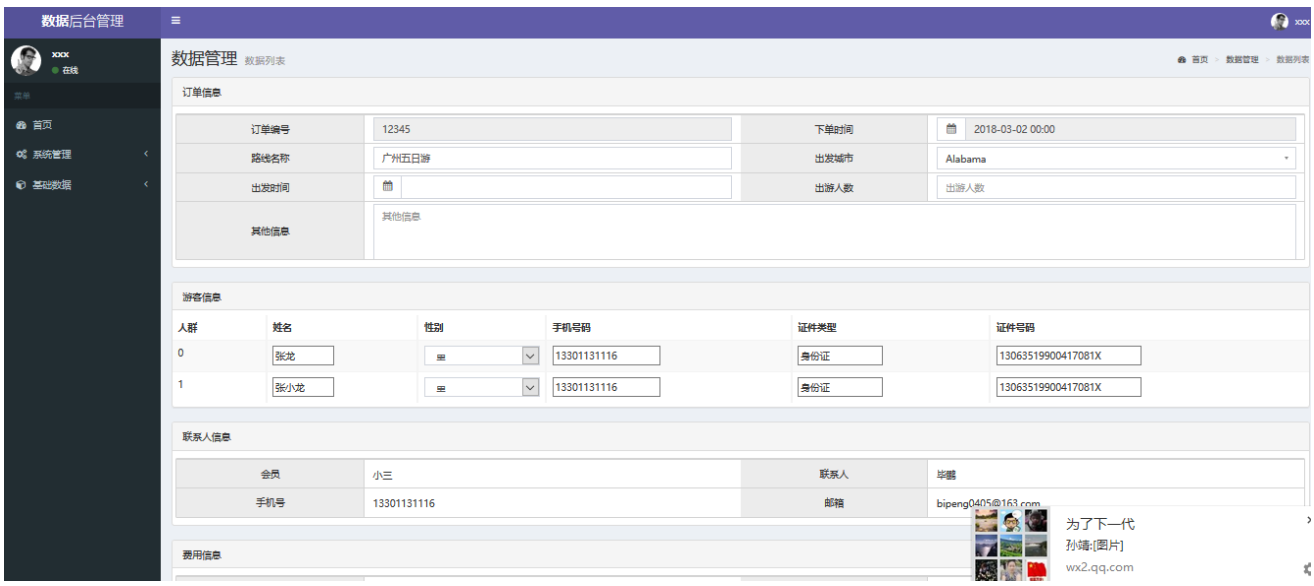
详细代码请查看今天课程资料

4.订单详情

在order-list.jsp页面上对"详情"添加链接

```
<button type="button" class="btn bg-olive btn-xs"
onclick="location.href='${pageContext.request.contextPath}/orders/findById.do?id=${orders.id}'">
详情</button>
```

4.1 订单详情 order-show.jsp



详细代码请查看今天课程资料

5.2 Controller

```
@RequestMapping("/findById.do")
    public ModelAndView findById(String id) throws Exception {
        Orders orders = ordersService.findById(id);
        ModelAndView mv = new ModelAndView();
        mv.setViewName("order-show");
        mv.addObject("orders", orders);
        return mv;
    }
```

5.3 Dao

IOrdersDao的findById方法

```
@Select("select * from orders where id=#{id}")
    @Results({
            @Result(id=true,column = "id",property = "id"),
            @Result(column = "orderNum",property = "orderNum"),
            @Result(column = "orderTime",property = "orderTime"),
            @Result(column = "orderStatus",property = "orderStatus"),
            @Result(column = "peopleCount",property = "peopleCount"),
            @Result(column = "payType",property = "payType"),
            @Result(column = "orderDesc",property = "orderDesc"),
            @Result(column = "productId",property = "product",one = @One(select =
"com.itheima.ssm.dao.IProductDao.findById")),
            @Result(column = "id",property = "travellers",many = @Many(select =
"com.itheima.ssm.dao.ITravellerDao.findByOrdersId")),
            @Result(column = "memberId",property = "member",one = @One(select =
"com.itheima.ssm.dao.IMemberDao.findById")),
    })
    Orders findById(String id) throws Exception;
```

IMemberDao的findById方法

```
@Select("select * from member where id=#{id}")
Member findById(String id) throws Exception;
```

ITravellerDao.findByOrdersId方法

```
  @Select("select * from traveller where id in (select travellerId from order_traveller where
orderId=#{ordersId})")
  List<Traveller> findByOrdersId(String ordersId) throws Exception;
```