SSM综合练习介绍

1.功能介绍

```
### 1.1 环境搭建
```

主要讲解maven工程搭建，以及基于oracle数据库的商品表信息，并完成SSM整合。

1.2 商品查询

基于SSM整合基础上完成商品查询，要掌握主面页面main.jsp及商品显示页面product-list.jsp页面的创建。

1.3 商品添加

进一步巩固SSM整合，并完成商品添加功能，要注意事务操作以及product-add.jsp页面生成。

1.4 订单查询

订单的查询操作，它主要完成简单的多表查询操作，查询订单时，需要查询出与订单关联的其它表中信息，所以大家一定要了解订单及其它表关联关系

1.5 订单分页查询

订单分页查询，我们使用的是mybatis分页插件PageHelper，要掌握PageHelper的基本使用。

1.6 订单详情查询

订单详情是用于查询某一个订单的信息，这个知识点主要考核学生对复杂的多表查询操作的掌握。

1.7 Spring Security 概述

Spring Security是 Spring 项目组中用来提供安全认证服务的框架，它的使用很复杂，我们在课程中只介绍了spring Security的基本操作，大家要掌握spring Security框架的配置及基本的认证与授权操作。

1.8 用户管理

用户管理中我们会介绍基于spring Security的用户登录、退出操作。以及用户查询、添加、详情有等操作，这些功能的练习是对前期SSM知识点的进一步巩固。

1.9 角色管理

角色管理主要完成角色查询、角色添加

1.10 资源权限管理

资源权限管理主要完成查询、添加操作，它的操作与角色管理类似，角色管理以及资源权限管理都是对权限管理的补充。

1.11 权限关联与控制

主要会讲解用户角色关联、角色权限关联，这两个操作是为了后续我们完成授权操作的基础，关于授权操作我们会在服务器端及页面端分别讲解

1.12 AOP日志处理

AOP日志处理，我们使用spring AOP切面来完成系统级别的日志收集。

2.数据库介绍

2.1 产品表

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	productNum	varchar2(50)	产品编号，唯一，不为空
3	productName	varchar2(50)	产品名称（路线名称）
4	cityName	varchar2(50)	出发城市
5	DepartureTime	timestamp	出发时间
6	productPrice	number	产品价格
7	productDesc	varchar2(500)	产品描述
8	productStatus	int	状态(0 关闭 1 开启)

2.2 订单表

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

2.3 会员表

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

2.4 旅客表

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

2.5 用户表

序号	字段名称	字段类型	字段描述
1	id	varchar2	无意义，主键uuid
2	email	varchar2	非空，唯一
3	username	varchar2	用户名
5	password	varchar2	密码（加密）
6	phoneNum	varchar2	电话
7	status	int	状态0 未开启 1 开启

2.5 角色表

序号	字段名称	字段类型	字段描述
1	id	varchar2	无意义，主键uuid
2	roleName	varchar2	角色名
3	roleDesc	varchar2	角色描述

2.6 资源权限表

序号	字段名称	字段类型	字段描述
1	id	varchar2	无意义，主键uuid
2	permissionName	varchar2	权限名
3	url	varchar2	资源路径

2.7 日志表

序号	字段名称	字段类型	字段描述
1	id	VARCHAR2	主键 无意义uuid
2	visitTime	timestamp	访问时间
3	username	VARCHAR2	操作者用户名
4	ip	VARCHAR2	访问ip
5	url	VARCHAR2	访问资源url
6	executionTime	int	执行时长
7	method	VARCHAR	访问方法

SSM 环境搭建与产品操作

1.环境准备

1.1 数据库与表结构

1.1.1 创建用户与授权

数据库我们使用Oracle

Oracle 为每个项目创建单独user，oracle数据表存放在表空间下，每个用户有独立表空间

创建用户及密码：

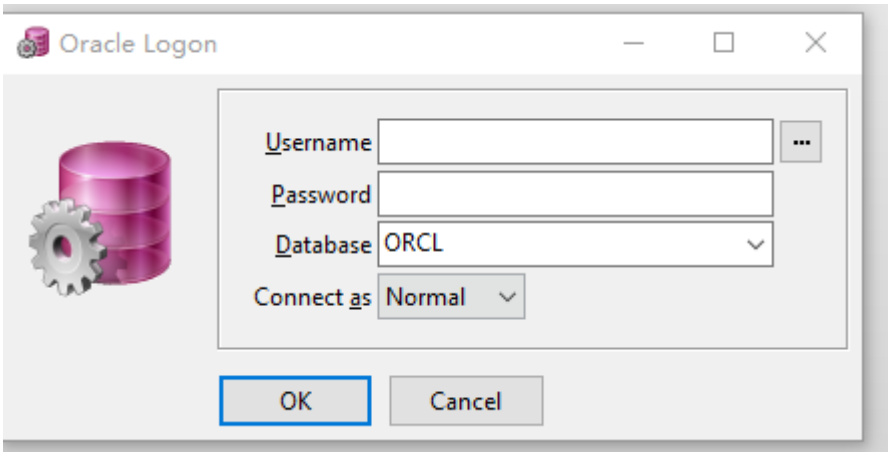
```
语法[创建用户]： create user 用户名 identified by 口令[即密码];
例子： create user test identified by test;
```

授权

```
语法： grant connect, resource to 用户名;
 例子： grant connect, resource to test;
```

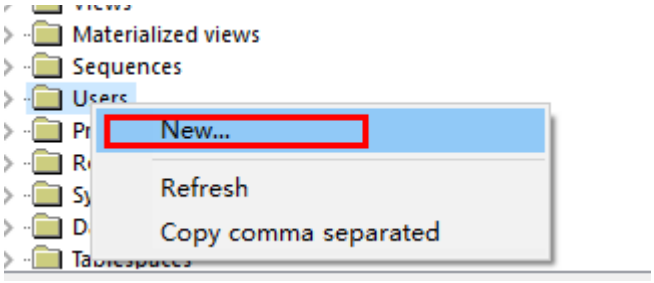
PL/SQL Developer是一个集成开发环境，专门面向Oracle数据库存储程序单元的开发PL/SQL Developer侧重于易用性、代码品质和生产力，充分发挥Oracle应用程序开发过程中的主要优势。

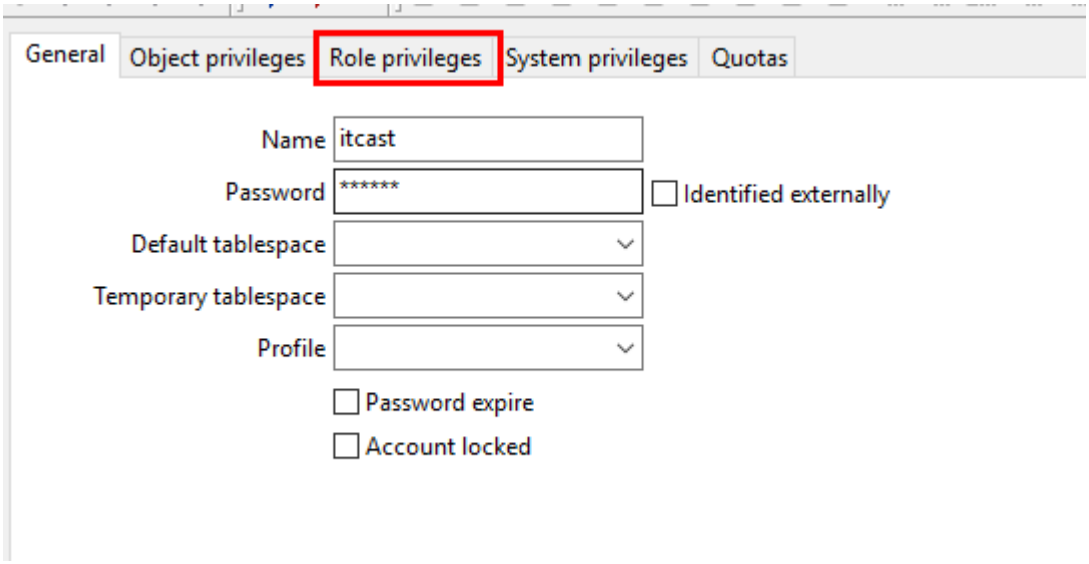
- 连接oracle数据库



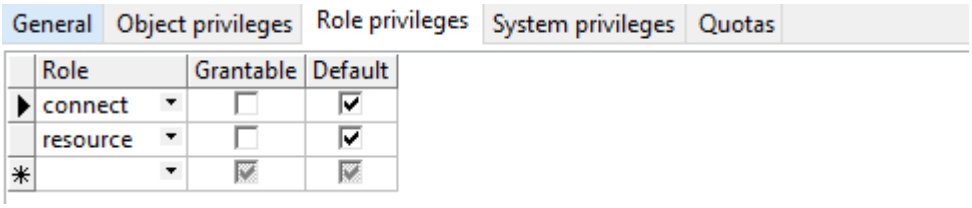
- 创建用户及授权

  a) 创建用户





b) 授权



对象权限是指针对于某一张表的操作权限,系统权限是指对表的CRUD操作权限, 角色权限是系统权限的集合,我们设置 时，一般是设置角色权限，设置resource与connect

1.1.2 创建表

产品表信息描述

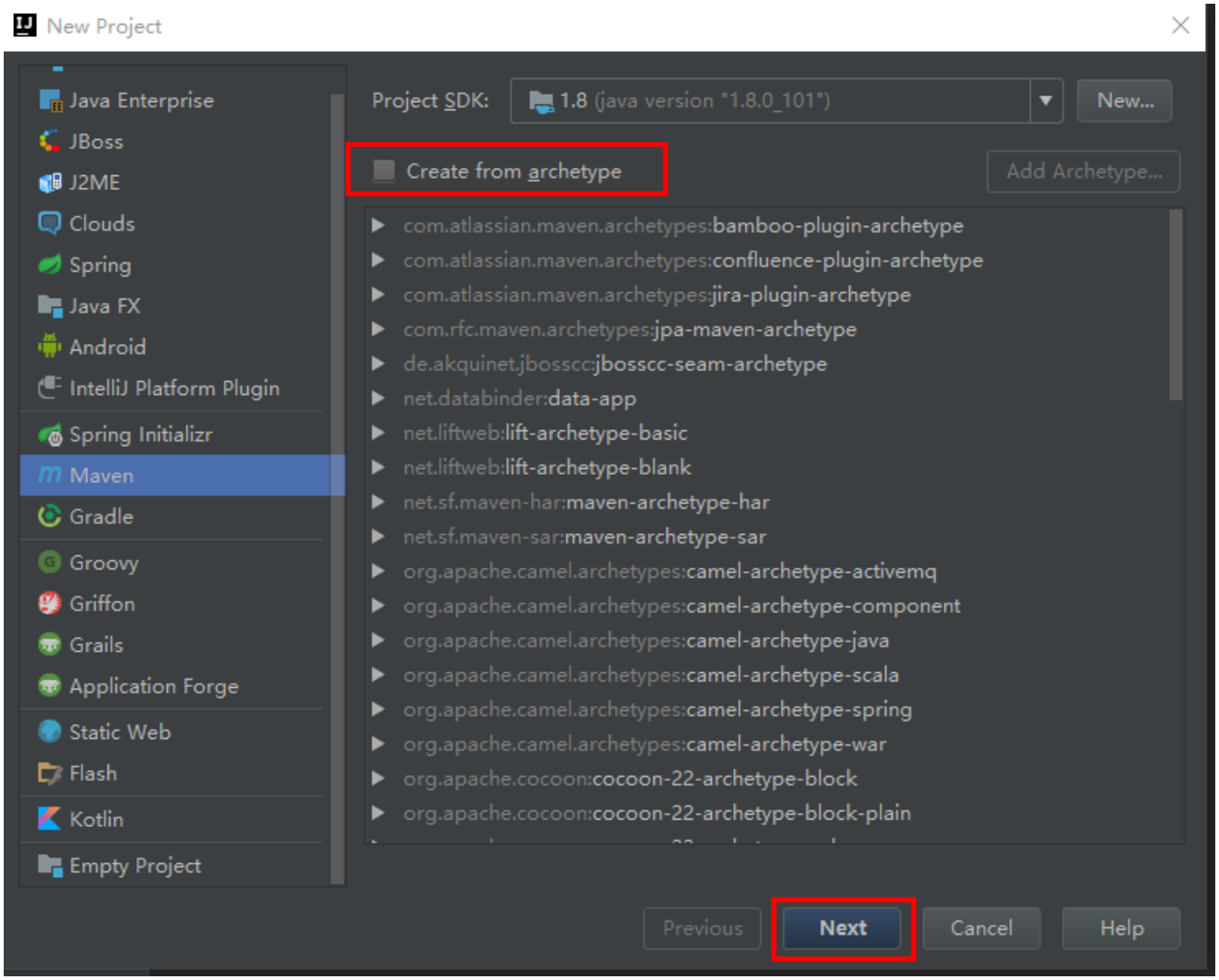
序号	字段名称	字段类型	字段描述
1	id	varchar2(32)	无意义，主键uuid
2	productNum	varchar2(50)	产品编号，唯一，不为空
3	productName	varchar2(50)	产品名称（路线名称）
4	cityName	varchar2(50)	出发城市
5	DepartureTime	timestamp	出发时间
6	productPrice	number	产品价格
7	productDesc	varchar2(500)	产品描述
8	productStatus	int	状态(0 关闭 1 开启)

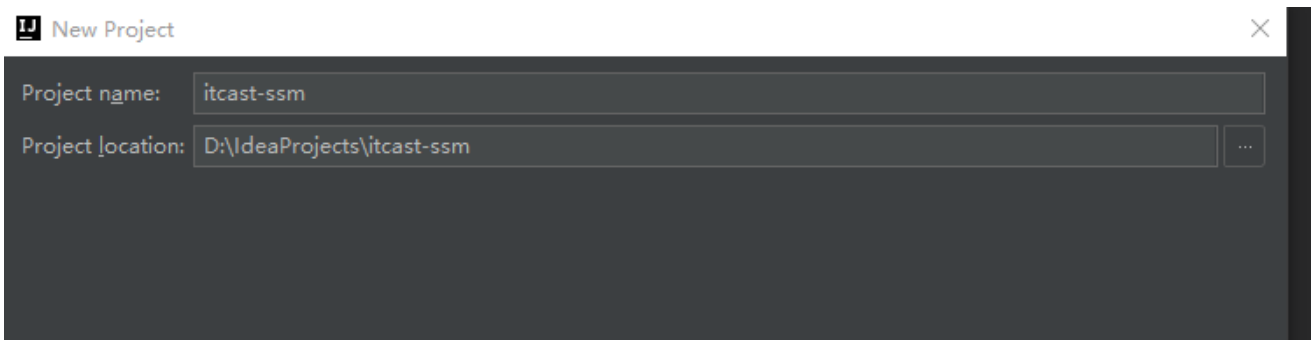
创建表sql

```
CREATE TABLE product(
  id varchar2(32) default SYS_GUID() PRIMARY KEY,
  productNum VARCHAR2(50) NOT NULL,
  productName VARCHAR2(50),
  cityName VARCHAR2(50),
  DepartureTime timestamp,
  productPrice Number,
  productDesc VARCHAR2(500),
  productStatus INT,
  CONSTRAINT product UNIQUE (id, productNum)
)

insert into PRODUCT (id, productnum, productname, cityname, departuretime, productprice,
productdesc, productstatus)
values ('676C5BD1D35E429A8C2E114939C5685A', 'itcast-002', '北京三日游', '北京', to_timestamp('10-
10-2018 10:10:00.000000', 'dd-mm-yyyy hh24:mi:ss.ff'), 1200, '不错的旅行', 1);
insert into PRODUCT (id, productnum, productname, cityname, departuretime, productprice,
productdesc, productstatus)
values ('12B7ABF2A4C544568B0A7C69F36BF8B7', 'itcast-003', '上海五日游', '上海', to_timestamp('25-
04-2018 14:30:00.000000', 'dd-mm-yyyy hh24:mi:ss.ff'), 1800, '魔都我来了', 0);
insert into PRODUCT (id, productnum, productname, cityname, departuretime, productprice,
productdesc, productstatus)
values ('9F71F01CB448476DAFB309AA6DF9497F', 'itcast-001', '北京三日游', '北京', to_timestamp('10-
10-2018 10:10:00.000000', 'dd-mm-yyyy hh24:mi:ss.ff'), 1200, '不错的旅行', 1);
```

1.2 maven工程搭建

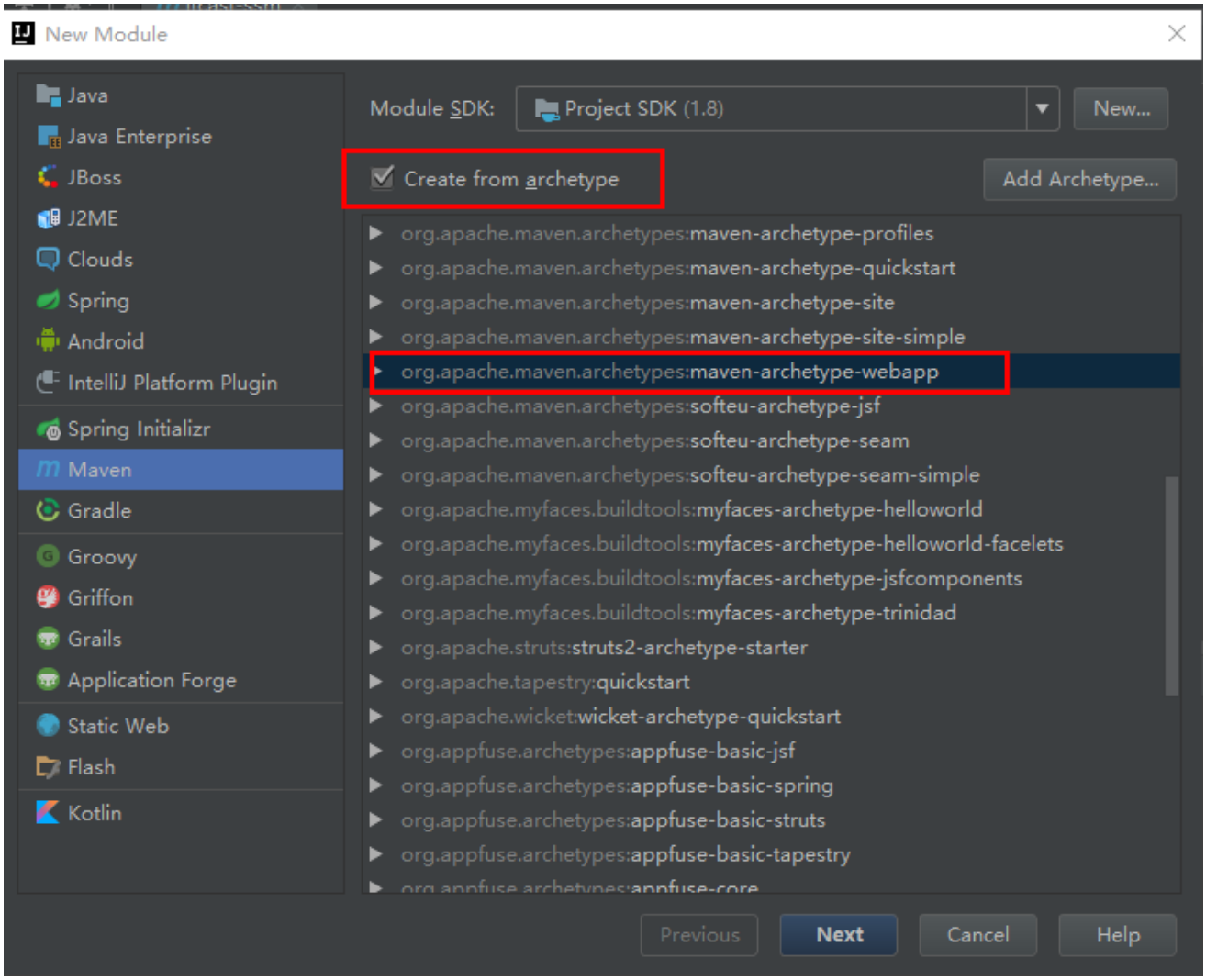
1.2.1 创建maven工程





创建子模块

itcast-ssm-web itcast-ssm-domain itcast-ssm-service itcast-ssm-dao itcast-ssm-utils 其中创建itcast-ssm-web时注意我们选择一个web工程



1.2.2 pom.xml

```
<properties>
     <spring.version>5.0.2.RELEASE</spring.version>
     <slf4j.version>1.6.6</slf4j.version>
     <log4j.version>1.2.12</log4j.version>
     <oracle.version>11.2.0.1.0</oracle.version>
     <mybatis.version>3.4.5</mybatis.version>
     <spring.security.version>5.0.1.RELEASE</spring.security.version>
 </properties>

 <dependencies>        <!-- spring -->
     <dependency>
         <groupId>org.aspectj</groupId>
         <artifactId>aspectjweaver</artifactId>

         <version>1.6.8</version>
```

```
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-aop</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context-support</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-web</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-orm</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-beans</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-test</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>${spring.version}</version>

        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-tx</artifactId>
            <version>${spring.version}</version>
        </dependency>

        <dependency>
```

```
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.12</version>
        <scope>test</scope>
    </dependency>

    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>3.1.0</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>javax.servlet.jsp</groupId>
        <artifactId>jsp-api</artifactId>
        <version>2.0</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>jstl</groupId>
        <artifactId>jstl</artifactId>
        <version>1.2</version>
    </dependency>        <!-- log start -->
    <dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>${log4j.version}</version>
    </dependency>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-api</artifactId>
        <version>${slf4j.version}</version>
    </dependency>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-log4j12</artifactId>
        <version>${slf4j.version}</version>
    </dependency>        <!-- log end -->

    <dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis</artifactId>
        <version>${mybatis.version}</version>
    </dependency>
    <dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis-spring</artifactId>
        <version>1.3.0</version>
    </dependency>
    <dependency>
        <groupId>c3p0</groupId>
        <artifactId>c3p0</artifactId>

        <version>0.9.1.2</version>
```

```
            <type>jar</type>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>com.github.pagehelper</groupId>
            <artifactId>pagehelper</artifactId>
            <version>5.1.2</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-web</artifactId>
            <version>${spring.security.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-config</artifactId>
            <version>${spring.security.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-core</artifactId>
            <version>${spring.security.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-taglibs</artifactId>
            <version>${spring.security.version}</version>
        </dependency>

        <dependency>
            <groupId>com.oracle</groupId>
            <artifactId>ojdbc14</artifactId>
            <version>${oracle.version}</version>
        </dependency>

        <dependency>
            <groupId>javax.annotation</groupId>
            <artifactId>jsr250-api</artifactId>
            <version>1.0</version>
        </dependency>

    </dependencies>
    <build>
         <pluginManagement>
            <plugins>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-compiler-plugin</artifactId>
                    <version>3.2</version>
                    <configuration>
                        <source>1.8</source>
                        <target>1.8</target>

                        <encoding>UTF-8</encoding>
```

```
                    <showWarnings>true</showWarnings>
                </configuration>
            </plugin>
        </plugins>
    </pluginManagement>
</build>
```

1.3编写实体类

```
public class Product {
    private String id; // 主键
    private String productNum; // 编号 唯一
    private String productName; // 名称
    private String cityName; // 出发城市
    private Date departureTime; // 出发时间
    private String departureTimeStr;
    private double productPrice; // 产品价格
    private String productDesc; // 产品描述
    private Integer productStatus; // 状态 0 关闭 1 开启
    private String productStatusStr;
 }
```

1.4 编写业务接口

```
public interface IProductService {

    List<Product> findAll() throws Exception;

}
```

1.5 编写持久层接口

```
public interface IProductDao {

    @Select("select * from product")
    List<Product> findAll() throws Exception;
}
```

2.SSM整合与产品查询

2.1 Spring环境搭建

2.1.1.编写Spring配置文件applicationContext.xml

```
<!-- 配置 spring 创建容器时要扫描的包 -->
<!-- 开启注解扫描，管理service和dao -->
<context:component-scan base-package="com.itheima.ssm.service">
</context:component-scan>
<context:component-scan base-package="com.itheima.ssm.dao">
</context:component-scan>
```

2.1.2.使用注解配置业务层

```
@Service
public class ProductServiceImpl implements IProductService{

    @Override
    public List<Product> findAll() throws Exception {
        return null;
    }

}
```

2.2 Spring MVC 环境搭建

2.2.1.web.xml配置Spring MVC核心控制器

```
<!-- 配置 spring mvc 的核心控制器 -->
<servlet>
    <servlet-name>springmvcDispatcherServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <!-- 配置初始化参数，用于读取 springmvc 的配置文件 -->
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:springmvc.xml</param-value>
    </init-param>
    <!-- 配置 servlet 的对象的创建时间点：应用加载时创建。取值只能是非 0 正整数，表示启动顺 序 -->
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>springmvcDispatcherServlet</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
<!-- 配置 springMVC 编码过滤器 -->
<filter>
    <filter-name>CharacterEncodingFilter</filter-name>
    <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
    <!-- 设置过滤器中的属性值 -->
```

```
    <init-param>
        <param-name>encoding</param-name>
        <param-value>UTF-8</param-value>
    </init-param>
    <!-- 启动过滤器 -->
    <init-param>
        <param-name>forceEncoding</param-name>
        <param-value>true</param-value>
    </init-param>
</filter>
<!-- 过滤所有请求 -->
<filter-mapping>
    <filter-name>CharacterEncodingFilter</filter-name>
    <url-pattern>*.do</url-pattern>
</filter-mapping>
```

2.2.2.Spring MVC配置文件springmvc.xml

```
<!-- 扫描controller的注解，别的不扫描 -->
<context:component-scan base-package="com.itheima.ssm.controller">
</context:component-scan>

<!-- 配置视图解析器 -->
<bean id="viewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <!-- JSP文件所在的目录 -->
    <property name="prefix" value="/pages/" />
    <!-- 文件的后缀名 -->
    <property name="suffix" value=".jsp" />
</bean>

    <!-- 设置静态资源不过滤 -->
    <mvc:resources location="/css/" mapping="/css/**" />
    <mvc:resources location="/img/" mapping="/img/**" />
    <mvc:resources location="/js/" mapping="/js/**" />
    <mvc:resources location="/plugins/" mapping="/plugins/**" />

    <!-- 开启对SpringMVC注解的支持 -->
    <mvc:annotation-driven />
    <!--
        支持AOP的注解支持，AOP底层使用代理技术
        JDK动态代理，要求必须有接口
        cglib代理，生成子类对象，proxy-target-class="true" 默认使用cglib的方式
    -->
    <aop:aspectj-autoproxy proxy-target-class="true"/>
</beans>
```

2.2.3.编写Controller

- ProductController

```
@Controller
@RequestMapping("/product")
public class ProductController {

    @Autowired
    private IProductService productService;

    @RequestMapping("/findAll.do")
    public ModelAndView findAll() {
        return null;
    }
}
```

2.3 Spring与Spring MVC整合

```
<!-- 配置加载类路径的配置文件 -->
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath*:applicationContext.xml,classpath*:spring-security.xml</param-value>
</context-param>
<!-- 配置监听器 -->
<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
```

2.4 Spring与MyBatis整合

2.4.1.整合思路

把 mybatis 配置文件（mybatis.xml）中内容配置到 spring 配置文件中 同时，把 mybatis 配置文件的内容清掉。

```
<?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>

</configuration>
```

注意： 理 由于我们使用的是代理 Dao ， 的模式，Dao 具体实现类由 MyBatis 使用代理方式创建，所以此时 mybatis 配置文件不能删。 当我们整合 spring 和 mybatis 时，mybatis 创建的 Mapper.xml 文件名必须和 Dao 接口 文件 名一致

2.4.2.Spring接管mybatis的Session工厂

db.properties

```
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/ssm?useUnicode=true&characterEncoding=utf8
jdbc.username=root
jdbc.password=root
```

```
<context:property-placeholder location="classpath:db.properties"/>
<!-- 配置连接池 -->
<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>
<!-- 把交给IOC管理 SqlSessionFactory -->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
</bean>
```

2.4.3.自动扫描所有Mapper接口和文件

```
<!-- 扫描dao接口 -->
<bean id="mapperScanner" class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage" value="com.itheima.ssm.dao"/>
</bean>
```

2.4.4.配置Spring事务

```
<!-- 配置Spring的声明式事务管理 -->
<!-- 配置事务管理器 -->
<bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource"/>
</bean>
<tx:annotation-driven transaction-manager="transactionManager"/>
```

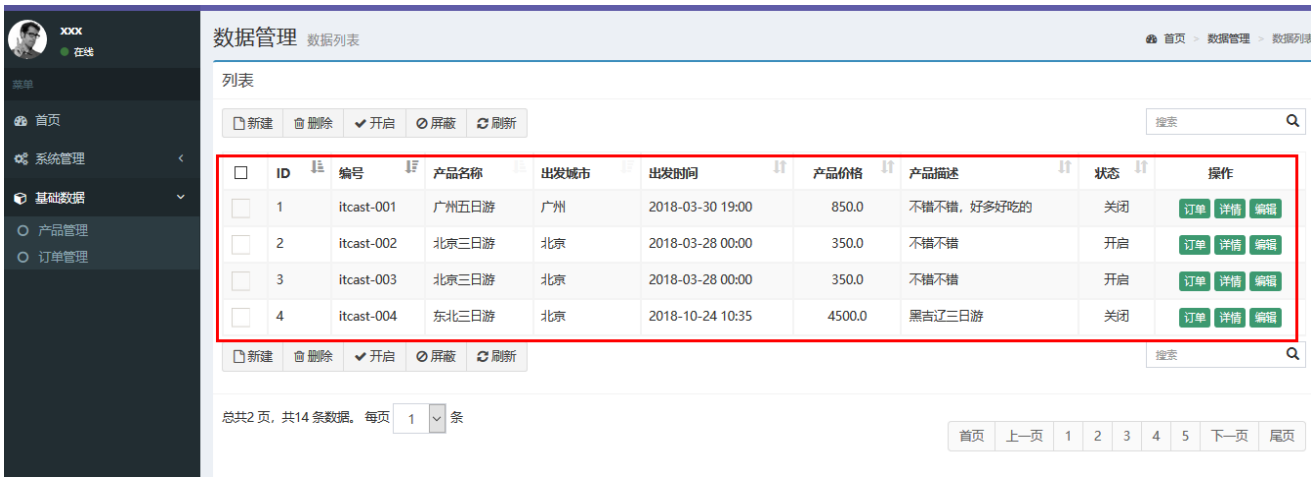
2.5 测试运行

2.5.1.编写jsp页面

2.5.1.1 请求发起页面 index.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
  pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>主页</title>
</head>
<body>
  <a href="${pageContext.request.contextPath}/product/findAll.do">查询产品信息</a>
</body>
</html>
```

2.5.1.2 显示产品页面 product-list.jsp



页面详细代码请查看今天课程资料

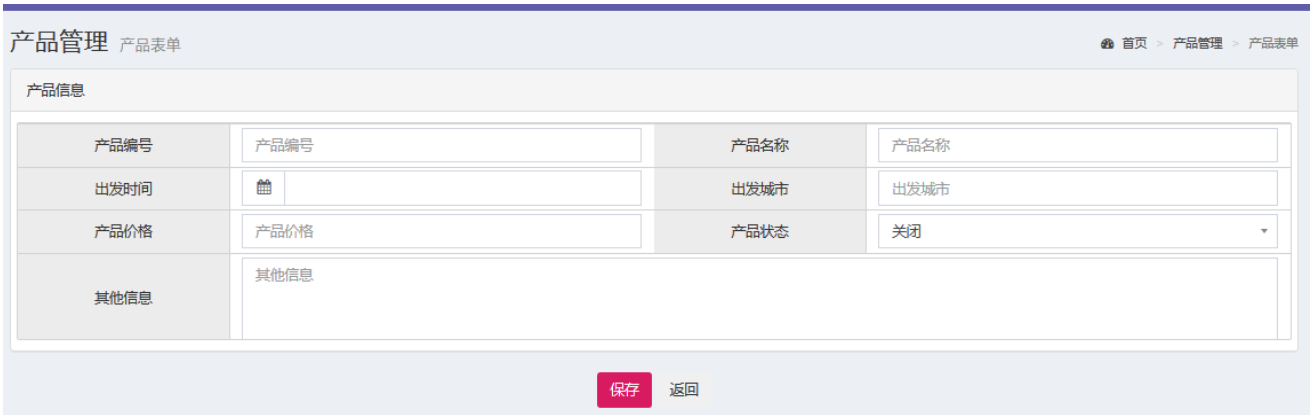
2.5.2.Controller

```
@Controller
@RequestMapping("/product")
public class ProductController {

    @Autowired
    private IProductService productService;

    @RequestMapping("/findAll.do")
    public ModelAndView findAll() throws Exception {
        ModelAndView mv = new ModelAndView();
        List<Product> products = productService.findAll();
        mv.addObject("productList", products);
        mv.setViewName("product-list");
        return mv;
    }
```

```
}
```

3.商品添加

3.1 商品添加页面 product-add.jsp



页面详细代码请查看今天课程资料

3.2 Controller

```
@Controller
@RequestMapping("/product")
public class ProductController {

    @Autowired
    private IProductService productService;

    @InitBinder
    public void initBinder(WebDataBinder binder) {
        binder.registerCustomEditor(Date.class, new MyDateEdit("yyyy-MM-dd HH:mm"));
    }


    @RequestMapping("/save.do")
    public String save(Product product) throws Exception {
        productService.save(product);
        return "redirect:findAll.do";
    }
    @RequestMapping("/findAll.do")
    public ModelAndView findAll() throws Exception {
        ModelAndView mv = new ModelAndView();
        List<Product> products = productService.findAll();
        mv.addObject("productList", products);
        mv.setViewName("product-list");

        return mv;
```

```
    }
}
```

3.3 Dao

```
public interface IProductDao {

    @Select("select * from product")
    List<Product> findAll() throws Exception;

    @Insert("insert into
product(productNum,productName,cityName,departureTime,productPrice,productDesc,productStatus)
values(#{productNum},#{productName},#{cityName},#{departureTime},#{productPrice},#
{productDesc},#{productStatus})")
    void save(Product product);
}
```

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。

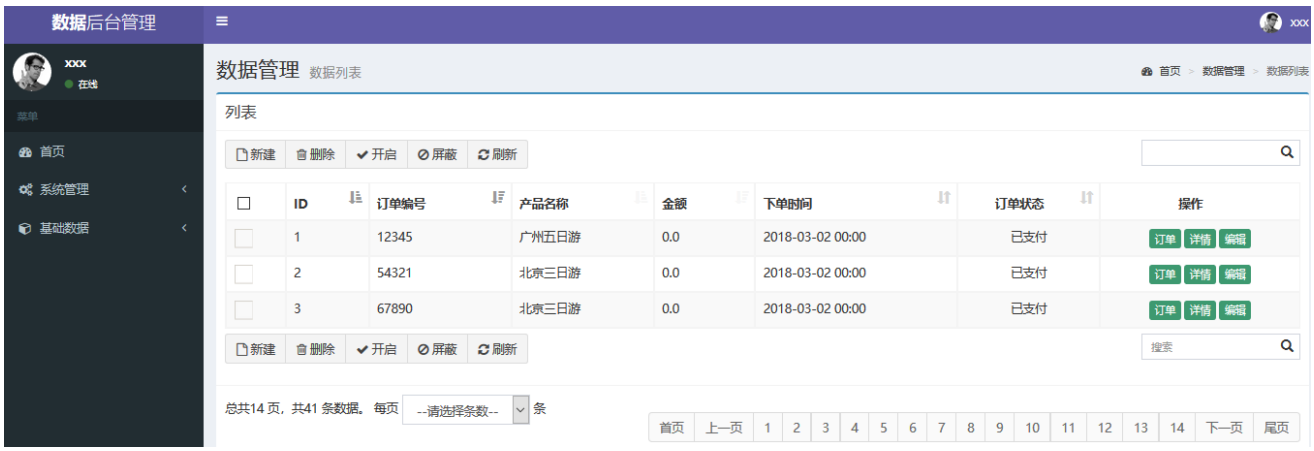
序号	字段名称	字段类型	字段描述
1	orderId	varchar2(32)	订单id
2	travellerId	varchar2(32)	旅客id

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

另一种方法是在sql语句中使用limit语句

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>
```

替换

```
<props>
    <prop key="helperDialect">oracle</prop>
    <prop key="reasonable">true</prop>
</props>
```

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

字符串

```
@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;
    }
```

用pageInfo Bean对象来作为信息来源

3.3.3 订单分页查询页面Order-list.jsp

pagehelper使用步骤

第一步 导入依赖

```
<dependency>
    <groupId>com.github.pagehelper</groupId>
    <artifactId>pagehelper</artifactId>
    <version>5.1.2</version>
</dependency>
```

第三步:在真下执行sql前，使用PageHelper来完成分页

```
//参数pageNum 是页码值    参数pageSize 代表是每页显示条数
PageHelper.startPage( pageNum: 1, pageSize: 5);
return ordersDao.findAll();
```

```
<!-- 把交给IOC管理 SqlSessionFactory -->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource"/>
    <!-- 传入PageHelper的插件 -->
    <property name="plugins">
        <array>
            <!-- 传入插件的对象 -->
            <bean class="com.github.pagehelper.PageInterceptor">
                <property name="properties">
                    <props>
                        <prop key="helperDialect">oracle</prop>
                        <prop key="reasonable">true</prop>
                    </props>
                </property>
            </bean>
        </array>
    </property>
</bean>
```

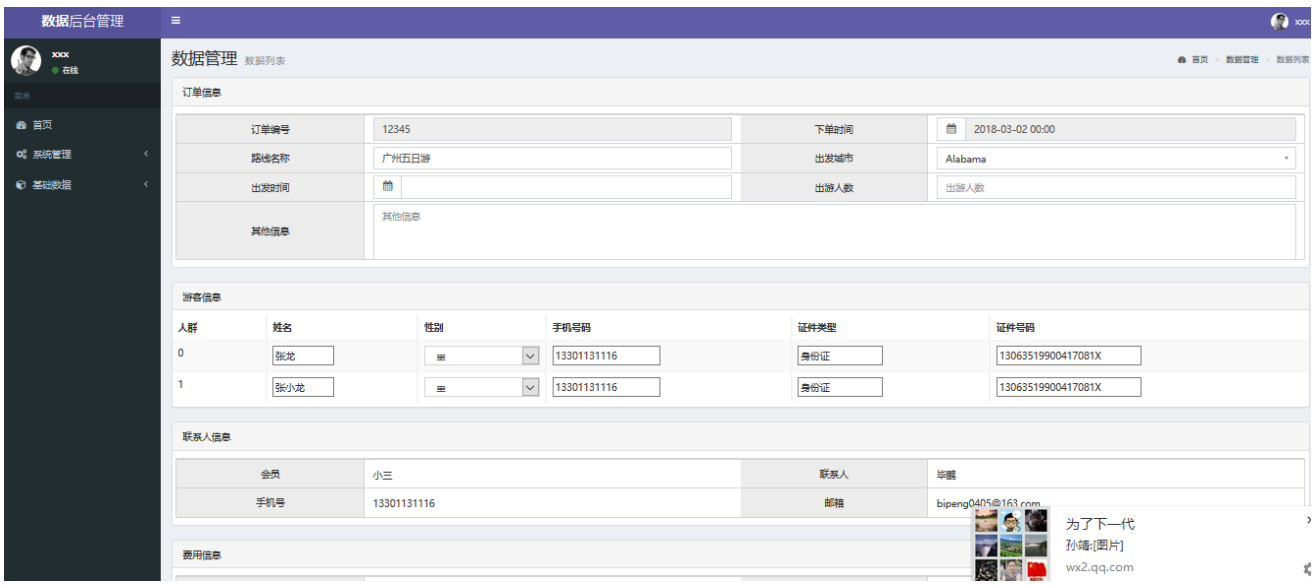
第二步：需要在spring配置文件中配置

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 inner join ORDER_TRAVELLER on id = travellerid where orderid = #{orderId}")

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

SSM权限操作

1.数据库与表结构

1.1 用户表

1.1.1 用户表信息描述users

序号	字段名称	字段类型	字段描述
1	id	varchar2	无意义，主键uuid
2	email	varchar2	非空，唯一
3	username	varchar2	用户名
5	password	varchar2	密码（加密）
6	phoneNum	varchar2	电话
7	status	int	状态0 未开启 1 开启

1.1.2 sql语句

```
CREATE TABLE users(
    id varchar2(32) default SYS_GUID() PRIMARY KEY,
    email VARCHAR2(50) UNIQUE NOT NULL,
    username VARCHAR2(50),
    PASSWORD VARCHAR2(50),
    phoneNum VARCHAR2(20),
    STATUS INT
)
```

1.1.3 实体类

```
public class UserInfo {

    private String id;
    private String username;
    private String email;
    private String password;
    private String phoneNum;
    private int status;
    private String statusStr;
    private List<Role> roles;

}
```

1.2 角色表

1.2.1 角色表信息描述role

序号	字段名称	字段类型	字段描述
1	id	varchar2	无意义，主键uuid
2	roleName	varchar2	角色名
3	roleDesc	varchar2	角色描述

1.2.2 sql语句

```
CREATE TABLE role(
    id varchar2(32) default SYS_GUID() PRIMARY KEY,
    roleName VARCHAR2(50) ,
    roleDesc VARCHAR2(50)
)
```

1.2.3 实体类

```
public class Role {

    private String id;
    private String roleName;
    private String roleDesc;
    private List<Permission> permissions;
    private List<User> users;
}
```

1.2.4 用户与角色关联关系

用户与角色之间是多对多关系，我们通过user_role表来描述其关联，在实体类中User中存在List，在Role中有List.
而角色与权限之间也存在关系，我们会在后面介绍。

```
CREATE TABLE users_role(
    userId varchar2(32),
    roleId varchar2(32),
    PRIMARY KEY(userId,roleId),
    FOREIGN KEY (userId) REFERENCES users(id),
    FOREIGN KEY (roleId) REFERENCES role(id)

)
```

1.3 资源权限表

1.3.1 权限资源表描述permission

序号	字段名称	字段类型	字段描述
1	id	varchar2	无意义，主键uuid
2	permissionName	varchar2	权限名
3	url	varchar2	资源路径

1.3.2 sql语句

```
CREATE TABLE permission(
    id varchar2(32) default SYS_GUID() PRIMARY KEY,
    permissionName VARCHAR2(50) ,
    url VARCHAR2(50)
)
```

1.3.3 实体类

```
public class Permission {

    private String id;
    private String permissionName;
    private String url;
    private List<Role> roles;
}
```

1.3.4.权限资源与角色关联关系

权限资源与角色是多对多关系，我们使用role_permission表来描述。在实体类Permission中存在List,在Role类中有List

```
CREATE TABLE role_permission(
    permissionId varchar2(32),
    roleId varchar2(32),
    PRIMARY KEY(permissionId,roleId),
    FOREIGN KEY (permissionId) REFERENCES permission(id),
    FOREIGN KEY (roleId) REFERENCES role(id)
)
```

2.Spring Security概述

2.1 Spring Security介绍

Spring Security 的前身是 Acegi Security ，是 Spring 项目组中用来提供安全认证服务的框架。(https://projects.spring.io/spring-security/) Spring Security 为基于J2EE企业应用软件提供了全面安全服务。特别是使用领先的J2EE解决方案-Spring框架开发的企业软件项目。人们使用Spring Security有很多种原因，不过通常吸引他们的是在J2EE Servlet规范或EJB规范中找不到典型企业应用场景的解决方案。 特别要指出的是他们不能再 WAR 或 EAR 级别进行移植。这样，如果你更换服务器环境，就要，在新的目标环境进行大量的工作，对你的应用系统进行重新配 置安全。使用Spring Security 解决了这些问题，也为你提供很多有用的，完全可以指定的其他安全特性。 安全包括两个主要操作。

- “认证”，是为用户建立一个他所声明的主体。主题一般式指用户，设备或可以在你系 统中执行动作的其他系统。
- “授权”指的是一个用户能否在你的应用中执行某个操作，在到达授权判断之前，身份的主题已经由 身份验证过程建立了。

这些概念是通用的，不是Spring Security特有的。在身份验证层面，Spring Security广泛支持各种身份验证模式，这些验证模型绝大多数都由第三方提供，或则正在开发的有关标准机构提供的，例如 Internet Engineering Task Force.作为补充，Spring Security 也提供了自己的一套验证功能。

Spring Security 目前支持认证一体化如下认证技术： HTTP BASIC authentication headers (一个基于IEFT RFC 的标准) HTTP Digest authentication headers (一个基于IEFT RFC 的标准) HTTP X.509 client certificate exchange (一个基于IEFT RFC 的标准) LDAP (一个非常常见的跨平台认证需要做法，特别是在大环境) Form-based authentication (提供简单用户接口的需求) OpenID authentication Computer Associates Siteminder JA-SIG Central Authentication Service (CAS，这是一个流行的开源单点登录系统) Transparent authentication context propagation for Remote Method Invocation and HttpInvoker (一个Spring远程调用协议)

Maven依赖

```
<dependencies>
    <dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-web</artifactId>
        <version>5.0.1.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-config</artifactId>
        <version>5.0.1.RELEASE</version>
    </dependency>
</dependencies>
```

2.2 Spring Security快速入门

2.2.1 pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
```

```
instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>spring_security_demo</groupId>
    <artifactId>SpringSecurity_quickStart</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>war</packaging>
    <properties>
        <spring.version>5.0.2.RELEASE</spring.version>
        <spring.security.version>5.0.1.RELEASE</spring.security.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-web</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context-support</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-test</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-jdbc</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-web</artifactId>
            <version>${spring.security.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-config</artifactId>
            <version>${spring.security.version}</version>

        </dependency>
```

```
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>3.1.0</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <!-- java编译插件 -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.2</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                    <encoding>UTF-8</encoding>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.tomcat.maven</groupId>
                <artifactId>tomcat7-maven-plugin</artifactId>
                <configuration>
                    <!-- 指定端口 -->
                    <port>8080</port>
                    <!-- 请求路径 -->
                    <path>/</path>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

2.2.2 web.xml

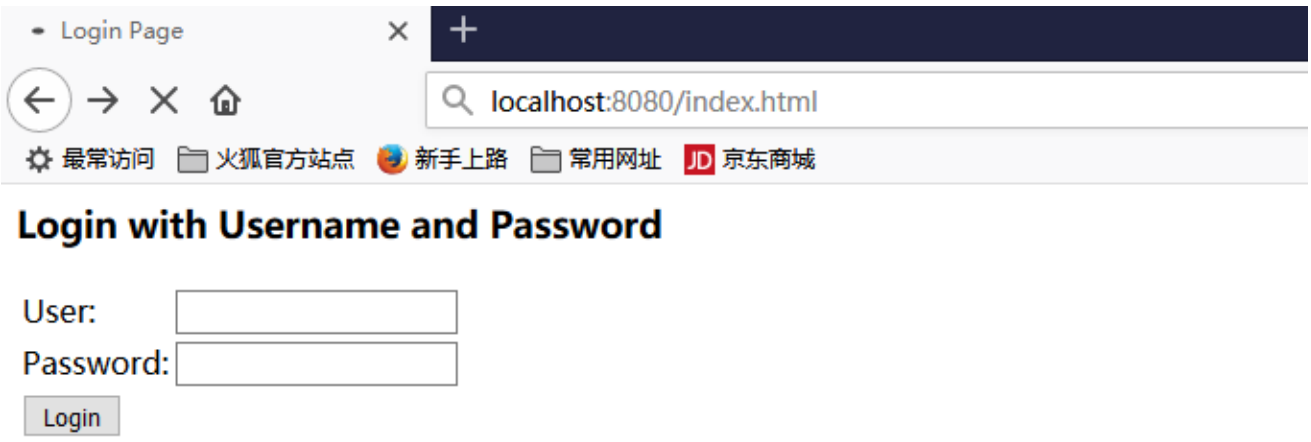
```
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:spring-security.xml</param-value>
    </context-param>
    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>
    <filter>
        <filter-name>springSecurityFilterChain</filter-name>
        <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>springSecurityFilterChain</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

2.2.3 spring security配置

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:security="http://www.springframework.org/schema/security"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/security
    http://www.springframework.org/schema/security/spring-security.xsd">
    <security:http auto-config="true" use-expressions="false">
        <!-- intercept-url定义一个过滤规则 pattern表示对哪些url进行权限控制，ccess属性表示在请求对应
的URL时需要什么权限，
            默认配置时它应该是一个以逗号分隔的角色列表，请求的用户只需拥有其中的一个角色就能成功访问对应
的URL -->
        <security:intercept-url pattern="/**" access="ROLE_USER" />
        <!-- auto-config配置后，不需要在配置下面信息 <security:form-login /> 定义登录表单信息
<security:http-basic
            /> <security:logout /> -->
    </security:http>
    <security:authentication-manager>
        <security:authentication-provider>
            <security:user-service>
                <security:user name="user" password="{noop}user"
                    authorities="ROLE_USER" />
                <security:user name="admin" password="{noop}admin"
                    authorities="ROLE_ADMIN" />
            </security:user-service>
        </security:authentication-provider>
    </security:authentication-manager>
</beans>
```

2.2.4 测试

我们在webapp下创建一个index.html页面，在页面中任意写些内容。



当我们访问index.html页面时发现会弹出登录窗口，可能你会奇怪，我们没有建立下面的登录页面，为什么Spring Security会跳到上面的登录页面呢？这是我们设置http的auto-config="true"时Spring Security自动为我们生成的。

2.2.5 使用自定义页面

2.2.5.1 spring-security.xml配置

```
 <?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:security="http://www.springframework.org/schema/security"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans.xsd
          http://www.springframework.org/schema/security
          http://www.springframework.org/schema/security/spring-security.xsd">

    <!-- 配置不过滤的资源（静态资源及登录相关） -->
    <security:http security="none" pattern="/login.html" />
    <security:http security="none" pattern="/failer.html" />

    <security:http auto-config="true" use-expressions="false">
        <!-- 配置资料连接，表示任意路径都需要ROLE_USER权限 -->
        <security:intercept-url pattern="/**" access="ROLE_USER" />
        <!-- 自定义登陆页面，login-page 自定义登陆页面 authentication-failure-url 用户权限校验失败之
后才会跳转到这个页面，如果数据库中没有这个用户则不会跳转到这个页面。
            default-target-url 登陆成功后跳转的页面。 注：登陆页面用户名固定 username，密码
password，action:login -->
        <security:form-login login-page="/login.html"
            login-processing-url="/login" username-parameter="username"
            password-parameter="password" authentication-failure-url="/failer.html"
            default-target-url="/success.html"
            />
        <!-- 登出， invalidate-session 是否删除session logout-url：登出处理链接 logout-success-
url：登出成功页面
            注：登出操作 只需要链接到 logout即可登出当前用户 -->
        <security:logout invalidate-session="true" logout-url="/logout"
            logout-success-url="/login.jsp" />
        <!-- 关闭CSRF,默认是开启的 -->
        <security:csrf disabled="true" />
    </security:http>

    <security:authentication-manager>
        <security:authentication-provider>
            <security:user-service>
                <security:user name="user" password="{noop}user"
                    authorities="ROLE_USER" />
                <security:user name="admin" password="{noop}admin"
                    authorities="ROLE_ADMIN" />
            </security:user-service>
        </security:authentication-provider>
    </security:authentication-manager>
</beans>
```

2.2.5.2 login.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<form action="login" method="post">
  <table>
      <tr>
          <td>用户名：</td>
          <td><input type="text" name="username" /></td>
      </tr>
      <tr>
          <td>密码：</td>
          <td><input type="password" name="password" /></td>
      </tr>
      <tr>
          <td colspan="2" align="center"><input type="submit" value="登录" />
              <input type="reset" value="重置" /></td>
      </tr>
  </table>
</form>
</body>
</html>
```

2.2.5.3 success.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
  success html<br>
  <a href="logout">退出</a>
</body>
</html>
```

2.2.5.4 failer.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>登录失败
</body>
</html>
```

2.3 Spring Security使用数据库认证

在Spring Security中如果想要使用数据进行认证操作，有很多种操作方式，这里我们介绍使用UserDetails、UserDetailsService来完成操作。

- UserDetails

```
public interface UserDetails extends Serializable {
    Collection<? extends GrantedAuthority> getAuthorities();
    String getPassword();
    String getUsername();
    boolean isAccountNonExpired();
    boolean isAccountNonLocked();
    boolean isCredentialsNonExpired();
    boolean isEnabled();
}
```

UserDetails是一个接口，我们可以认为UserDetails作用是于封装当前进行认证的用户信息，但由于其是一个接口，所以我们可以对其进行实现，也可以使用Spring Security提供的一个UserDetails的实现类User来完成操作

以下是User类的部分代码

```
public class User implements UserDetails, CredentialsContainer {

    private String password;
    private final String username;
    private final Set<GrantedAuthority> authorities;
    private final boolean accountNonExpired; //帐户是否过期
    private final boolean accountNonLocked; //帐户是否锁定
    private final boolean credentialsNonExpired; //认证是否过期
    private final boolean enabled; //帐户是否可用

}
```

- UserDetailsService

```
public interface UserDetailsService {

    UserDetails loadUserByUsername(String username) throws UsernameNotFoundException;
}
```

上面将UserDetails与UserDetailsService做了一个简单的介绍，那么我们具体如何完成Spring Security的数据库认证操作哪，我们通过用户管理中用户登录来完成Spring Security的认证操作。

3.用户管理

3.1 用户登录

看项目代码

spring security的配置

```
<security:authentication-manager>
       <security:authentication-provider user-service-ref="userService">
           <!-- 配置加密的方式
           <security:password-encoder ref="passwordEncoder"/>
           -->
       </security:authentication-provider>
   </security:authentication-manager>
```

3.1.1.登录页面login.jsp

详细页面请查看资料

3.1.2.Service

```
public interface IUserService extends UserDetailsService{


}
```

```
@Service("userService")
@Transactional
public class UserServiceImpl implements IUserService {

    @Autowired
    private IUserDao userDao;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        UserInfo userInfo = userDao.findByUsername(username);
        List<Role> roles = userInfo.getRoles();

        List<SimpleGrantedAuthority> authoritys = getAuthority(roles);
```

```
        User user = new User(userInfo.getUsername(), "{noop}" + userInfo.getPassword(),
                userInfo.getStatus() == 0 ? false : true, true, true, true, authoritys);

        return user;
    }

    private List<SimpleGrantedAuthority> getAuthority(List<Role> roles) {
        List<SimpleGrantedAuthority> authoritys = new ArrayList();
        for (Role role : roles) {
            authoritys.add(new SimpleGrantedAuthority(role.getRoleName()));
        }
        return authoritys;
    }
}
```

3.1.3.IUserDao

```
public interface IUserDao {

    @Select("select * from user where id=#{id}")
    public UserInfo findById(Long id) throws Exception;

    @Select("select * from user where username=#{username}")
    @Results({
              @Result(id = true, property = "id", column = "id"),
              @Result(column = "username", property = "username"),
              @Result(column = "email", property = "email"),
              @Result(column = "password", property = "password"),
              @Result(column = "phoneNum", property = "phoneNum"),
              @Result(column = "status", property = "status"),
              @Result(column = "id", property = "roles", javaType = List.class, many =
                       @Many(select = "com.itheima.ssm.dao.IRoleDao.findRoleByUserId")) })
    public UserInfo findByUsername(String username);
    }
```

3.2 用户退出

使用spring security完成用户退出，非常简单

- 配置

```
<security:logout invalidate-session="true" logout-url="/logout.do" logout-success-
url="/login.jsp" />
```

- 页面中

```
<a href="${pageContext.request.contextPath}/logout.do"
                                    class="btn btn-default btn-flat">注销</a>
```

配置 web.Xml

```
<!--SpringSecurity 的过滤器-->
<filter>
    <filter-name>springSecurityFilterChain</filter-name>
    <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>
<filter-mapping>
    <filter-name>springSecurityFilterChain</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>

<!—添加 spring-security 配置文件的路径-->
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:applicationContext.xml,classpath:spring-security.xml</param-value>
</context-param>
```

导入 spring-security.xml 配置文件

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:security="http://www.springframework.org/schema/security"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/security
    http://www.springframework.org/schema/security/spring-security.xsd">

    <!-- 配置不拦截的资源 -->
    <security:http pattern="/login.jsp" security="none"/>
    <security:http pattern="/failer.jsp" security="none"/>
    <security:http pattern="/css/**" security="none"/>
    <security:http pattern="/img/**" security="none"/>
    <security:http pattern="/plugins/**" security="none"/>

    <!--
       配置具体的规则
       auto-config="true" 不用自己编写登录的页面，框架提供默认登录页面
       use-expressions="false"     是否使用 SPEL 表达式（没学习过）
    -->
    <security:http auto-config="true" use-expressions="false">
       <!-- 配置具体的拦截的规则 pattern="请求路径的规则" access="访问系统的人，必须有 ROLE_USER 的角色" -->
       <security:intercept-url pattern="/**" access="ROLE_USER,ROLE_ADMIN"/>

       <!-- 定义跳转的具体的页面 -->
       <security:form-login
          login-page="/login.jsp"
```

```
        login-processing-url="/login.do"
        default-target-url="/index.jsp"
        authentication-failure-url="/failer.jsp"
       authentication-success-forward-url="/pages/main.jsp"
        <!—
        默认就是 username 和 password，不用写
        username-parameter="username"
        password-parameter="password"
              -->
    />

    <!-- 关闭跨域请求 -->
    <security:csrf disabled="true"/>

    <!-- 退出 -->
    <security:logout invalidate-session="true" logout-url="/logout.do"
logout-success-url="/login.jsp" />

</security:http>

<!-- 切换成数据库中的用户名和密码 -->
<security:authentication-manager>
    <security:authentication-provider user-service-ref="userService">
        <!-- 配置加密的方式 -->
      <!--<security:password-encoder ref="passwordEncoder"/>-->
    </security:authentication-provider>
</security:authentication-manager>

<!-- 配置加密类 -->
<bean id="passwordEncoder"
class="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder"/>

<!-- 提供了入门的方式，在内存中存入用户名和密码
<security:authentication-manager>
    <security:authentication-provider>
        <security:user-service>
            <security:user name="admin" password="{noop}admin" authorities="ROLE_USER"/>
        </security:user-service>
    </security:authentication-provider>
</security:authentication-manager>
-->

</beans>
```

domain 层

创建 UserInfo、Role、Permission

```
//与数据库中 users 对应
```

```java
public class UserInfo {
    private String id;
    private String username;
    private String email;
    private String password;
    private String phoneNum;
    private int status;
    private String statusStr;
    private List<Role> roles;
}

public class Permission {
    private String id;
    private String permissionName;
    private String url;
    private List<Role> roles;
}

public class Role {
    private String id;
    private String roleName;
    private String roleDesc;
    private List<Permission> permissions;
    private List<UserInfo> users;
}
```

service 层

IUserService 接口——继承 UserDetailsService

```java
import org.springframework.security.core.userdetails.UserDetailsService;

public interface IUserService extends UserDetailsService {
}
```

UserServiceImpl 实现方法——包装 UserInfo

```java
@Service("userService")
@Transactional(readOnly = true, propagation = Propagation.SUPPORTS)
public class UserServiceImpl implements IUserService {
    @Autowired
    private IUsersDao userDao;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        UserInfo userInfo = userDao.findByUsername(username);
        User user = new User(userInfo.getUsername(), "{noop}"+userInfo.getPassword(),
                userInfo.getStatus() == 1,true, true,
                true,getAuthorities(userInfo.getRoles()));
```

```
        return user;
    }

    private List<SimpleGrantedAuthority> getAuthorities(List<Role> roles) {
        List<SimpleGrantedAuthority> authoritys = new ArrayList<>();
        for (Role role : roles) {
            authoritys.add(new SimpleGrantedAuthority("ROLE_" + role.getRoleName()));
        }
        return authoritys;
    }
}
```

dao 层

IUsersDao

```
@Repository
public interface IUsersDao {

    @Select("select * from users where username = #{username}")
    @Results(id = "userInfoMap",
            value = {
                    @Result(id = true, column = "id", property = "id"),
                    @Result(column = "username", property = "username"),
                    @Result(column = "email", property = "email"),
                    @Result(column = "password", property = "password"),
                    @Result(column = "phoneNum", property = "phoneNum"),
                    @Result(column = "status", property = "status"),
                    @Result(column = "id", property = "roles", javaType = List.class,
                            many = @Many(select = "com.don.ssm.dao.IRoleDao.findRolesByUserId",
fetchType = FetchType.LAZY))

            }

    )
    UserInfo findByUsername(String username);

}
```

IRoleDao

```
@Repository
public interface IRoleDao {

    @Select("SELECT r.* from role r inner join users_role on roleid = r.id where userid = #{userId}
")
    List<Role> findRolesByUserId(String userId);
}
```

3.3 用户查询

3.3.1.用户查询页面 user-list.jsp

请在资料中查看具体代码

3.3.2.UserController

```
@Controller
@RequestMapping("/user")
public class UserControlller {

    @RequestMapping("/findAll.do")
    public ModelAndView findAll() throws Exception {
        List<UserInfo> users = userService.findAll();
        ModelAndView mv = new ModelAndView();
        mv.addObject("userlist", users);
        mv.setViewName("user-list");
        return mv;
    }
}
```

3.3.3.Dao

```
    @Select("select * from user")
    public List<UserInfo> findAll();
```

3.4 用户添加

3.4.1.用户添加页面 user-add.jsp

请在资料中查看具体页面代码

3.4.2.UserController

```
@Controller
@RequestMapping("/user")
public class UserControlller {
    @Autowired
    private IUserService userService;

    @RequestMapping("/save.do")
    public String save(UserInfo user) throws Exception {
        userService.save(user);
        return "redirect:findAll.do";
    }
    }
```

3.4.3.Service

要去掉{noop}，需要解密

```
@Service("userService")
@Transactional
public class UserServiceImpl implements IUserService {

    @Autowired
    private IUserDao userDao;

    @Autowired
    private PasswordEncoder passwordEncoder;

    @Override
    public void save(UserInfo user) throws Exception {
        user.setPassword(passwordEncoder.encode(user.getPassword()));
        userDao.save(user);
    }
}
```

前期我们的用户密码没有加密，现在添加用户时，我们需要对用户密码进行加密

```
 <!-- 配置加密类 -->
    <bean id="passwordEncoder"
class="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder"/>
```

3.4. 4.Dao

```
        @Insert("insert into user(email,username,password,phoneNum,status) value(#{email},#
{username},#{password},#{phoneNum},#{status})")
        public void save(UserInfo user) throws Exception;
```

3.5 用户详情

3.5.1.用户详情页面user-show.jsp

请在资料中查看页面详细代码

注意：需要添加js

```
$("#collapse-table").treetable({ expandable : true      });
```

3.5.2.UserController

```
@Controller
@RequestMapping("/user")
public class UserControlller {

        @Autowired
        private IUserService userService;
        @RequestMapping("/findById.do")
        public ModelAndView findById(@RequestParam(name = "id", required = true) Long id) throws
Exception {
                UserInfo user = userService.findById(id);
                ModelAndView mv = new ModelAndView();
                mv.addObject("user", user);
                mv.setViewName("user-show");
                return mv;
        }
        }
```

控制一下参数

3.5.3.Dao

```
@Select("select * from user where id=#{id}")
@Results({ @Result(id = true, property = "id", column = "id"), @Result(column = "username",
property = "username"),
                        @Result(column = "email", property = "email"), @Result(column =
"password", property = "password"),
                        @Result(column = "phoneNum", property = "phoneNum"), @Result(column =
"status", property = "status"),
                        @Result(column = "id", property = "roles", javaType = List.class, many =
@Many(select = "com.itheima.ssm.dao.IRoleDao.findRoleByUserId")) })
        public UserInfo findById(Long id) throws Exception;
```

```
@Select("select * from role where id in( select roleId from user_role where userId=#{userId})")
        @Results(
                        {
                                @Result(id=true,column="id",property="id"),
                                @Result(column="roleName",property="roleName"),
                                @Result(column="roleDesc",property="roleDesc"),

@Result(column="id",property="permissions",javaType=List.class,many=@Many(select="com.itheima.ssm
.dao.IPermissionDao.findByRoleId"))
                        })
        public List<Role> findRoleByUserId(Long userId);
```

我们需要将用户的所有角色及权限查询出来所以需要调用IRoleDao中的findRoleByUserId,而在IRoleDao中需要调用IPermissionDao的findByRoleId

```
@Select("select * from permission where id in (select permissionId from role_permission where
roleId=#{roleId})")
        public List<Permission> findByRoleId(Long roleId);
```

4.角色管理

4.1 角色查询

4.1.1.角色页面role-list.jsp

请在资料中查看页面详细代码

4.1.2.RoleControlller

```
@RequestMapping("/role")
@Controller
public class RoleController {

        @Autowired
        private IRoleService roleService;

        @RequestMapping("/findAll.do")
        public ModelAndView findAll() throws Exception {
                List<Role> roleList = roleService.findAll();
                ModelAndView mv = new ModelAndView();
                mv.addObject("roleList", roleList);
                mv.setViewName("role-list");
                return mv;
        }
}
```

4.1.3.Dao

```
        @Select("select * from role")
        public List<Role> findAll();
```

4.2 角色添加

4.2.1.角色添加页面role-add.jsp

请在页面中查看详细代码

4.2.2.RoleControlller

```
@RequestMapping("/role")
@Controller
public class RoleController {

        @Autowired
        private IRoleService roleService;

        @RequestMapping("/save.do")
        public String save(Role role) {
                roleService.save(role);

                return "redirect:findAll.do";
        }
}
```

4.2.3.Dao

```
@Insert("insert into role(roleName,roleDesc) value(#{roleName},#{roleDesc})")
        public void save(Role role);
```

5.资源权限管理

5.1 资源权限查询

5.1.1.权限资源页面permission-list.jsp

请在资料中查看详细代码

5.1.2.PermissionController

```
@RequestMapping("/permission")
@Controller
public class PermissionController {
```

```
        @Autowired
        private IPermissionService permissionService;

        @RequestMapping("/findAll.do")
        public ModelAndView findAll() throws Exception {
                List<Permission> permissionList = permissionService.findAll();
                ModelAndView mv = new ModelAndView();
                mv.addObject("permissionList", permissionList);
                mv.setViewName("permission-list");
                return mv;
        }
}
```

5.1.3.Dao

```
        @Select("select * from permission")
        public List<Permission> findAll();
```

5.2 资源权限添加

5.2.1.权限资源添加页面permission-add.jsp

请在资料中查看页面详细代码

5.2.2.PermissionController

```
@RequestMapping("/permission")
@Controller
public class PermissionController {

        @Autowired
        private IPermissionService permissionService;

        @RequestMapping("/save.do")
        public String save(Permission p) throws Exception {
                permissionService.save(p);
                return "redirect:findAll.do";
        }
}
```

5.2.3.Dao

```
        @Insert("insert into permission(permissionName,url) value(#{permissionName},#{url})")
        public void save(Permission p);
```

6.权限关联与控制

6.1 用户角色关联

用户与角色之间是多对多关系，我们要建立它们之间的关系，只需要在中间表user_role插入数据即可。

6.1.1. 用户角色关联相关页面

- 在user-list.jsp页面上添加链接

```
<a href="${pageContext.request.contextPath}/user/findUserByIdAndAllRole.do?id=${user.id}"
class="btn bg-olive btn-xs">添加角色</a>
```

- 展示可以添加角色的页面user-roe-add.jsp

请在资料中查看页面详细代码

6.1.2.UserController

- findUserByIdAndAllRole(Long id)方法

  此方法用于查找要操作的用户及可以添加的角色，参数是要操作的用户id

  ```
  @RequestMapping("/findUserByIdAndAllRole.do")
  public ModelAndView findUserByIdAndAllRole(Long id) throws Exception {
                  UserInfo user = userService.findById(id);
                  List<Role> roleList = roleService.findOtherRole(id);
                  ModelAndView mv = new ModelAndView();
                  mv.addObject("user", user);
                  mv.addObject("roleList", roleList);
                  mv.setViewName("user-role-add");
                  return mv;
          }
  ```

  调用IUserService的findById方法获取要操作的User

  调用IRoleService的findOtherRole方法用于获取可以添加的角色信息

- addRoleToUser(Long userId,Long[] ids)方法

  些方法用于在用户与角色之间建立关系，参数userId代表要操作的用户id,参数ids代表的是角色id数组

  ```
  @RequestMapping("/addRoleToUser.do")
          public String addRoleToUser(Long userId, Long[] ids) throws Exception {
                  userService.addRoleToUser(userId,ids);
                  return "redirect:findAll.do";
          }
  ```

6.1.3.Dao

- IRoleDao

```
@Select("select * from role where id not in( select roleId from user_role where userId=#
{id})")
public List<Role> findOtherRole(Long id);
```

用于查找可以添加的角色

- IUserDao

```
@Insert("insert into user_role(userId,roleId) value(#{userId},#{roleId})")
public void addRoleToUser(@Param("userId") Long userId, @Param("roleId") Long roleId);
```

用于添加用户与角色关系

6.2 角色权限关联

角色与权限之间是多对多关系，我们要建立它们之间的关系，只需要在中间表role_permission插入数据即可。

6.2.1. 角色权限关联相关页面

- 在role-list.jsp页面上添加链接

```
        <a href="${pageContext.request.contextPath}/role/findRoleByIdAndAllPermission.do?
id=${role.id}" class="btn bg-olive btn-xs">添加权限</a>
```

- 展示可以添加权限的页面roe-permission-add.jsp

请在资料中查看页面详细代码

6.2.2.RoleController

- findRoleByIdAndAllPermission(Long roleId)方法

  此方法用于查找要操作的角色及可以添加的权限，参数是要操作的角色id

```
@RequestMapping("/findRoleByIdAndAllPermission.do")
        public ModelAndView findRoleByIdAndAllPermission(@RequestParam(name = "id", required
= true) Long roleid)
                        throws Exception {
                ModelAndView mv = new ModelAndView();
                Role role = roleService.findById(roleid);
                mv.addObject("role", role);
                List<Permission> permissionList =
permissionService.findOtherPermission(roleid);
                mv.addObject("permissionList", permissionList);
                mv.setViewName("role-permission-add");
                return mv;
        }
```

调用IRoleService的findById方法获取要操作的Role

调用IPermissionService的findOtherPermission方法用于获取可以添加的权限信息

- addPermissionToRole(Long roleId,Long[] ids)方法

  些方法用于在角色与权限之间建立关系，参数roleId代表要操作的角色id,参数permissionIds代表的是权限id数组

```
@RequestMapping("/addPermissionToRole.do")
public String addPermissionToRole(@RequestParam(name = "roleId") Long roleId,
                @RequestParam(name = "ids") Long[] permissionIds) throws Exception {
                roleService.addPermissionToRole(roleId, permissionIds);
                return "redirect:findAll.do";
}
```

6.2.3.Dao

- IPermissionDao

```
@Select("select * from permission where id not in (select permissionId from role_permission
where roleId=#{roleId})")
public List<Permission> findOtherPermission(Long roleid);
```

  用于查找可以添加的权限

- IRoleDao

```
@Insert("insert into role_permission (roleId,permissionId) value (#{roleId},#
{permissionId})")
public void addPermissionToRole(@Param("roleId") Long roleId, @Param("permissionId") Long
permissionId);
```

  用于绑定角色与权限的关系

6.3 服务器端方法级权限控制

在服务器端我们可以通过Spring security提供的注解对方法来进行权限控制。Spring Security在方法的权限控制上支持三种类型的注解，JSR-250注解、@Secured注解和支持表达式的注解，这三种注解默认都是没有启用的，需要单独通过global-method-security元素的对应属性进行启用

6.3.1.开启注解使用

- 配置文件
  <security:global-method-security jsr250-annotations="enabled"/>
  <security:global-method-security secured-annotations="enabled"/>
  <security:global-method-security pre-post-annotations="disabled"/>
- 注解开启
  @EnableGlobalMethodSecurity ： Spring Security默认是禁用注解的，要想开启注解，需要在继承WebSecurityConfigurerAdapter的类上加@EnableGlobalMethodSecurity注解，并在该类中将AuthenticationManager定义为Bean。

6.3.2.JSR-250注解

- @RolesAllowed表示访问对应方法时所应该具有的角色

  示例：
  @RolesAllowed({"USER", "ADMIN"}) 该方法只要具有"USER", "ADMIN"任意一种权限就可以访问。这里可以省略前缀ROLE_，实际的权限可能是ROLE_ADMIN

- @PermitAll表示允许所有的角色进行访问，也就是说不进行权限控制
- @DenyAll是和PermitAll相反的，表示无论什么角色都不能访问

6.3.3.支持表达式的注解

- @PreAuthorize 在方法调用之前,基于表达式的计算结果来限制对方法的访问

  示例：
  ```
  @PreAuthorize("#userId == authentication.principal.userId or hasAuthority('ADMIN')")
  void changePassword(@P("userId") long userId ){ }
  ```
  这里表示在changePassword方法执行之前，判断方法参数userId的值是否等于principal中保存的当前用户的userId，或者当前用户是否具有ROLE_ADMIN权限，两种符合其一，就可以访问该方法。

- @PostAuthorize 允许方法调用,但是如果表达式计算结果为false,将抛出一个安全性异常

  示例：
  ```
  @PostAuthorize
  User getUser("returnObject.userId == authentication.principal.userId or
  hasPermission(returnObject, 'ADMIN')");
  ```

- @PostFilter 允许方法调用,但必须按照表达式来过滤方法的结果
- @PreFilter 允许方法调用,但必须在进入方法之前过滤输入值

6.3.4.@Secured注解

- @Secured注解标注的方法进行权限控制的支持，其值默认为disabled。

示例：
```
@Secured("IS_AUTHENTICATED_ANONYMOUSLY")
public Account readAccount(Long id);
@Secured("ROLE_TELLER")
```

6.4 页面端标签控制权限

在jsp页面中我们可以使用spring security提供的权限标签来进行权限控制

6.4.1.导入

- maven导入

```
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-taglibs</artifactId>
  <version>version</version>
</dependency>
```

- 页面导入

```
<%@taglib uri="http://www.springframework.org/security/tags" prefix="security"%>
```

6.4.2.常用标签

在jsp中我们可以使用以下三种标签，其中authentication代表的是当前认证对象，可以获取当前认证对象信息，例如用户名。其它两个标签我们可以用于权限控制

6.4.2.1 authentication

```
<security:authentication property="" htmlEscape="" scope="" var=""/>
```

- property： 只允许指定Authentication所拥有的属性，可以进行属性的级联获取，如“principle.username”，不允许直接通过方法进行调用
- htmlEscape：表示是否需要将html进行转义。默认为true。
- scope：与var属性一起使用，用于指定存放获取的结果的属性名的作用范围，默认我pageContext。Jsp中拥有的作用范围都进行进行指定
- var： 用于指定一个属性名，这样当获取到了authentication的相关信息后会将其以var指定的属性名进行存放，默认是存放在pageConext中

修改标签或者加入bean

```
<security:http auto-config="true" use-expressions="true">
 <!-- 配置具体的拦截的规则 pattern="请求路径的规则" access="访问系统的人，必须有ROLE_USER的角色" -->
<security:intercept-url pattern="/**" access="hasAnyRole('ROLE_USER','ROLE_ADMIN')"/>
```

6.4.2.2 authorize

需要修改spring-security文件

authorize是用来判断普通权限的，通过判断用户是否具有对应的权限而控制其所包含内容的显示

```
<security:authorize access="" method="" url="" var=""></security:authorize>
```

```
<bean id="webexpressionHandler" class="org.springframework.security.web.access.expression.DefaultWebSecurityExpressionHandler" />
```

- access： 需要使用表达式来判断权限，当表达式的返回结果为true时表示拥有对应的权限
- method：method属性是配合url属性一起使用的，表示用户应当具有指定url指定method访问的权限，method的默认值为GET，可选值为http请求的7种方法
- url：url表示如果用户拥有访问指定url的权限即表示可以显示authorize标签包含的内容
- var：用于指定将权限鉴定的结果存放在pageContext的哪个属性中

6.4.2.3 accesscontrollist

accesscontrollist标签是用于鉴定ACL权限的。其一共定义了三个属性：hasPermission、domainObject和var，其中前两个是必须指定的

```
<security:accesscontrollist hasPermission="" domainObject="" var=""></security:accesscontrollist>
```

- hasPermission：hasPermission属性用于指定以逗号分隔的权限列表
- domainObject：domainObject用于指定对应的域对象
- var：var则是用以将鉴定的结果以指定的属性名存入pageContext中，以供同一页面的其它地方使用

第一步

```
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-taglibs</artifactId>
  <version>version</version>
</dependency>
```

第二步 在jsp页面导入

```
<%@taglib uri="http://www.springframework.org/security/tags" prefix="security"%>
```

第三步 在页面上使用

authentication 可以获取当前正在操作的用户信息

```
<security:authentication
property="principal.username"></security:authentication>
```

authorize 用于控制页面上某些标签是否可以显示

```
<security:authorize access="hasRole('ADMIN')">
```

JSR-250注解使用介绍

第一步：需要在spring-security.xml中开启

```
<security:global-method-security jsr250-annotations="enabled">
```

第二步：在指定的方法上使用

```
//查询全部产品
@RequestMapping("/findAll.do")
@RolesAllowed("ADMIN")
public ModelAndView findAll() throws Exception {
    ModelAndView mv = new ModelAndView();
    List<Product> ps = productService.findAll();
    mv.addObject( attributeName: "productList", ps);
    mv.setViewName("product-list1");
    return mv;

}
```

第三步：必须在pom.xml文件中导入

```
<dependency>
    <groupId>javax.annotation</groupId>
    <artifactId>jsr250-api</artifactId>
    <version>1.0</version>
</dependency>
```

3.@Secured注解使用

1.开启使用

```
<security:global-method-security secured-annotations="enabled"></security:global-method-security>
```

2.在指定的方法上使用

```
@RequestMapping("/findAll.do")
@Secured("ROLE_ADMIN")
public ModelAndView findAll(@RequestParam(name = "page", required = tru
    ModelAndView mv = new ModelAndView();
    List<Orders> ordersList = ordersService.findAll(page, size);
    //PageInfo就是一个分页Bean
    PageInfo pageInfo = new PageInfo(ordersList);
    mv.addObject( attributeName: "pageInfo", pageInfo);
    mv.setViewName("orders-page-list");
    return mv;
}
```

注意，在使用JSR250注解时，可以省略ROLE_前缀，而我们现在使用的@Secured是不能省略前缀

表达式使用介绍

第一步 开启

```
<security:global-method-security pre-post-annotations="enabled" />
```

第二步 使用，可以使用SPEL表达式

```
//用户添加
@RequestMapping("/save.do")
@PreAuthorize("authentication.principal.username == 'tom'")
public String save(UserInfo userInfo) throws Exception {
    userService.save(userInfo);
    return "redirect:findAll.do";
}

@RequestMapping("/findAll.do")
@PreAuthorize("hasRole('ROLE_ADMIN')")
public ModelAndView findAll() throws Exception {
    ModelAndView mv = new ModelAndView();
    List<UserInfo> userList = userService.findAll();
    mv.addObject( attributeName: "userList", userList);
    mv.setViewName("user-list");
```

Spring EL 表达式

Spring Security 允许我们在定义 URL 访问或方法访问所应有的权限时使用 Spring EL 表达式，在定义所需的访问权限时如果对应的表达式返回结果为 true 则表示拥有对应的权限，反之则无。Spring Security 可用表达式对象的基类是 SecurityExpressionRoot，其为我们提供了如下在使用 Spring EL 表达式对 URL 或方法进行权限控制时通用的内置表达式。

表达式	描述
hasRole([role])	当前用户是否拥有指定角色。
hasAnyRole([role1,role2])	多个角色是一个以逗号进行分隔的字符串。如果当前用户拥有指定角色中的任意一个则返回 true。
hasAuthority([auth])	等同于 hasRole
hasAnyAuthority([auth1,auth2])	等同于 hasAnyRole
Principle	代表当前用户的 principle 对象
authentication	直接从 SecurityContext 获取的当前 Authentication 对象
permitAll	总是返回 true，表示允许所有的
denyAll	总是返回 false，表示拒绝所有的
isAnonymous()	当前用户是否是一个匿名用户
isRememberMe()	表示当前用户是否是通过 Remember-Me 自动登录的
isAuthenticated()	表示当前用户是否已经登录认证成功了。
isFullyAuthenticated()	如果当前用户既不是一个匿名用户，同时又不是通过 Remember-Me 自动登录的，则返回 true。

SSMAOP日志

1.数据库与表结构

1.1.日志表信息描述sysLog

序号	字段名称	字段类型	字段描述
1	id	VARCHAR2	主键 无意义uuid
2	visitTime	timestamp	访问时间
3	username	VARCHAR2	操作者用户名
4	ip	VARCHAR2	访问ip
5	url	VARCHAR2	访问资源url
6	executionTime	int	执行时长
7	method	VARCHAR	访问方法

1.2.sql语句

```
CREATE TABLE sysLog(
    id VARCHAR2(32) default SYS_GUID() PRIMARY KEY,
    visitTime timestamp,
    username VARCHAR2(50),
    ip VARCHAR2(30),
    url VARCHAR2(50),
    executionTime int,
    method VARCHAR2(200)
)
```

1.3.实体类

```
public class SysLog {

    private String id;
    private Date visitTime;
    private String visitTimeStr;
    private String username;
    private String ip;
    private String url;
    private Long executionTime;
    private String method;
  }
```

2.基于AOP日志处理

2.1.页面syslog-list.jsp

详细内容请查看资源中页面信息

2.2.创建切面类处理日志

在SpringMVC中开启aop注解支持，因为controller归它管

```
<!--
    支持AOP的注解支持，AOP底层使用代理技术
    JDK动态代理，要求必须有接口
    cglib代理，生成子类对象，proxy-target-class="true" 默认使用cglib的方式
-->
<aop:aspectj-autoproxy proxy-target-class="true"/>
```

```
@Component
@Aspect
public class LogAop {

    @Autowired
    private HttpServletRequest request;

    @Autowired
    private ISysLogService sysLogService;

    private Date startTime; // 访问时间
    private Class executionClass;// 访问的类
    private Method executionMethod; // 访问的方法
    // 主要获取访问时间、访问的类、访问的方法

    @Before("execution(* com.itheima.ssm.controller.*.*(..))")
    public void doBefore(JoinPoint jp) throws NoSuchMethodException, SecurityException {
        startTime = new Date(); // 访问时间
        // 获取访问的类
        executionClass = jp.getTarget().getClass();
        // 获取访问的方法
        String methodName = jp.getSignature().getName();// 获取访问的方法的名称

        Object[] args = jp.getArgs();// 获取访问的方法的参数
        if (args == null || args.length == 0) {// 无参数
            executionMethod = executionClass.getMethod(methodName); // 只能获取无参数方法
        } else {
            // 有参数，就将args中所有元素遍历，获取对应的Class,装入到一个Class[]
            Class[] classArgs = new Class[args.length];
```

获取ip需要request

```
<listener>
    <listener-class>org.springframework.web.context.request.RequestContextListener</listener-class>
</listener>
```

```
            for (int i = 0; i < args.length; i++) {
                classArgs[i] = args[i].getClass();
            }
            executionMethod = executionClass.getMethod(methodName, classArgs);// 获取有参数方法
        }
    }

    // 主要获取日志中其它信息，时长、ip、url...
    @After("execution(* com.itheima.ssm.controller.*.*(..))")
    public void doAfter(JoinPoint jp) throws Exception {

        // 获取类上的@RequestMapping对象
        if (executionClass != SysLogController.class) {
            RequestMapping classAnnotation = (RequestMapping)
executionClass.getAnnotation(RequestMapping.class);
            if (classAnnotation != null) {
                // 获取方法上的@RequestMapping对象
                RequestMapping methodAnnotation =
executionMethod.getAnnotation(RequestMapping.class);

                if (methodAnnotation != null) {

                    String url = ""; // 它的值应该是类上的@RequestMapping的value+方法上的
@RequestMapping的value

                    url = classAnnotation.value()[0] + methodAnnotation.value()[0];

                    SysLog sysLog = new SysLog();
                    // 获取访问时长
                    Long executionTime = new Date().getTime() - startTime.getTime();
                    // 将sysLog对象属性封装
                    sysLog.setExecutionTime(executionTime);
                    sysLog.setUrl(url);
                    // 获取ip
                    String ip = request.getRemoteAddr();
                    sysLog.setIp(ip);

                    // 可以通过securityContext获取，也可以从request.getSession中获取
                    SecurityContext context = SecurityContextHolder.getContext(); //
request.getSession().getAttribute("SPRING_SECURITY_CONTEXT")
                    String username = ((User)
(context.getAuthentication().getPrincipal())).getUsername();
                    sysLog.setUsername(username);

                    sysLog.setMethod("[类名]" + executionClass.getName() + "[方法名]" +
executionMethod.getName());

                    sysLog.setVisitTime(startTime);

                    // 调用Service，调用dao将sysLog insert数据库
                    sysLogService.save(sysLog);
                }

            }
```

```
        }
    }
}
```

在切面类中我们需要获取登录用户的username，还需要获取ip地址，我们怎么处理?

- username获取

  SecurityContextHolder获取

- ip地址获取

  ip地址的获取我们可以通过request.getRemoteAddr()方法获取到。

  在Spring中可以通过RequestContextListener来获取request或session对象。

2.3.SysLogController

```
@RequestMapping("/sysLog")
@Controller
public class SysLogController {

    @Autowired
    private ISysLogService sysLogService;

    @RequestMapping("/findAll.do")
    public ModelAndView findAll() throws Exception {
        ModelAndView mv = new ModelAndView();
        List<SysLog> sysLogs = sysLogService.findAll();
        mv.addObject("sysLogs", sysLogs);
        mv.setViewName("syslog-list");
        return mv;
    }
}
```

2.4.Service

```
@Service
@Transactional
public class SysLogServiceImpl implements ISysLogService {

    @Autowired
    private ISysLogDao sysLogDao;

    @Override
    public void save(SysLog log) throws Exception {
        sysLogDao.save(log);
    }


    @Override
```

```
    public List<SysLog> findAll() throws Exception {
        return sysLogDao.findAll();
    }
}
```

2.5.Dao

```
public interface ISysLogDao {

    @Select("select * from syslog")
    @Results({
        @Result(id=true,column="id",property="id"),
        @Result(column="visitTime",property="visitTime"),
        @Result(column="ip",property="ip"),
        @Result(column="url",property="url"),
        @Result(column="executionTime",property="executionTime"),
        @Result(column="method",property="method"),
        @Result(column="username",property="username")
    })
    public List<SysLog> findAll() throws Exception;

    @Insert("insert into syslog(visitTime,username,ip,url,executionTime,method) values(#
{visitTime},#{username},#{ip},#{url},#{executionTime},#{method})")
    public void save(SysLog log) throws Exception;

}
```

AOP日志中信息获取图片

1.操作者如何获取

可以通过Srping Security提供的一个

```
SecurityContext context=
SecurityContextHolder.getContext();

      User user = (User)
context.getAuthentication().getPrincipal();


String username = user.getUsername();
```

2.如何获取访问的IP

通过request对象
在web.xm文件中配置一个Listener
RequestContextListener

```
@Autowired
private HttpServletRequest request;
```

3.如何获取访问的URL

需要通过反射来完成操作

```
//获取url
if(clazz!=null&&method!=null&&clazz== LogAop.class){
    //1.获取类上的@RequestMapping("/orders")
    RequestMapping classAnnotation = (RequestMapping) clazz.getAnnotation(RequestMapping.class);
    if(classAnnotation!=null){
        String[] classValue = classAnnotation.value();
        //2.获取方法上的@RequestMapping(xxx)
        RequestMapping methodAnnotation = method.getAnnotation(RequestMapping.class);
        if(methodAnnotation!=null){
            String[] methodValue = methodAnnotation.value();
            url=classValue[0]+methodValue[0];
        }
    }
```

4.如何获取执行的时长

在后置通知中获取一个当前时间减去访问时间visitTime，就可以获取访问时长