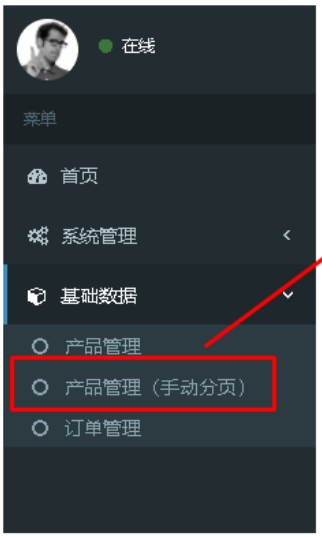
ssm练习第三天

第一章：分页助手PageHelper

第一节：手动分页

1、页面入口




```
<li id="system-setting"><a
    href="${pageContext.request.contextPath}/product/findAll">
        <i class="fa fa-circle-o"></i> 产品管理
</a></li>
<li id="system-setting"><a
        href="${pageContext.request.contextPath}/product/findByPage">
    <i class="fa fa-circle-o"></i> 产品管理（手动分页）
</a></li>
<li id="system-setting"><a
    href="${pageContext.request.contextPath}/order/findAll">
        <i class="fa fa-circle-o"></i> 订单管理
</a></li>
```

2、编写分页实体PageBean

```java
package com.itheima.domain;

import java.util.List;

public class PageBean<T> {

    private Integer pageNum;//当前页码
    private Integer pageSize;//当前页显示条数
    private Long totalCount;//总条数
    private Integer totalPage;//总页数
    private List<T> pageList;//当前页显示的数据列表

    //省略getter和setter方法... ...
}

```

3、编写Controller

```java
@RequestMapping("/findByPage")
public ModelAndView findByPage(
    @RequestParam(value = "pageNum",defaultValue = "1") Integer pageNum,
    @RequestParam(value = "pageSize",defaultValue = "5") Integer pageSize)
{
    //查询所有商品数据
    PageBean<Product> pageBean = productService.findByPage(pageNum,pageSize);
    ModelAndView modelAndView = new ModelAndView();
    modelAndView.addObject("pageBean",pageBean);
    modelAndView.setViewName("product-list-page");
    return modelAndView;
}
```

4、编写Service

接口

```java
PageBean<Product> findByPage(Integer pageNum,Integer pageSize);
```

实现

```java
@Override
public PageBean<Product> findByPage(Integer pageNum,Integer pageSize) {
    //封装PageBean实体
    PageBean<Product> pageBean = new PageBean<>();
    //1、private Integer pageNum;//当前页码
    pageBean.setPageNum(pageNum);
    //2、private Integer pageSize;//当前页显示条数
    pageBean.setPageSize(pageSize);
    //3、private Long totalCount;//总条数
    Long totalCount = productMapper.findProductCount();
    pageBean.setTotalCount(totalCount);
    //4、private Integer totalPage;//总页数
    int totalPage = (int) Math.ceil(1.0*totalCount/pageSize);
    pageBean.setTotalPage(totalPage);
    //5、private List<T> pageList;//当前页显示的数据列表
    int startIndex = (pageNum-1)*pageSize;
    List<Product> pageList = productMapper.findPageList(startIndex,pageSize);
    pageBean.setPageList(pageList);

    return pageBean;
}
```

5、编写Mapper

```
@Select("select count(*) from product")
Long findProductCount();

@Select("select * from product limit #{param1},#{param2}")
List<Product> findPageList(int startIndex, Integer pageSize);
```

6、编写页面

```
<!--数据列表展示-->
<c:forEach items="${pageBean.pageList}" var="product">

    <tr>
        <td><input name="ids" type="checkbox" value="${product.id}"></td>
        <td>${product.id}</td>

        <td>${product.productNum}</td>
        <td>${product.productName}</td>
        <td>${product.departureTime}</td>
        <td>${product.productStatus==1?"开启":"关闭"}</td>

        <td class="text-center">
            <button type="button" class="btn bg-olive btn-xs"
                    onclick='location.href="all-order-manage-edit.html"'>订单
</button>
            <button type="button" class="btn bg-olive btn-xs"
onclick='location.href="${pageContext.request.contextPath}/product/editUI.do?
id=${product.id}"'>查看</button>
        </td>
    </tr>

</c:forEach>

<!--分页条-->
<div class="box-footer">
    <div class="pull-left">
        <div class="form-group form-inline">
            总共${pageBean.totalPage}页，共${pageBean.totalCount} 条数据。 每页
            <select id="pageSize" onchange="gotoPage('1')" class="form-control">
                <option value="5">5</option>
                <option value="10">10</option>
                <option value="15">15</option>
                <option value="20">20</option>
            </select> 条
        </div>
    </div>

    <div class="box-tools pull-right">
        <ul class="pagination">
            <li><a href="javascript:gotoPage('1')">首页</a></li>
            <li>
                <a href="javascript:gotoPage('${pageBean.pageNum-1}')">上一页</a>
            </li>
```

```
            <c:forEach begin="1" end="${pageBean.totalPage}" var="page">
                <li class="${pageBean.pageNum==page?'active':''}">
                    <a href="javascript:gotoPage('${page}')">${page}</a>
                </li>
            </c:forEach>
            <li>
                <a href="javascript:gotoPage('${pageBean.pageNum+1}')">下一页</a>
            </li>
            <li><a href="javascript:gotoPage('${pageBean.totalPage}')" aria-label="Next">尾页</a></li>
        </ul>
    </div>
</div>
```

```
//js功能实现
function gotoPage(pageNum){
    var pageSize = $("#pageSize option:selected").val();
    //判断页码的有效性
    if(pageNum>=1&&pageNum<=${pageBean.totalPage}){
        location.href = "${pageContext.request.contextPath}/product/findByPage?pageNum="+pageNum+"&pageSize="+pageSize;
    }

}

$(function(){
    $("#pageSize option[value='${pageBean.pageSize}']").prop("selected",true);
})
```

第二节：分页助手PageHelper的使用

1、PageHelper简介

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。

网址：https://pagehelper.github.io/

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

2、PageHelper的环境搭建

添加PageHelper坐标（之前的pom中已将包含该坐标）

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

配置mybatis的PageHelper插件，mybatis的配置已经集成到spring的配置文件中了，所以在配置SqlSessionFactory时指定插件

```
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <!--数据源-->
    <property name="dataSource" ref="dataSource"></property>
    <!--mybatis的其他配置 -->
    <property name="plugins">
        <array>
            <bean class="com.github.pagehelper.PageInterceptor">
                <property name="properties">
                    <props>
                        <!-- 分页的相关配置参数 详细参数解析见附录 -->
                        <prop key="dialect">mysql</prop>
                    </props>
                </property>
            </bean>
        </array>
    </property>
</bean>
```

3、PageHelper的快速入门

在ssm_web的test测试目录中创建PageHelperTest测试类

```
package com.itheima.test;

import com.github.pagehelper.PageHelper;
import com.itheima.domain.Product;
import com.itheima.service.ProductService;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import java.util.List;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:applicationContext.xml")
public class PageHelperTest {

    @Autowired
    private ProductService productService;

    @Test
    public void testPageHelper(){

        /*
            设置分页数据
            pageNum:页码
            pageSize:当前页显示的条数
```

```
         */

        PageHelper.startPage(2,5);

        List<Product> all = productService.findAll();
        for (Product product : all) {
            System.out.println(product.getId());
        }
    }

}
```

4、PageHelper的API解析

4.1 设置分页参数

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。

```
public static <E> Page<E> startPage(int pageNum, int pageSize) {
    return startPage(pageNum, pageSize, true);
}
```

参数：pageNum：当前页；pageSize：当前页显示的条数

4.2 获得分页相关数据

创建PageInfo对象，将查询结果作为参数传递进构造，则可以通过PageInfo获得分页的相关信息：

```
//1、设置分页参数
PageHelper.startPage(2,5);
//2、查询数据列表
List<Product> all = productService.findAll();
for (Product product : all) {
    System.out.println(product.getId());
}
//3、创建PageInfo
PageInfo<Product> pageInfo = new PageInfo<>(all);
//4、获得分页的相关数据
System.out.println("总条数:"+pageInfo.getTotal());
System.out.println("总页数:"+pageInfo.getPages());
System.out.println("当前页:"+pageInfo.getPageNum());
System.out.println("当前页显示条数:"+pageInfo.getPageSize());
System.out.println("是否是第一页:"+pageInfo.isIsFirstPage());
System.out.println("是否是最后一页:"+pageInfo.isIsLastPage());
System.out.println("上一页是:"+pageInfo.getPrePage());
System.out.println("下一页是:"+pageInfo.getNextPage());

System.out.println("当前页数据是:");
List<Product> list = pageInfo.getList();
for (Product product : list) {
    System.out.println(product);
```

```
24  }
```

控制台打印结果：

```
总条数:7
总页数:2
当前页:2
当前页显示条数:5
是否是第一页:false
是否是最后一页:true
上一页是:1
下一页是:0
当前页数据是:
Product{id=12, productNum='222', productName='222', cityName='', departureTime='',
Product{id=13, productNum='333', productName='333', cityName='', departureTime='',
```

5、分页查询集成PageHelper

5.1、页面入口

5.2、编写Controller

```java
@RequestMapping("/findByPageHelper")
public ModelAndView findByPageHelper(
    @RequestParam(value = "pageNum",defaultValue = "1") Integer pageNum,
    @RequestParam(value = "pageSize",defaultValue = "5") Integer pageSize)
{
    PageInfo<Product> pageInfo = productService.findByPageHelper(pageNum,pageSize);
    ModelAndView modelAndView = new ModelAndView();
    modelAndView.addObject("pageInfo",pageInfo);
    modelAndView.setViewName("product-list-pagehelper");
    return modelAndView;
}
```

5.3、编写Service

接口

```
PageInfo<Product> findByPageHelper(Integer pageNum, Integer pageSize);
```

实现

```
@Override
public PageInfo<Product> findByPageHelper(Integer pageNum, Integer pageSize) {
    PageHelper.startPage(pageNum,pageSize);
    List<Product> list = productMapper.findAll();
    PageInfo<Product> pageInfo = new PageInfo<>(list);
    return pageInfo;
}
```

5.4、编写Mapper

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

5.5、编写页面

```
<!--数据列表展示-->
<c:forEach items="${pageInfo.list}" var="product">
    <tr>
        <td><input name="ids" type="checkbox" value="${product.id}"></td>
        <td>${product.id}</td>

        <td>${product.productNum}</td>
        <td>${product.productName}</td>
        <td>${product.departureTime}</td>
        <td>${product.productStatus==1?"开启":"关闭"}</td>

        <td class="text-center">
            <button type="button" class="btn bg-olive btn-xs"
                    onclick='location.href="all-order-manage-edit.html"'>订单
</button>
            <button type="button" class="btn bg-olive btn-xs"

 onclick='location.href="${pageContext.request.contextPath}/product/editUI.do?
id=${product.id}"'>查看</button>
        </td>
    </tr>
</c:forEach>

<!--分页条-->
<div class="box-footer">
    <div class="pull-left">
        <div class="form-group form-inline">
            总共${pageInfo.total}页，共${pageInfo.pages} 条数据。 每页
            <select id="pageSize" onchange="gotoPage('1')" class="form-control">
                <option value="5">5</option>
                <option value="10">10</option>
```

```
                <option value="15">15</option>
                <option value="20">20</option>
            </select> 条
        </div>
    </div>

    <div class="box-tools pull-right">
        <ul class="pagination">

            <li><a href="javascript:gotoPage('1')">首页</a></li>

            <li>
                <a href="javascript:gotoPage('${pageInfo.pageNum-1}')">上一页</a>
            </li>

            <c:forEach begin="1" end="${pageInfo.pages}" var="page">
                <li class="${pageInfo.pageNum==page?'active':''}">
                    <a href="javascript:gotoPage('${page}')">${page}</a>
                </li>
            </c:forEach>

            <li>
                <a href="javascript:gotoPage('${pageInfo.pageNum+1}')">下一页</a>
            </li>

            <li><a href="javascript:gotoPage('${pageInfo.pages}')" aria-label="Next">
尾页</a></li>
        </ul>
    </div>
</div>
```

```
//js功能实现
function gotoPage(pageNum){
    var pageSize = $("#pageSize option:selected").val();
    //判断页码的有效性
    if(pageNum>=1&&pageNum<=${pageInfo.pages}){
        location.href = "${pageContext.request.contextPath}/product/findByPageHelper?
pageNum="+pageNum+"&pageSize="+pageSize;
    }

}

$(function(){
    $("#pageSize option[value='${pageInfo.pageSize}']").prop("selected",true);
})
```

第二章：SpringSecurity安全框架

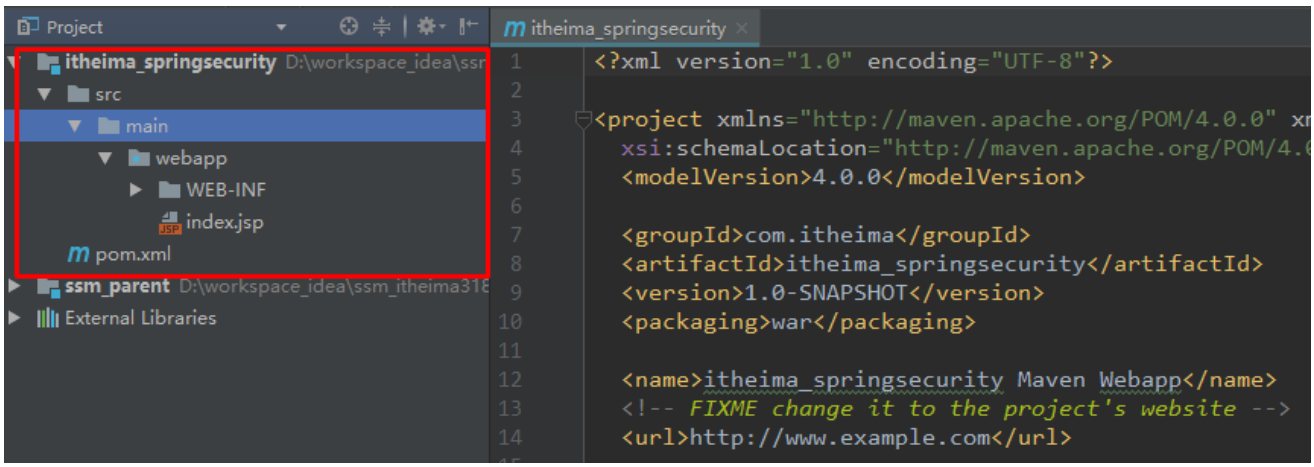
第一节：SpringSecurity的简介

Spring Security是一个能够为基于Spring的企业应用系统提供声明式的安全访问控制解决方案的安全框架。它提供了一组可以在Spring应用上下文中配置的Bean，充分利用了Spring IoC，DI（控制反转Inversion of Control ,DI:Dependency Injection 依赖注入）和AOP（面向切面编程）功能，为应用系统提供声明式的安全访问控制功能，减少了为企业系统安全控制编写大量重复代码的工作。

- “认证”，是为用户建立一个他所声明的主体。主题一般是指用户，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。

第二节：SpringSecurity的入门使用

1、创建itheima_springSecurity项目（war）



2、导入SpringSecurity的坐标（pom.xml）

```
<?xml version="1.0" encoding="UTF-8"?>
<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>com.itheima</groupId>
  <artifactId>itheima_springsecurity</artifactId>
  <version>1.0-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>
```

3、配置spring-security.xml

在类加载路径resources下创建spring-security.xml配置文件，配置认证和授权信息

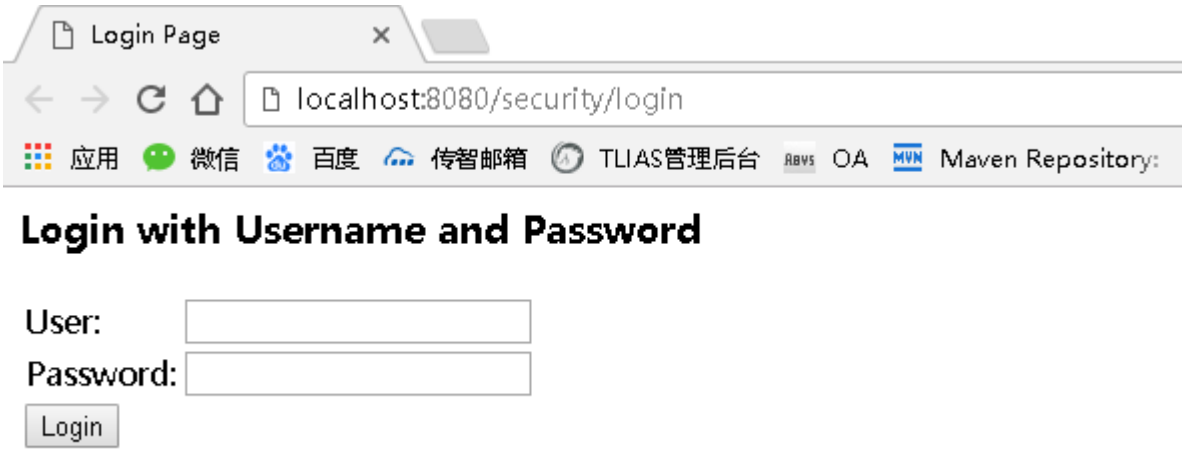
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:security="http://www.springframework.org/schema/security"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans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

    <!-- 1、配置认证信息 -->
    <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>

    <!--
        2、配置授权信息
        配置拦截的规则
        auto-config="使用自带的页面"
        use-expressions="是否使用spel表达式，如果使用表达式：hasRole('ROLE_USER')
    -->
    <security:http auto-config="true" use-expressions="false">
        <!-- 配置拦截的请求地址，任何请求地址都必须有ROLE_USER的权限 -->
        <security:intercept-url pattern="/**" access="ROLE_USER"/>
    </security:http>

</beans>
```

4、配置web.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
         version="3.1">

  <!--Spring监听器指定配置文件位置-->
  <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>

  <!--配置委派代理过滤器-->
  <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>

</web-app>
```

5、测试

访问index.jsp页面，如果当前用户没有登录认证的话，则跳转到SpringSecurity的内置登录页面



6、配置自定义登录页面

```xml
<?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">

    <!-- 把登陆页面不拦截 -->
    <security:http pattern="/login.html" security="none"/>
    <security:http pattern="/error.html" security="none"/>

    <!--
        配置拦截的规则
        auto-config="使用自带的页面"
        use-expressions="是否使用spel表达式，如果使用表达式：hasRole('ROLE_USER')
    -->
    <security:http auto-config="true" use-expressions="false">
        <!-- 配置拦截的请求地址，任何请求地址都必须有ROLE_USER的权限 -->
        <security:intercept-url pattern="/**" access="ROLE_USER"/>
        <!-- 配置具体的页面跳转 -->
        <security:form-login
                login-page="/login.html"
                login-processing-url="/login"
                default-target-url="/success.html"
                authentication-failure-url="/error.html"
        />

        <!-- 关闭跨站请求伪造 -->
        <security:csrf disabled="true"/>

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

    </security:http>

    <!-- 在内存中临时提供用户名和密码的数据 -->
    <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>
```

7、login.html页面

```
<!DOCTYPE html>
<html>
```

```
<head>
    <meta charset="UTF-8">
    <title>Insert title here</title>
</head>
<body>
<h1>自定义的登录页面</h1>
<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>
```

8、success.html页面

```
<!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>
```

9、error.html页面

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

第三节：项目集成SpringSecurity

1、添加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">

    <!-- 不拦截静态资源 -->
    <security:http pattern="/css/**" security="none"/>
    <security:http pattern="/img/**" security="none"/>
    <security:http pattern="/plugins/**" security="none"/>

    <!-- 把登陆页面不拦截 -->
    <security:http pattern="/login.jsp" security="none"/>

    <!--
        配置拦截的规则
        auto-config="使用自带的页面"
        use-expressions="是否使用spel表达式，如果使用表达式：hasRole('ROLE_USER')
    -->
    <security:http auto-config="true" use-expressions="false">
        <!-- 配置拦截的请求地址，任何请求地址都必须有ROLE_USER的权限 -->
        <security:intercept-url pattern="/**" access="ROLE_USER"/>
        <!-- 配置具体的页面跳转 -->
        <security:form-login
                login-page="/login.jsp"
                login-processing-url="/login"
                default-target-url="/index.jsp"
        />

        <!-- 关闭跨越请求 -->
        <security:csrf disabled="true"/>

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

    </security:http>

    <!-- 在内存中临时提供用户名和密码的数据 -->
    <security:authentication-manager>
        <!-- 提供服务类，去数据库查询用户名和密码 -->
        <security:authentication-provider user-service-ref="userService"/>
    </security:authentication-manager>

</beans>
```

2、修改web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" version="2.5">
  <display-name>ssm_web</display-name>


  <!--1、配置spring的监听器-->
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:applicationContext.xml,classpath:spring-
security.xml</param-value>
  </context-param>
  <listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-
class>
  </listener>

  <!--2、配置SptingMVC的前端控制器-->
  <servlet>
    <servlet-name>DispatcherServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>classpath:springmvc.xml,classpath:spring-security.xml</param-
value>
    </init-param>
    <!--服务器启动就创建该Servlet-->
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>DispatcherServlet</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>

  <!--3、乱码的解决方案-->
  <filter>
    <filter-name>CharacterEncodingFilter</filter-name>
    <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-
class>
    <init-param>
      <param-name>encoding</param-name>
      <param-value>UTF-8</param-value>
    </init-param>
  </filter>
  <filter-mapping>
    <filter-name>CharacterEncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>

```

```
  <!--4、springsecurity委派过滤器-->
  <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>

  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

3、创建用户表和用户实体

```
CREATE TABLE sys_user(
  id BIGINT PRIMARY KEY AUTO_INCREMENT,
  username VARCHAR(50),
  email VARCHAR(50) ,
  password VARCHAR(80),
  phoneNum VARCHAR(20),
  status INT
);
```

```
package com.itheima.domain;

public class SysUser {

    private Long id;
    private String username;
    private String email;
    private String password;
    private String phoneNum;
    private int status;

    //省略getter和setter方法... ...
}

```

4、修改spring-security.xml，修改认证方式为查询数据库

```xml
<!-- 在内存中临时提供用户名和密码的数据 -->
<security:authentication-manager>
    <!-- 提供服务类，去数据库查询用户名和密码 -->
    <security:authentication-provider user-service-ref="userService"/>
</security:authentication-manager>
```

5、编写UserService业务代码

userService接口，接口集成UserDetailsService接口

```java
package com.itheima.service;

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

public interface UserService extends UserDetailsService {
}
```

userServiceImpl实现

```java
package com.itheima.service.impl;

import com.itheima.dao.UserMapper;
import com.itheima.domain.SysUser;
import com.itheima.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.List;

@Service("userService")
public class UserServiceImpl implements UserService {

    @Autowired
    private UserMapper userMapper;

    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        // 先设置假的权限
        List<GrantedAuthority> authorities = new ArrayList<>();
        authorities.add(new SimpleGrantedAuthority("ROLE_USER"));
        // 通过用户名查询密码
        SysUser sysUser = userMapper.findByUsername(username);
        //判断认证是否成功
        if(sysUser!=null){
```

```
            User user = new User(username,"{noop}"+sysUser.getPassword(),authorities);
            return user;
        }

        return null;
    }
}

```

6、修改login.jsp的表单提交地址为login

附录：PageHelper分页插件的相关参数

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。