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





2、编写分页实体PageBean

```
1
    package com.itheima.domain;
2
3
   import java.util.List;
4
    public class PageBean<T> {
6
        private Integer pageNum;//当前页码
        private Integer pageSize;//当前页显示条数
8
9
        private Long totalCount;//总条数
        private Integer totalPage;//总页数
10
11
        private List<T> pageList;//当前页显示的数据列表
12
13
        //省略getter和setter方法......
14
15
```

3、编写Controller

```
1
    @RequestMapping("/findByPage")
    public ModelAndView findByPage(
 2
        @RequestParam(value = "pageNum", defaultValue = "1") Integer pageNum,
 3
        @RequestParam(value = "pageSize",defaultValue = "5") Integer pageSize)
4
 5
    {
        //查询所有商品数据
 6
 7
        PageBean<Product> pageBean = productService.findByPage(pageNum,pageSize);
        ModelAndView modelAndView = new ModelAndView();
 8
        modelAndView.addObject("pageBean",pageBean);
9
10
        modelAndView.setViewName("product-list-page");
11
        return modelAndView;
12
```

4、编写Service

接口

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

实现

```
1
    @Override
    public PageBean<Product> findByPage(Integer pageNum,Integer pageSize) {
2
3
        //封装PageBean实体
        PageBean<Product> pageBean = new PageBean<>();
 5
        //1、private Integer pageNum;//当前页码
        pageBean.setPageNum(pageNum);
 6
        //2、private Integer pageSize;//当前页显示条数
8
        pageBean.setPageSize(pageSize);
        //3、private Long totalCount;//总条数
10
        Long totalCount = productMapper.findProductCount();
        pageBean.setTotalCount(totalCount);
11
        //4、private Integer totalPage;//总页数
12
        int totalPage = (int) Math.ceil(1.0*totalCount/pageSize);
13
14
        pageBean.setTotalPage(totalPage);
        //5、private List<T> pageList;//当前页显示的数据列表
15
16
        int startIndex = (pageNum-1)*pageSize;
        List<Product> pageList = productMapper.findPageList(startIndex,pageSize);
17
        pageBean.setPageList(pageList);
18
19
20
        return pageBean;
21
```

5、编写Mapper

```
1  @Select("select count(*) from product")
2  Long findProductCount();
3
4  @Select("select * from product limit #{param1},#{param2}")
5  List<Product> findPageList(int startIndex, Integer pageSize);
```

6、编写页面

```
<! - - 数据列表展示 - - >
 1
2
    <c:forEach items="${pageBean.pageList}" var="product">
4
       <input name="ids" type="checkbox" value="${product.id}">
 5
           ${product.id}
6
7
8
           ${product.productNum}
9
           ${product.productName}
           ${product.departureTime}
10
           ${product.productStatus==1?"开启":"关闭"}
11
12
13
           <button type="button" class="btn bg-olive btn-xs"</pre>
14
15
                      onclick='location.href="all-order-manage-edit.html"'>订单
    </button>
               <button type="button" class="btn bg-olive btn-xs"</pre>
16
    onclick='location.href="${pageContext.request.contextPath}/product/editUI.do?
    id=${product.id}"'>查看</button>
17
           18
       19
20
   </c:forEach>
21
22
    <! - - 分页条 - - >
    <div class="box-footer">
23
       <div class="pull-left">
24
           <div class="form-group form-inline">
25
26
               总共${pageBean.totalPage}页,共${pageBean.totalCount} 条数据。 每页
               <select id="pageSize" onchange="gotoPage('1')" class="form-control">
27
28
                  <option value="5">5</option>
                  <option value="10">10</option>
29
                  <option value="15">15</option>
30
                  <option value="20">20</option>
31
               </select> 条
32
33
           </div>
34
       </div>
35
36
       <div class="box-tools pull-right">
           37
38
               <a href="javascript:gotoPage('1')">首页</a>
39
               <1i>>
                  <a href="javascript:gotoPage('${pageBean.pageNum-1}')">上一页</a>
40
41
```

```
42
             <c:forEach begin="1" end="${pageBean.totalPage}" var="page">
43
                 <a href="javascript:gotoPage('${page}')">${page}</a>
44
45
                 </c:forEach>
46
             <1i>>
47
                 <a href="javascript:gotoPage('${pageBean.pageNum+1}')">下一页</a>
48
49
             <a href="javascript:gotoPage('${pageBean.totalPage}')" aria-</a>
   label="Next">尾页</a>
51
          </div>
52
53
   </div>
```

```
//js功能实现
1
 2
    function gotoPage(pageNum){
 3
        var pageSize = $("#pageSize option:selected").val();
4
        //判断页码的有效性
 5
        if(pageNum>=1&&pageNum<=${pageBean.totalPage}){</pre>
            location.href = "${pageContext.request.contextPath}/product/findByPage?
 6
    pageNum="+pageNum+"&pageSize="+pageSize;
7
        }
8
9
    }
10
    $(function(){
11
        $("#pageSize option[value='${pageBean.pageSize}']").prop("selected",true);
12
13
   })
```

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)

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">
1
2
       <!--数据源-->
       cproperty name="dataSource" ref="dataSource"></property>
 3
4
       <!--mybatis的其他配置 -->
       cproperty name="plugins">
 6
           <array>
               <bean class="com.github.pagehelper.PageInterceptor">
                   cproperty name="properties">
8
9
                       10
                           <!-- 分页的相关配置参数 详细参数解析见附录 -->
11
                            key="dialect">mysql
12
                       </props>
                   </property>
13
14
               </bean>
15
           </array>
       </property>
16
17
   </bean>
```

3、PageHelper的快速入门

在ssm_web的test测试目录中创建PageHelperTest测试类

```
1
    package com.itheima.test;
2
3
   import com.github.pagehelper.PageHelper;
    import com.itheima.domain.Product;
   import com.itheima.service.ProductService;
   import org.junit.Test;
    import org.junit.runner.RunWith;
8
    import org.springframework.beans.factory.annotation.Autowired;
9
    import org.springframework.test.context.ContextConfiguration;
10
    import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
12
    import java.util.List;
13
    @RunWith(SpringJUnit4ClassRunner.class)
14
15
    @ContextConfiguration("classpath:applicationContext.xml")
    public class PageHelperTest {
16
17
18
        @Autowired
        private ProductService productService;
19
20
21
        @Test
22
        public void testPageHelper(){
23
            /*
24
25
                设置分页数据
26
                pageNum:页码
27
                pageSize: 当前页显示的条数
```

```
28
29
             PageHelper.startPage(2,5);
30
31
             List<Product> all = productService.findAll();
32
             for (Product product : all) {
33
                 System.out.println(product.getId());
34
35
             }
        }
36
37
38
```

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
   //1、设置分页参数
2
   PageHelper.startPage(2,5);
   //2、查询数据列表
3
   List<Product> all = productService.findAll();
   for (Product product : all) {
6
       System.out.println(product.getId());
7
   //3、创建PageInfo
8
9
   PageInfo<Product> pageInfo = new PageInfo<>(all);
10
   //4、获得分页的相关数据
11
   System.out.println("总条数:"+pageInfo.getTotal());
   System.out.println("总页数:"+pageInfo.getPages());
12
   System.out.println("当前页:"+pageInfo.getPageNum());
13
14
   System.out.println("当前页显示条数:"+pageInfo.getPageSize());
   System.out.println("是否是第一页:"+pageInfo.isIsFirstPage());
15
   System.out.println("是否是最后一页:"+pageInfo.isIsLastPage());
   System.out.println("上一页是:"+pageInfo.getPrePage());
17
18
   System.out.println("下一页是:"+pageInfo.getNextPage());
19
20
   System.out.println("当前页数据是:");
   List<Product> list = pageInfo.getList();
21
22
   for (Product product : list) {
23
       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

```
1
    @RequestMapping("/findByPageHelper")
 2
    public ModelAndView findByPageHelper(
 3
        @RequestParam(value = "pageNum", defaultValue = "1") Integer pageNum,
 4
        @RequestParam(value = "pageSize", defaultValue = "5") Integer pageSize)
 5
 6
        PageInfo<Product> pageInfo = productService.findByPageHelper(pageNum,pageSize);
        ModelAndView modelAndView = new ModelAndView();
 7
8
        modelAndView.addObject("pageInfo",pageInfo);
        modelAndView.setViewName("product-list-pagehelper");
 9
10
        return modelAndView;
11
```

5.3、编写Service

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

实现

```
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

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

5.5、编写页面

```
<! - - 数据列表展示 - - >
1
2
   <c:forEach items="${pageInfo.list}" var="product">
       4
           <input name="ids" type="checkbox" value="${product.id}">
           ${product.id}
 5
 6
 7
           ${product.productNum}
           ${product.productName}
8
9
           ${product.departureTime}
10
           ${product.productStatus==1?"开启":"关闭"}
11
12
           <button type="button" class="btn bg-olive btn-xs"</pre>
13
14
                      onclick='location.href="all-order-manage-edit.html"'>订单
    </button>
               <button type="button" class="btn bg-olive btn-xs"</pre>
15
16
    onclick='location.href="${pageContext.request.contextPath}/product/editUI.do?
   id=${product.id}"'>查看</button>
           17
       18
19
   </c:forEach>
20
21
   <!--分页条-->
    <div class="box-footer">
22
       <div class="pull-left">
23
24
           <div class="form-group form-inline">
               总共${pageInfo.total}页, 共${pageInfo.pages} 条数据。 每页
25
26
               <select id="pageSize" onchange="gotoPage('1')" class="form-control">
                  <option value="5">5</option>
27
                  <option value="10">10</option>
28
```

```
29
                 <option value="15">15</option>
30
                 <option value="20">20</option>
31
              </select> 条
32
          </div>
       </div>
33
34
       <div class="box-tools pull-right">
35
          36
37
38
              <a href="javascript:gotoPage('1')">首页</a>
39
              <1i>>
40
                 <a href="javascript:gotoPage('${pageInfo.pageNum-1}')">上一页</a>
41
42
              44
              <c:forEach begin="1" end="${pageInfo.pages}" var="page">
                 45
                     <a href="javascript:gotoPage('${page}')">${page}</a>
46
47
                 </c:forEach>
48
49
              <1i>>
50
                 <a href="javascript:gotoPage('${pageInfo.pageNum+1}')">下一页</a>
51
52
              53
              <a href="javascript:gotoPage('${pageInfo.pages}')" aria-label="Next">
   尾页</a>
55
          </div>
56
   </div>
57
```

```
1
    //js功能实现
 2
    function gotoPage(pageNum){
        var pageSize = $("#pageSize option:selected").val();
        //判断页码的有效性
        if(pageNum>=1&&pageNum<=${pageInfo.pages}){</pre>
 5
            location.href = "${pageContext.request.contextPath}/product/findByPageHelper?
 6
    pageNum="+pageNum+"&pageSize="+pageSize;
7
8
9
10
    $(function(){
11
12
        $("#pageSize option[value='${pageInfo.pageSize}']").prop("selected",true);
13
    })
```

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)

```
📭 itheima_springsecurity D:\workspace_idea\ssi
                                          <?xml version="1.0" encoding="UTF-8"?>
                                          Kproject xmlns="http://maven.apache.org/POM/4.0.0" x
                                            xsi:schemaLocation="http://maven.apache.org/POM/4.
    ▼ I webapp
                                            <modelVersion>4.0.0</modelVersion>
       ▶ ■ WEB-INF
         # index.jsp
                                            <groupId>com.itheima
                                            <artifactId>itheima_springsecurity</artifactId>
ssm_parent D:\workspace_idea\ssm_itheima318
                                            <version>1.0-SNAPSHOT
IIII External Libraries
                                            <packaging>war</packaging>
                                            <name>itheima_springsecurity Maven Webapp
                                            <!-- FIXME change it to the project's website
                                            <url>http://www.example.com</url>
```

2、导入SpringSecurity的坐标 (pom.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
 1
    cproject xmlns="http://maven.apache.org/POM/4.0.0"
 2
    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">
4
      <modelVersion>4.0.0</modelVersion>
 5
 6
      <groupId>com.itheima
      <artifactId>itheima springsecurity</artifactId>
      <version>1.0-SNAPSHOT</version>
 8
9
      <packaging>war</packaging>
10
11
      cproperties>
        <spring.version>5.0.2.RELEASE</spring.version>
12
13
        <spring.security.version>5.0.1.RELEASE</spring.security.version>
14
      </properties>
      <dependencies>
15
16
        <dependency>
17
          <groupId>org.springframework</groupId>
          <artifactId>spring-core</artifactId>
18
19
          <version>${spring.version}</version>
20
        </dependency>
21
        <dependency>
```

```
22
          <groupId>org.springframework</groupId>
23
          <artifactId>spring-web</artifactId>
          <version>${spring.version}</version>
24
25
        </dependency>
26
        <dependency>
27
          <groupId>org.springframework</groupId>
28
          <artifactId>spring-webmvc</artifactId>
29
          <version>${spring.version}</version>
30
        </dependency>
31
        <dependency>
32
          <groupId>org.springframework</groupId>
33
          <artifactId>spring-context-support</artifactId>
          <version>${spring.version}</version>
34
35
        </dependency>
        <dependency>
36
37
          <groupId>org.springframework</groupId>
38
          <artifactId>spring-test</artifactId>
39
          <version>${spring.version}</version>
40
        </dependency>
41
        <dependency>
42
          <groupId>org.springframework
          <artifactId>spring-jdbc</artifactId>
43
          <version>${spring.version}</version>
44
45
        </dependency>
46
        <dependency>
47
          <groupId>org.springframework.security</groupId>
48
          <artifactId>spring-security-web</artifactId>
          <version>${spring.security.version}</version>
49
50
        </dependency>
51
        <dependency>
52
          <groupId>org.springframework.security</groupId>
53
          <artifactId>spring-security-config</artifactId>
          <version>${spring.security.version}</version>
54
55
        </dependency>
        <dependency>
56
57
          <groupId>javax.servlet
58
          <artifactId>javax.servlet-api</artifactId>
59
          <version>3.1.0
60
          <scope>provided</scope>
61
        </dependency>
62
      </dependencies>
      <build>
63
64
        <plugins>
          <!-- java编译插件 -->
65
66
          <plugin>
67
            <groupId>org.apache.maven.plugins
            <artifactId>maven-compiler-plugin</artifactId>
68
69
            <version>3.2</version>
70
            <configuration>
71
              <source>1.8</source>
72
              <target>1.8</target>
73
              <encoding>UTF-8</encoding>
74
            </configuration>
```

```
75
          </plugin>
76
          <plugin>
            <groupId>org.apache.tomcat.maven
77
78
            <artifactId>tomcat7-maven-plugin</artifactId>
79
            <configuration>
              <!-- 指定端口 -->
20
              <port>8080</port>
81
              <!-- 请求路径 -->
82
83
              <path>/</path>
84
            </configuration>
85
          </plugin>
86
        </plugins>
      </build>
87
88
    </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"?>
1
 2
    <beans xmlns="http://www.springframework.org/schema/beans"</pre>
           xmlns:security="http://www.springframework.org/schema/security"
 3
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 4
           xsi:schemaLocation="http://www.springframework.org/schema/beans
 5
 6
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/security
 8
        http://www.springframework.org/schema/security/spring-security.xsd">
9
        <!-- 1、配置认证信息 -->
10
        <security:authentication-manager>
11
12
            <security:authentication-provider>
13
                <security:user-service>
                    <security:user name="admin" password="{noop}admin"</pre>
14
    authorities="ROLE_USER"/>
15
                </security:user-service>
16
            </security:authentication-provider>
17
        </security:authentication-manager>
18
        <!--
19
            2、配置授权信息
20
21
            配置拦截的规则
            auto-config="使用自带的页面"
22
23
            use-expressions="是否使用spel表达式",如果使用表达式: hasRole('ROLE USER')
24
        <security:http auto-config="true" use-expressions="false">
25
            <!-- 配置拦截的请求地址,任何请求地址都必须有ROLE_USER的权限 -->
26
            <security:intercept-url pattern="/**" access="ROLE_USER"/>
27
        </security:http>
29
30
    </beans>
```

4、配置web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
1
    <web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"</pre>
2
3
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
4
    http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
            version="3.1">
6
      <!--Spring监听器指定配置文件位置-->
      <context-param>
8
9
        <param-name>contextConfigLocation</param-name>
10
        <param-value>classpath:spring-security.xml</param-value>
      </context-param>
11
12
      tener>
        context.ContextLoaderListener/listener-
13
    classs
14
     </listener>
15
      <!--配置委派代理过滤器-->
      <filter>
17
18
        <filter-name>springSecurityFilterChain</filter-name>
19
        <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
20
      </filter>
21
      <filter-mapping>
        <filter-name>springSecurityFilterChain</filter-name>
22
23
        <url-pattern>/*</url-pattern>
      </filter-mapping>
24
25
26
    </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



Login with Username and Password

User:	
Password:	
Login	

6、配置自定义登录页面

```
<?xml version="1.0" encoding="UTF-8"?>
1
   <beans xmlns="http://www.springframework.org/schema/beans"</pre>
```

```
3
           xmlns:security="http://www.springframework.org/schema/security"
 4
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://www.springframework.org/schema/beans
 5
 6
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/security
        http://www.springframework.org/schema/security/spring-security.xsd">
 8
9
10
        <!-- 把登陆页面不拦截 -->
        <security:http pattern="/login.html" security="none"/>
11
12
        <security:http pattern="/error.html" security="none"/>
13
        <!--
14
            配置拦截的规则
15
            auto-config="使用自带的页面"
16
            use-expressions="是否使用spel表达式",如果使用表达式: hasRole('ROLE_USER')
18
        <security:http auto-config="true" use-expressions="false">
19
            <!-- 配置拦截的请求地址,任何请求地址都必须有ROLE_USER的权限 -->
20
21
            <security:intercept-url pattern="/**" access="ROLE_USER"/>
            <!-- 配置具体的页面跳转 -->
22
23
            <security:form-login</pre>
                    login-page="/login.html"
24
                    login-processing-url="/login"
25
                    default-target-url="/success.html"
26
                    authentication-failure-url="/error.html"
27
            />
28
29
            <!-- 关闭跨站请求伪造 -->
30
            <security:csrf disabled="true"/>
31
32
33
            <!-- 退出 -->
            <security:logout invalidate-session="true" logout-url="/logout" logout-</pre>
34
    success-url="/login.html"/>
35
        </security:http>
36
37
        <!-- 在内存中临时提供用户名和密码的数据 -->
38
39
        <security:authentication-manager>
            <security:authentication-provider>
40
                <security:user-service>
41
42
                    <security:user name="admin" password="{noop}admin"</pre>
    authorities="ROLE USER"/>
43
                </security:user-service>
44
            </security:authentication-provider>
45
        </security:authentication-manager>
46
    </beans>
```

7、login.html页面

```
1 <!DOCTYPE html>
2 <html>
```

```
3
  <head>
4
      <meta charset="UTF-8">
      <title>Insert title here</title>
  </head>
7
   <body>
   <h1>自定义的登录页面</h1>
8
   <form action="login" method="post">
9
10
      11
            用户名: 
12
13
            <input type="text" name="username" />
         14
15
         密码: 
16
17
            <input type="password" name="password" />
18
         19
20
            <input type="submit" value="登录" />
21
               <input type="reset" value="重置" />
22
         23
     24
  </form>
25
  </body>
26
  </html>
```

8、success.html页面

```
<!DOCTYPE html>
1
2
   <html>
   <head>
      <meta charset="UTF-8">
4
5
       <title>Insert title here</title>
   </head>
6
7
   <body>
   success html<br>
   <a href="logout">退出</a>
10
   </body>
   </html>
11
```

9、error.html页面

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"?>
1
2
    <beans xmlns="http://www.springframework.org/schema/beans"</pre>
           xmlns:security="http://www.springframework.org/schema/security"
 3
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4
           xsi:schemaLocation="http://www.springframework.org/schema/beans
 6
        http://www.springframework.org/schema/beans/spring-beans.xsd
 7
        http://www.springframework.org/schema/security
        http://www.springframework.org/schema/security/spring-security.xsd">
8
9
        <!-- 不拦截静态资源 -->
10
        <security:http pattern="/css/**" security="none"/>
11
        <security:http pattern="/img/**" security="none"/>
12
        <security:http pattern="/plugins/**" security="none"/>
13
14
        <!-- 把登陆页面不拦截 -->
15
        <security:http pattern="/login.jsp" security="none"/>
16
17
        <!--
18
19
            配置拦截的规则
            auto-config="使用自带的页面"
20
            use-expressions="是否使用spel表达式",如果使用表达式: hasRole('ROLE_USER')
21
22
        -->
        <security:http auto-config="true" use-expressions="false">
23
            <!-- 配置拦截的请求地址,任何请求地址都必须有ROLE USER的权限 -->
24
            <security:intercept-url pattern="/**" access="ROLE_USER"/>
25
26
            <!-- 配置具体的页面跳转 -->
27
            <security:form-login</pre>
                    login-page="/login.jsp"
28
29
                    login-processing-url="/login"
                    default-target-url="/index.jsp"
30
31
            />
32
            <!-- 关闭跨越请求 -->
33
            <security:csrf disabled="true"/>
34
35
36
            <!-- 退出 -->
            <security:logout invalidate-session="true" logout-url="/logout" logout-</pre>
37
    success-url="/login.html"/>
38
39
        </security:http>
40
        <!-- 在内存中临时提供用户名和密码的数据 -->
41
42
        <security:authentication-manager>
            <!-- 提供服务类,去数据库查询用户名和密码 -->
43
            <security:authentication-provider user-service-ref="userService"/>
44
45
        </security:authentication-manager>
46
47
    </beans>
```

2、修改web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
 1
    <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"</pre>
 2
    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">
 3
      <display-name>ssm web</display-name>
4
 5
      <!--1、配置spring的监听器-->
 6
 7
      <context-param>
 8
        <param-name>contextConfigLocation
9
        <param-value>classpath:applicationContext.xml,classpath:spring-
    security.xml
      </context-param>
10
      tener>
11
12
        tener-class>org.springframework.web.context.ContextLoaderListener/listener-
    class>
      </listener>
13
14
      <!--2、配置SptingMVC的前端控制器-->
15
16
      <servlet>
        <servlet-name>DispatcherServlet</servlet-name>
17
18
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
19
        <init-param>
20
          <param-name>contextConfigLocation</param-name>
21
          <param-value>classpath:springmvc.xml,classpath:spring-security.xml</param-</pre>
    value>
22
        </init-param>
        <!--服务器启动就创建该Servlet-->
23
        <load-on-startup>1</load-on-startup>
24
25
      </servlet>
26
      <servlet-mapping>
        <servlet-name>DispatcherServlet</servlet-name>
27
28
        <url-pattern>/</url-pattern>
29
      </servlet-mapping>
30
31
      <!--3、乱码的解决方案-->
32
33
        <filter-name>CharacterEncodingFilter</filter-name>
        <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-</pre>
34
    class>
35
        <init-param>
36
          <param-name>encoding</param-name>
37
          <param-value>UTF-8</param-value>
38
        </init-param>
      </filter>
39
40
      <filter-mapping>
41
        <filter-name>CharacterEncodingFilter</filter-name>
42
        <url-pattern>/*</url-pattern>
43
      </filter-mapping>
44
```

```
45
      <!--4、springsecurity委派过滤器-->
46
      <filter>
        <filter-name>springSecurityFilterChain</filter-name>
48
        <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
49
      </filter>
50
      <filter-mapping>
51
        <filter-name>springSecurityFilterChain</filter-name>
52
        <url-pattern>/*</url-pattern>
53
      </filter-mapping>
54
55
      <welcome-file-list>
56
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.htm</welcome-file>
57
        <welcome-file>index.jsp</welcome-file>
58
59
        <welcome-file>default.html</welcome-file>
60
        <welcome-file>default.htm</welcome-file>
        <welcome-file>default.jsp</welcome-file>
61
      </welcome-file-list>
62
63
    </web-app>
```

3、创建用户表和用户实体

```
CREATE TABLE sys_user(
1
2
     id BIGINT PRIMARY KEY AUTO INCREMENT,
3
     username VARCHAR(50),
4
     email VARCHAR(50),
5
     password VARCHAR(80),
6
     phoneNum VARCHAR(20),
7
     status INT
8
  );
```

```
1
    package com.itheima.domain;
 2
3
    public class SysUser {
 4
        private Long id;
 5
        private String username;
 6
        private String email;
8
        private String password;
9
        private String phoneNum;
        private int status;
10
11
        //省略getter和setter方法......
12
13
    }
14
```

4、修改spring-security.xml,修改认证方式为查询数据库

5、编写UserService业务代码

userService接口,接口集成UserDetailsService接口

```
package com.itheima.service;

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

public interface UserService extends UserDetailsService {
}
```

userServiceImpl实现

```
1
    package com.itheima.service.impl;
2
 3
   import com.itheima.dao.UserMapper;
   import com.itheima.domain.SysUser;
4
   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;
8
9
    import org.springframework.security.core.userdetails.User;
   import org.springframework.security.core.userdetails.UserDetails;
10
    import org.springframework.security.core.userdetails.UsernameNotFoundException;
11
    import org.springframework.stereotype.Service;
12
13
14
    import java.util.ArrayList;
    import java.util.List;
15
16
    @Service("userService")
17
18
    public class UserServiceImpl implements UserService {
19
20
        @Autowired
21
        private UserMapper userMapper;
22
23
        @Override
        public UserDetails loadUserByUsername(String username) throws
24
    UsernameNotFoundException {
            // 先设置假的权限
25
            List<GrantedAuthority> authorities = new ArrayList<>();
26
            authorities.add(new SimpleGrantedAuthority("ROLE USER"));
28
            // 通过用户名查询密码
            SysUser sysUser = userMapper.findByUsername(username);
29
            //判断认证是否成功
30
31
            if(sysUser!=null){
```

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

return user;
}

return null;
}

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 ,sqlserver ,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。