第五章 Shiro+SpringBoot整合开发

泽林.王峰 [2019.4.17 星期三]

授课目标

1、上章回顾

2、Shiro与SpringBoot整合工程搭建

3、配置文件的编写

4、自定义realm的编写

5、自定义异常处理

6、重写shiro过滤器完成验证码验证

7、"记住我"功能完成

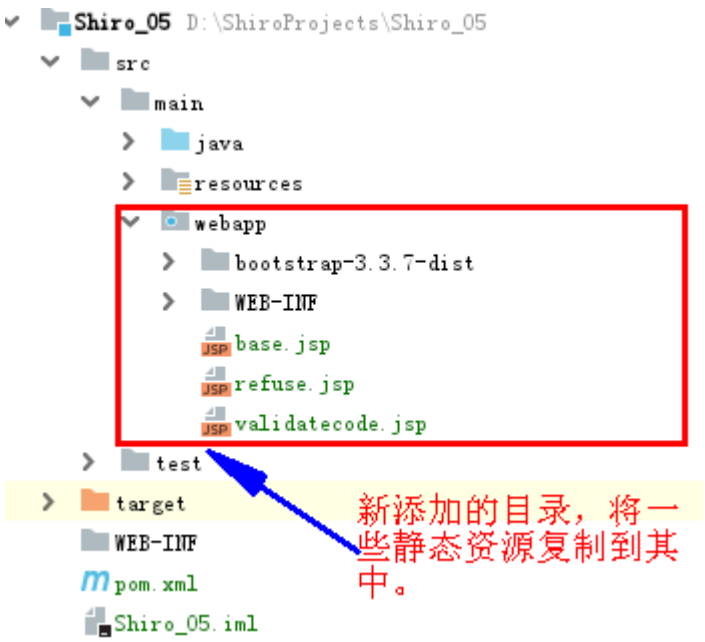
授课内容

1、上章回顾（略）

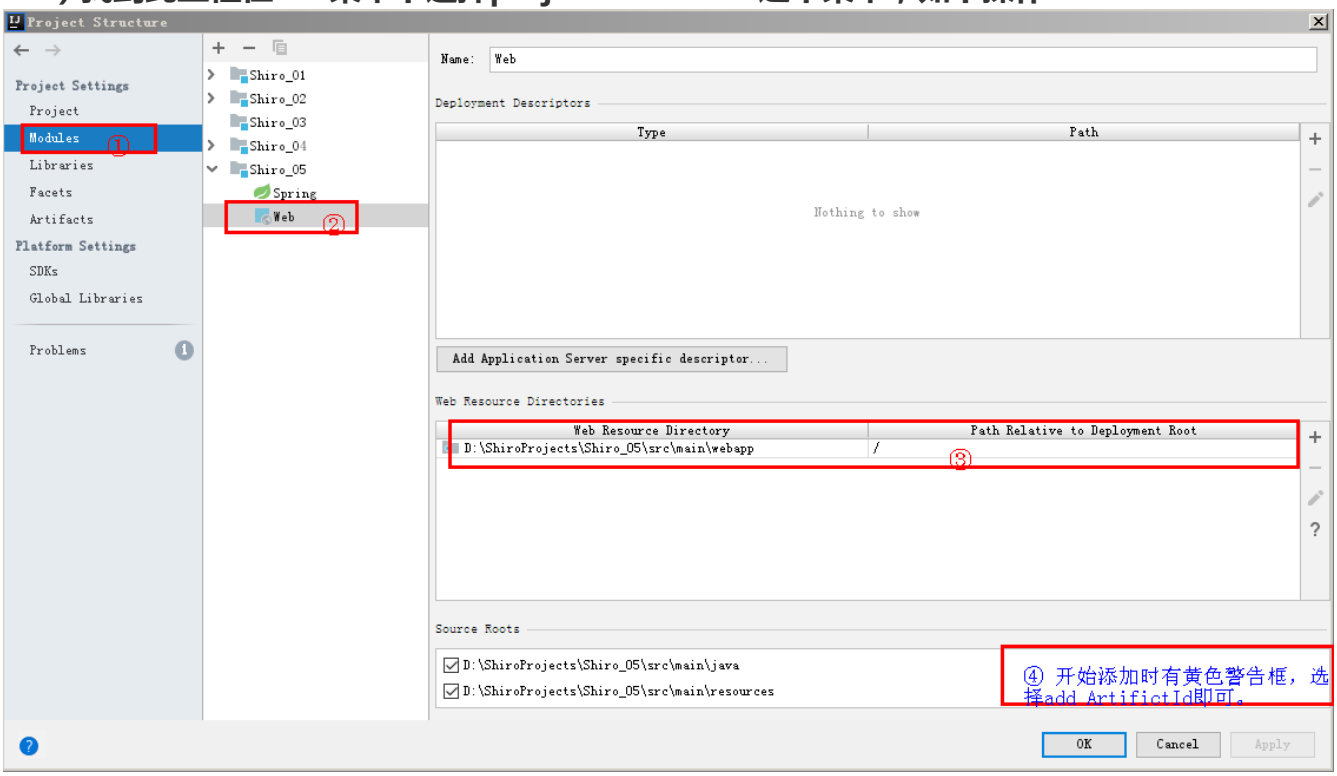
2、Shiro与SpringBoot整合工程搭建

2.1）在当前的工程下新建一个普通的maven工程Shiro_05。

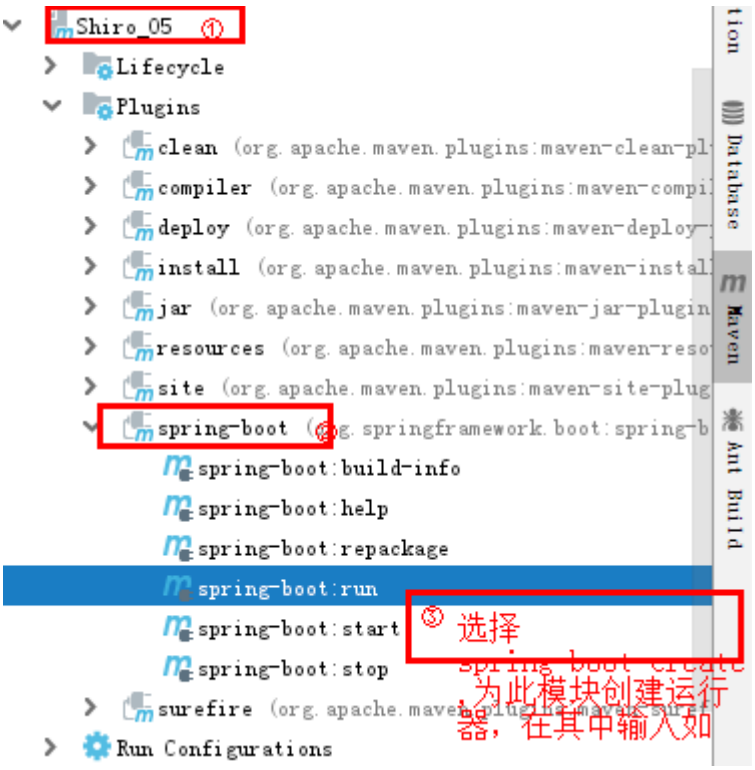
2.2）在Shiro_05工程下的src/main/添加webapp目录，如下图所示：



2.3）找到此工程在File菜单下选择project structure这个菜单，如下操作：



2.4）找到此工程在File菜单下选择project structure这个菜单，如下操作：



注意：上面的第③步输入的命令是: Springboot:run

2.5）在pom.xml文件中引入相关的依赖

```
<?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.zelin</groupId>
    <artifactId>Shiro_05</artifactId>
    <version>1.0-SNAPSHOT</version>
    <!--1.配置下面的依赖所用到的父工程-->
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.0.1.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <java.version>1.8</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
        </dependency>
        <!--2.与junit进行整合的starter-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <!--3.代表springboot与web的整合的starter-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <!--4.添加数据库-->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.47</version>
        </dependency>
        <!--5.添加mybatis与springboot整合的依赖-->
        <dependency>
            <groupId>org.mybatis.spring.boot</groupId>
            <artifactId>mybatis-spring-boot-starter</artifactId>
            <version>2.0.1</version>
        </dependency>
        <!-- 6.添加mybatis分页插件与springboot整合的依赖 -->
        <dependency>
            <groupId>com.github.pagehelper</groupId>
            <artifactId>pagehelper-spring-boot-starter</artifactId>
            <version>1.2.10</version>
        </dependency>
```

```
        <!-- 7.添加通过mapper与springboot整合的依赖 -->
        <dependency>
            <groupId>tk.mybatis</groupId>
            <artifactId>mapper-spring-boot-starter</artifactId>
            <version>2.1.5</version>
        </dependency>
        <!--8.添加druid连接池-->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid</artifactId>
            <version>1.0.14</version>
        </dependency>
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-lang3</artifactId>
            <version>3.3.2</version>
        </dependency>
        <!-- 关于shiro相关 -->
        <dependency>
            <groupId>org.apache.shiro</groupId>
            <artifactId>shiro-core</artifactId>
            <version>1.3.2</version>
        </dependency>
        <dependency>
            <groupId>org.apache.shiro</groupId>
            <artifactId>shiro-web</artifactId>
            <version>1.3.2</version>
        </dependency>
        <dependency>
            <groupId>org.apache.shiro</groupId>
            <artifactId>shiro-spring</artifactId>
            <version>1.3.2</version>
        </dependency>
        <dependency>
            <groupId>org.apache.shiro</groupId>
            <artifactId>shiro-ehcache</artifactId>
            <version>1.3.2</version>
        </dependency>
        <!--在springboot中添加热布署功能-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
        </dependency>
        <!--添加对jsp的依赖-->
        <dependency>
            <groupId>jstl</groupId>
            <artifactId>jstl</artifactId>
            <version>1.2</version>
        </dependency>
        <!-- 使用jsp引擎，springboot内置tomcat没有此依赖 -->
        <dependency>
            <groupId>org.apache.tomcat.embed</groupId>
            <artifactId>tomcat-embed-jasper</artifactId>
```

```
            <scope>provided</scope>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <!--如果使用springboot的热布署功能的话，需要添加如下的配置-->
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <fork>true</fork>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

2.6) 定义resources/application.yml文件

```
server:
  port: 9000
#配置spring相关
spring:
  datasource:
    driver-class-name: com.mysql.jdbc.Driver
    url: jdbc:mysql:///shiro
    username: root
    password: 123
  mvc:
    view:
      prefix: /WEB-INF/jsp/
      suffix: .jsp
  http:
    encoding:
      charset: UTF-8
#配置mybatis
mybatis:
  mapper-locations: mapper/*.xml
```

2.7) 定义SpringBoot的启动器：

```java
/**
 * @Author: Feng.Wang
 * @Company: Zelin.ShenZhen
 * @Description:
 * @Date: Create in 2019/4/17 09:29
 */
@SpringBootApplication
@MapperScan("com.zelin.mapper")
public class ShiroApplication {
    public static void main(String[] args) {
        SpringApplication.run(ShiroApplication.class);
    }
}
```

2.8) 定义异常与全局异常处理：

```java
/**
 * 自定义异常类
 */
public class MyException extends RuntimeException {
    private static final long serialVersionUID = 1L;
    //代表异常的错误信息
    private String message;
    public String getMessage() {
        return message;
    }
    public void setMessage(String message) {
        this.message = message;
    }
    public MyException() {
        super();
    }
    public MyException(String message) {
        super();
        this.message = message;
    }
}

```

```java
/**
 * @Author: Feng.Wang
 * @Company: Zelin.ShenZhen
 * @Description:
 * @Date: Create in 2019/4/16 09:56
 */
@ControllerAdvice
public class MyExceptionHandler {
    //定义异常处理器
    @ExceptionHandler(MyException.class)
    public String myException(MyException ex, Model model){
        //1.得到异常出错信息

```

```
        String message = ex.getMessage();
        //2.如果信息为null，就指定一个message值
        if(StringUtils.isEmpty(message)){
            message = "未知异常！";
        }
        model.addAttribute("message",message);
        //3.到错误页面进行处理(注意：这里的错误视图，不能为error，以免视图的错误视图冲突,报timestamp找不到异常！)
        return "errorpage";
    }
}
```

2.9）自定义realm：

```
package com.zelin.realm;

import com.zelin.pojo.SysPermission;
import com.zelin.pojo.SysUser;
import com.zelin.service.SysUserService;
import org.apache.shiro.authc.AuthenticationException;
import org.apache.shiro.authc.AuthenticationInfo;
import org.apache.shiro.authc.AuthenticationToken;
import org.apache.shiro.authc.SimpleAuthenticationInfo;
import org.apache.shiro.authz.AuthorizationInfo;
import org.apache.shiro.authz.SimpleAuthorizationInfo;
import org.apache.shiro.realm.AuthorizingRealm;
import org.apache.shiro.subject.PrincipalCollection;
import org.apache.shiro.util.ByteSource;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

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

/**
 * @Author: Feng.Wang
 * @Company: Zelin.ShenZhen
 * @Description:
 * @Date: Create in 2019/4/16 09:21
 */

public class CustomRealm extends AuthorizingRealm {
    @Autowired
    private SysUserService userService;
    //用户授权
        @Override
        protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {
            //1.取得主身份信息
            SysUser user = (SysUser) principals.getPrimaryPrincipal();
            List<SysPermission> permissions = null;
            if(null != user){
                System.out.println("开始查询数据库...");
```

```
                //2.取得此用户的权限列表
                permissions = userService.findPermissionsByUserCode(user.getUsercode());
                if(permissions != null && permissions.size() > 0){
                    user.setPermissions(permissions);
                }
            }
        //3.定义授权对象
        SimpleAuthorizationInfo authorizationInfo = new SimpleAuthorizationInfo();
        //4.将上面的用户的权限列表字符串赋值给当前的授权对象
        List<String> permissionList = new ArrayList<>();
        //5.转换上面的List<SysPermission>对象为List<String>对象
        for (SysPermission permission : permissions) {
            permissionList.add(permission.getPercode());
        }
        //6.将权限码列表与当前授权对象关联
        authorizationInfo.addStringPermissions(permissionList);
        //7.返回授权对象
        return authorizationInfo;
    }

    //用户认证
    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) throws AuthenticationException {
        //1.得到主身份信息
        String usercode = (String) token.getPrincipal();
        //2.如果用户名存在
        if(usercode != null){
            SysUser user = userService.findUserByUserCode(usercode);
            if(user == null) return null;
            //2.1)如果不为null，就查询其对应的菜单列表
            List<SysPermission> menus = userService.findMenusByUserCode(usercode);
            user.setMenus(menus);
            //2.2)取出数据库中的密码及加盐值
            String password = user.getPassword();
            String salt = user.getSalt();
            return new SimpleAuthenticationInfo(user,password,ByteSource.Util.bytes(salt),"aaa");
        }
       return null;
    }
}
```

2.10）重写表单过滤器，为了校验验证码：

```
/**
 * @Author: Feng.Wang
 * @Company: Zelin.ShenZhen
 * @Description: 自定义表单过滤器完成验证码的验证
 * @Date: Create in 2019/4/16 15:27
 */
public class CustomFormFilter extends FormAuthenticationFilter {
```

```
    @Override
    protected boolean onAccessDenied(ServletRequest request, ServletResponse response)
throws Exception {
        //1.将原始的请求与响应对象转换为基于http请求的对象
        HttpServletRequest req = (HttpServletRequest) request;
        HttpServletResponse resp = (HttpServletResponse) response;
        //2.从session中取出原始的验证码
        HttpSession session = req.getSession();
        String validateCode = (String) session.getAttribute("validateCode");
        //3.得到用户输入的验证码
        String validcode = req.getParameter("validcode");
        //4.比较两次验证码是否一样,如果两次验证码不相等，就将错误信息放到request对象中
        if(StringUtils.isNotEmpty(validcode) && !validateCode.equals(validcode)){
            req.setAttribute("shiroLoginFailure","validCodeError");
            return true;      //代表验证未通过
        }
        return super.onAccessDenied(request, response);
    }
}
```

2.11) 将pojo中的类实现序列化，否则，报错。

2.12) 定义登录的控制器login

```
/**
 * @Author: Feng.Wang
 * @Company: Zelin.ShenZhen
 * @Description:
 * @Date: Create in 2019/4/17 09:39
 */
@Controller
public class LogController {
    @RequestMapping("/login")
    //默认情况下，shiro的用户认证工作由FormAuthenticationFilter(authc)过滤器完成，
    //当出现异常时，shiro会将异常对象的类名放到以shiroLoginFailure为key的request
    //对象中，我们只需要通过此key取出对应的异常类名就可以知道，用户是何种异常
    public String login(HttpServletRequest request, HttpServletResponse response)
throws Exception {
        //1.得到异常的名称
        String exceptionName = (String) request.getAttribute("shiroLoginFailure");
        //2.根据异常的名称来判断执行的是何种异常，从而处理此异常
        //2.1)判断是否为null
        if(StringUtils.isNotEmpty(exceptionName)){
            if(UnknownAccountException.class.getName().equals(exceptionName)){
                throw new MyException("账户异常!");
            }else
if(IncorrectCredentialsException.class.getName().equals(exceptionName)){
                throw new MyException("用户名或密码输入有误!");
            }else if(UnauthorizedException.class.getName().equals(exceptionName)){
                throw new MyException("无权限访问异常!");
            }else if("validCodeError".equals(exceptionName)){
                throw new MyException("验证码错误！");
            }else{
```

```
                throw new Exception();
            }
        }
        //此方法只有在FormAuthenticationFilter认证失败时工作，认证成功返回页面就是上一次的页面
        return "login";
    }
}

```

2.13) 定义用户控制器

```
/**
 * @Author: Feng.Wang
 * @Company: Zelin.ShenZhen
 * @Description:
 * @Date: Create in 2019/4/17 10:53
 */
@Controller
@RequestMapping("/user")
public class UserController {
    @Autowired
    private SysUserService userService;

    /**
     * 根据登录用户动态显示菜单
     * @param model
     * @return
     */
    @RequestMapping("/listmenu")
    public String listmenu(Model model){
        try {
            //1.得到主体对象
            Subject subject = SecurityUtils.getSubject();
            //2.得到身份对象（当前登录的用户）
            SysUser user = (SysUser) subject.getPrincipal();
            //2.从此用户身上得到其有的菜单
            List<SysPermission> menus = user.getMenus();
            //3.将上面的用户菜单放到model中
            model.addAttribute("menus",menus);

        } catch (Exception e) {
            e.printStackTrace();
        }
        return "user/listmenu";
    }

    /**
     * 查询所有的用户
     * @return
     */
    @RequiresPermissions("user:userlistxxx")
    @RequestMapping("/userlist")
    public String  findAll(Model model){
```

```
        try {
             model.addAttribute("users",userService.findUsers());
        } catch (Exception e) {
            System.out.println("对不起，你无权访问此页面!");
        }
        return "user/list";
    }
}

```

2.14) 定义学生控制器

```
/**
 * @Author: Feng.Wang
 * @Company: Zelin.ShenZhen
 * @Description:
 * @Date: Create in 2019/4/8 10:50
 */
@Controller
@RequestMapping("/student")
public class StudentController {
    @Autowired
    private StudentService studentSerivce;
    @Autowired
    private ClassesService classesService;
    int pageSize = 5;       //代表每页的大小
    //查询所有的学生
    @RequestMapping("/tolist")
    @RequiresPermissions("student:tolist")   //说明具有student:tolist权限才能访问/tolist资源
    public String findAll(Model model){
        try {
            //1.查询所有的学生集合
            List<Student> students = studentSerivce.findStudents();
            System.out.println("----->" + students);
            //2.将上面的集合放到model中
            model.addAttribute("students",students);
            model.addAttribute("classes",classesService.findAll());
        } catch (Exception e) {
            e.printStackTrace();
        }
        //4.返回逻辑视图
        return "student/list";
    }
    /**
     * 查询所有学生带分页功能
     * @param page
     * @return
     */
    @RequiresPermissions("student:listpage")
    @RequestMapping("/listpage")
    public String listPage(@RequestParam(defaultValue = "1") int page,Model model){
        try {
```

```
            model.addAttribute("pr",studentSerivce.findAllStudents(page,pageSize));
            model.addAttribute("classes",classesService.findAll());
        } catch (Exception e) {
            e.printStackTrace();
        }
        return "student/list";
    }

    /**
     * 条件查询带分页
     * @param page
     * @param student
     * @return
     */
    @RequestMapping("/search")
    public String search(@RequestParam(defaultValue = "1") int page, Student student,
Model model){
        try {
            PageBean pageBean = studentSerivce.search(page,pageSize,student);
            model.addAttribute("pr",pageBean);
            model.addAttribute("page",page);
            model.addAttribute("classes",classesService.findAll());
        } catch (Exception e) {
            e.printStackTrace();
        }
        return "student/list";
    }
}
```

2.15) 定义学生列表页面/WEB-INF/jsp/student/list.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" isELIgnored="false" %>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@taglib prefix="shiro" uri="http://shiro.apache.org/tags" %>
<html>
<jsp:include page="/base.jsp"/>
<head>
    <meta charset="UTF-8">
    <style>
        .table {
            text-align: center;
        }
        .container {
            margin-top: 20px;
        }
        #form1 {
            margin-top: 10px;
            margin-left: 30px;
        }
    </style>
</head>
<body>
```

```
<div class="container">
    <div class="panel panel-primary">
        <div class="panel-heading">
            <h3 class="panel-title">学生查询</h3>
        </div>

        <form class="form-inline" id="form1" method="post"
              action="${pageContext.request.contextPath}/student/search.do">
            <input type="hidden" name="page" id="page">
            学生姓名：<input type="text" name="sname" value="${param.sname}"
placeholder="学生姓名关键字" class="form-control">
            学生住址：<input type="text" name="addr" value="${param.addr}"
placeholder="学生住址关键字" class="form-control">
            所在班级：
            <select name="cid" class="form-control">
                <option value="0">所有班级</option>
                <c:forEach items="${classes}" var="c">
                    <option value="${c.cid}"
${c.cid==param.cid?"selected":""}>${c.cname}</option>
                </c:forEach>
            </select>
            <input type="submit" value="查询" class="btn btn-default btn-sm">
            <%--如果有student:create这个权限，则下面的超链接就会显示出来--%>
            <shiro:hasPermission name="student:createxxx">
                <input type="button" value="添加学生" class="btn btn-success btn-sm" >
            </shiro:hasPermission>
            <a href="javascript:parent.location.href = '/logout.do'" class="btn btn-
success btn-sm">注销</a>
        </form>
        <table class="table table-bordered table-striped">
            <tr>
                <td>姓名</td>
                <td>性别</td>
                <td>年龄</td>
                <td>住址</td>
                <td>生日</td>
                <td>所在班级</td>
                <td>操作</td>
            </tr>
            <c:forEach items="${students}" var="stud">
                <tr>
                    <td>${stud.sname}</td>
                    <td>${stud.sex}</td>
                    <td>${stud.age}</td>
                    <td>${stud.addr}</td>
                    <td>
                        <fmt:formatDate value="${stud.birth}" pattern="yyyy-MM-dd E"/>
                    </td>
                    <td>
                            ${stud.classes.cname}
                    </td>
                    <td>
                        <a class="btn btn-primary btn-sm"
```

```
                href="${pageContext.request.contextPath}/student/toupdate.do?sid=${stud.sid}">修改</a>
                            <a class="btn btn-danger btn-sm"
                href="${pageContext.request.contextPath}/student/deleteBySid.do?sid=${stud.sid}"
                               onclick="return confirm('你真的要删除吗?')">删除</a>
                        </td>
                    </tr>
                </c:forEach>
            </table>
        </div>
    </div>
    <script>
        //执行提交 表单
        function skip(i) {
            //1.对表单中的隐藏域赋值
            $("#page").val(i);
            //2.提交表单
            $("#form1").submit();
        }

        //添加学生
        function addStudent() {
            location.href = "${pageContext.request.contextPath}/student/toadd.do";
        }

    </script>
    </body>
    </html>

```

2.16) 定义用户菜单显示/WEB-INF/jsp/user/listmenu.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@include file="/base.jsp"%>
<html>
<head>
    <title>用户登录</title>
    <style>
        .container{
            width: 500px;
            margin-top: 50px;
        }
        .form-signin{
            padding:5px;
        }
        .btn{
            margin-top: 20px;
        }
        .error{
            color:red;
        }
    </style>
```

```
</head>
<body>
<div class="container">
    <div class="panel panel-primary">
        <div class="panel-heading">
            <h3 class="panel-title">
                用户登录
            </h3>
        </div>
        <div class="panel-body">
        <form class="form-signin" action="${pageContext.request.contextPath}/login" method="post">
            <label>用户名</label>
            <input type="text"  name="username" class="form-control" placeholder="输入用户名" required autofocus>
            <label >密码</label>
            <input type="password" name="password" class="form-control" placeholder="输入密码" required>
            <label >输入验证码</label>
            <input type="text" name="validcode" style="width:360px" class="form-control" placeholder="输入验证码">
            <img src="${pageContext.request.contextPath}/validatecode.jsp" style="position: relative;top:-34px;left: 365px"/>
            <div style="padding:0;margin-top: -20px;">
                <input type="checkbox" name="rememberMe"> 记住我
            </div>
            <button class="btn btn-lg btn-primary btn-block" type="submit">登录</button>
            <span class="error">${message}</span>
        </form>
    </div>
    </div>
</div> <!-- /container -->
</body>
</html>

```

2.17) 定义出现异常时处理页面/WEB-INF/jsp/user/errorpage.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>异常处理页面</title>
</head>
<body>
    <h1>异常信息：<hr></h1>
    <h2>${message}</h2>
</body>
</html>
```

2.18) 定义未授权时访问的页面/WEB-INF/jsp/refuse.jsp

```jsp
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Access Control</title>
</head>
<body>
    <h2>无访问权限！</h2>
</body>
</html>
```

2.19）用户列表页面/WEB-INF/jsp/user/list.jsp

```jsp
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>用户列表</title>
</head>
<body>
    ${users}
</body>
</html>

```

2.20）定义缓存文件resources/cache/shiro-cache.xml

```xml
<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="../config/ehcache.xsd">
    <!--diskStore：缓存数据持久化的目录 地址  -->
    <diskStore path="e:\develop\ehcache" />
    <defaultCache
            maxElementsInMemory="1000"
            maxElementsOnDisk="10000000"
            eternal="false"
            overflowToDisk="false"
            diskPersistent="false"
            timeToIdleSeconds="120"
            timeToLiveSeconds="120"
            diskExpiryThreadIntervalSeconds="120"
            memoryStoreEvictionPolicy="LRU">
    </defaultCache>
    <!--name：cache的名字，用来识别不同的cache，必须惟一。-->
    <!--maxElementsInMemory：内存管理的缓存元素数量最大限值。-->
    <!--maxElementsOnDisk：硬盘管理的缓存元素数量最大限值。默认值为0，就是没有限制。-->
    <!--eternal：设定元素是否持久话。若设为true，则缓存元素不会过期。-->
    <!--overflowToDisk：设定是否在内存填满的时候把数据转到磁盘上。-->
</ehcache>
```

2.21）在webapp这个目录下定义validatecode.jsp这个生成验证码的jsp文件

```jsp
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="java.util.Random"%>
<%@ page import="java.io.OutputStream"%>
<%@ page import="java.awt.Color"%>
<%@ page import="java.awt.Font"%>
<%@ page import="java.awt.Graphics"%>
<%@ page import="java.awt.image.BufferedImage"%>
<%@ page import="javax.imageio.ImageIO"%>
<%
    int width = 60;
    int height = 32;
    //create the image
    BufferedImage image = new BufferedImage(width, height,
BufferedImage.TYPE_INT_RGB);
    Graphics g = image.getGraphics();
    // set the background color
    g.setColor(new Color(0xDCDCDC));
    g.fillRect(0, 0, width, height);
    // draw the border

    g.setColor(new Color(0x337ab7));
    g.drawRect(0, 0, width - 1, height - 1);
    // create a random instance to generate the codes
    Random rdm = new Random();
    String hash1 = Integer.toHexString(rdm.nextInt());
    System.out.print(hash1);
    // make some confusion
    for (int i = 0; i < 50; i++) {
        int x = rdm.nextInt(width);
        int y = rdm.nextInt(height);
        g.drawOval(x, y, 0, 0);
    }
    // generate a random code
    String capstr = hash1.substring(0, 4);
    //将生成的验证码存入session
    session.setAttribute("validateCode", capstr);
    g.setColor(new Color(0, 100, 0));
    g.setFont(new Font("Candara", Font.BOLD, 24));
    g.drawString(capstr, 8, 24);
    g.dispose();
    //输出图片
    response.setContentType("image/jpeg");
    out.clear();
    out = pageContext.pushBody();
    OutputStream strm = response.getOutputStream();
    ImageIO.write(image, "jpeg", strm);
    strm.close();
%>
```

3、本工程中关于Shiro的配置说明：

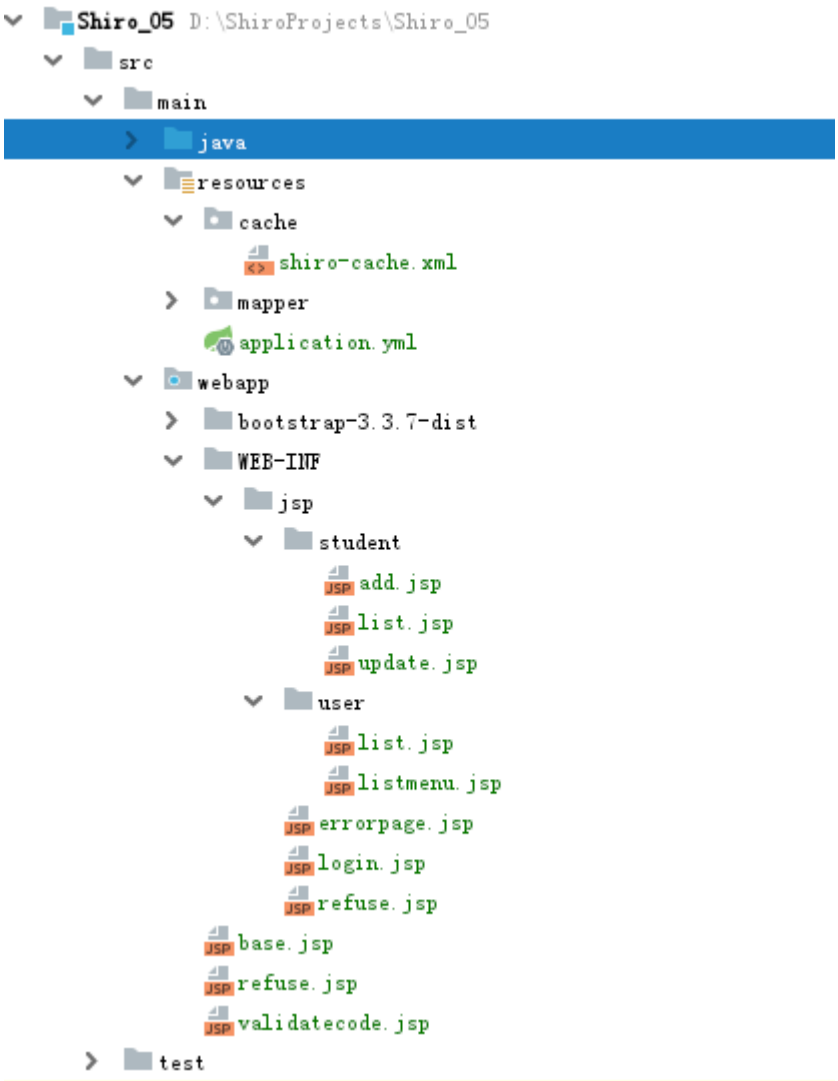
3.1）定义Shiro的相关配置信息，com.zelin.config包下

```java
/**
 * @Author: Feng.Wang
 * @Company: Zelin.ShenZhen
 * @Description: 配置ShiroFactoryFilterBean
 * @Date: Create in 2019/4/17 09:28
 */
@Configuration
public class ShiroConfig {
    @Bean(name = "shiroFilter")
    public ShiroFilterFactoryBean shiroFilterFactoryBean(SecurityManager securityManager){
        //1.定义ShiroFilterFactoryBean对象
        ShiroFilterFactoryBean shiroFilter = new ShiroFilterFactoryBean();
        //2.将securityManager与ShiroFilterFactoryBean对象绑定
        shiroFilter.setSecurityManager(securityManager);
        //3.设置一些其它的属性
        shiroFilter.setLoginUrl("/login");
        shiroFilter.setSuccessUrl("/user/listmenu");
        shiroFilter.setUnauthorizedUrl("/refuse.jsp");
        //4.设置自定义的过滤器
        Map<String, Filter> filters = new LinkedHashMap<String, Filter>();
        filters.put("authc", new CustomFormFilter());
        shiroFilter.setFilters(filters);
        //5.定义当前工程中使用的一系列的过滤器链(一定要使用LinkedHashMap，因为这些过滤器必须保证是有序的)
        Map<String, String> filterChainDefinitionMap = new LinkedHashMap<String, String>();

//        filterChainDefinitionMap.put("/**/*.js", "anon");
//        filterChainDefinitionMap.put("/**/*.png", "anon");
//        filterChainDefinitionMap.put("/**/*.jpg", "anon");
//        filterChainDefinitionMap.put("/**/*.css", "anon");
//        filterChainDefinitionMap.put("/**/*.map", "anon");
        //上面的配置，在本工程的情况下，可以使用下面的来代替
        filterChainDefinitionMap.put("/bootstrap-3.3.7-dist/**/**","anon");

        filterChainDefinitionMap.put("/validatecode.jsp", "anon");
        filterChainDefinitionMap.put("/refuse.jsp", "anon");
        filterChainDefinitionMap.put("/logout", "logout");
        filterChainDefinitionMap.put("/**", "authc");
        shiroFilter.setFilterChainDefinitionMap(filterChainDefinitionMap);
        return shiroFilter;
    }
    //配置安全管理器
    @Bean
    public SecurityManager securityManager(CustomRealm customRealm){
        DefaultWebSecurityManager securityManager = new DefaultWebSecurityManager();
        //添加自定义realm
        securityManager.setRealm(customRealm);
        //添加缓存管理器
        securityManager.setCacheManager(cacheManager());

        return securityManager;
```

```
51      }
52      //配置自定义realm
53      @Bean
54      public CustomRealm customRealm(HashedCredentialsMatcher matcher){
55          CustomRealm customRealm = new CustomRealm();
56          customRealm.setCredentialsMatcher(matcher);
57          return customRealm;
58      }
59      //配置凭证匹配器
60      @Bean
61      public HashedCredentialsMatcher hashedCredentialsMatcher(){
62          HashedCredentialsMatcher matcher = new HashedCredentialsMatcher();
63          matcher.setHashAlgorithmName("md5");
64          matcher.setHashIterations(1);
65          return matcher;
66      }
67      //配置缓存管理器
68      @Bean
69      public CacheManager cacheManager(){
70          EhCacheManager cacheManager = new EhCacheManager();
71          cacheManager.setCacheManagerConfigFile("classpath:cache/shiro-cache.xml");
72          return cacheManager;
73      }
74      //自定义过滤器完成验证码验证
75      @Bean
76      public CustomFormFilter customFormFilter(){
77          return new CustomFormFilter();
78      }
79      /**
80       * 配置shiro跟spring的关联
81       * 以下AuthorizationAttributeSourceAdvisor,DefaultAdvisorAutoProxyCreator两个类是为
    了支持shiro注解
82       * @param securityManager
83       * @return
84       */
85      @Bean
86      public AuthorizationAttributeSourceAdvisor
    authorizationAttributeSourceAdvisor(SecurityManager securityManager) {
87          AuthorizationAttributeSourceAdvisor advisor = new
    AuthorizationAttributeSourceAdvisor();
88          advisor.setSecurityManager(securityManager);
89          return advisor;
90      }
91
92      /**
93       * Spring的一个bean ， 由Advisor决定对哪些类的方法进行AOP代理
94       * @return
95       */
96      @Bean
97      public DefaultAdvisorAutoProxyCreator defaultAdvisorAutoProxyCreator() {
98          DefaultAdvisorAutoProxyCreator creator = new
    DefaultAdvisorAutoProxyCreator();
99          creator.setProxyTargetClass(true);
```

```
        return creator;
    }

    /**
     * lifecycleBeanPostProcessor是负责生命周期的 ， 初始化和销毁的类
     * （可选）
     * @return
     */
    @Bean
    public LifecycleBeanPostProcessor lifecycleBeanPostProcessor() {
        return new LifecycleBeanPostProcessor();
    }
    @Bean
    public SimpleMappingExceptionResolver createSimpleMappingExceptionResolver() {
        SimpleMappingExceptionResolver exceptionResolver = new
SimpleMappingExceptionResolver();
        //定义Properties对象，用于存放处理简单异常的属性
        Properties mappings = new Properties();
        mappings.setProperty("UnauthorizedException","refuse");   //参数1：要处理的异常
参数2：跳转的视图
        mappings.setProperty("UnauthenticatedException","refuse");
        exceptionResolver.setExceptionMappings(mappings);  // None by default
        exceptionResolver.setDefaultErrorView("refuse");    // No default
        exceptionResolver.setExceptionAttribute("ex");     // Default is "exception"
        return exceptionResolver;
    }
}
```

3.2) 本工程的截图如下：



4、“记住我”功能的实现：

4.1) /WEB-INF/jsp/login.jsp页面

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@include file="/base.jsp"%>
<html>
<head>
    <title>用户登录</title>
    <style>
        .container{
            width: 500px;
            margin-top: 50px;
        }
        .form-signin{
            padding:5px;
        }
        .btn{
            margin-top: 20px;
        }
        .error{
            color:red;
```

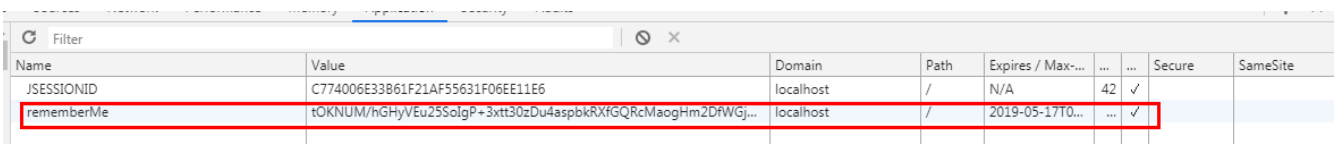
```
        }
    </style>
</head>
<body>
<div class="container">
    <div class="panel panel-primary">
        <div class="panel-heading">
            <h3 class="panel-title">
                用户登录
            </h3>
        </div>
        <div class="panel-body">
        <form class="form-signin" action="${pageContext.request.contextPath}/login" method="post">
            <label>用户名</label>
            <input type="text"  name="username" class="form-control" placeholder="输入用户名" required autofocus>
            <label >密码</label>
            <input type="password" name="password" class="form-control" placeholder="输入密码" required>
            <label >输入验证码</label>
            <input type="text" name="validcode" style="width:360px" class="form-control" placeholder="输入验证码">
            <img src="${pageContext.request.contextPath}/validatecode.jsp" style="position: relative;top:-34px;left: 365px"/>
            <div style="padding:0;margin-top: -20px;">
                <input type="checkbox" name="rememberMe"> 记住我
            </div>
            <button class="btn btn-lg btn-primary btn-block" type="submit">登录</button>
            <span class="error">${message}</span>
        </form>
    </div>
    </div>
</div> <!-- /container -->
</body>
</html>

```

4.2) 在com.zelin.config/ShiroConfig.java配置文件中添加如下内容：

```
@Configuration
public class ShiroConfig {
    @Bean(name = "shiroFilter")
    public ShiroFilterFactoryBean shiroFilterFactoryBean(SecurityManager securityManager){
         ShiroFilterFactoryBean shiroFilter = new ShiroFilterFactoryBean();
        //2.将securityManager与ShiroFilterFactoryBean对象绑定
        shiroFilter.setSecurityManager(securityManager);
        //3.设置一些其它的属性
        shiroFilter.setLoginUrl("/login");
        shiroFilter.setSuccessUrl("/user/listmenu");
        shiroFilter.setUnauthorizedUrl("/refuse.jsp");
```

```
        ...
        filterChainDefinitionMap.put("/logout", "logout");
        //添加“记住我”要放行的资源
        filterChainDefinitionMap.put("/user/**","user");
        filterChainDefinitionMap.put("/student/**","user");
        //对所有的资源进行拦截(需要认证才能访问)
        filterChainDefinitionMap.put("/**", "authc");
    }
    ...
    //配置安全管理器
    @Bean
    public SecurityManager securityManager(CustomRealm customRealm,RememberMeManager
rememberMeManager){
        DefaultWebSecurityManager securityManager = new DefaultWebSecurityManager();
        //添加自定义realm
        securityManager.setRealm(customRealm);
        //添加缓存管理器
        securityManager.setCacheManager(cacheManager());
        //添加rememberMeManager
        securityManager.setRememberMeManager(rememberMeManager);
        return securityManager;
    }
    ...
    //定义SimpleCookie用于定义rememberMeManager
    @Bean
    public SimpleCookie simpleCookie(){
        SimpleCookie simpleCookie = new SimpleCookie();
        simpleCookie.setMaxAge(2592000);
        simpleCookie.setName("rememberMe");
        return simpleCookie;
    }
    //定义rememberMeManager
    @Bean
    public RememberMeManager rememberMeManager(SimpleCookie simpleCookie){
        CookieRememberMeManager rememberMeManager = new CookieRememberMeManager();
        rememberMeManager.setCookie(simpleCookie);
        return rememberMeManager;
    }
}
```

4.3）测试，正常登录，再访问/user/listmenu资源，出现如下结果：




localhost:9000/user/listmenu
192.168.1.1
全栈开发实战：用V…
一份不错的vue.js…
Vue.js——60分钟…
Vue.js——60分钟…
vue.js快速开发详…
学习Vue.js:介绍-…
GitHub - keepfool…
用户菜单
学生管理
用户管理
分配角色
分配权限
学生查询
学生姓名：
学生姓名关键字
学生住址：
学生住址关键字
所在班级：
所有班级
查询
注销
姓名 性别 年龄 住址 生日 所在班级 操作
张三 男 20 上海 1995-07-13 星期四 1301班 修改 删除
五二 男 28 广州 2000-07-20 星期四 1301班 修改 删除
小红 女 19 杭州 1995-07-13 星期四 1302班 修改 删除
王二小 男 12 岳阳 1986-06-11 星期三 1302班 修改 删除
小2添 女 18 襄阳 1989-03-23 星期四 1302班 修改 删除