参考:

https://blog.csdn.net/yuanlaijike/article/details/80249235

前言

安全管理一直是任何系统都比较关注的内容，一个成熟的系统必然是在搭建的第一天就将安全考虑进来。那么我们所说的安全问题主要涉及啥呢，用户认证以及用户授权，通俗的说就是登录和权限。以前也涉及过通过shiro这样的框架实现，更是经历过自己写拦截器，过滤器的时代，而这边的Spring Security是一款基于Spring的安全框架，对于现在Java web开发已经几乎是绑定Spring的现状来说，无疑Spring Security将是一款最贴合现在web开发的安全管理框架。

Demo

首先，这边是基于有一个Spring boot的基础web项目（搭建见其它笔记）的情况下继续。

首先我们得在pom中引入Spring Security的支持。

```
<!-- Spring Security支持 -->
<dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

我们先来把准备工作做完，要实现用户认证，我们可以涉及两个方面，用户以及角色，这两者是多对多的关系，先将这边的表结构定义完。

```
/*
Navicat MySQL Data Transfer

Source Server         : xiaoyezi
Source Server Version : 50720
Source Host           : localhost:3306
```

```
Source Database       : test

Target Server Type    : MYSQL
Target Server Version : 50720
File Encoding         : 65001

Date: 2019-01-16 11:29:52
*/

SET FOREIGN_KEY_CHECKS=0;

-- ----------------------------
-- Table structure for sys_role
-- ----------------------------
DROP TABLE IF EXISTS `sys_role`;
CREATE TABLE `sys_role` (
  `id` int(11) NOT NULL,
  `name` varchar(255) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- ----------------------------
-- Records of sys_role
-- ----------------------------
INSERT INTO `sys_role` VALUES ('1', 'ROLE_ADMIN');
INSERT INTO `sys_role` VALUES ('2', 'ROLE_USER');

-- ----------------------------
-- Table structure for sys_user
-- ----------------------------
DROP TABLE IF EXISTS `sys_user`;
CREATE TABLE `sys_user` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(255) NOT NULL,
  `password` varchar(255) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8;

-- ----------------------------
-- Records of sys_user
-- ----------------------------
INSERT INTO `sys_user` VALUES ('1', 'admin', '123');
INSERT INTO `sys_user` VALUES ('2', 'xiaoming', '123');

-- ----------------------------
-- Table structure for sys_user_role
-- ----------------------------
DROP TABLE IF EXISTS `sys_user_role`;
CREATE TABLE `sys_user_role` (
  `user_id` int(11) NOT NULL,
  `role_id` int(11) NOT NULL,
  PRIMARY KEY (`user_id`,`role_id`),
```

```
  KEY `fk_role_id` (`role_id`),
  CONSTRAINT `fk_role_id` FOREIGN KEY (`role_id`) REFERENCES `sys_role` (`id`) ON DELETE
CASCADE ON UPDATE CASCADE,
  CONSTRAINT `fk_user_id` FOREIGN KEY (`user_id`) REFERENCES `sys_user` (`id`) ON DELETE
CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- ----------------------------
-- Records of sys_user_role
-- ----------------------------
INSERT INTO `sys_user_role` VALUES ('1', '1');
INSERT INTO `sys_user_role` VALUES ('2', '2');
```

上述脚本测试数据也建完，直接用mybatisgenerator生成相关的内容，这个就不贴代码了。

我们来看看关键处的代码，我们的Spring Security配置类。

```
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private CutomerUserDetailService userDetailService;

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(userDetailService).passwordEncoder(new
PasswordEncoder() {
            //密码加密，由各自系统决定，该处明文
            @Override
            public String encode(CharSequence charSequence) {
                return charSequence.toString();
            }

            @Override
            public boolean matches(CharSequence charSequence, String s) {
                return s.equals(charSequence.toString());
            }
        });
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
                .antMatchers("/", "/home").permitAll() //定义不需要拦截的路径
            .anyRequest().authenticated().and()
            .formLogin().loginPage("/login")//设值登录页
            .permitAll().and().logout().permitAll();
    }
```

```
    // @Autowired
    // public void configureGlobal(AuthenticationManagerBuilder auth) throws
    // Exception {
    // auth
    // .inMemoryAuthentication()
    // .withUser("user").password("password").roles("USER");
    // }

    @Override
    public void configure(WebSecurity web) throws Exception {
        // 设置拦截忽略文件夹，可以对静态资源放行
        web.ignoring().antMatchers("/css/**", "/js/**");
    }

}
```

该类的三个注解分别是标识该类是配置类、开启 Security 服务、开启全局 Securtiy 注解。该类注入的CutomerUserDetailService 也是待会要介绍的另一个重要的类，直接贴代码。

```
@Service("userDetailsService")
public class CutomerUserDetailService implements UserDetailsService {

    @Autowired
    private SysRoleMapper sysRoleMapper;

    @Autowired
    private SysUserMapper sysUserMapper;

    @Autowired
    private SysUserRoleMapper sysUserRoleMapper;

    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        List<GrantedAuthority> authorities = new ArrayList<>();

        SysUserExample sysUserExample = new SysUserExample();
        sysUserExample.createCriteria().andNameEqualTo(username);
        SysUser user = sysUserMapper.selectByExample(sysUserExample).get(0);

        if (user == null) {
            throw new UsernameNotFoundException("用户名不存在");
        }

        SysUserRoleExample sysUserRoleExample = new SysUserRoleExample();
        sysUserRoleExample.createCriteria().andUserIdEqualTo(user.getId());
        List<SysUserRoleKey> userRoleKeys =
sysUserRoleMapper.selectByExample(sysUserRoleExample);
```

```
            for (SysUserRoleKey userRoleKey : userRoleKeys) {
                SysRole role = sysRoleMapper.selectByPrimaryKey(userRoleKey.getRoleId());
                authorities.add(new SimpleGrantedAuthority(role.getName()));
            }

            return new User(user.getName(), user.getPassword(), authorities);
        }

}
```

通读一下代码逻辑应该清晰，UserDetailsService 其实Security提供了的默认的实现，但是肯定不会满足需求，所以一般我们会自己实现然后像WebSecurityConfig里一样使用。我们分析下这边干了啥，传入了username，然后查到了用户名，密码以及用户对应的角色们然后塞入Security提供的User类中。

紧接着就是我们得到控制器类以及页面，一起贴出。

```
@Controller
public class DemoController {

    @GetMapping("/")
    public String index() {
        return "index";
    }

    @GetMapping("/{url}")
    public String goThat(@PathVariable String url) {
        return url;
    }

    @GetMapping("/admin")
    @ResponseBody
    @PreAuthorize("hasRole('ROLE_ADMIN')")
    public String printAdmin() {
        return "如果你看见这句话，说明你有ROLE_ADMIN角色";
    }

    @GetMapping("/user")
    @ResponseBody
    @PreAuthorize("hasRole('ROLE_USER')")
    public String printUser() {
        return "如果你看见这句话，说明你有ROLE_USER角色";
    }
}
```

@PreAuthorize这个注解的作用就是在当前方法前执行，作用大概也能猜出，判断角色，不是不会执行后续方法。这就是配置中@EnableGlobalMethodSecurity(prePostEnabled = true)的含义，该处注解能生效都是这个的功劳。

index.html

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org"
      xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
<head>
<title>Spring Security入门</title>
</head>
<body>
      <h1>欢迎使用Spring Security!</h1>
      <p>
            点击 <a th:href="@{/hello}">这里</a> 打个招呼吧
      </p>
</body>
</html>
```

hello.html

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org"
      xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
<head>
<title>Hello World!</title>
</head>
<body>
      <a th:href="@{/admin}">检测ROLE_ADMIN角色</a>
   <a th:href="@{/user}">检测ROLE_USER角色</a>
      <form th:action="@{/logout}" method="post">
            <input type="submit" value="注销" />
      </form>
</body>
</html>
```

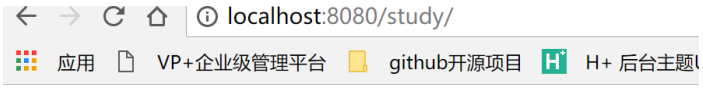
login.html

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org"
      xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
<head>
<title>Spring Security Example</title>
</head>
<body>
      <div th:if="${param.error}">用户名或密码错</div>
```

```
    <div th:if="${param.logout}">您已注销成功</div>
    <form th:action="@{/login}" method="post">
        <div>
            <label> 用户名 : <input type="text" name="username" />
            </label>
        </div>
        <div>
            <label> 密 码 : <input type="password" name="password" />
            </label>
        </div>
        <div>
            <input type="submit" value="登录" />
        </div>
    </form>
</body>
</html>
```

此处action的路径"/login"以及"username"和"password"最好不要改动，Spring Security默认识别这些（怎么不默认等我慢慢研究）。

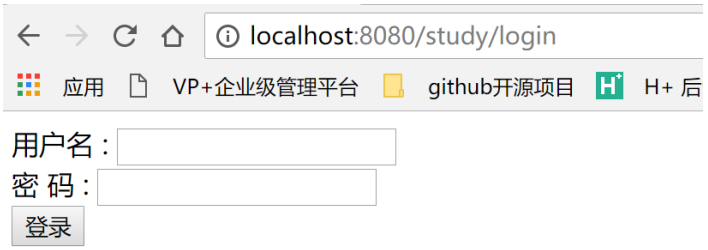
效果嘛，我慢慢来。



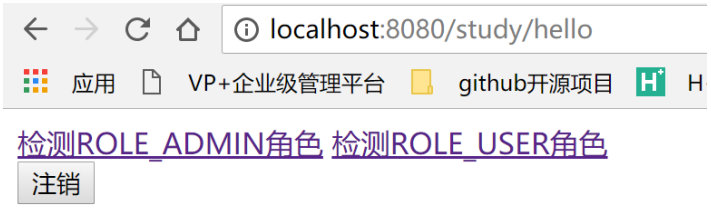
欢迎使用Spring Security!

点击 这里 打个招呼吧

点击应该是要访问hello，但没登录（显然hello不在访问白名单）这时候会去login。



我们拿测试数据的admin试试。



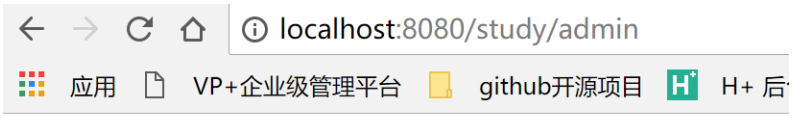
我们试试这两个检测，admin的角色很明显是ROLE_ADMIN，我们点后一个。

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Wed Jan 16 15:27:29 CST 2019
There was an unexpected error (type=Forbidden, status=403).
不允许访问

果然不行，我们返回点前一个。



如果你看见这句话，说明你有ROLE_ADMIN角色

这就ok了。