本章学习目标

- Shiro 核心功能
- Shiro 的 Web 集成
- Spring 与 Shiro 整合
- SpringBoot 整合 Shiro

1. Shiro 核心功能

1.1. RBAC 模型

在 RBAC 的模型，涉及到三个关键的元素：

1）用户：系统的使用用户（登录用户）

2）角色：拥有相同的权限的用户

3）权限：系统可以被用户操作的元素（例如：系统菜单，超链接，文件等）

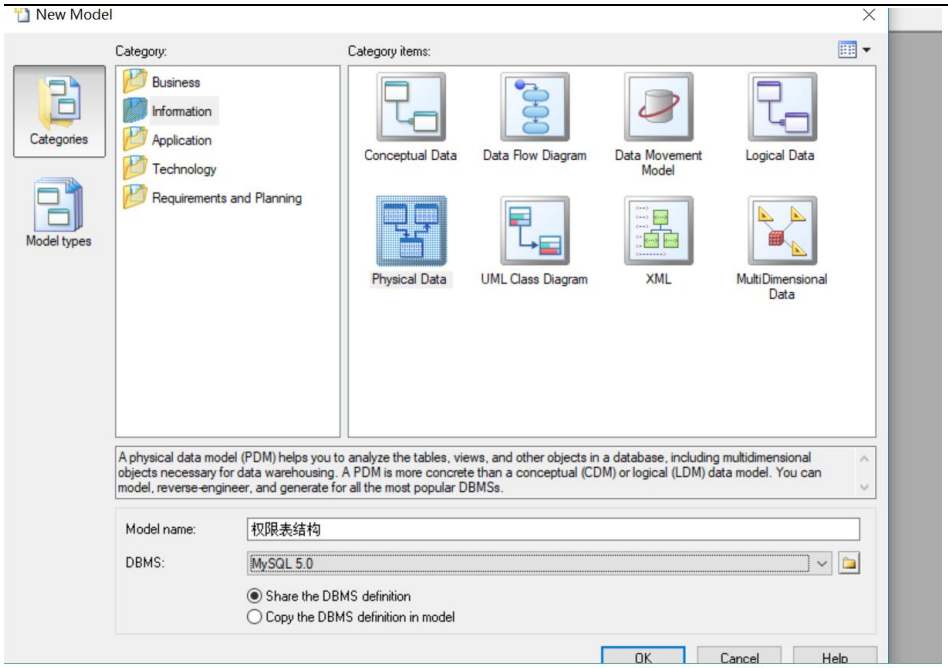
以上三个元素有一定的关系：

1）用户 和 角色 是多对多的关系

2）角色 和 权限 是多对多的关系

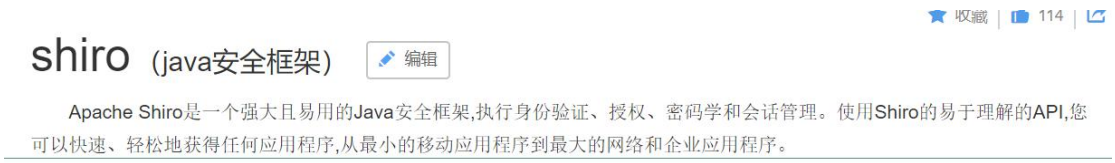
1.2. 设计权限表结构

使用 PowerDesigner

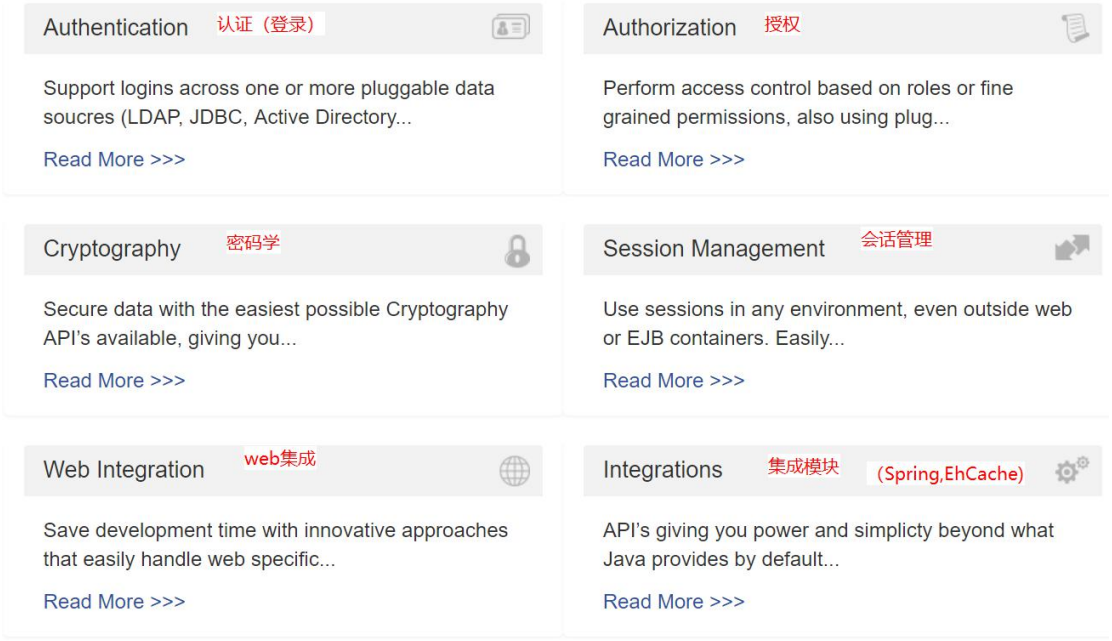


1.3. Shiro 框架简介

Shiro 是 Apache 组织的开源的 Java 安全框架。



Shiro 的 6 个核心功能：



其中，认证与授权是 Shiro 的基础核心功能。

1.4. Shiro 的三大核心 API

三个核心组件：Subject, SecurityManager 和 Realms.

Subject：即“当前操作用户”。但是，在Shiro中，Subject这一概念并不仅仅指人，也可以是第三方进程、后台帐户（Daemon Account）或其他类似事物。它仅仅意味着“当前跟软件交互的东西”。但考虑到大多数目的和用途，你可以把它认为是Shiro的“用户”概念。

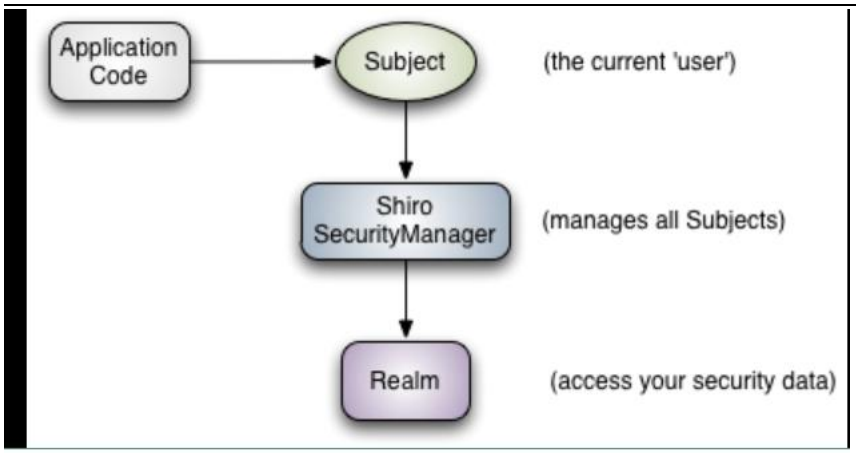
Subject代表了当前用户的安全操作，SecurityManager则管理所有用户的安全操作。

SecurityManager：它是Shiro框架的核心，典型的Facade模式，Shiro通过SecurityManager来管理内部组件实例，并通过它来提供安全管理的各种服务。

Realm： Realm充当了Shiro与应用安全数据间的“桥梁”或者“连接器”。也就是说，当对用户执行认证（登录）和授权（访问控制）验证时，Shiro会从应用配置的Realm中查找用户及其权限信息。

从这个意义上讲，Realm实质上是一个安全相关的DAO：它封装了数据源的连接细节，并在需要时将相关数据提供给Shiro。当配置Shiro时，你必须至少指定一个Realm，用于认证和（或）授权。配置多个Realm是可以的，但是至少需要一个。

Shiro内置了可以连接大量安全数据源（又名目录）的Realm，如LDAP、关系数据库（JDBC）、类似INI的文本配置资源以及属性文件等。如果缺省的Realm不能满足需求，你还可以插入代表自定义数据源的自己的Realm实现。

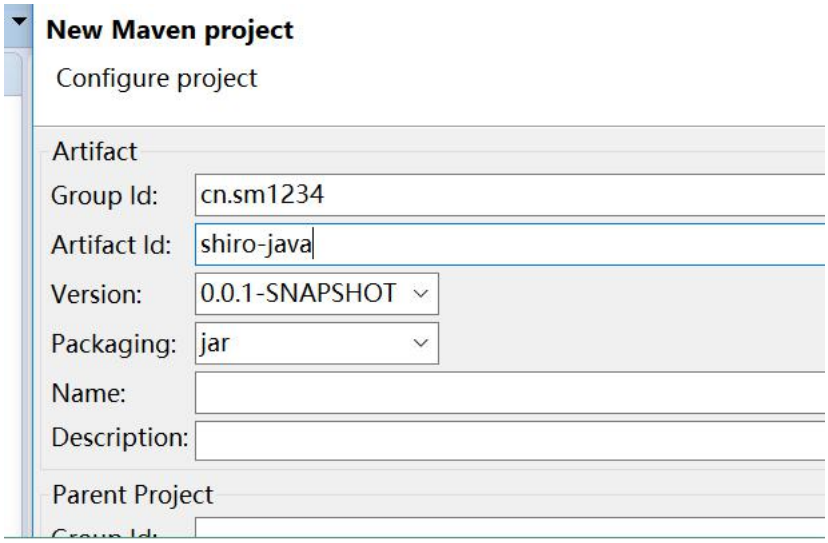


1）Subject 关联 SecurityManager

2）SecurityManager 关联 Realm

1.5. Shiro 的认证操作

1.5.1. 建立 maven 项目，导入 shiro 的坐标




```
<!-- 导入 shiro 的坐标 -->
        <dependency>
            <groupId>org.apache.shiro</groupId>
            <artifactId>shiro-all</artifactId>
            <version>1.3.2</version>
        </dependency>
```

注意：Shiro 底层依赖 commons-logging 包，如果不导入的话：

```
t org.apache.shiro.util.AbstractFactory.getInstance(AbstractFactory.java:47)
t cn.sm1234.shiro.Demo1.main(Demo1.java:25)
: java.lang.ClassNotFoundException: org.apache.commons.logging.LogFactory
t java.net.URLClassLoader$1.run(Unknown Source)
t java.net.URLClassLoader$1.run(Unknown Source)
t java.security.AccessController.doPrivileged(Native Method)
t java.net.URLClassLoader.findClass(Unknown Source)
t java.lang.ClassLoader.loadClass(Unknown Source)
t sun.misc.Launcher$AppClassLoader.loadClass(Unknown Source)
```

```
<dependency>
            <groupId>commons-logging</groupId>
            <artifactId>commons-logging</artifactId>
            <version>1.2</version>
        </dependency>
```

1.5.2. 建立自定义 Realm

```
package cn.sm1234.realms;

import org.apache.shiro.authc.AuthenticationException;
import org.apache.shiro.authc.AuthenticationInfo;
import org.apache.shiro.authc.AuthenticationToken;
import org.apache.shiro.authz.AuthorizationInfo;
import org.apache.shiro.realm.AuthorizingRealm;
import org.apache.shiro.subject.PrincipalCollection;

/**
 * 自定义 Realm
 * @author lenovo
 *
 */
public class MyRealm extends AuthorizingRealm{
```

```
    //授权方法：获取授权信息
    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection arg0) {
        return null;
    }

    //认证方法：获取认证信息
    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken arg0)
throws AuthenticationException {
        return null;
    }

}
```

1.5.3. 建立 ini 配置 SecurityManager 关联 Realm

建立 shiro.ini 配置，内容如下：

```
myRealm=cn.sm1234.realms.MyRealm

securityManager.realm=$myRealm
```

1.5.4. 编写 Shiro 的认证流程

```
package cn.sm1234.shiro;

import org.apache.shiro.SecurityUtils;
```

```
import org.apache.shiro.authc.AuthenticationToken;

import org.apache.shiro.authc.IncorrectCredentialsException;

import org.apache.shiro.authc.UnknownAccountException;

import org.apache.shiro.authc.UsernamePasswordToken;

import org.apache.shiro.config.IniSecurityManagerFactory;

import org.apache.shiro.mgt.SecurityManager;

import org.apache.shiro.subject.Subject;

import org.apache.shiro.util.Factory;


/**

 * 执行 Shiro 的认证流程

 * @author lenovo

 *

 */

public class Demo1 {


    public static void main(String[] args) {

        //1.创建安全管理器工厂

        Factory<org.apache.shiro.mgt.SecurityManager> factory = new

IniSecurityManagerFactory("classpath:shiro.ini");


        //2.创建安全管理器

        SecurityManager securityManager = factory.getInstance();


        //3.初始化 SecurityUtils 工具类

        SecurityUtils.setSecurityManager(securityManager);


        //4.从 SecurityUtils 工具中获取 Subject

        Subject subject = SecurityUtils.getSubject();
```

```
        //5.认证操作（登录）
        //AuthenticationToken: 用于封装用户输入的账户信息
        AuthenticationToken token = new UsernamePasswordToken("eric", "123456");

        try {
            subject.login(token);

            //如果 login 方法没有任何异常，代表认证成功
            System.out.println("登录成功");
        } catch (UnknownAccountException e) {
            //账户不存在
            System.out.println("账户不存在");
        }  catch (IncorrectCredentialsException e) {
            //密码错误
            System.out.println("密码错误");
        } catch (Exception e) {
            //系统错误
            System.out.println("系统错误");
        }
    }
}
```

1.5.5. 在 Realm 编写认证的逻辑

```
package cn.sm1234.realms;

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.authc.UsernamePasswordToken;

import org.apache.shiro.authz.AuthorizationInfo;

import org.apache.shiro.realm.AuthorizingRealm;

import org.apache.shiro.subject.PrincipalCollection;


/**

 * 自定义 Realm

 * @author lenovo

 *

 */

public class MyRealm extends AuthorizingRealm{


    //授权方法：获取授权信息

    @Override

    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection arg0) {

        return null;

    }


    //认证方法：获取认证信息

    @Override

    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken arg0)

throws AuthenticationException {

        System.out.println("执行认证方法...");


        //判断用户名是否存在，判断密码是否正确
```

```
        //1.如果获取用户输入的账户信息？
        UsernamePasswordToken token = (UsernamePasswordToken)arg0;
        String username = token.getUsername();

        //2.如果获取数据库的账户信息？
        //模拟数据库的账户信息
        String name = "jack";
        String password = "1234";



        //判断用户名
        if(!username.equals(name)){
            return null; // shiro 底层自动抛出 UnknownAccountException
        }

        //判断密码
        /**
         * 参数一：principal，用于把数据回传到 login 方法
         * 参数二：数据库的密码
         *     Shiro 底层对比密码的结果：
         *        1）密码正确：认证通过
         *        2）密码不正确：自动抛出 IncorrectCredentialsException
         *  参数三：realm 的名称，只有在多个 realm 的是才会使用
         */
        return new SimpleAuthenticationInfo("callback",password,"");
    }

}
```

1.6. Shiro 的授权操作

Shiro 的授权操作分为两种不同方式：

1）基于资源的授权

必须得到资源的授权码才可以访问该资源

2）基于角色的授权

必须得到该角色，才可以访问该资源

注意：如果要进行 Shiro 的授权操作，必须先完成 Shiro 的认证。

1.6.1. 基于资源的授权

```
//进行 Shiro 的授权
            //1.基于资源的授权
            //判断当前登录用户是否有“商品添加”功能
            //isPermitted():返回 true,有权限， false：没有权限
            System.out.println("productAdd="+subject.isPermitted("productAdd"));
```

Realm 的授权方法：

```
//授权方法：获取授权信息
    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection arg0) {
        System.out.println("执行授权方法...");

        SimpleAuthorizationInfo info = new SimpleAuthorizationInfo();

        //资源授权码
        info.addStringPermission("productAdd");
```

```
        return info;
    }
```

1.6.2. 基于角色的授权

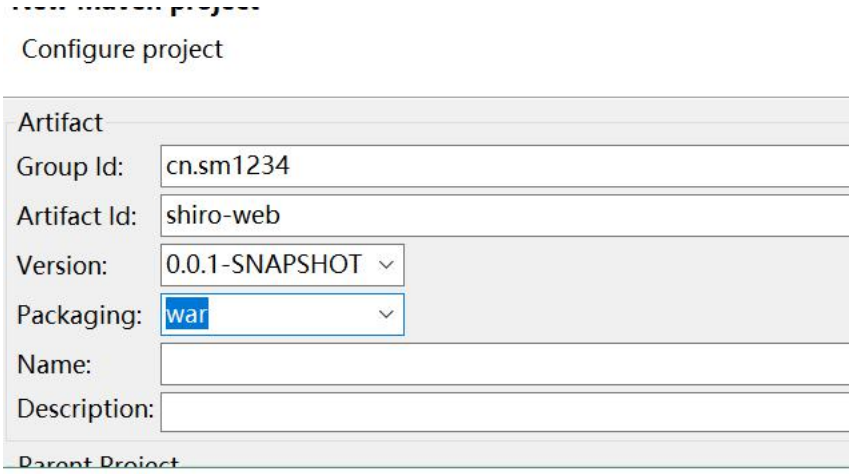
```
//2.基于角色的授权
            //判断当前登录用户是否为“超级管理员”
            System.out.println("admin="+subject.hasRole("admin"));
```

```
//授权方法：获取授权信息
    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection arg0) {
        System.out.println("执行授权方法...");

        SimpleAuthorizationInfo info = new SimpleAuthorizationInfo();

        //资源授权码
        //info.addStringPermission("productAdd");

        //进行通配符授权
        info.addStringPermission("product:*");

        //角色授权码
        info.addRole("admin");

        return info;
    }
```

2. Shiro 的 Web 集成

2.1. Shiro 与 Web 集成环境搭建

2.1.1. 建立 maven 的 web 项目，导入坐标




```
<!-- shiro 坐标 -->
        <dependency>
            <groupId>org.apache.shiro</groupId>
            <artifactId>shiro-all</artifactId>
            <version>1.3.2</version>
        </dependency>
        <dependency>
            <groupId>commons-logging</groupId>
            <artifactId>commons-logging</artifactId>
            <version>1.2</version>
        </dependency>

        <!-- servlet,jsp 坐标 -->
        <dependency>
            <groupId>javax.servlet</groupId>
```

```
            <artifactId>javax.servlet-api</artifactId>
            <version>3.0.1</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>jsp-api</artifactId>
            <version>2.0</version>
            <scope>provided</scope>
        </dependency>
```

2.1.2. 配置 web.xml（*）

```
  <!-- shiro 的 web 环境的初始化：监听器 -->
  <!--
     EnvironmentLoaderListener:在 web 应用中加载 shiro 的环境，加载 shiro.ini
   -->
  <listener>

     <listener-class>org.apache.shiro.web.env.EnvironmentLoaderListener</listener-cl
ass>
  </listener>
  <!-- 默认加载 WEB-INF/shiro.ini 文件，如果需要修改路径 -->
  <context-param>
     <param-name>shiroConfigLocations</param-name>
     <param-value>classpath:shiro.ini</param-value>
  </context-param>

  <!-- shiro 的过滤器 -->
  <filter>
```

```
    <filter-name>shiroFilter</filter-name>
    <filter-class>org.apache.shiro.web.servlet.ShiroFilter</filter-class>
  </filter>
  <filter-mapping>
    <filter-name>shiroFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
```

2.1.3. 定义 Realm

```
package cn.sm1234.realms;


import org.apache.shiro.authc.AuthenticationException;
import org.apache.shiro.authc.AuthenticationInfo;
import org.apache.shiro.authc.AuthenticationToken;
import org.apache.shiro.authz.AuthorizationInfo;
import org.apache.shiro.realm.AuthorizingRealm;
import org.apache.shiro.subject.PrincipalCollection;


public class MyRealm extends AuthorizingRealm{


    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection arg0) {
        System.out.println("执行了授权方法");
        return null;
    }


    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken arg0)
throws AuthenticationException {
```

```
        System.out.println("执行了认证方法");
        return null;
    }


}
```

2.1.4. 配置 shiro.ini

```
[main]
myRealm=cn.sm1234.realms.MyRealm
securityManager.realm=$myRealm
```

2.2. Shiro 的 web 内置过滤器（*）

shiro 的 web 内置过滤器，shiro 自己提供一些专门用于认证，授权的过滤器。

shiro 分为两种过滤器：

1）认证过滤器：

anon：用户不需要认证也可以访问

authc: 用户必须认证才可以访问

user：用户只要 rememberMe，就可以访问

2）授权过滤器

perms: 基于资源的授权过滤器

roles : 基于角色的授权过滤器

例如：

```
[main]
```

```
myRealm=cn.sm1234.realms.MyRealm

securityManager.realm=$myRealm



[urls]

/index.jsp=authc
```

2.3. 编写 Shiro 的认证操作

2.3.1. 登录页面

```
<%@ page language="java" contentType="text/html; charset=utf-8"

    pageEncoding="utf-8"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"

"http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8">

<title>登录页面</title>

</head>

<body>

<h3>用户登录</h3>

<form method="post" action="${pageContext.request.contextPath}/login">

  用户名：<input type="text" name="name"/><br/>

  密码：<input type="password" name="password"/><br/>

  <input type="submit" value="登录">

</form>

</body>

</html>
```

2.3.2. 编写 LoginServlet

```
package cn.sm1234.web;


import java.io.IOException;


import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;


import org.apache.shiro.SecurityUtils;

import org.apache.shiro.authc.AuthenticationToken;

import org.apache.shiro.authc.IncorrectCredentialsException;

import org.apache.shiro.authc.UnknownAccountException;

import org.apache.shiro.authc.UsernamePasswordToken;

import org.apache.shiro.subject.Subject;


/**

 * 登录 Servlet

 */

public class LoginServlet extends HttpServlet {

    private static final long serialVersionUID = 1L;



    protected void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

        //1.设置请求编码

        request.setCharacterEncoding("utf-8");
```

```
//2.接收用户名和密码
String name = request.getParameter("name");
String password = request.getParameter("password");

//3.调用 login 方法
//3.1 获取 Subject
Subject subject = SecurityUtils.getSubject();

AuthenticationToken token = new UsernamePasswordToken(name, password);

try {
    subject.login(token);

    //获取 Principal
    String dbName = (String)subject.getPrincipal();

    //把用户信息存储到 session
    request.getSession().setAttribute("userName", name);

    //登录成功
    response.sendRedirect(request.getContextPath()+"/index.jsp");
} catch (UnknownAccountException e) {
    request.setAttribute("msg", "用户名不存在");
    request.getRequestDispatcher("/login.jsp").forward(request, response);
} catch (IncorrectCredentialsException e) {
    request.setAttribute("msg", "密码错误");
    request.getRequestDispatcher("/login.jsp").forward(request, response);
}
```

```
    }


    protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        doGet(request, response);
    }


}
```

2.3.3. 编写 Realm 的认证方法

```
package cn.sm1234.realms;


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.authc.UsernamePasswordToken;
import org.apache.shiro.authz.AuthorizationInfo;
import org.apache.shiro.realm.AuthorizingRealm;
import org.apache.shiro.subject.PrincipalCollection;


public class MyRealm extends AuthorizingRealm{


    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection arg0) {
        System.out.println("执行了授权方法");
        return null;
```

```
    }


    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken arg0)
throws AuthenticationException {
        System.out.println("执行了认证方法");

        //1.获取用户输入的账户信息
        UsernamePasswordToken token = (UsernamePasswordToken)arg0;

        //模拟数据库的密码
        String name = "jack";
        String password = "1234";

        if(!token.getUsername().equals(name)){
            //用户不存在
            return null;
        }

        //返回密码
        return new SimpleAuthenticationInfo(name,password,"");
    }

}
```

注意：login 登录请求必须使用 anon 放行：

```
[main]

myRealm=cn.sm1234.realms.MyRealm
```

```
securityManager.realm=$myRealm


[urls]
/product/list.jsp=anon
/login=anon
/**=authc
```

2.4. 编写 Shiro 的授权操作

2.4.1. 使用 Shiro 内置授权过滤器

```
[main]
myRealm=cn.sm1234.realms.MyRealm
securityManager.realm=$myRealm


[urls]
/product/list.jsp=anon
/login=anon
/product/add.jsp=perms[product:add]
/**=authc
```

2.4.2. 配置未授权提示页面

```
[main]
perms.unauthorizedUrl=/unauth.jsp
myRealm=cn.sm1234.realms.MyRealm
securityManager.realm=$myRealm


[urls]
```

```
/product/list.jsp=anon

/login=anon

/product/add.jsp=perms[product:add]

/**=authc
```

2.4.3. 编写 Realm 的授权逻辑

```
package cn.sm1234.realms;


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.authc.UsernamePasswordToken;

import org.apache.shiro.authz.AuthorizationInfo;

import org.apache.shiro.authz.SimpleAuthorizationInfo;

import org.apache.shiro.realm.AuthorizingRealm;

import org.apache.shiro.subject.PrincipalCollection;


public class MyRealm extends AuthorizingRealm{


    @Override

    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection arg0) {

        System.out.println("执行了授权方法");

        SimpleAuthorizationInfo info = new SimpleAuthorizationInfo();

        info.addStringPermission("product:add");
```

```
        return info;
    }

    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken arg0)
throws AuthenticationException {
        System.out.println("执行了认证方法");

        //1.获取用户输入的账户信息
        UsernamePasswordToken token = (UsernamePasswordToken)arg0;

        //模拟数据库的密码
        String name = "jack";
        String password = "1234";

        if(!token.getUsername().equals(name)){
            //用户不存在
            return null;
        }

        //返回密码
        return new SimpleAuthenticationInfo(name,password,"");
    }

}
```

2.5. 分析 Shiro 内置过滤器的原理

authc：org.apache.shiro.web.filter.authc.FormAuthenticationFilter

anon：org.apache.shiro.web.filter.authc.AnonymousFilter

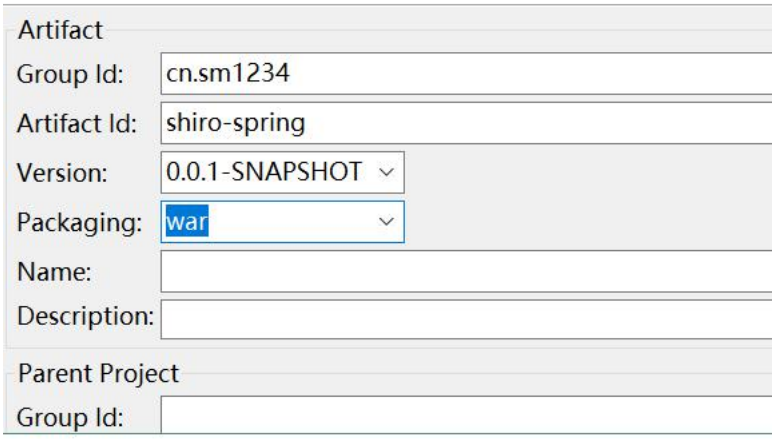
perms:org.apache.shiro.web.filter.authz.PermissionsAuthorizationFilter

roles:org.apache.shiro.web.filter.authz.RolesAuthorizationFilter

3. Spring 整合 Shiro（SSM 整合）

3.1. 搭建项目环境（导入 SpringMVC）

3.1.1. 建立 maven 项目，导入 Shiro 的坐标




```
<dependencies>

        <!-- shiro 坐标 -->

        <dependency>

            <groupId>org.apache.shiro</groupId>

            <artifactId>shiro-all</artifactId>

            <version>1.3.2</version>

        </dependency>
```

```
        <dependency>
            <groupId>commons-logging</groupId>
            <artifactId>commons-logging</artifactId>
            <version>1.2</version>
        </dependency>

        <!-- servlet,jsp 坐标 -->
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>3.0.1</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>jsp-api</artifactId>
            <version>2.0</version>
            <scope>provided</scope>
        </dependency>

        <!-- 导入 SpringMVC 支持 -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>4.3.3.RELEASE</version>
        </dependency>
    </dependencies>
```

3.1.2. 配置 web.xml，启动 SpringMVC

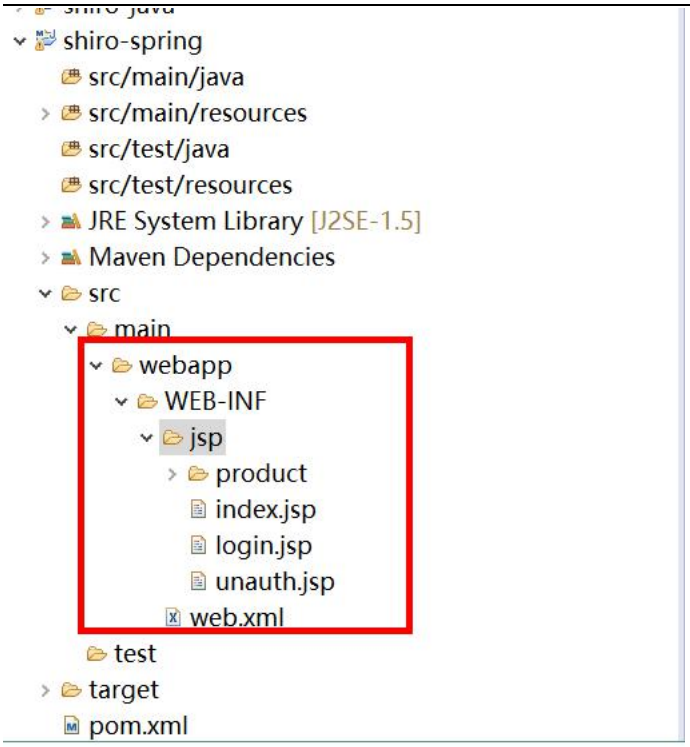
```
<!-- 启动 SpringMVC -->
 <servlet>
    <servlet-name>DispatcherServlet</servlet-name>

    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class
>
    <load-on-startup>1</load-on-startup>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:spring-mvc.xml</param-value>
    </init-param>
 </servlet>
 <servlet-mapping>
    <servlet-name>DispatcherServlet</servlet-name>
    <url-pattern>/</url-pattern>
 </servlet-mapping>
```

3.1.3. 配置 spring-mvc.xml

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

```
        http://www.springframework.org/schema/mvc/spring-mvc.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd">

    <!-- Controller 类扫描包 -->
    <contenxt:component-scan base-package="cn.sm1234.controller"/>

    <!-- 注解驱动 -->
    <mvc:annotation-driven></mvc:annotation-driven>

    <!-- 视图解析器 -->
    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <!-- 前缀 -->
        <property name="prefix" value="/WEB-INF/jsp/"/>
        <!-- 后缀 -->
        <property name="subffix" value=".jsp"></property>
    </bean>
</beans>
```

然后拷贝 shiro-web 项目的 jsp 页面到当前项目：



3.1.4. 编写 Controller 类

```
package cn.sm1234.controller;


import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;


/**

 * 主控制器

 * @author lenovo

 *

 */

@Controller

@RequestMapping("/")

public class MainController {


    @RequestMapping("/index")
```

```
    public String index(){
        return "index";
    }

    @RequestMapping("/toLogin")
    public String toLogin(){
        return "login";
    }

    @RequestMapping("/unAuth")
    public String unAuth(){
        return "unauth";
    }
}
```

```
package cn.sm1234.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

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

    @RequestMapping("/toAdd")
    public String toAdd(){
        return "product/add";
    }
```

```
    @RequestMapping("/toList")
    public String toList(){
        return "product/list";
    }

    @RequestMapping("/toUpdate")
    public String toUpdate(){
        return "product/update";
    }
}
```

3.2. Spring 整合 Shiro

3.2.1. 导入 Spring 的坐标

```
<!-- 导入 Spring 的坐标 -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-web</artifactId>
            <version>4.3.3.RELEASE</version>
        </dependency>
```

3.2.2. 配置 web.xml (*)

```
<!-- 配置 Spring 整合 Shiro 的过滤器 -->
  <!--
    DelegatingFilterProxy:这个类的作用是把请求拦截下来，把请求交给 Spirng 容器的一个 bean
```

```
处理（bean 的 id 就是 filter-name 的名称）
   -->
  <filter>
    <filter-name>shiroFilter</filter-name>

    <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-clas
s>
  </filter>
  <filter-mapping>
    <filter-name>shiroFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>

  <!-- 启动 Spring 环境 -->
  <listener>

    <listener-class>org.springframework.web.context.ContextLoaderListener</listener
-class>
  </listener>
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:applicationContext.xml</param-value>
  </context-param>
```

3.2.3. 编写 applicationContext.xml（*）

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

```
    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">

    <!-- Shiro 与 Spring 整合 -->
    <bean id="shiroFilter"
class="org.apache.shiro.spring.web.ShiroFilterFactoryBean">
        <!-- 关联 SecurityManager -->
        <property name="securityManager" ref="securityManager"/>
    </bean>

    <bean id="securityManager"
class="org.apache.shiro.web.mgt.DefaultWebSecurityManager">
        <!-- 关联 realm -->
        <property name="realm" ref="realm"/>
    </bean>

    <bean id="realm" class="cn.sm1234.realms.MyRealm"/>

</beans>
```

3.2.4. 编写 Realm

```
package cn.sm1234.realms;

import org.apache.shiro.authc.AuthenticationException;
import org.apache.shiro.authc.AuthenticationInfo;
import org.apache.shiro.authc.AuthenticationToken;
```

```
import org.apache.shiro.authz.AuthorizationInfo;

import org.apache.shiro.realm.AuthorizingRealm;

import org.apache.shiro.subject.PrincipalCollection;



public class MyRealm extends AuthorizingRealm{



    @Override

    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection arg0) {

        // TODO Auto-generated method stub

        return null;

    }



    @Override

    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken arg0) throws

AuthenticationException {

        // TODO Auto-generated method stub

        return null;

    }



}
```

3.3. 使用 Shiro 的认证过滤器拦截页面

```
<!-- Shiro 与 Spring 整合 -->

    <bean id="shiroFilter"

class="org.apache.shiro.spring.web.ShiroFilterFactoryBean">

        <!-- 关联 SecurityManager -->

        <property name="securityManager" ref="securityManager"/>
```

```
        <!-- 使用 Shiro 的内置过滤器拦截资源 -->
        <property name="filterChainDefinitions">
            <value>
                /product/toList=anon
                /**=authc
            </value>
        </property>

        <!-- 修改 shiro 的默认登录请求 -->
        <property name="loginUrl" value="/toLogin"/>
    </bean>
```

3.4. 编写 Shiro 的认证操作

3.4.1. 登录页面

```
<h3>用户登录</h3>
<font color="red">${msg}</font>
<form method="post" action="${pageContext.request.contextPath}/user/login">
  用户名：<input type="text" name="name"/><br/>
  密码：<input type="password" name="password"/><br/>
  <input type="submit" value="登录">
</form>
```

3.4.2. 编写 Controller

```
package cn.sm1234.controller;


import javax.servlet.http.HttpServletRequest;
```

```
import org.apache.shiro.SecurityUtils;
import org.apache.shiro.authc.AuthenticationToken;
import org.apache.shiro.authc.IncorrectCredentialsException;
import org.apache.shiro.authc.UnknownAccountException;
import org.apache.shiro.authc.UsernamePasswordToken;
import org.apache.shiro.subject.Subject;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

import cn.sm1234.domain.User;

@Controller
@RequestMapping("/user")
public class UserController {

    /**
     * 登录
     */
    @RequestMapping("/login")
    public String login(User user,HttpServletRequest request,Model model){

        //使用 Shiro 的认证操作
        //1. 获取 Subject 对象
        Subject subject = SecurityUtils.getSubject();

        //2.封装用户信息
        AuthenticationToken token = new UsernamePasswordToken(user.getName(),
```

```
user.getPassword());


        //3.执行认证
        try {
            subject.login(token);

            String userName = (String)subject.getPrincipal();

            request.getSession().setAttribute("userName", userName);

            //登录成功
            return "redirect:/index";
        } catch (UnknownAccountException e) {
            model.addAttribute("msg", "用户不存在");
        } catch (IncorrectCredentialsException e) {
            model.addAttribute("msg", "密码输入有误");
        }
        return "login";
    }
}
```

3.4.3. 编写 Realm 的认证操作

```
@Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken arg0)
throws AuthenticationException {


        // 1.获取用户输入的账户信息
        UsernamePasswordToken token = (UsernamePasswordToken) arg0;
```

```
        // 模拟数据库的密码
        String name = "jack";
        String password = "1234";

        if (!token.getUsername().equals(name)) {
            // 用户不存在
            return null;
        }

        // 返回密码
        return new SimpleAuthenticationInfo(name, password, "");
    }@Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken arg0)
throws AuthenticationException {

        // 1.获取用户输入的账户信息
        UsernamePasswordToken token = (UsernamePasswordToken) arg0;

        // 模拟数据库的密码
        String name = "jack";
        String password = "1234";

        if (!token.getUsername().equals(name)) {
            // 用户不存在
            return null;
        }

        // 返回密码
```

```
        return new SimpleAuthenticationInfo(name, password, "");
    }
```

注意：必须放行登录请求：

```
<!-- 使用 Shiro 的内置过滤器拦截资源 -->
        <property name="filterChainDefinitions">
            <value>
                /product/toList=anon
                /user/login=anon
                /**=authc
            </value>
        </property>
```

3.5. 编写 Shiro 的授权操作

3.5.1. 使用 Shiro 授权过滤器拦截资源

```
<!-- Shiro 与 Spring 整合 -->
    <bean id="shiroFilter"
class="org.apache.shiro.spring.web.ShiroFilterFactoryBean">
        <!-- 关联 SecurityManager -->
        <property name="securityManager" ref="securityManager"/>

        <!-- 使用 Shiro 的内置过滤器拦截资源 -->
        <property name="filterChainDefinitions">
            <value>
                /product/toList=anon
                /user/login=anon
                /product/toAdd=perms[product:add]
                /**=authc
            </value>
```

```
        </property>


        <!-- 修改 shiro 的默认登录请求 -->
        <property name="loginUrl" value="/toLogin"/>
    </bean>
```

3.5.2. 配置未授权提示页面

```
<!-- Shiro 与 Spring 整合 -->
    <bean id="shiroFilter"
class="org.apache.shiro.spring.web.ShiroFilterFactoryBean">
        <!-- 关联 SecurityManager -->
        <property name="securityManager" ref="securityManager"/>


        <!-- 使用 Shiro 的内置过滤器拦截资源 -->
        <property name="filterChainDefinitions">
            <value>
                /product/toList=anon
                /user/login=anon
                /product/toAdd=perms[product:add]
                /**=authc
            </value>
        </property>


        <!-- 修改 shiro 的默认登录请求 -->
        <property name="loginUrl" value="/toLogin"/>
        <!-- 添加未授权提示页面 -->
        <property name="unauthorizedUrl" value="/unAuth"/>
    </bean>
```

3.5.3. 编写 Realm 的授权代码

```
@Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection arg0) {
        SimpleAuthorizationInfo info = new SimpleAuthorizationInfo();

        //基于资源的授权
        info.addStringPermission("product:add");

        return info;
    }
```

3.6. 整合 MyBatis

3.6.1. 导入相关 jar 包

```
<!-- mybatis 的坐标 -->
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis</artifactId>
            <version>3.4.4</version>
        </dependency>
        <!-- mybatis 与 spring 整合 -->
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis-spring</artifactId>
            <version>1.3.0</version>
        </dependency>
        <!-- 连接池 -->
        <dependency>
```

```
            <groupId>com.alibaba</groupId>
            <artifactId>druid</artifactId>
            <version>1.1.7</version>
        </dependency>
        <!-- mysql 驱动程序 -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.41</version>
        </dependency>

        <!-- Spring 持久层支持 -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-jdbc</artifactId>
            <version>4.3.3.RELEASE</version>
        </dependency>
```

3.6.2. jdbc.properties

```
jdbc.url = jdbc:mysql://localhost:3306/shiro
jdbc.driverClassName = com.mysql.jdbc.Driver
jdbc.user = root
jdbc.password = root
```

3.6.3. 编写 applicationContext.xml 配置

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:p="http://www.springframework.org/schema/p"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:tx="http://www.springframework.org/schema/tx"
   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/context
       http://www.springframework.org/schema/context/spring-context.xsd
       http://www.springframework.org/schema/tx
       http://www.springframework.org/schema/tx/spring-tx.xsd">

    <!-- 加载 applicationContext-shiro.xml -->
    <import resource="classpath:applicationContext-shiro.xml"/>

    <!-- 加载 mybatis 与 Spring 相关配置 -->
    <!-- 加载 jdbc.properties 配置 -->
    <context:property-placeholder location="classpath:jdbc.properties"/>

    <!-- 创建数据源 -->
    <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource">
        <property name="url" value="${jdbc.url}"/>
        <property name="driverClassName" value="${jdbc.driverClassName}"/>
        <property name="username" value="${jdbc.user}"/>
        <property name="password" value="${jdbc.password}"/>
        <property name="maxActive" value="10"/>
    </bean>
```

```
    <!-- mybatis 与 spring 整合 -->
    <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
        <property name="dataSource" ref="dataSource"/>
        <!-- mybatis 别名包扫描 -->
        <property name="typeAliasesPackage" value="cn.sm1234.domain"/>
    </bean>

    <!-- mybatis 的 Mapper 接口的扫描 -->
    <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
        <property name="basePackage" value="cn.sm1234.dao"/>
    </bean>

    <!-- 开启 jdbc 事务 -->
    <bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource"/>
    </bean>

    <!-- 开启注解事务 -->
    <tx:annotation-driven transaction-manager="transactionManager"/>

    <context:component-scan base-package="cn.sm1234"/>
</beans>
```

3.6.4. 编写 Mapper 接口

```
package cn.sm1234.dao;

import cn.sm1234.domain.User;
```

```
public interface UserMapper {

    public User findByName(String name);
}
```

3.6.5. 编写 Mapper 映射文件

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="cn.sm1234.dao.UserMapper">

    <select id="findByName" parameterType="string" resultType="user">
        select id,
            name,
            password
            from
        shiro.t_user
            where name = #{value}
    </select>

</mapper>
```

3.6.6. 编写测试类

```
package cn.sm1234.test;
```

```
import javax.annotation.Resource;


import org.junit.Test;

import org.junit.runner.RunWith;

import org.springframework.test.context.ContextConfiguration;

import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;


import cn.sm1234.dao.UserMapper;

import cn.sm1234.domain.User;


@RunWith(SpringJUnit4ClassRunner.class)

@ContextConfiguration("classpath:applicationContext.xml")

public class UserMapperTest {


    @Resource

    private UserMapper userMapper;


    @Test

    public void testFindByName(){

        User user = userMapper.findByName("eric");

        System.out.println(user);

    }

}
```

3.7. 编写动态认证和授权代码

3.7.1. 动态认证代码

3.7.1.1. Mapper 接口

```
public interface UserMapper {


    public User findByName(String name);

}
```

3.7.1.2. Service 接口和实现

接口：

```
package cn.sm1234.service;


import cn.sm1234.domain.User;


public interface UserService {

    public User findByName(String name);

}
```

实现：

```
package cn.sm1234.service.impl;


import javax.annotation.Resource;


import org.springframework.stereotype.Service;
```

```
import org.springframework.transaction.annotation.Transactional;


import cn.sm1234.dao.UserMapper;
import cn.sm1234.domain.User;
import cn.sm1234.service.UserService;


@Service
@Transactional
public class UserServiceImpl implements UserService {


    @Resource
    private UserMapper userMapper;


    @Override
    public User findByName(String name) {
        return userMapper.findByName(name);
    }


}
```

3.7.1.3. MyReal

```
//认证方法
    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken arg0)
throws AuthenticationException {


        // 1.获取用户输入的账户信息
```

```
        UsernamePasswordToken token = (UsernamePasswordToken) arg0;
/*
        // 模拟数据库的密码
        String name = "jack";
        String password = "1234";

        if (!token.getUsername().equals(name)) {
            // 用户不存在
            return null;
        }*/


        //实现动态认证
        User dbUser = userService.findByName(token.getUsername());

        if(dbUser==null){
            //用户不存在
            return null;
        }

        // 返回密码
        return new SimpleAuthenticationInfo(dbUser, dbUser.getPassword(), "");
    }
```

3.7.2. 动态授权代码

模拟查询用户拥有的权限码 SQL 语句：

```
-- 查询某个用户目前拥有的权限
SELECT p.permission
    FROM t_user u
```

```
        INNER JOIN t_user_role ur ON u.id = ur.user_id
        INNER JOIN t_role_permission rp ON ur.role_id = rp.role_id
        INNER JOIN t_permission p ON rp.permission_id = p.id
    WHERE u.id = 2;
```

3.7.2.1. UserMapper 接口

```
package cn.sm1234.dao;


import java.util.List;


import cn.sm1234.domain.User;


public interface UserMapper {


    /**
     * 根据用户名查询用户
     * @param name
     * @return
     */
    public User findByName(String name);


    /**
     * 根据用户 ID 查询用户拥有的资源授权码
     */
    public List<String> findPermissionByUserId(Integer userId);
}
```

3.7.2.2. Mapper 映射文件

```
<select id="findPermissionByUserId" parameterType="int" resultType="string">
        SELECT p.permission
       FROM t_user u
          INNER JOIN t_user_role ur ON u.id = ur.user_id
          INNER JOIN t_role_permission rp ON ur.role_id = rp.role_id
          INNER JOIN t_permission p ON rp.permission_id = p.id
        WHERE u.id = #{value};
    </select>
```

3.7.2.3. Service 接口和实现

接口:

```
public List<String> findPermissionByUserId(Integer userId);
```

实现:

```
@Override
    public List<String> findPermissionByUserId(Integer userId) {
        return userMapper.findPermissionByUserId(userId);
    }
```

3.7.2.4. MyRealm

```
//授权方法
    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection arg0) {
        SimpleAuthorizationInfo info = new SimpleAuthorizationInfo();
```

```
        /*//基于资源的授权
        info.addStringPermission("product:add");

        //基于角色的授权
        info.addRole("admin");*/


        //给当前登录用户进行动态授权
        //1.获取当前用户的 pricipal
        Subject subject = SecurityUtils.getSubject();
        User dbUser = (User)subject.getPrincipal();

        //2.查询当前用户拥有的资源授权码
        List<String> perms = userService.findPermissionByUserId(dbUser.getId());
        if(perms!=null){
            //遍历授权
            for (String perm : perms) {
                if(!StringUtils.isEmpty(perm)){
                    info.addStringPermission(perm);
                }
            }
        }

        return info;
    }
```

注意此时的 shiro 过滤器配置：

```
<!-- 使用 Shiro 的内置过滤器拦截资源 -->
        <property name="filterChainDefinitions">
```

```
            <value>
                /user/login=anon
                /product/toList=perms[product:list]
                /product/toAdd=perms[product:add]
                /product/toUpdate=perms[product:update]
                /**=authc
            </value>
        </property>
```

3.8. 使用 Shiro 的 JSP 权限标签

3.8.1. 在 JSP 页面导入 shiro 标签库

```
<%@ taglib uri="http://shiro.apache.org/tags" prefix="shiro"%>
```

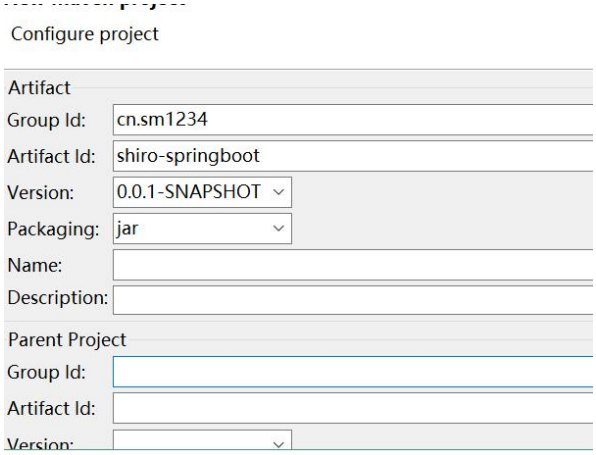
3.8.2. 使用 Shiro 权限标签

```
<shiro:hasPermission name="product:add">
<a href="${pageContext.request.contextPath}/product/toAdd">商品添加</a><br/>
</shiro:hasPermission>
<shiro:hasPermission name="product:update">
<a href="${pageContext.request.contextPath}/product/toUpdate">商品修改</a><br/>
</shiro:hasPermission>
<shiro:hasPermission name="product:list">
<a href="${pageContext.request.contextPath}/product/toList">商品列表</a><br/>
</shiro:hasPermission>
```

4. Spring Boot 整合 Shiro（整合 SSM）

4.1. 搭建 Spring Boot 项目环境

4.1.1. 建立 maven 项目，导入坐标




```
<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>

    <!-- Spring Boot 父工程 -->
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.5.4.RELEASE</version>
    </parent>

    <groupId>cn.sm1234</groupId>
    <artifactId>shiro-springboot</artifactId>
    <version>0.0.1-SNAPSHOT</version>
```

```
    <dependencies>
        <!-- web 支持，SpringMVC， Servlet 支持等 -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <!-- 导入 thymeleaf 支持 -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-thymeleaf</artifactId>
        </dependency>
    </dependencies>

    <!-- 属性 -->
    <properties>
        <!-- JDK 编译版本 -->
        <java.version>1.8</java.version>
        <!-- 把 thymeleaf 升级为 3.0 以上 -->
        <thymeleaf.version>3.0.2.RELEASE</thymeleaf.version>
        <thymeleaf-layout-dialect.version>2.0.4</thymeleaf-layout-dialect.version>
    </properties>

</project>
```

4.1.2. 编写 Controller

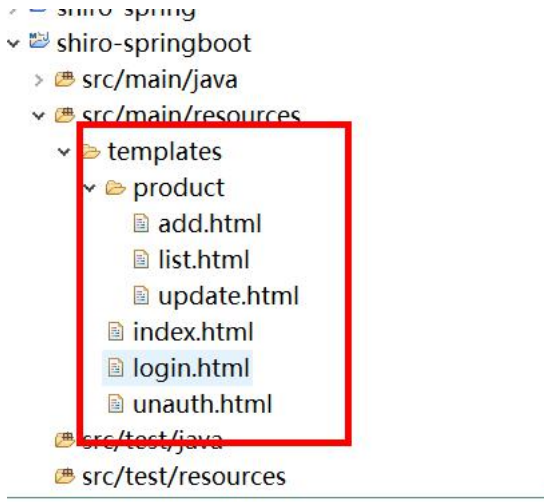
```
package cn.sm1234.controller;


import org.springframework.stereotype.Controller;
```

```
import org.springframework.web.bind.annotation.RequestMapping;


/**
 * 主控制器
 * @author lenovo
 *
 */
@Controller
@RequestMapping("/")
public class MainController {

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

    @RequestMapping("/toLogin")
    public String toLogin(){
        return "login";
    }

    @RequestMapping("/unAuth")
    public String unAuth(){
        return "unauth";
    }
}
```

```
package cn.sm1234.controller;
```

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

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

    @RequestMapping("/toAdd")
    public String toAdd(){
        return "product/add";
    }

    @RequestMapping("/toList")
    public String toList(){
        return "product/list";
    }

    @RequestMapping("/toUpdate")
    public String toUpdate(){
        return "product/update";
    }
}
```

4.1.3. 编写 html 页面



4.2. Shiro 整合

4.2.1. 导入 shiro 整合坐标

```
<dependency>
            <groupId>org.apache.shiro</groupId>
            <artifactId>shiro-spring</artifactId>
            <version>1.4.0</version>
        </dependency>
```

4.2.2. 编写 Shiro 配置类

```
package cn.sm1234.shiro;


import org.apache.shiro.spring.web.ShiroFilterFactoryBean;
import org.apache.shiro.web.mgt.DefaultWebSecurityManager;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
```

```
/**
 * Shiro 的配置类
 * @author lenovo
 *
 */
@Configuration
public class ShiroConfig {

    /**
     * 1.创建 ShiroFilterFactoryBean
     */
    @Bean
    public ShiroFilterFactoryBean shiroFilterFactoryBean(DefaultWebSecurityManager
securityManager){
        ShiroFilterFactoryBean bean = new ShiroFilterFactoryBean();

        //关联 SecurityManager
        bean.setSecurityManager(securityManager);

        return bean;
    }

    /**
     * 2.创建 SecurityManager
     */
    @Bean
    public DefaultWebSecurityManager defaultWebSecurityManager(MyRealm realm){
        DefaultWebSecurityManager manager = new DefaultWebSecurityManager();
```

```
        //关联 realm
        manager.setRealm(realm);

        return manager;
    }

    /**
     * 3.创建 Realm
     */
    @Bean
    public MyRealm myReal(){
        MyRealm realm = new MyRealm();
        return realm;
    }

}
```

4.3. 配置 Shiro 认证过滤器实现资源拦截

```
/**
     * 1.创建 ShiroFilterFactoryBean
     */
    @Bean
    public ShiroFilterFactoryBean shiroFilterFactoryBean(DefaultWebSecurityManager
securityManager){
        ShiroFilterFactoryBean bean = new ShiroFilterFactoryBean();

        //关联 SecurityManager
```

```
        bean.setSecurityManager(securityManager);


        Map<String,String> filterMap = new LinkedHashMap<>();

        //认证过滤器
        filterMap.put("/product/toAdd", "anon");
        filterMap.put("/**", "authc");

        //添加 shiro 过滤器
        bean.setFilterChainDefinitionMap(filterMap);


        //修改登录请求
        bean.setLoginUrl("/toLogin");
        return bean;
    }
```

4.4. 编写 Shiro 的认证操作

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>登录页面</title>
</head>
<body>
<h3>用户登录</h3>
<font color="red"></font>
<form method="post" action="/user/login">
  用户名：<input type="text" name="name"/><br/>
```

```
  密码：<input type="password" name="password"/><br/>
  <input type="submit" value="登录">
</form>
</body>
</html>
```

4.4.1. 编写 UserController

```
package cn.sm1234.controller;

import javax.servlet.http.HttpServletRequest;

import org.apache.shiro.SecurityUtils;
import org.apache.shiro.authc.AuthenticationToken;
import org.apache.shiro.authc.IncorrectCredentialsException;
import org.apache.shiro.authc.UnknownAccountException;
import org.apache.shiro.authc.UsernamePasswordToken;
import org.apache.shiro.subject.Subject;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

import cn.sm1234.domain.User;

@Controller
@RequestMapping("/user")
public class UserController {

    /**
     * 登录
```

```
     */
    @RequestMapping("/login")
    public String login(User user,HttpServletRequest request,Model model){
        //使用 shiro 进行登录

        Subject subject = SecurityUtils.getSubject();

        AuthenticationToken token = new UsernamePasswordToken(user.getName(),
user.getPassword());

        try {
            subject.login(token);

            //登录成功
            User dbUser = (User)subject.getPrincipal();

            request.getSession().setAttribute("userName", dbUser.getName());

            return "redirect:/index";
        } catch (UnknownAccountException e) {
            model.addAttribute("msg", "用户名不存在");
            return "login";
        } catch (IncorrectCredentialsException e) {
            model.addAttribute("msg", "密码");
            return "login";
        }
    }
}
```

4.4.2. 编写 MyReal 的代码

```
//认证
    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken arg0)
throws AuthenticationException {
        // 1.获取用户输入的账户信息
        UsernamePasswordToken token = (UsernamePasswordToken) arg0;

        // 模拟数据库的密码
        String name = "jack";
        String password = "1234";

        if (!token.getUsername().equals(name)) {
            // 用户不存在
            return null;
        }

        User dbUser = new User();
        dbUser.setName(name);
        dbUser.setPassword(password);

        // 返回密码
        return new SimpleAuthenticationInfo(dbUser, dbUser.getPassword(), "");
    }
```

4.4.3. 登录页面提示

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
```

```
"http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8">

<title>登录页面</title>

</head>

<body>

<h3>用户登录</h3>

<font color="red" th:text="${msg}"></font>

<form method="post" action="/user/login">

  用户名：<input type="text" name="name"/><br/>

  密码：<input type="password" name="password"/><br/>

  <input type="submit" value="登录">

</form>

</body>

</html>
```

4.5. 编写 shiro 的授权操作

```
    /**
     * 1.创建 ShiroFilterFactoryBean
     */
    @Bean
    public ShiroFilterFactoryBean shiroFilterFactoryBean(DefaultWebSecurityManager
securityManager){
        ShiroFilterFactoryBean bean = new ShiroFilterFactoryBean();


        //关联 SecurityManager

        bean.setSecurityManager(securityManager);
```

```
        Map<String,String> filterMap = new LinkedHashMap<>();


        //认证过滤器
        filterMap.put("/product/toAdd", "anon");
        //放行登录页面
        filterMap.put("/user/login", "anon");
        //授权过滤器
        filterMap.put("/product/toAdd", "perms[product:add]");
        filterMap.put("/product/toUpdate", "perms[product:update]");
        filterMap.put("/**", "authc");


        //添加 shiro 过滤器
        bean.setFilterChainDefinitionMap(filterMap);


        //修改登录请求
        bean.setLoginUrl("/toLogin");
        //添加未授权提示页面
        bean.setUnauthorizedUrl("/unAuth");
        return bean;
    }
```

```
@Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection
principals) {
        SimpleAuthorizationInfo info = new SimpleAuthorizationInfo();


        info.addStringPermission("product:add");
```

```
        return info;
    }
```

4.6. 整合 MyBatis

4.6.1. 导入 mybatis 相关坐标

```
<!-- SpringBoot 的 Mybatis 启动器 -->
        <dependency>
            <groupId>org.mybatis.spring.boot</groupId>
            <artifactId>mybatis-spring-boot-starter</artifactId>
            <version>1.1.1</version>
        </dependency>

        <!-- druid 连接池 -->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid</artifactId>
            <version>1.0.9</version>
        </dependency>

        <!-- mysql -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
        </dependency>
```

4.6.2. 配置 application.properties

```
spring.datasource.url=jdbc:mysql://localhost:3306/shiro
```

```
spring.datasource.driverClassName=com.mysql.jdbc.Driver

spring.datasource.username=root

spring.datasource.password=root


spring.datasource.type=com.alibaba.druid.pool.DruidDataSource


mybatis.type-aliases-package=cn.sm1234.domain
```

4.6.3. 编写 Mapper 接口和映射

4.6.3.1. Mapper 接口

```
package cn.sm1234.dao;


import cn.sm1234.domain.User;


public interface UserMapper {


    public User findByName(String name);

}
```

4.6.3.2. Mapper 映射

```
<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE mapper

PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="cn.sm1234.dao.UserMapper">

    <select id="findByName" parameterType="string" resultType="user">
```

```
        SELECT  id,
        NAME,
        PASSWORD
        FROM
        shiro.t_user
         where name = #{value}
    </select>
</mapper>
```

4.6.4. 修改 ShiroApplication 启动类

```
@SpringBootApplication
@MapperScan("cn.sm1234.dao")
public class ShiroApplication {


    public static void main(String[] args) {
        SpringApplication.run(ShiroApplication.class, args);
    }
}
```

4.6.5. 测试环境

```
package cn.sm1234.test;


import javax.annotation.Resource;


import org.junit.Test;
import org.junit.runner.RunWith;
```

```
import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;



import cn.sm1234.ShiroApplication;

import cn.sm1234.dao.UserMapper;

import cn.sm1234.domain.User;



@RunWith(SpringJUnit4ClassRunner.class)

@SpringBootTest(classes=ShiroApplication.class)

public class UserMapperTest {



    @Resource

    private UserMapper userMapper;



    @Test

    public void testFindByName(){

        User user = userMapper.findByName("eric");

        System.out.println(user);

    }

}
```

4.7. 改造为动态认证和授权

```
public class MyRealm extends AuthorizingRealm {



    @Resource

    private UserService userService;



    @Override
```

```
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection
principals) {
        SimpleAuthorizationInfo info = new SimpleAuthorizationInfo();

        //info.addStringPermission("product:add");

        //得到当前用户
        Subject subject = SecurityUtils.getSubject();
        User dbUser = (User)subject.getPrincipal();

        List<String> perms = userService.findPermissionByUserId(dbUser.getId());
        if(perms!=null){
            for (String perm : perms) {
                if(!StringUtils.isEmpty(perm)){
                    info.addStringPermission(perm);
                }
            }
        }

        return info;
    }

    //认证
    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken arg0)
throws AuthenticationException {
        // 1.获取用户输入的账户信息
        UsernamePasswordToken token = (UsernamePasswordToken) arg0;
```

```
        /*// 模拟数据库的密码
        String name = "jack";
        String password = "1234";

        if (!token.getUsername().equals(name)) {
            // 用户不存在
            return null;
        }

        User dbUser = new User();
        dbUser.setName(name);
        dbUser.setPassword(password);*/

        User dbUser = userService.findByName(token.getUsername());
        if(dbUser==null){
            //用户不存在
            return null;
        }

        // 返回密码
        return new SimpleAuthenticationInfo(dbUser, dbUser.getPassword(), "");
    }

}
```

4.8. Thymeleaf 整合 Shiro 权限标签

4.8.1. 导入整合坐标

```
<!-- thymeleaf 整合 shiro 标签 -->
```

```
        <dependency>
            <groupId>com.github.theborakompanioni</groupId>
            <artifactId>thymeleaf-extras-shiro</artifactId>
            <version>2.0.0</version>
        </dependency>
```

4.8.2. 配置 ShiroConfig

在类中添加一个方法：

```
    /**
     * 整合 Shiro 标签
     */
    @Bean
    public ShiroDialect shiroDialect(){
        return new ShiroDialect();
    }
```

4.8.3. 在 html 页面使用 shiro 标签

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>商品管理系统后台主页</title>
</head>
<body>
<h3>商品管理系统后台主页</h3>
```

```
当前用户名：<span th:text="${session.userName}"></span>


<hr/>
<span shiro:hasPermission="product:add">
<a href="/product/toAdd">商品添加</a><br/>
</span>
<span shiro:hasPermission="product:update">
<a href="/product/toUpdate">商品修改</a><br/>
</span>
<span shiro:hasPermission="product:list">
<a href="/product/toList">商品列表</a><br/>
</span>


</body>
</html>
```

4.9. Shiro 注销功能

4.9.1. idnex.hmtl

```
<a href="/user/logout">注销</a>
```

4.9.2. 编写 UserController

```
    /**
     * 注销方法
     */
    @RequestMapping("/logout")
```

```
    public String logout(){
        Subject subject = SecurityUtils.getSubject();
        subject.logout(); //shiro 底层删除 session 的会话信息
        return "redirect:/toLogin";
    }
```

4.10. 实现 RememberMe 功能

4.10.1. 配置 ShiroConfig 类

```
/**
 * Shiro 的配置类
 * @author lenovo
 *
 */
@Configuration
public class ShiroConfig {

    /**
     * 1.创建 ShiroFilterFactoryBean
     */
    @Bean
    public ShiroFilterFactoryBean shiroFilterFactoryBean(DefaultWebSecurityManager
securityManager){
        ShiroFilterFactoryBean bean = new ShiroFilterFactoryBean();

        //关联 SecurityManager
        bean.setSecurityManager(securityManager);

        Map<String,String> filterMap = new LinkedHashMap<>();
```

```
        //认证过滤器
        filterMap.put("/product/toAdd", "perms[product:add]");
        //放行登录页面
        filterMap.put("/user/login", "anon");
        //授权过滤器
        filterMap.put("/product/toList", "perms[product:list]");
        filterMap.put("/product/toUpdate", "perms[product:update]");
        //添加 user 过滤器
        filterMap.put("/index", "user"); // /index 的请求只要使用 rememberMe 功能，就可以访问了
        filterMap.put("/**", "authc");

        //添加 shiro 过滤器
        bean.setFilterChainDefinitionMap(filterMap);

        //修改登录请求
        bean.setLoginUrl("/toLogin");
        //添加未授权提示页面
        bean.setUnauthorizedUrl("/unAuth");
        return bean;
    }

    /**
     * 2.创建 SecurityManager
     */
    @Bean
    public DefaultWebSecurityManager defaultWebSecurityManager(MyRealm realm,CookieRememberMeManager rememberMeManager){
```

```
        DefaultWebSecurityManager manager = new DefaultWebSecurityManager();


        //关联 realm
        manager.setRealm(realm);

        //关联 rememberMeManager
        manager.setRememberMeManager(rememberMeManager);

        return manager;
    }

    //创建 CookieRememberMeManager
    @Bean
    public CookieRememberMeManager cookieRememberMeManager(SimpleCookie cookie){
        CookieRememberMeManager manager = new CookieRememberMeManager();
        manager.setCookie(cookie);
        return manager;
    }

    /**
     * RememberMe 的功能
     */
    //创建 Cookie
    @Bean
    public SimpleCookie simpleCookie(){
        SimpleCookie cookie = new SimpleCookie("rememberMe");
        //设置 cookie 的时间长度
        cookie.setMaxAge(120);
        //设置只读模型
```

```
        cookie.setHttpOnly(true);
        return cookie;
    }

    /**
     * 3.创建 Realm
     */
    @Bean
    public MyRealm myReal(){
        MyRealm realm = new MyRealm();
        return realm;
    }

    /**
     * 整合 Shiro 标签
     */
    @Bean
    public ShiroDialect shiroDialect(){
        return new ShiroDialect();
    }

}
```

4.10.2. 修改 login.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>登录页面</title>
</head>
<body>
<h3>用户登录</h3>
<font color="red" th:text="${msg}"></font>
<form method="post" action="/user/login">
  用户名：<input type="text" name="name"/><br/>
  密码：<input type="password" name="password"/><br/>
  是否记住我：<input type="checkbox" name="rememberMe" value="1"/><br/>
  <input type="submit" value="登录">
</form>
</body>
</html>
```

4.10.3. 修改 UserController

```
/**
     * 登录
     */
    @RequestMapping("/login")
    public String login(User user,String rememberMe,HttpServletRequest
request,Model model){
        //使用 shiro 进行登录

        Subject subject = SecurityUtils.getSubject();

        //AuthenticationToken token = new
UsernamePasswordToken(user.getName(), user.getPassword());
```

```
        UsernamePasswordToken token = new
UsernamePasswordToken(user.getName(), user.getPassword());

        //设置 remenmberMe 的功能
        if(rememberMe!=null && rememberMe.equals("1")){
            token.setRememberMe(true);
        }
        try {
            subject.login(token);

            //登录成功
            User dbUser = (User)subject.getPrincipal();

            request.getSession().setAttribute("userName",
dbUser.getName());

            return "redirect:/index";
        } catch (UnknownAccountException e) {
            model.addAttribute("msg", "用户名不存在");
            return "login";
        } catch (IncorrectCredentialsException e) {
            model.addAttribute("msg", "密码错误");
            return "login";
        }
    }
```

4.11. 自定义 Filter 找回 Session 信息

```
package cn.sm1234.filter;
```

```
import javax.servlet.ServletRequest;

import javax.servlet.ServletResponse;


import org.apache.shiro.session.Session;

import org.apache.shiro.subject.Subject;

import org.apache.shiro.web.filter.authc.FormAuthenticationFilter;


import cn.sm1234.domain.User;

/**

 * 自定义认证过滤器，加入 RememberMe 的功能

 * @author lenovo

 *

 */

public class UserFormAuthenticationFilter extends FormAuthenticationFilter {


    @Override

    protected boolean isAccessAllowed(ServletRequest request, ServletResponse

response, Object mappedValue) {

        Subject subject = getSubject(request, response);


        // 如果 isAuthenticated 为 false 证明不是登录过的，同时 isRememberd 为 true

        // 证明是没登陆直接通过记住我功能进来的

        if (!subject.isAuthenticated() && subject.isRemembered()) {


            // 获取 session 看看是不是空的

            Session session = subject.getSession(true);


            // 查看 session 属性当前是否是空的

            if (session.getAttribute("userName") == null) {
```

```
                // 如果是空的才初始化
                User dbUser = (User)subject.getPrincipal();
                //存入用户数据
                session.setAttribute("userName", dbUser.getName());
            }
        }

        // 这个方法本来只返回 subject.isAuthenticated() 现在我们加上
subject.isRemembered()
        // 让它同时也兼容 remember 这种情况
        return subject.isAuthenticated() || subject.isRemembered();
    }
}
```

4.12. 使用 Shiro 的加密算法改造登录

```
    /**
     * 登录
     */
    @RequestMapping("/login")
    public String login(User user,String rememberMe,HttpServletRequest request,Model
model){
        //使用 shiro 进行登录

        Subject subject = SecurityUtils.getSubject();

        //AuthenticationToken token = new UsernamePasswordToken(user.getName(),
user.getPassword());
```

```
        //使用 Shiro 对密码进行加密
        Md5Hash hash = new Md5Hash(user.getPassword(), user.getName(), 2);

        UsernamePasswordToken token = new UsernamePasswordToken(user.getName(),
hash.toString());

        //设置 remenmberMe 的功能
        if(rememberMe!=null && rememberMe.equals("1")){
            token.setRememberMe(true);
        }
        try {
            subject.login(token);

            //登录成功
            User dbUser = (User)subject.getPrincipal();

            request.getSession().setAttribute("userName", dbUser.getName());

            return "redirect:/index";
        } catch (UnknownAccountException e) {
            model.addAttribute("msg", "用户名不存在");
            return "login";
        } catch (IncorrectCredentialsException e) {
            model.addAttribute("msg", "密码错误");
            return "login";
        }
    }
```

4.13. Kaptcha 验证码

4.13.1. 导入 Kaptcha 坐标

```
<!-- 验证码 -->
        <dependency>
            <groupId>com.github.penggle</groupId>
            <artifactId>kaptcha</artifactId>
            <version>2.3.2</version>
        </dependency>
```

4.13.2. 编写 Kaptcha 配置类

```
package cn.sm1234.shiro;

import java.util.Properties;

import org.springframework.context.annotation.Bean;
import org.springframework.stereotype.Component;

import com.google.code.kaptcha.impl.DefaultKaptcha;
import com.google.code.kaptcha.util.Config;

@Component
public class KaptcharConfig {

    @Bean
    public DefaultKaptcha getDefaultKaptcha() {
        com.google.code.kaptcha.impl.DefaultKaptcha defaultKaptcha = new
com.google.code.kaptcha.impl.DefaultKaptcha();
```

```
        Properties properties = new Properties();
        properties.setProperty("kaptcha.border", "yes");
        properties.setProperty("kaptcha.border.color", "105,179,90");
        properties.setProperty("kaptcha.textproducer.font.color", "blue");
        properties.setProperty("kaptcha.image.width", "110");
        properties.setProperty("kaptcha.image.height", "40");
        properties.setProperty("kaptcha.textproducer.font.size", "30");
        properties.setProperty("kaptcha.session.key", "code");
        properties.setProperty("kaptcha.textproducer.char.length", "4");
        properties.setProperty("kaptcha.textproducer.font.names", "宋体,楷体,微软雅黑
");
        Config config = new Config(properties);
        defaultKaptcha.setConfig(config);

        return defaultKaptcha;
    }
}
```

4.13.3. KaptchaController

```
package cn.sm1234.controller;

import java.awt.image.BufferedImage;
import java.io.ByteArrayOutputStream;

import javax.annotation.Resource;
import javax.imageio.ImageIO;
import javax.servlet.ServletOutputStream;
import javax.servlet.http.HttpServletRequest;
```

```
import javax.servlet.http.HttpServletResponse;


import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;


import com.google.code.kaptcha.impl.DefaultKaptcha;


@Controller
public class KaptcharController {


    @Resource
    private DefaultKaptcha defaultKaptcha;


    @RequestMapping("/defaultKaptcha")
    public void defaultKaptcha(HttpServletRequest httpServletRequest,
HttpServletResponse httpServletResponse)
            throws Exception {
        byte[] captchaChallengeAsJpeg = null;
        ByteArrayOutputStream jpegOutputStream = new ByteArrayOutputStream();
        try {
            // 生产验证码字符串并保存到 session 中
            String createText = defaultKaptcha.createText();
            httpServletRequest.getSession().setAttribute("verifyCode", createText);
            // 使用生产的验证码字符串返回一个 BufferedImage 对象并转为 byte 写入到 byte 数组
中

            BufferedImage challenge = defaultKaptcha.createImage(createText);
            ImageIO.write(challenge, "jpg", jpegOutputStream);
        } catch (IllegalArgumentException e) {
            httpServletResponse.sendError(HttpServletResponse.SC_NOT_FOUND);
```

```
            return;
        }

        // 定义 response 输出类型为 image/jpeg 类型，使用 response 输出流输出图片的 byte 数
组
        captchaChallengeAsJpeg = jpegOutputStream.toByteArray();
        httpServletResponse.setHeader("Cache-Control", "no-store");
        httpServletResponse.setHeader("Pragma", "no-cache");
        httpServletResponse.setDateHeader("Expires", 0);
        httpServletResponse.setContentType("image/jpeg");
        ServletOutputStream responseOutputStream =
httpServletResponse.getOutputStream();
        responseOutputStream.write(captchaChallengeAsJpeg);
        responseOutputStream.flush();
        responseOutputStream.close();
    }
}
```

4.13.4. 在 login.html 使用验证码

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>登录页面</title>
</head>
<body>
<h3>用户登录</h3>
```

```
<font color="red" th:text="${msg}"></font>
<form method="post" action="/user/login">
  用户名：<input type="text" name="name"/><br/>
  密码：<input type="password" name="password"/><br/>
验证码：<input type="text" name="code"/><img src="/defaultKaptcha"
onclick="this.src='/defaultKaptcha?d='+new Date()"/>
<br/>
  是否记住我：<input type="checkbox" name="rememberMe" value="1"/><br/>
  <input type="submit" value="登录">
</form>
</body>
</html>
```

4.13.5. 验证用户输入是否正确

```
    /**
     * 登录
     */
    @RequestMapping("/login")
    public String login(User user,String rememberMe,String code,HttpServletRequest
request,Model model){
        //判断验证码是否正确
        if(!StringUtils.isEmpty(code)){
            //取出生成的验证码
            String verifyCode =
(String)request.getSession().getAttribute("verifyCode");
            if(!code.equals(verifyCode)){
                model.addAttribute("msg", "验证码输入有误");
                return "login";
```

```
            }
        }


        //使用 shiro 进行登录

        Subject subject = SecurityUtils.getSubject();

        //AuthenticationToken token = new UsernamePasswordToken(user.getName(),
user.getPassword());

        //使用 Shiro 对密码进行加密
        Md5Hash hash = new Md5Hash(user.getPassword(), user.getName(), 2);

        UsernamePasswordToken token = new UsernamePasswordToken(user.getName(),
hash.toString());

        //设置 remenmberMe 的功能
        if(rememberMe!=null && rememberMe.equals("1")){
            token.setRememberMe(true);
        }
        try {
            subject.login(token);

            //登录成功
            User dbUser = (User)subject.getPrincipal();

            request.getSession().setAttribute("userName", dbUser.getName());
```

```
            return "redirect:/index";
        } catch (UnknownAccountException e) {
            model.addAttribute("msg", "用户名不存在");
            return "login";
        } catch (IncorrectCredentialsException e) {
            model.addAttribute("msg", "密码错误");
            return "login";
        }
    }
```