

首页 资讯 精华 论坛 问答 博客 专栏 群组
您还未登录！[登录](#)



Veiking
浏览: 88429 次
性别:
来自: 深圳
 我现在离线

最近访客 [更多访客>>](#)

- wanghan1977
- jiangxxtt
- ziyahl
- ccing

- 博主相关
- [博客](#)
 - [微博](#)
 - [相册](#)
 - [收藏](#)
 - [留言](#)
 - [关于我](#)

- 文章分类
- [全部博客 \(18\)](#)
 - [Spring \(9\)](#)
 - [设计模式 \(2\)](#)
 - [Java \(14\)](#)
 - [github \(1\)](#)
 - [实用算法 \(1\)](#)
 - [SpringMVC \(3\)](#)
 - [Maven \(5\)](#)
 - [MyBatis \(4\)](#)
 - [Exception \(5\)](#)
 - [MySQL \(4\)](#)
 - [JDBC \(2\)](#)
 - [log4j \(2\)](#)
 - [JUnit \(1\)](#)
 - [JPA \(1\)](#)
 - [Hibernate \(0\)](#)
 - [Quartz \(2\)](#)
 - [集群 \(0\)](#)
 - [Springboot \(1\)](#)
 - [Security \(1\)](#)

社区版块

- [我的资讯 \(0\)](#)
- [我的论坛 \(0\)](#)
- [我的问答 \(0\)](#)

存档分类

- [2018-08 \(1\)](#)
- [2017-05 \(1\)](#)
- [2017-04 \(1\)](#)
- [更多存档...](#)

最新评论

[springBoot+security+mybatis 实现用户权限的数据库动态管理](#)

博客分类:

- [Springboot](#)
- [Security](#)
- [Java](#)

[Spring](#)
[SpringMVC](#)
[MyBatis](#)
[Maven](#)
[MySQL](#)
[SpringbootSecuritySpringSpringMVCMyBatis](#)

一、Spring Security 应用的概述

鉴于目前微服务的兴起，Spring周边方案的普及，以及 Spring Security 强大的和高度可定制的优良特性，最近关注了一下相关内容，顺便留个笔记心得，希望对大家有所帮助。

Spring Security 权限方案针对不同场景需求有多种不同的使用方法，在此，我们最终描述的是如何采用数据库存储配置，并通过自定义过滤器的实现方式，来进行对权限的权利，希望这个过程能加深对Spring Security的理解，如有初学者阅读，建议先简单了解下Spring Security 框架，以避免遭遇太多的疑惑。

先说大概，Spring Security，包括绝大部分的安全框架，都可以简单理解为两个核心：一个是认证，即看看这个请求用户存在不存在啊，密码对不对啊等，认证，来确保请求用户的合法性；另一个就是鉴权，即看看这个访问的资源，有没有权限，这个决定用户能做什么，不能做什么。敲黑板，两个重点核心：认证！鉴权！下面，我们将尝试下，看看在 Spring Security 框架内是如何完成这些功能的。

在这里，我们准备剖析 Spring Security 底层的基本逻辑，有些还需要就源码进行解读，这里只讲应用层面的东西。

先说认证，与本次实现密切相关的几个类或接口，是UserDetails、UserDetailsService、AuthenticationProvider，我们可以这么理解：UserDetails是用来封装用户的，用户的帐号信息啊、一些权限啊，帐号状态啊等信息，从数据库那里拿到，首先是要封装成UserDetails的样子，才可以在Spring Security框架中使用的；UserDetailsService，顾名思义，处理UserDetails的Service，它是提供去查询账号信息并封装成UserDetailsService的服务；AuthenticationProvider的主要工作是负责认证，从登录请求那里拿到帐号密码之类，然后再跟从数据库资源那里得到的UserDetails进行对比确认，如果发现不对劲儿，该报错报错，该提示提示，如果OK，则把这些信息揉巴成一团，封装成一个包含所有信息的认证对象，交给 Spring Security 框架进行管理，供后边有需要的时候随时取用。

接下来说鉴权，Spring Security 的鉴权方式有多种，我们大概捋一下，这里我们重点讲述如何通过自定义过滤器的鉴权方式，来实现数据库配置权限的动态管理，与此密切相关的几个核心类或接口分别是：AbstractSecurityInterceptor (Filter)、FilterInvocationSecurityMetadataSource和AccessDecisionManager。我们可以这么理解，FilterInvocationSecurityMetadataSource是权限资源管理器，它的主要工作就是根据请求的资源（路径），从数据库获取相对应的权限信息；AccessDecisionManager类似权限管理判断器，负责校验当前认证用户的权限，是否可以访问；AbstractSecurityInterceptor就是前边这两个角色负责表演的地方，拿到访问资源所需的权限，和认证用户的权限，对比，出结果，如果出现对比不成功，分分钟抛要一个拒绝访问的异常，403forbidden了！

在这里先把这几个类或者接口，默默的混个眼熟，认证相关：UserDetails、UserDetailsService、AuthenticationProvider；鉴权相关：AbstractSecurityInterceptor (Filter)、FilterInvocationSecurityMetadataSource和AccessDecisionManager，谁是干啥的，谁跟谁什么关系，大概就是那么个意思了，也能猜出 Spring Security 是怎么工作的。

接下来还会介绍下 Spring Security 的核心配置类：WebSecurityConfigurerAdapter，它的主要职责就是配置配置哪些资源不需要权限限制啊，哪些需要啊等等，以及做一些综合性的配置操作，以及 Spring Security 本身的注册等。

以上是 Spring Security 应用的一个概述，目的是有个简单的了解，提前混个眼熟，便于思路连续性的展开。

二、springBoot项目初建

在eclipse上怎么创建maven项目，我们就不多说了，方式很多种；这里讲，本次 Spring Security 的实现要用到的依赖主要有 Spring MVC、Spring Security、Mybatis、thymeleaf，我们用自己最熟悉的方式建个maven项目，然后修改pom.xml文件如下：

```
pom.xml
Xml代码 ✨ ☆
<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>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.0.1.RELEASE</version>
</parent>
<groupId>sec_test</groupId>
<artifactId>sec</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>war</packaging>

<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <spring-boot.version>2.0.1.RELEASE</spring-boot.version>
</properties>

<dependencies>
  <!-- spring-boot -->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
</dependencies>
```

```

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
<dependency>
  <groupId>org.thymeleaf.extras</groupId>
  <artifactId>thymeleaf-extras-springsecurity4</artifactId>
</dependency>
<dependency>
  <groupId>org.mybatis.spring.boot</groupId>
  <artifactId>mybatis-spring-boot-starter</artifactId>
  <version>1.3.2</version>
</dependency>
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>com.alibaba</groupId>
  <artifactId>druid</artifactId>
  <version>1.0.19</version>
</dependency>
<!-- https://mvnrepository.com/artifact/com.google.code.gson/gson -->
<dependency>
  <groupId>com.google.code.gson</groupId>
  <artifactId>gson</artifactId>
  <version>2.8.5</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.projectlombok/lombok -->
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <version>1.16.20</version>
  <scope>provided</scope>
</dependency>
</dependencies>

<build>
  <finalName>${project.name}</finalName>
  <pluginManagement>
    <plugins>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </pluginManagement>
</build>
</project>

```

Spring boot下，各个版本一般都是向下兼容略有不同，在这种简单的应用上基本体现不出太大的差异，我们遵循各自习惯去配置，开心就好，注意pom文件中，除了几个核心的，额外还有gson和lombok的引入，gson是为了方便输出对象日志；lombok是为了省去bean类中set/get方法，这个可以让代码看起来稍微简练些，首次使用需要提前安装下lombok的插件之类，感兴趣的可以自行百度下，也可以根据自己的习惯决定是否使用。

接下来我们在 src/main/resources 中创建一个 application.yml 作为springBoot项目的主配置文件，注意，这个.yml和.properties的配置方式，虽各有优劣长短，但功效是一样的，我们这里将采用 .yml 的方式，文件内容如下：

```

application.yml
Xml代码 ✨ ☆
server:
  port: 8090
  application:
    name: sec

spring:
  thymeleaf:
    mode: HTML5

```

```
encoding: UTF-8
content-type: text/html
cache: false           #开发时关闭缓存,不然没法看到实时页面!
prefix: classpath:/public/  #配置页面文件路径
suffix: .html          #配置页面默认后缀

datasource:
url: jdbc:mysql://127.0.0.1:3306/sec?useUnicode=true&characterEncoding=UTF-8
username: root
password: *****
```

这个配置文件就是设定一下服务端口啊，服务名称啊，还有thymeleaf相关的一些路径配置，以及一些数据源待用的参数，这个文件的配置参数会被系统默认加载，需要时直接取用，很方便。然后在主路径下创建一个含main方法的SecApplication类，做启动入口，如下：

```
SecApplication.java
Java代码 ☆ ☆
package com.veiking;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
/**
 * 项目启动入口
 * @author Veiking
 */
@SpringBootApplication
public class SecApplication {
    public static void main(String[] args) {
        SpringApplication.run(SecApplication.class, args);
    }
}
```

注意加标签@SpringBootApplication，表示这将是按照 Spring boot 项目的形式运行。然后直接右键运行启动，留意下输出窗口，看看什么情况，启动成功，注意，输出栏的日志里很突兀的大了这样一行代码：Using generated security password: XXXX7e44-e83c-460a-aeef-94249316XXXX，这个是 Spring Security 自带默认的，用户名为user，密码就是这串UUID一样的串儿，接下来，我们浏览器输入：http://localhost:8090，敲回车，自动跳转到了http://localhost:8090/login的路径，我们可以看到一个框架本身自带的登录页面：

Login with Username and Password

User:

Password:

Login

我们在窗口输入默认的用户名密码，提交，就得到了这样一个页面：

Welcome!

Click here to see a greeting.

好了，初步的 Spring Security 项目验证通过，项目创建完成。

三、数据库信息的创建

这一波操作我们要创建本次实现要用的数据库表了，按照一般节奏，我们先来五张表：s_user、s_role、s_permission 和 s_user_role、s_role_permission，简单介绍下，就是用户、角色、权限资源和他们的关联关系表，他们结构如下：

s_user

名	类型	长度	允许空值	
id	int	11	<input type="checkbox"/>	1
name	varchar	32	<input checked="" type="checkbox"/>	
password	varchar	32	<input checked="" type="checkbox"/>	

s_role

名	类型	长度	允许空值	
id	int	11	<input type="checkbox"/>	1
role	varchar	32	<input checked="" type="checkbox"/>	
describe	varchar	32	<input checked="" type="checkbox"/>	

s_permission

名	类型	长度	允许空值	
id	int	11	<input type="checkbox"/>	1
permission	varchar	32	<input checked="" type="checkbox"/>	
url	varchar	32	<input checked="" type="checkbox"/>	
describe	varchar	32	<input checked="" type="checkbox"/>	

s_user_role

名	类型	长度	允许空值	
fk_role_id	int	11	<input checked="" type="checkbox"/>	
fk_permission_id	int	11	<input checked="" type="checkbox"/>	

s_role_permission

名	类型	长度	允许空值	
fk_user_id	int	11	<input checked="" type="checkbox"/>	
fk_role_id	int	11	<input checked="" type="checkbox"/>	

我们顺便贴上结构代码，以便使用：

Sql代码 ☆

-- Table structure for `s_user`

```
DROP TABLE IF EXISTS `s_user`;
CREATE TABLE `s_user` (
  `id` int(11) NOT NULL,
  `name` varchar(32) DEFAULT NULL,
  `password` varchar(32) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

-- Table structure for `s_user_role`

```
DROP TABLE IF EXISTS `s_user_role`;
CREATE TABLE `s_user_role` (
  `fk_user_id` int(11) DEFAULT NULL,
  `fk_role_id` int(11) DEFAULT NULL,
  KEY `union_key` (`fk_user_id`,`fk_role_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

-- Table structure for `s_role`

```
DROP TABLE IF EXISTS `s_role`;
CREATE TABLE `s_role` (
  `id` int(11) NOT NULL,
  `role` varchar(32) DEFAULT NULL,
  `describe` varchar(32) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

-- Table structure for `s_role_permission`

```
DROP TABLE IF EXISTS `s_role_permission`;
CREATE TABLE `s_role_permission` (
  `fk_role_id` int(11) DEFAULT NULL,
  `fk_permission_id` int(11) DEFAULT NULL,
  KEY `union_key` (`fk_role_id`,`fk_permission_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

-- Table structure for `s_permission`

```
DROP TABLE IF EXISTS `s_permission`;
CREATE TABLE `s_permission` (
  `id` int(11) NOT NULL,
  `permission` varchar(32) DEFAULT NULL,
  `url` varchar(32) DEFAULT NULL,
  `describe` varchar(32) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

接下来我们新增一些用户数据，admin、veiking、xiaoming，添加一些记录，大概意思是，admin拥有所有权限，veiking只有hello、index相关权限，xiaoming什么权限都没有，添加数据记录的脚本如下：

Sql代码 ⚡ ☆

```
-----
-- Records of s_user
-----
INSERT INTO `s_user` VALUES ('1', 'admin', 'admin');
INSERT INTO `s_user` VALUES ('2', 'veiking', 'veiking');
INSERT INTO `s_user` VALUES ('3', 'xiaoming', 'xiaoming');

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

-----
-- Records of s_role
-----
INSERT INTO `s_role` VALUES ('1', 'R_ADMIN', '大总管，所有权限');
INSERT INTO `s_role` VALUES ('2', 'R_HELLO', '说hello相关的权限');

-----
-- Records of s_role_permission
-----
INSERT INTO `s_role_permission` VALUES ('1', '1');
INSERT INTO `s_role_permission` VALUES ('1', '2');
INSERT INTO `s_role_permission` VALUES ('1', '3');
INSERT INTO `s_role_permission` VALUES ('2', '1');
INSERT INTO `s_role_permission` VALUES ('2', '3');

-----
-- Records of s_permission
-----
INSERT INTO `s_permission` VALUES ('1', 'P_INDEX', '/index', 'index页面资源');
INSERT INTO `s_permission` VALUES ('2', 'P_ADMIN', '/admin', 'admin页面资源');
INSERT INTO `s_permission` VALUES ('3', 'P_HELLO', '/hello', 'hello页面资源');
```

好了，数据库表相关的内容是准备完成。

四、测试页面的准备

紧接着我们创建一些用来测试检验效果的页面：login.html、index、admin、hello 等页面，其中 login.html 是用来检验登录效果的，代码如下：

login.html

Html代码 ⚡ ☆

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta content="text/html; charset=UTF-8"/>
<title>登录</title>
<link rel="stylesheet" th:href="@{css/bootstrap.min.css}"/>
<style type="text/css">
body { padding: 20px; }
.starter-template { width: 350px; padding: 0 40px; text-align: center; }
</style>
</head>
<body>


    <a th:href="@{/index}"> INDEX</a>
    <a th:href="@{/admin}"> | ADMIN</a>
    <a th:href="@{/hello}"> | HELLO</a>
    <br/>

<hr/>
<div class="starter-template">
    <p th:if="{param.logout}" class="bg-warning">已成功注销
    有错误，请重试
    <h2>使用用户名密码登录</h2>
    <form name="form" th:action="@{/login}" action="/login" method="POST">
        <div class="form-group">
            <label for="username">账号</label>
            <input type="text" class="form-control" name="username" value="" placeholder="账号" />
```



```
}
}
```

SRole.java


Java代码 

```
package com.veiking.sec.bean;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

/**
 * 角色信息
 * @author Veiking
 */
@Data
@NoArgsConstructor
@AllArgsConstructor
public class SRole {
    private int id;
    private String role;
    private String describe;
}
```

SPermission.java

Java代码 

```
package com.veiking.sec.bean;


import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

/**
 * 访问资源信息
 * @author Veiking
 */
@Data
@NoArgsConstructor
@AllArgsConstructor
public class SPermission {
    private int id;
    private String permission;
    private String url;
    private String describe;
}
```

注意@Data、@NoArgsConstructor、@AllArgsConstructor这些注解，都是lombok帮助处理set/get和全参无参构造方法的，如果不喜欢，自行替换即可。

然后来处理查询接口，我们这里采用的是 mybatis 框架的方式，好了，创建几个对应的dao，代码如下：

SUserDao.java


Java代码 

```
package com.veiking.sec.dao;

import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Select;
import com.veiking.sec.bean.SUser;

/**
 * 用户信息查询
 * @author Veiking
 */
@Mapper
public interface SUserDao {
    /**
     * 根据用户名获取用户
     *
     * @param name
     * @return
     */
    @Select(value = " SELECT su.* FROM s_user su WHERE su.name = #{name} ")
    public SUser findSUserByName(String name);
}
```

SRoleDao.java

Java代码 

```
package com.veiking.sec.dao;
```



```

import java.util.List;
import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Select;
import com.veiking.sec.bean.SRole;
/**
 * 角色信息查询
 * @author Veiking
 */
@Mapper
public interface SRoleDao {
    /**
     * 根据用户ID获取角色列表
     * @param sUserId
     * @return
     */
    @Select(value=" SELECT sr.* FROM s_role sr " +
        " LEFT JOIN s_user_role sur ON sr.id = sur.fk_role_id " +
        " LEFT JOIN s_user su ON sur.fk_user_id = su.id " +
        " WHERE su.id = #{sUserId} ")
    public List<SRole> findSRoleListBySUserId(int sUserId);

    /**
     * 根据资源路径获取角色列表
     * @param sPermissionUrl
     * @return
     */
    @Select(value=" SELECT sr.* FROM s_role sr " +
        " LEFT JOIN s_role_permission srp ON sr.id = srp.fk_role_id " +
        " LEFT JOIN s_permission sp ON srp.fk_permission_id = sp.id " +
        " WHERE sp.url = #{sPermissionUrl} ")
    public List<SRole> findSRoleListBySPermissionUrl(String sPermissionUrl);
}

```

SPermissionDao.java

Java代码 ☆ ☆

```
package com.veiking.sec.dao;
```

```
import java.util.List;
```

```
import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Select;
```

```
import com.veiking.sec.bean.SPermission;
```

```

/**
 * 资源权限信息查询
 * @author Veiking
 */
@Mapper
public interface SPermissionDao {
    /**
     * 根据用户ID获取资源权限列表
     * @param sUserId
     * @return
     */
    @Select(value=" SELECT * FROM s_permission sp " +
        " LEFT JOIN s_role_permission srp ON sp.id = srp.fk_permission_id " +
        " LEFT JOIN s_role sr ON srp.fk_role_id = sr.id " +
        " LEFT JOIN s_user_role sur ON sr.id = sur.fk_role_id " +
        " LEFT JOIN s_user su ON sur.fk_user_id = su.id " +
        " WHERE su.id = #{sUserId} ")
    public List<SPermission> findSPermissionListBySUserId(int sUserId);


    /**
     * 根据资源路径获取资源权限列表
     * @param sPermissionUrl
     * @return
     */
    @Select(value=" SELECT * FROM s_permission sp WHERE sp.url = #{sPermissionUrl} ")
    public List<SPermission> findSPermissionListBySPermissionUrl(String sPermissionUrl);
}

```

请注意，这里的几个Dao查询接口是使用注解的方式实现谁，当然，一般mybatis框架通常使用的方式是dao接口+xml脚本，当然个人也是习惯用xml实现较为复杂逻辑的脚本，但是在相对简单逻辑的操作上，直接用注解的方式是清爽的不能再清爽；两者在实际运用中是等效的，也是可以一同使用。

这几个接口的主要作用是：通过用户名（登录名）来获取用户信息；通过用户ID、资源路径（请求路径）来获取角色列表和权限资源列表。紧接着，本着编程习惯，我们再搞一个服务接口，将上边几个dao的功能整合，统一对外提供数据服务：

SecurityDataService.java

Java代码 

```

package com.veiking.sec.service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.veiking.sec.bean.SPermission;
import com.veiking.sec.bean.SRole;
import com.veiking.sec.bean.SUser;
import com.veiking.sec.dao.SPermissionDao;
import com.veiking.sec.dao.SRoleDao;
import com.veiking.sec.dao.SUserDao;

/**
 * Security 数据服务
 *
 * @author Veiking
 */
@Service
public class SecurityDataService {
    @Autowired
    private SUserDao sUserDao;
    @Autowired
    private SRoleDao sRoleDao;
    @Autowired
    private SPermissionDao sPermissionDao;

    public SUser findSUserByName(String name) {
        return sUserDao.findSUserByName(name);
    }

    public List<SRole> findSRoleListBySUserId(int sUserId) {
        return sRoleDao.findSRoleListBySUserId(sUserId);
    }

    public List<SRole> findSRoleListBySPermissionUrl(String sPermissionUrl) {
        return sRoleDao.findSRoleListBySPermissionUrl(sPermissionUrl);
    }

    public List<SPermission> findSPermissionListBySUserId(int sUserId) {
        return sPermissionDao.findSPermissionListBySUserId(sUserId);
    }

    public List<SPermission> findSPermissionListBySPermissionUrl(String sPermissionUrl) {
        return sPermissionDao.findSPermissionListBySPermissionUrl(sPermissionUrl);
    }
}

```

这个service没有额外的操作，仅仅是传递dao的功能，OK，到此，Spring Security 需要用的数据服务等一些准备部分，我们都已经准备好了，下面的环节，就是重点了。

六、重点:Spring Security之用户认证


经过一番相当罗嗦的铺垫，终于迎来了正题，我们将在接下来的环节里，讲述 Spring Security 认证有关的东西。

首先，再次回顾，Spring Security 认证有关的重要类或接口:UserDetails、UserDetailsService、AuthenticationProvider，我们将尝试自定义封装UserDetails，经由UserDetailsService提供给AuthenticationProvider，然后和请求消息中获取的用户信息进行对比认证。

首先，为了刻意的来区分认证和鉴权这里啊范畴，我们先来卖个关子，在包主路径下创建俩包:authentication、authorization，这俩单词简直是很像了，也是特意才用这两个单词，是看到有位前辈在博客中调侃了他们，印象深刻:authentication即认证，authorization即鉴权，注意字母微小的差异下在逻辑实现中不同的含义。

好，在authentication包下来完成我们 Spring Security 的认证，先新建一个 VUserDetails 类来实现 UserDetails（注:在此，所有的重新实现，都将在原类或接口名称前缀加大写的V，此处仅为示例,如有仿例操作，请根据个人习惯；包括之前的类或接口名，也不是很符合java推荐的命名规则，这只是为了在名称上强调而强调，勿在意，更勿仿效），代码如下：

VUserDetails.java

Java代码 

```

package com.veiking.sec.authentication;

```

```

import java.util.Collection;
import java.util.List;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

```

```

import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.AuthorityUtils;
import org.springframework.security.core.userdetails.UserDetails;

import com.google.gson.Gson;
import com.veiking.sec.bean.SPermission;
import com.veiking.sec.bean.SRole;
import com.veiking.sec.bean.SUser;
/**
 * 用户信息的封装，包含用户名称密码及用户状态、权限等信息
 * @author Veiking
 */
public class VUserDetails extends SUser implements UserDetails{

    private static final long serialVersionUID = 1L;
    Gson gson = new Gson();
    Logger logger = LoggerFactory.getLogger(this.getClass());
    //用户角色列表
    private List<SRole> sRoleList = null;
    //用户资源权限列表
    private List<SPermission> sPermissionList = null;
    /**
     * 注意后边的这两个参数：sRoleList、sPermissionList
     * @param sUser
     * @param sRoleList
     * @param sPermissionList
     */
    public VUserDetails(SUser sUser, List<SRole> sRoleList, List<SPermission> sPermissionList) {
        super(sUser);
        this.sRoleList = sRoleList;
        this.sPermissionList = sPermissionList;
    }
    /**
     * 获取用户权限列表方法
     * 可以理解成，返回了一个List<String>，之后所谓的权限控制、鉴权，其实就是跟这个list里的String进行对比
     * 这里处理了角色和资源权限两个列表，可以这么理解，
     * 角色是权限的抽象集合，是为了更方便的控制和分配权限，而真正颗粒化细节方面，还是需要资源权限自己来做
     */
    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        StringBuilder authoritiesBuilder = new StringBuilder("");
        List<SRole> tempRoleList = this.getsRoleList();
        if (null != tempRoleList) {
            for (SRole role : tempRoleList) {
                authoritiesBuilder.append(",").append(role.getRole());
            }
        }
        List<SPermission> tempPermissionList = this.getsPermissionList();
        if (null != tempPermissionList) {
            for (SPermission permission : tempPermissionList) {
                authoritiesBuilder.append(",").append(permission.getPermission());
            }
        }
        String authoritiesStr = "";
        if(authoritiesBuilder.length()>0) {
            authoritiesStr = authoritiesBuilder.deleteCharAt(0).toString();
        }
        logger.info("VUserDetails getAuthorities [authoritiesStr={} ", authoritiesStr);
        return AuthorityUtils.commaSeparatedStringToAuthorityList(authoritiesStr);
    }

    @Override
    public String getPassword() {
        return super.getPassword();
    }

    @Override
    public String getUsername() {
        return super.getName();
    }

    /**
     * 判断账号是否已经过期，默认没有过期
     */
    @Override
    public boolean isAccountNonExpired() {
        // TODO Auto-generated method stub
        return true;
    }
}

```

```

/**
 * 判断账号是否被锁定，默认没有锁定
 */
@Override
public boolean isAccountNonLocked() {
    return true;
}

/**
 * 判断信用凭证是否过期，默认没有过期
 */
@Override
public boolean isCredentialsNonExpired() {
    return true;
}

/**
 * 判断账号是否可用，默认可用
 */
@Override
public boolean isEnabled() {
    return true;
}

public List<SRole> getRoleList() {
    return sRoleList;
}

public void setRoleList(List<SRole> sRoleList) {
    this.sRoleList = sRoleList;
}

public List<SPermission> getPermissionList() {
    return sPermissionList;
}

public void setPermissionList(List<SPermission> sPermissionList) {
    this.sPermissionList = sPermissionList;
}
}



```

注意这个VUserDetails，它继承SUser并实现了UserDetails，这个类主要功能就是封装用户信息，包括从SUser继承来的用户名密码等属性，还有两个角色和权限的列表。注意这个getAuthorities()，这个方法主要工作是提供一组框架定义的权限列表，可以留意下源码，这个并没有定义具体类型，我们这里就用String类型实现这个权限。

这里还要解释下，我们在getAuthorities方法里分别循环了两个列表来加工 Spring Security 需要权限信息，即 tempRoleList 和 tempPermissionList，可以这样理解，角色和权限的概念，角色本身是权限的抽象集合，是协助我们开发管理的東西，真正意义的东西还是颗粒细小的权限。添个插曲，在本人最初接触到权限设计的时候，总是傻傻的被二者的关系搞晕，加上一些实际应用的系统还乐此不疲的在权限命名上"ROLE"来"ROLE"去的，甚至一些方法命名本身也在混淆这二者(怀疑可能是英语的使用习惯之类的原因)，导致早先的我常常陷入对二者的理解困惑上，当然现在清晰的很多：在大块儿整体性的权限控制上，角色控制为主；细化到页面小块儿、按钮级别的，权限控制为主；一般再加上访问URL的过滤鉴权，基本上一套强壮的权限控制体系是稳稳的在这儿了。

最后注意下代码里的几个isXXX方法，这些是一些细节补充，一般默认，也可以重写控制下逻辑；紧接着我们新建一个 VUserDetailsService 类，来实现 UserDetailsService，代码如下：

VUserDetailsService.java

Java代码  

```
package com.veiking.sec.authentication;
```

```
import java.util.List;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
```

```
import com.veiking.sec.bean.SPermission;
import com.veiking.sec.bean.SRole;
import com.veiking.sec.bean.SUser;
import com.veiking.sec.service.SecurityDataService;
```

```
/**
 * 提供用户信息封装服务
 * @author Veiking
 */
@Service
```

```

public class VUserDetailsService implements UserDetailsService {

    @Autowired
    SecurityDataService securityDataService;
    /**
     * 根据用户输入的用户名返回数据源中用户信息的封装，返回一个UserDetails
     */
    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        SUser sUser = securityDataService.findSUserByName(username);
        //用户角色列表
        List<SRole> sRoleList = securityDataService.findSRoleListBySUserId(sUser.getId());
        //用户资源权限列表
        List<SPermission> sPermissionList = securityDataService.findSPermissionListBySUserId(sUser.getId());
        return new VUserDetails(sUser, sRoleList, sPermissionList);
    }
}

```

这个类基本上没啥好说的，服务提供者，就是一个搬运工，看这个loadUserByUsername () 方法，拿到用户基本信息、角色列表和资源权限列表后，构造一个 VUserDetails 对象，OK返回。接下来是一个小重点，我们创建一个 VAuthenticationProvider 类来实现 AuthenticationProvider，代码如下：

VAuthenticationProvider.java

Java代码 ☆ ☆

```

package com.veiking.sec.authentication;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.AuthenticationProvider;
import org.springframework.security.authentication.BadCredentialsException;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.AuthenticationException;
import org.springframework.stereotype.Component;

import com.google.gson.Gson;
/**
 * 认证提供者，校验用户，登录名密码，以及向系统提供一个用户信息的综合封装
 * @author Veiking
 */
@Component
public class VAuthenticationProvider implements AuthenticationProvider {
    Gson gson = new Gson();
    Logger logger = LoggerFactory.getLogger(this.getClass());
    @Autowired
    VUserDetailsService vUserDetailsService;
    /**
     * 首先，在用户登录的时候，系统将用户输入的的用户名和密码封装成一个Authentication对象
     * 然后，根据用户名去数据源中查找用户的数据，这个数据是封装成的VUserDetails对象
     * 接着，将两个对象进行信息比对，如果密码正确，通过校验认证
     * 最后，将用户信息（含身份信息、细节信息、密码、权限等）封装成一个对象，此处参考UsernamePasswordAuthenticationToken
     * 最后，会将这个对象交给系统SecurityContextHolder中（功能类似Session），以便后期随时取用
     */
    @Override
    public Authentication authenticate(Authentication authentication) throws AuthenticationException {
        String username = authentication.getName();
        String password = authentication.getCredentials().toString();
        logger.info("VAuthenticationProvider authenticate login user [username={}, password={}]", username, password);
        VUserDetails vUserDetails = (VUserDetails)vUserDetailsService.loadUserByUsername(username);
        logger.info("VAuthenticationProvider authenticate vUserDetails [vUserDetails={}]", gson.toJson(vUserDetails));
        if(vUserDetails == null){
            throw new BadCredentialsException("用户没有找到");
        }
        if (!password.equals(vUserDetails.getPassword())) {
            logger.info("VAuthenticationProvider authenticate BadCredentialsException [inputPassword={}, DBPassword={}", password, vUserDetails.getPassword());
            throw new BadCredentialsException("密码错误");
        }
        //认证校验通过后，封装UsernamePasswordAuthenticationToken返回
        return new UsernamePasswordAuthenticationToken(vUserDetails, password, vUserDetails.getAuthorities());
    }

    @Override
    public boolean supports(Class<?> authentication) {
        return true;
    }
}

```

```

}
```

这个实现类的核心就是authenticate方法，一步步看，系统会将用户在登录请求操作的时候，把输入的用户名密码等，封装到一个Authentication对象中，我们从这个对象里拿到用户名，通过 VUserDetailsService 获取到 VUserDetails 对象，然后拿这个对象的密码属性，和请求Authentication对象中获取的密码进行对比，如果一切OK，验证功过，然后再将这些所有信息，整体封装成一个Authentication对象（这里边我们用的是 UsernamePasswordAuthenticationToken），交给系统框架，后期可以随时取用。

好了，经过上面的工作，用户认证的逻辑已经完事儿了，我们要做访问工作，这里还要做些配置操作，这里分别新建俩类，代码如下：

WebMvcConfig.java

Java代码 ☆ ☆

```

package com.veiking.sec;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.ViewControllerRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
/**
 * 访问路径配置类
 * 可以理解成做简单访问过滤的，转发到相应的视图页面
 * @author Veiking
 */
@Configuration
public class WebMvcConfig implements WebMvcConfigurer {
    @Override
    public void addViewControllers(ViewControllerRegistry registry) {
        registry.addViewController("/login").setViewName("login");
        registry.addViewController("/").setViewName("index");
        registry.addViewController("/index").setViewName("index");
    }
}
```

PageController.java

Java代码 ☆ ☆

```

package com.veiking.sec.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
/**
 * 请求页面分发，注意和WebMvcConfig的对比，功能类似
 * @author Veiking
 */
@Controller
public class PageController {

    @RequestMapping("/admin")
    public String admin(Model model, String tt) {
        return "admin";
    }

    @RequestMapping("/hello")
    public String hello(Model model, String tt) {
        return "hello";
    }
}
```

WebMvcConfig 是一个简单的路径映射，功能跟在 PageController中实现的差不多，之所以多写一个PageController，是因为后边会有其他的功能演示。

然后我们还需创建一个 WebSecurityConfig 类来继承 WebSecurityConfigurerAdapter，代码如下：

WebSecurityConfig.java

Java代码 ☆ ☆

```

package com.veiking.sec;

import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.builders.WebSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
/**
 * Security 主配置文件
 * @author Veiking
 */
@Configuration
```

@EnableWebSecurity //开启Spring Security的功能

```
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    /**
     * 定义不需要过滤的静态资源 （等价于HttpSecurity的permitAll）
     */
    @Override
    public void configure(WebSecurity webSecurity) throws Exception {
        webSecurity.ignoring().antMatchers("/css/**");
    }

    /**
     * 安全策略配置
     */
    @Override
    protected void configure(HttpSecurity httpSecurity) throws Exception {
        httpSecurity
            .authorizeRequests()
            // 对于网站部分资源需要指定鉴权
            // .antMatchers("/admin/**").hasRole("ADMIN")
            // 除上面外的所有请求全部需要鉴权认证
            .anyRequest().authenticated().and()
            // 定义当需要用户登录时候，转到的登录页面
            .formLogin().loginPage("/login").defaultSuccessUrl("/index").permitAll().and()
            // 定义登出操作
            .logout().logoutSuccessUrl("/login?logout").permitAll().and()
            .csrf().disable()
            ;
        // 禁用缓存
        httpSecurity.headers().cacheControl();
    }
}
```

这个类是使用 Spring Security 的主配置入口，在这个配置文件中，正式启用 Spring Security 包括我们之前所讲的所有功能，这里主要留意一下负责安全策略配置的 configure () 方法，这个方法里可以定义登录登出等操作细节，以及一些静态资源的权限忽略之类的，甚至也是可以直接手动配权限的。

一切完事儿，我们运行 SecApplication ，开始验证之旅：

[INDEX](#) | [ADMIN](#) | [HELLO](#)

使用用户名密码登录

账号

veiking

密码

.....

登录

在登录页面，输入用户名密码:admin/admin，登录看看，随便点点跳跳，换成veiking/veiking试试，也可以输错试试，再试下登出:

INDEX

你好: [veiking](#)

[INDEX](#) | [ADMIN](#) | [HELLO](#)

注销

有错误，请重试

使用用户名密码登录

账号

veiking

密码

.....

登录

已成功注销

使用用户名密码登录

账号

veiking

密码

.....

登录

好了，这个简单的用户认证功能看来是可以了，我们接下来看看如何控制权限。

七、重点:Spring Security之鉴权-初试

认证OK，回想下，是否还记得，在VAuthenticationProvider的校验环节，我们在封装返回给系统的Authentication对象里，是提供了vUserDetails.getAuthorities()这个认证列表的，接下来看看，这个被交给系统的认证列表，是怎么体现的。

我们打开 hello.html 页面，在其中的几个导航跳转的信息上，添加一个 sec:authorize="hasAuthority('XXX')" 的代码，这样子的脚本，大概意思就是，只有名为'XXX'的角色或者权限的用户，登录之后才可以看到，如下：

```
hello.html
Html代码
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org"
      xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity4">
<head>
<meta content="text/html;charset=UTF-8"/>
<title>HELLO</title>
<link rel="stylesheet" th:href="@{css/bootstrap.min.css}"/>
<style type="text/css">
body { padding: 40px; }
</style>
</head>
<body>
  <h1>HELLO</h1>
  <br/>你好: <a sec:authentication="name"></a>

  <a sec:authorize="hasAuthority('P_INDEX')" th:href="@{/index}"> INDEX</a>
  <a sec:authorize="hasAuthority('P_ADMIN')" th:href="@{/admin}"> | ADMIN</a>
  <a sec:authorize="hasAuthority('P_HELLO')" th:href="@{/hello}"> | HELLO</a>
  <br/><br/>
  <form th:action="@{/logout}" method="post" sec:authorize="hasAuthority('R_ADMIN')">
    <input type="submit" class="btn btn-primary" value="注销"/>
  </form>
```



```
</body>
</html>
```

(注意, 在页面中使用 Spring Security 相关脚本, 要在<html>标签处添加 xmlns:th="http://www.thymeleaf.org"、xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity4" 等约束规范) 重新启动后, 分别用不同的用户, 登录后跳转到hello页面查看, 这时候可以看到, admin用户拥有较多权限, 都可以看到, veiking 用户只能看到index和hello导航, 而 xiaoming 用户什么都看不到了, 并且他们都不能看到注销按钮, 就是这个效果: [/size]

HELLO

你好: [veiking](#)
[INDEX](#) | [HELLO](#)

HELLO

你好: [xiaoming](#)

上边是从页面层面来进行权限控制的, 注意hasAuthority('XXX')中, 有用到R_ADMIN、P_ADMIN、P_HELLO不同类型的权限字眼, 包含角色和权限, 这个控制的颗粒度没有绝对的, 只要设计成规律可循、操作可行方案即可。

接下来, 打开 PageController, 在/admin处添加标签:@PreAuthorize("hasAuthority('R_ADMIN')"), 如下:

PageController.java

Java代码 ⭐ ☆

```
package com.veiking.sec.controller;
```

```
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
```

```
/**
```

```
 * 请求页面分发, 注意和WebMvcConfig的对比, 功能类似
```

```
 * @author Veiking
```

```
 */
```

```
@Controller
```

```
public class PageController {
```

```
    @RequestMapping("/admin")
```

```
    @PreAuthorize("hasAuthority('R_ADMIN')")
```

```
    public String admin(Model model, String tt) {
```

```
        return "admin";
```

```
    }
```

```
    @RequestMapping("/hello")
```

```
    public String hello(Model model, String tt) {
```

```
        return "hello";
```

```
    }
```

```
}
```

注意, 这个操作还需要在 WebSecurityConfig 类中加 @EnableGlobalMethodSecurity(prePostEnabled=true) 标签来, 开启注解控制权限, 然后配置 authenticationManagerBean 以供支持, 代码如下:

WebSecurityConfig.java

Java代码 ⭐ ☆

```
package com.veiking.sec;
```

```
import org.springframework.context.annotation.Bean;
```

```
import org.springframework.context.annotation.Configuration;
```

```

import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.builders.WebSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
/**
 * Security 主配置文件
 * @author Veiking
 */
@Configuration
@EnableWebSecurity //开启Spring Security的功能
@EnableGlobalMethodSecurity(prePostEnabled=true)//开启注解控制权限
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    /**
     * 定义不需要过滤的静态资源（等价于HttpSecurity的permitAll）
     */
    @Override
    public void configure(WebSecurity webSecurity) throws Exception {
        webSecurity.ignoring().antMatchers("/css/**");
    }

    /**
     * 安全策略配置
     */
    @Override
    protected void configure(HttpSecurity httpSecurity) throws Exception {
        httpSecurity
            .authorizeRequests()
            // 对于网站部分资源需要指定鉴权
            .antMatchers("/admin/**").hasRole("ADMIN")
            // 除上面外的所有请求全部需要鉴权认证
            .anyRequest().authenticated().and()
            // 定义当需要用户登录时候，转到的登录页面
            .formLogin().loginPage("/login").defaultSuccessUrl("/index").permitAll().and()
            // 定义登出操作
            .logout().logoutSuccessUrl("/login?logout").permitAll().and()
            .csrf().disable()
            ;
        // 禁用缓存
        httpSecurity.headers().cacheControl();
    }

    /**
     * 开启注解控制权限
     * 见Controller 中 @PreAuthorize("hasAuthority('XXX')")
     */
    @Bean
    @Override
    public AuthenticationManager authenticationManagerBean() throws Exception {
        return super.authenticationManagerBean();
    }
}

```

然后再次启动，用veiking登录，INDEX页面点击ADMIN导航——好，403 Forbidden了，对，被拦了，就是这个效果。

以上这些，是简单的对 Spring Security 鉴权操作的一些尝试，当然，如果是小规模功能开发，这些是可以满足的，如果想追求更为灵活的控制，就要重新是实现下过滤机制，接下来我们就尝试下从对数据库层面的配置，实现权限的动态管理。

八、重点:Spring Security之鉴权-过滤器

上边我们已尝试了经通过页面脚本和注解这两种方式的权限控制，接下来，我们尝试下通过数据库的权限配置，来过滤用户操作请求的。

跟认证对应，我们新建一个包，authorization，然后在这个里面来实现过滤请求方式的鉴权:先写一个 VFilterInvocationSecurityMetadataSource 类，来实现 FilterInvocationSecurityMetadataSource，这个可以简单理解成权限资源管理器，它的工作是通过用户的请求地址，来获取访问这个地址所需的权限，代码如下：

```

FilterInvocationSecurityMetadataSource.java
Java代码 ☆ ☆
package com.veiking.sec.authorization;

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

```

```

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.access.ConfigAttribute;
import org.springframework.security.access.SecurityConfig;
import org.springframework.security.web.FilterInvocation;
import org.springframework.security.web.access.intercept.FilterInvocationSecurityMetadataSource;
import org.springframework.stereotype.Component;

import com.google.gson.Gson;
import com.veiking.sec.bean.SPermission;
import com.veiking.sec.bean.SRole;
import com.veiking.sec.service.SecurityDataService;
/**
 * 权限资源管理器
 * 根据用户请求的地址，获取访问该地址需要的所需权限
 * @author Veiking
 */
@Component
public class VFilterInvocationSecurityMetadataSource implements FilterInvocationSecurityMetadataSource {
    Gson gson = new Gson();
    Logger logger = LoggerFactory.getLogger(this.getClass());
    @Autowired
    SecurityDataService securityDataService;

    @Override
    public Collection<ConfigAttribute> getAttributes(Object object) throws IllegalArgumentException {
        //获取请求起源路径
        String requestUrl = ((FilterInvocation) object).getRequestUrl();
        logger.info("VFilterInvocationSecurityMetadataSource getAttributes [requestUrl={}]", requestUrl);
        //登录页面就不需要权限
        if ("/login".equals(requestUrl)) {
            return null;
        }
        //用来存储访问路径需要的角色或权限信息
        List<String> tempPermissionList = new ArrayList<String>();
        //获取角色列表
        List<SRole> sRoleList = securityDataService.findSRoleListBySPermissionUrl(requestUrl);
        logger.info("VFilterInvocationSecurityMetadataSource getAttributes [sRoleList={}]", gson.toJson(sRoleList));
        for(SRole sRole : sRoleList) {
            tempPermissionList.add(sRole.getRole());
        }
        //径获取资源权限列表
        List<SPermission> sPermissionList = securityDataService.findSPermissionListBySPermissionUrl(requestUrl);
        logger.info("VFilterInvocationSecurityMetadataSource getAttributes [sPermissionList={}]", gson.toJson(sPermissionList));
        for(SPermission sPermission : sPermissionList) {
            tempPermissionList.add(sPermission.getPermission());
        }
        //如果没有权限控制的url，可以设置都可以访问，也可以设置默认不许访问
        if(tempPermissionList.isEmpty()) {
            return null;//都可以访问
            //tempPermissionList.add("DEFAULT_FORBIDDEN");//默认禁止
        }
        String[] permissionArray = tempPermissionList.toArray(new String[0]);
        logger.info("VFilterInvocationSecurityMetadataSource getAttributes [permissionArray={}]", gson.toJson(permissionArray));
        return SecurityConfig.createList(permissionArray);
    }

    @Override
    public Collection<ConfigAttribute> getAllConfigAttributes() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public boolean supports(Class<?> clazz) {
        return true;
    }
}

```

接着访问需要的权限资源拿到了，就得有判断的地方，新建一个 VAccessDecisionManager 来实现 AccessDecisionManager，这个类可以理解成权限管理判断器，他的主要工作就是鉴权，通过拿到的访问路径所需的权限，和用户所拥有的权限进行对比，判断用户是否有权限访问，代码如下：

VAccessDecisionManager.java
Java代码 ☆ ☆

```
package com.veiking.sec.authorization;
```

```
import java.util.Collection;
```

```
import org.springframework.security.access.AccessDecisionManager;
import org.springframework.security.access.AccessDeniedException;
import org.springframework.security.access.ConfigAttribute;
import org.springframework.security.authentication.InsufficientAuthenticationException;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.stereotype.Component;
/**
```

```
 * 权限管理判断器|校验用户是否有权限访问请求资源
```

```
 * @author Veiking
```

```
 */
```

```
@Component
```

```
public class VAccessDecisionManager implements AccessDecisionManager {
```

```
    @Override
```

```
    public void decide(Authentication authentication, Object object, Collection<ConfigAttribute> configAttributes)
        throws AccessDeniedException, InsufficientAuthenticationException {
```

```
        //当前用户所具有的权限
```

```
        Collection<? extends GrantedAuthority> userAuthorityList = authentication.getAuthorities();
```

```
        //访问资源所需的权限信息
```

```
        Collection<ConfigAttribute> needAuthoritieList = configAttributes;
```

```
        //依次循环对比，发现有匹配的即返回
```

```
        for(ConfigAttribute needAuthoritie: needAuthoritieList) {
```

```
            String needAuthoritieStr = needAuthoritie.getAttribute();
```

```
            for (GrantedAuthority userAuthority : userAuthorityList) {
```

```
                String userAuthorityStr = userAuthority.getAuthority();
```

```
                if (needAuthoritieStr.equals(userAuthorityStr)) {
```

```
                    return;
```

```
                }
```

```
            }
```

```
        //执行到这里说明没有匹配到应有权限
```

```
        throw new AccessDeniedException("权限不足!");
    }
```

```
    @Override
```

```
    public boolean supports(ConfigAttribute attribute) {
```

```
        return true;
    }
```

```
    @Override
```

```
    public boolean supports(Class<?> clazz) {
```

```
        return true;
    }
```

```
}
```

最后，要写一个过滤器，提供上边这些功能的工作场所，创建 VFilterSecurityInterceptor 类，继承 AbstractSecurityInterceptor 并实现 Filter，这就是个鉴权过滤器，代码如下：

```
VFilterSecurityInterceptor.java
```

```
Java代码 ✨ ☆
```

```
package com.veiking.sec.authorization;
```

```
import java.io.IOException;
```

```
import javax.servlet.Filter;
```

```
import javax.servlet.FilterChain;
```

```
import javax.servlet.FilterConfig;
```

```
import javax.servlet.ServletException;
```

```
import javax.servlet.ServletRequest;
```

```
import javax.servlet.ServletResponse;
```

```
import javax.servlet.annotation.WebFilter;
```

```
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.boot.web.servlet.ServletComponentScan;
```

```
import org.springframework.security.access.SecurityMetadataSource;
```

```
import org.springframework.security.access.intercept.AbstractSecurityInterceptor;
```

```
import org.springframework.security.access.intercept.InterceptorStatusToken;
```

```
import org.springframework.security.web.FilterInvocation;
```

```
import org.springframework.stereotype.Component;
```

```
/**
```

```
 * 访问鉴权过滤器
```

```
 * 该过滤器的作用就是，用户请求时，提供权限资源管理器和权限判断器工作的场所，实现鉴权操作
```

```

* @author Veiking
*/
@Component
@WebServletComponentScan
@WebFilter(filterName="vFilterSecurityInterceptor",urlPatterns="/*")
public class VFilterSecurityInterceptor extends AbstractSecurityInterceptor implements Filter {

    @Autowired
    private VFilterInvocationSecurityMetadataSource vFilterInvocationSecurityMetadataSource;

    @Autowired
    public void setMyAccessDecisionManager(VAccessDecisionManager vAccessDecisionManager) {
        super.setAccessDecisionManager(vAccessDecisionManager);
    }

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {

    }

    @Override
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
        throws IOException, ServletException {
        FilterInvocation filterInvocation = new FilterInvocation(request, response, chain);
        invoke(filterInvocation);
    }

    public void invoke(FilterInvocation filterInvocation) throws IOException, ServletException {
        // filterInvocation里面有一个被拦截的url
        // 里面调用VFilterInvocationSecurityMetadataSource的getAttributes(Object object)这个方法获取filterInvocation对应的所有权限
        // 再调用VAccessDecisionManager的decide方法来校验用户的权限是否足够
        InterceptorStatusToken interceptorStatusToken = super.beforeInvocation(filterInvocation);
        try {
            // 执行下一个拦截器
            filterInvocation.getChain().doFilter(filterInvocation.getRequest(), filterInvocation.getResponse());
        } finally {
            super.afterInvocation(interceptorStatusToken, null);
        }
    }

    @Override
    public void destroy() {

    }

    @Override
    public Class<?> getSecureObjectClass() {
        return FilterInvocation.class;
    }

    @Override
    public SecurityMetadataSource obtainSecurityMetadataSource() {
        return this.vFilterInvocationSecurityMetadataSource;
    }
}

```

这里边注意 @ServletComponentScan 和 @WebFilter(filterName="vFilterSecurityInterceptor",urlPatterns="/*") 注解，这个是确保过滤器自动注册并工作，否则过滤器无效。

接下来启动项目，用个个用户登录去看看，尤其是veiking和xiaoming用户，访问没有权限的连接时果断遭到限制，403 Forbidden！

Whitelabel Error Page

This application has no explicit mapping for /error, so you are

Wed Aug 22 19:33:00 CST 2018
There was an unexpected error (type=Forbidden, status=403).
Forbidden

九、结语

好了，经过这么一番折腾，学习应用 Spring Security 框架该接触到的一些知识点，都有所体现了。权限控制的本质，就是对比校验，其一般体现方式，就是过滤器。Spring Security 是提供了一种相对比较好的表现形式，给出了一个优良的范式，敲示例代码的本身，更重要的应该是为了帮助理解和学习，而不是为了实现而实现。

本文是足够罗嗦，也是个人为了加深在理解记忆，但有些地方甚至也是错误的、不合乎规范的，希望大家不要被误导，这只能说是一个提供理解的参考，帮助大家从懵懂到懂;还有需要注意的是，因spring版本不同导致的一些细节差异，可能会有小坑，还是需要注意下的。文中所涉及代码最后都在附件中，感兴趣的同学可以自行下载。

还有，喜欢的，扫下支付宝家电红包吧，哈哈哈！



代码附件：

<http://dl2.iteye.com/upload/attachment/0130/5515/3b1eeefa-fa64-3dd2-97c0-753bd96c1acc.rar>

[sec.rar](#) (79.2 KB)



下载次数: 4

[查看图片附件](#)

2019开年报告：Java坐稳榜首，彻底甩掉C++!

厉害了，我的java!

1
顶
0
踩

分享到： 

[Spring回顾之八—— Quartz在集群、分布...](#)

2018-08-22 21:52

浏览 3982

[评论\(0\)](#)

分类:[编程语言](#)

[查看更多](#)[评论](#)[发表评论](#)[您还没有登录,请您登录后再发表评论](#)

相关资源推荐

[springBoot+springSecurity 数据库动态管理用户、角色、权限 \(二\)](#)

序：本文使用springboot+mybatis+ SpringSecurity 实现数据库动态的管理用户、角色、权限管理本文细分角色和权限，并将用户、角色、权限和资源均采用数据库存储，并且自定义过滤器，代替原有的FilterSecurityInterceptor过滤器，并分别实现AccessDecisionManager、InvocationSecurityMetadataSource

[springboot+mybatis+ SpringSecurity 实现用户角色数据库管理\(一\)](#)

本文使用springboot+mybatis+ SpringSecurity 实现用户权限数据库管理实现用户和角色用数据库存储，而资源(url)和权限的对应采用硬编码配置。也就是角色可以访问的权限通过硬编码控制。角色和用户的关系通过数据库配置控制本文用户和角色的关系是多对多的关系。SpringSecurity 验证帐号密码

AuthenticationManager调用Provider，provide

[Spring_Security实现动态权限管理](#)

我所理解的动态权限就是RBAC(Role-Based Access Control)。就是可以自定义角色，配置角色可以访问哪些URL。然后给不同的角色设置不同的角色。为什么用Spring Security? 听说Spring Security是基于Shiro的。Shiro没用过。之所以用Spring Security是因为它安全。废话！是因为可以帮你防御csrf等攻击。其实现现在的Chrome浏览器

[Spring+MyBatis实践——登录与权限控制](#)

Spring+MyBatis实践——登录与权限控制 1、实现用户登录功能：通过session来实现用户登录功能。在用户登录时，将用户的相关信息放在HttpSession对象用，其中HttpSession对象可以通过HttpServletRequest的getSession方法获得。同时，HttpSession对象对应Jsp内置对象session，在jsp页面中也可以通过session对象使用HandlerInterceptor实现系统权限管理

背景 前端效果类似于下图：父子级菜单分别对应不同的操作权限、访问路径等。数据库建三张表： 1.operate 页面菜单配置表 id即为权限id，url为菜单对应页面地址，type控制菜单是否显示。 2.action 请求url配置表 action为对应操作请求的url，operate即为对应操作权限，log控制是否打印日志。 3.account用户表保存用户信...

[springboot + springsecurity + mybatis + mysql实现用户登录验证、记住密码、退出功能](#)

springsecurity 主要的功能是 验证 和 授权。springsecurity主要的工作原理是：当我们在登录页面中输入用户名和密码之后首先会进入到UsernamePasswordAuthenticationToken验证(Authentication)，然后生成的Authentication会被交由AuthenticationManager来进行管理而AuthenticationManager...

[Spring Boot + Security + Mybatis 简单权限控制 \(入门 + 记录篇\)](#)

Spring Boot Security 权限 这两天研究了一下权限管理框架。。查阅资料的过程中，JAVA中常用的安全框架有Shiro和Spring Security。Shiro比Spring Security学习起来更加简单，功能够用。而这两天的学习中，就我自己的体会而言，学习Spring Security还是有一定难度的。虽然它的扩展性非常的好，我们可以重载它默认类，重写方法...

[SpringBoot通过自己的配置文件或者从数据库spring_security动态配置url权限](#)

我使用Springboot的时候想自己做自己的配置文件的，用不了xml就重写了过滤器 首先需要了解spring security内置的各种filter： Alias Filter Class Namespace Element or Attribute CHANNEL_FILTER ChannelProcessingFilter http/

[Spring Boot集成Security使用数据库用户角色权限ROLE_问题](#)

[springboot+springmvc+mybatis+layui实现登录用户菜单权限管理好例子](#)

作为中小项目的框架或脚手架非常好用的，有后台用户菜单管理权限认证等基本功能，都、齐全。

[基于springboot的RBAC权限管理演示系统](#)

这是一个RBAC权限管理系统，即基于角色的用户权限控制，使用springboot框架开发，UI使用的是layui。。演示地址：

http://116.196.66.248:8090/page/index 欢迎大家下载。。。另外，建议使用IDEA导入项目。。

[springBoot+springSecurity 动态管理Restful风格权限 \(三\)](#)

上一篇博客 springBoot+springSecurity 数据库动态管理用户、角色、权限 (二) 只是实现了用户、角色、权限的动态管理，但是其权限管理是有缺陷的，他不支持restful风格的接口权限管理，因为他无法区分客户端的请求方式。本片博客是为了弥补此缺陷的，本篇博客将在 springBoot+springSecurity 数据库动态管理用户、角色、权限 (二) 的基础上进行修改使其

[Spring Boot MyBatis 动态数据源切换、多数据源、读写分离](#)

项目地址：https://github.com/helloworlde/SpringBoot-DynamicDataSource/tree/dev 在 Spring Boot 应用中使用到了 MyBatis 作为持久层框架，添加多个数据源，实现读写分离，减少数据库的压力 在这个项目中使用注解方式声明要使用的数据源，通过 AOP 查找注解，从而实现数据源的动态切换；该项目为 Product

[4. Spring Boot Security角色管理持久化实现](#)

1.概述,在第三章里大家学会了怎么初步使用Spring Boot 结合Spring Security来实现权限控制和角色管理,但是我们发现无论是使用那种方式角色管理和权限控制全部是在 xml中或则 配置类中写的,没有实现持久化,本次就为大家讲解怎么实现.本小结为大家讲 用户角色管理的实现,至于权限控制来下一个小结会讲. 2.表和数据 PDM文件路径: https://github.co

[springboot-shrio-mybatis登录验证与权限控制](#)

springboot-shrio-mybatis 一、背景 最近做的一个springboot项目中用到权限控制，网上也看了其他springboot集成shiro进行权限控制的文档。大多文档用户与角色为多对多关系，角色与权限多对多，我的项目需求用户与角色为单对单，角色与权限多对多。所以自己重新整理了表结构完成。 二、表结构 /* Navicat MySQL Data Transfer

[springBoot 动态数据源以及Mybatis多数据源](#)

前言在开发过程中可能需要用到多个数据源，比如一个项目(MySQL)就是和(SQL Server)混合使用，就需要使用多数据源；如果业务场景比较复杂，可以使用动态数据源，灵活切换，典型的应用就是读写分离。下面分两个模块来配置数据源，大家可以根据自己实际情况配置。多数据源禁用

DataSourceAutoConfiguration如果DataSourceAutoConfiguration不禁用的话，就会报

[springboot使用mybatis多数据源动态切换的实现](#)

需求：项目使用了读写分离，或者数据进行了分库处理，我们希望在操作不同的数据库的时候，我们的程序能够动态的切换到相应的数据库，执行相关的操作。首先，我们需要一个能够正常运行的springboot项目，配置mybatis并且能够正常的操作数据库（增删查改）现在开始实现：思路：现在项目的结构设计基本上基于MVC的，那么数据库的操作集中在dao层完成，主要业务逻辑在service层处理，c...

[SpringBoot整合Springsecurity实现数据库登录以及权限控制](#)

我们今天使用SpringBoot来整合SpringSecurity，来吧，不多BB 首先呢，是一个SpringBoot 项目，连接数据库，这里我使用的是mybaties.mysql， 下面是数据库的表 DROP TABLE IF EXISTS `xy_role`; CREATE TABLE `xy_role` (`xyr_id` int(11) NOT NULL AUTO_INCRE...

SpringMVC+Spring+Mybatis +Annotation 实现方法、按钮级别的细粒度权限控制

本文转载自：http://blog.csdn.net/ycyk_168/article/details/18456631 随着企业信息化的不断深入，各种各样的信息系统成为提高企业运营及管理效率的必备工具，越来越多的企业核心机密如销售机会、客户资料、设计方案等通过信息系统存储、备案、流转，这些核心资料一旦外泄，势必对企业造成极大损失。科技时代，信息是企业生存的命脉，信息的安全也必然成为企业

动态菜单/权限管理的实现

权限管理序 现在基本上大大小小的系统都由权限分配这一基础功能，不同的用户看到的界面不一样，能够使用的功能也不会尽然相同 所以我要让我的系统做到超级管理员可以看到/操作所有界面，而新闻管理员只能看到新闻管理模块，仓库管理员只能看到仓库管理模块等等。。。思想 动态菜单：系统里面有很多菜单可以进行点击操作，但是不同的角色用户能够操做的菜单肯定是不相同的，那么，我就需要用角色来区分用户...

springBoot+security+mybatis 实现用户权限的数据库动态管理 - Veiking的博客 - ITeye博客 2018-08-22 24

[b][size=large]一、Spring Security 应用的概述[/size]

创建数据库动态管理用户、角色、权限和资源 - Veiking的博客 - ITeye博客 2018-02-05 1169

使用spring Security3的四种方法概述 那么在Spring ...

ERP生产管理系统方案



动态菜单/权限管理的实现 2018-07-16 769 权限管理序 现在基本上大大小小的系统都由权限分...

MySQL数据库的用户权限管理 2018-03-19 489

MySQL数据库的用户权限管理在数据库方面有两个...

apache_shiro_管理用户权限与数据库交互 2017-11-09 1

apache_shiro管理用户权限与数据库交互，实现基...

根据用户权限动态生成菜单 2007-10-15 78 现在项目里有这个需求：用户登录的时候可以选择...

18年，他们都在用拼接屏了..



java实现动态权限(菜单管理)动态添加菜单，动 2018-10-25 738

首先是数据库设计 Menu表（菜单表） Role表（角...

动态创建数据库表 2013-03-18 2939 以前对数据库的操作基本上处于数据行阶段，在得...

用户权限管理动态显示该用户的有什么权限菜单 2018-07-12 638

此时需要 四张表 :1 用户表(user)2 菜单表(menu)3 ...

基于 RBAC 简单实现动态菜单及权限管理 2018-03-09 2147

RBAC(Role-Based Access Control, 基于角色访问...

【转载】 【权限控制】角色访问动态生成用户权限 2013-12-12 38839

【转载】 【权限控制】角色访问动态生成用户权限...

深圳夹丝玻璃厂家推荐



Spring+MyBatis实践——登录与权限控制 2015-01-08 1389

Spring+MyBatis实践——登录与权限控制 1、实现...

权限管理系统中的根据用户角色动态生成用户权限 2017-10-03 13225

权限管理系统中的根据用户角色动态生成用户权限...

关于权限管理数据库需要用到多少张表这个问题，...

操作系统和数据库的特权用户的权限没有分离？
2018-09-29 908

一、建议处置措施：（1）要求：建议操作系统与...



微信客服



QQ客服

QQ客服

kefu@csdn.net

客服论坛

400-660-0108

工作时间 8:30-22:00

关于我们 招聘 广告服务 网站地图

百度提供站内搜索 京ICP证19004658号

©1999-2019 北京创新乐知网络技术有限公司

网络110报警服务 经营性网站备案信息

北京互联网违法和不良信息举报中心

中国互联网举报中心

Global site tag (gtag.js) - Google Analytics