

1. Shiro入门

ApacheShiro是一个功能强大且易于使用的Java安全框架，提供了认证，授权，加密，和会话管理。

Shiro有三大核心组件：

Subject：即当前用户，在权限管理的应用程序里往往需要知道谁能够操作什么，谁拥有操作该程序的权利，shiro中则需要通过Subject来提供基础的当前用户信息，Subject不仅仅代表某个用户，与当前应用交互的任何东西都是Subject，如网络爬虫等。所有的Subject都要绑定到SecurityManager上，与Subject的交互实际上是被转换为与SecurityManager的交互。

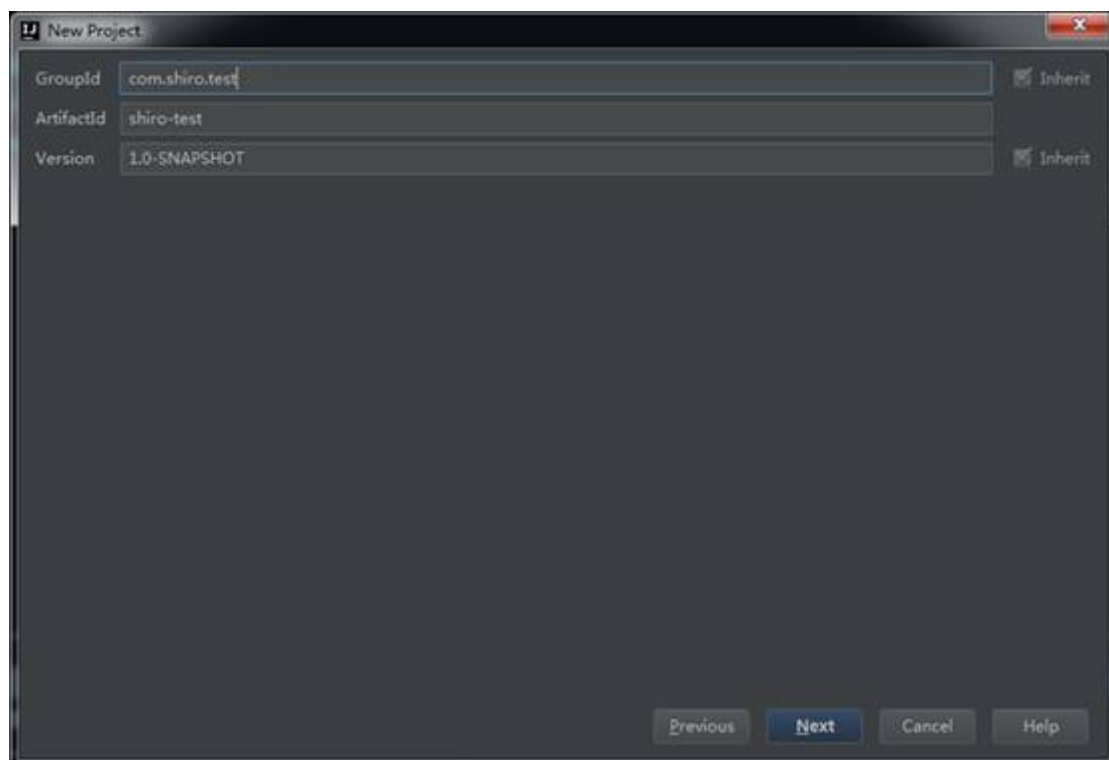
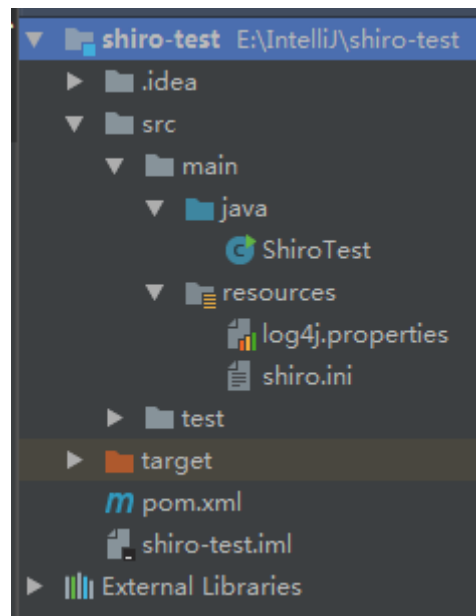
SecurityManager：即所有Subject的管理者，这是Shiro框架的核心组件，可以把他看做是一个Shiro框架的全局管理组件，用于调度各种Shiro框架的服务。作用类似于SpringMVC中的DispatcherServlet，用于拦截所有请求并进行处理。

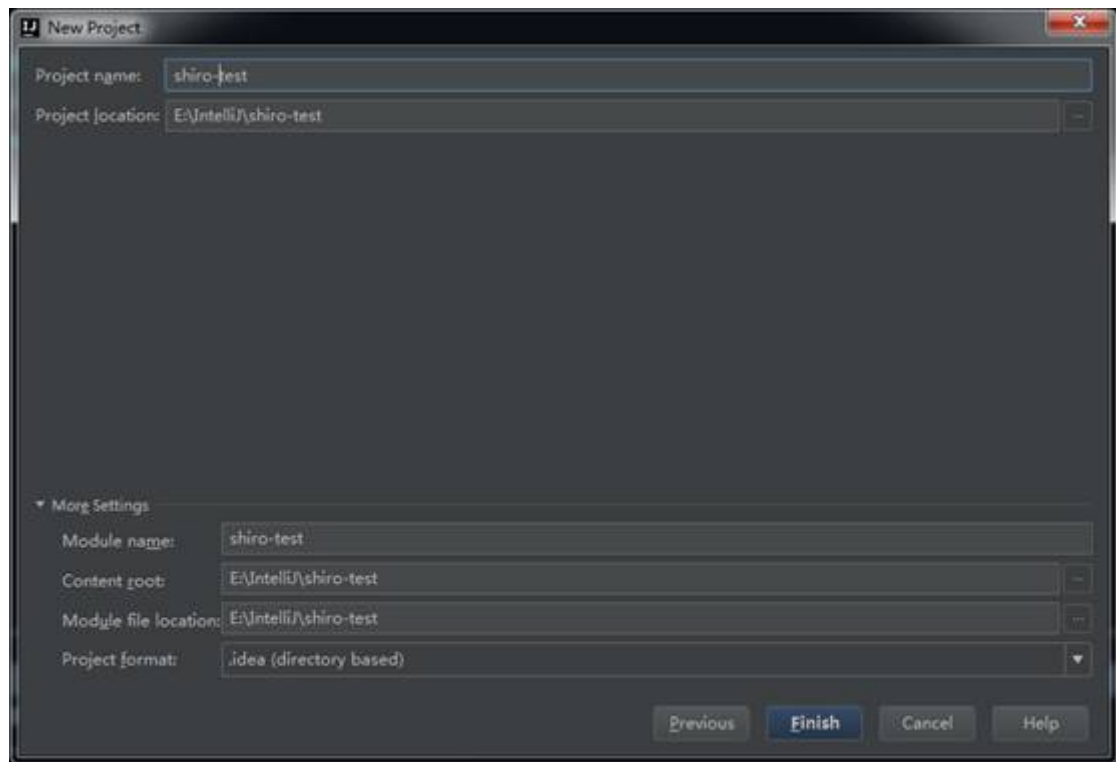
Realm：Realm是用户的信息认证器和用户的权限人证器，我们需要自己来实现Realm来自定义的管理我们自己系统内部的权限规则。SecurityManager要验证用户，需要从Realm中获取用户。可以把Realm看做是数据源。

1.1 搭建测试工程

新建一个普通的java工程，简单了解一下Shiro的API。

完整的工程目录如下：





1.2 导入jar包

在pom.xml中加入jar包的配置：

```
org.apache.shiro shiro-core 1.2.3   org.slf4j slf4j-log4j12 1.6.1   org.slf4j  
slf4j-api 1.6.1
```

使用log4j查看日志，需要在resources目录下创建log4j.properties配置文件：

```
log4j.rootLogger=INFO, stdout,  
  
log4j.appender.stdout=org.apache.log4j.ConsoleAppender  
  
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout  
  
log4j.appender.stdout.layout.ConversionPattern=%d %p [%c] -  
%m%n
```

1.3 编写shiro配置文件

在resources目录下编写shiro配置文件，shiro.ini：

```
#用户名=密码,角色1,角色2..., 角色n
```

```
[users]
```

```
root = secret, admin
```

```
guest = guest, guest
```

```
test = 123456, role1, role2
```

```
# -----  
-----
```

```
# Roles with assigned permissions
```

```
# roleName = perm1, perm2, ..., permN
```

```
# 角色名=权限1, 权限2...权限n
```

```
# -----  
-----
```

```
[roles]
```

```
admin = *
```

```
guest = guest
```

```
role1=perm1,perm2
```

```
role2=perm3
```

配置文件中包含两个部分用户[users]和角色[roles]

用户配置的格式是：

用户名=密码,角色1,角色2,...角色n

如：

test=123456,role1,role2

用户名是test，密码是123456，拥有两个角色role1和role2

一个用户可以具有多个角色。注意逗号是英文的。

角色配置的格式是：

角色名=权限1,权限2...权限n

如：

role1=perm1,perm2

角色名是role1，拥有perm1和perm2两个权限。

1.4 测试代码

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

import org.apache.shiro.authc.*;

import org.apache.shiro.config.IniSecurityManagerFactory;

import org.apache.shiro.mgt.SecurityManager;

import org.apache.shiro.session.Session;

import org.apache.shiro.subject.Subject;

import org.apache.shiro.util.Factory;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class ShiroTest {

    private static final transient Logger log =

        LoggerFactory.getLogger(ShiroTest.class);
```

```
public static void main(String[] args) {

    //1. 这里的SecurityManager是
    org.apache.shiro.mgt.SecurityManager

    // 而不是java.lang.SecurityManager

    // 加载配置文件

    Factory<SecurityManager> factory =

        new
        IniSecurityManagerFactory("classpath:shiro.ini");

    //2.解析配置文件，并且返回一些SecurityManger实例

    SecurityManager securityManager =
    factory.getInstance();

    //3.将SecurityManager绑定给SecurityUtils

    SecurityUtils.setSecurityManager(securityManager);

    // 安全操作，Subject是当前登录的用户

    Subject currentUser = SecurityUtils.getSubject();

    // 测试在应用的当前会话中设置属性

    Session session = currentUser.getSession();

    //放进去一个key和一个value

    session.setAttribute("someKey", "aValue");

    //根据key拿到value
```

```
String value = (String)
session.getAttribute("someKey");

if ("aValue".equals(value)) { //比较拿到的值和原来的值是否
一致

    log.info("检索到正确的值[" + value + "]);

}

//尝试进行登录用户，如果登录失败了，我们进行一些处理

if (!currentUser.isAuthenticated()) { //如果用户没有登录
过

    //new UsernamePasswordToken(用户名,密码)

    UsernamePasswordToken token =

        new UsernamePasswordToken("test",
"123456");

    token.setRememberMe(true); //是否记住用户

    try {

        currentUser.login(token);

        //当我们获登录用户之后

        log.info("用户 [" + currentUser.getPrincipal()
+ "] 登陆成功");

        // 查看用户是否有指定的角色

        if (currentUser.hasRole("admin")) {

            log.info("您有admin角色");

        } else {
```

```
        log.info("您没有admin角色");

    }

    if (currentUser.hasRole("role1")) {

        log.info("您有role1角色");

    } else {

        log.info("您没有role1角色");

    }


    // 查看用户是否有某个权限

    if (currentUser.isPermitted("perm1")) {

        log.info("您有perm1权限");

    } else {

        log.info("您没有perm1权限");

    }

    if (currentUser.isPermitted("guest")) {

        log.info("您有guest权限");

    } else {

        log.info("您没有guest权限");

    }


    //登出
```



```
        currentUser.logout();

    } catch (UnknownAccountException uae) {

        log.info(token.getPrincipal() + "账户不存在");

    } catch (IncorrectCredentialsException ice) {

        log.info(token.getPrincipal() + "密码不正确");

    } catch (LockedAccountException lae) {

        log.info(token.getPrincipal() + "用户被锁定了");

    } catch (AuthenticationException ae) {

        //无法判断是什么错了

        log.info(ae.getMessage());

    }

}

}

}
```

运行结果：

```
2017-10-06 13:41:56,154 INFO [ShiroTest] - 检索到正确的值[aValue]
2017-10-06 13:41:56,156 INFO [ShiroTest] - 用户 [test] 登陆成功
2017-10-06 13:41:56,156 INFO [ShiroTest] - 您没有admin角色
2017-10-06 13:41:56,156 INFO [ShiroTest] - 您有role1角色
2017-10-06 13:41:56,157 INFO [ShiroTest] - 您有perm1权限
2017-10-06 13:41:56,157 INFO [ShiroTest] - 您没有guest权限
```

注：rememberMe只能记住你登录过，但不会记住你是谁以及你的权限信息。

2. Shiro+MySQL动态权限验证

2.1 数据库设计

在实际开发中，用户名密码、角色、权限需要存在数据库中动态管理。一个简单的Shiro+MySQL的项目需要三张表：

用户表SHIRO_USER：

名	类型	长度	小数点	允许空值	
ID	int	11	0	<input type="checkbox"/>	1
PASSWORD	varchar	50	0	<input type="checkbox"/>	
USER_NAME	varchar	100	0	<input type="checkbox"/>	

测试数据：

ID	PASSWORD	USER_NAME
1	admin	admin@shiro.com
2	123456	test@shiro.com

用户角色表SHIRO_USER_ROLE：

名	类型	长度	小数点	允许空值	
ROLE_NAME	varchar	100	0	<input type="checkbox"/>	1
USER_NAME	varchar	100	0	<input type="checkbox"/>	2

测试数据：

ROLE_NAME	USER_NAME
admin	admin@shiro.com
guest	test@shiro.com
test	test@shiro.com

角色权限表SHIRO_ROLE_PERMISSION：

名	类型	长度	小数点	允许空值	
ROLE_NAME	varchar	100	0	<input type="checkbox"/>	1
PERM_NAME	varchar	100	0	<input type="checkbox"/>	2

测试数据：

ROLE_NAME	PERM_NAME
admin	*
test	perm1
test	perm2

2.2 工程搭建

工程的搭建与第1.1节中介绍的一样。需要在1.2节的基础上加上数据源的jar包和mysql的驱动包，数据源我们选用spring-jdbc：

```
<dependency>

    <groupId>mysql</groupId>

    <artifactId>mysql-connector-java</artifactId>

    <version>5.1.32</version>

</dependency>

<dependency>

    <groupId>org.springframework</groupId>

    <artifactId>spring-jdbc</artifactId>

    <version>4.3.11.RELEASE</version>

</dependency>
```

2.3 配置文件

配置文件与1.3中的不同，在使用jdbc数据源的时候，不需要指定user和roles，而是在配置文件中指定数据库连接信息和要执行的sql语句。

在resources文件夹下创建配置文件shiro-mysql.ini：

```
[main]

dataSource=org.springframework.jdbc.datasource.DriverManagerData
taSource

dataSource.driverClassName=com.mysql.jdbc.Driver
```

```
dataSource.url=jdbc:mysql://127.0.0.1:3306/数据库名

dataSource.username=用户名

#如果数据库没有密码，就不要写这行

dataSource.password=你的密码

jdbcRealm=org.apache.shiro.realm.jdbc.JdbcRealm

#是否检查权限

jdbcRealm.permissionsLookupEnabled = true

jdbcRealm.dataSource=$dataSource

#重写sql语句

#根据用户名查询出密码

jdbcRealm.authenticationQuery = select PASSWORD from
SHIRO_USER where USER_NAME = ?

#根据用户名查询出角色

jdbcRealm.userRolesQuery = select ROLE_NAME from
SHIRO_USER_ROLE where USER_NAME = ?

#根据角色名查询出权限

jdbcRealm.permissionsQuery = select PERM_NAME from
SHIRO_ROLE_PERMISSION WHERE ROLE_NAME = ?

securityManager.realms=$jdbcRealm
```

注意sql语句，每次只查询一个shiro要求查询的字段，如果写select *就会报错了。

ini配置文件要求必须是key=value的形式，如果有些人没有设置数据库的密码，就不要写对应的配置。只写”dataSource.password=”等号右面没有值会报错。

2.4 测试代码

测试代码与1.4中类似，只是读取的配置文件不同：

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

import org.apache.shiro.authc.*;

import org.apache.shiro.config.IniSecurityManagerFactory;

import org.apache.shiro.mgt.SecurityManager;

import org.apache.shiro.subject.Subject;

import org.apache.shiro.util.Factory;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class ShiroMySQLTest {

    private static final transient Logger log =

        LoggerFactory.getLogger(ShiroMySQLTest.class);

    public static void main(String[] args) {

        Factory<SecurityManager> factory =

            new

            IniSecurityManagerFactory("classpath:shiro-mysql.ini");

        SecurityManager securityManager =

            factory.getInstance();

        SecurityUtils.setSecurityManager(securityManager);

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

```
UsernamePasswordToken token =

    new UsernamePasswordToken("test@shiro.com",
"123456");

token.setRememberMe(true); //是否记住用户

try {

    currentUser.login(token);

    //当我们获登录用户之后

    log.info("用户 [" + currentUser.getPrincipal() +
"] 登陆成功");

    //查看用户是否有角色

    if (currentUser.hasRole("admin")) {

        log.info("您有admin角色");

    } else {

        log.info("您没有admin角色");

    }

    if (currentUser.hasRole("test")) {

        log.info("您有test角色");

    } else {

        log.info("您没有test角色");

    }

    // 查看用户是否有某个权限
```

```
        if (currentUser.isPermitted("perm1")) {

            log.info("您有perm1权限");

        } else {

            log.info("您没有perm1权限");

        }

        if (currentUser.isPermitted("guest")) {

            log.info("您有guest权限");

        } else {

            log.info("您没有guest权限");

        }

        //登出

        currentUser.logout();

    } catch (UnknownAccountException uae) {

        log.info(token.getPrincipal() + "账户不存在");

    } catch (IncorrectCredentialsException ice) {

        log.info(token.getPrincipal() + "密码不正确");

    } catch (LockedAccountException lae) {

        log.info(token.getPrincipal() + "用户被锁定了 ");

    } catch (AuthenticationException ae) {

        //无法判断是什么错了

        log.info(ae.getMessage());

    }

}
```

```
    }  
  
    }  
  
}
```

运行结果：

```
2017-10-06 17:27:25,153 INFO [ShiroMySqlTest] - 用户 [test@shiro.com] 登陆成功  
2017-10-06 17:27:25,172 INFO [ShiroMySqlTest] - 您没有admin角色  
2017-10-06 17:27:25,189 INFO [ShiroMySqlTest] - 您有test角色  
2017-10-06 17:27:25,208 INFO [ShiroMySqlTest] - 您有perm1权限  
2017-10-06 17:27:25,222 INFO [ShiroMySqlTest] - 您没有guest权限
```

3. Web中使用Shiro

本小节使用SpringJDBC、SpringMVC、Shiro实现一个简单的权限验证。实际开发中数据层可以使用Hibernate、Mybatis等数据层框架。本节的示例只是介绍Web工程中使用Shiro验证的基本流程，完整的权限系统会比这个示例复杂的多。为了简化代码，这个示例中没有写接口。

这个测试项目用第2小节中的数据进行测试。

工程中定义几个地址：

/gologin.html 不需要权限验证就可以访问

/login.html 不需要验证就可以访问

/doadd.html 要有perm1和perm2权限才可以访问，访问成功后页面显示add

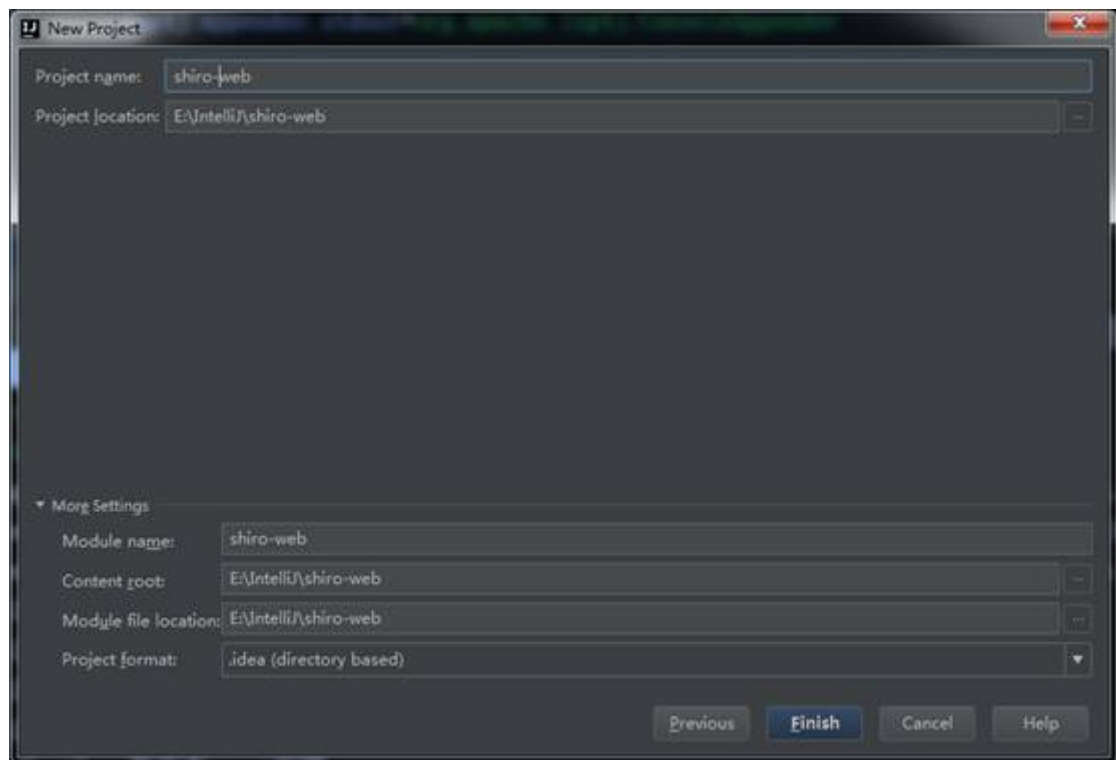
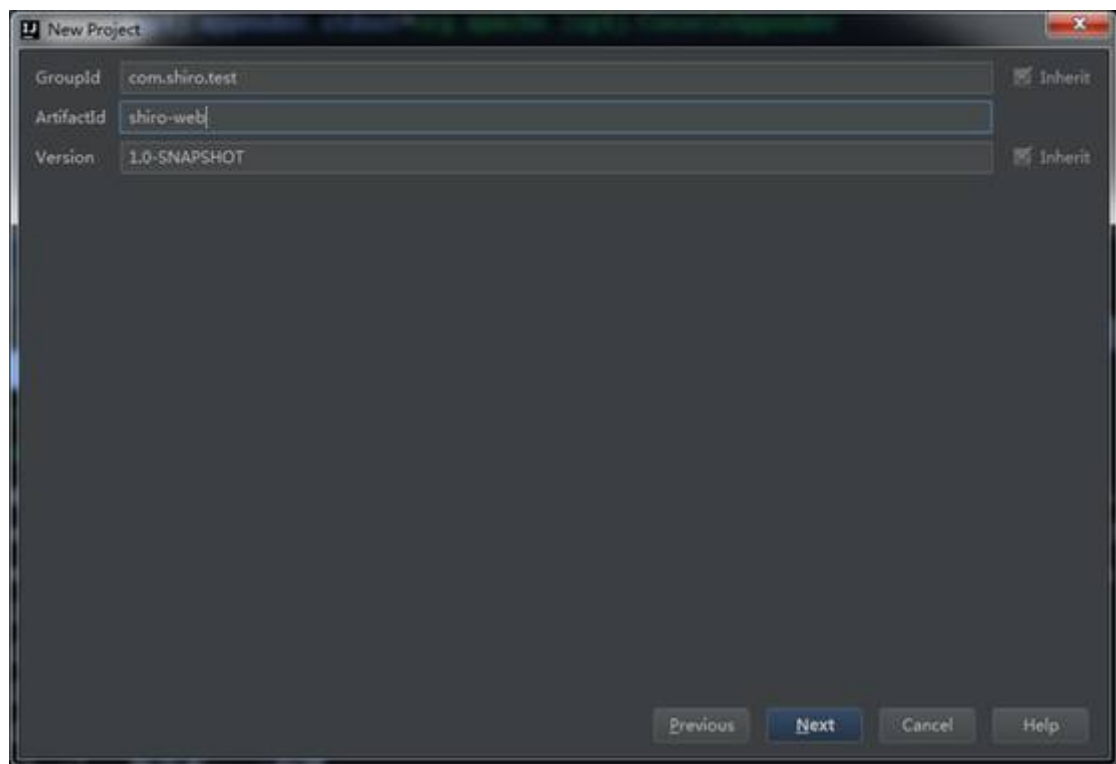
/doget.html 要有admin权限才可以访问，访问成功后页面显示get

/doupdate.html 要有perm1权限才能访问，访问成功后页面显示update

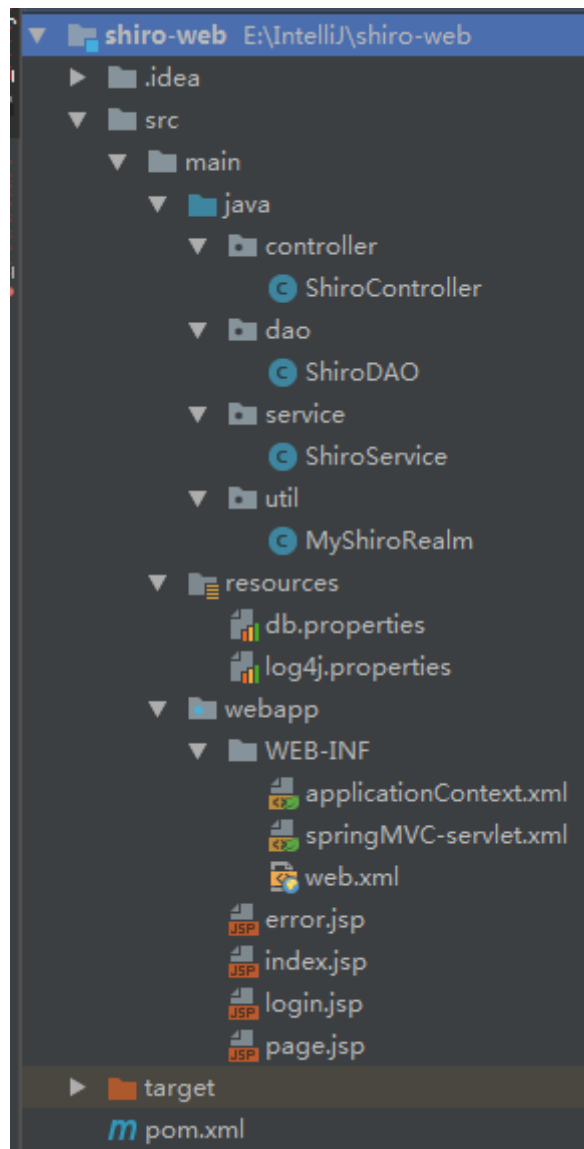
/dodel.html 要有perm2权限才可以访问，访问成功后页面显示del

第2小节的测试数据中，**test@shiro.com**这个用户由perm1和perm2两个权限，没有admin权限，所以/doget.html访问不到,会显示error。

3.1 创建web工程



完整的工程目录如下：



3.2 导入jar包

```
<!--shiro核心类库-->
```

```
<dependency>
```

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

```
    <artifactId>shiro-core</artifactId>
```

```
    <version>1.2.3</version>
```

```
</dependency>
```

```
<!--日志工具包-->
```

```
<dependency>

    <groupId>org.slf4j</groupId>

    <artifactId>slf4j-log4j12</artifactId>

    <version>1.6.1</version>

</dependency>

<dependency>

    <groupId>org.slf4j</groupId>

    <artifactId>slf4j-api</artifactId>

    <version>1.6.1</version>

</dependency>

<!--mysql驱动-->

<dependency>

    <groupId>mysql</groupId>

    <artifactId>mysql-connector-java</artifactId>

    <version>5.1.32</version>

</dependency>

<!--spring相关包-->

<dependency>

    <groupId>org.springframework</groupId>

    <artifactId>spring-web</artifactId>

    <version>4.3.11.RELEASE</version>
```

```
</dependency>

<dependency>

    <groupId>org.springframework</groupId>

    <artifactId>spring-webmvc</artifactId>

    <version>4.3.11.RELEASE</version>

</dependency>

<dependency>

    <groupId>org.springframework</groupId>

    <artifactId>spring-jdbc</artifactId>

    <version>4.3.11.RELEASE</version>

</dependency>

<dependency>

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

    <artifactId>shiro-spring</artifactId>

    <version>1.4.0</version>

</dependency>
```

3.3 DAO层

```
package dao;

import org.springframework.jdbc.core.JdbcTemplate;
```

```
import java.util.List;

public class ShiroDAO {

    private JdbcTemplate jdbcTemplate;

    public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {

        this.jdbcTemplate = jdbcTemplate;

    }

    /**
     * 根据用户名查询密码
     */

    public String getPasswordByUsername(String username) {

        String sql = "select PASSWORD from SHIRO_USER where USER_NAME = ?";

        String password = jdbcTemplate.queryForObject(sql, String.class, username);

        return password;

    }

}
```

```

/**

 * 查询当前用户对应的权限

 */

    public List<String> getPermissionByUserName(String
username) {

        String sql = "select P.PERM_NAME from
SHIRO_ROLE_PERMISSION P inner join SHIRO_USER_ROLE R on
P.ROLE_NAME=R.ROLE_NAME where R.USER_NAME = ?";

        List<String> perms = jdbcTemplate.queryForList(sql,
String.class, username);

        return perms;

    }

}

```

3.4 Service层

```

package service;

import
com.sun.scenario.effect.impl.sw.sse.SSEBlend_SRC_OUTPeer;

import dao.ShiroDAO;

import org.apache.shiro.SecurityUtils;

import org.apache.shiro.authc.*;

import org.apache.shiro.subject.Subject;

```

```
import java.util.List;

public class ShiroService {

    private ShiroDAO shiroDAO;

    public void setShiroDAO(ShiroDAO shiroDAO) {

        this.shiroDAO = shiroDAO;

    }

    /**

    * 登录

    */

    public void doLogin(String username, String password)
throws Exception {

        Subject currentUser = SecurityUtils.getSubject();

        if (!currentUser.isAuthenticated()) {

            UsernamePasswordToken token =

                new UsernamePasswordToken(username,
password);

            token.setRememberMe(true); //是否记住用户

            try {
```

```
        currentUser.login(token);//执行登录

    } catch (UnknownAccountException uae) {

        throw new Exception("账户不存在");

    } catch (IncorrectCredentialsException ice) {

        throw new Exception("密码不正确");

    } catch (LockedAccountException lae) {

        throw new Exception("用户被锁定了 ");

    } catch (AuthenticationException ae) {

        ae.printStackTrace();

        throw new Exception("未知错误");

    }

}

}

/**

 * 根据用户名查询密码

 */

public String getPasswordByUserName(String username) {

    return shiroDAO.getPasswordByUserName(username);

}
```



```

/**

 * 查询用户所有权限

 */

public List<String> getPermissionByUserName(String
username) {

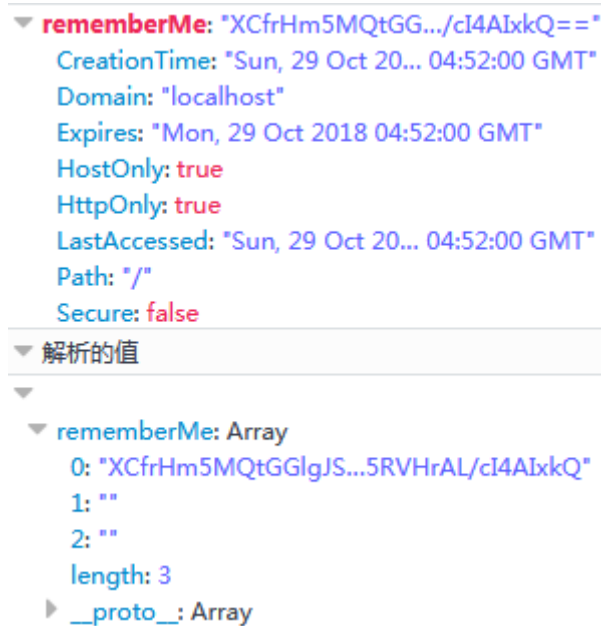
    return shiroDAO.getPermissionByUserName(username);

}

}

```

注：rememberMe后浏览器里会生成一个cookie：



如果访问的路径，要求权限是user，所有使用过rememberMe的用户就都可以访问。但是它只是记录你登录过，不会记住你是谁以及你的权限信息。

3.5 Controller

```
package controller;
```

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

import org.springframework.stereotype.Controller;

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

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

import org.springframework.web.servlet.ModelAndView;

import service.ShiroService;
```

```
@Controller
```

```
public class ShiroController {
```

```
    @Autowired
```

```
    private ShiroService shiroService;
```

```
    @RequestMapping("/gologin.html")
```

```
    public String goLogin() {
```

```
        return "/login.jsp";
```

```
    }
```

```
    @RequestMapping("/login.html")
```

```
    public ModelAndView login(String username, String
password) {
```

```
        try {
```

```

        shiroService.doLogin(username, password);

    } catch (Exception e) {

        return new ModelAndView("/error.jsp", "msg",
e.getMessage());

    }

    return new ModelAndView("/index.jsp");

}

@RequestMapping("/logout.html")

public String logout() {

    Subject currentUser = SecurityUtils.getSubject();

    currentUser.logout();

    return "/login.jsp";

}

```

```

/**

 * 模拟不同的请求，在配置文件中对请求进行权限拦截

 */

@RequestMapping("/do{act}.html")

public ModelAndView get(@PathVariable String act) {

    //简化代码，省略数据库操作

    //通过页面上显示的信息查看请求是否被拦截

    return new ModelAndView("/page.jsp", "page", act);
}

```

3.6 自定义Realm

```
@Override
```

```
protected AuthorizationInfo doGetAuthorizationInfo
```

```
(PrincipalCollection principalCollection) {
```

```
    //根据自己的需求编写获取授权信息，这里简化代码获取了用户对应的  
    所有权限
```

```
        String username =
```

```
            (String)
```

```
principalCollection.fromRealm(getName()).iterator().next();
```

```
        if (username != null) {
```

```
            List<String> perms =
```

```
shiroService.getPermissionByUserName(username);
```

```
            if (perms != null && !perms.isEmpty()) {
```

```
                SimpleAuthorizationInfo info = new
```

```
SimpleAuthorizationInfo();
```

```
                for (String each : perms) {
```

```
                    //将权限资源添加到用户信息中
```

```
                    info.addStringPermission(each);
```

```
                }
```

```
                return info;
```

```
            }
```

```
        }
```

```
        return null;
```

```
}

@Override

protected AuthenticationInfo doGetAuthenticationInfo

    (AuthenticationToken authenticationToken) throws
AuthenticationException {

    UsernamePasswordToken token = (UsernamePasswordToken)
authenticationToken;

    // 通过表单接收的用户名，调用currentUser.login的时候执行

    String username = token.getUsername();

    if (username != null && !"".equals(username)) {

        //查询密码

        String password =
shiroService.getPasswordByUserName(username);

        if (password != null) {

            return new SimpleAuthenticationInfo(username,
password, getName());

        }

    }

    return null;

}

}
```

3.7 配置文件

resources目录下的db.properties用于存放数据库配置信息：

```
jdbc.driver=com.mysql.jdbc.Driver

jdbc.url=jdbc:mysql://localhost:3306/数据库?
characterEncoding=utf-8

jdbc.username=用户名

jdbc.password=密码
```

log4j.properties

```
log4j.rootLogger=INFO, stdout,

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern=%d %p [%c] -
%m%n
```

WEB-INF目录下的applicationContext.xml：

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

<beans xmlns="http://www.springframework.org/schema/beans"

        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

        xmlns:context="http://www.springframework.org/schema/context"

        xsi:schemaLocation="

            http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans-
            4.2.xsd
```

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

```
<!--读取配置文件-->
```

```
<context:property-placeholder
location="classpath:db.properties" ignore-
unresolvable="true"/>
```

```
<!--从配置文件中获取数据源-->
```

```
<bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSo
urce">
```

```
<property name="driverClassName"
value="${jdbc.driver}"/>
```

```
<property name="url" value="${jdbc.url}"/>
```

```
<property name="username" value="${jdbc.username}"/>
```

```
<property name="password" value="${jdbc.password}"/>
```

```
</bean>
```

```
<!-- 配置Jdbc模板 -->
```

```
<bean id="jdbcTemplate"
class="org.springframework.jdbc.core.JdbcTemplate">
```

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

```
</bean>
```



```
<bean id="shiroDAO" class="dao.ShiroDAO">

    <property name="jdbcTemplate" ref="jdbcTemplate"/>

</bean>

<bean id="shiroService" class="service.ShiroService">

    <property name="shiroDAO" ref="shiroDAO"/>

</bean>

<bean id="myShiroRealm" class="util.MyShiroRealm">

    <property name="shiroService" ref="shiroService"/>

</bean>


<bean id="securityManager"
class="org.apache.shiro.web.mgt.DefaultWebSecurityManager">

    <property name="realm" ref="myShiroRealm"/>

</bean>

<bean id="shiroFilter"
class="org.apache.shiro.spring.web.ShiroFilterFactoryBean">

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

    <!--去登录的地址-->

    <property name="loginUrl" value="/gologin.html"/>

    <!--登录成功的跳转地址-->

    <property name="successUrl" value="/index.html"/>
```

```

        <!--验证失败的跳转地址-->

        <property name="unauthorizedUrl" value="/error.jsp"/>

        <!--定义过滤的规则-->

        <!--复杂的系统中，url和权限都可以从数据库中读取-->

        <!--anon是不需要验证，authc时需要验证，perms[admin]代表要
        admin权限-->

        <property name="filterChainDefinitions">

            <value>

                /gologin.html = anon

                /login.html = anon

                /doadd.html = authc, perms[perm1,perm2]

                /doget.html = authc, perms[admin]

                /doupdate.html = authc, perms[perm1]

                /dodel.html = authc, perms[perm2]

            </value>

        </property>

    </bean>

</beans>

```

shiro过滤器过滤属性含义： ****

securityManager：这个属性是必须的。

loginUrl：没有登录的用户请求需要登录的页面时自动跳转到登录页面，不是必须的属性，不输入地址的话会自动寻找项目web项目的根目录下的”/login.jsp”页面。

successUrl：登录成功默认跳转页面，不配置则跳转至"/"。如果登陆前点击的一个需要登录的页面，则在登录自动跳转到那个需要登录的页面。不跳转到此。

unauthorizedUrl：没有权限默认跳转的页面

其权限过滤器及配置释义：

anon:

例子/admins/**=anon 没有参数，表示可以匿名使用。

authc:

例如/admins/user/**=authc表示需要认证(登录)才能使用，没有参数

roles(角色): ****

例子/admins/user/=roles[admin],参数可以写多个，参数之间用逗号分割，**当有多个参数时，例如admins/user/=roles["admin,guest"],每个参数通过才算通过，相当于hasAllRoles()方法。**

perms (权限) **:**

例子/admins/user/=perms[add],参数可以写多个，**例如/admins/user/=perms["add, modify"]，当有多个参数时必须每个参数都通过才通过，想当于isPermittedAll()方法。**

rest: ****

例子/admins/user/=rest[user],根据请求的方法，**相当于/admins/user/=perms[user:method] ,其中method为post, get, delete等。**

port: ****

例子/admins/user/**=port[8081],当请求的url的端口不是8081是跳转到schmal://serverName:8081?queryString,其中schmal是协议http或https等，serverName是你访问的host,8081是url配置里port的端口， queryString是你访问的url里的? 后面的参数。

authcBasic: ****

例如/admins/user/**=authcBasic没有参数.表示httpBasic认证

ssl:

例子/admins/user/**=ssl没有参数，表示安全的url请求，协议为https

user:

例如/admins/user/**=user没有参数表示必须存在用户，当登入操作时不做检查

springMVC-servlet.xml:

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

<beans xmlns="http://www.springframework.org/schema/beans"

        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

        xmlns:context="http://www.springframework.org/schema/context"

        xsi:schemaLocation="

            http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans-4.2.xsd

            http://www.springframework.org/schema/context
            http://www.springframework.org/schema/context/spring-context-4.2.xsd">

    <context:annotation-config />

    <!-- 启动自动扫描 -->

    <context:component-scan base-package="controller">

        <!-- 制定扫包规则，只扫描使用@Controller注解的JAVA类 -->

        <context:include-filter type="annotation"
            expression="org.springframework.stereotype.Controller"/>

    </context:component-scan>
```

```
</beans>
```

web.xml:

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

<web-app xmlns="http://java.sun.com/xml/ns/javaee"

    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee

        http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"

    version="3.0">

    <display-name>Archetype Created Web Application</display-
name>

    <!--配置listener，在启动Web容器的时候加载Spring的配置-->

    <listener>

        <listener-
class>org.springframework.web.context.ContextLoaderListener</l
istener-class>

    </listener>

    <!--欢迎页面-->

    <welcome-file-list>

        <welcome-file>gologin.html</welcome-file>

    </welcome-file-list>

    <!--配置DispatcherServlet-->
```

```
<servlet>

    <servlet-name>springMVC</servlet-name>

    <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servl
et-class>

</servlet>

<servlet-mapping>

    <servlet-name>springMVC</servlet-name>

    <url-pattern>*.html</url-pattern>

</servlet-mapping>

<!-- 配置shiro的核心拦截器 -->

<filter>

    <filter-name>shiroFilter</filter-name>

    <filter-
class>org.springframework.web.filter.DelegatingFilterProxy</fi
lter-class>

</filter>

<filter-mapping>

    <filter-name>shiroFilter</filter-name>

    <url-pattern>*.html</url-pattern>

</filter-mapping>

</web-app>
```

3.8 页面

页面没有添加样式，将这些代码写在body中即可。

login.jsp

```
<form action="/login.html" method="post">

    username:<input type="text" name="username" /><br />

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

    <input type="submit" value="login" />

</form>
```

index.jsp

```
<a href="/doadd.html" target="_blank">add</a><br />

<a href="/dodel.html" target="_blank">del</a><br />

<a href="/doupdate.html" target="_blank">update</a><br />

<a href="/doget.html" target="_blank">get</a><br />
```

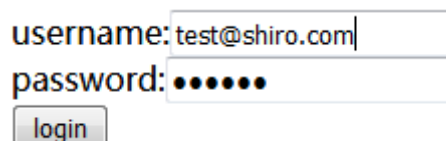
page.jsp

```
<h2>${page}</h2>
```

error.jsp

```
<h2>ERROR:${msg}</h2>
```

测试结果：

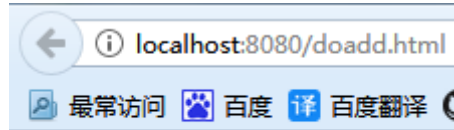


A screenshot of a web browser showing a login form. The form has two input fields: one for 'username' containing 'test@shiro.com' and one for 'password' containing seven dots. Below the fields is a button labeled 'login'.

登录成功：

[add](#)
[del](#)
[update](#)
[get](#)

点击add、del、update可以看到对应的页面：



add

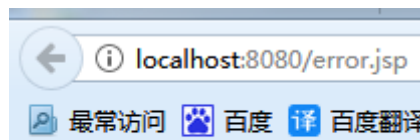


del



update

点击get会报错，并且直接跳转到error.jsp,因为没有权限：



ERROR:

3.9 动态配置过滤规则

在实际开发中，url和对应的访问权限可能需要从数据库中读取，我们可以定义一个工具类从数据库中读取访问权限并传递给Shiro。

```
package util;

import org.apache.shiro.config.Ini;

import org.apache.shiro.util.CollectionUtils;
```



```
import
org.apache.shiro.web.config.IniFilterChainResolverFactory;

import org.springframework.beans.factory.FactoryBean;


import java.text.MessageFormat;

import java.util.HashMap;

import java.util.LinkedHashSet;

import java.util.Map;

import java.util.Set;


public class MyChainDefinitions implements
FactoryBean<Ini.Section> {

    public static final String PREMISSION_STRING =
"perms[{0}]";

    public static final String ROLE_STRING = "roles[{0}]";

    private String filterChainDefinitions;


    public void setFilterChainDefinitions(String
filterChainDefinitions) {

        this.filterChainDefinitions = filterChainDefinitions;

    }
}
```

@Override

```
public Ini.Section getObject() throws Exception {

    //urls可以从数据库查出来，此处省略代码，直接模拟几条数据

    Set<String> urls = new LinkedHashSet<>();

    urls.add("/dotest1.html");

    urls.add("/dotest2.html");

    //每个url对应的权限也可以从数据库中查出来，这里模拟几条数据

    Map<String, String[]> permsMap = new HashMap<>();

    permsMap.put("/dotest1.html", new String[]{"perm1",
"admin"});

    permsMap.put("/dotest2.html", new String[]{"perm1"});


    //加载配置默认过滤链

    Ini ini = new Ini();

    ini.load(filterChainDefinitions);

    Ini.Section section =
ini.getSection(IniFilterChainResolverFactory.URLS);

    if (CollectionUtils.isEmpty(section)) {

        section =
ini.getSection(Ini.DEFAULT_SECTION_NAME);

    }

    for (String url : urls) {
```

```

        String[] perms = permsMap.get(url);

        StringBuilder permFilters = new StringBuilder();

        for (int i = 0; i < perms.length; i++) {

            permFilters.append(perms[i]).append(",");

        }

        //去掉末尾的逗号

        String str = permFilters.substring(0,
permFilters.length() - 1);

        //生成结果如: /dotest1.html = authc, perms[admin]

        section.put(url,
MessageFormat.format(PREMISSION_STRING, str));

    }

    return section;

}

@Override

public Class<?> getObjectType() {

    return this.getClass();

}

@Override

```

```
public boolean isSingleton() {  
  
    return false;  
  
}  
  
}
```

注意section中是以Map存放的数据，所以放入相同的key，后放的会覆盖先放的数据。

修改spring的配置：

```
<!--声明自定义规则-->  
  
<bean id="myChainDefinitions" class="util.MyChainDefinitions">  
  
    <!--静态的条件-->  
  
    <property name="filterChainDefinitions">  
  
        <value>  
  
            /gologin.html = anon  
  
            /login.html = anon  
  
            /doadd.html = authc, perms[perm1,perm2]  
  
            /doget.html = authc, perms[admin]  
  
            /doupdate.html = authc, perms[perm1]  
  
            /dodel.html = authc, perms[perm2]  
  
            /logout.html=user  
  
        </value>  
  
    </property>  
  
</bean>
```

将shiroFilter中的filterChainDefinitions替换掉：

```
<bean id="shiroFilter"
class="org.apache.shiro.spring.web.ShiroFilterFactoryBean">

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

    <property name="loginUrl" value="/gologin.html"/>

    <property name="successUrl" value="/index.html"/>

    <property name="unauthorizedUrl" value="/error.jsp"/>

    <!--定义过滤的规则-->

    <property name="filterChainDefinitionMap"
ref="myChainDefinitions"/>

</bean>
```

可以访问/dotest1.html和/dotest2.html查看拦截效果。

3.10 重写过滤器

/doadd.html = authc, perms[perm1,perm2]

shiro默认的拦截是要满足所有的条件，但有时我们只要满足其中的一个，用于拥有perm1或perm2任何一个条件就可以访问/doadd.html。这时我们可以重写过滤器，将and变成or

```
package util;

import javax.servlet.ServletException;

import javax.servlet.ServletResponse;
```

```
import org.apache.shiro.subject.Subject;

import org.apache.shiro.web.filter.authz.AuthorizationFilter;


public class MyShiroPermFilter extends AuthorizationFilter {

    @Override

    protected boolean isAccessAllowed

        (ServletRequest req, ServletResponse resp, Object
mappedValue)

        throws Exception {

        Subject subject = getSubject(req, resp);

        String[] permsArray = (String[]) mappedValue;


        if (permsArray == null || permsArray.length == 0) { //
没有权限限制

            return true;

        }

        for (int i = 0; i < permsArray.length; i++) {

            //如果是角色，就是subject.hasRole()

            //若当前用户是permsArray中的任何一个，则有权限访问

            if (subject.isPermitted(permsArray[i])) {

                return true;

            }

        }

    }

}
```

```
    }

    return false;

}

}
```

此处需要引入servlet的jar包：

```
<dependency>

    <groupId>javax.servlet</groupId>

    <artifactId>javax.servlet-api</artifactId>

    <version>3.1.0</version>

</dependency>
```

在spring中的配置，在id="shiroFilter"的bean中加入过滤拦截：

```
<bean id="shiroFilter"
class="org.apache.shiro.spring.web.ShiroFilterFactoryBean">

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

    <property name="loginUrl" value="/gologin.html"/>

    <property name="successUrl" value="/index.html"/>

    <property name="unauthorizedUrl" value="/error.jsp"/>

    <property name="filterChainDefinitionMap"
ref="myChainDefinitions"/>

    <!--修改后的过滤规则，从and变成or-->

    <property name="filters">

        <map>
```

```

        <entry key="perms">

            <bean class="util.MyShiroPermFilter"/>

        </entry>

    </map>

</property>

</bean>

```

3.11 rememberMe属性

rememberMe可以在浏览器中设置cookie，在spring配置中可以设置cookie的属性，如过期时间、cookie名字、加密的密钥等：

```

<bean id="rememberMeCookie"
class="org.apache.shiro.web.servlet.SimpleCookie">

    <constructor-arg value="rememberMeShiro"/><!-- 浏览器中
cookie的名字 -->

    <property name="httpOnly" value="true"/><!--document对象中
就看不到cookie了-->

    <property name="maxAge" value="2592000"/><!-- 30天 -->

</bean>

<!-- rememberMe管理器 -->

<bean id="rememberMeManager"
class="org.apache.shiro.web.mgt.CookieRememberMeManager">

    <!--密钥要16位，24位或32位的Base64。这个解密后是
1234567890abcdef-->

```



```

        <property name="cipherKey" value="#
        {T(org.apache.shiro.codec.Base64).decode('MTIzNDU2Nzg5MGFiY2Rl
        Zg==')}" />

        <property name="cookie" ref="rememberMeCookie" />

    </bean>

```

在securityManager中加入rememberMe中加入配置：

```

<bean id="securityManager"
class="org.apache.shiro.web.mgt.DefaultWebSecurityManager">

    <property name="realm" ref="myShiroRealm" />

    <!--加入rememberMe的设置-->

    <property name="rememberMeManager"
ref="rememberMeManager" />

</bean>

```



HttpOnly属性：

浏览器中通过document.cookie可以获取cookie属性，设置了HttpOnly=true，在脚本中就不能的到cookie了。可以避免cookie被盗用。