实验一，基于安全域的认证

1，创建SecurityDemo工程

2，配置web.xml文件

```
    <?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
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_2_5.xsd">

        <!-- 只有admin角色才能访问应用中的*.jsp资源 -->
        <security-constraint>
                <web-resource-collection>

<web-resource-name>Protected Area</web-resource-name>

<url-pattern>*.jsp</url-pattern>
                <auth-constraint>

<role-name>admin</role-name>
                </auth-constraint>
        </security-constraint>


        <!-- 指明当Web客户访问受保护的资源时，系统弹出的
登录对话框的类型 -->
```

```
        <login-config>
                <auth-method>BASIC</auth-method>
                <realm-name>Security</realm-name>

        </login-config>
</web-app>
```

3，配置tomcat/conf/tomcat-users.xml文件

```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
  <role rolename="admin"/>
  <user username="zhao" password="jun" roles="admin"/>
</tomcat-users>
```

4，在WebRoot下创建security.jsp文件

```
<%@ page language="java" import="java.util.*"
pageEncoding="GBK"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">
<html>
        <head>
                <title>My JSP 'security.jsp' starting
page</title>
        </head>

        <body>
                这是受保护的页面
        </body>
</html>
```

5，部署并访问应用

http://localhost:8080/SecurityDemo/security.jsp
出现系统对话框

出错---
HTTP Status 401 -

输入---
用户名：zhao
密码：jun
进入安全页面

补充：
在web.xml文件中部份标记的作用：
<web-resource-collection>：声明受保护的web资源
<web-resource-name>：标识受保护的web资源
<url-pattern>：指定受保护的URL路径
<http-method>：指定受保护HTTP方法，如GET，POST，如果没有定义HTTP方法，
那么所有HTTP方法都受保护。如果某种HTTP方法受到保护，则表
示当客户通过这种方法访问受保护资源时，要求通过安全验证
<auth-constrain>：声明可以访问受保护资源的角色，可以包含多个<role-name>子元素
<role-name>：指定可以访问受保护资源的角色，和服务器配置相同，如
tomcat/conf/tomcat-users.xml中的角色。

如果web应用采用BASIC验证，当客户访问受保护的资源时，浏览器会先弹出一个对话框，要求
用户输入用户名和密码。如果客户输入的用户名和密码正确，web服务器就允许他访问这些资源，

否则，在连接3次失败后，会显示一个错误消息页面。这个方法缺点是将用户名和密码从客户端传到web服务器时，在网络上传送的数据采用明文编码，全是可读文本，非常不安全。

如果web应用采用DIGEST验证，效果与前者相同，好处在于，先将用户名和密码加密码后再传送到web服务器，显然更为安全。

如果web应用采用FORM验证，用户必须自定义用于验证的表单和出错页面，
表单中用户名必须命名为j_username，密码必须命名为j_password。提交表单的action
必须为j_security_check。
并在web.xml中配置如下信息：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
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_2_5.xsd">

<!-- 只有admin角色才能访问应用中的*.jsp资源 -->
<security-constraint>
<web-resource-collection>

<web-resource-name>Protected Area</web-resource-name>

<url-pattern>*.jsp</url-pattern>
</web-resource-collection>
<auth-constraint>
```

```
<role-name>admin</role-name>
                        </auth-constraint>
        </security-constraint>


        <!-- 指明当Web客户访问受保护的资源时，系统弹出的登录对话框的类型 -->
        <login-config>
                        <auth-method>FORM</auth-method>
                        <realm-name>Security</realm-name>
                        <form-login-config>

<form-login-page>/check.jsp</form-login-page>

<form-error-page>/error.jsp</form-error-page>
                        </form-login-config>
        </login-config>
</web-app>
```