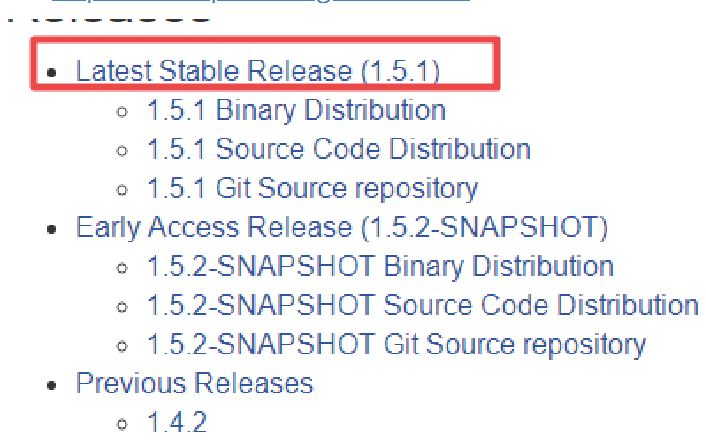
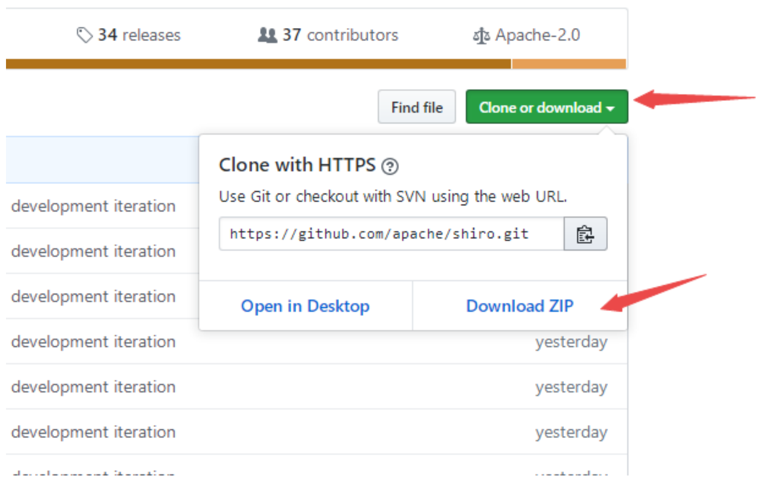
1 简介

- Apache Shiro是一个强大且易用的Java安全（权限）框架,执行身份验证、授权、密码和会话管理。
- Shiro 是 Java 的一个安全框架。目前，使用 Apache Shiro 的人越来越多，因为它相 当简单，对比 Spring
  Security，可能没有 Spring Security 做的功能强大，但是在实际工作时 可能并不需要那么复杂的东西，所以使用小而简单的Shiro 就足够了。
- Shiro可以非常容易的开发出足够好的应用，其不仅可以用在JavaSE环境，也可以用在JavaEE环境。
- Shiro具有认证、授权、加密、会话管理、Web集成和缓存等功能。
- 官网十分钟快速入门教程：http://shiro.apache.org/10-minute-tutorial.html
- 下载地址：http://shiro.apache.org/index.html



- Latest Stable Release (1.5.1)
  - 1.5.1 Binary Distribution
  - 1.5.1 Source Code Distribution
  - 1.5.1 Git Source repository
- Early Access Release (1.5.2-SNAPSHOT)
  - 1.5.2-SNAPSHOT Binary Distribution
  - 1.5.2-SNAPSHOT Source Code Distribution
  - 1.5.2-SNAPSHOT Git Source repository
- Previous Releases
  - 1.4.2

  - 也可以到GitHub下载：https://github.com/apache/shiro



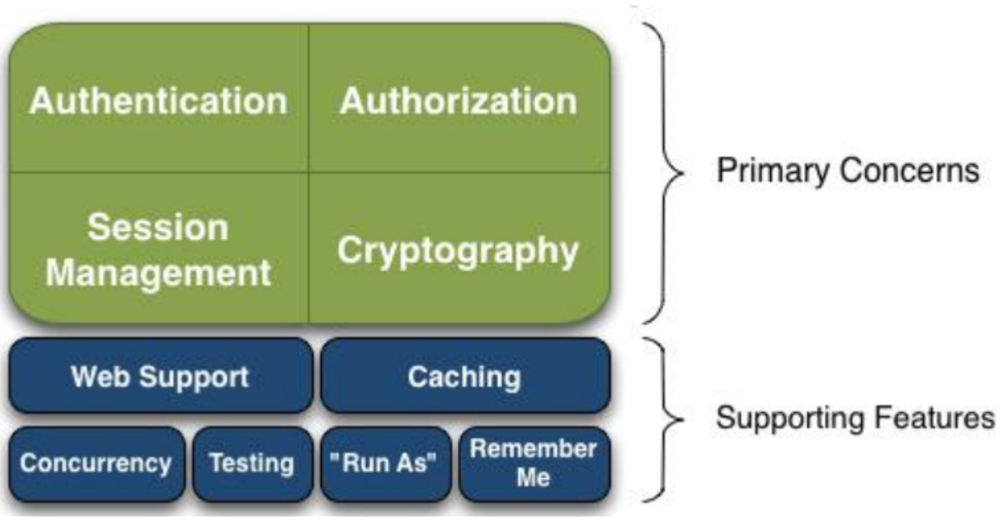
34 releases　37 contributors　Apache-2.0

Find file　Clone or download

Clone with HTTPS
Use Git or checkout with SVN using the web URL.
https://github.com/apache/shiro.git
Open in Desktop　Download ZIP

development iteration	
development iteration	
development iteration	
development iteration	yesterday
development iteration	yesterday
development iteration	yesterday

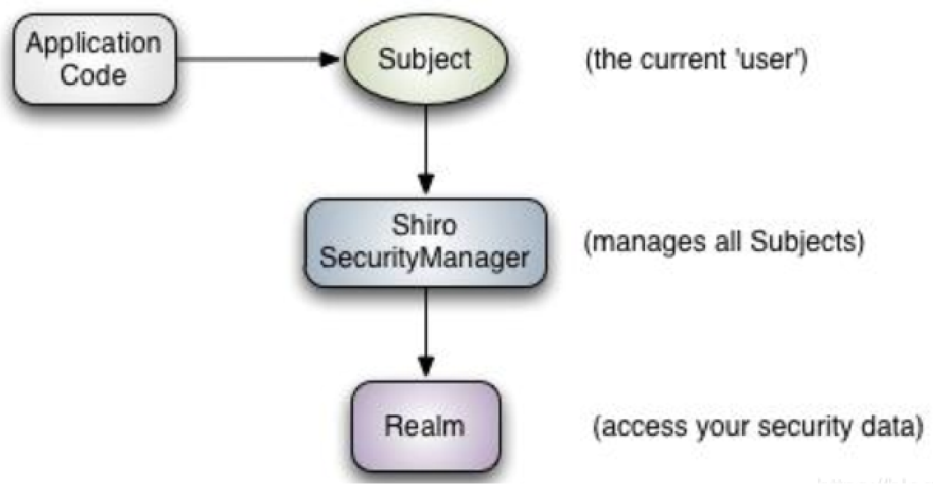
2 功能详述



- **Authentication**：身份认证、登录，验证用户是不是拥有相应的身份；
- **Authorization**：授权，即权限验证，验证某个已认证的用户是否拥有某个权限；即判断用户是否能做事情，常见的如：验证某个用户是否拥有某个角色。或者细粒度的验证某个用户对某个资源是否具有某个权限；
- **Session Manager**：会话管理，即用户登录后就是一次会话，在没有退出之前，它的所有信息都在会话中；会话可以是普通 JavaSE 环境的，也可以是如 Web 环境的；
- **Cryptography**：加密，保护数据的安全性，如密码加密存储到数据库，而不是明文存储；
- **Web Support**：Web 支持，可以非常容易的集成到 Web 环境；
- **Caching**：缓存，比如用户登录后，其用户信息、拥有的角色/权限不必每次去查，这样可以提高效率；
- **Concurrency**：shiro 支持多线程应用的并发验证，即如在一个线程中开启另一个线程，能把权限自动传播过去；
- **Testing**：提供测试支持；
- **Run As**：允许一个用户假装为另一个用户（如果他们允许）的身份进行访问；
- **Remember Me**：记住我，这个是非常常见的功能，即一次登录后，下次再来的话不用登录记住一点，Shiro 不会去维护用户、维护权限；这些需要我们自己去设计/提供；然后通过相应的接口注入给 Shiro 即可。

3 Shiro架构（外部）

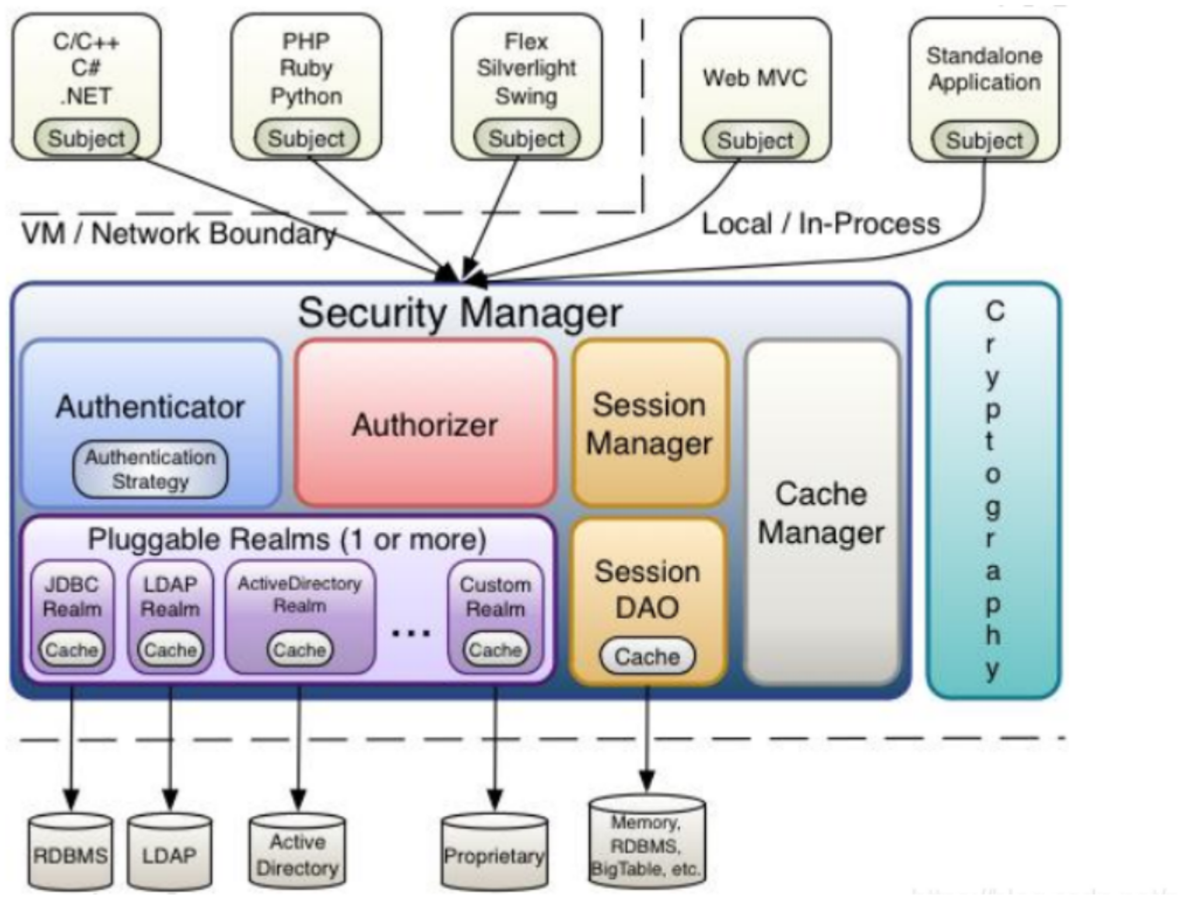
从外部来看Shiro，即从应用程序角度来观察如何使用Shiro完成工作：



对于我们而言，最简单的一个 Shiro 应用：应用代码通过 Subject 来进行认证和授权，而 Subject 又委托给 SecurityManager； 我们需要给 Shiro 的 SecurityManager 注入 Realm，从而让 SecurityManager 能得到合法
的用户及其权限进行判断。

- Subject
    - 主体，可以看到主体可以是任何可以与应用交互的“用户”
    - 是应用代码直接交互的对象，也就是说Shiro的对外API核心就是Subject
    - 代表了当前的用户，这个用户不一定是一个具体的人，与当前应用交互的任何东西都是Subject（如网络爬虫、机器人等）
    - 与Subject的所有交互都会委托给SecurityManager，Subject其实是一个门面，SecurityManager才是实际的执行者
- SecurityManager【核心】
    - 安全管理器，即所有与安全有关的操作都会与SecurityManager交互，并且管理着所有的Subject
    - 相 当 于 SpringMVC 中 的 DispatcherServlet 或 者 Struts2 中的FilterDispatcher，是 Shiro 的心脏
    - 所有具体的交互都通过 SecurityManager 进行控制，负责与Shiro的其他组件进行交互
    - 管理着所有 Subject、且负责进行认证和授权、及会话、缓存的管理
- Realm
    - 可以有 1 个或多个 Realm，可以认为是安全实体数据源，即用于获取安全实体的
    - 可以是 JDBC 实现，也可以是 LDAP 实现，或者内存实现等等
    - Shiro从Realm获取安全数据（如用户、角色、权限），就是说SecurityManager要验证用户身份，那么它需要从Realm获取相应的用户进行比较，来确定用户的身份是否合法
    - 也需要从Realm得到用户响应的角色、权限，进行验证用户的操作是否能够进行，可以把Realm看成DataSource。
    - 注意：Shiro不知道你的用户/权限存储在哪及以何种格式存储，所以我们一般在应用中都需要实现自己的 Realm

4 Shiro架构（外部）



- **Subject**：主体，可以看到主体可以是任何可以与应用交互的“用户”；
- **SecurityManager** ： 相 当 于 SpringMVC 中 的 DispatcherServlet 或 者 Struts2 中的
- **FilterDispatcher**；是 Shiro 的心脏；所有具体的交互都通过 SecurityManager 进行控制；它管理着所有 Subject、且负责进行认证和授权、及会话、缓存的管理。
- **Authenticator**：认证器，负责主体认证的，这是一个扩展点，如果用户觉得 Shiro 默认的不好，可以自定义实现；其需要认证策略（Authentication Strategy），即什么情况下算用户认证通过了；
- **Authrizer**：授权器，或者访问控制器，用来决定主体是否有权限进行相应的操作；即控制着用户能访问应用中的哪些功能；
- **Realm**：可以有 1 个或多个 Realm，可以认为是安全实体数据源，即用于获取安全实体的；可以是 JDBC 实现，也可以是 LDAP 实现，或者内存实现等等；由用户提供；注意：Shiro不知道你的用户/权限存储在哪及以何种格式存储；所以我们一般在应用中都需要实现自己的 Realm；
- **SessionManager**：如果写过 Servlet 就应该知道 Session 的概念，Session 呢需要有人去管理它的生命周期，这个组件就是 SessionManager；而 Shiro 并不仅仅可以用在 Web 环境，也可以用在如普通的 JavaSE 环境、EJB 等环境；所有呢，Shiro 就抽象了一个自己的 Session来管理主体与应用之间交互的数据；这样的话，比如我们在 Web 环境用，刚开始是一台Web 服务器；接着又上了台 EJB 服务器；这时想把两台服务器的会话数据放到一个地方，
这个时候就可以实现自己的分布式会话（如把数据放到 Memcached 服务器）；
- **SessionDAO**：DAO 大家都用过，数据访问对象，用于会话的 CRUD，比如我们想把 Session保存到数据库，那么可以实现自己的 SessionDAO，通过如 JDBC 写到数据库；比如想把Session 放到 Memcached 中，可以实现自己的 Memcached SessionDAO；另外 SessionDAO中可以使用 Cache 进行缓存，以提高性能；
- **CacheManager**：缓存控制器，来管理如用户、角色、权限等的缓存的；因为这些数据基本上很少去改变，放到缓存中后可以提高访问的性能
- **Cryptography**：密码模块，Shiro 提高了一些常见的加密组件用于如密码加密

5 搭建环境

这里使用Maven进行管理。

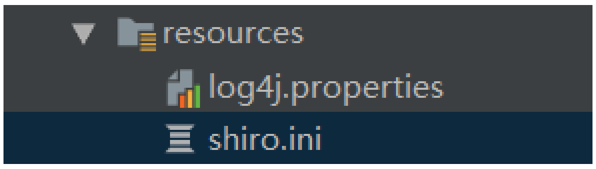
可以基于GitHub的QuickStart：https://github.com/apache/shiro/tree/master/samples/quickstart

和官网快速入门：http://shiro.apache.org/tutorial.html

- 导入依赖：

```xml
<dependencies>
    <!-- https://mvnrepository.com/artifact/org.apache.shiro/shiro-core -->
    <dependency>
        <groupId>org.apache.shiro</groupId>
        <artifactId>shiro-core</artifactId>
        <version>1.5.1</version>
    </dependency>


    <!-- configure logging -->
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>jcl-over-slf4j</artifactId>
        <version>1.7.21</version>
    </dependency>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-log4j12</artifactId>
        <version>1.7.21</version>
    </dependency>
    <dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>1.2.17</version>
    </dependency>
</dependencies>
```

- 因为导入了Log4j，这里配置一下

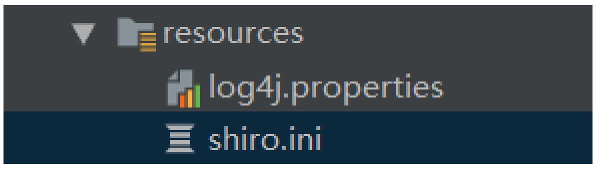


log4j.properties

```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

# General Apache libraries
log4j.logger.org.apache=WARN

# Spring
log4j.logger.org.springframework=WARN

# Default Shiro logging
log4j.logger.org.apache.shiro=INFO

# Disable verbose logging
```

```
log4j.logger.org.apache.shiro.util.ThreadContext=WARN
log4j.logger.org.apache.shiro.cache.ehcache.EhCache=WARN
```

- 配置shiro.ini文件

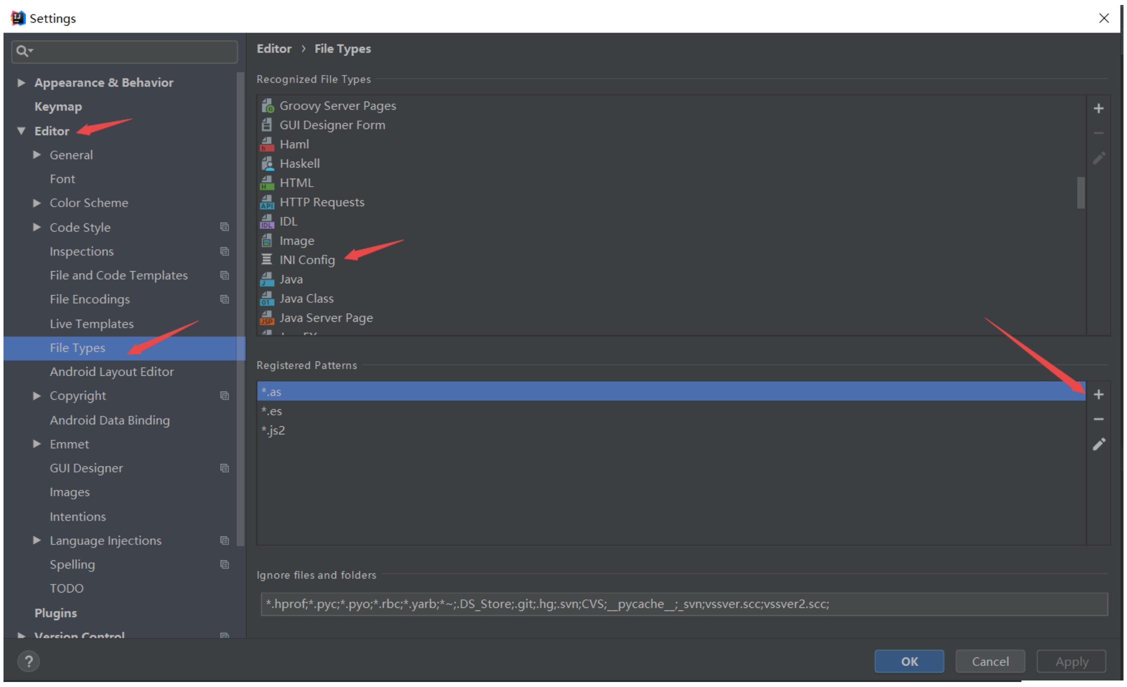


```
[users]
# user 'root' with password 'secret' and the 'admin' role
root = secret, admin
# user 'guest' with the password 'guest' and the 'guest' role
guest = guest, guest
# user 'presidentskroob' with password '12345' ("That's the same combination on
# my luggage!!!" ;)), and role 'president'
presidentskroob = 12345, president
# user 'darkhelmet' with password 'ludicrousspeed' and roles 'darklord' and 'schwartz'
darkhelmet = ludicrousspeed, darklord, schwartz
# user 'lonestarr' with password 'vespa' and roles 'goodguy' and 'schwartz'
lonestarr = vespa, goodguy, schwartz

# -----------------------------------------------------------------------------
# Roles with assigned permissions
#
# Each line conforms to the format defined in the
# org.apache.shiro.realm.text.TextConfigurationRealm#setRoleDefinitions JavaDoc
# -----------------------------------------------------------------------------
[roles]
# 'admin' role has all permissions, indicated by the wildcard '*'
admin = *
# The 'schwartz' role can do anything (*) with any lightsaber:
schwartz = lightsaber:*
# The 'goodguy' role is allowed to 'drive' (action) the winnebago (type) with
# license plate 'eagle5' (instance specific id)
goodguy = winnebago:drive:eagle5
```

这里可能会出现Idae无法识别ini后缀的文件，这里给出解决方法：

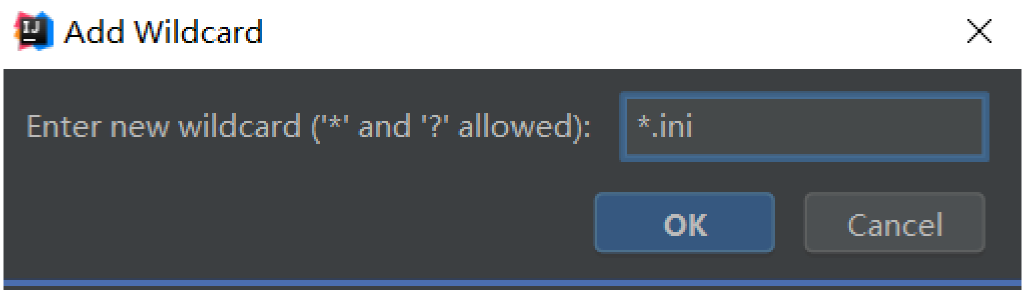
1. 下载插件ini（或者ini4Idea）===》重启Idea

2.



Ignore files and folders: *.hprof;*.pyc;*.pyo;*.rbc;*.yarb;*~;.DS_Store;.git;.hg;.svn;CVS;__pycache__;_svn;vssver.scc;vssver2.scc;

3.

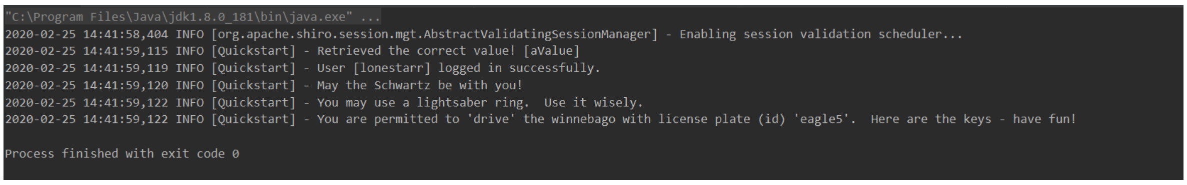


Enter new wildcard ('*' and '?' allowed): *.ini

6 Hello,Shiro【没有用SpringBoot】

```java
/*
 * Licensed to the Apache Software Foundation (ASF) under one
 * or more contributor license agreements.  See the NOTICE file
 * distributed with this work for additional information
 * regarding copyright ownership.  The ASF licenses this file
 * to you under the Apache License, Version 2.0 (the
 * "License"); you may not use this file except in compliance
 * with the License.  You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing,
 * software distributed under the License is distributed on an
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied.  See the License for the
 * specific language governing permissions and limitations
 * under the License.
 */

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;


/**
 * Simple Quickstart application showing how to use Shiro's API.
 *
 * @since 0.9 RC2
 */
public class Quickstart {

    private static final transient Logger log = LoggerFactory.getLogger(Quickstart.class);


    public static void main(String[] args) {

        // The easiest way to create a Shiro SecurityManager with configured
        // realms, users, roles and permissions is to use the simple INI config.
        // We'll do that by using a factory that can ingest a .ini file and
        // return a SecurityManager instance:

        // Use the shiro.ini file at the root of the classpath
        // (file: and url: prefixes load from files and urls respectively):
        Factory<SecurityManager> factory = new IniSecurityManagerFactory("classpath:shiro.ini");
        SecurityManager securityManager = factory.getInstance();

        // for this simple example quickstart, make the SecurityManager
        // accessible as a JVM singleton.  Most applications wouldn't do this
        // and instead rely on their container configuration or web.xml for
        // webapps.  That is outside the scope of this simple quickstart, so
        // we'll just do the bare minimum so you can continue to get a feel
        // for things.
        SecurityUtils.setSecurityManager(securityManager);

        // Now that a simple Shiro environment is set up, let's see what you can do:

        //获取当前的用户对象 Subject
        Subject currentUser = SecurityUtils.getSubject();

        // Do some stuff with a Session (no need for a web or EJB container!!!)
        //通过当前用户拿到Session
        Session session = currentUser.getSession();
        session.setAttribute("someKey", "aValue");
        String value = (String) session.getAttribute("someKey");
        if (value.equals("aValue")) {
            log.info("Retrieved the correct value! [" + value + "]");
        }

        // let's login the current user so we can check against roles and permissions:
        //判断当前的用户是否被认证
        if (!currentUser.isAuthenticated()) {
            //拿到Token：令牌
            UsernamePasswordToken token = new UsernamePasswordToken("lonestarr", "vespa");
            token.setRememberMe(true);//设置记住我
            try {
                currentUser.login(token);//执行登录操作
            } catch (UnknownAccountException uae) {//未知的账户（用户名不存在）
                log.info("There is no user with username of " + token.getPrincipal());
            } catch (IncorrectCredentialsException ice) {//密码错误
                log.info("Password for account " + token.getPrincipal() + " was incorrect!");
            } catch (LockedAccountException lae) {//用户被锁定(多次输入错误密码后）
                log.info("The account for username " + token.getPrincipal() + " is locked.  " +
                        "Please contact your administrator to unlock it.");
            }
            // ... catch more exceptions here (maybe custom ones specific to your application?
            catch (AuthenticationException ae) {
                //unexpected condition?  error?
            }
        }

        //say who they are:
        //print their identifying principal (in this case, a username):
        //获得当前用户的认证信息
        log.info("User [" + currentUser.getPrincipal() + "] logged in successfully.");

        //test a role:
        //当前用户是否拥有某个角色
        if (currentUser.hasRole("schwartz")) {
            log.info("May the Schwartz be with you!");
        } else {
            log.info("Hello, mere mortal.");
        }

        //test a typed permission (not instance-level)
        //用户拥有哪些权限【粗粒度】
        if (currentUser.isPermitted("lightsaber:wield")) {
            log.info("You may use a lightsaber ring.  Use it wisely.");
        } else {
            log.info("Sorry, lightsaber rings are for schwartz masters only.");
        }

        //a (very powerful) Instance Level permission:
        //更有力的判断用户是否拥有哪些权限【细粒度】
        if (currentUser.isPermitted("winnebago:drive:eagle5")) {
            log.info("You are permitted to 'drive' the winnebago with license plate (id) 'eagle5'.  " +
                    "Here are the keys - have fun!");
        } else {
            log.info("Sorry, you aren't allowed to drive the 'eagle5' winnebago!");
        }

        //all done - log out!
        //注销
        currentUser.logout();

        //结束启动
        System.exit(0);
    }
}/*
 * Licensed to the Apache Software Foundation (ASF) under one
 * or more contributor license agreements.  See the NOTICE file
 * distributed with this work for additional information
 * regarding copyright ownership.  The ASF licenses this file
 * to you under the Apache License, Version 2.0 (the
 * "License"); you may not use this file except in compliance
 * with the License.  You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing,
 * software distributed under the License is distributed on an
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied.  See the License for the
 * specific language governing permissions and limitations
 * under the License.
 */

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;


/**
 * Simple Quickstart application showing how to use Shiro's API.
 *
 * @since 0.9 RC2
 */
public class Quickstart {

    private static final transient Logger log = LoggerFactory.getLogger(Quickstart.class);


    public static void main(String[] args) {

        // The easiest way to create a Shiro SecurityManager with configured
        // realms, users, roles and permissions is to use the simple INI config.
        // We'll do that by using a factory that can ingest a .ini file and
        // return a SecurityManager instance:

        // Use the shiro.ini file at the root of the classpath
        // (file: and url: prefixes load from files and urls respectively):
        Factory<SecurityManager> factory = new IniSecurityManagerFactory("classpath:shiro.ini");
        SecurityManager securityManager = factory.getInstance();

        // for this simple example quickstart, make the SecurityManager
        // accessible as a JVM singleton.  Most applications wouldn't do this
        // and instead rely on their container configuration or web.xml for
        // webapps.  That is outside the scope of this simple quickstart, so
        // we'll just do the bare minimum so you can continue to get a feel
        // for things.
        SecurityUtils.setSecurityManager(securityManager);

        // Now that a simple Shiro environment is set up, let's see what you can do:

        // get the currently executing user:
        Subject currentUser = SecurityUtils.getSubject();

        // Do some stuff with a Session (no need for a web or EJB container!!!)
        Session session = currentUser.getSession();
        session.setAttribute("someKey", "aValue");
        String value = (String) session.getAttribute("someKey");
        if (value.equals("aValue")) {
            log.info("Retrieved the correct value! [" + value + "]");
        }

        // let's login the current user so we can check against roles and permissions:
        if (!currentUser.isAuthenticated()) {
            UsernamePasswordToken token = new UsernamePasswordToken("lonestarr", "vespa");
            token.setRememberMe(true);
            try {
                currentUser.login(token);
            } catch (UnknownAccountException uae) {
                log.info("There is no user with username of " + token.getPrincipal());
            } catch (IncorrectCredentialsException ice) {
                log.info("Password for account " + token.getPrincipal() + " was incorrect!");
            } catch (LockedAccountException lae) {
                log.info("The account for username " + token.getPrincipal() + " is locked.  " +
                        "Please contact your administrator to unlock it.");
            }
            // ... catch more exceptions here (maybe custom ones specific to your application?
            catch (AuthenticationException ae) {
                //unexpected condition?  error?
            }
        }

        //say who they are:
        //print their identifying principal (in this case, a username):
        log.info("User [" + currentUser.getPrincipal() + "] logged in successfully.");

        //test a role:
        if (currentUser.hasRole("schwartz")) {
            log.info("May the Schwartz be with you!");
        } else {
            log.info("Hello, mere mortal.");
        }

        //test a typed permission (not instance-level)
        if (currentUser.isPermitted("lightsaber:wield")) {
            log.info("You may use a lightsaber ring.  Use it wisely.");
        } else {
            log.info("Sorry, lightsaber rings are for schwartz masters only.");
        }

        //a (very powerful) Instance Level permission:
        if (currentUser.isPermitted("winnebago:drive:eagle5")) {
            log.info("You are permitted to 'drive' the winnebago with license plate (id) 'eagle5'.  " +
                    "Here are the keys - have fun!");
        } else {
            log.info("Sorry, you aren't allowed to drive the 'eagle5' winnebago!");
        }

        //all done - log out!
        currentUser.logout();

        System.exit(0);
    }
}
```

启动一下：



```
"C:\Program Files\Java\jdk1.8.0_181\bin\java.exe" ...
2020-02-25 14:41:58,404 INFO [org.apache.shiro.session.mgt.AbstractValidatingSessionManager] - Enabling session validation scheduler...
2020-02-25 14:41:59,115 INFO [Quickstart] - Retrieved the correct value! [aValue]
2020-02-25 14:41:59,119 INFO [Quickstart] - User [lonestarr] logged in successfully.
2020-02-25 14:41:59,120 INFO [Quickstart] - May the Schwartz be with you!
2020-02-25 14:41:59,122 INFO [Quickstart] - You may use a lightsaber ring.  Use it wisely.
2020-02-25 14:41:59,122 INFO [Quickstart] - You are permitted to 'drive' the winnebago with license plate (id) 'eagle5'.  Here are the keys - have fun!

Process finished with exit code 0
```

- 获取Subject

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

- 获取Session

```
Session session = currentUser.getSession();
```

- 判断当前用户是否被认证

```
currentUser.isAuthenticated()
```

- 获得当前用户的认证信息

```
currentUser.getPrincipal()
```

- 获得用户是否拥有哪些角色

```
currentUser.hasRole("role")
```

- 获得用户是否有哪些权限

```
currentUser.isPermitted("")
```

注意：参数不同，会有不一样的粒度

- 注销

```
currentUser.logout();
```

7 SpringBoot中集成

7.1 搭建环境

创建一个Module——>SpringBoot——>添加Web支持

导入Thymeleaf依赖：

```
<!--Thymeleaf-->
<dependency>
    <groupId>org.thymeleaf</groupId>
    <artifactId>thymeleaf-spring5</artifactId>
</dependency>
<dependency>
    <groupId>org.thymeleaf.extras</groupId>
    <artifactId>thymeleaf-extras-java8time</artifactId>
</dependency>
```

导入Shiro整合Spring的包：

```
<dependency>
    <groupId>org.apache.shiro</groupId>
    <artifactId>shiro-spring</artifactId>
    <version>1.5.1</version>
</dependency>
```

7.2 实现登录拦截

需求：当用户没有权限的时候【没登录】---》这里通过拦截请求模拟实现，跳转到登录页面

创建Realm：【因为不进行用户认证，这里先不写数据】

```
package com.kuang.config;

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;

/**
 * 自定义的Realm  extends AuthorizingRealm
 */
public class UserRealm extends AuthorizingRealm {
    /**
     * 《授权》
     *
     * @param principalCollection
     * @return
     */
    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principalCollection) {
        System.out.println("执行了=>授权doGetAuthorizationInfo方法");
        return null;
    }

    /**
     * 《认证》
     *
     * @param authenticationToken
     * @return
     * @throws AuthenticationException
     */
    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken authenticationToken) throws AuthenticationException {
        System.out.println("执行了=>认证doGetAuthenticationInfo方法");
        return null;
    }
}
```

配置类：

```java
package com.kuang.config;

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

import java.util.LinkedHashMap;
import java.util.Map;

/**
 * Shiro的配置类
 * <p>
 * 建议从下往上写，之间具有强联系
 */
@Configuration
public class ShiroConfig {

    //ShiroFilterFactoryBean
    @Bean
    public ShiroFilterFactoryBean
getShiroFilterFactoryBean(@Qualifier("securityManager")
DefaultWebSecurityManager defaultWebSecurityManager) {
        ShiroFilterFactoryBean factoryBean = new ShiroFilterFactoryBean();

        //设置安全管理器
        factoryBean.setSecurityManager(defaultWebSecurityManager);

        //添加Shiro的内置过滤器===>进行授权
        /*
            anno：无需认证就可以访问
            authc：必须认证才可以访问
            user：必须拥有“记住我”功能才能用
            perms：拥有对某个资源的权限才可以访问
            role：拥有某个角色权限才可以访问
         */
        Map<String, String> filterMap = new LinkedHashMap<>();
//        filterMap.put("/user/add", "anno");//代表“/user/add”可以被所有人访问
//        filterMap.put("/user/add", "authc");
//        filterMap.put("/user/update", "authc");
        filterMap.put("/user/*", "authc");//支持通配符*

        factoryBean.setFilterChainDefinitionMap(filterMap);

        //登录的请求【当没有权利进入某个模块的时候，跳转到登录页面】
        factoryBean.setLoginUrl("/toLogin");
        return factoryBean;
    }

    //DefaultWebSecurityManager
    @Bean(name = "securityManager")
    public DefaultWebSecurityManager
getDefaultWebSecurityManager(@Qualifier("userRealm") UserRealm userRealm) {
        DefaultWebSecurityManager securityManager = new
DefaultWebSecurityManager();

        //关联Realm
        securityManager.setRealm(userRealm);
        return securityManager;
    }

    //创建Realm对象，需要自定义类
    //并且注册到容器中【方法名就是别名、javaConfig】
    @Bean
    public UserRealm userRealm() {
        return new UserRealm();
    }


}
```

写几个页面：

- 首页

  ```html
  <!DOCTYPE html>
  <html lang="en" xmlns:th="http://www.thymeleaf.org">
  <head>
      <meta charset="UTF-8">
      <title>首页</title>
  </head>
  <body>

  <h1>首页</h1>
  <h3 th:text="${msg}"></h3>
  <hr/>

  <a th:href="@{/user/add}">add</a> |
  <a th:href="@{/user/update}">update</a>
  </body>
  </html>
  ```

- 功能页

  add.html：

  ```html
  <!DOCTYPE html>
  <html lang="en">
  <head>
      <meta charset="UTF-8">
      <title>Title</title>
  </head>
  <body>

  <h1>add</h1>

  </body>
  </html>
  ```

  update.html:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>

<h1>update</h1>

</body>
</html>
```

login.html:

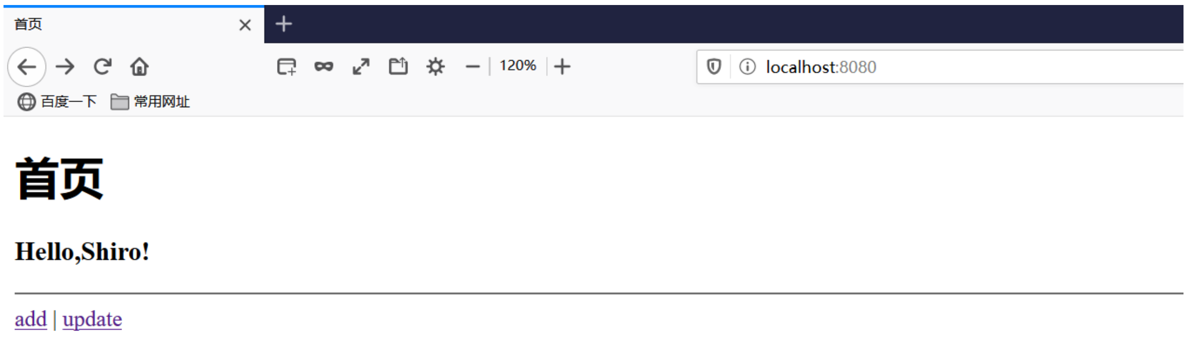
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>登录</h1>
<hr/>
<form action="" method="post">
    <p>用户名：<input type="text" name="username"></p>
    <p>密码：<input type="password" name="password"></p>
    <p><input type="submit" value="登录"></p>
</form>
</body>
</html>
```

路由跳转:

```java
package com.kuang.controller;

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

@Controller
public class MyController {
    @RequestMapping({"/", "/index", "/index.html"})
    public String toIndex(Model model) {
        model.addAttribute("msg", "Hello,Shiro!");
        return "index";
    }

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

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

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

- 测试：

  进入首页：


首页

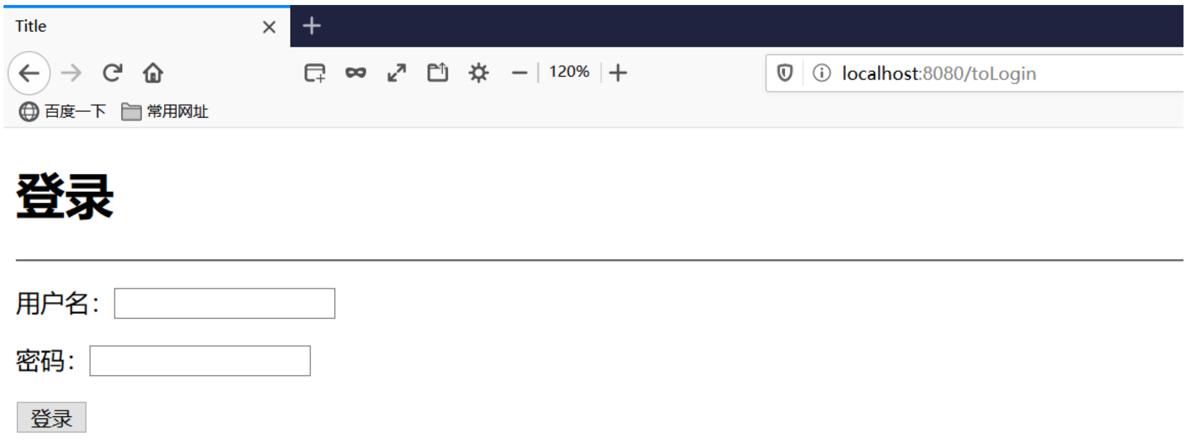
localhost:8080

首页

Hello,Shiro!

add | update

点击进入add.html，发现进不去，直接跳转到了登录页面：


Title

localhost:8080/toLogin

登录

用户名：

密码：

登录

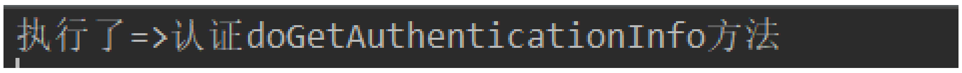
7.3 实现用户认证【没连数据库】

用户认证的执行主要在Realm的类中进行！

我们这里在Controller中接收前端传回的用户信息，所以直接在Controller中先进行验证：

```java
@RequestMapping("/login")
public String login(String username, String password, Model model) {
    //获取当前的用户
    Subject subject = SecurityUtils.getSubject();

    //封装用户的登录数据
    UsernamePasswordToken token = new UsernamePasswordToken(username,
password);

    try {
        //执行登录方法，如果没有异常，则证明成功
        subject.login(token);
        return "index";
    } catch (UnknownAccountException e) {//用户名不存在
        model.addAttribute("msg", "用户名不存在");
        return "login";
    } catch (IncorrectCredentialsException e) {//密码错误
        model.addAttribute("msg", "密码错误");
        return "login";
    }
}
```

当我们打开网页，输入错误密码后，发现：



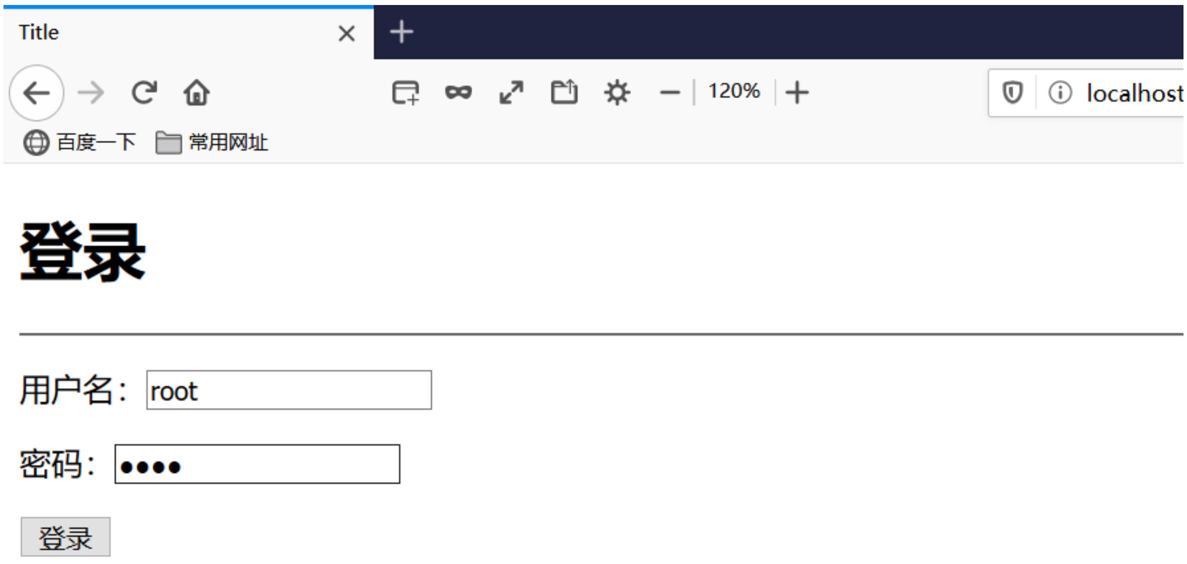
```
执行了=>认证doGetAuthenticationInfo方法
```

所以我们判断，用户名和密码的认证是在Realm类中的doGetAuthenticationInfo进行的：

```java
/**
 * 《认证》
 *
 * @param token
 * @return
 * @throws AuthenticationException
 */
@Override
protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) throws AuthenticationException {
    System.out.println("执行了=>认证doGetAuthenticationInfo方法");

    //用户名、密码应到数据库中取
    String username = "root";
    String password = "root";

    UsernamePasswordToken userToken = (UsernamePasswordToken) token;

    //进行用户名认证
    if(!userToken.getUsername().equals(username) ){
        return null;//抛出异常 UnknownAccountException 用户名不存在
    }

    //密码认证由Shiro做，防止泄露
    return new SimpleAuthenticationInfo("",password,"");
}
```

这时，我们再一次测试：

输入正确的登录信息：

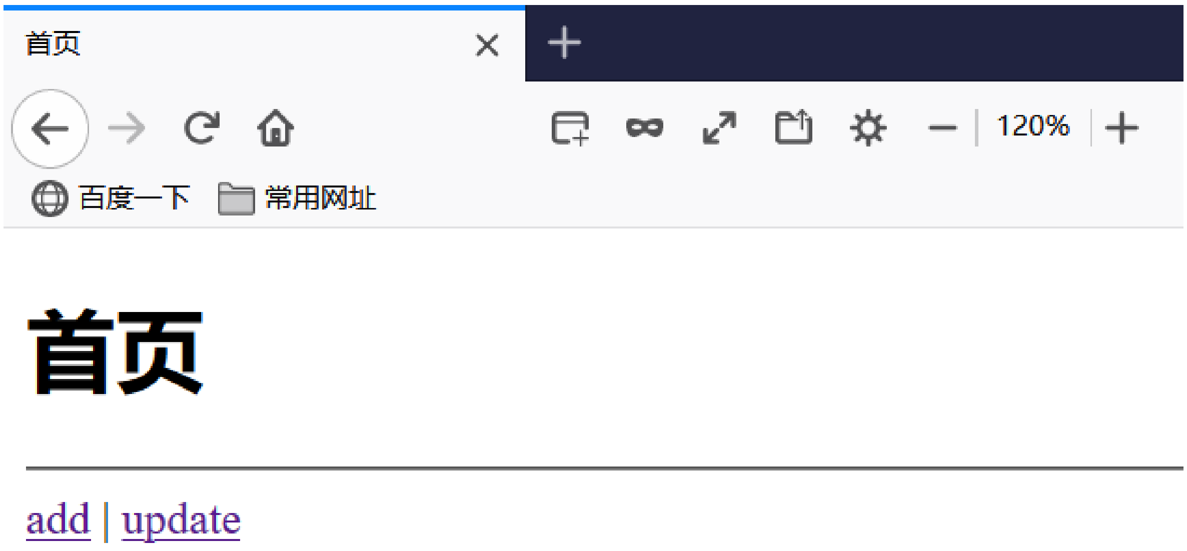


登录

用户名：root

密码：●●●●

跳转成功：



首页

add | update

7.4 整合Mybatis，实现用户认证【连接数据库】

7.4.1 环境搭建

- mysql驱动：

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

- druid数据源：

```xml
<!--Druid数据源-->
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>druid</artifactId>
    <version>1.1.21</version>
</dependency>
```

- 可以来一个日志：

```xml
<!--Log4j-->
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.17</version>
</dependency>
```

- 导入Mybatis整合SpringBoot：

```xml
<dependency>
    <groupId>org.mybatis.spring.boot</groupId>
    <artifactId>mybatis-spring-boot-starter</artifactId>
    <version>2.1.1</version>
</dependency>
```

- 配置文件：

application.yaml:【数据源的配置】

```yaml
spring:
  datasource:
    username: root
    password: mynewroot
    #?serverTimezone=UTC解决时区的报错
    url: jdbc:mysql://localhost:3306/mybatis?serverTimezone=UTC&useUnicode=true&characterEncoding=utf-8
    driver-class-name: com.mysql.jdbc.Driver
    type: com.alibaba.druid.pool.DruidDataSource

    #Spring Boot 默认是不注入这些属性值的，需要自己绑定
    #druid 数据源专有配置
    initialSize: 5
    minIdle: 5
    maxActive: 20
    maxWait: 60000
    timeBetweenEvictionRunsMillis: 60000
    minEvictableIdleTimeMillis: 300000
    validationQuery: SELECT 1 FROM DUAL
    testWhileIdle: true
    testOnBorrow: false
    testOnReturn: false
    poolPreparedStatements: true

    #配置监控统计拦截的filters，stat:监控统计、log4j：日志记录、wall：防御sql注入
    #如果允许时报错  java.lang.ClassNotFoundException: org.apache.log4j.Priority
    #则导入 log4j 依赖即可，Maven 地址：
https://mvnrepository.com/artifact/log4j/log4j
    filters: stat,wall,log4j
    maxPoolPreparedStatementPerConnectionSize: 20
    useGlobalDataSourceStat: true
    connectionProperties:
druid.stat.mergeSql=true;druid.stat.slowSqlMillis=500
```

application.properties：【myabtis的一些配置】

```
# 绑定Mybatis
mybatis.type-aliases-package=com.kuang.pojo
mybatis.mapper-locations=classpath:mapper/*.xml
```

7.4.2 pojo、mapper、service

- pojo：

```java
package com.kuang.pojo;

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

import java.io.Serializable;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class User implements Serializable {
    private int id;
    private String name;
    private String pwd;
}
```

- mapper

```java
@Repository
@Mapper
public interface UserMapper {
    public User queryUserByName(String name);
}
```

- mapper.xml

```xml
<?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="com.kuang.mapper.UserMapper">
    <select id="queryUserByName" resultType="user">
        select * from mybatis.user where name = #{name}
    </select>
</mapper>
```

- service

```java
package com.kuang.service;

import com.kuang.pojo.User;

public interface UserService {
    public User queryUserByName(String name);
}
```

- serviceImpl

```java
package com.kuang.service;

import com.kuang.mapper.UserMapper;
import com.kuang.pojo.User;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class UserServiceImpl implements UserService {
    @Autowired
    private UserMapper userMapper;

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

7.4.3 添加数据库验证

在UserRealm中：

先注入：

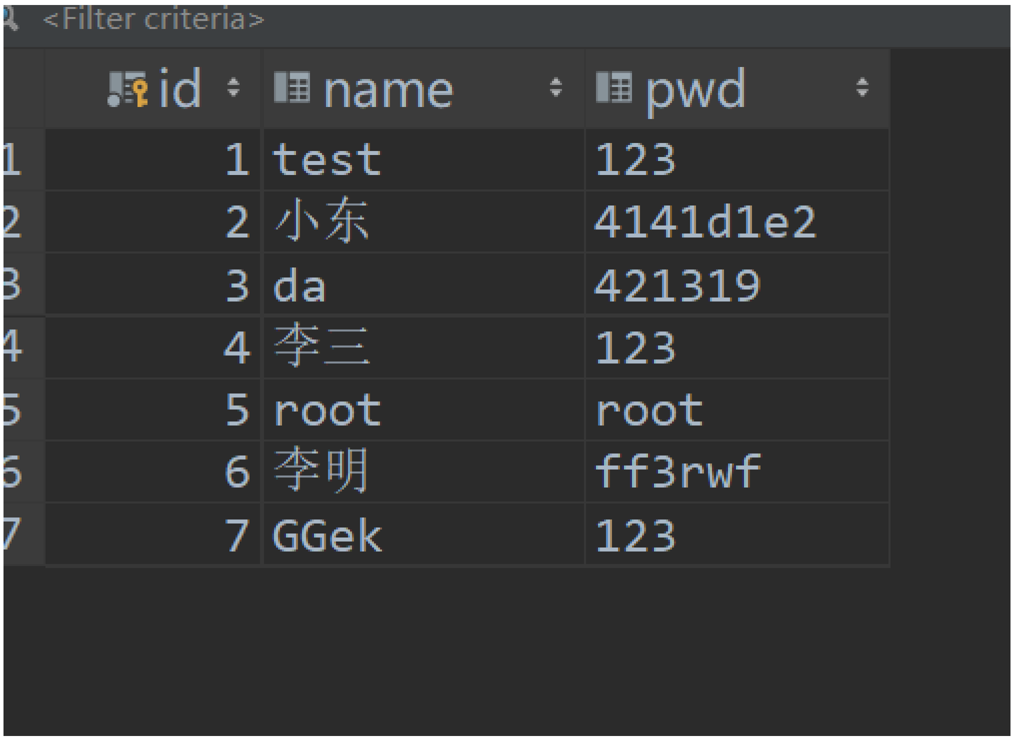
```java
@Autowired
private UserServiceImpl userService;
```

认证方法：

```java
/**
     * 《认证》
     *
     * @param token
     * @return
     * @throws AuthenticationException
     */
    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) throws AuthenticationException {
        System.out.println("执行了=>认证doGetAuthenticationInfo方法");


        UsernamePasswordToken userToken = (UsernamePasswordToken) token;
        //用户名、密码应到数据库中取
        User user = userService.queryUserByName(userToken.getUsername());

        if(null == user){//用户名不存在
            return null;//抛出异常 UnknownAccountException 用户名不存在
        }

//        //进行用户名认证
//        if(!userToken.getUsername().equals(username) ){
//            return null;//抛出异常 UnknownAccountException 用户名不存在
//        }

        //密码认证由Shiro做，防止泄露【加密了】
        //加密方式： MD5   MD5盐值加密
        return new SimpleAuthenticationInfo("",user.getPwd(), "");
    }
```

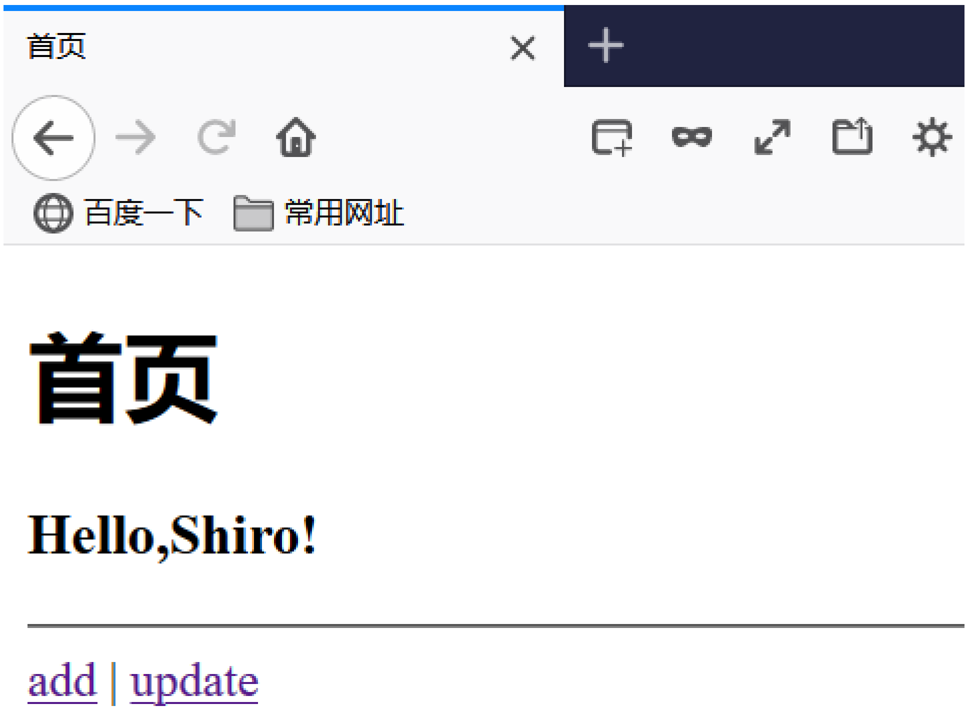
7.4.4 测试运行

数据库的数据：



	id	name	pwd
1	1	test	123
2	2	小东	4141d1e2
3	3	da	421319
4	4	李三	123
5	5	root	root
6	6	李明	ff3rwf
7	7	GGek	123

点击进入add：

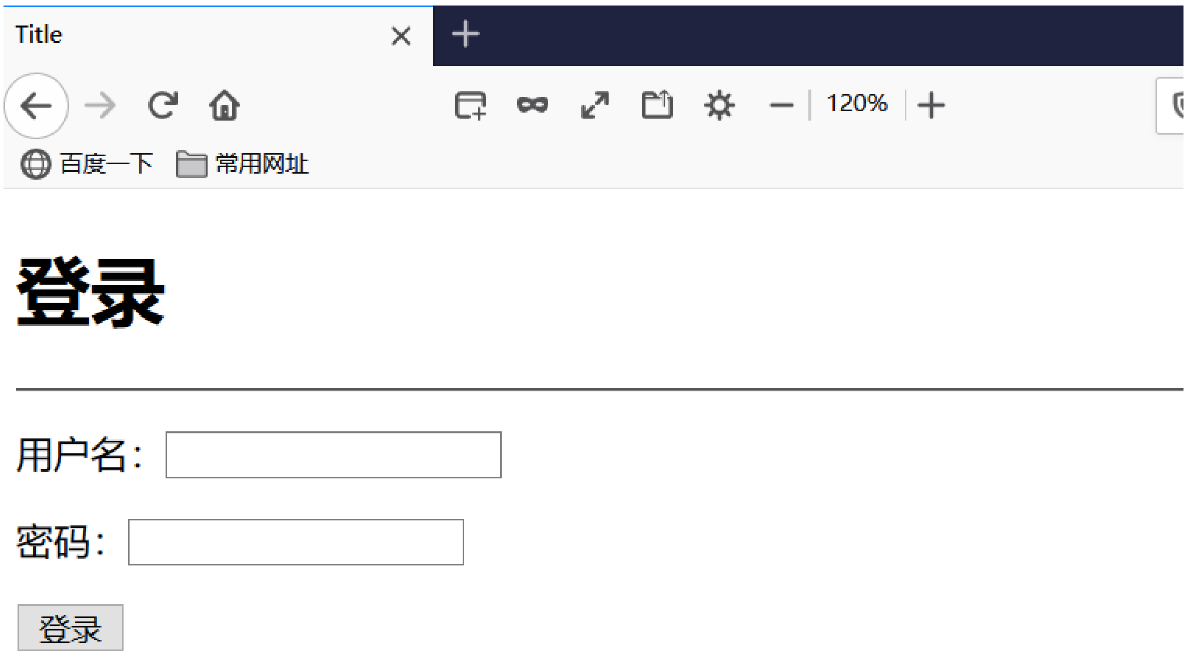

首页

首页

Hello,Shiro!

add | update

没登录【没权限】，跳转到了登录界面：


Title

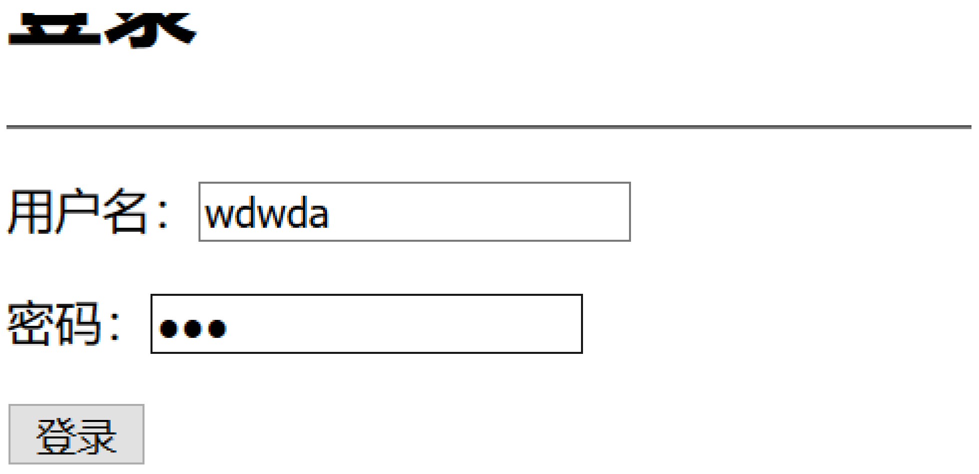
登录

用户名：

密码：

登录

- 输入错误的用户名：

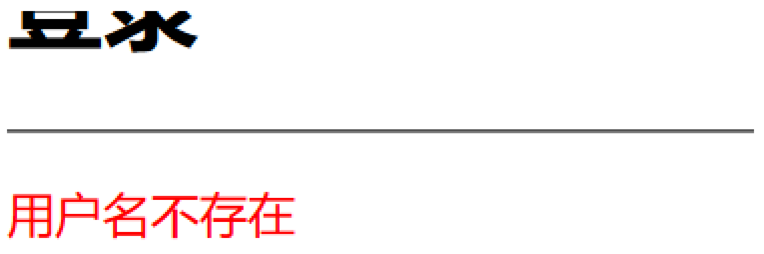


登录

用户名： wdwda

密码： ●●●

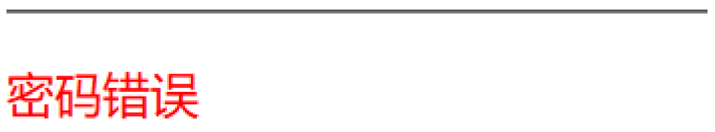
登录



登录

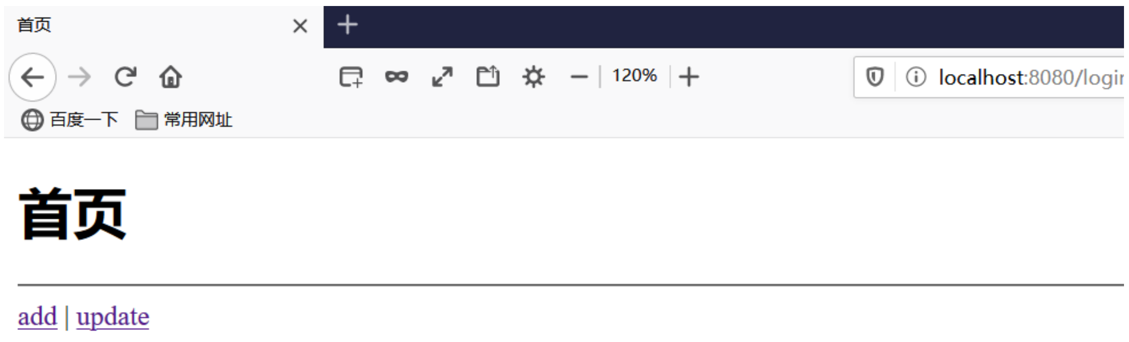
用户名不存在

- 输入错误的密码



密码错误

- 输入正确的登录信息



首页

localhost:8080/logi

120%

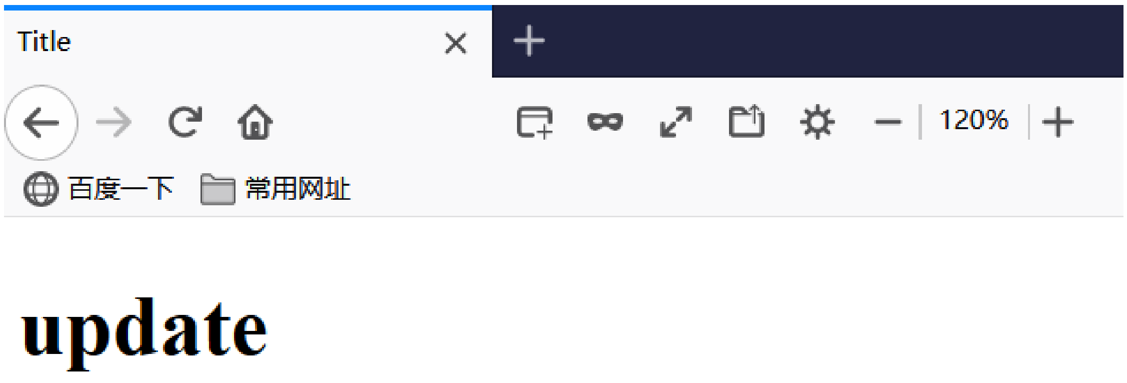
百度一下 常用网址

首页

add | update

验证成功，进入首页！

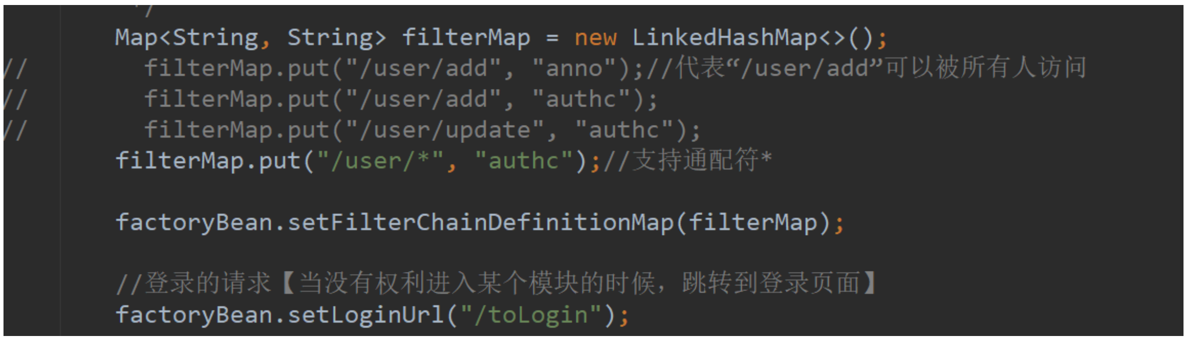
- 进入update



update

7.5 用户请求授权

之前我们只做了一个简单的拦截请求，还算不上是授权：



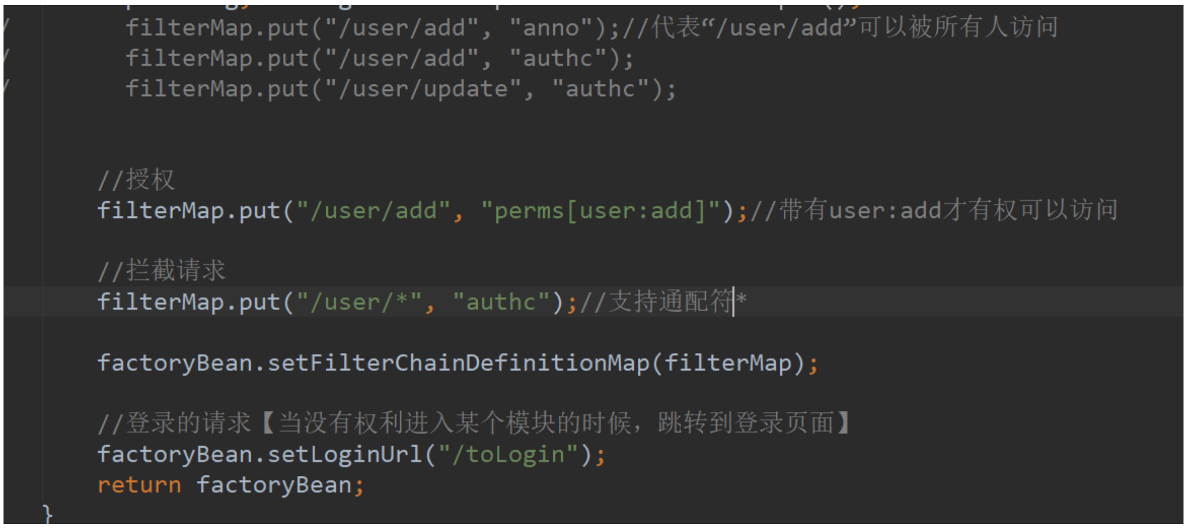
```
    Map<String, String> filterMap = new LinkedHashMap<>();
//      filterMap.put("/user/add", "anno");//代表“/user/add”可以被所有人访问
//      filterMap.put("/user/add", "authc");
//      filterMap.put("/user/update", "authc");
    filterMap.put("/user/*", "authc");//支持通配符*

    factoryBean.setFilterChainDefinitionMap(filterMap);

    //登录的请求【当没有权利进入某个模块的时候，跳转到登录页面】
    factoryBean.setLoginUrl("/toLogin");
```

现在对这个大板块进行进一步的学习。

7.5.1 先禁止进入/user/add请求

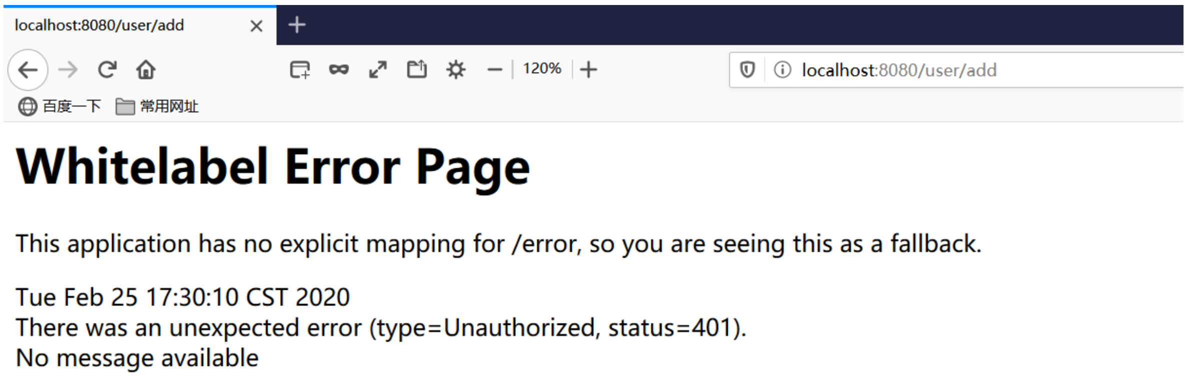
在拦截请求的上面加上：

```
//授权
filterMap.put("/user/add", "perms[user:add]");//带有user:add才有权可以访问
```



```
/       filterMap.put("/user/add", "anno");//代表“/user/add”可以被所有人访问
/       filterMap.put("/user/add", "authc");
/       filterMap.put("/user/update", "authc");


    //授权
    filterMap.put("/user/add", "perms[user:add]");//带有user:add才有权可以访问

    //拦截请求
    filterMap.put("/user/*", "authc");//支持通配符*

    factoryBean.setFilterChainDefinitionMap(filterMap);

    //登录的请求【当没有权利进入某个模块的时候，跳转到登录页面】
    factoryBean.setLoginUrl("/toLogin");
    return factoryBean;
}
```

这时我们再想进入add就会报错：


localhost:8080/user/add

Whitelabel Error Page

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

Tue Feb 25 17:30:10 CST 2020
There was an unexpected error (type=Unauthorized, status=401).
No message available

7.5.2 未授权跳转页面

先设置:

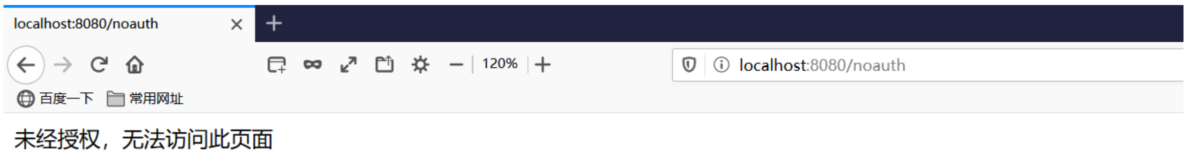
```java
//设置未授权的界面
factoryBean.setUnauthorizedUrl("/noauth");
```

控制层跳转:

```java
@RequestMapping("/noauth")
@ResponseBody
public String Unauthorized(){
    return "未经授权，无法访问此页面";
}
```

测试:

当进入add的时候:


localhost:8080/noauth

未经授权，无法访问此页面

7.5.3 给用户授予权限

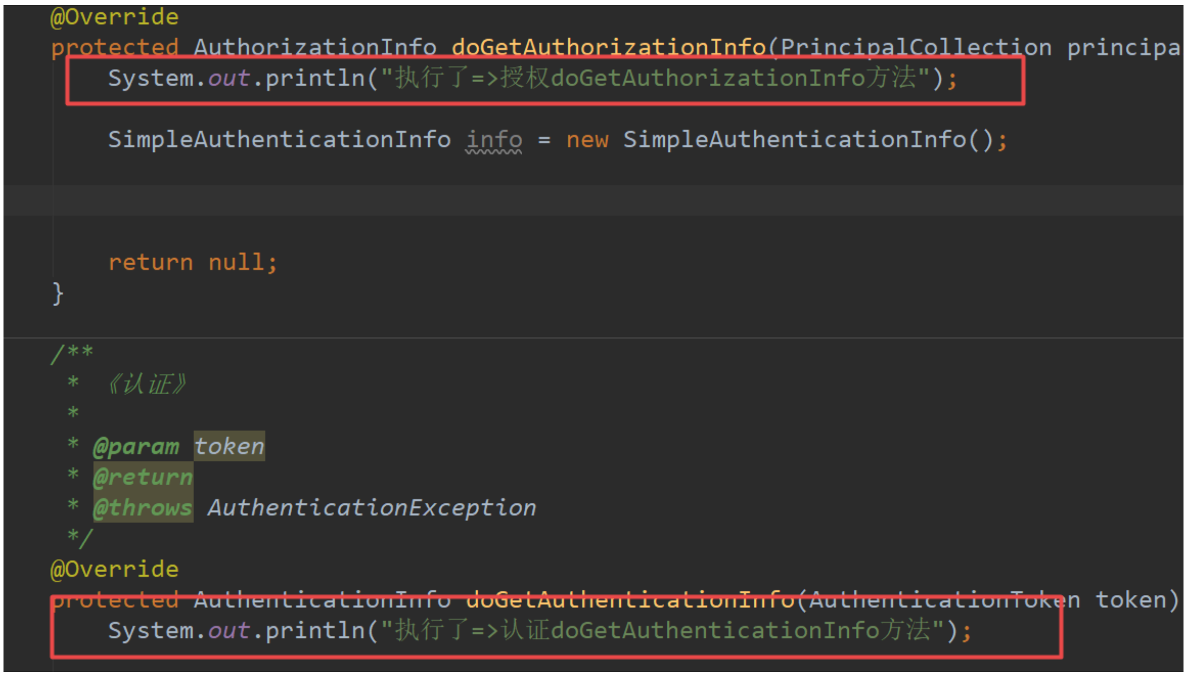
上面的权限都是针对所有人的，显然不合理，不够完善。

我们的授权应该在我们自定义的Realm中。

这里我们说一个重要的东西:

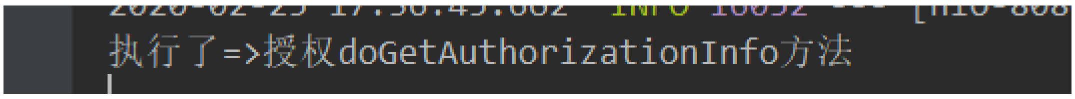
- 登录失败，会进行认证
- 进入需要权限的页面，会进行授权

这就是我们处理的思路。我们之前在自定义的UserRealm中有这个东西:



```
@Override
protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principa
    System.out.println("执行了=>授权doGetAuthorizationInfo方法");

    SimpleAuthenticationInfo info = new SimpleAuthenticationInfo();


    return null;
}

/**
 * 《认证》
 *
 * @param token
 * @return
 * @throws AuthenticationException
 */
@Override
protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token)
    System.out.println("执行了=>认证doGetAuthenticationInfo方法");
```

而刚才进入需要权限页面的时候：

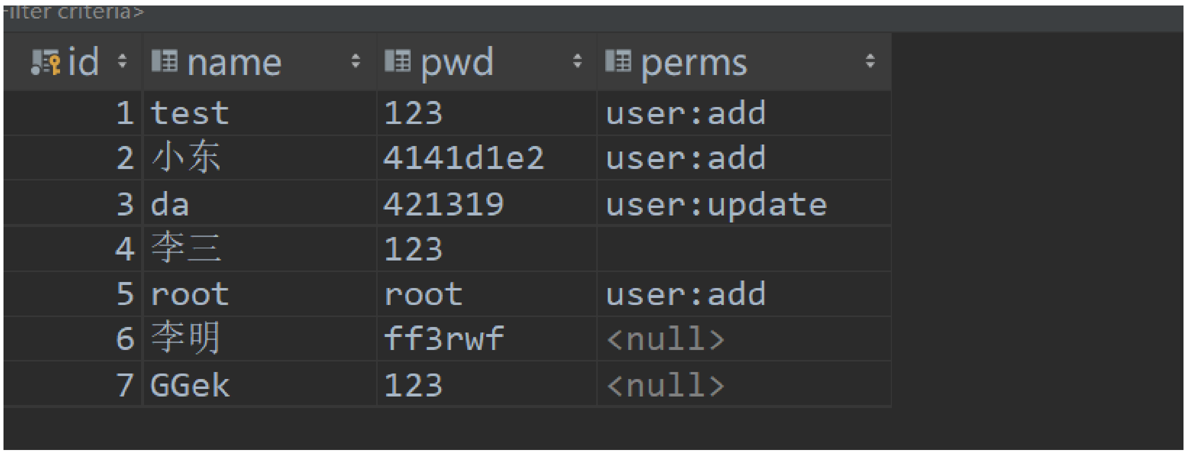


```
执行了=>授权doGetAuthorizationInfo方法
```

所以，当进入授权的时候，执行了doGetAuthorizationInfo这个方法。

于是我们应该在这个方法下面进行用户的授权。

先列出用户的权限：



id	name	pwd	perms
1	test	123	user:add
2	小东	4141d1e2	user:add
3	da	421319	user:update
4	李三	123	
5	root	root	user:add
6	李明	ff3rwf	<null>
7	GGek	123	<null>

配置类：

```
//授权，当没有授权的时候，跳转到未授权的界面
filterMap.put("/user/add", "perms[user:add]");//带有user:add才有权可以访问
filterMap.put("/user/update", "perms[user:update]");//带有user:update才有权可以访问
```

自定义的Realm中进行授权：

```
/**
     * 《授权》
     *
     * @param principalCollection
     * @return
     */
```

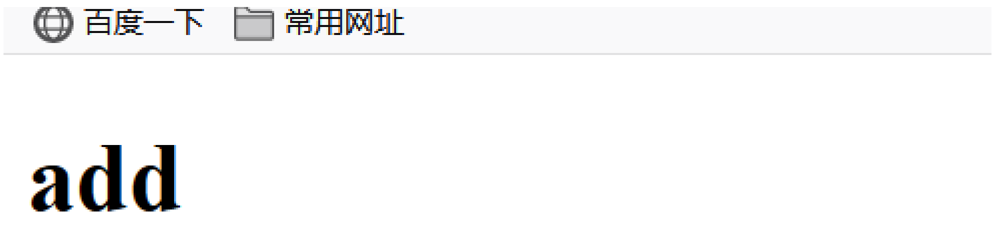
```java
    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principalCollection) {
        System.out.println("执行了=>授权doGetAuthorizationInfo方法");

        SimpleAuthorizationInfo info = new SimpleAuthorizationInfo();

        //拿到当前登录的对象
        Subject subject = SecurityUtils.getSubject();

        //return new SimpleAuthenticationInfo(user,user.getPwd(), "");传递参数user了，所以这里才取得到
        User currentUser = (User) subject.getPrincipal();//拿到User对象

        //设置当前用户的权限
        info.addStringPermission(currentUser.getPerms());

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

        return info;
    }
```

注意，这里在protected AuthenticationInfo doGetAuthenticationInfo的return添加了一个参数user：

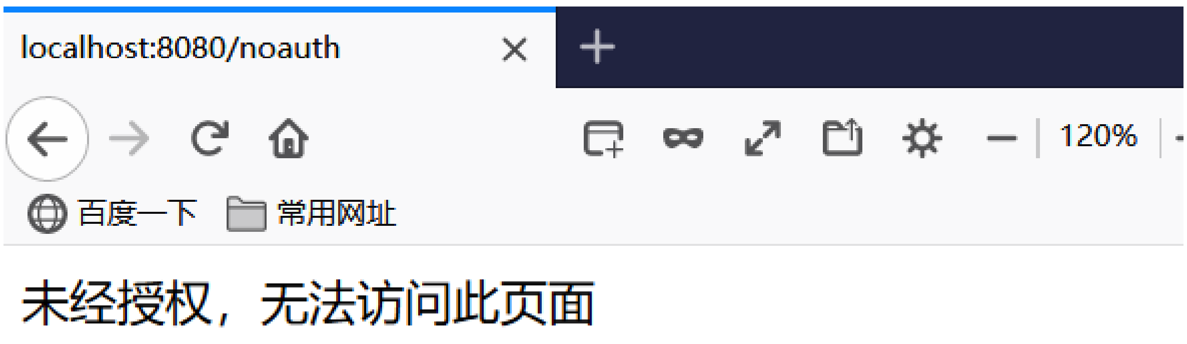
```java
return new SimpleAuthenticationInfo(user, user.getPwd(), "");
```

测试：

【root用户，进得去add，进不去update】



add



未经授权，无法访问此页面

7.5.4 结合Thymeleaf

用户拥有哪些权限，才展示哪些菜单【add/update】。

- 导入依赖：

```xml
<!--Shiro和Thymeleaf整合-->
<dependency>
    <groupId>com.github.theborakompanioni</groupId>
    <artifactId>thymeleaf-extras-shiro</artifactId>
    <version>2.0.0</version>
</dependency>
```

- 命名空间

  ```
  xmlns:shiro="http://www.thymeleaf.org/thymeleaf-extras-shiro"
  ```

- 注册Bean：

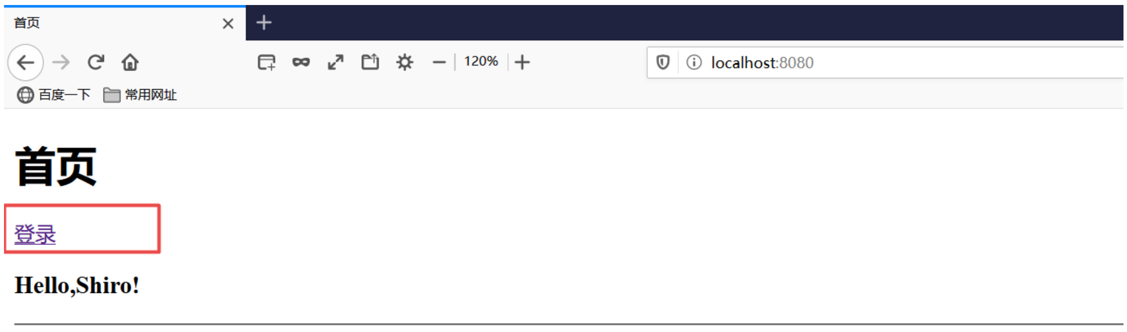
```
//整合Shiro和Thymeleaf===>ShiroDialect
@Bean
public ShiroDialect shiroDialect() {
    return new ShiroDialect();
}
```

- 直接在前端使用

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org"
      xmlns:shiro="http://www.thymeleaf.org/thymeleaf-extras-shiro">
<head>
    <meta charset="UTF-8">
    <title>首页</title>
</head>
<body>

<h1>首页</h1>


<div th:if="${session.loginUser == null}">
    <a th:href="@{/toLogin}">登录</a>
</div>


<h3 th:text="${msg}"></h3>
<hr/>

<div shiro:hasPermission="user:add">
    <a th:href="@{/user/add}">add</a>
</div>
<div shiro:hasPermission="user:update">
    <a th:href="@{/user/update}">update</a>
</div>


</body>
</html>
```

- 测试

未登录的时候：

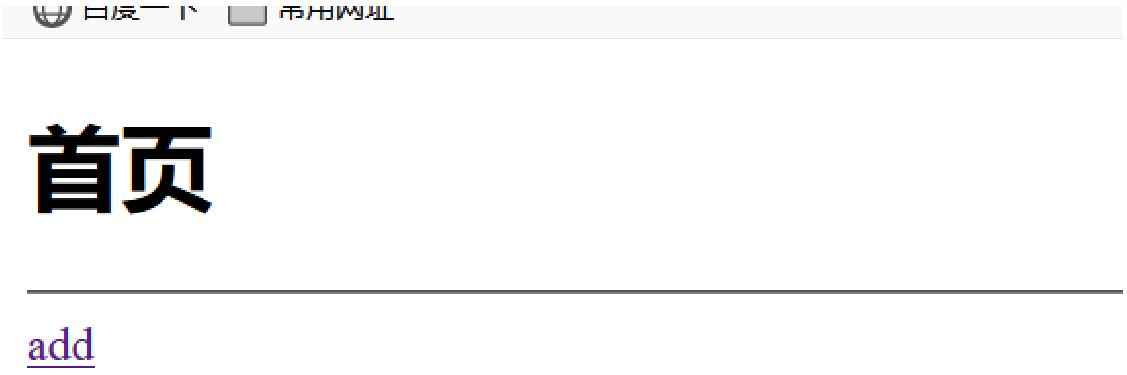


首页

登录

Hello,Shiro!

登录具有add权限的root用户：



首页

add