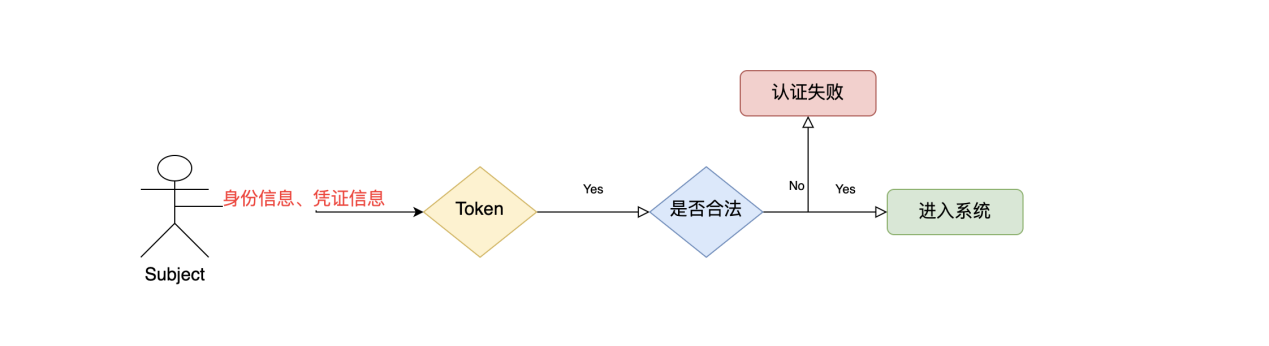


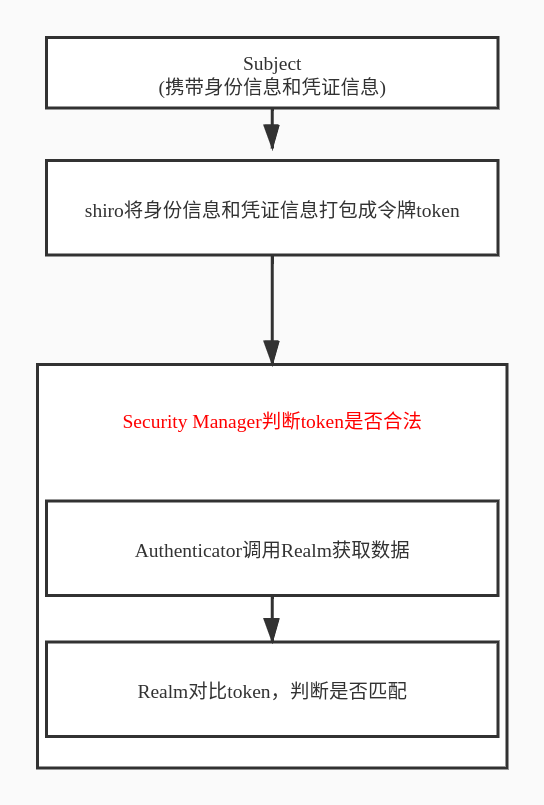
### **1.Shiro 核心架构**



|  |  |
| --- | --- |
| Authenticator | 认证 |
| Authorizer | 授权 |
| Session Manager  (shiro 管理在web环境下的核心会话) | 在web环境下，shiro要解决认证和用户的授权。用户在认证通过后，要将用户的标记存储在对应的容器上。（以session的形式存储用户，用来对用户的会话进行追踪） |
| Session DAO | 对会话数据进行增删改查 |
| Cache manager  (缓存用户的权限) | 在数据不发生变化的情况下，用户的授权仅仅是在第一次会调用数据库。拿到用户的权限信息后会统一放到cache manager。后续获取用户权限时，直接从cachem manager中获取，减少数据库的访问压力。 |
| Cryptography | 密码生成器 |
| Pluggable Realms | ??????????????????/ |

### **2.认证流程**





### **3.Shiro的配置文件 .ini结尾的文件**

(1).ini 可以书写比较复杂的数据格式

(2)配置文件放置的位置：/src/main/resources/\*\*

(3)整合springboot用不到ini配置文件。Ini配置文件用来学习shiro书写系统中相关权限数据。

### **4.单机模式的认证**

|  |
| --- |
| **public static void** main(String[] args) {   *//1.创建安全管理器对象* DefaultSecurityManager securityManager = **new** DefaultSecurityManager();   *//2.给安全管理器设置realm* securityManager.setRealm(**new** IniRealm(**"classpath:shiro.ini"**));   *//3.SecurityUtils 给全局安全工具类设置安全管理器* SecurityUtils.*setSecurityManager*(securityManager);   *//4.关键对象 subject 主体* Subject subject = SecurityUtils.*getSubject*();   *//5.创建令牌* UsernamePasswordToken token = **new** UsernamePasswordToken(**"xiaochen"**,**"123"**);   **try**{  System.***out***.println(**"认证状态: "**+ subject.isAuthenticated());   **subject.login(token);*//用户认证***System.***out***.println(**"认证状态: "**+ subject.isAuthenticated());  }**catch** (UnknownAccountException e){  e.printStackTrace();  System.***out***.println(**"认证失败: 用户名不存在~"**);  }**catch** (IncorrectCredentialsException e){  e.printStackTrace();  System.***out***.println(**"认证失败: 密码错误~"**);  } } |

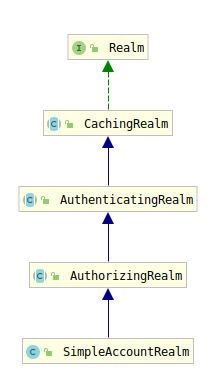
### **5.认证源码解析：**

（1）最终执行用户名比较：SimpleAccountRealm(**extends** AuthorizingRealm).doGetAuthenticationInfo

在这个方法中完成了用户名的校验

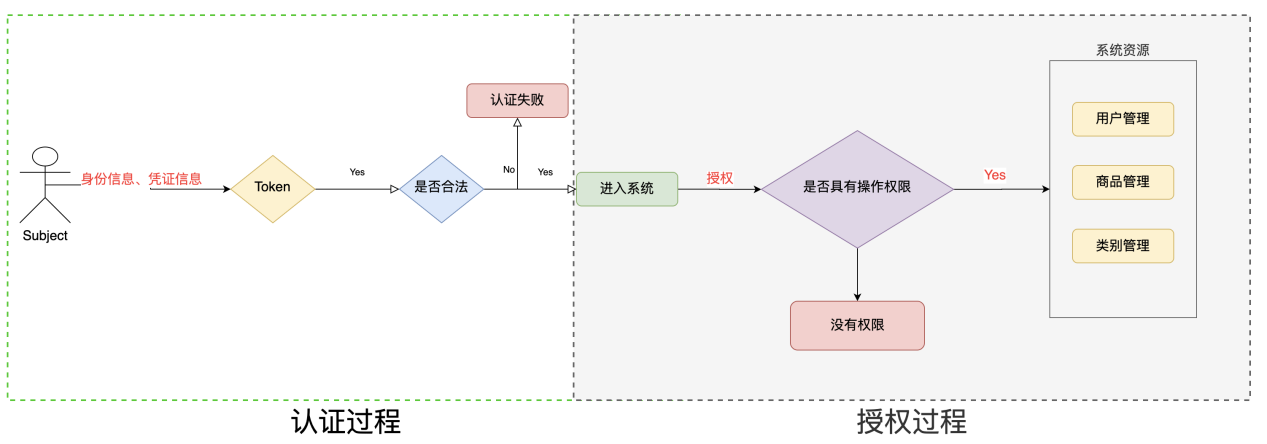
（2）最终密码校验是在AuthenticatingRealm.assertCredentialsMatch

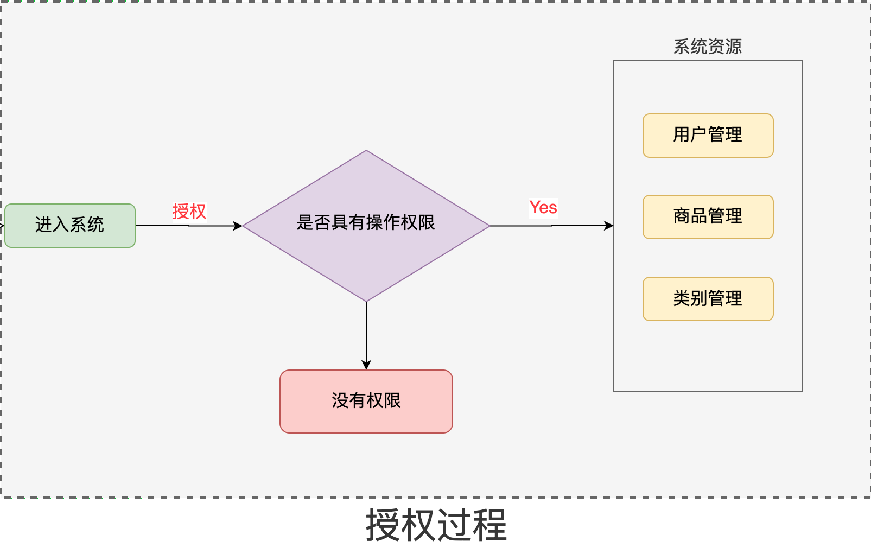
结论：用户名的校验**extends** AuthorizingRealm,重写doGetAuthenticationInfo方法。密码的校验交由shiro自动处理。密码可能需要做加密处理等等。



|  |  |  |
| --- | --- | --- |
| AuthenticatingRealm | 认证realm | doGetAuthenticationInfo |
| AuthorizingRealm | 授权realm | doGetAuthorizationInfo |

### **5.授权流程**



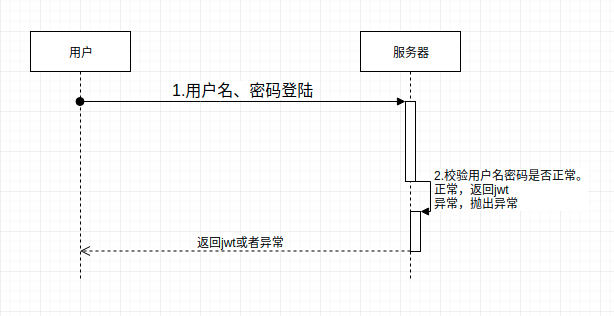


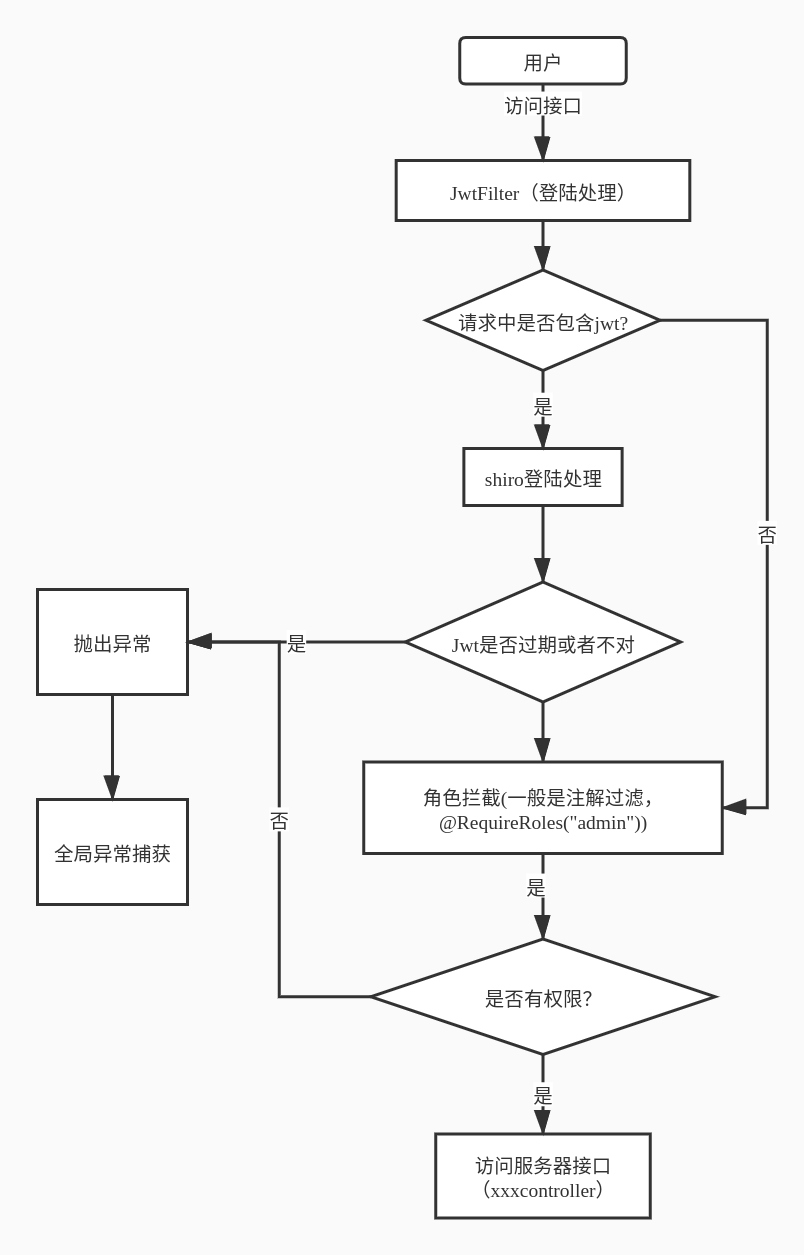
6.为什么使用JWT,而不是cookie?

手机或者小程序访问服务器时，是没有cookie的（浏览器自带cookie）。

JWT：无session的token认证

7.Jwt登陆流程：





8.全局异常捕获：

<https://www.cnblogs.com/UncleWang001/p/10949318.html>

#### **@ControllerAdvice 注解，可以用于定义@ExceptionHandler、@InitBinder、@ModelAttribute，并应用到所有@RequestMapping中。**

|  |
| --- |
| @ControllerAdvice  public class MyControllerAdvice {  /\*\*  \* 应用到所有@RequestMapping注解方法，在其执行之前初始化数据绑定器  \* @param binder  \*/  @InitBinder  public void initBinder(WebDataBinder binder) {}  /\*\*  \* 把值绑定到Model中，使全局@RequestMapping可以获取到该值  \* @param model  \*/  @ModelAttribute  public void addAttributes(Model model) {  model.addAttribute("author", "Magical Sam");  }  /\*\*  \* 全局异常捕捉处理  \* @param ex  \* @return  \*/  @ResponseBody  @ExceptionHandler(value = Exception.class)  public Map errorHandler(Exception ex) {  Map map = new HashMap();  map.put("code", 100);  map.put("msg", ex.getMessage());  return map;  }  } |

#### **启动应用后，被 @ExceptionHandler、@InitBinder、@ModelAttribute 注解的方法，都会作用在 被 @RequestMapping 注解的方法上。**

|  |
| --- |
| @ModelAttribute：在Model上设置的值，对于所有被 @RequestMapping 注解的方法中，都可以通过 ModelMap 获取，如下：  @RequestMapping("/home")  public String home(ModelMap modelMap) {  System.out.println(modelMap.get("author"));  }  //或者 通过@ModelAttribute获取  @RequestMapping("/home")public String home(@ModelAttribute("author") String author) {  System.out.println(author);  } |
| @ExceptionHandler 拦截了异常，我们可以通过该注解实现自定义异常处理。其中，@ExceptionHandler 配置的 value 指定需要拦截的异常类型，上面拦截了 Exception.class 这种异常。  /\*\*全局异常捕捉处理\*/  @ResponseBody  @ExceptionHandler(value = Exception.class)  public Map errorHandler(Exception ex) {  Map map = new HashMap();  map.put("code", 100);  map.put("msg", ex.getMessage());  return map;  }  /\*\*拦截捕捉自定义异常 MyException.class\*/  @ResponseBody  @ExceptionHandler(value = MyException.class)  public Map myErrorHandler(MyException ex) {  Map map = new HashMap();  map.put("code", ex.getCode());  map.put("msg", ex.getMsg());  return map;  } |

**补充：如果全部异常处理返回json，那么可以使用 @RestControllerAdvice 代替**