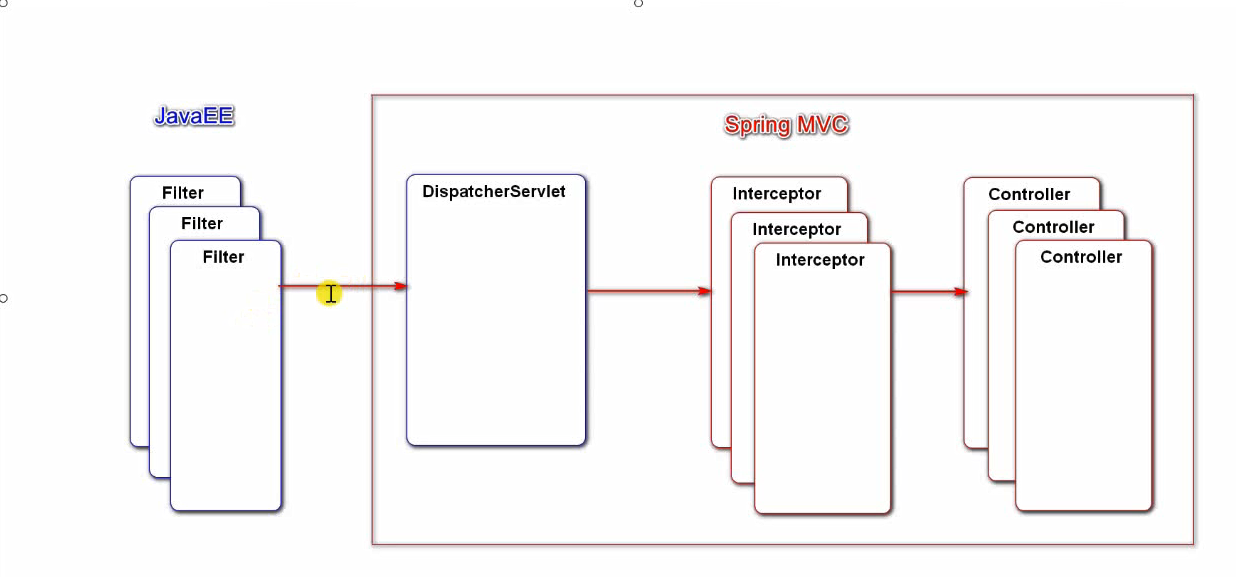
对系统提供安全性的保障，主要是管理系统用户的权限，管理权限主要（核心）从两方面入手认证和授权，认证：判断用户是否登录，如果未登录不可能去访问私信，发帖等功能（相当于以前的登录表单（cookis、threadlocal那个））；授权：认证后的进一步判断，判断是否有访问此功能的权限，比如给帖子的加精、置顶等，这些都需要高级别的权限（管理员、版主），授权主要为了判断当前用户是否有访问这个功能的权限，这些都可以通过security来实现

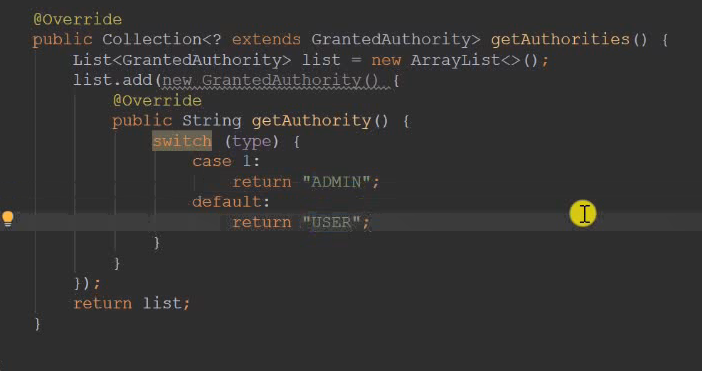


Spring security底层的大概机制：spring MVC会将请求统一在dispatcherServlet进行处理，之后将请求传给控制器，可能会有拦截器进行拦截，dispatcherServlet实现了javaEE的servlet接口，filter过滤器会拦截对servlet的请求，security的底层会利用filter来实现系统安全，权限控制的判断时机较为靠前，在请求还未到dispatcherServlet就会被拦截



User实体类要实现userDetails接口，实现接口中的方法：账号是否过期、账号是否锁定、凭证未过期、账号是否可用、

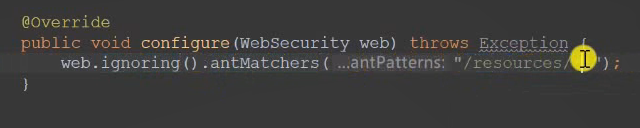
获取用户权限（返回的是权限集合，一个元素是一个权限）



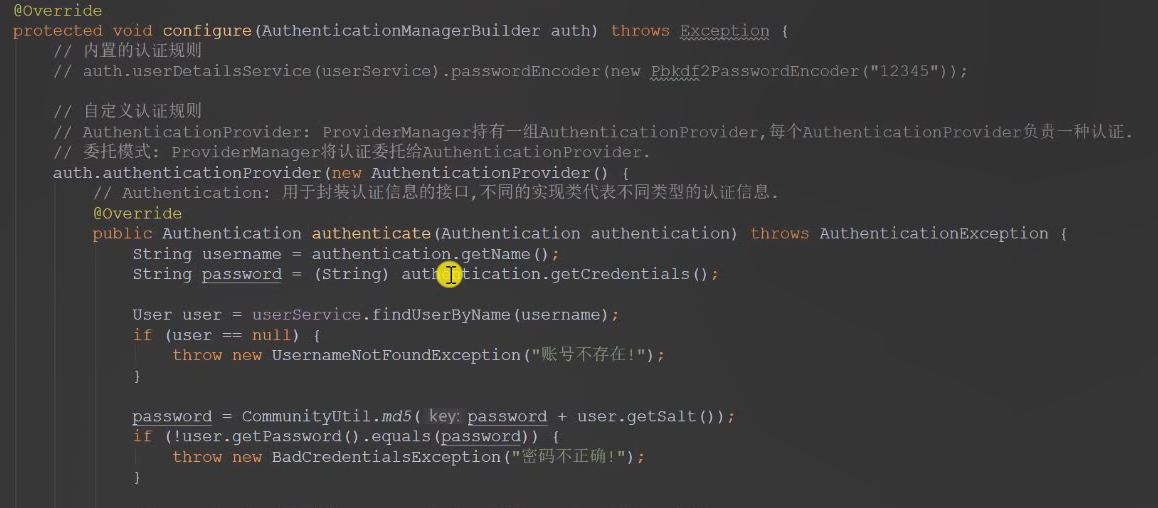
Userservice实现userDetailsService接口，实现方法：根据用户名查用户

配置security配置类（security不需要每个功能都去实现一遍，直接在配置类中实现即可）：要继承webSecurityConfigurerAdapter，需要注入userServiceDetails的实现类，因为其底层需要

首先需要重写的方法是拦截忽略哪些路径，视频中忽略了静态资源下的路径



认证：supports是为了判断支持哪种类型的认证，因为可能是微信二维码那种类型，咱们的是输入账号密码类型





授权逻辑（感觉一个逻辑是一个filter）：

登陆的逻辑：

登录页面也是一种授权，授权你可以访问登录后的网站信息，首先要获取到登录的路径（request），之后才能通过认证的想过逻辑判断是否给你授权，所以还需要写成功的逻辑和失败的逻辑（配置器），成功后重定向到首页路径，让你访问首页；失败则回到登录页面，但想要把本次的错误信息也给到登录页面，就不能重定向了，因为你重定向后相当于是一个新的请求，但你的错误信息是在本次请求逻辑中的，下次请求是得不到的，需要传参，所以需要通过转发请求给登录页面路径

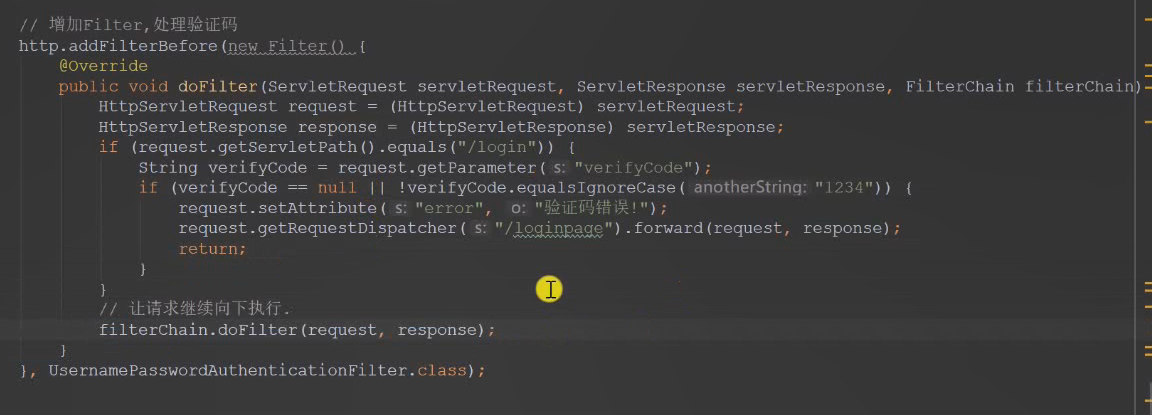
退出的逻辑：直接在成功的配置器中重定向到首页

授权配置：获取私信的路径，之后设置普通用户和管理员可以访问

获取管理（置顶、加精）的路径，设置管理员可以访问（若通过则可以去访问这个路径，进入到控制层相应的返回体中）

权限不匹配的返回路径（需要在控制层另外设置返回体）

因为security在用户登录验证的filter中没有处理验证码的filter，所以需要新增一个filter，通过if来控制拦截请求的范围，如果验证码错误直接return，如果通过或者不是登陆请求，继续执行下一步操作（可能传给另一个filter也可能是servlet）



在控制层：

认证成功后会将结果（账户名、密码）通过SecurityContextHolder存入SecurityContext中，想要获取直接调用就可以

例如：登陆请求（security已经判断过用户名密码是否正确，不用像以前一样再判断一遍账户密码，直接获取就好）

