Echo 的登录认证和授权是怎么做的

原创 飞天小牛肉 [飞天小牛肉](javascript:void(0);)

**飞天小牛肉**

微信号 CS-Wiki

功能介绍 InfoQ & 阿里云签约作者，分享原创技术干货和成长经验

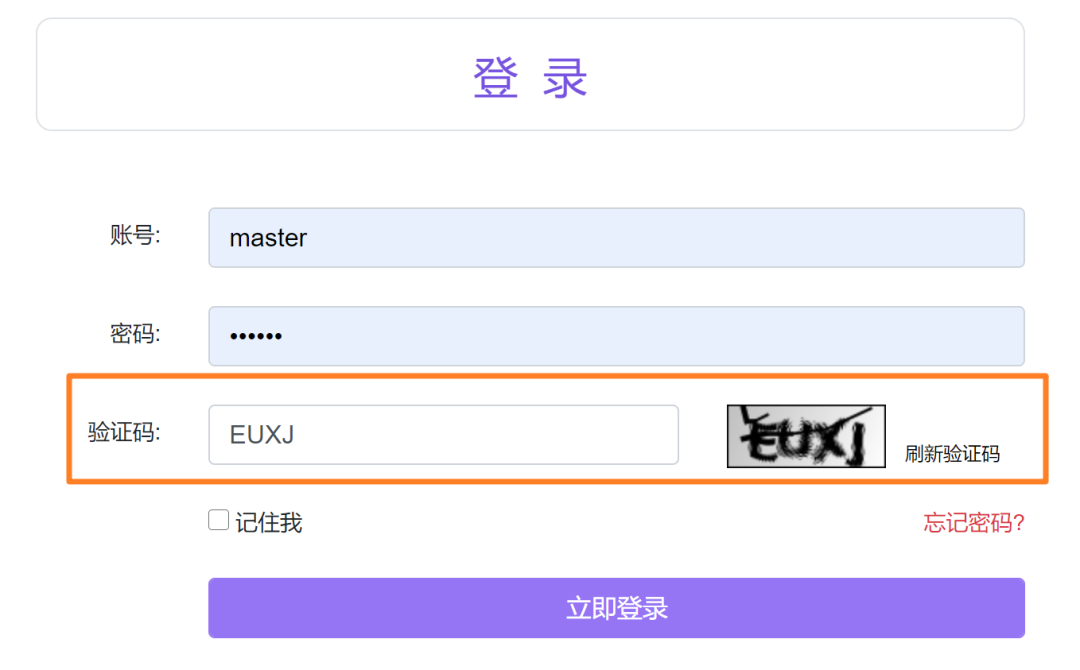
2021-03-28[原文](https://mp.weixin.qq.com/s?__biz=MzI0NDc3ODE5OQ==&mid=2247486159&idx=2&sn=175c9037e8fc37169b63074e9127b512&chksm=e959df77de2e566135d4c23da666716d5c7e3f69a11a40f6793b016cac0768ed83acfe473bd3&scene=178&cur_album_id=1744497649518493697#rd) 发表于

收录于合集 #Echo 学习教程 18个

这篇教程鸽的有点久了，抱歉各位 

**验证码**

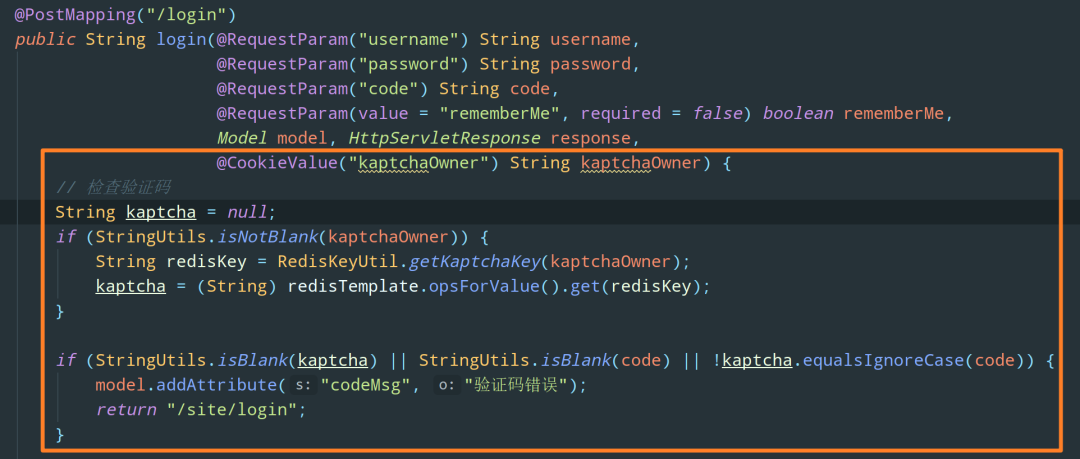
首先，登录的时候会随机生成验证码，如何把这个验证码和当前用户对应起来，实现验证码的校验呢？



显然，由于这个时候用户还没有登录，我们是没有办法通过用户的 id 来唯一的对应它的验证码的。所以这个时候我们考虑生成一个随机的 id 来暂时的代替这个用户，将其和对应的验证码**暂时**存入 Redis 中（60s）。并且在 Cookie 中暂时存一份为这个用户生成的随机 id（60s）。

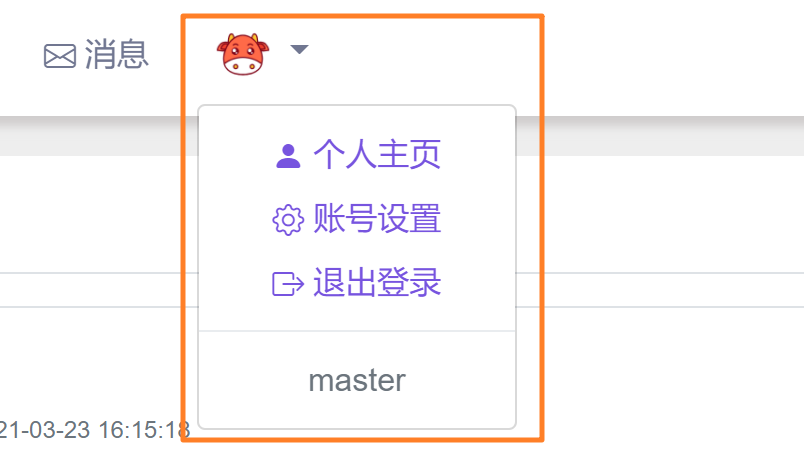


这样，当用户点击登录按钮后，就会去 Cookie 中获取这个随机 id，然后去 Redis 中查询对应的验证码，判断用户输入的验证码是否一致。



**登录认证并持有用户状态**

OK，用户输入用户名和密码并且校验完验证码之后，就登录成功了，那我们如何在一次请求中去保存这个用户的状态？如何回显用户的信息呢？



为此，我们设计了一个 LoginTicket 类：



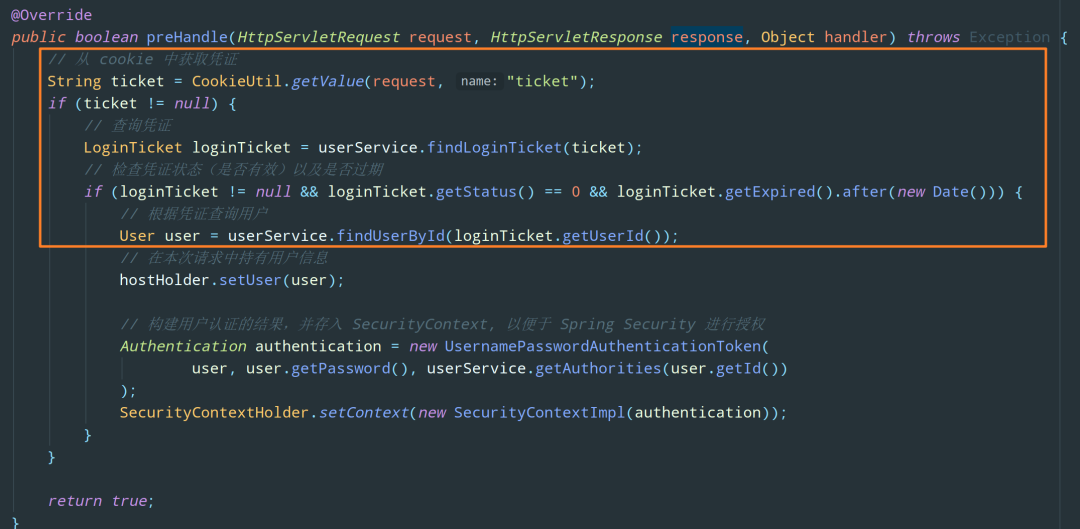
解释一下，每个用户登录成功后，我们都会为其生成一个随机的唯一的登录凭证实体类对象 LoginTicket（包含用户 id、登录凭证字符串 ticket、是否有效、过期时间），我们把这个登录凭证实体类对象存储在 Redis 中（key 就是登录凭证字符串 ticket）。而所谓登录凭证的无效，就是指用户登出后，这个凭证就会被设置为无效状态；凭证的默认过期时间是 1000s。这段代码在 UserService 中：



并且，我们在 Cookie 中也同样存储了一份登录凭证的字符串 ticket，过期时间和 Redis 中的是一样的。点击记住我可以延长过期时间。这段代码在 LoginController 中：



OK，存储完 LoginTicket 后，我们就可以根据它来获取用户的状态了。我们定义了一个拦截器 LoginTicketInterceptor，**每次请求之前都会从 Cookie 获取到 ticket，然后根据 ticket 去 Redis 中查看这个用户的登录凭证 LoginTicket 是否过期和是否有效**，只有登录凭证有效且没有过期才会执行请求，不然就会跳转到登录界面。



如果该用户的登录凭证有效且没有过期，那我们就可以在本次请求中持有这个用户的信息了。如果持有呢？一般来说可以使用 Session，但是 Session 无法在分布式存储中发挥有效的作用。详细来说就是：客户端发送一个请求给服务器，经过负载均衡后该请求会被分发到集群中多个服务器中的其中一个，由于不同的服务器可能含有不同的 Web 服务器，而 Web 服务器之间并不能发现其他 Web 服务器中保存的 Session 信息，这样，它就会再次重新生成一个 JSESSIONID，导致之前的状态丢失。

所以这里我们考虑使用 ThreadLocal 保存用户信息，ThreadLocal 在每个线程中都创建了一个用户信息副本，也就是说每个线程都可以访问自己内部的用户信息副本变量。关于 ThreadLocal 的详细内容会放在【技术要点篇】部分，来看下 HostHolder 类：



关于拦截器做的事情，我们来梳理一下：

1）在 Controller 执行之前：检查登录凭证状态，若登录凭证有效且未过期则在本次请求中持有该用户信息



2）在模板引擎之前：将用户信息存入 modelAndView，便于模板引擎调用

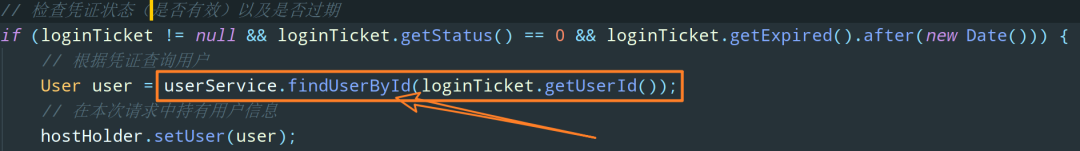


3）在 Controller 执行之后（即服务端对本次请求做出响应后）：清理本次请求持有的用户信息（也就是 ThreadLocal 的 remove，如果没有即时 remove 会导致 OOM）

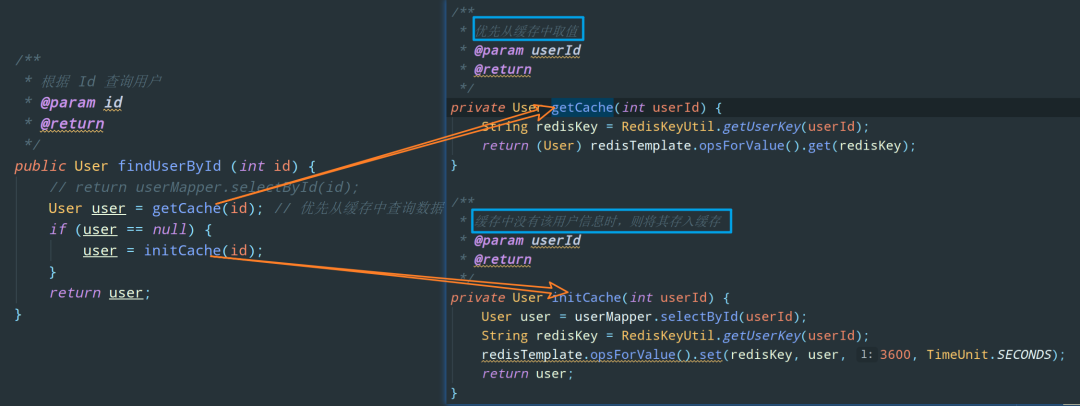


**性能优化**

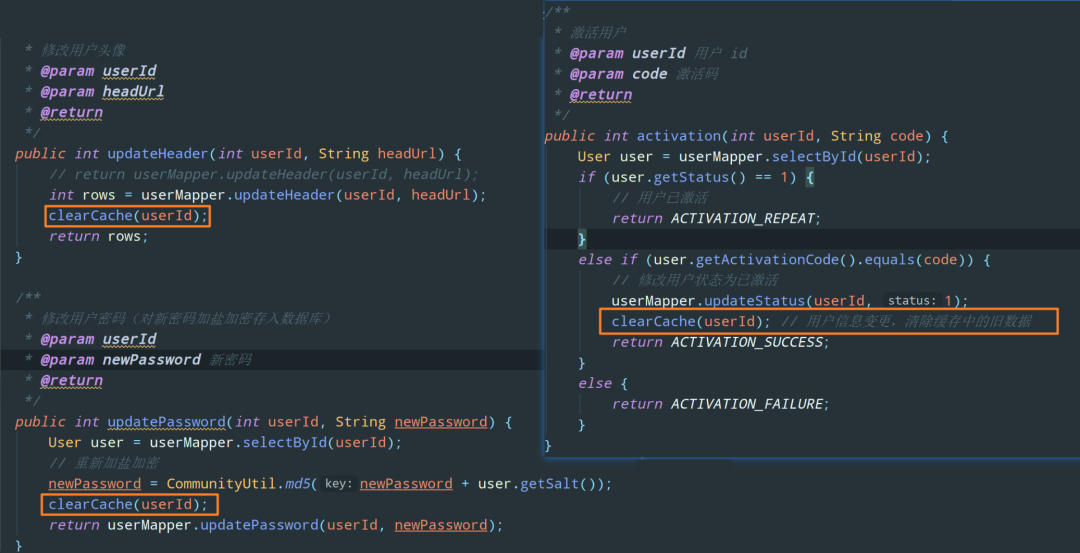
这里有一个点我们进行了稍微的优化。就是我们的拦截器在每次请求前通过 Cookie 去 Redis 中查询登录凭证 LoginTicket 然后获取到用户 id 后，需要去数据库中查询用户信息，然后才能在本次请求中持有用户信息。



显然，每次请求前都需要经过这个步骤，这个访问数据库的频率还是很频繁的。因此我们考虑把登录成功的用户信息在 Redis 中保存一会，拦截器每次查询前先去 Redis 中查询，如果 Redis 中没有再去查询数据库，然后写进 Redis。OK，我们来看看 findUserById 方法具体是怎么实现的：



缓存和数据库的一致性问题的话，使用的是**旁路缓存模式**，也就是先更新数据库，然后直接删除缓存中的数据。比如对于修改用户密码、修改用户头像、激活用户后用户 status 的改变等，这些涉及数据库表中字段更新的操作，都需要删除缓存：



可能有同学就会问了，**为什么是直接删除缓存，而不是也相应的更新缓存呢**？

答案很简单，在多线程的环境下，假设线程 A 更新了数据库中的某个字段为 1，如果在线程 A 提交之前，线程 B 又修改了这个字段为 2 并且先于线程 A 做了提交，那么线程 A 接下来提交的数据就是脏数据。直接删除缓存可以避免这个问题。

总的来说，这个认证流程是这样的：

* 用户登录 —> 生成登录凭证存入 Redis，Cookie 中存一份 key
* 每次执行请求都会通过 Cookie 去 Redis 中查询该用户的登陆凭证是否过期和是否有效。点击记住我可以延长登录凭证的过期时间，用户退出则其登录凭证变为无效状态
* 根据这个登录凭证对应的用户 id，去数据库中查询这个用户信息
* 使用 ThreadLocal 在本次请求中一直持有这个用户信息
* 优化点：每次请求前都需要去数据库查询这个用户信息，访问频率比较高，所以我们考虑把登录成功的用户信息在 Redis 中保存一会，拦截器每次查询前先去 Redis 中查询，然后缓存和数据库的一致性问题的话，使用的是旁路缓存模式，也就是先更新数据库，然后直接删除缓存中的数据。

**授权**

认证的话上面大家也看到了，是我们自己写的逻辑，跳过了 Spring Security，那我们就需要把我们自己做的逻辑认证的结果存入 SecurityContext，以便于 Spring Security 进行授权：



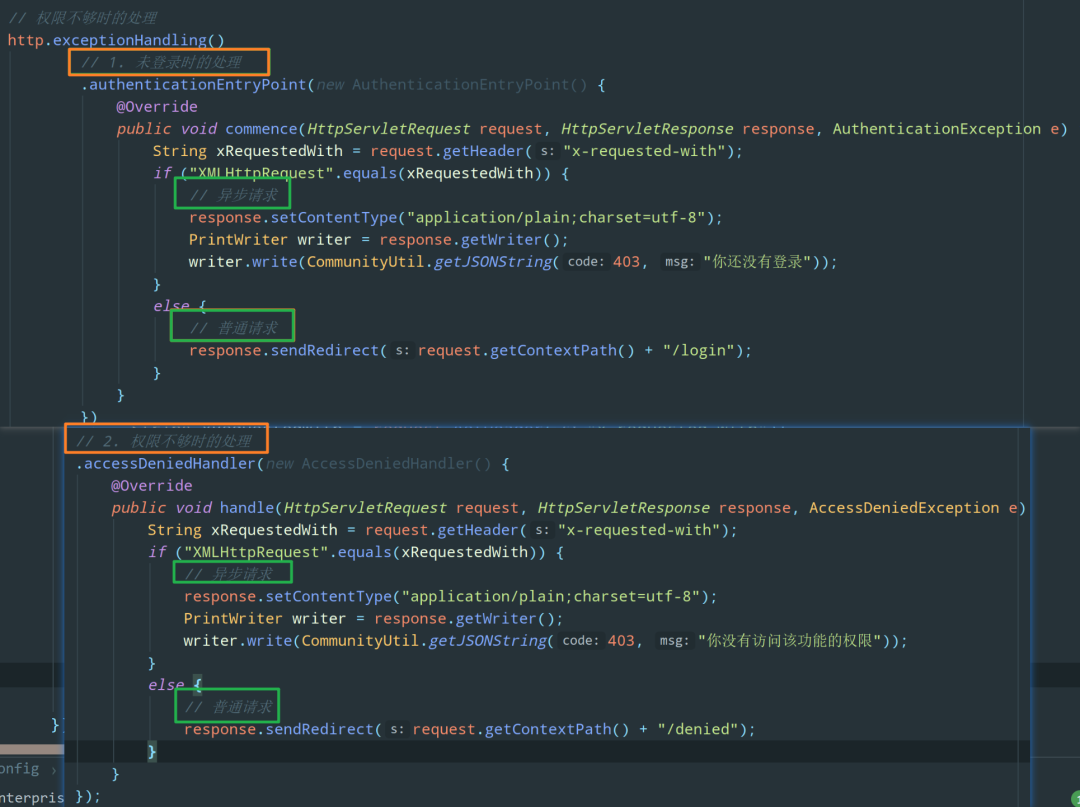
getAuthorities 就是从数据库中获取某个用户的权限（用户的权限/类型 type 是存在数据库表中的）



自定义这些权限拥有访问哪些路径的权力，比如：

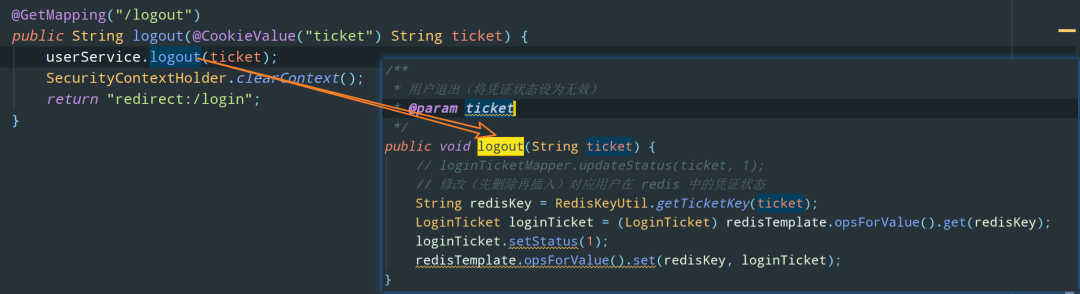


另外，还需要定义一下权限不够时需要做哪些处理，注意区分下异步请求和普通请求，对于异步请求我们返回一个 JSON 字符串，对于普通请求我们直接返回错误界面即可：



**登出**

Spring Security 底层会默认拦截 /logout 请求，进行退出处理，由于退出的逻辑我们也自己实现了（将该用户的 LoginTicket 状态设置为无效）：



所以我们赋予 Spring Security 一个根本不存在的退出路径，使得程序能够执行到我们自己编写的退出代码：





* 博主小硕在读，深耕 Java，目前在维护一个教程类仓库 CS-Wiki「Gitee 官方推荐项目，现已 1.5k+ star，仓库地址：https://gitee.com/veal98/CS-Wiki」，公众号上的文章也会在此同步更新，欢迎各位前来交流学习。
* 准备春招秋招的小伙伴可以参考我的这个论坛项目 Echo「Gitee 官方推荐项目，现已 600+ star，仓库地址：https://gitee.com/veal98/Echo」。配套教程正在同步更新中，公众号后台回复 "Echo" 即可免费获取。
* 另外，虽然现在本号仍然很小，粉丝也没多少，不过我还是建了一个交流群『小牛肉和它的小伙伴们』，感兴趣的各位可以下方扫码加我微信回复 "进群"，我拉你进群：





公众号开通较晚，暂时没有留言功能，可以点击下方链接进行留言

点此进入留言

### 精选留言

暂无...