链滴

解刨 SOLO 认证流程

作者：someone27889

原文链接：https://ld246.com/article/1577104533996

来源网站：链滴

0.要搞个东西呀所以不得不解析一下代码.一行一行走

1.开始

点击开始使用时调用

```
@RequestProcessing(value = "/login/redirect", method = HttpMethod.GET)
    public void redirectAuth(final RequestContext context) {
    // 从当前域里拿到站点地址
        String referer = context.param("referer");
        if (StringUtils.isBlank(referer)) {
            referer = Latkes.getServePath();
        }
    // 生成一个16位随机的东西然后跟入站点地址
        String state = RandomStringUtils.randomAlphanumeric(16) + referer;
    // 放到集合里
        STATES.add(state);
    // 构造 请求url 请求hacpai,回调/login/callback
        final String loginAuthURL = "https://hacpai.com/login?goto=" + Latkes.getServePath() +
"/login/callback";
    // 加入state
        final String path = loginAuthURL + "?state=" + URLs.encode(state);
        // 重定向
    context.sendRedirect(path);
    }
```

2.进入Hacpai

猜一下

1.输入用户名密码

原文链接：解刨 SOLO 认证流程

2.查询用户信息

3.截取goto路径

4.重定向 http://站点?state=state&userId=userId&userName=userName

3.返回站点

```
@RequestProcessing(value = "/login/callback", method = HttpMethod.GET)
  // `synchronized` 或许是为了极端情况下1000000人跑这个东西然后你拿着我的东西登进去了?
   public synchronized void authCallback(final RequestContext context) {
   // 首先验证state是不是本站发出的
      String state = context.param("state");
      if (!STATES.contains(state)) {
         context.sendError(400);

         return;
      }
   // 是的话就删了,免得被人用了
      STATES.remove(state);
      String referer = URLs.decode(state);
   // 从第16位截取出站点地址
      referer = StringUtils.substring(referer, 16);

      final Response response = context.getResponse();
      final Request request = context.getRequest();
   // 拿到用户ID,名称,头像,我头像贼帅
      final String openId = context.param("userId");
      final String userName = context.param(User.USER_NAME);
      final String userAvatar = context.param("avatar");
   // 看看是不是有这个用户啊
      JSONObject user = userQueryService.getUserByGitHubId(openId);
   // 如果没有
      if (null == user) {
      // 是不是没初始化博客呢还,没初始化就初始化
         if (!initService.isInited()) {
            final JSONObject initReq = new JSONObject();
            initReq.put(User.USER_NAME, userName);
            initReq.put(UserExt.USER_AVATAR, userAvatar);
            initReq.put(UserExt.USER_B3_KEY, openId);
            initReq.put(UserExt.USER_GITHUB_ID, openId);
            initService.init(initReq);
         } else {
            user = userQueryService.getUserByName(userName);
            if (null == user) {
               final JSONObject addUserReq = new JSONObject();
               addUserReq.put(User.USER_NAME, userName);
               addUserReq.put(UserExt.USER_AVATAR, userAvatar);
               addUserReq.put(User.USER_ROLE, Role.VISITOR_ROLE);
               addUserReq.put(UserExt.USER_GITHUB_ID, openId);
               addUserReq.put(UserExt.USER_B3_KEY, openId);
               try {
         // 这我估计是加到用户列表里去了,没往下看了
                  userMgmtService.addUser(addUserReq);
               } catch (final Exception e) {
```

```
                    LOGGER.log(Level.ERROR, "Registers via oauth failed", e);
                    context.sendError(500);

                    return;
                }
            } else {
                user.put(UserExt.USER_GITHUB_ID, openId);
                user.put(User.USER_NAME, userName);
                user.put(UserExt.USER_AVATAR, userAvatar);
                try {
                    userMgmtService.updateUser(user);
                } catch (final Exception e) {
                    LOGGER.log(Level.ERROR, "Updates user GitHub id failed", e);
                    context.sendError(500);

                    return;
                }
            }
        } else {
            user.put(User.USER_NAME, userName);
            user.put(UserExt.USER_AVATAR, userAvatar);
            try {
                userMgmtService.updateUser(user);
            } catch (final Exception e) {
                LOGGER.log(Level.ERROR, "Updates user name failed", e);
                context.sendError(500);

                return;
            }
        }

        user = userQueryService.getUserByName(userName);
        if (null == user) {
            LOGGER.log(Level.WARN, "Can't get user by name [" + userName + "]");
            context.sendError(404);

            return;
        }
// 如果通过了校验,那就登录
        Solos.login(user, response);
        context.sendRedirect(referer);
        LOGGER.log(Level.INFO, "Logged in [name={0}, remoteAddr={1}] with oauth", userName,
Requests.getRemoteAddr(request));
    }
```

4.造Cookie

```
public static void login(final JSONObject user, final Response response) {
        try {
    // 获取用户ID,如果没有就用Kes.OBJECT_ID:oId
            final String userId = user.optString(Keys.OBJECT_ID);
            final JSONObject cookieJSONObject = new JSONObject();
     // map放userId进oid建
```

```
            cookieJSONObject.put(Keys.OBJECT_ID, userId);
            final String b3Key = user.optString(UserExt.USER_B3_KEY);
    // 生成一个8位的带数字和字母的随机字符串
            final String random = RandomStringUtils.randomAlphanumeric(8);
    // emmm,Token b3key+随机字符串,存入MAP(原先是有密码混入的呦)
            cookieJSONObject.put(Keys.TOKEN, b3Key + ":" + random);
    // AES 加密 JSON的toString,key为lake配置文件中的cookieSecret,如果没有配置,那就是8位随机
字+字母字符串
            final String cookieValue = Crypts.encryptByAES(cookieJSONObject.toString(), COOKIE
SECRET);
// cookie: io.netty.handler.codec.http.cookie.Cookie cookie;
// CookieNAME:lake配置中的cookieName如果没有那就是随机的
//cookieValue是刚才那个MapToSring混入COOKIE_SECRET然后AES加密后的
            final Cookie cookie = new Cookie(COOKIE_NAME, cookieValue);
    // Cookie时间限制
            cookie.setMaxAge(COOKIE_EXPIRY);
    // Cookie路径
            cookie.setPath("/");
      // 我不知道这是啥
            cookie.setHttpOnly(true);
    // 放入Response中
            response.addCookie(cookie);
        } catch (final Exception e) {
            LOGGER.log(Level.WARN, "Can not write cookie", e);
        }
    }
```

5.然后重定向到主页