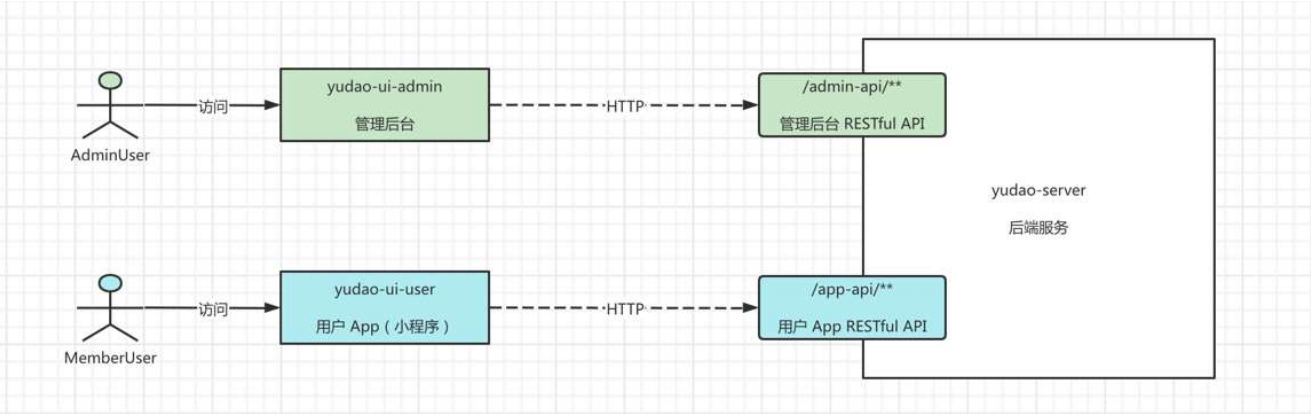


🍌 用户体系

系统提供了 2 种类型的用户，分别满足对应的管理后台、用户 App 场景。

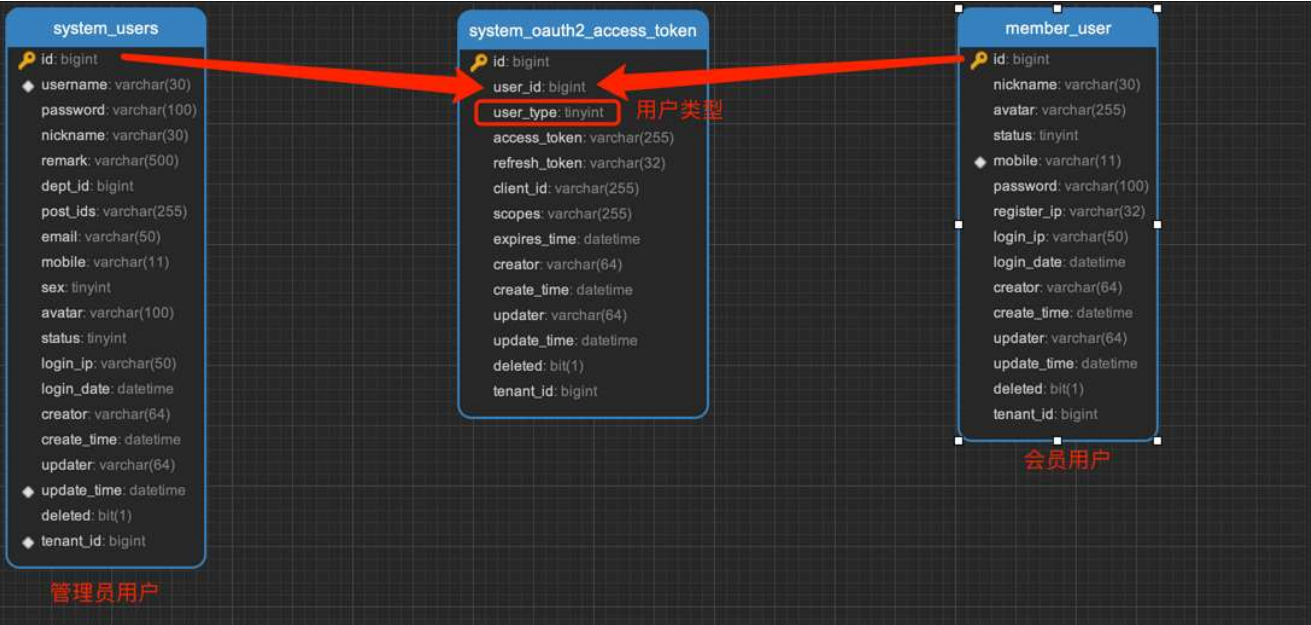


- AdminUser 管理员用户，前端访问 [yudao-ui-admin-vue3](#) 管理后台，后端访问 [/admin-api/**](#) RESTful API 接口。
- MemberUser 会员用户，前端访问 [yudao-ui-user](#) 用户 App，后端访问 [/app-api/**](#) RESTful API 接口。

虽然是不同类型的用户，他们访问 RESTful API 接口时，都通过 Token 认证机制，具体可见《开发指南 —— 功能权限》。

1. 表结构

2 种类型的时候，采用不同数据库的表进行存储，管理员用户对应 [system_users](#) 表，会员用户对应 [member_user](#) 表。如下图所示：



为什么不使用统一的用户表？



确实可以采用这样的方案，新增 `type` 字段区分用户类型。不同用户类型的信息字段，例如说上图的 `dept_id`、`post_ids` 等等，可以增加拓展表，或者就干脆“冗余”在用户表中。

不过实际项目中，不同类型的用户往往是不同的团队维护，并且这也是绝大多团队的实践，所以我们采用了多个用户表的方案。

如果表需要关联多种类型的用户，例如说上述的 `system_oauth2_access_token` 访问令牌表，可以通过 `user_type` 字段进行区分。并且 `user_type` 对应 `UserTypeEnum` 全局枚举，代码如下：

```
/**
 * 全局用户类型枚举
 */
@AllArgsConstructor
@Getter
public enum UserTypeEnum implements IntArrayValuable {

    MEMBER( value: 1, name: "会员"), // 面向 c 端，普通用户
    ADMIN( value: 2, name: "管理员"); // 面向 b 端，管理后台
```

2. 如何获取当前登录的用户？

使用 `SecurityFrameworkUtils` 提供的如下方法，可以获得当前登录用户的信息：

2.1 获取当前用户信息

```
public static LoginUser getLoginUser()
```

2.2 获取当前用户编号（最常用）

```
public static Long getLoginUserId()
```

2.3 获取当前用户昵称

```
public static LoginUser getLoginUserNickname()
```

注意，仅适合 `AdminUser` 管理员用户！

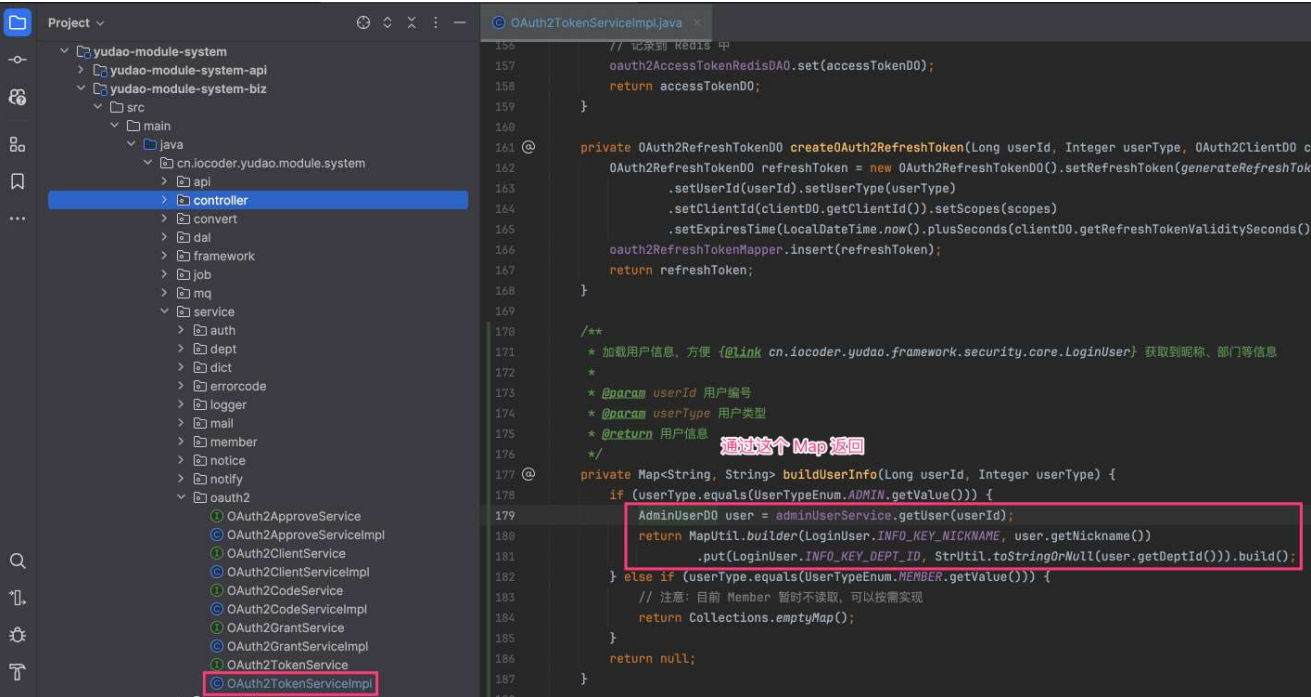
2.4 获取当前用户部门

```
public static Long getLoginUserDeptId()
```

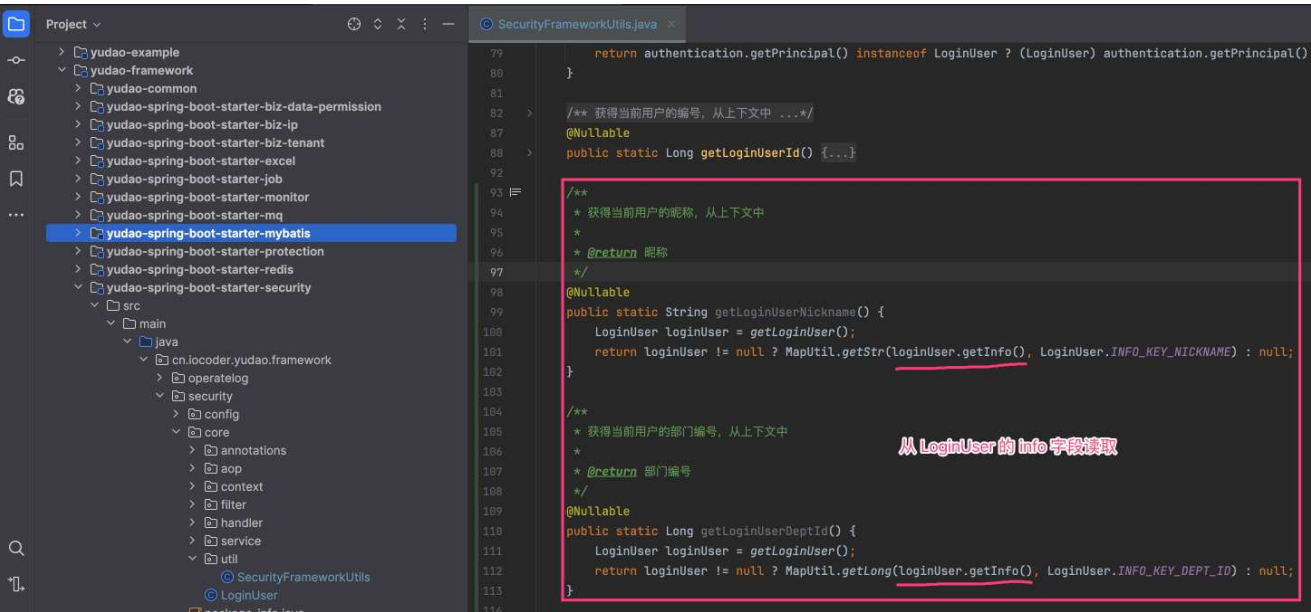
注意，仅适合 AdminUser 管理员用户！

2.5 获取更多信息

① 在 OAuth2TokenServiceImpl 的 #buildUserInfo(...) 方法中，补充读取更多的用户信息，例如说 mobile 、 sex 等等。如下图所示：



② 在 SecurityFrameworkUtils 新增对应的 getXXX() 静态方法，参考如下图所示：



3. 账号密码登录

3.1 管理后台的实现

使用 `username` 账号 + `password` 密码进行登录，由 `AuthController` 提供 `/admin-api/system/auth/login` 接口。代码如下：

```
@PostMapping("/login")
@Operation(summary = "使用账号密码登录")
public CommonResult<AuthLoginRespVO> login(@RequestBody @Valid AuthLoginReqVO reqVO,
    String token = authService.login(reqVO, getClientIP(), getUserAgent());
    // 返回结果
    return success(AuthLoginRespVO.builder().token(token).build());
}
```

如何关闭验证码？

参见《后端手册——验证码》文档。

3.2 用户 App 的实现

使用 `mobile` 手机 + `password` 密码进行登录，由 `AppAuthController` 提供 `/app-api/member/auth/login` 接口。代码如下：

```
@PostMapping("/login")
@Operation(summary = "使用手机 + 密码登录")
public CommonResult<AppAuthLoginRespVO> login(@RequestBody @Valid AppAuthLoginReqVO reqVO,
    String token = authService.login(reqVO, getClientIP(), getUserAgent());
    // 返回结果
    return success(AppAuthLoginRespVO.builder().token(token).build());
}
```

4. 手机验证码登录

4.1 管理后台的实现

① 使用 `mobile` 手机号获得验证码，由 `AuthController` 提供 `/admin-api/system/auth/send-sms-code` 接口。代码如下：

```
@PostMapping("/send-sms-code")
@Operation(summary = "发送手机验证码")
public CommonResult<Boolean> sendSmsCode(@RequestBody @Valid AuthSendSmsReqVO reqVO) {
    authService.sendSmsCode(getLoginUserId(), reqVO);
    return success(true);
}
```

② 使用 `mobile` 手机 + `code` 验证码进行登录，由 `AppAuthController` 提供 `/admin-api/system/auth/sms-login` 接口。代码如下：

```
@PostMapping("/sms-login")
@Operation(summary = "使用短信验证码登录")
public CommonResult<AuthLoginRespVO> smsLogin(@RequestBody @Valid AuthSmsLoginReqVO reqVO) {
    String token = authService.smsLogin(reqVO, getClientIP(), getUserAgent());
    // 返回结果
    return success(AuthLoginRespVO.builder().token(token).build());
}
```

4.2 用户 App 的实现

① 使用 `mobile` 手机号获得验证码，由 `AppAuthController` 提供 `/app-api/member/auth/send-sms-code` 接口。代码如下：

```
@PostMapping("/send-sms-code")
@Operation(summary = "发送手机验证码")
public CommonResult<Boolean> sendSmsCode(@RequestBody @Valid AppAuthSendSmsReqVO reqVO) {
    authService.sendSmsCode(getLoginUserId(), reqVO);
    return success(true);
}
```

② 使用 `mobile` 手机 + `code` 验证码进行登录，由 `AppAuthController` 提供 `/app-api/member/auth/sms-login` 接口。代码如下：

```
@PostMapping("/sms-login")
@Operation(summary = "使用手机 + 验证码登录")
```

```
public CommonResult<AppAuthLoginRespVO> smsLogin(@RequestBody @Valid AppAuthSmsl
    String token = authService.smsLogin(reqVO, getClientIP(), getUserAgent());
    // 返回结果
    return success(AppAuthLoginRespVO.builder().token(token).build());
}
```

如果用户未注册，会自动使用手机号进行注册会员用户。所以，`/app-api/member/user/sms-login` 接口也提供了用户注册的功能。

5. 三方登录

详细参见《开发指南——三方登录》文章。

5.1 管理后台的实现

① 跳转第三方平台，来获得三方授权码，由 `AuthController` 提供 `/admin-api/system/auth/social-auth-redirect` 接口。代码如下：

```
@GetMapping("/social-auth-redirect")
@Operation(summary = "社交授权的跳转")
@Parameters({
    @Parameter(name = "type", description = "社交类型", required = true),
    @Parameter(name = "redirectUri", description = "回调路径")
})
public CommonResult<String> socialAuthRedirect(@RequestParam("type") Integer typ
    @RequestParam("redirectUri") Str
    return CommonResult.success(socialUserService.getAuthorizeUrl(type, redirect
}
```

② 使用 `code` 三方授权码进行快登录，由 `AuthController` 提供 `/admin-api/system/auth/social-login` 接口。代码如下：

```
@PostMapping("/social-login")
@Operation(summary = "社交快捷登录，使用 code 授权码")
public CommonResult<AuthLoginRespVO> socialQuickLogin(@RequestBody @Valid AuthSc
    String token = authService.socialLogin(reqVO, getClientIP(), getUserAgent())
    // 返回结果
    return success(AuthLoginRespVO.builder().token(token).build());
}
```

③ 使用 `socialCode` 三方授权码 + `username` + `password` 进行绑定登录，直接使用 `/admin-api/system/auth/login` 账号密码登录的接口，区别在于额外带上 `socialType` +

`socialCode` + `socialState` 参数。

5.2 用户 App 的实现

① 跳转第三方平台，来获得三方授权码，由 `AppAuthController` 提供 `/app-api/member/auth/social-auth-redirect` 接口。代码如下：

```
@GetMapping("/social-auth-redirect")
@Operation(summary = "社交授权的跳转")
@Parameters({
    @Parameter(name = "type", description = "社交类型", required = true),
    @Parameter(name = "redirectUri", description = "回调路径")
})
public CommonResult<String> socialAuthRedirect(@RequestParam("type") Integer type
    @RequestParam("redirectUri") String redirectUri) {
    return CommonResult.success(socialUserService.getAuthorizeUrl(type, redirectUri));
}
```

② 使用 `code` 三方授权码进行快登录，由 `AppAuthController` 提供 `/app-api/member/auth/social-login` 接口。代码如下：

```
@PostMapping("/social-login")
@Operation(summary = "社交快捷登录，使用 code 授权码")
public CommonResult<AppAuthLoginRespVO> socialQuickLogin(@RequestBody @Valid AuthLoginReqVO reqVO) {
    String token = authService.socialLogin(reqVO, getClientIP(), getUserAgent());
    // 返回结果
    return success(AuthLoginRespVO.builder().token(token).build());
}
```

③ 使用 `socialCode` 三方授权码 + `username` + `password` 进行绑定登录，直接使用 `/app-api/system/auth/login` 手机验证码登录的接口，区别在于额外带上 `socialType` + `socialCode` + `socialState` 参数。

④ 【微信小程序特有】使用 `phoneCode` + `loginCode` 实现获取手机号并一键登录，由 `AppAuthController` 提供 `/app-api/member/auth/weixin-mini-app-login` 接口。代码如下：

```
@PostMapping("/weixin-mini-app-login")
@Operation(summary = "微信小程序的一键登录")
public CommonResult<AppAuthLoginRespVO> weixinMiniAppLogin(@RequestBody @Valid AuthLoginReqVO reqVO) {
    return success(authService.weixinMiniAppLogin(reqVO));
}
```

```
}
```

6. 注册

6.1 管理后台的实现

管理后台目前又支持用户注册，但是通过在「系统管理」>「用户管理」菜单，进行添加用户，由 `UserController` 提供 `/admin-api/system/user/create` 接口。代码如下：

```
@PostMapping("/create")
@Operation(summary = "新增用户")
@PreAuthorize("@ss.hasPermission('system:user:create')")
public CommonResult<Long> createUser(@Valid @RequestBody UserCreateReqVO reqVO) {
    Long id = userService.createUser(reqVO);
    return success(id);
}
```

6.2 用户 App 的实现

手机验证码登录时，如果用户未注册，会自动使用手机号进行注册会员用户。所以，`/app-api/system/user/sms-login` 接口也提供了用户注册的功能。

7. 用户登出

用户登出的功能，统一使用 Spring Security 框架，通过删除用户 Token 的方式来实现。代码如下：

YudaoWebSecurityConfigurerAdapter.java

184

@Override

185

protected void configure(HttpSecurity httpSecurity) throws Exception {

186

// 登出

187

httpSecurity

188

// 开启跨域

189

.cors().and() HttpSecurity

190

// CSRF 禁用, 因为不使用 Session

191

.csrf().disable()

192

// 基于 token 机制, 所以不需要 Session

193

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS).and()

194

.headers().frameOptions().disable().and()

195

// 一堆自定义的 Spring Security 处理器

196

.exceptionHandling().authenticationEntryPoint(authenticationEntryPoint).exceptionHandlingConfigurer<HttpSecurity>

197

.accessDeniedHandler(accessDeniedHandler).and() HttpSecurity

198

// 登出地址的配置

199

.logout().logoutSuccessHandler(logoutSuccessHandler).logoutRequestMatcher(request -> // 匹配多种用户类型的登出

200

StrUtil.equalsAny(request.getRequestURI(), buildAdminApi(url: "/system/logout"), 管理员用户的登出

201

buildAppApi(url: "/member/logout"))); 会员用户的登出

202

}

LogoutSuccessHandlerImpl.java

17

/** 自定义退出处理器 ...*/

22

@AllArgsConstructor

23

public class LogoutSuccessHandlerImpl implements LogoutSuccessHandler {

24

25

private final SecurityProperties securityProperties;

26

27

private final MultiUserDetailsAuthenticationProvider authenticationProvider;

28

29

@Override

30

public void onLogoutSuccess(HttpServletRequest request, HttpServletResponse response, Authentication authentication) {

31

// 执行退出

32

String token = SecurityFrameworkUtils.obtainAuthorization(request, securityProperties.getTokenHeader());

33

if (StrUtil.isNotBlank(token)) {

34

authenticationProvider.logout(request, token); 删除用户 Token 来作废

35

}

36

// 返回成功

37

ServletUtils.writeJSON(response, CommonResult.success(null));

38

}

39

40

}

差别在于使用的 API 接口不同，管理员用户使用 `/admin-api/system/logout` ，会员用户使用 `/app-api/member/logout` 。