

[🏠 / 开发指南 / 后端手册](#)[👤 芋道源码](#) [📅 2022-03-07](#)

🟡 SaaS 多租户【字段隔离】

本章节，将介绍多租户的基础知识、以及怎样使用多租户的功能。

相关的视频教程：

- [01、如何实现多租户的 DB 封装？](#)
- [02、如何实现多租户的 Redis 封装？](#)
- [03、如何实现多租户的 Web 与 Security 封装？](#)
- [04、如何实现多租户的 Job 封装？](#)
- [05、如何实现多租户的 MQ 与 Async 封装？](#)
- [06、如何实现多租户的 AOP 与 Util 封装？](#)
- [07、如何实现多租户的管理？](#)
- [08、如何实现多租户的套餐？](#)

1. 多租户是什么？

多租户，简单来说是指**一个**业务系统，可以为**多个**组织服务，并且组织之间的数据是**隔离**的。

例如说，在服务上部署了一个 [yudao-cloud](#) 系统，可以支持多个不同的公司使用。这里的**一个公司就是一个租户**，每个用户必然属于某个租户。因此，用户也只能看见自己租户下面的内容，其它租户的内容对他是不可见的。

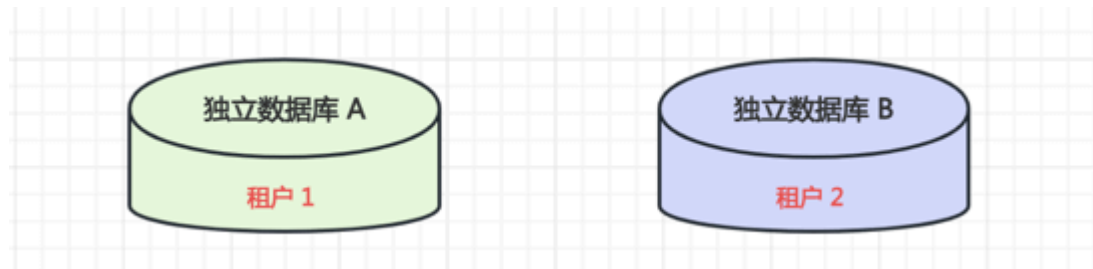
2. 数据隔离方案

多租户的数据隔离方案，可以分成分成三种：

1. DATASOURCE 模式：独立数据库
2. SCHEMA 模式：共享数据库，独立 Schema
3. COLUMN 模式：共享数据库，共享 Schema，共享数据表

2.1 DATASOURCE 模式

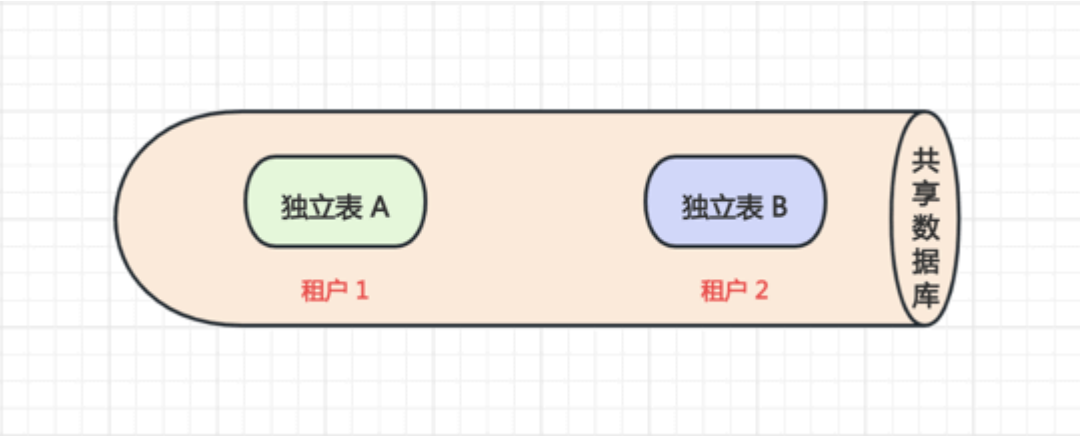
一个租户一个数据库，这种方案的用户数据隔离级别最高，安全性最好，但成本也高。



- 优点：为不同的租户提供独立的数据库，有助于简化数据模型的扩展设计，满足不同租户的独特需求；如果出现故障，恢复数据比较简单。
- 缺点：增大了数据库的安装数量，随之带来维护成本和购置成本的增加。

2.2 SCHEMA 模式

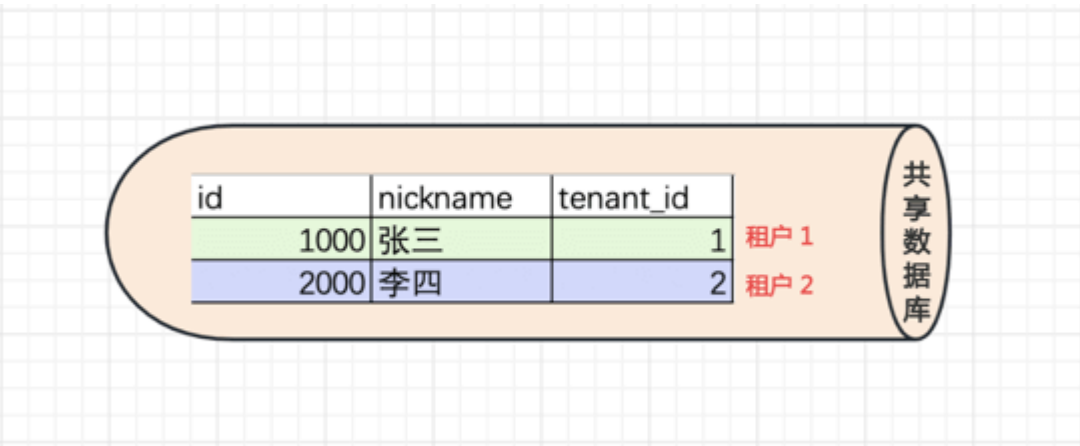
多个或所有租户共享数据库，但一个租户一个表。



- 优点：为安全性要求较高的租户提供了一定程度的逻辑数据隔离，并不是完全隔离；每个数据库可以支持更多的租户数量。
- 缺点：如果出现故障，数据恢复比较困难，因为恢复数据库将牵扯到其他租户的数据；如果需要跨租户统计数据，存在一定困难。

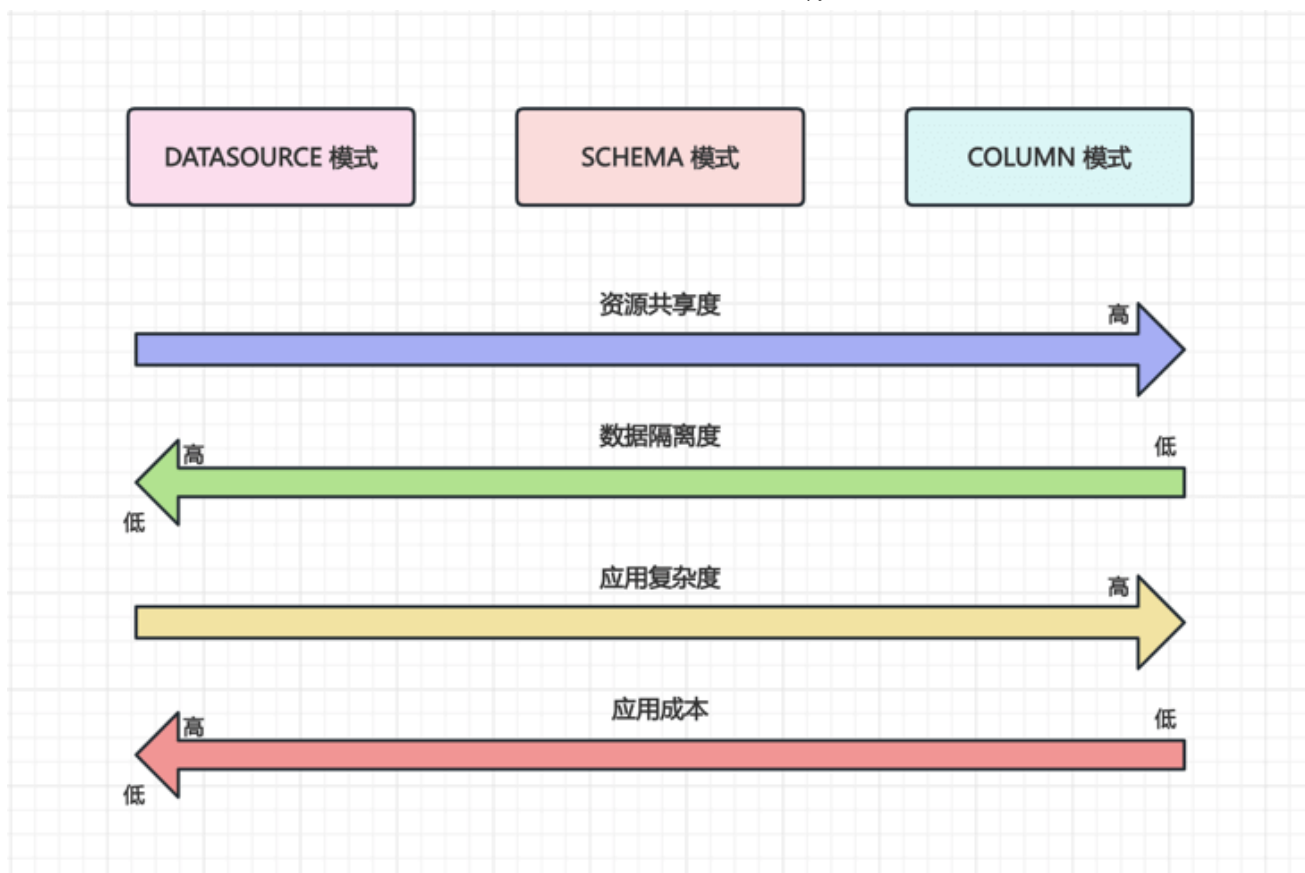
2.3 COLUMN 模式

共享数据库，共享数据架构。租户共享同一个数据库、同一个表，但在表中通过 `tenant_id` 字段区分租户的数据。这是共享程度最高、隔离级别最低的模式。



- 优点：维护和购置成本最低，允许每个数据库支持的租户数量最多。
- 缺点：隔离级别最低，安全性最低，需要在设计开发时加大对安全的开发量；数据备份和恢复最困难，需要逐表逐条备份和还原。

2.4 方案选择



- 一般情况下，可以考虑采用 COLUMN 模式，开发、运维简单，以最少的服务器为最多的租户提供服务。
- 租户规模比较大，或者一些租户对安全性要求较高，可以考虑采用 DATASOURCE 模式，当然它也相对复杂的多。
- 不推荐采用 SCHEMA 模式，因为它的优点并不明显，而且它的缺点也很明显，同时对复杂 SQL 支持一般。

提问：项目支持哪些模式？

目前支持最主流的 DATASOURCE 和 COLUMN 两种模式。而 SCHEMA 模式不推荐使用，所以暂时不考虑实现。

考虑到让大家更好的理解 DATASOURCE 和 COLUMN 模式，拆成了两篇文章：

- 《SaaS 多租户【字段隔离】》：讲解 COLUMN 模式
- 《SaaS 多租户【数据库隔离】》：讲解 DATASOURCE 模式

3. 多租户的开关

系统有两个配置项，设置为 `true` 时开启多租户，设置为 `false` 时关闭多租户。

注意，两者需要保持一致，否则会报错！

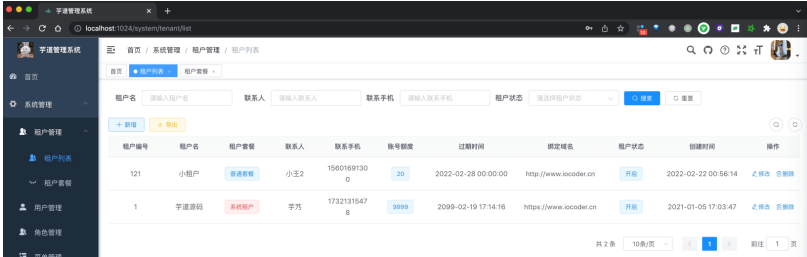
| 配置项 | 说明 | 配置文件 |
|-----------------------|------|--|
| yudao.server.tenant | 后端开关 |  |
| VUE_APP_TENANT_ENABLE | 前端开关 |  |

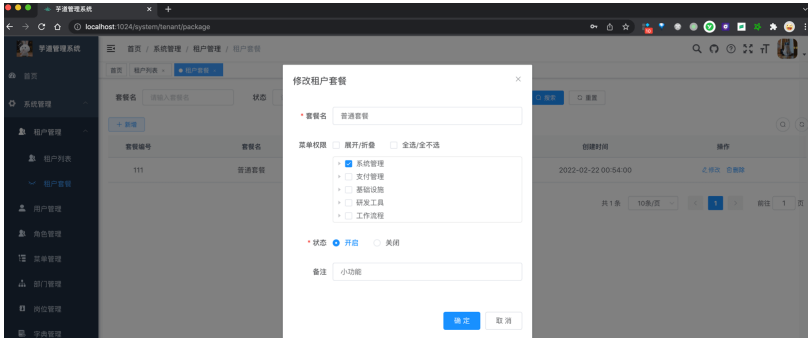
疑问：为什么要设置两个配置项？

前端登录界面需要使用到多租户的配置项，从后端加载配置项的话，体验会比较差。

3. 多租户的业务功能

多租户主要有两个业务功能：

| 业务功能 | 说明 | 界面 | 代码 |
|------|-------------------|--|----------|
| 租户管理 | 配置系统租户，创建对应的租户管理员 |  | 后端 前端 |

| 业务功能 | 说明 | 界面 | 代码 |
|------|---------------------------|--|----------|
| 租户套餐 | 配置租户套餐，自定每个租户的菜单、操作、按钮的权限 |  | 后端 前端 |

下面，我们来新增一个租户，它使用 COLUMN 模式。

① 点击 [租户套餐] 菜单，点击 [新增] 按钮，填写租户的信息。

添加租户

* 租户名

测试租户

* 租户套餐

普通套餐

* 联系人

芋道

联系手机

15601691300

* 用户名称

admin

* 用户密码

.....

* 账号额度

50

* 过期时间

2023-03-31

* 绑定域名

http://www.iocoder.cn

数据源

master

* 租户状态

开启

关闭

column 模式，选择 master 数据源即可

确定

取消

② 点击 [确认] 按钮，完成租户的创建，它会自动创建对应的租户管理员、角色等信息。

```
mysql> SELECT * FROM system_tenant;
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | name | contact_user_id | contact_name | contact_mobile | status | domain | package_id | expire_time | account_count | creator | create_time | updater | update_time |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | 芋道源码 | NULL | 新创建的租户 | 17321315478 | 0 | https://www.iocoder.cn | 0 | 2019-02-19 17:14:16 | 9999 | 1 | 2021-01-05 17:03:47 | 1 | 2022-02-23 12:15:11 | 0 |
| 121 | 小租户 | 110 | 小王2 | 15081691300 | 0 | http://www.iocoder.cn | 111 | 2022-02-28 00:00:00 | 20 | 1 | 2022-02-23 00:08:45 | 1 | 2022-02-23 00:08:45 | 0 |
| 122 | 测试租户 | 113 | 芋道 | 15081691300 | 0 | https://www.iocoder.cn | 111 | 2022-04-30 00:00:00 | 50 | 1 | 2022-03-07 21:37:58 | 1 | 2022-03-07 21:37:58 | 0 |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
3 rows in set (0.01 sec)

mysql> SELECT * FROM system_user WHERE tenant_id = 122; 查询创建的用户
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | username | password | deleted | tenant_id | nickname | remark | dept_id | post_ids | email | mobile | sex | avatar | status | login_ip | login_date | creator | create_time |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 113 | aoteman | $2a$10$1AEsjpPIIsc3DFzsoX9.L.ykydbPGkVyea0Ch.J2Sq5G0vB6GZmy | 0 | 122 | 芋道 | NULL | NULL | NULL |  | 15081691300 | 0 |  | 0 | 1 | NULL | 1 | 2022-03-07 21:37:58 | 1 |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> SELECT * FROM system_role WHERE tenant_id = 122; 查询创建的角色
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | name | code | sort | data_scope | data_scope_dept_ids | status | type | remark | creator | create_time | updater | update_time | deleted | tenant_id |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 111 | 租户管理员 | tenant_admin | 0 | 1 |  | 0 | 1 | 系统自动生成 | 1 | 2022-03-07 21:37:58 | 1 | 2022-03-07 21:37:58 | 0 | 122 |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql>
```

③ 退出系统，登录刚创建的租户。



至此，我们已经完成了租户的创建。

4. 多租户的技术组件

技术组件 [yudao-spring-boot-starter-biz-tenant](#) [🔗](#)，实现透明化的多租户能力，针对 Web、Security、DB、Redis、AOP、Job、MQ、Async 等多个层面进行封装。

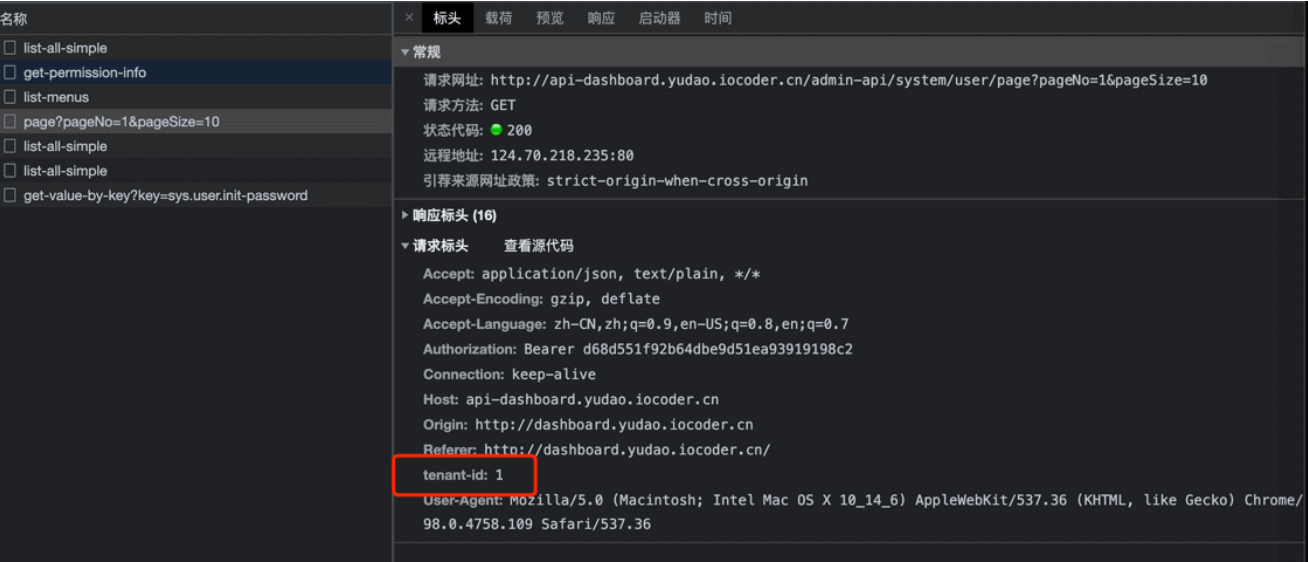
4.1 租户上下文

[TenantContextHolder](#) [🔗](#) 是租户上下文，通过 `ThreadLocal` 实现租户编号的共享与传递。通过调用 `TenantContextHolder` 的 `#getTenantId()` 静态方法，获得当前的租户编号。绝大多数情况下，并不需要。

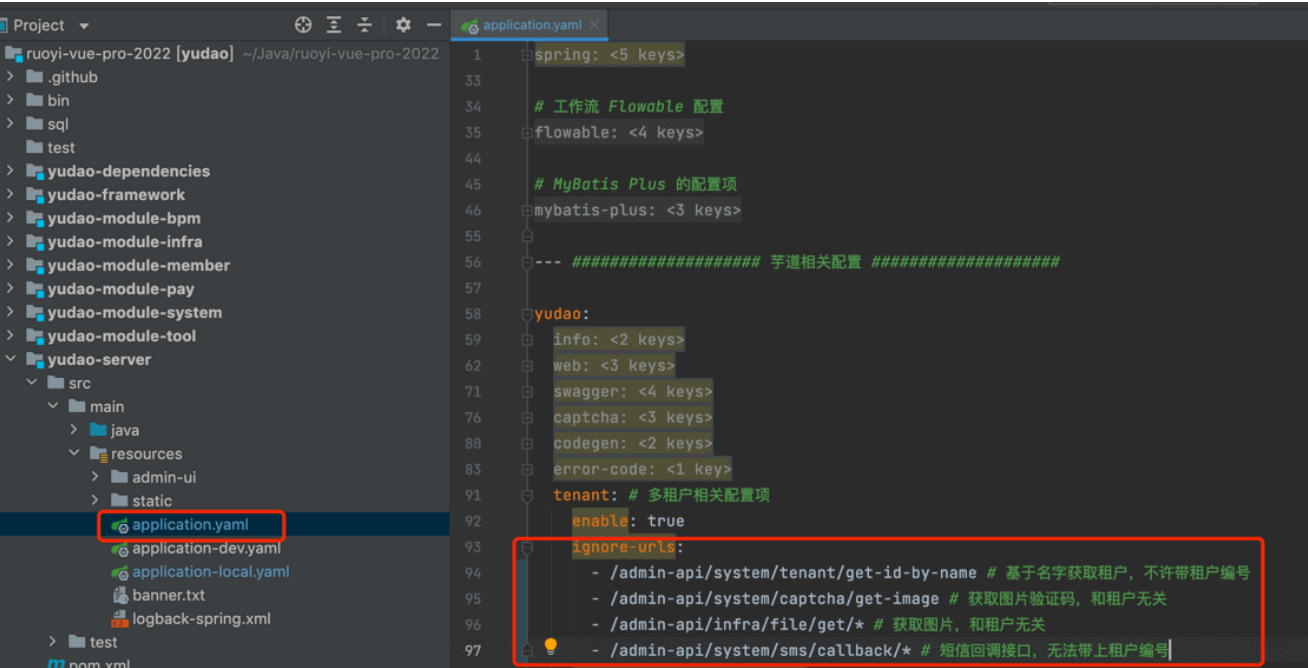
4.2 Web 层【重要】

实现可见 `web` 包。

默认情况下，前端的每个请求 Header 必须带上 `tenant-id`，值为租户编号，即 `system_tenant` 表的主键编号。



如果不带该请求头，会报“租户的请求未传递，请进行排查”错误提示。
😬 通过 `yudao.tenant.ignore-urls` 配置项，可以设置哪些 URL 无需带该请求头。例如说：



4.3 Security 层

实现可见 `security` 包。

主要是校验登录的用户，校验是否有权限访问该租户，避免越权问题。

4.4 DB 层【重要】

实现可见 [db](#) 包。

COLUMN 模式，基于 MyBatis Plus 自带的[多租户](#)功能实现。

核心：每次对数据库操作时，它会[自动拼接](#) `WHERE tenant_id = ?` 条件来进行租户的过滤，并且基本支持所有的 SQL 场景。

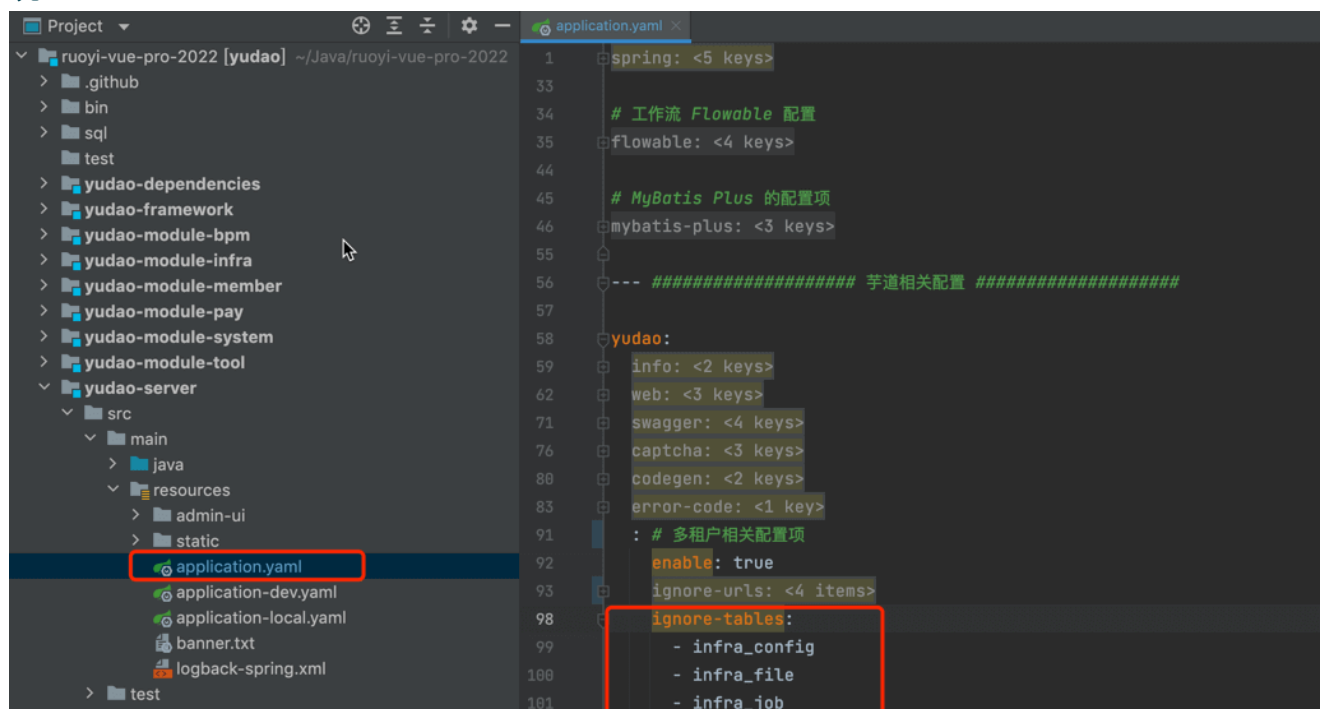
如下是具体方式：

- ① **需要**开启多租户的表，必须添加 `tenant_id` 字段。例如说 `system_users` 、`system_role` 等表。

```
CREATE TABLE `system_role` (  
  `id` bigint NOT NULL AUTO_INCREMENT COMMENT '角色ID',  
  `name` varchar(30) CHARACTER NOT NULL COMMENT '角色名称',  
  `tenant_id` bigint NOT NULL DEFAULT '0' COMMENT '租户编号',  
  PRIMARY KEY (`id`) USING BTREE  
) ENGINE=InnoDB AUTO_INCREMENT=1 COMMENT='角色信息表';
```

并且该表对应的 DO 需要使用到 `tenantId` 属性时，建议继承 `TenantBaseDO` 类。

- ② **无需**开启多租户的表，需要添加表名到 `yudao.tenant.ignore-tables` 配置项目。例如说：



如果不配置的话，MyBatis Plus 会自动拼接 `WHERE tenant_id = ?` 条件，导致报 `tenant_id` 字段不存在的错误。

4.5 Redis 层

实现可见 [redis](#) 包。

使用方式一：基于 Spring Cache + Redis【推荐】

只需要一步，在方法上添加 Spring Cache 注解，例如说 `@Cachable`、`@CachePut`、`@CacheEvict`。

具体的实现原理，可见 [TenantRedisCacheManager](#) 的源码。

注意！！！默认配置下，Spring Cache 都开启 Redis Key 的多租户隔离。如果不需要，可以将 Key 添加到 `yudao.tenant.ignore-cache` 配置项中。如下图所示：

```
application.yml x
124   license-url: https://gitee.com/zhijiantianya/ruoyi-vue-pro/blob/master/LICENSE
125   captcha:
126     enable: true # 验证码的开关, 默认为 true
127   codegen:
128     base-package: ${yudao.info.base-package}
129     db-schemas: ${spring.datasource.dynamic.datasource.master.name}
130   error-code: # 错误码相关配置项
131 >   constants-class-list: <6 items>
132   tenant: # 多租户相关配置项
133     enable: true
134     ignore-urls: <8 items>
135     ignore-tables: <42 items>
136     ignore-caches:
137       - permission_menu_ids
138       - oauth_client
139       - notify_template
140       - mail_account
141       - mail_template
142       - sms_template
```

使用方式二：基于 RedisTemplate + TenantRedisKeyDefine

暂时没有合适的封装，需要在自己 format Redis Key 的时候，手动将 `:t{tenantId}` 后缀拼接上。

这也是为什么，我推荐你使用 Spring Cache + Redis 的原因！

4.6 AOP【重要】

实现可见 [aop](#) 包。

① 声明 `@TenantIgnore` 注解在方法上，标记指定方法不进行租户的自动过滤，避免自动拼接 `WHERE tenant_id = ?` 条件等等。

例如说: `RoleServiceImpl` 的 `#initLocalCache()` 方法, 加载**所有**租户的角色到内存进行缓存, 如果不声明 `@TenantIgnore` 注解, 会导致租户的自动过滤, 只加载了某个租户的角色。

```
// RoleServiceImpl.java
public class RoleServiceImpl implements RoleService {

    @Resource
    @Lazy // 注入自己, 所以延迟加载
    private RoleService self;

    @Override
    @PostConstruct
    @TenantIgnore // 忽略自动多租户, 全局初始化缓存
    public void initLocalCache() {
        // ... 从数据库中, 加载角色
    }

    @Scheduled(fixedDelay = SCHEDULER_PERIOD, initialDelay = SCHEDULER_PERIOD)
    public void schedulePeriodicRefresh() {
        self.initLocalCache(); // <x> 通过 self 引用到 Spring 代理对象
    }
}
```

有一点要格外注意, 由于 `@TenantIgnore` 注解是基于 Spring AOP 实现, 如果是**方法内部的调用**, 避免使用 `this` 导致不生效, 可以采用上述示例的 `<x>` 处的 `self` 方式。

② 使用 `TenantUtils` 的 `#execute(Long tenantId, Runnable runnable)` 方法, 模拟指定租户(`tenantId`), 执行某段业务逻辑(`runnable`)。

例如说: 在 `TenantServiceImpl` 的 `#createTenant(...)` 方法, 在创建完租户时, 需要模拟该租户, 进行用户和角色的创建。如下图所示:

```

168 @Override
169 @Transactional(rollbackFor = Exception.class)
170 public Long createTenant(TenantCreateReqV0 createReqV0) {
171     // 校验套餐被禁用
172     TenantPackageD0 tenantPackage = tenantPackageService.validTenantPackage(createReqV0.getPackageId());
173
174     // 创建租户
175     TenantD0 tenant = TenantConvert.INSTANCE.convert(createReqV0);
176     tenantMapper.insert(tenant);
177
178     TenantUtils.execute(tenant.getId(), () -> { 模拟租户，复用逻辑
179         // 创建角色
180         Long roleId = createRole(tenantPackage);
181         // 创建用户，并分配角色
182         Long userId = createUser(roleId, createReqV0);
183         // 修改租户的管理员
184         tenantMapper.updateById(new TenantD0().setId(tenant.getId()).setContactUserId(userId));
185     });
186     // 发送刷新消息
187     TransactionSynchronizationManager.registerSynchronization(afterCommit() -> {
188         tenantProducer.sendTenantRefreshMessage();
189     });
190     return tenant.getId();
191 }
192
193 private Long createUser(Long roleId, TenantCreateReqV0 createReqV0) {
194     // 创建用户
195     Long userId = userService.createUser(TenantConvert.INSTANCE.convert02(createReqV0));
196     // 分配角色
197     permissionService.assignUserRole(userId, singleton(roleId));
198     return userId;
199 }
200
201 private Long createRole(TenantPackageD0 tenantPackage) {
202     // 创建角色
203     RoleCreateReqV0 reqV0 = new RoleCreateReqV0();
204     reqV0.setName(RoleCodeEnum.TENANT_ADMIN.getName()).setCode(RoleCodeEnum.TENANT_ADMIN.getCode())
205         .setSort(0).setRemark("系统自动生成");
206     Long roleId = roleService.createRole(reqV0, RoleTypeEnum.SYSTEM.getType());
207     // 分配权限
208     permissionService.assignRoleMenu(roleId, tenantPackage.getMenuIds());
209 }

```

4.7 Job 【重要】

实现可见 [job](#) 包。

声明 `@TenantJob` 注解在 Job 方法上，实现并行遍历每个租户，执行定时任务的逻辑。

4.8 MQ

实现可见 [mq](#) 包。

通过租户对 MQ 层面的封装，实现租户上下文，可以继续传递到 MQ 消费的逻辑中，避免丢失的问题。实现原理是：

- 发送消息时，MQ 会将租户上下文的租户编号，记录到 Message 消息头 `tenant-id` 上。
- 消费消息时，MQ 会将 Message 消息头 `tenant-id`，设置到租户上下文的租户编号。

4.9 Async

实现可见 [YudaoAsyncAutoConfiguration](#) 类。

通过使用阿里开源的 [TransmittableThreadLocal](#) 组件，实现 Spring Async 执行异步逻辑时，租户上下文可以继续传递，避免丢失的问题。

4.10 RPC

实现可见 [mq](#) 包。

RPC 使用 Feign 调用时，会自动将租户上下文的租户编号，设置到 HTTP 请求头 `tenant-id` 上。

在 Provider 服务端，会自动将 HTTP 请求头 `tenant-id`，设置到租户上下文的租户编号。

[← OAuth 2.0 \(SSO 单点登录\)](#)

[SaaS 多租户【数据库隔离】→](#)



Theme by [Vdoing](#) | Copyright © 2019-2023 芋道源码 | MIT License