芋道源码 2023-03-01

SaaS 多租户【数据库隔离】

本章节，讲解 SaaS 租户的 DATASOURCE 模式，实现数据库级别的隔离。
注意，需要前置阅读 《SaaS 多租户【字段隔离】》 文档。

0. 极速体验

① 克隆 https://gitee.com/zhijiantianya/ruoyi-vue-pro 仓库，并切换到 `feature/dev-yunai` 分支。
② 创建 `ruoyi-vue-pro-master` 、 `ruoyi-vue-pro-tenant-a` 、 `ruoyi-vue-pro-tenant-b` 三个数据库。
③ 下载 `多租户多db.zip` 并解压，将 SQL 导入到对应的数据库中。

> **友情提示：**
> 随着版本的迭代，SQL 脚本可能过期。如果碰到问题，可以在星球给我反馈下。

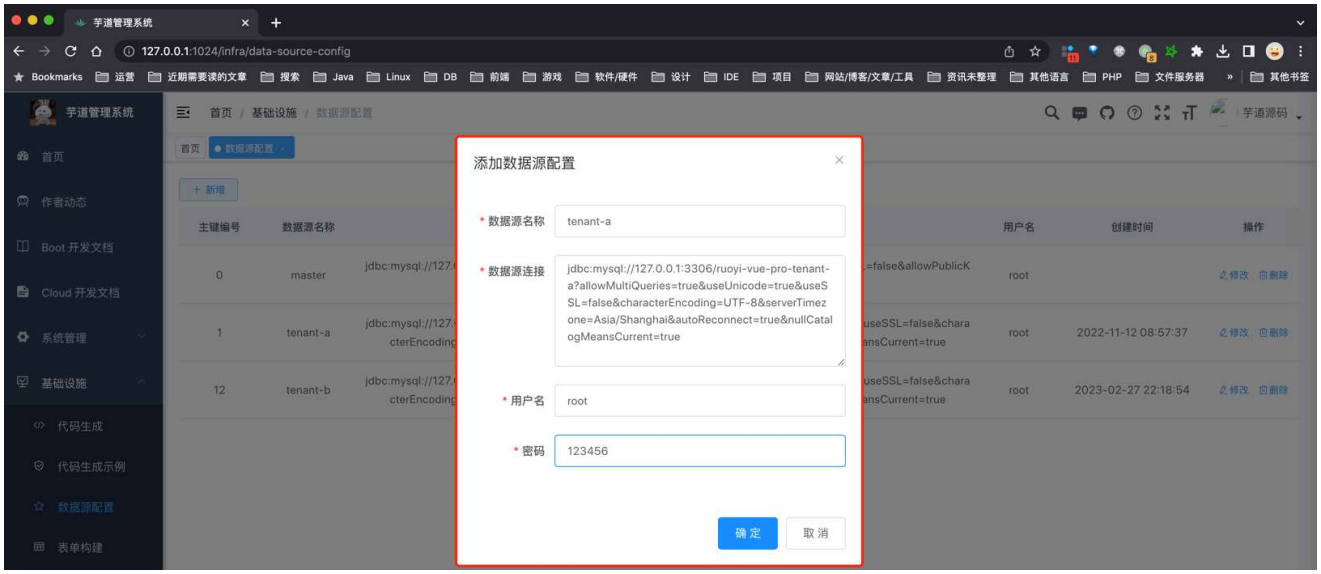
④ 启动前端和后端项目，即可愉快的体验了。

1. 实现原理

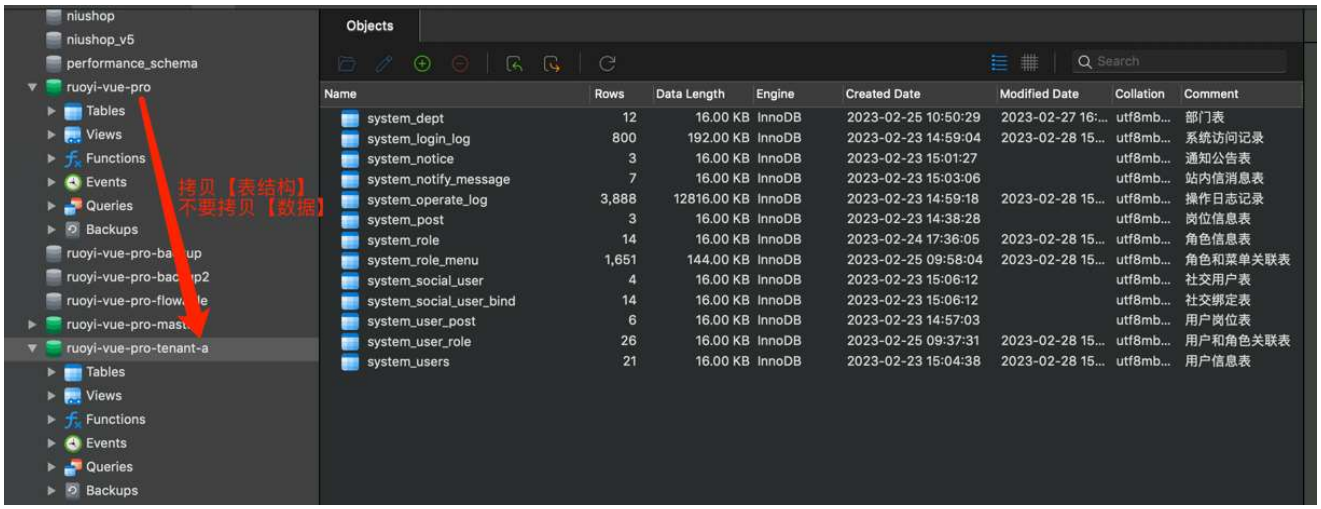
DATASOURCE 模式，基于 dynamic-datasource 进行拓展实现。
核心：每次对数据库操作时，动态切换到该租户所在的数据源，然后执行 SQL 语句。

2. 功能演示

我们来新增一个租户，使用 DATASOURCE 模式。
① 点击 [基础设施 -> 数据源配置] 菜单，点击 [新增] 按钮，新增一个名字为 `tenant-a` 数据源。



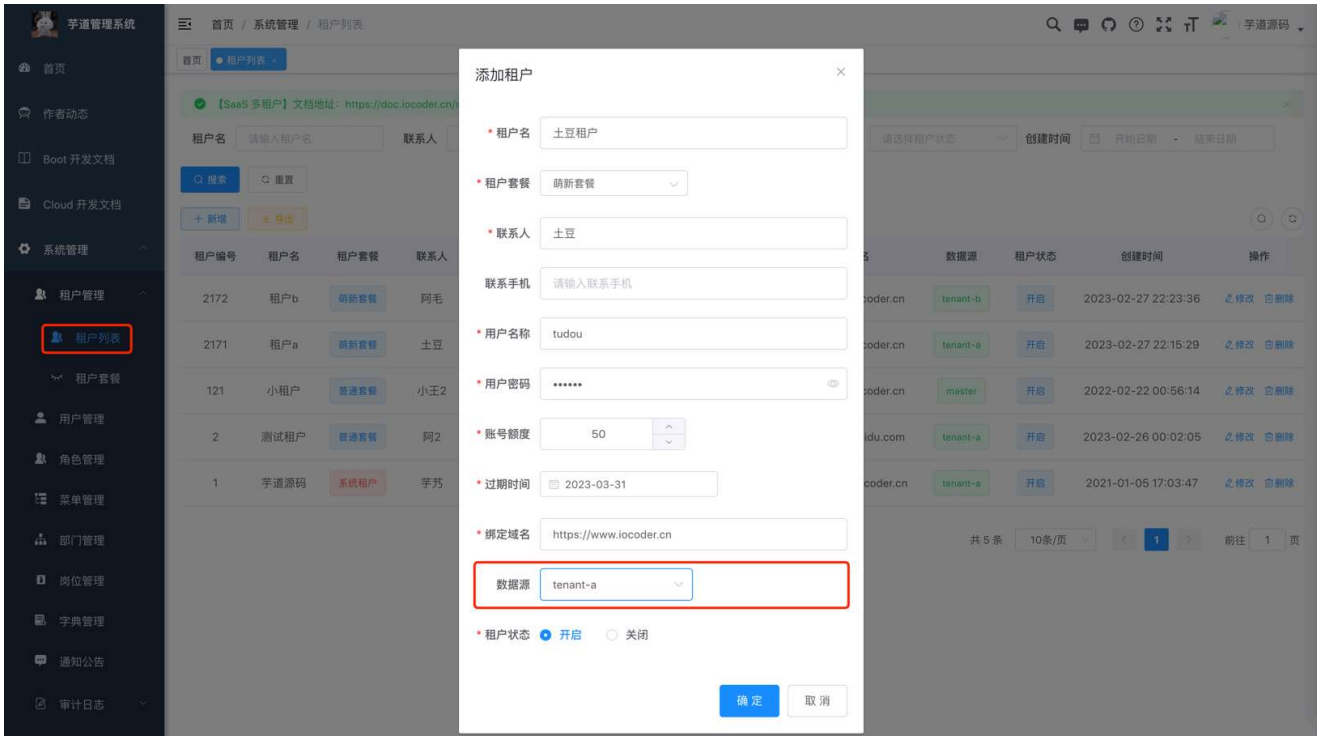
然后，手动将如下表拷贝到 `ruoyi-vue-pro` 主库中的如下表，拷贝到 `ruoyi-vue-pro-tenant-a` 库中。如下图所示：



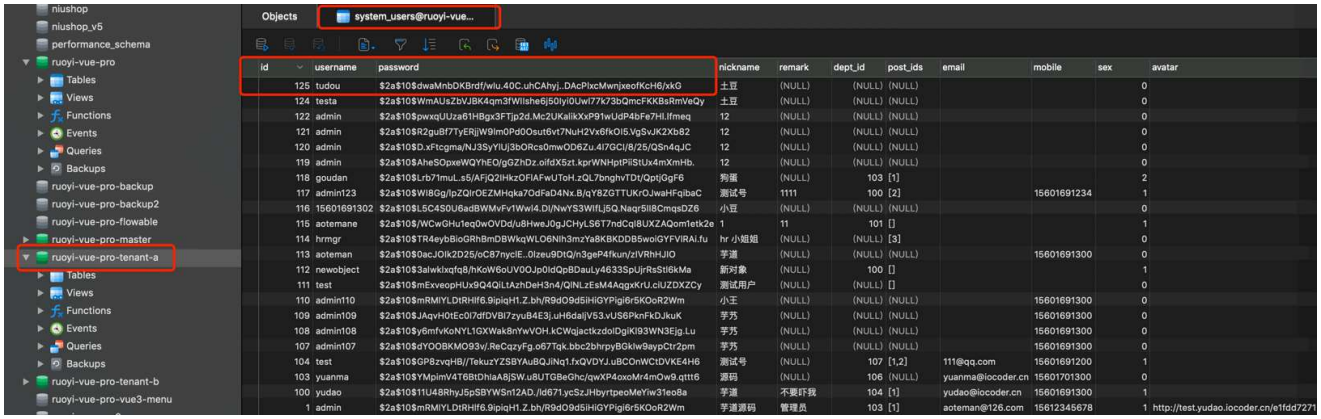

```
system_dept
system_login_log
system_notice
system_notify_message
system_operate_log
system_post
system_role
system_role_menu
system_social_user
system_social_user_bind
system_user_post
system_user_role
system_users
```

> **友情提示：**
>
> 随着版本的迭代，可能需要拷贝更多的表。如果碰到问题，可以在星球给我反馈下。

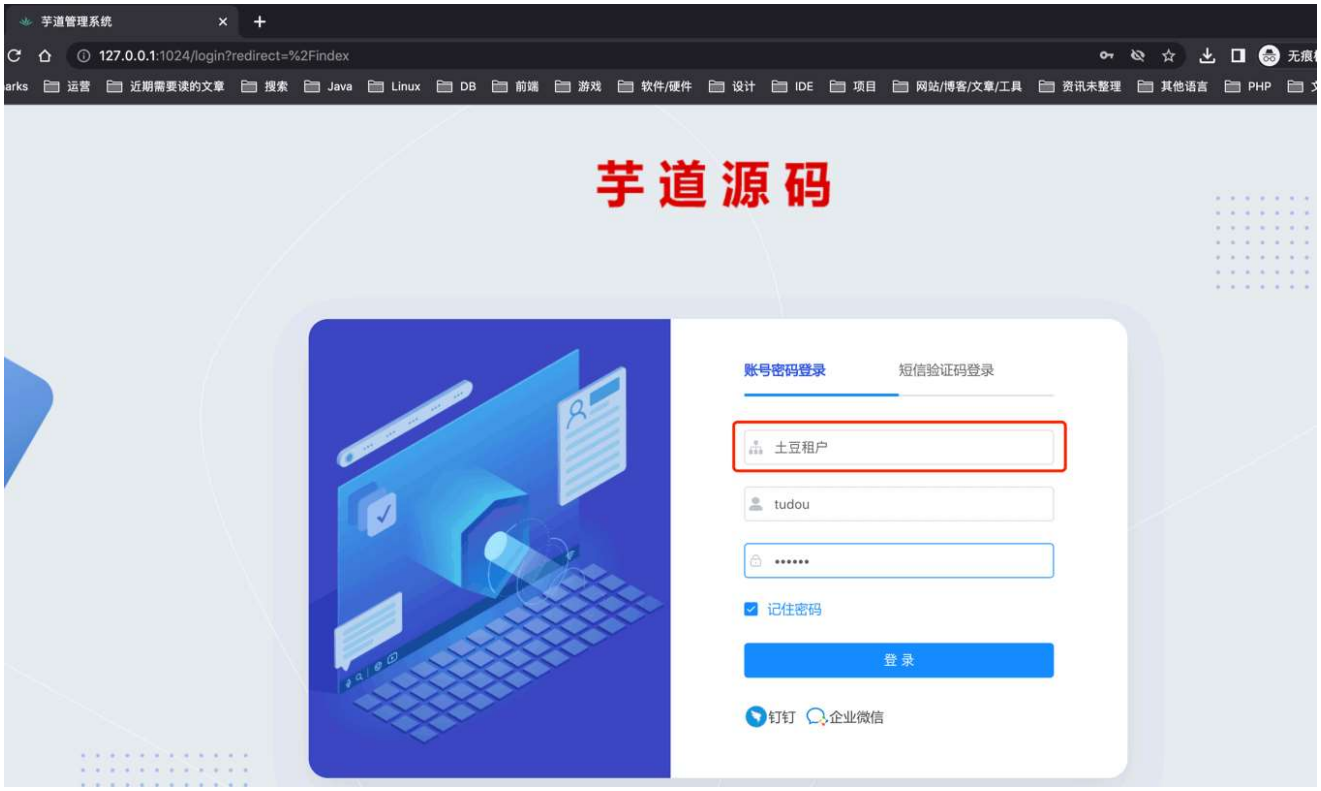
② 点击 [基础设施 -> 租户管理] 菜单，点击 [新增] 按钮，新增一个名字为 `土豆租户` 的租户，并使用 `tenant-a` 数据源。如下图所示：



此时，在 `ruoyi-vue-pro-tenant-a` 库中，可以查询到对应的租户管理员、角色等信息。如下图所示：



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。

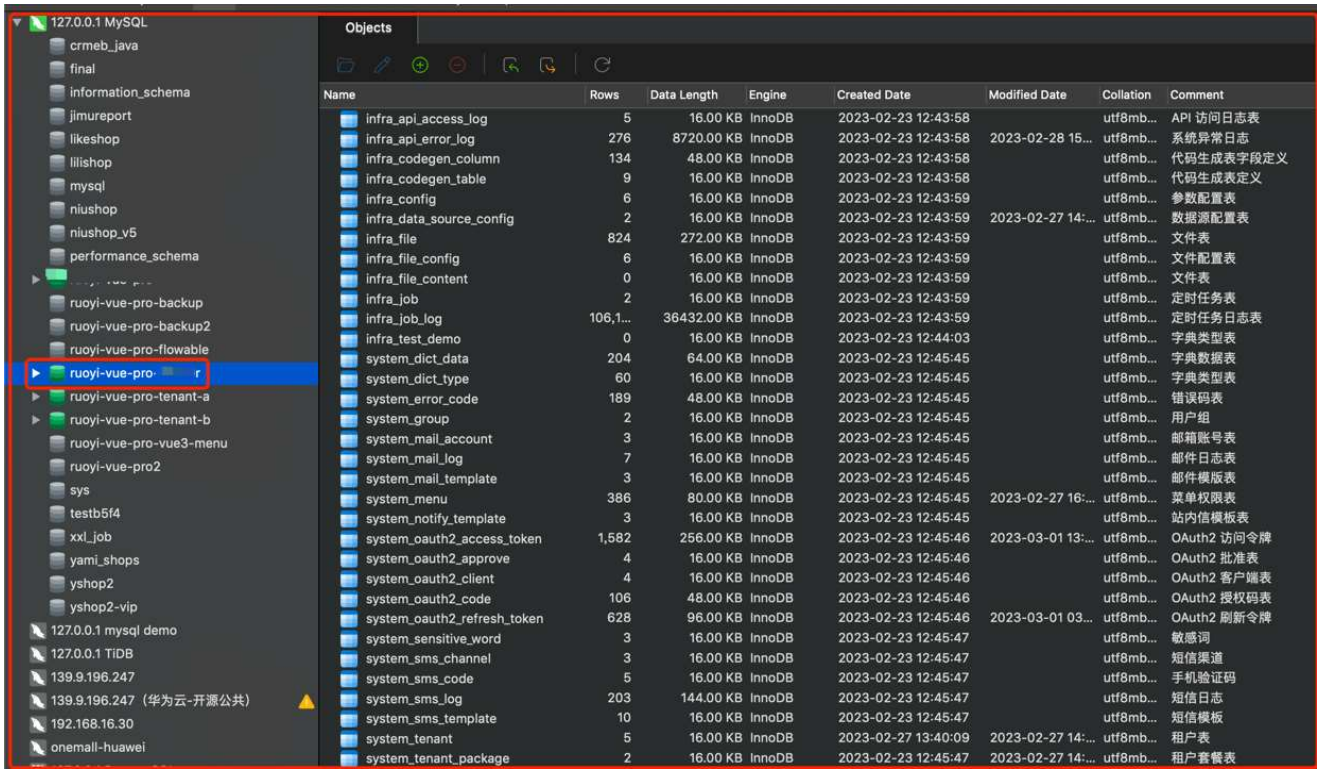
> **补充说明：**
>
> 后续在使用时，建议把拷贝到其它租户数据库的表，从 `ruoyi-vue-pro` 主库中进行删除。
>
> 目的是，主库只保留所有租户共享的全局表。例如说，菜单表、定时任表等等。

3. 创建表

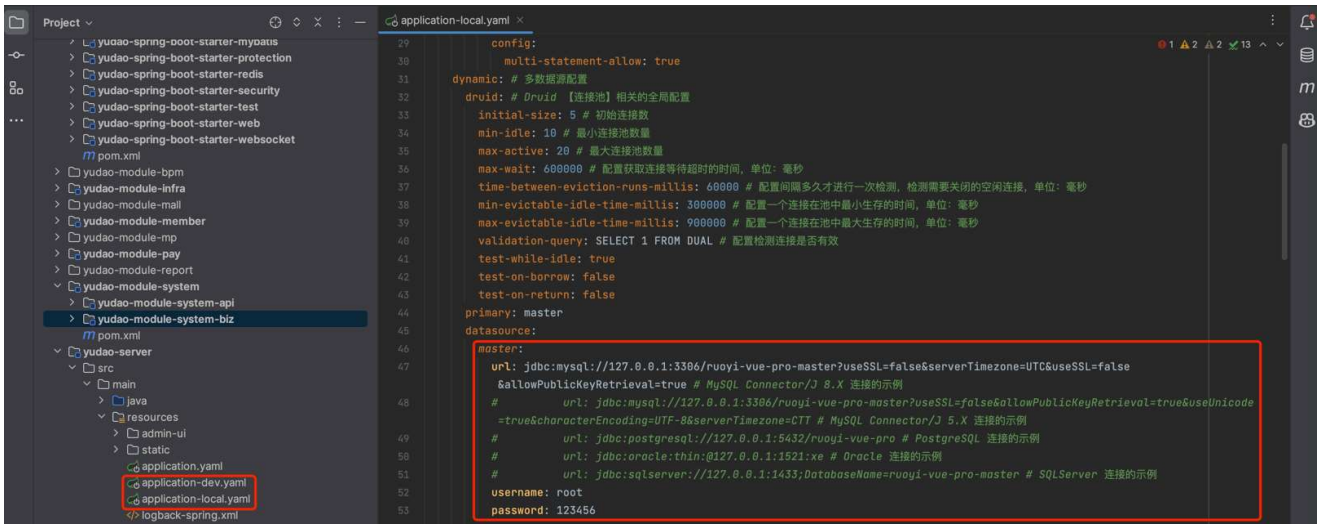
在使用 DATASOURCE 模式时，数据库可以分为两种：主库、租户库。

3.1 主库

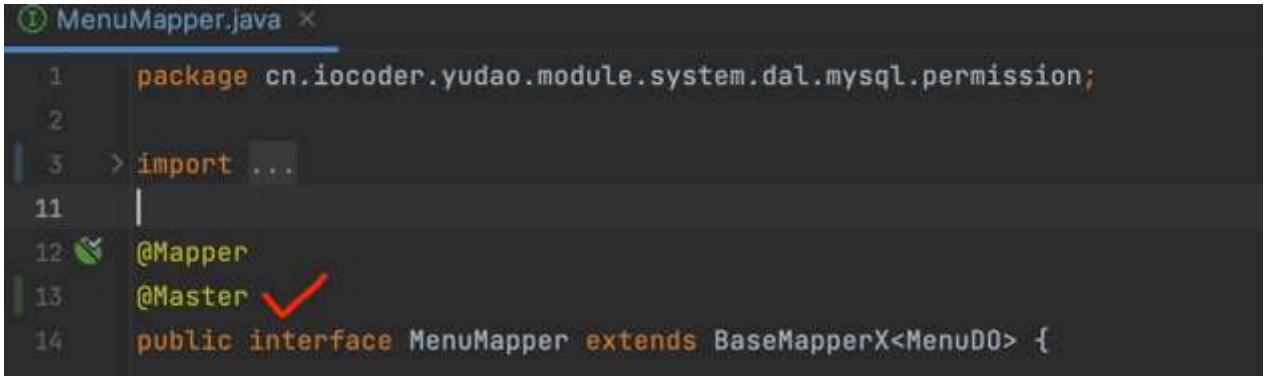
① 存放所有租户共享的表。例如说：菜单表、定时任务表等等。如下图所示：



② 对应 `master` 数据源，配置在 `application-{env}.yaml` 配置文件。如下图所示：



③ 每个主库对应的 Mapper，必须添加 `@Master` 注解。例如说：

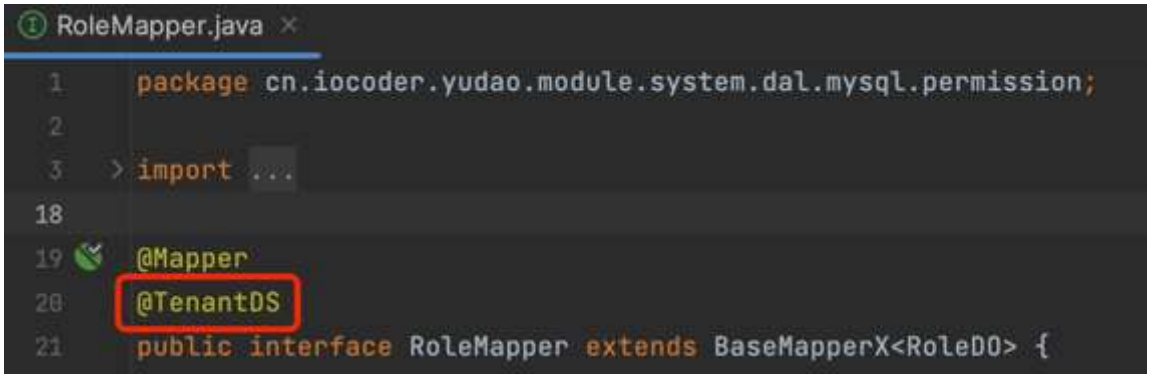


3.2 租户库

① 存放每个租户的表。例如说：用户表、角色表等等。

② 在 [基础设施 -> 数据源配置] 菜单中，配置数据源。

③ 每个主库对应的 Mapper，必须添加 `@TenantDS` 注解。例如说：



3.3 租户字段

① 考虑到拓展性，在使用 DATASOURCE 模式时，默认会叠加 COLUMN 模式，即还有 `tenant_id` 租户字段：

- 在 `INSERT` 操作时，会自动记录租户编号到 `tenant_id` 字段。
- 在 `SELECT` 操作时，会自动添加 `WHERE tenant_id = ?` 查询条件。

如果你不需要，可以直接删除 TenantDatabaseInterceptor 类，以及它的 Bean 自动配置。

拓展性，指的是部分【大】租户独立数据库，部分【小】租户共享数据。

② 也因为叠加了 COLUMN 模式，**主库**的表需要根据情况添加 `tenant_id` 字段。

- 情况一：不需要添加 `tenant_id` 字段。例如说：菜单表、定时任务表等等。注意，需要把表名添加到 `yudao.tenant.ignore-tables` 配置项中。
- 情况二：需要 `tenant_id` 字段。例如说：访问日志表、异常日志表等等。目的，排查是哪个租户的系统级别的日志。

4. 多数据源事务

使用 DATASOURCE 模式后，可能一个操作涉及到多个数据源。例如说：创建租户时，即需要操作主库，也需要操作租户库。

考虑到多数据的数据一致性，我们会采用事务的方式，而使用 Spring 事务时，会存在多数据库无法切换的问题。不了解的胖友，可以阅读 《MyBatis Plus 的多数据源 `@DS` 切换不起作用了，谁的锅》 文章。

多数据源的事务方案，是一个老生常谈的问题。比较主流的，有如下两种，都是相对重量级的方案：

1. 使用 Atomikos 实现 JTA 分布式事务，配置复杂，性能较差。
2. 使用 Seata 实现分布式事务，使用简单，性能不错，但是需要额外引入 Seata Server 服务。

4.1 本地事务

考虑到项目是单体架构，不适合采用重量级的事务，因此采用 dynamic-datasource 提供的 “**本地事务**” 轻量级方案。

它的实现原理是：自定义 `@DSTransactional` 事务注解，替代 Spring `@Transactional` 事务注解。

- 在逻辑执行成功时，循环提交每个数据源的事务。
- 在逻辑执行失败时，循环回滚每个数据源的事务。

但是它存在一个风险点，如果数据库发生异常（例如说宕机），那么本地事务就可能会存在数据不一致的问题。例如说：

- ① 主库的事务提交
- ② 租户库发生异常，租户的事务提交失败
- 结果：主库的数据已经提交，而租户库的数据没有提交，就会导致数据不一致。

因此，如果你的系统对数据一致性要求很高，那么请使用 Seata 方案。

4.2 使用示例

在最外层的 Service 方法上，添加 `@DSTransactional` 注解。例如说，创建租户的 Service 方法：

TenantServiceImpl.java

```java
@Override
@DSTransactional // 多数据源，使用 @DSTransactional 保证本地事务，以及数据源的切换
public Long createTenant(TenantCreateReqVO createReqVO) {
    // 校验套餐被禁用
    TenantPackageDO tenantPackage = tenantPackageService.validTenantPackage(createReqVO.getPackageId());

    // 创建租户
    TenantDO tenant = TenantConvert.INSTANCE.convert(createReqVO);
    tenantMapper.insert(tenant);

    TenantUtils.execute(tenant.getId(), () -> {
        // 创建角色
        Long roleId = createRole(tenantPackage);
        // 创建用户，并分配角色
        Long userId = createUser(roleId, createReqVO);
        // 修改租户的管理员
        tenantMapper.updateById(new TenantDO().setId(tenant.getId()).setContactUserId(userId));
    });
    return tenant.getId();
}

private Long createUser(Long roleId, TenantCreateReqVO createReqVO) {
    // 创建用户
    Long userId = userService.createUser(TenantConvert.INSTANCE.convert02(createReqVO));
    // 分配角色
    permissionService.assignUserRole(userId, singleton(roleId));
    return userId;
}

private Long createRole(TenantPackageDO tenantPackage) {
    // 创建角色
    RoleCreateReqVO reqVO = new RoleCreateReqVO();
    reqVO.setName(RoleCodeEnum.TENANT_ADMIN.getName()).setCode(RoleCodeEnum.TENANT_ADMIN.getCode())
            .setSort(0).setRemark("系统自动生成");
    Long roleId = roleService.createRole(reqVO, RoleTypeEnum.SYSTEM.getType());
    // 分配权限
    permissionService.assignRoleMenu(roleId, tenantPackage.getMenuIds());
    return roleId;
}
```

注意，里面不能嵌套有 Spring 自带的事务，就是上图中【黄圈】的 Service 方法不能使用 Spring `@Transactional` 注解，否则会导致数据源无法切换。

如果【黄圈】的 Service 自身还需要事务，那么可以使用 `@DSTransactional` 注解。

← SaaS 多租户【字段隔离】 WebSocket 实时通信→