芋道源码 2022-12-31

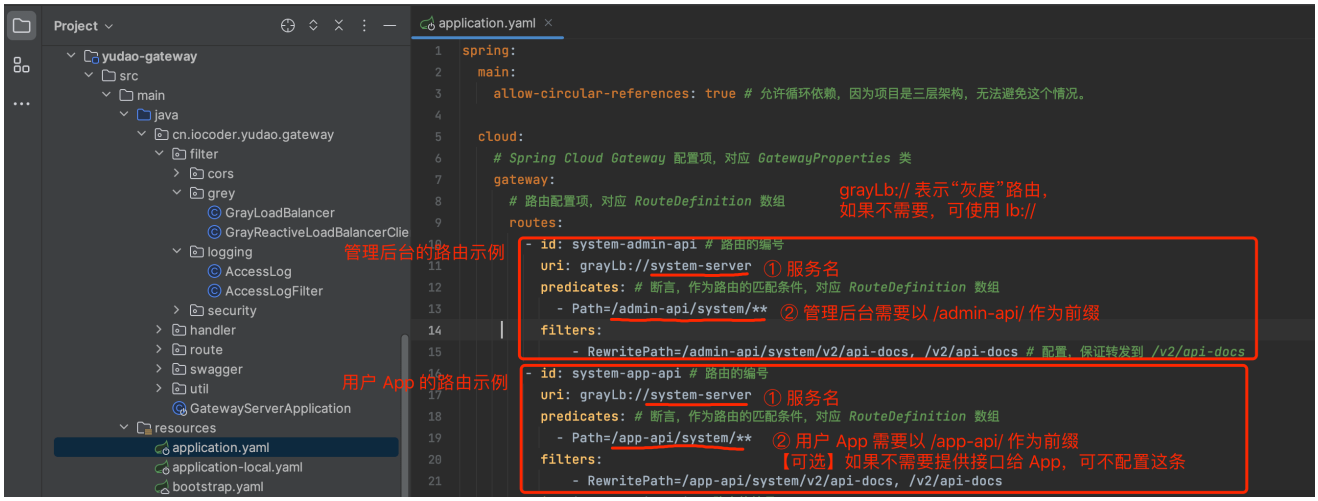
服务网关 Spring Cloud Gateway

`yudao-gateway` 模块，基于 Spring Cloud Gateway 构建 API 服务网关，提供用户认证、服务路由、灰度发布、访问日志、异常处理等功能。

> **友情提示：如何学习 Spring Cloud Gateway?**
>
> 阅读《芋道 Spring Cloud 网关 Spring Cloud Gateway 入门》文章。

1. 服务路由

新建服务后，在 `application.yaml` 配置文件中，需要添加该服务的路由配置。示例如下图：

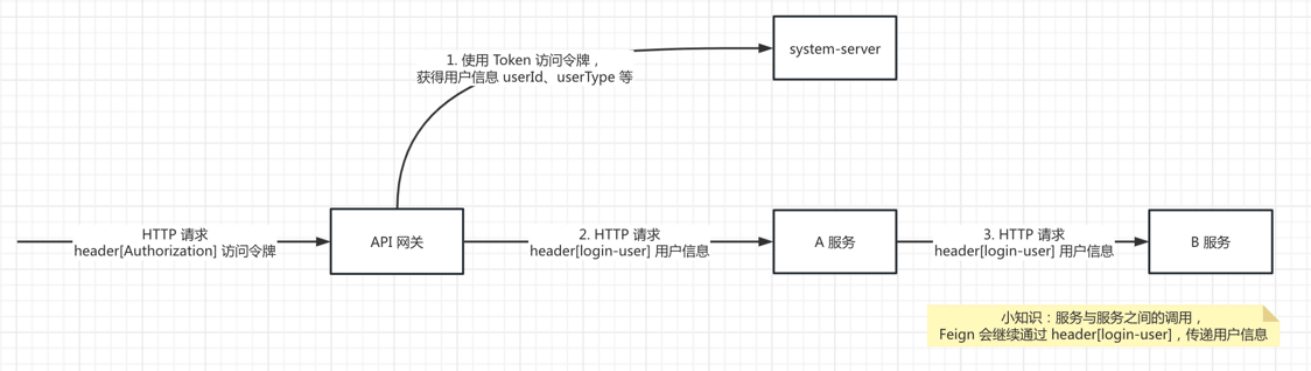


2. 用户认证

由 `filter/security` 包实现，无需配置。

TokenAuthenticationFilter 会获得请求头中的 `Authorization` 字段，调用 `system-server` 服务，进行用户认证。

- 如果认证成功，会将用户信息放到 `login-user` 请求头，转发到后续服务。后续服务可以从 `login-user` 请求头，解析到用户信息。
- 如果认证失败，依然会转发到后续服务，由该服务决定是否需要登录，是否需要校验权限。



考虑到性能，API 网关会本地缓存 Token 与用户信息，每次收到 HTTP 请求时，异步从 `system-server` 刷新本地缓存。

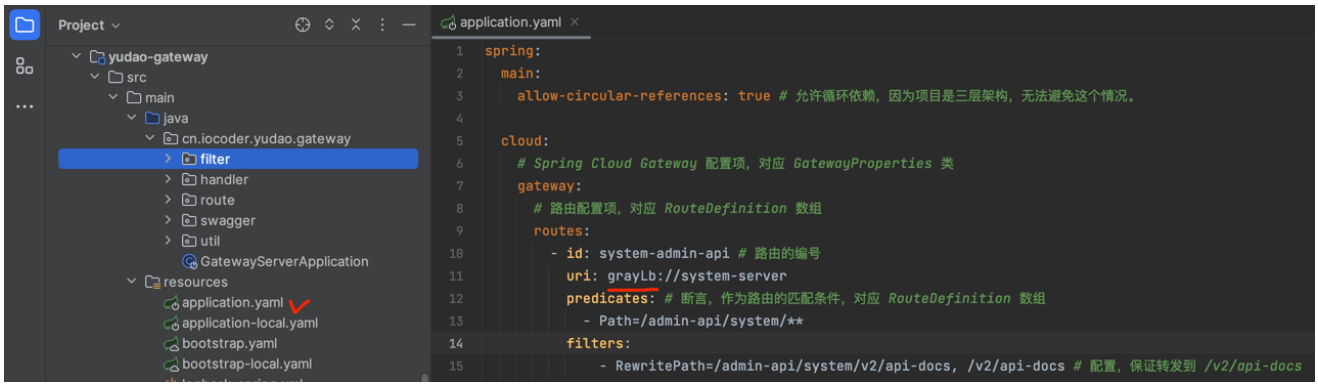
3. 灰度发布

由 `filter/grey` 包实现，实现原理如下：

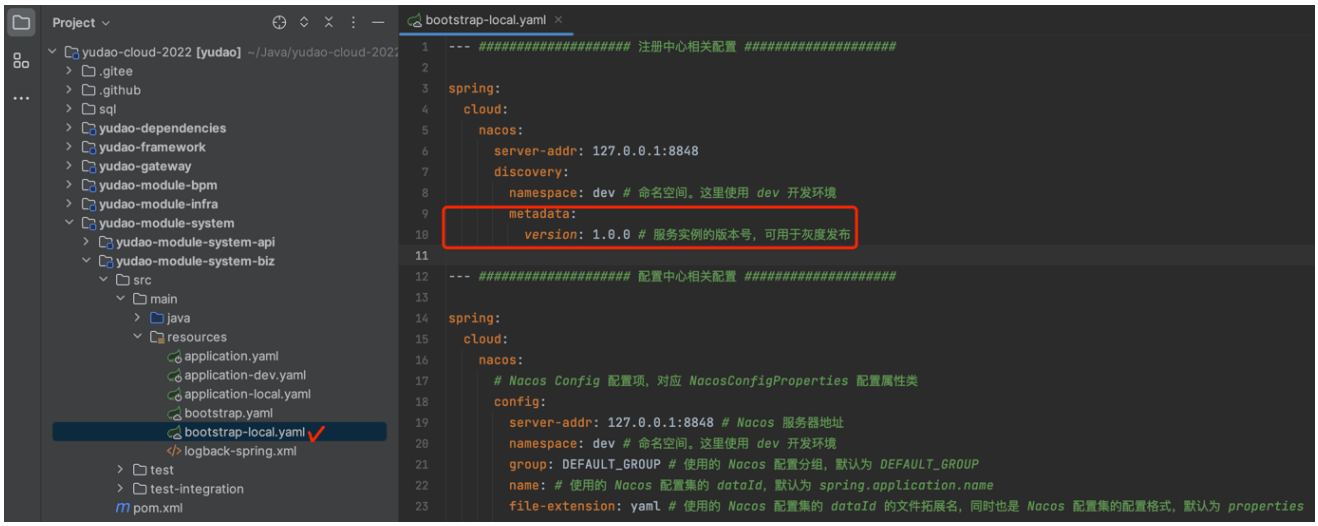
```
/**
 * 灰度 {@link GrayLoadBalancer} 实现类
 *
 * 根据请求的 header[version] 匹配，筛选满足 metadata[version] 相等的服务实例列表，然后随机 + 权重进行选择一个
 * 1. 假如请求的 header[version] 为空，则不进行筛选，所有服务实例都进行选择
 * 2. 如果 metadata[version] 都不相等，则不进行筛选，所有服务实例都进行选择
 *
 * 注意，考虑到实现的简易，它的权重是使用 Nacos 的 nacos.weight，所以随机 + 权重也是基于 {@link NacosBalancer} 筛选。
 * 也就是说，如果你不使用 Nacos 作为注册中心，需要微调一下筛选的实现逻辑
 *
 * @author 芋道源码
 */
@RequiredArgsConstructor
@Slf4j
public class GrayLoadBalancer implements ReactorServiceInstanceLoadBalancer {
```

所以在使用灰度时，如要如下配置：

① 第一步，【网关】配置服务的路由配置使用 `grebLb://` 协议，指向灰度服务。例如说：



② 第二步，【服务】配置服务的版本 `version` 配置。例如说：



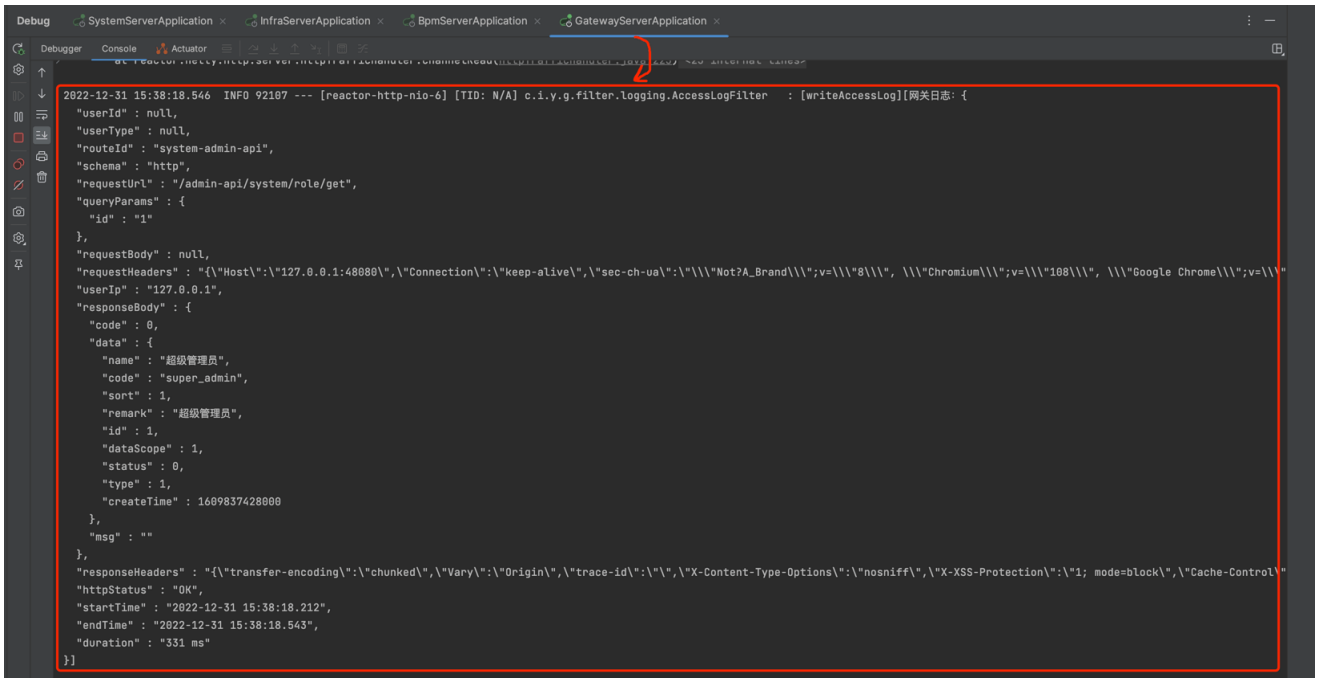
③ 第三步，请求 API 网关时，请求头带上想要 `version` 版本。

可能想让用户的请求带上 `version` 请求头比较难，可以通过 Spring Cloud Gateway 修改请求头，通过 User Agent、Cookie、登录用户等信息，来判断用户想要的版本。详细的解析，可见《Spring Cloud Gateway 实现灰度发布功能》文章。

4. 访问日志

由 `filter/logging` 包实现，无需配置。

每次收到 HTTP 请求时，会打印访问日志，包括 Request、Response、用户等信息。如下图所示：



5. 异常处理

由 GlobalExceptionHandler 累实现，无需配置。

请求发生异常时，会翻译异常信息，返回给用户。例如说：

```
{
  "code": 500,
  "data": null,
  "msg": "系统异常"
}
```

6. 动态路由

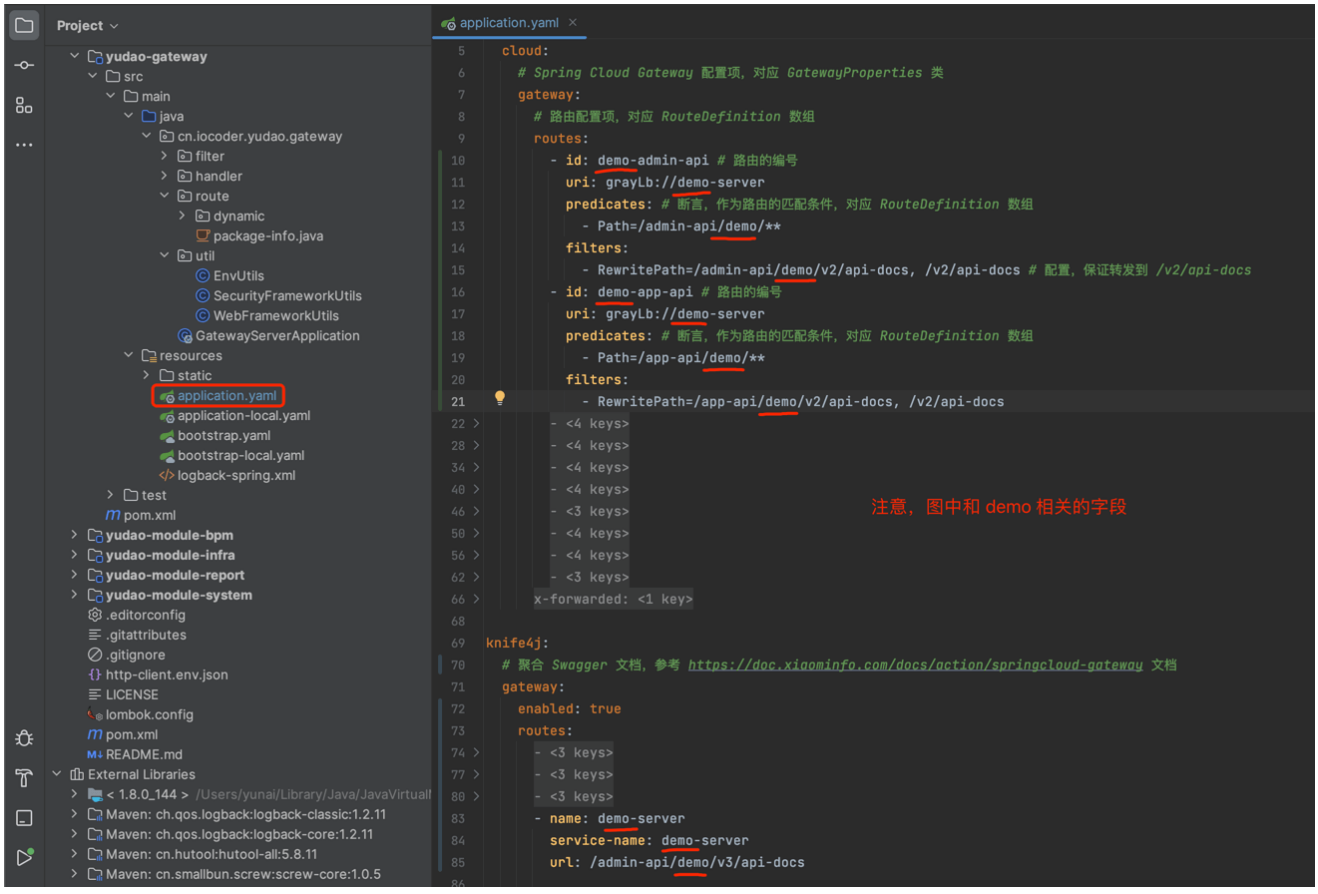
在 Nacos 配置发生变化时，Spring Cloud Alibaba Nacos Config 内置的监听器，会监听到配置刷新，最终触发 Gateway 的路由信息刷新。

参见《芋道 Spring Cloud 网关 Spring Cloud Gateway 入门》博客的「6. 基于配置中心 Nacos 实现动态路由」小节。

使用方式：在 Nacos 新增 DataId 为 `gateway-server.yaml` 的配置，修改 `spring.cloud.gateway.routes` 配置项。

7. Swagger 接口文档

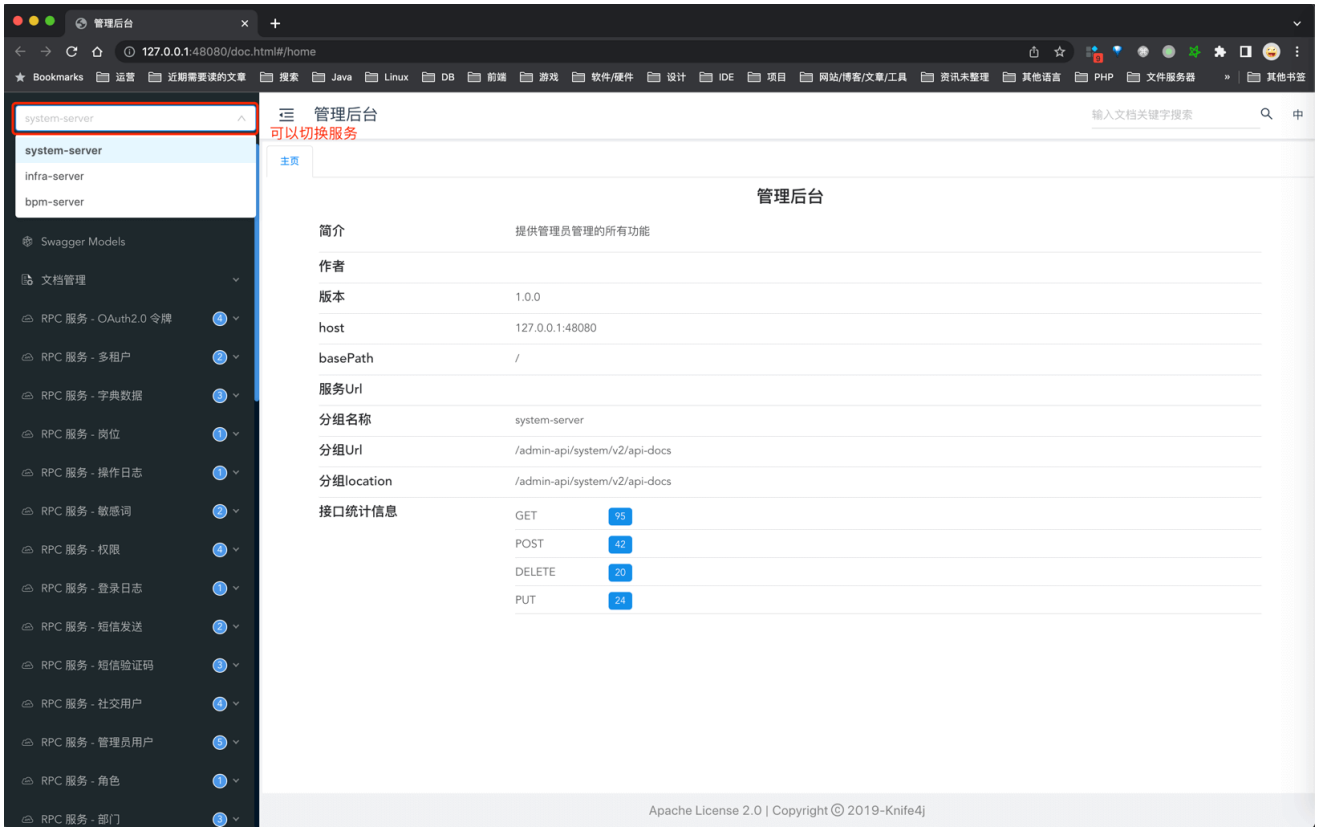
基于 Knife4j 实现 Swagger 接口文档的网关聚合。需要路由配置如下：



> **友情提示：图中的 /v2/ 都改成 /v3/，或者以下面的文字为准！！！**

- 管理后台的接口： `- RewritePath=/admin-api/{服务的基础路由}/v3/api-docs, /v3/api-docs`
- 用户 App 的接口： `- RewritePath=/app-api/{服务的基础路由}/v3/api-docs, /v3/api-docs`
- Knife4j 配置： `knife4j.gateway.routes` 添加

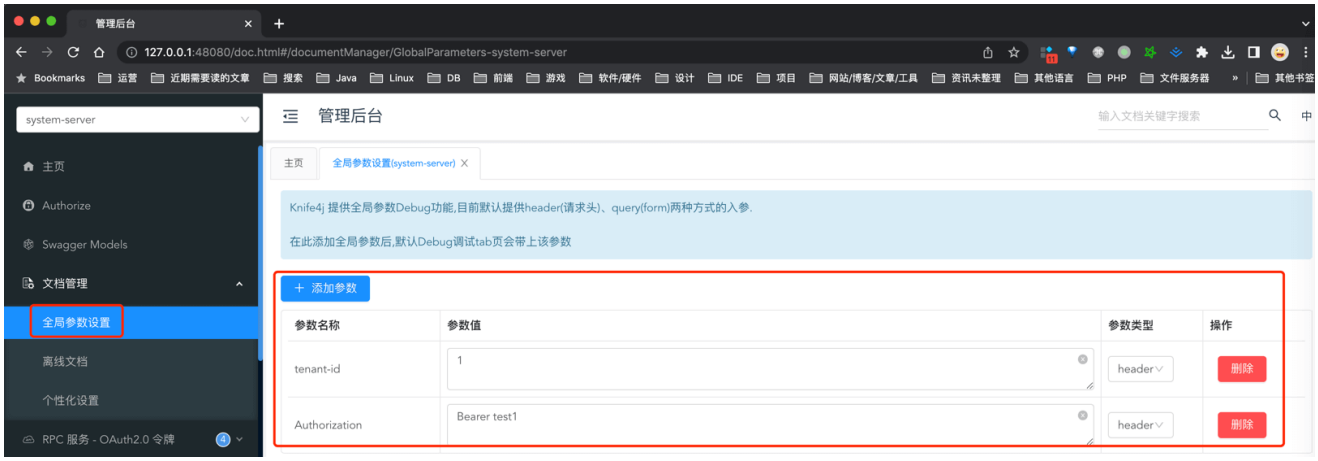
浏览器访问 http://127.0.0.1:48080/doc.html 地址，可以看到所有接口的信息。如下图所示：



7.1 如何调用

○ 点击左边「文档管理 - 全局参数设置」菜单，设置 `header-id` 和 `Authorization` 请求头。如下图所示：

```
tenant-id: 1
Authorization: Bearer test1
```

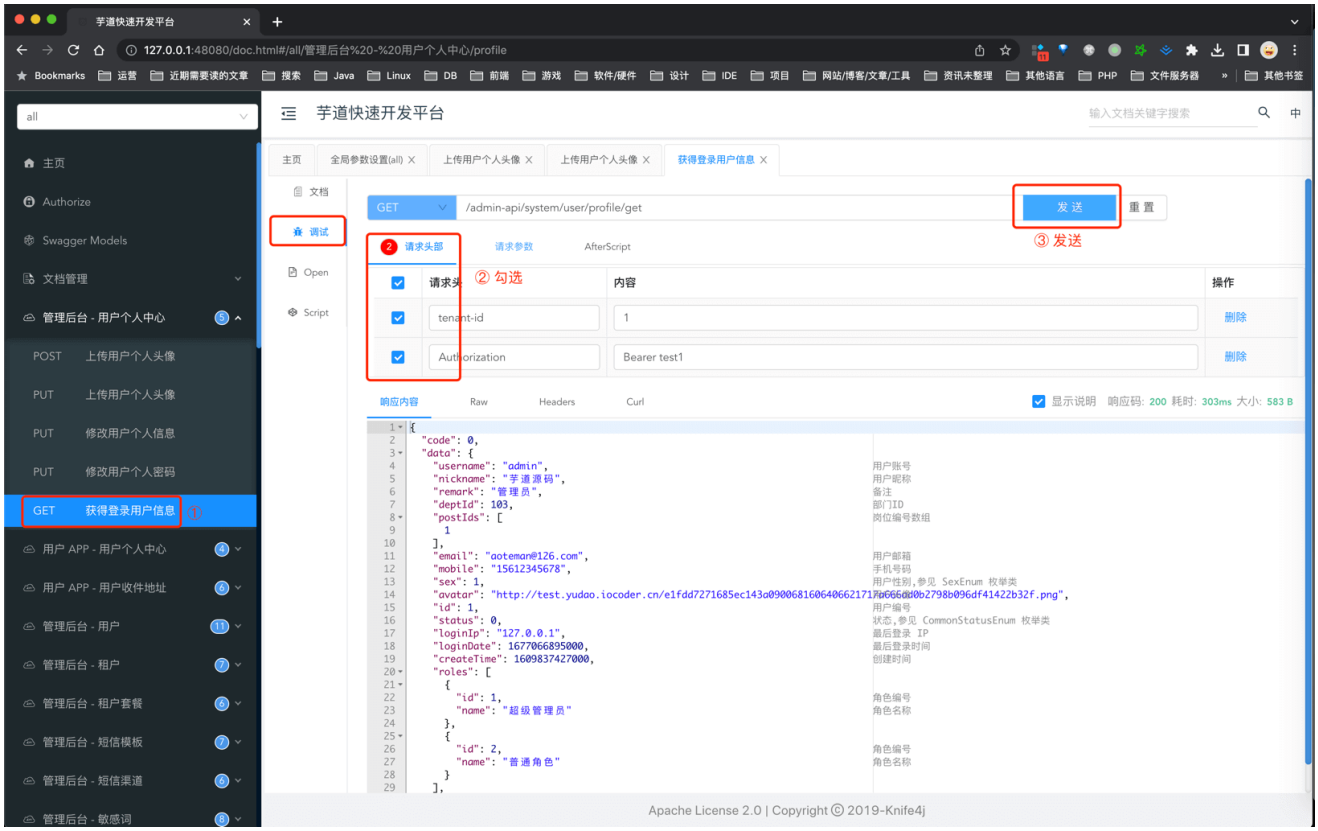


添加完后，需要 F5 刷新下网页，否则全局参数不生效。

① 点击任意一个接口，进行接口的调用测试。这里，使用「管理后台 - 用户个中心」的“获得登录用户信息”举例子。

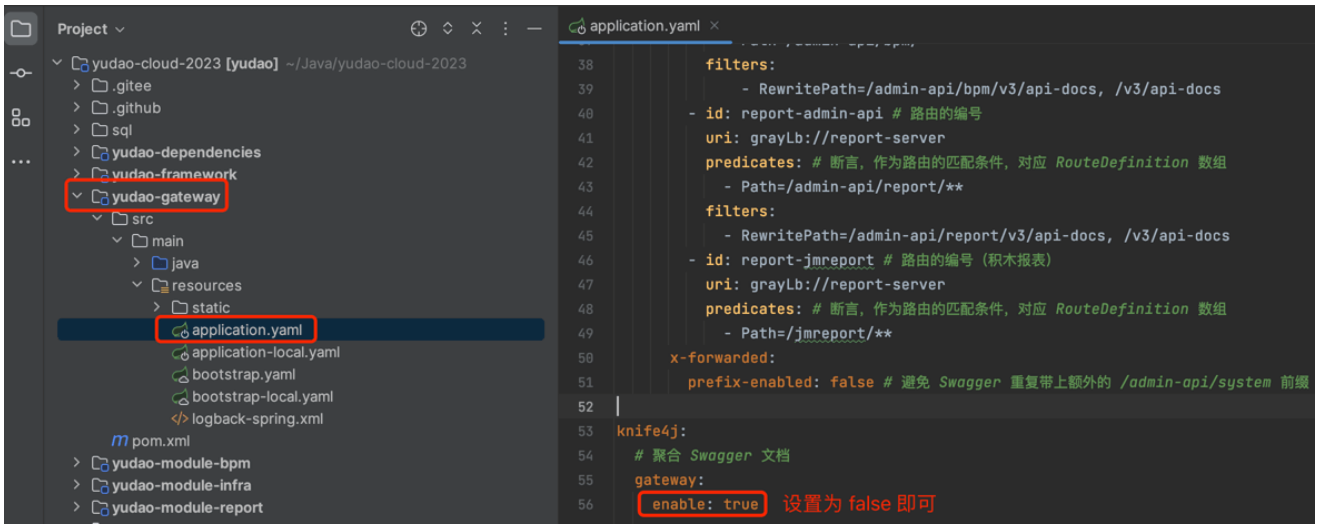
② 点击左侧「调试」按钮，并将请求头部的 `header-id` 和 `Authorization` 勾选上。其中，`header-id` 为租户编号，`Authorization` 的 `"Bearer test"` 后面为用户编号（模拟哪个用户操作）。

③ 点击「发送」按钮，即可发起一次 API 的调用。



7.2 如何关闭

如果想要禁用 Swagger 功能，可通过 `knife4j.gateway.enabled` 配置项为 `false` 。一般情况下，建议 prod 生产环境进行禁用，避免发生安全问题。



8. Cors 跨域处理

由 `filter/cors` 包实现，无需配置。

← 配置中心 Nacos

服务调用 Feign→