外部项目接入氢信SaaS手册

氢信SaaS基座Api文档

Swagger API文档Swagger UI（等待云服务器下发）

项目环境

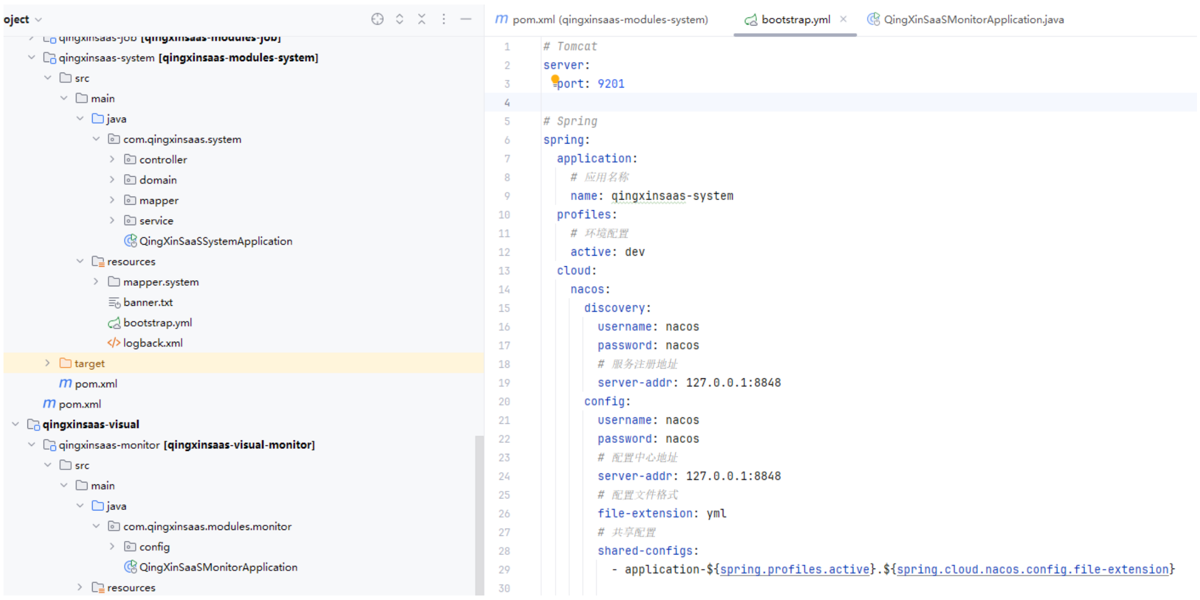
外部项目接入氢信SaaS需要适配cloud环境

```
        <!-- SpringCloud Alibaba Nacos -->
        <dependency>
            <groupId>com.alibaba.cloud</groupId>
            <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
        </dependency>

        <!-- SpringCloud Alibaba Nacos Config -->
        <dependency>
            <groupId>com.alibaba.cloud</groupId>
            <artifactId>spring-cloud-starter-alibaba-nacos-config</artifactId>
        </dependency>

        <!-- SpringCloud Alibaba Sentinel -->
        <dependency>
            <groupId>com.alibaba.cloud</groupId>
            <artifactId>spring-cloud-starter-alibaba-sentinel</artifactId>
        </dependency>
        <!-- SpringCloud Openfeign -->
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-openfeign</artifactId>
        </dependency>

        <!-- SpringCloud Loadbalancer -->
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-loadbalancer</artifactId>
        </dependency>
```


```
oject
  qingxinsaas-job [qingxinsaas-modules-job]
  qingxinsaas-system [qingxinsaas-modules-system]
    src
      main
        java
          com.qingxinsaas.system
            controller
            domain
            mapper
            service
            QingXinSaaSSystemApplication
        resources
          mapper.system
          banner.txt
          bootstrap.yml
          logback.xml
    target
    pom.xml
  pom.xml
qingxinsaas-visual
  qingxinsaas-monitor [qingxinsaas-visual-monitor]
    src
      main
        java
          com.qingxinsaas.modules.monitor
            config
            QingXinSaaSMonitorApplication
        resources
```

pom.xml (qingxinsaas-modules-system) | bootstrap.yml | QingXinSaaSMonitorApplication.java

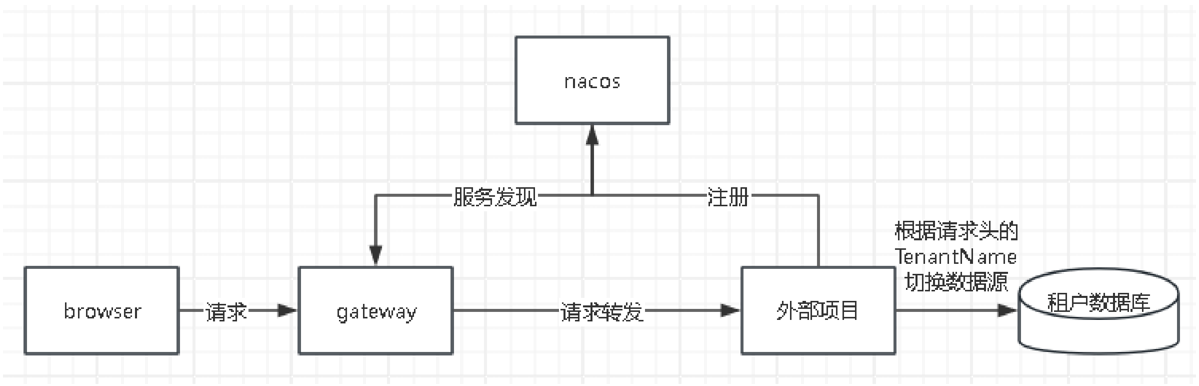
```yaml
# Tomcat
server:
  port: 9201

# Spring
spring:
  application:
    # 应用名称
    name: qingxinsaas-system
  profiles:
    # 环境配置
    active: dev
  cloud:
    nacos:
      discovery:
        username: nacos
        password: nacos
        # 服务注册地址
        server-addr: 127.0.0.1:8848
      config:
        username: nacos
        password: nacos
        # 配置中心地址
        server-addr: 127.0.0.1:8848
        # 配置文件格式
        file-extension: yml
        # 共享配置
        shared-configs:
          - application-${spring.profiles.active}.${spring.cloud.nacos.config.file-extension}

```

在bootstrap.yml中配置nacos信息，并登录nacos，为项目创建对应的yml配置文件。

租户数据隔离



外部项目通过注册到 Nacos 实现服务发现，并且 Gateway 将携带 `TenantName` 的请求转发到相应的外部项目。外部项目可以通过自定义的 `Filter` 或 `Interceptor` 实现租户信息的拦截和数据源的切换。

切换数据源基于dynamic datasource手动切换数据源的方式

- Filter

```java
  @Component
  public class DynamicDataSourceFilter implements Filter {
      @Override
      public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain) throws IOException, ServletException {
          // 从请求头中获取tenant
          String url = request.getServletPath();
          String tenant = request.getHeader("tenant");
          if (tenant == null) {
            tenant = "quake-yun";
          }

          if (StringUtils.isNotBlank(tenant)) {
            if (!dynamicRoutingDataSource.existDataSource(tenant)) {

                //查询租户对应的数据源信息
                MasterTenant masterTenant =
masterTenantService.selectMasterTenant(tenant);
```

```
                if (masterTenant == null) {
                    throw new RuntimeException("无此租户:" + tenant);
                } else if ("2".equals(masterTenant.getStatus())) {
                    throw new RuntimeException("租户[" + tenant + "]已停用");
                } else if (masterTenant.getExpirationDate() != null) {
                    if 
(masterTenant.getExpirationDate().before(DateUtils.getNowDate())) {
                        throw new RuntimeException("租户[" + tenant + "]已过
期");
                    }
                }

                //设置租户数据源信息
                Map<String, Object> map = new HashMap<>();
                map.put("driverClassName", "com.mysql.cj.jdbc.Driver");
                map.put("url", masterTenant.getUrl());
                map.put("username", masterTenant.getUsername());
                map.put("password", masterTenant.getPassword());
                map.put("uniqueResourceName", tenant);
                dynamicRoutingDataSource.addDataSource(tenant, map);
                log.info("&&&&&&&&&&& 已设置租户:{} 连接信息: {}", tenant, 
masterTenant);
            }else {
                log.info("&&&&&&&&&&& 当前租户:{}", tenant);
            }

        } else {
            throw new RuntimeException("缺少租户信息");
        }
        // 切换租户数据源
        DynamicDataSourceContextHolder.setDataSourceType(tenant);

          try {
              // 继续处理请求
              chain.doFilter(request, response);
          } finally {
              // 清除数据源标识
           DynamicDataSourceContextHolder.clearDataSourceType();
          }
    }

    // 根据请求信息确定数据源标识
    private String determineDataSourceKey(ServletRequest request) {
        // 根据请求参数、请求头等进行逻辑判断，返回相应的数据源标识
        // ...
    }

    // 其他方法实现，如初始化和销毁方法
    // ...
}
```

- Interceptor

```
@Slf4j
@Component
public class TenantDatabaseInterceptor implements HandlerInterceptor {
```

```java
    @Resource
    private DynamicDataSource dynamicRoutingDataSource;

    @Resource
    private IMasterTenantService masterTenantService;

    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse
response, Object handler) throws Exception {

        String url = request.getServletPath();
        String tenant = request.getHeader("tenant");
        if (tenant == null) {
            tenant = "quake-yun";
        }

        log.info("&&&&&&&&&&&&&&&& 租户拦截 &&&&&&&&&&&&&&&&");
        if (StringUtils.isNotBlank(tenant)) {
            if (!dynamicRoutingDataSource.existDataSource(tenant)) {

                MasterTenant masterTenant =
masterTenantService.selectMasterTenant(tenant);
                if (masterTenant == null) {
                    throw new RuntimeException("无此租户:" + tenant);
                } else if ("2".equals(masterTenant.getStatus())) {
                    throw new RuntimeException("租户[" + tenant + "]已停用");
                } else if (masterTenant.getExpirationDate() != null) {
                    if
(masterTenant.getExpirationDate().before(DateUtils.getNowDate())) {
                        throw new RuntimeException("租户[" + tenant + "]已过
期");
                    }
                }

                Map<String, Object> map = new HashMap<>();
                map.put("driverClassName", "com.mysql.cj.jdbc.Driver");
                map.put("url", masterTenant.getUrl());
                map.put("username", masterTenant.getUsername());
                map.put("password", masterTenant.getPassword());
                map.put("uniqueResourceName", tenant);
                dynamicRoutingDataSource.addDataSource(tenant, map);
                log.info("&&&&&&&&&&& 已设置租户:{} 连接信息: {}", tenant,
masterTenant);
            }else {
                log.info("&&&&&&&&&&& 当前租户:{}", tenant);
            }

        } else {
            throw new RuntimeException("缺少租户信息");
        }
        // 为了单次请求，多次连接数据库的情况，这里设置localThread，
AbstractRoutingDataSource的方法去获取设置数据源
        DynamicDataSourceContextHolder.setDataSourceType(tenant);

        return true;
    }

    @Override
    public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler, ModelAndView modelAndView) throws Exception {
        // 请求结束删除localThread
        DynamicDataSourceContextHolder.clearDataSourceType();
    }
}
```

多语言支持

多语言实现可参考后台手册 | RuoYi。