**[#](https://pig4cloud.com/" \l "spring-cloud-gateway-%E6%95%B0%E6%8D%AE%E5%BA%93%E5%AD%98%E5%82%A8%E8%B7%AF%E7%94%B1%E4%BF%A1%E6%81%AF%E7%9A%84%E6%89%A9%E5%B1%95%E6%96%B9%E6%A1%88) Spring Cloud Gateway 数据库存储路由信息的扩展方案**

[**#**](https://pig4cloud.com/#%E5%8A%A8%E6%80%81%E8%B7%AF%E7%94%B1%E8%83%8C%E6%99%AF) **动态路由背景**

​ 无论你在使用Zuul还是Spring Cloud Gateway 的时候，官方文档提供的方案总是基于配置文件配置的方式

例如：

# zuul 的配置形式

routes:

pig-auth:

path: /auth/\*\*

serviceId: pig-auth

stripPrefix: true

# gateway 的配置形式

routes:

- id: pigx-auth

uri: lb://pigx-auth

predicates:

- Path=/auth/\*\*

filters:

- ValidateCodeGatewayFilter

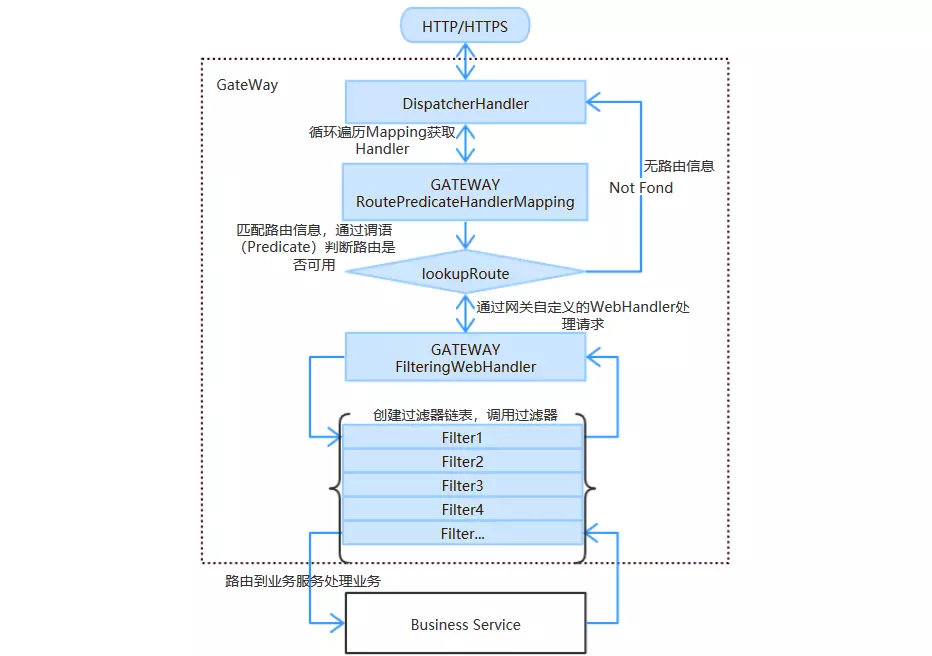
配置更改需要重启服务，不能满足实际生产过程中的动态刷新、实时变更的业务需求。

​ 基于以上分析 **pig**已经提供了基于Zuul版本的动态路由功能，附[Git 地址传送门](https://gitee.com/log4j/pig),效果如下图可以实时配置修改刷新。



[**#**](https://pig4cloud.com/#spring-cloud-gateway-%E8%B7%AF%E7%94%B1%E5%8A%A0%E8%BD%BD%E6%BA%90%E7%A0%81) **Spring Cloud Gateway 路由加载源码**

1. DispatcherHandler 接管用户请求
2. RoutePredicateHandlerMapping 路由匹配
   1. 根据RouteLocator获取 RouteDefinitionLocator
   2. 返回多个RouteDefinitionLocator.getRouteDefinitions()的路由定义信息
3. FilteringWebHandler执行路由定义中的filter 最后路由到具体的业务服务中



[**#**](https://pig4cloud.com/#spring-cloud-gateway-%E9%BB%98%E8%AE%A4%E5%8A%A8%E6%80%81%E8%B7%AF%E7%94%B1%E5%AE%9E%E7%8E%B0) **Spring Cloud Gateway 默认动态路由实现**

GatewayControllerEndpoint 基于actuate端点的默认实现，支持JVM 级别的动态路由，不能序列化存储



// 上图动态路由的信息保存的默认实现是基于内存的实现

public class InMemoryRouteDefinitionRepository implements RouteDefinitionRepository {

private final Map<String, RouteDefinition> routes = synchronizedMap(new LinkedHashMap<String, RouteDefinition>());

@Override

public Mono<Void> save(Mono<RouteDefinition> route){}

@Override

public Mono<Void> delete(Mono<String> routeId){}

@Override

public Flux<RouteDefinition> getRouteDefinitions(){}

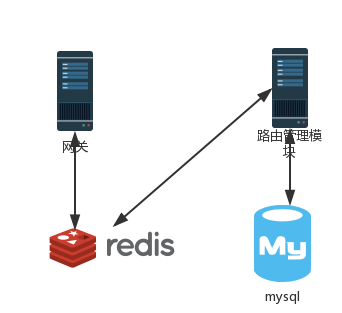
}

[**#**](https://pig4cloud.com/#%E6%89%A9%E5%B1%95%E5%9F%BA%E4%BA%8Emysql-%2B-redis%E5%AD%98%E5%82%A8%E5%88%86%E5%B8%83%E5%BC%8F%E5%8A%A8%E6%80%81%E7%BB%84%E4%BB%B6) **扩展基于Mysql + Redis存储分布式动态组件**

[**#**](https://pig4cloud.com/#%E4%B8%BA%E4%BB%80%E4%B9%88%E4%BD%BF%E7%94%A8mysql%E7%9A%84%E5%90%8C%E6%97%B6%EF%BC%8C%E5%8F%88%E8%A6%81%E4%BD%BF%E7%94%A8redis%EF%BC%9F) **为什么使用Mysql的同时，又要使用Redis？**

1. spring cloud gateway 基于webflux 背压，暂时不支持mysql 数据库
2. Redis-reactive 支持 spring cloudgateway 的背压，同时还可以实现分布式，高性能

[**#**](https://pig4cloud.com/#%E6%89%A9%E5%B1%95%E6%80%9D%E8%B7%AF) **扩展思路**



1. 增加一个路由管理模块，参考GatewayControllerEndpoint实现，启动时加载数据库中配置文件到Redis
2. 网关模块重写RouteDefinitionRepository，getRouteDefinitions（）取Redis中读取即可实现
3. 前端配合 json-view 类似插件，直接修改展示。

[**#**](https://pig4cloud.com/#%E5%85%B7%E4%BD%93%E5%AE%9E%E7%8E%B0) **具体实现**

1. 路由管理模块核心处理逻辑，获取路由和更新路由

/\*\*

\* @author lengleng

\* @date 2018年11月06日10:27:55

\* <p>

\* 动态路由处理类

\*/

@Slf4j

@AllArgsConstructor

@Service("sysRouteConfService")

public class SysRouteConfServiceImpl extends ServiceImpl<SysRouteConfMapper, SysRouteConf> implements SysRouteConfService {

private final RedisTemplate redisTemplate;

private final ApplicationEventPublisher applicationEventPublisher;

/\*\*

\* 获取全部路由

\* <p>

\* RedisRouteDefinitionWriter.java

\* PropertiesRouteDefinitionLocator.java

\*

\* @return

\*/

@Override

public List<SysRouteConf> routes() {

SysRouteConf condition = new SysRouteConf();

condition.setDelFlag(CommonConstant.STATUS\_NORMAL);

return baseMapper.selectList(new EntityWrapper<>(condition));

}

/\*\*

\* 更新路由信息

\*

\* @param routes 路由信息

\* @return

\*/

@Override

public Mono<Void> editRoutes(JSONArray routes) {

// 清空Redis 缓存

Boolean result = redisTemplate.delete(CommonConstant.ROUTE\_KEY);

log.info("清空网关路由 {} ", result);

// 遍历修改的routes，保存到Redis

List<RouteDefinitionVo> routeDefinitionVoList = new ArrayList<>();

routes.forEach(value -> {

log.info("更新路由 ->{}", value);

RouteDefinitionVo vo = new RouteDefinitionVo();

Map<String, Object> map = (Map) value;

Object id = map.get("routeId");

if (id != null) {

vo.setId(String.valueOf(id));

}

Object predicates = map.get("predicates");

if (predicates != null) {

JSONArray predicatesArray = (JSONArray) predicates;

List<PredicateDefinition> predicateDefinitionList =

predicatesArray.toList(PredicateDefinition.class);

vo.setPredicates(predicateDefinitionList);

}

Object filters = map.get("filters");

if (filters != null) {

JSONArray filtersArray = (JSONArray) filters;

List<FilterDefinition> filterDefinitionList

= filtersArray.toList(FilterDefinition.class);

vo.setFilters(filterDefinitionList);

}

Object uri = map.get("uri");

if (uri != null) {

vo.setUri(URI.create(String.valueOf(uri)));

}

Object order = map.get("order");

if (order != null) {

vo.setOrder(Integer.parseInt(String.valueOf(order)));

}

redisTemplate.setHashValueSerializer(new Jackson2JsonRedisSerializer<>(RouteDefinitionVo.class));

redisTemplate.opsForHash().put(CommonConstant.ROUTE\_KEY, vo.getId(), vo);

routeDefinitionVoList.add(vo);

});

// 逻辑删除全部

SysRouteConf condition = new SysRouteConf();

condition.setDelFlag(CommonConstant.STATUS\_NORMAL);

this.delete(new EntityWrapper<>(condition));

//插入生效路由

List<SysRouteConf> routeConfList = routeDefinitionVoList.stream().map(vo -> {

SysRouteConf routeConf = new SysRouteConf();

routeConf.setRouteId(vo.getId());

routeConf.setFilters(JSONUtil.toJsonStr(vo.getFilters()));

routeConf.setPredicates(JSONUtil.toJsonStr(vo.getPredicates()));

routeConf.setOrder(vo.getOrder());

routeConf.setUri(vo.getUri().toString());

return routeConf;

}).collect(Collectors.toList());

this.insertBatch(routeConfList);

log.debug("更新网关路由结束 ");

this.applicationEventPublisher.publishEvent(new RefreshRoutesEvent(this));

return Mono.empty();

}

}

1. 网关自定义RedisRouteDefinitionRepository
2. /\*\*
3. \* @author lengleng
4. \* @date 2018/10/31
5. \* <p>
6. \* redis 保存路由信息，优先级比配置文件高
7. \*/
8. @Slf4j
9. @Component
10. @AllArgsConstructor
11. public class RedisRouteDefinitionWriter implements RouteDefinitionRepository {
12. private final RedisTemplate redisTemplate;
13. @Override
14. public Mono<Void> save(Mono<RouteDefinition> route) {
15. return route.flatMap(r -> {
16. RouteDefinitionVo vo = new RouteDefinitionVo();
17. BeanUtils.copyProperties(r, vo);
18. log.info("保存路由信息{}", vo);
19. redisTemplate.opsForHash().put(CommonConstant.ROUTE\_KEY, r.getId(), vo);
20. return Mono.empty();
21. });
22. }
23. @Override
24. public Mono<Void> delete(Mono<String> routeId) {
25. routeId.subscribe(id -> {
26. log.info("删除路由信息{}", id);
27. redisTemplate.opsForHash().delete(CommonConstant.ROUTE\_KEY, id);
28. });
29. return Mono.empty();
30. }
31. /\*\*
32. \* 动态路由入口
33. \*
34. \* @return
35. \*/
36. @Override
37. public Flux<RouteDefinition> getRouteDefinitions() {
38. redisTemplate.setHashValueSerializer(new Jackson2JsonRedisSerializer<>(RouteDefinitionVo.class));
39. List<RouteDefinitionVo> values = redisTemplate.opsForHash().values(CommonConstant.ROUTE\_KEY);
40. List<RouteDefinition> definitionList = new ArrayList<>();
41. values.forEach(vo -> {
42. RouteDefinition routeDefinition = new RouteDefinition();
43. BeanUtils.copyProperties(vo, routeDefinition);
44. definitionList.add(vo);
45. });
46. log.debug("redis 中路由定义条数： {}， {}", definitionList.size(), definitionList);
47. return Flux.fromIterable(definitionList);
48. }
49. }

3.库表定义

