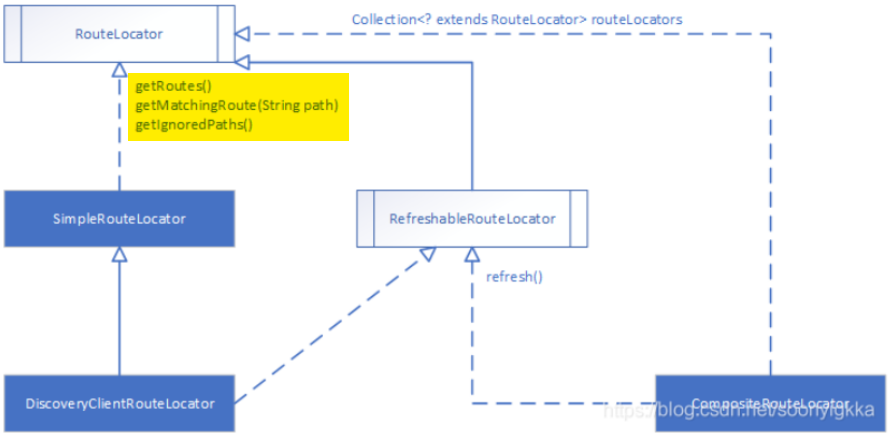
1.RouteLocation详解

1.1 RouteLocation关系

继承、组合关系见下图



1.1.1 SimpleRouteLocator详解

SimpleRouteLocator代码相对比较简单，主要看下LocateRoutes方法，根据配置文件（application.yaml）注入到zuulProperties，转换成LinkedHashMap<String, ZuulRoute>,所以SimpleRouteLocator的路由数据是读取配置文件中的数据。

```
protected Map<String, ZuulRoute> locateRoutes() {
        LinkedHashMap<String, ZuulRoute> routesMap = new LinkedHashMap<String, ZuulRoute>();
        for (ZuulRoute route : this.properties.getRoutes().values()) {
            routesMap.put(route.getPath(), route);
        }
        return routesMap;
    }
```

1.1.2 DiscoveryClientRouteLocator详解

此路由的主要作用是去注册中心获取service列表，并实现了RefreshableRouteLocator的refresh接口，默认每30s刷新注册中心数据。可以看到其实现逻辑是通过discovery去查询注册中心的微服务注册数据。

```
    @Override
    protected LinkedHashMap<String, ZuulRoute> locateRoutes() {
        LinkedHashMap<String, ZuulRoute> routesMap = new LinkedHashMap<String, ZuulRoute>();
        routesMap.putAll(super.locateRoutes());
        if (this.discovery != null) {
            Map<String, ZuulRoute> staticServices = new LinkedHashMap<String, ZuulRoute>();
            for (ZuulRoute route : routesMap.values()) {
                String serviceId = route.getServiceId();
```

```
            if (serviceId == null) {
                serviceId = route.getId();
            }
            if (serviceId != null) {
                staticServices.put(serviceId, route);
            }
        }
        // Add routes for discovery services by default
        List<String> services = this.discovery.getServices();
        String[] ignored = this.properties.getIgnoredServices()
                .toArray(new String[0]);
        for (String serviceId : services) {
            // Ignore specifically ignored services and those that were manually
            // configured
            String key = "/" + mapRouteToService(serviceId) + "/**";
            if (staticServices.containsKey(serviceId)
                    && staticServices.get(serviceId).getUrl() == null) {
                // Explicitly configured with no URL, cannot be ignored
                // all static routes are already in routesMap
                // Update location using serviceId if location is null
                ZuulRoute staticRoute = staticServices.get(serviceId);
                if (!StringUtils.hasText(staticRoute.getLocation())) {
                    staticRoute.setLocation(serviceId);
                }
            }
            if (!PatternMatchUtils.simpleMatch(ignored, serviceId)
                    && !routesMap.containsKey(key)) {
                // Not ignored
                routesMap.put(key, new ZuulRoute(key, serviceId));
            }
        }
```

```
        }
        if (routesMap.get(DEFAULT_ROUTE) != null) {
            ZuulRoute defaultRoute = routesMap.get(DEFAULT_ROUTE);
            // Move the defaultServiceId to the end
            routesMap.remove(DEFAULT_ROUTE);
            routesMap.put(DEFAULT_ROUTE, defaultRoute);
        }
        LinkedHashMap<String, ZuulRoute> values = new LinkedHashMap<>();
        for (Entry<String, ZuulRoute> entry : routesMap.entrySet()) {
            String path = entry.getKey();
            // Prepend with slash if not already present.
            if (!path.startsWith("/")) {
                path = "/" + path;
            }
            if (StringUtils.hasText(this.properties.getPrefix())) {
                path = this.properties.getPrefix() + path;
                if (!path.startsWith("/")) {
                    path = "/" + path;
                }
            }
            values.put(path, entry.getValue());
        }
        return values;
    }
```

1.1.3 CompositeRouteLocator

复合路由器，主要是依赖注入了RouteLocator集合，对相关接口的实现都是遍历spring ioc容器里面存在的RouteLocator，并循环执行相对应的接口。

```
    @Override
    public Collection<String> getIgnoredPaths() {
        List<String> ignoredPaths = new ArrayList<>();
        for (RouteLocator locator : routeLocators) {
            ignoredPaths.addAll(locator.getIgnoredPaths());
        }
        return ignoredPaths;
    }

    @Override
    public List<Route> getRoutes() {
        List<Route> route = new ArrayList<>();
        for (RouteLocator locator : routeLocators) {
            route.addAll(locator.getRoutes());
        }
        return route;
    }

    @Override
    public Route getMatchingRoute(String path) {
        for (RouteLocator locator : routeLocators) {
            Route route = locator.getMatchingRoute(path);
            if (route != null) {
                return route;
            }
        }
        return null;
    }

    @Override
    public void refresh() {
        for (RouteLocator locator : routeLocators) {
            if (locator instanceof RefreshableRouteLocator) {
                ((RefreshableRouteLocator) locator).refresh();
            }
        }
    }
```

1.1.4 总结

SimpleRouteLocator，DiscoveryClientRouteLocator，CompositeRouteLocator的代码比较简单，整体的逻辑还是比较复杂，设计比较巧妙，可扩展性很强，注意以下细节点：

1）SimpleRouteLocator是@EnableZuulServer注入的，DiscoveryClientRouteLocator是@EnableZuulProxy注入的。

2）SimpleRouteLocator处理的是配置文件，DiscoveryClientRouteLocator处理的是注册中心的配置信息。若需要扩展定义如定时从数据库中拉取配置信息，则需要自定义RouteLocator实现RefreshableRouteLocator

3）SimpleRouteLocator，DiscoveryClientRouteLocator的注入方式均使用的@ConditionOnMissingBean，这意味着如果是zuulProxy注入，则只会加载DiscoveryClientRouteLocator，因为DiscoveryClientRouteLocator是SimpleRouteLocator类型，这也方便自定义RouteLocator取代原RouteLocator

举例来说：@EnableZuulProxy注入，新建CustomerRouteLocator继承SimpleRouteLocator，则会加载DiscoveryClientRouteLocator，CustomerRouteLocator

若新建CustomerRouteLocator继承DiscoveryClientRouteLocator，则只会加载CustomerRouteLocator

4）CompositeRouteLocator的注入方式为@primary，所以在其他类中定义private RouteLocator routeLocator则模式加载的是CompositeRouteLocator。所以如果其他类中调用routeLocator.getRoutes()方法，则默认会循环调用所有被加载注入的RouteLocator，其他方法也一样会有相关逻辑。

5）CompositeRouteLocator的构造函数中AnnotationAwareOrderComparator.sort(rl)，表示会对RouteLocator进行排序，排序的方式是实现Order接口的getOrder（）方法。SimpleRouteLocator，DiscoveryClientRouteLocator的getOrder（）均返回0.