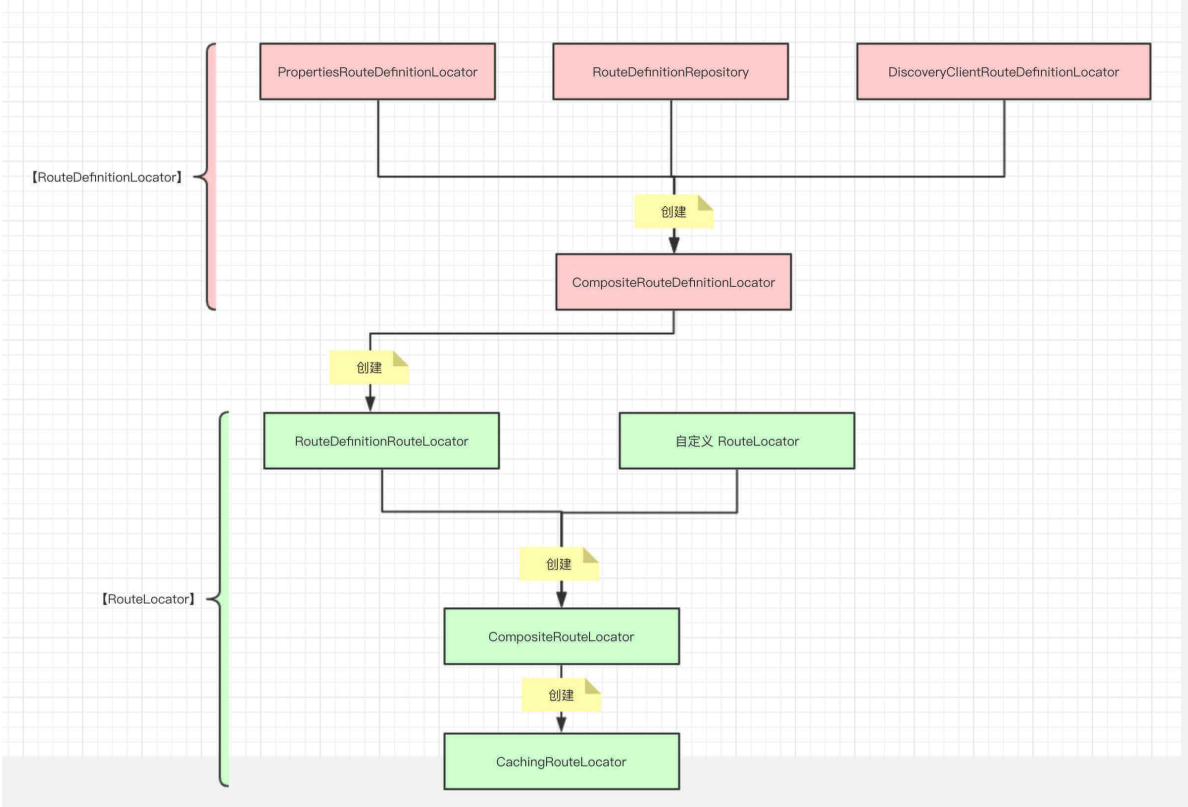
Spring-Cloud-Gateway源码系列学习

版本 v2.2.6.RELEASE

RouteDefinitionLocator 与 RouteLocator整体设计

RouteDefinitionLocator 与 RouteLocator设计图:



tip：RouteDefinitionLocator的职责是将各种配置源的配置数据转化成RouteDefinition，而RouteLocator的职责是把RouteDefinition转化成Route，关于RouteDefinition与Route等基础组件的介绍可以参考本系列#Spring-Cloud-Gateway基础组件学习，在RoutePredicateHandlerMapping类里面定义了一个routeLocator，可以看出在路由选择是根据RouteLocator#getRoutes来获取Flux，然后尝试匹配

```
private final RouteLocator routeLocator;
```

RouteDefinitionLocator相关类简介

- **RouteDefinitionLocator**：各种数据源的RouteDefinitionLocator顶级接口，里面只有一个方法，获得RouteDefinition流，Flux是Reactor的类

  ```
  Flux<RouteDefinition> getRouteDefinitions()
  ```

- **PropertiesRouteDefinitionLocator**：从配置文件(YAML / Properties等)读取并解析成Flux，解析部分使用了Spring-boot的配置功能（站在巨人的肩膀上），即标注了@ConfigurationProperties的GatewayProperties

- **RouteDefinitionRepository**：继承 RouteDefinitionLocator、RouteDefinitionWriter，是个空接口，代表从存储器(内存 / Redis / MySQL 等)读取并解析成Flux，但在v2.2.6.RELEASE版本只发现InMemoryRouteDefinitionRepository(内存的RouteDefinitionRepository实现)一个实现
    - **RouteDefinitionWriter**：是个接口，里面只有两个方法，保存和删除，需要子类实现，即需要InMemoryRouteDefinitionRepository实现

```
//保存一个RouteDefinition，Flux、Mono都是Publisher<T>，而Flux表示0-多个元素的流，而Mono表示最多一个元素的流
Mono<Void> save(Mono<RouteDefinition> route);

//根据routeId删除RouteDefinition
Mono<Void> delete(Mono<String> routeId);
```

- **DiscoveryClientRouteDefinitionLocator**：从注册中心(Eureka / Consul / Zookeeper / Etcd 等)读取并解析成Flux
- **CompositeRouteDefinitionLocator**：组合上面的配置文件、存储器、注册中心的 RouteDefinitionLocator 的实现，委派设计模式，为 RouteDefinitionRouteLocator 提供统一入口。

```
//RouteDefinitionLocator的数据流
private final Flux<RouteDefinitionLocator> delegates;

//routeId生成策略
private final IdGenerator idGenerator;
```

RouteLocator相关类简介

- **RouteLocator**：各种RouteLocator顶级接口，里面只有一个方法，获得Route Flux流

```
Flux<Route> getRoutes();
```

- **RouteDefinitionRouteLocator**：通过getRoutes()方法把RouteDefinitionLocator的Flux转成Flux，里面涉及到List转成链式的AsyncPredicate，合并defaultFilters 与 RouteDefinition的List，变成List，最后调用Route的建造者模式变成一个Route
- **自定义RouteLocator**：

```
@Bean
public RouteLocator customRouteLocator(RouteLocatorBuilder builder) {
    return builder.routes()
            .route(r -> r.host("**.abc.org").and().path("/image/png")
                .filters(f ->
                        f.addResponseHeader("X-TestHeader", "foobar"))
                .uri("http://httpbin.org:80")
            )
            .build();
}
```

- **CompositeRouteLocator**：组合了RouteDefinitionRouteLocator和自定义RouteLocator实现类，为 RoutePredicateHandlerMapping 提供统一入口访问路由

- **CachingRouteLocator**：缓存路由的 RouteLocator 实现类。RoutePredicateHandlerMapping 调用 CachingRouteLocator 的 RouteLocator#getRoutes() 方法，获取路由，其中CachingRouteLocator还实现ApplicationListener接口，证明它是事件监听者(观察者)，在onApplicationEvent方法中对缓存进行刷新

根据GatewayAutoConfiguration 源码验证 RouteDefinitionLocator 与 RouteLocator之间关系

```
public class GatewayAutoConfiguration{

    //实例化PropertiesRouteDefinitionLocator的这个Bean需要注入GatewayProperties
    @Bean
    @ConditionalOnMissingBean
    public PropertiesRouteDefinitionLocator propertiesRouteDefinitionLocator(
            GatewayProperties properties) {
        return new PropertiesRouteDefinitionLocator(properties);
    }

    //InMemoryRouteDefinitionRepository装载到了IOC容器(唯一一个存储器型
RouteDefinitionRepository)
    @Bean
    @ConditionalOnMissingBean(RouteDefinitionRepository.class)
    public InMemoryRouteDefinitionRepository inMemoryRouteDefinitionRepository()
{
        return new InMemoryRouteDefinitionRepository();
    }

     //首先List<RouteDefinitionLocator> routeDefinitionLocators是集合类型依赖注入，
IOC容器会找所有RouteDefinitionLocator类型的Bean，也就是  收集了配置文件、存储器、注册中心的
各种RouteDefinitionLocator，其次这个Bean标注了@Primary，也就代表如果其他如果要注入
RouteDefinitionLocator类型Bean就会注入CompositeRouteDefinitionLocator，因为它是
Primary的，通过这样统一各种RouteDefinitionLocator的入口
    @Bean
    @Primary
    public RouteDefinitionLocator routeDefinitionLocator(
            List<RouteDefinitionLocator> routeDefinitionLocators) {
        return new CompositeRouteDefinitionLocator(
                Flux.fromIterable(routeDefinitionLocators));
    }

    //RouteDefinitionRouteLocator,注入的就是CompositeRouteDefinitionLocator
    @Bean
    public RouteLocator routeDefinitionRouteLocator(GatewayProperties
properties,
            List<GatewayFilterFactory> GatewayFilters,
            List<RoutePredicateFactory> predicates,
            RouteDefinitionLocator routeDefinitionLocator,
            @Qualifier("webFluxConversionService") ConversionService
conversionService) {
        return new RouteDefinitionRouteLocator(routeDefinitionLocator,
predicates,
                GatewayFilters, properties, conversionService);
    }
```

```
    //首先List<RouteLocator> routeLocators是集合类型依赖注入，IOC容器会找所有
RouteLocator类型的Bean，也就是收集RouteDefinitionRouteLocator和自定义RouteLocator 的各
种RouteLocator其次这个Bean标注了@Primary，也就代表如果其他如果要注入RouteLocator类型的
Bean就会注入CachingRouteLocator，因为它是Primary的，特别关注的是
RoutePredicateHandlerMapping的注入，而CachingRouteLocator里面写死了
newCompositeRouteLocator(Flux.fromIterable(routeLocators))，即统一了入口又把Route缓
存了
    @Bean
    @Primary
    @ConditionalOnMissingBean(name = "cachedCompositeRouteLocator")
    // TODO: property to disable composite?
    public RouteLocator cachedCompositeRouteLocator(List<RouteLocator>
routeLocators) {
        return new CachingRouteLocator(
                new CompositeRouteLocator(Flux.fromIterable(routeLocators)));
    }

    //因为CachingRouteLocator的Bean标注了@Primary，所以这里RouteLocator注入的就是
CachingRouteLocator，所以RoutePredicateHandlerMapping#getHandlerInternal-
>lookupRoute方法里调用的是CachingRouteLocator的getRoutes()来进行获取所有路由
    @Bean
    public RoutePredicateHandlerMapping routePredicateHandlerMapping(
            FilteringWebHandler webHandler, RouteLocator routeLocator,
            GlobalCorsProperties globalCorsProperties, Environment environment)
{
        return new RoutePredicateHandlerMapping(webHandler, routeLocator,
                globalCorsProperties, environment);
    }
}
```