核心入口类 org.springframework.cloud.gateway.config.GatewayAutoConfiguration

```
@Configuration
@ConditionalOnProperty(name = "spring.cloud.gateway.enabled", matchIfMissing
@EnableConfigurationProperties
@AutoConfigureBefore(HttpHandlerAutoConfiguration.class)
@AutoConfigureAfter({ GatewayLoadBalancerClientAutoConfiguration.class,
        GatewayClassPathWarningAutoConfiguration.class })
@ConditionalOnClass(DispatcherHandler.class)
public class GatewayAutoConfiguration {
```

1. 提供核心类的初始化， 定义各种bean， 以Filter为核心
2. 主要包括一下bean对象，

GatewayProperties(加载配置文件),
NettyRoutingFilter,
ForwardedHeadersFilter,

RouteLocatorBuilder(为代码中定义RouteLocator提供入口),
PropertiesRouteDefinitionLocator(route配置文件定义解析入口),
InMemoryRouteDefinitionRepository,
RouteDefinitionLocator(整合所有实现了RouteDefinitionLocator接口的类),
RouteDefinitionRouteLocator(实现了RouteLocator， 最终对外提供getRoutes，并注入了 GatewayFilterFactory和RoutePredicateFactory对象，GatewayFilterFactory和RoutePredicateFactory是负责filter和predicate的生产),

FilteringWebHandler (引用所有的GlobalFilter，由org.springframework.cloud.gateway.handler.FilteringWebHandler#handle处理流程)

RoutePredicateHandlerMapping (属于流程控制类, 引用RouteLocator, FilteringWebHandler对请求进行处理， 核心方法是org.springframework.cloud.gateway.handler.RoutePredicateHandlerMapping#getHandlerInternal， 本质是一个handllermapping对象)

Predicate Factory beans(负责route的生产，其中包含AfterRoutePredicateFactory, CookieRoutePredicateFactory, HeaderRoutePredicateFactory, PathRoutePredicateFactory)

GatewayFilter Factory beans (负责filter的生产，HystrixGatewayFilterFactory, ModifyRequestBodyGatewayFilterFactory, RedirectToGatewayFilterFactory, StripPrefixGatewayFilterFactory)

```
/unchecked/
private List<GatewayFilter> loadGatewayFilters(String id, List<FilterDefinition> filterDefinitions) {
    List<GatewayFilter> filters = filterDefinitions.stream()
            .map(definition -> {
                GatewayFilterFactory factory = this.gatewayFilterFactories.get(definition.getName());
                if (factory == null) {
                    throw new IllegalArgumentException("Unable to find GatewayFilterFactory with name " + definition.getNam
                }
                Map<String, String> args = definition.getArgs();
                if (logger.isDebugEnabled()) {
                    logger.debug( o: "RouteDefinition " + id + " applying filter " + args + " to " + definition.getName());
                }

                Map<String, Object> properties = factory.shortcutType().normalize(args, factory, this.parser, this.beanFact

                Object configuration = factory.newConfig();

                ConfigurationUtils.bind(configuration, properties,
                        factory.shortcutFieldPrefix(), definition.getName(), validator);

                GatewayFilter gatewayFilter = factory.apply(configuration);
                if (this.publisher != null) {
                    this.publisher.publishEvent(new FilterArgsEvent( source: this, id, properties));
                }
```

2. GatewayLoadBalancerClientAutoConfiguration 生成 LoadBalancerClientFilter，需要使用一个 LoadBalancerClient， LoadBalancerClientFilter 负责处理 lb:// 开头的的url,

```
@Configuration
@ConditionalOnClass({LoadBalancerClient.class, RibbonAutoConfiguration.class, DispatcherHandler.class})
@AutoConfigureAfter(RibbonAutoConfiguration.class)
public class GatewayLoadBalancerClientAutoConfiguration {

    // GlobalFilter beans

    @Bean
    @ConditionalOnBean(LoadBalancerClient.class)
    @ConditionalOnMissingBean(LoadBalancerClientFilter.class)
    public LoadBalancerClientFilter loadBalancerClientFilter(LoadBalancerClient client) {
        return new LoadBalancerClientFilter(client);
    }
}
```

```java
public class LoadBalancerClientFilter implements GlobalFilter, Ordered {

    private static final Log log = LogFactory.getLog(LoadBalancerClientFilter.class);
    public static final int LOAD_BALANCER_CLIENT_FILTER_ORDER = 10100;

    protected final LoadBalancerClient loadBalancer;

    public LoadBalancerClientFilter(LoadBalancerClient loadBalancer) {
        this.loadBalancer = loadBalancer;
    }

    @Override
    public int getOrder() { return LOAD_BALANCER_CLIENT_FILTER_ORDER; }

    @Override
    public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain) {
        URI url = exchange.getAttribute(GATEWAY_REQUEST_URL_ATTR);
        String schemePrefix = exchange.getAttribute(GATEWAY_SCHEME_PREFIX_ATTR);
        if (url == null || (!"lb".equals(url.getScheme()) && !"lb".equals(schemePrefix))) {
            return chain.filter(exchange);
        }
        //preserve the original url
        addOriginalRequestUrl(exchange, url);

        log.trace( o: "LoadBalancerClientFilter url before: " + url);

        final ServiceInstance instance = choose(exchange);

        if (instance == null) {
            throw new NotFoundException("Unable to find instance for " + url.getHost());
        }
```

TODO 研究 org.springframework.cloud.netflix.ribbon.RibbonLoadBalancerClient
com.netflix.loadbalancer.BaseLoadBalancer
org.springframework.cloud.gateway.filter.factory.HystrixGatewayFilterFactory

3. GatewayMetricsAutoConfiguration

4.

FilteringWebHandler

```
#org.springframework.cloud.gateway.handler.FilteringWebHandler#handle
```

List List org.springframework.cloud.gateway.filter.LoadBalancerClientFilter#filter

```
org.springframework.cloud.gateway.config.GatewayLoadBalancerClientAutoConfiguratio
n
处理 lb://

org.springframework.cloud.netflix.ribbon.RibbonLoadBalancerClient#choose()
```

```
org.springframework.cloud.netflix.ribbon.SpringClientFactory
```