

Spring Cloud 与微服务 网关

许进(xujin.org)

About Me

- 目前对Spring Cloud进行定制工作，曾就职于**饿了么**移动基础架构组
- 网站:<http://xujin.org>
- Spring Cloud中国社区创始人 <http://springcloud.cn>
- 曾就职于**唯品会**基础架构部，从事**中间件**开发工作。
- 曾就职于**唯品会金融事业部**，参与唯品会**金融平台**(<https://jinrong.vip.com/>)的设计与开发
- 曾参与基于云计算IaaS的**平安科技云**和**国泰君安证券云**的设计与开发。

大纲

- 网关介绍
- Spring Cloud Zuul
- 网关现状
- 如何自研网关
- Q&A

Spring Cloud在国内使用情况

1. 中国联通子公司

http://fp.baidu.com/feedland/video?entry=box_searchbox_feed&id=144115189637730162&from=timeline&isappinstalled=0

2. 上海米么金服

3. 指点无限(北京)科技有限公司

4. 易保软件 目前在定制开发中

<http://www.ebaotech.com/cn/>

5. 广州简法网络

6. 深圳睿云智合科技有限公司

持续交付产品基于Spring Cloud研发 <http://www.wise2c.com>

7. 猪八戒网

8. 上海云首科技有限公司

9. 华为

整合netty进来用rpc 包括nerflix那套东西 需要注意的是sleuth traceid的传递需要自己写。tps在物理机上能突破20w

10. 东软

11. 南京云帐房网络科技有限公司

12. 四众互联(北京)网络科技有限公司

13. 深圳摩令技术科技有限公司

14. 广州万表网

15. 视觉中国

16. 上海泰苍信息科技有限公司-买单侠

17. 爱油科技(大连)有限公司

爱油科技基于SpringCloud的微服务实践

18. 广发银行

19. 卖货郎(<http://www.51mhl.com/>)

20. 拍拍贷

21. 甘肃电信

22. 新浪商品部

23. 春秋航空

24. 冰鉴科技

25. 万达网络科技集团-共享商业平台-共享供应链中心

26. 网易乐得技术团队

27. 饿了么某技术团队

28. 高阳捷迅信息科技-话费中心业务平台-凭证查询及收单系统

数据在统计之中, 会一直持续更新, 敬请期待!





网关及常见功能

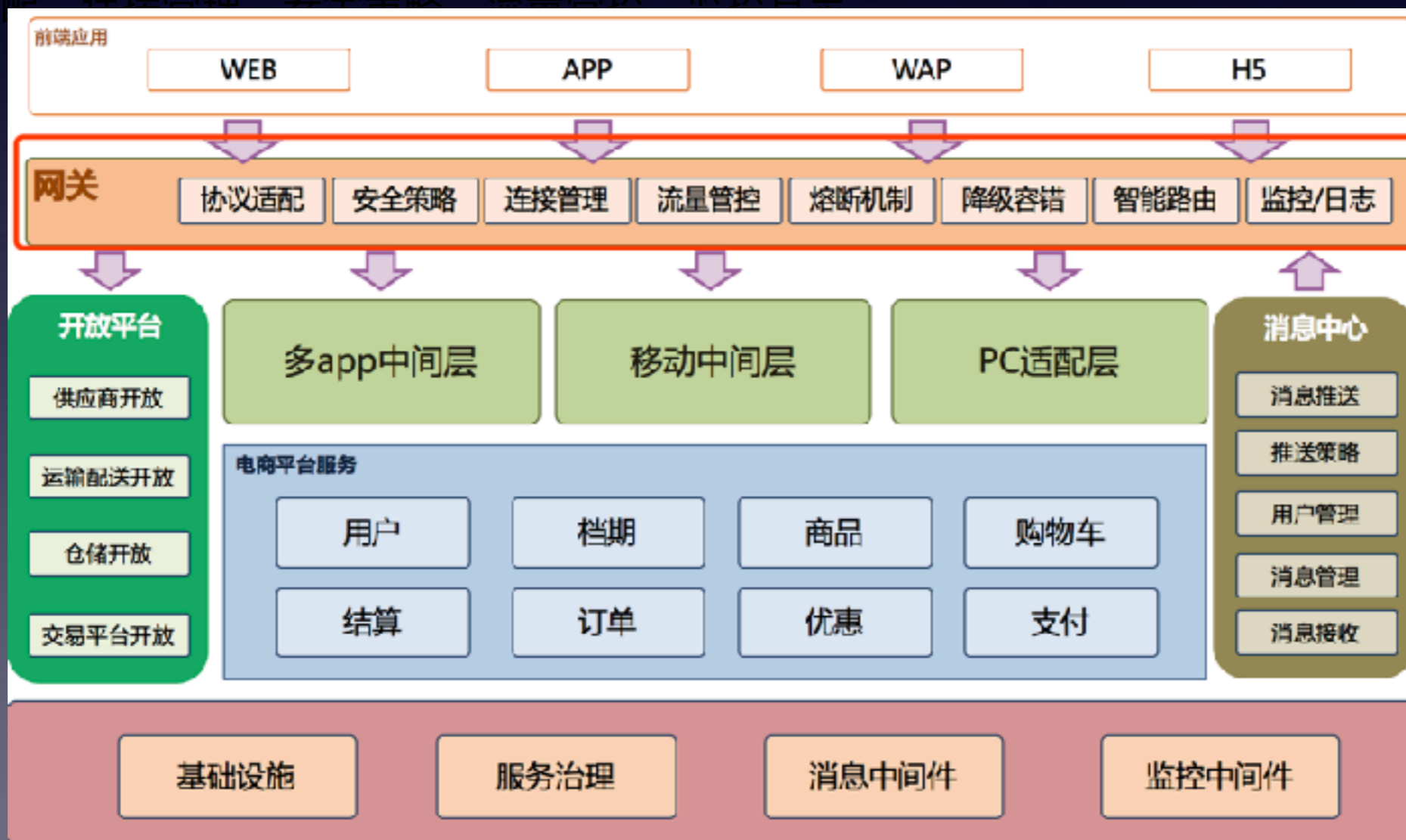
“企业级**应用防火墙**，一夫当关，万夫莫开”。

网关介绍

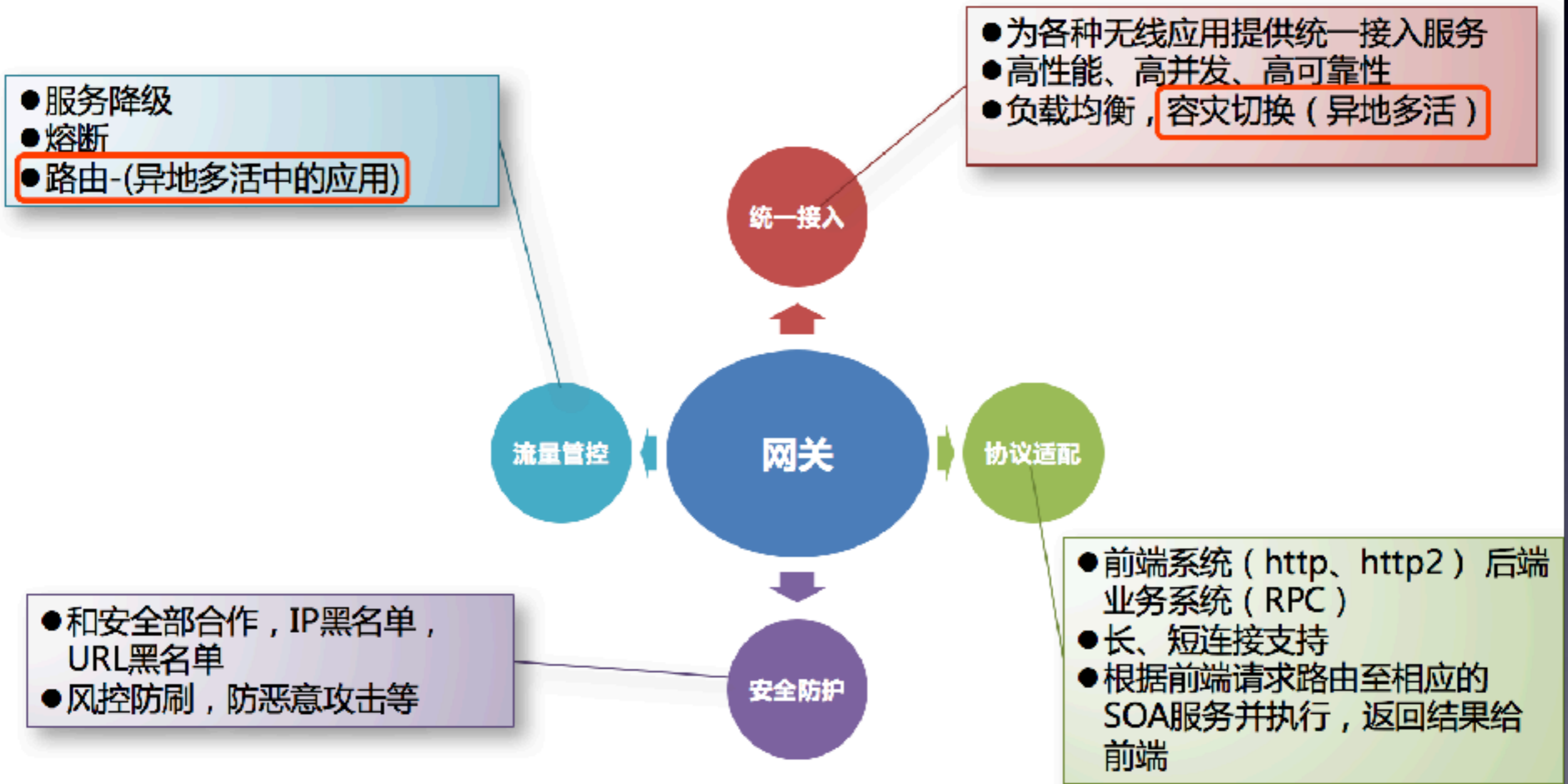
API 网关提供 **API 全托管服务**，丰富的 **API 管理功能**，辅助企业管理大规模的 **API**，以降低管理成本和安全风险。——来自阿里云的定义。

包括协议适配、协议转发、安全策略(WAF)、防刷，流量、监控日志

协议适配 连接管理 安全策略 流量管控 熔断机制 降级容错 智能路由 监控日志



网关常用功能



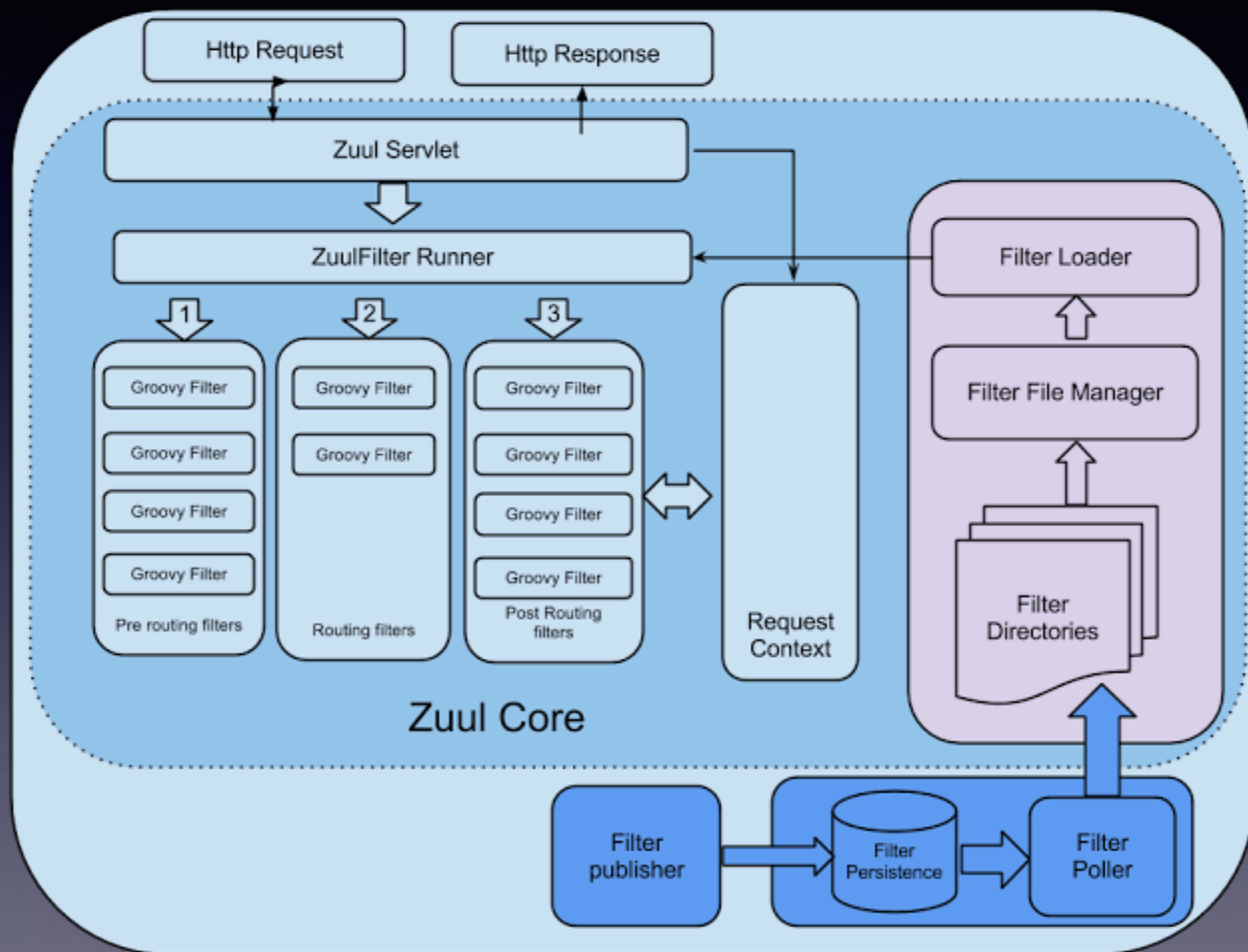


zone: 一个ezone对应一个数据中心；一个订单在一个zone中完成；ezone中的数据相互复制
 global ezone: 对强一致性有要求的数据放在global zone, global zone在单个数据中心写入, 复制到其他数据中心

Spring Cloud Zuul

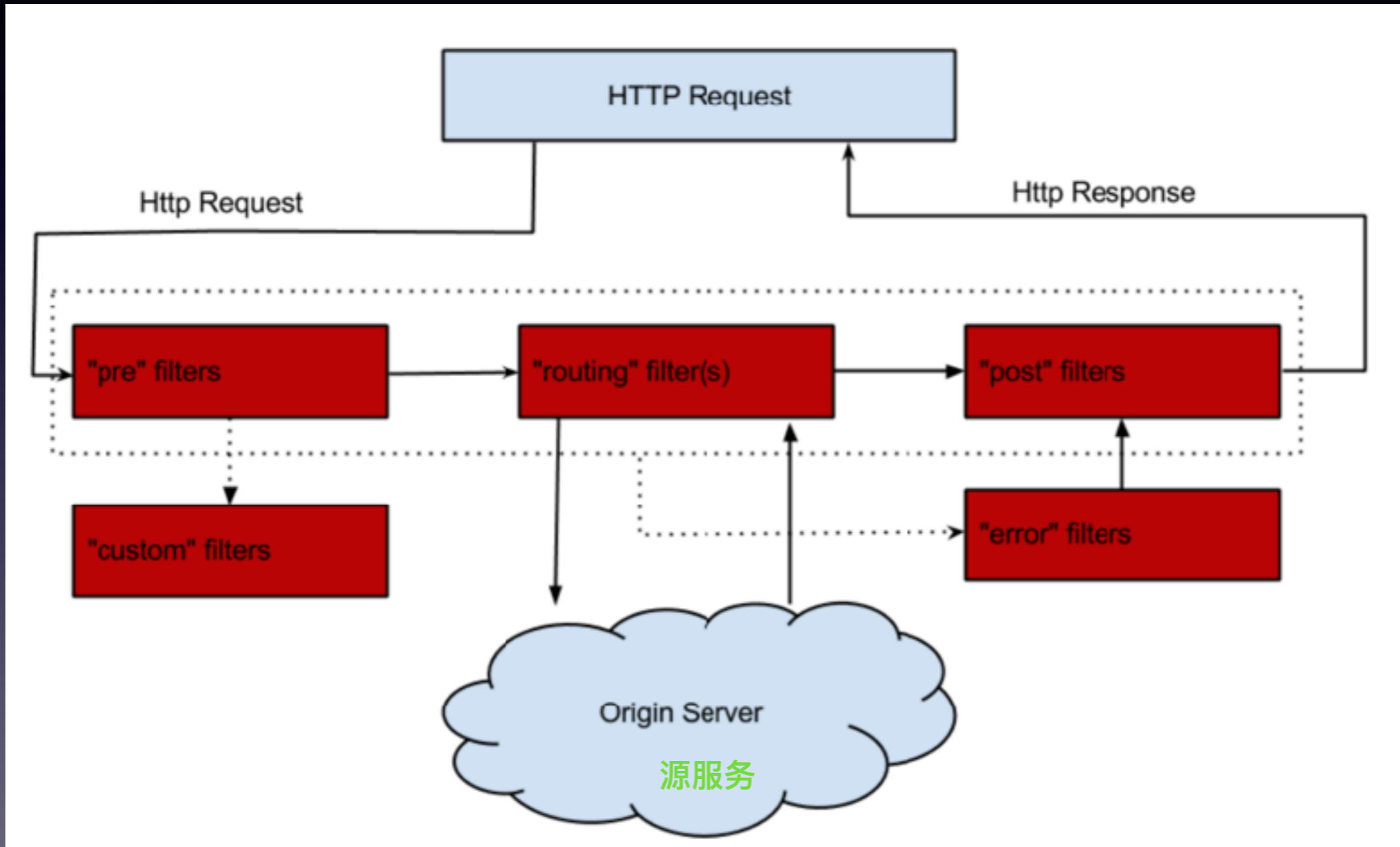
Spring Cloud Zuul 通过与 Spring Cloud Eureka 进行整合，将自身注册到 Eureka Server 中，与 Eureka, Ribbon, Hystrix 等整合，同时从 Eureka 中获得了所有其它微服务的实例信息。

Zuul的架构图

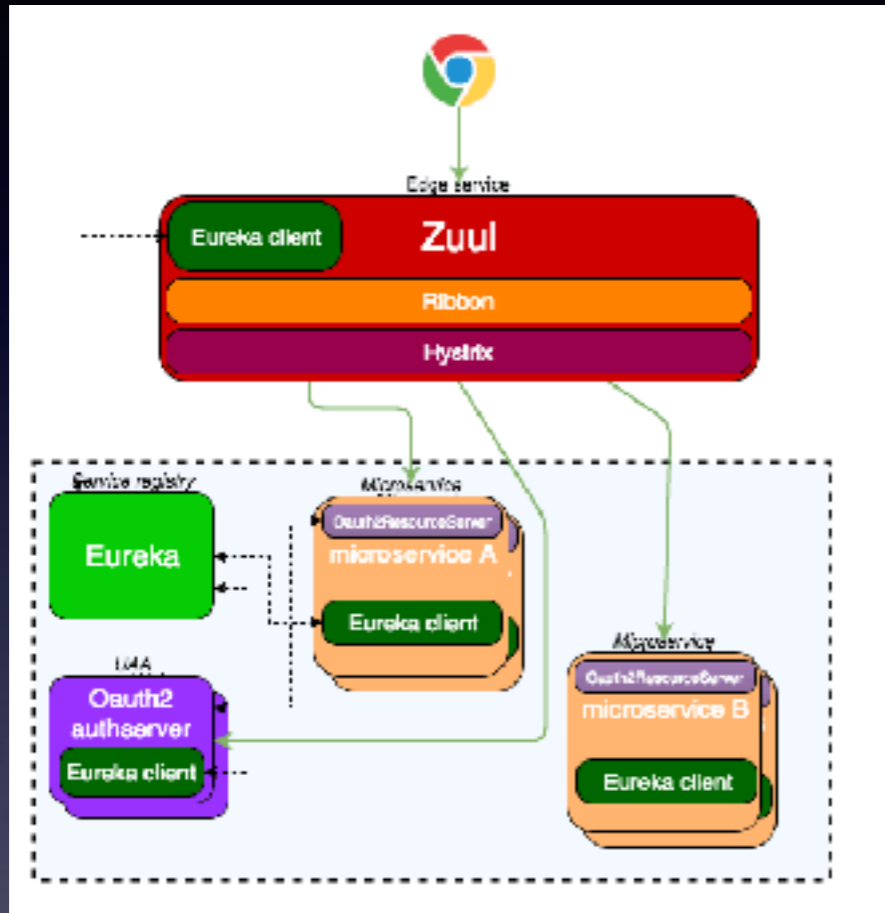


<https://github.com/Netflix/zuul>

Zuul的处理生命周期



Spring Cloud Zuul



@EnableZuulProxy

```
1 server:
2   port: 8040
3 spring:
4   application:
5     name: sc-zuul-first-zuul
6 eureka:
7   client:
8     service-url:
9       defaultZone: http://localhost:8761/eureka/
10 instance:
11   prefer-ip-address: true
12
13 zuul:
14   routes:
15     sc-zuul-first-provider: /order/**
16
17
```

```
<dependencies>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-zuul</artifactId>
  </dependency>
  <!-- 多了eureka starter -->
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-eureka</artifactId>
  </dependency>
</dependencies>
```

请求转发, 服务路由, 熔断

<http://xujin.org/sc/sc-zuul-01/>

请求转发

当注解为@EnableZuulProxy时，测试转发。通过访问网关的URL: <http://localhost:8041/api-url/sc/order/1> 可以正常的把请求的url转发到<http://localhost:8000/sc/order/2>

```
1 server.port=8041
2 spring.application.name=sc-zuul-first-zuul-no-eureka
3
4 zuul.routes.api-url.path=/api-url/**
5 zuul.routes.api-url.url=http://localhost:8000/
6
7
```

```
OrderController.java 33
17 @RestController
18 public class OrderController {
19
20     private static final Logger logger = LoggerFactory.getLogger(OrderController.class);
21
22     @Autowired
23     private OrderService orderService;
24
25     @GetMapping("/sc/order/{id}")
26     public OrderModel findOrderById(@PathVariable Long id) {
27         OrderModel orderModel = orderService.findOrderB
28         logger.info("Zuul路由到服务提供者①");
29         return orderModel;
30     }
31
```

localhost:8041/api-url/sc/order/1

```
{"orderNo":null,"createTime":null,"payTime":null}
```

默认路由规则

说明默认情况下，Zuul会代理所有注册到Eureka Server的微服务，并且Zuul的路由规则如下：

`http://ZUUL_HOST:ZUUL_PORT/微服务在Eureka上的serviceld/**` 会被转发到`serviceld`对应的微服务。

`http://localhost:8040/sc-zuul-first-provider/sc/order/2`



localhost:8040/sc-zuul-first-provider/sc/order/2

```
{"orderNo":2,"createTime":1493548213495,"payTime":null}
```

网关的负载均衡

<http://localhost:8040/sc-zuul-first-provider/sc/order/2>
通过网关访问**服务提供者**，负载均衡**打出对应的日志**

```
er [0;39m [2m:[0;39m Zuul路由到服务提供者①  
er [0;39m [2m:[0;39m Zuul路由到服务提供者②  
[0;39m [2m:[0;39m Resolving eureka endpoints via configuration
```

集成Hystrix

<http://localhost:8040/hystrix.stream>

```
localhost:8040/hystrix.stream

ping:

data: {"type":"HystrixCommand","name":"sc-zuul-first-
provider","group":"RibbonCommand","currentTime":1493548636014,"isCircuitBreakerOpen":false,"errorPercentage":0
CountEmit":0,"rollingCountExceptionsThrown":0,"rollingCountFailure":0,"rollingCountFallbackEmit":0,"rollingCou
lbackSuccess":0,"rollingCountResponsesFromCache":0,"rollingCountSemaphoreRejected":0,"rollingCountShortCircuit
rentExecutionCount":0,"rollingMaxConcurrentExecutionCount":0,"latencyExecute_mean":10,"latencyExecute":
{"0":10,"25":10,"50":10,"75":10,"90":10,"95":10,"99":10,"99.5":10,"100":10},"latencyTotal_mean":10,"latencyTot
{"0":10,"25":10,"50":10,"75":10,"90":10,"95":10,"99":10,"99.5":10,"100":10},"propertyValue_circuitBreakerReque
itBreakerErrorThresholdPercentage":50,"propertyValue_circuitBreakerForceOpen":false,"propertyValue_circuitBree
egy":"SEMAPHORE","propertyValue_executionIsolationThreadTimeoutInMilliseconds":1000,"propertyValue_executionTi
e_executionIsolationThreadPoolKeyOverride":null,"propertyValue_executionIsolationSemaphoreMaxConcurrentRequest
StatisticalWindowInMilliseconds":10000,"propertyValue_requestCacheEnabled":true,"propertyValue_requestLogEnabl

ping:

data: {"type":"HystrixCommand","name":"sc-zuul-first-
provider","group":"RibbonCommand","currentTime":1493548636506,"isCircuitBreakerOpen":false,"errorPercentage":0
CountEmit":0,"rollingCountExceptionsThrown":0,"rollingCountFailure":0,"rollingCountFallbackEmit":0,"rollingCou
lbackSuccess":0,"rollingCountResponsesFromCache":0,"rollingCountSemaphoreRejected":0,"rollingCountShortCircuit
rentExecutionCount":0,"rollingMaxConcurrentExecutionCount":0,"latencyExecute_mean":10,"latencyExecute":
{"0":10,"25":10,"50":10,"75":10,"90":10,"95":10,"99":10,"99.5":10,"100":10},"latencyTotal_mean":10,"latencyTot
{"0":10,"25":10,"50":10,"75":10,"90":10,"95":10,"99":10,"99.5":10,"100":10},"propertyValue_circuitBreakerReque
itBreakerErrorThresholdPercentage":50,"propertyValue_circuitBreakerForceOpen":false,"propertyValue_circuitBree
egy":"SEMAPHORE","propertyValue_executionIsolationThreadTimeoutInMilliseconds":1000,"propertyValue_executionTi
e_executionIsolationThreadPoolKeyOverride":null,"propertyValue_executionIsolationSemaphoreMaxConcurrentRequest
StatisticalWindowInMilliseconds":10000,"propertyValue_requestCacheEnabled":true,"propertyValue_requestLogEnabl
```


Spring-Cloud-Gateway

<https://github.com/spring-cloud-incubator/spring-cloud-gateway>

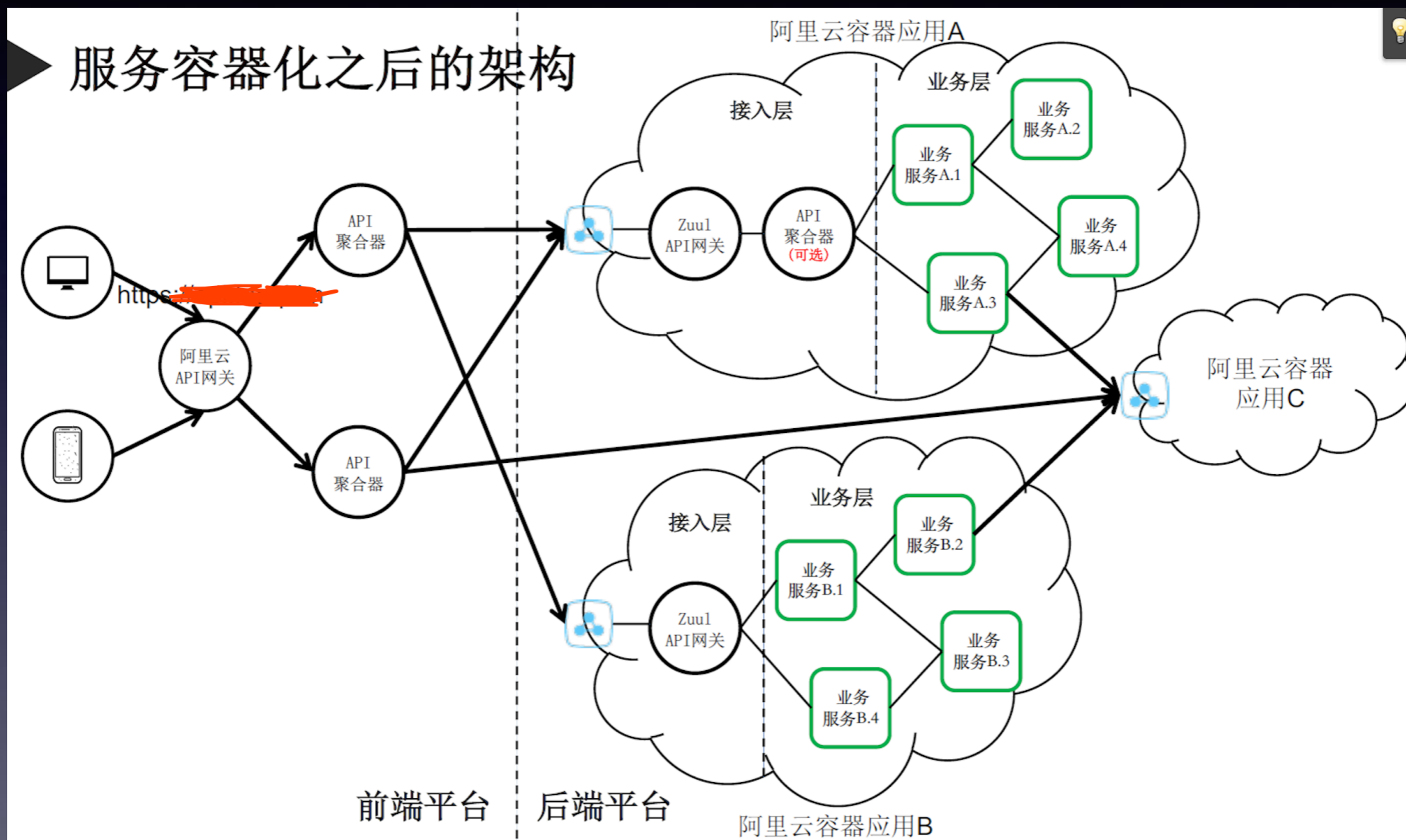
The screenshot shows the GitHub repository page for `spring-cloud-incubator / spring-cloud-gateway`. The repository is described as "A Gateway built on **Spring Framework 5.0** and **Spring Boot 2.0** providing routing and more." The repository statistics show 200 commits, 5 branches, 0 releases, and 1 contributor. The commit history shows a recent commit by `spencerglbb` with the message "adds .editorconfig". The commit details list several files and their changes:

File	Change
<code>.mvn</code>	Updates for reactor-bom Bismuth
<code>docs</code>	Move back to Predicate<ServerWebExchange>
<code>spring-cloud-gateway-core</code>	Fix broken test
<code>spring-cloud-gateway-dependencies</code>	Upgrade spring-cloud libs to 2.0.0
<code>spring-cloud-gateway-sample</code>	Add todo
<code>spring-cloud-starter-gateway</code>	adds spring.provides to starter

The screenshot shows the GitHub repository page for `Netflix / zuul`. The repository is described as "Zuul is a gateway service that provides dynamic routing, monitoring, resiliency, and more." The repository statistics show 296 commits, 12 branches, 45 releases, and 1 contributor. The commit history shows a recent commit by `mikeycohen` with the message "Merge pull request #312 from hdurix/readme-ippon-link". The commit details list several files and their changes:

File	Change
<code>codequality</code>	Remove unneeded files
<code>gradle</code>	Move to gradle 2.6 and nebula.netflixoss 3.0.0
<code>zuul-core</code>	Update SurgicalDebugFilter.groovy
<code>zuul-netflix-webapp</code>	Fix Javadoc issues: Reformat dangling comments
<code>zuul-netflix</code>	Fix for unit-test breakage due to admin.enabled prop
<code>zuul-simple-webapp</code>	Change the default origin in zuul-simple-webapp to be

阿里云中使用网关



那这样大家是否就觉得OK了?

为什么要自研网关

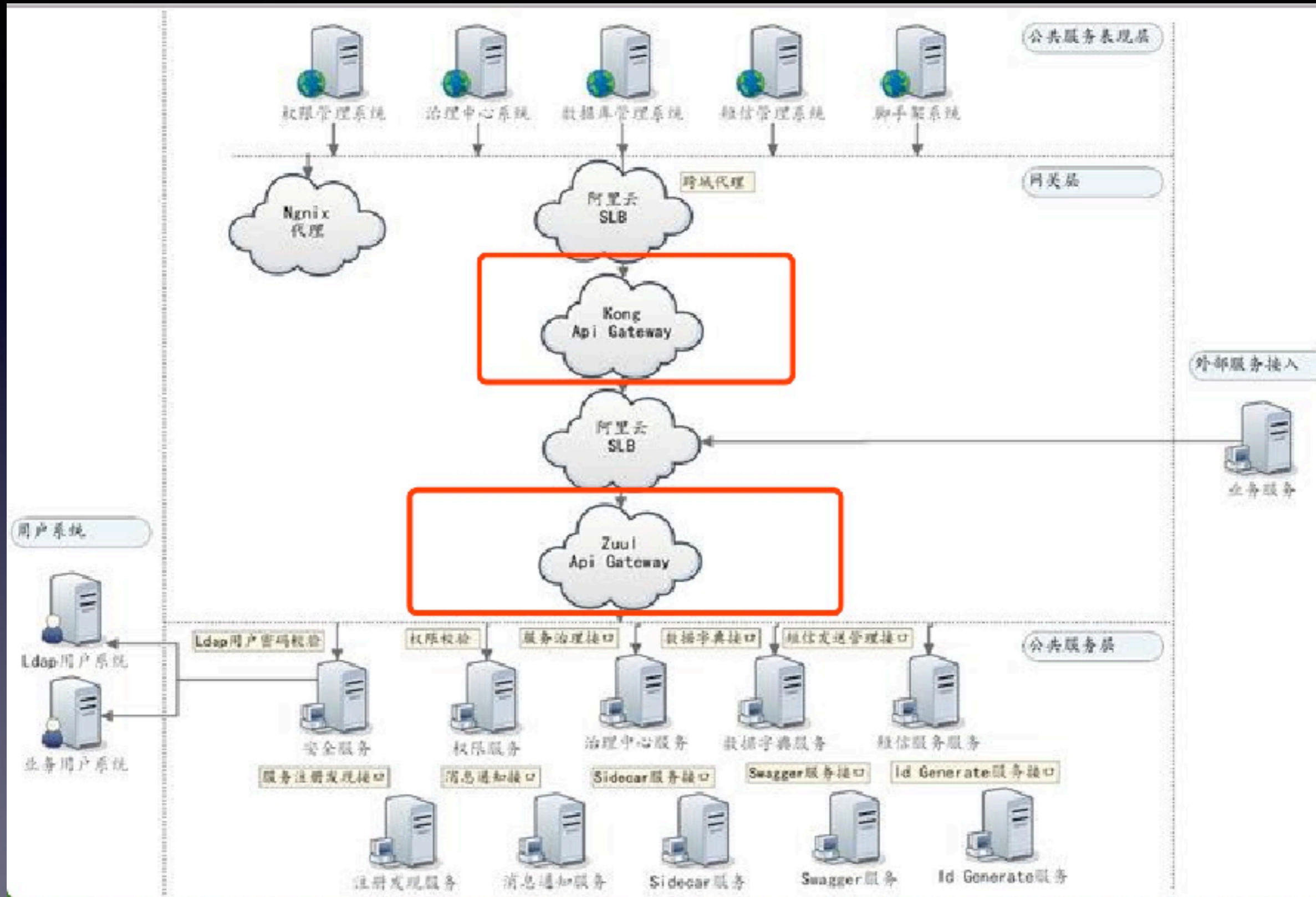
- 1.网关配置**实时生效**，配置灰度，回滚等
- 2.网关的**性能**，特别是**防刷，限流，WAF**等
- 3.动态Filter，目前Zuul可以做到动态Filter，Filter配置下发，**实时动态Filter**
- **4.对网关的监控，告警，流量调拨，网关集群。**
- 5.**流程审计**，增加**Dsboard**便捷的操作。

如何自研网关？

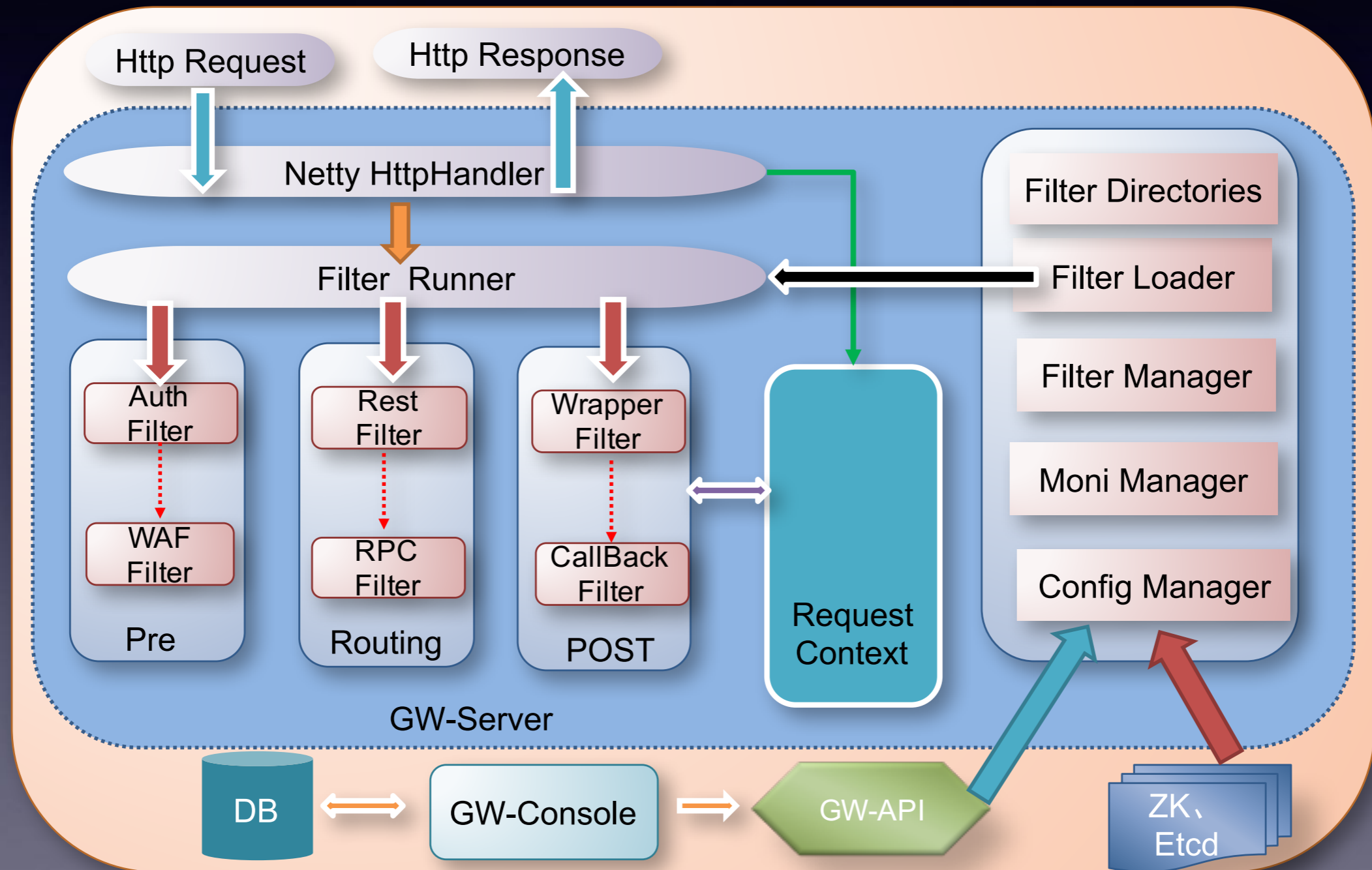
“架构如何设计？性能怎么样？HA怎么保证，一丢丢的问题出来”。



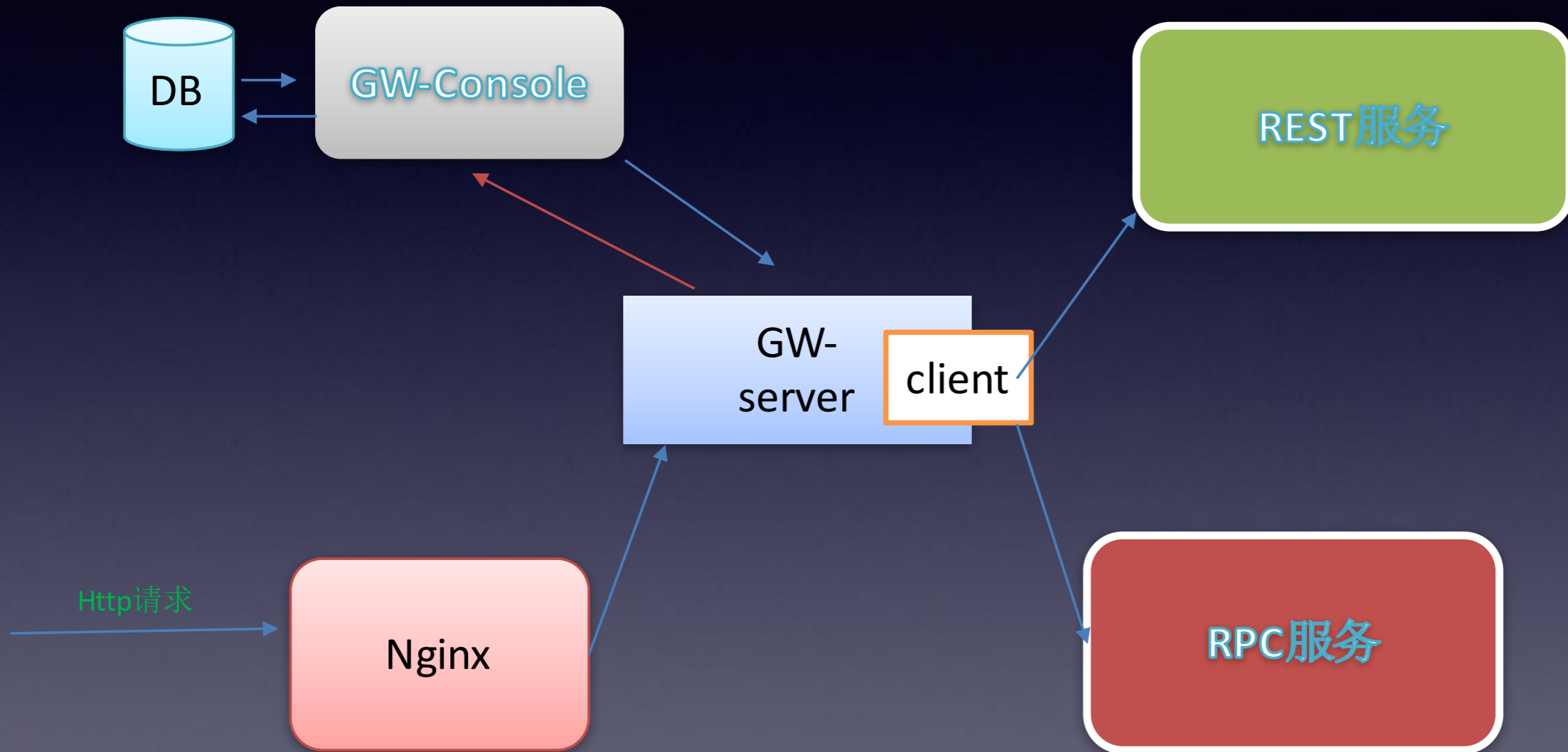
复杂的部署结构



设计-基于Netty的GW架构

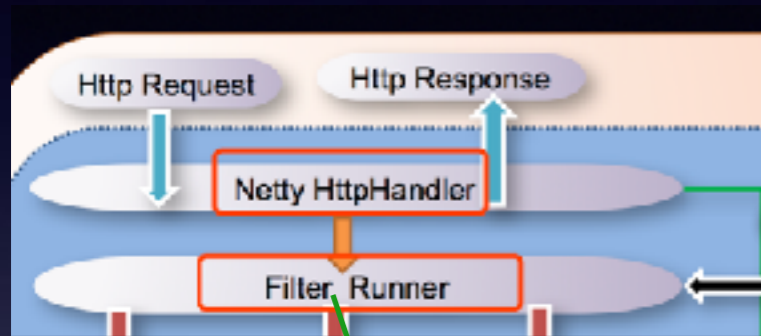


GW数据交互



设计-源码实现

RequestContext



```
@Sharable
public class HttpConnectionHandler extends ChannelInboundHandlerAdapter {

    |

    public HttpConnectionHandler() {

    }

    @Override
    public void channelRead(ChannelHandlerContext ctx, Object msg) throws Exception {
```

```
public class JanusRunner {
    private static Logger logger = LoggerFactory.getLogger(JanusRunner.class);

    public static void run(SessionContext context) {
        try {
            DefaultFilterPipeline.getInstance().get(HttpProtocolCheckFilter.DEFAULT_NAME).fireSelf(context);
        } catch (Throwable e) {
            errorProcess(context, e);
        }
    }
}
```


网关与SC服务治理整合

```
@SpringBootApplication
@EnableDiscoveryClient
public class JanusServerAppliacion {

    private static Logger logger = LoggerFactory.getLogger(JanusServerAppliacion.class);

    // 非SSL的监听HTTP端口
    public static int httpPort = 8081;

    public static void main(String[] args) throws Exception {

        ConfigurableApplicationContext context = SpringApplication.run(JanusServerAppliacion.class, args);

        logger.info("services: {}", context.getBean("discoveryClient", DiscoveryClient.class).getServices());

        logger.info("Gateway Server Application Start...");
        // 解析启动参数
        parseArgs(args);

        // 初始化网关Filter和配置
        logger.info("init Gateway Server ...");
        JanusBootStrap.initGateway();

        logger.info("start netty Server...");
        final JanusNettyServer gatewayServer = new JanusNettyServer();
        // 启动HTTP容器
        gatewayServer.startServer(httpPort);

    }
}
```

网关与SC服务治理整合

```
JanusServerApplication.java application.properties
1
2 server.port=8082
3
4 spring.application.name=janus-server
5
6 #eureka.client.service-url.defaultZone=http://mse.zhonganonline.com:8761/eureka
7
8 eureka.client.service-url.defaultZone=http://localhost:8761/eureka/
9
10 configpath=/apps/conf/janus/
11
12 janus_console_url=janus-cluster.api.xujin.com
13
14 janus_return_ok_url=/do_not_delete/noc.gif;/favicon.ico;
15
16
```

基于Netty的请求入口

```
/**
 * 从ChannelRead做读取请求数据,放入Janus Server处理的上下文
 */
@Override
public void channelRead(ChannelHandlerContext ctx, Object msg) throws Exception {
    /**
     * DefaultFullHttpRequest(decodeResult:
     * failure(io.netty.handler.codec.TooLongFrameException: An HTTP line is larger
     * than 10 bytes.), version: HTTP/1.0, content: EmptyByteBufBE) GET /bad-request
     * HTTP/1.0 如果http协议报错的处理
     */
    JanusRequest request = null;
    JanusHandlerContext context = null;
    try {
        FullHttpRequest fullHttpRequest = (FullHttpRequest) msg;
        /**
         * 检查path,netty中为uri,判断http头,如果是http开头,说明path为:
         * http://mapi.abc.com:80/hello?fields=1. 该处需要做两件事情: 1: 将path变成/hello?fields=1
         * 2:将http header的host设置为mapi.abc.com:80
         */
        String uri = fullHttpRequest.getUri();
        if (StringUtils.startsWithIgnoreCase(uri, "http")) {
            int pos = StringUtils.indexOf(uri, "://");
            if (pos != -1 && pos < 5) {
                int slashPos = StringUtils.indexOf(uri, "/", pos + 3);
                fullHttpRequest.setUri(StringUtils.substring(uri, slashPos));
                HttpHeaders.setHost(fullHttpRequest,
                    StringUtils.substring(uri, pos + 3, slashPos));
            }
        }
        // 设置channel
        context = new JanusHandlerContext();
        context.setInboundChannel(ctx.channel());

        // 设置request
        request = new NettyJanusRequest(fullHttpRequest, ctx.channel(), context);
        context.setRequest(request);

        // netty未对参数的非法性进行校验,这里增加对参数的非法性进行校验,避免错误的参数透传到后端。
    } catch (Exception e) {
        // ...
    }
}
```

Filter责任链的设计

```
public interface Filter {
    public String name();

    public void init();

    public void run(AbstractFilterContext filterContext, SessionContext sessionContext) throws Exception;

    public boolean isValid();
}
```

```
public abstract class AbstractFilter implements Filter {
    public final static String PRE_FILTER_NAME = "JANUS_FILTER_";
    protected boolean isValid = true;

    @Override
    public void init() {
        isValid = true;
    }

    @Override
    public void run(AbstractFilterContext filterContext, SessionContext sessionContext) throws Exception {
        filterContext.fireNext(sessionContext);
    }

    @Override
    public boolean isValid() {
        return isValid;
    }

    public void setValid(boolean isValid) {
        this.isValid = isValid;
    }
}
```

Filter数据的CRUD

```
public abstract class EventAbstractFilter<T> extends AbstractFilter implements AbstractFilterObserver<T> {  
  
    @Override  
    public void add(Long resourceId, T value) {  
  
    }  
  
    @Override  
    public void delete(Long resourceId, T value) {  
  
    }  
  
    @Override  
    public void update(Long resourceId, T value) {  
  
    }  
  
    @Override  
    public void start() {  
  
    }  
  
    @Override  
    public void complete() {  
  
    }  
  
}
```

每个Filter有自己的缓存数据，
缓存数据的CRUD，通过观察者模式按key更新

示例WAF-Filter

```
public class WafSafeFilter extends EventAbstractFilter<RuleV0> {
    private static final Logger logger = LoggerFactory.getLogger(WafSafeFilter.class);
    public static String DEFAULT_NAME = PRE_FILTER_NAME
        + WafSafeFilter.class.getSimpleName().toUpperCase();

    private Map<Long, Trie> whiteTireMap = new ConcurrentHashMap<Long, Trie>();
    private Map<Long, Trie> blackTireMap = new ConcurrentHashMap<Long, Trie>();
    private Map<Long, Boolean> blackStatusMap = new ConcurrentHashMap<Long, Boolean>();

    public static String className = WafSafeFilter.class.getSimpleName();
    public static String classMethod = "run";

    @Override
    public String name() {
        return DEFAULT_NAME;
    }

    @Override
    public void init() {
        setValid(EnvUtil.getBoolean(name(), "true"));
    }

    @Override
    public void run(AbstractFilterContext filterContext, SessionContext sessionContext) {
        // 获取Ip
        String ip = sessionContext.getRequest()
            .getHeader(JanusProduceConstants.X_FORWARDED_FOR);
        // 获取当前请求的域名id
    }
}
```

异步转发

```
@Override
public void run(final AbstractFilterContext filterContext,
    final JanusHandleContext janusHandleContext) throws Exception {
    ServiceInstance serviceInstance = SpringCloudHelper.getServiceInstanceByLB(
        janusHandleContext, janusHandleContext.getAPIInfo().getRouteServiceId());
    // 生成发送的Request对象
    FullHttpRequest outboundRequest = getOutBoundHttpRequest(janusHandleContext);
    HttpHeaders headers = outboundRequest.headers();
    String address = headers.get(HttpHeaders.Names.HOST);
    AsyncHttpRequest.builder()
        .remoteAddress(
            serviceInstance.getHost() + ":" + serviceInstance.getPort())
        .sessionContext(janusHandleContext).httpMethod(HttpMethod.GET)
        .uri("/sc/order/2")
        /**
         * connection holding 500ms
         */
        .holdingTimeout(PropertyConfig.janusHttpPoolOauthMaxHolding).build()
        .execute(new SimpleHttpCallback(janusHandleContext) {
            @Override
            public void onSuccess(FullHttpResponse result) {
                // testResult(result);
                janusHandleContext.setRestFullHttpResponse(result);
                // 跳转到下一个Filter
                filterContext.fireNext(janusHandleContext);
            }
        });

    @Override
    public void onError(Throwable e) {
        janusHandleContext.setResponseHttpCode(HttpStatusCode.HTTP_BAD_GATEWAY);
        REST_INVOKER_ERROR_EXCEPTION.setMessage(e.getMessage());
        JanusFilterRunner.errorProcess(janusHandleContext,
            REST_INVOKER_ERROR_EXCEPTION);
    }

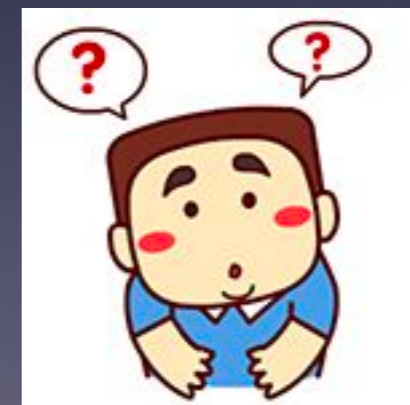
    @Override
    public void onTimeout() {
        janusHandleContext
            .setResponseHttpCode(HttpStatusCode.HTTP_GATEWAY_TIMEOUT);
    }
}
```

网关启动初始化Filter

```
private static void initJavaFilters() {  
    // 前置Filter  
    DefaultFilterPipeline.getInstance().addLastSegment(new HttpProtocolCheckFilter(),  
        new ApiInfoMappingFilter(), new WafSafeFilter(),  
        new MapiAuthSignatureFilter(), new ProtocolAdaptorFilter());  
    // RPC Filter  
    DefaultFilterPipeline.getInstance().addLastSegment();  
  
    // rest Filter  
    DefaultFilterPipeline.getInstance().addLastSegment(new RestRequestUriFilter(),  
        new RestRequestBodyFilter(), new RestRequestHeaderFilter(),  
        new RestInvokerFilter(), new RestResponseHeaderFilter());  
  
    // 后置流程  
    DefaultFilterPipeline.getInstance().addLastSegment(  
        new CommonResponseHeaderFilter(), new ResponseSendFilter());  
  
    // Error Filter  
    DefaultFilterPipeline.getInstance().addLastSegment(new ErrorFilter());  
  
    // 注册监听事件  
    for (Filter filter : DefaultFilterPipeline.getInstance().getAllFilter()) {  
        filter.init();  
        if (filter instanceof EventAbstractFilter) {  
            EventAbstractFilter eventFilter = (EventAbstractFilter) filter;  
            if (eventFilter.getPublishType() != null) {  
                ConfigManager.getInstance().addObserver(eventFilter);  
            }  
        }  
    }  
}
```

```
filterContext.fireNext(sessionContext);|
```

观察者模式处理



动态Filter

```
DefaultFilterPipeline.getInstance().get(ErrorFilter.DEFAULT_NAME).fireSelf(sessionContext);
```


网关设计原则

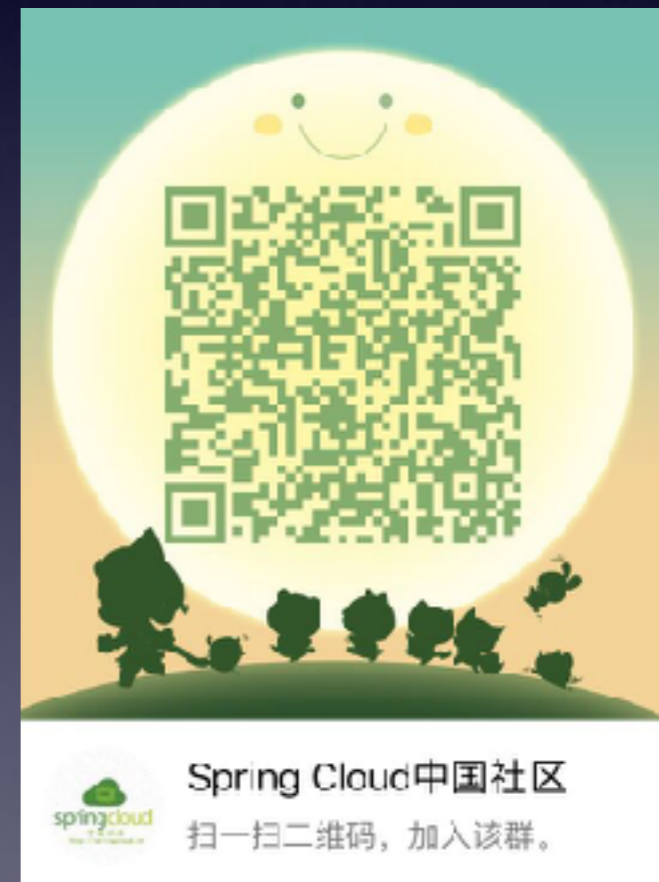
- 1.每个Filter基于**责任链**，只做专一的一件事
- 2.每个Filter有各自**独立**的数据
- 3.损耗性能的Filter**顺序往后放**
- 4.启动读取配置顺序，先**远端**，若**远端失败**，则**读取本地**。
- 5.集群网关，要注意**数据的diff和灰度**
- 6.尽量做到和服务治理**框架解耦**，易于**接入**，易于**升级**

Q&A

欢迎加我微信互撩



微信号



社区QQ群