



Aliware DUBBO Brand

dubbo-go实践分享 tuya open-gateway篇

Speaker : panty

目 录

CONTENT

01

选择dubbo-go
(需求由来)

02

gateway设计

03

gateway实现
(踩坑)

04

成果和展望

1

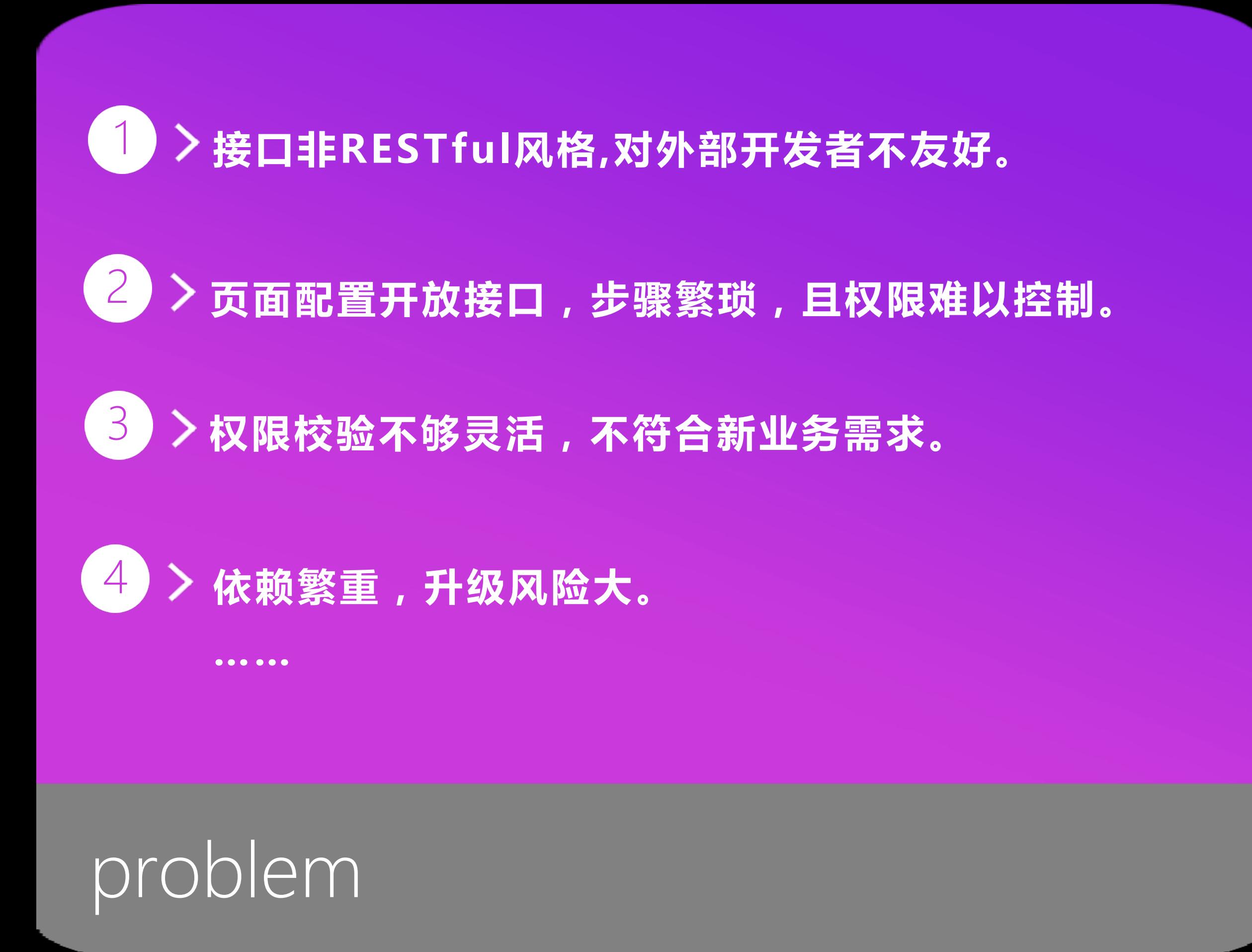
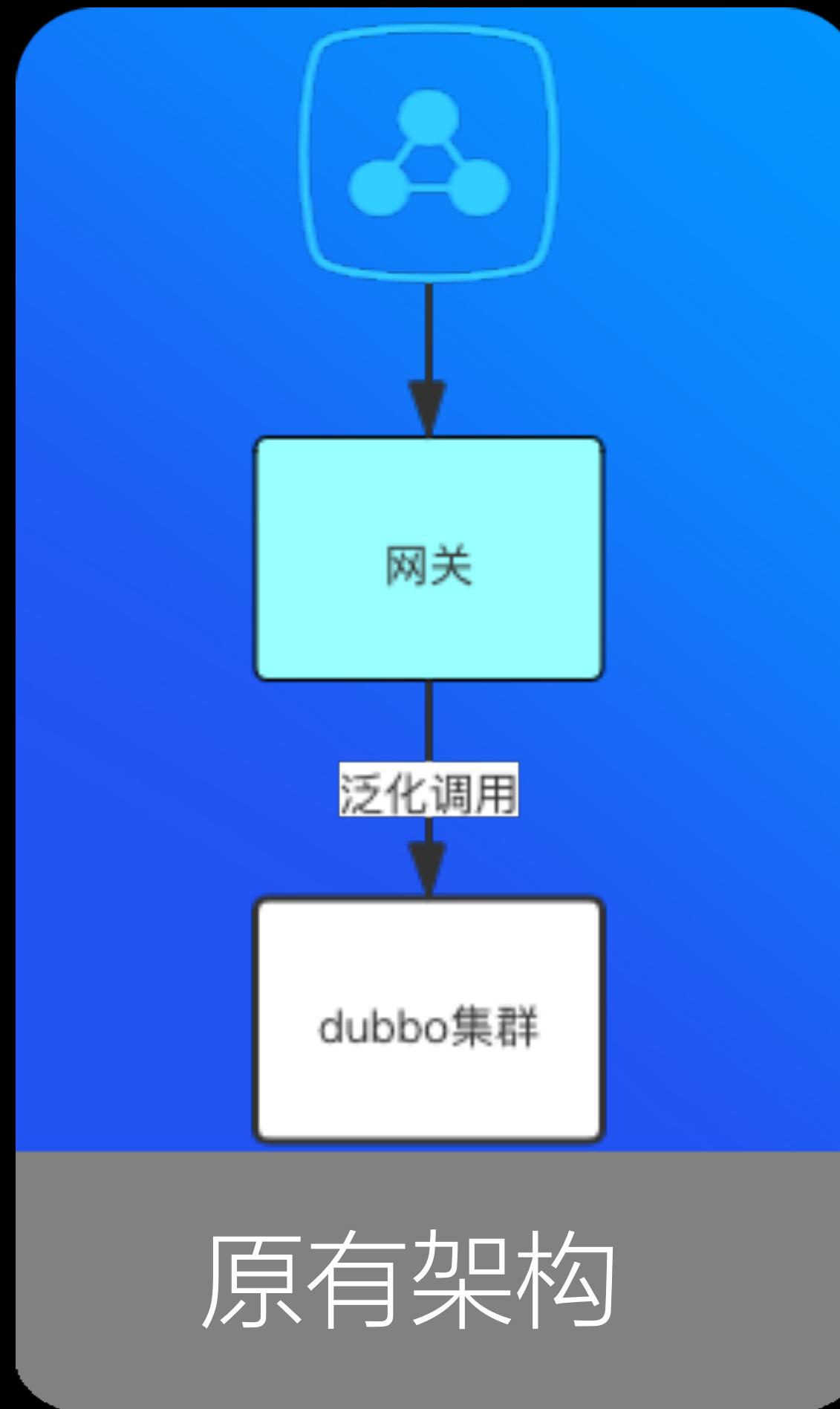
选择dubbo-go



总体背景：

19年年初，公司已有重量级dubbo集群，想引入go语言做一些中间件工作，如定时、消息队列、网关等等。急需一种方案能够使go服务无缝接入目前的业务集群。从那时起就比较关注dubbo-go项目。

当时有多种方案，也已经有一些go项目通过grpc做了起来。但是对已有java集群的侵入代价较大，一些项目难以推广。



- **open-gateway**希望达到的愿景：
 - 1.通过一个golang的项目闭环，搭建公司go开发上线基础环境，沉淀一些go的公共组件，并且验证go服务无侵入加入java集群的可能性。
 - 2.通过该网关，暴露接口。能做到最少浸入，最少配置，解决目前暴露的问题和需求。
 - 3.go编码的网关能从性能上对现有项目有所突破。

- 选择**dubbo-go**的理由：

- 1.项目需求最优的选择。
- 2.dubbo-go社区活跃。
- 3.本人对go喜爱



2

gateway设计



01

simple

02

security

03

High Availability

○ 减少配置和依赖

在满足需求基础上尽可能的减少依赖，借鉴dubbo管理思想，做到部署方便、使用便捷。

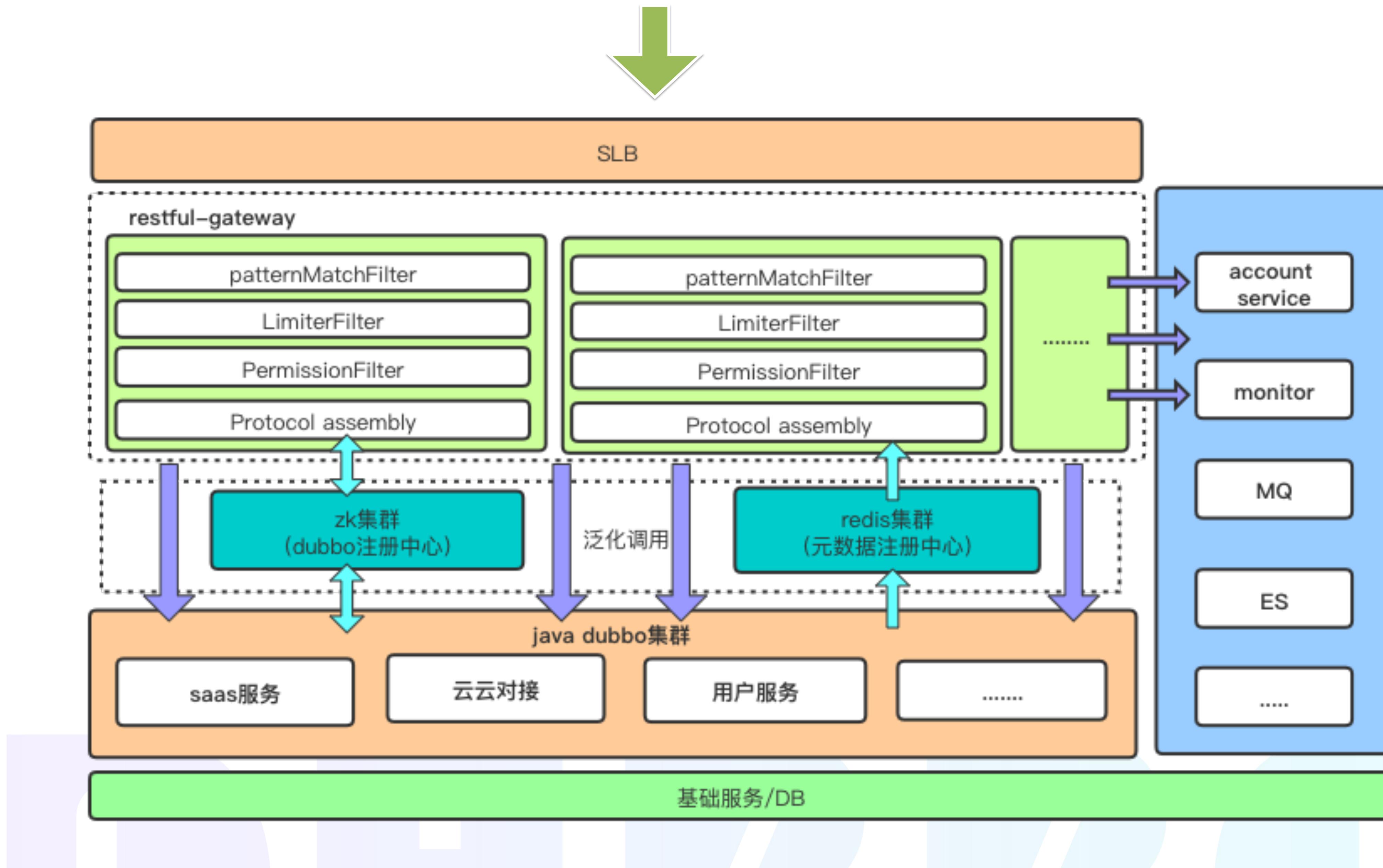
○ 灵活权限校验、安全扫描

启动时增加敏感信息扫描，权限策略一接口一配置，灵活、安全。

○ 高可用

集群部署，本地持久化注册信息。支持各种降级策略。dubbo-go注册中心调优。

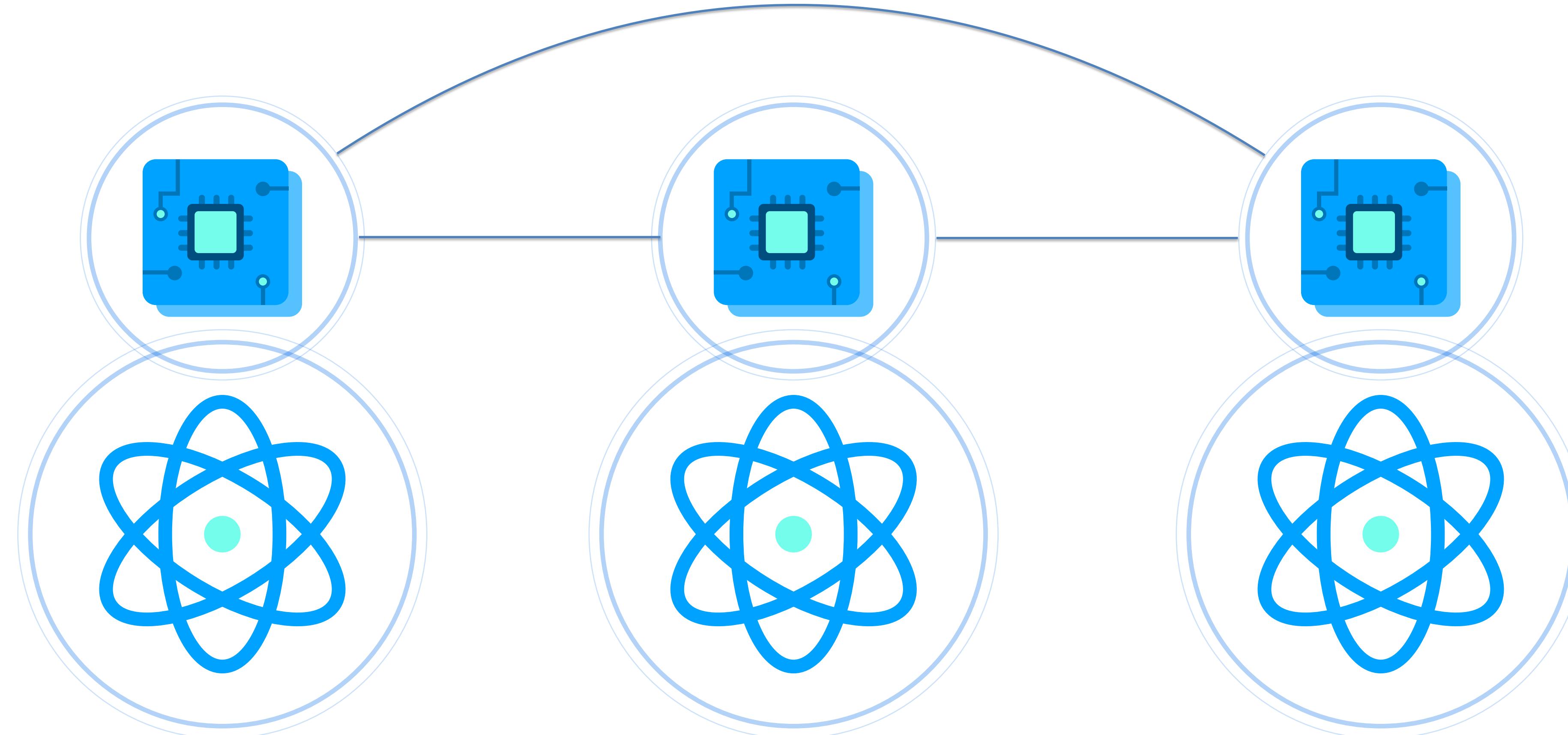
外部调用方



多区专线打通，网关提供指定区域调用api

网关

dubbo集群



3

gateway实现



针对该网关项目对dubbo-go优化（踩坑）：

01 实现dubbo-go泛化调用

网关系统建立在泛化调用的基础上，此项必须要实现的功能。

02 提升hessian-go对java支持

开发中，遇到大量序列化适配问题。感谢社区同学们帮助扫除障碍。目前[apache/dubbo-go-hessian2](#)已经很好的支持了dubbo协议中java和go的交互。

03 加强服务治理可靠性

项目使用zk注册中心，开发期间发现并解决数个dubbo-go的zk模块的bug和可优化点。目前运行良好，以实际验证了生产环境的可用性。

04 性能调优

dubbo-go transport层使用getty提供读写分离的连接池。在项目开发测试过程中，检测并修复缺陷、并通过经验调参，优化并发性能。

05 监控、限流

目前社区伙伴正在从事的工作，这也是提升服务可靠性不可缺少的一环。

泛化调用的实现：generic_filter

```
func (ef *GenericFilter) Invoke(invoker protocol.Invoker, invocation protocolInvocation) protocol.Result {  
    if invocation.MethodName() == constant.GENERIC && len(invocation.Arguments()) == 3 {  
        oldArguments := invocation.Arguments()  
  
        if oldParams, ok := oldArguments[2].([]interface{}); ok {  
            newParams := make([]hessian.Object, len(oldParams))  
            for i := range oldParams {  
                newParams = append(newParams, hessian.Object(struct2MapAll(oldParams[i])))  
            }  
            newArguments := []interface{}{  
                oldArguments[0],  
                oldArguments[1],  
                newParams,  
            }  
            newInvocation := invocation2.NewRPCInvocation(invocation.MethodName(), newArguments, invocation.Attachments())  
            newInvocation.SetReply(invocation.Reply())  
            return invoker.Invoke(newInvocation)  
        }  
    }  
    return invoker.Invoke(invocation)  
}
```

递归，将参数中的struct
全部转化为map

使用示例

依赖准备

```
<dependency>
  <groupId>com.tuya.gateway</groupId>
  <artifactId>gateway-client</artifactId>
  <version>1.0-SNAPSHOT</version>
</dependency>
```

加入spring bean 网关highway路径扫描

```
<context:component-scan base-package="com.tuya.gateway"/>
```

```
public interface IHelloService {
    @PathMapping("/user/sayHello")
    String sayHello(String uid); //测试token的uid

    @PathMapping(value = "/user/name", method = RequestMethod.POST)
    User getUser(String uid, String name); //会自动抓取http, body和url中获取到的参数

    @PathMapping(value = "/user/void", method = RequestMethod.POST)
    void sleep(User user); //从http请求中抓取类中的属性，拼装成vo

    @PathMapping(value = "/user/nopara")
    User getNoPara(); //返回值默认能将vo对象的驼峰属性名转换成下划线返回对饮的json字段

    //请求中有内置字段的同名字段，获取该字段需要特殊配置，如例子中uid为内置字段，而程序需要获取的是请求中的字段
    @PathMapping(value = "/user/sayHello/{uid}", tokenUidIsNeed = false)
    User sayHellobyUid(String uid);

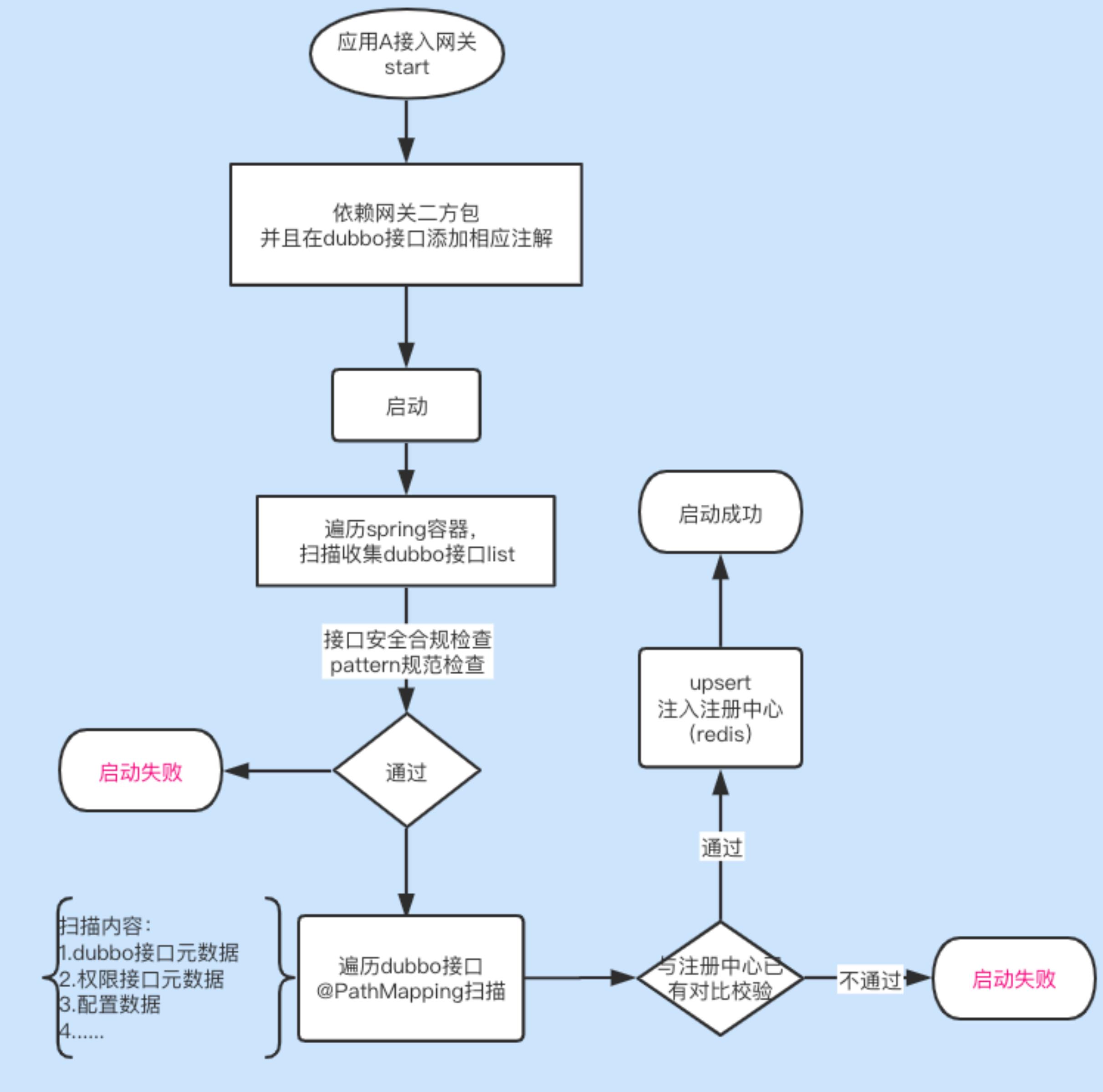
    @PathMapping(value = "/user/context")
    User context(String uid, Context context); //网关提供网关上下文字段。
}
```



```
@PermissionFilter(value = IHelloService.class, biz = "user", role = "youke", version = "1.0.0"
    authParam = {@AuthParam(value = "gid,groupid", position = PositionEnum.BODY),
    @AuthParam(value = "clientId", position = PositionEnum.HEADERS),
    @WhiteParam(value = "Yd", paramName = "name", position = PositionEnum.BODY)})
@PathMapping(value = "/user/name", method = RequestMethod.POST)
User getUser(String uid, String name);
```

业务应用启动流程：

```
@Service  
public class DubboStarterListener implements ApplicationListener<ContextRefreshedEvent>  
{  
    @Autowired  
    private ApplicationContextUtil applicationContextUtil;  
  
    @Override  
    public void onApplicationEvent(ContextRefreshedEvent contextRefreshedEvent) {  
        List<UrlPathInfo> urlPathInfoList = ApplicationContextUtil.getUrlPathInfos();  
        CacheDubboUtil.cacheUrlPath(urlPathInfoList);  
    }  
}
```



注册信息举例：

```
{  
    "key": "POST:/v1.0/hello/{uid}/add",  
    "interfaceName": "com.tuya.hello.service.template.IUserServer",  
    "methodName": "addUser",  
    "parameterTypes": ["com.tuya.gateway.Context", "java.lang.String", "com.tuya.hello.User"],  
    "parameterNames": ["context", "uid", "userInfo"],  
    "updateTimestamp": "1234567890",  
    "permissionDO": {  
        "biz": "base",  
        "role": "DEFAULT",  
        "methodName": "validate",  
        "interfaceName": "com.tuya.hello.service.IPermissionServer",  
        "version": "1.0.0",  
        "timeout": 5000  
    },  
    "voMap": {  
        "userInfo": {  
            "name": "java.lang.String",  
            "sex": "java.lang.String",  
            "age": "java.lang.Integer"  
        }  
    },  
    "parameterNameHumpToLine": true,  
    "resultFiledHumpToLine": false,  
    "highwayContextIsNeed": false,  
    "tokenUidIsNeed": false,  
    "protocolName": "dubbo"  
}
```



注册信息由接入网关的provider应用启动时写入redis



网关通过定时更新、Redis Pubsub等等机制获取到注册信息



网关可通过Redis Pubsub信号写入本地磁盘，做容灾备份，防止注册中心奔溃。

调参 : client.yml

```
protocol_conf:  
dubbo:  
    reconnect_interval: 0  
    connection_number: 2  
    heartbeat_period: "30s"  
    session_timeout: "180s"  
    fail_fast_timeout: "5s"  
    pool_size: 4  
    pool_ttl: 600  
    # gr_pool_size is recommended to be set to [cpu core number] * 100  
    gr_pool_size: 200  
    # queue_len is recommended to be set to 64 or 128  
    queue_len: 64  
    # queue_number is recommended to be set to gr_pool_size / 20  
    queue_number: 10  
    getty_session_param:  
        compress_encoding: false  
        tcp_no_delay: true  
        tcp_keep_alive: true  
        keep_alive_period: "120s"  
        tcp_r_buf_size: 262144  
        tcp_w_buf_size: 65536  
        pkg_rq_size: 1024  
        pkg_wq_size: 512  
        tcp_read_timeout: "1s" #1  
        tcp_write_timeout: "5s" #5  
        wait_timeout: "5s"  
        max_msg_len: 1024000  
        session_name: "client"
```

getty session的可拥有连接数

session空闲关闭时间

gettyRPCClientConnPool大小

也就是每一个DubboInvoker拥有session的数量

每次读取单包长度

每次写入单包长度

分包读取单个包读取超时时间

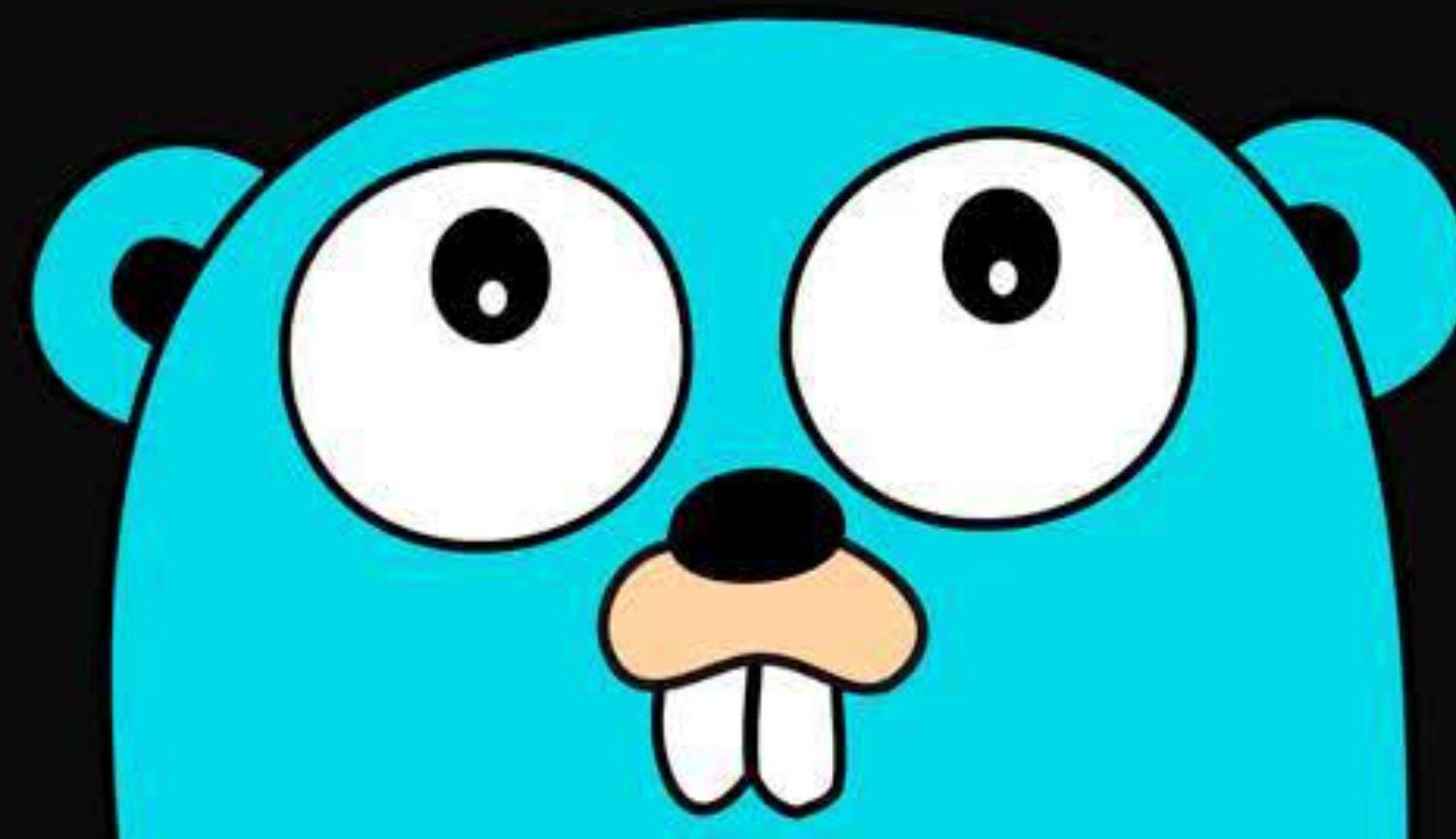
分包写单个包写入超时时间

单词请求最大传递数据长度

4

成果和展望





1000W+

网关服役2个月
日流量破千万

2000+

压测网关API单台2c8G机器
QPS可达2000+

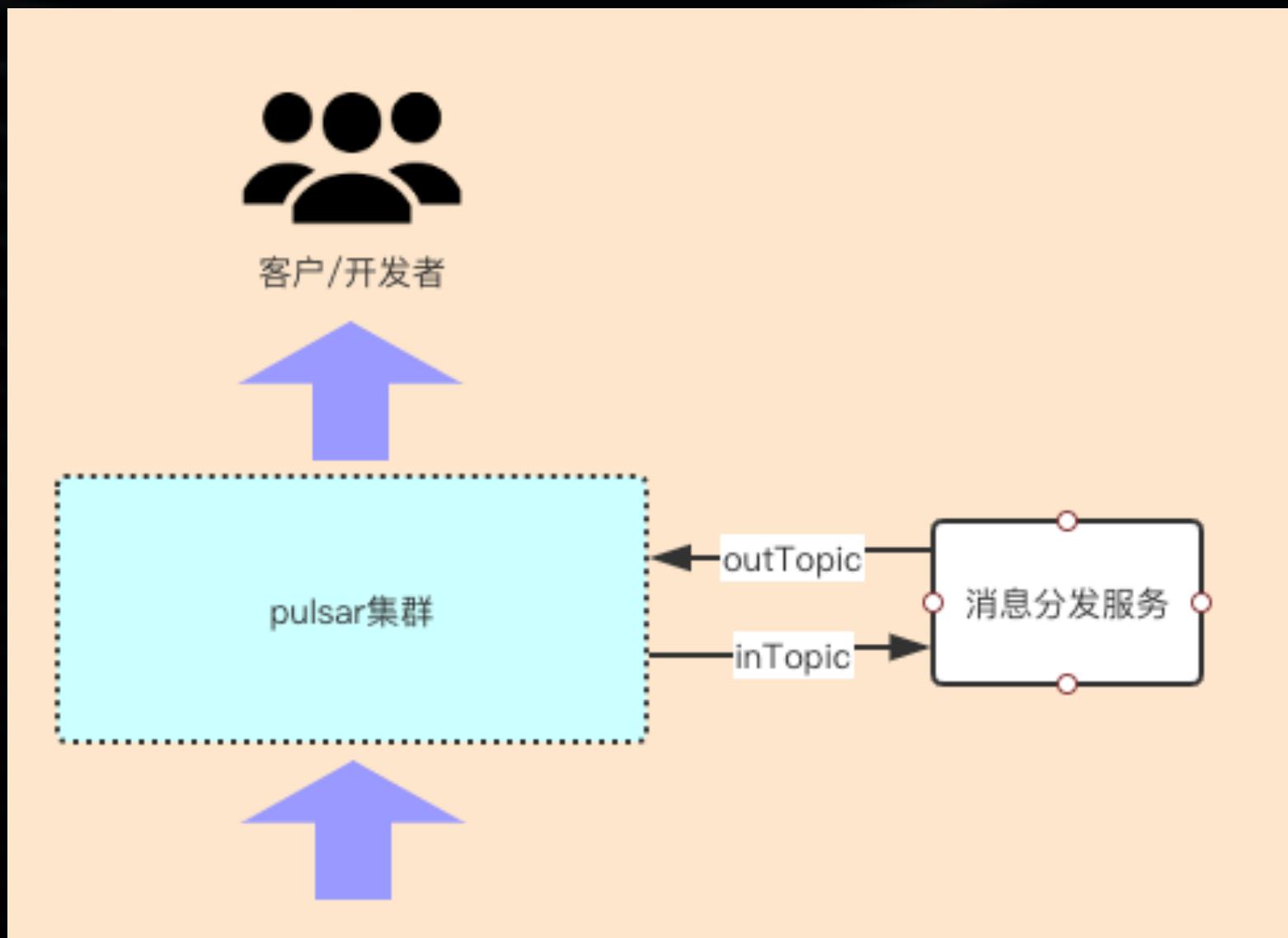
4000W+

网关集群日均dubbo调
用次数超过4000W

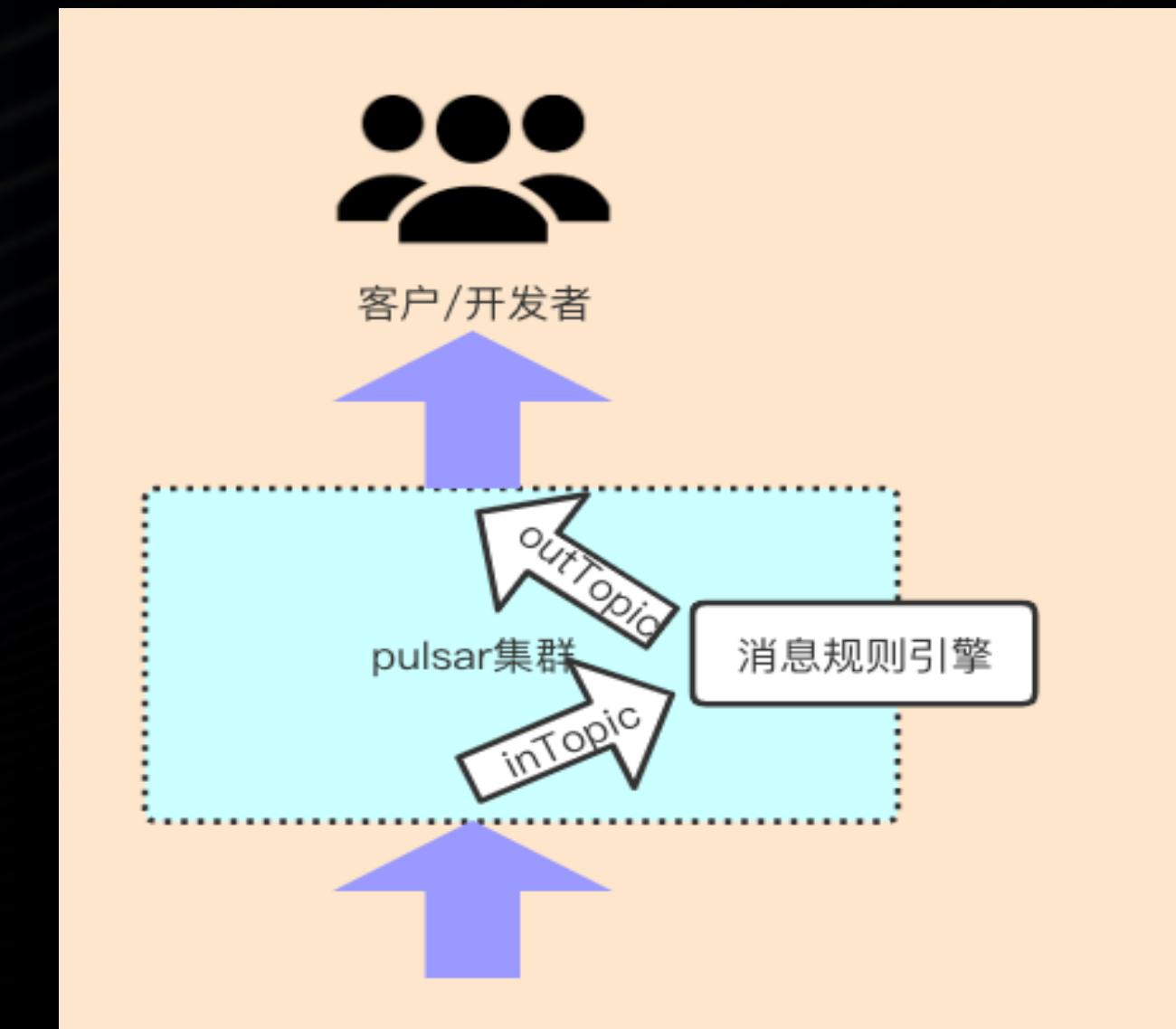
1000+

暴露restful接口
超过1000个

更多的dubbo-go场景的尝试



原外部消息推送架构



正在重构的推送框架

dubbo-go展望

01

可靠

作为基础rpc框架，稳定可靠永远是第一位。

02

灵活

能适配更多的使用场景，将社区的活跃转化为使用者的热度

03

顺势

积极的向最新的技术理念靠拢，顺应大众的需求趋势

04

特色

dubbo-go相对于java dubbo以及其他 rpc 框架，需要引入自己的场景特色



thank you !