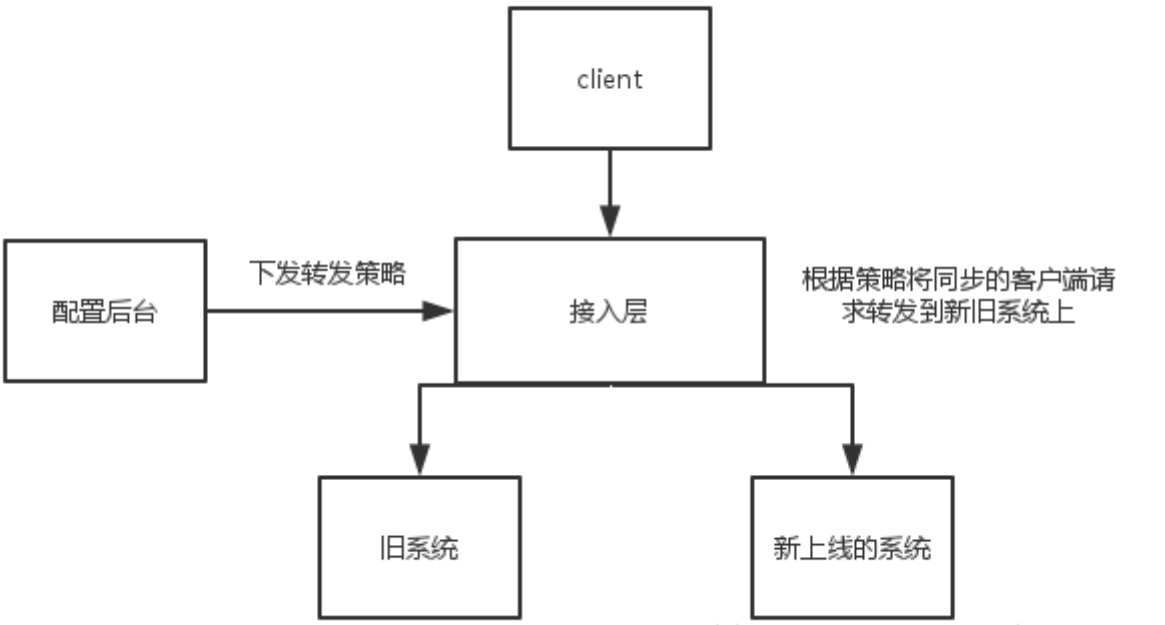
灰度发布方案

一、灰度发布定义

灰度发布（又名金丝雀发布）是指在黑与白之间，能够平滑过渡的一种发布方式。在其上可以进行A/B testing，即让一部分用户继续用产品特性A，一部分用户开始用产品特性B，如果用户对B没有什么反对意见，那么逐步扩大范围，把所有用户都迁移到B 上面来。灰度发布可以保证整体系统的稳定，在初始灰度的时候就可以发现、调整问题，以保证其影响度。



二、实现思路方向

1、在代码中做。

一套线上环境，代码中做开关，对于不同的用户走不同的逻辑

2、在接入层做。

多套(隔离的)线上环境，接入层针对不同用户转发到不同的环境中

两种方案的优缺点：

方案	优点	缺点
在代码中做	灵活，粒度细；一套代码(环境)运维成本低	灰度逻辑侵入代码
在接入层做	无需（少）侵入代码；风险小	多套线上环境，运维成本高

灵活的灰度方案一般需要在接入层实现，具体就是自定义负载均衡策略实现。

下面介绍在接入层使用的方式，第一是在nginx层实现（使用ngx+lua），第二是在网关层实现（spring-cloud-zuul）。

第三是dubbo的灰度，项目中如果使用dubbo，有可能会需要dubbo服务的灰度实现。

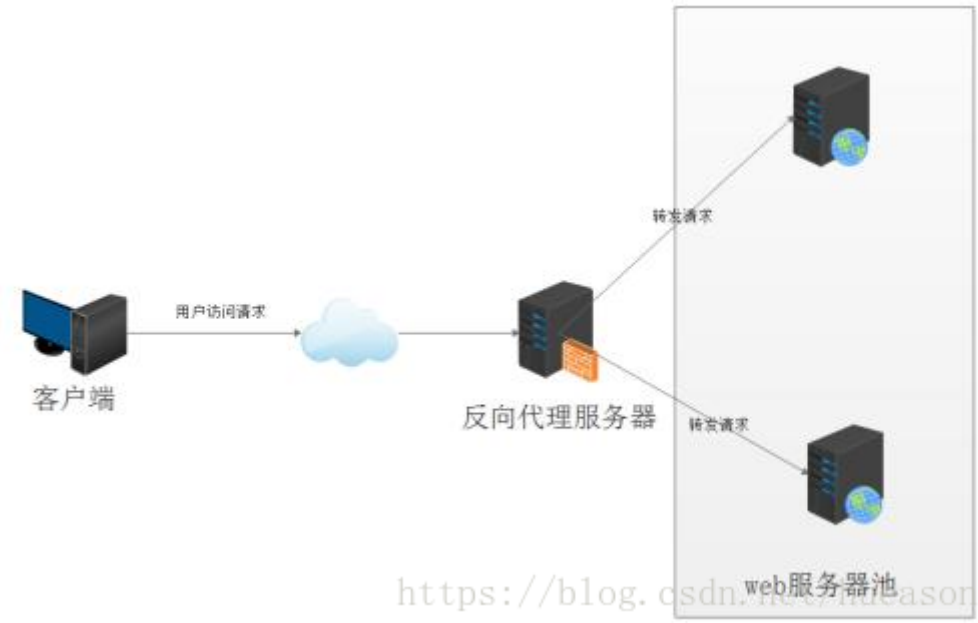
负载均衡又可分为服务端负载均衡和客户端负载均衡

服务器端负载均衡：例如Nginx，通过Nginx进行负载均衡，先发送请求，然后通过负载均衡算法，在多个服务器之间选择一个进行访问；即在服务器端再进行负载均衡算法分配。

客户端负载均衡：例如ribbon或者dubbo，客户端会有一个服务器地址列表，在发送请求前通过负载均衡算法选择一个服务器，然后进行访问，这是客户端负载均衡；即在客户端就进行负载均衡算法分配。

三、nginx灰度方案说明

Nginx是一款轻量级的Web服务器/反向代理服务器以及电子邮件代理服务器。



nginx.cof

```
http{
    # 待选服务器列表
    upstream myproject{
        # ip_hash指令，将同一用户引入同一服务器。
        ip_hash;
        server 125.219.42.4 fail_timeout=60s;
        server 172.31.2.183;
        }

    server{
                # 监听端口
                listen 80;
                # 根目录下
                location / {
                    # 选择哪个服务器列表
                    proxy_pass http://myproject;
                }

            }
}
```


负载均衡策略：轮询（默认）、weight、ip_hash、fair（响应时间）、url_hash

简单的根据cookie进行灰度

```
upstream tts_V6 {
    server 192.168.3.81:5280 max_fails=1 fail_timeout=60;
}
upstream tts_V7 {
    server 192.168.3.81:5380 max_fails=1 fail_timeout=60;
}
upstream default {  #通过upstream default + weight节点控制权重
    server 192.168.3.81:5280 max_fails=1 fail_timeout=60 weight=5;
    server 192.168.3.81:5380 max_fails=1 fail_timeout=60 weight=1;
}
server {
    listen 80;
    server_name test.taotaosou.com;
    access_log logs/test.taotaosou.com.log main buffer=32k;
    #match cookie
    set $group "default";
    if ($http_cookie ~* "tts_version_id=tts1"){ #动态控制路由
        set $group tts_V6;
    }
    if ($http_cookie ~* "tts_version_id=tts2"){
        set $group tts_V7;
    }
    location / {
        proxy_pass http://$group;
        proxy_set_header  Host       $host;
        proxy_set_header  X-Real-IP    $remote_addr;
        proxy_set_header  X-Forwarded-For $proxy_add_x_forwarded_for;
        index index.html index.htm;
    }
 }
```


Openresty

Nginx有很多的特性和好处，但是在Nginx上开发成了一个难题，Nginx模块需要用C开发，而且必须符合一系列复杂的规则，最重要的用C开发模块 必须要熟悉Nginx的源代码，使得开发者对其望而生畏。为了开发人员方便，所以接下来我们要介绍一种整合了Nginx和lua的框架，那就是OpenResty。

OpenResty® 是一个基于 Nginx 与 Lua 的高性能 Web 平台，其内部集成了大量精良的 Lua 库、第三方模块以及大多数的依赖项。用于方便地搭建能够处理超高并发、扩展性极高的动态 Web 应用、Web 服务和动态网关。

Openresty学习地址：https://moonbingbing.gitbooks.io/openresty-best-practices/content/base/intro.html

新浪微博开源项目

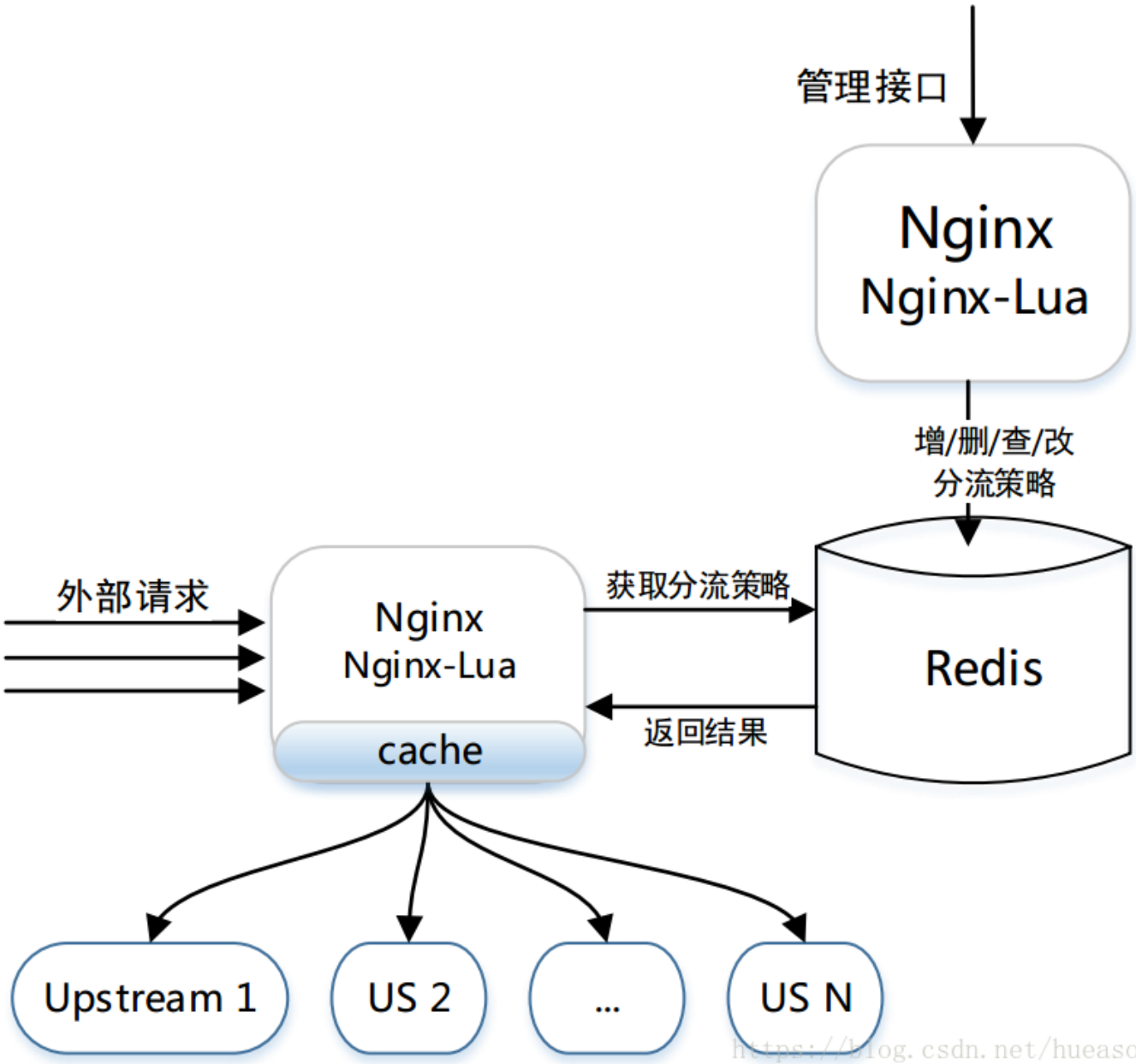
git地址：https://github.com/CNSRE/ABTestingGateway

ABTestingGateway是一个可以动态设置分流策略的灰度发布系统，工作在7层，基于nginx和ngx-lua开发，使用 redis 作为分流策略数据库，可以实现动态调度功能。

ABTestingGateway是在 nginx 转发的框架内，在转向 upstream 前，根据 用户请求特征 和 系统的分流策略 ，查找出目标upstream，进而实现分流。

nginx实现的灰度系统中，分流逻辑往往通过 rewrite 阶段的 if 和rewrite 指令等实现，优点是性能较高，缺点是功能受限、容易出错，以及转发规则固定，只能静态分流。针对这些缺点，ABTestingGateway，采用ngx-lua 实现系统功能，通过启用lua-shared-dict和lua-resty-lock作为系统缓存和缓存锁，系统获得了较为接近原生nginx转发的性能。

架构图：



特性：

支持多种分流方式，目前包括iprange、uidrange、uid尾数和指定uid分流

动态设置分流策略，即时生效，无需重启

可扩展性，提供了开发框架，开发者可以灵活添加新的分流方式，实现二次开发

高性能，压测数据接近原生nginx转发

灰度系统配置写在nginx配置文件中，方便管理员配置

适用于多种场景：灰度发布、AB测试和负载均衡等

new feature: ***支持多级分流***

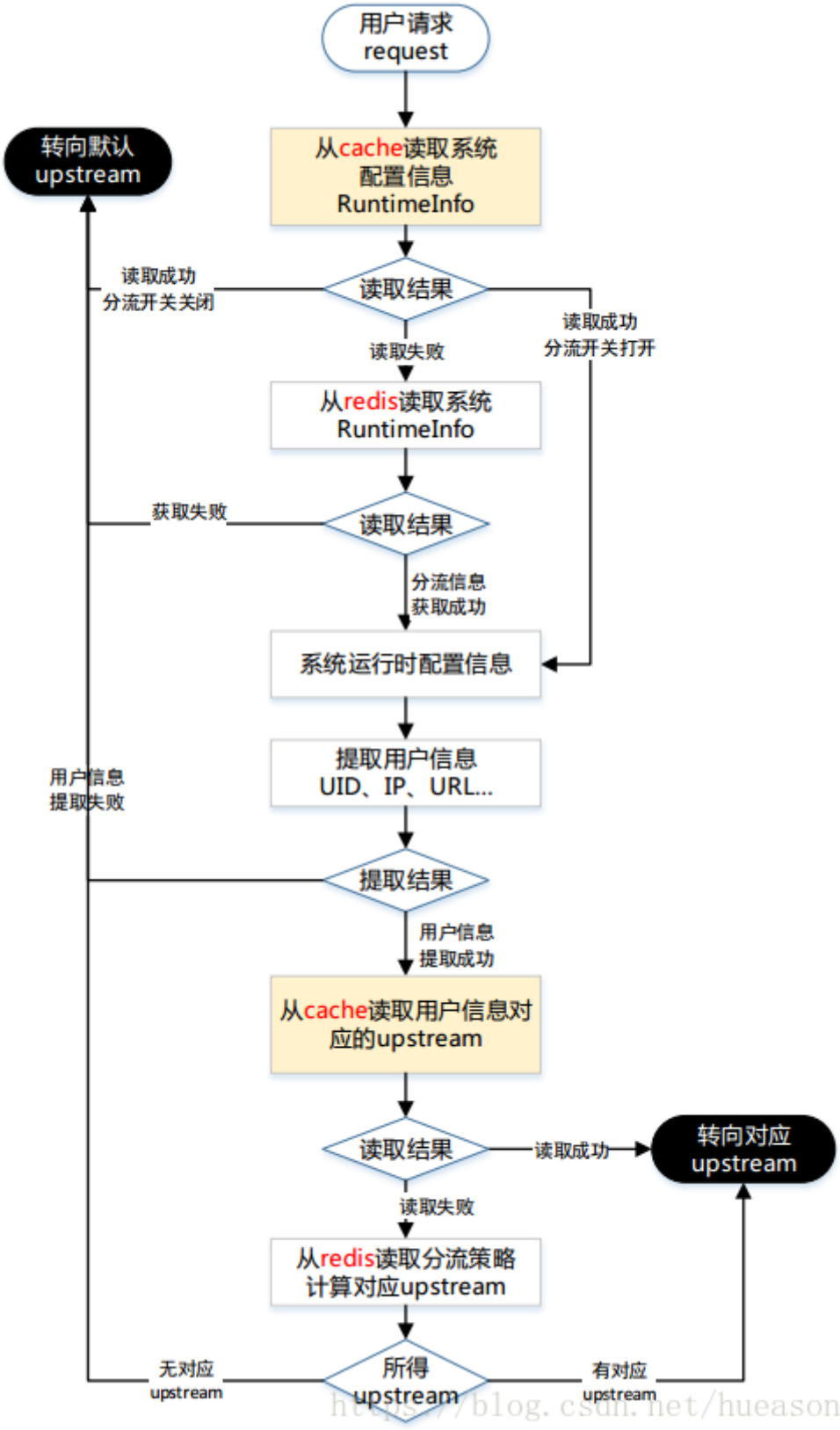
实现原理：在代理转发前，使用rewrite_by_lua_file模块重写到目标upstream

```
location / {

    error_log  logs/vhost_error.log debug;

    set $hostkey $server_name;
    set $sysConfig api_root_sysConfig;
    set $kv_upstream kv_api_root_upstream;
    set $backend 'stable';
    rewrite_by_lua_file '../diversion/diversion.lua';
    proxy_pass http://$backend;
}
```

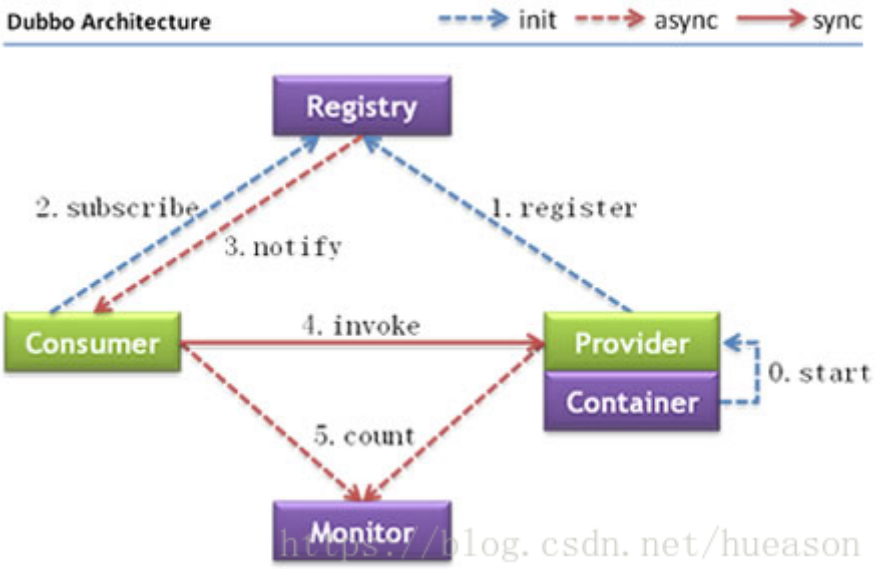

diversion.lua逻辑：



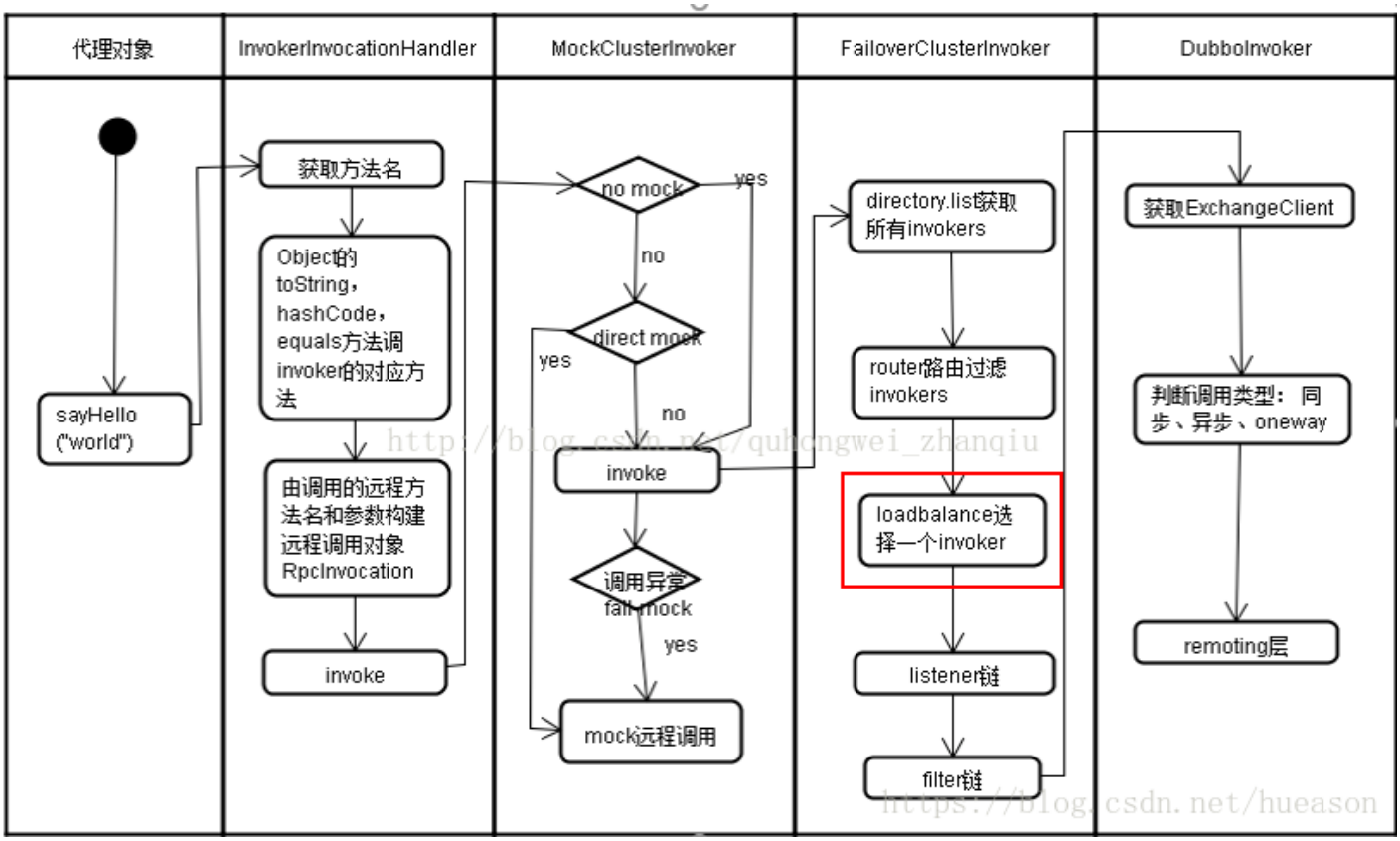
例：略

四、dubbo灰度方案说明

Dubbo架构



Dubbo服务调用过程



Loadbalance（负载均衡）说明

在集群负载均衡时，Dubbo 提供了多种均衡策略，缺省为 `random` 随机调用

负载均衡策略Random(随机)、RoundRobin（轮询）、LeastActive（最小活跃调用数）、ConsistentHash（一致性Hash）

负载均衡配置：

服务端服务级别

```
<dubbo:service interface="..." loadbalance="roundrobin" />
```

客户端服务级别

```
<dubbo:reference interface="..." loadbalance="roundrobin" />
```

服务端方法级别

```
<dubbo:service interface="...">
    <dubbo:method name="..." loadbalance="roundrobin"/>
</dubbo:service>
```

客户端方法级别

```
<dubbo:reference interface="...">
    <dubbo:method name="..." loadbalance="roundrobin"/>
</dubbo:reference>
```


自定义负载均衡实现：

扩展示例

Maven 项目结构：

```
src
 |-main
    |-java
        |-com
            |-xxx
                |-XxxLoadBalance.java (实现LoadBalance接口)
    |-resources
        |-META-INF
            |-dubbo
                |-com.alibaba.dubbo.rpc.cluster.LoadBalance (纯文本文件，内容为：xxx=com.xxx.
```

XxxLoadBalance.java：

```
package com.xxx;

import com.alibaba.dubbo.rpc.cluster.LoadBalance;
import com.alibaba.dubbo.rpc.Invoker;
import com.alibaba.dubbo.rpc.Invocation;
import com.alibaba.dubbo.rpc.RpcException;

public class XxxLoadBalance implements LoadBalance {
    public <T> Invoker<T> select(List<Invoker<T>> invokers, Invocation invocation) throws R
        // ...
    }
}
```

META-INF/dubbo/com.alibaba.dubbo.rpc.cluster.LoadBalance：

```
xxx=com.xxx.XxxLoadBalance
```


例：

实现LoadBalance接口，或者继承AbstractLoadBalance 重写策略；

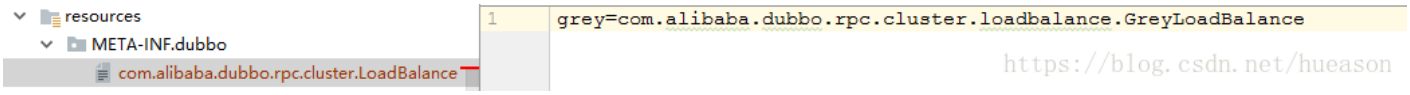
```
/**
 * @Author: hueason
 * @Date: 2018/7/6/006 11:32
 */
public class GreyLoadBalance extends AbstractLoadBalance {

    public static final String NAME = "grey";
    public static final int greyPort = 22112;
    private final Random random = new Random();

    @Override
    protected <T> Invoker<T> doSelect(List<Invoker<T>> invokers, URL url, Invocation invocation) {
        List<Invoker<T>> list = new ArrayList<>();
        for (Invoker invoker : invokers) {
            list.add(invoker);
        }
        Long userId = 0l;
        Class<?>[] parameterTypes = invocation.getParameterTypes();
        if (parameterTypes.length > 0 && parameterTypes[0].equals(Long.class)) {
            userId = (Long) invocation.getArguments()[0];
        }

        Iterator<Invoker<T>> iterator = list.iterator();
        while (iterator.hasNext()) {
            Invoker<T> invoker = iterator.next();
            int port = invoker.getUrl().getPort();
            String address = invoker.getUrl().getAddress();
            String backupAddress = invoker.getUrl().getBackupAddress();
            List<URL> backupUrls = invoker.getUrl().getBackupUrls();
```


根据dubbo SPI发现机制，还需要在resources下添加META-INF/dubbo/com.alibaba.dubbo.rpc.cluster.LoadBalance



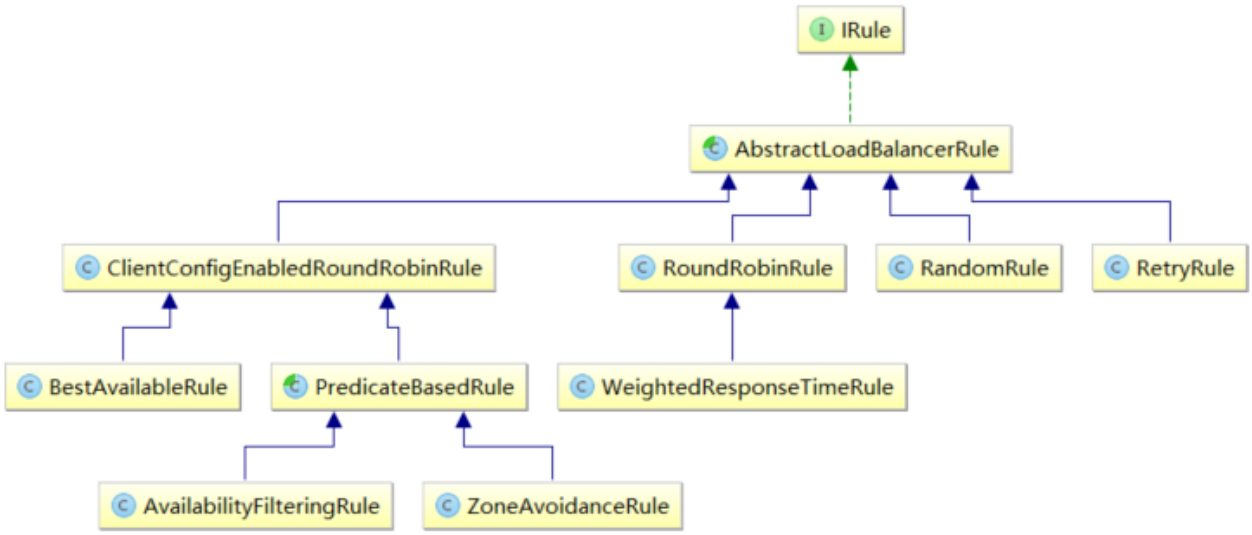
demo逻辑：目标服务的端口和灰度服务端口的一致，并且请求方法的第一个参数类型是Long(userId)并且是灰度用户，则判断为灰度服务，否则按照默认随机调用其余非灰度服务

```
@Override
protected <T> Invoker<T> doSelect(List<Invoker<T>> invokers, URL url, Invocation invocation) {
    List<Invoker<T>> list = new ArrayList<>();
    for (Invoker invoker : invokers) {
        list.add(invoker);
    }
    Long userId = 0l;
    Class<?>[] parameterTypes = invocation.getParameterTypes();
    if (parameterTypes.length > 0 && parameterTypes[0].equals(Long.class)) {
        userId = (Long) invocation.getArguments()[0];
    }

    Iterator<Invoker<T>> iterator = list.iterator();
    while (iterator.hasNext()) {
        Invoker<T> invoker = iterator.next();
        int port = invoker.getUrl().getPort();
        //找到灰度服务
        if (greyPort == port) {
            if (userId > 0 && this.checkUserIdInclude(userId)) {
                //确认是需要路由的请求
                //    String ip = invoker.getUrl().getIp();
                return invoker;
            } else {
                iterator.remove();
            }
        }
    }
    //找不到灰度服务或者 不需要路由   --默认按权重随机调用RandomLoadBalance
    return this.randomSelect(list, url, invocation);
```


四、springCloud灰度方案说明

Ribbon 提供了几个负载均衡的组件，其目的就是让请求转给合适的服务器处理

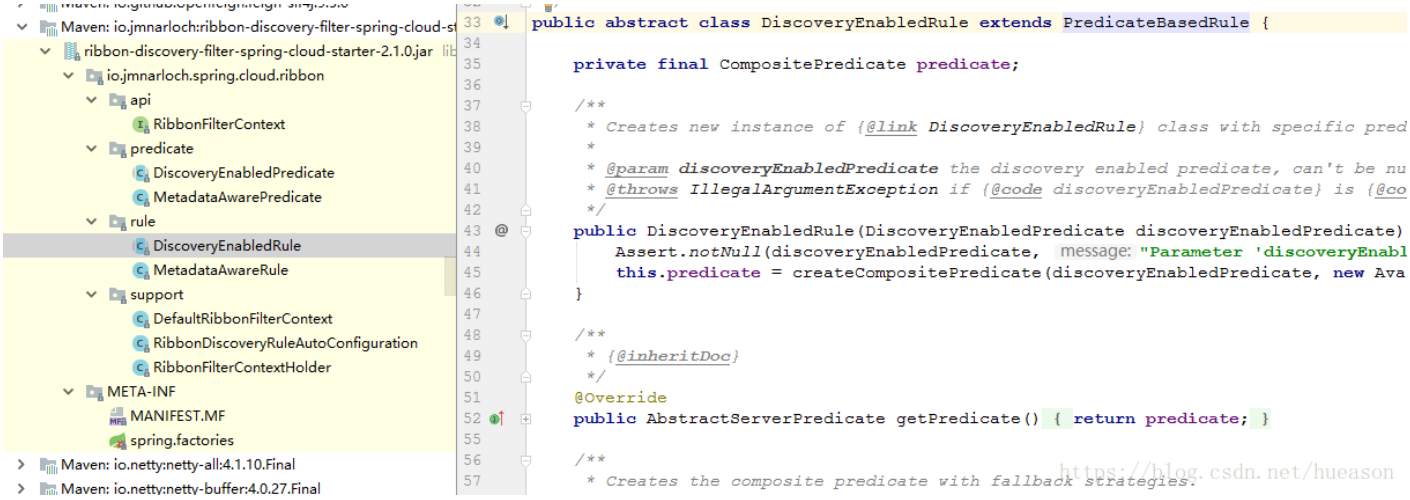


默认轮询

自定义策略需要继承AbstractLoadBalancerRule

开源方案：

自定义DiscoveryEnabledRule继承PredicateBaseRule



首先在请求开始处，实现自己的灰度逻辑，比如下面的demo根据请求url如果包含‘version’向holder中添加route为A的标识，否则添加route为B的标识。 (Holder本质是一个localThread)

```java
public class DynamicRoutesFilter extends ZuulFilter {


    @Override
    public int filterOrder() { return PRE_DECORATION_FILTER_ORDER; }

    @Override
    public String filterType() { return PRE_TYPE; }

    @Override
    public boolean shouldFilter() {...}

    @Override
    public Object run() {
        RequestContext ctx = RequestContext.getCurrentContext();
        HttpServletRequest request = ctx.getRequest();
        log.info("requestURI:" + request.getRequestURI());
        if (request.getRequestURI().contains("version")) {
            // put the serviceId in `RequestContext`
            RibbonFilterContextHolder.getCurrentContext()
                    .add("route", "A");
        } else {
            RibbonFilterContextHolder.getCurrentContext()
                    .add("route", "B");
        }

        return null;
    }
}
```


在目标服务添加matadateMap

```yaml
eureka:
  instance:
    instance-id: ${spring.cloud.client.ipAddress}:${server.port}
    prefer-ip-address: true
    metadata-map:
      route: A
```


PredicateBaseRule中使用google提供的pridicate，MetadataAwarePredicate中实现apply方法判断发现的服务是否是目标服务

```
lic class MetadataAwarePredicate extends DiscoveryEnabledPredicate {

 /**
  * {@inheritDoc}
  */
 @Override
 protected boolean apply(DiscoveryEnabledServer server) {

     final RibbonFilterContext context = RibbonFilterContextHolder.getCurrentContext();
     final Set<Map.Entry<String, String>> attributes = Collections.unmodifiableSet(context.
     final Map<String, String> metadata = server.getInstanceInfo().getMetadata();
     return metadata.entrySet().containsAll(attributes);
 }
```