目录

展开

1. 前言

1.1. 里约智能网关核心运行逻辑

以下是智能网关的核心运行逻辑图:

可以看出，智能网关主进程可以使用规则插件，包括HTTP插件和TCP插件。
其中规则插件中又可以定义各种规则匹配和请求处理、响应处理的filter。
综上，里约智能网关的二次开发，包括规则开发和过滤器开发。

1.2. 目的

本文档指导如何搭建一个二次开发环境，并阐述基本的规则开发和过滤器开发。

2. 环境搭建

2.1. 安装依赖

智能网关依赖node.js,可以使用最新版本，点击前往下载。

2.2. 开发工具

本实例的开发工具为Visual studio code,可以使用最新版本，点击前往下载。

2.3. 工程目录

二次开发环境的工程目录如下:

直接下载：smartgate_lab.zip
最小化的智能网关开发环境只有三个文件:
bin/smartgate.js 智能网关的主.js文件。(闭源，代码已混淆)
etc/settings.json 启动配置文件

```
{
    "AppName":"accessgate",
```

```
    "configRedis-1": {
      "sentinels": [{
        "host": "172.16.0.40",
        "port": 26379
      }],
      "password": "ee06167b10a177f60766d35baa81955d",
      "name": "default"
    },
    "MasterConfigUrl-n": "etc/config-yn.json",
    "MasterConfigUrl-2": "etc/config-2.json",
    "MasterConfigUrl": "etc/config.json",
    "configRedis-2": {
      "sentinels": [{
        "host": "9.134.68.236",
        "port": 26379
      }],
      "password": "ee06167b10a177f60766d35baa81955d",
      "name": "default"
    },
    "LogPath": "./log/",
    "UseHourLogPrefix": true,
    "bindPorts": "8888",
    "bindIP": "0.0.0.0",
    "cachePort": 8001,
    "workerCount": 0,
    "counterInterval": 10000
  }
```

注意:

AppName:只有在redis作为配置中心时有用,对应到smartproxy:config下面的key值，即如果这个值是accessgate，则使用smartproxy:config:accessgate的配置

configRedis：配置中心为redis时启用

MasterConfigUrl:配置中心为文件启用，使用config.json这个规则配置文件，网关的规则都配置在这个文件中。

注意: MasterConfigUrl和configRedis要完全匹配才生效，例如上例的configRedis-2是不起作用的（可用于多种配置只有一个生效时的场景）。

2.4. 启动开发环境

在终端下执行node bin/smartgate.js即可启动开发环境。

3. 智能网关规则开发

智能网关规则相关对象参考文档：SmartProxy_compressed.pdf

3.1. 里约规则插件运行流程

以下是里约规则插件运行的流程:

用户请求某url。

通过规则插件进行匹配。

调用request预处理filter,处理请求头、参数、报文后把请求转给目标服务。

目标服务响应并发回给网关。

响应内容发给网关后，网关使用response处理的filter进行响应头、响应报文处理后发回给用户。

其中规则匹配流程如下:

可以看出，请求会一直匹配规则，直接匹配到规则为止。

3.2. routeRule路由规则

智能网关的路由规则表现为一个json对象，例如:

```
{
  "routeRule_test":[
  {
   "appName":"testHello",
   "host":"www.test.com",
   "path": "^/hello",
   "customMatcher":"return headers['test']",
   "targetHost": "192.168.1.100:80",
   "targetPath": "/hello",
   "requestBodyFilter": "$:=rf.require('fs').readFileSync('etc/testlog-1-reqbody.js').toString()",
   "responseBodyFilter": "$:=rf.require('fs').readFileSync('etc/testlog-1-resbody.js').toString()",
   "responseHeadersFilter": "$:=rf.require('fs').readFileSync('etc/testlog-1-resheaders.js').toString()",
   "requestHeadersFilter": "$:=rf.require('fs').readFileSync('etc/testlog-1-reqheaders.js').toString()"
  }
  ]
}
```

路由规则的属性有如下几种:

标记属性

appName为标记属性，即不会对规则的行为造成影响，但数据可以被智能网关读取，例如本例中的appName会被读取并保存在access日志中，表示本次请求命中了这条规则，用于运营分析或者故障排查。

条件属性

其中host和path、customMatcher为条件属性，即满足条件时就会命中这条规则，一个规则中使用多个条件属性时逻辑上是“与”的关系，即这些条件都命中时才会满足。其中的customMatcher是比较特殊的条件属性，要命中时，其中的javascript代码值为true。本例是请求http://www.test.com/hello且http头中有test字段值且不为空时命中。

执行属性

targetHost和targetPath为执行属性，即控制命中规则后的执行。本例是把请求转发给http://192.168.1.100:80/hello这个地址。

requestBodyFilter、responseBodyFilter、responseHeadersFilter、requestHeadersFilter是一系列特殊的执行属性，命中后会执行其中的javascript代码。代码可以写在文本里面，也可以写在文件中，如果写在文件中，文件要放到etc目录下面，且在规则里面使用fs对象在启动时读取。过滤器的开发在后续章节会重点介绍。

更多属性请参考附件中的文档。

3.3. connectRule代理规则

routeRule可以看成把智能网关变成一个反向代理，而connectRule则可以看成把智能网关变成一个正向代理，即用智能网关作为代理服务器。以下是一个常见的connectRule代理规则:

```
{
  "appName": "Internet Email connect",
  "customMatcher": "return rule.targets[req.headers.host]&amp;amp;amp;amp; req.headers['proxy-authorization']=='Basic eXF0Onc5VmpYJDZnOUJ4N01vQA==",
  "targets": {
    "smtp.exmail.qq.com:465": 1
  },
  "connectSecondProxy": [
      "19.15.63.251:80","19.15.63.44:80","19.15.63.197:80","19.15.0.145:80","19.15.0.153:80","19.15.0.160:80"
  ]
```

```
10.  }
```

这条规则允许网关作为代理服务器，但只能访问smtp.exmail.qq.com:465这个地址向外发送邮件，并且要通过connectSecondProxy标识的二级代理服务器。

4. 智能网关过滤器开发

4.1. 智能网关内置对象

智能网关内置对象都可以在过滤器中通过rf保留字调用。例如:

```
1.  rf.require('net').isIP('10.0.0.1')
```

详细文档见附件

4.2. 智能网关过滤器类型

过滤器包含了头过滤器、内容过滤器及数据块过滤器三种。

头过滤器

- 请求头过滤器
- 响应头过滤器

内容过滤器

- 请求内容过滤器
- 请求参数处理器
- 响应内容过滤器
- 响应参数处理器

数据块过滤器分为

- 请求数据块过滤器
- 响应数据块过滤器

以上过滤器详细说明见附件

4.3. 智能网关过滤器的作用域

从作用域上看，过滤器分为全局过滤器和规则过滤器。

全局过滤器对所有经过网关实例的请求都有效。

规则过滤器只有在规则配置时才能生效。

tips:全局过滤器会优先规则过滤器运行。

4.4. 常用过滤器开发示例代码

4.4.1. 调用公安部三部api网关

使用过滤器拦截对api网关的请求，并修改请求报文，用于代理对公安部三部的api网关的调用。
需安装到api网关相关规则的requestBodyFilter中。

```
1.  let data;
2.     let enc = encodeURIComponent;
3.     let path = req.parsedUrl.path;
4.     if (path.split('/').length &gt; 5) {
5.        path = path.split('/').slice(0, 5).join('/');
6.     }
7.     if (buf) {
8.        data = enc(body);
9.     } else {
```

```
        data = '';
    }
    let newParam = {
        senderID: 'C10-00000054',
        serviceID: '',
        serviceAlias: '/netapi3/gdga/zwtogaapi',
        serviceDescript: {
            header: req.headers,
            data: data,
            serviceName: req.url.split('/ebus')[1],
            serviceMethod: req.method.toLowerCase()
        }
    }
    req.sp_options.method = 'POST';
    req.sp_options.path = '/dataex/queryrs/query';
    rf.writeLog('aio_requestBodyFilterapi', [JSON.stringify(newParam)]);
    return JSON.stringify(newParam);
```

4.4.2. 根据错误码跳转

根据错误码跳转到不同的错误页面，需安装到最外层接入/准入网关全局requestBodyFilter中。

```
if (res.rawHeaders &amp;&amp; res.rawHeaders.length) {
    for (let i = 0; i &lt; res.rawHeaders.length; i++) {
        if (res.rawHeaders[i] &amp;&amp; res.rawHeaders[i].toLowerCase().trim() == 'connection') {
            headers['connection'] = res.rawHeaders[i + 1];
            break;
        }
    }
}
let wtHost = JSON.parse(rf.getConfig('wtHost')) || {}, host = req.headers['host'];
if (wtHost[host]) {
    let redirectUrl = 'https://static.gdzwfw.gov.cn/common/';
    if (res.statusCode &gt;= 400 &amp;&amp; res.statusCode &lt; 500) {
        if (res.statusCode == 403) {
            redirectUrl = redirectUrl + 'tip-403.html';
        } else if (res.statusCode == 404) {
            redirectUrl = redirectUrl + 'tip-404.html';
        } else {
            redirectUrl = redirectUrl + 'tip-4xx.html';
        }
        return Promise.reject({statusCode: 302, headers: {Location: redirectUrl}, body: ''});
    } else if (res.statusCode &gt;= 500) {
        if (res.statusCode == 502) {
            redirectUrl = redirectUrl + 'tip-502.html';
        } else if (res.statusCode == 503) {
            redirectUrl = redirectUrl + 'tip-503.html';
        } else if (res.statusCode == 504) {
            redirectUrl = redirectUrl + 'tip-504.html';
        } else {
            redirectUrl = redirectUrl + 'tip-5xx.html';
        }
        return Promise.reject({statusCode: 302, headers: {Location: redirectUrl}, body: ''});
```

```
    }
}
```

注意 使用下列配置做上述脚本的输入，即为启用跳转的白名单

```
"wtHost":{
    "www.gdzwfw.gov.cn":"1",
    "static.gdzwfw.gov.cn":"2",
    "qykb.gdzwfw.gov.cn":"3",
    "tyrz.gdbs.gov.cn":"4"
}
```

4.4.3. 使用API网关调用需做身份认证第三方系统服务

安装到requestBodyFilter中。

```
let redisClient = rf.redis.GetMasterRedisClient();
let clientId = 'd287365337fb4ee5a8e15d3bf356e269';//数据总线应用
let clientSecret = 'a7f593ec918a4aa7b7c75bca8715c181';//数据总线应用
//let clientId = '';
//let clientSecret = '';
let url = 'http://127.0.0.1:8080';
let reqHeaders = {
  'x-tif-paasid': "huawei",
  'content-type': 'application/x-www-form-urlencoded'
};
let reqBody = {};
let token = {};
reqBody.grant_type = 'client_credentials';
reqBody.client_id = req.headers['appkey'] || clientId;
reqBody.client_secret = req.headers['appsecret'] || clientSecret;
reqBody.scope = 'default';
const promise = new Promise(function(resolve,reject){
  let body = rf.require('querystring').stringify(reqBody);
  if (redisClient.ready === true) {
    redisClient.get('appid:'+ reqBody.client_id,function(err,value){
      if(err){
        err.code = 1333;
        err.statusCode = 502;
        reject(err);
      }else if(!value){
        rf.http.post(url + '/ebus/huawei/v1/apigw/oauth2/token',body,reqHeaders)
        .then(function(ret){
          let obj = ret;
          if(obj.access_token){
            token.value = obj.token_type + ' ' + obj.access_token;
            token.exp = obj.expires_in;
            resolve(token);
          }else{
            err = new Error(obj.error_msg);
            err.code = 1335;
            err.statusCode = 200;
            reject(err);
          }
```

```
            }).catch(function(err){
                err.code = 1334;
                err.statusCode = 500;
                reject(err);
            });
        }else {
            redisClient.ttl('appid:' + reqBody.client_id,function(err,time){
                if(err){
                    err.code = 1333;
                    err.statusCode = 502;
                    reject(err);
                }else{
                    if(time &gt;= 0&amp;amp;amp;time&lt;=300){
                        rf.http.post(url + '/ebus/huawei/v1/apigw/oauth2/token',body,reqHeaders)
                        .then(function(ret){
                            let obj = ret;
                            if(obj.access_token){
                                token.value = obj.token_type + ' ' + obj.access_token;
                                token.exp = obj.expires_in;
                                resolve(token);
                            }else{
                                err = new Error(obj.error_msg);
                                err.code = 1335;
                                err.statusCode = 200;
                                reject(err);
                            }
                        }).catch(function(err){
                            err.code = 1334;
                            err.statusCode = 500;
                            reject(err);
                        });
                    }else{
                        token.value = value;
                        resolve(token);
                    };
                };
            });
        };
    });
  } else {
    let err = new Error('redis is not ready now');
    err.code = 1337;
    err.statusCode = 500;
    reject(err);
  }
});
return promise.then(function(token){
  if(token.exp){
    let exp = token.exp - 30;
    redisClient.set('appid:' + reqBody.client_id,token.value,'EX',exp,function(err){
      if(err){
        err.code = 1332;
```

```
                err.statusCode = 502;
                reject(err);
            }
        });
    };
    headers['authorization'] = token.value;
}).catch(function(err){
    return Promise.reject({statusCode:err.statusCode,headers:{
        'Content-Type': 'application/json'},
      body:`{"errcode":${err.code},"errmsg":"${err.message}"}`});
});
```

4.4.4. 复杂案例：集成数据权限

需求一：数据资源目录服务是公安部标准服务，主要包括查询检索类服务、比对订阅类服务、数据推送服务、数据操作服务等标准服务，这些服务为通用性服务，每个通用的服务必须要根据资源目录中的数据集ID来提供服务。从后端系统目前提供的能力来看，客户端在访问服务端的通用性服务时，会将数据资源目录封装到HTTP报文的包体中，服务端解析以后会根据自身的权限体系判断是否返回请求的数据资源。未来，腾讯的API网关将统一在数据域内对外提供服务，即，将API网关部署到客户端和后端系统数据资源服务之间，因此需要API网关提供数据权限的申请、审批、校验等功能。计划单独开发API网关的“外挂”应用，实现通用服务和数据资源目录的配置管理、申请审批、服务接口等功能，当客户端通过API网关访问后端数据资源服务时，由API网关从“外挂”应用中根据用户ID动态获取应用ID以及数据资源目录的对应信息，与客户端请求报文中解析出来的数据资源目录进行校验。如果相互匹配，则放通，如果不匹配，则返回错误信息。
需求二：在通过腾讯网关访问后端系统的服务的时候 在腾讯网关把请求转发到后端系时要把应用标识（passId），ip(client_ip),请求时间(start_time),请求参数(request_content),服务名称（name），请求方式（method），请求单位名称（unitname），组成一个json字符串再base64加密,参数信息增加heard。
通过智能网关代理获取数据权限

5. 规则和过滤器的安装

规则和过滤器可以使用多种方式进行安装。
routeRule和connectRule最终会安装在配置管理中心的路由表中，key的名称为routeRule_0_test或者connectRule_1_test，规则以字母的顺序进行排序（升序），排在前面的规则会优先执行。

5.1. 使用里约控制台

当前版本的里约控制台在创建应用和服务时会自动生成规则到redis配置管理中心中，并且在注册站点和服务时可以在UI设置过滤器。

tips:里约控制台不能安装自定义规则和全局过滤器。

5.2. 使用sgadmin

sgadmin应用有一个强大的规则编辑器，可以用来管理routeRule和connectRule。
增加路由表:

增加路由规则:

规则过滤器可以在路由规则编辑器中直接设置:

全局过滤器也可以在sgadmin中安装:

如果不想修改系统内置的全局过滤器而想加入自己的过滤器，也可以在在全局设置-自定义安装过滤器，命名规则为：过滤器名称_自定义后缀，如requestHeadersFilter_copyNonce。

5.3. 使用redis管理工具

当使用redis作为配置管理中心时，可以直接修改redis中相关的记录进行规则和过滤器安装。里约默认安装的配置会在下图如图所示的key中。

tips:redis管理工具可以作为修改规则失败导致智能网关无法正常访问的补救工具。

5.4. 直接修改config.json

当使用config.json文件作为配置管理中心时，可以直接修改这个文件安装规则和过滤器。（接入网关一般会使用文件来配置）。以下是一个比较典型的config.json文件: