

Dubbo源码解析（四十）集群——router

[dubbo](#)  阅读约 30 分钟

集群——router

目标：介绍dubbo中集群的路由，介绍dubbo-cluster下router包的源码。

前言

路由规则决定一次dubbo服务调用的目标服务器，分为条件路由规则和脚本路由规则，并且支持可扩展。

源码分析

(一) ConditionRouterFactory

```
public class ConditionRouterFactory implements RouterFactory {  
  
    public static final String NAME = "condition";  
  
    @Override  
    public Router getRouter(URL url) {  
        // 创建一个ConditionRouter  
        return new ConditionRouter(url);  
    }  
}
```

该类是基于条件表达式规则路由工厂类。

(二) ConditionRouter

该类是基于条件表达式的路由实现类。关于给予条件表达式的路由规则，可以查看官方文档：

官方文档地址：<http://dubbo.apache.org/zh-cn...>

1. 属性

```

private static final Logger logger = LoggerFactory.getLogger(ConditionRouter.class);
/**
 * 分组正则匹配
 */
private static Pattern ROUTE_PATTERN = Pattern.compile("([&!=,]* )\\s*([^&!=,\\s]+)");
/**
 * 路由规则 URL
 */
private final URL url;
/**
 * 路由规则的优先级，用于排序，优先级越大越靠前执行，可不填，缺省为 0
 */
private final int priority;
/**
 * 当路由结果为空时，是否强制执行，如果不强制执行，路由结果为空的路由规则将自动失效，可不填，缺省为 false。
 */
private final boolean force;
/**
 * 消费者匹配条件集合，通过解析【条件表达式 rule 的 `=>` 之前半部分】
 */
private final Map<String, MatchPair> whenCondition;
/**
 * 提供者地址列表的过滤条件，通过解析【条件表达式 rule 的 `=>` 之后半部分】
 */
private final Map<String, MatchPair> thenCondition;

```

2.构造方法

```

public ConditionRouter(URL url) {
    this.url = url;
    // 获得优先级配置
    this.priority = url.getParameter(Constants.PRIORITY_KEY, 0);
    // 获得是否强制执行配置
    this.force = url.getParameter(Constants.FORCE_KEY, false);
    try {
        // 获得规则
        String rule = url.getParameterAndDecoded(Constants.RULE_KEY);
        if (rule == null || rule.trim().length() == 0) {
            throw new IllegalArgumentException("Illegal route rule!");
        }
        rule = rule.replace("consumer.", "").replace("provider.", "");
        int i = rule.indexOf("=>");
        // 分割消费者和提供者规则
        String whenRule = i < 0 ? null : rule.substring(0, i).trim();
        String thenRule = i < 0 ? rule.trim() : rule.substring(i + 2).trim();
        Map<String, MatchPair> when = StringUtils.isBlank(whenRule) || "true".equals(whenRule) ? new
        HashMap<String, MatchPair>() : parseRule(whenRule);
        Map<String, MatchPair> then = StringUtils.isBlank(thenRule) || "false".equals(thenRule) ? null :
        parseRule(thenRule);
        // NOTE: It should be determined on the business level whether the `When condition` can be empty or not.
        this.whenCondition = when;
        this.thenCondition = then;
    } catch (ParseException e) {
        throw new IllegalStateException(e.getMessage(), e);
    }
}

```

3.MatchPair

```

private static final class MatchPair {
    /**
     * 匹配的值的集合
     */
    final Set<String> matches = new HashSet<String>();
    /**
     * 不匹配的值的集合
     */
    final Set<String> mismatches = new HashSet<String>();

    /**
     * 判断value是否匹配matches或者mismatches
     * @param value
     * @param param
     * @return
     */
    private boolean isMatch(String value, URL param) {
        // 只匹配 matches
        if (!matches.isEmpty() && mismatches.isEmpty()) {
            for (String match : matches) {
                if (UrlUtils.isMatchGlobPattern(match, value, param)) {
                    // 匹配上了返回true
                    return true;
                }
            }
        }
        // 没匹配上则为false
    }
}

```

该类是内部类，封装了匹配的值，每个属性条件。并且提供了判断是否匹配的方法。

4.parseRule

```

private static Map<String, MatchPair> parseRule(String rule)
    throws ParseException {
    Map<String, MatchPair> condition = new HashMap<String, MatchPair>();
    // 如果规则为空，则直接返回空
    if (StringUtils.isBlank(rule)) {
        return condition;
    }
    // Key-Value pair, stores both match and mismatch conditions
    MatchPair pair = null;
    // Multiple values
    Set<String> values = null;
    // 正则表达式匹配
    final Matcher matcher = ROUTE_PATTERN.matcher(rule);
    // 一个一个匹配
    while (matcher.find()) { // Try to match one by one
        String separator = matcher.group(1);
        String content = matcher.group(2);
        // Start part of the condition expression.
        // 开始条件表达式
        if (separator == null || separator.length() == 0) {
            pair = new MatchPair();
            // 保存条件
            condition.put(content, pair);
        }
        // The KV part of the condition expression
        else if ("&".equals(separator)) {

```

该方法是根据规则解析路由配置内容。具体的可以参照官网的配置规则来解读这里每一个分割取值作为条件的过程。

5.route

```

@Override
public <T> List<Invoker<T>> route(List<Invoker<T>> invokers, URL url, Invocation invocation)
    throws RpcException {
    // 为空，直接返回空 Invoker 集合
    if (invokers == null || invokers.isEmpty()) {
        return invokers;
    }
    try {
        // 如果不匹配 `whenCondition`，直接返回 `invokers` 集合，因为不需要走 `whenThen` 的匹配
        if (!matchWhen(url, invocation)) {
            return invokers;
        }
        List<Invoker<T>> result = new ArrayList<Invoker<T>>();
        // 如果thenCondition为空，则直接返回空
        if (thenCondition == null) {
            logger.warn("The current consumer in the service blacklist. consumer: " + NetUtils.getLocalHost() +
", service: " + url.getServiceKey());
            return result;
        }
        // 遍历invokers
        for (Invoker<T> invoker : invokers) {
            // 如果thenCondition匹配，则加入result
            if (matchThen(invoker.getUrl(), url)) {
                result.add(invoker);
            }
        }
    }
}

```

该方法是进行路由规则的匹配，分别对消费者和提供者进行匹配。

6.matchCondition

```

private boolean matchCondition(Map<String, MatchPair> condition, URL url, URL param, Invocation invocation) {
    Map<String, String> sample = url.toMap();
    // 是否匹配
    boolean result = false;
    // 遍历条件
    for (Map.Entry<String, MatchPair> matchPair : condition.entrySet()) {
        String key = matchPair.getKey();
        String sampleValue;
        //get real invoked method name from invocation
        // 获得方法名
        if (invocation != null && (Constants.METHOD_KEY.equals(key) || Constants.METHODS_KEY.equals(key))) {
            sampleValue = invocation.getMethodName();
        } else {
            //
            sampleValue = sample.get(key);
            if (sampleValue == null) {
                sampleValue = sample.get(Constants.DEFAULT_KEY_PREFIX + key);
            }
        }
        if (sampleValue != null) {
            // 如果不匹配条件值，返回false
            if (!matchPair.getValue().isMatch(sampleValue, param)) {
                return false;
            } else {
                // 匹配则返回true
                result = true;
            }
        }
    }
}

```

该方法是匹配条件的主要逻辑。

(三) ScriptRouterFactory

该类是基于脚本的路由规则工厂类。

```
public class ScriptRouterFactory implements RouterFactory {  
  
    public static final String NAME = "script";  
  
    @Override  
    public Router getRouter(URL url) {  
        // 创建ScriptRouter  
        return new ScriptRouter(url);  
    }  
  
}
```

(四) ScriptRouter

该类是基于脚本的路由实现类

1. 属性

```
private static final Logger logger = LoggerFactory.getLogger(ScriptRouter.class);  
  
/**  
 * 脚本类型 与 ScriptEngine 的映射缓存  
 */  
private static final Map<String, ScriptEngine> engines = new ConcurrentHashMap<String, ScriptEngine>();  
  
/**  
 * 脚本  
 */  
private final ScriptEngine engine;  
  
/**  
 * 路由规则的优先级，用于排序，优先级越大越靠前执行，可不填，缺省为 0 。  
 */  
private final int priority;  
  
/**  
 * 路由规则  
 */  
private final String rule;  
  
/**  
 * 路由规则 URL  
 */  
private final URL url;
```

2. route

```

@Override
@SuppressWarnings("unchecked")
public <T> List<Invoker<T>> route(List<Invoker<T>> invokers, URL url, Invocation invocation) throws RpcException
{
    try {
        List<Invoker<T>> invokersCopy = new ArrayList<Invoker<T>>(invokers);
        Compilable compilable = (Compilable) engine;
        // 创建脚本
        Bindings bindings = engine.createBindings();
        // 设置invokers, invocation, context
        bindings.put("invokers", invokersCopy);
        bindings.put("invocation", invocation);
        bindings.put("context", RpcContext.getContext());
        // 编译脚本
        CompiledScript function = compilable.compile(rule);
        // 执行脚本
        Object obj = function.eval(bindings);
        // 根据结果类型, 转换成 (List<Invoker<T>>) 类型返回
        if (obj instanceof Invoker[]) {
            invokersCopy = Arrays.asList((Invoker<T>[])) obj);
        } else if (obj instanceof Object[]) {
            invokersCopy = new ArrayList<Invoker<T>>();
            for (Object inv : (Object[]) obj) {
                invokersCopy.add((Invoker<T>) inv);
            }
        } else {
    
```

该方法是根据路由规则选择invoker的实现逻辑。

(五) FileRouterFactory

该类是装饰者，对RouterFactory进行了功能增强，增加了从文件中读取规则。

```

public class FileRouterFactory implements RouterFactory {

    public static final String NAME = "file";

    /**
     * 路由工厂
     */
    private RouterFactory routerFactory;

    public void setRouterFactory(RouterFactory routerFactory) {
        this.routerFactory = routerFactory;
    }

    @Override
    public Router getRouter(URL url) {
        try {
            // Transform File URL into Script Route URL, and Load
            // file:///d:/path/to/route.js?router=script ==> script:///d:/path/to/route.js?type=js&rule=<file-
content>
            // 获得 router 配置项, 默认为 script
            String protocol = url.getParameter(Constants.ROUTER_KEY, ScriptRouterFactory.NAME); // Replace
original protocol (maybe 'file') with 'script'
            String type = null; // Use file suffix to config script type, e.g., js, groovy ...
            // 获得path
            String path = url.getPath();
            // 获得类型
        
```

后记

该部分相关的源码解析地址：<https://github.com/CrazyHZM/i...>

该文章讲解了集群中关于路由规则实现的部分。接下来我将开始对集群模块关于Mock部分进行讲解。