

title: 精尽 Dubbo 源码分析 —— API 配置（三）之服务消费者 date: 2018-01-13 tags:

categories: Dubbo permalink: Dubbo/configuration-api-3

摘要: 原创出处 <http://www.iocoder.cn/Dubbo/configuration-api-3/> 「芋道源码」欢迎转载，保留摘要，谢谢！

- 1. 概述
- 2. AbstractReferenceConfig
- 3. ConsumerConfig
- 4. ReferenceConfig
- 666. 彩蛋



扫一扫二维码关注公众号

关注后，可以看到

「RocketMQ」

「MyCAT」

所有源码解析文章

— 近期更新「Sharding-JDBC」中 —

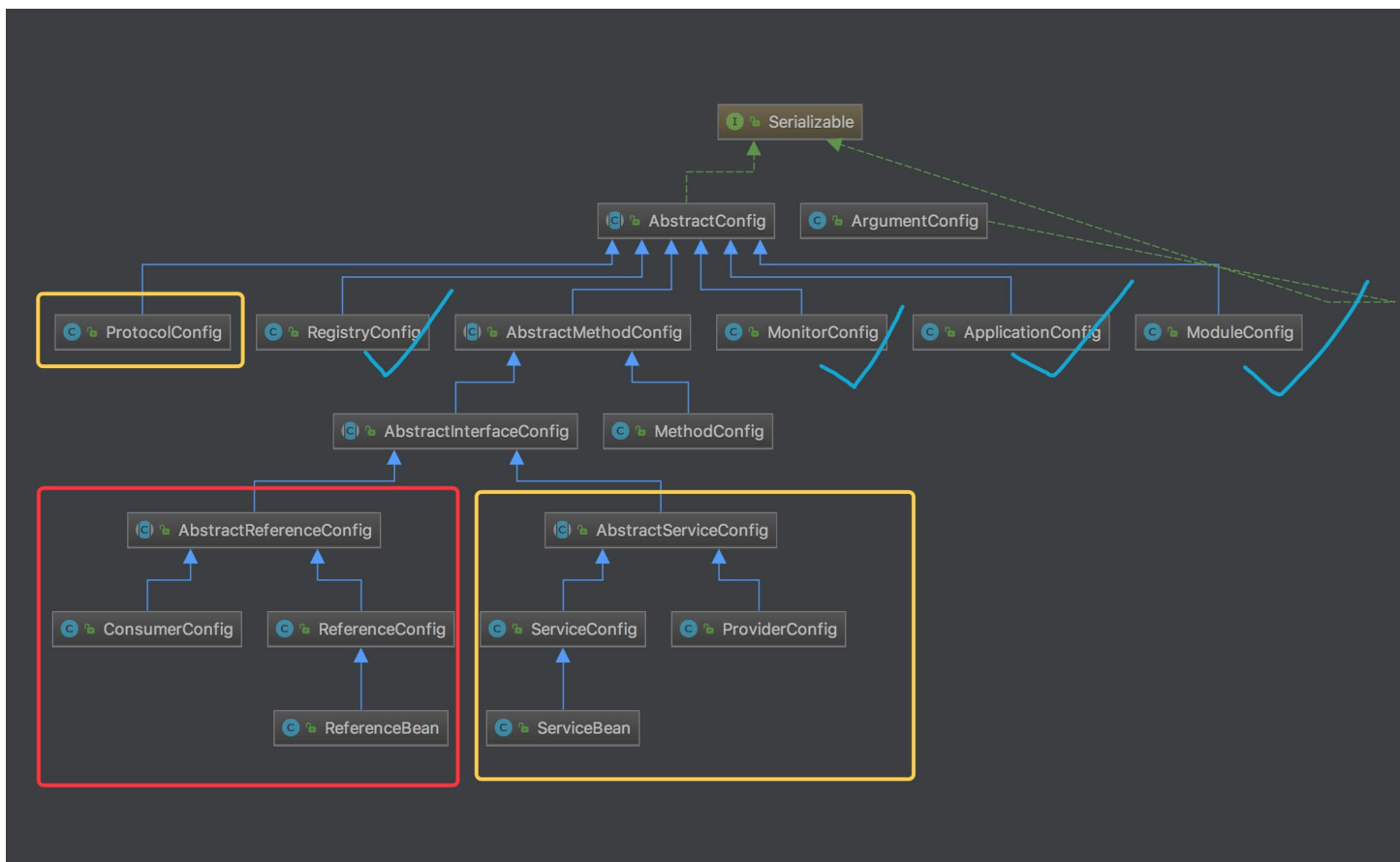
你有233个小伙伴已经关注

😊😊😊 关注微信公众号：【芋道源码】有福利：

1. RocketMQ / MyCAT / Sharding-JDBC 所有源码分析文章列表
2. RocketMQ / MyCAT / Sharding-JDBC 中文注释源码 GitHub 地址
3. 您对于源码的疑问每条留言都将得到认真回复。甚至不知道如何读源码也可以请教噢。
4. 新的源码解析文章实时收到通知。每周更新一篇左右。
5. 认真的源码交流微信群。

1. 概述

本文接 《API 配置（二）之服务提供者》，分享服务消费者相关的配置。



- 红框部分，consumer-side

还是老样子，我们先来看一段 《Dubbo 用户指南 —— API 配置》，服务消费者的初始化代码：

```
// 当前应用配置
ApplicationConfig application = new ApplicationConfig();
application.setName("yyy");

// 连接注册中心配置
RegistryConfig registry = new RegistryConfig();
registry.setAddress("10.20.130.230:9090");
registry.setUsername("aaa");
registry.setPassword("bbb");

// 注意：ReferenceConfig为重对象，内部封装了与注册中心的连接，以及与服务提供方的连接

// 引用远程服务
ReferenceConfig<XxxService> reference = new ReferenceConfig<XxxService>(); // 此实例
// 很重，封装了与注册中心的连接以及与提供者的连接，请自行缓存，否则可能造成内存和连接泄漏
reference.setApplication(application);
reference.setRegistry(registry); // 多个注册中心可以用setRegistries()
reference.setInterface(XxxService.class);
reference.setVersion("1.0.0");

// 和本地bean一样使用xxxService
```

```
XxxService xxxService = reference.get(); // 注意：此代理对象内部封装了所有通讯细节，对象较重，请缓存复用
```

2. AbstractReferenceConfig

`com.alibaba.dubbo.config.AbstractReferenceConfig`，实现 `AbstractInterfaceConfig`，抽象引用配置类。

- 具体属性的解释，需要寻找在《Dubbo 用户指南 —— dubbo:reference》或《Dubbo 用户指南 —— dubbo:consumer》文档。

3. ConsumerConfig

`com.alibaba.dubbo.config.ConsumerConfig`，实现 `AbstractReferenceConfig`，服务消费者缺省值配置。

- 具体属性的解释，参见《Dubbo 用户指南 —— dubbo:consumer》文档。

4. ReferenceConfig

`com.alibaba.dubbo.config.ReferenceConfig`，服务消费者引用服务配置类。

- 具体属性的解释，参见《Dubbo 用户指南 —— dubbo:consumer》文档。

下面，我们进入正戏。

在文初的 `ReferenceConfig` 的初始化示例代码中，最后调用的是 `ServiceConfig#get()` 方法。从方法的命名，我们可以看出，获取引用服务。该方法主要做了如下几件事情：

1. 进一步初始化 `ReferenceConfig` 对象。
2. 校验 `ReferenceConfig` 对象的配置项。
3. 使用 `ReferenceConfig` 对象，生成 Dubbo URL 对象数组。
4. 使用 Dubbo URL 对象，应用服务。



本文重点在服务提供者相关的配置，因此只解析 **1+2+3** 部分(不包括 4)。代码如下：

```
1: public synchronized T get() {
```

```

2:    // 已销毁, 不可获得
3:    if (destroyed) {
4:        throw new IllegalStateException("Already destroyed!");
5:    }
6:    // 初始化
7:    if (ref == null) {
8:        init();
9:    }
10:   return ref;
11: }

```

- 第 2 至 5 行: 若已经销毁(`destroyed = true`), 抛出异常。
- 第 7 至 9 行: 若未初始化, 调用 `#init()` 方法, 进行初始化。
- 第 10 行: 返回引用服务。

`#init()` 方法, 代码如下:

友情提示, 该方法并未拆分更多的小方法, 所以超级长, 近 200+ 行。

```

1: private void init() {
2:     // 已经初始化, 直接返回
3:     if (initialized) {
4:         return;
5:     }
6:     initialized = true;
7:     // 校验接口名非空
8:     if (interfaceName == null || interfaceName.length() == 0) {
9:         throw new IllegalStateException("<dubbo:reference interface=\"\" /> in
interface not allow null!");
10:    }
11:    // 拼接属性配置 (环境变量 + properties 属性) 到 ConsumerConfig 对象
12:    // get consumer's global configuration
13:    checkDefault();
14:    // 拼接属性配置 (环境变量 + properties 属性) 到 ReferenceConfig 对象
15:    appendProperties(this);
16:    // 若未设置 `generic` 属性, 使用 `ConsumerConfig.generic` 属性。
17:    if (getGeneric() == null && getConsumer() != null) {
18:        setGeneric(getConsumer().getGeneric());
19:    }
20:    // 泛化接口的实现
21:    if (ProtocolUtils.isGeneric(getGeneric())) {
22:        interfaceClass = GenericService.class;
23:    // 普通接口的实现
24:    } else {
25:        try {

```

```

26:         interfaceClass = Class.forName(interfaceName, true, Thread.currentThread().getContextClassLoader());
27:     } catch (ClassNotFoundException e) {
28:         throw new IllegalStateException(e.getMessage(), e);
29:     }
30:     // 校验接口和方法
31:     checkInterfaceAndMethods(interfaceClass, methods);
32: }
33: // 直连提供者, 参见文档《直连提供者》https://dubbo.gitbooks.io/dubbo-user-book/demos/explicit-target.html
34: // 【直连提供者】第一优先级, 通过 -D 参数指定, 例如 java -Dcom.alibaba.xxx.XxxService=dubbo://localhost:20890
35: String resolve = System.getProperty(interfaceName);
36: String resolveFile = null;
37: // 【直连提供者】第二优先级, 通过文件映射, 例如 com.alibaba.xxx.XxxService=dubbo://localhost:20890
38: if (resolve == null || resolve.length() == 0) {
39:     // 默认先加载, `${user.home}/dubbo-resolve.properties` 文件, 无需配置
40:     resolveFile = System.getProperty("dubbo.resolve.file");
41:     if (resolveFile == null || resolveFile.length() == 0) {
42:         File userResolveFile = new File(new File(System.getProperty("user.home")), "dubbo-resolve.properties");
43:         if (userResolveFile.exists()) {
44:             resolveFile = userResolveFile.getAbsolutePath();
45:         }
46:     }
47:     // 存在 resolveFile, 则进行文件读取加载。
48:     if (resolveFile != null && resolveFile.length() > 0) {
49:         Properties properties = new Properties();
50:         FileInputStream fis = null;
51:         try {
52:             fis = new FileInputStream(new File(resolveFile));
53:             properties.load(fis);
54:         } catch (IOException e) {
55:             throw new IllegalStateException("Unload " + resolveFile + ", cause: " + e.getMessage(), e);
56:         } finally {
57:             try {
58:                 if (null != fis) fis.close();
59:             } catch (IOException e) {
60:                 logger.warn(e.getMessage(), e);
61:             }
62:         }
63:         resolve = properties.getProperty(interfaceName);
64:     }
65: }
66: // 设置直连提供者的 url
67: if (resolve != null && resolve.length() > 0) {

```

```
68:         url = resolve;
69:         if (logger.isWarnEnabled()) {
70:             if (resolveFile != null && resolveFile.length() > 0) {
71:                 logger.warn("Using default dubbo resolve file " + resolveFile
+ " replace " + interfaceName + "" + resolve + " to p2p invoke remote service.");
72:             } else {
73:                 logger.warn("Using -D" + interfaceName + "=" + resolve + " to
p2p invoke remote service.");
74:             }
75:         }
76:     }
77:     // 从 ConsumerConfig 对象中, 读取 application、module、registries、monitor
配置对象。
78:     if (consumer != null) {
79:         if (application == null) {
80:             application = consumer.getApplication();
81:         }
82:         if (module == null) {
83:             module = consumer.getModule();
84:         }
85:         if (registries == null) {
86:             registries = consumer.getRegistries();
87:         }
88:         if (monitor == null) {
89:             monitor = consumer.getMonitor();
90:         }
91:     }
92:     // 从 ModuleConfig 对象中, 读取 registries、monitor 配置对象。
93:     if (module != null) {
94:         if (registries == null) {
95:             registries = module.getRegistries();
96:         }
97:         if (monitor == null) {
98:             monitor = module.getMonitor();
99:         }
100:    }
101:    // 从 ApplicationConfig 对象中, 读取 registries、monitor 配置对象。
102:    if (application != null) {
103:        if (registries == null) {
104:            registries = application.getRegistries();
105:        }
106:        if (monitor == null) {
107:            monitor = application.getMonitor();
108:        }
109:    }
110:    // 校验 ApplicationConfig 配置。
111:    checkApplication();
112:    // 校验 Stub 和 Mock 相关的配置
```

```
113:     checkStubAndMock(interfaceClass);
114:     // 将 `side`, `dubbo`, `timestamp`, `pid` 参数, 添加到 `map` 集合中。
115:     Map<String, String> map = new HashMap<String, String>();
116:     Map<Object, Object> attributes = new HashMap<Object, Object>();
117:     map.put(Constants.SIDE_KEY, Constants.CONSUMER_SIDE);
118:     map.put(Constants.DUBBO_VERSION_KEY, Version.getVersion());
119:     map.put(Constants.TIMESTAMP_KEY, String.valueOf(System.currentTimeMillis()
));
120:     if (ConfigUtils.getPid() > 0) {
121:         map.put(Constants.PID_KEY, String.valueOf(ConfigUtils.getPid()));
122:     }
123:     // methods、revision、interface
124:     if (!isGeneric()) {
125:         String revision = Version.getVersion(interfaceClass, version);
126:         if (revision != null && revision.length() > 0) {
127:             map.put("revision", revision);
128:         }
129:
130:         String[] methods = Wrapper.getWrapper(interfaceClass).getMethodNames()
; // 获得方法数组
131:         if (methods.length == 0) {
132:             logger.warn("NO method found in service interface " + interfaceCla
ss.getName());
133:             map.put("methods", Constants.ANY_VALUE);
134:         } else {
135:             map.put("methods", StringUtils.join(new HashSet<String>(Arrays.asL
ist(methods)), ","));
136:         }
137:     }
138:     map.put(Constants.INTERFACE_KEY, interfaceName);
139:     // 将各种配置对象, 添加到 `map` 集合中。
140:     appendParameters(map, application);
141:     appendParameters(map, module);
142:     appendParameters(map, consumer, Constants.DEFAULT_KEY);
143:     appendParameters(map, this);
144:     // 获得服务键, 作为前缀
145:     String prefix = StringUtils.getServiceKey(map);
146:     // 将 MethodConfig 对象数组, 添加到 `map` 集合中。
147:     if (methods != null && !methods.isEmpty()) {
148:         for (MethodConfig method : methods) {
149:             // 将 MethodConfig 对象, 添加到 `map` 集合中。
150:             appendParameters(map, method, method.getName());
151:             // 当 配置了 `MethodConfig.retry = false` 时, 强制禁用重试
152:             String retryKey = method.getName() + ".retry";
153:             if (map.containsKey(retryKey)) {
154:                 String retryValue = map.remove(retryKey);
155:                 if ("false".equals(retryValue)) {
156:                     map.put(method.getName() + ".retries", "0");
```



```

157:         }
158:     }
159:     // 将带有 @Parameter(attribute = true) 配置对象的属性, 添加到参数集合
    。参见《事件通知》https://dubbo.gitbooks.io/dubbo-user-book/demos/events-notify.html
160:     appendAttributes(attributes, method, prefix + "." + method.getName
    ());
161:     // 检查属性集中的事件通知方法是否正确。若正确, 进行转换。
162:     checkAndConvertImplicitConfig(method, map, attributes);
163: }
164: }
165:
166: // 以系统环境变量( DUBBO_IP_TO_REGISTRY ) 作为服务注册地址, 参见 https://github
    b.com/dubbo/dubbo-docker-sample 项目。
167: String hostToRegistry = ConfigUtils.getSystemProperty(Constants.DUBBO_IP_T
    O_REGISTRY);
168: if (hostToRegistry == null || hostToRegistry.length() == 0) {
169:     hostToRegistry = NetUtils.getLocalHost();
170: } else if (isInvalidLocalHost(hostToRegistry)) {
171:     throw new IllegalArgumentException("Specified invalid registry ip from
    property:" + Constants.DUBBO_IP_TO_REGISTRY + ", value:" + hostToRegistry);
172: }
173: map.put(Constants.REGISTER_IP_KEY, hostToRegistry);
174:
175: // 添加到 StaticContext 进行缓存
176: //attributes are stored by system context.
177: StaticContext.getSystemContext().putAll(attributes);
178:
179: // 省略【引用服务】
185: }

```

- 第 2 至 6 行: 若已经初始化(`initialized = true`) 时, 直接返回。否则, 标记已经初始化。
- 第 7 至 10 行: 校验接口名 `interfaceName` 非空。
- 第 13 行: 调用 `#checkDefault()` 方法, 读取属性配置(环境变量 + properties 属性)到 `ConsumerConfig` 对象。
 - 关于“属性配置”, 在《[精尽 Dubbo 源码解析 —— 属性配置](#)》详细解析。
 - 😊 直接点击方法查看, 较为简单, 已经添加详细注释。
- 第 15 行: 调用 `#appendProperties(config)` 方法, 读取属性配置(环境变量 + properties 属性)到 `ReferenceConfig` 对象 (自己)
- 第 16 至 19 行: 若未设置 `generic` 属性, 使用 `ConsumerConfig.generic` 属性。
- 第 20 至 22 行: 泛化接口的实现。
 - 《[Dubbo 用户指南 —— 泛化引用](#)》
- 第 23 至 32 行: 普通接口的实现。
 - 第 60 至 64 行: 根据 `interfaceName` , 获得对应的接口类, 并赋值给 `interfaceClass`

-
- 第 31 行：调用 `#checkInterfaceAndMethods(interfaceClass, methods)` 方法，检查接口和方法。
 - 😊 直接点击方法查看，较为简单，已经添加详细注释。
- 第 33 至 76 行：直连提供者。
 - 《Dubbo 用户指南 —— 直连提供者》
 - 😊 中间有一些逻辑处理，胖友看下代码的注释。结合文档。
- 第 77 至 109 行：从 ConsumerConfig、ModuleConfig、ApplicationConfig 配置对象，复制 `application module registries monitor` 给 ReferenceConfig (自己)。
- 第 111 行：调用 `#checkApplication()` 方法，校验 ApplicationConfig 配置。
 - 😊 直接点击方法查看，较为简单，已经添加详细注释。
- 第 113 行：调用 `#checkStubAndMock(interfaceClass)` 方法，校验 Stub 和 Mock 相关的配置。
 - 【TODO 8005】 芋艿，后续继续研究
- 第 115 行：创建参数集合 `map`，用于下面创建 Dubbo URL 的 `parameters` 属性。
- 第 116 至 122 行：将 `side dubbo timestamp timestamp pid` 添加到 `map` 中。
- 第 123 至 137 行：将 `interface methods revision` 到 `map` 中。
- 第 139 至 143 行：调用 `#appendParameters(map, config)` 方法，将各种配置对象添加到 `map` 中。
 - 😊 `#appendParameters(map, config)` 方法，在《API 配置（一）之应用》有详细解析。
- 第 146 至 164 行：调用 MethodConfig 对象数组，添加到 `map` 中。
 - 目的是将每个 MethodConfig 和其对应的 ArgumentConfig 对象数组，添加到 `map` 中。
 - 第 160 行：调用 `#appendAttributes(parameters, config, prefix)` 方法，将 `@Parameter(attribute = true)` 配置对象的属性，添加到参数集合。在《API 配置（一）之应用》有详细解析。
 - 《Dubbo 用户指南 —— 事件通知》
 - 第 162 行：调用 `#checkAndConvertImplicitConfig(method, map, attributes)` 方法，检查属性集合中的事件通知方法是否正确。若正确，进行转换。
 - 😊 直接点击方法查看，较为简单，已经添加详细注释。
- 第 166 至 173 行：以系统换将变量 (`DUBBO_IP_TO_REGISTRY`) 作为服务注册地址，参见 [dubbo-docker-sample](#) 项目。
- 第 177 行：添加到 `com.alibaba.dubbo.rpc.StaticContext` 进行缓存。
 - 目的是《Dubbo 用户指南 —— 事件通知》。
- 第 179 行：省略【服务引用】逻辑。

666. 彩蛋



一本正经的水更
和 [《API 配置（二）之服务提供者》](#) 大体类似。