title: 精尽 Dubbo 源码分析 —— 属性配置 date: 2018-01-16 tags: categories: Dubbo permalink: Dubbo/configuration-properties

---

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

---

1. 概述

首先，我们来看看**属性配置**的定义：

> FROM 《Dubbo 用户指南 —— 属性配置》

> 如果公共配置很**简单**，没有多注册中心，多协议等情况，或者想多个 Spring 容器想共享配置，可以使用 `dubbo.properties` 作为缺省配置。
>
> Dubbo 将自动加载 classpath 根目录下的 `dubbo.properties`，可以通过JVM启动参数 `-Ddubbo.properties.file=xxx.properties` 改变缺省配置位置。

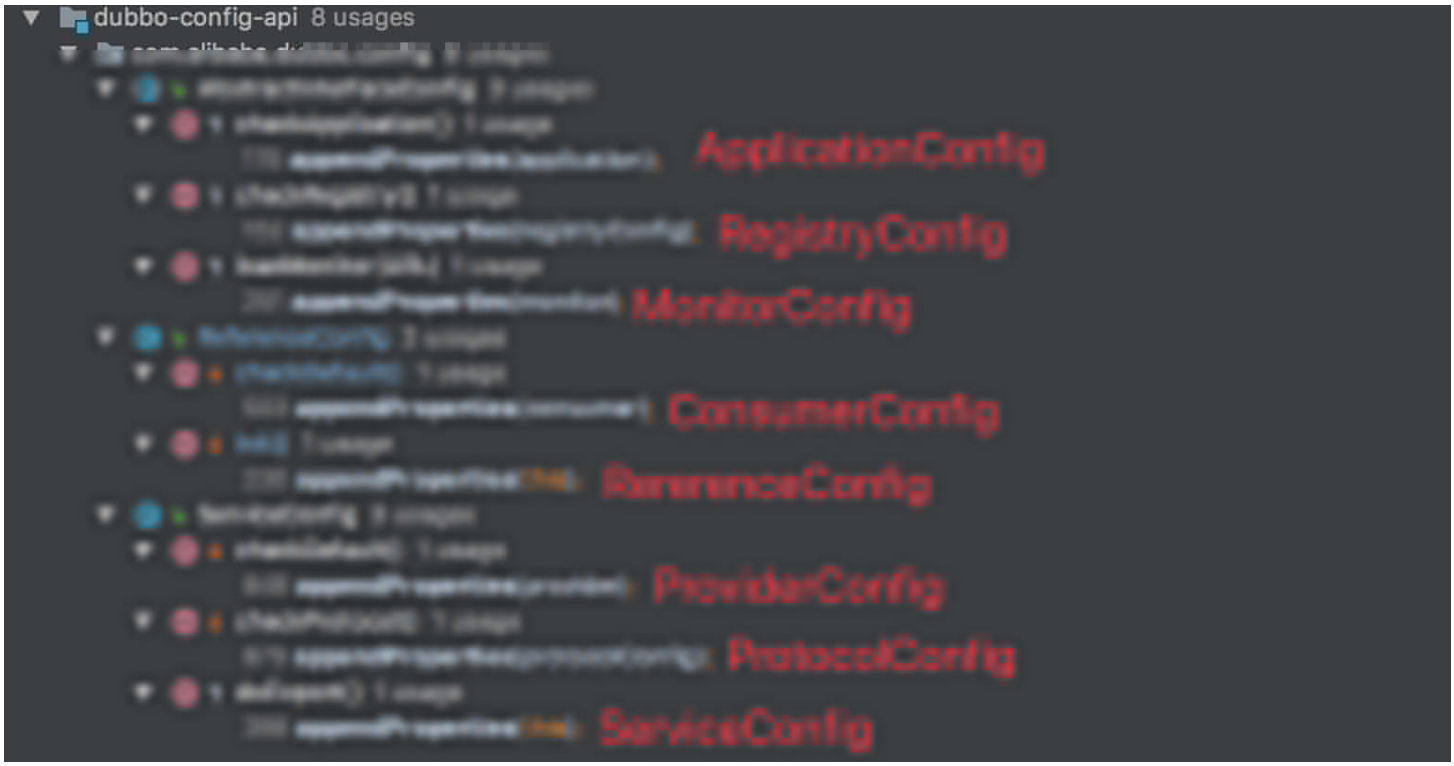
从定义上，很关键的一个词是 “**简单**” 。

- **属性配置**，不支持多注册中心，多协议等情况，原因见代码。
- **外部化配置**，能够解决上述的问题，感兴趣的胖友可以自己看下 《Dubbo 外部化配置（Externalized Configuration）》 。当然，这块内容后面分享，不在本文的范畴。

OK ，下面在开始看看具体代码之前，胖友先仔细阅读下 《Dubbo 用户指南 —— 属性配置》 ，有助于下面代码的理解。

2. AbstractConfig

在 AbstractConfig 中，提供了 `#appendProperties(config)` 方法，读取**启动参数变量**和 **properties 配置**到配置对象。在前面的几篇文章里，我们多次看到这个方法被调用，如下图所示：



代码如下：

```
1: protected static void appendProperties(AbstractConfig config) {
```

```
    if (config == null) {
        return;
    }
    String prefix = "dubbo." + getTagName(config.getClass()) + ".";
    Method[] methods = config.getClass().getMethods();
    for (Method method : methods) {
        try {
            String name = method.getName();
            if (name.length() > 3 && name.startsWith("set") && Modifier.isPublic(method.getModifiers()) // 方法是 public 的 setting 方法。
                    && method.getParameterTypes().length == 1 && isPrimitive(method.getParameterTypes()[0])) { // 方法的唯一参数是基本数据类型
                // 获得属性名，例如 `ApplicationConfig#setName(...)` 方法，对应的属性名为 name 。
                String property = StringUtils.camelToSplitName(name.substring(3, 4).toLowerCase() + name.substring(4), ".");

                // 【启动参数变量】优先从带有 `Config#id` 的配置中获取，例如：`dubbo.application.demo-provider.name` 。
                String value = null;
                if (config.getId() != null && config.getId().length() > 0) {
                    String pn = prefix + config.getId() + "." + property; // 带有 `Config#id`
                    value = System.getProperty(pn);
                    if (!StringUtils.isBlank(value)) {
                        logger.info("Use System Property " + pn + " to config dubbo");
                    }
                }
                // 【启动参数变量】获取不到，其次不带 `Config#id` 的配置中获取，例如：`dubbo.application.name` 。
                if (value == null || value.length() == 0) {
                    String pn = prefix + property; // // 不带 `Config#id`
                    value = System.getProperty(pn);
                    if (!StringUtils.isBlank(value)) {
                        logger.info("Use System Property " + pn + " to config dubbo");
                    }
                }
                if (value == null || value.length() == 0) {
                    // 覆盖优先级为：启动参数变量 > XML 配置 > properties 配置，因此需要使用 getter 判断 XML 是否已经设置
                    Method getter;
                    try {
                        getter = config.getClass().getMethod("get" + name.substring(3), new Class<?>[0]);
                    } catch (NoSuchMethodException e) {
                        try {
```

```
 39:                             getter = config.getClass().getMethod("is" + name.s
ubstring(3), new Class<?>[0]);
 40:                         } catch (NoSuchMethodException e2) {
 41:                             getter = null;
 42:                         }
 43:                     }
 44:                     if (getter != null) {
 45:                         if (getter.invoke(config, new Object[0]) == null) { //
 使用 getter 判断 XML 是否已经设置
 46:                             //  【properties 配置】优先从带有 `Config#id` 的配置中
获取，例如：`dubbo.application.demo-provider.name` 。
 47:                             if (config.getId() != null && config.getId().lengt
h() > 0) {
 48:                                 value = ConfigUtils.getProperty(prefix + confi
g.getId() + "." + property);
 49:                             }
 50:                             //  【properties 配置】获取不到，其次不带 `Config#id`
的配置中获取，例如：`dubbo.application.name` 。
 51:                             if (value == null || value.length() == 0) {
 52:                                 value = ConfigUtils.getProperty(prefix + prope
rty);
 53:                             }
 54:                             //  【properties 配置】老版本兼容，获取不到，最后不带 `
Config#id` 的配置中获取，例如：`dubbo.protocol.name` 。
 55:                             if (value == null || value.length() == 0) {
 56:                                 String legacyKey = legacyProperties.get(prefix
 + property);
 57:                                 if (legacyKey != null && legacyKey.length() >
0) {
 58:                                     value = convertLegacyValue(legacyKey, Conf
igUtils.getProperty(legacyKey));
 59:                                 }
 60:                             }
 61:
 62:                         }
 63:                     }
 64:                 }
 65:                 // 获取到值，进行反射设置。
 66:                 if (value != null && value.length() > 0) {
 67:                     method.invoke(config, new Object[]{convertPrimitive(method
.getParameterTypes()[0], value)});
 68:                 }
 69:             }
 70:         } catch (Exception e) {
 71:             logger.error(e.getMessage(), e);
 72:         }
 73:     }
 74: }
```

- 第 5 行：获得配置项**前缀**。此处的 `#getTagName(Class<?>)` 方法，使用配置类的类名，获得对应的属性标签。该方法代码如下：

```
1: /**
2:  * 配置类名的后缀
3:  * 例如，ServiceConfig 后缀为 Config；ServiceBean 后缀为 Bean。
4:  */
5: private static final String[] SUFFIXES = new String[]{"Config", "Bean"};
6:
7: /**
8:  * 获取类名对应的属性标签，例如，ServiceConfig 对应为 service 。
9:  *
10:  * @param cls 类名
11:  * @return 标签
12:  */
13: private static String getTagName(Class<?> cls) {
14:     String tag = cls.getSimpleName();
15:     for (String suffix : SUFFIXES) {
16:         if (tag.endsWith(suffix)) {
17:             tag = tag.substring(0, tag.length() - suffix.length());
18:             break;
19:         }
20:     }
21:     tag = tag.toLowerCase();
22:     return tag;
23: }
```

- 第 6 行：获得配置类的所有方法，用于下面通过**反射**获得配置项的属性名，再用属性名，去读取**启动参数变量**和 **properties 配置**到配置对象。

- 第 10 至 11 行：public && setting 方法 && **唯一**参数为基本类型。

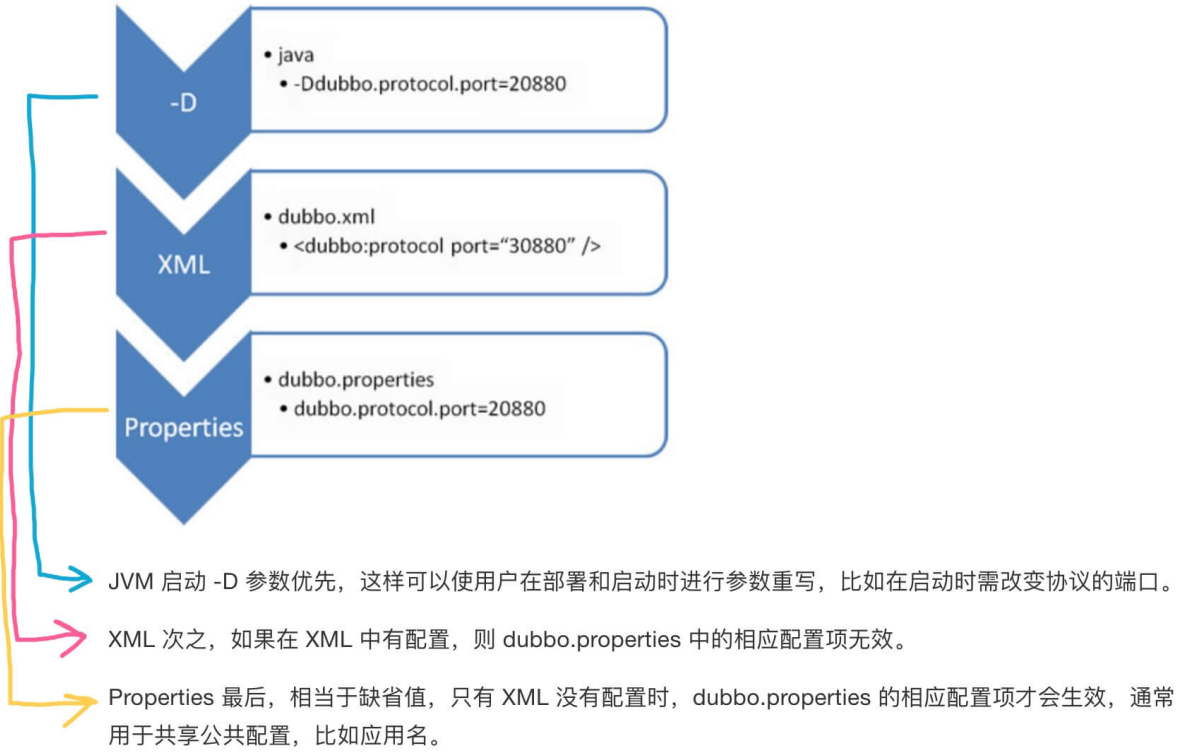
  - 其中**唯一**参数为**基本类型**，决定了一个配置对象无法设置另外一个配置对象**数组**为属性，即**没有多注册中心，多协议等情况**。例如，ServiceConfig 无法通过**属性配置**设置多个 ProtocolConfig 对象。
  - 当然上述问题，正如文初所说，《Dubbo 外部化配置（Externalized Configuration）》已经支持。

  - 另外，**属性配置**和**外部化配置**有一定的相似点：

    > FROM 《Dubbo 外部化配置（Externalized Configuration）》
    >
    > 在 Dubbo 官方用户手册的“属性配置”章节中，`dubbo.properties` 配置属性能够映

> 射到 `ApplicationConfig` 、 `ProtocolConfig` 以及 `RegistryConfig` 的字段。从某种意义上来说， `dubbo.properties` 也是 Dubbo 的外部化配置。

- 第 13 行：获得属性名。例如， `ApplicationConfig#setName(...)` 方法，对应的属性名为 `"name"` 。
- 读取的**覆盖策略**如下：

> 
>

- 第 15 至 31 行：优先从**【启动参数变量】**获取配置项的值。
  - 🙂 有**两种**情况，胖友细看下注释。
- 第 33 至 45 行：因为 XML配置 的优先级**大于** properties配置，因此需要**获取并使用** getting 方法，判断配置对象**已经拥有**该配置项的值。如果有，则不从 properties配置 读取对应的值。
- 第 46 至 59 行：最后从**【properties配置】**获取配置项的值。
  - 🙂 有**三种**情况，前两种和**【启动参数变量】**相同。
  - 最后一种，主要是兼容老版本的配置项。代码如下：

```
1: /**
2:  * 新老版本的 properties 的 key 映射
3:  *
4:  * key：新版本的配置 映射
```

```
 5:  * value：旧版本的配置 映射
 6:  *
 7:  * 来自 2012/3/8 下午 5：51 cb1f705 提交
 8:  * DUBBO-251 增加API覆盖dubbo.properties的测试，以及旧版本配置项测试。
 9:  */
10: private static final Map<String, String> legacyProperties = new HashMa
p<String, String>();
11:
12: /**
13:  * 将键对应的值转换成目标的值。
14:  *
15:  * 因为，新老配置可能有一些差异，通过该方法进行转换。
16:  *
17:  * @param key 键
18:  * @param value 值
19:  * @return 转换后的值
20:  */
21: private static String convertLegacyValue(String key, String value) {
22:     if (value != null && value.length() > 0) {
23:         if ("dubbo.service.max.retry.providers".equals(key)) {
24:             return String.valueOf(Integer.parseInt(value) - 1);
25:         } else if ("dubbo.service.allow.no.provider".equals(key)) {
26:             return String.valueOf(!Boolean.parseBoolean(value));
27:         }
28:     }
29:     return value;
30: }
```

- x

- 第 65 至 68 行：有值，通过反射进行设置到配置对象中。
- 第 70 至 72 行：逻辑中间发生异常，**不抛出异常**，仅打印错误日志。

666. 彩蛋

聚有趣的灵魂
聊有趣的技术
读有趣的源码
写有趣的代码