title: 精尽 Dubbo 源码分析 —— API 配置（一）之应用 date: 2018-01-07 tags: categories: Dubbo permalink: Dubbo/configuration-api-1

摘要: 原创出处 http://www.iocoder.cn/Dubbo/configuration-api-1/ 「芋道源码」欢迎转载，保留摘要，谢谢！

- 1. 概述
- 2. 配置一览
- 3. Config
  - 3.1 AbstractConfig
  - 3.2 ApplicationConfig
  - 3.3 RegistryConfig
  - 3.4 ModuleConfig
  - 3.5 MonitorConfig
  - 3.6 ArgumentConfig
- 4. URL
- 5. @Parameter
- 666. 彩蛋

---

1. 概述

我们都“知道”，Dubbo 的配置是非常“灵活”的。

例如，目前提供了四种**配置方式**：

1. API 配置
2. 属性配置
3. XML 配置
4. 注解配置

ps：🙂 后续的几篇文章也是按照这样的顺序，解析 Dubbo 配置的源码。

再例如，可灵活设置的**配置项**：

> FROM 《Dubbo 用户指南 —— schema 配置参考手册》
>
> 所有配置项分为三大类，参见下表中的"作用"一列。

> - 服务发现：表示该配置项用于服务的注册与发现，目的是让消费方找到提供方。
> - 服务治理：表示该配置项用于治理服务间的关系，或为开发测试提供便利条件。
> - 性能调优：表示该配置项用于调优性能，不同的选项对性能会产生影响。
>
> 所有配置最终都将转换为 Dubbo URL 表示，并由服务提供方生成，经注册中心传递给消费方，各属性对应 URL 的参数，参见配置项一览表中的 "对应URL参数" 列。

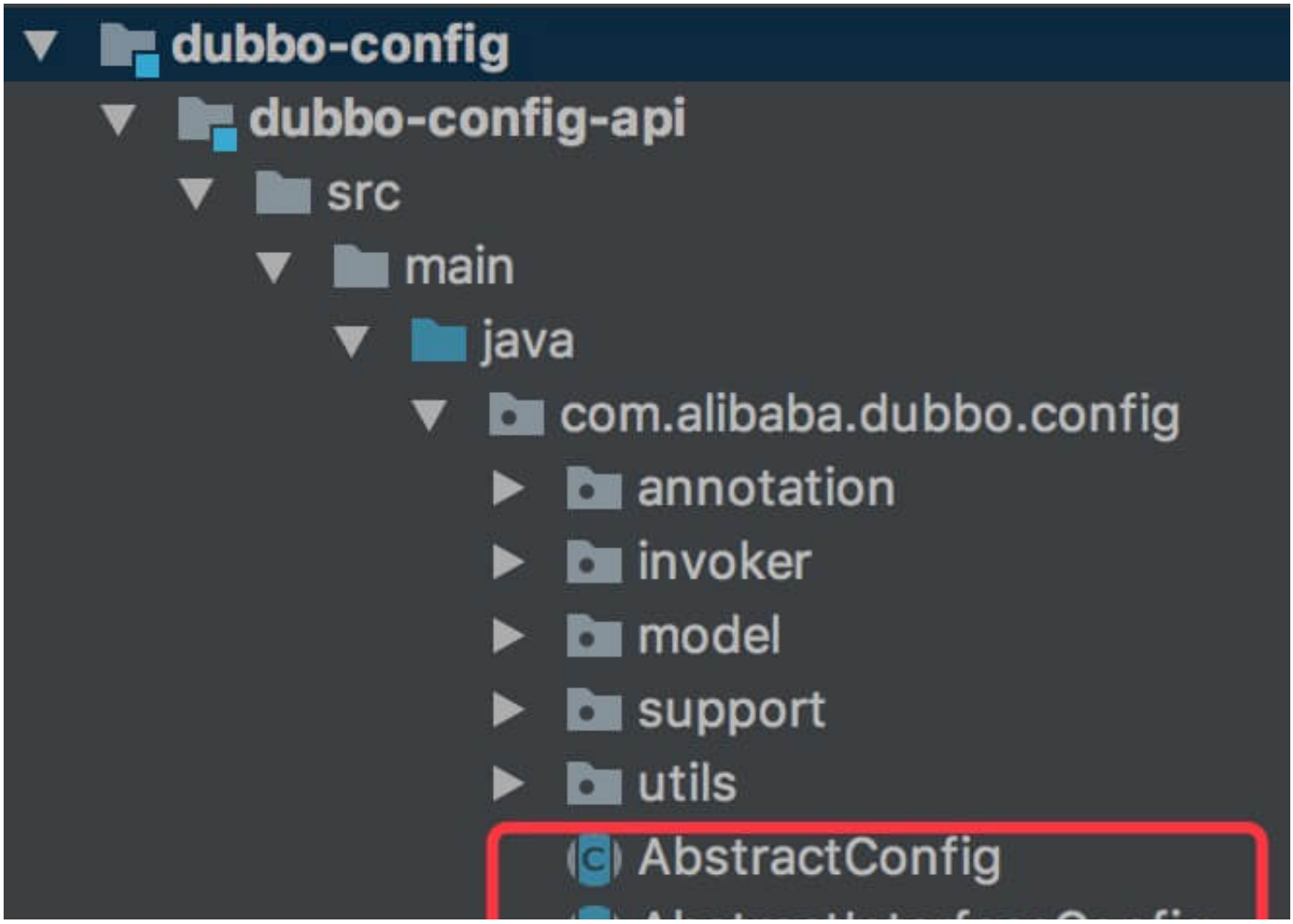
ps：🙂 可能转换成 Dubbo URL 不太好理解。良心如笔者，后续有文章会贯串它。

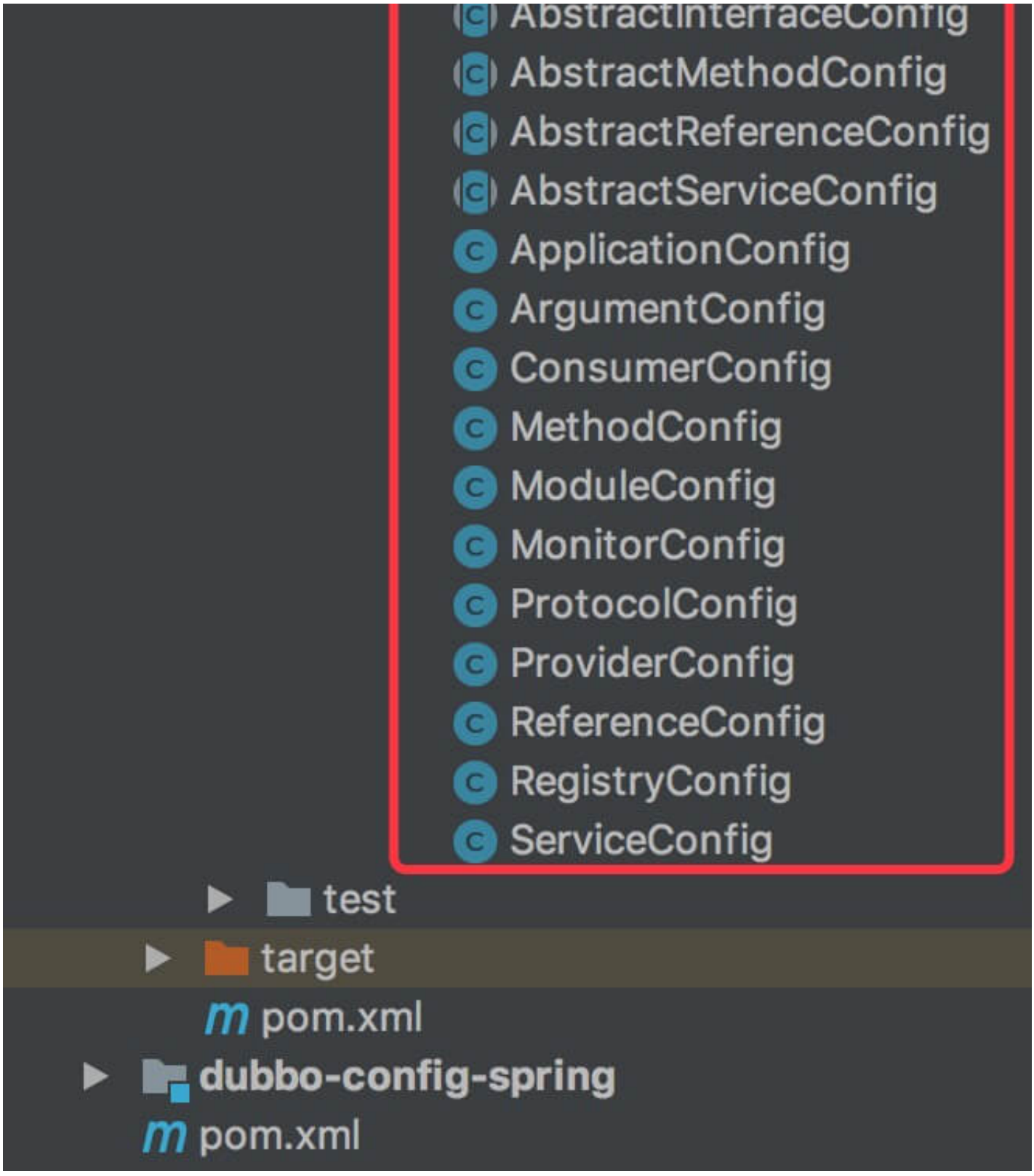
当然，凡事都有两面性，在社区里也存在建议的声音，例如：《ISSUE#738：XML配置项重新梳理》：

> 目前有一些配置项存在暴露的位置不正确、暴露不全面、文档和含义不匹配等问题，期望在2.5.7版本将已知问题予以整理修复
>
> **如果使用中有遇到的配置问题，请在评论中列出以便改进**

2. 配置一览

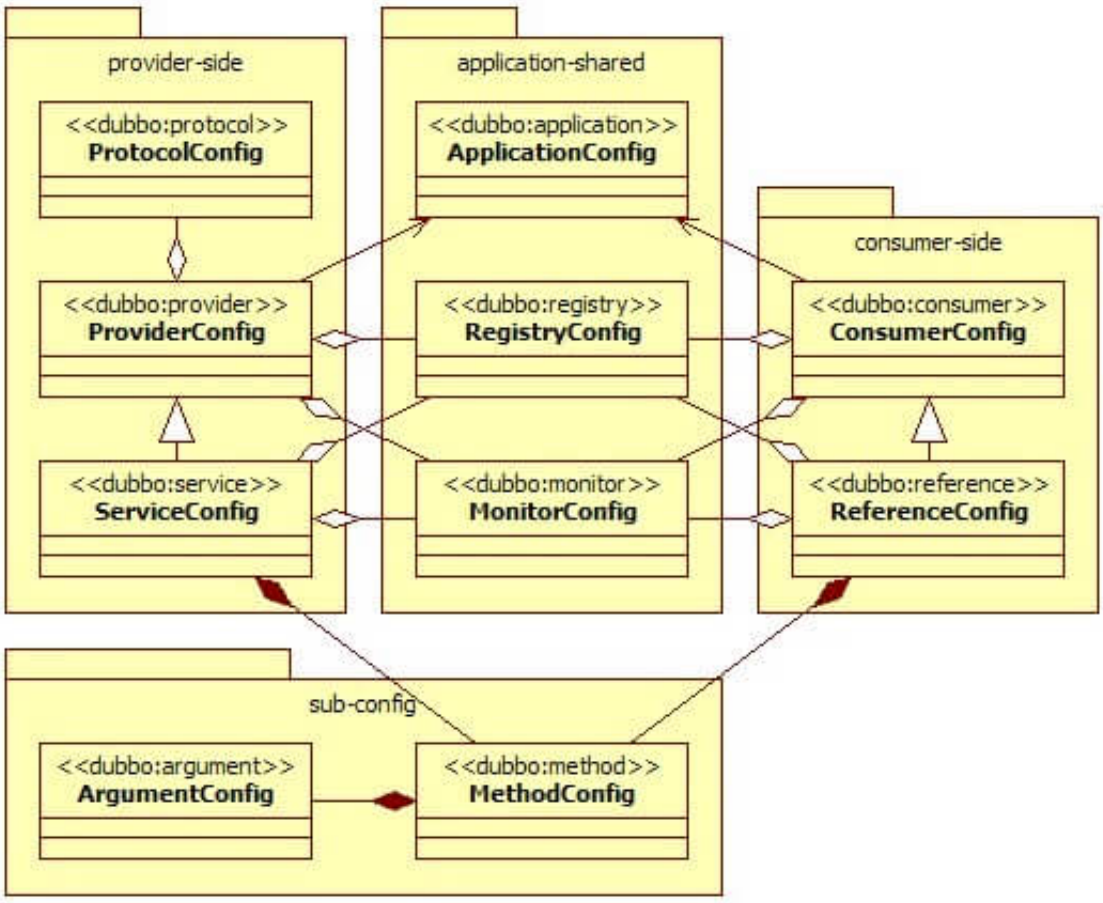
我们来看看 `dubbo-config-api` 的**项目结构**，如下图所示：





一脸懵逼，好多啊。下面我们来整理下**配置之间的关系**，如下图所示：

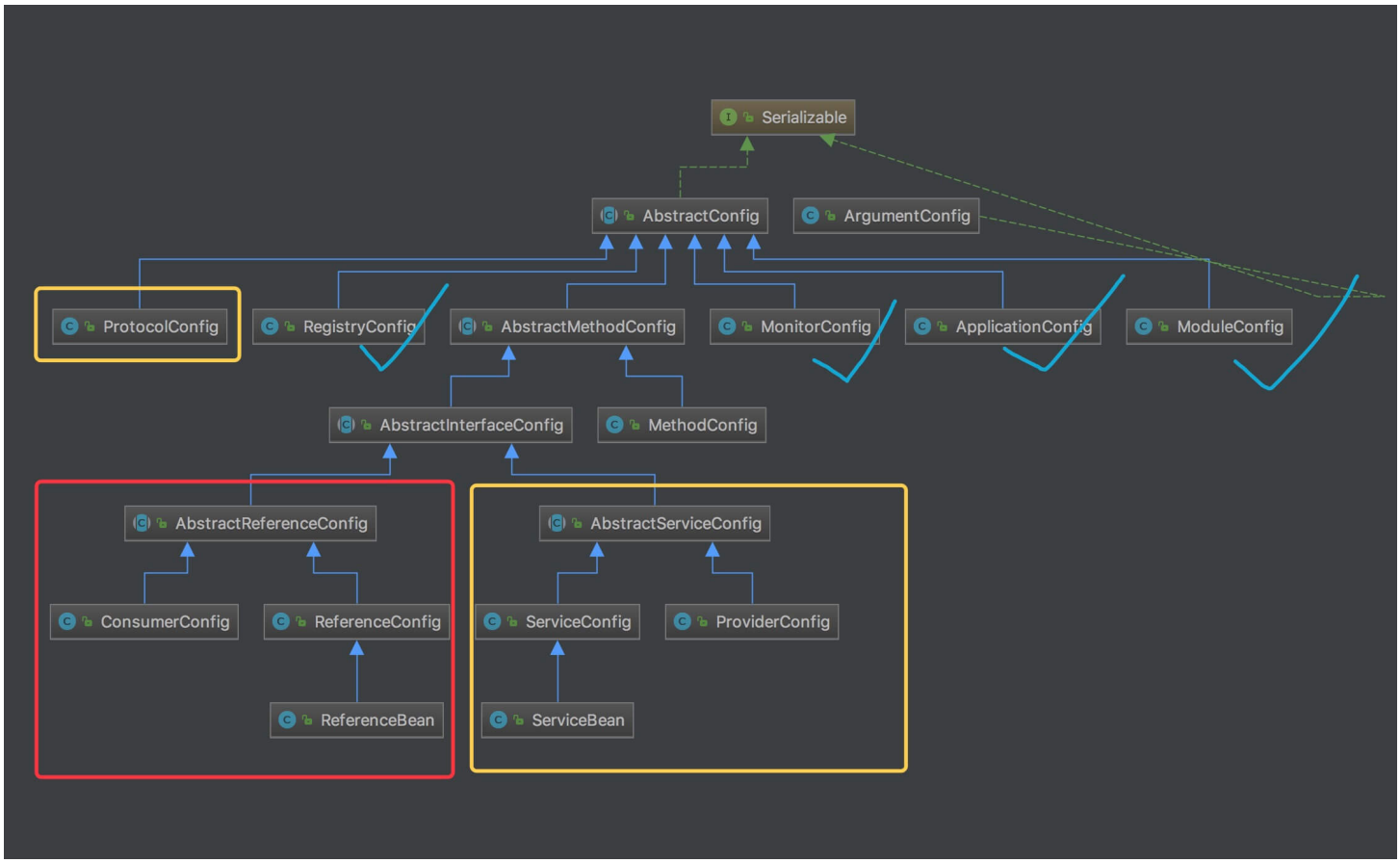
> FROM 《Dubbo 用户指南 —— XML 配置》



从这张图中，可以看出分成**四个**部分：

1. application-shared
2. provider-side
3. consumer-side
4. sub-config

实际上，上图和目前版本的代码会存在一点点出入，我们在看看实际的**类关系**，如下图所示：



- **红勾**部分，application-shared ，在本文进行分享。
- **黄框**部分，provider-side ，在 《API 配置（二）之服务提供者》 分享。
- **红框**部分，consumer-side ，在 《API 配置（三）之服务消费者》 分享。
- **其他**部分，sub-config ，在 《API 配置（二）之服务提供者》 分享。

3. Config

我们先来看一段 《Dubbo 用户指南 —— API 配置》 ，提供的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
```

- 可以看到，创建了 ApplicationConfig 和 RegistryConfig 对象，设置到 ReferenceConfig 对象。
- 如果创建 ModuleConfig 或 MonitorConfig 对象，也是可以设置到 ReferenceConfig 对象中。

3.1 AbstractConfig

`com.alibaba.dubbo.config.AbstractConfig` ，**抽象**配置类，除了 ArgumentConfig ，我们可以看到所有的配置类都继承该类。

AbstractConfig 主要提供配置解析与校验相关的工具方法。下面我们开始看看它的代码。

`id` 属性，配置对象的编号，适用于除了 API 配置**之外**的三种配置方式，标记一个配置对象，可用于对象之间的引用。例如 XML 的 `<dubbo:service provider="${PROVIDER_ID}">` ，其中 `provider` 为 `<dubbo:provider>` 的 ID 属性。

那为什么说不适用 API 配置呢？直接 `#setXXX(config)` 对象即可。

配置项校验的工具方法，例如属性值长度限制、格式限制等等，比较简单。相关代码如下：

- 静态属性
- 静态方法

`#appendParameters(parameters, config, prefix)` 方法，将配置对象的属性，添加到参数集合。代码如下：

> 在看具体代码之前，我们先来了解 「4. URL」 和 「5. @Parameter」 。

```
  1: protected static void appendParameters(Map<String, String> parameters, Object config, String prefix) {
```

```
 2:     if (config == null) {
 3:         return;
 4:     }
 5:     Method[] methods = config.getClass().getMethods();
 6:     for (Method method : methods) {
 7:         try {
 8:             String name = method.getName();
 9:             if ((name.startsWith("get") || name.startsWith("is"))
10:                     && !"getClass".equals(name)
11:                     && Modifier.isPublic(method.getModifiers())
12:                     && method.getParameterTypes().length == 0
13:                     && isPrimitive(method.getReturnType())) { // 方法为获取基本类型，public 的 getting 方法。
14:                 Parameter parameter = method.getAnnotation(Parameter.class);
15:                 if (method.getReturnType() == Object.class || parameter != null && parameter.excluded()) {
16:                     continue;
17:                 }
18:                 // 获得属性名
19:                 int i = name.startsWith("get") ? 3 : 2;
20:                 String prop = StringUtils.camelToSplitName(name.substring(i, i + 1).toLowerCase() + name.substring(i + 1), ".");
21:                 String key;
22:                 if (parameter != null && parameter.key() != null && parameter.key().length() > 0) {
23:                     key = parameter.key();
24:                 } else {
25:                     key = prop;
26:                 }
27:                 // 获得属性值
28:                 Object value = method.invoke(config, new Object[0]);
29:                 String str = String.valueOf(value).trim();
30:                 if (value != null && str.length() > 0) {
31:                     // 转义
32:                     if (parameter != null && parameter.escaped()) {
33:                         str = URL.encode(str);
34:                     }
35:                     // 拼接，详细说明参见 `Parameter#append()` 方法的说明。
36:                     if (parameter != null && parameter.append()) {
37:                         String pre = parameters.get(Constants.DEFAULT_KEY + "." + key); // default. 里获取，适用于 ServiceConfig =》ProviderConfig 、ReferenceConfig =》ConsumerConfig 。
38:                         if (pre != null && pre.length() > 0) {
39:                             str = pre + "," + str;
40:                         }
41:                         pre = parameters.get(key); // 通过 `parameters` 属性配置，例如 `AbstractMethodConfig.parameters` 。
42:                         if (pre != null && pre.length() > 0) {
```

```
 43:                                 str = pre + "," + str;
 44:                             }
 45:                         }
 46:                         if (prefix != null && prefix.length() > 0) {
 47:                             key = prefix + "." + key;
 48:                         }
 49:                         parameters.put(key, str);
 50: //                          System.out.println("kv:" + key + "\t" + str);
 51:                     } else if (parameter != null && parameter.required()) {
 52:                         throw new IllegalStateException(config.getClass().getSimpl
eName() + "." + key + " == null");
 53:                     }
 54:                 } else if ("getParameters".equals(name)
 55:                         && Modifier.isPublic(method.getModifiers())
 56:                         && method.getParameterTypes().length == 0
 57:                         && method.getReturnType() == Map.class) { // `#getParamete
rs()` 方法
 58:                     Map<String, String> map = (Map<String, String>) method.invoke(
config, new Object[0]);
 59:                     if (map != null && map.size() > 0) {
 60:                         String pre = (prefix != null && prefix.length() > 0 ? pref
ix + "." : "");
 61:                         for (Map.Entry<String, String> entry : map.entrySet()) {
 62:                             parameters.put(pre + entry.getKey().replace('-', '.'),
 entry.getValue());
 63:                         }
 64:                     }
 65:                 }
 66:             } catch (Exception e) {
 67:                 throw new IllegalStateException(e.getMessage(), e);
 68:             }
 69:         }
 70: }
```

- `parameters` ，参数集合。实际上，该集合会用于 `URL.parameters` 。
- `config` ，配置对象。
- `prefix` ，属性前缀。用于配置项添加到 `parameters` 中时的前缀。
- 第 5 行：获得所有方法的数组，为下面通过**反射**获得配置项的值做准备。
- 第 6 行：**循环**每个方法。
- 第 9 至 13 行：方法为获得**基本类型** + `public` 的 getting 方法。
  - 第 14 至 17 行：返回值类型为 Object 或排除(`@Parameter.exclue=true`)的配置项，跳过。
  - 第 19 至 26 行：获得配置项**名**。
  - 第 28 至 48 行：获得配置项**值**。中间有一些逻辑处理，胖友看下代码的注释。
  - 第 49 行：添加配置项到 `parameters` 。

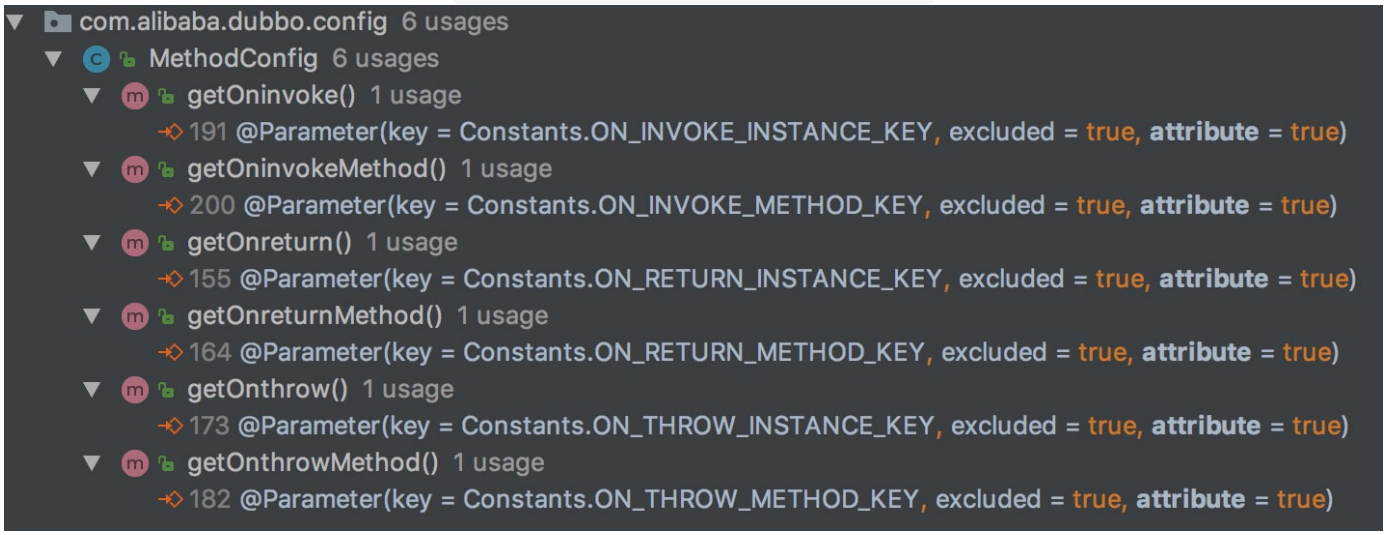
    - 第 51 至 53 行：当 `@Parameter.required = true` 时，校验配置项非空。
- 第 54 至 57 行：当方法为 `#getParameters()` 时，例如 。
    - 第 58 行：通过反射，获得 `#getParameters()` 的返回值为 `map` 。
    - 第 59 至 64 行：将 `map` 添加到 `parameters` ，kv 格式为 `prefix:entry.key` `entry.value` 。
    - 因此，通过 `#getParameters()` 对应的属性，**动态设置配置项，拓展出非 Dubbo 内置好的逻辑**。

`#appendAttributes(parameters, config, prefix)` 方法，将 `@Parameter(attribute = true)` 配置对象的属性，添加到参数集合。代码如下：

```
  1: protected static void appendAttributes(Map<Object, Object> parameters, Object
config, String prefix) {
  2:     if (config == null) {
  3:         return;
  4:     }
  5:     Method[] methods = config.getClass().getMethods();
  6:     for (Method method : methods) {
  7:         try {
  8:             String name = method.getName();
  9:             if ((name.startsWith("get") || name.startsWith("is"))
 10:                     && !"getClass".equals(name)
 11:                     && Modifier.isPublic(method.getModifiers())
 12:                     && method.getParameterTypes().length == 0
 13:                     && isPrimitive(method.getReturnType())) { // 方法为获取基本
类型，public 的 getting 方法。
 14:                 Parameter parameter = method.getAnnotation(Parameter.class);
 15:                 if (parameter == null || !parameter.attribute())
 16:                     continue;
 17:                 // 获得属性名
 18:                 String key;
 19:                 if (parameter != null && parameter.key() != null && parameter.
key().length() > 0) {
 20:                     key = parameter.key();
 21:                 } else {
 22:                     int i = name.startsWith("get") ? 3 : 2;
 23:                     key = name.substring(i, i + 1).toLowerCase() + name.substr
ing(i + 1);
 24:                 }
 25:                 // 获得属性值，存在则添加到 `parameters` 集合
 26:                 Object value = method.invoke(config, new Object[0]);
 27:                 if (value != null) {
 28:                     if (prefix != null && prefix.length() > 0) {
 29:                         key = prefix + "." + key;
 30:                     }
```

```
31:                     parameters.put(key, value);
32:                 }
33:             }
34:         } catch (Exception e) {
35:             throw new IllegalStateException(e.getMessage(), e);
36:         }
37:     }
38: }
```

- 不同于 `#appendAttributes(parameters, config, prefix)` 方法，主要用于 《Dubbo 用户指南 —— 事件通知》 ，注解 `@Parameter(attribute = true)` 的属性如下图：



- 第 9 至 13 行：方法为获得**基本类型** + `public` 的 getting 方法。
- 第 14 至 16 行：**需要**(`@Parameter.exclue=true`)的配置项。
- 第 17 至 24 行：获得配置项**名**。
- 第 26 至 30 行：获得配置项**值**。
- 第 31 行：添加配置项到 `parameters` 。

`#appendProperties(config)` 方法，读取环境变量和 properties 配置到配置对象。在 《精进 Dubbo 源码解析 —— 属性配置》 详细解析。

`#appendAnnotation(annotationClass, annotation)` 方法，读取注解配置到配置对象。在 《精进 Dubbo 源码解析 —— 注解配置》 详细解析。

3.2 ApplicationConfig

`com.alibaba.dubbo.config.ApplicationConfig` ，应用配置。

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

3.3 RegistryConfig

`com.alibaba.dubbo.config.RegistryConfig` ，注册中心配置。

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

3.4 ModuleConfig

`com.alibaba.dubbo.config.ModuleConfig` ，模块信息配置。

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

3.5 MonitorConfig

`com.alibaba.dubbo.config.MonitorConfig` ，监控中心配置。

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

3.6 ArgumentConfig

`com.alibaba.dubbo.config.ArgumentConfig` ，方法参数配置。

- 具体属性的解释，参见 《Dubbo 用户指南 —— dubbo:argument》 文档。
- 该配置类设置到 MethodConfig 对象中，在 《API 配置（二）之服务提供者》 我们会看到。
- 在 《Dubbo 用户指南 —— 参数回调》 特性中使用。

4. URL

`com.alibaba.dubbo.common.URL` ，Dubbo URL 。代码如下：

```
public final class URL implements Serializable {

    /**
     * 协议名
     */
    private final String protocol;
    /**
```

```
     * 用户名
     */
    private final String username;
    /**
     * 密码
     */
    private final String password;
    /**
     * by default, host to registry
     * 地址
     */
    private final String host;
    /**
     * by default, port to registry
     * 端口
     */
    private final int port;
    /**
     * 路径（服务名）
     */
    private final String path;
    /**
     * 参数集合
     */
    private final Map<String, String> parameters;

    // ... 省略其他代码

}
```

- 上文我们提到**所有配置最终都将转换为 Dubbo URL 表示，并由服务提供方生成，经注册中心传递给消费方，各属性对应 URL 的参数，参见配置项一览表中的 "对应URL参数" 列**。那么一个 Service 注册到注册中心的格式如下：

```
dubbo://192.168.3.17:20880/com.alibaba.dubbo.demo.DemoService?anyhost=true&application=demo-provider&default.delay=-1&default.retries=0&default.service.filter=demoFilter&delay=-1&dubbo=2.0.0&generic=false&interface=com.alibaba.dubbo.demo.DemoService&methods=sayHello&pid=19031&side=provider&timestamp=1519651641799
```

  - 格式为 `protocol://username:password@host:port/path?key=value&key=value` ，通过 `URL#buildString(...)` 方法生成。

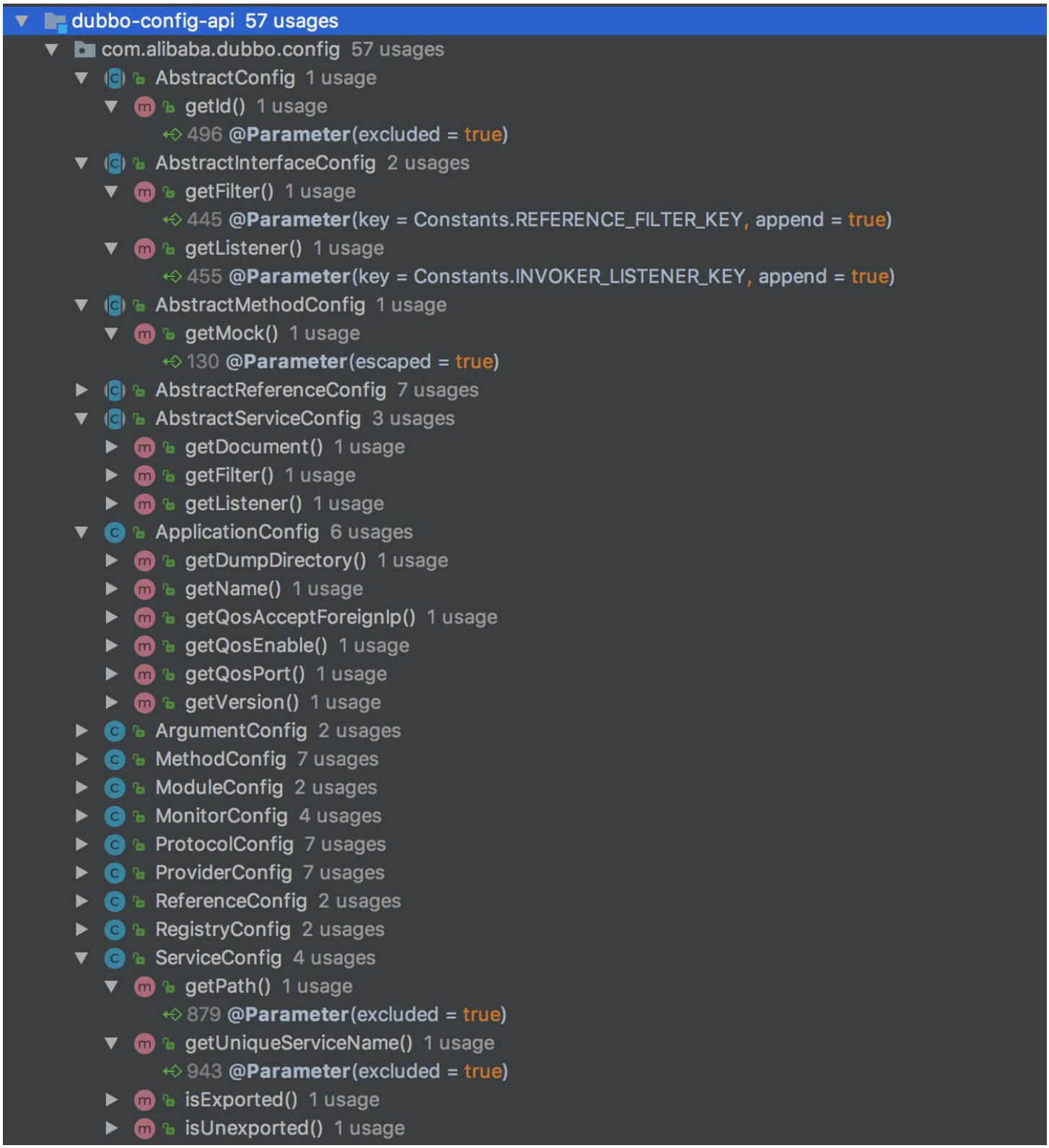
- `parameters` 属性，参数集合。从上面的 Service URL 例子我们可以看到，里面的 `key=value` ，实际上就是 Service 对应的配置项。该属性，通过 `AbstractConfig#appendParameters(parameters, config, prefix)` 方法生成。

- 🙂 在后续的文章中，我们会发现 URL 作为一个**通用模型**，贯穿整个 RPC 流程。

5. @Parameter

`com.alibaba.dubbo.config.support.@Parameter` ，Parameter 参数**注解**，用于 Dubbo URL 的 `parameters` 拼接。

在配置对象的 getting 方法上，我们可以看到该注解的使用，例如下图：



@Parameter 代码如下：

```
@Documented
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.METHOD})
public @interface Parameter {

    /**
     * 键（别名）
```

```
     */
    String key() default "";

    /**
     * 是否必填
     */
    boolean required() default false;

    /**
     * 是否忽略
     */
    boolean excluded() default false;

    /**
     * 是否转义
     */
    boolean escaped() default false;

    /**
     * 是否为属性
     *
     * 目前用于《事件通知》http://dubbo.io/books/dubbo-user-book/demos/events-notify.html
     */
    boolean attribute() default false;

    /**
     * 是否拼接默认属性，参见 {@link com.alibaba.dubbo.config.AbstractConfig#appendParameters(Map, Object, String)} 方法。
     *
     * 我们来看看 `#append() = true` 的属性，有如下四个：
     *   + {@link AbstractInterfaceConfig#getFilter()}
     *   + {@link AbstractInterfaceConfig#getListener()}
     *   + {@link AbstractReferenceConfig#getFilter()}
     *   + {@link AbstractReferenceConfig#getListener()}
     *   + {@link AbstractServiceConfig#getFilter()}
     *   + {@link AbstractServiceConfig#getListener()}
     * 那么，以 AbstractServiceConfig 举例子。
     *
     * 我们知道 ProviderConfig 和 ServiceConfig 继承 AbstractServiceConfig 类，那么 `filter` , `listener` 对应的相同的键。
     * 下面我们以 `filter` 举例子。
     *
     * 在 ServiceConfig 中，默认会<b>继承</b> ProviderConfig 配置的 `filter` 和 `listener` 。
     * 所以这个属性，就是用于，像 ServiceConfig 的这种情况，从 ProviderConfig 读取父属性。
     *
```

```
     * 举个例子，如果 `ProviderConfig.filter=aaaFilter` ，`ServiceConfig.filter=bbbFilter` ，最终暴露到 Dubbo URL 时，参数为 `service.filter=aaaFilter,bbbFilter` 。
     */
    boolean append() default false;
```

- 胖友可以简单看下代码中的**注释**，结合具体使用的方法，在细细理解。

666. 彩蛋

可能胖友看完之后，可能会有点懵逼。
貌似没什么实质的内容。
关于配置的文章，主要用于后续的初始化做铺垫。