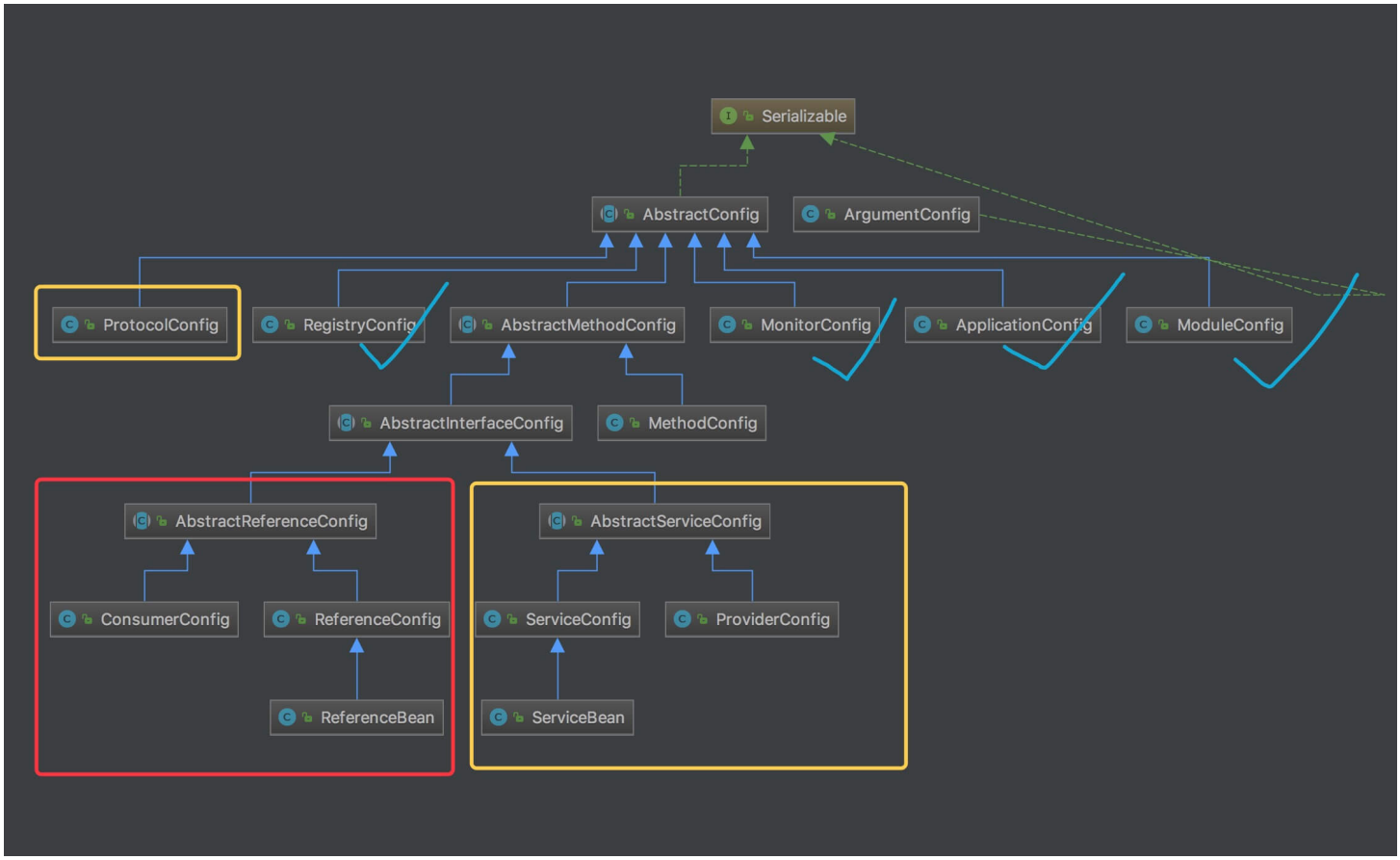
title: 精尽 Dubbo 源码分析 —— API 配置（二）之服务提供者 date: 2018-01-10 tags: categories: Dubbo permalink: Dubbo/configuration-api-2

摘要: 原创出处 http://www.iocoder.cn/Dubbo/configuration-api-2/ 「芋道源码」欢迎转载，保留摘要，谢谢！

---

1. 概述

本文接 《API 配置（一）之应用》 ，分享**服务提供者**相关的配置：包括 provider-config 和 sub-config 部分。



- **黄框**部分，provider-side
- **其他**部分，sub-config

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

```
// 服务实现
XxxService xxxService = new XxxServiceImpl();

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

// 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");

// 服务提供者协议配置
ProtocolConfig protocol = new ProtocolConfig();
protocol.setName("dubbo");
protocol.setPort(12345);
protocol.setThreads(200);

// 注意：ServiceConfig为重对象，内部封装了与注册中心的连接，以及开启服务端口

// 服务提供者暴露服务配置
ServiceConfig<XxxService> service = new ServiceConfig<XxxService>(); // 此实例很重，封装了与注册中心的连接，请自行缓存，否则可能造成内存和连接泄漏
service.setApplication(application);
service.setRegistry(registry); // 多个注册中心可以用setRegistries()
service.setProtocol(protocol); // 多个协议可以用setProtocols()
service.setInterface(XxxService.class);
service.setRef(xxxService);
service.setVersion("1.0.0");

// 暴露及注册服务
service.export();
```

- 相比 ReferenceConfig 的初始化，会多创建 ProtocolConfig 对象，设置到 ServiceConfig 对象中。

> 友情提示：本文前面部分会比较琐碎，重点在 「8. ServiceConfig」 部分。

2. ProtocolConfig

`com.alibaba.dubbo.config.ProtocolConfig` ，服务提供者协议配置。

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

3. AbstractMethodConfig

`com.alibaba.dubbo.config.AbstractMethodConfig` ，方法级配置的抽象类。

- **部分**属性的解释，参见 《Dubbo 用户指南 —— dubbo:method》 文档。

4. MethodConfig

`com.alibaba.dubbo.config.MethodConfig` ，继承 AbstractMethodConfig ，方法级配置。

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

5. AbstractInterfaceConfig

`com.alibaba.dubbo.config.AbstractInterfaceConfig` ，继承 AbstractMethodConfig ，抽象接口配置类。

- 具体属性的解释，**需要寻找**在 《Dubbo 用户指南 —— dubbo:service》 或 《Dubbo 用户指南 —— dubbo:reference》 文档。
- 下面的方法，会在 「8. ServiceConfig」 的初始化被调用，胖友可需要的时候，点击查看。
- `#checkApplication()` 方法，校验 ApplicationConfig 配置。**实际上**，该方法会初始化 ApplicationConfig 的配置属性。
  - 🙂 直接点击方法查看，较为简单，已经添加详细注释。
- `#checkRegistry()` 方法，校验 RegistryConfig 配置。**实际上**，该方法会初始化 RegistryConfig 的配置属性。
  - 🙂 直接点击方法查看，较为简单，已经添加详细注释。
- `#checkInterfaceAndMethods(interfaceClass, methods)` 方法，校验接口和方法。主要是两方面：
  - 1、 接口类非空，并是接口
  - 2、 方法在接口中已定义
  - 🙂 直接点击方法查看，较为简单，已经添加详细注释。
- `#checkStubAndMock(interfaceClass)` 方法，校验 Stub 和 Mock 相关的配置。
  - 【TODO 8005】 芋艿，后续继续研究
- 以上未列举的 `#loadRegistries(provider)` 和 `#loadMonitor(registryURL)` 方法，在**后续文章**需要使用到时，在详细分享。

6. AbstractServiceConfig

`com.alibaba.dubbo.config.AbstractServiceConfig` ，实现 AbstractInterfaceConfig ，抽象服务配置类。

- 具体属性的解释，**需要寻找**在 《Dubbo 用户指南 —— dubbo:service》 或 《Dubbo 用户

指南 —— dubbo:provider》 文档。

7. ProviderConfig

`com.alibaba.dubbo.config.ProviderConfig` ，实现 AbstractServiceConfig ，服务提供者缺省值配置。

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

8. ServiceConfig

`com.alibaba.dubbo.config.ServiceConfig` ，服务提供者暴露**服务配置类**。

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

下面，我们进入**正戏**。

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

1. **进一步初始化** ServiceConfig 对象。
2. **校验** ServiceConfig 对象的配置项。
3. 使用 ServiceConfig 对象，**生成** Dubbo URL 对象**数组**。
4. 使用 Dubbo URL 对象，**暴露服务**。

😈 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 10 行：当 `export` 或 `delay` 未配置时，从 ProviderConfig 对象读取。
- 第 11 至 14 行：当配置不需要暴露服务(`export = false`)时，直接返回。
- 第 17 至 22 行：当配置延迟暴露(`delay > 0`)时，使用 `delayExportExecutor` **延迟**调度，调用 `#doExport()` 方法。
  - 《Dubbo 用户指南 —— 延迟暴露》
- 第 23 至 26 行：立即暴露，调用 `#doExport()` 方法。

`#doExport()` 方法，代码如下：

```
  1: protected synchronized void doExport() {
  2:     // 检查是否可以暴露，若可以，标记已经暴露。
  3:     if (unexported) {
  4:         throw new IllegalStateException("Already unexported!");
  5:     }
  6:     if (exported) {
  7:         return;
  8:     }
  9:     exported = true;
 10:     // 校验接口名非空
 11:     if (interfaceName == null || interfaceName.length() == 0) {
 12:         throw new IllegalStateException("<dubbo:service interface=\"\" /> inte
rface not allow null!");
 13:     }
 14:     // 拼接属性配置（环境变量 + properties 属性）到 ProviderConfig 对象
 15:     checkDefault();
 16:     // 从 ProviderConfig 对象中，读取 application、module、registries、monitor、
protocols 配置对象。
 17:     if (provider != null) {
 18:         if (application == null) {
 19:             application = provider.getApplication();
 20:         }
 21:         if (module == null) {
 22:             module = provider.getModule();
 23:         }
 24:         if (registries == null) {
 25:             registries = provider.getRegistries();
 26:         }
 27:         if (monitor == null) {
 28:             monitor = provider.getMonitor();
 29:         }
 30:         if (protocols == null) {
 31:             protocols = provider.getProtocols();
 32:         }
 33:     }
 34:     // 从 ModuleConfig 对象中，读取 registries、monitor 配置对象。
```

```
 35:     if (module != null) {
 36:         if (registries == null) {
 37:             registries = module.getRegistries();
 38:         }
 39:         if (monitor == null) {
 40:             monitor = module.getMonitor();
 41:         }
 42:     }
 43:     // 从 ApplicationConfig 对象中，读取 registries、monitor 配置对象。
 44:     if (application != null) {
 45:         if (registries == null) {
 46:             registries = application.getRegistries();
 47:         }
 48:         if (monitor == null) {
 49:             monitor = application.getMonitor();
 50:         }
 51:     }
 52:     // 泛化接口的实现
 53:     if (ref instanceof GenericService) {
 54:         interfaceClass = GenericService.class;
 55:         if (StringUtils.isEmpty(generic)) {
 56:             generic = Boolean.TRUE.toString();
 57:         }
 58:     // 普通接口的实现
 59:     } else {
 60:         try {
 61:             interfaceClass = Class.forName(interfaceName, true, Thread.current
Thread().getContextClassLoader());
 62:         } catch (ClassNotFoundException e) {
 63:             throw new IllegalStateException(e.getMessage(), e);
 64:         }
 65:         // 校验接口和方法
 66:         checkInterfaceAndMethods(interfaceClass, methods);
 67:         // 校验指向的 service 对象
 68:         checkRef();
 69:         generic = Boolean.FALSE.toString();
 70:     }
 71:     // 处理服务接口客户端本地代理( `local` )相关。实际目前已经废弃，使用 `stub` 属性
，参见 `AbstractInterfaceConfig#setLocal` 方法。
 72:     if (local != null) {
 73:         // 设为 true，表示使用缺省代理类名，即：接口名 + Local 后缀
 74:         if ("true".equals(local)) {
 75:             local = interfaceName + "Local";
 76:         }
 77:         Class<?> localClass;
 78:         try {
 79:             localClass = ClassHelper.forNameWithThreadContextClassLoader(local
);
```

```
 80:         } catch (ClassNotFoundException e) {
 81:             throw new IllegalStateException(e.getMessage(), e);
 82:         }
 83:         if (!interfaceClass.isAssignableFrom(localClass)) {
 84:             throw new IllegalStateException("The local implementation class "
+ localClass.getName() + " not implement interface " + interfaceName);
 85:         }
 86:     }
 87:     // 处理服务接口客户端本地代理( `stub` )相关
 88:     if (stub != null) {
 89:         // 设为 true，表示使用缺省代理类名，即：接口名 + Stub 后缀
 90:         if ("true".equals(stub)) {
 91:             stub = interfaceName + "Stub";
 92:         }
 93:         Class<?> stubClass;
 94:         try {
 95:             stubClass = ClassHelper.forNameWithThreadContextClassLoader(stub);
 96:         } catch (ClassNotFoundException e) {
 97:             throw new IllegalStateException(e.getMessage(), e);
 98:         }
 99:         if (!interfaceClass.isAssignableFrom(stubClass)) {
100:             throw new IllegalStateException("The stub implementation class " +
 stubClass.getName() + " not implement interface " + interfaceName);
101:         }
102:     }
103:     // 校验 ApplicationConfig 配置。
104:     checkApplication();
105:     // 校验 RegistryConfig 配置。
106:     checkRegistry();
107:     // 校验 ProtocolConfig 配置数组。
108:     checkProtocol();
109:     // 读取环境变量和 properties 配置到 ServiceConfig 对象。
110:     appendProperties(this);
111:     // 校验 Stub 和 Mock 相关的配置
112:     checkStubAndMock(interfaceClass);
113:     // 服务路径，缺省为接口名
114:     if (path == null || path.length() == 0) {
115:         path = interfaceName;
116:     }
117:     // TODO 芋艿
118:     doExportUrls();
119:     // TODO 芋艿，等待 qos
120:     ProviderModel providerModel = new ProviderModel(getUniqueServiceName(), th
is, ref);
121:     ApplicationModel.initProviderModel(getUniqueServiceName(), providerModel);
122: }
```

- 第 2 至 9 行：检查是否可以暴露。若可以，标记已经暴露(`exported = true`)。
- 第 10 至 13 行：校验接口名 `interfaceName` 非空。
- 第 15 行：调用 `#checkDefault()` 方法，读取**属性配置**(环境变量 + properties 属性)到 ProviderConfig 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 16 至 33 行：从 ProviderConfig 对象中，读取 `application`、`module`、`registries`、`monitor`、`protocols` 对象。
- 第 34 至 42 行：从 ModuleConfig 对象中，读取 `registries`、`monitor` 对象。
- 第 43 至 51 行：从 ApplicationConfig 对象中，读取 `registries`、`monitor` 对象。
- 第 52 至 57 行：泛化接口的实现。
  - 《Dubbo 用户指南 —— 泛化接口》
- 第 58 至 70 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 66 行：调用 `#checkInterfaceAndMethods(interfaceClass, methods)` 方法，检查接口和方法。
    - 🙂 本文有已经有这个方法的解析。
  - 第 68 行：调用 `#checkRef()` 方法，校验指向的 Service 对象。
  - 第 69 行：标记 `generic` 为**非**泛化实现。
- 第 71 至 86 行：处理服务接口客户端本地代理(`local`)相关。**实际目前已经废弃，此处主要用于兼容**，使用 `stub` 属性，参见 `AbstractInterfaceConfig#setLocal(local)` 方法的**注释说明**。
- 第 87 至 102 行：处理服务接口客户端本地代理(`stub` 属性，参见 `AbstractInterfaceConfig#setLocal(local)`)相关。
- 第 104 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 106 行：调用 `#checkRegistry()` 方法，校验 RegistryConfig 配置。
  - 🙂 直接点击方法查看，较为简单，已经添加详细注释。
- 第 108 行：调用 `#checkProtocol()` 方法，校验 ProtocolConfig 配置数组。
  - 🙂 直接点击方法查看，较为简单，已经添加详细注释。
- 第 110 行：调用 `#appendProperties(config)` 方法，读取**属性配置**(环境变量 + properties 属性)到 ServiceConfig 对象（**自己**）。
- 第 112 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 113 至 116 行：服务路径 `path` 为空时，缺省为接口名。
- 第 118 行：调用 `#doExportUrls()` 方法，**暴露服务**。此方法包含了我们上述的 **3+4** 部分。

- 第 119 至 121 行：// TODO 芋艿，等待 qos

因为本文不分享 **4** 部分，所以下面我们只看 `#doExportUrls()` 方法中，调用 `#doExportUrlsFor1Protocol(protocolConfig, registryURLs)` 方法，和 **3** 有关的部分。代码如下：

```
  1: private void doExportUrlsFor1Protocol(ProtocolConfig protocolConfig, List<URL>
 registryURLs) {
  2:     // 协议名
  3:     String name = protocolConfig.getName();
  4:     if (name == null || name.length() == 0) {
  5:         name = "dubbo";
  6:     }
  7:
  8:     // 将 `side`, `dubbo`, `timestamp`, `pid` 参数，添加到 `map` 集合中。
  9:     Map<String, String> map = new HashMap<String, String>();
 10:     map.put(Constants.SIDE_KEY, Constants.PROVIDER_SIDE);
 11:     map.put(Constants.DUBBO_VERSION_KEY, Version.getVersion());
 12:     map.put(Constants.TIMESTAMP_KEY, String.valueOf(System.currentTimeMillis()
));
 13:     if (ConfigUtils.getPid() > 0) {
 14:         map.put(Constants.PID_KEY, String.valueOf(ConfigUtils.getPid()));
 15:     }
 16:     // 将各种配置对象，添加到 `map` 集合中。
 17:     appendParameters(map, application);
 18:     appendParameters(map, module);
 19:     appendParameters(map, provider, Constants.DEFAULT_KEY); // ProviderConfig
, 为 ServiceConfig 的默认属性，因此添加 `default` 属性前缀。
 20:     appendParameters(map, protocolConfig);
 21:     appendParameters(map, this);
 22:     // 将 MethodConfig 对象数组，添加到 `map` 集合中。
 23:     if (methods != null && !methods.isEmpty()) {
 24:         for (MethodConfig method : methods) {
 25:             // 将 MethodConfig 对象，添加到 `map` 集合中。
 26:             appendParameters(map, method, method.getName());
 27:             // 当 配置了 `MethodConfig.retry = false` 时，强制禁用重试
 28:             String retryKey = method.getName() + ".retry";
 29:             if (map.containsKey(retryKey)) {
 30:                 String retryValue = map.remove(retryKey);
 31:                 if ("false".equals(retryValue)) {
 32:                     map.put(method.getName() + ".retries", "0");
 33:                 }
 34:             }
 35:             // 将 ArgumentConfig 对象数组，添加到 `map` 集合中。
 36:             List<ArgumentConfig> arguments = method.getArguments();
 37:             if (arguments != null && !arguments.isEmpty()) {
 38:                 for (ArgumentConfig argument : arguments) {
```

```
 39:                     // convert argument type
 40:                     if (argument.getType() != null && argument.getType().lengt
h() > 0) { // 指定了类型
 41:                         Method[] methods = interfaceClass.getMethods();
 42:                         // visit all methods
 43:                         if (methods != null && methods.length > 0) {
 44:                             for (int i = 0; i < methods.length; i++) {
 45:                                 String methodName = methods[i].getName();
 46:                                 // target the method, and get its signature
 47:                                 if (methodName.equals(method.getName())) { //
找到指定方法
 48:                                     Class<?>[] argTypes = methods[i].getParame
terTypes();
 49:                                     // one callback in the method
 50:                                     if (argument.getIndex() != -1) { // 指定单
个参数的位置 + 类型
 51:                                         if (argTypes[argument.getIndex()].getN
ame().equals(argument.getType())) {
 52:                                             // 将 ArgumentConfig 对象，添加到 `m
ap` 集合中。
 53:                                             appendParameters(map, argument, me
thod.getName() + "." + argument.getIndex()); // `${methodName}.${index}`
 54:                                         } else {
 55:                                             throw new IllegalArgumentException
("argument config error : the index attribute and type attribute not match :index :
" + argument.getIndex() + ", type:" + argument.getType());
 56:                                         }
 57:                                     } else {
 58:                                         // multiple callbacks in the method
 59:                                         for (int j = 0; j < argTypes.length; j
++) {
 60:                                             Class<?> argClazz = argTypes[j];
 61:                                             if (argClazz.getName().equals(argu
ment.getType())) {
 62:                                                 // 将 ArgumentConfig 对象，添加
到 `map` 集合中。
 63:                                                 appendParameters(map, argument
, method.getName() + "." + j); // `${methodName}.${index}`
 64:                                                 if (argument.getIndex() != -1
&& argument.getIndex() != j) { // 多余的判断，因为 `argument.getIndex() == -1` 。
 65:                                                     throw new IllegalArgumentE
xception("argument config error : the index attribute and type attribute not match
:index :" + argument.getIndex() + ", type:" + argument.getType());
 66:                                                 }
 67:                                             }
 68:                                         }
 69:                                     }
 70:                                 }
```

```
                            }
                        }
                    } else if (argument.getIndex() != -1) { // 指定单个参数的位置
                        // 将 ArgumentConfig 对象，添加到 `map` 集合中。
                        appendParameters(map, argument, method.getName() + "." + argument.getIndex()); // `${methodName}.${index}`
                    } else {
                        throw new IllegalArgumentException("argument config must set index or type attribute.eg: <dubbo:argument index='0' .../> or <dubbo:argument type=xxx .../>");
                    }

                }
            }
        } // end of methods for
    }

    // generic、methods、revision
    if (ProtocolUtils.isGeneric(generic)) {
        map.put("generic", generic);
        map.put("methods", Constants.ANY_VALUE);
    } else {
        String revision = Version.getVersion(interfaceClass, version);
        if (revision != null && revision.length() > 0) {
            map.put("revision", revision); // 修订号
        }

        String[] methods = Wrapper.getWrapper(interfaceClass).getMethodNames(); // 【TODO 8003】Wrapper
        if (methods.length == 0) {
            logger.warn("NO method found in service interface " + interfaceClass.getName());
            map.put("methods", Constants.ANY_VALUE);
        } else {
            map.put("methods", StringUtils.join(new HashSet<String>(Arrays.asList(methods)), ","));
        }
    }
    // token , 参见《令牌校验》https://dubbo.gitbooks.io/dubbo-user-book/demos/token-authorization.html
    if (!ConfigUtils.isEmpty(token)) {
        if (ConfigUtils.isDefault(token)) {
            map.put("token", UUID.randomUUID().toString());
        } else {
            map.put("token", token);
        }
    }
```

```
111:     // 协议为 injvm 时，不注册，不通知。
112:     if ("injvm".equals(protocolConfig.getName())) {
113:         protocolConfig.setRegister(false);
114:         map.put("notify", "false");
115:     }
116:     // export service
117:     String contextPath = protocolConfig.getContextpath();
118:     if ((contextPath == null || contextPath.length() == 0) && provider != null
) {
119:         contextPath = provider.getContextpath();
120:     }
121:
122:     // host、port
123:     String host = this.findConfigedHosts(protocolConfig, registryURLs, map);
124:     Integer port = this.findConfigedPorts(protocolConfig, name, map);
125:
126:     // 创建 Dubbo URL 对象
127:     URL url = new URL(name, host, port, (contextPath == null || contextPath.le
ngth() == 0 ? "" : contextPath + "/") + path, map);
128:
129:     // 配置规则，参见《配置规则》https://dubbo.gitbooks.io/dubbo-user-book/demos/
config-rule.html
130:     if (ExtensionLoader.getExtensionLoader(ConfiguratorFactory.class)
131:             .hasExtension(url.getProtocol())) {
132:         url = ExtensionLoader.getExtensionLoader(ConfiguratorFactory.class)
133:                 .getExtension(url.getProtocol()).getConfigurator(url).configur
e(url);
134:     }
135:
136:     // 省略【服务暴露】逻辑
137: }
```

- 第 2 至 6 行：协议名空时，缺省 `"dubbo"` 。
- 第 9 行：创建参数集合 `map` ，用于下面创建 Dubbo URL 的 `parameters` 属性。
- 第 10 至 15 行：将 `side` `dubbo` `timestamp` `timestamp` `pid` 添加到 `map` 中。
- 第 16 至 21 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》
- 第 22 至 83 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。
  - 🙂 代码比较冗长，胖友耐心看注释，建议进行调试每种情况。
- 第 85 至 102 行：将 `generic` `methods` `revision` 到 `map` 中。
  - `revision` ，可能比较难理解，在 「10. Version」 详细解析。

- 第 103 至 110 行：将 `token` 添加到 `map` 中。
  - 《Dubbo 用户指南 —— 令牌验证》
- 第 111 至 115 行：当协议为 `injvm` 时，添加 `notify = false` 到 `map` 中，表示不注册，不通知。
  - 《Dubbo 用户指南 —— 本地调用》
- 第 116 至 120 行：获得 `contextPath` ，基础路径，即java web应用中常说的context path 。
- 第 123 行：调用 `#this.findConfigedHosts(protocolConfig, registryURLs, map)` 方法，获得注册到注册中心的服务提供者 Host 。
  - 《Dubbo 用户指南 —— 主机绑定》
  - 《dubbo注册服务IP解析异常及IP解析源码分析》
  - 指定服务注册地址，参见 dubbo-docker-sample 示例项目。
  - 🙂 代码比较冗长，胖友耐心看注释，建议进行调试每种情况。
- 第 124 行：调用 `#findConfigedHosts(protocolConfig, name, map)` 方法，获得注册到注册中心的服务提供者 Port 。
  - 🙂 代码比较冗长，胖友耐心看注释，建议进行调试每种情况。
- 第 127 行：创建 Dubbo URL 。
- 第 129 至 134 行：配置规则，后续详细解析。
  - 《Dubbo 用户指南 —— 配置规则》
- 第 136 行：**省略**【服务暴露】逻辑。

9. 为什么继承？？？

我们以 ServiceConfig 和 ProviderConfig 来举例子，两者都继承 AbstractServiceConfig。
从属性上，两者有相同的属性，例如 `group` / `version` 。
同时，也存在着一些差异，例如 `ServiceConfig.interfaceName` / `ProviderConfig.host` 。

另外，我们在看看 ServiceConfig 和 MethodConfig ，两者都继承 AbstractMethodConfig。 在 ServiceConfig 中，可以配置下属所有方法的 `retries` 次数，也可以在 MethodConfig 中**自定义** `retries` 次数。

通过继承，获得相同的属性。

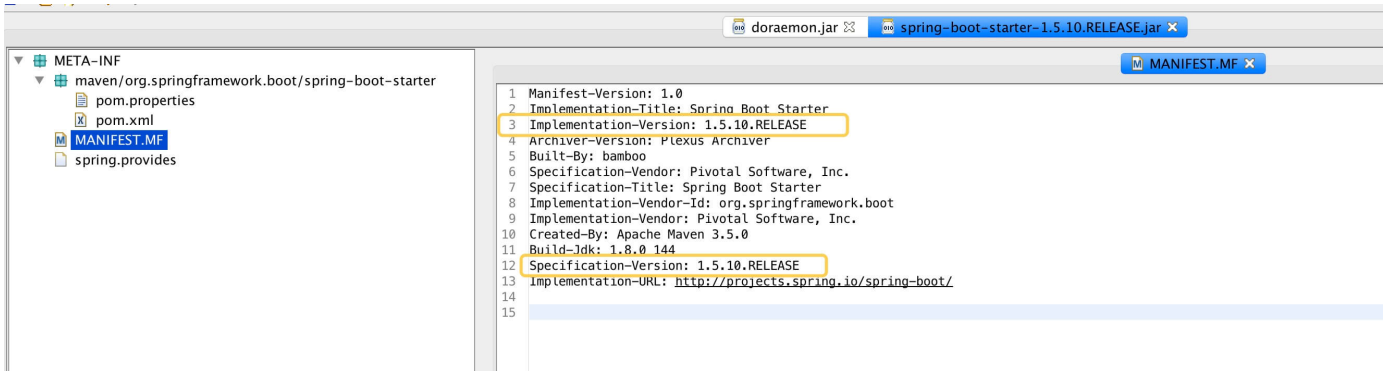
10. Version

`Version#getVersion(cls, defaultVersion)` 方法，获得版本号。代码如下：

```java
public static String getVersion(Class<?> cls, String defaultVersion) {
    try {
        // find version info from MANIFEST.MF first
        String version = cls.getPackage().getImplementationVersion();
        if (version == null || version.length() == 0) {
            version = cls.getPackage().getSpecificationVersion();
        }
        if (version == null || version.length() == 0) {
            // guess version fro jar file name if nothing's found from MANIFEST.MF
            CodeSource codeSource = cls.getProtectionDomain().getCodeSource();
            if (codeSource == null) {
                logger.info("No codeSource for class " + cls.getName() + " when getVersion, use default version " + defaultVersion);
            } else {
                String file = codeSource.getLocation().getFile();
                if (file != null && file.length() > 0 && file.endsWith(".jar")) {
                    file = file.substring(0, file.length() - 4);
                    int i = file.lastIndexOf('/');
                    if (i >= 0) {
                        file = file.substring(i + 1);
                    }
                    i = file.indexOf("-");
                    if (i >= 0) {
                        file = file.substring(i + 1);
                    }
                    while (file.length() > 0 && !Character.isDigit(file.charAt(0))) {
                        i = file.indexOf("-");
                        if (i >= 0) {
                            file = file.substring(i + 1);
                        } else {
                            break;
                        }
                    }
                    version = file;
                }
            }
        }
        // return default version if no version info is found
        return version == null || version.length() == 0 ? defaultVersion : version;
    } catch (Throwable e) {
        // return default version when any exception is thrown
        logger.error("return default version, ignore exception " + e.getMessage(), e);
        return defaultVersion;
```

```
43:     }
44: }
```

- 第 3 至 7 行：从 `MAINFEST.MF` 中获得版本号。以 spring-boot-starter-1.5.10.RELEASE.jar 举例子：



- 第 8 至 36 行：若获取不到，从 jar 包**命名**中**可能**带的版本号作为结果。例如上面的例子，`1.5.10.RELEASE` 。
- 第 38 行：返回版本号。若不存在，返回默认版本号。

666. 彩蛋

比较冗长。
是不是看的一脸蒙圈。
没关系滴，随着对 Dubbo 源码的愈发了解，配置类很多疑惑的地方，会慢慢解开。

裤子都脱了，果敢的继续加油！