# dubbo配置规则详解

# DUBBO配置规则详解

# DUBBO包含很多内容，如果想了解DUBBO第一步就是启动它，从而可以很好的使用它，那么如何更好的使用呢？就需要知道DUBBO的各个配置项，以及它可以通过哪些途径进行配置。个人对配置的理解，就好比时对动物的驯服，如何很好的驯服一头猛兽，那就需要知道它各种习性，从而调整，已达到自己期望的结果。这篇不对DUBBO有哪些配置项可以配置，但是通过这篇文章，你应该能够知道DUBBO可以进行哪些配置。本文会通过分析DUBBO加载配置源码的分析，来使得大家对DUBBO的配置一块有更加深入的了解。从而达到“驯服”DUBBO，以使得它成为你们自己的DUBBO。

DUBBO在配置这一块做的确实很完美，提供很很多参数，以及提供了多种渠道。下面进入正题，看看DUBBO怎么加载配置的。在讲这些之前，先给大家介绍一下在DUBBO源码层面定义了哪些类来存储各个模块的配置项，从而了解DUBBO可以对哪些模块进行配置。

## 哪些东西可以配置

由于大部分项目都会使用Spring，而且DUBBO也提供了通过Spring来进行配置，那么先从这里进行着手。DUBBO加载Spring的集成时在dubbo-config下面的dubbo-config-spring模块下面，其中有一个类DubboNamespaceHandler，它实现了Spring提供的接口NamespaceHandlerSupport。那么Spring怎么发现整个实现类的呢？在该模块的META-INF文件夹下有两个文件： spring.handlers和spring.schemas，这两个文件里面制定了dubbo的namespace的XSD文件的位置以及dubbo的namespace由DubboNamespaceHandler来处理解析。说了这么多废话，只是想说明Spring是怎么解析<dubbo:.../>配置的。

知道了DUBBO和Spring关于配置一块时怎么整合的之后，那么你应该就不会诧异Spring怎么那么聪明，能够解析dubbo的namespace。接下来看看DubboNamespaceHandler类里面有什么东西。

public class DubboNamespaceHandler extends NamespaceHandlerSupport {

static {

Version.checkDuplicate(DubboNamespaceHandler.class);

}

public void init() {

registerBeanDefinitionParser("application", new DubboBeanDefinitionParser(ApplicationConfig.class, true));

registerBeanDefinitionParser("module", new DubboBeanDefinitionParser(ModuleConfig.class, true));

registerBeanDefinitionParser("registry", new DubboBeanDefinitionParser(RegistryConfig.class, true));

registerBeanDefinitionParser("monitor", new DubboBeanDefinitionParser(MonitorConfig.class, true));

registerBeanDefinitionParser("provider", new DubboBeanDefinitionParser(ProviderConfig.class, true));

registerBeanDefinitionParser("consumer", new DubboBeanDefinitionParser(ConsumerConfig.class, true));

registerBeanDefinitionParser("protocol", new DubboBeanDefinitionParser(ProtocolConfig.class, true));

registerBeanDefinitionParser("service", new DubboBeanDefinitionParser(ServiceBean.class, true));

registerBeanDefinitionParser("reference", new DubboBeanDefinitionParser(ReferenceBean.class, false));

registerBeanDefinitionParser("annotation", new DubboBeanDefinitionParser(AnnotationBean.class, true));

}

}

可以看到再init方法里面都是调用一个方法registerBeanDefinitionParser，但是参数略微有些不同。registerBeanDefinitionParser方法的第一个参数是dubbo的namespace下面节点名称，第二个参数时该节点由谁来进行解析。例如：registerBeanDefinitionParser("application", new DubboBeanDefinitionParser(ApplicationConfig.class, true));是解析<[dubbo:application../](http://dubboapplication..)>配置信息的，依此类推通过Spring可以配置<[dubbo:module../](http://dubbomodule..)>,<[dubbo:registry../](http://dubboregistry..)>等等，就不一一列举了。至于每个标签配置的作用，由于不是本篇的内容，所以这里就不做过多的介绍。

通过上面应该清楚知道DUBBO可以配置哪些？这个问题应该不会再困扰你了。下面看看DUBBO是怎么加载这些配置项的。

## 如何读取我们的配置

### 通过Spring的Bean配置读取

在项目中，会配置Spring的XML（虽然DUBBO也支持注解形式，但是个人不是很推崇，因为这样会将DUBBO整合到你的项目源码里面，而我的建议是不要讲DUBBO和你的项目绑的太紧，我的建议是DUBBO的配置和项目隔离，从而方便管理，加入以后项目不使用DUBBO，而使用其他的分布式RPC框架，对项目的调整会比较小），比如经常会通过下面的配置项引用一个远程的服务：

<dubbo:reference id="demoService" interface="com.alibaba.dubbo.demo.DemoService" timeout="2000" check="false"/>

通过上面的内容，应该知道[dubbo:reference](http://dubboreference)将会由new DubboBeanDefinitionParser(ReferenceBean.class, false)来解析（虽然看上去貌似所有配置都是用DubboBeanDefinitionParser来解析，但是有很大的不同）。那看看类DubboBeanDefinitionParser里面做了什么事情。

public class DubboBeanDefinitionParser implements BeanDefinitionParser {

private static final Logger logger = LoggerFactory.getLogger(DubboBeanDefinitionParser.class);

private final Class<?> beanClass;

private final boolean required;

public DubboBeanDefinitionParser(Class<?> beanClass, boolean required) {

this.beanClass = beanClass;

this.required = required;

}

public BeanDefinition parse(Element element, ParserContext parserContext) {

return parse(element, parserContext, beanClass, required);

}@SuppressWarnings("unchecked")private static BeanDefinition parse(Element element, ParserContext parserContext, Class<?> beanClass, boolean required) {....}

}

首先看类的定义，DubboBeanDefinitionParser实现了Spring的BeanDefinitionParser接口，该接口时专门用来解析Bean的定义的（一看类名就应该知道），并且实现了public BeanDefinition parse(Element element, ParserContext parserContext)方法（别看整个方法返回了一个BeanDefinition对象，其实Spring并没有利用整个返回的对象，具体你可以看看Spring的源码，所以要把Bean的定义注入到Spring容器中，就需要手动的往Spring中注入，因为Spring没有给我们来做这件事请。），然后调用了静态方法private static BeanDefinition parse(...)，那么该类主要就是在静态的方法parse上了。由于该方法内容实在是太长了，不便粘贴出全部内容，我只分析主要的部分。在DubboBeanDefinitionParser构造方法参数上有一个Class<?> beanClass参数，它就是指定讲当前标签配置内容转换成对应类的BeanDefinition并且注入到Spring容器中。

private static BeanDefinition parse(Element element, ParserContext parserContext, Class<?> beanClass, boolean required) {

RootBeanDefinition beanDefinition = new RootBeanDefinition();

beanDefinition.setBeanClass(beanClass);

beanDefinition.setLazyInit(false);

String id = element.getAttribute("id");

if ((id == null || id.length() == 0) && required) {

//构造一个Bean的ID

}

....

parserContext.getRegistry().registerBeanDefinition(id, beanDefinition);

....

for (Method setter : beanClass.getMethods()) {

String name = setter.getName();

if (name.length() > 3 && name.startsWith("set")

&& Modifier.isPublic(setter.getModifiers())

&& setter.getParameterTypes().length == 1) {

Class<?> type = setter.getParameterTypes()[0];

String property = StringUtils.camelToSplitName(name.substring(3, 4).toLowerCase() + name.substring(4), "-");

props.add(property);

Method getter = null;

try {

getter = beanClass.getMethod("get" + name.substring(3), new Class<?>[0]);

} catch (NoSuchMethodException e) {

try {

getter = beanClass.getMethod("is" + name.substring(3), new Class<?>[0]);

} catch (NoSuchMethodException e2) {

}

}

if (getter == null

|| ! Modifier.isPublic(getter.getModifiers())

|| ! type.equals(getter.getReturnType())) {

continue;

}

if ("parameters".equals(property)) {

parameters = parseParameters(element.getChildNodes(), beanDefinition);

} else if ("methods".equals(property)) {

parseMethods(id, element.getChildNodes(), beanDefinition, parserContext);

} else if ("arguments".equals(property)) {

parseArguments(id, element.getChildNodes(), beanDefinition, parserContext);

}

.....

beanDefinition.getPropertyValues().addPropertyValue(property, reference);

......

上面代码很显然看到是通过反射的形式获取类的get/set方法然，从而判断该参数是否可以通过Spring注入进去，最后添加到beanDefinition中，并且注入到Spring容器中。上面则是从Spring中加载配置的机制。

### 通过java运行命令-D以及properties中获取配置

上面内容看到DUBBO可以对哪些模块进行配置，并且通过哪些类来存储这些配置信息，例如：ReferenceBean,RegistryConfig,ServiceBean等，如果进去看看这些类的定义会发现他们都继承了AbstractConfig抽象类，该抽象类中定义了protected static void appendProperties(AbstractConfig config)方法，参数时一个实现它自己的子类，接下来看看它做了什么：

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())

&& method.getParameterTypes().length == 1 && isPrimitive(method.getParameterTypes()[0])) {

String property = StringUtils.camelToSplitName(name.substring(3, 4).toLowerCase() + name.substring(4), "-");

String value = null;

if (config.getId() != null && config.getId().length() > 0) {

String pn = prefix + config.getId() + "." + property;

value = System.getProperty(pn);

if(! StringUtils.isBlank(value)) {

logger.info("Use System Property " + pn + " to config dubbo");

}

}

if (value == null || value.length() == 0) {

String pn = prefix + property;

value = System.getProperty(pn);

if(! StringUtils.isBlank(value)) {

logger.info("Use System Property " + pn + " to config dubbo");

}

}

if (value == null || value.length() == 0) {

Method getter;

try {

getter = config.getClass().getMethod("get" + name.substring(3), new Class<?>[0]);

} catch (NoSuchMethodException e) {

try {

getter = config.getClass().getMethod("is" + name.substring(3), new Class<?>[0]);

} catch (NoSuchMethodException e2) {

getter = null;

}

}

if (getter != null) {

if (getter.invoke(config, new Object[0]) == null) {

if (config.getId() != null && config.getId().length() > 0) {

value = ConfigUtils.getProperty(prefix + config.getId() + "." + property);

}

if (value == null || value.length() == 0) {

value = ConfigUtils.getProperty(prefix + property);

}

if (value == null || value.length() == 0) {

String legacyKey = legacyProperties.get(prefix + property);

if (legacyKey != null && legacyKey.length() > 0) {

value = convertLegacyValue(legacyKey, ConfigUtils.getProperty(legacyKey));

}

}

}

}

}

if (value != null && value.length() > 0) {

method.invoke(config, new Object[] {convertPrimitive(method.getParameterTypes()[0], value)});

}

}

} catch (Exception e) {

logger.error(e.getMessage(), e);

}

}

}

上面方法一开始是生产当前配置的前缀String prefix = "dubbo." + getTagName(config.getClass()) + "."，此处可以看到dubbo的所有配置项都是以dubbo.开始的，接下来是通过getTagName方法生成当前配置类的配置名称，进去看看getTagName是怎么为各个配置类生成各自的配置名的。

private static final String[] SUFFIXS = new String[] {"Config", "Bean"};private static String getTagName(Class<?> cls) {

String tag = cls.getSimpleName();

for (String suffix : SUFFIXS) {

if (tag.endsWith(suffix)) {

tag = tag.substring(0, tag.length() - suffix.length());

break;

}

}

tag = tag.toLowerCase();

return tag;

}

分析上面的代码很清楚的知道是怎么生成各自配置类的配置名的，比如：ReferenceBean通过该方法会返回reference。所以关于引用远程Bean的配置都是以dubbo.reference.开头的。生成当前配置类的前缀之后，那么还是按照管理反射出当前类的所有set方法，接下来就是从System.getProperty中取，如果其中没有则会从ConfigUtils.getProperty中取。其中需要注意的是：从System.getProperty中取值并没有判断当前属性是否有值(通过spring注入的)，而从ConfigUtils.getProperty中取会判断是否有值，如果没有才会从其中取，这里可以说明**DUBBO配置项的优先级java -D优先于Spring配置，Spring配置优先于**properties**文件的配置，这也符合一般项目的规则**。不知道大家注意到没，不管是System.getProperty还是ConfigUtils.getProperty都会取两次，一次是String pn = prefix + config.getId() + "." + property,另一次是String pn = prefix + property，这两种的区别就是多了一个config.getId()，可能会好奇config.getId()是什么内容呢？在介绍Spring加载配置的时候，应该知道config对象其实是Spring容器里面的，那么这里的getId，其实就是我们在配置Spring的Bean时候配置的ID。例如:

<dubbo:reference id="demoService" interface="com.alibaba.dubbo.demo.DemoService" timeout="2000" check="false" />

该配置会产生一个ReferenceBean对象，那么此时的config.getId()就是demoService。所以String pn = prefix + config.getId() + "." + property配置的目的是有没有针对当前某个Bean的配置项。比如：配置dubbo消费端的超时时间，一般通过dubbo.reference.timeout，这其实是指定消费端所有Bean的超时时间。有时候我们需要指定某个Bean超时时间可以通过dubbo.reference.{beanId}.timeout来指定，例如上面的可以通过dubbo.reference.demoService.timeout来配置该Bean的超时时间。

到此对DUBBO所有配置途径以及其原理进行了分析和介绍，貌似还没说怎么取发现dubbo有哪些配置。这里教大家怎么去发现DUBBO有哪些配置。上面介绍过，DUBBO将配置项设置到各个配置类的实体中都是通过判断是否有get/set方法，到这里大家应该清楚怎么去发现了吧？就是如果想了解dubbo某个模块的配置，直接到对应的配置类中看它有哪些字段，知道它的字段名，以及确定你要配置的范围，是全局还是某个bean，还有你想通过什么途径来配置，按照dubbo提供的规则很好确定对应的字段怎么来进行配置。那么你可能会说，字段时一个单词知道是怎么配置，那么如果字段时多个单词组合的呢？由于java的编码风格一般时驼峰格式第：第一个单词首字母小写后面单词首字母大写，那么这种情况对应dubbo来说怎么获取该配置项呢？看下面:

//name是get/set方法名 String property = StringUtils.camelToSplitName(name.substring(3, 4).toLowerCase() + name.substring(4), "-");

不难发现对于驼峰，dubbo时通过-字符来分割各个单词。其实到这里基本上可以很好的配置DUBBO了，但是有一个上面的途径如果需要调整参数都需要重启应用，达不到动态调整，于是dubbo为了能够满足在项目不重启的情况下可以动态的调整配置参数。

### 通过DUBBO管理应用添加动态配置

这种方式主要原理就是通过管理器将动态参数发布到注册中心（zookeeper）中，然后各个节点可以获得最新的配置变更，然后进行动态调整。想知道这一块内容，需要取看看类RegistryDirectory的实现，该类它实现了NotifyListener。那么它就可以监听到注册中心的任何变更。再了解RegistryDirectory之前，先看看DUBBO对每个服务在注册中心存储了哪些信息？如果用的时zookeeper，那么你会在每个服务节点目录下面看到一下几个目录consumers,providers,configurators,routers。其中动态配置时放在configurators节点目录下。服务消费端会监听configurators目录变更，如果变更则会调用RegistryDirectory的void notify(List<URL> urls)方法。监听configurators目录变更触发的void notify(List<URL> urls)方法时，urls的是类似override://...，表示将覆盖调用该服务的某些配置(dubbo中对所有的调用配置都是通过URL的形式来展示的)，讲这些URL上面的参数信息替换到调用服务端的URL上面取，并且重新构造该服务的Invoke对象，从而达到更新参数的目的。

### 可以给接口的方法配置参数

整个场景在实际项目中是存在的，比如一个接口存在多个方法，有时候讲参数配置现在再接口层面还是不够的，需要精确到方法级别，例如对接口调用的超时时间设置。dubbo对与方法级别的配置提供了两种途径。

#### Spring Bean的方式配置

<!--lang:xml--> <dubbo:reference id="demoService" interface="com.alibaba.dubbo.demo.DemoService" timeout="2000" check="false" > <dubbo:method name="sayHello" timeout="1000"/> </dubbo:reference>

通过多嵌套一个[dubbo:method](http://dubbomethod)标签来实现。

#### 动态配置

在DUBBO的管理器提供了动态配置的功能，通过添加动态配置，以及指定通知的消费端来指定某个服务的某个方法调整参数。它那里配置的规则时方法名加配置项的名称，例如：key配置成：sayHello.timeout，value=“10000”。那么管理器会在注册中心的对应服务的configurators添加一条override://...?sayHello.timeout=10000节点，指定消费端会监听到这个变更，执行上面内容的流程。

到此关于DUBBO配置已经介绍完毕，可能有一些地方还时没说明清楚，存在疑虑。如果有疑问欢迎提问或者自己取看看相关的DUBBO源码，那一定会了解更加清楚。