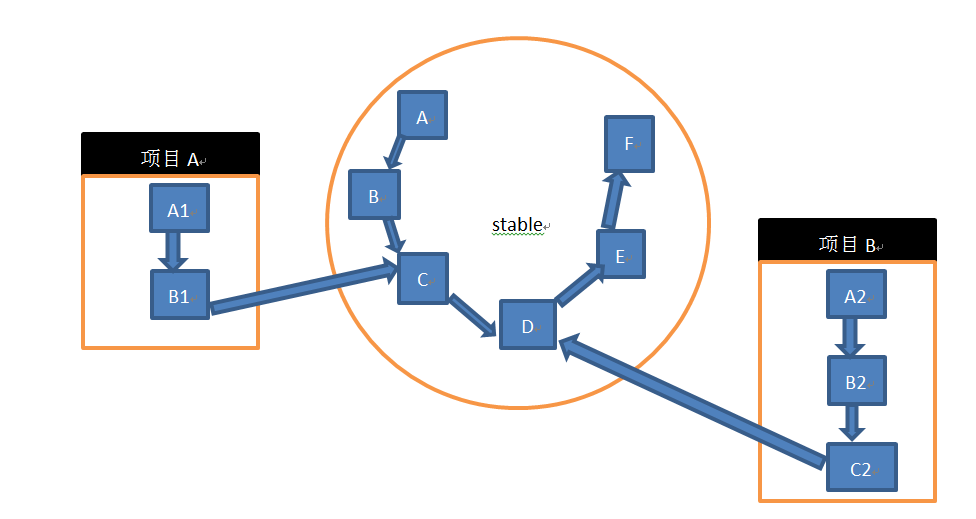
# dubbo服务环境隔离实现

一般选择DUBBO来进行服务管理，都是在分布式应用的前提下，涉及到多个子系统之间的调用，DUBBO所做的事情就是维护各个子系统暴露的接口和自动发现对应接口的远程地址从而实现分布式RPC服务管理。

有了上面前提之后，那么在项目开发和测试过程中涉及到一个问题，就是接口的联调。如果每个子系统自己维护自己系统的联调环境，那么可能会导致别人调用接口的不稳定，因为环境是系统自己人来维护，可能挂了也可能调整接口没通知相关人员，这对开发接口联调测试是一个问题。那么如何做好这件事情呢？下面提出了STABLE环境的概念，一看字面意思就知道是一个稳定的环境，这个环境是和线上保持同步的，并且不是由开发负责维护，而是有专门的运维人负责维护，这样STABLE环境就相对比较稳定，那么调用这个环境的接口也就比较稳定了（你可能会问，怎么调用这个环境的接口？在项目的dubbo.properties里面把注册中心指向STABLE环境即可）。那么问题来了，STABLE不是开发维护，那么会导致如果一个项目涉及多个子系统变更呢？要说明这个问题，我先来个图先：



上面是整个STABLE环境的调用图，不管是哪个项目，将涉及到改动的子系统迁移出来构造一个子环境，然后最后一个节点切入到stable环境中，这样既保证了接口联调的稳定性，也确保了各个项目的开发并行化。关于STABLE环境的介绍不是本篇的内容，所以不做过多的解释。下面谈谈DUBBO怎么隔离各个子环境的服务。

## 直连加不发布服务

DUBBO的配置属性里面对消费端提供了不从注册中心发现服务的机制，直接配置远程接口的地址，这样可以保证消费端连接到制定的环境接口。这样消费端是解决了问题，但是服务提供端呢？如上图的B1它即是消费端也是服务提供端，它提供A1所依赖的接口，那么如果B1将它的服务发布到注册中心里面(这里需要提醒，STABLE环境机制里面所有子环境公用一个注册中心)，那么势必会导致stable环境里面的A会发现B1提供的服务？势必会导致stable环境的不稳定（stable环境的机制是stable环境只能进不能出，就是不能调用外部其他子环境的服务）？所以B1不能发布服务到注册中心，dubbo也提供了相关的配置属性来支持这一点。下面我例举出通过哪些配置可以实现这种方案：

**服务消费端:**

DUBBO在消费端提供了一个url的属性来指定某个服务端的地址

<dubbo:reference interface="com.alibaba.dubbo.demo.HelloWorldService" check="false" id="helloWorldService"/>

默认的方式是从注册中心发现接口为com.alibaba.dubbo.demo.HelloWorldService的服务，但是如果需要直连，可以在dubbo.properties下面配置dubbo.reference.helloWorldService.url=[dubbo://ip:port/com.alibaba.dubbo.demo.HelloWorldService](dubbo://ipport)可以通过配置dubbo.reference.url=[dubbo://ip:port/](dubbo://ipport)来让某个消费者系统的服务都指向制定的服务器地址（关于配置信息可以参考[《DUBBO配置规则详解》](http://my.oschina.net/bieber/blog/378638)）

**服务提供端：**

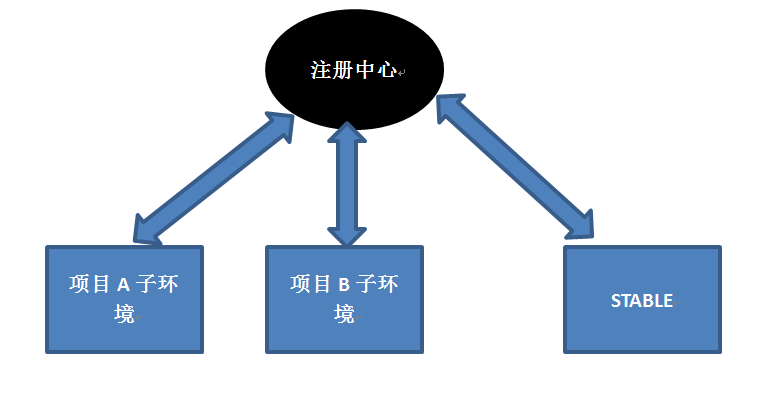
只需要在dubbo.properties里面添加dubbo.registry.register=false即表示当前系统的服务不发布到注册中心。

这种方式服务发布和服务消费就和注册中心没一点关系了，个人感觉这是一个退步，我们用dubbo就是它的服务管理，而这种方案是直接将我们打入了原始社会。这样也会导致如果一个项目设计的子系统很多，那么搭建一个项目的子环境将会比较头疼，因为你要配置那些直连地址。

**注意：**这里为什么一直通过配置在dubbo.properties文件中来达到目的？其实dubbo提供了多种配置的渠道（见[《DUBBO配置规则详解》](http://my.oschina.net/bieber/blog/378638)）。因为是为了达到环境的隔离，最好不用为了切换环境而调整源码，这样容易导致将调整的代码发布到产线，所以排除通过Spring的XML来配置。一般情况下dubbo.properties可以被定义为是存放环境的配置，因为不同的环境注册中心地址不一样，如果将这些地址信息配置在Spring里面，难免会带来失误。所以建议dubbo.properties文件不要放在项目中，而是放在环境的容器里面，通过容器来加载这个文件（比如JBOSS，可以将这个文件放在modules下面），这样对代码会比较稳定。

## 通过服务分组或者版本号来隔离

熟悉DUBBO的童鞋应该知道DUBBO对每个接口都支持分组和版本号，然后服务消费方指定调用哪个分组或者哪个版本号就可以调用对应的接口。那么通过这个来描述一下怎么通过它们来隔离。在谈这些之前还是先上一个各个子系统和注册中心的关系图：



通过给每个子环境分配一个分组来实现各个子环境在一个组里面，从而实现各个环境的隔离。具体操作如下：

**服务消费方**

<dubbo:reference interface="com.alibaba.dubbo.demo.HelloWorldService" check="false" id="helloWorldService"/>

针对上面的接口只能调用指定的分组，可以在dubbo.properties中添加dubbo.reference.helloWorldService.group=test，那么该接口只会从test分组中发现对应接口的服务了。也可以将所有服务都指向某个分组dubbo.reference.group=test。

**服务提供方**

<dubbo:service interface="com.alibaba.dubbo.demo.HelloWorldService" id="helloWorldRemote" ref="helloWorld"/>

针对上面的接口发布到指定的分组，也是在dubbo.properties中添加dubbo.service.helloWorldRemote.group=test，那么该服务就发布到了test分组,同样也可以将当前系统所有服务发布到指定分组dubbo.service.group=test。

而通过版本号也是类似的方案，只是配置的属性不是group而是version，这里就不赘述了。

这个方案看上去很好，不需要再配置直连的地址了，而是通过分组的方案来实现环境的隔离。但是如果你看过dubbo的官方文档，你可能知道group在dubbo的定义是服务接口有多种实现而进行分组的（version也是类似），不是进行环境上面隔离的，所以虽然dubbo提供了这种功能，但是设计的目的不是做这种事情的，那么就不能这么硬拉过来，不然会导致团队开发理解不一直出现问题。另外这种方案会导致注册中心比较混乱，因为注册中心是所以环境公用的，那么会导致一个注册中心中存在多个环境的接口，也不便于维护。

说了这么多，那么有没有一个比较合理的方案来实现环境的隔离呢？据我了解dubbo的原生并没有提供，需要对dubbo进行小小的改造。下面谈谈这个小小的改造怎么个改造法！

## 注册中心分组实现隔离

细心的童鞋可能知道dubbo在配置注册中心的时候有group字段，可以通过dubbo.registry.group=test来实现注册中心的分组，但是这有个问题，如果配置了这个，那么当前系统的服务发现和服务注册都会到这个组里面来进行，不能分别对服务发现和服务注册单独配置，也不能对某个接口进行配置。所以沿着这个想法，我对dubbo进行了小小的改造，在dubbo的服务发布和服务消费添加了注册中心分组的概念。既然要对注册中心进行分组配置，那么就需要了解怎么将分组告诉注册中心，以及分组在注册中心是如何体现的，这里我就以Zookeeper注册中心为例，看看它是怎么实现的。

dubbo中zookeeper的注册中心由ZookeeperRegistry类实现的，看看它的构造函数就你就清楚了：

public ZookeeperRegistry(URL url, ZookeeperTransporter zookeeperTransporter) {

super(url);

if (url.isAnyHost()) {

throw new IllegalStateException("registry address == null");

}

String group = url.getParameter(Constants.GROUP\_KEY, DEFAULT\_ROOT);

if (! group.startsWith(Constants.PATH\_SEPARATOR)) {

group = Constants.PATH\_SEPARATOR + group;

}

this.root = group;

zkClient = zookeeperTransporter.connect(url);

zkClient.addStateListener(new StateListener() {

public void stateChanged(int state) {

if (state == RECONNECTED) {

try {

recover();

} catch (Exception e) {

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

}

}

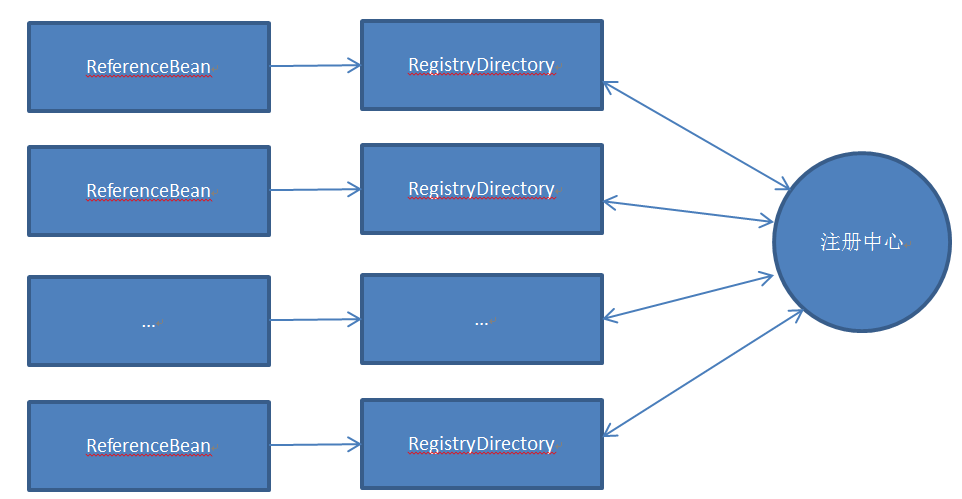
}

});

}

可以看到注册中心接受的是一个URL对象（dubbo内部和外部同学都是通过URL来实现的），并且从其中获取group参数，如果没有则是默认的dubbo。那么你就不难理解为什么dubbo发布到zookeeper的根节点是dubbo了，这个其实是组名。那么不同组服务，将会在zookeeper不同的根节点下面。

在谈这些之前先看看dubbo中发布服务，关联远程服务和注册中心的关系。



上图是服务引用和注册中心关系图，服务发布也是类似，他发布服务的时候会向制定的注册中心发布服务。基于上面我在这两个类中添加了一个registryGroup属性,由于ReferenceBean和ServiceBean都继承了AbstractInterfaceConfig抽象类，那么在这个抽象类中加入字段registryGroup那么服务消费和服务发布里面都可以读取到该字段，添加完该字段之后，就可以通过dubbo.properties文件配置该属性,在[《DUBBO配置规则详解》](http://my.oschina.net/bieber/blog/378638)有讲过怎么配置。我这里列举一下对于属性registryGroup怎么来配置：

服务消费端:

在dubbo.properties文件中添加dubbo.reference.registry-group=test那么当前系统的所有服务应用都会从注册中心的test组中去发现服务，当然也可以通过dubbo.reference.beanId.registry-group=test来指定某个服务从test组中发现服务。

服务提供端：

也是在dubbo.properties文件中添加类似的内容，只是将上面的reference改成service即可。

这里是配置，在服务发布和服务发现都读到这个配置之后，怎么体现到注册中心里的分组中呢？因为这里毕竟不是直接配置注册中心的分组(dubbo.registry.group)，所以需要调整一下dubbo的代码来将这个属性添加到服务发现和服务注册的注册中心中。这里主要调整了三个类，其中一个是常量类中添加了一个常量。总共对dubbo的代码修改不超过10行。   
下面列举一下我的代码调整：

显示添加一个常量,在Constants中添加了public static final String REGISTRY\_GROUP\_KEY = "registry.group";主要是为了避免字符串硬编码。

上面有说过AbstractInterfaceConfig类，在该类中添加了一个字段private String registryGroup;并且生成get/set方法，好让dubbo帮我们注入这个属性（见[《DUBBO配置规则详解》](http://my.oschina.net/bieber/blog/378638)）。

**服务消费端代码调整：**

对ReferenceConfig（ReferenceBean的父类）的createProxy方法进行了调整，这个方法入参的map是ReferenceBean的所有参数K/V对。对该方法下面一段进行了调整:

else { // 通过注册中心配置拼装URL

List<URL> us = loadRegistries(false);

if (us != null && us.size() > 0) {

for (URL u : us) {

URL monitorUrl = loadMonitor(u);

if (monitorUrl != null) {

map.put(Constants.MONITOR\_KEY, URL.encode(monitorUrl.toFullString()));

}

u=u.addParameterAndEncoded(Constants.REFER\_KEY, StringUtils.toQueryString(map));

if(map.containsKey(Constants.REGISTRY\_GROUP\_KEY)){

u=u.addParameter(Constants.GROUP\_KEY,map.get(Constants.REGISTRY\_GROUP\_KEY));

}

urls.add(u);

}

}

if (urls == null || urls.size() == 0) {

throw new IllegalStateException("No such any registry to reference " + interfaceName + " on the consumer " + NetUtils.getLocalHost() + " use dubbo version " + Version.getVersion() + ", please config <dubbo:registry address=\"...\" /> to your spring config.");

}

}

就是判断当前类有没有配置registryGroup，如果配置了添加到注册中心的分组属性中，那么这个服务就会从这个分组的注册中心去发现服务了。

**服务提供方调整：**

这部分是对AbstractInterfaceConfig的方法loadRegistries进行了调整，该方法是加载发布服务的注册中心URL，所以只需要在其返回的URL里面添加group参数即可。具体代码如下：

protected List<URL> loadRegistries(boolean provider) {

checkRegistry();

List<URL> registryList = new ArrayList<URL>();

if (registries != null && registries.size() > 0) {

for (RegistryConfig config : registries) {

String address = config.getAddress();

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

address = Constants.ANYHOST\_VALUE;

}

String sysaddress = System.getProperty("dubbo.registry.address");

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

address = sysaddress;

}

if (address != null && address.length() > 0

&& ! RegistryConfig.NO\_AVAILABLE.equalsIgnoreCase(address)) {

Map<String, String> map = new HashMap<String, String>();

appendParameters(map, application);

appendParameters(map, config);

map.put("path", RegistryService.class.getName());

map.put("dubbo", Version.getVersion());

map.put(Constants.TIMESTAMP\_KEY, String.valueOf(System.currentTimeMillis()));

if (ConfigUtils.getPid() > 0) {

map.put(Constants.PID\_KEY, String.valueOf(ConfigUtils.getPid()));

}

if (! map.containsKey("protocol")) {

if (ExtensionLoader.getExtensionLoader(RegistryFactory.class).hasExtension("remote")) {

map.put("protocol", "remote");

} else {

map.put("protocol", "dubbo");

}

}

List<URL> urls = UrlUtils.parseURLs(address, map);

for (URL url : urls) {

url = url.addParameter(Constants.REGISTRY\_KEY, url.getProtocol());

url = url.setProtocol(Constants.REGISTRY\_PROTOCOL);

if ((provider && url.getParameter(Constants.REGISTER\_KEY, true))

|| (! provider && url.getParameter(Constants.SUBSCRIBE\_KEY, true))) {

if(!StringUtils.isEmpty(this.getRegistryGroup())){

url=url.addParameter(Constants.GROUP\_KEY,this.getRegistryGroup());

}

registryList.add(url);

}

}

}

}

}

return registryList;

}

到此，关于这方案的介绍基本完毕。这种就可以使得每个环境在一个独立的注册中心的分组中，可以很好的维护，并且发布服务不会凌乱，对服务的配置即可以全局设置，也可以对单个服务进行配置。基本上满足了环境隔离的需要。