欧阳思海 2.1k

Dubbo 一篇文章就够了：从入门到实战

zookeeper java 发布于 2019-07-28 • 约 82 分钟

一 为什么需要 dubbo

很多时候，其实我们使用这个技术的时候，可能都是因为项目需要，所以，我们就用了，但是，至于为什么我们需要用到这个技术，可能自身并不是很了解的，但是，其实了解技术的来由及背景知识，对于理解一项技术还是有帮助的，那么，dubbo是怎么被提上日程的呢?

在互联网的发展过程中，在以前，我们只需要一个服务器，将程序全部打包好就可以，但是，随着流量的增大，常规的垂直应用架构已无法应对，所以，架构就发生了演变。

1 单一应用架构

2 应用和数据库单独部署

3 应用和数据库集群部署

4 数据库压力变大，读写分离

5 使用缓存技术加快速度

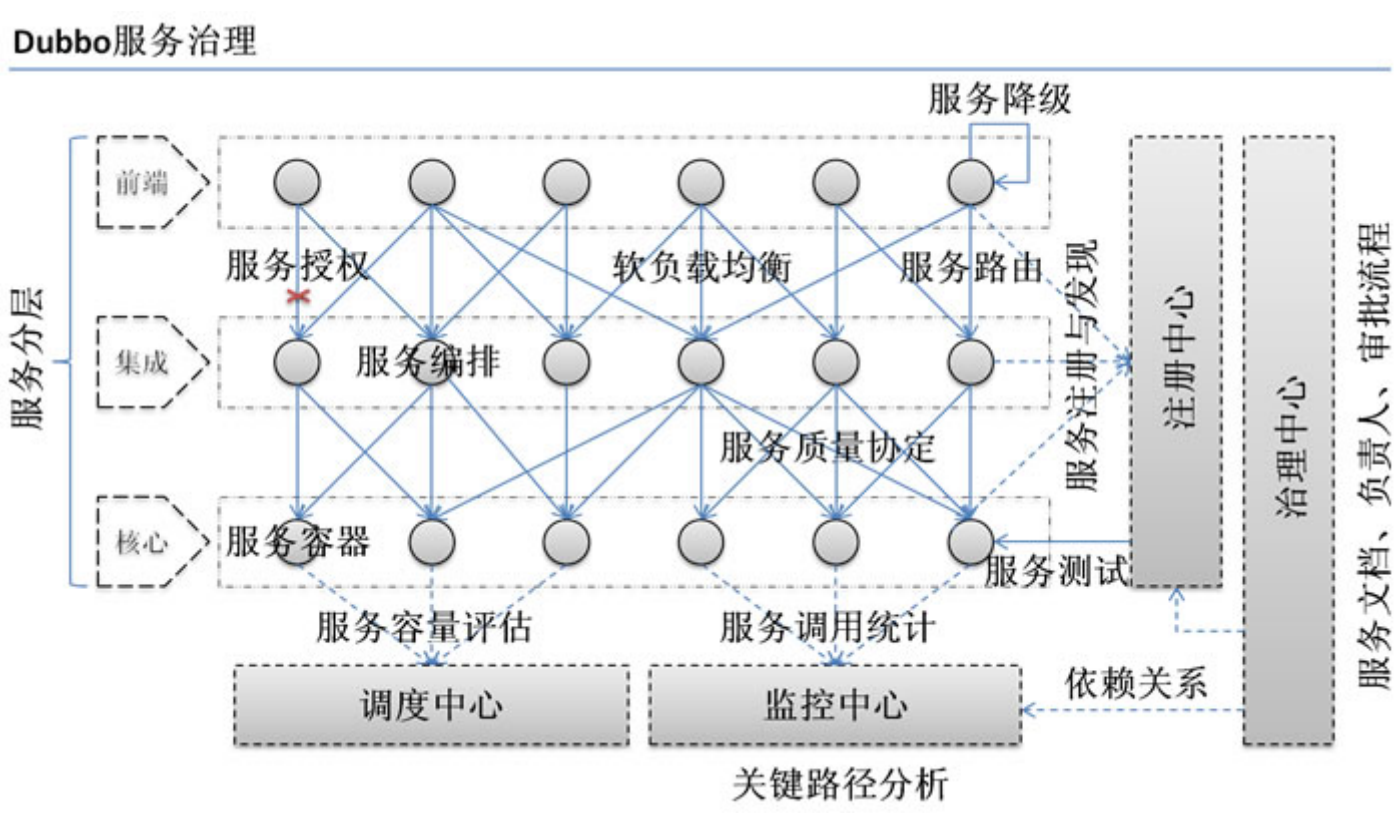
6 数据库分库分表

7 应用分为不同的类型拆分

发展到这个阶段的时候，我们发现，应用与应用之间的关系已经十分的复杂了，就会出现以下几个问题（以下摘录于官网）：

① 当服务越来越多时，服务 URL 配置管理变得非常困难，F5 硬件负载均衡器的单点压力也越来越大。
② 当进一步发展，服务间依赖关系变得错踪复杂，甚至分不清哪个应用要在哪个应用之前启动，架构师都不能完整的描述应用的架构关系。
③ 接着，服务的调用量越来越大，服务的容量问题就暴露出来，这个服务需要多少机器支撑？什么时候该加机器?

为了解决这由于架构的演变所产生的问题几个问题，于是，dubbo 产生了。当然，解决这个问题的技术不止 dubbo 。



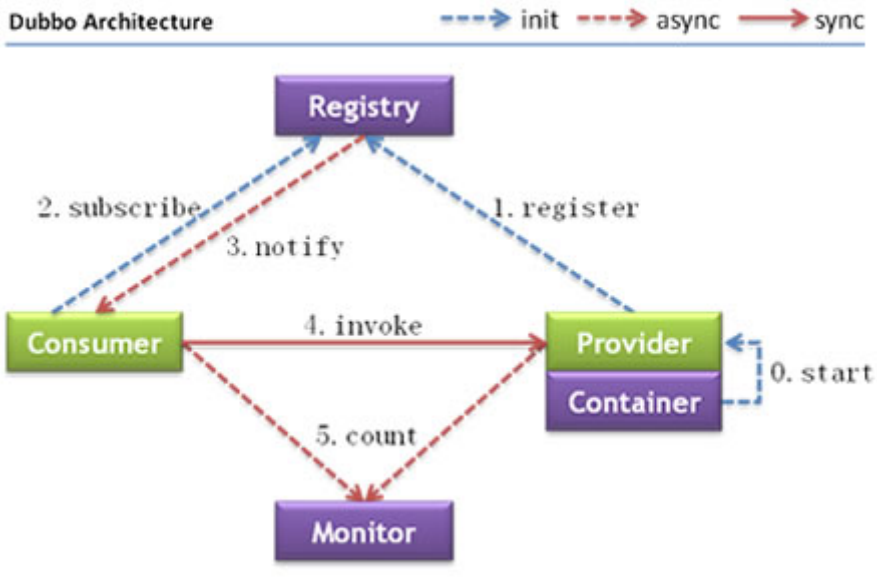
从上面 Dubbo 的服务治理图我们就可以看到，Duboo 很好了解决了上面所出现的一些问题。

所以，当你的系统架构发展到了这种阶段的时候，就需要考虑使用 Dubbo 了。

二 Dubbo 技术架构

我们已经非常清楚的知道为什么在我们的系统中需要 Dubbo 这项技术了，下面，我们接着唠叨唠叨 Dubbo 的架构。

首先，上一张图（摘自官网）。



看到图之后，可能你对上面的几个概念还是一脸懵逼，无从下手，下面，带你看看这几个角色到底是什么意思？

节点角色说明

节点	角色说明
Provider	暴露服务的服务提供方
Consumer	调用远程服务的服务消费方
Registry	服务注册与发现的注册中心
Monitor	统计服务的调用次数和调用时间的监控中心
Container	服务运行容器

看了这几个概念后似乎发现，其实 Dubbo 的架构也是很简单的（其实现细节是复杂的），为什么这么说呢，有没有发现，其实很像**生产者-消费者**模型。只是在这种模型上，加上了**注册中心和监控中心**，用于管理提供方提供的**url**，以及管理整个过程。

那么，整个发布-订阅的过程就非常的简单了。

- 启动容器，加载，**运行服务提供者。**
- 服务提供者在启动时，在注册中心**发布注册**自己提供的**服务。**
- 服务消费者在启动时，在注册中心**订阅**自己所需的**服务。**

如果考虑**失败或变更**的情况，就需要考虑下面的过程。

- 注册中心返回服务提供者地址列表给消费者，如果有变更，注册中心将基于长连接推送变更数据给消费者。
- 服务消费者，从提供者地址列表中，基于软负载均衡算法，选一台提供者进行调用，如果调用失败，再选另一台调用。
- 服务消费者和提供者，在内存中累计调用次数和调用时间，定时每分钟发送一次统计数据到监控中心。

通过这番讲解，我相信 Dubbo 的架构我们也轻车熟路了，那就直接入手，开车吧。

三 Dubbo 开始入门

终于走到这一步了，写到这里停了大概一周的时间，主要原因还是最近项目太忙，赶着交差呢，今天希望能一鼓作气，完完整整的写完 dubbo 的基础篇！

3.1 服务端

首先，我们先把服务端的接口写好，因为其实 dubbo 的作用简单来说就是给消费端提供接口。

■ 接口定义

```
/**
 * xml方式服务提供者接口
 */
public interface ProviderService {

    String SayHello(String word);
}
```

这个接口非常简单，只是包含一个 SayHello 的方法。

接着，定义它的实现类。

```
/**
 * xml方式服务提供者实现类
 */
public class ProviderServiceImpl implements ProviderService{

    public String SayHello(String word) {
        return word;
    }
}
```

这样我们就把我们的接口写好了，那么我们应该怎么将我们的服务暴露出去呢?

■ 导入 maven 依赖

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.ouyangsihai</groupId>
    <artifactId>dubbo-provider</artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>
        <!-- https://mvnrepository.com/artifact/com.alibaba/dubbo -->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>dubbo</artifactId>
            <version>2.6.6</version>
        </dependency>
        <dependency>
            <groupId>org.apache.zookeeper</groupId>
            <artifactId>zookeeper</artifactId>
```

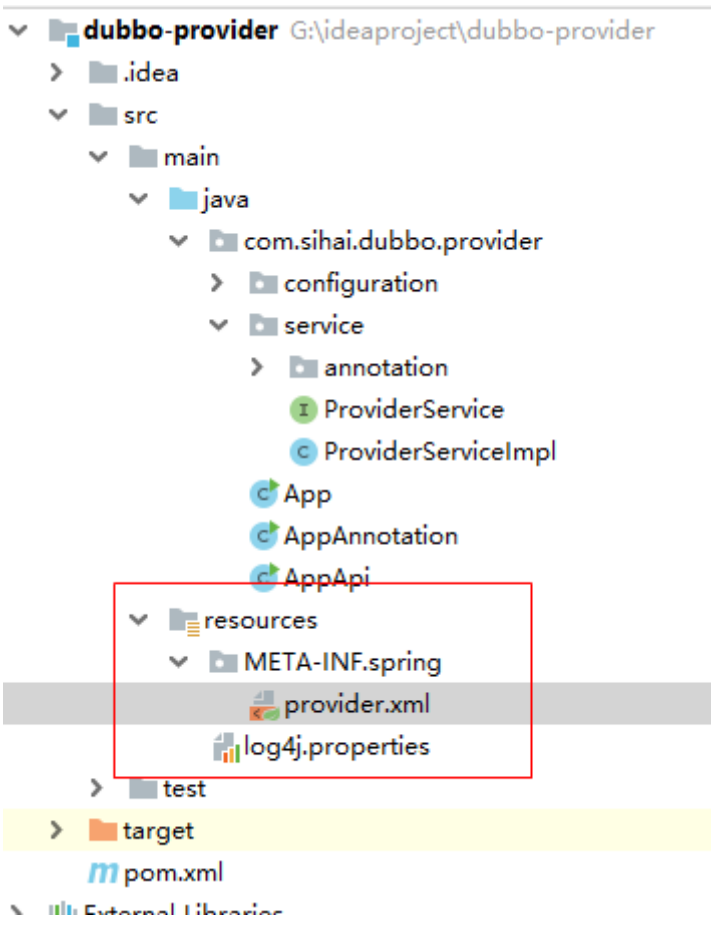
这里使用的 dubbo 的版本是 `2.6.6` ，需要注意的是，如果你只导入 dubbo 的包的时候是**会报错**的，**找不到 netty 和 curator 的依赖**，所以，在这里我们需要把这两个的依赖加上，就不会报错了。

另外，这里我们使用 zookeeper 作为注册中心。

到目前为止，dubbo 需要的环境就已经可以了，下面，我们就把上面刚刚定义的接口暴露出去。

■ 暴露接口（xml 配置方法）

首先，我们在我们项目的 resource 目录下**创建 META-INF.spring 包**，然后再创建 **provider.xml** 文件，名字可以任取哦，如下图。




```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd        http://code.alibabatech.com/schema/dubbo
http://code.alibabatech.com/schema/dubbo/dubbo.xsd">

    <!--当前项目在整个分布式架构里面的唯一名称，计算依赖关系的标签-->
    <dubbo:application name="provider" owner="sihai">
        <dubbo:parameter key="qos.enable" value="true"/>
        <dubbo:parameter key="qos.accept.foreign.ip" value="false"/>
        <dubbo:parameter key="qos.port" value="55555"/>
    </dubbo:application>

    <dubbo:monitor protocol="registry"/>

    <!--dubbo这个服务所要暴露的服务地址所对应的注册中心-->
    <!--<dubbo:registry address="N/A"/>-->
    <dubbo:registry address="N/A" />

    <!--当前服务发布所依赖的协议；webserovice、Thrift、Hessain、http-->
    <dubbo:protocol name="dubbo" port="20880"/>

    <!--服务发布的配置，需要暴露的服务接口-->
    <dubbo:service
```

① 上面的文件其实就是类似 spring 的配置文件，而且，dubbo 底层就是 spring。

② **节点：dubbo:application**

就是整个项目在分布式架构中的唯一名称，可以在 `name` 属性中配置，另外还可以配置 `owner` 字段，表示属于谁。

下面的参数是可以不配置的，这里配置是因为出现了端口的冲突，所以配置。

③ **节点：dubbo:monitor**

监控中心配置， 用于配置连接监控中心相关信息，可以不配置，不是必须的参数。

④ **节点：dubbo:registry**

配置注册中心的信息，比如，这里我们可以配置 zookeeper 作为我们的注册中心。`address` 是注册中心的地址，这里我们配置的是 `N/A` 表示由 dubbo 自动分配地址。或者说是一种直连的方式，不通过注册中心。

⑤ **节点：dubbo:protocol**

服务发布的时候 dubbo 依赖什么协议，可以配置 dubbo、webserovice、Thrift、Hessain、http等协议。

⑥ **节点：dubbo:service**

这个节点就是我们的重点了，当我们服务发布的时候，我们就是通过这个配置将我们的服务发布出去的。`interface` 是接口的包路径，`ref` 是第 ⑦ 点配置的接口的 bean。
⑦ 最后，我们需要像配置 spring 的接口一样，配置接口的 bean。

到这一步，关于服务端的配置就完成了，下面我们通过 `main 方法`将接口发布出去。

■ 发布接口

```
package com.sihai.dubbo.provider;

import com.alibaba.dubbo.config.ApplicationConfig;
import com.alibaba.dubbo.config.ProtocolConfig;
import com.alibaba.dubbo.config.RegistryConfig;
import com.alibaba.dubbo.config.ServiceConfig;
import com.alibaba.dubbo.container.Main;
import com.sihai.dubbo.provider.service.ProviderService;
import com.sihai.dubbo.provider.service.ProviderServiceImpl;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import java.io.IOException;

/**
 * xml方式启动
 *
 */
public class App
{
    public static void main( String[] args ) throws IOException {
        //加载xml配置文件启动
        ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("META-
INF/spring/provider.xml");
        context.start();
        System.in.read(); // 按任意键退出
    }
```

发布接口非常简单，因为 dubbo 底层就是依赖 spring 的，所以，我们只需要通过 `ClassPathXmlApplicationContext` 拿到我们刚刚配置好的 xml ，然后调用 `context.start()` 方法就启动了。

看到下面的截图，就算是启动成功了，接口也就发布出去了。

你以为到这里就结束了了，并不是的，我们拿到 **dubbo 暴露出去的 url**分析分析。

dubbo 暴露的 url

```
dubbo://192.168.234.1:20880/com.sihai.dubbo.provider.service.ProviderService?
anyhost=true&application=provider&bean.name=com.sihai.dubbo.provider.service.ProviderService&bind.ip=192.168.234.1&b
ind.port=20880&dubbo=2.0.2&generic=false&interface=com.sihai.dubbo.provider.service.ProviderService&methods=SayHello
&owner=sihai&pid=8412&qos.accept.foreign.ip=false&qos.enable=true&qos.port=55555&side=provider&timestamp=15620772893
80
```

分析

① 首先，在形式上我们发现，其实这么牛逼的 dubbo 也是用**类似于 http 的协议**发布自己的服务的，只是这里我们用的是 **dubbo 协议。**
② `dubbo://192.168.234.1:20880/com.sihai.dubbo.provider.service.ProviderService`
上面这段链接就是 `?` 之前的链接，构成：**协议://ip:端口/接口。**发现是不是也没有什么神秘的。
③
`anyhost=true&application=provider&bean.name=com.sihai.dubbo.provider.service.ProviderService&bind.ip=192.168.234.1&bind.port=20880&dubbo=2.0.2&generic=false&interface=com.sihai.dubbo.provider.service.ProviderService&methods=SayHello&owner=sihai&pid=8412&qos.accept.foreign.ip=false&qos.enable=true&qos.port=55555&side=provider&timestamp=1562077289380`
`?` 之后的字符串，分析后你发现，这些都是刚刚在 `provider.xml` 中配置的字段，然后通过 `&` 拼接而成的，闻到了 `http` 的香味了吗?

终于，dubbo 服务端入门了。下面我们看看拿到了 url 后，怎么消费呢?

3.2 消费端

上面提到，我们在服务端提供的只是点对点的方式提供服务，并没有使用注册中心，所以，下面的配置也是会有一些不一样的。

■ 消费端环境配置

首先，我们在消费端的 resource 下建立配置文件 `consumer.xml`。


```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd        http://code.alibabatech.com/schema/dubbo
http://code.alibabatech.com/schema/dubbo/dubbo.xsd">

    <!--当前项目在整个分布式架构里面的唯一名称，计算依赖关系的标签-->
    <dubbo:application name="consumer" owner="sihai"/>

    <!--dubbo这个服务所要暴露的服务地址所对应的注册中心-->
    <!--点对点的方式-->
    <dubbo:registry address="N/A" />
    <!--<dubbo:registry address="zookeeper://localhost:2181" check="false"/>-->

    <!--生成一个远程服务的调用代理-->
    <!--点对点方式-->
    <dubbo:reference id="providerService"
                     interface="com.sihai.dubbo.provider.service.ProviderService"
                     url="dubbo://192.168.234.1:20880/com.sihai.dubbo.provider.service.ProviderService"/>

    <!--<dubbo:reference id="providerService"
                     interface="com.sihai.dubbo.provider.service.ProviderService"/>-->

</beans>
```

分析

① 发现这里的 `dubbo:application` 和 `dubbo:registry` 是一致的。

② `dubbo:reference` ： 我们这里采用**点对点**的方式，所以，需要配置在服务端暴露的 url 。

■ maven 依赖

和服务端一样

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.ouyangsihai</groupId>
    <artifactId>dubbo-consumer</artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencies>
        <dependency>
            <groupId>com.ouyangsihai</groupId>
            <artifactId>dubbo-provider</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>
        <!-- https://mvnrepository.com/artifact/com.alibaba/dubbo -->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>dubbo</artifactId>
```

■ 调用服务

```java
package com.sihai.dubbo.consumer;

import com.alibaba.dubbo.config.ApplicationConfig;
import com.alibaba.dubbo.config.ReferenceConfig;
import com.alibaba.dubbo.config.RegistryConfig;
import com.sihai.dubbo.provider.service.ProviderService;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import java.io.IOException;

/**
 * xml的方式调用
 *
 */
public class App
{
    public static void main( String[] args ) throws IOException {

        ClassPathXmlApplicationContext context=new ClassPathXmlApplicationContext("consumer.xml");
        context.start();
        ProviderService providerService = (ProviderService) context.getBean("providerService");
        String str = providerService.SayHello("hello");
        System.out.println(str);
        System.in.read();

    }
```

这里和服务端的发布如出一辙。

如此，我们就成功调用接口了。

四 加入 zookeeper 作为注册中心

在前面的案例中，我们没有使用任何的注册中心，而是用一种直连的方式进行的。但是，实际上很多时候，我们都是使用 dubbo + zookeeper 的方式，使用 zookeeper 作为注册中心，这里，我们就介绍一下 zookeeper 作为注册中心的使用方法。

这里，我们在前面的入门实例中进行改造。

4.1 服务端

在服务端中，我们只需要修改 provider.xml 即可。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd        http://code.alibabatech.com/schema/dubbo
http://code.alibabatech.com/schema/dubbo/dubbo.xsd">

    <!--当前项目在整个分布式架构里面的唯一名称，计算依赖关系的标签-->
    <dubbo:application name="provider" owner="sihai">
        <dubbo:parameter key="qos.enable" value="true"/>
        <dubbo:parameter key="qos.accept.foreign.ip" value="false"/>
        <dubbo:parameter key="qos.port" value="55555"/>
    </dubbo:application>

    <dubbo:monitor protocol="registry"/>

    <!--dubbo这个服务所要暴露的服务地址所对应的注册中心-->
    <!--<dubbo:registry address="N/A"/>-->
    <dubbo:registry address="zookeeper://localhost:2181" check="false"/>

    <!--当前服务发布所依赖的协议；webserovice、Thrift、Hessain、http-->
    <dubbo:protocol name="dubbo" port="20880"/>

    <!--服务发布的配置，需要暴露的服务接口-->
    <dubbo:service
```

重点关注这句话

```
<dubbo:registry address="zookeeper://localhost:2181" />
```

在 address 中，使用我们的 zookeeper 的地址。

如果是 **zookeeper 集群**的话，使用下面的方式。

```
<dubbo:registry protocol="zookeeper" address="192.168.11.129:2181,192.168.11.137:2181,192.168.11.138:2181"/>
```

服务端的配置就好了，其他的跟 **入门案例** 一样。

4.2 消费端

跟服务端一样，在消费端，我们也只需要修改 `consumer.xml` 即可。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd        http://code.alibabatech.com/schema/dubbo
http://code.alibabatech.com/schema/dubbo/dubbo.xsd">

    <!--当前项目在整个分布式架构里面的唯一名称，计算依赖关系的标签-->
    <dubbo:application name="consumer" owner="sihai"/>

    <!--dubbo这个服务所要暴露的服务地址所对应的注册中心-->
    <!--点对点的方式-->
    <!--<dubbo:registry address="N/A" />-->
    <dubbo:registry address="zookeeper://localhost:2181" check="false"/>

    <!--生成一个远程服务的调用代理-->
    <!--点对点方式-->
    <!--<dubbo:reference id="providerService"
                     interface="com.sihai.dubbo.provider.service.ProviderService"
                     url="dubbo://192.168.234.1:20880/com.sihai.dubbo.provider.service.ProviderService"/>-->

    <dubbo:reference id="providerService"
                     interface="com.sihai.dubbo.provider.service.ProviderService"/>

</beans>
```

① 注册中心配置跟服务端一样。

```
<dubbo:registry address="zookeeper://localhost:2181"/>
```

② dubbo:reference

由于我们这里使用 zookeeper 作为注册中心，所以，跟点对点的方式是不一样的，这里不再需要 dubbo 服务端提供的 url 了，只需要直接引用服务端提供的接口即可。

```
<dubbo:reference id="providerService"
                     interface="com.sihai.dubbo.provider.service.ProviderService"/>
```

好了，消费端也配置好了，这样就可以使用**修改的入门案例**，重新启动运行了。

```
2019-07-03 00:02:48,528 INFO [com.alibaba.dubbo.registry.zookeeper.ZookeeperRegistry] -  [DUBBO] Sul
2019-07-03 00:02:48,546 INFO [com.alibaba.dubbo.registry.zookeeper.ZookeeperRegistry] -  [DUBBO] No
2019-07-03 00:02:48,660 INFO [com.alibaba.dubbo.remoting.transport.AbstractClient] -  [DUBBO] Succe
2019-07-03 00:02:48,660 INFO [com.alibaba.dubbo.remoting.transport.AbstractClient] -  [DUBBO] Start
2019-07-03 00:02:48,734 INFO [com.alibaba.dubbo.config.AbstractConfig] -  [DUBBO] Refer dubbo servi
hello
```

同样成功了。

这时候的区别在于，**将 dubbo 发布的 url 注册到了 zookeeper，消费端从 zookeeper 消费，zookeeper 相当于一个中介，给消费者提供服务。**

你以为这就完了？不，好戏才刚刚开始呢。

五 多种配置方式

在**入门实例**的时候，我们使用的是 **xml 配置**的方式，对 dubbo 的环境进行了配置，但是，官方还提供了其他的配置方式，这里我们也一一分解。

5.1 API配置方式

这种方式其实官方是**不太推荐**的，**官方推荐使用 xml 配置的方式**，但是，在有的时候测试的时候，还是可以用的到的，另外，为了保证完整性，这些内容还是有必要讲讲的。

首先还是回到服务端工程。

■ 服务端

这里我们使用 **api 的方式**配置，所以，`provider.xml` 这个配置文件就暂时不需要了，我们只需要在上面的 `AppApi` 这个类中的 `main` 方法中用 api配置及启动即可。

```
package com.sihai.dubbo.provider;

import com.alibaba.dubbo.config.ApplicationConfig;
import com.alibaba.dubbo.config.ProtocolConfig;
import com.alibaba.dubbo.config.RegistryConfig;
import com.alibaba.dubbo.config.ServiceConfig;
import com.sihai.dubbo.provider.service.ProviderService;
import com.sihai.dubbo.provider.service.ProviderServiceImpl;

import java.io.IOException;

/**
 * Api方式启动
 * api的方式调用不需要其他的配置，只需要下面的代码即可。
 * 但是需要注意，官方建议：
 * Api方式用于测试用例使用，推荐xml的方式
 */
public class AppApi
{
    public static void main( String[] args ) throws IOException {

        // 服务实现
        ProviderService providerService = new ProviderServiceImpl();

        // 当前应用配置
        ApplicationConfig application = new ApplicationConfig();
```

分析

看到上面的代码是不是云里雾里，不要慌，我们通过对照 xml 的方式分析一下。

> registry 的 xml 方式

```
<dubbo:registry protocol="zookeeper" address="localhost:2181"/>
```

> API 的方式

```
RegistryConfig registry = new RegistryConfig();
registry.setAddress("zookeeper://localhost:2181");
```

`dubbo:registry`节点对应`RegistryConfig` ，xml 的属性对应 API 方式用 `set` 方法就可以了。对比之下，你就会发现，如果 API 的方式不熟悉，可以对照 xml 配置方式就可以。

其他 API

```
org.apache.dubbo.config.ServiceConfig
org.apache.dubbo.config.ReferenceConfig
org.apache.dubbo.config.ProtocolConfig
org.apache.dubbo.config.RegistryConfig
org.apache.dubbo.config.MonitorConfig
org.apache.dubbo.config.ApplicationConfig
org.apache.dubbo.config.ModuleConfig
org.apache.dubbo.config.ProviderConfig
org.apache.dubbo.config.ConsumerConfig
org.apache.dubbo.config.MethodConfig
org.apache.dubbo.config.ArgumentConfig
```

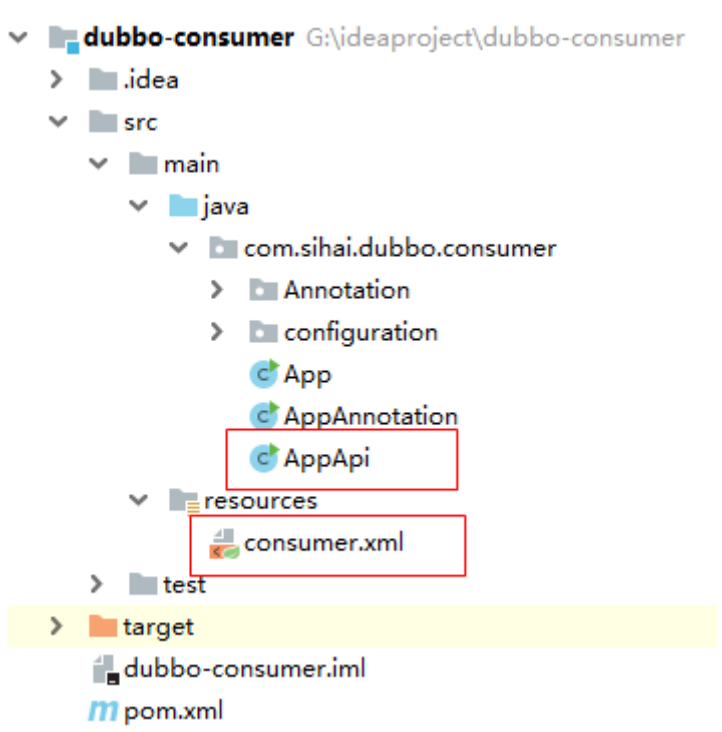
更详细的可以查看官方文档：

http://dubbo.apache.org/zh-cn...

我们再看看我配置的消费端的 Api 方式。

■ 消费端

同样，我们不需要 consumer.xml 配置文件了，只需要在 `main` 方法中启动即可。




```
package com.sihai.dubbo.consumer;

import com.alibaba.dubbo.config.ApplicationConfig;
import com.alibaba.dubbo.config.ReferenceConfig;
import com.alibaba.dubbo.config.RegistryConfig;
import com.sihai.dubbo.provider.service.ProviderService;

/**
 * api的方式调用
 * api的方式调用不需要其他的配置，只需要下面的代码即可。
 * 但是需要注意，官方建议：
 * Api方式用于测试用例使用，推荐xml的方式
 */
public class AppApi {

    public static void main(String[] args) {
        // 当前应用配置
        ApplicationConfig application = new ApplicationConfig();
        application.setName("consumer");
        application.setOwner("sihai");

        // 连接注册中心配置
        RegistryConfig registry = new RegistryConfig();
        registry.setAddress("zookeeper://localhost:2181");

        // 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
```

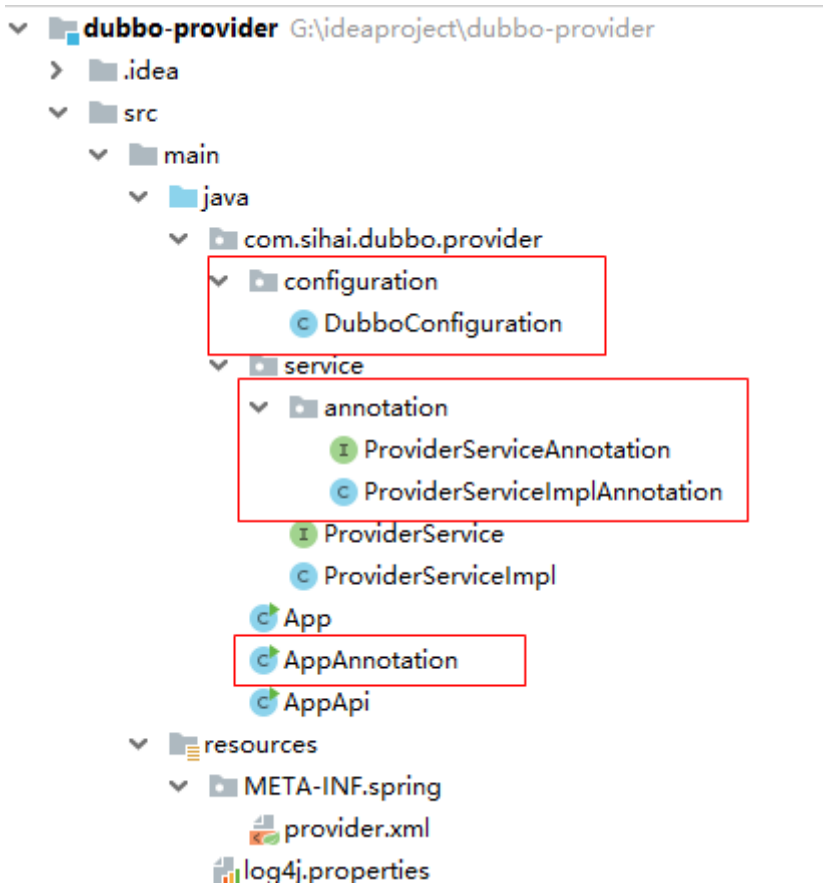
这部分的 API 配置的方式就到这了，注意：**官方推荐 xml 的配置方法。**

5.2 注解配置方式

注解配置方式还是需要了解一下的，现在微服务都倾向于这种方式，这也是以后发展的趋势，0 配置应该是这几年的趋势。

那么如何对 dubbo 使用注解的方式呢？我们先看服务端。

■ 服务端



第一步：定义接口及实现类，在上面的截图中的 annotation 包下

```
package com.sihai.dubbo.provider.service.annotation;

/**
 * 注解方式接口
 */
public interface ProviderServiceAnnotation {
    String SayHelloAnnotation(String word);
}
```

```
package com.sihai.dubbo.provider.service.annotation;

import com.alibaba.dubbo.config.annotation.Service;

/**
 * 注解方式实现类
 */
@Service(timeout = 5000)
public class ProviderServiceImplAnnotation implements ProviderServiceAnnotation{

    public String SayHelloAnnotation(String word) {
        return word;
    }
}
```

@Service

`@Service` 用来配置 Dubbo 的服务提供方。

第二步：组装服务提供方。通过 Spring 中 `Java Config` 的技术（`@Configuration`）和 annotation 扫描（`@EnableDubbo`）来发现、组装、并向外提供 Dubbo 的服务。

```java
package com.sihai.dubbo.provider.configuration;

import com.alibaba.dubbo.config.ApplicationConfig;
import com.alibaba.dubbo.config.ProtocolConfig;
import com.alibaba.dubbo.config.ProviderConfig;
import com.alibaba.dubbo.config.RegistryConfig;
import com.alibaba.dubbo.config.spring.context.annotation.EnableDubbo;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

/**
 * 注解方式配置
 */
@Configuration
@EnableDubbo(scanBasePackages = "com.sihai.dubbo.provider.service.annotation")
public class DubboConfiguration {

    @Bean // #1 服务提供者信息配置
    public ProviderConfig providerConfig() {
        ProviderConfig providerConfig = new ProviderConfig();
        providerConfig.setTimeout(1000);
        return providerConfig;
    }

    @Bean // #2 分布式应用信息配置
    public ApplicationConfig applicationConfig() {
```

分析

- 通过 @EnableDubbo 指定在`com.sihai.dubbo.provider.service.annotation` 下扫描所有标注有 `@Service` 的类
- 通过 `@Configuration` 将 `DubboConfiguration` 中所有的 `@Bean` 通过 `Java Config` 的方式组装出来并注入给 Dubbo 服务，也就是标注有 `@Service` 的类。这其中就包括了：
    - ProviderConfig：服务提供方配置
    - ApplicationConfig：应用配置
    - RegistryConfig：注册中心配置
    - ProtocolConfig：协议配置

看起来很复杂，其实。。。

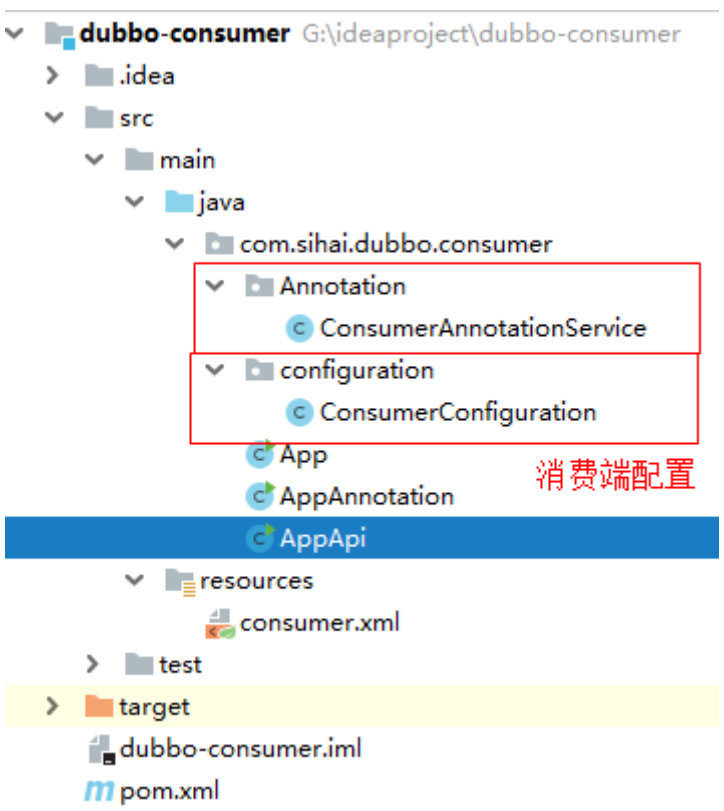
第三步：启动服务

```
package com.sihai.dubbo.provider;

import com.alibaba.dubbo.config.spring.context.annotation.DubboComponentScan;
import com.sihai.dubbo.provider.configuration.DubboConfiguration;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import sun.applet.Main;

import java.io.IOException;

/**
 * 注解启动方式
 */
public class AppAnnotation {

    public static void main(String[] args) throws IOException {
        AnnotationConfigApplicationContext context = new
AnnotationConfigApplicationContext(DubboConfiguration.class);
        context.start();
        System.in.read();
    }
}
```

发现输出下面信息就表示 success 了。

■ 消费端

同样我们下看看消费端的工程，有一个感性认识。



第一步：引用服务

```
package com.sihai.dubbo.consumer.Annotation;

import com.alibaba.dubbo.config.annotation.Reference;
import com.sihai.dubbo.provider.service.annotation.ProviderServiceAnnotation;
import org.springframework.stereotype.Component;

/**
 * 注解方式的service
 */
@Component("annotatedConsumer")
public class ConsumerAnnotationService {

    @Reference
    private ProviderServiceAnnotation providerServiceAnnotation;

    public String doSayHello(String name) {
        return providerServiceAnnotation.SayHelloAnnotation(name);
    }
}
```

在 `ConsumerAnnotationService` 类中，通过 `@Reference` 引用服务端提供的类，然后通过方法调用这个类的方式，给消费端提供接口。

注意：如果这里找不到 `ProviderServiceAnnotation` 类，请在服务端先把服务端工程用 `Maven intall` 一下，然后将服务端的依赖放到消费端的 `pom` 中。如下：

```
<dependency>
          <groupId>com.ouyangsihai</groupId>
            <artifactId>dubbo-provider</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
```

第二步：组装服务消费者

这一步和服务端是类似的，这里就不在重复了。

```
package com.sihai.dubbo.consumer.configuration;

import com.alibaba.dubbo.config.ApplicationConfig;
import com.alibaba.dubbo.config.ConsumerConfig;
import com.alibaba.dubbo.config.RegistryConfig;
import com.alibaba.dubbo.config.spring.context.annotation.EnableDubbo;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

import java.util.HashMap;
import java.util.Map;

/**
 * 注解配置类
 */
@Configuration
@EnableDubbo(scanBasePackages = "com.sihai.dubbo.consumer.Annotation")
@ComponentScan(value = {"com.sihai.dubbo.consumer.Annotation"})
public class ConsumerConfiguration {
    @Bean // 应用配置
    public ApplicationConfig applicationConfig() {
        ApplicationConfig applicationConfig = new ApplicationConfig();
        applicationConfig.setName("dubbo-annotation-consumer");
        Map<String, String> stringStringMap = new HashMap<String, String>();
        stringStringMap.put("qos.enable","true");
```

第三步：发起远程调用

在 `main` 方法中通过启动一个 `Spring Context`，从其中查找到组装好的 Dubbo 的服务消费者，并发起一次远程调用。

```
package com.sihai.dubbo.consumer;

import com.sihai.dubbo.consumer.Annotation.ConsumerAnnotationService;
import com.sihai.dubbo.consumer.configuration.ConsumerConfiguration;
import com.sihai.dubbo.provider.service.ProviderService;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import java.io.IOException;

/**
 * 注解方式启动
 *
 */
public class AppAnnotation
{
    public static void main( String[] args ) throws IOException {

        AnnotationConfigApplicationContext context = new
AnnotationConfigApplicationContext(ConsumerConfiguration.class);
        context.start(); // 启动
        ConsumerAnnotationService consumerAnnotationService = context.getBean(ConsumerAnnotationService.class);
        String hello = consumerAnnotationService.doSayHello("annotation"); // 调用方法
        System.out.println("result: " + hello); // 输出结果

    }
```

结果

六 常用场景

在下面的讲解中，都会是以 `xml` 配置的方式来讲解的，这也是 dubbo 官方比较推荐的方式。以下的操作都是在服务端的 `xml` 配置文件和消费端的配置文件来讲解的。

6.1 启动时检查

Dubbo 缺省会在启动时检查依赖的服务是否可用，不可用时会抛出异常，阻止 Spring 初始化完成，以便上线时，能及早发现问题，默认 `check="true"。

但是，有的时候，我们并不是都需要启动时就检查的，比如测试的时候，我们是需要更快速的启动，所以，这种场景的时候，我们是需要关闭这个功能的。

下面，我们看看如何使用这个功能。

在服务端注册的时候（客户端注册时同样适用）；

```
<dubbo:registry protocol="zookeeper" address="localhost:2181,localhost:2182,localhost:2183" check="false"/>
```

在客户端引用服务端服务的时候；

```
<dubbo:reference check="false" id="providerService"
                     interface="com.sihai.dubbo.provider.service.ProviderService"/>
```

就是这么简单，就是这么强！

6.2 集群容错

dubbo 也是支持集群容错的，同时也有很多可选的方案，其中，默认的方案是 `failover`，也就是重试机制。

首先，我们先把所有的容错机制都整理一遍，然后再看看使用。

集群模式	说明	使用方法
Failover Cluster	失败自动切换，当出现失败，重试其它服务器。通常用于读操作，但重试会带来更长延迟。可通过 retries="2" 来设置重试次数(不含第一次)。	cluster="xxx" xxx：集群模式名称 ， 例如cluster="failover"
Failfast Cluster	快速失败，只发起一次调用，失败立即报错。通常用于非幂等性的写操作，比如新增记录。	
Failsafe Cluster	失败安全，出现异常时，直接忽略。	
Failback Cluster	失败自动恢复，后台记录失败请求，定时重发。通常用于消息通知操作。	
Forking Cluster	并行调用多个服务器，只要一个成功即返回。通常用于实时性要求较高的读操作，但需要浪费更多服务资源。可通过 forks="2" 来设置最大并行数。	
Broadcast Cluster	广播调用所有提供者，逐个调用，任意一台报错则报错。通常用于通知所有提供者更新缓存或日志等本地资源信息。	

使用实例

在发布服务或者引用服务的时候设置

```
<!--服务发布的配置，需要暴露的服务接口-->
    <dubbo:service cluster="failover" retries="2"
            interface="com.sihai.dubbo.provider.service.ProviderService"
            ref="providerService"/>
```

```
<dubbo:reference cluster="failover" retries="2" check="false" id="providerService"
                    interface="com.sihai.dubbo.provider.service.ProviderService"/>
```

6.3 负载均衡

负载均衡想必是一个再熟悉不过的概念了，所以，dubbo 支持也是再正常不过了，这里也总结一下 dubbo 支持的负载均衡的一些方案及使用方法。

负载均衡模式	说明	使用方法
Random LoadBalance	随机 按权重设置随机概率	<dubbo:service loadbalance="xxx"/> xxx：负载均衡方法
RoundRobin LoadBalance	轮询 按公约后的权重设置轮询比率。	
LeastActive LoadBalance	最少活跃调用数 相同活跃数的随机，活跃数指调用前后计数差。	
ConsistentHash LoadBalance	一致性 Hash 相同参数的请求总是发到同一提供者。 当某一台提供者挂时，原本发往该提供者的请求，基于虚拟节点，平摊到其它提供者，不会引起剧烈变动。	

6.4 直连提供者

在开发及测试环境下，经常需要绕过注册中心，只测试指定服务提供者，所以，这种情况下，我们只需要直接连接服务端的地即可，其实，这种方法在前面的讲解已经使用到了，第一种讲解的方式就是这种方式，因为这种方式简单。

使用

```
<dubbo:reference id="providerService"
                     interface="com.sihai.dubbo.provider.service.ProviderService"
                     url="dubbo://192.168.234.1:20880/com.sihai.dubbo.provider.service.ProviderService"/>
```

说明：可以看到，只要在消费端在 `dubbo:reference` 节点使用 `url` 给出服务端的方法即可。

6.5 只订阅

只订阅就是只能够订阅服务端的服务，而不能够注册。

引用官方的使用场景如下：

> 为方便开发测试，经常会在线下共用一个所有服务可用的注册中心，这时，如果一个正在开发中的服务提供者注册，可能会影响消费者不能正常运行。
> 可以让服务提供者开发方，只订阅服务(开发的服务可能依赖其它服务)，而不注册正在开发的服务，通过直连测试正在开发的服务。

```
<dubbo:registry register="false" protocol="zookeeper" address="localhost:2181,localhost:2182,localhost:2183"
check="false"/>
```

① 使用只订阅方式

当在服务提供端使用 `register="false"` 的时候，我们使用下面的方式获取服务端的服务；

```
<dubbo:reference cluster="failover" retries="2" check="false" id="providerService"
                     interface="com.sihai.dubbo.provider.service.ProviderService"/>
```

启动信息

发现，这时候并不是向注册中心 zookeeper 注册，而只是做了发布服务和启动netty。

② 不使用只订阅方式

```
<dubbo:registry protocol="zookeeper" address="localhost:2181,localhost:2182,localhost:2183" check="false"/>
```

启动信息

可以发现，这里就向注册中心 zookeeper 注册了。

6.6 只注册

只注册正好跟前面的只订阅相反，这个时候可以向注册中心注册，但是，消费端却不能够读到服务。

应用场景

> 如果有两个镜像环境，两个注册中心，有一个服务只在其中一个注册中心有部署，另一个注册中心还没来得及部署，而两个注册中心的其它应用都需要依赖此服务。这个时候，可以让服务提供者方只注册服务到另一注册中心，而不从另一注册中心订阅服务。

使用说明

```
<dubbo:registry subscribe="false" address="localhost:2181"></dubbo:registry>
```

在服务端的 dubbo:registry 节点下使用 subscribe="false" 来声明这个服务是只注册的服务。

这个时候消费端调用的时候是不能调用的。

```
Exception in thread "main" com.alibaba.dubbo.rpc.RpcException: No provider available from registry localhost:2181 for service com.sihai.du
    at com.alibaba.dubbo.registry.integration.RegistryDirectory.doList(RegistryDirectory.java:588)
    at com.alibaba.dubbo.rpc.cluster.directory.AbstractDirectory.list(AbstractDirectory.java:75)
    at com.alibaba.dubbo.rpc.cluster.support.AbstractClusterInvoker.list(AbstractClusterInvoker.java:277)
    at com.alibaba.dubbo.rpc.cluster.support.AbstractClusterInvoker.invoke(AbstractClusterInvoker.java:238)
    at com.alibaba.dubbo.rpc.cluster.support.wrapper.MockClusterInvoker.invoke(MockClusterInvoker.java:75)
    at com.alibaba.dubbo.rpc.proxy.InvokerInvocationHandler.invoke(InvokerInvocationHandler.java:52)
    at com.alibaba.dubbo.common.bytecode.proxy0.SayHello(proxy0.java)
    at com.sihai.dubbo.consumer.App.main(App.java:22)
```

6.7 多协议机制

在前面我们使用的协议都是 dubbo 协议，但是 dubbo 除了支持这种协议外还支持其他的协议，比如，rmi、hessian等，另外，而且还可以用多种协议同时暴露一种服务。

使用方法

① 一种接口使用一种协议

先声明多种协议

```
<!--当前服务发布所依赖的协议；webserovice、Thrift、Hessain、http-->
    <dubbo:protocol name="dubbo" port="20880"/>
    <dubbo:protocol name="rmi" port="1099" />
```

然后在发布接口的时候使用具体协议

```
<!--服务发布的配置，需要暴露的服务接口-->
    <dubbo:service cluster="failover" retries="2"
            interface="com.sihai.dubbo.provider.service.ProviderService"
            ref="providerService"/>
    <dubbo:service cluster="failover" retries="2"
                   interface="com.sihai.dubbo.provider.service.ProviderService"
                   ref="providerService" protocol="rmi"/>
```

在输出日志中，就可以找到rmi发布的接口。

```
rmi://192.168.234.1:1099/com.sihai.dubbo.provider.service.ProviderService?
anyhost=true&application=provider&bean.name=com.sihai.dubbo.provider.service.ProviderService&cluster=failover&dubbo=
2.0.2&generic=false&interface=com.sihai.dubbo.provider.service.ProviderService&methods=SayHello&owner=sihai&pid=796&
retries=2&side=provider&timestamp=1564281053185, dubbo version: 2.6.6, current host: 192.168.234.1
```

② 一种接口使用多种协议

声明协议和上面的方式一样，在发布接口的时候有一点不一样。

```
<dubbo:service cluster="failover" retries="2"
                   interface="com.sihai.dubbo.provider.service.ProviderService"
                   ref="providerService" protocol="rmi,dubbo"/>
```

说明：protocol属性，可以用,隔开，使用多种协议。

6.8 多注册中心

Dubbo 支持同一服务向多注册中心同时注册，或者不同服务分别注册到不同的注册中心上去，甚至可以同时引用注册在不同注册中心上的同名服务。

■ 服务端多注册中心发布服务

一个服务可以在不同的注册中心注册，当一个注册中心出现问题时，可以用其他的注册中心。

注册

```
<!--多注册中心-->
    <dubbo:registry protocol="zookeeper" id="reg1" timeout="10000" address="localhost:2181"/>
    <dubbo:registry protocol="zookeeper" id="reg2" timeout="10000" address="localhost:2182"/>
    <dubbo:registry protocol="zookeeper" id="reg3" timeout="10000" address="localhost:2183"/>
```

发布服务

```
<!--服务发布的配置，需要暴露的服务接口-->
    <dubbo:service cluster="failover" retries="2"
            interface="com.sihai.dubbo.provider.service.ProviderService"
            ref="providerService" registry="reg1"/>
    <dubbo:service cluster="failover" retries="2"
                   interface="com.sihai.dubbo.provider.service.ProviderService"
                   ref="providerService" protocol="rmi" registry="reg2"/>
```

说明：使用registry="reg2"指定该接口使用的注册中心，同时也可以使用多个，用，隔开，例如，registry="reg1,,reg2"。

■ 消费端多注册中心引用服务

首先，先向不同注册中心注册;

```
<!--多注册中心-->
    <dubbo:registry protocol="zookeeper" id="reg1" timeout="10000" address="localhost:2181"/>
    <dubbo:registry protocol="zookeeper" id="reg2" timeout="10000" address="localhost:2182"/>
    <dubbo:registry protocol="zookeeper" id="reg3" timeout="10000" address="localhost:2183"/>
```

其次，不同的消费端服务引用使用不同的注册中心;

```
!--不同的服务使用不同的注册中心-->
    <dubbo:reference cluster="failover" retries="2" check="false" id="providerService"
                     interface="com.sihai.dubbo.provider.service.ProviderService" registry="reg1"/>
    <dubbo:reference cluster="failover" retries="2" check="false" id="providerService2"
                     interface="com.sihai.dubbo.provider.service.ProviderService" registry="reg2"/>
```

说明：上面分别使用注册中心1和注册中心2。

6.9 多版本

不同的服务是有版本不同的，版本可以更新并且升级，同时，不同的版本之间是不可以调用的。

```
<!--服务发布的配置，需要暴露的服务接口-->
    <dubbo:service cluster="failover" retries="2"
            interface="com.sihai.dubbo.provider.service.ProviderService"
            ref="providerService" registry="reg1" version="1.0.0"/>
    <dubbo:service cluster="failover" retries="2"
                   interface="com.sihai.dubbo.provider.service.ProviderService"
                   ref="providerService" protocol="rmi" registry="reg2" version="1.0.0"/>
```

加入了版本控制。

6.10 日志管理

dubbo 也可以将日志信息记录或者保存到文件中的。

① 使用accesslog输出到log4j

```
<dubbo:protocol accesslog="true" name="dubbo" port="20880"/>
    <dubbo:protocol accesslog="true" name="rmi" port="1099" />
```

② 输出到文件

```
<dubbo:protocol accesslog="http://localhost/log.txt" name="dubbo" port="20880"/>
    <dubbo:protocol accesslog="http://localhost/log2.txt" name="rmi" port="1099" />
```

七 总结

这篇文章就到这里了，主要讲了一下几个内容

1、为什么需要dubbo

2、dubbo架构简析

3、dubbo入门

4、zookeeper注册中心加入dubbo

5、dubbo多种配置方式（xml、api、注解）

6、常用场景介绍

下一篇文章，将讲讲源码分析。

> 文章有不当之处，欢迎指正，如果喜欢微信阅读，你也可以关注我的**微信公众号**：好好学java，获取优质学习资源。

欧阳思海

2.1k

关注作者

8 条评论

得票 •时间

撰写评论 ...

提交评论

小小智： 您好 我是不是先安装dubbo以及zookeeper?

回复 • 2019-08-29

yaosir： yes，嗯哼

回复 • 2019-11-06

yaosir： webservice 啦啦啦

回复 • 2019-11-06

fengzhi： 文章很好，95分啊，有几个地方有小瑕疵。6.9多版本的DEMO代码没有体现多版本；6.5第1条客户端应该是URL才能获取服务端的服务，你写的是正常配置无法获取到的。因为只是订阅没有提供服务。

• 回复 • 2019-11-15

小白： 小白不太明白，能否写下正确的配置代码，感谢

• 回复 • 2019-11-28

撸小鱼： 很好的文章，学习了，感谢

• 回复 • 2月14日

shuangyueliao： 有没有代码链接

• 回复 • 3月1日

王廷荣： 写的挺基础，应该是看过官网文档

• 回复 • 3月29日

推荐阅读

新手flex布局入门篇，看这篇文章就够了

CSS介绍盒子模型什么是flexboxflex-directionflex-wrapflex-flowjustify-contentalign-itemsalign-contentorderflex-growflex-shrinkfle...

黎跃春 • 阅读 7

学习HTML5 Canvas这一篇文章就够了

<canvas>是HTML5新增的，一个可以使用脚本(通常为JavaScript)在其中绘制图像的HTML元素。它可以用来制作照片集或者制作简...

linvic • 阅读 668

Sinon 入门,看这篇文章就够了

Author:bugallWechat:bugallFEmail:769088641@qq.com项目地址:https:github.combugallnod...当我们在开发前端项目的时候,很多...

李腾飞 • 阅读 10

spring boot的maven配置依赖

我们通过引用spring-boot-starter-parent，添加spring-boot-starter-web可以实现web项目的功能，当然不使用spring-boot-start-w...

帅帅的波 • 阅读 104

如何优雅的选择字体(font-family)

大家都知道，在不同操作系统、不同游览器里面默认显示的字体是不一样的，并且相同字体在不同操作系统里面渲染的效果也不尽...

koreyhan • 阅读 69

React-Redux 入门教程

Actionindex.jsxReducerindex.jsxStorestore.jsxCounter.jsxStorestore.js具体代码实现请gitclonehttps:github.comjeromehan...

jerome • 阅读 22

理解vuex -- vue的状态管理模式

备注：本文的示例等代码将会采用es6的语法。先引用vuex官网的话：状态管理模式、集中式存储管理一听就很高大上，蛮吓人的...

weish1995 • 阅读 1k

短网址(short URL)系统的原理及其实现

此文不再维护更新，可以查看原文做一个短链接生成器，可以将一个长链接缩短成一个短链接。发车前，和大家说一下如果不想重...

小猿大圣 • 阅读 113