

# 使用 Dubbo 搭建一个简单的分布式系统

## 一、前言

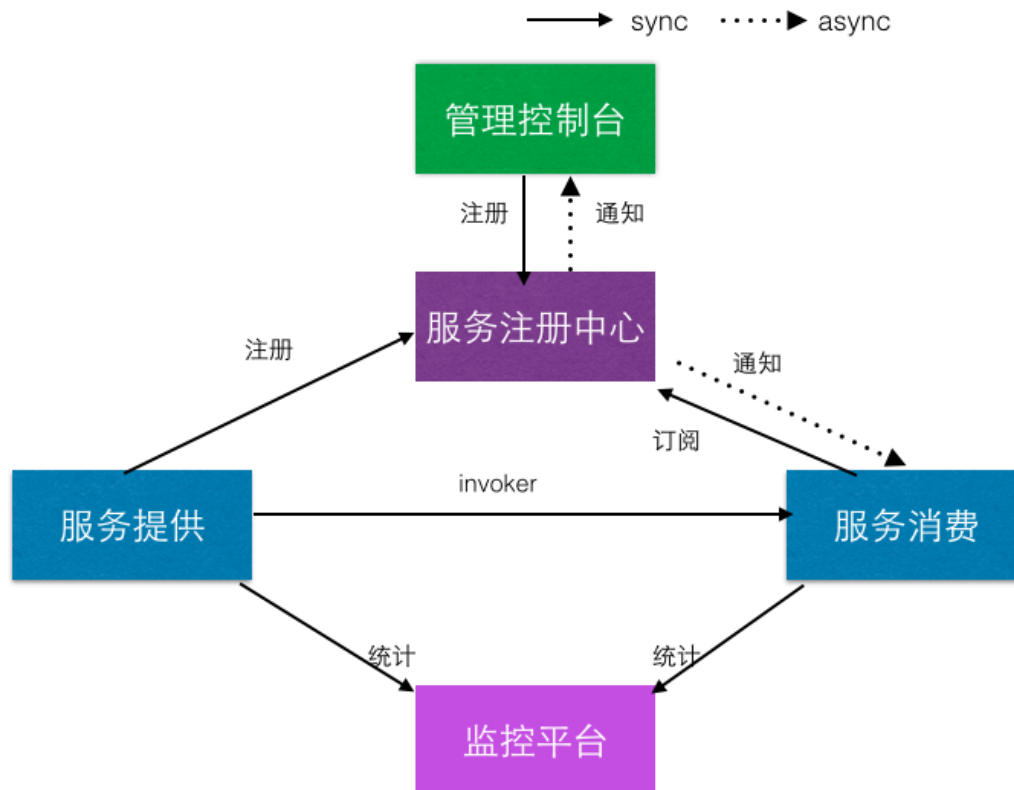
随着阿里巴巴开源的高性能分布式 RPC 框架 Dubbo 正式进入 Apache 孵化器，Dubbo 又火了一把。本场 Chat 作为 Dubbo 系列开端，先教大家使用 Dubbo 搭建一个简单的分布式系统，因为要研究一个东西的原理，必须先能把环境搭建起来，并且会使用它。

在这个系统里面会包含服务提供者，服务消费者，服务注册中心（本 Chat 使用 ZooKeeper），管理控制台（Dubbo-Admin），监控平台（Dubbo-Monitor），麻雀虽小，却五脏俱全。

通过本 Chat 你将能学到（文章内有 Demo 源码）：

- 五大组件的关系。
- 如何基于 Spring 配置搭建一个简单的分布式系统。
- 如何基于 Dubbo API 搭建一个简单的分布式系统。
- 何为服务端异步调用，如何使用异步调用，使用异步调用好处是什么。
- 何为泛化调用，如何使用泛化调用，什么时候使用泛化调用。

## 二、五大组件关系



- 服务提供方在启动时候会注册自己提供的服务到服务注册中心。
- 服务消费方在启动时候会去服务注册中心订阅自己需要的服务，然后服务注册中心异步把消费方需要的服务接口的提供者的地址列表返回给服务消费方，服务消费方根据路由规则和设置的负载均衡算法选择一个服务提供者 ip 进行调用。
- 监控平台主要用来统计服务的调用次数和调用耗时，服务消费者和提供者，在内存中累计调用次数和调用耗时，并定时每分钟发送一次统计数据到监控中心，监控中心则使用数据绘制图表来显示，监控平台不是分布式系统必须的，但是这些数据有助于系统运维和调优。服务提供者和消费者可以直接配置监控平台的地址，也可以通过服务注册中心来获取。
- 管理控制台主要提供路由规则，动态配置，服务降级，访问控制，权重调整，负载均衡，等管理功能。管理控制台直接与服务注册中心打交道，从服务注册中心获取所有注册的服务和消费方法；并且可以通过管理台界面设置服务消费端的路由规则，动态配置等信息并注册到服务管理台，这些信息会被通知到服务消费端。管理控制台也不是分布式系统必备的组件，但是有了他我们可以对服务进行很好的治理和监控

### 三、服务注册中心的搭建

本文我们讲解 Apache ZooKeeper 的搭建。

- 首先你需要到 <http://zookeeper.apache.org/releases.html> 下载一个 zk 的包，本文作者使用的是 zookeeper-3.4.11 这个版本，如下图：

```
-rw-r--r--@ 1 zhuizhumengxiang staff 36668066 3 18 09:18 zookeeper-3.4.11.tar.gz
```

解压该包后，如下图：

```
→ /Users/zhuizhumengxiang/workspace/mytool/dubbo/zookeeper-3.4.11 git:(master) X ls -al
total 3184
drwxr-xr-x  3 zhuizhumengxiang staff    102  2 28 13:00
drwxr-xr-x@ 24 zhuizhumengxiang staff    816  2 28 13:00 .
drwxr-xr-x  13 zhuizhumengxiang staff    442  3 18 09:18 ..
-rw-r--r--@  1 zhuizhumengxiang staff   11938 11  2 02:47 LICENSE.txt
-rw-r--r--@  1 zhuizhumengxiang staff   3132 11  2 02:47 NOTICE.txt
-rw-r--r--@  1 zhuizhumengxiang staff   1585 11  2 02:47 README.md
-rw-r--r--@  1 zhuizhumengxiang staff   1770 11  2 02:47 README_packaging.txt
drwxr-xr-x@ 12 zhuizhumengxiang staff    408  3  7 12:32 bin
-rw-r--r--@  1 zhuizhumengxiang staff   87943 11  2 02:47 build.xml
drwxr-xr-x@  6 zhuizhumengxiang staff    204  3 18 09:28 conf
drwxr-xr-x@ 10 zhuizhumengxiang staff    340 11  2 02:47 contrib
drwxr-xr-x  2 zhuizhumengxiang staff     68  2 28 13:00 data
drwxr-xr-x@ 22 zhuizhumengxiang staff    748 11  2 02:54 dist-maven
drwxr-xr-x@ 49 zhuizhumengxiang staff   1666 11  2 02:52 docs
-rw-r--r--@  1 zhuizhumengxiang staff   8197 11  2 02:47 ivy.xml
-rw-r--r--@  1 zhuizhumengxiang staff   1709 11  2 02:47 ivysettings.xml
drwxr-xr-x@ 13 zhuizhumengxiang staff    442 11  2 02:52 lib
drwxr-xr-x@  5 zhuizhumengxiang staff    170 11  2 02:47 recipes
drwxr-xr-x@ 14 zhuizhumengxiang staff    476 11  2 02:52 src
-rw-r--r--@  1 zhuizhumengxiang staff  1478279 11  2 02:49 zookeeper-3.4.11.jar
-rw-r--r--@  1 zhuizhumengxiang staff    195 11  2 02:52 zookeeper-3.4.11.jar.asc
-rw-r--r--@  1 zhuizhumengxiang staff     33 11  2 02:49 zookeeper-3.4.11.jar.md5
-rw-r--r--@  1 zhuizhumengxiang staff     41 11  2 02:49 zookeeper-3.4.11.jar.shal
-rw-r--r--@  1 zhuizhumengxiang staff   4466  2 28 13:00 zookeeper.out
```

- 然后修改 zookeeper-3.4.11/conf 文件夹里面的 zoo.cfg 文件。
  - 设置配置项 dataDir 为一个存在的以 data 结尾的目录；
  - 设置 zk 的监听端口 clientPort=2181；
  - 设置 zk 心跳检查间隔 tickTime = 2000；
  - 设置 Follower 服务器启动时候从 Leader 同步完毕数据能忍受多少个心跳时间间隔数 initLimit=5。
  - 设置运行过程中 Leader 同步数据到 Follower 后，Follower 回复信息到 Leader 的超时时间 syncLimit=2，如果 Leader 超过 syncLimit 个 tickTime 的时间长度，还没有收到 Follower 响应，那么就认为这个 Follower 已经不在线了：

```
tickTime = 2000
dataDir = /Users/zhuizhumengxiang/workspace/mytool/dubbo/data
clientPort = 2181
initLimit = 5
syncLimit = 2
~
~
```

- 最后在 zookeeper-3.4.11/bin 下运行 sh zkServer.sh start-foreground 就会启动 zk，会有下面输出：

```

➔ /Users/zhuizhumengxiang/workspace/mytool/dubbo/zookeeper-3.4.11/bin git:(master) ✕ sh zkServer.sh start-foreground
ZooKeeper JMX enabled by default
Using config: /Users/zhuizhumengxiang/workspace/mytool/dubbo/zookeeper-3.4.11/bin/./conf/zoo.cfg
2018-03-08 15:25:04,287 [myid:] - INFO [main:QuorumPeerConfig@136] - Reading configuration from: /Users/zhuizhumengxiang/workspace/mytool/dubbo/zookeeper-3.4.11/bin/./conf/zoo.cfg
2018-03-08 15:25:04,301 [myid:] - INFO [main:DatadirCleanupManager@78] - autopurge.snapRetainCount set to 3
2018-03-08 15:25:04,301 [myid:] - INFO [main:DatadirCleanupManager@79] - autopurge.purgeInterval set to 0
2018-03-08 15:25:04,301 [myid:] - INFO [main:DatadirCleanupManager@101] - Purge task is not scheduled.
2018-03-08 15:25:04,302 [myid:] - WARN [main:QuorumPeerMain@116] - Either no config or no quorum defined in config, running in standalone mode
2018-03-08 15:25:04,339 [myid:] - INFO [main:QuorumPeerConfig@136] - Reading configuration from: /Users/zhuizhumengxiang/workspace/mytool/dubbo/zookeeper-3.4.11/bin/./conf/zoo.cfg
2018-03-08 15:25:04,340 [myid:] - INFO [main:ZooKeeperServerMain@98] - Starting server
2018-03-08 15:25:04,447 [myid:] - INFO [main:Environment@100] - Server environment:zookeeper.version=3.4.11-37e277162d567b55a07d1755f0b31c32e93c01a0, built on 11/01/2017 18:06 GMT
2018-03-08 15:25:04,447 [myid:] - INFO [main:Environment@100] - Server environment:host.name=30.8.60.75
2018-03-08 15:25:04,447 [myid:] - INFO [main:Environment@100] - Server environment:java.version=1.8.0_101
2018-03-08 15:25:04,448 [myid:] - INFO [main:Environment@100] - Server environment:java.vendor=Oracle Corporation
2018-03-08 15:25:04,448 [myid:] - INFO [main:Environment@100] - Server environment:java.home=/Library/Java/JavaVirtualMachines/jdk1.8.0_101.jdk/Contents/Home/jre
2018-03-08 15:25:04,449 [myid:] - INFO [main:Environment@100] - Server environment:java.class.path=/Users/zhuizhumengxiang/workspace/mytool/dubbo/zookeeper-3.4.11/bin/./build/classes:/Users/zhuizhumengxiang/workspace/mytool/dubbo/zookeeper-3.4.11/bin/./build/lib/*:/Users/zhuizhumengxiang/workspace/mytool/dubbo/zookeeper-3.4.11/bin/./lib/slf4j-log4j12-1.6.1.jar:/Users/zhuizhumengxiang/workspace/mytool/dubbo/zookeeper-3.4.11/bin/./lib/slf4j-api-1.6.1.jar:/Users/zhuizhumengxiang/workspace/mytool/dubbo/zookeeper-3.4.11/bin/./lib/netty-3.10.5.Final.jar:/Users/zhuizhumengxiang/workspace/mytool/dubbo/zookeeper-3.4.11/bin/./lib/log4j-1.2.16.jar:/Users/zhuizhumengxiang/workspace/mytool/dubbo/zookeeper-3.4.11/bin/./lib/jline-0.9.94.jar:/Users/zhuizhumengxiang/workspace/mytool/dubbo/zookeeper-3.4.11/bin/./lib/audience-annotations-0.5.0.jar:/Users/zhuizhumengxiang/workspace/mytool/dubbo/zookeeper-3.4.11/bin/./zookeeper-3.4.11.jar:/Users/zhuizhumengxiang/workspace/mytool/dubbo/zookeeper-3.4.11/bin/./src/java/lib/*:/Users/zhuizhumengxiang/workspace/mytool/dubbo/zookeeper-3.4.11/bin/./conf:
2018-03-08 15:25:04,449 [myid:] - INFO [main:Environment@100] - Server environment:java.library.path=/Users/zhuizhumengxiang/Library/Java/Extensions:/Library/Java/Extensions:/Network/Library/Java/Extensions:/System/Library/Java/Extensions:/usr/lib/java:
2018-03-08 15:25:04,450 [myid:] - INFO [main:Environment@100] - Server environment:java.io.tmpdir=/var/folders/28/t0qddfd92v1c0hndbphd7jjw0000gn/T/
2018-03-08 15:25:04,450 [myid:] - INFO [main:Environment@100] - Server environment:java.compiler=dNA
2018-03-08 15:25:04,452 [myid:] - INFO [main:Environment@100] - Server environment:os.name=Mac OS X
2018-03-08 15:25:04,452 [myid:] - INFO [main:Environment@100] - Server environment:os.arch=x86_64
2018-03-08 15:25:04,453 [myid:] - INFO [main:Environment@100] - Server environment:os.version=10.10.5
2018-03-08 15:25:04,453 [myid:] - INFO [main:Environment@100] - Server environment:user.name=zhuizhumengxiang
2018-03-08 15:25:04,453 [myid:] - INFO [main:Environment@100] - Server environment:user.home=/Users/zhuizhumengxiang
2018-03-08 15:25:04,453 [myid:] - INFO [main:Environment@100] - Server environment:user.dir=/Users/zhuizhumengxiang/workspace/mytool/dubbo/zookeeper-3.4.11/bin
2018-03-08 15:25:04,472 [myid:] - INFO [main:ZooKeeperServer@825] - tickTime set to 2000
2018-03-08 15:25:04,472 [myid:] - INFO [main:ZooKeeperServer@834] - minSessionTimeout set to -1
2018-03-08 15:25:04,473 [myid:] - INFO [main:ZooKeeperServer@843] - maxSessionTimeout set to -1
2018-03-08 15:25:04,502 [myid:] - INFO [main:ServerCnxnFactory@117] - Using org.apache.zookeeper.server.NIOServerCnxnFactory as server connection factory
2018-03-08 15:25:04,520 [myid:] - INFO [main:NIOServerCnxnFactory@89] - binding to port 0.0.0.0/0.0.0.0:2181

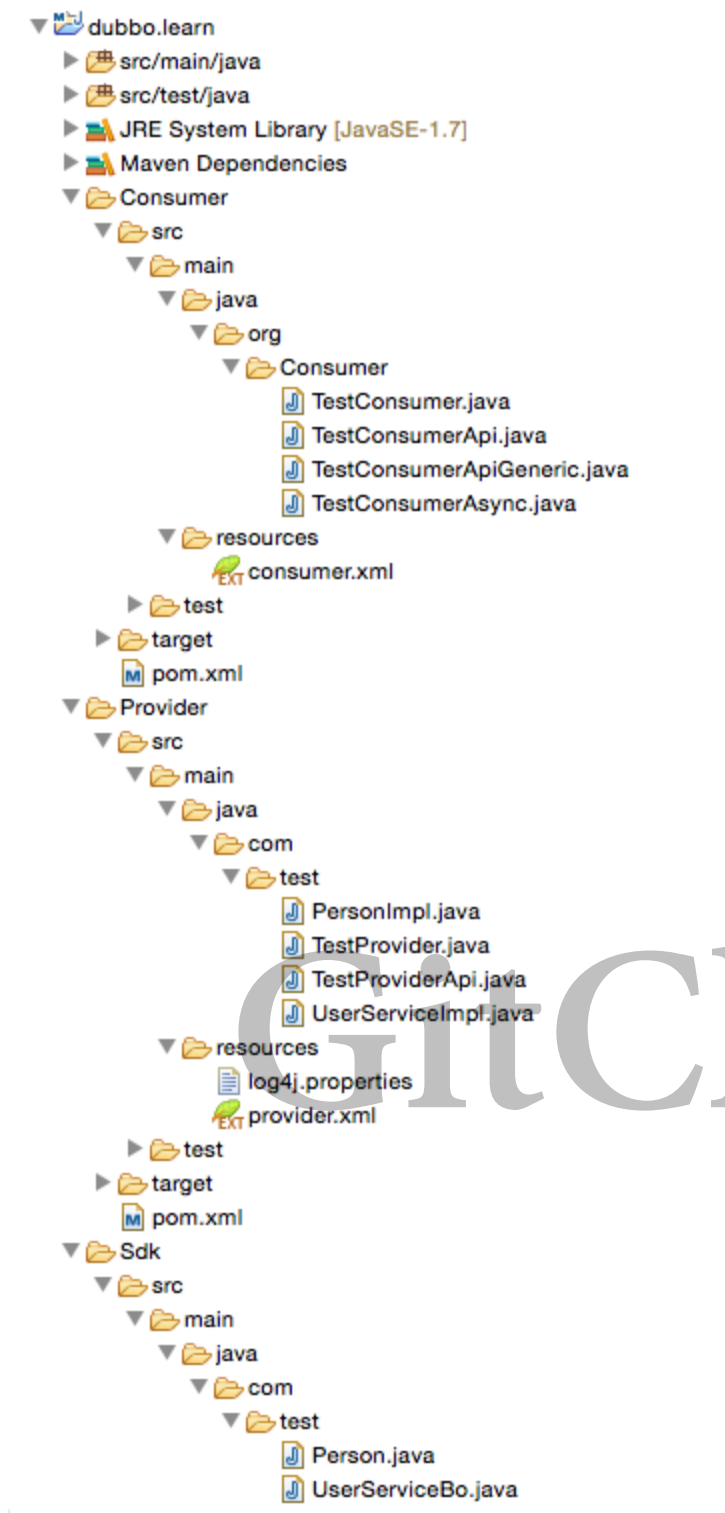
```

可知 zk 在端口 2181 进行监听，至此服务注册中心搭建完毕。

## 四、服务提供方与服务消费方搭建

### 4.1 本文 Demo 结构介绍

首先讲解下本文使用的 Demo 的结构，Demo 使用 Maven 聚合功能，里面有三个模块，目录如下：



- 其中 Consumer 模块为服务消费者，里面 TestConsumer 和 consumer.xml 组成了基于 Spring 配置方式的服务调用，TestConsumerApi 是基于 Dubbo API 方式的服务调用，TestConsumerApiGeneric 是泛化方式的服务调用，TestConsumerAsync 是异步调用的方式。
- 其中 Provider 模块为服务提供者，里面 TestProvider 和 provider.xml 组成了基于 Spring 配置方式的服务提供，TestProviderApi 是基于 Dubbo API 的服务提供，UserServiceImpl 为服务实现类。
- 其中 SDK 模块是一个二方包，用来存放服务提供者所有的接口，是为了代码复用使用，在服务提供者和消费者的模块里面都需要引入这个二方包。

其中 SDK 里面的接口定义源码如下：

```

public interface UserServiceBo {
    String sayHello(String name);
    String sayHello2(String name);
    String testPojo(Person person);
}

```

在 SDK 模块执行 `mvn clean install` 命令会安装该模块的 Jar 到本地仓库，要想在其他模块引入该 Jar，必须要先执行这个安装步骤。

## 4.2 基于 Spring 配置的服务提供方与消费方搭建

### 4.2.1 基于 Spring 配置的服务提供方搭建

Provider 模块为服务提供者，作用是注册提供的服务到 zk，并使用 Netty 服务监听服务消费端的链接。里面 `TestProvider` 和 `provider.xml` 组成了基于 XML 方式的服务提供，`UserServiceImpl` 为服务实现类。

- 首先需要在 Provider 模块里面引入 SDK 模块，因为 Provider 模块需要用到 `UserServiceBo` 接口（需要在 SDK 模块执行 `mvn clean install` 命令会安装该模块的 Jar 到本地仓库）。
- 然后实现 `UserServiceBo` 接口为 `UserServiceImpl`，代码如下：

```

public class UserServiceImpl implements UserServiceBo{

    @Override
    public String sayHello(String name) {
        //让当前当前线程休眠2s
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        return name;
    }

    @Override
    public String sayHello2(String name) {
        //让当前当前线程休眠2s
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

```

```

        return name;
    }

    @Override
    public String testPojo(Person person) {
        return JSON.toJSONString(person);
    }
}

```

- 然后 provider.xml 的内容如下:

```

<?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:context="http://www.springframework.org/schema/context"
    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://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-
4.0.xsd
        http://code.alibabatech.com/schema/dubbo
        http://code.alibabatech.com/schema/dubbo/dubbo.xsd">

    <!-- 提供方应用信息，用于计算依赖关系 -->
    <dubbo:application name="dubboProvider" />

    <!-- 使用zookeeper注册中心暴露服务地址 -->
    <dubbo:registry address="zookeeper://127.0.0.1:2181" />

    <!-- 用dubbo协议在20880端口暴露服务 -->
    <dubbo:protocol name="dubbo" port="20880" />
    <!-- 启用monitor模块 -->
    <dubbo:monitor protocol="registry" />

    <bean id="userService" class="com.test.UserServiceImpl" />

    <!-- 声明需要暴露的服务接口 -->
    <dubbo:service interface="com.test.UserServiceBo"
ref="userService"
        group="dubbo" version="1.0.0" timeout="3000"/>

</beans>

```

- 然后日志文件 log4j.properties 内容如下:

```

log4j.rootLogger=INFO,A1
log4j.appender.A1=org.apache.log4j.ConsoleAppender
log4j.appender.A1.layout=org.apache.log4j.PatternLayout

```



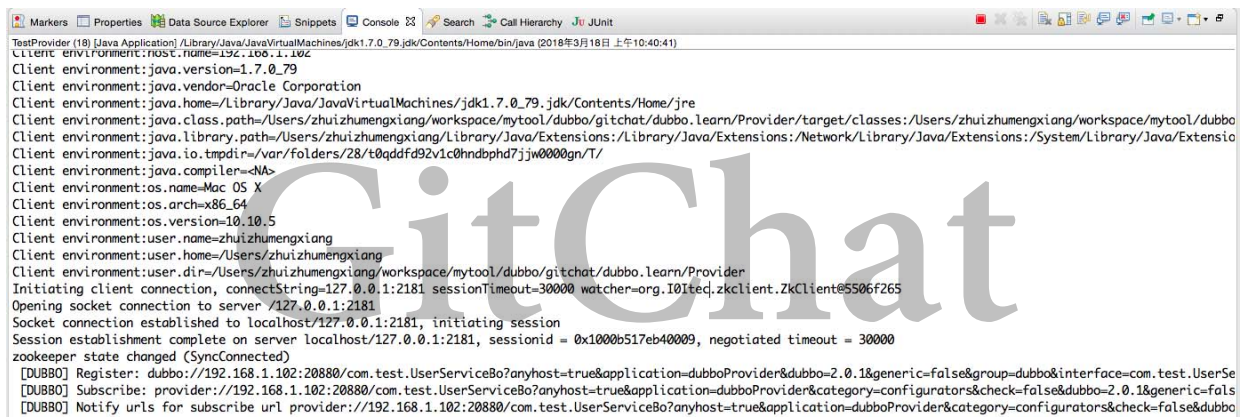
- 然后，编写服务发布测试类 TestProvider，代码如下：

```
public class TestProvider {

    public static void main(String[] arg) throws
    InterruptedException {
        ClassPathXmlApplicationContext context = new
        ClassPathXmlApplicationContext("classpath:provider.xml");

        //挂起当前线程，如果没有改行代码，服务提供者进程会消亡，服务消费者就
        发现不了提供者了
        Thread.currentThread().join();
    }
}
```

- 最后，运行 TestProvider 类，输出如下：



```
TestProvider (18) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.7.0_79.jdk/Contents/Home/bin/java (2018年3月18日 上午10:40:41)
Client environment: host.name=192.168.1.102
Client environment: java.version=1.7.0_79
Client environment: java.vendor=Oracle Corporation
Client environment: java.home=/Library/Java/JavaVirtualMachines/jdk1.7.0_79.jdk/Contents/Home/jre
Client environment: java.class.path=/Users/zhuizhumengxiang/workspace/mytool/dubbo/gitchat/dubbo.learn/Provider/target/classes:/Users/zhuizhumengxiang/workspace/mytool/dubbo
Client environment: java.library.path=/Users/zhuizhumengxiang/Library/Java/Extensions:/Library/Java/Extensions:/Network/Library/Java/Extensions:/System/Library/Java/Extensio
Client environment: java.io.tmpdir=/var/folders/28/t0qddfd92v1c0hndbphd7jjw0000gn/T/
Client environment: java.compiler=<NA>
Client environment: os.name=Mac OS X
Client environment: os.arch=x86_64
Client environment: os.version=10.10.5
Client environment: user.name=zhuizhumengxiang
Client environment: user.home=/Users/zhuizhumengxiang
Client environment: user.dir=/Users/zhuizhumengxiang/workspace/mytool/dubbo/gitchat/dubbo.learn/Provider
Initiating client connection, connectString=127.0.0.1:2181 sessionTimeout=30000 watcher=org.I01tec.zkclient.ZkClient@5506f265
Opening socket connection to server /127.0.0.1:2181
Socket connection established to localhost/127.0.0.1:2181, initiating session
Session establishment complete on server localhost/127.0.0.1:2181, sessionId = 0x1000b517eb40009, negotiated timeout = 30000
zookeeper state changed (SyncConnected)
[DUBBO] Register: dubbo://192.168.1.102:20880/com.test.UserServiceBo?anyhost=true&application=dubboProvider&dubbo=2.0.1&generic=false&group=dubbo&interface=com.test.UserService
[DUBBO] Subscribe: provider://192.168.1.102:20880/com.test.UserServiceBo?anyhost=true&application=dubboProvider&category=configurators&check=false&dubbo=2.0.1&generic=false
[DUBBO] Notify urls for subscribe url provider://192.168.1.102:20880/com.test.UserServiceBo?anyhost=true&application=dubboProvider&category=configurators&check=false&dubbo=2.0.1&generic=false
```

说明当前服务已经注册了 ZooKeeper 了。

#### 4.2.2 基于Spring配置的服务消费方搭建

Consumer 模块为服务消费方，服务消费端主要是从 zk 获取自己需要的服务提供者的 ip 列表，然后根据路由规则选择一个 ip 进行远程调用。里面 TestConsumer 和 consumer.xml 组成了基于 XML 方式的服务调用。

- 首先需要在 Consumer 模块里面引入 SDK 模块，因为 Consumer 模块需要用到 UserServiceBo 接口（泛化调用时候不需要这个步骤）。
- 然后 consumer.xml 内容如下：

```
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:context="http://www.springframework.org/schema/context"
        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://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-
4.0.xsd
    http://code.alibabatech.com/schema/dubbo
    http://code.alibabatech.com/schema/dubbo/dubbo.xsd">

```

<!-- 消费方应用名，用于计算依赖关系，不是匹配条件，不要与提供方一样 -->

```
<dubbo:application name="dubboConsumer" />
```

<!-- 使用multicast广播注册中心暴露发现服务地址 -->

```
<dubbo:registry protocol="zookeeper"
```

```
address="zookeeper://127.0.0.1:2181" />
```

<!-- 启动monitor-->

```
<dubbo:monitor protocol="registry" />
```

<!-- 生成远程服务代理，可以和本地bean一样使用demoService -->

```
<dubbo:reference id="userService"
```

```
interface="com.test.UserServiceBo" group="dubbo" version="1.0.0"
```

```
timeout="3000"/>
```

```
</beans>
```

- 然后测试服务消费类 TestConsumer 代码如下：

```

public class TestConsumer {

    public static void main(String[] args) {
        ClassPathXmlApplicationContext context = new
        ClassPathXmlApplicationContext(
            new String[] { "classpath:consumer.xml" });

        final UserServiceBo demoService = (UserServiceBo)
        context.getBean("userService");

        System.out.println(demoService.sayHello("哈哈"));
    }
}

```

- 最后运行 TestConsumer，会输出：

```

TestConsumer [Java Application] /Library/Java/JavaVirtualMachines/jdk1.7.0_79.jdk/Contents/Home/bin/java (2018年3月18日 上午11:07:56)
log4j:WARN No appenders could be found for logger (org.springframework.core.env.StandardEnvironment).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
哈哈

```

说明服务消费端已经正常调用了服务提供方的服务了。

**注：**至此一个经典的含有服务提供者，服务消费者，服务注册中心的简单分布式系统搭建完毕了。

## 4.3 基于 Dubbo API 方式的服务提供方与消费方搭建

### 4.3.1 基于Dubbo API 方式的服务提供方搭建

其中 Provider 模块为服务提供者，里面 TestProviderApi 是基于 Dubbo API 的服务提供， UserServiceImpl 为服务实现类。

- 首先需要在 Provider 模块里面引入 SDK 模块，这个不变。
- 然后实现 UserServiceBo 接口为 UserServiceImpl，这个也不变。
- 然后编写 Dubbo API 服务提供测试代码 TestProviderApi 代码如下：

```
public class TestProviderApi {

    public static void main(String[] arg) throws
    InterruptedException {

        // (4.3.1-1) 等价于 <bean id="userService"
class="com.test.UserServiceImpl" />
        UserServiceBo userService = new UserServiceImpl();
        // (4.3.1-2) 等价于 <dubbo:application
name="dubboProvider" />
        ApplicationConfig application = new ApplicationConfig();
        application.setName("dubboProvider");

        // (4.3.1-3) 等价于 <dubbo:registry
address="zookeeper://127.0.0.1:2181" />
        RegistryConfig registry = new RegistryConfig();
        registry.setAddress("127.0.0.1:2181");
        registry.setProtocol("zookeeper");

        // (4.3.1-4) 等价于 <dubbo:protocol name="dubbo"
port="20880" />
        ProtocolConfig protocol = new ProtocolConfig();
        protocol.setName("dubbo");
        protocol.setPort(20880);

        //4.3.1-5) 等价于 <dubbo:monitor protocol="registry" />
        MonitorConfig monitorConfig = new MonitorConfig();
        monitorConfig.setProtocol("registry");

        //4.3.1-6) 等价于 <dubbo:service
interface="com.test.UserServiceBo" ref="userService"
//group="dubbo" version="1.0.0" timeout="3000"/>
        ServiceConfig<UserServiceBo> service = new
ServiceConfig<UserServiceBo>(); // 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.setMonitor(monitorConfig);
        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(UserServiceBo.class);
        service.setRef(userService);
        service.setVersion("1.0.0");
        service.setGroup("dubbo");
        service.setTimeout(3000);
        service.export();

//4.3.1-8) 挂起当前线程
        Thread.currentThread().join();
    }
}

```

### 4.3.2 基于Dubbo API 方式的服务消费方搭建

其中 Consumer 模块为服务消费者，里面 TestConsumerApi 是基于 Dubbo API 方式的服务调用。

- 首先需要在 Consumer 模块里面引入 SDK 模块，这个不变。
- 编写基于 Dubbo API 消费服务的测试类 TestConsumerApi 代码如下：

```

public class TestConsumerApi {
    public static void main(String[] args) throws
InterruptedException {
        // 等价于 <dubbo:application name="dubboConsumer" />
        ApplicationConfig application = new ApplicationConfig();
        application.setName("dubboConsumer");

        // 等价于 <dubbo:registry protocol="zookeeper"
address="zookeeper://127.0.0.1:2181" />
        RegistryConfig registry = new RegistryConfig();
        registry.setAddress("127.0.0.1:2181");
        registry.setProtocol("zookeeper");

        //等价于 <dubbo:monitor protocol="registry" />
        MonitorConfig monitorConfig = new MonitorConfig();
        monitorConfig.setProtocol("registry");

        //等价于<dubbo:reference id="userService"
interface="com.test.UserServiceBo"
        //group="dubbo" version="1.0.0" timeout="3000" />
        ReferenceConfig<UserServiceBo> reference = new
ReferenceConfig<UserServiceBo>(); // 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(UserServiceBo.class);
    }
}

```

```

        reference.setVersion("1.0.0");
        reference.setGroup("dubbo");
        reference.setTimeout(3000);
        reference.setInjvm(false);
        reference.setMonitor(monitorConfig);

        UserServiceBo userService = reference.get();
        System.out.println(userService.sayHello("哈哈"));
        Thread.currentThread().join();
    }
}

```

#### 4.4 服务消费端泛化调用

前面我们讲解基于 Spring 和基于 Dubbo API 方式搭建一个简单的分布式系统时候服务消费端是引入了一个 SDK 二方包的，里面存放了服务提供端提供的所有接口类，所以需要引入接口类是因为服务消费端一般是基于接口使用 JDK 代理实现远程调用的。

泛化接口调用方式主要用于服务消费端没有 API 接口类及模型类元（比如入参和出参的 POJO 类）的情况下使用；这时候参数及返回值中由于没有对应的 POJO 类，所以所有 POJO 均转换为 Map 表示。使用泛化调用时候服务消费模块不在需要引入 SDK 二方包。

下面基于 Dubbo API 来实现异步调用，在 Consumer 模块里面 TestConsumerApiGeneric 是泛化调用的方式，代码如下：

```

public class TestConsumerApiGeneric {
    public static void main(String[] args) throws IOException {

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

        // 连接注册中心配置
        RegistryConfig registry = new RegistryConfig();
        registry.setAddress("127.0.0.1:2181");
        registry.setProtocol("zookeeper");

        // 泛型参数设置为GenericService
        ReferenceConfig<GenericService> reference = new
ReferenceConfig<GenericService>();
        reference.setApplication(application);
        reference.setRegistry(registry);
        reference.setVersion("1.0.0");
        reference.setGroup("dubbo");
        reference.setTimeout(3000);

        //设置为泛化

```

```

        reference.setInterface("com.test.UserServiceBo");
        reference.setGeneric(true);

        //用com.alibaba.dubbo.rpc.service.GenericService替代所有接口引用
        GenericService userService = reference.get(); //

        // 基本类型以及Date,List,Map等不需要转换, 直接调用,如果返回值为POJO也将自动转成Map
        Object result = userService.$invoke("sayHello", new
String[] { "java.lang.String"},
        new Object[] { "哈哈哈哈哈"});

        System.out.println(JSON.json(result));

        //POJO参数转换为map
        Map<String, Object> map = new HashMap<String, Object>();
        map.put("class", "com.test.PersonImpl");
        map.put("name", "jiaduo");
        map.put("password", "password");

        result = userService.$invoke("testPojo", new String[] {
"com.test.Person" }, new Object[] { map });
        System.out.println((result));
    }
}

```

这里由于 sayHello 的参数是 String, 没有很好的体现参数转换为 Map, 下面我们具体来说下 POJO 参数转换 Map 的含义。

比如服务提供者提供的一个接口的 testPojo(Person person) 方法的参数为如下 POJO:

```

package com.test;

public class PersonImpl implements Person {
    private String name;
    private String password;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
}

```

则 POJO 数据：

```
Person person = new PersonImpl();
person.setName("jiaduo");
person.setPassword("password");
```

正常情况调用接口是使用：

```
servicePerson.testPojo(person);
```

泛化调用下需要首先转换 person 为 Map 后如下表示：

```
Map<String, Object> map = new HashMap<String, Object>();
// 注意：如果参数类型是接口，或者List等丢失泛型，可通过class属性指定类型。
map.put("class", "com.test.PersonImpl");
map.put("name", "jiaduo");
map.put("password", "password");
```

然后使用下面方法进行泛化调用：

```
servicePerson.$invoke("testPojo", new String[]
{"com.test.Person"}, new Object[]{map});
```

泛化调用通常用于框架集成，比如：实现一个通用的服务测试框架，可通过 GenericService 调用所有服务实现，而不需要依赖服务实现方提供的接口类以及接口的入参和出参的 POJO 类。

## 4.5 服务消费端异步调用

无论前面我们讲解的正常调用还是泛化调用也好，都是进行同步调用的，也就是服务消费方发起一个远程调用后，调用线程要被阻塞挂起，直到服务提供方返回。

本节来讲解下服务消费端异步调用，异步调用是指服务消费方发起一个远程调用后，不等服务提供方返回结果，调用方法就返回了，也就是当前线程不会被阻塞，这就允许调用方同时调用多个远程方法。

在 Consumer 模块里面 TestConsumerAsync 是泛化调用，代码如下：

```
public class TestConsumerAsync {
    public static void main(String[] args) throws
        InterruptedException, ExecutionException {
        // 当前应用配置
        ApplicationConfig application = new ApplicationConfig();
```



```

        application.setName("dubboConsumer");

        // 连接注册中心配置
        RegistryConfig registry = new RegistryConfig();
        registry.setAddress("127.0.0.1:2181");
        registry.setProtocol("zookeeper");

        // 引用远程服务
        ReferenceConfig<UserServiceBo> reference = new
ReferenceConfig<UserServiceBo>();
        reference.setApplication(application);
        reference.setRegistry(registry);
        reference.setInterface(UserServiceBo.class);
        reference.setVersion("1.0.0");
        reference.setGroup("dubbo");
        reference.setTimeout(3000);

        // (1) 设置为异步调用
        reference.setAsync(true);

        // 和本地bean一样使用xxxService
        UserServiceBo userService = reference.get();

        long startTime = System.currentTimeMillis() / 1000;

        // (2) 因为异步调用, 此处返回null
        System.out.println(userService.sayHello("哈哈"));
        // 拿到调用的Future引用, 当结果返回后, 会被通知和设置到此Future
        Future<String> userServiceFutureOne =
RpcContext.getContext().getFuture();

        // (3) 因为异步调用, 此处返回null
        System.out.println(userService.sayHello2("哈哈2"));
        // 拿到调用的Future引用, 当结果返回后, 会被通知和设置到此Future
        Future<String> userServiceFutureTwo =
RpcContext.getContext().getFuture();

        // (4) 阻塞到get方法, 等待结果返回
        System.out.println(userServiceFutureOne.get());
        System.out.println(userServiceFutureTwo.get());
        long endTime = System.currentTimeMillis() / 1000;

        System.out.println("costs:" + (endTime - startTime));

    }
}

```

运行上面代码输出:

```

Markers Properties Data Source Explorer Snippets Console Search Call Hierarchy JUnit
TestConsumerAsync [Java Application] /Library/Java/JavaVirtualMachines/jdk1.7.0_79.jdk/Contents/Home/bin/java (2018年3月18日 上午11:46:49)
log4j:WARN No appenders could be found for logger (com.alibaba.dubbo.common.logger.LoggerFactory).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
null
null
哈哈
哈哈2
costs:2

```

其中代码 (2) (3) 处输出 null, 说明开启异步调用后调用方直接返回 null。

输出 costs:2 说明异步调用生效了, 因为 sayHello 和 sayHello2 方法内都 sleep 了2s的, 如果是顺序调用则会耗时至少4s,这里耗时2s说明两次调用是并发进行的。

异步调用是基于 NIO 的非阻塞实现并行调用, 客户端不需要启动多线程即可完成并行调用多个远程服务, 相对调用不同的服务使用不同线程来说开销较小。

## 五、管理控制台搭建

开源的 Dubbo 的服务管理控制台是阿里巴巴内部裁剪版本, 开源部分主要包含: 路由规则, 动态配置, 服务降级, 访问控制, 权重调整, 负载均衡, 等管理功能。

- 首先我们需要到 <https://github.com/alibaba/dubbo/tree/2.5.x> 下载 Dubbo 的源码, 解压后如下:

```

➔ /Users/zhuizhumengxiang/Downloads/dubbo-2.5.x git:(master) ✗ ls -al
total 192
drwxr-xr-x@ 33 zhuizhumengxiang staff 1122  3 15 18:57 .
drwx-----+ 73 zhuizhumengxiang staff 2482  3 18 11:47 ..
-rw-r--r--@  1 zhuizhumengxiang staff 10244  3 15 18:57 .DS_Store
-rwxr-xr-x@  1 zhuizhumengxiang staff   248  3 12 23:43 .gitignore
drwxr-xr-x@  3 zhuizhumengxiang staff   102  3 12 23:43 .mvn
-rwxr-xr-x@  1 zhuizhumengxiang staff   203  3 12 23:43 .travis.yml
-rwxr-xr-x@  1 zhuizhumengxiang staff  3219  3 12 23:43 CODE_OF_CONDUCT.md
-rwxr-xr-x@  1 zhuizhumengxiang staff  2331  3 12 23:43 CONTRIBUTING.md
-rwxr-xr-x@  1 zhuizhumengxiang staff 11358  3 12 23:43 LICENSE
-rwxr-xr-x@  1 zhuizhumengxiang staff  6170  3 12 23:43 NOTICE
-rwxr-xr-x@  1 zhuizhumengxiang staff  1404  3 12 23:43 PULL_REQUEST_TEMPLATE.md
-rwxr-xr-x@  1 zhuizhumengxiang staff  1273  3 12 23:43 README.md
drwxr-xr-x@  3 zhuizhumengxiang staff   102  3 12 23:43 codestyle
drwxr-xr-x@  5 zhuizhumengxiang staff   170  3 14 17:19 dubbo
drwxr-xr-x@  8 zhuizhumengxiang staff   272  3 16 11:48 dubbo-admin
drwxr-xr-x@  5 zhuizhumengxiang staff   170  3 14 17:17 dubbo-cluster
drwxr-xr-x@  5 zhuizhumengxiang staff   170  3 14 17:17 dubbo-common
drwxr-xr-x@  5 zhuizhumengxiang staff   170  3 12 23:43 dubbo-config
drwxr-xr-x@  8 zhuizhumengxiang staff   272  3 12 23:43 dubbo-container
drwxr-xr-x@  6 zhuizhumengxiang staff   204  3 12 23:43 dubbo-demo
drwxr-xr-x@  5 zhuizhumengxiang staff   170  3 12 23:43 dubbo-filter
drwxr-xr-x@  3 zhuizhumengxiang staff   102  3 12 23:43 dubbo-maven
drwxr-xr-x@  5 zhuizhumengxiang staff   170  3 12 23:43 dubbo-monitor
drwxr-xr-x@  4 zhuizhumengxiang staff   136  3 12 23:43 dubbo-plugin
drwxr-xr-x@  8 zhuizhumengxiang staff   272  3 12 23:43 dubbo-registry
drwxr-xr-x@ 11 zhuizhumengxiang staff   374  3 12 23:43 dubbo-remoting
drwxr-xr-x@ 13 zhuizhumengxiang staff   442  3 12 23:43 dubbo-rpc
drwxr-xr-x@  6 zhuizhumengxiang staff   204  3 15 18:57 dubbo-simple
drwxr-xr-x@  7 zhuizhumengxiang staff   238  3 12 23:43 dubbo-test
drwxr-xr-x@  5 zhuizhumengxiang staff   170  3 14 17:16 hessian-lite
-rwxr-xr-x@  1 zhuizhumengxiang staff  6510  3 12 23:43 mvnw
-rwxr-xr-x@  1 zhuizhumengxiang staff  4995  3 12 23:43 mvnw.cmd
-rwxr-xr-x@  1 zhuizhumengxiang staff 22417  3 12 23:43 pom.xml

```

- 在 dubbo-2.5.x 目录执行 mvn clean package -Dmaven.test.skip=true 会生成如下结果：

```
[INFO] -----
[INFO] Reactor Summary:
[INFO]
[INFO] dubbo-parent .....
SUCCESS [2.857s]
[INFO] Hessian Lite(Alibaba embed version) .....
SUCCESS [6.518s]
[INFO] dubbo-common .....
SUCCESS [7.706s]
[INFO] dubbo-container .....
SUCCESS [0.057s]
[INFO] dubbo-container-api .....
SUCCESS [3.398s]
[INFO] dubbo-container-spring .....
SUCCESS [1.099s]
[INFO] dubbo-container-jetty .....
SUCCESS [1.164s]
[INFO] dubbo-container-log4j .....
SUCCESS [1.196s]
[INFO] dubbo-container-logback .....
SUCCESS [1.138s]
[INFO] dubbo-remoting .....
SUCCESS [0.045s]
[INFO] dubbo-remoting-api .....
SUCCESS [3.615s]
[INFO] dubbo-remoting-netty .....
SUCCESS [1.959s]
[INFO] dubbo-remoting-mina .....
SUCCESS [1.602s]
[INFO] dubbo-remoting-grizzly .....
SUCCESS [1.752s]
[INFO] dubbo-remoting-p2p .....
SUCCESS [2.148s]
[INFO] dubbo-remoting-http .....
SUCCESS [2.154s]
[INFO] dubbo-remoting-zookeeper .....
SUCCESS [2.165s]
[INFO] dubbo-remoting-netty4 .....
SUCCESS [2.721s]
[INFO] dubbo-rpc .....
SUCCESS [0.060s]
[INFO] dubbo-rpc-api .....
SUCCESS [3.397s]
[INFO] dubbo-rpc-default .....
SUCCESS [3.278s]
[INFO] dubbo-rpc-injvm .....
SUCCESS [1.706s]
```

```
[INFO] dubbo-rpc-rmi .....
SUCCESS [1.373s]
[INFO] dubbo-rpc-hessian .....
SUCCESS [1.791s]
[INFO] dubbo-rpc-http .....
SUCCESS [1.210s]
[INFO] dubbo-rpc-webservice .....
SUCCESS [1.816s]
[INFO] dubbo-cluster .....
SUCCESS [2.690s]
[INFO] dubbo-registry .....
SUCCESS [0.033s]
[INFO] dubbo-registry-api .....
SUCCESS [2.548s]
[INFO] dubbo-monitor .....
SUCCESS [0.020s]
[INFO] dubbo-monitor-api .....
SUCCESS [1.309s]
[INFO] dubbo-filter .....
SUCCESS [0.023s]
[INFO] dubbo-filter-validation .....
SUCCESS [1.474s]
[INFO] dubbo-filter-cache .....
SUCCESS [1.395s]
[INFO] dubbo-registry-default .....
SUCCESS [1.402s]
[INFO] dubbo-monitor-default .....
SUCCESS [1.258s]
[INFO] dubbo-registry-multicast .....
SUCCESS [1.336s]
[INFO] dubbo-config .....
SUCCESS [0.018s]
[INFO] dubbo-config-api .....
SUCCESS [3.498s]
[INFO] dubbo-config-spring .....
SUCCESS [3.326s]
[INFO] dubbo-rpc-thrift .....
SUCCESS [2.115s]
[INFO] dubbo-rpc-memcached .....
SUCCESS [1.476s]
[INFO] dubbo-rpc-redis .....
SUCCESS [1.367s]
[INFO] dubbo-registry-zookeeper .....
SUCCESS [1.597s]
[INFO] dubbo-registry-redis .....
SUCCESS [2.111s]
[INFO] dubbo-plugin .....
SUCCESS [0.027s]
[INFO] dubbo-qos .....
SUCCESS [2.574s]
[INFO] dubbo .....
SUCCESS [2.662s]
```

```
[INFO] dubbo-simple .....
SUCCESS [0.017s]
[INFO] dubbo-registry-simple .....
SUCCESS [5.246s]
[INFO] dubbo-monitor-simple .....
SUCCESS [11.301s]
[INFO] dubbo-admin .....
SUCCESS [11.472s]
[INFO] dubbo-demo .....
SUCCESS [0.034s]
[INFO] dubbo-demo-api .....
SUCCESS [1.097s]
[INFO] dubbo-demo-provider .....
SUCCESS [1.892s]
[INFO] dubbo-demo-consumer .....
SUCCESS [1.473s]
[INFO] dubbo-test .....
SUCCESS [0.022s]
[INFO] dubbo-test-benchmark .....
SUCCESS [5.324s]
[INFO] dubbo-test-compatibility .....
SUCCESS [0.017s]
[INFO] dubbo-test-spring3 .....
SUCCESS [1.719s]
[INFO] dubbo-test-integration .....
SUCCESS [0.149s]
[INFO] dubbo-test-examples .....
SUCCESS [3.413s]
[INFO] -----
-----
[INFO] BUILD SUCCESS
[INFO] -----
-----
[INFO] Total time: 2:16.562s
[INFO] Finished at: Wed Mar 14 17:19:06 CST 2018
[INFO] Final Memory: 38M/654M
[INFO] -----
-----
```

然 后 进 入 `/Users/zhuizhumengxiang/Downloads/dubbo-2.5.x/dubbo-admin/target` 目录, 会发现生成了 `dubbo-admin-2.5.10.war`。

```

➔ /Users/zhuizhumengxiang/Downloads/dubbo-2.5.x/dubbo-admin/target git:(master) X ls -al
total 66928
drwxr-xr-x 10 zhuizhumengxiang staff 340 3 16 11:48 .
drwxr-xr-x@ 8 zhuizhumengxiang staff 272 3 16 11:48 ..
drwxr-xr-x 4 zhuizhumengxiang staff 136 3 16 11:48 classes
drwxr-xr-x 10 zhuizhumengxiang staff 340 3 14 17:18 dubbo-admin-2.5.10
-rw-r--r-- 1 zhuizhumengxiang staff 220234 3 14 17:18 dubbo-admin-2.5.10-sources.jar
-rw-r--r-- 1 zhuizhumengxiang staff 34043045 3 14 17:18 dubbo-admin-2.5.10.war
drwxr-xr-x 3 zhuizhumengxiang staff 102 3 14 17:18 generated-sources
drwxr-xr-x 3 zhuizhumengxiang staff 102 3 16 11:48 m2e-wtp
drwxr-xr-x 3 zhuizhumengxiang staff 102 3 14 17:18 maven-archiver
drwxr-xr-x 19 zhuizhumengxiang staff 646 3 16 11:48 test-classes

```

- 然后解压拷贝 dubbo-admin-2.5.10.war 后拷贝到一个 Servlet 容器，本文使用 Tomcat 容器，可以在 <https://tomcat.apache.org/> 这里下载 Tomcat 二进制包，然后拷贝 dubbo-admin-2.5.10 到 apache-tomcat-7.0.10/webapps 目录下：

```

➔ /Users/zhuizhumengxiang/workspace/mytool/dubbo/apache-tomcat-7.0.10/webapps git:(master) X ls -al
total 16
drwxr-xr-x@ 8 zhuizhumengxiang staff 272 3 15 12:42 .
drwxr-xr-x@ 14 zhuizhumengxiang staff 476 3 15 16:40 ..
-rw-r--r--@ 1 zhuizhumengxiang staff 6148 3 15 12:42 .DS_Store
drwxr-xr-x 19 zhuizhumengxiang staff 646 12 31 15:02 ROOT
drwxr-xr-x 49 zhuizhumengxiang staff 1666 12 31 15:02 docs
drwx----- 10 zhuizhumengxiang staff 340 3 14 17:30 dubbo-admin-2.5.10
drwxr-xr-x@ 10 zhuizhumengxiang staff 340 12 31 15:02 host-manager
drwxr-xr-x@ 11 zhuizhumengxiang staff 374 12 31 15:02 manager

```

- 然后进入 dubbo-admin-2.5.10/WEB-INF 目录修改 dubbo.properties 文件：

```

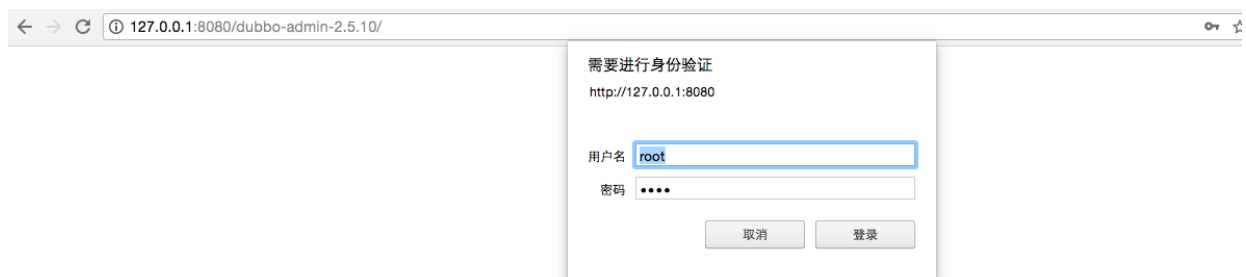
dubbo.registry.address=zookeeper://127.0.0.1:2181
dubbo.admin.root.password=root
dubbo.admin.guest.password=guest

```

这里主要 dubbo.registry.address 为 zk 的地址。

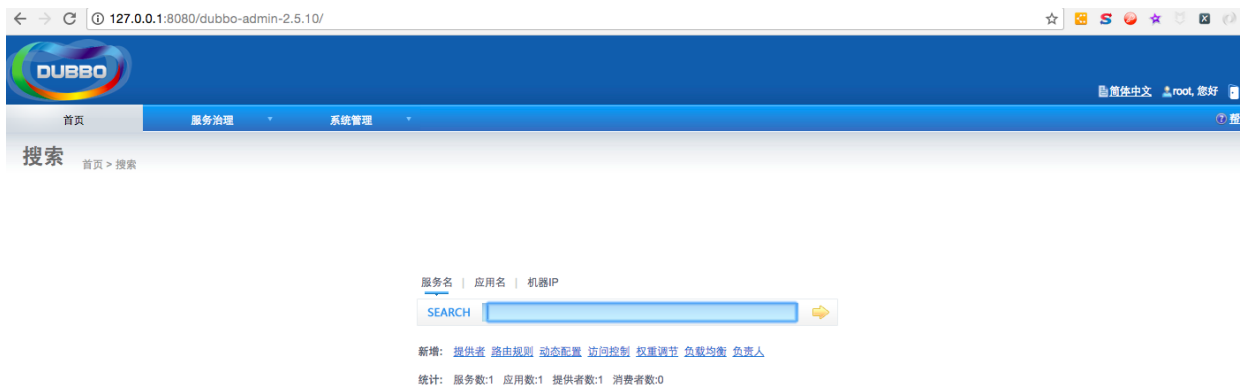
- 最后在 apache-tomcat-7.0.10/bin 下执行 `sh catalina.sh run` 启动 Tomcat。

启动后访问 <http://127.0.0.1:8080/dubbo-admin-2.5.10/> 会出现下面界面：



用户名和密码都输入root后，单击登陆，进入下面界面：





至此说明管理控制台搭建完毕了。

现在你就可以使用管理控制台管理和查看服务信息了。

比如你可以搜索一个服务：dubbo/com.test.UserServiceBo:1.0.0：



单击搜索后页面跳转到：



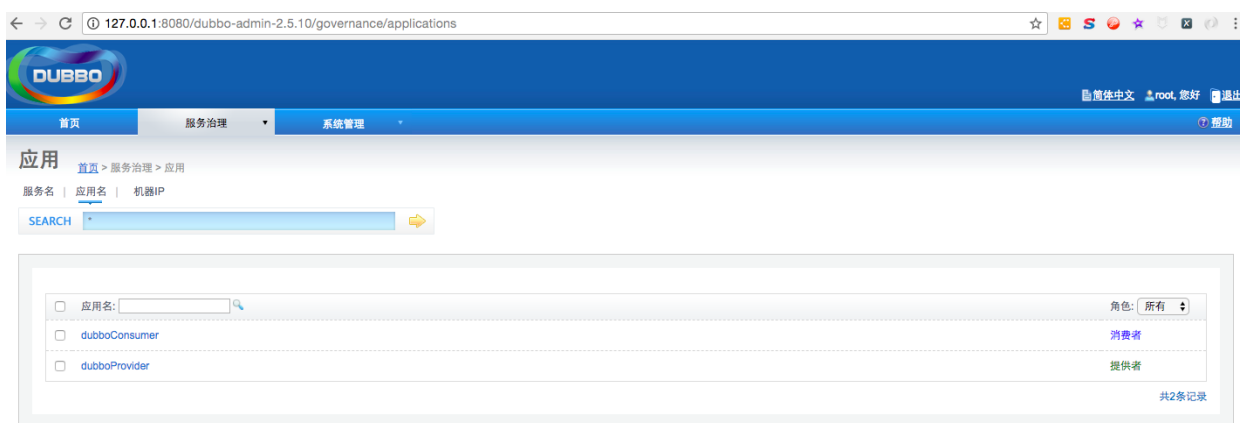
比如你可以查看服务提供者：



比如你可以查看服务消费者：



比如你可以查看当前都有哪些应用：



**注：**管理控制台不是使用 Dubbo 搭建分布式系统必须的，但是有了它我们可以对服务进行很好的治理和监控。

## 六、监控平台的搭建

dubbo-monitor 主要用来统计服务的调用次数和调用时间的监控中心，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，监控中心则使用数据绘制图表来显示。

- 服务消费方和提供方需要显示开启 monitor。

如果使用 Spring 配置的服务消费方和提供方，则需要在对应 XML 添加下面配置：

```
<dubbo:monitor protocol="registry"/>
```

其中 protocol 为"registry"，表示服务提供方和消费方从服务注册中心发现监控中心 (monitor) 地址。

如果使用的 Dubbo API 方式需要首先创建一个 MonitorConfig 对象。

```
MonitorConfig monitorConfig = new MonitorConfig();  
monitorConfig.setProtocol("registry");
```

然后调用 reference.setMonitor(monitorConfig); 设置到消费配置对象里面。

- 进入 /Users/zhuizhumengxiang/Downloads/dubbo-2.5.x/dubbo-simple/dubbo-monitor-simple/target 目录，会发现生成了 dubbo-monitor-simple-2.5.10-assembly.tar.gz：



```
→ /Users/zhuizhumengxiang/Downloads/dubbo-2.5.x/dubbo-simple/dubbo-monitor-simple/target git:(master) X ls  
archive-tmp  
classes  
dependency-maven-plugin-markers  
dubbo  
dubbo-monitor-simple-2.5.10  
dubbo-monitor-simple-2.5.10-assembly.tar.gz  
dubbo-monitor-simple-2.5.10-sources.jar  
dubbo-monitor-simple-2.5.10.jar
```

- 解压 dubbo-monitor-simple-2.5.10-assembly.tar.gz，进入 dubbo-monitor-simple-2.5.10/conf/ 目录修改 dubbo.properties：

```
dubbo.container=log4j,spring,registry,jetty  
dubbo.application.name=simple-monitor  
dubbo.application.owner= jiaduo  
dubbo.registry.address=zookeeper://127.0.0.1:2181  
dubbo.protocol.port=7070  
dubbo.jetty.port=8081  
dubbo.jetty.directory=/Users/zhuizhumengxiang/Downloads/dubbo-2.5.x/dubbo-simple/dubbo-monitor-simple/target/dubbo-monitor-simple-2.5.10/monitor  
dubbo.charts.directory=/Users/zhuizhumengxiang/Downloads/dubbo-2.5.x/dubbo-simple/dubbo-monitor-simple/target/dubbo-monitor-simple-2.5.10/monitor/charts  
dubbo.statistics.directory=/Users/zhuizhumengxiang/Downloads/dubbo-2.5.x/dubbo-simple/dubbo-monitor-simple/target/dubbo-monitor-simple-2.5.10/monitor/statistics
```

```
dubbo.log4j.file=logs/dubbo-monitor-simple.log
dubbo.log4j.level=WARN
```

其中 `dubbo.registry.address=zookeeper://127.0.0.1:2181` 设置注册中心地址，这里设置为 zk 的地址；

其中 `dubbo.protocol.port=7070`，是 monitor 提供的远程服务监听端口，服务提供者和消费者会调用这个端口提供的服务，发送统计信息到 monitor。

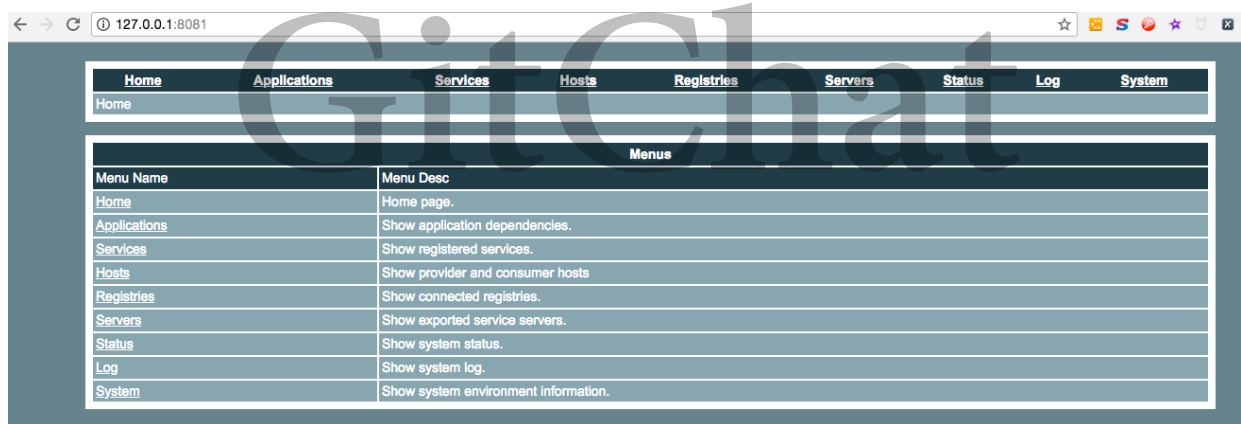
`dubbo.charts.directory` 和 `dubbo.statistics.directory` 为 monitor 本地存放的监控数据文件的位置；

`dubbo.jetty.port=8081`，设置 Jetty 容器的监听地址，类似于 Tomcat 的 8080 端口，这里设置为 8081；

- 然后进入 `dubbo-monitor-simple-2.5.10/bin`，执行 `sh start.sh` 启动 monitor；

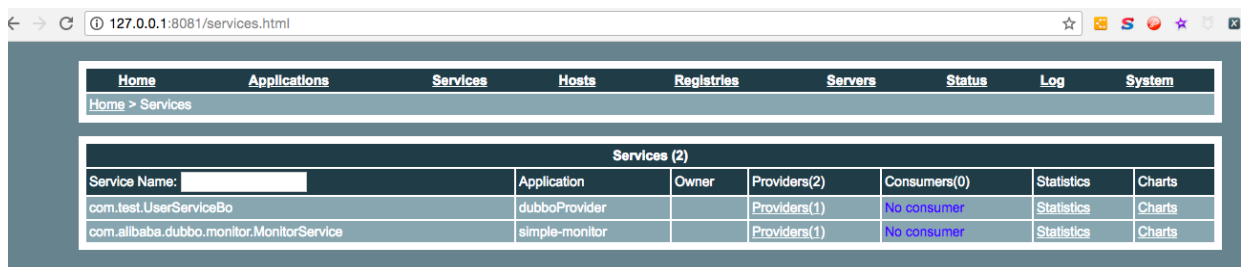
```
➔ /Users/zhuizhumengxiang/Downloads/dubbo-2.5.x/dubbo-simple/dubbo-monitor-simple/target/dubbo-monitor-simple-2.5.10/bin git:(master) X sh start.sh
-e Starting the simple-monitor ...-e .-e .-e .-e .-e .-e .OK!
PID: 29747
STDOUT: logs/stdout.log
```

至此 monitor 启动了，访问 `http://127.0.0.1:8081/` 会出现下面界面：

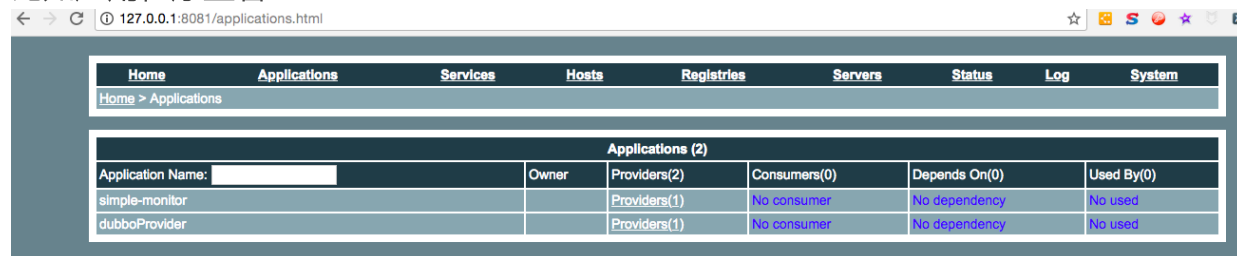


然后我们就可以使用监控平台做一些事情了。

比如服务查看：



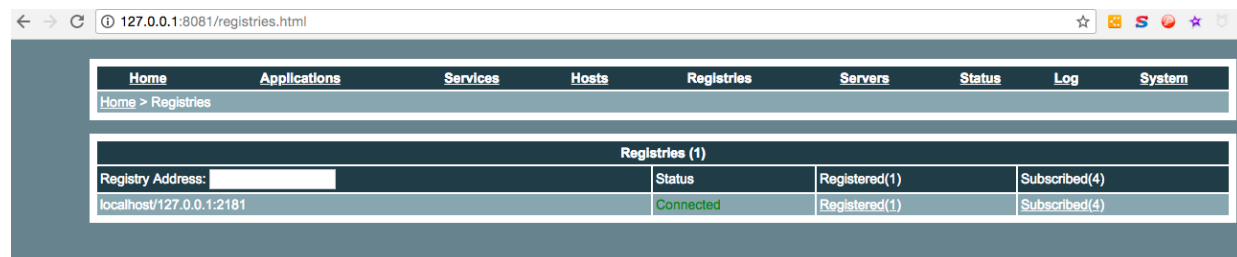
比如应用程序查看：



The screenshot shows the Dubbo Admin web interface at 127.0.0.1:8081/applications.html. It features a navigation bar with tabs: Home, Applications, Services, Hosts, Registries, Servers, Status, Log, and System. The 'Applications' tab is selected, showing a table of applications. The table has columns for Application Name, Owner, Providers, Consumers, Depends On, and Used By. Two applications are listed: 'simple-monitor' and 'dubboProvider'.

| Application Name | Owner | Providers(2) | Consumers(0) | Depends On(0) | Used By(0) |
|------------------|-------|--------------|--------------|---------------|------------|
| simple-monitor   |       | Providers(1) | No consumer  | No dependency | No used    |
| dubboProvider    |       | Providers(1) | No consumer  | No dependency | No used    |

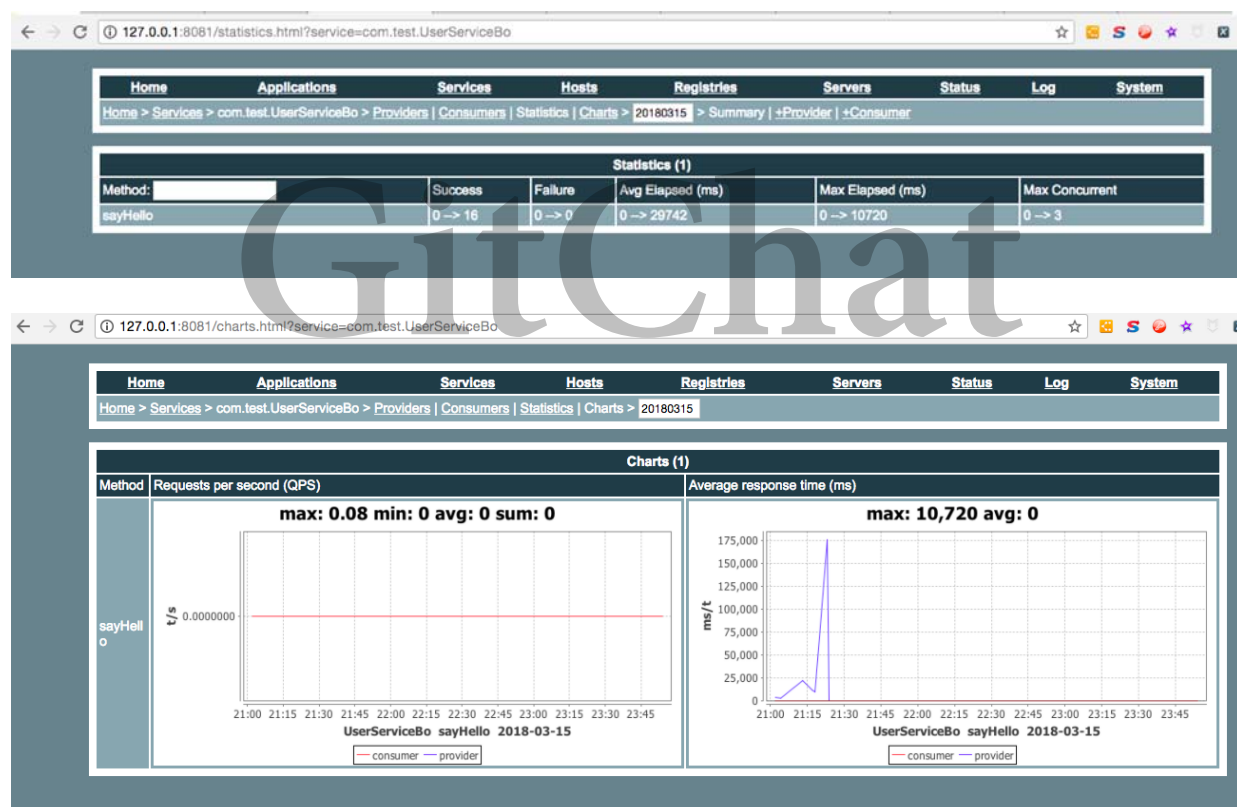
比如服务注册中心查看：



The screenshot shows the Dubbo Admin web interface at 127.0.0.1:8081/registries.html. It features a navigation bar with tabs: Home, Applications, Services, Hosts, Registries, Servers, Status, Log, and System. The 'Registries' tab is selected, showing a table of registries. The table has columns for Registry Address, Status, Registered, and Subscribed. One registry is listed: 'localhost/127.0.0.1:2181'.

| Registry Address         | Status    | Registered(1) | Subscribed(4) |
|--------------------------|-----------|---------------|---------------|
| localhost/127.0.0.1:2181 | Connected | Registered(1) | Subscribed(4) |

比如调用情况统计：



**注：** dubbo-monitor 也不是使用 Dubbo 搭建分布式系统必须的组件，但是它用来统计服务的调用次调和调用时间的监控中心，这些数据有助于系统运维和调优。

## 七、总结

本文使用 Dubbo 搭建了一个含有服务消费者，服务注册中心（本 Chat 使用 ZooKeeper），管理控制台（Dubbo-Admin），监控平台（Dubbo-Monitor），麻雀虽小，却五脏俱全，并且讲解了服务消费端异步调用和泛化调用使用，环境搭建完毕后，后续

会继续讲解 Dubbo 的一些使用方法和原理剖析的知识，敬请期待。最后如果有想要本文完整 Demo 的读者可以在读者圈留下您的邮箱。

# GitChat