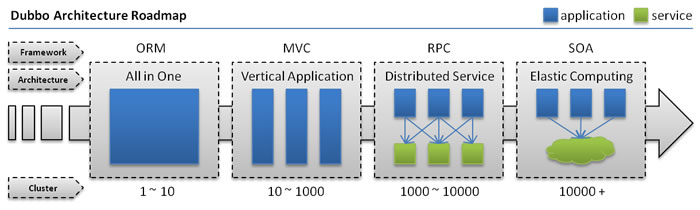
# 一 架构演变：



## 1 传统垂直MVC项目

1.1 垂直架构图

表现层

业务逻辑层

数据访问层

MySql

Oracal

三层系统的分层是结构

1.2 垂直架构缺点

随着业务的不断发展，应用规模越来越大，问题就越凸显：

牵一发而动全身；一处崩溃，全线崩盘

== 复杂应用的开发维护成本变高，一个功能出现问题整个系统就得重新打包，部署效率逐渐降低。

== 团队协作效率变成，公共功能重复开发，代码重复率太高。

== 系统可靠性变成，流量，负载均衡，数据库压力变大，因为在一进程中，如果出现内存溢出等故障，将导致整个节点崩溃，然后集群中的其他节点也会如此。（一处崩溃，全线崩盘）

== 维护和定制困难，无法随时拆分，修改一处，而动全身。

== 新功能上线周期变长。因为公共功能的变更导致测试工作量激增，因为重复代码多，一个地方修改需要同时修改多个地方，然后修改后继续测试。

== 新功能无法独立打包测试。需要和整个系统进行一起打包测试，出现bug会导致整个系统重新部署，强耦合导致效率低下。

## 2 RPC架构

2.1 RPC概述

RPC（Remote Procedure Call）远程过程调用协议，他是一种通过网络从计算机程序上请求服务，而不需要了解底层网络技术的协议。

RPC协议假定某些传输协议存在，如：TCP或UDP，为通信程序之间携带信息数据。在OSI网络通信模型中，RPC跨越了传输层和应用层。

RPC使的开发（包括网络分布式）多程序在内的应用程序更加容易。

RPC采用客户机/服务器模式。请求程序就是一个客户机，而服务提供程序就是服务器。

首先，客户机调用进程发送一个有进程参数的调用信息到服务进程，然后应答信息。

在服务器，进程保持睡眠状态直到调用信息到达为止。

当一个调用信息到达，服务器获得进程参数，计算结果，发送答复消息，然后等待下一个调用信息；

最后，客户端调用进程结果，然后调用执行继续进行。

client functions

client stub

sockets

server functions

server stub(skeleton)

sockets

1

2

3

8

network

10

9

5

6

7

4

client

server

ketnel

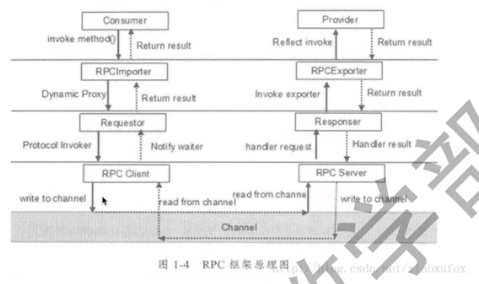
ketnel

2.2 RPC 架构分为三部分：

服务提供者：运行在服务器端，提供服务接口定义与服务实现类

服务消费者：运行在客户端，通过远程代理对象调用远程服务。

服务中心 ：运行在服务器端，负责将本地服务发布成远程服务，管理远程服务，提供给服务消费者使用。



2.3 RPC特点

== 远程服务提供者需要以某种形式提供服务的调用相关信息，包括但是不限于服务接口定义，数据结构，或者中间态的服务定义文件。服务调用者需要通过一定的途径获取远程服务的调用相关信息，例如：接口的定义jar包等。

== 远程代理对象：服务调用者调用的服务实际是远程服务的本地代理，对于java而言，他的实现就是JDK动态代理，通过动态代理拦截机制，将本地调用封装成远程服务调用。

== 通信：RPC框架与具体的协议无关，双方遵从约定好的即可，比如可以是Http invoke，可以使rmi invoke，也可以是其他任意的二进制压缩协议

== 序列化：远程通信，需要将对象转换成二进制数据进行网络传输，所以需要将数据序列化，不同的序列化框架支持的数据类型，数据包大小，异常类型或者性能不同。

== 不同RPC框架针对的场景不同，因此技术选择也个不相同，一些框架支持多种序列化框架，甚至支持用户自定义序列化框架。

2.4 RPC框架问题

== 在大规模服务化之前，应用可能只通过RPC框架简单的暴露和引用远程服务，通过配置的url地址进行远程调用，路由通过F5硬件负载均衡进行简单的负载均衡。

=== 当服务越来越多，服务的url越来越多，管理越来越困难，负载均衡单点压力变大，此时需要一个服务的注册中心，动态的注册和发现服务，是服务位置透明，消费者在本地缓存服务提供者列表，实现软负载均衡，可以降低对F5硬件负载均衡的依赖，降低硬件成本。

== 随着业务的发展，服务间的依赖关系变得错综复杂，甚至分不清哪个应用需要在哪个应能用之前启动，需要一个分布式消息跟踪系统可视化展示服务调用链

路，用于依赖分析，业务调用路径梳理，防止服务架构腐化。

== 服务的调用量越来越大，服务的容量问题就出现了，某个服务需要多少机器支撑，什么时候该加机器，服务上线容易下线难，上线的审批，下线的通知，需要统一的服务生命周期管理流程进行管控，不同的服务安全，权限不同，如何保证敏感数据服务不被误用，服务的访问安全策略如何制定。

== 服务化后随之而来的就是服务治理问题，纯粹的RPC框架服务治理能力都不强悍，需要通过服务框架+服务治理来完成。

解决问题：需要以下的东西

服务注册中心 – 负载均衡 – 链路追踪

出现了：SOA服务架构

2.5 常用RPC架构

Thrift

Hadoop的Avro-RPC

Hessian

gPRC

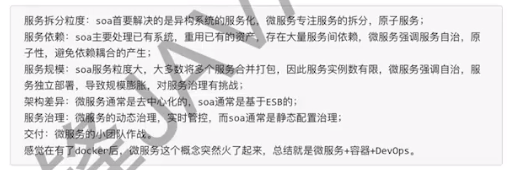
## 3 SOA服务化架构（其实就是RPC+服务治理中心）

## 4 微服务

MSA也是一种服务化架构风格，正流行，服务划分

== 原子服务，粒度细

== 独立部署，主要是容器

MSA与SOA的对比：

4.1 反向代理

正向代理：有明确的目的地：告诉VPN我要去谷歌--> VPN ->Google

反向代理：没有明确目的地，有结果就行。比如：搜索java 。🡪Google🡪由谷歌从多个服务器中找一个服务器搜索结果给你

== 正向代理：有明确的对接人，这件事必须让他来做。

== 反向代理：没有明确对接人，谁做都行，我只要结果。

反向代理：

需要服务

正向代理：

就要服务1

服务管理中心:

随机找一服务

服务1

服务2

服务3

我要哪个，给我哪个

给我哪个，我要哪个

4.3 解决负载均衡中随机访问服务器session不同频繁登录问题

== 设置让同一个请求只去某一个服务器（修改nginx轮询策略）

== 让session在服务器间共享（修改tomcat配置和项目的web.xml）

== 使用redis来存储服务端的 session，然后每个服务器用redis中的session对比请求带来的session

# 二 Dubbo介绍

## 1 什么是Dubbo

1.1一款分布式服务框架

1.2高性能和透明化的RPC远程服务调用方案

1.3 SOA服务治理方案

每天为2千多个服务提供大于30亿次访问量支持，并被广泛应用于阿里巴巴集团的各成员站点以及别的公司的业务中。

Dubbo就是资源调度和治理中心的管理工具。

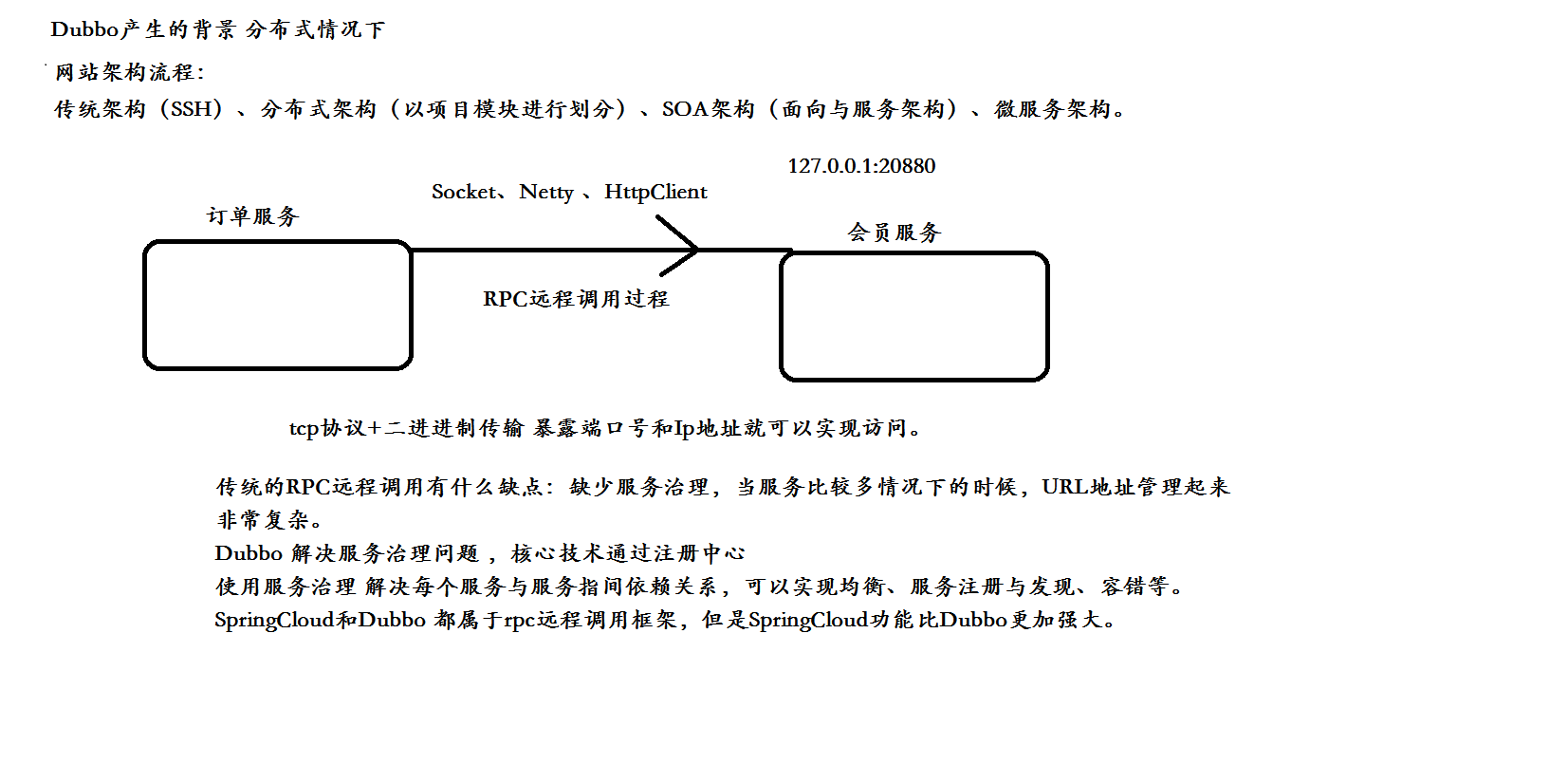
主要功能：类似于webservice的关于系统之间的通信的框架，并且可以统计和管理服务之间的调用情况。（被谁调用，调用次数，服务使用情况）

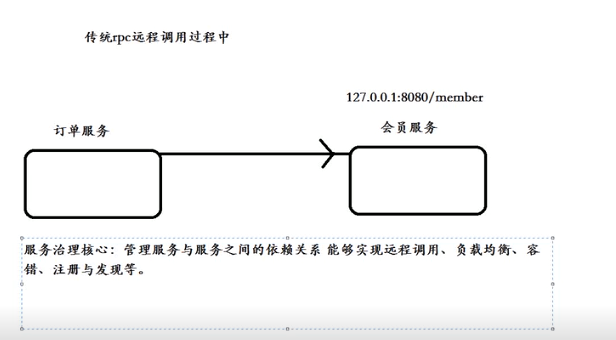
系统间通信

1 使用WebService：效率不高，他是基于SOAP协议（Http+xml）.

2 使用Resful形式的服务：http+json.很多项目中应用。如果服务越来越多，服务于服务之间的调用关系复杂，调用服务的Url管理复杂，什么时候添加机器难以确定

3 使用dubbo。使用rpc协议进行远程调用，直接使用socket通信，传输效率高，并且可以统计出系统之间的调用关系、调用次数、管理服务。





## 2 dubbo协议支持

Dubbo支持Dubbo、rmi、hessian、http、webservice、thrift、redis等多种协议，

但是Dubbo官网是推荐我们使用Dubbo协议的。

Dubbo RPC服务框架支持丰富的传输协议、序列化方式等通讯相关的配置和扩展。

dubbo执行一次RPC请求的过程大致如下：

1 消费者（Consumer）向注册中心（Registry）执行RPC请求，

2 注册中心分配服务URL并路由到具体服务提供方（Provider），

3 消费者和服务提供方建立网络连接，

4 服务提供方在本地创建连接池对象并提供远程服务，

对于长连接类型协议（如dubbo协议）将保持连接，减少握手认证，调用过程中可以避免频繁建立和断开连接导致的性能开销，保持长连接需要有心跳包的发送，所以对于非频繁调用的服务保持连接同样会有消耗。

更多关于dubbo详细介绍请参照官方文档（[http://alibaba.github.io/dubbo-doc-static/Home-zh.htm](http://alibaba.github.io/dubbo-doc-static/Home-zh.htm" \t "_blank)）。

1. 支持常见的传输协议：

（Dubbo、RMI）->TCP、（Http、Hessain、WebService）->Http等，

其中Dubbo和RMI协议基于TCP实现，

Hessian和WebService基于HTTP实现。

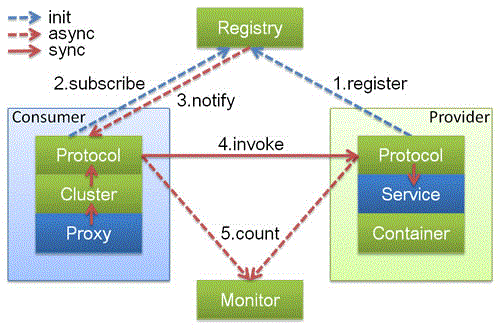
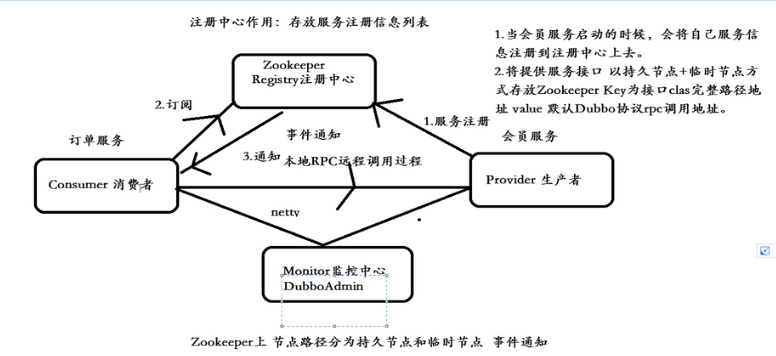
2、传输框架：Netty、Mina、以及基于servlet等方式。

3、序列化方式：dubbo、Hessian2、JSON（[fastjson](https://github.com/AlibabaTech/fastjson) 实现）、JAVA、SOAP 等。

　　基于dubbo 2.5.3框架，使用zookeeper作为dubbo服务注册中心，分别以单线程和多线程的方式测试以下方案：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Protocol** | **Transporter** | **Serialization** | **Remark** |
| **A** | dubbo 协议 | netty | hessian2 |  |
| **B** | dubbo 协议 | netty | dubbo |  |
| **C** | dubbo 协议 | netty | java |  |
| **D** | RMI 协议 | netty | java |  |
| **E** | RMI 协议 | netty | hessian2 |  |
| **F** | Hessian 协议 | servlet | hessian2 | Hessian，基于tomcat容器 |
| **G** | WebService 协议 | servlet | SOAP | CXF，基于tomcat容器 |

## 3 Dubbo架构原理



### 3.1 角色区分:

**1.Provider: 暴露服务的服务提供方（生产者）。**

**2.Consumer: 调用远程服务的服务消费方（消费者）。**

**3.Registry: 服务注册与发现的注册中心。**

**4.Monitor: 统计服务的调用次数和调用时间的监控中心。**

### 3.2 调用流程:

**0.服务容器负责启动，加载，运行服务提供者。**

**1.服务提供者在启动时，向注册中心注册自己提供的服务。**

**2.服务消费者在启动时，向注册中心订阅自己所需的服务。**

**3.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。**

**4.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。**

**5.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**

### 3.3 架构特点：

分别是连通性、健壮性、伸缩性、以及向未来架构的升级性。

## 4 Dubbo注册中心

对于服务提供方，它需要发布服务，而且由于应用系统的复杂性，服务的数量、类型也不断膨胀；

对于服务消费方，它最关心如何获取到它所需要的服务，而面对复杂的应用系统，需要管理大量的服务调用。

而且，对于服务提供方和服务消费方来说，他们还有可能兼具这两种角色，即既需要提供服务，又需要消费服务。

通过将服务统一管理起来，可以有效地优化内部应用对服务发布/使用的流程和管理。服务注册中心可以通过特定协议来完成服务对外的统一。

Dubbo提供的注册中心有如下几种类型可供选择：

Multicast注册中心

Zookeeper注册中心

Redis注册中心

Simple注册中心

## 5 Dubbo的优缺点

优点：

1 透明化的远程方法调用

- 像调用本地方法一样调用远程方法；只需简单配置，没有任何API侵入。

2 软负载均衡及容错机制

可在内网替代nginx lvs等硬件负载均衡器。

3 服务注册中心自动注册 & 配置管理

-不需要写死服务提供者地址，注册中心基于接口名自动查询提供者ip。

使用类似zookeeper等分布式协调服务作为服务注册中心，可以将绝大部分项目配置移入zookeeper集群。

4 服务接口监控与治理

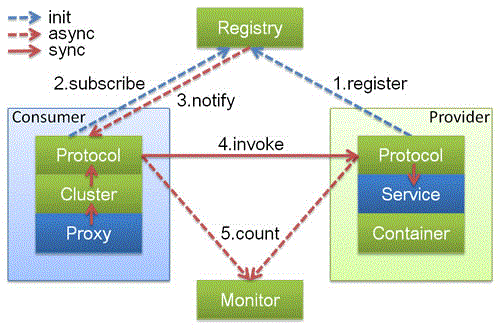
-Dubbo-admin与Dubbo-monitor提供了完善的服务接口管理与监控功能，针对不同应用的不同接口，可以进行多版本，多协议，多注册中心管理。

缺点：

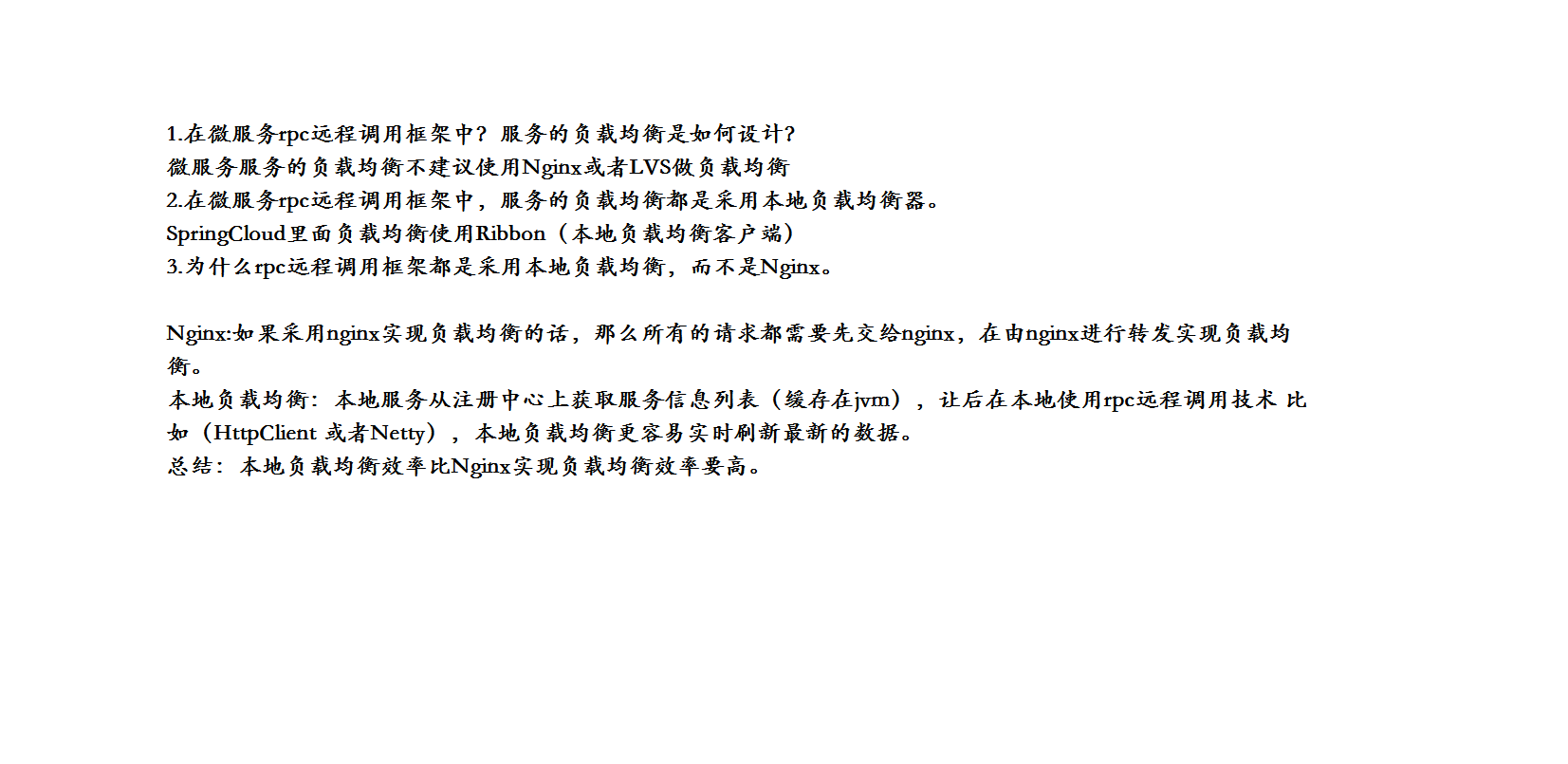
只支持JAVA语言

## 6 RPC负载均衡实现

### 6.1 负载均衡



在RPC远程调用框架中负载均衡大多数都采用本地负载均衡器，比如SpringCloud中Ribbon组件直接从注册中心上获取 服务列表信息，然后在本地实现RPC远程调用，同样的是Dubbo框架中的负载均衡也是采用本地负载均衡器。



### 6.2 软负载和硬负载区别

软负载:

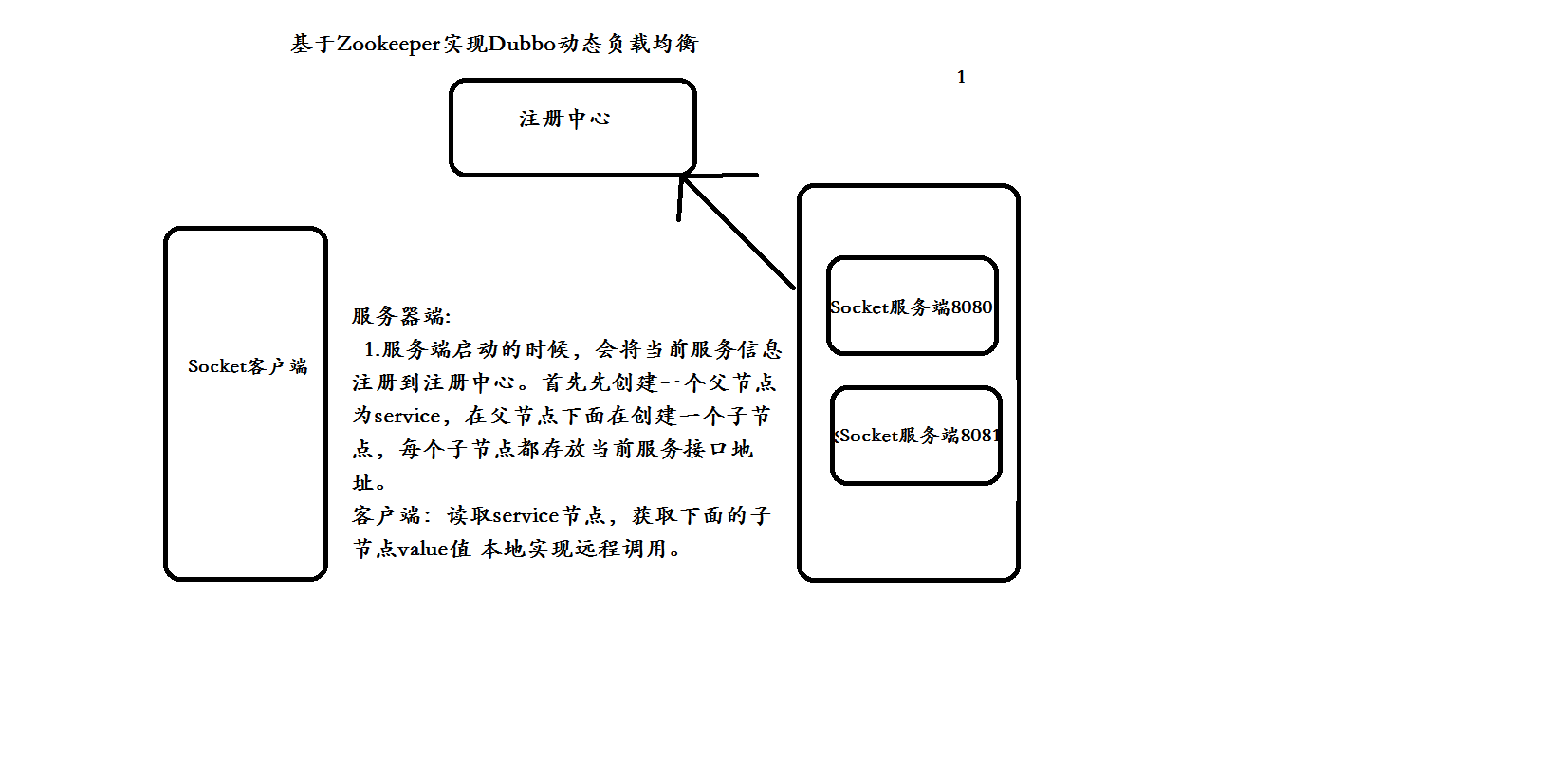
软件负载均衡通过服务器端上安装的负载均软件或者使用本地负载算法实现负载均衡功能。

例如:LVS、 Nginx VS Haproxy

硬负载:

F5负载均衡是硬件负载均衡的一种。硬件负载均衡，顾名思义，在服务器节点之间安装专门的硬件进行负载均衡的工作。

### 6.3 基于Zookeeper实现Dubbo动态负载均衡、



服务器端：

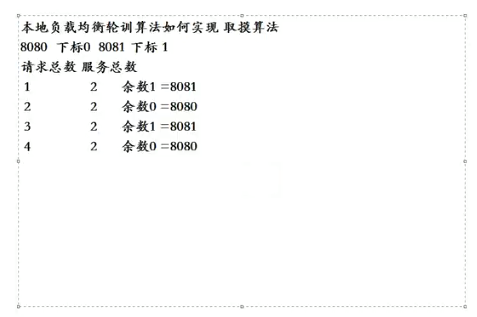
启动的服务的时候注册到，Zookeeper注册中心上，创建一个持久节点为service,在service持久节点下面创建多个不同的临时节点存放服务列表信息，临时节点内容存放服务调用地址。

客户端：

从Zookeeper节点上获取最新service持久节点下面服务节点信息，让后在本地使用负载均衡算法，随机分配调用远程服务。

如何实现动态负载均衡？

客户端采用事件监听service持久节下面节点是否发生变化，如果发生变化Zookeeper服务器端会及时的将最新的数据推送给Zookeeper客户端。



# 三 Dubbo快速入门

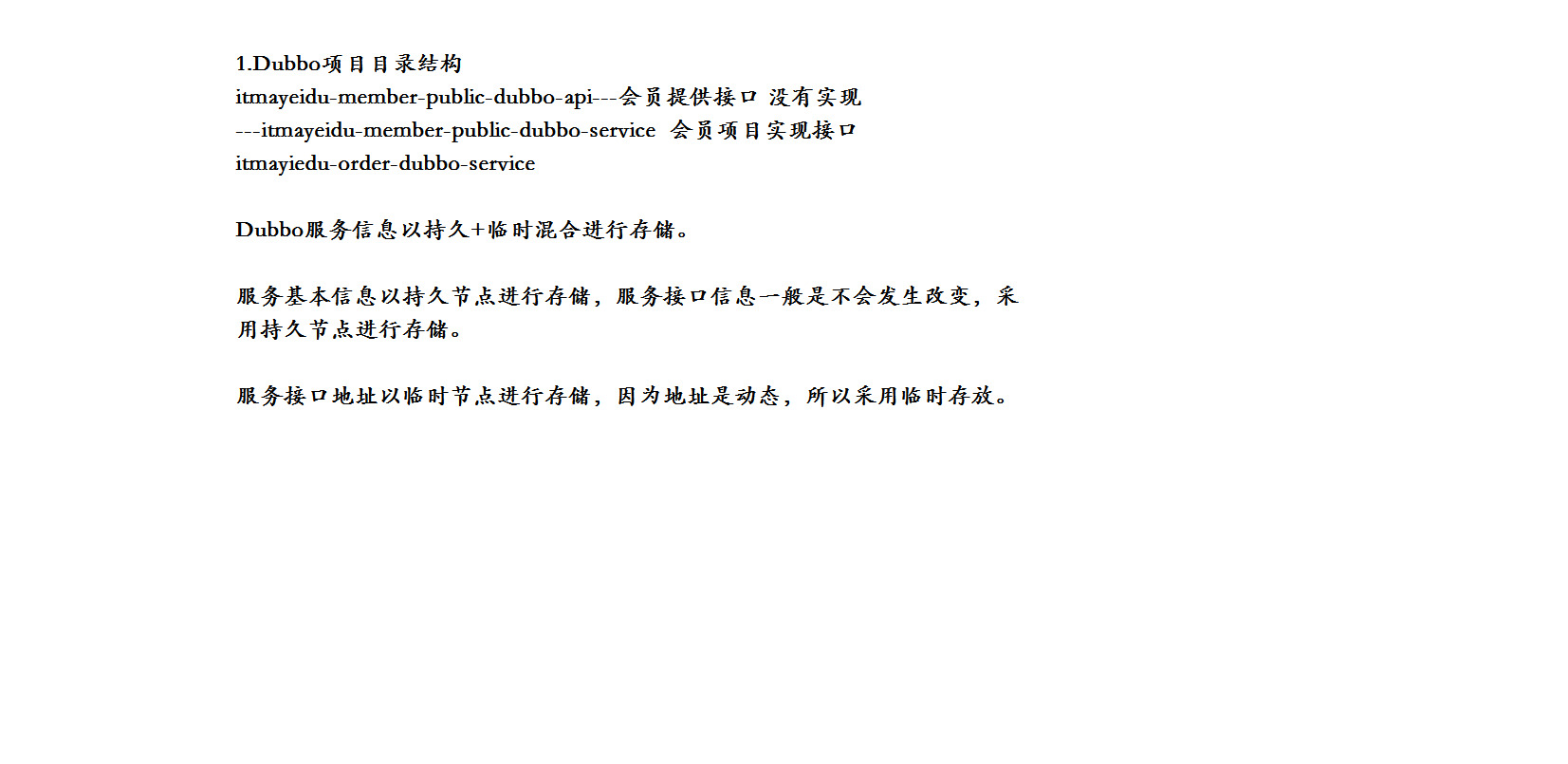
配置四大步：

你是谁？ 名字

来自哪？ ip+port+包

去哪里？ registry注册中心

干什么？ service/reference



## 1 一个项目需要的东西：

registry -zookeeper

provider -service

consumer -controller

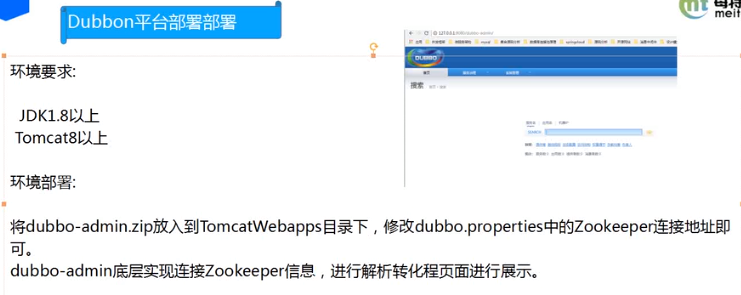
container -spring

monitor（监控、非必须）

环境步骤：

1. 安装Zookeepr启动
2. 创建Maven项目搭建生产者和消费者
3. 安装DubboAdmin平台，实现监控

将dubbo-admin.zip放入到TomcatWebapps目录下，修改dubbo.properties中的Zookeeper连接地址即可。



## 2 Zookeeper介绍与安装

Dubbo注册中心采用的是Zookeeper。

为什么采用Zookeeper呢？🡪服务的注册与发现

Zookeeper是一个分布式的服务框架，是树型的目录进行服务数据的存储，能做服务的自动注册和发现，并集群管理数据 ，这里能很好的作为Dubbo服务的注册中心。

Dubbo能与Zookeeper做到集群部署，当提供者出现断电等异常停机时，Zookeeper注册中心能自动删除提供者信息，当提供者重启时，能自动恢复注册数据，以及订阅请求

### 2.1 安装配置：

1 解压 **tar -xzvf zookeeper.tar.gz -C /usr/zookeeper (注意后边的文件夹，要写全路径，不然会说不存在该文件夹)**

2 修改配置文件 conf下的zoo\_simple.cfg 修改为zoo.cfg

3 zoo.cfg修改如下：先创建文件夹：data

修改dataDir= /usr/zookeeper/zookeeper-3.4.13/data

### 2.2 开启服务：

zkServer.sh start

确认是否开启：

zkServer.sh status

[root@localhost bin]# ./zkServer.sh status

ZooKeeper JMX enabled by default

Using config: /usr/zookeeper/zookeeper-3.4.13/bin/../conf/zoo.cfg

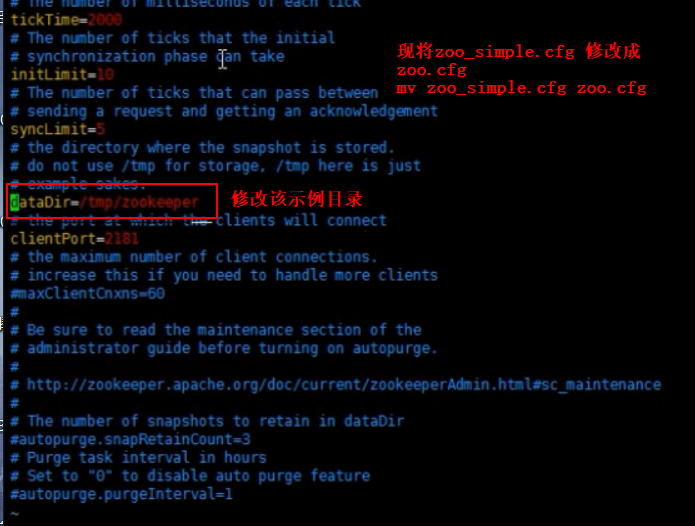
Mode: standalone 代表启动成功

如果无法访问：可设置防火墙开发端口

firewall-cmd --zone=public --add-port=2181/tcp --permanent  
 重启防火墙：

systemctl restart firewalld.service

设置：



## 3 生产者 provider

### 要做的事：

#### 1 知道注册中心的位置：

<!--配置注册中心  
 address 注册中心地址  
 protocol值得是注册中心的协议  
 -->  
 <dubbo:registry address="192.168.153.136:2181" protocol="zookeeper"></dubbo:registry>

#### 2 给自己起名字：

<!--配置本服务的别名：目的是在后台，看到本服务的别名，便于区分-->  
 <dubbo:application name="MyService"></dubbo:application>

#### 3 进行注册：告知自己的ip和端口

ip会在注册时自动获取：

<!--发布服务  
 interface 代表服务的类型（接口）  
 ref表示具体的发布类  
 timeout超时时间  
 -->  
 <dubbo:service interface="cn.htb.TestService" ref="testServiceImpl" timeout="6000"></dubbo:service>

告知自己的端口：

<!--配置服务端口  
 ip+端口：是确认唯一服务  
 name：是和zookeeper交互的中介名称  
 -->  
 <dubbo:protocol name="dubbo" port="12345"></dubbo:protocol>

### 1 pom.xml 文件：

<?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">  
 <parent>  
 <artifactId>dubbo</artifactId>  
 <groupId>cn.htb</groupId>  
 <version>1.0-SNAPSHOT</version>  
 </parent>  
 <modelVersion>4.0.0</modelVersion>  
  
 <artifactId>ServiceImpl</artifactId>  
 <packaging>war</packaging>  
  
 <name>cn.htb.ServiceImpl Maven Webapp</name>  
 <!-- *FIXME change it to the project's website* -->  
 <url>http://www.example.com</url>  
  
 <properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 <maven.compiler.source>1.7</maven.compiler.source>  
 <maven.compiler.target>1.7</maven.compiler.target>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.11</version>  
 <scope>test</scope>  
 </dependency>  
 <!—springmvc 依赖包 -->

<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-webmvc</artifactId>  
 <version>5.0.8.RELEASE</version>  
 </dependency>

<!—顶级父类 依赖包 -->  
 <dependency>  
 <groupId>cn.htb</groupId>  
 <artifactId>service</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 </dependency>

<!—日志文件依赖包，可能会无效，情况消费者 -->  
 <dependency>  
 <groupId>org.slf4j</groupId>  
 <artifactId>slf4j-api</artifactId>  
 <version>1.7.25</version>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <finalName>cn.htb.ServiceImpl</finalName>  
 <pluginManagement><!-- lock down plugins versions to avoid using Maven defaults (may be moved to parent pom) -->  
 <plugins>

<!—tomcat插件，来保证服务在不同的tomcat中，以指定的端口运行-->  
 <plugin>  
 <groupId>org.apache.tomcat.maven</groupId>  
 <artifactId>tomcat7-maven-plugin</artifactId>  
 <version>2.2</version>  
 <configuration>  
 <port>9000</port>  
 <path>/</path>  
 </configuration>  
 </plugin>  
 <plugin>  
 <artifactId>maven-clean-plugin</artifactId>  
 <version>3.1.0</version>  
 </plugin>  
 <!-- see http://maven.apache.org/ref/current/maven-core/default-bindings.html#Plugin\_bindings\_for\_war\_packaging -->  
 <plugin>  
 <artifactId>maven-resources-plugin</artifactId>  
 <version>3.0.2</version>  
 </plugin>  
 <plugin>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <version>3.8.0</version>  
 </plugin>  
 <plugin>  
 <artifactId>maven-surefire-plugin</artifactId>  
 <version>2.22.1</version>  
 </plugin>  
 <plugin>  
 <artifactId>maven-war-plugin</artifactId>  
 <version>3.2.2</version>  
 </plugin>  
 <plugin>  
 <artifactId>maven-install-plugin</artifactId>  
 <version>2.5.2</version>  
 </plugin>  
 <plugin>  
 <artifactId>maven-deploy-plugin</artifactId>  
 <version>2.8.2</version>  
 </plugin>  
 </plugins>  
 </pluginManagement>  
 </build>  
</project>

### 2 Dubbo配置文件：

<?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.xsd  
 http://code.alibabatech.com/schema/dubbo http://code.alibabatech.com/schema/dubbo/dubbo.xsd  
 ">  
 <!--扫描组件-->  
 <context:component-scan base-package="cn.htb"></context:component-scan>  
 <!--使用dubbo的另一外一个原因，可以和spring无缝连接-->  
 <!--目的：  
 找到注册中心，告诉他我是谁，我的端口是多少  
 -->  
 <!--配置本服务的别名：目的是在后台，看到本服务的别名，便于区分-->  
 <dubbo:application name="MyService"></dubbo:application>  
 <!--配置注册中心  
 address 注册中心地址  
 protocol值得是注册中心的协议  
 -->  
 <dubbo:registry address="192.168.153.136:2181" protocol="zookeeper"></dubbo:registry>  
   
 <!--发布服务  
 interface 代表服务的类型（接口）  
 ref表示具体的发布类  
 timeout超时时间  
 -->  
 <dubbo:service interface="cn.htb.TestService" ref="testServiceImpl" timeout="6000"></dubbo:service>  
  
 <!--配置服务端口  
 ip+端口：是确认唯一服务  
 name：是和zookeeper交互的中介名称  
 -->  
 <dubbo:protocol name="dubbo" port="12345"></dubbo:protocol>  
</beans>

### 3 log4j.properties

log4j.rootLogger=info, stdout  
  
log4j.appender.stdout=org.apache.log4j.ConsoleAppender  
log4j.appender.stdout.Target=System.out  
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout  
log4j.appender.stdout.layout.ConversionPattern=%d %p [%c] - %m%n

### 4 web.xml

<!DOCTYPE web-app PUBLIC  
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"  
 "http://java.sun.com/dtd/web-app\_2\_3.dtd" >  
  
<web-app>  
 <display-name>Archetype Created Web Application</display-name>  
 <!-- spring 容器加载配置文件 -->  
 <context-param>  
 <param-name>contextConfigLocation</param-name>  
 <param-value>classpath\*:applicationContext.xml</param-value>  
 </context-param>  
 <!-- 启动监听器，启动加载文件 -->  
 <listener>  
 <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  
 </listener>  
</web-app>

### 5 service实现类

@Service("testServiceImpl")  
public class ServiceImpl implements TestService {  
 @Override  
 public String getName(String name) {  
 return "你输入的数据是： "+name;  
 }  
}

## 4 消费者consumer

### 要做的事：

#### 1 知道注册中心在哪

<!--配置注册中心  
 address 注册中心地址  
 protocol值得是注册中心的协议  
 -->  
 <dubbo:registry address="192.168.153.136:2181" protocol="zookeeper"></dubbo:registry>

#### 2 给自己取名字、

<!--配置本服务的别名：目的是在后台，看到本服务的别名，便于区分-->  
 <dubbo:application name="MyController"></dubbo:application>

#### 3 告知自己需要什么

<!--  
 指定需要的服务 不写id可能会出错  
 -->  
 <dubbo:reference interface="cn.htb.TestService" id="teseServiceImpl"></dubbo:reference>

### 1 pom.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">  
 <parent>  
 <artifactId>dubbo</artifactId>  
 <groupId>cn.htb</groupId>  
 <version>1.0-SNAPSHOT</version>  
 </parent>  
 <modelVersion>4.0.0</modelVersion>  
  
 <artifactId>webcontroller</artifactId>  
 <packaging>war</packaging>  
  
 <name>webcontroller Maven Webapp</name>  
 <!-- *FIXME change it to the project's website* -->  
 <url>http://www.example.com</url>  
  
 <properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 <maven.compiler.source>1.7</maven.compiler.source>  
 <maven.compiler.target>1.7</maven.compiler.target>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.11</version>  
 <scope>test</scope>  
 </dependency>  
 <!-- springmvc 依赖，如果父级有此依赖可以不加 -->  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-webmvc</artifactId>  
 <version>5.0.8.RELEASE</version>  
 </dependency>  
 <!-- 依赖的服务接口 jar包 -->  
 <dependency>  
 <groupId>cn.htb</groupId>  
 <artifactId>service</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 </dependency>  
 <!-- dubbo依赖jar包，版本号不同，可能会出现不同问题，此版本号没有问题，如果父级有此依赖，可以不加 -->  
 <dependency>  
 <groupId>com.alibaba</groupId>  
 <artifactId>dubbo</artifactId>  
 <version>2.5.9</version>  
 <!--排除依赖，因为本文件中已经加入了spring5.0.8-->  
 <exclusions>  
 <exclusion>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring</artifactId>  
 </exclusion>  
 </exclusions>  
 </dependency>  
 <!-- zookeeper客户端依赖包 -->  
 <dependency>  
 <groupId>com.101tec</groupId>  
 <artifactId>zkclient</artifactId>  
 <version>0.11</version>  
 </dependency>  
 <dependency>  
 <groupId>org.slf4j</groupId>  
 <artifactId>slf4j-api</artifactId>  
 <version>1.7.25</version>  
 </dependency>  
 <!-- 如果上述的日志文件无效，则打开该文件进行运行，之后再重新使用上边那个 -->  
 <!--<dependency>-->  
 <!--<groupId>org.apache.logging.log4j</groupId>-->  
 <!--<artifactId>log4j-core</artifactId>-->  
 <!--<version>2.3</version>-->  
 <!--</dependency>-->  
 <!--<dependency>-->  
 <!--<groupId>log4j</groupId>-->  
 <!--<artifactId>log4j</artifactId>-->  
 <!--<version>1.2.17</version>-->  
 <!--</dependency>-->  
 <!--<dependency>-->  
 <!--<groupId>org.apache.logging.log4j</groupId>-->  
 <!--<artifactId>log4j-api</artifactId>-->  
 <!--<version>2.10.0</version>-->  
 <!--</dependency>-->  
 </dependencies>  
  
 <build>  
 <finalName>webcontroller</finalName>  
 <pluginManagement><!-- lock down plugins versions to avoid using Maven defaults (may be moved to parent pom) -->  
 <!--tomcat插件，可以让项目已制定的端口运行在不同的tomcat中-->  
 <plugins>  
 <plugin>  
 <groupId>org.apache.tomcat.maven</groupId>  
 <artifactId>tomcat7-maven-plugin</artifactId>  
 <version>2.2</version>  
 <configuration>  
 <port>8000</port>  
 <path>/</path>  
 </configuration>  
 </plugin>  
 <plugin>  
 <artifactId>maven-clean-plugin</artifactId>  
 <version>3.1.0</version>  
 </plugin>  
 <!-- see http://maven.apache.org/ref/current/maven-core/default-bindings.html#Plugin\_bindings\_for\_war\_packaging -->  
 <plugin>  
 <artifactId>maven-resources-plugin</artifactId>  
 <version>3.0.2</version>  
 </plugin>  
 <plugin>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <version>3.8.0</version>  
 </plugin>  
 <plugin>  
 <artifactId>maven-surefire-plugin</artifactId>  
 <version>2.22.1</version>  
 </plugin>  
 <plugin>  
 <artifactId>maven-war-plugin</artifactId>  
 <version>3.2.2</version>  
 </plugin>  
 <plugin>  
 <artifactId>maven-install-plugin</artifactId>  
 <version>2.5.2</version>  
 </plugin>  
 <plugin>  
 <artifactId>maven-deploy-plugin</artifactId>  
 <version>2.8.2</version>  
 </plugin>  
 </plugins>  
 </pluginManagement>  
 </build>  
</project>

### 2 springmvc-servlet.xml

<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:mvc="http://www.springframework.org/schema/mvc"  
 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.xsd  
 http://www.springframework.org/schema/mvc  
 http://www.springframework.org/schema/context/spring-mvc.xsd  
 http://code.alibabatech.com/schema/dubbo  
 http://code.alibabatech.com/schema/dubbo/dubbo.xsd  
 ">  
 <!--乱码处理-->  
 <!--<mvc:annotation-driven>-->  
 <!--<mvc:message-converters>-->  
 <!--<bean class="org.springframework.http.converter.StringHttpMessageConverter">-->  
 <!--<property name="supportedMediaTypes">-->  
 <!--<list>-->  
 <!--<value>application/json;charset=UTF-8</value>-->  
 <!--</list>-->  
 <!--</property>-->  
 <!--</bean>-->  
 <!--</mvc:message-converters>-->  
 <!--</mvc:annotation-driven>-->  
  
 <!--扫描组件-->  
 <context:component-scan base-package="cn.htb"></context:component-scan>  
 <!--使用dubbo的另一外一个原因，可以和spring无缝连接-->  
 <!--目的：  
 找到注册中心，告诉他我是谁，我的端口是多少  
 -->  
 <!--配置本服务的别名：目的是在后台，看到本服务的别名，便于区分-->  
 <dubbo:application name="MyController"></dubbo:application>  
 <!--配置注册中心  
 address 注册中心地址  
 protocol值得是注册中心的协议  
 -->  
 <dubbo:registry address="192.168.153.136:2181" protocol="zookeeper"></dubbo:registry>  
   
 <!--  
 指定需要的服务  
 -->  
 <dubbo:reference interface="cn.htb.TestService" id="teseServiceImpl"></dubbo:reference>  
</beans>

服务消费者：

//配置本服务的名称

<dubbo:application name=”orderService”>

//配置注册中心

<dubbo:registry address=”192.168.153.136:2181” protocol=”zookeeper”>

//告知服务中心要什么

id不写，可能会出问题

<dubbo:reference interface=”cn.htb.TestService” id=”testService”>

### 3 日志文件如上

### 4 web.xml

<!DOCTYPE web-app PUBLIC  
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"  
 "http://java.sun.com/dtd/web-app\_2\_3.dtd" >  
  
<web-app>  
 <!--<filter>-->  
 <!--<filter-name>characterEncodingFilter</filter-name>-->  
 <!--<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>-->  
 <!--<init-param>-->  
 <!--<param-name>encoding</param-name>-->  
 <!--<param-value>UTF-8</param-value>-->  
 <!--</init-param>-->  
 <!--<init-param>-->  
 <!--<param-name>forceEncoding</param-name>-->  
 <!--<param-value>true</param-value>-->  
 <!--</init-param>-->  
 <!--</filter>-->  
 <!--<filter-mapping>-->  
 <!--<filter-name>characterEncodingFilter</filter-name>-->  
 <!--<url-pattern>/\*</url-pattern>-->  
 <!--</filter-mapping>-->  
  
 <servlet>  
 <servlet-name>springmvc</servlet-name>  
 <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  
 <init-param>  
 <param-name>contextConfigLocation</param-name>  
 <param-value>classpath\*:springmvc-servlet.xml</param-value>  
 </init-param>  
 <load-on-startup>2</load-on-startup>  
 </servlet>  
 <servlet-mapping>  
 <servlet-name>springmvc</servlet-name>  
 <url-pattern>\*.action</url-pattern>  
 </servlet-mapping>  
  
  
</web-app>

### 5 controller

package cn.htb;  
  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.web.bind.annotation.RequestMapping;  
  
import javax.annotation.Resource;  
  
@Controller  
public class MyController {  
 @Resource  
 private TestService testServiceImpl;  
  
 @RequestMapping("getName")  
 public String getName(String name,Model model){  
 String data = testServiceImpl.getName(name);  
 model.addAttribute("name",data);  
 System.*out*.println(data);  
 return "jso.jsp";  
 }  
}

## 5 别的示例

### 5.1 创建Maven工程

项目结构：

主要分三大模块：

Itmayiedu-dubbo-api : 存放公共接口；

Itmayiedu-dubbo-consumer :　调用远程服务；

Itmayiedu-dubbo-provider : 提供远程服务。

### 5.2 Itmayiedu-dubbo-api 新建接口

|  |
| --- |
| **public** **interface** DemoApiService {  **public** String getUser(Long userId);  } |

### 5.3 生产者提供服务接口

#### 5.3.1 itmayiedu-dubbo-provider 实现接口业务逻辑

|  |
| --- |
| **public** **class** DemoApiServiceImpl **implements** DemoApiService {  **public** String getUser(Long userId) {  System.***out***.println("生产者调用消费者服务接口userId:" + userId);  **return** "yushengjun";  }  } |

#### 5.3.2 Maven依赖信息

|  |
| --- |
| <dependencies>  <dependency>  <groupId>com.itmayiedu</groupId>  <artifactId>itmayiedu-dubbo-api</artifactId>  <version>0.0.1-SNAPSHOT</version>  </dependency>  <dependency>  <groupId>com.101tec</groupId>  <artifactId>zkclient</artifactId>  <version>0.10</version>  </dependency>  <dependency>  <groupId>commons-logging</groupId>  <artifactId>commons-logging</artifactId>  <version>1.2</version>  </dependency>  <dependency>  <groupId>org.jboss.netty</groupId>  <artifactId>netty</artifactId>  <version>3.2.5.Final</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  <version>4.3.9.RELEASE</version>  </dependency>  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>dubbo</artifactId>  <version>2.5.3</version>  <exclusions>  <exclusion>  <groupId>org.springframework</groupId>  <artifactId>spring</artifactId>  </exclusion>  <exclusion>  <groupId>org.jboss.netty</groupId>  <artifactId>netty</artifactId>  </exclusion>  </exclusions>  </dependency>  </dependencies> |

#### 5.3.3 新增 dubbo-provider.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"*>  <!--定义了提供方应用信息，用于计算依赖关系；在 dubbo-admin 或 dubbo-monitor 会显示这个名字，方便辨识 -->  <dubbo:application name=*"demotest-provider"* />  <!--使用 zookeeper 注册中心暴露服务，注意要先开启 zookeeper -->  <dubbo:registry address=*"zookeeper://localhost:2181"* />  <!-- 用dubbo协议在20880端口暴露服务 -->  <dubbo:protocol name=*"dubbo"* port=*"20880"* />  <!--使用 dubbo 协议实现定义好的 api.PermissionService 接口 -->  <dubbo:service interface=*"com.itmayiedu.demo.api.service.DemoApiService"*  ref=*"demoService"* protocol=*"dubbo"* />  <!--具体实现该接口的 bean -->  <bean id=*"demoService"* class=*"com.itmayiedu.demo.api.service.impl.DemoApiServiceImpl"* />  </beans> |

#### 5.4.4 启动发布服务

|  |
| --- |
| ClassPathXmlApplicationContext context = **new** ClassPathXmlApplicationContext("provider.xml");  System.***out***.println(context.getDisplayName() + ": here");  context.start();  System.***out***.println("服务已经启动...");  System.***in***.read(); |

### 5.4 消费者调用服务接口

#### 5.4.1 Maven依赖信息

|  |
| --- |
| <dependencies>  <dependency>  <groupId>com.itmayiedu</groupId>  <artifactId>itmayiedu-dubbo-api</artifactId>  <version>0.0.1-SNAPSHOT</version>  </dependency>  <dependency>  <groupId>com.101tec</groupId>  <artifactId>zkclient</artifactId>  <version>0.10</version>  </dependency>  <dependency>  <groupId>commons-logging</groupId>  <artifactId>commons-logging</artifactId>  <version>1.2</version>  </dependency>  <dependency>  <groupId>org.jboss.netty</groupId>  <artifactId>netty</artifactId>  <version>3.2.5.Final</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  <version>4.3.9.RELEASE</version>  </dependency>  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>dubbo</artifactId>  <version>2.5.3</version>  <exclusions>  <exclusion>  <groupId>org.springframework</groupId>  <artifactId>spring</artifactId>  </exclusion>  <exclusion>  <groupId>org.jboss.netty</groupId>  <artifactId>netty</artifactId>  </exclusion>  </exclusions>  </dependency>  </dependencies> |

#### 5.4.2 调用服务接口

|  |
| --- |
| ClassPathXmlApplicationContext context = **new** ClassPathXmlApplicationContext("consumer.xml");  context.start();  System.***out***.println("consumer start");  DemoApiService demoApiService = context.getBean(DemoApiService.**class**);  String result = demoApiService.getUser(1l);  System.***out***.println("result:" + result); |

#### 5.4.3 消费者配置文件

# 四SpringBoot整合Dubbo

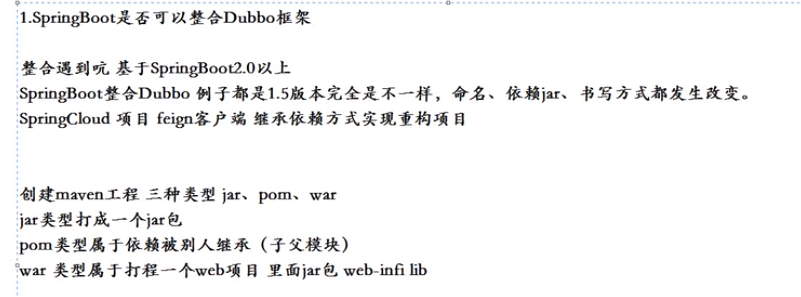
## 1 注意事项：

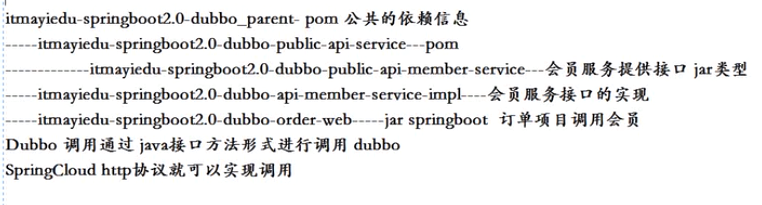
SpringBoot1.0版本与SpringBoot2.0整合版本区别很大。

@Service 使用阿里巴巴注解注入服务容器

@Reference 获取服务注解

@EnableDubbo 开启Dubbo服务





itmayiedu-dubbo-parent

--------------itmayiedu-dubbo-api

------------------------itmayiedu-dubbo-api-member

--------------itmayiedu-dubbo-api-member-impl

--------------itmayiedu-dubbo-api-order-impl

## 2 根项目 itmayiedu-dubbo-parent

### Maven依赖信息

|  |
| --- |
| <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2.0.1.RELEASE</version>  </parent>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>  <java.version>1.8</java.version>  <curator-framework.version>4.0.1</curator-framework.version>  <zookeeper.version>3.4.13</zookeeper.version>  <dubbo.starter.version>0.2.0</dubbo.starter.version>  </properties>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <dependency>  <groupId>com.alibaba.boot</groupId>  <artifactId>dubbo-spring-boot-starter</artifactId>  <version>${dubbo.starter.version}</version>  </dependency>  <dependency>  <groupId>org.apache.curator</groupId>  <artifactId>curator-framework</artifactId>  <version>${curator-framework.version}</version>  </dependency>  <dependency>  <groupId>org.apache.zookeeper</groupId>  <artifactId>zookeeper</artifactId>  <version>${zookeeper.version}</version>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build> |

## 3 生产者

itmayiedu-dubbo-api-member-impl

### 3.1 Maven依赖信息

|  |
| --- |
| <dependencies>  <dependency>  <groupId>com.itmayiedu</groupId>  <artifactId>itmayiedu-dubbo-api-member</artifactId>  <version>0.0.1-SNAPSHOT</version>  </dependency>  </dependencies> |

### 3.2 application.yml

|  |
| --- |
| server:  port: 8080  dubbo:  application:  name: member  protocol:  name: dubbo  port: 20880  registry:  address: zookeeper://127.0.0.1:2181  scan:  base-packages: com.itmayiedu.api.member.impl |

### 3.3 发布服务

|  |
| --- |
| @Service  **public** **class** MemberServiceImpl **implements** MemberService {  @Override  **public** String getUser() {  **return** "订单服务调用会员服务接口";  }  } |

注意:@Service 注解引入为阿里巴巴注解

### 3.4 启动服务

|  |
| --- |
| @EnableDubbo  @SpringBootApplication  **public** **class** AppMember {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(AppMember.**class**, args);  }  } |

@EnableDubbo 开启Dubbo服务功能

## 4 消费者

itmayiedu-dubbo-api-order-impl

### 4.1 Maven依赖信息

|  |
| --- |
| <dependencies>  <dependency>  <groupId>com.itmayiedu</groupId>  <artifactId>itmayiedu-dubbo-api-member</artifactId>  <version>0.0.1-SNAPSHOT</version>  </dependency>  </dependencies> |

### 4.2 application.yml

|  |
| --- |
| server:  port: 8081  ###dubbo 注册服务名称  dubbo:  application:  name: order  ###dubbo服务地址  protocol:  name: dubbo  port: 20880  registry:  address: zookeeper://localhost:2181  consumer:  timeout: 5000 |

### 4.3 调用服务

|  |
| --- |
| @Reference  **private** MemberService memberService;  @RequestMapping("/orderToMember")  **public** String orderToMember() {  **return** memberService.getUser();  } |

使用该 @Reference调用dubbo接口

### 4.4启动项目

|  |
| --- |
| @EnableDubbo  @SpringBootApplication  **public** **class** AppOrder {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(AppOrder.**class**, args);  }  } |

## 5 扩充：

### 5.1 服务提供者配置：

从配置中获取内容：该service注解是Dubbo中的service注解，不是spring中的注解，目的是创建该对象并发布为服务

@Service(version=”1.0”,application=”${dubbo.application.name}”,protocol=”${dubbo.protocol.id}”,registry=”${dubbo.registry.address}”)

### 5.2 消费者：

服务注入使用注解：不再使用@Resource 或@Autowired

public class TestController{

@Reference（version=”1.0” application=””）

private TestService testService;

}

启动类：

包扫描

@SpringBootApplication(scanBasePackages={“cn.htb”})

public class SpringApp{

public static void main(String[] args){

SpringApplicaton.run(SpringApp.class);

}

}

# 五 SpringCloud与Dubbo区别

为什么放弃Dubbo 使用SpringCloud？

相同点：SpringCloud 和Dubbo可以实现RPC远程调用框架，可以实现服务治理。

不同点:

SpringCloud是一套目前比较网站微服务框架了，整合了分布式常用解决方案遇到了问题注册中心Eureka、负载均衡器Ribbon ，客户端调用工具Rest和Feign，分布式配置中心Config，服务保护Hystrix，网关Zuul Gateway ，服务链路Zipkin，消息总线Bus等。

从架构上分析

Dubbo内部实现功能没有SpringCloud（全家桶）强大，目前来说SpringCloud是比较完善的微服务框架。

Dubbo核心只是实现服务治理，缺少分布式解决方案的整合（如：配置中心、追踪链路、网关、消息总线等），如果需要用到这些组件，需要整合其他框架。

从更新迭代速度分析

Dubbo目前更新速度没有SpringCloud快，到SpringCloud2.0后SpringCloud会越来完善和稳定。

从开发背景角度分析

Dubbo的开发背景是阿里巴巴， 在中国也推出了非常多的优秀的开源框架

但是在SpringCloud的背景是Spring家族，Spring是专注于企业级开源框架开发，在中国，或者在整个世界上Spring框架都应用的非常广泛。所有相对来说SpringCloud的背景比Dubbo更加强大。

最后总结下：

如果学习Dubbo的话，学习其他的分布式解决方案需要自己组装，反而如果学习SpringCloud，它已经把整个常用分布式解决都整合好了。

# 六 当当网Dubbox框架

支持REST风格远程调用（HTTP + JSON/XML)：基于非常成熟的JBoss RestEasy框架，在dubbo中实现了REST风格（HTTP + JSON/XML）的远程调用，以显著简化企业内部的跨语言交互，同时显著简化企业对外的Open API、无线API甚至AJAX服务端等等的开发。事实上，这个REST调用也使得Dubbo可以对当今特别流行的“微服务”架构提供基础性支持。 另外，REST调用也达到了比较高的性能，在基准测试下，HTTP + JSON与Dubbo 2.x默认的RPC协议（即TCP + Hessian2二进制序列化）之间只有1.5倍左右的差距，详见文档中的基准测试报告。

详细文档:https://github.com/dangdangdotcom/dubbox

