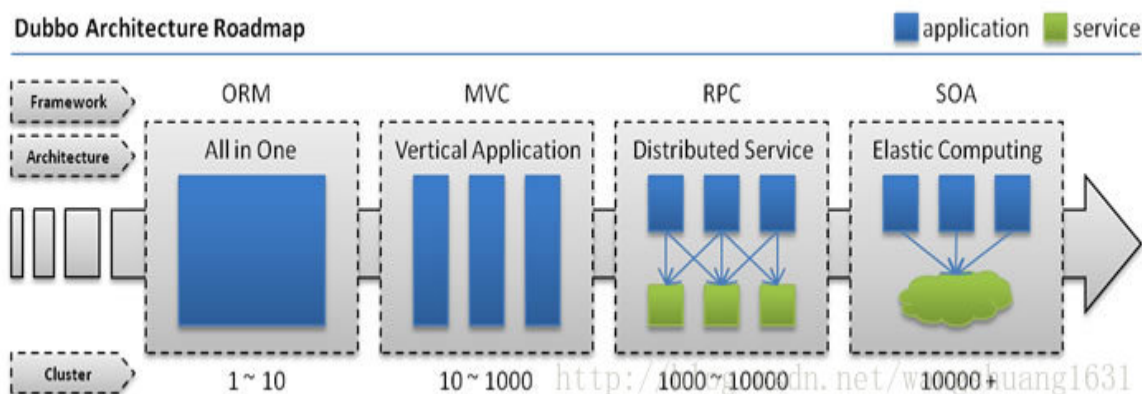


Dubbo 服务治理框架

Dubbo 概述

Dubbo 的背景

随着互联网的发展，网站应用的规模不断扩大，常规的垂直应用架构已无法应对，分布式服务架构以及流动计算架构势在必行，亟需一个治理系统确保架构有条不紊的演进。



- 单一应用架构

当网站流量很小时，只需一个应用，将所有功能都部署在一起，以减少部署节点和成本。此时，用于简化增删改查工作量的 数据访问框架 (ORM) 是关键。

- 垂直应用架构

当访问量逐渐增大，单一应用增加机器带来的加速度越来越小，将应用拆成互不相干的几个应用，以提升效率。

此时，用于加速前端页面开发的 Web 框架 (MVC) 是关键。

- 分布式服务架构

当垂直应用越来越多，应用之间交互不可避免，将核心业务抽取出来，作为独立的服务，逐渐形成稳定的服务中心，使前端应用能更快速的响应多变的市场需求。

此时，用于提高业务复用及整合的 分布式服务框架 (RPC) 是关键。

- 流动计算架构

当服务越来越多，容量的评估，小服务资源的浪费等问题逐渐显现，此时需增加一个调度中心基于访问压力实时管理集群容量，提高集群利用率。

此时，用于提高机器利用率的 资源调度和治理中心 (SOA) 是关键。

什么是 Dubbo

Dubbo 是一个分布式服务框架，致力于提供高性能和透明化的 RPC 远程服务调用方案，SOA 服务治理方案。简单的说，dubbo 就是个服务框架，如果没有分布式的需求，其实是不需要用的，只有在分布式的时候，才有 dubbo 这样的分布式服务框架的需求，并且本质上是个服务调用，说白了就是个远程服务调用的分布式框架。告别 Web Service 模式中的 wsdl，以服务者与消费者的方式在 dubbo 上注册)。

其核心部分包含：

1. 远程通讯：提供对多种基于长连接的 NiO 框架抽象封装，包括多种线程模型，序列化，以及“请求一响应”模式的信息交换方式。
2. 集群容错：提供基于接口方法的透明远程过程调用，包括多协议支持。以及负载均衡，失败容错，地址路由，动态配置等集群支持。
3. 自动发现：基于注册中心目录服务，使用服务消费能动态查找服务提供方，使地址透明，使用服务提供方可以平滑增加或减少服务器

Dubbo 能做什么



问题：

服务的 URL 管理非常困难(rmi://、http:*、)、F5 负载均衡器的单点压力(硬件成本)

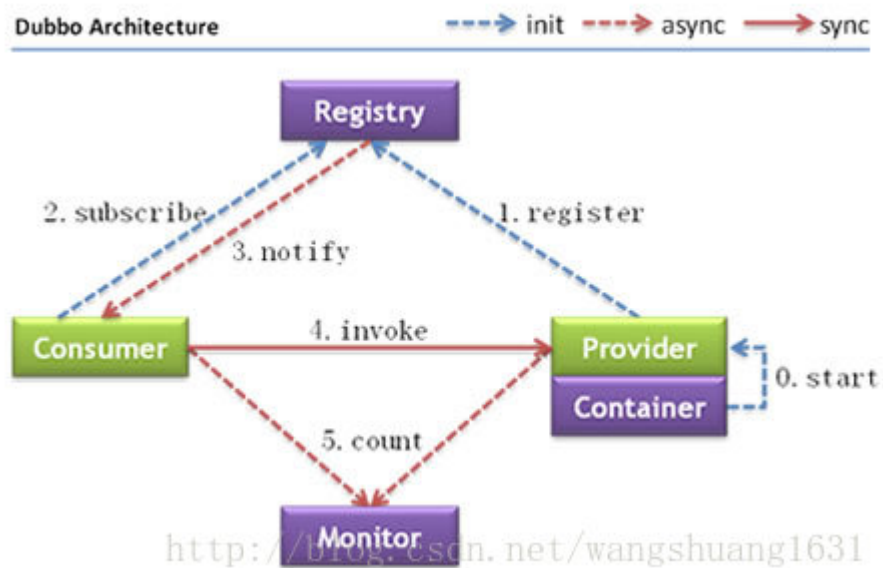
各个服务之间依赖管理非常复杂

各个服务之间如何进行监控

1. 透明化的远程方法调用，就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 F5 等硬件负载均衡器，降低成本，减少单点。
3. 服务自动注册与发现，不再需要写死服务提供方地址，注册中心基于接口名查询服务提供者的 IP 地址，并且能够平滑添加或删除服务提供者。
4. Dubbo 采用全 Spring 配置方式，透明化接入应用，对应用没有任何 API 侵入，只需用 Spring 加载 Dubbo 的配置即可，Dubbo 基于 Spring 的 Schema 扩展进行加载。



Dubbo 架构



节点角色说明:

Provider: 暴露服务的提供方。

Consumer: 调用远程服务的消费方。

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。 马了的

Container: 服务器容器 kən'teɪnə(r)

调用关系说明:

服务容器负责启动、加载、运行服务提供者。

服务提供者在启动时，向注册中心注册自己提供的服务。

服务消费者在启动时，向注册中心订阅自己所需的服务。

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。

- 连通性:

注册中心负责服务地址的注册与查找，相当于目录服务，服务提供者和消费者只在启动时与注册中心交互，注册中心不转发请求，压力较小

监控中心负责统计各服务调用次数，调用时间等，统计先在内存汇总后每分钟一次发送到监控中心服务器，并以报表展示

服务提供者向注册中心注册其提供的服务，并汇报调用时间到监控中心，此时间不包含网络开销
服务消费者向注册中心获取服务提供者地址列表，并根据负载算法直接调用提供者，同时汇报调用时间到监控中心，此时间包含网络开销

注册中心，服务提供者，服务消费者三者之间均为长连接，监控中心除外

注册中心通过长连接感知服务提供者的存在，服务提供者宕机，注册中心将立即推送事件通知消费者

注册中心和监控中心全部宕机，不影响已运行的提供者和消费者，消费者在本地缓存了提供者列表

注册中心和监控中心都是可选的，服务消费者可以直连服务提供者

健壮性:

监控中心宕掉不影响使用，只是丢失部分采样数据

数据库宕掉后，注册中心仍能通过缓存提供服务列表查询，但不能注册新服务

注册中心对等集群，任意一台宕掉后，将自动切换到另一台

注册中心全部宕掉后，服务提供者和服务消费者仍能通过本地缓存通讯

服务提供者无状态，任意一台宕掉后，不影响使用

服务提供者全部宕掉后，服务消费者应用将无法使用，并无限次重连等待服务提供者恢复

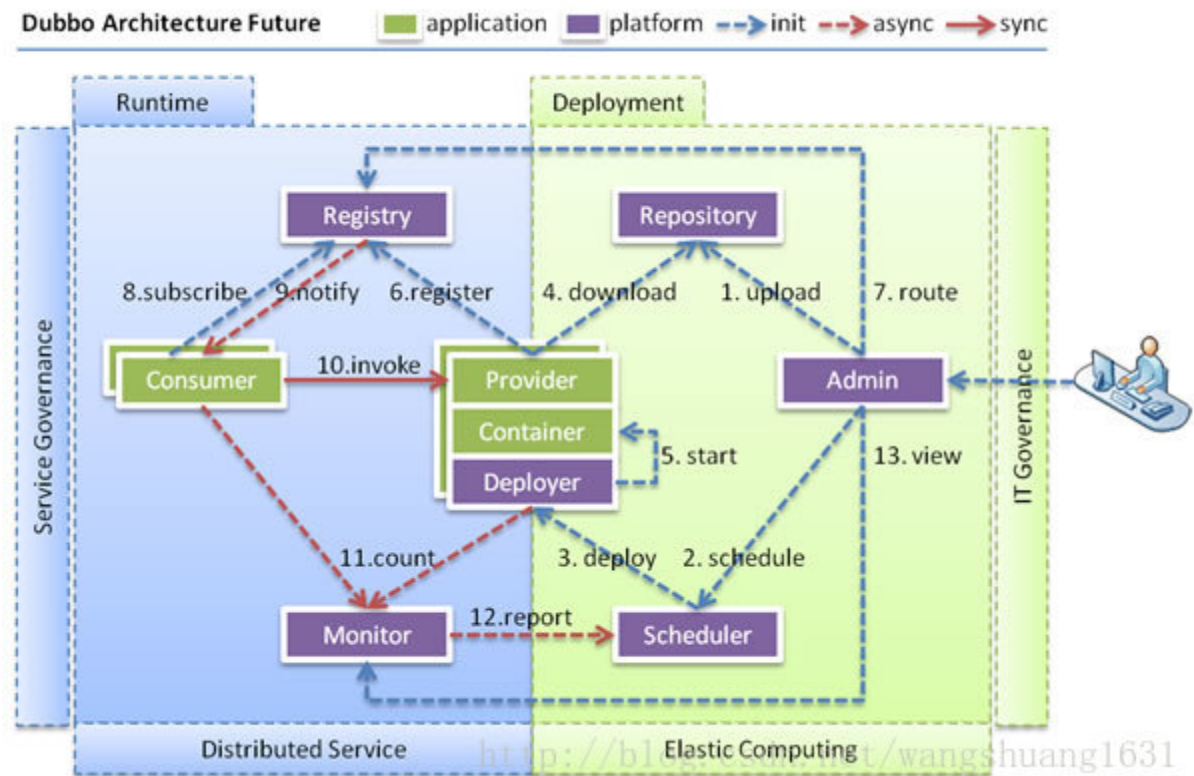
伸缩性:

注册中心为对等集群，可动态增加机器部署实例，所有客户端将自动发现新的注册中心

服务提供者无状态，可动态增加机器部署实例，注册中心将推送新的服务提供者信息给消费者

升级性:

当服务集群规模进一步扩大，带动 IT 治理结构进一步升级，需要实现动态部署，进行流动计算，现有分布式服务架构不会带来阻力：



Deployer：自动部署服务的本地代理。

Repository：仓库用于存储服务应用发布包。

Scheduler：调度中心基于访问压力自动增减服务提供者。

Admin：统一管理控制台。

Register：注册中心

Producer：生产者

Consumer：消费者

Subscribe: 订阅

Notify: 通知

Invoke: 调用

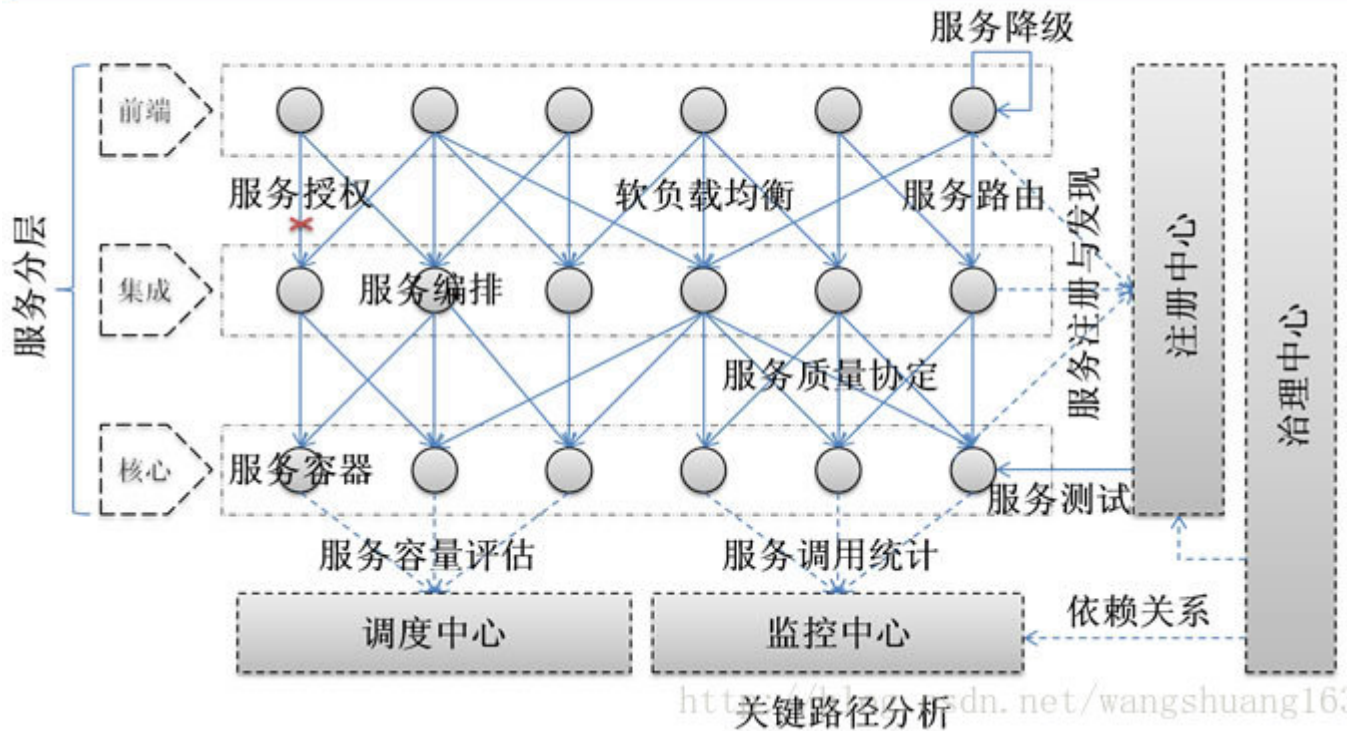
MONITOR: 监控

Container 容器

Eureka：SpringCloud 注册中心

Dubbo 服务治理

Dubbo 服务治理



在大规模服务化之前，应用可能只是通过 RMI 或 Hessian 等工具，简单的暴露和引用远程服务，通过配置服务的 URL 地址进行调用，通过 F5 等硬件进行负载均衡。

(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。

此时需要一个服务注册中心，动态的注册和发现服务，使服务的位置透明。

并通过在消费方获取服务提供方地址列表，实现软负载均衡和 Failover，降低对 F5 硬件负载均衡器的依赖，也能减少部分成本。

(2) 当进一步发展，服务间依赖关系变得错综复杂，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。

这时，需要自动画出应用间的依赖关系图，以帮助架构师理清关系。

(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？

为了解决这些问题，第一步，要将服务现在每天的调用量，响应时间，都统计出来，作为容量规划的参考指标。

其次，要可以动态调整权重，在线上，将某台机器的权重一直加大，并在加大的过程中记录响应时间的变化，直到响应时间到达阈值，记录此时的访问量，再以此访问量乘以机器数反推总容量。

Dubbo 环境搭建

启动 zookeeper

使用 zookeeper 作为服务注册中心

创建 itmayiedu-interface 工程提供会员接口

创建 itmayiedu-interface 工程提供服务接口

```
//会员服务接口
public interface UserService {
    //使用userId查询 用户信息
    public String getUser(Long userId);
}
```

创建 itmayiedu-member-provider 工程生产者

创建 itmayiedu-member-provider 工程提供服务接口,生产者主要发布服务.

Maven 依赖参数

```
<dependencies>
  <dependency>
    <groupId>com.itmayiedu</groupId>
    <artifactId>itmayiedu-interface</artifactId>
    <version>0.0.1-SNAPSHOT</version>
  </dependency>
  <dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>dubbo</artifactId>
    <version>2.5.6</version>
  </dependency>
  <!-- 添加zk客户端依赖 -->
  <dependency>
    <groupId>com.github.sgroschupf</groupId>
    <artifactId>zkclient</artifactId>
```

```
<version>0.1</version>

</dependency>

</dependencies>
```

定义一个 Service 实现服务接口

```
public class UserServiceImpl implements UserService {

    public String getUser(Long userId) {

        System.out.println("###会员服务接受参数开始userId:" + userId);

        if (userId == 1) {

            return "余胜军";

        }

        if (userId == 2) {

            return "张杰";

        }

        System.out.println("###会员服务接受参数结束###");

        return "未找到用户...";

    }

}
```

发布 Dubbo 服务

新建配置文件 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 ">

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

    <!-- 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29014端口暴露服务 -->
<dubbo:protocol name="dubbo" port="29014" />
<!-- 声明需要暴露的服务接口 -->
<dubbo:service interface="com.itmayiedu.service.UserService"
    ref="userService" />
<!-- 具体的实现bean -->
<bean id="userService" class="com.itmayiedu.member.service.impl.UserServiceImpl" />
</beans>
```

启动 Dubbo 服务

```
// 启动会员服务
public class MemberServer {

    public static void main(String[] args) throws IOException {
        ClassPathXmlApplicationContext applicationContext = new ClassPathXmlApplicationContext("provider.xml");
        applicationContext.start();
        System.out.println("会员服务启动成功");
        System.in.read();
    }
}
```

创建 itmayiedu-order-consumer 工程消费者

Maven 依赖参数

```
<dependencies>
    <dependency>
        <groupId>com.itmayiedu</groupId>
        <artifactId>itmayiedu-interface</artifactId>
        <version>0.0.1-SNAPSHOT</version>
    </dependency>
    <dependency>
        <groupId>com.alibaba</groupId>
        <artifactId>dubbo</artifactId>
        <version>2.5.6</version>
    </dependency>
<!-- 添加zk客户端依赖 -->
```

```
<dependency>
    <groupId>com.github.sgroschupf</groupId>
    <artifactId>zkclient</artifactId>
    <version>0.1</version>
</dependency>
</dependencies>
```

消费会员服务接口

```
public class OrderService {
    public static void addOrder() {
        ClassPathXmlApplicationContext applicationContext = new ClassPathXmlApplicationContext("consumer.xml");
        applicationContext.start();
        System.out.println("###order服务,开始调用会员服务");
        UserService userService=(UserService) applicationContext.getBean("userService");
        String userName = userService.getUser(11);
        System.out.println("###order服务,结束调用会员服务,userName:" + userName);
    }
    public static void main(String[] args) {
        addOrder();
    }
}
```

配置文件参数

```
<?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
http://www.springframework.org/schema/beans/spring-beans.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="consumer" />

    <!-- 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" />

    <!-- 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.itmayiedu.service.UserService" />
</beans>
```

Dubbo 支持哪些协议?

Dubbo 协议

Dubbo 缺省协议采用单一长连接和 NIO 异步通讯, 适合于小数据量大并发的服务调用, 以及服务消费者机器数远大于服务提供者机器数的情况。Dubbo 缺省协议不适合传送大数据量的服务, 比如传文件, 传视频等, 除非请求量很低。

Hessian 协议

Hessian 协议用于集成 Hessian 的服务, Hessian 底层采用 Http 通讯, 采用 Servlet 暴露服务, Dubbo 缺省内嵌 Jetty 作为服务器实现。Hessian 是 Caucho 开源的一个 RPC 框架: <http://hessian.caucho.com>, 其通讯效率高于 WebService 和 Java 自带的序列化。

基于 Hessian 的远程调用协议:

连接个数: 多连接

连接方式: 短连接

传输协议: HTTP

传输方式: 同步传输

序列化: Hessian 二进制序列化

适用范围: 传入传出参数数据包较大, 提供者比消费者个数多, 提供者压力较大, 可传文件。

适用场景: 页面传输, 文件传输, 或与原生 hessian 服务互操作

HTTP 协议

此协议采用 spring 的 HttpInvoker 的功能实现,

连接个数: 多个

连接方式: 长连接

连接协议: http

传输方式: 同步传输

序列化: 表单序列化

适用范围: 传入传出参数数据包大小混合, 提供者比消费者个数多, 可用浏览器查看, 可用表单或 URL 传入参数, 暂不支持传文件。

适用场景: 需同时给应用程序和浏览器 JS 使用的服务。

RMI 协议

采用 JDK 标准的 `java.rmi.*` 实现, 采用阻塞式短连接和 JDK 标准序列化方式

Java 标准的远程调用协议:

连接个数：多连接

连接方式：短连接

传输协议：TCP

传输方式：同步传输

序列化：Java 标准二进制序列化

适用范围：传入传出参数数据包大小混合，消费者与提供者个数差不多，可传文件。

适用场景：常规远程服务方法调用，与原生 RMI 服务互操作

Dubbo-admin 管理平台搭建

步骤：

将 dubbo-admin.zip 解压到 webapps 目录下

修改 dubbo.properties zk 注册中心连接地址连接信息

启动 tomcat 即可



Dubbo 集群、负载均衡、容错

步骤

修改配置文件：

provider.xml 端口号 `<dubbo:protocol name="dubbo" port="29015" />`

启动两个服务。



Dubbo

什么是 Dubbox?

Dubbox 环境搭建

生产者环境搭建

定义一个接口

```
@Path("/user")
public interface UserService {
    @GET
    @Path("/getUser/{id : \\d+}")
    @Consumes({ MediaType.APPLICATION_JSON })
    public String getUser(@PathParam("id") Integer id);
}
```

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: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 ">

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

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

    <!-- 用rest协议在8080端口暴露服务 -->
```

```
<dubbo:protocol name="rest" port="8081" />
<!-- 声明需要暴露的服务接口 -->
<dubbo:service interface="com.itmayiedu.service.UserService" ref="userService"
/>

<!-- 和本地bean一样实现服务 -->
<bean id="userService" class="com.itmayiedu.service.impl.UserServiceImpl" />
</beans>
```

测试访问地址

<http://127.0.0.1:8081/user/getUser/2>

消费者环境搭建

Java 代码

```
@Path("/user")
public interface UserService {
    @GET
    @Path("/getUser/{id : \\d+}")
    @Consumes({ MediaType.APPLICATION_JSON })
    public String getUser(@PathParam("id") Integer id);
}
```

dubbo-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"
    xmlns:util="http://www.springframework.org/schema/util"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
        http://code.alibabatech.com/schema/dubbo
        http://code.alibabatech.com/schema/dubbo/dubbo.xsd
        http://www.springframework.org/schema/util
        http://www.springframework.org/schema/util/spring-util-3.0.xsd">
```

```
<!-- 提供方应用信息，用于计算依赖关系 -->
<dubbo:application name="consumer" />
<dubbo:registry address="zookeeper://127.0.0.1:2181" />
<!-- 生成远程服务代理，可以像使用本地bean -->
<dubbo:reference interface="com.itmayiedu.service.UserService"
    id="userService" check="false" />
</beans>
```

Doubbox 依赖

```
<dependencies>
    <dependency>
        <groupId>com.alibaba</groupId>
        <artifactId>dubbo</artifactId>
        <version>2.8.4</version>
    </dependency>
    <!-- 添加zk客户端依赖 -->
    <dependency>
        <groupId>com.github.sgroschupf</groupId>
        <artifactId>zkclient</artifactId>
        <version>0.1</version>
    </dependency>
    <dependency>
        <groupId>com.itmayiedu</groupId>
        <artifactId>itmayiedu-dubbo-service</artifactId>
        <version>0.0.1-SNAPSHOT</version>
    </dependency>
    <dependency>
        <groupId>org.jboss.resteasy</groupId>
        <artifactId>resteasy-jaxrs</artifactId>
        <version>3.0.7.Final</version>
    </dependency>
    <dependency>
        <groupId>org.jboss.resteasy</groupId>
        <artifactId>resteasy-client</artifactId>
        <version>3.0.7.Final</version>
    </dependency>
    <dependency>
        <groupId>javax.validation</groupId>
        <artifactId>validation-api</artifactId>
        <version>1.0.0.GA</version>
    </dependency>
</dependencies>
```



```
<!-- 如果要使用json序列化 -->
<dependency>
    <groupId>org.jboss.resteasy</groupId>
    <artifactId>resteasy-jackson-provider</artifactId>
    <version>3.0.7.Final</version>
</dependency>

<!-- 如果要使用xml序列化 -->
<dependency>
    <groupId>org.jboss.resteasy</groupId>
    <artifactId>resteasy-jaxb-provider</artifactId>
    <version>3.0.7.Final</version>
</dependency>

<!-- 如果要使用netty server -->
<dependency>
    <groupId>org.jboss.resteasy</groupId>
    <artifactId>resteasy-netty</artifactId>
    <version>3.0.7.Final</version>
</dependency>

<!-- 如果要使用Sun HTTP server -->
<dependency>
    <groupId>org.jboss.resteasy</groupId>
    <artifactId>resteasy-jdk-http</artifactId>
    <version>3.0.7.Final</version>
</dependency>

<!-- 如果要使用tomcat server -->
<dependency>
    <groupId>org.apache.tomcat.embed</groupId>
    <artifactId>tomcat-embed-core</artifactId>
    <version>8.0.11</version>
</dependency>
<dependency>
    <groupId>org.apache.tomcat.embed</groupId>
    <artifactId>tomcat-embed-logging-juli</artifactId>
    <version>8.0.11</version>
</dependency>
<dependency>
    <groupId>com.esotericsoftware.kryo</groupId>
    <artifactId>kryo</artifactId>
    <version>2.24.0</version>
</dependency>
```

```
<dependency>
    <groupId>de.javakaffee</groupId>
    <artifactId>kryo-serializers</artifactId>
    <version>0.26</version>
</dependency>
<dependency>
    <groupId>de.ruedigermoeller</groupId>
    <artifactId>fst</artifactId>
    <version>1.55</version>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-core</artifactId>
    <version>2.3.3</version>
</dependency>
<dependency>
    <groupId>org.mortbay.jetty</groupId>
    <artifactId>jetty</artifactId>
    <version>7.0.0.pre5</version>
</dependency>
</dependencies>
```

常见 Dubbo 面试题

什么是 Dubbo?

Dubbo是一个RPC远程调用框架，分布式服务治理框架

什么是Dubbo服务治理？

服务与服务之间会有很多个Url、依赖关系、负载均衡、容错、自动注册服务。

Dubbo 有哪些协议？

默认用的dubbo协议、Http、RMI、Hessian

Dubbo 整个架构流程

分为四大模块

生产者、消费者、注册中心、监控中心

生产者：提供服务

消费者: 调用服务

注册中心:注册信息(redis、zk)

监控中心:调用次数、关系依赖等。

首先生产者将服务注册到注册中心 (zk) , 使用zk持久节点进行存储, 消费订阅zk节点, 一旦有节点变更, zk通过事件通知传递给消费者, 消费可以调用生产者服务。

服务与服务之间进行调用, 都会在监控中心中, 存储一个记录。

Dubbox 与 Dubbo 区别?

Dubbox使用http协议+rest风格传入json或者xml格式进行远程调用。

Dubbo使用Dubbo协议。

RPC 远程调用框架

SpringCloud、dubbo、Dubbox、thint、Hessian...

Rpc 其实就是远程调用, 服务与服务之间相互进行通讯。

目前主流 用 http+json

SpringCloud 与 Dubbo 区别?

相同点:

dubbo 与 springcloud 都可以实现 RPC 远程调用。

dubbo 与 springcloud 都可以使用分布式、微服务场景下。

区别:

dubbo 有比较强的背景,在国内有一定影响力。

dubbo 使用 zk 或 redis 作为注册中心

springcloud 使用 eureka 作为注册中心

dubbo 支持多种协议, 默认使用 dubbo 协议。

Springcloud 只能支持 http 协议。

Springcloud 是一套完整的微服务解决方案。

Dubbo 目前已经停止更新, SpringCloud 更新速度快。

常见错误

在启动 dubbo-admin 时我们会先启动 zookeeper, 如果项目跑到 zkclient.ZkEventThread -

Starting ZkClient event thread. 就不走了，那么就是 zookeeper 没有跑起来，如果你在 zookeeper 的目录下运行了 `./zookeeper-3.3.6/bin/zkServer.sh start` 还是这样的话，那么请执行 `./zookeeper-3.3.6/bin/zkServer.sh status` 命令看看 zookeeper 是否跑起来了，如果没跑起来很可能是因为你 zookeeper 下的 conf 目录没有 zoo.cfg

解决办法：吧 conf 目录下的 zoo_sample.cfg 复制一份为 zoo.cfg.

