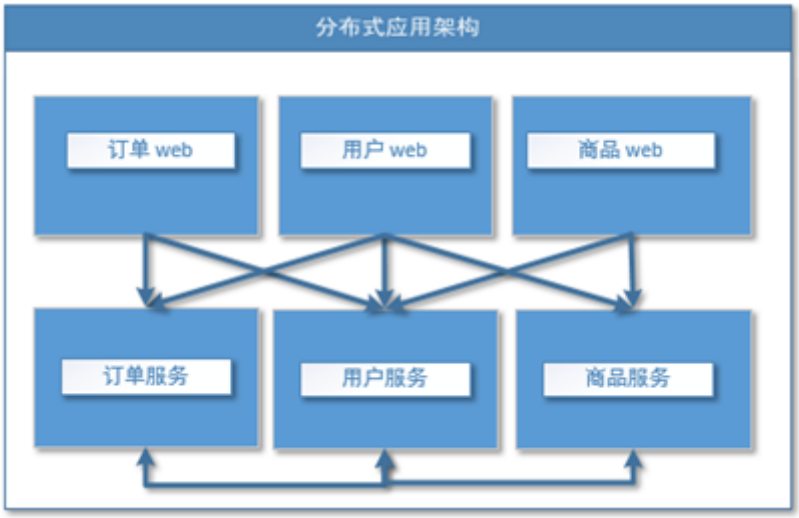
dubbo笔记（1）基础知识

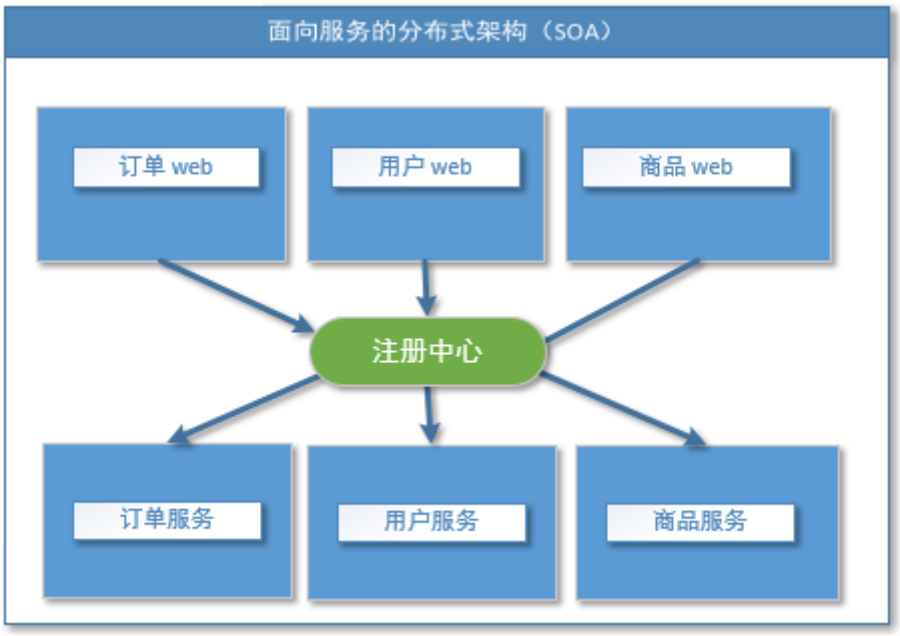
一、SOA和RPC介绍

1、SOA

随着互联网的发展，应用规模不断扩大，应用之间的交互不可避免，这时将核心业务抽取出来，作为独立的服务，用于提高业务复用及整合，逐渐形成了分布式服务架构。

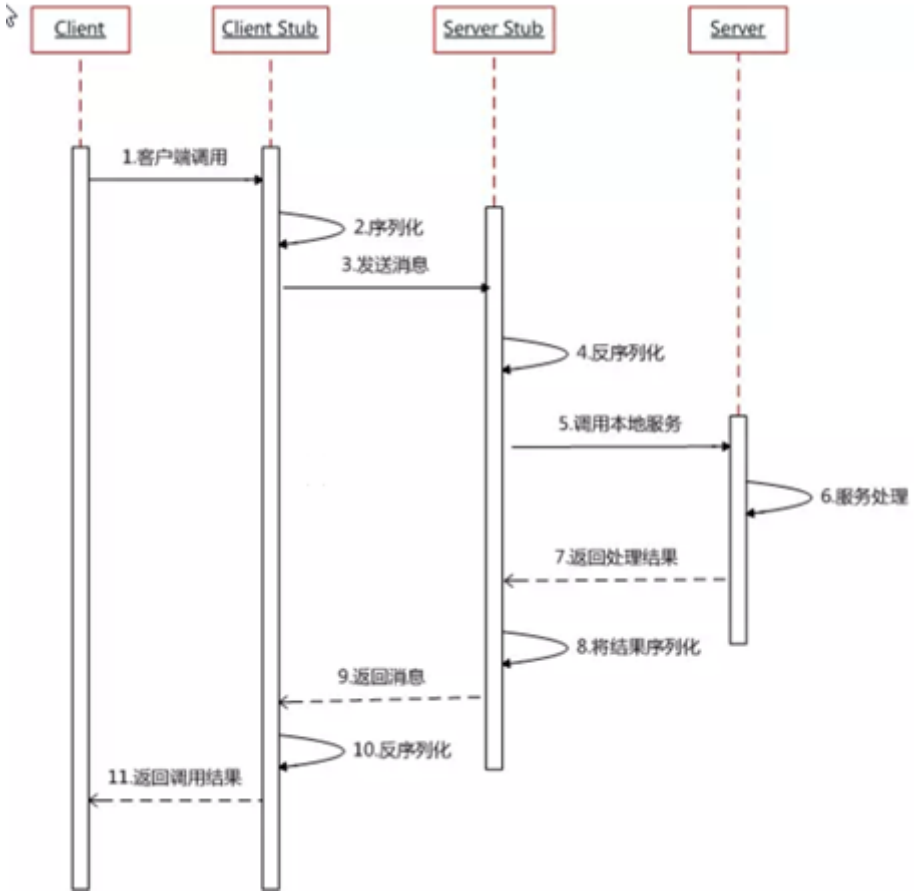


当服务越来越多，容量的评估，小服务资源的浪费等问题逐渐显现，此时需增加一个调度中心基于访问压力实时管理集群容量，提高集群利用率。此时，用于**提高机器利用率的资源调度和治理中心(SOA)[Service Oriented Architecture]**是关键。



2、RPC

RPC【Remote Procedure Call】是指远程过程调用，是一种进程间通信方式，他是一种技术的思想，而不是规范。它允许程序调用另一个地址空间（通常是共享网络的另一台机器上）的过程或函数，而不用程序员显式编码这个远程调用的细节。即程序员无论是调用本地的还是远程的函数，本质上编写的调用代码基本相同。



> RPC两个核心模块，通讯和序列化。
>
> 序列化要求传输的消息（参数和返回值）必须实现java.io.Serializable接口。

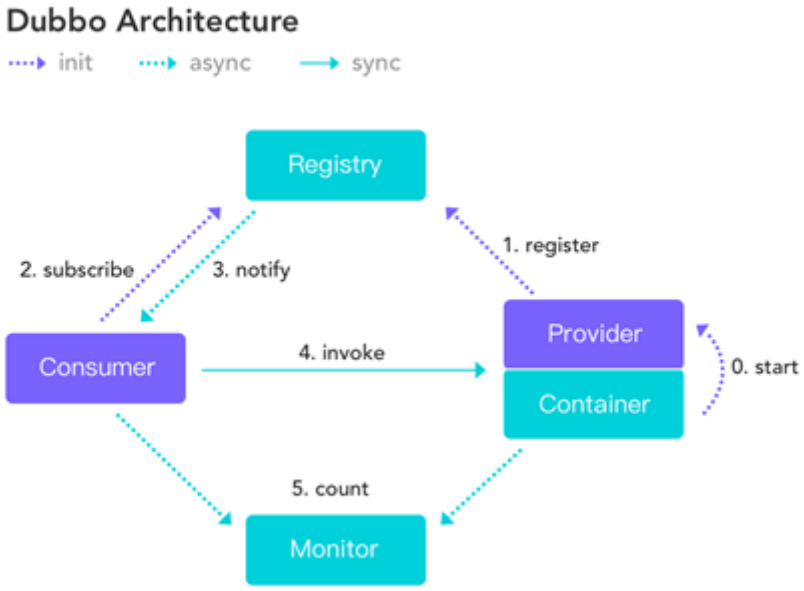
二、dubbo核心概念

1、简介

Apache Dubbo (incubating) |ˈdʌbəʊ| 是一款高性能、轻量级的开源Java RPC框架，它提供了三大核心能力：面向接口的远程方法调用，智能容错和负载均衡，以及服务自动注册和发现。

> 官网：http://dubbo.apache.org/

2、基本概念



服务提供者（Provider）：暴露服务的服务提供方，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（Consumer）：调用远程服务的服务消费方，服务消费者在启动时，向注册中心订阅自己所需的服务，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。

注册中心（Registry）：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

监控中心（Monitor）：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

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。

三、dubbo环境搭建

1、安装zookeeper【linux】

1.1、安装zookeeper

① 下载zookeeper

> 网址 https://archive.apache.org/dist/zookeeper/zookeeper-3.6.1/

② 解压

```
grx@ubuntu:~/Downloads$ tar -zxvf apache-zookeeper-3.6.1-bin.tar.gz
```

③ 移动到指定位置

```
grx@ubuntu:~/Downloads$ mv apache-zookeeper-3.6.1-bin /usr/local/
```

1.2、配置zookeeper

① 初始化zookeeper配置文件

拷贝/usr/local/zookeeper/conf/zoo_sample.cfg，到同一个目录下改个名字叫zoo.cfg

```
grx@ubuntu:/usr/local/apache-zookeeper-3.6.1-bin/conf$ cp zoo_sample.cfg zoo.cfg
```

② 启动zookeeper

```
grx@ubuntu:/usr/local/apache-zookeeper-3.6.1-bin/bin$ bash zkServer.sh start
ZooKeeper JMX enabled by default
Using config: /usr/local/apache-zookeeper-3.6.1-bin/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
grx@ubuntu:/usr/local/apache-zookeeper-3.6.1-bin/bin$
```

2、安装dubbo-admin管理控制台【linux】

dubbo本身并不是一个服务软件。它其实就是一个jar包能够帮你的java程序连接到zookeeper，并利用zookeeper消费、提供服务。所以你不用在Linux上启动什么dubbo服务。

但是为了让用户更好的管理监控众多的dubbo服务，官方提供了一个可视化的监控程序，不过这个监控即使不装也不影响使用。

2.1、下载dubbo-admin

```
git clone https://github.com/apache/dubbo-admin/tree/master
```

2.2、修改dubbo-admin配置

修改 src\main\resources\application.properties 指定zookeeper地址

```
5 spring.guest.password=guest
6
7 dubbo.registry.address=zookeeper://127.0.0.1:2181
```

2.3、打包dubbo-admin

```
mvn clean package -Dmaven.test.skip=true
```

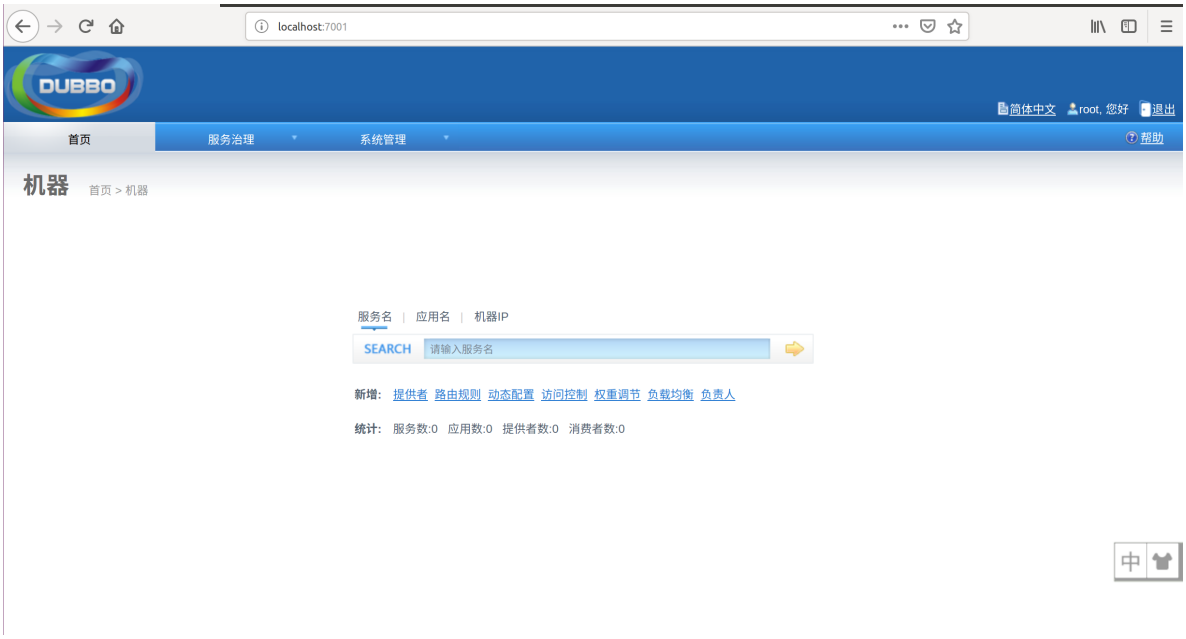
2.4、运行dubbo-admin

```
java -jar dubbo-admin-0.0.1-SNAPSHOT.jar
```

在浏览器输入地址： http://localhost:7001/

默认用户名：root

默认密码：root



四、dubbo-helloworld

1、提出需求

某个电商系统，订单服务需要调用用户服务获取某个用户的所有地址；

我们现在需要创建两个服务模块进行测试

模块	功能
订单服务web模块	创建订单等
用户服务service模块	查询用户地址等

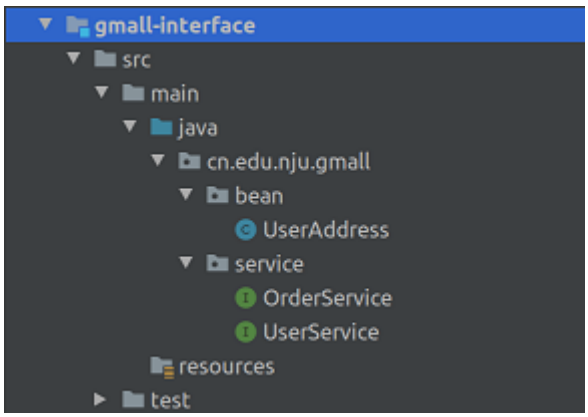
测试预期结果：

订单服务web模块在A服务器，用户服务模块在B服务器，A可以远程调用B的功能。

2、工程架构图

3、创建模块

3.1、gmall-interface：公共接口层（model，service，exception...）



① bean模型：

> 注意实现 java.io.Serializable 接口

```
public class UserAddress implements Serializable {
    private Integer id;
    private String userAddress;
    private String userId;
    private String consignee;
    private String phoneNum;
    private String isDefault;
}
```

② service接口：cn.edu.nju.gmall.service.UserService.getUserAddressList(String userId);

3.2、boot-user-service-provider：用户模块（对用户接口的实现）

① pom.xml中导入gmall-interface依赖

```
        <dependency>
            <groupId>cn.edu.nju</groupId>
            <artifactId>gmall-interface</artifactId>
            <version>1.0-SNAPSHOT</version>
            <scope>compile</scope>
        </dependency>
```

② 实现接口

```
public class UserServiceImpl implements UserService {
    public List<UserAddress> getUserAddressList(String userId) {
        UserAddress address1 = new UserAddress(1, "beijing", "1", "li", "888",
"Y");
        UserAddress address2 = new UserAddress(2, "hangzhou", "1", "wang",
"666", "N");
        return Arrays.asList(address1, address2);
    }
}
```

3.4、boot-order-service-consumer：订单模块（调用用户模块）

① pom.xml中导入gmall-interface依赖

```
        <dependency>
            <groupId>cn.edu.nju</groupId>
            <artifactId>gmall-interface</artifactId>
            <version>1.0-SNAPSHOT</version>
            <scope>compile</scope>
        </dependency>
```

② 调用接口

```
public class OrderServiceImpl implements OrderService {

    UserService userService;

    public List<UserAddress> initOrder(String userId) {
        System.out.println("userId = " + userId);
        return userService.getUserAddressList(userId);
    }
}
```

现在这样是无法进行调用的。我们在boot-order-service-consumer引入了gmall-interface，但是interface的实现是boot-user-service-provider，我们并没有引入，而且实际他可能还在别的服务器中。

4、使用dubbo改造

4.1、改造boot-user-service-provider作为服务提供者

① 导入pom依赖

```
        <dependency>
            <groupId>com.alibaba.boot</groupId>
            <artifactId>dubbo-spring-boot-starter</artifactId>
            <version>0.2.0</version>
        </dependency>
```

② 编写application.properties配置文件

```
dubbo.application.name=user-service-provider

dubbo.registry.address=127.0.0.1:2181
dubbo.registry.protocol=zookeeper

dubbo.protocol.name=dubbo
dubbo.protocol.port=20880

dubbo.monitor.protocol=registry
```

③ 添加注解

在接口实现类上添加注解

```
@Service    // com.alibaba.dubbo.config.annotation.Service，注意和spring的@Service区分
@Component  // 添加到Spring容器中
public class UserServiceImpl implements UserService {
```

在SpringBoot启动类上添加注解

```
@EnableDubbo // 启动dubbo服务
@SpringBootApplication
public class BootUserServiceProviderApplication {
```

4.2、改造boot-order-service-consumer作为服务消费者

① 导入pom依赖

```
        <dependency>
            <groupId>com.alibaba.boot</groupId>
            <artifactId>dubbo-spring-boot-starter</artifactId>
            <version>0.2.0</version>
        </dependency>
```

② 编写application.properties配置文件

```
# 端口不要和tomcat冲突
server.port=8081

dubbo.application.name=order-service-consumer
dubbo.registry.address=zookeeper://127.0.0.1:2181
dubbo.monitor.protocol=registry
```

③ 添加注解

在使用接口时添加注解

```
@Service // 添加到Spring容器
public class OrderServiceImpl implements OrderService {

    @Reference // 使用dubbo提供的reference注解引用远程服务
    UserService userService;
```

在SpringBoot启动类上添加注解

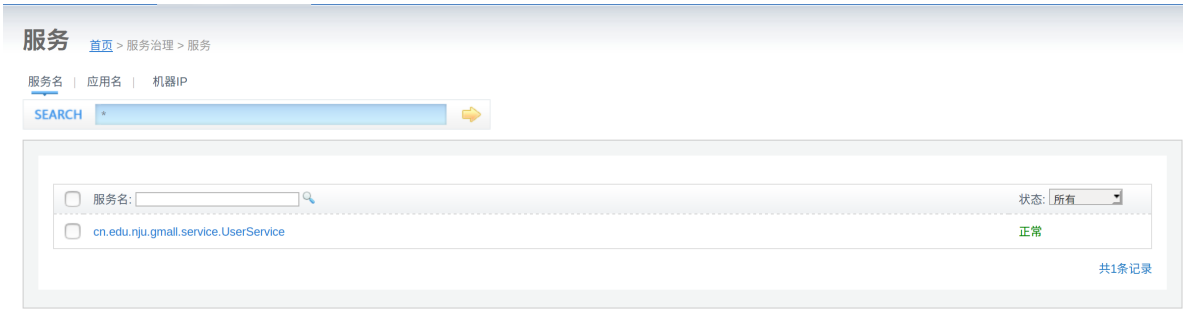
```
@EnableDubbo // 启动dubbo服务
@SpringBootApplication
public class BootUserServiceProviderApplication {
```

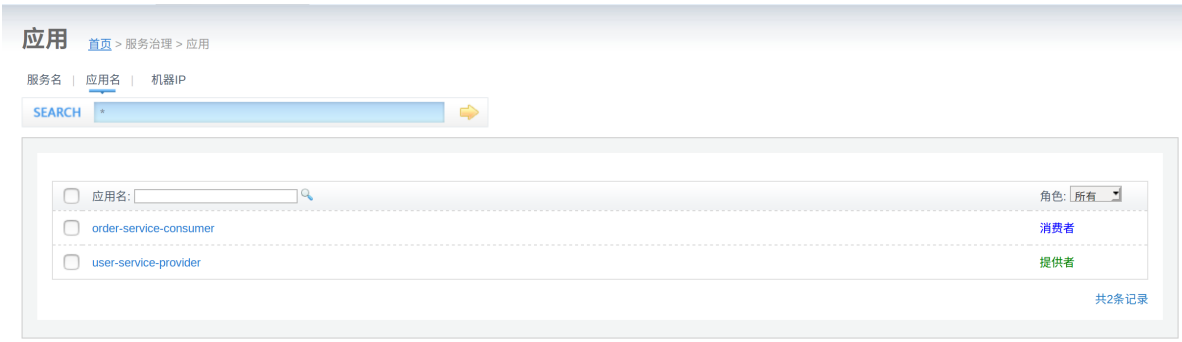
4.3、测试调用

写一个controller用于测试

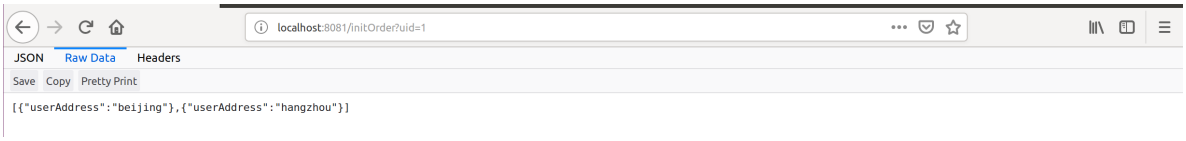
```
@Controller
public class OrderController {

    @Autowired
    OrderService orderService;

    @ResponseBody
    @RequestMapping("/initOrder")
    public List<UserAddress> initOrder(@RequestParam("uid") String userId) {
        return orderService.initOrder(userId);
    }
}
```

启动boot-user-service-provider和boot-order-service-consumer，在dubbo-admin会看到相应服务和应用：





访问boot-order-service-consumer的initOrder请求，会调用UserService获取用户地址



调用成功。说明我们order已经可以调用远程的UserService了；