Dubbo

目录

contents

PART 1 Dubbo 简介

什么是 Dubbo

dubbo 是阿里巴巴公司提供一个开源高性能分布式服务框架，
可以通过 rpc 进行输出和输入功能，以及服务治理方案，
和 spring 进行无缝隙整合

服务框架：消费者（consumer） 提供者（provider）
服务框架就是给请求提供服务

分布式：一个完整的项目分不同模块，对不同的进行拆分模块，
分别部署到不同的服务器上，就是为了提高性能
高延迟高并发解决单一性

rpc：远程过程调用协议，远程访问计算机
通信 socket nio netty mina 等通信技术

PART 2 Dubbo 基础概念

为什么要使用 Dubbo

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

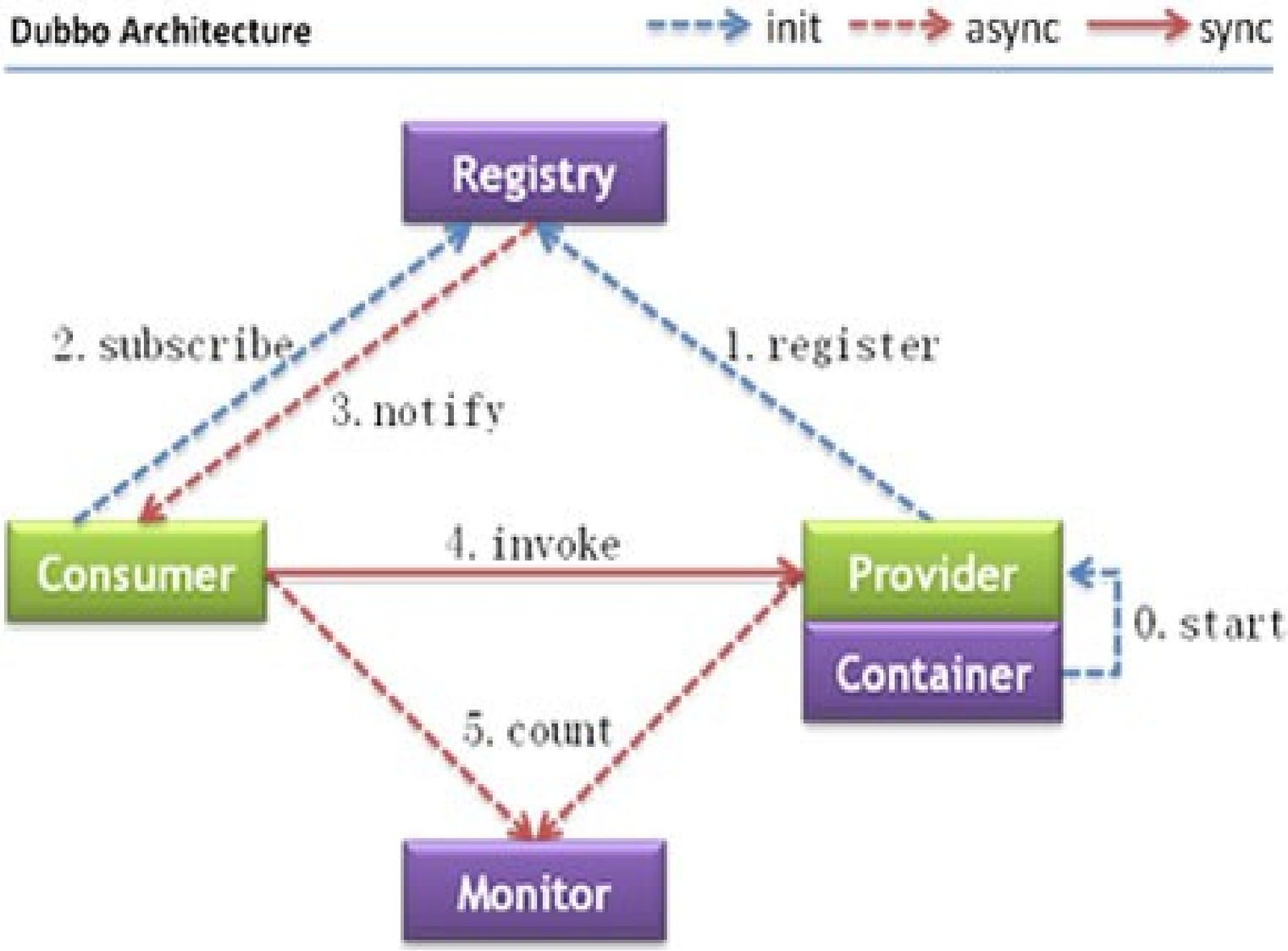
单一应用架构
当网站流量很小时，只需一个应用，将所有功能都部署在一起，以减少部署节点和成本。此时，用于简化增删改查工作量的数据访问框架 (ORM) 是关键。
垂直应用架构
当访问量逐渐增大，单一应用增加机器带来的加速度越来越小，将应用拆成互不相干的几个应用，以提升效率。此时，用于加速前端页面开发的 Web 框架 (MVC) 是关键。
分布式服务架构
当垂直应用越来越多，应用之间交互不可避免，将核心业务抽取出来，作为独立的服务，逐渐形成稳定的服务中心，使前端应用能更快速的响应多变的市场需求。此时，用于提高业务复用及整合的分布式服务框架 (RPC) 是关键。
流动计算架构
当服务越来越多，容量的评估，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。

当服务越来越多时，服务 URL 配置管理变得非常困难，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，降低对硬件负载均衡器的依赖，也能减少部分成本。
当进一步发展，服务间依赖关系变得错踪复杂，甚至分不清哪个应用要在哪个应用之前启动，架构师都不能完整的描述应用的架构关系。 这时，需要自动画出应用间的依赖关系图，以帮助架构师理清理关系。
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，直到响应时间到达阀值，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 主要是用在大型企业里面，因为大型企业里面系统繁多，业务线复杂，而且效率优势非常重要的一块，这里 dubbo 的优势就非常明显了

同步调用与异步调用

Dubbo 架构


Dubbo Architecture
init
async
sync
Registry
2. subscribe
3. notify
1. register
Consumer
4. invoke
Provider
Container
0. start
5. count
Monitor

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

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。

PART 3 Dubbo 怎么玩

Dubbo 示例

Dubbo 使用条件 - 默认 dubbo 协议

使用 maven 工程
使用注册中心 zookeeper
导入 jar 包

```
<!-- dubbo -->
<dependency>
 <groupId>com.alibaba</groupId>
   <artifactId>dubbo</artifactId>
   <version>2.5.8</version>
</dependency>
<!-- zookeeper 客户端驱动 -->
<dependency>
   <groupId>org.apache.zookeeper</groupId>
  <artifactId>zookeeper</artifactId>
   <version>3.4.6</version>
</dependency>
<!--  zookeeper 客户端 zkclient 驱动 -->
<dependency>
  <groupId>com.github.sgroschupf</groupId>
   <artifactId>zkclient</artifactId>
   <version>0.1</version>
</dependency>
```

Dubbo 解析生产者 - 默认 dubbo 协议

因为使用的是 maven 工程 可以将接口单独创建一个工程 方便生产者和消费者依赖

```
// 定义 dubbo 生产者接口，将接口暴露给消费者
public interface  接口名称 {
    // 定义业务接口方法
    public  返回值类型 方法名称（参数类型 参数变量名称）;
}
```

Dubbo 中定义方法参数时，如果是对象需要将对象序列化

```
public class  对象 implements Serializable{}
```

```
//dubbo 生产者实现类
public class  接口实现类名称 implements  接口名称 {
  @Override
  public  返回类型 方法名称（参数类型  参数变量名称）{
    // 业务逻辑
    return    返回类型；
  }
  // 实现多个方法
}
```

通过 spring 定义 dubbo 生产者配置

```
<?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="工程名称命名"  />

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

```
    <!--
    用 dubbo 协议在 20880 端口暴露服务
      name 表示定义的协议  默认 dubbo
      port 表示定义的端口号 默认 20880
      -->
      <dubbo:protocol name="dubbo" port="20880" />

      <!--
    声明需要暴露的服务接口
      interface 表示生产者暴露给消费者的接口
      ref 表示引入某个生产者，并且发布生产者实现类，但是暴露接口
       -->
      <dubbo:service
         interface="接口名称"
         ref="生产者实现类别名" />

      <!--  和本地 bean 一样实现服务  -->
      <bean id="生产者实现类别名"
            class="需要实例化的生产者实现类" />

</beans>
```

在运行生产者服务时，先启动 zookeeper 服务端，
为了让生产者接口和请求地址注册到 zookeeper 中；
这时再去运行 Provider 类；
再开启 zookeeper 客户端查看信息是否注册成功
输入命令： ls /dubbo　查看注册接口


```
public class Provider {
    public static void main(String[] args) throws Exception {
        ClassPathXmlApplicationContext context =
        new ClassPathXmlApplicationContext("spring 配置 dubbo 生产者 .xml");
        context.start();
        System.in.read(); //  按任意键退出
    }
}
```

Dubbo 解析消费者 - 默认 dubbo 协议

消费者工程也是 maven 工程
消费者和生产者引入 jar 包一致
这个时候消费者需要依赖生产者暴露的接口
（如果接口是一个独立工程可以依赖）
或者自定义接口，需要和生产者提供的接口一致

通过 spring 定义 dubbo 消费者配置

```
<?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">

<!-- 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="工程名称命名" />
```

```
    <!--
    使用 zookeeper 注册调用暴露服务地址
    例如 address="zookeeper://127.0.0.1:2181"
    -->
    <dubbo:registry address="zookeeper 地址 " />

    <!--
    生成远程服务代理，可以和本地 bean 一样使用
    接口为什么能创建对象？ 动态代理
    -->
    <dubbo:reference
        interface=" 消费者调用生产者暴露的接口（这个接口有可能依赖，有可能自定义） "
    id=“ 消费者别名，哪里业务需要使用消费者，注入即可 " />

</beans>
```

```
public class Consumer {
    public static void main(String[] args) throws Exception {
        ClassPathXmlApplicationContext context =
                new  ClassPathXmlApplicationContext("spring 配置 dubbo 消费
者 .xml");
        context.start();
        dubbo:reference 元素 interface 属性接口类型 别名  =
          (dubbo:reference 元素 interface 属性接口类型 )context.getBean(" 引
入 dubbo:reference 元素 id 属性别名 "); //  获取远程服务代理
     返回类型  result =  别名调用方法 (" 参数 ");
    }
}
```

dubbo 解析注解

官方：如果你曾使用旧版 annotation 配置，请删除所有相关配置，我们将在下个版本删除所有旧版配置项。

```
<dubbo:annotation package="com.etoak.dubbo" />
```

在 2.5.7 及以上版本支持 springboot

如果使用注解版，需要在 spring 配置文件中的生产者和消费者加入 dubbo:annotation 元素即可
生产者去除 dubbo:service 元素
消费者去除 dubbo:reference 元素

服务提供方

Service 注解暴露服务

```
// 引入注解
import com.alibaba.dubbo.config.annotation.Service;
// 实例化生产者实现类对象，并 dubbo 发布生产者实现类服务，只暴露接口
@Service(timeout = 5000)
public class AnnotateServiceImpl implements AnnotateService {
    // ...
}
```

服务消费方

Reference 注解引用服务

```
public class AnnotationConsumeService {
// 注入消费者接口，消费者调用生产者暴露的接口
    @com.alibaba.dubbo.config.annotation.Reference
    public AnnotateService annotateService;
    // ...
}
```

@Service 注解和 @Reference 注解都由 dubbo:annotation 元素扫描

dubbo 解析生产者 -hessian 协议

Hessian 是一个轻量级的 remoting onhttp 工具，使用简单的方法提供了 RMI 的功能。
相比 WebService， Hessian 更简单、快捷。采用的是二进制 RPC 协议
，因为采用的是二进制协议，所以它很适合于发送二进制数据。

这里 hessian 用到的 jar 包和 dubbo 协议一致
只是多个 hessian.jar 包

```
<dependency>
   <groupId>com.caucho</groupId>
   <artifactId>hessian</artifactId>
   <version>4.0.7</version>
</dependency>
```

Hessian 创建生产者工程为 web 工程

```
public interface  接口名称 {
     //hessian 适合上传二进制信息这里定义参数类型为 byte 数据
    返回类型 方法名称 (byte[] b);
}
```

```
public class 生产者实现类名称 implements 接口名称 {
  @Override
  public 返回类型 方法名称(byte[] b) {
  // 业务逻辑...
  }
}
```

通过 spring 定义 dubbo 生产者配置

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

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

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

<!-- 暴露服务 -->
<dubbo:protocol name="dubbo" port="20886" />
```

```
    <!--
    contextpath 是当前项目名称,
    用 hessian 协议在 8080( 当前项目名称的 servlet 容器的端口 ) 端口暴露服务,
    server 这里是使用的 web.xml 里面配置的 servlet  这个 servlet 访问 hessian 实现类 -->
     <dubbo:protocol name="hessian" port=" 当前项目端口号 "
         contextpath=" 当前项目名称 " server="servlet" />

     <!--
     需要指定 hessian 协议，否则会在上面设置的协议中随机调用，就会一会成功一会失败
     protocol 属性表示指定协议
     当前配置中有 2 个协议
     <dubbo:protocol /> duubo 、 hessian
     path 属性表示通过请求地址访问 hessian 实现类
     contextpath 属性和 path 属性组装请求地址
     例如 / 当前项目名称 /hessian 自定义地址
     -->
    <dubbo:service  protocol="hessian"
                    interface=" 接口名称 " ref=" 生产者实现类别名 "
                    path="hessian 自定义地址 "/>

    <bean id=" 生产者实现类别名 " class=" 生产者实现类 "></bean>

</beans>
```

在 web.xml 中配置

```
<!-- spring 配置 -->
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>classpath:applicationContext.xml</param-value>
</context-param>
<listener>
  <listener-class>
       org.springframework.web.context.ContextLoaderListener
  </listener-class>
</listener>
<!-- hessian 协议 server="servlet"
消费者访问生产者时会触发 DispatcherServlet 通过放射访问实现类 -->
<servlet>
  <servlet-name>dubbo</servlet-name>
  <servlet-class>
       com.alibaba.dubbo.remoting.http.servlet.DispatcherServlet
  </servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>dubbo</servlet-name>
  <url-pattern>/*</url-pattern>
</servlet-mapping>
```

dubbo 解析消费者 -hessian 协议

消费者工程也是 maven 工程
消费者和生产者引入 jar 包一致
这个时候消费者需要依赖生产者暴露的接口
（如果接口是一个独立工程可以依赖）
或者自定义接口，需要和生产者提供的接口一致

通过 spring 定义 dubbo 消费者配置

```
<?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">

<!-- 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="工程名称命名" />
```

```
    <!--
    使用 zookeeper 注册调用暴露服务地址
    例如 address="zookeeper://127.0.0.1:2181"
    -->
    <dubbo:registry address="zookeeper 地址 " />

    <!--
    生成远程服务代理，可以和本地 bean 一样使用
    接口为什么能创建对象？ 动态代理
    -->
    <dubbo:reference
        interface=" 消费者调用生产者暴露的接口（这个接口有可能依赖，有可能自定义） "
    id=“ 消费者别名，哪里业务需要使用消费者，注入即可 " />

</beans>
```

```
public class Consumer {
    public static void main(String[] args) throws Exception {
        ClassPathXmlApplicationContext context =
                new  ClassPathXmlApplicationContext("spring 配置 dubbo 消费
者 .xml");
        context.start();
        dubbo:reference 元素 interface 属性接口类型 别名  =
          (dubbo:reference 元素 interface 属性接口类型 )context.getBean(" 引
入 dubbo:reference 元素 id 属性别名 "); //  获取远程服务代理
     返回类型  result =  别名调用方法 (" 参数 ");
    }
}
```

SpringBoot 配置 dubbo

注解配置

需要 2.5.7 及以上版本支持
服务提供方
Service 注解暴露服务

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

@Service(timeout = 5000)
public class AnnotateServiceImpl implements AnnotateService {
    // ...
}
```

javaconfig 形式配置公共模块

```
@Configuration
public class DubboConfiguration {

    @Bean
    public ApplicationConfig applicationConfig() {
        ApplicationConfig applicationConfig = new ApplicationConfig();
        applicationConfig.setName("provider-test");
        return applicationConfig;
    }

    @Bean
    public RegistryConfig registryConfig() {
        RegistryConfig registryConfig = new RegistryConfig();
        registryConfig.setAddress("zookeeper://127.0.0.1:2181");
        registryConfig.setClient("curator");
        return registryConfig;
    }
}
```

指定 dubbo 扫描路径

```
@SpringBootApplication
@DubboComponentScan(basePackages = "com.alibaba.dubbo.test.service.impl")
public class ProviderTestApp {
    // ...
}
```

服务消费方
Reference 注解引用服务

```
public class AnnotationConsumeService {

    @com.alibaba.dubbo.config.annotation.Reference
    public AnnotateService annotateService;

    // ...
}
```

javaconfig 形式配置公共模块

```
@Configuration
public class DubboConfiguration {

    @Bean
    public ApplicationConfig applicationConfig() {
        ApplicationConfig applicationConfig = new ApplicationConfig();
        applicationConfig.setName("consumer-test");
        return applicationConfig;
    }

    @Bean
    public ConsumerConfig consumerConfig() {
        ConsumerConfig consumerConfig = new ConsumerConfig();
        consumerConfig.setTimeout(3000);
        return consumerConfig;
    }

    @Bean
    public RegistryConfig registryConfig() {
        RegistryConfig registryConfig = new RegistryConfig();
        registryConfig.setAddress("zookeeper://127.0.0.1:2181");
        registryConfig.setClient("curator");
        return registryConfig;
    }
}
```

指定 dubbo 扫描路径

```
@SpringBootApplication
@DubboComponentScan(basePackages = "com.alibaba.dubbo.test.service")
public class ConsumerTestApp {
    // ...
}
```