Dubbo核心原理之源码分析

Dubbo是Alibaba开源的分布式服务框架，它最大的特点是按照分层的方式来架构，使用这种方式可以使各个层之间解耦合(或者最大限度地松耦合)。从服务模型的角度来看，Dubbo采用的是一种非常简单的模型，要么是提供方提供服务，要么是消费方消费服务，所以基于这一点可以抽象出服务提供方(Provider)和服务消费方(Consumer)两个角色。

Dubbo是什么

简单说呢，Dubbo用起来就和EJB、WebService差不多，调用一个远程的服务(或者JavaBean)的时候在本地有一个接口，就像调用本地的方法一样去调用，它底层帮你实现好你的方法参数传输和远程服务运行结果传回之后的返回，就是RPC的一种封装啦~

当然，这个只是Dubbo的最基本的功能，它的特点是：

1.它主要是使用高效的网络框架和序列化框架，让分布式服务之间调用效率更高。

2.采用

注册中心

管理众多的服务接口地址，当你想调用服务的时候只需要跟注册中心询问即可，不用像使用WebService一样每个服务都得记录好接口调用方式。

3.

监控中心

：实现对服务方和调用方之间运行状态的监控，还能控制服务的优先级、权限、权重、上下线等，让整个庞大的分布式服务系统的维护和治理比较方便。

4.高可用：有个服务宕机了?注册中心就会从服务列表去掉该节点。还是调用到了?客户端会向注册中心请求另一台可用的服务节点重新调用。注册中心宕机?注册中心也能实现高可用(ZooKeeper)。

5.负载均衡：采用软负载均衡算法实现对多个相同服务的节点的请求负载均衡。

Dubbo注册中心

上面已经安装完成了zookeeper的注册中心了，这个注册中心主要就是负责dubbo的所有服务地址列表维护，并且可以通过在ZooKeeper节点中设置相应的值来实现对这个服务的权重、优先级、是否可用、路由、权限等的控制。

你可以先记住，之后在Dubbo的管理控制台对服务的一堆治理策略设置和调整，实际上就是修改了注册中心中的服务对应的配置数据(即修改了zookeeper中服务对应的节点的配置数据)。

之后

Consumer

从注册中心请求到服务的数据时就能根据这些配置数据进行相应的治理配置参数的代码执行生效。

Dubbo样例服务开发

这里我用maven构建项目，在Spring环境中配置Provider和Consumer。

先说明使用的依赖：

org.springframework

spring-core

4.2.3.RELEASE

org.springframework

spring-beans

4.2.3.RELEASE

org.springframework

spring-context

4.2.3.RELEASE

org.springframework

spring-test

4.2.3.RELEASE

org.springframework

spring-tx

4.2.3.RELEASE

org.springframework

spring-web

4.2.3.RELEASE

org.springframework

spring-webmvc

4.2.3.RELEASE

com.alibaba

dubbo

2.8.4

javassist

javassist

3.12.0.GA

org.jboss.netty

netty

LATEST

com.101tec

zkclient

0.10

org.slf4j

slf4j-log4j12

1.7.12

Provider

：

声明服务的接口：

public interface IMyDemo { String sayHello(String name);

}

对接口进行实现(这里是Provider，需要真的实现，之后在Consumer端调用接口之后实际就是在这里的实现代码执行所需逻辑的)：

public class MyDemo implements IMyDemo { @Override

public String sayHello(String name) {

String hello = "hello " + name;

System.out.println(hello); return hello;

}

}

本地测试一下这个服务是否可用，这里还没用到Dubbo，只是先测试一下Spring容器是否有问题：

@org.junit.Testpublic void testDubbo() throws InterruptedException {

ApplicationContext providerContext = new ClassPathXmlApplicationContext("provider.xml");

IMyDemo demo = providerContext.getBean(IMyDemo.class);

System.out.println(demo.sayHello("world"));

Thread.sleep(60000);

}

RPC之 Dubbo 实现

主要为三点，动态代理、反射、socket网络编程

看过很多讲dubbo原理的文章，总感觉太抽象，偶然间看到一个直播课堂讲dubbo原理。结合了一个订单的例子现场画笔工具画图，直观很多。截屏记录下来。(提供技术直播的为老马的北京尚学堂，感谢)

客户端使用动态代理的方式，“假装”实现了createOrder方法。

方法相关的数据通过序列化，进入到socket服务器。dubbo的socket实现为Netty。

服务端从socket服务器取出数据，通过反射的方式找到“真实”的服务实现。

服务端的方法在服务启动时已注入。

IMG_4519.PNG

服务发现层，可用zookeeper。zookeeper保证了CP(一致性，分区容错性)。缺点：master节点挂掉时，需要时间重新选择master，这段时间内注册中心将不可用。

注意：服务端可消费端注册成功后，通讯只走socket服务器，不会经过注册中心。

Dubbo

一、dubbo核心技术简介

远程服务的调用流程，dubbo本质上就是解决了此问题。

远程服务器调用流程

1.客户端发起接口调用

2.服务中间件进行路由选址：找到具体接口实现的服务地址

3.客户端将请求信息进行编码(序列化: 方法名，接口名，参数，版本号等)

4.建立与服务端的通讯(不是调度中心，而是客户端与服务端直连)

5.服务端将接收到的信息进行反编码(反序列化)

6.根据信息找到服务端的接口实现类

7.将执行结果反馈给客户端

针对上面的调用流程，结合dubbo的服务架构，是不是对dubbo的了解又深入了一些?同学们有空也可以根据上述的流程自己实现一遍简单的远程调用，下面为dubbo核心模块用到的一些技术，可以提前做一些知识储备。

核心技术

1.java多线程

2.JVM

3.网络通讯(NIO)

4.动态代理

5.反射

6.序列化

7.路由节点管理(zookeeper)

二、dubbo实际中常用配置

常见的一些业务场景和dubbo配置。

分包

建议将服务接口，服务模型，服务异常等均放在API包中，因为服务模型及异常也是API的一部分，同时，这样做也符合分包原则：重用发布等价原则(REP)，共同重用原则(CRP)

如果需要，也可以考虑在API包中放置一份spring的引用配置，这样使用方，只需在Spring加载过程中引用此配置即可，配置建议放在模块的包目录下，以免冲突，如：
com/alibaba/china/xxx/dubbo-reference.xml

粒度

服务接口尽可能大粒度，每个服务方法应代表一个功能，而不是某功能的一个步骤，否则将面临分布式事务问题，Dubbo暂未提供分布式事务支持。

服务接口建议以业务场景为单位划分，并对相近业务做抽象，防止接口数量爆炸

不建议使用过于抽象的通用接口，如：Map query(Map)，这样的接口没有明确语义，会给后期维护带来不便。

版本

每个接口都应定义版本号，为后续不兼容升级提供可能，如：

建议使用两位版本号，因为第三位版本号通常表示兼容升级，只有不兼容时才需要变更服务版本。

当不兼容时，先升级一半提供者为新版本，再将消费者全部升为新版本，然后将剩下的一半提供者升为新版本。

兼容性

服务接口增加方法，或服务模型增加字段，可向后兼容，删除方法或删除字段，将不兼容，枚举类型新增字段也不兼容，需通过变更版本号升级。

各协议的兼容性不同，参见： 服务协议

枚举值

如果是完备集，可以用Enum，比如：ENABLE, DISABLE。

如果是业务种类，以后明显会有类型增加，不建议用Enum，可以用String代替。

如果是在返回值中用了Enum，并新增了Enum值，建议先升级服务消费方，这样服务提供方不会返回新值。

如果是在传入参数中用了Enum，并新增了Enum值，建议先升级服务提供方，这样服务消费方不会传入新值。

序列化

服务参数及返回值建议使用POJO对象，即通过set,get方法表示属性的对象。

服务参数及返回值不建议使用接口，因为数据模型抽象的意义不大，并且序列化需要接口实现类的元信息，并不能起到隐藏实现的意图。

服务参数及返回值都必需是byValue的，而不能是byRef的，消费方和提供方的参数或返回值引用并不是同一个，只是值相同，Dubbo不支持引用远程对象。

异常

建议使用异常汇报错误，而不是返回错误码，异常信息能携带更多信息，以及语义更友好，

如果担心性能问题，在必要时，可以通过override掉异常类的fillInStackTrace()方法为空方法，使其不拷贝栈信息，

查询方法不建议抛出checked异常，否则调用方在查询时将过多的try...catch，并且不能进行有效处理，

服务提供方不应将DAO或SQL等异常抛给消费方，应在服务实现中对消费方不关心的异常进行包装，否则可能出现消费方无法反序列化相应异常。

调用

不要只是因为是Dubbo调用，而把调用Try-Catch起来。Try-Catch应该加上合适的回滚边界上。

对于输入参数的校验逻辑在Provider端要有。如有性能上的考虑，服务实现者可以考虑在API包上加上服务Stub类来完成检验。

版本控制

在dubbo最佳实践中有提到，所有接口都应定义版本，在这里有几点需要注意下，接口服务如果更新频繁，并且兼容老版本的，不建议更改版本号，因为dubbo这边对除 * 以外的版本号，都是采用完全匹配的方式进行匹配。即服务端的版本号如果从1.0升级为1.1，并且未保留原有的1.0的服务，那么客户端必须同时也将服务版本号升级为1.1，否则将无法匹配到远处服务。

博主在自己项目中的版本使用规则如下，仅供参考：

•版本号采用两位，x.x 第一位表示需要非兼容升级，第二位表示兼容升级

•bug fix程度的升级不改版本号

•版本升级的时候，保证老版本服务的继续使用，同时部署新老版本，等客户端全部升级完成后，再考虑下架老版本服务

•版本可以细化配置到具体的接口 ，但是我们建议以通用配置来控制版本号

•

对服务进行调优

dubbo的服务调用有很多默认配置，这些配置可能会引起服务调用业务上的错误，需要特别注意的有以下几点：

otimeout，调用超时时间，默认为1000毫秒，即超过1000毫秒没有返回数据，就会执行重试机制

oretries，失败重试次数，默认为2，即失败(超时)之后的重试次数

oconnections，对每个提供者的最大链接数，默认为100，建议根据服务器配置进行调整

oloadbalance，负载均衡策略，默认为random

oasync, 是否异步执行，默认为false

odelay, 延迟注册服务时间，默认为0，建议不同的接口把暴露服务时间错开，避免dubbo爆端口被占用错误(博主曾深受其害)

以上的几点，如果服务端与客户端都同时进行了配置，则客户端优先级更高。

以下是根据我们服务器性能与业务需求的部分通用配置.

当某接口执行时间非常长的时候，常见的有三种方式去处理：

•忽略返回值,配置return为true

•

•

•配置为异步

•

•

•

•配置为回调的方式

服务端配置：

•

•

•

•

•

•

接口实现：

```
package com.lijian.dubbo.service.impl;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;
import com.lijian.dubbo.listener.MyListener;
import com.lijian.dubbo.service.CallbackService;
public class CallbackServiceImpl implements CallbackService {
private final Maplisteners = new ConcurrentHashMap();
public CallbackServiceImpl() {
Thread t = new Thread(new Runnable() {
public void run() {
while (true) {
try {
for (Map.Entryentry : listeners
.entrySet()) {
try {
entry.getValue().changed(
getChanged(entry.getKey()));
} catch (Throwable t) {
listeners.remove(entry.getKey());
}
}
Thread.sleep(5000); // 定时触发变更通知
} catch (Throwable t) { // 防御容错
```

```
t.printStackTrace();
}
}
}
});
t.setDaemon(true);
t.start();
}
public void addListener(String key, MyListener listener) {
listeners.put(key, listener);
listener.changed(getChanged(key)); // 发送变更通知
}
private String getChanged(String key) {
return "Changed: "
+ new SimpleDateFormat("yyyy-MM-dd HH:mm:ss")
.format(new Date());
}
}
```

对参数进行校验

使用spring的同学对@Validated 注解肯定不会感到陌生，可以对请求参数进行格式校验:

controller代码：

```
@RequestMapping(value = {""},method = RequestMethod.POST)
@ResponseBody
public GeneralResult addAdvertising(@Validated @RequestBody AdvertisingForm form){
return GeneralResult.newBuilder().setResult(advertisingService.addAdvertising(form));
}
```

AdvertisingForm部分代码：

```
public class AdvertisingForm {
```

```
@NotEmpty(message = "标题不能为空")
private String title;
@NotEmpty(message = "照片不能为空")
private String photo;
...
}
```

在dubbo中，同样可以使用validate功能进行格式校验.

需要被校验的User类：

```
package com.lijian.dubbo.beans;
import java.io.Serializable;
import javax.validation.constraints.Min;
import org.hibernate.validator.constraints.NotEmpty;
public class User implements Serializable{
private static final long serialVersionUID = 83320693853054146291;
@NotEmpty(message="姓名不可为空")
private String name;
@Min(value=18,message="年龄必须大于18岁")
private Integer age;
public String getName() {
return name;
}
public void setName(String name) {
this.name = name;
}
public Integer getAge() {
return age;
}
public void setAge(Integer age) {
```

this.age = age;

}

}

dubbo 的validate配置：

接口服务的调用如下：

```
package com.lijian.dubbo.consumer.main;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.lijian.dubbo.beans.User;

import com.lijian.dubbo.consumer.action.UserAction;

public class ValidateMainClass {

@SuppressWarnings("resource")

public static void main(String[] args){

ClassPathXmlApplicationContext context = new
ClassPathXmlApplicationContext("applicationContext.xml");

context.start();

UserAction userAction = context.getBean(UserAction.class);

User user = new User();

// 如果年龄小于18，会报出 Caused by: javax.validation.ConstraintViolationException: Failed
to validate service: com.lijian.dubbo.service.ValidateService, method: insert, cause:
[ConstraintViolationImpl{interpolatedMessage='年龄必须大于18岁', propertyPath=age,
rootBeanClass=class com.lijian.dubbo.beans.User, messageTemplate='年龄必须大于18岁'}]

// user.setAge(19);

user.setAge(17);

// 如果name为空，会报出 Caused by: javax.validation.ConstraintViolationException: Failed to
validate service: com.lijian.dubbo.service.ValidateService, method: insert, cause:
[ConstraintViolationImpl{interpolatedMessage='姓名不可为空', propertyPath=name,
rootBeanClass=class com.lijian.dubbo.beans.User, messageTemplate='姓名不可为空'}]

user.setName("我是大帅哥");

System.out.println(userAction.addUser(user));

}

}
```