一、前言

部门去年年中开始各种改造，第一步是模块服务化，这边初选dubbo试用在一些非重要模块上，慢慢引入到一些稍微重要的功能上，半年时间，学习过程及线上使用遇到的些问题在此总结下。

整理这篇文章差不多花了两天半时间，请尊重劳动成果，如转载请注明出处

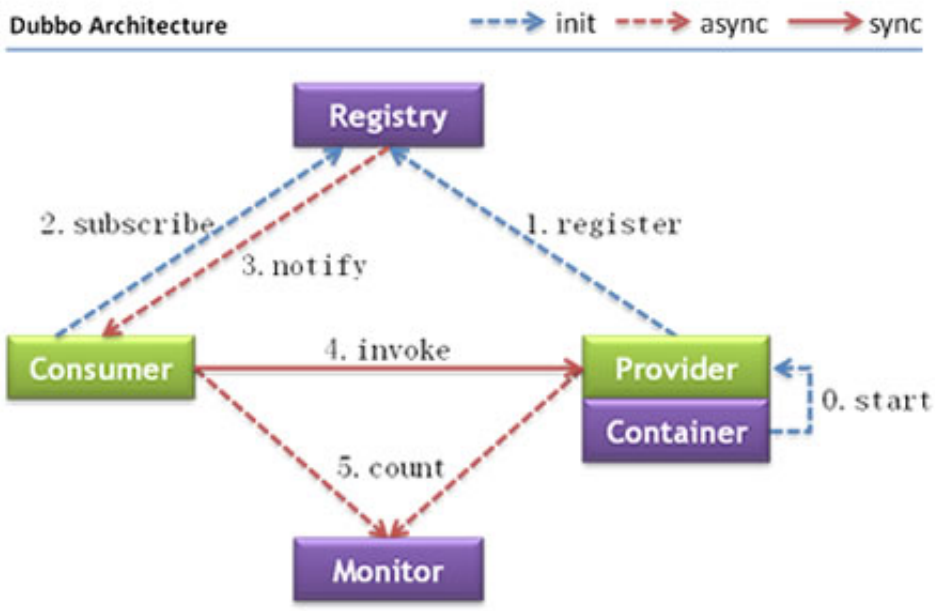
http://blog.csdn.net/hzzhoushaoyu/article/details/43273099

二、什么是dubbo

Dubbo是阿里巴巴提供的开源的SOA服务化治理的技术框架，据说只是剖出来的一部分开源的，但一些基本的需求已经可以满足的，而且扩展性也非常好（至今没领悟到扩展性怎么做到的），通过spring bean的方式管理配置及实例，较容易上手且对应用无侵入。更多介绍可戳http://alibaba.github.io/dubbo-doc-static/Home-zh.htm。

三、如何使用dubbo

1.服务化应用基本框架



如上图所示，一个抽象出来的基本框架，consumer和provider是框架中必然存在的，

Registry做为全局配置信息管理模块，推荐生产环境使用Registry，可实时推送现存活的服务提供者，Monitor一般用于监控和统计RPC调用情况、成功率、失败率等情况，让开发及运维了解线上运行情况。

应用执行过程大致如下：

- 服务提供者启动，根据协议信息绑定到配置的IP和端口上，如果已有服务绑定过相同IP和端口的则跳过
- 注册服务信息至注册中心
- 客户端启动，根据接口和协议信息订阅注册中心中注册的服务，注册中心将存活的服务地址通知到客户端，当有服务信息变更时客户端可以通过定时通知得到变更信息
- 在客户端需要调用服务时，从内存中拿到上次通知的所有存活服务地址，根据路由信息和负载均衡机制选择最终调用的服务地址，发起调用
- 通过filter分别在客户端发送请求前和服务端接收请求后，通过异步记录一些需要的信息传递到monitor做监控或者统计

2.服务接口定义

一般单独有一个jar包，维护服务接口定义、RPC参数类型、RPC返回类型、接口异常、接口用到的常量，该jar包中不处理任何业务逻辑。

比如命名api-0.1.jar，在api-0.1.jar中定义接口

```
public interface UserService
{
    public RpcResponseDto isValidUser(RpcAccountRequestDto
requestDto) throws new RpcBusinessException, RpcSystemException;
}
```

并在api-0.1.jar中定义RpcResponseDto,RpcAccountRequestDto,RpcBusinessException,RpcSystemException。

服务端通过引用该jar包实现接口并暴露服务，客户端引用该jar包引用接口的代理实例。

3.注册中心

开源的dubbo已支持4种组件作为注册中心，我们部门使用推荐的zookeeper做为注册中心，由于就瓶颈来说不会出现在注册中心，风险较低，未做特别的研究或比较。

- zookeeper，推荐集群中部署奇数个节点，由于zookeeper挂掉一半的机器集群

就不可用，所以部署4台和3台的集群都是在挂掉2台后集群不可用

- redis
- multicast,广播受到网络结构的影响，一般本地不想搭注册中心的话使用这种调用
- dubbo简易注册中心

对于zookeeper客户端，dubbo在2.2.0之后默认使用zkclient，2.3.0之后提供可选配置Curator，提到这个点的原因主要是因为zkclient发现一些问题：①服务器在修改服务器时间后zkClient会抛出日志错误之类的异常然后容器（我们使用resin）挂掉了，也不能确定就是zkClient的问题，接入dubbo之前无该问题②dubbo使用zkclient不传入连接zookeeper等待超时时间，使用默认的Integer.MAX_VALUE，这样在zookeeper连不上的情况下不报错也无法启动；目前我们准备寻找其他解决方案，比如使用curator试下，还没正式投入。

4.服务端

配置应用名

```
<dubbo:application name="test"/>
```

配置dubbo注解识别处理器，不指定包名的话会在spring bean中查找对应实例的类配置了dubbo注解的

```
<dubbo:annotation/>
```

配置注册中心，通过group指定注册中心分组，可通过register配置是否注册到该注册中心以及subscribe配置是否从该注册中心订阅

```
<dubbo:registry address="zookeeper://127.0.0.1:2181/" group="test"/>
```

配置服务协议，多网卡可通过IP指定绑定的IP地址，不指定或者指定非法IP的情况下会绑定在0.0.0.0，使用Dubbo协议的服务会在初始化时建立长连接

```
<dubbo:protocol name="dubbo" port="20880" accesslog="d:/access.log"></dubbo:protocol>
```

通过xml配置文件配置服务暴露，首先要有个spring bean实例（无论是注解配置的还是配置文件配置的），在下面ref中指定bean实例ID，作为服务实现类

```
<dubbo:service interface="com.web.foo.service.FirstDubboService" ref="firstDubboServiceImpl" version="1.0"></dubbo:service>
```

通过注解方式配置服务暴露，Component是Spring bean注解，Service是dubbo的注解（不要和spring bean的service注解弄混），如前文所述，dubbo注解只会在spring bean中被识别

```
@Component
@Service(version="1.0")
public class FirstDubboServiceImpl implements FirstDubboService
{

        @Override
        public void sayHello(TestDto test)
        {
                System.out.println("Hello World!");
        }

}
```

5.客户端

同服务端配置应用名、注解识别处理器和注册中心。

配置客户端reference bean。客户端跟服务端不同的是客户端这边没有实际的实现类的，所以配置的dubbo:reference实际会生成一个spring bean实例，作为代理处理Dubbo请求，然后其他要调用处直接使用spring bean的方式使用这个实例即可。xml配置文件配置方式,id即为spring bean的id,之后无论是在spring配置中使用ref="firstDubboService"还是通过@Autowired注解都OK

```
<dubbo:reference
interface="com.web.foo.service.FirstDubboService"
                version="1.0" id="firstDubboService" >
</dubbo:reference>
```

另外开发、测试环境可通过指定Url方式绕过注册中心直连指定的服务地址，避免注册中心中服务过多，启动建立连接时间过长，如

```
<dubbo:reference
interface="com.web.foo.service.FirstDubboService"
                version="1.0" id="firstDubboService"
url="dubbo://127.0.0.1:20880/"></dubbo:reference>
```

注解配置方式引用，

```
@Component
public class Consumer
{
        @Reference(version="1.0")
        private FirstDubboService service;
```

```
        public void test()
        {
                TestDto test = new TestDto();
                test.setList(Arrays.asList(new String[]{"a",
"b"}));
                test.setTest("t");
                service.sayHello(test);
        }
}
```

Reference被识别的条件是spring bean实例对应的当前类中的field，如上是直接修饰spring bean当前类中的属性

这个地方看了下源码，本应该支持当前类和父类中的public set方法，但是看起来是个BUG，Dubbo处理reference处部分源码如下

```
        Method[] methods = bean.getClass().getMethods();
        for (Method method : methods) {
            String name = method.getName();
            if (name.length() > 3 && name.startsWith("set")
                    && method.getParameterTypes().length == 1
                    && Modifier.isPublic(method.getModifiers())
                    && !
Modifier.isStatic(method.getModifiers())) {
                try {
                        Reference reference =
method.getAnnotation(Reference.class);
                        if (reference != null) {
                                Object value = refer(reference,
method.getParameterTypes()[0]);
                                if (value != null) {
                                        method.invoke(bean, new
Object[] {  });//?？这里不是应该把value作为参数调用么，而且为什么上面if条件判断参数为1这里不传参数
                                }
                        }
                } catch (Throwable e) {
                    logger.error("Failed to init remote service
reference at method " + name + " in class " +
```

```
bean.getClass().getName() + ", cause: " + e.getMessage(), e);
                    }
                }
            }
```

6.监控中心

如果使用Dubbo自带的监控中心，可通过简单配置即可，先通过github获得dubbo-monitor的源码，部署启动后在应用配置如下

```
<dubbo:monitor protocol="registry" /> <!--通过注册中心获取monitor地址后建立连接-->

<dubbo:monitor
address="dubbo://127.0.0.1:7070/com.alibaba.dubbo.monitor.MonitorService" /> <!--绕过注册中心直连monitor，同consumer直连-->
```

7.服务路由

最重要辅助功能之一，可随时配置路由规则调整客户端调用策略，目前dubbo-admin中已提供基本路由规则的配置UI，到github下载源码部署后很容易找到地方，这里简单介绍下怎么用路由。

下面是dubbo-admin的新建路由界面，可配置信息都在图片中有，

比如现在我们有10.0.0.1~3三台消费者和10.0.0.4~6三台服务提供者，想让1和2调用4，3调用5和6的话，则可以配置两个规则，

1.消费者IP：10.0.0.1,10.0.0.2 ；提供者IP：10.0.0.4

2.消费者IP：10.0.0.3；提供者IP：10.0.0.5,10.0.0.6

另外，IP地址支持结尾为*匹配所有，如10.0.0.*或者10.0.*等。

不匹配的配置规则和匹配的配置规则是一致的。

配置完成后可在消费者标签页查看路由结果

8.负载均衡

dubbo提供4种负载均衡方式：

- Random，随机，按权重配置随机概率，调用量越大分布越均匀，默认是这种方式
- RoundRobin，轮询，按权重设置轮询比例，如果存在比较慢的机器容易在这台机器的请求阻塞较多
- LeastActive，最少活跃调用数，不支持权重，只能根据自动识别的活跃数分配，不能灵活调配
- ConsistentHash，一致性hash，对相同参数的请求路由到一个服务提供者上，如果有类似灰度发布需求可采用

dubbo的负载均衡机制是在客户端调用时通过内存中的服务方信息及配置的负责均衡策略选择，如果对自己系统没有一个全面认知，建议先采用random方式。

9.dubbo过滤器

有需要自己实现dubbo过滤器的，可关注如下步骤：

1. dubbo初始化过程加载META-INF/dubbo/internal/，META-INF/dubbo/，META-INF/services/三个路径(classloaderresource)下面的com.alibaba.dubbo.rpc.Filter文件
2. 文件配置每行Name=FullClassName，必须是实现Filter接口
3. @Activate标注扩展能被自动激活
4. @Activate如果group（provider|consumer）匹配才被加载
5. @Activate的value字段标明过滤条件，不写则所有条件下都会被加载，写了则只有dubbo URL中包含该参数名且参数值不为空才被加载

如下是dubbo rpc access log的过滤器，仅对服务提供方有效，且参数中需要带accesslog，也就是配置protocol或者serivce时配置的accesslog="d:/rpc_access.log"

```
@Activate(group = Constants.PROVIDER, value = Constants.ACCESS_LOG_KEY)
public class AccessLogFilter implements Filter {
}
```

10.其他特性

http://alibaba.github.io/dubbo-doc-

static/User+Guide-zh.htm#UserGuide-zh-%3Cdubbo%3Amonitor%2F%3E

可关注以上链接内容，dubbo提供较多的辅助功能特性，大多目前我们暂时未使用到，后续我们这边关注到的两个特性可能会再引进来使用：

- 结果缓存，省得自己再去写一个缓存，对缓存没有特殊要求的话直接使用dubbo的好了
- 分组合并，对RPC接口不同的实现方式分别调用然后合并结果的一种调用模式，比如我们要查用户是否合法，一种我们要查是否在黑名单，同时我们还要关注登录信息是否异常，然后合并结果

四、前车之鉴

这个主要是在整个学习及使用过程中记录的，以及一些同事在初识过程问过我的，这边做了整理然后直接列举在下面：

1.服务版本号

- 引用只会找相应版本的服务

```
<dubbo:serviceinterface="com.xxx.XxxService" ref="xxxService" version="1.0" />
<dubbo:referenceid="xxxService" interface="com.xxx.XxxService" version="1.0"/>
```

- 为了今后更换接口定义发布在线时，可不停机发布，使用版本号

2.暴露一个内网一个外网IP问题

~~为了在测试环境提供一个内网访问的地址和一个办公区访问的地址。~~

~~•增加一个指定IP为内网地址的服务协议~~

~~•增加一个不指定IP的服务协议，但是在/etc/hosts中hostname对应的IP要为外网IP~~

上面这种方案是一开始使用的方案，后面发现dubbo在启动过程无论是否配路由还是会一个个去连接，虽然不影响启动，但是由于存在超时所以会影响启动时间，而且每台机器还得特别配置指定IP，后面使用另外一套方案：

1. 服务不配置ip，绑定到0.0.0.0，自动获取保证获取到是内网IP注册到注册中心

即可，如果不是想要的IP，可以在/etc/hosts中通过绑定Hostname指定IP

2. 内网访问方式通过注册中心或者直连指定内网IP和端口

3. 外网访问方式通过直连指定外网IP和端口

使用这种方式需要注意做好防火墙控制等，比如在线默认也是不指定IP，会绑定在0.0.0.0，如果非法人员知道调用的外网IP和端口，而且可以直接访问就麻烦了（如果在应用中做IP拦截也成，需要注意有防范措施）。

3.dubbo reference注解问题

前文介绍使用时已经提到过，@Reference只能在spring bean实例对应的当前类中使用，暂时无法在父类使用；如果确实要在父类声明一个引用，可通过配置文件配置dubbo:reference，然后在需要引用的地方跟引用spring bean一样就行

4.服务超时问题

目前如果存在超时，情况基本都在如下几点：

- 客户端耗时大，也就是超时异常时的client elapsed xxx，这个是从创建Future对象开始到使用channel发出请求的这段时间，中间没有复杂操作，只要CPU没问题基本不会出现大耗时，顶多1ms属于正常
- IOThread繁忙，默认情况下，dubbo协议一个客户端与一个服务提供者会建立一个共享长连接，如果某个客户端处于特别繁忙而且一直往一个服务提供者塞请求，可能造成IOThread阻塞，一般非常特殊的情况才会出现
- 服务端工作线程池中线程全部繁忙，接收消息后塞入队列等待，如果等待时间比预想长会引起超时
- 网络抖动，如果上述情况都排除了，还出现在请求发出后，服务接收请求前超过预想时间，只能归类到网络抖动了，需要SA一起查看问题
- 服务自身耗时大，这个需要应用自身做好耗时统计，当出现这种情况的时候需要用数据来说明问题及规划优化方案，建议采用缓存埋点的方式统计服务中各个执行阶段的耗时情况，最终如果超过预想时间则把缓存统计的耗时情况打日志，减少日志量，且能够得到更明确的信息

现在我们应用使用过程中发现两种类型的耗时，一种我们目前只能归类到网络抖动，后续需要找运维一起关注这个问题，另外一种是由于一些历史原因，数据库查询容易发生抖动，总有一个时间点会突然多出很多超时。

5.服务保护

服务保护的原则上是避免发生类似雪崩效应，尽量将异常控制在服务周围，不要扩散开。

说到雪崩效应，还得提下dubbo自身的重试机制，默认3次，当失败时会进行重试，这样在某个时间点出现性能问题，然后调用方再连续重复调用，很容易引起雪崩，建议的话还是很据业务情况规划好如何进行异常处理，何时进行重试。

服务保护的话，目前我们主要从以下几个方面来实施，也不成熟，还在摸索：

- 考虑服务的dubbo线程池类型（fix线程池的话考虑线程池大小）、数据库连接池、dubbo连接数限制是否都合适
- 考虑服务超时时间和重试的关系，设置合适的值
- 一定时间内服务异常数较大，则可考虑使用failfast让客户端请求直接返回或者让客户端不再请求

经领导推荐，还在学习Release it，后续有其他想法，再回头来编辑。

6.zkclient的问题

前文已经提到过zkclient有两个问题，修改服务器时间会导致容器挂掉；dubbo使用zkclient没有传超时时间导致zookeeper无法连接的时候，直接阻塞Integer.MAX_VALUE。

正在调研curator，目前只能说curator不会在无法连接的时候直接阻塞。

另外zkclient和curator的jar包应该都是jdk1.6编译的，所以系统还在jdk1.5以下的话无法使用。

7.注册中心的分组group和服务的不同实现group

这两个东西完全不同的概念，使用的时候不要弄混了。

registry上可以配置group，用于区分不同分组的注册中心，比如在同一个注册中心下，有一部分注册信息是要给开发环境用的，有一部分注册信息时要给测试环境用的，可以分别用不同的group区分开，目前对这个理解还不透彻，大致就是用于区分不同环

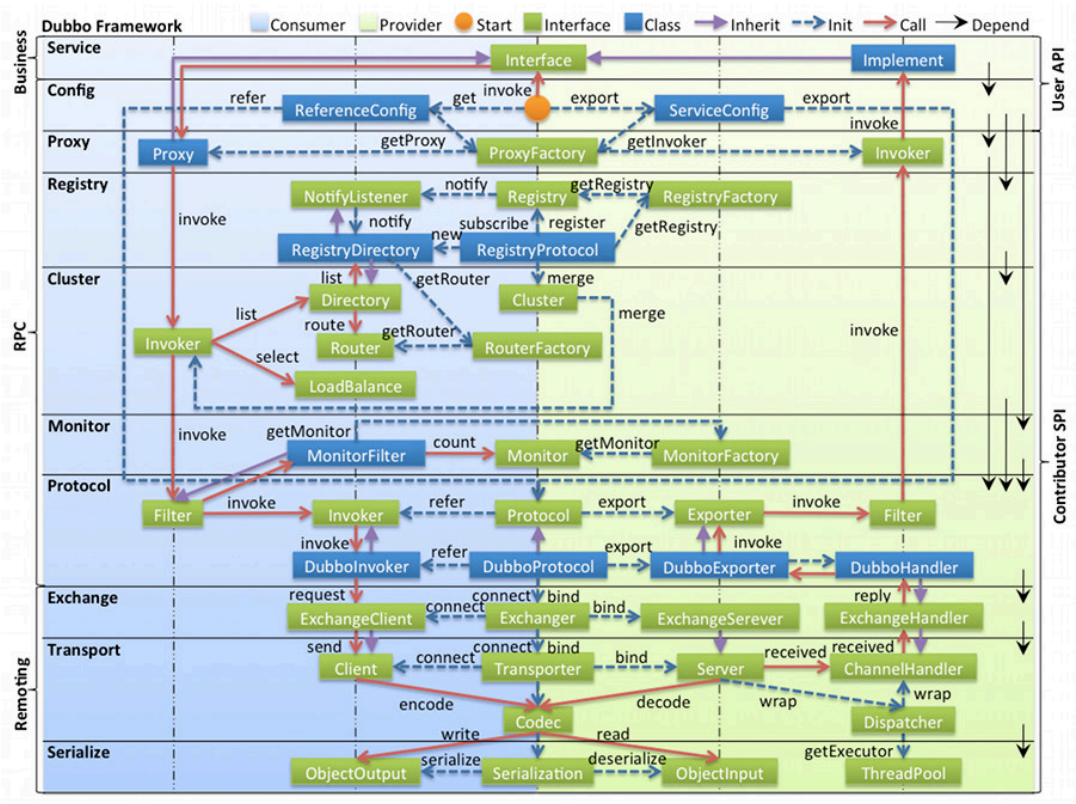
境。

service和reference上也可以配置group，这个用于区分同一个接口的不同实现，只有在reference上指定与service相同的group才会被发现，还有前文提到的分组合并结果也是用的这个。

五、dubbo如何工作的

其实dubbo整个框架内容并不算大，仔细看的话可能最多两天看完一遍，但是目前还是没领悟到怎么做到的扩展性，学习深度还不够~

要学习dubbo源码的话，必须要拿出官方高清大图才行。



这张图看起来挺复杂的样子，真正拆分之后对照源码来看会发现非常清晰、简单直观。

1.如何跟进源码

入口就是各种dubbo配置项的解析，<dubbo:xxx />都是spring namespace，可以看到dubbo jar包下META-INF里面的spring.handlers，自定义的spring namespace处理器。

对于spring不太熟的同学可以先了解下这个功能，入口都在这里，解析成功后每个

<dubbo:xxx />配置项都对应一个spring实例。

2.服务提供者

首先把这张图拆分成三块，首先是服务端剖去网络传输模块，也就是大图中的右上角。

这里主要抽几个主要的类，从服务初始化到接收消息的流程简单说明下，有兴趣的再对照源码看下会比较清晰。

- ServiceBean

继承ServiceConfig，做为服务配置管理和配置信息校验，每一个dubbo:service配置或者注解都会对应生成一个ServiceBean的实例，维护当前服务的配置信息，并把一些全局配置塞入到该服务配置中。
另外ServiceBean本身是一个InitializingBean，在afterPropertiesSet时通过配置信息引导服务绑定和注册。
可以留意到ServiceBean还实现了ApplicationListener，在全部spring bean加载完成后判断是否延迟加载的逻辑。

- ProtocolFilterWrapper

经过serviceBean引导后进入该类，这个地方注意下，Protocol使用的装饰模式，叶子只有DubboProtocol和RegistryProtocol，在中间调用中会绕来绕去，而且registry会走一遍这个流程，然后在RegistryProtocol中暴露服务再走一遍，注意每个类的作用，不要被绕昏了就行，第一次跟进代码的时候没留意就晕头转向的。
在这之前其实还有个ProtocolListenerWrapper，封装监听器，在服务暴露后通知到监听器，没有复杂逻辑，如果没特殊需求可以先绕过。
再来说ProtocolFIlterWrapper，这个类的作用就是串联filter调用链，如果有看过struts或者spring mvc拦截器源码的应该不会陌生。

- RegistryProtocol

注册中心协议，如果配置了注册中心地址，每次服务暴露肯定首先引导进入这个类中，如果没有注册中心连接则会先创建连接，然后再引导真正的服务协议暴露流程，会再走一次ProtocolFilterWrapper的流程（这次引导到的叶子是DubboProtocol）。
在服务暴露返回后，会再执行服务信息的注册和订阅操作。

- DubboProtocol

这个类的export相对较简单，就是引导服务bind server socket。
另外该类还提供了一个内部类，用于处理接收请求，就是下面要提到的

ExchangeHandler。

- DubboProtocol$ExchangeHandler

接收反序列化好的请求消息，然后根据请求信息找到执行链，将请求再丢入执行链，让其最终执行到实现类再将执行结果返回即整个过程完成。

3.客户端

客户端模块与服务端模块比较类似，只是刚好反过来，一个是暴露服务，一个是引用服务，然后客户端多出路由和负载均衡。

- ReferenceBean

继承ReferenceConfig，维护配置信息和配置信息的校验，该功能与ServiceBean类似
其本身还实现了FactoryBean，作为实例工厂，创建远程调用代理类；而且如果不指定为init的reference都是在首次getBean的时候调用到该factoryBean的getObject才进行初始化
另外实现了InitializingBean，在初始化过程中引导配置信息初始化和构建init的代理实例

- InvokerInvocationHandler

看到这个类名应该就知道是动态代理的handler，这里作为远程调用代理类的处理器在客户端调用接口时引导进入invoker调用链

- ProtocolFIlterWrapper

与Service那边的功能类似，构建调用链

- RegistryProtocol

与service那边类似，如果与注册中心还没有连接则建立连接，之后注册和订阅，再根据配置的策略返回相应的clusterInvoker
比service那边有个隐藏较深的逻辑需要留意的，就是订阅过程，RegistryDirectory作为订阅监听器，在订阅完成后会通知到RegistryDirectory，然后会刷新invoker，进入引导至DubboProtocol的流程，与变更的service建立长连接，第一次发生订阅时就会同步接收到通知并将已存在的service存到字典

- DubboProtocol

在订阅过程中发现有service变更则会引导至这里，与服务建立长连接，整个过程为了得到串联执行链Invoker

- ClusterInvoker

ClusterInvoker由RegistryProtocol构建完成后，内部封装了Directory，在调用时会从Directory列举存活的service对应的Invoker,Directory作为被通知对象，在service有变更时也会及时得到通知
调用时在集群中发现存在多节点的话都会通过clusterInvoker来根据配置抉择最终调用的节点，包括路由方式、负载均衡等
dubbo本身支持的节点调用策略包括比如failoverClusterInvoker在失败时进行重试其他节点，failfastClusterInvoker在失败时返回异常，mergeableClusterInvoker则是对多个实现结果进行合并的等等很多

- DubboInvoker

承接上层的调用信息，作为调用结构的叶子，将信息传递到exchange层，主要用来和echange交互的功能模块

4.网络传输层

从exchange往下都是算网络传输，包括做序列化、反序列化，使用Netty等IO框架发送接收消息等逻辑，先前看的时候没有做统一梳理，后续有机会再来编辑吧。