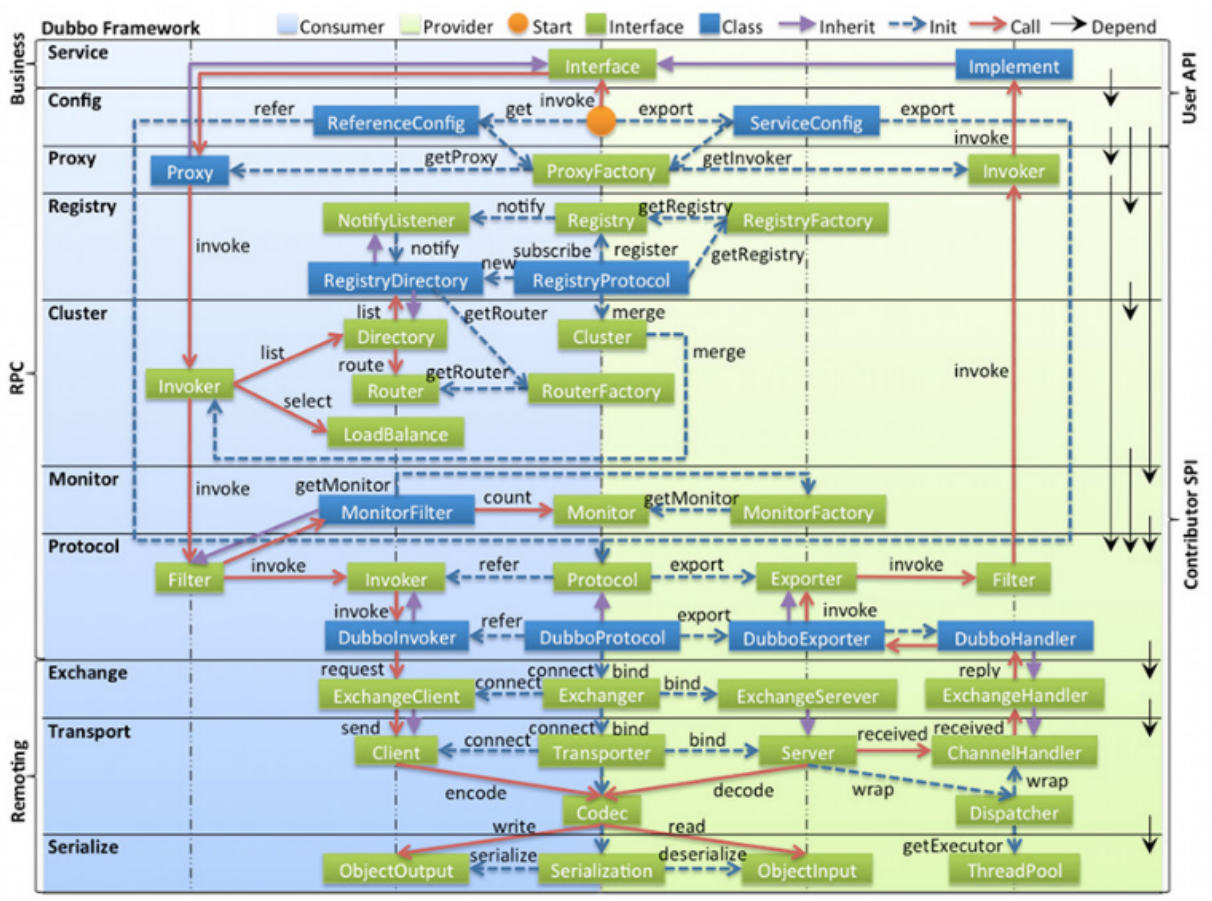
dubbo框架整体设计



Service: 面向接口编程，写一个接口和实现类
RPC: 就是完成远程过程调用
Config: 配置层
Proxy: 代理层
Registry: 注册中心层，完成服务的发现与服务的注册
Cluster: 路由层及负载均衡层 **Monitor:** 监控层
Protocol: 远程调用层，封装RPC调用
Remoting: 远程通信层
Exchange: 信息交换层，服务端，客户端之间的信息交换
Transport: 传输层
Serialize: 序列化层

1.最顶上九个图标，代表本图中的对象和流程。

2.图中左边淡蓝色（Consumer）为服务消费者使用的接口，右边淡绿色（Provider）为服务提供方使用的接口，位于中轴线上的为双方都使用的接口。

3.图中从下至上分为十层，各层均为单向依赖，右边的黑色箭头（Depend）代表层之间的依赖关系，每一层都可以剥离上层被复用，其中，Service和Config层为API,其他各层均为SPI.

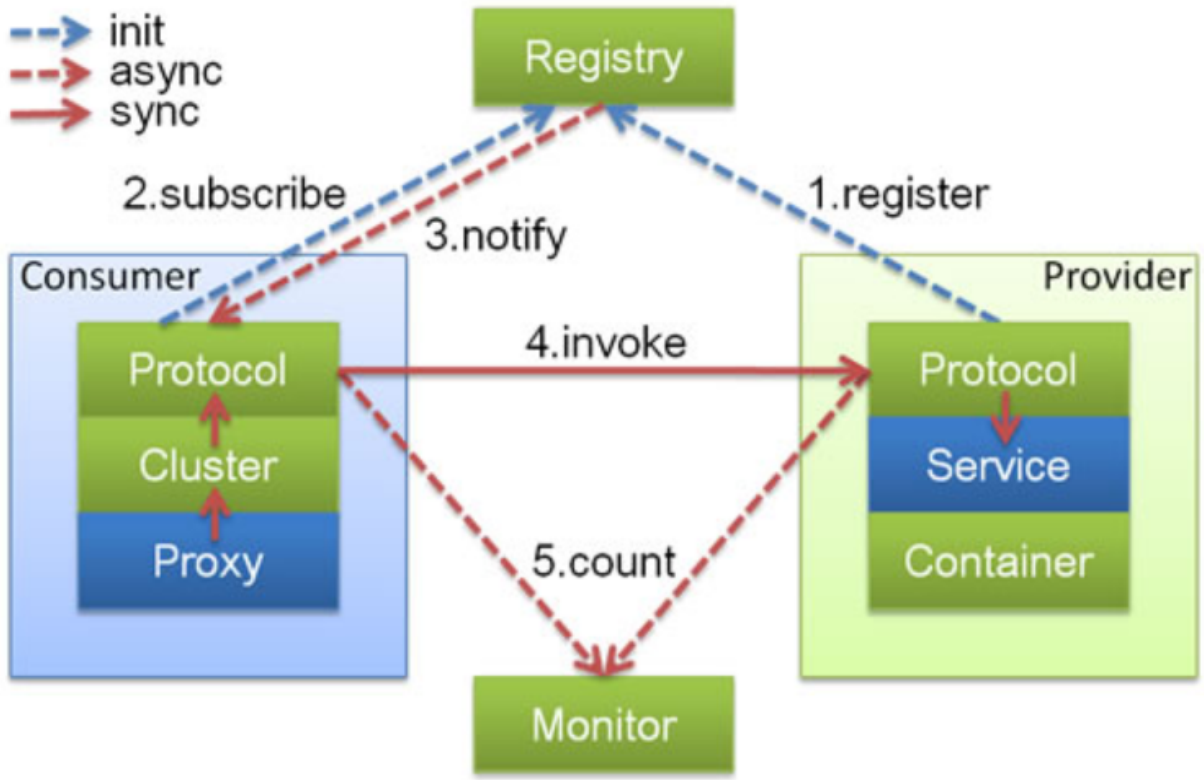
3.图中绿色小块（Interface）为扩展接口，蓝色小块（Class）为实现类，图中只显示用于关联各层的实现类。

4.图中蓝色虚线（init）为初始化过程，即启动时组装链，红色实现（Call）为方法调用过程，即运行时调时链。紫色三角箭头（Inherit）为继承，可以把子类看做父类的同一个节点，线上的文字为地搜用的方法。

Dubbo中的URL统一模型

下面一直提到一个词语就是URL,我在网上找到了Dubbo之URL详解，点击查看

dubbo依赖关系

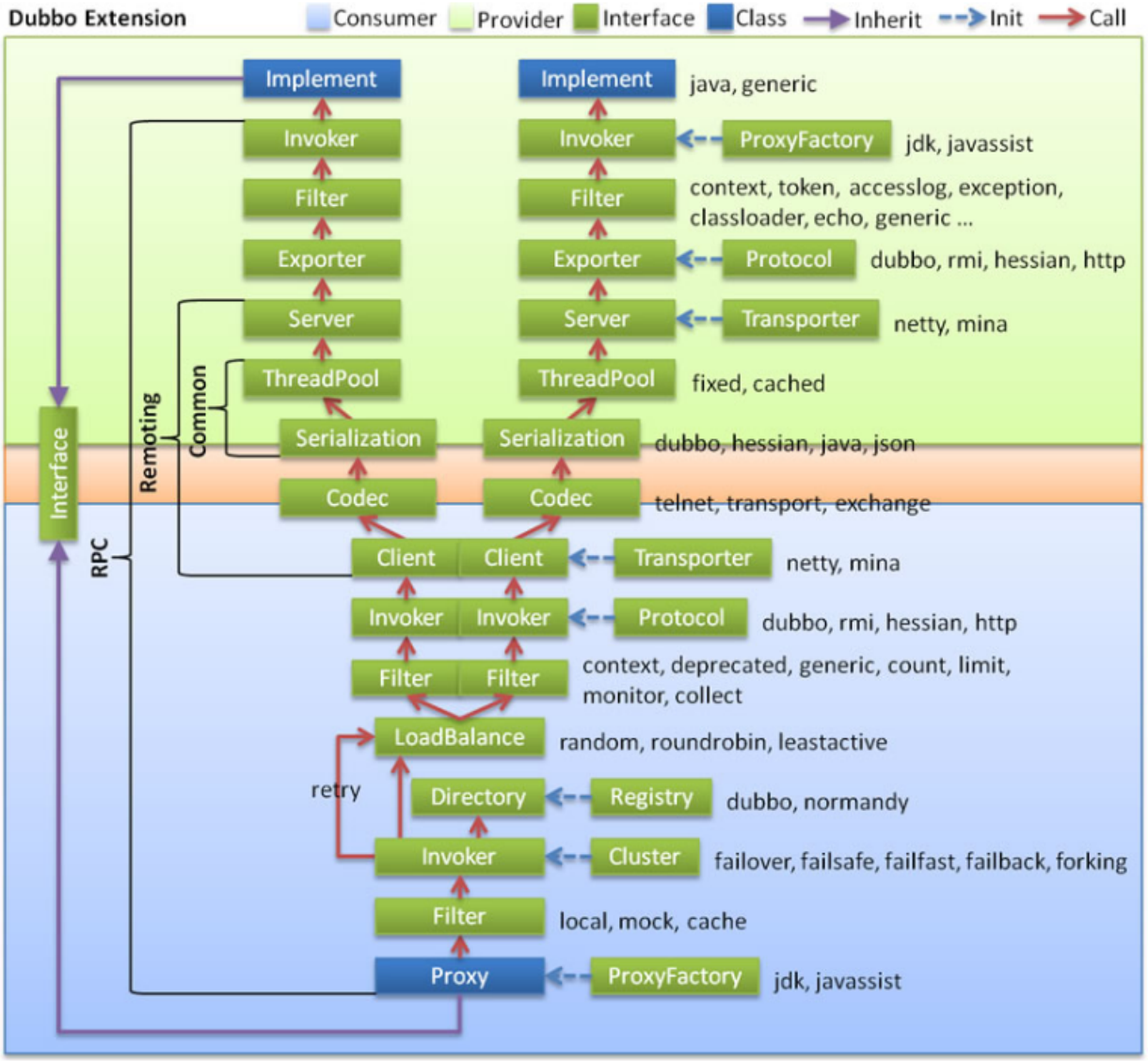


图例说明：

- 图中小方块Protocol,Cluster,Proxy,Service,Container,Registry,Monitor代表层或模块，蓝色的表示与业务有交互，绿色的表示只对Dubbo内部交互。
- 图中北京方块Consumer,Provider,Registry,Monitor代表部署逻辑拓扑节点。
- 图中蓝色虚线为初始化时调用，红色虚线为运行时异步调用，红色实线为运行时同步调用。
- 图中只包含RPC的层，不包含Remoting的层，Remoting整体都隐含在Protocol中。

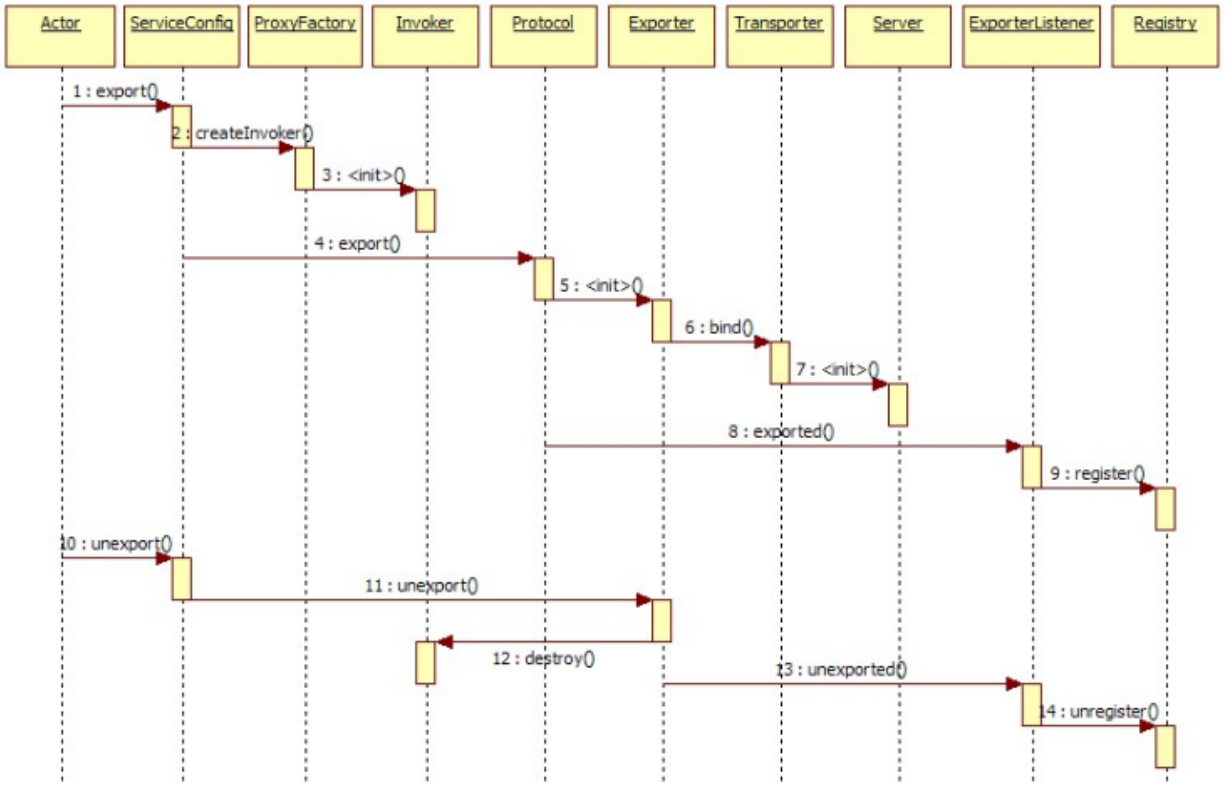
调用链

展开总设计图的红色调用链，如下：



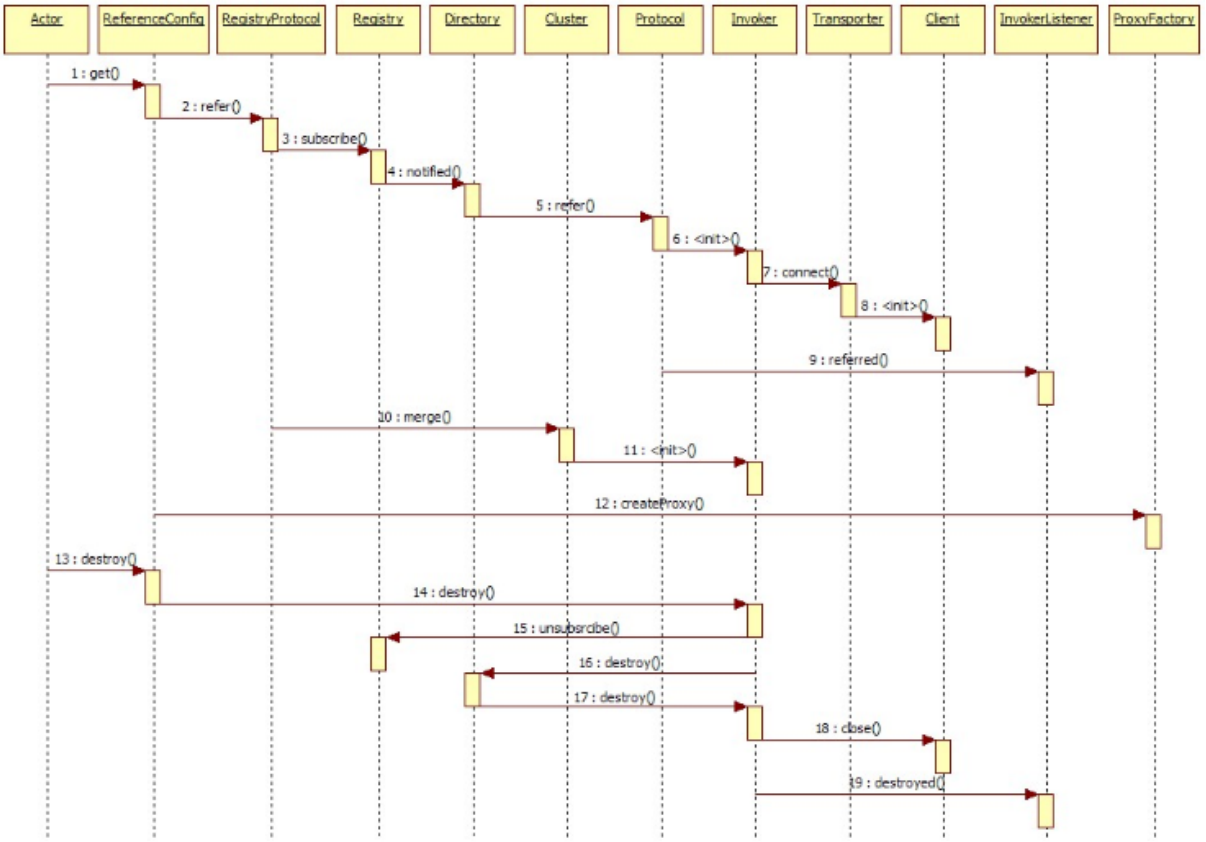
提供方暴露时序图

展开上面dubbo依赖关系右边提供方暴露服务的蓝色初始化链，时序图如下：（官网这块描述有误，看的时候一定要自己思考）



消费方引用时序图

展开上面dubbo依赖关系左边消费方引用服务的蓝色初始化链，时序图如下：（官网这块描述有误，看的时候一定要自己思考）



初始化过程细节

1.解析服务

基于dubbo.jar内的META-INF/spring.handlers配置，Spring在遇到dubbo名称空间时，会回调DubboNamespaceHandler.

所有dubbo的标签，都统一用DubboBeanDefinitionParser进行解析，基于一对一属性映射，将xml标签解析为Bean对象。

在ServiceConfig.export()或ReferenceConfig.get()初始化时，将Bean对象转换URL格式，所有Bean属性转化为URL的参数。

2.暴露服务

1).只暴露服务端口：
在没有注册中心，直接暴露提供者的情况下，ServiceConfig解析出的URL的格式为：
`dubbo://service-host/com.foo.FooService?version=1.0.0`（service-host是service的主机地址，例如：我的阿里云地址：119.23.108.42）
基于扩展点自适应机制(在扩展类的jar包内，放扩展点配置文件META-INF/dubbo/接口全限定名)，内容为:配置名=扩展实现类全限定名，多个实现类用换行符分隔

2).向注册中心暴露服务：
在有注册中心，需要注册提供者地址的情况下，ServiceConfig解析出的URL的格式为：
`registry://registry-host/org.apache.dubboregistry.RegistryService?export=URL.encode("dubbo://service-host/com.foo.FooService?version=1.0.0")`

基于扩展点自适应机制，通过URL的registry://协议头，就会调用RegistryProtocol的export()方法，将export参数中的提供者URL,先注册到注册中心。

再重新传给Protocol扩展点进行暴露：`dubbo://service-host/com.foo.FooService?version=1.0.0`,然后基于扩展点自适应机制，通过提供者URL的dubbo://协议头识别，就会调用DubboProtocol的export()方法，打开服务端口。

3.引用服务

1).直连引用服务
在没有注册中心，直连提供者的情况下，ReferenceConfig解析出来的URL的格式为：
`dubbo://service-host/com.foo.FooService?version-1.0.0`

基于扩展点自适应机制，通过URL的dubbo://协议识别，直接调用DubboProtocol的refer()方法，返回提供者引用。

2).从注册中心发现引用服务

在有注册中心，通过注册中心发现提供者地址的情况下，ReferenceConfig解析出的URL的格式为：
`registry://registry-host/org.apache.dubbo.registry.RegistryService?refer=URL.encode("consumer://consumer-host/com.foo.FooService?version=1.0.0")`

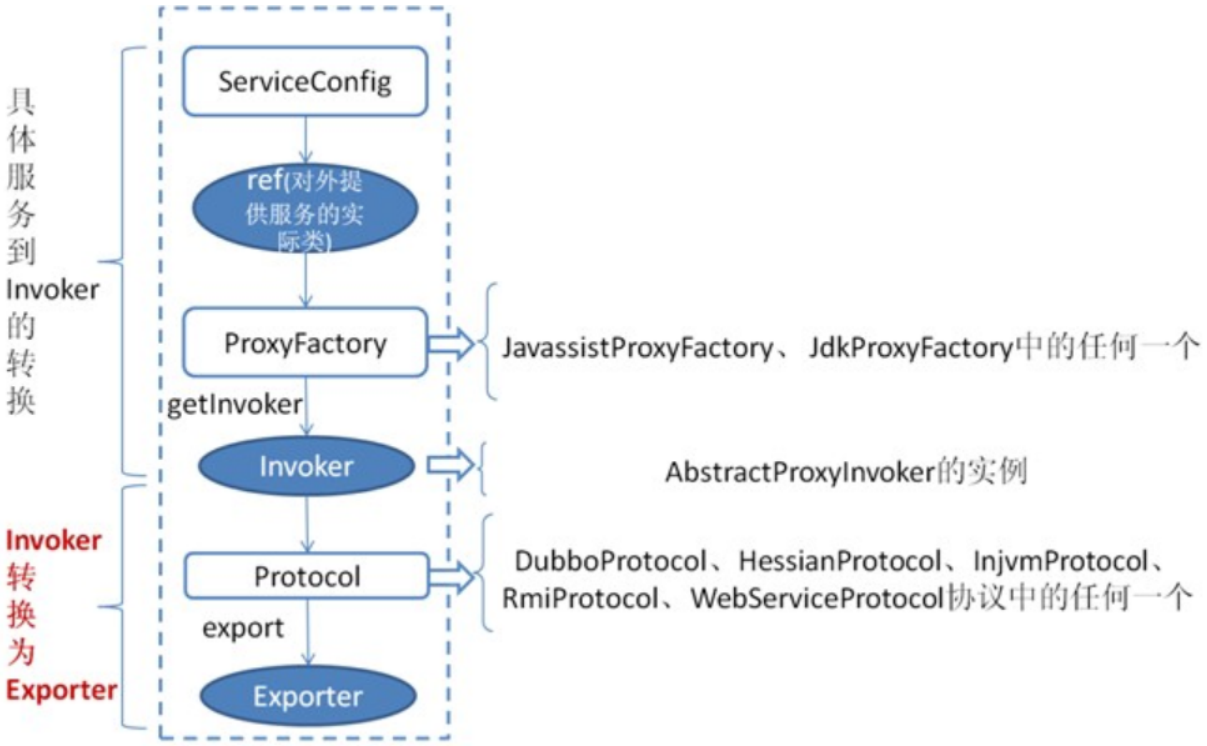
基于扩展点自适应机制，通过URL的registry://协议头识别，就会调用RegistryProtocol的`refer()`方法,基于refer参数中的条件，查询提供者URL,如：`dubbo://service-host/com.foo.FooService?version=1.0.0`

基于扩展点自适应机制，通过提供者URL的dubbo://协议识别，就会调用DubboProtocol的refer()方法，得到提供者引用。

然后RegistryProtocol将多个提供者引用，通过cluster扩展点，伪装成单个提供者引用返回。

远程调用细节

服务提供者暴露一个服务的详细过程



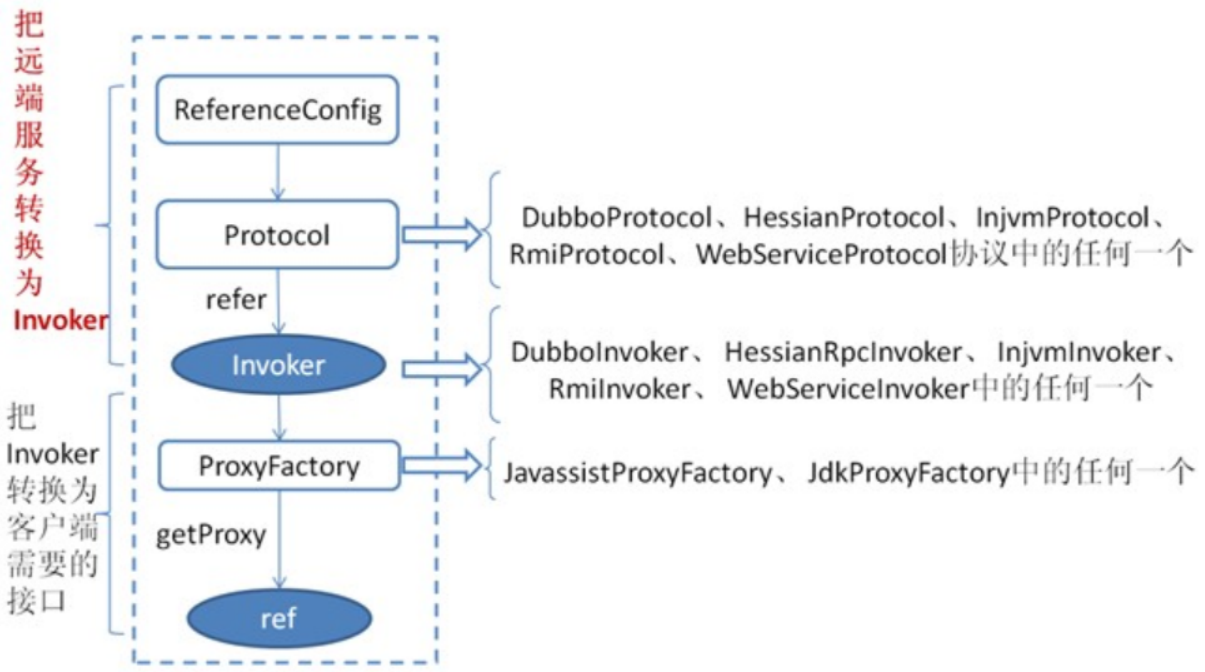
上图是服务提供者暴露服务的主过程：

首先ServiceConfig类拿到对外提供服务的 实际类ref(如：HelloWorldImpl),然后通过ProxyFactory类的getInvoker方法使用ref生成一个AbstractProxyInvoker实例，到这一步就完成具体服务到Invoker的转化，接下来就是Invoker转换为Exporter的过程。

Dubbo处理服务暴露的关键就在Invoker转换到Exporter的过程，上图中的红色部分，我主要以Dubbod这种典型协议的实现来说明一下：

DUbbo协议的Invoker转为Exporter发生在DubboProtocol类的export方法，它主要是打开socket侦听服务，并接收客户端发来的各种请求，通讯细节由DUbbo自己实现。

服务消费者消费一个服务的详细过程：

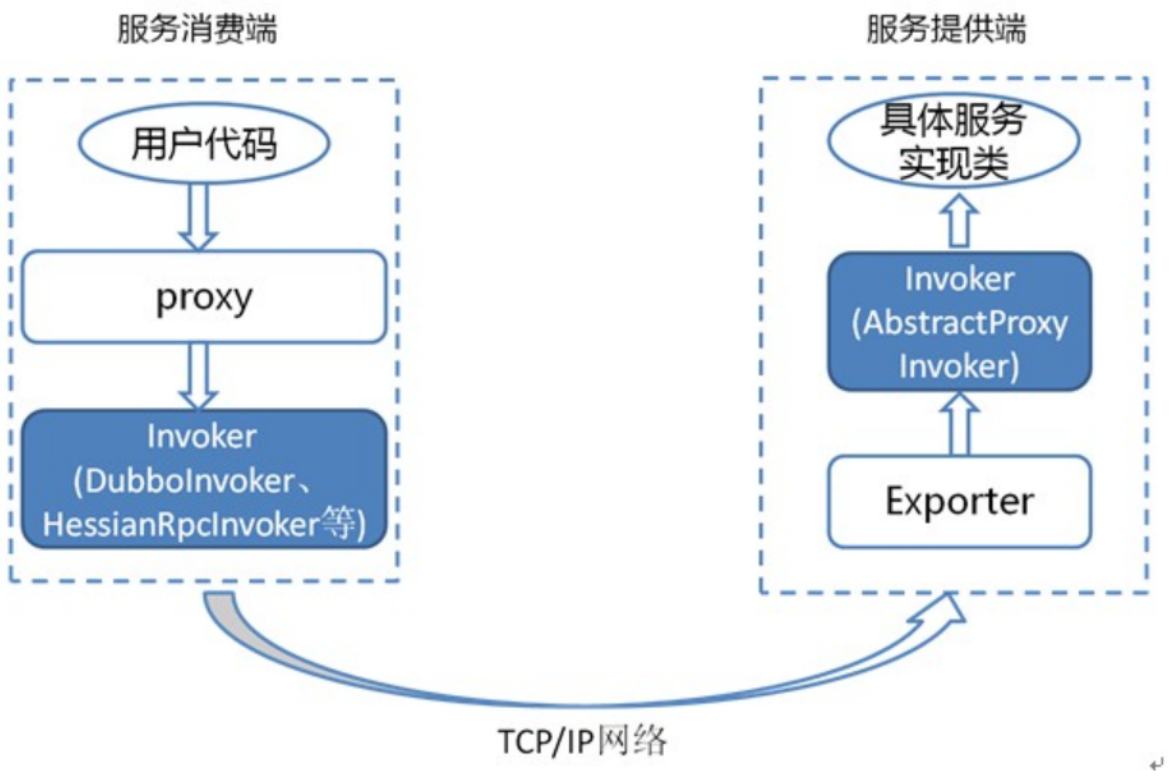


上图是服务消费的主过程：

首先ReferenceConfig类的init方法调用Protocol的refer方法生成Invoker实例（如上图中的红色部分），这是服务消费的关键，接下来把Invoker转换为客户端需要的接口（如：HelloWorld）。

满眼都是Invoker

由于Invoker是DUbbo领域模型中非常重要的一个概念，很多设计思路都是向它靠拢，这就使得Invoker渗透在整个实现代码里，对弈刚开始接触Dubbo的人，确实容易给搞混了。下面我们用一个精简的图来说明最重要的两种Invoker:服务提供Invoker和服务消费Invoker



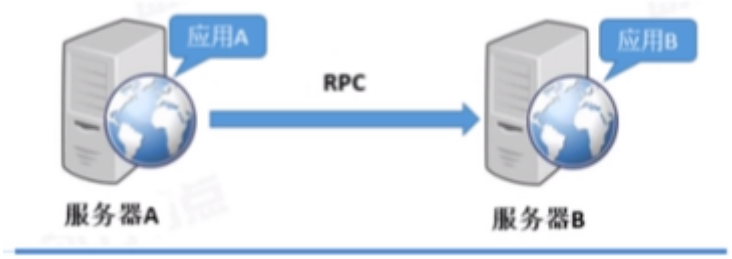
RPC是什么

RPC: Remote Procedure Call远程过程调用

RPC的核心就是序列化和反序列化

序列化 是将对象的状态信息转换为可以存储或传输形式的过程，一般讲一个对象存储至一个储存媒介，例如档案或是记忆体缓冲等。在网络传输过程中，可以是字节或者xml等格式；而字节的或xml编码格式可以还原成完全相同的对象，这个过程就是反序列化。

比如有两台服务器A和B,一个应用部署到A服务器，另一台用用部署到B服务器上，如果A应用想要调用B应用提供的方法，由于他们不在一台机器上，也就是他们不在一个JVM内存空间，无法直接调用，需要通过网络进行调用，那么这个调用过程就是RPC。如下图：



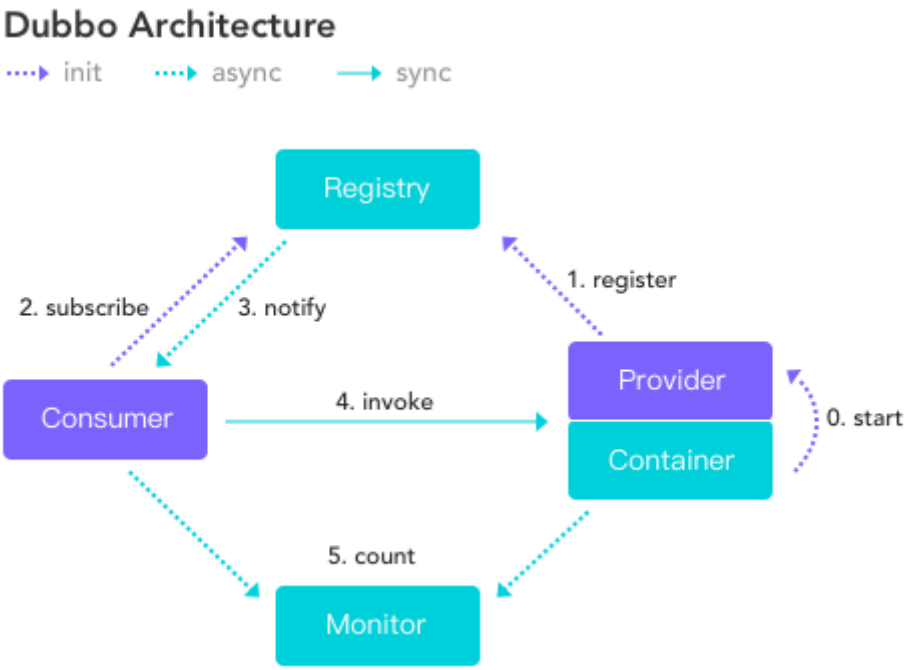
同一个JVM下，一个方法可以直接调用另一个方法。
不同的JVM,一个项目的方法通过RPC调用另一个项目的方法。
RPC调用过程如何实现：
1.基于tcp协议的RPC:
1).序列化与反序列化
2).socket 3).反射

2.基于http协议的RPC: 1).xml
2).json
3).http,如下图：

dubbo已经帮我们实现了这一切，但dubbo不仅仅是rpc,还有服务治理

Dubbo的整体架构

Dubbo五大核心部件：
1.Provider:提供服务的服务提供方（生产者）
2.Consumer:调用远程服务的服务消费方（消费者）
3.Registry:服务注册与发现的注册中心（注册中心）
4.Monitor:统计服务调用次数和调用时间的监控中心（监控中心）
5.Container:服务运行容器（运行容器）



0：容器启动
1.服务之策
2.消费者去注册中心订阅
3.当提供者新增方法，注册中心通知消费者

4.消费者调用提供者
5.监控中心统计调用次数，时间

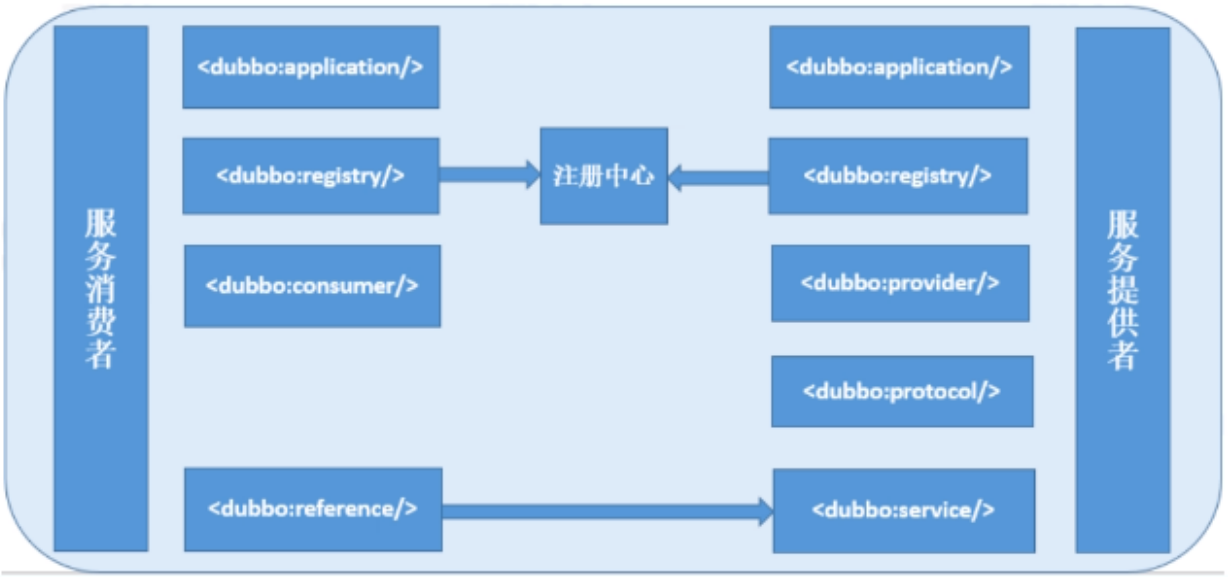
创建dubbo项目一定要在pom.xml中添加如下依赖：

```
<dependency>
     <groupId>com.alibaba</groupId>
     <artifactId>dubbo</artifactId>
     <version>2.6.2</version>
   </dependency>
```

dubbo协议

1.**dubbo://**
1).Dubbo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远大于服务提供者机器的情况。
2).Dubbo协议底层默认使用的是netty,性能非常优秀，官方推荐使用此协议。
3).Dubbo协议不适合大数据量的服务，比如传文件，传视频等，除非请求量很低
4).<dubbo:protocal name="dubbo" port="20880"/>,dubbo协议默认端口是20880

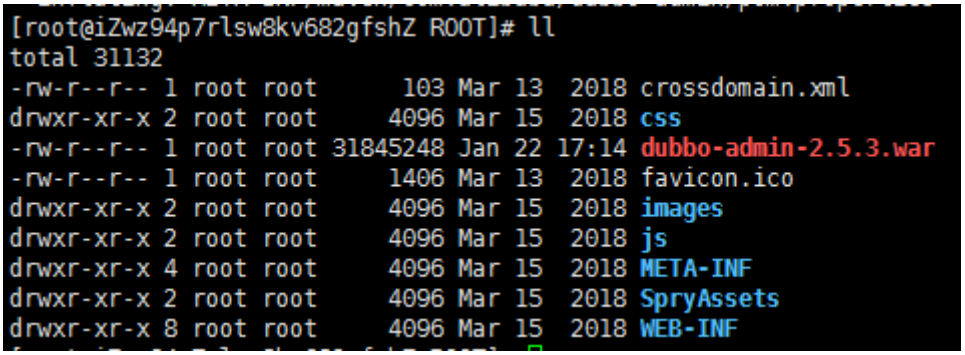
dubbo各个标签的图示解说：



1.公共标签：
1).<dubbo://application>
2).<dubbo://registry>

2.服务提供者标签：
1).<dubbo:provider>
2).<dubbo://protocol>
3).<dubbo://service>

3.服务消费者标签：
1).<dubbo:consumer>
2).<dubbo:reference>



灰度发布

基于Dubbo服务的治理，是否可以支持业务级别的灰度发布、是否基于业务参数的路由转发。例如以GIS为例，当发布一个新版本时，是否可以以按照解析地址或合作伙伴来区分，版本发布之初，只希望地址为：广东省的解析请求发送到新版本，而其他的地址请求还是使用旧版；或者根据合作伙伴例如UCP(优享寄)的请求转发到新版本服务器，其他合作伙伴还是转发到旧版，实现业务级别的灰度发布，控制新版本的影响范围。例如OMS系统，可以根据合作伙伴，将重量级客户的请求转发到单独的服务器集群，确保其高可用。

高可用

zookeeper注册中心宕机，还可以消费dubbo暴露的服务

原因：
1.监控中心宕掉不影响使用，只是丢失部分数据。
2.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。
3.注册中心对等集群，任意一台宕掉后，将自动切换到另一台。
4.注册中心全部宕机后，服务提供者和服务消费者仍能通过本地缓存通讯。
5.服务提供者无状态，任意一台宕掉后，不影响使用。
6.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。

服务降级

当服务压力剧增的情况下，根据实际业务情况及流量，对一些服务和页面有策略的不处理或换种简单的方式处理，从而释放服务器资源以保证核心交易正常运作或高效运作。

一次完整的RPC调用流程(同步调用)

1.服务消费方(client)调用以本地调用方式调用服务
2.client stub接收到调用后负责将方法，参数等组装成能够进行网络传输的消息体。
3.client stub找到服务地址，并将消息发送到服务端。
4.server stub收到消息后进行解码。
5.server stub根据解码结果调用本地的服务。

6.本地服务执行并将结果返回给server stub.
7.server stub将返回打包成消息并发送至消费方。
8.client stub接收到消息，并进行解码。
9.服务消费方得到最终结果。

RPC框架的目标就是要2~8这些步骤都封装起来，这些细节对用户来说是透明的，不可见。

dubbo源码拉取

点击下载dubbo源码 `https://github.com/apache/incubator-dubbo`

1.dubbo-demo是简单的dubbo项目，可以自己调试看一下，三种方式实现的

2.dubbo-common是公共逻辑模块：提供工具和通用模块

3.dubbo-remoting是远程通信模块：提供通用的客户端和服务端的通讯功能

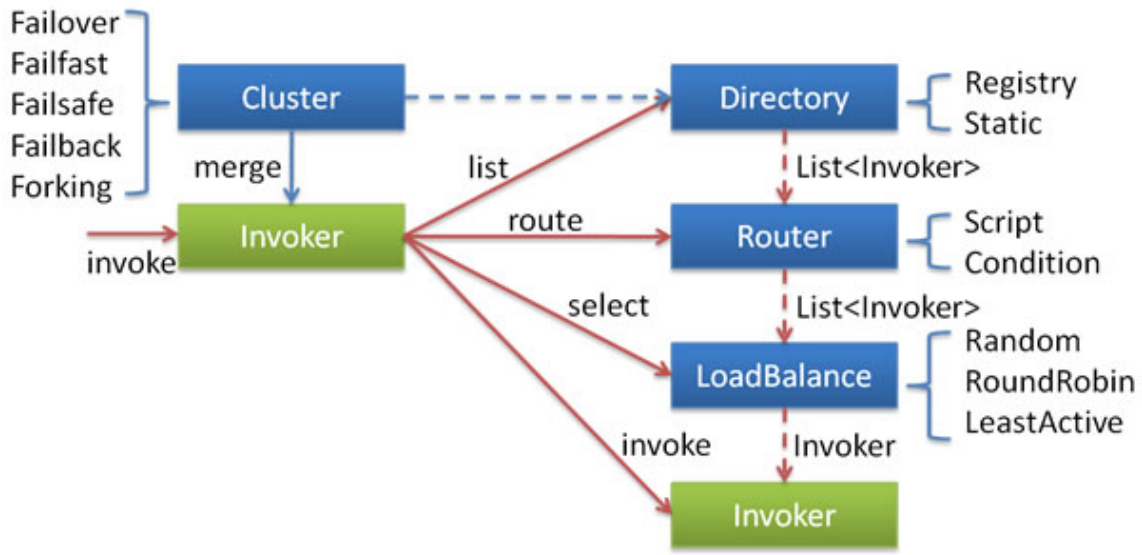
4.dubbo-rpc是远程调用模块：抽象各种协议，以及动态代理，只包含一对一的调用，不关心集群的管理（集群相关管理由dubbo-cluster提供特性）

5.dubbo-cluster是集群模块，将多个服务提供方伪装成一个提供方，包括：负载均衡，集群容错，路由，分组聚合等。集群的地址列表可以是静态配置的，也可以是由注册中心下发。

6.dubbo-registry是注册中心模块：基于注册中心下发地址的集群地址，以及对各种注册中心的抽象。

7.dubbo-monitor:监控模块：统计服务调用次数，调用时间的，调用链跟踪的服务。

集群容错



1.这里的Invoker是provider的一个可调用Service的抽象，Invoker封装了Provider地址及Service接口信息。

2.Directory代表多个Invoker,可以把它看成List<Invoker>，但与List不同的是，它的值可能是动态变化的，比如注册中心推送变更。

3.cluster将Directory中的多个Invoker伪装成一个Invoker,对上层透明，伪装过程包含了容错逻辑，调用失败后，重试另一个。

4.Router负责从多个Invoker中按路由规则选出子集，比如读写分离，应用隔离等。

5.LoadBalance负责从多个Invoker中选出具体的一个用于本次调用，选的过程包含了负载均衡算法，调用失败后，需要重选。

Invoker

1.Invoker是是实体域，它是Dubbo的核心模型，其他模型都像它靠拢，或转换成它。

2.它代表一个执行体，可向它发起invoker调用。

3.它有可能是一个本地的实现，也有可能是一个远程的实现，也有可能是一个集群实现。