从Dubbo看RPC

吴庆超 – 互联网研发部

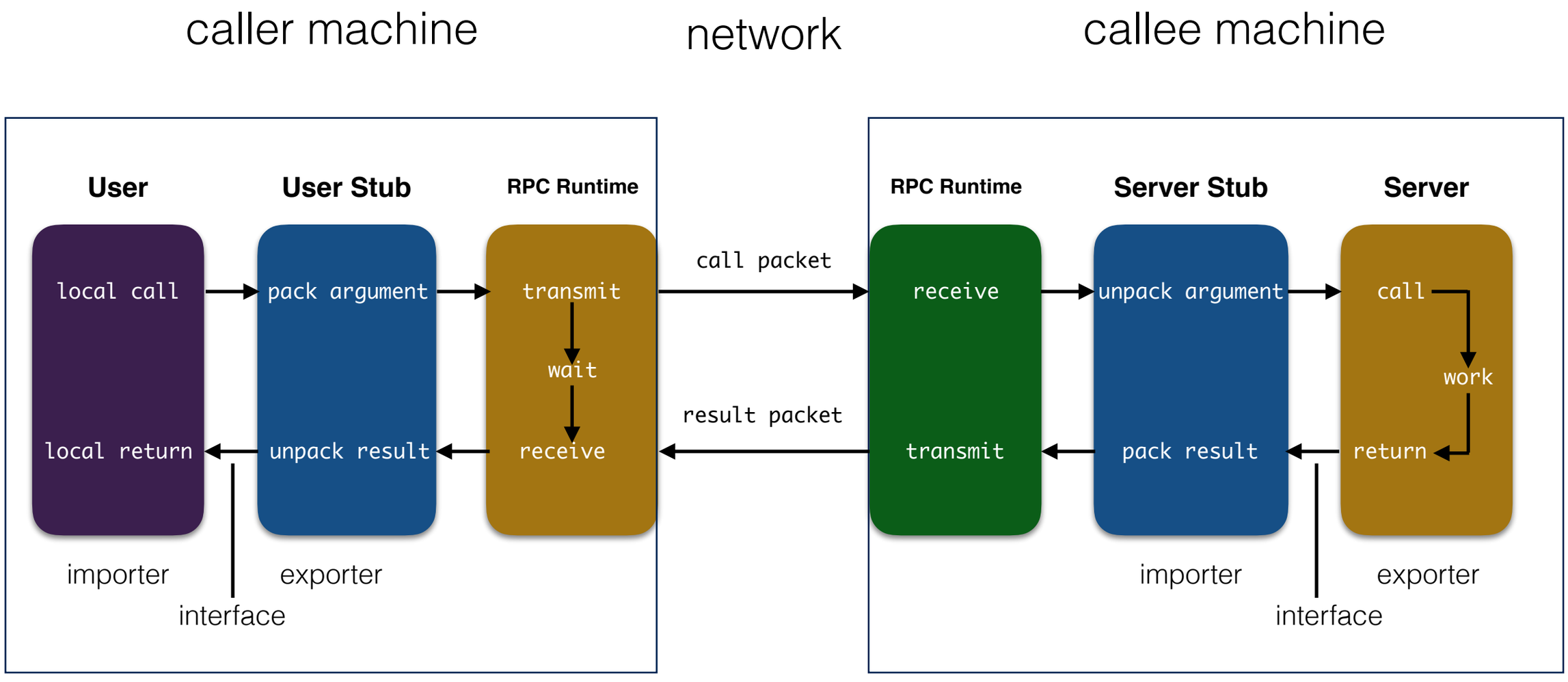
大纲

- RPC概念
- RPC模型
- RPC目标
- Dubbo模型
- Dubbo – 服务暴露
- Dubbo – 服务引用
- Dubbo – 注册中心 （注册服务 && 引用服务）
- Dubbo – 集群
- Dubbo – 负载均衡
- Dubbo – 路由
- Dubbo – ExtensionLoader （插件化）
- Dubbo – 启动流程（Spring）

RPC概念

- RPC: 远程过程调用(Remote Process Call)
- 1983年由ANDREW D. BIRRELL 和BRUCE JAY NELSON 提出[http://birrell.org/andrew/papers/ImplementingRPC.pdf]
- 调用另一个地址空间的方法（过程／函数），但不用显式编码远程调用的细节。
- 目标是更容易的构建分布式计算/应用，既能提供远程调用能力，又不失本地调用的语义简洁性。
- 为实现该目标，RPC 框架需提供一种透明调用机制让使用者不必显式的区分本地调用和远程调用。

RPC模型


caller machine
network
callee machine
User
User Stub
RPC Runtime
RPC Runtime
Server Stub
Server
local call
pack argument
transmit
call packet
receive
unpack argument
call
wait
work
result packet
local return
unpack result
receive
transmit
pack result
return
importer
exporter
interface
importer
exporter
interface

RPC目标

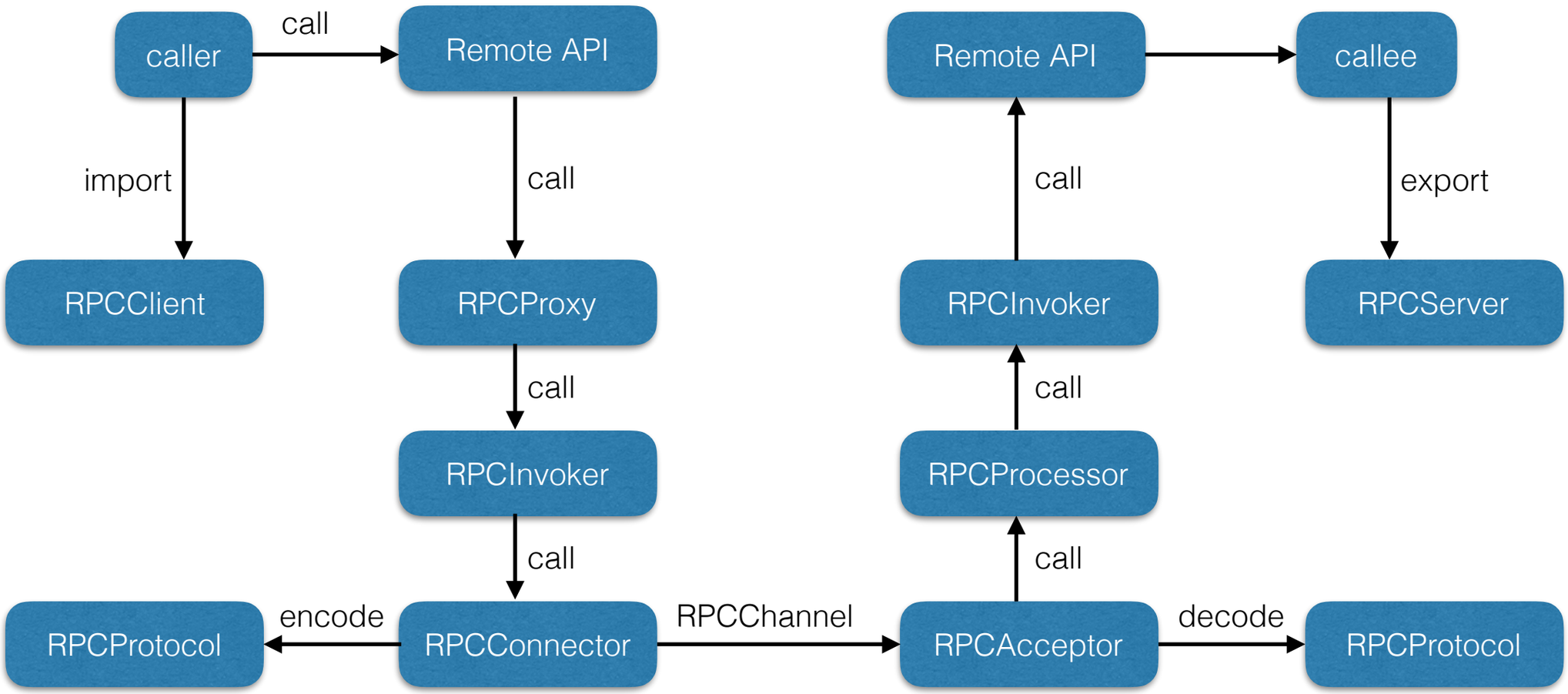
- 简单： RPC 概念的语义十分清晰和简单，这样建立分布式计算（系统）就更容易
- 高效：过程调用看起来十分简单而且高效。
- 安全：

Dubbo

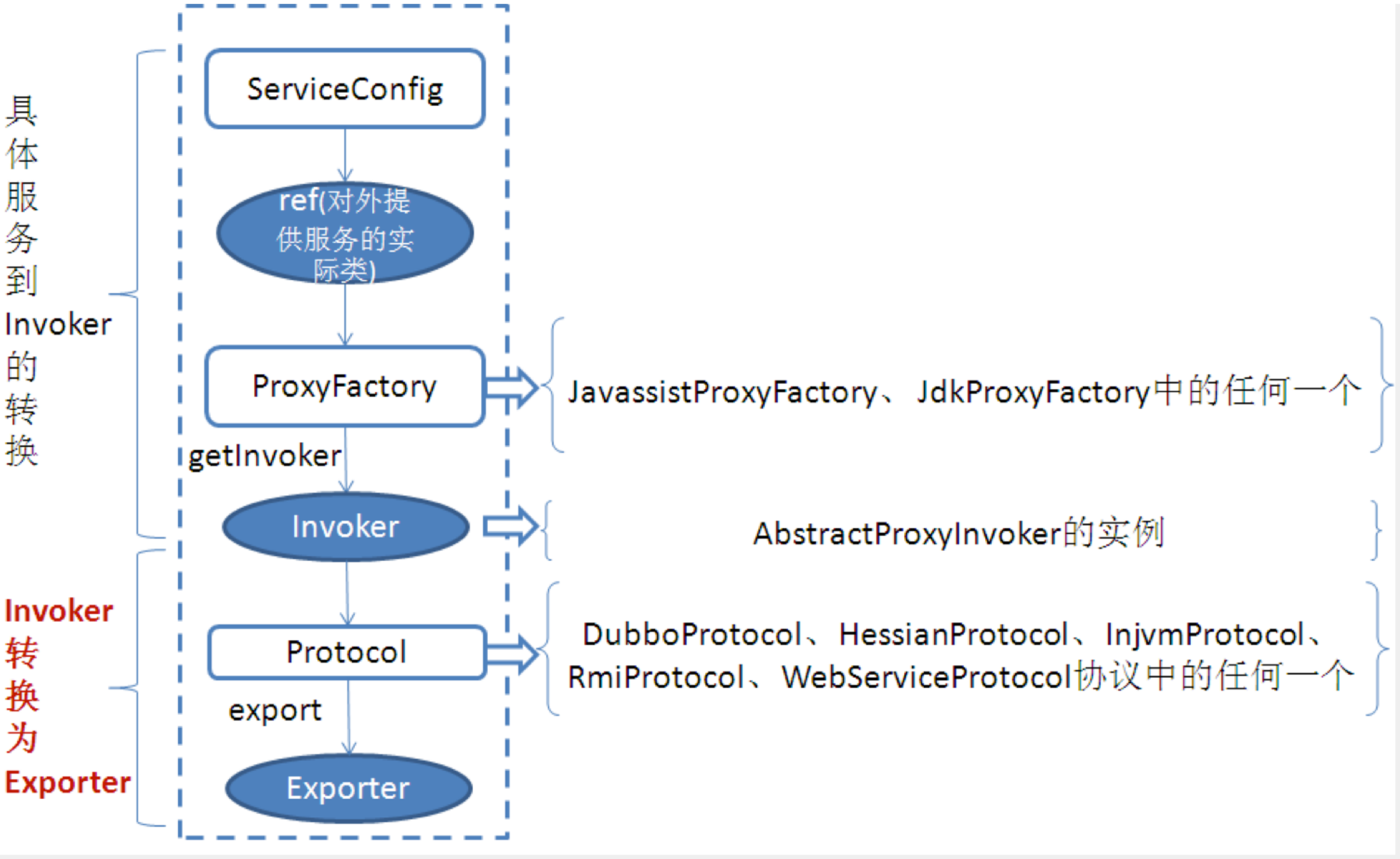
- DUBBO是一个分布式服务框架，致力于提供高性能和透明化的RPC远程服务调用方案。

- 是阿里巴巴SOA服务化治理方案的核心框架，每天为2,000+个服务提供3,000,000,000+次访问量支持，并被广泛应用于阿里巴巴集团的各成员站点。［http://www.dubbo.io］

Dubbo模型

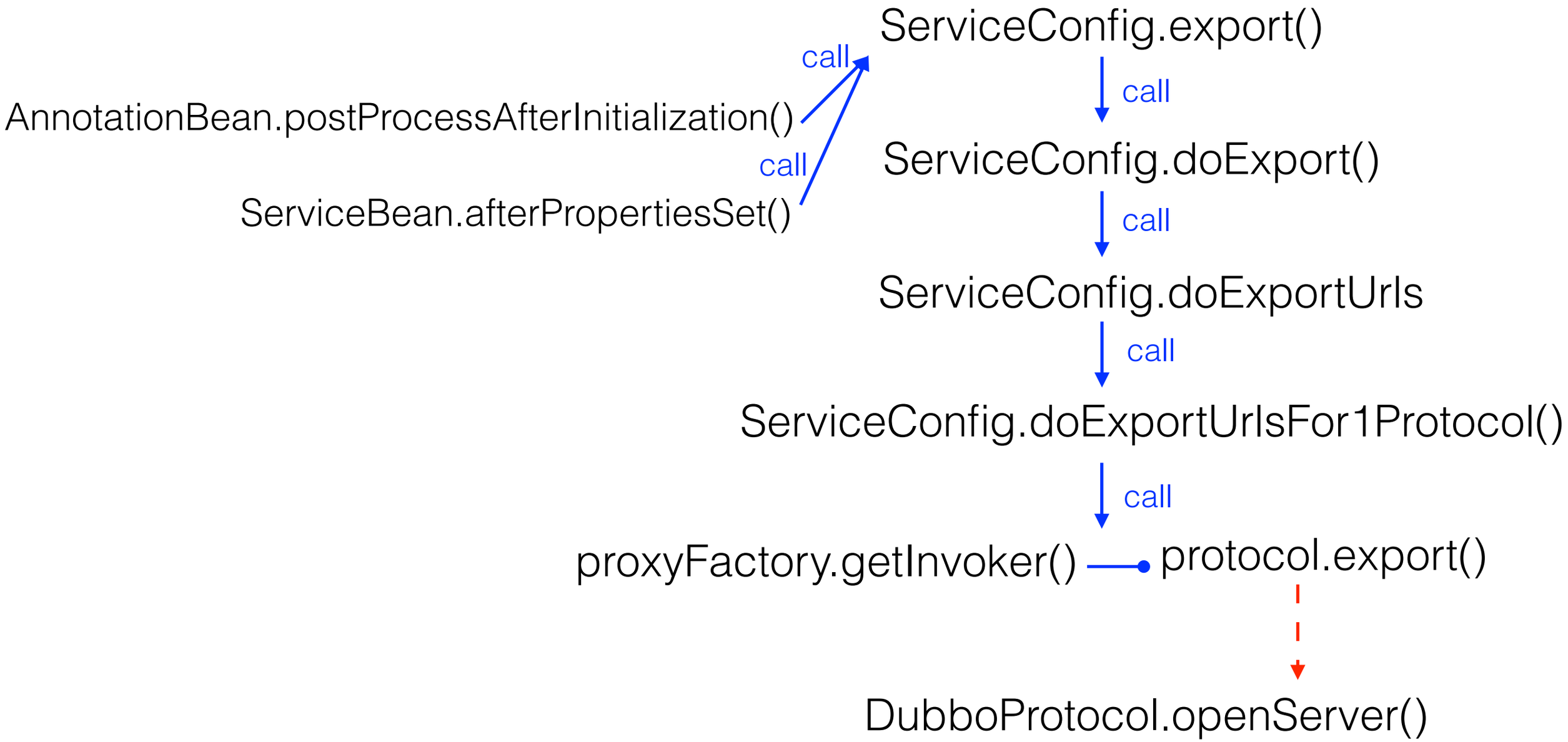

caller
call
Remote API
import
call
RPCClient
RPCProxy
call
RPCInvoker
call
RPCProtocol
encode
RPCConnector
RPCChannel
RPCAcceptor
decode
RPCProtocol
call
RPCProcessor
call
Remote API
call
RPCInvoker
callee
export
RPCServer

服务暴露

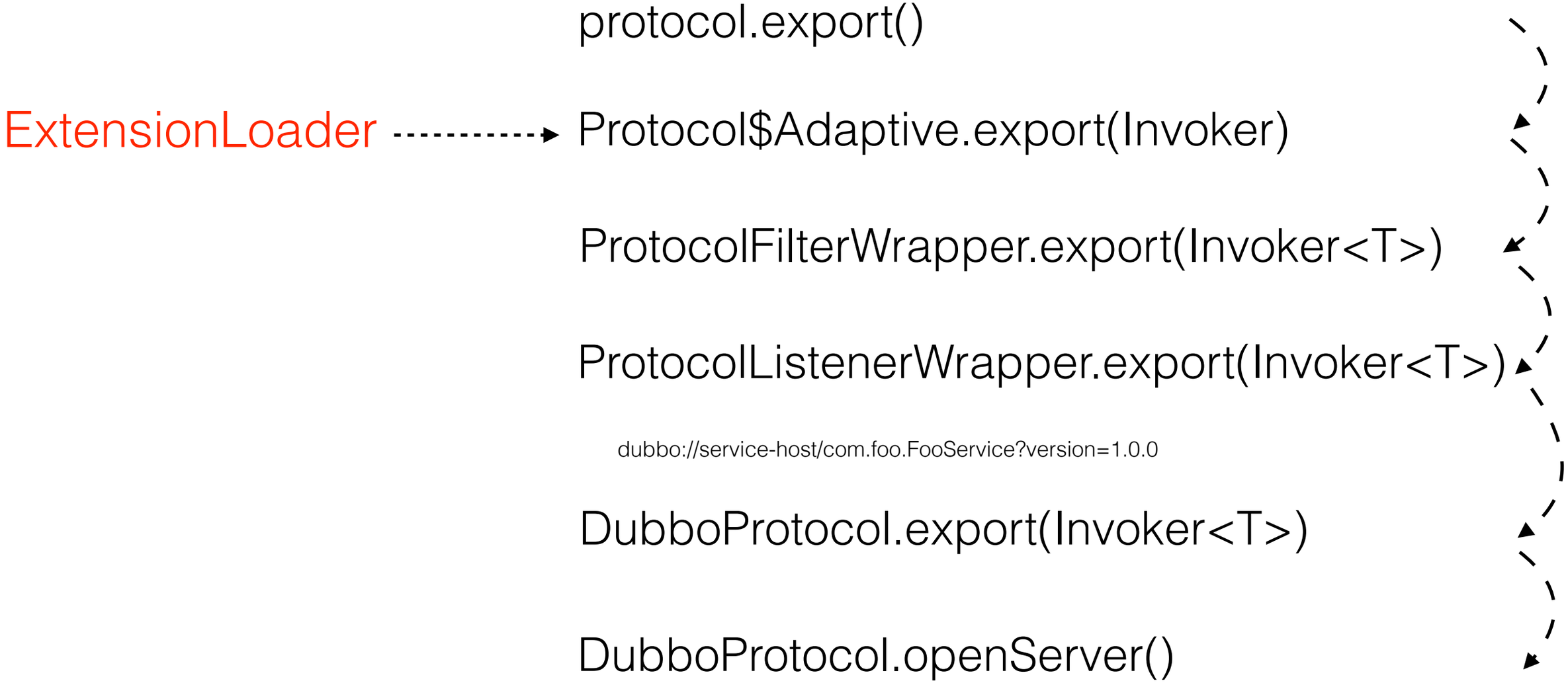


引自dubbo官方文档

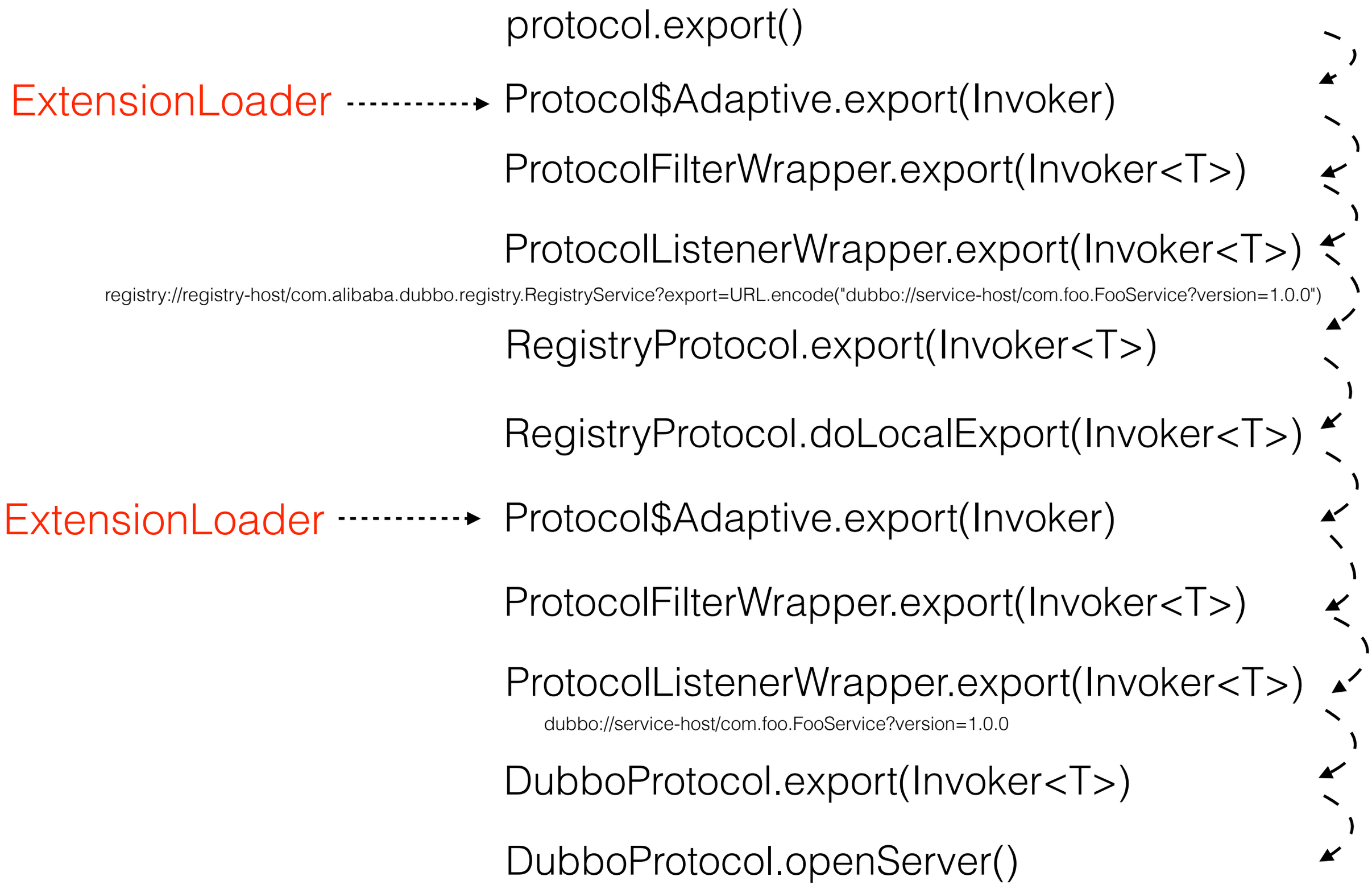
Dubbo服务暴露


ServiceConfig.export()
call
AnnotationBean.postProcessAfterInitialization()
call
ServiceBean.afterPropertiesSet()
call
ServiceConfig.doExport()
call
ServiceConfig.doExportUrls
call
ServiceConfig.doExportUrlsFor1Protocol()
call
proxyFactory.getInvoker()
protocol.export()
DubboProtocol.openServer()

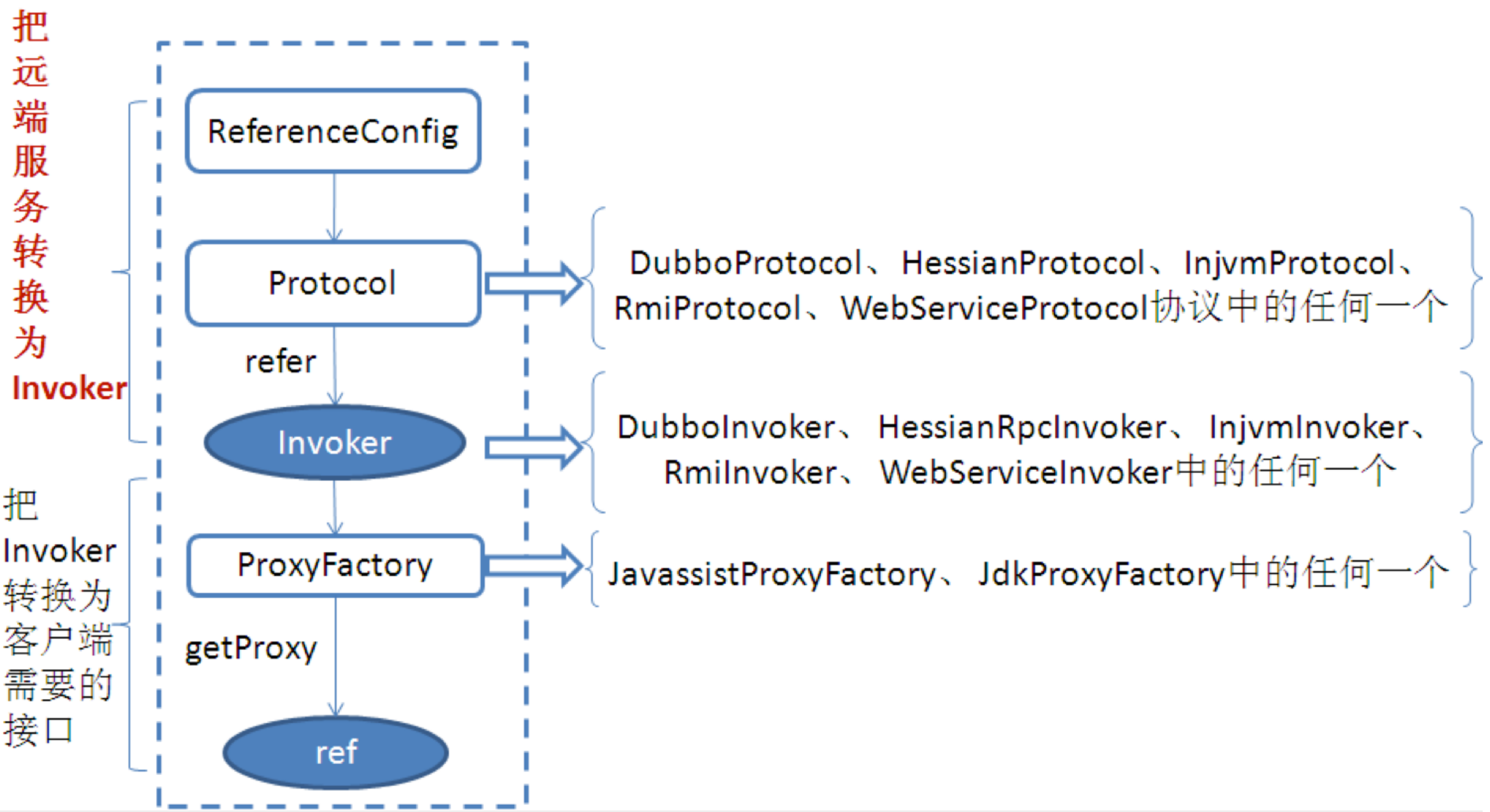
Dubbo服务暴露-无注册中心



Dubbo服务暴露-单个注册中心



Dubbo服务引用



引自dubbo官方文档

Dubbo服务引用

AnnotationBean.postProcessBeforeInitialization()

ReferenceConfig.afterPropertiesSet()

无注册中心／单个注册中心 protocol.refer()

多个注册中心 cluster.join()

AnnotationBean.refer()

ReferenceBean.getObject()

ReferenceConfig.get()

ReferenceConfig.Init()

ReferenceConfig.CreateProxy()

ProxyFactory.getProxy()

ProxyFactory$Adaptive.getProxy()

StubProxyFactoryWrapper.getProxy()

JavassistProxyFactory.getProxy()

Dubbo服务引用 – 无注册中心

Dubbo服务引用 – 单个注册中心

Protocol.refer()

↓ call

Protocol$Adaptive.refer()

↓ call

ProtocolFilterWrapper.refer()

↓ call

ProtocolListenerWrapper.refer()

registry://registry-host/com.alibaba.dubbo.registry.RegistryService?refer=URL.encode("consumer://consumer-host/com.foo.FooService?version=1.0.0")

call

RegistryProtocol.refer()

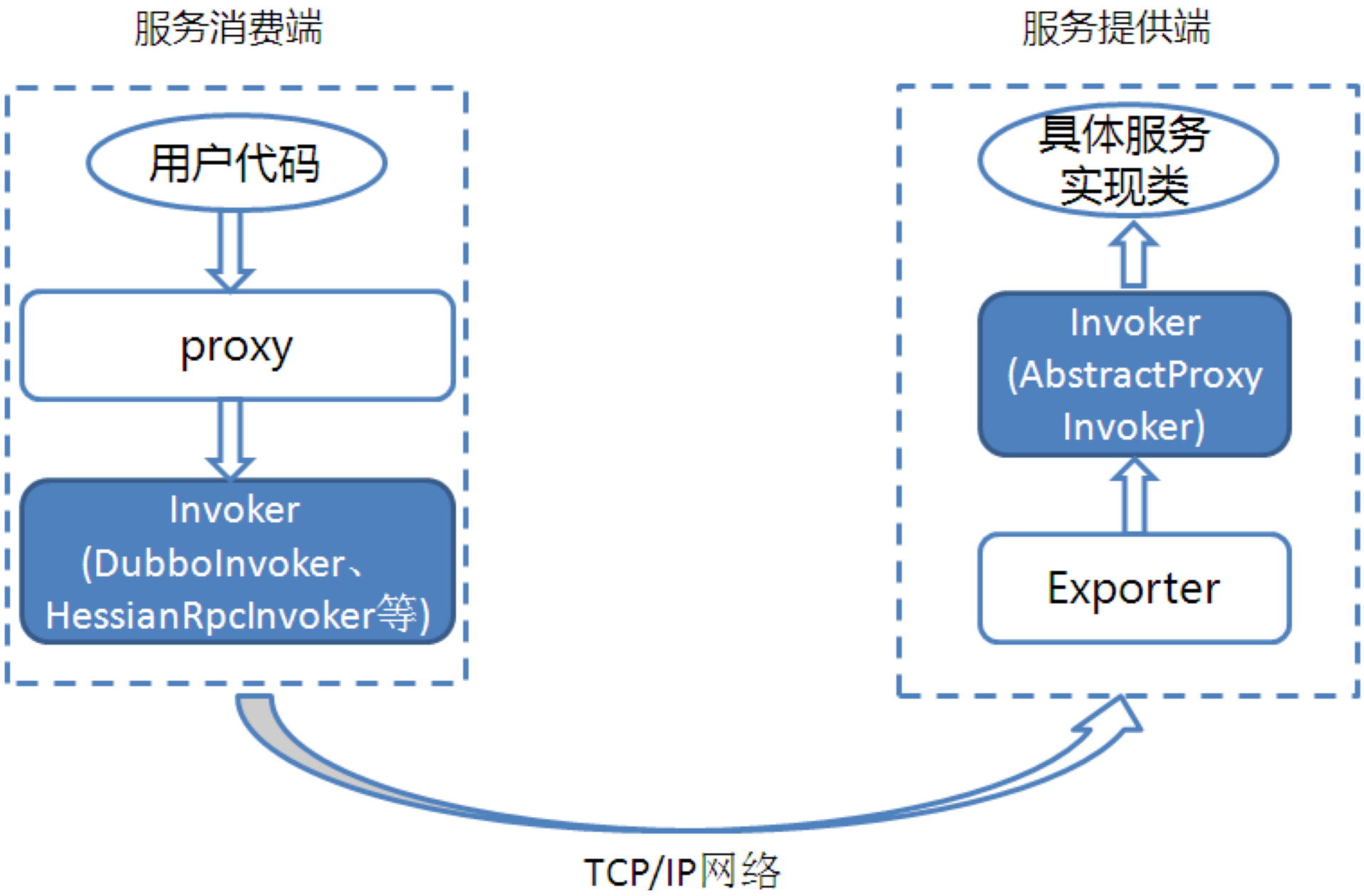
RegistryProtocol.doRefer()　　注册服务自身

Cluster$Adaptive.join()

MockClusterWrapper.join()

FailOverCluster.join()　　伪装成单个提供者引用Invoker

Dubbo服务引用 – 多个注册中心



引自dubbo官方文档

Dubbo服务调用 – 无注册中心

DubboInvoker.invoke()

```
if (isOneway) {
    /**
     * add by woodle
     * 执行的方法不需要返回值: 直接使用ExchangeClient的send方法
     */
    boolean isSent = getUrl().getMethodParameter(methodName, Constants.SENT_KEY, false);
    currentClient.send(inv, isSent);
    RpcContext.getContext().setFuture(null);
    return new RpcResult();
} else if (isAsync) {
    /**
     * add by woodle
     * 执行的方法的结果需要异步返回: 使用ExchangeClient的request方法，返回一个ResponseFuture,
     * 通过ThreadLocal方式与当前线程绑定，未等服务器端响应结果就直接返回
     */
    ResponseFuture future = currentClient.request(inv, timeout) ;
    RpcContext.getContext().setFuture(new FutureAdapter<Object>(future));
    return new RpcResult();
} else {
    /**
     * add by woodle
     * - 执行的方法的结果需要同步返回: 使用ExchangeClient的request方法，返回一个ResponseFuture,
     * 一直阻塞到服务器端返回响应结果
     */
    RpcContext.getContext().setFuture(null);
    return (Result) currentClient.request(inv, timeout).get();
}
```

Dubbo服务调用 – 有注册中心

invoker.invoke()

InvokerInvocationHandler

MockClusterInvoker(如果配置了Mock，则直接调用本地Mock类)

FailoverClusterInvoker(负载均衡，容错机制，默认在发生错误的情况下，进行两次重试)

RegistryDirectory$InvokerDelegete

ConsumerContextFilter

FutureFilter

DubboInvoker

不仅仅是一个RPC框架，也能承担SOA任务

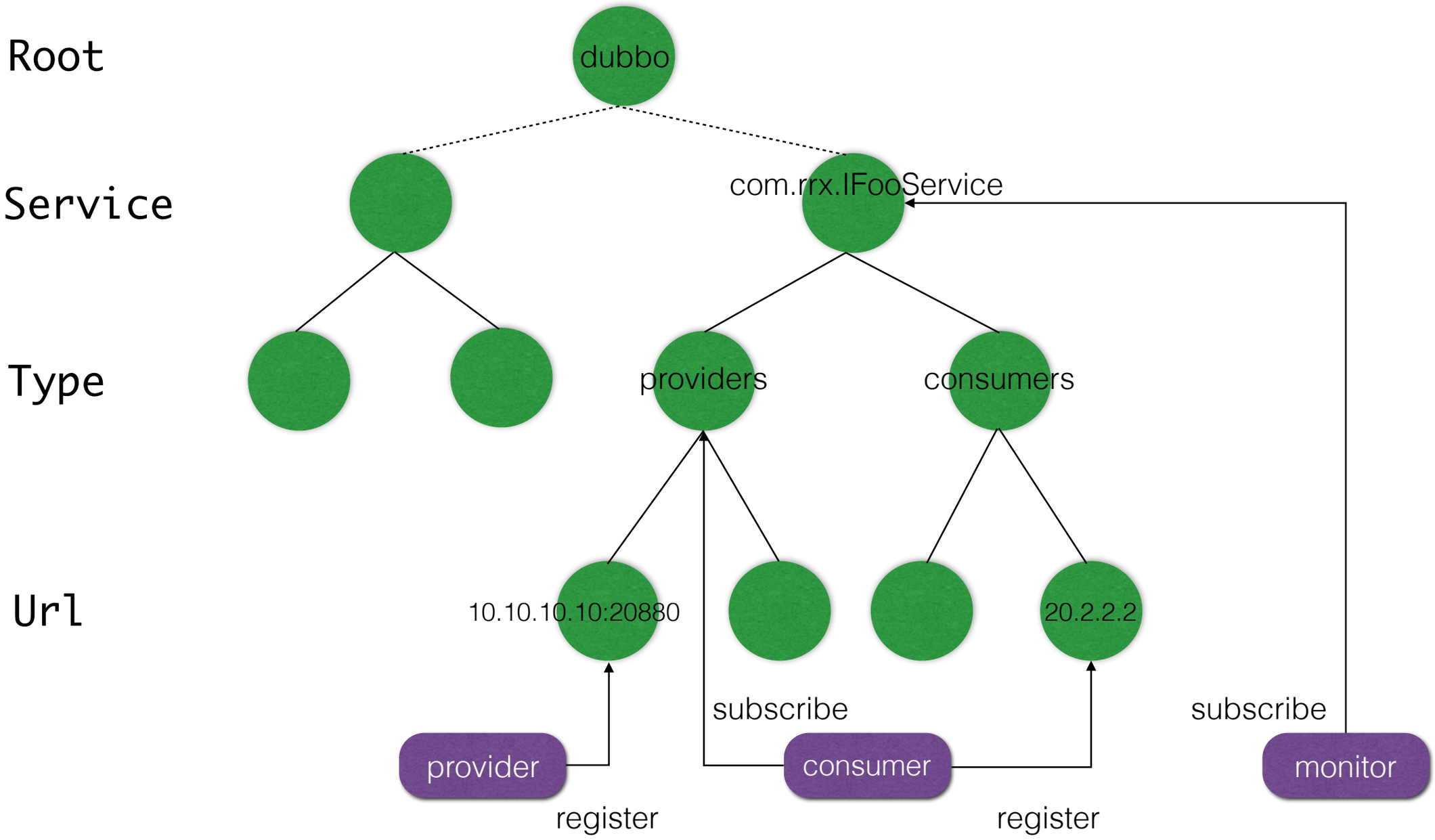
- Rigistry：服务的注册与发现
- Cluster：集群
- LoadBalance：负载均衡
- Router：路由配置
- 其它feature

注册中心

- Multicast
- Zookeeper
- Redis
- Simple

注册中心

Zookeeper Registry Architecture



注册服务


RegistryProtocol.export()
RegistryProtocol.getRegistry()
RegistryFactory$Adaptive.getRegistry()
AbstractRegistryFactory.getRegistry()
ZookeeperRegistryFactory.createRegistry()
ZookeeperRegistry
FailbackRegistry.register()
ZookeeperRegistry.doRegister()

订阅服务

FailbackRegistry.subscribe()

ZookeeperRegistry.dodoSubscribe()

ExtensionLoader

扩展点

CacheFactory
Compiler
ExtensionFactory
LoggerAdapter
Serialization
StatusChecker
DataStore
ThreadPool
Container
PageHandler
MonitorFactory
RegistryFactory

TelnetHandler
ZookeeperTransporter
ExporterListener
Filter
InvokerListener
ChannelHandler
Codec
Codec2
Dispatcher
Transporter
Exchanger
enerator

Protocol
ProxyFactory
Cluster
ConfiguratorFactory
LoadBalance
Merger
RouterFactory
RuleConverter
ClassNameG
Networker
Validation
HttpBinder

```
package <扩展点接口所在包>;

public class <扩展点接口名>$Adpative implements <扩展点接口> {
    public <有@Adaptive注解的接口方法>(<方法参数>) {
        if(是否有URL类型方法参数?) 使用该URL参数
        else if(是否有方法类型上有URL属性) 使用该URL属性
        # <else 在加载扩展点生成自适应扩展点类时抛异常，即加载扩展点失败！>

        if(获取的URL == null) {
            throw new IllegalArgumentException("url == null");
        }

        根据@Adaptive注解上声明的Key的顺序，从URL获致Value，作为实际扩展点名。
        如URL没有Value，则使用缺省扩展点实现。如没有扩展点， throw new IllegalStateException("Fail to get extension");

        在扩展点实现调用该方法，并返回结果。
    }

    public <有@Adaptive注解的接口方法>(<方法参数>) {
        throw new UnsupportedOperationException("is not adaptive method!");
    }
}
```

Q & A

dubbo协议

Reference