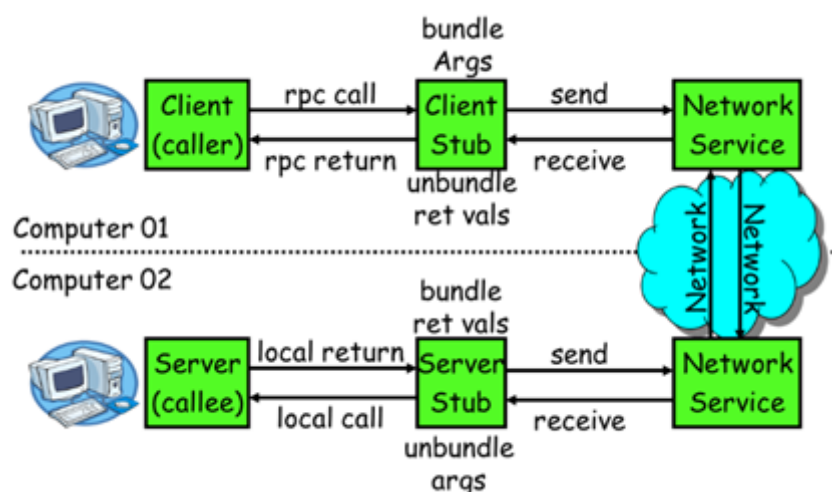


dubbo笔记 (4) dubbo原理

一、RPC&Netty原理

1. RPC原理



一次完整的RPC调用流程（同步调用，异步另说）如下：

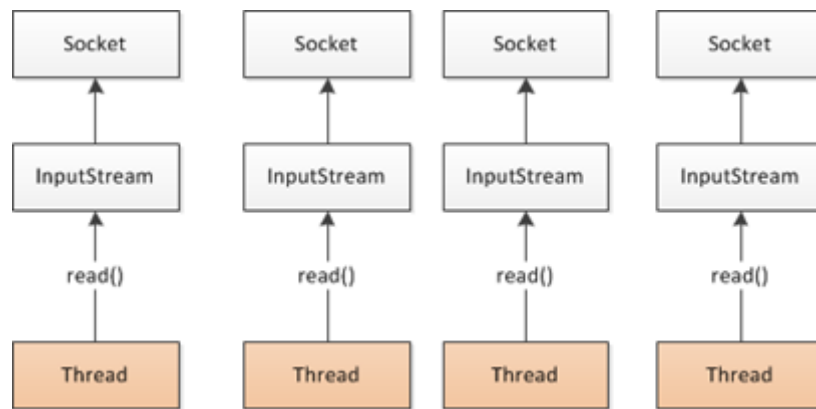
- 1) 服务消费方（client）调用以本地调用方式调用服务；
- 2) client stub接收到调用后负责将方法、参数等组装成能够进行网络传输的消息体；
- 3) client stub找到服务地址，并将消息发送到服务端；
- 4) server stub收到消息后进行解码；
- 5) server stub根据解码结果调用本地的服务；
- 6) 本地服务执行并将结果返回给server stub；
- 7) server stub将返回结果打包成消息并发送至消费方；
- 8) client stub接收到消息，并进行解码；
- 9) 服务消费方得到最终结果。

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，不可见的。

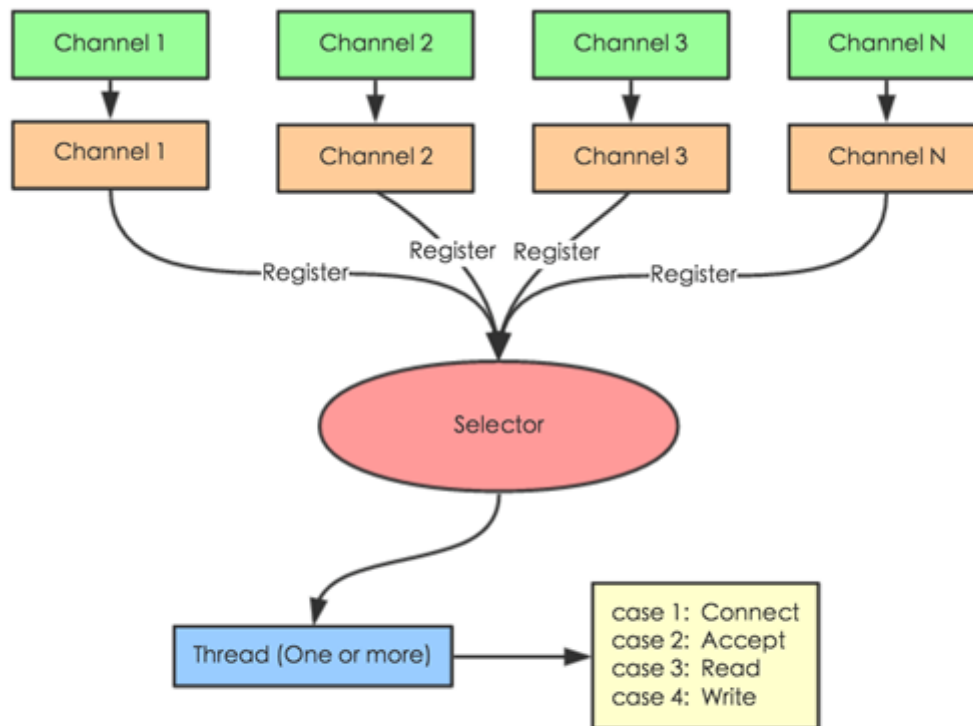
2. netty通信原理

Netty是一个异步事件驱动的网络应用程序框架，用于快速开发可维护的高性能协议服务器和客户端。它极大地简化并简化了TCP和UDP套接字服务器等网络编程。

BIO (Blocking IO)



NIO (Non-Blocking IO)

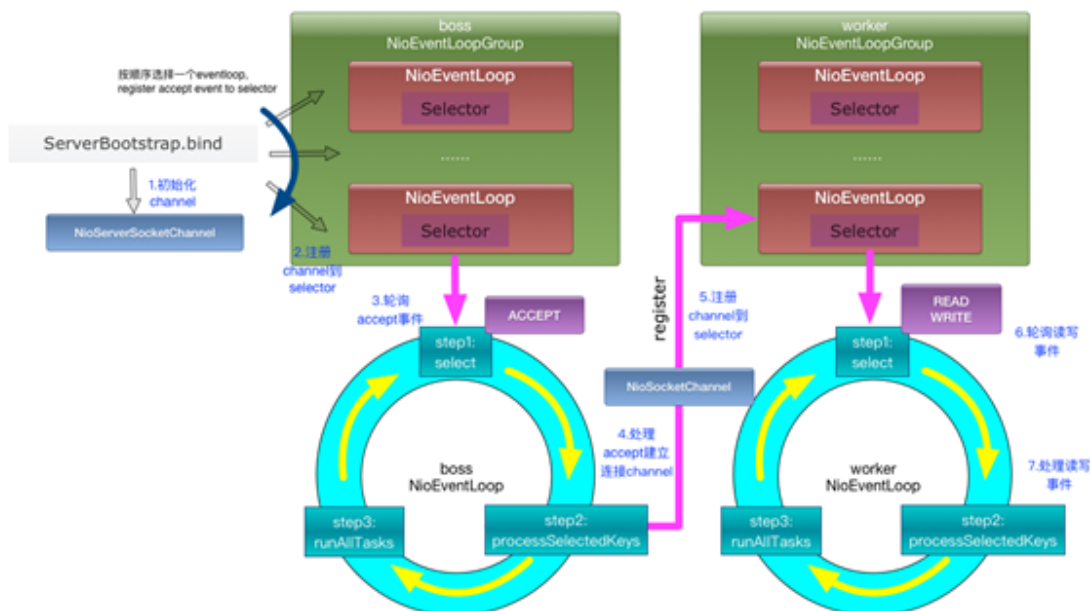


NIO Selector模型

Selector 一般称为**选择器**，也可以翻译为**多路复用器**，

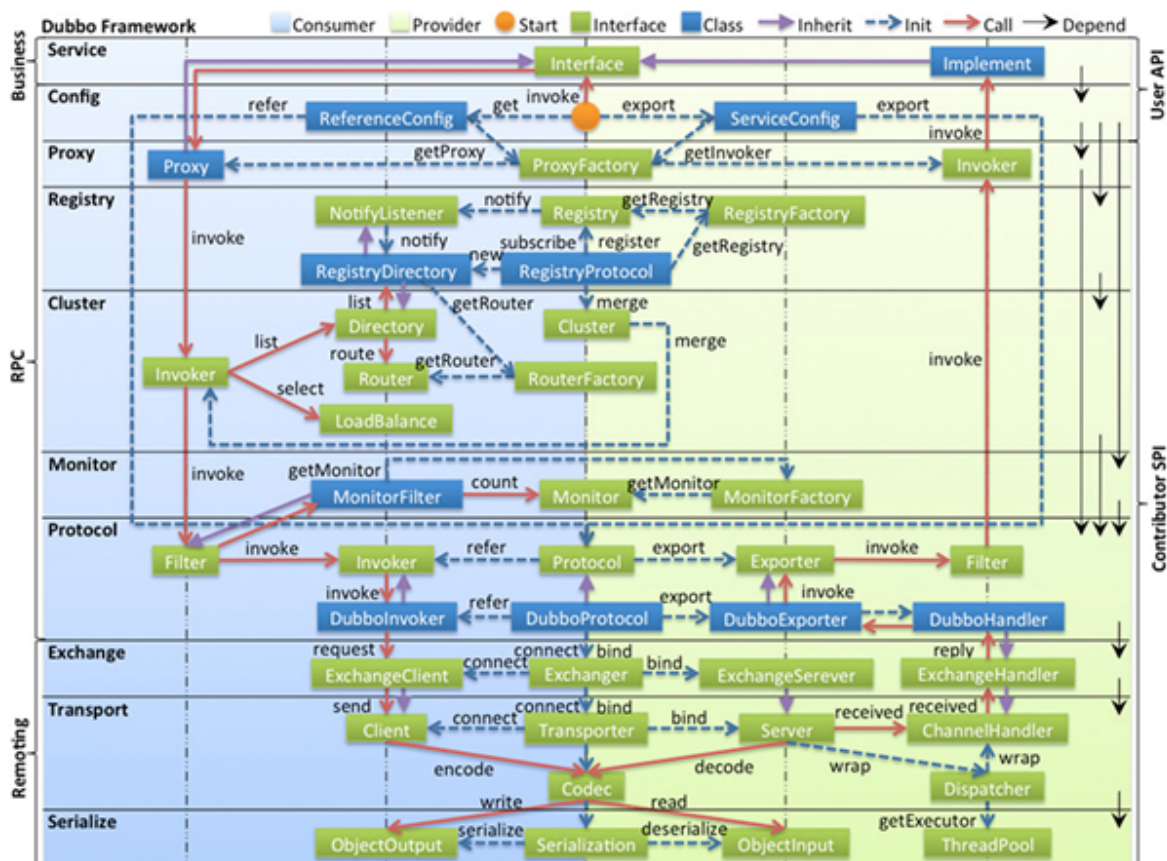
Connect（连接就绪）、Accept（接受就绪）、Read（读就绪）、Write（写就绪）

Netty基本原理：



二、框架设计

1. 整体设计



图例说明：

- 图中左边淡蓝背景的为服务消费方使用的接口，右边淡绿色背景的为服务提供方使用的接口，位于中轴线上的为双方都用到的接口。
- 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，其它各层均为 SPI。
- 图中绿色小块的为扩展接口，蓝色小块为实现类，图中只显示用于关联各层的实现类。

- 图中蓝色虚线为初始化过程，即启动时组装链，红色实线为方法调用过程，即运行时调时链，紫色三角箭头为继承，可以把子类看作父类的同一个节点，线上的文字为调用的方法。

2. 各层说明

- **config 配置层**：对外配置接口，以 ServiceConfig, ReferenceConfig 为中心，可以直接初始化配置类，也可以通过 spring 解析配置生成配置类
- **proxy 服务代理层**：服务接口透明代理，生成服务的客户端 Stub 和服务端 Skeleton, 以 ServiceProxy 为中心，扩展接口为 ProxyFactory
- **registry 注册中心层**：封装服务地址的注册与发现，以服务 URL 为中心，扩展接口为 RegistryFactory, Registry, RegistryService
- **cluster 路由层**：封装多个提供者的路由及负载均衡，并桥接注册中心，以 Invoker 为中心，扩展接口为 Cluster, Directory, Router, LoadBalance
- **monitor 监控层**：RPC 调用次数和调用时间监控，以 Statistics 为中心，扩展接口为 MonitorFactory, Monitor, MonitorService
- **protocol 远程调用层**：封装 RPC 调用，以 Invocation, Result 为中心，扩展接口为 Protocol, Invoker, Exporter
- **exchange 信息交换层**：封装请求响应模式，同步转异步，以 Request, Response 为中心，扩展接口为 Exchanger, ExchangeChannel, ExchangeClient, ExchangeServer
- **transport 网络传输层**：抽象 mina 和 netty 为统一接口，以 Message 为中心，扩展接口为 Channel, Transporter, Client, Server, Codec
- **serialize 数据序列化层**：可复用的一些工具，扩展接口为 Serialization, ObjectInput, ObjectOutput, ThreadPool

3. 模块分包

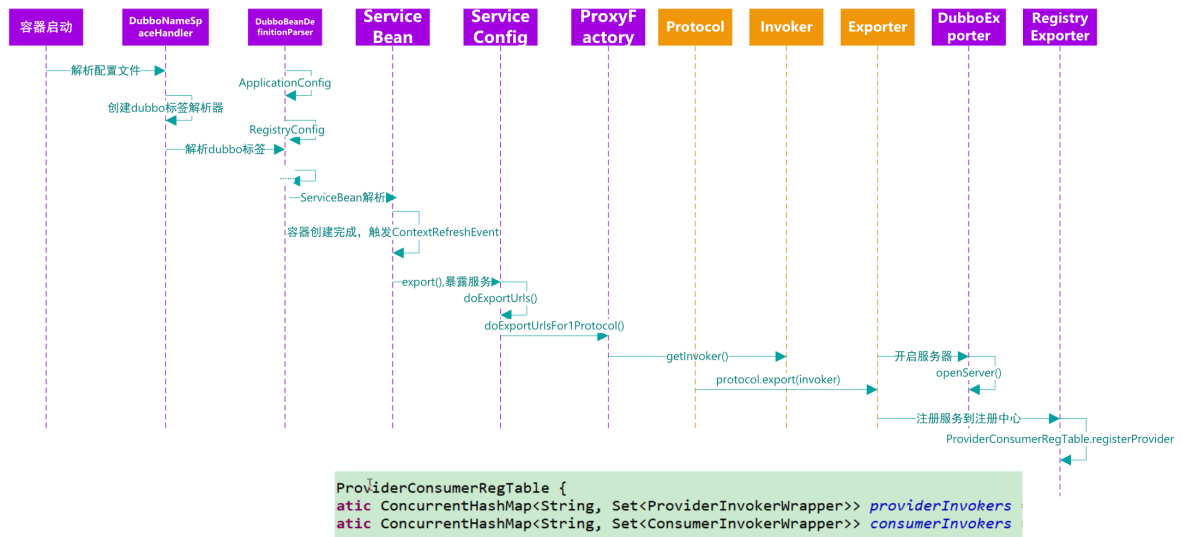
模块说明：

- **dubbo-common 公共逻辑模块**：包括 Util 类和通用模型。
- **dubbo-remoting 远程通讯模块**：相当于 Dubbo 协议的实现，如果 RPC 用 RMI 协议则不需要使用此包。
- **dubbo-rpc 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。
- **dubbo-cluster 集群模块**：将多个服务提供方伪装为一个提供方，包括：负载均衡, 容错, 路由等，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。
- **dubbo-registry 注册中心模块**：基于注册中心下发地址的集群方式，以及对各种注册中心的抽象。
- **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。
- **dubbo-config 配置模块**：是 Dubbo 对外的 API，用户通过 Config 使用 Dubbo，隐藏 Dubbo 所有细节。
- **dubbo-container 容器模块**：是一个 Standalone 的容器，以简单的 Main 加载 Spring 启动，因为服务通常不需要 Tomcat/JBoss 等 Web 容器的特性，没必要用 Web 容器去加载服务。

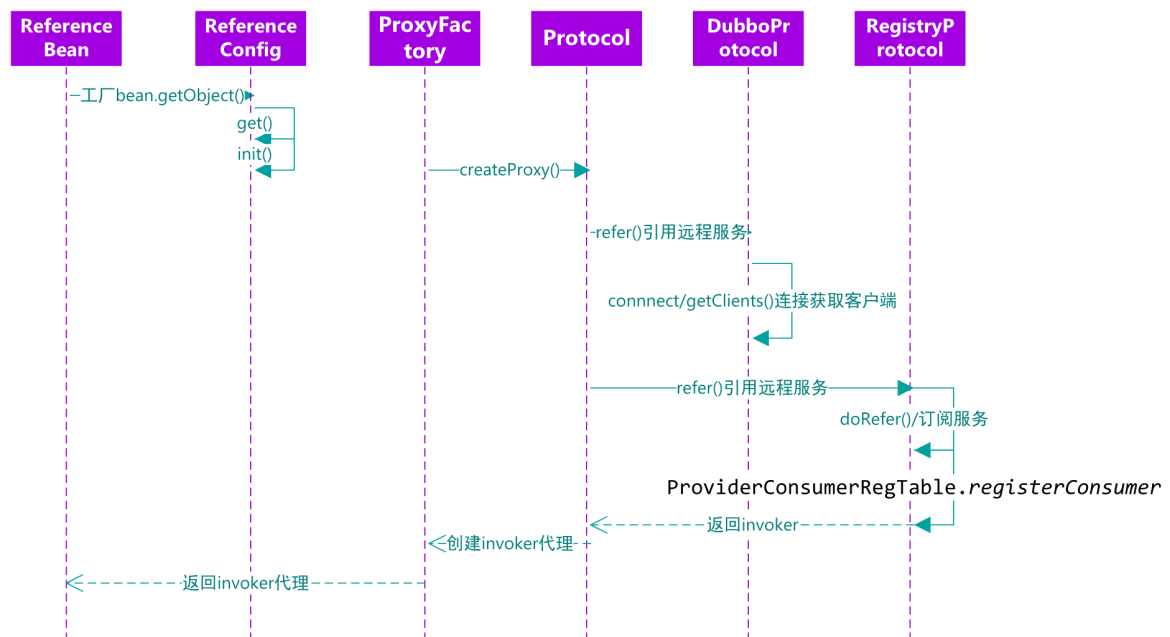
整体上按照分层结构进行分包，与分层的不同点在于：

- container 为服务容器，用于部署运行服务，没有在层中画出。
- protocol 层和 proxy 层都放在 rpc 模块中，这两层是 rpc 的核心，在不需要集群也就是只有一个提供者时，可以只使用这两层完成 rpc 调用。
- transport 层和 exchange 层都放在 remoting 模块中，为 rpc 调用的通讯基础。
- serialize 层放在 common 模块中，以便更大程度复用。

三、服务暴露流程



四、服务引用流程



五、服务调用流程

