

极客时间 Java 进阶训练营

第 18 课

分布式服务-Dubbo 技术详解



KimmKing

Apache Dubbo/ShardingSphere PMC



Apache Dubbo/ShardingSphere PMC

前某集团高级技术总监/阿里架构师/某银行北京研发中心负责人

阿里云 MVP、腾讯 TVP、TGO 会员

10 多年研发管理和架构经验

熟悉海量并发低延迟交易系统的设计实现

目录

1. Dubbo 框架介绍*
2. Dubbo 技术原理*
3. Dubbo 应用场景
4. Dubbo 最佳实践*
5. 如何看 Dubbo 源码
6. 总结回顾与作业实践

1. Dubbo 框架介绍*

Dubbo 的发展历史



开源期（2011-2013，横空出世）：Dubbo 是阿里巴巴 B2B 开发的，2011年开源。

沉寂期（2013-2017，潜龙在渊）：2013年到2017年，Dubbo 的维护程度很低。

复兴期（2017-2019，朝花夕拾）：2017年8月份重启维护，2018年2月加入 Apache 孵化器，2019年5月顺利毕业。

Dubbo 的发展历史

Dubbo 产生于阿里巴巴 B2B 的实际业务需要。

淘系的 HSF，随着 B2B 退市，团队分流，导致 Dubbo 停滞。

2013年以后，国内 Dubbo 存量用户一直最多，增量 Spring Cloud 后来居上。

类似 IE 浏览器与 Chrome 浏览器。

当当、京东等公司的服务化都是基于 Dubbo 实现的，四大行都有使用。

Dubbo 的主要功能

Apache Dubbo 是一款高性能、轻量级的开源 Java 服务框架

六大核心能力：

面向接口代理的**高性能 RPC 调用**，**智能负载均衡**，**服务自动注册和发现**，**高度可扩展能力**，**运行期流量调度**，可视化的**服务治理与运维**。



面向接口代理的高性能RPC调用

提供高性能的基于代理的远程调用能力，服务以接口为粒度，为开发者屏蔽远程调用底层细节。



智能负载均衡

内置多种负载均衡策略，智能感知下游节点健康状况，显著减少调用延迟，提高系统吞吐量。



服务自动注册与发现

支持多种注册中心服务，服务实例上下线实时感知。



高度可扩展能力

遵循微内核+插件的设计原则，所有核心能力如 Protocol、Transport、Serialization 被设计为扩展点，平等对待内置实现和第三方实现。



运行期流量调度

内置条件、脚本等路由策略，通过配置不同的路由规则，轻松实现灰度发布，同机房优先等功能。



可视化的服务治理与运维

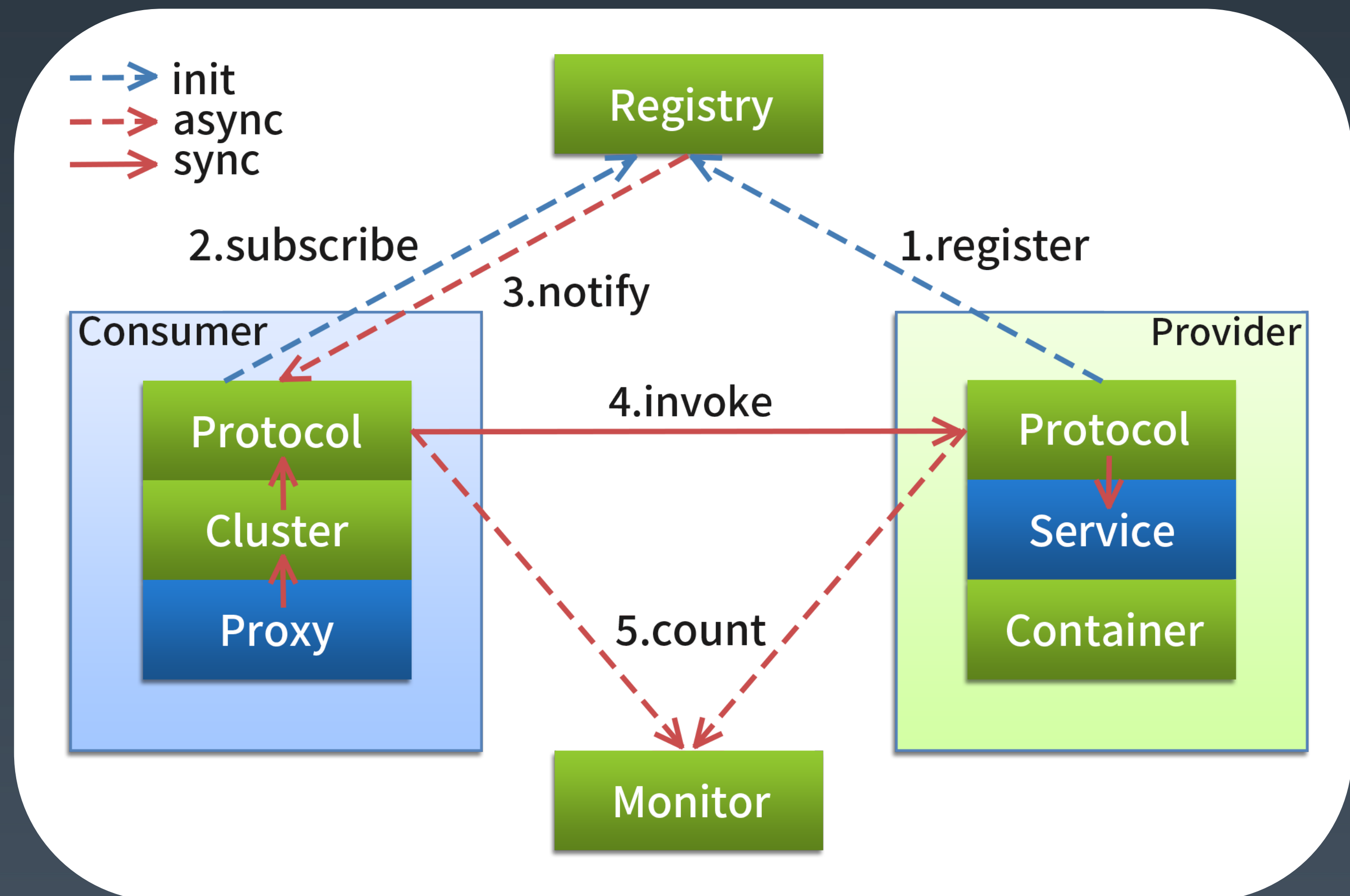
提供丰富服务治理、运维工具：随时查询服务元数据、服务健康状况及调用统计，实时下发路由策略、调整配置参数。

Dubbo 的主要功能

基础功能：RPC 调用

- 多协议（序列化、传输、RPC）
- 服务注册发现
- 配置、元数据管理

框架分层设计，可任意组装和扩展。



Dubbo 的主要功能

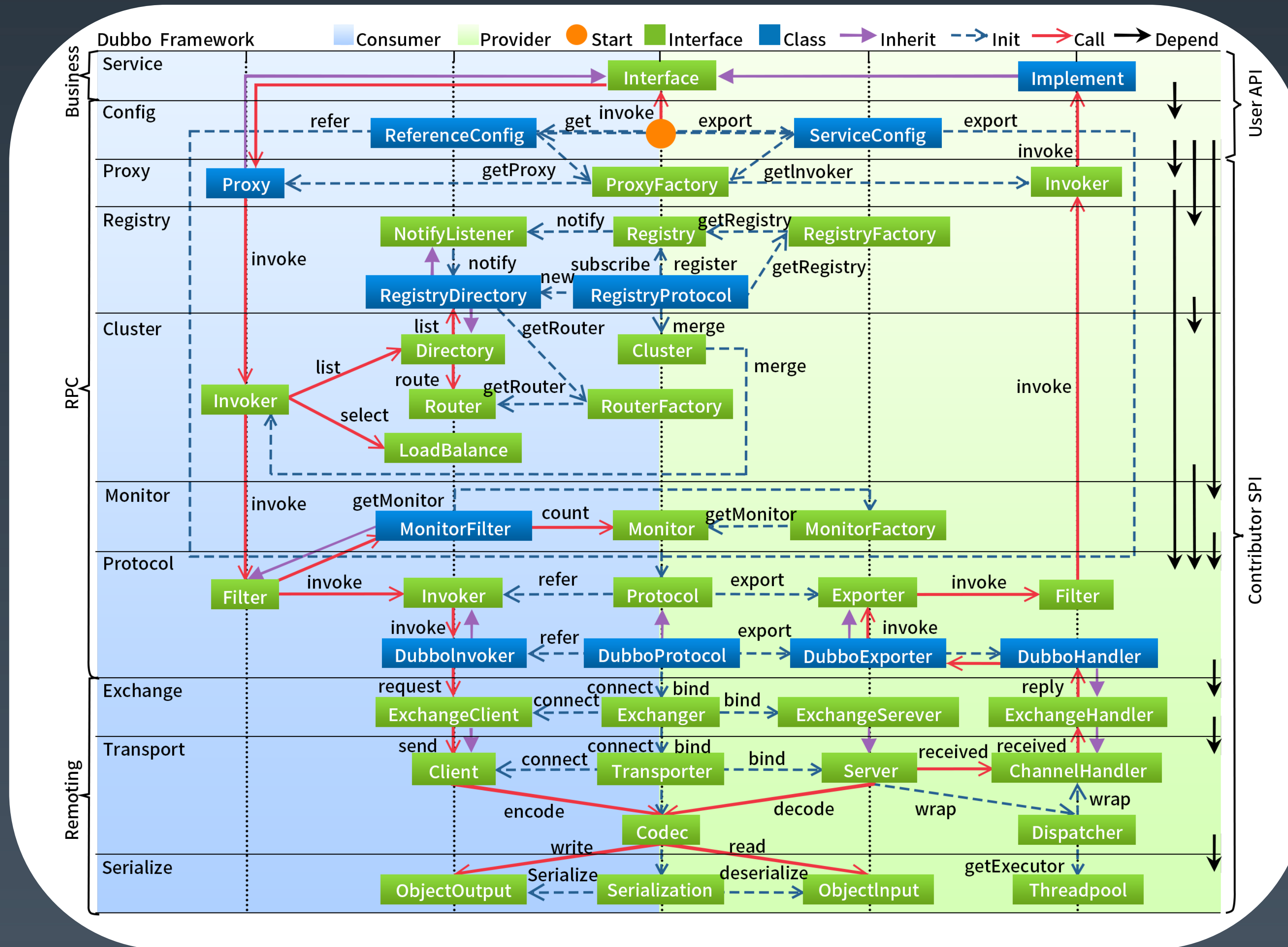
扩展功能：集群、高可用、管控

- 集群，负载均衡
- 治理，路由，
- 控制台，管理与监控

灵活扩展+简单易用，是 Dubbo 成功的秘诀。

2. Dubbo 技术原理*

整体架构



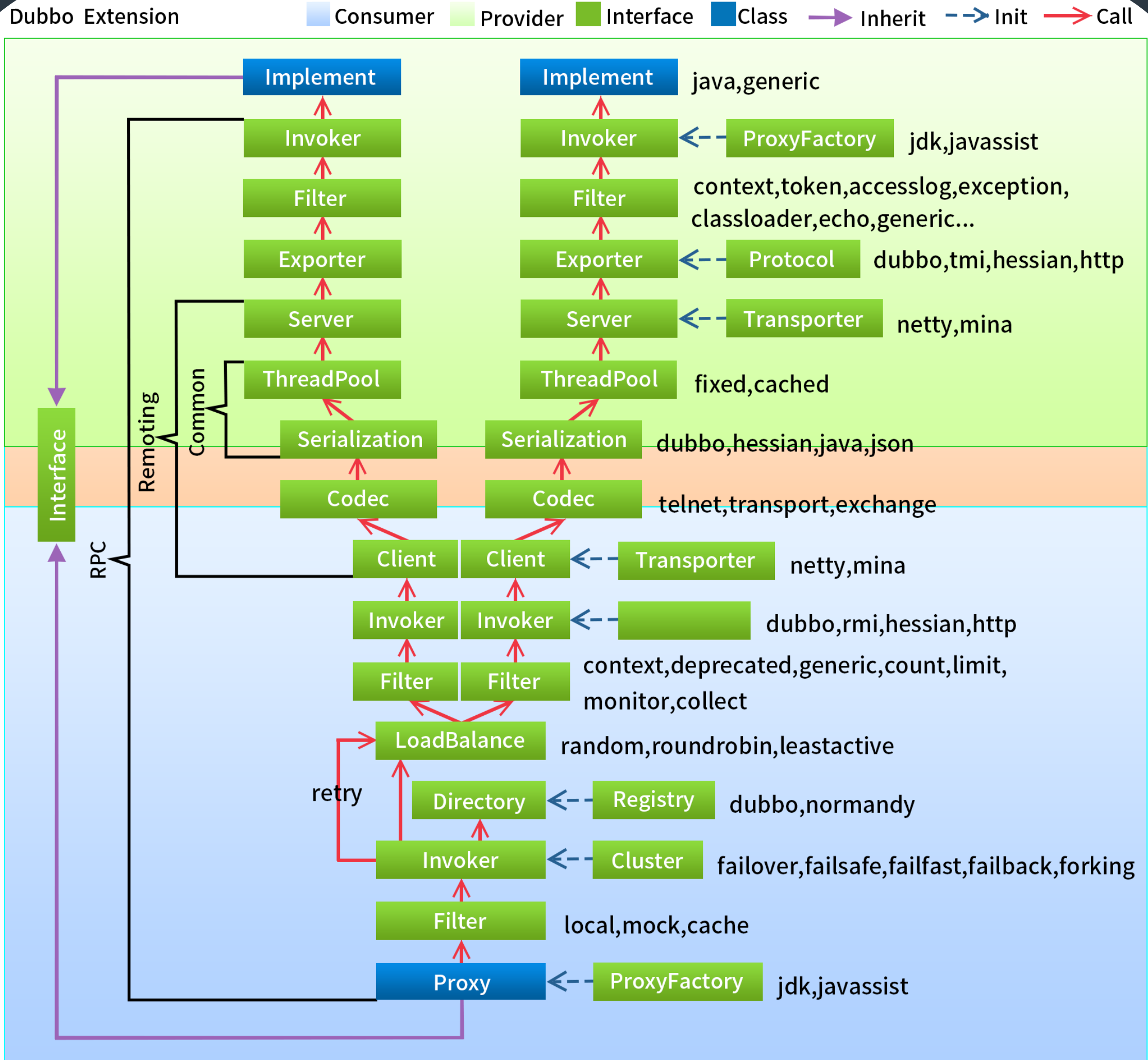
整体架构

1. **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
2. **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
3. **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
4. **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
5. **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

整体架构

6. **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
7. **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
8. **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
9. **serialize** 数据序列化层：可复用的一些工具，扩展接口为 Serialization, ObjectInput, ObjectOutput, ThreadPool

框架设计



SPI 的应用

SPI 与 API

ServiceLoader 机制

META-INF/接口全限定名，文件内容为实现类（ShardingSphere 使用）

其他两个类似的机制：Callback 与 EventBus

Dubbo 的 SPI 扩展，最关键的 SPI：Protocol

xxx=com.alibaba.xxx.XxxProtocol

启动时装配，并缓存到 ExtensionLoader 中。

服务如何暴露

以 InjvmProtocol 为例,

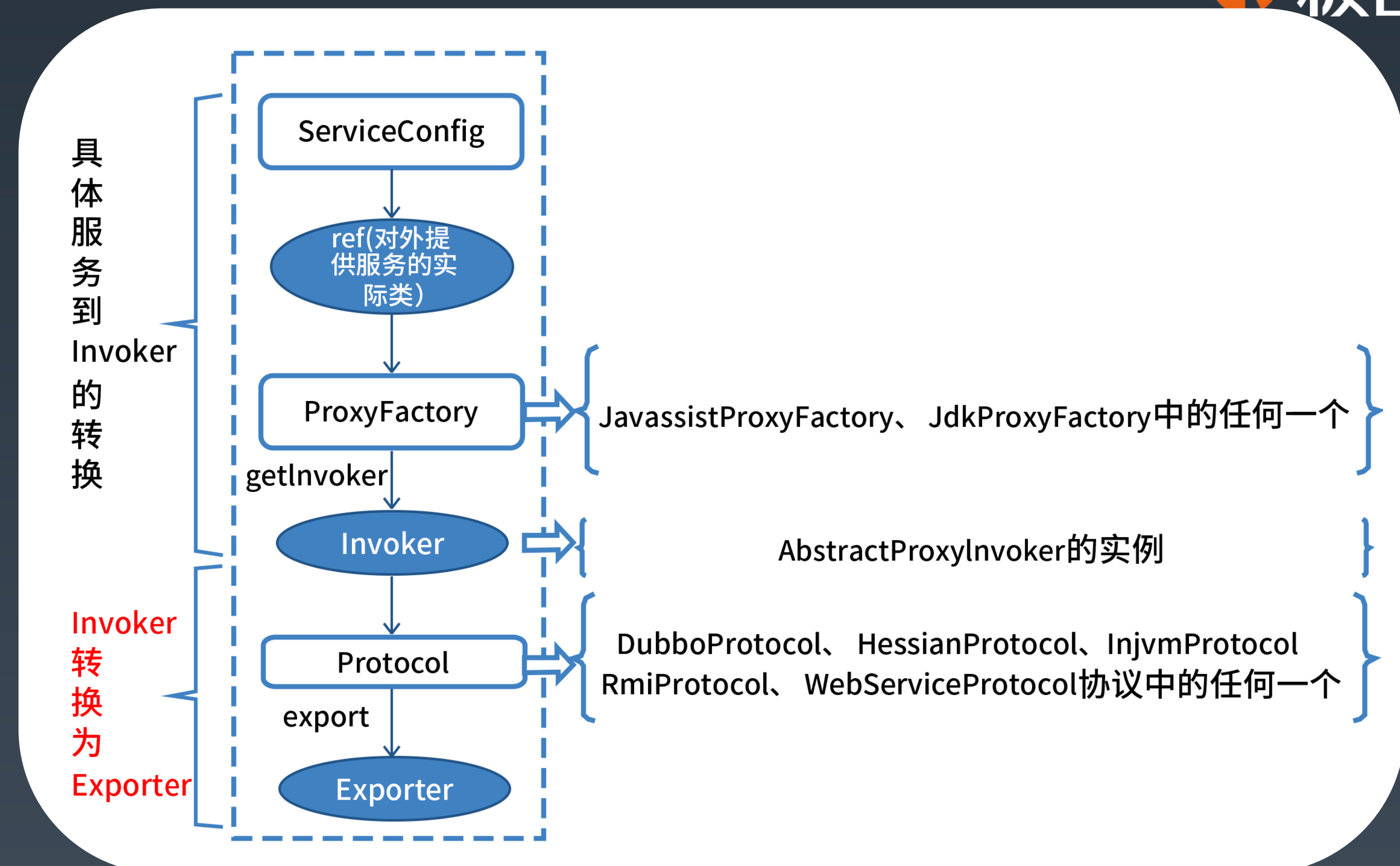
InjvmProtocol

InjvmExporter

XX Invoker

注意：服务使用 URL 表示：

dubbo://127.0.0.1:12345/io.kimmking.dubbo.demo.api.UserService?anyhost=true&application=dubbo-demo-provider&bind.ip=127.0.0.1&bind.port=12345&deprecated=false&dubbo=2.0.2&dynamic=true&generic=false&interface=io.kimmking.dubbo.demo.api.UserService&metadata-type=remote&methods=findById&pid=15467&qos.enable=false&release=2.7.7&revision=1.0.0&side=provider×tamp=1607936477124&version=1.0.0

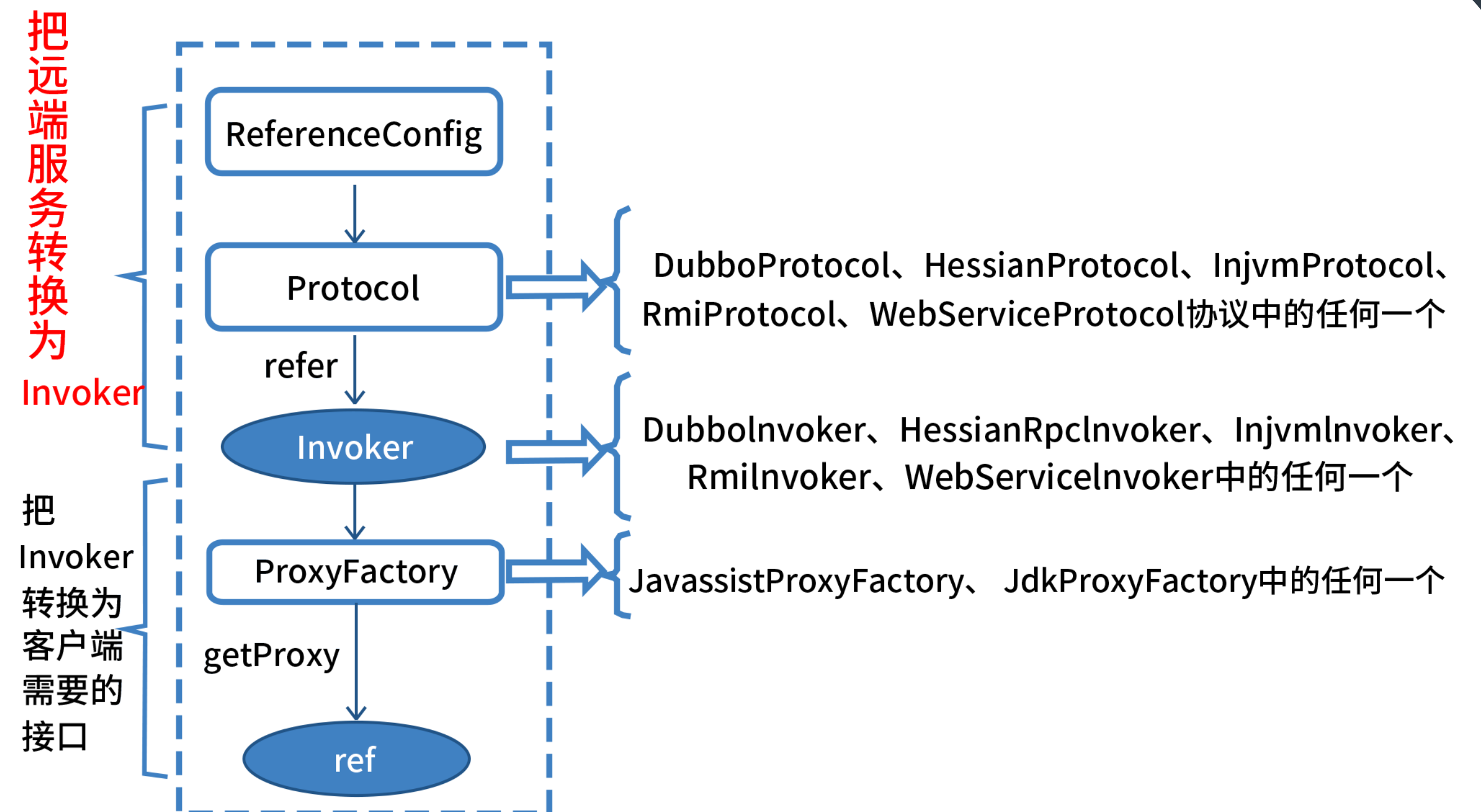


服务如何引用

ServiceReference

ReferenceConfig

createProxy 中创建 Invoker



集群与路由

Cluster

-- **Directory** : return List<Invoker>

-- **Router** : 选取此次调用可以提供服务的 invoker 集合
Condition, Script, Tag

-- **LoadBalance** : 从上述集合选取一个作为最终调用者
Random, RoundRobin, ConsistentHash

泛化引用

GenericService

当我们知道接口、方法和参数，不用存根方式，而是用反射方式调用任何服务。

方法1：

```
<dubbo:reference id="barService" interface="com.foo.BarService" generic="true" />
GenericService barService = (GenericService) applicationContext.getBean("barService");
Object result = barService.$invoke("sayHello", new String[] { "java.lang.String" }, new Object[] { "World" });
```

方法2：

```
ReferenceConfig<GenericService> reference = new ReferenceConfig<GenericService>();
reference.setInterface("com.xxx.XxxService");
reference.setVersion("1.0.0");
reference.setGeneric(true);
GenericService genericService = reference.get();
genericService.$invoke.....
```

隐式传参

Context 模式

```
RpcContext.getContext().setAttachment("index", "1");
```

此参数可以传播到 RPC 调用的整个过程。

大家觉得是如何实现的？

mock

Mock

```
<dubbo.reference id="helloService" interface="io.kimmking.HelloService" mock="true"
timeout="1000" check="false">
```

需要实现一个 io.kimmking.HelloServiceMock 类

可以方便用来做测试。

还记得 mock 功能实现在全局架构的哪一层吗？

3. Dubbo 应用场景

分布式服务化改造

业务系统规模复杂，垂直拆分改造

- 数据相关改造
- 服务设计
- 不同团队的配合
- 开发、测试运维

开放平台

平台发展的两个模式：开放模式、容器模式。

将公司的业务能力开发出来，形成开发平台，对外输出业务或技术能力。

API 与 SPI，分布式服务化与集中式 ESB

直接作为前端使用的后端（BFF）

基于 Dubbo 实现 BFF

作为 BFF（Backend For Frontend）给前端（Web 或 Mobile）提供服务。

一般不太建议这种用法。

灵活性，更好的支持前台业务，向中台发展。

通过服务化建设中台

基于 Dubbo 实现业务中台

将公司的所有业务服务能力，包装成 API，形成所谓的业务中台。

前端业务服务，各个业务线，通过调用中台的业务服务，灵活组织自己的业务。

从而实现服务的服用能力，以及对于业务变化的快速响应。

4. Dubbo 最佳实践

开发分包

建议将服务接口、服务模型、服务异常等均放在 API 包中，因为服务模型和异常也是 API 的一部分，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)。

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。

环境隔离与分组

怎么做多环境的隔离？

- 1、部署多套？
- 2、多注册中心机制
- 3、group 机制
- 4、版本机制

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。

参数配置

通用参数以 consumer 端为准，如果 consumer 端没有设置，使用 provider 数值

建议在 Provider 端配置的 Consumer 端属性有：

timeout：方法调用的超时时间

retries：失败重试次数，缺省是 2

loadbalance：负载均衡算法，缺省是随机 random。

actives：消费者端的最大并发调用限制，即当 Consumer 对一个服务的并发调用到上限后，新调用会阻塞直到超时，可以配置在方法或服务上。

建议在 Provider 端配置的 Provider 端属性有：

threads：服务线程池大小

executes：一个服务提供者并行执行请求上限，即当 Provider 对一个服务的并发调用达到上限后，新调用会阻塞，此时 Consumer 可能会超时。可以配置在方法或服务上。

容器化部署

注册的IP问题，容器内提供者使用的 IP，如果注册到 zk，消费者无法访问。

两个解决办法：

1、docker 使用宿主网络

```
docker xxx -net xxxxx
```

2、docker 参数指定注册的IP和端口，-e

DUBBO_IP_TO_REGISTRY – 注册到注册中心的 IP 地址

DUBBO_PORT_TO_REGISTRY – 注册到注册中心的端口

DUBBO_IP_TO_BIND – 监听 IP 地址

DUBBO_PORT_TO_BIND – 监听端口

运维与监控

Admin 功能较简单，大规模使用需要定制开发，整合自己公司的运维监控系统。

可观测性：tracing、metrics、logging(ELK)

- APM(skywalking, pinpoint, cat, zipkin, ...)

- Prometheus+Grafana

分布式事务

柔性事务, SAGA、TCC、AT

- Seata

- hmily + dubbo

不支持 XA

重试与幂等

服务调用失败默认重试2次，如果接口不是幂等的，会造成业务重复处理。



















如何设计幂等接口？

1、去重-->(bitmap --> 16M), 100w

2、类似乐观锁机制,

5. 如何看 Dubbo 源码

项目结构与重点模块

- >  dubbo-cluster
- >  dubbo-common
- >  dubbo-compatible
- >  dubbo-config
- >  dubbo-configcenter
- >  dubbo-container
- >  dubbo-demo
- >  dubbo-dependencies
- >  dubbo-dependencies-bom
- >  dubbo-distribution
- >  dubbo-filter
- >  dubbo-metadata
- >  dubbo-monitor
- >  dubbo-plugin
- >  dubbo-registry
- >  dubbo-remoting
- >  dubbo-rpc
- >  dubbo-serialization

核心重点模块：

common

config

filter

rpc/remoting/serialization

集群与分布式：

cluster

registry/configcenter/metadata

学习复杂项目源码的技巧

1、不要一上来就关注与每行代码，挑重点代码，重点类，重点方法看。

（跟看书一样）

2、代码是静态的，实际运行时是什么调用关系，怎么处理，数据是什么格式、怎么传递。想要了解这些，最好在跑起来的 demo 里 Dubbo 重点类和方法打断点，debug 的过程跟踪看。

（推荐做笔记）

3、最好组成一个小组，大家相互深入研究，然后相互分享，节省时间，系统学习。

（费曼学习法）

具体的学习方法

实现一个新的RPC，主要是实现入口点 `XXXProtocol`；
同理，了解一个RPC，也需要去 debug `XXXProtocol`。

例如 DubboProtocol：

- 1) 想了解 Dubbo 协议下服务如何暴露，在 DubboProtocol 的 `export` 方法加断点；
- 2) 想了解 Dubbo 协议下服务如何引用，在 DubboProtocol 的 `protocolBindingRefer` 方法加断点；
- 3) 想了解 Dubbo 协议下一个请求进来如何处理，在 DubboProtocol 的 `requestHandler` 的 `reply` 方法加断点。

6.总结回顾与作业实践

第 18 课总结回顾

Dubbo 框架介绍

Dubbo 技术原理

Dubbo 应用场景

Dubbo 最佳实践

第 18 课作业实践

- 1、（选做）按课程第二部分练习各个技术点的应用。
- 2、（选做）按 dubbo-samples 项目的各个 demo 学习具体功能使用。
- 3、（**必做**）结合 dubbo+hmily，实现一个 TCC 外汇交易处理，代码提交到 github：
 - 1) 用户 A 的美元账户和人民币账户都在 A 库，A 使用1美元兑换7人民币；
 - 2) 用户 B 的美元账户和人民币账户都在 B 库，B 使用7人民币兑换1美元；
 - 3) 设计账户表，冻结资产表，实现上述两个本地事务的分布式事务。
- 4、（挑战☆☆）尝试扩展 Dubbo
 - 1) 基于上次作业的自定义序列化，实现 Dubbo 的序列化扩展；
 - 2) 基于上次作业的自定义 RPC，实现 Dubbo 的 RPC 扩展；
 - 3) 在 Dubbo 的 filter 机制上，实现 REST 权限控制，可参考 dubbox；
 - 4) 实现一个自定义 Dubbo 的 Cluster/Loadbalance 扩展，如果一分钟内调用某个服务/提供者超过10次，则拒绝提供服务直到下一分钟；
 - 5) 整合 Dubbo+Sentinel，实现限流功能；
 - 6) 整合 Dubbo 与 Skywalking，实现全链路性能监控。